

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Информационная система управления
платными подписками

Студент

(подпись, дата)

Изгарев Владислав Алексеевич
(ФИО полностью)

Руководитель

(подпись, дата)

Афонин А.Ю.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Изгарев Владислав Алексеевич гр. 21ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Информационная система управления платными подписками

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

2. Руководитель работы к.т.н, доцент кафедры «МО и ПЭВМ» Афонин А.Ю

3. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Назначение: Информационная система управления платными подписками

2) Среда выполнения: Windows 11

Языковые средства разработки: JavaScript, Golang

Инструментальные средства: VS Code, Docker, Visual Paradigm, npm

Технология разработки: клиент-серверная архитектура, REST API, HTML, архитектура чистого кода

3) Основные функции информационной системы:

а) регистрация и аутентификация пользователей;

б) создание, редактирование и удаление личных и групповых подписок;

в) приглашение участников в групповые подписки и выход из них;

г) подтверждение переводов денежных средств от участников;

д) фильтрация подписок по статусу;

е) прогноз трат пользователя на подписки;

ж) генерация PDF-отчёта прогноза трат на подписки;

з) взаимодействие с клиентским приложением через REST API;

и) обеспечение контроля доступа к действиям через механизм авторизации.

4.	Перечень	подлежащих	разработке
вопросов			
1) Анализ предметной области и постановка задачи			
2) Анализ требований и проектирование			
3) Реализация			
4) Контроль качества программного обеспечения			
5) Оформление пояснительной записки			

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 210 листов, 46 рисунков, 23 таблицы, 20 использованных источников, 5 приложений.

ИНФОРМАЦИОННАЯ СИСТЕМА, ПЛАТНЫЕ ПОДПИСКИ, СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, КЛИЕНТСКОЕ ПРИЛОЖЕНИЕ, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, ЧИСТАЯ АРХИТЕКТУРА, JWT-АУТЕНТИФИКАЦИЯ, ГРУППОВАЯ ПОДПИСКА, ЛИЧНАЯ ПОДПИСКА, REST.

Объектом разработки является информационная система для поддержки процессов управления личными и групповыми платными подписками.

Цель работы – разработка программного обеспечения, позволяющего пользователям создавать, редактировать и удалять подписки, управлять участниками групповых подписок, отслеживать статусы и суммы платежей, а также получать аналитическую информацию и генерировать отчёты прогнозов по расходам.

Разработка серверной части приложения велась на языке программирования Golang в среде разработки Microsoft Visual Code. В качестве базы данных было использовано PostgreSQL.

Разработка клиентского приложения производилась на языке программирования JavaScript в среде разработки использовалась Microsoft Visual Code.

Разработка проведена на операционной системе Windows 11. Информационная система предназначена для использования через веб-интерфейс и развёртывается в Docker-контейнерах.

					ПГУ 09.03.04 – 08БР212.9 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата				
Разраб.		Изгарев В.А.			Информационная система управления платными подписками. Пояснительная записка	Лит.	Лист	Листов
Пров.		Афонин А.Ю.					4	210
Реценз.						Группа 21ВП2		
Н. контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	8
1 Анализ предметной области и постановка задачи.....	10
1.1 Анализ предметной области	10
1.1.1 Обзор бизнес-модели «Подписка».....	10
1.1.2 Понятие и классификация платных подписок.....	11
1.1.3 Проблемы и потребности пользователей при управлении подписками.....	13
1.2 Анализ программного обеспечения аналогичного назначения ..	13
1.2.1 Обзор программного обеспечения SubsCrab	13
1.2.2 Обзор программного обеспечения Subsee.....	15
1.2.3 Обзор программного обеспечения Skaffer	15
1.2.4 Сравнительная характеристика программного обеспечения аналогичного назначения.....	16
1.3 Постановка задачи на разработку	18
1.4 Планирование и функционально-стоимостный анализ разработки.....	20
2 Анализ требований к информационной системе	28
2.1 Анализ функциональных требований информационной системы	28
2.2 Требования к серверному программному обеспечению.....	31
2.3 Требования к клиентскому программному обеспечению	32
3 Проектирование информационной системы	34
3.1 Архитектура информационной системы	34
3.2 Проектирование базы данных	35

3.2.1	Концептуальное моделирование базы данных	35
3.2.2	Логическое моделирование базы данных	39
3.3	Проектирование серверного программного обеспечения	43
3.3.1	Назначение и функции серверного приложения	43
3.3.2	Проектирование архитектуры серверного приложения	44
3.3.3	Механизмы безопасности	45
3.4	Проектирование клиентского программного обеспечения	46
3.4.1	Назначение и функции клиентского приложения	46
3.4.2	Проектирование пользовательского интерфейса	46
3.4.3	Проектирование структуры клиентского приложения	48
3.5	Обоснование выбора средств разработки и технологий	49
4	Разработка программного обеспечения	51
4.1	Физическое моделирование базы данных	51
4.2	Разработка серверного приложения	55
4.2.1	Структура серверного приложения	55
4.2.2	Взаимодействие с базой данных	58
4.2.3	Реализация аутентификации и авторизации	61
4.2.4	Реализация отправки уведомлений о списании денежных средств	64
4.2.5	Реализация интерфейса взаимодействия с клиентским приложением	66
4.3	Разработка клиентского приложения	68
4.3.1	Структура клиентского приложения	68
4.3.2	Разработка пользовательского интерфейса	72
4.4	Развертывание информационной системы	87

5	Тестирование.....	90
5.1	Функциональное тестирование	90
5.2	Модульное тестирование	94
5.3	Метрики качества кода.....	98
5.3.1	Общие сведения о метриках качества кода.....	98
5.3.2	Метрики SLOC	98
5.3.3	Метрика Холстеда.....	99
	Заключение	102
	Список источников	104
	Приложение А (обязательное) Код скрипта создания базы данных	106
	Приложение Б (обязательное) Код серверного приложения.....	108
	Приложение В (обязательное) Код клиентского приложения	158
	Приложение Г (рекомендуемое) Результаты функционального тестирования	186
	Приложение Д (рекомендуемое) Код модульных тестов	195

Введение

В современном мире становится больше сервисов и приложений, которые предоставляют свои услуги на основе бизнес-модели платных подписок. Данная модель используется в большинстве сфер – начиная от музыкальных платформ и облачных хранилищ до образовательных ресурсов и программного обеспечения. В связи с увеличением количества предоставляемых услуг на основе данной модели, необходим удобный инструмент, который бы позволял эффективно управлять подписками, отслеживать финансовые затраты, вести их учет и анализ.

Актуальность данной работы обусловлена в необходимости структурирования информации о финансовых расходах пользователя информационной системы, а также потребностью в своевременном уведомлении о предстоящих платежах, автоматическом расчете суммарных расходов, формирования отчетности для более глубокой прозрачности затрат на подписки, а также объединение нескольких пользователей в группы с возможностью распределения стоимости оплаты между участниками и уведомлением о внесении средств участником на счет владельца.

Целью данной выпускной квалификационной работы является разработка информационной системы, которая направлена на пользователей, имеющие платные подписки с потребностью их отслеживания и мониторинга суммарных расходов, генерации отчетов за определенные интервалы времени, формирования групповых подписок.

Для достижения сформированной цели были выделены следующие задачи:

- сделать анализ предметной области;
- провести анализ программного обеспечения аналогичного назначения;
- реализовать проектирование информационной системы;
- разработать программное обеспечение в соответствии с требованиями;

- выполнить тестирование программного обеспечения.

В результате выполнения выпускной квалификационной работы была реализована информационная система с учетом разработанного проектирования и в соответствии с выявленными требованиями к ней.

1 Анализ предметной области и постановка задачи

1.1 Анализ предметной области

1.1.1 Обзор бизнес-модели «Подписка»

Бизнес-модель «Подписка» – это модель, суть которой заключается в том, что потребитель оплачивает фиксированной суммой доступ к продукту или услугам на регулярной основе [1].

В основном, большинство компаний используют ежемесячный тип регулярной оплаты своих услуг, однако также встречаются квартальные и ежегодные платежи.

Данная модель позволяет пользователям получать постоянный доступ к продуктам или услугам без необходимости совершать разовые покупки, что позволяет автоматизировать процесс оплаты.

Также бизнес-модель предоставляет различные типы тарифов подписки, которые способствуют увеличению прибыли путем привлечения большинства пользователей с разными потребностями.

Имеются следующие преимущества бизнес-модели:

- регулярный и предсказуемый доход;
- долгосрочные отношения с клиентами;
- различные типы планов подписки для клиентов.

За счет регулярных платежей предоставляется возможность определить доход от предоставляемых услуг на основании расчета количества клиентов и их размера суммы оплаты, что позволяет более точно планировать бюджет бизнеса. Также предсказуемый доход снижает зависимость от сезонных или экономических колебаний.

Долгосрочные отношения с клиентами позволяют бизнесу глубоко понять их потребности и предпочтения, за счет регулярного взаимодействия и получению обратной связи. Это позволяет компаниям корректировать свои продукты и услуги для удовлетворения изменяющихся требований рынка.

Различные типы планов подписки предоставляют клиентам дополнительные возможности предоставляемого продукта или услуги, тем самым повышая доход и автоматически удовлетворяет увеличившиеся потребности пользователя.

Также существуют следующие недостатки данной модели бизнеса:

- зависимость от удержания клиентов;
- возможный отток клиентов и лояльность;
- поддержание постоянного интереса.

Компания, использующая данную бизнес-модель ориентирована на конечных клиентов, которые подписаны на продукт, и зависима от них. Следственно, если они начнут отказываться от подписки, то доходы будут падать.

Также важным фактором будет являться лояльность клиентов, так как если условия у конкурентной компании по подписке на услугу будут лучше, это может спровоцировать отток клиентов, следовательно, упадок доходности.

Со временем использования подписки, клиенты могут терять интерес к ней. Его необходимо поддерживать путем постоянной поддержки и обновления продукта или услуги, а также проведением эксклюзивных акций и событий.

Данная бизнес-модель пользуется большой популярностью и успехом, показывая её актуальность в текущий момент времени.

1.1.2 Понятие и классификация платных подписок

Исходя из обзора бизнес-модели выше, можно сформировать определение платной подписки.

Платная подписка – это модель доступа пользователя к продукту или услуги компании за счет регулярного внесения фиксированной суммы.

Платные подписки могут классифицироваться по следующим признакам:

- сроку длительности;
- по содержанию и назначению;

- по уровням доступа;
- по способу получения услуги;
- по видам подписки.

Срок длительности может варьироваться от недели до года, но наиболее популярным сроком у компаний, предоставляющих свои услуги, является месяц. Помимо временного срока, длительность может зависеть от использованного ресурса, выделяемого в рамках подписки.

По назначению и виду предоставляемых услуг платные подписки могут разделяться на медиа-подписки, подписки на доступ к программному обеспечению, образовательные, сервисные.

Уровни доступа определяет набор функциональных возможностей для различных потребностей клиента. В основном уровни доступа делятся на базовый, который включает ограниченный функционал, стандартный, содержащий основной пакет с ключевыми функциями и премиум с расширенным функционалом и улучшенной поддержкой.

Есть два способа получения услуги платной подписки:

- онлайн;
- офлайн.

Онлайн способ используют преимущественно компании, которые предоставляют цифровые продукты или услуги в интернете. Офлайн способ выбирают компании, торгующие физическими товарами или продуктами, например, спросом пользуется ежемесячная доставка воды или цветов на дом.

Платные подписки делятся на следующие виды:

- индивидуальные;
- групповые.

Индивидуальные подписки нацелены на одного пользователя, предоставляя ему персональный доступ к услугам. Групповые подписки предназначены для использования услуг кругом лиц в равном доступе или с

определенными ограничениями. Её оформляет владелец, который имеет право приглашать участников.

1.1.3 Проблемы и потребности пользователей при управлении подписками

Одной из проблем является увеличение времени анализа финансовых затрат при большом количестве подписок, привязанных к различным банковским счетам. Пользователю приходится выгружать отчеты из используемых банковских систем и вручную проводить подсчеты с анализом платежей.

Не у всех компании, которые предоставляют групповую подписку, имеется напоминания участникам о предстоящем списании и исходя из этого происходят случаи, когда владелец подписки не получает нужную сумму вовремя и оплата происходит только за счет его личных средств. Отчетливо это видно на примере, когда группа лиц пользуется индивидуальной подпиской как групповой. В этом случае владелец лично напоминает участникам об необходимости внесения платы на его счет за подписку.

Также хочется выделить потребность в гибкой настройке уведомлений, чтобы они приходили пользователям в соответствии с их желанием. Например, для одних клиентов важно, чтобы им напоминали о списании за день, а другим нужны уведомления за три дня или семь дней. Благодаря таким напоминаниям пользователи всегда будут знать, когда произойдет оплата, что обеспечивает контроль за финансовыми средствами.

1.2 Анализ программного обеспечения аналогичного назначения

1.2.1 Обзор программного обеспечения SubsCrab

SubsCrab – это менеджер подписок, представленный компанией «Лаборатория Касперского». Данное программное обеспечение устанавливается на мобильные устройства с операционными системами IOS и Android.

К преимуществам приложения можно отнести:

- добавление и отслеживание подписки с дальнейшим их редактированием как основной информации, так и уведомлений;
- использования обширной и богатой базой данных с 4 тысячами существующих платных сервисов с 11 тысячами конкретных тарифов;
- автоматическое добавление подписки с помощью сканирования писем электронной почты;
- архивирования неактуальных подписок;
- организация подписок по спискам;
- кросс-платформенность;
- получение аналитики расходов и её отображение в календарном формате.

Недостатками данного приложения является:

- ограниченный функционал бесплатной версии;
- ограничение в настройке длительности периода подписки.

Основным способом добавления подписок является ручной. Пользователь самостоятельно вносит информацию о периоде оплаты, даты первой оплаты, уведомлений, методе оплаты, выбирать цвет подписки и загружать свое фото неё. Также приложение имеет возможность добавлять подписки с помощью сканирования писем почтового ящика, из которых оно автоматически определяет тариф и дату списания средств.

Само программное обеспечение имеет платный функционал, который представляется по подписке. В базовой версии пользователь может иметь только пять активных подписок и причем добавлять он их может только ручным способом. В платная же версия расширит его возможности, а именно позволяет добавлять неограниченное число подписок с функционалом создания списков, а также уведомлять пользователя о промокодах на тарифы подписок, которые хранятся в базе самого приложения. Таким образом, платная версия снимает ограничения, позволяя пользоваться всем предоставляемым функционалом.

1.2.2 Обзор программного обеспечения Subsee

Subsee – это трекинг подписок, представленный компанией «Tabee Networking» в формате мобильного приложения для устройств на базе IOS, которое представляет инструменты контроля над списанием средств и своевременного получения уведомлений с анализом финансовой активности.

Преимуществами приложения является:

- наличие 600 шаблонов подписок;
- поддержка и настройка уведомления;
- предоставление базовой аналитики о затратах;
- просмотр истории платежей;
- возможность приостановки, возобновления и архивирования подписок;
- одноразовая оплата полного доступа к полному функционалу приложения.

Недостатками данного приложения является:

- ограниченный базовый функционал;
- отсутствие календарного представления в аналитике;
- отсутствие автоматизации добавления подписок;
- только для операционной системы IOS;
- проблемы с кодировкой у некоторых подписок.

Subsee также имеет базовый функционал, который позволяет добавлять до 5 активных подписок с возможностью аналитики и уведомлениями. Для увеличения их количества необходимо провести оплату один раз. Полные функциональные возможности будут действовать бессрочно.

1.2.3 Обзор программного обеспечения Skaffer

Skaffer – это мобильный менеджер подписок для отслеживания расходов, который разработан под операционную систему IOS.

Преимуществами приложения является:

- набор существующих подписок;
- синхронизации с iCloud;
- разовая оплата с бессрочным доступом к полному функционалу;
- возможность отключения автопродления подписки;
- архивирование подписок.

Недостатками приложения является:

- ограниченный базовый функционал;
- отсутствует автоматическое обнаружение подписок;
- только для операционной системы IOS.

Важно обратить внимание, что у данного мобильного менеджера платных подписок достаточно ограничен базовый функционал. Это означает, что пользователь может иметь только 5 активных подписок, что не позволяет полностью интегрировать данное приложение в свою жизнедеятельность. Также в базовой версии невозможно посмотреть аналитику затрачиваемых средств на платные подписки и в целом нет информации о тратах за текущий месяц. Из подобной информации только известна плата за подписки, которую пользователь сам выставляет при добавлении.

Если пользователь захочет дополнить функционал единоразовой оплатой Pro версии приложения, то ему разблокируется вышеперечисленный функционал, а также появится возможность добавить несколько напоминаний на каждую активную подписку с синхронизацией на iCloud, которое обеспечивает резервное копирования для сохранности при смене устройства.

1.2.4 Сравнительная характеристика программного обеспечения аналогичного назначения

В ходе анализа и проведения сравнительной характеристики были получены результаты, которые выделяют преимущества и недостатки трех программных продуктов аналогичного назначения. Результаты сравнительного анализа представлены в таблице 1.

Таблица 1 – Результаты сравнительно анализа

Критерии	SubsCrab	Subsee	Skuffer
Платформы	IOS, Android	IOS	IOS
Добавление платных подписок	Вручную, по шаблону, автоматически	Вручную, по шаблону	Вручную, по шаблону
Каталог шаблонов	4000 сервисов	600 сервисов	Около 100 сервисов
Автоматическое обнаружение	Сканирование электронной почты	Отсутствует	Отсутствует
Уведомления о платежах	Настраиваемые, ограничены тремя вариантами: за один день, за три дня, за один день	Настраиваемые, ограничены следующими вариантами: в этот же день, за один день, за три дня, за пять дней, за 7 дней, за 10 дней, за 14 дней	Настраиваемые, множественные напоминания с гибкой настройкой
Финансовая аналитика	Расходы на месяц, все расходы, расходы за год, расходы за последний год	Расходы за месяц, расходы за 12 месяцев	В бесплатной версии отсутствует
Модель монетизации	Подписка 1 год 549 рублей или 2 года за 999 рублей за Premium	Разовая оплата 249 за Premium версию	Разовая оплата 249 за Premium версию
Наличие бесплатной версии	Присутствует	Присутствует	Присутствует
Ограничение бесплатной версии	Лимит в пять активных подписок	Лимит в пять активных подписок	Лимит в пять активных подписок, отсутствие аналитики

На основе полученных результатах сравнения, можно сделать вывод, что наилучшим вариантом для пользователей будет являться SubsCrub, так как он представляет инструмент автоматизации по добавлению подписок, наиболее большее число шаблонов, аналитику, которая сильно не ограничивается в бесплатной версии, календарь, показывающий дни списания средств. Также приложением можно пользоваться как на Android, так и на IOS.

Более бюджетным и не менее функциональным вариантом будет являться Subsee, которое используется только на IOS. Данное приложение имеет более дешевую цену за Premium версию приложения, однако отсутствуют средства автоматизации обнаружения платных подписок. При этом в бесплатной версии доступна аналитика, которая менее информативная, чем в SubsCrub, но при этом отражает основную и важную информацию.

Если нужна более гибкая настройка своевременных уведомлений, то стоит выбрать приложение Skaffer, оно предоставляет больше параметров для конфигурирования уведомлений. Имеет такую же цену за Premium, как и у Subsee, однако в бесплатной версии отсутствует функционал аналитики, что может стать важным фактором при выборе приложения. Работает на операционной системе IOS.

Хочется отметить, что все рассматриваемые приложения являются мобильными и нативными под конкретные платформы, что вносит определенные ограничения для пользователей, а также отсутствует возможность работы с групповыми подписками. Все приложения ориентированы на личные подписки. Разрабатываемая информационная система в выпускной квалификационной работе восполнит отсутствие бесплатной функциональности без каких-либо ограничений с гибкой настройкой своевременных уведомлений и обеспечит кроссплатформенность системы.

1.3 Постановка задачи на разработку

Целью данной работы является создание информационной системы по управлению платными подписками, которая должна позволять регистрацию

новых пользователей, а также вход уже существующих с возможность редактирования своего профиля и выход из аккаунта.

Основным функционалом информационно системы будет являться взаимодействие с платными подписками. Это подразумевает то, что пользователь должен иметь возможность добавлять личную или групповую платную подписку, внося данные с настройкой своевременного уведомления, с предоставляемым функционалом редактирования, приостановки, возобновления, удаления и фильтрацией подписок.

Также важно, чтобы системы предоставляла анализ трат на подписки с инфографикой и возможность сгенерировать pdf-файл, чтобы пользователь мог отследить финансовые затраты.

Стоит отметить, что система должна иметь функционал, связанный с групповыми подписками. Это означает возможность владельцу приглашать в подписку других пользователей и разделять её сумму между участниками. Также участник групповой подписки имеет возможность присоединиться к ней или выйти по собственному желанию, а владелец имеет функционал удаления участников.

Для реализации информационной системы необходимо выявить функциональные требования, а также спроектировать и реализовать клиентское, серверное приложение с базой данных.

Клиентское приложение, с которым будут взаимодействовать пользователи, должно иметь понятный и информативный интерфейс с удобной навигацией по основным страницам с возможностью адаптации экрана как под мобильные устройства, так и под ПК формат.

Серверное приложение будет отвечать основную логику работы системы. Оно будет принимать данные от клиентского приложения о каких-либо взаимодействиях с подписками и их изменениями, а также регистрировать новых пользователей и осуществлять вход в систему существующих. При этом важно обеспечить стабильную работу и защиту пользовательских данных при взаимодействии между клиентом и сервером.

Также для хранения информации о пользователях и их подписок необходимо спроектировать базу данных, с которой будет взаимодействовать серверное приложение.

1.4 Планирование и функционально-стоимостный анализ разработки

Важным этапом в данной работе является планирование и функционально-стоимостный анализ.

Планирование позволяет в установленные сроки разработать программное обеспечение с требуемым качеством. В нем учитываются риски и ограничения, ресурсы, сроки. Основной целью планирования является достижение поставленных целей самым эффективным способом с помощью анализа, конкретизацией и упорядочиванием действий.

Функционально-стоимостный анализ – метод системного исследования функций отдельного изделия или технологического, производственного процесса, организационной структуры, ориентированный на повышение эффективности использования ресурсов путем оптимизации соотношения между потребительскими свойствами объекта исследования и затратами на его разработку, производство и эксплуатацию [2].

Основной целью функционально-стоимостного анализа является в достижении максимальной реализации поставленных целей при рациональном использовании ресурсов.

Планирование выполнялось с учётом логической последовательности этапов разработки, взаимозависимости задач и предполагаемых трудозатрат на каждом этапе. Данный подход способствует более чёткому контролю хода выполнения работ.

В таблице 2 предоставлены задачи проекта, которые были сформулированы в ходе планирования, с их последовательностью и длительностью выполнения. Для графика была выбрана шестидневная рабочая неделя.

Таблица 2 – Список задач проекта

№	Задача	Длительность	Предшественник
1	Начало работы над проектом	-	-
2	Анализ предметной области и постановка задачи на разработку	-	-
3	Анализ предметной области	2 дня	1
4	Анализ программного обеспечения аналогичного назначения	2 дня	3
5	Планирование и функционально-стоимостный анализ	2 дня	4
6	Анализ предметной области и постановка задачи на разработку завершены	-	5
7	Анализ требований к информационной системе	-	2
8	Анализ функциональных требований информационной системы	2 дня	6
9	Анализ требований к серверному приложению	2 дня	8
10	Анализ требований к клиентскому приложению	1 день	9
11	Анализ требований к информационной системе завершены	-	10
12	Проектирование информационной системы	-	7
13	Проектирование клиент-серверной архитектур	2 дня	11
14	Проектирование базы данных	1 день	13
15	Проектирование серверного приложения	3 дня	14
16	Проектирование интерфейса	2 дня	15
17	Проектирование клиентского приложения	3 дня	16

Продолжение таблицы 2

№	Задача	Длительность	Предшественник
18	Выбор технологий и инструментов разработки	1 день	17
19	Проектирование информационной системы завершено	-	18
20	Разработка информационной системы	-	12
21	Развертывание базы данных	1 день	19
22	Разработка серверного приложения	5 дней	21
23	Разработка клиентского приложения	5 дней	22
24	Разработка информационной системы завершена	-	23
25	Тестирование	-	20
26	Тестирование программных средств	4 дня	24
27	Оценка качеств программного кода	2 дня	26
28	Тестирование завершено	-	27
29	Сдача проекта	-	25
30	Оформление документации	4 дня	28
31	Ввод в эксплуатацию	1 день	30
32	Сдача проекта завершена	-	31
33	Конец работы над проектом	-	32

Следующим этапом будет являться определением ресурсов и назначение их на задачи проекта. В таблице 3 предоставлен список используемых ресурсов.

Таблица 3 – Список используемых ресурсов

Название ресурса	Тип ресурса	Стандартная ставка	Сверхурочная ставка
Аналитик	Трудовой	270 Руб/ч	
Дизайнер	Трудовой	260 Руб/ч	
Программист	Трудовой	300 Руб/ч	
Тестировщик	Трудовой	260 Руб/ч	
Интернет	Затраты	-	-
Компьютер	Материальный	40 000 руб	

В таблице 4 отображено распределение ресурсов по задачам проекта.

Таблица 4 – Распределение ресурсов по задачам

№	Задача	Ресурсы	Единицы
1	Начало работы над проектом	-	-
2	Анализ предметной области и постановка задачи на разработку	-	-
3	Анализ предметной области	Аналитик	100%
4	Анализ программного обеспечения аналогичного назначения	Аналитик	100%
5	Планирование и функционально-стоимостный анализ	Аналитик	100%
6	Анализ предметной области и постановка задачи на разработку завершены	-	-
7	Анализ требований к информационной системе	-	-
8	Анализ функциональных требований информационной системы	Аналитик	100%
9	Анализ требований к серверному приложению	Аналитик	100%
		Программист	50%
10	Анализ требований к клиентскому приложению	Аналитик	100%
		Программист	50%
11	Анализ требований к информационной системе завершены	-	-
12	Проектирование информационной системы	-	-
13	Проектирование клиент-серверной архитектур	Программист	100%
14	Проектирование базы данных	Программист	100%

Продолжение таблицы 4

№	Задача	Ресурсы	Единицы
15	Проектирование серверного приложения	Программист	100%
16	Проектирование интерфейса	Дизайнер	100%
		Программист	50%
17	Проектирование клиентского приложения	Программист	100%
		Дизайнер	50%
18	Выбор технологий и инструментов разработки	Программист	100%
19	Проектирование информационной системы завершено	-	-
20	Разработка информационной системы	-	-
21	Развертывание базы данных	Программист	100%
		Интернет	100 Руб
22	Разработка серверного приложения	Программист	100%
		Интернет	300 Руб
23	Разработка клиентского приложения	Программист	100%
		Интернет	300 Руб
24	Разработка информационной системы завершена	-	-
25	Тестирование	-	-
26	Тестирование программных средств	Тестирующий	100%
		Интернет	300 Руб
27	Оценка качеств программного кода	Тестирующий	100%
28	Тестирование завершено	-	-
29	Сдача проекта	-	-
30	Оформление документации	Программист	100%
31	Ввод в эксплуатацию	Программист	100%
		Интернет	250 Руб
32	Сдача проекта завершена	-	-
33	Конец работы над проектом	-	-

Для расчета затрат на разработку, составления графика задач с определением их длительность и также всего проекта было использовано приложение Microsoft Office Project.

Microsoft Office Project – программа управления проектами разработки корпорации Microsoft, поставляемая как самостоятельный продукт, так и в составе расширенных редакций пакета Microsoft Office.

Основными функциями данного приложения являются управление проектами, ресурсами, контролирование сроков и бюджета, анализ критического пути, отчетность и визуализация, в том числе построение диаграммы Ганта.

В Microsoft Office Project был создан проект, куда было внесена информация о перечне задач с назначенными на них ресурсами. После этого была сформирована диаграмма Ганта, которая позволяет визуальным образом отобразить временные рамки выполнения задач, представленная на рисунке 1.

За дату начало проекта было выбрано 8 апреля 2025 года. С учетом сформированной диаграммы, окончание работ по проекту запланировано на 13 июня 2025 года.

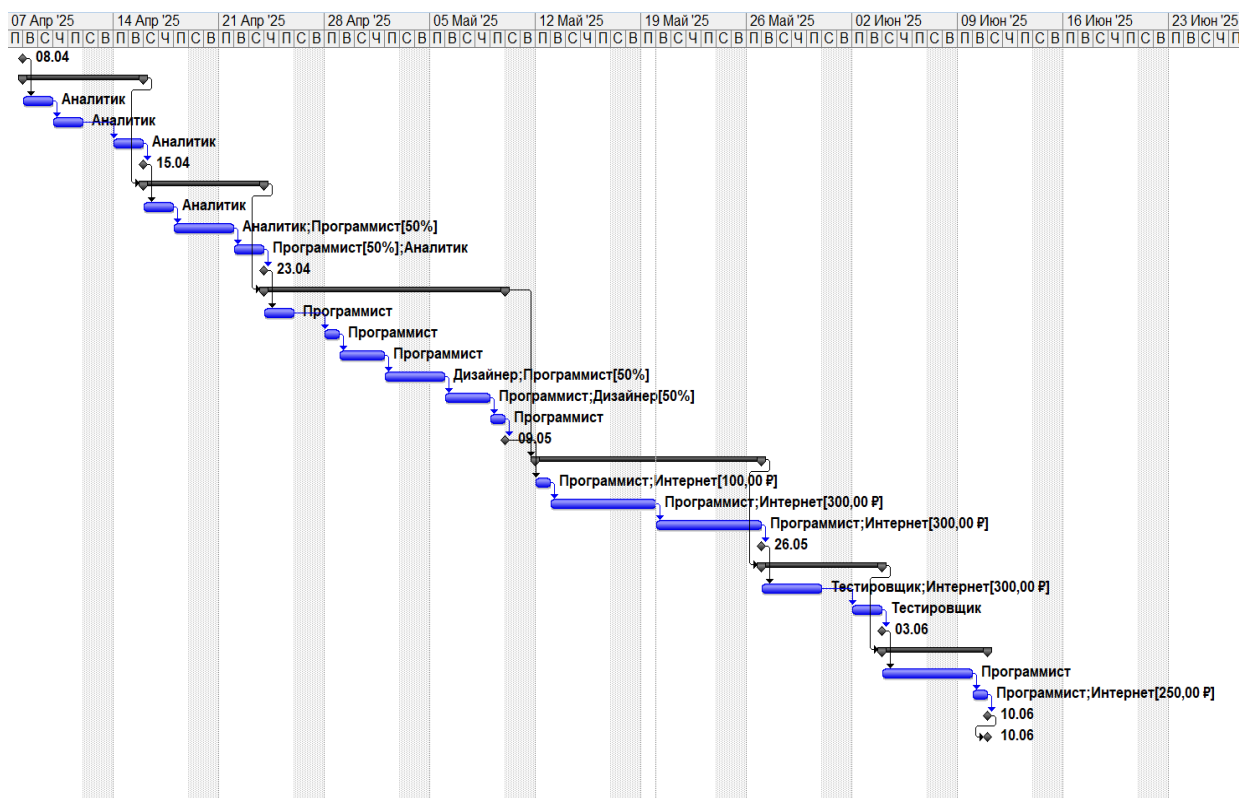


Рисунок 1 – Диаграмма Ганта с распределением ресурсов

Далее была сформирована отчетность по использованию ресурсов, представленных на рисунке 2.

Использование ресурсов от Вт 20.05.25
Система управления платными подписками

	Итого
Программист	232 ч
Анализ требований к серверному приложению	8 ч
Анализ требований к клиентскому приложению	8 ч
Проектирование клиент-серверной архитектур	16 ч
Проектирование базы данных	8 ч
Проектирование серверного приложения	24 ч
Проектирование интерфейса	8 ч
Проектирование клиентского приложения	24 ч
Выбор технологий и инструментов разработки	8 ч
Развертывание базы данных	8 ч
Разработка серверного приложения	40 ч
Разработка клиентского приложения	40 ч
Оформление документации	32 ч
Ввод в эксплуатацию	8 ч
Аналитик	88 ч
Анализ предметной области	16 ч
Анализ программного обеспечения аналогичного назначения	16 ч
Планирование и функционально-стоимостный анализ	16 ч
Анализ функциональных требований информационной системы	16 ч
Анализ требований к серверному приложению	16 ч
Анализ требований к клиентскому приложению	8 ч
Дизайнер	28 ч
Проектирование интерфейса	16 ч
Проектирование клиентского приложения	12 ч
Тестировщик	48 ч
Тестирование программных средств	32 ч
Оценка качества программного кода	16 ч
Интернет	
Развертывание базы данных	
Разработка серверного приложения	
Разработка клиентского приложения	
Тестирование программных средств	
Ввод в эксплуатацию	
Компьютер	
Итого	396 ч

Рисунок 2 – Отчет об использовании ресурсов проекта

Данный отчет показывает, что время использования ресурсов за все время длительности проекта будет составлять 396 часов. Далее рисунке 3 представлен отчет об общих затратах.

Отчет о бюджете от Вт 20.05.25
Система управления платными подписками

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты	Базовые	Отклонение
22	Разработка серверного приложения	0,00 Р	Пропорциональное	12 300,00 Р	0,00 Р	12 300,00 Р
23	Разработка клиентского приложения	0,00 Р	Пропорциональное	12 300,00 Р	0,00 Р	12 300,00 Р
17	Проектирование клиентского прилож	0,00 Р	Пропорциональное	10 320,00 Р	0,00 Р	10 320,00 Р
30	Оформление документации	0,00 Р	Пропорциональное	9 600,00 Р	0,00 Р	9 600,00 Р
26	Тестирование программных средств	0,00 Р	Пропорциональное	8 620,00 Р	0,00 Р	8 620,00 Р
15	Проектирование серверного прилож	0,00 Р	Пропорциональное	7 200,00 Р	0,00 Р	7 200,00 Р
9	Анализ требований к серверному пр	0,00 Р	Пропорциональное	6 720,00 Р	0,00 Р	6 720,00 Р
16	Проектирование интерфейса	0,00 Р	Пропорциональное	6 560,00 Р	0,00 Р	6 560,00 Р
13	Проектирование клиент-серверной с	0,00 Р	Пропорциональное	4 800,00 Р	0,00 Р	4 800,00 Р
10	Анализ требований к клиентскому п	0,00 Р	Пропорциональное	4 560,00 Р	0,00 Р	4 560,00 Р
3	Анализ предметной области	0,00 Р	Пропорциональное	4 320,00 Р	0,00 Р	4 320,00 Р
4	Анализ программного обеспечения	0,00 Р	Пропорциональное	4 320,00 Р	0,00 Р	4 320,00 Р
5	Планирование и функционально-сто	0,00 Р	Пропорциональное	4 320,00 Р	0,00 Р	4 320,00 Р
8	Анализ функциональных требований	0,00 Р	Пропорциональное	4 320,00 Р	0,00 Р	4 320,00 Р
27	Оценка качества программного кода	0,00 Р	Пропорциональное	4 160,00 Р	0,00 Р	4 160,00 Р
31	Ввод в эксплуатацию	0,00 Р	Пропорциональное	2 650,00 Р	0,00 Р	2 650,00 Р
21	Развертывание базы данных	0,00 Р	Пропорциональное	2 500,00 Р	0,00 Р	2 500,00 Р
14	Проектирование базы данных	0,00 Р	Пропорциональное	2 400,00 Р	0,00 Р	2 400,00 Р
18	Выбор технологий и инструментов р	0,00 Р	Пропорциональное	2 400,00 Р	0,00 Р	2 400,00 Р
1	Начало работы проекта	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
6	Анализ предметной области и поста	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
11	Анализ требований к информацион	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
19	Проектирование информационной с	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
24	Разработка информационной систем	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
28	Тестирование завершено	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
32	Сдача проекта завершена	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
33	Конец работы над проектом	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
		0,00 Р		114 370,00 Р	0,00 Р	114 370,00 Р

Рисунок 3 – Отчет о общих затратах проекта

В соответствии с отчетом о общих затратах проекта можно сделать вывод, что для разработки информационной системе по управлению платными подписками необходимо 114370 рублей.

2 Анализ требований к информационной системе

2.1 Анализ функциональных требований информационной системы

На основании постановки задачи на разработку были выявлены следующие функциональные требования к информационной системе:

- зарегистрироваться в системе;
- войти в систему;
- выйти из системы;
- подписаться на уведомления;
- добавить личную платную подписку;
- добавить групповую платную подписку;
- редактировать личную платную подписку;
- редактировать групповую платную подписку;
- вывести прогноз трат на подписки;
- сгенерировать pdf-файл с прогнозом трат на подписки;
- отфильтровать подписки по статусу;
- приостановить подписку;
- возобновить подписку;
- удалить подписку;
- подтвердить перевод средств на групповую подписку;
- пригласить участника в групповую подписку;
- присоединиться к групповой подписке;
- выйти из групповой подписки;
- удалить участника из групповой подписки.

Также была сформирована диаграмма вариантов использования с учетом вышеперечисленных требований, которая отражает основные взаимодействия пользователя и помогает визуализировать поведение системы при выполнении ключевых операций [3]. Диаграмма представлена на рисунке 4.

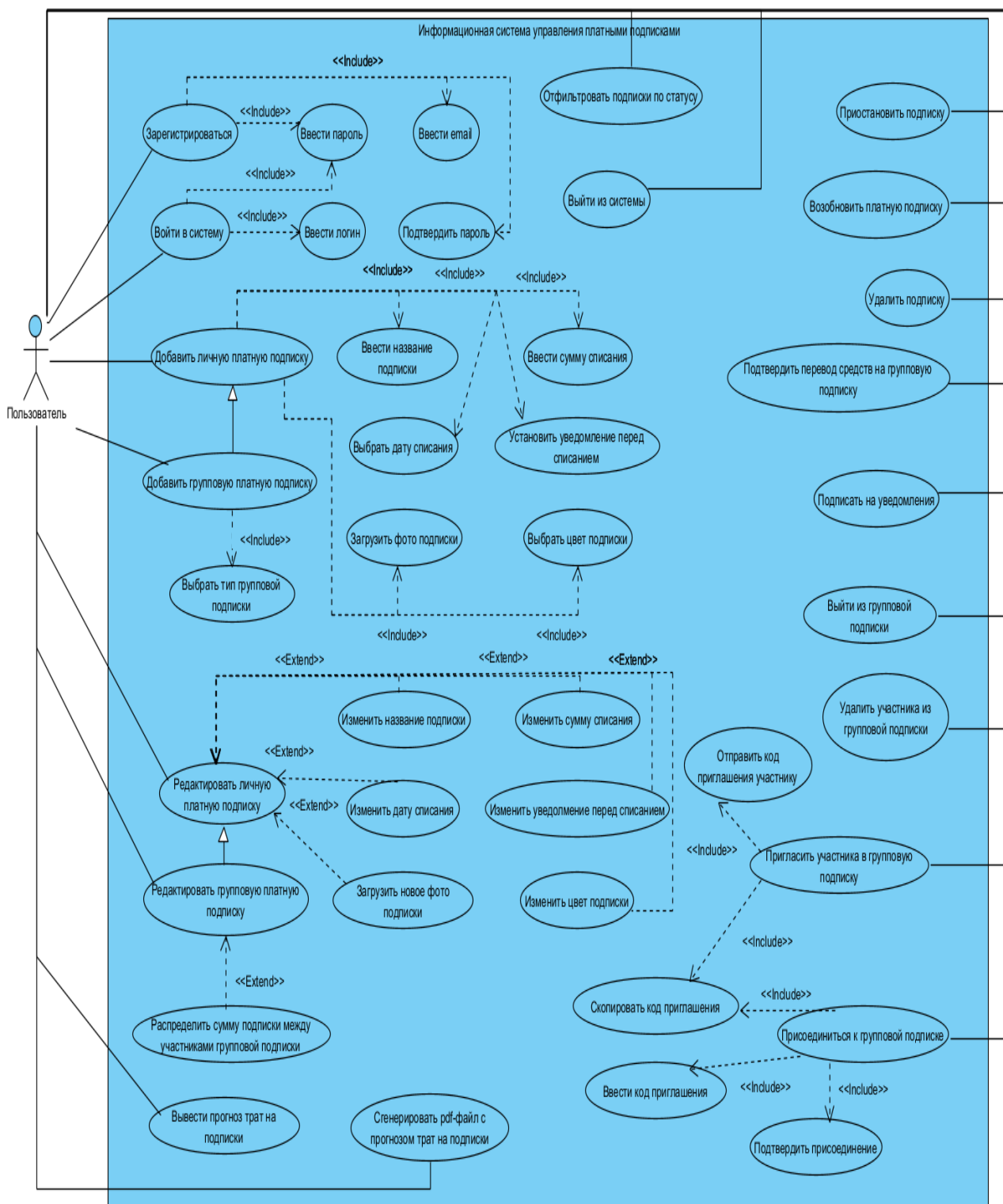


Рисунок 4 – Диаграмма вариантов использования информационной системы управления платными подписками

На рисунке 5 представлена спецификация варианта использования «Добавить личную платную подписку».

Добавить личную платную подписку	
Id: 1	
Описание:	Добавление личной платной подписки в информационную систему
Действующие лица:	Пользователь, система
Предусловие:	Пользователь авторизован
Основной поток:	<ol style="list-style-type: none"> 1. Пользователь нажатием кнопки вызывает окно добавления платной подписки 2. Система выводит окно добавления платной подписки 3. Пользователь заполняет обязательные поля новой платной подписки 4. Пользователь устанавливает тип подписки на личный 5. Пользователь подтверждает правильность внесенных данных, нажатием кнопки 6. Система принимает и сохраняет новую личную подписку пользователя
Постусловие:	Личная подписка добавлена и отображается в системе

Рисунок 5 – Спецификация варианта использования «Добавить личную платную подписку»

На рисунке 6 представлена спецификация варианта использования «Редактировать групповую платную подписку».

Редактировать групповую платную подписку	
Id: 2	
Описание:	Редактирование групповой платной подписки пользователем в информационной системе
Действующие лица:	Пользователь, система
Предусловие:	Пользователь авторизован
Основной поток:	<ol style="list-style-type: none"> 1. Пользователь выбирает групповую платную подписку из списка или из календаря 2. Пользователь вызывает окно редактирования групповой платной подписки 3. Система выводит пользователю окно редактирования групповой платной подписки 4. Пользователь вносит изменения в поля групповой платной подписки 5. Пользователь подтверждает внесенные данные нажатием кнопки 6. Система фиксирует изменения и сохраняет их
Постусловие:	Групповая подписка отредактирована, изменения отображаются в системе

Рисунок 6 – Спецификация варианта использования «Редактировать групповую платную подписку»

На рисунке 7 представлена спецификация варианта использования «Приостановить платную подписку».

Приостановить платную подписку	
Id: 3	
Описание:	Приостановление учета платной подписки в информационной системе
Действующие лица:	Пользователь, система
Предусловие:	Пользователь авторизован
Основной поток:	<ol style="list-style-type: none"> 1. Пользователь выбирает платную подписку из списка или из календаря 2. Пользователь с помощью нажатия на кнопку приостанавливает платную подписку 3. Система фиксирует действие пользователя и изменяет статус подписки
Постусловие:	Платная подписка приостановлена и не учитывается в анализе информационной система

Рисунок 7 – Спецификация варианта использования «Приостановить платную подписку»

В конечном итоге была сформирована диаграмма вариантов использования, которая отображает все выявленные функциональные требования информационной системы управления платными подписками, а также были рассмотрены некоторые спецификации вариантов использования.

2.2 Требования к серверному программному обеспечению

Серверное приложение является важной частью информационной системы. Оно отвечает за работоспособность бизнес-логики, обработку и хранения данных пользователей, взаимодействие с клиентским приложением через REST API, возможности регистрации нового и авторизации существующего пользователя [4].

Сервер должен обеспечивать безопасную аутентификацию пользователей с надежным хранением информации о их аккаунтах. Также пароли не должны храниться в открытом виде, они должны быть преобразованы

криптографическими хэш-функциями, что делает невозможным восстановить пароль в случае утечки данных [5].

Обязательно должны быть API-эндпоинты, с помощью которых серверное приложение будет принимать запросы на обработку различных действий с подписками от пользователя информационной системы.

Особенно важно, чтобы все операции были доступны только авторизованным пользователям, а каждое действие проверялось на возможность его совершить с учетом прав доступа. Это обеспечивает конфиденциальность данных и позволяет защитить от несанкционированного доступа к чужим данным.

В результате серверное приложение будет являть центральным компонентом разрабатываемой информационной системы, которое совмещает в себе функциональные возможности с защитными механизмами, обеспечивающие корректное и безопасное взаимодействие с остальными компонентами.

2.3 Требования к клиентскому программному обеспечению

Клиентское приложение представляет собой часть пользовательскую часть информационной системы, которая обеспечивает пользователю взаимодействие через графический интерфейс в браузере. Через него пользователь имеет доступ к функциональности системы посредством интерфейса. Также оно осуществляет отправку запросов серверному приложению и отображение в понятной форме пользователю приходящих ответов.

Основными функциями клиентского приложения является авторизация, регистрация пользователей, отображение подписок с возможностью их редактирования и добавления, изменения статусов подписок, вывод анализа финансовых расходов, а также получение уведомления о скором списании средств. Взаимодействие между клиентским и серверным приложением будет обеспечиваться через REST API с использованием формата JSON.

Интерфейс должен включать в себя визуальные элементы навигации по системе, кнопки, формы для ввода данных. Также он должен иметь главную страницу, на которой отображается календарь и анализ расходов за месяц с элементами навигации по другим разделам, страница с подписками, отображающая все подписки, которые имеет пользователь, форма редактирования подписок, страница анализа с возможностью настраивать интервал времени, а также с выводом инфографики.

Клиентская часть ответственна за валидацию вводимых данных пользователем перед отправкой на сервер, что позволяет на этапе ввода указать на ошибки. Однако основная проверка будет происходить на серверной части информационной системы.

Для обеспечения безопасности данных пользователя клиентское приложение не должно хранить сохранять пароли ли другую подобную информацию в открытом виде.

Таким образом, клиентское приложение предоставляет доступ к функциям системы через интерфейс, с которым пользователь взаимодействует для получения желаемого результата.

3 Проектирование информационной системы

3.1 Архитектура информационной системы

Информационная система будет иметь трехуровневую клиент-серверную архитектуру.

Клиент-серверная архитектура – архитектурный стиль, который разделяет функциональность на интерфейсную и серверную [6].

Трехуровневой архитектура – архитектурная модель, имеющая три компонента, а именно клиентское приложение, серверное приложение и базу данных [6].

С помощью клиентского приложения пользователь взаимодействует с самой системой через интерфейс, который отображает данные, получаемые с сервера. Серверное приложение является промежуточным уровнем, содержащим обработку данных с их валидацией, реализацию бизнес-логики, аутентификацию и авторизацию пользователей. База данных отвечает за обеспечение целостности, безопасности хранения, доступности и управление данными. В ней будет храниться вся информация о пользователях и их подписках.

Каждый компонент является автономным и взаимодействуют с другими компонентами через определенные интерфейсы и протоколы. Это обеспечивает независимое масштабирование и изменение каждого уровня отдельно, что повышает сопровождение системы и облегчает разработку и тестирование.

В рамках проектирования архитектуры были определены роли и задачи каждого из уровней, что позволило достичь высокой степени модульности и упростить процессы разработки и сопровождения. Разделение на клиентский, серверный и уровень хранения данных также позволило обеспечить более гибкую интеграцию с внешними системами и реализацию дополнительных сервисов в будущем.

На рисунке 8 изображена диаграмма развертывания, которая отображает архитектуру информационной системы.

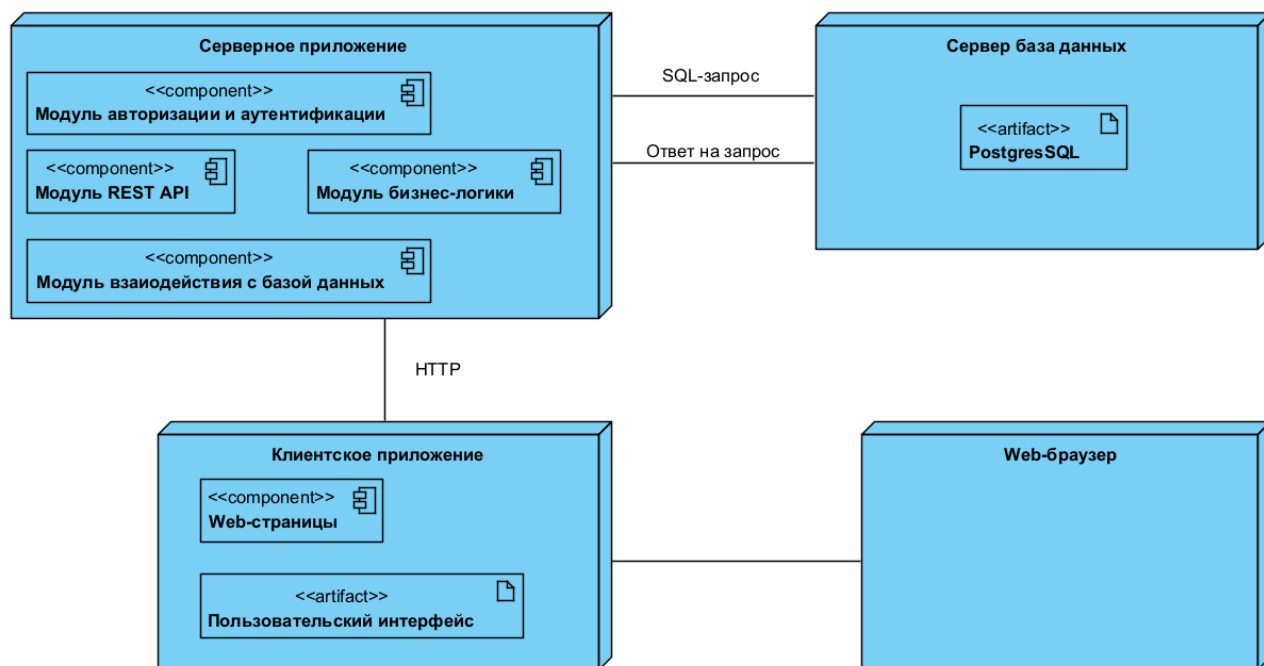


Рисунок 8 – Диаграмма развертывания информационной системы

Выбор в пользу данной архитектуры был сделан из-за гибкости и удобства разработки, так как наличие независимостей компонентов позволяет их проектировать и разрабатывать отдельно друг от друга, также изменение одного уровня минимально влияют на другие компоненты системы. Стоит выделить повышенную безопасность данного архитектурного стиле, так как обработка данных и их хранения находятся на разных уровнях, что уменьшает риск несанкционированного доступа к данным напрямую через клиентское приложение, так как каждый запрос проходит через строгий контроль и проверку на серверном приложении.

3.2 Проектирование базы данных

3.2.1 Концептуальное моделирование базы данных

Для модели базы данных были выделены сущности, а именно пользователь, директория, метаданные файла, которые представлены на ERD-диаграмме концептуальной модели, изображенная на рисунке 9 [7].

Entity Relationship Diagram (ERD) – модель данных, описывающая структуру информации с помощью сущностей, атрибутов и связей.

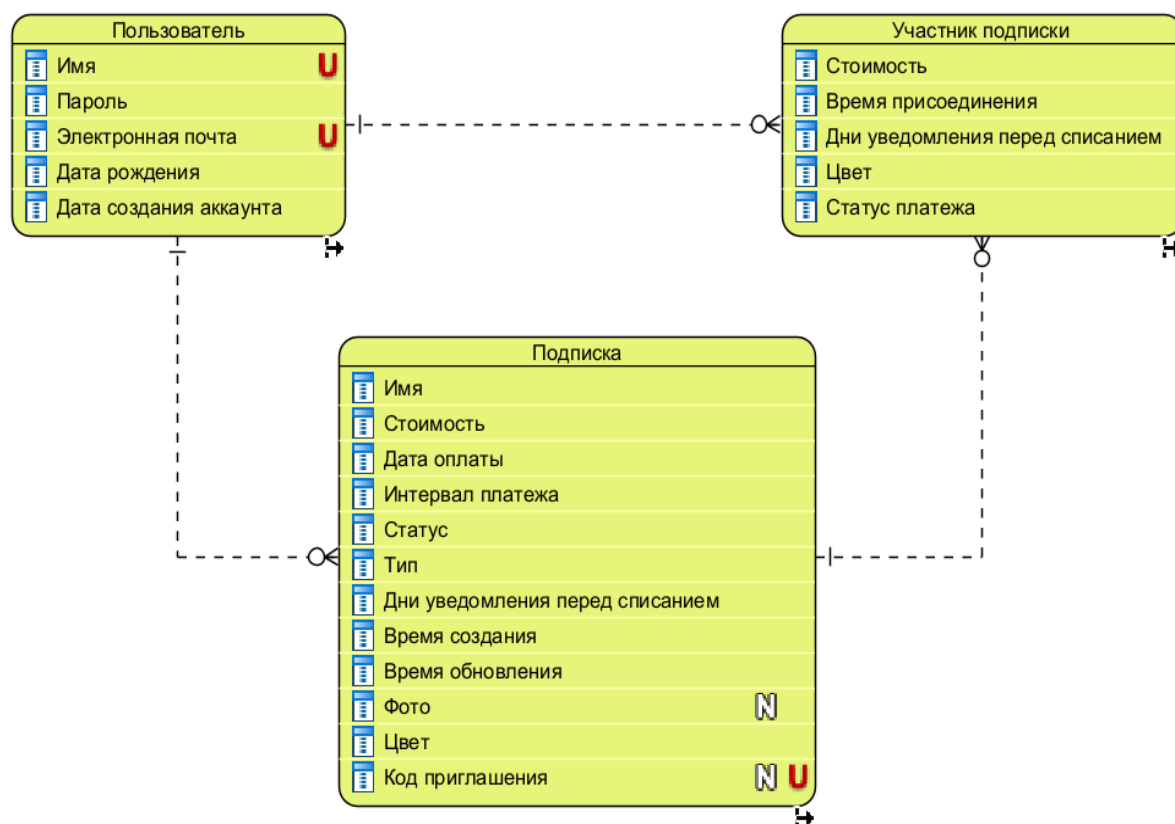


Рисунок 9 – ERD-диаграмма концептуальной модели базы данных

В таблице 5 приведено описание сущностей.

Таблица 5 – Описание сущностей

Название сущности	Описание	Категория
Пользователь	Описывает учетной записи пользователя	Стержневая
Подписка	Включает в себя необходимую информацию	Стержневая
Участник подписки	Хранят информацию о участнике, который подключен к групповой подписке	Характеристическая

Также описание атрибутов, которые выделяют характеристики и свойства конкретной сущности, представлено в таблице 6.

Таблица 6 – Описание атрибутов

Название сущности	Атрибут	Описание атрибута	Ограничения
Пользователь	Имя	Имя пользователя, отображается в интерфейсе	Уникальность
	Пароль	Хэш пароля, используется для аутентификации	
	Электронная почта	Email-адрес пользователя, используется для входа и уведомлений	Уникальность
	Дата рождения	Дата рождения пользователя	
	Дата создания аккаунта	Дата регистрации пользователя в системе	
Подписка	Имя	Название подписки, задаётся пользователем	
	Стоимость	Общая сумма подписки	
	Дата оплаты	Дата ближайшего списания	
	Интервал платежа	Частота списаний в днях	
	Статус	Статус подписки, который имеет значение либо активная, либо приостановлена	
	Тип	Тип подписки, а именно групповая или личная	
	Дни уведомления перед списанием	Количество дней до списания, когда отправляется уведомление	
	Время создания	Дата и время создания подписки	
	Время обновления	Дата и время последнего изменения подписки	

Продолжение таблицы 6

Название сущности	Атрибут	Описание атрибута	Ограничения
	Фото	Изображение или иконка подписки	
	Цвет	Цветовая метка для визуального обозначения подписки	
	Код приглашения	Уникальный код для приглашения других пользователей	Уникальность
Участник подписки	Стоимость	Индивидуальная сумма пользователя в подписке	
	Время присоединения	Дата, когда пользователь присоединился к подписке	
	Дни уведомления перед списанием	Индивидуальное количество дней до уведомления	
	Цвет	Цветовая метка участника в рамках подписки	
	Статус платежа	Обозначает, оплачен ли текущий период участия	

Описание связей между сущностями, отражающее типы отношений, степень связи и обязательность, приведено в таблице 7 и служит основой для построения логической модели базы данных.

Таблица 7 – Описание связей между сущностями

Сущность	Сущность	Описание связи	Степень связи	Обязательность
Пользователь	Подписка	Пользователь имеет множество подписок. Подписка принадлежит одному пользователю.	1:M	Не обязательно

Продолжение таблицы 7

Сущность	Сущность	Описание связи	Степень связи	Обязательность
Пользователь	Участник подписки	Пользователь имеет возможность множество раз быть участником разных групповых подписок.	1:M	Не обязательно
Подписка	Участник подписки	Одна подписка содержит множество участников. Одни участник принадлежат одной подписке.	1:M	Необязательно

Благодаря концептуальной модели, выделилась структура системы, которая отображает свойства её элементов и причинно-следственные связи.

3.2.2 Логическое моделирование базы данных

Для построения логической модели базы данных была разработана ERD-диаграмма, содержащая сущности, которые будут участвовать в функционировании информационной системы, их атрибуты и взаимосвязи. Каждая сущность представляет собой объект предметной области и содержит набор атрибутов, характеризующие значимые свойства, необходимые для реализации бизнес-логики приложения [7].

Кроме определения структуры сущностей и их взаимосвязей, логическая модель базы данных позволяет выявить избыточность данных, определить правила целостности и обеспечить согласованность всех компонентов системы. Особое внимание при построении логической ERD-диаграммы уделялось корректной формализации связей между сущностями, а также обозначению первичных и внешних ключей, необходимых для реализации реляционной структуры данных.

Логическая ERD-диаграмма, представленная на рисунке 10.

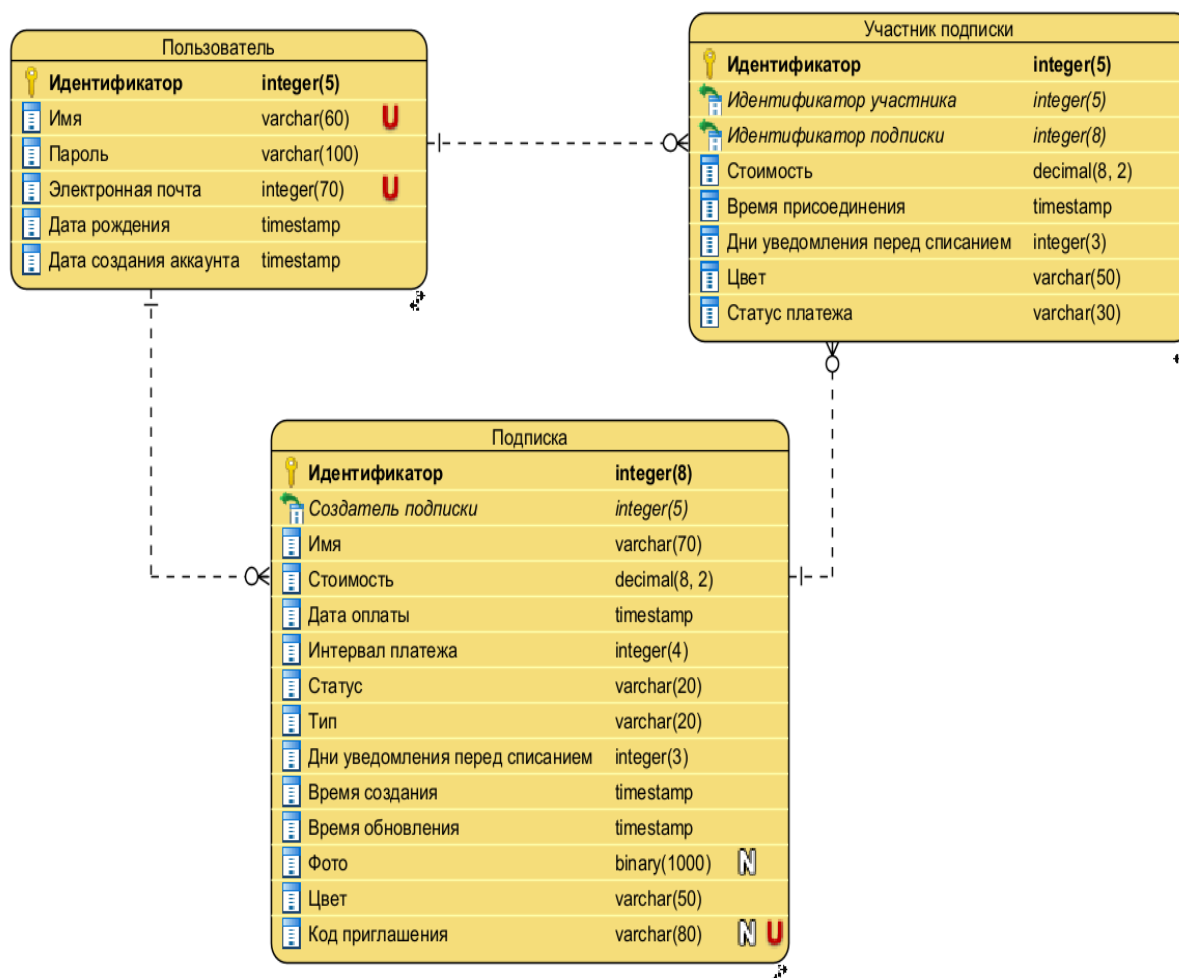


Рисунок 10 – Логическая ERD-диаграмма базы данных

Описание сущностей приведено в таблице 8.

Таблица 8 – Описание сущностей

Название сущности	Описание	Категория
Пользователь	Описывает учетной записи пользователя	Стержневая
Подписка	Включает в себя необходимую информацию	Стержневая
Участник подписки	Хранят информацию о участнике, который подключен к групповой подписке.	Характеристическая

Описание атрибутов представлено в таблице 9.

Таблица 9 – Описание атрибутов

Сущность	Атрибут	Тип данных	Размер	Роль	Обязательный	Ограничения
Пользователь	Идентификатор	Целый	5	РК	Да	1 – 100000
	Логин	Символьный	60		Да	Уникальность
	Хеш пароля	Символьный	50		Да	
	Электронная почта	Символьный	70		Да	
	Дата рождения	Дата/время			Да	
	Время создания аккаунта	Дата/время			Да	
Подписка	Идентификатор	Целый	8	РК	Да	1 – 100000000
	Создатель подписки	Целый	5	FK	Да	1 – 100000
	Имя	Символьный	20		Да	
	Стоимость	Вещественный	10		Да	
	Дата оплаты	Дата/время			Да	
	Интервал платежа	Целый	4		Да	1 – 10000
	Статус	Символьный	20		Да	
	Тип	Символьный	20		Да	
	Дни уведомления перед списанием	Целый	3		Да	1 – 1000
	Время создания	Дата/время			Да	
	Время обновления	Дата/время			Да	
	Фото	Двоичный			Нет	
	Цвет	Символьный	50			
	Код приглашения	Символьный	80		Нет	Уникальность
Участник групповой подписки	Идентификатор	Целый	4	РК	Да	1 – 10000
	Идентификатор участника	Целый	5	FK	Да	1 – 100000

Продолжение таблицы 9

Сущность	Атрибут	Тип данных	Размер	Роль	Обязательный	Ограничения
	Идентификатор подписки	Целый	8	FK	Да	1 – 100000000
	Стоимость	Вещественный	10		Да	
	Время присоединения	Дата/время	20		Да	
	Дни уведомления перед списанием	Целый	3		Да	1 – 1000
	Цвет	Символьный	50			
	Статус платежа	Символьный	30		Да	1 – 1000

Описание связей между сущностями представлено в таблице 10.

Таблица 10 – Описание связей между сущностями

Сущность	Сущность	Описание связи	Степень связи	Обязательность
Пользователь	Подписка	Пользователь имеет множество подписок. Подписка принадлежит одному пользователю.	1:M	Не обязательно
Пользователь	Участник подписки	Пользователь имеет возможность множество раз быть участником разных групповых подписок.	1:M	Не обязательно
Подписка	Участник подписки	Одна подписка содержит множество участников. Одни участник принадлежат одной подписке.	1:M	Необязательно

Выше описанные атрибуты в таблице и сущности позволяют организовать хранения данных для последующего их использования.

3.3 Проектирование серверного программного обеспечения

3.3.1 Назначение и функции серверного приложения

Серверное приложение является основным компонентом информационной системы управления платными подписками. Оно обеспечивает обработку клиентских запросов, безопасностью доступа, а также реализует бизнес-логику с управлением данных в соответствии с требованиями.

Главной задачей серверного приложения является обеспечение надежного и безопасного взаимодействия с клиентским интерфейсом, обработки входящих запросов, выполнение операций с базой данных, реализацией бизнес-правил системы и предоставления интерфейса для клиентского приложения.

Сервер также ответственен за:

- регистрацию и авторизацию пользователей;
- генерацию и проверку токенов аутентификации;
- управление подписками и участниками групп;
- создание аналитических отчетов;
- отправку уведомлений в Telegram пользователя;
- обработку пользовательских данных.

Кроме того, в систему была интегрирована отправка уведомлений через Telegram, что позволяет пользователю своевременно получать напоминания о предстоящих списаниях. Для этого реализовано взаимодействие с Telegram Bot API, обеспечивающее передачу сообщений напрямую в чат пользователя после его авторизации [8].

Особое внимание при проектировании было уделено безопасности и надёжности обработки пользовательских данных. Все маршруты серверного приложения проходят через систему аутентификации, что обеспечивает защиту информации и контроль доступа.

В ходе проектирование серверного приложения была разработана диаграмма деятельности, представленная на рисунке 11, которая описывает процесс добавления подписки в систему.

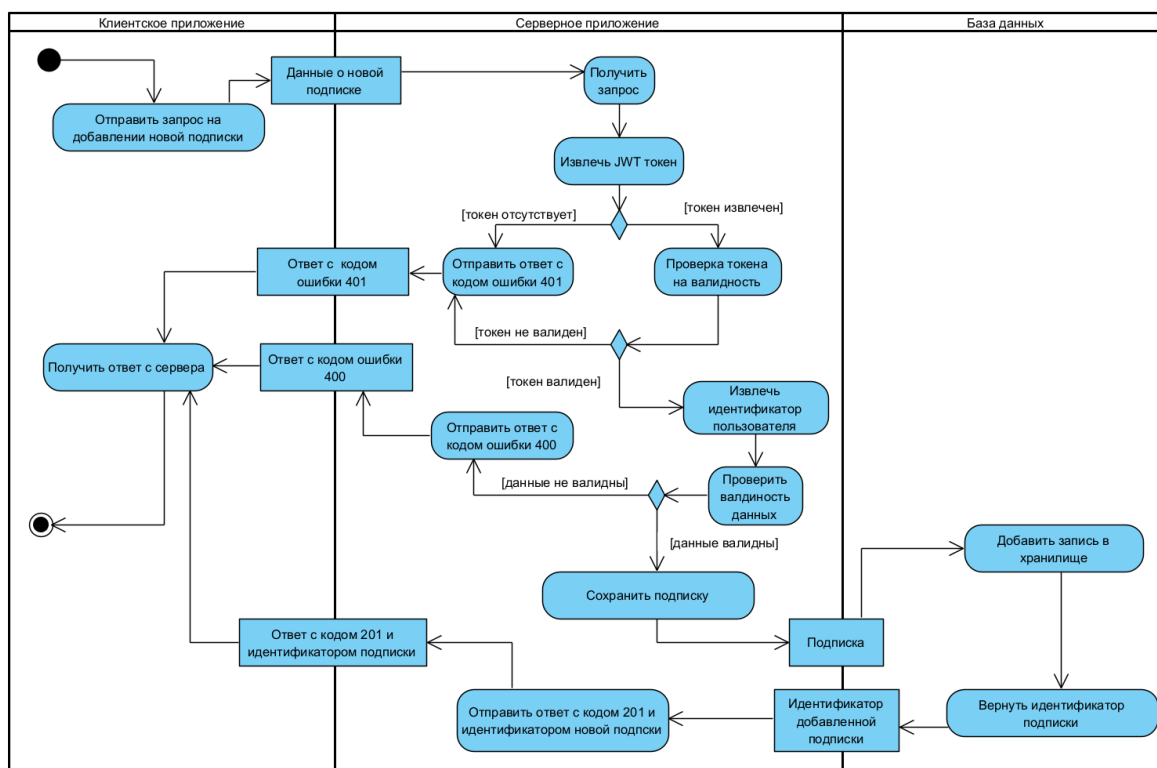


Рисунок 11 – Диаграмма деятельности серверного приложения при добавлении новой подписки

Данная диаграмма описывает процесс добавления новой подписки в информационную систему, а конкретно более подробно отображена деятельность серверного приложения.

3.3.2 Проектирование архитектуры серверного приложения

При разработке данной части информационной системы планируется придерживаться чистой архитектуре, которая состоит из нескольких принципах, помогающих создавать качественное и устойчивое программное обеспечение [9].

Основной принцип чистой архитектуры является разделение приложений на уровни с направленными зависимостями, где каждый выполняет свои задачи. Данные уровни в основном представлены в следующем варианте:

- уровень представления;
- уровень приложения;
- уровень домена;
- уровень инфраструктуры.

Уровень представления предназначен для взаимодействия с пользователем и обработкой его запросов. Уровень приложения координирует работу между уровнем представления и домен. Сам уровень домена содержит бизнес-логику, которую выполняет уровень приложения, и компоненты, отвечающие за работу с данными. Уровень инфраструктуры поддерживает структуру приложения и связь с внешними системами, например, база данных и взаимодействие с API других компонентов.

3.3.3 Механизмы безопасности

Для обеспечения безопасного доступа к функциональности серверного приложения планируется аутентификацию через JWT токены.

JSON Web Token(JWT) – открытый стандарт для создания токенов доступа, основанный на формате JSON [10]. Токены будут создаваться сервером, подписываться секретным ключом и передаваться клиенту, использующий данные токена для подтверждения подлинности аккаунта.

Дополнительно доступ к защищённым маршрутам серверного API возможен только при наличии действительного JWT-токена, передаваемого в заголовке HTTP-запроса. В случае отсутствия или недействительности токена сервер будет возвращать ошибку авторизации, предотвращая несанкционированный доступ.

Особое внимание уделяется обеспечению безопасности персональных данных пользователей. Их пароль не будет храниться в открытом виде в базе данных. Они будут преобразованы криптографическими хэш-функциями с применением соли.

Соль – строка входных данных, которая передается хеш-функции вместе с входным массивом данных для вычисления хэша [11].

В совокупности это обеспечивает невозможность восстановления оригинального пароля даже в случае утечки базы данных, что соответствует лучшим практикам в области защиты персональных данных.

3.4 Проектирование клиентского программного обеспечения

3.4.1 Назначение и функции клиентского приложения

Клиентское приложение – это приложение, которое представляет интерфейс, предоставляющий возможность пользователю взаимодействовать с информационной системой управления платными подписками. Главное назначение приложения – обеспечивать удобный, интуитивно понятный доступ ко всем возможностям системы в рамках единой веб-страницы.

Приложение клиентского уровня предоставляет пользователю следующие возможности:

- управление и просмотр текущих подписок;
- добавление новых подписок;
- отслеживание за предстоящими списаниями с помощью визуального календаря;
- анализ подписочных расходов с применением аналитических инструментов;
- изменение и настройка персональных данных профиля;
- участие в совместных подписках и управление ими.

Интерфейс нацелен на максимальное удобство для пользователя. Компоненты интерфейса обновляются динамически, без полной перезагрузки страницы, что обеспечивает быстроту и плавность. Взаимодействие с сервером происходит через уровень API. Данные передаются и обрабатываются на стороне клиента.

3.4.2 Проектирование пользовательского интерфейса

Для пользователя будут доступны следующие страницы:

- главная страница;
- страница подписок;
- страница аналитики;
- страница настройки профиля.

На всех страницах должно быть кнопка вызова меню, содержащие переходы на другие странице, с помощью которых пользователь будет перемещаться по клиентскому приложению и использовать предоставленный функционал.

На главной странице будет отображаться календарь подписок, с помощью которого пользователь может визуально видеть в какие даты будут происходить списание денежных средств в текущем месяце. При нажатии на дату будет отображаться на какую именно подписку будет производиться списание. Также планируется на этой странице предоставить общее сведение о сумме, которая будет в совокупности затрачена в этом месяце. Необходимо реализовать функционал, который позволяет при нажатии на подписку отобразить полную о ней информацию. Главная страница обязана содержать кнопку, с помощью которой пользователь может инициировать процесс добавления новой платной подписки в информационную систему.

Страница подписок отображает все подписки, которые имеет пользователь с возможностью добавить новые. Он может их фильтровать по их статусу, а также как и на главной странице имеет возможность узнать подробную информацию о подписке нажав на нее.

Страница аналитики подразумевает собой место, где пользователь информационной системы может увидеть сколько сейчас он тратит на платные подписки. Страница будет содержать круговую диаграмму, которая отображает общий финансовый объем затрачиваемый в месяц и долю каждой подписки, которая имеет активный статус, с опциональным выбором интервала и гистограмму, показывающую динамику месячных трат в диапазоне 6 месяцев с прокруткой в будущее.

Важно отметить страницу настроек профиля, с помощью которой пользователь имеет функционал изменения своих данных в информационной системе.

Также в клиентском приложении должны присутствовать модальные окна, с помощью которых можно добавлять новые подписки, редактировать личные и групповые подписки, присоединяться к подписке другого пользователя.

Важно добавить на все поля ввода информации проверку на корректность, чтобы предотвратить сбои и уязвимости в системе.

3.4.3 Проектирование структуры клиентского приложения

Клиентское приложение будет представлять собой Single Page Application приложение, работающая в браузере. Это означает, что будет единая страница, в которой будет динамически обновляться только нужные элементы. Такой подход позволяет ускорить отклик интерфейса и делает его более плавным, что повышает удобство пользования [12].

Архитектура клиентского приложения будет строиться по принципу модульности, который представляет собой структуризацию всех частей, выполняющих каждую свою функцию. Это позволяет обеспечить переиспользуемость, легкозаменяемость и масштабируемость компонентов.

Планируется внести в проектируемое приложение следующие модули:

- модуль маршрутизации;
- модуль страниц;
- общие компоненты пользовательского интерфейса;
- сервисный слой;
- слой обработки состояния и данных;
- модули валидации.

Модуль маршрутизации отвечает за отображение страниц при переходах пользователя, загружая нужные компоненты интерфейса в зависимости от адресной строки браузера.

Модуль страниц предоставляют пользователю функциональность информационной системы. Страницы взаимодействуют со сервисным слоем для получения и отправки данных на серверное приложение.

Общие компоненты пользовательского интерфейса активно используются на страницах. Этими компонентами являются модальные окна, поля ввода, кнопки, компоненты отображения подписок.

Сервисный слой обеспечивает передачу данных между клиентским приложением и серверным. Он должен обеспечивать обработку ответов и ошибок, а также управления токенами авторизации.

Слой обработки состояния данных необходим для динамического обновления компонентов, отображающие данные пользователю, которые приходят с сервера или при применении фильтрации подписок.

Модуль валидации обеспечивает контроль за вводимыми данными со стороны пользователя и предотвращает возможные уязвимости.

Клиентское приложение необходимо организовать таким образом, чтобы была возможность расширения функциональности без изменения существующих компонентов, что позволяет повысить уровень сопровождаемости, тестирования, разработки.

3.5 Обоснование выбора средств разработки и технологий

В базы данных была выбрана PostgreSQL – объектно-реляционная система управления базами данных. Она обеспечивает высокую надежность, поддержку транзакций, гибкую систему индексации и расширяемую архитектуру.

Для разработки серверного приложения был выбран язык Golang, известный своей высокой производительностью, простотой синтаксиса и встроенной поддержкой конкурентного выполнения. В качестве основы HTTP-сервера будет использоваться стандартная библиотека net/http, а маршрутизация будет реализована с помощью популярного фреймворка gorilla/mux, который облегчает управление маршрутами и параметрами запросов. Данный выбор

обоснован необходимостью создания легковесного и масштабируемого API для клиентского приложения.

Использование Telegram-бота для отправки уведомлений о предстоящих списаниях обусловлено его удобством, доступностью и высокой скоростью доставки сообщений. Telegram предоставляет простой в использовании API, который позволяет быстро интегрировать функцию отправки уведомлений в информационную систему. У большинства пользователей уже установлен мессенджер, что исключает необходимость установки дополнительного программного обеспечения. Такой подход обеспечивает надёжную обратную связь с пользователем и повышает уровень информированности, напоминая о важных событиях, связанных с подписками, в реальном времени.

Пользовательский интерфейс в клиентском приложении будет реализован с помощью библиотеки React.js, которая предоставляет компонентную архитектуру, высокую скорость рендеринга и обширные возможности по созданию SPA приложения.

React.js – JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов [13]. В связке с React.js будет использоваться React Router для маршрутизации между основными компонентами приложения и Tailwind CSS.

Tailwind CSS – CSS-фреймворк, позволяющий быстро и гибко оформлять интерфейс без необходимости написания большого количества собственных стилей.

Все основные компоненты информационной системы будут разворачиваться в изолированных Docker-контейнерах, что обеспечивает воспроизводимость среды, изоляцию зависимостей и удобство развертывания.

В качестве среды разработки планируется использовать Visual Studio Code – это кроссплатформенный редактор кода с широким набором расширений, поддержкой отладки, интеграцией с системами контроля версий и с удобством навигации по проектам. Visual Studio Code обеспечит комфортную разработку как клиентского приложения на React.js, так и серверного на Golang.

4 Разработка программного обеспечения

4.1 Физическое моделирование базы данных

В ходе разработки, была сформирована ERD-диаграмма физической модели базы данных, представлена на рисунке 12 [7].

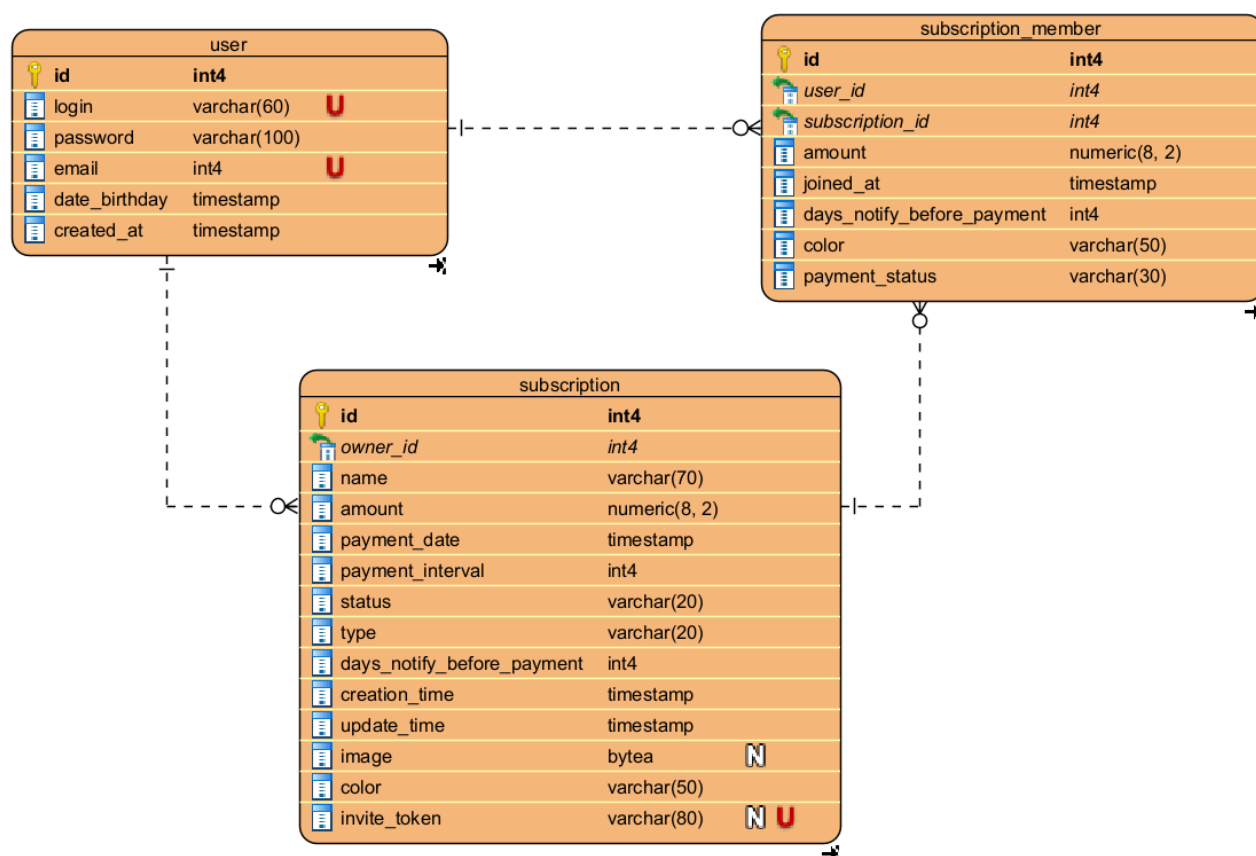


Рисунок 12 – Физическая ERD-диаграмма базы данных

Описание сущностей приведено в таблице 11.

Таблица 11 – Описание сущностей

Название сущности	Описание	Категория
Пользователь	Описывает учетной записи пользователя	Стержневая
Подписка	Включает в себя необходимую информацию	Стержневая
Участник подписки	Хранят информацию о участнике, который подключен к групповой подписке.	Характеристическая

Описание атрибутов представлено в таблице 12.

Таблица 12 – Описание атрибутов

Сущность	Атрибут	Тип данных	Размер	Роль	Обязательный	Ограничения
Пользователь	Идентификатор	Целый	5	РК	Да	1 – 100000
	Логин	Символьный	60		Да	Уникальность
	Хеш пароля	Символьный	50		Да	
	Электронная почта	Символьный	70		Да	
	Дата рождения	Дата/время			Да	
	Время создания аккаунта	Дата/время			Да	
Подписка	Идентификатор	Целый	8	РК	Да	1 – 100000000
	Создатель подписки	Целый	5	FK	Да	1 – 100000
	Имя	Символьный	20		Да	
	Стоимость	Вещественный	10		Да	
	Дата оплаты	Дата/время			Да	
	Интервал платежа	Целый	4		Да	1 – 10000
	Статус	Символьный	20		Да	
	Тип	Символьный	20		Да	
	Дни уведомления перед списанием	Целый	3		Да	1 – 1000
	Время создания	Дата/время			Да	
	Время обновления	Дата/время			Да	
	Фото	Двоичный			Нет	
	Цвет	Символьный	50			
	Код приглашения	Символьный	80		Нет	Уникальность
Участник групповой подписки	Идентификатор	Целый	4	РК	Да	1 – 10000
	Идентификатор участника	Целый	5	FK	Да	1 – 100000

Продолжение таблицы 12

Сущность	Атрибут	Тип данных	Размер	Роль	Обязательный	Ограничения
	Идентификатор подписки	Целый	8	FK	Да	1 – 100000000
	Стоимость	Вещественный	10		Да	
	Время присоединения	Дата/время	20		Да	
	Дни уведомления перед списанием	Целый	3		Да	1 – 1000
	Цвет	Символьный	50			
	Статус платежа	Символьный	30		Да	1 – 1000

Описание связей между сущностями представлено в таблице 13.

Таблица 13 – Описание связей между сущностями

Сущность	Сущность	Описание связи	Степень связи	Обязательность
Пользователь	Подписка	Пользователь имеет множество подписок. Подписка принадлежит одному пользователю.	1:M	Необязательно
Пользователь	Участник подписки	Пользователь имеет возможность множество раз быть участником разных групповых подписок.	1:M	Необязательно
Подписка	Участник подписки	Одна подписка содержит множество участников. Одни участник принадлежат одной подписке.	1:M	Необязательно

Автоматическими средствами Visual Paradigm был сгенерирован скрипт создания базы данных на языке SQL. Он представлен в листинге 1.

Листинг 1 – Скрипт создания базы данных

```
CREATE TABLE "user" (
    id SERIAL NOT NULL,
    login varchar(60) NOT NULL UNIQUE,
    password varchar(100) NOT NULL,
    email int4 NOT NULL UNIQUE,
    date_birthday timestamp NOT NULL,
    created_at timestamp NOT NULL,
    PRIMARY KEY (id));

CREATE TABLE subscription_member (
    id SERIAL NOT NULL,
    user_id int4 NOT NULL,
    subscription_id int4 NOT NULL,
    amount numeric(8, 2) NOT NULL,
    joined_at timestamp NOT NULL,
    days_notify_before_payment int4 NOT NULL,
    color varchar(50) NOT NULL,
    payment_status varchar(30) NOT NULL,
    PRIMARY KEY (id));

CREATE TABLE subscription (
    id SERIAL NOT NULL,
    owner_id int4 NOT NULL,
    name varchar(70) NOT NULL,
    amount numeric(8, 2) NOT NULL,
    payment_date timestamp NOT NULL,
    payment_interval int4 NOT NULL,
    status varchar(20) NOT NULL,
    type varchar(20) NOT NULL,
    days_notify_before_payment int4 NOT NULL,
    creation_time timestamp NOT NULL,
    update_time timestamp NOT NULL,
    image bytea,
    color varchar(50) NOT NULL,
    invite_token varchar(80) UNIQUE,
    PRIMARY KEY (id));

ALTER TABLE subscription_member ADD CONSTRAINT FKsubscription_member618585
FOREIGN KEY (user_id) REFERENCES "user" (id);
ALTER TABLE subscription ADD CONSTRAINT FKsubscription955457 FOREIGN
KEY (owner_id) REFERENCES "user" (id);
ALTER TABLE subscription_member ADD CONSTRAINT FKsubscription_member982510
FOREIGN KEY (subscription_id) REFERENCES subscription (id);
```

Скрипт также представлен в приложении А.

4.2 Разработка серверного приложения

4.2.1 Структура серверного приложения

Серверное приложение разработано в соответствии с принципами чистой архитектуры, а именно модульности и четкого разделения ответственностями между компонентами, что обеспечивает более простую поддержку, расширение и тестирование приложения.

На рисунке 13 представлена диаграмма компонентов, демонстрирующая структуру серверного приложения.

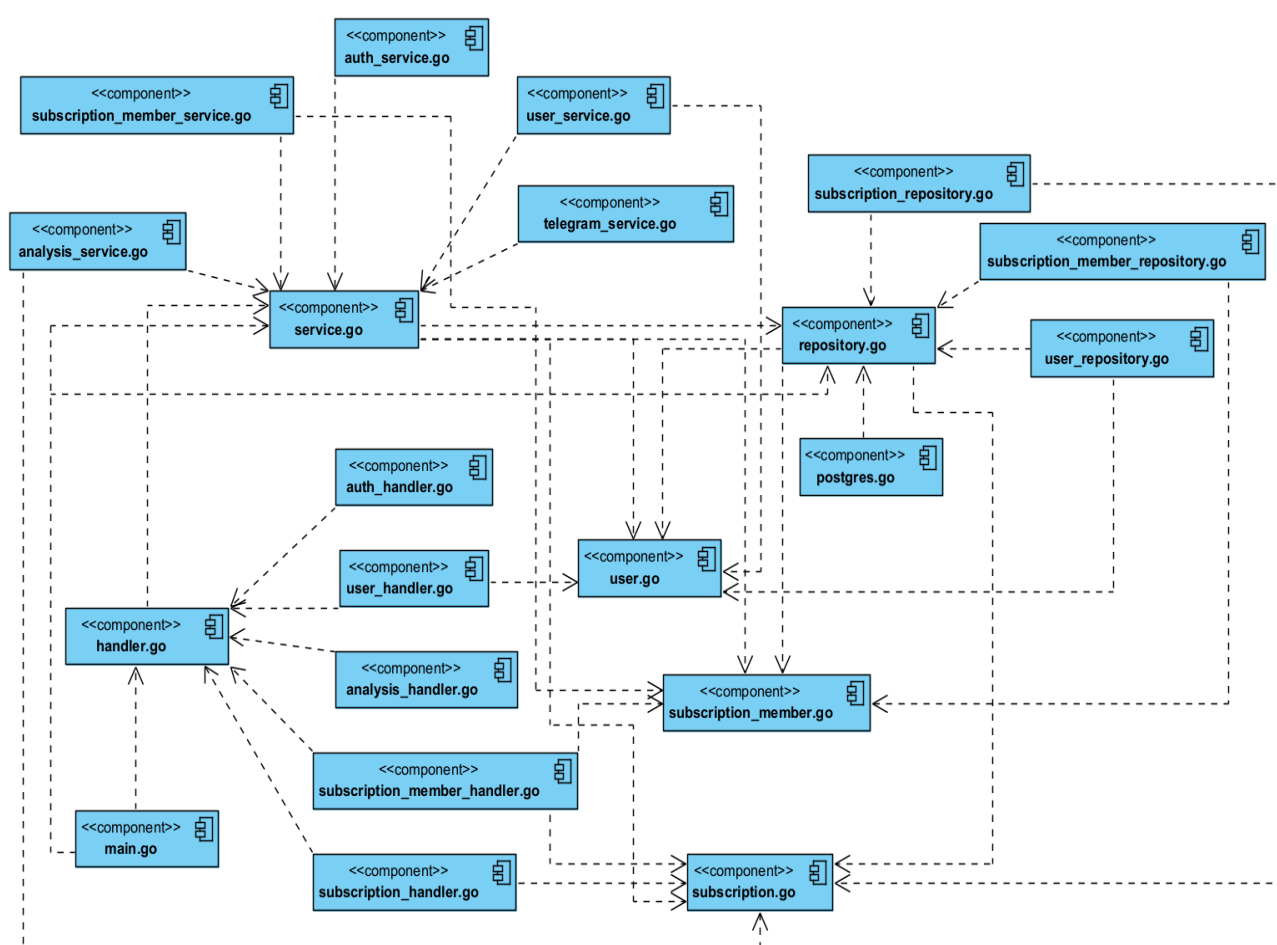


Рисунок 13 – Диаграмма компонентов серверного приложения

На диаграмме компонентов представлена структура серверного приложения и зависимости между основными файлами, сгруппированными по функциональному назначению. Компоненты отображены в виде отдельных модулей, каждый из которых отвечает за конкретный уровень архитектуры. В

таблице 14 приведено конкретное описание каждого компонента, присутствующего на диаграмме.

Таблица 14 – Описание компонентов

№	Наименование компонента	Описание
1	main.go	Точка входа в серверное приложение. Создаёт и связывает все слои и запускает HTTP-сервер
2	handler.go	Агрегирует все обработчики. Инициализируется в main.go и маршрутизирует HTTP-запросы
3	auth_handler.go	Обработчик HTTP-запросов, связанных с аутентификацией
4	user_handler.go	Обработчик HTTP-запросов, связанных с пользователями
5	analysis_handler.go	Обработчик аналитических данных и запросов
6	subscription_member_handler.go	Обработчик запросов, связанных с участниками подписки
7	subscription_handler.go	Обработчик запросов, связанных с подписками
8	service.go	Агрегатор бизнес-логики, объединяющий различные сервисы
9	auth_service.go	Сервис бизнес-логики для аутентификации пользователей
10	user_service.go	Сервис бизнес-логики для работы с пользователями
11	analysis_service.go	Сервис бизнес-логики для аналитики
12	telegram_service.go	Сервис взаимодействия с Telegram API
13	subscription_member_service.go	Бизнес-логика для управления участниками подписок
14	repository.go	Агрегатор интерфейсов доступа к данным. Не содержит реализаций

Продолжение таблицы 14

№	Наименование компонента	Описание
15	user_repository.go	Реализация интерфейса доступа к данным пользователей
16	subscription_member_repository.go	Реализация доступа к данным участников подписок
17	subscription_repository.go	Реализация доступа к данным подписок
18	postgres.go	Модуль инициализации подключения к PostgreSQL через sqlx
19	user.go	Модель пользователя. Используется во всех слоях
20	subscription_member.go	Модель участника подписки
21	subscription.go	Модель подписки

В таблице приведено описание компонентов, соответствующих архитектурной диаграмме серверной части приложения. Каждый компонент представлен в виде отдельного файла и отражает реализацию определённой функциональности в рамках многоуровневой архитектуры. Компоненты сгруппированы по слоям: точка входа, обработчики HTTP-запросов, бизнес-логика, доступ к данным, работа с базой данных и модели предметной области.

На основе описания компонентов в таблице 14, можно выделить следующие ключевые уровни серверного приложения:

- маршрутизатор;
- обработчики;
- сервисные компоненты;
- модели данных;
- репозиторий;
- компоненты авторизации и аутентификации;
- middleware.

Маршрутизатор отвечает за маршрутизацию входящих HTTP-запросов. Он инициализируется в `main.go` и использует компонент `handler.go` как точку подключения всех маршрутов. Его задача перенаправить запрос к соответствующему обработчику в зависимости от адреса и метода запроса. Обработчики принимают HTTP-запросы, извлекают из них параметры, выполняют первичную валидацию и вызывают соответствующие функции бизнес-логики. Каждый обработчик отвечает за определённую предметную область. Сервисные компоненты реализуют бизнес-логику приложения. Они обрабатывают данные, полученные от обработчиков, взаимодействуют с репозиториями и реализуют основные сценарии работы системы: управление пользователями и подписками, аналитика, уведомления и т. п. Модели данных описывают основные сущности предметной области, такие как пользователь, подписка, участник подписки. Эти структуры используются для передачи данных между слоями приложения. Репозиторий содержит интерфейсы и реализации для доступа к базе данных. Компоненты репозитория отвечают за выполнение CRUD-операций и изолируют бизнес-логику от конкретной реализации хранилища, предоставляя единый программный интерфейс. Компоненты авторизации и аутентификации обеспечивают проверку подлинности пользователей, генерацию и валидацию JWT-токенов, а также защиту маршрутов и операций, требующих доступа по ролям. Middleware – это промежуточные функции, которые выполняются до или после основных обработчиков. Они используются для логирования, централизованной обработки ошибок, проверки авторизации и других сквозных задач.

4.2.2 Взаимодействие с базой данных

Для взаимодействия серверного приложения с базой данных используется библиотека `sqlx`, представляющая собой удобное расширение над стандартной библиотекой языка Go для работы с реляционными базами данных [14]. Библиотека `sqlx` позволяет выполнять SQL-запросы с поддержкой автоматического маппинга результатов на структуры данных приложения, что

повышает удобство разработки и снижает вероятность ошибок, связанных с ручным преобразованием данных.

В серверном приложении применяется подход репозитория, который заключается в организации отдельного слоя абстракции для доступа к данным. Репозитории выступают промежуточным звеном между бизнес-логикой и базой данных, обеспечивая единообразный интерфейс и скрывая от внешних компонентов внутренние детали реализации операций с данными. Каждый репозиторий представляет собой компонент, реализующий интерфейс, содержащий необходимые методы для выполнения CRUD-операций с конкретными типами данных.

Ниже на листинге 2 приведен пример интерфейса репозитория платных подписок.

Листинг 2 – Код интерфейса репозитория платных подписок

```
type Subscription interface {
    Create(sub models.Subscription) (int, error)
    Update(sub models.Subscription) (models.Subscription, error)
    Get(subId uint) (models.Subscription, error)
    GetAllByStatus(status models.Status) ([]models.Subscription,
error)
    GetByInviteCode(inviteCode string) (models.Subscription,
error)
    GetAllByUser(userId uint) ([]models.Subscription, error)
    GetByUserAndStatus(userId uint, status models.Status)
([]models.Subscription, error)
    Delete(subId uint) error
}
```

Данный интерфейс имеет основные методы, которые позволяют доставать, изменять и удалять данные связанные с платными подписками. Его реализует структура `SubscriptionPostgres` – конкретная реализация взаимодействия с базой данных `PostgreSQL`, а конкретно с таблицей, которая хранит данные о подписках. В листинге 3 приведен код самой структуры с функцией её создания.

Листинг 3 – Код структуры `SubscriptionPostgres`

```
type SubscriptionPostgres struct {
    db *sqlx.DB
    log logger.Logger
}
```

```

}

func NewSubscriptionPostgres(db *sqlx.DB, log logger.Logger)
*SubscriptionPostgres {
    return &SubscriptionPostgres{db: db, log: log}
}

```

Выше приведенная структура содержит объект структуры `sqlx.DB`. Это обертка над `sql.DB`, которая добавляет удобные методы для работы с запросами и получения данных. С помощью данного объекта идет прямое взаимодействие с базой данных.

Далее в листинге 4 приведен код функции, демонстрирующий логику добавление платной подписки в информационную систему.

Листинг 4 – Код функции добавления платной подписки

```

func (subp *SubscriptionPostgres) Create(sub models.Subscription)
(int, error) {
    var id int
    query := fmt.Sprintf(`INSERT INTO %s (owner_id, name, amount,
payment_date, payment_interval, status, type,
days_notify_before_payment, created_at, updated_at, image,
color, invite_token)
values ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12,
$13) RETURNING id`, subscriptionsTable)

    row := subp.db.QueryRow(query, sub.OwnerId, sub.Name,
sub.Amount, sub.PaymentDate, sub.PaymentInterval, sub.Status,
sub.Type, sub.DaysNotifyBeforePayment, sub.CreatedAt,
sub.UpdatedAt, sub.Image, sub.Color, sub.InviteToken)
    if err := row.Scan(&id); err != nil {
        return 0, err
    }

    return id, nil
}

```

В коде выше отображает специфику сохранения новой платной подписки в базу данных. Необходимо сформировать SQL-запрос, в который нужно вставить нужные данные и далее с помощью объекта `sqlx.DB` его выполнить.

Чтобы получить объект `sqlx.DB` необходимо открыть соединение с работающей базой данных. Ниже приведен код в листинг 5, в котором представлена конкретная реализация подключения к базе данных PostgreSQL.

Листинг 5 – Код подключения к базе данных PostgreSQL

```

type ConfigPostgres struct {
    Host      string
    Port      string
    Username  string
    Password  string
    DBName    string
    SSLMode   string
}

func NewConfigPostgres(host string, port string, username string,
password string, DBname string, sslmode string) *ConfigPostgres {
    return &ConfigPostgres{
        Host:      host,
        Port:      port,
        Username:  username,
        Password:  password,
        DBName:    DBname,
        SSLMode:   sslmode,
    }
}

func NewPostgresDB(cfgpg *ConfigPostgres) (*sqlx.DB, error) {
    db, err := sqlx.Open("postgres", fmt.Sprintf("host=%s port=%s
user=%s dbname=%s password=%s sslmode=%s",
        cfgpg.Host, cfgpg.Port, cfgpg.Username, cfgpg.DBName,
        cfgpg.Password, cfgpg.SSLMode))
    if err != nil {
        return nil, err
    }

    err = db.Ping()
    if err != nil {
        return nil, err
    }

    return db, nil
}

```

В данном коде выполняется метод `Open`, куда передаем тип драйвера и строку подключения с необходимыми данными. Данные передаются с помощью вспомогательной структуры `ConfigPostgres`.

4.2.3 Реализация аутентификации и авторизации

Логика аутентификации и авторизации пользователей в серверном приложении было реализовано с помощью библиотеки `JWT v5` [15]. Это Golang пакет предназначенный для работы с JWT. Она упрощает создание, подпись,

верификацию и парсинг JWT-токенов, которые широко используются для аутентификации и обмена данными между сервисами. библиотека позволяет задавать как зарегистрированные claims, так и произвольные пользовательские поля.

В листинге 6 представлен код, в котором реализован логика аутентификации на основе JWT-токенов.

Листинг 6 – Код реализации сервиса аутентификации на основе JWT

```
package service

import (
    "time"

    "github.com/golang-jwt/jwt/v5"
)

type AuthJWTService struct {
    secretKey string
}

func NewAuthJWTService(sk string) *AuthJWTService {
    return &AuthJWTService{secretKey: sk}
}

func (ajs *AuthJWTService) GenerateToken(userId uint) (string,
error) {
    claims := jwt.MapClaims{
        "user_id": userId,
        "exp":      time.Now().Add(time.Hour * 24).Unix(), // Срок
        действия — 24 часа
    }

    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
    return token.SignedString( []byte(ajs.secretKey))
}

func (ajs *AuthJWTService) ValidateToken(tokenString string)
(uint, error) {
    token, err := ajs.GetJWTToken(tokenString)
    if err != nil {
        return 0, err
    }

    return ajs.GetUserIdFromToken(token)
}
```

```

func (ajs *AuthJWTService) GetJWTToken(tokenString string)
(*jwt.Token, error) {
    token, err := jwt.Parse(tokenString, func(token *jwt.Token)
(interface{}, error) {
        if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
            return nil, jwt.ErrTokenSignatureInvalid
        }
        return []byte(ajs.secretKey), nil
    })

    if err != nil || !token.Valid {
        return &jwt.Token{}, err
    }

    return token, err
}

func (ajs *AuthJWTService) GetUserIdFromToken(token *jwt.Token)
(uint, error) {
    claims, ok := token.Claims.(jwt.MapClaims)
    if !ok {
        return 0, jwt.ErrTokenInvalidClaims
    }

    userIDFloat, ok := claims["user_id"].(float64)
    if !ok {
        return 0, jwt.ErrTokenInvalidClaims
    }

    return uint(userIDFloat), nil
}

```

Структура `AuthJWTService` содержит логику работы с JWT-токенами и используется для аутентификации пользователей. Она содержит секретный ключ `secretKey`, используемый для подписи и верификации токенов. Метод `GenerateToken` создает новый JWT-токен с полем `user_id` и сроком действия 24 часа, подписанный алгоритмом HMAC SHA-256. Метод `ValidateToken` принимает строку токена, проверяет её подлинность с помощью `GetJWTToken`, а затем извлекает идентификатор пользователя из поля `user_id` через `GetUserIdFromToken`. Метод `GetJWTToken` парсит токен и проверяет его подпись, а `GetUserIdFromToken` достает значение `user_id` из payload и преобразует его в тип `uint`. Такой подход позволяет централизовать работу с аутентификацией, обеспечивая повторное использование и упрощая поддержку механизма безопасности в различных частях серверного приложения.

4.2.4 Реализация отправки уведомлений о списании денежных средств

В рамках данного модуля был реализован Telegram-бот, предназначенный для своевременного информирования пользователей о фактах списания денежных средств с их счетов. Основная задача бота оперативно уведомлять клиента о каждой финансовой операции, обеспечивая дополнительный уровень контроля над личными финансами пользователя.

Бот интегрирован с серверной частью приложения. Обработчик формирует уведомление, содержащее ключевую информацию и отправляет его в Telegram-аккаунт пользователя. Для этого используется Telegram Bot API, где сообщения отправляются с помощью HTTP-запросов к официальным методам Telegram.

Каждому пользователю предварительно присваивается уникальный Telegram ID, который хранится в базе данных и ассоциируется с его аккаунтом. Таким образом, после выполнения списания система находит Telegram ID пользователя и направляет уведомление непосредственно в его чат с ботом. Благодаря такой интеграции пользователи получают мгновенные уведомления о расходных операциях, что способствует более эффективному финансовому контролю.

Для этого используется встроенный планировщик задач на Go библиотека `cron`, которая позволяет запускать работу методов по расписанию. Планировщик настроен на ежедневный запуск в 00:00, и при каждом срабатывании он проходит по всем активным подпискам, проверяя, сколько дней осталось до следующего списания. Если установленный интервал оповещения совпадает с текущим интервалом до даты следующего платежа, система формирует сообщение с предупреждением и отправляет его пользователю в Telegram. Таким образом, пользователь получает напоминание заранее и может подготовиться к предстоящему списанию, пополнив счёт или отменив подписку. Это решение способствует повышению уровня контроля пользователя над тратами.

В листинге 7 приведен код, который отправляет уведомление пользователю о предстоящем списании средств.

Листинг 7 – Код отправки уведомления пользователю

```
func (s *Service) CheckSubscriptions() error {
    today := time.Now().UTC().Truncate(24 * time.Hour)

    subs, err := s.Subscription.GetAllByStatus(models.Active)
    if err != nil {
        return err
    }

    for _, sub := range subs {
        baseDate := sub.PaymentDate.Truncate(24 * time.Hour)
        intervalDays := int(sub.PaymentInterval)
        if intervalDays <= 0 {
            continue
        }

        var nextDue time.Time
        if baseDate.After(today) || sameYMD(baseDate, today) {
            nextDue = baseDate
        } else {
            deltaDays := int(today.Sub(baseDate).Hours() / 24)
            k := deltaDays / intervalDays
            nextDue = baseDate.AddDate(0, 0, (k+1)*intervalDays)
        }

        notifyDate := nextDue.AddDate(0, 0, -
int(sub.DaysNotifyBeforePayment))

        if sameYMD(today, notifyDate) {
            text := fmt.Sprintf(
                "🔔 <b>Напоминание</b>\n\n"+
                "По подписке «%s» запланировано списание
<b>%.2f₽</b>.\n"+
                "Дата следующего платежа: <b>%s</b>\n\n"+
                "Пожалуйста, не забудьте оплатить
вовремя!",
                sub.Name,
                sub.Amount,
                nextDue.Format("2006-01-02"),
            )

            user, err := s.User.Get(int(sub.OwnerId))
            if err != nil {
                s.log.Error(err.Error())
                continue
            }
        }
    }
}
```

```

        s.TelegramBotService.sendMessage(*user.ChatId,
text)
    }
}

return nil
}

```

Выше приведенная функция определяет дату следующего платежа по каждой подписке, вычисляет дату, когда нужно отправить уведомление с учётом заданного количества дней до списания, и, если эта дата совпадает с текущей, формирует и отправляет соответствующее сообщение через Telegram-бота.

4.2.5 Реализация интерфейса взаимодействия с клиентским приложением

Взаимодействие клиентского приложения с серверной частью осуществляется по протоколу HTTP с использованием архитектурного стиля REST. На стороне сервера реализован набор HTTP-эндпоинтов, сгруппированных по маршрутам в зависимости от предметной области, а именно пользователи, подписки, участники подписок, аналитика. представляют собой точки входа во внешнее программное API серверного приложения. С их помощью клиентская часть взаимодействует с сервером: отправляет запросы и получает ответы. Каждый эндпоинт связан с определённым маршрутом и методом и обрабатывает конкретный тип операции. Каждый маршрут обрабатывается соответствующим обработчиком, а доступ к защищённым ресурсам осуществляется через middleware-проверку токена авторизации. Ниже в таблице 15 представлены маршруты, связанные с операциями над пользователями.

Таблица 15 – Описание маршрутов, связанные с пользователем

Метод	Путь	Описание
POST	/user/register	Регистрация нового пользователя
POST	/user/login	Авторизация пользователя
GET	/user/{id}	Получение данных пользователя
PUT	/user/update/{id}	Обновление данных пользователя

Эти маршруты обеспечивают базовые операции с учетными записями пользователей. Они позволяют зарегистрироваться, авторизоваться, получить информацию по ID и изменить данные. Следующая таблица 16 содержит описание маршрутов, предназначенных для работы с подписками.

Таблица 16 – Описание маршрутов, связанных с подписками

Метод	Путь	Описание
POST	/subscriptions/add	Добавление новой подписки
GET	/subscriptions	Получение всех подписок
GET	/subscriptions/active	Получение активных подписок пользователя
GET	/subscriptions/inactive	Получение неактивных подписок
PUT	/subscriptions/update	Обновление данных подписки
DELETE	/subscriptions/delete/{id}	Удаление подписки по идентификатору

Данный набор маршрутов позволяет создавать, получать активные и неактивные, редактировать и удалять платные подписки. Доступ к маршрутам осуществляется только для авторизованных пользователей.

Далее в таблице 17 описаны маршруты управления групповыми подписками.

Таблица 17 – Описание маршрутов, связанных с участниками подписки

Метод	Путь	Описание
POST	/subscriptions/members/join	Присоединение пользователя к групповой подписке
DELETE	/subscriptions/members/exit	Выход из групповой подписки
GET	/subscriptions/members/subscription/{id}	Получение участников по идентификатору подписки
DELETE	/subscriptions/members/delete	Удаление участника из подписки
PUT	/subscriptions/members/change/amount	Изменение суммы списания с участника
PUT	/subscriptions/members/change/payment-status	Изменение статуса оплаты участника

Выше приведенные маршруты предназначены для управления групповыми подписками, а именно пользователь может присоединиться, выйти из подписки, получить всех участников или удалить конкретного и изменять сумму списания у участника или изменять статус оплаты.

В следующей таблице 18 представлены пути, необходимые для получения аналитической информации и формирования отчётов.

Таблица 18 – Описание маршрутов, связанных с аналитикой

Метод	Путь	Описание
GET	/analytics/info	Получение аналитических данных
GET	/analytics/download	Загрузка аналитического отчёта в формате PDF

Аналитический раздел предоставляет доступ к информации о подписках и действиях пользователей. Также реализована возможность выгрузки отчёта в формате PDF.

Помимо вышеописанных маршрутов, в приложении реализован специальный маршрут POST /validation используется для проверки действительности JWT-токена авторизации. Этот запрос выполняется клиентом для подтверждения своей подлинности и получения доступа к защищённым ресурсам. Сервер обрабатывает токен, извлекает из него идентификатор пользователя и возвращает информацию о валидности и сроке действия токена. Маршрут играет важную роль в поддержании безопасности приложения, особенно в сочетании с middleware-функцией авторизации.

4.3 Разработка клиентского приложения

4.3.1 Структура клиентского приложения

В соответствии с проектированием, клиентское приложение было реализовано на React.js с подходом Single Page Application. Весь интерфейс строится через index.html, в который React динамически вставляет компоненты,

что позволяет не перезагружать страницу, тем самым обеспечивая плавность интерфейса.

Структурно клиентское приложение делиться на следующие группы элементов:

- страницы;
- компоненты.

Страницы представляют собой основные разделы интерфейса, соответствующие отдельным маршрутам приложения, таким как главная страница, страница авторизации, профиль пользователя, страница подписок. Каждая страница содержит собственную логику отображения и взаимодействия с пользователем, а также объединяет необходимые компоненты интерфейса.

Компоненты являются переиспользуемыми строительными блоками пользовательского интерфейса. Они инкапсулируют визуальную часть и бизнес-логику конкретного элемента: модальные окна, защищённые маршруты, элементы подписок. Компоненты могут быть вложенными и комбинироваться между собой для формирования целостного интерфейса страниц. Это обеспечивает гибкость и повторное использование кода.

Таким образом, структура клиентского приложения формируется как иерархическая система, в которой компоненты низкого уровня служат строительными блоками для создания более сложных интерфейсных конструкций. Такой подход способствует упрощению разработки, повышению читаемости и сопровождаемости кода, а также облегчает тестирование отдельных функциональных частей пользовательского интерфейса. Кроме того, чёткое разделение на страницы и компоненты позволяет гибко управлять навигацией, повторно использовать элементы интерфейса и адаптировать приложение под различные пользовательские сценарии. Это особенно важно в условиях масштабируемости и расширения функциональности в будущем.

На рисунке 14 изображена диаграмма компонентов, которая отображает структуру клиентского приложения.

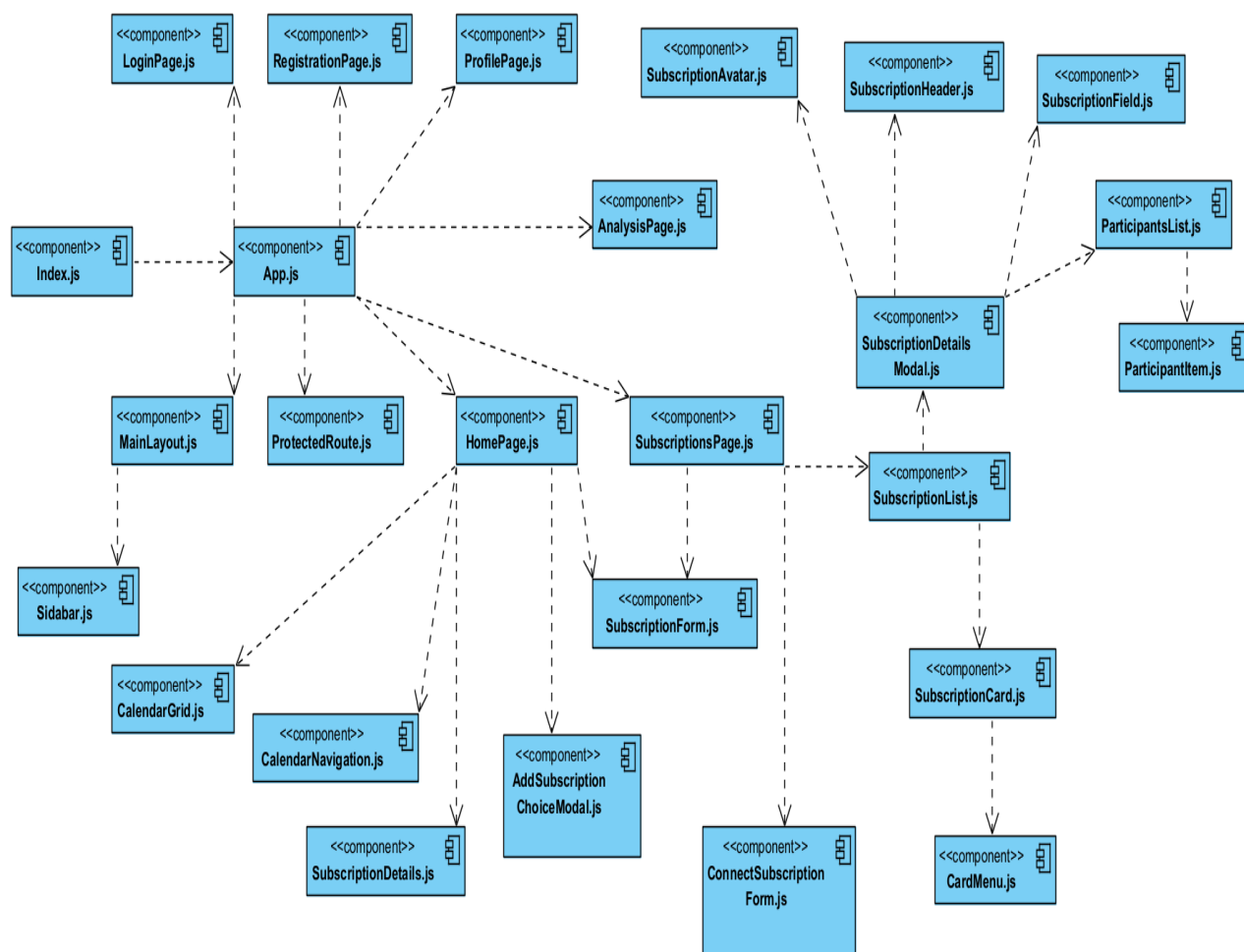


Рисунок 14 – Диаграмма компонентов клиентского приложения

Описание компонентов диаграммы представлено в таблице 19.

Таблица 19 – Описание компонентов клиентского приложения

№	Наименование компонента	Описание
1	index.js	Точка входа в приложение. Инициализирует рендеринг корневого компонента App.js
2	App.js	Основной компонент, задающий маршруты и структуру приложения
3	MainLayout.js	Макет для защищённых маршрутов. Включает навигацию, меню, боковую панель
4	ProtectedRoute.js	Компонент для ограничения доступа к маршрутам неавторизованным пользователям
5	LoginPage.js	Страница входа в приложение

Продолжение таблицы 19

№	Наименование компонента	Описание
6	RegistrationPage.js	Страница регистрации
7	HomePage.js	Главная страница, отображающая основные действия и календарь подписок
8	ProfilePage.js	Страница профиля пользователя с возможностью редактирования данных
9	AnalyticsPage.js	Страница аналитики, отображающая статистику по подпискам
10	SubscriptionsPage.js	Страница управления всеми подписками пользователя
11	Sidebar.js	Компонент бокового меню приложения
12	CalendarGrid.js	Компонент для отображения подписок в виде календаря
13	CalendarNavigation.js	Компонент навигации по календарю
14	SubscriptionDetails.js	Компонент отображения подробной информации о подписке
15	AddSubscriptionChoiceModal.js	Модальное окно выбора типа новой подписки
16	ConnectSubscriptionForm.js	Форма присоединение к новой подписке
17	SubscriptionForm.js	Форма создания или редактирования подписки
18	SubscriptionDetailsModal.js	Модальное окно с деталями подписки
19	SubscriptionList.js	Список подписок, отображаемый на странице
20	SubscriptionCard.js	Отдельная карточка подписки в списке
21	CardMenu.js	Контекстное меню управления подпиской
22	SubscriptionAvatar.js	Аватарка или иконка, ассоциированная с подпиской
23	SubscriptionHeader.js	Заголовок в модальном окне подписки, отображающий название и статус
24	SubscriptionField.js	Поля данных подписки в деталях
25	ParticipantsList.js	Список участников групповой подписки
26	ParticipantItem.js	Отдельный участник в списке участников подписки

Представленная таблица описывает ключевые компоненты клиентского приложения и их назначение в архитектуре пользовательского интерфейса. Компоненты логически разделены на маршрутизаторы, страницы и функциональные элементы интерфейса. Такая структура соответствует принципам модульного проектирования и облегчает сопровождение и масштабирование проекта. Центральным элементом является App.js, который связывает все маршруты и компоненты, обеспечивая маршрутизацию и организацию защищённых и публичных страниц. Каждая использует отдельные компоненты для отображения данных и взаимодействия с пользователем, что делает приложение гибким и легко расширяемым. Использование модальных окон, форм, карточек и списков позволяет повторно использовать компоненты в разных частях интерфейса, повышая читаемость и повторяемость кода. В совокупности, данная архитектура соответствует современным подходам к разработке SPA на React и способствует построению удобного, масштабируемого и поддерживаемого клиентского приложения.

4.3.2 Разработка пользовательского интерфейса

Пользовательский интерфейс должен быть логичен и понятен для более удобного взаимодействия пользователя с клиентским приложением.

На основе описанных требований были разработаны следующие страницы:

- страница входа;
- страница регистрации;
- главная страница;
- страница подписок;
- страница профиля;
- страница аналитики.

Каждая из страниц была разработана с учётом удобства навигации и адаптивности для различных устройств, включая мобильные и настольные платформы.

При первом заходе клиентского приложения пользователю отображается страница входа, представленная на рисунке 15.

Система управления подписками

Авторизация

Имя пользователя

Пароль

Войти

Нет аккаунта? Зарегистрируйтесь

Рисунок 15 – Страница входа в клиентское приложение

Основные элементы страницы:

- 1 – поле ввода имени пользователя;
- 2 – поле ввода пароля;
- 3 – кнопка входа в клиентское приложение;
- 4 – кнопка перехода на страницу регистрации.

Данная страница предназначена для входа пользователем в свой аккаунт информационной системы. После успешного входа пользователь автоматически перенаправляется на главную страницу, где отображается его активная информация по подпискам.

На странице предусмотрена проверка корректности введенных данных и отображение соответствующих сообщений об ошибках при неверном логине или пароле. Пользователь также может сохранить свои данные для автоматического входа при последующих запусках приложения.

Если пользователь зашел в первые и не имеет аккаунта, то при нажатии на кнопку «Зарегистрируетесь», он попадает на страницу регистрации, представленной на рисунке 16.

Система управления подписками

Регистрация

Имя пользователя

Email

Пароль

Подтверждение пароля

Зарегистрироваться

Есть аккаунта? [Выполните вход](#)

The diagram shows a registration form with the following components labeled with numbers:

- 1: Input field for Username (Имя пользователя)
- 2: Input field for Email (Email)
- 3: Input field for Password (Пароль)
- 4: Input field for Password Confirmation (Подтверждение пароля)
- 5: Registration button (Зарегистрироваться)
- 6: Login link (Выполните вход)

Рисунок 16 – Страница регистрации нового пользователя

Основные компоненты страницы:

- 1 – поле ввода имени пользователя;
- 2 – поле ввода электронный почты пользователя;
- 3 – поле ввода пароля;
- 4 – поле подтверждения пароля;
- 5 – кнопка регистрации;
- 6 – кнопка перехода на страницу входа в клиентское приложение.

Если пользователь уже зарегистрировался и попал на данную страницу, то с помощью кнопки «Выполнить вход», он будет перенаправлен на страницу входа в клиентское приложение. Перед регистрацией осуществляется проверка корректности введенных данных, включая уникальность логина и валидность адреса электронной почты. В случае успешной регистрации создается новая учетная запись и пользователь перенаправлен на главную страницу приложения, которая изображена на рисунке 17.

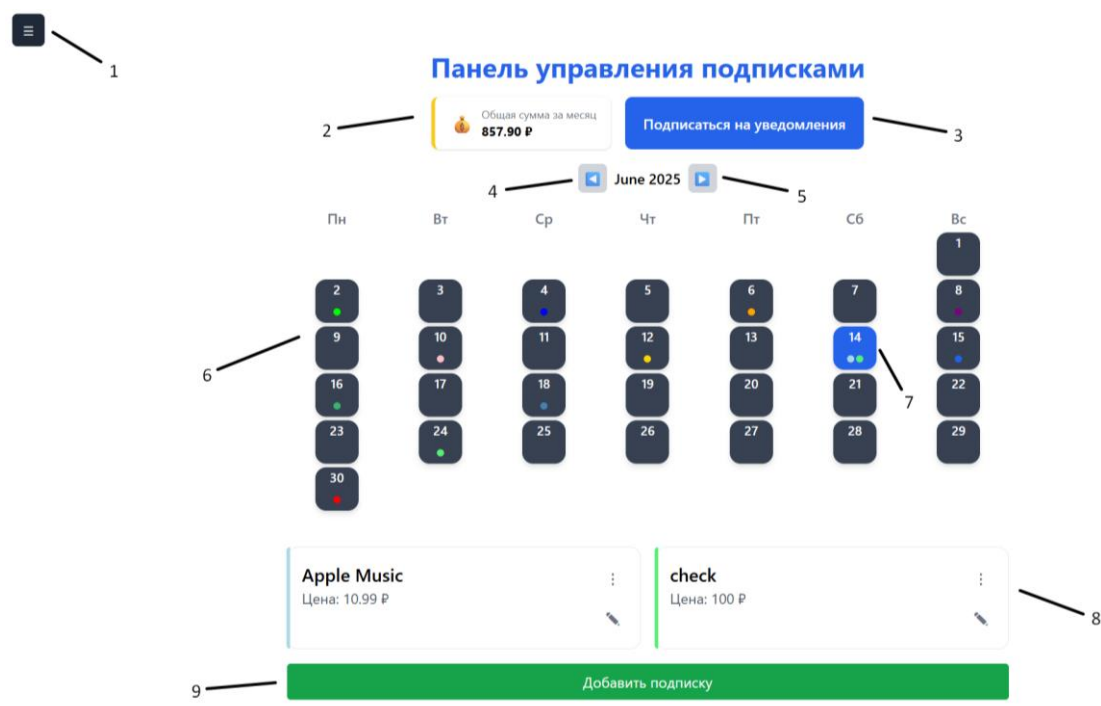


Рисунок 17 – Главная страница клиентского приложения

Данная страница содержит следующие элементы:

- 1 – кнопка вызова меню навигации;
- 2 – сведения о затраченной сумме на текущий месяц;
- 3 – кнопка, которая перекидывает пользователя на Telegram-бота информационной системы;
- 4 – кнопка перелистывания на месяц назад;
- 5 – кнопка перелистывания на месяц вперед;
- 6 – календарь подписок;
- 7 – дата в календаре подписок, на которую пользователь может нажать;
- 8 – зона отображения подписок выбранной даты;
- 9 – кнопка добавления новых подписок.

Главная страница отображает даты списания денежных средств с помощью календаря с возможностью выбора даты, чтобы узнать на какую именно подписку будет потрачено. Данная информация отображается в виде карточек подписок, в которых указывается сумма списания и дата. Календарь можно листать по

времени назад и вперед. Также имеется информативное окно, которое содержит сведения о сумме, затраченной в текущем месяце. Кнопка «Подписаться на уведомления» перекидывает пользователя на Telegram-бота, после взаимодействия с которым будут приходить уведомления в мессенджер. С данной страницы с помощью кнопки «Добавить подписку» пользователь может добавить новую подписку. Для перехода с этой страницы необходимо нажать на кнопку вызова навигационного меню, представленного на рисунке 18.

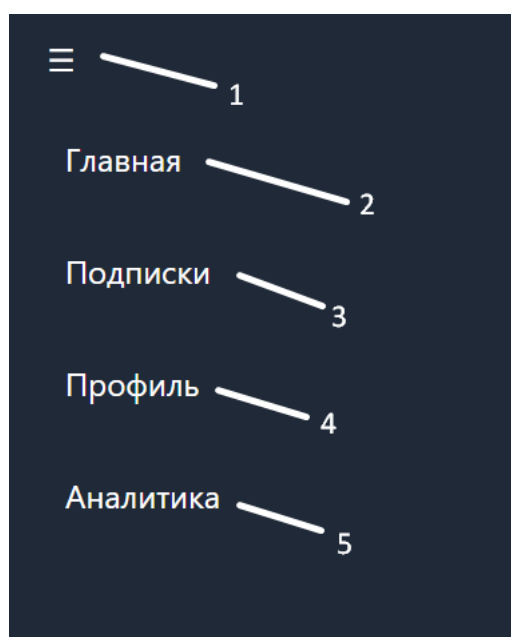


Рисунок 18 – Навигационное меню

Основные элементы навигационного меню:

- 1 – кнопка вызова или скрытия меню;
- 2 – кнопка перехода на главную страницу;
- 3 – кнопка перехода на страницу имеющихся подписок;
- 4 – кнопка перехода в профиль пользователя;
- 5 – кнопка перехода на страницу аналитики.

Главная функция данного компонента – предоставление возможности перехода между страницами клиентского приложения. Чтобы пользователю отобразились все его имеющиеся подписки, ему необходимо нажать на кнопку перехода на страницу подписок, которая изображена на рисунке 19.

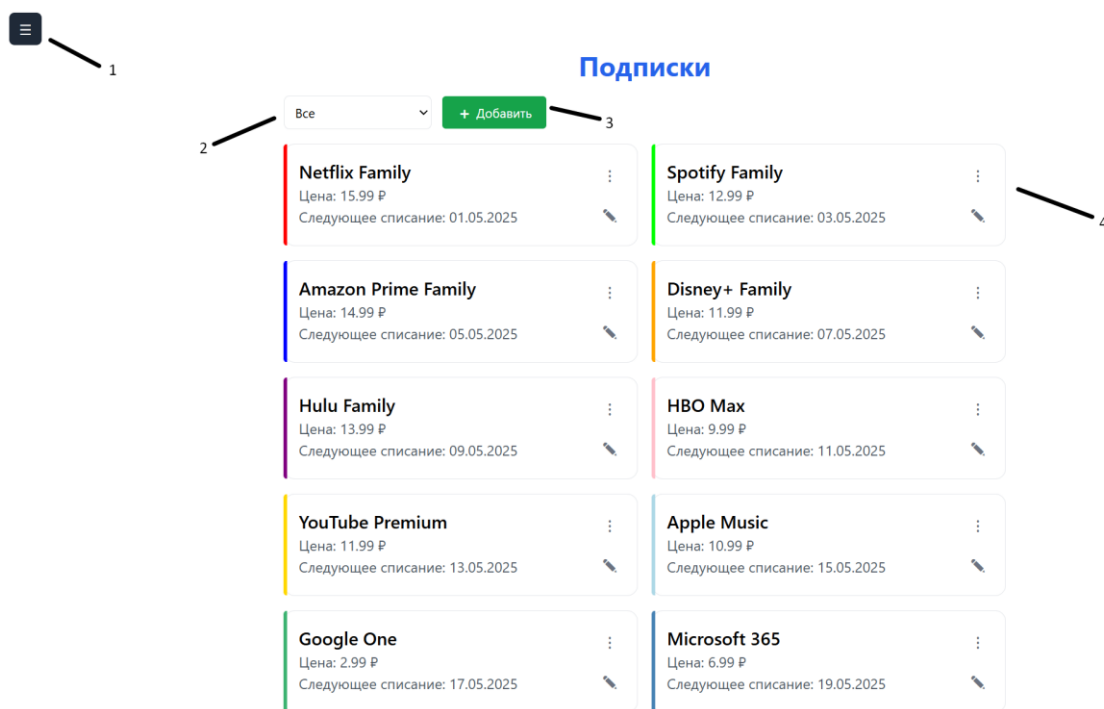


Рисунок 19 – Страница подписок

На данной странице представлены следующие компоненты:

- 1 – кнопка вызова или скрытия меню;
- 2 – выпадающий список статусов подписок;
- 3 – кнопка добавления новых подписок;
- 4 – список подписок, отфильтрованных по статусу.

С помощью этой страницы пользователь имеет возможность просмотреть все имеющиеся подписки с различным типом их фильтрации. Если необходимо добавить новую подписку в информационную систему, то нужна нажать на кнопку добавления новых подписок, которая вызывает модальное окно с вариациями добавления, которое изображено на рисунке 20.

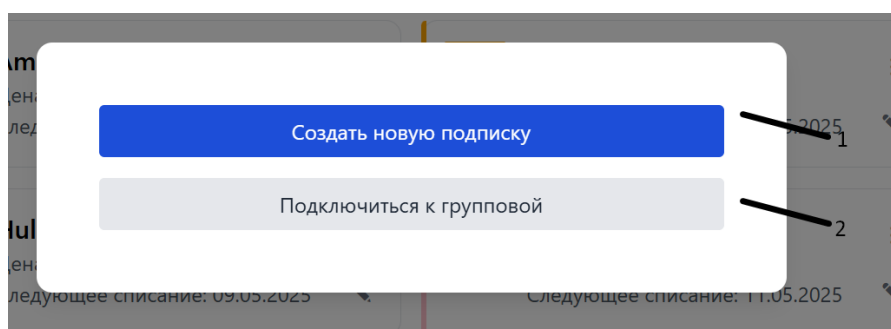


Рисунок 20 – Модальное окно с вариантами добавления подписки

Основные элементы модального окна:

- 1 – кнопка создания новой подписки;
- 2 – кнопка подключения к групповой подписки.

Данное модальное окно предоставляет возможность выбора способа добавления платной подписки, а именно либо создание новой или подключиться к групповой. Если пользователь захочет создать новую подписку, то ему отобразится форма в модальном окне, в которую он должен внести информацию, представленная на рисунке 21.

Рисунок 21 – Форма создания новой подписки

На данной форме главными компонентами являются:

- 1 – флаг, добавляющий дополнительное поле, которое относится к групповой подписки;
- 2 – поле кода приглашения к групповой подписке;
- 3 – поле ввода название подписки;

- 4 –кнопка генерации кода приглашения;
- 5 – поле ввода даты списания;
- 6 – поле ввода суммы списания;
- 7 – поле ввода интервала списания;
- 8 – поле ввода количества дней до списания, когда нужно уведомить пользователя;
- 9 – поле загрузки фото подписки;
- 10 – поле выбора цвета подписки;
- 11 – кнопка отмены создания новой подписки;
- 12 – кнопка добавления подписки.

Данная форма позволяет создать пользователю новую подписку с той информацией, которую он вносит в соответствующие поля.

Однако, если пользователь выберет подключиться к групповой подписке, то ему в модальном окне отобразится другая форма, изображенная на рисунке 22.

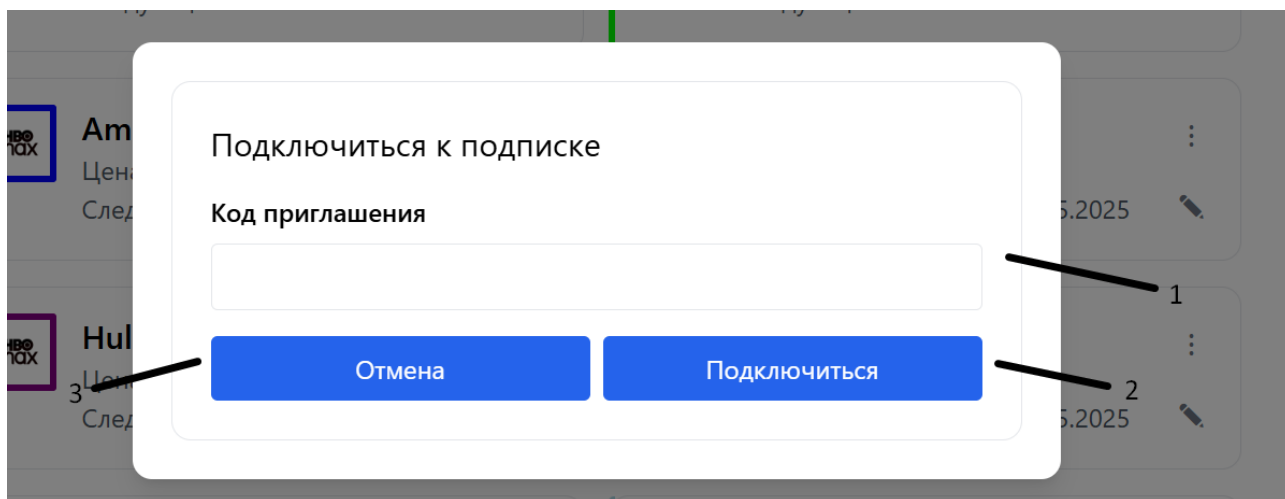


Рисунок 22 – Форма подключения к групповой подписке

На данной форме основными элементами являются:

- 1 – поле ввода кода приглашения;
- 2 – кнопка инициирующая подключение к групповой подписке;
- 3 – кнопка отмены подключения.

Текущая форма предоставляет возможность подключения пользователя к уже существующей групповой подписке.

На странице подписок и на главной существуют карточки подписок, пример который представлен на рисунке 23.

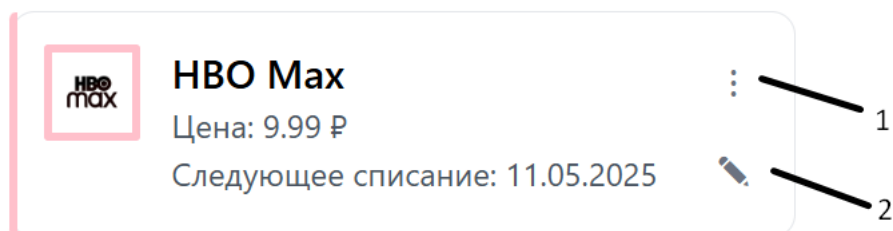


Рисунок 23 – Карточка платной подписки

Основные элементы на данной карточке следующие:

- 1 – кнопка, вызывающая выпадающий список действий;
- 2 – кнопка, вызывающая форму редактирования подписки.

Карточка отображает в себе основную информацию о подписке, а именно её цену и дату списания. Также пользователь имеет возможность нажать на нее и ему отобразится модальное окно, которое изображено на рисунке 24, с полной информацией о подписке.

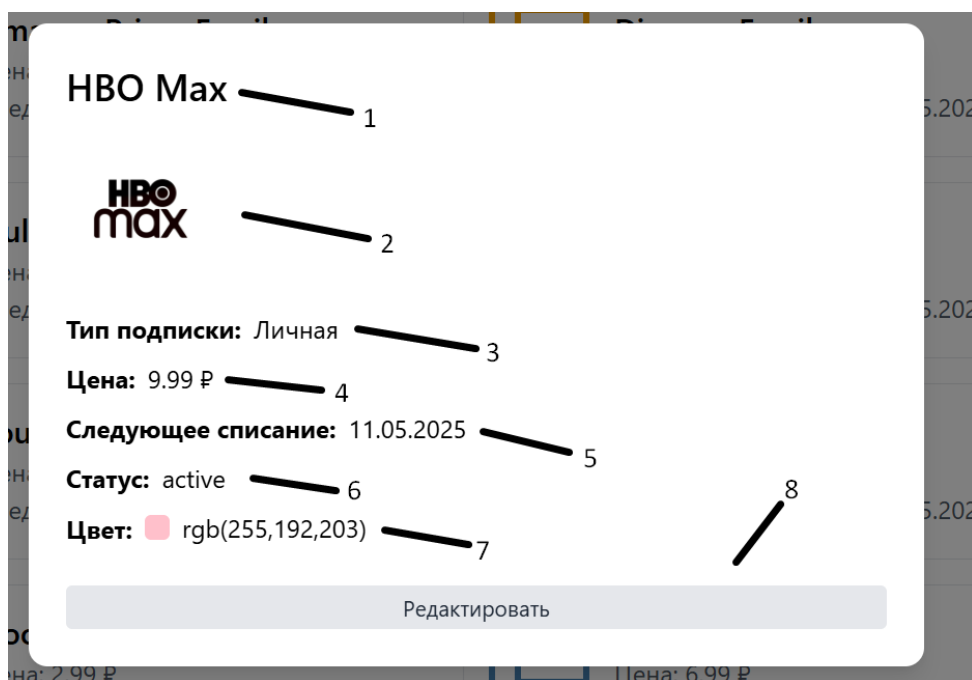


Рисунок 24 – Модальное окно с полной информацией о личной подписке

В этом модальном окне ключевыми элементами являются:

- 1 – название подписки;
- 2 – фото подписки;
- 3 – тип подписки;
- 4 – стоимость подписки;
- 5 – дата следующего списания;
- 6 – статус подписки;
- 7 – цвет подписки;
- 8 – кнопка редактирования данных подписки.

Данное модальное окно отображает всю информацию о платной подписке, которую пользователь может редактировать, нажав на кнопку «Редактировать». После этого действия, у него появиться форма, где элементы изменяться на поля ввода для новых данных. Сама форма продемонстрирована на рисунке 25.

Рисунок 25 – Форма редактирования личной подписки

Существуют следующие элементы данной формы:

- 1 – поле ввода нового названия подписки;
- 2 – поле загрузки фото;

- 3 – поле ввода цены подписки;
- 4 – поле изменения даты списания;
- 5 – поле изменения статуса подписки;
- 6 – поле изменения цвета подписки;
- 7 – кнопка сохранения изменений.

Пользователь должен ввести данные, которые он хочет внести новые данные и после нажатия на кнопку сохранения изменения сразу же вступят в силу. Если же тип подписки будет групповым, то полная информация о подписке видоизменится. На рисунке 26 продемонстрировано новые поля, которых не было в личной подписке.

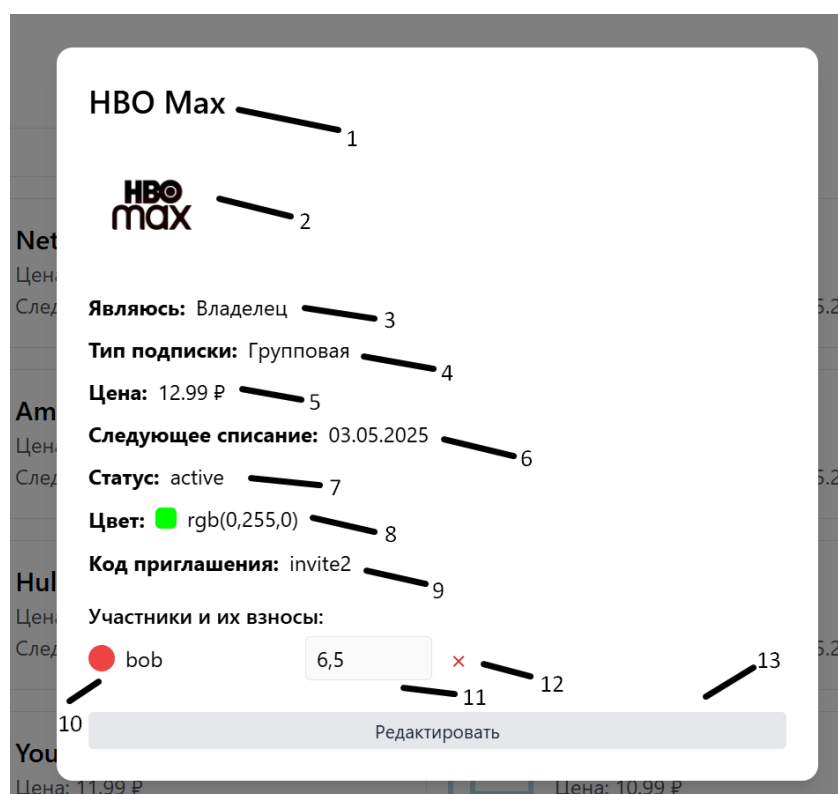


Рисунок 26 – Модальное окно с полной информацией групповой подписки

Основные элементы модального окна групповой подписки:

- 1 – название подписки;
- 2 – фото подписки;
- 3 – информация о том, кем является пользователь по отношению к подписке;

- 4 –тип подписки;
- 5 – стоимость подписки;
- 6 –дата следующего списания;
- 7 – статус подписки;
- 8 – цвет подписки;
- 9 – код приглашения;
- 10 – кнопка подтверждения или отмены подтверждения перевода финансовых средств владельцу;
- 11 – сумма участника;
- 12 – кнопка удаления участника из подписки;
- 13 – кнопка редактирования данных подписки.

В отличие от личной подписки, в групповую добавляется информация о коде приглашения, список участников, которые присоединились к подписке, обозначение роли пользователя. На рисунке 27 изображена возможность редактирования групповой подписки.

The screenshot shows a web form for editing a group subscription. At the top is the title 'HBO Max'. Below it is a file upload section with a button 'Выберите файл' and the text 'Файл не выбран'. The form includes several labeled fields: 'Являюсь: Владелец', 'Тип подписки: Групповая', 'Цена: 12,99', 'Следующее списание: 03.05.2025' (with a calendar icon), 'Статус: active', 'Цвет: [black swatch]', 'Код приглашения: invite2'. At the bottom, there is a section 'Участники и их взносы:' showing a participant 'bob' with a contribution of '6,5'. A green 'Сохранить' button is at the bottom. Numbered callouts 1 through 11 identify specific UI elements as defined in the text.

Рисунок 27 – Форма редактирования групповой подписки

Есть следующие ключевые элементы формы редактирования групповой подписки:

- 1 – поле ввода нового названия подписки;
- 2 – поле загрузки фото;
- 3 – информация о том, кем является пользователь по отношению к подписке;
- 4 – тип подписки;
- 5 – поле ввода цены подписки;
- 6 – поле изменения даты списания;
- 7 – поле изменения статуса подписки;
- 8 – поле изменения цвета подписки;
- 9 – код приглашения;
- 10 – поле изменения цены участника;
- 11 – кнопка сохранения изменений.

Также добавились новые поля, которые напрямую связаны с функционалом групповых подписок. Еще пользователь имеет возможность, изображенная на рисунке 28, изменять статусы подписки, не открывая модальное окно с полной информацией с помощью выпадающего списка с кнопками.

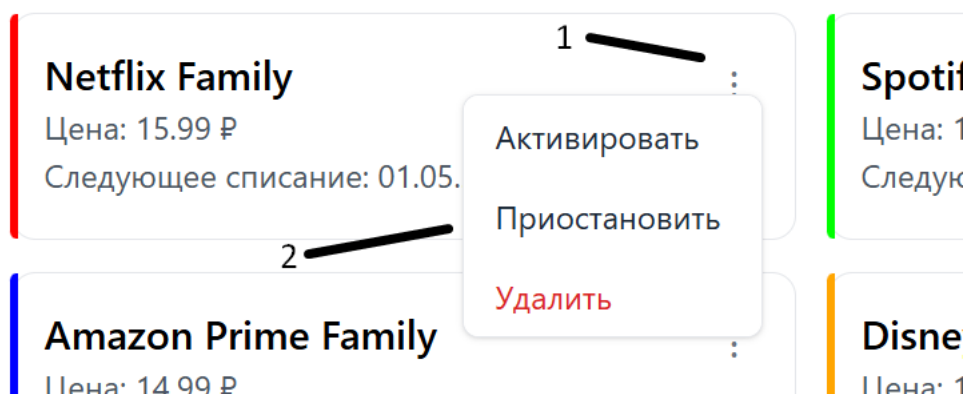


Рисунок 28 – Выпадающий список с кнопками карточки подписки

Основные элементы выпадающего списка:

- 1 – кнопка вызова выпадающего списка;

2 – содержимое выпадающего списка, состоящие из кнопок.

Наличие данного элемента в карточке обеспечивает удобное изменение статусов подписки без надобности вызывать карточку подписки и далее ей редактировать.

Если пользователь захочет изменить свои данные, а именно логин, электронную почту, дату рождения или пароль, то с помощью навигационного меню он может перейти на страницу настройки профиля, которая изображена на рисунке 29.

1

Настройки профиля

Логин

Email

Дата рождения

ДД.ММ.ГГГГ

Сохранить изменения

Сменить пароль

Новый пароль

Подтвердите пароль

Сменить пароль

2

3

4

5

6

7

8

Рисунок 29 – Страница настройки профиля пользователя

На данной странице есть следующие основные элементы:

- 1 – кнопка вызова или скрытия меню;
- 2 – поле ввода нового логина;
- 3 – поле ввода новой электронной почты;
- 4 – кнопка сохранения изменений;
- 5 – поле изменения даты рождения;
- 6 – поле ввода нового пароля;
- 7 – поле подтверждения пароля;

8 – кнопка смены пароля.

Страница настройки профиля позволяет по потребности пользователя изменять свои данные.

Для просмотра аналитики по платным подпискам, пользователю необходимо в контекстном меню нажать на кнопку «Аналитика», которая перебросит его на соответствующую страницу, которая продемонстрирована на рисунке 30.

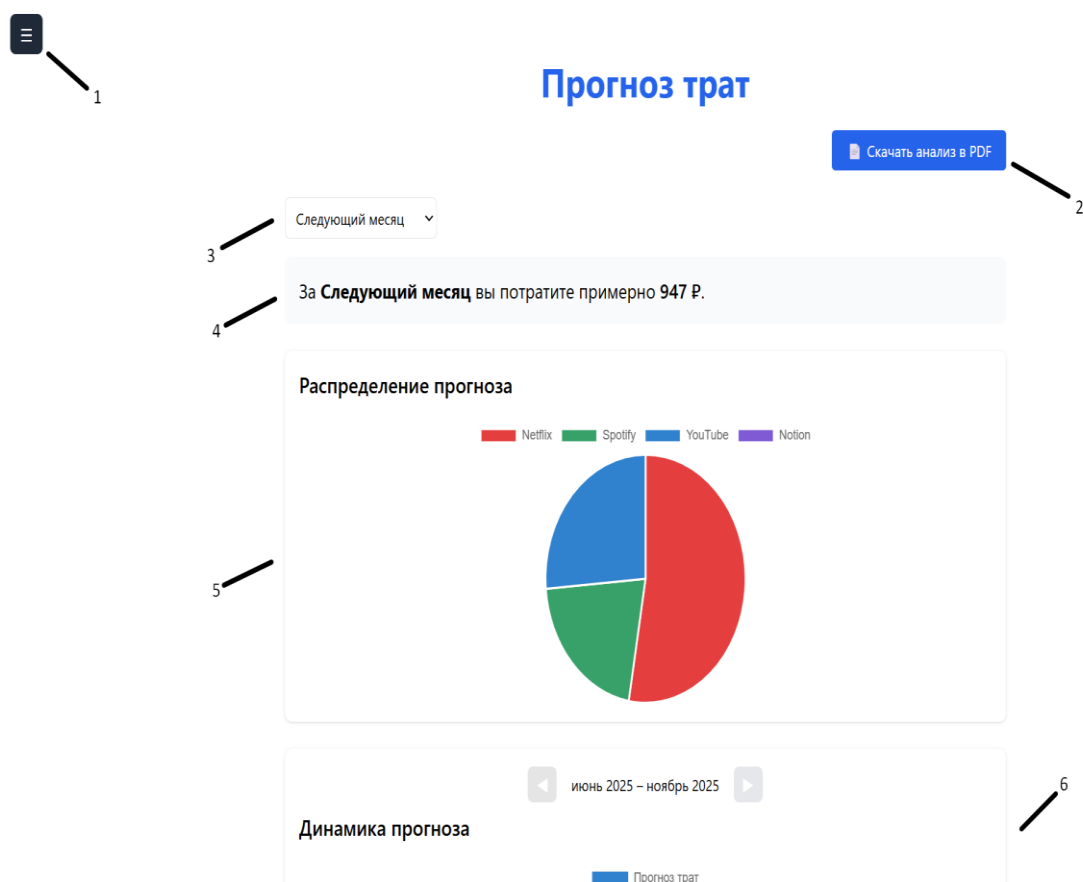


Рисунок 30 – Страница аналитики

Главные компоненты страницы аналитики:

- 1 – кнопка вызова или скрытия меню;
- 2 – кнопка, инициирующая скачивая аналитик в формате pdf;
- 3 – выпадающий список, с помощью которого можно настраивать интервалы аналитики;
- 4 – круговая диаграмма;
- 5 – гистограмма.

Данная страница показывает текущие затраты и прогноз на будущее с учетом текущих активных подписок, что дает больше финансового контроля пользователю.

4.4 Развертывание информационной системы

Развертывание информационной системы реализуется с использованием Docker и системы управления многоконтейнерными приложениями Docker Compose.

Docker и контейнеризация представляет собой платформу для разработки, доставки и запуска приложений в виде контейнеров, которые упаковывают приложение, а также все его зависимости в изолированную среду. Использование Docker позволяет облегчить воспроизводимость среды исполнения приложения, устранить проблемы с совместимостью и упростить масштабирование [16].

Информационная система имеет три независимых Docker-контейнера:

- контейнер серверного приложения;
- контейнер базы данных;
- контейнер клиентского приложения.

Все эти компоненты управляются при помощи Docker Compose. В листинге 8 представлен конфигурационный код файла docker-compose.yaml.

Листинг 8 – Конфигурационный код docker-compose.yaml

```
services:
  subscription-service:
    container_name: subscription-service
    build:
      context: ./subscription-service
      dockerfile: Dockerfile.dev
    env_file: .env
    environment:
      PORT_SRV: ${SRV_PORT}
      HOST_DB: postgres
      PORT_DB: ${DB_PORT}
      USER_DB: ${DB_USER}
      PASSWORD_DB: ${DB_PASSWORD}
      NAME_DB: ${DB_NAME}
```

```

    SSL_MODE: disable
    JWT_SECRET: ${JWT_SECRET}
volumes:
  - ./subscription-service:/app
ports:
  - "${SRV_PORT}:${SRV_PORT}"
depends_on:
  db:
    condition: service_healthy
    restart: true

db:
  image: postgres
  container_name: postgres
  restart: always
  env_file: .env
  environment:
    POSTGRES_USER: ${DB_USER}
    POSTGRES_PASSWORD: ${DB_PASSWORD}
    POSTGRES_DB: ${DB_NAME}
  ports:
    - "${DB_PORT}:${DB_PORT}"
  volumes:
    - data:/var/lib/postgresql/data
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U ${DB_USER} -d
${DB_NAME}"]
    interval: 10s
    retries: 5
    start_period: 30s
    timeout: 10s

client-app:
  container_name: client-app
  build:
    context: ./client-app
    dockerfile: Dockerfile.dev
  ports:
    - "3000:3000"
  volumes:
    - ./client-app:/app
    - /app/node_modules
  depends_on:
    - subscription-service

volumes:
  data:

```

Файл конфигурации `docker-compose.yaml` описывает развёртывание информационной системы. Контейнеры взаимодействуют между собой через указание зависимостей, обеспечивается контроль за доступностью сервисов, а

также используются Docker Volume для постоянного хранения данных. Технология Docker была выбрана, поскольку она обеспечивает изолированность компонентов, гарантирует идентичность сред разработки и продакшн, упрощает развёртывание, масштабирование и поддержку приложений, позволяет избежать проблем, связанных с различиями окружений.

5 Тестирование

5.1 Функциональное тестирование

Функциональное тестирование – проверка программного продукта или системы на соответствие его функциональным требованиям [17]. Данный вид тестирования помогает убедиться в корректности работы всех сформулированных сценариев использования информационно системы, выявить возможные ошибки и недочеты, связанные с обработкой данных, взаимодействием компонентов системы между собой, реализацией основной логики клиентского и серверного приложений, взаимодействием с пользовательским интерфейсом.

Функциональное тестирование является одним из типов тестирования методом черного ящика. Этот метод подразумевает, что при тестировании ничего не будет известно о внутреннем устройстве системы. Основное внимание уделяется входным и выходным данным.

В данной работе будет выполнено ручное функциональное тестирование. Это означает, что необходимо выполнить заранее подготовленные тест-кейсы, которые проверяют граничные и типовые условия.

Рассмотрим тестирование клиентского приложения, потому что только через него пользователь будет взаимодействовать с информационной системой. Результаты тестирования представлена в таблице 20.

Таблица 20 – Результаты функционального тестирования

Номер теста	Тестируемая ситуация	Описание теста	Ожидаемый результат	Наблюдаемый результат	Результат теста
1	Регистрация нового пользователя	Новый пользователь заходит в клиентское приложение и регистрируется	Пользователь зарегистрировался	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.1)

Продолжение таблицы 20

Номер теста	Тестируемая ситуация	Описание теста	Ожидаемый результат	Наблюдаемый результат	Результат теста
2	Вход пользователем в систему	Пользователь имеющий аккаунт выполняет вход в систему	Пользователь вошел в систему	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.2)
3	Выход пользователя из системы	Пользователь выходит из своего аккаунта в системе	Пользователь вышел из системы	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.3)
4	Добавление личной платной подписки	Пользователь авторизован и добавляет новую личную платную подписку	Пользователю добавилась личная платная подписку	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.4)
5	Добавление групповой платной подписки	Пользователь авторизован и добавляет новую групповую платную подписку	Пользователю добавилась личная платная подписку	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.4)
6	Редактирование личной платной подписки	Пользователь авторизован и изменяет все данные личной платной подписки	Личная платная подписка отредактирована	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.5)
7	Редактирование группой платной подписки	Пользователь авторизован и изменяет все данные групповой платной подписки	Групповая платная подписка отредактирована	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.5)

Продолжение таблицы 20

Номер теста	Тестируемая ситуация	Описание теста	Ожидаемый результат	Наблюдаемый результат	Результат теста
8	Вывод анализ трат на платные подписки	Пользователь переходит на страницу «Аналитика» и ему выводится анализ трат на подписки	Пользователь перешел на страницу «Аналитика», на которой отобразился анализ трат	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.6)
9	Скачивание pdf-файл с анализом трат на подписки	Пользователь на странице «Аналитика» нажимает на кнопку «Скачать анализ в PDF»	Пользователь скачал анализ трат в формате PDF	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.7)
10	Фильтрация платных подписок по статусу	Пользователь на странице «Подписки» выбирает статус подписки	Пользователю отобразились подписки с выбранным статусом	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.8)
11	Приостановка подписки	Пользователь на странице «Подписки» через контекстное меню выбирает статус «Приостановить»	У подписки изменился статус на «Приостановлена»	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.9)
12	Возобновление подписки	Пользователь на странице «Подписки» через контекстное меню выбирает статус «Возобновить»	У подписки изменился статус на «Активна»	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.10)

Окончание таблицы 20

Номер теста	Тестируемая ситуация	Описание теста	Ожидаемый результат	Наблюдаемый результат	Результат теста
13	Удаление подписки из системы	Пользователь на странице «Подписки» через контекстное меню нажимает на кнопку «Удалить»	Подписка удалена из информационной системы	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.11)
14	Подтверждение перевода денежных средств в групповой подписке	Участник заходит в групповую подписку и нажимает на кнопку «Подтвердить перевод»	Пользователь подтвердил перевод, действие зафиксировано в системе	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.12)
15	Выход из групповой подписки	Участник нажимает на кнопку «Выйти из подписки»	Подписка больше не отображается у пользователя	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.13)
16	Удаление участника групповой подписки	Владелец подписки нажимает на кнопку удаления рядом с участником	Участник больше не отображается в групповой подписке владельца	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.14)
17	Присоединение к групповой подписке	Пользователь копирует код приглашения и вставляет его в модальном окне подключения	У пользователя появилась групповая подписка	Операция выполнена. Результат соответствует ожидаемому	Тест пройден (рисунок Г.15)

В ходе функционального тестирования тесты были пройдены без ошибок.

5.2 Модульное тестирование

Модульным тестированием или юнит-тестированием называют автоматизированные тесты, которые быстро проверяют корректность работы небольшого фрагмента кода с поддержкой изоляции от другого кода [18].

Целью модульного тестирования является обеспечение стабильного роста программного проекта. Тесты, реализуемые в данном виде тестирования, позволяют удостовериться в том, что существующая функциональность работает после разработки нового функционала или рефакторинга. В данной работе были реализованы юнит-тесты сервисного слоя серверного приложения, которые проверяют работу прописанной бизнес логики. Модульное тестирование охватывает ключевые сценарии использования сервисов, включая операции с подписками, пользователями и участниками групп.

В таблице 21 представлены юнит-тесты с результатом их выполнения, в приложении Д приведен их код.

Таблица 21 – Результаты модульного тестирования

Тестируемая структура	Тест	Ожидаемый результат	Актуальный результат
SubscriptionMemberService	Должен создать нового участника подписки	Пройден	Пройден
	Должен вернуть всех участников по ID подписки	Пройден	Пройден
	Должен изменить сумму оплаты участника	Пройден	Пройден
	Должен изменить статус оплаты участника	Пройден	Пройден

Продолжение таблицы 21

Тестируемая структура	Тест	Ожидаемый результат	Актуальный результат
	Должен подтвердить участие пользователя в групповой подписке	Пройден	Пройден
	Должен удалить участника по запросу владельца	Пройден	Пройден
SubscriptionService	Должен создать новую подписку	Пройден	Пройден
	Должен обновить данные подписки	Пройден	Пройден
	Должен вернуть все подписки по статусу	Пройден	Пройден
	Должен вернуть подписку по ID	Пройден	Пройден
	Должен вернуть подписку по invite code	Пройден	Пройден
	Должен вернуть все подписки пользователя по ID	Пройден	Пройден
	Должен вернуть все подписки пользователя по ID	Пройден	Пройден

Продолжение таблицы 21

Тестируемая структура	Тест	Ожидаемый результат	Актуальный результат
	Должен удалить подписку	Пройден	Пройден
UserService	Должен создать нового пользователя	Пройден	Пройден
	Должен вернуть ошибку при некорректном создании	Пройден	Пройден
	Должен обновить данные пользователя	Пройден	Пройден
	Должен вернуть ошибку при некорректном обновлении	Пройден	Пройден
	Должен вернуть пользователя по ID	Пройден	Пройден
	Должен вернуть ошибку при некорректном Get по ID	Пройден	Пройден
	Должен вернуть пользователя по Login	Пройден	Пройден
	Должен вернуть ошибку при некорректном GetByLogin	Пройден	Пройден

Окончание таблицы 21

Тестируемая структура	Тест	Ожидаемый результат	Актуальный результат
	Должен удалить пользователя	Пройден	Пройден
	Должен вернуть ошибку при некорректном удалении	Пройден	Пройден

На рисунке 31 представлнены результаты модульного тестирования.

```

PS D:\GolangProjects\subscription-management-system\subscription-service> go test ./... -v
?      subscription-service/cmd/app      [no test files]
?      subscription-service/internal/api/handler [no test files]
?      subscription-service/internal/models [no test files]
?      subscription-service/internal/models/request [no test files]
?      subscription-service/internal/models/response [no test files]
?      subscription-service/internal/repository [no test files]
?      subscription-service/internal/utills [no test files]
?      subscription-service/pkg/json [no test files]
?      subscription-service/pkg/logger [no test files]
=== RUN   TestCountPaymentsInRange_NoPayments
--- PASS: TestCountPaymentsInRange_NoPayments (0.00s)
=== RUN   TestCountPaymentsInRange_SinglePayment
--- PASS: TestCountPaymentsInRange_SinglePayment (0.00s)
=== RUN   TestCountPaymentsInRange_MultiplePayments
--- PASS: TestCountPaymentsInRange_MultiplePayments (0.00s)
=== RUN   TestBuildPieDataRange_Calculation
--- PASS: TestBuildPieDataRange_Calculation (0.00s)
=== RUN   TestRussianMonth
--- PASS: TestRussianMonth (0.00s)
=== RUN   TestGetPeriodByInterval_NextMonth
--- PASS: TestGetPeriodByInterval_NextMonth (0.00s)
=== RUN   TestGetPeriodByInterval_Invalid
--- PASS: TestGetPeriodByInterval_Invalid (0.03s)
=== RUN   TestBuildBarDataRange_Calculation
--- PASS: TestBuildBarDataRange_Calculation (0.00s)
=== RUN   TestMaxFunction
--- PASS: TestMaxFunction (0.00s)
=== RUN   TestBuildAnalytics_Success
--- PASS: TestBuildAnalytics_Success (0.00s)
=== RUN   TestGetAnalyticsForPDF_Success
--- PASS: TestGetAnalyticsForPDF_Success (0.00s)
=== RUN   TestGenerateAnalyticsPDF_Minimal
--- PASS: TestGenerateAnalyticsPDF_Minimal (0.08s)
=== RUN   TestGenerateAndValidateToken_Success
--- PASS: TestGenerateAndValidateToken_Success (0.00s)
=== RUN   TestValidateToken_InvalidSignature
--- PASS: TestValidateToken_InvalidSignature (0.00s)
=== RUN   TestGetJWTToken_InvalidTokenString
--- PASS: TestGetJWTToken_InvalidTokenString (0.00s)
=== RUN   TestGetUserIdFromToken_InvalidClaims
--- PASS: TestGetUserIdFromToken_InvalidClaims (0.00s)
=== RUN   TestValidateToken_Expired
--- PASS: TestValidateToken_Expired (0.00s)
=== RUN   TestCreateSubscriptionMember
--- PASS: TestCreateSubscriptionMember (0.00s)
=== RUN   TestGetAllMembersInSubs
--- PASS: TestGetAllMembersInSubs (0.00s)
=== RUN   TestChangeAmountMembers
--- PASS: TestChangeAmountMembers (0.00s)
=== RUN   TestChangePaymentStatusMember
--- PASS: TestChangePaymentStatusMember (0.00s)
=== RUN   TestExistUserInGroupSubscription
--- PASS: TestExistUserInGroupSubscription (0.00s)
=== RUN   TestDeleteMemberFromSubscription_Success
--- PASS: TestDeleteMemberFromSubscription_Success (0.00s)
=== RUN   TestCreateSubscription_Success
--- PASS: TestCreateSubscription_Success (0.00s)
=== RUN   TestCreateSubscription_Error
--- PASS: TestCreateSubscription_Error (0.00s)
=== RUN   TestUpdateSubscription_Success
--- PASS: TestUpdateSubscription_Success (0.00s)
=== RUN   TestUpdateSubscription_Error
--- PASS: TestUpdateSubscription_Error (0.00s)

```

Рисунок 31 – Результаты модульного тестирования

Проведённое модульное тестирование показало корректную работу всех основных методов сервисного слоя, отвечающих за управление подписками, пользователями и участниками групп.

5.3 Метрики качества кода

5.3.1 Общие сведения о метриках качества кода

Метрики программного кода – это числовые индикаторы, используемые для измерения различных аспектов качества и характеристик программного кода. Они предоставляют информацию о структуре, сложности, поддерживаемости и других свойствах кода.

Метрики кода являются важным инструментом для разработчиков и команд разработки программного обеспечения, так как они помогают оценить и контролировать качество кода, выявлять и устранять проблемы, а также принимать обоснованные решения по оптимизации и улучшению кодовой базы. Кроме того, они могут выступать как параметры для сравнения результата переработки некоторого компонента, например, в формате «Было-стало».

5.3.2 Метрики SLOC

Количество строк кода (англ. Source Lines of Code – SLOC) – это метрика программного обеспечения, используемая для измерения его объёма с помощью подсчёта количества строк в тексте исходного кода.

Физические строки – это полный исходный код программы, включая строки комментариев [19].

Логические строки – это количество исполняемых операторов [19].

Функция приведена в листинге 9.

Листинг 9 – Метод изменения статуса оплаты у участника групповой подписки

```
func (sms *SubscriptionMemberService)
ChangePaymentStatusMember(userId uint, pscr
request.PaymentStatusChangeRequest) error {
    err := pscr.Validate()
    if err != nil {
```

```

        sms.log.Error(err.Error())
        return err
    }

    subMem, err := sms.repo.GetByUserIdAndSubId(userId, pscr.Id)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    subMem.Payment_status = pscr.PaymentStatus

    err = sms.repo.Update(subMem)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    return nil
}

```

В данной методе имеем:

- 20 физических строк кода;
- 14 логических строк кода;
- 0 строк комментариев.

Значение метрики SLOC равняется 20 физическим строкам кода и 14 логическим. Это нормальное значения для одной функции.

5.3.3 Метрика Холстеда

Метрика Холстеда относится к метрикам, вычисляемым на основании анализа числа строк и синтаксических элементов исходного кода программы [20].

Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

- $n1$ – число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций;
- $n2$ – число уникальных операндов программы;
- $N1$ – общее число операторов в программе;

– N2 – общее число операндов в программе.

Оцениваться данной метрикой будет вышеприведенную функцию, представленный в листинге 9.

Операторы приведены в таблице 22.

Таблица 22 – Операторы

Операторы	Число вхождений
:=	2
=	3
!=	3
if	3
return	4
.	10
()	7

Операнды приведены в таблице 23.

Таблица 23 – Операнды

Операнды	Число вхождений
err	8
pscr	3
subMem	3
userId	1
Id	1
PaymentStatus	1
Payment_status	1
nil	1
log	3
Error	6
repo	3
Validate	1
GetByUserIdAndSubId	1
Update	1

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 29 + 34 = 63,$$

где $N1$ – общее число операторов в программе;

$N2$ – общее число операторов в программе.

Словарь программы рассчитывается по формуле:

$$n = n1 + n2 = 6 + 14 = 20,$$

где $n1$ – число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций;

$n2$ – число уникальных операндов программы.

Объем программы рассчитывается по формуле:

$$NV = N * \log_2(n) = 63 * 4.32 = 272.2,$$

где N – длина программы;

n – словарь программы.

Сложность программы рассчитывается по формуле:

$$HD = (n1 / 2) * (N2 / n2) = (6 / 2) * (34 / 14) = 15.12,$$

где $n1$ – число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций;

$N2$ – общее число операторов в программе;

$n2$ – число уникальных операндов программы.

Метрика сложности ($HD = 15.12$) указывает на низкий уровень сложности программы, что характерно для хорошо структурированной, компактной и читаемой функции. Функция достаточно проста для сопровождения и тестирования. Количество потенциальных ошибок – крайне мало, что также подтверждает качество кода.

Заключение

В ходе выполнения выпускной квалификационной работы была разработана информационная система учёта платных подписок, предназначенная для хранения, анализа и управления платными подписками.

На начальном этапе был проведён анализ предметной области, в рамках которого изучены особенности взаимодействия пользователей с платными сервисами, классификация подписок, а также типовые задачи, связанные с управлением финансовыми обязательствами. Были определены цели разработки информационной системы, её назначение и ключевые функции, включая возможность создания индивидуальных и групповых подписок, автоматический учёт расходов, отображение аналитики и генерацию отчётности.

В рамках этапа планирования были сформулированы задачи проекта, построена диаграмма Ганта, определены ресурсы, сроки и последовательность выполнения работ. Также был выполнен анализ функциональных требований к информационной системе, включающий, описание вариантов использования и спецификации сценариев.

На этапе проектирования была определена архитектура информационной системы. В качестве архитектурного стиля была выбрана трёхуровневая клиент-серверная архитектура, обеспечивающая модульность, масштабируемость и удобство сопровождения. Также были разработаны диаграммы компонентов и развертывания, отражающие структуру и взаимодействие серверного и клиентского приложений. Проектирование осуществлялось с опорой на принципы чистой архитектуры, разделяющей бизнес-логику и инфраструктурные компоненты.

В процессе реализации серверное приложение было разработано на языке Go с использованием стандартной библиотеки `net/http`, маршрутизатора `gorilla/mux` и библиотеки `sqlx` для взаимодействия с базой данных PostgreSQL. Все обращения к базе данных реализованы через репозитории с интерфейсами,

что обеспечило модульность и удобство тестирования. Также была реализована безопасная аутентификация пользователей на основе JWT.

Клиентское приложение реализовано с использованием библиотеки React.js, а его пользовательский интерфейс построен с применением Tailwind CSS. В клиенте реализованы основные страницы: главная, подписки, аналитика и профиль, каждая из которых предоставляет доступ к соответствующему функционалу.

Развёртывание информационной системы осуществляется с помощью Docker и Docker Compose, каждый компонент запускается в отдельном контейнере.

Программное средство было протестировано с использованием функционального тестирования, в ходе которого подтверждена корректность работы основных пользовательских сценариев. Все тесты прошли успешно, система показала устойчивость и надёжность в работе.

Список источников

1. Subscription или бизнес-модель подписки: как работает и кому подходит [Электронный ресурс]. URL: <https://platrum.ru/blog/subscription-ili-biznes-model-podpiski> (дата обращения: 10.04.2025).
2. Е. Б. Никитина. Функционально-стоимостный анализ : учебное пособие; Пермский государственный национальный исследовательский университет [Электронный ресурс]. URL: <http://www.psu.ru/files/docs/science/books/uchebnie-posobiya/nikitina-funkcionalno-stoimostnyj-analiz.pdf> (дата обращения 15.04.2025).
3. Джим Арлоу. UML2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование/Джим Арлоу, Айла Нейштадт. – Санкт-Петербург, Издательство Символ-Плюс, 2007. – 624.
4. Массе, М. REST API Design Rulebook / М. Массе. – Sebepostal: O'Reilly Media, 2011. – 114 с.
5. Молдовян Н. А., Молдовян А. А., Еремеев М. А. Криптография: от примитивов к синтезу алгоритмов. – СПб.: БХВ Петербург, 2004. – 448 с.
6. Ричардс М., Форд Н. Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы. – СПб.: Питер, 2023. – 448 с.
7. Data Modeling: Conceptual vs Logical vs Physical Data Model [Электронный ресурс]. URL: <https://online.visual-paradigm.com/knowledge/visual-modeling/conceptual-vs-logical-vs-physical-data-model> (дата обращения 30.04.2025).
8. Telegram Bot API Getted Started – React [Электронный ресурс]. URL: <https://core.telegram.org/bots/api> (дата обращения 16.05.2025).
9. Clean Architecture [Электронный ресурс]. URL: <https://habr.com/ru/companies/otus/articles/732178/> (дата обращения 01.05.2025).
10. Подробнее о JWT / Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/842056/> (дата обращения 01.05.2025).

11. Соль | Энциклопедия «Касперского» [Электронный ресурс]. URL: <https://encyclopedia.kaspersky.ru/glossary/salt/?ysclid=mbnmrjb4xf890591596> (дата обращения 02.05.2025).
12. Кроутер, Р. Single Page Web Applications: JavaScript end-to-end / Р. Кроутер, А. Мюррей, К. Поуп, Н. Фрэйзер. – М.: ДМК Пресс, 2015. – 300 с.
13. Getted Started – React [Электронный ресурс]. URL: <https://react.dev/learn?url=https%3A%2F%2Freactjs.org%2Fdocs%2Fgetting-started.html> (дата обращения 05.05.2025).
14. Illustrated guide to SQLX [Электронный ресурс]. URL: <https://jmoiron.github.io/sqlx/> (дата обращения 10.05.2025).
15. jwt package – github.com/golang-jwt/jwt/v5 – Go Packages [Электронный ресурс]. URL: <https://pkg.go.dev/github.com/golang-jwt/jwt/v5> (дата обращения 15.05.2025)
16. Гош С. Docker без секретов: Пер. с англ. – СПб.: БХВ-Петербург, 2023. – 224 с.
17. Куликов, С. С. К90 Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 3-е изд. – Минск: Четыре четверти, 2020. – 312 с.
18. Хориков Владимир Х79 Принципы юнит-тестирования. – СПб.: Питер, 2021. – 320 с.
19. Analysis Of Source Lines Of Code(SLOC) Metric [Электронный ресурс]. URL: https://www.academia.edu/6765433/Analysis_Of_Source_Lines_Of_Code_SLOC_Metric#loswp-work-container (дата обращения 25.05.2025).
20. An Analysis of the Design and Definitions of Halstead's Metrics [Электронный ресурс]. URL: https://www.researchgate.net/publication/260843757_An_Analysis_of_the_Design_and_Definitions_of_Halstead's_Metrics (дата обращения 25.05.2025).

Приложение А
(обязательное)

Код скрипта создания базы данных

Листинг А.1

```

CREATE TABLE "user" (
    id          SERIAL NOT NULL,
    login       varchar(60) NOT NULL UNIQUE,
    password    varchar(100) NOT NULL,
    email       int4 NOT NULL UNIQUE,
    date_birthday timestamp NOT NULL,
    created_at  timestamp NOT NULL,
    PRIMARY KEY (id));

CREATE TABLE subscription_member (
    id          SERIAL NOT NULL,
    user_id     int4 NOT NULL,
    subscription_id int4 NOT NULL,
    amount      numeric(8, 2) NOT NULL,
    joined_at   timestamp NOT NULL,
    days_notify_before_payment int4 NOT NULL,
    color       varchar(50) NOT NULL,
    payment_status varchar(30) NOT NULL,
    PRIMARY KEY (id));

CREATE TABLE subscription (
    id          SERIAL NOT NULL,
    owner_id    int4 NOT NULL,
    name        varchar(70) NOT NULL,
    amount      numeric(8, 2) NOT NULL,
    payment_date timestamp NOT NULL,
    payment_interval int4 NOT NULL,
    status      varchar(20) NOT NULL,
    type        varchar(20) NOT NULL,
    days_notify_before_payment int4 NOT NULL,
    creation_time timestamp NOT NULL,
    update_time timestamp NOT NULL,
    image       bytea,
    color       varchar(50) NOT NULL,
    invite_token varchar(80) UNIQUE,
    PRIMARY KEY (id));

ALTER TABLE subscription_member ADD CONSTRAINT FKsubscription_member_user FOREIGN KEY (user_id) REFERENCES "user" (id);
ALTER TABLE subscription ADD CONSTRAINT FKsubscription_owner FOREIGN KEY (owner_id) REFERENCES "user" (id);
ALTER TABLE subscription_member ADD CONSTRAINT FKsubscription_member_subscription FOREIGN KEY (subscription_id) REFERENCES subscription (id);

```

Приложение Б
(обязательное)

Код серверного приложения

Листинг Б.1 – Код файл main.go

```

package main

import (
    "fmt"
    "net/http"
    "os"
    "time"

    "subscription-service/internal/api/handler"
    "subscription-service/internal/repository"
    service "subscription-service/internal/services"
    "subscription-service/internal/utils"
    "subscription-service/pkg/logger"
    "github.com/robfig/cron/v3"
    "github.com/rs/cors"
)

func main() {
    logFile, err := utils.SetupLogsFiles()
    if err != nil {
        panic(err)
    }

    slogLogger := utils.CreateSlogLogger(logFile)

    slog := logger.NewSlogAdapter(slogLogger)

    config := repository.NewConfigPostgres(
        os.Getenv("HOST_DB"),
        os.Getenv("PORT_DB"),
        os.Getenv("USER_DB"),
        os.Getenv("PASSWORD_DB"),
        os.Getenv("NAME_DB"),
        os.Getenv("SSL_MODE"),
    )

    db, err := repository.NewPostgresDB(config)
    if err != nil {
        slog.Error(err.Error())
    }

    secretKey := os.Getenv("JWT_SECRET")

    repos := repository.NewRepository(db, slog)

    tgService, err := service.NewTelegramBotService(repos, slog)
    if err != nil {
        slog.Error("Не удалось создать TelegramBotService")
    }

    go tgService.Start()
    slog.Info("TelegramBotService запущен")
}

```

```

service := service.NewService(repos, slog, secretKey)
handler := handler.NewHandler(service, slog)

routes := handler.InitRoutes()

c := cron.New() // без WithSeconds используется формат "с
минуты и выше"
// "0 0 * * *" = «каждый день в 00:00» по умолчанию в UTC
@every 1m
_, err = c.AddFunc("0 0 * * *", func() {
    if err := service.CheckSubscriptions(); err != nil {
        slog.Error("CheckSubscriptions: " + err.Error())
    }
})
if err != nil {
    slog.Error("failed to register daily cron: " +
err.Error())
} else {
    c.Start()
    slog.Info("Daily subscription-check cron started (00:00
UTC)")
}

routeWithCors := cors.New(cors.Options{
    AllowedOrigins: []string{"http://localhost:3000"},
    AllowedMethods: []string{"GET", "POST", "PUT", "DELETE",
"OPTIONS"},
    AllowedHeaders: []string{"Authorization", "Content-
Type"},
    AllowCredentials: true, // если используешь куки/сессии
}).Handler(routes)

slog.Info(fmt.Sprintf("Listening on %s",
os.Getenv("PORT_SRV")))

srv := &http.Server{
    Addr:           "0.0.0.0:" + os.Getenv("PORT_SRV"),
    Handler:        routeWithCors,
    ReadTimeout:    10 * time.Second,
    WriteTimeout:   10 * time.Second,
    MaxHeaderBytes: 1 << 20,
}

slog.Error(srv.ListenAndServe().Error())
}

```

Листинг Б.2 – Код файла handler.go

```

package handler

import (
    service "subscription-service/internal/services"

```

```

    "subscription-service/pkg/logger"
    "github.com/gorilla/mux"
)

type Handler struct {
    services *service.Service
    log      logger.Logger
}

func NewHandler(service *service.Service, log logger.Logger)
*Handler {
    return &Handler{services: service, log: log}
}

func (h *Handler) InitRoutes() *mux.Router {
    r := mux.NewRouter()

    r.HandleFunc("/validation",
h.AuthMiddleware(h.Validate)).Methods("POST")

    users := r.PathPrefix("/user").Subrouter()

    users.HandleFunc("/register", h.Register).Methods("POST")
    users.HandleFunc("/login", h.Login).Methods("POST")
    users.HandleFunc("/{id:[0-9]+}", h.Get).Methods("GET")
    users.HandleFunc("/update/{id:[0-9]+}",
h.Update).Methods("PUT")

    subscriptions := r.PathPrefix("/subscriptions").Subrouter()
    subscriptions.HandleFunc("/add",
h.AuthMiddleware(h.AddSubscription)).Methods("POST")
    subscriptions.HandleFunc("",
h.AuthMiddleware(h.GetAllSubscription)).Methods("GET")
    subscriptions.HandleFunc("/active",
h.AuthMiddleware(h.GetActiveSubscription)).Methods("GET")
    subscriptions.HandleFunc("/inactive",
h.AuthMiddleware(h.GetInactiveSubscription)).Methods("GET")
    subscriptions.HandleFunc("/update",
h.AuthMiddleware(h.UpdateSubscription)).Methods("PUT")
    subscriptions.HandleFunc("/delete/{id:[0-9]+}",
h.AuthMiddleware(h.Delete)).Methods("DELETE")

    subscriptionsMembers :=
r.PathPrefix("/subscriptions/members").Subrouter()
    subscriptionsMembers.HandleFunc("/join",
h.AuthMiddleware(h.JoinToGroupSubscription)).Methods("POST")
    subscriptionsMembers.HandleFunc("/exit",
h.AuthMiddleware(h.ExitFromGroupSubscription)).Methods("DELETE")
    subscriptionsMembers.HandleFunc("/subscription/{id:[0-9]+}",
h.AuthMiddleware(h.GetAllMembersInSubs)).Methods("GET")

```

```

        subscriptionsMembers.HandleFunc("/delete",
h.AuthMiddleware(h.DeleteMemberFromSubscription)).Methods("DELETE"
)

        updateMembers :=
subscriptionsMembers.PathPrefix("/change").Subrouter()
        updateMembers.HandleFunc("/amount",
h.AuthMiddleware(h.ChangeAmountMembers)).Methods("PUT")
        updateMembers.HandleFunc("/payment-status",
h.AuthMiddleware(h.ChangePaymentStatusUser)).Methods("PUT")

        analysis := r.PathPrefix("/analytics").Subrouter()
        analysis.HandleFunc("/info",
h.AuthMiddleware(h.GetAnalysisData)).Methods("GET")
        analysis.HandleFunc("/download",
h.AuthMiddleware(h.DownloadAnalysisInPDF)).Methods("GET")

        return r
}

```

Листинг Б.3 – Код файла subscription_handler.go

```

package handler

import (
    "fmt"
    "net/http"

    "subscription-service/internal/models"
    JSON "subscription-service/pkg/json"
)

func (h *Handler) AddSubscription(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на создание подписки")

    subscription := models.Subscription{}

    err := JSON.Parse(r, &subscription)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
        return
    }

    err = subscription.Validate()
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Invalid JSON")
        return
    }
}

```

```

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    userId, _ := h.services.Auth.GetUserIdFromToken(token)

    subscription.OwnerId = userId

    subId, err := h.services.Subscription.Create(subscription)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusInternalServerError,
err.Error(), "Failed to create a subscription")
        return
    }

    h.log.Info(fmt.Sprintf("Подписка с id: %d успешно добавлена",
subId))

    response := map[string]int{
        "new_sub_id": subId,
    }

    JSON.Response(w, http.StatusOK, response)
}

func (h *Handler) GetAllSubscription(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на получение все подписок пользователя по
id")

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    userId, _ := h.services.Auth.GetUserIdFromToken(token)

    subs, err := h.services.Subscription.GetAllByUserId(userId)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusInternalServerError,
err.Error(), "Failed to get all subscriptions")
        return
    }

    JSON.Response(w, http.StatusOK, subs)
}

func (h *Handler) GetActiveSubscription(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на получение все активных подписок
пользователя по id")

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)

```

```

        userId, _ := h.services.Auth.GetUserIdFromToken(token)

        subs, err :=
h.services.Subscription.GetByUserAndStatus(userId, models.Active)
        if err != nil {
            h.log.Error(err.Error())
            JSON.ErrorResponse(w, http.StatusInternalServerError,
err.Error(), "Failed to get active subscriptions")
            return
        }

        JSON.Response(w, http.StatusOK, subs)
    }

func (h *Handler) GetInactiveSubscription(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на получение все приостановленных подписок
пользователя по id")

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    userId, _ := h.services.Auth.GetUserIdFromToken(token)

    subs, err :=
h.services.Subscription.GetByUserAndStatus(userId,
models.InActive)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusInternalServerError,
err.Error(), "Failed to get inactive subscriptions")
        return
    }

    if len(subs) == 0 {

        JSON.Response(w, http.StatusOK, map[string]string{
            "response": "not have inactive subscription for this
user",
        })
        return
    }

    JSON.Response(w, http.StatusOK, subs)
}

func (h *Handler) UpdateSubscription(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на изменения подписки")

    subscription := models.Subscription{}

    err := JSON.Parse(r, &subscription)

```

```

        if err != nil {
            h.log.Error(err.Error())
            JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
            return
        }

        err = subscription.Validate()
        if err != nil {
            h.log.Error(err.Error())
            JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Invalid JSON")
            return
        }

        tokenString, _ := h.GetJWTTokenFromRequest(r)
        token, _ := h.services.Auth.GetJWTToken(tokenString)
        userId, _ := h.services.Auth.GetUserIdFromToken(token)

        subscription.OwnerId = userId

        fmt.Println(subscription)

        sub, err := h.services.Subscription.Update(subscription)
        if err != nil {
            h.log.Error(err.Error())
            JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Invalid JSON")
            return
        }

        JSON.Response(w, http.StatusOK, sub)
    }

    func (h *Handler) DeleteSubscription(w http.ResponseWriter, r
*http.Request) {

    }

```

Листинг Б.4 – Код файла subscription_member_handler.go

```

package handler

import (
    "net/http"
    "strconv"

    "subscription-service/internal/models"
    request "subscription-service/internal/models/request"
    JSON "subscription-service/pkg/json"
    "github.com/gorilla/mux"
)

```

```

// Handler'ы учатсника
func (h *Handler) JoinToGroupSubscription(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на присоединение к групповой подписки")

    inviteCode := request.InviteCode{}

    err := JSON.Parse(r, &inviteCode)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
        return
    }

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    userId, _ := h.services.Auth.GetUserIdFromToken(token)

    err =
h.services.SubscriptionMember.JoinToGroupSubscription(userId,
inviteCode)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed join to group subscription")
        return
    }

    JSON.Response(w, http.StatusOK, map[string]string{
        "response": "user succeseful join to group subscription",
    })
}

func (h *Handler) ExitFromGroupSubscription(w http.ResponseWriter,
r *http.Request) {
    h.log.Debug("Запрос на выход из групповой подписки")

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    userId, _ := h.services.Auth.GetUserIdFromToken(token)

    subId := models.SubscriptionId{}

    err := JSON.Parse(r, &subId)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
        return
    }
}

```

```

    err = h.services.ExitFromGroupSubscription(userId, subId)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed exit from group subscription")
        return
    }

    JSON.Response(w, http.StatusOK, map[string]string{
        "response": "succeseful exit from group subscription",
    })
}

func (h *Handler) ChangePaymentStatusUser(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на изменение статуса оплаты групповой
подписки")

    pscr := request.PaymentStatusChangeRequest{}

    err := JSON.Parse(r, &pscr)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
        return
    }

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    userId, _ := h.services.Auth.GetUserIdFromToken(token)

    err = h.services.ChangePaymentStatusMember(userId, pscr)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed change payment status")
        return
    }

    JSON.Response(w, http.StatusOK, map[string]string{
        "response": "succeseful change payment status",
    })
}

// Handler владельца
func (h *Handler) GetAllMembersInSubs(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на получение участников групповой
подписки")

    vars := mux.Vars(r)

```

```

    idStr := vars["id"]

    // Преобразовать в int, если нужно
    id, err := strconv.Atoi(idStr)
    if err != nil {
        http.Error(w, "Invalid ID", http.StatusBadRequest)
        return
    }

    members, err :=
h.services.SubscriptionMember.GetAllMembersInSubs(uint(id))
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
        return
    }

    JSON.Response(w, http.StatusOK, members)
}

func (h *Handler) ChangeAmountMembers(w http.ResponseWriter, r
*http.Request) {
    // id user bp payload
    h.log.Debug("Запрос на изменение цену участника")

    am := request.AmountMember{}

    err := JSON.Parse(r, &am)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
        return
    }

    err = h.services.ChangeAmountMembers(am)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed change amount")
        return
    }

    JSON.Response(w, http.StatusOK, map[string]string{
        "response": "succeseful change amount",
    })
}

func (h *Handler) DeleteMemberFromSubscription(w
http.ResponseWriter, r *http.Request) {

```

```

    h.log.Debug("Запрос на удаление участника из групповой
подписки")

    dm := request.DeleteMember{}

    err := JSON.Parse(r, &dm)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
        return
    }

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    ownerId, _ := h.services.Auth.GetUserIdFromToken(token)

    dm.OwnerId = ownerId

    err = h.services.DeleteMemberFromSubscription(dm)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed delete member from group subs")
        return
    }

    JSON.Response(w, http.StatusOK, map[string]string{
        "response": "successeful delete member from group subs",
    })
}

```

Листинг Б.5 – Код файла user_handler.go

```

package handler

import (
    "net/http"
    "strconv"

    "subscription-service/internal/models"
    request "subscription-service/internal/models/request"
    JSON "subscription-service/pkg/json"
    "github.com/gorilla/mux"
)

// Handler'ы учатсника
func (h *Handler) JoinToGroupSubscription(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на присоединение к групповой подписки")

    inviteCode := request.InviteCode{}

```

```

    err := JSON.Parse(r, &inviteCode)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
        return
    }

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    userId, _ := h.services.Auth.GetUserIdFromToken(token)

    err =
h.services.SubscriptionMember.JoinToGroupSubscription(userId,
inviteCode)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed join to group subscription")
        return
    }

    JSON.Response(w, http.StatusOK, map[string]string{
        "response": "user succeseful join to group subscription",
    })
}

func (h *Handler) ExitFromGroupSubscription(w http.ResponseWriter,
r *http.Request) {
    h.log.Debug("Запрос на выход из групповой подписки")

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    userId, _ := h.services.Auth.GetUserIdFromToken(token)

    subId := models.SubscriptionId{}

    err := JSON.Parse(r, &subId)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
        return
    }

    err = h.services.ExitFromGroupSubscription(userId, subId)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed exit from group subscription")
        return
    }
}

```

```

        JSON.Response(w, http.StatusOK, map[string]string{
            "response": "successeful exit from group subscription",
        })
    }

func (h *Handler) ChangePaymentStatusUser(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на изменение статуса оплаты групповой
подписки")

    pscr := request.PaymentStatusChangeRequest{}

    err := JSON.Parse(r, &pscr)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
        return
    }

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    userId, _ := h.services.Auth.GetUserIdFromToken(token)

    err = h.services.ChangePaymentStatusMember(userId, pscr)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed change payment status")
        return
    }

    JSON.Response(w, http.StatusOK, map[string]string{
        "response": "successeful change payment status",
    })
}

// Handler владельца
func (h *Handler) GetAllMembersInSubs(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на получение участников групповой
подписки")

    vars := mux.Vars(r)
    idStr := vars["id"]

    id, err := strconv.Atoi(idStr)
    if err != nil {
        http.Error(w, "Invalid ID", http.StatusBadRequest)
        return
    }
}

```

```

        members, err :=
h.services.SubscriptionMember.GetAllMembersInSubs(uint(id))
        if err != nil {
            h.log.Error(err.Error())
            JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
            return
        }

        JSON.Response(w, http.StatusOK, members)
    }

func (h *Handler) ChangeAmountMembers(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на изменение цену участника")

    am := request.AmountMember{}

    err := JSON.Parse(r, &am)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
        return
    }

    err = h.services.ChangeAmountMembers(am)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed change amount")
        return
    }

    JSON.Response(w, http.StatusOK, map[string]string{
        "response": "succeseful change amount",
    })
}

func (h *Handler) DeleteMemberFromSubscription(w
http.ResponseWriter, r *http.Request) {
    h.log.Debug("Запрос на удаление участника из групповой
подписки")

    dm := request.DeleteMember{}

    err := JSON.Parse(r, &dm)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed parse JSON")
    }
}

```

```

        return
    }

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    ownerId, _ := h.services.Auth.GetUserIdFromToken(token)

    dm.OwnerId = ownerId

    err = h.services.DeleteMemberFromSubscription(dm)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusBadRequest, err.Error(),
"Failed delete member from group subs")
        return
    }

    JSON.Response(w, http.StatusOK, map[string]string{
        "response": "succeseful delete member from group subs",
    })
}

```

Листинг Б.6 – Код файла auth.go

```

package handler

import (
    "fmt"
    "net/http"
    "strings"

    JSON "subscription-service/pkg/json"
)

func (h *Handler) AuthMiddleware(next http.HandlerFunc)
http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        tokenString, err := h.GetJWTTokenFromRequest(r)
        if err != nil {
            http.Error(w, err.Error(), http.StatusUnauthorized)
            return
        }

        _, err = h.services.Auth.ValidateToken(tokenString)
        if err != nil {
            http.Error(w, "Неверный токен",
http.StatusUnauthorized)
            return
        }

        next(w, r)
    }
}

```

```

func (h *Handler) GetJWTTokenFromRequest(r *http.Request) (string,
error) {
    authHeader := r.Header.Get("Authorization")
    if authHeader == "" {
        return "", fmt.Errorf("token not provided")
    }

    tokenString := strings.TrimPrefix(authHeader, "Bearer ")

    return tokenString, nil
}

func (h *Handler) Validate(w http.ResponseWriter, r *http.Request)
{
    h.log.Debug("Валидация успешна")
    JSON.Response(w, http.StatusOK, "validation success")
}

```

Листинг Б.7 – Код файла analysis_handler.go

```

package handler

import (
    "fmt"
    "net/http"
    "strconv"
    "time"

    JSON "subscription-service/pkg/json"
)

func (h *Handler) DownloadAnalysisInPDF(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на скачивание pdf-файла")

    params, err := h.ParseParams(r)
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    userId, _ := h.services.Auth.GetUserIdFromToken(token)

    result, err :=
h.services.Analysis.GetAnalysisDataInInterval(userId,
params.Interval, params.From, params.To, params.Offset)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusInternalServerError,
err.Error(), "Failed to get all subscriptions")
    }
}

```

```

        return
    }

    pdfBytes, err :=
h.services.Analysis.GenerateAnalyticsPDF(result)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    w.Header().Set("Content-Type", "application/pdf")
    w.Header().Set("Content-Disposition", "attachment;
filename=analytics.pdf")
    w.Write(pdfBytes)
}

func (h *Handler) GetAnalysisData(w http.ResponseWriter, r
*http.Request) {
    h.log.Debug("Запрос на получение все подписок пользователя по
id")

    params, err := h.ParseParams(r)
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }

    tokenString, _ := h.GetJWTTokenFromRequest(r)
    token, _ := h.services.Auth.GetJWTToken(tokenString)
    userId, _ := h.services.Auth.GetUserIdFromToken(token)

    result, err :=
h.services.Analysis.GetAnalysisDataInInterval(userId,
params.Interval, params.From, params.To, params.Offset)
    if err != nil {
        h.log.Error(err.Error())
        JSON.ErrorResponse(w, http.StatusInternalServerError,
err.Error(), "Failed to get all subscriptions")
        return
    }

    JSON.Response(w, http.StatusOK, result)
}

type Params struct {
    Interval string    // next_month, next_quarter, next_year или
custom
    From      time.Time // начало периода (если custom) или
вычисленное GetPeriodByInterval(...)
    To        time.Time // конец периода (если custom) или
вычисленное GetPeriodByInterval(...)
    Offset    int        // смещение для гистограммы

```

```

}

func (h *Handler) ParseParams(r *http.Request) (Params, error) {
    q := r.URL.Query()

    // 1. interval
    interval := q.Get("interval")
    if interval == "" {
        interval = "next_month"
    }

    // 2. offset
    offset := 0
    if offStr := q.Get("offset"); offStr != "" {
        if parsed, err := strconv.Atoi(offStr); err == nil {
            offset = parsed
        }
    }

    // 3. from/to (либо вычисляем через GetPeriodByInterval)
    now := time.Now()
    var from, to time.Time
    if interval == "custom" {
        fromStr := q.Get("from")
        toStr := q.Get("to")
        if fromStr == "" || toStr == "" {
            return Params{}, fmt.Errorf("for custom interval you must provide both 'from' and 'to'")
        }

        var err error
        from, err = time.Parse("2006-01-02", fromStr)
        if err != nil {
            return Params{}, fmt.Errorf("invalid 'from' date: %w", err)
        }
        to, err = time.Parse("2006-01-02", toStr)
        if err != nil {
            return Params{}, fmt.Errorf("invalid 'to' date: %w", err)
        }

        if from.After(to) {
            return Params{}, fmt.Errorf("'from' must be before or equal to 'to'")
        }
    } else {
        // predetermined intervals
        var err error
        from, to, err =
h.services.Analysis.GetPeriodByInterval(interval, now)
        if err != nil {

```

```

        return Params{}, fmt.Errorf("unsupported interval:
%s", interval)
    }
}

return Params{
    Interval: interval,
    From:     from,
    To:       to,
    Offset:   offset,
}, nil
}

```

Листинг Б.8 – Код файла service.go

```

package service

import (
    "fmt"
    "time"

    "subscription-service/internal/models"
    "subscription-service/internal/models/request"
    "subscription-service/internal/models/response"
    "subscription-service/internal/repository"
    "subscription-service/pkg/logger"
    "github.com/golang-jwt/jwt/v5"
)

type Auth interface {
    GenerateToken(userId uint) (string, error)
    ValidateToken(tokenString string) (uint, error)
    GetJWTToken(tokenString string) (*jwt.Token, error)
    GetUserIdFromToken(token *jwt.Token) (uint, error)
}

type User interface {
    Create(user models.User) (int, error)
    Update(user models.User) (models.User, error)
    Get(userId int) (models.User, error)
    GetByLogin(login string) (models.User, error)
    Delete(userId int) error
}

type Subscription interface {
    Create(sub models.Subscription) (int, error)
    Update(sub models.Subscription) (models.Subscription, error)
    Get(subId uint) (models.Subscription, error)
    GetAllByStatus(status models.Status) ([]models.Subscription,
error)
    GetByInviteCode(inviteCode string) (models.Subscription,
error)
    GetAllByUserId(userId uint) ([]models.Subscription, error)
}

```

```

        GetByUserAndStatus(userID uint, status models.Status)
        ([]models.Subscription, error)
        Delete(subId uint) error
    }

type SubscriptionMember interface {
    JoinToGroupSubscription(userID uint, inviteCode
request.InviteCode) error
    ExitFromGroupSubscription(userID uint, subId
models.SubscriptionId) error
    ChangePaymentStatusMember(userID uint, pscr
request.PaymentStatusChangeRequest) error
    GetAllMembersInSubs(subId uint) ([]response.User, error)
    ChangeAmountMembers(am request.AmountMember) error
    Get(subId uint) (models.SubscriptionMember, error)
    Delete(userID uint, subId uint) error
    ExistUserInGroupSubscription(userID uint, subId uint) (bool,
error)
    DeleteMemberFromSubscription(am request.DeleteMember) error
}

type Analysis interface {
    GetAnalysisDataInInterval(userID uint, interval string, from
time.Time, to time.Time, offset int) (response.Analytics, error)
    GetPeriodByInterval(interval string, now time.Time) (start
time.Time, end time.Time, err error)
    GenerateAnalyticsPDF(data response.Analytics) ([]byte, error)
}

type Service struct {
    Auth
    User
    Subscription
    SubscriptionMember
    Analysis
    *TelegramBotService
}

func NewService(r *repository.Repository, l logger.Logger,
secretKey string) *Service {
    sub := NewSubscriptionService(r, l)
    telegram, _ := NewTelegramBotService(r, l)
    return &Service{
        Auth:          NewAuthJWTService(secretKey),
        User:          NewUserService(r, l),
        Subscription:   sub,
        SubscriptionMember: NewSubscriptionMemberService(r, sub,
l),
        Analysis:       NewAnalysisService(sub, l),
        TelegramBotService: telegram,
    }
}

```

```

func (s *Service) CheckSubscriptions() error {
    today := time.Now().UTC().Truncate(24 * time.Hour)

    subs, err := s.Subscription.GetAllByStatus(models.Active)
    if err != nil {
        return err
    }

    for _, sub := range subs {
        baseDate := sub.PaymentDate.Truncate(24 * time.Hour)
        intervalDays := int(sub.PaymentInterval)
        if intervalDays <= 0 {
            continue
        }

        var nextDue time.Time
        if baseDate.After(today) || sameYMD(baseDate, today) {
            nextDue = baseDate
        } else {
            deltaDays := int(today.Sub(baseDate).Hours() / 24)
            k := deltaDays / intervalDays
            nextDue = baseDate.AddDate(0, 0, (k+1)*intervalDays)
        }

        notifyDate := nextDue.AddDate(0, 0, -
int(sub.DaysNotifyBeforePayment))

        if sameYMD(today, notifyDate) {
            text := fmt.Sprintf(
                "🔔 <b>Напоминание</b>\n\n"+
                "По подписке «%s» запланировано списание
<b>%.2f₽</b>.\n"+
                "Дата следующего платежа: <b>%s</b>\n\n"+
                "Пожалуйста, не забудьте оплатить
вовремя!",
                sub.Name,
                sub.Amount,
                nextDue.Format("2006-01-02"),
            )

            user, err := s.User.Get(int(sub.OwnerId))
            if err != nil {
                s.log.Error(err.Error())
                continue
            }

            s.TelegramBotService.sendMessage(*user.ChatId,
text)
        }
    }
}

```

```

        return nil
    }

    func sameYMD(a, b time.Time) bool {
        return a.Year() == b.Year() &&
            a.Month() == b.Month() &&
            a.Day() == b.Day()
    }

```

Листинг Б.9 – Код файла auth_service.go

```

package service

import (
    "time"

    "github.com/golang-jwt/jwt/v5"
)

type AuthJWTService struct {
    secretKey string
}

func NewAuthJWTService(sk string) *AuthJWTService {
    return &AuthJWTService{secretKey: sk}
}

func (ajs *AuthJWTService) GenerateToken(userId uint) (string,
error) {
    claims := jwt.MapClaims{
        "user_id": userId,
        "exp":      time.Now().Add(time.Hour * 24).Unix(), // Срок
        действия – 24 часа
    }

    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
    return token.SignedString([]byte(ajs.secretKey))
}

func (ajs *AuthJWTService) ValidateToken(tokenString string)
(uint, error) {
    token, err := ajs.GetJWTToken(tokenString)
    if err != nil {
        return 0, err
    }

    return ajs.GetUserIdFromToken(token)
}

func (ajs *AuthJWTService) GetJWTToken(tokenString string)
(*jwt.Token, error) {

```

```

    token, err := jwt.Parse(tokenString, func(token *jwt.Token)
(interface{}, error) {
    if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
        return nil, jwt.ErrTokenSignatureInvalid
    }
    return []byte(ajs.secretKey), nil
}))

    if err != nil || !token.Valid {
        return &jwt.Token{}, err
    }

    return token, err
}

func (ajs *AuthJWTService) GetUserIdFromToken(token *jwt.Token)
(uint, error) {
    claims, ok := token.Claims.(jwt.MapClaims)
    if !ok {
        return 0, jwt.ErrTokenInvalidClaims
    }

    userIDFloat, ok := claims["user_id"].(float64)
    if !ok {
        return 0, jwt.ErrTokenInvalidClaims
    }

    return uint(userIDFloat), nil
}

```

Листинг Б.10 – Код файла subscription_member_service.go

```

package service

import (
    "fmt"
    "time"

    "subscription-service/internal/models"
    request "subscription-service/internal/models/request"
    "subscription-service/internal/models/response"
    "subscription-service/internal/repository"
    "subscription-service/pkg/logger"
)

type SubscriptionMemberService struct {
    repo repository.SubscriptionMember
    sub  Subscription
    log  logger.Logger
}

func NewSubscriptionMemberService(r *repository.Repository, s
Subscription, log logger.Logger) *SubscriptionMemberService {

```

```

        return &SubscriptionMemberService{
            repo: r.SubscriptionMember,
            sub:  s,
            log:  log,
        }
    }

    func (sms *SubscriptionMemberService) Create(subMem
models.SubscriptionMember) (int, error) {
        return sms.repo.Create(subMem)
    }

    func (sms *SubscriptionMemberService) Get(subMemId uint)
(models.SubscriptionMember, error) {
        return models.SubscriptionMember{}, nil
    }

    func (sms *SubscriptionMemberService) Delete(userId uint, subId
uint) error {
        return sms.repo.Delete(userId, subId)
    }

    func (sms *SubscriptionMemberService)
JoinToGroupSubscription(userId uint, inviteCode
request.InviteCode) error {
        err := inviteCode.Validate()
        if err != nil {
            sms.log.Error(err.Error())
            return err
        }

        subs, err := sms.sub.GetByInviteCode(inviteCode.Code)
        if err != nil {
            sms.log.Error(err.Error())
            return err
        }

        isUserInGroup, err := sms.ExistUserInGroupSubscription(userId,
subs.Id)
        if err != nil {
            sms.log.Error(err.Error())
            return err
        }

        if isUserInGroup {
            err = fmt.Errorf("this user is already joined to a group
subscription")
            sms.log.Error(err.Error())
            return err
        }

        subMember := models.SubscriptionMember{

```

```

        UserId:                userId,
        SubscriptionId:        subs.Id,
        Amount:                0,
        Joined_at:             time.Now(),
        DaysNotifyBeforePayment: subs.DaysNotifyBeforePayment,
        Color:                 subs.Color,
        Payment_status:        models.NotPaid,
    }

    // Ограничить константно количество мест в групповой подписке
    _, err = sms.repo.Create(subMember)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    return nil
}

func (sms *SubscriptionMemberService)
ExistUserInGroupSubscription(userId uint, subId uint) (bool,
error) {
    return sms.repo.ExistUserInGroupSubscription(userId, subId)
}

func (sms *SubscriptionMemberService)
ExitFromGroupSubscription(userId uint, subId
models.SubscriptionId) error {
    err := subId.Validate()
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    err = sms.repo.Delete(3, subId.Id)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    return nil
}

func (sms *SubscriptionMemberService)
ChangePaymentStatusMember(userId uint, pscr
request.PaymentStatusChangeRequest) error {
    err := pscr.Validate()
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }
}

```

```

    subMem, err := sms.repo.GetByUserIdAndSubId(userId, pscr.Id)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    subMem.Payment_status = pscr.PaymentStatus

    err = sms.repo.Update(subMem)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    return nil
}

func (sms *SubscriptionMemberService) ChangeAmountMembers(am
request.AmountMember) error {
    err := am.Validate()
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    subMem, err := sms.repo.GetByUserIdAndSubId(am.UserId, am.Id)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    subMem.Amount = am.Amount

    err = sms.repo.Update(subMem)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    return nil
}

func (sms *SubscriptionMemberService) GetAllMembersInSubs(subId
uint) ([]response.User, error) {
    members, err := sms.repo.GetMembersBySubId(subId)
    if err != nil {
        sms.log.Error(err.Error())
        return []response.User{}, err
    }

    return members, nil
}

```

```

func (sms *SubscriptionMemberService)
DeleteMemberFromSubscription(am request.DeleteMember) error {
    err := am.Validate()
    if err != nil {

        return err
    }

    subP, err := sms.sub.Get(am.SubscriptionId.Id)
    if err != nil {
        return err
    }

    if subP.OwnerId != am.OwnerId {
        err = fmt.Errorf("this is not the action of the owner of
the subscription")
        return err
    }

    err = sms.repo.Delete(am.UserId, am.Id)
    if err != nil {
        return err
    }

    return nil
}

```

Листинг Б.11 – Код файла subscription_service.go

```

package service

import (
    "fmt"
    "time"

    "subscription-service/internal/models"
    request "subscription-service/internal/models/request"
    "subscription-service/internal/models/response"
    "subscription-service/internal/repository"
    "subscription-service/pkg/logger"
)

type SubscriptionMemberService struct {
    repo repository.SubscriptionMember
    sub  Subscription
    log  logger.Logger
}

func NewSubscriptionMemberService(r *repository.Repository, s
Subscription, log logger.Logger) *SubscriptionMemberService {
    return &SubscriptionMemberService{
        repo: r.SubscriptionMember,

```

```

        sub: s,
        log: log,
    }
}

func (sms *SubscriptionMemberService) Create(subMem
models.SubscriptionMember) (int, error) {
    return sms.repo.Create(subMem)
}

func (sms *SubscriptionMemberService) Get(subMemId uint)
(models.SubscriptionMember, error) {
    return models.SubscriptionMember{}, nil
}

func (sms *SubscriptionMemberService) Delete(userId uint, subId
uint) error {
    return sms.repo.Delete(userId, subId)
}

func (sms *SubscriptionMemberService)
JoinToGroupSubscription(userId uint, inviteCode
request.InviteCode) error {
    err := inviteCode.Validate()
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    subs, err := sms.sub.GetByInviteCode(inviteCode.Code)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    isUserInGroup, err := sms.ExistUserInGroupSubscription(userId,
subs.Id)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    if isUserInGroup {
        err = fmt.Errorf("this user is already joined to a group
subscription")
        sms.log.Error(err.Error())
        return err
    }

    subMember := models.SubscriptionMember{
        UserId:          userId,
        SubscriptionId:  subs.Id,
    }

```

```

        Amount:                                0,
        Joined_at:                             time.Now(),
        DaysNotifyBeforePayment: subs.DaysNotifyBeforePayment,
        Color:                                  subs.Color,
        Payment_status:                         models.NotPaid,
    }

    // Ограничить константно количество мест в групповой подписке
    _, err = sms.repo.Create(subMember)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    return nil
}

func (sms *SubscriptionMemberService)
ExistUserInGroupSubscription(userId uint, subId uint) (bool,
error) {
    return sms.repo.ExistUserInGroupSubscription(userId, subId)
}

func (sms *SubscriptionMemberService)
ExitFromGroupSubscription(userId uint, subId
models.SubscriptionId) error {
    err := subId.Validate()
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    err = sms.repo.Delete(3, subId.Id)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    return nil
}

func (sms *SubscriptionMemberService)
ChangePaymentStatusMember(userId uint, pscr
request.PaymentStatusChangeRequest) error {
    err := pscr.Validate()
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    subMem, err := sms.repo.GetByUserIdAndSubId(userId, pscr.Id)
    if err != nil {

```

```

        sms.log.Error(err.Error())
        return err
    }

    subMem.Payment_status = pscr.PaymentStatus

    err = sms.repo.Update(subMem)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    return nil
}

func (sms *SubscriptionMemberService) ChangeAmountMembers(am
request.AmountMember) error {
    err := am.Validate()
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    subMem, err := sms.repo.GetByUserIdAndSubId(am.UserId, am.Id)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    subMem.Amount = am.Amount

    err = sms.repo.Update(subMem)
    if err != nil {
        sms.log.Error(err.Error())
        return err
    }

    return nil
}

func (sms *SubscriptionMemberService) GetAllMembersInSubs(subId
uint) ([]response.User, error) {
    members, err := sms.repo.GetMembersBySubId(subId)
    if err != nil {
        sms.log.Error(err.Error())
        return []response.User{}, err
    }

    return members, nil
}

```

```

func (sms *SubscriptionMemberService)
DeleteMemberFromSubscription(am request.DeleteMember) error {
    err := am.Validate()
    if err != nil {

        return err
    }

    subP, err := sms.sub.Get(am.SubscriptionId.Id)
    if err != nil {
        return err
    }

    if subP.OwnerId != am.OwnerId {
        err = fmt.Errorf("this is not the action of the owner of
the subscription")
        return err
    }

    err = sms.repo.Delete(am.UserId, am.Id)
    if err != nil {
        return err
    }

    return nil
}

```

Листинг Б.12 – Код файла telegram_bot_service.go

```

package service

import (
    "fmt"
    "strconv"
    "strings"

    "subscription-service/internal/repository"
    "subscription-service/pkg/logger"
    tgbotapi "github.com/go-telegram-bot-api/telegram-bot-api"
)

type TelegramBotService struct {
    bot *tgbotapi.BotAPI
    repo repository.User
    log logger.Logger
}

func NewTelegramBotService(r *repository.Repository, logger
logger.Logger) (*TelegramBotService, error) {
    bot, err := tgbotapi.NewBotAPI("7591078053:AAFqi-
Eb3VbpUgVwhKrn8er_3l6o7T4fyUg")
    if err != nil {

```

```

        return nil, fmt.Errorf("не удалось создать Telegram Bot
API: %w", err)
    }
    bot.Debug = false // или true, если нужен детальный отладочный
ВЫВОД

    return &TelegramBotService{
        bot: bot,
        repo: r.User,
        log: logger,
    }, nil
}

func (svc *TelegramBotService) Start() {
    svc.log.Info("TelegramBotService: запускаем обработчик
апдейтов")

    updateConfig := tgbotapi.NewUpdate(0)
    updateConfig.Timeout = 60

    updates, err := svc.bot.GetUpdatesChan(updateConfig)
    if err != nil {
        svc.log.Error("Ошибка при создании канала апдейтов
Telegram")
        return
    }

    for update := range updates {
        if update.Message == nil {
            continue
        }
        svc.handleMessage(update.Message)
    }
}

func (svc *TelegramBotService) handleMessage(msg
*tgbotapi.Message) {
    chatID := msg.Chat.ID
    text := msg.Text

    svc.log.Info("Получено сообщение от Telegram")

    if strings.HasPrefix(text, "/start") {
        args := strings.TrimSpace(strings.TrimPrefix(text,
"/start"))
        if args == "" {
            reply := "Чтобы получать уведомления, нажмите
«Привязать Telegram» на сайте и затем сюда вернитесь."
            svc.sendMessage(chatID, reply)
            return
        }
    }
}

```

```

        userID, err := strconv.ParseUint(args, 10, 32)
        if err != nil {
            svc.sendTextMessage(chatID, "Неверный формат
идентификатора пользователя.")
            return
        }

        user, err := svc.repo.Get(uint(userID))

        user.ChatId = &chatID

        if _, err := svc.repo.Update(user); err != nil {
            svc.log.Error("Не удалось вставить/обновить chat_id
в БД")
            svc.sendTextMessage(chatID, "Ошибка на сервере,
повторите позже.")
            return
        }

        svc.log.Info("Успешно привязан Telegram к пользователю")
        reply := fmt.Sprintf("✔ Telegram успешно привязан к вашему
аккаунту (user_id=%d).", userID)
        svc.sendTextMessage(chatID, reply)
        return
    }
}

func (svc *TelegramBotService) sendTextMessage(chatID int64, text
string) {
    msg := tgbotapi.NewMessage(chatID, text)
    msg.ParseMode = "HTML"
    if _, err := svc.bot.Send(msg); err != nil {
        svc.log.Error("Не удалось отправить сообщение в
Telegram")
    }
}

```

Листинг Б.13 – Код файла user_service.go

```

package service

import (
    "subscription-service/internal/models"
    "subscription-service/internal/repository"
    "subscription-service/pkg/logger"
)

type UserService struct {
    repo repository.User
    log  logger.Logger
}

```

```

func NewUserService(r *repository.Repository, log logger.Logger)
*UserService {
    return &UserService{
        repo: r.User,
        log:  log,
    }
}

func (us *UserService) Create(user models.User) (int, error) {
    return us.repo.Create(user)
}

func (us *UserService) Update(user models.User) (models.User,
error) {
    return us.repo.Update(user)
}

func (us *UserService) Get(userId int) (models.User, error) {
    return us.repo.Get(uint(userId))
}

func (us *UserService) GetByLogin(login string) (models.User,
error) {
    return us.repo.GetByLogin(login)
}

func (us *UserService) Delete(userId int) error {
    return us.repo.Delete(userId)
}

```

Листинг Б.14 – Код файла repository.go

```

package repository

import (
    "subscription-service/internal/models"
    "subscription-service/internal/models/response"
    "subscription-service/pkg/logger"
    "github.com/jmoiron/sqlx"
)

type User interface {
    Create(user models.User) (int, error)
    Update(user models.User) (models.User, error)
    Get(userId uint) (models.User, error)
    GetByLogin(login string) (models.User, error)
    Delete(userId int) error
}

type Subscription interface {
    Create(sub models.Subscription) (int, error)
    Update(sub models.Subscription) (models.Subscription, error)
    Get(subId uint) (models.Subscription, error)
}

```

```

        GetAllByStatus(status models.Status) ([]models.Subscription,
error)
        GetByInviteCode(inviteCode string) (models.Subscription,
error)
        GetAllByUser(userId uint) ([]models.Subscription, error)
        GetByUserAndStatus(userId uint, status models.Status)
([]models.Subscription, error)
        Delete(subId uint) error
    }

type SubscriptionMember interface {
    Create(subMem models.SubscriptionMember) (int, error)
    Update(subMem models.SubscriptionMember) error
    GetByUserIdAndSubId(userId uint, subId uint)
(models.SubscriptionMember, error)
    GetMembersBySubId(subId uint) ([]response.User, error)
    Delete(userId uint, subId uint) error
    ExistUserInGroupSubscription(userId uint, subId uint) (bool,
error)
}

type Repository struct {
    User
    Subscription
    SubscriptionMember
}

func NewRepository(db *sqlx.DB, log logger.Logger) *Repository {
    return &Repository{
        User:          NewUserPostgres(db, log),
        Subscription:  NewSubscriptionPostgres(db, log),
        SubscriptionMember: NewSubscriptionMemberPostgres(db,
log),
    }
}

```

Листинг Б.15 – Код файла subscription_member_repository.go

```

package repository

import (
    "fmt"

    "subscription-service/internal/models"
    "subscription-service/internal/models/response"
    "subscription-service/pkg/logger"
    "github.com/jmoiron/sqlx"
)

type SubscriptionMemberPostgres struct {

```

```

    db *sqlx.DB
    log logger.Logger
}

func NewSubscriptionMemberPostgres(db *sqlx.DB, log logger.Logger)
*SubscriptionMemberPostgres {
    return &SubscriptionMemberPostgres{db: db, log: log}
}

func (submemp *SubscriptionMemberPostgres) Create(subMem
models.SubscriptionMember) (int, error) {
    var id int
    query := fmt.Sprintf(`INSERT INTO %s (user_id,
subscription_id, amount, joined_at,
                        days_notify_before_payment, color, payment_status)
                        values ($1, $2, $3, $4, $5, $6, $7) RETURNING id`,
subscriptionMembersTable)

    row := submemp.db.QueryRow(query, subMem.UserId,
subMem.SubscriptionId, subMem.Amount, subMem.Joined_at,
subMem.DaysNotifyBeforePayment, subMem.Color,
subMem.Payment_status)
    if err := row.Scan(&id); err != nil {
        return 0, err
    }

    return id, nil
}

func (submemp *SubscriptionMemberPostgres) Update(subMem
models.SubscriptionMember) error {
    query := fmt.Sprintf(`UPDATE %s SET user_id = $1,
subscription_id = $2, amount = $3, joined_at = $4,
                        days_notify_before_payment = $5, color = $6, payment_status =
$7 WHERE id = $8`, subscriptionMembersTable)

    _, err := submemp.db.Exec(query, subMem.UserId,
subMem.SubscriptionId,
subMem.Amount, subMem.Joined_at,
subMem.DaysNotifyBeforePayment,
subMem.Color, subMem.Payment_status, subMem.Id)
    if err != nil {
        return nil
    }
    return nil
}

func (submemp *SubscriptionMemberPostgres)
GetByUserIdAndSubId(userId uint, SubId uint)
(models.SubscriptionMember, error) {
    query := fmt.Sprintf("SELECT * FROM %s WHERE user_id = $1 AND
subscription_id = $2", subscriptionMembersTable)

```

```

var subMem models.SubscriptionMember

err := submemp.db.Get(&subMem, query, userId, SubId)
if err != nil {
    return models.SubscriptionMember{}, err
}

return subMem, nil
}

func (submemp *SubscriptionMemberPostgres) GetMembersBySubId(subId
uint) ([]response.User, error) {
    query := fmt.Sprintf(`
        SELECT u.id, u.login, sm.amount, u.email
        FROM %s u
        JOIN %s sm ON sm.user_id = u.id
        WHERE sm.subscription_id = $1
    `, usersTable, subscriptionMembersTable)

    var members []response.User

    err := submemp.db.Select(&members, query, subId)
    if err != nil {
        return []response.User{}, err
    }

    return members, nil
}

func (submemp *SubscriptionMemberPostgres) Delete(userId uint,
subId uint) error {
    query := fmt.Sprintf(`
        DELETE FROM %s
        WHERE user_id = $1
        AND subscription_id = $2
    `, subscriptionMembersTable)

    result, err := submemp.db.Exec(query, userId, subId)
    if err != nil {
        return err
    }

    rowAff, err := result.RowsAffected()
    if err != nil {
        return err
    }

    if rowAff == 0 {
        return fmt.Errorf("no subscriptions(member) have been
deleted")
    }
}

```

```

        return nil
    }

    func (submemp *SubscriptionMemberPostgres)
    ExistUserInGroupSubscription(userId uint, subId uint) (bool,
    error) {
        var exists bool

        query := fmt.Sprintf(`
            SELECT EXISTS (
                SELECT 1
                FROM %s
                WHERE user_id = $1
                AND subscription_id = $2
            );
        `, subscriptionMembersTable)

        if err := submemp.db.Get(&exists, query, userId, subId); err
        != nil {
            return false, err
        }

        return exists, nil
    }

```

Листинг Б.15 – Код файла subscription_repository.go

```

package repository

import (
    "fmt"

    "subscription-service/internal/models"
    "subscription-service/pkg/logger"
    "github.com/jmoiron/sqlx"
)

type SubscriptionPostgres struct {
    db *sqlx.DB
    log logger.Logger
}

func NewSubscriptionPostgres(db *sqlx.DB, log logger.Logger)
*SubscriptionPostgres {
    return &SubscriptionPostgres{db: db, log: log}
}

func (subp *SubscriptionPostgres) Create(sub models.Subscription)
(int, error) {
    var id int
    query := fmt.Sprintf(`INSERT INTO %s (owner_id, name, amount,
    payment_date, payment_interval, status, type,

```

```

        days_notify_before_payment, created_at, updated_at, image,
        color, invite_token)
        values ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12,
        $13) RETURNING id`, subscriptionsTable)

        row := subp.db.QueryRow(query, sub.OwnerId, sub.Name,
        sub.Amount, sub.PaymentDate, sub.PaymentInterval, sub.Status,
        sub.Type, sub.DaysNotifyBeforePayment, sub.CreatedAt,
        sub.UpdatedAt, sub.Image, sub.Color, sub.InviteToken)
        if err := row.Scan(&id); err != nil {
            return 0, err
        }

        return id, nil
    }

func (subp *SubscriptionPostgres) Update(sub models.Subscription)
(models.Subscription, error) {
    // Список колонок в том порядке, в котором будем сканировать в
    struct
    // (должен совпадать с полями в RETURNING)
    returningCols := `
        id,
        owner_id,
        name,
        amount,
        payment_date,
        payment_interval,
        status,
        type,
        days_notify_before_payment,
        created_at,
        updated_at,
        image,
        color,
        invite_token
    `

    // Собираем UPDATE ... RETURNING
    query := fmt.Sprintf(`
        UPDATE %s
        SET
            owner_id           = $1,
            name                = $2,
            amount              = $3,
            payment_date        = $4,
            payment_interval    = $5,
            status               = $6,
            type                 = $7,
            days_notify_before_payment = $8,
            created_at          = $9,
            updated_at          = $10,

```

```

        image                = $11,
        color                 = $12,
        invite_token          = $13
    WHERE id = $14
    RETURNING %s
`, subscriptionsTable, returningCols)

// Переменная, в которую запишем обновлённую строку
var updated models.Subscription

// QueryRow возвращает единственную строку с полями,
перечисленными в RETURNING
row := subp.db.QueryRow(
    query,
    sub.OwnerId,
    sub.Name,
    sub.Amount,
    sub.PaymentDate,
    sub.PaymentInterval,
    sub.Status,
    sub.Type,
    sub.DaysNotifyBeforePayment,
    sub.CreatedAt,
    sub.UpdatedAt,
    sub.Image,
    sub.Color,
    sub.InviteToken,
    sub.Id,
)

// Сканируем все поля в updated
err := row.Scan(
    &updated.Id,
    &updated.OwnerId,
    &updated.Name,
    &updated.Amount,
    &updated.PaymentDate,
    &updated.PaymentInterval,
    &updated.Status,
    &updated.Type,
    &updated.DaysNotifyBeforePayment,
    &updated.CreatedAt,
    &updated.UpdatedAt,
    &updated.Image,
    &updated.Color,
    &updated.InviteToken,
)
if err != nil {
    return models.Subscription{}, err
}

return updated, nil

```

```

}

func (subp *SubscriptionPostgres) Get(subId uint)
(models.Subscription, error) {
    query := fmt.Sprintf("SELECT * FROM %s WHERE id = %1",
subscriptionsTable)

    var sub models.Subscription

    err := subp.db.Get(&sub, query, subId)
    if err != nil {
        return models.Subscription{}, err
    }

    return sub, nil
}

func (subp *SubscriptionPostgres) GetAllByStatus(status
models.Status) ([]models.Subscription, error) {
    query := fmt.Sprintf(`SELECT * FROM %s WHERE status = %1`,
subscriptionsTable)

    var subs []models.Subscription

    err := subp.db.Select(&subs, query, status)
    if err != nil {
        return []models.Subscription{}, err
    }

    return subs, nil
}

func (subp *SubscriptionPostgres) GetByInviteCode(inviteCode
string) (models.Subscription, error) {
    query := fmt.Sprintf("SELECT * FROM %s WHERE invite_token =
%1", subscriptionsTable)

    var sub models.Subscription

    err := subp.db.Get(&sub, query, inviteCode)
    if err != nil {
        return models.Subscription{}, err
    }

    return sub, nil
}

func (subp *SubscriptionPostgres) GetAllByUser(userId uint)
([]models.Subscription, error) {
    query := fmt.Sprintf(`
        SELECT
            s.id,

```

```

        s.owner_id,
        s.name,

        -- 1. Сумма: доля участника или полная цена
        CASE
            WHEN s.type = 'group' AND sm.user_id = $1 THEN
sm.amount
            ELSE s.amount
        END AS amount,

        s.payment_date,
        s.payment_interval,
        s.status,
        s.type,

        -- 2. Дни до уведомления: участника или владельца
        CASE
            WHEN s.type = 'group' AND sm.user_id = $1 THEN
sm.days_notify_before_payment
            ELSE s.days_notify_before_payment
        END AS days_notify_before_payment,

        s.created_at,
        s.updated_at,
        s.image,

        -- 3. Цвет: цвет участника или цвет подписки
        CASE
            WHEN s.type = 'group' AND sm.user_id = $1 THEN
sm.color
            ELSE s.color
        END AS color,

        s.invite_token
    FROM %s AS s
    LEFT JOIN subscription_members AS sm
        ON s.id = sm.subscription_id
        AND sm.user_id = $1
    WHERE s.owner_id = $1
        OR sm.user_id = $1
    `, subscriptionsTable)

    var subs []models.Subscription
    if err := subp.db.Select(&subs, query, userId); err != nil {
        return nil, err
    }
    return subs, nil
}

func (subp *SubscriptionPostgres) GetByUserAndStatus(userId uint,
status models.Status) ([]models.Subscription, error) {

```

```

        query := fmt.Sprintf(`SELECT * FROM %s WHERE owner_id = $1 AND
status = $2`, subscriptionsTable)

        var subs []models.Subscription

        err := subp.db.Select(&subs, query, userId, status)
        if err != nil {
            return []models.Subscription{}, err
        }

        return subs, nil
    }

    func (subp *SubscriptionPostgres) Delete(subId uint) error {
        return nil
    }

```

Листинг Б.16 – Код файла user_repository.go

```

package repository

import (
    "errors"
    "fmt"

    "subscription-service/internal/models"
    "subscription-service/pkg/logger"
    "github.com/jmoiron/sqlx"
)

type UserPostgres struct {
    db *sqlx.DB
    log logger.Logger
}

func NewUserPostgres(db *sqlx.DB, log logger.Logger) *UserPostgres {
    return &UserPostgres{db: db, log: log}
}

func (up *UserPostgres) Create(user models.User) (int, error) {
    var id int
    query := fmt.Sprintf("INSERT INTO %s (login, email,
hash_password, date_birthday) values ($1, $2, $3, $4) RETURNING
id", usersTable)

    row := up.db.QueryRow(query, user.Login, user.Email,
user.HashPassword, user.DateBirthday)
    if err := row.Scan(&id); err != nil {
        return 0, err
    }

    return id, nil
}

```

```

}

func (up *UserPostgres) Update(user models.User) (models.User,
error) {
    // Формируем SQL-запрос, который обновляет все поля по id
    query := fmt.Sprintf(
        "UPDATE %s SET login = $1, email = $2, hash_password = $3,
date_birthday = $4, chat_id = $5 WHERE id = $6",
        usersTable,
    )

    _, err := up.db.Exec(
        query,
        user.Login,
        user.Email,
        user.HashPassword,
        user.DateBirthday,
        user.ChatId,
        user.Id,
    )
    if err != nil {
        return models.User{}, err
    }

    return user, nil
}

func (up *UserPostgres) Get(userId uint) (models.User, error) {
    var user models.User
    query := fmt.Sprintf("SELECT * FROM %s WHERE id = $1",
usersTable)
    err := up.db.Get(&user, query, userId)
    return user, err
}

func (up *UserPostgres) GetByLogin(login string) (models.User,
error) {
    var user models.User
    query := fmt.Sprintf("SELECT * FROM %s WHERE login = $1",
usersTable)
    err := up.db.Get(&user, query, login)
    return user, err
}

func (up *UserPostgres) Delete(userId int) error {
    query := fmt.Sprintf("DELETE FROM %s WHERE id = $1",
usersTable)
    res, err := up.db.Exec(query, userId)
    if err != nil {
        return err
    }
}

```

```

    affRows, err := res.RowsAffected()
    if affRows == 0 {
        return errors.New("User with this ID does not exist")
    }

    return err
}

```

Листинг Б.17 – Код файла postgres.go

```

package repository

import (
    "fmt"

    "github.com/jmoiron/sqlx"
    _ "github.com/lib/pq"
)

const (
    usersTable           = "users"
    subscriptionsTable   = "subscriptions"
    subscriptionMembersTable = "subscription_members"
)

type ConfigPostgres struct {
    Host      string
    Port      string
    Username  string
    Password  string
    DBName    string
    SSLMode   string
}

func NewConfigPostgres(host string, port string, username string,
    password string, DBname string, sslmode string) *ConfigPostgres {
    return &ConfigPostgres{
        Host:      host,
        Port:      port,
        Username:  username,
        Password:  password,
        DBName:    DBname,
        SSLMode:   sslmode,
    }
}

func NewPostgresDB(cfgpg *ConfigPostgres) (*sqlx.DB, error) {
    db, err := sqlx.Open("postgres", fmt.Sprintf("host=%s port=%s
    user=%s dbname=%s password=%s sslmode=%s",
        cfgpg.Host, cfgpg.Port, cfgpg.Username, cfgpg.DBName,
        cfgpg.Password, cfgpg.SSLMode))
    if err != nil {
        return nil, err
    }
}

```

```

    }

    err = db.Ping()
    if err != nil {
        return nil, err
    }

    return db, nil
}

```

Листинг Б.18 – Код файл user.go

```

package models

import (
    "time"

    validation "github.com/go-ozzo/ozzo-validation"
    "github.com/go-ozzo/ozzo-validation/is"
)

type User struct {
    Id            uint        `json:"id" db:"id"`
    Login         string      `json:"login" db:"login"`
    Password      string      `json:"password" // Поле с формы`
    HashPassword string      `db:"hash_password"`
    Email         string      `json:"email" db:"email"`
    DateBirthDay string      `json:"date_birthday"`
    db:"date_birthday"`
    Created_at    time.Time  `db:"created_at"`
    ChatId        *int64     `json:"chat_id,omitempty" db:"chat_id"`
}

func (u User) Validate() error {
    return validation.ValidateStruct(&u,
        validation.Field(&u.Login, validation.Required),
        validation.Field(&u.Password, validation.Required))
}

type NewUser struct {
    Login         string `json:"login"`
    Password      string `json:"password" // Поле с формы`
    Email         string `json:"email"`
    DateBirthDay string `json:"date_birthday"`
}

func (nw NewUser) Validate() error {
    return validation.ValidateStruct(&nw,
        validation.Field(&nw.Login, validation.Required),
        validation.Field(&nw.Password, validation.Required),
        validation.Field(&nw.Email, validation.Required,
is.Email),
        validation.Field(&nw.DateBirthDay, validation.Required))
}

```

```
}
```

Листинг Б.19 – Код файла subscription.go

```
package models

import (
    "time"

    validation "github.com/go-ozzo/ozzo-validation"
)

type Status string
type Type string

const (
    Active    Status = "active"
    InActive  Status = "inactive"
    Deleted   Status = "deleted"
)

const (
    Individual Type = "individual"
    Group      Type = "group"
)

type Subscription struct {
    Id                uint        `json:"id" db:"id"`
    OwnerId           uint        `json:"owner_id" db:"owner_id"`
    Name              string      `json:"name" db:"name"`
    Amount            float64     `json:"amount" db:"amount"`
    PaymentDate       time.Time   `json:"payment_date" db:"payment_date"`
    PaymentInterval   uint        `json:"payment_interval" db:"payment_interval"`
    Status             Status      `json:"status" db:"status"`
    Type              Type        `json:"type" db:"type"`
    DaysNotifyBeforePayment uint      `json:"days_notify_before_payment" db:"days_notify_before_payment"`
    CreatedAt         time.Time   `json:"created_at" db:"created_at"`
    UpdatedAt         time.Time   `json:"updated_at" db:"updated_at"`
    Image             []byte      `json:"image" db:"image"`
    Color             string      `json:"color" db:"color"`
    InviteToken       *string     `json:"invite_token" db:"invite_token"`
}

func (sscpt Subscription) Validate() error {
    return validation.ValidateStruct(&sscpt,
```

```

        validation.Field(&sscpt.Name, validation.Required),
        validation.Field(&sscpt.Amount, validation.Required),
        validation.Field(&sscpt.PaymentDate,
validation.Required),
        validation.Field(&sscpt.PaymentInterval,
validation.Required),
        validation.Field(&sscpt.Type, validation.Required),
        validation.Field(&sscpt.Color, validation.Required))
}

```

Листинг Б.20 – Код файла subscription_member.go

```

package models

import (
    "time"

    validation "github.com/go-ozzo/ozzo-validation"
)

type Status string
type Type string

const (
    Active    Status = "active"
    InActive  Status = "inactive"
    Deleted   Status = "deleted"
)

const (
    Individual Type = "individual"
    Group      Type = "group"
)

type Subscription struct {
    Id                uint        `json:"id" db:"id"`
    OwnerId           uint        `json:"owner_id" db:"owner_id"`
    Name              string      `json:"name" db:"name"`
    Amount            float64     `json:"amount" db:"amount"`
    PaymentDate       time.Time   `json:"payment_date" db:"payment_date"`
    PaymentInterval   uint        `json:"payment_interval" db:"payment_interval"`
    Status             Status      `json:"status" db:"status"`
    Type              Type        `json:"type" db:"type"`
    DaysNotifyBeforePayment uint      `json:"days_notify_before_payment" db:"days_notify_before_payment"`
    CreatedAt          time.Time   `json:"created_at" db:"created_at"`
    UpdatedAt          time.Time   `json:"updated_at" db:"updated_at"`
}

```

```

        Image                []byte        `json:"image" db:"image"`
        Color                string        `json:"color" db:"color"`
        InviteToken          *string    `json:"invite_token"
db:"invite_token"`
    }

    func (sscpt Subscription) Validate() error {
        return validation.ValidateStruct(&sscpt,
            validation.Field(&sscpt.Name, validation.Required),
            validation.Field(&sscpt.Amount, validation.Required),
            validation.Field(&sscpt.PaymentDate,
validation.Required),
            validation.Field(&sscpt.PaymentInterval,
validation.Required),
            validation.Field(&sscpt.Type, validation.Required),
            validation.Field(&sscpt.Color, validation.Required))
    }

```

Приложение В
(обязательное)

Код клиентского приложения

Листинг В.1 – Код файла App.js

```
import React from 'react';
import { BrowserRouter, Route, Routes, redirect } from 'react-router-dom';
import ProtectedRoute from "../Components/ProtectedRoute";
import MainLayout from "../Components/MainLayout";

import LoginPage from "../Pages/LoginPage";
import RegistrationPage from "../Pages/RegistrationPage";
import HomePage from "../Pages/HomePage";
import SubscriptionsPage from "../Pages/SubscriptionsPage";
import AnalyticsPage from "../Pages/AnalyticsPage";
import ProfilePage from "../Pages/ProfilePage";

function App() {
  return (
    <BrowserRouter>
      <Routes>
        {/* Публичные страницы входа и регистрации */}
        <Route path="/login" element={<LoginPage />} />
        <Route path="/registration" element={<RegistrationPage />} />
      </Routes>

      {/* Защищённые маршруты под MainLayout */}
      <Route element={<MainLayout />}>
        <Route
          path="/"
          element={
            <ProtectedRoute element={<HomePage />}
            redirectTo="/login" />
          }
        />
        <Route
          path="/subscriptions"
          element={
            <ProtectedRoute element={<SubscriptionsPage />}
            redirectTo="/login" />
          }
        />
        <Route
          path="/profile"
          element={
            <ProtectedRoute element={<ProfilePage />}
            redirectTo="/login" />
          }
        />
        <Route
          path="/analytics"
          element={
            <ProtectedRoute element={<AnalyticsPage />}
            redirectTo="/login" />
          }
        />
      </Route>
    </BrowserRouter>
  );
}
```

```

        />
      </Route>
    </Routes>
  </BrowserRouter>
);
}

```

```
export default App;
```

Листинг В.2 – Код файла HomePage.js

```

// src/Pages/HomePage.jsx
import React, { useState, useEffect, useCallback } from "react";
import { addDays, isBefore, parseISO } from "date-fns";
import SubscriptionForm from
"../Components/Subscription/SubscriptionForm";
import Sidebar from "../Components/Sidebar";
import CalendarNavigation from
"../Components/Calendar/CalendarNavigation";
import CalendarGrid from "../Components/Calendar/CalendarGrid";
import SubscriptionDetails from
"../Components/Subscription/SubscriptionDetails";
import Modal from "../Components/ModalsWindow/Modal";
import SuccessModal from "../Components/SuccessModal";

export default function HomePage() {
  const today = new Date();
  const [date, setDate] = useState(today);
  const [viewMonth, setViewMonth] = useState(today.getMonth());
  const [viewYear, setViewYear] = useState(today.getFullYear());
  const [menuOpen, setMenuOpen] = useState(false);
  const [showForm, setShowForm] = useState(false);
  const [showSuccessModal, setShowSuccessModal] = useState(false);
  const [successType, setSuccessType] = useState("");

  const [subscriptions, setSubscriptions] = useState([]);

  // Вынесенная функция загрузки подписок
  const fetchSubscriptions = useCallback(async () => {
    try {
      const token = localStorage.getItem("token");
      if (!token) return;

      const response = await
fetch("http://localhost:8080/subscriptions/active", {
  headers: {
    Authorization: `Bearer ${token}`,
  },
});
      if (!response.ok) {
        throw new Error("Не удалось загрузить подписки");
      }
      const data = await response.json();
    }
  }, []);
}

```

```

const endDate = new Date();
endDate.setFullYear(endDate.getFullYear() + 1);

const repeated = [];
data.forEach((sub) => {
  let currentDate = new Date(sub.payment_date);
  const interval = sub.payment_interval;

  while (isBefore(currentDate, endDate)) {
    repeated.push({
      id: sub.id,
      original_id: sub.id,
      name: sub.name,
      date: currentDate.toISOString().split("T")[0],
      icon: "",
      price: sub.amount,
      color: sub.color,
      type: sub.type,
      notify: sub.days_notify_before_payment,
    });
    currentDate = addDays(currentDate, interval);
  }
});

setSubscriptions(repeated);
} catch (error) {
  console.error("Ошибка загрузки подписок:", error);
}
}, []);

useEffect(() => {
  fetchSubscriptions();
}, [fetchSubscriptions]);

const handleMonthChange = (offset) => {
  let m = viewMonth + offset,
      y = viewYear;
  if (m > 11) {
    m = 0;
    y++;
  }
  if (m < 0) {
    m = 11;
    y--;
  }
  setViewMonth(m);
  setViewYear(y);
  setDate(new Date(y, m, 1));
  setShowForm(false);
};

```

```

const handleAdd = (data) => {
  // Можно оставить пустым или удалить, если не нужен
  setShowForm(false);
};

const handleSuccess = ({ type }) => {
  setShowForm(false);
  setSuccessType(type);
  setShowSuccessModal(true);
  fetchSubscriptions();
  setTimeout(() => setShowSuccessModal(false), 2000);
};

const monthTotal = subscriptions
  .filter((sub) => {
    if (!sub.date) return false;
    const subDate = parseISO(sub.date);
    return subDate.getMonth() === viewMonth &&
    subDate.getFullYear() === viewYear;
  })
  .reduce((sum, sub) => sum + (sub.price || 0), 0);

const handleNotifySubscription = () => {
  alert("Вы подписались на уведомления о списаниях за текущий
  месяц.");
};

return (
  <div className="relative">
    { /* Кнопка «гамбургер» */ }
    <button
      className="
        fixed top-2 left-2
        w-9 h-9 bg-gray-800
        text-white rounded-md text-sm
        z-50 flex items-center justify-center
        hover:bg-gray-700 focus:outline-none
      "
      onClick={() => setMenuOpen((o) => !o)}
    >
      ≡
    </button>
    <Sidebar menuOpen={menuOpen} setMenuOpen={setMenuOpen} />

    <div className="max-w-4xl mx-auto p-4 sm:p-6 pt-14">
      <h1 className="text-3xl font-bold text-center text-blue-
600 mb-4">
        Панель управления подписками
      </h1>

      { /*
        === ВАРИАНТ А: карточка + «обтягивающаяся» кнопка ===

```

Используем justify-center, чтобы этот блок был по центру экрана.

Но сами элементы внутри не растягиваются, потому что у кнопки явным образом НЕ стоит w-full или flex-1.

```

    */}
    <div className="flex justify-center items-center mb-4
space-x-4">
      {/* Блок суммы */}
      <div className="bg-white rounded-lg shadow px-4 py-2
flex items-center gap-2 border-l-4 border-yellow-400 h-[58px]">
        <span className="text-xl">💰</span>
        <div>
          <div className="text-xs text-gray-500">Общая сумма
за месяц</div>
          <div className="text-sm font-bold text-black">
            {monthTotal.toFixed(2)} ₽
          </div>
        </div>
      </div>

      <div className="flex items-center h-[58px]">
        <button
          onClick={handleNotifySubscription}
          className="
            flex                                // <== теперь можно, внутри
            items-center
            justify-center
            h-[58px]
            px-5
            bg-blue-600 text-white
            text-base font-semibold
            rounded-lg
            hover:bg-blue-700
            focus:outline-none
            whitespace-nowrap
          "
        >
          Подписаться на уведомления
        </button>
      </div>
    </div>

    <CalendarNavigation
      viewMonth={viewMonth}
      viewYear={viewYear}
      handleMonthChange={handleMonthChange}
    />

    <CalendarGrid
      date={date}

```

```

        setDate={setDate}
        viewMonth={viewMonth}
        viewYear={viewYear}
        subscriptions={subscriptions}
      />

      <div className="mt-10">
        <SubscriptionDetails date={date}
subscriptions={subscriptions} />
      </div>

      <div className="flex justify-center mt-4">
        <button
          onClick={() => setShowForm((v) => !v)}
          className="px-4 py-2 bg-green-600 text-white rounded
hover:bg-green-700"
        >
          {showForm ? "Заккрыть" : "Добавить подписку"}
        </button>
      </div>

      {showSuccessModal && (
        <SuccessModal
          title={`$${successType} подписка успешно добавлена!`}
          description={`$${successType} подписка успешно
добавлена в ваш список`}
          duration={2000}
          onClose={() => setShowSuccessModal(false)}
        />
      )}

      {showForm && (
        <Modal onClose={() => setShowForm(false)}>
          <SubscriptionForm
            initialDate={date}
            onSubmit={handleAdd}
            onCancel={() => setShowForm(false)}
            onSuccess={handleSuccess}
          />
        </Modal>
      )}
    </div>
  </div>
);
}

```

Листинг В.3 – Код файла AnalyticsPage.js

```

// src/Pages/AnalyticsPage.jsx
import React, { useState, useMemo } from "react";
import { Bar, Pie } from "react-chartjs-2";
import {
  Chart as ChartJS,

```

```

    CategoryScale,
    LinearScale,
    BarElement,
    ArcElement,
    Tooltip,
    Legend,
  } from "chart.js";
import { addMonths, format } from "date-fns";
import { ru } from "date-fns/locale";

ChartJS.register(
  CategoryScale,
  LinearScale,
  BarElement,
  ArcElement,
  Tooltip,
  Legend
);

// ——— СТУБ: НАБОР ПОДПИСОК ———
const SUBSCRIPTIONS = [
  { id: 1, name: "Netflix", price: 499 },
  { id: 2, name: "Spotify", price: 199 },
  { id: 3, name: "YouTube", price: 249 },
  { id: 4, name: "Notion", price: 0 },
];
// —————

const INTERVALS = [
  { key: "next_month", label: "Следующий месяц" },
  { key: "next_quarter", label: "Следующий квартал" },
  { key: "next_year", label: "Следующий год" },
  { key: "custom", label: "Пользовательский..." },
];

const MOCK_FORECAST = {
  next_month: SUBSCRIPTIONS.map((s) => ({ name: s.name, amount:
s.price })),
  next_quarter: SUBSCRIPTIONS.map((s) => ({ name: s.name, amount:
s.price * 3 })),
  next_year: SUBSCRIPTIONS.map((s) => ({ name: s.name, amount:
s.price * 12 })),
};

export default function AnalyticsPage() {
  const [interval, setInterval] = useState("next_month");
  const [customRange, setCustomRange] = useState({ from: "", to:
"" });
  const [monthOffset, setMonthOffset] = useState(0);

  const windowSize = 6;
  const totalMonths = 12;

```

```

const maxOffset = totalMonths - windowSize;

const baseDate = useMemo(() => {
  const now = new Date();
  return new Date(now.getFullYear(), now.getMonth(), 1);
}, []);

const data = useMemo(() => {
  if (interval !== "custom") return MOCK_FORECAST[interval] ||
[];
  return [];
}, [interval]);

const total = useMemo(
  () => data.reduce((sum, d) => sum + d.amount, 0),
  [data]
);

const pieData = useMemo(
  () => ({
    labels: data.map((d) => d.name),
    datasets: [
      {
        data: data.map((d) => d.amount),
        backgroundColor: ["#e53e3e", "#38a169", "#3182ce",
"#805ad5"],
      },
    ],
  }),
  [data]
);

const barData = useMemo(() => {
  const labels = Array.from({ length: windowSize }, (_, i) =>
    format(addMonths(baseDate, monthOffset + i), "LLLL yyyy", {
locale: ru }));
  const values = Array(windowSize).fill(total);
  return {
    labels,
    datasets: [
      {
        label: "Прогноз трат",
        data: values,
        backgroundColor: "#3182ce",
      },
    ],
  };
}, [total, monthOffset, baseDate]);

return (
  <div className="max-w-4xl mx-auto p-6 space-y-6">

```

```

<h1 className="text-4xl font-bold text-center text-blue-
600">
  Прогноз трат
</h1>

{/* Кнопка скачивания PDF (файл вернёт сервер) */}
<div className="flex justify-end">
  <a
    href="/api/analytics/download"
    target="_blank"
    rel="noopener noreferrer"
    className="
      inline-flex items-center gap-2
      px-4 py-2 bg-blue-600 text-white
      rounded hover:bg-blue-700
      text-sm focus:outline-none
    "
  >
     Скачать анализ в PDF
  </a>
</div>

{/* Контент */}
<div>
  {/* Выбор интервала */}
  <div className="flex flex-wrap items-center gap-4 mb-4">
    <select
      className="border rounded-md p-2 bg-white text-sm"
      value={interval}
      onChange={(e) => {
        setInterval(e.target.value);
        setMonthOffset(0);
      }}
    >
      {INTERVALS.map((opt) => (
        <option key={opt.key} value={opt.key}>
          {opt.label}
        </option>
      ))}
    </select>
    {interval === "custom" && (
      <div className="flex items-center space-x-2">
        <input
          type="date"
          value={customRange.from}
          onChange={(e) =>
            setCustomRange((r) => ({ ...r, from:
e.target.value })))
        </div>
        <div className="border rounded-md p-2 text-sm"
        />
      <span>—</span>
    )}
  </div>

```

```

        <input
          type="date"
          value={customRange.to}
          onChange={ (e) =>
            setCustomRange((r) => ({ ...r, to:
e.target.value })))
        }
        className="border rounded-md p-2 text-sm"
      />
    </div>
  )}
</div>

{/* Инфоблок с общей суммой */}
<div className="p-4 bg-gray-50 rounded-lg mb-6">
  {interval !== "custom" ? (
    <p className="text-lg">
      За{" "}
      <strong>{INTERVALS.find((i) => i.key ===
interval).label}</strong>{" "}
      вы потратите примерно{" "}
      <span className="font-semibold">{total} ₺</span>.
    </p>
  ) : (
    <p className="text-lg">
      За период с <strong>{customRange.from}</strong> по{"
"}
      <strong>{customRange.to}</strong> вы потратите
примерно{" "}
      <span className="font-semibold">{total} ₺</span>.
    </p>
  )}
</div>

{/* Графики в одну колонку */}
<div className="grid grid-cols-1 gap-6">
  {/* Круговая диаграмма */}
  <div className="p-4 bg-white rounded-lg shadow">
    <h2 className="text-xl font-semibold mb-4">
      Распределение прогноза
    </h2>
    <div className="h-64">
      <Pie data={pieData} options={{ maintainAspectRatio:
false }} />
    </div>
  </div>

  {/* Динамика прогнозных трат */}
  <div className="p-4 bg-white rounded-lg shadow">
    {/* Слайдер месяцев */}
    <div className="flex items-center justify-center gap-4
mb-2">

```

```

<button
  type="button"
  onClick={() => setMonthOffset((o) => Math.max(o -
1, 0))}

  disabled={monthOffset === 0}
  className="inline-flex flex-none items-center
justify-center w-8 h-8 bg-gray-200 rounded-md hover:bg-gray-300
disabled:opacity-50 disabled:cursor-not-allowed focus:outline-
none"
  >
    ◀
  </button>
  <span className="text-sm whitespace-nowrap">
    {format(addMonths(baseDate, monthOffset), "LLLL
yyyy", {
      locale: ru,
    })}
    &nbsp;-&nbsp;
    {format(
      addMonths(baseDate, monthOffset + windowSize -
1),
      "LLLL yyyy",
      { locale: ru }
    )}
  </span>
  <button
    type="button"
    onClick={() =>
      setMonthOffset((o) => Math.min(o + 1,
maxOffset))
    }
    disabled={monthOffset === maxOffset}
    className="inline-flex flex-none items-center
justify-center w-8 h-8 bg-gray-200 rounded-md hover:bg-gray-300
disabled:opacity-50 disabled:cursor-not-allowed focus:outline-
none"
    >
      ▶
    </button>
  </div>

  <h2 className="text-xl font-semibold mb-4">Динамика
прогноза</h2>
  <div className="h-64">
    <Bar data={barData} options={{ maintainAspectRatio:
false }} />
  </div>
</div>
</div>
</div>
);

```

```
}
```

Листинг В.4 – Код файла LoginPage.js

```
import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom'; // Импорт
компонента Link
import '../CSS/Login.css'; // Подключение CSS для стилизации
import SuccessModal from '../Components/SuccessModal';

const LoginPage = ({ element, redirectTo }) => {
  const navigate = useNavigate();
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [errors, setErrors] = useState({});
  const [authError, setAuthError] = useState('');
  const [showSuccessModal, setShowSuccessModal] = useState(false);

  const validateForm = () => {
    const newErrors = {};
    if (!username.trim()) {
      newErrors.username = 'Имя пользователя не должно быть
пустым';
    }
    if (!password.trim()) {
      newErrors.password = 'Пароль не должен быть пустым';
    }
    return newErrors;
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    // Сбрасываем предыдущие ошибки перед валидацией
    setErrors({});
    setAuthError('');

    const formErrors = validateForm();
    if (Object.keys(formErrors).length > 0) {
      setErrors(formErrors);
      return;
    }

    // Запрос к реальному API для проверки логина и пароля
    const isValid = await authenticateUser(username, password);

    if (!isValid) {
      setAuthError('Неверное имя пользователя или пароль');
    } else {
      setAuthError('');
      setShowSuccessModal(true);
    }
  };
};
```

```

const authenticateUser = async (username, password) => {
  try {
    const obj = {login: username, password: password}
    const response = await
fetch('http://localhost:8080/user/login', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(obj),
  });

    if (!response.ok) {

      return false;
    }

    const data = await response.json();
    localStorage.setItem('token', data.jwt_token);
    return true;
  } catch (error) {
    console.error('Ошибка при аутентификации:', error);
    return false;
  }
};

return (
  <div>
    {showSuccessModal && (
      <SuccessModal
        title="Вход выполнен успешно!"
        description="Вы будете перенаправлены на главную
страницу..."
        duration={2000}
        onClose={() => {
          setShowSuccessModal(false);
          navigate("/");
        }}
      />
    )}
    <header className="header">
      <h1>Система управления подписками</h1>
    </header>
    <div className="login-container">
      <form className="login-form" onSubmit={handleSubmit}>
        <h2>Авторизация</h2>

        {authError && <div className="auth-
error">{authError}</div>}

        <div className="form-group">

```

```

        <label htmlFor="username">Имя пользователя</label>
        <input
          type="text"
          id="username"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          className={errors.username ? 'input-error' : ''}
        />
        {errors.username && <small className="error-
text">{errors.username}</small>}
      </div>

      <div className="form-group">
        <label htmlFor="password">Пароль</label>
        <input
          type="password"
          id="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          className={errors.password ? 'input-error' : ''}
        />
        {errors.password && <small className="error-
text">{errors.password}</small>}
      </div>

      <button type="submit">Войти</button>
    </form>

    <p className="register-link">
      Нет аккаунта? <Link
to="/registration">Зарегистрируйтесь</Link>
    </p>

  </div>
</div>
);
};

```

```
export default LoginPage;
```

Листинг В.5 – Код страницы ProfilePage.js

```

import React, { useState, useEffect } from "react";

export default function ProfilePage() {
  const [formData, setFormData] = useState({
    login: "",
    email: "",
    date_birthday: "",
  });

  const [passwordData, setPasswordData] = useState({
    newPassword: "",
  });

```

```

    confirmPassword: "",
  });

  const [isSaving, setIsSaving] = useState(false);
  const [isPasswordSaving, setIsPasswordSaving] = useState(false);

  const token = localStorage.getItem("token");

  useEffect(() => {
    const fetchProfile = async () => {
      try {
        const response = await
fetch("http://localhost:8080/user/profile", {
  headers: {
    Authorization: `Bearer ${token}`,
  },
});
        const data = await response.json();
        setFormData({
          login: data.login || "",
          email: data.email || "",
          date_birthday: data.date_birthday
            ? data.date_birthday.slice(0, 10)
            : "",
        });
      } catch (err) {
        console.error("Ошибка загрузки профиля", err);
      }
    };

    fetchProfile();
  }, [token]);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prev) => ({ ...prev, [name]: value }));
  };

  const handlePasswordChange = (e) => {
    const { name, value } = e.target;
    setPasswordData((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsSaving(true);

    try {
      const response = await
fetch("http://localhost:8080/user/update", {
      method: "PUT",
      headers: {

```

```

        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify({
        login: formData.login,
        email: formData.email,
        date_birthday: new
Date(formData.date_birthday).toISOString(),
    }),
    });

    if (!response.ok) throw new Error("Ошибка при обновлении
профиля");

    alert("Данные успешно обновлены!");
} catch (err) {
    console.error(err);
    alert("Ошибка при сохранении изменений");
} finally {
    setIsSaving(false);
}
};

const handlePasswordSubmit = async (e) => {
    e.preventDefault();

    if (passwordData.newPassword !== passwordData.confirmPassword)
    {
        return alert("Пароли не совпадают");
    }

    setIsPasswordSaving(true);

    try {
        const response = await
fetch("http://localhost:8080/user/change-password", {
            method: "PUT",
            headers: {
                "Content-Type": "application/json",
                Authorization: `Bearer ${token}`,
            },
            body: JSON.stringify({
                new_password: passwordData.newPassword,
            }),
        });

        if (!response.ok) throw new Error("Ошибка при смене
пароля");

        alert("Пароль успешно обновлён");
        setPasswordData({ newPassword: "", confirmPassword: "" });
    } catch (err) {

```

```

        console.error(err);
        alert("Ошибка при смене пароля");
    } finally {
        setIsPasswordSaving(false);
    }
};

return (
    <div className="max-w-3xl mx-auto p-6">
        <h1 className="text-2xl font-bold text-blue-600 mb-4">Настройки профиля</h1>

        {/* Форма основных данных */}
        <form
            onSubmit={handleSubmit}
            className="bg-white border rounded-xl p-6 shadow space-y-4"
        >
            <div>
                <label className="block mb-1 font-medium">Логин</label>
                <input
                    type="text"
                    name="login"
                    value={formData.login}
                    onChange={handleChange}
                    className="w-full border px-3 py-2 rounded-md"
                    required
                />
            </div>

            <div>
                <label className="block mb-1 font-medium">Email</label>
                <input
                    type="email"
                    name="email"
                    value={formData.email}
                    onChange={handleChange}
                    className="w-full border px-3 py-2 rounded-md"
                    required
                />
            </div>

            <div>
                <label className="block mb-1 font-medium">Дата
                рождения</label>
                <input
                    type="date"
                    name="date_birthday"
                    value={formData.date_birthday}
                    onChange={handleChange}
                    className="w-full border px-3 py-2 rounded-md"
                    required
                />
            </div>
        </form>
    </div>
);

```

```

        />
    </div>

    <div className="flex justify-end">
        <button
            type="submit"
            disabled={isSaving}
            className={`px-4 py-2 rounded text-white ${
                isSaving ? "bg-gray-400" : "bg-blue-600 hover:bg-
blue-700"
            }`}
        >
            {isSaving ? "Сохраняем..." : "Сохранить изменения"}
        </button>
    </div>
</form>

{/* Форма смены пароля */}
<form
    onSubmit={handlePasswordSubmit}
    className="bg-white border rounded-xl p-6 shadow space-y-4
mt-6"
>
    <h2 className="text-lg font-semibold text-gray-
800">Сменить пароль</h2>

    <div>
        <label className="block mb-1 font-medium">Новый
пароль</label>
        <input
            type="password"
            name="newPassword"
            value={passwordData.newPassword}
            onChange={handlePasswordChange}
            className="w-full border px-3 py-2 rounded-md"
        />
    </div>

    <div>
        <label className="block mb-1 font-medium">Подтвердите
пароль</label>
        <input
            type="password"
            name="confirmPassword"
            value={passwordData.confirmPassword}
            onChange={handlePasswordChange}
            className="w-full border px-3 py-2 rounded-md"
        />
    </div>

    <div className="flex justify-end">
        <button

```

```

        type="submit"
        disabled={isPasswordSaving}
        className={`px-4 py-2 rounded text-white ${
            isPasswordSaving ? "bg-gray-400" : "bg-blue-600
hover:bg-blue-700"
        }`}
    >
        {isPasswordSaving ? "Сохраняем..." : "Сменить пароль"}
    </button>
</div>
</form>
</div>
);
}

```

Листинг В.6 – Код файла RegistrationPage.js

```

import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import '../CSS/Register.css'; // Подключение CSS для стилизации
import SuccessModal from '../Components/SuccessModal';

const RegistrationPage = ({ element, redirectTo }) => {
    const navigate = useNavigate();
    const [username, setUsername] = useState('');
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [confirmPassword, setConfirmPassword] = useState('');
    const [errors, setErrors] = useState({});
    const [registerError, setRegisterError] = useState('');
    const [showSuccessModal, setShowSuccessModal] = useState(false);

    const validateForm = () => {
        const newErrors = {};

        if (!username.trim()) {
            newErrors.username = 'Имя пользователя не должно быть пустым';
        }
        if (!email.trim()) {
            newErrors.email = 'Email не должен быть пустым';
        } else if (!/^[\s@]+\.[\s@]+\.[\s@]+$/\.test(email)) {
            newErrors.email = 'Неверный формат email';
        }
        if (!password.trim()) {
            newErrors.password = 'Пароль не должен быть пустым';
        }
        if (password !== confirmPassword) {
            newErrors.confirmPassword = 'Пароли не совпадают';
        }

        return newErrors;
    };
};

```

```

const handleSubmit = async (e) => {
  e.preventDefault();

  // Сбрасываем предыдущие ошибки перед валидацией
  setErrors({});
  setRegisterError('');

  const formErrors = validateForm();
  if (Object.keys(formErrors).length > 0) {
    setErrors(formErrors);
    return;
  }

  setShowSuccessModal(true);
  setTimeout(() => {
    setShowSuccessModal(false);
    navigate("/login");
  }, 2000);

  // Запрос к реальному API для регистрации пользователя
  await registerUser(username, email, password);
};

const registerUser = async (username, email, password) => {
  try {
    const user = {
      login: username,
      password: password,
      email: email
    }
    console.log(user)
    const response = await
fetch('http://localhost:8040/users/create', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(user),
});

    if (!response.ok) {
      console.log(response.text())
      return false;
    }

    return true;
  } catch (error) {
    console.log('Ошибка при регистрации:', error);
    return false;
  }
};

```

```

return (
  <div>
    {showSuccessModal && (
      <SuccessModal
        title="Регистрация успешна!"
        description="Вы будете перенаправлены на страницу
входа..."
        duration={2000}
        onClose={() => {
          setShowSuccessModal(false);
          navigate("/login");
        }}
      />
    )}
    <header className="header">
      <h1>Система управления подписками</h1>
    </header>
    <div className="register-container">
      <form className="register-form" onSubmit={handleSubmit}>
        <h2>Регистрация</h2>

        {registerError && <div className="register-
error">{registerError}</div>}

        <div className="form-group">
          <label htmlFor="username">Имя пользователя</label>
          <input
            type="text"
            id="username"
            value={username}
            onChange={(e) => setUsername(e.target.value)}
            className={errors.username ? 'input-error' : ''}
          />
          {errors.username && <small className="error-
text">{errors.username}</small>}
        </div>

        <div className="form-group">
          <label htmlFor="email">Email</label>
          <input
            type="email"
            id="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className={errors.email ? 'input-error' : ''}
          />
          {errors.email && <small className="error-
text">{errors.email}</small>}
        </div>

        <div className="form-group">

```

```

        <label htmlFor="password">Пароль</label>
        <input
            type="password"
            id="password"
            value={password}
            onChange={ (e) => setPassword(e.target.value)}
            className={errors.password ? 'input-error' : ''}
        />
        {errors.password && <small className="error-
text">{errors.password}</small>}
    </div>

    <div className="form-group">
        <label htmlFor="confirmPassword">Подтверждение
пароля</label>
        <input
            type="password"
            id="confirmPassword"
            value={confirmPassword}
            onChange={ (e) => setConfirmPassword(e.target.value)}
            className={errors.confirmPassword ? 'input-error' :
''}
        />
        {errors.confirmPassword && <small className="error-
text">{errors.confirmPassword}</small>}
    </div>

    <button type="submit">Зарегистрироваться</button>
</form>

    <p className="register-link">
        Есть аккаунта? <Link to="/login">Выполните вход</Link>
    </p>
</div>
</div>
);
};

export default RegistrationPage;

```

Листинг В.7 – Код файла SubscriptionPage.js

```

import React, { useState, useEffect } from "react";
import { format } from "date-fns";
import SubscriptionForm from
"../Components/Subscription/SubscriptionForm";
import SubscriptionList from
"../Components/Subscription/SubscriptionList";
import Modal from "../Components/ModalsWindow/Modal"
import AddSubscriptionChoiceModal from
"../Components/ModalsWindow/AddSubscriptionChoiceModal";
import ConnectSubscriptionForm from
"../Components/ModalsWindow/ConnectSubscriptionForm";

```

```

export default function SubscriptionsPage() {
  const [statusFilter, setStatusFilter] = useState("all");
  const [showForm, setShowForm] = useState(false);
  const [modalMode, setModalMode] = useState(null);
  const [subscriptions, setSubscriptions] = useState([]);

  const fetchSubscriptions = async () => {
    try {
      const token = localStorage.getItem("token");
      if (!token) return;

      const response = await
fetch("http://localhost:8080/subscriptions", {
  headers: {
    Authorization: `Bearer ${token}`,
  },
});

      if (!response.ok) throw new Error("Ошибка загрузки
подписок");

      const data = await response.json();

      const formatted = data.map((sub) => ({
        ...sub, // сохраняем всё как есть
        price: sub.amount, // для совместимости с UI
        nextDate: sub.payment_date?.split("T")[0] || "", //
добавляем для UI
      }));
      setSubscriptions(formatted);
    } catch (error) {
      console.error("Ошибка:", error);
    }
  };

  useEffect(() => {
    fetchSubscriptions();
  }, []);

  const handleAdd = (data) => {
    const newSub = {
      id: Date.now(),
      name: data.name,
      nextDate: data.date,
      price: Number(data.price),
      status: "active",
      logo: "", // позже из data.avatar
      color: data.color,
    };
  };

```

```

    };
    setSubscriptions((arr) => [...arr, newSub]);
    setShowForm(false);
  };

  const handleChangeStatus = async (subId, newStatus) => {
    const sub = subscriptions.find((s) => s.id === subId);
    if (!sub) return;

    try {
      const token = localStorage.getItem("token");
      const updated = {
        ...sub,
        status: newStatus,
      };

      const response = await
fetch("http://localhost:8080/subscriptions/update", {
  method: "PUT",
  headers: {
    "Content-Type": "application/json",
    Authorization: `Bearer ${token}`,
  },
  body: JSON.stringify(updated),
});

      if (!response.ok) throw new Error("Не удалось изменить
статус");

      // 🔄 Перезагружаем подписки после успешного обновления
      await fetchSubscriptions();
    } catch (err) {
      console.error("Ошибка при обновлении статуса:", err);
      alert("Ошибка при обновлении статуса подписки");
    }
  };

  const filtered = subscriptions.filter(
    (s) => statusFilter === "all" || s.status === statusFilter
  );

  // вызывается из choice -> create
  const handleCreate = async () => {
    setModalMode(null);
    await fetchSubscriptions();
  };

  // вызывается из choice -> connect
  const handleConnect = (code) => {
    // TODO: API connect by code
    console.log("Connecting with code", code);
    setModalMode(null);
  };

```

```

};

return (
  <div className="max-w-4xl mx-auto p-4 sm:p-6">
    <h1 className="text-3xl font-bold text-center text-blue-600
mb-4">
      Подписки
    </h1>

    <div className="flex items-center mb-4 space-x-3">
      { /* Селект */ }
      <select
        value={statusFilter}
        onChange={(e) => setStatusFilter(e.target.value)}
        className="border rounded-md p-2 bg-white text-sm"
      >
        <option value="all">Все</option>
        <option value="active">Активные</option>
        <option value="inactive">Приостановленные</option>
      </select>

      { /* Компактная кнопка */ }
      <button
        onClick={() => setModalMode("choice")}
        className="
          inline-flex
          flex-none           /* запрещаем растягиваться */
          w-auto              /* авто-ширина по содержимому */
          items-center
          gap-1
          px-4
          py-2
          bg-green-600
          text-white
          rounded
          hover:bg-green-700
          text-sm
        "
      >
        <span className="text-lg font-bold leading-none">+
</span>
        <span>Добавить</span>
      </button>

    </div>

    {showForm && (
      <Modal onClose={() => setShowForm(false)}>
        <SubscriptionForm onSubmit={handleAdd} onCancel={() =>
setShowForm(false)} />
      </Modal>
    ) }
  )

```

```

<SubscriptionList
  title={
    statusFilter === "all"
      ? "Все подписки"
      : statusFilter === "active"
      ? "Активные подписки"
      : "Приостановленные подписки"
  }
  subscriptions={filtered}
  onChangeStatus={handleChangeStatus}
/>

{modalMode === "choice" && (
  <AddSubscriptionChoiceModal
    onCreateNew={() => setModalMode("create")}
    onConnectExisting={() => setModalMode("connect")}
    onClose={() => setModalMode(null)}
  />
)}

{modalMode === "create" && (
  <Modal onClose={() => setModalMode(null)}>
    <SubscriptionForm
      onSubmit={handleCreate}
      onCancel={() => setModalMode(null)}
    />
  </Modal>
)}

{modalMode === "connect" && (
  <Modal onClose={() => setModalMode(null)}>
    <ConnectSubscriptionForm
      onSubmit={handleConnect}
      onCancel={() => setModalMode(null)}
    />
  </Modal>
)}
</div>
);
}

```

Листинг В.8 – Код файла ProtectedRoute.js

```

import React, { useEffect, useState } from 'react';
import { Navigate } from 'react-router-dom';

const ProtectedRoute = ({ element, redirectTo }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(null); //
  null для состояния загрузки

  const validateToken = async (token) => {

```

```

    try {
      const response = await
fetch('http://localhost:8080/validation', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    Authorization: `Bearer ${token}`,
  },
});

    if (!response.ok) {
      throw new Error('Invalid token');
    }

    return true; // Token валиден
  } catch (error) {
    console.error(error);
    return false; // Token не валиден
  }
};

useEffect(() => {
  const checkAuth = async () => {
    const token = localStorage.getItem('token'); // Получение
токена из localStorage или другого источника
    if (token) {
      const isValid = await validateToken(token); // Валидация
токена
      setIsAuthenticated(isValid);
    } else {
      setIsAuthenticated(false);
    }
  };

  checkAuth();
}, []);

if (isAuthenticated === null) {
  return <div>Проверка авторизации...</div>; // Спиннер или
индикатор загрузки
}

return isAuthenticated ? element : <Navigate to={redirectTo} />;
};

export default ProtectedRoute;

```

Приложение Г
(рекомендуемое)

Результаты функционального тестирования

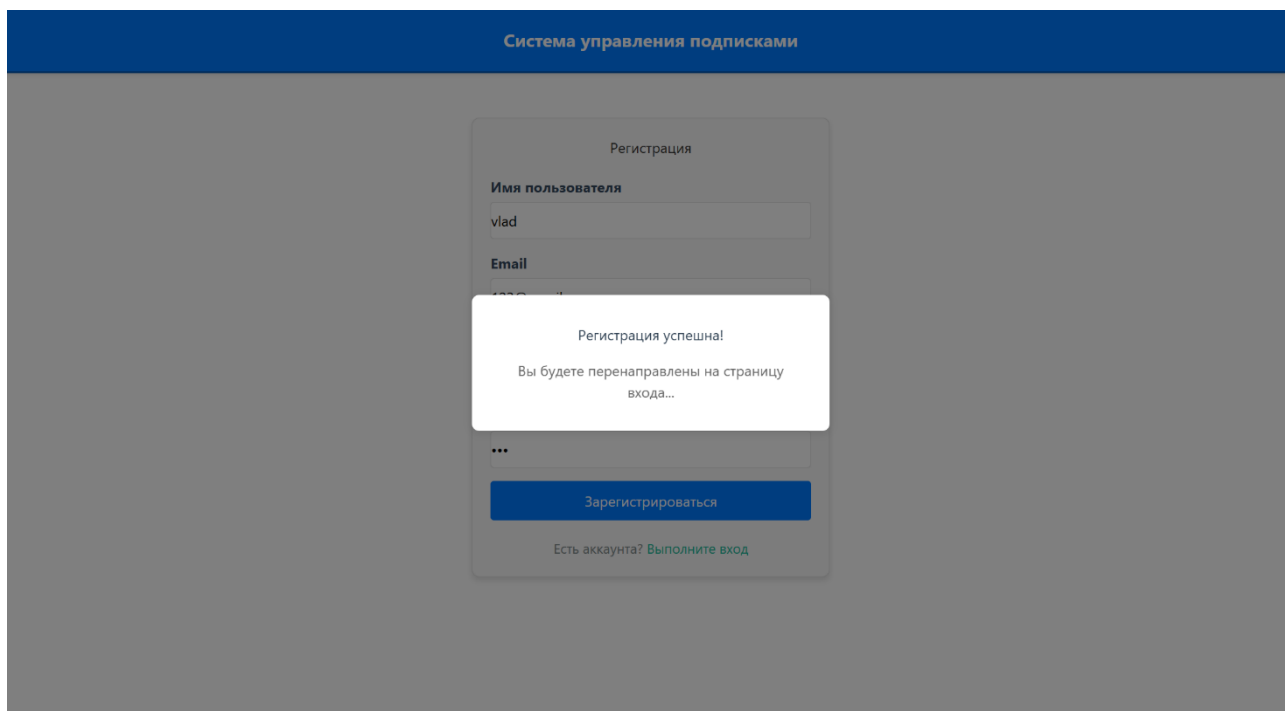


Рисунок Г.1 – Регистрация нового пользователя

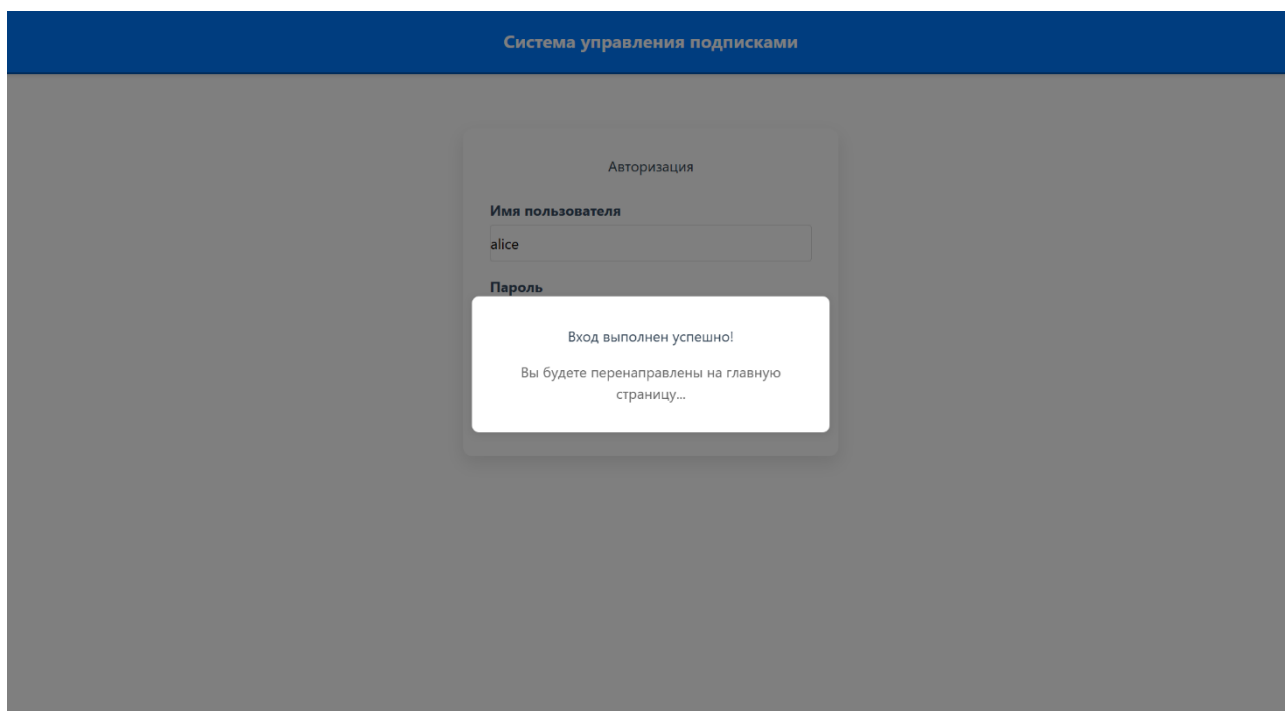


Рисунок Г.2 – Вход пользователем в систему

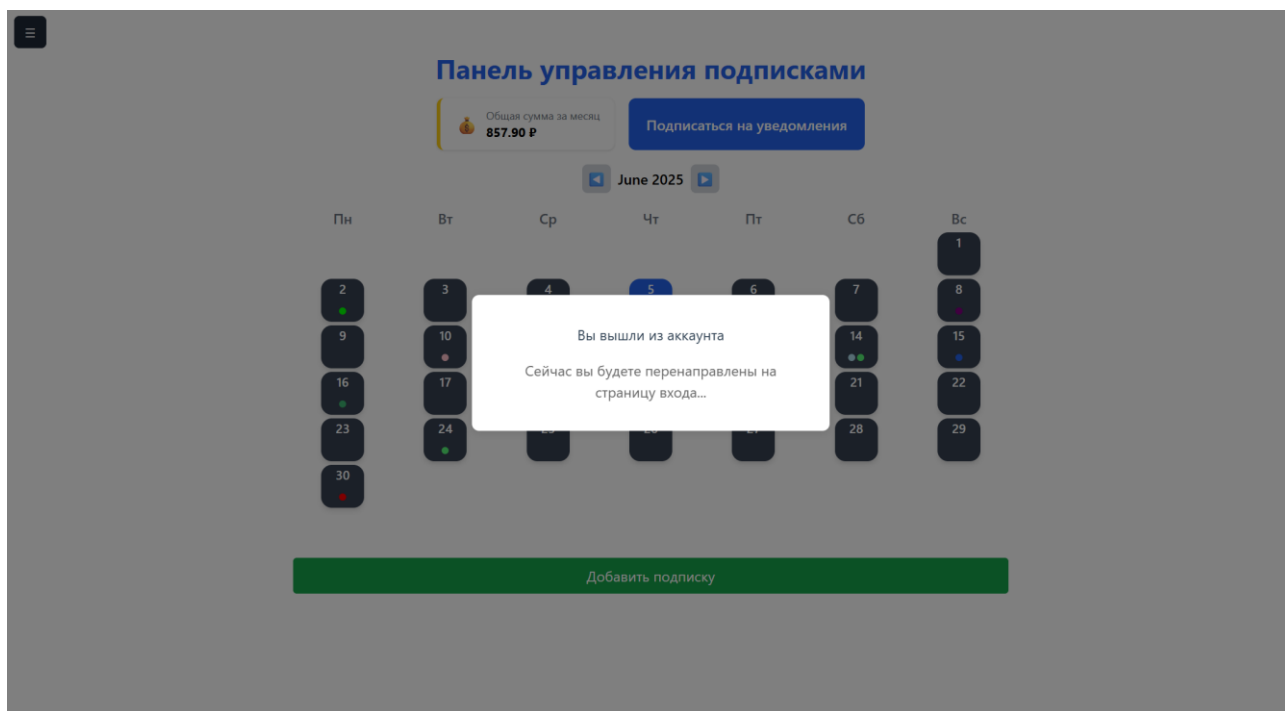


Рисунок Г.3 – Выход пользователя из системы

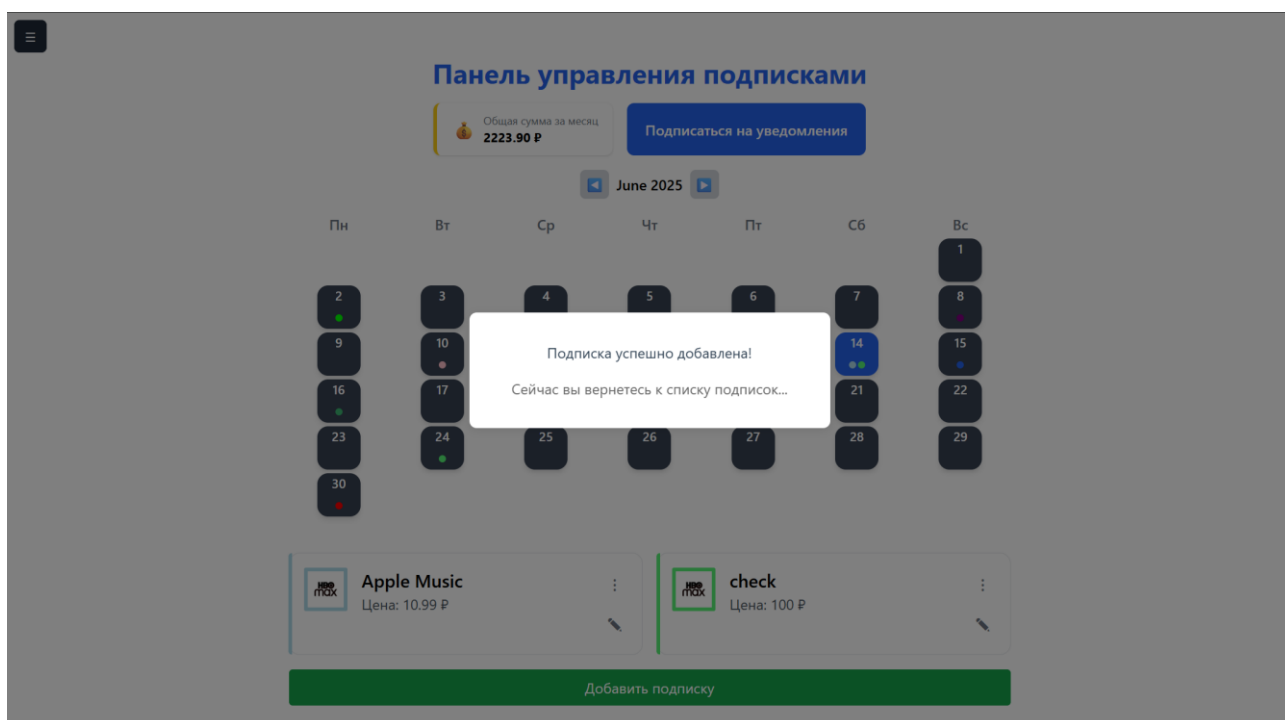


Рисунок Г.4 – Добавление платной подписки

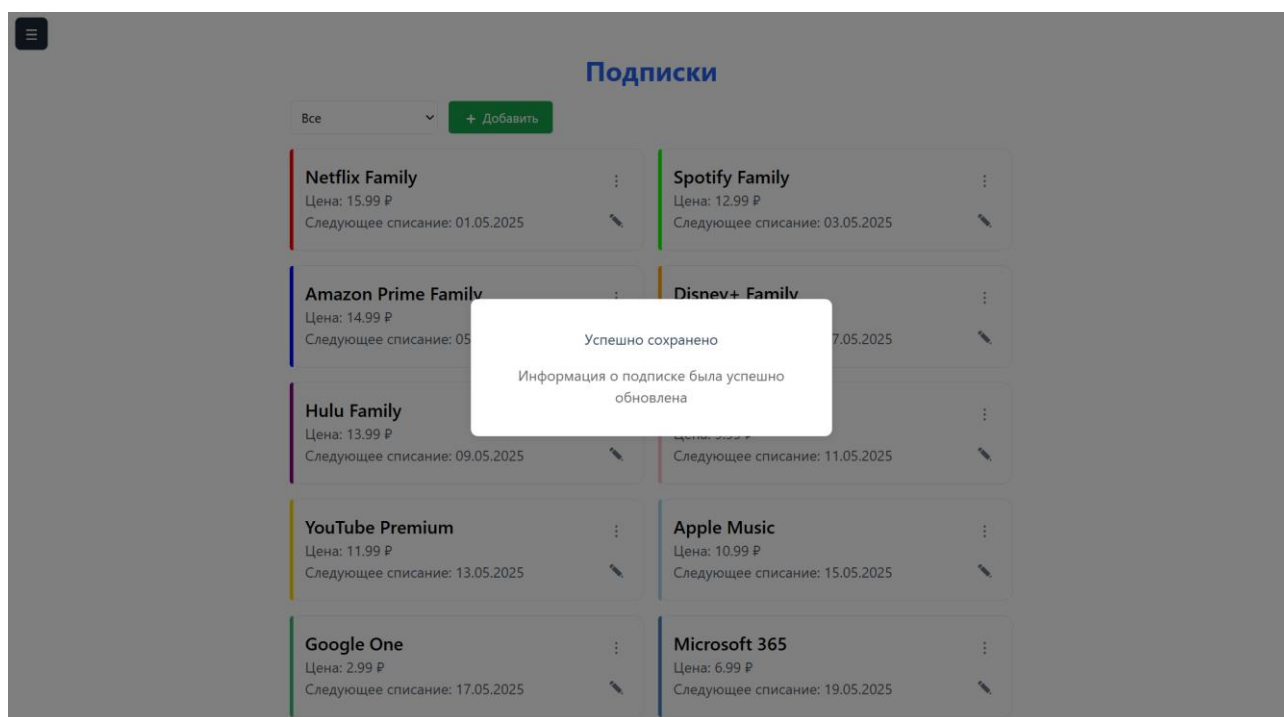


Рисунок Г.5 – Редактирование платной подписки

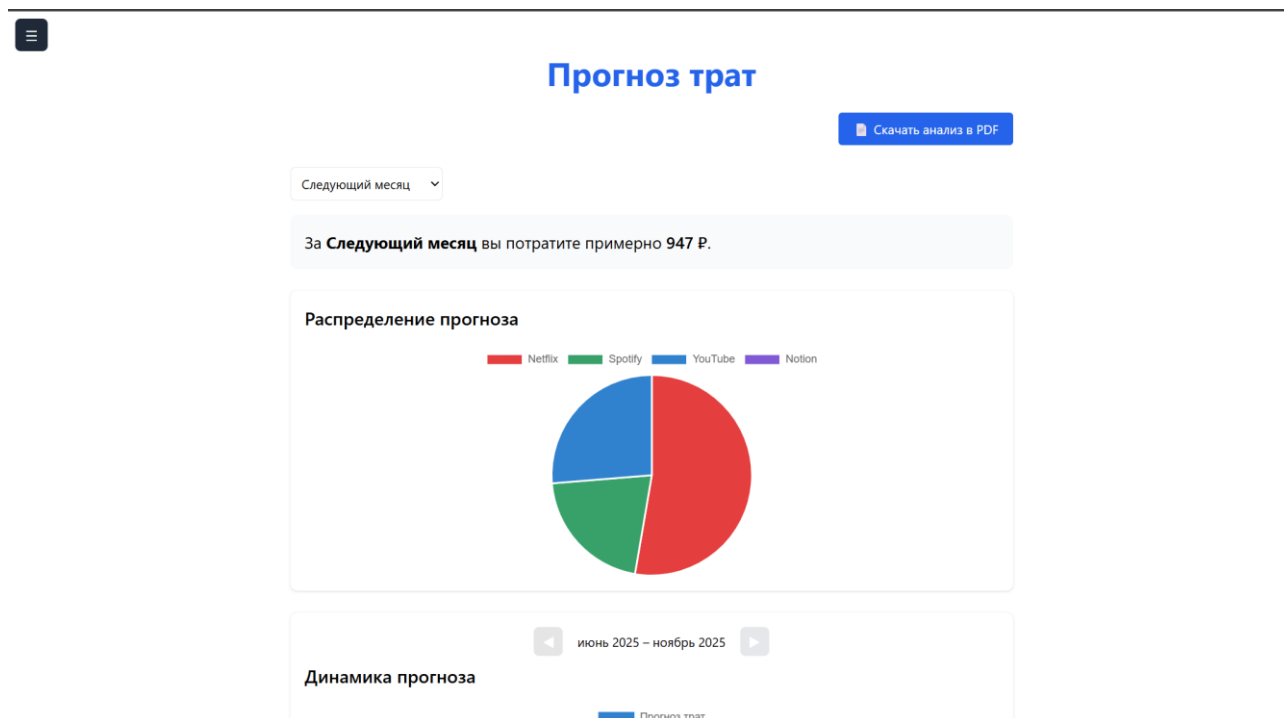


Рисунок Г.6 – Вывод прогноза трат на платные подписки

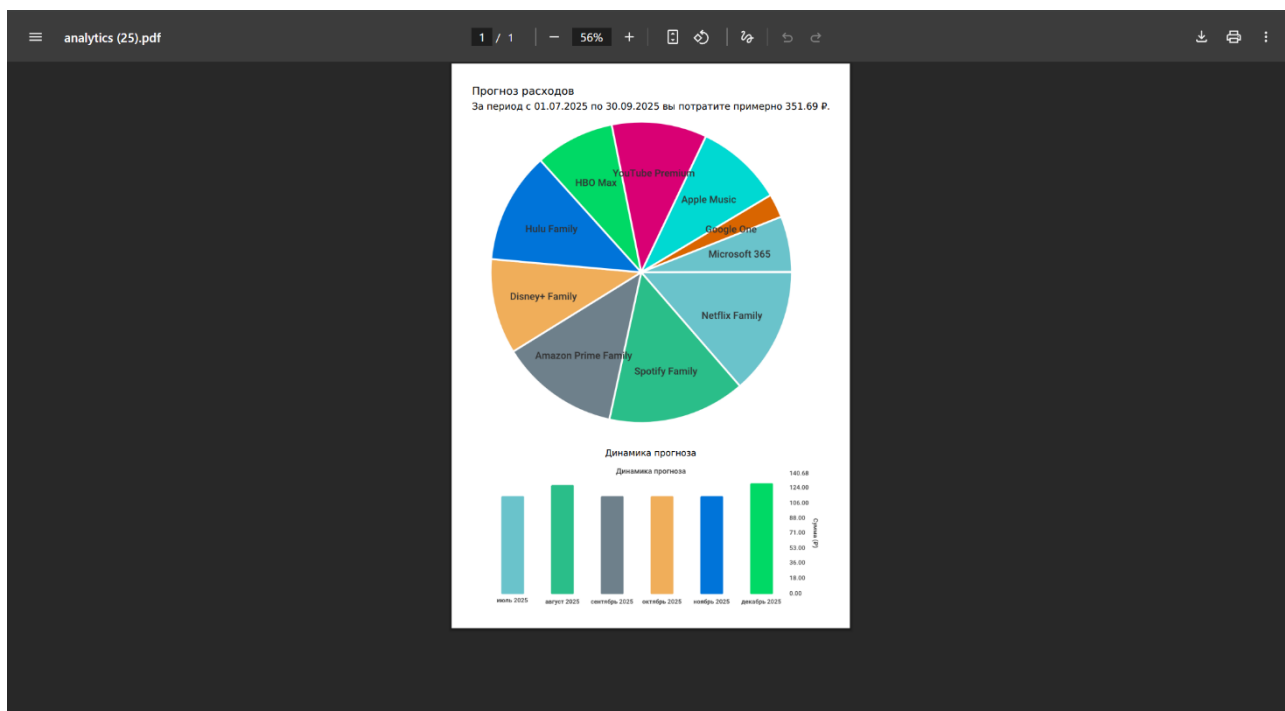


Рисунок Г.7 – Скачивание pdf-файла с прогнозом трат на платные подписки

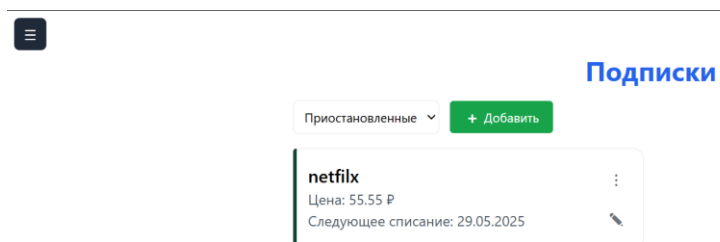


Рисунок Г.8 – Фильтрация платных подписок по статусу

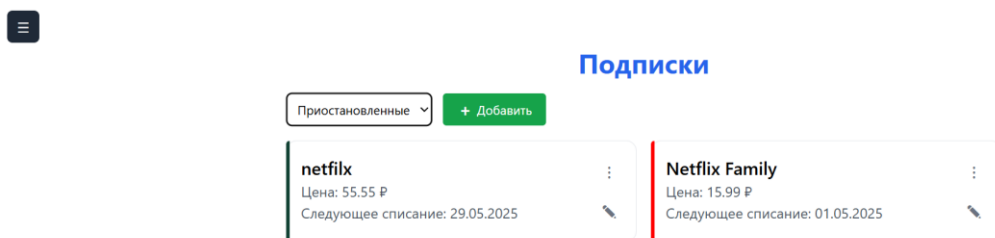


Рисунок Г.9 – Приостановка подписки

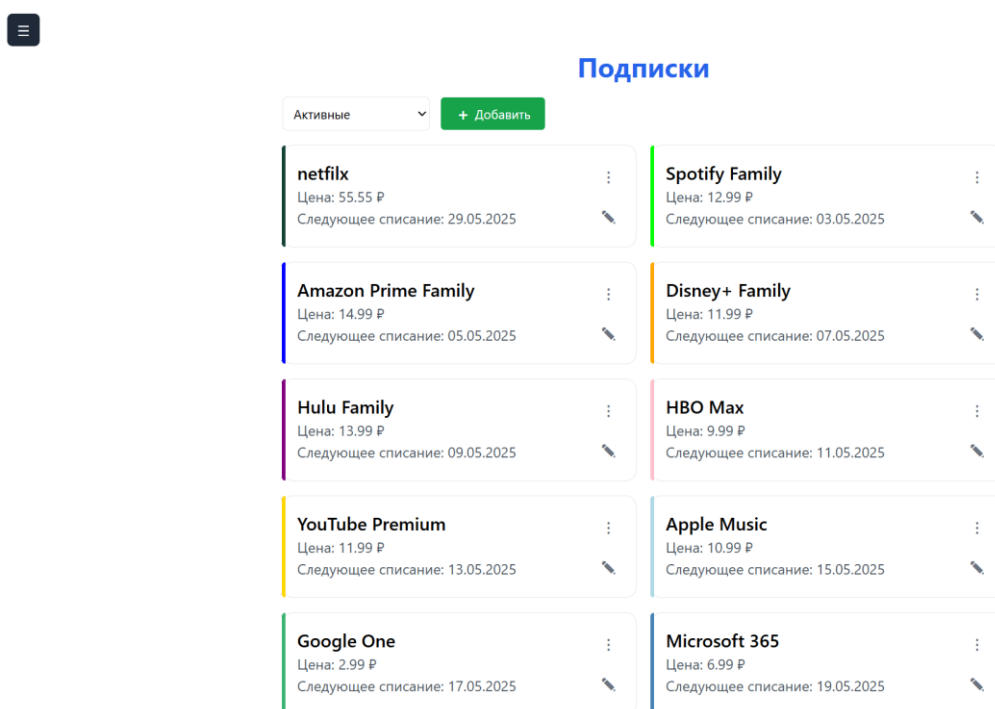


Рисунок Г.10 – Возобновление подписки

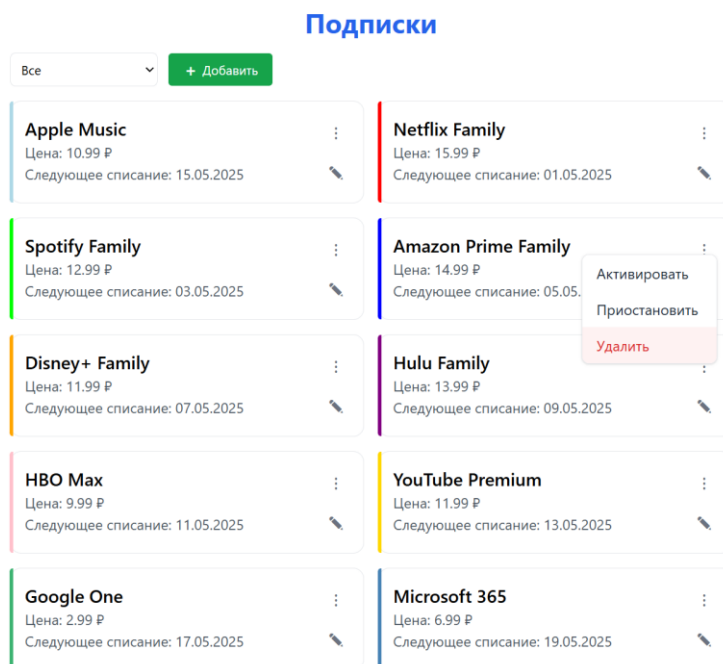


Рисунок Г.11 – Удаление подписки

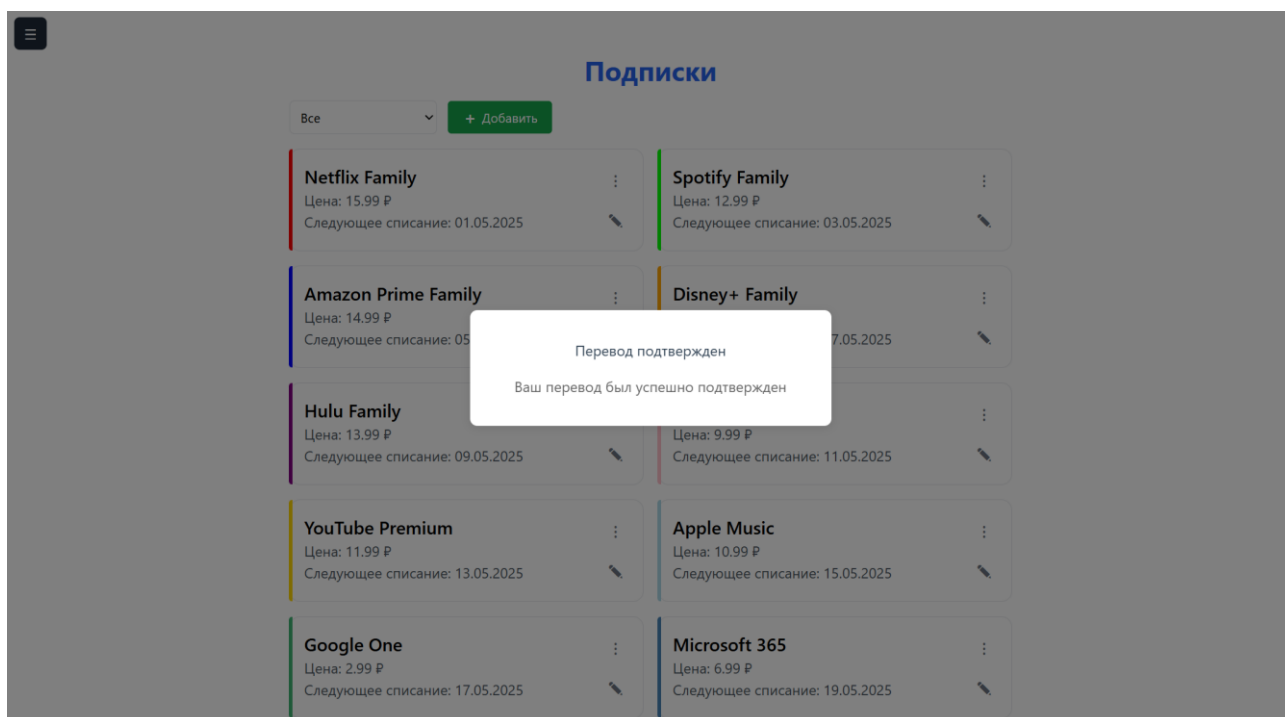


Рисунок Г.12 – Подтверждение перевода денежных средств в групповой подписке

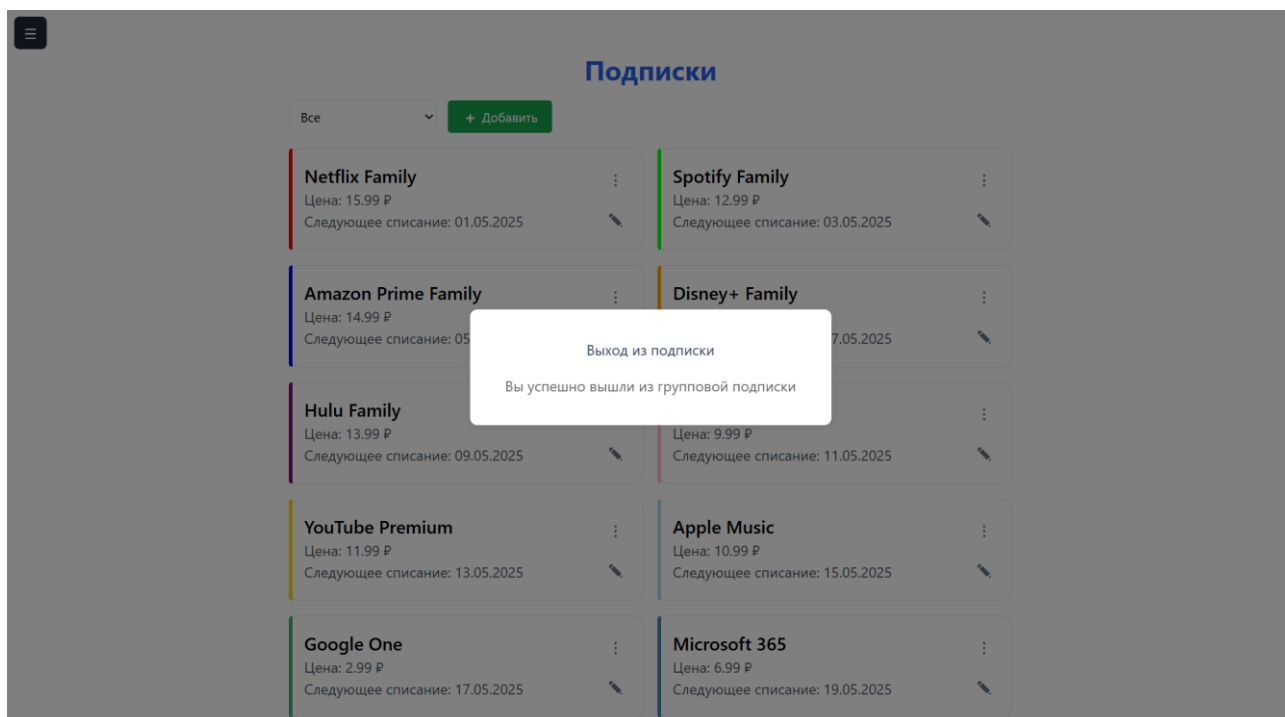


Рисунок Г.13 – Выход из групповой подписки

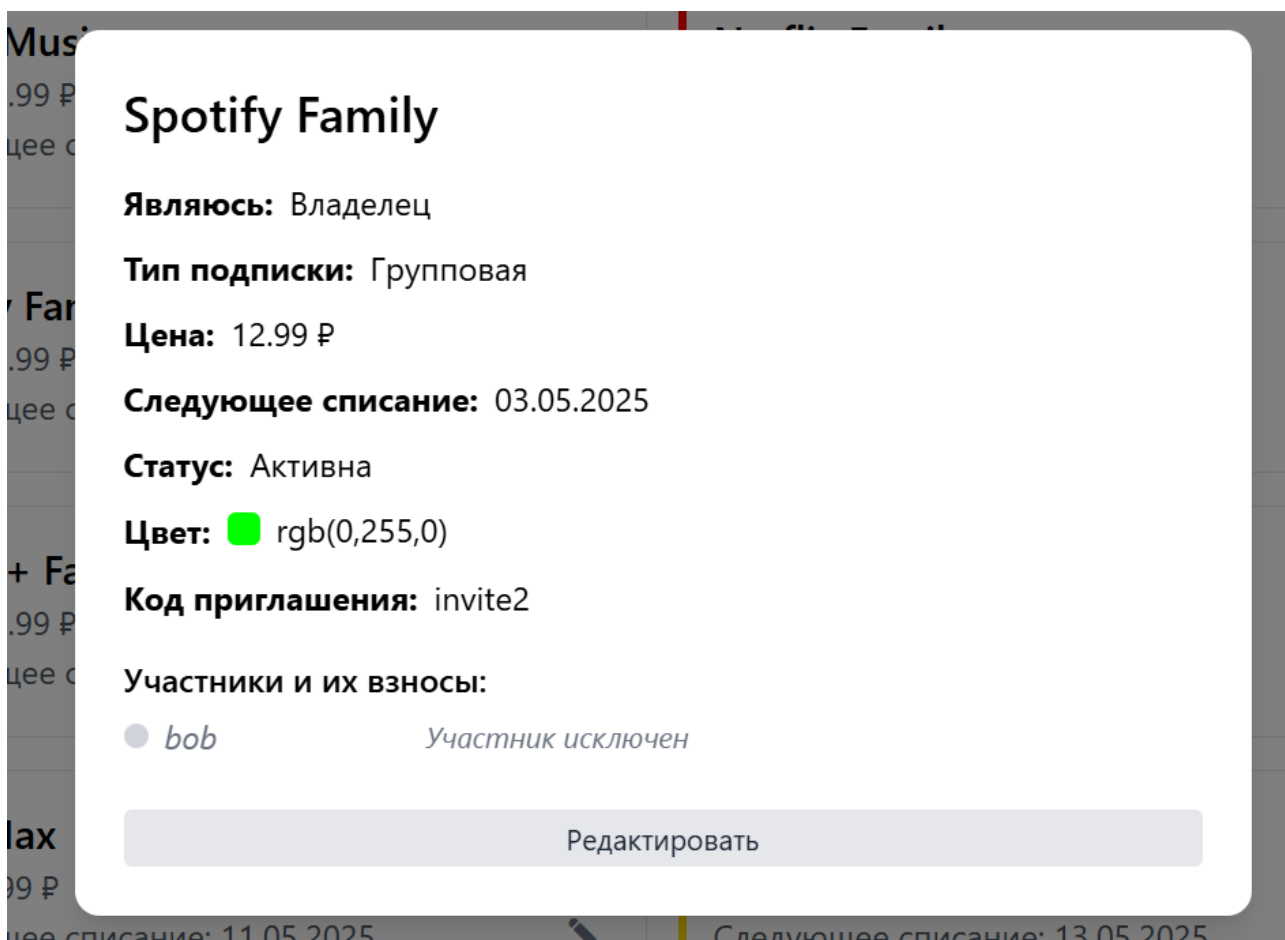


Рисунок Г.14 – Удаление участник групповой подписки

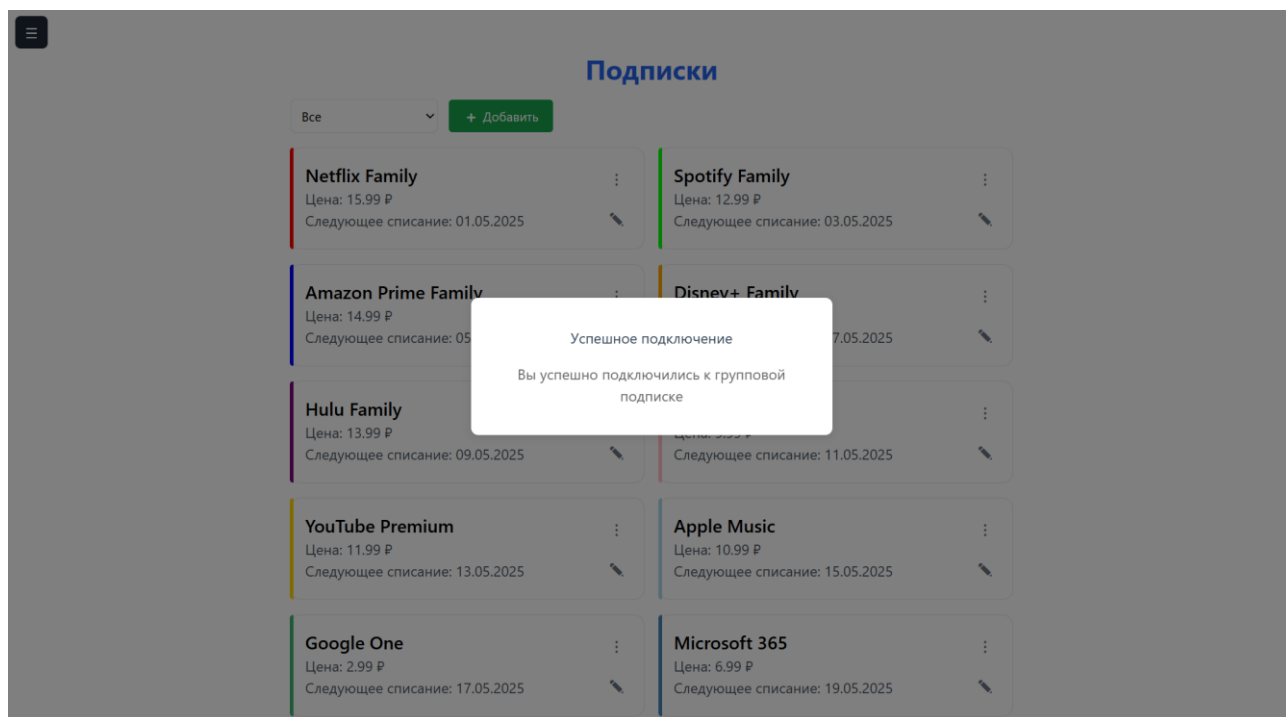


Рисунок Г.15 – Присоединение к групповой подписке

Приложение Д
(рекомендуемое)
Код модульных тестов

Листинг Д.1 – Файл subscription_member_service.go

```

package service

import (
    "testing"
    "time"

    "github.com/stretchr/testify/assert"
    "github.com/stretchr/testify/mock"

    "subscription-service/internal/models"
    "subscription-service/internal/models/request"
    "subscription-service/internal/models/response"
    "subscription-service/internal/repository"
)

// -----
// 1. Тест Create (CreateSubscriptionMember)
// -----

func TestCreateSubscriptionMember(t *testing.T) {
    repo := new(MockSubscriptionMemberRepo)
    subSvc := new(MockSubService)
    log := new(MockLogger)

    svc := NewSubscriptionMemberService(&repository.Repository{
        SubscriptionMember: repo,
    }, subSvc, log)

    input := models.SubscriptionMember{
        UserId:           1,
        SubscriptionId:   1,
        Amount:           0,
        Joined_at:        time.Now(),
        DaysNotifyBeforePayment: 5,
        Color:            "blue",
        Payment_status:   models.NotPaid,
    }

    repo.On("Create", input).Return(42, nil)

    id, err := svc.Create(input)
    assert.NoError(t, err)
    assert.Equal(t, 42, id)

    repo.AssertExpectations(t)
}

// -----
// 2. Тест GetAllMembersInSubs
// -----

```

```

func TestGetAllMembersInSubs(t *testing.T) {
    repo := new(MockSubscriptionMemberRepo)
    subSvc := new(MockSubService)
    log := new(MockLogger)

    svc := NewSubscriptionMemberService(&repository.Repository{
        SubscriptionMember: repo,
    }, subSvc, log)

    mockUsers := []response.User{
        {Id: 1, Login: "user1", Amount: 10.0, Email:
"u1@example.com"},
        {Id: 2, Login: "user2", Amount: 20.0, Email:
"u2@example.com"},
    }

    repo.On("GetMembersBySubId", uint(1)).Return(mockUsers, nil)

    users, err := svc.GetAllMembersInSubs(1)
    assert.NoError(t, err)
    assert.Len(t, users, 2)
    assert.Equal(t, mockUsers, users)

    repo.AssertExpectations(t)
}

// -----
// 3. Test ChangeAmountMembers
// -----

func TestChangeAmountMembers(t *testing.T) {
    repo := new(MockSubscriptionMemberRepo)
    subSvc := new(MockSubService)
    log := new(MockLogger)

    svc := NewSubscriptionMemberService(&repository.Repository{
        SubscriptionMember: repo,
    }, subSvc, log)

    subID := models.SubscriptionId{Id: 1}
    req := request.AmountMember{
        SubscriptionId: subID,
        UserId:         1,
        Amount:         150,
    }

    subMem := models.SubscriptionMember{
        UserId:         1,
        SubscriptionId: 1,
        Amount:         0,
        Joined_at:      time.Now(),
        Color:          "red",
    }

```

```

        Payment_status: models.NotPaid,
    }

    repo.On("GetByUserIdAndSubId", uint(1),
uint(1)).Return(subMem, nil)
    repo.On("Update",
mock.AnythingOfType("models.SubscriptionMember")).Return(nil)

    err := svc.ChangeAmountMembers(req)
    assert.NoError(t, err)

    repo.AssertExpectations(t)
}

// -----
// 4. Тест ChangePaymentStatusMember
// -----

func TestChangePaymentStatusMember(t *testing.T) {
    repo := new(MockSubscriptionMemberRepo)
    subSvc := new(MockSubService)
    log := new(MockLogger)

    svc := NewSubscriptionMemberService(&repository.Repository{
        SubscriptionMember: repo,
    }, subSvc, log)

    subID := models.SubscriptionId{Id: 1}
    req := request.PaymentStatusChangeRequest{
        SubscriptionId: subID,
        PaymentStatus: models.Paid,
    }

    subMem := models.SubscriptionMember{
        UserId:      1,
        SubscriptionId: 1,
        Amount:      0,
        Joined_at:    time.Now(),
        Color:        "green",
        Payment_status: models.NotPaid,
    }

    repo.On("GetByUserIdAndSubId", uint(1),
uint(1)).Return(subMem, nil)
    repo.On("Update",
mock.AnythingOfType("models.SubscriptionMember")).Return(nil)

    err := svc.ChangePaymentStatusMember(1, req)
    assert.NoError(t, err)

    repo.AssertExpectations(t)
}

```

```

// -----
// 5. Тест ExistUserInGroupSubscription
// -----

func TestExistUserInGroupSubscription(t *testing.T) {
    repo := new(MockSubscriptionMemberRepo)
    subSvc := new(MockSubService)
    log := new(MockLogger)

    svc := NewSubscriptionMemberService(&repository.Repository{
        SubscriptionMember: repo,
    }, subSvc, log)

    // Мокаем, что пользователь с ID=1 уже есть в подписке с ID=10
    repo.On("ExistUserInGroupSubscription", uint(1),
uint(10)).Return(true, nil)

    exists, err := svc.ExistUserInGroupSubscription(1, 10)
    assert.NoError(t, err)
    assert.True(t, exists)

    repo.AssertExpectations(t)
}

// -----
// 6. Тест DeleteMemberFromSubscription (успех)
// -----

func TestDeleteMemberFromSubscription_Success(t *testing.T) {
    repo := new(MockSubscriptionMemberRepo)
    subSvc := new(MockSubService)
    log := new(MockLogger)

    svc := NewSubscriptionMemberService(&repository.Repository{
        SubscriptionMember: repo,
    }, subSvc, log)

    req := request.DeleteMember{
        UserId:      2,
        OwnerId:     10,
        SubscriptionId: models.SubscriptionId{Id: 1},
    }

    // Мокируем: subSvc.Get(1) вернёт модель с OwnerId=10
    subSvc.On("Get", uint(1)).Return(models.Subscription{OwnerId:
10}, nil)
    // Мокируем: repo.Delete(2,1) вернёт nil
    repo.On("Delete", uint(2), uint(1)).Return(nil)

    err := svc.DeleteMemberFromSubscription(req)
    assert.NoError(t, err)
}

```

```

    subSvc.AssertExpectations(t)
    repo.AssertExpectations(t)
}

```

Листинг Д.2 – Файл subscription_service_test.go

```

package service

import (
    "errors"
    "testing"

    "github.com/stretchr/testify/assert"

    "subscription-service/internal/models"
    "subscription-service/internal/repository"
)

func TestCreateSubscription_Success(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    sub := models.Subscription{Id: 1, Name: "Test"}

    repo.On("Create", sub).Return(42, nil)

    id, err := svc.Create(sub)
    assert.NoError(t, err)
    assert.Equal(t, 42, id)

    repo.AssertExpectations(t)
}

func TestCreateSubscription_Error(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    sub := models.Subscription{Id: 1, Name: "Test"}

    repo.On("Create", sub).Return(0, errors.New("db error"))

    id, err := svc.Create(sub)
    assert.Error(t, err)
    assert.Equal(t, 0, id)
}

```

```

    repo.AssertExpectations(t)
}

func TestUpdateSubscription_Success(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    sub := models.Subscription{Id: 1, Name: "Old"}
    updated := models.Subscription{Id: 1, Name: "New"}

    repo.On("Update", sub).Return(updated, nil)

    res, err := svc.Update(sub)
    assert.NoError(t, err)
    assert.Equal(t, updated, res)

    repo.AssertExpectations(t)
}

func TestUpdateSubscription_Error(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    sub := models.Subscription{Id: 1, Name: "Old"}

    repo.On("Update", sub).Return(models.Subscription{},
errors.New("update failed"))

    res, err := svc.Update(sub)
    assert.Error(t, err)
    assert.Empty(t, res)

    repo.AssertExpectations(t)
}

func TestGetAllByStatus_Success(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    status := models.Active
    subs := []models.Subscription{
        {Id: 1, Name: "A"},
        {Id: 2, Name: "B"},
    }

```

```

    }

    repo.On("GetAllByStatus", status).Return(subs, nil)

    res, err := svc.GetAllByStatus(status)
    assert.NoError(t, err)
    assert.Equal(t, subs, res)

    repo.AssertExpectations(t)
}

func TestGetAllByStatus_Error(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    status := models.InActive

    repo.On("GetAllByStatus", status).Return(nil, errors.New("not
found"))

    res, err := svc.GetAllByStatus(status)
    assert.Error(t, err)
    assert.Nil(t, res)

    repo.AssertExpectations(t)
}

func TestGetSubscription_Success(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    sub := models.Subscription{Id: 1, Name: "Test"}

    repo.On("Get", uint(1)).Return(sub, nil)

    res, err := svc.Get(1)
    assert.NoError(t, err)
    assert.Equal(t, sub, res)

    repo.AssertExpectations(t)
}

func TestGetSubscription_Error(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{

```

```

        Subscription: repo,
    }, log)

    repo.On("Get", uint(2)).Return(models.Subscription{},
errors.New("not exist"))

    res, err := svc.Get(2)
    assert.Error(t, err)
    assert.Empty(t, res)

    repo.AssertExpectations(t)
}

func TestGetByInviteCode_Success(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    token := "abc"

    sub := models.Subscription{Id: 1, InviteToken: &token}

    repo.On("GetByInviteCode", "abc").Return(sub, nil)

    res, err := svc.GetByInviteCode("abc")
    assert.NoError(t, err)
    assert.Equal(t, sub, res)

    repo.AssertExpectations(t)
}

func TestGetByInviteCode_Error(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    repo.On("GetByInviteCode",
"xyz").Return(models.Subscription{}, errors.New("not found"))

    res, err := svc.GetByInviteCode("xyz")
    assert.Error(t, err)
    assert.Empty(t, res)

    repo.AssertExpectations(t)
}

func TestGetAllByUserId_Success(t *testing.T) {
    repo := new(MockSubscriptionRepo)

```

```

    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    subs := []models.Subscription{
        {Id: 1, Name: "A"},
        {Id: 2, Name: "B"},
    }

    repo.On("GetAllByUser", uint(5)).Return(subs, nil)

    res, err := svc.GetAllByUserId(5)
    assert.NoError(t, err)
    assert.Equal(t, subs, res)

    repo.AssertExpectations(t)
}

func TestGetAllByUserId_Error(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    repo.On("GetAllByUser", uint(5)).Return(nil, errors.New("db
error"))

    res, err := svc.GetAllByUserId(5)
    assert.Error(t, err)
    assert.Nil(t, res)

    repo.AssertExpectations(t)
}

func TestGetByUserAndStatus_Success(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    subs := []models.Subscription{
        {Id: 3, Name: "X"},
    }

    repo.On("GetByUserAndStatus", uint(5),
models.Active).Return(subs, nil)

    res, err := svc.GetByUserAndStatus(5, models.Active)
    assert.NoError(t, err)

```

```

    assert.Equal(t, subs, res)

    repo.AssertExpectations(t)
}

func TestGetByUserAndStatus_Error(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    repo.On("GetByUserAndStatus", uint(5),
models.InActive).Return(nil, errors.New("error"))

    res, err := svc.GetByUserAndStatus(5, models.InActive)
    assert.Error(t, err)
    assert.Nil(t, res)

    repo.AssertExpectations(t)
}

func TestDeleteSubscription_Success(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    // Ожидаем, что вызов repo.Delete(5) вернёт nil
    repo.On("Delete", uint(5)).Return(nil)

    err := svc.Delete(5)
    assert.NoError(t, err)

    repo.AssertExpectations(t)
}

func TestDeleteSubscription_Error(t *testing.T) {
    repo := new(MockSubscriptionRepo)
    log := new(MockLogger)
    svc := NewSubscriptionService(&repository.Repository{
        Subscription: repo,
    }, log)

    // Ожидаем, что вызов repo.Delete(7) вернёт ошибку
    repo.On("Delete", uint(7)).Return(errors.New("delete failed"))

    err := svc.Delete(7)
    assert.Error(t, err)

    repo.AssertExpectations(t)
}

```

```
}
```

Листинг Д.3 – Файл user_service_test.go

```
package service

import (
    "errors"
    "testing"
    "time"

    "github.com/stretchr/testify/assert"

    "subscription-service/internal/models"
    "subscription-service/internal/repository"
)

func TestCreateUser_Success(t *testing.T) {
    repo := new(MockUserRepo)
    log := new(MockLogger)
    svc := NewUserService(&repository.Repository{
        User: repo,
    }, log)

    // Используем реальные поля модели User
    birth := "2000-01-01"
    created := time.Now()
    chatId := int64(12345)
    user := models.User{
        Id:          1,
        Login:       "testuser",
        Password:    "plainpass",
        HashPassword: "hashedpass",
        Email:       "test@example.com",
        DateBirthday: birth,
        Created_at:  created,
        ChatId:      &chatId,
    }

    repo.On("Create", user).Return(100, nil)

    id, err := svc.Create(user)
    assert.NoError(t, err)
    assert.Equal(t, 100, id)

    repo.AssertExpectations(t)
}

func TestCreateUser_Error(t *testing.T) {
    repo := new(MockUserRepo)
    log := new(MockLogger)
    svc := NewUserService(&repository.Repository{
        User: repo,
    }, log)
```

```

    }, log)

    user := models.User{
        Id:          2,
        Login:       "testuser2",
        Password:    "pass2",
        HashPassword: "hash2",
        Email:       "u2@example.com",
        DateBirthDay: "1990-05-10",
        Created_at:  time.Now(),
        ChatId:      nil,
    }

    repo.On("Create", user).Return(0, errors.New("db error"))

    id, err := svc.Create(user)
    assert.Error(t, err)
    assert.Equal(t, 0, id)

    repo.AssertExpectations(t)
}

func TestUpdateUser_Success(t *testing.T) {
    repo := new(MockUserRepo)
    log := new(MockLogger)
    svc := NewUserService(&repository.Repository{
        User: repo,
    }, log)

    original := models.User{
        Id:          3,
        Login:       "old",
        Password:    "oldpass",
        HashPassword: "oldhash",
        Email:       "old@example.com",
        DateBirthDay: "1985-12-12",
        Created_at:  time.Now(),
        ChatId:      nil,
    }
    updated := models.User{
        Id:          3,
        Login:       "new",
        Password:    "newpass",
        HashPassword: "newhash",
        Email:       "new@example.com",
        DateBirthDay: "1985-12-12",
        Created_at:  original.Created_at,
        ChatId:      nil,
    }

    repo.On("Update", original).Return(updated, nil)

```

```

    res, err := svc.Update(original)
    assert.NoError(t, err)
    assert.Equal(t, updated, res)

    repo.AssertExpectations(t)
}

func TestUpdateUser_Error(t *testing.T) {
    repo := new(MockUserRepo)
    log := new(MockLogger)
    svc := NewUserService(&repository.Repository{
        User: repo,
    }, log)

    user := models.User{
        Id:          4,
        Login:       "fail",
        Password:    "pass",
        HashPassword: "hash",
        Email:       "fail@example.com",
        DateBirthDay: "1970-07-07",
        Created_at:  time.Now(),
        ChatId:      nil,
    }

    repo.On("Update", user).Return(models.User{},
errors.New("update failed"))

    res, err := svc.Update(user)
    assert.Error(t, err)
    assert.Equal(t, models.User{}, res)

    repo.AssertExpectations(t)
}

func TestGetUser_Success(t *testing.T) {
    repo := new(MockUserRepo)
    log := new(MockLogger)
    svc := NewUserService(&repository.Repository{
        User: repo,
    }, log)

    user := models.User{
        Id:          5,
        Login:       "get",
        Password:    "pass",
        HashPassword: "hash",
        Email:       "get@example.com",
        DateBirthDay: "2001-03-03",
        Created_at:  time.Now(),
        ChatId:      nil,
    }
}

```

```

    repo.On("Get", uint(5)).Return(user, nil)

    res, err := svc.Get(5)
    assert.NoError(t, err)
    assert.Equal(t, user, res)

    repo.AssertExpectations(t)
}

func TestGetUser_Error(t *testing.T) {
    repo := new(MockUserRepo)
    log := new(MockLogger)
    svc := NewUserService(&repository.Repository{
        User: repo,
    }, log)

    repo.On("Get", uint(6)).Return(models.User{}, errors.New("not
found"))

    res, err := svc.Get(6)
    assert.Error(t, err)
    assert.Equal(t, models.User{}, res)

    repo.AssertExpectations(t)
}

func TestGetByLogin_Success(t *testing.T) {
    repo := new(MockUserRepo)
    log := new(MockLogger)
    svc := NewUserService(&repository.Repository{
        User: repo,
    }, log)

    user := models.User{
        Id:          7,
        Login:        "loginUser",
        Password:     "pass",
        HashPassword: "hash",
        Email:        "login@example.com",
        DateBirthday: "1995-11-11",
        Created_at:   time.Now(),
        ChatId:       nil,
    }

    repo.On("GetByLogin", "loginUser").Return(user, nil)

    res, err := svc.GetByLogin("loginUser")
    assert.NoError(t, err)
    assert.Equal(t, user, res)

    repo.AssertExpectations(t)
}

```

```

}

func TestGetByLogin_Error(t *testing.T) {
    repo := new(MockUserRepo)
    log := new(MockLogger)
    svc := NewUserService(&repository.Repository{
        User: repo,
    }, log)

    repo.On("GetByLogin", "unknown").Return(models.User{},
errors.New("not found"))

    res, err := svc.GetByLogin("unknown")
    assert.Error(t, err)
    assert.Equal(t, models.User{}, res)

    repo.AssertExpectations(t)
}

func TestDeleteUser_Success(t *testing.T) {
    repo := new(MockUserRepo)
    log := new(MockLogger)
    svc := NewUserService(&repository.Repository{
        User: repo,
    }, log)

    repo.On("Delete", 9).Return(nil)

    err := svc.Delete(9)
    assert.NoError(t, err)

    repo.AssertExpectations(t)
}

func TestDeleteUser_Error(t *testing.T) {
    repo := new(MockUserRepo)
    log := new(MockLogger)
    svc := NewUserService(&repository.Repository{
        User: repo,
    }, log)

    repo.On("Delete", 10).Return(errors.New("delete failed"))

    err := svc.Delete(10)
    assert.Error(t, err)

    repo.AssertExpectations(t)
}

```