МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

«Программная реализация web-приложения системы учета абонентских платежей для городской телефонной сети»

Студент _____ Гречихин Данила Андреевич
(подпись, дата) (ФИО полностью)

Руководитель _____ _____ Кольчугина Е.А. _____
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

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

Заведующий кафедрой _____ __ Козлов А.Ю. __
(подпись) (фамилия, инициалы)

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

Секретарь ГЭК _____ _ Попова Н.А. ___
(подпись) (фамилия, инициалы)

Пенза, 2023

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

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«__»_____ _2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

БАКАЛАВРА

1. Студент_____Гречихин Данила Андреевич_____гр._19ВП2_факультета___ВТ_____
<Фамилия, имя, отчество полностью>

2.Тема работы_«Программная реализация web-приложения системы учета абонентских платежей для городской телефонной сети»_____

Тема утверждена приказом ПГУ №_____от "_____"_____20_____

3. Руководитель работы_Кольчугина Е.А._____

4. Задание на работу (назначение разработки, исходные данные и т.п.)_____
_____Среда разработки: Windows 10, язык программирования C# 10.0 на платформе .NET 6, платформа для создания веб-приложений ASP .NET Core 6, ORM-система Entity Framework Core 6, движок пользовательских представлений Razor_____
_____Среда выполнения: кроссплатформенная на основе браузеров Google Chrome, Mozilla Firefox, Microsoft Edge_____
_____Разработать веб-приложение, которое позволяет вести учет абонентских платежей для городской телефонной сети_____
_____Основные функции веб-приложения:_____
_____– регистрация и учет пользователей;_____
_____– учет поступающих платежей;_____
_____– выявление задолжников;_____
_____– ведение списка услуг;_____
_____– учет объема предоставленных услуг_____
_____Технология разработки: ООП, MVC_____
_____Языки программирования: C# 10.0, HTML5, CCS3, JavaScript_____
_____

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области
2) Анализ функциональных требований
3) Планирование разработки и оценка бюджета
4) Проектирование web-приложения
5) Программная реализация
6) Контроль качества
7) Выполнение пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 21.05.2023	
5	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2022 г.

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 146 листов, 37 рисунков, 20 таблиц, 18 использованных источников, 16 приложений.

АНАЛИЗ, ПРОЕКТИРОВАНИЕ, РАЗРАБОТКА, ТЕСТИРОВАНИЕ, WEB-ПРИЛОЖЕНИЕ, ГОРОДСКАЯ ТЕЛЕФОННАЯ СЕТЬ, УЧЕТ ПЛАТЕЖЕЙ.

Объект разработки – web-приложение системы учета абонентских платежей для городской телефонной сети.

Цель работы – изучить предметную область и провести её анализ, выделить процессы, разработать и протестировать программный продукт.

Технология разработки: .NET 6, C# 10.0, ASP.NET Core 6, Entity Framework Core 6, Razor, HTML, CSS, Bootstrap, JavaScript, Microsoft SQL Server 2018, MSTest, Moq.

В результате выполнения работы было спроектировано и создано web-приложение, предназначенное для учета платежей абонентов городской телефонной сети.

Содержание

Содержание 5
Введение 7
1 Постановка задачи и анализ предметной области 10
1.1 Описание предметной области 10
1.2 Анализ программного обеспечения аналогичного назначения 11
1.3 Сравнительный анализ аналогов 20
1.4 Исследование вариантов использования 21
1.5 Анализ требований 29
1.6 Постановка задачи 31
2 Проектирование web-приложения системы учета и анализа абонентских платежей для городской телефонной сети 33
2.1 Архитектура приложения 33
2.2 Проектирование алгоритмов 34
2.3 Проектирование структур данных 36
2.4 Проектирование пользовательского интерфейса 45
3 Планирование разработки и оценка бюджета 50
3.1 Календарное и ресурсное планирование 50
3.2 Расчет бюджета 53
4 Программная реализация системы учета абонентских платежей 58
4.1 Кодирование системы учета абонентских платежей 58
4.2 Развертывание web-приложения 68
5 Анализ качества программного продукта 70
5.1 Тестирование 70
5.2 Метрики кода 74
Заключение 81
Список использованных источников 83
Приложение А. Листинг кода файла Program.cs 85
Приложение Б. Листинг кода файла IPaymentService.cs 88
Приложение В. Листинг кода файла PaymentService.cs 90

Приложение Г. Листинг кода файла PaymentsController.cs 97
Приложение Д. Листинг кода файла Payments/Index.cshtml 103
Приложение Е. Листинг кода файла е PaymentsEmployeeController.cs..... 102
Приложение Ж. Листинг кода файла PaymentsEmployee/Index.cshtml 109
Приложение И. Листинг кода файла DebtorsController.cs 111
Приложение К. Листинг кода файла Debtors/Index.cshtml 115
Приложение Л. Листинг кода файла Account/Manage/Index.cshtml 118
Приложение М. Листинг кода файла Account/Manage/Index.cshtml.cs 121
Приложение Н. Листинг кода файла _Layout.cshtml..... 126
Приложение П. Листинг кода файла PbxContext.cs 129
Приложение Р. Web-страницы пользовательского интерфейса..... 134
Приложение С. Результаты модульного тестирования..... 138
Приложение Т. Листинг кода модульных тестов для модуля работы с платежами 142

Введение

Современный мир немыслим без телефонной связи, которая является одним из основных средств общения и передачи информации. Телефонная связь предоставляет различные услуги абонентам, такие как междугородняя и международная связь, интернет, факсимильная связь и другие. Абоненты городской телефонной сети хотят иметь возможность удобно и быстро оплачивать свои счета, получать детализацию расходов, следить за балансом и задолженностью, а также управлять своими услугами и тарифами в любое время и с любого устройства.

За эти услуги абоненты должны вносить плату, которая зависит от тарифов, времени разговора, количества и типа услуг. Для учета абонентских платежей необходима специальная система, которая позволяет автоматизировать процесс расчетов с абонентами, контролировать их задолженности, учитывать поступающие платежи, объем предоставленных услуг и т. д.

Одним из возможных способов реализации такой системы является разработка web-приложения, которое обеспечивает доступ к системе учета абонентских платежей через интернет. Web-приложение – это программное обеспечение, которое работает на сервере и взаимодействует с пользователем через браузер [1]. Преимуществами web-приложений являются:

– универсальность: web-приложение может работать на любой платформе и операционной системе, которая поддерживает браузер;

– доступность: web-приложение может быть доступно для пользователей в любое время и из любого места, где есть подключение к интернету;

– безопасность: web-приложение может использовать различные механизмы защиты данных, такие как шифрование, аутентификация и авторизация;

– масштабируемость: web-приложение может легко адаптироваться к изменению объема и сложности данных и требований пользователей.

Такое приложение повышает удовлетворенность абонентов, снижает нагрузку на операторов и кассиров, упрощает взаимодействие между абонентами и провайдерами услуг связи, повышает эффективность и надежность системы учета платежей, а подобные web-приложения в целом являются перспективным направлением развития телекоммуникационных технологий, которое отвечает потребностям и ожиданиям современного рынка связи.

Для разработки web-приложения системы учета абонентских платежей для городской телефонной сети необходимо решить следующие задачи:

– изучить литературу на тему проектирования и разработки веб-приложений;

– проанализировать предметную область и определить программные средства аналогичного назначения, провести их анализ;

– выбрать подходящие технологии для реализации web-приложения, такие как языки программирования, средства разработки, протоколы передачи данных и т. д.;

– провести планирование и расчет бюджета разрабатываемого программного продукта;

– спроектировать структуру базы данных для хранения информации о абонентах, услугах, тарифах, платежах и других данных;

– определить структуру разрабатываемого веб-приложения;

– разработать интерфейс web-приложения для взаимодействия с пользователем;

– реализовать функциональность web-приложения для выполнения основных операций по учету абонентских платежей, таких как регистрация и учет абонентов, учет поступающих платежей, выявление задолжников, формирование счетов и квитанций, ведение списка услуг и учет объема предоставленных услуг;

– произвести тестирование реализованного web-приложения.

Целью данной дипломной работы является разработка web-приложения системы учета абонентских платежей для городской телефонной сети с использованием современных технологий программирования.

1 Постановка задачи и анализ предметной области

1.1 Описание предметной области

В данной работе рассматриваются следующие задачи:

- учет абонентов сети;
- ведение списка услуг;
- учет предоставленных услуг;
- прием и обработка платежей;
- выявление задолжников;
- выставление счетов и печать квитанций.

Городская телефонная сеть хранит данные о своих абонентах. У абонента может стоять телефон одного из трех типов: основной, параллельный или спаренный. За каждым абонентом закреплен свой номер телефона. Каждому номеру телефона соответствует адрес (индекс, район, улица, дом, квартира).

Абоненты обязаны платить абонентскую плату. При неуплате после уведомления в течение двух суток отключается абонент. Включение производится при оплате стоимости включения и абонентской платы.

Абонентов любой АТС можно подразделить на простых и льготных. Льготники платят только 50% абонентской платы. В соответствии с этим рассчитывается размер абонентской платы.

В качестве пользователей приложения могут выступать как абоненты телефонной сети, так и сотрудники, которые могут просматривать поступающие платежи и сформированные системой отчеты, изменять условия абонентских тарифов, а также просматривать и отклонять либо принимать заявки абонентов.

Формат web-приложения является удобным и актуальным вариантом для перечисленных задач. Web-приложение может быть доступно из любого места и на любом устройстве с доступом в интернет. Это позволяет абонентам быстро и удобно получать информацию о своих услугах и платежах. Кроме того, веб-приложение может быть легко интегрировано с другими системами учета и

обработки данных, а разделение прав может быть реализовано с помощью механизма авторизации.

1.2 Анализ программного обеспечения аналогичного назначения

Важной частью анализа предметной области является поиск и сравнение программного обеспечения аналогичного назначения.

Подавляющее большинство приложений-аналогов представляют собой решения крупных компаний-операторов либо городских телефонных сетей (ГТС). Для анализа среди них можно выделить следующие популярные web-ресурсы:

– сайт Московской городской телефонной сети (МГТС);

– сайт Муниципального предприятия «Городская телефонная сеть» г. Железногорск (МП «ГТС»);

– сайт Костромской городской телефонной сети (КГТС);

– сайт Муниципального унитарного предприятия «Городская телефонная сеть» (МУП «СУН»);

– сайт Псковской городской телефонной сети (ПГТС).

Рассмотрим указанные выше аналоги, чтобы выделить их особенности и различия.

Первое web-приложение – сайт Московской городской телефонной сети [2].

МГТС предлагает услуги связи и интернета для физических и юридических лиц в Москве и Московской области. На сайте можно подключиться к услугам телефонной сети, узнать тарифы и акции, оплатить счета, получить техническую поддержку и прочее. Вход для абонентов осуществляется через личный кабинет через логин и пароль. При отсутствии логина и пароля, возможна регистрация на сайте через номер телефона или договора, однако для этого нужно являться подключенным абонентом телефонной сети. Присутствует возможность отправки заявки удаленно: для этого необходимо позвонить на номер оператора.

Для контроля задолженности можно проверять баланс в личном кабинете или по телефону оператора. В случае просрочки платежа, услуги могут быть заблокированы до погашения задолженности. Для разблокировки нужно оплатить счет (например, в личном кабинете).

Пользовательский интерфейс МГТС представлен на рисунке 1.

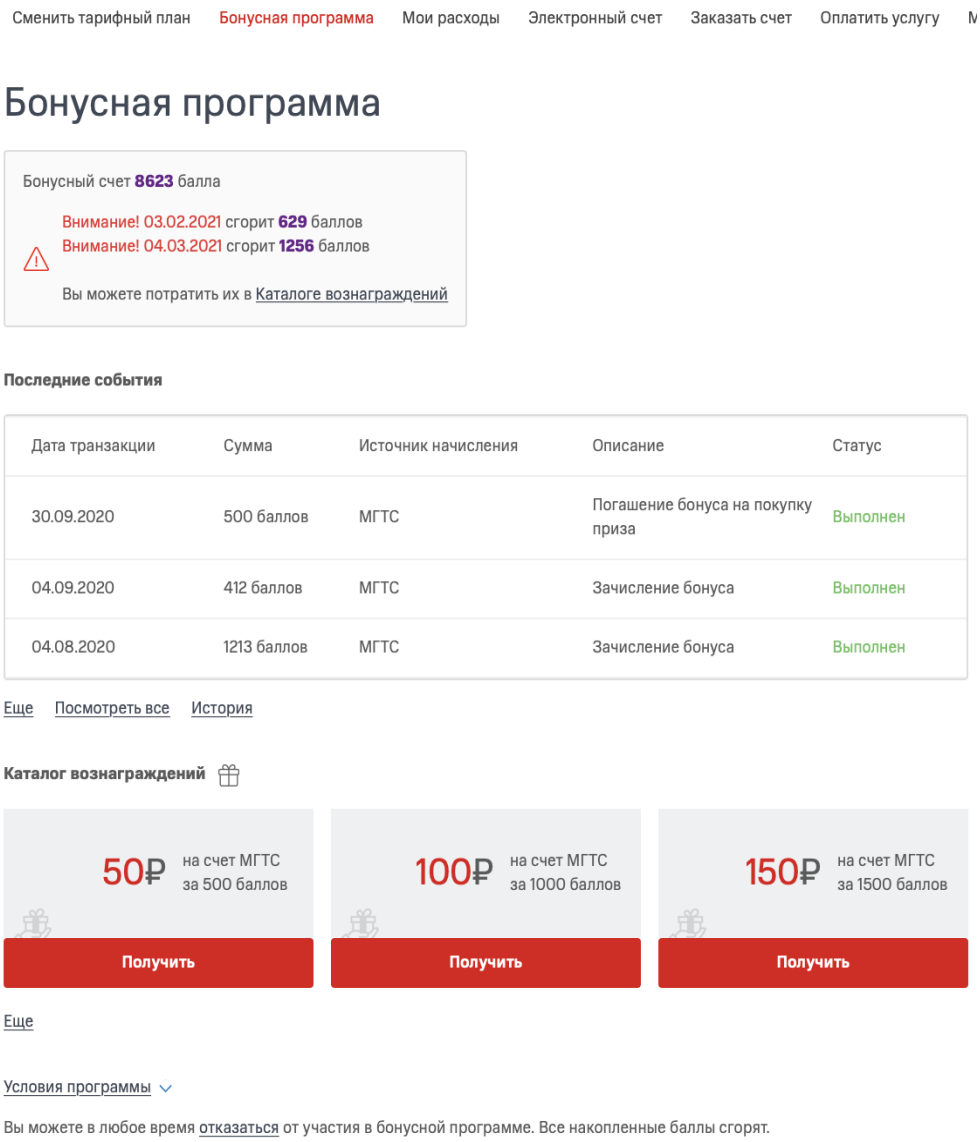

Рисунок 1 – Пользовательский интерфейс приложения «Московская городская телефонная сеть»

Данное приложение обладает следующими функциональными возможностями:

– просмотр расходов;
– просмотр и оплата услуг;

– выставление счетов;

– смена абонентского тарифа;

– вход в личный кабинет по номеру телефона либо номеру счета из договора.

У приложения можно выделить следующие недостатки:

– удаленное подключение производится лишь при связи с оператором по телефону – это может быть неудобно для пользователей;

– приложение предоставляет услуги для бизнеса, которые могут быть неинтересны для рядового пользователя (видеонаблюдение, охрана и настройка оборудования) – данные функциональные возможности создают дополнительную нагрузку на сервер, что может проявляться в нестабильной работе приложения;

– помимо озвученного ранее недостатка, на главной странице присутствует реклама услуг и сервисов МТС, занимающая большую часть пользовательского интерфейса, что делает его менее интуитивным и удобным в навигации, а также поиске нужных функций.

Еще одно приложение аналогичного назначения – сайт городской телефонной сети города Железногорск [3].

Функциональные возможности этого приложения схожи с предыдущим: на сайте также можно узнать о тарифах и акциях, подключиться к услугам, оплатить счета и получить техническую поддержку.

Вход возможен только при помощи логина и пароля; если пользователь забыл пароль или логин, восстановить их можно только через техническую поддержку по номеру телефона оператора.

В случае просрочки абонентом платежа, услуги могут быть заблокированы до погашения задолженности. Отключение услуг также возможно лишь при условии полного погашении имеющейся задолженности на дату расторжения.

Пользовательский интерфейс данного приложения представлен на рисунке 2.

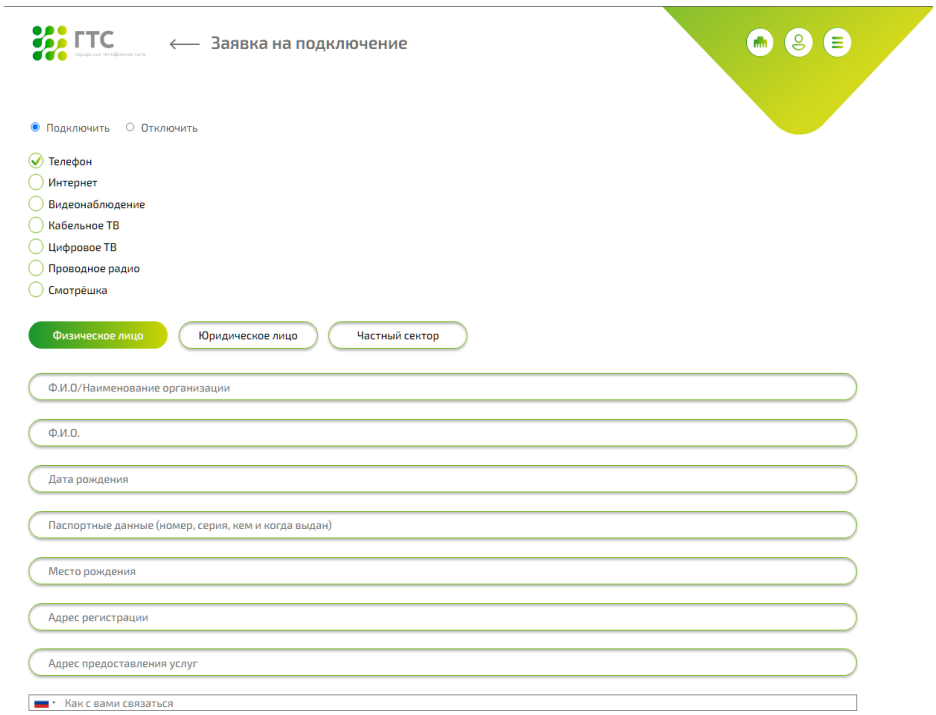

Рисунок 2 – Пользовательский интерфейс приложения «Городская телефонная сеть» г. Железногорск

Недостатки приложения:

– для отправки заявки на подключение необходимо указать номер телефона, по которому с пользователем свяжется оператор – приложение может быть неудобным для пользователей, которые не хотят связываться с оператором по телефону или могут быть недоступны во время звонка от оператора;

– для того чтобы отправить заявку на подключение, необходимо предоставить сайту свои паспортные данные и адрес регистрации;

– у данного приложения, как и у предыдущего есть функциональные возможности, не являющимися актуальными для большинства пользователей (услуги для юридических лиц и частного сектора).

Следующее приложение – сайт Костромской городской телефонной сети [4].

Приложение предоставляет услуги связи, интернета, кабельного телевидения и видеонаблюдения для физических и юридических лиц в Костроме и Костромской области. На сайте можно ознакомиться с предоставляемыми услугами, заказать подключение, оплатить счета, получить консультацию и

прочее. В отличие от предыдущих приложений, у данного приложения есть возможность указать электронную почту вместо телефона в качестве контактной информации перед подключением.

Для доступа в личный кабинет пользователю необходимо ввести свой логин и пароль. Если логин и пароль не были получены, их можно получить лишь в кассах ОАО «КГТС», что является одним из главных недостатков данного приложения.

В случае неоплаты абонентом предоставленных ему услуг связи до указанного срока, доступ к услугам телефонной связи будет приостановлен. Узнать о наличии задолженностей абонент может двумя способами:

– воспользовавшись автоматической справочной службой по расчетам;

– воспользовавшись услугами сотрудников ОАО КГТС.

Как можно заметить, проверить задолженность в личном кабинете приложения не представляется возможным, что является недостатком.

Присутствуют также следующие недостатки:

– в приложении можно выбрать старую версию интерфейса, но несмотря на то, что это может быть удобно для пользователей, привыкших к именно такой версии, обмен данными в ней происходит по протоколу HTTP (все описанные выше приложения работают по протоколу HTTPS), в котором не предусмотрены дополнительные меры по обеспечению безопасных коммуникаций, что создает значительные риски для безопасности пользователя [5];

– некоторые гиперссылки элементов пользовательского интерфейса (к примеру, страница «Для бизнеса») ведут именно на старую версию сайта – помимо угроз безопасности, подобные web-страницы могут отображаться некорректно в некоторых браузерах или устройстве по той причине, что данная версия сайта не поддерживается разработчиками и может использовать устаревшие компоненты;

– доступ в личный кабинет возможен только с внутренней сети ОАО «КГТС», в связи с чем приложение может быть неудобным для пользователей, которые не могут получить логин и пароль в кассах или не хотят связываться с операторами по телефону, либо хотят войти в личный кабинет с другой сети (например, находясь вдали от дома и/или используя мобильный интернет).

Пользовательский интерфейс приложения представлен ниже, на рисунке 3.

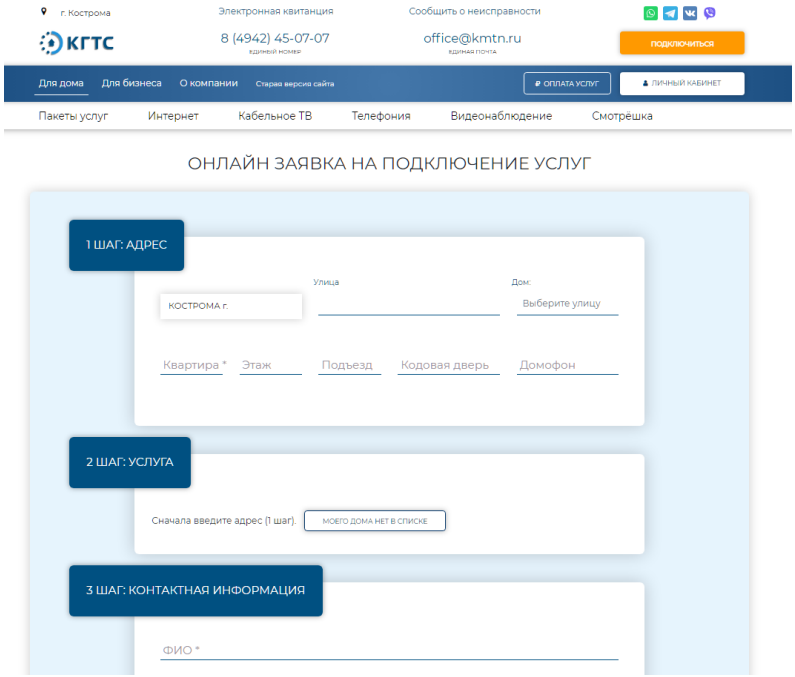

Рисунок 3 – Пользовательский интерфейс приложения «Костромская городская телефонная сеть»

Еще одним приложением, рассматриваемое в качестве аналога, является приложение Муниципального унитарного предприятия «Городская телефонная сеть» [6].

Приложение предлагает услуги телефонной связи, телевидения и систем безопасности для физических и юридических лиц в городе Краснознаменск Московской области.

Доступ в личный кабинет осуществляется по номеру телефона и паролю, который можно получить лишь при звонке по номеру телефона оператора.

У данного приложения нет каких-либо функциональных возможностей, отличающих его от аналогов, описанных ранее.

В случае образования задолженности на лицевом счете абонента, нарушения, оператор приостановит оказание услуг абоненту. Возобновление услуги производится после погашения задолженности. Узнать о наличии задолженности можно в личном кабинете либо обратившись по номеру телефона оператора.

Приложение имеет ряд недостатков, среди которых можно выделить следующее:

– нет возможности отправить заявку на подключение через пользовательский интерфейс приложении, договор на оказание услуг можно заключить только в отделе предприятия;

– отсутствует возможность получать счета или оплачивать предоставляемые услуги из личного кабинета – это возможно только в кассе предприятия и терминалах;

– пользовательский интерфейс приложения статичен: большая часть информации представлена в виде обычного текста, навигация между разделами происходит через гиперссылки;

– в пользовательском интерфейсы наблюдаются недоработки: часть гиперссылок на момент обращения ведет на несуществующие web-страницы (к примеру, пункт «Договор на оказание услуг» меню «Клиентам», рисунок 4).

Последнее рассматриваемое приложение аналогичного назначения – web-приложение Псковской городской телефонной сети [7].

Данное web-приложение предлагает услуги телефонии, телевидения и интернета для физических и юридических лиц в Пскове и Псковской области, а также аренда каналов связи и прочие услуги для операторов.

Узнать о наличии задолженности можно лишь в центре обслуживания клиентов либо через звонок оператору. В случае просрочки платежа, услуги могут быть заблокированы до погашения задолженности.

Возможность удаленной подачи заявки на подключения присутствует: пользователь должен выбрать тариф и ввести свое имя, адрес проживания и адрес электронной почты.

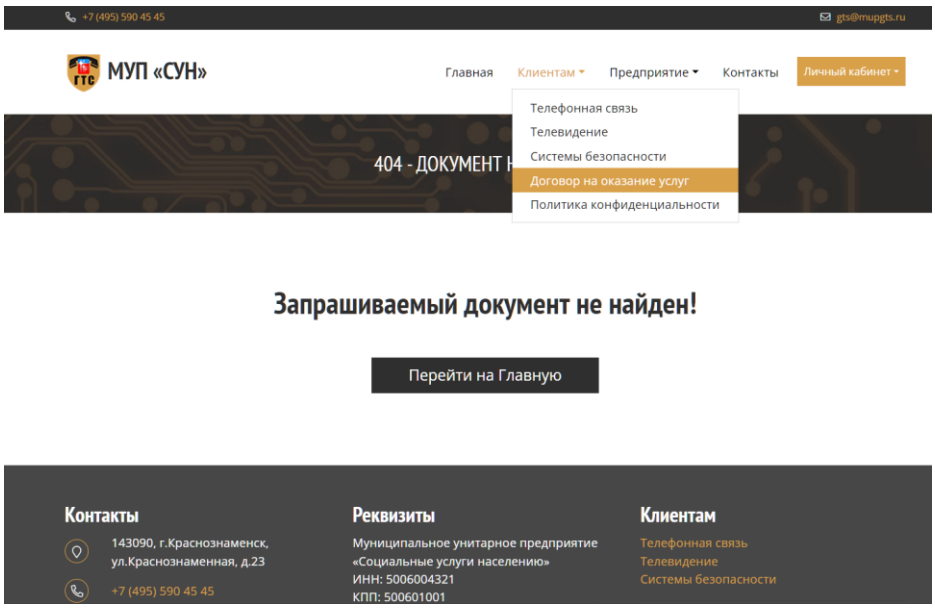

Рисунок 4 – Наблюдаемая ошибка в пользовательском интерфейсе приложения Муниципального унитарного предприятия «Городская телефонная сеть»

На рисунке 5 представлена web-страница, отражающая часть корректно отображаемого пользовательского интерфейса приложения.

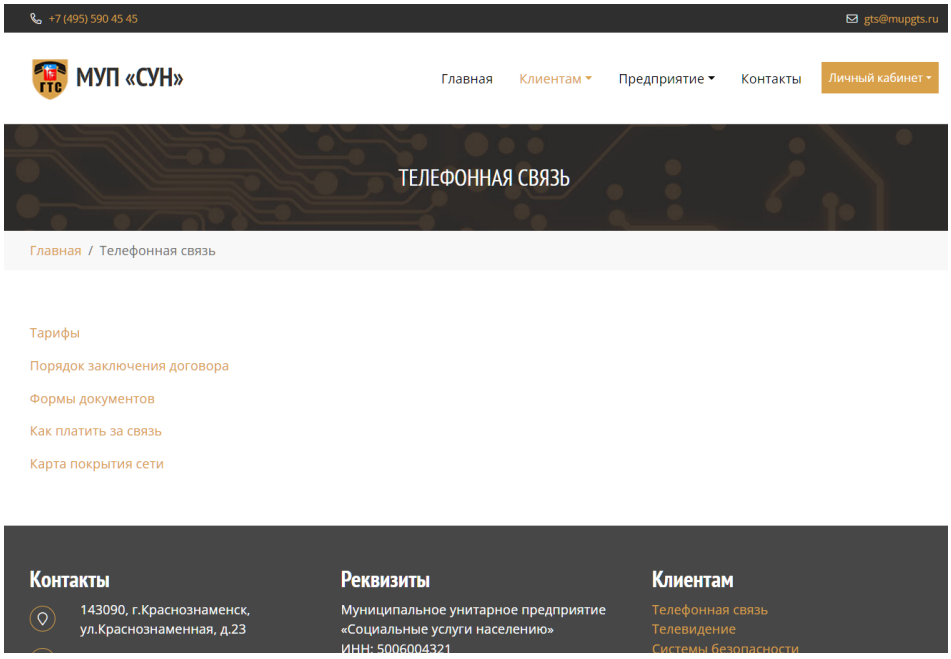

Рисунок 5 – Пользовательский интерфейс приложения Муниципального унитарного предприятия «Городская телефонная сеть»

Номер телефона, как и у приложения Костромской городской телефонной сети, вводить необязательно, что является преимуществом.

В качестве недостатков можно привести следующее:

– у приложения нет личного кабинета для пользователей, в отличие от всех остальных приложений-аналогов;

– большую часть пользовательского интерфейса занимает реклама услуг оператора родительской компании, что делает его менее удобным и информативным;

– приложение выполняет коммуникацию по протоколу HTTP, что, как было сказано ранее, представляет большую угрозу безопасности пользователя и является большим недостатком, так как данные, которые пользователь отправляет при отправке заявки на подключение услуг, могут быть перехвачены злоумышленником.

Пользовательский интерфейс приложения Псковской городской телефонной сети приведен на рисунке 6.

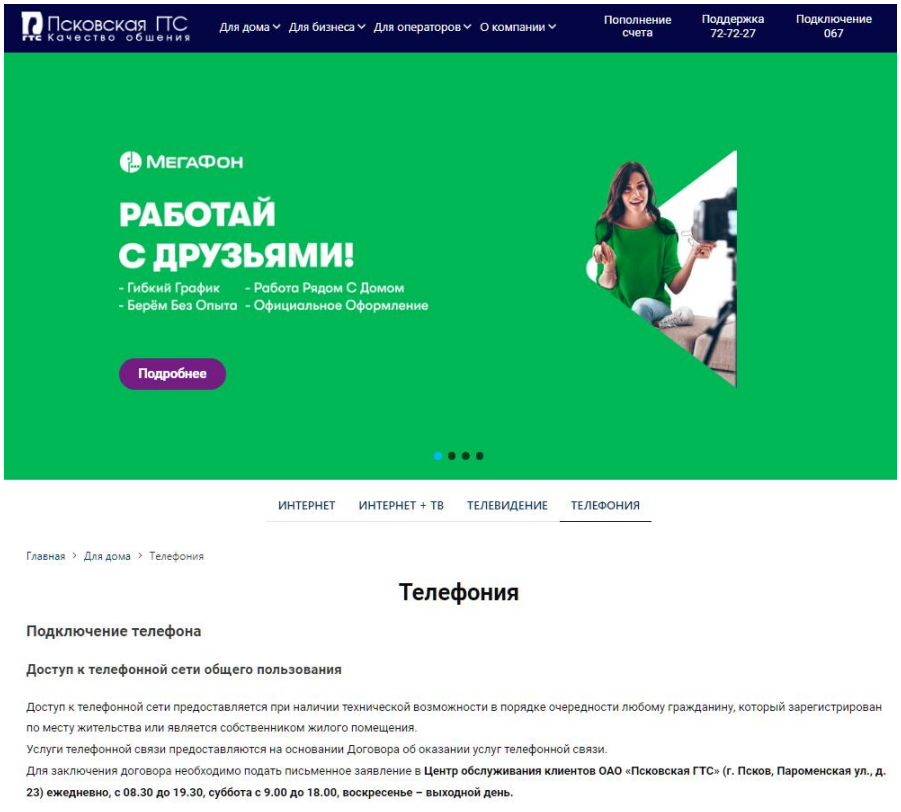

Рисунок 6 – Пользовательский интерфейс приложения «Псковская городская телефонная сеть»

Можно заметить, что каждое из рассмотренных приложений обладает своими особенностями недостатками, из чего следует необходимость в объективной оценке и сравнительном анализе приложений между собой.

1.3 Сравнительный анализ аналогов

После того, как были рассмотрены приведенные выше приложения аналогичного назначения, необходимо подвести итоги и составить оценку.

Для этого приведем таблицу, отражающую основные функции и черты, присутствующие либо отсутствующие в каждом web-приложении (таблица 1).

Были выделены следующие критерии:

– возможность отправки заявки на подключение;

– наличие личного кабинета;

– возможность узнать о наличии задолженностей в личном кабинете;

– возможность задать собственный логин и пароль при подключении;

– возможность указать адрес электронной почты вместо номера телефона;

– использование протокола HTTPS;

– бесплатность продукта;

– отсутствие рекламы.

Таблица 1 – Сравнение аналогов

Приложение	МГТС	МП «ГТС»	КГТС	МУП «СУН»	ПГТС
Возможность отправки заявки на подключение	Да	Да	Да	Нет	Да (частично)
Возможность задать собственный логин и пароль при отправке заявки на подключение	Нет	Нет	Нет	Нет	Нет
Возможность указания электронной почты вместо номера телефона	Нет	Нет	Да	Нет	Да
Наличие личного кабинета	Да	Да	Да (частично)	Да	Нет

Продолжение таблицы 1

Приложение	МГТС	МП «ГТС»	КГТС	МУП «СУН»	ПГТС
Возможность узнать о наличии задолженностей в личном кабинете	Да	Да	Нет	Да	Нет
Наличие рекламы	Да	Нет	Да	Нет	Да
Бесплатность	Нет	Нет	Нет	Нет	Нет
Использование протокола HTTPS	Да	Да	Да (частично)	Да	Нет

Как можно заметить, ни одно из приложений не удовлетворяет всем необходимым требованиям. Ни в одном приложении не является возможным указать свой собственный логин и пароль при отправке заявки на подключение – как правило, для входа требуется номер телефона или договора, а пароль можно получить только при обращении к оператору по телефону. Кроме того, большинство из рассмотренных приложений содержат рекламу, а в контексте задачи, поставленной в рамках данной работы, обладают излишними функциональными возможностями (как правило, это услуги телевидения и интернета).

Все эти особенности следует учесть при разработке собственного web-приложения. Стоит также отметить, что разрабатываемое приложение является бесплатным программный продукт и не будет содержать рекламу. Все это может сделать разработанное приложение подходящим и удобным решением для небольших компаний, предоставляющих услуги телефонии и не нуждающихся в прочих функциональных возможностях.

1.4 Исследование вариантов использования

Для того, чтобы определить процессы, протекающие в системе учета платежей, необходимо проанализировать наиболее вероятные сценарии использования. В данной работе рассматривается web-приложение системы учета абонентских платежей для городской телефонной сети.

Одной из особенностей разрабатываемого приложения является разделение ролей пользователей на абонентов и сотрудников, что требует использования механизмов аутентификации и авторизации.

Аутентификация используется для того, чтобы определить конкретного пользователя, выполняющего запрос к системе, авторизация же используется, чтобы определить, имеет ли конкретный пользователь право на выполнение соответствующего действия [8].

Web-приложение должно обеспечивать возможность прохождения регистрации для пользователей, а также предоставлять для зарегистрированных абонентов следующие возможности:

– просматривать список услуг и тарифов;
– просматривать информацию о задолженности и платежах;
– отправлять заявку на подключение и отключение;
– отправлять заявку на смену тарифа.

В то же время приложение должно предоставлять следующие возможности для сотрудников телефонной сети:

– просматривать список поступающих платежей;
– вносить информацию о новых платежах;
– просматривать список задолжников;
– просматривать отчеты;
– принимать или отклонять абонентские заявки.

На основе приведенных выше задач была сформирована диаграмма вариантов использования.

Диаграмма вариантов использования (также называемая диаграммой прецедентов) представляет собой граф особого вида, элементами которого являются актеры (множество ролей, исполняемых при взаимодействии в системе), интерфейсы (совокупность операций, обеспечивающие функциональность для актеров), сами варианты использования (определение последовательностей действий), а также различные виды связей между ними.

Подобная графическая нотация позволяет наглядно отобразить поведение, которое ожидается от проектируемой системы [9].

Диаграмма вариантов использования для разрабатываемого web-приложения представлена на рисунке 7.

На данной диаграмме присутствуют три актера. Два из них («Абонент» и «Сотрудник») представляют собой категории пользователей приложения.

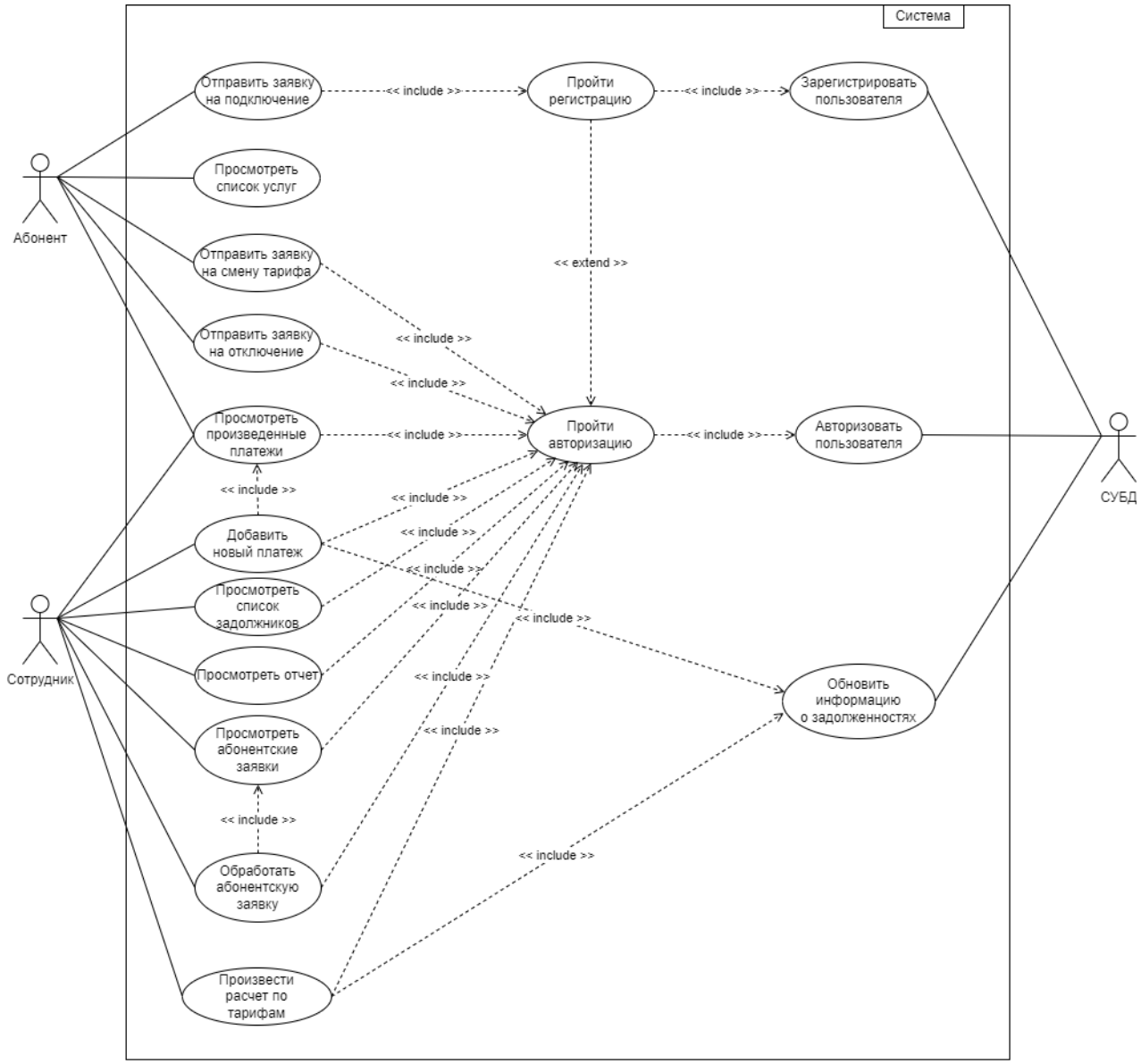

Рисунок 7 – Диаграмма вариантов использования для проектируемого web-приложения

В зависимости от категории пользователям предоставляются определенные права на выполнение операций в системе.

Для основных вариантов использования были составлены спецификации. Данные спецификации приведены далее.

Спецификация первого прецедента включает в себя вариант использования «Пройти регистрацию», так как до подключения пользователь еще не является абонентом (рисунок 8).

Название: Отправить заявку на подключение
ID: 1
Краткое описание: Пользователь (абонент) вводит необходимые данные и отправляет заявку на подключение.
Главные актеры: Абонент, СУБД.
Второстепенные актеры: Нет.
Предусловие: Система загружена, база данных подключена к приложению, отображен пользовательский интерфейс.
Основной поток: Система предоставляет пользователю форму для ввода данных. Пользователь вводит необходимые данные (ФИО, адрес). Пользователь вводит данные для регистрации (логин, пароль). Пользователь отправляет заявку. Система проверяет соответствие введенных данных формату после чего, в случае соответствия, регистрирует пользователя и добавляет в базу данных его заявку на подключение.
Постусловие: Пользователь зарегистрирован, заявка сохранена в базе данных.

Рисунок 8 – Спецификация варианта использования «Отправить заявку на подключение»

Перейдем к следующим сценарием прецедентов («Просмотреть список услуг» и «Отправить заявку на смену тарифа) на рисунках 9 и 10.

Название: Просмотреть список услуг
ID: 2
Краткое описание: Пользователь (абонент или сотрудник) открывает страницу со списком предоставляемых услуг.
Главные актеры: Абонент
Второстепенные актеры: Нет.
Предусловие: Система загружена, база данных подключена к приложению, отображен пользовательский интерфейс.
Основной поток: Система получает список услуг из базы данных и выводит в пользовательский интерфейс.
Постусловие: Список услуг отображен.

Рисунок 9 – Спецификация варианта использования «Просмотреть список услуг»

В отличие от предыдущего варианта использования, для данного прецедента не требуется авторизация, т. к. любой пользователь может просматривать список услуг.

Название: Отправить заявку на смену тарифа
ID: 3
Краткое описание: Пользователь (абонент) выбирает необходимый тариф и отправляет заявку на смену тарифа.
Главные актеры: Абонент, СУБД
Второстепенные актеры: Нет.
Предусловие: Система загружена, база данных подключена к приложению, отображен пользовательский интерфейс, пользователь зарегистрирован.
Основной поток: Пользователь выполняет авторизацию. Система получает список тарифов из базы данных и выводит в пользовательский интерфейс. Пользователь выбирает нужный тариф и отправляет заявку. Система сохраняет заявку в базе данных. В пользовательском интерфейсе выводится подтверждение о выполнении операции.
Постусловие: Заявка сохранена в базе данных.

Рисунок 10 – Спецификация варианта использования «Отправить заявку на смену тарифа»

Следующая спецификация, «Отправить заявку на отключение», устроена схожим образом (рисунок 11).

Название: Отправить заявку на отключение
ID: 4
Краткое описание: Пользователь (абонент) отправляет заявку на отключение.
Главные актеры: Абонент, СУБД
Второстепенные актеры: Нет.
Предусловие: Система загружена, база данных подключена к приложению, отображен пользовательский интерфейс, пользователь зарегистрирован.
Основной поток: Пользователь выполняет авторизацию. База данных проверяет наличие задолженностей у пользователя, после чего отображает в пользовательском интерфейсе сообщение о необходимости погасить задолженность (при ее наличии) либо отображает элемент управления для отправки заявки. Пользователь отправляет заявку на отключение. Система сохраняет заявку в базе данных.
Постусловие: Заявка сохранена в базе данных.

Рисунок 11 – Спецификация варианта использования «Отправить заявку на отключение»

Важная часть данной спецификации – проверка наличия задолженностей абонента. Данная проверка обусловлена тем, что абонент обязан погасить задолженность за услуги, предоставленные ранее.

Следующие спецификации приведены на рисунках 12 и 13.

Название: Просмотреть произведенные платежи
ID: 5
Краткое описание: Пользователь (абонент или сотрудник) заходит на страницу со списком платежей.
Главные актеры: Абонент, Сотрудник, СУБД
Второстепенные актеры: Нет.
Предусловие: Система загружена, база данных подключена к приложению, отображен пользовательский интерфейс, пользователь зарегистрирован.
Основной поток: Пользователь выполняет авторизацию. База данных получает список платежей абонента (в случае, если авторизован пользователь с ролью «Абонент») либо список всех платежей (если авторизован сотрудник). Система выводит список платежей в пользовательском интерфейсе.
Постусловие: Список платежей отображен.

Рисунок 12 – Спецификация варианта использования «Просмотреть произведенные платежи»

В данном варианте использования абонент может просматривать лишь собственные платежи, в то время как сотрудник может просматривать все произведенные платежи.

Название: Добавить новый платеж
ID: 6
Краткое описание: Пользователь (сотрудник) вводит данные необходимые данные и добавляет платеж.
Главные актеры: Сотрудник, СУБД
Второстепенные актеры: Нет.
Предусловие: Система загружена, база данных подключена к приложению, отображен пользовательский интерфейс, пользователь зарегистрирован, отображен список платежей.
Основной поток: Пользователь выполняет авторизацию. Система отображает форму добавления платежа. Пользователь вводит необходимые данные (номер телефона, дата и сумма платежа). Система проверяет соответствие введенных данных формату и добавляет платеж в базу данных (если данные соответствуют формату) либо выводит сообщение о несоответствии формату в пользовательский интерфейс. В случае успешного добавления платежа система обновляет данные о задолженности соответствующего абонента.
Постусловие: Платеж добавлен в базу данных

Рисунок 13 – Спецификация варианта использования «Добавить новый платеж»

Данный вариант использования включает в себя прецедент «Просмотреть произведенные платежи», т. к. он инициируется на странице списка платежей. Сюда также включен прецедент «Обновить информацию о задолженностях», так как платеж способен погасить образовавшуюся ранее задолженность.

Следующие сценарии приведены на рисунках 14 и 15.

Название: Посмотреть список задолжников
ID: 7
Краткое описание: Пользователь (сотрудник) открывает страницу со списком задолжников.
Главные актеры: Сотрудник, СУБД
Второстепенные актеры: Нет.
Предусловие: Система загружена, база данных подключена к приложению, отображен пользовательский интерфейс, пользователь зарегистрирован.
Основной поток: Пользователь выполняет авторизацию. Система получает список всех пользователей, имеющих задолженности, из базы данных и выводит на пользовательский интерфейс.
Постусловие: Список задолжников выведен в пользовательском интерфейсе.

Рисунок 14 – Спецификация варианта использования «Посмотреть список задолжников»

Для выполнения сценария данного прецедента требуется авторизация и привилегии сотрудника, вследствие чего вариант использования расширяется вариантом использования «Пройти авторизацию».

Название: Просмотреть отчет
ID: 8
Краткое описание: Пользователь (сотрудник) открывает страницу отчета, формируемого системой.
Главные актеры: Сотрудник, СУБД
Второстепенные актеры: Нет.
Предусловие: Система загружена, база данных подключена к приложению, отображен пользовательский интерфейс, пользователь зарегистрирован.
Основной поток: Пользователь выполняет авторизацию. Система получает проходит по записям истории тарифа и/или платежей в базе данных за последний месяц и рассчитывает финансовые показатели, создавая отчетность по предоставленным услугам. Система выводит рассчитанные показатели в пользовательском интерфейсе.
Постусловие: Отчет выведен в пользовательском интерфейсе.

Рисунок 15 – Спецификация варианта использования «Просмотреть отчет»

Ситуация с требованием авторизации для представленного выше прецедента и предыдущего идентична.

Рассмотрим последние три прецедента (рисунки 16 – 18).

Название: Просмотреть абонентские заявки
ID: 9
Краткое описание: Пользователь (сотрудник) открывает страницу со списком пользовательских заявок.
Главные актеры: Сотрудник, СУБД
Второстепенные актеры: Нет.
Предусловие: Система загружена, база данных подключена к приложению, отображен пользовательский интерфейс, пользователь зарегистрирован.
Основной поток: Пользователь выполняет авторизацию. Система получает из базы данных список всех активных пользовательских заявок и выводит в пользовательском интерфейсе.
Постусловие: Список абонентских заявок выведен в пользовательском интерфейсе.

Рисунок 16 – Спецификация варианта использования «Просмотреть абонентские заявки»

Данный вариант использования также расширяется упомянутым прецедентом из-за требований авторизации.

Название: Обработать абонентскую заявку
ID: 10
Краткое описание: Пользователь (сотрудник) принимает или отклоняет абонентскую заявку.
Главные актеры: Сотрудник, СУБД
Второстепенные актеры: Нет.
Предусловие: Система загружена, база данных подключена к приложению, отображен пользовательский интерфейс, пользователь зарегистрирован, отображен список абонентских заявок.
Основной поток: Пользователь выполняет авторизацию. Система получает из базы данных список всех активных пользовательских заявок и выводит в пользовательском интерфейсе.
Постусловие: Список абонентских заявок выведен в пользовательском интерфейсе.

Рисунок 17 – Спецификация варианта использования «Обработать абонентскую заявку»

Сценарий описанного выше варианта использования включает в себя выполнение сценария «Просмотреть абонентские заявки», так как для того, чтобы вынести решение по отдельной заявку, нужно открыть список всех заявок.

Кроме того, для его выполнения тоже требуется авторизация.

Название: Произвести расчет по тарифам
ID: 11
Краткое описание: Пользователь (сотрудник) инициирует процесс ежемесячного тарифного расчета.
Главные актеры: Сотрудник, СУБД
Второстепенные актеры: Нет.
Предусловие: Система загружена, база данных подключена к приложению, отображен пользовательский интерфейс, пользователь зарегистрирован.
Основной поток: Пользователь выполняет авторизацию. Пользователь инициирует процесс тарифного расчета. Система получает из базы данных список подключенных пользователей, не имеющих задолженностей, и снимает со счета каждого сумму тарифного платежа. Система создает отчет, отражающий финансовые показатели проведенной операции, и выводит его в пользовательском интерфейсе.
Постусловие: Проведен тарифный расчет, финансовый отчет выведен в пользовательском интерфейсе.

Рисунок 18 – Спецификация варианта использования «Произвести расчет по тарифам»

Из рисунка следует, что указанный на последнем рисунке сценарий, как и все варианты использования, предназначенные для сотрудников, расширяется прецедентом «Пройти авторизацию», а также включает в себя сценарий прецедента «Обновить информацию о задолженностях» – это связано с тем, что баланс на счету абонента может стать отрицательным, что может повлечь за собой скорое (в течение нескольких дней) временное прекращение предоставления услуг по причине образования задолженности.

1.5 Анализ требований

Наличие явного набора функциональных требований очень важно при проектировании системы, так как это позволяет определить ее перечень ее функциональных возможностей до начала программирования и свести к

минимуму изменения, вносимые после начала разработки, так как явно определенные требования могут быть проанализированы и утверждены в соответствии с запросами пользователя, а не предположениями разработчика [10].

В связи с этим сформируем функциональные требования на основе полученных прецедентов и спецификаций, составленных ранее для основных вариантов использования.

Приложение должно позволять пользователям с ролью «Абонент» выполнять следующие действия:

– просматривать список предоставляемых услуг;
– просматривать собственные произведенные платежи;
– отправлять заявку на подключение к телефонной сети;
– отправлять заявку на смену тарифа;
– проверять баланс абонентского счета и наличие задолженностей.

Для пользователей с ролью «Сотрудник» приложение должно позволять выполнять следующие операции:

– просматривать список всех поступающих платежей;
– вносить информацию о новых платежах;
– просматривать список абонентов, имеющих задолженности;
– просматривать сформированные системой отчеты об объеме предоставленных услуг;
– принимать или отклонять пользовательские заявки на подключение/отключение/смену тарифа.

Роль пользователя должна однозначно определяться при входе в систему (аутентификации).

Вся информация о пользователях, заявках, платежах, а также все остальные данные, имеющие отношение к телефонной сети, должны храниться в базе данных.

1.6 Постановка задачи

В соответствии с описанными ранее требованиями должно быть спроектировано и реализовано web-приложение для системы учета абонентских платежей городской телефонной сети.

Поскольку разрабатываемое приложение является web-приложением, оно будет разделено на две части:

– серверная часть, обрабатывающая запросы и выполняющая коммуникацию с базой данных;

клиентская часть (web-браузер), отображающая пользовательский интерфейс и взаимодействующая с серверной часть посредством HTTP-запросов.

Для хранения данных будет использована реляционная база данных под управлением СУБД Microsoft SQL Server 2018 – системы, полностью ориентированной на web-приложения, которая может находиться как на том же компьютере, что и серверная часть (локально), так и на удаленном сервере.

Для связи реляционной модели с программной моделью будет использоваться ORM-система Entity Framework Core.

Для написания серверной части должен использоваться язык программирования C# версии 10.0 на платформе .NET 6.

Основным средством для решения поставленной задачи является платформа для создания web-приложений ASP.NET Core 6. Это универсальное и популярное кроссплатформенное средство с открытым исходным кодом для разработки и отладки web-приложений на платформе .NET.

Для определения роли (категории пользователя) и разделения прав помимо аутентификации должна проводиться авторизация пользователя. Механизм аутентификации и авторизации будет обеспечиваться программной библиотекой ASP.NET Identity.

Для проведения тестирования будет применено автоматизированное средство MSTest, имеющее интеграцию со средой разработки

Microsoft Visual Studio. Для подмены программных зависимостей во время модульного тестирования будет применена библиотека Moq.

Для реализации статических элементов страниц и шаблонов необходимо использовать язык разметки HTML 5 и CSS3. Для интеграции программного кода на языке C# в разметку будет использоваться движок пользовательских представлений Razor.

Часть динамических сценариев на страницах будет реализована при помощи JavaScript. Для изменения стилей страниц будут использованы каскадные таблицы стилей (CSS).

Пользовательский интерфейс должен корректно отображаться при использовании следующих браузеров:

- Google Chrome 60 и выше.
- Firefox 60 и выше;
- Microsoft Edge 88 и выше;
- Safari 12 и выше.

Для выполнения данного требования будет использован CSS-фреймворк Bootstrap 5, позволяющий быстро разработать адаптивный пользовательский интерфейс, поддерживающий большинство современных браузеров.

2 Проектирование web-приложения системы учета и анализа абонентских платежей для городской телефонной сети

2.1 Архитектура приложения

Как уже было сказано ранее, приложение состоит из нескольких частей. Клиентской частью выступает веб-страница в браузере пользователя, в то время как серверная часть приложения разворачивается на удаленной машине и устанавливает соединение с сервером базы данных.

Данный процесс можно отобразить при помощи диаграммы, представленной на рисунке 19.

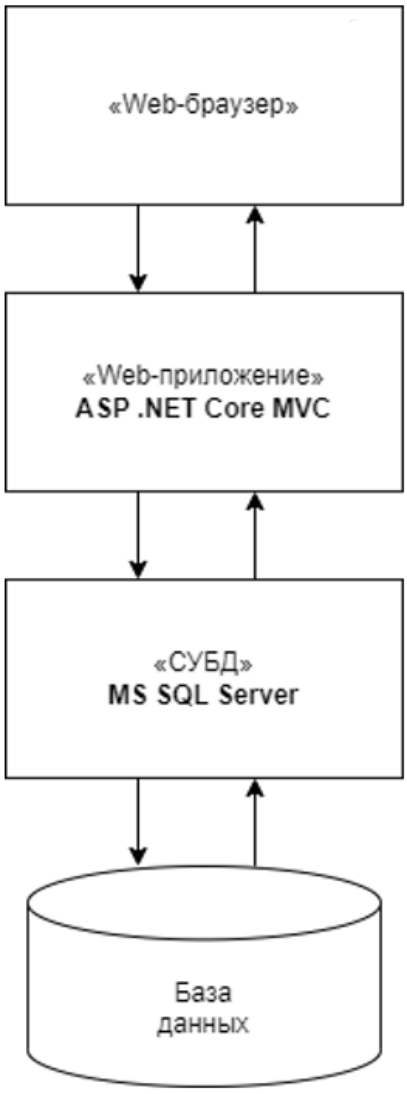

Рисунок 19 – Диаграмма, отображающая архитектуру проектируемого приложения

При реализации серверной части приложения будет использован шаблон проектирования MVC. Данный шаблон предполагает разделение приложения на три основных уровня обработки информации:

– Model (модель, предоставляющая данные);

– View (представление, то есть пользовательский интерфейс приложения, отображающий модель и ее изменения);

– Controller (контроллер, отвечающий за интерпретацию действия пользователя в представлении и изменения модели) [11].

Как можно заметить, компоненты на диаграмме взаимодействуют между собой: web-приложение обменивается данными с СУБД, которая управляет самой базой данных. При этом web-приложение также коммуницирует с web-браузером, принимая и обрабатывая его HTTP-запросы и отправляя ответы. С web-браузером в свою очередь взаимодействует пользователь.

2.2 Проектирование алгоритмов

Для того чтобы приложение могло корректно и эффективно функционировать, необходимо правильно спроектировать алгоритмы, соответствующие установленным требованиям.

В качестве примера был выбран алгоритм обновления информации о задолженностях абонентов – одного из наиболее часто выполняемых бизнес-процессов разрабатываемого web-приложения. Для его наглядного отображения воспользуемся диаграммой деятельности (рисунок 20).

На данной диаграмме показан процесс обновления информации задолженностей абонентов: алгоритм проходит по каждому из абонентов и, если абонент не отключен от городской телефонной сети полностью (выполняется проверка статуса), производит сравнение баланса на его лицевом счете. В случае, если баланс абонента отрицательный, а статус задолженности ему еще не присвоен, то данному абоненту назначается задолженность. Если же баланс положительный, но за абонентом все еще числится задолженность – она списывается.

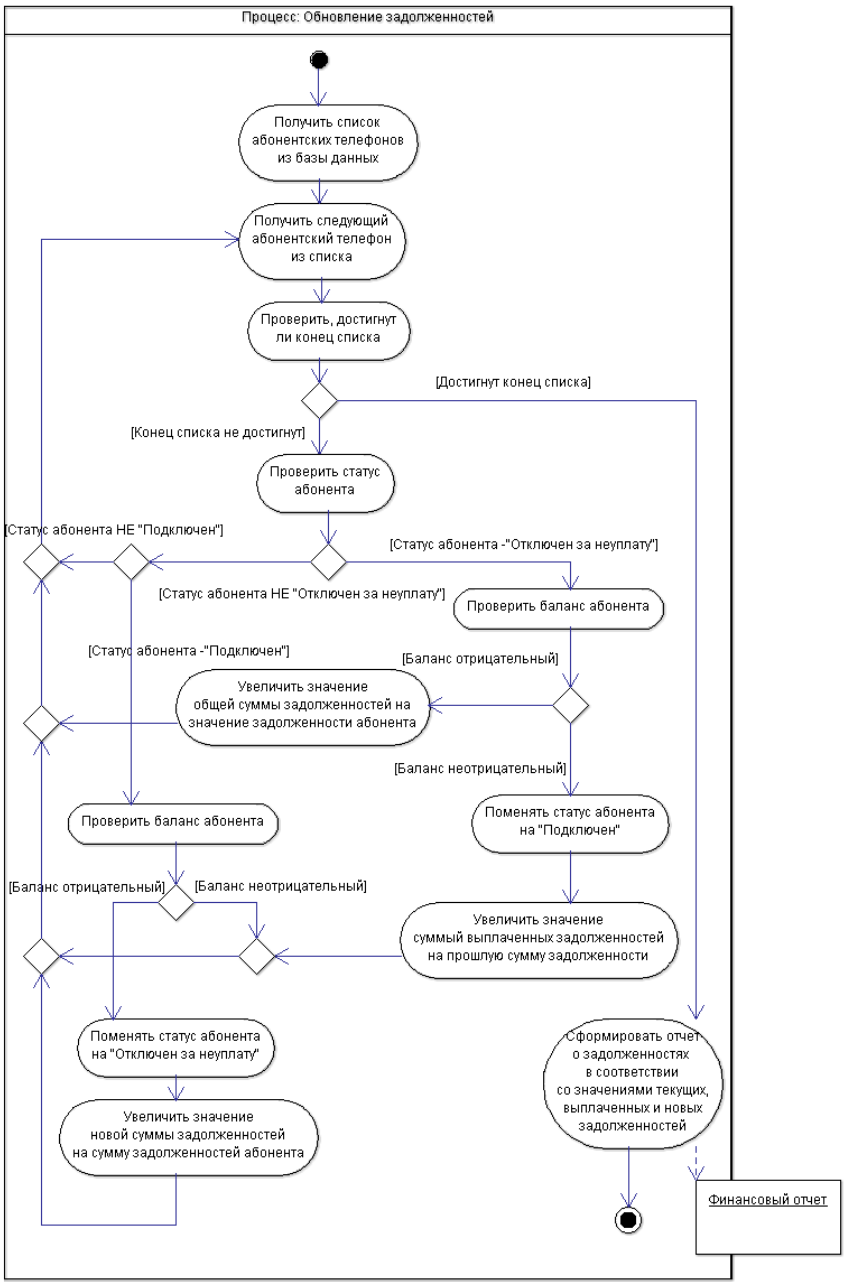

Рисунок 20 – Диаграмма деятельности алгоритма обновления информации о задолженностях

Результатом работы алгоритма, помимо обновления данных, является отчет, в котором содержится рассчитанные показатели выплаченных, новых и общих задолженностей.

2.3 Проектирование структур данных

Для хранения данных городской телефонной сети в приложении используются базы данных.

База данных – это компьютеризированная система хранения записей, предназначенная для хранения информации и предоставления пользователям возможностей ее извлечения и изменения. К информации может относиться все необходимое для моделирования предметной области [12].

Перед созданием базы данных необходимо продумать ее структуру – выполнить проектирование базы данных.

Для отражения структуры базы данных приложения воспользуемся диаграммой «сущность-связь» в нотации IDEF1X. Подобная диаграмма отражает набор атрибутов каждой сущности, а также связи между ними. Диаграмма для базы данных проектируемого приложения представлена на рисунке 21.

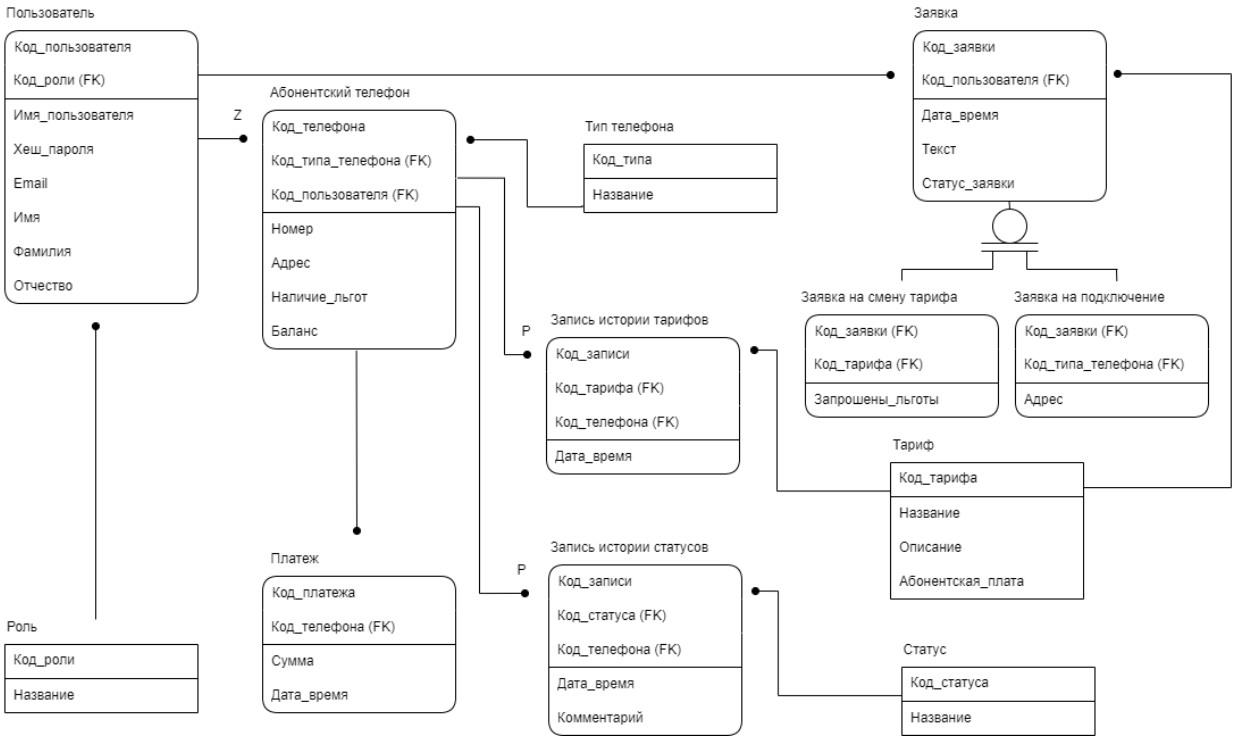

Рисунок 21 – IDEF1X-диаграмма базы данных проектируемого приложения

В качестве СУБД используется Microsoft SQL Server. База данных состоит из 10 сущностей (таблиц).

Информация об абонентах и сотрудниках хранится в таблице «Пользователь», а информация об их ролях – в таблице «Роль».

Данные о телефоне абонента, совмещенные с его лицевым счетом, хранятся в таблице «Абонентский телефон», возможные типы телефонов сохранены в таблице «Тип телефона».

Данные о тарифах и истории тарифов абонента расположены в таблицах «Тариф» и «Запись истории тарифов» соответственно. Для присвоения статусов задолженности, а также а также статусов подключенных и отключенных абонентов используются таблицы «Статус» и «Запись истории статусов».

Последние рассматриваемые таблицы – «Заявка», «Заявка на подключение» и «Заявка на смену тарифа». Данные таблицы представляют абонентские заявки. На диаграмме они соединены связью категоризации, поскольку часть атрибутов разделены между двумя типами сущности.

Далее были сформированы таблицы, отображающие характеристики базы данных/ Столбец «Роль» в таблицах показывает, какие атрибуты сущностей являются первичными (PK) и внешними (FK) ключами.

Рассмотрим первые таблицы (таблицы 5 – 8).

Таблица 5 – Таблица «SubscriberPhone» базы данных

№	Название	Идентификатор	Тип данных	Роль
1	Идентификатор	Id	int	PK
2	Номер телефона	Number	nvarchar(16)	-
3	Адрес	Address	nvarchar(256)	-
4	Идентификатор типа телефона	PhoneTypeId	int	FK
5	Наличие льгот	IsReducedRate	bit	-
6	Баланс	AccountBalance	decimal(18,2)	-

Как было сказано ранее, таблица абонентских телефонов также выступает как лицевой счет, храня информацию о балансе, а также наличии льгот – данный факт определяется значением столбца, представленного типом данных «bit», принимающим одно из двух значения (ложь/истина). Данная таблица связана отношением «один-к-одному» со следующей таблицей.

Таблица 6 – Таблица «User» базы данных

№	Название	Идентификатор	Тип данных	Роль
1	Идентификатор	Id	int	PK
2	Имя пользователя	UserName	nvarchar(256)	-
3	Адрес электронной почты	Email	nvarchar(256)	-
4	Хеш пароля	PasswordHash	nvarchar(max)	-
5	Имя	FirstName	nvarchar(32)	-
6	Фамилия	LastName	nvarchar(32)	-
7	Отчество	Patronymic	nvarchar(32)	-
8	Идентификатор телефона	PhoneId	int	FK

Таблица пользователей, приведенная выше, является основной таблицей, участвующей в механизме аутентификации и авторизации.

Данная таблица связана отношением «один-ко-многим» с таблицей «Role».

Таблица 7 – Таблица «Role» базы данных

№	Название	Идентификатор	Тип данных	Роль
1	Идентификатор	Id	int	PK
2	Название роли	Name	nvarchar(256)	-

Так же, как и предыдущая таблица, таблица ролей используется в web-приложении для обеспечения авторизации и разделения прав пользователей.

Таблица 8 – Таблица «Payment» базы данных

№	Название	Идентификатор	Тип данных	Роль
1	Идентификатор	Id	int	PK
2	Идентификатор телефона	PhoneId	int	FK
3	Сумма платежа	Amount	decimal(18,2)	-
4	Дата и время	DateTime	datetime2	-

Как можно заметить, таблица платежей связана также отношением с таблицей абонентских телефонов.

Далее рассмотрим таблицы 9 – 12.

Таблица 9 – Таблица «Tariff» базы данных

№	Название	Идентификатор	Тип данных	Роль
1	Идентификатор	Id	int	PK
2	Название	Name	nvarchar(256)	-
3	Абонентская плата	SubscriptionFee	decimal(18,2)	-

Таблица тарифов связана с таблицей «TariffHistoryEntry», служащей для хранения истории тарифов.

Таблица 10 – Таблица «TariffHistoryEntry» базы данных

№	Название	Идентификатор	Тип данных	Роль
1	Идентификатор	Id	int	PK
2	Идентификатор телефона	PhoneId	int	FK
3	Идентификатор тарифа	TariffId	int	FK
4	Дата выбора тарифа	DateTime	datetime2	-

Схожим образом устроены следующие две таблицы.

Таблица 11 – Таблица «PhoneStatus» базы данных

№	Название	Идентификатор	Тип данных	Роль
1	Идентификатор	Id	int	PK
2	Название	Name	nvarchar(256)	-
3	Абонентская плата	SubscriptionFee	decimal(18,2)	-

Таблица «PhoneStatus» представляет три возможных статуса абонентского телефона: подключен, отключен и отключен за неуплату.

Следующая таблица представляет запись истории статусов.

Таблица 12 – Таблица «StatusHistoryEntry» базы данных

№	Название	Идентификатор	Тип данных	Роль
1	Идентификатор	Id	int	PK

Продолжение таблицы 12

№	Название	Идентификатор	Тип данных	Роль
2	Идентификатор телефона	PhoneId	int	FK
3	Идентификатор статуса	StatusId	int	FK
4	Дата присвоения статуса	DateTime	datetime2	-
5	Комментарий	Comment	nvarchar(128)	-

Последним столбцом таблицы является строка для короткого комментария об абоненте и/или его текущем статусе.

Следующая таблица (таблица 13) служит для представления типа телефона (основной, спаренный, параллельный).

Таблица 13 – Таблица «PhoneType» базы данных

№	Название	Идентификатор	Тип данных	Роль
1	Идентификатор	Id	int	PK
2	Название типа	Name	nvarchar(32)	-

Последняя таблица, «Request», хранит абонентские заявки, обрабатываемые сотрудниками телефонной сети (таблица 14).

Таблица 14 – Таблица «Request» базы данных

№	Название	Идентификатор	Тип данных	Роль
1	Идентификатор	Id	int	PK
2	Текст заявки	Text	nvarchar(4096)	-
3	Дискриминатор типа	Discriminator	nvarchar(max)	-
4	Идентификатор тарифа	TariffId	int	FK
5	Идентификатор типа телефона	PhoneTypeId	int	FK
6	Запрошены льготы	IsReducedRate	bit	-
7	Адрес	Address	nvarchar(256)	-
8	Статус заявки	IsAccepted	bit (nullable)	-

Как можно заметить из таблицы, тип заявки определяется столбцов дискриминатора, выступающего в роли разделителя. Данный столбец необходим, так как в одной таблице хранятся данные как для заявок на подключение (адрес и идентификатор типа телефона), так и для заявок на смену тарифа (идентификатор тарифа и запрос на предоставление льгот).

Далее необходимо провести физическое моделирование базы данных. Для выполнения данного этапа был написан программный код на языке C# с применением библиотеки Entity Framework Core (EF Core).

EF Core – это библиотека, которую разработчики программного обеспечения могут использовать для доступа к базам данных. EF Core разработан как инструмент объектно-реляционного отображения (ORM). Эти инструменты занимаются отображением между двумя мирами: реляционной базой данных (с собственным программным интерфейсом) и объектно-ориентированным миром классов кода программного обеспечения. Основное преимущество EF – предоставить разработчикам программного обеспечения возможность быстрого написания кода доступа к базе данных на языке, который они, возможно, знают лучше, чем SQL [13].

Также была применена библиотека ASP .NET Core Identity, предоставляющая реализацию механизмов аутентификации, авторизации, хеширования паролей, шифрования данных и тому подобное [14].

Entity Framework предоставляет два варианта создания базы данных: EF может создать ее за вас, используя подход «Сначала код» (code-first), или вы можно предоставить существующую базу данных, созданную за пределами EF (этот подход называется «Сначала база данных» или database-first).

Был применен первый подход, подразумевающий, что базы данных еще не существует, а ее создание происходит путем написания программного кода, который будет использован библиотекой для инициализации таблиц и настройки соответствующих ограничений.

Для каждой сущности был составлен класс с набором свойств, представляющих атрибуты соответствующей сущности, а также двумя

конструкторами: открытым, для создания экземпляров класса внутри приложения, и конструктор без параметров для создания объектов при загрузке из базы данных.

Для некоторых сущностей также определены навигационные свойства (Navigation Property), задающие связи «один-ко-многим» и «один-к-одному» между сущностями, а также обеспечивающие доступ к объектам сущностей, Id которых выступает в роли внешних ключей. Ограничения данных для сущностей настраиваются путем написания классов-конфигураторов.

Пример класса, написанного для сущности «Абонентский телефон», приведен на листинге 1.

Листинг 1 – Исходный код класса сущности «SubscriberPhone»

```
public class SubscriberPhone
{
     public SubscriberPhone(string number, PhoneType phoneType,
          string address) : this(number)
     {
          PhoneType = phoneType;
          Address = address;
     }

     protected SubscriberPhone() : this(default)
     {
     }

     private SubscriberPhone(string number)
     {
          Number = number;
          Payments = new List<Payment>();
          TariffHistory = new List<TariffHistoryEntry>();
          StatusHistory = new List<StatusHistoryEntry>();
     }

     public int Id { get; set; }
     public string Number { get; set; }
     public string Address { get; set; }
     public int PhoneTypeId { get; set; }
     public virtual PhoneType PhoneType { get; set; }
     public bool IsReducedRate { get; set; }
     public decimal AccountBalance { get; set; }
     public virtual ICollection<Payment> Payments { get; set; }
     public virtual ICollection<TariffHistoryEntry> TariffHistory
          { get; set; }
```

```
     public virtual ICollection<StatusHistoryEntry> StatusHistory
          { get; set; }
     public int CurrentStatusId => CurrentStatus.Id;
     public PhoneStatus CurrentStatus =>
          StatusHistory.LastOrDefault()?.Status;
}
```

На листинге 2 представлен исходный класс конфигуратора, соответствующего данной сущности.

Листинг 2 – Исходный код класса-конфигуратора для сущности «SubscriberPhone»

```
public class SubscriberPhoneEntityTypeConfiguration :
     IEntityTypeConfiguration<SubscriberPhone>
{
     private const int AddressMaxLength = 256;
     private const int NumberMaxLength = 16;

     public void Configure(EntityTypeBuilder<SubscriberPhone>
          builder)
     {
          builder.Property(p => p.Number)
               .IsRequired()
               .HasMaxLength(NumberMaxLength);

          builder.Property(p => p.Address)
               .IsRequired()
               .HasMaxLength(AddressMaxLength);

          builder.Ignore(p => p.CurrentStatus)
               .Ignore(p => p.CurrentStatusId);

          builder.HasIndex(p => p.Number)
               .IsUnique();
     }
}
```

Настройка сущности происходит в единственном методе класса «Configure». Код, представленный на упомянутом выше листинге, среди прочего, задает максимальную длину для атрибутов «Number» и «Address», а также указывает для номера ограничение уникальности, не позволяющее добавить в таблицу двух телефонов с одинаковыми номерами.

Все классы сущностей объединяются в контексте базы данных (DbContext), который представляет класс, объявляющий наборы данных (DbSet) нужных сущностей, а также метод настройки модели базы OnModelCreating(), в

котором также можно задать нужные ограничения (связи «многие-ко-многим», индексы и т. п.), однако, как замечено выше, под настройку ограничений сущностей предпочтительнее отводить отдельные классы-конфигураторы. Здесь же происходит поиск данных классов через вызов соответствующего метода. В конструктор контекста передается строка подключения для импортирования необходимых настроек из файла конфигурации проекта, включающих в себя, среди прочего, выбранного поставщика данных (MS SQL Server) и название базы данных.

Исходный код класса контекста базы данных приведен на листинге 3.

Листинг 3 – Исходный код класса контекста базы данных

```
public class PbxContext :
     IdentityDbContext<User, IdentityRole<int>, int>
{
     public PbxContext(DbContextOptions<PbxContext> options) :
          base(options)
     {
     }

     public virtual DbSet<PhoneType>
          PhoneTypes { get; set; }

     public virtual DbSet<SubscriberPhone>
          SubscriberPhones { get; set; }

     public virtual DbSet<PhoneStatus>
          PhoneStatus { get; set; }

     public virtual DbSet<StatusHistoryEntry>
          StatusHistoryEntries { get; set; }

     public virtual DbSet<Tariff>
          Tariffs { get; set; }

     public virtual DbSet<TariffHistoryEntry>
          TariffHistoryEntries { get; set; }

     public virtual DbSet<Payment>
          Payments { get; set; }
     public virtual DbSet<Call>
          Calls { get; set; }

     protected override void OnModelCreating(ModelBuilder
          modelBuilder)
```

```
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.ApplyConfigurationsFromAssembly(
            typeof(CallEntityTypeConfiguration).Assembly);
    }
}
```

Стоит отметить, что данный класс наследуется от класса IdentityDbContext – части библиотеки ASP .NET Core Identity. Данный тип контекста выполняет конфигурацию классов «User», «Role» и ряда других для обеспечения работы механизмов авторизации и аутентификации, а также разделения привилегий пользователей.

Непосредственно создание базы данных происходит посредством использования миграций – автоматически генерируемых файлов с кодом на языке C#, предназначенных для последовательного внесения изменений в схему базы данных [15].

Как следует из определения, миграции позволяют не только создавать, но и вносить изменения в уже созданные базы данных, а также отменять уже внесенные изменения, «откатывая» базу данных до исходного состояния.

2.4 Проектирование пользовательского интерфейса

Вся клиентская часть приложения системы учета абонентских платежей является web-сайтом, т. е. системой из страниц, представляющих из себя документы, написанные на языке разметки гипертекста (HTML). Данные страницы можно просмотреть с помощью web-браузера.

Страницы клиентской части должны быть спроектированы таким образом, чтобы они корректно отображались во всех современных браузерах. Интерфейс должен быть логично структурирован и интуитивно понятен пользователю. Для этого все страницы сайта должны иметь схожую структуру.

В соответствии с заданными требованиями был спроектирован общий макет сайта. Графический интерфейс всех страниц сайта должен содержать следующие элементы:

– верхнюю панель, состоящую из заголовка с левой стороны и кнопки доступа к личному кабинету с правой стороны;

– навигационное меню в верхней части сайта для обеспечения доступа ко всем разделам из любого другого раздела;

– основную область, отображающую содержимое выбранной страницы сайта.

Ниже, на рисунке 22, представлен общий макет страницы сайта. Как можно видеть, сверху находится панель с навигационном меню, обеспечивающими навигацию по основным страницам сайта:

– главная страница («О сайте»);

– страница со списком («Услуги»);

– страница со списком тарифов («Тарифы»);

– страница личного кабинета («Личный кабинет»).

На главной странице в области содержимого страницы будет размещена информация о сайте.

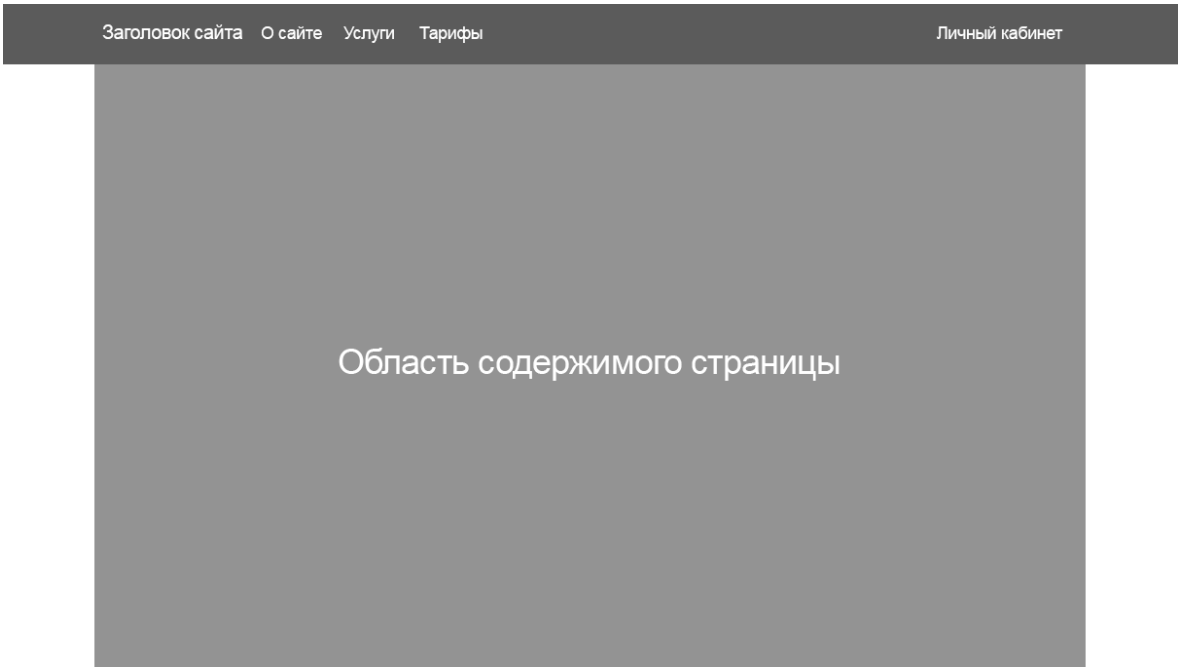

Рисунок 22 – Общий макет страницы сайта

Для остальных страниц были сформированы отдельные макеты.

Рассмотрим макет страницы «Услуги» (рисунок 23).

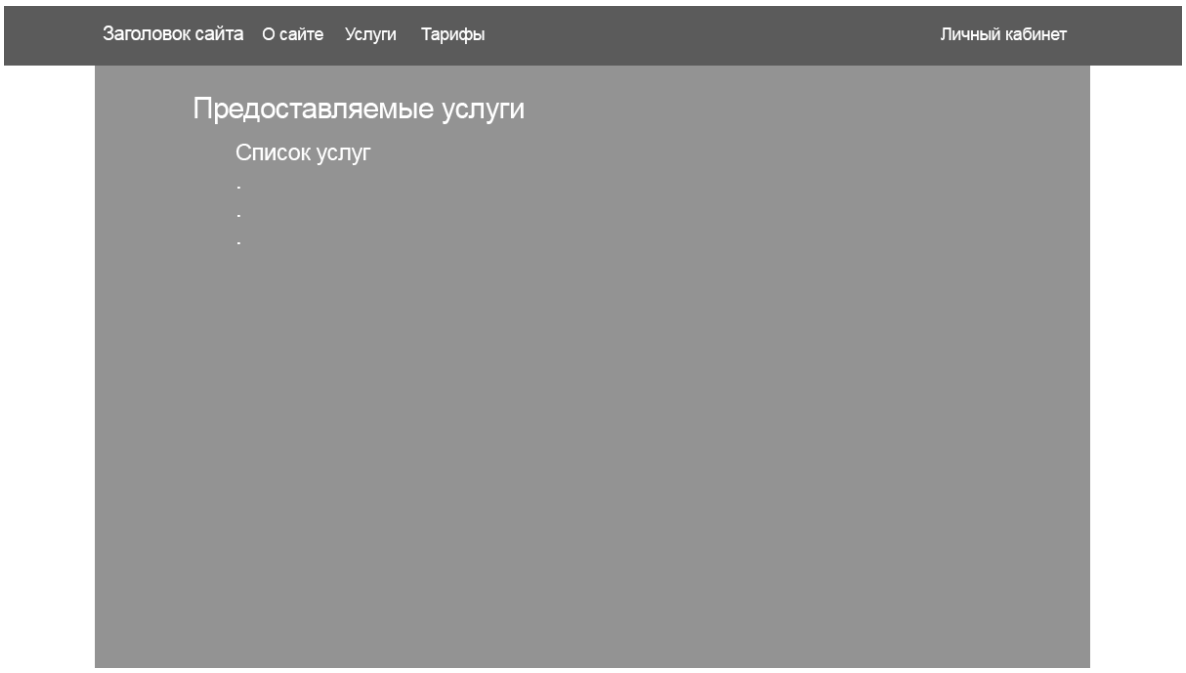

Рисунок 23 – Макет страницы «Услуги»

Страница тарифов схожа со страницей предоставляемых услуг (рисунок 24).

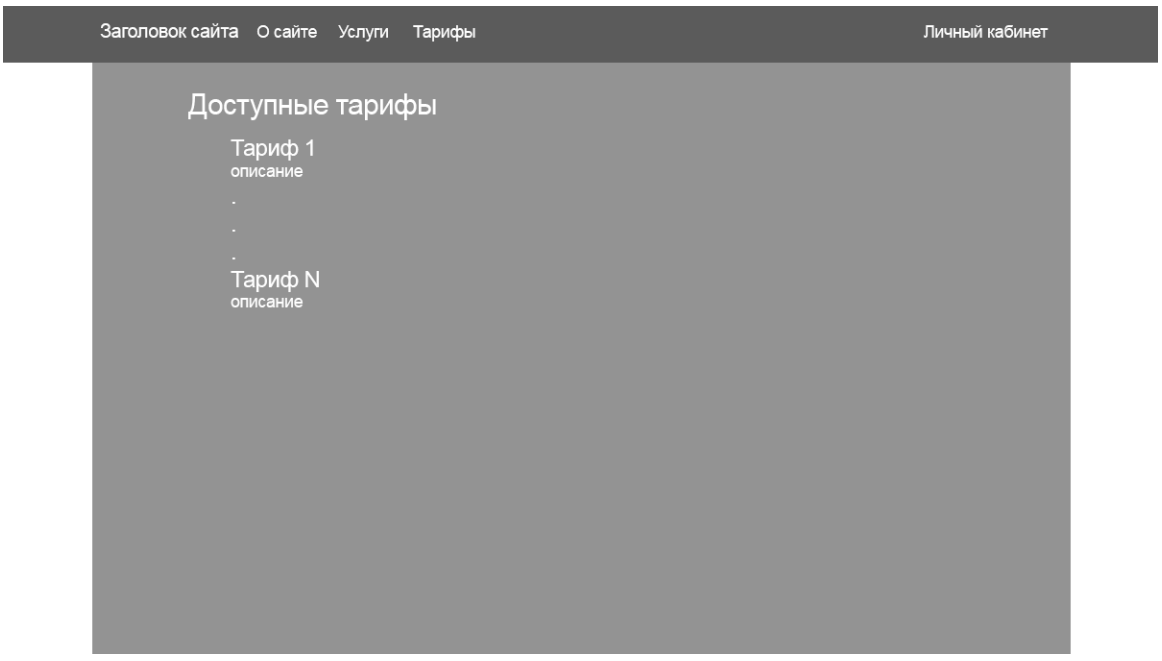

Рисунок 24 – Макет страницы «Тарифы»

Далее рассмотрим макет личного кабинета. Поскольку возможности пользователей различаются в зависимости от их роли, было создано два макета

личного кабинета: для рядового пользователя (абонента) и для сотрудника (рисунки 25, 26).

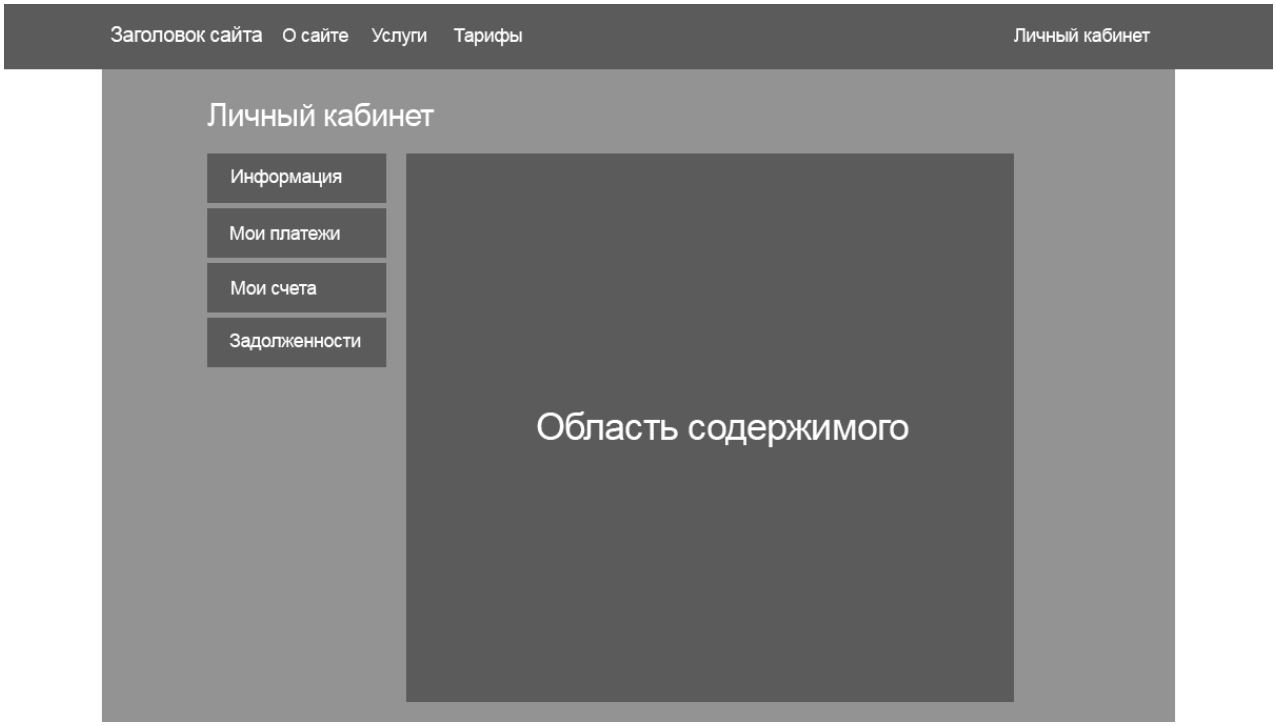

Рисунок 25 – Макет личного кабинета абонента

Область личного кабинета разделена на несколько вкладок:

– «Информация» (данные абонента);

– «Мои платежи» (список платежей абонента);

– «Мои счета» (счета и квитанции, доступные абоненту);

– «Задолженности» (отображает задолженность по оплате услуг при ее наличии).

При выборе каждой вкладки меняется содержимое в соответствующей области. Данный макет отражает функциональные возможности абонентов, определенные ранее при анализе вариантов использования.

Страница личного кабинета сотрудника внешне схожа со страницей личного кабинета абонента, однако название и содержимое ее вкладок отличается в соответствии с функциональными требованиями, определенными ранее для пользователей-сотрудников.

На странице присутствуют следующие вкладки:

– «Информация» (данные сотрудника);

– «Платежи» (список платежей и меню добавления платежа»);

– «Отчеты» (страница формирования отчетов по объему предоставленных услуг);

– «Заявки» (список заявок абонентов на подключение/отключение/смену тарифа);

– «Задолжниками» (страница с возможностями просмотра задолжников и выполнения месячных расчетов по тарифам).

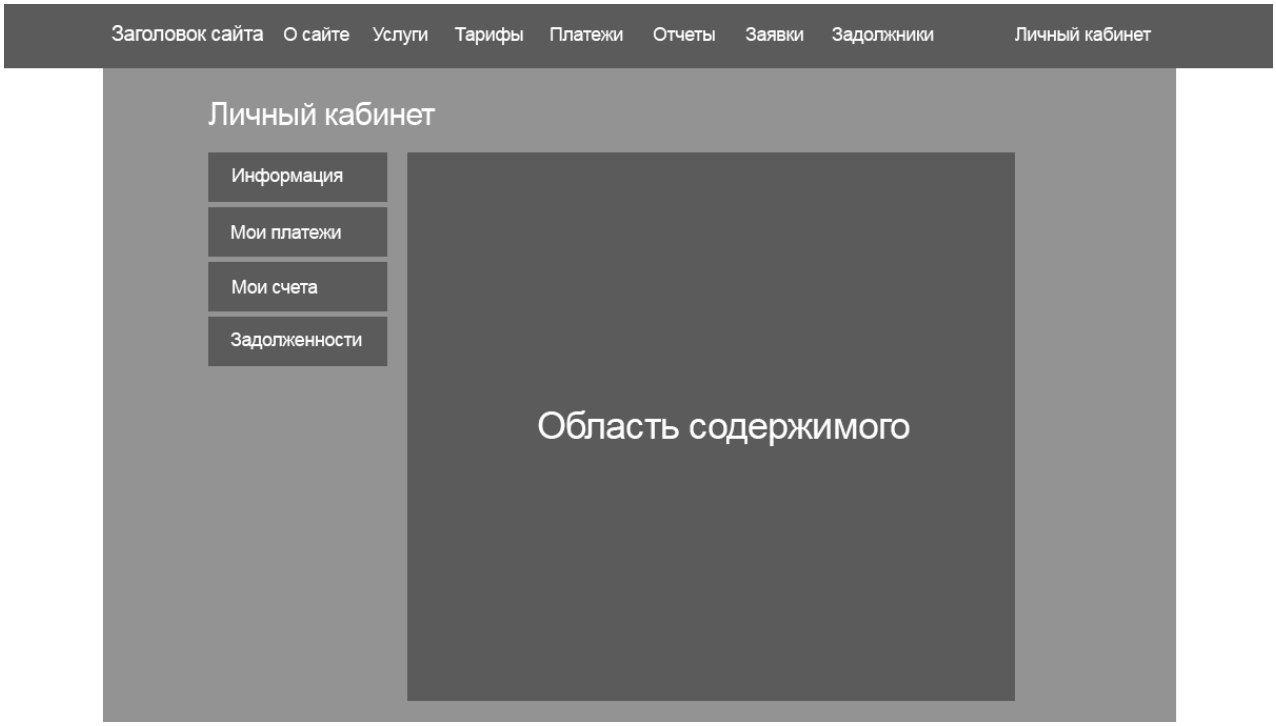

Рисунок 26 – Макет личного кабинета сотрудника

Представленные выше макеты похожи по структуре и отражают примерное расположение элементов и наполнение страницы.

3 Планирование разработки и оценка бюджета

3.1 Календарное и ресурсное планирование

Календарное и ресурсное планирование проекта является необходимым этапом в процессе проектирования и разработки автоматизированной системы. Планирование помогает контролировать ход выполнения работ по созданию информационной системы.

Календарный план проекта определяет продолжительность работ и их взаимозависимость. Диаграмма Ганта демонстрирует в наглядной графической форме такие показатели, как:

- сроки выполнения работ по проекту;
- длительность этапов;
- перечень необходимых ресурсов и т. д.

Кроме списка работ календарное планирование включает разработку ресурсной модели проекта. Среди всех ресурсов можно выделить несколько основных категорий:

- трудовые;
- материальные;
- невозобновляемые.

Ресурсы оказывают различное влияние на их использование и затраты, которые формируют бюджет проекта. Для получения плана-графика проекта с оптимальным использованием ресурсов и минимальными сроками проведем календарное планирование. Составим план стадий и этапов создания web-приложения.

Для составления плана будет использован Microsoft Project. Проект был начат 3 апреля 2023 года.

План, содержащий подробную информацию о наименовании и продолжительности (а также дата начала и дата окончания) стадий и этапов разработки (задач), а также лиц, выполняющих их представлен на рисунке 27. Всего при планировании разработки было выделено 13 задач.

Название задачи	Длительности	Начало	Окончание	Предшественники	Названия ресурсов
Начало проекта	0 дней	Пн 03.04.23	Пн 03.04.23		
Анализ предметной области и требований к системе	**7 дней**	**Пн 03.04.23**	**Вт 11.04.23**		
Изучение предметной области	1 день	Пн 03.04.23	Пн 03.04.23	1	Компьютер[1];Интернет;Аналитик
Анализ программного обеспечение аналогичного назначения	2 дней	Вт 04.04.23	Ср 05.04.23	3	Аналитик;Интернет;Компьютер[1]
Исследование вариантов использования	1 день	Чт 06.04.23	Чт 06.04.23	4	Аналитик;Интернет;Компьютер[1]
Анализ требований	1 день	Пт 07.04.23	Пт 07.04.23	5	Аналитик;Интернет;Компьютер[1]
Постановка задачи	2 дней	Пн 10.04.23	Вт 11.04.23	6	Аналитик;Интернет;Компьютер[1]
Проектирование программных средств	**13 дней?**	**Ср 12.04.23**	**Пт 28.04.23**	**7**	
Проектирование архитектуры	2 дней	Ср 12.04.23	Чт 13.04.23	7	Аналитик[50%];Программист (Backend)[50%];Интернет;Компьютер[1]
Проектирование алгоритмов	2 дней	Пт 14.04.23	Пн 17.04.23	9	Программист (Backend);Компьютер[1];Интернет
Проектирование структур данных	4 дней	Вт 18.04.23	Пт 21.04.23	10	Программист (Backend);Компьютер[1];Интернет
Проектирование пользовательского интерфейса	5 дней	Пн 24.04.23	Пт 28.04.23	11	Программист (Frontend)[50%];Дизайнер[50%];Компьютер[2];Интернет
Реализация программных средств	**13 дней**	**Пн 01.05.23**	**Ср 17.05.23**	**12**	
Кодирование web-приложения	8 дней	Пн 01.05.23	Ср 10.05.23	12	Программист (Backend)[50%];Программист (Frontend)[50%];Компьютер[2];Интернет
Развертывание web-приложения	5 дней	Чт 11.05.23	Ср 17.05.23	14	Программист (Backend);Компьютер[1];Интернет
Анализ качества программного продукта	**3 дней**	**Чт 18.05.23**	**Пн 22.05.23**	**15**	
Тестирование приложения	2 дней	Чт 18.05.23	Пт 19.05.23	15	Тестировщик[50%];Компьютер[3];Интернет;Программист (Backend)[25%];Программист (Frontend)[25%]
Метрики кода	1 день	Пн 22.05.23	Пн 22.05.23	17	Тестировщик;Компьютер[1];Интернет
Конец проекта	0 дней	Вт 23.05.23	Вт 23.05.23	18	

Рисунок 27 – Список задач проектирования

Не последнее значения занимает и использование ресурсов, а также расчет их стоимости (рисунок 28). Подобное планирование зачастую критически необходимо для успешного выполнения проекта.

	Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка
1	Программист (Backend)	Трудовой		ПБ		100%	600,00 ₽/ч
2	Программист (Frontend)	Трудовой		ПФ		100%	520,00 ₽/ч
3	Дизайнер	Трудовой		Д		100%	380,00 ₽/ч
4	Тестировщик	Трудовой		Т		100%	420,00 ₽/ч
5	Аналитик	Трудовой		А		100%	400,00 ₽/ч
6	Компьютер	Материальный		К			99 780,00 ₽
7	Принтер	Материальный		П			25 120,00 ₽
8	Бумага	Материальный		Б			320,00 ₽
9	Интернет	Затраты		И			

Рисунок 28 – Список ресурсов проекта

После назначения задач была получена полноценная диаграмма Ганта. На ней наглядно видно связи между задачами, используемые ресурсы и даты начала и окончания. Данная диаграмма приведена далее на рисунке 29.

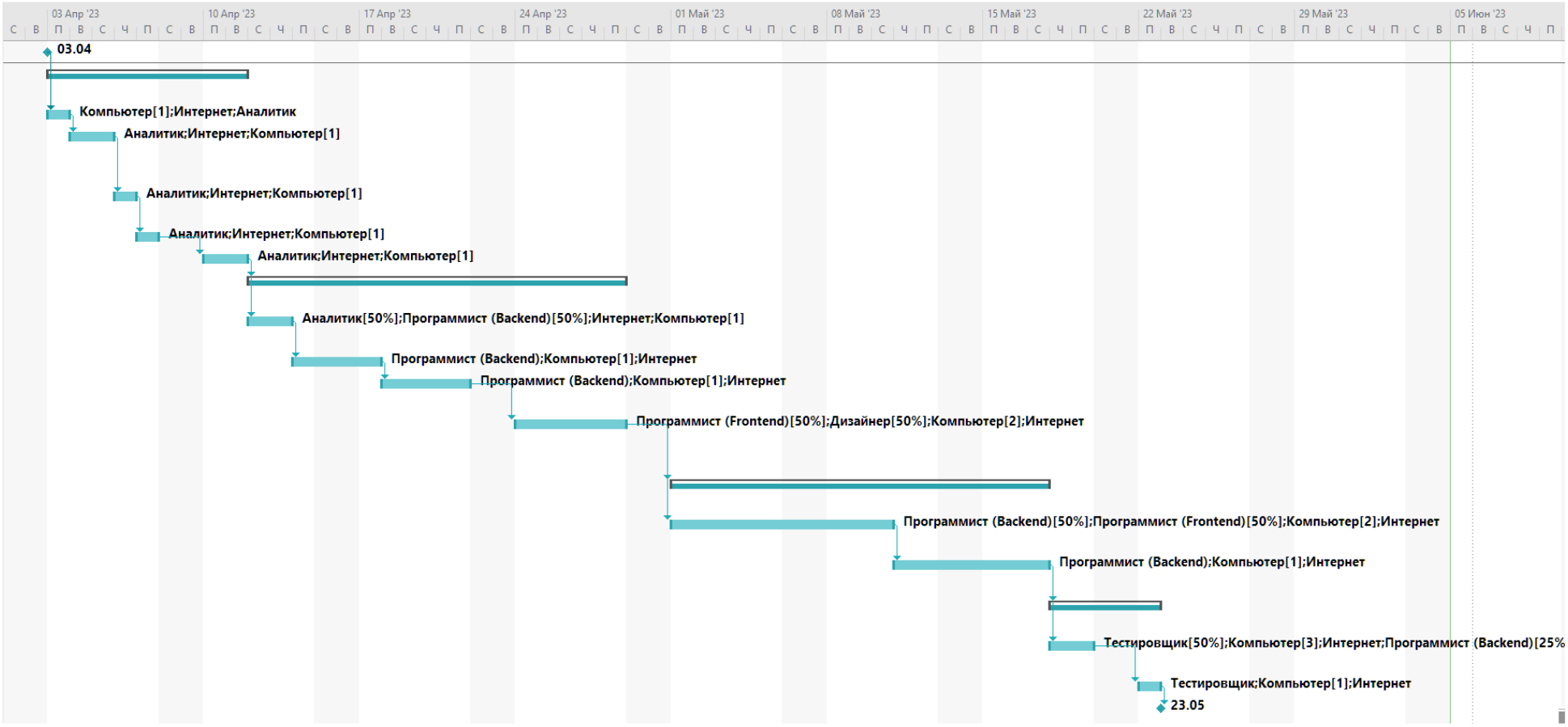

Рисунок 29 – Диаграмма Ганта для проекта проектируемого приложения

В результате анализа задач и этапов, а также их продолжительности, получилось выяснить, что проектирование и разработка web-приложения займет 288 часов.

3.2 Расчет бюджета

Для корректного расчета бюджета необходимо учесть следующие факторы:

- фонды основной и дополнительной заработной платы;
- социальные выплаты;
- материалы;
- амортизацию;
- прочие затраты.

Вначале проведем расчеты амортизационных отчислений. Амортизацией называют перенос стоимости основной части основных производственных фондов на стоимость готовой продукции, необходимый для равномерного распределения расходов.

Стоимость соответствующих показателей приведена в таблице 15.

Таблица 15 – Амортизационные отчисления

№	Наименование оборудования	Стоимость, руб.	Срок полезного использования, лет	Норма амортизации, %	Амортизация, руб./год
1	Компьютер с ПО: MS Office 2019; MS Windows 11	99780	5	20	19956
2	Принтер HP Laser 135w	25120	5	20	5024
3	Транспортно-заготовительные расходы (1%)	1249			
	Итого	126149			26229

Расчет бюджета проекта поможет убедиться в рациональности проектирования и разработки программного продукта.

Для расчета амортизационных отчислений используют следующую формулу:

$$А = С_{пер} * НА,$$

где А – амортизационные отчисления (руб.);

$С_{пер}$ – первоначальная стоимость основных средств (руб.);

НА – норма амортизации.

Норма амортизации рассчитываются по формуле:

$$НА = \frac{1}{СПИ},$$

где СПИ – срок полезного использования основных средств.

Для используемых ресурсов (принтера и компьютера) СПИ составляет 5 лет.

Рассчитаем их норму амортизации и амортизационные отчисления:

$$НА_к = \frac{1}{5} = 0{,}2,$$

$$НА_п = \frac{1}{5} = 0{,}2,$$

$$А_к = 99780 * 0{,}2 = 19956,$$

$$А_к = 25120 * 0{,}2 = 5024,$$

$$А_г = 19956 + 5024 + 1249 = 26229,$$

$$А_м = \frac{26229}{12} = 2185{,}75,$$

$$А_{итог} = 2185{,}75 * 1 = 2185{,}75.$$

Расчет бюджета проекта будет вестись за один месяц.

Затраты на материалы (3 пачки бумаги) с учетом ТЗР составят:

$$материалы = (3 * 320) * 101\% = 969{,}6 \text{ (руб.)}.$$

Далее рассчитаем фонд основной и дополнительной заработной платы (вознаграждение за труд в зависимости от квалификации).

В таблице 16 приведена трудоемкость определенных ранее задач, а также данные расчетов фондов заработной платы.

Далее будет рассмотрен расчет оплаты труда на примере расчета заработной платы аналитика проекта.

Таблица 16 – Расчет основной и дополнительной заработной платы

Исполнитель	Трудоемкость, часов	Ставка, руб./час	Основная заработная плата, руб.	Дополнительная заработная плата, руб.	Социальные выплаты, руб.
Аналитик	64	400	44828,16	4482,82	14793,29
Тестировщик	16	420	11206,98	1120,70	3698,30
Дизайнер	20	380	14008,73	1400,87	4622,88
Программист (Frontend)	56	520	39224,43	3922,44	12944,06
Программист (Backend)	132	600	92457,59	9245,76	30511,00
Итого	288		201725,88	20172,59	66569,54

Оплата за труд рассчитывается следующим образом:

$$ОЗП = С_ч * Т_р,$$

где $Т_р$ – трудоемкость;

$С_ч$ – тарифная ставка.

Тарифная ставка рассчитывается по формуле:

$$С_ч = \frac{(МРОТ * Т_к)}{160},$$

где МРОТ – минимальный размер оплаты труды = 16242 (руб.);

$Т_к$ – тарифный коэффициент;

160 – количество часов в месяц по ТК РФ.

$$Тк_{ан} = 6,9,$$

$$Сч_{ан} = \frac{(16242 * 6,9)}{160} = 700,44 \text{ (руб./час)},$$

$$ОЗП_{ан} = 700,44 * 64 = 44828,16 \text{ (руб.)}.$$

Мотивация труда оценивается по дополнительной заработной плате, рассчитывающейся по формуле:

$$ДЗП = 10\% * ОЗП,$$

$$ДЗП_{ан} = 10\% * 44828,16 = 4482,82 \text{ (руб.)}.$$

После этого необходимо рассчитать социальные выплаты (СВ), состоящие из фонда социального страхования (ФСС, 2,9%), пенсионного фонда РФ (ПФР, 22%) и фонда обязательного медицинского страхования (ФОМС, 5,1%).

$$СВ = 30\% * (ОЗП + ДЗП),$$

$$СВ = 30\% * (44828{,}16 + 4482{,}82) = 14793{,}29 \text{ (руб.)}.$$

Последняя статья расходов – расходы на электроэнергию (ЭЭ) и доступ в Интернет.

$$\text{Прочие затраты} = ЭЭ + \text{Интернет}.$$

ЭЭ рассчитывается по формуле:

$$ЭЭ = T_c * W * T_{раб},$$

где T_c = стоимость 1 кв/ч = (4,24 руб.);

W – энергомощность компьютера = 400 Вт/ч = 0,4 (кВ/ч);

$T_{раб}$ – время создания (программный продукт разрабатывался 288 часов).

$$ЭЭ = 4{,}24 * 0{,}4 * 288 = 488.45 \text{ (руб.)}.$$

Работа над программным продуктом велась 288 часов, но услуга доступа в Интернет оплачивается помесячно. Стоимость данной услуги равняется 650 руб./месяц.

Так как расчет ведется за один месяц, расходы на интернет составляют 650 рублей.

$$\text{Интернет} = 650 * 1 = 650 \text{ (руб.)},$$

$$\text{прочие затраты} = 488{,}45 + 650 = 1138{,}45 \text{ (руб.)}.$$

После того, как все статьи расходов были рассчитаны, была составлена таблица, отражающая структуру затрат, а также ее диаграмма.

Структура затрат представлена в таблице 17.

Таблица 17 – Структура затрат на проектирование и разработку

Наименование статьи расходов	Затраты, руб.	%
Амортизация	2185,75	0,75
Материалы	969,69	0,33

Проодолжение таблицы 17

Фонд основной и дополнительной заработной платы	221 898,47	75,79
Социальные выплаты	66 569,54	22,74
Прочие	1138,45	0,39
Итого	292 761,90	

Диаграмма структуры затрат представлена на рисунке 30.

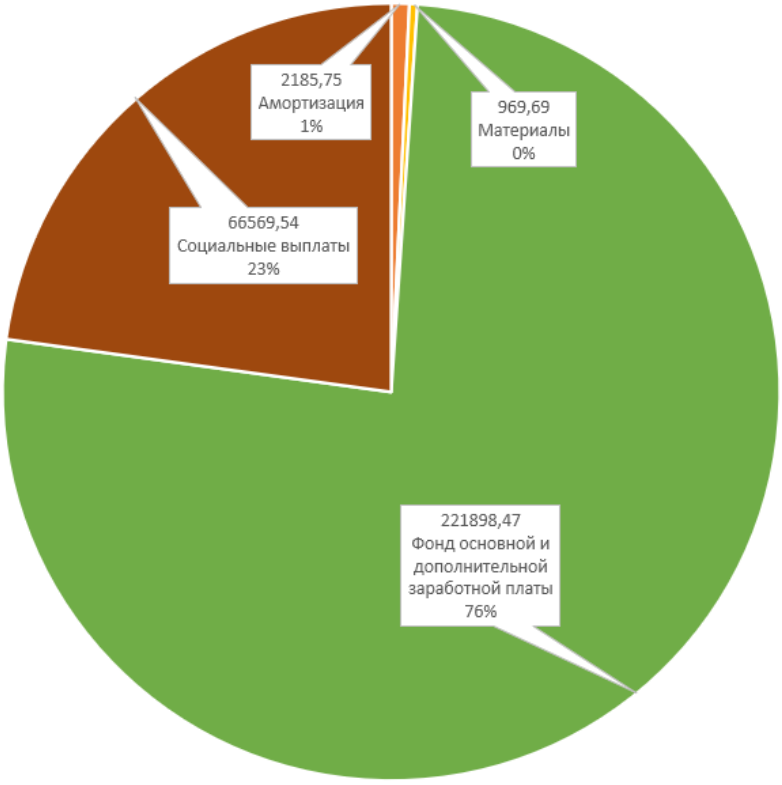

Рисунок 30 – Диаграмма структуры затрат на проектирование и разработку

В результате анализа структуры расходов можно выделить основную статью расходов (фонды заработной платы), а также рассчитать общую сумму затрат: проектирование и разработка программного продукта «Web-приложение системы учета абонентских платежей для городской телефонной сети» составит 292761,90 руб.

4 Программная реализация системы учета абонентских платежей

4.1 Кодирование системы учета абонентских платежей

Спроектированное приложение было разработано в соответствии с определенными заранее требованиями. Как было отмечено ранее, серверная часть должна быть разработана в соответствии с шаблоном MVC.

Для начала рассмотрим, как связаны модули разработанного приложения. Было выделено 4 модуля:

– Domain;
– Infrastructure;
– Application;
– Api.

Данные модули также отражают структуру программного решения в среде разработки Visual Studio (рисунок 31).

Под каждый из модулей выделен отдельный проект, входящий в решение.

Первый модуль (Domain) представляет домен или модель предметной области. В проекте данного модуля содержатся следующие папки:

– Entities (папка с классами сущностей);
– Utilities (папка со вспомогательными методами);
– Extensions (папка с классами расширения).

Также в него входят следующие файлы с исходным кодом, соответствующие сущностям предметной области:

– User.cs (сущность «Пользователь»);
– SubscriberPhone.cs (сущность «Абонентский телефон»);
– PhoneType.cs (сущность «Тип телефона»);
– PhoneStatus.cs (сущность «Статус телефона»);
– PhoneStatusEnum.cs (перечисление значений статусов телефона);
– StatusHistoryEntry.cs (сущность «Запись истории статусов»);
– Tariff.cs (сущность «Тариф»);
– TariffHistoryEntry.cs (сущность «Запись истории тарифов»);

– Payment.cs (сущность «Платеж»);

– Request.cs (сущность «Запрос»);

– ConnectRequest.cs (сущность «Заявка на подключение»);

– ChangeTariffRequest.cs (сущность «Заявка на смену тарифа»).

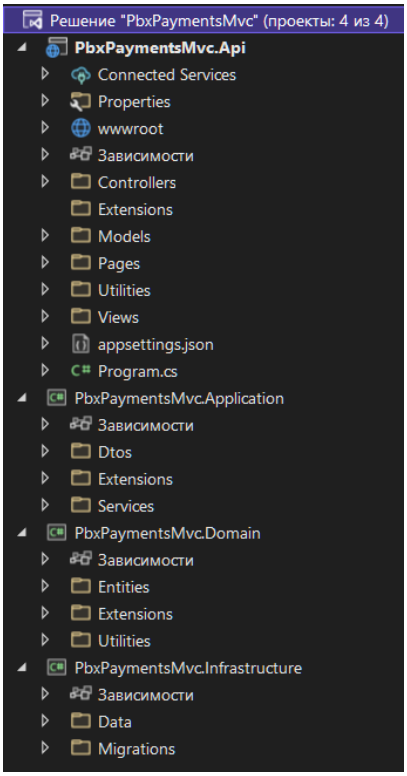

Рисунок 31 – Структура решения разработанного приложения

Следующий модуль – Infrastructure. Как следует из названия, это слой инфраструктуры приложения. Данный модуль отвечает за связь приложения со сторонними сервисами, такими как база данных или сервис электронной почты. В данном случае здесь находятся классы конфигурации сущностей (папка Data/Config), а также класс контекста базы данных из Entity Framework. Кроме того, именно в этом проекте в папке Migrations располагаются миграции БД, выполняющие создание и изменение структуры базы данных в соответствии с классами сущностей и их конфигурациями из Entity Framework.

Данный модуль содержит следующие файлы исходного кода:

– PbxContext.cs (контекст БД);

– RequestEntityTypeConfiguration.cs (класс-конфигуратор сущности «Заявка»);

– ConnectRequestEntityTypeConfiguration.cs (класс-конфигуратор сущности «Заявка на подключение»);

– PhoneStatusEntityTypeConfiguration.cs (класс-конфигуратор сущности «Статус телефона»);

– PhoneTypeEntityTypeConfiguration.cs (класс-конфигуратор сущности «Тип телефона»);

– StatusHistoryEntryEntityTypeConfiguration.cs (класс-конфигуратор сущности «Запись истории статуусов»);

– SubscriberPhone EntityTypeConfiguration.cs (класс-конфигуратор сущности «Абонентский телефон»);

– TariffEntityTypeConfiguration.cs (класс-конфигуратор сущности «Тариф»);

– UserEntityTypeConfiguration.cs (класс-конфигуратор сущности «Пользователь»).

Далее идет модуль Application. Он представляет слой бизнес-логики приложения. В проекте данного модуля находятся следующие папки:

– Dtos (классы объектов передачи данных);

– Extensions (классы с методами расширения);

– Services (классы-сервисы с бизнес-логикой).

В данном модуле объявляются и определяются объекты передачи данных (DTO), используемые для передачи информации между отдельными частями web-приложения.

Использование данных объектов вместо доменных объектов позволяет уменьшить связанность модулей: таким образом, изменение объекта модели предметной области не отразится (не обязательно отразится) на объекте передачи данных и наоборот. Также это позволяет скрыть важную информацию при передаче на клиент (к примеру, пароль при передаче объекта пользователя).

Также здесь расположены интерфейса сервисов, предоставляющих методы для совершения бизнес-процессов (к примеру, добавления платежей и формирования отчетов), а также классы с реализацией данных интерфейсов.

В данном модуле находятся следующие файлы с исходным кодом:

– ConnectRequestDto (DTO сущности «Заявка на подключение»);

– PaymentDto (DTO сущности «Платеж»);

– PhoneStatusDto (DTO сущности «Статус телефона»);

– PhoneTypeDto (DTO сущности «Тип телефона»);

– StatusHistoryEntryDto (DTO сущности «Запись истории статусов»);

– SubscriberPhoneDto (DTO сущности «Абонентский телефон»);

– TariffDto (DTO сущности «Тариф»);

– UserDto (DTO сущности «Пользователь»);

– ClaimsPrincipalExtensions.cs (класс расширения для механизма аутентификации);

– DiExtensions.cs (класс расширения для регистрации сервисов в контейнере зависимостей);

– IConnectRequestService.cs и ConnectRequestService.cs (интерфейс и реализация сервиса заявок на подключение);

– IFeeService.cs и FeeService.cs (интерфейс и реализация сервиса оплаты услуг);

– IPaymentService.cs и ConnectRequestService.cs (интерфейс и реализация сервиса абонентских платежей);

– IPhoneTypeService.cs и ConnectRequestService.cs (интерфейс и реализация сервиса управления типами телефонов);

– ISubscriberPhoneService.cs и ConnectRequestService.cs (интерфейс и реализация сервиса управления абонентскими телефонами);

– ITariffService.cs и ConnectRequestService.cs (интерфейс и реализация сервиса управления тарифами);

– IReportService.cs и ReportService.cs (интерфейс и реализация сервиса формирования отчетов);

– IUserService.cs и UserService.cs (интерфейс и реализация сервиса управления пользователями).

Последний модуль (Api) содержит непосредственную реализацию web-приложения. Проект данного модуля содержит следующие папки:

– wwwroot (папка с ресурсами HTML-разметки, таблицы стилей и пользовательских сценариев на JavaScript, а также изображений и иконок);

– Utilities (папка со вспомогательными сервисами);

– Pages (папка со страницами аутентификации и авторизации ASP.NET Identity на движке представлений Razor);

– Models (папка с классами моделей);

– Views (папка с представлениями);

– Controllers (папка с контроллерами).

Как можно заметить, именно здесь реализуется само web-приложение (по MVC-шаблону).

В папке wwwroot содержатся следующие ресурсы:

– site.css (таблица стилей сайта);

– error-modal.js (сценарии на JavaScript модального окна ошибок);

– error-modal.js (сценарии на JavaScript модального окна отчетов);

– JavaScript-файлы фреймворка Bootstrap 5;

– JavaScript-файлы фреймворка jQuery;

– favicon.ico (иконка Web-сайта, отображаема в браузере на стороне клиента).

В папке Utilities содержатся следующие файлы:

– AccountNavigationPages.cs (вспомогательный класс для навигации по личному кабинету);

– RoleHelper (вспомогательный класс, определяющий роли пользователей).

В папке Pages содержатся следующие файлы представлений:

– _ValidationScriptsPartial.cshtml (частичное представление с сценариями валидации пользовательских данных);

– _ViewImports.cshtml (частичное представление со списком импортируемых пространств имен);

– _ViewStart.cshtml (частичное представление, инициализирующее макет личного кабинета);

– _StatusMessage.cshtml (частичное представление сообщений статуса);

– AccessDenied.cshtml (страница «Доступ запрещен»);

– Login.cshmtl (страница входа);

– Register.cshtml (страница регистрации);

– RegisterConfirmation.cshtml (страница подтверждения регистрации);

– ResendEmailConfirmation.cshtml (страница подтверждения повторной отправки письма);

– ResetPassword.cshtml (страница смены пароля);

– ResetPasswordConfirmation.cshtml (страница подтверждения смены пароля);

– Logout.cshtml (страница выхода);

– Lockout.cshtml (страница временной блокировки после череды неудачных попыток входа);

– ForgotPassword.cshtml (страница восстановления пароля);

– ForgotPasswordConfirmation.cshtml (страница подтверждения восстановления пароля);

– ConfirmEmail.cshtml (страница подтверждения Email);

– ConfirmEmailChange.cshtml (страница подтверждения смены Email);

– _AccountNav.cshtml (частично представление с панелью навигации личного кабинета);

– _Layout.cshtml (частично представление макета личного кабинета);

– ChangePassword.cshtml (страница смены пароля в личном кабинете);

– Email.cshtml (страница настроек Email в личном кабинете);

– Index.cshtml (главная страница личного кабинета).

Оставшиеся файлы представляют собой связанные между собой части MVC-структуры (модели из папки Models, представления из папки Views и контроллеры из папки Controllers). Для их наглядного представления воспользуемся диаграммой компонентов (рисунок 32).

Также в данном модуле находится главный файл программы, в котором происходит выполнение процесса конфигурации и непосредственно запуск серверной части web-приложения.

В нем так же происходит построение дерева (графа) зависимостей и его разрешение. Главный файл программы выбран для этой цели не случайно. Дело в том, что корень композиции должен находиться как можно ближе к точке входа приложения, и это должно быть единственное место, где применяется контейнер зависимостей.

Использование контейнера зависимостей в других частях программы в большинстве случаев является злоупотреблением. Подобный архитектурный шаблон называется локатором сервисов (Service Locator) и является анти-паттерном. При его использовании у объектов появляется доступ к неограниченному набору зависимостей, при этом сам локатор не гарантирует факт их доставки, кроме того, зависимости самих объектов могут стать неявными, становится тяжелее изменить зависимость в зависимости от ее контекста. Все это негативно влияет на качество кода, повышая сложность его дальнейшей поддержки и тестирования [16].

Как можно заметить на данной диаграмме, в приложении 7 основных связок модель/представление/контроллер:

– Главная страница (контроллер HomeController.cs, представление Index.cshtml);

– Страница платежей (контроллер PaymentsController.cs, представления Index.cshtml, Details.cshtml, Edit.cshtml, Delete.cshtml, Create.cshtml, _PaginationNav.cshtml);

– Страница заявок (контроллер ConnectRequestController.cs, модели ConnectRequestIndexViewModel.cs, ConnectRequestViewModel.cs и

AcceptConnectRequestViewModel.cshtml, представления Index.cshtml, Details.cshtml, Create.cshtml, Accept.cshtml, Reject.cshtml, _PaginationNav.cshtml);

– Страница услуг (контроллер ServicesController.cshtml, представление Index.cshtml);

– Страница тарифов (контроллер TariffsController.cshtml, модеи TariffsIndexViewModel.cs, TariffViewModel.cs, представления Index.cshtml, Details.cshtml Edit.cshtml);

– Страница отчетов (контроллер ReportsController.cs, модели ReportPeriodViewModel.cs, ReportViewModel.cs, представления Index.cshtml, Create.cshtml, ErrorModal.cshtml).

Данная диаграмма является укрупненной.

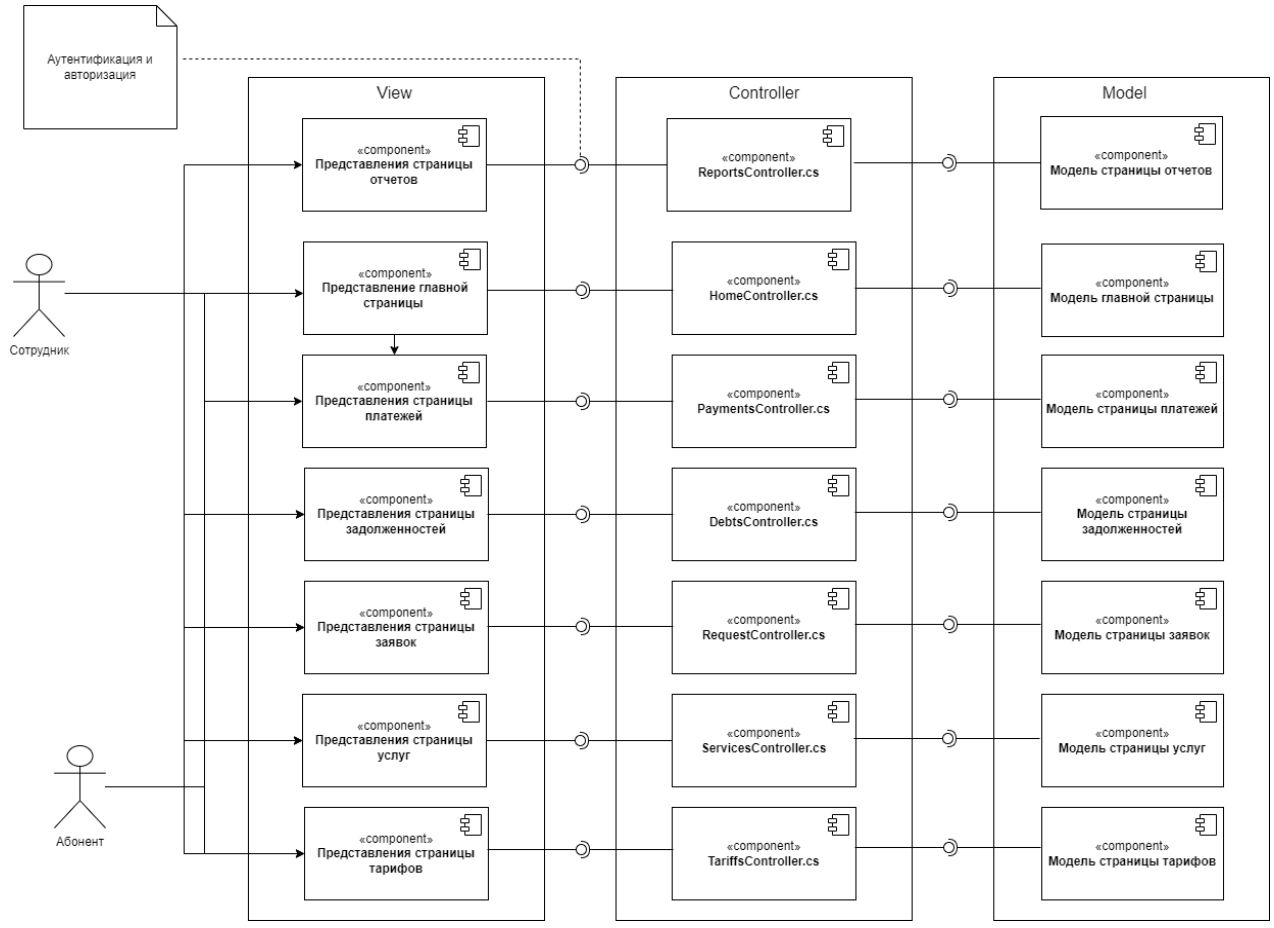

Рисунок 32 – Диаграмма компонентов разработанного приложения

Рассмотрим подробнее, как устроена одна из частей, отображенная на диаграмме компонентов – представления, контроллер и модель платежей.

Для этого воспользуемся диаграммой классов (рисунок 33).

На диаграмме отображены следующие классы:

– Payment.cs – класс доменной сущности «Платеж»;

– SubscriberPhone.cs – класс доменной сущности «Абонентский телефон».

– PaymentDto.cs – класс объекта передачи данных «Платеж»;

– SubscriberPhoneDto.cs – класс объекта передачи данных «Абонентский телефон»;

– PaymentViewModel.cs – класс модели сущности «Платеж» (представляет часть модели в контексте MVC);

– PaymentsIndexViewModel.cs – класс модели страницы со списком платежей (представляет часть модели в MVC);

– IPaymentService.cs – интерфейс сервиса для управления платежами;

– ISubscriberPhoneService.cs – интерфейс сервиса для управления абонентскими телефонами;

– PaymentService.cs – реализация интерфейса IPaymentService;

– SubscriberPhoneService.cs – реализация интерфейса ISubscriberPhoneService;

– PaymentsController.cs – контроллер, управляющий запросами со стороны клиента (его методы являются обработчиками HTTP запросов, отсылающие обратно ответ в виде cshtml-представлений);

– Index.cshtml – представление страницы со списком платежей;

– Details.cshtml – представление страницы конкретного платежа;

– Create.cshtml – представление страницы добавления платежа;

– Edit.cshtml – представление страницы редактирования платежа;

– Delete.cshtml – представление страницы удаления платежа.

Далее представлен пример пользовательского интерфейса. На рисунке 34 отображена страница списка платежей. Остальные страницы представлены в приложении Р.

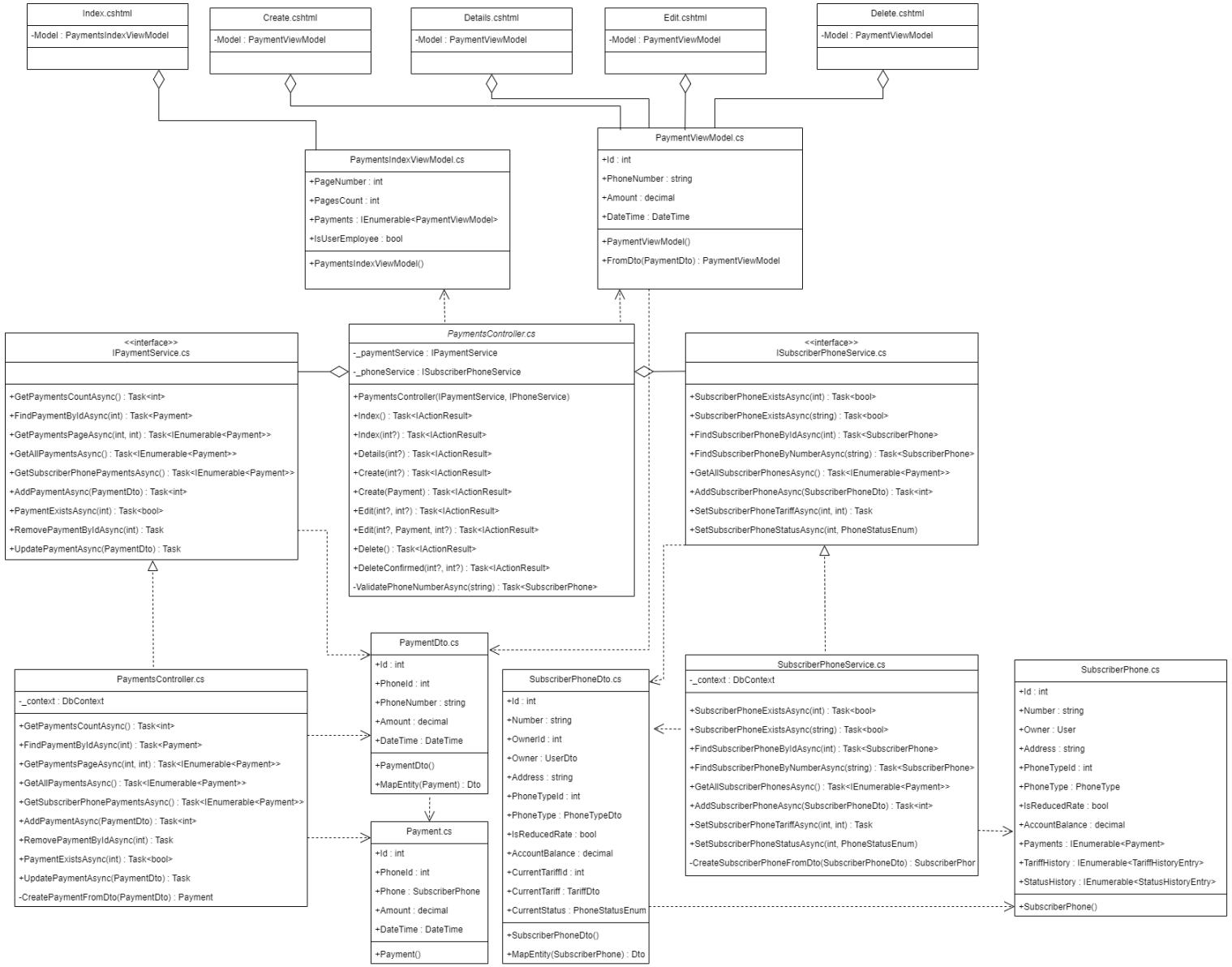

Рисунок 33 – Диаграмма классов одного из MVC-компонентов разработанного приложения

Поскольку в системе может храниться большое количество платежей, для удобства просмотра больших списков, а также оптимизации запросов к серверу и базе данных, реализована пагинация (разбиение списка на страницы).

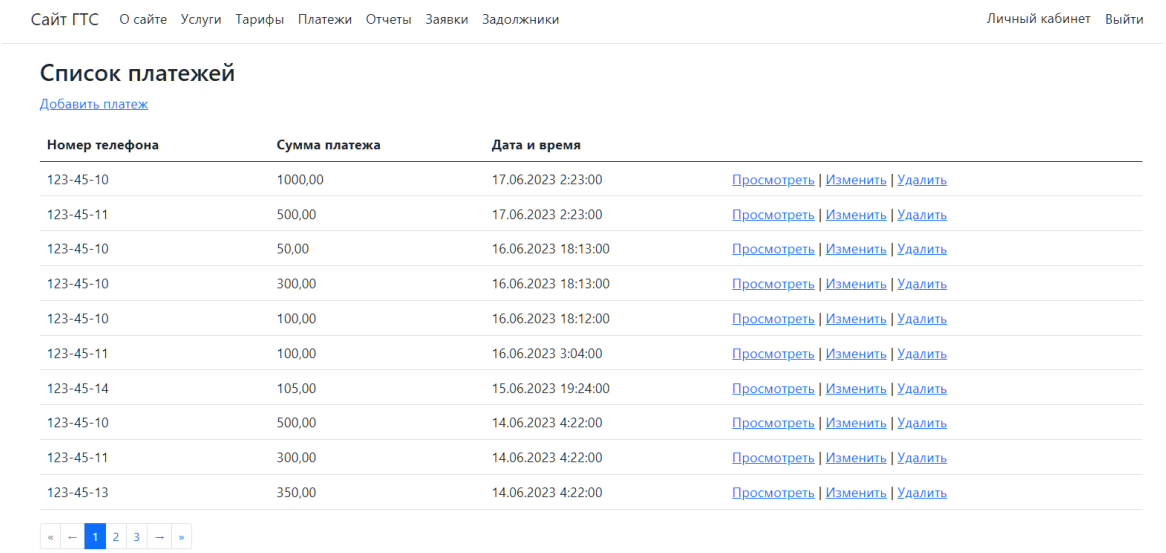

Рисунок 34 – Страница списка платежей разработанного приложения

Таким образом, был рассмотрен одна из семи MVC-компонент разработанного приложения. Остальные части обладают схожей структурой, так как были разработаны с тем же подходом.

4.2 Развертывание web-приложения

Как уже было неоднократно замечено, приложение логически разделено на серверную и клиентскую части: клиентской частью выступает веб-страница в браузере пользователя, в то время как серверная часть приложения разворачивается на удаленной машине.

Для работы web-приложения системы учета абонентских платежей необходим web-сервер с поддержкой платформы .NET 6, включающую среду исполнения CLR. Web-сервер также должен устанавливать соединение с MS SQL Server 2018, либо разворачивать ее локально.

Платформа .NET является кроссплатформенной и позволяет использовать одну из следующих операционных систем для установки серверной части:

– Windows 10/11;

– Windows Server;

– macOS;

– Linux.

В качестве клиента приложения выступает любой современный web-браузер для любых платформ.

Для установки и использования системы учета абонентских платежей необходимо поместить все файлы веб-приложения в папку домена веб-сервера и настроить подключение к базе данных.

На рисунке 35 представлена диаграмма развертывания разработанного веб-приложения.

На данной диаграмме развертывания показано, как связаны между собой обрабатывающие узлы, на которых выполняется разработанное web-приложение, а также, компоненты, размещенные в этих узлах.

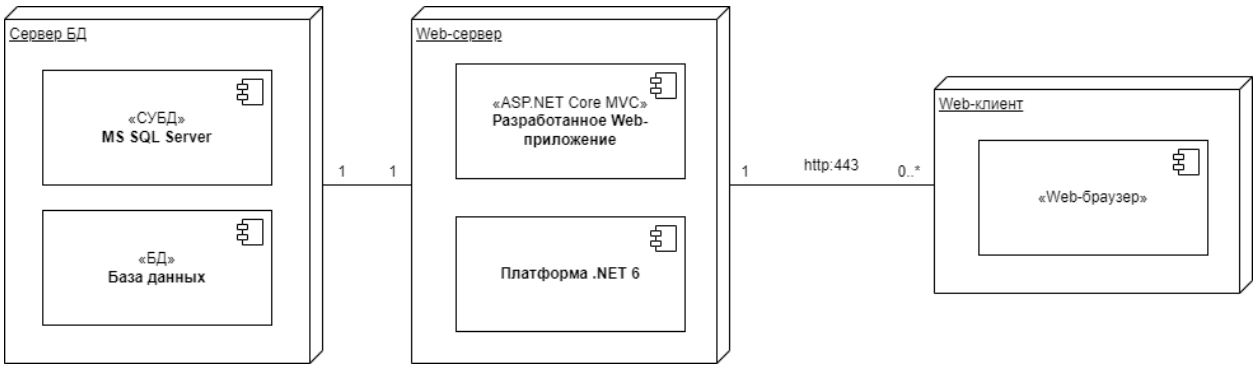

Рисунок 35 – Диаграмма развертывания веб-приложения

Соединение между клиентом и сервером устанавливается через порт 443 – это объясняется тем, что обмен данными осуществляется по защищенному HTTPS-протоколу.

В результате выполнения данного этапа было реализовано web-приложение системы учета абонентских платежей для городской телефонной сети. Исходный код его основных компонентов приведен в приложениях А – П.

5 Анализ качества программного продукта

5.1 Тестирование

Для того чтобы убедиться в корректности работы программного обеспечения и повысить качество программного продукта, обнаружив и исправив дефекты программного продукта, выполнятся его тестирование. Существуют различные методологии тестирования, например:

- функциональное тестирование;
- системное тестирование;
- регрессионное тестирование;
- модульное тестирование и т. п.

В качестве методики тестирования программных компонентов было выбрано модульное тестирование (или юнит-тестирование).

Модульное тестирование позволяет проверить на корректность отдельные модули исходного кода программы. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Кроме того, цель подобных тестов – обеспечение стабильной поддержки программного проекта, которое усложняется со временем. Причина этого заключается в том, что каждый раз, когда в коде проекта происходят изменения, увеличивается степень его программной энтропии (беспорядка в системе). Если не принять должных мер (например, постоянной чистки и рефакторинга), код постепенно усложняется и дезорганизуется. Исправление одной ошибки приводит к появлению новых ошибок, а изменение в одной части проекта нарушает работоспособность в нескольких других, и со временем код становится ненадежным. Кроме того, его становится все труднее вернуть в стабильное состояние. Тесты помогают справиться с этой тенденцией. Они становятся средством, которое обеспечивает защиту против большинства регрессий,

помогая удостовериться в том, что существующая функциональность работает даже после разработки новой функциональности или рефакторинга кода.

Также сама возможность покрытия кода тестами – хороший критерий определения качества этого кода, но он работает только в одном направлении. Это хороший негативный признак – он выявляет низкокачественный код с относительно высокой точностью. Если вдруг обнаружится, что код трудно протестировать, это верный признак того, что код нуждается в улучшении. Плохое качество обычно проявляется в сильной связности (tight coupling) кода; это означает, что части кода недостаточно четко изолированы друг от друга, что в свою очередь создает сложности с их раздельным тестированием.

Существуют также интеграционные тесты, проверяющие, как система работает в интеграции с внепроцессными зависимостями.

Тривиальный и переусложненный коды тестироваться не должны. Если все внепроцессные зависимости заменить тестовыми заглушками, то никакие зависимости не будут совместно использоваться между тестами, благодаря чему эти тесты останутся быстрыми и сохранят свою изоляцию друг от друга [17].

Тестовая заглушка или мок (mock) – общий термин, который описывает все разновидности фиктивных зависимостей в тестах. Тестовые заглушки используются прежде всего для упрощения тестирования; они передаются тестируемой системе вместо реальных зависимостей, настройка или сопровождение которых могут быть сопряжены с определенными сложностями.

Для данной программы юнит-тестирование проводилось с применением технологии MSTest, имеющей интеграцию с используемой средой разработки MS Visual Studio. Для создания тестовых заглушек зависимостей использовался изоляционный фреймворк Moq.

Всего было написано 52 теста покрывающие весь слой бизнес-логики web-приложения.

Представления не подвергались тестированию в виду того, что практически вся логика пользовательского интерфейса и вся бизнес-логика находится в классах контроллеров и сервисов; в представлениях лишь

осуществляется отправка запросов, обрабатываемых контроллерами, а также ввод данных форм, следовательно, такой код можно отнести к тривиальному и не нуждающемуся в тестировании.

Каждый тест проверяет правильность выполнения отдельной функции класса (на некоторые функции приходится несколько тестов). При тестировании проверяется корректность работы лишь тестируемого класса, поведение его зависимостей не проверяется.

Перед каждым тестом создаются тестовые заглушки зависимостей необходимого класса и передаются в его конструктор. Для заглушек сервисов и репозиториев при помощи Moq настроено возвращение тестовых значений и выполнение необходимой логики.

Для примера была взята таблица тестирования класса IPaymentService, отвечающего за управление абонентскими платежами (таблица 18).

Код для описанных выше тестов приведен в приложении Т, результаты всех 52 пройденных тестов представлены – в приложении С.

Таблица 18 – План тестирования класса PaymentService

№ п/п	Тестируемый метод	Тест	Результат	Тест пройден
1	FindPaymentByIdAsync	Получение платежа по идентификатору	Возвращен платеж, соответствующий идентификатору	+
2	FindPaymentByIdAsync	Получение платежа по несуществующему идентификатору	Возвращено нулевое значение	+
3	GetAllPaymentsAsync	Получение списка всех платежей	Возвращен список со всеми платежами	+
4	GetSubscriberPaymentsAsync	Получение списка всех платежей конкретного абонента	Возвращен список со всеми платежами, соответствующими заданному абоненту	+
5	GetSubscriberPaymentsAsync	Получение всех платежей у абонента, не совершавшего платежи	Возвращен пустой список	+

Продолжение таблицы 18

№ п/п	Тестируемый метод	Тест	Результат	Тест пройден
6	AddPaymentAsync	Добавление нового платежа.	Платеж добавлен, баланс телефона увеличен	+
7	RemovePaymentByIdAsync	Удаление существующего платежа по идентификатору	Удален платеж, соответствующий идентификатору	+
8	RemovePaymentByIdAsync	Удаление платежа по несуществующему идентификатору	Изменений нет	+
9	PaymentExistsAsync	Проверка существования платежа по идентификатору	Возвращено значение истины	+
10	UpdatePaymentAsync	Обновление данных платежа.	Данные платежа обновлены	+

Результаты пройденных тестов представлены на рисунке 36.

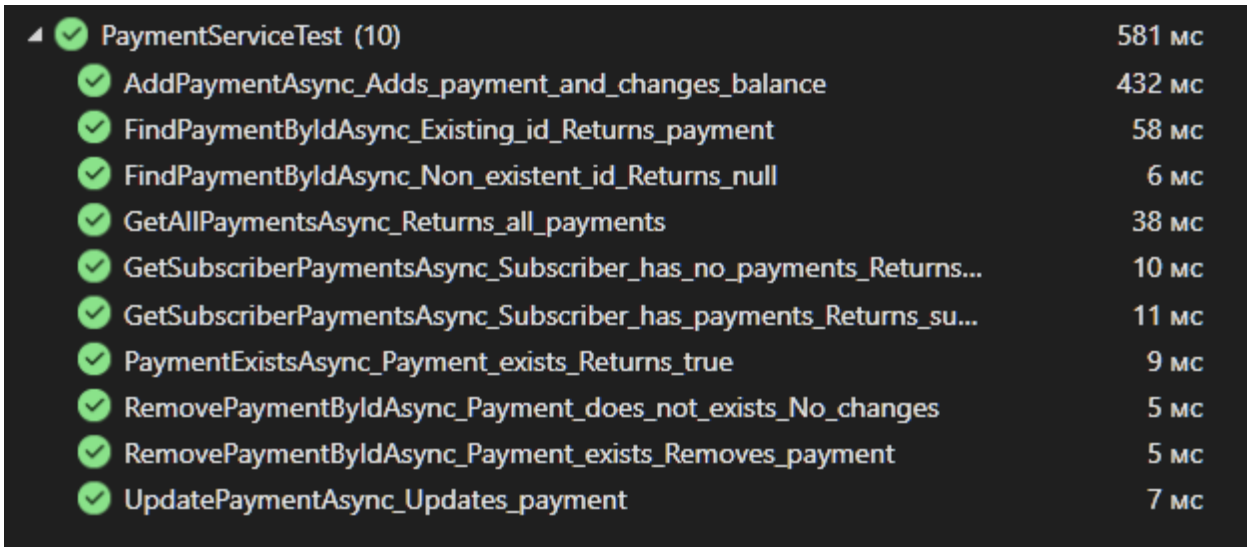

Рисунок 36 – Результаты тестирования класса PaymentService

Полученные результаты тестирования показали, что все результаты тестов соответствуют ожидаемым, следовательно, тестируемые модули работают корректно.

Проведение модульного тестирование позволило установить соответствие результатов работы web-приложения ожидаемым, что свидетельствует о корректности его функционирования.

5.2 Метрики кода

После реализации web-приложения необходимо убедиться в качестве написанного программного кода. Для этого воспользуемся специальными метриками.

Для разработанной программы были рассчитаны метрики кода в среде MS Visual Studio.

Ключевой метрикой кода в данном случае является индекс удобства поддержки (MI). Данная метрика представляет комплексный показатель качества кода и рассчитывается по формуле, зависящей от вычислительной сложности кода (HB, количество операторов), структурной сложности кода (CC, количество различных ветвей) и количества строк кода (LoC):

$$\mathrm{MI} = \mathrm{MAX}\left(0, (171 - 5{,}2 * \ln(\mathrm{HV}) - 0{,}23 * \mathrm{CC} - 16{,}2 * \ln(\mathrm{LoC})) * \frac{100}{171}\right).$$

После применения автоматизированного программного средства, встроенного в Visual Studio, были получены следующие показатели, представленные в таблице 19.

Значения данной метрики (от 0 до 100) показывают, насколько хорошо код поддерживается и насколько просто внести туда какие-либо изменения, а также насколько легко он читается.

При этом хорошим показателем считают числовые значения от 20 до 100.

Таблица 19 – Метрики кода разработанного web-приложения

Метрика	Значение
Размерно-ориентированные метрики	
Количество строк кода	23886
Логических строк кода	6753
Количество строк комментариев	266
Объектно-ориентированные метрики	
Количество классов	177
Количество методов	519

Продолжение таблицы 19

Метрика	Значение
Сложность кода	
Средняя цикломатическая сложность по классу	1,9

Помимо показателей, представленных выше, были получены показатели Maintainability Index для основных классов. Данные приведены в таблице 20.

В данном случае средним значением индекса удобства поддержки является 78, однако этот показатель не совсем точный, так как учитывает код, автоматически сгенерированный средством создания миграций.

Таблица 20 – Показатели качества кода компонентов приложения

№	Компонент	MI
1	DebtorsController	67
2	DebtsController	64
3	HomeController	89
4	PaymentsController	63
5	PaymentsEmployeeController	64
6	ReportsController	65
7	RequestsController	67
8	RequestsEmployeeController	61
9	ServicesController	100
10	TariffsController	67
11	AcceptConnectRequestViewModel	84
12	ConnectRequestsIndexViewModel	100
13	ConnectRequestViewModel	89
14	DebtorsIndexViewModel	100
15	DebtorViewModel	90
16	DebtsIndexViewModel	100
17	ErrorViewModel	95
18	ModalViewModel	93
19	PaymentsIndexViewModel	100
20	PaymentViewModel	81
21	PostModalViewModel	91

Продолжение таблицы 20

№	Компонент	MI
22	ReportPeriodViewModel	100
23	ReportViewModel	100
24	StatusMessageViewModel	94
25	TariffsIndexViewModel	100
26	TariffViewModel	81
27	PaginatedEntityViewModel	100
28	PaginatedViewModel<TEntity>	100
29	PaginatedViewModelBase	97
30	AccountNavigationPages	90
31	RoleHelper	93
32	PbxContext	99
33	PbxContextFactory	72
34	CallEntityTypeConfiguration	79
35	ConnectRequestEntityTypeConfiguraiton	83
36	PhoneStatusEntityTypeConfiguration	85
37	PhoneTypeEntityTypeConfiguration	85
38	StatusHistoryEntryEntityTypeConfiguration	75
39	SubscriberPhoneEntityTypeConfiguration	75
40	TariffEntityTypeConfiguration	84
41	UserEntityTypeConfiguration	69
42	ConnectRequestDto	90
43	PaymentDto	91
44	PhoneStatusDto	92
45	PhoneTypeDto	91
46	StatusHistoryEntryDto	91
47	SubscriberPhoneDto	89
48	TariffDto	90
49	UserDto	90
50	ClaimsPrincipalExtensions	70
51	DiExtensions	67
52	ConnectRequestService	74
53	DebtCalculationResult	94

Продолжение таблицы 20

№	Компонент	MI
54	FeeCalculationResult	95
55	FeeService	63
56	PaymentService	71
57	PhoneTypeService	81
58	ReportService	63
59	SubscriberPhoneService	68
60	TariffService	80
61	UserService	81
62	IConnectRequestService	100
63	IFeeService	100
64	IPaymentService	100
65	IPhoneTypeService	100
66	IReportService	100
67	ISubscriberPhoneService	100
68	ITariffService	100
69	IUserService	100
70	Call	93
71	ChangeTariffRequest	100
72	ConnectRequest	100
73	DisconnectRequest	100
74	Payment	95
75	PhoneStatus	100
76	PhoneStatusEnum	93
77	PhoneType	100
78	Request	100
79	StatusHistoryEntry	94
80	SubscriberPhone	90
81	Tariff	97
82	TariffHistoryEntry	96
83	User	90
84	PhoneStatusEnumExtensions	96
85	PhoneStatusEnumHelper	80

Окончание таблицы 20

№	Компонент	MI
	Средний показатель качества кода проекта	87,6

При исключении автоматически сгенерированного кода и кода модульных тестов из расчета итоговый результат повышается до 87,6, что является отличным показателем для основных компонентов приложения.

Далее вручную рассчитаем метрики цикломатической сложности для одного из методов. Для этого возьмём метод, который отвечает за обновление данных о задолженностях по абонентским платежам и составим граф потока и управления. Данный граф приведен на рисунке 37.

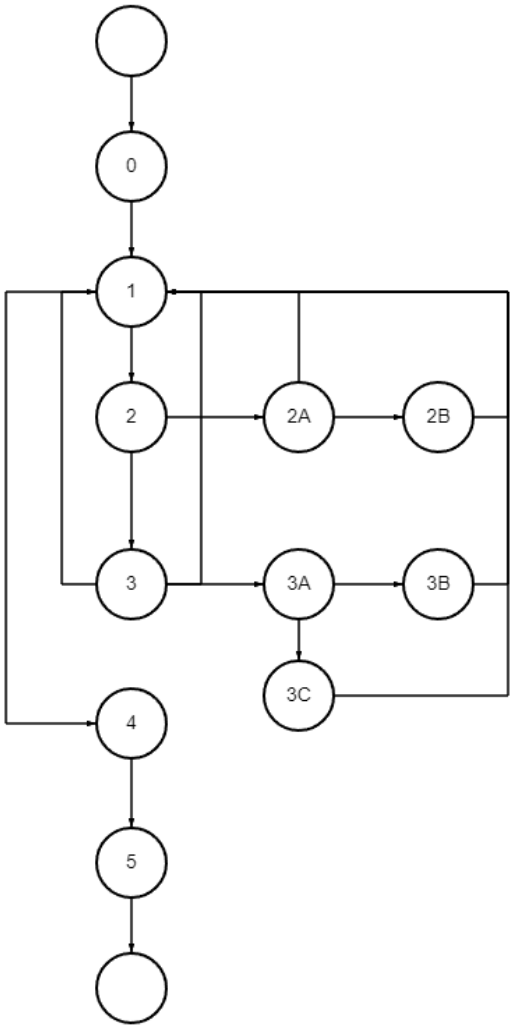

Рисунок 37 – Граф потока и управления метода обновления задолженностей

Код рассматриваемого метода приведен ниже (листинг 4).

Листинг 4 – Исходный код метода обновления задолженностей на языке C#

```
public DebtCalculationResult UpdateDebts()
{
     // Блок 0 - инициализация переменных
     decimal debtsTotal = 0;
     decimal debtsPaid = 0;
     decimal newDebt = 0;

     foreach (var phone in _phones)
     {
          // Блок 1 - переходим к следующему объекту в цикле
          // (либо завершаем цикл)

          // Блок 2 - проверяем, подключен ли абонент
          if (_statusService.IsPhoneConnected(phone))
          {
               // Блок 2А - проверяем, отрицательный ли баланс у
               // абонента
               if (phone.AccountBalance < 0)
               {
                    // Блок 2В - изменяем новую сумму
                    // задолженностей
                    // и назначаем статус задолженности абоненту
                    newDebt += -phone.AccountBalance;
                    _statusService.SetDebtStatus(phone);
               }
          }
          // Блок 3 - проверяем, является ли абонент должником
          else if (_statusService.IsPhoneDebtor(phone))
          {
               // Блок 3А - проверяем, является ли баланс
               // у абонента неотрицательным
               if (phone.AccountBalance >= 0)
               {
                    // Блок 3Б - обновляем сумму выплаченных
                    // задолженностей
                    // и меняем статус абонента на "Подключен"
                    DateTime debtDate =
                         phone.StatusHistory.Last().DateTime;

                    decimal paymentsAmountSinceDebt =
                         CalculateTotalPaymentAmount(phone,
                         debtDate);

                    debtsPaid += Math.Abs(phone.AccountBalance -
                         paymentsAmountSinceDebt);

                    _statusService.SetConnectedStatus(phone);
               }
               else
               {
```

```
                    // Блок 3В - обновляем общую сумму
                    // задолженностей
                    debtsTotal += -phone.AccountBalance;
                }
            }
        }

        // Блок 4 - сохраняем изменения в БД
        _context.SaveChanges();

        // Блок 5 - возвращаем объект очета по операции
        return new DebtCalculationResult(debtsTotal
            + newDebt, debtsPaid, newDebt);
}
```

Для вычисления цикломатической сложности V(G) применяется следующая формула:

$$V(G) = e - n + 2p,$$

где e – количество дуг;

n – количество вершин;

p – число компонент связности.

Граф содержит 13 вершин, 18 дуг и одну компоненту связности. Таким образом, значение цикломатической сложности равно 7.

$$V(G) = 18 - 13 + 2 = 7.$$

Как и у многих метрик в отрасли, нет точного и универсального ограничения для цикломатической сложности. Однако NIST235 указывает на то, что ограничение в 10 является хорошей отправной точкой [18]. Следовательно, полученное значение цикломатической сложности свидетельствует о том, что уровень сложности приемлемый.

В результате выполнение данного этапа работы удалось показать, что написанный программный код обладает относительно высоким качеством.

Заключение

Результатом выполнения преддипломной практики является спроектированное web-приложение системы учета задолженностей для городской телефонной сети.

В ходе выполнения этапа анализа требований были рассмотрены приложения-аналоги и осуществлен их анализ: были выделены преимущества и недостатки, а также функциональные особенности и возможности каждого из них. Также было проведено сравнение аналогов между собой. Результатом данных сравнений стали функциональные требования к проектируемому web-приложению. Также были выявлены дополнительные функциональные требования, проведен их анализ.

В ходе выполнения этапа проектирования приложения определена архитектура приложения. Принято решение использовать клиент-серверное взаимодействие между web-браузером и серверной частью приложения. Для реализации web-приложения выбран шаблон MVC. Проведено проектирования алгоритмов, создана диаграмма деятельности для одного из алгоритмов приложения. Спроектированы структуры данных и созданы макеты интерфейса. Проектирование структур данных включило в себя построение диаграммы «сущность-связь», а также физической моделей данных, разработанной с помощью Entity Framework Core подходом «Code-First». В качестве СУБД был выбран MS SQL Server 2018.

Был составлен план и проведена оценка бюджета. В ходе выполнения данного этапа выявлены временные и финансовые затраты на разработку и необходимые ресурсы. Результатом стала диаграмма Ганта и диаграмма структуры расходов. Анализ показателей позволил выяснить, что разработка и проектирование программного продукта займет 288 часов, а суммарные расходы составят 292761,90 рублей.

Спроектированное приложение было реализовано на языке C# с помощью платформы для разработки web-приложения ASP.NET Core. Были составлены диаграммы компонентов и развертывания, а также диаграмма классов.

Было проведено модульное тестирование разработанного приложения, что позволило определить корректность функционирования программных компонентов. Кроме того, были вычислены метрики кода, показавшие высокие показатели его качества.

В результате выполнения работы все поставленные задачи были выполнены.

Список использованных источников

1. Полуэктова, Н. Р. Разработка веб-приложений: учебное пособие для вузов / Н. Р. Полуэктова. – М.: Издательство Юрайт, 2022. – 204 с.

2. Главная страница сайта Московской ГТС // МГТС [Электронный ресурс]. URL: https://mgts.ru/ (дата обращения: 29.05.2023).

3. Главная страница сайта ГТС г. Железногорск // ГТС г. Железногорск [Электронный ресурс]. URL: https://www.k26.ru/ (дата обращения: 29.05.2023).

4. Главная страница сайта Костромской ГТС // КГТС [Электронный ресурс]. URL: https://www.kostroma.net/ (дата обращения: 29.05.2023).

5. Оливер, В. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 5-е изд. / В. Оливер, Н. Оливер. – СПб.: Питер, 2016. – 992 с.

6. Главная страница сайта МУП «ГТС» // МУП «ГТС» [Электронный ресурс]. URL: https://mupgts.ru/ (дата обращения: 29.05.2023).

7. Главная страница сайта Псковской ГТС // ПГТС [Электронный ресурс]. URL: http://gtspskov.ru/ (дата обращения: 30.05.2023).

8. Лок, Э. ASP.Net Core в действии / Э. Лок. – М.: ДМК Пресс, 2021. – 906 с.

9. Макконнелл, С. Совершенный код, 2-е изд. / С. Макконнелл. – М.: Издательство «Русская редакция», 2018. – 896 с.

10. Буч, Г. Язык UML. Руководство пользователя. 2-е изд. / Г. Буч, Д. Рамбо, И. Якобсон. – М.: ДМК Пресс, 2006. – с. 252–255.

11. Гамма, Э. Приемы объектно-ориентированного программирования. Паттерны проектирования / Э. Гамма, Р. Хельм, Р. Джонсон, Дж. Влиссидес. – СПб: Питер, 2001 – 368 с.

12. Смит, Дж. П. Entity Framework Core в действии / Дж. П. Смит. – М.: ДМК Пресс, 2022. – 690 с.

13. Дейт, К. Дж. Введение в системы баз данных, 8-е издание / К. Дж. Дейт. – М.: Издательский дом «Вильяме», 2005. – 1328 с.

14. Введение в Identity в ASP.NET Core // Документация Microsoft [Электронный ресурс]. URL:

https://learn.microsoft.com/ru-ru/aspnet/core/security/authentication/identity?view=aspnetcore-6.0 (дата обращения: 29.05.2023).

15. Управление схемами баз данных // Документация Microsoft [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/ef/core/managing-schemas/ (дата обращения: 30.05.2023).

16. Seemann, M. Dependency Injection Principles, Practices, Patterns / M. Seemann, S. van Deursen. – N.Y.: Manning Publications Co., 2019. – 562 p.

17. Хориков, В. Принципы юнит-тестирования. / В. Хориков. – СПб.: Питер, 2021. – 320 с.

18. Watson, A. H. Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric / A. H. Watson, T. J. McCabe. // National Institute of Standards and Technology. – 1996. – 123 p.

Приложение А

ЛИСТИНГ КОДА ФАЙЛА PROGRAM.CS

(рекомендуемое)

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Localization;
using Microsoft.EntityFrameworkCore;
using PbxPaymentsMvc.Application.Extensions;
using PbxPaymentsMvc.Domain.Entities;
using PbxPaymentsMvc.Infrastructure.Data;
using System.Globalization;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllersWithViews();

builder.Services.AddServiceLayer();

builder.Services.AddDbContext<PbxContext>(options =>
{
    options.UseLazyLoadingProxies();

options.UseSqlServer(builder.Configuration.GetConnectionString("Pb
xDbConnection"));
});

builder.Services.Configure<RouteOptions>(options =>
{
    options.LowercaseUrls = true;
});

builder.Services.AddDefaultIdentity<User>(options =>
    {
        options.SignIn.RequireConfirmedAccount = true;
    })
    .AddRoles<IdentityRole<int>>()
    .AddEntityFrameworkStores<PbxContext>();

builder.Services.Configure<IdentityOptions>(options =>
{
    options.Password.RequireDigit = true;
    options.Password.RequireLowercase = true;
    options.Password.RequireNonAlphanumeric = true;
    options.Password.RequireUppercase = true;
    options.Password.RequiredLength = 6;
    options.Password.RequiredUniqueChars = 1;

    options.Lockout.DefaultLockoutTimeSpan =
TimeSpan.FromMinutes(5);
    options.Lockout.MaxFailedAccessAttempts = 5;
    options.Lockout.AllowedForNewUsers = true;

    options.User.AllowedUserNameCharacters =

"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-
._@+";
    options.User.RequireUniqueEmail = false;
```

```
});

builder.Services.ConfigureApplicationCookie(options =>
{
    options.AccessDeniedPath = new
PathString("/account/AccessDenied");
    options.LoginPath = new PathString("/account/login");
});

var app = builder.Build();

var defaultCulture = new CultureInfo("ru-RU");
var localizationOptions = new RequestLocalizationOptions
{
    DefaultRequestCulture = new RequestCulture(defaultCulture),
    SupportedCultures = new List<CultureInfo> { defaultCulture },
    SupportedUICultures = new List<CultureInfo> { defaultCulture }
};
app.UseRequestLocalization(localizationOptions);

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}"
);
app.MapRazorPages();

app.Run();
```

Приложение Б

ЛИСТИНГ КОДА ФАЙЛА IPAYMENTSERVICE.CS

(рекомендуемое)

```
using System;
using PbxPaymentsMvc.Application.Dtos;

namespace PbxPaymentsMvc.Application.Services.Abstractions
{
    public interface IPaymentService
    {
        Task<int> GetPaymentsCountAsync();

        Task<PaymentDto> FindPaymentByIdAsync(int paymentId);
        Task<IEnumerable<PaymentDto>> GetPaymentsPageAsync(int
pageIndex, int pageSize);
        Task<IEnumerable<PaymentDto>> GetAllPaymentsAsync();
        Task<IEnumerable<PaymentDto>>
GetSubscriberPaymentsAsync(int subscriberPhoneId);

        Task<int> AddPaymentAsync(PaymentDto payment);
        Task RemovePaymentByIdAsync(int paymentId);

        Task<bool> PaymentExistsAsync(int paymentId);
        Task UpdatePaymentAsync(PaymentDto payment);
    }
}
```

Приложение В

ЛИСТИНГ КОДА ФАЙЛА PAYMENTSERVICE.CS

(рекомендуемое)

```
using System;
using System.Data;
using Microsoft.EntityFrameworkCore;
using PbxPaymentsMvc.Domain.Entities;
using PbxPaymentsMvc.Infrastructure.Data;
using PbxPaymentsMvc.Application.Dtos;
using PbxPaymentsMvc.Application.Services.Abstractions;

namespace PbxPaymentsMvc.Application.Services
{
    public class PaymentService : IPaymentService
    {
        private readonly PbxContext _context;

        private readonly ISubscriberPhoneService _phoneService;

        public PaymentService(PbxContext context,
ISubscriberPhoneService phoneService)
        {
            _context = context;
            _phoneService = phoneService;
        }

        public async Task<int> GetPaymentsCountAsync()
        {
            return await _context.Payments.CountAsync();
        }

        public async Task<PaymentDto> FindPaymentByIdAsync(int
paymentId)
        {
            Payment payment = await _context.Payments
                .Include(p => p.Phone)
                .AsNoTracking()
                .FirstOrDefaultAsync(m => m.Id == paymentId);

            return PaymentDto.MapEntity(payment);
        }

        public async Task<IEnumerable<PaymentDto>>
GetPaymentsPageAsync(int pageIndex, int pageSize)
        {
            if (pageIndex < 0) throw new ArgumentException("Page
index cannot be less than zero.");
            if (pageSize < 0) throw new ArgumentException("Page
size cannot be less than zero.");

            return await _context.Payments
                .Include(p => p.Phone)
                .AsNoTracking()
                .OrderByDescending(p => p.DateTime)
                .Skip(pageIndex * pageSize)
                .Take(pageSize)
```

```
                .Select(p => PaymentDto.MapEntity(p))
                .ToListAsync();
        }

        public async Task<IEnumerable<PaymentDto>>
GetAllPaymentsAsync()
        {
            return await _context.Payments
                .Include(p => p.Phone)
                .AsNoTracking()
                .OrderByDescending(p => p.DateTime)
                .Select(p => PaymentDto.MapEntity(p))
                .ToListAsync();
        }

        public async Task<IEnumerable<PaymentDto>>
GetSubscriberPaymentsAsync(int subscriberPhoneId)
        {
            return await _context.Payments
                .Include(p => p.Phone)
                .AsNoTracking()
                .Where(p => p.PhoneId == subscriberPhoneId)
                .OrderByDescending(p => p.DateTime)
                .Select(p => PaymentDto.MapEntity(p))
                .ToListAsync();
        }

        public async Task<int> AddPaymentAsync(PaymentDto
paymentDto)
        {
            if (paymentDto == null) throw new
ArgumentNullException(nameof(paymentDto));
            if (paymentDto.Amount <= 0) throw new
ArgumentException("Amount cannot be less or equal to 0.");

            Payment payment = CreatePaymentFromDto(paymentDto);
            _context.Payments.Attach(payment);
            _context.Entry(payment).Reference(p =>
p.Phone).Load();

            payment.Phone.AccountBalance += payment.Amount;
            if (payment.Phone.AccountBalance >= 0 &&
payment.Phone.CurrentStatus == PhoneStatusEnum.Debtor)
            {
                await
_phoneService.SetSubscriberPhoneStatusAsync(payment.PhoneId,
PhoneStatusEnum.Connected);
            }

            await _context.SaveChangesAsync();
            return payment.Id;
        }
```

```
        public async Task RemovePaymentByIdAsync(int paymentId)
        {
            Payment payment = await
_context.Payments.FindAsync(paymentId);
            if (payment != null)
            {
                _context.Payments.Remove(payment);
            }

            await _context.SaveChangesAsync();
        }

        public async Task UpdatePaymentAsync(PaymentDto
paymentDto)
        {
            Payment payment = CreatePaymentFromDto(paymentDto);
            _context.Payments.Update(payment);
            await _context.SaveChangesAsync();
        }

        public async Task<bool> PaymentExistsAsync(int paymentId)
        {
            return await
_context.Payments.AsNoTracking().AnyAsync(e => e.Id == paymentId);
        }

        private static Payment CreatePaymentFromDto(PaymentDto
paymentDto)
        {
            System.Diagnostics.Debug.Assert(paymentDto.PhoneId >
0);

            return new Payment()
            {
                Id = paymentDto.Id,
                PhoneId = paymentDto.PhoneId,
                Amount = paymentDto.Amount,
                DateTime = paymentDto.DateTime
            };
        }
    }
}
```

Приложение Г

ЛИСТИНГ КОДА ФАЙЛА PAYMENTSCONTROLLER.CS

(рекомендуемое)

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using PbxPaymentsMvc.Api.Models;
using PbxPaymentsMvc.Api.Utilities;
using PbxPaymentsMvc.Application.Dtos;
using PbxPaymentsMvc.Application.Extensions;
using PbxPaymentsMvc.Application.Services.Abstractions;

namespace PbxPaymentsMvc.Api.Controllers
{
    [Route("Account/Payments")]
    public class PaymentsController : Controller
    {
        private readonly IPaymentService _paymentService;
        private readonly ISubscriberPhoneService _phoneService;
        private readonly IUserService _userService;

        public PaymentsController(
            IPaymentService paymentService,
            ISubscriberPhoneService subscriberPhoneService,
            IUserService userService)
        {
            _paymentService = paymentService;
            _phoneService = subscriberPhoneService;
            _userService = userService;
        }

        [HttpGet]
        [Authorize(Roles =
$"{RoleHelper.SubscriberRoleName},{RoleHelper.EmployeeRoleName}")]
        public async Task<IActionResult> Index()
        {
            IEnumerable<PaymentDto> payments =
Array.Empty<PaymentDto>();

            UserDto user = await
_userService.FindUserByIdAsync(User.GetUserId());
            if (user.PhoneId != null)
            {
                payments = await
_paymentService.GetSubscriberPaymentsAsync(user.PhoneId.Value);
            }

            bool isEmployee =
User.IsInRole(RoleHelper.EmployeeRoleName);
            var viewModel = new PaymentsIndexViewModel()
            {
                Entities = payments.Select(p =>
PaymentViewModel.FromDto(p)),
                IsUserEmployee = isEmployee
            };
```

```
            return View(viewModel);
        }

        [HttpGet("Details/{id}")]
        [Authorize(Roles =
$"{RoleHelper.SubscriberRoleName},{RoleHelper.EmployeeRoleName}")]
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            PaymentDto payment = await
_paymentService.FindPaymentByIdAsync(id.Value);
            UserDto user = await
_userService.FindUserByIdAsync(User.GetUserId());
            if (payment == null || payment.PhoneId !=
user.PhoneId)
            {
                return NotFound();
            }

            return (payment == null) ? NotFound() :
View(PaymentViewModel.FromDto(payment));
        }

        [HttpGet("Create")]
        [Authorize(Roles = RoleHelper.EmployeeRoleName)]
        public IActionResult Create()
        {
            return View();
        }

        // To protect from overposting attacks, enable the
specific properties you want to bind to.
        // For more details, see
http://go.microsoft.com/fwlink/?LinkId=317598
        [HttpPost("Create")]
        [ValidateAntiForgeryToken]
        [Authorize(Roles = RoleHelper.EmployeeRoleName)]
        public async Task<IActionResult>
Create([Bind("Id,PhoneNumber,Amount,DateTime")] PaymentViewModel
payment)
        {
            SubscriberPhoneDto phone = await
ValidatePhoneNumberAsync(payment.PhoneNumber);
            if (ModelState.IsValid)
            {
                var paymentDto = new PaymentDto()
                {
                    Id = payment.Id,
```

```
                    PhoneId = phone.Id,
                    PhoneNumber = payment.PhoneNumber,
                    Amount = payment.Amount,
                    DateTime = payment.DateTime
                };

                await _paymentService.AddPaymentAsync(paymentDto);
                return RedirectToAction(nameof(Index));
            }

            return View(payment);
        }

        [Authorize]
        [HttpGet("Edit/{id}")]
        [Authorize(Roles = RoleHelper.EmployeeRoleName)]
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            PaymentDto payment = await
_paymentService.FindPaymentByIdAsync(id.Value);
            if (payment == null)
            {
                return NotFound();
            }

            var viewModel = PaymentViewModel.FromDto(payment);
            return View(viewModel);
        }

        // To protect from overposting attacks, enable the
specific properties you want to bind to.
        // For more details, see
http://go.microsoft.com/fwlink/?LinkId=317598
        [Authorize]
        [HttpPost("Edit/{id}")]
        [ValidateAntiForgeryToken]
        [Authorize(Roles = RoleHelper.EmployeeRoleName)]
        public async Task<IActionResult> Edit(int id,
[Bind("Id,PhoneNumber,Amount,DateTime")] PaymentViewModel payment)
        {
            if (id != payment.Id)
            {
                return NotFound();
            }

            SubscriberPhoneDto phone = await
ValidatePhoneNumberAsync(payment.PhoneNumber);
            if (ModelState.IsValid)
```

```
            {
                var paymentDto = new PaymentDto()
                {
                    Id = payment.Id,
                    PhoneId = phone.Id,
                    PhoneNumber = payment.PhoneNumber,
                    Amount = payment.Amount,
                    DateTime = payment.DateTime
                };

                try
                {
                    await
_paymentService.UpdatePaymentAsync(paymentDto);
                }
                catch (DbUpdateConcurrencyException)
                {
                    bool paymentExists = await
_paymentService.PaymentExistsAsync(payment.Id);
                    if (!paymentExists)
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }

            return View(payment);
        }

        [Authorize]
        [HttpGet("Delete/{id}")]
        [Authorize(Roles = RoleHelper.EmployeeRoleName)]
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            PaymentDto payment = await
_paymentService.FindPaymentByIdAsync(id.Value);
            bool phoneExists = await
_phoneService.SubscriberPhoneExistsAsync(payment.PhoneId);
            if (payment == null || !phoneExists)
            {
                return NotFound();
            }
```

```
            var viewModel = PaymentViewModel.FromDto(payment);
            return View(viewModel);
        }

        [Authorize]
        [HttpPost("Delete/{id}")]
        [ValidateAntiForgeryToken]
        [Authorize(Roles = RoleHelper.EmployeeRoleName)]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            await _paymentService.RemovePaymentByIdAsync(id);
            return RedirectToAction(nameof(Index));
        }

        private async Task<SubscriberPhoneDto>
ValidatePhoneNumberAsync(string phoneNumber)
        {
            SubscriberPhoneDto phone = await
_phoneService.FindSubscriberPhoneByNumberAsync(phoneNumber);
            if (phone == null)
            {

ModelState.AddModelError(nameof(PaymentViewModel.PhoneNumber),
"Телефона с таким номером не существует");
            }

            return phone;
        }
    }
}
```

Приложение Д

ЛИСТИНГ КОДА ФАЙЛА PAYMENTS/INDEX.CSHTML

(рекомендуемое)

```
@model PaymentsIndexViewModel

@{
    ViewData["Title"] = "Список платежей";
    ViewData["ActivePage"] = AccountNavigationPages.Payments;
    Layout = "/Pages/Account/Manage/_Layout.cshtml";
}

<h3>@ViewData["Title"]</h3>

<div class="scrollable-table">
    <table class="table scrollable-table-inner">
        <thead>
            <tr>
                <th>Номер телефона</th>
                <th>Сумма платежа</th>
                <th>Дата и время</th>
                <th></th>
            </tr>
        </thead>
        <tbody>
        @foreach (var item in Model.Entities)
        {
            <tr>
                <td>@Html.DisplayFor(modelItem =>
item.PhoneNumber)</td>
                <td>@Html.DisplayFor(modelItem =>
item.Amount)</td>
                <td>@Html.DisplayFor(modelItem =>
item.DateTime)</td>
                <td>
                    <a asp-action="Details" asp-route-
id="@item.Id">Просмотреть</a>
                    @if (Model.IsUserEmployee)
                    {
                        <span>|</span>
                        <a asp-action="Edit" asp-route-
id="@item.Id">Изменить</a>
                    }
                </td>
            </tr>
        }
        </tbody>
    </table>
</div>
```

Приложение Е

ЛИСТИНГ КОДА ФАЙЛА PAYMENTSEMPLOYEECONTROLLER.CS

(рекомендуемое)

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using PbxPaymentsMvc.Api.Models;
using PbxPaymentsMvc.Api.Utilities;
using PbxPaymentsMvc.Application.Dtos;
using PbxPaymentsMvc.Application.Services.Abstractions;

namespace PbxPaymentsMvc.Api.Controllers
{
    [Route("Payments")]
    [Authorize(Roles = RoleHelper.EmployeeRoleName)]
    public class PaymentsEmployeeController : Controller
    {
        private const int PaymentsOnPage = 10;

        private readonly IPaymentService _paymentService;
        private readonly ISubscriberPhoneService _phoneService;

        public PaymentsEmployeeController(
            IPaymentService paymentService,
            ISubscriberPhoneService phoneService)
        {
            _paymentService = paymentService;
            _phoneService = phoneService;
        }

        [HttpGet]
        public async Task<IActionResult> Index()
        {
            int firstPageNumber = 1;
            return await Index(firstPageNumber);
        }

        [HttpGet("{pageNumber}")]
        public async Task<IActionResult> Index(int? pageNumber)
        {
            int paymentsCount = await
               _paymentService.GetPaymentsCountAsync();
            int pagesCount =
               (int)Math.Ceiling((double)paymentsCount /
               PaymentsOnPage);

            if (pageNumber == null)
            {
                return NotFound();
            }

            if (pageNumber <= 0) pageNumber = 1;
            if (pageNumber > pagesCount) pageNumber = pagesCount;

            int pageIndex = pageNumber.Value - 1;
```

```
            IEnumerable<PaymentDto> payments = await
_paymentService.GetPaymentsPageAsync(pageIndex, PaymentsOnPage);

            var viewModel = new PaymentsIndexViewModel()
            {
                Entities = payments.Select(p =>
                    PaymentViewModel.FromDto(p)),
                PageNumber = pageNumber.Value,
                PagesCount = pagesCount,
                IsUserEmployee = true
            };

            return View(viewModel);
        }

        [HttpGet("Details/{id}")]
        public async Task<IActionResult> Details(int? id,
          [FromQuery] int? pageNumber)
        {
            if (id == null)
            {
                return NotFound();
            }

            PaymentDto payment = await
               _paymentService.FindPaymentByIdAsync(id.Value);
            if (payment == null)
            {
                return NotFound();
            }

            var viewModel = PaymentViewModel.FromDto(payment);
            viewModel.PageNumber = pageNumber ?? 1;

            return View(viewModel);
        }

        [HttpGet("Create")]
        public  IActionResult Create([FromQuery] int? pageNumber)
        {
            var dateTime = DateTime.Now;
            dateTime = new DateTime(
                dateTime.Year, dateTime.Month, dateTime.Day,
                dateTime.Hour, dateTime.Minute, 0,
                dateTime.Kind
            );

            var viewModel = new PaymentViewModel()
               { DateTime = dateTime };
            viewModel.PageNumber = pageNumber ?? 1;

            return View(viewModel);
        }
```

```
// To protect from overposting attacks, enable the
  specific properties you want to bind to.
// For more details, see
  http://go.microsoft.com/fwlink/?LinkId=317598
[HttpPost("Create")]
[ValidateAntiForgeryToken]
public async Task<IActionResult>
  Create([Bind("Id,PhoneNumber,Amount,DateTime")]
  PaymentViewModel payment)
{
    if (payment == null)
    {
        return NotFound();
    }

    SubscriberPhoneDto phone = await
       ValidatePhoneNumberAsync(payment.PhoneNumber);
    if (ModelState.IsValid)
    {
        var paymentDto = new PaymentDto()
        {
            Id = payment.Id,
            PhoneId = phone.Id,
            PhoneNumber = payment.PhoneNumber,
            Amount = payment.Amount,
            DateTime = payment.DateTime
        };

        await _paymentService.AddPaymentAsync(paymentDto);
        return RedirectToAction(nameof(Index));
    }

    return View(payment);
}

[Authorize]
[HttpGet("Edit/{id}")]
public async Task<IActionResult> Edit(int? id, [FromQuery]
  int? pageNumber)
{
    if (id == null)
    {
        return NotFound();
    }

    PaymentDto payment = await
       _paymentService.FindPaymentByIdAsync(id.Value);
    if (payment == null)
    {
        return NotFound();
    }
```

```
    var viewModel = PaymentViewModel.FromDto(payment);
    viewModel.PageNumber = pageNumber ?? 1;

    return View(viewModel);
}

// To protect from overposting attacks, enable the
  specific properties you want to bind to.
// For more details, see
  http://go.microsoft.com/fwlink/?LinkId=317598
[Authorize]
[HttpPost("Edit/{id}")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(
    int id,
    [Bind("Id,PhoneNumber,Amount,DateTime")]
       PaymentViewModel payment,
    [FromQuery] int? pageNumber)
{
    if (payment == null || id != payment.Id)
    {
        return NotFound();
    }

    SubscriberPhoneDto phone = await
       ValidatePhoneNumberAsync(payment.PhoneNumber);
    if (ModelState.IsValid)
    {
        var paymentDto = new PaymentDto()
        {
            Id = payment.Id,
            PhoneId = phone.Id,
            PhoneNumber = payment.PhoneNumber,
            Amount = payment.Amount,
            DateTime = payment.DateTime
        };

        try
        {
            await
                 _paymentService.UpdatePaymentAsync(
                 paymentDto);
        }
        catch (DbUpdateConcurrencyException)
        {
            bool paymentExists = await
                 _paymentService.PaymentExistsAsync(
                 payment.Id);
            if (!paymentExists)
            {
                return NotFound();
            }
            else
```

```
            {
                throw;
            }
        }

        return RedirectToAction(nameof(Index),
            new { pageNumber });
    }

    payment.PageNumber = pageNumber ?? 1;

    return View(payment);
}

[HttpGet("Delete/{id}")]
public async Task<IActionResult> Delete(int? id,
  [FromQuery] int? pageNumber)
{
    if (id == null)
    {
        return NotFound();
    }

    PaymentDto payment = await
       _paymentService.FindPaymentByIdAsync(id.Value);
    if (payment == null)
    {
        return NotFound();
    }

    var viewModel = PaymentViewModel.FromDto(payment);
    viewModel.PageNumber = pageNumber ?? 1;

    return View(viewModel);
}

[HttpPost("Delete/{id}")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id,
   [FromQuery] int? pageNumber)
{
    await _paymentService.RemovePaymentByIdAsync(id);
    return RedirectToAction(nameof(Index), pageNumber);
}

private async Task<SubscriberPhoneDto>
  ValidatePhoneNumberAsync(string phoneNumber)
{
    SubscriberPhoneDto phone = await
       _phoneService.FindSubscriberPhoneByNumberAsync(
       phoneNumber);
    if (phone == null)
    {
```

```
                ModelState.AddModelError(
                     nameof(PaymentViewModel.PhoneNumber),
                     "Телефона с таким номером не существует"
                );
            }

            return phone;
        }
    }
}
```

Приложение Ж

ЛИСТИНГ КОДА ФАЙЛА PAYMENTSEMPLOYEE/INDEX.CSHTML

(рекомендуемое)

```
@model PaymentsIndexViewModel
@{
    ViewData["Title"] = "Список платежей";
}

<h3>@ViewData["Title"]</h3>
<p><a asp-action="Create" asp-route-
pageNumber="@Model.PageNumber">Добавить платеж</a></p>

<div class="scrollable-table">
    <table class="table scrollable-table-inner">
        <thead>
            <tr>
                <th>Номер телефона</th>
                <th>Сумма платежа</th>
                <th>Дата и время</th>
                <th></th>
            </tr>
        </thead>
        <tbody>
        @foreach (var item in Model.Entities)
        {
            <tr>
                <td>@Html.DisplayFor(modelItem =>
item.PhoneNumber)</td>
                <td>@Html.DisplayFor(modelItem =>
item.Amount)</td>
                <td>@Html.DisplayFor(modelItem =>
item.DateTime)</td>
                <td>
                    <a asp-action="Details" asp-route-
id="@item.Id" asp-route-
pageNumber="@Model.PageNumber">Просмотреть</a>
                    <span>|</span> <a asp-action="Edit" asp-route-
id="@item.Id" asp-route-
pageNumber="@Model.PageNumber">Изменить</a>
                    <span>|</span> <a asp-action="Delete" asp-
route-id="@item.Id" asp-route-
pageNumber="@Model.PageNumber">Удалить</a>
                </td>
            </tr>
        }
        </tbody>
    </table>
</div>

@if (Model.PagesCount > 1)
{
    <partial name="_PaginationNav" for=@Model/>
}
```

Приложение И

ЛИСТИНГ КОДА ФАЙЛА DEBTORSCONTROLLER.CS

(рекомендуемое)

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using PbxPaymentsMvc.Api.Models;
using PbxPaymentsMvc.Api.Utilities;
using PbxPaymentsMvc.Application.Dtos;
using PbxPaymentsMvc.Application.Services;
using PbxPaymentsMvc.Application.Services.Abstractions;

namespace PbxPaymentsMvc.Api.Controllers
{
    public class DebtorsController : Controller
    {
        private readonly IFeeService _feeService;
        private readonly ISubscriberPhoneService
_subscriberPhoneService;

        public DebtorsController(
            IFeeService feeService,
            ISubscriberPhoneService subscriberPhoneService)
        {
            _feeService = feeService;
            _subscriberPhoneService = subscriberPhoneService;
        }

        // GET: DebtorsController
        [Authorize(Roles = RoleHelper.EmployeeRoleName)]
        public async Task<IActionResult> Index()
        {
            DebtorsIndexViewModel viewModel = await
CreateIndexViewModel();
            return View(viewModel);
        }

        // GET: DebtorsController/Details/5
        [Authorize(Roles = RoleHelper.EmployeeRoleName)]
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            SubscriberPhoneDto phone = await
_subscriberPhoneService.FindSubscriberPhoneByIdAsync(id.Value);
            if (phone == null || phone.Owner == null)
            {
                return NotFound();
            }

            bool hasDebt = (phone.CurrentStatus ==
Domain.Entities.PhoneStatusEnum.Debtor || phone.AccountBalance <
0);
```

```
            return (hasDebt == false) ? NotFound() :
View(DebtorViewModel.FromDto(phone));
        }

        [HttpPost]
        [Authorize(Roles = RoleHelper.EmployeeRoleName)]
        public async Task<IActionResult> Update()
        {
            DebtCalculationResult debts = await
_feeService.UpdateDebtsAsync();
            string reportText =
                $"Выплаченные задолженности: {debts.DebtsPaid}
₽\n" +
                $"Образовавшиеся задолженности: {debts.NewDebts}
₽\n" +
                $"Новая общая сумма задолженностей:
{debts.DebtsTotal} ₽";

            var modalViewModel = new ModalViewModel("reportModal",
"Отчет о выполнении операции", reportText);
            DebtorsIndexViewModel viewModel = await
CreateIndexViewModel(modalViewModel);
            return View("Index", viewModel);
        }

        [HttpPost]
        [Authorize(Roles = RoleHelper.EmployeeRoleName)]
        public async Task<IActionResult> Pay()
        {
            FeeCalculationResult fees = await
_feeService.MakeTariffPaymentsAsync();
            string reportText =
                $"Всего оплат по тарифам: {fees.FeeCount}\n" +
                $"Общая сумма оплат: {fees.FeesTotal} ₽";

            var modalViewModel = new ModalViewModel("reportModal",
"Отчет о выполнении операции", reportText);
            DebtorsIndexViewModel viewModel = await
CreateIndexViewModel(modalViewModel);
            return View("Index", viewModel);
        }

        private async Task<DebtorsIndexViewModel>
CreateIndexViewModel(ModalViewModel modalViewModel = null)
        {
            IEnumerable<SubscriberPhoneDto> debtorPhones = await
_feeService.GetDebtorPhonesAsync();
            IEnumerable<DebtorViewModel> debtors =
debtorPhones.Select(phone => DebtorViewModel.FromDto(phone));
            decimal totalDebts = debtors.Where(d =>
d.AccountBalance < 0).Sum(d => -d.AccountBalance);

            var viewModel = new DebtorsIndexViewModel()
```

```
            {
                Debtors = debtors,
                Modal = modalViewModel,
                TotalDebts = totalDebts
            };

            return viewModel;
        }
    }
}
```

Приложение К

ЛИСТИНГ КОДА ФАЙЛА DEBTORS/INDEX.CSHTML

(рекомендуемое)

```
@model DebtorsIndexViewModel

@{
    ViewData["Title"] = "Задолженности и расчет по тарифам";

    var updateDebtsModal = new
        PostModalViewModel("updateDebtsModal", "Обновление
        задолженностей",
        "Вы уверены, что хотите обновить задолженности?",
        "Debtors", "Update");

    var payFeesModal = new PostModalViewModel("payFeesModal",
        "Расчет по тарифам",
        "Вы уверены, что хотите провести расчет по тарифам?",
        "Debtors", "Pay");
}

<h1>@ViewData["Title"]</h1>

<table class="table" style="overflow: scroll;">
    <thead>
        <tr>
            <th>Номер</th>
            <th>Адрес</th>
            <th>ФИО</th>
            <th>Баланс счета</th>
            <th>Текущий статус</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Debtors)
        {
            <tr>
                <td>@Html.DisplayFor(modelItem =>
                    item.Number)</td>
                <td>@Html.DisplayFor(modelItem =>
                    item.Address)</td>
                <td>@Html.DisplayFor(modelItem =>
                    item.OwnerFullName)</td>
                <td>@Html.DisplayFor(modelItem =>
                    item.AccountBalance) ₽</td>
                <td>@Html.DisplayFor(modelItem =>
                    item.CurrentStatus)</td>
                <td>
                    <a asp-action="Details"
                        asp-route-id="@item.Id">Просмотр</a>
                </td>
            </tr>
        }
        <tr>
            <td><b>Всего</b></td>
            <td></td>
```

```
            <td></td>
            <td><b>@Model.TotalDebts ₽</b></td>
            <td></td>
            <td></td>
        </tr>
    </tbody>
</table>

<div class="mt-4">
    <button type="button" class="btn btn-outline-primary
        btn-sm me-2" data-bs-toggle="modal"
        data-bs-target="#@updateDebtsModal.Id">
        Обновить задолженности
    </button>

    <button type="button" class="btn btn-outline-primary btn-sm"
data-bs-toggle="modal" data-bs-target="#@payFeesModal.Id">
        Произвести расчет по тарифам
    </button>
</div>

<partial name="_ConfirmationModal" model="updateDebtsModal"/>
<partial name="_ConfirmationModal" model="payFeesModal"/>
<partial name="_ReportModal" for="Modal" />

@section Scripts
{
    <script src="@Url.Content("~/js/report-modal.js")"></script>
}
```

Приложение Л

ЛИСТИНГ КОДА ФАЙЛА ACCOUNT/MANAGE/INDEX.CSHTML

(рекомендуемое)

```
@page
@model IndexModel

@{
    ViewData["Title"] = "Информация профиля";
    ViewData["ActivePage"] = AccountNavigationPages.Index;
    Layout = "/Pages/Account/Manage/_Layout.cshtml";
}

<h3>@ViewData["Title"]</h3>

<partial name="_StatusMessage" for="NotConnectedViewModel" />
<partial name="_StatusMessage" for="StatusMessageViewModel" />

<div class="row">
    <div class="col-md-6">
        <form id="profile-form" method="post">
            <div asp-validation-summary="ModelOnly"
               class="text-danger"></div>
            <div class="form-floating mt-2 mb-2">
                <input asp-for="Username" class="form-control"
                    disabled/>
                <label asp-for="Username" class="form-label">
                    Имя пользователя
                </label>
            </div>
            <div class="form-floating mt-2 mb-2">
                <input asp-for="Input.LastName"
                    class="form-control" />
                <label asp-for="Input.LastName"
                    class="form-label"></label>
                <span asp-validation-for="Input.LastName"
                    class="text-danger"></span>
            </div>
            <div class="form-floating mt-2 mb-2">
                <input asp-for="Input.FirstName"
                    class="form-control" />
                <label asp-for="Input.FirstName"
                    class="form-label"></label>
                <span asp-validation-for="Input.FirstName"
                    class="text-danger"></span>
            </div>
            <div class="form-floating mt-2 mb-2">
                <input asp-for="Input.Patronymic"
                    class="form-control" />
                <label asp-for="Input.Patronymic"
                    class="form-label"></label>
                <span asp-validation-for="Input.Patronymic"
                    class="text-danger">
                </span>
            </div>
            <button id="update-profile-button" type="submit"
               class="w-100 btn btn-primary">Сохранить</button>
```

```
        </form>
    </div>
</div>
@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

Приложение М

ЛИСТИНГ КОДА ФАЙЛА ACCOUNT/MANAGE/INDEX.CSHTML.CS

(рекомендуемое)

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using PbxPaymentsMvc.Api.Models;
using PbxPaymentsMvc.Api.Utilities;
using PbxPaymentsMvc.Application.Services.Abstractions;
using PbxPaymentsMvc.Domain.Entities;
using System.ComponentModel.DataAnnotations;

namespace PbxPaymentsMvc.Api.Areas.Identity.Pages.Account.Manage
{
    public class IndexModel : PageModel
    {
        private readonly UserManager<User> _userManager;
        private readonly SignInManager<User> _signInManager;

        public IndexModel(
            UserManager<User> userManager,
            SignInManager<User> signInManager)
        {
            _userManager = userManager;
            _signInManager = signInManager;
        }

        public string Username { get; set; }

        public string PhoneNumber { get; set; }

        public decimal AccountBalance { get; set; }

        [TempData]
        public string StatusMessage { get; set; }

        [TempData]
        public string StatusMessageType { get; set; }

        [TempData]
        public string NotConnectedMessage { get; set; }

        public StatusMessageViewModel StatusMessageViewModel =>
new(StatusMessage, StatusMessageType);
        public StatusMessageViewModel NotConnectedViewModel =>
new(NotConnectedMessage, "warning");

        [BindProperty]
        public InputModel Input { get; set; }

        public bool IsConnected { get; set; }

        public class InputModel
        {
            [Required(ErrorMessage = "Имя не введено")]
```

```
            [StringLength(32, ErrorMessage = "Имя должно содержать
от {2} до {1} символов.", MinimumLength = 3)]
            [Display(Name = "Имя")]
            public string FirstName { get; set; }

            [Required(ErrorMessage = "Фамилия не введена")]
            [StringLength(32, ErrorMessage = "Фамилия должна
содержать от {2} до {1} символов.", MinimumLength = 3)]
            [Display(Name = "Фамилия")]
            public string LastName { get; set; }

            [StringLength(32, ErrorMessage = "Отчество не может
быть длинее {1} символов.")]
            [Display(Name = "Отчество")]
            public string Patronymic { get; set; }
        }

        private async Task LoadAsync(User user)
        {
            Username = await _userManager.GetUserNameAsync(user);
            Input = new InputModel
            {
                LastName = user.LastName,
                FirstName = user.FirstName,
                Patronymic = user.Patronymic
            };
        }

        public async Task<IActionResult> OnGetAsync()
        {
            if (!HttpContext.User.Identity.IsAuthenticated)
            {
                return RedirectToPage("/Account/Login");
            }

            User user = await _userManager.GetUserAsync(User);
            if (user == null)
            {
                return NotFound($"Не удалось получить пользователя
с ID '{_userManager.GetUserId(User)}'.");
            }

            IsConnected = (user.PhoneId != null);

            if (!IsConnected &&
!User.IsInRole(RoleHelper.EmployeeRoleName))
            {
                NotConnectedMessage = "Чтобы пользоваться услугами
телефонной сети, отправьте заявку на подключение.";
            }

            if (IsConnected && (user.Phone.AccountBalance < 0 ||
user.Phone.CurrentStatus == PhoneStatusEnum.Debtor))
```

```
            {
                StatusMessage = "Уважаемый абонент! Обращаем
внимание, что у вас образовалась задолженность!\n" +
                    "В случае непогашения задолженности вовремя
предоставление услуг будет ограничено.";
                StatusMessageType = "danger";
            }

            await LoadAsync(user);
            return Page();
        }

        public async Task<IActionResult> OnPostAsync()
        {
            if (!HttpContext.User.Identity.IsAuthenticated)
            {
                return RedirectToPage("/Account/Login");
            }

            User user = await _userManager.GetUserAsync(User);
            if (user == null)
            {
                return NotFound($"Не удалось получить пользователя
с ID '{_userManager.GetUserId(User)}'.");
            }

            if (!ModelState.IsValid)
            {
                await LoadAsync(user);
                return Page();
            }

            bool modified = false;

            string newLastName = Input.LastName?.Trim() ?? "";
            if (newLastName != user.LastName)
            {
                user.LastName = newLastName;
                modified = true;
            }

            string newFirstName = Input.FirstName?.Trim() ?? "";
            if (newFirstName != user.FirstName)
            {
                user.FirstName = newFirstName;
                modified = true;
            }

            string newPatronymic = Input.Patronymic?.Trim() ?? "";
            if (newPatronymic != user.Patronymic)
            {
                user.Patronymic = newPatronymic;
                modified = true;
```

```
            }

            if (modified)
            {
                IdentityResult editProfileResult = await
_userManager.UpdateAsync(user);
                if (!editProfileResult.Succeeded)
                {
                    StatusMessage = "Возникла ошибка при изменении
данных профиля.";
                    StatusMessageType = "danger";
                    return RedirectToPage();
                }
                else
                {
                    StatusMessage = "Данные профиля были
изменены.";
                    StatusMessageType = "success";
                }
            }
            else
            {
                StatusMessage = "Данные профиля не изменены.";
                StatusMessageType = "secondary";
            }

            await _signInManager.RefreshSignInAsync(user);
            return RedirectToPage();
        }
    }
}
```

Приложение Н

ЛИСТИНГ КОДА ФАЙЛА _LAYOUT.CSHTML

(рекомендуемое)

```
@{
    bool isEmployee = User.IsInRole(RoleHelper.EmployeeRoleName);
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>@ViewData["Title"] - PbxPaymentsMvc</title>
    <link rel="stylesheet"
href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-
version="true" />
    <link rel="stylesheet" href="~/PbxPaymentsMvc.styles.css" asp-
append-version="true" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm
navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container-fluid">
                <a class="navbar-brand" asp-area="" asp-
controller="Home" asp-action="Index">Сайт ГТС</a>
                <button class="navbar-toggler" type="button" data-
bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-
controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle
navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-
flex justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-
area="" asp-controller="Home" asp-action="Index">О сайте</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-
area="" asp-controller="Services" asp-action="Index">Услуги</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-
area="" asp-controller="Tariffs" asp-action="Index">Тарифы</a>
                        </li>
                        @if (isEmployee)
                        {
                            <li class="nav-item">
                                <a class="nav-link text-dark" asp-
area="" asp-controller="PaymentsEmployee" asp-
action="Index">Платежи</a>
```

```
                            </li>
                            <li class="nav-item">
                                <a class="nav-link text-dark" asp-
area="" asp-controller="Reports" asp-action="Index">Отчеты</a>
                            </li>
                            <li class="nav-item">
                                <a class="nav-link text-dark" asp-
area="" asp-controller="RequestsEmployee" asp-
action="Index">Заявки</a>
                            </li>
                            <li class="nav-item">
                                <a class="nav-link text-dark" asp-
area="" asp-controller="Debtors" asp-action="Index">Задолжники</a>
                            </li>
                        }
                    </ul>
                    <partial name="_LoginPartial" />
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <!--
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2023 - PbxPaymentsMvc - <a asp-area="" asp-
controller="Home" asp-action="Privacy">Privacy</a>
        </div>
    </footer>
    -->

    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script
src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

Приложение П

ЛИСТИНГ КОДА ФАЙЛА PBXCONTEXT.CS

(рекомендуемое)

```
using System;
using PbxPaymentsMvc.Domain.Entities;
using PbxPaymentsMvc.Infrastructure.Data.Config;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

namespace PbxPaymentsMvc.Infrastructure.Data
{
    public class PbxContext :
          IdentityDbContext<User, IdentityRole<int>, int>
    {
        public PbxContext(DbContextOptions<PbxContext> options) :
              base(options)
        {
        }

        // For unit testing
        protected PbxContext() : base()
        {
        }

        public virtual DbSet<PhoneType> PhoneTypes
          { get; set; }
        public virtual DbSet<SubscriberPhone> SubscriberPhones
          { get; set; }
        public virtual DbSet<PhoneStatus> PhoneStatus
          { get; set; }
        public virtual DbSet<StatusHistoryEntry>
          StatusHistoryEntries { get; set; }

        public virtual DbSet<Tariff> Tariffs { get; set; }

        public virtual DbSet<TariffHistoryEntry>
          TariffHistoryEntries { get; set; }

        public virtual DbSet<Payment> Payments { get; set; }

        public virtual DbSet<Call> Calls { get; set; }

        public virtual DbSet<Request> Requests { get; set; }

        public virtual DbSet<ConnectRequest>
          ConnectRequests { get; set; }

        public virtual DbSet<ChangeTariffRequest>
          ChangeTariffRequests { get; set; }

        public virtual DbSet<DisconnectRequest>
          DisconnectRequests { get; set; }

        protected override void OnModelCreating(ModelBuilder
          modelBuilder)
```

```
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.ApplyConfigurationsFromAssembly(
               typeof(CallEntityTypeConfiguration).Assembly);
        }
    }
}
```

Приложение Р

WEB-СТРАНИЦЫ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

(рекомендуемое)

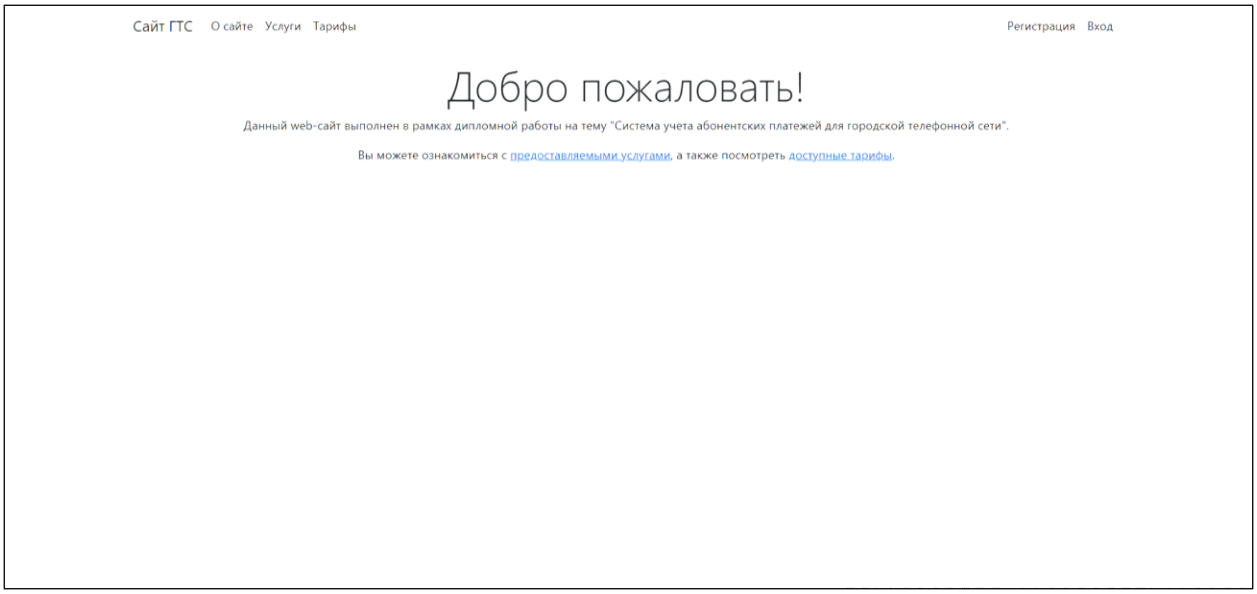

Рисунок Р.1 – Главная страница

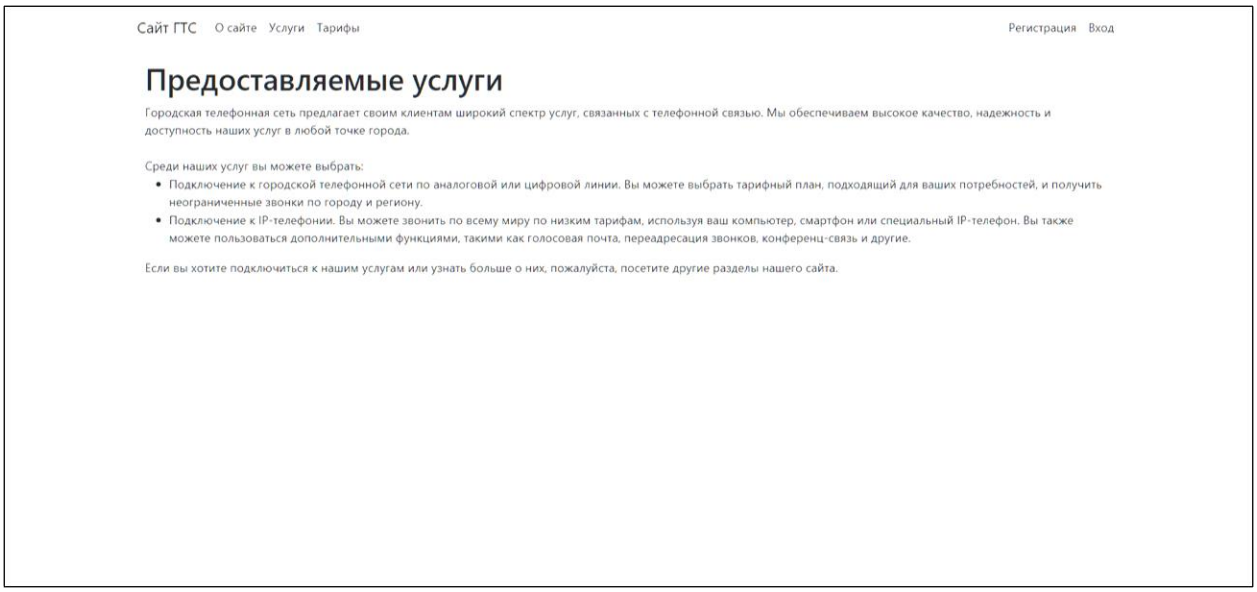

Рисунок Р.2 – Страница со списком предоставляемых услуг

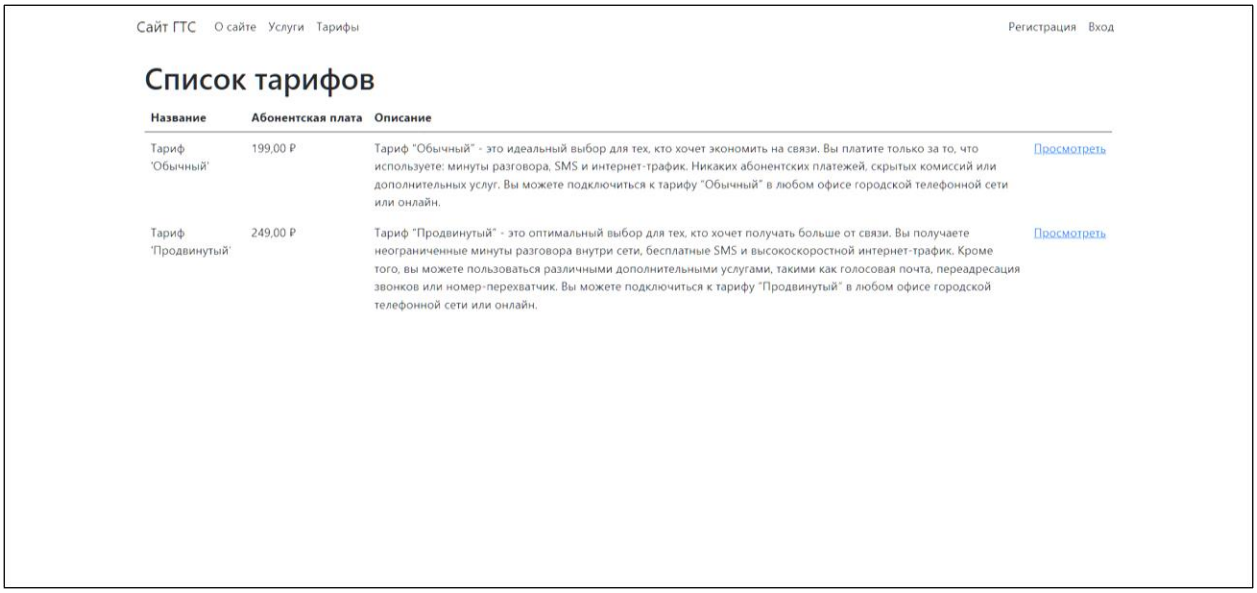

Рисунок Р.3 – Страница со списком тарифов

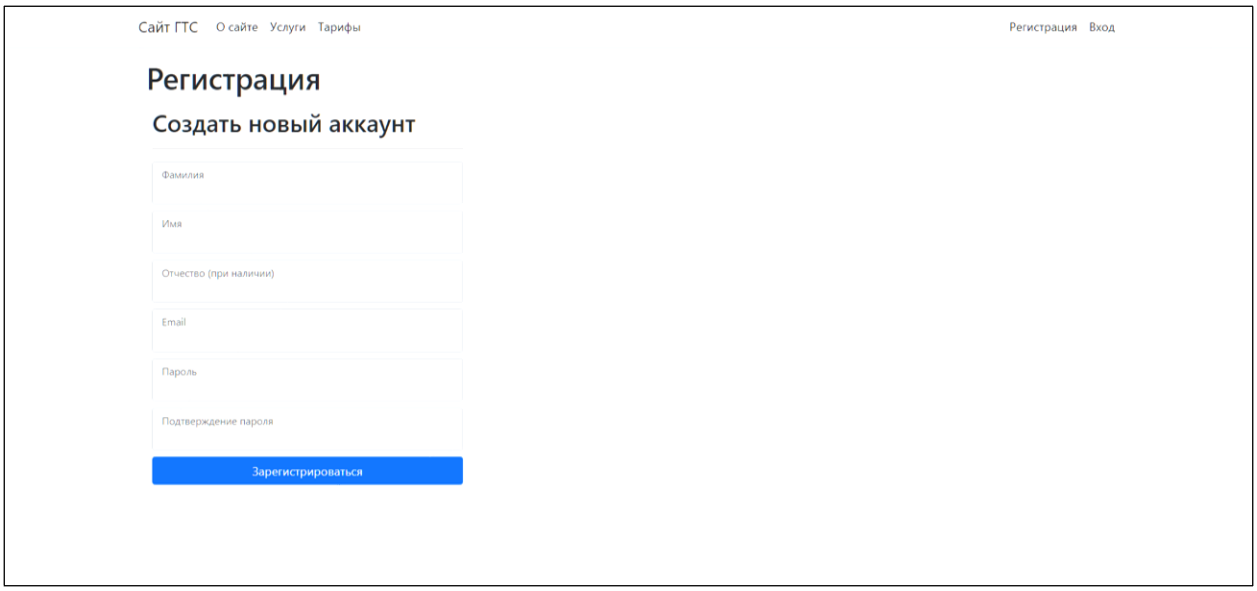

Рисунок Р.4 – Страница регистрации

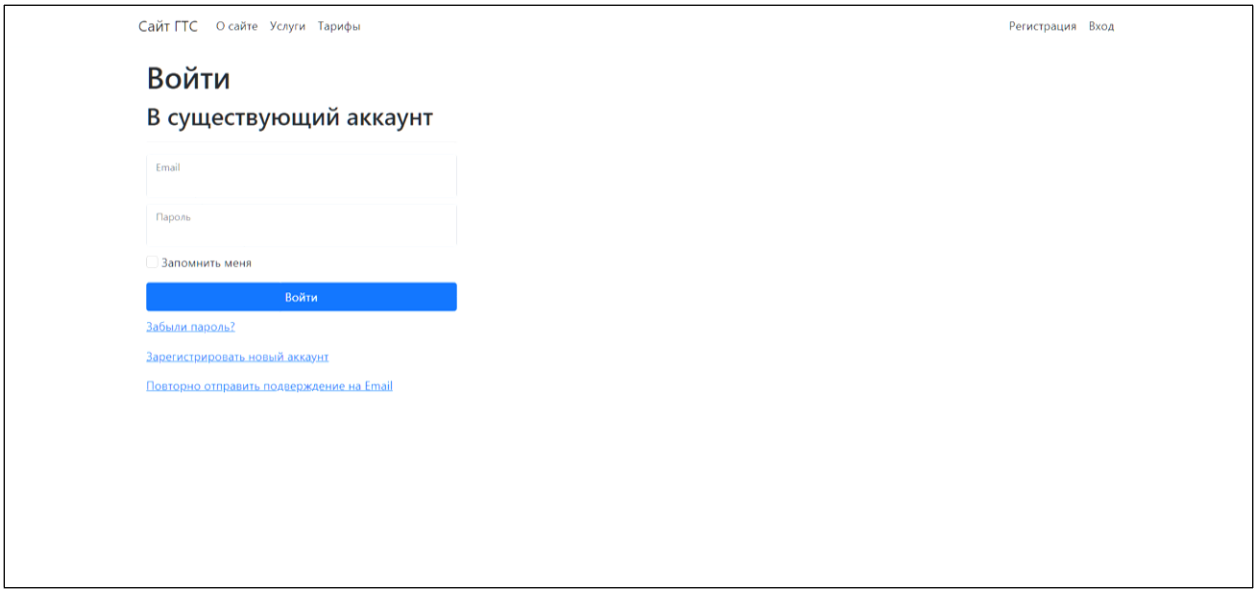

Рисунок Р.5 – Страница аутентификации

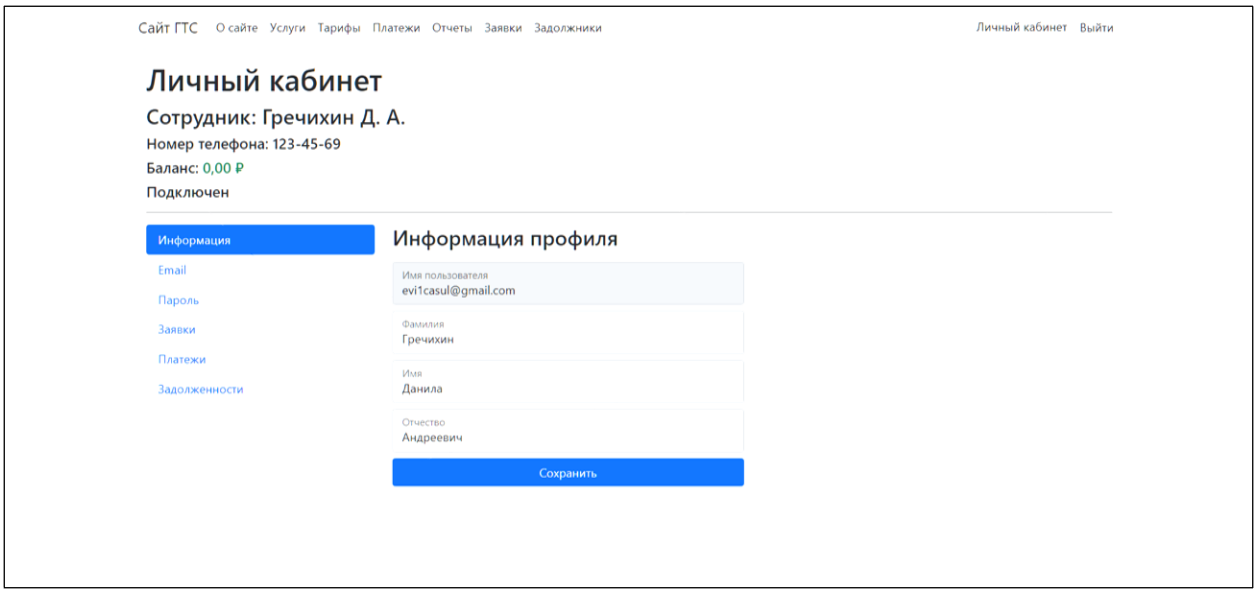

Рисунок Р.6 – Страница личного кабинета

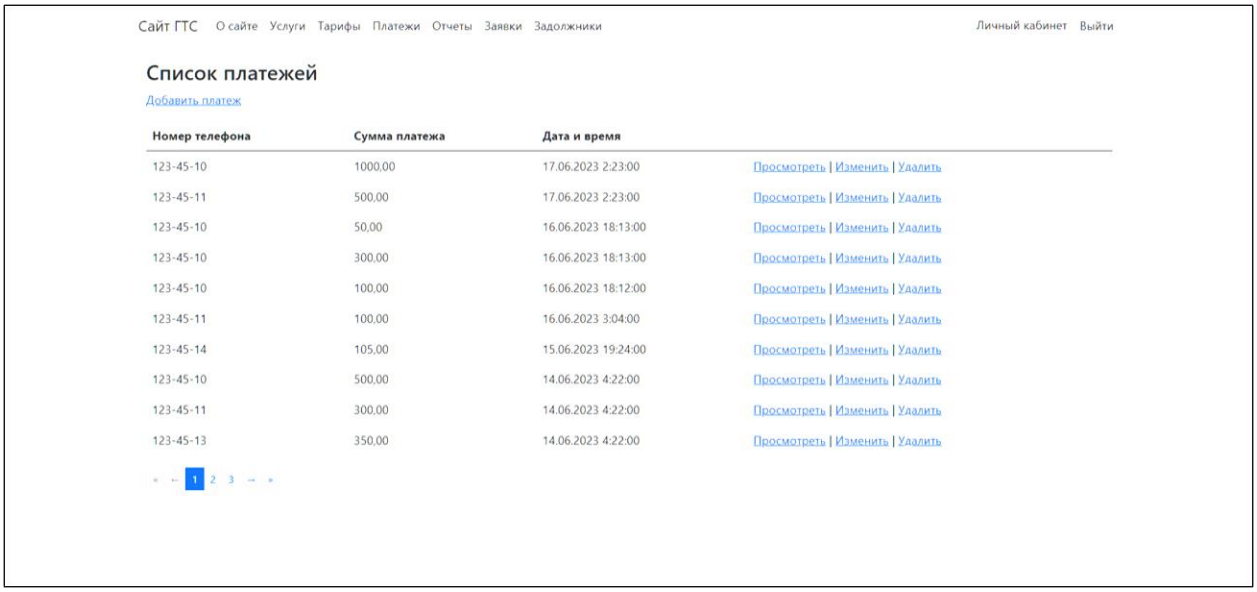

Рисунок Р.7 – Страница со списком платежей

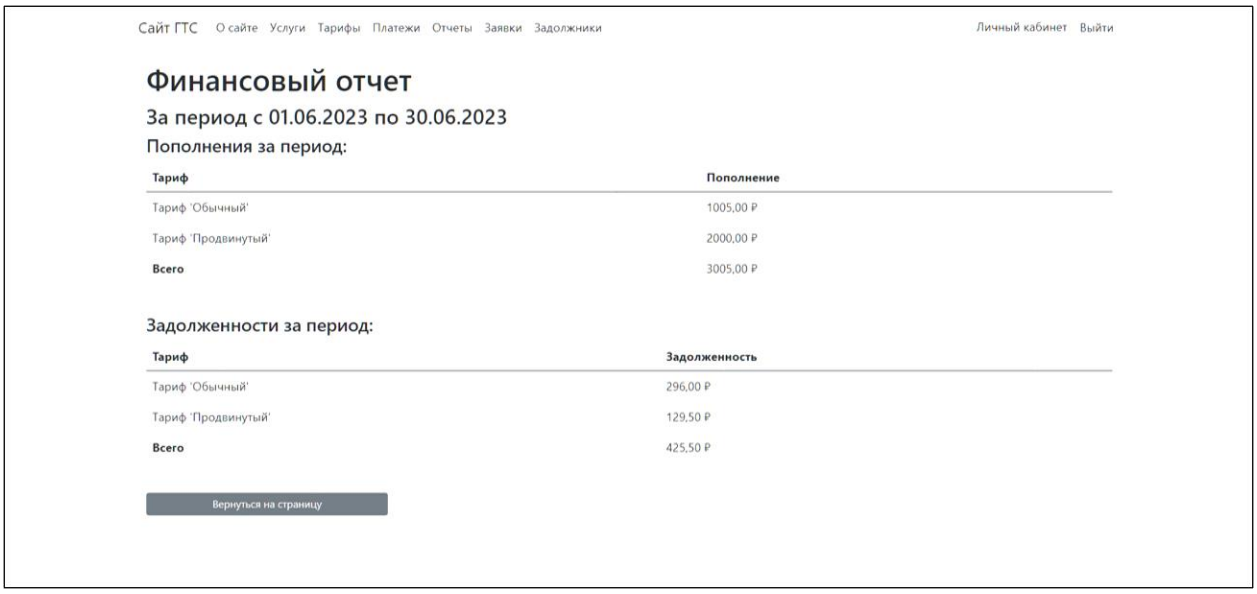

Рисунок Р.8 – Страница формирования финансового отчета

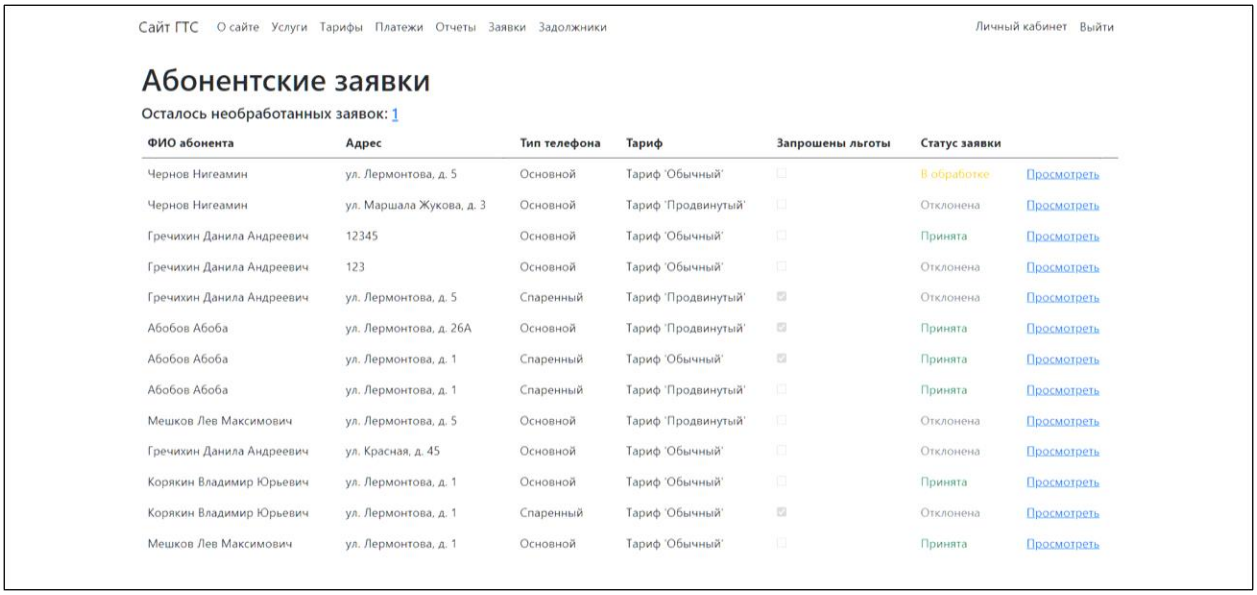

Рисунок Р.9 – Страница абонентских заявок

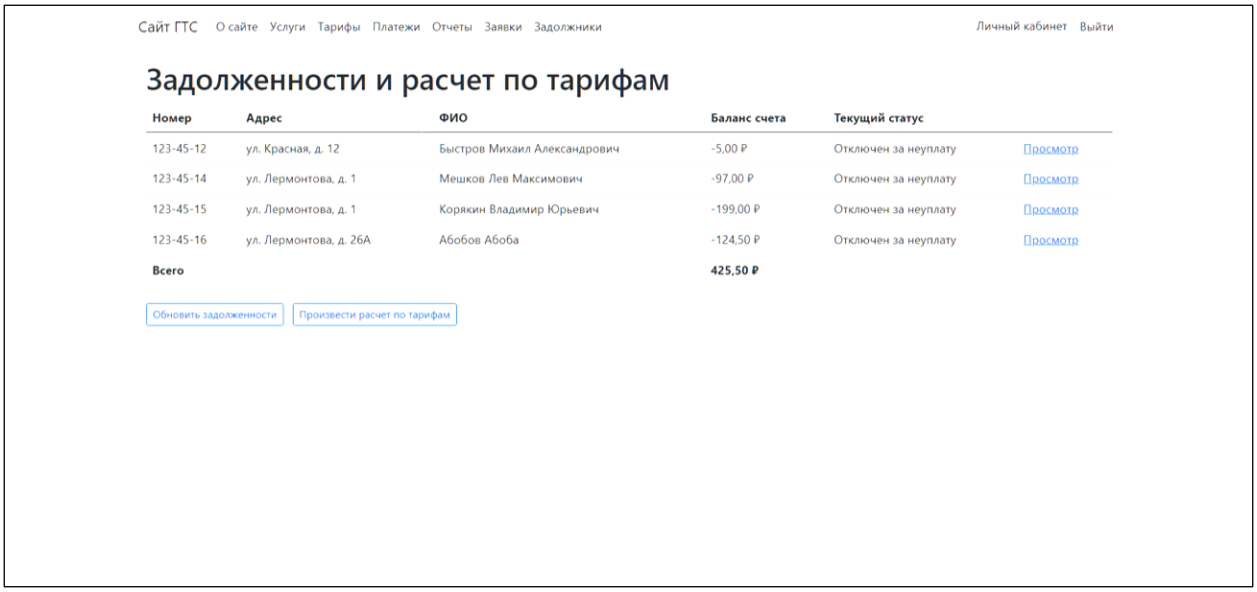

Рисунок Р.10 – Страница со списком задолжников

Приложение С

РЕЗУЛЬТАТЫ МОДУЛЬНОГО ТЕСТИРОВАНИЯ

(рекомендуемое)

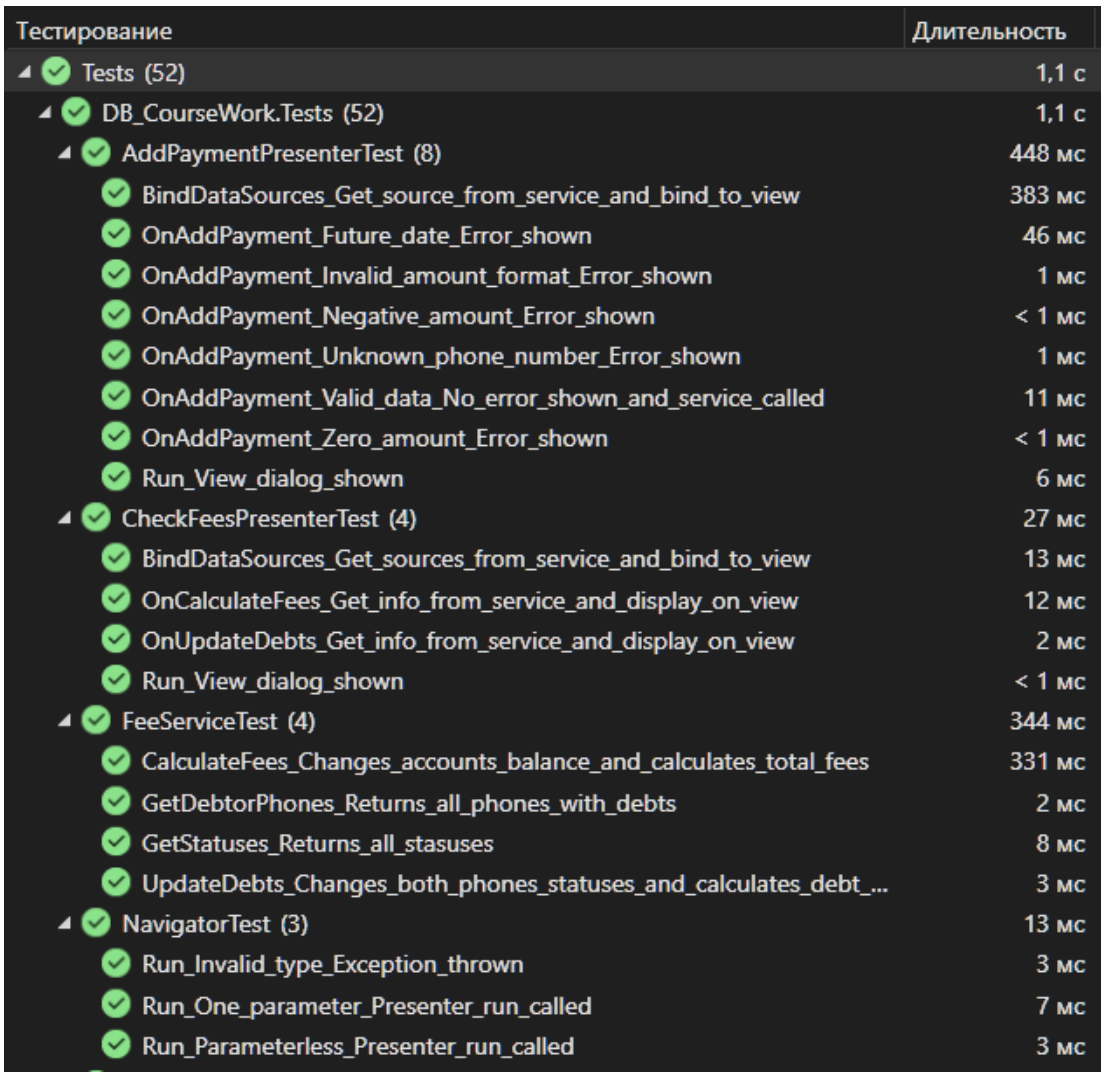

Рисунок С.1 – Результаты модульного тестирования разработанного приложения

Приложение Т

ЛИСТИНГ КОДА МОДУЛЬНЫХ ТЕСТОВ ДЛЯ МОДУЛЯ РАБОТЫ С ПЛАТЕЖАМИ

(рекомендуемое)

```
using Microsoft.EntityFrameworkCore;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using MockQueryable.Moq;
using Moq;
using PbxPaymentsMvc.Application.Dtos;
using PbxPaymentsMvc.Application.Services;
using PbxPaymentsMvc.Application.Services.Abstractions;
using PbxPaymentsMvc.Domain.Entities;
using PbxPaymentsMvc.Infrastructure.Data;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace PbxPaymentsMvc.Tests
{
    [TestClass]
    public class PaymentServiceTest
    {
        private Mock<PbxContext> _contextMock;
        private Mock<ISubscriberPhoneService> _phoneServiceMock;
        private List<Payment> _payments;
        private List<SubscriberPhone> _phones;

        [TestInitialize]
        public void InitializeTest()
        {
            _phones = new List<SubscriberPhone>()
            {
                new SubscriberPhone("100-00-01", null, "")
                    { Id = 1 },
                new SubscriberPhone("100-00-02", null, "")
                    { Id = 2 }
            };

            Mock<DbSet<SubscriberPhone>> phonesMock =
_phones.AsQueryable().BuildMockDbSet();

            _payments = new List<Payment>()
            {
                new Payment(_phones[0], 100, DateTime.Now)
                    { Id = 1, PhoneId = _phones[0].Id },
                new Payment(_phones[0], 150, DateTime.Now)
                    { Id = 2, PhoneId = _phones[0].Id },
                new Payment(_phones[0], 200, DateTime.Now)
                    { Id = 3, PhoneId = _phones[0].Id }
            };

            Mock<DbSet<Payment>> paymentsMock =
               _payments.AsQueryable().BuildMockDbSet();

            paymentsMock.Setup(p =>
               p.Add(It.IsAny<Payment>())).Callback<Payment>(p =>
```

```
{
    p.Id = _payments.Count + 1;
    _payments.Add(p);
});

paymentsMock.Setup(p =>
   p.Attach(It.IsAny<Payment>()))
   .Callback<Payment>(p =>
{
    p.Id = _payments.Count + 1;
    _payments.Add(p);
});

paymentsMock.Setup(p =>
   p.FindAsync(It.IsAny<object[]>()))
   .Returns<object[]>(id =>
{
    return
        ValueTask.FromResult(_payments.FirstOrDefault(
             payment => payment.Id == (int)id[0]));
});

paymentsMock.Setup(p =>
   p.Remove(It.IsAny<Payment>())).Callback<Payment>(p
   =>
   {
        _payments.Remove(p);
   });

paymentsMock.Setup(p =>
   p.Update(It.IsAny<Payment>())).Callback<Payment>(p
   =>
   {
        _payments.FirstOrDefault(payment =>
        payment.Id == p.Id).Amount = p.Amount;
   });

paymentsMock.Setup(m =>
   m.RemoveRange(It.IsAny<IEnumerable<Payment>>()))
   .Callback<IEnumerable<Payment>>(payments =>
{
    foreach (var payment in payments.ToList())
    {
        _payments.Remove(payment);
    }
});

_contextMock = new Mock<PbxContext>();
_contextMock.Setup(c =>
   c.SubscriberPhones).Returns(phonesMock.Object);
_contextMock.Setup(c =>
   c.Payments).Returns(paymentsMock.Object);
```

```
        _phoneServiceMock =
           new Mock<ISubscriberPhoneService>();
        _phoneServiceMock
            .Setup(s =>
                s.SetSubscriberPhoneStatusAsync(
                     It.IsAny<int>(),
                          It.IsAny<PhoneStatusEnum>(),
                     It.IsAny<string>()))
            .Callback<int, PhoneStatusEnum, string>((id,
                status, comment) =>
            {
                var statusEntry = new StatusHistoryEntry(id,
                     status, DateTime.Now, comment);
                _phones.FirstOrDefault(p => p.Id ==
                     id)?.StatusHistory?.Add(statusEntry);
            });
    }

    [TestMethod]
    public async Task
      FindPaymentByIdAsync_Existing_id_Returns_payment()
    {
        var service = new PaymentService(_contextMock.Object,
           _phoneServiceMock.Object);

        int paymentId = _payments.First().Id;
        PaymentDto payment = await
           service.FindPaymentByIdAsync(paymentId);

        Assert.IsNotNull(payment);
        Assert.AreEqual(paymentId, payment.Id);
    }

    [TestMethod]
    public async Task
      FindPaymentByIdAsync_Non_existent_id_Returns_null()
    {
        var service = new PaymentService(_contextMock.Object,
           _phoneServiceMock.Object);
        int paymentId = 0;

        PaymentDto payment = await
           service.FindPaymentByIdAsync(paymentId);

        Assert.IsNull(payment);
    }

    [TestMethod]
    public async Task
      GetAllPaymentsAsync_Returns_all_payments()
    {
        var service = new PaymentService(_contextMock.Object,
           _phoneServiceMock.Object);
```

```
        List<int> sourceSetIds = _payments.Select(p =>
           p.Id).ToList();

        List<int> returnedPaymentIds = (await
           service.GetAllPaymentsAsync()).Select(p =>
           p.Id).ToList();

        CollectionAssert.AreEquivalent(sourceSetIds,
           returnedPaymentIds);
    }

    [TestMethod]
    public async Task
      GetSubscriberPaymentsAsync_Subscriber_has_payments_
      Returns_subscriber_payments()
    {
        var service = new PaymentService(_contextMock.Object,
           _phoneServiceMock.Object);
        int subscriberId = _phones[0].Id;
        List<int> sourceSetIds = _payments.Where(p =>
           p.PhoneId == subscriberId)
           .Select(p => p.Id).ToList();

        List<int> returnedPaymentIds = (await
           service.GetSubscriberPaymentsAsync(subscriberId))
           .Select(p => p.Id).ToList();

        CollectionAssert.AreEquivalent(sourceSetIds,
           returnedPaymentIds);
    }

    [TestMethod]
    public async Task
      GetSubscriberPaymentsAsync_Subscriber_has_no_payments_
      Returns_no_payments()
    {
        var service = new PaymentService(_contextMock.Object,
           _phoneServiceMock.Object);
        int subscriberId = _phones[1].Id;

        IEnumerable<PaymentDto> returnedPayments = await
           service.GetSubscriberPaymentsAsync(subscriberId);

        Assert.AreEqual(0, returnedPayments.Count());
    }

    [TestMethod]
    public async Task
      AddPaymentAsync_Adds_payment_and_changes_balance()
    {
        var service = new PaymentService(_contextMock.Object,
           _phoneServiceMock.Object);
        int subscriberId = _phones[0].Id;
```

```
        decimal amount = 100;

        decimal oldBalance = _phones[0].AccountBalance;
        int oldCount = _payments.Count;

        await service.AddPaymentAsync(new PaymentDto()
        {
            PhoneId = subscriberId,
            Amount = amount,
            DateTime = DateTime.Now
        });

        decimal newBalance = _phones[0].AccountBalance;
        int newCount = _payments.Count;
        Assert.AreNotEqual(oldCount, newCount);
        Assert.AreEqual(oldCount + 1, newCount);
        //Assert.AreNotEqual(oldBalance, newBalance);
        //Assert.AreEqual(oldBalance + amount, newBalance);
    }

    [TestMethod]
    public async Task
      RemovePaymentByIdAsync_Payment_exists_Removes_payment()
    {
        var service = new PaymentService(_contextMock.Object,
           _phoneServiceMock.Object);
        Payment payment = _payments.First();
        int oldCount = _payments.Count;

        await service.RemovePaymentByIdAsync(payment.Id);
        int newCount = _payments.Count;

        Assert.AreNotEqual(oldCount, newCount);
        Assert.AreEqual(oldCount - 1, newCount);
        CollectionAssert.DoesNotContain(_payments, payment);
    }

    [TestMethod]
    public async Task
      RemovePaymentByIdAsync_Payment_doesnt_exists_No_changes()
    {
        var service = new PaymentService(_contextMock.Object,
           _phoneServiceMock.Object);
        int paymentId = 0;
        int oldCount = _payments.Count;

        await service.RemovePaymentByIdAsync(paymentId);
        int newCount = _payments.Count;

        Assert.AreEqual(oldCount, newCount);
    }

    [TestMethod]
```

```
        public async Task
PaymentExistsAsync_Payment_exists_Returns_true()
        {
            var service = new PaymentService(_contextMock.Object,
               _phoneServiceMock.Object);
            int paymentId = _payments.First().Id;

            bool exists = await
               service.PaymentExistsAsync(paymentId);

            Assert.IsTrue(exists);
        }

        [TestMethod]
        public async Task UpdatePaymentAsync_Updates_payment()
        {
            var service = new PaymentService(_contextMock.Object,
               _phoneServiceMock.Object);
            Payment payment = _payments.First();
            decimal oldAmount = payment.Amount;
            decimal newAmount = oldAmount + 100;

            await service.UpdatePaymentAsync(new PaymentDto()
            {
                Id = payment.Id,
                PhoneId = payment.PhoneId,
                Amount = newAmount,
                DateTime = payment.DateTime
            });

            Assert.AreNotEqual(oldAmount,
               _payments.First().Amount);
            Assert.AreEqual(newAmount, _payments.First().Amount);
        }
    }
}
```