

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

Веб-приложение планирования путешествий

Студент	_____	<u>Морозов Александр Валентинович</u>
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	<u>Балашова И.Ю.</u>
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	<u>Такташкин Д.В.</u>
	(подпись, дата)	(фамилия, инициалы)

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

Заведующий кафедрой	_____	<u>Макарычев П.П.</u>
	(подпись)	(фамилия, инициалы)

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

Секретарь ГЭК	_____	<u>Попова Н.А.</u>
	(подпись)	(фамилия, инициалы)

Пенза, 2020

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

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ П.П. Макарычев
«__» _____ 2020 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Морозов Александр Валентинович гр. 16ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Веб-приложение планирования путешествий

Тема утверждена приказом ПГУ № 1488/о от "30" декабря 2019 г.

2. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Балашова И.Ю.

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

1) *Назначение: приложение предназначено для планирования поездок*

2) *Среда выполнения: операционная система Windows, начиная с версии Windows 7.*

Средства разработки: языки программирования JavaScript, SQL.

Инструментальные средства: IntelliJ IDEA

Технологии разработки: ООП, UML.

3) *Основные функции веб-приложения планирования путешествий;*

а) вход в аккаунт;

б) регистрация в приложении;

в) планирование поездки;

г) сохранение поездки;

д) удаление поездки;

е) планирование маршрута поездки;

ж) планирование досуга в поездке;

з) составление списка покупок;

и) публикация плана поездки в соцсетях;

к) печать плана поездки.

4. Перечень подлежащих разработке вопросов

Анализ предметной области

Анализ требований

Проектирование

Реализация

Контроль качества ПО:

– *Тестирование*

– *Анализ качества кода*

План разработки и оценка бюджета

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований	10	13.04.2020 – 17.04.2020	
2	Проектирование	25	29.04.2020 – 01.05.2020	
3	Реализация	35	02.05.2020 – 23.05.2020	
4	Контроль качества	15	24.05.2020 – 29.05.2020	
5	План разработки бюджета	5	30.05.2020 – 04.06.2020	
6	Оформление пояснительной записки	10	05.06.2020 – 13.06.2020	

Дата выдачи задания " 30 " декабря 2019 г.

Руководитель бакалаврской работы _____

(подпись, дата)

Задание к исполнению принял студент _____

(подпись, дата)

Работу к защите допустить

Декан факультета _____

(подпись, дата)

Реферат

Пояснительная записка содержит 100 листов, 50 рисунков, 25 таблиц, 14 использованных источников.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, ТУРИЗМ, UML, БАЗА ДАННЫХ, КЛИЕНТ, СЕРВЕР, POSTGRE SQL, JAVA SCRIPT.

Объект разработки – веб-приложение планирования путешествий.

Цель работы – разработать веб-приложение, позволяющее пользователю спланировать свои будущие поездки.

Технология разработки – клиентская часть приложения реализована на языке JavaScript с использованием фреймворка React.js, серверная часть реализована на платформе Node.js с использованием фреймворка Express.js, а в качестве системы управления базами данных использовался PostgreSQL.

Результаты работы – в ходе выполнения выпускной квалификационной работы разработано веб-приложение планирования путешествий. Все требования к разработке системы соблюдены.

					ПГУ 09.03.02–8ВР161.19 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Морозов А.В.			«Веб-приложение планирования путешествий». Пояснительная записка	Лит.	Лист	Листов
Провер.		Балашова И.Ю.					3	100
Н. Контр.		Такташкин Д.В.				Группа 16ВИ1		
Утверд.		Макарычев П.П.						

Содержание

Введение.....	6
1 Анализ требований, бизнес процессов и постановка задачи на разработку веб-приложения планирования путешествий	7
1.1 Анализ приложений планирования путешествий.....	7
1.2 Постановка задачи.....	11
1.3 Анализ функциональных требований	12
1.4 Анализ требований к пользовательскому интерфейсу приложения	15
1.5 Анализ требований к средствам разработки приложения	16
1.6 Функциональное моделирование	17
2 Проектирование веб-приложения планирования путешествий	21
2.1 Базовая архитектура системы	21
2.2 Логическая модель базы данных	21
2.3 Физическая модель базы данных.....	25
2.4 Проектирование программных средств	26
2.4.1 Диаграмма классов.....	26
2.4.2 Диаграмма состояний	27
2.4.3 Диаграмма деятельности	28
2.4.4 Диаграмма последовательности	30
2.4.5 Диаграмма кооперации.....	31
2.5 Проектирование пользовательского интерфейса.....	32
3 Реализация веб-приложения планирования путешествий	39
3.1 Диаграмма компонентов.....	39
3.2 Диаграмма развертывания.....	41
4 Контроль качества веб-приложения планирования путешествий	43

4.1 Функциональное тестирование.....	43
4.1.1 Определение классов эквивалентности	43
4.1.2 Анализ граничных условий.....	44
4.1.3 Построение функциональной диаграммы	46
4.1.4 Парное тестирование	47
4.2 Структурное тестирование.....	48
4.2.1 Метод покрытия операторов.....	50
4.2.2 Метод покрытия условий	50
4.2.3 Метод покрытия маршрутов	51
4.3 Метрики программного кода проекта.....	52
4.4 Планирование и оценка бюджета проекта.....	54
Заключение	60
Список использованных источников	61
Приложение А. Глоссарий	63
Приложение Б. Листинг клиентского приложения на языке JavaScript.....	65
Приложение В. Листинг серверного приложения на языке JavaScript.....	86
Приложение Г. Листинг обработки запросов к базе данных	91
Приложение Д. Результаты тестирования	95

Введение

В современном обществе Интернет играет большую роль в жизни человека. Он является одним из основных источников информации и предоставляет инструменты для ее поиска. Следовательно, высока роль средств, с помощью которых человек может получать информацию из Интернета. Разработка веб-приложения может позволить использовать современные технологии для работы с информацией, а также обеспечить доступ к ней большому количеству пользователей.

Туризм является одной из самых быстро развивающихся отраслей мировой экономики. Развитие туризма в стране может позволить улучшить экономическое и социальное состояние, так как он играет значительную роль в формировании валового внутреннего продукта, а также помогает в создании рабочих мест и обеспечении занятости населения [1].

Большая часть подготовительных к поездке действий, таких как бронирование отеля, покупка авиабилетов, получение информации о предстоящей поездке, совершается в Интернете. Создание приложения, позволяющего перенести туда планирование поездки, упростит процесс сбора и подготовки к ней [2].

Целью данной выпускной квалификационной работы является разработка веб-приложения планирования путешествий.

Актуальность разработки такого приложения обусловлена высокой востребованностью в сфере туризма инструмента для планирования будущих поездок и ограниченным функционалом присутствующих на рынке приложений. Большая часть приложений не предлагает инструментарий для того, чтобы сохранить план поездки, или же поделиться им в социальных сетях и вынуждают пользователя пользоваться большим количеством сторонних сервисов.

1 Анализ требований, бизнес процессов и постановка задачи на разработку

1.1 Анализ приложений планирования путешествий

Перед планированием путешествия в Интернете пользователь чаще всего уже принял часть ключевых решений – куда, зачем и когда он едет. Также у путешественника, скорее всего, есть ряд предпочтений – бюджет, любимые бренды и так далее.

Большинство приложений не дают возможности хранить свой список поездок. Они предлагают небольшой функционал, например, просмотр погоды или создание списков нужных для поездки вещей, а также не дают возможности пользователю поделиться полученной информацией в удобном виде. Пользователю приходится использовать большое количество различных ресурсов и вести списки вручную, например, с помощью Excel или Word на что уходит большое количество времени.

Поэтому создание приложения, которое позволит создать план поездки, включающий в себя дату, местоназначение, цель, список необходимых вещей и покупок, может значительно упростить процесс подготовки к путешествию.

Было проведено исследование наиболее популярных приложений планирования путешествий, таких как: «Tripit», «CheckandPack», «CheckMyTrip».

Приложение «Tripit» представлено как в виде веб-приложения, так и мобильного приложения. Данное приложение предоставляет возможность сформировать поездку, создать список планов, сохранить данные своих документов, просмотреть статистику совершённых путешествий, а также настроить автоматический импорт данных о поездке на электронную почту. Мобильное приложение удобно в использовании и имеет дружелюбный интерфейс. Основным минусом этого приложения можно назвать отсутствие

поддержки русского языка. Интерфейс приложения представлен на рисунке 1.

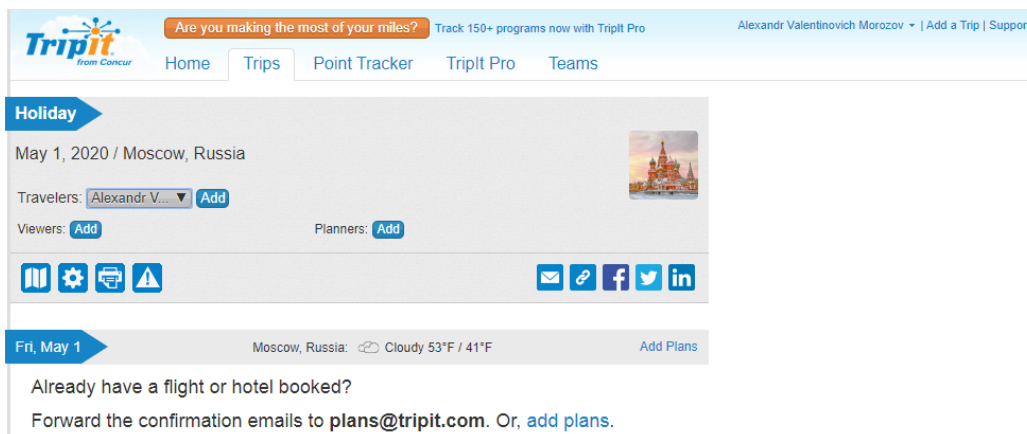


Рисунок 1 – Интерфейс приложения «Tripit»

Приложение «CheckandPack» представлено как в виде веб-приложения, так и мобильного приложения. Данное приложение предоставляет возможность сформировать список необходимых в поездке вещей, записать текущие траты, добавить посещаемые города и добавить заметки к поездке. Основными недостатками являются отсутствие русского языка и ограниченность функционала приложения. Интерфейс приложения представлен на рисунке 2.

Рисунок 2 – Интерфейс приложения «CheckandPack»

Приложение «CheckMyTrip» представлено только в виде мобильного приложения. Данное приложение предоставляет возможность сохранить данные о рейсе, отеле, арендованном автомобиле, а также о предстоящих мероприятиях. Основными недостатками являются отсутствие веб версии приложения, что лишает возможности пользоваться этим приложением не с мобильного устройства, а также ограниченность его функционала. Интерфейс приложения представлен на рисунке 3.

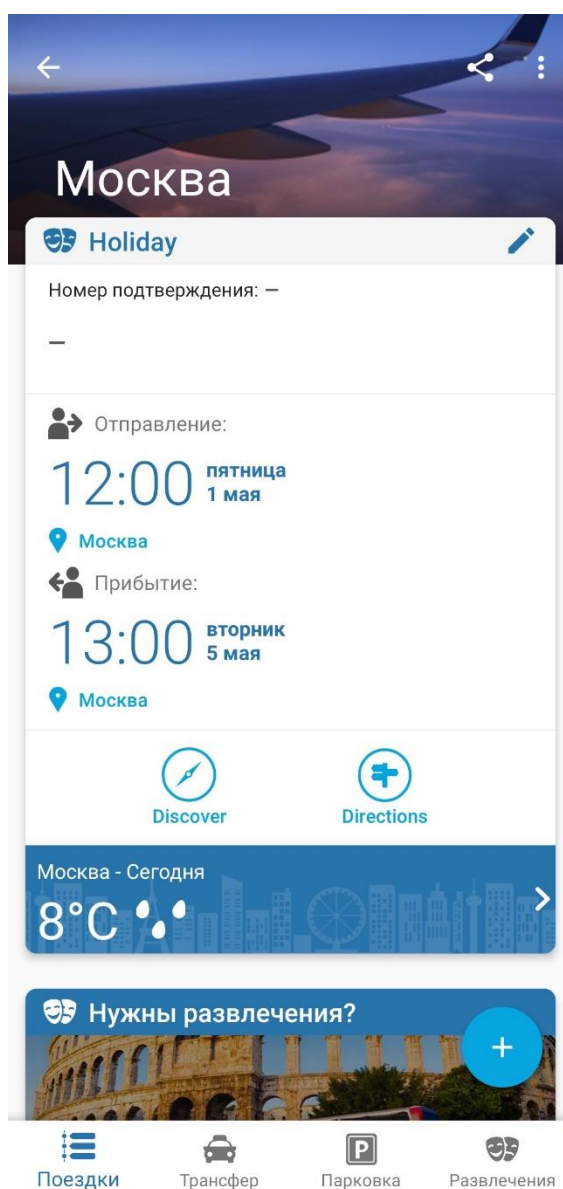


Рисунок 3 – Интерфейс приложения «CheckMyTrip»

Рассмотренные программы обладают различным функционалом и характеристиками. В результате проведенного анализа были выделены

критерии, напрямую влияющие на удобство и эффективность применения приложения:

- возможность создания списка необходимых в поездке вещей;
- возможность поделиться информацией о поездке со спутниками;
- возможность создания плана поездки на выбранные даты;
- эргономичный интерфейс;
- планирование покупок;
- наличие русского языка.

Полученные результаты сравнения приведены в таблице 1.

Таблица 1 – Сравнение по критериям

Функциональные характеристики	«Tripit»	«Check and Pack»	«Check My Trip»
Создание списка необходимых вещей	Нет	В зависимости от планов на поездку создается список вещей, которые необходимо взять с собой	Нет
Возможность поделиться планом поездки	Возможно поделиться с помощью социальных сетей, отправить на почту, а также получить прямую ссылку на план путешествия.	Нет	Нет
Эргономичный интерфейс	Удобный и понятный интерфейс, позволяющий быстро понять где найти нужный функционал	Простой и удобный интерфейс	Простой и удобный интерфейс. Основное неудобство – отсутствие возможности создать отдельную поездку, не привязывая ее к рейсу, отелю или мероприятию.
Возможность покупки билетов через приложение	Нет	Нет	Возможно купить билеты на различные экскурсии и мероприятия.

Продолжение таблицы 1

Функциональные характеристики	«Tripit»	«Check and Pack»	«Check My Trip»
Управление планами во время путешествия	Возможно создавать планы на каждый день и добавлять дополнительную информацию к ним.	Возможно создать простой список действий на поездку.	Нет
Поддержка русского языка	Нет	Нет	Есть
Планирование покупок	Нет	Нет	Нет

Из результатов сравнения можно сделать вывод, что каждое приложение обладает определенным функционалом, но не решает всех проблем, с которыми может столкнуться путешественник. В путешествиях люди часто совершают покупки. Использование приложения для их планирования может значительно упростить этот процесс, но у рассмотренных приложений данный функционал не реализован. Также, у большей части рассмотренных приложений отсутствует поддержка русского языка, что сокращает аудиторию, которая может пользоваться приложением.

Таким образом, при разработке веб-приложения планирования путешествий будут учтены данные недостатки и реализован весь необходимый функционал.

1.2 Постановка задачи

Данная выпускная квалификационная работа посвящена разработке веб-приложения планирования путешествий. Приложение создается для того, чтобы упростить для пользователя подготовку к будущим поездкам.

Так как приложение ориентировано на обычных пользователей, оно должно обладать простым и интуитивно понятным интерфейсом. Для удобного хранения данных и доступа к ним необходимо разработать базу

данных (БД), которая будет хранить информацию о пользователях и их поездках.

1.3 Анализ функциональных требований

Анализ функциональных требований является важным этапом проектирования системы. На данном этапе необходимо сформулировать функционал разрабатываемого приложения. Для того, чтобы отобразить эти требования, была спроектирована диаграмма вариантов использования в нотации Unified Modeling Language (UML). Диаграмма вариантов использования представлена на рисунке 4.

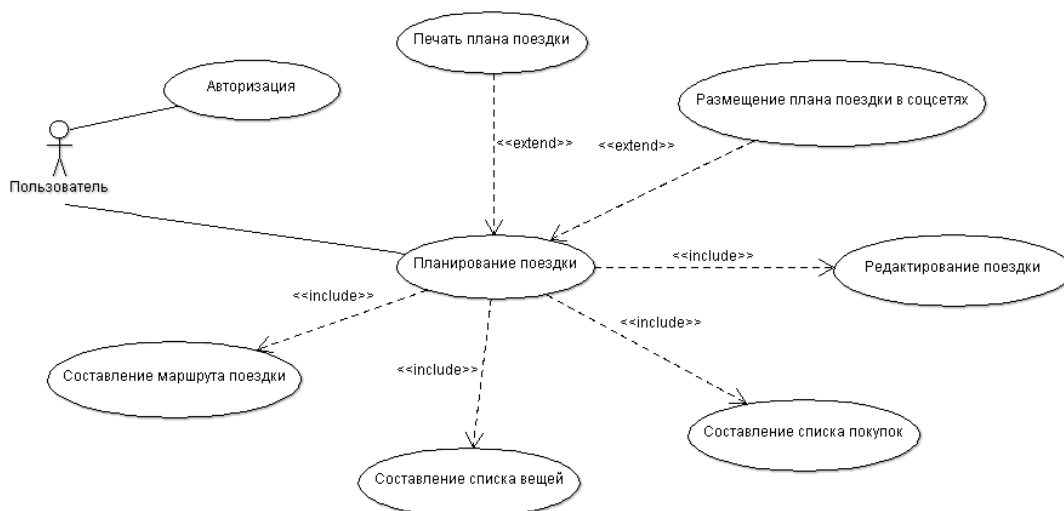


Рисунок 4 – Диаграмма вариантов использования

Актером в данной диаграмме является пользователь приложения – лицо, которому необходимо спланировать свои будущие поездки. Для него были выделены следующие функции: «Авторизация», «Управление списком поездок».

«Авторизация» – пользователь без аккаунта может зарегистрироваться в системе, при желании воспользоваться дополнительным функционалом. Если у него есть аккаунт, то он может войти в него.

«Планирование поездки» – пользователь может спланировать поездку: добавить даты начала и конца поездки, место назначения и цель, составить

список вещей, нужных в поездке, сформировать список покупок. Также пользователь может распечатать план поездки и поделиться им в соцсетях.

Далее будут подробнее рассмотрены основные прецеденты диаграммы [3].

Описание прецедента «Авторизация» представлено в таблице 2.

Таблица 2 – Описание прецедента «Авторизация»

Название пункта	Описание
Краткое описание	Пользователь авторизуется в приложении
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловие	Пользователь входит в приложение
Основной поток	Пользователь авторизуется
Постусловие	Пользователь авторизован
Альтернативные потоки	1. Пользователь не авторизуется и ему доступен стандартный функционал приложения. 2. Пароль или логин введены не верно и система уведомляет пользователя об ошибке.

В ходе авторизации пользователь попытается войти в систему, а в случае ошибки получить обратную связь.

Описание прецедента «Планирование поездки» представлено в таблице 3.

Таблица 3 – Описание прецедента «Планирование поездки»

Название пункта	Описание
Краткое описание	Пользователь составляет план поездки
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловие	Пользователь входит в приложение
Основной поток	Пользователь переходит на страницу интересующей его поездки
Точка расширения	1. Необходимо распечатать план поездки 2. Необходимо поделиться планом поездки в соцсетях
Постусловие	На странице отображается информация о планируемой поездке
Альтернативные потоки	Нет

Пользователь может просматривать и редактировать план поездки, а также распечатать его или поделиться им в соцсетях.

Описание прецедента «Составление маршрута поездки» представлено в таблице 4.

Таблица 4 – Описание прецедента «Составление маршрута поездки»

Название пункта	Описание
Краткое описание	Пользователь составляет маршрут поездки
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловие	Пользователь переходит на страницу поездки
Основной поток	Пользователь выбирает даты поездки, местоназначение и цель
Постусловие	Маршрут поездки составлен
Альтернативные потоки	Нет

Пользователь может вносить и редактировать информацию о маршруте поездки.

Описание прецедента «Составление списка вещей» представлено в таблице 5.

Таблица 5 – Описание прецедента «Составление списка вещей»

Название пункта	Описание
Краткое описание	Пользователь формирует список вещей
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловие	Пользователь переходит на страницу поездки
Основной поток	Пользователь формирует список из вещей, которые ему нужно взять с собой, и вносит информацию в таблицу
Постусловие	Таблица со списком вещей заполнена
Альтернативные потоки	Нет

Пользователь может просматривать и заполнять список вещей в соответствии с их категориями.

Описание прецедента «Составление списка покупок» представлено в таблице 6.

Таблица 6 – Описание прецедента «Составление списка покупок»

Название пункта	Описание
Краткое описание	Пользователь формирует список покупок
Главные актеры	Пользователь
Второстепенные актеры	Нет
Название пункта	Описание
Предусловие	Пользователь переходит на страницу поездки
Основной поток	Пользователь формирует список из вещей, которые ему нужно купить за время поездки, и вносит их в таблицу
Постусловие	Таблица со списком покупок заполнена
Альтернативные потоки	Нет

Пользователь может просматривать и заполнять список покупок, планировать затраты.

Описание прецедента «Составление списка покупок» представлено в таблице 7.

Таблица 7 – Описание прецедента «Редактирование поездки»

Название пункта	Описание
Краткое описание	Пользователь редактирует поездку
Главные актеры	Пользователь
Второстепенные актеры	Нет
Предусловие	Пользователь переходит на страницу поездки
Основной поток	Пользователь редактирует информацию о поездке
Постусловие	Информация о поездке изменена
Альтернативные потоки	Нет

Пользователь может вносить изменения в информацию о поездке.

В результате анализа предметной области выявлено, что система должна обеспечивать возможность:

- зарегистрироваться в системе;
- войти в систему при наличии аккаунта;
- управлять списком поездок;
- планировать текущую поездку;
- сохранить поездку для последующего использования;
- распечатать план поездки;

- поделиться информацией поделиться планом поездки в соцсетях.

1.4 Анализ требований к пользовательскому интерфейсу приложения

Требования к пользовательскому интерфейсу являются одними из основных для определения функционала и дизайна приложения. При анализе принципов материального дизайна были выведены следующие требования к интерфейсу приложения:

- простота и понятность;
- высокая скорость работы;
- обратная связь на действия пользователя;
- однородность элементов, выполняющих схожие функции;
- схожий графический стиль всех элементов;
- возможность вернуться к предыдущему состоянию.

1.5 Анализ требований к средствам разработки приложения

На основе изучения различных средств для разработки приложения был выведен ряд требований:

- поддержка языка JavaScript;
- удобная работа с клиентской и серверной частью приложения, а также с БД;
- удобное подключение и работа с различными сторонними фреймворками.

Исходя из вышеперечисленных требований в качестве среды разработки была выбрана IntelliJ IDEA, которая в полном объеме отвечает заданным требованиям.

В качестве системы управления базой данных (СУБД) была выбрана PostgreSQL, которая является объектно-реляционной и имеет открытый исходный код. Она поддерживает большую часть стандарта SQL и предлагает множество современных функций: сложные запросы, внешние

ключи, триггеры, изменяемые представления, транзакционная целостность, а также многоверсионность [4].

Для реализации серверной части была выбрана программная платформа Node.js. Экосистема Node предоставляет фреймворки, библиотеки и утилиты, ускоряющие разработку и поощряющие написание хорошего кода [5]. Также были выбраны фреймворки:

- Express – один из самых популярных фреймворков, предназначен для создания веб-приложений и API;
- Passport.js – это промежуточное программное обеспечение для авторизации под node.js, которое предоставляет возможность использовать большой набор различных стратегий, а также поддерживает аутентификацию с помощью большого количества сервисов;
- Pg-promise – упрощает асинхронный доступ к БД.

Для реализации клиентской части были выбраны фреймворки:

- React.js – один из самых популярных фреймворков, предназначен для разработки интерфейсов, он представляет собой библиотеку для создания пользовательских интерфейсов, которая помогает определить пользовательский интерфейс один раз и при последующем изменении состояния приложения, пользовательский интерфейс перестраивается для реагирования на изменения [6];
- React-router – упрощает создание маршрутизации в приложении;
- Redux.js – фреймворк, облегчающий управление состоянием приложения, помогающий писать приложения со стабильным поведением, которые запускаются на клиенте, сервере и легко тестируются [7];
- Redux-Thunk – это промежуточное программное обеспечение, позволяющее работать с Redux.js асинхронно;
- Axios – упрощает работу с REST API;
- Material-UI – набор компонентов интерфейса, реализующих язык дизайна Google, упрощающий дизайн приложения.

1.6 Функциональное моделирование

Для разрабатываемого приложения планирования путешествий были определены бизнес-процессы и представлены с помощью диаграмм IDEF0, DFD и IDEF3. Для построения этих диаграмм было использовано CASE-средство CA ERwin Process Modeler.

Диаграмма IDEF0 позволяет представить и описать бизнес-процессы. Ее моделирование начинается с представления контекстной диаграммы – самого верхнего и абстрактного уровня. Контекстная диаграмма приложения планирования путешествий представлена на рисунке 5.

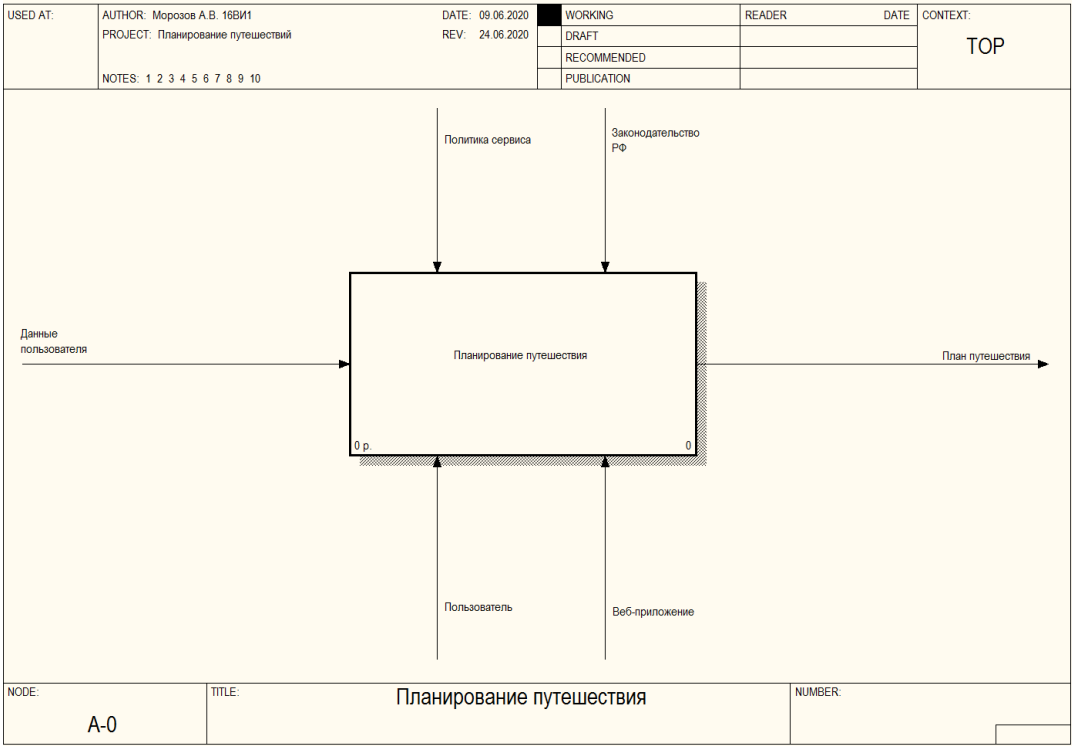


Рисунок 5 – Контекстная диаграмма

На этой диаграмме отображена основная функция системы.

После построения контекстной диаграммы происходит ее разделение – декомпозиция, путем создания диаграммы более низкого уровня. Диаграмма декомпозиции представлена на рисунке 6.

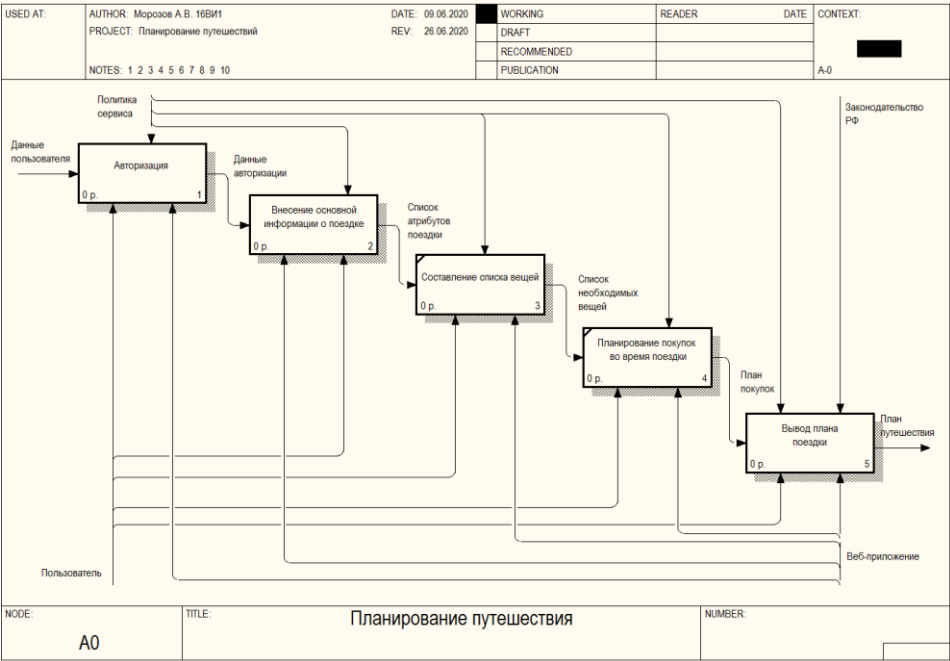


Рисунок 6 – Диаграмма декомпозиции

Пользователь авторизуется, вносит в приложение основную информацию о предстоящей поездке, после чего он формирует список вещей и определяет необходимые покупки. После этого клиенту предоставляется готовый план его будущей поездки.

Бизнес-процесс вывода информации о поездке декомпозирован с помощью DFD. Диаграмма потоков данных представлена на рисунке 7.

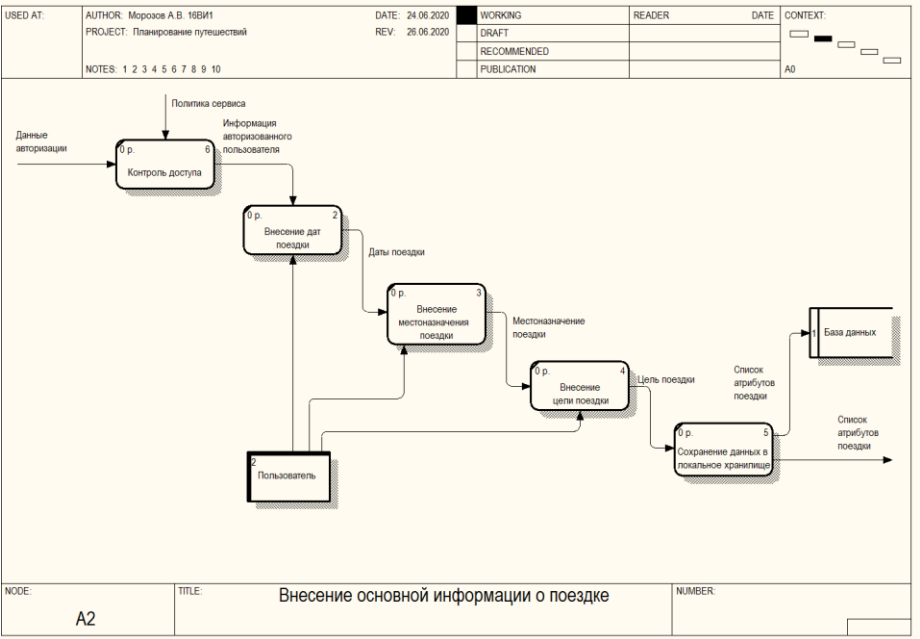


Рисунок 7 – Диаграмма потоков данных

После авторизации пользователь выбирает нужную поездку, информация о которой читается из БД. Затем он вносит основную информацию о поездке, такую как даты, местоназначение и цель. После чего эти данные сохраняются в локальное хранилище, чтобы избежать их потери.

Бизнес-процесс внесения общей информации о поездке декомпозирован с помощью IDEF3 – нотации, описывающей процессы, происходящие в системе. Диаграмма представлена на рисунке 8.

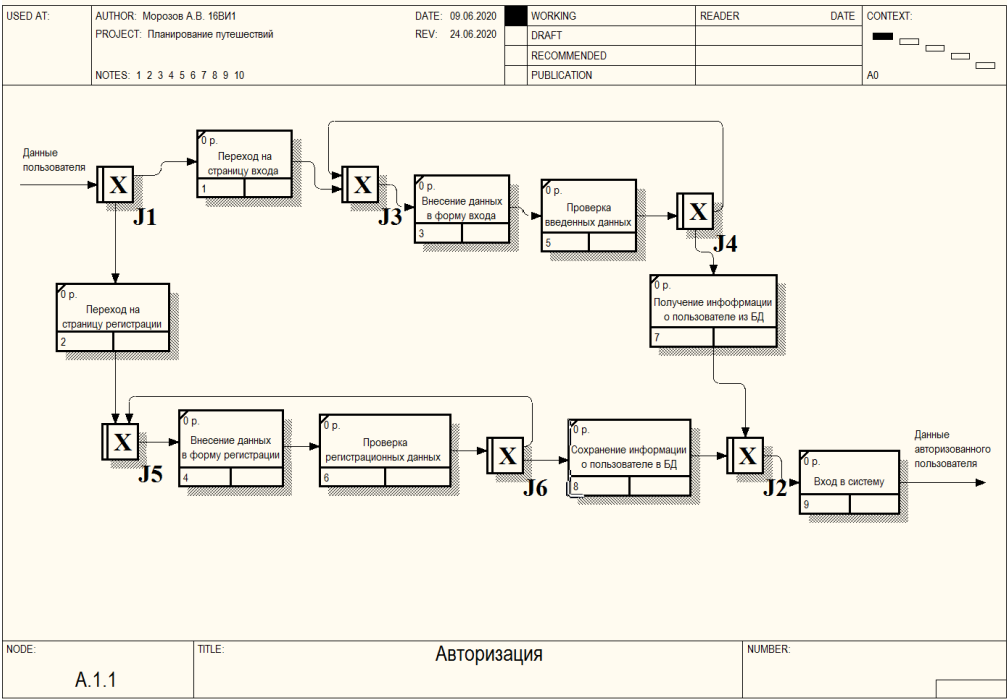


Рисунок 8 – Диаграмма IDEF3

Пользователь выбирает войти в приложение или зарегистрироваться. Чтобы войти в аккаунт, зарегистрированный пользователь переходит на страницу входа и вносит данные в форму. Если данные прошли проверку, то из БД выбирается информация о пользователе, иначе пользователю предлагается повторить ввод данных. Если пользователь решает зарегистрироваться, то он переходит на страницу регистрации и вносит данные в форму. Если данные прошли проверку, то происходит сохранение данных о пользователе в БД, иначе пользователю предлагается повторить ввод данных.

2 Проектирование веб-приложения планирования путешествий

2.1 Базовая архитектура системы

Разрабатываемое приложение планирования путешествий имеет двухуровневую архитектуру клиент-сервер. Архитектура системы представлена на рисунке 9.



Рисунок 9 – Клиент-серверная архитектура приложения

Данный вид архитектуры включает в себя три основных компонента: клиент, сервер и БД.

Первый компонент – клиент, представляет из себя компьютер пользователя, через который он может взаимодействовать с приложением. Клиент преобразует запросы пользователя и отправляет их на сервер, а также принимает ответы и преобразует их в понятный пользователю вид.

Вторым компонентом является сервер. Он обрабатывает запросы, которые приходят с клиента и отправляет запросы к БД.

Третьим компонентом является БД, в которой хранится информация о пользователе и его поездках.

2.2 Логическая модель базы данных

На основе проанализированной предметной области была построена логическая модель БД. Логическая модель не привязана к какой-либо СУБД и представляет из себя абстрактный взгляд на данные.

Проектирование было проведено с помощью CASE-средства CA Erwin Data Modeler в нотации IDEF1x [8].

Построенная логическая модель представлена на рисунке 10.

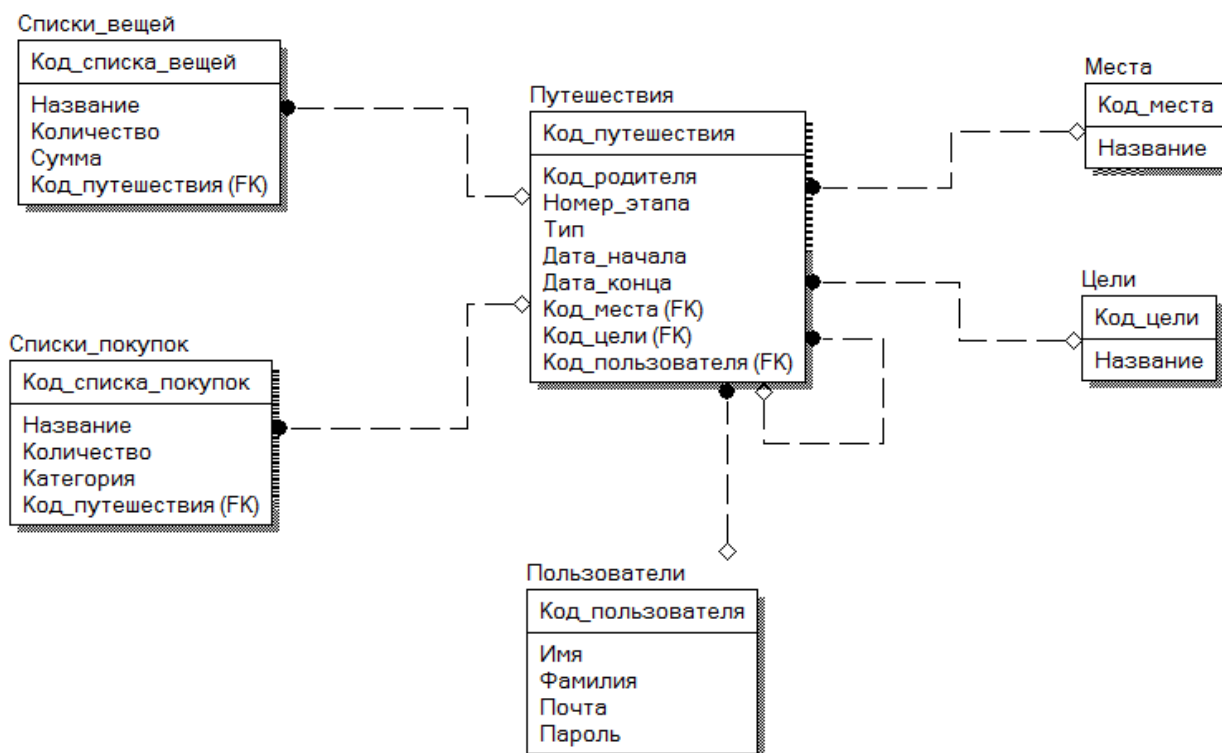


Рисунок 10 – Логическая схема БД

После того, как были выделены сущности и определены отношения между ними, необходимо нормализовать структуру БД.

Спроектированная логическая модель БД не включает в себя неявные сущности, атрибуты, зависящие только от части уникального ключа, и транзитивных зависимостей, следовательно, она соответствует третьей НФ.

Далее будут подробнее рассмотрены спецификации сущностей логической модели.

Спецификация сущности «Путешествия» приведена в таблице 8.

Таблица 8 – Спецификация сущности «Путешествия»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Код_путешествия	Номер путешествия	Целый	10	'0-9'	Нет	РК	1736
Код_родителя	Номер родителя	Целый	10	'0-9'	Да		3123

Продолжение таблицы 8

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Номер_этапа	Номер этапа в поездке	Целый	10	'0-9'	Да		3
Тип	Тип поездки	Тип_Поездки	7	'поездка', 'этап'	Да		этап
Дата_начала	Дата начала поездки	Дата	8	01.01.1970 – 31.12.2040	Да		24.05.2020
Дата_конца	Дата конца поездки	Дата	8	01.01.1970 – 31.12.2040	Да		07.06.2020
Код_места	Номер места	Целый	10	'0-9'	Да	FK	186
Код_цели	Номер цели	Целый	10	'0-9'	Да	FK	451

Сущность «Путешествия» предназначена для хранения списка поездок пользователя.

Спецификация сущности «Места» приведена в таблице 9.

Таблица 9 – Спецификация сущности «Места»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Код_места	Номер места	Целый	10	'0-9'	Нет	PK	181
Название	Наименование места	Символьный	64	'А-я', 'А-z'	Нет		Москва

Сущность «Места» – это справочник, в котором храниться список городов и стран.

Спецификация сущности «Цели» приведена в таблице 10.

Таблица 10 – Спецификация сущности «Цели»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Код_цели	Номер цели	Целый	10	'0-9'	Нет	PK	451
Название	Наименование цели	Символьный	64	'А-я', 'А-z'	Нет		Командировка

Сущность «Цели» – это справочник, в котором храниться список возможных целей поездки.

Спецификация сущности «Списки вещей» приведена в таблице 11.

Таблица 11 – Спецификация сущности «Списки вещей»

Наименование	Описание	Тип данных	Размер- ность	Область допустимых значений	Возмож- ность значения Null	Роль	Приме р
Код_списка_вещей	Номер списка вещей	Целый	10	'0-9'	Нет	PK	121
Название	Наименование вещи	Символьный	128	'0-9', 'А-я', 'A-z'	Да		Пас- порт
Количество	Количество вещей	Символьный	64	'0-9', 'А-я', 'A-z'	Да		3 шт
Категория	Наименование категории	Символьный	64	'0-9', 'А-я', 'A-z'	Да		Одеж- да
Код_путешествия	Номер путешествия	Целый	10	'0-9'	Нет	FK	1921

Сущность «Списки вещей» предназначена для хранения необходимых в поездке предметов.

Спецификация сущности «Списки покупок» приведена в таблице 12.

Таблица 12 – Спецификация сущности «Списки покупок»

Наименование	Описание	Тип данных	Размер- ность	Область допустимых значений	Возмож- ность значения Null	Роль	При- мер
Код_списка_покупок	Номер списка покупок	Целый	10	'0-9'	Нет	PK	614
Название	Наименование покупки	Символьный	128	'0-9', 'А-я', 'A-z'	Да		Курт- ка
Количество	Количество покупаемых вещей	Символьный	64	'0-9', 'А-я', 'A-z'	Да		1 шт
Сумма	Стоимость покупки	Символьный	64	'0-9', 'А-я', 'A-z'	Да		100 \$
Код_путешествия	Номер путешествия	Целый	10	'0-9'	Нет	FK	1941

Сущность «Списки покупок» предназначена для хранения списка предметов и услуг, которые пользователь планирует приобрести.

Спецификация сущности «Пользователи» приведена в таблице 13.

Таблица 13 – Спецификация сущности «Пользователи»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Код_пользователя	Номер пользователя	Целый	10	'0-9'	Нет	PK	61
Имя	Имя пользователя	Символьный	128	'А-я', 'А-z'	Нет		Александр
Фамилия	Фамилия пользователя	Символьный	128	'А-я', 'А-z'	Нет		Иванов
Почта	Email пользователя	Символьный	128	'0-9', 'А-z', '@', '.'	Нет		alex@gmail.com
Пароль	Пароль пользователя	Символьный	128	'0-9', 'А-z'	Нет		tw21s1r

Сущность «Пользователи» предназначена для хранения информации о пользователях приложения.

В дальнейшем произойдет переход от логической к физической модели базы данных.

2.3 Физическая модель базы данных

На основе логической модели БД была спроектирована физическая модель. Физическая модель отличается от логической тем, что учитывает особенности той СУБД, в которой будет реализована система.

Построенная физическая модель представлена на рисунке 11.

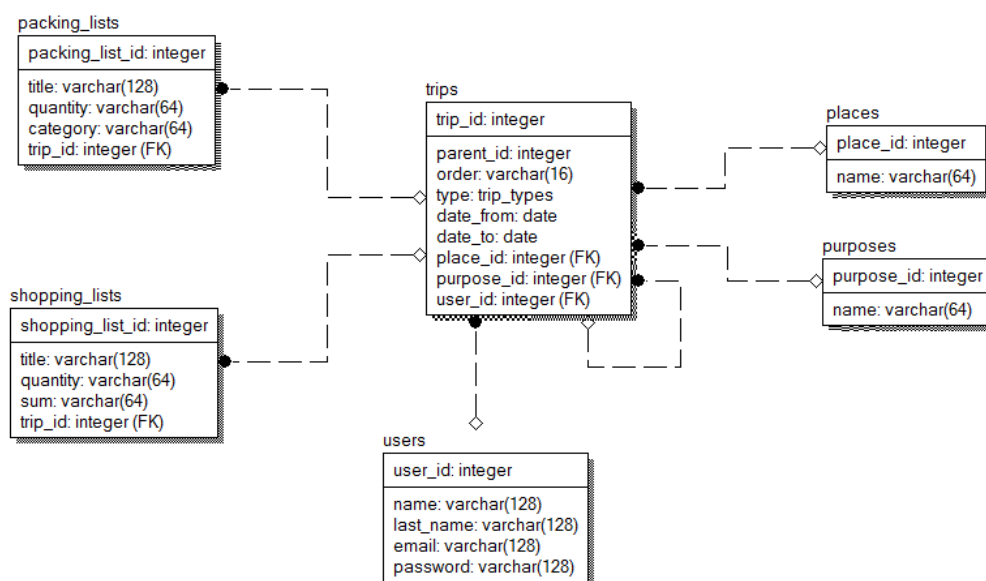


Рисунок 11 – Физическая схема БД

После проектирования, данная модель была реализована в СУБД PostgreSQL. В процессе реализации сущности модели стали таблицами, атрибуты – столбцами, идентификаторы таблиц – первичными ключами, а связи между таблицами были реализованы с помощью вторичных ключей [9].

2.4 Проектирование программных средств

2.4.1 Диаграмма классов

Для приложения планирования путешествий была спроектирована диаграмма классов, представленная на рисунке 12.

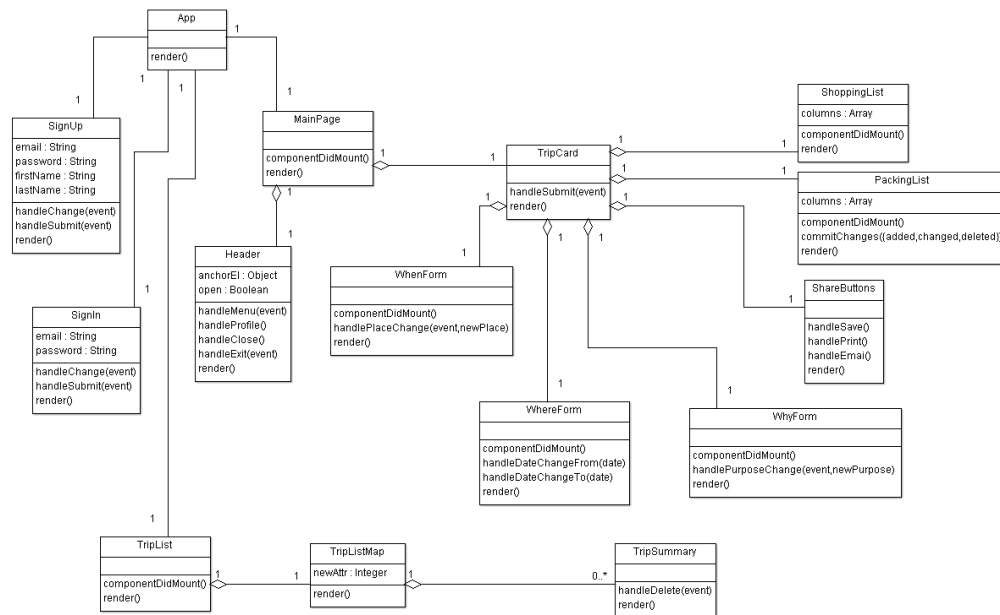


Рисунок 12 – Диаграмма классов приложения планирования путешествий

На диаграмме представлены классы:

- App – класс, содержащий маршрутизацию;
- SignUp – класс страницы регистрации;
- SignIn – класс страницы входа;
- TripList – класс страницы списка поездок;
- TripListMap – класс, реализующий вывод списка поездок;
- TripSummary – класс, реализующий вывод краткой информации о поездке;

- MainPage – класс главной страницы приложения;
- Header – класс «шапки» приложения;
- TripCard – класс, реализующий вывод информации о выбранной поездке;
- WhenForm – класс формы ввода времени поездки;
- WhereForm – класс формы ввода места назначения поездки;
- WhyForm – класс формы ввода цели поездки;
- ShoppingList – класс таблицы списка покупок;
- PackagingList – класс таблицы списка вещей;
- ShareButtons – класс кнопок вывода данных.

2.4.2 Диаграмма состояний

Для отображения варианта использования «Сохранение поездки» была разработана диаграмма последовательности, представленная на рисунке 13.

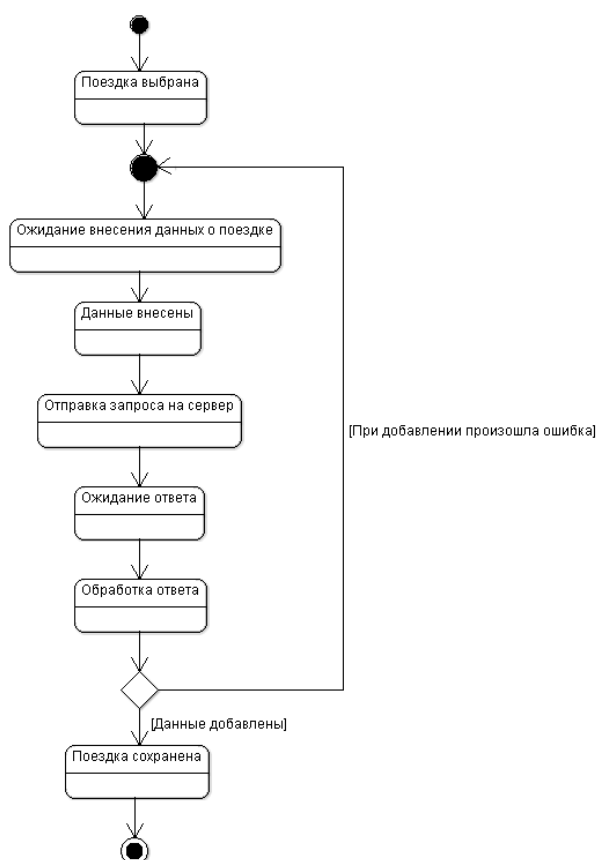


Рисунок 13 – Диаграмма состояний варианта использования «Сохранение поездки»

Диаграмма описывает состояния, в которых бывает система при сохранении поездки. Ее начало совпадает выбором поездки, а конец – с сохранением поездки в БД. Состояние «Поездка выбрана» обозначает, что пользователь выбрал интересующую его поездку. «Ожидание внесения данных о поездке» – состояние обозначающее, что система находится в ожидании того, когда пользователь заполнит данные о планируемой им поездке. «Данные внесены» говорит о том, что пользователь внес в систему данные о поездке. Состояние «Отправка на сервер» обозначает, что клиент отправил данные сохраняемой поездки на сервер, «Ожидание ответа» – клиент ожидает ответа от сервера, о том сохранились ли данные в БД или нет, а «Обработка ответа» означает, что клиенту пришел ответ с сервера и он обрабатывает его результат. Состояние «Поездка сохранена» говорит об успешном добавлении данных в БД.

Пользователь выбирает интересующую его поездку. Далее система попадает в состояние ожидания того, когда пользователь внесет данные о поездке. После внесения этих данных клиент отправляет на сервер запрос на сохранение их в БД. После того как ответ приходит на клиент он обрабатывается. Если добавить данные в базу не удалось пользователю предлагается возможность изменить вносимые данные. При успешном добавлении данных поездка сохраняется.

2.4.3 Диаграмма деятельности

Диаграмма деятельности представляет собой блок-схему, показывающую деятельности и последовательность их выполнения. Для отображения варианта использования «Планирование путешествия» была разработана диаграмма деятельности, представленная на рисунке 14.

Диаграмма описывает, как происходит планирование поездки. Начало диаграммы – подготовка к планированию поездки, конец – готовый план поездки.

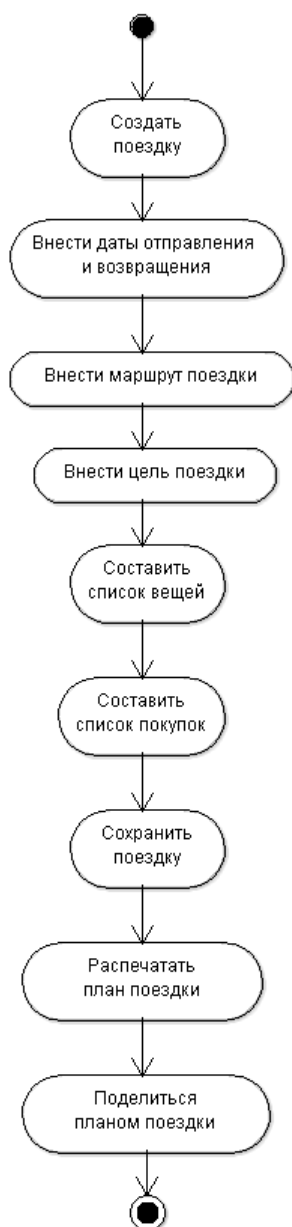


Рисунок 14 – Диаграмма деятельности варианта использования
«Планирование путешествия»

Пользователь создает поездку. Далее он вносит информацию о поездке в приложение: выбирает даты поездки, местоназначение и цель, составляет список вещей, которые нужно взять с собой, а также список покупок. После этого пользователь сохраняет поездку, печатает ее план, а также делится планом поездки в соцсетях.

2.4.4 Диаграмма последовательности

Диаграмма последовательности описывает жизненный цикл объекта, а также взаимодействие действующих лиц в рамках этого жизненного цикла [10]. Для отображения варианта использования «Регистрация» была разработана диаграмма последовательности, представленная на рисунке 15.

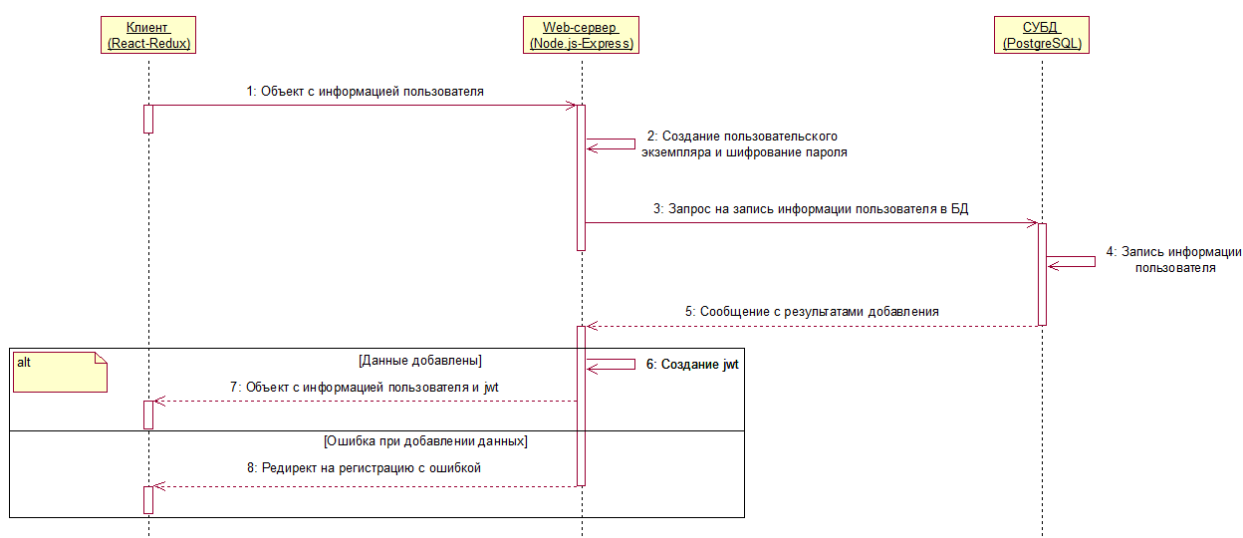


Рисунок 15 – Диаграмма последовательности варианта использования «Регистрация»

Диаграмма описывает процесс регистрации пользователя в приложении [11]. Она включает в себя три объекта: «Клиент (React-Redux)», «Web-сервер (Node.js-Express)» и «СУБД (PostgreSQL)», которые взаимодействуют в рамках сценария и обмениваются сообщениями.

При регистрации, на клиенте создается объект с информацией пользователя и отправляется на Web-сервер. Там создается пользовательский экземпляр и шифруется пароль с использованием bcrypt. После этого на сервере формируется запрос на запись информации пользователя в БД и отправляется СУБД с помощью pg-promise. Там данные пользователя сохраняются в базу и формируется ответ серверу, который отправляется ему в виде promise. Если данные были добавлены, то на сервере создается токен доступа – json web token, который используется для аутентификации в приложении, и объект с информацией пользователя и jwt отправляются на

клиент. В случае, если при добавлении данных возникла ошибка, то происходит редирект на страницу регистрации, а на клиент передается возникшая ошибка.

2.4.5 Диаграмма кооперации

Диаграмма кооперации раскрывает отношения между объектами и их последовательность. Для отображения варианта использования «Добавление поездки» была разработана диаграмма кооперации, представленная на рисунке 16.

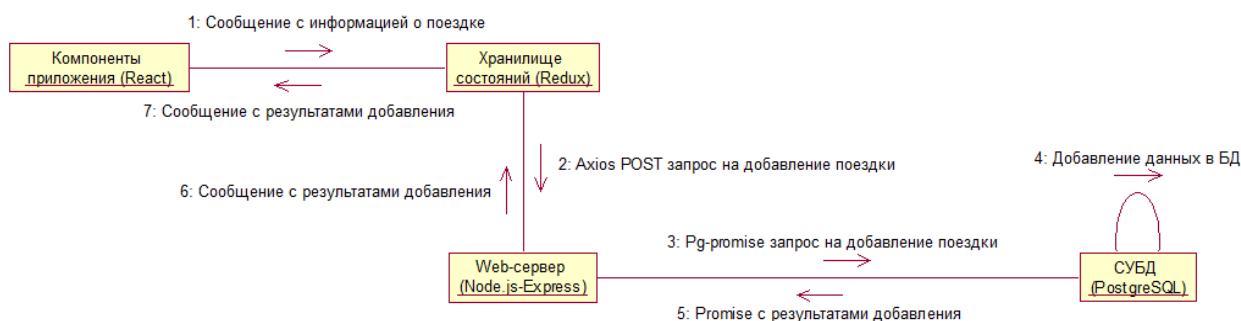


Рисунок 16 – Диаграмма кооперации варианта использования «Добавление поездки»

Диаграмма описывает процесс добавления поездки в базу данных. Она включает в себя четыре объекта: «Компоненты приложения (React)», «Хранилище состояний (Redux)», «Web-сервер (Node.js-Express)» и «СУБД (PostgreSQL)», которые обмениваются сообщениями.

После добавления новой поездки, Компонент приложения через Хранилище состояний с помощью Axios POST запроса, отправляет сообщение с информацией о сохраняемой поездке на Web-сервер. Там это сообщение обрабатывается и, с помощью Pg-promise, формируется запрос к базе данных на сохранение поездки. СУБД сохраняет данные о поездке и формирует ответ, который приходит на Web-сервер в виде Promise. Там формируется сообщение с результатами добавления в БД и через Хранилище состояний отправляется в Компонент приложения.

2.5 Проектирование пользовательского интерфейса

На основании разработанной диаграммы вариантов использования был разработан графический интерфейс веб-приложения планирования путешествий.

Для входа в аккаунт, пользователю необходимо ввести E-mail и пароль от своего аккаунта. Страница входа представлена на рисунке 17.

The image shows a login form titled "Войти" (Login) with a green lock icon. It contains the following elements:

- 1**: Points to the "Почта *" (Email) input field, which contains the text "alexandr.v.mzv@gmail.com".
- 2**: Points to the "Пароль *" (Password) input field, which contains masked characters ".....".
- 3**: Points to the "Запомнить меня" (Remember me) checkbox.
- 4**: Points to the "ВОЙТИ" (Login) button.
- 5**: Points to the link "Нет аккаунта? Зарегистрироваться" (No account? Register).

Рисунок 17 – Страница входа

Основные элементы интерфейса, изображенные на рисунке 17:

- 1 – поле ввода E-mail;
- 2 – поле ввода пароля;
- 3 – флаговая кнопка запоминания пользователя;
- 4 – кнопка входа;
- 5 – кнопка перехода на страницу регистрации.

Если у пользователя нет аккаунта, то он может создать его. Страница регистрации представлена на рисунке 18.

The diagram shows a registration form titled "Зарегистрироваться" (Register) with a green lock icon above it. The form contains the following elements:

- 1**: Input field for "Имя *" (Name), containing "Александр".
- 2**: Input field for "Фамилия *" (Surname), containing "Морозов".
- 3**: Input field for "Почта *" (Email), containing "alexandr.v.mzv@gmail.com".
- 4**: Input field for "Пароль *" (Password), containing ".....".
- 5**: A purple button labeled "ЗАРЕГИСТРИРОВАТЬСЯ" (REGISTER).
- 6**: A link labeled "Уже есть аккаунт? Войти" (Already have an account? Log in).

Рисунок 18 – Страница регистрации

Основные элементы интерфейса, изображенные на рисунке 18:

- 1 – поле ввода имени;
- 2 – поле ввода фамилии;
- 3 – поле ввода E-mail;
- 4 – поле ввода пароля;
- 5 – кнопка регистрации;
- 6 – кнопка перехода на страницу входа.

Не входя в аккаунт, пользователь может воспользоваться базовой версией приложения. Страница планирования путешествия для не авторизованного пользователя представлена на рисунке 19.

Планирование путешествий

ВХОД РЕГИСТРАЦИЯ

Когда
Туда: 21/06/2020
Обратно: 21/06/2020

Куда
Место назначения

Зачем
Тип поездки

Взять с собой

ДОБАВИТЬ	Название	Количество
Нет записей		

Купить

Действия	Название	Количество	Сумма
Нет записей			

5 Записей | 1-0 из 0

8 [Печать] 9 [Email]

Рисунок 19 – Страница поездки для не авторизованного пользователя

Основные элементы интерфейса, изображенные на рисунке 19:

- 1 – поля ввода даты начала и конца поездки;
- 2 – кнопка перехода на страницу входа;
- 3 – кнопка перехода на страницу регистрации;
- 4 – поле ввода места назначения;
- 5 – поле ввода цели поездки;
- 6 – таблица списка вещей;
- 7 – таблица списка покупок;
- 8 – кнопка «Печать»;

9 – кнопка «Поделиться».

Если пользователь вошел в аккаунт, ему предоставляется возможность сохранить свои поездки в приложении. Страница поездки для авторизованного пользователя представлена на рисунке 20.

Планирование путешествий

1: Кнопка «Поделиться» (Share button) in the top right corner.

2: Профильная иконка пользователя (User profile icon) in the top right corner.

3: Место назначения (Destination) dropdown menu, currently showing 'Москва' (Moscow).

4: Тип поездки (Trip type) dropdown menu, currently showing 'Пеший туризм' (Hiking).

5: Секция 'Взять с собой' (Take with you) containing a list of items to take.

6: Секция 'Купить' (Buy) containing a list of items to buy.

7: Кнопка «Сохранить» (Save button) in the bottom bar.

8: Кнопка «Печать» (Print button) in the bottom bar.

9: Кнопка «Поделиться» (Share button) in the bottom bar.

Взять с собой

ДОБАВИТЬ	Название	Количество	
>	Категория: Одежда		
>	Категория: Электроника		
▼	Категория: Документы и деньги		
ИЗМЕНИТЬ	УДАЛИТЬ	Билеты	2 шт
ИЗМЕНИТЬ	УДАЛИТЬ	Паспорт	1 шт

Купить

Действия	Название	Количество	Сумма
	Куртка	1 шт	120 \$
	Наушники	1 шт	75 \$
	Рубашка	3 шт	50 \$
	Вода	3 литра	300 р

5 Записей | 1-4 из 4

Рисунок 20 – Страница поездки для авторизованного пользователя

Основные элементы интерфейса, изображенные на рисунке 20:

1 – поля ввода даты начала и конца поездки;

- 2 – кнопка выхода из аккаунта;
- 3 – поле ввода места назначения;
- 4 – поле ввода цели поездки;
- 5 – таблица списка вещей;
- 6 – таблица списка покупок;
- 7 – кнопка сохранения поездки;
- 8 – кнопка «Печать»;
- 9 – кнопка «Поделиться».

На странице поездки пользователю доступна таблица, которая позволяет записать и категоризировать вещи, необходимые в поездке. Интерфейс, реализующий этот функционал, имеет табличный вид. Форма интерфейса представлена на рисунке 21.

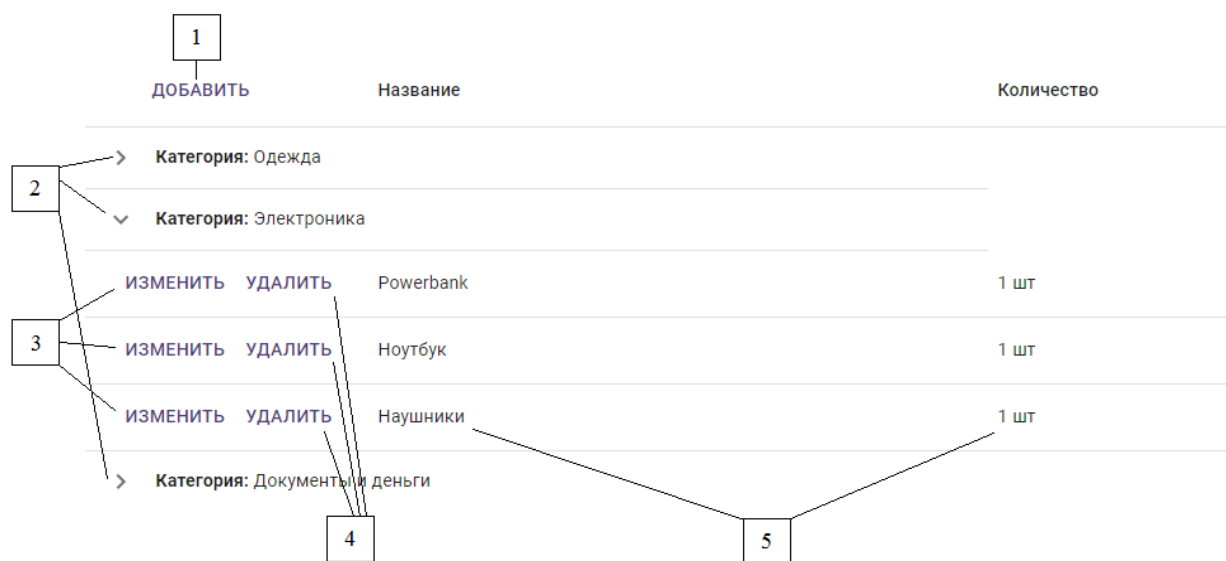


Рисунок 21 – Таблица списка вещей

Основные элементы интерфейса, изображенные на рисунке 21:

- 1 – кнопка добавления записи;
- 2 – категории записей;
- 3 – кнопки удаления записи;
- 4 – кнопки изменения записи;
- 5 – записи.

На странице текущей поездки, пользователь может спланировать покупки, которые он хочет совершить в поездке. Интерфейс, реализующий этот функционал, выполнен в виде таблицы, которая представлена на рисунке 22.

Действия	Название	Количество	Сумма
	Куртка	1 шт	120 \$
	Наушники	1 шт	75 \$
	Рубашка	3 шт	50 \$
	Вода	3 литра	300 р

5 Записей | 1-4 из 4

Рисунок 22 – Таблица списка покупок

Основные элементы интерфейса, изображенные на рисунке 22:

- 1 – окно поиска записей;
- 2 – кнопка добавления записи;
- 3 – кнопки удаления записи;
- 4 – кнопки изменения записи;
- 5 – записи;
- 6 – кнопка выбора количества записей на странице в таблице;
- 7 – кнопки переключения страниц в таблице;

Также, авторизованному пользователю доступна возможность просмотра и редактирования списка его поездок. При добавлении или выборе поездки происходит переход на страницу ввода информации. Страница, отображающая список поездок, представлена на рисунке 23.

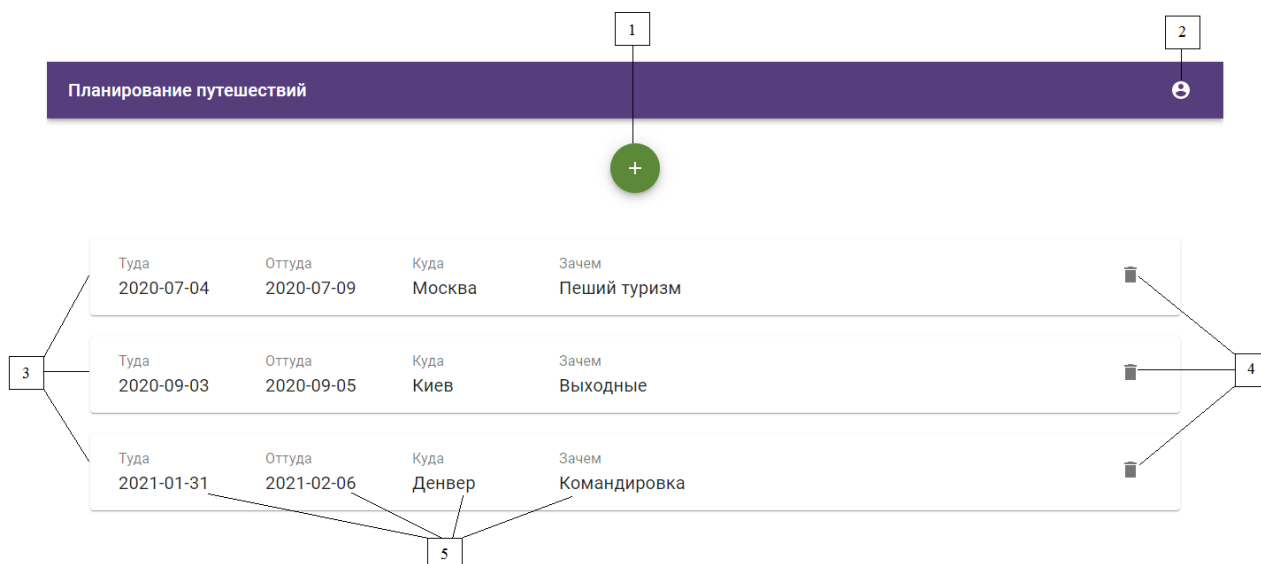


Рисунок 23 – Страница списка поездок

Основные элементы интерфейса, изображенные на рисунке 23:

- 1 – кнопка добавления новой поездки;
- 2 – кнопка выхода из аккаунта;
- 3 – кнопки перехода на поездку;
- 5 – кнопки удаления поездки;
- 4 – краткая информация о поездках.

Таким образом, был спроектирован логичный и простой в использовании интерфейс. В приложении были реализованы такие принципы материального дизайна, как: схожий графический стиль элементов, обратная связь на действия пользователя, и однородность элементов, выполняющих схожие функции.

3 Реализация веб-приложения планирования путешествий

3.1 Диаграмма компонентов

Диаграмма компонентов описывает физическое представление системы, ее архитектуру, а также устанавливает зависимости между программными компонентами. Данная диаграмма для приложения планирования путешествий представлена на рисунке 24.

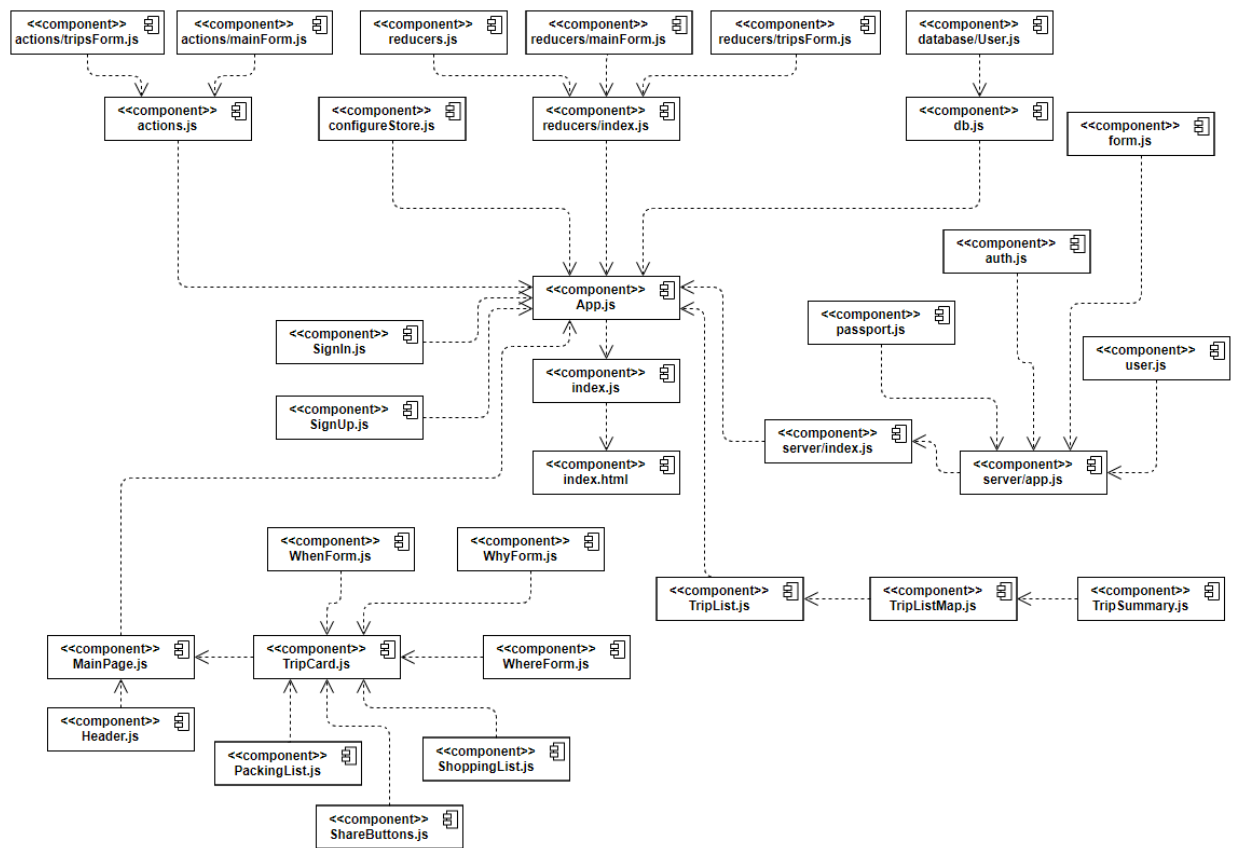


Рисунок 24 – Диаграмма компонентов

Описание компонентов, отображенных на диаграмме:

- actions/tripsForm.js – содержит обработку действий, необходимых для получения краткой информации о поездке;
- actions/mainForm.js – содержит обработку действий, необходимых для получения информации о поездке, а также ее сохранения и удаления;

- actions.js – содержит обработку действий, необходимых для авторизации;
- configureStore.js – реализует хранилище состояний;
- reducers.js – обрабатывает действия необходимые для авторизации и модифицирует хранилище состояний;
- reducers/tripsForm.js – обрабатывает действия необходимые для получения краткой информации о поездке и модифицирует хранилище состояний;
- reducers/mainForm.js – обрабатывает действия необходимые для получения информации о поездке, а также ее сохранения и удаления и модифицирует хранилище состояний;
- reducers/index.js – объединяет обработчики действий;
- database/User.js – реализует функции получения информации о пользователе через pg-promise.js;
- db.js – инициализирует СУБД через pg-promise.js;
- passport.js – определяет стратегии авторизации в приложении;
- auth.js – реализует запросы к СУБД для авторизации пользователя;
- user.js – реализует запросы к СУБД на получение информации о пользователе;
- form.js – реализует запросы к СУБД для получения информации о поездках, а также их добавления и удаления;
- server/app.js – модуль запуска веб-сервера;
- server/index.js – основной модуль веб-сервера;
- SignUp.js – страница регистрации;
- SignIn.js – страница входа;
- TripList.js – страница списка поездок;
- TripListMap.js – реализует вывод списка поездок;

- TripSummary.js – реализует вывод краткой информации о поездке;
- MainPage.js – главная страница приложения;
- Header.js – «шапка» приложения;
- TripCard.js – реализует вывод информации о текущей поездке;
- WhenForm.js – форма ввода времени поездки;
- WhereForm.js – форма ввода места назначения поездки;
- WhyForm.js – форма ввода цели поездки;
- ShoppingList.js – таблица списка покупок;
- PackagingList.js – таблица списка вещей;
- ShareButtons.js – кнопки вывода данных;
- App.js – содержит маршрутизацию приложения;
- index.js – корневой компонент приложения;
- index.html – содержит базовую структуру и точку входа в приложение.

3.2 Диаграмма развертывания

Диаграмма развертывания показывает, как будет выглядеть система при ее развертывании на физических узлах. Диаграмма развёртывания содержит графические изображения устройств, процессов и связей между ними.

Система состоит из ПК пользователя, с помощью которого он осуществляет взаимодействие с ней через web браузер, frontend сервера, написанного на node.js, позволяющего принимать запросы пользователя, отправлять REST запросы на сервер БД, получать ответ и передавать его пользователю, СУБД PostgreSQL, позволяющей принимать запросы frontend сервера, обрабатывать их и возвращать ответ.

Диаграмма развертывания приложения планирования путешествий представлена на рисунке 25.

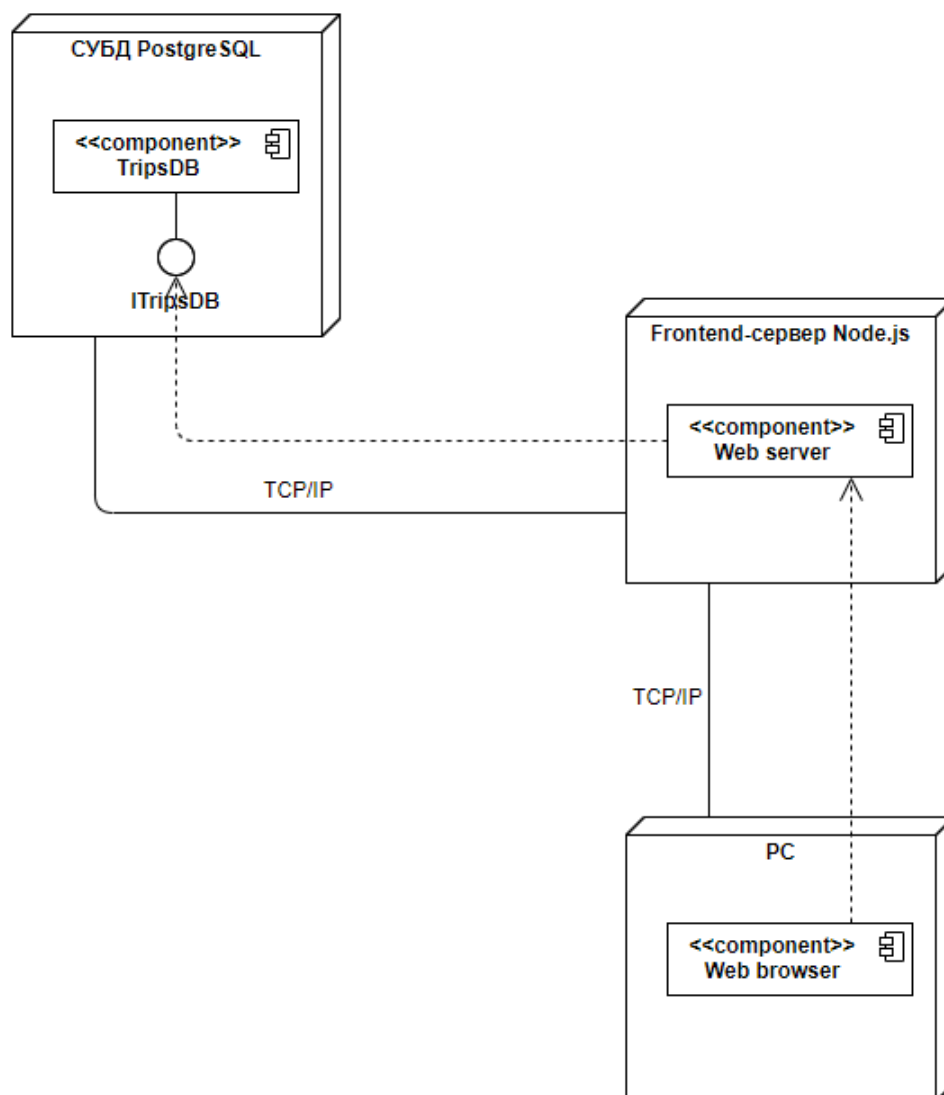


Рисунок 25 – Диаграмма развертывания

Взаимодействие СУБД и Frontend сервера происходит через интерфейс БД ITripsDB. Взаимодействие ПК пользователя с frontend сервером и frontend сервера с сервером БД происходит через TCP/IP соединение.

4 Контроль качества веб-приложения планирования путешествий

Тестирование программного обеспечения – один из этапов жизненного цикла ПО, направленный на проверку поведения программы, относительно требований к ней, на конечном наборе тестов. Далее приведено функциональное и структурное тестирование приложения планирования путешествий.

4.1 Функциональное тестирование

Функциональное тестирование или тестирование методом «чёрного ящика» – это тестирование программного обеспечения, определяющее его способность выполнять задачи, необходимые пользователям, при определенных условиях [12].

Функциональное тестирование может быть рассмотрено на примере прецедента «Вход». В данном прецеденте для успешного входа в приложение нужно ввести корректный E-mail и пароль, которые зарегистрированы в системе.

4.1.1 Определение классов эквивалентности

Разработка тестов методом эквивалентного разбиения осуществляется в два этапа: выделение классов эквивалентности и построение тестов.

Для тестирования поля ввода «E-mail» были выделены следующие классы эквивалентности, представленные в таблице 14.

Таблица 14 – Классы эквивалентности

Поле E-mail	Правильные классы	Неправильные классы
Количество символов в строке	1) [6,40]	2) [0,6),(40,+]
Наличие «@»	3) Да	4) Нет
Латинские буквы	5) [A-Z],[a-z]	6) [A-Я],[a-я]
Специальные символы	7) [-],[_],[.]	8) Все остальные

На основании классов эквивалентности были сформированы возможные ситуации, которые представлены в таблице 15.

Таблица 15 – Функциональное тестирование поля ввода «E-mail»

Входящие значения	Предполагаемый результат	Класс эквивалентности	Результат
alexandr.mzv@gmail.com	Успех	1, 3, 5, 7	Успех (рисунок Д.1)
alexandr.mzv@gmail.com	Ошибка	4	Ошибка (рисунок Д.2)
почта_работа@mail.ru	Ошибка	6	Ошибка (рисунок Д.3)
a@g.c	Ошибка	2	Ошибка (рисунок Д.4)
alexandr-qwerty-qwerty-qwerty-qwerty@gmail.com	Ошибка	2	Ошибка (рисунок Д.5)
alex#andr*mzv@gmail.com	Ошибка	8	Ошибка (рисунок Д.6)

Результаты тестирования приведены в приложении Д.

4.1.2 Анализ граничных условий

Анализ граничных условий может позволить найти ошибки, находящиеся на границах классов эквивалентности, а также может включать тесты на данных, выходящих за допустимый диапазон значений. В качестве исходного поля для тестирования было выбрано поле ввода «E-mail». Для него были выделены следующие границы:

- физическая граница;
- произвольная граница;
- логическая граница.

Физическая граница – количество введенных символов равно нулю.

Произвольная граница – количество введенных символов должно быть в промежутке от 6 до 40 включительно.

Логическая граница – введенное значение должно состоять из латинских букв, цифр, одного символа «@», а также может включать такие специальные символы, как «-», «_» и «.».

Произвольная граница представлена на рисунке 26.



Рисунок 26 – Произвольная граница ввода поля «E-mail»

Логическая граница представлена на рисунке 27.

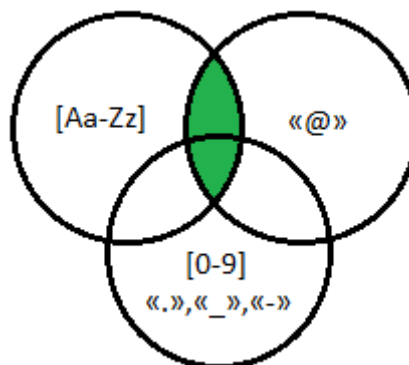


Рисунок 27 – Логическая граница ввода поля «E-mail»

Тестовые наборы приведены в таблице 16.

Таблица 16 – Граничные условия

Граничное условие	Значение	Результат
Пустая строка	[]	Ошибка (рисунок Д.7)
Количество символов в строке меньше 6	a@g.c	Ошибка (рисунок Д.8)
Количество символов в строке равно 6	a@g.ru	Успех (рисунок Д.9)
Количество символов в строке равно 40	alexander.v.morozov.06.09.1998@gmail.com	Успех (рисунок Д.10)
Количество символов в строке больше 40	alexander.valentinovich.morozov.06.09.1998@gmail.com	Ошибка (рисунок Д.11)
Строка содержит только допустимые символы	alexandr.mzv@gmail.com	Успех (рисунок Д.12)
Строка содержит недопустимые символы	alex//andr**mzv@gmail.com	Ошибка (рисунок Д.13)
Строка не содержит «@»	alexandr.mzv@gmail.com	Успех (рисунок Д.14)
Строка содержит русские буквы	почта@mail.ru	Ошибка (рисунок Д.15)

Результаты тестирования приведены в приложении Д.

4.1.3 Построение функциональной диаграммы

Метод функциональных диаграмм позволяет выбирать высоко результативные тесты.

Для тестирования были выбраны поля ввода «Email» и «Password».

Составляющие функциональной диаграммы:

- на сайт зашли;
- нет соединения с сервером;
- нет интернета;
- вход;
- ошибка соединения.

Функциональная диаграмма для выбранных классов эквивалентности представлена на рисунке 28.

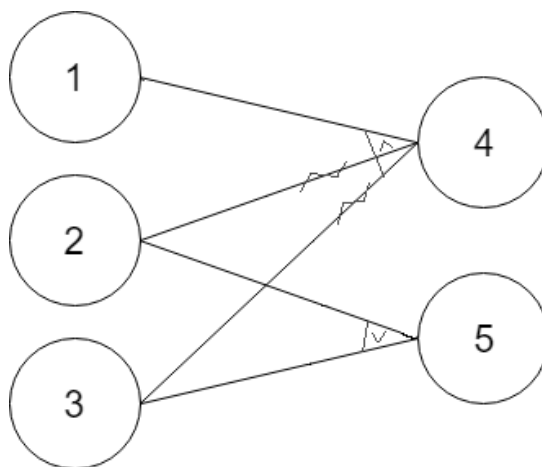


Рисунок 28 – Функциональная диаграмма

Причины и результаты тестов представлены в таблице 17.

Таблица 17 – Причины и тесты

Причины/тесты	1	2	3	4
1	0	0	0	1
2	0	1	1	0

Продолжение таблицы 17

Причины/тесты	1	2	3	4
3	1	0	1	0
4	0	0	0	1
5	1	1	1	0

Результатом построения функциональной диаграммы является то, что при отсутствии ошибок соединения и правильно введенных данных, приложение работает корректно, иначе появляется сообщение об ошибке.

4.1.4 Парное тестирование

Парное тестирование – это техника формирования наборов тестовых данных из полного набора входных данных в системе, которая позволяет существенно сократить количество тест-кейсов.

Для проведения парного тестирования были выбраны поле ввода «Email». Для парного тестирования было использовано программное обеспечение «allpairs». Запуск производится из консоли. Описание входных классов эквивалентности представлено в таблице 18.

Таблица 18 – Описание классов эквивалентности

Name	Lastname	Email	Password
goodName	goodLastname	goodEmail	goodPassword
badName	badLastname	badEmail	badPassword
nullName	nullLastname	[1,6)	[1,6)
otherName	otherLastname	[6,40]	[6,40]
defisName	defisLastname	(40,+]	(40,+]

Результаты создания тестовых случаев представлены на рисунке 29.

case	name	lastname	email	password	pairings
1	goodName	goodLastname	goodEmail	goodPassword	6
2	goodName	badLastname	badEmail	badPassword	6
3	goodName	nullLastname	[1,6)	[1,6)	6
4	goodName	otherLastname	[6,40]	[6,40]	6
5	goodName	defisLastname	(40,+]	(40,+]	6
6	badName	goodLastname	badEmail	[1,6)	6
7	badName	badLastname	goodEmail	[6,40]	6
8	badName	nullLastname	[6,40]	goodPassword	6
9	badName	otherLastname	[1,6)	badPassword	6
10	badName	defisLastname	goodEmail	badPassword	4
11	nullName	goodLastname	[1,6)	[6,40]	6
12	nullName	badLastname	[6,40]	[1,6)	6
13	nullName	nullLastname	goodEmail	(40,+]	6
14	nullName	otherLastname	badEmail	goodPassword	6
15	nullName	defisLastname	badEmail	[6,40]	4
16	otherName	goodLastname	[6,40]	badPassword	6
17	otherName	badLastname	[1,6)	goodPassword	6
18	otherName	nullLastname	badEmail	[6,40]	5
19	otherName	otherLastname	goodEmail	[1,6)	6
20	otherName	defisLastname	[1,6)	(40,+]	4
21	defisName	goodLastname	(40,+]	goodPassword	5
22	defisName	badLastname	badEmail	(40,+]	5
23	defisName	nullLastname	[6,40]	badPassword	4
24	defisName	otherLastname	(40,+]	[1,6)	4
25	defisName	defisLastname	[6,40]	goodPassword	3
26	badName	goodLastname	(40,+]	(40,+]	3
27	nullName	badLastname	(40,+]	badPassword	4
28	defisName	nullLastname	(40,+]	[6,40]	3
29	otherName	otherLastname	[6,40]	(40,+]	2
30	otherName	defisLastname	(40,+]	[1,6)	2
31	defisName	~goodLastname	goodEmail	~badPassword	1

Рисунок 29 – Результаты создания тестовых случаев

В результате функционального тестирования было выявлено, что при авторизации пользователя в приложении и при работе в не авторизованном режиме, ошибок не возникает.

4.2 Структурное тестирование

Структурное тестирование (тестирование по принципу «белого ящика») характеризуется степенью, в которой тесты покрывают логику (исходный текст) программы. Данная методология тестирования предполагает выполнение каждого пути в программе [13].

В данной работе тестирование по принципу белого ящика включает в себя метод покрытия операторов, метод покрытия условий и метод покрытия маршрутов.

Тестирование может быть проведено на примере прецедента «Авторизация». Его блок-схема представлена на рисунке 30.

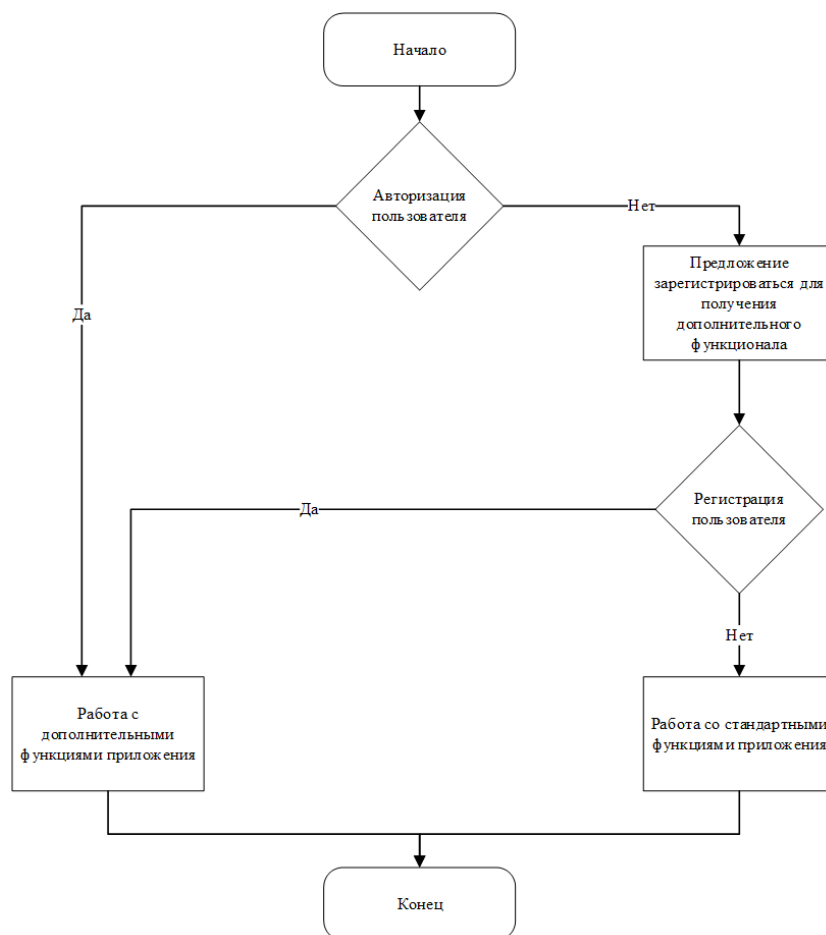


Рисунок 30 – Блок-схема прецедента «Авторизация»

На основании блок-схемы был построен функциональный граф алгоритма, который представлен на рисунке 31.

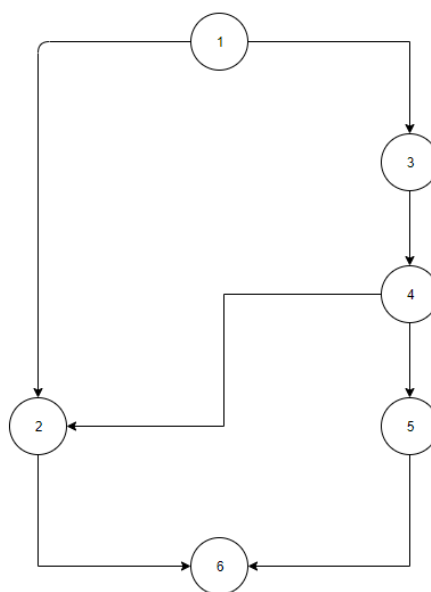


Рисунок 31 – Функциональный граф прецедента «Авторизация»

Вершины функционального графа представляют из себя операторы тестируемой программы, а дуги – показывают, как передается управление между ними.

4.2.1 Метод покрытия операторов

Критерии покрытия операторов подразумевает выполнение каждого оператора программы по крайней мере один раз.

Для выполнения каждого оператора было составлено два теста. Результаты приведены в таблице 19.

Таблица 19 – Результат тестирования методом покрытия операторов

Операторы	Тест	Фактический результат	Результат тестирования
1, 2, 6	Пользователь зарегистрирован и вошёл в свой аккаунт.	Пользователь получает доступ к дополнительным функциям приложения.	Успешно.
1, 3, 4, 5, 6	Пользователь не зарегистрирован и не регистрируется.	Пользователь получает доступ к стандартным функциям приложения.	Успешно.

Тестирование методом покрытия операторов показало, что при входе пользователя в аккаунт, а также при использовании приложения без входа в аккаунт, ошибок не возникает.

4.2.2 Метод покрытия условий

Суть метода покрытия условий заключается в том, чтобы составить число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз. Также должно соблюдаться условие выполнения каждого оператора по крайней мере один раз.

Для выполнения каждого условия было составлено три теста. Результаты приведены в таблице 20.

Таблица 20 – Результат тестирования методом покрытия условий.

Условия	Тест	Фактический результат	Результат тестирования
1 – да	Пользователь зарегистрирован	Пользователь получает доступ к дополнительным функциям приложения.	Успешно.
1 – нет, 2 – да	Пользователь не зарегистрирован, но регистрируется	Пользователь получает доступ к дополнительным функциям приложения.	Успешно.
1 – нет, 2 – нет	Пользователь не зарегистрирован, и не регистрируется	Пользователь получает доступ к стандартным функциям приложения.	Успешно.

Тестирование методом покрытия условий показало, что при регистрации пользователя и при использовании приложения, не входя в аккаунт, ошибок не возникает.

4.2.3 Метод покрытия маршрутов

Метод покрытия маршрутов является наиболее полным и результативным. Он предполагает прохождение по каждому возможному пути программы.

Для выполнения всех маршрутов было составлено три теста. Результаты приведены в таблице 21.

Таблица 21 – Результат тестирования методом покрытия условий

Маршрут	Тест	Фактический результат	Результат тестирования
1, 2, 6	Пользователь зарегистрирован и вошёл в свой аккаунт.	Пользователь получает доступ к дополнительным функциям приложения.	Успешно.
1, 3, 2, 6	Пользователь не зарегистрирован, но регистрируется	Пользователь получает доступ к дополнительным функциям приложения.	Успешно.
1, 3, 4, 5, 6	Пользователь не зарегистрирован и не регистрируется.	Пользователь получает доступ к стандартным функциям приложения.	Успешно.

Тестирование методом покрытия маршрутов показало, что при регистрации пользователя, входе, а также при использовании приложения, не входя в аккаунт, ошибок не возникает.

4.3 Метрики программного кода проекта

Определение качества кода играет большую роль при создании приложения. Одним из основных критериев качества кода является его сложность, так как от нее во многом зависит количество времени и сил, которые необходимо будет затратить на разработку.

В оценке качества программы могут помочь использование метрик кода. С их помощью можно оценить сложность кода разрабатываемого приложения, объем работ, для его реализации, а также стилистику написания кода программы.

Для оценки качества кода разрабатываемого приложения, была выбрана метрика Холстеда. Основу этой метрики составляют четыре характеристики:

- число уникальных операторов n_1 ;
- число уникальных операндов n_2 ;
- общее число операторов N_1 ;
- общее число операндов N_2 .

С помощью метрики Холстеда проанализируем качество кода для варианта использования «Сохранение». Операнды и операторы приведены в таблице 22.

Таблица 22 – Операнды и операторы

Операторы	Число вхождений	Операнды	Число вхождений
let	4	user_id	7
=	24	trip	9
:	13	shoppingList	3
...	2	packingList	2

Продолжение таблицы 22

Операторы	Число вхождений	Операнды	Число вхождений
return	2	req	3
=>	4	res	3
map	2	sl_id	1
stringify	3	pl_id	1
toString	3	id	2
json	4	date_from	7
one	1	date_to	7
{}	5	place	7
.	58	purpose	7
declare	1	shopping_list	8
begin	1	packing_list	7
end	1	error	2
if	2	msg	1
end if	2		
elseif	1		
+	4		
insert	3		
update	3		
delete	3		
select	15		
count	6		
>	3		
>=	1		
-	2		
n1 = 28	N1 = 173	n2 = 17	N2 = 77

На основе данной таблицы рассчитаем метрики кода по Холстеду.

Длина кода программы рассчитывается по формуле:

$$L = N1 + N2 = 173 + 77 = 250.$$

Словарь программы рассчитывается по формуле:

$$V = n_1 + n_2 = 28 + 17 = 45.$$

Объем кода вычисляется по формуле:

$$P = L \cdot \log(2)V = 63,41.$$

Сложность кода можно рассчитать по формуле:

$$D = (n_1 / 2) \cdot (N_2 / n_2) = (28 / 2) \cdot (77 / 17) = 413,3.$$

Результаты вычисленных метрик показывают, что сложность поддержки кода, для заданного варианта использования, находится на среднем уровне.

4.4 Планирование и оценка бюджета проекта

Управление проектами – это деятельность, в ходе которой достигаются поставленные задачи и цели проекта, с учетом различных рисков, ограничений на ресурсы, качества, трудозатрат. Ключевым фактором в организации эффективного управления проектом является наличие заранее определенного плана и минимизация отклонений от него в процессе управления.

В качестве программного обеспечения управления проектом был выбран Microsoft Project 2013, который позволяет упростить и сделать более удобным планирование проекта, а также визуализировать прогресс разработки на диаграмме Ганта [14]. Перед началом планирования проекта необходимо определить список задач. Названия, виды, связи, длительность и предшественники задач проекта приведены в таблице 23.

Таблица 23 – Задачи проекта

№	Название	Вид Задачи	Предшественники	Длительность
1	Начало реализации проекта	Веха		-
2	Анализ предметной области	Задача		1
3	Анализ программных аналогов	Задача	2	2

Продолжение таблицы 23

№	Название	Вид Задачи	Предшественники	Длительность
4	Постановка задачи	Фаза	3	-
5	Анализ требований	Задача	3	1
6	Анализ бизнес процессов	Задача	5	1
7	Выбор средств разработки	Задача	6	1
8	Постановка задачи завершена	Веха	7	-
9	Проектирование	Фаза	4	-
10	Проектирование архитектуры данных	Задача	8	3
11	Проектирование базы данных	Задача	10	2
12	Проектирование программных средств	Задача	11	3
13	Проектирование пользовательского интерфейса	Задача	12	3
14	Проектирование завершено	Веха	13	-
15	Реализация	Фаза	9	-
16	Реализация базы данных	Задача	14	3
17	Реализация пользовательского интерфейса	Задача	16	4
18	Реализация программных средств	Задача	17	7
19	Реализация завершена	Веха	18	-
20	Отладка и тестирование	Фаза	15	-
21	Отладка серверной части	Задача	19	1
22	Отладка клиентской части	Задача	21	2
23	Функциональное тестирование	Задача	22	1
24	Структурное тестирование	Задача	23	1
25	Рефакторинг	Задача	24	3
26	Анализ качества кода	Задача	25	2
27	Отладка и тестирование завершены	Веха	26	-

Окончание таблицы 23

№	Название	Вид Задачи	Предшественники	Длительность
28	Завершение проекта	Фаза	20	-
29	Составление программной документации	Задача	27	2
30	Сдача в эксплуатацию	Задача	29	2
31	Проект завершен	Веха	30	-
32	Конец проекта	Веха	31	-

После определения задач проекта необходимо определить ресурсы, их типы и соответствующие им затраты. Перечень ресурсов проекта приведен в таблице 24.

Таблица 24 – Ресурсы и затраты проекта

№	Название ресурса	Тип ресурса	Затраты		Начисление
			Стандартная ставка	Ставка сверхур.	
1	Программист	Трудовой	20 000 р./мес.	300 р./ч.	Пропорциональное
2	Аналитик	Трудовой	12 500 р./мес.	175 р./ч.	Пропорциональное
3	Тестировщик	Трудовой	15 000 р./мес.	200 р./ч.	Пропорциональное
4	Бумага	Материальный	250 р.		Пропорциональное
5	Флэш-карта	Материальный	500 р.		Пропорциональное
6	Интернет	Затраты			Пропорциональное

Далее выделенные ресурсы должны быть соотнесены с задачами проекта. Распределение ресурсов приведено в таблице 25.

Таблица 25 – Распределение ресурсов проекта по задачам

№ п/п	Название	Ресурс	Единицы (затраты)
1	Начало реализации проекта	Бумага	1 уп.
2	Анализ предметной области	Аналитик Интернет	100 % 200 р.

Продолжение таблицы 25

№ п/п	Название	Ресурс	Единицы (затраты)
3	Анализ программных аналогов	Программист Аналитик Интернет	100 % 100 % 300 р.
4	Анализ требований	Программист Аналитик	50 % 50 %
5	Анализ бизнес процессов	Программист Аналитик	70 % 20 %
6	Выбор средств разработки	Программист Интернет	100 % 200 р.
7	Проектирование архитектуры данных	Программист	100 %
8	Проектирование базы данных	Программист	80 %
9	Проектирование программных средств	Программист	100 %
10	Проектирование пользовательского интерфейса	Программист	70 %
11	Реализация базы данных	Программист	70%
12	Реализация пользовательского интерфейса	Программист	100 %
13	Реализация программных средств	Программист	100 %
14	Отладка серверной части	Программист Тестировщик	70 % 70 %
15	Отладка клиентской части	Программист Тестировщик	70 % 70 %
16	Функциональное тестирование	Тестировщик	100 %
17	Структурное тестирование	Тестировщик	100 %
18	Рефакторинг	Тестировщик	100 %
19	Анализ качества кода	Тестировщик Программист	100 % 30 %
20	Составление программной документации	Программист Аналитик Тестировщик Бумага Флэш-карта Интернет	100 % 100 % 100 % 1 уп. 1 шт. 250 р.
21	Сдача в эксплуатацию	Программист Аналитик Тестировщик	100 % 100 % 100 %

В Microsoft Office Project для планирования процесса разработки программного обеспечения был создан проект, дата начала которого 13.04.20.

В него были внесены выделенные задачи, ресурсы и затраты. Результат добавления ресурсов в проект приведен на рисунке 32.

		Название ресурса ▾	Тип ▾	Краткое название ▾	Макс. единиц ▾	Стандартная ставка ▾	Ставка сверхурочн ▾	Начисление ▾	Базовый календарь ▾
1		Программист	Трудовой	П	100%	125,00 Р/ч	250,00 Р/ч	Пропорциональ	Стандартный
2		Аналитик	Трудовой	А	100%	85,00 Р/ч	170,00 Р/ч	Пропорциональ	Стандартный
3		Тестировщик	Трудовой	Т	100%	100,00 Р/ч	200,00 Р/ч	Пропорциональ	Стандартный
4		Бумага	Материальный	Б		250,00 Р		Пропорциональ	
5		Флэш-карта	Материальный	Ф		500,00 Р		Пропорциональ	
6		Интернет	Затраты	И				Пропорциональ	

Рисунок 32 – Лист ресурсов

После внесения данных в проект, Microsoft Project построит диаграмму Ганта, которая является визуальным представлением календарного плана. Диаграмма Ганта проекта приведена на рисунке 33.

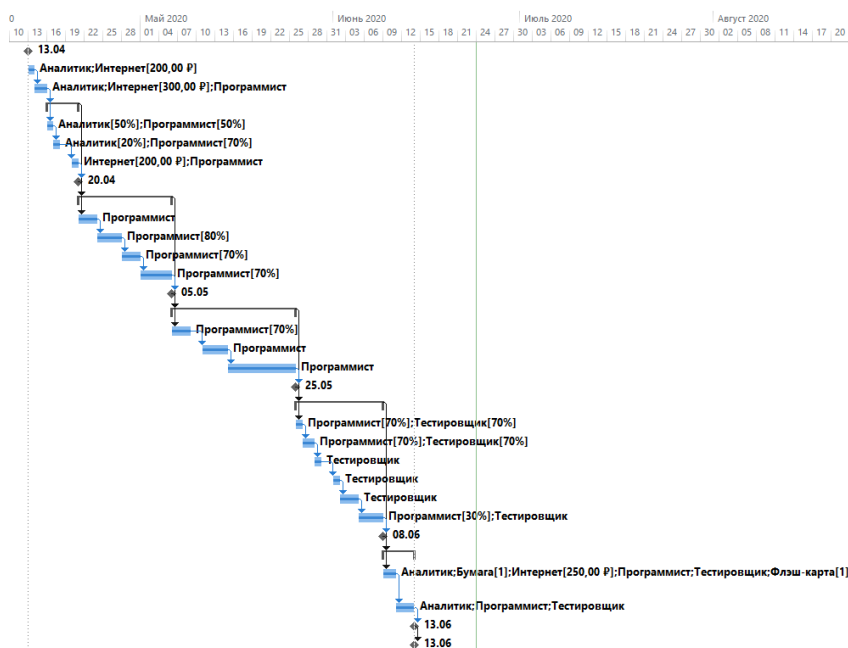


Рисунок 33 – Диаграмма Ганта разработанного проекта

В Microsoft Project был получен обзорный отчет проекта планирования разработки приложения планирования путешествий, представленный на рисунке 34. В данном отчете приведены данные о дате начала и окончания проекта, его длительности, проценте завершения проекта и проценте завершения всех задач верхнего уровня.

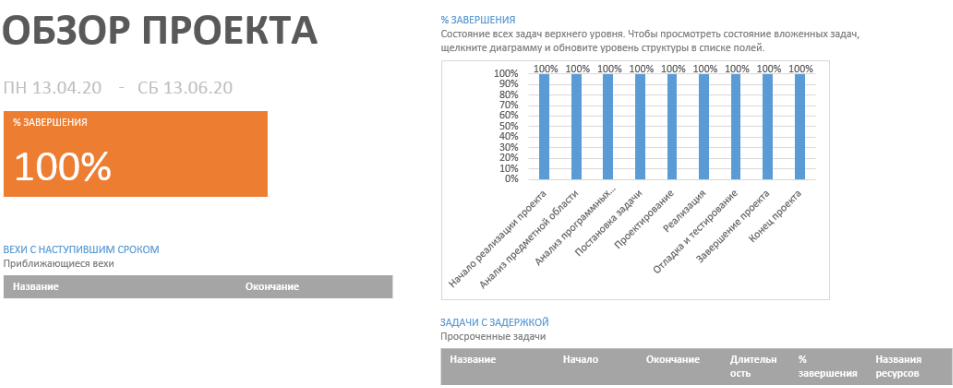


Рисунок 34 – Обзорный отчет проекта

Также с помощью Microsoft Project можно построить отчет по движению денежных средств. Отчет по затратам представлен на рисунке 35.

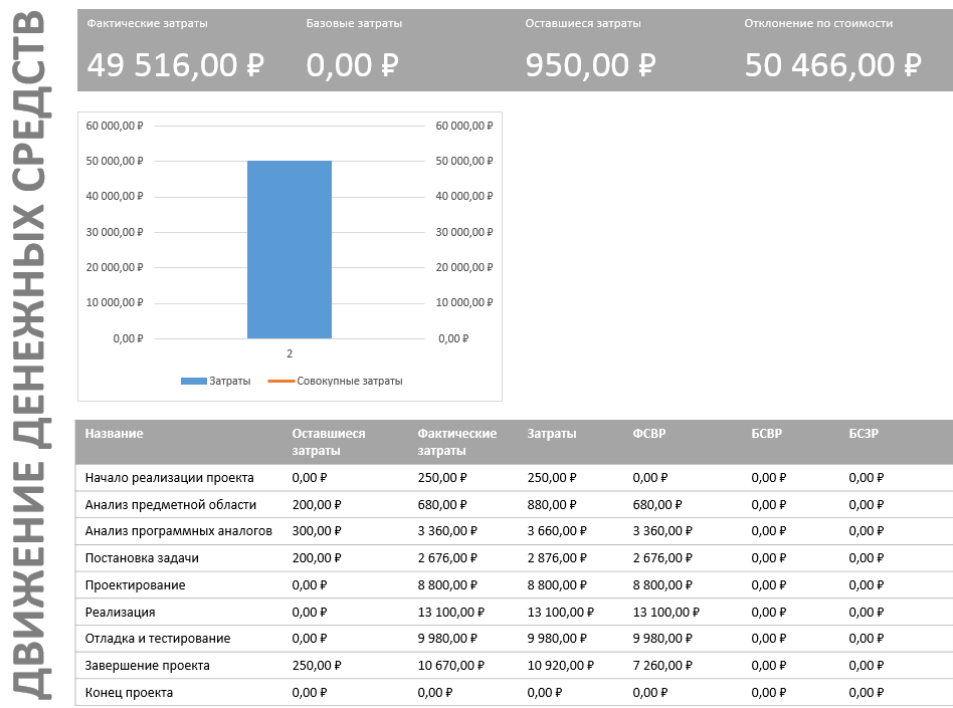


Рисунок 35 – Отчет по движению денежных средств

По результатам отчетов видно, что срок выполнения проекта составил 45 дней, а бюджет проекта – 50 466 рублей. При помощи оценки сроков и бюджета проекта можно сделать вывод о том, что процесс разработки приложения является экономически выгодным.

Заключение

Разработано веб-приложение планирования путешествий, имеющее клиент-серверную архитектуру. Клиентская часть была реализована на языке JavaScript с использованием фреймворка, серверная часть – на платформе Node.js с использованием фреймворка Express.js, в качестве СУБД использовался PostgreSQL.

Приложение обладает необходимым для планирования поездки функционалом. В отличие от известных аналогов оно объединяет в себе самые важные для путешественника функции в одном приложении, также дает возможность создать список покупок и поддерживает русский язык. Кроме этого, с помощью приложения можно составить маршрут поездки на выбранные даты, составить список вещей, , поделиться планом поездки в соцсетях. Для приложения спроектирован простой и понятный интерфейс, удобный в использовании как с персонального компьютера, так и с мобильного устройства.

Проведено функциональное и структурное тестирование, а также определены метрики программного кода проекта. По результатам проведенного тестирования можно сделать вывод, что приложение функционирует корректно.

В ходе планирования и оценки бюджета разработки в Microsoft Project были добавлены основные задачи, необходимые для реализации проекта. Была построена диаграмма Ганта и получена статистика проекта, по которой можно сделать вывод, что длительность реализации проекта составляет 45 дней, все поставленные задачи решены, и проект завершен на 100%. Затраты составляют 50 466 руб., что говорит об экономической выгодности данного проекта.

Таким образом, в результате выполнения выпускной квалификационной работы разработано веб-приложение планирования путешествий, отвечающее всем требованиям технического задания.

Список использованных источников

1. Ефремова, М. В. Основы технологии туристского бизнеса: учебное пособие / М. В. Ефремова. – М.: Издательство «Ось-89», 2005. – 191 с.
2. Чудновский, А. Д. Управление потребительскими предпочтениями в сфере отечественного туризма и гостеприимства, и основные направления реализации туристского продукта: учебное пособие / А. Д. Чудновский, М. А. Жукова. – М.: Федеральное агентство по туризму, 2014. – 304 с.
3. Арлоу, Д. UML 2 и Унифицированный процесс. Второе издание / Д. Арлоу, А. Нейштадт. – СПб.: Символ-Плюс, 2007. – 624 с.
4. Документация к Postgres Pro Standard 9.6 [Электронный ресурс]. – URL: <https://postgrespro.ru/docs/postgrespro/9.6/index> (дата обращения 12.04.2020).
5. Браун, И. Веб-разработка с применением Node и Express / И. Браун. – СПб.: Питер, 2017. – 336 с.
6. Стефанов, С. React.js. Быстрый старт / С. Стефанов. – СПб.: Питер, 2017. – 304 с.
7. Javascript-фреймворки: тенденции 2019 года [Электронный ресурс]. – URL: <https://habr.com/ru/company/plarium/blog/433926> (дата обращения 13.02.2020).
8. Горбаченко, В. И. Проектирование информационных систем с СА ERwin Modeling Suite 7.3: учебное пособие / В. И. Горбаченко, Г. Ф. Убиенных, Г. В. Бобрышева – Пенза: Изд-во ПГУ, 2012. – 154 с.
9. Стоунз, Р. PostgreSQL. Основы / Р. Стоунз, Н. Мэттью. – М.: Символ-Плюс, 2002. – 640 с.
10. Леоненков, А. В. Самоучитель UML 2 / А. В. Леоненков – Издательство: БХВ-Петербург, 2007. – 576 с.
11. Аутентификация в приложении с использованием jwt. [Электронный ресурс]. – URL: <https://github.com/auth0/node-jsonwebtoken> (дата обращения: 13.02.2020).

12. Тамре, Л. Введение в тестирование программного обеспечения / Л. Тамре. – Издательство: Вильямс, 2003. – 368 с.
13. Майерс, Г. Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Сандлер – Издательство: Вильямс, 2012. – 272 с.
14. Куперштейн, В. И. Microsoft Project в управлении проектами. / В. И. Куперштейн, А. В. Цветкова. – СПб.: БХВ-Петербург, 2011. – 415 с.

Приложение А
ГЛОССАРИЙ
(справочное)

База данных (БД) – совокупность данных, организованных в соответствии с определенными правилами, хранящаяся в памяти компьютера, характеризующая состояние предметной области.

Система управления базами данных (СУБД) – совокупность программных средств, позволяющих создавать и использовать базы данных.

UML (Unified Modeling Language) – язык моделирования, предназначенный для проектирования систем, объектно-ориентированного анализа и моделирования бизнес-процессов.

Объектно-ориентированное программирование (ООП) – метод программирования, в основе которого лежит представление программы в виде группы объектов, каждый из которых является экземпляром определенного класса.

Программное обеспечение (ПО) – совокупность программ, обеспечивающих работоспособность компьютера и позволяющий решать с его помощью различные задачи.

Приложение Б
ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ НА ЯЗЫКЕ
JAVASCRIPT
(рекомендуемое)

Листинг Б.1 – Код модуля actions/mainForm.js на языке JavaScript

```

export const tripDataSend = tripInfo => {
  return () => {
    axios.post("/api/trips/send", tripInfo)
      .then((response) => {
        if (response.status !== 200) {
          throw Error(response.statusText);
        }
      })
      .catch(() => console.log("tripDataSend error"));
  }
};

export const tripDataGet = user => {
  return dispatch => {
    axios.get('/api/trips/get_trips_summary', user)
      .then((response) => {
        if (response.status !== 200) {
          throw Error(response.statusText);
        }
        dispatch(tripsSummary(response.data.trips));
      })
      .catch(() => console.log("tripDataGet error"));
  }
};

export const currentTripDataGet = trip_id => {
  return dispatch => {
    dispatch(dataIsLoading(true));
    axios.get(`/api/trips/get_current_trip?trip_id=${trip_id}`)
      .then((response) => {
        if (response.status !== 200) {
          throw Error(response.statusText);
        }
        if (response.data.trip.o_packing_list) {
          response.data.trip.o_packing_list.forEach((item, index) => {
            item.id = index;
          });
        }
        dispatch(currentTripData(response));
        dispatch(dataIsLoading(false));
      })
      .catch((err) => console.log(err));
  }
};

export const tripDataDelete = trip_id => {
  return () => {
    axios.post(`/api/trips/delete?trip_id=${trip_id}`)
      .then((response) => {
        if (response.status !== 200) {
          throw Error(response.statusText);
        }
      })
      .catch(() => console.log("tripDataDelete error"));
  }
};

export const onLogout = () => {
  return dispatch => {
    dispatch({ type: 'USER_LOGGED_OUT' });
  };
};

```

Листинг Б.2 – Код модуля actions.js на языке JavaScript

```
export function actions(user) {
  return (dispatch) => {
    axios.post("/api/auth/register", {user})
      .then((response) => {
        if (response.status !== 200) {
          throw Error(response.statusText);
        }
        dispatch(token(response.data.token));
      })
      .catch(() => console.log("actions error"));
  };
}
export const userLoginFetch = user => {
  return dispatch => {
    axios.post("/api/auth/", {email: user.email, password: user.password })
      .then((response) => {
        if (response.status !== 200) {
          throw Error(response.statusText);
        }
        dispatch(token(response.data.token));
      })
      .catch(() => console.log("loginFetch error"));
  };
}
```

Листинг Б.3 – Код модуля reducers/mainForm.js на языке JavaScript

```
let initialState = {
  dateFrom: new Date(),
  dateTo: new Date(),
};
export function tripId(state = 0 , action) {
  if (action.type === 'TRIP_ID')
    return { ...state, id: action.id };
  return state;
}
function tripDates(state = initialState, action) {
  switch(action.type) {
    case 'TRIP_DATE_FROM':
      return { ...state, dateFrom: action.dateFrom };
    case 'TRIP_DATE_TO':
      return { ...state, dateTo: action.dateTo };
    default:
      return state;
  }
}
export function tripPlace(state = { place: { label: '' } }, action) {
  if (action.type === 'TRIP_PLACE')
    return { ...state, place: action.place };
  return state;
}
export function tripPurpose(state = { purpose: { label: '' } }, action) {
  if (action.type === 'TRIP_PURPOSE')
    return { ...state, purpose: action.purpose };
  return state;
}
function tripShoppingList(state = [ ], action) {
  switch(action.type) {
    case 'ADD_SHOPPING_LIST':
      return [...state, action.newData];
    case 'UPDATE_SHOPPING_LIST':
```

```

        let index = state.indexOf(action.oldData);
        return _.fill(_.slice(state), action.newData, index, index + 1);
    case 'DELETE_SHOPPING_LIST':
        return _.difference(state, [action.oldData]);
    case 'SET_SHOPPING_LIST':
        return action.data.map((item) => {
            return { ...item };
        });
    default:
        return state;
    }
}
function tripPackingList(state = [ ], action) {
    switch(action.type) {
        case 'SET_PACKING_LIST':
            return action.data.map((item) => {
                return { ...item };
            });
        default:
            return state;
    }
}
export default combineReducers({ tripId, tripDates, tripPlace,
    tripPurpose, tripShoppingList, tripPackingList,
});

```

Листинг Б.4 – Код модуля reducers/tripsForm.js на языке JavaScript

```

export function dataIsLoading(state = false, action) {
    if (action.type === 'DATA_IS_LOADING') return action.isLoading;
    return state;
}
export function tripsSummary(state = {}, action) {
    if (action.type === 'GET_TRIPS_SUMMARY') return action.trips;
    return state;
}
export function currentTripData(state = {}, action) {
    if (action.type === 'GET_CURRENT_TRIP') return action.trip;
    return state;
}
export default combineReducers({ dataIsLoading, tripsSummary,
    currentTripData,
});

```

Листинг Б.5 – Код модуля reducers.js на языке JavaScript

```

export function token(state = '', action) {
    if (action.type === 'SAVE_TOKEN') return action.token;
    return state;
}
export function userInfo(state = {}, action) {
    if (action.type === 'USER_INFO') return action.user;
    return state;
}

```

Листинг Б.6 – Код модуля reducers/index.js на языке JavaScript

```

const appReducer = combineReducers({ token, userInfo, mainForm, tripsForm, });
const rootReducer = (state, action) => {
    // when a logout action is dispatched it will reset redux state
    if (action.type === 'USER_LOGGED_OUT') {
        state = undefined;
    }
}

```

```

    return appReducer(state, action);
  };
  export default rootReducer;

```

Листинг Б.7 – Код модуля configureStore.js на языке JavaScript

```

export default function configureStore(initialState) {
  function select(state) {
    return state.token;
  }
  let currentToken;
  function handleChange() {
    let previousToken = currentToken;
    currentToken = select(store.getState());
    if (previousToken !== currentToken) {
      if (currentToken !== '')
        localStorage.setItem('token', currentToken);
      axios.defaults.headers.common['Authorization'] = 'Bearer ' +
currentToken;
    }
  }
  const store = createStore(rootReducer, initialState, composeWithDevTools(
    applyMiddleware(thunk),
  ));
  store.subscribe(handleChange);
  return store;
}

```

Листинг Б.8 – Код модуля SignIn.js на языке JavaScript

```

const styles = theme => ({
  paper: { marginTop: theme.spacing(8), display: 'flex', flexDirection:
'column', alignItems: 'center', },
  avatar: { margin: theme.spacing(1), backgroundColor:
theme.palette.secondary.main,
  },
  form: { width: '100%', marginTop: theme.spacing(1), },
  submit: { margin: theme.spacing(3, 0, 2), },
});
class SignIn extends Component {
  state = { email: "", password: "", errorText: "", };
  isValidEmail = email => {
    let regEmail = /^[a-z0-9_-]+\.[a-z0-9_-]+@[a-z0-9_-]+(\.[a-z0-9_-
]+)*\.[a-z]+$/;
    if (email.length < 6) return "Слишком короткий email";
    if (email.length > 40) return "Слишком длинный email";
    if (!regEmail.test(email)) return "Неверный формат email";
    return "";
  };
  handleChangeEmail = event => {
    this.setState({
      [event.target.name]: event.target.value
    });
    this.state.errorText = this.isValidEmail(event.target.value);
  };
  handleChangePassword = event => {
    this.setState({
      [event.target.name]: event.target.value
    });
  };
  handleSubmit = event => {
    event.preventDefault();
    this.props.userLoginFetch(this.state);
  };
}

```

```

};
render() {
  const {classes} = this.props;
  if (this.props.token)
    return (<Redirect to="/trips"/>);
  return (
    <Container component="main" maxWidth="xs">
      <CssBaseline/>
      <div className={classes.paper}>
        <Avatar className={classes.avatar}>
          <LockOutlinedIcon/>
        </Avatar>
        <Typography component="h1" variant="h5">
          Войти
        </Typography>
        <form className={classes.form} noValidate
onSubmit={this.handleSubmit}>
          <TextField
            error={this.state.errorText.length !== 0 }
            helperText={this.state.errorText}
            variant="outlined" margin="normal" required fullWidth
            id="email" label="Почта" name="email" autoComplete="email"
            autoFocus value={this.state.email}
onChange={this.handleChangeEmail}
          />
          <TextField
            variant="outlined" margin="normal" required fullWidth
            name="password" label="Пароль" type="password" id="password"
            autoComplete="current-password" value={this.state.password}
            onChange={this.handleChangePassword}
          />
          <FormControlLabel
            control={<Checkbox value="remember" color="primary"/>}
            label="Запомнить меня"
          />
          <Button
            disabled={ this.state.errorText.length !== 0 ||
this.state.email.length === 0 }
            type="submit" fullWidth variant="contained" color="primary"
            className={classes.submit}
          >
            Войти
          </Button>
          <Grid container>
            <Grid item xs>
              <Link component={LinkRouter} to="/register" variant="body2">
                {"Нет аккаунта? Зарегистрироваться"}
              </Link>
            </Grid>
          </Grid>
        </form>
      </div>
    </Container>
  );
}
}
SignIn.propTypes = {
  classes: PropTypes.object.isRequired,
};
const mapStateToProps = (state) => {
  return {
    token: state.token,
  };
};

```

```

};
const mapDispatchToProps = dispatch => ({
  userLoginFetch: userInfo => dispatch(userLoginFetch(userInfo))
});
export default connect(mapStateToProps,
mapDispatchToProps)(withStyles(styles)(SignIn));

```

Листинг Б.9 – Код модуля SignUp.js на языке JavaScript

```

const styles = theme => ({
  paper: { marginTop: theme.spacing(8), display: 'flex',
    flexDirection: 'column', alignItems: 'center',
  },
  avatar: { margin: theme.spacing(1), backgroundColor:
theme.palette.secondary.main,
  },
  form: { width: '100%', marginTop: theme.spacing(3), },
  submit: {
    margin: theme.spacing(3, 0, 2),
  },
});
class SignUp extends Component {
  state = { firstName: "", lastName: "", email: "",
    password: "", errorText: "",
  };
  isValidEmail = email => {
    let regEmail = /^[a-z0-9_-]+\.[a-z0-9_-]+@[a-z0-9_-
]+)*\.[a-z]+$/;
    if (email.length < 6) return "Слишком короткий email";
    if (email.length > 40) return "Слишком длинный email";
    if (!regEmail.test(email)) return "Неверный формат email";
    return "";
  };
  handleChangeEmail = event => {
    this.setState({ [event.target.name]: event.target.value });
    this.state.errorText = this.isValidEmail(event.target.value);
  };
  handleChange = event => {
    this.setState({ [event.target.name]: event.target.value });
  };
  handleSubmit = event => { event.preventDefault();
    this.props.userPostFetch(this.state)
  };
  render() {
    const {classes} = this.props;
    if (this.props.token)
      return (<Redirect to="/" />);
    return (
      <Container component="main" maxWidth="xs">
        <CssBaseline />
        <div className={classes.paper}>
          <Avatar className={classes.avatar}>
            <LockOutlinedIcon />
          </Avatar>
          <Typography component="h1" variant="h5">
            Зарегистрироваться
          </Typography>
          <form className={classes.form} noValidate
onSubmit={this.handleSubmit}>
            <Grid container spacing={2}>
              <Grid item xs={12} sm={6}>
                <TextField
                  autoComplete="fname" name="firstName"

```

```

        variant="outlined" required fullWidth
        id="firstName" label="Имя" autoFocus
        value={this.state.firstName}
        onChange={this.handleChange}
      />
    </Grid>
    <Grid item xs={12} sm={6}>
      <TextField
        variant="outlined" required fullWidth
        id="lastName" label="Фамилия" name="lastName"
        autoComplete="lname"
        value={this.state.lastName}
        onChange={this.handleChange}
      />
    </Grid>
    <Grid item xs={12}>
      <TextField
        error={this.state.errorText.length !== 0 }
        helperText={this.state.errorText}
        variant="outlined" required fullWidth
        id="email" label="Почта" name="email"
        autoComplete="email"
        value={this.state.email}
        onChange={this.handleChangeEmail}
      />
    </Grid>
    <Grid item xs={12}>
      <TextField
        variant="outlined" required fullWidth
        type="password"
        id="password" name="password" label="Пароль"
        autoComplete="current-password"
        value={this.state.password}
        onChange={this.handleChange}
      />
    </Grid>
  </Grid>
  <Button
    disabled={ (this.state.errorText.length !== 0 ||
this.state.email.length === 0) }
    type="submit"
    fullWidth
    variant="contained"
    color="primary"
    className={classes.submit}
  >
    Зарегистрироваться
  </Button>
  <Grid container>
    <Grid item>
      <Link component={LinkRouter} to="/auth" variant="body2">
        Уже есть аккаунт? Войти
      </Link>
    </Grid>
  </Grid>
</form>
</div>
</Container>
);
}
}
SignUp.propTypes = {
  classes: PropTypes.object.isRequired,

```



```

        id="date-picker-inline-to" label="Обратно"
        value={this.props.dateTo}
        onChange={this.handleChangeTo}
        KeyboardButtonProps={{
          'aria-label': 'change date',
        }}
      />
    </Box>
  </Grid>
</MuiPickersUtilsProvider>
</Typography>
</Paper>
)
}
}
const mapStateToProps = state => ({
  token: state.token,
  dateFrom: state.mainForm.tripDates.dateFrom,
  dateTo: state.mainForm.tripDates.dateTo,
  currentTripData: state.tripsForm.currentTripData.data,
});
const mapDispatchToProps = dispatch => ({
  dateFromChange: dateFrom => dispatch(tripDateFrom(dateFrom)),
  dateToChange: dateTo => dispatch(tripDateTo(dateTo)),
});
export default connect(mapStateToProps,
mapDispatchToProps)(withStyles(styles)(WhenForm));

```

Листинг Б.11 – Код модуля WhyForm.js на языке JavaScript

```

const styles = theme => ({ root: { flexGrow: 1, },
  paper: { padding: theme.spacing(2), textAlign: 'left',
    color: theme.palette.text.secondary,
  },
  typography: {fontSize: 20, },
});
class WhyForm extends Component {
  componentDidMount() {
    if (this.props.token) {
      if (this.props.currentTripData) {
        this.props.purposeChange({label:
this.props.currentTripData.trip.o_trip.purpose});
      }
    }
  }
  handlePurposeChange = (event, newPurpose) => {
    this.props.purposeChange(newPurpose);
  };
  render() {
    const {classes} = this.props;
    return (
      <Paper className={classes.paper}>
        <Box mb={2}>
          <Typography className={classes.typography}>
            Зачем
          </Typography>
        </Box>
        <Typography>
          <Autocomplete
            id="country-select-demo" value={this.props.purpose}
            style={{ width: 300 }}
            value={this.props.purpose}
            onChange={this.handlePurposeChange}

```

```

        options={countries}
        classes={{
          option: classes.option,
        }}
        autoHighlight
        getOptionLabel={(option) => option.label}
        renderOption={(option) => (
          <React.Fragment> {option.label} </React.Fragment>
        )}
        renderInput={(params) => (
          <TextField
            {...params}
            label="Тип поездки" variant="outlined"
            inputProps={{
              ...params.inputProps,
              autoComplete: 'new-password', // disable autocomplete and
autofill
            }}
          />
        )}
      />
    </Typography>
  </Paper>
);
}
}
const countries = [
  { label: 'Командировка' }, { label: 'Кемпинг' }, { label: 'Выходные' },
  { label: 'Отдых на пляже' }, { label: 'Круиз' }, { label: 'Пеший туризм' },
  { label: 'Другое' },
];
const mapStateToProps = state => ({
  token: state.token,
  purpose: state.mainForm.tripPurpose.purpose,
  currentTripData: state.tripsForm.currentTripData.data,
});
const mapDispatchToProps = dispatch => ({
  purposeChange: purpose => dispatch(tripPurpose(purpose)),
});
export default connect(mapStateToProps,
mapDispatchToProps)(withStyles(styles)(WhyForm));

```

Листинг Б.12 – Код модуля WhereForm.js на языке JavaScript

```

const styles = theme => ({
  root: { flexGrow: 1, },
  paper: { padding: theme.spacing(2), color: theme.palette.text.secondary, },
  typography: { fontSize: 20, },
});
class WhereForm extends Component {
  componentDidMount() {
    if (this.props.token) {
      if (this.props.currentTripData) {
        this.props.placeChange({label:
this.props.currentTripData.trip.o_trip.place});
      }
    }
  }
  handlePlaceChange = (event, newPlace) => {
    this.props.placeChange(newPlace);
  };
  render() {
    const {classes} = this.props;

```

```

return (
  <Paper className={classes.paper}>
    <Box mb={2}>
      <Typography className={classes.typography}>
        Куда
      </Typography>
    </Box>
    <Typography>
      <Autocomplete
        id="destination-select-demo"
        style={{ width: 300 }}
        value={this.props.place}
        onChange={this.handlePlaceChange}
        options={cities}
        classes={{
          option: classes.option,
        }}
        autoHighlight
        getOptionLabel={option => option.label}
        renderOption={option => (
          <React.Fragment> {option.label} </React.Fragment>
        )}
        renderInput={(params) => (
          <TextField
            {...params}
            label="Место назначения"
            variant="outlined"
            inputProps={{
              ...params.inputProps,
              autoComplete: 'new-password', // disable autocomplete and
autofill
            }}
          </>
        )}
      </Typography>
    </Paper>
  );
}
}
const cities = [
  { label: 'Москва' }, { label: 'Нью-Йорк' }, { label: 'Сеул' },
  { label: 'Лондон' }, { label: 'Токио' },];
const mapStateToProps = state => ({
  token: state.token,
  place: state.mainForm.tripPlace.place,
  currentTripData: state.tripsForm.currentTripData.data,
});
const mapDispatchToProps = dispatch => ({
  placeChange: place => dispatch(tripPlace(place)),
});
export default connect(mapStateToProps,
mapDispatchToProps)(withStyles(styles)(WhereForm));

```

Листинг Б.13 – Код модуля PackingList.js на языке JavaScript

```

const styles = theme => ({
  root: { flexGrow: 1, },
  paper: { padding: theme.spacing(2), textAlign: 'left',
    color: theme.palette.text.secondary,
  },
  typography: { fontSize: 20, },
});

```

```

const editColumnMessages = {
  addCommand: 'Добавить', editCommand: 'Изменить', deleteCommand: 'Удалить',
  commitCommand: 'Сохранить', cancelCommand: 'Отменить',
};
const tableMessages = { noData: 'Нет записей', };
let currentCategory = "";
const setCurrentCategory = grNames => {
  currentCategory = grNames.length > 0 ? grNames[grNames.length - 1] : "";
};
const getRowId = row => row.id;
class PackingList extends Component {
  componentDidMount() {
    if (this.props.token) {
      if (this.props.currentTripData) {
        if (this.props.currentTripData.trip.o_packing_list) {
this.props.setTripPackingList(this.props.currentTripData.trip.o_packing_list)
;
          } else
            this.props.setTripPackingList([]);
        }
      }
    }
    state = {
      columns: [
        { title: 'Название', name: 'title' },
        { title: 'Количество', name: 'quantity' },
        { title: 'Категория', name: 'category' },
      ],
    };
    commitChanges = ({ added, changed, deleted }) => {
      let changedRows;
      if (added) {
        const startingAddedId = this.props.packingList.length > 0 ?
this.props.packingList[this.props.packingList.length - 1].id + 1 : 0;
        changedRows = [
          ...this.props.packingList,
          ...added.map((row, index) => ({
            id: startingAddedId + index,
            category: currentCategory,
            ...row,
          })),
        ];
      }
      if (changed) {
        changedRows = this.props.packingList.map(row => (changed[row.id] ? {
...row, ...changed[row.id] } : row));
      }
      if (deleted) {
        const deletedSet = new Set(deleted);
        changedRows = this.props.packingList.filter(row =>
!deletedSet.has(row.id));
      }
      this.props.setTripPackingList(changedRows);
    };
    render() {
      const {classes} = this.props;
      return (
        <Paper className={classes.paper}>
          <Box mb={2}>
            <Typography className={classes.typography}>
              Взять с собой
            </Typography>
          </Box>
        </Paper>
      );
    }
  }
}

```

```

        </Typography>
      </Box>
      <Typography>
        <Grid
          rows={this.props.packingList}
          columns={this.state.columns}
          getRowId={getRowId}
        >
          <EditingState onCommitChanges={this.commitChanges}/>
          <GroupingState
            grouping={[{ columnName: 'category' }]}
            onExpandedGroupsChange={ (expandedGroups) =>
setCurrentCategory(expandedGroups) }
          />
          <IntegratedGrouping />
          <Table messages={tableMessages} />
          <TableHeaderRow />
          <TableEditRow />
          <TableGroupRow />
          <TableEditColumn showAddCommand showEditCommand
            showDeleteCommand
            width={175} messages={editColumnMessages}
          />
        </Grid>
      </Typography>
    </Paper>
  );
}
}
const mapStateToProps = state => ({
  token: state.token,
  packingList: state.mainForm.tripPackingList,
  currentTripData: state.tripsForm.currentTripData.data,
});
const mapDispatchToProps = dispatch => ({
  setTripPackingList: data => dispatch(setTripPackingList(data)),
});
export default connect(mapStateToProps,
mapDispatchToProps)(withStyles(styles)(PackingList));

```

Листинг Б.14 – Код модуля ShareButtons.js на языке JavaScript

```

const styles = theme => ({
  root: { flexGrow: 1, },
  paper: { padding: theme.spacing(2), textAlign: 'center',
    color: theme.palette.text.secondary,
  },
});
class ShareButtons extends Component {
  handleSave = () => {
    console.log(this.props.tripInfo);
    this.props.tripDataSend(this.props.tripInfo);
    this.props.history.push('/trips');
  };
  handlePrint = () => { console.log("print") };
  handleEmail = () => { console.log("email") };
  render() {
    const {classes} = this.props;
    return (
      <Grid container justify="space-around" >
        <Paper className={classes.paper} elevation={0}>
          {this.props.token && (
            <IconButton

```

```

        aria-label="save trip" aria-controls="menu-appbar"
        aria-haspopup="true" onClick={this.handleSave}
      >
        <Fab color="primary"> <SaveSharpIcon /> </Fab>
      </IconButton>
    )}
    <IconButton
      aria-label="save trip" aria-controls="menu-appbar"
      aria-haspopup="true"
      onClick={this.handlePrint}
    >
      <Fab color="primary"> <PrintSharpIcon /> </Fab>
    </IconButton>
    <IconButton
      aria-label="save trip" aria-controls="menu-appbar"
      aria-haspopup="true"
      onClick={this.handleEmail}
    >
      <Fab color="primary"> <EmailSharpIcon /> </Fab>
    </IconButton>
  </Paper>
</Grid>
);
}
}
const mapStateToProps = state => ({
  token: state.token,
  tripInfo: state.mainForm,
});
const mapDispatchToProps = dispatch => ({
  tripDataSend: tripInfo => dispatch(tripDataSend(tripInfo))
});
export default connect(mapStateToProps,
  mapDispatchToProps)(withStyles(styles)(ShareButtons));

```

Листинг Б.15 – Код модуля TripCard.js на языке JavaScript

```

const styles = theme => ({
  root: { flexGrow: 1, },
  paper: { padding: theme.spacing(2), textAlign: 'center',
    color: theme.palette.text.secondary,
  },
});
class TripCard extends Component {
  handleSubmit = event => {
    event.preventDefault();
    this.props.userLoginFetch(this.state)
  };

  render() {
    const {classes} = this.props;
    return (
      <div className={classes.root}>
        <Container fixed>
          <Grid container justify="center" alignItems="stretch"
            spacing={3}
          >
            <Grid item xs={12}> <Box mt={2}> <WhenCard /> </Box> </Grid>
            <Grid item xs={12}> <WhereCard /> </Grid>
            <Grid item xs={12}> <WhyCard /> </Grid>
            <Grid item xs={12}> <PackingListCard/> </Grid>
            <Grid item xs={12}> <ShoppingCardList/> </Grid>
            <Grid item xs={12}>

```

```

        <ShareButtons history={this.props.history}/>
      </Grid>
    </Grid>
  </Container>
</div>
);
}
}

```

```
export default (withStyles(styles)(TripCard));
```

Листинг Б.16 – Код модуля Header.js на языке JavaScript

```

const styles = theme => ({
  root: { flexGrow: 1, marginTop: 20, },
  title: { flexGrow: 1, },
});
class Header extends Component {
  state = { anchorEl: undefined, open: false, };
  handleMenu = event => {
    this.setState({ open: true, anchorEl: event.currentTarget });
  };
  handleProfile = () => {
    console.log("profile");
    this.setState({ open: false });
  };
  handleClose = () => {
    console.log("close");
    this.setState({ open: false });
  };
  handleExit = (event) => {
    this.setState({ open: false, anchorEl: null });
    event.preventDefault();
    localStorage.removeItem("token");
    this.props.logoutUser();
  };
  render() {
    const {classes} = this.props;
    return (
      <AppBar position="sticky">
        <Toolbar>
          <Typography variant="h6" className={classes.title}>
            Планирование путешествий
          </Typography>
          {this.props.token && (
            <div>
              <IconButton
                aria-label="account of current user" color="inherit"
                aria-controls="menu-appbar" aria-haspopup="true"
                onClick={this.handleMenu}
              >
                <AccountCircle/>
              </IconButton>
              <Menu
                id="menu-appbar"
                anchorEl={this.state.anchorEl}
                anchorOrigin={{ vertical: 'top', horizontal: 'right', }}
                keepMounted
                transformOrigin={{ vertical: 'top', horizontal: 'right', }}
                open={this.state.open}
                onClose={this.handleClose}
              >
                <MenuItem onClick={this.handleProfile}>Profile</MenuItem>

```

```

        <MenuItem onClick={this.handleExit}>Exit</MenuItem>
      </Menu>
    </div>
  )}
  {!this.props.token && (
    <div>
      <Button color='inherit' component={Link}
to='/auth'>Вход</Button>
      <Button color='inherit' component={Link}
to='/register'>Регистрация</Button>
    </div>
  )}
</Toolbar>
</AppBar>
);
}
}
Header.propTypes = { classes: PropTypes.object.isRequired, };
const mapStateToProps = state => ({ token: state.token, });
const mapDispatchToProps = dispatch => ({
  logoutUser: () => dispatch(onLogout()),
});
export default connect(mapStateToProps,
mapDispatchToProps)(withStyles(styles)(Header));

```

Листинг Б.17 – Код модуля MainPage.js на языке JavaScript

```

const styles = theme => ({
  root: { flexGrow: 1, },
  title: { flexGrow: 1, },
});
class MainPage extends Component {
  componentDidMount() {
    if (this.props.token) {
      if (this.props.match) {
        this.props.currentTripDataGet(this.props.match.params.id);
        this.props.idChange(this.props.match.params.id);
      } else this.props.idChange(0);
    }
  }
  render() {
    const {classes} = this.props;
    if (this.props.dataIsLoading)
      return (
        <div className={classes.root}> <Header/> </div>
      );
    return (
      <div className={classes.root}>
        <Header/> <TripCard history={this.props.history} />
      </div>
    );
  }
}
MainPage.propTypes = {
  classes: PropTypes.object.isRequired,
  open: PropTypes.object.isRequired,
};
const mapStateToProps = state => ({
  token: state.token,
  dataIsLoading: state.tripsForm.dataIsLoading,
  currentTripData: state.tripsForm.currentTripData,
});
const mapDispatchToProps = dispatch => ({

```

```

    idChange: id => dispatch(tripId(id)),
    currentTripDataGet: tripId => dispatch(currentTripDataGet(tripId)),
  });
export default connect(mapStateToProps,
  mapDispatchToProps)(withStyles(styles)(MainPage));

```

Листинг Б.18 – Код модуля TripSummary.js на языке JavaScript

```

const styles = theme => ({ root: { flexGrow: 1, },
  paper: { padding: theme.spacing(2), textAlign: 'center',
    color: theme.palette.text.secondary,
  },
  typographySecondary: { textAlign: 'left', },
  typography: { fontSize: 20, textAlign: 'left', },
});
class TripSummary extends Component {
  handleDelete = event => {
    event.preventDefault();
    this.props.deleteTrip(this.props.trip.id);
    this.props.tripDataGet();
  };
  render() {
    let trip = this.props.trip;
    const {classes} = this.props;
    return (
      <Paper className={classes.paper}>
        <Grid container direction="row"
          justify="space-between" alignItems="flex-start"
        >
          <Box ml={2}>
            <Grid container direction="row"
              justify="flex-start" alignItems="flex-start"
            >
              <Box width={150}>
                <Grid item>
                  <Typography className={classes.typographySecondary}>
                    Тутда
                  </Typography>
                  <Box color="text.primary">
                    <Typography className={classes.typography}>
                      {trip.date_from.substring(0, 10)}
                    </Typography>
                  </Box>
                </Grid>
              </Box>
              <Box ml={2} width={150}>
                <Grid item>
                  <Typography className={classes.typographySecondary}>
                    Оттуда
                  </Typography>
                  <Box color="text.primary">
                    <Typography className={classes.typography}>
                      {trip.date_to.substring(0, 10)}
                    </Typography>
                  </Box>
                </Grid>
              </Box>
              <Box ml={2} width={150}>
                <Grid item>
                  <Typography className={classes.typographySecondary}>
                    Куда
                  </Typography>
                  <Box color="text.primary">

```

```

        <Typography className={classes.typography}>
          {trip.place}
        </Typography>
      </Box>
    </Grid>
  </Box>
  <Box ml={2} width={150}>
    <Grid item>
      <Typography className={classes.typographySecondary}>
        Зачем
      </Typography>
      <Box color="text.primary">
        <Typography className={classes.typography}>
          {trip.purpose}
        </Typography>
      </Box>
    </Grid>
  </Box>
</Grid>
<Box mr={2} >
  <IconButton
    aria-label="delete"
    onClick={this.handleDelete}
  >
    <DeleteSharpIcon/>
  </IconButton>
</Box>
</Grid>
</Paper>
);
}
}
const mapStateToProps = state => ({ token: state.token, });
const mapDispatchToProps = dispatch => ({
  deleteTrip: trip_id => dispatch(tripDataDelete(trip_id)),
  tripDataGet: () => dispatch(tripDataGet()),
});
export default connect(mapStateToProps,
mapDispatchToProps)(withStyles(styles)(TripSummary));

```

Листинг Б.19 – Код модуля TripListMap.js на языке JavaScript

```

const styles = theme => ({
  plainLink: { textDecoration: 'none', }
});
class TripListMap extends Component {
  render() {
    const {classes} = this.props;
    if (Array.isArray(this.props.tripsSummary))
      return (this.props.tripsSummary.map((trip) => (
        <Grid item xs={12} className = { classes.plainLink } component={Link}
to={`/trip/${trip.id}`}>
        <TripSummaryCard trip={trip}/>
      </Grid>
    )));
    return(
      <div>Loading</div>
    )
  }
}
export default withStyles(styles)(TripListMap);

```

Листинг Б.20 – Код модуля TripsList.js на языке JavaScript

```
const styles = theme => ({
  root: { flexGrow: 1, },
  paper: { padding: theme.spacing(2), textAlign: 'center',
    color: theme.palette.text.secondary,
  },
});

class TripsList extends Component {
  componentDidMount() { this.props.tripDataGet(); }
  render() {
    const {classes} = this.props;
    if (!this.props.token)
      return (<Redirect to="/" />);
    return (
      <div className={classes.root}>
        <Header />
        <Container fixed>
          <Grid container justify="center"
            alignItems="stretch" spacing={3}
          >
            <Grid item xs={12}>
              <Paper className={classes.paper} elevation={0}>
                <IconButton
                  aria-label="add new trip"
                  aria-controls="menu-appbar"
                  aria-haspopup="true"
                  component={Link} to="/"
                >
                  <Fab color="secondary"> <AddIcon /> </Fab>
                </IconButton>
              </Paper>
            </Grid>
            <TripListMap tripsSummary={this.props.tripsSummary} />
          </Grid>
        </Container>
      </div>
    );
  }
}

const mapStateToProps = state => ({
  token: state.token,
  tripsSummary: state.tripsForm.tripsSummary,
});
const mapDispatchToProps = dispatch => ({
  tripDataGet: () => dispatch(tripDataGet()),
});
export default connect(mapStateToProps,
  mapDispatchToProps)(withStyles(styles)(TripsList));
```

Листинг Б.21 – Код модуля App.js на языке JavaScript

```
class App extends Component {
  render() {
    const { location } = this.props;
    return (
      <div>
        <Switch>
          <Route location={location} path="/" exact
            render={() => {
              return (<MainPage history={this.props.history} />);
            }}
          />
        </Switch>
      </div>
    );
  }
}
```

```

    <Route
      location={location} path='/auth' exact component={SignIn}
    />
    <Route
      location={location} path='/register' exact component={SignUp}
    />
    <Route
      location={location} path='/trips' exact component={TripsListCard}
    />
    <Route
      location={location} path='/trip/:id' exact component={MainPage}
    />
  </Switch>
</div>
)
}
}
export default connect(null, mapDispatchToProps)(App);

```

Листинг Б.22 – Код модуля index.js на языке JavaScript

```

const store = configureStore();
const theme = createMuiTheme({
  palette: {
    primary: { main: '#563d7c' },
    secondary: { main: '#5b8938' },
  },
}, ruRU);
ReactDOM.render(
  <BrowserRouter>
    <Provider store={store}>
      <ThemeProvider theme={theme}>
        <Route component={App}/>
      </ThemeProvider>
    </Provider>
  </BrowserRouter>,
  document.getElementById('root')
);
serviceWorker.unregister();

```

Приложение В
ЛИСТИНГ СЕРВЕРНОГО ПРИЛОЖЕНИЯ НА ЯЗЫКЕ JAVASCRIPT
(рекомендуемое)

Листинг В.1 – Код модуля auth.js на языке JavaScript

```
import bcrypt from "bcrypt";
import express from 'express';
const router = express.Router();
import jwt from 'jsonwebtoken';
import passport from 'passport';
import User from '../../database/User';
import db from '../../database/db';
router.post('/', function (req, res, next) {
  passport.authenticate('local', {session: false}, (err, user, info) => {
    if (err || !user) {
      return res.status(400).json({
        message: info ? info.message : 'Login failed',
        user : user
      });
    }
    req.login(user, {session: false}, (err) => {
      if (err) { res.send(err); }
      const token = jwt.sign(user, 'your_jwt_secret');
      return res.json({user, token});
    });
  })
  (req, res);
});
router.post('/register', (req, res) => {
  let { user } = req.body;
  let passwordBcrypt = bcrypt.hashSync(user.password, User.salt);
  db.one('INSERT INTO users(name, last_name, password, email) VALUES(${name},
${last_name}, ${password}, ${email}) RETURNING id',
    { name: user.firstName,
      last_name: user.lastName,
      password: passwordBcrypt,
      email: user.email
    }
  )
  .then((newUser) => {
    let userForToken = { id: newUser.id, user: user.email, name:
user.firstName };
    const token = jwt.sign(userForToken, 'your_jwt_secret');

    res.json({userForToken, token});
  })
  .catch(error => { res.json({ error: { msg: 'error' } }); });
});
module.exports = router;
```

Листинг В.2 – Код модуля form.js на языке JavaScript

```
import express from 'express';
import db from '../../database/db';
import axios from "axios/index";
import {Link} from "react-router-dom";
const router = express.Router();
router.post('/send', (req, res) => {
  let user_id = req.user.id;
  let trip = req.body;
  let shoppingList = trip.tripShoppingList.map((sl) => {
    return { sl_id: 0, ...sl } });
  let packingList = trip.tripPackingList.map((pl) => {
    return { pl_id: 0, ...pl } });
  console.log(JSON.stringify(shoppingList).toString());
  console.log(trip);
```

```

    db.one('select trips_persistence(${id}, ${user_id}, ${date_from}::date,
    ${date_to}::date, ${place}, ${purpose}, ${shopping_list}, ${packing_list})',
    { id: Number(trip.tripId.id),
      user_id: user_id,
      date_from: trip.tripDates.dateFrom,
      date_to: trip.tripDates.dateTo,
      place: trip.tripPlace.place.label,
      purpose: trip.tripPurpose.purpose.label,
      shopping_list: JSON.stringify(shoppingList).toString(),
      packing_list: JSON.stringify(packingList).toString(),
    }
  )
  .then((operation) => { console.log(operation); res.json({w: 'ww'}); })
  .catch(error => {
    console.log(error);
    res.json({ error: { msg: 'error' } });
  });
  console.log('end');
});

router.post('/delete', (req, res) => {
  db.result('select trips_persistence(${id})', // ${id}
  { id: req.query.trip_id, }
  .then((result) => { console.log(result); res.json({w: 'ww'}); })
  .catch(error => {
    console.log(error);
    res.json({ error: { msg: 'error' } });
  });
});

router.get('/get_trips_summary', (req, res) => {
  let user_id = req.user.id;
  db.any('SELECT id, date_from, date_to, get_place_name(place_id) as place,
  get_purpose_name(purpose_id) as purpose FROM\n' +
  'trips WHERE user_id = ${user_id} AND parent_id isnull',
  { user_id: user_id, }
  )
  .then((trips) => { res.json({trips}); })
  .catch(error => {
    console.log(error);
    res.json({ error: { msg: 'error' } });
  });
});

router.get('/get_current_trip', (req, res) => {
  db.one('select * from trip_summary_json(${id})',
  { id: req.query.trip_id, }
  )
  .then((trip) => { res.json({trip}); })
  .catch(err => { console.log(error); res.json({err:{ msg: 'error' } });
  });
});
module.exports = router;

```

Листинг В.3 – Код модуля user.js на языке JavaScript

```

var express = require('express');
var router = express.Router();
import db from '../../database/db';
import User from '../../database/User';
router.get('/', function(req, res, next) {
  res.send('respond with a resource');
});

```

```

router.get('/profile', function(req, res, next) {
  console.log(req);
  User.findById(req.user.id)
    .then(user => { res.send({ user: user }) })
    .catch(err => { res.send(err); });
});
module.exports = router;

```

Листинг В.4 – Код модуля passport.js на языке JavaScript

```

const passport = require('passport');
const passportJWT = require("passport-jwt");
const ExtractJWT = passportJWT.ExtractJwt;
const LocalStrategy = require('passport-local').Strategy;
const JWTStrategy = passportJWT.Strategy;
import User from '../database/User';
passport.use(new LocalStrategy({
  usernameField: 'email',
  passwordField: 'password'
}),
function (email, password, cb) {

  return User.findOne({email, password})
    .then(user => {
      console.log(User.checkPassword(password, user.password));
      if (!user || !User.checkPassword(password, user.password)) {
        return cb(null, false, {message: 'Incorrect email or password.'});
      }
      return cb(null, { id: user.id, user: email, name: user.name }, {
        message: 'Logged In Successfully'
      });
    })
    .catch(err => { return cb(err); });
}
);

passport.use(new JWTStrategy({
  jwtFromRequest: ExtractJWT.fromAuthHeaderAsBearerToken(),
  secretOrKey : 'your_jwt_secret'
}),
function (jwtPayload, cb) {
  return User.findById(jwtPayload.id)
    .then(user => { console.log(jwtPayload); return cb(null, user); })
    .catch(err => { console.log(err); return cb(err); });
}
);

```

Листинг В.5 – Код модуля server/app.js на языке JavaScript

```

var express = require('express');
var path = require('path');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
var user = require('./routes/user');
var auth = require('./routes/auth');
var form = require('./routes/form');
const passport = require('passport');
require('./passport');
var app = express();
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');
app.use(logger('dev'));

```

```

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static('build'));
app.use('/api/user', passport.authenticate('jwt', {session: false}), user);
app.use('/api/auth', auth);
app.use('/api/trips', passport.authenticate('jwt', {session: false}), form);
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});
app.use(function(err, req, res, next) {
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};
  res.status(err.status || 500);
});
module.exports = app;

```

Листинг В.6 – Код модуля server/index.js на языке JavaScript

```

import app from './app'
app.listen(8080, () => console.log('Running on localhost:8080'));

```

Листинг В.7 – Код модуля database/User.js на языке JavaScript

```

import db from './db';
import bcrypt from 'bcrypt';
const User = {
  salt: bcrypt.genSaltSync(10),
  findOne(login) {
    return db.one('SELECT id, password, name FROM users WHERE email = ${email}', { email: login.email });
  },
  findById(id) {
    return db.one('SELECT id, password, email, name, last_name FROM users WHERE id = ${id}', { id: id });
  },
  checkPassword(password, passwordHash) {
    return bcrypt.compareSync(password, passwordHash);
  },
};
export default User;

```

Листинг В.8 – Код модуля db.js на языке JavaScript

```

const pgp = require('pg-promise')({
  schema: 'tp',
});
const cn = {
  host: 'localhost',
  port: 5432,
  database: 'postgres',
  //schema: 'tp',
  user: 'tp_api_user',
  password: 'postgre',
  max: 30 // use up to 30 connections
};

const db = pgp(cn);
export default db;

```

Приложение Г
ЛИСТИНГ ОБРАБОТКИ ЗАПРОСОВ К БАЗЕ ДАННЫХ
(рекомендуемое)

Листинг Г.1 – Код функции get_place_id на языке plpgsql

```
create or replace function get_place_id(p_place character varying) returns
integer
    language plpgsql
as $$
DECLARE
    r_id integer;

BEGIN
    SELECT id FROM places WHERE places.name = p_place INTO r_id;
    IF NOT FOUND THEN
        INSERT INTO places(name) VALUES(p_place) RETURNING id INTO r_id;
    END IF;
    RETURN r_id;
END;
$$
;
```

Листинг Г.2 – Код функции get_place_name на языке plpgsql

```
create or replace function get_place_name(p_place_id integer) returns
character varying
    language plpgsql
as $$
DECLARE
    r_name varchar(64);

BEGIN
    SELECT name FROM places WHERE places.id = p_place_id INTO r_name;
    RETURN r_name;
END;
$$
;
```

Листинг Г.3 – Код функции get_purpose_id на языке plpgsql

```
create or replace function get_purpose_id(p_purpose character varying)
returns integer
    language plpgsql
as $$

DECLARE
    r_id integer;

BEGIN
    SELECT id FROM purposes WHERE purposes.name = p_purpose INTO r_id;
    IF NOT FOUND /*r_id ISNULL*/ THEN
        INSERT INTO purposes(name) VALUES(p_purpose) RETURNING id INTO r_id;
    END IF;
    RETURN r_id;
END;
$$
;
```

Листинг Г.4 – Код функции get_purpose_name на языке plpgsql

```
create or replace function get_purpose_name(p_purpose_id integer) returns
character varying
    language plpgsql
as $$
DECLARE
    r_name varchar(64);
```

```

BEGIN
    SELECT name FROM purposes WHERE purposes.id = p_purpose_id INTO r_name;
    RETURN r_name;
END;
$$
;

```

Листинг Г.5 – Код функции trip_summary_json на языке plpgsql

```

create or replace function trip_summary_json(p_trip_id integer, OUT o_trip
json, OUT o_shopping_list json, OUT o_packing_list json) returns record
    language plpgsql
as $$
begin
    select row_to_json(trips)
    from (select id, date_from, date_to, get_place_name(place_id) as place,
get_purpose_name(purpose_id) as purpose
        from trips WHERE id = p_trip_id) as trips into o_trip;

    select json_agg(shopping_list)
    from (select id as sl_id, title, quantity, sum
        from shopping_list WHERE trip_id = p_trip_id) as shopping_list into
o_shopping_list;

    select json_agg(packing_list)
    from (select id as pl_id, title, quantity, category
        from packing_list WHERE trip_id = p_trip_id) as packing_list into
o_packing_list;

    return;
end;
$$
;

```

Листинг Г.6 – Код функции trips_persistence на языке plpgsql

```

create or replace function trips_persistence(p_id integer) returns integer
    language plpgsql
as $$
begin
    return trips_persistence(0 - p_id, 0, null, null, 0, 0, null, null);
end;
$$
;

```

Листинг Г.7 – Код функции trips_persistence на языке plpgsql

```

create or replace function trips_persistence(p_id integer, p_user_id integer,
p_date_from date, p_date_to date, p_place character varying, p_purpose
character varying, p_shopping_list text, p_packing_list text) returns integer
    language plpgsql
as $$
begin
    return trips_persistence(p_id, p_user_id, p_date_from,
        p_date_to, get_place_id(p_place),
get_purpose_id(p_purpose), p_shopping_list, p_packing_list);
end;
$$
;

```

Листинг Г.8 – Код функции trips_persistence на языке plpgsql

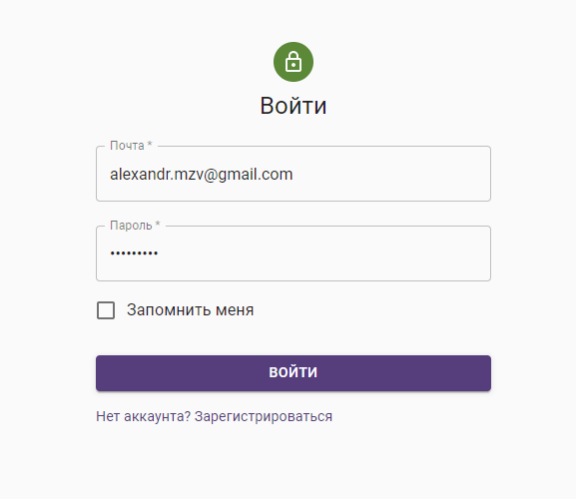
```

create or replace function trips_persistence(p_id integer, p_user_id integer,
p_date_from date, p_date_to date, p_place_id integer, p_purpose_id integer,
p_shopping_list text, p_packing_list text) returns integer
    language plpgsql
as $$
DECLARE
    r_id integer;
    r_sl integer;
    r_pl integer;
begin
    if p_id > 0 then
        update trips
            SET (date_from, date_to, place_id, purpose_id) =
                (select p_date_from, p_date_to, p_place_id, p_purpose_id)
            where id = p_id returning id into r_id;
    elseif p_id = 0 THEN
        insert into trips(user_id, date_from, date_to, place_id, purpose_id)
            values (p_user_id, p_date_from, p_date_to, p_place_id, p_purpose_id)
            returning id into r_id;
    else
        delete from shopping_list where trip_id = 0 - p_id;
        delete from trips where id = 0 - p_id returning id into r_id;
    end if;
    if p_id >= 0 then
        WITH
            pm as (select * from
json_populate_recordset(null::shopping_list_type, p_shopping_list::json)),
            del as (delete from shopping_list as sl where trip_id = r_id and not
sl.id = any (select sl_id from pm where sl_id > 0) returning sl.id),
            upd as (update shopping_list as sl_upd
set (title, quantity, sum) =
(select pm.title, pm.quantity, pm.sum) from pm
WHERE sl_upd.id = pm.sl_id and not (sl_upd.title = pm.title and
sl_upd.quantity = pm.quantity and sl_upd.sum = pm.sum)
returning sl_upd.id),
            ins as (insert into shopping_list(trip_id, title, quantity, sum)
select r_id, title, quantity, sum from pm as sl where sl_id = 0 returning id)
select (select count(*) from del) + (select count(*) from upd) + (select
count(*) from ins) into r_sl;

        WITH
            pm as (select * from
json_populate_recordset(null::packing_list_type, p_packing_list::json)),
            del as (delete from packing_list as pl where trip_id = r_id and not
pl.id = any (select pl_id from pm where pl_id > 0) returning pl.id),
            upd as (update packing_list as pl_upd
set (title, quantity, category) =
(select pm.title, pm.quantity, pm.category) from pm
WHERE pl_upd.id = pm.pl_id and not (pl_upd.title = pm.title and
pl_upd.quantity = pm.quantity and pl_upd.category = pm.category)
returning pl_upd.id),
            ins as (insert into packing_list(trip_id, title, quantity, category)
select r_id, title, quantity, category from pm as pl where pl_id = 0
returning id)
select (select count(*) from del) + (select count(*) from upd) + (select
count(*) from ins) into r_pl;
    end if;
    return r_id;
end ;
$$
;

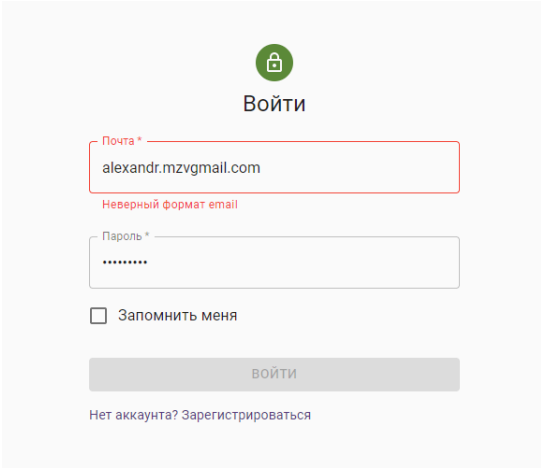
```

Приложение Д
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
(рекомендуемое)



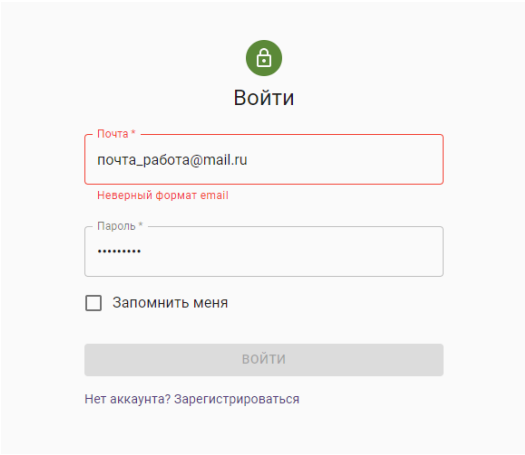
The screenshot shows a login form titled "Войти" (Login) with a green lock icon. It contains two input fields: "Почта *" (Email) with the value "alexandr.mzv@gmail.com" and "Пароль *" (Password) with masked characters. Below the password field is a checkbox labeled "Запомнить меня" (Remember me). A purple "ВОЙТИ" (Login) button is positioned below the checkbox. At the bottom, there is a link "Нет аккаунта? Зарегистрироваться" (No account? Register).

Рисунок Д.1 – Функциональное тестирование методом определения классов эквивалентности, тест 1



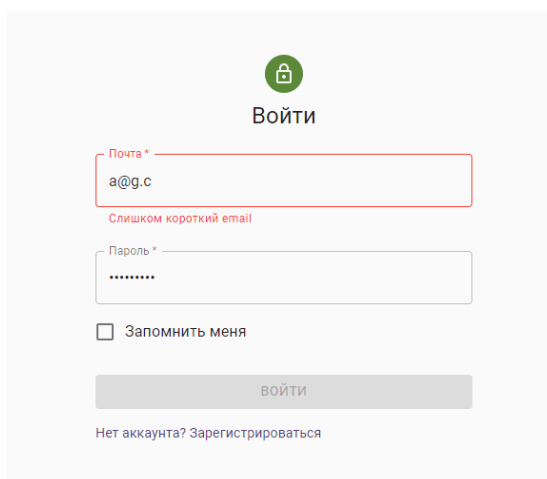
The screenshot shows the same login form, but the email field now contains "alexandr.mzv@gmail.com". A red border highlights the email field, and a red error message "Неверный формат email" (Invalid email format) is displayed below it. The "ВОЙТИ" button is now greyed out. The "Запомнить меня" checkbox and the registration link remain unchanged.

Рисунок Д.2 – Функциональное тестирование методом определения классов эквивалентности, тест 2



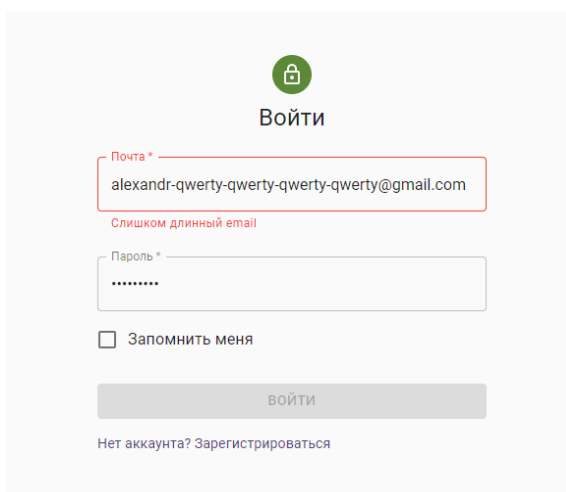
The screenshot shows the login form with the email field containing "почта_работа@mail.ru". Similar to the previous test, a red border highlights the email field, and a red error message "Неверный формат email" is shown below it. The "ВОЙТИ" button is greyed out. The "Запомнить меня" checkbox and the registration link are still present.

Рисунок Д.3 – Функциональное тестирование методом определения классов эквивалентности, тест 3



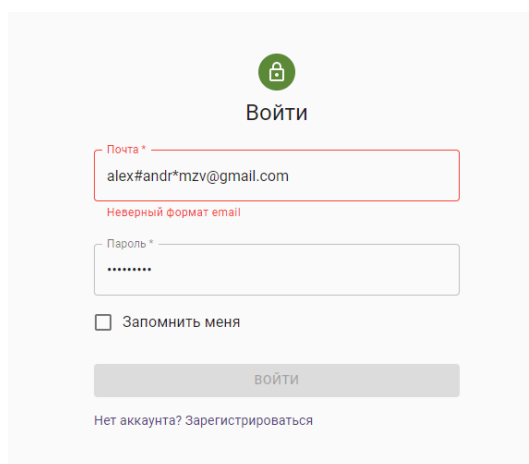
The screenshot shows a login form titled "Войти" (Login) with a green lock icon. The "Почта *" (Email) field contains "a@g.c" and is highlighted with a red border. Below it, a red error message reads "Слишком короткий email" (Email is too short). The "Пароль *" (Password) field contains masked characters. There is a checkbox for "Запомнить меня" (Remember me) and a "ВОЙТИ" (Login) button. At the bottom, a link says "Нет аккаунта? Зарегистрироваться" (No account? Register).

Рисунок Д.4 – Функциональное тестирование методом определения классов эквивалентности, тест 4



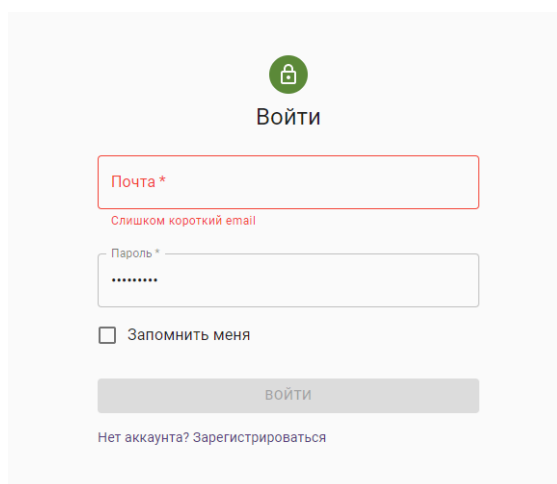
The screenshot shows the same login form. The "Почта *" field contains a long email address: "alexandr-qwerty-qwerty-qwerty-qwerty@gmail.com". It is highlighted with a red border, and a red error message below it reads "Слишком длинный email" (Email is too long). The "Пароль *" field is masked. The "Запомнить меня" checkbox and "ВОЙТИ" button are visible, along with the "Нет аккаунта? Зарегистрироваться" link at the bottom.

Рисунок Д.5 – Функциональное тестирование методом определения классов эквивалентности, тест 5



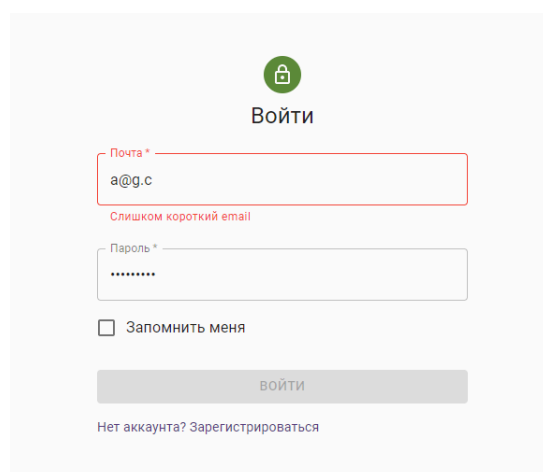
The screenshot shows the same login form. The "Почта *" field contains "alex#andr*mzv@gmail.com" and is highlighted with a red border. A red error message below it reads "Неверный формат email" (Invalid email format). The "Пароль *" field is masked. The "Запомнить меня" checkbox and "ВОЙТИ" button are visible, along with the "Нет аккаунта? Зарегистрироваться" link at the bottom.

Рисунок Д.6 – Функциональное тестирование методом определения классов эквивалентности, тест 6



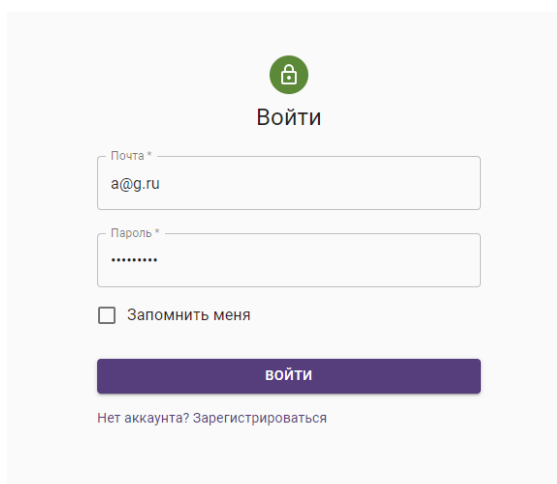
The screenshot shows a login form titled "Войти" (Login) with a green lock icon. The "Почта *" (Email) field is highlighted with a red border, and a red error message "Слишком короткий email" (Email is too short) is displayed below it. The "Пароль *" (Password) field is visible with masked characters. There is a checkbox for "Запомнить меня" (Remember me) and a grey "ВОЙТИ" (Login) button. At the bottom, there is a link "Нет аккаунта? Зарегистрироваться" (No account? Register).

Рисунок Д.7 – Функциональное тестирование методом анализа граничных условий, тест 1



This screenshot is identical to the previous one, showing the login form with the "Почта *" field highlighted in red and the error message "Слишком короткий email".

Рисунок Д.8 – Функциональное тестирование методом анализа граничных условий, тест 2



The screenshot shows the same login form, but the "ВОЙТИ" button is now purple. The "Почта *" field contains the text "a@g.ru". The "Пароль *" field is masked. The checkbox "Запомнить меня" and the link "Нет аккаунта? Зарегистрироваться" are still present.

Рисунок Д.9 – Функциональное тестирование методом анализа граничных условий, тест 3

The screenshot shows a login form with the title "Войти" (Login) and a green lock icon. The form contains two input fields: "Почта *" (Email) with the value "alexander.v.morozov.06.09.1998@gmail.com" and "Пароль *" (Password) with masked characters. Below the password field is a checkbox labeled "Запомнить меня" (Remember me). A purple button labeled "ВОЙТИ" (Login) is positioned below the checkbox. At the bottom, there is a link "Нет аккаунта? Зарегистрироваться" (No account? Register).

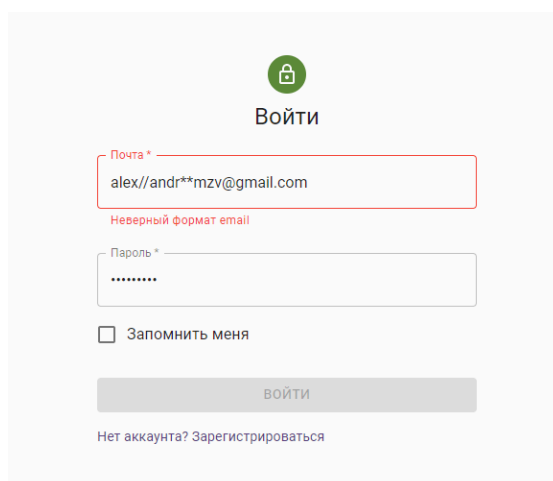
Рисунок Д.10 – Функциональное тестирование методом анализа граничных условий, тест 4

The screenshot shows the same login form as Figure D.10, but with a red border around the "Почта *" field. The email address is "alexander.valentinovich.morozov.06.09.1998@gmail.com". Below the field, a red error message reads "Слишком длинный email" (Email is too long). The "Пароль *" field is masked, and the "Запомнить меня" checkbox is present. The "ВОЙТИ" button is now greyed out. The link "Нет аккаунта? Зарегистрироваться" remains at the bottom.

Рисунок Д.11 – Функциональное тестирование методом анализа граничных условий, тест 5

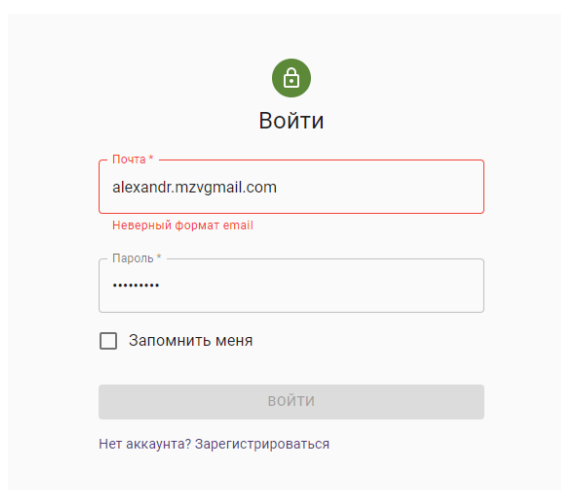
The screenshot shows the login form with the title "Войти" (Login) and a green lock icon. The "Почта *" field contains the email "alexandr.mzv@gmail.com". The "Пароль *" field is masked. The "Запомнить меня" checkbox is present. A purple button labeled "ВОЙТИ" (Login) is visible. At the bottom, there is a link "Нет аккаунта? Зарегистрироваться" (No account? Register).

Рисунок Д.12 – Функциональное тестирование методом анализа граничных условий, тест 6



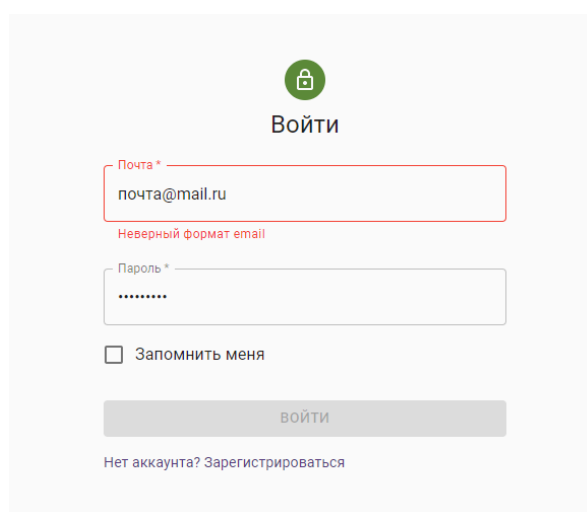
The screenshot shows a login page titled "Войти" (Login) with a green lock icon. The "Почта *" (Email) field contains "alex//andr**mzv@gmail.com" and has a red border and a red error message "Неверный формат email" (Invalid email format). The "Пароль *" (Password) field contains "*****". There is a checkbox for "Запомнить меня" (Remember me) and a "ВОЙТИ" (Login) button. At the bottom, there is a link "Нет аккаунта? Зарегистрироваться" (No account? Register).

Рисунок Д.13 – Функциональное тестирование методом анализа граничных условий, тест 7



The screenshot shows a login page titled "Войти" (Login) with a green lock icon. The "Почта *" (Email) field contains "alexandr.mzv@gmail.com" and has a red border and a red error message "Неверный формат email" (Invalid email format). The "Пароль *" (Password) field contains "*****". There is a checkbox for "Запомнить меня" (Remember me) and a "ВОЙТИ" (Login) button. At the bottom, there is a link "Нет аккаунта? Зарегистрироваться" (No account? Register).

Рисунок Д.14 – Функциональное тестирование методом анализа граничных условий, тест 8



The screenshot shows a login page titled "Войти" (Login) with a green lock icon. The "Почта *" (Email) field contains "почта@mail.ru" and has a red border and a red error message "Неверный формат email" (Invalid email format). The "Пароль *" (Password) field contains "*****". There is a checkbox for "Запомнить меня" (Remember me) and a "ВОЙТИ" (Login) button. At the bottom, there is a link "Нет аккаунта? Зарегистрироваться" (No account? Register).

Рисунок Д.15 – Функциональное тестирование методом анализа граничных условий, тест 9