МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ РАДИОФИЗИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Кафедра телекоммуникаций и информационных технологий

СИНИЦА
Владислав Игоревич

РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ТУРИСТИЧЕСКОГО СЕРВИСА

Дипломная работа

Научный руководитель:
кандидат физ.-мат. наук,
доцент Е.А. Чудовская

Допущена к защите
«___» _____ 2022 г.
Зав. кафедрой телекоммуникаций
и информационных технологий
кандидат физ.-мат. наук, доцент
_____ Ю.И. Воротницкий

Минск, 2022

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

ООП	–	Объектно-ориентированное программирование
ПО	–	Программное обеспечение
СУБД	–	Система Управления Базами Данных
API	–	Application Programming Interface
DTO	–	Data Transfer Object
HTTP	–	Hyper Text Transfer Protocol
IDE	–	Integrated development environment
JSON	–	JavaScript Object Notation
JWS	–	JSON Web Signature
JWT	–	JSON Web Tokens
MVC	–	Model View Controller
MVP	–	Minimal Viable Product
REST	–	Representation State Transfer
SQL	–	Structured Query Language
URL	–	Uniform Resource Locator
UUID	–	Universal Unique Identifier
XML	–	Extensible Markup Language

РЕФЕРАТ

Дипломная работа: 79 с., 39 рис., 19 источников, 1 прил.

ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, КЛИЕНТ-СЕРВЕР, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, ДИЗАЙН-СИСТЕМА, API, JWT, БИЗНЕС-ЛОГИКА, БЕЗОПАСНОСТЬ ДОСТУПА

Объект исследования – автоматизированные информационные среды приложения AeroPlan на базе мобильных операционных систем, а также серверное взаимодействие с интегрируемыми сервисами.

Цель работы – исследование методов и технологий разработки высококачественного программного обеспечения и применение полученных знаний при проектировании и реализации серверной части приложения и сторонних взаимодействий системы.

В процессе выполнения дипломной работы были выделены основные технические и архитектурные требования к системе, согласно которым производилось дальнейшее проектирование, разработка, а также тестирование высоконагруженного программного обеспечения сервиса AeroPlan.

Также была представлена визуальная и программная реализация основной функциональности приложения, которая выполняется посредством современных и наиболее актуальных технологий разработки.

Глубокий анализ в сфере безопасности, позволил реализовать современные подходы защиты пользовательской информации от несанкционированного доступа и последующего хищения данных. Были проведены действия над увеличением производительности и отказоустойчивости приложения, что поможет обеспечить бесперебойную работу сервиса.

Таким образом, удалось создать уникальный сервис предоставления услуг, целью которого является стать полезным инструментом в руках любого пользователя, планирующего свою поездку.

РЭФЕРАТ

Дыпломная работа: 79 с., 39 мал., 19 крыніц, 1 дадатак.

ЖЫЦЦЁВЫ ЦЫКЛ ПРАГРАМНАГА ЗАБЕСПЯЧЭННЯ, КЛІЕНТ-СЕРВЕР, МІКРАСЭРВІСНАЯ АРХІТЭКТУРА, ДЫЗАЙН-СІСТЭМА, API, JWT, БІЗНЕС-ЛОГІКА, БЯСПЕКА ДОСТУПУ

Аб’ект даследвання – аўтаматызаваныя інфармацыйныя асяроддзі дадатку AeroPlan на базе мабільных аперацыйных сістэм, а таксама сервернае узаемадзеянне з інтэгравanымі сэрвісамі.

Мэта работы – даследаванне метадаў і тэхналогій распрацоўкі высакаякаснага праграмнага забеспячэння і прымяненне атрыманых ведаў пры праектаванні і рэализацыі сервернай часткі дадатку і іншых узаемадзеянняў сістэмы.

У працэсе выканання дыпломнай работы былі выдзелены асноўныя тэхнічныя і архітэктурныя патрабаванні к сістэме, паводле якіх праводзілася далейшае праектаванне, распрацоўка і тэсціраванне высоканагружанага праграмнага забеспячэння сэрвісу AeroPlan.

Таксама была прадстаўлена візуальная і праграмная рэалізацыя асноўнай функцыянальнасці дадатку, якая выконваецца з дапамогай сучасных і найбольш актуальных тэхналогій распрацоўкі.

Глыбокі аналіз у сферы бяспекі, дазволіў рэалізаваць сучасныя падыходы абароны карыстацкай інфармацыі ад несанкцыянаванага доступу і наступнага крадзяжу звестак. Былі праведзены дзеянні над павелічэннем прадукцыйнасці і адмоваўстойлівасці прыкладання, што дапаможа забяспечыць бесперабойную працу сэрвісу.

Такім чынам, атрымалася стварыць унікальны сэрвіс падавання паслуг, мэтай якога з'яўляецца стаць карыснай прыладай у руках любога карыстальніка, які плануе сваю паездку.

ABSTRACT

Thesis: 79 pages, 39 drawings, 19 sources, 1 appendix.

SOFTWARE LIFECYCLE, CLIENT-SERVER, MICROSERVICE ARCHITECTURE, DESIGN SYSTEM, API, JWT, BUSINESS LOGIC, ACCESS SECURITY

The object of research – automated information environments of AeroPlan application based on mobile operating systems and also server interaction with integrable services.

Objective – researching methods and technologies for developing high-quality software and applying this knowledge to the design and implementation of the back-end part of the application and also external system interactions.

During the thesis, the main technical and architectural requirements for the system were identified, according to which further design, development and testing of high-load software of the AeroPlan service was carried out.

A visual and software implementation of the main functionality of the application was also presented, which is performed through modern and most current development technologies.

In-depth analysis in the field of security made it possible to implement modern approaches to protect user information from unauthorized access and subsequent data theft. Actions were taken to increase the performance and fault tolerance of the application, which will help ensure the uninterrupted operation of the service.

Thus, a unique service was created with order to become a useful tool in the hands of any user planning his trip.

ВВЕДЕНИЕ

На сегодняшний день туризм представляет собой огромный рынок услуг по перевозкам, экскурсиям и различного рода развлечениям. И несмотря на то, что убытки мирового туризма по данным ООН за последние пару лет насчитывают несколько триллионов долларов, нет сомнений, что после выхода из кризиса, страны вновь откроют свои границы и на курорты поедут туристы. И тем из них, которые не обращаются за услугами туристических компаний, необходима помощь в планировании своих поездок [1].

Исходя из этого нами было принято решение о создании такого универсального мобильного приложения туристических предложений и услуг AeroPlan, который станет для пользователей настоящим путеводителем и главным помощником в путешествии по всему миру.

Сервис AeroPlan предлагает туристам самостоятельно создать свое путешествие, найти билеты, отели, сориентироваться по ценам на еду, передвижение и экскурсии, узнать итоговую стоимость поездки и приобрести ее полностью или по частям. А в будущем, позволит общаться с другими туристами и делиться с ними своими вариантами путешествий.

Приложение AeroPlan – это настоящий start up проект, идея и реализация которого, включая название, дизайн и структуру всего проекта, является нашим оригинальным решением. Целью проекта является достижение продуктом высокого рейтинга в мобильной туристической индустрии, завоевание доверия и симпатий пользователей, а также постоянное развития функционала приложения.

В рамках дипломной работы основными целями ставятся:

1) Исследование современных принципов и технологий обработки информации, проектирования высоконагруженных систем и разработки высококачественного программного обеспечения.
2) Разработка высококачественного программного обеспечения с применением стандартов и методологий, полученных в ходе профессиональной и учебной деятельности.

Задачи, решаемые в ходе дипломной работы:

1) Изучить и исследовать существующие методы получения, обработки и хранения информации с целью реализации безопасного и надежного программного обеспечения.
2) Сформировать требования, учитывающие все нюансы и особенности проектируемой системы.
3) Разработать наиболее перспективные алгоритмы, решающие задачи безопасности, производительности и отказоустойчивости системы, а также конфиденциальности, целостности и доступности данных.

4) Спроектировать архитектуру разрабатываемого программного обеспечения согласно сформулированным требованиям, а также проанализировать вероятные варианты расширения и дальнейшее развитие архитектурных решений.

5) Разработать программное обеспечение с реализацией назначенных требований. Разработать структуру и базу данных системы.

6) Заключить партнерские соглашения и получить API сервисов предоставления информации и туристических услуг.

7) Реализовать основной дизайн визуального представления и взаимодействия с ним.

8) Реализовать основную функциональность бизнес-логики приложения и протестировать внутренние и внешние взаимодействия системы.

9) Разработать внутреннюю проектную документацию. Сформировать дальнейший план сопровождения и развития проекта.

В первой главе дипломной работы рассматриваются теоретические основы исследования существующих методов и лучших практик получения, обработки и хранения информации, а также идет описание требований к проектируемой системе.

Во второй главе произведен анализ текущей архитектуры программного обеспечения согласно сформулированным требованиям, а также рассмотрен вариант текущего перехода к микросервисной архитектуре.

Третья глава посвящена целям и задачам базы данных приложения AeroPlan, представлена ее структурная модель, а также расписаны таблицы, связи и основные особенности разработанного информационного хранилища.

В четвертой главе описаны этапы заключения соглашений и подключения к API партнерских сервисов. Также в этой главе представлена дизайнерская составляющая и визуальное представление мобильного интерфейса и WEB-интерфейса, разработанных в рамках данной дипломной работы.

Пятая глава посвящена описанию всего процесса разработки, используемым технологиям и конечному результату реализованного программного обеспечения. В этой главе рассматриваются вопросы тестирования, работы продукта, его дальнейшее развитие и сопровождение.

ГЛАВА 1
ХАРАКТЕРИСТИКА ТЕХНИЧЕСКИХ РЕШЕНИЙ И АНАЛИЗ ТРЕБОВАНИЙ ПРИ РАЗРАБОТКЕ СЕРВЕРА ПРИЛОЖЕНИЯ

1.1 Основные этапы разработки программного обеспечения

Проектирование программного обеспечения представляет собой процесс построения приложений реальных размеров и практической значимости, удовлетворяющих заданным требованиям функциональности и производительности. Программирование – это один из видов деятельности, входящих в цикл разработки программного обеспечения [2].

Рассматривая сам жизненный цикл программного обеспечения, зачастую выделяют шесть основных этапов, продемонстрированных на рисунке 1.1:

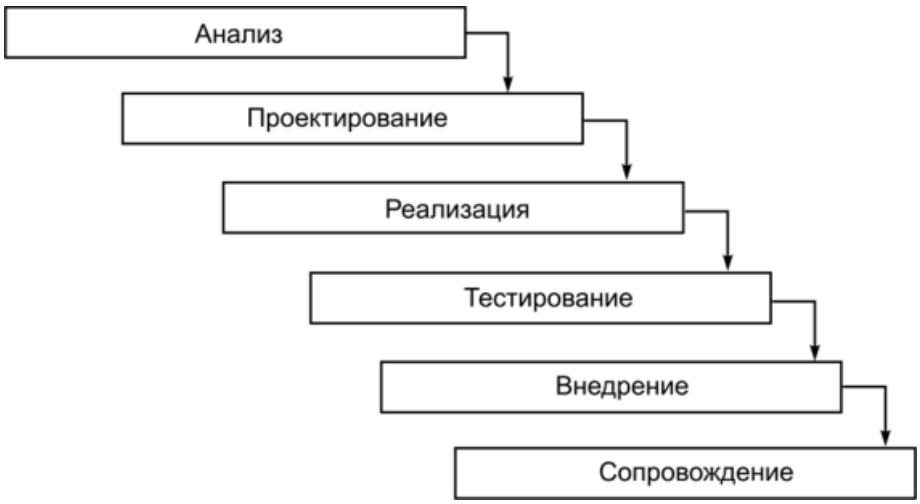

Рисунок 1.1 – Этапы жизненного цикла программного обеспечения

1) Анализ требований – это та часть процесса разработки программного обеспечения, в котором производится сбор требований как к разрабатываемому продукту, так и ко всей системе в целом. На этом этапе также производятся процессы систематизации, выявления зависимостей и документирование.

2) Проектирование – это процесс, при котором прорабатываются решения, удовлетворяющие требованиям, поставленным на предыдущем этапе. При проектировании определяются внутренние свойства системы, разрабатываются архитектура, методы и алгоритмы приложения.

3) Разработка – это процесс прямой разработки спроектированной системы. На этом этапе реализуются все поставленные задачи и идеи предшествующего этапа, с использованием лучших практик и паттернов, обеспечивающих высокую производительность и работоспособность всей системы согласно назначенным требованиям.

4) Тестирование – это процесс проверки работоспособности всех составных частей программного обеспечения с целью выявления программных дефектов и дальнейшее их устранение. На этом этапе важно тщательно проработать всю бизнес-логику приложения и все процессы, циркулирующие в системе, а также настроить автоматизированную проверку на случай дальнейшей разработки кода приложения.

5) Внедрение и эксплуатация – процесс развертывания инфраструктуры, обеспечивающей дальнейшую жизнь программного обеспечения и его взаимодействия с конечным пользователем. На этом этапе также производятся процессы продвижения продукта и его анализ.

6) Сопровождение – этап последующего развития и улучшения проекта, а также исправления дефектов, пропущенных на этапе тестирования.

При разработке программного обеспечения применялся объектно-ориентированный подход, который помогает распределить сущности проектирования и исходный код по определенным частям (модулям и классам), которые в свою очередь более конкретно отражают проблему предметной области, что эффективно подходит при наличии постоянных изменений внутри проекта.

ООП – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования [3].

1.2 Цель и сущность приложения

Целью настоящего проекта является создание сервиса, позволяющего пользователю спланировать стоимость поездки, сократить время на поиск нужной информации и получить готовый план, подходящий под бюджет путешественника, включающий в себя практически все аспекты будущей поездки, сэкономив его время и усилия.

К тому же пользователь приложения получает возможность разработать маршрут и свою собственную карту путешествия, которой в дальнейшем может делиться со своими друзьями, близкими и миллионами других путешественников по всему миру, а также сохранять и комментировать понравившиеся планы путешествий других пользователей.

Удобный интерфейс, больший спектр услуг, интерактивность, геймификация и многое другое позволят не только быстро и подробно разработать путешествие в любую точку мира, но окунуться в него уже на этапе планирования.

1.3 Анализ подходов получения данных в проектируемой области

Для предоставления всех данных и услуг конечному пользователю, получение которых является одной из задач приложения, необходим достаточно большой объем информации о разных отраслях туризма определенных областей по всему миру. Получение, анализ и хранения такого огромного количества информации является практически нерешаемой задачей и требует высокого финансирования, затрат по времени и выделенной памяти. К тому же, современные тенденции продемонстрировали скорость изменения не только туристической, но и общественной жизни по всему миру в зависимости от ситуаций, что требует постоянного обновления уже существующих данных.

Таким образом требуется решение, которое дает возможность получения актуальной информации по всем необходимым направлениям как туристической отрасли, так и общественной жизни в кратчайшие сроки без необходимости хранить данные на своих серверах.

Наиболее популярным и удобным решением этой проблемы является получения доступа к API специализированных на определенной тематике сторонних сервисов, которые предоставляют всю необходимую информацию по выделенному запросу. Большинство крупных компаний на определённом этапе разрабатывают API для клиентов или для внутреннего использования. Каждый раз, когда пользователь посещает какую-либо страницу в сети, он взаимодействует с API удалённого сервера.

Сервисы, предоставляющие данные своим клиентам, имеют надежную, отказоустойчивую систему, за счет чего нет проблем с получением информации. Однако в случае подобных проблем или же, если тот или иной сервис не имеет необходимых данных по запросу, можно заключить партнерское соглашений дополнительно еще с одним подобным сервисом. В таком случае, при возникновении исключительной ситуации одного сервиса, можно получить данные у другого и при этом без значительных затрат по времени и ресурсам.

Благодаря интеграции с сервисами предоставления услуг, собственная система получает возможность получения всех актуальных данных без необходимости их хранения на своих серверах. Такой подход значительно уменьшает сложность базы данных, бизнес-логику приложения и избавляет от лишней функциональности, не относящейся к идеи самого приложения.

Таким образом, одной из основных задач ставится заключение партнерских соглашений с подобными системами, реализация взаимодействия с ними, а также разработка бизнес-логики по получению, обработке и анализу данных. В следующих главах прилагается подробное описание архитектурного взаимодействия, а также самого процесса интеграции с подобными сервисами-партнерами.

1.4 Имеющиеся реализации и их особенности

Перед началом разработки любого программного обеспечения необходимо качественно проанализировать имеющиеся реализации подобных систем и их влияние на рынок. Если имеющиеся реализации полностью покрывают интересы клиентской базы, то конкурировать в такой среде будет крайне сложно и появляется необходимость задуматься об актуальности самой идеи.

Анализируя конкурентный рынок в области туристических мобильных приложений, нами были выделены два основных сервиса подобных услуг:

– Sygic Travel (разработчиком является компания Sygic);
– Go Trips (разработчиком является компания Google);

Приложение Go Trips в основном использует сервисы Google для поиска и создания путешествия для пользователей. Большей частью оно ориентировано на бронирование билетов, отелей и аренду автомобилей через собственные сервисы. Данное приложение не отличается большим сектором функциональности и не пользуется особо высоким спросом на европейском и азиатском рынке.

Сервис Sygic Travel является приложением-путеводителем по городам с интеграцией офлайн-карт. Оно наиболее идейно приближено к идее AeroPlan, поэтому является основным конкурентом для нас.

Наличие конкурентного рынка с одной стороны способствует стремлению постоянно развиваться и занимать лидирующие позиции в проектируемой области, а это в свою очередь означает развитие самого приложения, однако с другой стороны появляется проблема получения пользовательского внимания, что плохо может сказаться для раскрутки нового продукта.

1.5 Характеристика технического решения

Программное решение в первую очередь ориентировано на персональные мобильные устройства под операционными системами Android и IOS (находится на этапе разработки), так как они являются наиболее удобным инструментом для оперативного и постоянного взаимодействия со службой.

К тому же существует расширение на WEB-платформу (WEB-сайт) под устройства с любой операционной системой. На странице приложения можно ознакомится с основной информацией о приложении, узнать последние новости о мировом туризме, а также войти, зарегистрироваться в приложении или сбросить пароль от своего аккаунта в нем.

География проекта охватывает аудиторию любой страны мира за счет интернационализации языка, понятного интерфейса и использования всемирно известных туристических и транспортных сервисов предоставления услуг.

1.6 Требование к приложению и системе

В последнее время значительно возросло число успешных хакерских атак на пользовательские приложения. В результате таких атак похищаются не только денежные средства, но и пользовательские данные (данные банковских карт, пользовательской конфиденциальной информации). Это связано с тем, что современные приложения имеют массу уязвимостей разной степени риска.

Целью сервиса AeroPlan ставится установления строгих требований к системе, обслуживающей приложение, и безусловное их выполнение. В первую очередь особое внимание уделяется транзакциям, хранению пользовательских данных, безопасности, расширяемости и надежности самого приложения.

1.6.1 Безопасность

Под безопасностью приложений подразумеваются меры по обеспечению безопасности как на уровне системы, направленные на защиту данных от кражи, так и самого кода приложения. Эти меры затрагивают проблемы, которые могут возникнуть во время проектирования и разработки приложения, а также подходы, нацеленные на защиту системы после развертывания.

К обеспечению безопасности приложений относится использование различных процедур, оборудования и программного обеспечения для определения уязвимых мест и сведения их количества к минимуму. Обеспечение безопасности приложений включает в себя разработку, тестирование и добавление механизмов защиты на уровне приложений, направленных на предотвращение уязвимостей, делающих возможным, в частности, несанкционированный доступ к приложению и внесение изменений. О подобных уязвимостях был подготовлен доклад на 78-й научной конференциях студентов и аспирантов БГУ и опубликована в сборнике материалов.

К тому же, приложение AeroPlan размещается на облачном сервере и стоит понимать: хоть владельцы подобных хостинги и уверяют своих клиентов в высокой надежности и безопасности их систем, не всегда можно надеется на отказоустойчивость и безопасность от взломов. Облачная среда предоставляет общий доступ к ресурсам, поэтому необходимы особые меры безопасности, чтобы гарантировать пользователям конфиденциальность, целостность и доступность их персональных данных.

Таким образом основными требованиями к безопасности являются: шифрование паролей и важных данных пользователей при сохранении в базу данных и их транзакции, надежные системы аутентификации и авторизации, реализация строгого контроля доступа на сервере, обеспечение правильного баланса между онлайн и офлайн функциями приложений.

1.6.2 Техническая поддержка

Жизненный цикл мобильного приложения не ограничивается лишь подготовительным этапом сбора информации, разработкой концепта на основании предложенных идей и создания уже готового работоспособного продукта. Далее идет не менее ответственная и крайне важная стадия – техническая поддержка.

Приложение должно изменяться в соответствии с тенденциями и трендами всей индустрии, полностью отвечать всем новым требованиям пользователей. Без техподдержки любое развитие просто остановится, пользователи утратят к нему интерес, а привлекать новых клиентов будет все сложнее. Технологии настолько стремительно развиваются, что приложение достаточно быстро может устареть. И чтобы этого не произошло, очень важно поддерживать в нем актуальность – это главная задача технической поддержки.

Основными требованиями к технической поддержке являются:

– адаптация ПО для новых версий Android и IOS для качественной и бесперебойной работы на любых устройствах;

– ввод нового функционала для поддержки актуальности;

– адаптация интерфейса под актуальные предпочтения пользователей;

– налаживание способов взаимодействия с пользователями для реагирования на их информирование о существующих проблемах;

– оптимизация и модернизация существующего кода приложения.

1.6.3 Надежность

Действительно серьёзные проекты должны работать без перебоев даже в случае отказа отдельных подсистем. Причинами сбоев являются: выход из строя серверного системы, сбои программного обеспечения, аварии на уровне дата-центров. Но этих рисков можно избежать или минимизировать их последствия.

Надежность с точки зрения информационной безопасности – это исключение или минимизация рисков утечки пользовательских данных (как со стороны приложения, так и по вине самого пользователя) [4].

Отказоустойчивость приложения – это способность системы продолжать полноценно работать при выходе из строя отдельных компонентов – серверов или каналов связи, сбоев на уровне отдельных модулей системы и т.д.

Для обеспечения нормального уровня доступности не нужно создавать отказоустойчивую систему: достаточно просто качественно написанного кода приложения, адекватных процессов сопровождения и пользование услугами профессиональных хостинговых компаний, которые резервируют каналы связи, питание и охлаждение оборудования.

1.6.4 Интернационализация и локализация

Интернационализация и локализация – это процессы придания продукту свойств определенной народности, местности, расположения. Для успешной реализации продукта во все необходимые страны, нужно уделять внимание не только техническому переводу элементов интерфейса на язык страны-покупателя. Само значение слов, технологий, размещения кнопок, текстовых полей, изображений не должно хоть как-то затрагивать чью-либо религию или культуру. Не должно возникнуть проблем с восприятием конечным клиентом готового продукта. И область программного обеспечения попадает под данные правила [5].

Локализация – процесс адаптации программного продукта к языку и культуре клиента. Данный процесс адаптации включает в себя:

- перевод пользовательского интерфейса;
- перевод документации;
- контроль формата даты и времени;
- внимание к денежным единицам;
- внимание к правовым особенностям;
- раскладка клавиатуры пользователя;
- контроль символики и цветов;
- толкование текста, символов, знаков.

Кроме этого, в некоторых случаях повышенного внимания к соблюдению требований локализации, возможно влияние на проектирование приложения.

Интернационализация – более обобщенное понятие, подразумевающее проектирование и реализацию программного продукта или документации таким образом, который максимально упростит локализацию приложения.

Интернационализация включает в себя:

1) Создание продукта с учетом возможности кодировки Unicode (стандарт кодирования, поддерживающий практически все языки мира).
2) Создание в приложении возможности поддержки элементов, которые невозможно локализовать обычным образом (вертикальный текст в азиатских странах, чтение справа налево в арабских странах и т.д.).
3) Возможность загрузки локализованных элементов в будущем при желании пользователя.

Термин интернационализации не обязывает переводить текст программ или документацию на другой язык, он подразумевает разработку приложений таким образом, который сделает локализацию максимально простой и удобной, а также позволит избежать проблем при интеграции продукта для стран с отличающейся культурой.

1.6.5 Платформенные требования и расширяемость

Программное решение в первую очередь ориентировано на персональные мобильные устройства на операционной системе Android версии не ниже 6.0 (84.9 % всех устройств на базе ОС Android). В перспективах развития сервиса стоит решение о реализации IOS версии приложения. Также имеется Web-интерфейс приложения, включающий в себя вопросы аутентификации, а также лендинговую страницу с подробной информацией о приложении. В перспективе ставится задача развития WEB-клиента до полноценного сайта со всеми возможностями приложения.

Поэтому одним из основных требований является беспрепятственное расширение сервиса на различные платформы, а также обеспечение сервера приложения гибкой архитектурой, способствующей расширению в функциональном и идейном плане.

1.7 Обзор программных средств для реализации поставленных задач

В приложении AeroPlan клиентская часть реализуется под операционную систему Android в среде разработки Android Studio на языке программирования Kotlin. Серверная часть располагается на хостинге Heroku и реализуется в среде разработки IntelliJ IDEA Ultimate на языке программирования Java.

IntelliJ IDEA предлагает чрезвычайно эффективный и тщательный рефакторинг, мгновенную навигацию по коду, легкий доступ к сопоставлениям HTTP-запросов, а также простую и качественную интеграцию со сторонними сервисами и программами.

Сама разработка производится с использованием Spring Boot, Spring Data и Spring Security с помощью фреймворка автоматической сборки проекта Maven.

В качестве типа архитектурны была выбрана модель MVC (Model View Controller). Данная модель является наиболее подходящей при разработке приложения AeroPlan из-за своей смысловой концепции разделения данных, пользовательского интерфейса и управляющей логики. Остальные основные технологии и концепции будут рассмотрены более подробно в самой работе.

1.8 Основные результаты и выводы по главе

Таким образом, в первой главе были представлены этапы жизненного цикла приложения, которые являются своего рода планом данной дипломной работы, произведен анализ основных требований к серверу, выбраны основные среды и технологии разработки программного обеспечения туристического сервиса AeroPlan.

ГЛАВА 2
РЕАЛИЗАЦИЯ АРХИТЕКТУРНЫХ РЕШЕНИЙ ПРИ ПРОЕКТИРОВАНИИ ПРИЛОЖЕНИЯ

2.1 Общее описание архитектуры взаимодействия сервера

Сервер приложения AeroPlan является высоконагруженной системой, так как ему приходится взаимодействовать не только с запросами, поступающими от множества клиентских приложений, расположенных на операционных системах Android и IOS (на данный момент находится на этапе разработки), но и взаимодействовать с сервисами-партнерами для получения всей необходимой информации.

К тому же существует WEB-клиент приложения, который также на постоянной основе взаимодействует с сервером. Общая схема архитектуры приложения, а также взаимодействия сервера с клиентскими и сервисными компонентами представлена на рисунке 2.1.

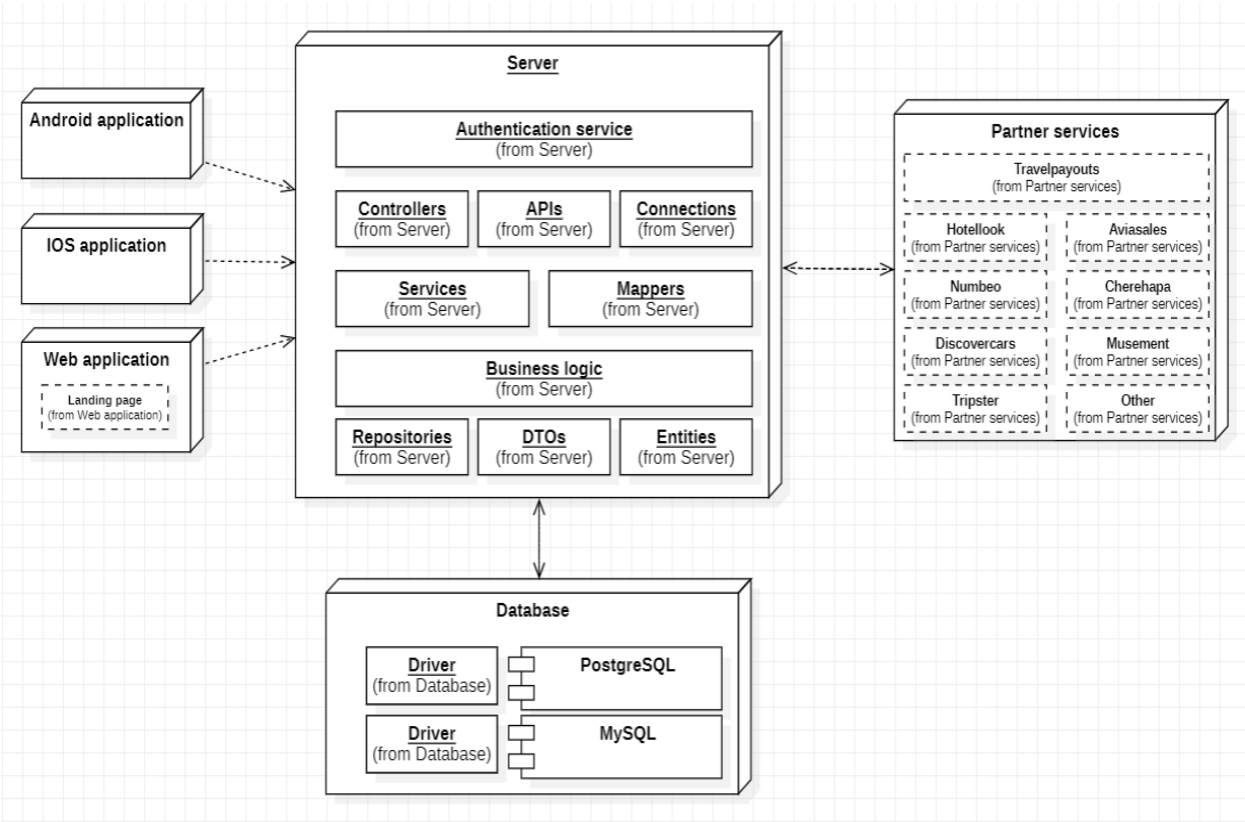

Рисунок 2.1 – Общая схема архитектуры приложения

Таким образом, внутренняя архитектура сервера приложения обязана обеспечить гарант выполнения всех требований, разработанных при анализе и начале проектировании системы, названных в предыдущей главе, а именно: безопасность, надежность, отказоустойчивость и расширяемость.

2.2 Внутренняя архитектура сервера приложения

Несмотря на большое количество взаимодействий и интеграций, созданное нами приложение AeroPlan является классическим примером приложения с клиент-серверной архитектурой. При таком типе архитектуры происходит явное разделения клиентской части, которая предоставляет интерфейс пользователям и возможность запрашивать необходимые им услуги и сервер – сердце приложения, через которое ведется централизованная настройка и управление, регулируется целостность данных и кода, соблюдается безопасность пользовательской информации [6].

Стоит обратить внимание на то, что при взаимодействии серверной части приложения AeroPlan с клиентской стороной, будь то WEB-клиент или клиенты на мобильных операционных системах, он выполняет функции сервера, как показано на рисунке 2.1. Однако, при взаимодействии сервера приложения со сторонними сервисами-партнерами, с которыми была проведена интеграция для получения данных, он сам выполняет роль клиента по отношению к ним. Наглядно такое взаимодействие показано на рисунке 2.2.

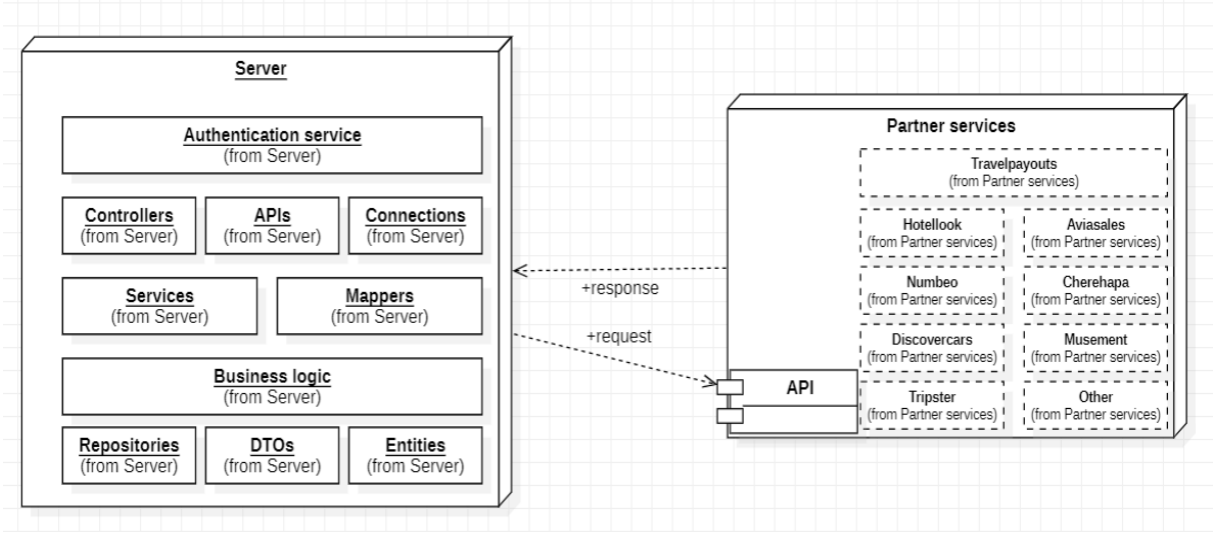

Рисунок 2.2 – Схема взаимодействия сервера с сервисами-партнерами

Таким образом, можем наблюдать случай, когда в общей архитектуре сервер также выполняет роль клиента по отношению к серверам сторонних сервисов. Такой подход является частой практикой и требует правильного и логического разделения монолитного приложения на отдельные модули. Однако, все чаще стал применятся подход, который разбивает приложение на небольшие автономные компоненты (микросервисы) с четко определенными интерфейсами. И приложение AeroPlan на пути к данной тенденции.

2.3 Переход к микросервисной архитектуре

Как стало понятно из представленной ранее модели архитектуры приложения: бизнес-логика состоит из модулей, каждый из которых представляет собой набор доменных объектов. Такими модулями в системе являются сервис аутентификации, управление путешествиями, управление предоставлением услуг пользователю и ряд других. Также есть модули, взаимодействующие с внешними системами. Некоторые из них направлены на обслуживание запросов путем обращения к бизнес-логике – это относится к REST API и пользовательскому веб-интерфейсу. Другая часть модулей бизнес-логики направлена получать доступ к базе данных и работать с различными облачными сервисами.

Такой подход не является негативным и имеет ряд преимуществ [7]:

1) Простота разработки – IDE и другие инструменты разработки сосредоточены на построении единого приложения.
2) Легкость внесения радикальных изменений – можно поменять код и структуру базы данных, а затем собрать и развернуть полученный результат.
3) Простота тестирования – есть возможность написать сквозные тесты, которые запускали бы приложение, обращались к REST API и проверяли пользовательский интерфейс.
4) Простота развертывания – разработчику достаточно было скопировать файл приложения на сервер.
5) Легкость масштабирования – простая возможность запускать несколько экземпляров приложения, размещенных за балансировщиком нагрузки.

Однако со временем разработка, тестирование и масштабирование таких систем усложняется.

Одна из проблем, которая может появится – это медленная разработка. Приходится иметь дело с замедлением ежедневных технических задач. Большое приложение перегружает и замедляет их IDE. Сборка кода занимает много времени. Более того, из-за своей величины приложение долго запускается. В итоге затягивается цикл написания, сборки, запуска и тестирования кода, что плохо сказывается на продуктивности и производительности [7].

Еще одна проблема, которая может возникнуть – это трудности с масштабированием. Такая проблема может возникнуть, так как требования к ресурсам разных программных модулей конфликтуют между собой. Поскольку модули входят в одно и то же приложение, приходится идти на компромисс при выборе серверной конфигурации [7].

Так же стоит отметить, что большой монолитной системе сложно добиться надежности приложения. В работе промышленной среды часто возникают перебои. Одна из причин – то, что из-за большого размера приложения его сложно как следует протестировать. Недостаточное тестирование означает, что ошибки попадают в итоговую версию программы. Что еще хуже, приложению не хватает локализации неисправностей, поскольку все модули выполняются внутри одного процесса. Время от времени ошибка в одном модуле (например, утечка памяти) приводит к поочередному сбою всех экземпляров системы [7].

Таким образов нами было принято решения о начале перехода к микросервисной архитектуре приложения. Такое действие поможет избежать ситуации, когда приложение разрастется до размеров, когда смена архитектуры будет означать переписывание всего проекта с нуля.

В сравнении с монолитной, микросервисная архитектура имеет следующие преимущества [7]:

1) Она делает возможными непрерывные доставку и развертывание крупных, сложных приложений.
2) Сервисы получаются небольшими и простыми в обслуживании.
3) Сервисы развертываются независимо друг от друга.
4) Сервисы масштабируются независимо друг от друга.
5) Микросервисная архитектура обеспечивает автономность команд разработчиков.
6) Она позволяет экспериментировать и внедрять новые технологии.
7) В ней лучше изолированы неполадки.

Еще одним преимуществом микросервисной архитектуры является то, что каждый сервис получается сравнительно небольшим. Разработчикам проще разобраться в таком коде. Компактная кодовая база не замедляет работу IDE, что повышает продуктивность. К тому же все сервисы запускаются намного быстрее, чем большой монолит, что тоже делает разработку более продуктивной и ускоряет развертывание [7].

Используя данный подход, была разработана микросервисная архитектура приложения AeroPlan, представленная на рисунке 2.3.

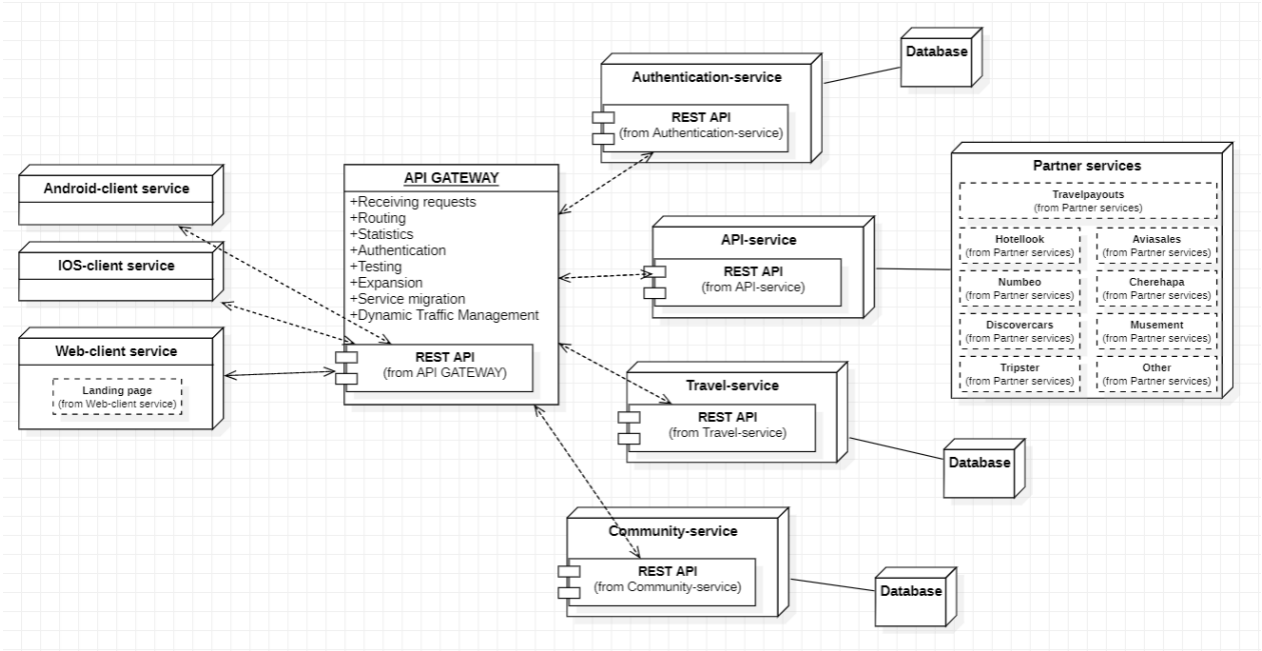

Рисунок 2.3 – Схема микросервисной архитектуры приложения AeroPlan

Таким образом, можно заметить весомые отличия между двумя типами архитектуры одного приложения. Поменялась не только структура схемы, но и полностью весь принцип взаимодействия составных частей системы.

Если при монолитной архитектуре за все процессы отвечал единый сервис, где, к примеру, бизнес-логика процесса аутентификации и логика доступа к API партнерских сервисов находились фактически в одном месте, то при микросервисной архитектуре заметно явное разделения обязанностей по конкретным, отдельным сервисным приложениям, что упрощает понимание всей системы в целом.

При таком типе архитектуры, все запросы, поступающие от клиентских приложений, приходят на сервис Gateway, который в свою очередь анализирует: идентифицирован ли пользователь в системе и не истек ли срок жизни токена, какой тип информации интересует пользователя и куда его перенаправить.

После анализа он определяет с учетом балансировки нагрузки один из микросервисов, который способен обработать запрос и перенаправляет его на этот сервис. В данном случае было выделено четыре микросервиса:

1) Authentication-service – занимается вопросами регистрации и аутентификации пользователя, сброса пароля, обновление учетных данных и токенов клиента приложения.

2) API-service – решает вопросы получения данных от сервисов-партнеров. К этому сервису могут обращаться как пользователи, так и другие сервисы самого приложения.

3) Travel-service – сервис выполняет основную бизнес-логику приложения по созданию, проектированию и редактированию пользовательских путешествий.

4) Community-service – предназначен для бизнес-логики, выполняющий функциональность взаимодействия пользователей между собой: общение, обмен путешествиями, комментирование и т.д.

Также стоит отметить, что для более быстрой и удобной работы при таком типе архитектуры, для каждого отдельного сервиса используется своя база данных. Таким образом структурная композиция базы получается более лаконичная, уменьшается избыточность и упрощается общее взаимодействие системы.

Конечно, у микросервисной архитектуры также существует ряд недостатков и проблем. К таким можно отнести [7]:

1) Сложно подобрать подходящий набор сервисов.

2) Сложность распределенных систем затрудняет разработку, тестирование и развертывание.

3) Развертывание функций, охватывающих несколько сервисов, требует тщательной координации.

Само по себе решение о том, когда следует переходить на микросервисную архитектуру, является нетривиальным. Однако такой шаг на раннем этапе позволяет решит ряд проблем, с которыми придется столкнуться команде разработчиков в будущем.

ГЛАВА 3
РАЗРАБОТКА БАЗЫ ДАННЫХ И ВЗАИМОДЕЙСТВИЙ ВНУТРЕННИХ ПРОЦЕССОВ

3.1 Цель и выполняемые задачи базы данных

Целями проектируемой базы данных являются: безопасное хранение пользовательской информации, данных о путешествиях и предпочтениях пользователей, а также иная структурная информация с соблюдением мер по обеспечении конфиденциальности, целостности и доступности информации.

Система в целом предоставляет возможность администрирования (получение, редактирование и удаление) определенных данных авторизованными пользователя имеющие разрешение (управление только своими данными).

Таким образом выполняются следующие задачи:

1) Задача учета как данных аутентификации, так и общей информации, предоставленных системе конечным пользователем путем хранения такой информации как: имя, фамилия, username, email, номера телефона и ряда остальных параметров.
2) Задача учета связей пользователя с другими пользователями в системе путем создания сообществ и членства пользователя в них.
3) Задача учета путешествий и всех составных его частей, возможность разбиения путешествия на части, а также принадлежность путешествий конечному пользователю или сообществу.
4) Задача хранения настроек привилегий на изменения данных в путешествиях, если данное путешествие принадлежит сообществу.

Также ставится задача по обеспечению нормализации базы данных с целью уменьшение избыточности данных, а также избегания проблем со связностью данных и рекурсивным извлечением при запросах. Приведение баз в нормализованный вид считается хорошей практикой и требует соблюдения как минимум трех нормальных:

1) Первая нормальная форма требует соответствия исходной таблицы требованиям, предъявляемым к отношениям [8]:
   - таблица не должна иметь повторяющихся записей;
   - все атрибуты должны быть простыми (скалярными).
2) Таблица находится во второй нормальной форме, если [8]:
   - она удовлетворяет условиям первой нормальной формы;
   - любое поле, не входящее в ключ, должно однозначно и полно идентифицироваться значением первичного ключа. Если первичный ключ является составным, то остальные поля должны зависеть от его полного выражения, а не от части.

3) Таблица находится в третьей нормальной форме, если [8]:
   - она удовлетворяет условиям второй нормальной формы;
   - ни одно из полей таблицы, не входящих в ключ, не должно идентифицироваться с помощью другого поля, не входящего в ключ.

В рамках выполнения задач и формировании структуры базы данных была выполнена ее нормализация в трех нормальных формах и частичная денормализация для оптимизации получения выборок, а также компактного и логического хранения данных.

3.2 Результирующая схема данных

Методология IDEF1X – один из подходов к семантическому моделированию данных, основанный на модели сущность-связь и позволяющий построить модель данных, эквивалентную реляционной модели в третьей нормальной форме.

Результирующая схема данных в третьей нормальной форме с частичной денормализацией в формате IDEF1X, выполненная посредством CASE систем: Star UML. Приведенные таблицы и связи изображены на рисунке 3.1.

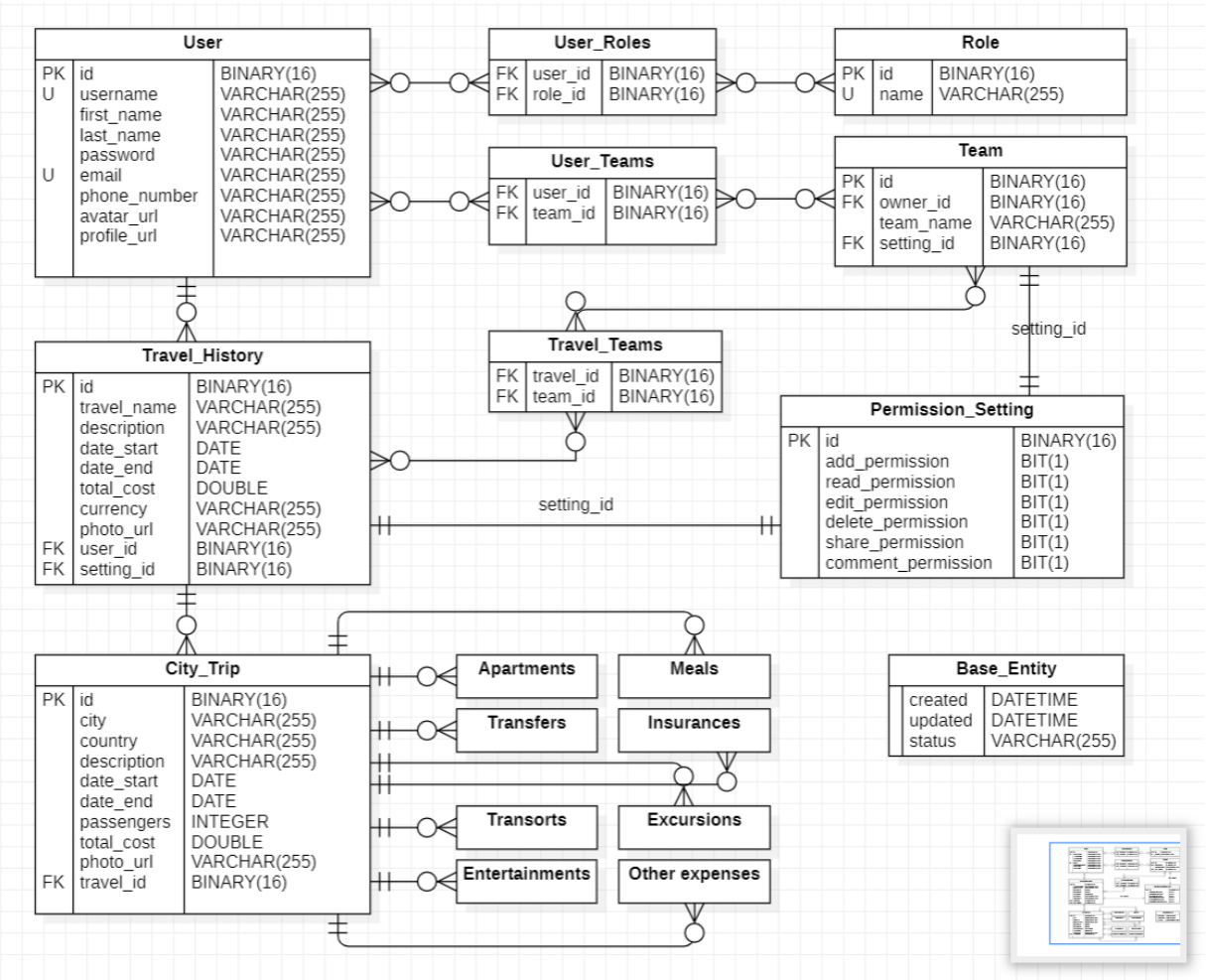

Рисунок 3.1 – Результирующая схема данных

3.3 Описание таблиц и связей

Дадим подробное описание таблиц (названия полей (атрибутов), тип данных, размеры полей, информацию о том, что будет храниться в полях и в каком виде), входящих в состав разработанной базы данных и выполняемые вышеупомянутые задачи. Также дадим описание ключей, значений по умолчанию, уникальных ограничений и проверочных ограничений (где и для чего нужны). Опишем индексацию в базе (укажем какие поля индексируются и какого типа индекс) для каждой таблицы.

3.3.1 Описание полей таблицы User

В таблице User хранится такая информации как:

1) Id (binary, 16 байт) – уникальный идентификатор пользователя в таблице выступающий в роли primary key, который хранится в формате UUID.
2) Username (varchar, 255 символов) – пользовательское имя, отображаемое в приложении. Является уникальным полем, которое хранится в виде строки (вторичный, простой и уникальный индекс).
3) First name (varchar, 255 символов) – настоящее имя пользователя для явной идентификации его другими пользователями приложения. Поле не является индексируемым и обязательным.
4) Last name (varchar, 255 символов) – настоящая фамилия пользователя для идентификации его другими пользователями приложения. Поле не является индексируемым и обязательным.
5) Password (varchar, 255 символов) – является символьным полем и служит для хранения пароля пользователя в строго зашифрованном виде. Не является обязательным, создается при регистрации пользователя.
6) Email (varchar, 255 символов) – адрес электронной почты пользователя, служит для определения пользователя в приложении и обратной связи с ним. Является уникальным полем, которое хранится в виде строки (вторичный, простой и уникальный индекс).
7) Phone number (varchar, 255 символов) – номер телефона пользователя. Не является обязательным и используется при регистрации, аутентификации и обратной связи с пользователем.
8) Avatar URL (varchar, 255 символов) – является строковым полем хранящим в себе ссылку на фотографию пользователя, отображаемой в сети и приложении. Необязательное значение.
9) Profile URL (varchar, 255 символов) – является строковым полем хранящим в себе ссылку на профиль или социальную сеть пользователя, отображаемой в сети и приложении. Необязательное значение.

3.3.2 Описание полей таблицы TravelHistory

В таблице TravelHistory хранится следующая информации:

1) Id (binary, 16 байт) – уникальный идентификатор путешествия в таблице выступающий в роли primary key, который хранится в формате UUID.

2) Travel name (varchar, 255 символов) – поле хранит название путешествия, назначенное системой автоматически или заданное пользователем. Является обязательным полем, но не уникальным.

3) Description (varchar, 255 символов) – поле содержит общее описание путешествия, назначенное пользователем. Необязательное значение.

4) Date start (date) – поле содержит в себе дату начала путешествия. Является обязательным полем, но не уникальным и не индексируемым.

5) Date end (date) – поле содержит в себе дату окончания путешествия. Является обязательным полем, но не уникальным и не индексируемым.

6) Passengers (integer) – хранит в себе значения о количестве человек, участвующих в поездке. Поле является обязательным с начальным значением 1.

7) Total cost (double) – не обязательное поле хранит в себе значение общей стоимости путешествия в валюте, указанной в поле currency.

8) Currency (varchar, 255 символов) – содержит денежное значение, в которое идет перевод всех значений при подсчете.

9) Photo URL (varchar, 255 символов) – является строковым полем хранящим в себе ссылку на фотографию или картинку, отображаемой в приложении для конкретного путешествия. Необязательное значение.

А также таблица имеет два внешних ключа:

1) User id (binary, 16 байт) – поле является foreign key с references на id в таблице User и служит для предоставления информации о том, какому пользователю принадлежит данное путешествие. При вызове будет передаваться вся сущность, а при отображении браться только необходимые данные (вторичный, простой, неуникальный индекс).

2) Setting id (binary, 16 байт) – поле является foreign key с references на id в таблице Travel Settings и служит для предоставления информации о том, какие привилегии имеют другие пользователи (кроме создателя путешествия) на администрирование данной сущностью: изменение, удаление, просмотр и прочие. При вызове будет передаваться вся сущность, а при отображении браться только необходимые данные (вторичный, простой, неуникальный индекс).

3.3.3 Описание полей таблицы CityTrip

В таблице CityTrip хранится следующая информации:

1) Id (binary, 16 байт) – уникальный идентификатор поездки в таблице выступающий в роли primary key, который хранится в формате UUID.

2) City (varchar, 255 символов) – поле хранит название города, в который была совершена поездка в рамках путешествия. Является обязательным полем, но не уникальным.

3) Country (varchar, 255 символов) – поле хранит название страны, в которой находится данный город. Является обязательным полем, но не уникальным.

4) Description (varchar, 255 символов) – поле содержит общее описание поездки в городе, назначенное пользователем. Необязательное значение.

5) Date start (date) – поле содержит в себе дату начала поездки. Является обязательным полем, но не уникальным и не индексируемым.

6) Date end (date) – поле содержит в себе дату окончания поездки. Является обязательным полем, но не уникальным и не индексируемым.

7) Passengers (integer) – хранит в себе значения о количестве человек, участвующих в поездке. Поле является обязательным с начальным значением 1.

8) Total cost (double) – не обязательное поле хранит в себе значение общей стоимости поездки в текущей валюте.

9) Photo URL (varchar, 255 символов) – является строковым полем хранящим в себе ссылку на фотографию или картинку, отображаемой в приложении для конкретной поездки. Необязательное значение.

А также таблица имеет один внешний ключ:

1) Travel id (binary, 16 байт) – поле является foreign key с references на id в таблице Travel History и служит для предоставления информации о том, к какому путешествию принадлежит данная поездка. При вызове будет передаваться вся сущность, а при отображении браться только необходимые данные (вторичный, простой, неуникальный индекс).

Особенности взаимодействия таблицы CityTrip с детализируемыми ее таблицами такими как: Apartments, Transfers, Transports, Meals, Excursions, Entertainments, Insurances и Other Expenses – будут подробно описаны в следующем разделе текущей главы, посвященной особенностям базы данных. Сейчас стоит лишь уточнить, что из значений перечисленных таблиц, складывается общая стоимость поездки в конкретный город. А сумма всех поездок, входящих в рамки одного путешествия, является обще суммой путешествия целиком.

3.3.4 Общее описание полей составляющих таблиц CityTrip

Как уже было отмечено ранее, данные и общая стоимость поездки для таблицы CityTrip получаются путем вычисления всех значений детализирующих (или, говоря иначе, составляющих путем связи One to Many) ее таблиц таких как: Apartments, Transfers, Transports, Meals, Excursions, Entertainments, Insurances и Other Expenses. Каждая из этих таблиц имеет схожий набор данных и полей, и отличающихся лишь тематикой представления и значениями.

В данных таблицах имеется общий набор одинаковых полей:

1) Id (binary, 16 байт) – уникальный идентификатор значения в таблице выступающий в роли primary key, который хранится в формате UUID.

2) Name (varchar, 255 символов) – поле хранит название значения, будь то отель, билет или страховка. Является обязательным полем, но не уникальным.

3) Description (varchar, 255 символов) – поле содержит общее описание поездки в городе, назначенное пользователем. Необязательное значение.

4) Number members (integer) – хранит в себе значения о количестве человек, участвующих в поездке. Поле является обязательным с начальным значением 1.

5) Amount (double) – поле хранит в себе значение стоимости значения в текущей валюте за одного человека.

6) Total cost (double) – поле хранит в себе значение общей стоимости значений в текущей валюте за всех участников текущей поездки.

7) Value URL (varchar, 255 символов) – является строковым полем хранящим в себе ссылку на покупку или просмотр выбранного значения, для отображения в приложении для конкретной поездки. Необязательное значение.

Также таблицы имеет один внешний ключ:

1) Trip id (binary, 16 байт) – поле является foreign key с references на id в таблице City Trip и служит для предоставления информации о том, к какой поездке принадлежит данное значение. При вызове будет передаваться вся сущность, а при отображении браться только необходимые данные (вторичный, простой, неуникальный индекс).

К тому же стоит отметить, что в таблице Apartments дополнительно имеются такие поля, как: check in и check out (означающие даты заселения и выселения соответственно), hotel stars (означающее количество звезд у отеля), а для таблиц Transports и Transfers существует поле type указывающее на принадлежность транспорта к тому или иному типу.

3.4 Описание особенностей базы данных

Как и у любого крупного проекта, у базы данных сервиса AeroPlan есть ряд особенностей, отличающих ее от других подобных решений. Для того, чтобы разработчиком было понятно назначение таковых особенностей, а дальнейшее развитее связей в базе данных происходило быстрее и эффективней, их принято описывать и документировать.

3.4.1 Назначение сущности BaseEntity

Сущность BaseEntity служит для отображения повторяющихся полей в различных таблицах. Такой подход уменьшает кодовую базу и делает отображение более лаконичным в программном плане.

В приложении повторяющимися полями были выделены следующий:

1) Created (datetime) – дата создания конкретной сущности.
2) Updated (datetime) – дата обновления конкретной сущности.
3) Status (varchar, 255 байт) – текущий статус сущности (ACTIVE, NOT_ACTIVE, DELETED).

Сущность BaseEntity применяется в приложении для таблиц User, TravelHistory, CityTrip и носит больше администраторский характер. Однако создание подобной сущности для детализирующих таблиц CityTrip посчиталось не целесообразным, так как значение этих таблиц носит характер постоянного взаимодействия с ними и явное описание полей и колонок является более уместным и удобным в данном случае.

3.4.2 Назначение таблицы Role

Таблица Role хранит в себе всего два поля:

1) Id (binary, 16 байт) – уникальный идентификатор значения в таблице выступающий в роли primary key, который хранится в формате UUID.
2) Name (varchar, 255 символов) – поле хранит название значения роли пользователя. Является обязательным и уникальным полем.

Таблицы User и Role соединены связью Many to Many, так как у одного пользователя может быть несколько ролей и, соответственно, у одной роли может бать множество пользователей. При таком взаимодействии создаётся смежная таблица, которая сопоставляет первичные значения одной таблицы, с первичными значениями второй.

В приложении роли пользователей служат для осуществления ролевой модели безопасности, авторизации пользователя в системе, назначения ему прав доступа на выполнения определенных действий.

3.4.3 Денормализация детализирующих таблиц

Под детализирующими таблицами понимаются следующие: Apartments, Transfers, Transports, Meals, Excursions, Entertainments, Insurances и Other Expenses. Такое формальное название в приложении им было дано, так как данные, содержащиеся в них, являются составными элементами (деталями) всего путешествия в целом.

Данные в таблицах служат для вычисления общей стоимости каждой поездки в конкретном путешествии. А сумма всех поездок, входящих в рамки одного путешествия, является обще суммой путешествия целиком.

Большинство колонок данных таблиц повторяется, в некоторых случая отличается лишь наименование. Однако такое логическое разделение помогает определить назначение конкретных компонентов всего путешествия и предоставляет пользователям возможность всячески менять значения в этих

таблицах, а разработчикам беспрепятственно добавлять дополнительные колонки, в случае такой необходимости и тем самым, не вредя всей системе в целом. Таким образов выполняется одно из поставленных нами требований на этапе анализа и проектирования приложения, а именно расширяемость.

3.4.4 Предварительная разработка таблиц Team и PermissionSetting

Таблицы Team и PermissionSetting не были подробно описаны в данной работе, так как функциональность, затрагивающая работу с ними еще не реализована на клиентской стороне приложения по причине больших временных затрат на разработку всей логики. В общей схеме базы данных на рисунке 3.1 представлено будущее взаимодействие с ними и описаны их поля.

Team – сущность, позволяющая объединять пользователей в отдельные сообщества (будь то бизнес команда для организации командировки или семья, собирающаяся в путешествие).

PermissionSetting – сущность, позволяющая контролировать процесс изменения, удаления и прочих действий другими пользователями по отношению к путешествию или сообществу, которое создал один из пользователь. Назначать и менять права может только такой пользователь-владелец.

3.5 Основные результаты и выводы по главе

Таким образом, правильно выстроенная структура базы данным должна служит залогом бесперебойной и качественной работы всего приложения в целом. В результате разработки были учтены основные нюансы и особенности, а также протестирована логика связей и всех взаимодействий.

ГЛАВА 4
РЕАЛИЗАЦИЯ ВЗАИМОДЕЙСТВИЙ С API СЕРВИСОВ-ПАРТНЕРОВ. ВИЗУАЛЬНОЕ ПРЕДСТАВЛЕНИЕ ОСНОВНОЙ ФУНКЦИОНАЛЬНОСТИ

4.1 Получение партнерского доступа к API туристический сервисов

На сегодняшний день практически каждый онлайн-сервис нуждается в различных данных, получаемых из сторонних источников. Те или иные данные предоставляются сервисами через API.

Аббревиатура API расшифровывается как «Application Programming Interface» (интерфейс программирования приложений, программный интерфейс приложения). Большинство крупных компаний на определённом этапе разрабатывают API для клиентов или для внутреннего использования. Каждый раз, когда пользователь посещает какую-либо страницу в сети, он взаимодействует с API удалённого сервера.

API – это составляющая часть сервера, которая получает запросы и отправляет ответы. Как и любой другой сервис, AeroPlan нуждается в сторонних API для своей работы.

Однако далеко не каждый дистрибьютор API предоставляет к нему доступ на бесплатной основе. Многие из них требуют оплату за определенное количество запросов, совершенных к серверу и аутентификацию посредством передачи в query запроса ключа аутентификации «api-key», который предоставляется пользователю API. В зависимости от количества ежемесячных запросов, цена доступа к API может сильно варьироваться.

4.1.1 Подключение сервисов

Согласно политике Travelpayouts, на сегодняшний день для подключения к сервисам-партнёрам, необходимо создать рабочий проект, описывающий: идею, название, источники трафика, ссылку на проект, географию, а также аудиторию. Выполнив эти шаги, клиент получает возможность взаимодействовать и интегрироваться с различными сервисами.

Однако, каждый сервис – отдельная организация со своими требованиями и политиками, потому в праве отказать в подключении по разным причинам, что делает процесс подключения партнерских программ довольно сложным и трудоемким.

Опишем процесс на примере сервисов, к которым мы подключались и опишем трудности, с которыми столкнулись.

4.1.2 Сервис поиска авиабилетов

Поиск авиабилетов в приложении будет осуществляться по данным, полученным с помощью сервиса Aviasales, подключение к которому представлено на рисунке 4.1. Данный сервис предоставляет широкий спектр услуг по поиску и покупке авиабилетов в различные точки мира [9].

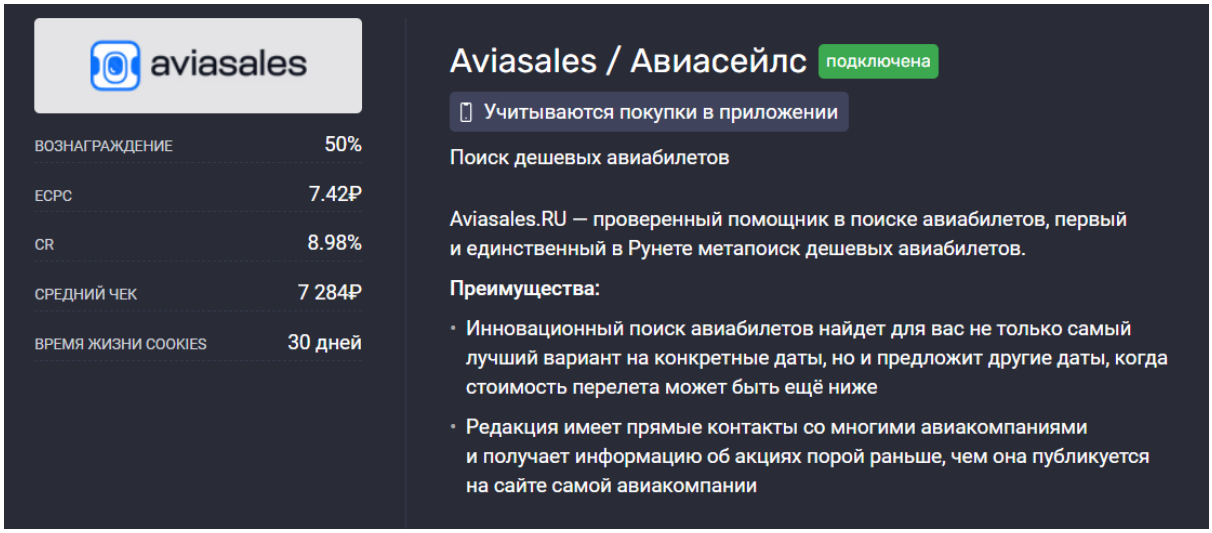

Рисунок 4.1 – Подключение к сервису Aviasales

Интегрируя с данным сервисом, наше приложение, при вводе пользователем необходимых данных (выбор способа трансфера, дат вылета и прилета, а также выбор городов, откуда и куда будут совершаться трансферы), предоставит возможные билеты по параметрам, после чего пользователь сможет как купить их, так и просто сохранить для дальнейших действий. Преимуществами является:

– инновационный поиск авиабилетов находит не только самый лучший вариант на конкретные даты, но и предлагает другие даты, когда стоимость перелета может быть ещё ниже;

– редакция имеет прямые контакты со многими авиакомпаниями и получает информацию об акциях порой раньше, чем она публикуется на сайте самой авиакомпании.

Особенности подключения. Исходя из того факта, что сервису AeroPlan необходимо получать актуальные данные об авиабилетах (билеты на определенные даты и маршруты, календарь цен, карта цен и т.д.), из имеющихся инструментов, которые предоставляют Aviasales (генератор ссылок, баннеры, поисковые формы и пр.), основным инструментом является доступ к API. Для получения доступа, было необходимо:

– рассказать об идее проекта;

– предоставить дизайн-макеты;

– пояснить, почему не подходят вышеперечисленные инструменты;

– предоставить алгоритм взаимодействия с API.

После продолжительных переговоров и выполнения вышеупомянутых необходимых условий, сотрудники поддержки Aviasales согласились дать доступ к их API, предоставив личный ключ аутентификации для доступа к данным.

4.1.3 Сервис поиска и бронирования отелей

Поиск и бронирование отелей в приложении будет осуществляться по данным, полученным с помощью сервиса Hotellook. Hotellook – сервис, позволяющий находить и сравнивать цены на отели по всему миру. Hotellook.ru создан в 2013 году командой крупнейшего в России поисковика авиабилетов Aviasales.ru [9].

Подключение к данному сервису представлено на рисунке 4.2.

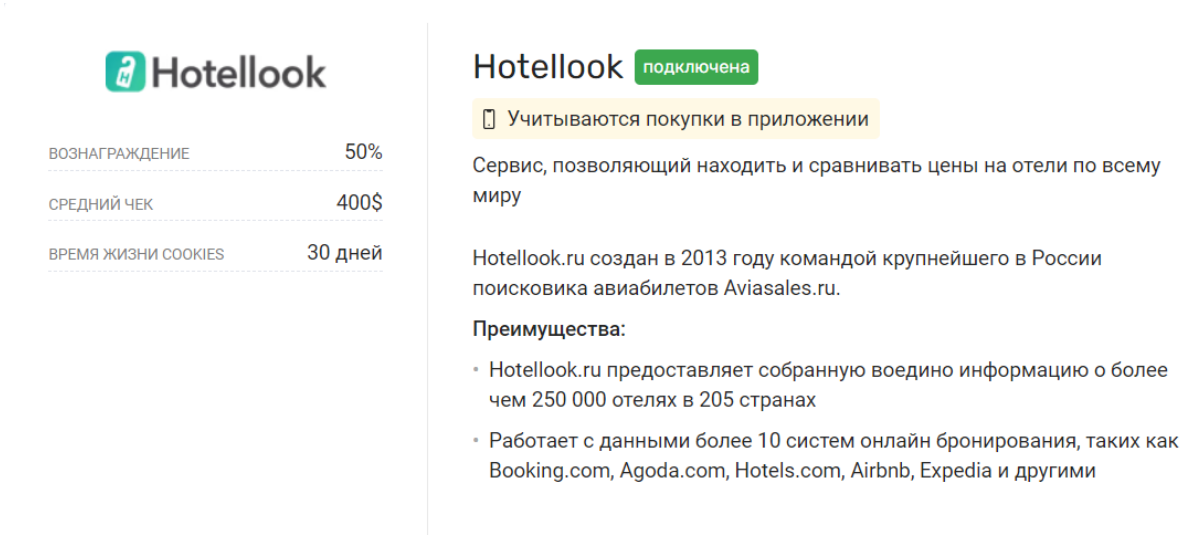

Рисунок 4.2 – Подключение к сервису Hotellook

Как и в случае с предыдущим сервисом, после ввода пользователем необходимых параметров для поиска места проживания, приложение, взаимодействуя с сервисом Hotellook предложит клиенту наилучшие варианты, после чего тот сможет выбрать из всех подходящий и далее либо забронировать, либо сохранить для дальнейших действий.

Преимущества:

– Hotellook.ru предоставляет собранную воедино информацию о более чем 250 000 отелях в 205 странах;

– работает с данными более 10 систем онлайн бронирования, таких как Booking.com, Agoda.com, Hotels.com, Airbnb, Expedia и другими.

Особенности подключения. Учитывая, что из инструментов, предоставляемых Hotellook (генератор ссылок, баннеры, поисковые формы и пр.) нам необходим только API данных отелей, который не требует наличия

ключа аутентификации (api-key), подключение к сервису было на уровне предоставления информации о проекте. В перспективе, когда в проекте будет организован поиск отелей, потребуется получить доступ дополнительно к API поиска отелей соответственно. Для этого потребуется провести переговоры с отделом поддержки Hotellook и предоставить данные, которые они потребуют.

4.1.4 Сервис аренды автомобилей

Аренда автомобилей в приложении будет осуществляться с помощью сервиса DiscoverCars.com. Discovercars.com – это отмеченный наградами сайт сравнения проката автомобилей (World Travel Awards, Financial Times 1000, Inc. 5000). Клиенты могут найти, сравнить и арендовать автомобиль по всему миру.

Подключение к сервису представлено на рисунке 4.3.

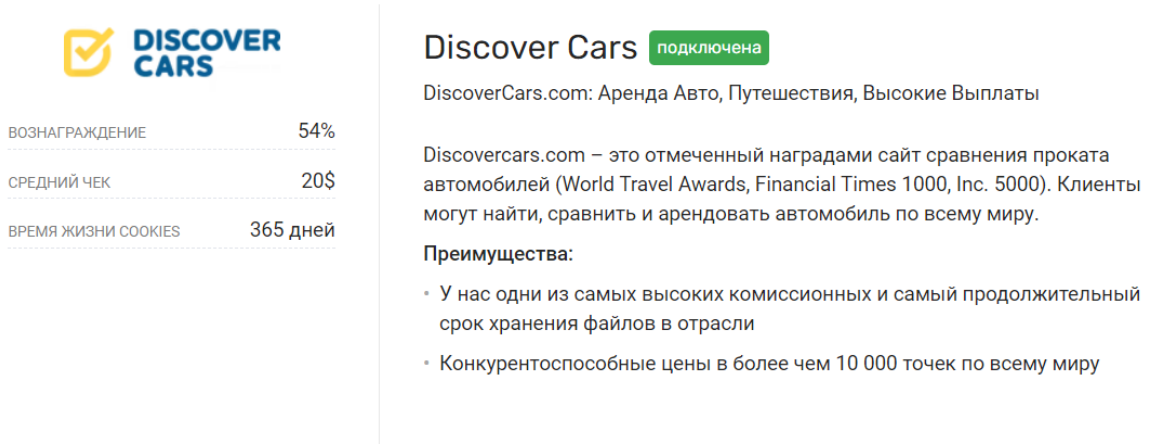

Рисунок 4.3 – Подключение к сервису Discover Cars

Преимущества:

– сервис имеет самый продолжительный срок хранения файлов в отрасли;

– конкурентоспособные цены в более чем 10 000 точек по всему миру.

Особенности подключения. Подключение на уровне предоставления доступа к проекту и информации о нем. В переговорах не было необходимости.

4.1.4 Сервис туров и мероприятий

Поиск популярных экскурсий, достопримечательностей, а также информации о мероприятиях будет происходить по средствам нескольких сервисов: Musement и Tripster. Пользователь сможет не только просматривать и выбирать экскурсии и места, которые хотел бы посетить, но и бронировать экскурсии, мероприятия и билеты для посещения достопримечательностей в любой точке мира [9].

Подключение к данным сервисам представлено на рисунке 4.4.

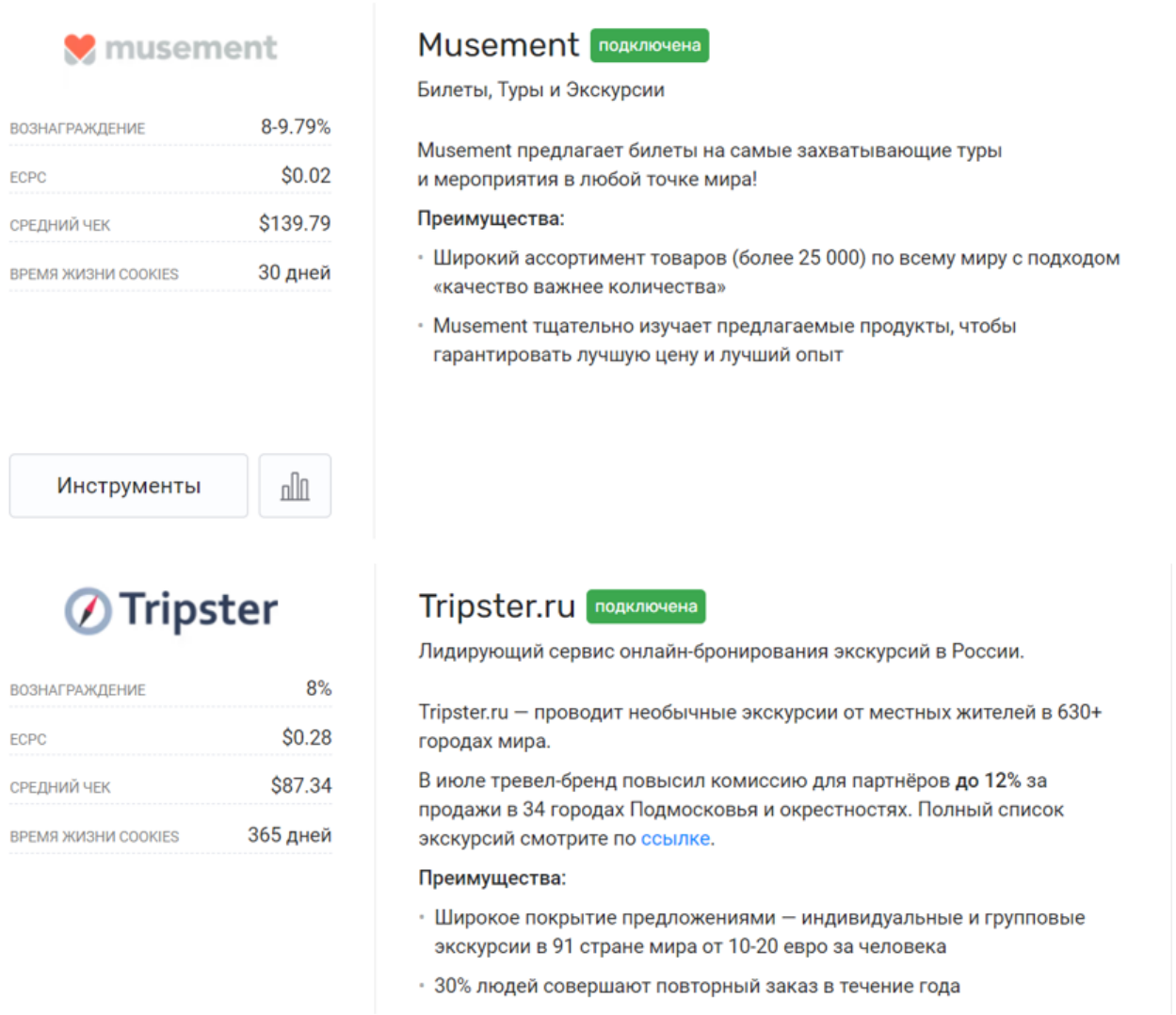

Рисунок 4.4 – Подключение к сервису Musement и Tripster

Musement предлагает билеты на самые захватывающие туры и мероприятия в любой точке мира.

Преимущества:

– широкий ассортимент товаров (более 25 000) по всему миру с подходом «качество важнее количества»;

– Musement тщательно изучает предлагаемые продукты, чтобы гарантировать лучшую цену и лучший опыт.

Tripster – лидирующий сервис онлайн-бронирования экскурсий в России. Tripster.ru – проводит необычные экскурсии от местных жителей в 630 городах мира.

Преимущества:

– широкое покрытие предложениями – индивидуальные и групповые экскурсии в 91 стране мира от 10-20 евро за человека;

– 30 % людей совершают повторный заказ в течение года.

Особенности подключения. Для двух сервисов подключение происходило на уровне предоставления доступа к проекту и информации о нем. В переговорах не было необходимости.

4.1.5 Сервис страховок

Вопросы туристического страхования будут решаться с помощью сервиса Cherehapa. Cherehapa – онлайн-сервис по сравнению и продаже туристических страховых услуг от крупнейших страховых компаний [9].

Подключение к сервису Cherehapa представлено на рисунке 4.5.

Рисунок 4.5 – Подключение к сервису Cherehapa

Преимущества:

– логика выдачи предложений пользователям нацелена на увеличение среднего чека;

– постоянное отслеживание конверсий продаж, тестирование с целью увеличения конверсий.

Особенности подключения. Для данного сервиса подключение также происходило на уровне предоставления доступа к проекту и информации о нем, а значит переговорах не было необходимости.

4.1.6 Расчет проживание на время путешествия

Расчет проживание на время путешествия будет производится бизнес-логикой самого приложения за счет получения информации о ценах на продукцию в различных городах от сервиса Numbeo [10].

Numbeo – это крупнейшая в мире база данных о стоимости жизни. Анализируя информацию о пользовательских предпочтениях, Numbeo предоставит все необходимые данные о стоимости на те или иные вещи в заданном регионе (продукты, одежду, такси и т.д.), а приложение AeroPlan подсчитает общую сумму для пользователя и сохранит ее для дальнейшего анализа и развития персонализации.

Преимущества:

– Numbeo является глобальной базой данных с информацией о качестве жизни, включая показатели жилья, предполагаемый уровень преступности и качество здравоохранения, а также многие другие статистические данные.

Особенности подключения. Чтобы подать заявку на бесплатный доступ к API в академических целях, необходимо было отправить электронное письмо с темой «Бесплатное академическое использование API» и включить следующую информацию:

- название проекта, краткое описание приложения и системы;
- университет;
- имя и электронная почта профессора;
- веб-страница отдела персонала университета;
- имя и электронная почта студента;
- продолжительность использования.

Лимит использования API академической лицензии – 10000 запросов в день. После продолжительных переговоров и выполнения вышеупомянутых необходимых условий, сотрудники поддержки Numbeo.com согласились дать доступ к их API, предоставив личный ключ аутентификации для доступа к данным.

4.2 Дизайн и описание клиентской функциональности

Напомним, что основной идеей сервиса является возможность создания пользователем своего персонального путешествия с расчетом общей стоимости. Такая возможность появляется благодаря сотрудничеству с партнерскими туристическими сервисами, процесс подключения к которым был описан ранее. Теперь пользователь, установивший приложение, имеет следующие возможности:

1) Создание своего личного путешествия.
2) Поиск и добавление в него городов из разных стран.
3) Настройка пользовательских фильтров.
4) Выбор дат и количества дней, проведенных в каждом конкретном городе.
5) Расчет стоимости путешествия с разбиением по городам.
6) Получение персональных предложений и советов.
7) Просмотр ленты новостей о путешествиях.
8) Поиск информации по городам.

Далее наглядно будет продемонстрирован пользовательский интерфейс и возможности взаимодействия с ним.

4.2.1 Понимание дизайнерской системы

Любое мобильное приложение в цифровом мире нашего времени сталкивается с жесткой конкуренцией. Чтобы выжить на рынке достаточно важно иметь запоминающийся дизайн и удобный интерфейс с возможностью простого взаимодействия с ним. Такие вещи помогут влиять на мнение клиентов и вырабатывать спрос на продаваемый продукт.

Системы проектирования описывают элементы пользовательского интерфейса, основываясь на идентификационных данных торговой марки и ее использовании, что позволяет дизайнерам и разработчикам проще работать над продуктом, а также легко понять потребителям [11].

Известно, что дизайн-система представляет собой совокупность трех сущностей:

1) Визуальный язык – та часть приложения, которую мы видим.
2) Framework – библиотека визуального языка, его код и проектирование.
3) Guidelines – набор правил, с информацией о там, как должно все выглядеть и каким образом применяться.

Таким образом, дизайн-система является отдельным и очень важным продуктом внутри любого IT-продукта. Рассмотрим подробнее названные ранее сущности.

4.2.2 Визуальный язык

Визуальная составляющая дизайн-системы включает в себя множество элементов, основные из которых: цвета, шрифты, формы объектов, иконки, изображения, взаимодействия, анимации, UI-компоненты, звуки.

Цвет и шрифт помогают приложению стать более заметным в глазах пользователей. Есть возможность создать или приобрести (как в случае приложения AeroPlan) свой собственный шрифт, который будет выделять данное приложение из общей массы и делать его более оригинальным.

Формы объектов и иконки также помогают приложению стать более заметным на рынке. Зачастую, к данным и прочим объектам накладывается анимация, что придает большую динамику и интерактивность приложению. Такая динамика иконок и UI-компонентов (элементы интерфейса приложения на ОС Android), была реализована как на WEB-сайте, так и в самом приложении AeroPlan.

Задача визуального языка – передать ценности бренда потребителю. Визуальный язык является одним из факторов того, каким пользователи продукта запоминают тот или иной продукт [11].

4.2.3 Фреймворк

Фреймворк – это совокупность библиотек и кода для реализации возможностей визуального языка. Благодаря нему все визуальные элементы можно взять из библиотеки и легко применить во всех своих продуктах.

Фреймворки гораздо упрощают разработку. Нет необходимости множество раз реализовывать один и тот же функционал. Зачастую стандартные решения Фреймворки уже содержат в себе и стоит лишь указать на то, что хотите сделать. Со стороны разработчиков сам код приложения становиться короче, более гибким и понятным, что также упрощает этап разработки, тестирования и поддержки. Также стоит отметит богатые документации по уже готовым решениям. Таким образом, Фреймворк является централизованной библиотекой элементов визуального языка, которая позволит создавать и использовать повторно каждую деталь продукта [11].

4.2.4 Guidelines

Для того, чтобы избежать трудностей во взаимодействии с дизайнерскими системами, необходимо вырабатывать структурированные правила.

Такие правила применения элементов дизайн-системы позволяют упорядочить процесс работы для всех. Они задают определенные рамки и формируют понимание, что любое изменение дизайн-системы может затронуть множество экранов и сценариев взаимодействия пользователя в продукте. [11].

4.2.5 Назначение дизайн-систем

Можно выделить как минимум пять основных задач, которые выполняет любая дизайн-система в каком-либо приложении:

1) Автоматизация.
2) Итеративность.
3) Консистентность.
4) Синхронизация.
5) Прототипирование.

Автоматизация определенно является ключевым мотивом использования подобных систем. Дизайн-система позволит автоматизировать процессы и выиграть время на другие задачи. Например, дизайнеры смогут сформировать библиотеку компонентов в графическом редакторе Figma и применять их во всех необходимых макетах. Разработчики, в свою очередь, просто используют ранее реализованные компоненты из фреймворка [11].

Итеративность определяет возможность наследования и автоматического применения изменений в элементах дизайн-системы, что является мощным инструментом для продуктовой команды. Так имеется возможность простого внедрения косметических изменений или добавление дополнительной функциональности в конкретном месте. Таким образом шаг за шагом можно улучшать продукт без серьезных последствий.

Но следует и понимать, что любое изменение в системе несет в себе определенный риск, что является большой ответственностью для любого

разработчика. Поэтому любая коррекция должна быть сознательной и аргументированной [11].

Консистентность говорит о том, что объекты и сущности, которые мы используем в продукте и которые выполняют одну и ту же работу, должны стремиться к унификации и идентичности.

Пользуясь в продукте идентично реализованными функциями (даже с разной функциональной задумкой), они становятся привычными и удобными: происходит интуитивное понимание того, как работает и выглядит тот или иной объект, его расположение и назначение [11].

Все элементы в IT-продукте должны обладать единым языком общения на основе четких правил. С ними новые члены продуктовой команды выполняют задачи в рамках продукта, не теряя времени на разбор чужих реализаций [11].

Также дизайн-система позволяет быстро собрать прототип из готовых элементов и проверить идейную задумку разработчика за счет своей наглядности и простоты реализации. Это, в свою очередь, экономит бюджет проекта и дает быстрые результаты.

4.2.6 UI / UX-дизайн мобильного приложения

Эта ценность является, по сути, совокупностью всех предыдущих преимуществ. Ручная работа с визуальным языком, на которую раньше тратилось много времени, становится максимально автоматизированной.

UI / UX-дизайн мобильного приложения играет ключевую роль в успехе мобильного приложения. Аббревиатура UI обозначает пользовательский интерфейс. Это способ взаимодействия пользователей с мобильным приложением. Пользовательский интерфейс включает в себя все элементы управления, кнопки, блоки и элементы приложения.

Основная цель пользовательского интерфейса – обеспечить легкое, приятное и эффективное взаимодействие между пользователем и приложением. Процесс разработки UI включает в себя выбор цвета, фирменного стиля и последних принципов дизайна.

После создания дизайн-системы можно выделить больше времени на исследование и детальное проектирование пользовательских сценариев продукта.

4.2.7 Недостатки дизайн-системы

Дизайн-система имеет и свои минусы.

Во-первых, дизайн-система – это дорого. Нужны ресурсы для ее создания и непрерывной поддержки. Дизайн-система должна развиваться вместе с продуктом. В противном случае между ними возникнут отличия: продукт будет развиваться, а дизайн-система будет постепенно отставать от продукта и терять свою актуальность.

Если бренд сформулировал новые правила визуального и вербального взаимодействия с клиентами, то визуальный язык дизайн-системы необходимо адаптировать. Дизайн-система должна идти в ногу с бизнесом и нести правильную идеологию бренда, развиваясь вместе с ним [11].

Однако хочется отметить, что поддержка дизайн-системы в какой-то степени упрощается благодаря использованию современных инструментов, например, редактора Figma. У него достаточно функционала, чтобы дизайнеры могли с комфортом формировать и развивать UI-kit.

Проектирование дизайна сервиса AeroPlan осуществлялось в приложении Figma, которое является онлайн-сервисом для разработки интерфейсов и прототипирования с возможностью организации совместной работы в режиме реального времени.

4.2.8 Дизайн-система приложения AeroPlan

Исходя из вышеупомянутых аргументов в пользу создания дизайн системы, было принято решение создать таковую для сервиса AeroPlan. Для разработки использовался сервис Figma UI Kit и его инструменты. При помощи этих инструментов были продуманы следующие аспекты дизайн системы: типографика представлена на рисунке 4.6, цветовая палитра на рисунке 4.7, модальные окна и основные компоненты представлены на рисунке 4.8, из которых проектировались экраны мобильного приложения.

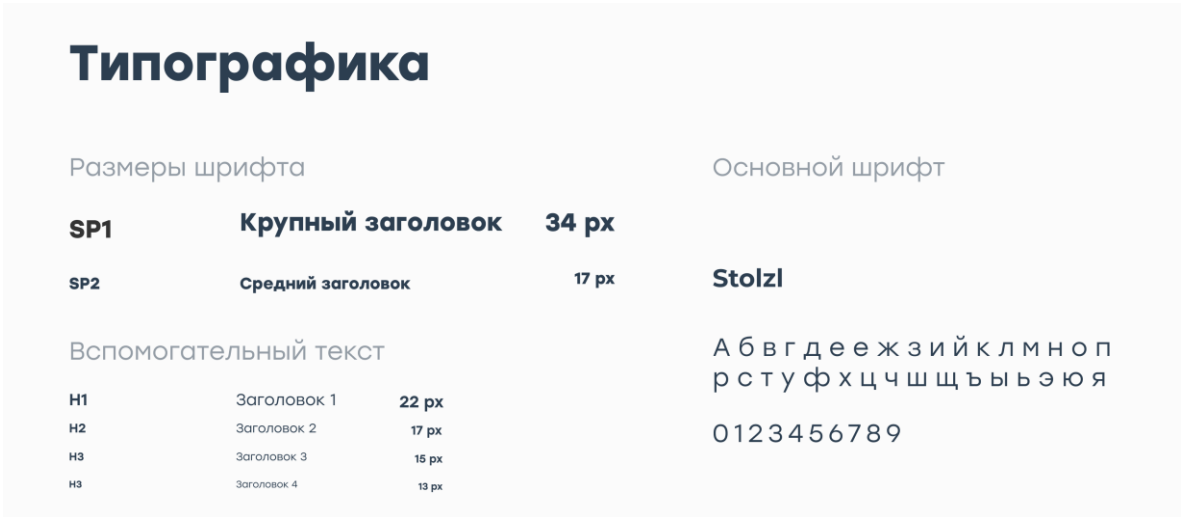

Рисунок 4.6 – Типографика

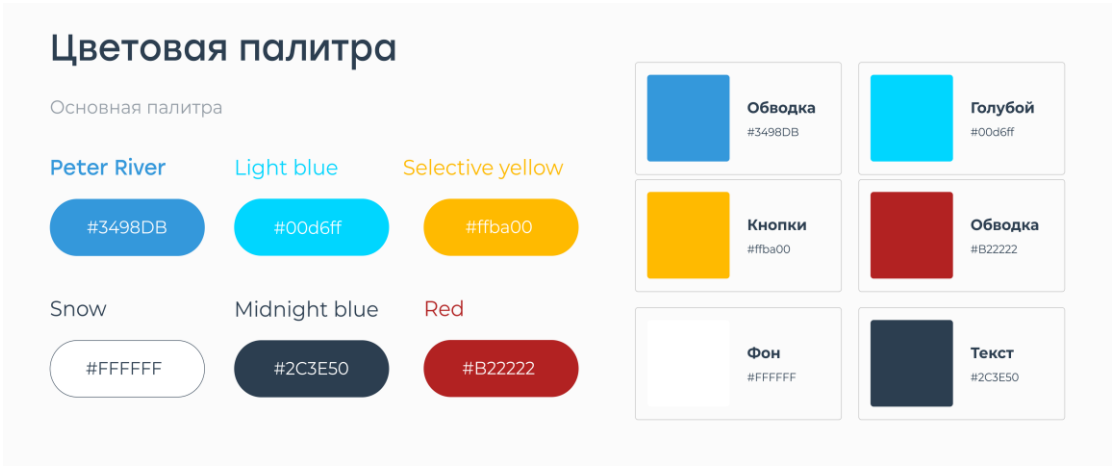

Рисунок 4.7 – Цветовая палитра

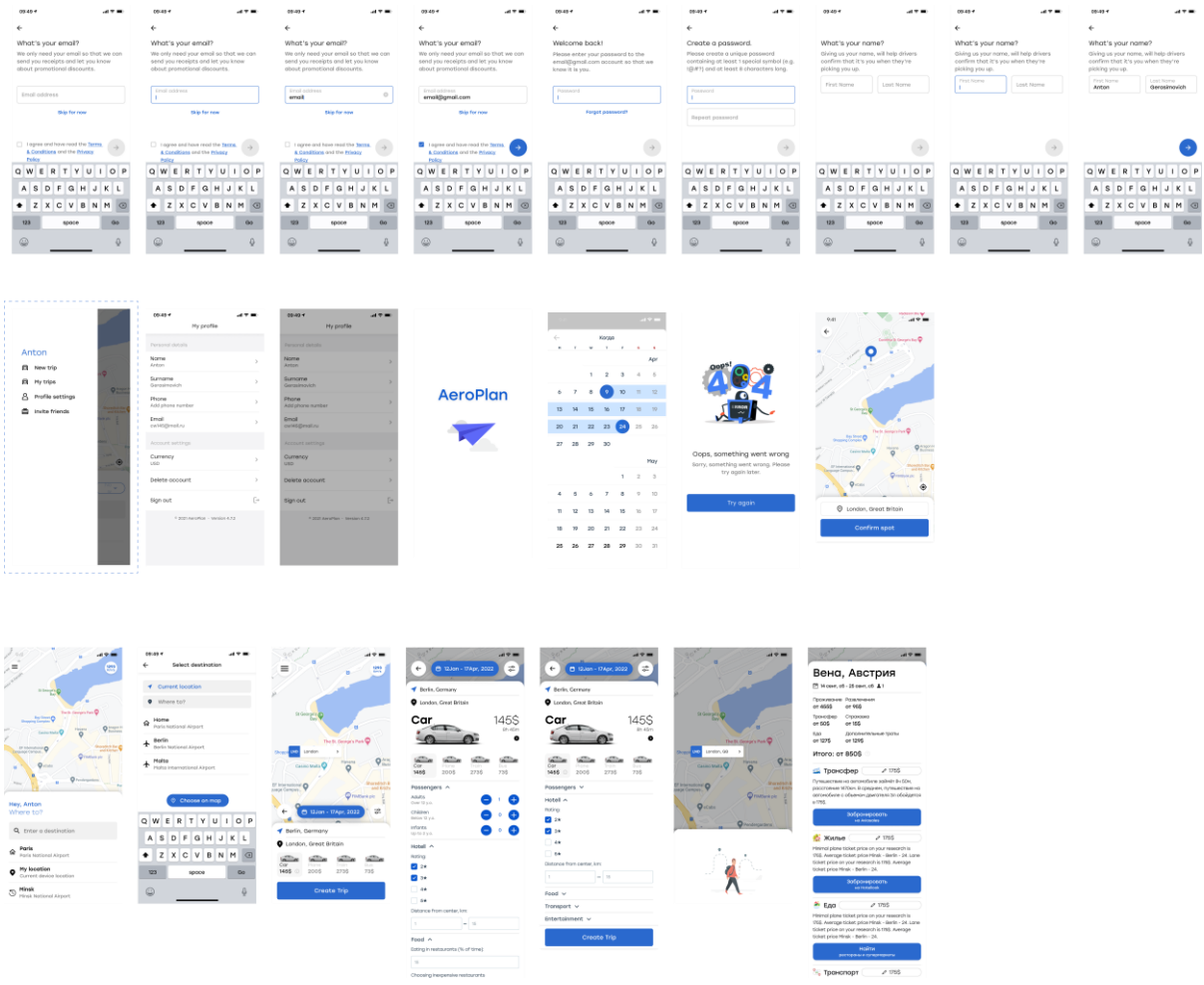

Рисунок 4.8 – Основные компоненты приложения

Создав дизайн систему, мы перешли к проектированию экранов мобильного приложения, о чем подробно будет описано в последующих пунктах работы.

4.3 Визуальное представление основной функциональности приложения

Визуальное представление любого приложения является основополагающим вопросом успеха того или иного проекта на рынке. Создание дизайнерской системы является, конечно, необходимым условием реализации качественного продукта, но далеко не достаточным.

Таким образом, в данном разделе опишем спроектированные экраны мобильного приложения и подробно назовем функционал каждого. Будут рассмотрены экраны следующих разделов приложения:

1) Экран регистрации и аутентификации.
2) Главный экран приложения.
3) Раздел «Мои путешествия».
4) Разделы созданных путешествий.
5) Web-интерфейс приложения.

Web-интерфейс приложения реализован на серверной стороне системы и служит в качестве вспомогательной системы для просмотра актуальной информации на сайте (реализация полноценного Web-клиента планируется в перспективе), регистрация, аутентификация и сброс пароля в случае его утери.

Web-интерфейс является кроссплатформенным решением взаимодействия с приложением AeroPlan, работающим через браузер любого устройства.

4.3.1 Регистрация и аутентификация

По умолчанию для лучшего пользовательского опыта приложение не просит пользователя зарегистрироваться или войти в свой аккаунт, оставляя за ним возможность моментально перейти к процессу планирования путешествия. Однако, если он захочет пользоваться сервисом на нескольких устройствах или, например, делиться своими путешествиями, он должен создать свой аккаунт. Соответствующую кнопку можно найти в меню.

Для успешного создания аккаунта нужно совершить несколько определенных шагов. На рисунках 4.9 и 4.10 показаны экраны регистрации.

1) Запрос на ввод адреса электронной почты.
2) Запрос на ввод пароля.
3) Запрос на ввод имени и фамилии (не является обязательным).
4) Запрос на ввод номера телефона (не является обязательным). При вводе номера телефона на него будет отправлен смс-код, подтверждающий принадлежность этого номера конкретному пользователю. Этот код необходимо ввести в соответствующую форму.

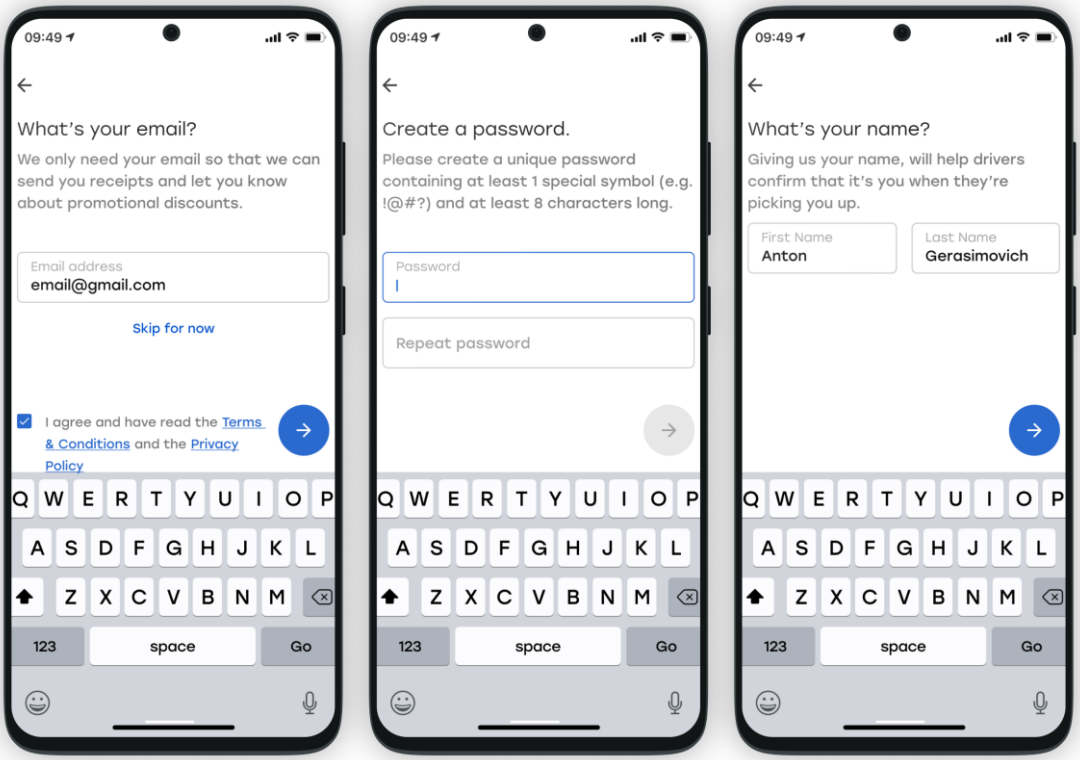

Рисунок 4.9 – Экран регистрации по username/password

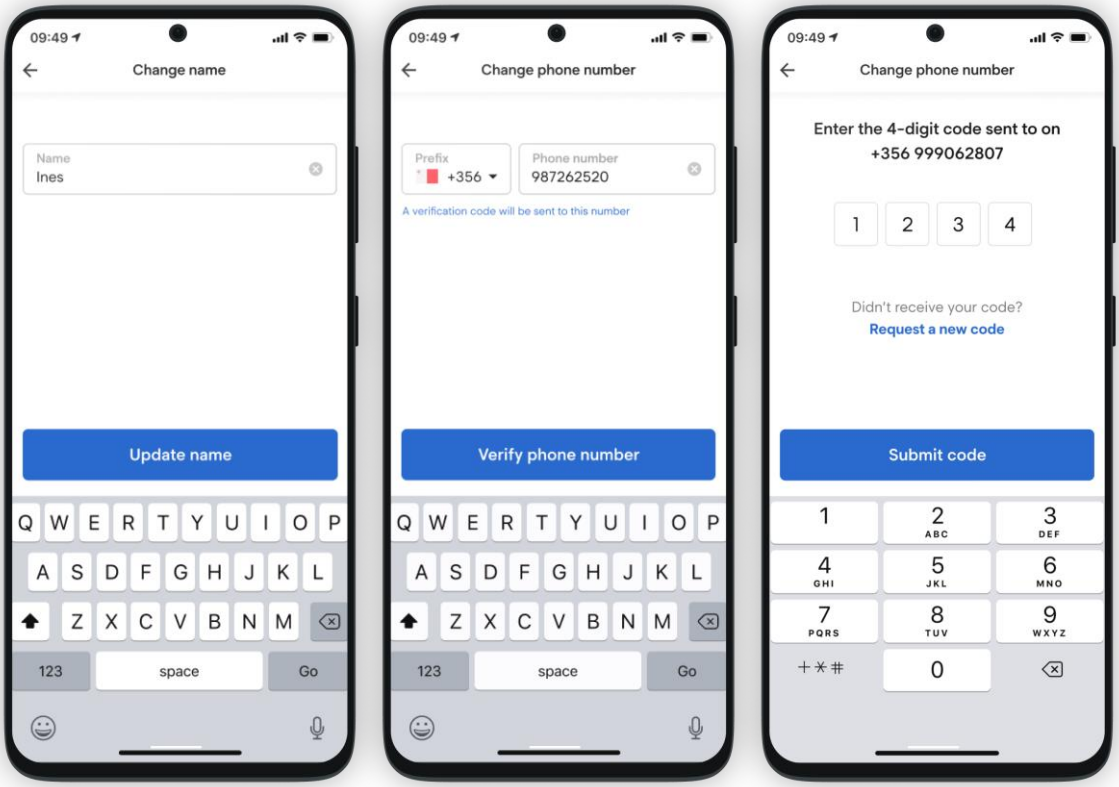

Рисунок 4.10 – Экран регистрации и проверка по номеру телефона

4.3.2 Главная страница приложения

При входе в приложение, пользователя встречает главный экран, показанный на рисунке 4.11, позволяющий моментально создать путешествие путем нажатия на один из предложенных городов, релевантных по отношению к текущей геолокации пользователя. Туристу будут предложены либо популярные направления, либо его предыдущие запросы поиска, сохраненные в локальной базе данных. Нажатие на любой из них дает возможность пропустить несколько шагов и практически сразу перейти к выбору способа передвижения и настройке создаваемого путешествия.

Боковое меню, продемонстрированное на рисунке 4.11 справа, открывается нажатием на кнопку в верхнем левом углу. В нем находятся кнопки навигации к следующим разделам сервиса: создание/редактирование аккаунта, создание путешествия, мои путешествия, пригласить друзей, дополнительная информация.

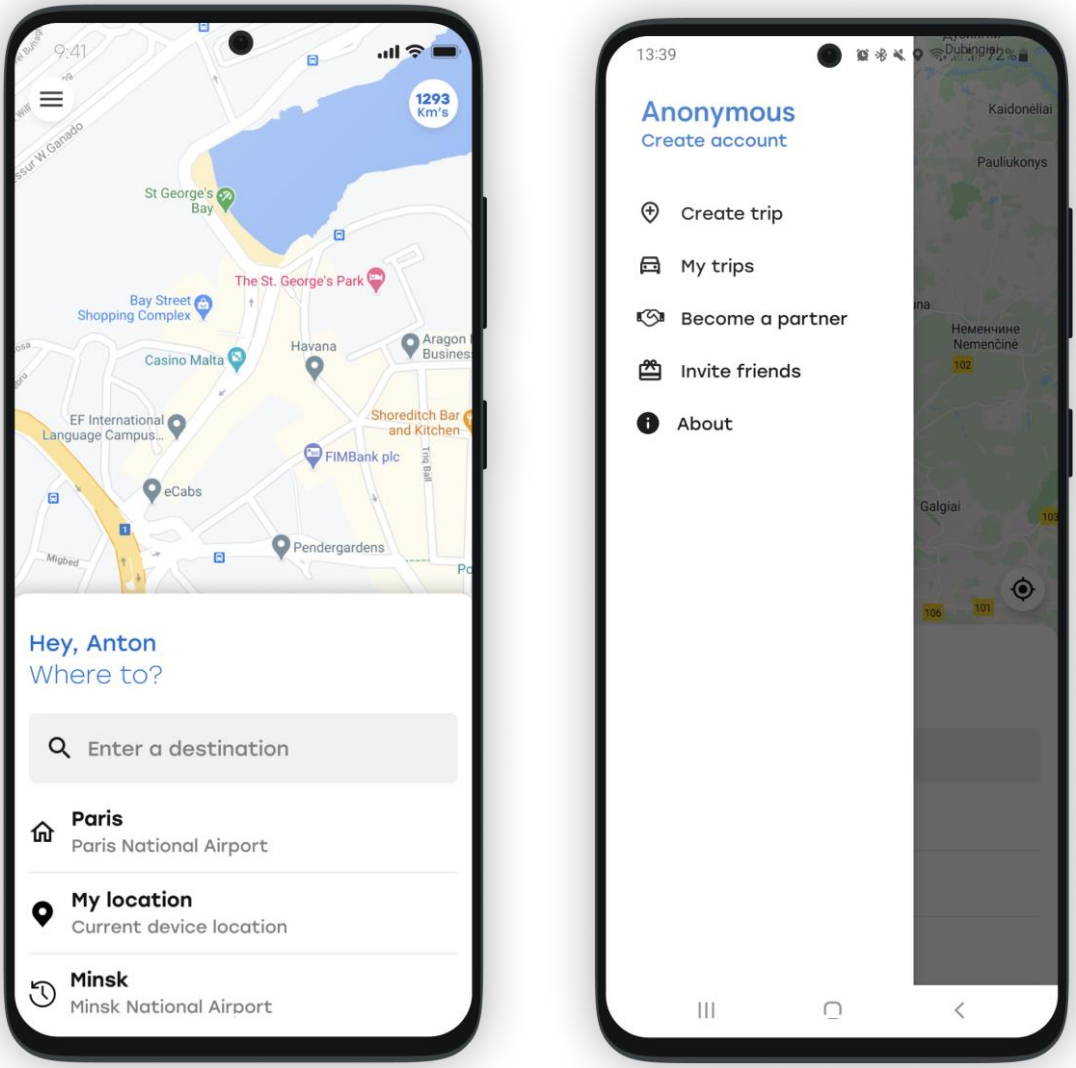

Рисунок 4.11 – Главный экран приложения

4.3.3 Раздел «Мои путешествия»

В разделе «Мои путешествия», экраны которого представлен на рисунке 4.12, пользователь может найти свои уже созданные и сохраненные путешествия, открыть любое из них, а также создать новое. Также у пользователя есть возможность редактировать все составные компоненты путешествия и удалять их.

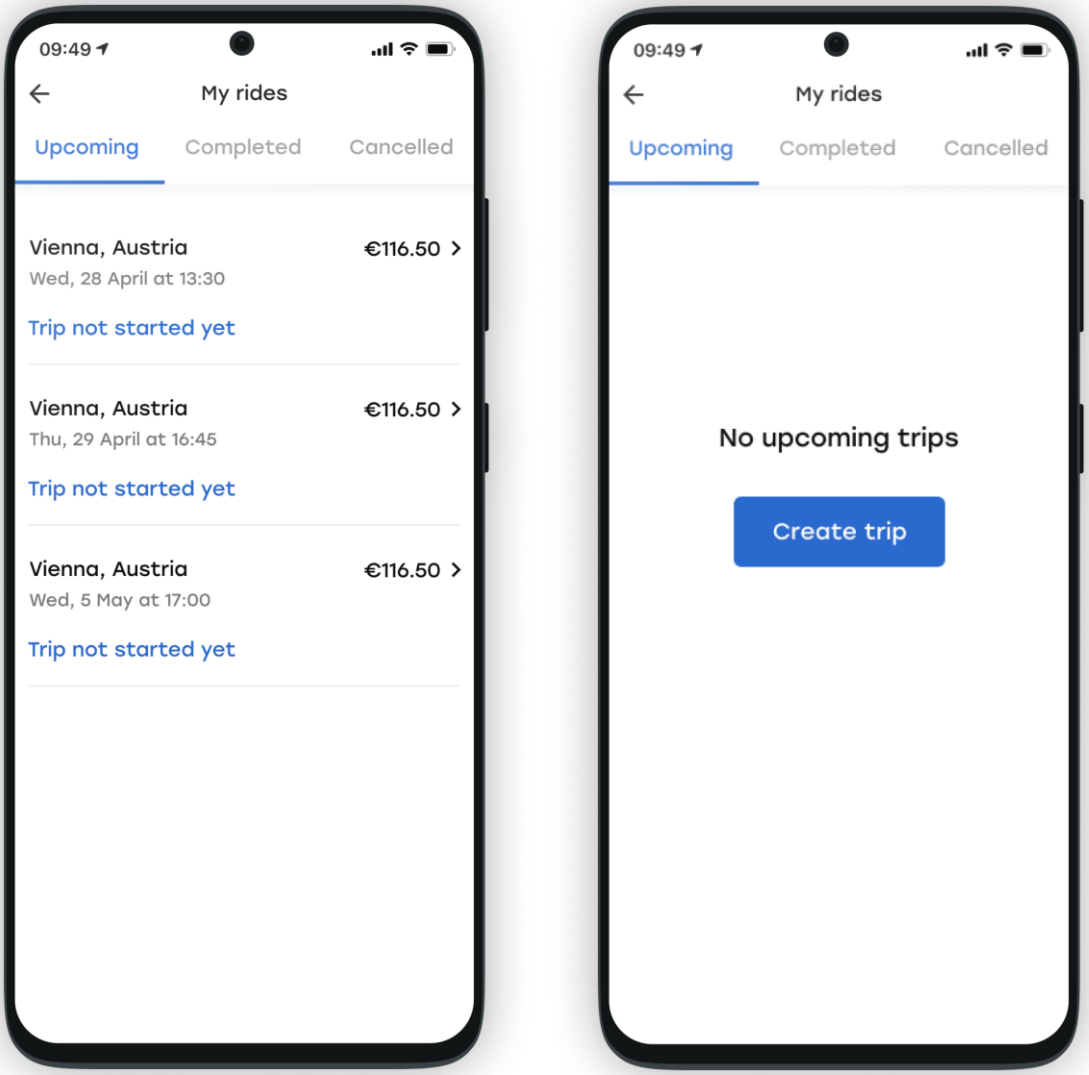

Рисунок 4.12 – Экран «Мои путешествия»

При нажатии на кнопку «Create trip», показанной на правом экране рисунка 4.12, пользователь перенаправляется на экран создания нового путешествия.

4.3.4 Раздел «Создание путешествия»

В этом разделе находятся поля ввода городов отправления и назначения, поиск которых осуществляется через партнерский сервис Aviasales.

Возможность выбора локации на карте запланирована на ближайшее будущее и имеет высокий приоритет, так как требует серьезных технических решений, денег и большого количества времени. Также планируется добавить дополнительные партнерские сервисы для выбора альтернативных способов передвижения, такие как поезд, автобус или свой транспорт. Реализация подобных взаимодействий также серьезных технических решений и расширенного денежного капитала.

При выборе городов отправления и назначения, пользователь перенаправляется на экран настройки нового путешествия, показанного на рисунке 4.13.

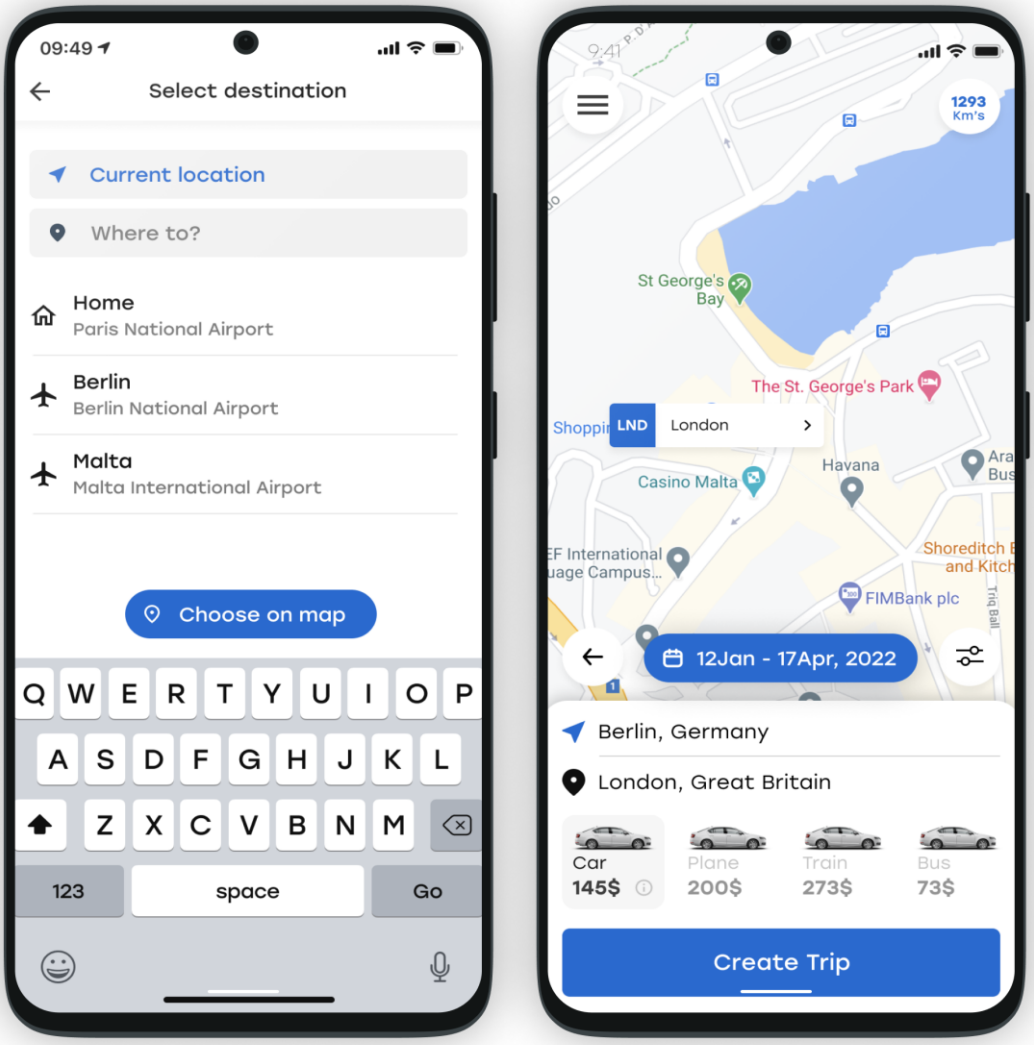

Рисунок 4.13 – Экран «Создание путешествия»

На этом экране у пользователя есть возможность выбрать: тип транспорта, на котором он предпочитает добираться из точки А в точку Б, даты отправления и возвращения. Также присутствует возможность отредактировать точку отправления и назначения.

Нажатие на кнопку фильтров открывает возможность еще точнее настроить предпочтения по будущему путешествию. Экраны фильтров в свернутом и развернутом виде представлены на рисунке 4.14.

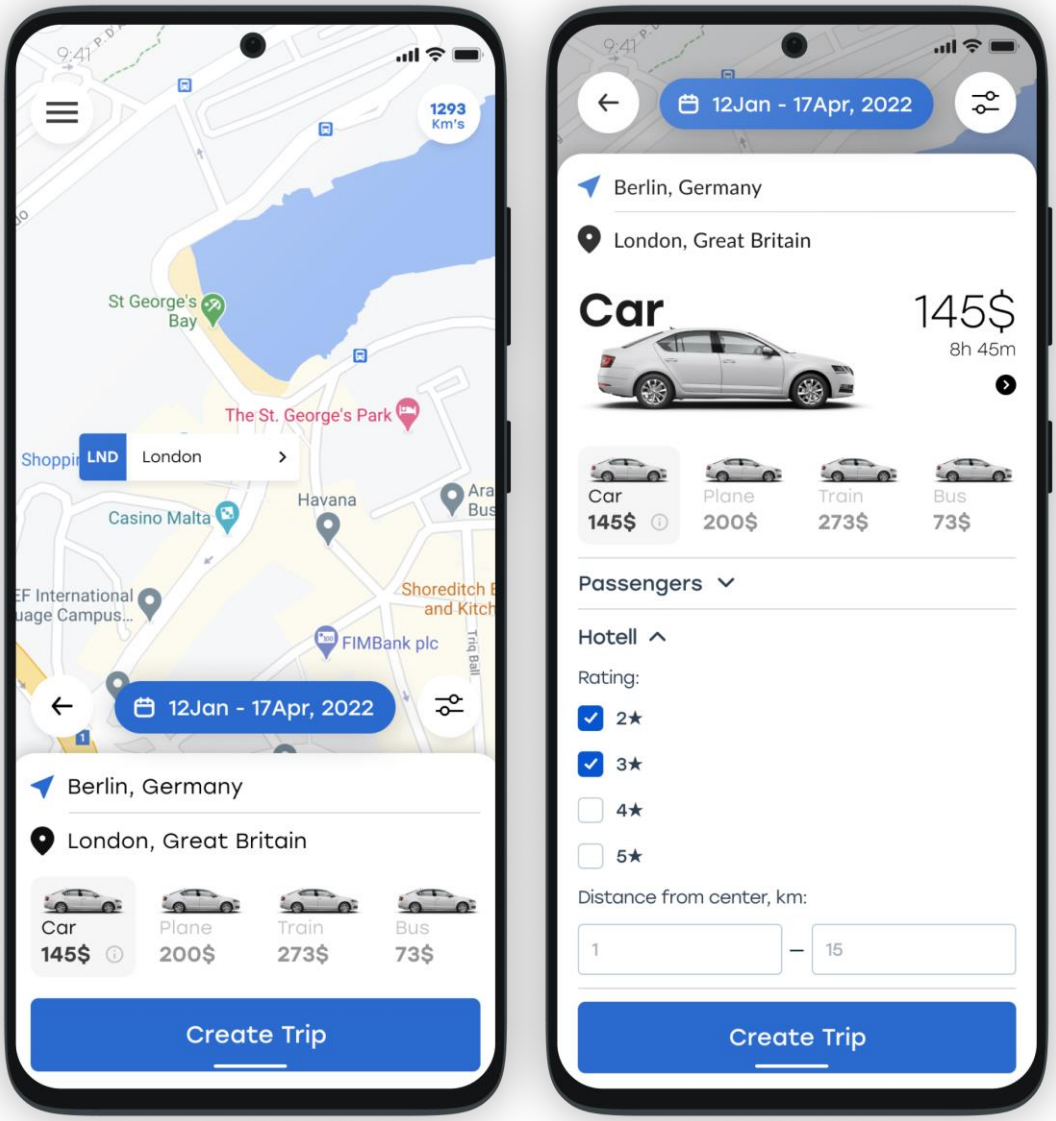

Рисунок 4.14 – Экраны фильтров создания путешествия

Каждый из фильтров позволяет получить необходимый уровень персонализации для алгоритма подсчета стоимости каждого аспекта путешествия.

В развернутом виде данный экран дает следующие возможности, необходимые для точного и персонализированного подсчёта цены будущей поездки:

1) Настройка количества пассажиров.

2) Настройка предпочтений по жилью (количество звезд и расстояние от центра города назначения).

3) Настройка предпочтений по еде.

4) Настройка предпочтений по транспорту.

5) Настройка предпочтений по развлечениям.

Также при создании путешествия может возникнуть необходимость пребывания в нескольких различных городах на определенном сроке. Приложение предоставляет своим пользователям такую возможность с помощью функционала сложного маршрута, результат которого продемонстрирован на рисунках 4.15 и 4.16.

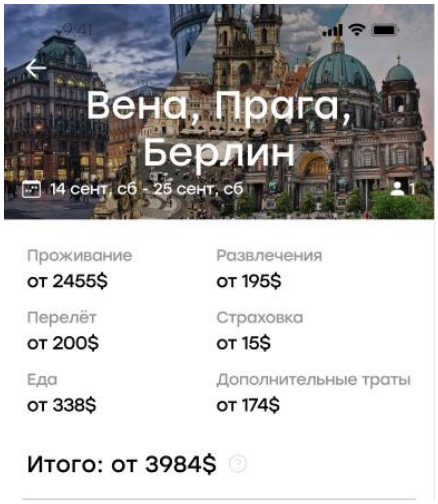

Рисунок 4.15 – Созданное путешествие

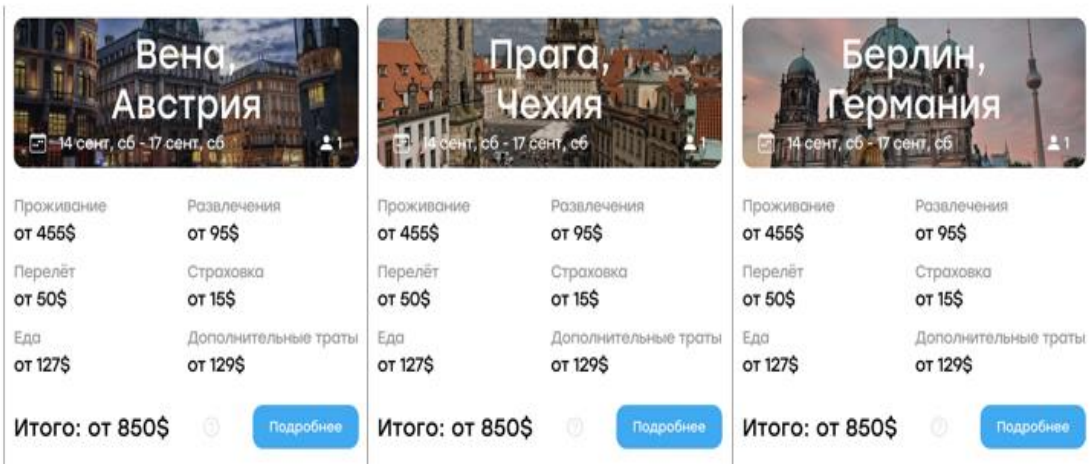

Рисунок 4.16 – Сложный маршрут

Данный функционал необходим в случае, когда маршрут пользователя включает в себя как минимум два разных города назначения. Он позволяет добавить до N различных городов и выбрать для каждого из них соответствующие даты. Помимо этого, на данном экране сохраняется та же функциональность, что и на экране обычного создания путешествия.

4.3.5 Управление созданным путешествием

По нажатию на кнопку «Создать путешествие», отправляется запрос на сервер, где он обрабатывается путем выполнения запросов к различным туристическим сервисам, а также подсчёта каждого из пунктов путешествия по особой формуле, зависящей от различных параметров, в том числе и фильтров, заданных пользователем. После создания путешествия, пользователь попадает на экран, где он может наблюдать все города, добавленные им в путешествие, промежуточные подсчёты по каждому из них, а также глобальный подсчёт стоимости путешествия представленном на рисунке 4.17.

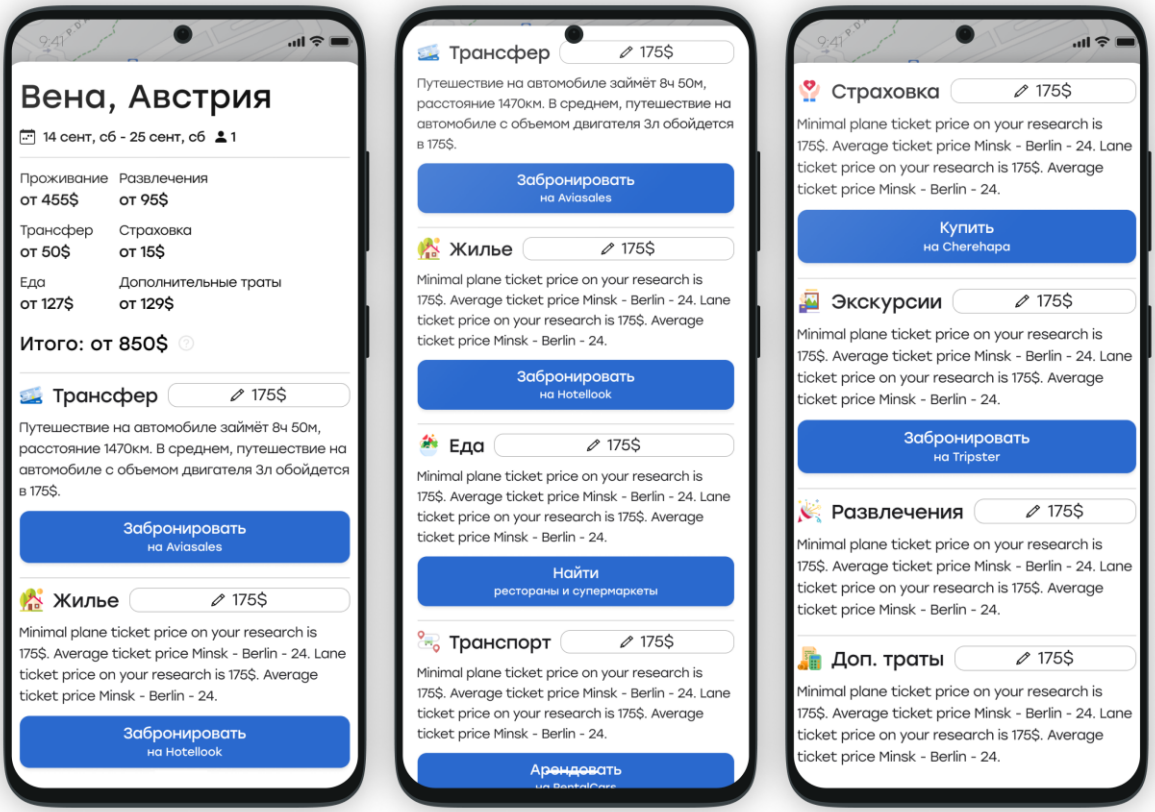

Рисунок 4.17 – Подробности созданного путешествия

На этом экране пользователь может ознакомиться с детальным подсчетом по выбранному городу, увидеть подробности, а также вручную изменить тот или иной пункт подсчета, если он уже приобрел товар или услугу из данной категории.

Также предусмотрена возможность перейти по ссылке, ведущей на сайт сервиса-партнера. На партнерском веб-сайте турист имеет возможность приобрести те или иные услуги или товары, которые его интересуют, по той цене, что была указана в приложении.

4.3.6 Вспомогательные экраны

В приложении присутствует достаточно большое количество вспомогательных экранов, необходимых для его полноценной работы. Экраны представлены на рисунках 4.18 и 4.19:

1) Экран профиля – содержит в себе функционал, позволяющий изменить различные данные пользователя, выйти из аккаунта или удалить его.

2) Диалоговый экран удаления аккаунта – по нажатию на кнопку «удалить аккаунт», аккаунт пользователя, а также все персональные данные будут полностью и безвозвратно удалены из базы данных сервера и мобильного приложения.

3) Экран ошибки – показывается в случае возникновения критической ошибки в приложении и уведомляет пользователя о ней.

4) Календарь – позволяет туристу выбрать даты своей будущей поездки.

5) Экран загрузки – позволяет туристу понять, что его запрос обрабатывается системой.

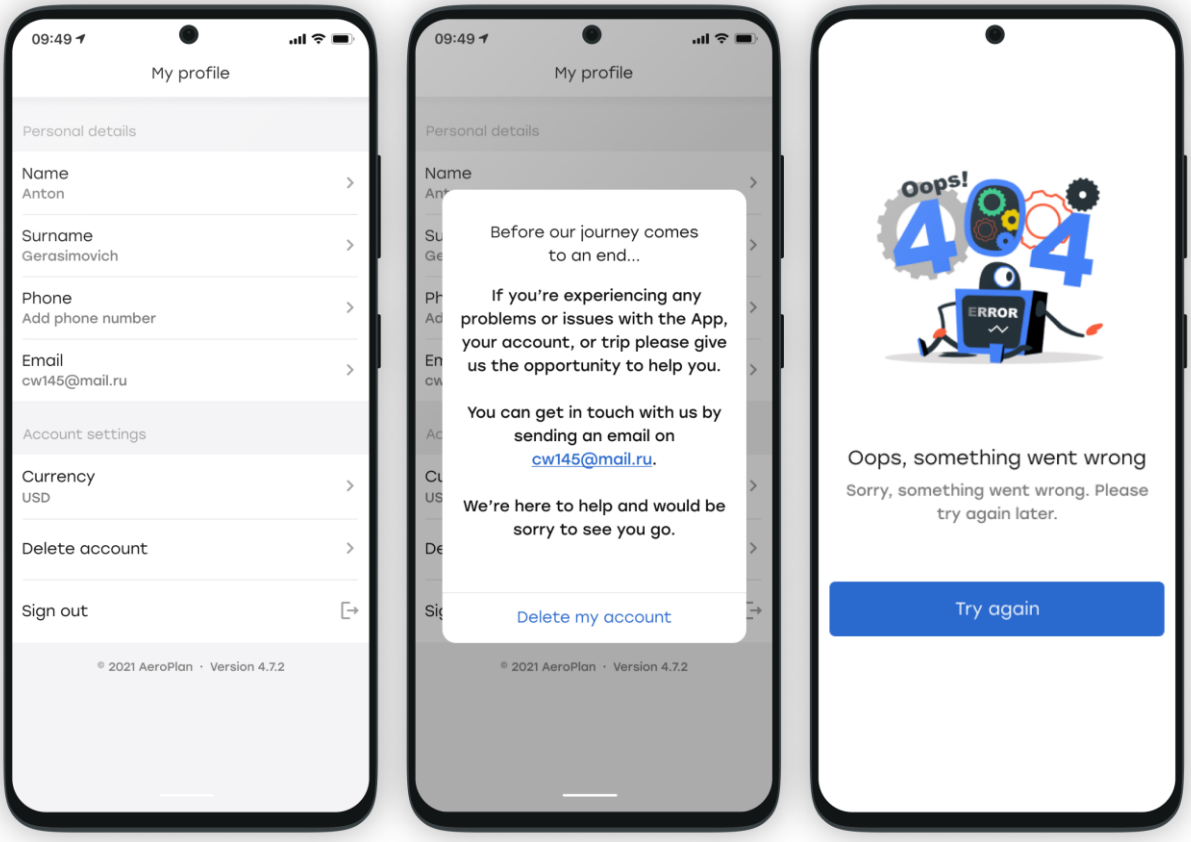

Рисунок 4.18 – Экраны профиля, ошибки и диалоговое окно удаления аккаунта

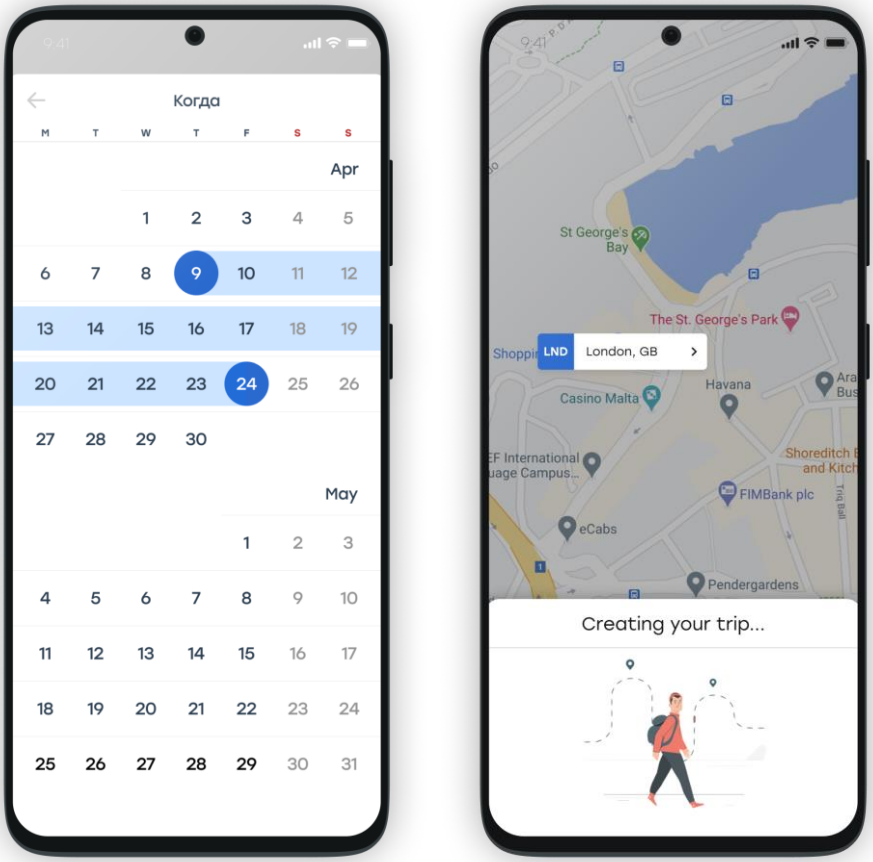

Рисунок 4.19 – Календарь и экран загрузки

4.3.7 Web-интерфейс приложения

Наличие Web-интерфейса дает пользователю возможность получить необходимую информацию и данные независимо от платформы, на которой он работает в данный момент. Будь то компьютер или телефон с любой операционной системой, клиент всегда сможет получить доступ к своему аккаунту просто воспользовавшись строкой браузера. Приложение AeroPlan предоставляет такую возможность.

В данном приложении Web-интерфейс служит также в качестве механизма для сброса паролей в случае, если пользователь забыл и не имеет возможности войти в свой аккаунт. Сброс пароля осуществляется с помощью подтверждения своей личности через электронную почту.

Опишем в данном пункте главы только общий интерфейс приложения, а подробности работы и разработки будут рассмотрены далее.

Заходя в приложение, пользователь сразу же попадает на главный экран, который изображен на рисунке 4.20, с тремя возможными действиями:

1) Посетить сайт – дает возможность перехода на лендинговую страницу.
2) Ввести свой login (экран на рисунке 4.21).
3) Зарегистрироваться в приложении (экран на рисунке 4.22).

Сайт самого приложения является лишь вспомогательным компонентом, так как основной идеей являлось создание именно мобильного приложения. Web-интерфейс расширяет его возможности.

Системы регистрации и аутентификации будут описана подробно в пятой главе. На данном этапе продемонстрируем лишь общий вид экранов.

Рисунок 4.20 – Главный экран

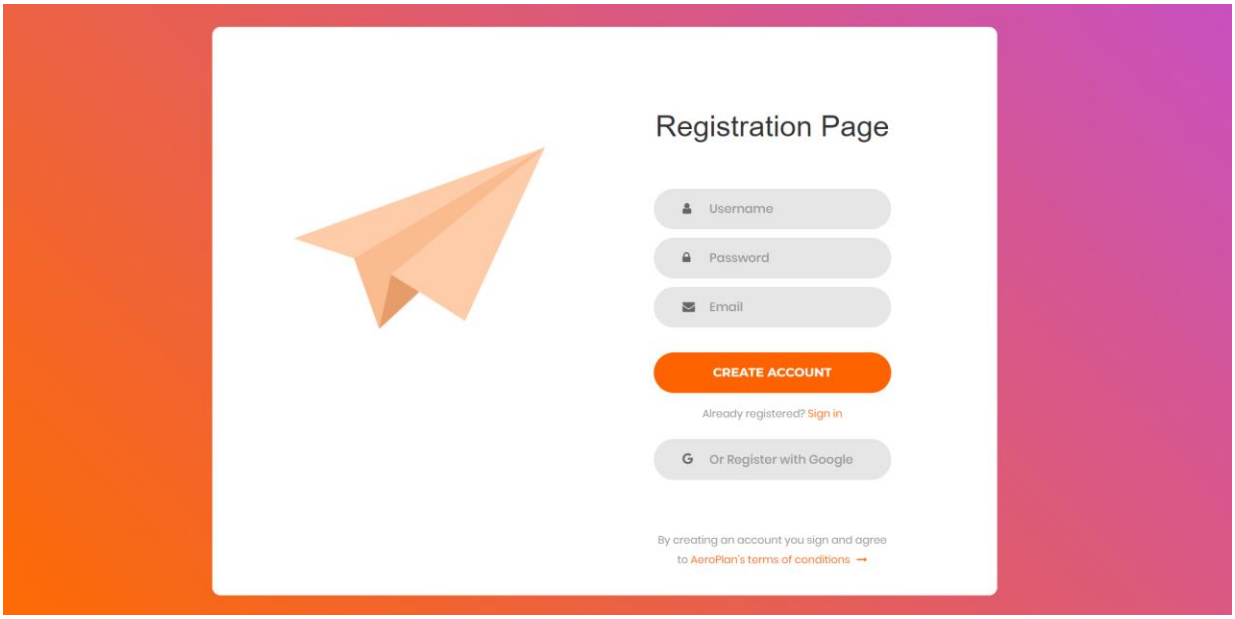

Рисунок 4.21 – Экран регистрации пользователя

На экранах регистрации и аутентификации видим, что пользователь может произвести свою идентификацию в системе не только через стандартный username и password, но и также с помощью google-аутентификации. Это значительно ускоряет и упрощает работу как для самого клиента (быстрота ввода данных и входа в систему), так и для приложения по получению необходимых данных.

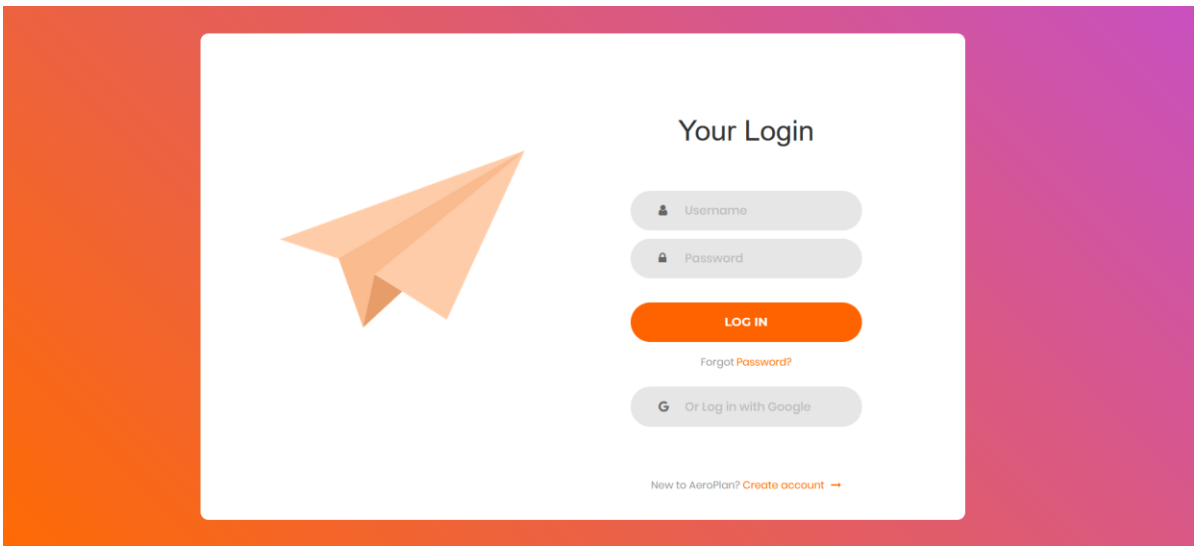

Рисунок 4.22 – Экран аутентификации пользователя

В случае возникновения ошибки (к примеру, с кодом 404), пользователя перенаправляет на интерфейс с подробной информацией о возникшей проблеме. Пример такого экрана представлен на рисунке 4.23. После есть возможность вернуться на главный экран.

Рисунок 4.23 – Экран ошибки 404 – NOT FOUND

4.4 Заключение по реализации визуального представления

Таким образом, пользователь приложения AeroPlan может буквально за пару минут создать свое путешествие, персонализировать его, получить подробную информацию по каждому из аспектов путешествия, его цену, и возможность приобрести конкретные товары или услуги: авиабилеты, отели, страховку и т.д. А удобный Web-интерфейс предоставит всю необходимую информацию о сервисе и приложении AeroPlan.

ГЛАВА 5
РЕАЛИЗАЦИЯ ТЕХНИЧЕСКИХ РЕШЕНИЙ И БИЗНЕС-ЛОГИКИ. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

5.1 Общие описание задач серверной части приложения

Серверная часть приложения по праву считается сердцем и мозгом приложения. Механизмы и логика, реализованные здесь, управляют всеми процессами системы, позволяют совершать все необходимые, предусмотренные разработчиками, операции и удовлетворят потребности своих клиентов. Поэтому качественная и продуманная реализация сервера приложения является ключевой задачей при разработке программного обеспечения такого типа.

В предыдущих главах мы прогнозировали требования ко всей системе и ее составным частям, а также расписали задачи. Согласно этим требованиям, была спроектирована как архитектура самого приложения, так и ее базы данных. После был рассмотрен дизайн и пользовательский интерфейс, который нацелен на отображение всей идейной функциональности приложения.

Таким образом для реализации описанной функциональности необходима хорошо продуманная и качественная реализация следующих элементов:

1) Внутренней бизнес-логики приложения.
2) Взаимодействий со сторонними сервисами для получения данных.
3) Продуманность транзакций и запросов в базу данных.
4) Взаимодействия с клиентской стороной приложения.

Выделенные пункты были отмечены, так как являются наиболее уязвимыми к разного рода проблемам и ошибкам, которые могут возникнуть по различным причинам, не зависящих от разработчика: отказ работоспособности облачного хоста, отсутствие связи с базами данных или потеря одной, высокая загруженность сервера, ошибки клиентской стороны и прочие.

Также данные элементы берут на себя основное время выполнения операций, отвечающих на те или иные запросы со стороны пользователя. А так как приложение ставит задачу максимального уменьшения времени ответа на предоставление разного рода информации или действий, то реализация легковесных взаимодействий, а также выбор технологий, позволяющих быстро и качественно получить необходимое решение, являются приоритетными задачами.

Таким образом в следующих пунктах данной главы будут подробно рассмотрены упомянутые моменты, а также их реализация, с помощью современных технологий разработки.

5.2 Взаимодействия сервера приложения с API сторонних сервисов

Как уже упоминалось ранее, большее количество всех необходимых данных для пользователя сервис AeroPlan предоставляет благодаря получению и агрегированию информации от других сервисов через их API. Теперь, получив доступ к данным API, имеем возможность приступать к реализации логики обработки получаемой информации.

Принцип запросов к каждому из сервиса-партнера примерно одинаков, отличаются лишь передаваемые данные, типы запросов, а также методы защищенности транзакций. Вся передача информации в сети через те или иные технологии осуществиться по определенным архитектурным принципам. И перед тем, как будут приведены конкретные примеры используемых технологий, дадим короткое пояснение по терминологии.

5.2.1 Общая терминология и структура взаимодействий

Напомним, что REST представляет собой архитектурный стиль для обеспечения стандартов между компьютерными системами в Интернете, что упрощает взаимодействие систем друг с другом, включая универсальные способы обработки и передачи состояний ресурсов по HTTP/HTTPS. На данный момент REST подход является наиболее популярным и востребованным при разработке клиент-серверных приложений [12].

REST подход предоставляет:

– независимое внедрение различных компонентов;

– масштабируемость взаимодействий систем;

– равенство интерфейсов;

– наличие промежуточных компонентов, способствует снижению задержки и увиливанию безопасности.

Преимуществами REST являются:

– передача данных в том же виде, что и сами данные, без дополнительных внутренних прослоек;

– каждая единица информации (ресурс) однозначно определяется URL – это значит, что URL по сути является первичным ключом для единицы данных. Причем совершенно не имеет значения, в каком формате находятся данные по адресу;

– управление информацией ресурса целиком и полностью основывается на протоколе передачи данных.

Наиболее распространенными протоколами являются HTTP/HTTPS. Приложение AeroPlan использует протокол HTTPS для общения как со сторонними сервисами-партнёрами (в некоторых случаях HTTP), так и с клиентской стороной приложения [12].

Для названных протоколов действие над данными задается с помощью определенных методов, основные из которых:

– GET (запрос на получение определенного ресурса или коллекции ресурсов по определенному идентификатору);

– POST (запрос на создание нового ресурса);

– PUT (запрос на обновление конкретного ресурса определенному идентификатору);

– DELETE (запрос на удаление определенного ресурса по идентификатору).

Стоит отметить, что зачастую данные в сети передаются в формате JSON так как такой формат используется в REST API. Данные JSON записываются в виде пар ключ-значение. Ключ – это название параметра, который мы передаем серверу. Он служит маркером для принимающей запрос системы. В качестве значений в JSON могут быть использованы:

– JSON-объект (неупорядоченное множество пар «ключ-значение»);

– массив;

– число (целое или вещественное);

– литералы true / false и null;

– строка.

Системы, которые следуют парадигме REST, не имеют состояния, что означает, что серверу не нужно ничего знать о том, в каком состоянии находится клиент, и наоборот. Таким образом, и сервер, и клиент могут понимать любое полученное сообщение, даже не просматривая предыдущие сообщения [13].

5.2.2 Обзор программных технологий

Для того, чтобы обратиться к тому или иному сервису и получить от него необходимые данные, используется технология Retrofit.

Retrofit – это клиент REST для Java и Android. Он позволяет относительно легко получать и загружать JSON (или другие структурированные данные) через веб-службу на основе REST [14].

В Retrofit настраивается конвертер, который будет использоваться для сериализации данных (процесс перевода структуры данных в последовательность байтов) и их конвертации. Обычно для JSON используется GSON (позволяет конвертировать объекты JSON в Java-объекты и наоборот), но также можно добавить собственные конвертеры для обработки XML или других протоколов.

Retrofit использует библиотеку OkHttp для HTTP-запросов.

OkHttp – это эффективный клиент HTTP и HTTP / 2 для приложений Android и Java. Он поставляется с расширенными функциями, такими как пул

соединений (если HTTP / 2 недоступен), прозрачное сжатие GZIP и кэширование ответов, чтобы полностью избежать повторных запросов в сети [15].

Он также может восстанавливаться после распространенных проблем с подключением, а в случае сбоя подключения, если служба имеет несколько IP-адресов, она может повторить запрос на альтернативные адреса. На высоком уровне OkHttp-клиент предназначен как для блокировки синхронных, так и для неблокируемых асинхронных вызовов.

Таким образом, Retrofit генерирует URL, по уже заданным правилам, что действительно очень удобно. Всё что нужно это прописать интерфейс. После получения объекта в Retrofit можно использовать конвертер, который может сразу предоставить объект в нужном виде для последующей работы. Также есть возможность настроить Timeouts по выполнению запросов и получению ответов, что весьма полезно.

5.2.3 Взаимодействие с API и данными сервисов-партнеров

Продемонстрируем примеры классов их настройку и взаимодействия, обеспечивающие запросы на сторонние сервисы на примере обращения к сервису Aviasales. В следующих пунктах будет описана клиентская настройка на стороне сервера AeroPlan, настройка API и реализация прямых запросов через контроллеры самого приложения.

5.2.3.1 Клиентская настройка

Клиентская настройка, представленная на рисунке 5.1.

В данном классе представлен синхронизированный метод getApi через который и осуществляется запрос на сторонний сервис. В нем настраивается экземпляр класса OkHttpClient, отвечающий за время максимального соединения и чтения данных, чтобы избежать длительного ожидания ответа в случае долгой обработки.

Использование тайм-аутов полезно для прерывания вызова, когда его одноранговый узел недоступен. Сбои в сети могут быть из-за проблем с подключением клиентов, проблем с доступностью сервера или иных проблем. OkHttp поддерживает тайм-ауты подключения, чтения и записи. В приложении тайм-ауты были установлены на чтение и на соединение с сервисом по 60 секунд.

Экземпляр класса Retrofit отвечает за создание самого API: установки серверного URL для дальнейшего обращения, добавляется конвертер, про который было сказано выше, и добавляются настройки клиента.

После этого создается instance для создания только одного экземпляра API посредством метода create, в который передается интерфейс, определяющий конкретные запросы к выбранному сервису-партнеру.

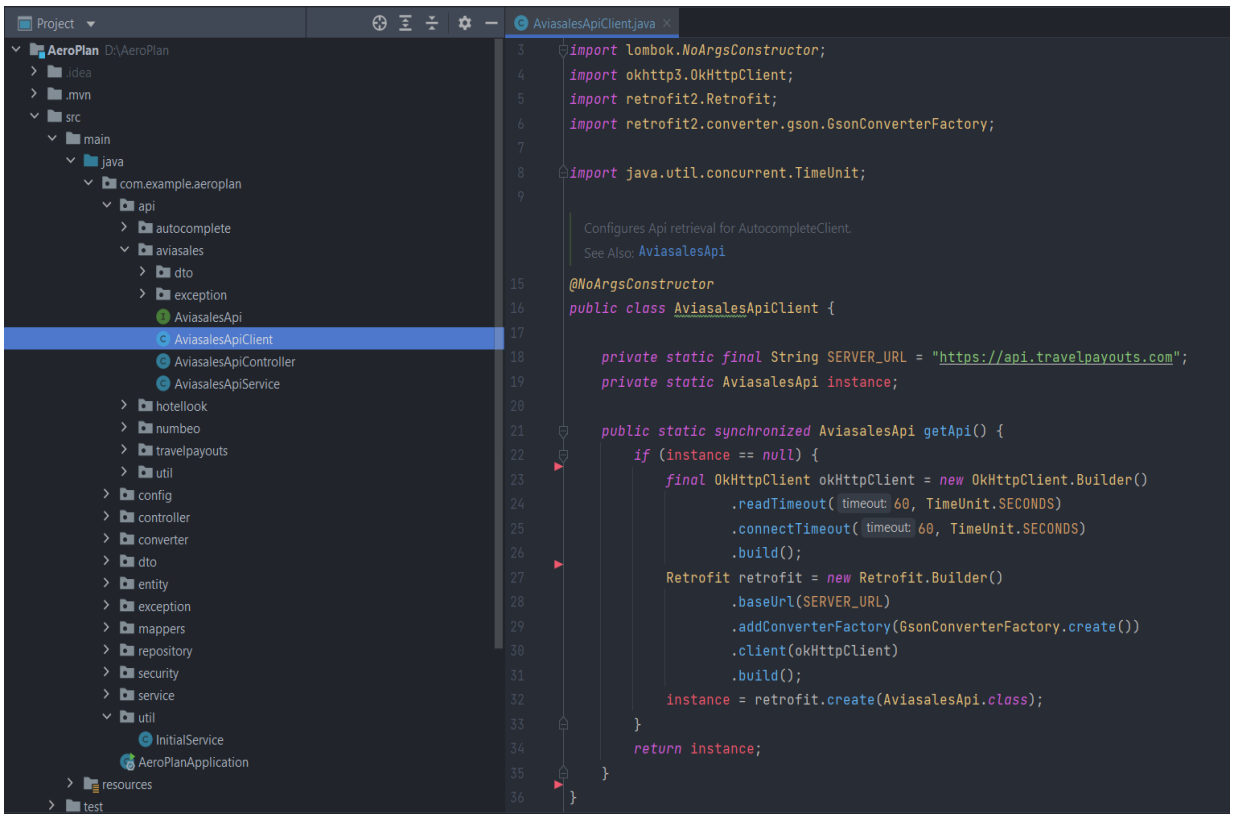

Рисунок 5.1 – Клиентская настройка к сервису Aviasales

5.2.3.2 API для связи с сервисами-партнерами

API для связи с сервисами-партнерами, представленная на рисунке 5.2.

Как уже говорилось ранее: для протоколов HTTP/HTTPS действие над данными задается с помощью определенных методов. При обращении данных от сервисов используются только два метода: GET и POST, так как мы только получаем данные без возможности их изменения или удаления. GET запрос формируем в случае, когда все данные мы закладываем в Path (путь отправки) без наличие Body (тела запроса), а POST – когда передаем тело запроса. Это правило использования данных методов.

Как видим из рисунка 5.2, каждый из методов имеет аннотацию с указанием метода отправки запроса и пути с входящими в него параметрами (относительного URL ресурса).

URL-адрес запроса может обновляться динамически с использованием блоков замены и параметров метода. Блок замены – это буквенно-цифровая строка, окруженная символами «{}». Соответствующий параметр должен быть аннотирован той же строкой.

Также можно добавить параметры запроса с помощью аннотации Query. В качестве параметров Query заносим данные, необходимые для получения информации. Все форматы запросов, ответов, входных параметров и их тип прописаны в документациях сервисов, которые необходимо подробно изучать и реализовывать. В противном случае запрос будет безуспешным и данные не

будут получены. При проблеме получения данных приходит сообщение об ошибке.

Можно также указать объект HTTP-запроса при помощи Body аннотации как показано на рисунке 5.2 в методе getFlightSearchData. Объект ответа от сервиса также будет преобразован с помощью конвертера, указанного в Retrofit экземпляре.

Методы имеют Call – ответ на запрос в том формате, котором мы запросили (формат должен совпадать с тем, что отдает сервис, в противном случае данные не смогут быть получены). Стоит отметить, что Call экземпляры могут выполняться синхронно или асинхронно. Каждый экземпляр можно использовать только один раз.

Документация по запросам и ответам бралась с официальных сайтов сервисов или с помощью партнерской сети Travelpayouts, которая отлично описывает назначение и тип используемого метода для запроса к выбранному сервису, всех его параметров, а также и сам ответ на запрос в формате JSON:

– приводит примеры ответа в виде готового JSON и данных в заголовке;
– подробную информацию по каждому параметру;
– объясняет нюансы, которые могут происходить при ответе.

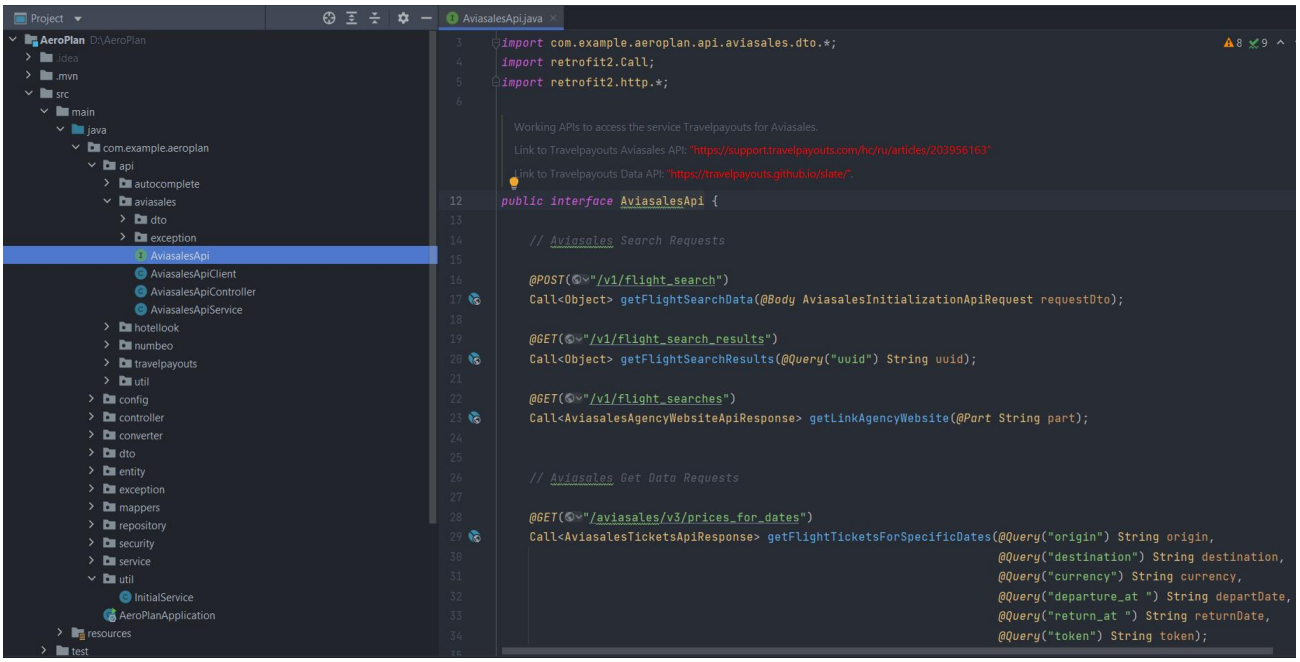

Рисунок 5.2 – Настройка API к сервису Aviasales с примерами

5.2.3.3 Отправка и получение данных с использованием готовых API

Отправка и получение данных от сервисов-партнеров с использованием готовых API через прямые запросы к приложению AeroPlan, представленная на рисунке 5.3.

Отправка данных осуществляется посредствам готовых API с использованием полученных от пользователя данных. Также в каждый запрос добавляется token, идентифицирующий сервис отправителя (в данном случае AeroPlan) для предоставления ему прав на это, который необходимо передавать в адресной строке параметром token, адрес сайта (хост автора запроса), специальный партнерский маркер (пять уникальных цифр), выданный системе сервисом Travelpayouts, и специальную сигнатуру, включающая в себя зашифрованные алгоритмом хеширования MD5 данные запроса в виде сгруппированного в алфавитном порядке списка параметров.

Этот же котроллер получает ответ на отправленный запрос, конвертирует данные в необходимое для нас DTO и затем отправляет вызвавшей его стороне: пользователю или на сервисную часть.

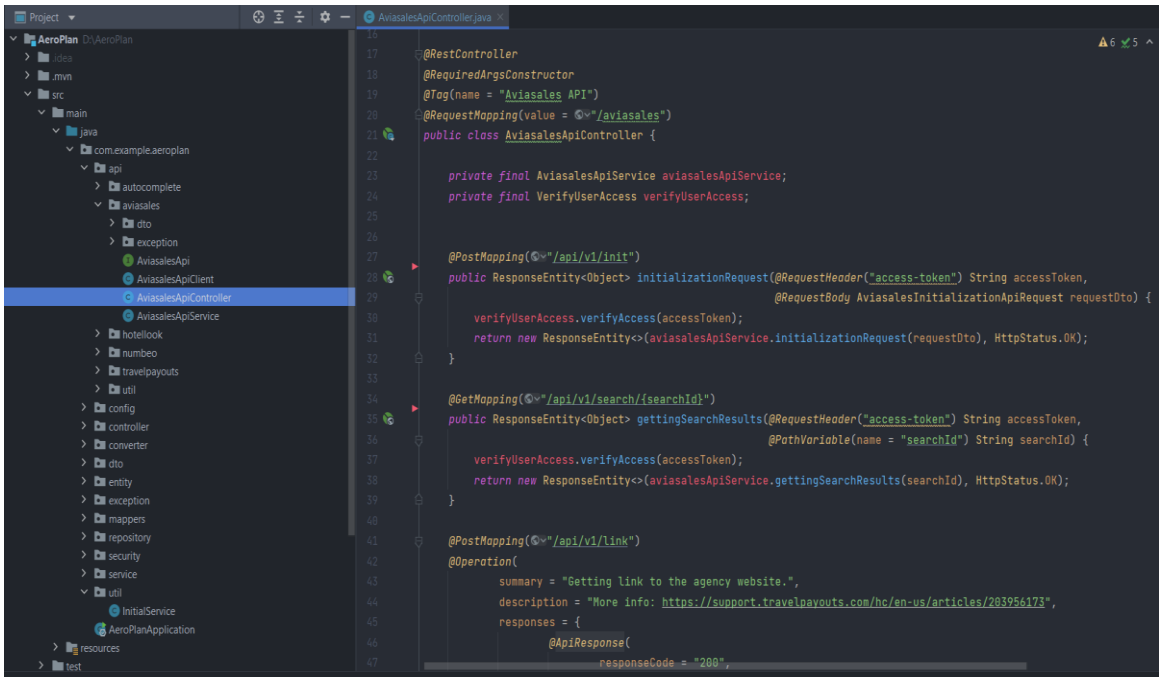

Рисунок 5.3 – Контроллер Aviasales с примером обращения к API

За бизнес-логику взаимодействий контроллеров приложения и API сервисов-партнеров отвечает сервисные классы приложения, которые описывают все необходимые взаимодействия, обрабатывают входящие данные, анализируют запросы и ответы. Через такие сервисы приложения ведутся запросы к API не только от контроллеров, но и от других сервисных классов системы.

5.3 Общая схема взаимодействия бизнес-логики приложения

Покажем, как происходит взаимодействие данных технологий, а также все общие взаимодействие всех компонентов системы (от клиента к серверу, от сервера к базе данных и сервисам-партнерам по API).

Рассмотрим ситуацию, когда пользователь оправляет запрос на инициализацию и первоначальный подсчет путешествия в определенный город другой страны на выбранные даты с назначением специализированных фильтров, конкретизирующих его предпочтения. Примерная схема работы системы показан на рисунке 5.4.

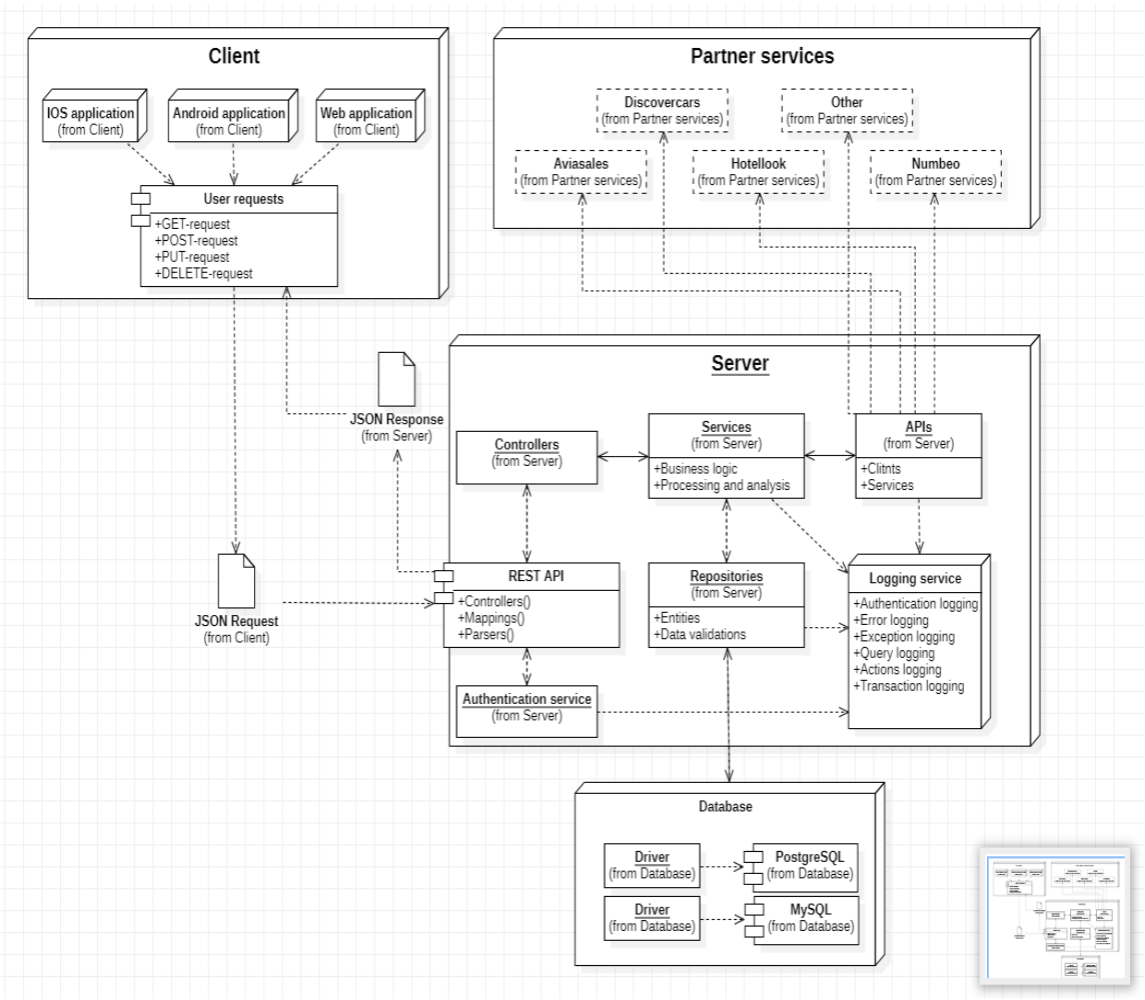

Рисунок 5.4 – Принцип внутреннего взаимодействия бизнес-логики приложения

1) Как только пользователь с клиентской части приложения отправит запрос на получение определенных данных, запрос в формате JSON приходит на сервер на определенный контроллер посредством HTTPS протокола.

2) После того, как данные были получены, они преобразуются в так называемый объект DTO для дальнейшей передачи этих данных между подсистемами приложения AeroPlan.

3) В соответствии с требуемыми данными для запроса на тот или иной API сервиса-партнёра, из DTO в сервисной части приложения берутся данные и формируется запрос либо же несколько запросов.

4) Это запрос с помощью вышеописанных технологий Retrofit и OkHttp, отправляется на конкретные API сервисов-партнёров, где те в свою очередь формируют ответ по интересующим нам данным.

5) Этот ответ возвращается в виде указанного сервисом объекта, при этом конвертируется прописанным в настройках Retrofit образом и также обрабатывается в сервисной части приложения AeroPlan.

6) Полученные данные обрабатываются с помощью основной бизнес-логики приложения, после чего часть отправляется непосредственно клиенту приложения в ответ на его запрос, а часть сохраняется в базу данных для дальнейшего хранения и анализа.

7) Основные действия, происходящие в системе, логируются посредством специальных библиотек и заносятся в отдельные системы хранения подобной администраторской информации. Таковыми действиями могут являться: информация об ошибках либо исключительных ситуациях, данные авторизации пользователей, информация о запросах и другие.

5.4 Взаимодействия сервера приложения с облачным хранилищем

Для хранения, обработки и предоставления данных пользователя в приложении AeroPlan используется управляемая служба базы данных SQL Postgres, которая размещается на облачном хостинге Heroku.

Heroku Postgres предоставляет веб-панель управления, возможность обмениваться запросами с помощью фрагментов данных, а также множество команд управления, доступных через интерфейс командной строки и планов для разных уровней обслуживания.

На данный момент был выбран бесплатный тарифный план с ограниченным числом запросов и размером базы данных. Однако этого плана вполне достаточно для первоначального тестирования и проверки качества работы запросов. Как только количество информационных данных увеличится и не станут соответствовать текущему плану, расширить базу данных не составит никакого труда.

База данных приложения AeroPlan служит для хранения и предоставления следующих данных и информации пользователя:

1) Персональные данные пользователя – данные, с помощью которых осуществляется идентификации и последующая аутентификация, а также авторизация в системе, возможность иметь обратную связь, а также оплата и получение пользовательских рекомендаций от приложения;

2) Пользовательская информация – данные, полученные и сохраненные от пользователя в ходе использования им приложения. Пользовательская информация формируется на основе предыдущих запросов и предпочтений клиента только при условии его согласия.

3) Метаинформация – прочие данные, служащие для определенно назначенных целей.

База данных позволяет осуществлять безопасное хранение персональных данных посредствам шифрования паролей и встроенных внутренних систем защиты на самом хостинге. Обращение к базе предусматривает защиту от SQL-инъекций и прочих атак, а также осуществляется только для явно авторизированных в системе пользователей в соответствии с ролью, которой они обладают в приложении.

Стоит отметить, что информация о составленных планах путешествий, а также остальные данные на облачный сервер сохраняются только для зарегистрированных в системе пользователей и только при их согласии. Такой подход позволяет избежать излишнего переполнения информационного хранилище данными анонимных пользователей.

5.5 Взаимодействия сервера с клиентской стороной приложения

За взаимодействие сервера с клиентской стороной приложения отвечает технология Spring MVC. Фреймворк Spring Web MVC разработан на основе платформы, DispatcherServlet, которая отправляет запросы обработчикам с настраиваемыми сопоставлениями обработчиков, разрешением представления, языковым стандартом и разрешением темы, а также поддержкой загрузки файлов [16].

Обработчик по умолчанию на основе @Controlleru @RequestMapping аннотации, предлагая широкий спектр гибких методов обработки. С появлением Spring 3.0 этот @Controller механизм также позволяет создавать веб-сайты и приложения RESTfull с помощью @PathVariable аннотаций и других функций.

Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.

– Модель (model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

– Представление (view) отвечает за отображение данных модели пользователю, реагируя на изменения модели.

– Контроллер (controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Как уже было сказано: вся логика работы Spring MVC построена вокруг DispatcherServlet, который принимает и обрабатывает все HTTP-запросы (из UI) и ответы на них. Рабочий процесс обработки запроса DispatcherServlet-ом проиллюстрирован на следующей диаграмме, представленной на рисунке 5.5.

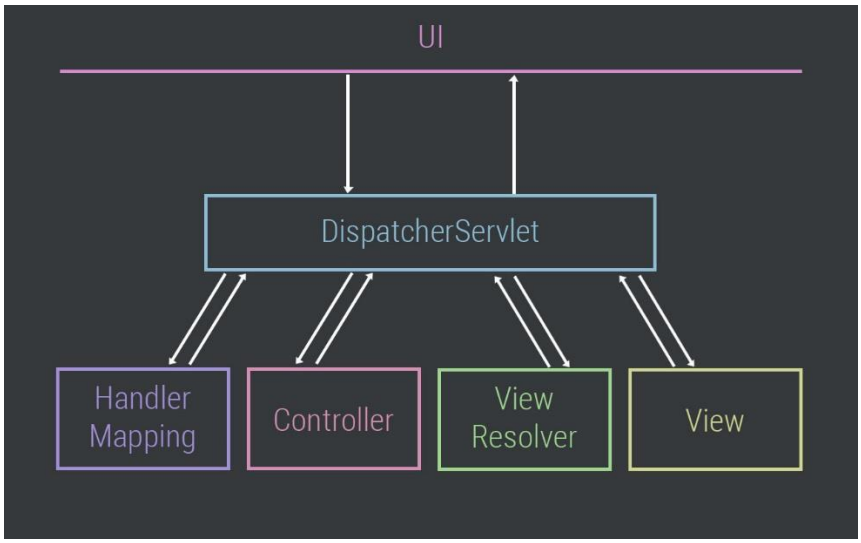

Рисунок 5.5 – Аспекты работы Spring MVC

Приведем последовательность событий, соответствующая входящему HTTP-запросу со стороны клиента [17]:

1) После получения HTTP-запроса DispatcherServlet обращается к интерфейсу HandlerMapping, который определяет, какой контроллер должен быть вызван, после чего, отправляет запрос в нужный из них.
2) Контроллер принимает запрос и вызывает соответствующий служебный метод, основанный на GET или POST. Вызванный метод определяет данные модели, основанные на определенной бизнес-логике, и возвращает в DispatcherServlet имя отображения (view).
3) При помощи интерфейса ViewResolver DispatcherServlet определяет, какое отображения нужно использовать по полученному имени.
4) После того, как отображения создано, DispatcherServlet отправляет данные модели в виде атрибутов или JSON в отображение, который в конечном итоге отображается в браузере или Android приложении.

Так как мы в основном работаем с Android приложением, в которое данные передаются в формате JSON, то в качестве данных для получения и отправки по сети являются определенные объекты (@ResponseBody), а также информация, содержащаяся в адресе самого запроса (@PathVariable, @RequestParam, @RequestPart) и в headers (@RequestHeader). К примеру, у метода может быть любое количество @PathVariable аннотаций, а соответственно и значений [12].

В приложении AeroPlan имеется реализация большого числа запросов, поступающих с клиентской стороны приложения. Но суть работы механизма, который был описан выше остается неизменным для каждого.

5.6 Внутренняя реализация бизнес-логики приложения

Бизнес-логика – это настраиваемые правила или алгоритмы, которые обрабатывают обмен информацией между базой данных и пользовательским интерфейсом. Бизнес-логика – это часть компьютерной программы, которая содержит информацию, определяет или ограничивает работу бизнеса [18].

Такие бизнес-правила представляют собой операционные политики, которые обычно выражаются в истинных или ложных двоичных файлах. Бизнес-логику можно увидеть в рабочих процессах, которые они поддерживают, например, в последовательностях или этапах, которые подробно определяют надлежащий поток информации или данных и, следовательно, процесс принятия решений. Бизнес-логика также известна как «логика предметной области».

Внутренняя бизнес-логика приложения состоит не только из оптимальной реализации запросов и ответов к (от) клиентской стороне приложения, запросов на сервисы-партнеры и реализаций транзакций к базе данных.

Она включает в себя объемный пласт работы по подсчету стоимости тех или иных путешествий пользователя (как в конкретных городах, так и по всему путешествию в целом). Ведь необходимо оптимизировать все запросы, организовать многопоточную среду для выполнения всех операций, а также учесть все возможные проблемы и ошибки, которые могут возникнуть в высоконагруженной системе [18].

5.6.1 Описание и реализация основных аспектов бизнес-логики

Бизнес-логика приложения AeroPlan настолько объемная, что в рамках данной работы не представляется возможным описать весь реализованный функционал. Однако кратко изложим суть основной идеи создания путешествия, первоначальный подсчет, а также сохранение всех деталей поездки пользователя. Программная реализация алгоритма обработки данных и краткая структура методов приведена в приложении А дипломной работы.

При создании путешествия на стороне клиентской части приложения формируется запрос, по выбранным пользователем параметров и установленных значений фильтров. Эти данные преобразуются в JSON-запрос, который далее отправляется на определенный контроллер сервера методом POST.

После получения сервером запроса, происходит проверка аутентификации пользователя на уровне фильтрации данных на сервер, а также сверка токена доступа. При успешной идентификации пользователя в системе, отправленный

им запрос передается в серверную часть приложения. С помощью методов, представленных в приложении А, генерируется ответ:

1) Первым шагом является формирование запросов на сторонние сервисы для получения данных. Эти запросы сначала переходят на API-сервис в самом приложении и тот в свою очередь адресует их определенным системам. Ответы также обрабатываются API-сервисом, конвертируются и возвращаются на сервис, выполняющий основную бизнес-логику.

2) Далее данные передаются на методы, формирующие на основе их путешествие пользователя: генерируют название и отображение, структурируют полученную информацию и сохраняют ее, выполняют все подсчеты. Результат сгенерированного путешествия сохраняется через репозитории приложения в базу данных, для дальнейшего его редактирование и предоставления.

3) После полного подсчета и сохранения всех данных формируется ответ пользователю посредствам работы конвертера, преобразующие данные из базы в определенную сущность, не содержащей в себе всей объемной информации. Эта сущность отправляется JSON-ответом клиентской части приложения.

Таким образом мы кратко описали одну из наиболее интересной идеи приложения. По мимо этого реализовано множество алгоритмов прямого запроса к API-сервису для получение любой интересующей информации как по билетам, так и о качестве жизни в конкретном регионе. Также реализованы методы настройки, редактирования и прочие возможности по управлению данными пользователя, путешествия и всех его составных частей.

5.6.2 Тестирование основных аспектов приложения

В ходе разработки программного обеспечения были выполнены операции по настройке и проверки автоматического тестированию основных методов приложения посредством фреймворка Junit 5. Для проверки работы сервисной части приложения использовался фреймворк Mockito, который позволяет проверить определенную функциональность приложения, не затрагивая работу с базой данных. В ходе проведения такого тестирование, был выявлен ряд ошибок как на уровне базы данных, так и в самом приложении. В итоге все проблемы были устранены, а реализация протестирована на настоящих данных.

Также, посредством сервиса Insomnia, выполнялось тестирование всех API-запросов на сервер с клиентской стороны и запросы от сервера к сервисам-партнерам. Такой подход позволил устранить все неполадки с выполнением запросов до того момента, как функционал был испытан front-end разработчиками.

5.7 Реализация Web-интерфейса приложения

Реализация Web-интерфейса приложения AeroPlan являлось необходимым решением для создания логики для сброса пароля у пользователей в случае, когда клиент мог забыть его или потерять. Также был реализован функционал регистрации, аутентификации и добавлена страница с информацией о приложении.

Так как пароль является важнейшим инструментом входа того или иного пользователя в систему, его ключом для авторизации в ней, он является лакомой добычей для злоумышленников, пытающих получить доступ к системе или аккаунту другого пользователя.

И такой функционал, как сброс и смена пароля, является уязвимым местом в системе, которое необходимо тщательно продумать и надежно реализовать. Поэтому далее представим механизм решения данной задачи.

В случае, когда пользователь захочет обновить свой пароль, ему необходимо ввести свой электронный ящик, зарегистрированный в системе AeroPlan, как показано на рисунке 5.6. Процесс регистрации и аутентификации был описан в четвертой главе данной работы.

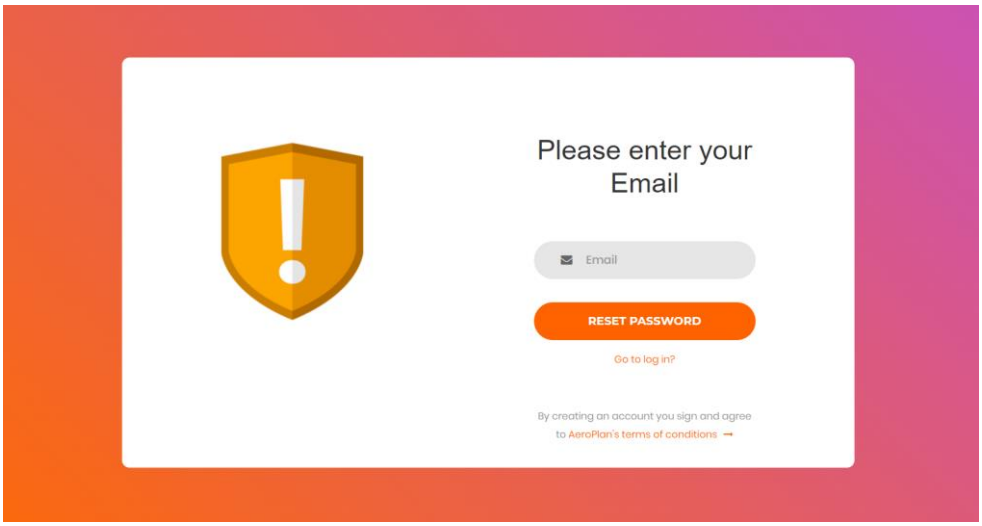

Рисунок 5.6 – Экран ввода электронной почты

В случае наличие такового, пользователю автоматически будет выслано на него письмо, с ссылкой, подтверждающей его личность через специальный токен, созданный на серверной стороне, по которой нужно перейти для дальнейших действий. Примерный вид письма указан на рисунке 5.7. А на экране сайта появится сообщение с просьбой проверит наличие письма, как указано на рисунке 5.8.

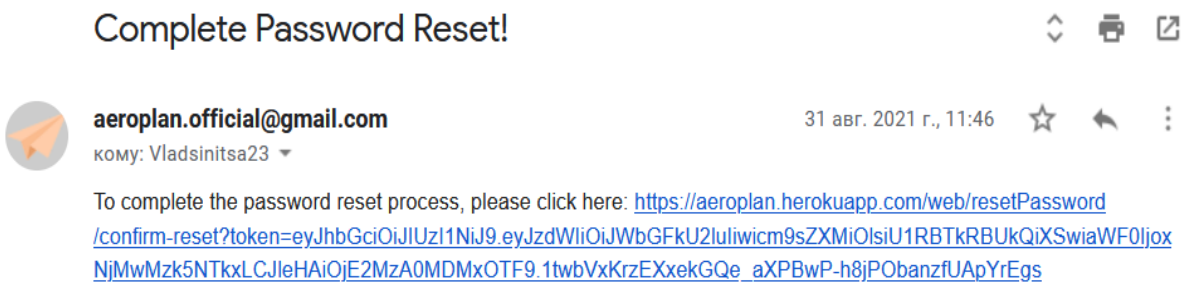

Рисунок 5.7 – Письмо подтверждения сброса пароля

В противном случае будет получено сообщение об ошибке авторизации и клиента попросят повторить попытку ввода электронной почты еще раз.

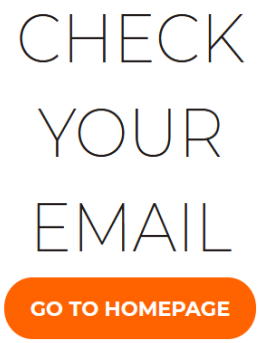

Рисунок 5.8 – Экран с сообщением об оправке письма

После того, как пользователь перейдет по ссылке из письма, система безопасности приложения еще раз проверит: действительно ли это пользователь дал согласие на смену пароля и, в случае успеха, перенаправит автоматически на экран смены пароля, как показано на рисунке 5.9.

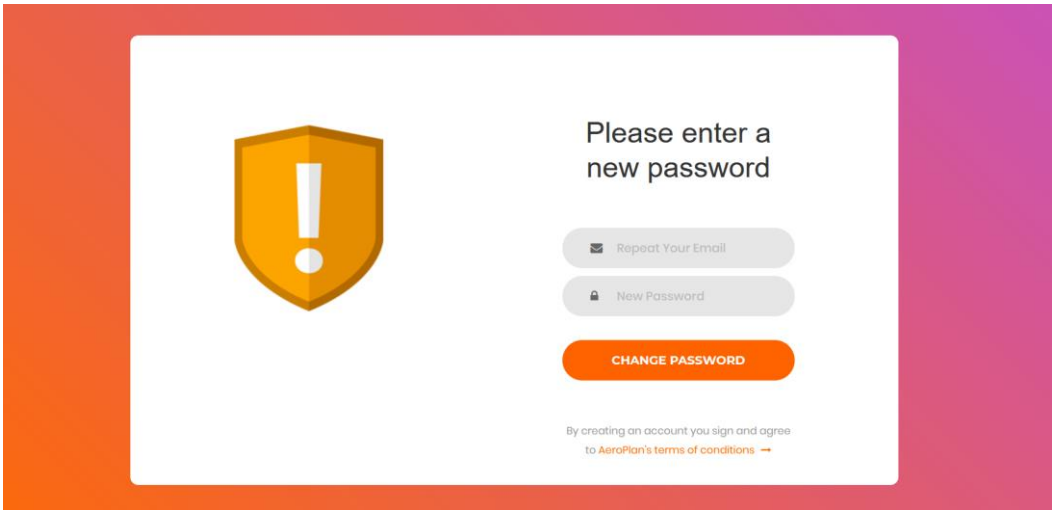

Рисунок 5.9 – Экран смены пароля

5.8 Реализация системы безопасности приложения

В мире растет количество мобильных приложений. При этом сохраняется тенденция увеличения хакерских атак с целью хищения пользовательских данных: банковских карт, пользовательской конфиденциальной информации и паролей. Поэтому вопрос безопасности приложения является основным при разработке.

На 78-й конференции 2021 года мной был коротко изложен материал о статистики уязвимостей мобильных и серверных приложений, распространенных угроз и методы их предотвращению, а также достаточно подробно разобраны основы использования технологий JWT при разработке качественной авторизации и аутентификации. Поэтому ряд определений (таких как токен, Access token, Refresh token и т.д.) и замечаний будет опущен с отсылкой на данный источник [19].

В приложении AeroPlan существует четыре роли пользователей при их авторизации в системе для предоставления прав доступа к определённым ресурсам:

1) Anonymous – данная роль присваивается пользователю, который только скачал приложение и не посчитал необходимым для себя на тот момент регистрироваться в нем. Такой пользователь будет обладать возможностью выполнять действия, предусмотренные MVP, но при этом возможность приобретать биллеты, бронировать отели и т.д. он теряет. Для этого ему будет необходимо пройти этап короткой регистрации.

2) Standard – пользователь, прошедший стандартную регистрацию в системе AeroPlan и обладает доступом к расширенным возможностям приложения. По данному пользователю ведется подробный анализ его предпочтений (персонализация), в результате чего приложение сможет давать определенные рекомендации согласно его желаниям. Покупка биллетов, бронирование номеров отелей и другие возможности доступны такому пользователю после привязки его банковской карты.

3) Google – пользователь, прошедший регистрацию через свой google-аккаунт. Пользователь обладает такими же возможностями в приложении, что и пользователь, прошедший через стандартную регистрацию, однако в данном случае пароль не требуется. В случае его повторного входы в систему (login), клиенту придется вновь использовать свой google-аккаунт для этого.

4) Privileged – пользователь, оформивший подписку на платный контент и функционал приложения. Такой пользователь обладает возможностью использовать все возможности приложения AeroPlan.

Разделение прав доступа к определенным ресурсам осуществляется фреймворком Spring Security. Имея определенный конфигурационный класс, в

котором прописаны условия по каждой роли пользователя, система фреймворка запрещает обращение к тем методам и функциям, права на которое не имеет пользователь [16].

Разделение прав осуществляется с помощью информации, полученной при взаимодействии технологий JWT и Spring Security. JWT предоставляет данные о пользователе в системе посредством токена, передаваемым тем по сети, а именно Access Token, работа которого представлена на рисунке 5.10.

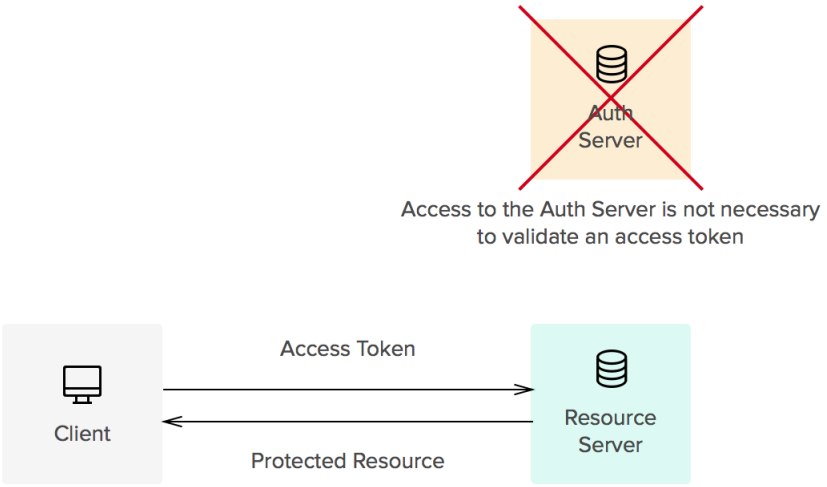

Рисунок 5.10 – Работа токена доступа

Токены доступа несут в себе необходимую информацию для прямого доступа к ресурсу. когда клиент передает такой токен серверу, управляющему ресурсом, этот сервер может прочитать информацию, содержащуюся в токене, и сам решить, авторизован пользователь или нет (никаких проверок на сервере авторизации не требуется). Это одна из причин, по которой токены должны быть подписаны (например, с использованием JWS).

Токены доступа обычно имеют короткий срок действия. Такой подход предусмотрен по той причине, если злоумышленник завладеет данным токеном, он сможет получить доступ к ресурсу под видом настоящего пользователя, однако на достаточно короткое время. И по истечению срока действия токена (когда он станет недействителен), злоумышленник потеряет возможность пользоваться сервисом. Чем меньше время использования, тем меньше вреда нанесет злоумышленник, обладая таким доступом.

Для обновления токена доступа предусмотрен токен обновления. Токены обновления несут информацию, необходимую для получения нового токена доступа. Другими словами, всякий раз, когда требуется access token для доступа к определенному ресурсу, клиент может использовать токен обновления для его получения, который выдается сервером аутентификации. Схема представлена на рисунке 5.11. Общие варианты использования включают получение новых токенов доступа после истечения срока действия старых или получение доступа

к новому ресурсу в первый раз. Токены обновления также могут истекать, но они довольно долговечны. К refresh token обычно предъявляются строгие требования к хранению, чтобы гарантировать отсутствие утечки.

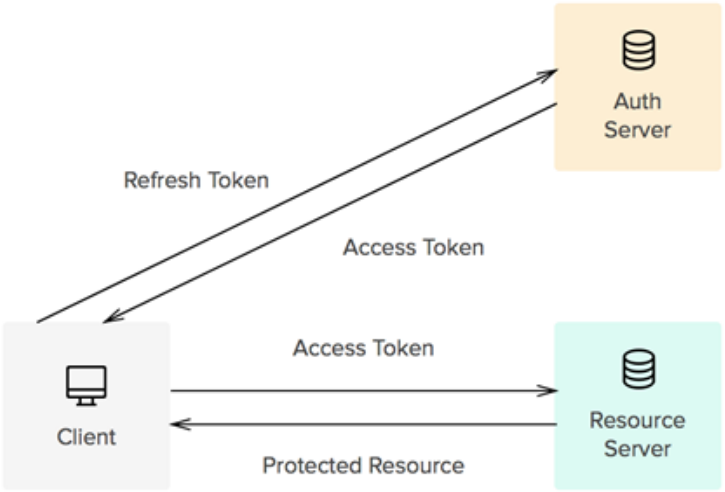

Рисунок 5.11 – Работа токена обновления

По истечению срока действия токена обновления, а также при желании пользователя его обновления, клиент приложения производит процесс идентификации своей личности по средству использования своего google-аккаунта или с использованием username и пароля, после чего система проверяет на действительность введение данные, ищет пользователя в системе и, в случае успешного аутентификации, обновляет оба токена и предоставляет доступ к системе.

Таким образом решение использование сразу нескольких токенов: доступа и обновления – повышает безопасность работы и позволяет сократить время ожидания от сервера.

5.9 Основные результаты и выводы

В ходе реализации основной функциональности сервиса был задействован широкий спектр методов и принципов для разработки высококачественного программного обеспечения, а также современные технологии, решающие поставленные задачи.

По результатам ручного и автоматизированного тестирования разработанная функциональность приложения выполняет все назначенные ей требования, а значит текущее состояние продукта готово к эксплуатации пользователями, дальнейшему развитию и продвижению.

ЗАКЛЮЧЕНИЕ

Данная дипломная работа демонстрирует полный цикл разработки программного продукта начиная от формирование самой идеи, проработки требований и архитектуры и заканчивая разработкой дизайнерской, визуальной составляющих, а главное – созданием полного серверного взаимодействия и тестированием его работы.

Подводя итог, в результате выполнения работы:

1) Было выполнено формирование целей, задач и идейной составляющей туристического сервиса AeroPlan, а также его коммерческих и маркетинговых аспектов.
2) Изучены имеющиеся реализации и выполнен анализ современных подходов получения требуемых данных в проектируемой области.
3) Изучены основные методы и технологии разработки высококачественного программного обеспечения. Выявлены интересующие архитектурные и методологические принципы разработки.
4) Сформированы требования к приложению и к системе в целом. Выбраны технологии и паттерны их разрешения.
5) Спроектирована и реализована архитектурная модель взаимодействия сервера приложения с внешними и внутренними компонентами.
6) Спроектирована и реализована база данных с использованием нескольких СУБД, проведена ее нормализация.
7) Реализовано подключение к API сторонних сервисов. Сформированы архитектурные и программные решения по взаимодействию с ними. Разработаны собственные API для предоставления широкого спектра возможности пользователям сервиса AeroPlan.
8) Выполнена реализация дизайнерских решений и визуального представления под мобильные операционные системы, а также WEB-интерфейс под информационные и регистрационные нужды.
9) Реализованы технические решения, выполняющие установленные системе требования; прописана бизнес-логика, а также реализовано взаимодействие с клиентской стороной приложения. Текущая функциональность была тщательно протестирована программными методами как для серверной части приложения, так и для ее мобильных составляющих.
10) Результаты работы докладывалась на 78-й научной конференции студентов и аспирантов БГУ, которая опубликована в сборнике материалов 78-й научной конференции.

Таким образом, в данной работе была рассмотрена визуальная и программная реализация основной функциональности приложения AeroPlan, которая выполняется посредством современных и наиболее актуальных технологий разработки.

Глубокий анализ в сфере безопасности, позволил реализовать все современные подходы, как защиты пользовательской информации от злоумышленников, так и сохранность данных и их доступность, а работа над увеличением производительности и отказоустойчивости приложения поможет обеспечить доверие пользователей и надежность работы всего сервиса.

Детальное проектирование и планирование мобильного приложения позволяет создать уникальный сервис, который получит возможность распространиться по всему миру, набрать популярность и стать полезным инструментом в руках любого туриста, планирующего свою поездку.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Исследования Яндекса [Электронный ресурс]. – Режим доступа к ресурсу: https://hoteliernews.ru/turizm-v-2020-ot-kaliningrada-do-kamchatki
2. Брауде, Э. Технология разработки программного обеспечения. – СПб.: Питер, 2004.
3. Гуськова, О.И. Объектно-ориентированное программирование в Java. – МПГУ: Москва, 2018
4. Надежность, доступность и отказоустойчивость сайтов и веб-приложений [Электронный ресурс]. – Режим доступа к ресурсу: https://web-creator.ru/articles/about_failover
5. Интернационализация и локализация [Электронный ресурс]. – Режим доступа к ресурсу: https://qalight.ua/baza-znaniy/internatsionalizatsiya-i-lokalizatsiya
6. Волушкова, В.Л. Архитектурные решения java для доступа к данным: учеб. пособие. – Тверь: Твер. гос. ун-т, 2019. – 137 с.
7. Ричардсон, К. Микросервисы. Паттерны разработки и рефакторинг. – СПб.: Питер, 2019. – 544 с.: ил. — (Серия «Библиотека программиста»).
8. Скакун В. В. Системы управления базами данных: пособие v. С42 / авт.-сост. В. В. Скакун. – Минск: БГУ, 2007. – 116 с.
9. Программы подключения сервисов Travelpayouts [Электронный ресурс]. – Режим доступа к ресурсу: https://app.travelpayouts.com/programs
10. Документация подключения сервиса Numbeo [Электронный ресурс]. – Режим доступа к ресурсу: https://www.numbeo.com/api/doc.jsp
11. Дизайн-система [Электронный ресурс]. – Режим доступа к ресурсу: https://habr.com/ru/post/524554
12. Design web services using the REST paradigm [Электронный ресурс]. – Режим доступа к ресурсу: https://www.codecademy.com/article/what-is-rest
13. Web development: JSON [Электронный ресурс]. – Режим доступа к ресурсу: https://www.w3schools.com/whatis/whatis_json.asp
14. A type-safe HTTP client for Android and Java [Электронный ресурс]. – Режим доступа к ресурсу: https://square.github.io/retrofit
15. A Guide to OkHttp [Электронный ресурс]. – Режим доступа к ресурсу: https://www.baeldung.com/guide-to-okhttp
16. Web MVC Framework [Электронный ресурс]. – Режим доступа к ресурсу: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html
17. Spring MVC – основные принципы [Электронный ресурс]. – Режим доступа к ресурсу: https://habr.com/ru/post/336816
18. Business Logic [Электронный ресурс]. – Режим доступа к ресурсу: https://www.investopedia.com/terms/b/businesslogic.asp
19. Проблемы безопасности серверной части Android приложений и пути их решения [Электронный ресурс]. – Режим доступа к ресурсу: https://nirs.bsu.by/index.php/events/conferences/konferentsiya-bgu

ПРИЛОЖЕНИЕ А

Основной инициализирующий алгоритм

Основной алгоритм, инициализирующий путешествие пользователя по назначенным запросам и фильтрам:

1) Контролер, принимающий запрос от пользователя, первоначально идентифицируя его. После возвращения результата от сервисной части приложения, контроллер преобразует данный результат в ответ, который далее отправляет пользователю.

```
public ResponseEntity<TravelInitialResponse>initTravel(@RequestHeader("access-token") String accessToken,
                                                       @RequestBody TravelInitialRequest request) {
    UserEntity user = userService.findByToken(accessToken);
    TravelHistoryEntity result = initialTravelService.generate(user, request);
    TravelInitialResponse response = travelInitialResponseMapper.travelHistoryEntityToDto(result);
    response.setCoordinates(request.getWayPoints().get(1).getCoordinates());
    return new ResponseEntity<>(response, HttpStatus.OK);
}
```

2) Пример запроса на создание путешествия в формате JSON:

```
{
   "wayPoints": [
            {
                    "code": "MSQ",
                    "name": "Minsk",
                    "type": "city",
                    "countryName": "Belarus",
                    "countryCode": "BY",
                    "mainAirportName": null,
                    "coordinates": {
                            "lon": "27.5666667",
                            "lat": "53.9"
                    }
            },
            {
                    "code": "AYT",
                    "name": "Antal",
                    "type": "city",
                    "countryName": "Turkey",
                    "countryCode": "TR",
                    "mainAirportName": null,
                    "coordinates": {
                            "lon": "30.80135",
                            "lat": "36.89928"
                    }
            }
   ],
   "departDate":"2022-05-25",
   "returnDate":"2022-06-10",
   "passengers": {
            "adults":2,
            "children":0,
            "infants":0
```

```
    },
    "transferType":"PLANE",
    "direct":true,
    "currency":"usd",
    "filters": {
            "hotelStars":[3],
            "distanceFromCenter":5.0,
            "expensiveRestaurant":15.0,
            "inexpensiveRestaurant":85.0,
            "coffeePreference":"HIGH",
            "alcoholicPreference":"LOW",
            "goingOutPreference":"NEVER",
            "takingTaxi":"2/1",
            "publicTransport":"2/1",
            "carRent":"Car sharing"
    }
}
```

3) Ответ, получаемый от сервисной части приложения в формате JSON:

```
{
    "id": "d4d92e46-cc83-44cb-a50a-d0b23a7fa520",
    "travelName": "Minsk to Antal",
    "description": "An unforgettable journey to Antal",
    "photoUrl": "https://lifeglobe.net/x/entry/1403/Rome.jpg",
    "dateStart": "2022-05-25",
    "dateEnd": "2022-06-10",
    "totalCost": 2241.005,
    "currency": "usd",
    "coordinates": {
            "lon": "30.80135",
            "lat": "36.89928"
    }
}
```

4) Метод (функция) генерации запросов к сервисам-партнерам, обеспечивающий дальнейшую обработку ответов от названных сервисов, а также реализующий функционал по созданию путешествия (вид и структура метода укорочены).

```
public TravelHistoryEntity generate(UserEntity user, TravelInitialRequest request) {
    AviasalesTicketsApiRequest aviasalesRequest = new AviasalesTicketsApiRequest();
    aviasalesRequest.setCurrency(request.getCurrency());
    aviasalesRequest.setOrigin(request.getWayPoints().get(0).getCode());
    aviasalesRequest.setDestination(request.getWayPoints().get(1).getCode());
    aviasalesRequest.setDepartDate(request.getDepartDate());
    aviasalesRequest.setReturnDate(request.getReturnDate());
    aviasalesRequest.setDirect(request.getDirect());
    aviasalesRequest.setLimit("10");
    aviasalesRequest.setSorting("price");

    HotellookPriceApiRequest hotellookRequest = new HotellookPriceApiRequest();
    hotellookRequest.setLocation(request.getWayPoints().get(1).getName());
    hotellookRequest.setCheckIn(request.getDepartDate());
    hotellookRequest.setCheckOut(request.getReturnDate());
```

```
            hotellookRequest.setCurrency(request.getCurrency());

  ApiResponses responses = new ApiResponses();
responses.setAviasalesTicketsResponse(aviasalesApiService.getTicketsForSpecificDates(aviasalesRequest))
responses.setHotellookPriceResponses(hotellookApiService.getDisplaysPriceOfHotelRooms(hotelRequest))
responses.setNumbeoPriceResponse(numbeoApiService.getCityPrices(request.getWayPoints().get(1))
return generateTravelService.generateTravelHistory(user, responses, request);
 }
```

5) Методы сбора, анализа, подсчета и сохранение данных путешествия. Представлена только два основных метода, принимающие подсчитанные и обработанные данные всех составных частей путешествия.

```
        public TravelHistoryEntity generateTravelHistory(UserEntity user, ApiResponses apiResponses,
TravelInitialRequest request) {
            TravelHistoryDto travel = new TravelHistoryDto();
            travel.setTravelName(request.getWayPoints().get(0).getName()+"to"                    +
request.getWayPoints().get(1).getName());
            travel.setDescription("An unforgettable journey to " + request.getWayPoints().get(1).getName());
            travel.setPhotoUrl("https://lifeglobe.net/x/entry/1403/Rome.jpg");
            travel.setCurrency(request.getCurrency());
            travel.setDateStart(new SimpleDateFormat("yyyy-MM-dd").parse(request.getDepartDate()));
            travel.setDateEnd(new SimpleDateFormat("yyyy-MM-dd").parse(request.getReturnDate()));
            TravelHistoryEntity travelHistory = travelHistoryService.create(travel, user.getId());
            travelHistory.setCreated(new Date());
            travelHistory.setUpdated(new Date());
            travelHistory.setStatus(StatusType.ACTIVE);
            CityTripEntity trip = generateCityTrip(travelHistory, apiResponses, request);
            travelHistory.setTotalCost(trip.getTotalCost());
            return travelHistoryService.edit(travelHistory, user.getId());
        }

        public CityTripEntity generateCityTrip(TravelHistoryEntity travel, ApiResponses apiResponses,
TravelInitialRequest request) {
            CityTripDto trip = new CityTripDto();
            trip.setCity(apiResponses.getNumbeoPriceResponse().getName());
            trip.setCountry(apiResponses.getNumbeoPriceResponse().getName());
            trip.setDescription("Great time in " + request.getWayPoints().get(1).getName());
            trip.setDateStart(travel.getDateStart());
            trip.setDateEnd(travel.getDateEnd());
            trip.setPhotoUrl("https://lifeglobe.net/x/entry/1403/Rome.jpg");
            trip.setPassengers(calculateNumberOfMember(request));
            CityTripEntity tripEntity = cityTripService.create(trip, travel.getId());
            CityTripEntity forCalculate = new CityTripEntity();
            forCalculate.setApartments(generateApartment(tripEntity, apiResponses, request));
            forCalculate.setEntertainments(generateEntertainment(tripEntity, apiResponses, request));
            forCalculate.setExcursions(generateExcursion(tripEntity, apiResponses, request));
            forCalculate.setInsurances(generateInsurance(tripEntity, apiResponses, request));
            forCalculate.setMeals(generateMeal(tripEntity, apiResponses, request));
            forCalculate.setTransfers(generateTransfer(tripEntity, apiResponses, request));
            forCalculate.setTransports(generateTransport(tripEntity, apiResponses, request));
            Double totalCost = calculateTravelService.calculateTripTotalCost(forCalculate);
            tripEntity.setTotalCost(totalCost);
            return cityTripService.edit(tripEntity, travel.getId());
        }
```