

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

Факультет
вычислительной техники

Кафедра
МО и ПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Мобильное приложение продажи транспортных услуг
автовокзала**

Студент _____ Пункевич Роман Витальевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Балашова И.Ю.
(подпись, дата) (фамилия, инициалы)

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

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

Заведующий кафедрой _____ Макарычев П.П.
(подпись) (фамилия, инициалы)

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

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

Пенза, 2020

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

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

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

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Пункевич Роман Витальевич гр. 16ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Мобильное приложение продажи транспортных услуг автовокзала

Тема утверждена приказом ПГУ № _____ от " ____ " _____ 2020 г.

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

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

1) *Назначение: мобильное приложение предназначено для просмотра расписания, покупки билетов на автобусные междугородние рейсы.*

2) *Среда выполнения: операционная система Android, начиная с версии Android 5.0.*

Средства разработки: языки программирования Java, XML.

Инструментальные средства: Android Studio, Firebase Console.

Технологии разработки: ООП, UML.

3) *Основные функции мобильного приложения продажи транспортных услуг автовокзала:*

а) регистрация в приложении;

б) авторизация в приложении;

в) просмотр расписания выбранного города (станции);

г) просмотр рейсов между двумя городами (станциями);

д) построение маршрута с пересадками при отсутствии прямого сообщения;

е) ведение клиентской базы;

ж) покупка билета на рейс;

з) выбор посадочного места в транспорте при покупке билета;

и) просмотр купленных билетов;

к) возврат купленных билетов;

4. Перечень подлежащих разработке вопросов

Анализ предметной области

Анализ требований

Проектирование

Реализация

Контроль качества ПО:

– Тестирование

– Анализ качества кода

План разработки и оценка бюджета

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований	10	13.04.2020 – 20.04.2020	
2	Проектирование	25	21.04.2020 – 01.05.2020	
3	Реализация	35	04.05.2020 – 29.05.2020	
4	Контроль качества	15	01.06.2019 – 05.06.2019	
5	План разработки бюджета	5	06.06.2019 – 09.06.2019	
6	Оформление пояснительной записки	10	10.06.2020 – 13.06.2020	

Дата выдачи задания «___» _____ 2019 г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 100 листов, 46 рисунков, 13 использованных источников.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ПРОДАЖИ ТРАНСПОРТНЫХ УСЛУГ АВТОВОКЗАЛА, АВТОВОКЗАЛ, БАЗА ДАННЫХ, JAVA, КЛИЕНТ, СЕРВЕР, FIREBASE.

Объект разработки – мобильное приложение продажи транспортных услуг автовокзала, состоящая из базы данных, которая содержит данные о пользователях, станциях и расписании, а также из мобильного клиентского приложения для взаимодействия с данными из базы данных.

Цель работы – разработать мобильное приложение продажи транспортных услуг автовокзала, предназначенное для просмотра пользователем расписания, а также осуществления им покупки билетов на выбранные рейсы.

Технология разработки – для создания серверной части используется Firebase от компании Google, для создания клиентской части используется среда программирования Android Studio и языки Java и XML. Процесс разработки мобильного приложения продажи транспортных услуг автовокзала будет основываться на методологии разработки RUP.

Результаты работы – в ходе выполнения выпускной квалификационной работы было создано мобильное приложение продажи транспортных услуг автовокзала, имеющее клиент-серверную архитектуру. Все требования к разработке системы соблюдены.

					ПГУ 09.03.02 – 8БР161.23 ПЗ							
Изм	Лист	№ докум.	Подпись	Дата	Мобильное приложение продажи транспортных услуг автовокзала Пояснительная записка			Лит.		Лист	Листов	
Разраб.		Пункевич Р. В.								3	100	
Пров.		Балашова И. Ю.										
Реценз.												
Н. контр.		Такташкин Д. В.										
Утв.												

Содержание

Введение	6
1 Анализ предметной области, требований и бизнес процессов	8
1.1 Анализ предметной области	8
1.2 Анализ мобильных приложений автовокзалов.....	10
1.3 Постановка задачи.....	17
1.4 Анализ требований	18
1.5 Анализ бизнес-процессов предметной области.....	21
1.6 Средства разработки мобильного приложения	26
2 Проектирование мобильного приложения.....	28
2.1 Базовая архитектура мобильного приложения	28
2.2 Технология разработки мобильного приложения	29
2.3 Проектирование структуры данных мобильного приложения.....	30
2.3.1 Логическая модель базы данных мобильного приложения	31
2.3.2 Физическая модель базы данных мобильного приложения.....	32
2.4 Проектирование программных средств	33
2.4.1 Проектирование мобильного приложения.....	33
2.4.2 Диаграмма последовательности мобильного приложения	34
2.4.3 Диаграмма деятельности мобильного приложения.....	36
2.4.4 Диаграмма кооперации мобильного приложения	38
2.4.5 Диаграмма состояний мобильного приложения.....	39
2.5 Проектирование графического интерфейса мобильного приложения.....	40
3 Реализация программного обеспечения мобильного приложения.....	50
3.1 Диаграмма компонентов мобильного приложения	50

3.2 Представление развертывания мобильного приложения.....	51
4 Контроль качества программного обеспечения мобильного приложения ..	53
4.1 Функциональное тестирование мобильного приложения	53
4.1.1 Определение классов эквивалентности.....	53
4.1.2 Анализ граничных условий	54
4.1.3 Построение функциональной диаграммы.....	56
4.1.4 Парное тестирование.....	57
4.2 Метрики программного кода проекта	59
4.3 Планирование проекта и оценка бюджета	60
Заключение	68
Список использованных источников	69
Приложение А. Глоссарий.....	71
Приложение Б. Листинг кода программы клиентского приложения на языке Java.....	73
Приложение В. Результаты тестирования	90

Введение

В Российской Федерации из-за огромной площади, неравномерного распределения населения и производства роль транспорта в обеспечении нормального функционирования хозяйства с каждым годом неуклонно возрастает. Важную роль в транспортной системе как экономической составляющей страны играют пассажирские перевозки. Согласно опубликованной Министерством транспорта Российской Федерации статистике [1], основными видами междугородних и пригородных транспортных сообщений являются железнодорожный и автобусный транспорт.

В последние годы автобусные перевозки занимают покинутую во многих регионах железнодорожным транспортом нишу пригородных пассажирских перевозок. Такая же картина наблюдается и в междугородних перевозках, где уже который год два этих вида транспорта практически поровну делят общий объем перевозок этого вида сообщения, причем в некоторых регионах автобусные перевозки занимают доминирующее положение.

Естественным образом информационные технологии внедряются в транспортный сектор экономики страны, и автобусные пригородные и междугородние перевозки не являются исключением. Мобильные и веб-приложения уже давно доказали свою эффективность в сфере заказа такси, просмотра расписания движения городского общественного транспорта и стали неотъемлемой частью жизни любого городского жителя. В настоящее время на рынке программного обеспечения, обеспечивающего поддержку деятельности автовокзала, наблюдается наплыв большого количества однотипных приложений, адаптированных под разные регионы, имеющих разный функционал, возможности и интерфейс. При этом существующие решения в сфере междугородних и пригородных перевозок не могут в полной мере удовлетворить потребности современного пользователя. В

первую очередь это обусловлено тем, что большинство существующих мобильных приложений автовокзала имеют ограниченную функциональность и поддерживают продажу билетов автобусных станций в пределах одного региона. Также существенным недостатком имеющихся программных решений, сказывающимся на качестве предоставляемых услуг, является отсутствие возможности построения маршрутов с пересадками между двумя населенными пунктами при отсутствии прямого рейса. Поэтому разработка мобильного приложения, которое обладало бы такой возможностью и также содержало в себе все преимущества существующих решений, является задачей своевременной и актуальной.

Целью выпускной квалификационной работы является разработка мобильного приложения продажи транспортных услуг автовокзала.

Для достижения указанной цели были поставлены следующие задачи:

- проведение анализа предметной области и существующих информационных систем аналогичного назначения;
- постановка задачи на разработку мобильного приложения продажи транспортных услуг автовокзала;
- определение требований к разрабатываемому приложению;
- выполнение функционального моделирования системы;
- проектирование мобильного приложения продажи транспортных услуг автовокзала;
- реализация мобильного приложения;
- тестирование мобильного приложения продажи транспортных услуг автовокзала.

1 Анализ предметной области, требований и бизнес процессов

1.1 Анализ предметной области

На сегодняшний день автобусные междугородные и пригородные пассажирские перевозки занимают лидирующие позиции в этих видах транспортных сообщений, незначительно уступая железнодорожному транспорту (в междугородних перевозках 40,8% пассажиров у железнодорожного транспорта против 30% у автобусного) [1]. Такая высокая доля автобусного транспорта в структуре междугородних пассажирских перевозок связана с более низкими, по сравнению с железнодорожным транспортом, ценами проездных билетов. Также автобусные перевозки обладают более диверсифицированными предложениями касательно расписания рейсов. Например, количество автобусных рейсов «Пенза – Москва» равно девяти, в то время как железнодорожных всего два, что также может являться решающим фактором выбора типа транспорта для своей поездки пассажиром.

Такая большая доля перевезенных пассажиров в сфере пригородного и междугороднего автобусного сообщения обязывает организации, так или иначе относящихся к этой категории деятельности, иметь в своем арсенале, помимо исправной транспортной техники и команды водителей, современные технические решения продажи своих услуг.

Автовокзал имеет расписание движения своих автобусов, причем время отправления и время прибытия может часто меняться по тем или иным причинам, при возникновении такой ситуации встает вопрос о своевременном оповещении пассажира о произошедших изменениях.

Перевозчик заинтересован в получении прибыли, а прибыль в этом бизнесе получается за счет продаж билетов на рейсы. В веб и мобильных приложениях давно реализован механизм безналичной оплаты товаров и услуг, и автобусные пригородные и междугородные перевозок не являются исключением.

Автовокзал обладает большим автобусным парком, зачастую укомплектованным разными моделями, что подразумевает под собой разное количество посадочных мест в салоне и схему их расположения. Помимо этого, в мобильном приложении важно учитывать проданные и забронированные места, чтобы случайно не продать их повторно.

Далеко не все населенные пункты соединены между собой прямым транспортным сообщением, из-за чего у пассажира может сложиться неверное мнение, что на автобусе до нужной ему точки не добраться. В реальности, практически любой населенный пункт является достижимым непрямым сообщением, но для этого нужно рассчитывать варианты маршрутов. Но большое количество существующих мобильных приложений решило не реализовывать возможность построения маршрутов с пересадками, а некоторые и вовсе удаляют из своей базы данных недоступные населенные пункты, из-за чего их список становится крайне скуден.

Автовокзал, как организация, заинтересован в ведении базы данных маршрутов, перевозчиков, населенных пунктов, а также своих клиентов – пассажиров. Обладая такими источниками информации, автовокзал имеет возможность проводить грамотную маркетинговую политику.

В связи с вышесказанным прослеживается необходимость разработки мобильного приложения, позволяющего отображать актуальное расписание автобусных рейсов, осуществлять покупку билетов на рейс с возможностью выбора посадочного места, проходить процесс регистрации и авторизации в приложении, вести базу данных. Разрабатываемое приложение, в отличие от большинства существующих мобильных решений, должно поддерживать поиск автобусных маршрутов и продажу билетов для широкого перечня населенных населенных пунктов, при этом в нем должна быть реализована возможность построения маршрутов с пересадками при отсутствии прямых сообщений.

1.2 Анализ мобильных приложений автовокзалов

В настоящее время существует достаточно большое количество отечественных мобильных приложений, предоставляющие возможности отображения расписания движений автобусов пригородного и междугороднего сообщения и покупку билетов на эти рейсы. Однако в силу большой площади нашей страны и огромной длиной дорожной сети, большинство существующих приложений представляют ограниченный список городов и населенных пунктах в своих разработках, а те мобильные решения, которые все-таки предоставляют полный список населенных пунктов, не предоставляют возможности построения маршрутов с пересадками при отсутствии прямого сообщения.

Тем не менее, существующие мобильные решения обладают набором базовых функций, которыми удовлетворяют потребности современного пользователя. Самыми популярными приложениями междугородних и пригородных перевозок в магазинах приложений являются: «Автовокзал Онлайн», «Автовокзал Easy Way», «Автобусы tutu», «Автовокзалы.ру», «НА АВТОБУС».

Все перечисленные приложения объединяет следующий набор функционала:

- просмотр прямых маршрутов между населенными пунктами;
- бронирование и покупка билета;
- отслеживание изменений в расписании.

Мобильное приложение «Автовокзал Онлайн» помимо всего перечисленного имеет развитый механизм бронирования мест в салоне автобуса, который включает в себе детальную схему салона, нумерацию мест и отображение статуса места (куплено, забронировано, свободно). Главные недостатки этого мобильного решения – жесткая географическая привязка (приложение работает только в Иркутской области и Республике Бурятия), а также отсутствие механизма возврата билета.

Интерфейс приложения «Автовокзал Онлайн» продемонстрирован на рисунке 1.

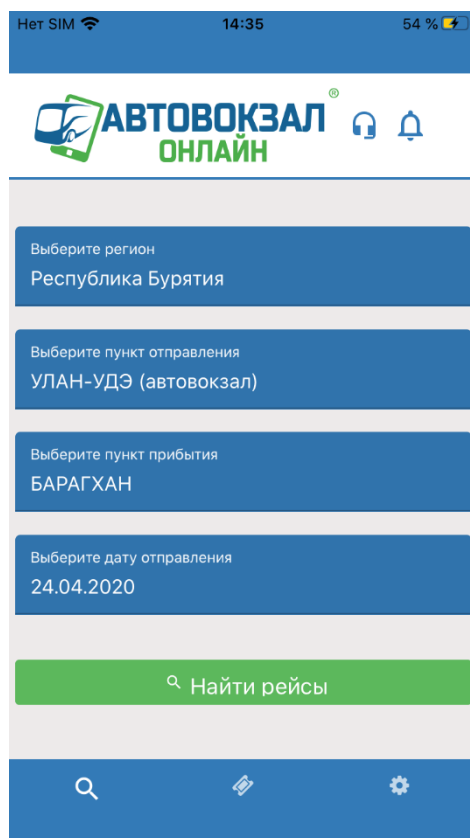


Рисунок 1 – Интерфейс приложения «Автовокзал Онлайн»

Приложение «Автовокзал Easy Way» схоже по функционалу с рассмотренным выше. Оно также имеет географические ограничения – приложение рассчитано на работу в южных регионах Уральского федерального округа, а также в некоторых регионах Центрального федерального округа. Реализован механизм регистрации и авторизации. Просмотр посадочных мест, отображение расписания рейсов конкретного города, построение маршрутов с пересадками отсутствуют. Из преимуществ данного мобильного решения можно выделить удобный процесс покупки билета, а также реализованный механизм возврата билетов. Графический интерфейс мобильного приложения «Автовокзал Easy Way» представлен на рисунке 2.

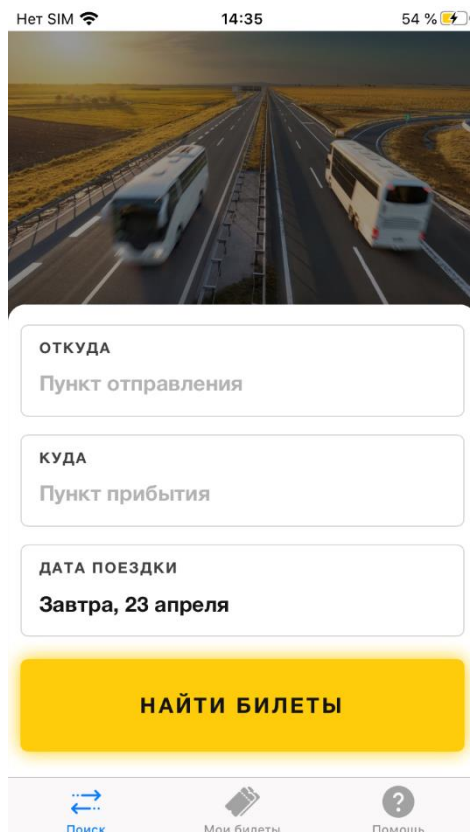


Рисунок 2 – Интерфейс приложения «Автовокзал EasyWay»

Приложения «Автобусы tutu», «Автовокзалы.ру» и «НА АВТОБУС» можно назвать приложениями-агрегаторами. Они аккумулируют в себе информацию по всем рейсам в стране и предоставляют свой функционал для покупки билетов. Самым мощным по функционалу приложением среди всех аналогов является «Автобусы tutu». Это мобильное решение позволяет:

- проходить процесс регистрации и авторизации в приложении;
- полный охват городов и населенных пунктов страны;
- бронирование и покупка билетов;
- уведомление об изменениях в движении рейсов, на которые пассажиром был куплен билет;
- возврат билета.

В виду большой базы городов и рейсов, в данной системе примитивно реализована возможность просмотра посадочных мест в салоне. Фактически, пользователю предоставляются только номера сидений без схемы салона, что

затрудняет выбор и делает его неудобным. Главное меню приложения «Автобусы tutu» представлено на рисунке 3.

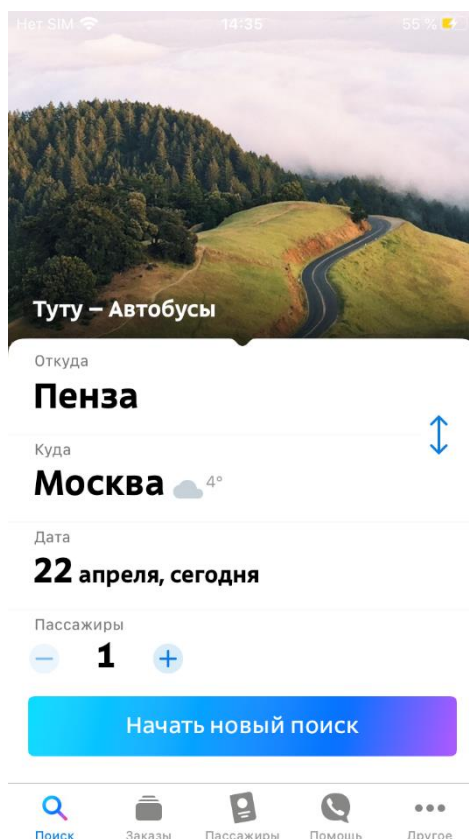


Рисунок 3 – Интерфейс приложения «Автобусы tutu»

Приложение «Автовокзалы.ру» агрегирует в себе информацию по всем автовокзалам страны, позволяет осуществлять покупку и делать возврат билетов, просматривать прямые маршруты между городами. Оно уступает в популярности приложению «Автобусы tutu», хотя и обладают схожим функционалом. Из явных недостатков данного приложения можно выделить то, что фактически отсутствует механизм аутентификации, который представлен лишь в виде привязки экземпляра приложения к конкретному устройству. То есть, если пользователь купил билет через конкретный смартфон, то просмотреть свой профиль он сможет только с него. При покупке билета в этом приложении отображается лишь примерное количество свободных мест без возможности выбора конкретного посадочного места. Интерфейс приложения «Автовокзалы.ру» представлен на рисунке 4.

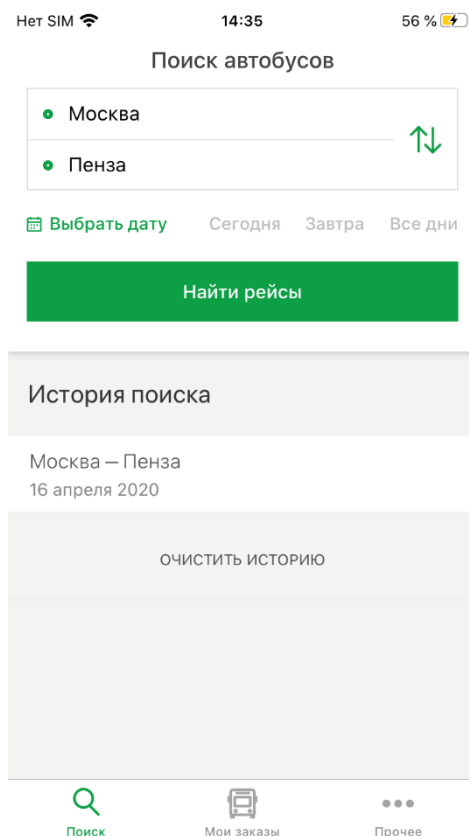


Рисунок 4 – Интерфейс приложения «Автовокзалы.ру»

Приложение «НА АВТОБУС» обладает схожим с приложением «Автовокзалы.ру» функционалом. Главным отличием является отсутствие информации о количестве свободных мест на выбранном рейсе. Как следствие, отсутствуют возможность выбора конкретного посадочного места, построение маршрута с пересадками.

В отличие от «Автовокзалы.ру» в приложении реализован механизм аутентификации. Из общих достоинств можно выделить большую базу городов, возможность возврата билета, удобный процесс покупки билета, отслеживание изменений в расписании. Одной из уникальных особенностей данного программного решения является наличие механизма связи с технической службой поддержки через встроенный в приложение текстовый чат. Графический интерфейс мобильного приложения «НА АВТОБУС» представлен на рисунке 5.

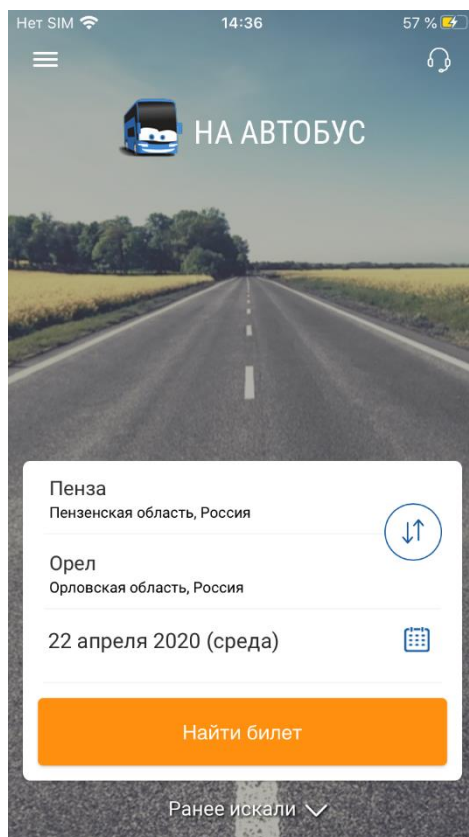


Рисунок 5 – Интерфейс приложения «НА АВТОБУС»

В процессе анализа представленных мобильных решений были выделены следующие функциональные характеристики:

- просмотр расписания (всех рейсов из выбранного пункта);
- просмотр прямых маршрутов;
- построение расписания маршрутов с пересадками;
- предложение обратного маршрута;
- просмотр свободных мест;
- бронирование билета;
- покупка билета;
- отслеживание изменений в расписании;
- обмен, возврат билета;
- ведение базы пассажиров;
- большая база городов.

Для наглядности сравнения систем по функциональным возможностям сведем результаты анализа в таблицу 1.

Таблица 1 – Сравнение по функциональным характеристикам

Характеристика	Автовокзал Онлайн	Автовокзал Easy Way	Автобусы Tutu	Автовокзалы.ру	НА АВТОБУС
Просмотр расписания	Нет	Нет	Нет	Нет	Нет
Просмотр прямых маршрутов	Да	Да	Да	Да	Да
Построение расписания маршрутов с пересадками	Нет	Нет	Нет	Нет	Нет
Предложение обратного маршрута	Нет	Нет	Нет	Нет	Нет
Просмотр свободных мест	Да	Нет	Только список номеров мест	Только примерное количество свободных мест	Нет
Бронирование билета	Да	Да	Да	Да	Да
Покупка билета	Да	Да	Да	Да	Да
Отслеживание изменений в расписании	Да	Да	Да	Да	Да
Обмен, возврат билета	Нет	Да	Да	Да	Да
Ведение базы пассажиров	Да	Да	Да	Нет	Да
Большая база городов	Нет	Нет	Да	Да	Да

В ходе анализа аналогичных по функциональному назначению мобильных приложений установлено, что все решения обладают базовыми необходимыми функциями для работы приложения – просмотр прямых маршрутов и покупка билетов. Некоторые из проанализированных приложений не имеют механизма возврата билета, что является очень

важным аспектом для систем такого типа. Также можно отметить, что разработчики приложений данного профиля не придают большого значения вопросу выбора посадочных мест в транспорте, во многих приложениях выбор отсутствует как таковой, в других он представлено обычным списком номеров, чья полезность остается под вопросом.

Если систематизировать общие недостатки, то получим следующий список:

- отсутствие просмотра рейсов, исходящих с выбранного автовокзала (населенного пункта);
- отсутствие механизма построения маршрутов с пересадками между городами при отсутствии прямого сообщения;
- отсутствие предложения пользователю оформить обратный билет;
- слабая реализация просмотра посадочных мест в транспорте;
- отсутствие учета льготных категорий пассажиров;
- малое географическое покрытие.

Выявленные недостатки анализируемых приложений подчеркнули необходимость разработки решения, в котором бы отсутствовали перечисленные недочеты.

Таким образом, было принято решение реализовать собственное мобильное приложение продаж транспортных услуг автовокзала, обладающего преимуществами и лишенного недостатков рассмотренных аналогов.

1.3 Постановка задачи

Данная выпускная квалификационная работа посвящена разработке мобильного приложения продажи транспортных услуг автовокзала. Приложение создается для автоматизации процесса просмотра расписания, построения маршрутов и покупки билетов на рейсы в онлайн режиме с помощью смартфона на платформе Android.

Так как программа ориентирована на пользователя, она должна обладать удобным и легко осваиваемым интерфейсом, обеспечивающий простой и комфортный доступ функциям мобильного приложения.

Для надежного хранения и обработки информации необходимо разработать базу данных (БД), содержащую информацию о пользователях, маршрутах, станциях и купленных пользователем билетов. Мобильное приложение должно обладать следующим функционалом:

- регистрация и авторизация пользователей;
- просмотр страницы профиля пользователя;
- просмотр расписания выбранной станции на выбранную дату отправления;
- просмотр маршрутов между двумя wybranными станциями на выбранную дату отправления;
- построение маршрута с пересадками при отсутствии прямого сообщения между двумя станциями;
- покупка билета на выбранный рейс;
- возврат купленного билета;
- просмотр списка купленных билетов.

В процессе проектирования и разработки необходимо предусмотреть модуль автоматического или автоматизированного добавления новых рейсов и станций администратором приложения.

1.4 Анализ требований

Для отражения функциональных требований к разрабатываемому мобильному приложению была спроектирована модель в форме диаграммы вариантов использования в нотации UML. Диаграмма прецедентов является исходным концептуальным представлением системы в процессе ее проектирования и разработки [2]. Диаграмма вариантов использования представлена на рисунке 6.

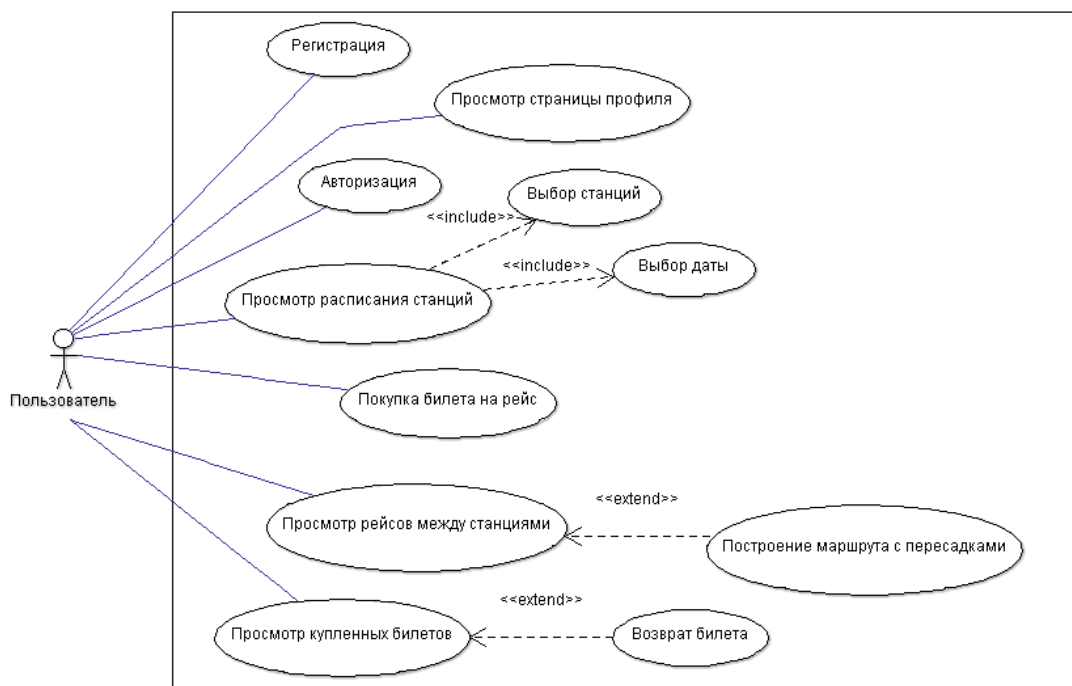


Рисунок 6 – Диаграмма вариантов использования

Для отображения связи между действующими лицами и вариантами использования на диаграмме используются отношения ассоциации, включения, расширения и обобщения.

Приложение должно обеспечивать выполнение следующих функций:

- регистрация;
- авторизация;
- просмотр страницы профиля;
- просмотр расписания станций (включая выбор станции и даты отправления);
- просмотр рейсов между двумя станциями;
- покупка билета на выбранный рейс;
- просмотр купленных билетов.

Краткое описание выделенных на диаграмме действующих лиц приведено в таблице 2.

Таблица 2 – Описание действующих лиц

Название	Описание
Пользователь	Использует все представленные на диаграмме функции приложения после успешной авторизации.

Основными прецедентами приложения можно назвать «Просмотр рейсов между двумя станциями» и «Покупка билета на рейс», именно реализация этих функций является наиболее важной задачей в данной работе, поэтому необходимо рассмотреть их более подробно.

Диаграмма варианта использования «Просмотр рейсов между двумя станциями» содержит функции: ввод пункта отправления, ввод пункта прибытия, выбор даты, просмотр результатов, покупка билета на рейс и просмотр маршрутов с пересадками. Диаграмма представлена на рисунке 7.

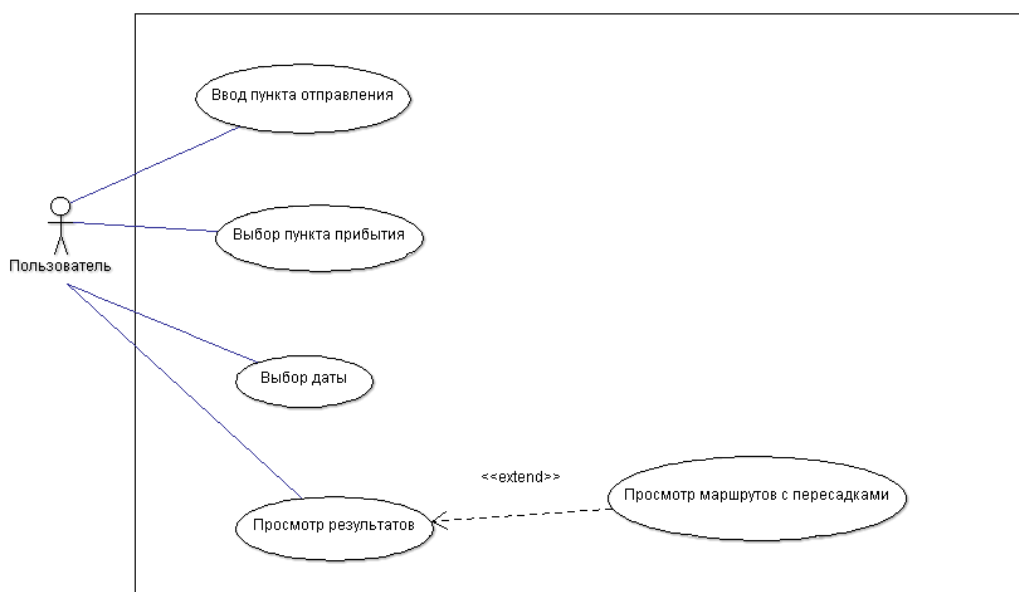


Рисунок 7 – Диаграмма вариантов использования «Просмотр рейсов между двумя станциями»

«Покупка билета на рейс» – приложение позволяет пользователю купить билет на выбранный рейс в выбранную дату в онлайн режиме. Диаграмма представлена на рисунке 8.

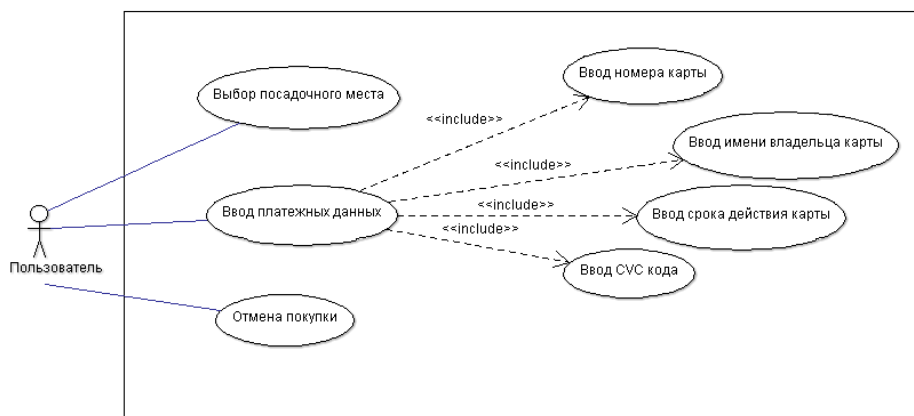


Рисунок 8 – Диаграмма вариантов использования «Покупка билета на рейс»

Диаграмма варианта использования «Покупка билета на рейс» содержит функции: выбор посадочного места, отмена покупки, а также ввод платежных данных, включающий в себя ввод номера карты, ввод имени владельца карты, ввод срока действия карты, ввод CVC кода.

1.5 Анализ бизнес-процессов предметной области

Для графического представления бизнес-процессов предметной области было использовано CASE-средство верхнего уровня AllFusion Process Modeler (BPwin). BPwin представляет собой инструмент для создания моделей, позволяющих анализировать, документировать и планировать изменения сложных бизнес-процессов [3]. Данное CASE-средство включает три стандартные методологии: IDEF0 (функциональное моделирование), DFD (моделирование потоков данных) и IDEF3 (моделирование потоков работ).

IDEF0 используется для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающие эти функции. Диаграмма бизнес-процессов в нотации IDEF0 представлена на рисунке 9.

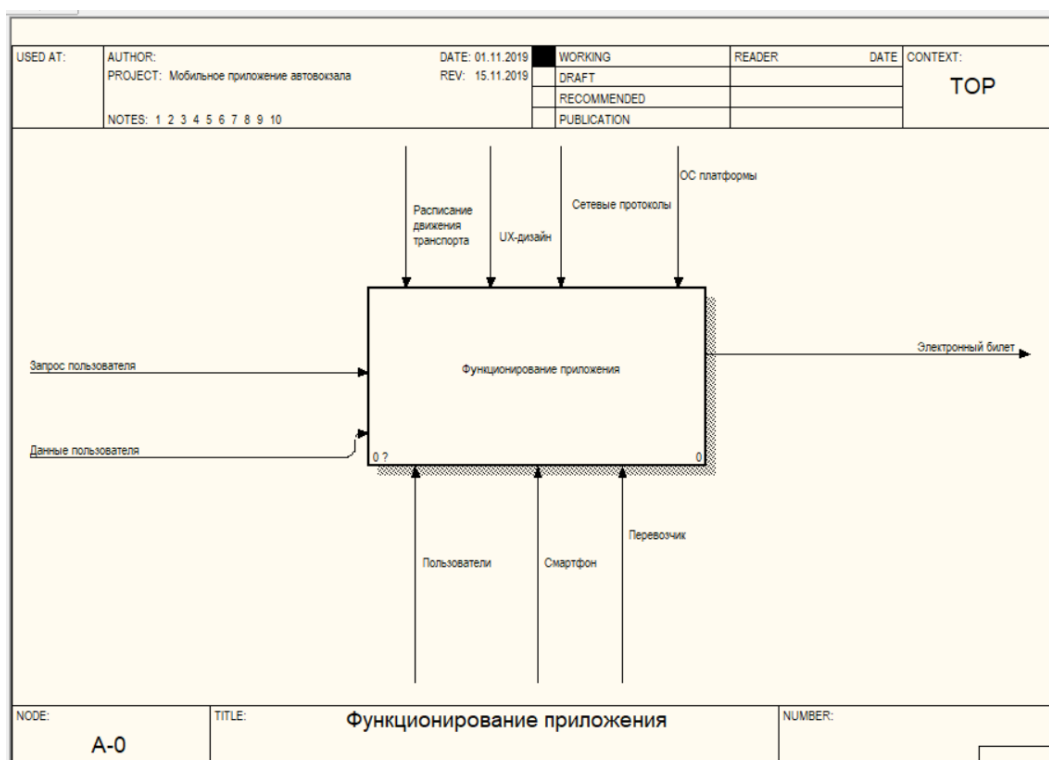


Рисунок 9 – Контекстная диаграмма в нотации IDEF0

Входами являются запрос пользователя, например, рейса между городами или просмотр расписания конкретной станции. Данными пользователя в данном случае являются ФИО пользователя, email и пароль от учетной записи. Управляющими воздействиями считаются расписание движение транспорта, сетевые протоколы передачи данных, ОС мобильного устройства, а также рекомендации UX-дизайна приложения. Механизмами являются пользователи, смартфон и перевозчик. Результатом функционирования мобильного приложения продажи транспортных услуг автовокзала является электронный билет.

Основная функция, выделенная на контекстной диаграмме, была разложена (декомпозирована) на составные части посредством создания дочерней диаграммы следующего, более низкого уровня. Диаграмма декомпозиции представлена на рисунке 10.

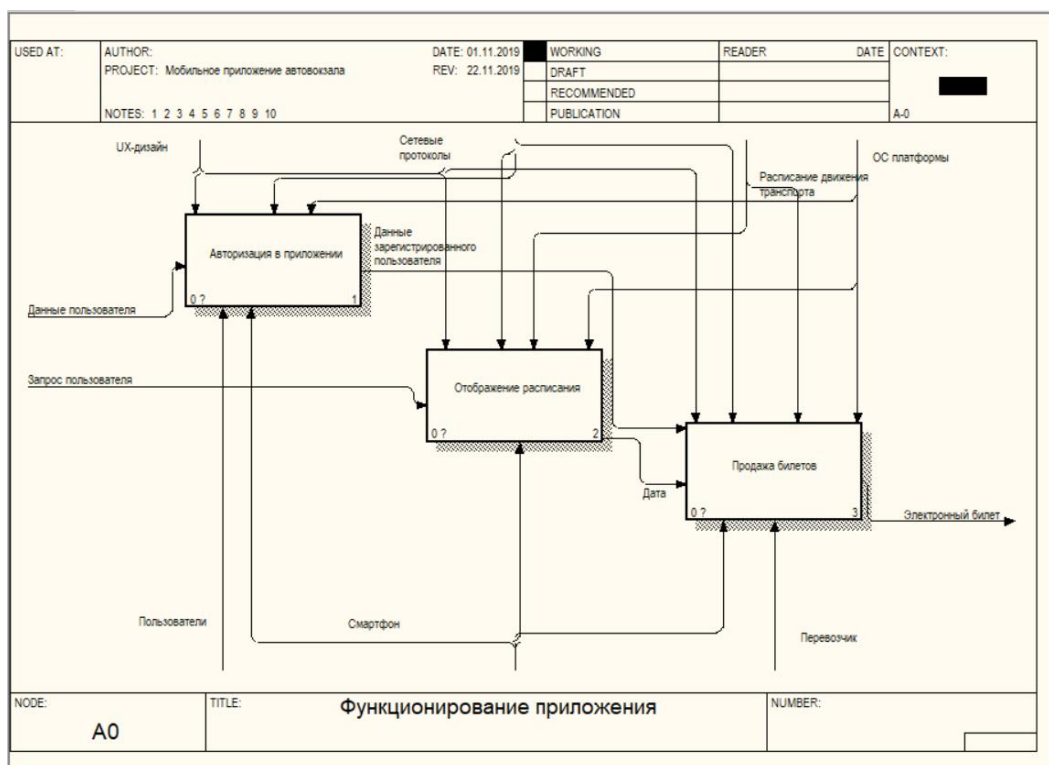


Рисунок 10 – Диаграмма декомпозиции функционирования приложения продажи транспортных услуг автовокзала

Пользователь авторизуется в приложении, после чего, он может оформить покупку билета. Приложение использует данные зарегистрированного пользователя для продажи билета, а также данные о рейсе, предоставляемые перевозчиком. Всё взаимодействие происходит посредством обмена данными с сервером через сеть интернет. Обработка данных происходит непосредственно на смартфоне, работающем на платформе Android. Окна авторизации и отображения расписания выполнены в стиле, рекомендованными стандартами UX-дизайна [4].

DFD – это методология графического структурного анализа, предназначенная для описания процессов, которые преобразуют данные, потоков данных, которые переносят данные, активных объектов, которые являются источниками и получателями данных, и хранилищ данных, которые пассивно хранят данные. Диаграмма потоков данных процесса оформления проживания клиента представлена на рисунке 11.

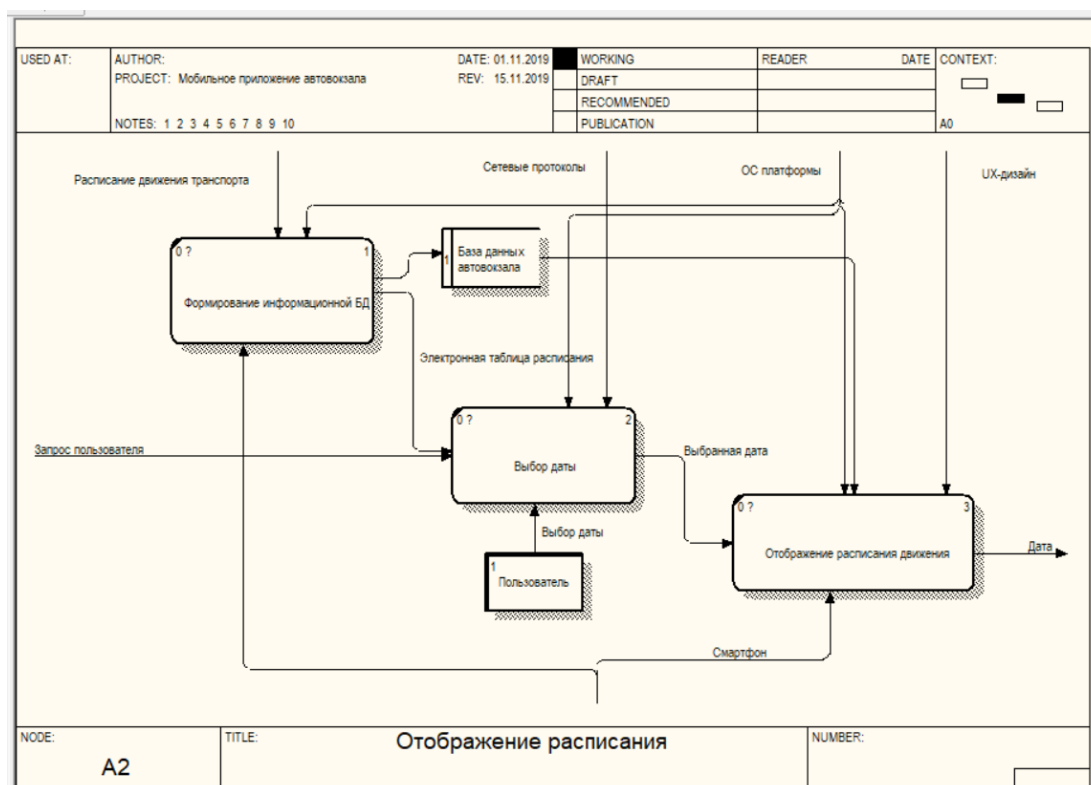


Рисунок 11 – Диаграмма декомпозиции «Отображение расписания» в нотации DFD

Отображение расписания происходит после получения соответствующего запроса пользователя приложению. Пользователь выбирает дату из представленного приложением таблицы, которая сформирована на основе данных из базы данных автовокзала. На основе расписания движения транспорта формируется информационная база данных, используемая затем в базе данных автовокзала. Расписание формируется самой организацией (автовокзалом) и в определенном заранее формате используется для формирования базы данных. Обмен данными с сервером происходит в сети интернет, их обработка происходит непосредственно на смартфоне, работающем на платформе Android.

С помощью диаграммы в нотации IDEF3 был декомпозирован процесс авторизации в приложении. Диаграмма представлена на рисунке 12.

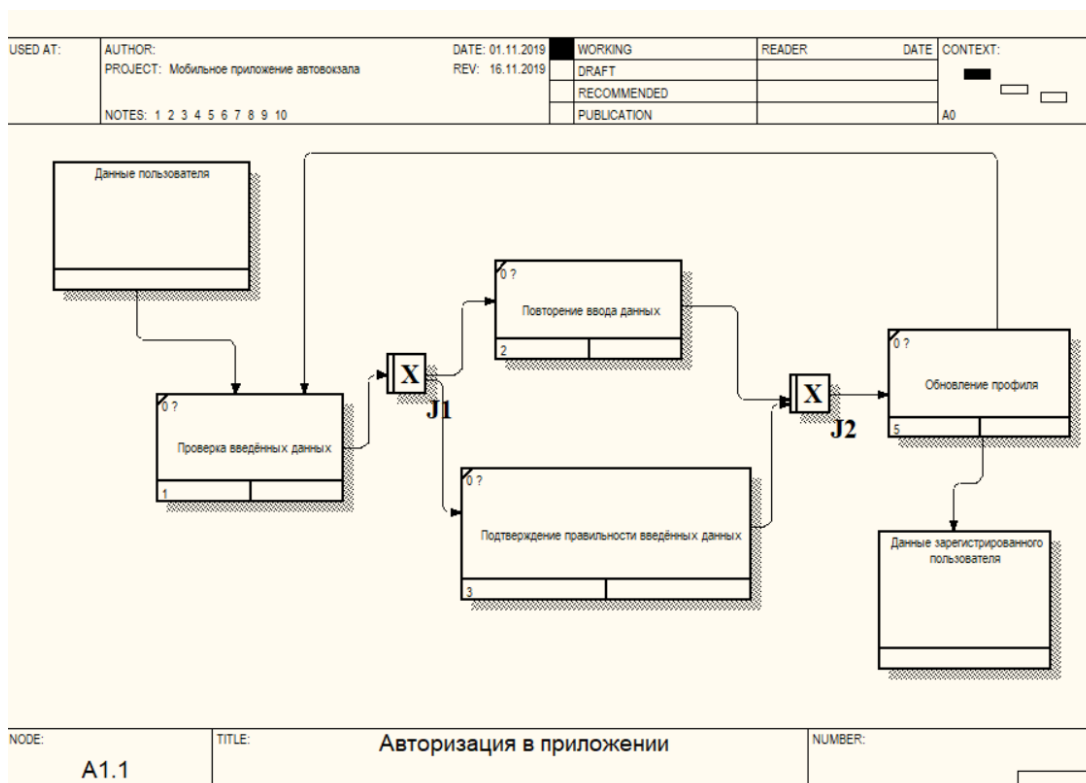


Рисунок 12 – Декомпозиция процесса авторизации в приложении в нотации IDEF3

Методология IDEF3 позволяет рассмотреть конкретный процесс с учетом последовательности выполняемых операций. Процесс авторизация в приложении начинается с проверки введенных данных и, если данные введены верно, происходит обновление профиля и пользователь может пользоваться основными функциями приложения, иначе программа запросит повторный ввод данных.

Также в нотацию IDEF0 был декомпозирован процесс продажи билетов. Процесс начинается с получением выбранной даты пользователем, затем пользователь выбирает посадочное место в транспорте. После успешного выбора места пользователь должен пройти процесс оплаты билета. В случае успешной операции оплаты программа получает подтверждение об этом и формируется электронный билет, который затем пользователь может использовать как проездной документ. В формировании билета участвуют данные пользователя, дата отправления, выбранное место. Обмен с сервером происходит через сеть интернет, обработка данных

происходит непосредственно на смартфоне, работающем на платформе Android.

Диаграмма декомпозиции представлена на рисунке 13.

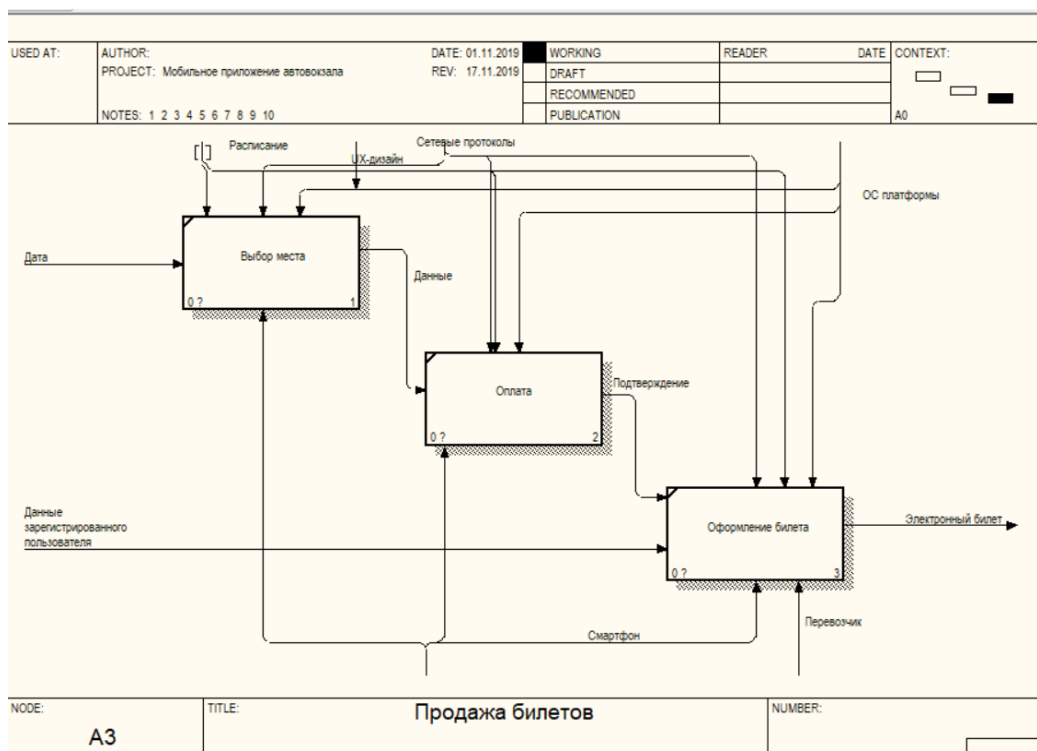


Рисунок 13 – Декомпозиция процесса продажи билета в нотации IDEF0

Таким образом, можно сказать, что использованное CASE-средство идеально подходит для анализа бизнес-процессов.

1.6 Средства разработки мобильного приложения

Для создания мобильного приложения продажи транспортных услуг автовокзала была выбрана платформа Android, в качестве языка программирования был выбран Java, а в качестве среды разработки клиентской части была выбрана Android Studio, серверная часть реализована на платформе Firebase от Google, а язык разметки XML был выбран для реализации графической части приложения.

Выбор объектно-ориентированного языка программирования (ООП) Java обусловлен тем, что данный язык имеет универсальные методы, типы данных и большую библиотеку всевозможных надежных решений,

обеспечивающими наивысшую безопасность и производительность разработки программ [5]. Программный интерфейс приложения (API) ОС Android очень похож на API языка Java, и Android поддерживает если не все доступные в SDK классы, то, по крайней мере, наиболее важные, следовательно, язык Java является наиболее правильным выбором для написания программ для ОС Android.

Android Studio обладает всеми необходимыми инструментами для разработки полноценных мобильных приложений [6]. Среди явных преимуществ данной среды разработки можно выделить поддержку языка Java, легко настраиваемый эмулятор мобильного устройства, графический редактор XML.

Серверная часть реализуется инструментами платформы Firebase. Эта компания предоставляет облачные услуги для разработчиков, обладая готовыми решениями в сфере аутентификации и работе с данными [7]. Преимущество Firebase заключается в том, что данная платформа является одним из сервисов компании Google, которая и занимается разработкой ОС Android. Поэтому подключение своего проекта к данной системе занимает минимальное количество времени, взамен разработчик получает готовые средства аналитики, механизм аутентификации и документно-ориентированную базу данных NoSQL.

Базой данных в мобильном приложении будет являться документно-ориентированная база NoSQL, СУБД которой предоставляется платформой Firebase. В основе документно-ориентированных СУБД лежат документные хранилища, имеющие структуру дерева [8]. Структура дерева начинается с корневого узла и может содержать несколько внутренних и листовых узлов. Главное преимущество данного вида базы данных является формат хранения данных идентичный JSON, широко используемый в веб разработке, что позволяет легко обрабатывать полученные данные и отправлять новые средствами языка Java.

Графическая часть мобильного приложения реализована на языке разметки XML. В Android Studio имеется возможность писать, как сам код на языке XML с поддержкой синтаксиса, так и воспользоваться графическим редактором для верстки визуальной части.

2 Проектирование мобильного приложения

2.1 Базовая архитектура мобильного приложения

Разрабатываемое мобильное приложение продажи транспортных услуг автовокзала имеет архитектуру клиент-сервер. Клиент-серверная архитектура приложения представлена на рисунке 14.

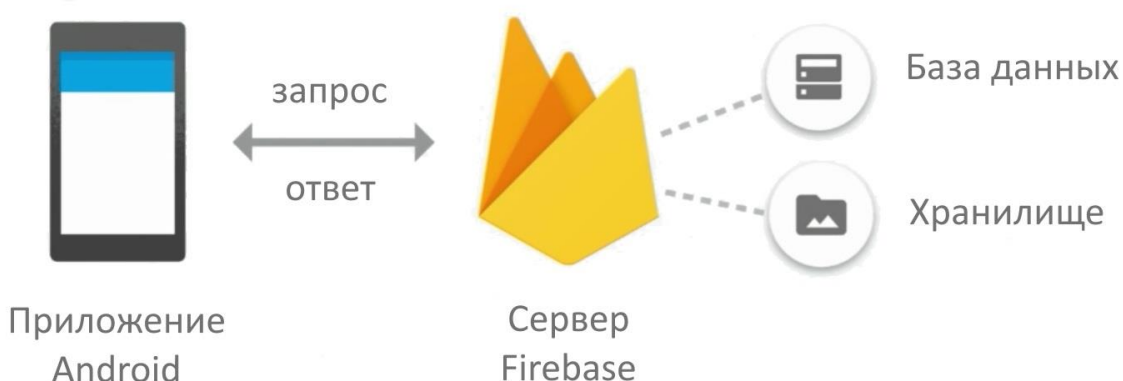


Рисунок 14 – Клиент-серверная архитектура мобильного приложения

Основным преимуществом организации мобильного приложения по архитектуре клиент-сервер является удачное сочетание централизованного хранения, обслуживания и распределенного доступа к информации.

В представленной на рисунке 14 архитектуре показаны главные компоненты системы: приложение Android, сервер Firebase, база данных и хранилище.

Клиентом в данной конфигурации является приложение Android. Пользователь использует свой смартфон для работы с мобильным приложением продажи транспортных услуг автовокзала. Технически это

выглядит как процесс обмена запросами и ответами между приложением на смартфоне и сервером Firebase. Стоит уточнить тот факт, что в данной конфигурации клиент является «толстым», т.е. берёт на себя основную вычислительную работу. Ответ от сервера приходит необработанным, с лишними данными, и уже непосредственно программными методами самого приложения происходит обработка полученного массива данных в соответствии с заданными параметрами.

Сервер Firebase обрабатывает поступающие запросы от пользователя, которыми могут являются запросы к получению, записи или изменению данных, а также процесс аутентификации. Фактически, сервер Firebase представляет собой облачную услугу компании Google, и разработчик использует готовое API для работы с базой данных, имея ограниченную возможность кастомизации сервера.

База данных реализована с помощью подхода NoSQL. Это относительно новая парадигма проектирования баз данных, направленная на решение проблем масштабируемости и доступности, за счет атомарности и согласованности данных. В базах Firebase данные хранятся в документно-ориентированном виде, схожим с JSON-форматом представления данных.

Хранилище в этой конфигурации сохраняет в себе медиа файлы, такие, например, как фотография профиля пользователя. Это сделано с целью облегчить фактический вес мобильного приложения на устройстве пользователя.

2.2 Технология разработки мобильного приложения

Процесс разработки мобильного приложения продажи транспортных услуг автовокзала будет основываться на методологии разработки RUP (Rational Unified Process). RUP это одна из самых известных и эффективных методологий управления процессом разработки программных продуктов, использующих итеративную модель разработки. Итерационная разработка программ в RUP подразумевает разбиение проекта на несколько

подпроектов, которые прорабатываются по порядку, и каждая итерация разработки четко определена набором задач, которые должны быть решены в конце итерации [9]. В результате RUP обеспечивает создание в заданные сроки и в рамках установленного бюджета качественного ПО, которое полностью отвечает поставленным требованиям заказчика. Методология RUP наибольшее внимание акцентирует начальным этапам разработки программы и, следовательно, направлена на снижение коммерческих и технических рисков, за счет того, что внести коррективы и обнаружить ошибки можно еще на ранних стадиях разработки программы.

Особенностью RUP является то, что в результате работы над проектом создаются и развиваются модели, позволяющие рассмотреть будущую систему, ее объекты и их взаимодействие еще до вложения значительных средств в разработку, увидеть ее глазами будущих пользователей снаружи и разработчиков изнутри еще до создания первой строки исходного кода [10]. Большинство моделей представляются наборами диаграмм UML.

При проектировании и реализации программных средств будут построены модель проектирования, модель реализации и модель развертывания.

Модель проектирования – это объектная модель, которая описывает физическую реализацию требований и служит для выработки структуры программы на основе этих требований. Создание данной модели включает построение диаграммы классов системы, диаграммы деятельности, диаграммы последовательности, диаграммы кооперации, а также диаграммы состояний для отдельных объектов.

Модель реализации описывает, как реализуются в виде компонентов элементы модели проектирования. Данная модель представляется диаграммой компонентов.

Модель развертывания служит для описания физического размещения компонентов распределенной системы по вычислительным узлам системы. Для отображения данной модели строится диаграмма развертывания.

2.3 Проектирование структуры данных мобильного приложения

Проектирование структуры данных осуществлялось с помощью Case-средства ERwin Data Modeler 7.3 и его методологии IDEF1X. Благодаря простоте интерфейса, а также поддержке большого количества платформ и специфических особенностей организации физической памяти различных серверных СУБД ERwin является наиболее известным и доступным средством моделирования данных [11].

Для создания структуры данных в стандарте IDEF1X были рассмотрены два уровня представления модели данных – логический и физический.

2.3.1 Логическая модель базы данных мобильного приложения

На основе проведенного анализа предметной области и функциональных требований к системе была построена модель «сущность-связь» для БД на логическом уровне. Логический уровень – это абстрактный взгляд на данные, в котором данные представлены также, как и реальном мире. Логическая модель включает сущности, атрибуты и связи, а также полностью отражает суть выбранной предметной области.

Проектирование логической модели БД было произведено с помощью CASE-средства моделирования данных CA ERwin Data Modeler в нотации IDEF1X. Методология IDEF1X разработана специально для проектирования реляционных БД. Несмотря на то, что используемая нами база данных является документно-ориентированной, построение модели данных в IDEF1X является допустимым, при непосредственной реализации будет произведена корректировка модели в соответствии со спецификой документно-ориентированного формата.

При построении логической модели были выделены следующие сущности: пользователь, билет, маршрут, отправления, назначения, транспортное средство, тип транспортного средства, а также остановочные пункты.

Построенная логическая модель в нотации IDEF1X представлена на рисунке 15.



Рисунок 15 – Логическая модель базы данных

В дальнейшем будет произведен переход от логического уровня модели БД к физическому уровню.

2.3.2 Физическая модель базы данных мобильного приложения

На основе логической модели данных была построена физическая модель, зависящая от реализации конкретной СУБД.

Полная атрибутивная физическая модель базы данных для мобильного приложения продажи транспортных услуг автовокзала в нотации IDEF1X представлена на рисунке 16.

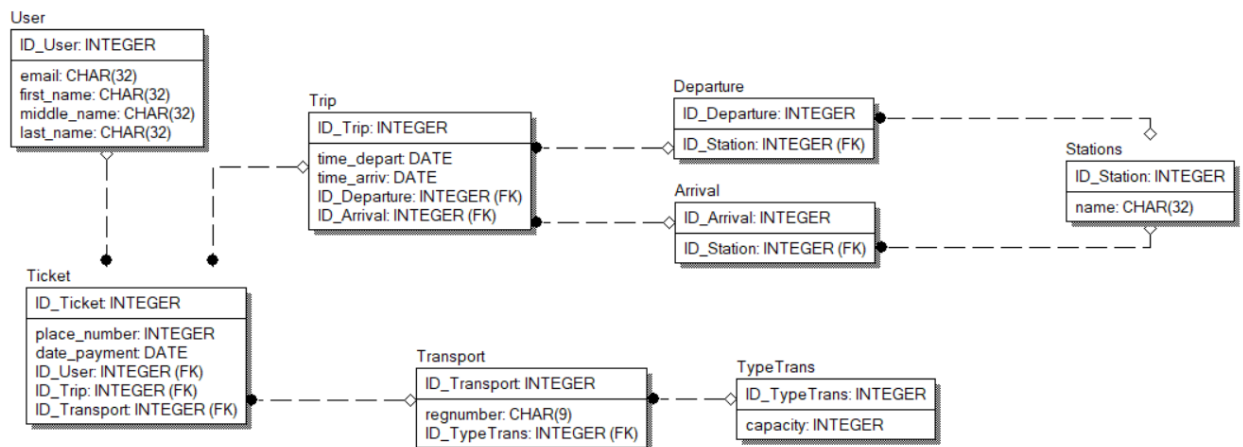


Рисунок 16 – Физическая модель базы данных

В базе данных Firebase предусмотрены целые и символьные типы данных, тип данных «date» целесообразно представлять в виде символьного массива определенного формата для успешной программной работы с ним.

2.4 Проектирование программных средств

2.4.1 Проектирование мобильного приложения

Согласно модели предметной области, была спроектирована диаграмма программных классов, представленная на рисунке 17.

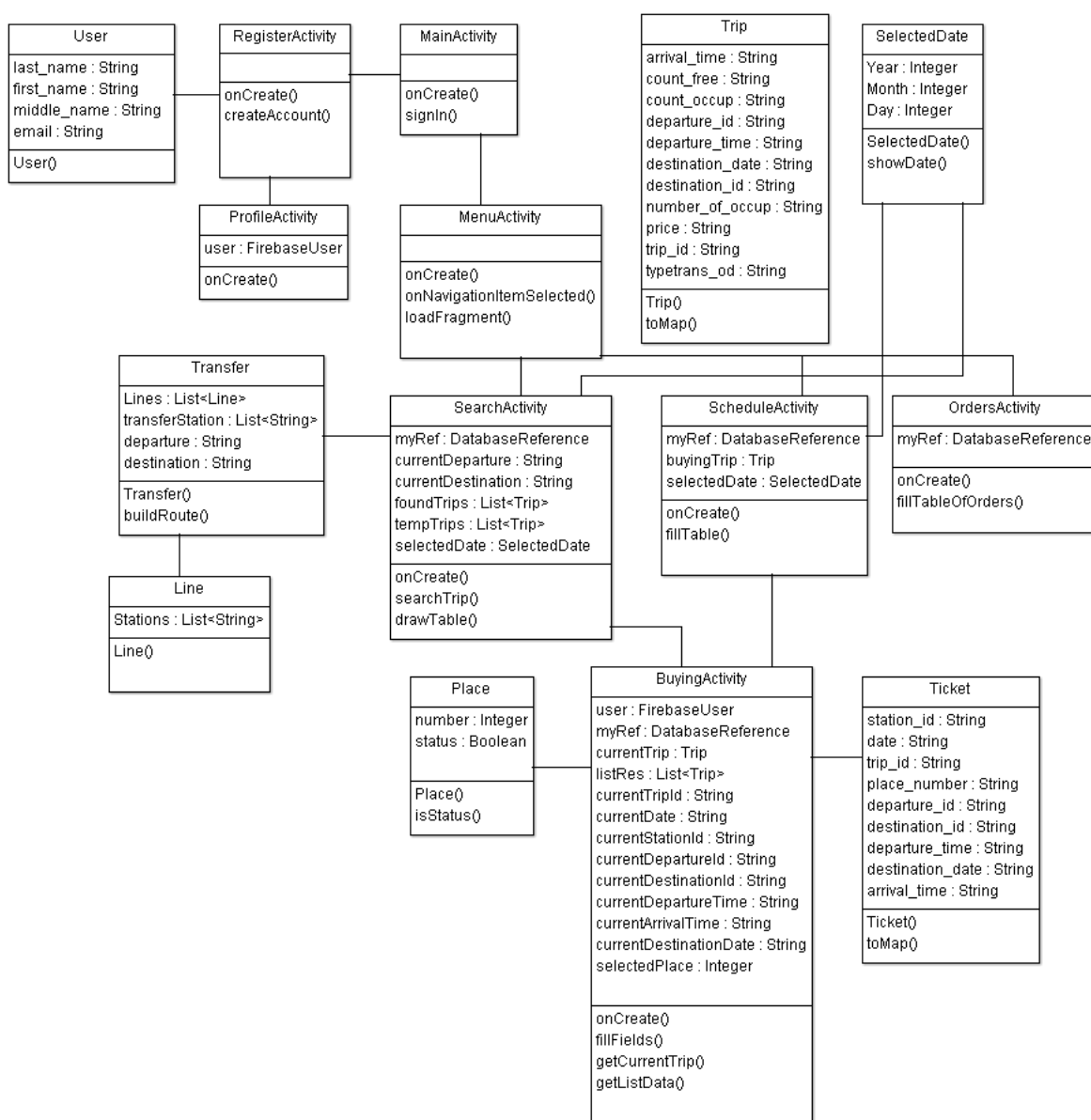


Рисунок 17 – Диаграмма программных классов

Условно классы в данном мобильном приложении можно разделить на две группы: классы, наследуемые от класса AppCompatActivity (т.н. активности) и все остальные. Краткое описание представленных на диаграмме классов приведено в таблице 3.

Таблица 3 – Описание классов

№	Класс	Описание
1	2	3
1.	MainActivity	Точка входа в программу. Класс окна авторизации. На окне находятся два поля для ввода логина и пароля и две кнопки: «регистрация» и «войти».
2.	RegisterActivity	Класс окна регистрации пользователя. Окно открывается при нажатии соответствующей кнопки в окне авторизации.
3.	MenuActivity	Класс окна меню. Состоит из двух элементов: фрагмента и нижнего меню. Содержимое фрагмента меняется в зависимости от выбранного пункта меню.
4.	ProfileActivity	Класс окна страницы профиля пользователя. Содержит основные данные пользователя и фото профиля, а также кнопку «выйти из профиля».
5.	SearchActivity	Класс окна поиска рейсов. Содержит в себе механизм создания запроса к базе данных, а также механизм обработки ответа от сервера. Состоит из полей ввода, календаря и таблицы, в которую будет выведен результирующий набор данных.
6.	ScheduleActivity	Класс окна расписания. Выводит информацию на экран о расписании выбранного города на выбранную дату.
7.	OrdersActivity	Класс окна списка купленных билетов пользователя. Выводит список купленных билетов пользователя, а также механизм отмены покупки.
8.	BuyingActivity	Класс окна оформления покупки билета. Окно покупки вызывается после нажатия соответствующей кнопки в окне поиска рейсов или расписания.
9.	User	Класс User содержит данные о пользователе. Класс используется при авторизации и регистрации.
10.	Transfer	Класс Transfer реализует механизм поиска маршрута с пересадками.
11.	Line	Класс Line это программное представление такого понятия, как шоссе или трассе, объекты этого класса используются в классе Transfer.
12.	Trip	Класс Trip это программное представление понятия «рейс». Используется при поиске рейсов, отображении расписания и покупке билета.
13.	Ticket	Класс Ticket содержит информацию о билете,

Продолжение таблицы 3

1	2	3
		используется для покупки билета и отображения списка купленных билетов.
14.	Place	Класс Place реализует механизм и графическое представление схематичного расположения мест в транспорте.
15.	SelectedDate	Класс SelectedDate необходим для обработки выбранной в календаре даты.

В ходе разработки мобильного приложения будет реализовано 15 классов, часть из которых отвечает за визуальную функциональность приложения, другая часть за работу с данными предметной области.

2.4.2 Диаграмма последовательности мобильного приложения

Диаграмма последовательности описывает сценарии системы. В данном виде диаграмм демонстрируются объекты и их коммуникационный аспект. Другими словами, отражаются взаимодействующие объекты, которые обмениваются между собой некоторой информацией [2].

Каждый объект имеет собственную линию жизни, означающую период времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях [2]. Также каждый объект может передавать другому объекту фокус управления, что означает активацию деятельности этого объекта.

Диаграмма описывает процесс покупки билета. Она включает в себя три объекта: пассажир (пользователь), клиентское приложение и сервер БД, которые взаимодействуют в рамках сценария и обмениваются сообщениями. Сервером БД в данном случае выступает платформа Firebase в целом. Для отображения процесса «Покупка билета» была построена диаграмма последовательности, представленная на рисунке 18.

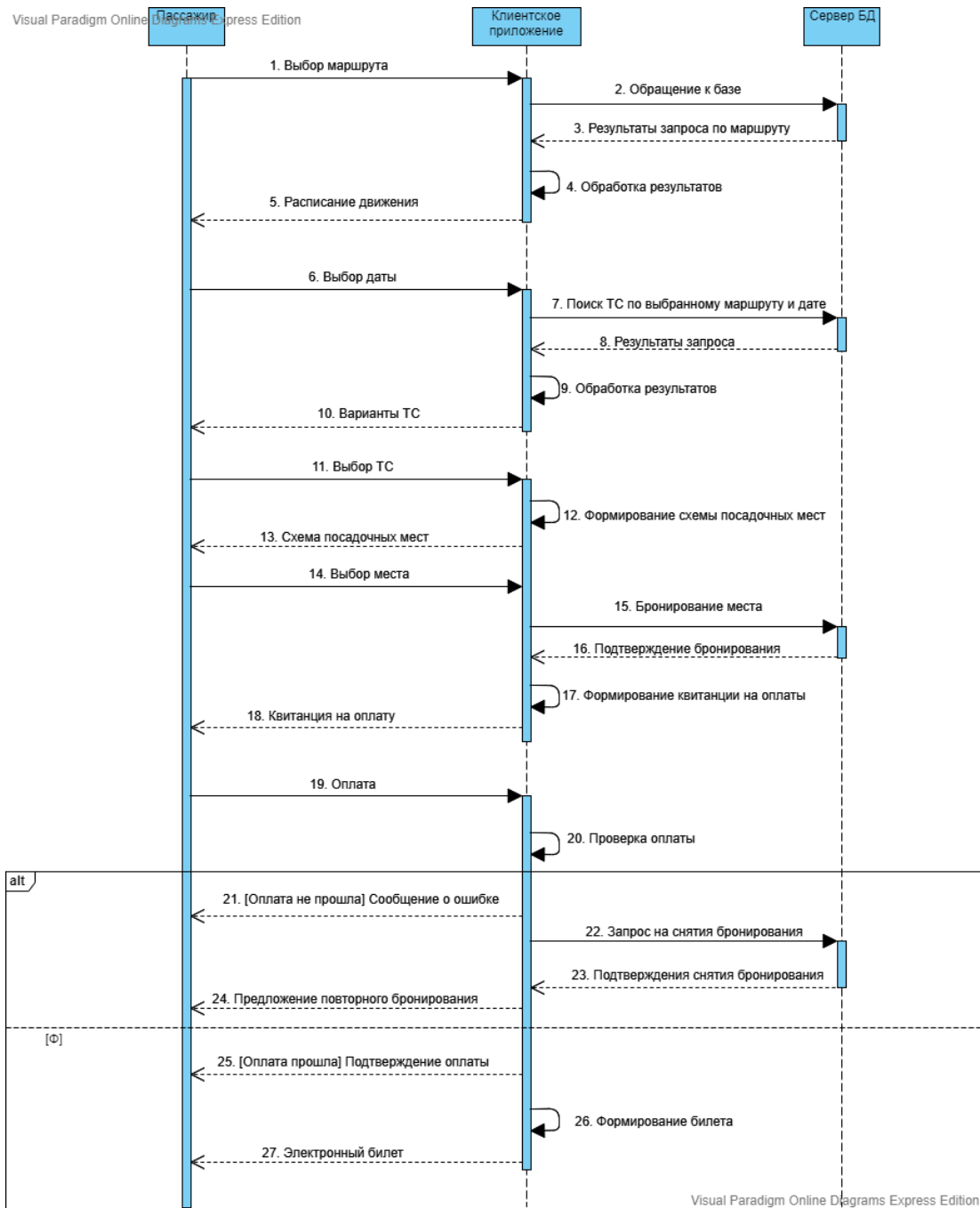


Рисунок 18 – Диаграмма последовательности процесса «Покупка билета»

Пассажир выбирает маршрут, дату, ТС и место в нем. После успешной обработки данных пассажир оплачивает билет. В случае успешной оплаты пассажир получает билет.

Клиентское приложение обрабатывает действия пассажира. После выбора пассажиром маршрута клиентское приложения обращается к серверу за получением соответствующему запросу результирующему набору данных, затем обрабатывает его и возвращает пользователю расписание движения

транспорта, где пассажир может выбрать конкретную дату. После выбора пассажиром даты клиентское приложение посылает пассажиру список ТС, которые выбирает пользователь. После выбора пассажиром ТС клиентское приложение снова обращается к серверу для получения данных. Получив их, оно формирует схему посадочных мест в ТС и отправляет его пассажиру. После выбора пассажиром места в ТС клиентское приложение отправляет запрос на бронирование соответствующего места на сервер, затем отправляет квитанцию (или форму заполнения платежных данных) на оплату билета пассажиру. В случае успешной оплаты клиентское приложение формирует и возвращает пассажиру билет, в случае неуспешной – предложение повторной оплаты.

2.4.3 Диаграмма деятельности мобильного приложения

Диаграмма деятельности показывает разбиение некоторой деятельности на ее составные части. Также на данной диаграмме отображаются логика и последовательность действий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой простой операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии.

Последовательность действий при покупке билета начинается с авторизацией пользователем в приложении, затем он выбирает пункт отправления и пункт прибытия, дату и рейс. После выбора рейса, приложении формирует схему посадочных мест в транспортном средстве, которое работает на этом рейсе, чтобы пользователь мог выбрать свободное. При выполнении перечисленных действий, пользователь оплачивает билет и получает его на электронной почте. Сценария работы пользователя с программой была реализована диаграмма деятельности, представленная на рисунке 19.

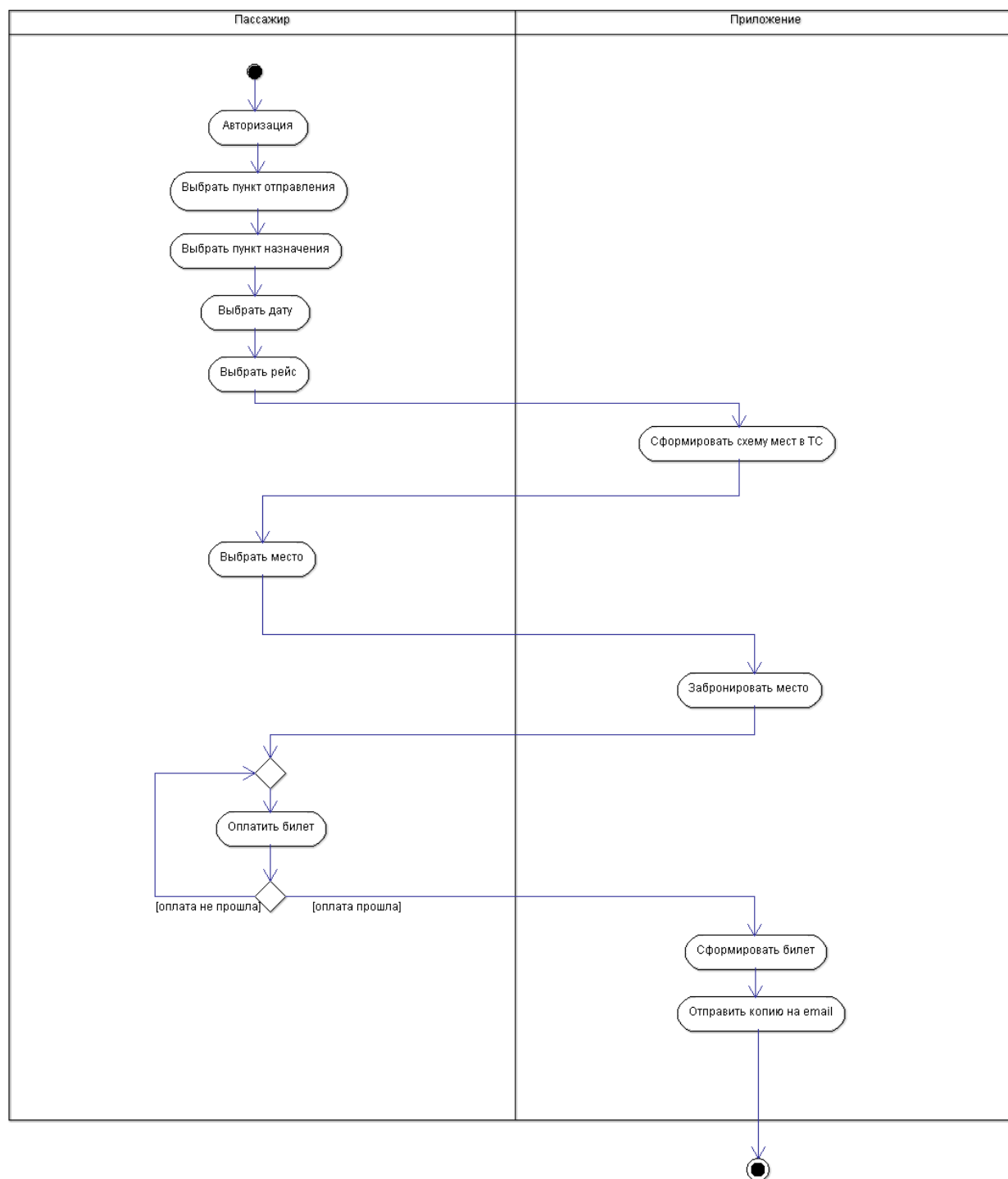


Рисунок 19 – Диаграмма деятельности «Покупка билета»

Диаграмма описывает процесс покупки билета в мобильном приложении продажи транспортных услуг автовокзала. Она состоит из двух дорожек, которые представляют собой: пассажира (пользователя) и приложение.

2.4.4 Диаграмма кооперации мобильного приложения

Диаграмма кооперации описывает последовательность взаимодействия объектов и их структурные аспекты отношений. Объекты представлены в виде прямоугольников. Динамические связи изображаются в виде линий, где стрелками указывается направление потоков сообщений между объектами, а порядковый номер устанавливает очередность взаимодействия.

Для отображения последовательности обмена сообщениями в процессе покупки билета была построена диаграмма кооперации, представленная на рисунке 20.

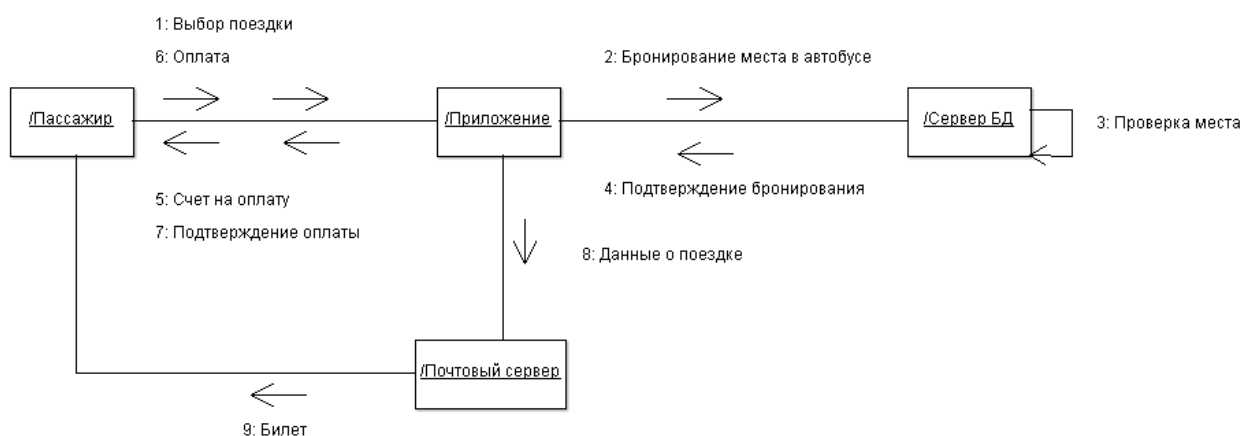


Рисунок 20 – Диаграмма кооперации процесса покупки билета

Диаграмма описывает процесс покупки билета в приложении. Она включает в себя четыре объекта: пассажир, приложение, сервер БД и почтовый сервер, которые обмениваются сообщениями.

Объект «Пассажир» выбирает поездку, т.е. дату отправления, место отправления и назначения, а также посадочное место. Производит оплату поездки, после чего ему на электронную почту приходит электронный билет.

Объект «Приложение» получает от пользователя данные о поездке, бронирует место в автобусе (чтобы во время процедуры оплаты данное место не купил кто-то другой). После подтверждения бронирования от Сервера БД, отправляет пассажиру форму для оплаты поездки. После успешной оплаты пассажиром поездки, отправляет данные о поездке на почтовый сервер.

Объект «Сервер БД» осуществляет проверку места в автобусе (свободно/занято/забронировано).

Объект «Почтовый сервер» отправляет пассажиру электронный билет на электронную почту. Фактически представляет собой программный механизм.

2.4.5 Диаграмма состояний мобильного приложения

Диаграмма состояний демонстрирует, как объект переходит из одного состояния в другое. В данной диаграмме отражается процесс смены состояний объекта при наступлении некоторого события и выполнении определенных условий.

Диаграмма состояний класса «Билет» представлена на рисунке 21.

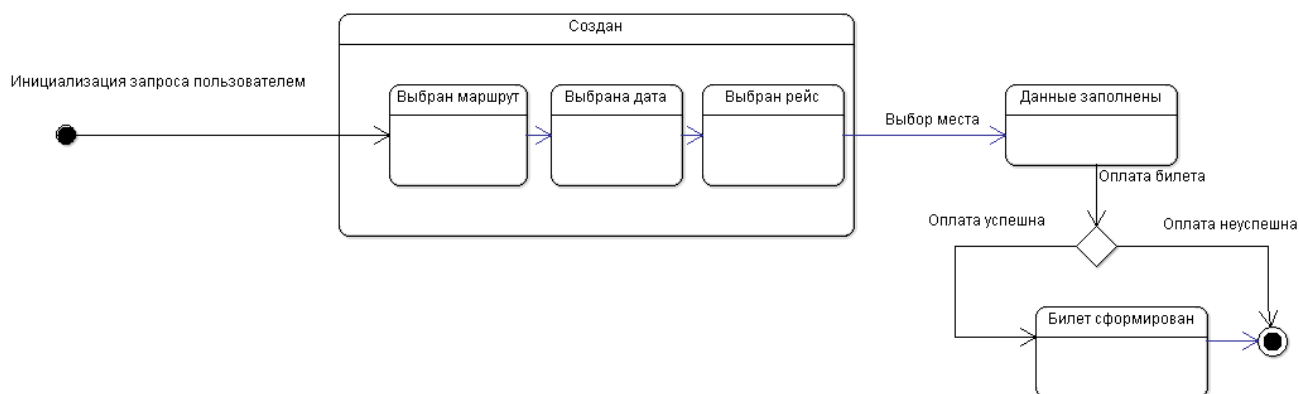


Рисунок 21 – Диаграмма состояний класса «Билет»

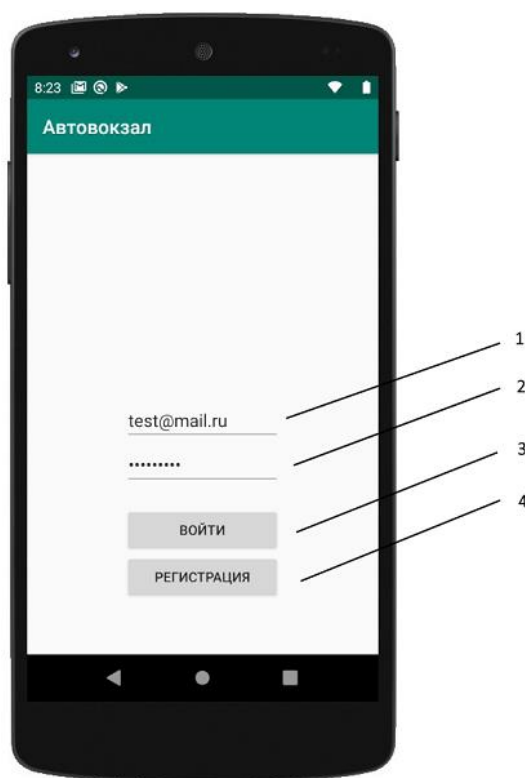
Во время пребывания в состоянии «Выбран маршрут» производится выбор пассажиром точки отправления и точки назначения. «Выбрано время» означает, что пассажир выбрал желаемые дату и время, «Выбрано ТС» означает, что пассажир выбрал ТС. Состояние «Создан» означает, что основные значения билета выбраны. Переход в состояние «Данные заполнены» производится после выбора посадочного места пассажиром. «Билет сформирован» достигается после успешной оплаты.

2.5 Проектирование пользовательского интерфейса мобильного приложения

Опираясь на диаграмму вариантов использования, был спроектирован графический интерфейс мобильного приложения продажи транспортных услуг автовокзала.

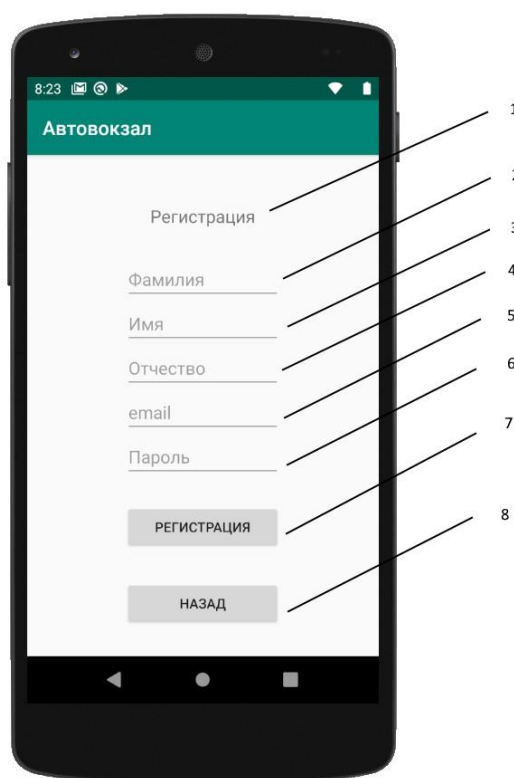
Специфика разработки графического интерфейса для мобильных приложений заключается в ограниченности пространства для размещения визуальных компонентов, это следует учитывать при проектировании страниц [4].

После запуска мобильного приложения пользователя встречает страницы авторизации. Интерфейс страницы авторизации представлен на рисунке 22.



- 1 – поле ввода электронный почты, компонент EditText;
 - 2 – поле ввода пароля, компонент EditText;
 - 3 – кнопка входа, компонент Button;
 - 4 – кнопка открытия страницы регистрации, компонент Button.
- Рисунок 22 – Графический интерфейс страницы авторизации

Графический интерфейс окна регистрации, представлен на рисунке 23.



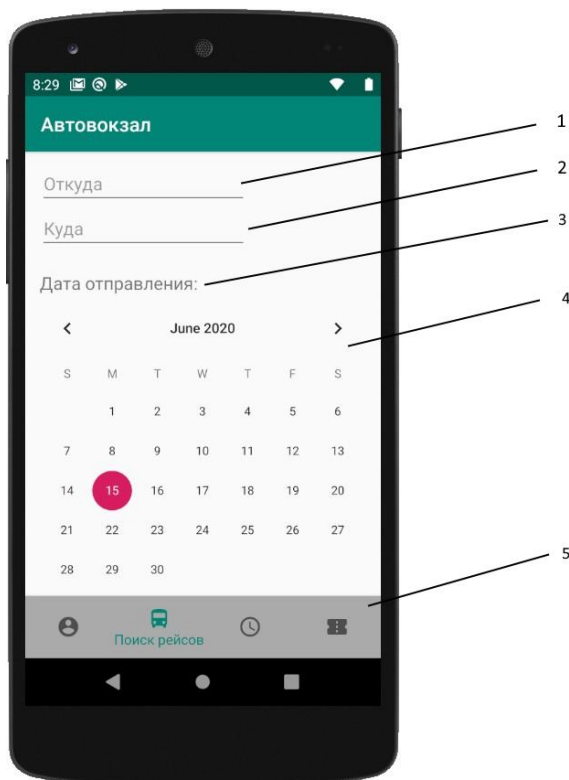
- 1 – надпись, компонент TextView;
- 2 – поле ввода фамилии пользователя, компонент EditText;
- 3 – поле ввода имени пользователя, компонент EditText;
- 4 – поле ввода отчества пользователя, компонент EditText;
- 5 – поле ввода email пользователя, компонент EditText;
- 6 – поле ввода пароля пользователя, компонент EditText;
- 7 – кнопка подтверждения регистрации, компонент Button;
- 8 – кнопка возврата к странице входа, компонент Button.

Рисунок 23 – Графический интерфейс страницы регистрации

После осуществления входа в приложение, пользователю открывается основная рабочая область программы, состоящая из нижнего меню и центральной области, визуального компонента Fragment, содержимое которого зависит от выбранного пункта меню.

Меню состоит из 4 элементов: страница профиля, страница поиска рейсов, страница расписания и страница списка заказов пользователя. После запуска приложения компонент Fragment по умолчанию содержит в себе страницу поиска рейсов. При переключении между этими четырьмя страницами меню всегда остается внизу текущей страницы.

Графический интерфейс меню и страницы поиска рейсов представлен на рисунке 24.



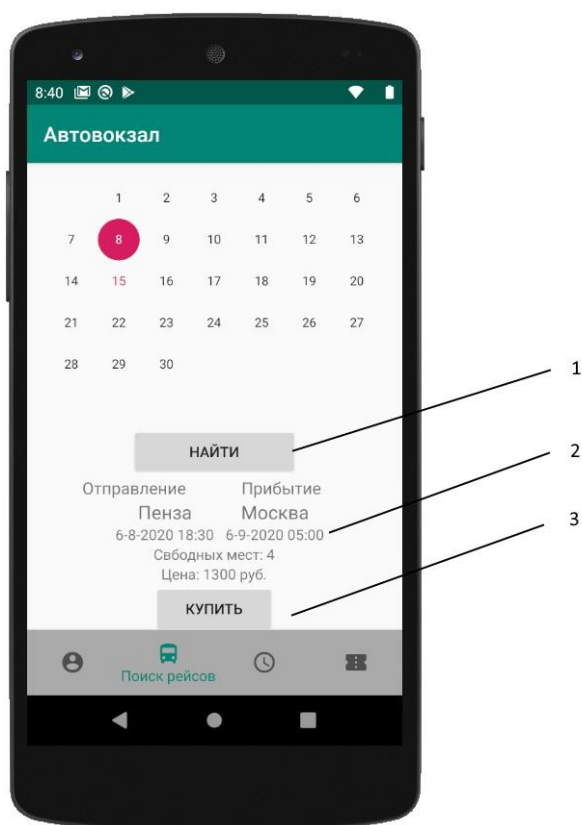
- 1 – поле ввода пункта отправления, компонент EditText;
- 2 – поле ввода пункта назначения, компонент EditText;
- 3 – надпись, компонент TextView;
- 4 – календарь, компонент CalendarView;
- 5 – меню, компонент BottomNavigationView.

Рисунок 24 – Графический интерфейс страницы поиска рейсов

Всё перечисленное содержимое находится внутри контейнера ScrollView. Если не все компоненты умецаются на текущем экране, то пользователь может «пролистать» вниз текущую страницу, где обнаружит не уместившиеся компоненты, в данном случае кнопку «Найти».

Если пользователь ввёл названия пункта отправления и пункта прибытия, а также выбрал интересующую его дату отправления, то при нажатии на кнопку «Найти» будет отображена таблица с рейсами, удовлетворяющим запросу пользователя.

Продолжение страницы поиска рейсов представлено на рисунке 25.



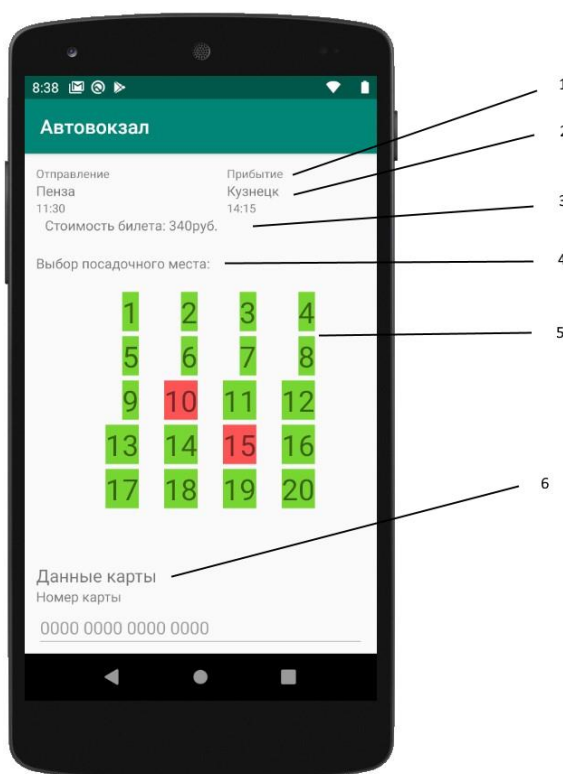
- 1 – кнопка инициализации поиска рейсов, компонент Button;
- 2 – информация о рейсе, компоненты TextView;
- 3 – кнопка покупки билета, компонент Button.

Рисунок 25 – Графический интерфейс страницы поиска рейсов

Результирующие наборы, т.е. информация по каждому билету представлена в компоненте TextView. Для организации визуально удобного, вертикально и горизонтально выравненного отображения используется компонент контейнер TableLayout, вертикально заполненный компонентами TableRow, которые в свою очередь содержат в себе компоненты TextView. Заполнение TableLayout происходит программно.

При нажатии на кнопку «Купить» пользователю откроется страница оформления покупки билета. На этой странице дублируется информация по выбранному рейсу, отображается схема посадки в автобусе, отображая свободные и купленные места.

Графический интерфейс страницы покупки представлен на рисунке 26.



- 1 – надпись заголовка таблицы, компонент TextView;
- 2 – строки пункт отправления и прибытия, компонент TextView;
- 3 – строка стоимости билета, компонент TextView;
- 4 – надпись, компонент TextView;
- 5 – схема мест в автобусе, компонент GridView;
- 6 – надпись, компонент TextView.

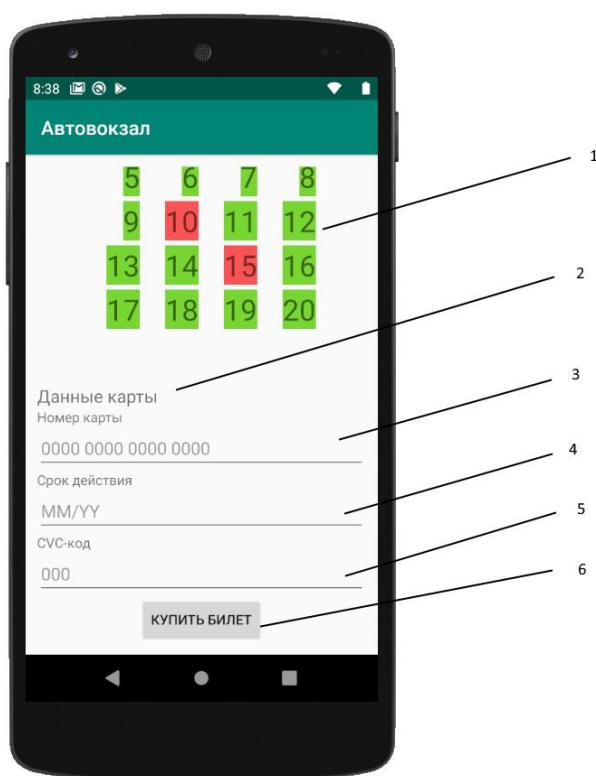
Рисунок 26 – Графический интерфейс страницы покупки билета

Данные о рейсе передаются на страницу покупки с помощью объекта класса Intent, что исключает несогласование данных на странице поиска и покупки.

Схема расположения мест в автобусе реализована с помощью компонента GridView, где ячейки играют роль посадочных мест. Зеленый места означают, что они свободны, красные – заняты.

После выбора места, пользователь может приступить к процедуре оплаты билета. Содержимое всей страницы находится в компоненте ScrollView, поэтому пользователю необходимо пролистать страницу вниз, для того, чтобы увидеть поля заполнения платежных данных.

Продолжение страницы покупки представлено на рисунке 27.



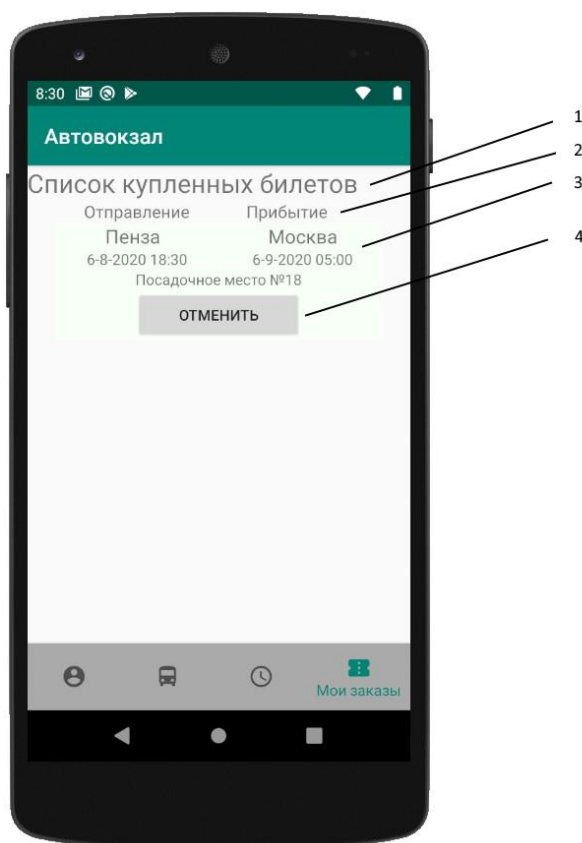
- 1 – схема мест в автобусе, компонент GridView;
- 2 – надпись, компонент TextView;
- 3 – поле ввода номера карты, компонент EditText;
- 4 – поле ввода срока действия карты, компонент EditText;
- 5 – поле ввода срока CVC-кода карты, компонент EditText;
- 6 – кнопка покупки, компонент Button.

Рисунок 27 – Графический интерфейс страницы покупки билета

В разделе «Данные карты», пользователю необходимо заполнить поля: номер карты, срок действия и CVC-код. После корректного ввода перечисленных данных и нажатия на кнопку «Купить» будет произведена покупка билета.

В данном окне руководителю выводится информация по проживанию клиентов в вариантах размещения по заявкам и без, даются возможности настраивать вывод списка записей о проживании, осуществлять поиск нужной записи, оформлять заселение клиента без предварительного бронирования, а также оформлять пользование услугами в период проживания и контролировать оплату проживания.

Посмотреть список купленных билетов можно на странице «Мои заказы», интерфейс которой представлен на рисунке 28.



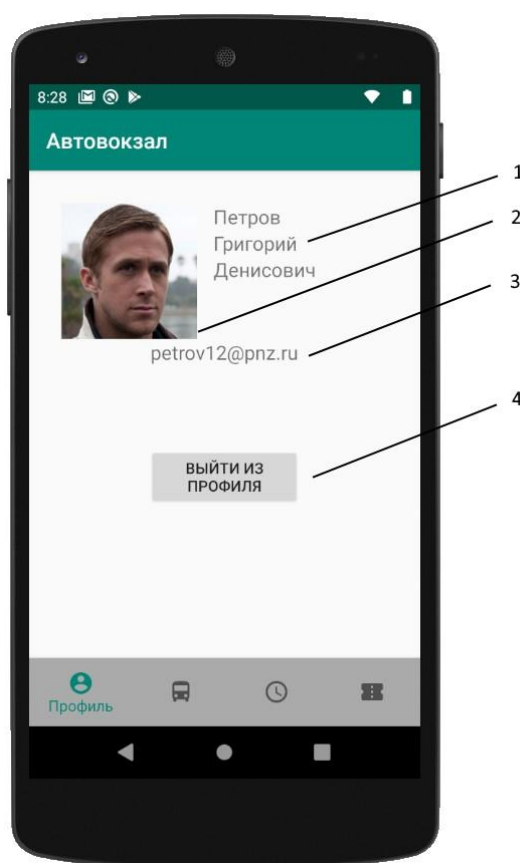
- 1 – надпись, компонент TextView;
- 2 – надпись, компонент TextView;
- 3 – информация о билете, компонент TextView;
- 4 – кнопка отмены покупки билета, компонент Button.

Рисунок 28 – Графический интерфейс страницы с заказами

Открытие данной страницы осуществляется через нижнее меню. Страница состоит из статических надписей, а также компонента-контейнера `TableLayout`, содержащий в себе `TableRow`. Заполнение данной таблицы происходит динамически, данные, отображаемые в таблице, получаются из базы данных `Firestore`.

При нажатии на кнопку «Отменить» происходит отмена соответствующей поездки.

Для открытия страницы профиля пользователя необходимо нажать на соответствующий элемент в нижнем меню. Графический интерфейс данной страницы представлен на рисунке 29.



- 1 – ФИО пользователя, компонент TextView;
- 2 – изображение профиля пользователя, компонент ImageView;
- 3 – email пользователя, компонент TextView;
- 4 – кнопка выхода из профиля, компонент Button.

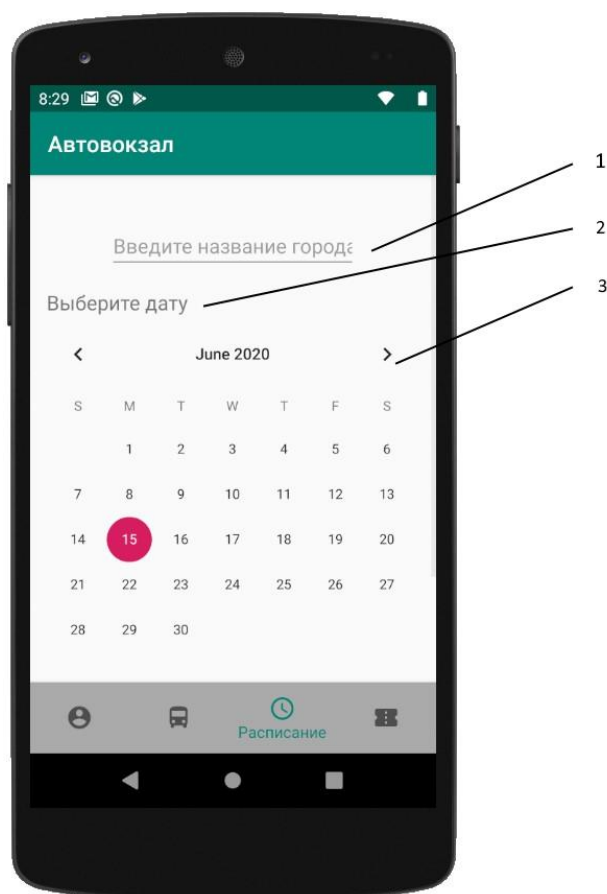
Рисунок 29 – Графический интерфейс страницы профиля

Страница профиля представляет собой содержание основной информации о пользователе. Данные, указанные в профиле, составляют персональные данные пассажира в билете.

Фото профиля является необязательным, но рекомендуемым для установки параметром.

При нажатии на кнопку «Выйти из профиля» происходит разлогирование пользователя из приложения и его возвращает на страницу авторизации.

Для просмотра всех рейсов конкретной станции в конкретную дату существует страница «Расписание». Перейти на неё можно с помощью соответствующего элемента в нижнем меню приложения. Графический интерфейс страницы расписания представлен на рисунке 30.



- 1 – поле ввода названия города, компонент EditText;
- 2 – надпись, компонент TextView;
- 3 – календарь, компонент CalendarView.

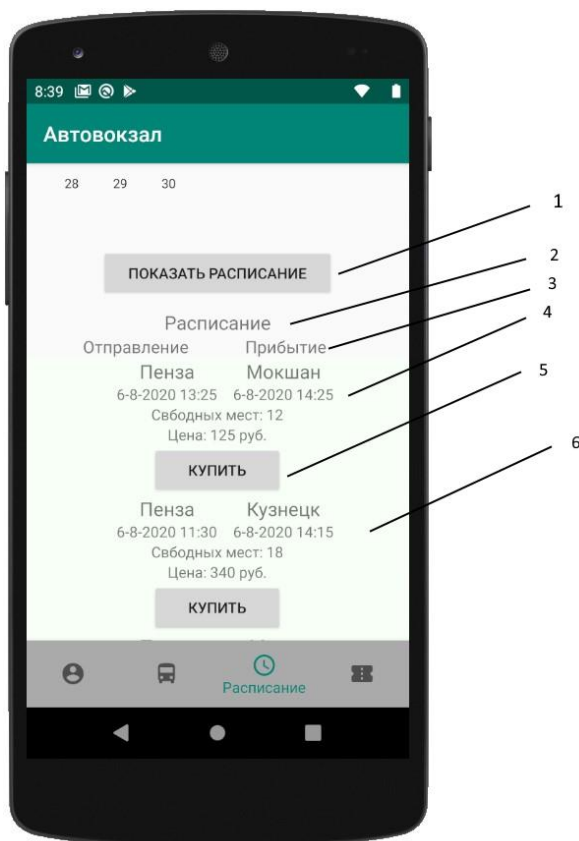
Рисунок 30 – Графический интерфейс страницы расписания

Страница с расписанием содержит поле ввода названия города или станции, календарь, а также кнопку «Показать расписание». Всё содержимое находится внутри компонента ScrollView, для случая, если экрана смартфона не хватит для отображения всех компонентов страницы.

При вводе названия станции, выборе необходимой даты в календаре и нажатия кнопки «Показать расписание», внизу страницы появляется результирующая таблица расписания рейсов соответствующего города. Технически это реализовано с помощью контейнера TableLayout, который динамически заполняется данными из базы Firebase. Элементами контейнера являются компоненты TableRow, включающая в себе основную информацию о рейсе и кнопку «Купить», при нажатии на которую открывается страница

покупки билета, идентичная той, что открывается со страницы поиска рейсов.

Продолжение графического интерфейса страницы расписания представлено на рисунке 31.



- 1 – кнопка показа расписания, компонент Button;
- 2 – надпись, компонент TextView;
- 3 – надпись, компонент TextView;
- 4 – информация о рейсе, компонент TextView;
- 5 – кнопка покупки билета, компонент Button.

Рисунок 31 – Графический интерфейс страницы расписания

Таким образом, графический интерфейс всего приложения состоит из страниц регистрации и авторизации, основных страниц меню (профиль пользователя, поиск рейсов, расписание и заказы пользователя), а также страницы оформления покупки билета. Проектирование интерфейса было выполнено в соответствии с базовыми рекомендациями UX дизайна.

3 Реализация программного обеспечения мобильного приложения

3.1 Диаграмма компонентов мобильного приложения

Программные средства разрабатываемого мобильного приложения продажи транспортных услуг автовокзала состоят из компонентов, которые представлены на рисунке 32.

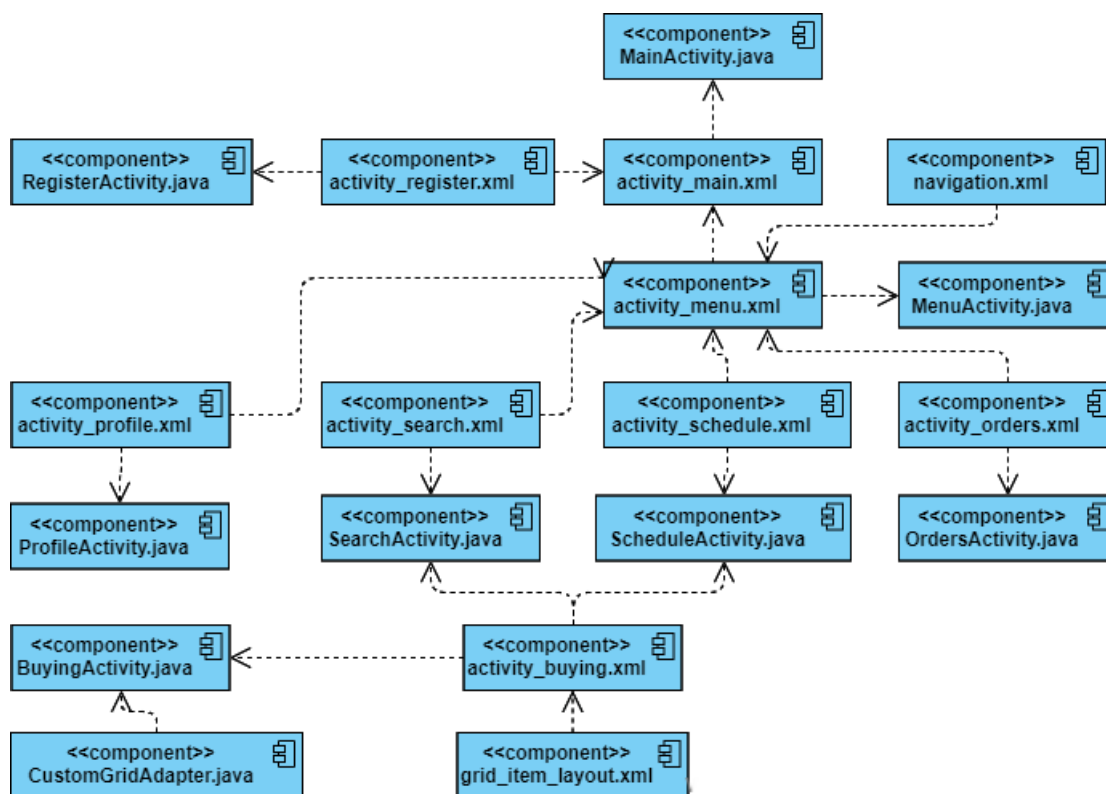


Рисунок 32 – Диаграмма компонентов

Файлы с расширением .xml являются графическими представлениями приложения, файлы с расширением .java являются файлами классов, написанных на языке Java.

Описание компонентов представлено ниже:

- MainActivity.java – основной файл, точка входа приложения, страница авторизации;
- activity_main.xml – страница авторизации;
- activity_register.xml – страница регистрации;
- RegisterActivity.java – класс для активности activity_register.xml;

- activity_menu.xml – главная страница приложения;
- MenuActivity.java – класс для активности activity_menu.xml;
- navigation.xml – файл верстки нижней панели меню, находящееся в activity_menu.xml;
- activity_profile.xml – страница профиля;
- ProfileActivity.java – класс для активности activity_profile.xml;
- activity_search.xml – страница поиска рейсов;
- SearchActivity.java – класс для активности activity_search.xml;
- activity_schedule.xml – страница расписания рейсов;
- ScheduleActivity.java – класс для активности activity_schedule.xml;
- activity_orders.xml – страница списка заказов;
- OrdersActivity.java – класс для активности activity_orders.xml;
- activity_buying.xml – страница оформления покупки билета;
- BuyingActivity.java – класс для активности activity_buying.xml;
- CustomGridAdapter.java – класс для работы с GridView, используемого для отображения схемы посадочных мест;
- grid_item_layout.xml – верстка визуального стиля ячейки в GridView, используемого для отображения схемы посадочных мест.

3.2 Представление развертывания мобильного приложения

Диаграмма развёртывания содержит графические изображения процессоров, устройств, процессов и связей между ними. Диаграмма развертывания является единой для системы в целом и всецело отражает особенности ее реализации [2].

Диаграмма развертывания мобильного приложения продажи транспортных услуг автовокзала представлена на рисунке 33.

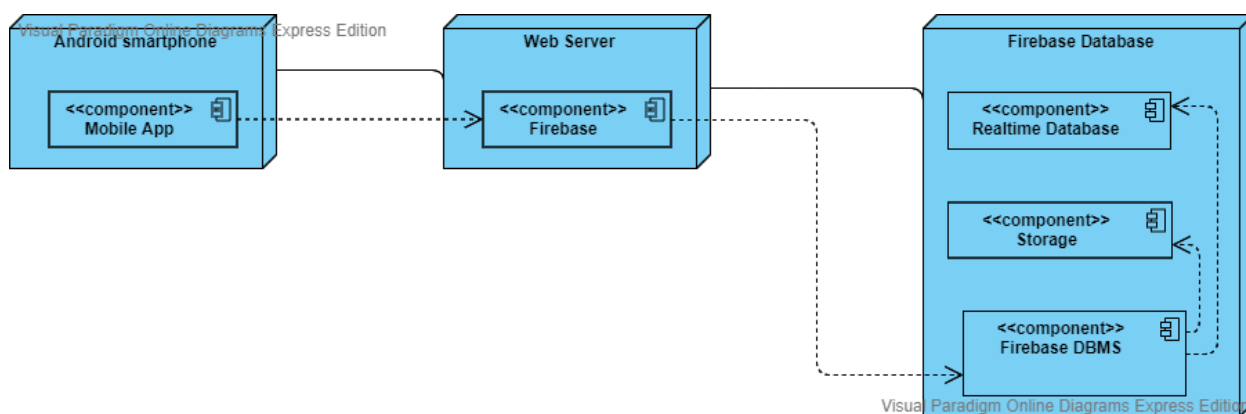


Рисунок 33 – Диаграмма развертывания

Архитектура состоит из 3 основных составляющих: смартфона на платформе Android, сервера Firebase и базы данных Firebase. С помощью смартфона пользователь осуществляет взаимодействие непосредственно с мобильным приложением продажи транспортных услуг автовокзала. Сервер Firebase позволяют принимать запросы пользователя, отправлять запросы на Firebase Database, получать ответ и возвращать его пользователю. Firebase Database, реализованный в формате документно-ориентированной базы данных, позволяют принимать запросы Firebase сервера, обрабатывать их и возвращать результат в формате JSON.

4 Контроль качества программного обеспечения мобильного приложения

Тестирование представляет собой вид деятельности в процессе разработки программного обеспечения, который направлен на обнаружение несоответствий между поведением программы и предъявленными к ней требованиями на конечном наборе тестовых данных с целью повышения качества программного продукта. Тестирование должно включать тесты различных категорий, каждая из которых отличается целевыми задачами, тестируемыми компонентами программы и методами оценки результатов. Совместное и систематическое применение различных тестов, позволяет в ходе тестирования в большем объеме выявить все дефекты программного обеспечения и достичь его высокого качества.

4.1 Функциональное тестирование мобильного приложения

Функциональное тестирование предназначено для обнаружения несоответствий между реальным поведением реализованных функций и ожидаемым поведением в соответствии с исходными требованиями. Наборы тестов, созданные для реализации функционального тестирования, должны строиться для всех входных условий, а также на границах областей допустимых значений [12].

Ручное функциональное тестирование было рассмотрено на примере варианта использования «Регистрация», представленного на диаграмме вариантов использования в разделе 1.4.

4.1.1 Определение классов эквивалентности

Как правило, исчерпывающее входное тестирование программы невозможно. Следовательно, тестирование сводится к использованию техники, при которой функционал (диапазон возможных входных значений) разделяется на группы значений, эквивалентных по воздействию на систему.

Классы эквивалентности, выделенные для поля ввода email, представлены в таблице 4.

Таблица 4 – Классы эквивалентности

Поле email	Правильные классы	Неправильные классы
Наличие @	1) Да	2) Нет
Размер строки	3) [6,35]	4) [0,6),(35,+]
Латинские буквы	5) [A-Z],[a-z]	6) [A-Я],[a-я]
Спец. символы	7) [-],[_],[.]	8) Все остальные
Первый символ-буква	9) Да	10) Нет

На основании классов эквивалентности были сформированы возможные ситуации при вводе email, которые приведены в таблице 5.

Таблица 5 – Возможные ситуации при вводе email

Входящие значения	Предполагаемый результат	Класс эквивалентности	Результат
ivanovgrisha@mail.ru	Успех	1, 3, 5,9	Успех (рисунок В.1)
ivanovOmail.ru	Ошибка	2	Ошибка (рисунок В.2)
иванов@mail.ru	Ошибка	6	Ошибка (рисунок В.3)
ivaaaaaaaaaaaaaaaaaaaaa aaaaaaaaanov@yandex. ru	Ошибка.	4	Ошибка (рисунок В.4)
ivan*ov!@mail.ru	Ошибка	8	Ошибка (рисунок В.5)
ivanov_Gr98@mail.ru	Успех	1,3,5,7,9	Успех (рисунок В.6)
98ivanov@ list.ru	Ошибка	10	Ошибка (рисунок В.7)

Результаты тестирования приведены в приложении В.

4.1.2 Анализ граничных условий

Анализ граничных условий – это техника проверки ошибок, которые могут возникать непосредственно на, выше или ниже границ входных и выходных классов эквивалентности.

В ходе анализа граничных условий были выделены следующие границы:

- физические границы: пустая строка;
- логические границы: наличие символа @, первый символ – латинская буква.
- произвольные границы: [Aa-Zz], [0123456789], [.), [], [-], количество символов в пределах [6;35];
- технологические границы: char(35).

Произвольные границы схематично представлены на рисунках 34 и 35.



Рисунок 34 – Произвольные границы для поля ввода email

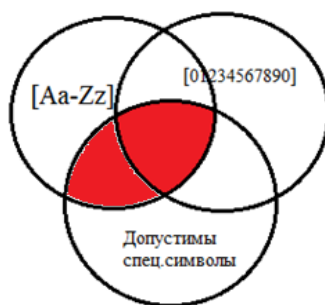


Рисунок 35 – Произвольные границы для поля ввода email

Тестовые наборы для граничных условий приведены в таблице 6.

Таблица 6 – Тестирование граничных условий

№	Тест	Значение	Границы	Ожидаемый результат	Результат
1	Вводится пустая строка	[]	1	Ошибка	Ошибка (рисунок В.8)
2	Строка содержит недопустимые специальные символы	Ivan*h@mail.ru	3	Ошибка	Ошибка (рисунок В.9)
3	Строка содержит символы русского алфавита	ПункевичР@mail.ru	3	Ошибка	Ошибка (рисунок В.10)
4	Строка содержит допустимые символы	Punkevich_E.N98@mail.ru	2,3,4	Успех	Успех (рисунок В.11)
5	Вводится строка из 36 символов.	Romaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa@mail.ru	3,4	Ошибка	Ошибка (рисунок В.12)
6	Вводится строка из 6-х символов.	r@l.ru	3	Успех	Успех (рисунок В.13)
7	Вводится строка из 35 символов.	Romaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa@mail.ru	3	Успех.	Успех (рисунок В.14)
8	Вводится строка из 5 символов.	r@m.a	3	Ошибка	Ошибка (рисунок В.15)
9	Вводится строка без символа @	ivanovOmail.ru	2	Ошибка	Ошибка (рисунок В.16)
10	Вводится строка без допустимых специальных символов	michaeljackson@mailru	3	Ошибка	Ошибка (рисунок В.17)

Результаты тестирования приведены в приложении В.

4.1.3 Построение функциональной диаграммы

Недостатком анализа граничных условий и эквивалентного разбиения является то, что они не исследуют комбинаций входных условий. Этот недостаток позволяет устранить построение функциональных диаграмм.

В ходе тестирования для наглядности использования граничных условий ввода email, была построена функциональная диаграмма, представленная на рисунке 36.

Составляющие функциональной диаграммы:

- наличие символа @;
- длина в пределах [6,35];
- первый символ-буква;
- наличие латинских букв;
- наличие цифр;
- наличие специальных символов;
- первый символ-буква латинского алфавита;
- наличие буквы или цифры;
- успех;
- ошибка.

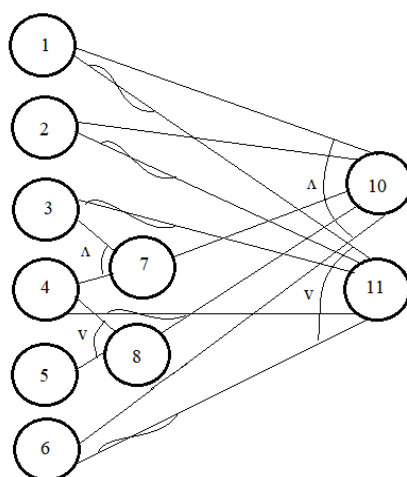


Рисунок 36 – Функциональная диаграмма

Путем методического прослеживания состояний условий диаграммы, она была преобразована в таблицу решений (таблица 7), представленную ниже. Каждый столбец таблицы решений соответствует тесту.

Таблица 7 – Таблица решений

	1	2	3	4	5	6	7	8
1	0	-	-	-	-	-	1	1
2	-	0	-	-	-	-	1	1
3	-	-	0	-	-	-	1	1
4	-	-	-	0	0	-	1	1
5	-	-	-	-	0	-	0	1
6	-	-	-	-	-	0	1	1
7	-	-	0	0	-	-	1	1
8	-	-	-	-	0	-	1	1
10	0	0	0	0	0	0	1	1
11	1	1	1	1	1	1	0	0

Единица в таблице соответствует истинному значению, а ноль ложному значению. Таблица решений используется для покрытия тестами всех возможных вариантов введения email при регистрации в системе.

4.1.4 Парное тестирование

Парное тестирование позволяет формировать тестовые данные, которые зависят от нескольких входных параметров. Тестовые наборы составляются так, чтобы каждое значение параметра хотя бы один раз сочеталось с каждым значением остальных тестируемых параметров.

При регистрации пользователю необходимо ввести имя, фамилию, отчество, номер телефона, email и пароль. Для парного тестирования было использовано программное обеспечение «allpairs». Описание входных классов эквивалентности представлено на рисунке 37.

Name	Surname	Lastname	Phone	Login	Password
GoodName	Latin_S	Latin_L	11symbol	GoodLogin	GoodPassword
Russian_N	Russian_S	Russian_L	more11	BadLogin	BadPassword
Other_N	Other_Sur	Other_L	NotNum	[1,6]	[1,8]
NullName	NullSur	NullLast	Number	[6,35]	[8,30]
(30,+)Name	(30,+)Sur	(30,+)Last	Null	(35,+)	(30,+)
DefisN	DefisS	DefisL	Less11	Null	Null

Рисунок 37 – Входные классы эквивалентности

Результаты создания тестовых случаев приведены на рисунке 38.

TEST CASES						
case	Name	Surname	Lastname	Phone	Login	Password
1	Latin_N	Latin_S	Latin_L	11symbol	GoodLogin	GoodPass
2	Latin_N	Russian_S	Russian_L	more11	BadLogin	BadPassw
3	Latin_N	Other_Sur	Other_L	NotNum	[1,6]	[1,8]
4	Latin_N	NullSur	NullLast	Number	[6,35]	[8,30]
5	Latin_N	(30,+)Sur	(30,+)Last	Null	(35,+)	(30,+)
6	Latin_N	DefisS	DefisL	Less11	Null	Null
7	Russian_N	Latin_S	Russian_L	NotNum	[6,35]	(30,+)
8	Russian_N	Russian_S	Latin_L	Number	[1,6]	Null
9	Russian_N	Other_Sur	NullLast	11symbol	BadLogin	GoodPass
10	Russian_N	NullSur	Other_L	more11	GoodLogin	BadPassw
11	Russian_N	(30,+)Sur	DefisL	11symbol	[1,6]	[8,30]
12	Russian_N	DefisS	(30,+)Last	more11	[6,35]	[1,8]
13	Other_N	Latin_S	Other_L	Number	BadLogin	[1,8]
14	Other_N	Russian_S	NullLast	NotNum	GoodLogin	[8,30]
15	Other_N	Other_Sur	Latin_L	more11	(35,+)	[8,30]
16	Other_N	NullSur	Russian_L	11symbol	Null	[1,8]
17	Other_N	(30,+)Sur	Latin_L	Less11	[6,35]	BadPassw
18	Other_N	DefisS	Russian_L	Null	[1,6]	GoodPass
19	NullName	Latin_S	NullLast	more11	[1,6]	(30,+)
20	NullName	Russian_S	Other_L	11symbol	[6,35]	Null
21	NullName	Other_Sur	Russian_L	Number	GoodLogin	BadPassw
22	NullName	NullSur	Latin_L	NotNum	BadLogin	GoodPass
23	NullName	(30,+)Sur	Russian_L	Less11	(35,+)	[1,8]
24	NullName	DefisS	Latin_L	Null	Null	[8,30]
25	(30,+)Name	Latin_S	(30,+)Last	Less11	GoodLogin	[8,30]
26	(30,+)Name	Russian_S	DefisL	Null	BadLogin	[1,8]
27	(30,+)Name	Other_Sur	Other_L	Null	[6,35]	GoodPass
28	(30,+)Name	NullSur	NullLast	Less11	[1,6]	BadPassw
29	(30,+)Name	(30,+)Sur	Other_L	more11	Null	GoodPass
30	(30,+)Name	DefisS	NullLast	11symbol	(35,+)	BadPassw
31	DefisN	Latin_S	DefisL	NotNum	(35,+)	BadPassw
32	DefisN	Russian_S	(30,+)Last	Number	Null	GoodPass
33	DefisN	Other_Sur	NullLast	Null	GoodLogin	Null
34	DefisN	NullSur	Other_L	Less11	BadLogin	(30,+)
35	DefisN	(30,+)Sur	NullLast	NotNum	Null	[1,8]
36	DefisN	DefisS	Other_L	Number	(35,+)	[8,30]
37	(30,+)Name	Other_Sur	(30,+)Last	NotNum	BadLogin	Null
38	(30,+)Name	NullSur	DefisL	Number	GoodLogin	(30,+)
39	DefisN	Other_Sur	DefisL	more11	[6,35]	(30,+)
40	DefisN	NullSur	(30,+)Last	11symbol	[1,6]	Null
41	Russian_N	Latin_S	Latin_L	Null	Null	BadPassw
42	Russian_N	Russian_S	Russian_L	Less11	(35,+)	GoodPass
43	Other_N	(30,+)Sur	(30,+)Last	11symbol	GoodLogin	(30,+)
44	Other_N	DefisS	DefisL	more11	BadLogin	Null
45	NullName	(30,+)Sur	DefisL	Number	BadLogin	GoodPass
46	NullName	DefisS	(30,+)Last	NotNum	GoodLogin	BadPassw
47	(30,+)Name	Other_Sur	Latin_L	Less11	Null	(30,+)
48	(30,+)Name	NullSur	Russian_L	Null	(35,+)	Null
49	DefisN	Latin_S	Russian_L	~11symbol	BadLogin	[8,30]
50	DefisN	Russian_S	Latin_L	~more11	GoodLogin	[1,8]
51	~Latin_N	Latin_S	~Other_L	~Null	~[1,6]	Null
52	~Latin_N	Russian_S	~NullLast	~Less11	~[6,35]	(30,+)
53	~Latin_N	(30,+)Sur	~Latin_L	~NotNum	~BadLogin	Null
54	~Latin_N	DefisS	~Russian_L	~Number	~GoodLog	(30,+)

Рисунок 38 – Входные классы эквивалентности

В результате функционального тестирования ошибок при регистрации пользователя выявлено не было.

4.2 Метрики программного кода проекта

Метрики кода являются совокупностью оценок ПО, служащей рекомендательными характеристиками и представляющую более полную информацию о программном коде. С помощью метрик можно определить сложность разрабатываемого проекта, оценить объем работ, стилистику разрабатываемой программы и усилия, потраченные каждым разработчиком для реализации того или иного решения.

Цикломатическая сложность представляет счетное число линейно независимых маршрутов через программный код. Специальная свободно распространяемая программа LocMetrics позволяет проанализировать исходный код программы. Результат анализа кода мобильного приложения продажи транспортных услуг автовокзала представлен на рисунке 39.

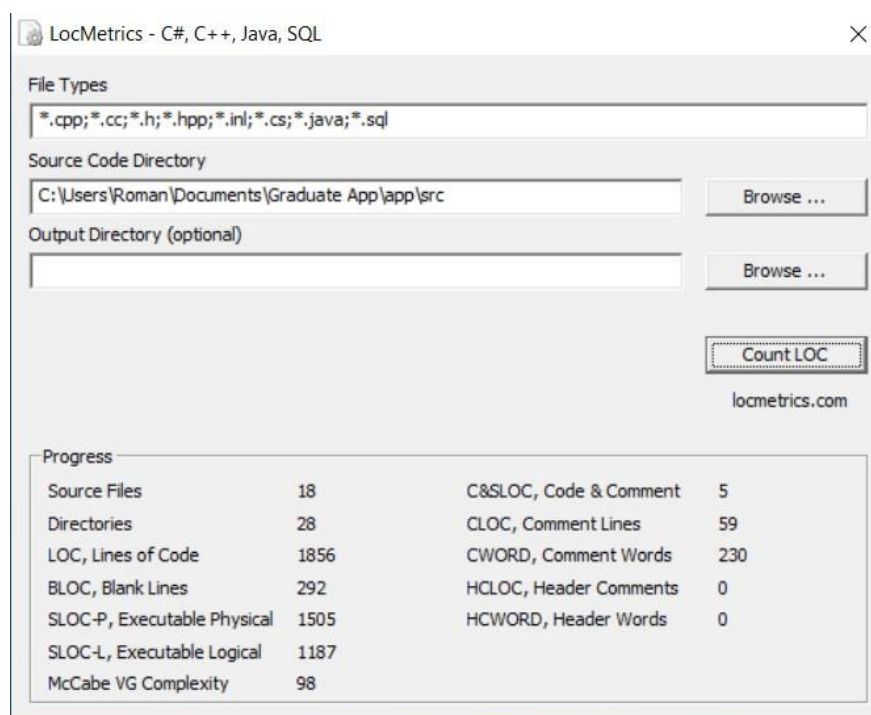
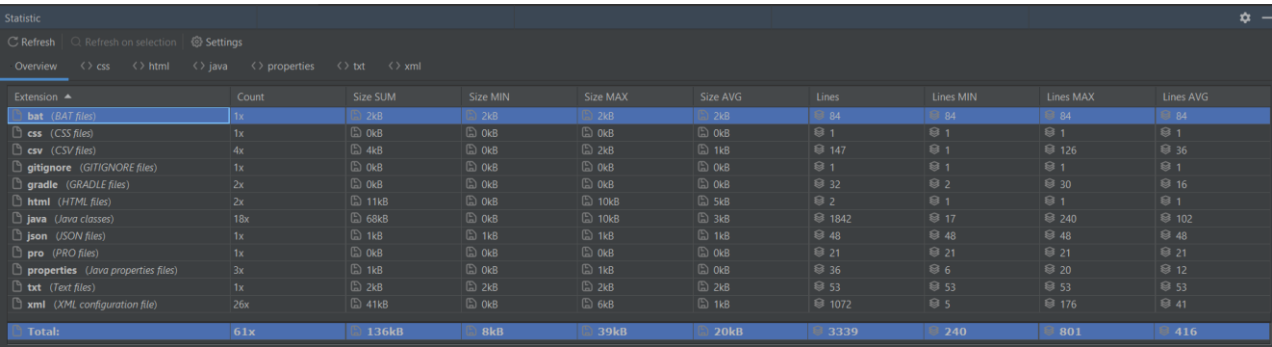


Рисунок 39 – Анализ кода в LocMetrics

Основной интересующий нас параметр – это цикломатическая сложность (McCabe VG Complexity). Значение равно 98, что говорит нам о том, что программа является достаточно сложной.

Android Studio обладает возможностью подключения плагина расчета статистики программного проекта. Статистика приведена на рисунке 40.



Extension	Count	Size SUM	Size MIN	Size MAX	Size AVG	Lines	Lines MIN	Lines MAX	Lines AVG
bat (BAT files)	1x	2kB	2kB	2kB	2kB	84	84	84	84
css (CSS files)	1x	0kB	0kB	0kB	0kB	1	1	1	1
csv (CSV files)	4x	4kB	0kB	2kB	1kB	147	1	125	36
gitignore (GITIGNORE files)	1x	0kB	0kB	0kB	0kB	1	1	1	1
gradle (GRADLE files)	2x	0kB	0kB	0kB	0kB	32	2	30	16
html (HTML files)	2x	11kB	0kB	10kB	5kB	2	1	1	1
java (Java classes)	18x	68kB	0kB	10kB	3kB	1842	17	240	102
json (JSON files)	1x	1kB	1kB	1kB	1kB	48	48	48	48
pro (PRO files)	1x	0kB	0kB	0kB	0kB	21	21	21	21
properties (Java properties files)	3x	1kB	0kB	1kB	0kB	36	6	20	12
txt (Text files)	1x	2kB	2kB	2kB	2kB	53	53	53	53
xml (XML configuration file)	26x	41kB	0kB	1kB	1kB	1072	5	176	41
Total:	61x	1.36kB	8kB	39kB	20kB	3339	240	801	416

Рисунок 40 – Статистика проекта

Статистика кода Android Studio демонстрирует то, какие языки и технологии задействованы в проекте. В данном случае видно, что наиболее сильно используется язык программирования Java и язык разметки XML.

4.3 Планирование и оценка бюджета проекта

Управление проектами – это деятельность, в ходе которой намечаются и достигаются цели проекта с учетом рисков и ограничений на трудозатраты, ресурсы и качество. Чтобы достигнуть положительного результата в проектном управлении, нужно иметь заранее составленный план, рационально применять свои знания и инструментальные средства.

При планировании проектов есть несколько основных групп планирования:

- планирование проектных задач;
- планирование расписания проекта;
- планирование команды проекта;
- управление рисками.

При проведении планирования проекта было выделено 28 задач. Перечень задач и их виды, предшественники и длительность каждой задачи представлены в таблице 8.

Таблица 8 – Перечень задач проекта

№	Название	Вид задачи	Предшественник	Длительность
1	Начало реализации проекта	Веха	-	-
2	Анализ требований и постановка задачи	Фаза	-	-
3	Анализ программных аналогов	Задача	1	2
4	Постановка задачи	Задача	3	1
5	Анализ функциональных требований	Задача	3	2
6	Выбор средств разработки	Задача	4;5	2
7	Анализ требований завершен	Веха	6	-
8	Проектирование приложений	Фаза	2	-
9	Функциональное моделирование	Задача	7	3
10	Проектирование структур данных	Задача	9	2
11	Проектирование программных средств	Задача	10	4
12	Проектирование пользовательского интерфейса	Задача	10	2
13	Проектирование завершено	Веха	11;12	-
14	Реализация	Фаза	8	-
15	Реализация структур данных	Задача	13	3
16	Реализация пользовательского интерфейса	Задача	15	2
17	Реализация программных средств	Задача	16	15
18	Реализация завершена	Веха	17	-
19	Отладка и тестирование	Фаза	14	-
20	Отладка серверной части	Задача	18	3
21	Рефакторинг	Задача	21	1
22	Анализ качества кода	Задача	22	2
23	Отладка и тестирование завершена	Веха	23	-
24	Завершение проекта	Фаза	19	-
25	Составление документации	Задача	24	2
26	Сдача в эксплуатацию	Задача	26	1
27	Проект завершен	Веха	27	-
28	Конец проекта	Веха	28	-

Для реализации каждой задачи данного проекта необходимо определить ресурсы, а также их тип и стоимостную оценку. Перечень всех ресурсов проекта представлен в таблице 9.

Таблица 9 – Ресурсы и затраты проекта

№	Название ресурса	Тип ресурса	Затраты		Начисление
			Стандартная ставка	Ставка сверхур.	
1	Программист	Трудовой	50 000 р./мес.	250 р./ч.	Пропорциональное
2	Аналитик	Трудовой	40 000 р./мес.	150 р./ч.	Пропорциональное
3	Тестировщик	Трудовой	30 000 р./мес.	200 р./ч.	Пропорциональное
4	Бумага	Материальный	300 р.	-	Пропорциональное
5	USB накопитель	Материальный	500 р.	-	Пропорциональное
6	Смартфон	Материальный	29 990р.	-	Пропорциональное
7	Компьютер	Материальный	60 000р.	-	Пропорциональное
8	Интернет	Затраты	-	-	Пропорциональное

Далее выделенные ресурсы необходимо распределить по задачам. Распределение ресурсов по задачам приведено в таблице 10.

Таблица 10 – Назначение ресурсов

№	Название	Ресурсы	Единицы (затраты)
1	2	3	4
1	Начало реализации проекта	Смартфон Компьютер	1шт. 1шт.
2	Анализ программных аналогов	Интернет Аналитик	350р 100%
3	Постановка задачи	Программист	50%
4	Анализ функциональных требований	Программист Аналитик Интернет	50% 100% 350р
5	Выбор средств разработки	Интернет Программист	350р 100%
6	Функциональное моделирование	Программист	100%
7	Проектирование структур данных	Программист	100%
8	Проектирование программных средств	Программист	70%
9	Проектирование пользовательского интерфейса	Программист	30%

Продолжение таблицы 10

1	2	3	4
10	Реализация структур данных	Программист	100%
11	Реализация пользовательского интерфейса	Программист	100%
12	Реализация программных средств	Программист	100%
13	Отладка серверной части	Программист Тестировщик	50% 100%
14	Структурное тестирование	Тестировщик	100%
15	Рефакторинг	Программист	100%
16	Анализ качества кода	Тестировщик Программист	100% 50%
17	Составление документации	Бумага Аналитик Программист Тестировщик USB накопитель Интернет	1уп. 100% 100% 100% 1шт. 350р
18	Сдача в эксплуатацию	Аналитик Программист Тестировщик	100% 100% 100%

Для планирования процесса разработки мобильного приложения продажи транспортных услуг автовокзала была выбрана среда Microsoft Office Project. Созданный проект имеет дату начала 13.04.20. В него были занесены задачи проекта, проанализированные ресурсы и соответствующие им затраты. Результат внесения ресурсов в проект представлен на рисунке 41.

	Название ресурса	Краткое название	Тип	Начисление	Стандартная ставка	Ставка сверхурочных
1	Программист	П	Трудовой	Пропорциональное	50 000,00 руб./мес	250,00 руб./ч
2	Аналитик	А	Трудовой	Пропорциональное	40 000,00 руб./мес	150,00 руб./ч
3	Тестировщик	Т	Трудовой	Пропорциональное	30 000,00 руб./мес	200,00 руб./ч
4	Бумага	Б	Материальный	Пропорциональное	300,00 руб.	
5	USB накопитель	У	Материальный	Пропорциональное	500,00 руб.	
6	Смартфон	С	Материальный	Пропорциональное	29 990,00 руб.	
7	Компьютер	К	Материальный	Пропорциональное	60 000,00 руб.	
8	Интернет	И	Затраты	Пропорциональное		

Рисунок 41 – Ресурсы проекта

Чтобы задача могла быть выполнена, для нее выделяются ресурсы материальные (оборудование) и трудовые (сотрудники). Выделение ресурса

задаче называется назначением [13]. Результаты назначений ресурсов в среде Microsoft Office Project представлены на рисунке 42.

		Название задачи	Длительность	Начало	Окончание	Предше	Названия ресурсов
1	✓	Начало реализации проекта	0 дней	Пн 13.04.20	Пн 13.04.20		Смартфон[0];Компьютер[0]
2	✓	Анализ требований и постановка	6 дней	Пн 13.04.20	Пн 20.04.20		
3	✓	Анализ программных аналогов	2 дней	Пн 13.04.20	Вт 14.04.20	1	Аналитик;Интернет[350,00 руб.]
4	✓	Постановка задачи	1 день	Ср 15.04.20	Ср 15.04.20	3	Программист[50%]
5	✓	Анализ функциональных требо	2 дней	Ср 15.04.20	Чт 16.04.20	3	Программист[50%];Аналитик;Интернет[350,00 руб.]
6	✓	Выбор средств разработки	2 дней	Пт 17.04.20	Пн 20.04.20	4,5	Программист;Интернет[350,00 руб.]
7	✓	Анализ требований завершен	0 дней	Пн 20.04.20	Пн 20.04.20	6	
8	✓	Проектирование приложений	9 дней	Вт 21.04.20	Пт 01.05.20	2	
9	✓	Функциональное моделировани	3 дней	Вт 21.04.20	Чт 23.04.20	7	Программист
10	✓	Проектирование структур данн	2 дней	Пт 24.04.20	Пн 27.04.20	9	Программист
11	✓	Проектирование программных	4 дней	Вт 28.04.20	Пт 01.05.20	10	Программист[70%]
12	✓	Проектирование пользовател	2 дней	Вт 28.04.20	Ср 29.04.20	10	Программист[30%]
13	✓	Проектирование завершено	0 дней	Пт 01.05.20	Пт 01.05.20	11;12	
14	✓	Реализация	20 дней	Пн 04.05.20	Пт 29.05.20	8	
15	✓	Реализация структур данных	3 дней	Пн 04.05.20	Ср 06.05.20	13	Программист
16	✓	Реализация пользовательского	2 дней	Чт 07.05.20	Пт 08.05.20	15	Программист
17	✓	Реализация программных сред	15 дней	Пн 11.05.20	Пт 29.05.20	16	Программист
18	✓	Реализация завершена	0 дней	Пт 29.05.20	Пт 29.05.20	17	
19	✓	Отладка и тестирование	8 дней	Пн 01.06.20	Ср 10.06.20	14	
20	✓	Отладка серверной част	3 дней	Пн 01.06.20	Ср 03.06.20	18	Программист[50%];Тестировщик
21	✓	Структурное тестирование	2 дней	Чт 04.06.20	Пт 05.06.20	20	Тестировщик
22	✓	Рефакторинг	1 день	Пн 08.06.20	Пн 08.06.20	21	Программист
23	✓	Отладка и тестирование завер	0 дней	Ср 10.06.20	Ср 10.06.20	22	
24	✓	Завершение проекта	2 дня	Чт 11.06.20	Сб 13.06.20	19	
25	✓	Составление документации	2 дня	Чт 11.06.20	Пт 12.06.20	23	Аналитик;Бумага[1];Интернет[350,00 руб.];Программис
26	✓	Сдача в эксплуатацию	1 день	Пт 12.06.20	Пт 12.06.20	25	Аналитик;Программист;Тестировщик
27	✓	Проект завершен	0 дней	Сб 13.06.20	Сб 13.06.20	26	
28	✓	Конец проекта	0 дней	Сб 13.06.20	Сб 13.06.20	27	

Рисунок 42 – Назначение ресурсов задачам

После внесения в проект поставленных задач с распределением по датам реализации и ресурсам среда Microsoft Office Project автоматически построит диаграмму Ганта. Диаграмма Ганта представлена на рисунке 43.

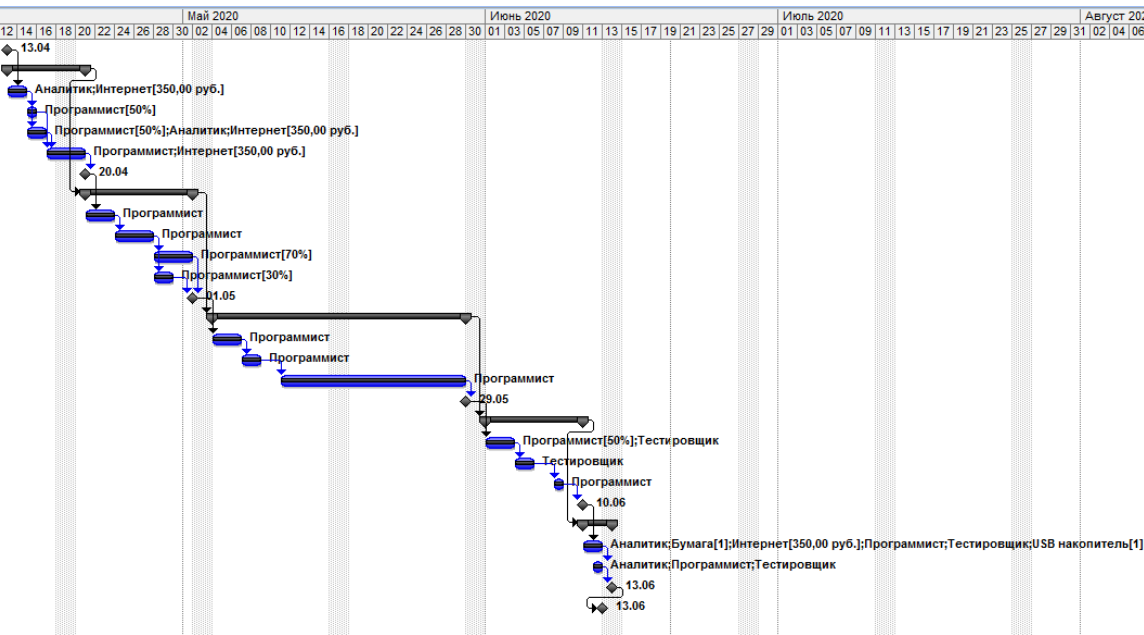


Рисунок 43 – Диаграмма Ганта

Диаграмма Ганта иллюстрирует календарный график выполнения работ по проекту.

При помощи Microsoft Office Project был сформирован обзорный отчет проекта планирования разработки мобильного приложения продажи транспортных услуг автовокзала. В этом отчете содержатся данные о задаче и количестве затраченных средств на ее выполнение. Обзорный отчет проекта представлен на рисунке 44.

<u>Даты</u>			
Начало:	Пн 13.04.20	Окончание:	Сб 13.06.20
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	Пн 13.04.20	Факт. окончание:	Сб 13.06.20
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
<u>Длительность</u>			
Текущий план:	45 дней	Оставшаяся:	0 дней
Базовый план:	0 дней?	Фактическая:	45 дней
Отклонение:	45 дней	Процент завершения:	100%
<u>Трудозатраты</u>			
Текущий план:	419,2 ч	Оставшиеся:	0 ч
Базовый план:	0 ч	Фактические:	419,2 ч
Отклонение:	419,2 ч	Процент завершения:	100%
<u>Затраты</u>			
Текущий план:	121 700,00 руб.	Оставшиеся:	1 400,00 руб.
Базовый план:	0,00 руб.	Фактические:	120 300,00 руб.
Отклонение:	121 700,00 руб.		
<u>Состояние задачи</u>		<u>Состояние ресурса</u>	
Еще не начатые задачи	0	Трудовые ресурсы:	4
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дост	0
Завершенные задачи:	28	Материальные ресурсы:	4
Итого задач:	28	Итого ресурсов:	8

Рисунок 44 – Обзорный отчет проекта

В этом отчете содержатся данные о дате начала и конца проекта, длительность и процент завершения всех задач верхнего уровня. Так же построенной таблице отчёта содержатся сведения о затратах для всех задач верхнего уровня. Срок реализации проекта составил 45 рабочих дней с учетом выходных дней и праздников, а бюджет проекта – 121 700 рублей. Каждая задача успешно выполнена в срок и завершена на 100 %.

Также Microsoft Office Project позволил построить отчет по движению денежных средств. Отчет по затратам представлен на рисунке 45.

	2020	2021	Итого
Начало реализации проекта			
Анализ требований и постановка задачи			
Анализ программных аналогов	4 350,00 руб.		4 350,00 руб.
Постановка задачи	1 250,00 руб.		1 250,00 руб.
Анализ функциональных требований	6 850,00 руб.		6 850,00 руб.
Выбор средств разработки	5 350,00 руб.		5 350,00 руб.
Анализ требований завершен			
Проектирование приложений			
Функциональное моделирование	7 500,00 руб.		7 500,00 руб.
Проектирование структур данных	5 000,00 руб.		5 000,00 руб.
Проектирование программных средств	7 000,00 руб.		7 000,00 руб.
Проектирование пользовательского интерфейса	1 500,00 руб.		1 500,00 руб.
Проектирование завершено			
Реализация			
Реализация структур данных	7 500,00 руб.		7 500,00 руб.
Реализация пользовательского интерфейса	5 000,00 руб.		5 000,00 руб.
Реализация программных средств	37 500,00 руб.		37 500,00 руб.
Реализация завершена			
Отладка и тестирование			
Отладка серверной част	8 250,00 руб.		8 250,00 руб.
Структурное тестирование	3 000,00 руб.		3 000,00 руб.
Рефакторинг	2 500,00 руб.		2 500,00 руб.
Отладка и тестирование завершена			
Завершение проекта			
Составление документации	13 150,00 руб.		13 150,00 руб.
Сдача в эксплуатацию	6 000,00 руб.		6 000,00 руб.
Проект завершен			
Конец проекта			
Итого	121 700,00 руб.		121 700,00 руб.

Рисунок 45 – Отчет о движении денежных средств

Отчет по исполнителям представлен на рисунке 46.

Ид.	Название ресурса	Трудозатраты				
1	Программист	299,2 ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
4	Постановка задачи	50%	4 ч	0 дн	Ср 15.04.20	Ср 15.04.20
5	Анализ функциональных требований	50%	5 ч	0 дн	Ср 15.04.20	Чт 15.04.20
6	Выбор средств разработки	100%	15 ч	0 дн	Пт 17.04.21	Пн 20.04.20
9	Функциональное моделирование	100%	24 ч	0 дн	Вт 21.04.20	Чт 23.04.20
10	Проектирование структур данных	100%	15 ч	0 дн	Пт 24.04.21	Пн 27.04.20
11	Проектирование программных средств	70%	22,4 ч	0 дн	Вт 25.04.20	Пт 01.05.21
12	Проектирование пользовательского интерфейса	30%	4,5 ч	0 дн	Вт 25.04.20	Ср 29.04.20
15	Реализация структур данных	100%	24 ч	0 дн	Пн 04.05.20	Ср 05.05.20
16	Реализация пользовательского интерфейса	100%	15 ч	0 дн	Чт 07.05.20	Пт 05.05.21
17	Реализация программных средств	100%	120 ч	0 дн	Пн 11.05.20	Пт 29.05.21
20	Отладка серверной части	50%	12 ч	0 дн	Пн 01.05.20	Ср 03.05.20
22	Рефакторинг	100%	5 ч	0 дн	Пн 05.05.21	Пн 05.05.20
25	Сдача в эксплуатацию	100%	5 ч	0 дн	Пт 12.05.21	Пт 12.05.21
26	Составление документации	100%	15 ч	0 дн	Чт 11.05.20	Пт 12.05.21
2	Аналитик	56 ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
3	Анализ программных аналогов	100%	15 ч	0 дн	Пн 13.04.20	Вт 14.04.20
25	Составление документации	100%	15 ч	0 дн	Чт 11.05.20	Пт 12.05.21
26	Сдача в эксплуатацию	100%	5 ч	0 дн	Пт 12.05.21	Пт 12.05.21
5	Анализ функциональных требований	100%	15 ч	0 дн	Ср 15.04.20	Чт 15.04.20
3	Тестировщик	64 ч				
Ид.	Название задачи	Единицы	Трудозатраты	Задержка	Начало	Окончание
21	Структурное тестирование	100%	15 ч	0 дн	Чт 04.05.20	Пт 05.05.21
20	Отладка серверной части	100%	24 ч	0 дн	Пн 01.05.20	Ср 03.05.20
25	Сдача в эксплуатацию	100%	5 ч	0 дн	Пт 12.05.21	Пт 12.05.21
26	Составление документации	100%	15 ч	0 дн	Чт 11.05.20	Пт 12.05.21

Рисунок 46 – Отчет по делам исполнителей

По данному отчету можно выделить наиболее важных исполнителей что позволяет принимать грамотные управленческие решения. Исходя из приведенных отчетов, а также при помощи оценки бюджета и сроков проекта можно сделать вывод, что разработка мобильного приложения продажи транспортных услуг автовокзала является экономически выгодным.

Заключение

Разработано мобильное приложение продажи транспортных услуг автовокзала, имеющее клиент-серверную архитектуру. Клиентская часть работы реализована на языке объектно-ориентированного программирования Java в Android Studio, а серверная часть с помощью платформы Firebase от компании Google.

Данное мобильное приложение позволяет пользователю просматривать расписание выбранного города или станции, искать рейсы между городами, а также строить маршруты с пересадками при отсутствии прямого сообщения. Пользователю также доступна возможность приобретения билета на выбранный рейс с возможностью выбора посадочного места и, в случае необходимости, оформление возврата билета в режиме онлайн со своего смартфона.

На этапе анализа качества программного обеспечения было проведено функциональное тестирование и определены метрики программного кода проекта. Из результата анализа результатов тестирования следует вывод, что разработанное мобильное приложение продажи транспортных услуг автовокзала функционирует корректно.

В ходе планирования и оценки бюджета разработки проекта с помощью программного продукта Microsoft Project выяснено, что длительность разработки проекта равна 45 дням, а бюджет равен 121 700 рублям. Учитывая востребованность в подобном программном продукте, результат оценки бюджета и длительности разработки проекта говорит о его экономической выгоды и конкурентоспособности.

Таким образом, в результате выполнения выпускной квалификационной работы разработано мобильное приложение продажи транспортных услуг автовокзала, отвечающее всем требованиям технического задания.

Список использованных источников

1. Статистика Министерства транспорта РФ // Сайт Министерства транспорта РФ [Электронный ресурс]. URL: <https://www.mintrans.ru/ministry/results/180/documents> (дата обращения 10.05.2020).
2. Киммел, П. UML. Основы визуального анализа и проектирования / П. Киммел. – М.: НТ Пресс, 2008. – 272 с.
3. Дубейковский, В. И. Эффективное моделирование с Allfusion Process Modeler 4.1.4 и Allfusion PM / В.И. Дубейковский – Издательство: Диалог-МИФИ, 2007. – 382 с.
4. Дворянин, Д. М. К вопросу о понятиях UX UI дизайна / Д. М. Дворянин, К. С. Скубников // Аллея науки. – 2017. – С. 781-785.
5. Парамонов, И. В. Язык программирования Java и Java-технологии / И.В. Парамонов. – Издательство: ЯрГУ, 2006. – 91 с.
6. Введение в платформу Android. // Сайт о программировании Metanit.com [Электронный ресурс]. URL: <https://metanit.com/java/android/1.1.php> (дата обращения 17.05.2020).
7. Firebase: платформа разработки приложений. // Сайт разработчиков Firebase [Электронный ресурс]. URL: <https://firebase.appttractor.ru> (дата обращения 20.05.2020).
8. Высокопроизводительные нереляционные базы данных с гибкими моделями данных // Сайт компании Amazon.com [Электронный ресурс]. URL: <https://aws.amazon.com/ru/nosql/> (дата обращения 21.05.2019).
9. Хаф, Л. Методологии разработки программного обеспечения. Rational Unified Process / Л. Хаф // КомпьютерПресс, 2004. – С. 177 – 180.
10. Рабочие процессы RUP и диаграммы UML. // Computer-Aided Software Engineering Club [Электронный ресурс]. URL: http://www.caseclub.ru/articles/rup_uml.html (дата обращения 28.05.2019).

11. Горбаченко, В. И. Проектирование информационных систем с СА Erwin Modeling Suite / В. И. Горбаченко, Г. Ф. Убиенных, Г. В. Бобрышева – Пенза.: Изд-во ПГУ, 2012. – 154 с.
12. Михалицын, М. М. Технология программирования. Проектирование и тестирование программ: Учеб. пособие / М. М. Михалицын – Новосибирск.: СГГА, 2003. – 73 с.
13. Куперштейн, В. И. Microsoft Project в управлении проектами. / В. И. Куперштейн, А.В. Цветкова. – СПб.: БХВ-Петербург, 2011. – 415 с.

Приложение А
ГЛОССАРИЙ
(справочное)

Мобильное приложение (МП) – программное обеспечение, предназначенное для работы на смартфонах, планшетах и других мобильных устройствах, разработанное для конкретной платформы (iOS, Android, Windows Phone и т. д.).

База данных (БД) – организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

Система управления базами данных (СУБД) – совокупность языковых и программных средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных одним или несколькими пользователями.

UML (Unified Modeling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

Приложение Б
ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ НА ЯЗЫКЕ JAVA
(рекомендуемое)

Листинг Б.1 – Файл класса на языке Java «BuyingActivity.java»

```

public class BuyingActivity extends AppCompatActivity {
    FirebaseAuth mAuth;
    FirebaseDatabase mDatabase;
    DatabaseReference myRef;
    FirebaseUser user;
    Trip currentTrip;
    TextView textTicketDepart;
    TextView textTicketArr;
    TextView textTicketTimeDepart;
    TextView textTicketTimeArr;
    TextView textTicketPrice;
    TextView textSelectedPlace;
    GridView gridView;
    Button btnTicketBuy;
    EditText editNumCard;
    EditText editDateCard;
    EditText editCVCCard;
    List<Trip> listRes;
    String currentTripId;
    String currentDate;
    String currentStationId;
    String currentDepartureId;
    String currentDestinationId;
    String currentDepartureTime;
    String currentArrivalTime;
    String currentDestinationDate;
    int selectedPlace;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_buying);
        final Bundle arguments = getIntent().getExtras();
        textTicketDepart = findViewById(R.id.textTicketDepart);
        textTicketArr = findViewById(R.id.textTicketArr);
        textTicketTimeDepart = findViewById(R.id.textTicketTimeDepart);
        textTicketTimeArr = findViewById(R.id.textTicketTimeArr);
        textTicketPrice = findViewById(R.id.textTicketPrice);
        gridView = (GridView) findViewById(R.id.gridView);
        btnTicketBuy = findViewById(R.id.btnTicketBuy);
        editNumCard = findViewById(R.id.editNumCard);
        editDateCard = findViewById(R.id.editDateCard);
        editCVCCard = findViewById(R.id.editCVCCard);
        textSelectedPlace = findViewById(R.id.textSelectedPlace);
        selectedPlace = -1;
        currentTripId = arguments.get("trip_id").toString();
        currentDate = arguments.get("date").toString();
        currentStationId = arguments.get("station_id").toString();
        currentDepartureId = arguments.get("departure_id").toString();
        currentDestinationId = arguments.get("destination_id").toString();
        currentDepartureTime = arguments.get("departure_time").toString();
        currentArrivalTime = arguments.get("arrival_time").toString();
        currentDestinationDate = arguments.get("destination_date").toString();
        user = mAuth.getInstance().getCurrentUser();
        myRef = FirebaseDatabase.getInstance().getReference();
        listRes = new ArrayList<>();

        myRef.child("stations").child(currentStationId).child(currentDate).addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

```

```

        for (DataSnapshot dataValues : dataSnapshot.getChildren()) {
            Trip trip = dataValues.getValue(Trip.class);
            listRes.add(trip);
        }
        fillFields();
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {

    }

    });

}

public void fillFields()
{
    getCurrentTrip(listRes);
    List<Place> image_details = getListData();
    gridView.setAdapter(new CustomGridAdapter(this, image_details));
    gridView.setOnItemClickListener(new AdapterView.OnItemClickListener()
    {
        @Override
        public void onItemClick(AdapterView<?> a, View v, int position,
long id) {
            selectedPlace = position + 1;
            textSelectedPlace.setText("Выбранное место: " +
selectedPlace);
            Toast.makeText(BuyingActivity.this, "Выбрано место "
+ Integer.toString(position + 1) ,
Toast.LENGTH_LONG).show();
        }
    });
    textTicketDepart.setText(currentTrip.getDeparture_id());
    textTicketArr.setText(currentTrip.getDestination_id());
    textTicketTimeDepart.setText(currentTrip.getDeparture_time());
    textTicketTimeArr.setText(currentTrip.getArrival_time());
    textTicketPrice.setText("Стоимость билета: " + currentTrip.getPrice()
+ "руб.");

    btnTicketBuy.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Ticket ticket = new Ticket(currentStationId, currentDate,
currentTripId,
Integer.toString(selectedPlace), currentDepartureId,
currentDestinationId,
currentDepartureTime, currentDestinationDate,
currentArrivalTime);
            Map<String, Object> postValues = ticket.toMap();
            Map<String, Object> childUpdates = new HashMap<>();
            childUpdates.put(user.getUid() + currentTripId, postValues);

myRef.child("users").child(user.getUid()).child("tickets").updateChildren(chi
ldUpdates);
        }
    });
}

public void getCurrentTrip(List<Trip> listRes)
{
    for (int i = 0; i < listRes.size(); i++) {
        if (listRes.get(i).getTrip_id().equals(currentTripId)) {
            currentTrip = new Trip(listRes.get(i));

```

```

        return;
    }
}

private List<Place> getListData() {
    int capacity = 0;
    switch (currentTrip.getTypetrans_od()){
        case ("1"):
            capacity = Place.type_1;
            break;
        case ("2"):
            capacity = Place.type_2;
            break;
        default:
            capacity = 40;
            break;
    }
    String numberOccup[] = currentTrip.getNumber_of_occup().split(",");
    List<String> listOccup = Arrays.asList(numberOccup);
    List<Place> listPlaces = new ArrayList<Place>();
    for (int i = 1; i <= capacity; i++)
    {
        Place place = new Place(i, true);
        if (listOccup.contains(Integer.toString(i)))
            place.setStatus(false);
        listPlaces.add(place);
    }
    return listPlaces;
}
}

```

Листинг Б.2 – Файл класса на языке Java «MainActivity.java»

```

public class MainActivity extends AppCompatActivity {
    FirebaseAuth mAuth;
    FirebaseAuth.AuthStateListener mAuthListener;
    Button btnRegister;
    Button btnSignIn;
    EditText editMainEmail;
    EditText editMainPassword;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mAuth = FirebaseAuth.getInstance();
        btnRegister = findViewById(R.id.btnRegister);
        btnSignIn = findViewById(R.id.btnSignIn);
        editMainEmail = findViewById(R.id.editMainEmail);
        editMainPassword = findViewById(R.id.editMainPassword);
        btnRegister.setOnClickListener(
            new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Intent intent = new Intent(MainActivity.this,
RegisterActivity.class);
                    startActivity(intent);
                }
            }
        );
    }
}

```

```

        btnSignIn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (isEmailValid(editMainEmail.getText().toString()))
                    signin(editMainEmail.getText().toString(),
editMainPassword.getText().toString());
                else
                    Toast.makeText(MainActivity.this, "Email введен
некорректно",
                                Toast.LENGTH_SHORT).show();
            }
        });

        mAuthListener = new FirebaseAuth.AuthStateListener() {
            @Override
            public void onAuthStateChanged(@NonNull FirebaseAuth
firebaseAuth) {
                FirebaseUser user = firebaseAuth.getCurrentUser();
                if (user != null) {
                    Intent intent = new Intent(MainActivity.this,
MenuActivity.class);
                    startActivity(intent);
                    Log.d("1", "Sign in");
                } else {
                    Log.d("1", "Sign in error");
                }
            }
        };

        public void signin(String email, String password) {
            mAuth.signInWithEmailAndPassword(email,
password).addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        Toast.makeText(MainActivity.this, "Авторизация успешна",
Toast.LENGTH_LONG).show();
                        Intent intent = new Intent(MainActivity.this,
MenuActivity.class);
                        startActivity(intent);
                        Log.d("1", "Sign in");
                    } else {
                        Toast.makeText(MainActivity.this, "Авторизация успешна",
Toast.LENGTH_SHORT).show(); // ИЗМЕНИТЬ!!!!!!!
                        Log.d("1", "Sign in error");
                    }
                }
            });
        }

        public boolean isEmailValid(String email) {
            if (email.equals("")) // проверка на пустоту
                return false;
            if (email.isEmpty()) // повторная проверка на пустоту
                return false;
            if (email.length() > 35 || email.length() < 6) // длина от 6 до 35
СИМВОЛОВ
                return false;
            if (!email.contains("@")) // содержит символ @
                return false;
            if (!email.contains(".")) // содержит символ .
                return false;

```

```

        if (email.matches("^[a-яА-Я]+@[a-z0-9.-]+\\.[a-z]{2,6}$")) // только
латинские буквы
            return false;
        if (email.matches("^[!#;%:*]+$")) // содержит лишние символы
            return false;
        if (email.contains("*"))
            return false;
        if (!email.substring(0, 1).matches("^[a-zA-Z]$")) // первый символ
латин буква
            return false;
        return true;
    }
}

```

Листинг Б.3 – Файл класса на языке Java «MenuActivity.java»

```

public class MenuActivity extends AppCompatActivity {
    private FirebaseAuth mAuth;
    private DatabaseReference myRef;
    private FirebaseAnalytics mFirebaseAnalytics;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_menu);

        BottomNavigationView navigation = findViewById(R.id.navigation);

        navigation.setNavigationItemSelectedListener(mOnNavigationItemSelectedListener);

        navigation.setSelectedItemId(R.id.navigation_search);
        mFirebaseAnalytics = FirebaseAnalytics.getInstance(this);
        FirebaseUser user = mAuth.getInstance().getCurrentUser();
        myRef = FirebaseDatabase.getInstance().getReference();
    }

    private BottomNavigationView.OnNavigationItemSelectedListener
mOnNavigationItemSelectedListener
        = new BottomNavigationView.OnNavigationItemSelectedListener() {

        @Override
        public boolean onNavigationItemSelectedListener(@NonNull MenuItem item) {
            switch (item.getItemId()) {
                case R.id.navigation_profile:
                    loadFragment(ProfileActivity.newInstance());
                    return true;
                case R.id.navigation_search:
                    loadFragment(SearchActivity.newInstance());
                    return true;
                case R.id.navigation_orders:
                    loadFragment(OrdersActivity.newInstance());
                    return true;
                case R.id.navigation_schedule:
                    loadFragment(ScheduleActivity.newInstance());
                    return true;
            }
            return false;
        }
    };

    private void loadFragment(Fragment fragment) {
        FragmentTransaction ft =
getSupportFragmentManager().beginTransaction();

```

```

        ft.replace(R.id.fl_content, fragment);
        ft.commit();
    }
}

```

Листинг Б.4 – Файл класса на языке Java «OrdersActivity.java»

```

public class OrdersActivity extends Fragment {

    FirebaseAuth      mAuth;
    DatabaseReference myRef;
    FirebaseUser      user;

    View              view;
    TableLayout        tableOrders;
    Button             tempButton;

    //List<Ticket>      listTicket;

    public OrdersActivity() {

    }

    public static OrdersActivity newInstance() {
        return new OrdersActivity();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        view = inflater.inflate(R.layout.activity_orders, container, false);

        tableOrders = view.findViewById(R.id.tableOrders);
        tempButton = new Button(view.getContext());

        user = mAuth.getInstance().getCurrentUser();
        myRef = FirebaseDatabase.getInstance().getReference();

        //listTicket = new ArrayList<>();

        FirebaseUser user = mAuth.getInstance().getCurrentUser();
        myRef = FirebaseDatabase.getInstance().getReference();

        myRef.child("users").child(user.getId()).child("tickets").addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                List<Ticket> listTicket = new ArrayList<>();
                for (DataSnapshot dataValues : dataSnapshot.getChildren()) {
                    Ticket ticket = dataValues.getValue(Ticket.class);
                    listTicket.add(ticket);
                }
                fillTableOfOrders(listTicket);
            }
            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
    }
}

```

```

        return view;
    }

    public void fillTableOfOrders(final List<Ticket> listTicket) {
        for (int i = 0; i < listTicket.size(); i++) {
            TableRow rowOrderRoute = new TableRow(view.getContext());
            rowOrderRoute.setGravity(Gravity.LEFT);
            textOrderDepart.setWidth(tableOrders.getWidth() / 2);
            textOrderDepart.setText(listTicket.get(i).departure_id + " ");
            rowOrderRoute.addView(textOrderDepart);
            rowOrderRoute.addView(textOrderArrival);
            tableOrders.addView(rowOrderRoute);
            TableRow rowOrderTime = new TableRow(view.getContext());
            rowOrderTime.setGravity(Gravity.LEFT);
            timeOrderDepart.setText(listTicket.get(i).getDate() + " " +
                listTicket.get(i).departure_time);
            timeOrderArrival.setWidth(tableOrders.getWidth() / 2);
            timeOrderArrival.setText(listTicket.get(i).getDestination_date()
+ " " + listTicket.get(i).arrival_time);
            timeOrderArrival.setTextSize(14);
            timeOrderArrival.setGravity(Gravity.CENTER);
            rowOrderTime.addView(timeOrderDepart);
            rowOrderTime.addView(timeOrderArrival);
            tableOrders.addView(rowOrderTime);
            TableRow rowOrderPlace = new TableRow(view.getContext());
            rowOrderPlace.setGravity(Gravity.CENTER_HORIZONTAL);
            TextView textOrderPlace = new TextView(view.getContext());
            textOrderPlace.setText("Посадочное место №" +
listTicket.get(i).place_number);
            textOrderPlace.setTextSize(14);
            textOrderPlace.setGravity(Gravity.CENTER);
            rowOrderPlace.addView(textOrderPlace);
            tableOrders.addView(rowOrderPlace);
            TableRow rowBtnCancel = new TableRow(view.getContext());
            rowBtnCancel.setGravity(Gravity.CENTER_HORIZONTAL);
            final Button btnCancel = new Button(view.getContext());
            btnCancel.setText("Отменить");
            btnCancel.setId(i);
            btnCancel.setGravity(Gravity.CENTER);
            btnCancel.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Toast.makeText(view.getContext(), "Поездка отменена",
                        Toast.LENGTH_SHORT).show();
                }
            });
            rowBtnCancel.addView(btnCancel);
            tableOrders.addView(rowBtnCancel);
        }
    }
}

```

Листинг Б.5 – Файл класса на языке Java «ProfileActivity.java»

```

public class ProfileActivity extends Fragment {
    FirebaseAuth mAuth;
    private DatabaseReference myRef;
    FirebaseUser user;
    View view;
    TextView LastName;
    TextView FirstName;
    TextView MiddleName;
}

```

```

    TextView Email;
    ImageView Photo;
    Button btnUnsign;
    TableLayout tableAdminPanel;

    public ProfileActivity() {
    }

    public static ProfileActivity newInstance() {
        return new ProfileActivity();
    }

    @Override
    public View onCreateView(final LayoutInflater inflater, ViewGroup
container,
                                Bundle savedInstanceState) {

        view = inflater.inflate(R.layout.activity_profile, container, false);
        tableAdminPanel = view.findViewById(R.id.tableAdminPanel);
        LastName = view.findViewById(R.id.profileLastName);
        FirstName = view.findViewById(R.id.profileFirstName);
        MiddleName = view.findViewById(R.id.profileMiddleName);
        Email = view.findViewById(R.id.profileEmail);
        Photo = view.findViewById(R.id.profilePhoto);
        btnUnsign = view.findViewById(R.id.btnUnsign);

        btnUnsign.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mAuth.signOut();
            }
        });

        user = mAuth.getInstance().getCurrentUser();
        myRef = FirebaseDatabase.getInstance().getReference();

        FirebaseStorage storage = FirebaseStorage.getInstance();
        StorageReference gsReference =
storage.getReferenceFromUrl("gs://busstationapp-dd602.appspot.com/ryan.jpg");

        final long ONE_MEGABYTE = 1024 * 1024;
        gsReference.getBytes(ONE_MEGABYTE).addOnSuccessListener(new
OnSuccessListener<byte[]>() {
            @Override
            public void onSuccess(byte[] bytes) {
                Bitmap bitmap = BitmapFactory.decodeByteArray(bytes, 0,
bytes.length);
                Photo.setImageBitmap(bitmap);
            }
        });

        myRef.child("users").child(user.getId()).addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                User appuser = dataSnapshot.getValue(User.class);
                LastName.setText(appuser.last_name);
                FirstName.setText(appuser.first_name);
                MiddleName.setText(appuser.middle_name);
                Email.setText(appuser.email);
            }
        })
    }

```

```

        @Override
        public void onCancelled(DatabaseError error) {
            Log.d("1", "get data from DB error");
        }
    });

    if (user.getEmail().equals("test@mail.ru")) {
        // Строка кнопка "купить"
        TableRow rowBtnSendData = new TableRow(view.getContext());
        rowBtnSendData.setGravity(Gravity.CENTER_HORIZONTAL);
        final Button btnSendData = new Button(view.getContext());
        btnSendData.setText("Send data");
        btnSendData.setGravity(Gravity.CENTER);
        btnSendData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Map<String, Object> postValues = trip.toMap();
                Map<String, Object> childUpdates = new HashMap<>();
                childUpdates.put(new_trip_id, postValues);
                myRef.child("stations").child(new_departure_id)

                .child(new_departure_date).updateChildren(childUpdates);
            }
        });
        rowBtnSendData.addView(btnSendData);
        tableAdminPanel.addView(rowBtnSendData);
    }

    return view;
}
}

```

Листинг Б.6 – Файл класса на языке Java «RegisterActivity.java»

```

public class RegisterActivity extends AppCompatActivity {
    FirebaseAuth mAuth;
    DatabaseReference myRef;
    static final String TAG = "EmailPassword";
    Button btnReg;
    Button btnBack;
    EditText editLastName;
    EditText editFirstName;
    EditText editMiddleName;
    EditText editEmail;
    EditText editPassword;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_register);
        btnReg = findViewById(R.id.btnReg);
        btnBack = findViewById(R.id.btnBack);
        editLastName = findViewById(R.id.editLastName);
        editFirstName = findViewById(R.id.editFirstName);
        editMiddleName = findViewById(R.id.editMiddleName);
        editEmail = findViewById(R.id.editEmail);
        editPassword = findViewById(R.id.editPassword);
        mAuth = FirebaseAuth.getInstance();
        btnReg.setOnClickListener(new View.OnClickListener() {

```



```

                                Toast.makeText(RegisterActivity.this,
"Ошибка",
                                Toast.LENGTH_SHORT).show();
                                }
                            }
                        });
                    }
                }
            }
        }
    }
}

```

Листинг Б.7 – Файл класса на языке Java «ScheduleActivity.java»

```

package com.example.busstationapp;
public class ScheduleActivity extends Fragment {
    FirebaseAuth mAuth;
    DatabaseReference myRef;
    View view;
    EditText editSchedule;
    CalendarView calendarSchedule;
    Button showSchedule;
    TableLayout table;
    Trip buyingTrip;
    SelectedDate selectedDate;

    public ScheduleActivity() {
    }

    public static ScheduleActivity newInstance() {
        return new ScheduleActivity();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        view = inflater.inflate(R.layout.activity_schedule, container,
false);

        editSchedule = view.findViewById(R.id.editSchedule);
        calendarSchedule = view.findViewById(R.id.calendarSchedule);
        showSchedule = view.findViewById(R.id.btnShowSchedule);
        table = view.findViewById(R.id.tableSchedule);

        FirebaseUser user = mAuth.getInstance().getCurrentUser();
        myRef = FirebaseDatabase.getInstance().getReference();
        selectedDate = new SelectedDate(6,9,2020); // дефолтное значение для
безопасности

        calendarSchedule.setOnDateChangeListener(new
CalendarView.OnDateChangeListener() {
            @Override
            public void onSelectedDayChange(CalendarView view, int year,
                int month, int dayOfMonth) {
                selectedDate.setDay(dayOfMonth);
                selectedDate.setMonth(month + 1);
                selectedDate.setYear(year);

                Toast.makeText(view.getContext(), selectedDate.showDate(),
Toast.LENGTH_LONG).show();
            }
        });

        showSchedule.setOnClickListener(new View.OnClickListener() {
            @Override

```

```

        public void onClick(View v) {
            if (!editSchedule.getText().equals("")) {

myRef.child("stations").child(editSchedule.getText().toString())
                .child(selectedDate.showDate())
                .addListenerForSingleValueEvent(new

ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot
dataSnapshot) {
                List<Trip> listRes = new ArrayList<>();
                for (DataSnapshot dataValues :
dataSnapshot.getChildren()) {
                    Trip trip = dataValues.getValue(Trip.class);
                    listRes.add(trip);
                }
                fillTable(listRes);
            }

            @Override
            public void onCancelled(@NonNull DatabaseError
databaseError) {

            }
        });
    }
}

return view;
}

public void fillTable(final List<Trip> listRes) {
    for (int i = 0; i < listRes.size(); i++) {
        // Строка <пункт_отправления> <пункт прибытия>
        buyingTrip = listRes.get(i);
        TableRow rowRoute = new TableRow(view.getContext());
        rowRoute.setGravity(Gravity.CENTER_HORIZONTAL);
        TextView textNameDepart = new TextView(view.getContext());
        TextView textNameArrival = new TextView(view.getContext());
        textNameDepart.setText(listRes.get(i).getDeparture_id());
        rowRoute.addView(textNameDepart);
        rowRoute.addView(textNameArrival);
        table.addView(rowRoute);
        // Строка <время отправления> <время прибытия>
        TableRow rowTimeDepartArr = new TableRow(view.getContext());
        rowTimeDepartArr.setGravity(Gravity.CENTER_HORIZONTAL);
        textTimeDepart.setGravity(Gravity.CENTER);
        textTimeArrival.setText(listRes.get(i).getDestination_date() + "
" + listRes.get(i).getArrival_time());
        textTimeArrival.setTextSize(14);
        TextView(view.getContext());
        textCountPlaces.setText("Свободных мест: " +
listRes.get(i).getCount_free());
        textCountPlaces.setTextSize(14);
        textCountPlaces.setGravity(Gravity.CENTER);
        rowCountPlaces.addView(textCountPlaces);
        table.addView(rowCountPlaces);
        // Строка "цена билета"
        TableRow rowPrice = new TableRow(view.getContext());
        rowPrice.setGravity(Gravity.CENTER_HORIZONTAL);
        TextView textPrice = new TextView(view.getContext());

```

```

        textPrice.setText("Цена: " + listRes.get(i).getPrice() + "
руб.");
        textPrice.setTextSize(14);
        textPrice.setGravity(Gravity.CENTER);
        rowPrice.addView(textPrice);
        table.addView(rowPrice);
        // Строка кнопка "купить"
        final Button btnBuy = new Button(view.getContext());
        btnBuy.setText("Купить");
        btnBuy.setId(i);
        btnBuy.setGravity(Gravity.CENTER);
        btnBuy.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(view.getContext(),
BuyingActivity.class);
                intent.putExtra("trip_id",
listRes.get(btnBuy.getId()).getTrip_id());
                intent.putExtra("station_id", editSchedule.getText());
                intent.putExtra("date", selectedDate.showDate());
                intent.putExtra("departure_id",
listRes.get(btnBuy.getId()).getDeparture_id());
                intent.putExtra("destination_id",
listRes.get(btnBuy.getId()).getDestination_id());
                intent.putExtra("departure_time",
listRes.get(btnBuy.getId()).getDeparture_time());
                intent.putExtra("arrival_time",
listRes.get(btnBuy.getId()).getArrival_time());
                intent.putExtra("destination_date",
listRes.get(btnBuy.getId()).getDestination_date());
                startActivity(intent);
            }
        });
        rowBtnBuy.addView(btnBuy);
        table.addView(rowBtnBuy);
    }
}

```

Листинг Б.8 – Файл класса на языке Java «SearchActivity.java»

```

public class SearchActivity extends Fragment {
    FirebaseAuth mAuth;
    DatabaseReference myRef;
    FirebaseUser user;
    View view;
    Button btnSearch;
    EditText editDeparture;
    EditText editArrival;
    CalendarView calendarSearch;
    TableLayout tableSearch;
    String currentDeparture;
    String currentDestination;
    List<Trip> foundTrips;
    List<Trip> tempTrips;
    SelectedDate selectedDate;
    public SearchActivity() {

    }

    public static SearchActivity newInstance() {

```

```

        return new SearchActivity();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        view = inflater.inflate(R.layout.activity_search, container, false);
        tableSearch = view.findViewById(R.id.tableSearch);
        btnSearch = view.findViewById(R.id.btnSearch);
        editDeparture = view.findViewById(R.id.editDeparture);
        editArrival = view.findViewById(R.id.editArrival);
        calendarSearch = view.findViewById(R.id.calendarSearch);
        selectedDate = new SelectedDate();
        calendarSearch.setOnDateChangeListener(new
CalendarView.OnDateChangeListener() {
            @Override
            public void onSelectedDayChange(CalendarView view, int year,
            int month, int dayOfMonth) {
                selectedDate.setDay(dayOfMonth);
                selectedDate.setMonth(month + 1);
                selectedDate.setYear(year);
                Toast.makeText(view.getContext(), selectedDate.showDate(),
Toast.LENGTH_LONG).show();
            }
        });
        user = mAuth.getInstance().getCurrentUser();
        myRef = FirebaseDatabase.getInstance().getReference();
        foundTrips = new ArrayList<>();
        tempTrips = new ArrayList<>();
        btnSearch.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                currentDeparture = editDeparture.getText().toString();
                currentDestination = editArrival.getText().toString();

myRef.child("stations").child(currentDeparture).child(selectedDate.showDate()
).addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot
dataSnapshot) {
                List<Trip> listRes = new ArrayList<>();
                for (DataSnapshot dataValues :
dataSnapshot.getChildren()) {
                    Trip trip = dataValues.getValue(Trip.class);
                    listRes.add(trip);
                }
                searchTrip(listRes);
            }

            @Override
            public void onCancelled(@NonNull DatabaseError
databaseError) {

            }

        });
    }

    return view;
}

public void searchTrip(List<Trip> listRes) {

```

```

        foundTrips.clear();
        for (int i = 0; i < listRes.size(); i++) {
            if
(listRes.get(i).getDestination_id().equals(currentDestination))
                foundTrips.add(listRes.get(i));
        }
        if (foundTrips.size() == 0) {
            int temp = -1;
            Transfer transfer = new Transfer(currentDeparture,
currentDestination);
            if (transfer.buildRoute()) {
                if (transfer.transferStation.size() == 1) {
                    temp = 0;
                }
                for (int i = 0; i < listRes.size(); i++) {
                    if
(listRes.get(i).getDestination_id().equals(transfer.transferStation.get(temp)
))
                        foundTrips.add(listRes.get(i));
                    }
                //Находим маршрут <первый_пересадочный_пункт> ->
<пункт_прибытия>

myRef.child("stations").child(foundTrips.get(0).getDestination_id())
                .child(selectedDate.showDate())
                .addListenerForSingleValueEvent(new
ValueEventListener() {
                    @Override
                    public void onDataChange(@NonNull DataSnapshot
dataSnapshot) {
                        List<Trip> listRes = new ArrayList<>();
                        for (DataSnapshot dataValues :
dataSnapshot.getChildren()) {
                            Trip trip = dataValues.getValue(Trip.class);
                            listRes.add(trip);
                        }
                        getLastPoint(listRes);
                    }

                    @Override
                    public void onCancelled(@NonNull DatabaseError
databaseError) {

                    }

                });
            }
        }
        else
            drawTable();
    }

    public void getLastPoint(List<Trip> listRes) {
        for (int i = 0; i < listRes.size(); i++) {
            if
(listRes.get(i).getDestination_id().equals(currentDestination))
                foundTrips.add(listRes.get(i));
            }
            drawTable();
        }

    public void drawTable() {
        btnBuy.setOnClickListener(new View.OnClickListener() {

```

```

        @Override
        public void onClick(View v) {
            Intent intent = new Intent(view.getContext(),
BuyingActivity.class);
            intent.putExtra("trip_id",
foundTrips.get(btnBuy.getId()).getTrip_id());
            intent.putExtra("station_id",
editDeparture.getText().toString());
            intent.putExtra("date", selectedDate.showDate());
            intent.putExtra("departure_id",
foundTrips.get(btnBuy.getId()).getDeparture_id());
            intent.putExtra("destination_id",
foundTrips.get(btnBuy.getId()).getDestination_id());
            intent.putExtra("departure_time",
foundTrips.get(btnBuy.getId()).getDeparture_time());
            intent.putExtra("arrival_time",
foundTrips.get(btnBuy.getId()).getArrival_time());
            intent.putExtra("destination_date",
foundTrips.get(btnBuy.getId()).getDestination_date());
            startActivity(intent);
        }
    });
    rowBtnBuy.addView(btnBuy);
    tableSearch.addView(rowBtnBuy);
}
}
}

```

Приложение В
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
(рекомендуемое)

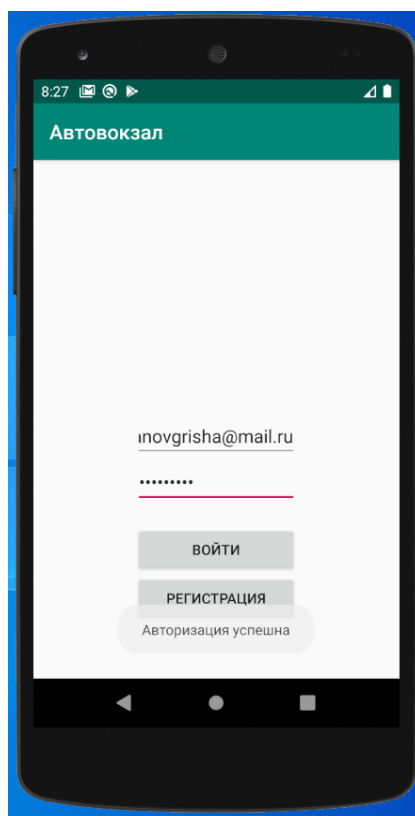


Рисунок В.1 – Тестирование методом определения классов эквивалентности, тест 1

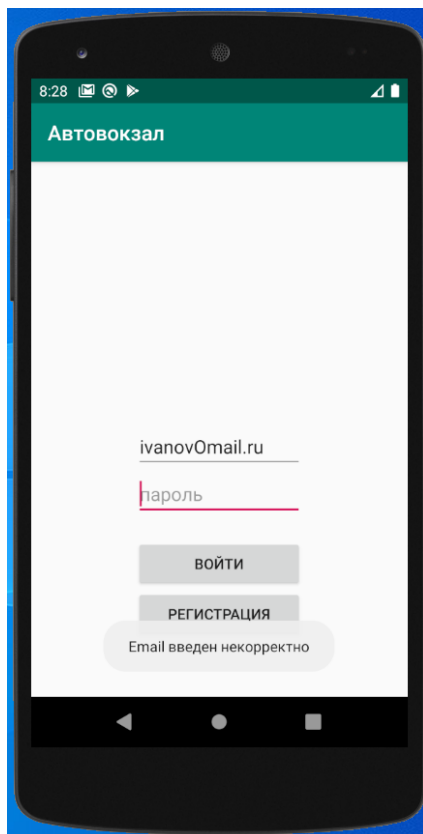


Рисунок В.2 – Тестирование методом определения классов эквивалентности, тест 2

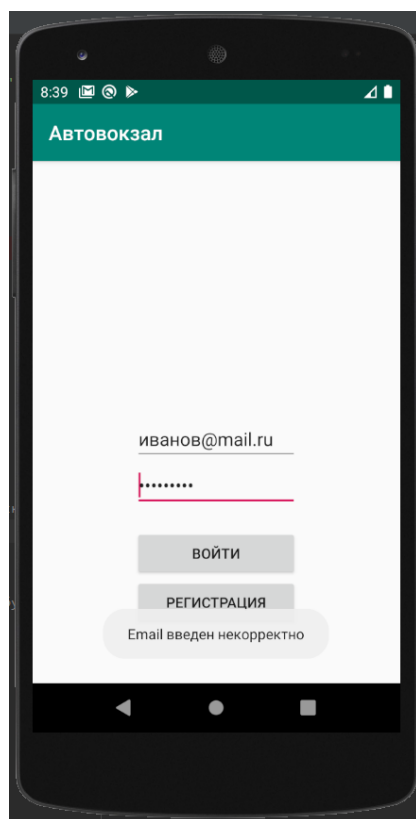


Рисунок В.3 – Тестирование методом определения классов эквивалентности, тест 3

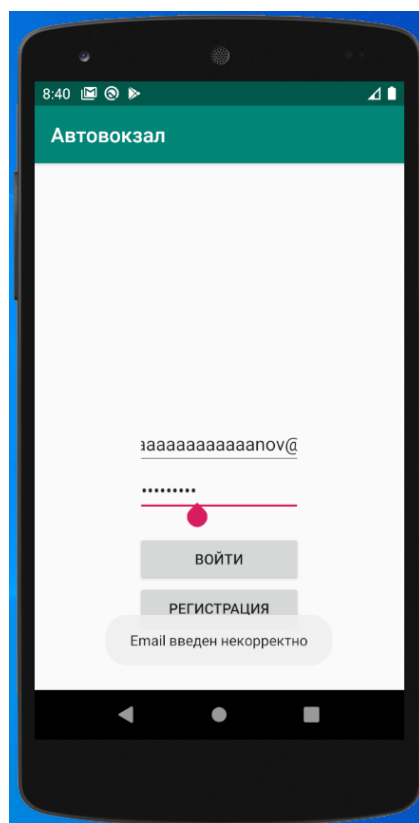


Рисунок В.4 – Тестирование методом определения классов эквивалентности, тест 4

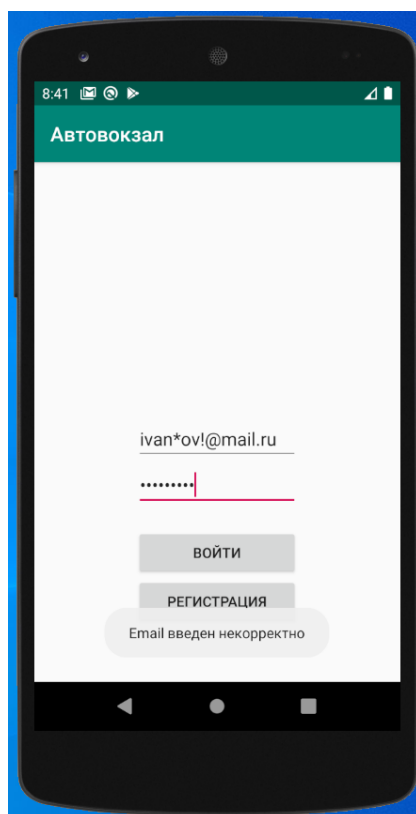


Рисунок В.5 – Тестирование методом определения классов эквивалентности, тест 5

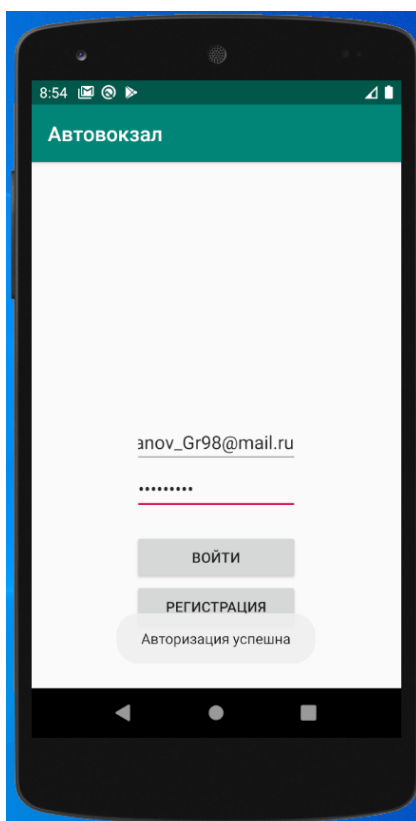


Рисунок В.6 – Тестирование методом определения классов эквивалентности, тест 6

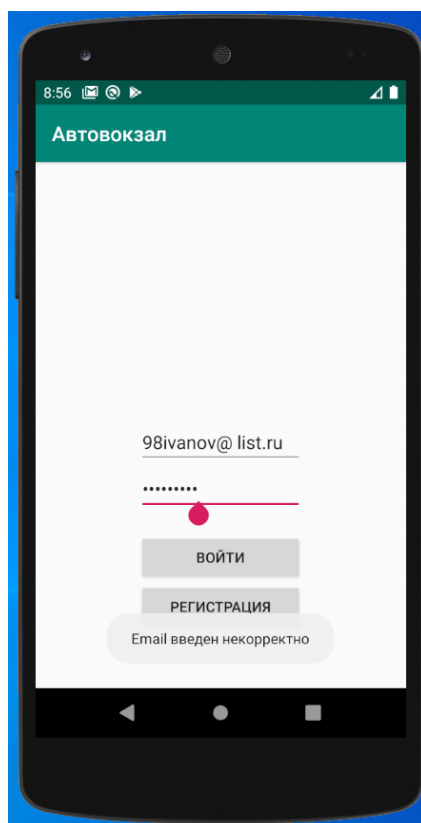


Рисунок В.7 – Тестирование методом определения классов эквивалентности, тест 7

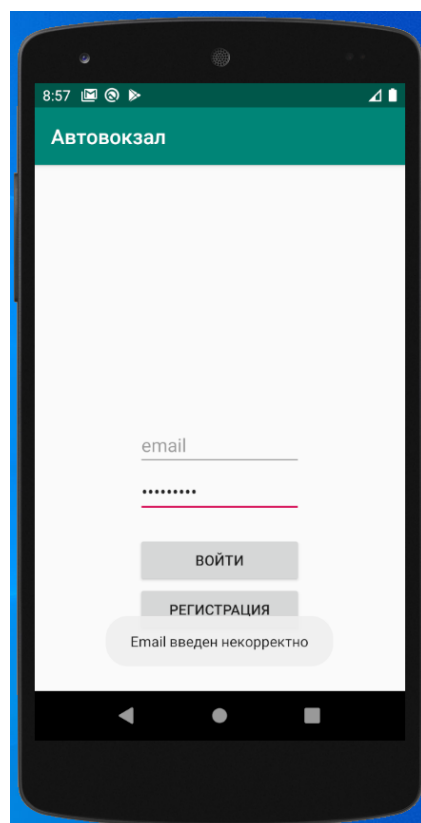


Рисунок В.8 – Тестирование методом анализа граничных условий, тест 1

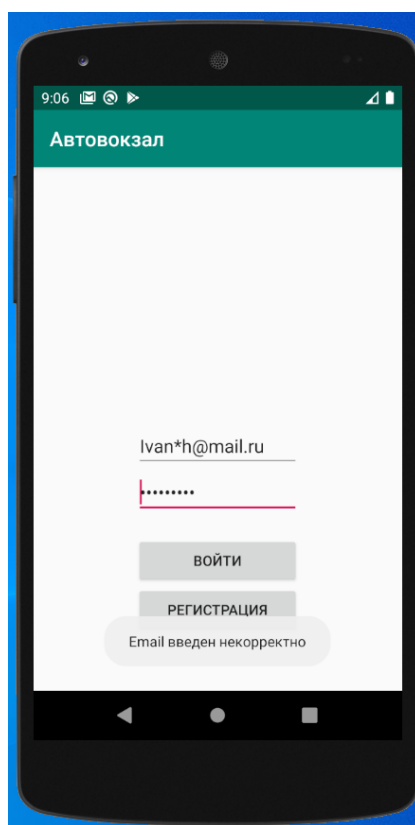


Рисунок В.9 – Тестирование методом анализа граничных условий, тест 2

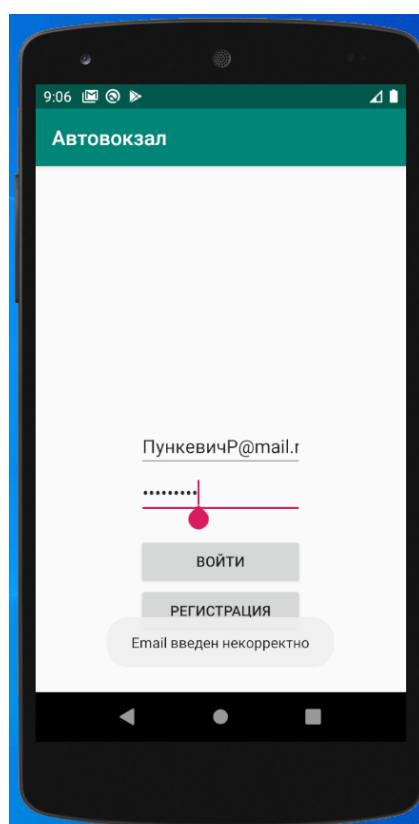


Рисунок В.10 – Тестирование методом анализа граничных условий, тест 3

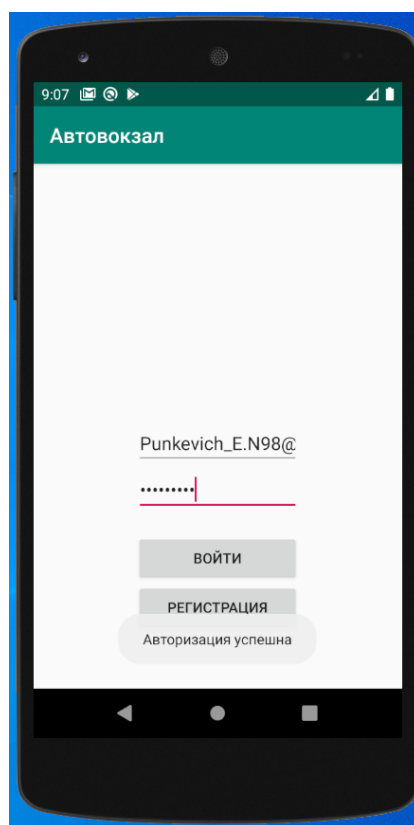


Рисунок В.11 – Тестирование методом анализа граничных условий, тест 4

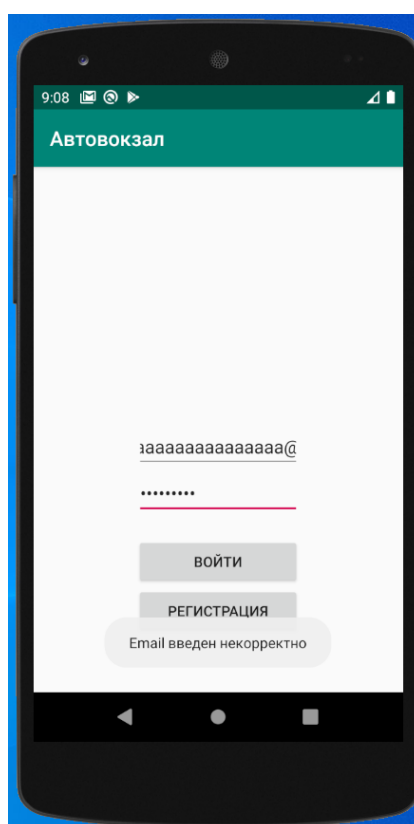


Рисунок В.12 – Тестирование методом анализа граничных условий, тест 5

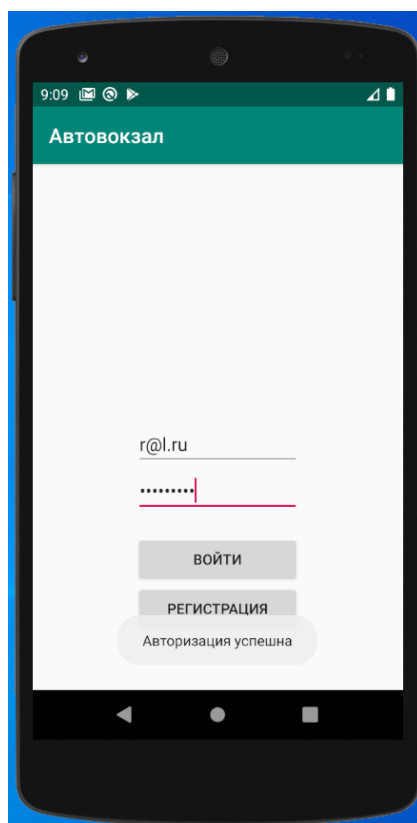


Рисунок В.13 – Тестирование методом анализа граничных условий, тест 6

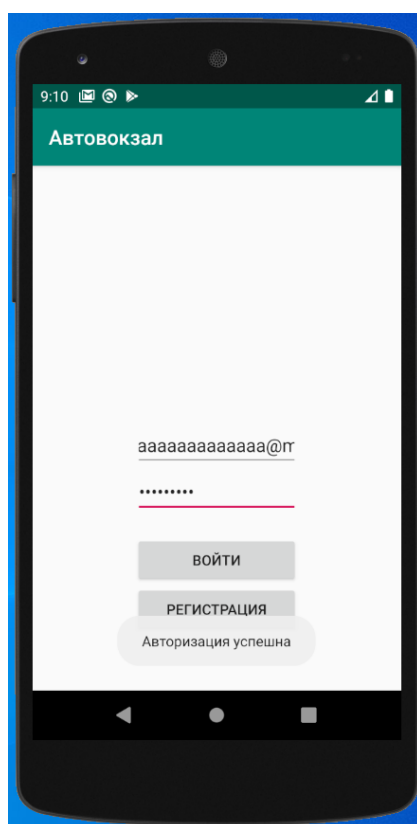


Рисунок В.14 – Тестирование методом анализа граничных условий, тест 7

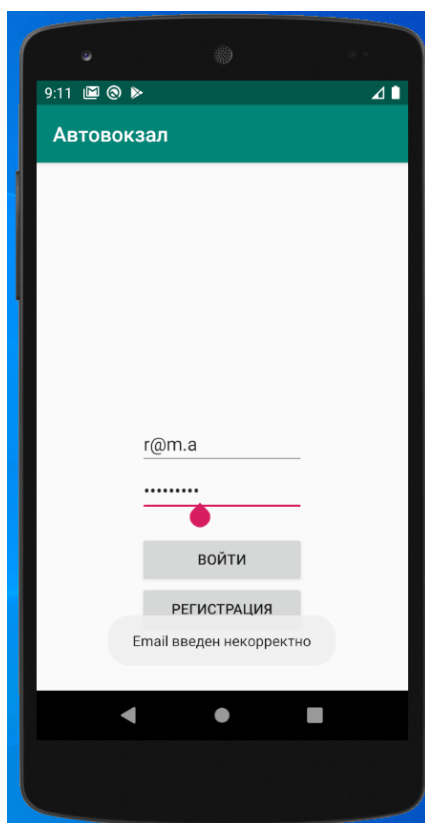


Рисунок В.15 – Тестирование методом анализа граничных условий, тест 8

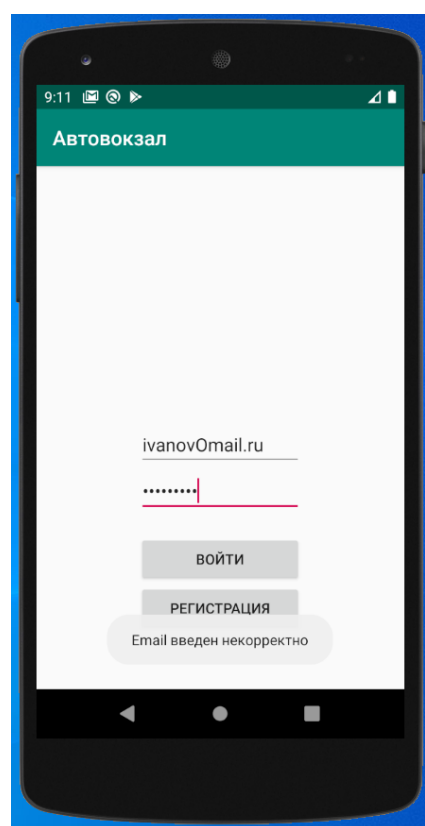


Рисунок В.16 – Тестирование методом анализа граничных условий, тест 9

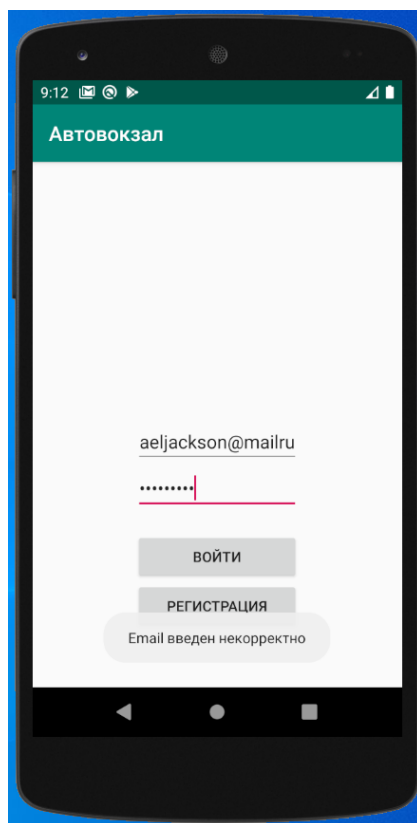


Рисунок В.17 – Тестирование методом анализа граничных условий, тест 10