МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Факультет *компьютерных наук*

Кафедра *программирования и информационных технологий*

Приложение по подбору авиабилетов

Курсовой проект

09.03.02 *Информационные системы и технологии*

Программная инженерия в информационных системах

Допущен к защите

Зав. Кафедрой _____*С.Д. Махортов, к.ф.- м.н., доцент* __.__.20__

Обучающийся _____*А.Е. Копылова 3 курс, д/о*

Обучающийся _____*О.А. Транина 3 курс, д/о*

Руководитель _____ *Х.А. Полещук, аспирант*

Воронеж 2019

Содержание

Введение

В современном мире мы наблюдаем проникновение информационных технологий практически во все сферы человеческой деятельности, не стала исключением и сфера предоставления услуг по поиску авиабилетов.

Учитывая темп жизни современного человека, время является важнейшим ресурсом. Не удивительно, что с распространением интернета приобрели высокую популярность различные сервисы онлайн-поиска для мобильных устройств. Ведь они позволяют существенно упростить и ускорить процесс получения информации о различных услугах. Очевидно, что использование программного приложения для предоставления информации о перелетах и ценах дает преимущества такие, как:

- экономия времени
- более удобная форма поиска с широким выбором фильтров
- более разнообразные предложения

При разработке проекта используется каскадная модель, которая включает в себя следующие этапы:

1. Определение требований
2. Проектирование
3. Конструирование (также «реализация» либо «кодирование»)
4. Воплощение
5. Тестирование и отладка (также «верификация»)
6. Инсталляция
7. Поддержка

Также в процессе разработки будет использоваться парадигма объектно-ориентированного программирования.

1. Постановка задачи

Целью курсового проекта является создание приложения, работающего на операционной системе Android.

Основную функциональность разрабатываемого приложения отражает диаграмма прецедентов, изображенная на рисунке 1.

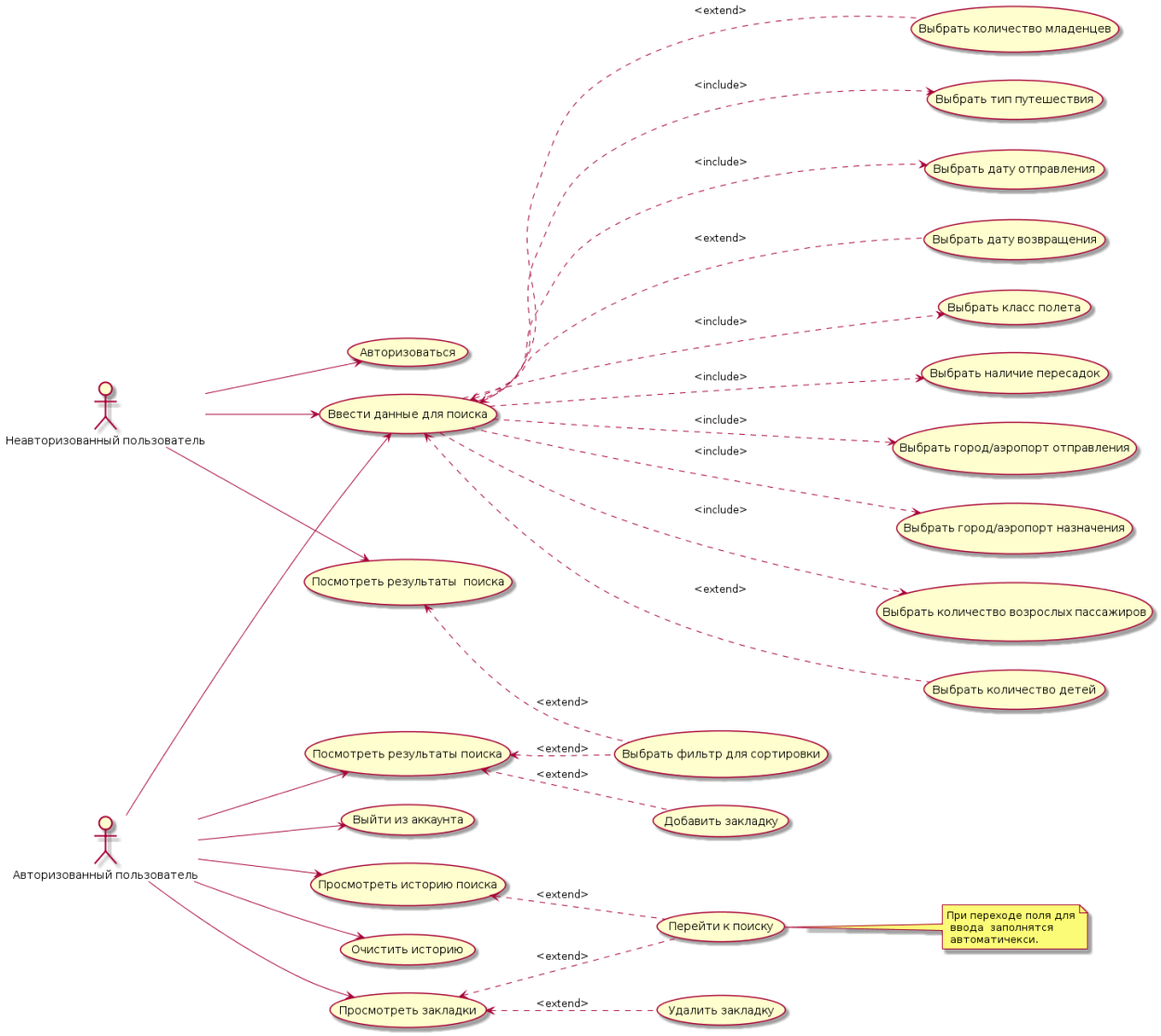

Рисунок 1 - Диаграмма прецедентов

Неавторизованный пользователь обладает следующими возможностями:

— авторизация;

— поиск авиабилетов на рейсы в один конец с возможностью добавления информации о количестве пассажиров, типе путешествия (в один или оба конца), класса полета (эконом или бизнес), наличии пересадок;
— поиск авиабилетов на рейсы в оба конца с возможностью добавления информации о количестве пассажиров, типе путешествия (в один или оба конца), (эконом или бизнес), наличии пересадок;
— сортировка результатов поиска.

Авторизованный пользователь в дополнение к возможностям неавторизованного пользователя обладает следующими возможностями:

— добавление маршрута в закладки;
— просмотр закладок;
— удаление закладки;
— просмотр истории поиска;
— удаление элемента истории поиска;
— очищение истории поиска;
— возможность не сохранять историю поиска**;**
— переход со страницы закладок и истории поиска к поиску авиабилетов с заполнением данных поиска;
— выход из аккаунта.

Система должна соответствовать следующим требованиям:

1. Использование нескольких источников данных.
2. Подбор не только прямых рейсов, но и рейсов с пересадками.
3. Реализовать все возможности пользователей, описанные выше.

Завершенный проект представляет собой полностью функционирующее Android-приложение, соответствующее требованиям, описанным выше.

2. Анализ

2.1. Анализ предметной области

Для поиска авиабилетов минимально необходимой информацией являются следующие данные:

- пункт отправления;
- пункт назначения;
- количество взрослых пассажиров;
- количество детей;
- количество младенцев;
- дата отправления;
- тип салона (бизнес-класс или эконом).

Однако часто билеты покупают с обратной дорогой или, как это еще называют, в обе стороны. В этом случае к уже перечисленным данным добавляются данные об обратном билете, в частности дата отправления из пункта назначения.

Также при поиске рейсов из пункта назначения в пункт прибытия необходимо учитывать не только прямые рейсы, но и рейсы с пересадками, то есть такие, у которых есть промежуточные точки остановок в пути. Однако не для всех пассажиров такие рейсы являются приемлемыми, так как они существенно снижают комфорт путешествия, добавляя к времени в дороге время на саму пересадку. Для таких случаев следует предоставить возможность пользователю приложения убрать такие рейсы из итоговых результатов поиска.

По причине того, что результатом поиска, как правило, является длинный список разнообразных вариантов рейсов в пункт назначения, пользователю следует предоставить возможность сортировки результатов по следующим критериям:

— по убыванию цены;
— по возрастанию цены;
— по времени в дороге (от самого долгого путешествия к самому короткому);
— по времени в дороге (от самого короткого путешествия к самому долгому);
— по количеству пересадок (от большего к меньшему);
— по количеству пересадок (от меньшего к большему).

2.2. Сравнение с аналогами

В области поиска авиабилетов существует большое количество мобильных приложений, работающих на операционной системе Android. Рассмотрим наиболее крупные из них:

— Tutu.ru

Туту.ру (tutu.ru) – российский онлайн сервис путешествий, предоставляющий услуги по поиску и покупки билетов на все виды транспорта, а именно ж/д и авиа перевозки и автобусы. Также предоставляет необходимую информацию путешественникам, например, телефоны авиакомпаний и вокзалов, проезды в аэропорты. Минусом данного приложения является невозможность добавить маршрут или билет в закладки.

— Аэрофлот

Аэрофлот – официальное мобильное приложение ПАО Аэрофлот, предоставляющее возможность поиска, бронирование и покупки авиабилетов. Также предлагает некоторые дополнительные сервисы, такие как регистрация на рейс, онлайн табло и статус рейса, расписание

регулярных рейсов. Однако данное приложение предоставляет информацию только о билетах авиакомпании «Аэрофлот».

— Tickets.ru

Tickets.ru – российский онлайн сервис путешествий, предоставляющий услуги по поиску и покупки билетов регулярных авиакомпаний с наиболее актуальными ценами. Недостатком данного приложения является отсутствие возможности сохранить информацию о маршруте и билете для ускорения дальнейшего использования.

2.3. Анализ задач

2.3.1. Задача поиска авиабилетов

Рассмотрим задачу поиска авиабилетов. Данная задача включает в себя следующие этапы:

1. Проверка данных, введенных пользователем. Если данные введены не корректно, то пользователю предлагают ввести их повторно.
2. Проверка существования введенных городов. Если города не существуют, то пользователю выводиться сообщение о том что города не найдены.
3. Отправка данных, введенных на форме, на сервер.
4. Формирование и отправление запроса с данными на сторонний ресурс (API сервер) сервером.
5. Обработка сервером ответа, если таковой получен. Если ответ не получен или произошла какая-либо ошибка, то пользователю выводится сообщение об этом и предлагается ввести данные для поиска повторно.
6. Отображение итоговых результатов поиска пользователю.

2.3.2. Задача хранения данных пользователя и приложения

Рассмотрим задачу хранения данных пользователя, приложения и его настроек. Данную задачу можно подразделить на две подзадачи, а именно:

— хранение закладок;
— хранение истории поиска.

2.3.2.1. Подзадача хранения закладок

Рассмотрим подзадачу хранения закладок. Закладка содержит в себе следующую информацию:

— пункт отправления;
— пункт назначения;
— количество взрослых пассажиров;
— количество детей;
— количество младенцев;
— наличие пересадок;
— тип путешествия (в один конец или с обратной дорогой);
— класс обслуживания (эконом или бизнес).

Для выполнения данной подзадачи следует реализовать следующие возможности пользователя:

— Сохранение выбранного маршрута в закладки.
— Просмотр всех закладок.
— Удаление закладки.
— Переход с закладки к форме поиска с заполненными полями, соответствующими информации закладки.

2.3.2.2. Подзадача хранения истории поиска

Рассмотрим подзадачу хранения истории поиска. Отдельным элементом истории поиска является совокупность информации, введенной

пользователем для поиска рейсов. Это значит, что в данном элементе содержится информация о следующем:

- пункт отправления,
- пункт назначения,
- количество взрослых пассажиров,
- количество детей,
- количество младенцев,
- дата отправления.
- дата отправления из пункта назначения (если билеты в оба конца)
- наличие пересадок
- тип путешествия (в один конец или с обратной дорогой);
- класс обслуживания (эконом или бизнес).

Также данная подзадача включает в себя задачи по реализации возможности пользователя:

- просматривать историю поиска;
- удалять отдельный элемент истории поиска;
- очистить всю историю поиска;
- переход с элемента истории поиска к форме поиска с заполненными полями, соответствующими информации закладки.

2.3.3. Задача отображения пользовательского интерфейса

Рассмотрим задачу отображения пользовательского интерфейса. Данная задача включает в себя разработку и оформление следующих интерфейсов и страниц:

- форма для ввода данных для поиска;
- страница, содержащая результаты поиска с возможностями добавить маршрут в закладки и отсортировать результаты;

— страница, содержащая информацию о конкретной единице результата поиска;
— страница, содержащая список закладок пользователя с возможностью удаления закладки и перехода по клику на закладку к форме ввода данных с передачей данных закладки;
— страница, содержащая историю поиска с возможностью ее очищения перехода по клику на элемент истории поиска к форме ввода данных с передачей данных элемента истории поиска (здесь же находиться элемент пользовательского интерфейса, позволяющий не сохранять историю поиска).

2.4. Анализ средств реализации

В качестве средств реализации системы поиска авиабилетов были выбраны следующие технологии:

1. Android SDK – средство разработки мобильных приложений для операционной системы Android. Чертой, отличающей от других средств разработки, является наличие широких функциональных возможностей, позволяющих запускать тестирование и отладку исходных кодов, оценивать работу приложения в режиме совместимости с различными версиями ОС Android.
2. В качестве СУБД была выбрана PostgreSQL, так как оно является наиболее популярным в данный момент и регулярно обновляемым.
3. Spring Boot – мощнейший фреймворк, для которого существует большое количество стандартных библиотек и решений, что сильно упрощает процесс разработки и позволяет закончить проект в сжатые сроки.
4. Для реализации Android-приложения был выбран высокоуровневый язык программирования Java, так как он является проверенным временем выбором. Существует большое количество различных

материалов по написанию Android-приложения на Java и большое количество обучающих статей.

5. Для облегчения документации предоставляемых API использован SWAGGER.

Для реализации отдельных подзадач были выбраны следующие библиотеки и средства:

- Протокол прикладного уровня HTTPS и библиотека Retrofit 2 Android для взаимодействия с внешними API серверами.
- Библиотека Glide для асинхронной загрузки изображений из сети интернет.
- Фреймворки JUnit и Espresso для тестирования.

2.5. Графическое описание работы системы

Для удобства описания работы системы была использована графическая нотация UML. В данном разделе представлены диаграммы и описания, где они требуются. Описана работа системы для авторизованного пользователя (для неавторизованного работа аналогична за исключением ограничений, описанных в постановке задачи).

2.5.1. Диаграммы состояний

Для описания состояний, в которых находится сервер при основном сценарии (поиске билетов, просмотра результатов и добавления маршрута в закладки) составлена диаграмма, изображенная на рисунке 2.

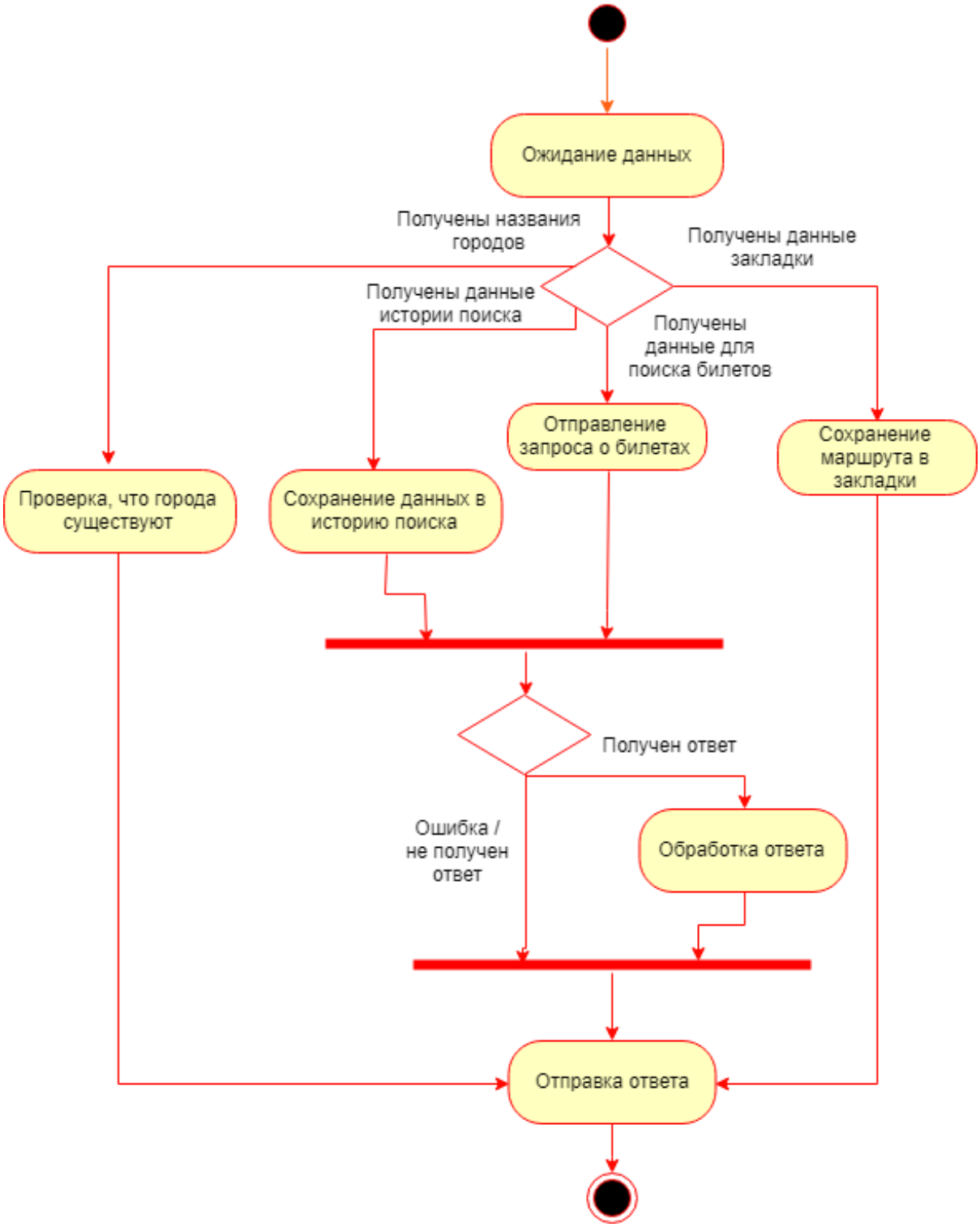

Рисунок 2 – Диаграмма состояний сервера основного сценария

Для описания состояний, в которых находится приложение при основном сценарии, составлена диаграмма, изображенная на рисунке 3.

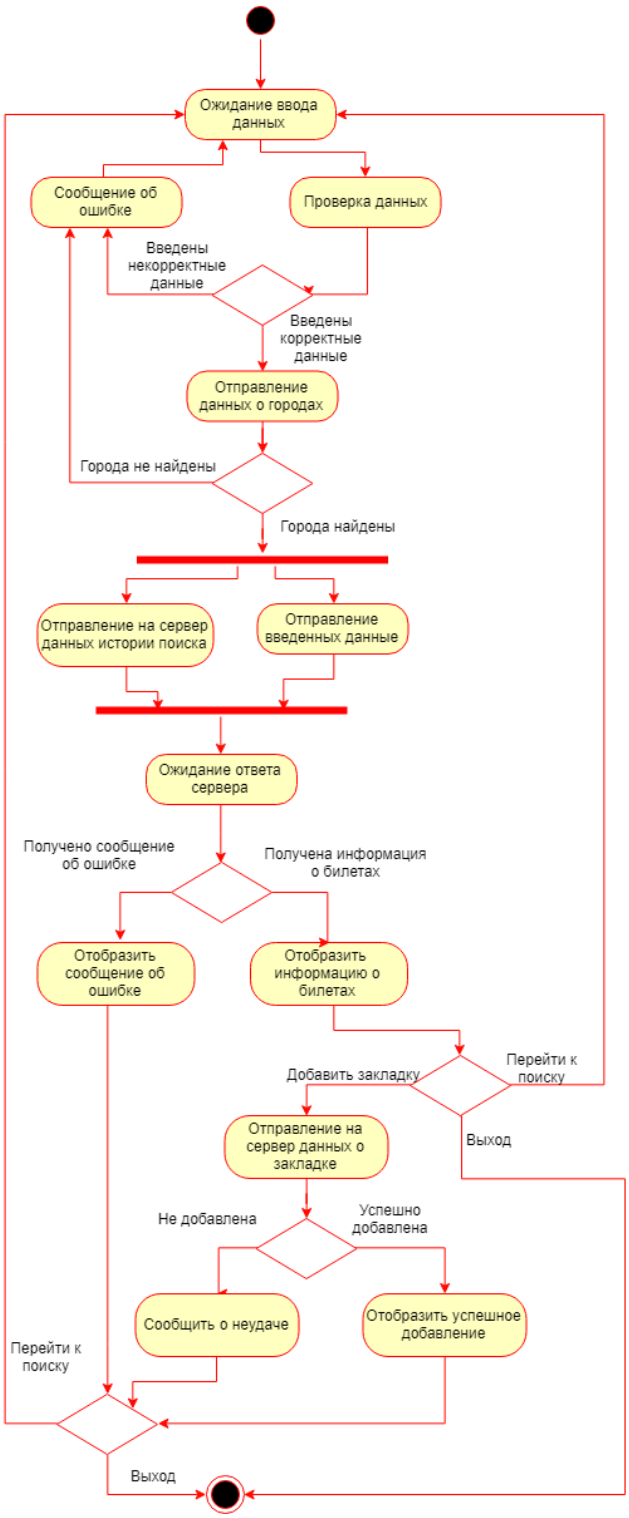

Рисунок 3 – диаграмма состояний приложения при основном сценарии

Диаграмма состояний пользователя при основном сценарии представлена в приложении А.

Диаграмма состояний приложения при взаимодействии пользователя с закладками представлена на рисунке 4.

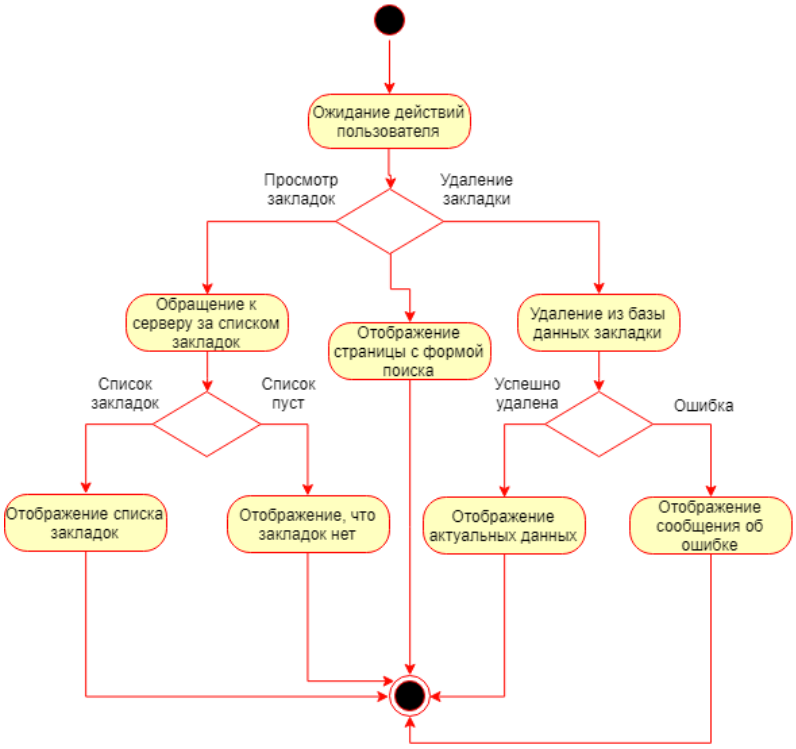

Рисунок 4 – Диаграмма состояний приложения (закладки)

Диаграмма состояний приложения при взаимодействии пользователя с историей поиска изображена на рисунке 5.

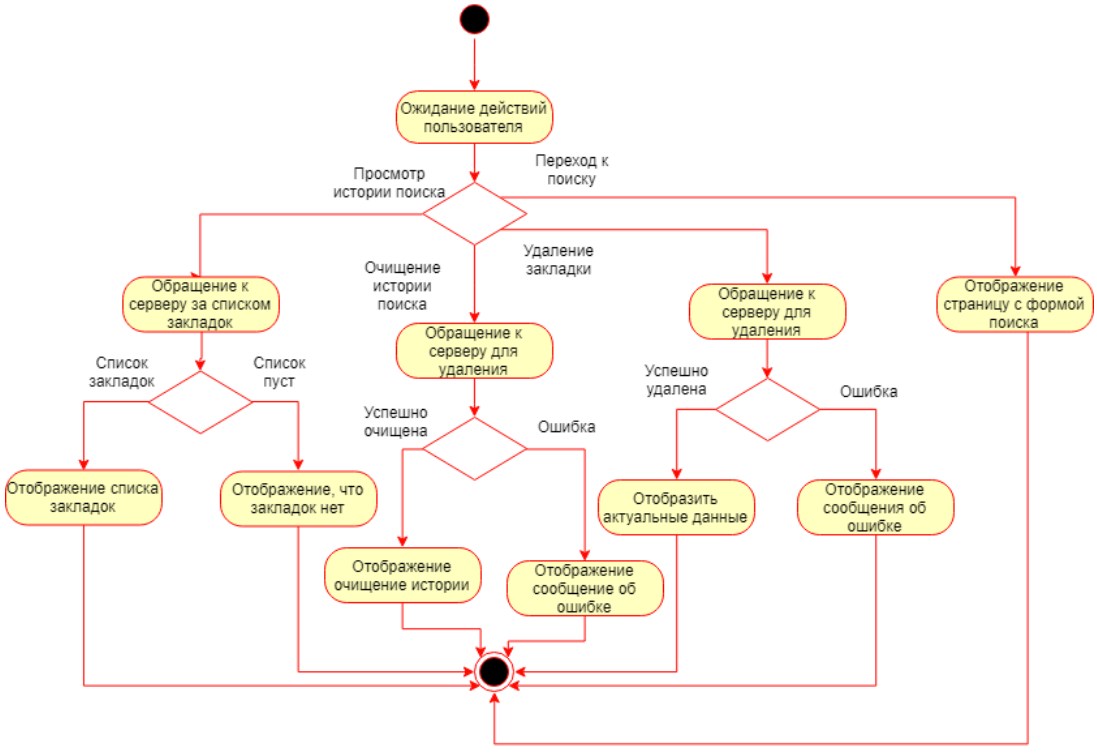

Рисунок 5 – Диаграмма состояний приложения (история поиска)

2.5.2. Диаграммы активности

Диаграммы активности являются расширениями диаграмм состояний, находящихся в предыдущем разделе.

Диаграмма активности основного сценария изображена на рисунке 6. На данной диаграмме присутствуют 4 части (дорожки): пользователь, приложение, сервер и сторонние источники (API).

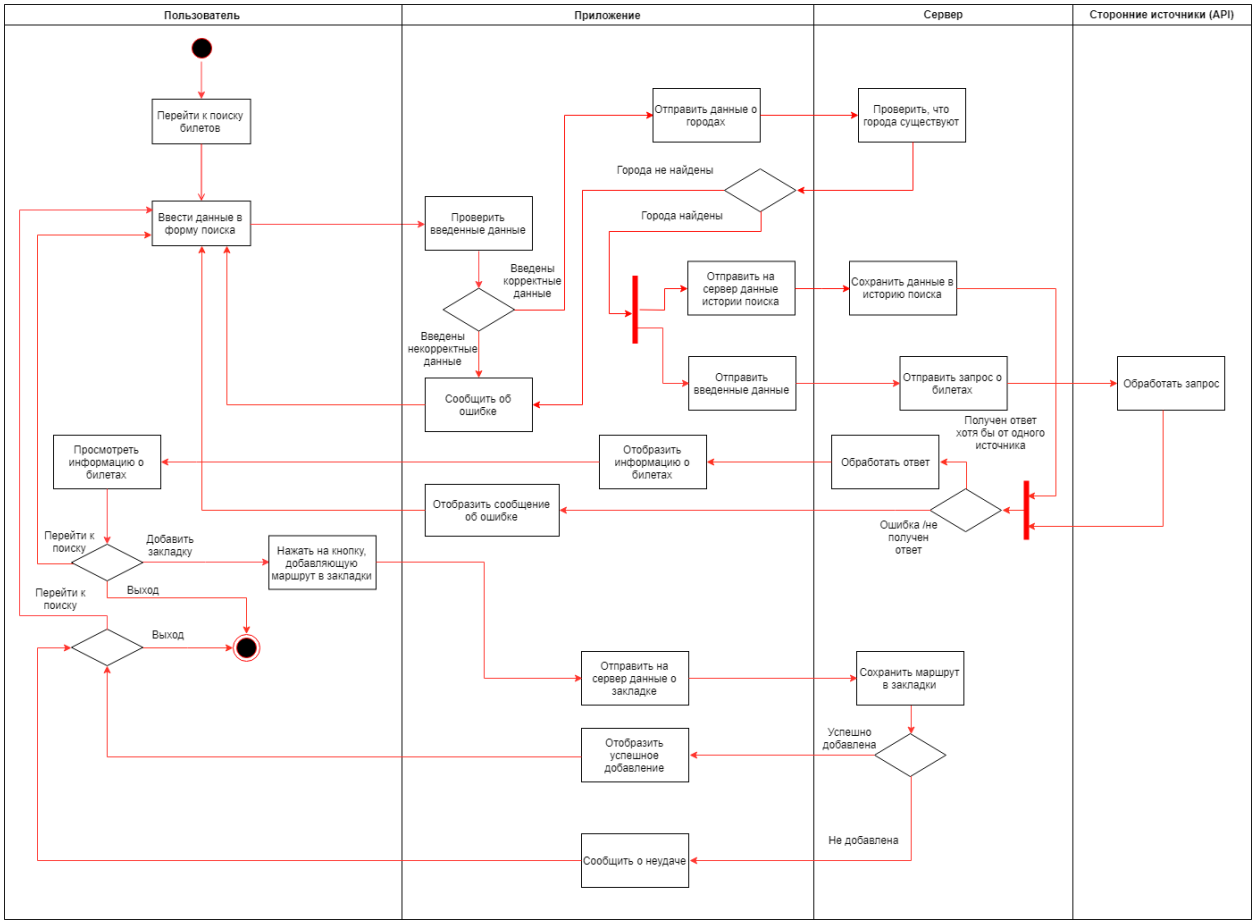

Рисунок 6 – Диаграмма активности основного сценария

Диаграмма активности взаимодействия пользователя с закладками изображена на рисунке 7. На данной диаграмме присутствуют 3 части (дорожки): пользователь, приложение и сервер.

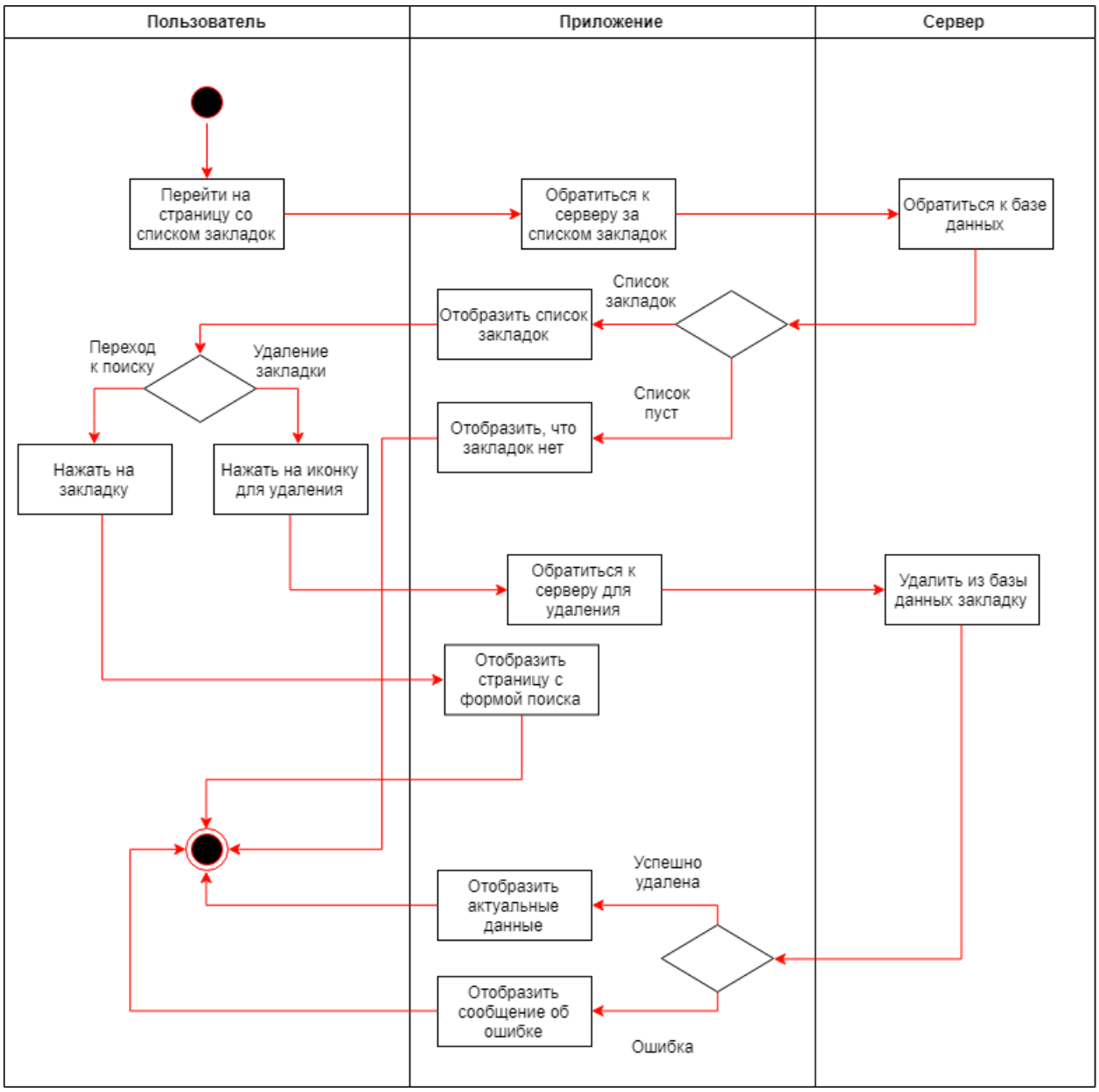

Рисунок 7 – Диаграмма активности (закладки)

Диаграмма активности взаимодействия пользователя с историей поиска изображена на рисунке 8. На данной диаграмме присутствуют 3 части (дорожки): пользователь, приложение и сервер.

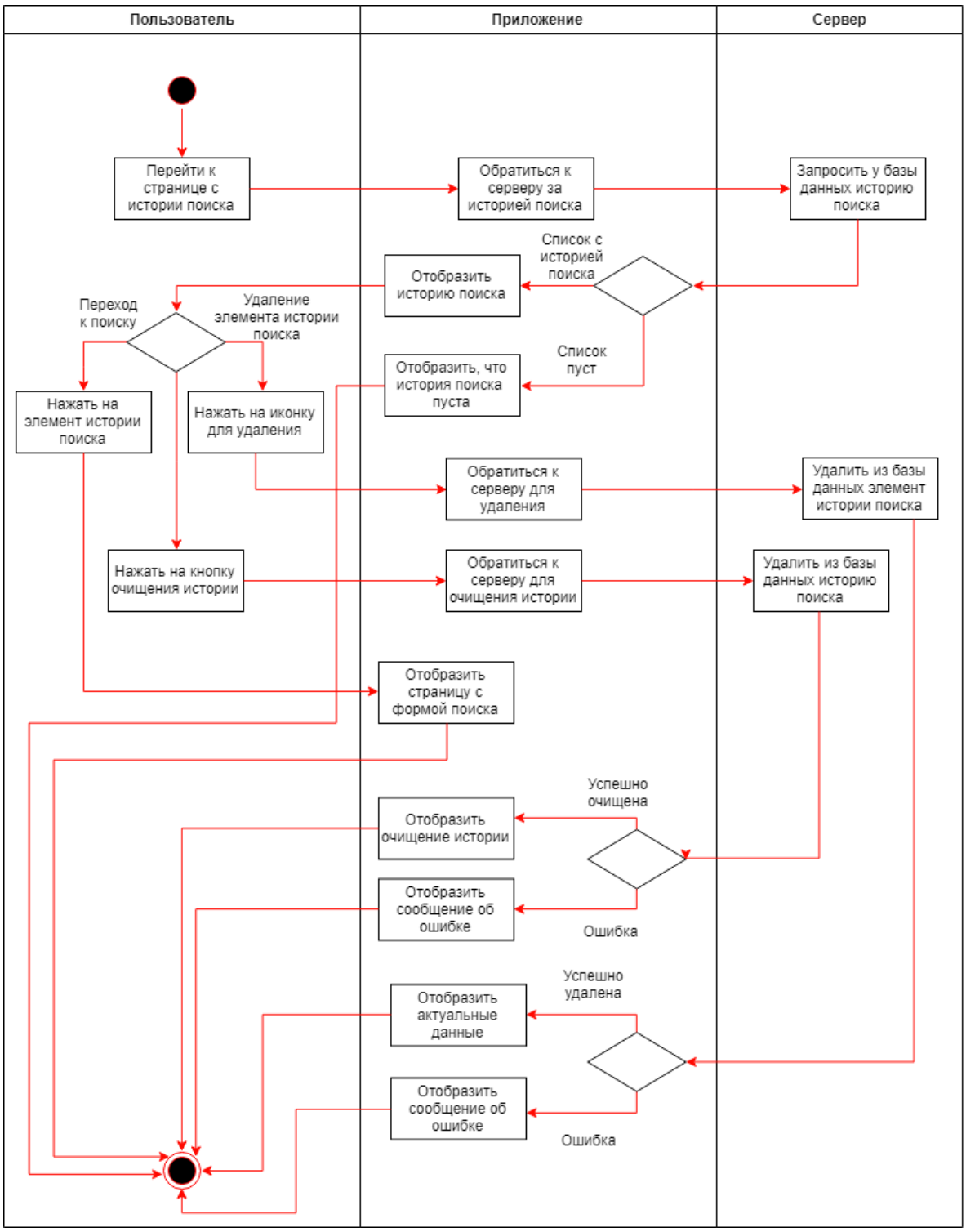

Рисунок 8 – Диаграмма активности (история поиска)

2.5.3. Диаграммы последовательностей

Диаграммы последовательностей основного сценария изображены на рисунках 9 (поиск билетов и отображение результатов) и 10 (добавление маршрута в закладки).

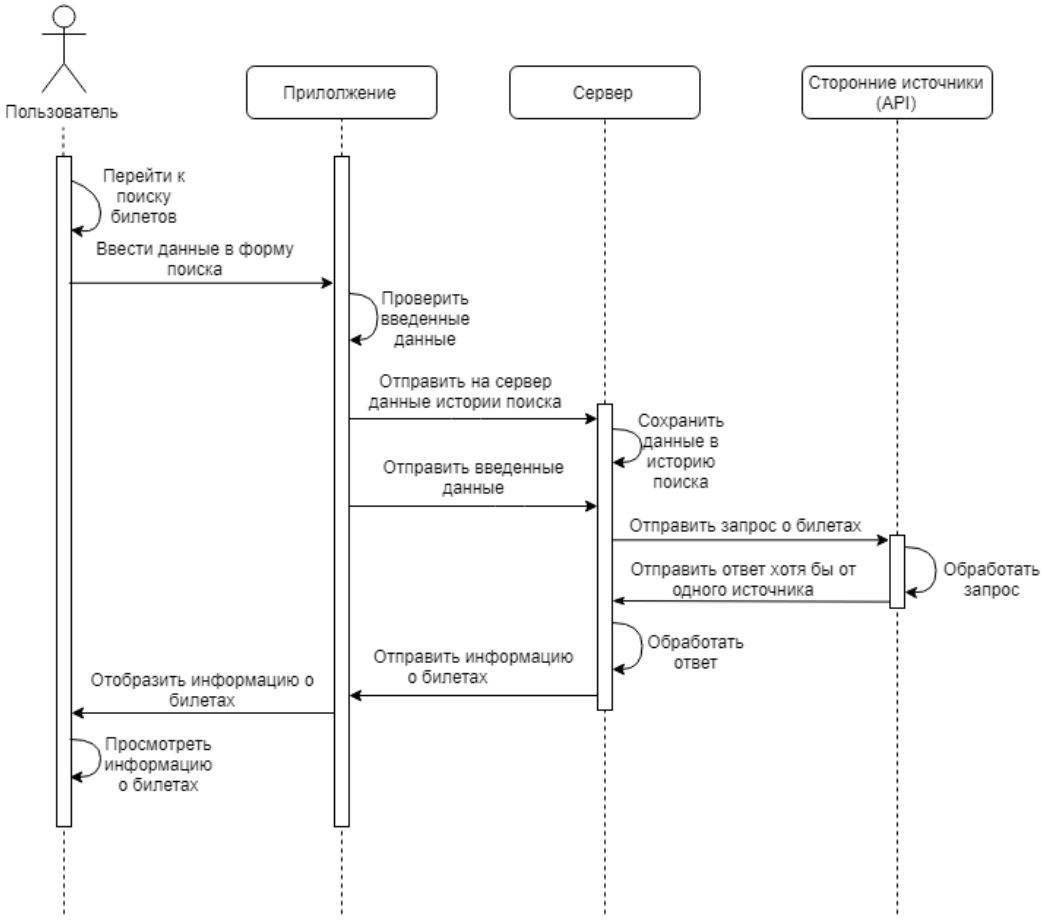

Рисунок 9 – Диаграмма последовательностей основного сценария (поиска билетов и отображение результатов)

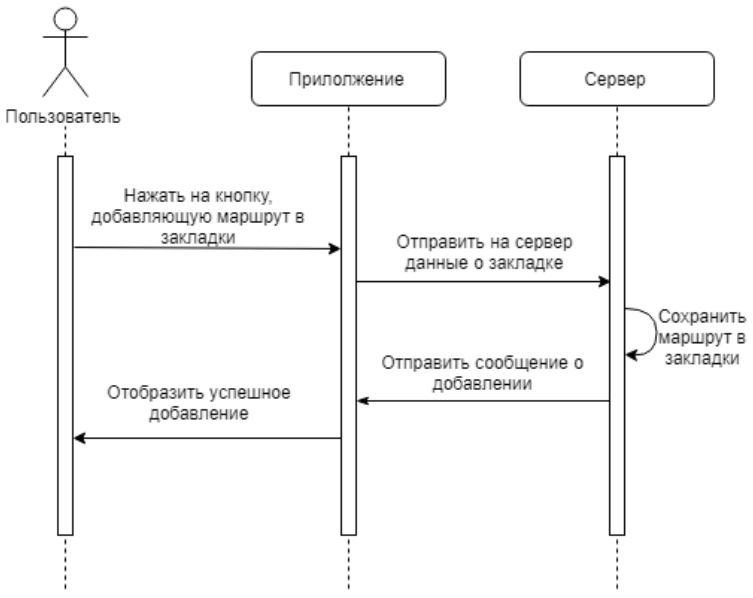

Рисунок 10 – Диаграмма последовательностей основного сценария (добавление маршрута в закладки)

Диаграммы последовательностей взаимодействия пользователя с закладками изображены на рисунках 11 (переход на форму поиска) и 12 (удаление закладки).

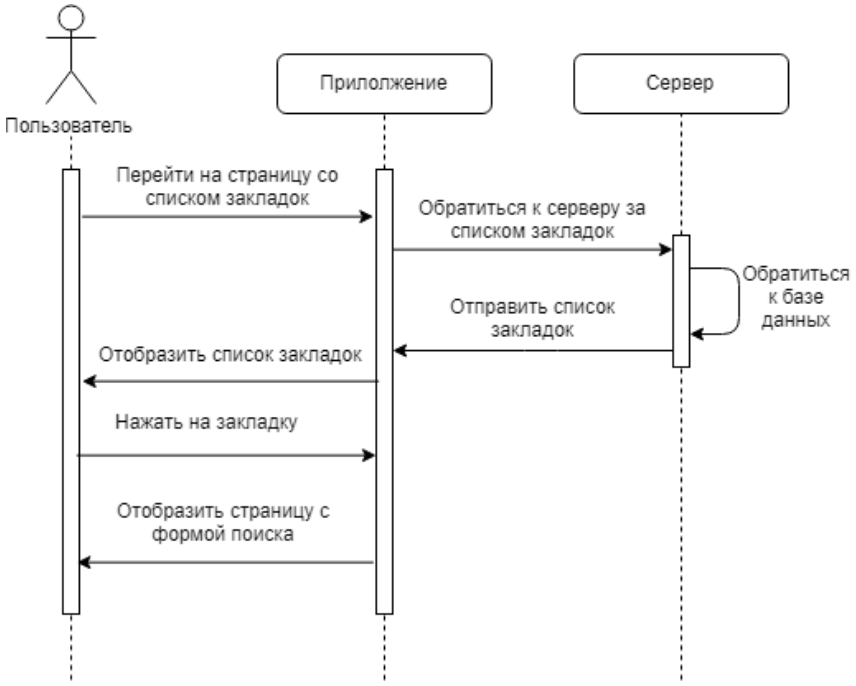

Рисунок 11 – Диаграмма взаимодействия (переход на форму поиска)

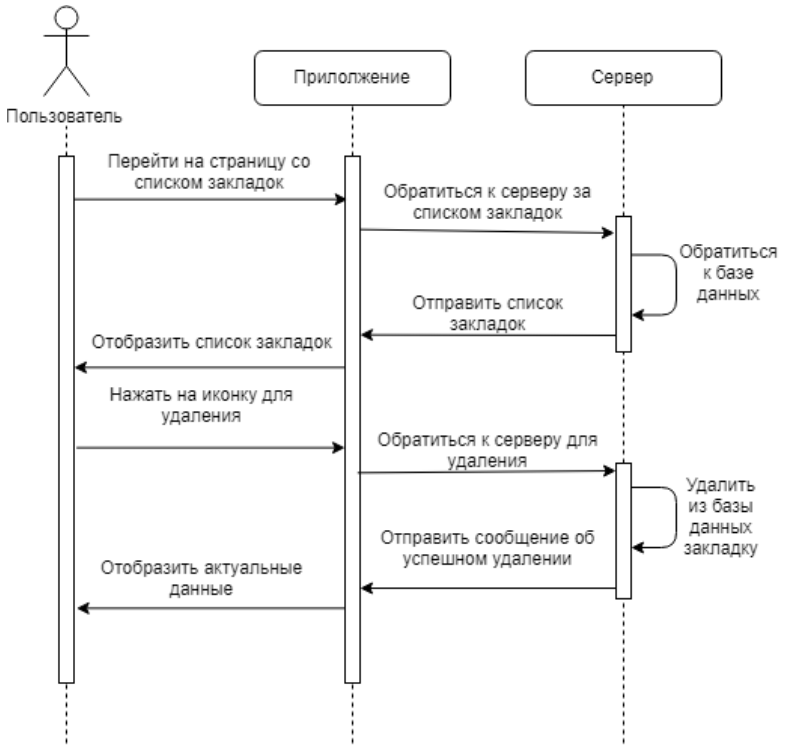

Рисунок 12 – Диаграмма взаимодействия (удаление закладки)

Диаграммы последовательностей взаимодействия пользователя с историей поиска изображены на рисунках 13 (переход к поиску), 14 (удаление элемента истории поиска) и 15 (очищение истории поиска).

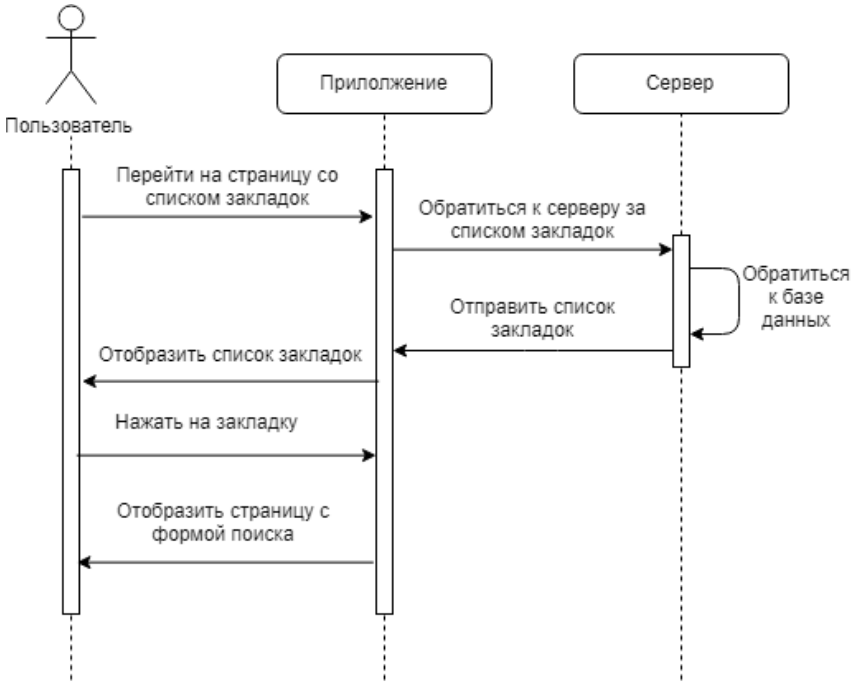

Рисунок 13 – Диаграмма последовательностей (переход к поиску)

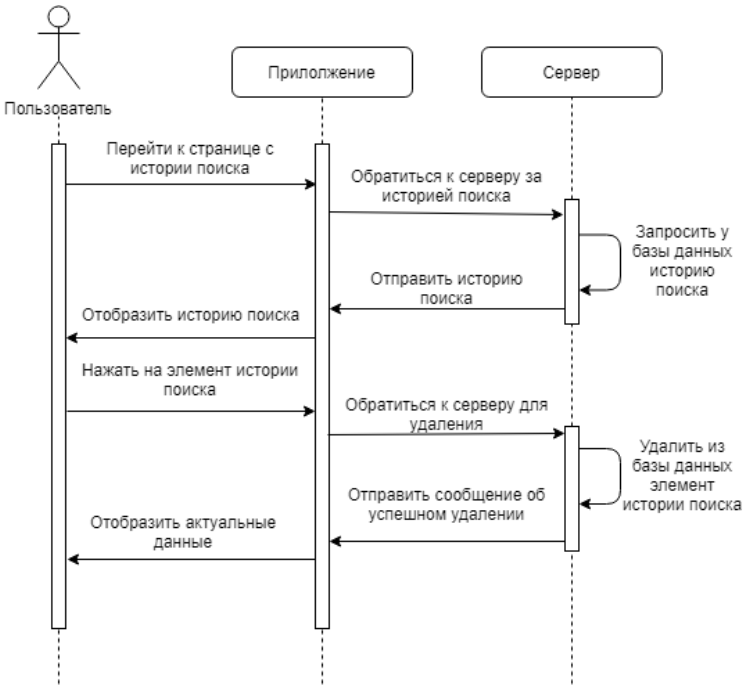

Рисунок 14 – Диаграмма последовательностей (удаление элемента истории поиска)

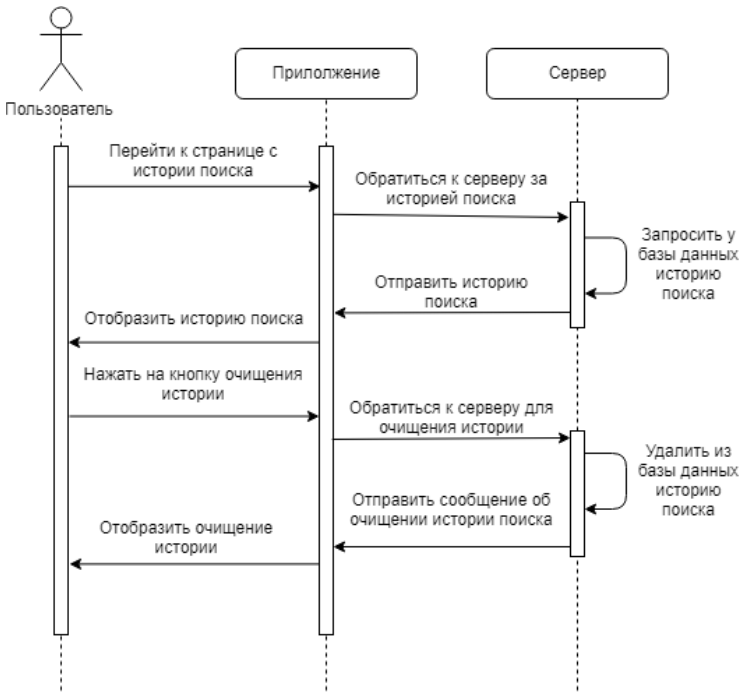

Рисунок 15 – Диаграмма последовательностей (очищение истории поиска)

2.5.4. Диаграммы коммуникаций

В данном разделе приведены диаграммы взаимодействия между компонентами системы.

Диаграмма коммуникаций поиска билетов и отображения результатов пользователю изображена на рисунке 16.

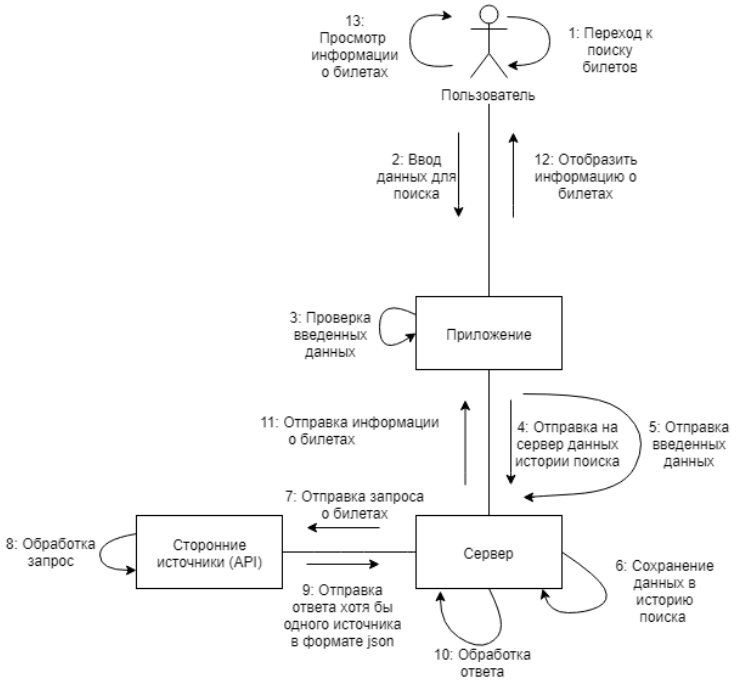

Рисунок 16 – Диаграмма коммуникаций подбора билетов

Диаграмма коммуникаций добавления маршрута в закладки представлена на рисунке 17.

Рисунок 17 – Диаграмма коммуникаций добавления в закладки

Диаграммы коммуникаций удаления элемента истории поиска, перехода на форму поиска и очищения истории поиска на рисунках 18, 19 и 20 соответственно.

Рисунок 18 – Диаграмма коммуникаций удаления элемента истории

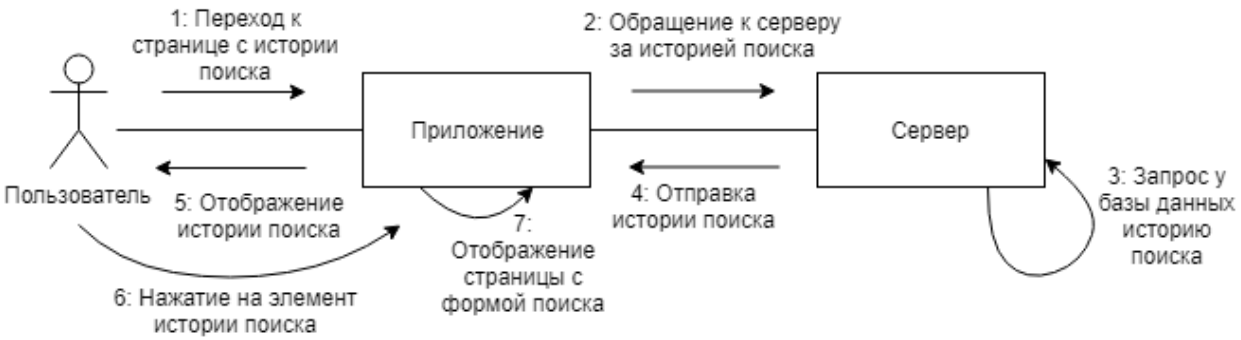

Рисунок 19 – Диаграмма коммуникаций перехода на форму поиска

Рисунок 20 – Диаграмма коммуникаций очищения истории

Диаграмма коммуникаций удаления закладки выглядит аналогично изображенной на рисунке 18, а перехода на форму поиска аналогично рисунку 19.

2.5.5. Диаграмма развертывания

Приведенная на рисунке 21 диаграмма визуализирует элементы и компоненты программы, которые существуют на этапе ее исполнения.

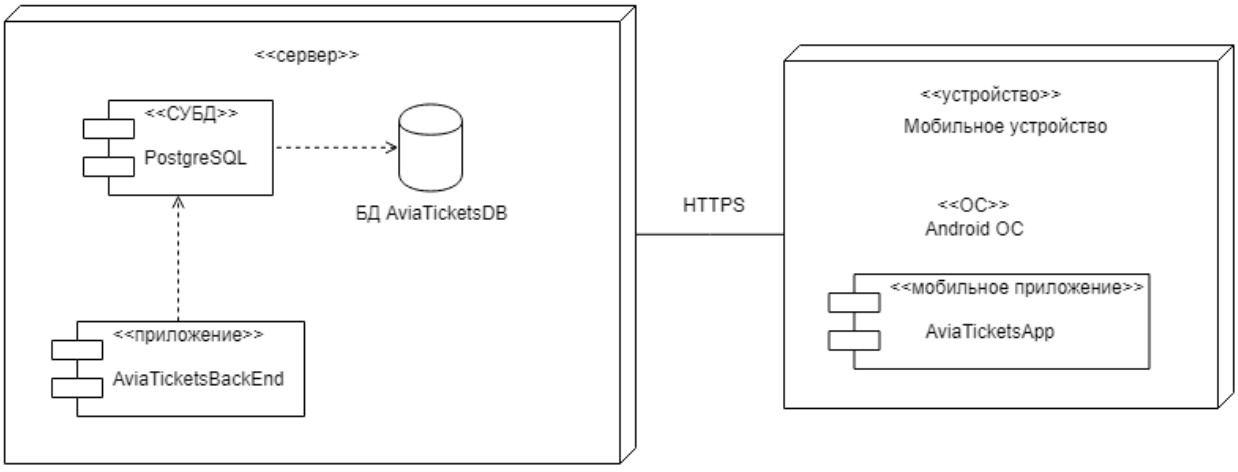

Рисунок 21 – Диаграмма развертывания

3. Реализация

В данной главе описана реализация всех подсистем приложения по поиску авиабилетов, задачи которых описаны в главе «Анализ» в разделе «Анализ задач».

3.1. Задача поиска авиабилетов

Входные данные для поиска организованы, как показано на диаграмме классов, изображенной на рисунке 22. Класс SearchData содержит в себе всю необходимую для поиска информацию, а именно:

- origin и destination – это пункты отправления и назначения, представляющие собой поле типа SearchPlace и содержащие название и код города;
- outboundDate и inboundDate – даты отправления и возвращения (в случае поездки в одну сторону дата возвращения будет null)
- adultsCount, childrenCount, infantsCount – количество пассажиров (взрослых, детей, младенцев);
- flightType – тип полета (в одну или обе стороны)
- transfers – поле, отвечающее за наличие билетов с пересадками в результатах поиска;
- cabinClass – класс полета (бизнес или эконом).

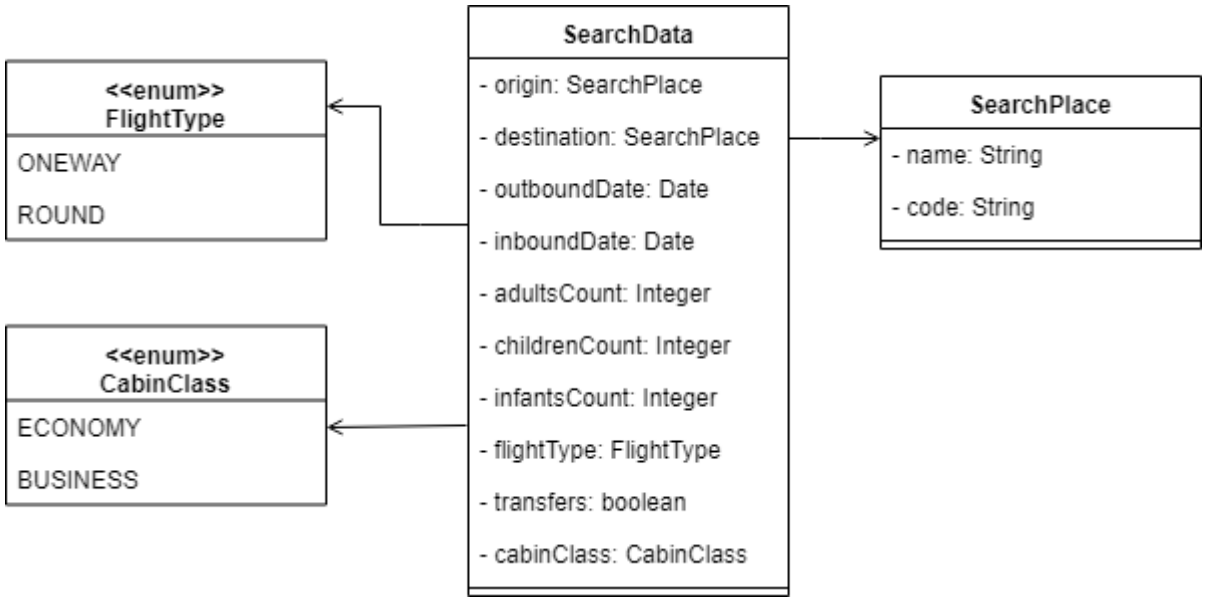

Рисунок 22 – Диаграмма классов входных данных

Для представления данных результатов поиска использована организация классов, изображенная на рисунке 23.

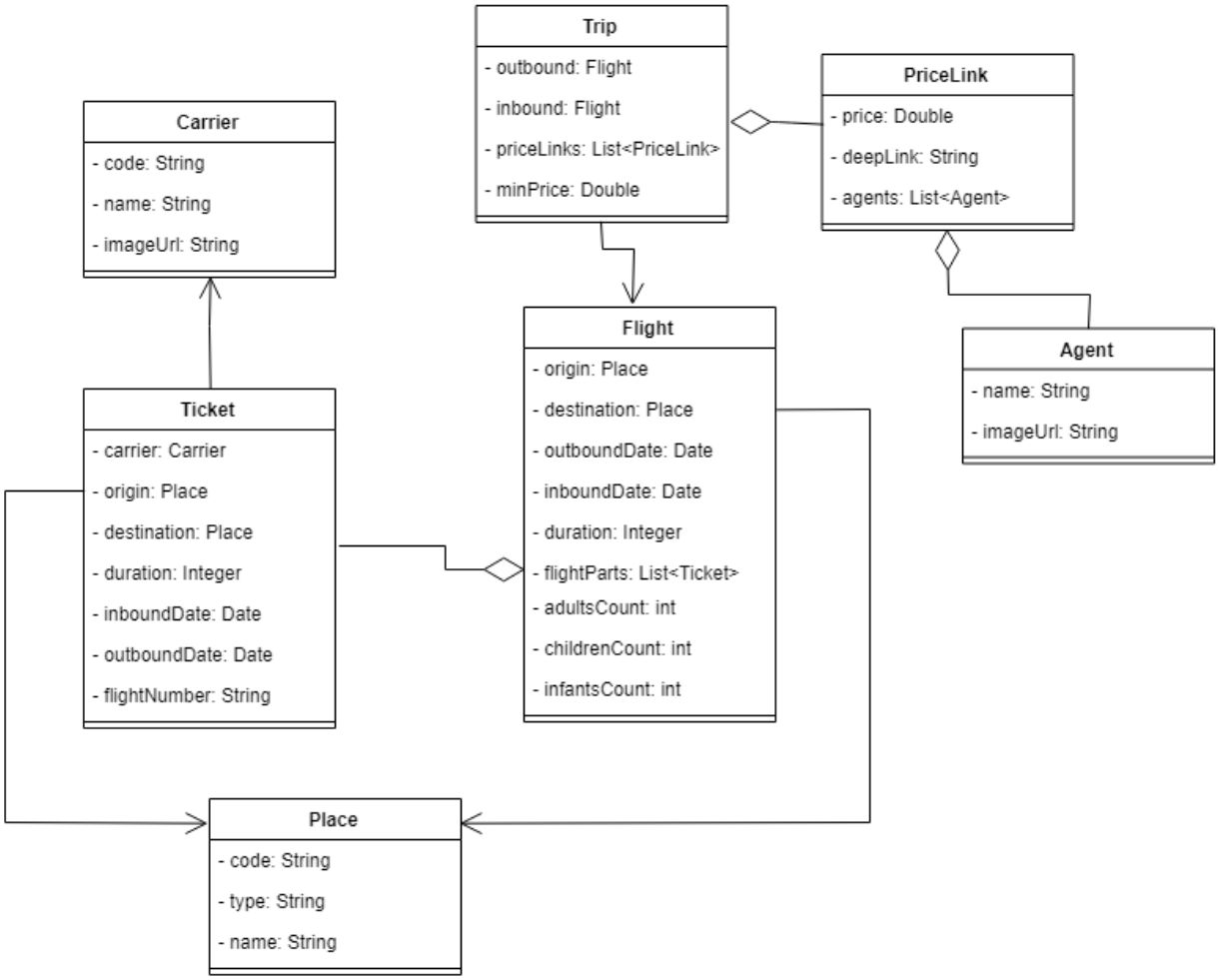

Рисунок 23 – Диаграмма классов выходных данных

Здесь класс Trip является основным и содержит в себе данные о полете в пункт назначения (outbound), о полете назад (inbound), список ссылок для покупки билета (priceLinks) и минимальная цена среди найденных билетов (minPrice).

Класс Flight описывает рейс. В нем содержатся следующие поля:

— origin и destination – это пункты отправления и назначения, представляющие собой поле типа Place и содержащие название (name) и код (code) города, а также тип пункта назначения (type) (город или аэропорт) ;

— outboundDate и inboundDate – даты отправления и возвращения (в случае поездки в одну сторону дата возвращения будет null)

— adultsCount, childrenCount, infantsCount – количество пассажиров (взрослых, детей, младенцев);

— duration – длительность рейса;

— flightParts – список билетов рейса (Ticket) (рейс может состоять из нескольких перелетов), которых содержится информация о перевозчике (класс Carrier), пункты отправления и назначения (класс Place), длительность полета, даты отправления и назначения, а также номер рейса (flightNumber);

Для уточнения диаграммы, изображенной на рисунке 23, была составлена диаграмма объектов, которая представлена на рисунке 24.

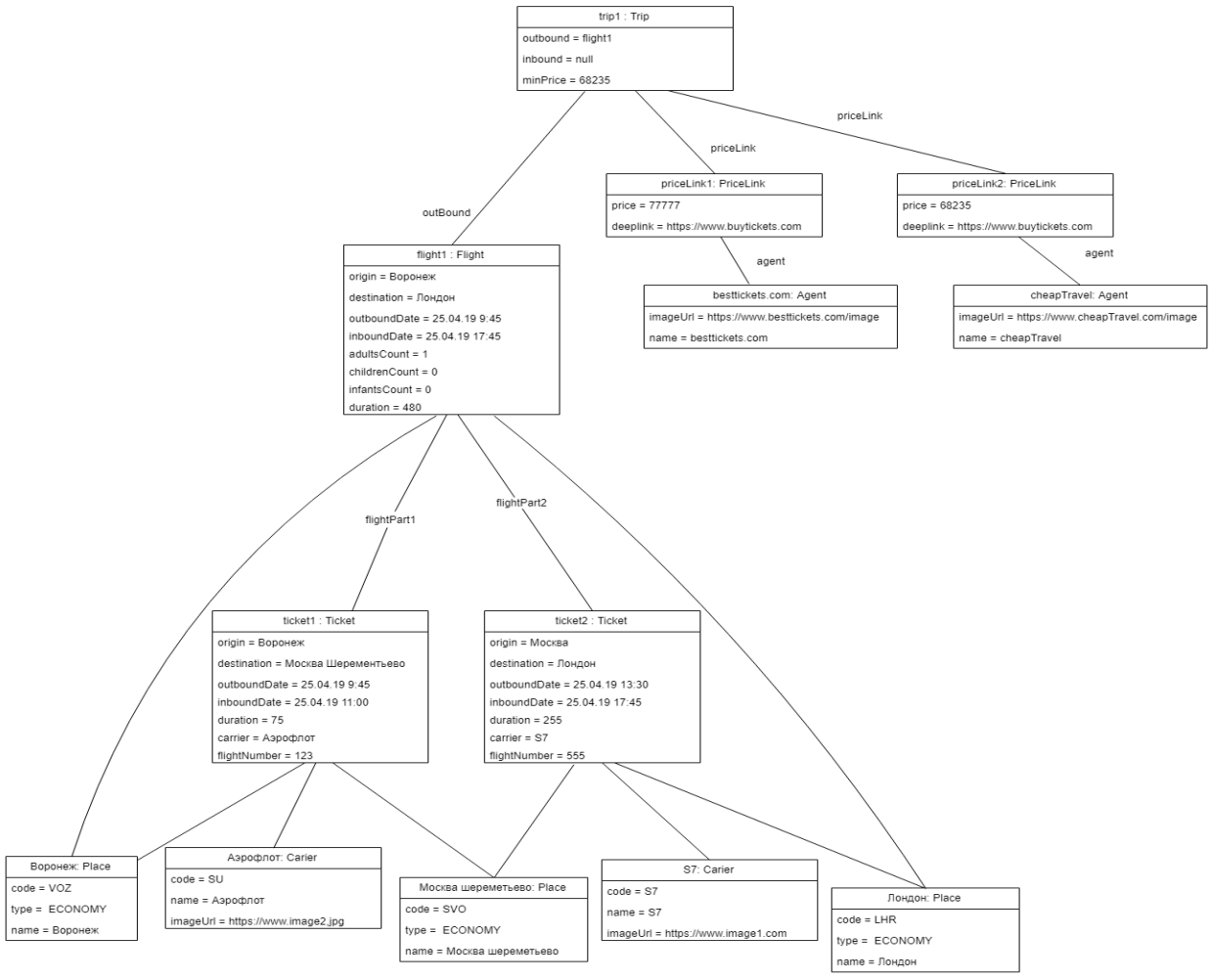

Рисунок 24 – Диаграмма объектов

Поиск билетов осуществляется на сервере с помощью запросов к сторонним источникам данных (далее API-источники). Были использованы SkyScanner API и Kiwi API. Для запросов к API-источникам и обработки ответов была использована организация классов, изображенная в виде диаграммы классов на рисунке 25.

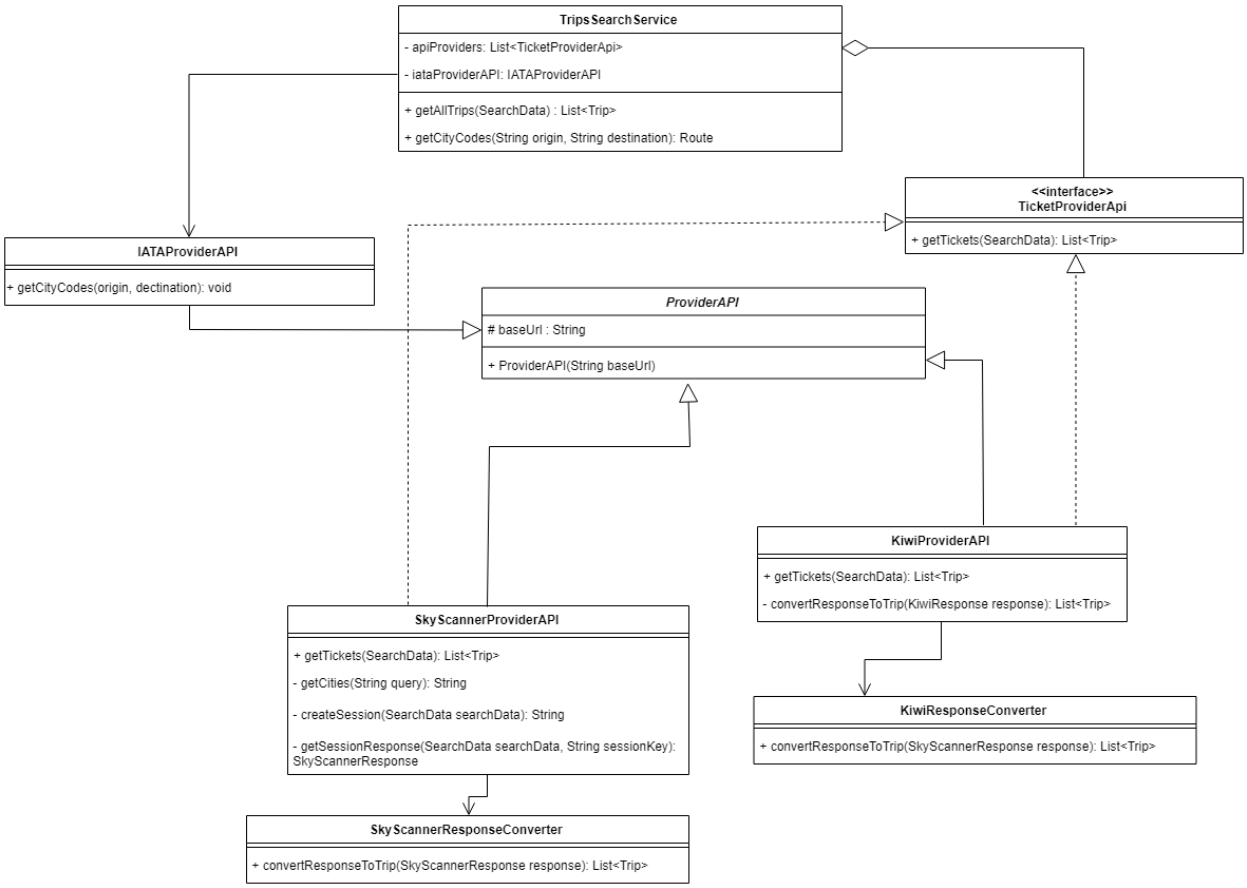

Рисунок 25 – Диаграмма классов общения с API-источниками

Класс ProviderAPI является абстрактным и содержит ссылку на API-источник (baseUrl). У данного класса есть три класса-наследника:

— SkyScannerProviderAPI, реализующий интерфейс TicketProviderApi и осуществляющий запрос к API-источнику SkyScanner;
— KiwiProviderAPI, реализующий интерфейс TicketProviderApi осуществляющий запрос к API-источнику Kiwi;
— IATAProviderAPI, осуществляющий запрос для получения кода города по его названию.

Для преобразования ответов API-источников к виду выходных данных описанных выше были использованы классы SkyScannerResponseConverter (для SkyScanner API) и KiwiResponseConverter (для Kiwi API).

Класс TripsSearchService является основным, в данном классе содержатся методы:

— getAllTrips – возвращает список найденных билетов в форме выходных данных по полученным входным данным в форме, описанной выше;

— getCityCodes – возвращает IATA-код городов назначения и отправления их его названиям.

3.2. Задача хранения данных пользователя

Для хранения истории поиска и закладок была выбрана база данных PostgreSQL. На рисунке 26 изображена схема базы данных, используемой в проекте.

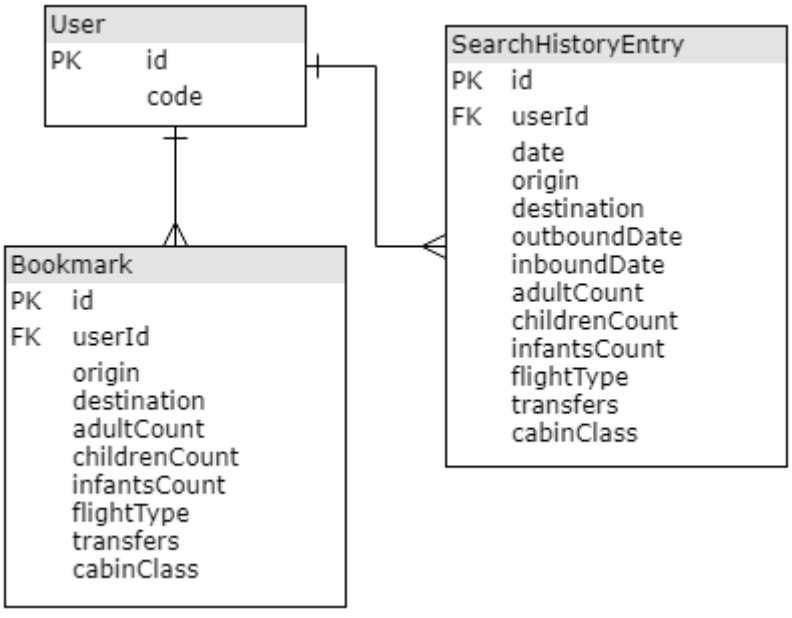

Рисунок 26 – Схема базы данных приложения

В данной схеме есть три таблицы:

— User

В этой таблице находится информация о пользователях, их id и code, определяемый id аккаунта Google.

— SerchHistoryEntry

Эта таблица предназначена для хранения истории поиска. Кроме id пользователя здесь находится названия городов отправления и прибытия, дата отправления и возвращения, количество пассажиров, наличие пересадок, класс и тип полета.

— Bookmark

В этой таблице хранятся закладки добавленные пользователем. Закладка включает в себя города отправления и прибытия, количество пассажиров, наличие пересадок, класс и тип перелета. Также здесь хранится id пользователя, которому принадлежит закладка.

3.3. Задача предоставления данных пользователю

3.3.1. Предоставление API для клиента на серверной части

Диаграмма классов для предоставления информации из базы данных об истории поиска изображена на рисунке 27.

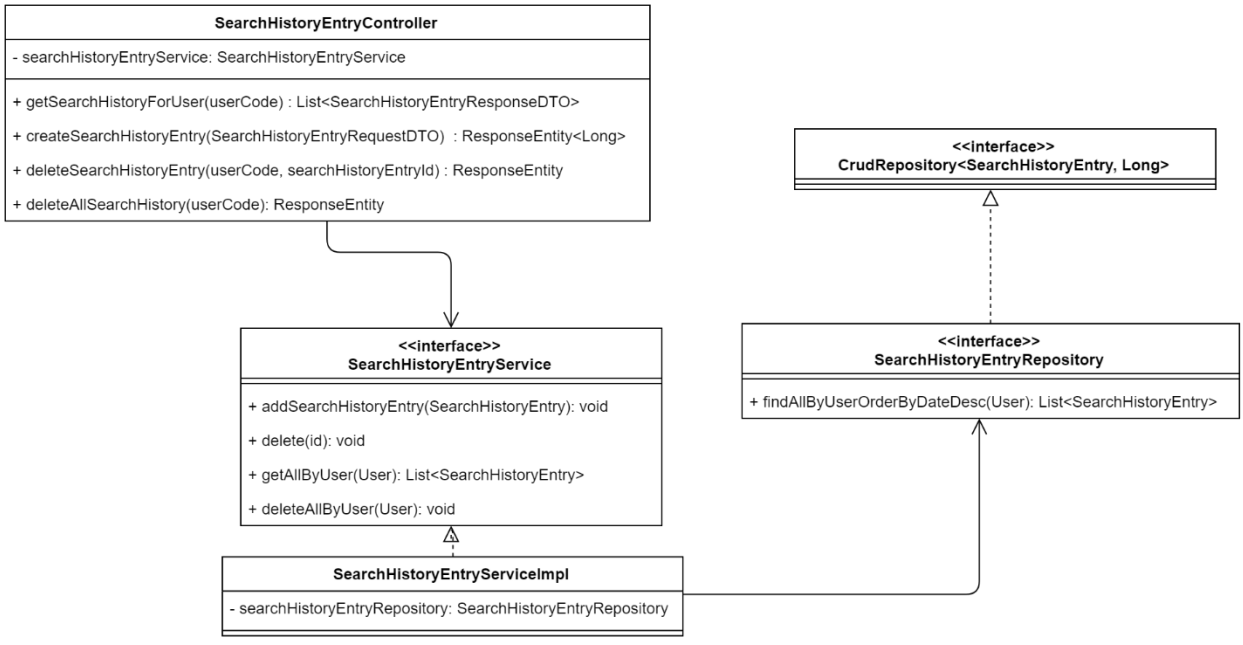

Рисунок 27 – Диаграмма классов предоставления API

Для извлечения данных и внесения изменений в базу данных об истории поиска используется интерфейс SearchHistoryEntryRepository, который наследуется от стандартного класса CrudRepository из фреймворка Spring Data JPA. Так же здесь есть метод findAllByUserOrderByDateDesc, который возвращает историю поиска для конкретного пользователя, упорядоченную по дате в порядке убывания.

В качестве посредника между пользователем и слоем данных использует класс SearchHistoryEntryServiceImpl, который реализует интерфейс SearchHistoryEntryService.

В данном классе есть экземпляр класса SearchHistoryEntryRepository, с помощью которого осуществляется работа с базой данных. Также в данном классе есть методы для добавления элемента истории поиска, получения всей истории пользователя, удаления одного элемента по id и удаления всей истории пользователя.

Для предоставления данных клиентской части используется класс SearchHistoryEntryController. В данном классе содержится экземпляр класса SearchHistoryEntryService.

Также в данном классе реализована обработка следующих запросов:

— GET-запрос по адресу "/users/{userCode}/search-history", где userCode – это код пользователя, с помощью функции getSearchHistoryForUser. Результатом является список элементов всей истории поиска по конкретному польщователю в формате JSON.
— POST-запрос по адресу "/search-history" с помощью функции createSearchHistoryEntry. В ходе выполнения данного запроса

будет добавлен элемент истории поиска, который передан в теле запроса.

— DELETE-запрос по адресу "/search-history/{userCode}/{searchHistoryEntryId}", где userCode – это код пользователя, searchHistoryEntryId – это id элемента истории поиска. В результате выполнения данного запроса будет удален элемент истории поиска с id, равным searchHistoryEntryId для соответствующего пользователя.

— DELETE-запрос по адресу "/search-history/{userCode}", где userCode – это код пользователя. В результате выполнения данного запроса будет удалена вся история поиска для соответствующего пользователя.

Взаимодействия с остальными API реализовано аналогично описанному выше.

3.3.2. Получение данных от сервера на клиентской части

Диаграмма классов для получения данных об истории поиска от серверной части изображена на рисунке 28.

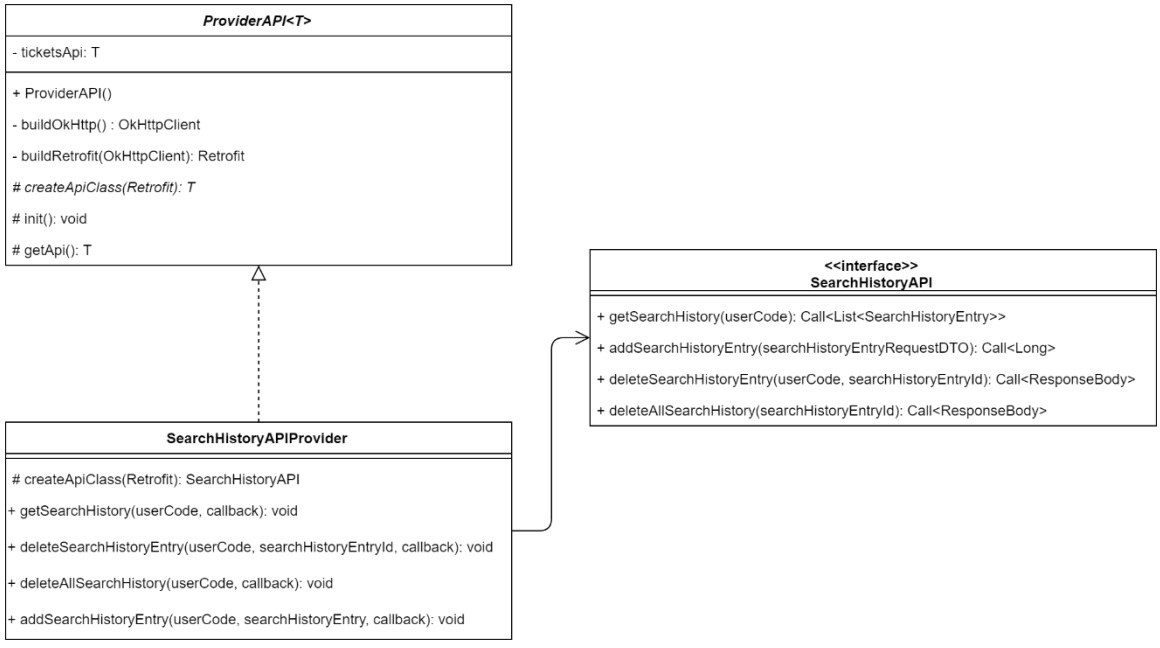

Рисунок 28 – Диаграмма классов получения данных из API серверной части

В интерфейсе SearchHistoryAPI описаны те же адреса и методы запросов, которые указаны на серверной части.

Запросы к API серверной части осуществляются с помощью библиотеки Retrofit2, инициализация всех объектов, необходимых для работы данной библиотеки вынесена в методы абстрактного класса ProviderAPI.

Класс SearchGistoryAPIProvider, предоставляющий доступ к API истории поиска наследуется от класса ProviderAPI. В нем есть методы для получения всей истории поиска, добавления одного элемента, удаления одного элемента и удаления всей истории поиска для конкретного пользователя. В качестве аргумента в эти функции передается интерфейс Callback, который имплементируется при вызове функции, он необходим для возврата данных, когда будет получен ответ от серверной части, или для возвращения сообщения об ошибке.

Получение данных из остальных API серверной части реализовано аналогично.

3.4. Задача отображения пользовательского интерфейса

Для отображения всех страниц, представляющих собой пользовательский интерфейс, была использована архитектура MVP, позволяющая отделить отображение интерфейса и логику работы с ним.

На рисунке 29 изображена часть диаграммы классов, с помощью которых реализована данная задача. По причине того, что каждая из страниц пользовательского интерфейса имеет похожую структуру классов, здесь подробно описана лишь страница с закладками. Остальные страницы реализованы аналогично.

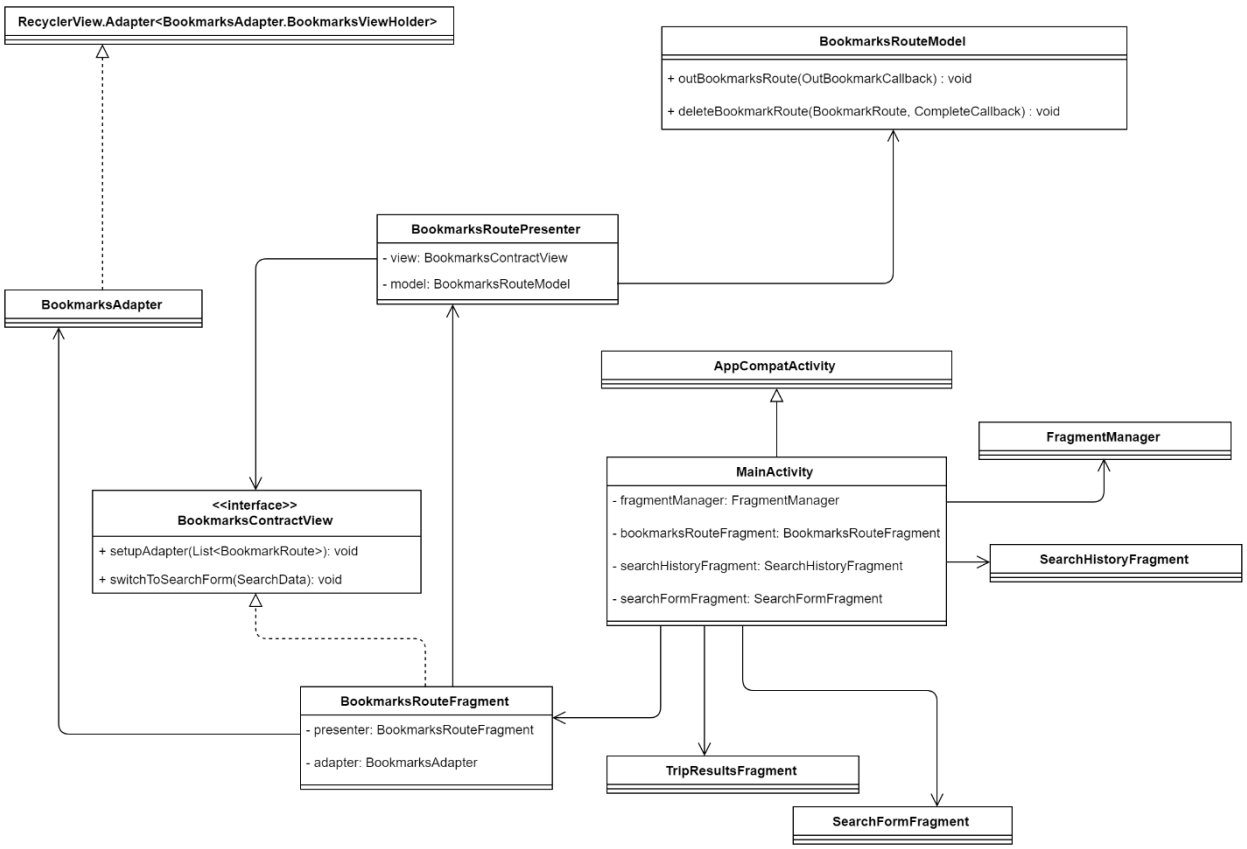

Рисунок 29 – Диаграмма классов отображения пользовательского интерфейса

Класс MainActivity наследуется от стандартного класса AppCompatActivity. Этот класс отвечает за отображение главного экрана, в нем содержится экземпляр стандартного класса FragmentManager, который отвечает за смену фрагментов, в зависимости от действий пользователей, а также сами фрагменты:

— SearchHistoryFragment, отвечающий за отображение истории поиска;
— SearchFormFragment, отвечающий за отображение формы поиска;
— TripResultFragment, отвечающий за отображение полученных результатов поиска билетов;
— BookmarkRouteFragment, отвечающий за отображение закладок.

В классе BookmarksRouteModel содержаться методы, осуществляющие взаимодействие с API, предоставленные серверной частью, а именно:

- outBookmarksRoute – метод, отвечающий за получение всех закладок, добавленных пользователем;
- deleteBookmarkRoute – метод, отвечающий за удаление закладки.

Закладки отображаются в виде списка элементов, реализованного в виде RecyclerView, для чего необходим класс BookmarkAdapter, который наследуется от стандартного класса RecyclerView.Adapter.

С целью введения архитектуры MVP введены классы BookmarksRouteFragment, BookmarksRoutePresenter и интерфейс BookmarksContractView, обеспечивающий связь BookmarksRouteFragment и BookmarksRoutePresenter.

Интерфейс BookmarksContractView определяет, какие действия возможны для отображения пользователю.

Класс BookmarksRouteFragment, реализующий интерфейс BookmarkContractView содержит в себе следующие поля:

— экземпляр класса BookmarksRoutePresenter, который оповещается при всех действиях пльзователя.
— Экземпляр класса BookmarkAdapter, для отображения списка элементов истории и взаимодействия с ними.

В качестве посредника между интерфейсом и слоем данный используется класс BookmarksRoutePresenter, в котором содержатся экземпляры класса BookmarksRouteModel и интерфейса BookmarkContractView. Данный класс, в зависимости от действий пользователя, принимает решения о том, какие методы класса BookmarksRouteModel и интерфейса BookmarkContractView необходимо использовать. В то время как методы этих классов лишь оповещают BookmarksRoutePresenter о своих изменениях.

3.5. Задача авторизации пользователя

На рисунке 30 представлена диаграмма классов, необходимых для авторизации пользователя.

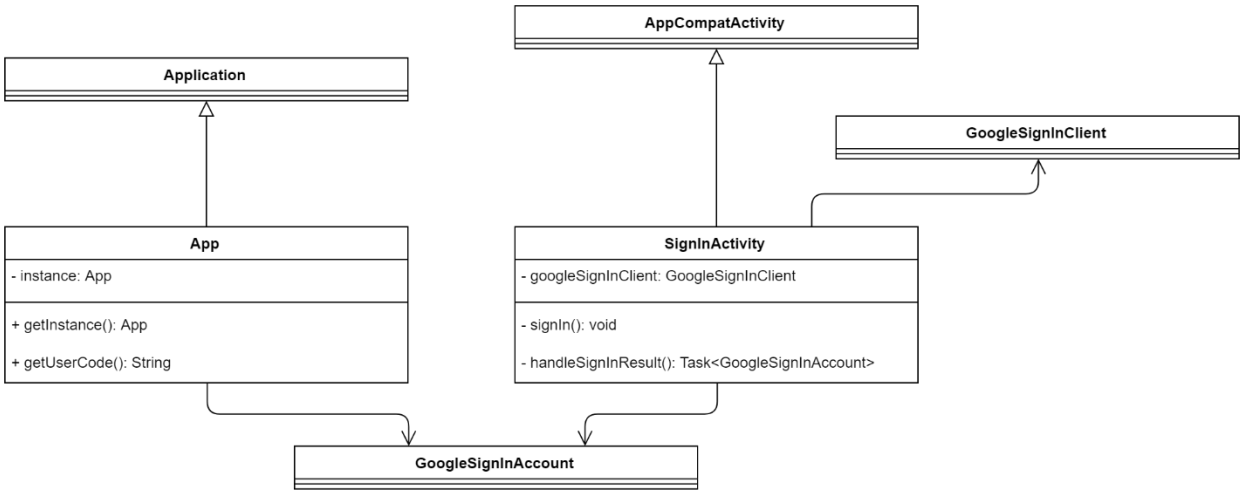

Рисунок 30 – Диаграмма классов, необходимых для авторизации пользователя

Класс SignInActivity наследуется от стандартного класса AppCompatActivity, данный класс отвечает за авторизацию пользователя в приложении. В нем содержится экземпляр стандартного класса GoogleSignInClient, который необходим для запуска формы авторизации. Также в этом классе находятся следующие методы:

— singIn(), который вызывается при нажатии пользователем на кнопку для входа в аккаунт. В этом методе вызывается открытые формы выбора аккаунта Google.
— handleSignInResult(), который вызывается после авторизации. В случае успешной авторизации этот метод открывает страницу поиска билетов, а в случае неуспешной – отображает сообщение об ошибке.

Для того, чтобы из любой точки приложения было возможно получить код авторизованного пользователя был добавлен метод getUserCode() в

основной класс приложения App, который наследуется от стандартного класса Application. Этот метод отвечает за получение аккаунта пользователя и возвращает его id.

4. Интерфейс

Пользовательский интерфейс приложения представлен в виде трех основных страниц и двух дополнительных, которые выполнены в бело-фиолетовой цветовой гамме с добавлением градиентного отображения цветов. В верхней части экрана находиться меню, где можно отключить или включить сохранение истории поиска, а также войти или выйти из аккаунта. В нижней части находиться навигационная панель, позволяющая перейти к страницам поиска, закладкам или истории поиска.

На странице поиска, изображенной на рисунке 31 слева, находится форма, включающая в себя:

- поля для ввода названий городов;
- кнопку для смены местами городов отправления и назначения;
- свиток для выбора типа путешествия (в одну или обе стороны);
- поля для ввода количества пассажиров (взрослых, детей и младенцев);
- поля для ввода дат с клавиатуры или с помощью виджета календаря;
- кнопку, осуществляющую поиск билетов.

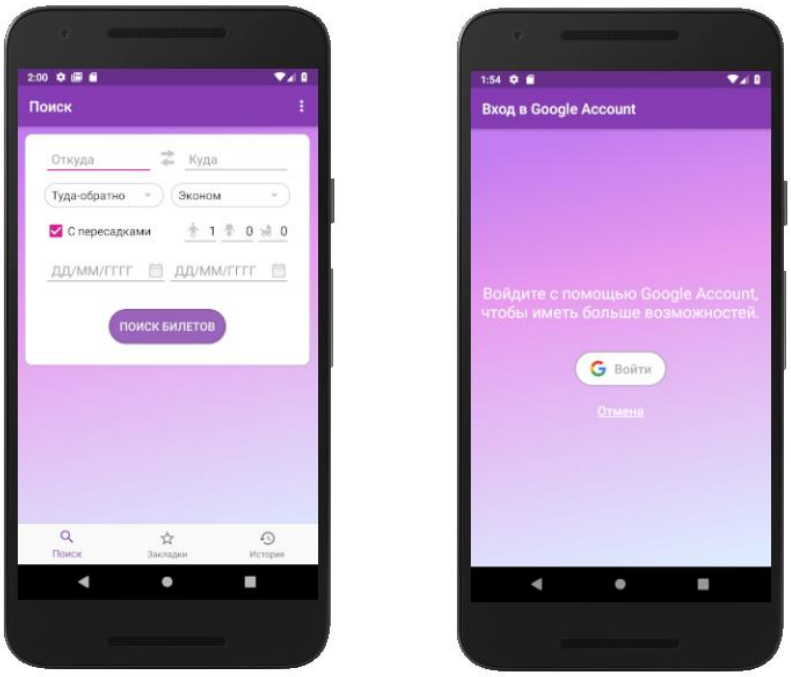

Рисунок 31 – Страница поиска и авторизации

Закладки и история поиска доступны только авторизированному пользователю. Для неавторизированного пользователя данные страницы отображают сообщение о необходимости авторизации и ссылку для ее осуществления. На рисунке 31 справа представлена страница авторизации через Google Account.

Для авторизованного пользователя страница с закладками, изображенная на рисунке 32 слева, отражает список добавленных ранее закладок. Справа от каждой из них находиться кнопка, осуществляющая удаление закладки. При нажатии на закладку происходит переход на страницу поиска, при этом поля ввода заполняются информацией, хранящейся в закладке.

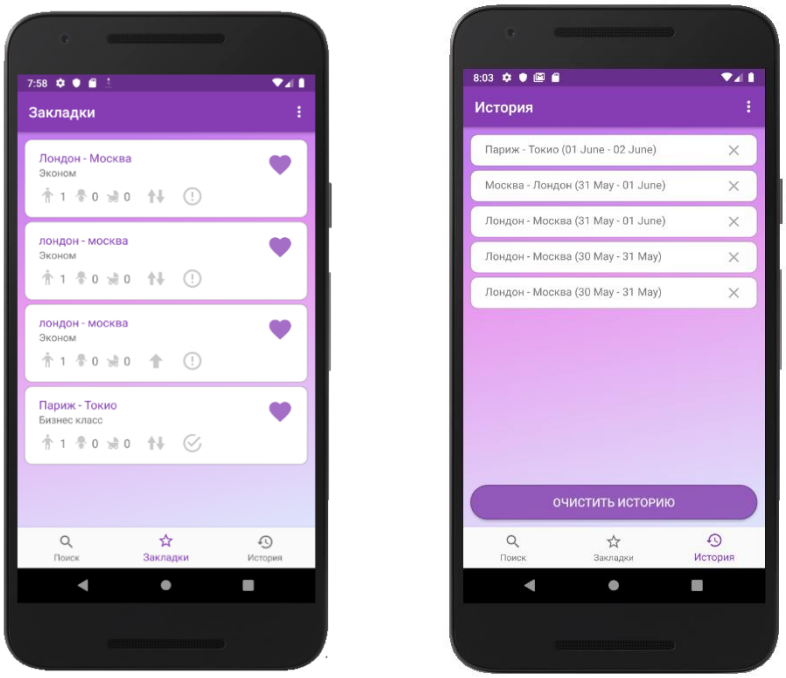

Рисунок 32 – Страницы закладок и истории поиска

Аналогично закладкам страница истории поиска, изображенная на рисунке 32 справа, содержит в себе список элементов истории поиска. В нижней части страницы находиться кнопка, позволяющая очистить историю поиска.

На рисунке 33 слева представлена страница отображения результатов поиска. Здесь в верхней части слева находиться кнопка добавления маршрута в закладки, изображенная в виде не закрашенного сердца (после добавления маршрута в закладки иконка сердца становиться закрашенной), а справа находиться свиток, позволяющий выбрать фильтр сортировки. При нажатии на конкретный результат поиска открывается страница, изображенная на рисунке 33 справа и отражающая подробную информацию о билете и ссылки для его покупки.

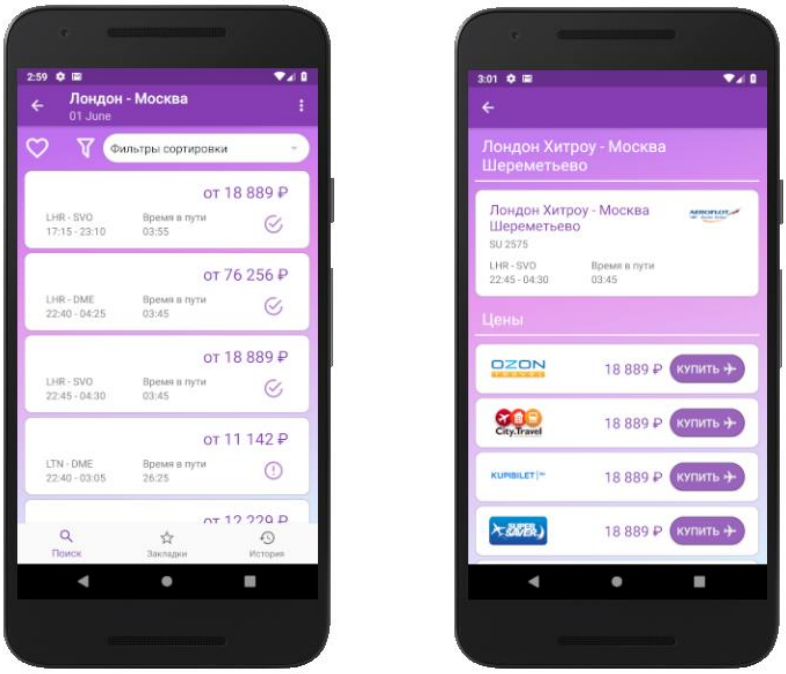

Рисунок 33 – Результаты поиска

При нажатии на иконки всплывает подсказка.

5. Тестирование

В целях проверки работоспособности системы в целом и подсистем приложения были проведены тесты по принципу «белого ящика» следующего вида:

— модульное тестирование;

— интеграционное тестирование;

— системное тестирование (smoke и sanity тесты на устройствах с версиями операционной системы Android 5 – 9 ручным методом и end-to-end тестирование);

— GUI тесты.

5.1. Модульное тестирование

Для проверки работоспособности функций подсистем было проведено автоматизированное модульное тестирование. Тестовые сценарии и результаты выполнения данного вида тестирования для серверной части системы представлены на рисунке 34.

Номер	Вид тестирования	Название	Шаги	Ожидаемый результат	Статус
AviaTicketsBackEnd (Back-end)					
1	Модульное тестирование	price_shouldFindMinPriceLink	1. Заполняется список из экемпляров класса PriceLink.	Объекты равны	Пройден
			2. Вызывается функция TripUtils.getMinPriceLink, в качестве аргумента передается сформированный список.		
			3. Объект из списка с минимальной ценой и объект, полученный в результате выполнения функции проверяются на равенство.		
2	Модульное тестирование	date_shouldConvertDateToString	1. Формируется ожидаемый результат вида "28-12-2015".	Строки равны	Пройден
			2. Формируется экземпляр класса Date, с помощью класса Calendar, соответствующий дате из сформированной строки.		
			3. Вызывается функция DateConvert.getStringFromDate, в качестве аргумента передается сформированная дата.		
			4. Сформированная строка и строка, полученная в результате вызова функции проверяются на равенство.		
3	Модульное тестирование	date_shouldConvertFromString	1. Вызывается функция DateConvert.getDateFromRequest, в качетсве агрумента передается строка вида "15-10-2018".	Даты равны	Пройден
			2. Формируется экземпляр класса Date, с помощью класса Calendar, соответствующий дате из сформированной строки.		
			3. Сформированная дата и дата, полученная в результате вызова функции проверяются на равенство.		
4	Модульное тестирование	responseConvert_shouldConvertKiwiResponse	1. Формируется ожидаемый результат списка объектов класса Trip.	Элементы списков равны	Пройден
			2. Формируется экземпляр класса KiwiResponse имитирующий ответ Kiwi.com.		
			3. Вызывается функция KiwiResponseConverter.convertResponseToTrip, в качестве аргумента передается сформированный объект KiwiResponse.		
			4. Сформированный изначально список и список, полученный в результате выполнения функции, проверяются на равенство.		
5	Модульное тестирование	responseConvert_shouldConvertSkyScannerResponse	1. Формируется ожидаемый результат списка объектов класса Trip.	Элементы списков равны	Пройден
			2. Формируется экземпляр класса SkyScannerResponse имитирующий ответ SkyScanner.		
			3. Вызывается функция SkyScannerResponseConverter.convertResponseToTrip, в качестве аргумента передается сформированный объект SkyScannerResponse.		
			4. Сформированный изначально список и список, полученный в результате		

Рисунок 34 – Модульное тестирование (сервер)

На рисунке 35 приведены тестовые сценарии и результаты модульного тестирования клиентской части приложения.

Номер	Вид тестирования	Название	Шаги	Ожидаемый результат	Статус
AviaTickets (Front-end)					
1	Модульное тестирование	date_shouldConvertFromString	1. Вызывается функция DateConvert.getDateFromStringWithSlashes, в качетсве агрумента передается строка вида "15/10/2018". 2. Формируется экземпляр класса Date, с помощью класса Calendar, соответствующий дате из сформированной строки. 3. Сформированная дата и дата, полученная в результате вызова функции проверяются на равенство.	Даты равны	Пройден
2	Модульное тестирование	date_shouldNotConvertFromString	1. Вызывается функция DateConvert.getDateFromStringWithSlashes, в качетсве агрумента передается строка вида "15.10.2018". 2. Проверяется, равна ли полученная дата null.	Дата равна null	Пройден
3	Модульное тестирование	date_shouldCovertDuration	1. Вызывается функция DateConvert.getDurationString, в качестве агрумента передается 75 минут. 2. Полученная в результате вызова строка сравнивается со строкой "01:15".	Строки равны	Пройден
4	Модульное тестирование	date_shouldNotCovertDuration	1. Вызывается функция DateConvert.getDurationString, в качестве агрумента передается -19 минут. 2. Проверяется, равна ли полученная строка null.	Строка равна null	Пройден
5	Модульное тестирование	date_shouldConvertDateToString	1. Формируется ожидаемый результат вида "28/12/2015". 2. Формируется экземпляр класса Date, с помощью класса Calendar, соответствующий дате из сформированной строки. 3. Вызывается функция DateConvert.getDateWithSlashes, в качестве аргумента передается сформированная дата. 4. Сформированная строка и строка, полученная в результате вызова функции проверяются на равенство.	Строки равны	Пройден
6	Модульное тестирование	date_shouldConvertAnotherDateToString	Аналогично прыдыдущему тесту, но с другой датой ("08/01/2015").	Строки равны	Пройден
7	Модульное тестирование	price_shouldFormatPrice	1. Формируется ожидаемый результат вида "112 631". 2. Вызывается функция StringUtils.formatPrice, в качестве аргумента передается цена 112630.9. 3. Сформированная строка и строка, полученная в результате вызова функции проверяются на равенство.	Строки равны	Пройден
8	Модульное тестирование	price_shouldFormatAnotherPrice	Аналогично прыдыдущему тесту, но с другой ценой ("5 671").	Строки равны	Пройден
9	Модульное тестирование	sort_shouldSortByPriceAsc	1. Заполняется список из экемпляров класса Trip. 2. Создается список из элементов первого списка, упорядоченный по цене по возрастанию. 3. Вызывается метод sortTrips, в качестве аргументов передается список неотсортированных элементов и переменная SortFilterType.MIN_PRICE, обозначающая сортировку по цене по возрастанию. 4. Списки проверяются на равенство.	Элементы списков равны	Пройден
10	Модульное тестирование	sort_shouldSortByPriceDesc	1. Заполняется список из экемпляров класса Trip. 2. Создается список из элементов первого списка, упорядоченный по цене по убыванию. 3. Вызывается метод sortTrips, в качестве аргументов передается список неотсортированных элементов и переменная SortFilterType.MAX_PRICE, обозначающая сортировку по цене по убыванию. 4. Списки проверяются на равенство.	Элементы списков равны	Пройден
11	Модульное тестирование	sort_shouldSortByTransfersAsc	1. Заполняется список из экемпляров класса Trip. 2. Создается список из элементов первого списка, упорядоченный по количеству пресадок по возрастанию. 3. Вызывается метод sortTrips, в качестве аргументов передается список неотсортированных элементов и переменная SortFilterType.MIN_TRANSFERS, обозначающая сортировку по количеству пресадок по возрастанию. 4. Списки проверяются на равенство.	Элементы списков равны	Пройден
12	Модульное тестирование	sort_shouldSortByTransfersDesc	1. Заполняется список из экемпляров класса Trip. 2. Создается список из элементов первого списка, упорядоченный по количеству пресадок по убыванию. 3. Вызывается метод sortTrips, в качестве аргументов передается список неотсортированных элементов и переменная SortFilterType.MAX_TRANSFERS, обозначающая сортировку по количеству пресадок по убыванию. 4. Списки проверяются на равенство.	Элементы списков равны	Пройден
13	Модульное тестирование	sort_shouldSortByTimeAsc	1. Заполняется список из экемпляров класса Trip. 2. Создается список из элементов первого списка, упорядоченный по времени в пути по возрастанию. 3. Вызывается метод sortTrips, в качестве аргументов передается список неотсортированных элементов и переменная SortFilterType.MIN_TIME, обозначающая сортировку по времени в пути по возрастанию. 4. Списки проверяются на равенство.	Элементы списков равны	Пройден
14	Модульное тестирование	sort_shouldSortByTimeDesc	1. Заполняется список из экемпляров класса Trip. 2. Создается список из элементов первого списка, упорядоченный по времени в пути по убыванию. 3. Вызывается метод sortTrips, в качестве аргументов передается список неотсортированных элементов и переменная SortFilterType.MAX_TIME, обозначающая сортировку по времени в пути по убыванию. 4. Списки проверяются на равенство.	Элементы списков равны	Пройден

Рисунок 35 – Модульное тестирование (клиент)

По результатам модульного тестирования можно сделать вывод, что приложение работает корректно как в серверной, так и в клиентской части.

5.2. Интеграционное тестирование

Для проверки работоспособности в целом и взаимосвязи между частями системы были проведены автоматизированные интеграционные

тесты. Тестовые сценарии и результаты данного вида тестирования представлены на рисунке 36.

Номер	Вид тестирования	Название	Шаги	Ожидаемый результат	Статус
1	Интеграционное тестирование	users_shouldGetAllUsers	1. Выполняется Get-запрос к localhost:port/users. 2. Проверяется полученный код состояния. 3. Проверяется, что полученный ответ не равен null.	Код состояния равен 200, полученный ответ не равен null	Пройден
2	Интеграционное тестирование	users_shouldCreateUser	1. Выполняется Post-запрос к localhost:port/users. 2. Проверяется полученный код состояния. 3. Проверяется, что полученный ответ не равен null.	Код состояния равен 200, полученный ответ не равен null	Пройден
3	Интеграционное тестирование	bookmarks_shouldGetAllBookmarkForUser	1. Выполняется Get-запрос к localhost:port/users/{code}/bookmarks, где code - это код пользователя. 2. Проверяется полученный код состояния. 3. Проверяется, что полученный ответ не равен null.	Код состояния равен 200, полученный ответ не равен null	Пройден
4	Интеграционное тестирование	bookmarks_shouldCreateBookmark	1. Выполняется Post-запрос к localhost:port/bookmarks. 2. Проверяется полученный код состояния. 3. Проверяется, что полученный ответ не равен null.	Код состояния равен 200, полученный ответ не равен null	Пройден
5	Интеграционное тестирование	bookmarks_shouldFindBookmark	1. Выполняется Get-запрос к localhost:port/bookmarks/find. 2. Проверяется полученный код состояния со всеми необходимыми параметрами запроса. 3. Проверяется, что полученный ответ не равен null.	Код состояния равен 200, полученный ответ не равен null	Пройден
6	Интеграционное тестирование	bookmarks_shouldDeleteBookmark	1. Выполняется Post-запрос к localhost:port/bookmarks для добавления новой закладки. 2. Проверяется полученный код состояния. 3. Проверяется, что полученный ответ не равен null. 4. Выполняется Get-запрос к localhost:port/users/{code}/bookmarks, где code - это код пользователя, чтобы получить все закладки пользователя. 5. Проверяется полученный код состояния. 6. Проверяется, что полученный ответ не равен null. 7. Проверяется, что в полученном списке закладок есть хотя бы одна. 8. Выполняется Delete-запрос к localhost:port/bookmarks/{id}, где id - это id первой закладки из полученного списка. 9. Выполняется Get-запрос к localhost:port/users/{code}/bookmarks, где code - это код пользователя, чтобы получить все закладки пользователя. 10. Проверяется полученный код состояния. 11. Проверяется, что полученный ответ не равен null. 12. Проверяется, что количество закладок, полученных во второй раз на один меньше, чем в первый	Код состояния во всех случаях равен 200, полученный ответ во всех случаях не равен null, закладка была добавлена, затем была удалена	Пройден
7	Интеграционное тестирование	searchHistory_shouldGetAllSearchHistoryForUser	1. Выполняется Get-запрос к localhost:port/users/{code}/search-history, где code - это код пользователя. 2. Проверяется полученный код состояния. 3. Проверяется, что полученный ответ не равен null.	Код состояния равен 200, полученный ответ не равен null	Пройден
8	Интеграционное тестирование	searchHistory_shouldAddSearchHistoryEntry	1. Выполняется Post-запрос к localhost:port/search-history. 2. Проверяется полученный код состояния. 3. Проверяется, что полученный ответ не равен null.	Код состояния равен 200, полученный ответ не равен null	Пройден
9	Интеграционное тестирование	searchHistory_shouldDeleteSearchHistoryEntry	1. Выполняется Post-запрос к localhost:port/search-history для добавления нового элемента истории поиска. 2. Проверяется полученный код состояния. 3. Проверяется, что полученный ответ не равен null. 4. Выполняется Get-запрос к localhost:port/users/{code}/search-history, где code - это код пользователя, чтобы получить всю истории поиска пользователя. 5. Проверяется полученный код состояния. 6. Проверяется, что полученный ответ не равен null. 7. Проверяется, что в полученном списке элементов истории поиска есть хотя бы один. 8. Выполняется Delete-запрос к localhost:port/search-history/{code}/{id}, где id - это id первого элемента из полученного списка. 9. Выполняется Get-запрос к localhost:port/users/{code}/search-history, где code - это код пользователя, чтобы получить всю истории поиска пользователя. 10. Проверяется полученный код состояния. 11. Проверяется, что полученный ответ не равен null. 12. Проверяется, что количество элементов полученных во второй раз на один меньше, чем в первый	Код состояния во всех случаях равен 200, полученный ответ во всех случаях не равен null, элемент был добавлен, затем был удален	Пройден
10	Интеграционное тестирование	searchHistory_shouldDeleteAllSearchHistory	1. Дважды выполняется Post-запрос к localhost:port/search-history для добавления нового элемента истории поиска. 2. Проверяется полученный код состояния. 3. Проверяется, что полученный ответы не равны null. 4. Выполняется Get-запрос к localhost:port/users/{code}/search-history, где code - это код пользователя, чтобы получить всю истории поиска пользователя. 5. Проверяется полученный код состояния. 6. Проверяется, что полученный ответ не равен null. 7. Выполняется Delete-запрос к localhost:port/search-history/{code} для удаления всей истории. 8. Выполняется Get-запрос к localhost:port/users/{code}/search-history, где code - это код пользователя, чтобы получить всю истории поиска пользователя. 9. Проверяется полученный код состояния. 10. Проверяется, что полученный ответ не равен null. 11. Проверяется, что количество элементов полученных во второй раз равно 0.	Код состояния во всех случаях равен 200, полученный ответ во всех случаях не равен null, элементы были добавлены, затем все удалены	Пройден
11	Интеграционное тестирование	cities_shouldGetCities	1. Выполняется Get-запрос к localhost:port/cities с необходимымы параметрами. 2. Проверяется полученный код состояния. 3. Проверяется, что полученный ответ не равен null.	Код состояния равен 200, полученный ответ не равен null	Пройден
12	Интеграционное тестирование	trips_shouldGetTrips	1. Выполняется Get-запрос к localhost:port/trips с необходимымы параметрами. 2. Проверяется полученный код состояния. 3. Проверяется, что полученный ответ не равен null.	Код состояния равен 200, полученный ответ не равен null	Пройден

Рисунок 36 – Интеграционное тестирование

Как видно из рисунка 36, все тесты пройдены успешно, что означает корректное функционирование системы в целом.

5.3. Системное тестирование

С целью убедиться, что система корректно работает на всех наиболее используемых на данный момент версиях операционной системы Android, было проведено ручное тестирование по составленным заранее тестовым сценариям, информация о которых содержится в приложении В на рисунке В.1. Данный тест призван проверить работоспособность основной функциональности системы не только в хороших сценариях, но и плохих.

Были проведены тесты на следующих устройствах:

- Pixel 3 XL API 28 Android 9.0;
- Pixel 2 API 27 Android 8.1;
- Pixel 2 XL API 26 Android 8.0;
- Pixel API 25 Android 7.1;
- Nexus 5X API 24 Android 7.0;
- Nexus 9 API 23 Android 6.0;
- Nexus 7 API 22 Android 5.1;
- Nexus 5 API 21 Android 5.0;

Результаты тестирования на устройствах приведены в приложении В на рисунках В.2 и В.3. По ним можно сделать вывод, что система функционирует корректно на всех протестированных устройствах для всех тестовых сценариев кроме номера 5. При выполнении данного тестового сценария на всех протестированных устройствах система при вводе несуществующей даты ищет билеты на дату равную последнему числу месяца плюс лишние дни (например, для 32 мая 2019 найдены билеты для 1 июня 2019).

Также было проведено end-to-end тестирование для незарегистрированного и зарегистрированного пользователей. Тестовые

сценарии и результаты представлены на рисунке 37. Как видно, тестирование прошло успешно.

Незарегистрированный пользователь	Запустить приложение	Приложение запустилось Отобразилась страница авторизации	Соответствует ожидаемому	Пройден
	Нажать "Отмена" на странице авторизации	Отобразилась страница поиска билетов	Соответствует ожидаемому	Пройден
	Ввести параметры для поиска и нажать на кнопку "Поиск билетов"	Отобразилась страница с найденными билетами, соответсвующими введенным данным	Соответствует ожидаемому	
	Выбрать фильтры сортировки	Список результатов отсортировался согласно выбранному типу сортировки	Соответствует ожидаемому	Пройден
	Нажать на иконку сердечка	Вывелось сообщение о необходимостти авторизоваться	Соответствует ожидаемому	
	Нажать на элемент результата	Отобразилась страница с подробной информацией о билетах	Соответствует ожидаемому	Пройден
	Перейти на страницу с закладками	Отобразилась страница с сообщением о необходимости авторизации	Соответствует ожидаемому	
	Перейти на страницу с историей поиска	Отобразилась страница с сообщением о необходимости авторизации	Соответствует ожидаемому	Пройден
	В меню в верхней части страницы выбрать пункт войти в Google аккаунт	Отобразилась страница авторизации	Соответствует ожидаемому	
Зарегистрированный пользователь	Нажать "Войти" на странице авторизации и выбрать аккаунт для входа	Произошла авторизация Отобразилась страница поиска билетов	Соответствует ожидаемому	Пройден
	Ввести параметры для поиска и нажать на кнопку "Поиск билетов"	Отобразилась страница с найденными билетами, соответсвующими введенным данным	Соответствует ожидаемому	Пройден
	Выбрать фильтры сортировки	Список результатов отсортировался согласно выбранному типу сортировки	Соответствует ожидаемому	Пройден
	Нажать на иконку сердечка	Маршрут добавился в закладки Иконка сердечка стала закрашенной	Соответствует ожидаемому	Пройден
	Повторно нажать на иконку сердечка	Иконка сердечка стала закрашенной Закладка удалилась	Соответствует ожидаемому	Пройден
	Нажать на элемент результата	Отобразилась страница с подробной информацией о билетах	Соответствует ожидаемому	Пройден
	Перейти на страницу с закладками	Отобразилась страница со списком добавленных закладок	Соответствует ожидаемому	Пройден
	Нажать на иконку сердечка справа от закладки	Закладка удалилась	Соответствует ожидаемому	Пройден
	Нажать на закладку	Произошел переход на станицу поиска, праметры поиска заполнены данными из закладки	Соответствует ожидаемому	Пройден
	Перейти на страницу с историей поиска	Отобразилась страница с историей поиска		Пройден
	Нажать на крестик справа от элемента истории поиска	Элемент удален	Соответствует ожидаемому	Пройден
	Нажать на элемент истории поиска	Произошел переход на станицу поиска, праметры поиска заполнены данными из закладки	Соответствует ожидаемому	Пройден
	Нажать на кнопку "Очистить историю поиска"	История поиска очищена	Соответствует ожидаемому	Пройден
	В меню в верхней части страницы снять галочку с пункта "Не сохранять историю поиска"	При поиске история не сохранилась	Соответствует ожидаемому	Пройден
	В меню в верхней части страницы выбрать пункт выйти из Google аккаунт	Произошел выход из аккаунта Отобразилась страница авторизации	Соответствует ожидаемому	Пройден

Рисунок 37 – End-to-end тестирование

5.4. GUI-тесты

Для проверки корректной работы пользовательского интерфейса было проведено автоматизированное GUI-тестирование, тестовые сценарии и

результаты которого представлены на рисунке 38. Исходя из результатов, можно сделать вывод, что пользовательский интерфейс работает правильно.

Номер	Вид тестирования	Название	Шаги	Ожидаемый результат	Статус
2	UI	mainActivityActionBarTitlesTest	1. Переход на вкладку "Поиск". 2. Проверяется, равен ли заголовок слову "Поиск". 3. Переход на вкладку "Закладки". 4. Проверяется, равен ли заголовок слову "Закладки". 5. Переход на вкладку "История". 6. Проверяется, равен ли заголовок слову "История".	Заголовки должны соответствовать вкладкам	Пройден
3	UI	mainActivityMenuOptionsTest	1. Наживается кнопка открытия меню. 2. Проверяется, открылось ли меню.	Меню открылось	Пройден
4	UI	searchFormChangeCitiesTest	1. Открывается форма поиска. 2. Вводится город оправления. 3. Вводится город прибытия. 4. Нажимается кнопка "поменять города местами". 5. Проверяется, поменялись ли введенные данные местами.	Введенные данные должны поменяться местами	Пройден
5	UI	searchFormDateFromTest	1. Открывается форма поиска. 2. Нажимажется кнопка "Открыть календарь" у поля ввода даты отправления. 3. Проверяется, открылся ли календарь для выбора даты.	Календарь открылся	Пройден
6	UI	searchFormDateToTest	Аналогично предыдущему тесту для кнопки у поля ввода даты возвращения.	Календарь открылся	Пройден
7	UI	tooManyPassengersTest	1. Открывается форма поиска. 2. В поле "количество взрослых пассажиров" вводится "9". 3. Нажимается кнопка "поиск". 4. Проверяется, появилось ли соответсвующее сообщение об ошибке.	Появилось сообщение об ошибка	Пройден
8	UI	tooManySumPassengersTest	1. Открывается форма поиска. 2. В поле "количество взрослых пассажиров" вводится "5". 3. В поле "количество детей" вводится "4". 4. Нажимается кнопка "поиск". 5. Проверяется, появилось ли соответсвующее сообщение об ошибке.	Появилось сообщение об ошибка	Пройден
9	UI	noPassengersTest	1. Открывается форма поиска. 2. В поле "количество взрослых пассажиров" вводится "0". 3. В поле "количество детей" вводится "0". 4. В поле "количество младенцев" вводится "0". 5. Нажимается кнопка "поиск". 6. Проверяется, появилось ли соответсвующее сообщение об ошибке.	Появилось сообщение об ошибка	Пройден
10	UI	tooManyInfantsTest	1. Открывается форма поиска. 2. В поле "количество взрослых пассажиров" вводится "2". 3. В поле "количество детей" вводится "3". 4. Нажимается кнопка "поиск". 5. Проверяется, появилось ли соответсвующее сообщение об ошибке.	Появилось сообщение об ошибка	Пройден
11	UI	noDateFromTest	1. Открывается форма поиска. 2. Вводятся данные о пассажирах. 3. Поля для ввода дат оставляются пустыми. 4. Нажимается кнопка "поиск". 5. Проверяется, появилось ли соответсвующее сообщение об ошибке.	Появилось сообщение об ошибка	Пройден
12	UI	dateFromWrongFormatTest	1. Открывается форма поиска. 2. Вводятся данные о пассажирах. 3. В поле ввода даты отправления вводится дата в формате "ДД.ММ.ГГГГ". 4. Нажимается кнопка "поиск". 5. Проверяется, появилось ли соответсвующее сообщение об ошибке.	Появилось сообщение об ошибка	Пройден
13	UI	dateToDisabledTest	1. Открывается форма поиска. 2. Вводятся данные о пассажирах. 3. Выбирается тип полета "В одну сторону". 4. Проверяется, стало ли поле для ввода даты возвращения недоступным для изменения.	Поле для ввода даты возвращение недоступно для изменения	Пройден
14	UI	noDateToTest	1. Открывается форма поиска. 2. Вводятся данные о пассажирах. 3. Выбирается тип полета "Туда-обратно". 4. Вводится дата отправления. 5. Поле для ввода даты возвращения оставляется пустым. 6. Нажимается кнопка "поиск". 7. Проверяется, появилось ли соответсвующее сообщение об ошибке.	Появилось сообщение об ошибка	Пройден
15	UI	dateToWrongFormatTest	1. Открывается форма поиска. 2. Вводятся данные о пассажирах. 3. Выбирается тип полета "Туда-обратно". 4. Вводится дата отправления. 5. В поле ввода даты возвращения вводится дата в формате "ДД.ММ.ГГГГ". 6. Нажимается кнопка "поиск". 7. Проверяется, появилось ли соответсвующее сообщение об ошибке.	Появилось сообщение об ошибка	Пройден
16	UI	yesterdayDateFromTest	1. Открывается форма поиска. 2. Вводятся данные о пассажирах. 3. В поле ввода даты отправления вводится вчерашняя дата. 4. Нажимается кнопка "поиск". 5. Проверяется, появилось ли соответсвующее сообщение об ошибке.	Появилось сообщение об ошибка	Пройден
17	UI	dateToAfterDateFromTest	1. Открывается форма поиска. 2. Вводятся данные о пассажирах. 3. Выбирается тип полета "Туда-обратно". 4. Вводится дата отправления. 5. Вводится дата возвращения, которая раньше даты отправления. 6. Нажимается кнопка "поиск". 7. Проверяется, появилось ли соответсвующее сообщение об ошибке.	Появилось сообщение об ошибка	Пройден
18	UI	yesterdayDateToTest	1. Открывается форма поиска. 2. Вводятся данные о пассажирах. 3. Выбирается тип полета "Туда-обратно". 4. Вводится дата отправления. 5. В поле ввода даты возвращения вводится вчерашняя дата. 6. Нажимается кнопка "поиск". 7. Проверяется, появилось ли соответсвующее сообщение об ошибке.	Появилось сообщение об ошибка	Пройден

Рисунок 38 – GUI-тестирование

5.5. Вывод

Исходя из результатов тестов, существуют мелкие незначительные недочеты, касающиеся проверки правильности введенных данных, а именно введенная вручную некорректная дата (с числом, большим, чем количество дней во введенном месяце) приводит к поиску билетов на более позднюю дату.

Однако приведенные выше недочеты не влияют существенно на работоспособность системы в целом и ее основных частей. Можно сделать вывод о том, что система полностью работоспособна и соответствует поставленной задаче и требованиям, описанным в главе 1 и разделе 2.2 данной работы.

Заключение

В ходе выполнения был проведен анализ предметной области поиска авиабилетов, были составлены диаграммы активности, состояния, прецедентов, последовательностей коммуникаций и развертывания.

Результатом выполнения было реализовано Android-приложение по поиску билетов, которое реализует следующие возможности:

- авторизоваться;
- найти авиабилеты на рейсы в один конец с возможностью добавления информации о количестве пассажиров, типе путешествия (в один или оба конца), классе полета (эконом или бизнес), наличии пересадок;
- найти авиабилеты на рейсы в оба конца с возможностью добавления информации о количестве пассажиров, типе путешествия (в один или оба конца), классе полета (эконом или бизнес), наличии пересадок;
- сортировать результаты поиска.

Также для авторизованного пользователя в дополнение к возможностям неавторизованного пользователя приложение обеспечивает следующие возможности:

- добавить маршрут в закладки;
- просматривать закладки;
- удалять закладки;
- просматривать историю поиска;
- удалять элемент истории поиска;
- очищать историю поиска;
- не сохранять историю поиска**;**
- переходить со страницы закладок и истории поиска к поиску авиабилетов с заполнением данных поиска;
- выйти из аккаунта.

Для поиска авиабилетов приложение использует несколько сторонних источников данных. Также система подбирает не только прямые рейсы, но и с пересадками.

Было проведено тестирование системы и установлено, что она работоспособна и функционирует корректно на практически всех входных данных.

Список использованных источников

1 Start Android - учебник по Android для начинающих и продвинутых [сайт] – URL: http://www.fandroid.info (дата обращения: 28.02.2019 – 31.05.2019)

2 Уроки по разработке Android -приложений [сайт] – URL: https://www.fandroid.info (дата обращения: 28.02.2019 – 31.05.2019)

3 Мэтт Вайсфельд. Объектно-ориентированное мышление – Санкт-Петербург: Питер, 2014. – 304 с.

4 Spring Framework Documentation [сайт] – URL: https://docs.spring.io/spring/docs/current/spring-framework-reference/ (дата обращения: 20.05 – 31.05.2019)

Приложение А

Приложение В

№	Описание	Предварительные условия	Шаги	Ожидаемый результат	Вид тестирования
1	Запуск приложения	-	Нажать на кионку приложения	Приложение запустилось без ошибок Отобразилась страница авторизации	Smoke
2	Ввод корректных данных для поиска билетов в обе стороны	Открыта страница с формой для ввода информации поиска билетов Наличие подключения к интернету	1. Ввести город отправления 2. Ввести город направления 3. Ввести дату отправления 4. Ввести дату возвращения 5. Ввод количества пассажиров 6. Выбор класса обслуживания 7. Выбор с пересадками или без 8. Нажать кнопку "поиск билетов"	Найдены билеты в обе стороны Отобразилась страница с о списком результатов	Smoke
3	Ввод корректных данных для поиска билетов в одну стороны	Открыта страница с формой для ввода информации поиска билетов Наличие подключения к интернету	1. Ввести город отправления 2. Ввести город направления 3. Ввести дату отправления 4. Ввод количества пассажиров 5. Выбор класса обслуживания 6. Выбор с пересадками или без 7. Нажать кнопку "поиск билетов"	Найдены билеты в одну стороны Отобразилась страница с о списком результатов	Smoke
4	Ввод несуществующего города	аналогично пункту 2	аналогично пункту 2 или 3	Поиск билетов не осущетсвляется Выводиться сообщение "Город не найден"	Sanity
5	Введена неверная дата	аналогично пункту 2	аналогично пункту 2 или 3	Поиск билетов не осущетсвляется Выводиться сообщение "Неверная дата"	Sanity
6	Поиск без подключения к интернету	Открыта страница с формой для ввода информации поиска билетов Подключения к интернету отсутствует	аналогично пункту 2 или 3	Поиск билетов не осущетсвляется Выводиться сообщение "Нет ответа"	Sanity
7	Сохранение истории	Осуществлен поиск билетов пройдена авторизация		Введенная пользователем информация сохранилась	Smoke
8	Условия сохранения истоии поиска	Приложение запущено пройдена авторизация	В меню в верхней части приложения отметить галочкой пункт "Сохранять историю поиска" В меню в верхней части приложения снять галочку с пункта пункт "Сохранять историю поиска"	Введенная пользователем информация не сохраняется Введенная пользователем информация сохраняется	Smoke
9	Сортировки результатов поиска	открыта страница с результатами поиска билетов	В врехней части экрана выбрать тип сортировки	Все типы сортировок работают успешно	Smoke
10	Добавление закладки	открыта страница с результатами поиска билетов пройдена авторизация	В верхней части экрана нажать на сердечко	Закладка успешно добавлена в списиок закладок Сердечко стало закрашенным	Smoke
11	Удаление закладки	пройдена авторизация открыта страница с результатами поиска билетов	В верхней части экрана нажать на закрашенное сердечко	Сердечко стало не закрашенным Закладка успешно удалена	Smoke
		открыта страница со списком закладок пройдена авторизация	В правой части закладки нажать на сердечко	Закладка успешно удалена	Smoke
12	Удаление элемента истории поиска	открыта страница с историей поиска пройдена авторизация	В правой части закладки нажать на крестик	Элемент истории поиска успешно удален	Smoke
13	Очищение истории поиска	открыта страница с историей поиска пройдена авторизация	В нижней части страницы нажать на кнопку "Очистить историю"	История поиска успешно очищена	Smoke
14	Ссылки на покупку билетов	открыта страница с подробным описанием билета	В разделе "Цены" нажать на кнопку "Купить"	Открывается страница в браузере с формой покупки билетов, которая заполнена нужными данными	Smoke
15	Подсказки пользователю	Приложение запущено	Нажать на какую-либо иконку	Всплывает подсказка о том, что означает иконка	Smoke
16	Авторизоваться	Приложение запущено	Нажать на кнопку войти Выбрать аккаунт	авторизация успешна	Smoke
17	Выйти из системы	Приложение запущено пройдена авторизация	В меню в верхней части экрана выбрать опцию "Выйти из системы"	Выход из системы успешен	Smoke
18	Испозьзовать приложение без авторизации	открыт экран авторизации	Нажать на кнопку отмена	Доступен только поиск билетов	Smoke

Рисунок В.1 – Тестовые сценарии

№	Pixel 3 XL API 28 Android 9.0		Pixel 2 API 27 Android 8.1		Pixel 2 XL API 26 Android 8.0		Pixel API 25 Android 7.1	
	Фактический результат	Статус	Фактический результат	Статус	Фактический результат	Статус	Фактический результат	Статус
1	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
2	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
	соответсвтует ожидаемому		соответсвтует ожидаемому		соответсвтует ожидаемому		соответсвтует ожидаемому	
3	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
	соответсвтует ожидаемому		соответсвтует ожидаемому		соответсвтует ожидаемому		соответсвтует ожидаемому	
4	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
5	При вводе несуществующей даты ищутся билеты на дату равную последнее число месяца + лишние дни (напрмиер, для 32 мая 2019 найдены билеты для 1 июня 2019)	Не пройден	При вводе несуществующей даты ищутся билеты на дату равную последнее число месяца + лишние дни (напрмиер, для 32 мая 2019 найдены билеты для 1 июня 2019)	Не пройден	При вводе несуществующей даты ищутся билеты на дату равную последнее число месяца + лишние дни (напрмиер, для 32 мая 2019 найдены билеты для 1 июня 2019)	Не пройден	При вводе несуществующей даты ищутся билеты на дату равную последнее число месяца + лишние дни (напрмиер, для 32 мая 2019 найдены билеты для 1 июня 2019)	Не пройден
6	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
7	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
8	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
9	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
10	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
11	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
12	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
13	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
14	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
15	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
16	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
17	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
18	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
19	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
20	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
21	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден

Рисунок В.2 – Тестирование устройств (1)

№	Nexus 5X API 24 Android 7.0		Nexus 9 API 23 Android 6.0		Nexus 7 API 22 Android 5.1		Nexus 5 API 21 Android 5.0	
	Фактический результат	Статус	Фактический результат	Статус	Фактический результат	Статус	Фактический результат	Статус
1	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
2	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
	соответсвтует ожидаемому		соответсвтует ожидаемому		соответсвтует ожидаемому		соответсвтует ожидаемому	
3	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
	соответсвтует ожидаемому		соответсвтует ожидаемому		соответсвтует ожидаемому		соответсвтует ожидаемому	
4	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
5	При вводе несуществующей даты ищутся билеты на дату равную последнее число месяца + лишние дни (напрмиер, для 32 мая 2019 найдены билеты для 1 июня 2019)	Не пройден	При вводе несуществующей даты ищутся билеты на дату равную последнее число месяца + лишние дни (напрмиер, для 32 мая 2019 найдены билеты для 1 июня 2019)	Не пройден	При вводе несуществующей даты ищутся билеты на дату равную последнее число месяца + лишние дни (напрмиер, для 32 мая 2019 найдены билеты для 1 июня 2019)	Не пройден	При вводе несуществующей даты ищутся билеты на дату равную последнее число месяца + лишние дни (напрмиер, для 32 мая 2019 найдены билеты для 1 июня 2019)	Не пройден
6	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
7	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
8	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
9	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
10	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
11	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
12	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
13	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
14	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
15	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
16	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
17	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
18	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
19	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
20	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден
21	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден	соответсвтует ожидаемому	Пройден

Рисунок В.3 – Тестирование устройств (2)