МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ОБРАЗОВАНИЯ**

**“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

Факультет *компьютерных наук*

Кафедра *программирования и информационных технологий*

*Приложение по подбору авиабилетов*

*Курсовая работа*

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\_\_

Обучающиеся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*А.Е. Копылова, О.А. Транина 3 курс, д/о*

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*, ст. преподаватель*

Воронеж 2018

# Содержание

[Содержание 2](#_Toc8308442)

[Введение 4](#_Toc8308443)

[1. Постановка задачи 5](#_Toc8308444)

[2. Анализ 7](#_Toc8308445)

[2.1. Анализ предметной области 7](#_Toc8308446)

[2.2. Анализ задач 9](#_Toc8308447)

[2.2.1. Задача поиска авиабилетов 9](#_Toc8308448)

[2.2.2. Задача хранения данных пользователя, приложения и его настроек 9](#_Toc8308449)

[2.2.2.1. Подзадача хранения закладок 10](#_Toc8308450)

[2.2.2.2. Подзадача хранения истории поиска 10](#_Toc8308451)

[2.2.3. Задача отображения пользовательского интерфейса 11](#_Toc8308452)

[2.3. Анализ средств реализации 12](#_Toc8308453)

[2.4. Диаграммы 12](#_Toc8308454)

[3. Реализация 21](#_Toc8308455)

[3.1. Задача поиска авиабилетов 21](#_Toc8308456)

[3.2. Задача хранения данных пользователя, приложения и его настроек 21](#_Toc8308457)

[3.2.1. Подзадача хранения закладок 21](#_Toc8308458)

[3.2.2. Подзадача хранения истории поиска 22](#_Toc8308459)

[3.3. Задача отображения пользовательского интерфейса 24](#_Toc8308460)

[Заключение 24](#_Toc8308461)

[Скелет приложения 25](#_Toc8308462)

[Подсистема поиска авиабилетов 25](#_Toc8308463)

[Подсистема хранения истории поиска (класс SearchHistoryRepository) 31](#_Toc8308464)

[Подсистема хранения закладок 31](#_Toc8308465)

[Подсистема отображения пользовательских интерфейсов. 32](#_Toc8308466)

[Вид пользовательского интерфейса 48](#_Toc8308467)

[Список источников 54](#_Toc8308468)

[Приложения 54](#_Toc8308469)

# 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.

# Постановка задачи

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

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

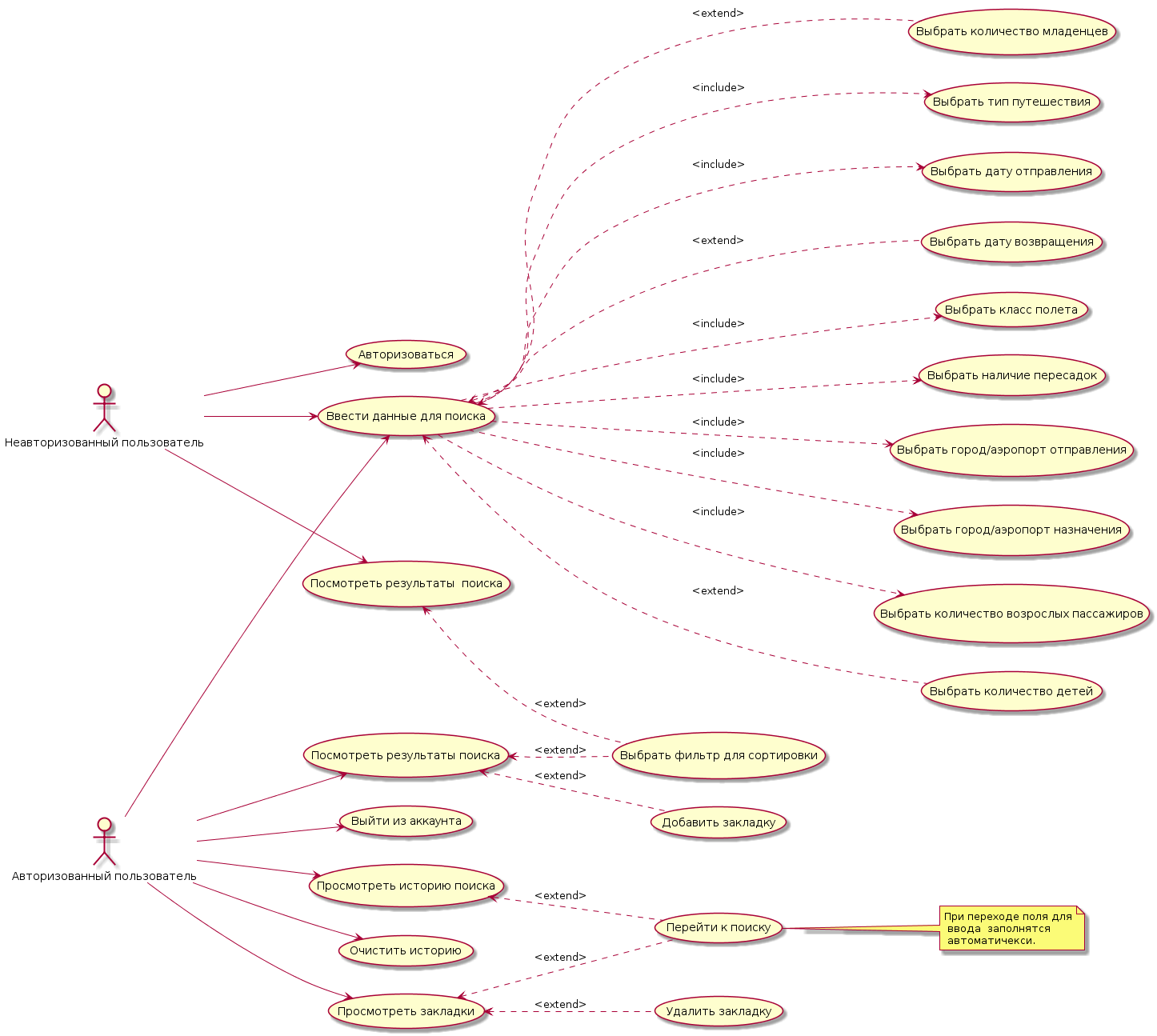


Рисунок 1 - Диаграмма прецедентов

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), наличии пересадок;
* поиск авиабилетов на рейсы в оба конца с возможностью добавления информации о количестве пассажиров, типе путешествия (в один или оба конца), 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 (API).
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.

# Анализ

## 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).

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.

## Анализ задач

### 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.

### 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.

#### 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.

#### 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.

### 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).

## 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 данные о пользователе было решено хранить в базе данных на сервере.

Для написания логики работы системы был выбран высокоуровневый язык программирования Java, так как

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.

## 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).

### 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.

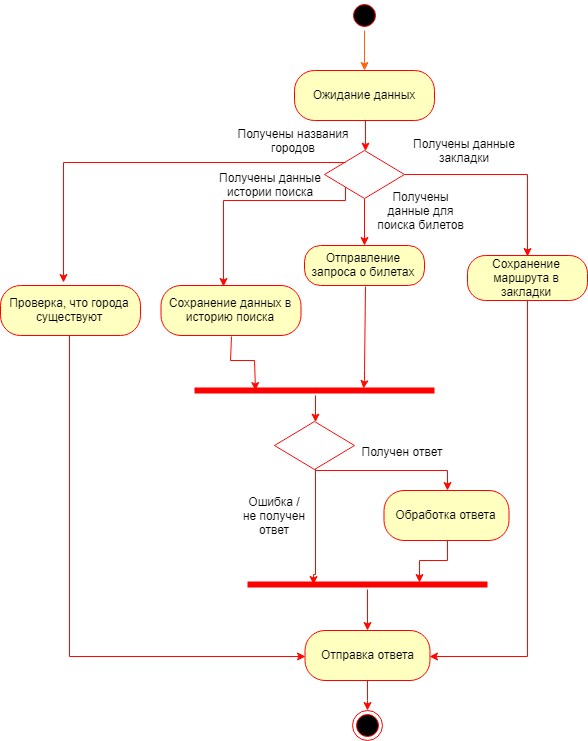


Рисунок 2 – Диаграмма состояний сервера основного сценария

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

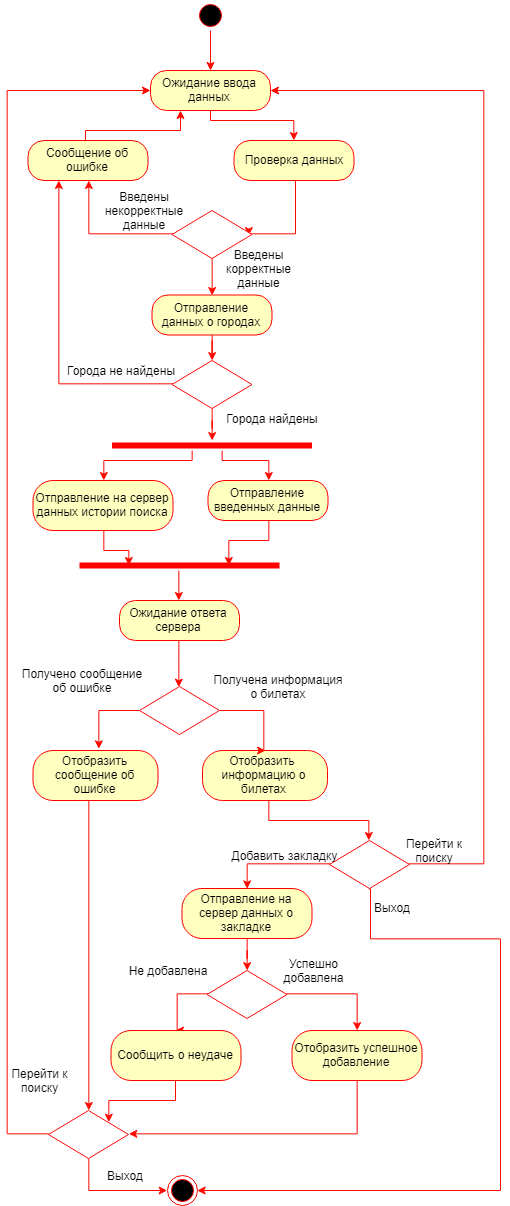


Рисунок 3 – диаграмма состояний приложения при основном сценарии

Диаграмма состояний пользователя при основном сценарии представлена в приложении А.

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

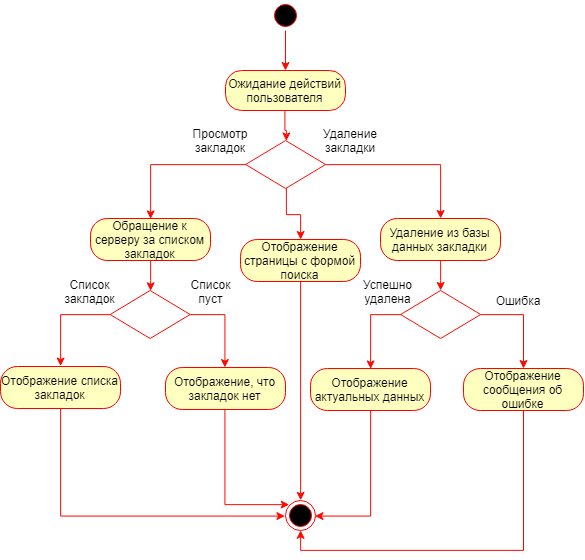


Рисунок 4 – Диаграмма состояний приложения (закладки)

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

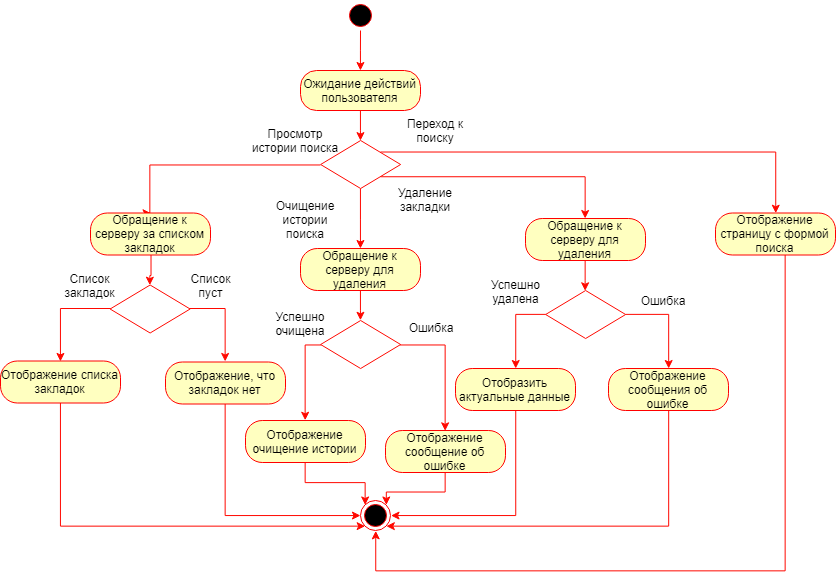


Рисунок 5 – Диаграмма состояний приложения (история поиска)

### 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).

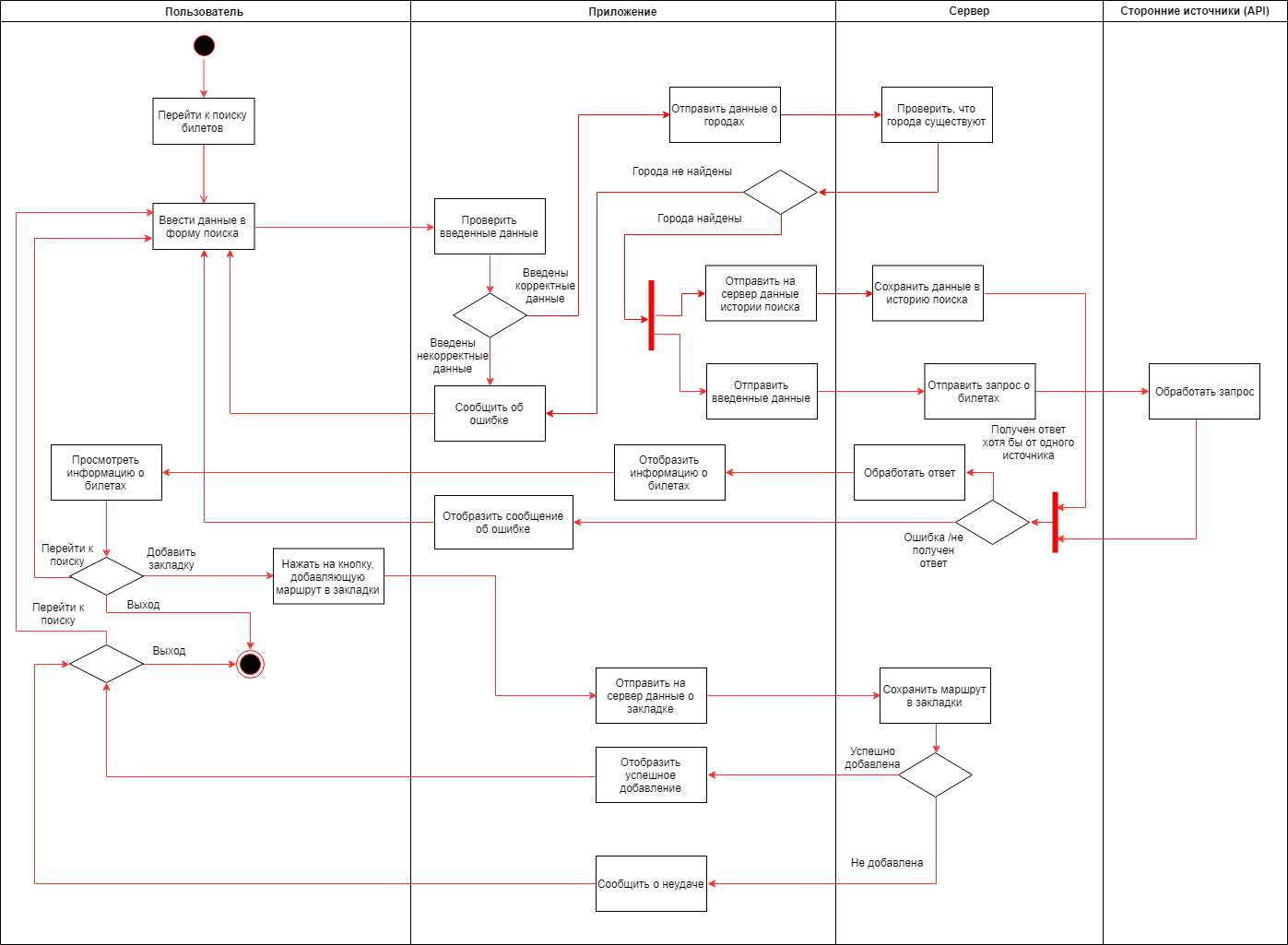


Рисунок 6 – Диаграмма активности основного сценария

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

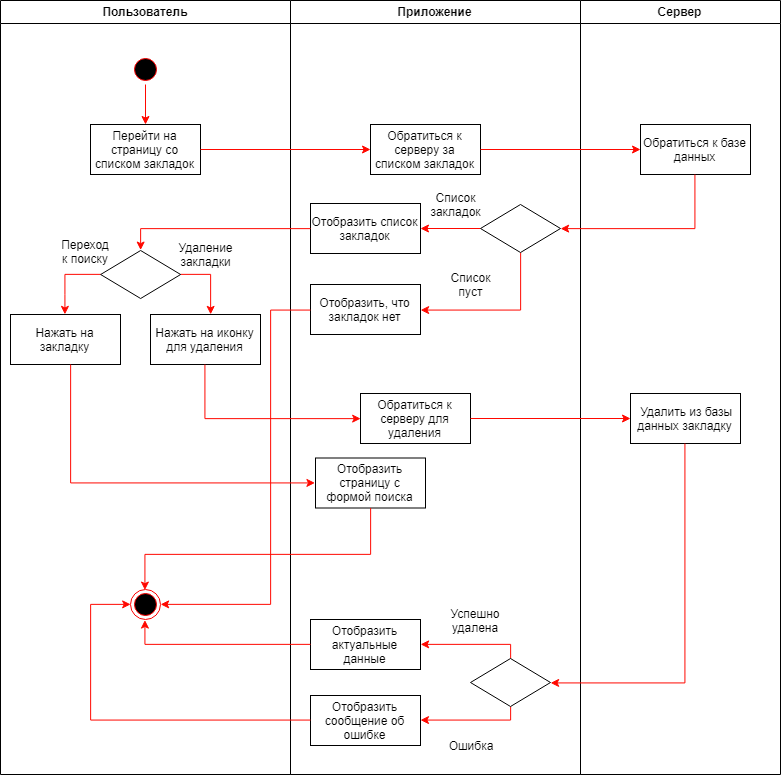


Рисунок 7 – Диаграмма активности (закладки)

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

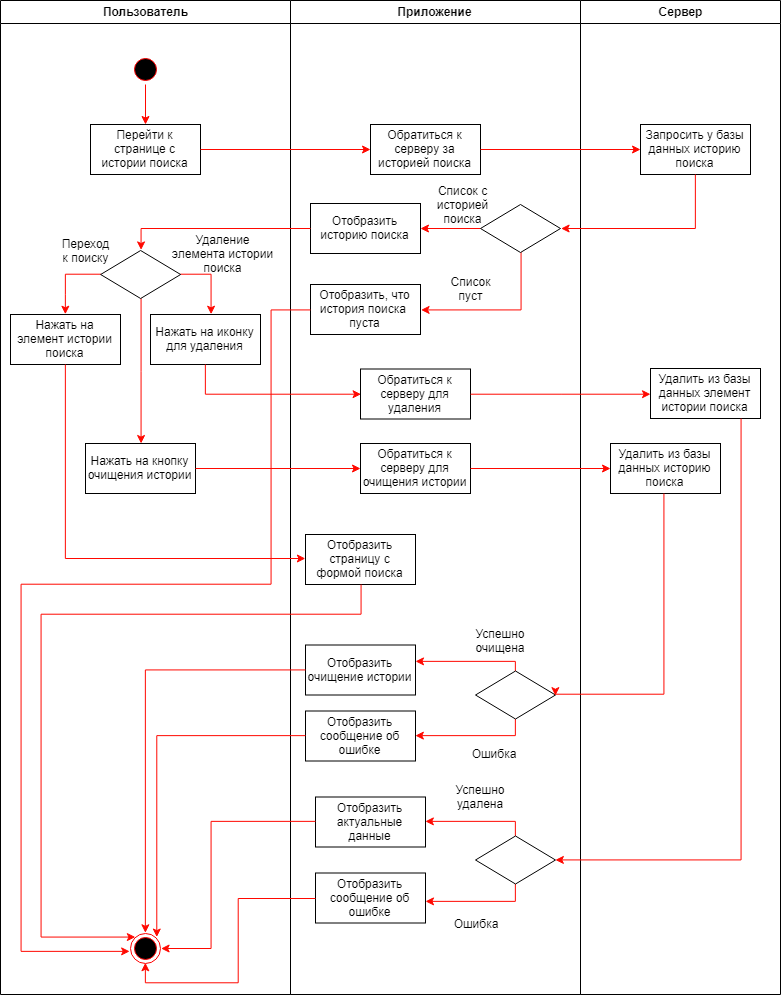


Рисунок 8 – Диаграмма активности (история поиска)

### Диаграммы последовательностей

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

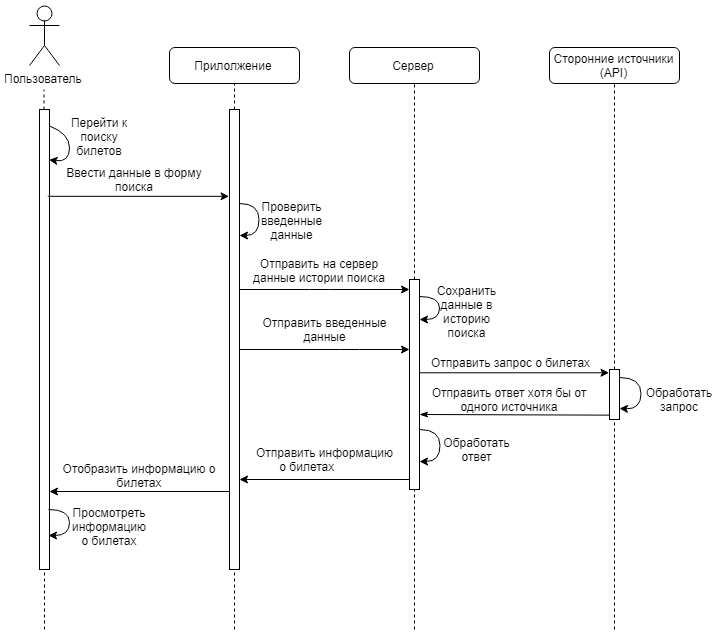


Рисунок 9 – Диаграмма последовательностей основного сценария (поиска билетов и отображение результатов)

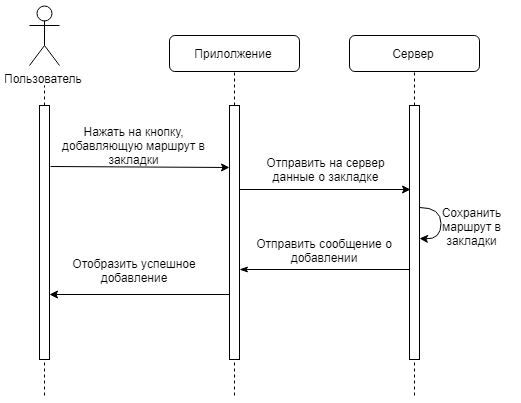


Рисунок 10 – Диаграмма последовательностей основного сценария (добавление маршрута в закладки)

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

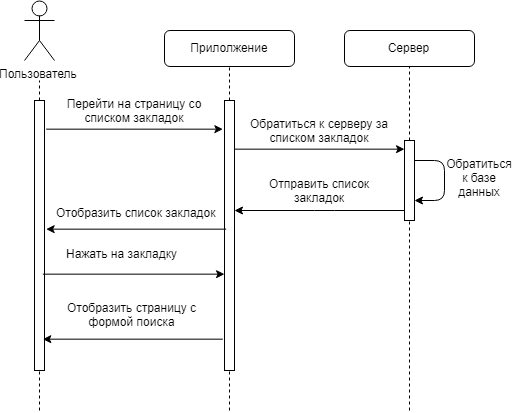


Рисунок 11 – Диаграмма взаимодействия (переход на форму поиска)

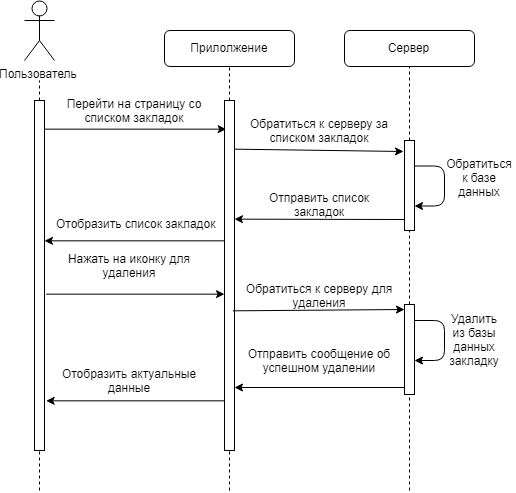


Рисунок 12 – Диаграмма взаимодействия (удаление закладки)

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

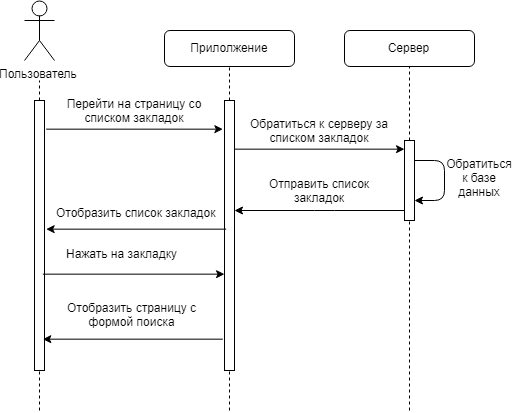


Рисунок 13 – Диаграмма последовательностей (переход к поиску)

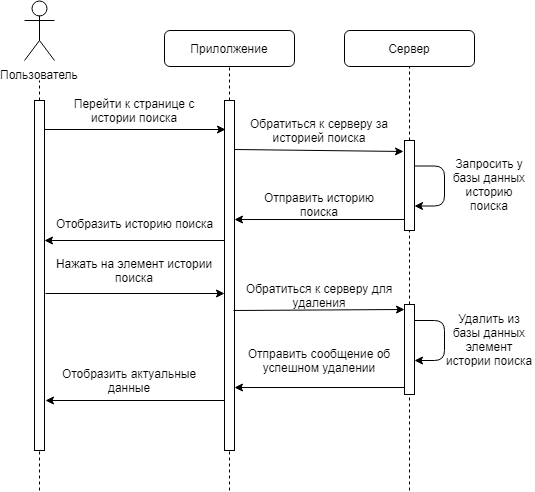


Рисунок 14 – Диаграмма последовательностей (удаление элемента истории поиска)

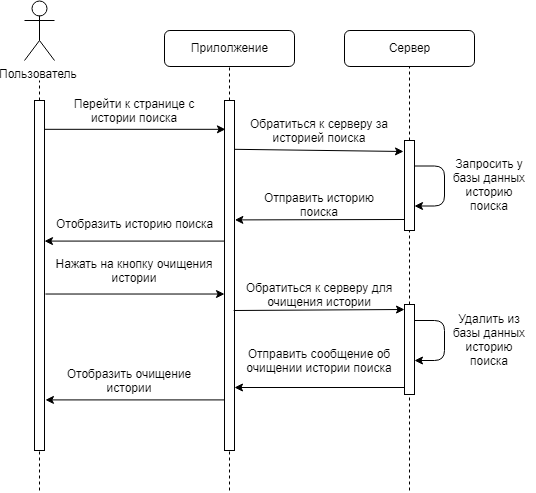


Рисунок 15 – Диаграмма последовательностей (очищение истории поиска)

### Диаграммы коммуникаций

В данном разделе приведены диаграммы взаимодействия между компонентами системы.

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

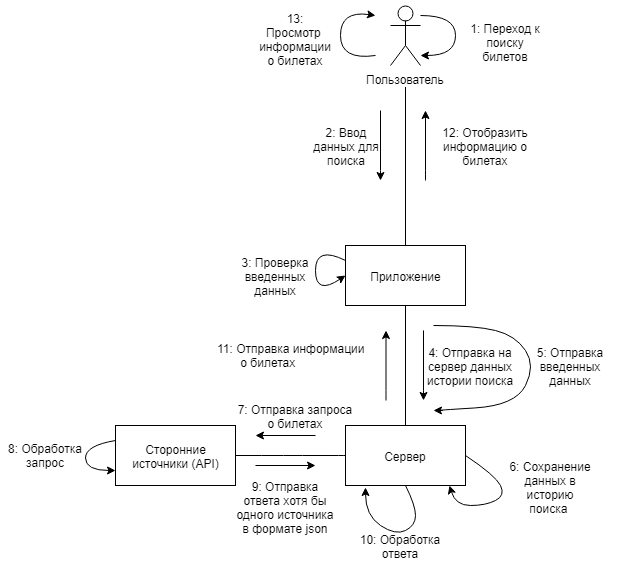


Рисунок 16 – Диаграмма коммуникаций подбора билетов

Диаграмма коммуникаций добавления маршрута в закладки представлена на рисунке 17.

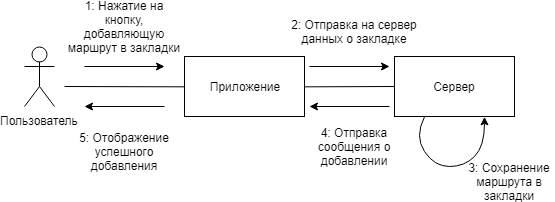


Рисунок 17 – Диаграмма коммуникаций добавления в закладки

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

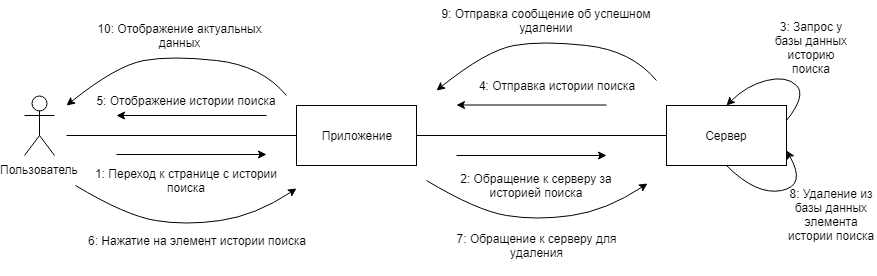


Рисунок 18 – Диаграмма коммуникаций удаления элемента истории

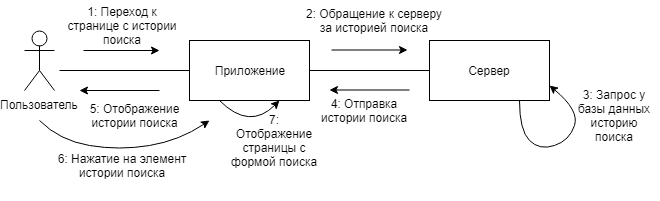


Рисунок 19 – Диаграмма коммуникаций перехода на форму поиска



Рисунок 20 – Диаграмма коммуникаций очищения истории

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

### Диаграмма развертывания

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

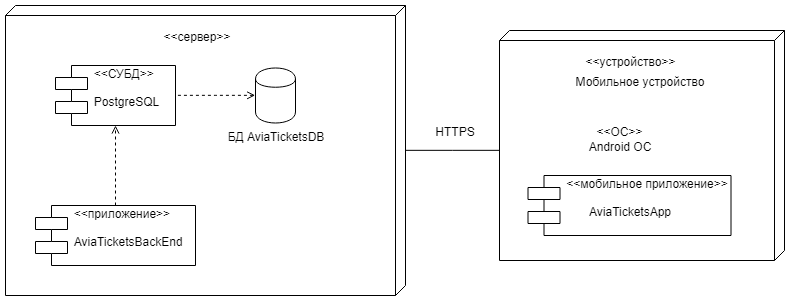


Рисунок 21 – Диаграмма развертывания

# Реализация

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

## 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 возращения 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).

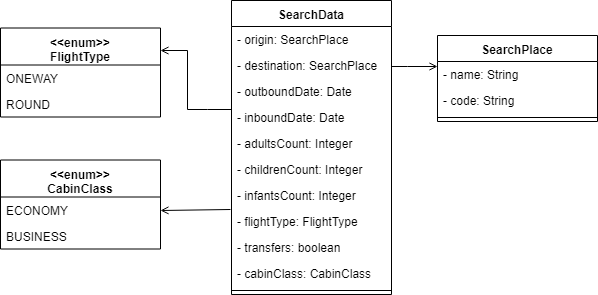


Рисунок 22 – Диаграмма классов входных данных

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

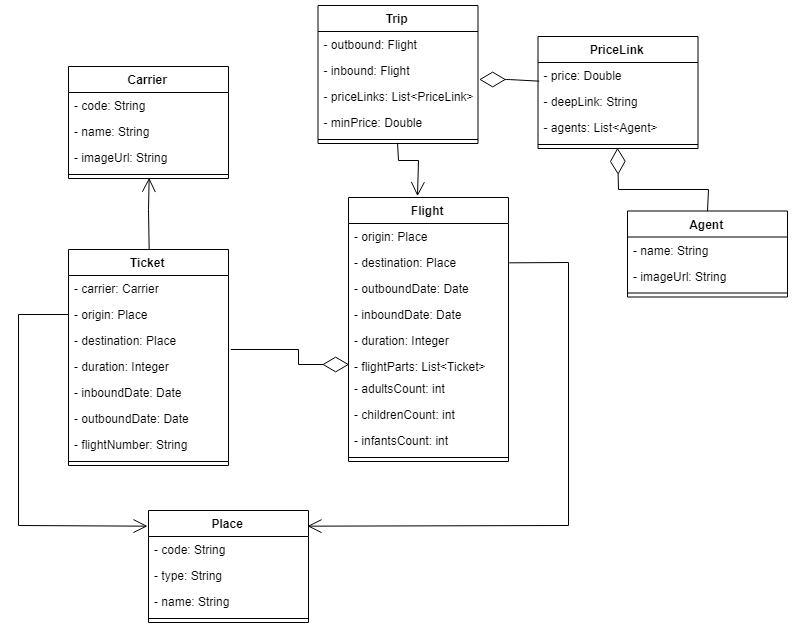


Рисунок 23 – Диаграмма классов выходных данных

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 возращения 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.

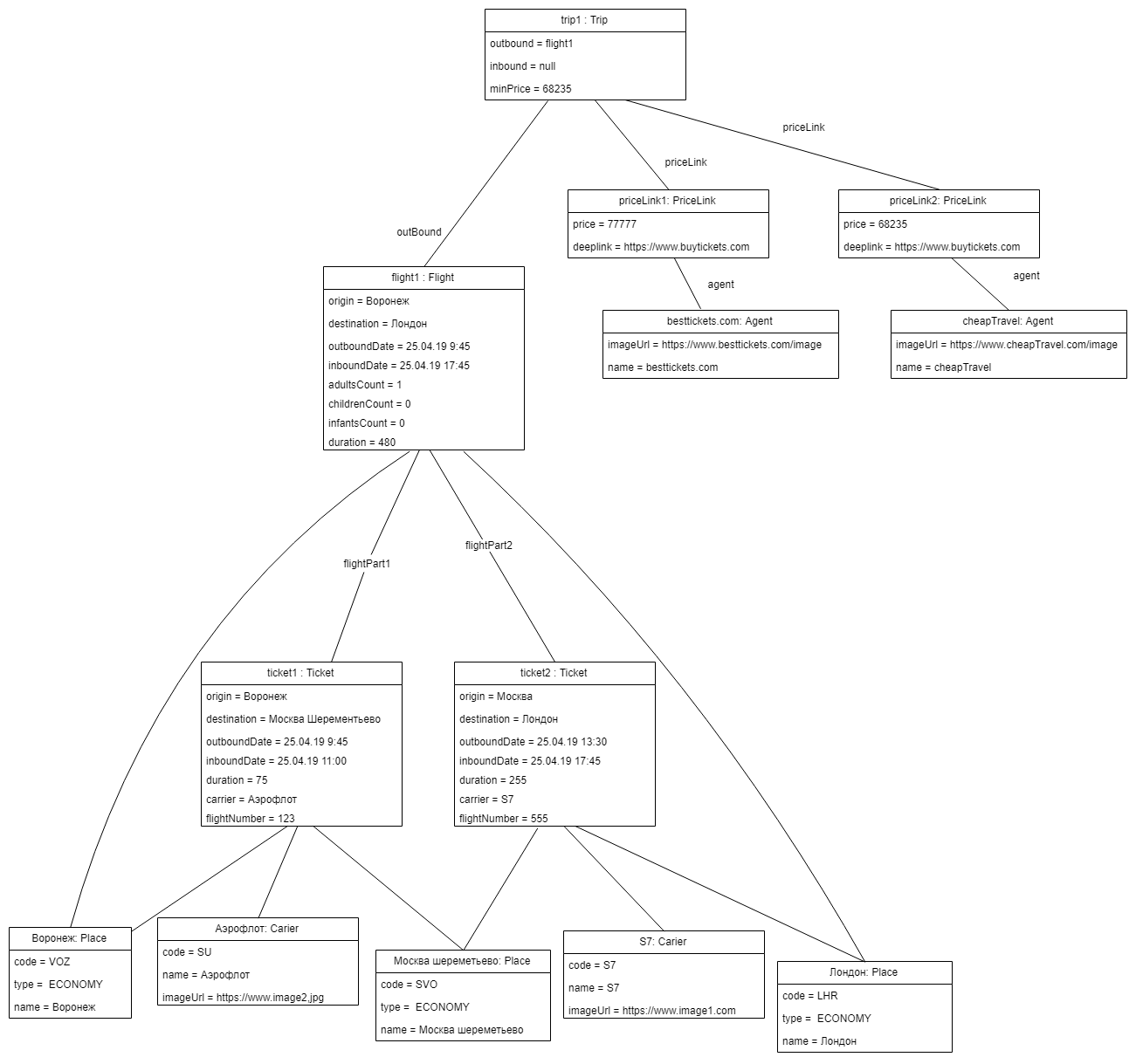


Рисунок 24 – Диаграмма объектов

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.

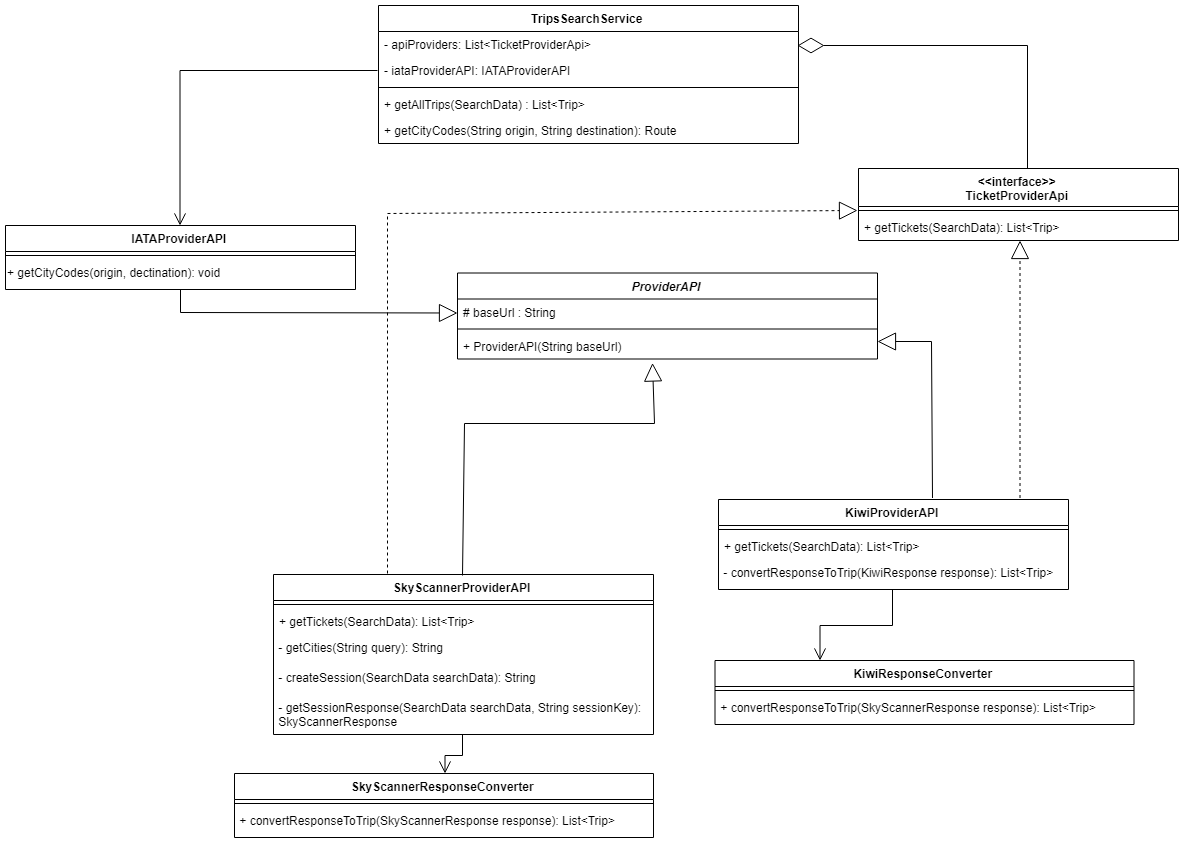


Рисунок 25 – Диаграмма классов общения с API-источниками

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 код города по его названию.

## Задача хранения данных пользователя и приложения

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

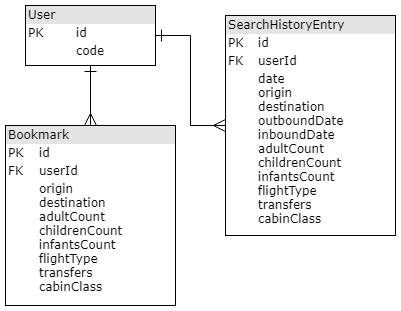


Рисунок 26 – Схема базы данных

Таблица

### Подзадача хранения закладок

Для хранения закладок пользователя выбрана база данных PostgreSQL. Все закладки хранятся в таблице, которая имеет структуру, изображенную на рисунке \*.

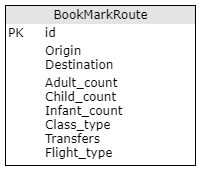


Рисунок \*. Таблица закладок.

Для связи с базой данных используется библиотека Room, для чего была использована организация классов, изображенная рисунке \*.

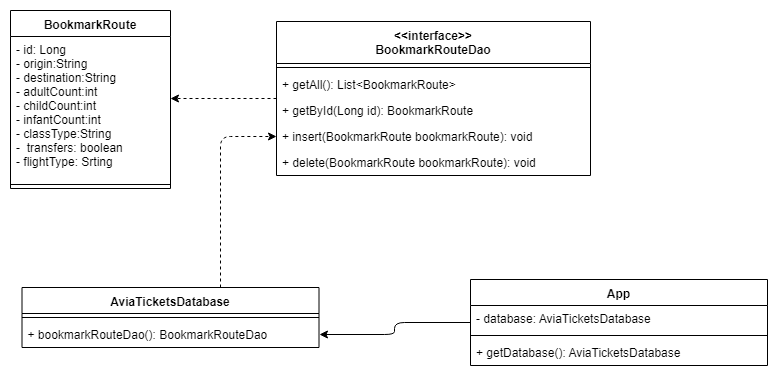


Рисунок \*. Диаграмма классов общения с базой данных.

Класс BookmarkRoute необходим для проекции строки базы данных.

Класс AviaTicketsDatabase является наследником класс RoomDatabase и необходим для работы с BookmarkRouteDao.

Интерфейс BookmarkRouteDao необходим для манипулирования данными, а именно:

* метод getAll() возвращает все закладки, содержащиеся в базе данных на момент вызова;
* метод getById(Long id) возвращает закладку, id которой равен полученному;
* метод insert (BookmarkRoute bookmarkRoute) добавляет в базу данных полученную закладку;
* метод delete (BookmarkRoute bookmarkRoute) удаляет из базы данных полученную закладку.

Класс App необходим для реализации паттерна одиночка, которая обеспечивает существование только одного экземпляра класса AviaTicketsDatabase на все приложение.

### Подзадача хранения истории поиска

Для хранения истории поиска и пользовательских настроек, позволяющих не сохранять историю поиска, было использовано постоянное хранилище данных Android SharedPreferences.

Для реализации данной подзадачи создан класс SearchHistoryRepository, изображенные на рисунке \*. В данном классе находятся строковые константы:

* PREFS\_NAME – имя, по которому можно обратиться к части памяти, в которой хранится история поиска;
* SEARCH\_DATA\_KEY – ???

Также данный класс содержит методы, обеспечивающие функциональность, реализуемую в данной подзадаче:

* метод getAllSearchData() – возвращает все элементы, хранящиеся на момент вызова в списке элементов истории поиска;
* метод addSearchData(SerachData) – добавляет в список элементов истории поиска полученный элемент;
* метод RemoveSearchData(index) – удаляет из списка элементов истории поиска элемент по его индексу;
* метод RemoveAll() – очищает историю поиска;
* метод getSharedPreferences() – возвращает все элементы истории поиска, хранящиеся на момент вызова в SharedPreferences
* метод putToSharedPrefeernces() – добавляет в SharedPreferences, список элементов истории поиска;
* метод … - сохраняет настройки приложения (хранить историю или нет).

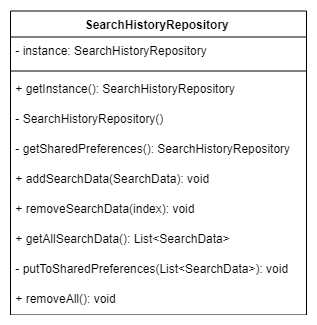


Рисунок \*. Диаграмма классов хранения истории.

## 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 \* изображена часть диаграммы классов, с помощью которых и 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.

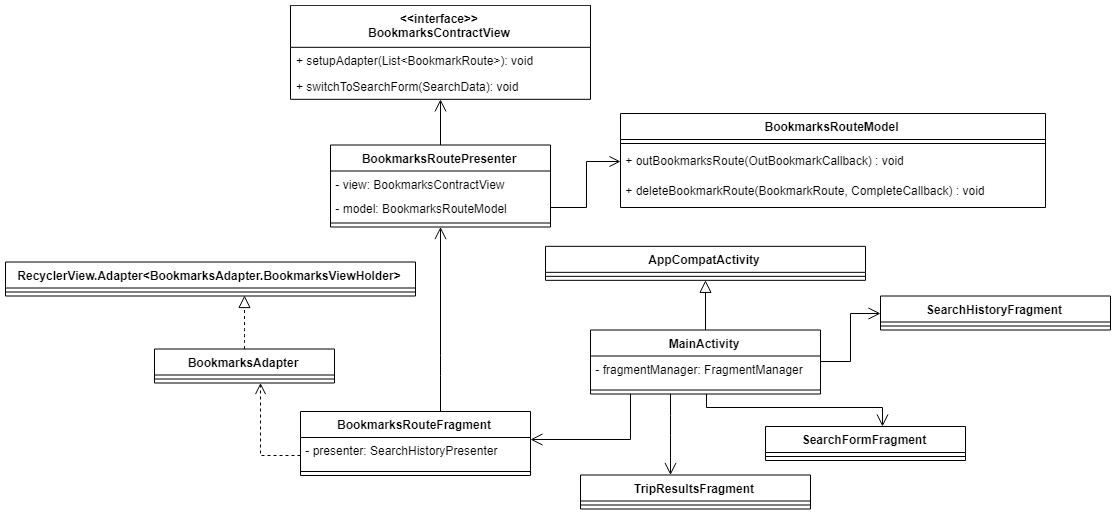


Рисунок \* - Диаграмма классов отображения пользовательского интерфейса

Для введения архитектуры MVP введены классы SearchHistoryModel, SearchHistoryPresenter, SearchHistoryFragment, а также интерфейс SearchHistoryContractView, который обеспечивает связь между двумя последними.

Класс SearchHistoryModel содержит в себе поля и методы, осуществляющие вызов методов работы с SharedPreferences, а именно:

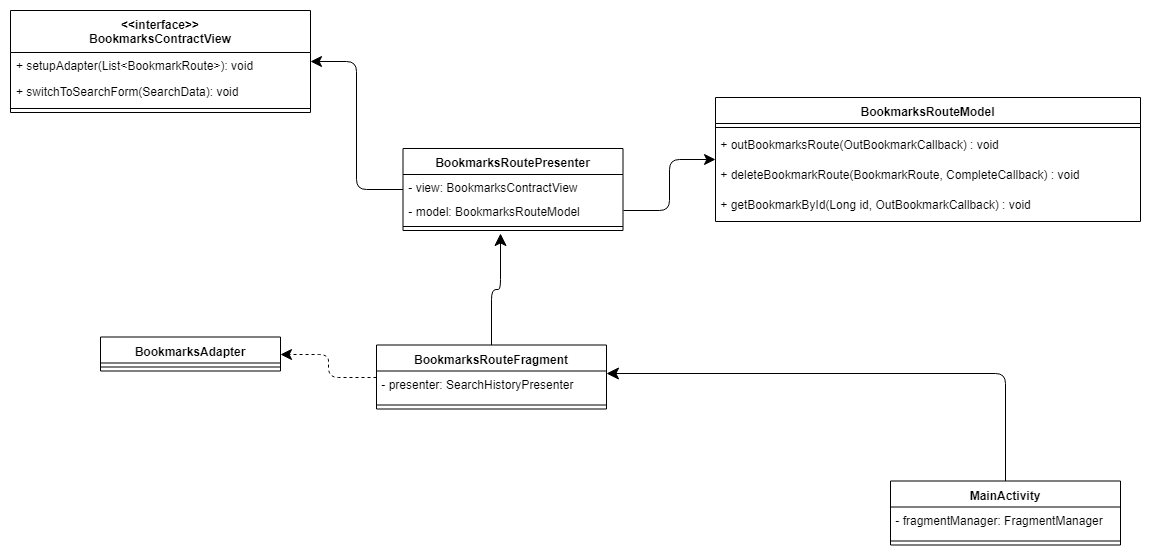
* List<SearchData> getAll() возвращает весь список истории поиска.

История поиска отображается в виде списка элементов, реализованного в виде RecyclerView, для чего необходим класс SearchHistoryAdapter, имеющий следующие поля и методы:

Класс SearchHistoryFragment необходим для отображения пользовательского интерфейса, здесь находятся поля и методы:

### Закладки

Для отображения страницы с закладками и обеспечения ее работы была использована архитектура, изображенная в виде диаграммы классов на рисунке \*.



Рисунок\* - Диаграмма классов отображения страницы с закладками

В классе BookmarksRouteModel содержаться методы, осуществляющие работу с базой данных, а именно:

* + - outBookmarksRoute(OutBookmarkCallback callback) – метод, отвечающий за вывод всех закладок, содержащихся в базе данных на момент вызова;
    - deleteBookmarkRoute(BookmarkRoute value, CompleteCallback callback) – метод, отвечающий за удаление закладки.

Здесь OutBookmarkCallback и CompleteCallback – это интерфейсы, обеспечивающие взаимосвязь с асинхронными запросами.

По причине того, что взаимодействие с базой данных – это тяжеловесный процесс, который может сильно нагружать приложение и замедлять отображение пользовательского интерфейса, вводятся классы BookmarksRouteOut, BookmarksRouteDelete и BookmarksRouteGetById, которые являются наследниками базового класса AsyncTask. Таким образом, запросы к базе данных осуществляются в отдельном асинхронном потоке.

Закладки отображаются в виде списка элементов, реализованного в виде RecyclerView, для чего необходим класс BookMarkAdapter, имеющий поля и методы, аналогичные полям и методам класса SearchHistoryAdapter, описанном в пункте 3.3.4.

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

Класс BookmarksRouteFragment необходим для отображения пользовательского интерфейса, здесь находятся методы:

* + - метод onCreate(Bundle savedInstanceState)
    - метод View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
    - метод setAdapter(List<BookmarkRoute> bookmarkRoutes) назанчает адаптер для RecyclerView;
    - метод switchToSearchForm(SearchData searchData) осуществляет переход к форме поиска с заполнением полей информацией, сохраненной в закладке.

Для вызова методов связи с базой данных использован класс BookmarksRoutePresenter, содержащий в себе следующие поля и методы:

* статичное поле List<BookmarkRoute> bookmarkRoutes – список закладок;
* метод attachView(BookmarksRouteFragment activity)
* метод detachView()
* метод viewIsReady()
* метод loadBookmarks() выводит все закладки;
* метод delete(int index) удаляет закладку по ее индексу в списке всех закладок;
* метод itemChosen(SearchData searchData)

# 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 \* 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.

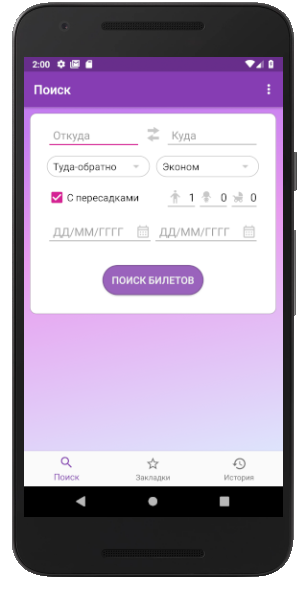
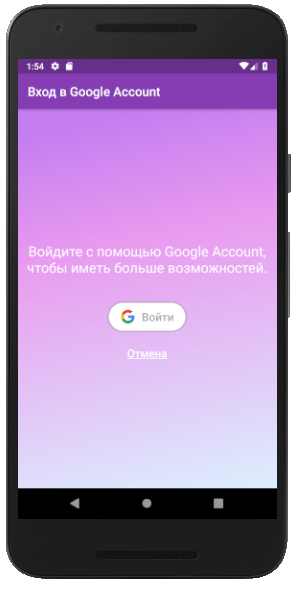
 

Рисунок \* – Страница поиска и авторизации

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 \* справа представлена страница авторизации через Google Account.

Для авторизованного пользователя страница с закладками, изображенная на рисунке \* 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.

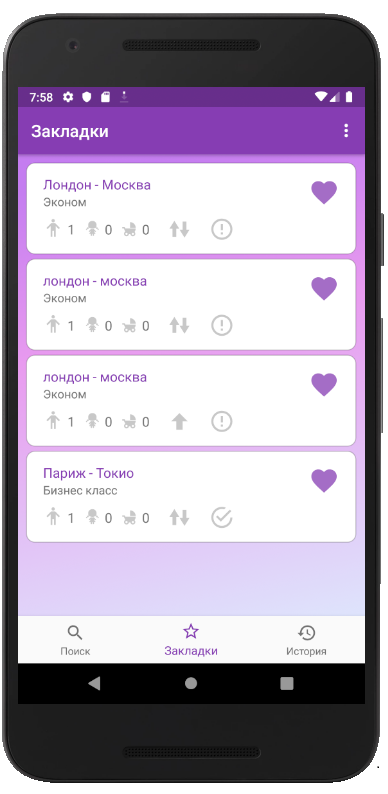
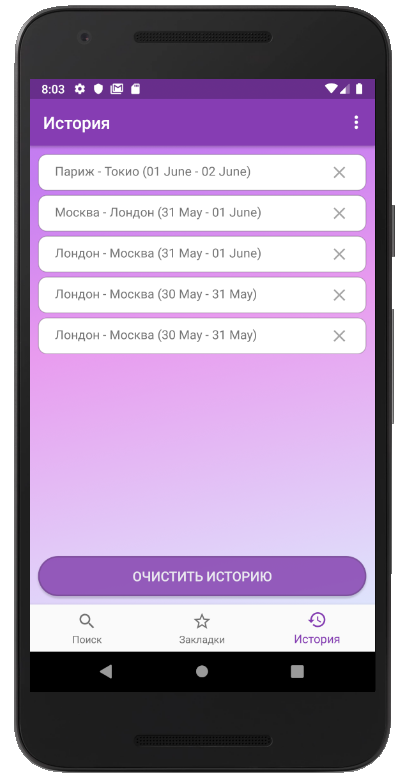
 

Рисунок \* – Страницы закладок и истории поиска

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

На рисунке \* 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\* справа и отражающая подробную информацию о билете и ссылки для его покупки.

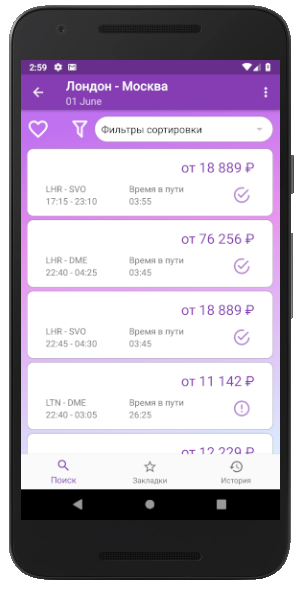
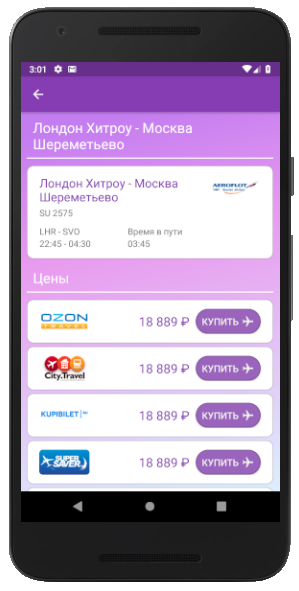
 

Рисунок \* – Результаты поиска

При нажатии на иконки всплывает подсказка.

# Тестирование

В целях проверки работоспособности системы в целом и подсистем приложения были проведены тесты следующего вида:

* модульное тестирование;
* интеграционное тестирование;
* smoke и sanity тесты на устройствах с версиями операционной системы Android 5 – 9 ручным методом и end-to-end тестирование;
* ui тесты.

## Модульное тестирование

Для проверки работоспособности функций подсистем было проведено автоматизированное модульное тестирование. Тест кейсы и результаты выполнения данного вида тестирования для серверной части системы представлены на рисунке \*.

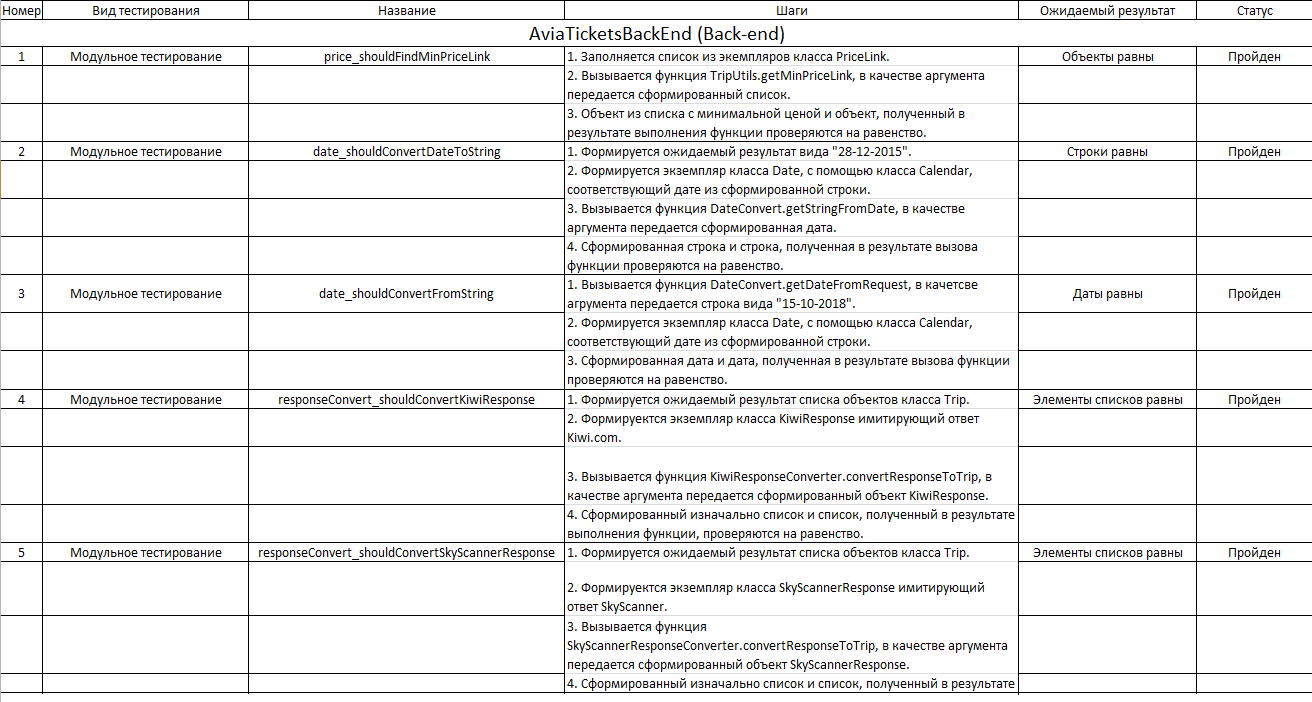


Рисунок \* – Модульное тестирование (сервер)

На рисунке \* приведены тест кейсы и результаты модульного тестирования клиентской части приложения. По результатам модульного тестирования можно сделать вывод, что приложение работает корректно как в серверной, так и в клиентской части.

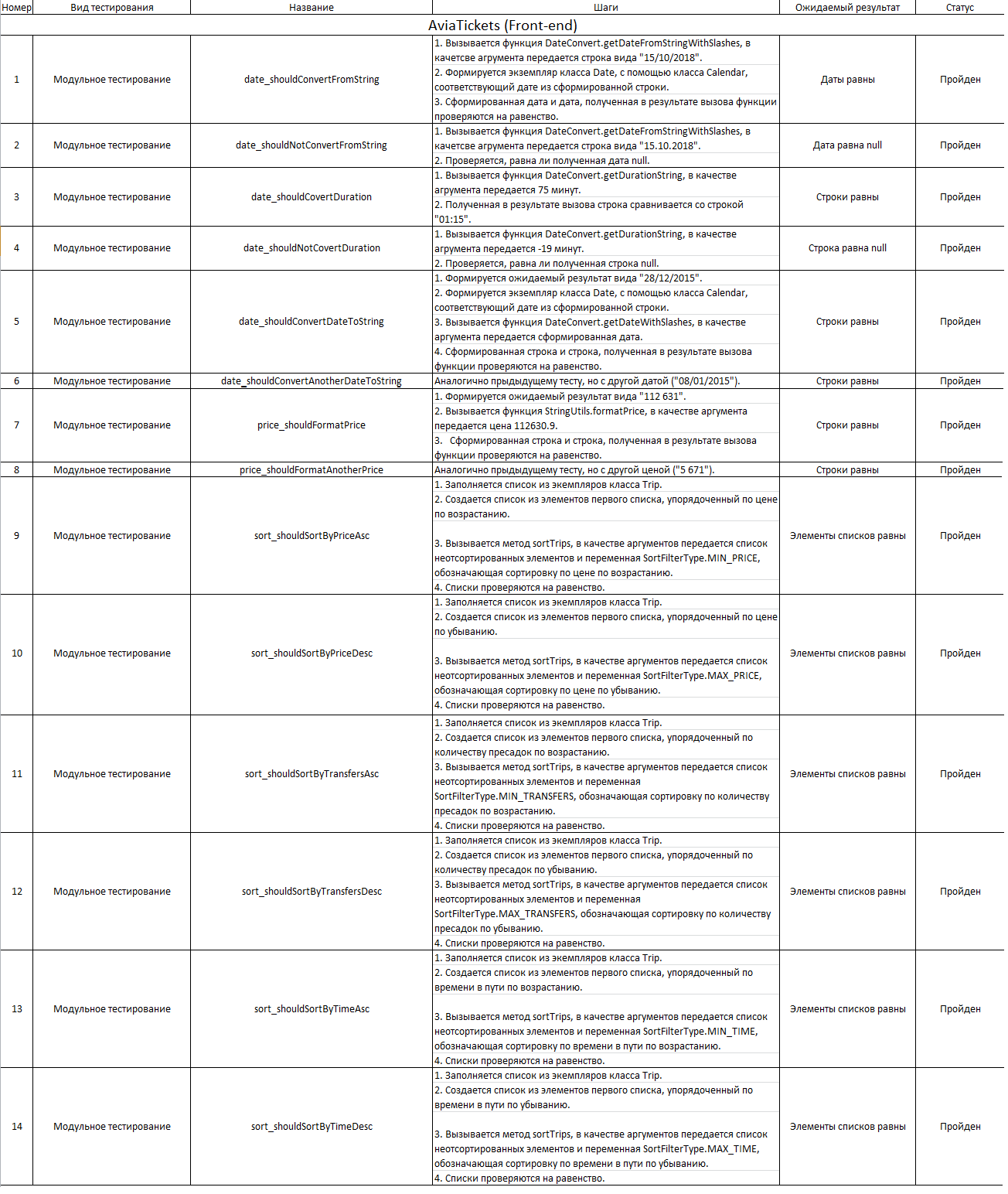


Рисунок \* – Модульное тестирование (клиент)

## Интеграционное тестирование

Для проверки работоспособности в целом и взаимосвязи между частями системы были проведены автоматизированные интеграционные тесты. Тест кейсы и результаты данного вида тестирования представлены на рисунке \*.



Рисунок \* – Интеграционное тестирование

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

## Тестирование ручным методом

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 тест кейсам, информация о которых содержится в приложении Bна рисунке B.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 B на рисунках B.2 и B.3. По ним можно сделать вывод, что система функционирует корректно на всех протестированных устройствах для всех тест кейсов кроме номера 5. При выполнении данного тест кейса 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. Тест кейсы и результаты представлены на рисунке \*. Как видно, тестирование прошло успешно.

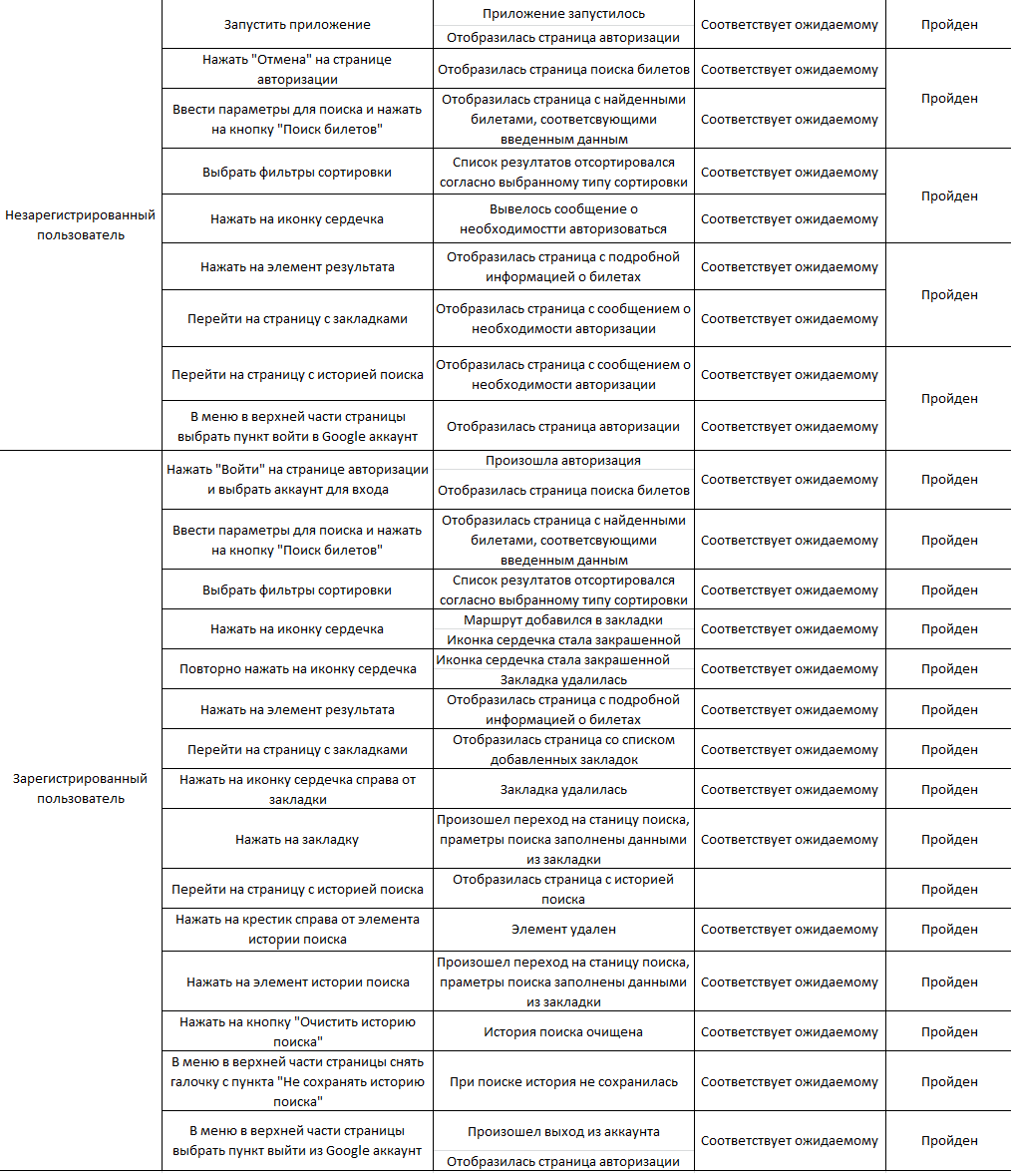


Рисунок \* – End-to-end тестирование

## UI-тесты

Для проверки корректной работы пользовательского интерфейса было проведено автоматизированное ui-тестирование, тест кейсы и результаты которого представлены на рисунке\*. Исходя из результатов, можно сделать вывод, что пользовательский интерфейс работает правильно.

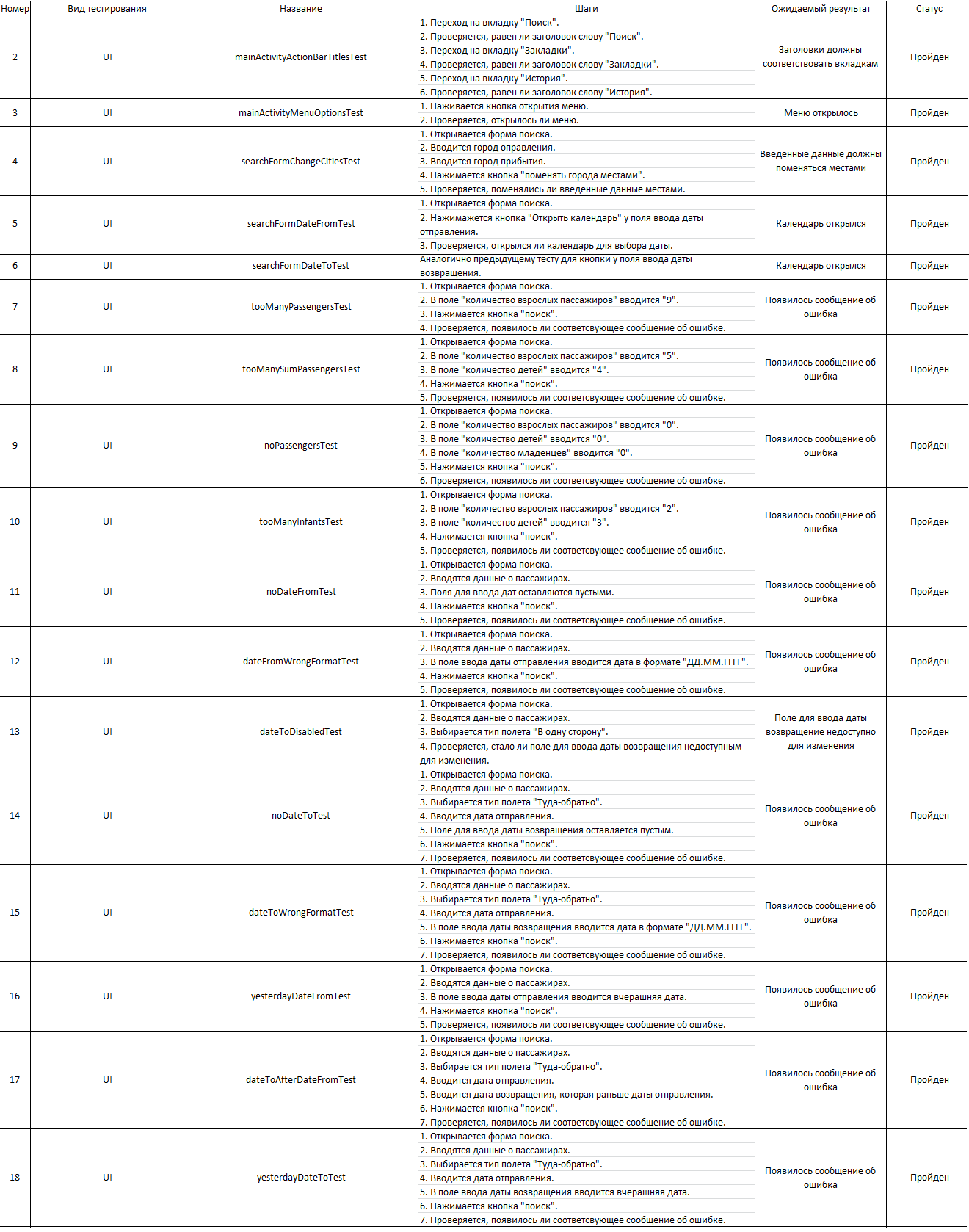


Рисунок \* – UI-тестирование

## 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), наличии пересадок;
* найти авиабилеты на рейсы в оба конца с возможностью добавления информации о количестве пассажиров, типе путешествия (в один или оба конца), 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.

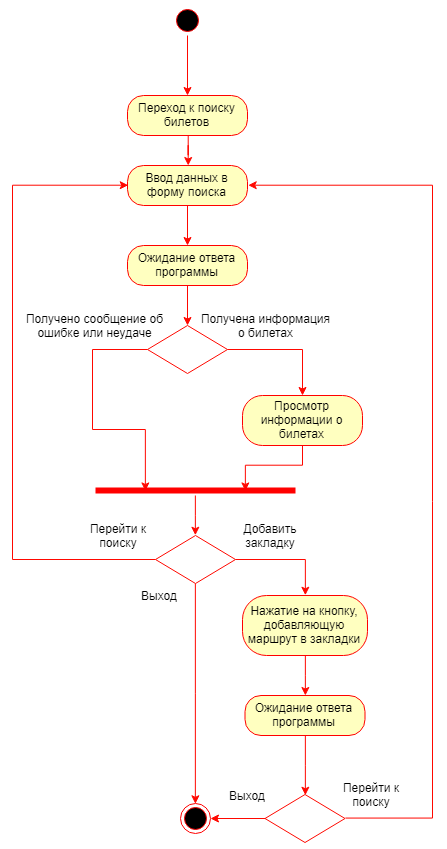
Для поиска авиабилетов приложение использует несколько сторонних источников данных(API). Также система подбирает не только прямые рейсы, но и с пересадками.

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 андроид-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 B

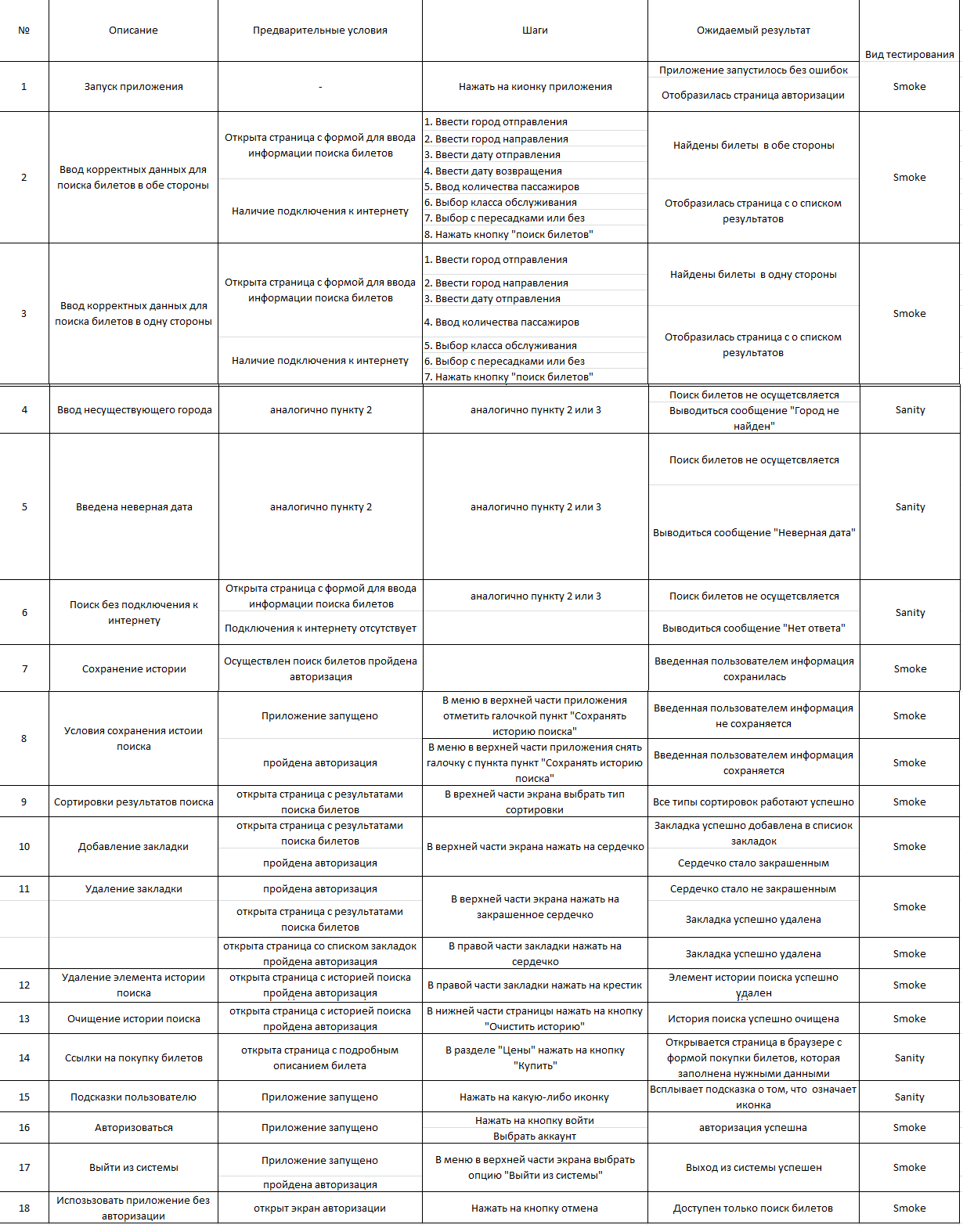


Рисунок B.1 – Тест кейсы

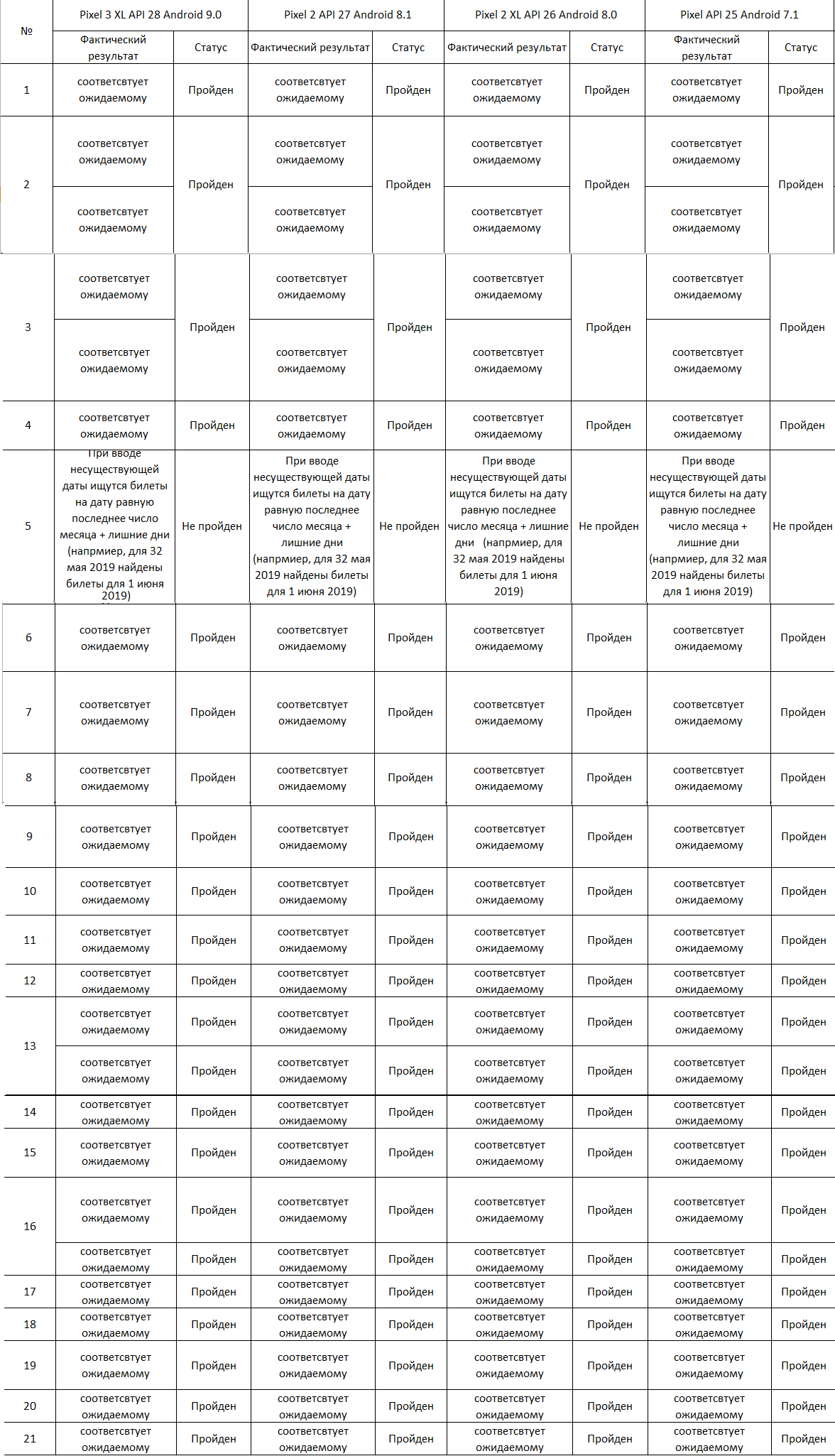


Рисунок B.2 – Тестирование устройств (1)

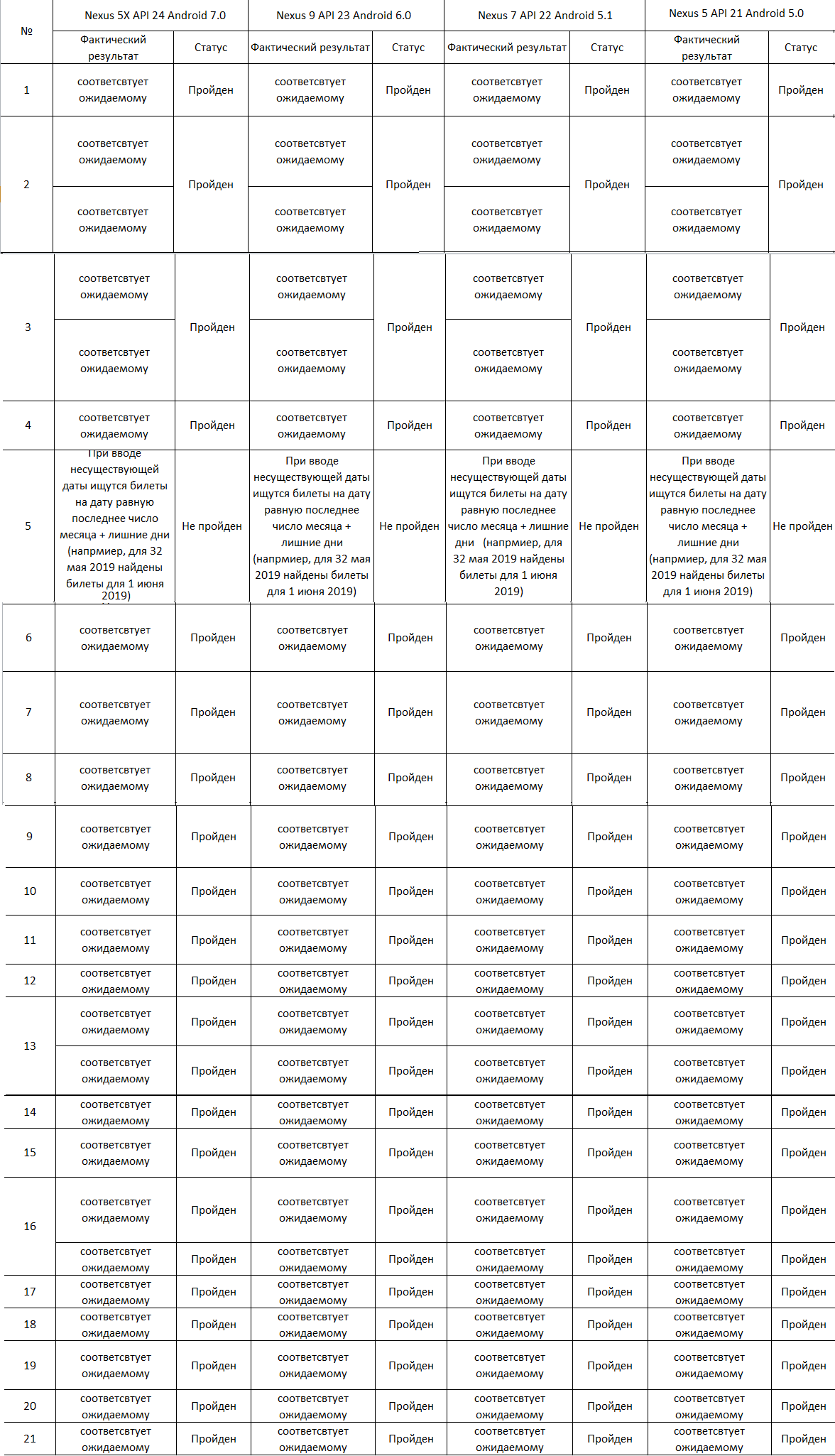


Рисунок B.3 – Тестирование устройств (2)