# Оглавление

[Цель](#_nmsqdypopp79) 4

[Основные определения](#_rl38duul3exj) 4

[Сокращения](#_m94xfji3lfy6) 5

[Введение](#_7q608jvxwjdg) 5

[Требования к проекту](#_ajyw0370trws) 6

[Проектирование системы](#_ihl17gt41j4o) 6

[Выбор технологий](#_qlejl0mkvgz5) 12

[Выбор используемых библиотек](#_iwpl18twy4wh) 14

[Описание технических решений](#_rf006iqk4u83) 14

[Заключение](#_iejuqa4i48xd) 17

[Список используемых источников](#_yj73ifrvmi13) 17

[Список приложений](#_z3fa0ybvjmjo) 18

# Цель

Реализовать систему по поиску авиарейсов. Создать сервер, который подключается к базе данных, хранящей информацию об авиарейсах. Реализовать мобильное приложение, которое позволяет запрашивать данные по пунктам вылета, прилета и дате с сервера и получать список доступных рейсов, соответствующих этим данным.

# Основные определения

*Сервер* - программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам.

*Мобильное приложение* - программное обеспечение, предназначенное для работы на смартфонах, планшетах и других мобильных устройствах.

*База данных* - представленная в объективной форме совокупность самостоятельных материалов, систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины.

*SQL* - язык программирования, применяемый для создания, модификации и управления данными в реляционной бд, управляемой соответствующей системой управления бд.

*REST API* - архитектурный стиль взаимодействия компонентов распределённого приложения в сети.

*Heroku* - облачная PaaS-платформа, поддерживающая ряд языков программирования.

*Travis CI* - распределённыйвеб-сервис для сборки и тестирования программного обеспечения, использующий GitHub в качестве хостинга исходного кода.

# Сокращения

*БД* – база данных;

*МП* - мобильное приложение;

*СУБД* - система управления базами данных;

# Введение

Сложно найти человека, который никогда бы не летал на самолете, количество рейсов растет с каждым днем. Зачастую возникает проблема поиска наилучшего варианта авиаперелета, и люди теряются во множестве сайтов авиакомпаний, выбирая в итоге не самый оптимальный и выгодный вариант.

Учитывая, что в настоящее время популярность мобильных устройств растет, наиболее удобным способом предоставления данных потребителю является МП. Его преимущества: быстрота, удобство в использовании, понятный для любого пользователя интерфейс.

Принимая все это во внимание, мы решили реализовать максимально удобное приложение для поиска авиарейсов.

Данные о рейсах будут храниться в БД, обеспечивая быстрый поиск и изменение их. Сервер позволит получать необходимую информацию.

# Требования к проекту

1. Автономная работа на сервере

2. Простота в использовании

3. Поддержка Android

4. Хранение данных в БД

5. Реализация REST API

# Проектирование системы

Система проекта состоит из трех модулей:

1. МП
2. Сервер
3. БД

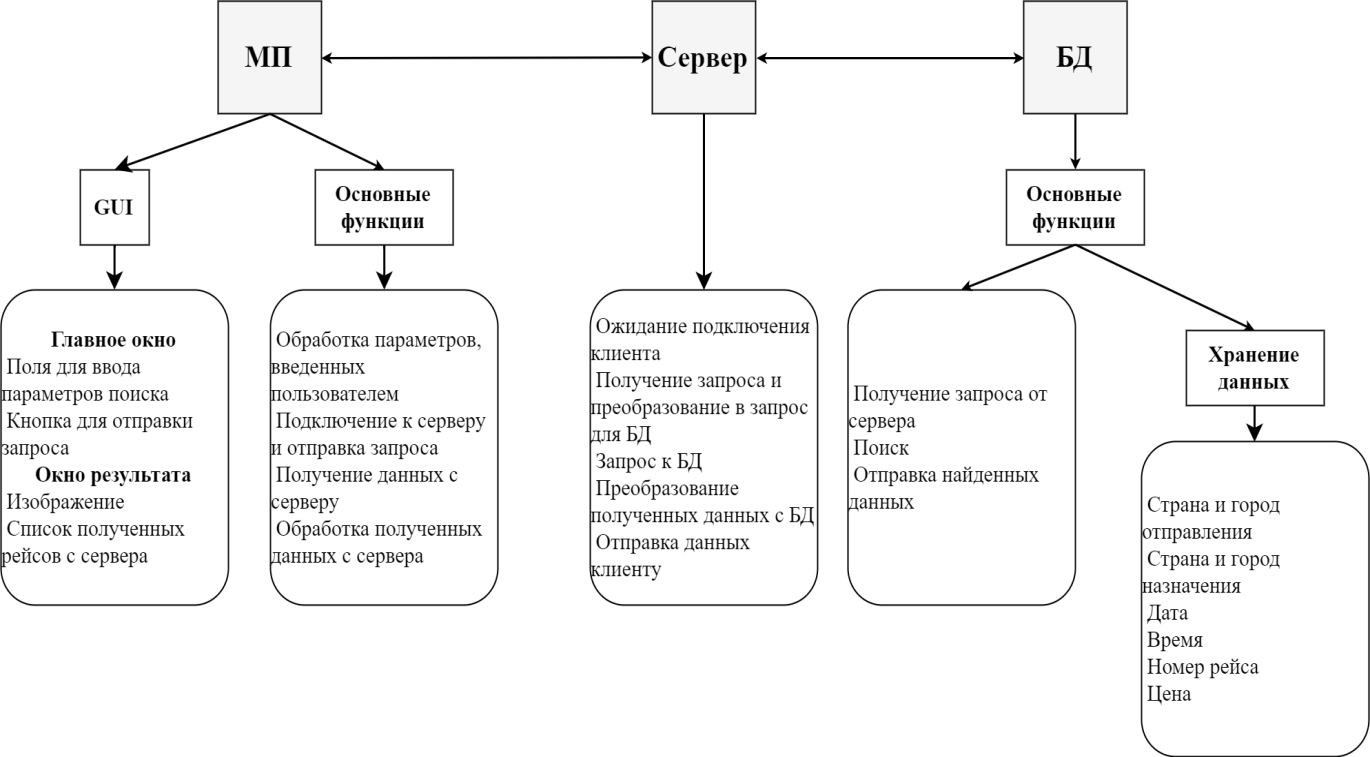


Рисунок 1 - Схема связи модулей системы.

**Мобильное приложение**

МП предоставляет пользователю возможность ввести определенные данные для поиска по БД, а затем отображает искомые рейсы.

Главный экран МП содержит 3 поля для ввода данных таких, как пункт отправления, пункт назначения и дата, а также кнопку для отправки запроса. Дата выбирается в выпадающем окне календаря. Введенные данные отправляются запросом на сервер. При попытке отправить пустую форму или при отсутствии подключения к серверу приложение выдает ошибку.

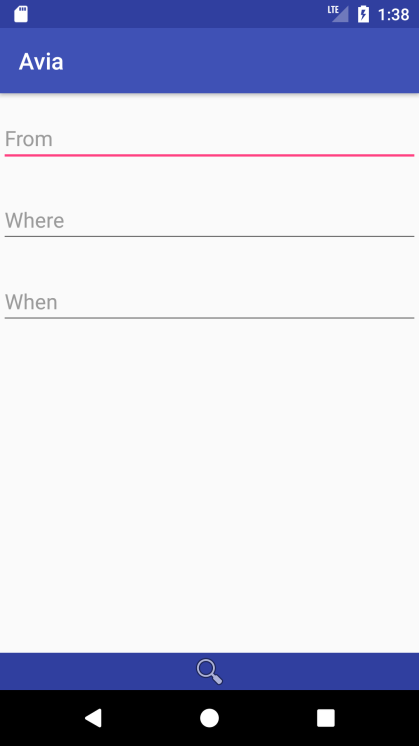


Рисунок 2 - Главный экран

Полученные данные отображаются на экране в виде таблицы со столбцами: номер рейса, дата, время и цена. Изображение загружается из локальных ресурсов и представляет собой город назначения. Если рейсы, запрашиваемые по параметрам, отсутствуют в БД, приложение выдает соответствующее сообщение. Также на экране имеется кнопка для возврата на главный экран.

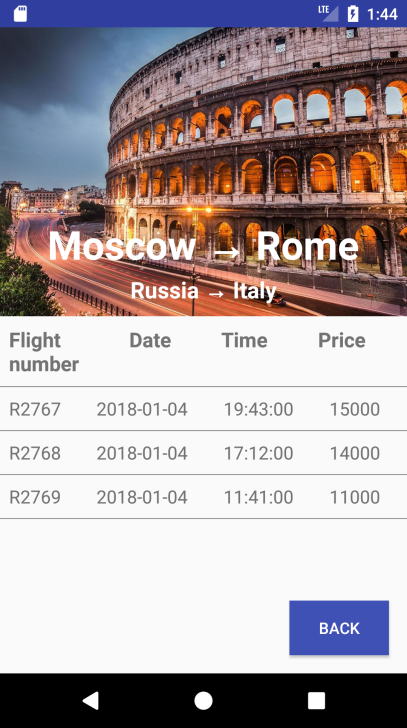


Рисунок 3 - Экран результата

Функционально приложение обрабатывает введенные строки, преобразуя их в запрос, подключается к серверу, получает искомые данные и преобразует их в удобный для чтения вид.

**Сервер**

Сервер поддерживает архитектуру REST API. Взаимодействие происходит по протоколу HTTP. HTTP действие над данными задается с помощью методов: GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Данные запроса передаются в виде query-строки при использовании метода GET.

Сервер располагается на облачной платформе Heroku, что обеспечивает его непрерывную работу и предоставляет возможность обращения к нему любого пользователя, имеющего доступ в интернет.

Подключение клиента к серверу происходит через порт. Сервер принимает GET-запрос. Ожидаемый формат запроса:

*?from=<пункт отправления>&where=<пункт назначения>&when=<дата полета в формате yyyy-mm-dd>*

Далее происходит обработка запроса, впоследствии полученные данные используются для SQL-запроса, отправляемого на БД. Если запрос, полученный сервером, не соответствует формату, то вернется сообщение “Wrong request”. Если же рейсов, соответствующих заданным параметрам нет, то сервер вернет клиенту сообщение “No tickets!”. В ином случае, пользователь получит список подходящих рейсов.

**База данных**

Для БД используем СУБД PostgreSQL. БД располагается на Heroku. Данные хранятся в трех таблицах:

1. Таблица стран “Countries”:

* id страны
* название страны

1. Таблица городов “Cities”:

* id города
* id страны
* название города

1. Таблица рейсов “Flights”:

* номер рейса
* дата
* время
* пункт отправления (город)
* пункт назначения (город)
* цена

Страны и города имеют id, для более удобного заполнения таблиц. Таким образом не требуется каждый раз вводить название города чтобы создать новый рейс, достаточно лишь указать его “номер”. Впоследствии, при создании запроса id городов и стран соотносятся с их названиями.

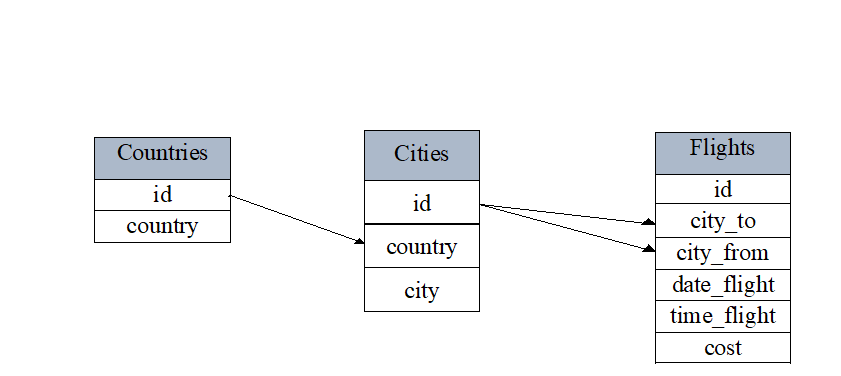
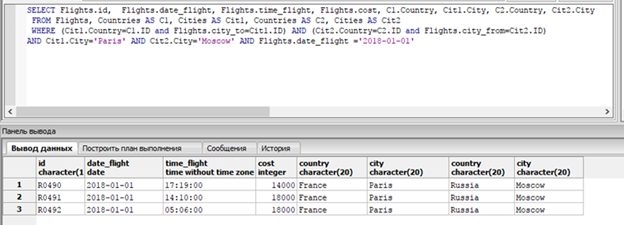


Рисунок 4 - Схема БД

БД принимает SQL-запрос с сервера и производит поиск по заданным параметрам, а именно: город вылета, город прилета и дата, когда совершается перелет. Ответ, возвращаемый на сервер, содержит строки с информацией о рейсе, состоящие из номера рейса, даты и времени вылета, цены билета и пунктов назначения и отправления (страна и город).

Рисунок 5 - Пример SQL-запроса к БД  


# Выбор технологий

Для написания МП был выбран язык программирования Java.

Одним из требований проекта была поддержка Android, что являлось основным критерием при выборе языка программирования.

Java является официальным языком разработки для Android, поэтому был выбран для рассмотрения. Одним из важных преимуществ Java является довольно высокая безопасность по сравнению со многими другими языками программирования. Поскольку все приложения выполняются и, соответственно, контролируются Java-машиной, то при любой попытке несанкционированного доступа или соединения программа просто прерывает свою работу. Помимо этого к языку существует огромное количество бесплатных программных библиотек, написанных программистами всего мира. К недостаткам языка следует отнести чрезмерную нагрузку на оперативную память оборудования.

Одним из главных конкурентов Java на данный момент можно назвать Kotlin. Kotlin — статически типизированный язык программирования. Можно выделить такие достоинства, как полная совместимость с Java, простой и компактный синтаксис, что сокращает объем кода и повышает читаемость, также заложенные в синтаксис языка правила создания кода позволяют избегать мелких трудно отслеживаемых ошибок. Несмотря на очевидные преимущества, Kotlin все еще уступает Java по быстродействию, а также в связи с новизной языка количество инструментов и библиотек существенно меньше чем на Java.

Выбирая язык программирования для написания сервера мы рассматривали PHP и Node.js.

PHP - язык серверных сценариев, который можно внедрить в HTML-страницу. Он имеет ряд преимуществ таких, как легкость изучения, эффективная поддержка БД, в том числе используемый в данном проекте PostgreSQL, кроме того PHP поддерживается любым хостингом.

Однако, Node.js имеет ряд своих преимуществ. Прежде всего, все механизмы обработки запросов и прочих операций ввода/вывода построены на событиях. Это означает, что в Node.js нет никакого способа, чтобы заблокировать работающий в данный момент поток. Каждая операция в Node.js выполняется асинхронно. Это является огромным преимуществом, особенно при работе с БД. В отличие от PHP, в Node.js приложение постоянно работает и его нужно инициализировать только один раз.

# Выбор используемых библиотек

Для развертывания сервера была выбрана облачная платформа Heroku. Это удобная бесплатная площадка для микро проектов. Для загрузки проекта heroku предлагает удобные инструменты, одним из которых является GitHub. Heroku поддерживает множество языков программирования, одним из которых является Node.js. Так же Heroku предлагает различные инструменты и сервисы для развития и эксплуатации проекта. Она обеспечивает простое взаимодействие проекта с БД. Так как по умолчанию Heroku предоставляет СУБД PostgreSQL. Именно поэтому PostgreSQL и используется для создание БД в данном проекте.

# Описание технических решений

Сервер реализован с помощью языка Node.js, использованы модули http, url и pg (для работы с PostgreSQL).

* Метод createServer() создает сервер. Возвращает объект http.Server. Для обработки подключений в метод createServer передается функция с двумя параметрами: request (хранит информацию о запросе, представляет объект http.IncomingMessage) и response (управляет отправкой ответа, представляет объект http.ServerResponse).
* Метод объекта сервера listen() позволяет серверу прослушивать и обрабатывать входящие подключения. В качестве параметра передается номер порта, по которому запускается сервер.
* Метод url() объекта http.IncomingMessage возвращает запрошенный URL-адрес.
* Метод end() объекта http.ServerResponse отправляет клиенту запрошенные данные. В качестве параметра передается обработанный ответ с БД.
* Метод connect() создает подключение к БД.
* Метод query() отправляет запрос к БД. В качестве параметров передаются запрос и функция, описывающая обработку полученного от БД ответа.
* Метод end() закрывает подключение сервера к БД.

Приложение с сервером размещено на облачной платформе Heroku.После создания приложения, следует настроить его автоматическое развертывание.

Созданный проект подключается к существующему git-репозиторию с сервером. Для этого выбирается метод развертывания GitHub.

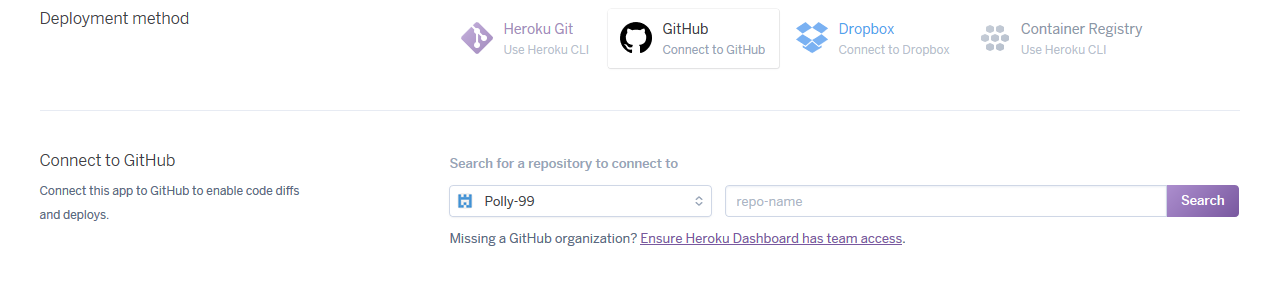


Рисунок 6 - Выбор метода развертывания

После выбирается организация, в которой находится нужный репозиторий, вводится его название и нажимается ‘connect’.

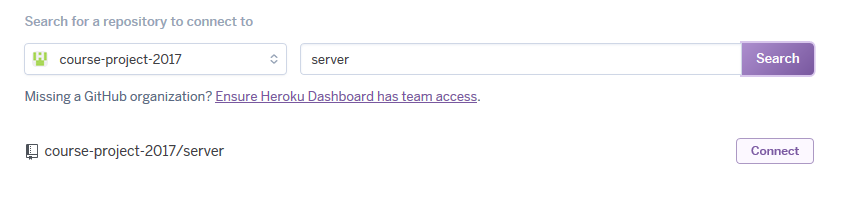
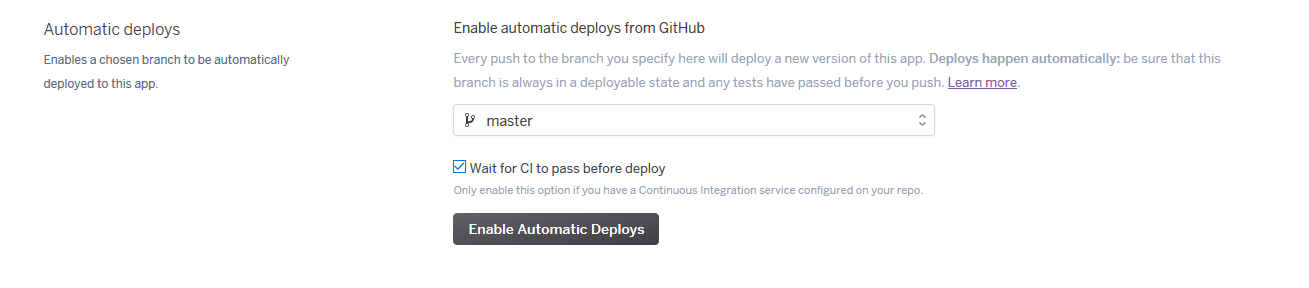


Рисунок 7 - Подключение к существующему репозиторию

Для настройки ожидания успешного прохождения построения CI, в данном случае Travis CI, к которому подключен проект, выбирается пункт ‘Wait for CI to pass before deploy’.

Рисунок 8 - Подключение ожидания CI

Для настройки автоматического развертывания с git-репозитория следует нажать ‘Enable Automatic Deploys’.

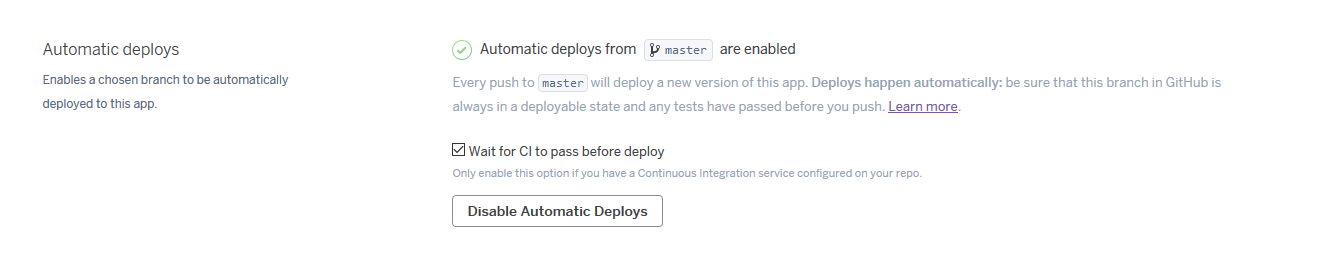


Рисунок 9 - Подключение автоматического развертывания

Таким образом, после любого изменения в репозитории на GitHub оно будет проходить проверку в Travis CI, после успешного прохождения которого проект на Heroku будет автоматически обновляться.

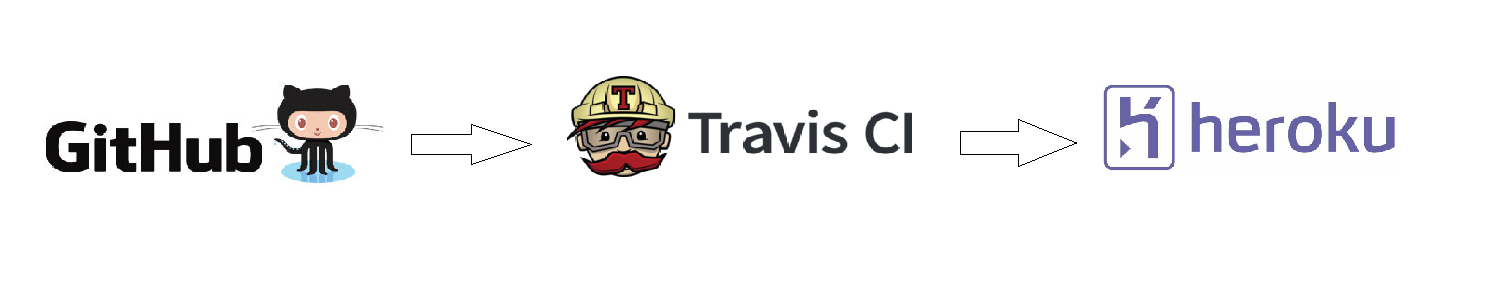


Рисунок 10 - Порядок развертывания

# Заключение

В результате в курсовом проекте нами были реализованы МП на языке программирования Java для ОС Android, сервер на Node.js и БД на PostgreSQL, развернутые на облачной платформе Heroku. Настроена передача запросов и ответов между МП и сервером, сервером и БД.

В дальнейшем планируется развивать и улучшать проект. Будет расширена БД, предоставлена возможность поиска обратных билетов, а также нахождение полетов с пересадками.

# 

# Список используемых источников

1. Грабер, М. Введение в SQL/М. Грабер - М.:ЛОРИ, 1996. - 375 с.
2. Android Developers [Electronic resource], - Mode access: https://developer.android.com
3. Getting Started on Heroku with Node.js [Electronic resource], - Mode access: https://devcenter.heroku.com/articles/getting-started-with-nodejs

# 

# Список приложений

Приложение А. Документация кода МП

Приложение Б. Исходный код