МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ОБРАЗОВАНИЯ**

**“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

Факультет *компьютерных наук*

Кафедра *программирования и информационных технологий*

*Telegram-бот для взаимодействия с Github.*

*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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_*Д.А. Кравченко 3 курс, д/о*

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_*М.В. Старкин 3 курс, д/о*

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_*И.В. Мущенко 3 курс, д/о*

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Х.А. Полещук, аспирант*

Воронеж 2019

# Содержание

[Введение 4](#_Toc10527300)

[1. Постановка задачи 5](#_Toc10527301)

[2. Анализ 7](#_Toc10527302)

[2.1. Анализ предметной области 7](#_Toc10527303)

[2.2. Сравнение с аналогами 8](#_Toc10527304)

[2.3. Анализ средств реализации 8](#_Toc10527305)

[2.4. Графическое описание работы системы 10](#_Toc10527306)

[2.4.1. Диаграммы состояний 10](#_Toc10527307)

[2.4.2. Диаграммы активности 13](#_Toc10527308)

[2.4.3. Диаграммы последовательностей 16](#_Toc10527309)

[2.4.4. Диаграмма развертывания 18](#_Toc10527310)

[3. Реализация 19](#_Toc10527311)

[3.1. Реализация серверной части 20](#_Toc10527312)

[3.1.1 Слой Models 20](#_Toc10527313)

[3.1.2 Слой Service 21](#_Toc10527314)

[3.1.3 Слой DAO 21](#_Toc10527315)

[3.1.4 Слой Controller 22](#_Toc10527316)

[3.1.5 Реализация базы данных 22](#_Toc10527317)

[4. Интерфейс 24](#_Toc10527318)

[5. Тестирование 29](#_Toc10527319)

[5.1. Модульное тестирование 29](#_Toc10527320)

[5.2. Системное тестирование 30](#_Toc10527321)

[5.3. Вывод 31](#_Toc10527322)

[Заключение 33](#_Toc10527323)

[Список использованных источников 33](#_Toc10527324)

# Введение

Современное программирование невозможно представить без использования систем контроля версий. Разработка ведется в командах, состоящих из множества людей, поэтому необходимо знать обо всех изменениях, произошедших с репозиториями, в разработке содержания которых ты участвуешь.

Также у всех команд существуют общие чаты в мессенджерах, в которых они обсуждают вопросы, связанные с процессом разработки. Поэтому возможное взаимодействие системы контроля версий и мессенджера выглядит крайне привлекательно: участники смогут получать оперативную информацию обо всех изменениях, произошедших в репозиториях, а также структурировать процесс разработки проектов.

Такое приложение позволило бы экономить время, улучшить взаимосвязь между участниками процесса, структурировать процесс разработки.

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 разработка Telegram-бота, способного оповещать пользователя путем отправки сообщений в чат об изменениях в указанных им репозиториях.

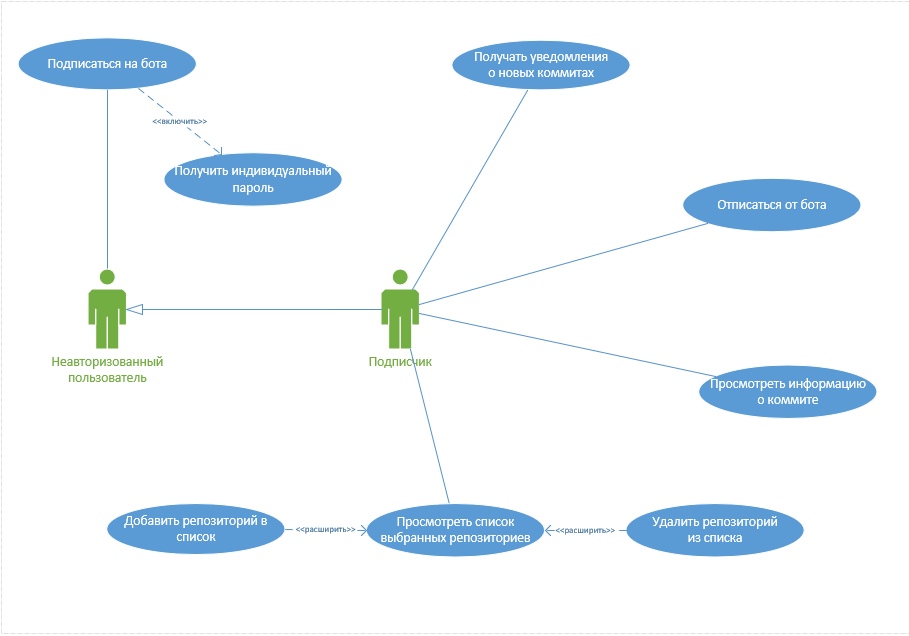
Основную функциональность разрабатываемого приложения отражает диаграмма прецедентов, изображенная на рисунке 1.

Рисунок 1 - Диаграмма вариантов использования

Неавторизованный пользователь обладает следующими возможностями:

* подписка на Telegram-бот;

Авторизованный пользователь обладает следующими возможностями:

* получать уведомления о новых коммитах;
* отписаться от Telegram-бота;
* просмотреть информацию о коммите;
* просмотр списка выбранных репозиториев;
* удаление репозитория из списка;

Система должна соответствовать следующим требованиям:

1. Использование Git API.
2. Своевременное оповещение об изменениях.
3. Авторизация на сайте посредством использования Github.

Завершенный проект представляет собой полностью функционирующего Telegram-бота, соответствующее требованиям, описанным выше.

# Анализ

## Анализ предметной области

Бот - специальная [программа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), выполняющая [автоматически](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) и/или по заданному [расписанию](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D0%B5) какие-либо действия через [интерфейсы](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81), предназначенные для [людей](https://ru.wikipedia.org/wiki/%D0%9B%D1%8E%D0%B4%D0%B8).

GitHub — сервис онлайн-хостинга репозиториев, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом.

Для того, чтобы получать все обновления от GitHub, нам потребуется настроить GitHub таким образом, чтобы он посылал нам информацию об обновлениях, а также создать приложение, который эту информацию будет получать и обрабатывать.

Для получения данных от Github о каком-либо пользователе, необходимо знать логин этого пользователя. Соответственно, используя авторизацию на нашем сервисе с помощью Github, мы получим необходимые данные о пользователе для последующего получения информации.

С [GitHub API](https://developer.github.com/v3/) можно взаимодействовать посредством GET, POST запросов. GitHub ответит нам в [формате JSON](http://www.json.org/json-ru.html), который представляет из себя достаточно удобный способ представления информации в виде списков. Пример ответа [GitHub API](https://developer.github.com/v3/) представлен на листинге 1.

Листинг 1. Пример ответа Git Api.

{

"login":"user",

"id":988885,

"avatar\_url":"https://avatars.githubusercontent.com/u/988885?v=3",

"gravatar\_id":"",

"url":"https://api.github.com/users/user",

"html\_url":"https://github.com/user",

"type":"User",

"site\_admin":false,

"name":"Userov user",

"public\_repos":2,

"public\_gists":11,

"followers":12,

"following":20,

"created\_at":"2011-08-18T14:54:56Z",

"updated\_at":"2015-12-23T17:54:08Z"

}

## Сравнение с аналогами

Не существует официального аналога Telegram-ботов, реализующих оповещение об изменениях пользовательских репозиториев, лишь множество вариантов пользовательских реализаций.

## Анализ средств реализации

В качестве средств реализации системы оповещений об изменении пользовательских репозиториев были выбраны:

1. Для разработки клиентской части приложения было принято решение использовать Angular 7.0 – это открытая и свободная платформа для разработки веб-приложений. Фреймворк обеспечивает разработку SPA-приложения (одностраничного приложения), предоставляя для разработки проекта такой функционал, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизация внутри приложения и так далее. Взаимодействие с фреймворком Angular построено на языке TypeScript, компилируемого в JavaScript. Одно из главных отличий языка – система типизации, позволяющая облегчить работу в больших проектах.
2. В качестве СУБД была выбрана PostgreSQL, так как данная база является одной из самых популярных и поддерживаемых в мире, а также имеющая очень удобный инструмент для работы с ней, такой как pgAdmin4.
3. Spring Boot – мощнейший фреймворк, предоставляющий огромный функционал, облегчающий разработку веб-приложений.
4. Для реализации серверной был выбран высокоуровневый язык программирования Java, так как отсутствует зависимость от используемой платформы, а также данный язык программирования обладает автоматическим управлением памятью, высокой стабильностью и огромным количеством вспомогательных материалов.
5. Для заключения контракта между модулями и документированием был использован Swagger.

Для взаимодействия c REST API использовался протокол HTTPS.

## Графическое описание работы системы

Для удобства описания работы системы была использована графическая нотация UML. В данном разделе представлены диаграммы и описания, где они требуются.

### Диаграммы состояний

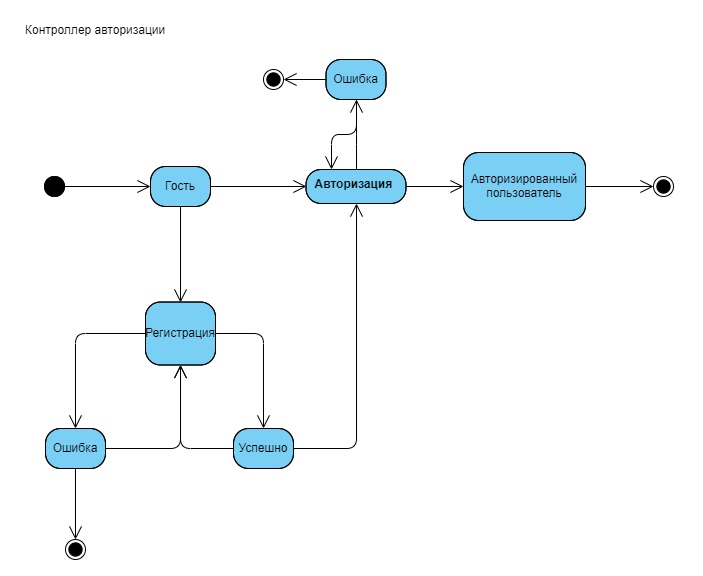
Для описания состояний, в которых находится сервер при авторизации, составлена диаграмма, изображенная на рисунке 2.

Рисунок 2 - Диаграмма состояния «Авторизация»

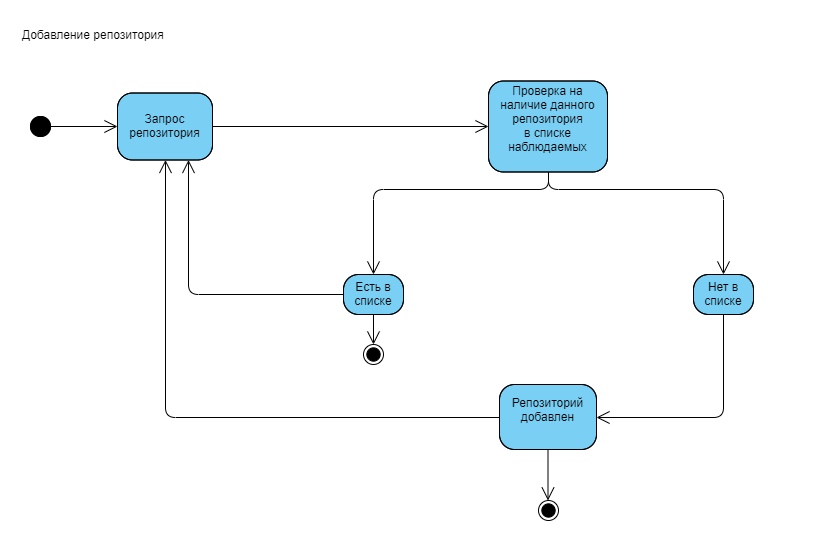
Для описания состояний, в которых находится сервер при добавлении репозитория, составлена диаграмма, изображенная на рисунке 3.

Рисунок 3 - Диаграмма состояния «Добавление репозитория»

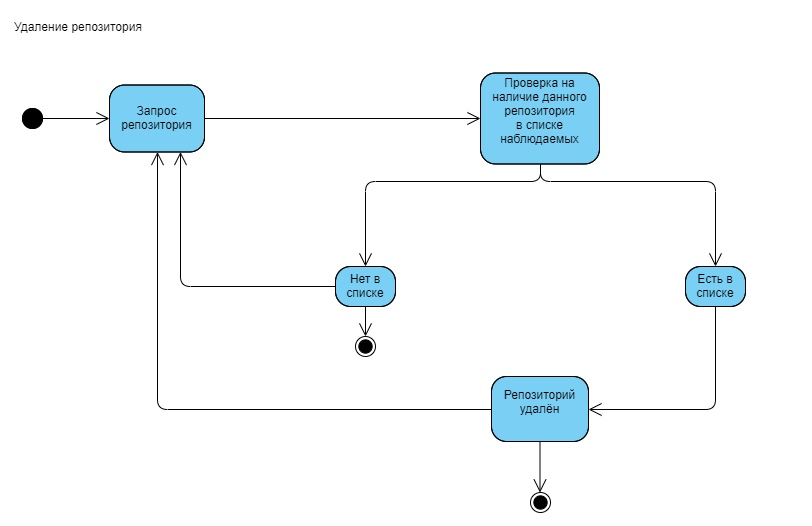
Для описания состояний, в которых находится сервер при удалении репозитория, составлена диаграмма, изображенная на рисунке 4.

Рисунок 4 - Диаграмма состояния «Удаление репозитория»

### Диаграммы активности

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

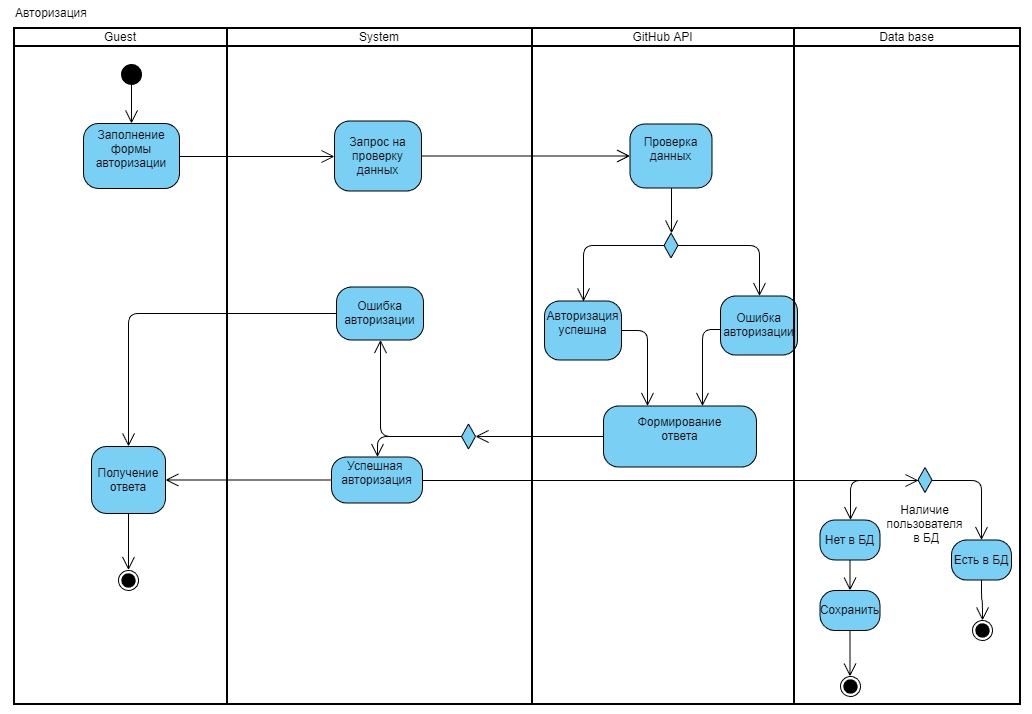
Диаграмма активности авторизации изображена на рисунке 5. На данной диаграмме присутствуют 4 части: Guest, Sustem, Database и сторонние источники (API).

Рис. 5. Диаграмма активности «Авторизация»

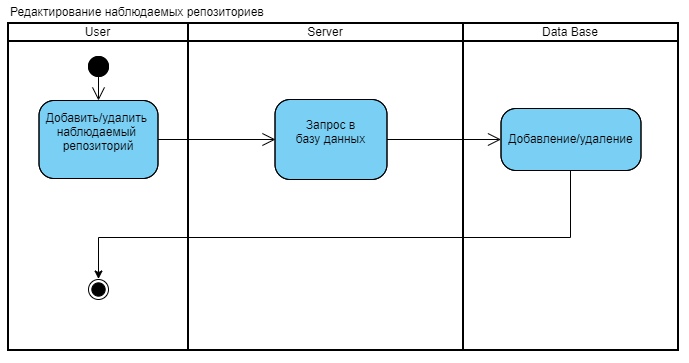
Диаграмма активности добавление/удаление репозитория изображена на рисунке 6. На данной диаграмме присутствуют 3 части: User, Server и Data base.

Рис. 5. Диаграмма активности «Добавление/удаление репозитория»

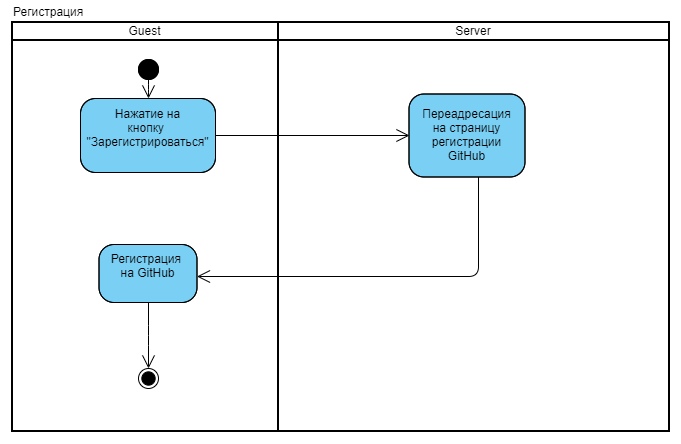
Диаграмма активности регистрации изображена на рисунке 6. На данной диаграмме присутствуют 3 части (дорожки): пользователь, приложение и сервер.

Рис. 6. Диаграмма активности «Регистрация»

### Диаграммы последовательностей

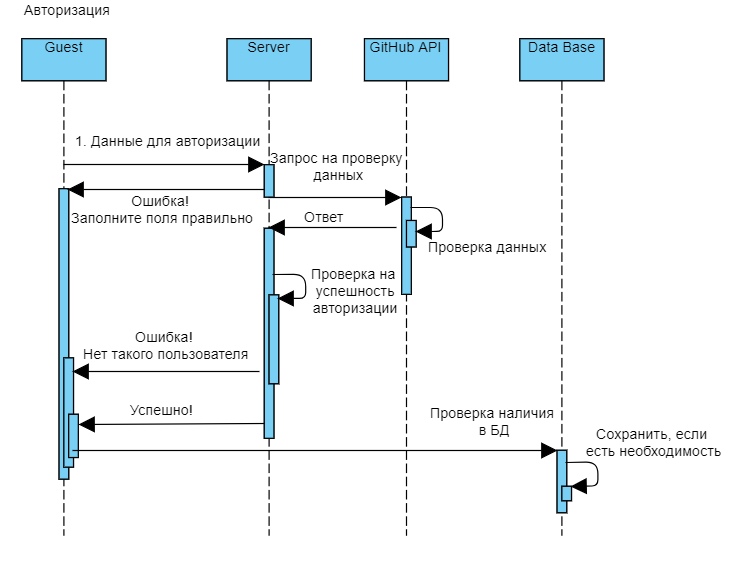
Диаграмма последовательностей авторизации изображена на рисунке 7.

Рис. 7. Диаграмма последовательности «Авторизация»

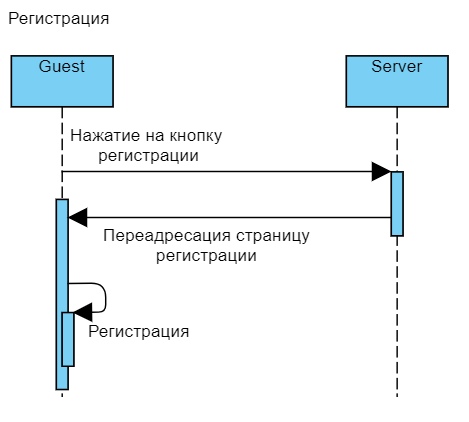
Диаграмма последовательностей регистрации изображена на рисунке 8.

Рис. 8. Диаграмма последовательности «Регистрация»

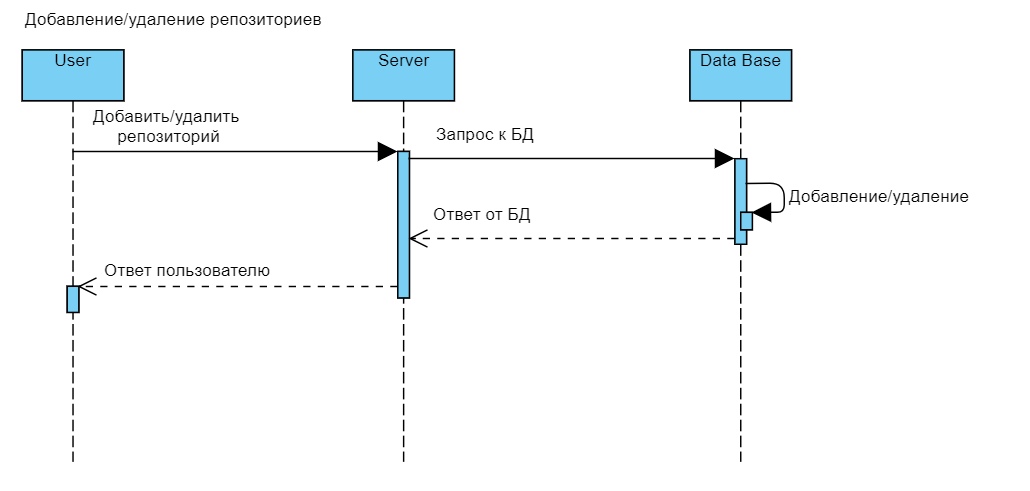
Диаграмма последовательностей добавление/удаление изображена на рисунке 9.

Рис. 9. Диаграмма последовательности «Добавление/Удаление репозитория»

### Диаграмма развертывания

Приведенная на рисунке 10 диаграмма визуализирует элементы и компоненты программы, которые существуют на этапе ее исполнения.

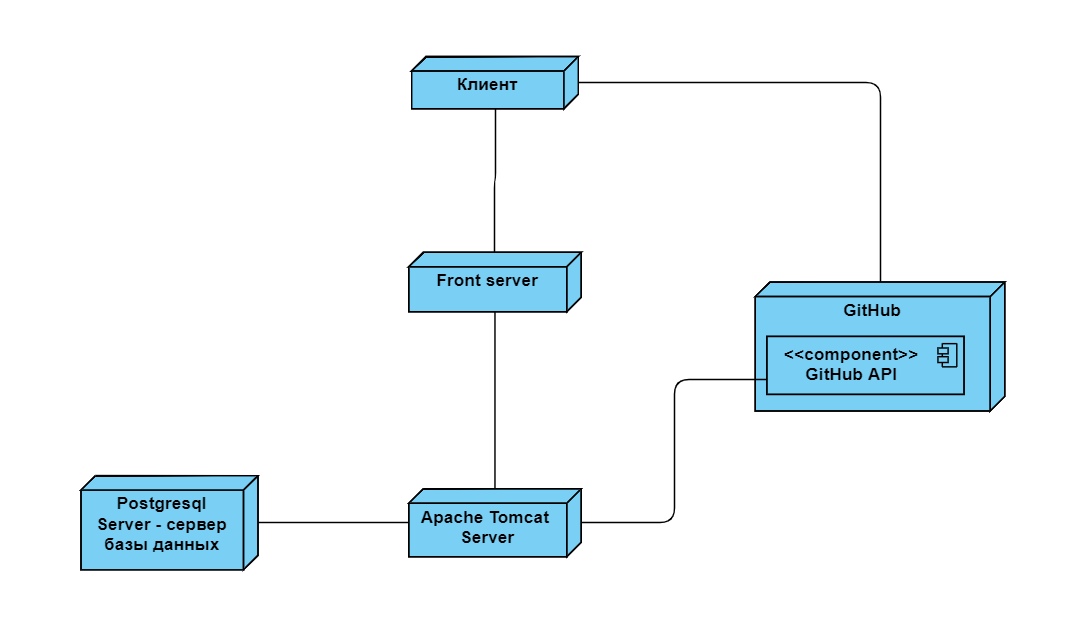


Рис. 10. Диаграмма развертывания

# Реализация

В данной главе описана реализация всех подсистем приложения по поиску авиабилетов, задачи которых описаны в главе «Анализ» в разделе «Анализ задач». Для улучшения восприятия диаграмм классов в некоторых из них были отражены только методы, существенные для понимания основной логики работы системы и ее частей.

## Реализация серверной части

Серверная часть разделена на несколько уровней, каждый из которых реализует свой функционал:

- Model;

- Service;

-DAO

- Controller;

Для работы программы была выбрана ORM – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков, создавая «виртуальную объектную модель базы данных». То есть в программе реализована объектная модель данных, а в базе данных хранится реляционная. Она генерируется и поддерживается на концепции JPA - технологии, обеспечивающей объектно-ориентированное отображение простых JAVA объектов и предоставляющих API для сохранения, получения и управления ими. По своей сути JPA – это спецификация, то есть документ, утверждённый как стандарт, описывающий все аспекты технологии.

В качестве реализации данной спецификации был выбран Hibernate Framework – библиотека для java, предназначенная для решения задач объектно-реляционного отображения, упрощающая работу с базой данных за счёт:

- Уменьшения повторяющегося кода;

- Упрощения создания новых сущностей и управления транзакциями;

- Поддерживания наследования, ассоциации и коллекции;

Рассмотрим подробнее каждый из описанных выше уровней.

### 3.1.1 Слой Models

Директория model содержит models классы – сущности, с помощью которых гораздо легче работать с базой данных:

- Повышается скорость разработки за счёт уменьшения количества однообразного вспомогательного кода;

- Приложение не привязано к конкретной СУБД;

Такими model классами в данном проекте выступают:

* GithubUser – класс, который описывает поля, присущие пользователю данного приложения.
* Repository – класс, который описывает поля, присущие github репозиторию.

Model классы, отражают сущности из базы данных. В них описаны все поля из базы данных и методы для работы с ними.

### 3.1.2 Слой Service

Данный слой содержит бизнес логику проекта. При реализации слой Service обращается к DAO, обрабатывает полученные данные, и возвращает результат в Controller, после чего тот отправляет ответ клиентской части.

В данном проекте присутствуют следующие сервисы:

* GithubUserService – для работы с пользователями данного приложения.
* RepositoryService - для работы с репозиториями данного приложения.

Также, стоит отметить, что каждый сервис имеет интерфейс, который он реализует. Сделано это для повышения гибкости и управляемости кода. Помимо этого, каждый сервис имеет свою реализацию. А всё это вместе – как раз и есть слой сервисов.

## 3.1.3 Слой DAO

DAO – data access object. Это объект, который предоставляет абстрактный интерфейс к какому-либо типу базы данных или механизму хранения. В нашей задаче данный слой наследуется от класса JpaRepository. Это класс, которым мы можем воспользоваться, и который идёт вместе с фреймворком Spring. Он помогает упростить написание запросов к базе данных. Теперь не будет sql запросов, а будут длинные и интуитивно понятные имена методов. В данном проекте также используются два интерфейса, которые помогают работать и с GithubUser и Repository.

## 3.1.4 Слой Controller

Этот слой, пожалуй, самый важный из всех предыдущих. Так как именно он связывает серверную часть и клиентскую. Все запросы с клиентской части прямиком отправляются на сервер и обрабатываются контроллерами. В данном приложении реализованы два контроллера:

* UserController – первый обрабатывает запросы, которые прямиком связаны с пользователем приложения
* SearchController – второй, который служит больше для решения вспомогательных задач (таких как, например, поиск)

## 3.1.5 Реализация базы данных

Стоит отметить, что в данном проекте реализована база данных, для хранения пользователей и репозиториев. Схему базы данных вы можете увидеть на рисунке 11.

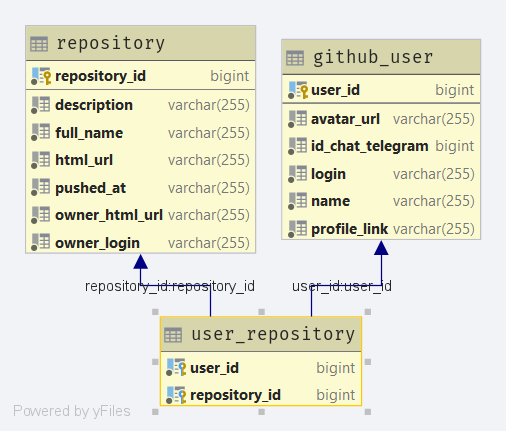


Рисунок 11 - Схема базы данных

Рассмотрим каждую таблицу подробнее:

1. Repository – таблица, которая хранит в себе список репозиториев, на которые подписаны пользователи. В ней присутствуют следующие поля:
   1. Repository\_id – идентификатор репозитория.
   2. Description – описание репозитория.
   3. Full\_name – полное имя репозитория (вида userLogin/repositoryName).
   4. Html\_url – ссылка на данный репозиторий.
   5. Pushed\_at – дата, когда был создан репозиторий.
   6. Owner\_html\_url – ссылка на владельца репозитория.
   7. Owner\_login – логин владельца репозитория.
2. Github\_user – таблица, хранящая в себе информацию о пользователях нашего приложения. Каждый пользователь имеет следующие поля:
   1. User\_id – идентификатор пользователя.
   2. Avatar\_url – ссылка на аватар пользователя.
   3. Id\_chat\_telegram – идентификатор, который используется телеграммом для идентификации.
   4. Login – GitHub логин пользователя
   5. Name – имя пользователя
   6. Profile\_link – ссылка на профиль пользователя
3. User\_repository – таблица, реализующая отношение многие ко многим. Таким образом, в ней содержатся всего два поля:
   1. User\_id – идентификатор пользователя.
   2. Repository\_id – идентификатор репозитория.

# Интерфейс

Пользовательский интерфейс сервиса состоит из четырёх основных страниц и одной дополнительной, на которой находятся ссылки на создателей данного приложения. В верхнем блоке каждой страницы находится меню, содержащее:

* Ссылку на главную страницу;
* Поле для поиска репозиториев;
* Навигация по доступным страницам;

В нижней части находится панель, содержащая ссылку на страницу с информацией о создателях приложения, на страницу факультета и на сайт университета.

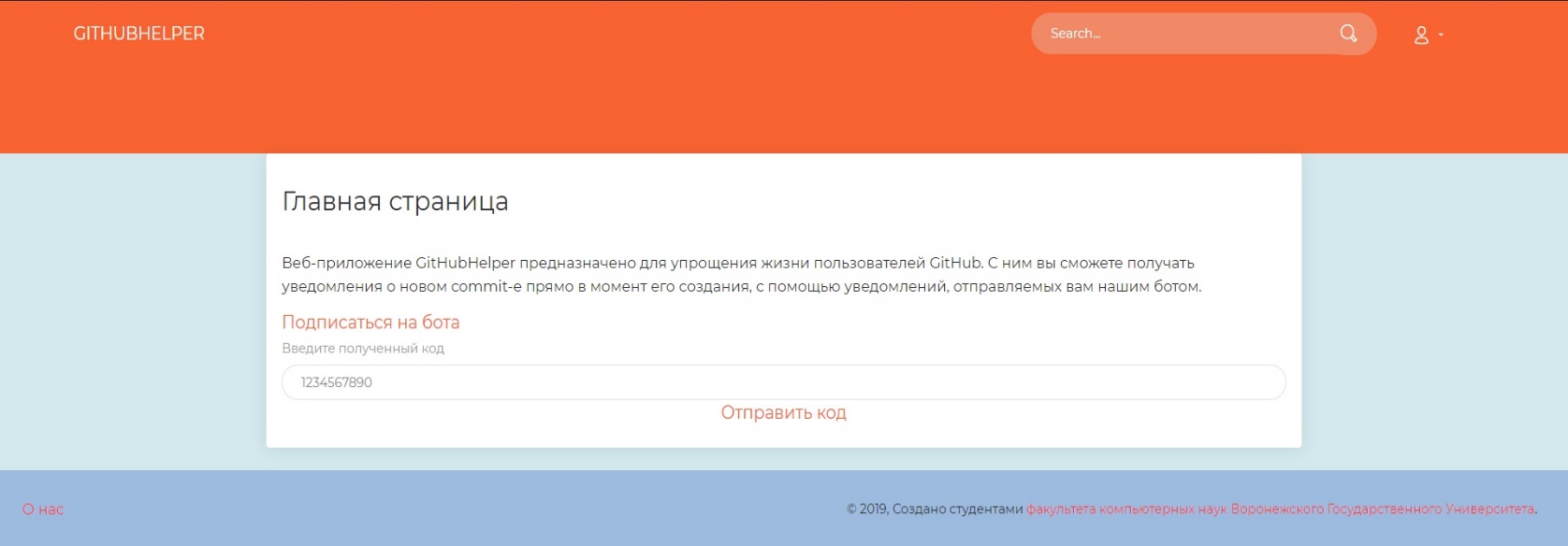
На главной странице изображённой на рисунке 12, находится информация о данном сервисе и ссылка на telegram-бот-а. Также поле для ввода chatId, который пользователь получает после подписки на бота.

Рисунок 12 - Интерфейст главной страницы

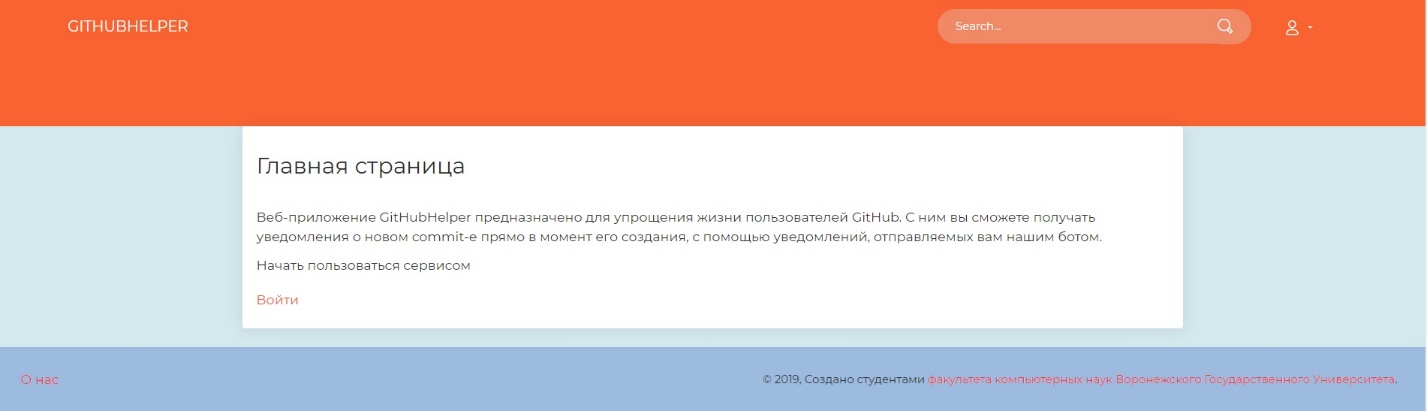
После отправки кода, внешний вид главной страницы изменяется, как показано на рисунке 13.

Рисунок 13 - Интерфейс главной страницы после отправки chatId

Нажав на кнопку «Войти», пользователя перебрасывает на страницу авторизации через GitHub, это необходимо для того, чтобы работать с данными о пользователе. После ввода пользователь становится авторизованным. И его перенаправляет обратно на страницу данного приложения.

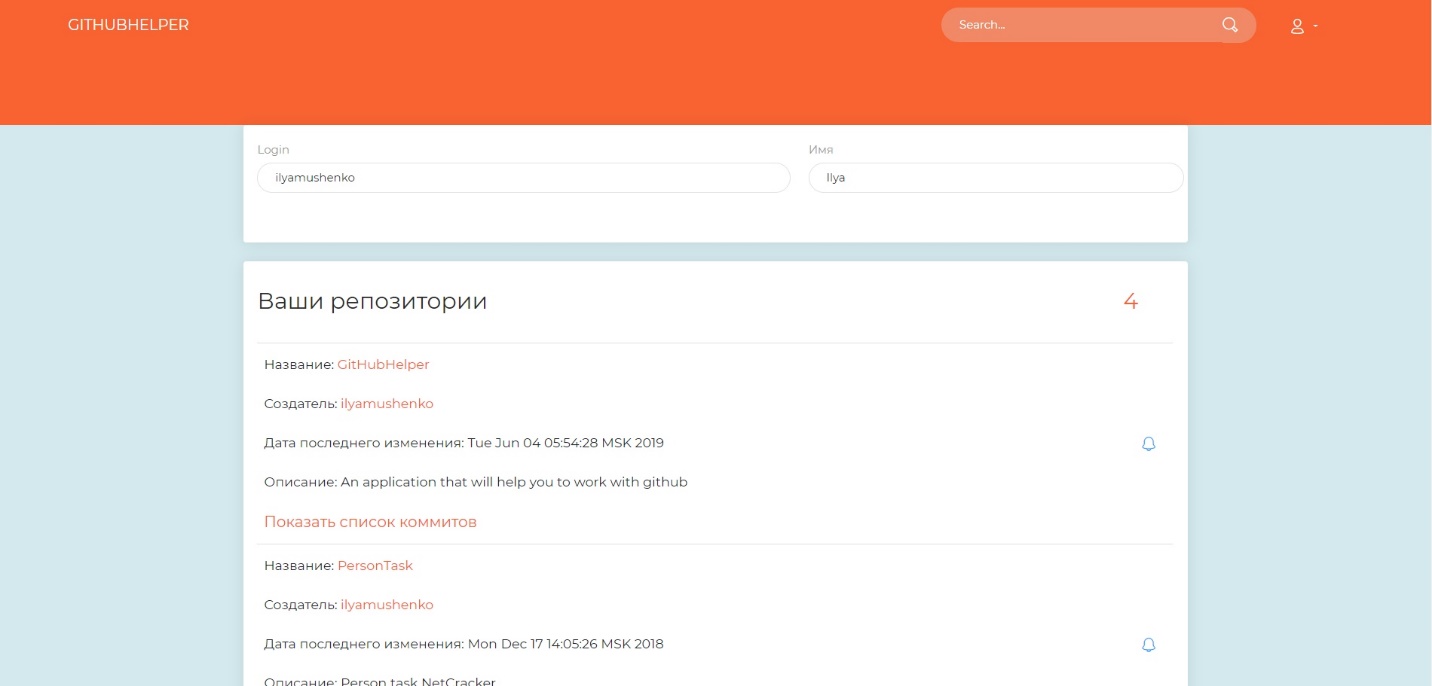
На этой странице находится информация о пользователе и его репозиториях. Её внешний вид представлен на рисунке 14.

Рисунок 14 - Страница с репозитория пользователя

Пользователь может:

* Просмотреть сведения о каждом репозитории;
* Подписаться на уведомления о коммитах даного репозитория;
* Увидеть список коммитов для каждого конкретного репозитория;

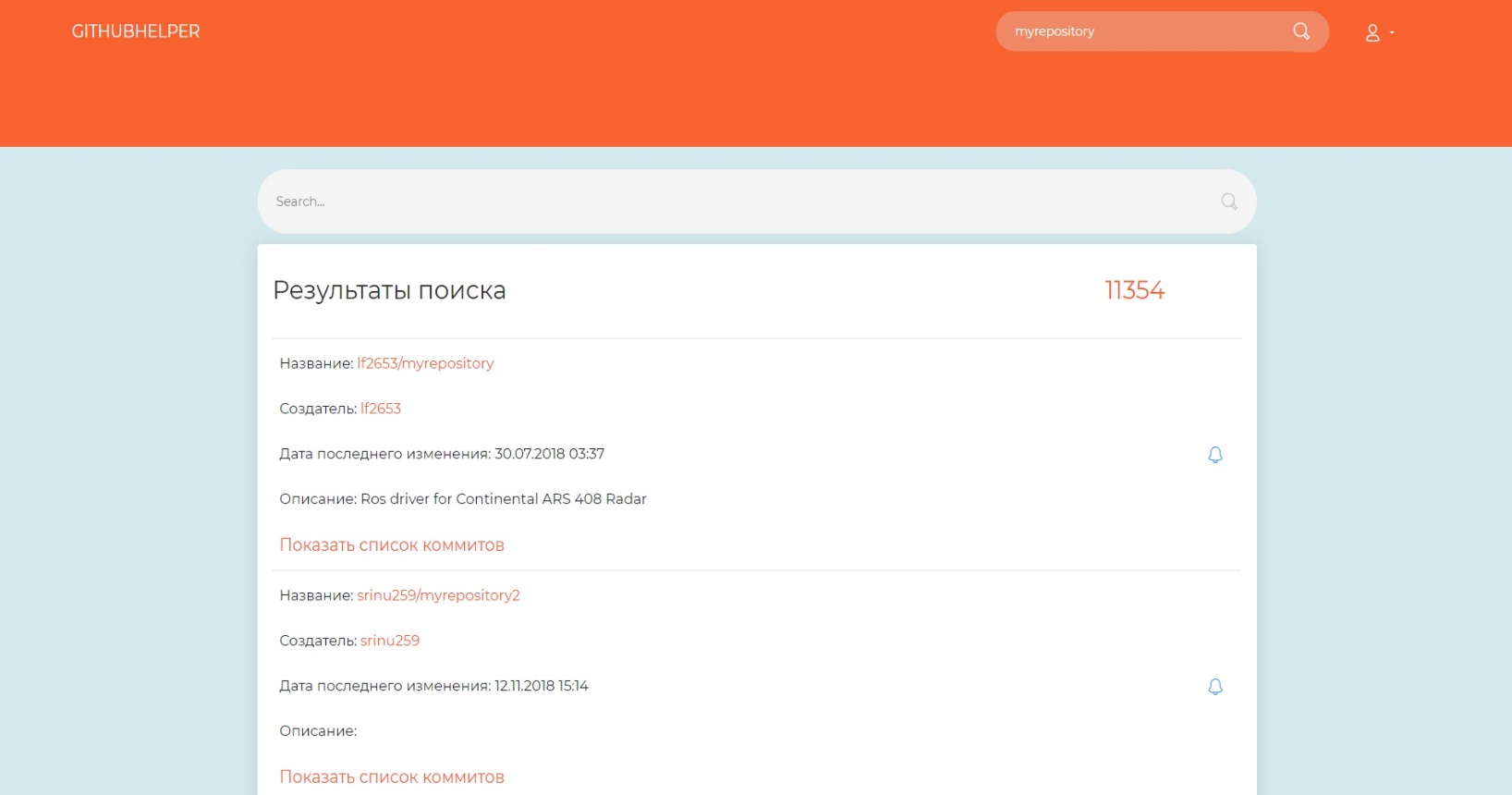
Также практически на каждой странице есть возможность воспользоваться поиском и попасть на страницу с его результатами, которая показана на рисунке 15.

Рисунок 15 - Страница с результатами поиска

На данной странице отображены все результаты поиска максимум по 30 репозиториев на странице. Это реализовано за счёт пагинации, которая уже встроена в GitHub API. Также предусмотрена возможность просмотра истории коммитов, чтобы пользователю было легче принять решение: подписываться или нет.

Страница, на которой отображены отслеживаемые репозитории, представлена на рисунке 16.

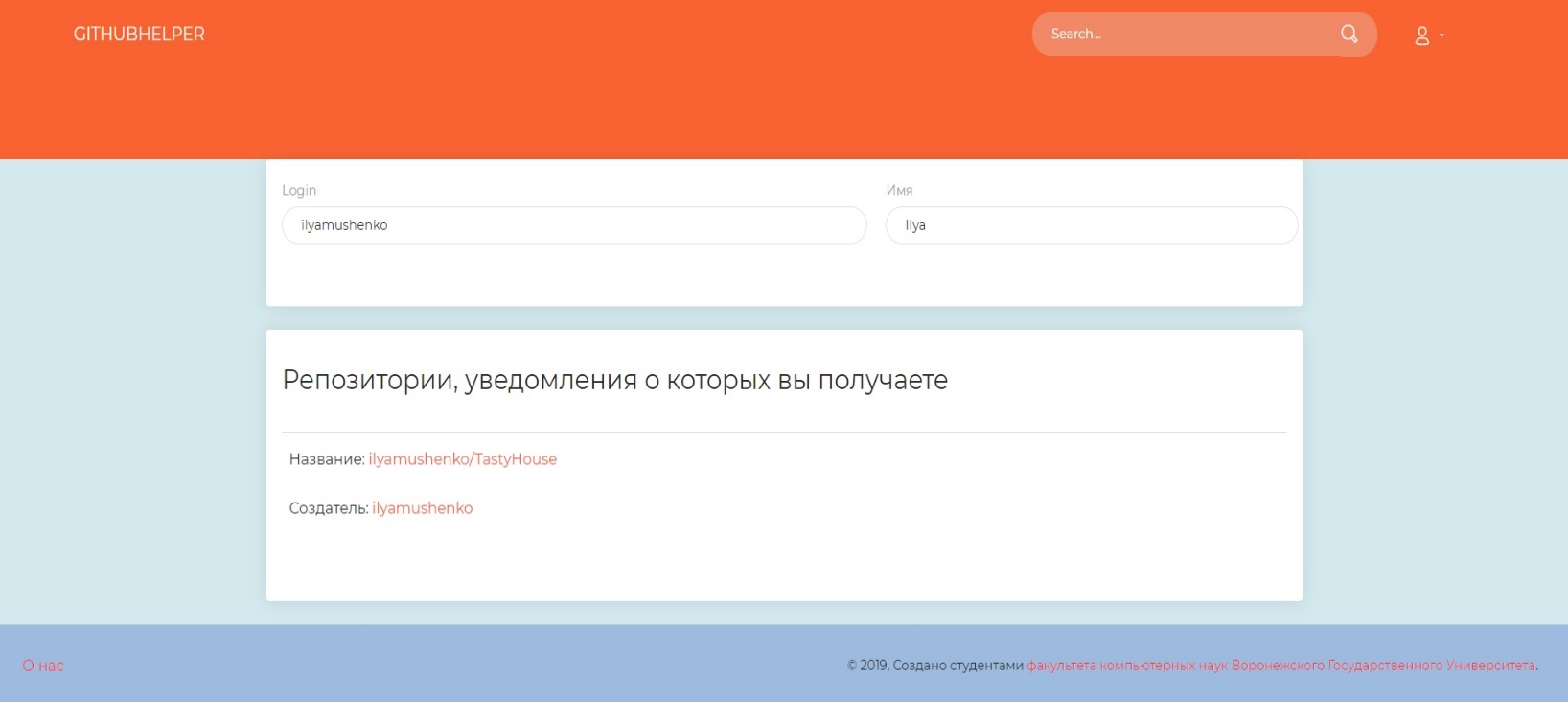


Рисунок 16 - Страница с репозиториями, на которые подписан пользователь

# Тестирование

Для того, чтобы удостовериться в работоспособности приложения и его сервисов, на всем пути разработки проводилось разнообразное тестирование. Были проведены тесты по принципу «белого ящика» следующего вида:

* модульное тестирование;
* системное тестирование;

## Модульное тестирование

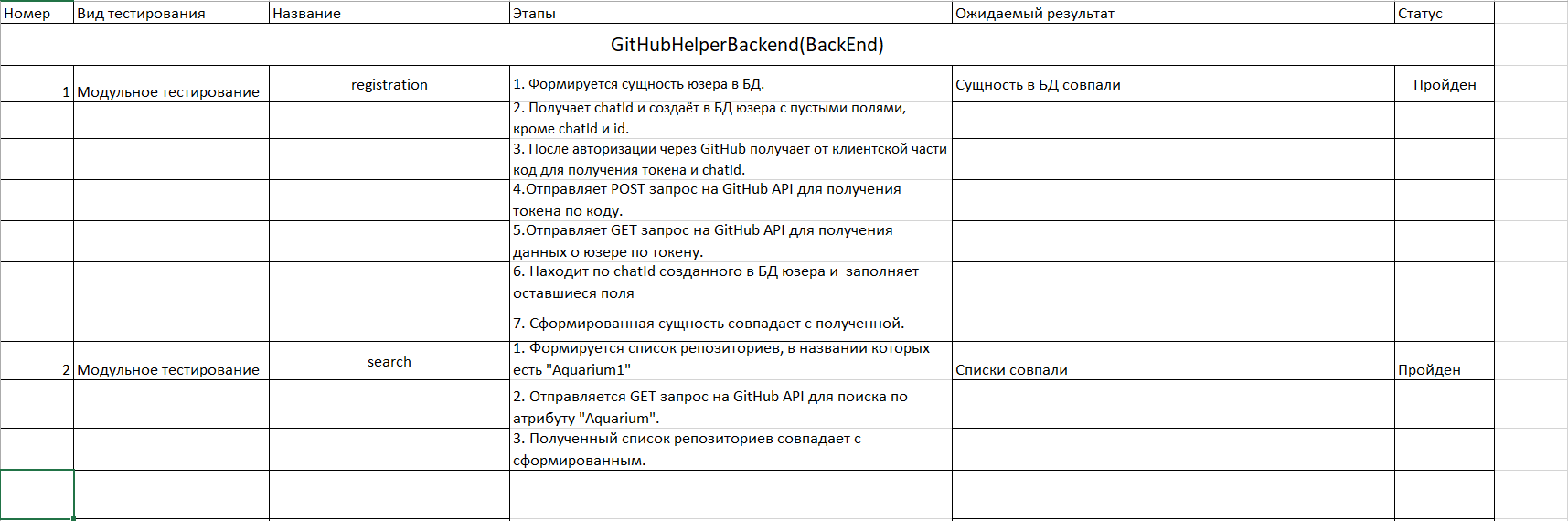
Для проверки работоспособности функций подсистем было проведено автоматизированное модульное тестирование. Тестовые сценарии и результаты выполнения данного вида тестирования для серверной части системы представлены на рисунке 17.

Рисунок 17 - Модульное тестирование для серверной части

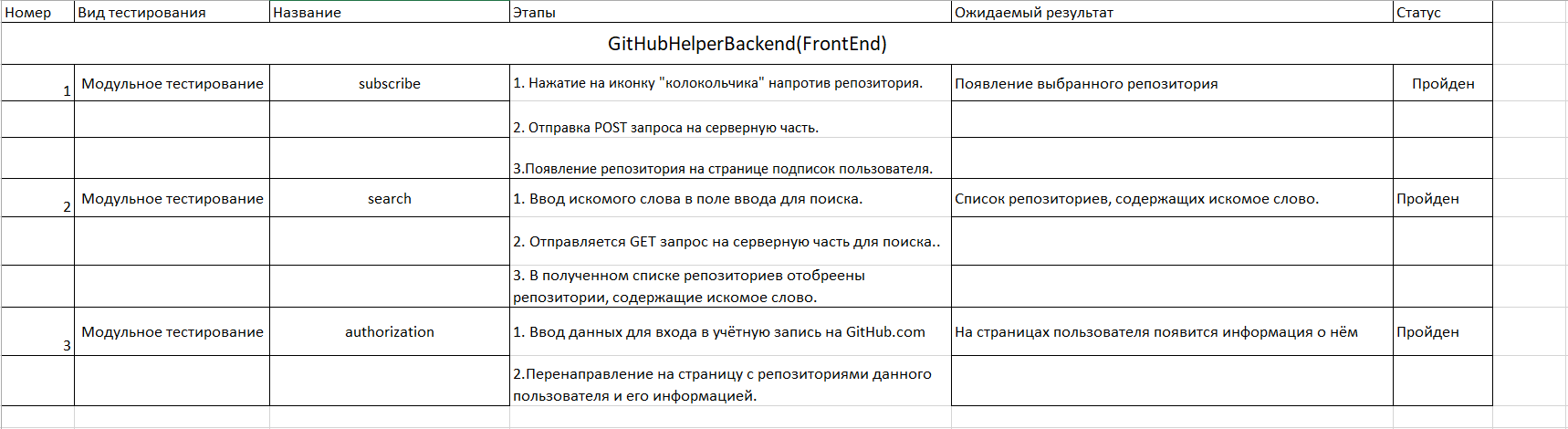
На рисунке 18 приведены тестовые сценарии и результаты модульного тестирования клиентской части приложения.

Рисунок 18 - Модульное тестирование для клиентской части

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

## Системное тестирование

Чтобы убедиться, что приложение работает корректно для всех возможных браузеров, были проведены тесты по заранее составленным сценариям. Сценарий проверяет основной функционал сервиса, реализованный на клиентской части.

Тесты были проведены в следующих браузерах:

* GoogleChrome;
* Yandex Browser;
* Mozilla FireFox;
* Vivaldi;

Результаты показали, что приложение корректно работает для всех представленных браузеров.

Также было проведено end-to-end тестирование для незарегистрированного и зарегистрированного пользователей. Тестовые сценарии и результаты представлены на рисунке 19. Как видно, тестирование прошло успешно.

Рисунок 19 - End-to-end тестирование

## Вывод

Исходя из результатов тестов, не было обнаружено серьёзных ошибок или ситуаций, в которых сервис работал бы некорректно.

# Заключение

В ходе написания курсовой был разработан сервис, реализующий процесс уведомления его пользователей о новых коммитах с помощью telegram-бота. Для это был реализован следующий функционал:

* Уведомление пользователей о коммитах сразу после их создания;
* Поиск репозиториев по названию;

В процессе разработки сервиса были получены:

* Опыт в командной работе;
* Знания в области web-разработке;
* Опыт в планировании разработки проекта;

# Список использованных источников

1. Алимова Т. Диверсификация деятельности малых предприятий // Вопросы экономики. 1997. - № 6. - С. 130-137.
2. Аванесова Г.А. Сервисная деятельность. – М.: Аспект-пресс, 2010. – 467с.
3. Уоллс К. Spring в действии. / К. Уоллс – Москва: Техносфера, 2013. – 138 с.
4. Хеффельфингер Д. Java EE 6 и сервер приложений GlassFish 3. / Д. Хеффельфингер – СПб.: Питер, 2013. – 86 с.
5. Шефер К. Spring 4 для профессионалов. / К. Шефер, К. Хо, Р. Харроп - Москва.: Техносфера, 2015. – 109 с.
6. Файн Я. Angular и TypeScript. Сайтостроение для профессионалов. / Я. Файн А. Моисеев – СПб.: Питер, 2018 – 19 с.
7. Блох Д. Java. Эффективное программирование. / Д. Блох – СПБ.: БХВ-Питер, 2014 – С. 184-192
8. Гонсалвес Э. Изучаем Java EE 7. / Э. Гонсалвес – Москва: Инфа –М, 2016. – С. 237-359
9. Основы Hibernate – URL: <https://habr.com/ru/post/29694> (дата обращения 27.02.2019).
10. Rest на примере Spring MVC – URL: <https://devcolibri.com/rest-на-примере-spring-mvc> (дата обращения 07.05.2019).
11. Краткое руководство: связываем ASP.NET Core Web API + Angular 5 – URL: <https://habr.com/ru/post/349522> (дата обращения 15.05.2019)
12. Раздел о языке программирования Java - URL: https://metanit.com/java (дата обращения 08.052019)