МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Факультет *компьютерных наук*

Кафедра *программирования и информационных технологий*

Разработка Telegram бота «GitHubHelper»

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, ассистент*

Воронеж 2019

Содержание

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 разработчиков 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.

Целью данного курсового проекта была поставлена разработка бота для мессенджера в связке с web-приложением для его конфигурации пользователем.

Для достижения цели были определены следующие задачи:

— Определение требований
— Проектирование приложения
— Разработка базы данных для хранения пользовательской информации.
— Разработка программной реализации бота для мессенджера для оповещения пользователя.
— Разработка программной реализации web-приложения.
— Тестирование.

Используемые определения

«Сайт» - часть проекта «GitHubHelper», являющийся web-приложением с использование базы данных SQL.

«Bot» - часть проекта «GitHubHelper», специальная программа, выполняющая автоматически и/или по заданному расписанию какие-либо действия через интерфейсы, предназначенные для людей. В данном случае является чат-ботом, предоставляющим определенный функционал для взаимодействия с web-сервисом GitHub.

«Гость» - неавторизованный на портале человек, пользующийся ограниченным функционалом сайта и не имеющий доступа к функционалу чат-бота.

«Пользователь» - авторизованный на портале человек, пользующийся функционалом сайта и бота.

«Репозиторий»- репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранящие операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы.

«Коммит» - (от англ. commit - зд. фиксировать) - сохранение, фиксация (в архиве, репозитории и др.) изменений в программном коде.

1. Постановка задачи

Целью курсового проекта является разработка бота, способного оповещать пользователя о коммитах путем отправки сообщений в специальный чат об изменениях в указанных им репозиториях. Также необходимо создать веб-приложение, с помощью которого можно настроить данные оповещения.

Система должна хранить информацию о репозиториях, которые выбрал пользователь для оповещения. Данные сведения включают:

— Идентификатор репозитория;

— Полное название;

— Ссылку на страницу репозитория на GitHub;

— Логин владельца репозитория;

— Ссылка на страницу создателя на GitHub;

— Дату последнего изменения содержания;

— Описание

Также в системе необходимо хранить данные о пользователях:

— Идентификатор пользователя

— Логин, под которым пользователь зарегистрирован на GitHub

— Ссылка на страницу данного пользователя на GitHub

— ChatId – уникальный код, присвоенный индивидуально каждому пользователю ботом

— Имя

У системы нет необходимости хранить пароль от учётной записи пользователя на GitHub, поскольку была реализована авторизация напрямую через данный ресурс.

При добавлении пользователем репозитория в список, интересующих его создаётся соответствующая запись, содержащая идентификаторы пользователя и репозитория.

Таким образом, в системе предусмотрены два уровня доступа:

— Гость

— Пользователь

Также системой предусмотрены следующие ограничения:

— Пользователь может привязать к аккаунту бота только один аккаунт GitHub

— Гость не может просмотреть список репозиториев, не авторизовавшись

— Пользователь может настроить уведомления только через веб-приложение

Система должна соответствовать следующим требованиям:

— Использование Git API.

— Оповещение в реальном времени об изменениях.

— Авторизация на сайте посредством использования Github.

— Возможность поиска репозиториев других пользователей.

Завершенный проект представляет собой полностью функционирующего бота и web-приложение, соответствующие требованиям, описанным выше.

2. Анализ

2.1. Анализ предметной области

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.

В первую очередь необходимо определить возможности пользователя разрабатываемого приложения:

— авторизация с помощью аккаунта Github;

— настройка списка репозиториев, о которых пользователь системы хочет получать оповещения;

— поиск репозиториев.

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

В качестве приложения, которое будет оповещать пользователя об изменениях выбранных им репозиториев, несмотря на другие возможные способы реализации, был выбран бот для мессенджера.

Мессенджеры приватны, требуют меньше ресурсов, работают на более дешевых устройствах и пока еще, в отличие от соцсетей, не забиты лишней информацией, навязчивой рекламой и чужими новостями.

Бот в мессенджере не требует дополнительной загрузки, не занимает место на экране мобильного телефона или компьютерного стола, не занимает места в памяти устройства.

В качестве мессенджера было решено использовать Telegram, даже не смотря на попытки Роскомнадзора заблокировать его на территории Российской Федерации, по следующим причинам:

— огромная пользовательская база, после запрета на использование выросшая еще больше;
— удобное API для взаимодействия;
— запрет не распространяется на запуск ботов для данного мессенджера.

2.2. Анализ существующих решений

1. Telegram бот, оповещающий о тригеррах Zabbix.

Плюсы:

— Выбор способа оповещения;
— Оповещения в реальном времени;
— Наличие пользовательского интерфейса;

Минусы:

— Необходимость вручную писать скрипт для критериев оповещения;
— Создаются файлы со скриптом на диске, занимая пространство.

2. Slack с интеграцией Github.

Плюсы:

— Оповещения в реальном времени;
— Интеграция с многими платформами;

Минусы:

— Slack – платная платформа, что может принести неудобства для начинающих программистов;
— Отсутствие полноценной поддержки видеозвонков и общих чатов(общие чаты только через добавление в команды клиентов, партнеров);

— Отсутствие web-приложения с пользовательским интерфейсом.

2.3.Диаграмма вариантов использования

После анализа предметной области и определения необходимых возможностей пользователя, необходимо разработать диаграмму вариантов использования данной системы. Диаграмма вариантов использования представлена на рисунке 1.

Рисунок 1. Диаграмма вариантов использования

Неавторизованный пользователь обладает следующими возможностями:

— подписка на Telegram-бот;

Авторизованный пользователь обладает следующими возможностями:

— просмотр собственных репозиториев;

— получение уведомлений о новых коммитах;

— отписка от Telegram-бота;

— просмотр истории коммитов;

— поиск репозиториев по названию;

— добавление репозитория в список

— просмотр списка выбранных репозиториев;

— удаление репозитория из списка;

2.4.Схема базы данных

После того, как были определены роли и варианты использования системы, необходимо определить, какие данные надо хранить в базе данных. Схема базы данных представлена на рисунке 2.

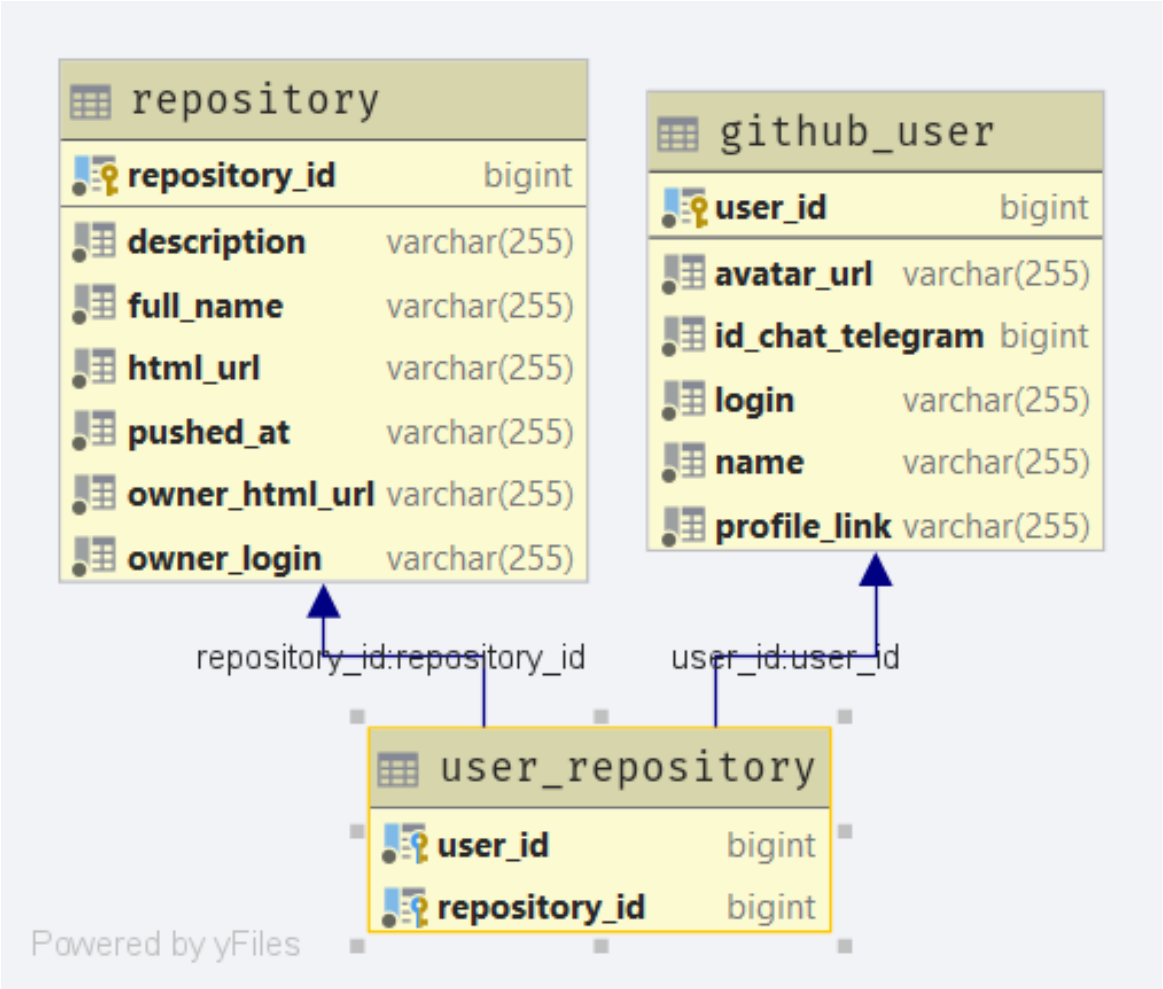

Рисунок 2. Схема базы данных

Рассмотрим каждую таблицу подробнее:

1. Repository – таблица, которая хранит в себе список репозиториев, на которые подписаны пользователи. В ней присутствуют следующие поля:
   a. Repository_id – идентификатор репозитория.
   b. Description – описание репозитория.

c. Full_name – полное имя репозитория (вида userLogin/repositoryName).
d. Html_url – ссылка на данный репозиторий.
e. Pushed_at – дата, когда был создан репозиторий.
f. Owner_html_url – ссылка на владельца репозитория.
g. Owner_login – логин владельца репозитория.

2. Github_user – таблица, хранящая в себе информацию о пользователях нашего приложения. Каждый пользователь имеет следующие поля:
   a. User_id – идентификатор пользователя.
   b. Avatar_url – ссылка на аватар пользователя.
   c. Id_chat_telegram – идентификатор, который используется телеграммом для идентификации.
   d. Login – GitHub логин пользователя
   e. Name – имя пользователя
   f. Profile_link – ссылка на профиль пользователя
3. User_repository – таблица, реализующая отношение многие ко многим. Таким образом, в ней содержатся всего два поля:
   a. User_id – идентификатор пользователя.
   b. Repository_id – идентификатор репозитория.

2.5.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. Преимуществом этого фреймворка является наличие множества готовых компонентов, использование которых значительно упрощает процесс разработки клиентской части. 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 доступной, лёгкой в использовании и 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.

2.6.Графическое описание работы системы

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

2.6.1. Диаграммы состояний

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

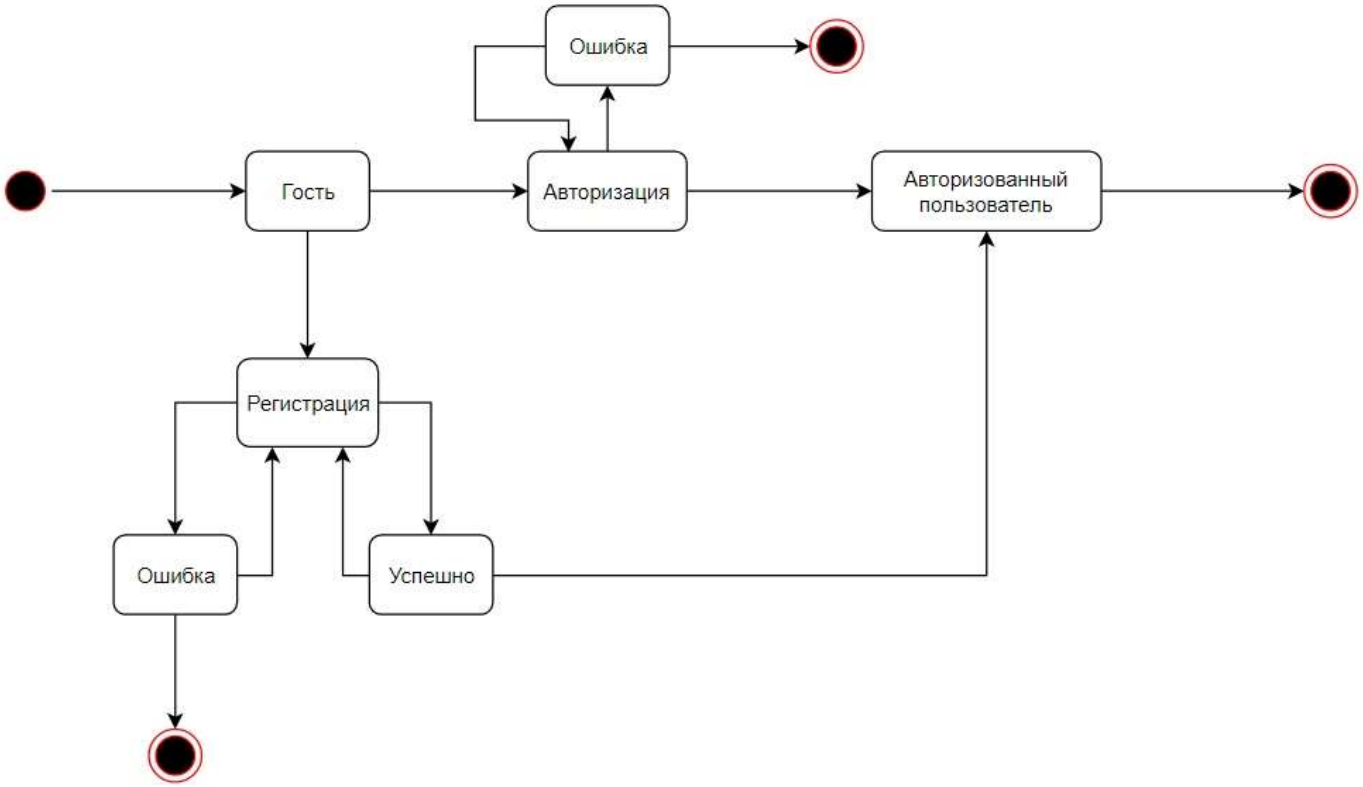

Рисунок 3. Диаграмма состояния «Авторизация»

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

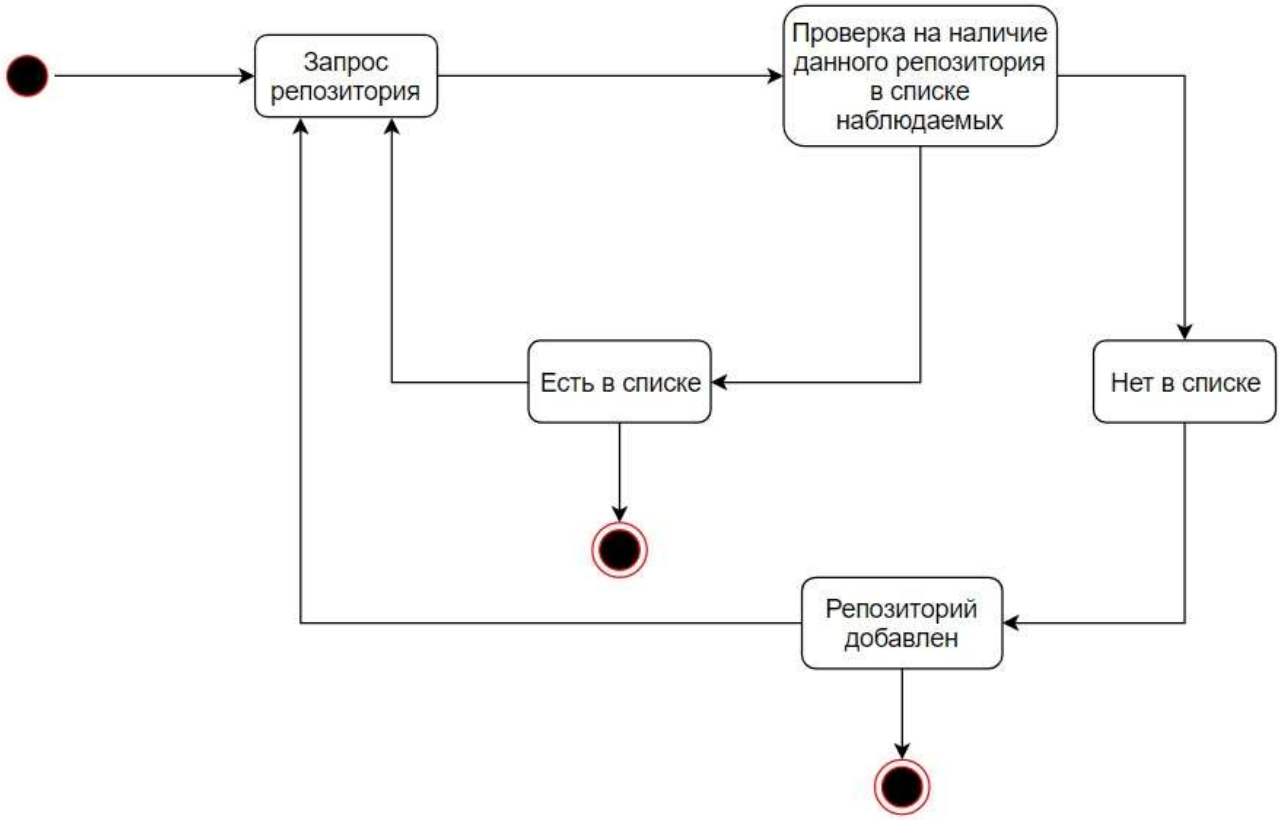

Рисунок 4. Диаграмма состояния «Добавление репозитория»

Для описания состояний, в которых находится сервер при удалении репозитория, составлена диаграмма, изображенная на рисунке 5.

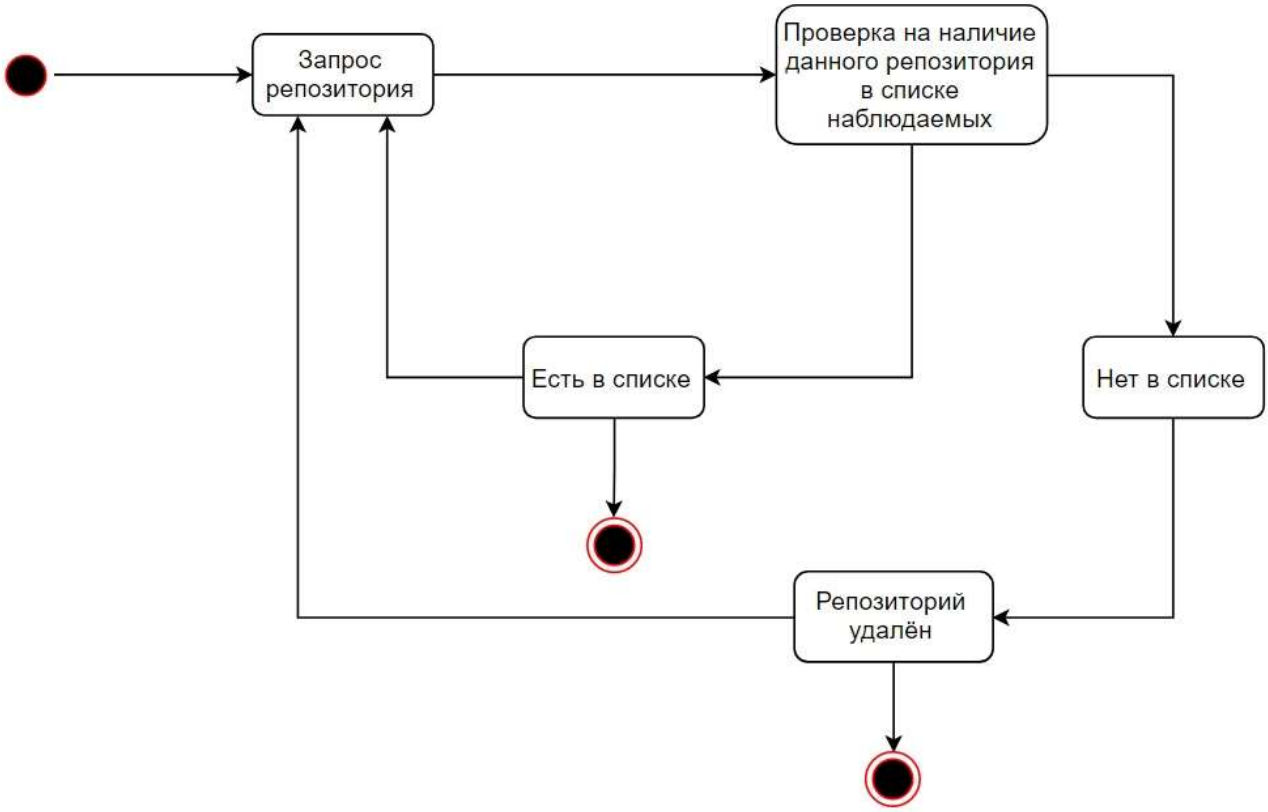

Рисунок 5. Диаграмма состояния «Удаление репозитория»

2.6.2. Диаграммы активности

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

Диаграмма активности авторизации изображена на рисунке 6. На данной диаграмме присутствуют 4 части: Guest, Sustem, Database и сторонние источники. (API).

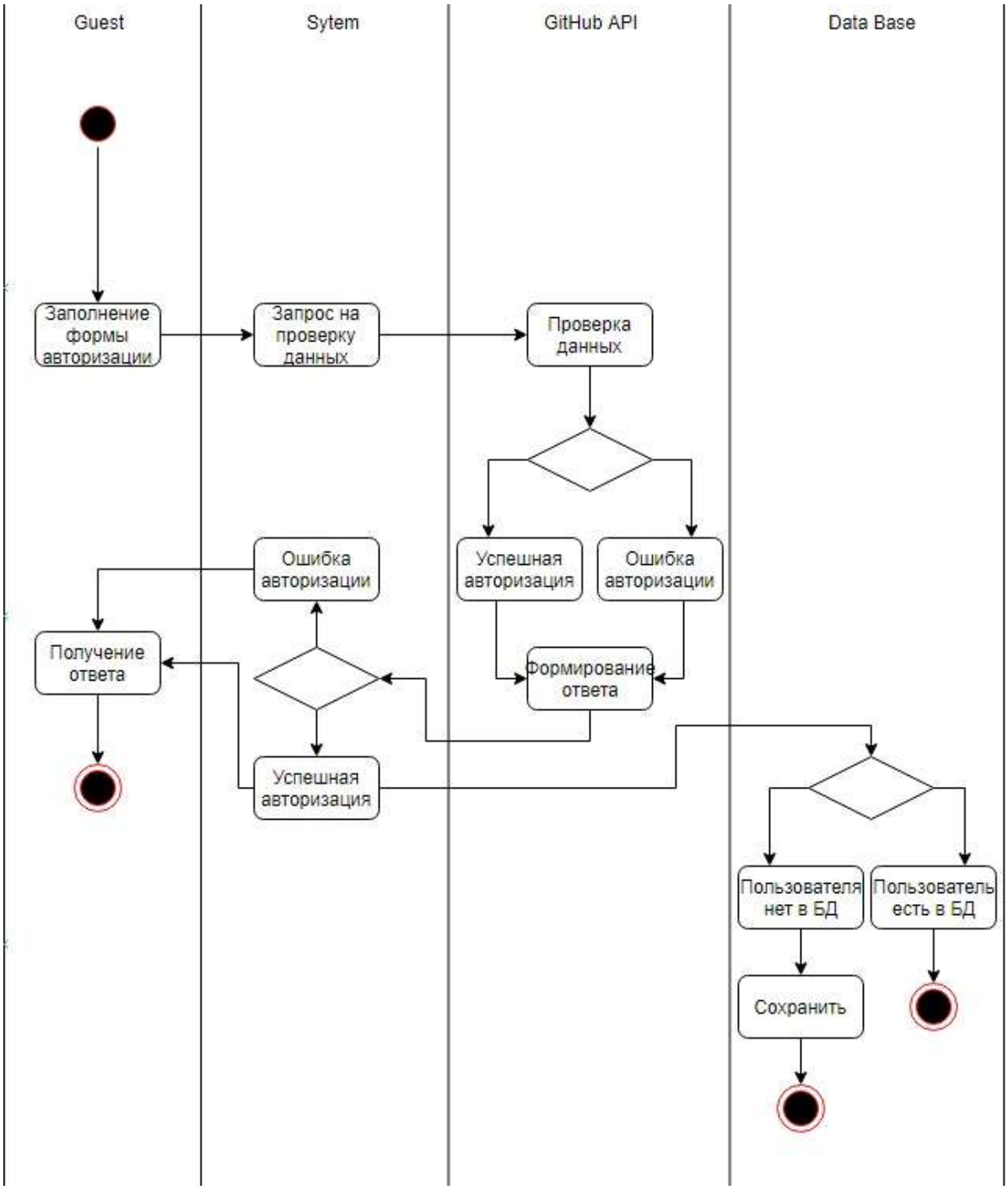

Рис. 6. Диаграмма активности «Авторизация»

Диаграмма активности добавление/удаление репозитория изображена на рисунке 7. На данной диаграмме присутствуют 3 части: User, Server и Data base.

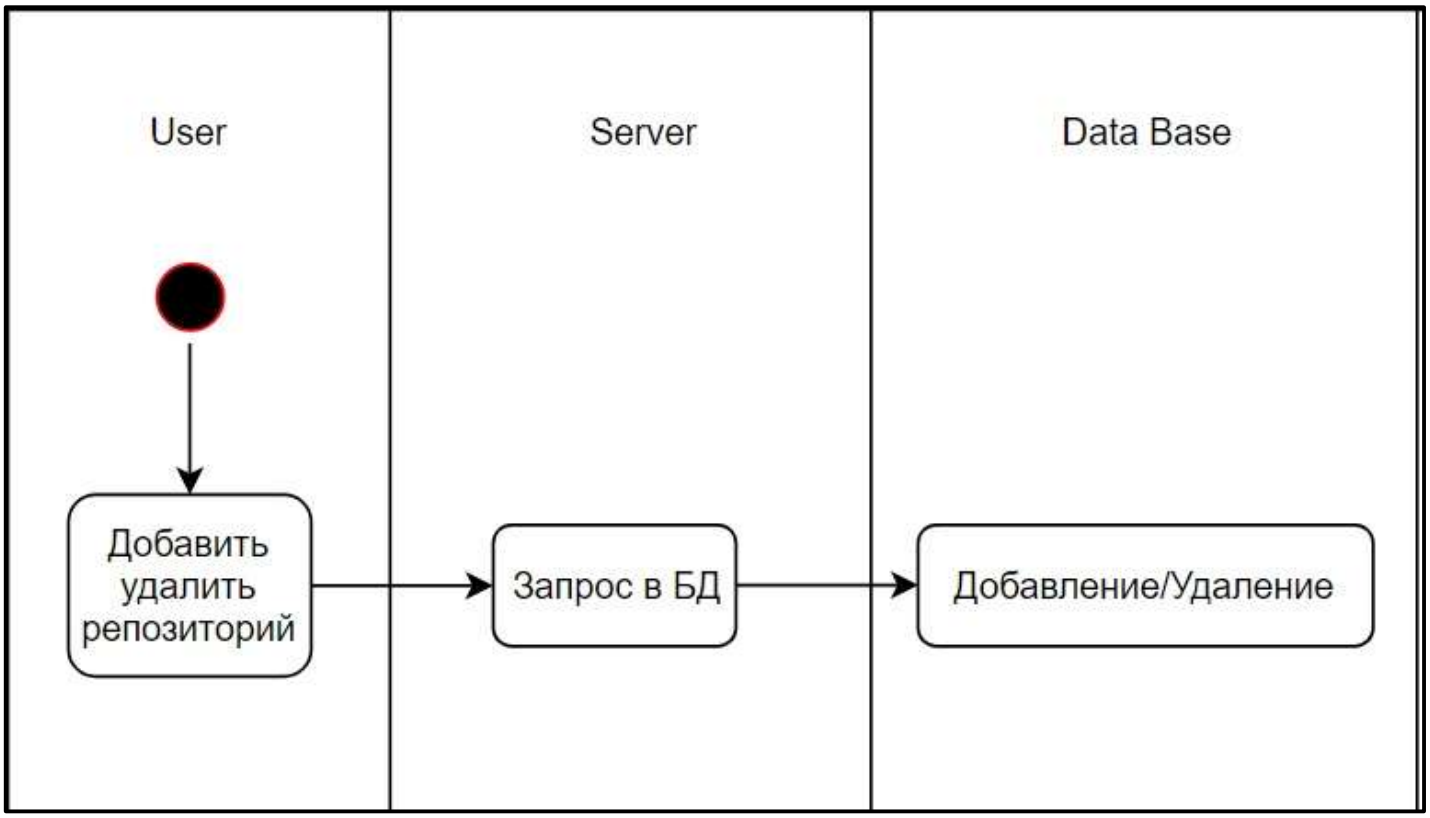

Рис. 7. Диаграмма активности «Добавление/удаление репозитория»

Диаграмма активности регистрации изображена на рисунке 8. На данной диаграмме присутствуют 3 части (дорожки): пользователь(User), приложение(System), база данных (Data Base) и Github API.

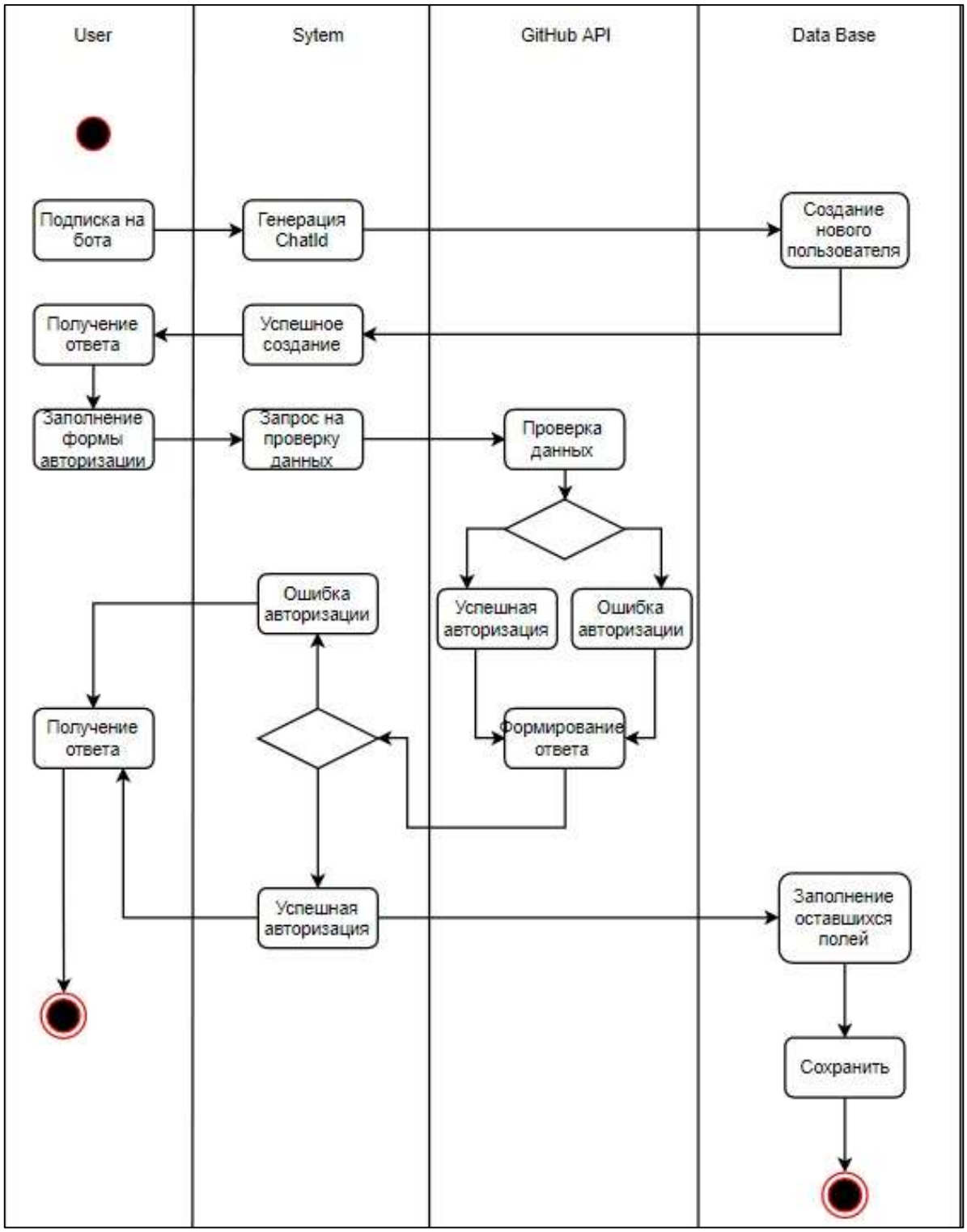

Рис. 8. Диаграмма активности «Регистрация»

2.6.3. Диаграммы последовательностей

Диаграмма последовательностей авторизации изображена на рисунке 9.

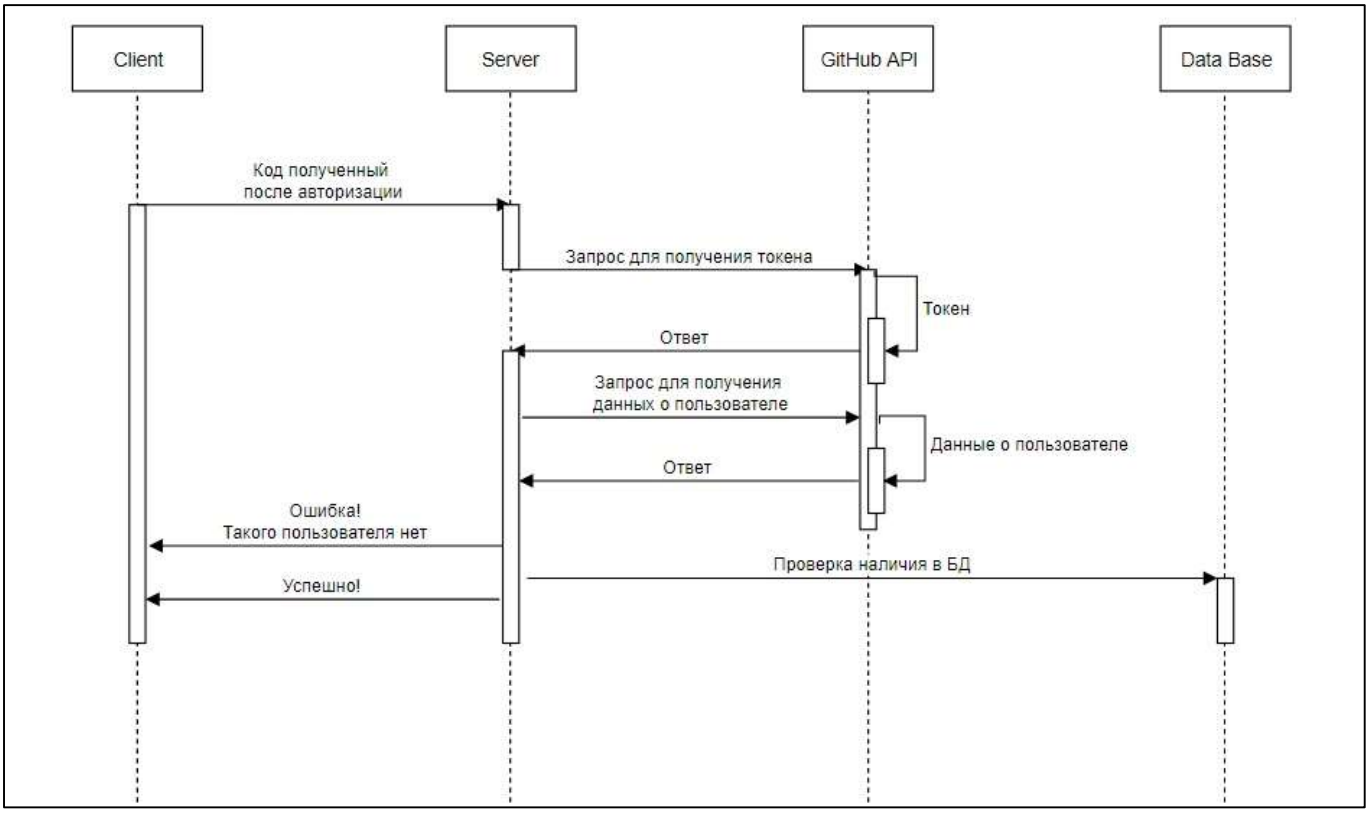

Рис. 9. Диаграмма последовательности «Авторизация»

Диаграмма последовательности регистрации изображена на рисунке 10.

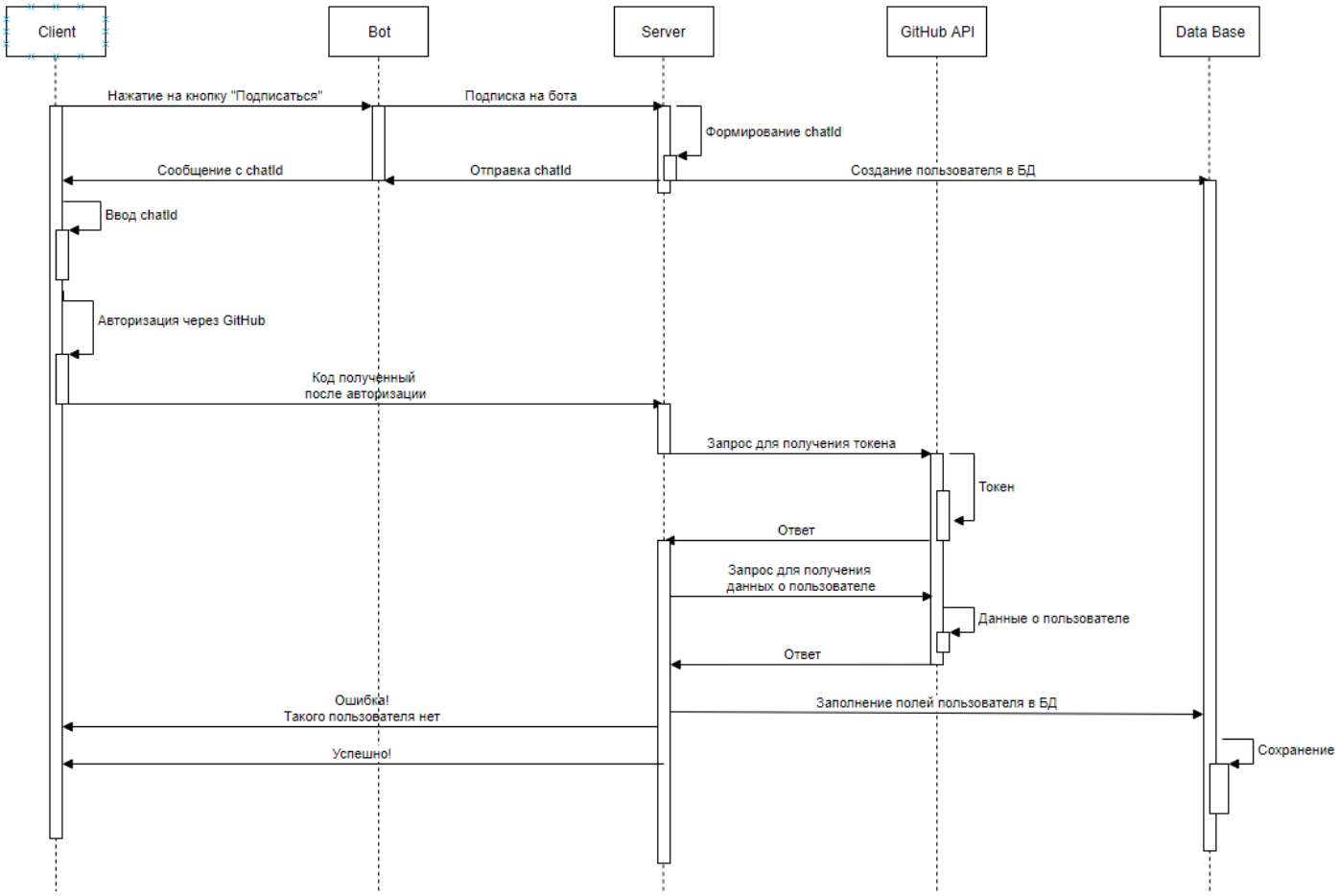

Рис. 10. Диаграмма последовательности «Регистрация»

Диаграмма последовательностей добавление/удаление изображена на рисунке 11.

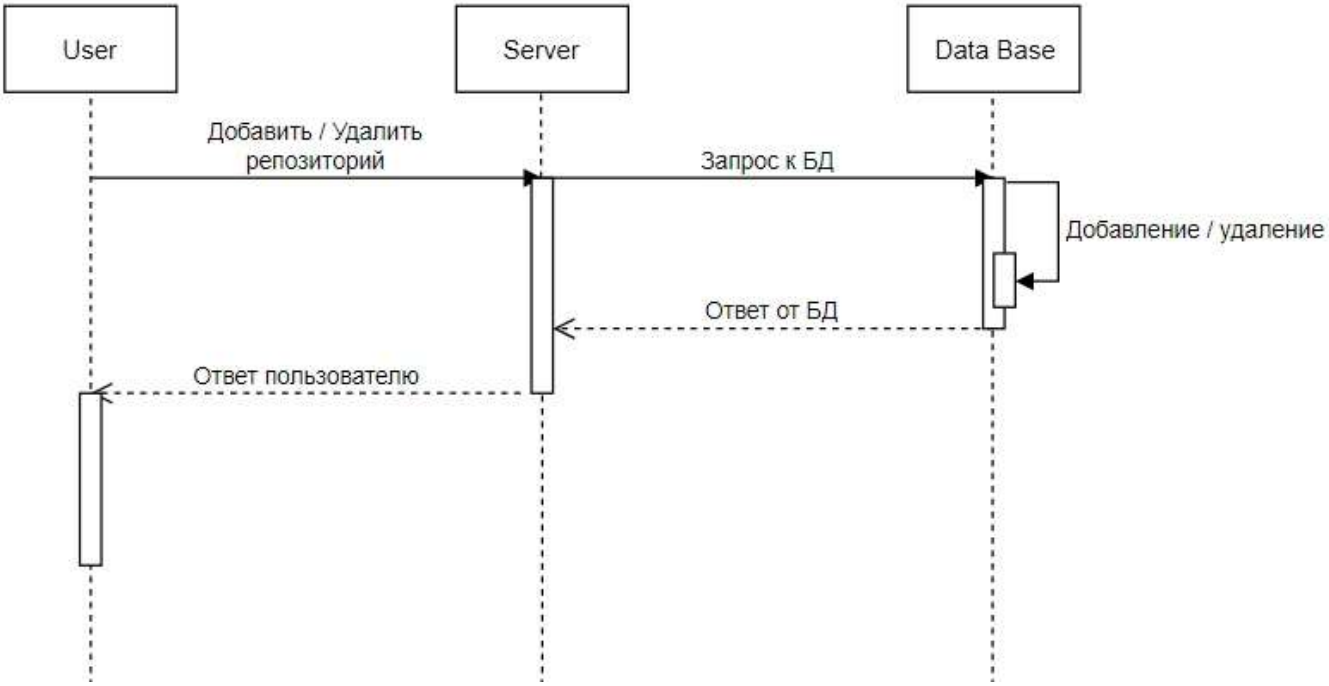

Рис. 11. Диаграмма последовательности «Добавление/Удаление репозитория»

2.6.4. Диаграмма развертывания

Приведенная на рисунке 12 диаграмма визуализирует элементы и компоненты программы, которые существуют на этапе ее исполнения.

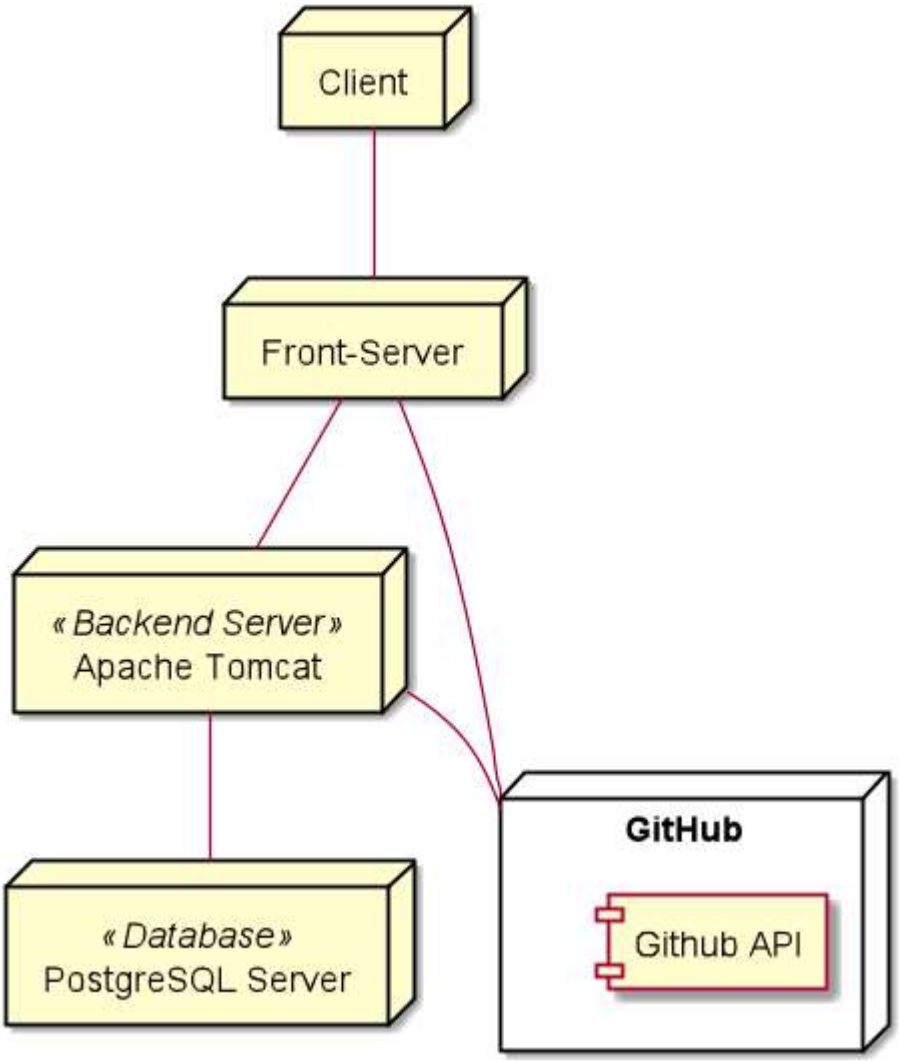

Рис. 12. Диаграмма развертывания

3. Реализация

В данной главе описана реализация серверной части приложения.

3.1.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.

3.2.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.

3.3.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.

3.4.Слой DAO

DAO – data access object. Это объект, который предоставляет абстрактный интерфейс к какому-либо типу базы данных или механизму хранения. В данной задаче 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.

3.5.Слой Controller

Этот слой, пожалуй, самый важный из всех перечисленных, 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:

- RepositoryController – используется для взаимодействия с репозиториями
- RegistrationController – используется для регистрации.

3.6.Оповещения

Оповещения были реализованы с помощью Webhooks. Это технология, позволяющая получать различную информацию от Github в реальном времени.

Для корректной работы было необходимо настроить сервер, после чего на него будут приходить POST запросы, в теле которых будет содержаться информация о коммитах. Эти запросы обрабатывает UserController, вызывающий соответствующие методы бота для оповещения пользователя.

3.7. Telegram-бот

Для написания telegram-бота использовалось официальное API для работы с telegram. Были реализованы два класса:

- BotTelegram
- TelegramGithHubBotApi

На рисунке 13 изображена диаграмма классов для бота.

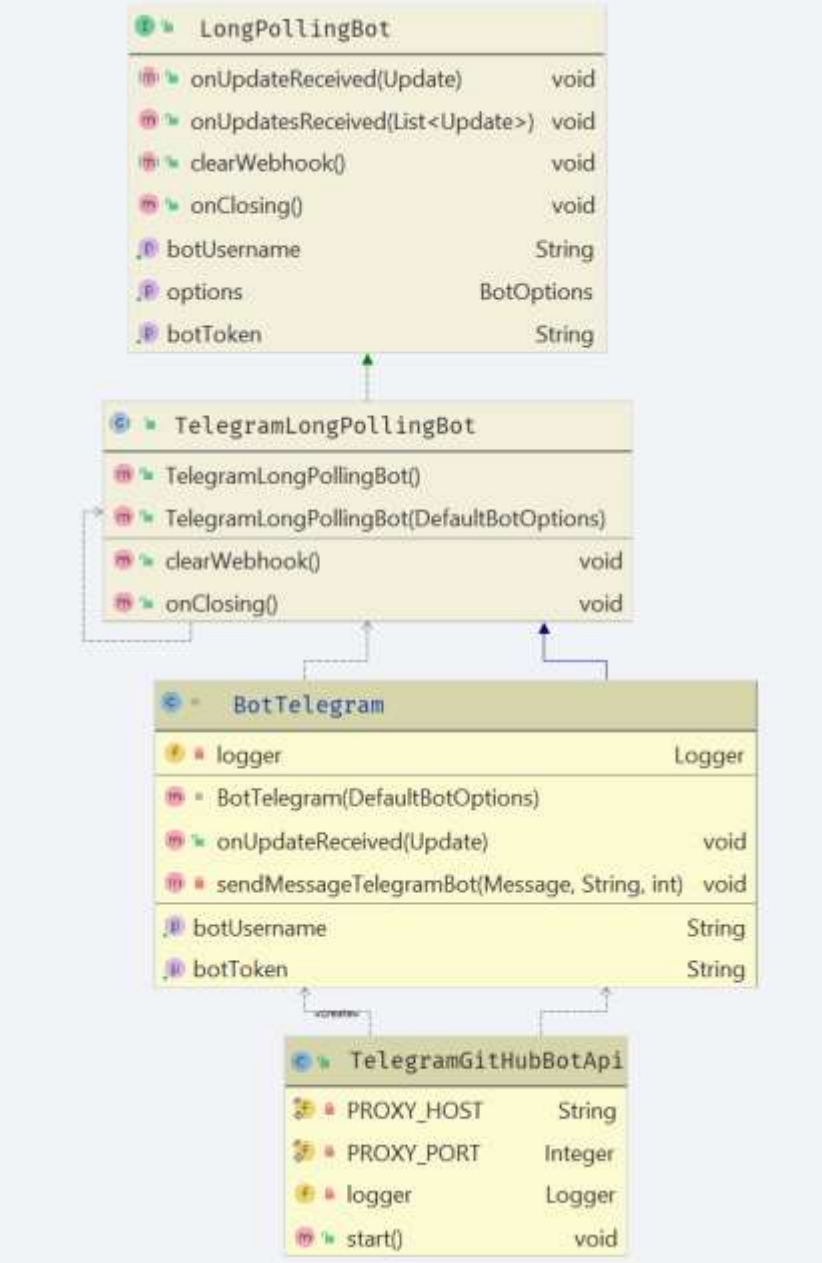

Рисунок 13 – Диаграмма классов для telegram-бота

Стоит отметить, что класс TelegramLongPollingBot и интерфейс LongPollingBot – части API, а не результат работы нашей команды.

В приложении создан специально создан класс TelegramGutHubBotApi, что бы при использовании разработчики могли не вдаваться в подробности реализации самого бота. Достаточно будет просто создать объект класса упомянутого выше и вызвать метод start(). Основным методом в классе BotTelegram является переопределённый метод onUpdateRecived(Update). Это обработчик события. Когда боту поступают сообщения или появляется новый пользователь, то вызывается именно этот метод.

Данного бота можно найти по уникальному chat_id @GitHubHelperTgBot.

4. 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 14, находится информация о данном сервисе и кнопка «Войти». После нажатия на неё пользователя переадресовывает на страницу авторизации через GitHub, это необходимо для того, чтобы работать с данными о пользователе. После ввода данные проверяются и пользователя переадресовывают на страницу регистрации (рисунок 15) или на страницу пользователя (рисунок 17) в

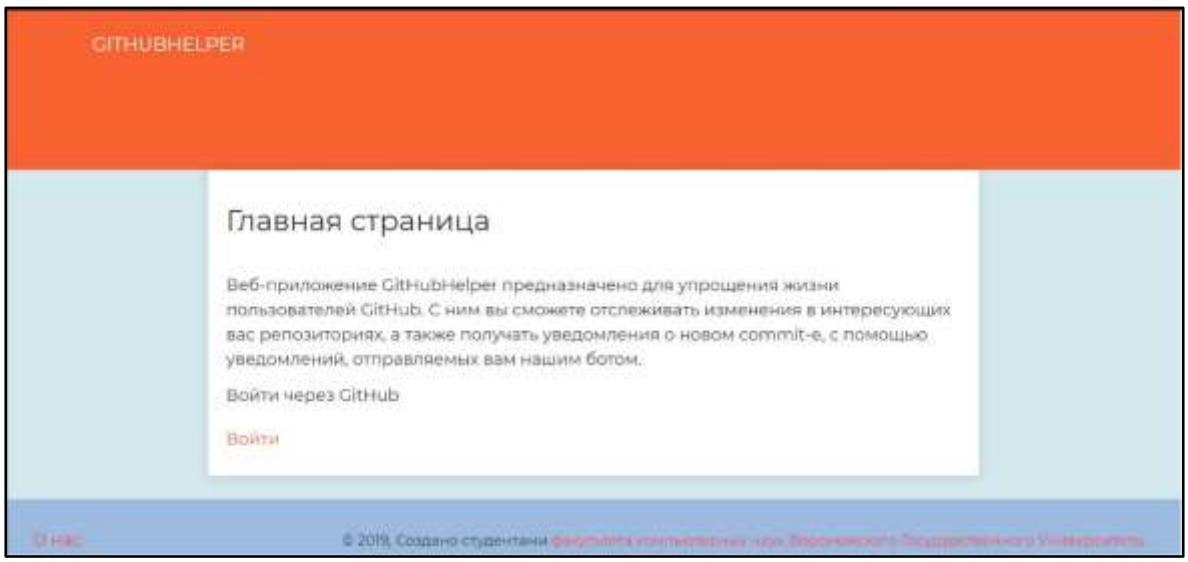

Рисунок 14 - Интерфейс главной страницы

зависимости от того, был ли он зарегистрирован ранее.

На данной странице (рисунок 15) пользователь может зарегистрироваться в данном сервисе. Для этого на странице находится ссылка на Telegram-бота. После перехода по ней пользователь попадает в чат с Telegram ботом, где он получает chatId, необходимый для дальнейшей работы на сайте. На рисунке 16 изображен скриншот чат-бота.

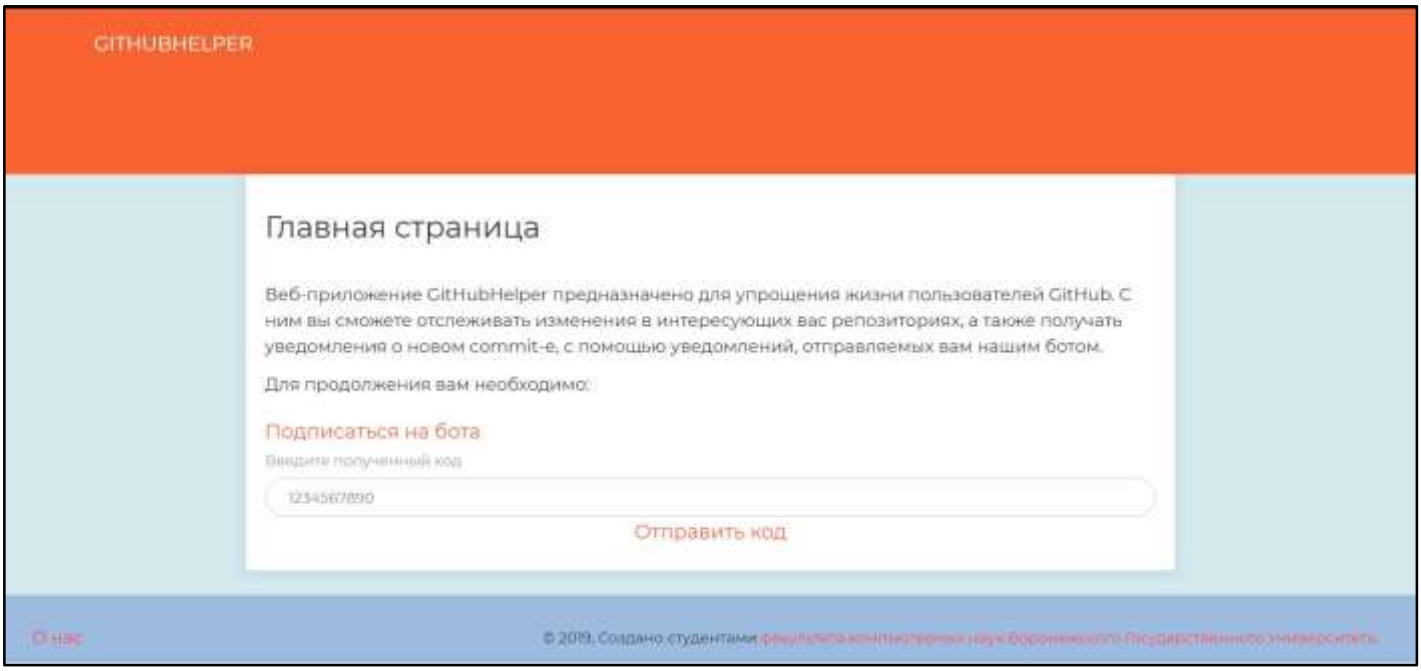

Рисунок 15 - Интерфейс главной страницы после авторизации

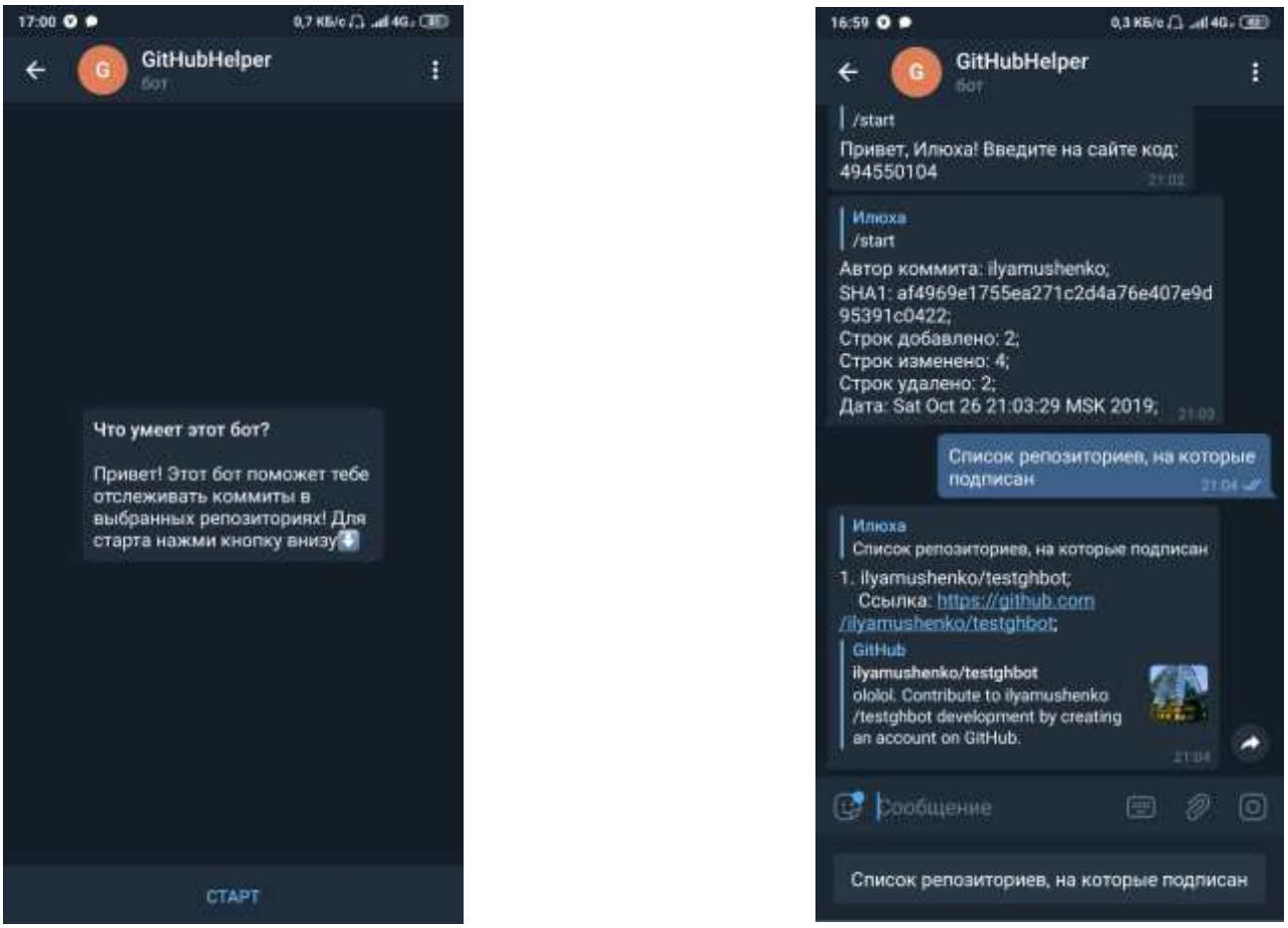

Рисунок 16 - Скриншоты Telegram бота "GitHubHelper"

На этой странице находится информация о пользователе и его репозиториях. Её внешний вид представлен на рисунке 17.

Пользователь может:

— Просмотреть сведения о каждом репозитории;

— Подписаться на уведомления о коммитах даного репозитория;

— Увидеть список коммитов для каждого конкретного репозитория;

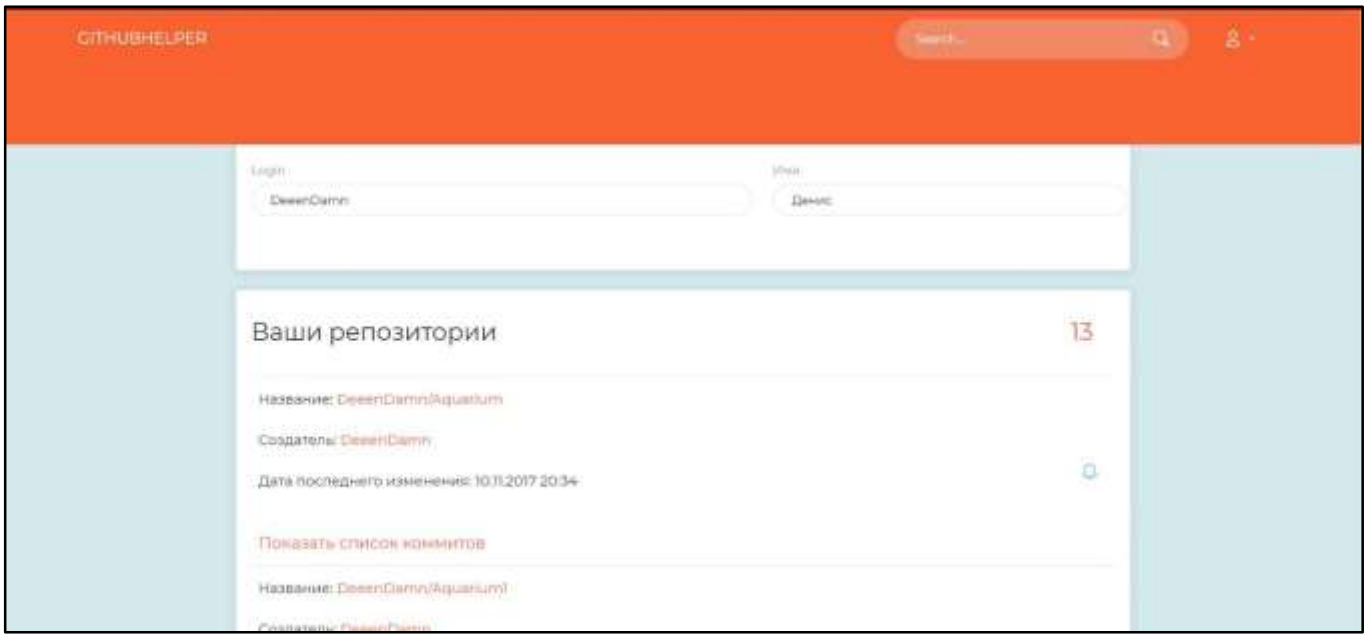

Рисунок 17 - Страница с репозитория пользователя

Также практически на каждой странице есть возможность воспользоваться поиском и попасть на страницу с его результатами, которая показана на рисунке 18.

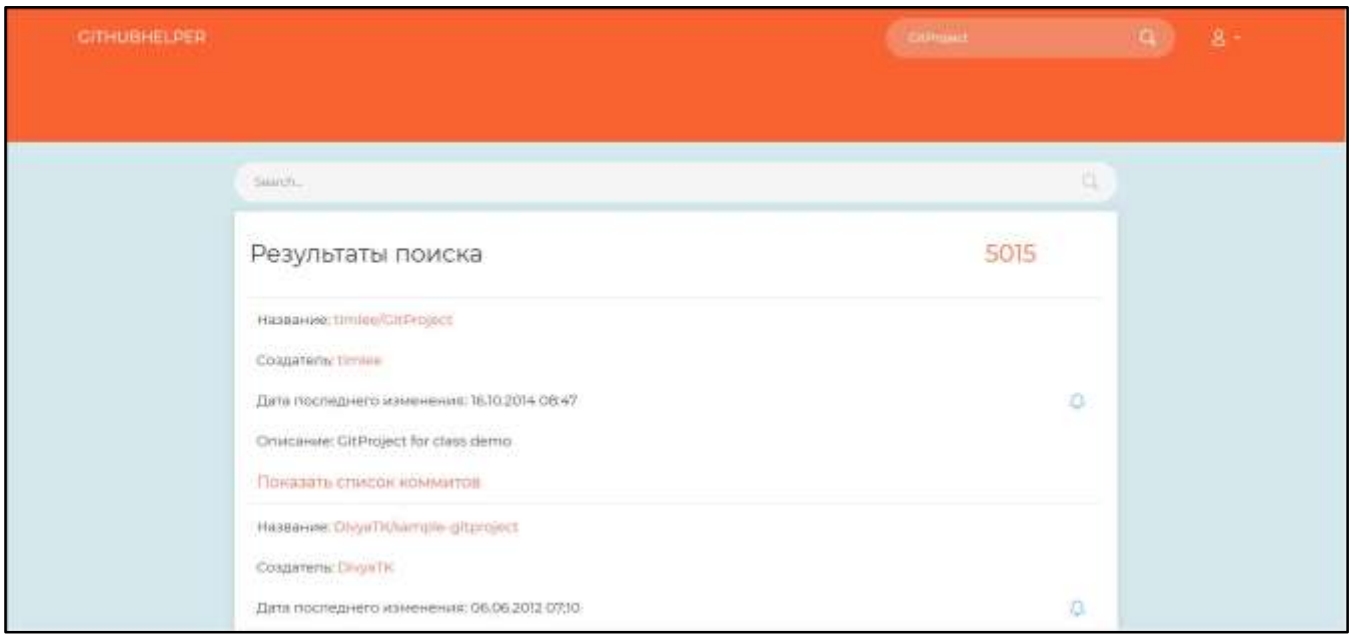

Рисунок 18 - Страница с результатами поиска

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 19.

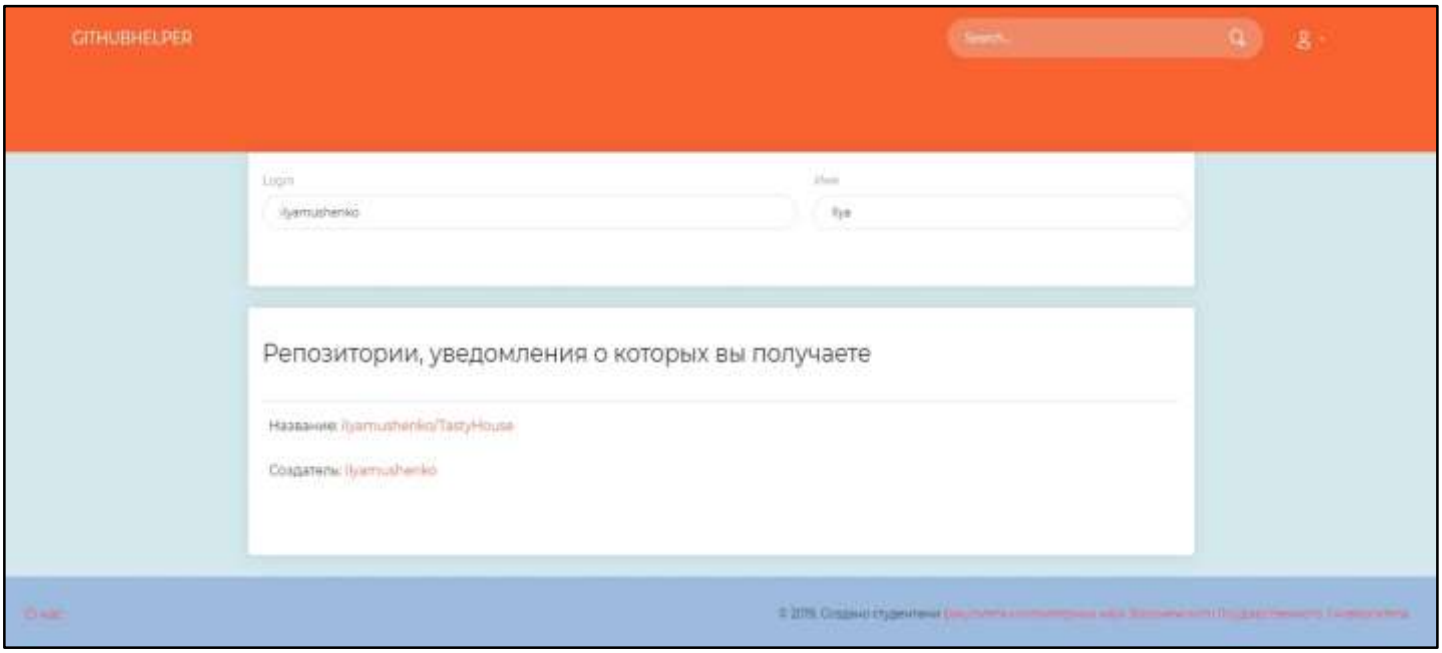

Рисунок 19 - Страница с репозиториями, на которые подписан пользователь

5. 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;

— E2E тестирование;

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 20.

Номер	Вид тестирования	Название	Этапы	Ожидаемый результат	Статус
GitHubHelperBackend(BackEnd)					
1	Модульное тестирование	registration	1. Формируется сущность юзера в БД.	Сущность в БД совпали	Пройден
			2. Получает chatId и создаёт в БД юзера с пустыми полями, кроме chatId и id.		
			3. После авторизации через GitHub получает от клиентской части код для получения токена и chatId.		
			4.Отправляет POST запрос на GitHub API для получения токена по коду.		
			5.Отправляет GET запрос на GitHub API для получения данных о юзере по токену.		
			6. Находит по chatId созданного в БД юзера и заполняет оставшиеся поля		
			7. Сформированная сущность совпадает с полученной.		
2	Модульное тестирование	search	1. Формируется список репозиториев, в названии которых есть "Aquarium1"	Списки совпали	Пройден
			2. Отправляется GET запрос на GitHub API для поиска по атрибуту "Aquarium".		
			3. Полученный список репозиториев совпадает с сформированным.		

Рисунок 20 - Модульное тестирование для серверной части

На рисунке 21 приведены тестовые сценарии и результаты модульного тестирования клиентской части приложения.

Номер	Вид тестирования	Название	Этапы	Ожидаемый результат	Статус
GitHubHelperBackend(FrontEnd)					
1	Модульное тестирование	subscribe	1. Нажатие на иконку "колокольчика" напротив репозитория.	Появление выбранного репозитория	Пройден
			2. Отправка POST запроса на серверную часть.		
			3.Появление репозитория на странице подписок пользователя.		
2	Модульное тестирование	search	1. Ввод искомого слова в поле ввода для поиска.	Список репозиториев, содержащих искомое слово.	Пройден
			2. Отправляется GET запрос на серверную часть для поиска..		
			3. В полученном списке репозиториев отобреены репозитории, содержащие искомое слово.		
3	Модульное тестирование	authorization	1. Ввод данных для входа в учётную запись на GitHub.com	На страницах пользователя появится информация о нём	Пройден
			2.Перенаправление на страницу с репозиториями данного пользователя и его информацией.		

Рисунок 21 - Модульное тестирование для клиентской части

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

5.2.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.

5.3.End-to-end тестирование

Также было проведено end-to-end тестирование для незарегистрированного и зарегистрированного пользователей. Тестовые сценарии и результаты представлены на рисунке 22. Как видно, тестирование прошло успешно.

Роль	Действие	Результат	Решение	Статус	
Неавторизованный пользователь (гость)	Перейти по ссылки на telegram-бота	Ссылка сработала, пользователь перешёл на страницу	Соответствует ожидаемому	Пройден	
	Отправить код	Кнопка сработала, появилась новая для авторизации чере GitHub	Соответствует ожидаемому	Пройден	
	Авторизоваться через GitHub	Ссылка перенаправила пользователя на страницу приложения	Соответствует ожидаемому	Пройден	
Авторизованный пользователь	Ввести название искомого репозитория в поле поиска	Откроется страница с результатами поиска по данному запросу	Соответствует ожидаемому	Пройден	
	Нажать на "Посмотреть список коммитов"	Открылся список коммитов конкретного репозитория	Соответствует ожидаемому	Пройден	
	Нажать на название репозитория/логин пользователя	Открылась страница репозитория/юзера на github.com	Соответствует ожидаемому	Пройден	
	Нажать на иконку "колокольчика"	Репозиторий был добавлен с список интересующих пользователя	Соответствует ожидаемому	Пройден	

Рисунок 22 - End-to-end тестирование

5.4.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;

— Авторизация с помощью Github

В процессе разработки сервиса были получены:

— Опыт работы в команде;

— Знания в области web-разработке;

— Опыт в планировании разработки проекта;

Отчёт по ролям

1. **Техническое Задание**

   Работали: Мущенко И. В., Старкин М. В.

   Ответственный: Старкин М. В.

2. **Описание предметной области**

   Работали: Мущенко И. В., Старкин М. В.

   Ответственный: Старкин М. В.

3. **Отчётный документ**

   Работали: Кравченко Д. А., Мущенко И. В., Старкин М. В.

   Ответственный: Кравченко Д. А.

4. **Диаграмма вариантов использования**

   Работали: Кравченко Д. А.

   Ответственный: Кравченко Д. А.

5. **Модульная схема**

   Работали: Кравченко Д. А., Старкин М. В.

   Ответственный: Кравченко Д. А.

6. **Диаграмма активности, последовательностей и состояний**

   Работали: Кравченко Д. А., Мущенко И. В.

   Ответственный: Кравченко Д. А.

7. **Диаграмма развёртывания**

   Работали: Мущенко И. В.

   Ответственный: Мущенко И. В.

8. **Диаграмма классов для бота**

   Работали: Мущенко И. В.

   Ответственный: Мущенко И. В.

9. **Схема базы данных**

   Работали: Мущенко И. В., Старкин М. В.

   Ответственный: Мущенко И. В.

10. **Frontend**

    Работали: Кравченко Д. А.

Ответственный: Кравченко Д. А.,

11. **Backend: разработка бота**

    Работали: Мущенко И. В.

    Ответственный: Мущенко И. В.

12. **Backend: связь с базой данных и сторонними источниками**

    Работали: Старкин М. В.

    Ответственный: Старкин М. В.

13. **Объединение backend-а и frontend-а**

    Работали: Кравченко Д. А., Мущенко И. В.

    Ответственный: Мущенко И. В.

14. **Интеграция swagger-а**

    Работали: Старкин М. В., Мущенко И. В.

    Ответственный: Старкин М. В.

15. **Тестирование**

    Работали: Кравченко Д. А., Старкин М. В.

    Ответственный: Кравченко Д. А.

16. **Запись демо-видео**

    Работали: Кравченко Д. А., Мущенко И. В.

    Ответственный: Мущенко И. В.

17. **Подготовка презентации**

    Работали: Кравченко Д. А., Мущенко И. В., Старкин М. В.

    Ответственный: Мущенко И. В.

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-na-primere-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)