ФЕДЕРАЛЬНОЕ АГЕНСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ

ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ

БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО

ОБРАЗОВАНИЯ

МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ

(МТУСИ)

Кафедра Информационной безопасности

Курсовая работа

По дисциплине

«Разработка безопасного программного обеспечения»

На тему

«Мобильное приложение»

Выполнили студенты группы БАС2101 Герейакаев Гаджи

Казинский Кирилл

Сердюк Данил

Проверил доц. кафедры \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2023

Оглавление

[Анализ выбранной темы 3](#_Toc145757782)

[Проектирование серверной части 5](#_Toc145757783)

[Проектирование структуры базы данных 6](#_Toc145757784)

[Разработка 7](#_Toc145757785)

[Тестирование 8](#_Toc145757786)

[Описание клиентской части 9](#_Toc145757787)

[Демонстрация функционала 10](#_Toc145757788)

# Анализ выбранной темы

Тема: Мобильное приложение агрегатор с новостной сводкой и погодой в городе пользователя на платформе iOS с использованием серверной части на Java и фреймворка Spring Bootы

1. Введение:

Мобильные приложения-агрегаторы пользуются большой популярностью из-за своей функциональности В данной курсовой работе рассматривается разработка мобильного приложения-агрегатора на платформе iOS с учетом серверной части, реализованной на Java с использованием фреймворка Spring Boot.

2. Цели и задачи:

- Целью данной работы является разработка полнофункционального мобильного приложения-агрегатора на устройствах iOS.

- Задачи включают в себя проектирование пользовательского интерфейса приложения, реализацию функциональности для загрузки, хранения и просмотра данных, а также создание серверной части для обеспечения аутентификации и авторизации пользователей.

3. Технологический стек:

- Мобильное приложение будет разработано для платформы iOS, что предполагает использование языка программирования Swift и инструментария Apple Xcode.

- Серверная часть будет создана с использованием Java и фреймворка Spring Boot, что обеспечит высокую производительность и масштабируемость приложения.

- Для хранения данных будет использоваться реляционная база данных (например, PostgreSQL или MySQL).

- Для методов, реализующих логику, будут написаны модульные тесты, которые будут выполняться во время сборки проекта.

- Взаимодействие с базой данных будет происходить при помощи фреймворка Hibernate и языка запросов HQL

4. Функциональные требования:

- Регистрация и аутентификация пользователей: Пользователи должны иметь возможность создавать аккаунты, а также входить в приложение с помощью своих учетных данных.

- Просмотр данных о погоде на ближайшие 7 дней

- Чтение последних новостей из города пользователя.

5. Ожидаемые результаты:

- Разработка мобильного приложения для iOS с функциональностью приложения-агрегатора

- Разработка серверной части на Java с использованием Spring Boot для обеспечения аутентификации, авторизации и управления данными пользователей и книг.

- Работающее приложение с возможностью регистрации, входа, просмотра погода и новостей.

# Проектирование серверной части

1. Модель *OpenWeatherMapResponse*:

- Создан класс *OpenWeatherMapResponse*, представляющий структуру данных, получаемых от OpenWeatherMap API.

- Для каждого поля в API-ответе созданы соответствующие переменные класса с аннотациями `@Getter` и `@JsonIgnoreProperties`.

- Разбиты поля на несколько внутренних классов: Weather, Main, Wind, Rain, Clouds, и Sys. Каждый из этих классов представляет подсекцию данных в ответе API и также содержит геттеры для полей.

2. Конфигурация безопасности:

- Создан класс *SecurityConfig*, который использует аннотации `@Configuration` и `@EnableWebSecurity`.

- В классе определен бин *SecurityFilterChain*, который конфигурирует безопасность вашего приложения.

3. Контроллер *WeatherController*:

- Создан класс *WeatherController*, который обрабатывает HTTP-запросы, связанные с погодой.

- Использует аннотацию @RestController и задает базовый URL пути "/weather".

- В контроллере определен метод *getWeatherData*, который принимает параметр "cityName" из URL-пути и возвращает объект *WeatherData*.

4. Модель *WeatherData*:

- Создан класс *WeatherData*, представляющий структуру данных о погоде, которые будут возвращены клиенту.

- В классе определены переменные для хранения информации о погоде и аннотация `@Getter` для создания геттеров.

- Также определены сеттеры для установки значений полей класса.

5. Сервис *WeatherService*:

- Создан класс *WeatherService*, который выполняет запросы к OpenWeatherMap API и обрабатывает полученные данные.

- В сервисе используется ключ API для доступа к данным OpenWeatherMap.

- Метод *getWeatherData* выполняет HTTP GET-запрос к API, получает ответ в виде объекта *OpenWeatherMapResponse*, и конвертирует его данные в объект *WeatherData*.

# Проектирование структуры базы данных

Мы создали базу данных PostgreSQL для нашего приложения, которая содержит следующие таблицы:

1. *Users*: В этой таблице хранятся данные о пользователях нашего приложения. Каждый пользователь имеет уникальный идентификатор (user\_id), имя пользователя (username), хешированный пароль (password) и ссылку (role\_id) на роль пользователя, которая хранится в таблице *Roles*. Роль пользователя связана с таблицей *Roles* по полю role\_id.

2. *Roles*: В этой таблице хранятся роли пользователей. Каждая роль имеет уникальный идентификатор (role\_id) и имя роли (role\_name). Роли определяют различные уровни доступа и привилегии пользователей в вашем приложении.

3. *Cities*: Эта таблица содержит информацию о городах. Каждый город имеет уникальный идентификатор (city\_id), название города (city\_name) и страну, в которой находится город (country).

4. *Subscriptions*: Эта таблица связывает пользователей с городами, на которые они подписались. Каждая подписка имеет уникальный идентификатор (subscription\_id) и ссылки (user\_id и city\_id) на соответствующие записи в таблицах *Users* и *Cities*. Это позволяет отслеживать, какие пользователи подписаны на какие города.

Эта структура базы данных предоставляет необходимую основу для нашего приложения, позволяя хранить информацию о пользователях, их ролях, городах и подписках. Таблица *Subscriptions* может быть полезной для реализации функциональности отслеживания погоды в выбранных городах пользователями.

Создание схемы *weatherapp* помогает ограничить доступ к таблицам только из этой схемы, обеспечивая лучшую организацию и безопасность базы данных.

# Разработка

Наше iOS приложение представляет собой простое приложение для получения и отображения данных о погоде. Ниже приведено краткое описание ключевых компонентов приложения:

1. *ContentView*: Это основное представление приложения. Он содержит текстовое поле для ввода названия города, кнопку для запроса данных о погоде и отображение результатов запроса. Некоторые ключевые элементы:

- Используется переменная `@State` для отслеживания введенного пользователем названия города и для управления состоянием приложения.

- При вводе города и нажатии кнопки "Поиск", вызывается метод *fetchWeatherData(cityName:)* из объекта *weatherDataVM*, который обрабатывает запрос данных о погоде.

- Отображение состояний приложения: "Загрузка..." при выполнении запроса, сообщение об ошибке, если запрос завершился с ошибкой, и результаты запроса о погоде.

2. *WeatherDataViewModel*: Это класс, представляющий модель данных для приложения. Класс *WeatherDataViewModel* отвечает за обработку запросов к серверу и управление данными о погоде. Ключевые аспекты:

- Использует аннотацию `@Published` для публикации изменений данных, таким образом, обновления могут автоматически отражаться на пользовательском интерфейсе.

- Метод *fetchWeatherData(cityName:)* выполняет запрос данных о погоде на сервер с использованием URLSession, обрабатывает ошибки и декодирует данные с сервера в объект *WeatherData*.

3. *WeatherData*: Это структура данных, которая соответствует структуре данных, возвращаемой сервером. Она реализует протокол Codable, что позволяет декодировать данные JSON из серверного ответа в эту структуру.

4. *WeatherDataFieldView*: Это представление для отображения данных о погоде. Каждый *WeatherDataFieldView* содержит информацию о конкретном аспекте погоды, таком как температура, условия, давление, влажность и т. д. Он содержит значок, заголовок и значение.

Наше iOS приложение позволяет пользователям вводить название города и получать информацию о текущей погоде в этом городе с использованием нашего сервера, который в свою очередь обращается к OpenWeatherMap API. Приложение отображает полученные данные о погоде в удобном и понятном формате, предоставляя пользователю актуальную информацию о погоде.

# Тестирование

# Описание клиентской части

# Демонстрация функционала

