

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационныхтехнологий

**Отчет по практической работе №5**

по дисциплине «Разработка клиент-серверных мобильных приложений»

|  |  |
| --- | --- |
| **Выполнил:**  Студент группы ИКБО-11-22 | Гришин Андрей Валерьевич |
| **Проверила:** | Исабекова Ольга Александровна |

МОСКВА 2025

**Практическая работа 5**

# Задание 1

В предыдущей практической работе мы разработали сервер с возможностью регистрации и авторизации. Но данные, которые вводит пользователь не сохраняются. Соответственно, чтобы решить данную проблему необходимо подключить базу данных, в которой будут сохраняться пользовательские данные. Мы будем работать с PostgreSQL.

Для работы с БД воспользуемся фрэймворком Exposed. Exposed — это библиотека для работы с базами данных в Kotlin, разработанная JetBrains. Она предоставляет два основных подхода: DSL (Domain Specific Language) и DAO (Data Access Objects).

На рисунках 1 и 2 видно подключение необходимых библиотек к проекту.

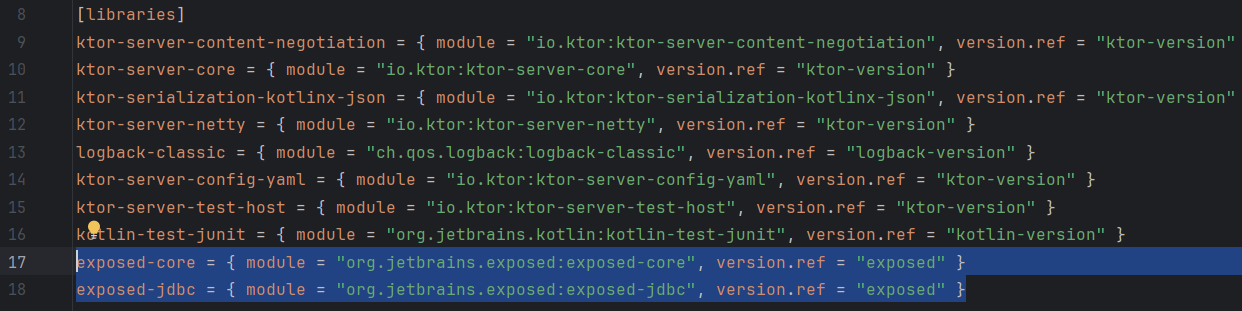


Рисунок 1 - Файл libs.versions.toml

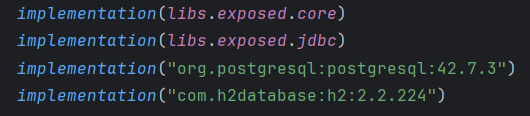


Рисунок 2 - Добавление зависимостей в файле build.gradle.kts

Далее создадим базу данных. PgAdmin4 был скачан и настроен заранее. На рисунке 3 представлена созданная база данных.

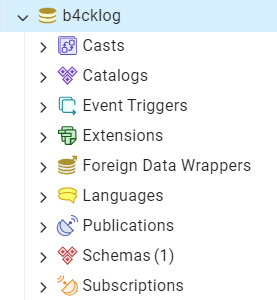


Рисунок 3 – База данных

Создадим 2 таблицы. Первая таблица (users) будет содержать данные о пользователях (рисунок 4). Также заполним информацию о столбцах (рисунок 5).

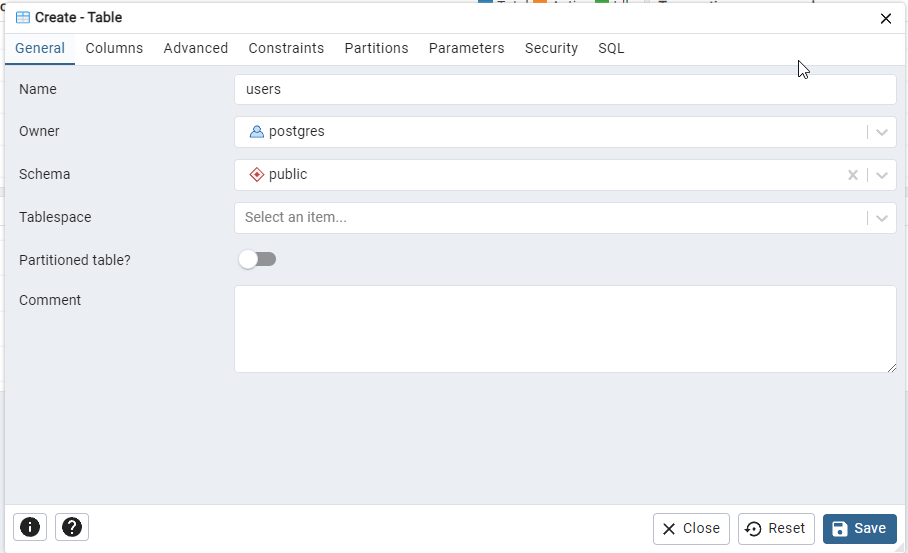


Рисунок 4 - Создание таблицы users

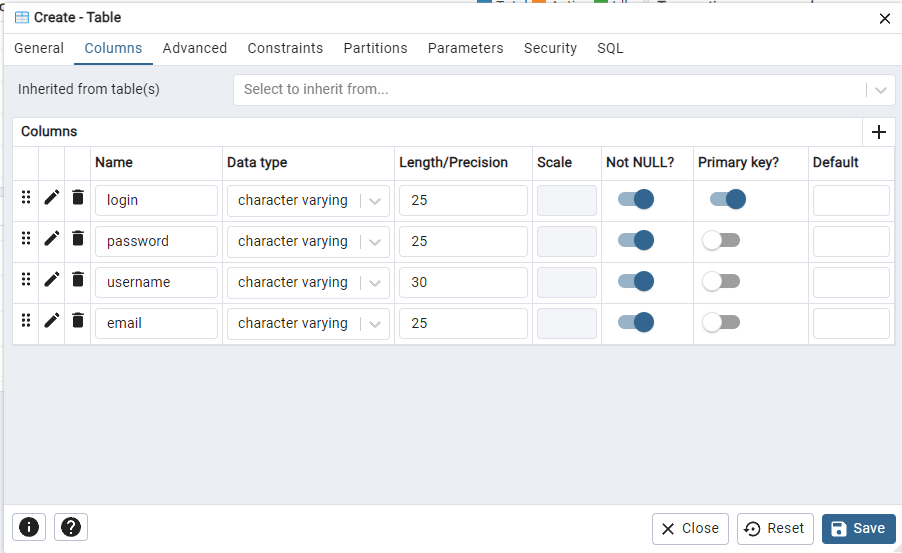


Рисунок 5 - Поля таблицы users

Вторая таблица (tokens) будет хранить логины пользователей и привязанные к ним токены (рисунок 6). Поля таблицы отображены на рисунке 7.

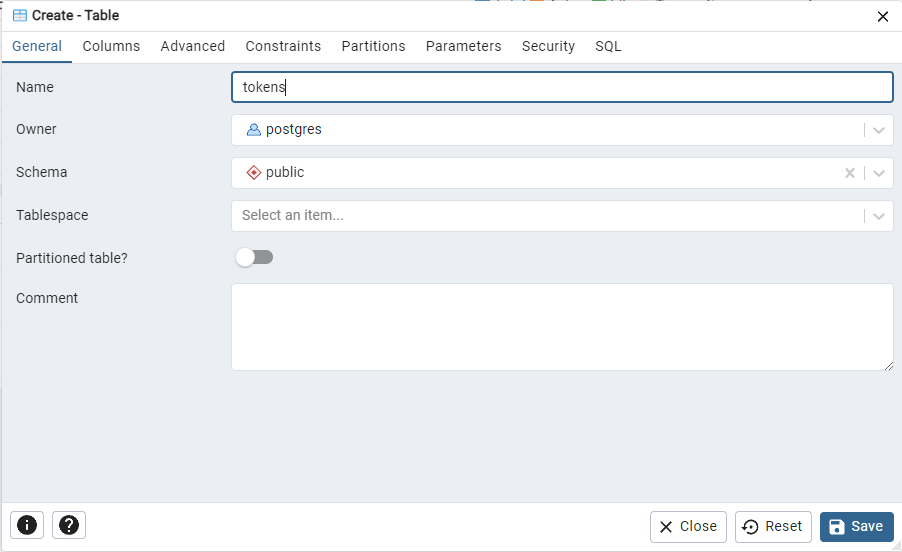


Рисунок 6 - Создание таблицы tokens

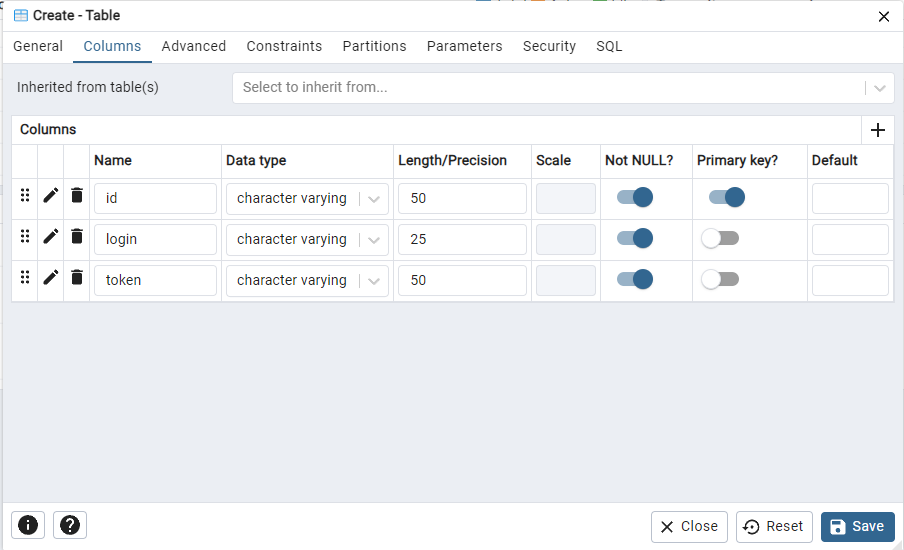


Рисунок 7 - Поля таблицы tokens

После создания БД настроим подключение созданного сервера к ней.

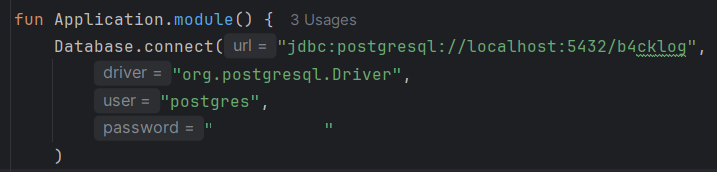
Результат подключения представлен на рисунке 8.

Рисунок 8 – Подключение сервера к базе данных

Далее создадим новый пакет в проекте под названием database, и внутри него еще 2 пакета: tokens и users, в которых будут содержаться файлы, необходимые для работы БД. Создадим классы UserDTO и TokenDTO, в которых пропишем структуру соответствующих таблиц БД (рисунок 9, 10).

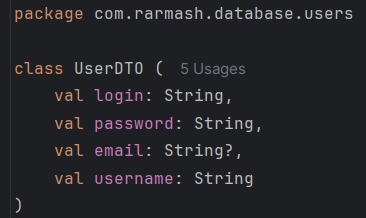


Рисунок 9 - Класс UserDTO

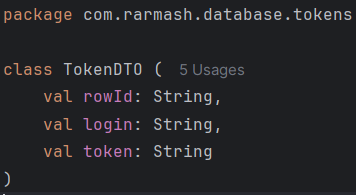


Рисунок 10 - Класс TokenDTO

DTO, или объект передачи данных, используется для переноса данных между слоями приложения, особенно при взаимодействии с API или при обмене данными между клиентом и сервером. DTO обычно не содержат бизнес-логики; они просто представляют собой контейнеры для данных. Основная цель DTO — минимизировать объём данных, которые необходимо передать по сети или между компонентами системы.

Создадим класс Tokens (рисунок 11). В данном классе описаны поля таблицы tokens, созданной БД, а также метод insert, который добавляет новую запись о токене пользователя в таблицу tokens.

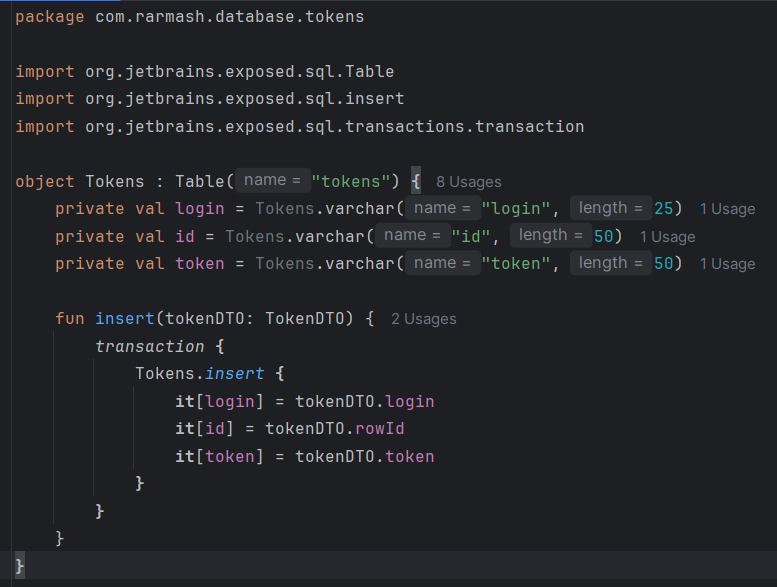


Рисунок 11 – Класс Tokens

Далее создадим класс Users (рисунок 12). В данном классе описаны поля таблицы users, метод insert, функционирующий аналогично методу insert из класса Tokens, а также метод fetchUser, который выполняет поиск пользователя в базе данных по логину и возвращает его данные в виде DTO- объекта.

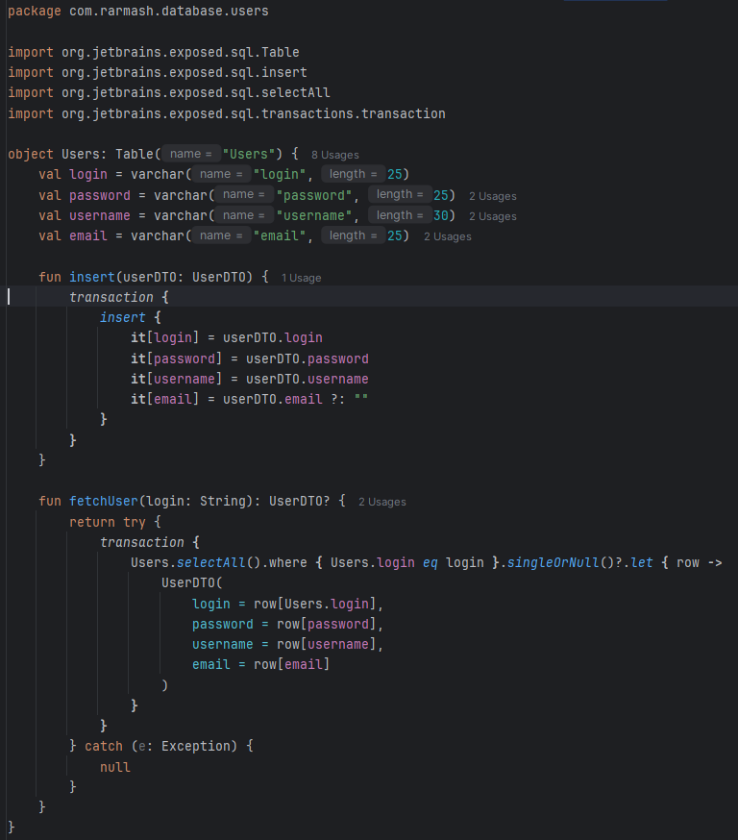


Рисунок 12 - Класс Users

Теперь необходимо изменить реализацию функции авторизации из 4 практической работы, таким образом, чтобы данные пользователей сохранялись не в кэше, а в созданной БД. Для этого класс LoginRemote оставим без изменений и создадим класс LoginController (рисунок 13).

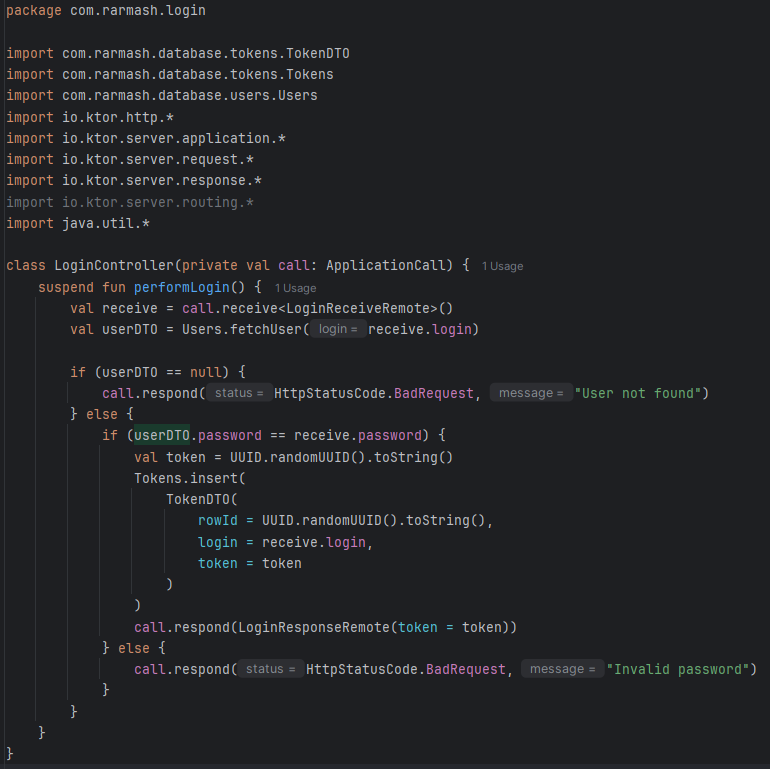


Рисунок 13 - Класс LoginController

В классе LoginRouting создадим экземпляр класса LoginController и вызовем функцию performLogin (рисунок 14).



Рисунок 14 – Класс LoginRouting

Далее изменим реализацию регистрация пользователя. Для этого класс RegisterRemote оставим без изменений и создадим класс RegisterController (рисунок 15).

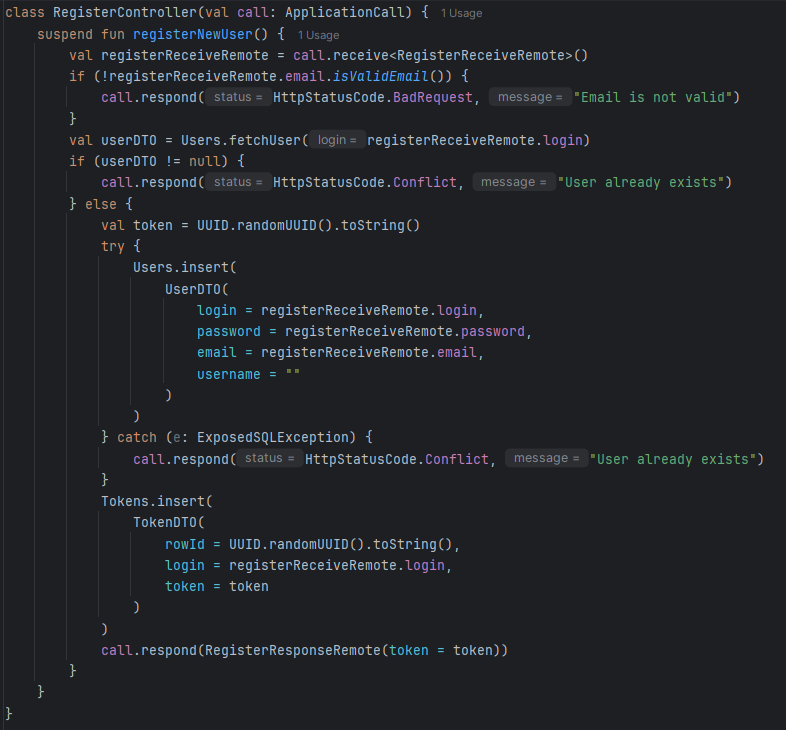


Рисунок 15 – Класс RegisterController

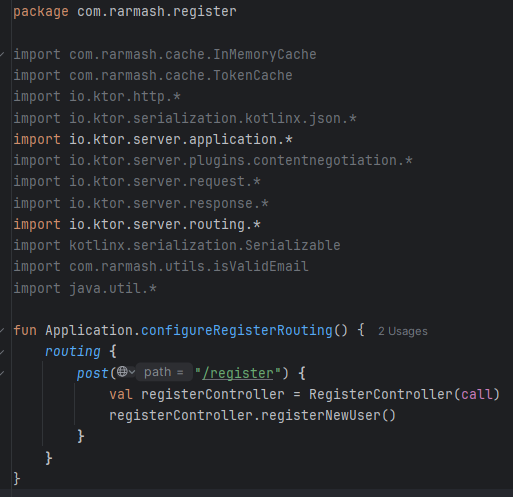
В классе RegisterRouting создадим экземпляр класса RegisterController и вызовем функцию registerNewUser (рисунок 16).

Рисунок 16 – Класс RegisterRouting

Теперь проверим работоспособность сервера с помощью Postman.

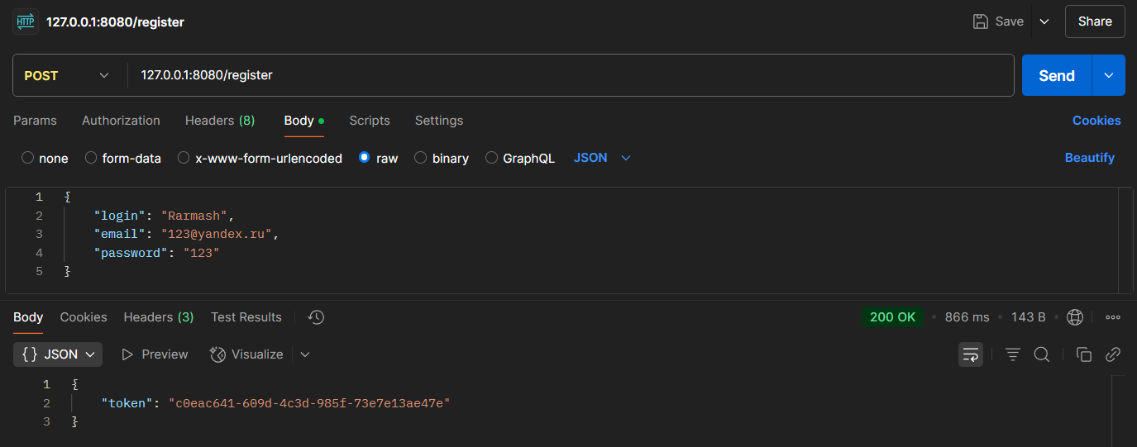
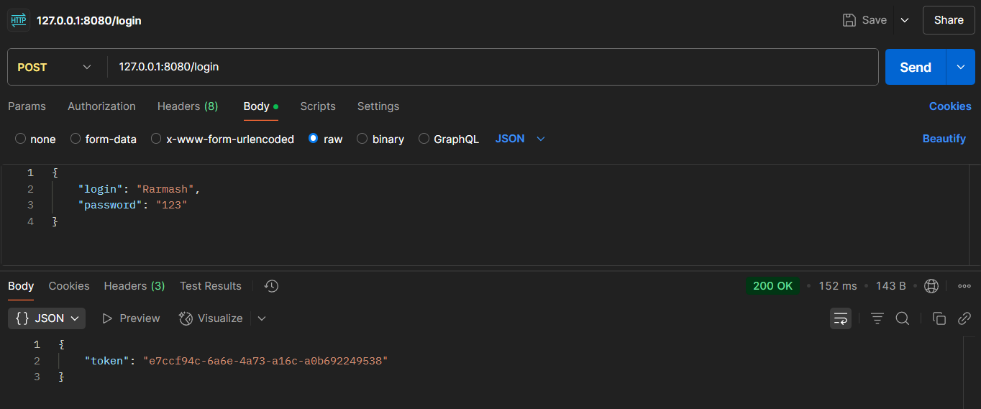
Попробуем зарегистрировать пользователя (рисунок 17).

Рисунок 17 – Отправка запроса на регистрацию пользователя

Так как в ответ был получен токен, значит регистрация прошла успешно.

Попробуем авторизовать зарегистрированного пользователя (рисунок

18).

Рисунок 18 – Отправка запроса на авторизацию пользователя

И снова в ответ был получен токен, значит пользователь успешно авторизован.

Перейдем в pgAdmin 4 и проверим, что записи о пользователе были добавлены в БД. На рисунке 19 отражено содержимое таблицы tokens, на рисунке 20 – users. Записи о пользователе успешно добавились в БД.

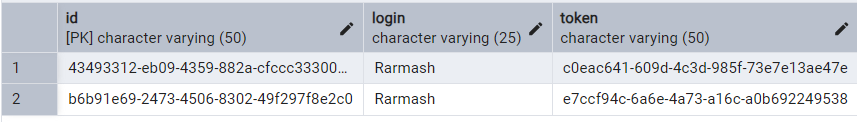


Рисунок 19 – Таблица tokens



Рисунок 20 – Таблица users

Теперь свяжем созданный сервер и БД с клиентом. Для этого создайте простейшее мобильное приложение с двумя экранами – регистрации и авторизации. Создадим приложение с помощью Jetpack Compose.

Для начала добавим е интерфейс AuthApi, в котором укажите, какие HTTP-запросы может выполнять приложение. Результат представлен на листинге 1.

Далее создадим объект ApiClient, в котором необходимо настроить подключение к серверу. Результат создания представлен на листинге 2.

Создадим класс AuthModels. Данный код определяет Data Transfer Objects (DTO) для работы с API аутентификации. На листинге 3 представлен код данного класса.

Создадим функцию для логина loginUser. Авторизация создается запрос через Retrofit. Код представлен на листинге 4, а визуальное представление интерфейса на рисунке 21.

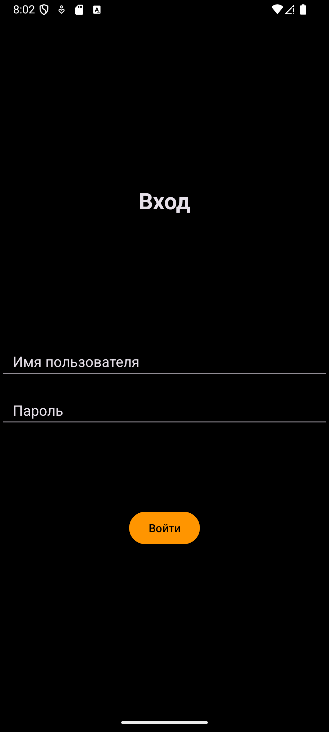


Рисунок 21 – Интерфейс экрана авторизации

Далее создадим функцию для регистрации registerUser. Регистрация создается запрос через Retrofit. Код представлен на листинге 5, а визуальное представление интерфейса на рисунке 22.

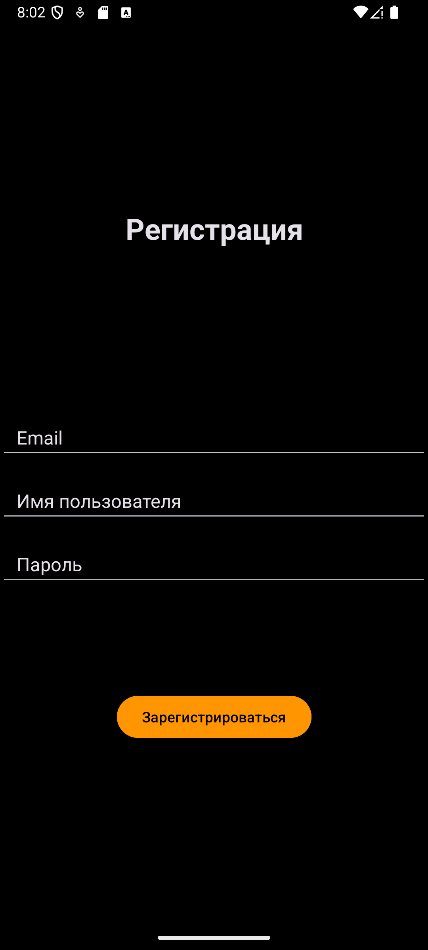


Рисунок 22 – Интерфейс экрана регистрации

Далее проверим работоспособность взаимодействия клиента и сервера. Запустим сервер и клиент, затем в окне регистрации введем данные пользователя и увидим результат (рисунок 23).

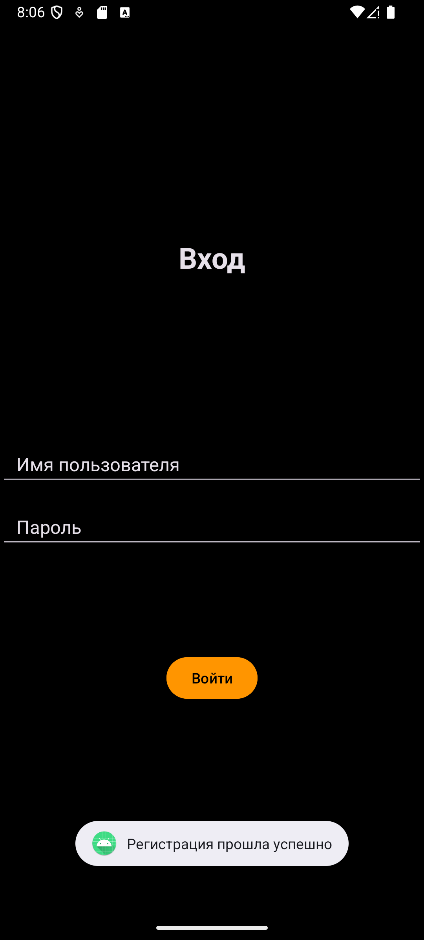


Рисунок 23 – Получение уведомления об успешной регистрации и переход на новый экран

Видно уведомление, так что регистрация успешна. Проверим через

PgAdmin. На рисунке 24 представлена таблица users, а на рисунке 25 таблица tokens.



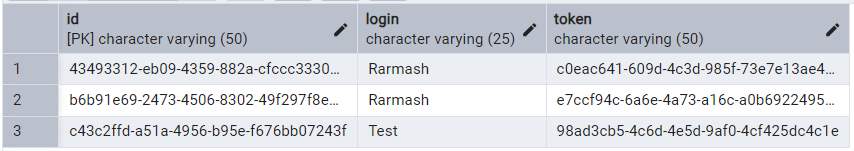
Рисунок 24 – Данные, сохраненные в таблице users

Рисунок 25 - Данные, сохраненные в таблице tokens Как видно, данные успешно сохранились в таблицу.

Введем на экране авторизации данные зарегистрированного пользователя и нажмем кнопку «Войти» (рисунок 26). Если все выполнено верно на экране отобразится сообщение об удачном входе и будет произведён переход на новый экран.

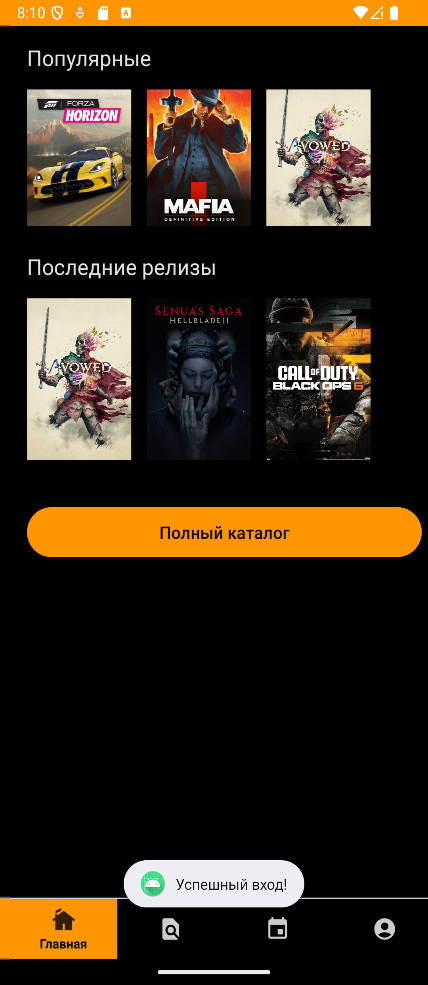


Рисунок 26 – Получение уведомления об успешной авторизации

Так как все выполнено верно на экране отобразится сообщение об удачном входе.

# ВЫВОД

В ходе практической работы была успешно реализована серверная часть приложения с использованием фреймворка Ktor на языке Kotlin, демонстрирующая создание простого HTTP-сервера с функционалом регистрации и авторизации пользователей и сохранением их данных в базу данных postgres. Так же было создано простое мобильное приложение для проверки взаимодействия сервера с клиентом. При помощи проверки было выявлено, что сервер и клиент работают корректно.

# СПИСОК ЛИТЕРАТУРЫ

1. Сомон., П. И. Волшебство Kotlin: руководство / П. И. Сомон.; перевод с английского А. Н. Киселева.. — Москва: ДМК Пресс, 2020. — 536 с.

— ISBN 978-5-97060-801-2. — Текст: электронный // Лань: электронно- библиотечная система. — URL: https://e.lanbook.com/book/140599 (дата обращения: 15.02.2024). — Режим доступа: для авториз. пользователей.

1. Калгина, И. С. Разработка мобильных приложений: учебное пособие / И. С. Калгина. — Чита: ЗабГУ, 2022. — 163 с. — ISBN 978-5-9293- 3137- 4. — Текст: электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/363323 (дата обращения: 15.02.2024). — Режим доступа: для авториз. пользователей.
2. Алпатов, А. Н. Архитектура, проектирование и разработка программных средств: учебное пособие / А. Н. Алпатов, И. Е. Рогов. — Москва: РТУ МИРЭА, 2023. — 120 с. — ISBN 978-5-7339-1972-0. — Текст:

электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/386189 (дата обращения: 15.02.2024). — Режим доступа: для авториз. пользователей.

# ПРИЛОЖЕНИЕ

На листинге 1 представлен интерфейс AuthApi, в котором указано, какие HTTP-запросы может выполнять приложение.

*Листинг 1 – Код AuthApi*

|  |
| --- |
| package com.rarmash.b4cklog.network  import retrofit2.http.Body import retrofit2.http.POST import retrofit2.Call   interface AuthApi {  @POST("login")  fun login(@Body request: LoginRequest): Call<AuthResponse>   @POST("register")  fun register(@Body request: RegisterRequest): Call<AuthResponse> } |

На листинге 2 представлен объект ApiClient, в котором настроено подключение к серверу.

*Листинг 2 – Код ApiClient*

|  |
| --- |
| package com.rarmash.b4cklog.network  import retrofit2.Retrofit import retrofit2.converter.gson.GsonConverterFactory  object ApiClient {  private const val BASE\_URL = "http://10.0.2.2:8080/"   private val retrofit = Retrofit.Builder()  .baseUrl(BASE\_URL)  .addConverterFactory(GsonConverterFactory.create())  .build()   val authApi: AuthApi = retrofit.create(AuthApi::class.*java*) } |

На листинге 3 представлен класс AuthModels, который определяет Data Transfer Objects (DTO) для работы с API аутентификации.

*Листинг 3 – Код AuthModels*

|  |
| --- |
| package com.rarmash.b4cklog.network  data class LoginRequest (  val login: String,  val password: String )  data class RegisterRequest (  val login: String,  val password: String,  val email: String )  data class AuthResponse (  val token: String ) |

На листинге 4 представлена функция loginUser, которая отвечает за авторизацию пользователя.

*Листинг 4 – Код loginUser*

|  |
| --- |
| private fun loginUser(login: String, password: String) {  val call = ApiClient.authApi.login(LoginRequest(login, password))  call.enqueue(object: Callback<AuthResponse> {  override fun onResponse(call: Call<AuthResponse>, response: Response<AuthResponse>) {  if (response.*isSuccessful*) {  Toast.makeText(this@LoginActivity, "Успешный вход!", Toast.*LENGTH\_SHORT*).show()  val intent = Intent(this@LoginActivity, MainActivity::class.*java*)  startActivity(intent)  } else {  Toast.makeText(this@LoginActivity, "Ошибка входа", Toast.*LENGTH\_SHORT*).show()  }  }   override fun onFailure(call: Call<AuthResponse>, t: Throwable) {  Toast.makeText(this@LoginActivity, "Ошибка сети: ${t.message}", Toast.*LENGTH\_SHORT*).show()  }  }) } |

На листинге 5 представлена функция registerUser, которая отвечает за экран регистрации.

*Листинг 5 – Код registerUser*

|  |
| --- |
| private fun registerUser(login: String, password: String, email: String) {  val call = ApiClient.authApi.register(RegisterRequest(login, password, email))  call.enqueue(object: Callback<AuthResponse> {  override fun onResponse(call: Call<AuthResponse>, response: Response<AuthResponse>) {  if (response.*isSuccessful*) {  val intent = Intent(this@SignUpActivity, LoginActivity::class.*java*)  Toast.makeText(this@SignUpActivity, "Регистрация прошла успешно", Toast.*LENGTH\_SHORT*).show()  startActivity(intent)  } else {  Toast.makeText(this@SignUpActivity, "Ошибка регистрации", Toast.*LENGTH\_SHORT*).show()  }  }   override fun onFailure(call: Call<AuthResponse>, t: Throwable) {  Toast.makeText(this@SignUpActivity, "Ошибка сети: ${t.message}", Toast.*LENGTH\_SHORT*).show()  Log.d("SignUpActivity", "ERROR: ${t.message.*toString*()}")  }  }) } |