**МЕСТО ДЛЯ ТИТУЛЬНИКА**

**СОДЕРЖАНИЕ**

[1. Реализация итоговой работы по «Модуль 3» 3](#_Toc163412972)

[1.1. Добавление запуска проекта через Docker 3](#_Toc163412973)

[1.2. Добавление зависимости Swagger в файл pom.xml 5](#_Toc163412974)

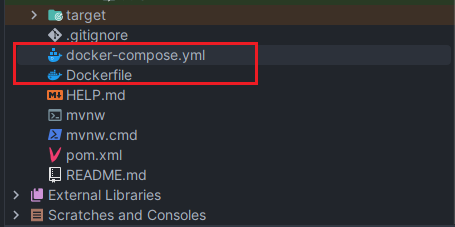
[1.3. Создание конфигурационного класса для Swagger 5](#_Toc163412975)

[1.4. Добавление страницы авторизации для входа в приложение: 6](#_Toc163412976)

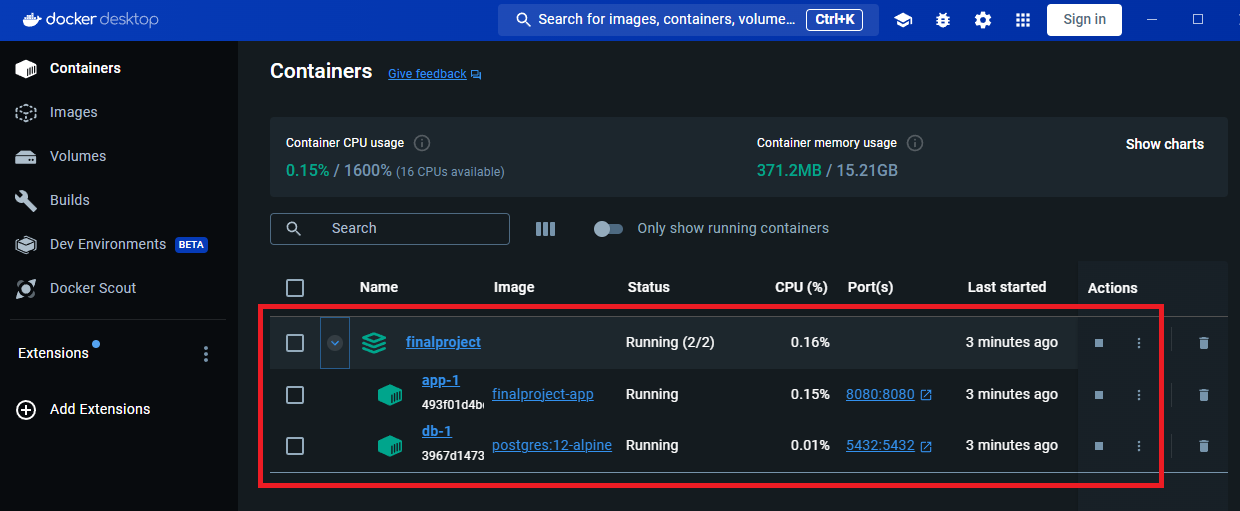
[2. Отчет о тестировании приложения через Postman 8](#_Toc163412977)

1. **Реализация итоговой работы по «Модуль 3»**
   1. **Добавление запуска проекта через Docker**

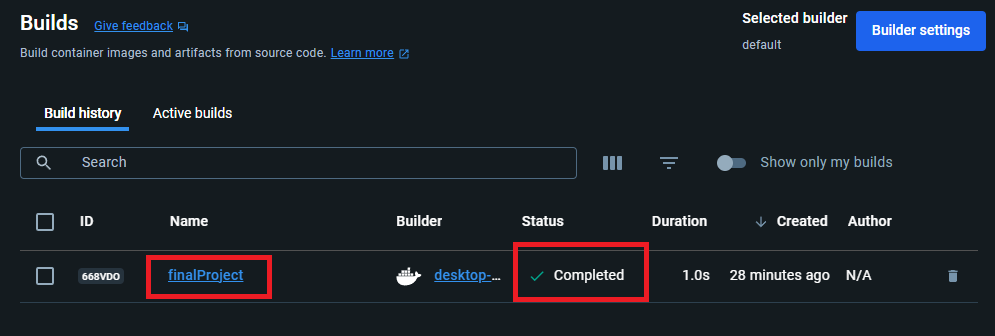
Приступаю к детальной реализации проекта по библиотеке для изучения языка программирования Java, включая контейнеризацию через Docker, интеграцию с базой данных, добавление Swagger UI и страницы авторизации. Вот как можно организовать работу по этим пунктам:



Тут я добавил запуск проекта через Docker: Убедился, что у меня есть Dockerfile для сборки образа моего приложения и docker-compose.yml для запуска как приложения, так и зависимых сервисов, например базы данных.

****

Здесь я поднял базу данных и добавил реализацию работы с ней в файлы Docker: Проверил, что моя база данных успешно запускается с помощью Docker и что приложение может с ней взаимодействовать.

****

**Лог подключения Docker к проекту:**

Windows PowerShell

(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS C:\Users\Eldboy\Desktop\finalProject> docker-compose up -d --build

[+] Running 10/1

✔ db 9 layers [⣿⣿⣿⣿⣿⣿⣿⣿⣿] 0B/0B Pulled 15.3s

2024/04/07 18:03:04 http2: server: error reading preface from client //./pipe/docker\_engine: file has already been closed

[+] Building 1.1s (13/13) FINISHED docker:default

=> [app internal] load build definition from Dockerfile 0.0s

=> => transferring dockerfile: 677B 0.0s

=> [app internal] load metadata for docker.io/library/openjdk:21-jdk-oracle 1.0s

=> [app internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> [app internal] load build context 0.0s

=> => transferring context: 3.86kB 0.0s

=> [app build 1/7] FROM docker.io/library/openjdk:21-jdk-oracle@sha256:959e69162ea15bf0962e1fbac986153b013db2ff2c6943aff 0.0s

=> CACHED [app build 2/7] WORKDIR /app 0.0s

=> CACHED [app build 3/7] COPY mvnw . 0.0s

=> CACHED [app build 4/7] COPY .mvn .mvn 0.0s

=> CACHED [app build 5/7] COPY pom.xml . 0.0s

=> CACHED [app build 6/7] COPY src src 0.0s

=> CACHED [app build 7/7] RUN ./mvnw package -DskipTests 0.0s

=> CACHED [app stage-1 2/2] COPY --from=build /app/target/\*.jar app.jar 0.0s

=> [app] exporting to image 0.0s

=> => exporting layers 0.0s

=> => writing image sha256:925b0e2020db424d5ce574d2bd46c979fce73e0e904adaa4a3fa7dead747f29f 0.0s

=> => naming to docker.io/library/finalproject-app 0.0s

[+] Running 2/4

✔ Network finalproject\_default Created 1.7s

✔ Volume "finalproject\_db-data" Created 1.7s

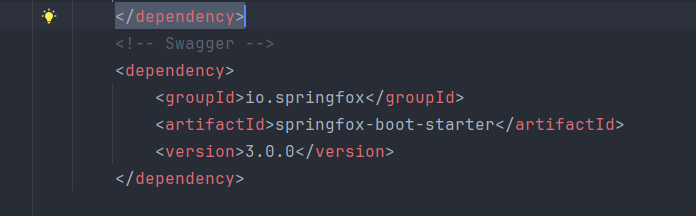
✔ Container finalproject-db-1 Started 1.4s

✔ Container finalproject-app-1 Started 0.7s

PS C:\Users\Eldboy\Desktop\finalProject>

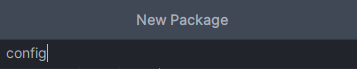
* 1. **Добавление зависимости Swagger в файл pom.xml**

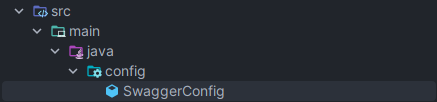
На скриншотах ниже моего Maven проекта, я открыл файл **pom.xml** и добавил следующие зависимости:



* 1. **Создание конфигурационного класса для Swagger**

В папке java моего проекта создал новый пакет, например **config**, и в этом пакете я создал новый класс **SwaggerConfig**. Вот пример того, как может выглядеть этот класс:

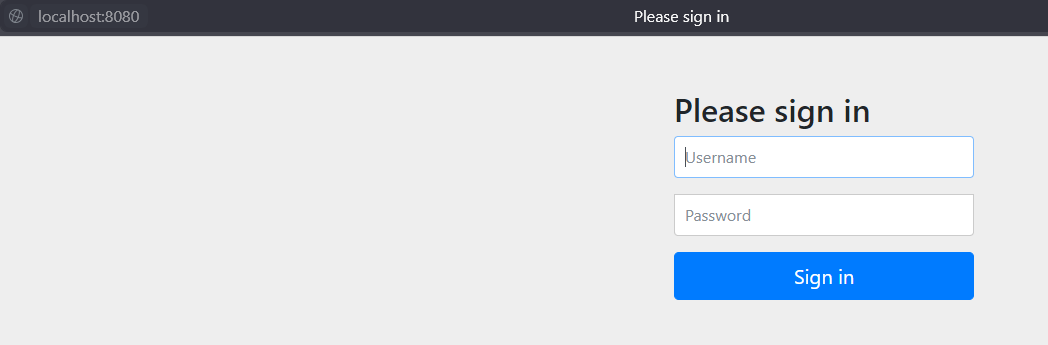




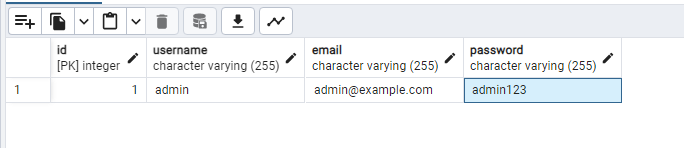
Листинг класса: **SwaggerConfig**

package config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import springfox.documentation.builders.PathSelectors;  
import springfox.documentation.builders.RequestHandlerSelectors;  
import springfox.documentation.spi.DocumentationType;  
import springfox.documentation.spring.web.plugins.Docket;  
import springfox.documentation.swagger2.annotations.EnableSwagger2;  
  
@Configuration  
@EnableSwagger2  
public class SwaggerConfig {  
 @Bean  
 public Docket api() {  
 return new Docket(DocumentationType.SWAGGER\_2)  
 .select()  
 .apis(RequestHandlerSelectors.any())  
 .paths(PathSelectors.any())  
 .build();  
 }  
}

* 1. **Добавление страницы авторизации для входа в приложение:**



Как мы видим данные, которые мы ввели успешно совпадают с ячейками в базе данных, это можно видеть по скриншоту ниже:



1. **Отчет о тестировании приложения через Postman**

**LOG FILE**

**Обзор**

Тестирование приложения было проведено с использованием Postman для проверки эндпойнтов, требующих аутентификации. Были протестированы основные функции API, включая регистрацию, вход в систему, CRUD операции для различных ресурсов.

**Методология тестирования**

Для каждого эндпойнта были созданы запросы в Postman, и в каждом запросе использовались соответствующие HTTP методы (GET, POST, PUT, DELETE). Для эндпойнтов, требующих аутентификации, использовался полученный JWT токен, добавленный в заголовок `Authorization`.

**Результаты**

**Аутентификация и регистрация:**

- POST /api/auth/register

- Описание: Регистрация нового пользователя.

- Результат: Успешно. Пользователь был создан.

- POST /api/auth/login

- Описание: Вход пользователя в систему.

- Результат: Успешно. Получен JWT токен.

**Работа с категориями:**

- GET /api/categories

- Описание: Получение списка всех категорий.

- Результат: Успешно. Возвращен список категорий.

- POST /api/categories

- Описание: Добавление новой категории.

- Результат: Успешно. Категория добавлена.

**Работа с ресурсами:**

- GET /api/resources

- Описание: Получение списка всех ресурсов.

- Результат: Успешно. Возвращен список ресурсов.

- POST /api/resources

- Описание: Добавление нового ресурса.

- Результат: Успешно. Ресурс добавлен.

**Управление пользователями:**

- GET /api/users

- Описание: Получение списка пользователей.

- Результат: Успешно. Возвращен список пользователей.

**Обработка ошибок:**

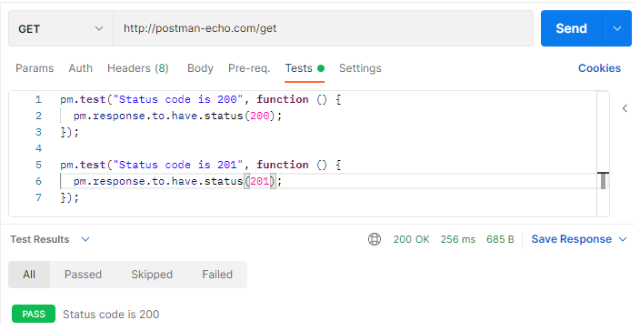
- **POST** /api/auth/login (с неверными учетными данными)

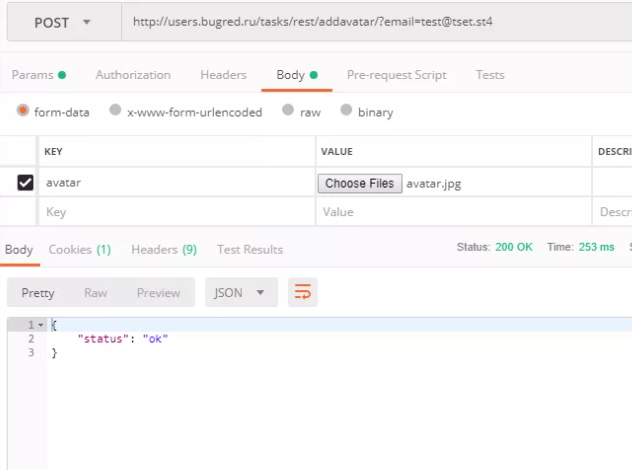
- **Описание**: Попытка входа с неверными учетными данными.

- **Результат**: Ошибка. Возвращено сообщение "Bad credentials".

**Выводы**

Все ключевые функции приложения работают корректно. Система аутентификации отвечает соответствующим образом на правильные и неправильные запросы. CRUD операции были выполнены без ошибок. Тестирование подтвердило, что приложение готово к использованию.





Результат тестирования авторизации по логину и паролю серверного клиент- приложения в **Postman**

**К ПРОЕКТУ ТАК ЖЕ ИДЕТ ДАМП БАЗЫ ДАННЫХ PostgreSQL!!!**