**Министерство науки и высшего образования РФ**

**Государственное образовательное учреждение высшего профессионального образования**

**Белгородский Государственный Технологический Университет им В. Г. Шухова**

Кафедра программного обеспечения вычислительной техники и

автоматизированных систем.

Лабораторная работа №3 по дисциплине

«Технологии Web-программирования»

Тема: Серверное программирование.

Выполнил:

студент группы ПВ-41

Шапаренко А.А.

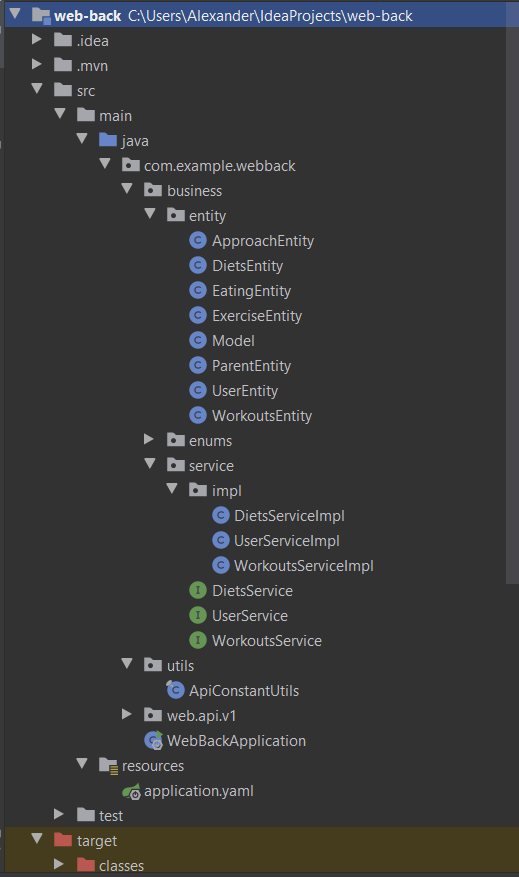
Проверил: Картамышев С.В.

Белгород 2020 г.

**Цель работы:** познакомиться с основами backend разработки web-приложений на языке Java. Познакомиться с основами работы docker. Познакомиться с фреймворком Spring-Boot 2и научиться разворачивать проект, производить его настройку. Научится работать с API в приложении Postman.

**Выполение**

Стандартная структура spring приложения с системой сборки maven



**pom.xml** – файл в котором будут находится все необходимые нам зависимост

**Dockerfile** – файл, в котом будет находится конфигурация нашего Docker-образа

**README.md** – файл, в котором будет находиться инструкция по использованию.

**App** – класс, точка запуска приложения (настройка конфигурации и многое другое)

**Controller** – класс, контроллер. Реализация endpoints

Далее был настроен файл конфигурации Dockerfile

FROM openjdk:11  
ADD target/web-back-0.0.1-SNAPSHOT.jar web-back-0.0.1-SNAPSHOT.jar  
EXPOSE 8082  
ENTRYPOINT ["java", "-jar", "web-back-0.0.1-SNAPSHOT.jar"]

Все необходимы зависимости были подключены в pom.xml

*<?***xml version**="1.0" **encoding**="UTF-8"*?>*<project **xmlns**="http://maven.apache.org/POM/4.0.0" **xmlns:xsi**="http://www.w3.org/2001/XMLSchema-instance"  
 **xsi:schemaLocation**="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.3.7.BUILD-SNAPSHOT</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.example</groupId>  
 <artifactId>web-back</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>web-back</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>11</java.version>  
 <org.mapstruct.version>1.3.0.Final</org.mapstruct.version>  
 <org.projectlombok.version>1.18.10</org.projectlombok.version>  
 <org.apache.maven.plugins.version>3.6.0</org.apache.maven.plugins.version>  
 <io.springfox.springfox-swagger2.version>2.9.2</io.springfox.springfox-swagger2.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-webflux</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.mapstruct</groupId>  
 <artifactId>mapstruct</artifactId>  
 <version>${org.mapstruct.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <scope>runtime</scope>  
 <optional>true</optional>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 <exclusions>  
 <exclusion>  
 <groupId>org.junit.vintage</groupId>  
 <artifactId>junit-vintage-engine</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>${org.apache.maven.plugins.version}</version>  
 <configuration>  
 <source>${java.version}</source>  
 <target>${java.version}</target>  
 <annotationProcessorPaths>  
 <path>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>${org.projectlombok.version}</version>  
 </path>  
 <path>  
 <groupId>org.mapstruct</groupId>  
 <artifactId>mapstruct-processor</artifactId>  
 <version>${org.mapstruct.version}</version>  
 </path>  
 </annotationProcessorPaths>  
 <compilerArgs>  
 <compilerArg>  
 -Amapstruct.defaultComponentModel=spring  
 </compilerArg>  
 </compilerArgs>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
 <repositories>  
 <repository>  
 <id>spring-milestones</id>  
 <name>Spring Milestones</name>  
 <url>https://repo.spring.io/milestone</url>  
 </repository>  
 <repository>  
 <id>spring-snapshots</id>  
 <name>Spring Snapshots</name>  
 <url>https://repo.spring.io/snapshot</url>  
 <snapshots>  
 <enabled>true</enabled>  
 </snapshots>  
 </repository>  
 </repositories>  
 <pluginRepositories>  
 <pluginRepository>  
 <id>spring-milestones</id>  
 <name>Spring Milestones</name>  
 <url>https://repo.spring.io/milestone</url>  
 </pluginRepository>  
 <pluginRepository>  
 <id>spring-snapshots</id>  
 <name>Spring Snapshots</name>  
 <url>https://repo.spring.io/snapshot</url>  
 <snapshots>  
 <enabled>true</enabled>  
 </snapshots>  
 </pluginRepository>  
 </pluginRepositories>  
  
</project>

В данной зависимости находятся все необходимые библиотеки для работы контроллеров, а так же Tomcat(HTPP сервер работающий по принципу создания пула потоков для принятия запросов).

Для того чтобы запустить проект необходимо создать точку входа

package com.example.webback;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class WebBackApplication {  
  
 public static void main(String[] args) {  
  
 SpringApplication.*run*(WebBackApplication.class, args);  
 }  
  
}

Так выглядит базовый класс запуска приложения. Аннотация @SpringBootApplication которая автоматически конфигурирует Tomcat, внедряет зависимости и т.д.

Был добавлен API модуль версии 1 со следующими контроллерами:

**DietsController**

@RestController  
@RequestMapping(***DIETS***)  
@AllArgsConstructor  
public class DietsController {  
  
 private final DietsService **dietsService**;  
  
 @GetMapping("/{id}")  
 public DietsEntity findById(@PathVariable Integer id){  
 return **dietsService**.getById(id);  
 }  
  
 @GetMapping(***ALL***)  
 public List<DietsEntity> findAll(){  
 return **dietsService**.getAll();  
 }  
}

**UserService**

@RestController  
@RequestMapping(***USER***)  
@AllArgsConstructor  
public class UserController {  
  
 private final UserService **userService**;  
  
 @GetMapping("/{id}")  
 public UserEntity findById(@PathVariable Integer id){  
  
 return **userService**.getById(id);  
 }  
}

**WorkoutsController**

@RestController  
@RequestMapping(***WORKOUTS***)  
@AllArgsConstructor  
public class WorkoutsController {  
  
 private final WorkoutsService **workoutsService**;  
  
  
 @GetMapping("/{id}")  
 public WorkoutsEntity findById(@PathVariable Integer id){  
  
 return **workoutsService**.getById(id);  
 }  
  
 @GetMapping(***ALL***)  
 public List<WorkoutsEntity> findAll(){  
 return **workoutsService**.getAll();  
 }

Аннотация @RestContoller – указывает что данный класс является частью sprint-context и указывает что данный класс будет принимать http запросы и по умолчанию возвращать данные в формате Json.

@RequestMappint – ulr контроллера

@GetMappint – указывает какой тип запроса принимает данный метод. В нашем случае это Get запрос. Так же он содержит url, к которому идет обращение. (/user/1)

Модели представлены в виде классов:

**WorkoutsEntity**

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class WorkoutsEntity extends Model<Integer> {  
  
 private String **info**;  
  
 private WorkoutsTypeEnum **type**;  
  
 private ComplexityEnum **complexity**;  
  
 private String **previewUri**;  
  
 private List<ExerciseEntity> **exercise**;  
  
}

**UserEntity**

@Data  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class UserEntity {  
  
 private Integer **id**;  
  
 private String **firstName**;  
  
 private String **secondName**;  
  
 private String **photoUri**;  
  
 private Integer **growth**;  
  
 private Double **weight**;  
  
}

**DietsEntity**

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class DietsEntity extends Model<Integer>{  
  
 private String **info**;  
 private DietsTypeEnum **type**;  
 private String **previewUri**;  
 private List<EatingEntity> **eating**;  
  
}

Так же объявлены сервисы(в них описана бизнес логика).

**DietsService**

@Service  
public class DietsServiceImpl implements DietsService {  
  
 private EatingEntity initEatingEntity(Integer id, String info, String name){  
 EatingEntity eatingEntity = new EatingEntity();  
 eatingEntity.setId(id);  
 eatingEntity.setInfo(info);  
 eatingEntity.setName(name);  
 return eatingEntity;  
 }  
  
 private DietsEntity init(Integer id, DietsTypeEnum type, String uri, String info, String name, List<EatingEntity> eating){  
 DietsEntity dietsEntity = new DietsEntity();  
 dietsEntity.setName(name);  
 dietsEntity.setInfo(info);  
 dietsEntity.setPreviewUri(uri);  
 dietsEntity.setType(type);  
 dietsEntity.setId(id);  
 dietsEntity.setEating(eating);  
 return dietsEntity;  
 }  
  
 private List<DietsEntity> initList(){  
 List<EatingEntity> eatingEntities = new ArrayList<>();  
 eatingEntities.add(initEatingEntity(1,"TEST", "NAME"));  
 eatingEntities.add(initEatingEntity(2,"TEST", "NAME"));  
 eatingEntities.add(initEatingEntity(3,"TEST", "NAME"));  
  
 List<DietsEntity> dietsEntities = new ArrayList<>();  
 dietsEntities.add(init(1, DietsTypeEnum.***SLIMMING***, "uri", "info", "name", eatingEntities));  
 dietsEntities.add(init(2, DietsTypeEnum.***WEIGHT\_GAIN***, "uri2", "info2", "name2", eatingEntities));  
  
 return dietsEntities;  
 }  
  
  
 @Override  
 public DietsEntity getById(Integer id) {  
 return initList().stream().filter(i -> i.getId().equals(id)).findFirst().get();  
 }  
  
 @Override  
 public List<DietsEntity> getAll() {  
 return initList();  
 }  
}

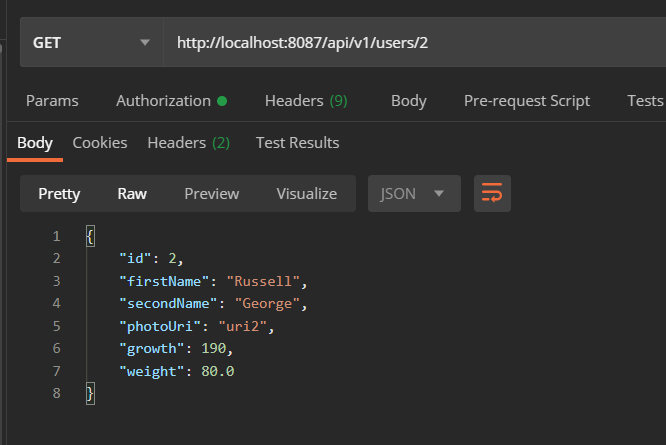
**UserService**

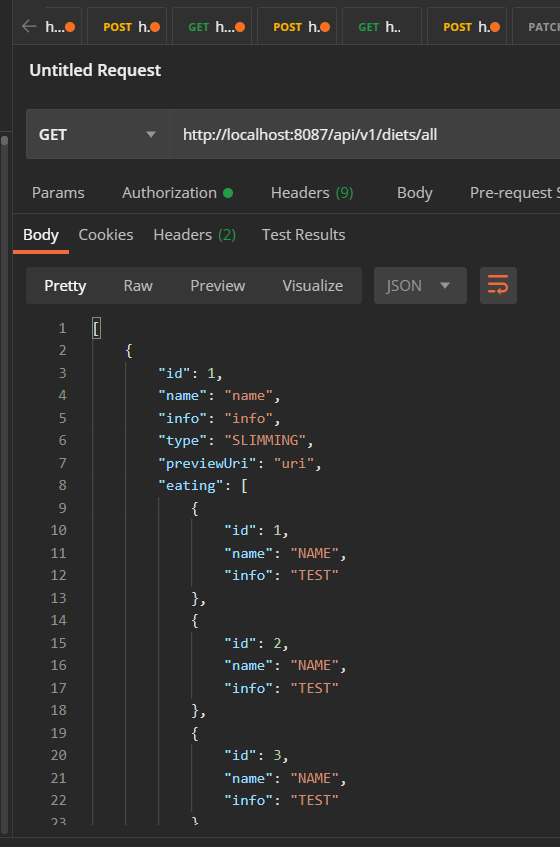
@Service  
public class UserServiceImpl implements UserService {  
  
  
 private List<UserEntity> init(){  
 List<UserEntity> userEntities = new ArrayList<>();  
 userEntities.add(new UserEntity(1, "Андреев", "Олег", "uri1", 170, 70.0 ));  
 userEntities.add(new UserEntity(2, "Russell", "George", "uri2", 190, 80.0 ));  
 userEntities.add(new UserEntity(3, "Немыкина", "Алина", "uri2", 165, 50.0 ));  
 return userEntities;  
 }  
  
 @Override  
 public UserEntity getById(Integer id) {  
 return init().stream().filter(i -> i.getId().equals(id)).findFirst().get();  
 }  
}

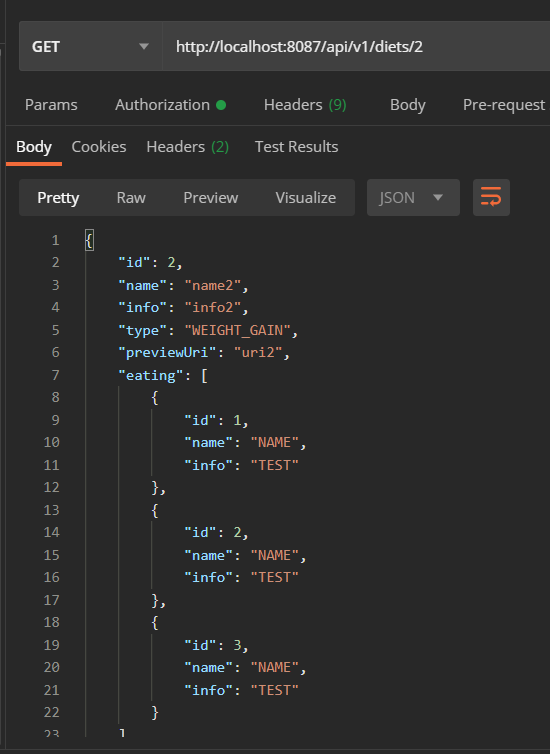
**WorkoutsService**

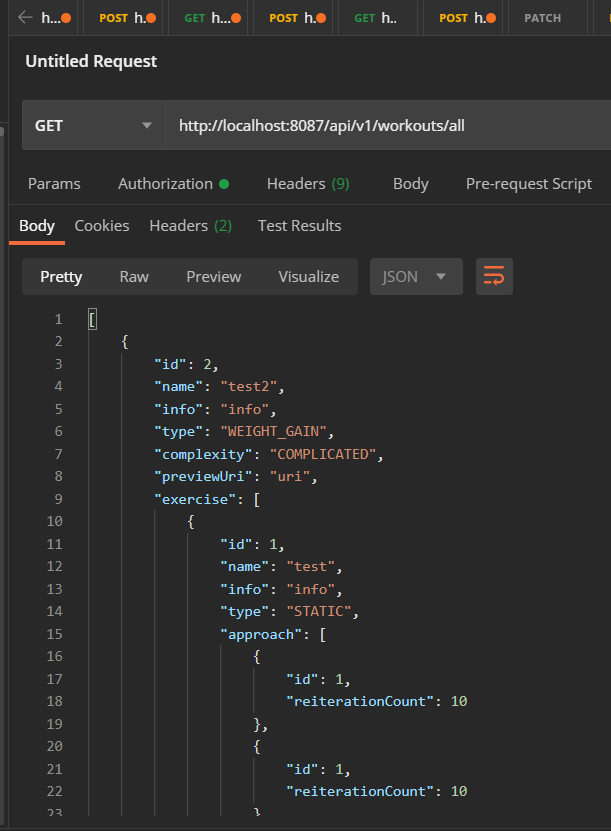
@Service  
public class WorkoutsServiceImpl implements WorkoutsService {  
  
 private List<ApproachEntity> initApproachEntitys(){  
 ApproachEntity approachEntity = new ApproachEntity();  
 approachEntity.setId(1);  
 approachEntity.setReiterationCount(10);  
 List<ApproachEntity> approachEntities = new ArrayList<>();  
 approachEntities.add(approachEntity);  
 approachEntities.add(approachEntity);  
 approachEntities.add(approachEntity);  
 return approachEntities;  
 }  
  
 private List<ExerciseEntity> initExerciseEntity(){  
 ExerciseEntity exerciseEntity = new ExerciseEntity();  
 exerciseEntity.setId(1);  
 exerciseEntity.setName("test");  
 exerciseEntity.setInfo("info");  
 exerciseEntity.setType(ExerciseTypeEnum.***STATIC***);  
 exerciseEntity.setApproach(initApproachEntitys());  
  
 List<ExerciseEntity> exerciseEntities = new ArrayList<>();  
 exerciseEntities.add(exerciseEntity);  
 exerciseEntities.add(exerciseEntity);  
 exerciseEntities.add(exerciseEntity);  
 return exerciseEntities;  
 }  
  
 private List<WorkoutsEntity> init(){  
 WorkoutsEntity workoutsEntity = new WorkoutsEntity();  
 workoutsEntity.setId(1);  
 workoutsEntity.setName("test");  
 workoutsEntity.setComplexity(ComplexityEnum.***AVERAGE***);  
 workoutsEntity.setInfo("info");  
 workoutsEntity.setPreviewUri("uri");  
 workoutsEntity.setExercise(initExerciseEntity());  
 workoutsEntity.setType(WorkoutsTypeEnum.***WEIGHT\_GAIN***);  
  
 WorkoutsEntity workoutsEntity2 = new WorkoutsEntity();  
 workoutsEntity.setId(2);  
 workoutsEntity.setName("test2");  
 workoutsEntity.setComplexity(ComplexityEnum.***COMPLICATED***);  
 workoutsEntity.setInfo("info");  
 workoutsEntity.setPreviewUri("uri");  
 workoutsEntity.setExercise(initExerciseEntity());  
 workoutsEntity.setType(WorkoutsTypeEnum.***WEIGHT\_GAIN***);  
  
 List<WorkoutsEntity> workoutsEntities = new ArrayList<>();  
 workoutsEntities.add(workoutsEntity);  
 workoutsEntities.add(workoutsEntity2);  
 return workoutsEntities;  
  
  
 }  
  
  
 @Override public WorkoutsEntity getById(Integer id) {  
 return init().stream().filter(i -> i.getId().equals(id)).findFirst().get();  
 }  
  
 @Override public List<WorkoutsEntity> getAll() {  
 return init();  
 }  
}

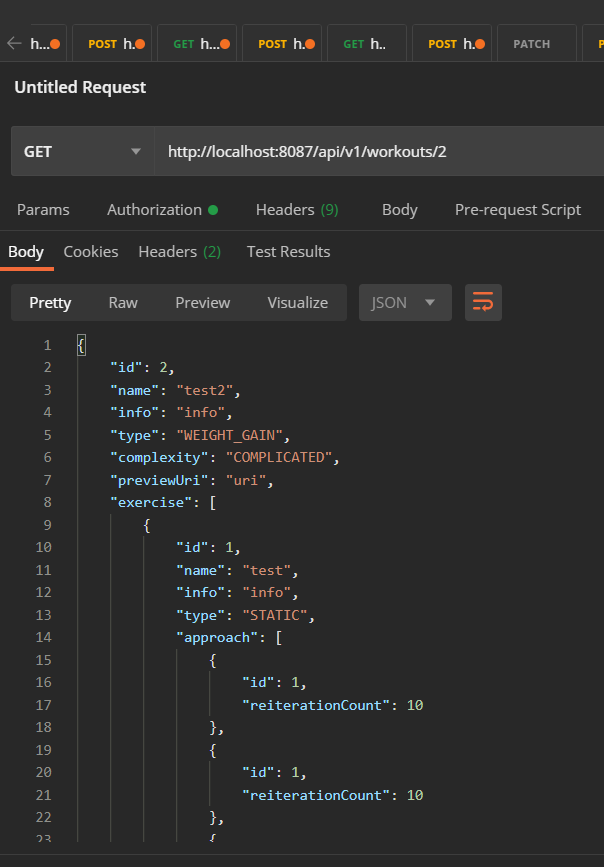
проверены запросы в Postman:



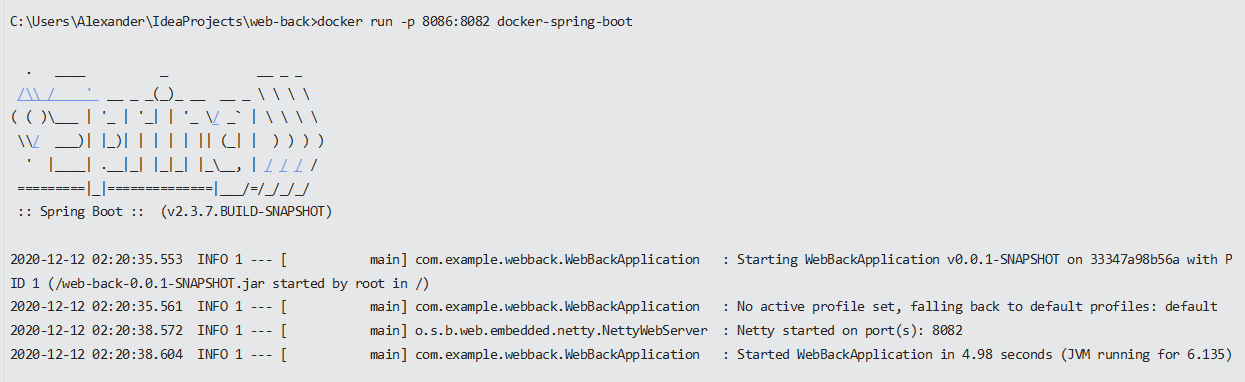








Скриншот запущенного Docker образа:



**Вывод:** В результате выполнения работы были рассмотрены основы работы docker с фреймфорком Sprint-Boot.