

ОТЧЕТ
ПО РЕЗУЛЬТАТАМ ВЫПОЛНЕНИЯ ЗАДАНИЯ
ДЕМОНСТРАЦИОННОГО ЭКЗАМЕНА
Вариант №1

Студент: Почернин Владислав Сергеевич

Организация: НИЯУ МИФИ

Группа: М24-535

Дата: 12.07.2025

Содержание

1	Задание	2
1.1	Постановка задачи	2
1.2	Обязательные требования	2
1.3	Все остальные детали реализации - на усмотрение разработчика	2
2	Приложение на основе Spring Boot	4
2.1	Инициализация проекта	4
2.2	Написание кода	4
2.2.1	Пакет model	4
2.2.2	Пакет repository	5
2.2.3	Пакет service	6
2.2.4	Пакет controller	6
2.2.5	Наполнение БД	7
2.2.6	Конфигурация	8
2.3	Работа приложения	8
2.3.1	Запуск	8
3	Заключение	9

1 Задание

1.1 Постановка задачи

Создать небольшое Java web-приложение на основе Spring Boot со следующим функционалом:

- 1) При обращении по эндпоинту `user-api/v1/users` (GET метод) приложение возвращает json-ответ со списком всех доступных пользователей.
- 2) При обращении по эндпоинту `user-api/v1/users` (POST метод) происходит добавление нового пользователя с заданными параметрами.
- 3) При обращении по эндпоинту `user-api/v1/additional-info` (GET метод) приложение возвращает json-ответ со списком всех пользователей, возраст которых больше либо равен заданному, отсортированный по имени (`firstName`) в алфавитном порядке. Заданный возраст передается в строке запроса в виде параметра (например, `user-api/v1/additional-info?age=18`), при этом данный эндпоинт `user-api/v1/additional-info` должен быть универсальным, т.е. работать для любого значения возраста.

1.2 Обязательные требования

- Пользователь описывается сущностью `User` с полями `Long id`, `String firstName`, `Integer age`, `Country country`, `Country` - это enum с названиями стран с некоторым набором возможных значений (заполнить минимум 5 произвольными странами).
- Данные всех пользователей хранятся в базе данных в таблице `users`, добавление новых пользователей также происходит в эту таблицу.
- Java, Spring Boot.
- Модули Spring WEB, Spring Data JPA.
- Приложение разбито на слои: репозиторий, сервис, (REST-)контроллер (использовать соответствующие Spring аннотации).
- Для взаимодействия с БД подключить к проекту и использовать in-memory H2 database.
- Функционал, описанный в пунктах 1, 2 и 3, реализовать через JPA репозиторий для сущности `User` (без написания SQL запросов).
- При запуске приложения должна создаваться таблица `users` минимум с 5 записями - по одной для каждой страны (допускается реализовать при помощи SQL скриптов).

1.3 Все остальные детали реализации - на усмотрение разработчика

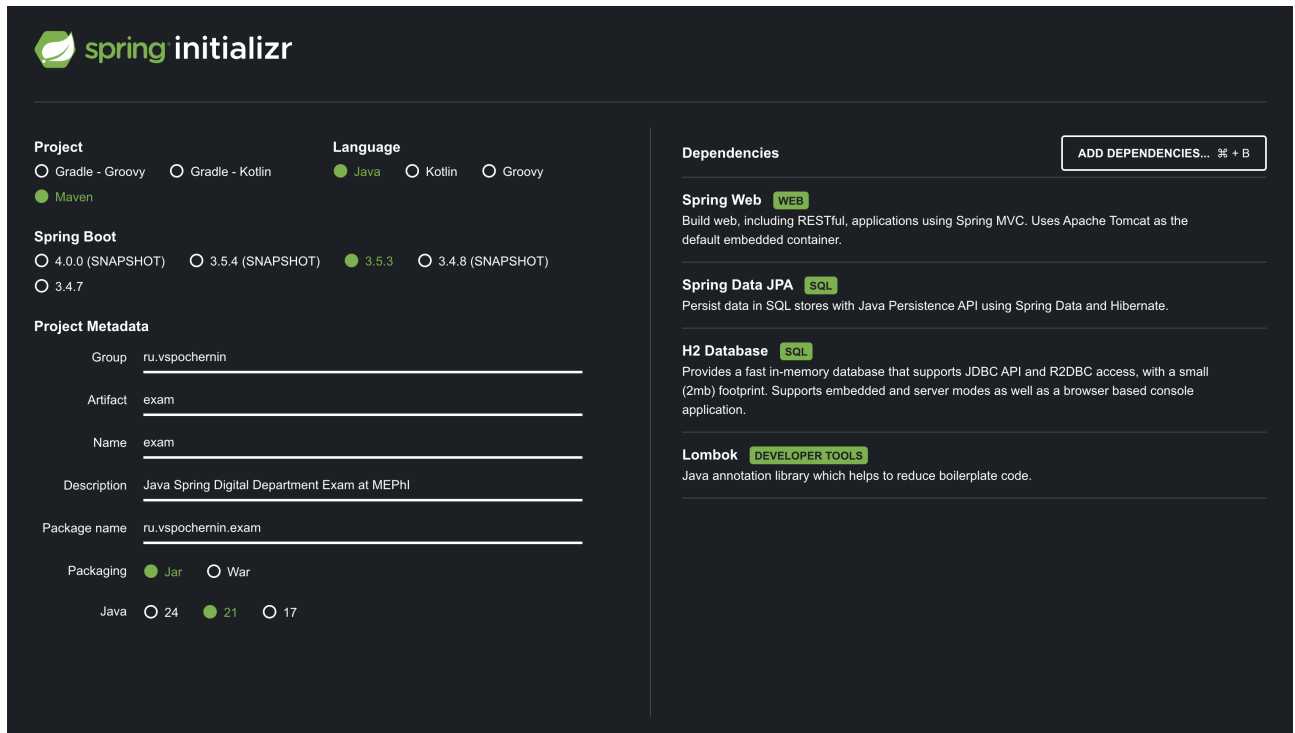
- Пояснение: под эндпоинтом понимается следующее - по умолчанию локально запущенное Spring Boot приложение работает на `http://localhost:8080/`, соответственно, для добавления нового пользователя необходимо отправить POST запрос (информация о новом пользователе указывается в теле запроса) на `http://localhost:8080/user-api/v1/users`.

- Для отправки запросов к приложению (для локального тестирования работы приложения) можно использовать любой REST клиент (Insomnia, Postman, IntelliJ IDEA Http Client, ...).
- Пояснение при использовании Community версии IntelliJ IDEA: для создания проекта использовать ссылку <https://start.spring.io/>. Там же произвести выбор необходимых зависимостей. Затем скачать сгенерированный проект и открыть его при помощи IntelliJ IDEA Community.

2 Приложение на основе Spring Boot

2.1 Инициализация проекта

Для инициализации проекта воспользуемся сервисом <https://start.spring.io/>.



The screenshot shows the Spring Initializr web form. On the left, under 'Project', 'Maven' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', '3.5.3' is selected. The 'Project Metadata' section includes fields for Group (ru.vspochernin), Artifact (exam), Name (exam), Description (Java Spring Digital Department Exam at MEPhI), and Package name (ru.vspochernin.exam). Under 'Packaging', 'Jar' is selected, and under 'Java', '21' is selected. On the right, the 'Dependencies' section shows 'Spring Web' (WEB), 'Spring Data JPA' (SQL), 'H2 Database' (SQL), and 'Lombok' (DEVELOPER TOOLS) all selected. A button 'ADD DEPENDENCIES... % + B' is at the top right of the dependencies section.

Рис. 1: Инициализация проекта

Мы будем использовать Java версии 21, сборщик Maven, а также актуальную на момент написания отчета версию Spring Boot - 3.5.3.

Из зависимостей были выбраны:

- Spring Web - для создания сервера, который сможет отвечать на HTTP запросы.
- Spring Data JPA - для взаимодействия с нашей H2 базой данных с помощью объектно-реляционного отображения.
- H2 Database - драйвер для работы H2 базы данных.
- Lombok - вспомогательный инструмент для уменьшения количества boilerplate кода.

2.2 Написание кода

2.2.1 Пакет model

Для начала реализуем классы-модели.

Создадим enum `Country`, задающий 5 стран:

```
package ru.vspochernin.exam.model;  
  
public enum Country {  
  
    RUSSIA,  
    USA,
```

```
FRANCE,  
GERMANY,  
JAPAN  
}
```

А также класс пользователя, который будет связан с таблицей `users` базы данных:

```
package ru.vspochernin.exam.model;  
  
import jakarta.persistence.*;  
import lombok.*;  
  
@Entity  
@Table(name = "users")  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class User {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    private String firstName;  
  
    private Integer age;  
  
    @Enumerated(EnumType.STRING)  
    private Country country;  
  
    public User(String firstName, Integer age, Country country) {  
        this.firstName = firstName;  
        this.age = age;  
        this.country = country;  
    }  
}
```

Важными моментами здесь является аннотация `@Table(name = "users")`, задающая название таблицы БД, а также аннотация `@Enumerated(EnumType.STRING)`, объясняющая, что enum нужно обрабатывать как строку (без нее в базу записывались бы порядковые номера). `@GeneratedValue(strategy = GenerationType.IDENTITY)` над полем `id` делегирует установку ID на уровень базы данных, нам не нужно будет задумываться об этом (например, при добавлении пользователей мы будем использовать конструктор без поля `id`, но база данных сама проставит это поле в модель).

2.2.2 Пакет repository

Далее реализуем класс-репозиторий для пользователя:

```
package ru.vspochernin.exam.repository;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import ru.vspochernin.exam.model.User;  
  
import java.util.List;
```

```
public interface UserRepository extends JpaRepository<User, Long> {

    List<User> findByAgeGreaterThanEqualOrderByFirstNameAsc(Integer age);
}
```

Здесь мы добавили метод `findByAgeGreaterThanEqualOrderByFirstNameAsc` для того, чтобы использовать его в эндпоинте `additional-info`.

Важный момент: хотя в задании сказано использовать соответствующие слоям Spring аннотации, в репозитории совсем не обязательно ставить аннотацию `@Repository`, в случае, если это JPA Repository (<https://stackoverflow.com/questions/44069367/repository-not-necessary-when-implementing-jparepository>).

2.2.3 Пакет service

Создадим класс-сервис для нашей программы:

```
package ru.vspochernin.exam.service;

import org.springframework.stereotype.Service;
import ru.vspochernin.exam.model.User;
import ru.vspochernin.exam.repository.UserRepository;

import java.util.List;

@Service
public class UserService {

    private final UserRepository userRepository;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    public User addUser(User user) {
        return userRepository.save(user);
    }

    public List<User> getUsersByAgeGteSorted(Integer age) {
        return userRepository.findByAgeGreaterThanEqualOrderByFirstNameAsc(age);
    }
}
```

Класс использует созданный ранее репозиторий, и в нем реализованы методы для получения всех пользователей, добавления нового пользователя, а также для получения списках всех пользователей, возраст которых больше либо равен заданному, отсортированный по имени (`firstName`) в алфавитном порядке.

2.2.4 Пакет controller

Наконец, реализуем контроллер для обработки входящих запросов:

```
package ru.vspochernin.exam.controller;
```

```

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import ru.vspochernin.exam.model.User;
import ru.vspochernin.exam.service.UserService;

import java.util.List;

@RestController
@RequestMapping("/user-api/v1")
public class UserController {

    private final UserService userService;

    public UserController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping("/users")
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }

    @PostMapping("/users")
    @ResponseStatus(HttpStatus.CREATED)
    public User addUser(@RequestBody User user) {
        return userService.addUser(user);
    }

    @GetMapping("/additional-info")
    public List<User> getByAge(@RequestParam Integer age) {
        return userService.getUsersByAgeGteSorted(age);
    }
}

```

Для соответствия условиям задачи здесь задается базовый путь с помощью аннотации `@RequestMapping("/user-api/v1")`. А в случае создания пользователя также возвращается статус 201 CREATED (вместо стандартного 200 OK) с помощью аннотации `@ResponseStatus(HttpStatus.CREATED)`.

2.2.5 Наполнение БД

Для наполнения базы данных при старте приложения был модернизирован созданный при инициализации проекта файл `ExamApplication`.

```

package ru.vspochernin.exam;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import ru.vspochernin.exam.model.Country;
import ru.vspochernin.exam.model.User;
import ru.vspochernin.exam.repository.UserRepository;

@SpringBootApplication
public class ExamApplication {

    public static void main(String[] args) {

```



```

        SpringApplication.run(ExamApplication.class, args);
    }

    @Bean
    public CommandLineRunner initDatabase(UserRepository userRepository) {
        return args -> {
            userRepository.save(new User("Alexey", 15, Country.RUSSIA));
            userRepository.save(new User("John", 30, Country.USA));
            userRepository.save(new User("Pierre", 14, Country.FRANCE));
            userRepository.save(new User("Hans", 40, Country.GERMANY));
            userRepository.save(new User("Hideo", 61, Country.JAPAN));
        };
    }
}

```

В него был добавлен бин `CommandLineRunner`, который при запуске сохранял в репозиторий (то есть в нашу базу) 5 пользователей.

2.2.6 Конфигурация

Для того, чтобы наше приложение работало корректно, нам нужно настроить его в файле `application.properties`:

```

spring.application.name=exam

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true

spring.h2.console.enabled=true

```

В начале файла задается имя приложения. Затем настраивается соединение с БД (адрес, тип драйвера, логин и пароль). Далее мы указываем нужный нам диалект, просим Hibernate самому создавать схему БД, а также устанавливаем отображение всех sql запросов, которые генерирует и исполняет Hibernate в консоли. Последняя строка включает админку для нашей БД, к которой можно будет обратиться по адресу `http://localhost:8080/h2-console/`.

2.3 Работа приложения

2.3.1 Запуск

Запустим наше приложение

3 Заключение