

**ОТЧЕТ**  
**ПО РЕЗУЛЬТАТАМ ВЫПОЛНЕНИЯ ЗАДАНИЯ**  
**ДЕМОНСТРАЦИОННОГО ЭКЗАМЕНА**  
Вариант №1

**Студент:** Почернин Владислав Сергеевич

**Организация:** НИЯУ МИФИ

**Группа:** М24-535

**Дата:** 12.07.2025

## Содержание

<b>1</b>	<b>Задание</b>	<b>2</b>
1.1	Постановка задачи . . . . .	2
1.2	Обязательные требования . . . . .	2
1.3	Все остальные детали реализации - на усмотрение разработчика . . . . .	2
<b>2</b>	<b>Приложение на основе Spring Boot</b>	<b>4</b>
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
2.3.2	Проверка запросов . . . . .	10
<b>3</b>	<b>Заключение</b>	<b>18</b>

# 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

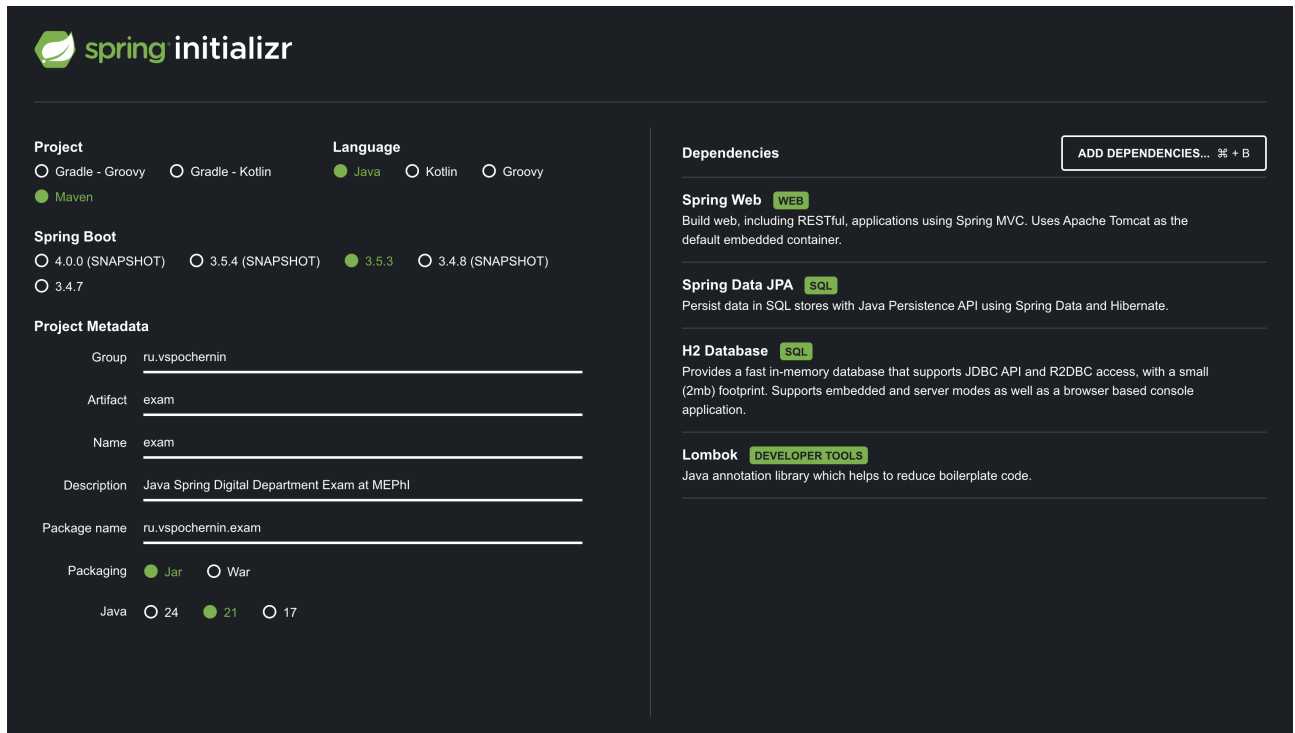
- Пояснение: под эндпоинтом понимается следующее - по умолчанию локально запущенное Spring Boot приложение работает на `http://localhost:8080/`, соответственно, для добавления нового пользователя необходимо отправить POST запрос (информация о новом пользователе указывается в теле запроса) на `http://localhost:8080/user-api/v1/users`.

- 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 interface. 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 contains: Group (ru.vspochernin), Artifact (exam), Name (exam), Description (Java Spring Digital Department Exam at MEPhI), Package name (ru.vspochernin.exam), Packaging (Jar), and Java version (21). On the right, under 'Dependencies', 'Spring Web' (WEB), 'Spring Data JPA' (SQL), 'H2 Database' (SQL), and 'Lombok' (DEVELOPER TOOLS) are selected. An 'ADD DEPENDENCIES...' button is visible 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  
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> findByAgeGreaterThanOrEqualToOrderByFirstNameAsc(Integer age);  
}
```

Здесь мы добавили метод `findByAgeGreaterThanOrEqualToOrderByFirstNameAsc` 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.findByAgeGreaterThanOrEqualToOrderByFirstNameAsc(age);  
    }  
}
```

Класс использует созданный ранее репозиторий, и в нем реализованы методы для получения всех пользователей, добавления нового пользователя, а также для получения списка 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 админ-панель для нашей БД, к которой можно будет обратиться по адресу `http://localhost:8080/h2-console/`.

## 2.3 Работа приложения

### 2.3.1 Запуск

Включим наше приложение, запустим метод `main()` класса `ExamApplication`.

```
Run ExamApplication x
Console
2025-07-12T15:40:59.548+03:00 INFO 48624 --- [exam] [main] org.hibernate.orm.connections.pooling : HHN10001005: Database info:
Database JDBC URL (Connecting through datasource 'HikariDataSource (HikariPool-1)')
Database driver: undefined/unknown
Database version: 2.3.232
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2025-07-12T15:40:59.745+03:00 INFO 48624 --- [exam] [main] o.h.m.i.EntityInstantiator$PojoStandard : HHN000102: No default (no-argument) constructor for class: ru.vspochernin.examin.model.User (class must be in:
2025-07-12T15:40:59.803+03:00 INFO 48624 --- [exam] [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHN000409: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integ
Hibernate: drop table if exists users cascade
Hibernate: create table users (age integer, id bigint generated by default as identity, first_name varchar(255), country enum ('FRANCE','GERMANY','JAPAN','RUSSIA','USA'), primary key (id))
2025-07-12T15:40:59.816+03:00 INFO 48624 --- [exam] [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2025-07-12T15:40:59.919+03:00 WARN 48624 --- [exam] [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view ren
2025-07-12T15:41:00.032+03:00 INFO 48624 --- [exam] [main] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:testdb'
2025-07-12T15:41:00.059+03:00 INFO 48624 --- [exam] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2025-07-12T15:41:00.063+03:00 INFO 48624 --- [exam] [main] ru.vspochernin.examin.ExamApplication : Started ExamApplication in 1.32 seconds (process running for 3.12)
Hibernate: insert into users (age,country,first_name,id) values (?,?,?,default)
Hibernate: insert into users (age,country,first_name,id) values (?,?,?,default)
Hibernate: insert into users (age,country,first_name,id) values (?,?,?,default)
Hibernate: insert into users (age,country,first_name,id) values (?,?,?,default)
Hibernate: insert into users (age,country,first_name,id) values (?,?,?,default)
```

Рис. 2: Запуск приложения

Из логов в консоли можно увидеть, что у нас успешно инициализировалась база данных, создавалась схема БД, а также было добавлено 5 пользователей.

Чтобы убедиться в этом, зайдем в админ-панель базы данных по адресу <http://localhost:8080/h2-console/>:

русский

НастройкиИнструментыПомощь

Логин

Сохранить настройки:

Generic H2 (Embedded)

Имя настройки:

Generic H2 (Embedded)

СохранитьУдалить

Класс драйвера:

org.h2.Driver

JDBC URL:

jdbc:h2:mem:testdb

Имя пользователя:

sa

Пароль:

СоединитьсяТестовое соединение

Рис. 3: Вход в админ-панель

И выполним поиск по всем пользователям:

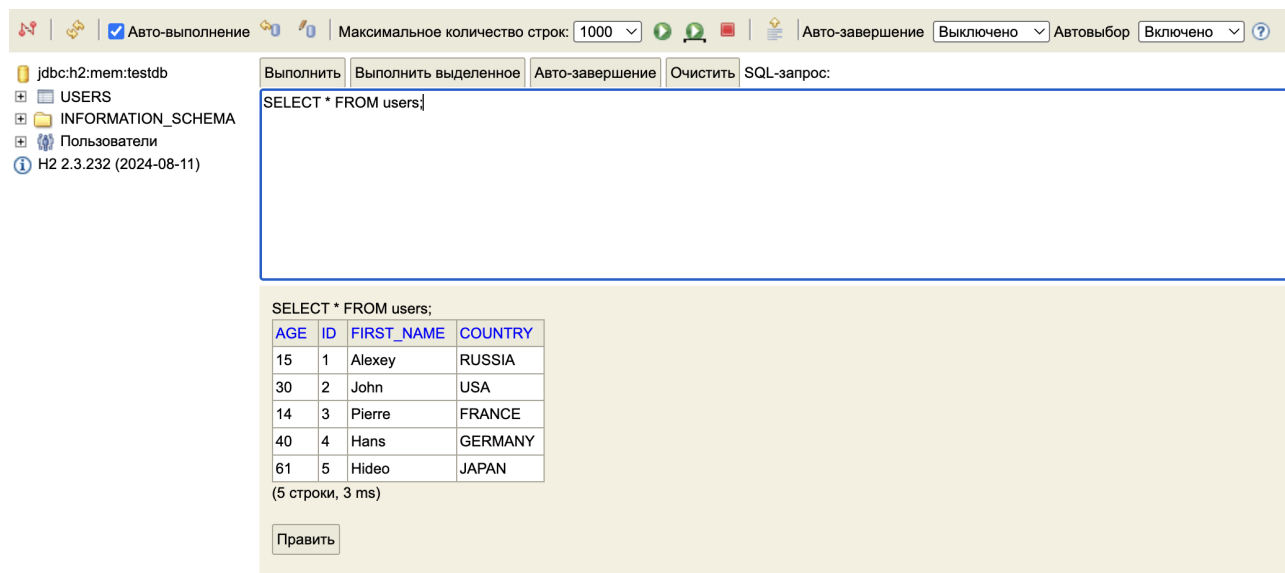


Рис. 4: Поиск по всем пользователям

Как можно заметить, в базу действительно добавились те пользователи, которых мы указывали в коде.

### 2.3.2 Проверка запросов

Проверим работу наших запросов. Для этого будем использовать Postman. Выполним первый запрос и получим список всех пользователей:

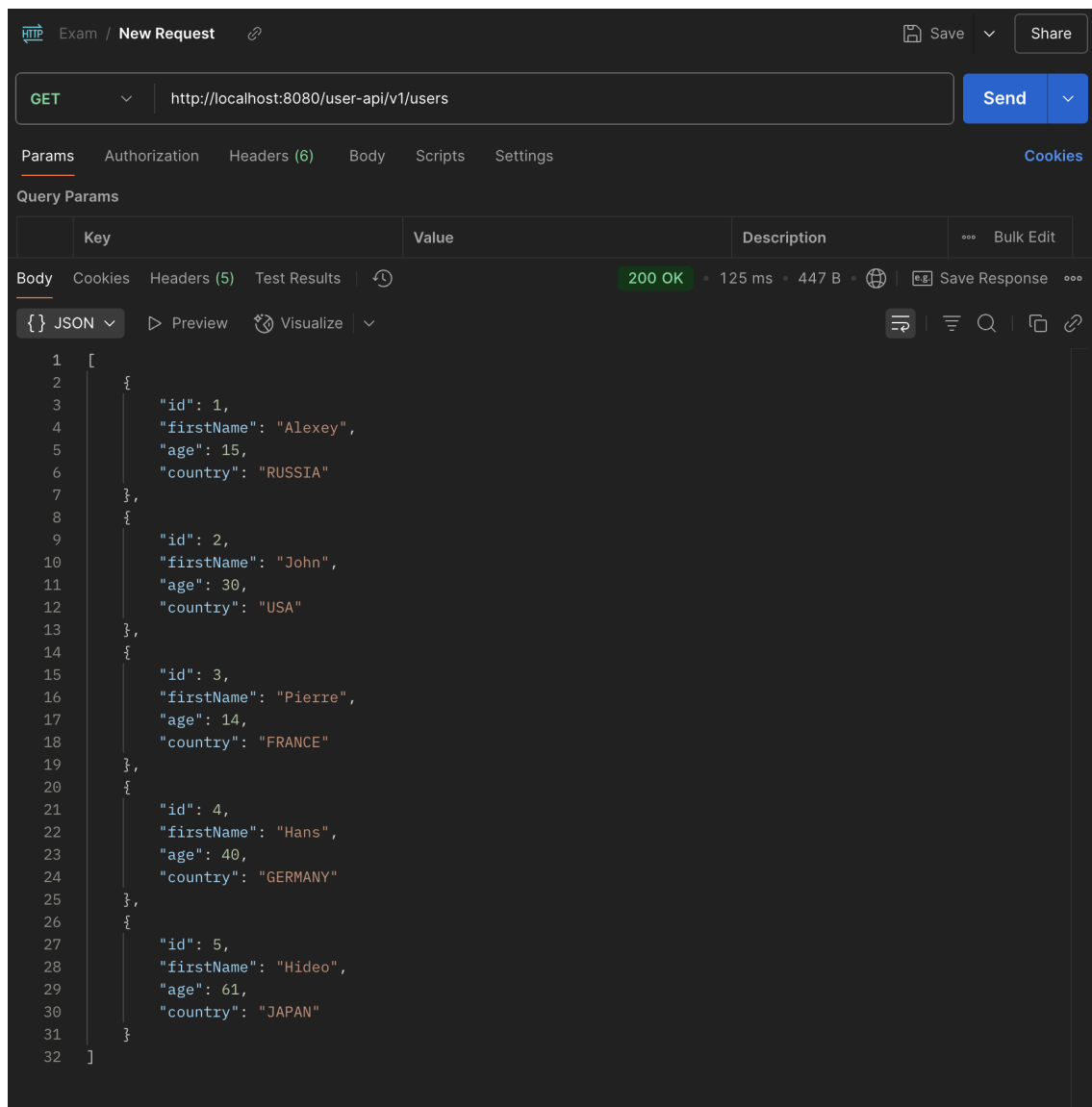


Рис. 5: Получение списка всех пользователей

Далее выполним два запроса, чтобы добавить двух новых пользователей:

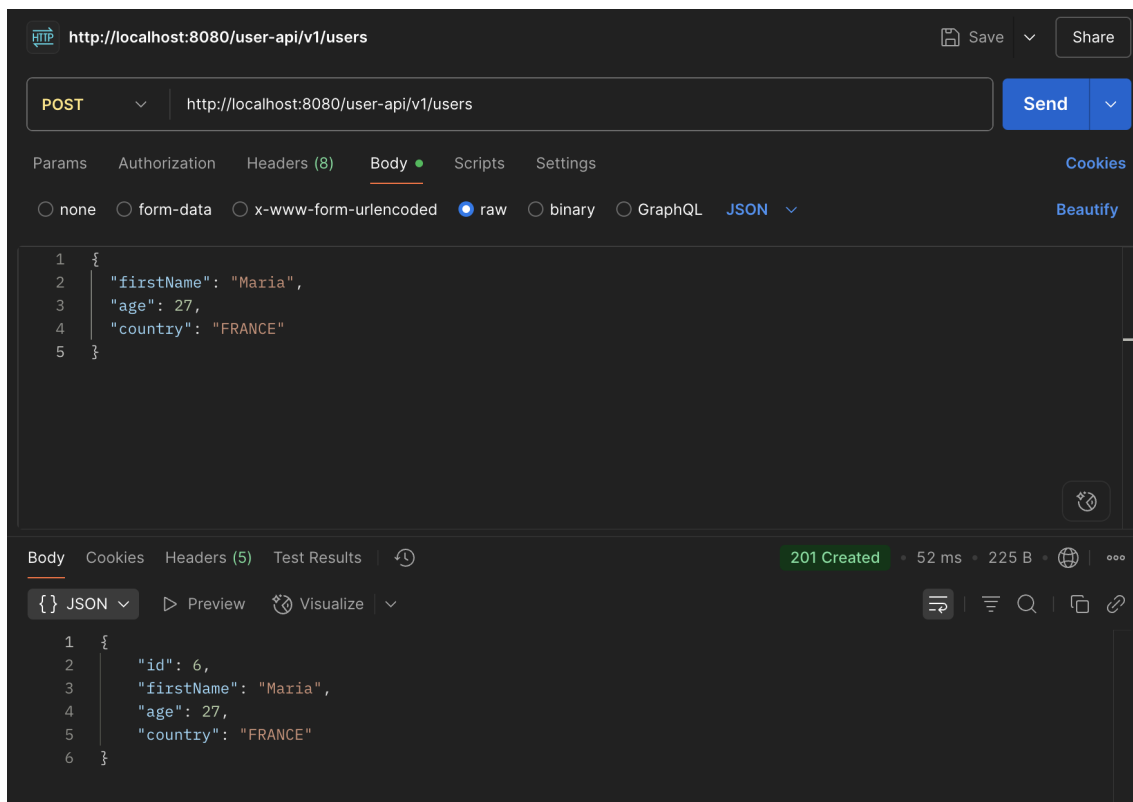


Рис. 6: Добавление пользователя Marina

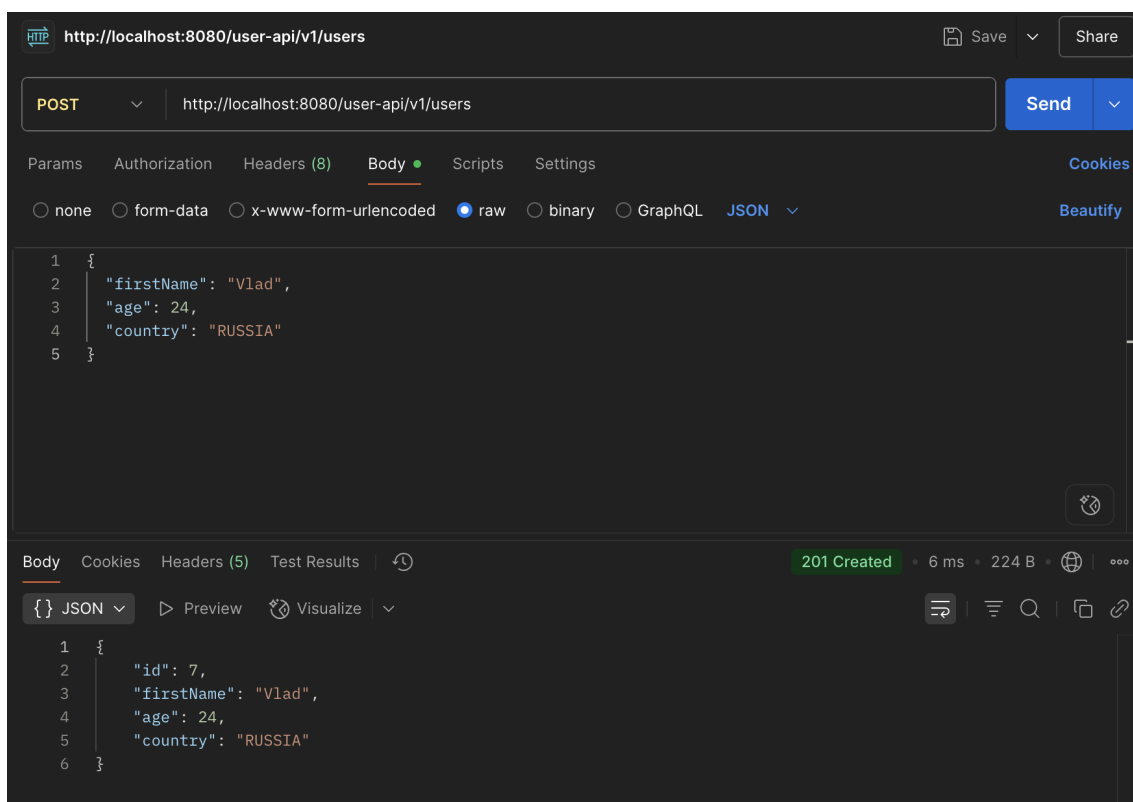


Рис. 7: Добавление пользователя Vlad

Снова запросив всех пользователей, увидим, что новые также появились в списке:

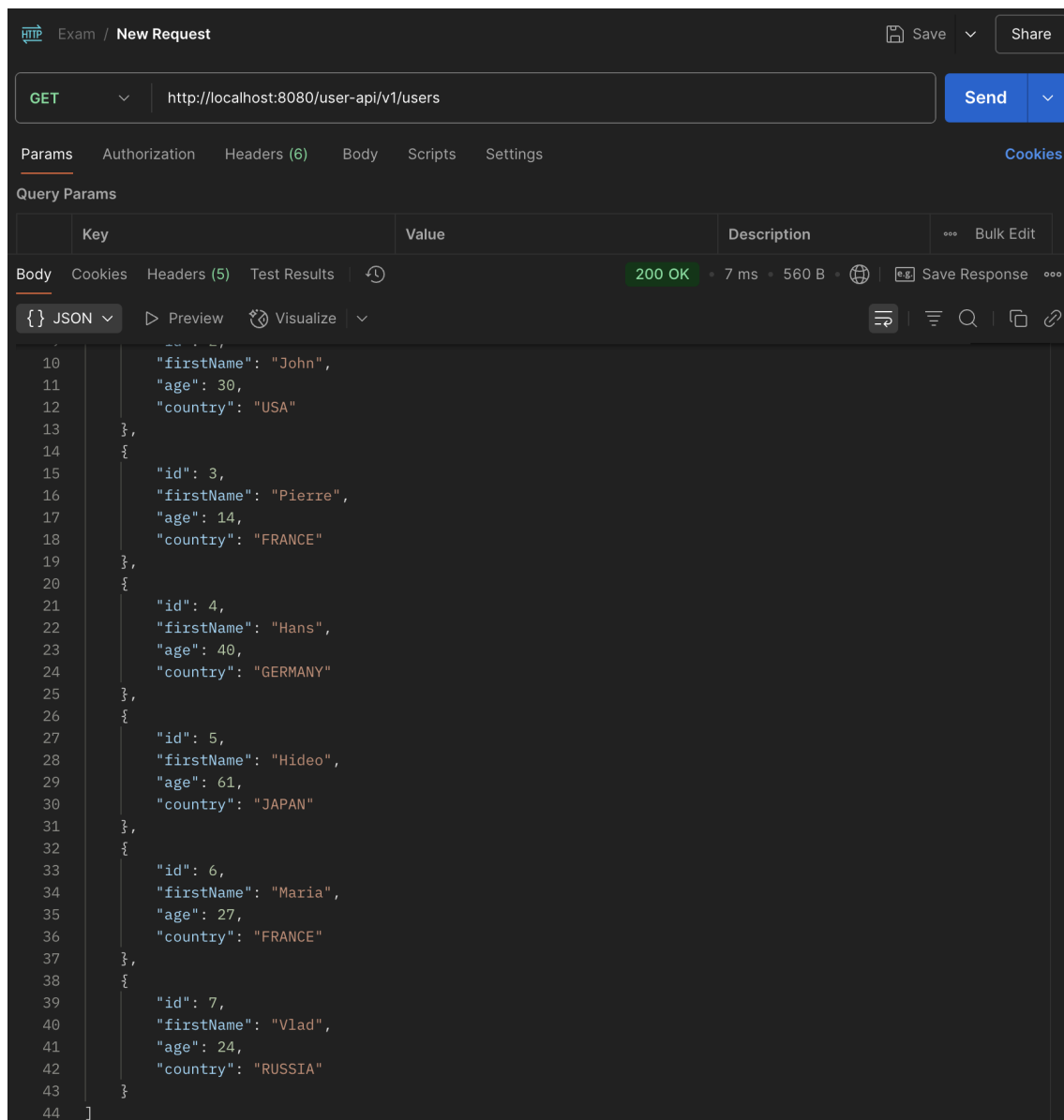


Рис. 8: Получение списка всех пользователей после добавления новых

Наконец, протестируем эндпоинт `additional-info`.  
Сначала забудем передать ему параметр `age`:

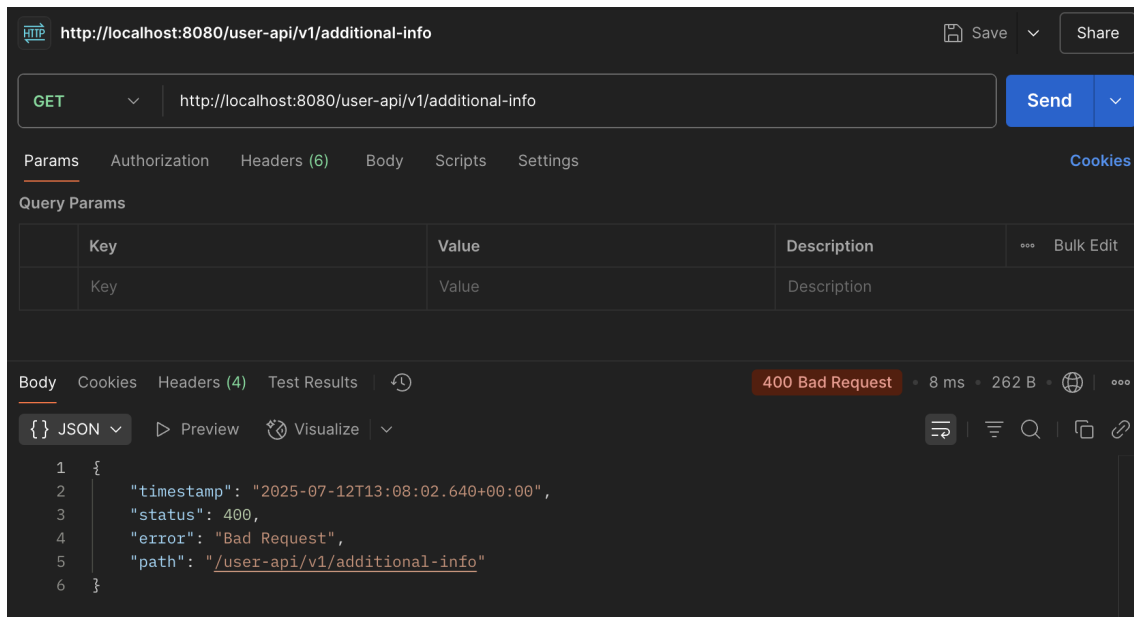


Рис. 9: Ошибка на отсутствие параметра age

Ту же ошибку получим при попытке передать в age НЕ число:

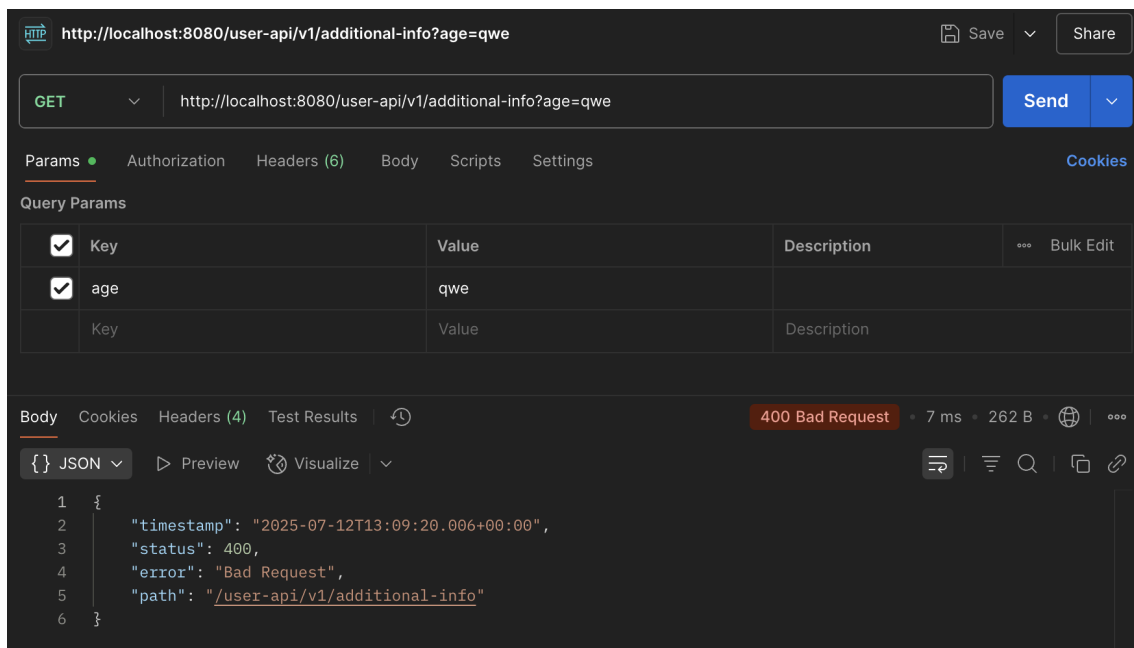


Рис. 10: Ошибка на передачу в age НЕ числа

Запишем в age отрицательное число:

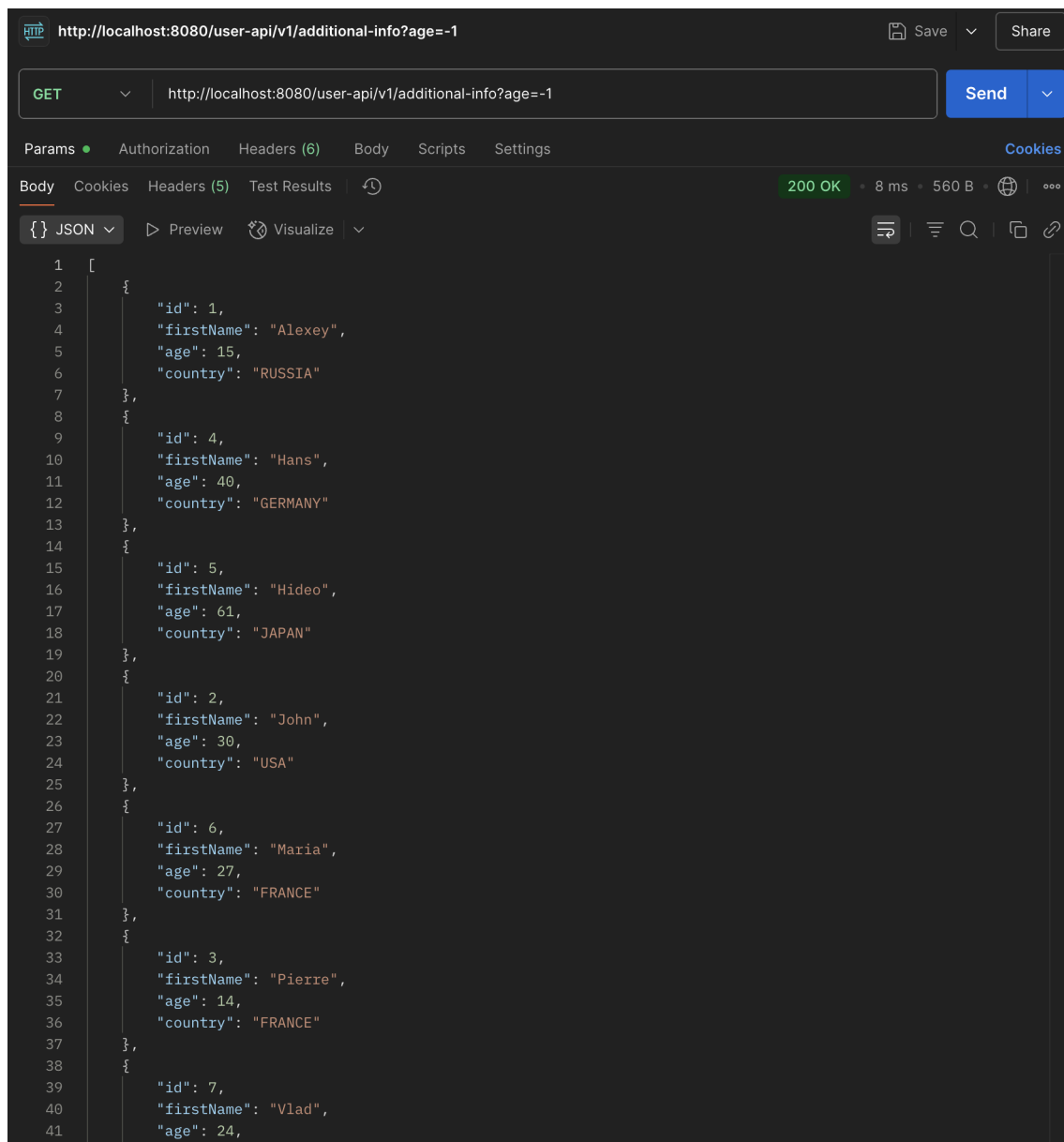


Рис. 11: Отрицательное число в age

Нам вернутся все пользователи, но отсортированные по имени по алфавиту. Запишем в age 30:



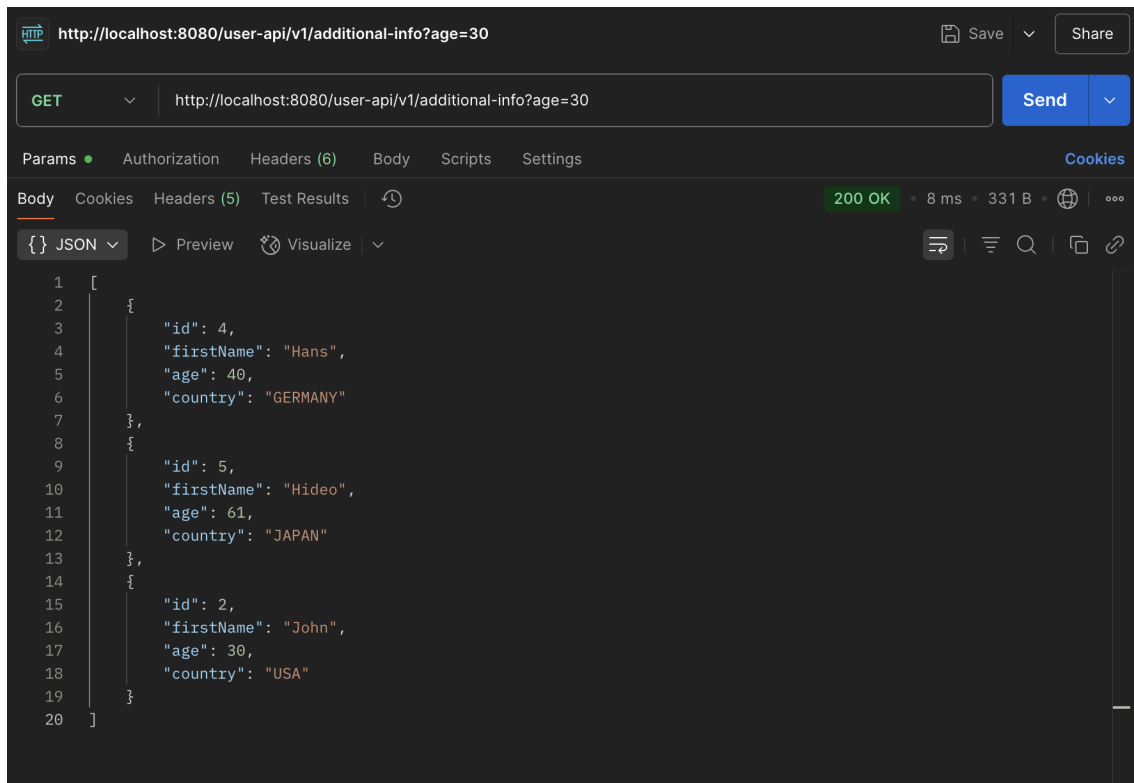


Рис. 12: Число 30 в age

Нам вернутся только пользователи с возрастом 30 и выше, отсортированные по имени по алфавиту.

Наконец, запишем в age 80:

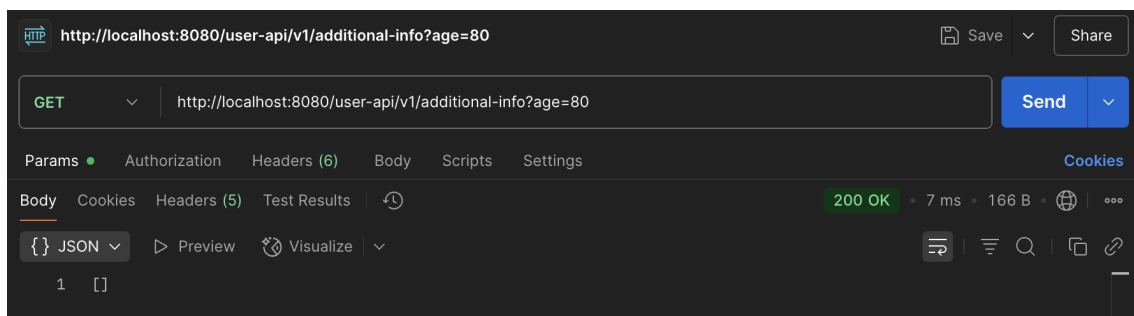


Рис. 13: Число 80 в age

И закономерно получим пустой список пользователей.

Для верности заглянем в базу данных через админ-панель:

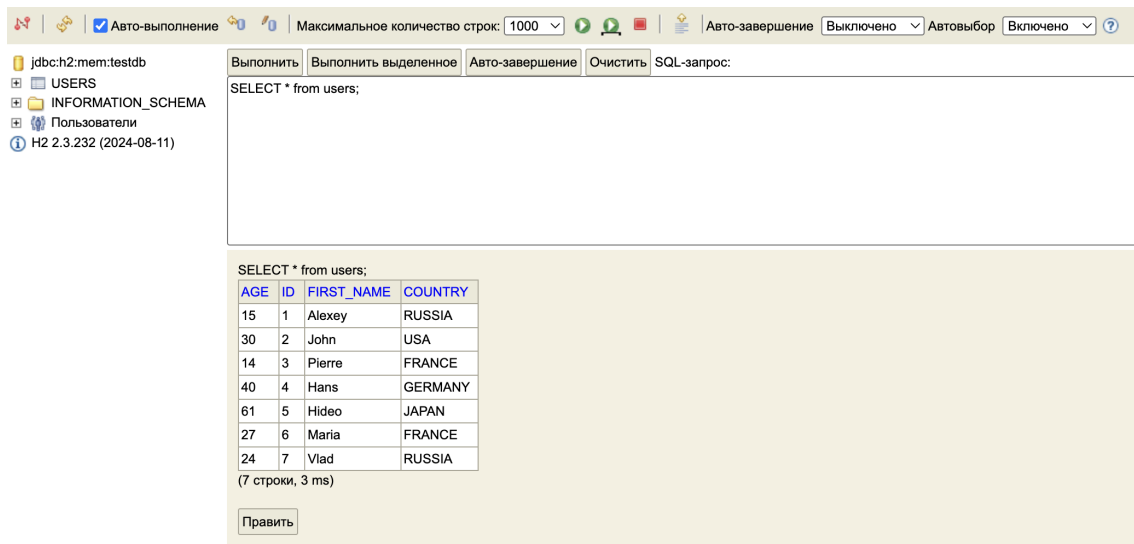


Рис. 14: Админ-панель после добавления пользователей

Увидим, что добавленные нами пользователи также там появились.

Также интересно посмотреть, что происходило в консоли во время наших запросов:

```

Hibernate: select u1_0.id,u1_0.age,u1_0.country,u1_0.first_name from users u1_0
Hibernate: insert into users (age,country,first_name,id) values (?,?,?,default)
Hibernate: insert into users (age,country,first_name,id) values (?,?,?,default)
Hibernate: select u1_0.id,u1_0.age,u1_0.country,u1_0.first_name from users u1_0
Hibernate: select u1_0.id,u1_0.age,u1_0.country,u1_0.first_name from users u1_0 where u1_0.age>=? order by u1_0.first_name
2025-07-12T16:05:48.725+03:00 WARN 60648 --- [exam] [nio-8080-exec-9] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler$ExceptionHandler:100]
2025-07-12T16:08:02.639+03:00 WARN 60648 --- [exam] [nio-8080-exec-2] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler$ExceptionHandler:100]
2025-07-12T16:09:20.004+03:00 WARN 60648 --- [exam] [nio-8080-exec-3] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler$ExceptionHandler:100]
Hibernate: select u1_0.id,u1_0.age,u1_0.country,u1_0.first_name from users u1_0 where u1_0.age>=? order by u1_0.first_name
Hibernate: select u1_0.id,u1_0.age,u1_0.country,u1_0.first_name from users u1_0 where u1_0.age>=? order by u1_0.first_name
Hibernate: select u1_0.id,u1_0.age,u1_0.country,u1_0.first_name from users u1_0 where u1_0.age>=? order by u1_0.first_name

```

Рис. 15: Консоль во время запросов

Hibernate переводил наш код в обычные SQL запросы и выполнял их, а когда мы вводили некорректные данные для age - срабатывал `DefaultHandlerExceptionResolver` (он вызывается 3 раза, а не два, поскольку я вызвал метод с некорректным age еще один лишний раз).

### 3 Заключение

В ходе выполнения задания было разработано и протестировано простое Spring Boot веб-приложение, реализующее работу с сущностью пользователя в in-memory базе данных H2.

Все требования технического задания были выполнены: реализованы необходимые REST-эндпоинты, соблюдена многослойная архитектура, обеспечена пользователей при старте.

Работа позволила мне повторить навыки проектирования структурированного Java-приложения на Spring Boot, поработать с JPA и настройкой in-memory базы данных, а также отработать основные принципы построения REST API. Применяя данные принципы можно расширять данное приложение, а также создавать новые.