

Практическая работа Java

Настройка БД

```
1  spring:
2    datasource:
3      url: jdbc:h2:mem:taskdb
4      driverClassName: org.h2.Driver
5      username: sa
6      password: password
7    h2:
8      console:
9        enabled: true
10   jpa:
11     hibernate:
12       ddl-auto: update
```

Создание модели задачи

```
1  package com.example.demo.model;
2
3  import jakarta.persistence.Entity;
4  import jakarta.persistence.GeneratedValue;
5  import jakarta.persistence.GenerationType;
6  import jakarta.persistence.Id;
7  import lombok.Data;
8  import lombok.NoArgsConstructor;
9  import lombok.AllArgsConstructor;
10
11  import java.time.LocalDateTime;
12
13  @Entity
14  @Data
15  @NoArgsConstructor
16  @AllArgsConstructor
17  public class Task {
18      @Id
19      @GeneratedValue(strategy = GenerationType.IDENTITY)
20      private Long id;
21      // Ctrl+L to chat, Ctrl+K to generate
22      private String title;
23      private String description;
24      private String status;
25      private LocalDateTime createdAt;
26  }
```

Создание репозитория для задач

```

1  package com.example.demo.repository;
2
3  import com.example.demo.model.Task;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7  @Repository
8  public interface TaskRepository extends JpaRepository<Task, Long> {
9  } |

```

REST контроллер

класс для обработки ошибок

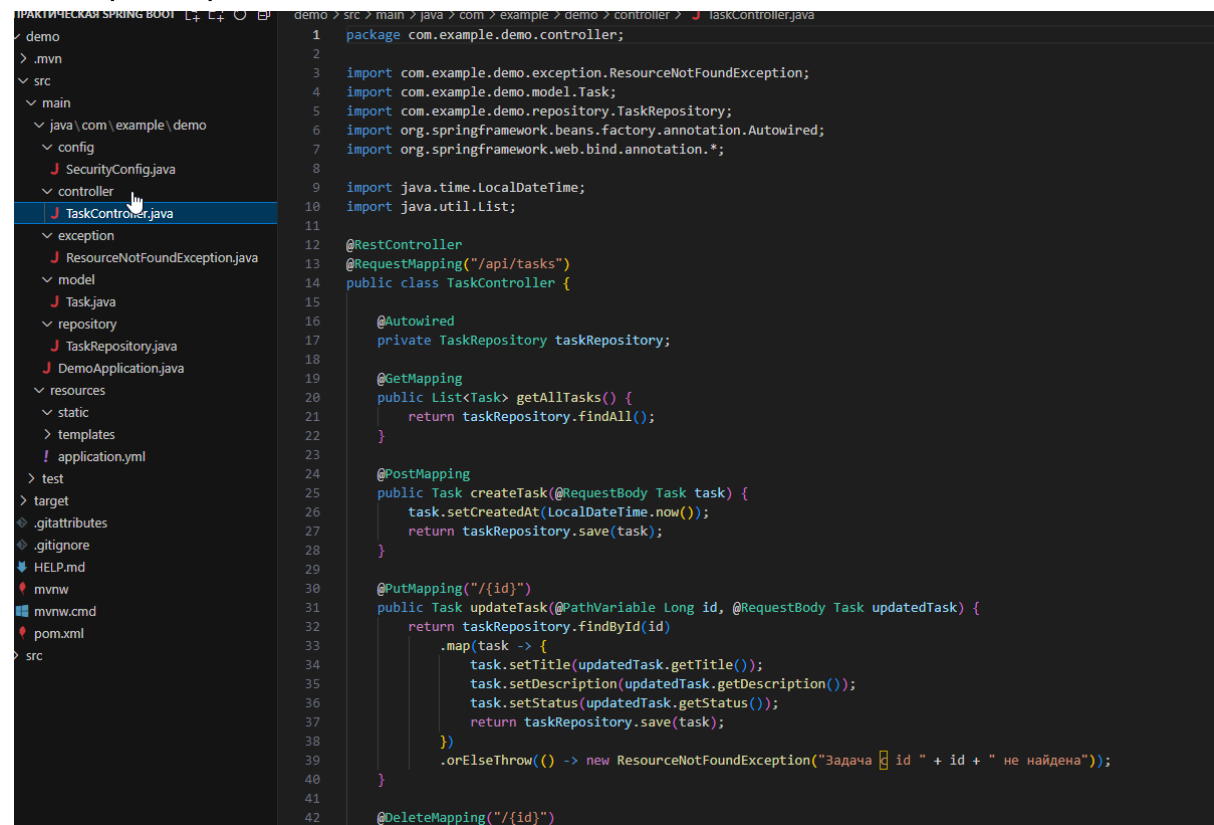
```

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(String message) {
        super(message);
    }
}

```

и контроллер:



```

demo \src> main> java> com> example> demo> controller> TaskController.java
1  package com.example.demo.controller;
2
3  import com.example.demo.exception.ResourceNotFoundException;
4  import com.example.demo.model.Task;
5  import com.example.demo.repository.TaskRepository;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.web.bind.annotation.*;
8
9  import java.time.LocalDateTime;
10 import java.util.List;
11
12 @RestController
13 @RequestMapping("/api/tasks")
14 public class TaskController {
15
16     @Autowired
17     private TaskRepository taskRepository;
18
19     @GetMapping
20     public List<Task> getAllTasks() {
21         return taskRepository.findAll();
22     }
23
24     @PostMapping
25     public Task createTask(@RequestBody Task task) {
26         task.setCreatedAt(LocalDateTime.now());
27         return taskRepository.save(task);
28     }
29
30     @PutMapping("/{id}")
31     public Task updateTask(@PathVariable Long id, @RequestBody Task updatedTask) {
32         return taskRepository.findById(id)
33             .map(task -> {
34                 task.setTitle(updatedTask.getTitle());
35                 task.setDescription(updatedTask.getDescription());
36                 task.setStatus(updatedTask.getStatus());
37                 return taskRepository.save(task);
38             })
39             .orElseThrow(() -> new ResourceNotFoundException("Задача с id " + id + " не найдена"));
40     }
41
42     @DeleteMapping("/{id}")

```

Добавление безопасности

добавим зависимость в pom.xml

```
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
</dependencies>
```

обновляем application.yml

```
demo > src > main > resources > ! application.yml
1  spring:
2    datasource:
3      url: jdbc:h2:mem:taskdb
4      driverClassName: org.h2.Driver
5      username: sa
6      password: password
7    h2:
8      console:
9        enabled: true
10   jpa:
11     hibernate:
12       ddl-auto: update
13
14   security:
15     user:
16       name: admin
17       password: admin123
```

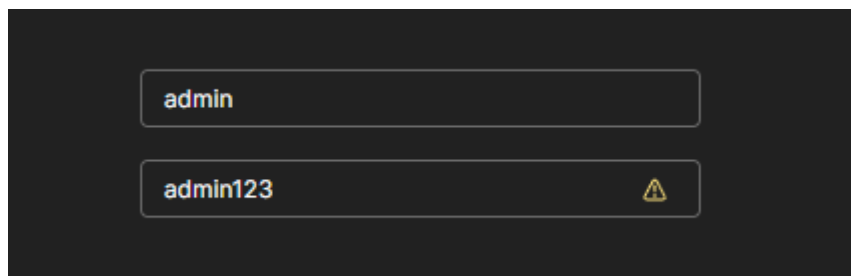
Создадим конфигурацию безопасности

```
demo 1 package com.example.demo.config;
      2
      3 import org.springframework.context.annotation.Bean;
      4 import org.springframework.context.annotation.Configuration;
      5 import org.springframework.security.config.Customizer;
      6 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
      7 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
      8 import org.springframework.security.web.SecurityFilterChain;
      9
     10 @Configuration
     11 @EnableWebSecurity
     12 public class SecurityConfig {
     13
     14     @Bean
     15     public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
     16         http
     17             .csrf(csrf -> csrf.disable())
     18             .headers(headers -> headers
     19                 .frameOptions()
     20                 .sameOrigin()
     21             )
     22             .authorizeHttpRequests(auth -> auth
     23                 .requestMatchers("/h2-console/**").permitAll()
     24                 .requestMatchers("/api/tasks/**").authenticated()
     25                 .anyRequest().authenticated()
     26             )
     27             .httpBasic(Customizer.withDefaults());
     28
     29         return http.build();
     30     }
     31 }
```

Тестирование приложения

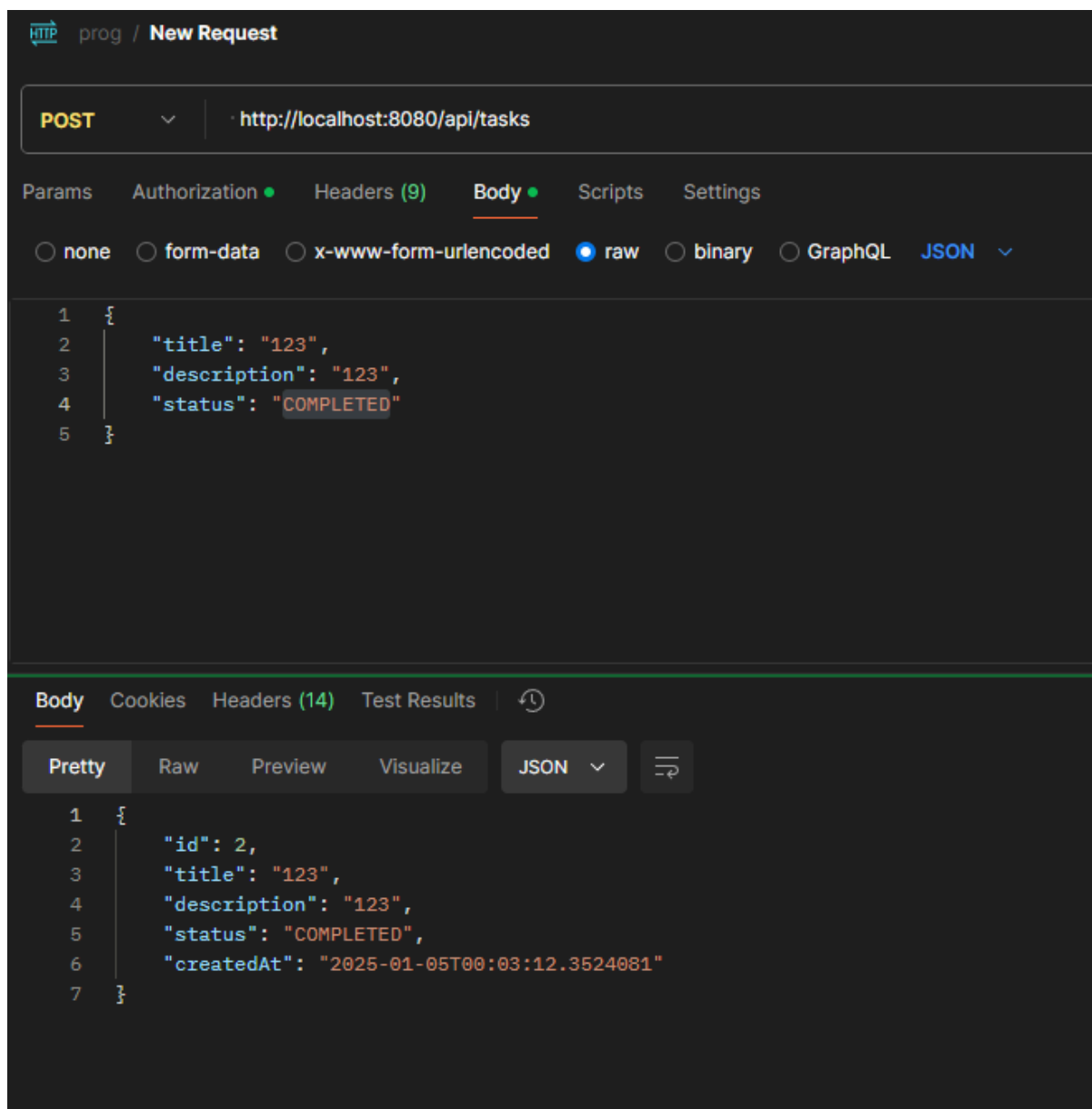
1. Создание задачи (POST)

указываем basic auth в постмане



admin

admin123



HTTP prog / New Request

POST http://localhost:8080/api/tasks

Params Authorization Headers (9) Body Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {
2   "title": "123",
3   "description": "123",
4   "status": "COMPLETED"
5 }
```

Body Cookies Headers (14) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "title": "123",
4   "description": "123",
5   "status": "COMPLETED",
6   "createdAt": "2025-01-05T00:03:12.3524081"
7 }
```

2. Проверка в H2

← ↻ 🌐 localhost:8080 H2 Console

English ▾ Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded) ▾

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:taskdb

User Name: sa

Password:

Connect Test Connection

← ↻ 🌐 localhost:8080 H2 Console 90 %

🔊 🔍 ☒ Auto commit 🏷️ 📄 Max rows: 1000 🟢 🟢 📄 Auto complete Off ▾ Auto select On ▾ ?

📁 jdbc:h2:mem:taskdb

📁 TASK

📁 INFORMATION_SCHEMA

👤 Users

📄 H2 2.2.224 (2023-09-17)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM TASK;

ID	CREATED_AT	DESCRIPTION	STATUS	TITLE
1	2025-01-04 21:07:00.221665	Описание первой задачи	NEW	Первая задача

(1 row, 0 ms)

Edit

3. Получение списка задач (GET)

The screenshot shows the HTTP client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/api/tasks
- Auth Type:** Basic Auth
- Username:** admin
- Password:** admin123
- Status:** 200 OK
- Response Body (Pretty):**

```
1  [
2    {
3      "id": 1,
4      "title": "Первая задача",
5      "description": "Описание первой задачи",
6      "status": "NEW",
7      "createdAt": "2025-01-04T21:07:00.221665"
8    },
9    {
10     "id": 2,
11     "title": "Первая задача",
12     "description": "Описание первой задачи",
13     "status": "NEW",
14     "createdAt": "2025-01-04T21:07:22.156837"
15   }
16 ]
```

4. Создание второй задачи (POST)

The screenshot displays a REST client interface with a dark theme. At the top, the URL bar shows 'http://localhost:8080/api/tasks' with a 'POST' method selected. A 'Send' button is visible. Below the URL bar, tabs for 'Params', 'Auth', 'Headers (9)', 'Body', 'Scripts', and 'Settings' are present, with 'Body' being the active tab. The 'Body' tab shows a JSON payload in 'raw' mode:

```
1 {
2   "title": "Вторая задача",
3   "description": "Описание второй задачи",
4   "status": "IN_PROGRESS"
5 }
```

 Below the body editor, the response status is '200 OK' with a response time of '71 ms' and a size of '596 B'. A 'Save Response' button is also present. At the bottom, the response is displayed in 'Pretty' mode as JSON:

```
1 {
2   "id": 3,
3   "title": "Вторая задача",
4   "description": "Описание второй задачи",
5   "status": "IN_PROGRESS",
6   "createdAt": "2025-01-04T21:07:52.557658"
7 }
```

5. Обновление задачи (PUT)

The screenshot displays an HTTP client interface with a dark theme. At the top, the breadcrumb 'prog /' is followed by 'New Request'. On the right, there are 'Save' and 'Share' buttons. The main area shows a 'PUT' method selected from a dropdown, with the URL 'http://localhost:8080/api/tasks/1' entered. A 'Send' button is to the right. Below the URL bar, tabs for 'Params', 'Auth', 'Headers (9)', 'Body', 'Scripts', and 'Settings' are visible, with 'Body' being the active tab. Under the 'Body' tab, 'raw' is selected, and 'JSON' is chosen from a dropdown. The request body is a JSON object:

```
{
  "title": "Обновленная первая задача",
  "description": "Новое описание первой задачи",
  "status": "COMPLETED"
}
```

. The bottom section shows the response status '200 OK' with a green background, and details: '96 ms', '628 B', and a globe icon. There are also 'Save Response' and three dots. Below this, tabs for 'Pretty', 'Raw', 'Preview', and 'Visualize' are shown, with 'Pretty' selected. The response body is a JSON object:

```
{
  "id": 1,
  "title": "Обновленная первая задача",
  "description": "Новое описание первой задачи",
  "status": "COMPLETED",
  "createdAt": "2025-01-04T21:07:00.221665"
}
```

prog / New Request

PUT Send

Params Auth Headers (9) **Body** Scripts Settings Cookies

raw JSON Beautify

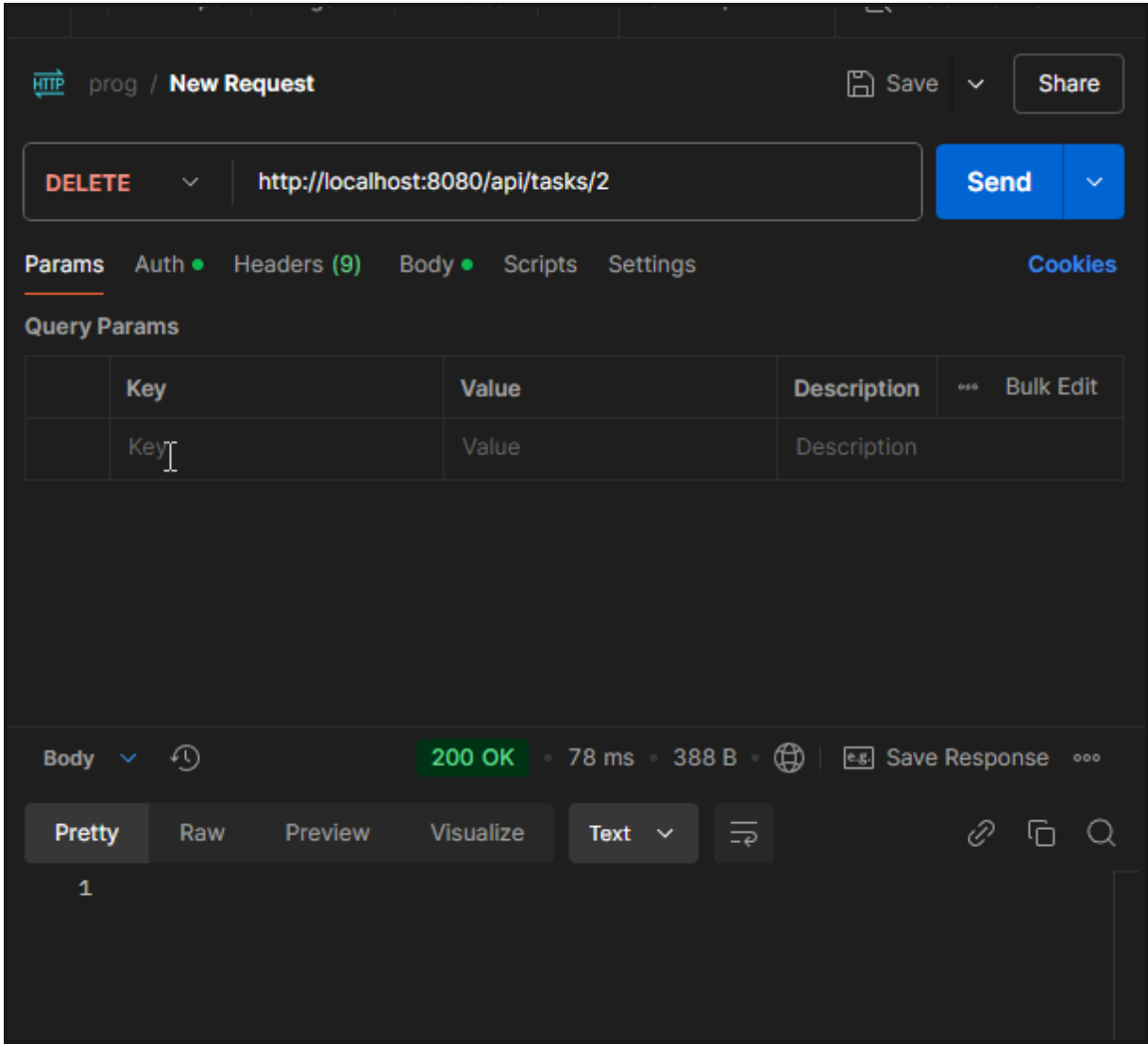
```
1 {
2   "title": "Обновленная первая задача",
3   "description": "Новое описание первой задачи",
4   "status": "COMPLETED"
5 }
```

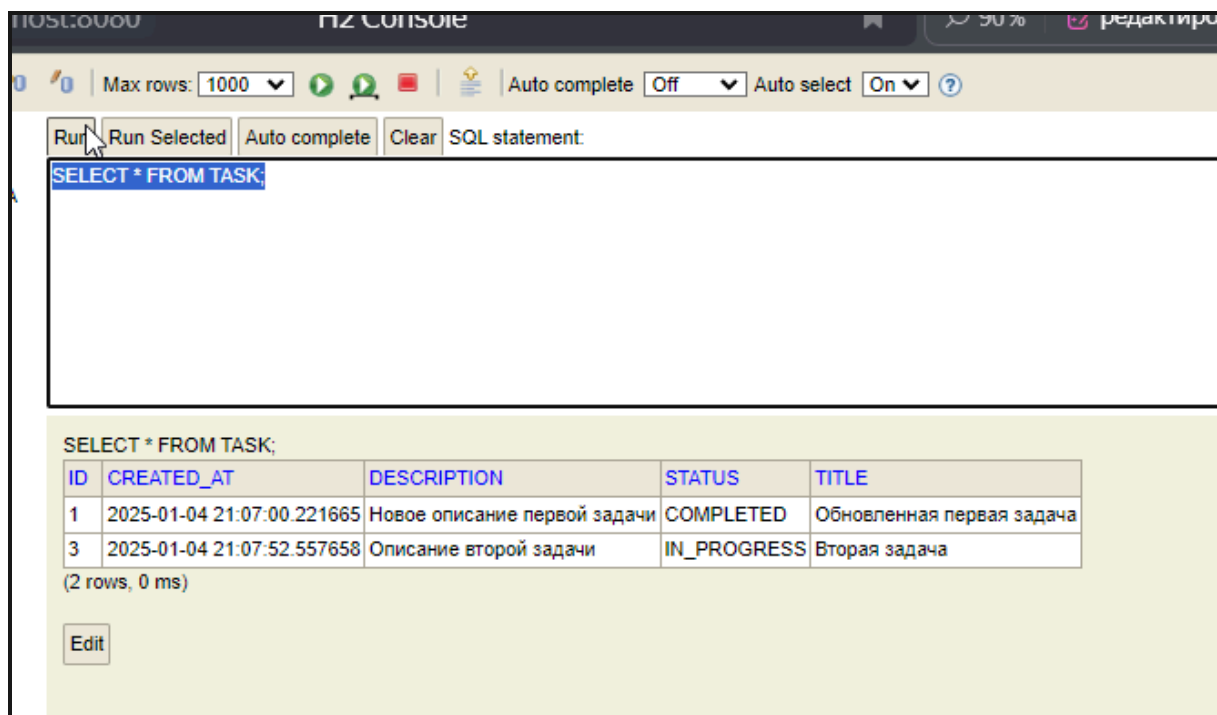
Body 200 OK • 96 ms • 628 B • Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "title": "Обновленная первая задача",
4   "description": "Новое описание первой задачи",
5   "status": "COMPLETED",
6   "createdAt": "2025-01-04T21:07:00.221665"
7 }
```


6. Удаление задачи (DELETE)





GET /api/tasks - Get all tasks
GET /api/tasks/{id} - Get task by ID
POST /api/tasks - Create new task

PUT /api/tasks/{id} - Update task
DELETE /api/tasks/{id} - Delete task