OTH-Regensburg
Übungen zur Vorlesung
Softwareentwicklung

Übung Nr. 8
Restful APIs mit RestController, RestTemplate und Hateoas

Aufgabe 1 – Erstellen eines neuen Projekts und der Abhängigkeiten

• Importieren Sie die folgenden Abhängigkeiten, um Spring Security,Hateoas und JPA zu verwenden:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.hateoas</groupId>
   <artifactId>spring-hateoas</artifactId>
</dependency>
<dependency>
   <groupId>com.h2database</groupId>
   <artifactId>h2</artifactId>
   <scope>runtime</scope>
</dependency>
```

Fügen Sie der Datei application.properties diese Werte hinzu:

```
spring.main.allow-bean-definition-overriding=true



spring.messages.basename=messages
server.error.include-binding-errors=always

#DB
spring.datasource.url=jdbc:h2:mem:testdb;MODE=MySQL;NON_KEYWORDS=USER;
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=create-drop


#spring.data.jpa.repositories.bootstrap-mode=default


spring.jpa.defer-datasource-initialization=true

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.jpa.show-sql=true
#hibernate.auto_quote_keyword=true
```

Fügen Sie der Datei data.sql diese Werte hinzu:

```
INSERT INTO workshop (description) VALUES ( 'IoT');
INSERT INTO workshop (description) VALUES ( 'ChatGPT');
INSERT INTO workshop (description) VALUES ( 'Scientific Writing');
```

Aufgabe 2- Erstellen Sie die Workshop-Entität, RestController, Service und Repository:

```
@Entity
public class Workshop {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String description;
   // getters and setters
}
```

```
public interface WorkshopRepository extends JpaRepository<Workshop, Long> {
}
```

```
@Service
public class WorkshopService {
   @Autowired
   private WorkshopRepository workshopRepository;

   public List<Workshop> getAllWorkshops() {
      return workshopRepository.findAll();
   }

   public Workshop getWorkshopById(Long id) {
      return workshopRepository.findById(id).orElse(null);
   }

   public Workshop createWorkshop(Workshop workshop) {
      return workshopRepository.save(workshop);
   }

   public Workshop updateWorkshop(Workshop workshop) {
      return workshopRepository.save(workshop);
   }

   public void deleteWorkshop(Long id) {
      workshopRepository.deleteById(id);
   }
}
```

```
@RestController
@RequestMapping("/api/workshops")
public class WorkshopController {
   @Autowired
   private WorkshopService workshopService;

   @GetMapping
   public ResponseEntity<CollectionModel<EntityModel<Workshop>>> getAllWorkshops() {
      List<Workshop> workshops = workshopService.getAllWorkshops();
      List<EntityModel<Workshop>> workshopModels = workshops.stream()
            .map(workshop -> EntityModel.of(workshop,
                  linkTo(methodOn(WorkshopController.class).getWorkshopById(workshop.getId())).withSelfRel(),
                  linkTo(methodOn(WorkshopController.class).getAllWorkshops()).withRel("workshops")))
            ).collect(Collectors.toList());
      return ResponseEntity.ok(CollectionModel.of(workshopModels));
   }

@GetMapping("/{id}")
public ResponseEntity<EntityModel<Workshop>> getWorkshopById(@PathVariable Long id) {
   Workshop workshop = workshopService.getWorkshopById(id);

   if (workshop == null) {
      return ResponseEntity.notFound().build();
   }

   EntityModel<Workshop> entityModel = EntityModel.of(workshop,
         linkTo(methodOn(WorkshopController.class).getWorkshopById(id)).withSelfRel(),
         linkTo(methodOn(WorkshopController.class).getAllWorkshops()).withRel("workshops"));

   return ResponseEntity.ok(entityModel);
}

   // Implement other endpoints for POST, PUT, and DELETE operations,
   // using similar HATEOAS principles to provide links to related resources.
}
```

Aufgabe 2 –Testen Sie die API mit Postman

- **.GET** **/api/workshops** **retrieves all Workshops**

- Legen Sie die HTTP-Methode fest: Wählen Sie „GET“ aus dem Dropdown-Menü.
- Legen Sie die Anforderungs-URL fest: Geben Sie die URL Ihres API-Endpunkts ein, z. B. http://localhost:8080/api/workshops.
- Senden Sie die Anfrage: Klicken Sie auf die Schaltfläche „Senden“, um die Anfrage auszuführen.
- Überprüfen Sie die Antwort:
- Statuscode: Überprüfen Sie den HTTP-Statuscode im Antwortheader. Ein 200 OK zeigt eine erfolgreiche Anfrage an.
- Antworttext: Untersuchen Sie den JSON-Antworttext, um die Liste der Workshops und ihrer Eigenschaften zu überprüfen.
  - HATEOAS-Links: Überprüfen Sie die Eigenschaft _links in der Antwort, um die Selbst- und Sammlungslinks anzuzeigen.

Aufgabe 3 –Testen Sie die API mit Postman

- **.GET** **/api/workshops/id** **retrieve an specific Workshop**

- Legen Sie die Anforderungs-URL fest: Geben Sie die URL Ihres API-Endpunkts ein, z. B. http://localhost:8080/api/workshops/1. Ersetzen Sie 1 durch die aktuelle ID der Werkstatt, die Sie abrufen möchten.
- Senden Sie die Anfrage: Klicken Sie auf die Schaltfläche „Senden“, um die Anfrage auszuführen.
- Überprüfen Sie die Antwort:
  - Statuscode: Überprüfen Sie den HTTP-Statuscode. Ein 200 OK weist auf eine erfolgreiche Anfrage hin, während ein 404 Not Found darauf hinweist, dass die Werkstatt nicht gefunden wurde.
  - Antworttext: Untersuchen Sie den JSON-Antworttext, um die Details des abgerufenen Workshops zu überprüfen.
  - HATEOAS-Links: Überprüfen Sie die Eigenschaft _links in der Antwort, um die Selbst- und Sammlungslinks anzuzeigen.

Andere Endpunkte für POST-, PUT- und DELETE-Vorgänge sind (optional) zu implementieren.

Aufgabe 4 - Verwenden einer externen API

- Erstellen Sie ein neues Spring Boot-Projekt.
- Fügen Sie die erforderlichen Abhängigkeiten zu Ihrer pom.xml hinzu.

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

- Erstellen Sie die Post-Klasse:

```
public class Post {
   private long userId;
   private long id;
   private String title;
   private String body;

   // getters and setters
}
```

- Erstellen Sie den PostClient:

```
@Service
public class PostClient {
   @Autowired
   private RestTemplate restTemplate;

   public List<Post> getPosts() {
      String apiUrl = "https://jsonplaceholder.typicode.com/posts";
      ResponseEntity<List<Post>> response = restTemplate.exchange(
         apiUrl,
         HttpMethod.GET,
         null,
         new ParameterizedTypeReference<List<Post>>() {}
      );
      // This was in the original exercise sheet, but causes problems with Jackson deserialization
      // ResponseEntity<List<Post>> response = restTemplate.getForEntity(apiUrl, List.class);

      if (response.getStatusCode() == HttpStatus.OK) {
         return response.getBody();
      } else {
         throw new RuntimeException("Failed to fetch posts: " + response.getStatusCode());
      }
   }
}
```

- Erstellen Sie einen Controller, um die Daten verfügbar zu machen:

```
@RestController
@RequestMapping("/api/posts")
public class PostController {
   @Autowired
   private PostClient postClient;

   @GetMapping
   public List<Post> getPosts() {
      return postClient.getPosts();
   }
}
```

Aufgabe 5 – Testen Sie die Anwendung

- Starten Sie Ihre Spring Boot-Anwendung.
- Verwenden Sie Tools wie Postman oder curl, um eine GET-Anfrage an Ihren API-Endpunkt zu senden, z. B. http://localhost:8080/api/posts.
- Überprüfen Sie die JSON-Antwort, um sicherzustellen, dass die Liste der Beiträge korrekt abgerufen wird.