Programação para Internet

Bacharelado em Ciência da Computação – FACOM – UFU

Prof. Flávio de Oliveira Silva, Ph.D.

Este laboratório é uma sequência do Laboratório 05. O objetivo deste laboratório é criar um serviço **REST** utilizando o Spring Boot e funcionalidades associadas (Spring Validation, Spring Data JPA, Spring H2 Database). O "controller" deste projeto será responsável por expor um conjunto de operações baseadas nos princípios do REST. Este laboratório utilizar persistência em memória baseada no banco de dados H2.

1. Este roteiro leva em conta que o laboratório 05 foi concluído e que, portanto, o ambiente de desenvolvimento está criado.
2. Adicione os componentes do Spring necessários como indicado na figura baixo. Selecionar: Spring Boot Dev Tools; Validation; Spring Boot Actuator; Spring Data JPA; H2 Database e Spring Web.

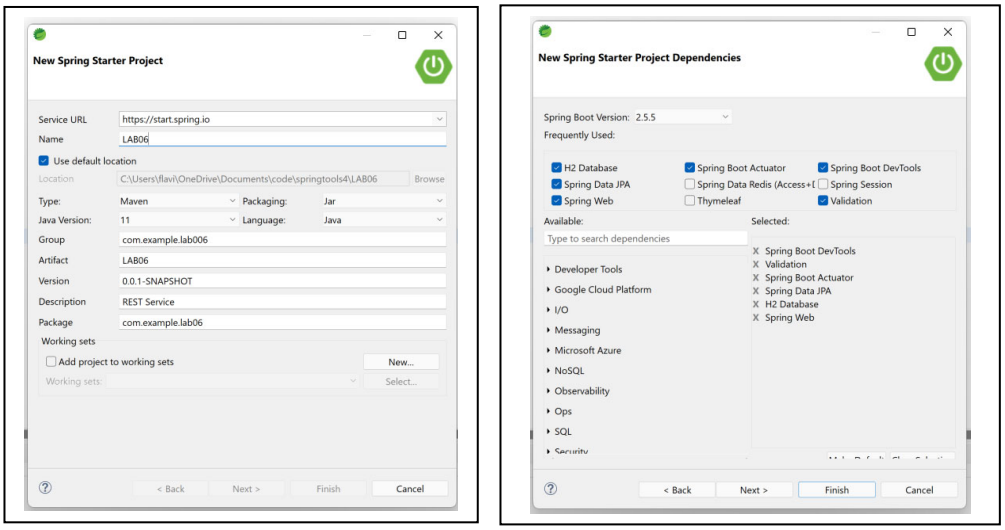

Figura 1 - Configuração do Spring Starter

3. Crie uma classe que representa um conceito do domínio da aplicação, neste caso será a classe **User.** Esta classe contém anotações para indicar tanto que será persistida (**@entity**) e que utiliza as restrições de validação anotadas em seus atributos. O código completo da classe **User** está indicado na Figura 2.1
4. A persistência será feita utilizando a classe JpaRepository do Spring Data JPA. Para isto é necessário JpaRepository e indicar que a entidade é um objeto da classe User. Crie uma interface chamada **UserRepository** e utilize o código mostrado na Figura 3.
5. Crie uma classe auxiliar que será responsável por carregar em memória, no banco de dados H2 alguns usuários iniciais. Esta classe será chamada **UserLoadDatabase** e seu código está indicado na Figura 4.
6. Este serviço será uma aplicação baseada em REST apenas pela maneira de criar os métodos no "Controller". O Código completo da classe **UserRESTController** está indicado na Figura 5.
   Esta classe contém uma anotação @RestController e faz um mapeamento dos métodos HTTP GET, POST e DELETE que recebe e devolve respostas utilizando a notação JSON.
7. Utilizando o CURL faça um teste do GET (all e one) e anote as repostas. Neste caso o CURL será executado na mesma máquina (localhost) onde o serviço está sendo executado.

```
curl -v localhost:8080/api/user
curl -v localhost:8080/api/user/1
```

```java
package com.example.lab06;
import javax.persistence.Entity;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
@Entity
public class User {
   private @Id @GeneratedValue Long id;
   @NotNull
   @Size(min=3, max=30)
   private String name;
   @Email
   private String email;

   public User() {};

   public User(String name, String email) {
      this.name = name;
      this.email = email;
   };

   public Long getId() {
      return id;
   }

   public void setId(Long id) {
      this.id = id;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getEmail() {
      return email;
   }
   public void setEmail(String email) {
      this.email = email;
   }
   @Override
   public String toString() {
      return "User [id=" + id + ", name=" + name + ", email=" + email + ", toString()=" + super.toString() + "]";
   }


}
```

Figura 2 - Entidade User

```java
package com.example.lab06;

import org.springframework.data.jpa.repository.JpaRepository;
interface UserRepository extends JpaRepository<User, Long> {
}
```

Figura 3 - Interface UserRepository

```java
package com.example.lab06;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
class UserLoadDatabase {
  private static final Logger log = LoggerFactory.getLogger(UserLoadDatabase.class);

  @Bean
  CommandLineRunner initDatabase(UserRepository repository) {
    return args -> {
      log.info("Preloading " + repository.save(new User("Steve Jobs", "steve@apple.com")));
      log.info("Preloading " + repository.save(new User("Bill gates", "bill@ms.com")));
      log.info("Preloading " + repository.save(new User("Elon Musk", "elon@tesla.com")));
    };
  }
}
```

Figura 4 - Classe UserLoadDatabase

```java
package com.example.lab06;

import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/user")
public class UserRESTController {

        @Autowired
    private UserRepository useRepository;

    @GetMapping
    public List<User> findAllUsers() {
        // Implement
         return useRepository.findAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> findUserById(@PathVariable(value = "id") long id) {
       // Implement
         Optional<User> user = useRepository.findById(id);

        if(user.isPresent()) {
            return ResponseEntity.ok().body(user.get());
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    @PostMapping
    public User saveEmployee(@Validated @RequestBody User user) {
        // Implement
         return useRepository.save(user);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<String> deleteEmployeeById(@PathVariable(value = "id") long id) {
       // Implement
         Optional<User> user = useRepository.findById(id);

        if(user.isPresent()) {
         useRepository.deleteById(id);
            return ResponseEntity.ok().body("Deleted");
        } else {
            return ResponseEntity.notFound().build();
        }
    }
}
```

Figura 5 - UserRESTController

8. Utilizando o CURL faça um teste do POST. Adicione um novo usuário e faça depois um GET. Anote as respostas.
   `curl --location --request POST 'http://localhost:8080/api/user' --header 'Content-Type: application/json' --data-raw '{"name": "Larry Page", "email": "larry@oracle.com" }'`
9. Utilizando o CURL faça um teste do DELETE
   `curl --request DELETE http://localhost:8080/api/user/4`
10. Realize os mesmos testes utilizando o Postman. (https://www.postman.com/downloads/)
11. Proponha uma nova classe que represente um conceito do seu domínio de aplicação. Vamos considerar que o nome desta classe é: **CDomainClass**. Como feito no **passo 3**, crie a definição desta classe no pacote **com.example.lab06.** Nesta classe utilize as anotações para entidade (**@entity**). É necessário um atributo para a chave primária (**Long id**). Além deste, fique livre para definir os atributos desta classe da maneira mais conveniente. Adicione as validações que entender que são adequadas.
12. Para permitir a persistência, crie uma interface chamada **CDomainClassRepository** e a mesma abordagem mostrada no Passo 4.

13. Utilizando o mesmo conceito apresentado no Passo 5, crie uma classe **CDomainClassLoadDatabase** que será responsável por carregar na memória três objetos da Classe **CDomainClass**
14. Como no Passo 6, crie um controlador baseado nos princípios REST que será responsável por tratar os métodos GET, POST e DELETE. Utilize o código do **UserRESTController como base. Esta classe será chamada CDomainClassRESTController.**
15. Adicione um método responsável pela edição (PUT) de um objeto da classe **CDomainClass.** Utilize a anotação **@PutMapping**
16. Utilizando o CURL ou o Postman faça o teste de todos os métodos GET (all e one), POST, DELETE, PUT. Anote os comandos utilizando ou o texto RAW(Postman) tanto do **Request** quando da **Response**.

17. Exporte o projeto para um arquivo .zip e prepare um documento com todas as respostas solicitadas. Para exportar utilize a opção: **File → Export ... → General - Archive File**

18. Faça o upload deste arquivo **.ZIP** para uma pasta no **OneDrive** ou **GoogleDrive**.

19. Envie sua resposta na Atividade do TEAMS – **LAB06-SpringBoot-REST** e na resposta coloque o link da pasta compartilhada que foi criada e que contém o resultado das atividades práticas. A tarefa deve ser encerrada até o dia 20/01/2023 (próxima sexta-feira) conforme indicado no MS TEAMS.