Apresentação Contabilidade & Jovem Tech

Luis Felipe Bispo Silva

Data: 21/07/2024

Documentação da Aplicação Java SeguroUniMed

Tecnologias Utilizadas

1. JDK 11
2. Maven 3

Estrutura do Projeto

A aplicação é estruturada da seguinte maneira:

- com.example.api: Pacote principal que contém a classe ApiApplication.
- com.example.api.domain: Pacote que contém as entidades de domínio, como Customer.
- com.example.api.model: Pacote que contém outras classes de modelo, como Endereco e EnderecoViaCep.
- com.example.api.repository: Pacote que contém as interfaces de repositório para acesso aos dados, como CustomerRepository e EnderecoRepository.
- com.example.api.service: Pacote que contém as classes de serviço, como CustomerService e AddressService.
- com.example.api.web.rest: Pacote que contém os controladores REST, como CustomerController.
- com.example.api.config: Pacote que contém as classes de configuração, como WebConfig.

Endpoints

Listar Todos os Clientes

- Endpoint: /customers
- Método: GET
- Descrição: Retorna a lista de todos os clientes cadastrados.

Filtrar Clientes

Buscar por nome

- Endpoint: /customers/name/{name}
- Método: GET
- Descrição: Retorna o cliente com o nome especificado.

Buscar por email

- Endpoint: /customers/email/{email}
- Método: GET
- Descrição: Retorna o cliente com o email especificado.

Buscar por gênero

- Endpoint: /customers/gender/{gender}
- Método: GET
- Descrição: Retorna os clientes com o gênero especificado.

Paginação de Clientes

- Endpoint: /customers/paged
- Método: GET
- Descrição: Retorna uma página de clientes.
- Parâmetros:
    - page: Número da página (default = 0).
    - size: Tamanho da página (default = 10).

Criar Novo Cliente

- Endpoint: /customers
- Método: POST
- Descrição: Cria um novo cliente.
- Validações:
    - O nome é obrigatório.
    - O email é obrigatório e deve ser um email válido.
- Exemplo de Requisição:

```
{
  "name": " Homem Aranha ",
  "email": " aranha@vingadores.com ",
  "gender": "M",
```

```
  "enderecos": [
    {
      "cep": "01001-000",
      "logradouro": "Praça da Sé",
      "bairro": "Sé",
      "localidade": "São Paulo",
      "estado": "SP"
    }
  ]
}
```

Editar Cliente

- Endpoint: /customers/{id}
- Método: PUT
- Descrição: Edita as informações de um cliente existente.
- Validações:
  - O nome é obrigatório.
  - O email é obrigatório e deve ser um email válido.
- Exemplo de Requisição:

```
{
  "name": " Homem Aranha ",
  "email": " aranha@vingadores.com ",
  "gender": "M",
  "enderecos": [
    {
      "cep": "02001-000",
      "logradouro": "Avenida Paulista",
      "bairro": "Bela Vista",
      "localidade": "São Paulo",
      "estado": "SP"
    }
  ]
}
```

Excluir Cliente

- Endpoint: /customers/{id}
- Método: DELETE
- Descrição: Exclui um cliente pelo ID.

Validação dos Dados

Customer

- name: Não pode ser vazio.
- email: Não pode ser vazio e deve ser um email válido.

Cadastro de Múltiplos Endereços para um Cliente

Ao cadastrar um cliente, é possível adicionar uma lista de endereços ao cliente. Cada endereço é associado ao cliente utilizando o campo customer_id.

Consumo de API Externa para Cadastro de Endereços

Ao cadastrar ou editar um endereço, é possível inserir apenas o CEP. Os dados do endereço são carregados via consumo do serviço https://viacep.com.br/.

Filtragem de Clientes por Cidade e Estado

Buscar por cidade

- Endpoint: /customers/cidade/{cidade}
- Método: GET
- Descrição: Retorna os clientes que possuem endereços na cidade especificada.

Buscar por estado

- Endpoint: /customers/estado/{estado}
- Método: GET
- Descrição: Retorna os clientes que possuem endereços no estado especificado.

Configuração CORS

A configuração CORS está definida na classe WebConfig para permitir o acesso de origens específicas.

Descrição Detalhada das Classes

ApiApplication

A classe ApiApplication é a classe principal da aplicação Spring Boot. Ela é anotada com @SpringBootApplication, o que é uma combinação das anotações @Configuration, @EnableAutoConfiguration, e @ComponentScan. O método main inicializa a aplicação usando SpringApplication.run(ApiApplication.class, args).

```
package com.example.api;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ApiApplication {
   public static void main(String[] args) {
      SpringApplication.run(ApiApplication.class, args);
   }
}
```

Customer

A classe Customer é uma entidade JPA que representa um cliente. Ela é anotada com @Entity e @Table(name = "customers"). Os campos da classe são mapeados para colunas da tabela customers usando anotações como @Id, @GeneratedValue, @Column, etc.

```
package com.example.api.domain;

import javax.persistence.*;
import java.util.List;

@Entity
@Table(name = "customers")
public class Customer {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @Column(nullable = false)
   private String name;

   @Column(nullable = false, unique = true)
   private String email;

   @Column
   private String gender;

   @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL, orphanRemoval =
true)
   private List<Endereco> enderecos;

   // Getters e setters
```

```
}
```

Endereco

A classe Endereco é uma entidade JPA que representa um endereço. Ela é anotada com @Entity e @Table(name = "enderecos"). Os campos da classe são mapeados para colunas da tabela enderecos. Ela tem um relacionamento ManyToOne com a entidade Customer.

```
package com.example.api.model;

import javax.persistence.*;

@Entity
@Table(name = "enderecos")
public class Endereco {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String cep;

    @Column(nullable = false)
    private String logradouro;

    @Column(nullable = false)
    private String bairro;

    @Column(nullable = false)
    private String localidade;

    @Column(nullable = false)
    private String estado;

    @ManyToOne
    @JoinColumn(name = "customer_id", nullable = false)
    private Customer customer;

    // Getters e setters
}
```

EnderecoViaCep

A classe EnderecoViaCep é usada para mapear a resposta da API ViaCEP. Não é uma entidade JPA, mas sim uma classe de modelo que facilita a integração com a API externa.

```
package com.example.api.model;

public class EnderecoViaCep {
    private String cep;
    private String logradouro;
    private String bairro;
    private String localidade;
    private String uf;

    // Getters e setters
}
```

CustomerRepository

A interface CustomerRepository estende JpaRepository e define métodos para operações CRUD e consultas específicas para a entidade Customer.

```
package com.example.api.repository;

import com.example.api.domain.Customer;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface CustomerRepository extends JpaRepository<Customer, Long> {
    List<Customer> findByName(String name);
    Customer findByEmail(String email);
    List<Customer> findByGender(String gender);
}
```

EnderecoRepository

A interface EnderecoRepository estende JpaRepository e define métodos para operações CRUD e consultas específicas para a entidade Endereco.

```
package com.example.api.repository;

import com.example.api.model.Endereco;
import org.springframework.data.jpa.repository.JpaRepository;

public interface EnderecoRepository extends JpaRepository<Endereco, Long> {
}
```

CustomerService

A classe CustomerService contém a lógica de negócios relacionada aos clientes. Ela usa CustomerRepository para acessar os dados e inclui métodos para criar, editar, excluir e buscar clientes.

```
package com.example.api.service;

import com.example.api.domain.Customer;
import com.example.api.model.Endereco;
import com.example.api.repository.CustomerRepository;
import com.example.api.repository.EnderecoRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class CustomerService {
    @Autowired
    private CustomerRepository customerRepository;

    @Autowired
    private EnderecoRepository enderecoRepository;

    public List<Customer> findAll() {
        return customerRepository.findAll();
    }

    public Optional<Customer> findById(Long id) {
        return customerRepository.findById(id);
    }
```

```
    public Customer save(Customer customer) {
        return customerRepository.save(customer);
    }

    public void deleteById(Long id) {
        customerRepository.deleteById(id);
    }

    public List<Customer> findByName(String name) {
        return customerRepository.findByName(name);
    }

    public Customer findByEmail(String email) {
        return customerRepository.findByEmail(email);
    }

    public List<Customer> findByGender(String gender) {
        return customerRepository.findByGender(gender);
    }
}
```

AddressService

A classe AddressService contém a lógica de negócios relacionada aos endereços. Ela usa EnderecoRepository e a API ViaCEP para buscar informações de endereço.

```
package com.example.api.service;

import com.example.api.model.Endereco;
import com.example.api.model.EnderecoViaCep;
import com.example.api.repository.EnderecoRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import java.util.Optional;

@Service
public class AddressService {
    @Autowired
    private EnderecoRepository enderecoRepository;
```

```
    public Optional<Endereco> findById(Long id) {
        return enderecoRepository.findById(id);
    }

    public Endereco save(Endereco endereco) {
        return enderecoRepository.save(endereco);
    }

    public void deleteById(Long id) {
        enderecoRepository.deleteById(id);
    }

    public EnderecoViaCep findAddressByCep(String cep) {
        RestTemplate restTemplate = new RestTemplate();
        String url = "https://viacep.com.br/ws/" + cep + "/json/";
        return restTemplate.getForObject(url, EnderecoViaCep.class);
    }
}
```

CustomerController

A classe CustomerController é o controlador REST para a entidade Customer. Ela define os endpoints da API e utiliza CustomerService para executar a lógica de negócios.

```
package com.example.api.web.rest;

import com.example.api.domain.Customer;
import com.example.api.service.CustomerService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/customers")
public class CustomerController {
    @Autowired
    private CustomerService customerService;
```

```
    @GetMapping
    public List<Customer> getAllCustomers() {
        return customerService.findAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Customer> getCustomerById(@PathVariable Long id) {
        Optional<Customer> customer = customerService.findById(id);
        return customer.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
    }

    @PostMapping
    public Customer createCustomer(@RequestBody Customer customer) {
        return customerService.save(customer);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Customer> updateCustomer(@PathVariable Long id,
@RequestBody Customer customerDetails) {
        Optional<Customer> customer = customerService.findById(id);
        if (customer.isPresent()) {
            Customer updatedCustomer = customer.get();
            updatedCustomer.setName(customerDetails.getName());
            updatedCustomer.setEmail(customerDetails.getEmail());
            updatedCustomer.setGender(customerDetails.getGender());
            updatedCustomer.setEnderecos(customerDetails.getEnderecos());
            return ResponseEntity.ok(customerService.save(updatedCustomer));
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteCustomer(@PathVariable Long id) {
        if (customerService.findById(id).isPresent()) {
            customerService.deleteById(id);
            return ResponseEntity.ok().build();
        } else {
            return ResponseEntity.notFound().build();
        }
    }
}
```

Integração e Configuração

As classes se comunicam usando a injeção de dependência do Spring. Por exemplo, CustomerController injeta CustomerService, que por sua vez injeta CustomerRepository. Isso promove um design desacoplado e facilita a manutenção e os testes.

A configuração CORS na classe WebConfig permite que a aplicação seja acessada de origens específicas, configurando os métodos HTTP permitidos, os cabeçalhos e outras opções.

```
package com.example.api.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {
   @Override
   public void addCorsMappings(CorsRegistry registry) {
      registry.addMapping("/**")
            .allowedOrigins("http://127.0.0.1:5500")
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowedHeaders("*")
            .allowCredentials(true)
            .maxAge(3600);
   }
}
```