RELATÓRIO DE REFATORAÇÃO – ClienteResource.java

ARQUITETURA DE SOFTWARE

- Bruno Henrique Gouveia da Silva
- Emanuel Vitor de Holanda Feitosa
- João Vitor Silva Rosa
- Vinicius Tenório Lira
- Vitor Geovani Melo Silva

CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

São Paulo

2025

1. Classe Original

A classe ClienteResource original implementa um controlador REST básico com operações CRUD:

package com.sistema.resources;

```
import com.sistema.entities.Cliente;
import com.sistema.services.ClienteService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import javax.validation.Valid;
import java.net.URI;
import java.util.List;

@RestController
@RequestMapping("/api/clientes")
public class ClienteResource {

  @Autowired
  private ClienteService clienteService;

  @GetMapping
  public ResponseEntity<List<Cliente>> findAll() {
    List<Cliente> clientes = clienteService.findAll();
    return ResponseEntity.ok(clientes);
  }
```

```
  @GetMapping("/{id}")
  public ResponseEntity<Cliente> findById(@PathVariable Long id) {
    Cliente cliente = clienteService.findById(id);
    return ResponseEntity.ok(cliente);
  }

  @PostMapping
  public ResponseEntity<Cliente> create(@Valid @RequestBody Cliente cliente) {
    cliente = clienteService.save(cliente);
    URI uri = ServletUriComponentsBuilder.fromCurrentRequest()
          .path("/{id}")
          .buildAndExpand(cliente.getId())
          .toUri();
    return ResponseEntity.created(uri).body(cliente);
  }

  @PutMapping("/{id}")
  public ResponseEntity<Cliente> update(@PathVariable Long id, @Valid
@RequestBody Cliente cliente) {
    cliente = clienteService.update(id, cliente);
    return ResponseEntity.ok(cliente);
  }

  @DeleteMapping("/{id}")
  public ResponseEntity<Void> delete(@PathVariable Long id) {
    clienteService.delete(id);
    return ResponseEntity.noContent().build();
```

```
  }
}
```

Problemas identificados:

- Responsabilidade única violada (controla HTTP + lógica de negócio)
- Acoplamento direto com ClienteService
- Dificuldade para testes unitários
- Falta de flexibilidade para diferentes estratégias de operações

2. Design Pattern Escolhido: Strategy

O **Strategy Pattern** foi escolhido para separar as diferentes estratégias de operações CRUD, permitindo:

- Melhor organização do código
- Facilidade para adicionar novas operações
- Testabilidade aprimorada
- Redução do acoplamento

3. Implementação Refatorada

Interface Strategy

```
package com.sistema.strategies;

import org.springframework.http.ResponseEntity;

public interface ClienteOperationStrategy<T> {
  ResponseEntity<T> execute();
}
```

Estratégias Concretas

```
package com.sistema.strategies.impl;
```

```
import com.sistema.entities.Cliente;
import com.sistema.services.ClienteService;
import com.sistema.strategies.ClienteOperationStrategy;
import org.springframework.http.ResponseEntity;
import java.util.List;

public class FindAllClientsStrategy implements
ClienteOperationStrategy<List<Cliente>> {
  private final ClienteService clienteService;

  public FindAllClientsStrategy(ClienteService clienteService) {
    this.clienteService = clienteService;
  }

  @Override
  public ResponseEntity<List<Cliente>> execute() {
    List<Cliente> clientes = clienteService.findAll();
    return ResponseEntity.ok(clientes);
  }
}

package com.sistema.strategies.impl;

import com.sistema.entities.Cliente;
import com.sistema.services.ClienteService;
import com.sistema.strategies.ClienteOperationStrategy;
import org.springframework.http.ResponseEntity;
```

```
public class FindClientByIdStrategy implements
ClienteOperationStrategy<Cliente> {
   private final ClienteService clienteService;
   private final Long id;

   public FindClientByIdStrategy(ClienteService clienteService, Long id) {
      this.clienteService = clienteService;
      this.id = id;
   }

   @Override
   public ResponseEntity<Cliente> execute() {
      Cliente cliente = clienteService.findById(id);
      return ResponseEntity.ok(cliente);
   }
}


package com.sistema.strategies.impl;

import com.sistema.entities.Cliente;
import com.sistema.services.ClienteService;
import com.sistema.strategies.ClienteOperationStrategy;
import org.springframework.http.ResponseEntity;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;
import java.net.URI;

public class CreateClientStrategy implements ClienteOperationStrategy<Cliente>
{
   private final ClienteService clienteService;
```

```
    private final Cliente cliente;

    public CreateClientStrategy(ClienteService clienteService, Cliente cliente) {
        this.clienteService = clienteService;
        this.cliente = cliente;
    }

    @Override
    public ResponseEntity<Cliente> execute() {
        Cliente savedCliente = clienteService.save(cliente);
        URI uri = ServletUriComponentsBuilder.fromCurrentRequest()
                .path("/{id}")
                .buildAndExpand(savedCliente.getId())
                .toUri();
        return ResponseEntity.created(uri).body(savedCliente);
    }
}

package com.sistema.strategies.impl;

import com.sistema.entities.Cliente;
import com.sistema.services.ClienteService;
import com.sistema.strategies.ClienteOperationStrategy;
import org.springframework.http.ResponseEntity;

public class UpdateClientStrategy implements ClienteOperationStrategy<Cliente>
{
    private final ClienteService clienteService;
```

```
    private final Long id;
    private final Cliente cliente;

    public UpdateClientStrategy(ClienteService clienteService, Long id, Cliente
cliente) {
        this.clienteService = clienteService;
        this.id = id;
        this.cliente = cliente;
    }

    @Override
    public ResponseEntity<Cliente> execute() {
        Cliente updatedCliente = clienteService.update(id, cliente);
        return ResponseEntity.ok(updatedCliente);
    }
}

package com.sistema.strategies.impl;

import com.sistema.services.ClienteService;
import com.sistema.strategies.ClienteOperationStrategy;
import org.springframework.http.ResponseEntity;

public class DeleteClientStrategy implements ClienteOperationStrategy<Void> {
    private final ClienteService clienteService;
    private final Long id;

    public DeleteClientStrategy(ClienteService clienteService, Long id) {
```

```
        this.clienteService = clienteService;
        this.id = id;
    }

    @Override
    public ResponseEntity<Void> execute() {
        clienteService.delete(id);
        return ResponseEntity.noContent().build();
    }
}
```

4. Classe Resource Refatorada

```
package com.sistema.resources;

import com.sistema.entities.Cliente;
import com.sistema.services.ClienteService;
import com.sistema.strategies.impl.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;

@RestController
@RequestMapping("/api/clientes")
public class ClienteResource {
```

```
  @Autowired
  private ClienteService clienteService;

  @GetMapping
  public ResponseEntity<List<Cliente>> findAll() {
    return new FindAllClientsStrategy(clienteService).execute();
  }

  @GetMapping("/{id}")
  public ResponseEntity<Cliente> findById(@PathVariable Long id) {
    return new FindClientByIdStrategy(clienteService, id).execute();
  }

  @PostMapping
  public ResponseEntity<Cliente> create(@Valid @RequestBody Cliente cliente) {
    return new CreateClientStrategy(clienteService, cliente).execute();
  }

  @PutMapping("/{id}")
  public ResponseEntity<Cliente> update(@PathVariable Long id, @Valid
@RequestBody Cliente cliente) {
    return new UpdateClientStrategy(clienteService, id, cliente).execute();
  }

  @DeleteMapping("/{id}")
  public ResponseEntity<Void> delete(@PathVariable Long id) {
    return new DeleteClientStrategy(clienteService, id).execute();
  }
```

```
}
```

5. Benefícios da Refatoração

✅ **Separação de Responsabilidades**: Cada strategy tem uma responsabilidade específica
✅ **Extensibilidade**: Fácil adição de novas operações
✅ **Testabilidade**: Strategies podem ser testadas isoladamente
✅ **Reutilização**: Strategies podem ser reutilizadas em outros contextos
✅ **Manutenibilidade**: Código mais organizado e legível