CRUD SpringBoot

Anotações baseadas no desafio 3 CRUD de Cliente.
03 DESAFIO CRUD de clientes.pdf

Criar Entidade

Criar classe dentro do pacote entities com os atributos solicitados.

```
// Mapeamento do JPA
@Entity
@Table(name = "tb_client") // Nome da tabela no banco de dados
public class Client {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY) // Id auto
incrementável
private Long id;
private String name;
@Column(unique = true) // Não recebe CPF duplicados
private String cpf;
private Double income;
private LocalDate birthDate;
private Integer children;

// Criar os construtores, get e set, equals e hashCode (se
necessário)
…
}
```

Criar o import do SQL

Criar imports com dados significativos que atendam as especificações da entidade.
Exemplo do desafio 3:

```
INSERT INTO tb_client (name, cpf, income, birth_date, children)
VALUES ('Thiago Freitas', '67346463005', 3000.0, '1995-07-16', 2);
INSERT INTO tb_client (name, cpf, income, birth_date, children)
VALUES ('Matheus Rodrigues', '37514747060', 1300.0, '2000-06-02',
0);
INSERT INTO tb_client (name, cpf, income, birth_date, children)
VALUES ('Rosangela Soares', '16428645017', 1400.0, '1992-11-08',
2);
INSERT INTO tb_client (name, cpf, income, birth_date, children)
VALUES ('Raquel Freitas', '62322445088', 600.0, '2021-06-04', 0);
INSERT INTO tb_client (name, cpf, income, birth_date, children)
VALUES ('Victoria Soares', '11691071099', 900.0, '2017-07-04', 0);
```

```
INSERT INTO tb_client (name, cpf, income, birth_date, children)
VALUES ('André Soares', '92831966000', 5000.0, '1990-03-22', 0);
INSERT INTO tb_client (name, cpf, income, birth_date, children)
VALUES ('Andreza Rodrigues', '76342125040', 2000.0, '1993-05-03',
1);
INSERT INTO tb_client (name, cpf, income, birth_date, children)
VALUES ('Maria Aparecida', '76293289072', 1700.0, '1975-08-12',
5);
INSERT INTO tb_client (name, cpf, income, birth_date, children)
VALUES ('Bruce Wayne', '06997935052', 7000.0, '1985-12-11', 3);
INSERT INTO tb_client (name, cpf, income, birth_date, children)
VALUES ('Tony Stark', '86270299039', 15000.0, '1975-02-27', 1);
```

Check no H2

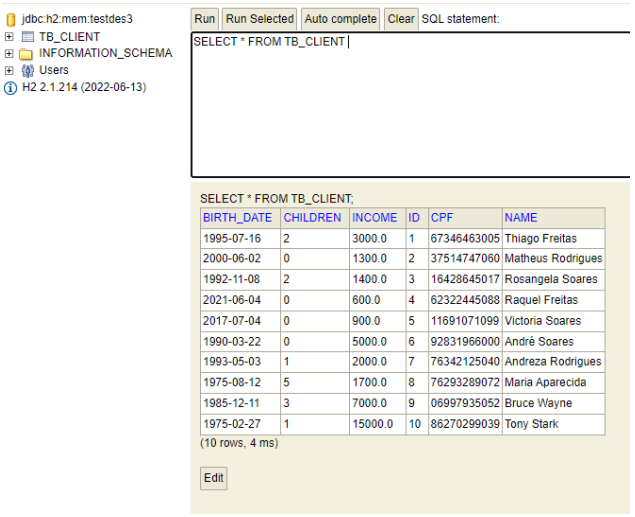
Configurar o arquivo **application.properties** com:

```
spring.profiles.active=test
spring.jpa.open-in-view=false
```

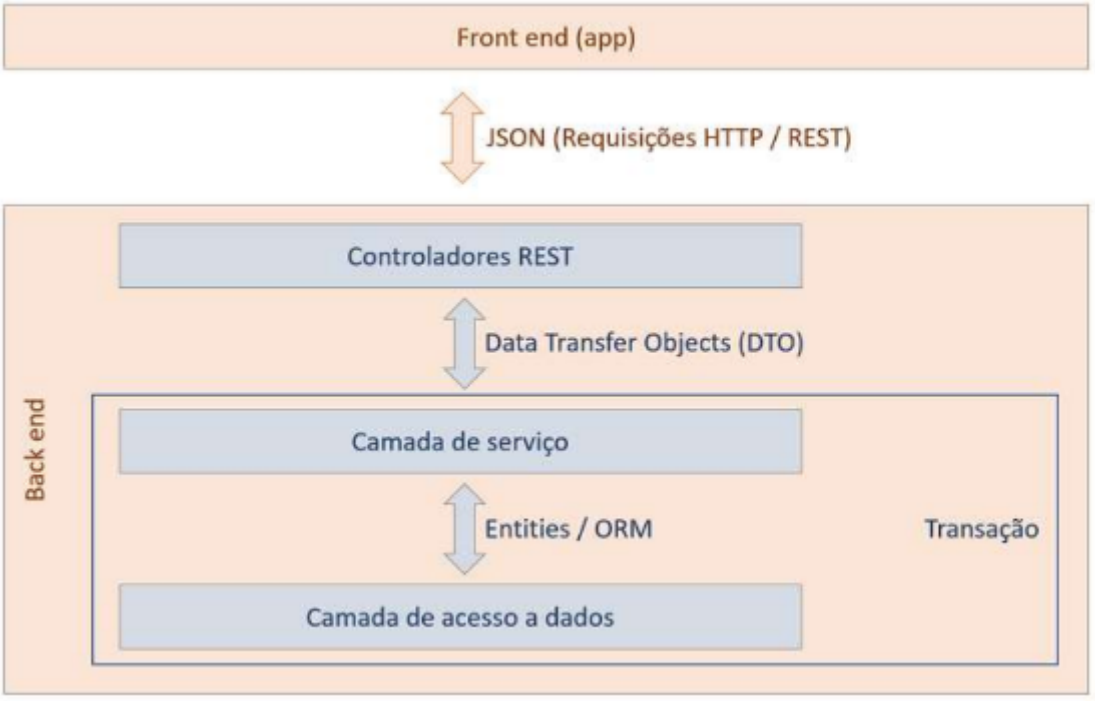
Configurar o arquivo **application-test.properties** com:

```
# Dados de conexão com o banco H2
spring.datasource.url=jdbc:h2:mem:testdes3
spring.datasource.username=sa
spring.datasource.password=

# Configuração do cliente web do banco H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

# Configuração para mostrar o SQL no console
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Após rodar o programa e obter acesso, verificar se os imports tiveram sucesso no H2.
No navegador, abrir com http://localhost:8080/h2-console e logar com os dados do **application-test.properties**



Padrão Camadas



O Controlador conversa com o Serviço por DTO que conversa com o Repository por Entidade.
DTO = Projeção de dados necessários, possuindo assim várias representações.

Repository

Criar uma interface para a **entidade específica** para realizar operações com o banco de dados.
Exemplo do desafio 3:

```
public interface ClientRepository extends JpaRepository<Client,
Long> {
```

```
}
```

DTO específico

Criar classe com o nome da entidade a ser usada + o nome DTO dentro do pacote dto. Adicionar os atributos necessários a serem exibidos. Criar construtor comum e construtor com entidade. Para DTO adicionar apenas os getters.
Exemplo do desafio 3:

```
public class ClientDTO {
private Long id;
@NotBlank(message = "Nome NÃO pode ser vazio!")
@Size(min = 3, message = "O nome precisa ter pelo menos 3
caracteres!")
private String name;
private String cpf;
private Double income;
@PastOrPresent(message = "Datas FUTURAS não são aceitas!")
private LocalDate birthDate;
private Integer children;

public ClientDTO() {
}

public ClientDTO(Long id, String name, String cpf, Double income,
LocalDate birthDate, Integer children) {
this.id = id;
this.name = name;
this.cpf = cpf;
this.income = income;
this.birthDate = birthDate;
this.children = children;
}

public ClientDTO(Client entity) {
id = entity.getId();
name = entity.getName();
cpf = entity.getCpf();
income = entity.getIncome();
birthDate = entity.getBirthDate();
children = entity.getChildren();
}
…
}
```

Service

Criar o Service dentro do pacote services. Neste, serão feitas as solicitações ao banco de dados.
Exemplo do desafio 3:

```
// Arquitetura por padrão de camadas
// Aqui é o serviço que chama o repository para realizar a busca
no banco de dados
@Service
public class ClientService {
@Autowired // Dependencia | Chamando o repository
private ClientRepository repository;
…
}
```

Controllers

Criar controlador de resposta a requisições, com o **service específico** injetado.
Exemplo do desafio 3:

```
@RestController // Responder pela web
@RequestMapping(value = "/clients") // Rota de mapeamento da Web
public class ClientController {
@Autowired // Dependencia | Chamando o serviço
private ClientService service;
…
}
```

Resposta Customizada (Postman)

Criar um objeto com o formato do erro desejado, apenas com métodos GET, na pasta dto.
Exemplo do desafio 3:

```
public class CustomError {
private Instant timestamp;
private Integer status;
private String error;
private String path;

public CustomError(Instant timestamp, Integer status, String
error, String path) {
this.timestamp = timestamp;
this.status = status;
this.error = error;
this.path = path;
}
```

Classe para Controller Advice

Pacote handlers dentro de controllers. Classe para definir tratamentos globais para exceções específicas, sem precisar ficar colocando try-catch em várias partes do código.
Exemplo do desafio 3:

```
@ControllerAdvice
public class ControllerExceptionHandler {
@ExceptionHandler(ResourceNotFoundException.class) // Interceptar
a Exceção
public ResponseEntity<CustomError>
resourceNotFoundException(ResourceNotFoundException e,
HttpServletRequest request /* Obter a Url que deu exceção*/) {
HttpStatus status = HttpStatus.NOT_FOUND; // Erro 404
CustomError err = new CustomError(Instant.now(), status.value(),
e.getMessage(), request.getRequestURI());
// Retornar o objeto
return ResponseEntity.status(status).body(err);
}

@ExceptionHandler(DataBaseException.class)
public ResponseEntity<CustomError> database(DataBaseException e,
HttpServletRequest request) {
HttpStatus status = HttpStatus.BAD_REQUEST; // Erro 400
CustomError err = new CustomError(Instant.now(), status.value(),
e.getMessage(), request.getRequestURI());
return ResponseEntity.status(status).body(err);
}

}
```

Resposta Customizada de Validações

Na pasta dto, criar a classe FieldMessage.
Exemplo do desafio 3:

```
public class FieldMessage {
private String fieldName;
private String message;

public FieldMessage(String fieldName, String message) {
this.fieldName = fieldName;
this.message = message;
}

public String getFieldName() {
```

```
return fieldName;
}

public String getMessage() {
return message;
}

}
```

Na pasta dto, criar a classe ValidationError.
Exemplo do desafio 3:

```
public class ValidationError extends CustomError{
// Lista de atributos e erros
private List<FieldMessage> errors = new ArrayList<>();

public ValidationError(Instant timestamp, Integer status, String
error, String path) {
super(timestamp, status, error, path);
}

public List<FieldMessage> getErrors() {
return errors;
}

public void addError(String fieldName, String message) {
errors.add(new FieldMessage(fieldName, message));
}
}
```

Adicionar no ControllerExceptionHandler.
Exemplo do desafio 3:

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity<CustomError>
methodArgumentNotValid(MethodArgumentNotValidException e,
HttpServletRequest request) {
HttpStatus status = HttpStatus.UNPROCESSABLE_ENTITY; // Erro 422
ValidationError err = new ValidationError(Instant.now(),
status.value(), "Dados Inválidos",
request.getRequestURI());
// Verificar se há erros e adicionar na lista de erros
for (FieldError f : e.getBindingResult().getFieldErrors()) {
err.addError(f.getField(), f.getDefaultMessage());
}
// Retornar o objeto
return ResponseEntity.status(status).body(err);
}
```

Exceções personalizadas

Com as exceções personalizadas podemos retornar mensagens significativas aos possíveis erros.
Exemplo do desafio 3:

```
public class ResourceNotFoundException extends RuntimeException{

public ResourceNotFoundException(String msg) {
super(msg); }
}
```

```
public class DataBaseException extends RuntimeException{

public DataBaseException(String msg) {

super(msg);
}
}
```

READ - Busca por Id específico

O controller chama o serviço com o nome do método desejado
Exemplo do desafio 3:

```
// ResponseEntity = Padronização de retorno de resposta

// Buscar produto por Id
@GetMapping(value = "/{id}") // Retorno da consulta
public ResponseEntity<ClientDTO> findById(@PathVariable Long id) {
ClientDTO dto = service.findById(id);
return ResponseEntity.ok(dto); // Retornar Status 200
}
```

O serviço chama o repositório com o método desejado.
Exemplo do desafio 3:

```
@Transactional(readOnly = true) // Lock de leitura - Implementar
busca no banco de dados
public ClientDTO findById(Long id) {
// Buscar no banco de dados o Id e atribuir na variável
Client client = repository.findById(id).
// Se não encontrar o Id, lança exceção
orElseThrow(()-> new ResourceNotFoundException("Recurso não
encontrado!"));

// Converter o Product para ProductDTO e retornar para o
controlador
return new ClientDTO(client);
}
```

READ - FindAll Páginado

Buscar todos os elementos de forma paginada.
O controller chama o serviço com o nome do método desejado
Exemplo do desafio 3:

```
// ------- Buscar todos os clientes de forma
paginada---------------
@GetMapping
public ResponseEntity<Page<ClientDTO>> findAll(Pageable pageable)
{
// Pageable = Listagem paginada
Page<ClientDTO> dto = service.findAll(pageable);
return ResponseEntity.ok(dto); // Retornar Status 200
}
```

O serviço chama o repositório com o método desejado.
Exemplo do desafio 3:

```
// ------- Buscar todos os Clientes de forma Paginada
-------------
@Transactional(readOnly = true)
public Page<ClientDTO> findAll(Pageable pageable) {
// Buscar no banco de dados a lista de Produtos | Pageable =
Listagem paginada
Page<Client> result = repository.findAll(pageable);
// Converter a lista de Product para ProductDTO e retornar para o
controlador
return result.map(x -> new ClientDTO(x));
}
```

POST - Adicionar

O controller chama o serviço com o nome do método desejado
Exemplo do desafio 3:

```
// ------- Adicionar um novo Cliente no Banco
-----------------------
@PostMapping
public ResponseEntity<ClientDTO> insert
(@Valid /* Para checar as validações inseridas no DTO*/
@RequestBody ClientDTO dto) {
// @RequestBody = Corpo da requisição
dto = service.insert(dto); // Chamar o serviço de inserção e
passar os dados
// URI = link do recurso criado | Boa prática
```

```
URI uri =
ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")
.buildAndExpand(dto.getId()).toUri();
return ResponseEntity.created(uri).body(dto);// Retorno Customizado
Status 201 Created
}
```

O serviço chama o repositório com o método desejado.
Exemplo do desafio 3:

```
// ------- Adicionar um novo Cliente no Banco -------------
@Transactional
public ClientDTO insert(ClientDTO dto) {
// Criar um Cliente
Client entity = new Client();
// Copiar os dados do dto para a entidade
copyDtoToEntity(dto, entity);
// Salvar o Cliente no Repository (banco de dados)
entity = repository.save(entity);
// Converter e retornar em DTO
return new ClientDTO(entity);
}
```

PUT - Atualizar de forma idempotente

O controller chama o serviço com o nome do método desejado
Exemplo do desafio 3:

```
// ------- Atualizar um novo Cliente no Banco
----------------------
@PutMapping(value = "/{id}")
public ResponseEntity<ClientDTO> update(@PathVariable Long id,
@Valid @RequestBody ClientDTO dto) {
dto = service.update(id, dto); // Chamar o serviço de atualização
com o Id passado e as infos de atualização
return ResponseEntity.ok(dto); // Retorno Customizado Status 200
}
```

O serviço chama o repositório com o método desejado.
Exemplo do desafio 3:

```
// ------- Atualizar um Cliente no Banco -------------
@Transactional
public ClientDTO update(Long id, ClientDTO dto) {
try {
// Instanciar um Cliente com a referência do Id
Client entity = repository.getReferenceById(id);
// Copiar os dados do dto para a entidade
copyDtoToEntity(dto, entity);
// Salvar no banco de dados
```

```
entity = repository.save(entity);
// Retornar como DTO
return new ClientDTO(entity);
}
// Tratamento para quando tentar atualizar um Cliente que não
existe
catch (EntityNotFoundException e) {
// Se não encontrar o Id, lança exceção
throw new ResourceNotFoundException("Dados Inválidos");
}
}
```

DELETE

O controller chama o serviço com o nome do método desejado
Exemplo do desafio 3:

```
// ------- Deletar um novo Cliente no Banco
-----------------------
@DeleteMapping(value = "/{id}")
public ResponseEntity<Void> delete(@PathVariable Long id) {
service.delete(id); // Chamar o serviço de deletar com o Id
passado
return ResponseEntity.noContent().build(); // Retorno Customizado
Status 204 = Sem retorno
}
```

O serviço chama o repositório com o método desejado.
Exemplo do desafio 3:

```
// ------- Deletar um Cliente -------------
@Transactional(propagation = Propagation.SUPPORTS)
public void delete(Long id) {
if (!repository.existsById(id)) {
// Se não encontrar o Id, lança exceção
throw new ResourceNotFoundException("Recurso não encontrado");
}
try {
repository.deleteById(id);
}
catch (DataIntegrityViolationException e) {
throw new DataBaseException("Falha de integridade referencial");
}
}
```