CRUD

Um CRUD em Java se refere a um conjunto de operações básicas que podem ser realizadas em um banco de dados ou em qualquer outra fonte de dados. CRUD é um acrônimo para Create (Criar), Read (Ler), Update (Atualizar) e Delete (Excluir).

- **Create (Criar):** Envolve a criação de novos registros ou entradas na fonte de dados. Isso geralmente é feito através da inserção de novos dados no banco de dados.
- **Read (Ler):** Envolve a leitura ou recuperação de dados existentes da fonte de dados. Isso inclui operações como selecionar registros de um banco de dados com base em determinados critérios.
- **Update (Atualizar):** Envolve a modificação dos dados existentes na fonte de dados. Isso pode incluir a atualização de valores em um registro específico do banco de dados.
- **Delete (Excluir):** Envolve a remoção de registros ou entradas da fonte de dados. Isso implica na exclusão de registros específicos de um banco de dados.

```
import java.sql.*;

public class Main {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/seu_banco_de_dados";
        String usuario = "seu_usuario";
        String senha = "sua_senha";

        try (Connection conexao = DriverManager.getConnection(url, usuario, senha)) {
            // CREATE
            String sqlCreate = "INSERT INTO tabela_exemplo (campo1, campo2) VALUES (?, ?)";
            PreparedStatement createStatement = conexao.prepareStatement(sqlCreate);
            createStatement.setString(1, "valor1");
            createStatement.setString(2, "valor2");
            createStatement.executeUpdate();
            System.out.println("Novo registro criado com sucesso!");
```

```
        // READ
        String sqlRead = "SELECT * FROM tabela_exemplo";
        Statement readStatement = conexao.createStatement();
        ResultSet resultado = readStatement.executeQuery(sqlRead);
        while (resultado.next()) {
            System.out.println("Campo1: " + resultado.getString("campo1"));
            System.out.println("Campo2: " + resultado.getString("campo2"));
        }

        // UPDATE
        String sqlUpdate = "UPDATE tabela_exemplo SET campo1 = ? WHERE id = ?";
        PreparedStatement updateStatement = conexao.prepareStatement(sqlUpdate);
        updateStatement.setString(1, "novo_valor");
        updateStatement.setInt(2, 1); // Supondo que queremos atualizar o registro com ID = 1
        updateStatement.executeUpdate();
        System.out.println("Registro atualizado com sucesso!");

        // DELETE
        String sqlDelete = "DELETE FROM tabela_exemplo WHERE id = ?";
        PreparedStatement deleteStatement = conexao.prepareStatement(sqlDelete);
        deleteStatement.setInt(1, 1); // Supondo que queremos excluir o registro com ID = 1
        deleteStatement.executeUpdate();
        System.out.println("Registro excluído com sucesso!");
      } catch (SQLException e) {
        System.out.println("Erro: " + e.getMessage());
      }
    }
}
```

Browser

Um navegador da web, ou browser, é um aplicativo de software que permite aos usuários acessar, visualizar e interagir com informações na internet. Ele interpreta e exibe páginas da web, geralmente escritas em HTML, CSS e JavaScript, e permite que os usuários naveguem por essas páginas, sigam links, assistam a vídeos, preencham formulários e muito mais. Os navegadores também armazenam dados de navegação, como histórico e cookies, e oferecem recursos como favoritos e abas para facilitar a experiência do usuário na web. Exemplos populares incluem Google Chrome, Mozilla Firefox, Microsoft Edge e Safari.

HTTP

HTTP (Hypertext Transfer Protocol) é o protocolo usado para transferir dados na internet. Ele permite que navegadores solicitem e recebam páginas da web de servidores, seguindo um modelo de requisição e resposta. O HTTP é stateless, utiliza diferentes métodos (GET, POST, etc.), cabeçalhos para informações adicionais e códigos de status para indicar o resultado da operação.

- **Stateless (Sem estado):** Cada requisição HTTP é tratada de forma independente, sem conhecimento do estado das requisições anteriores. Isso significa que cada requisição é tratada isoladamente, sem memória das requisições anteriores.
- **Métodos HTTP:** O HTTP define vários métodos (ou verbos) que indicam a ação a ser realizada no recurso especificado. Os principais métodos incluem GET (para obter dados), POST (para enviar dados para o servidor), PUT (para atualizar dados), DELETE (para excluir dados) e outros.
- **Cabeçalhos HTTP:** Os cabeçalhos HTTP fornecem informações adicionais sobre a requisição ou resposta, como o tipo de conteúdo, cookies, autenticação, codificação de caracteres e muito mais.
- **Códigos de status HTTP:** Os códigos de status são números de três dígitos enviados pelo servidor em resposta a uma requisição HTTP, indicando o sucesso ou a falha da operação.

  Alguns exemplos:

• **200: OK - Indica que a requisição foi bem-sucedida.** • **404: Não Encontrado - Indica que o recurso solicitado não foi encontrado no servidor.** • **500: Erro Interno do Servidor - Indica que ocorreu um erro no servidor ao processar a requisição.**	• **302: Redirecionamento Temporário - Indica que o recurso solicitado foi movido temporariamente para uma nova localização.** • **401: Não Autorizado - Indica que a requisição requer autenticação ou as credenciais fornecidas não são válidas.** • **403: Proibido - Indica que o servidor entende a requisição, mas se recusa a atendê-la por razões legais ou de permissão.**

Model

A Model em uma aplicação Spring, o termo Model faz parte do padrão de design MVC (Model-View-Controller), que é comumente usado em estruturas como Spring MVC.

Em Spring MVC, o "Model" é uma parte do padrão MVC e é usado para representar os dados que serão enviados para a visualização (View) para serem renderizados. Ele geralmente consiste em um objeto Java POJO (Plain Old Java Object) que contém os dados a serem exibidos na interface do usuário. Um exemplo simples de como você pode usar Model em uma aplicação Spring MVC:

Suponha que você queira exibir o nome de um usuário em uma página da web. Você pode criar uma classe Java chamada User para representar um usuário:

```
public class User {
    private String name;

    // Getters e Setters
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Em seguida, você pode ter um controlador (Controller) que processa a requisição e preenche o modelo com os dados do usuário:

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class UserController {

    @GetMapping("/user")
```

```
    public String getUser(Model model) {
        // Criar um objeto User e definir o nome
        User user = new User();
        user.setName("John Doe");

        // Adicionar o objeto User ao modelo
        model.addAttribute("user", user);

        // Retornar o nome da página ou template que exibirá os dados do usuário
        return "userPage";
    }
}
```

Rest

REST = CONTROLLER

CAMADA REST "SEM REGRAS NEGOCIAIS"

REST (Representational State Transfer) é um estilo arquitetural comumente usado para criar APIs (Application Programming Interfaces) na web. Ele se baseia em princípios simples e padronizados para facilitar a comunicação entre sistemas distribuídos na internet.

Os princípios fundamentais do REST incluem:

Recursos: Os recursos são as entidades que um sistema REST manipula, como usuários, posts, produtos, etc. Cada recurso é identificado por um URI (Uniform Resource Identifier).

Operações semânticas: REST utiliza métodos HTTP padronizados para operações em recursos. Os principais métodos são GET (para recuperar dados), POST (para criar novos recursos), PUT (para atualizar recursos existentes) e DELETE (para excluir recursos).

Representações: Os recursos podem ter várias representações, como JSON, XML, HTML, etc. O cliente e o servidor podem negociar a representação mais adequada durante a comunicação.

Estado da aplicaçã: O estado da aplicação é transferido entre cliente e servidor nas requisições. O servidor não mantém o estado da sessão do cliente, tornando as operações independentes.

Interface uniforme: REST utiliza uma interface uniforme entre clientes e servidores, o que simplifica o desenvolvimento e a manutenção de sistemas distribuídos.

Spring Boot: Com Spring Boot, você pode criar facilmente uma API RESTful usando o módulo Spring MVC. Spring Boot simplifica muito o desenvolvimento, configuração e implantação de aplicativos Java.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}

@RestController
class HelloController {

    @GetMapping("/hello")
    public String hello() {
        return "Hello, world!";
    }

    @GetMapping("/hello/{name}")
    public String helloName(@PathVariable String name) {
        return "Hello, " + name + "!";
    }
}
```

Service

A pasta service em Java é um componente que executa tarefas específicas dentro de uma aplicação. Ele encapsula funcionalidades e lógica de negócios que podem ser compartilhadas e reutilizadas por outros componentes. No contexto de uma aplicação Spring Boot, um serviço é frequentemente anotado com `@Service`, o que permite que o Spring o gerencie e o injete em outras partes da aplicação. Em resumo, um serviço em Java é uma maneira de organizar e modularizar o código, promovendo a reutilização e a manutenção mais fácil. **Exemplo:**

Temos uma classe ExemploService que é anotada com @Service, indicando que é um componente de serviço gerenciado pelo Spring. No método obterMensagem(), retornamos uma simples mensagem. A classe principal Application está anotada com @SpringBootApplication, indicando que é uma aplicação Spring Boot. Quando executada, o método main inicia a aplicação.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.stereotype.Service;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}

@Service
class ExemploService {

    public String obterMensagem() {
        return "Olá, este é um serviço de exemplo!";
    }
}
```

DAO

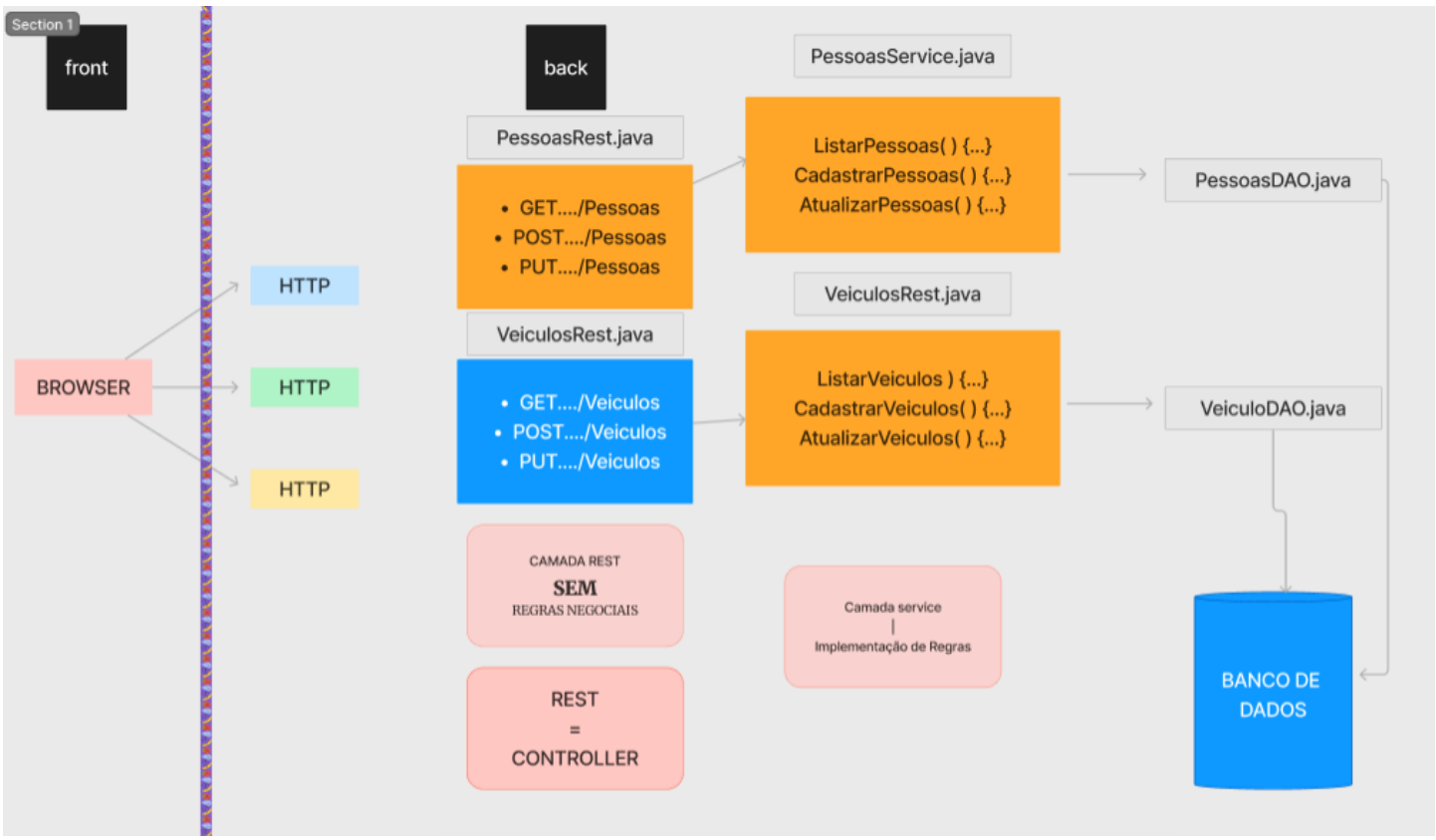
DAO geralmente representa uma classe que implementa o padrão de acesso a dados conhecido como "Data Access Object" (DAO). Este padrão é usado para separar a lógica de acesso a dados do restante da aplicação, permitindo que a aplicação interaja com a camada de dados de forma independente do banco de dados subjacente.Em um projeto Java, especialmente em uma aplicação baseada em Spring, você pode ter um arquivo DAO.java para cada entidade do seu modelo de dados. Por exemplo, se você tiver uma entidade User, você pode ter um UserDAO.java para lidar com operações de banco de dados relacionadas a usuários.

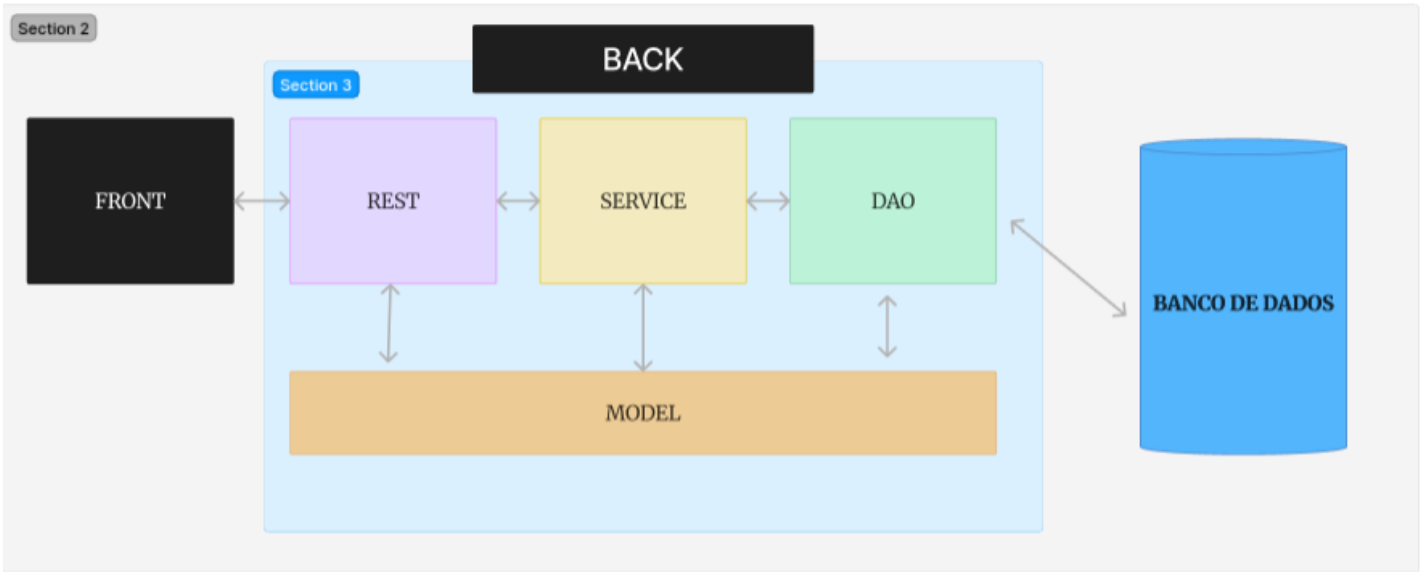
Exemplo:

```
import java.util.List;

public interface UserDAO {
    // Métodos para operações CRUD (Create, Read, Update, Delete) relacionadas a usuários
    void save(User user);
    User findById(Long id);
    List<User> findAll();
    void update(User user);
    void delete(User user);
}
```

Neste exemplo, UserDAO é uma interface que define métodos para operações básicas de CRUD em usuários, como salvar, encontrar por ID, encontrar todos, atualizar e excluir.A implementação concreta desses métodos será feita em uma classe concreta que implementa esta interface.
Exemplo:

```
import java.util.ArrayList;
import java.util.List;

public class UserDAOImpl implements UserDAO {

    private List<User> userList = new ArrayList<>();

    @Override
    public void save(User user) {
        userList.add(user);
    }

    @Override
    public User findById(Long id) {
        for (User user : userList) {
            if (user.getId().equals(id)) {
                return user;
            }
        }
        return null;
    }
}
```


Section 1
front
back
PessoasService.java
PessoasRest.java
ListarPessoas( ) {...}
CadastrarPessoas( ) {...}
AtualizarPessoas( ) {...}
PessoasDAO.java
• GET..../Pessoas
• POST..../Pessoas
• PUT..../Pessoas
HTTP
VeiculosRest.java
VeiculosRest.java
BROWSER
HTTP
ListarVeiculos) {...}
CadastrarVeiculos( ) {...}
AtualizarVeiculos( ) {...}
VeiculoDAO.java
• GET..../Veiculos
• POST..../Veiculos
• PUT..../Veiculos
HTTP
CAMADA REST
SEM
REGRAS NEGOCIAIS
Camada service
|
Implementação de Regras
BANCO DE DADOS
REST
=
CONTROLLER


Section 2
BACK
Section 3
FRONT
REST
SERVICE
DAO
BANCO DE DADOS
MODEL