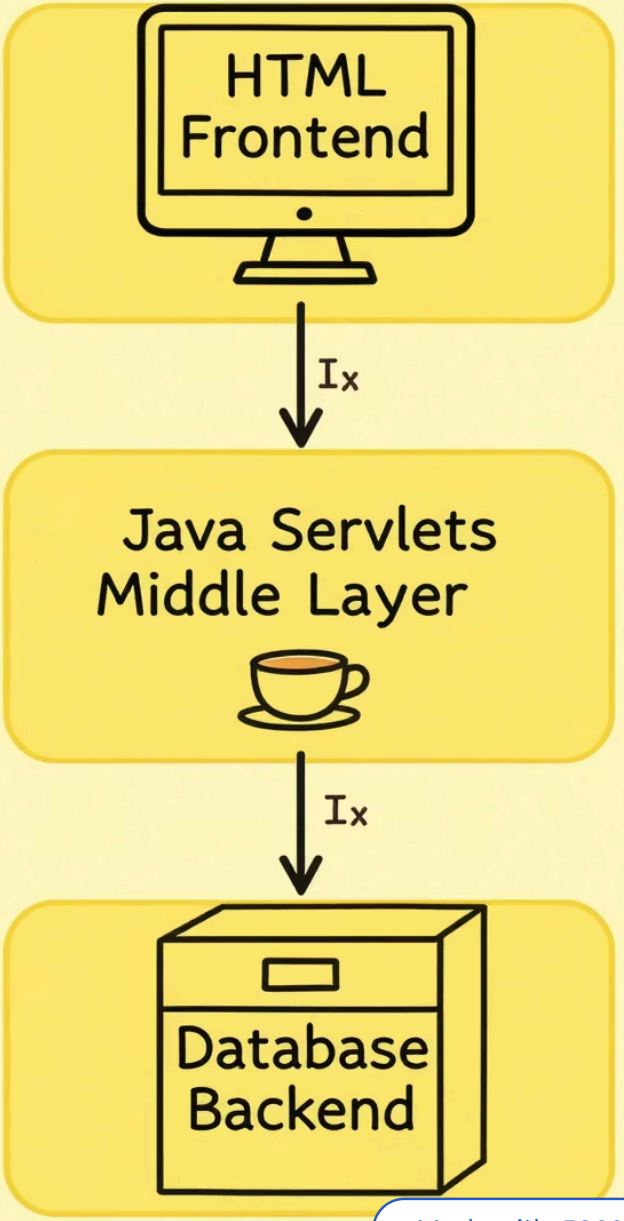
Sistema de Gerenciamento de Fretes - TransCarga

Apresentação do sistema web em Java para gerenciamento de fretes da empresa fictícia TransCarga, demonstrando uma arquitetura em três camadas com aplicação HTML, negócios em Servlets Java e persistência com JPA e MariaDB.

Introdução ao Sistema TransCarga

A TransCarga é uma empresa fictícia de transporte de cargas que atua em todo o território nacional. Este estudo de caso demonstra a criação de um sistema web em Java, com arquitetura em três camadas:

- **Aplicação (HTML)**
  Interfaces simples para entrada e visualização de dados
- **Negócios (Servlets Java)**
  Lógica de controle e processamento
- **Persistência (JPA com MariaDB)**
  Gerenciamento e armazenamento de dados



Arquitetura em Três Camadas com MVC

MODEL (Modelo)

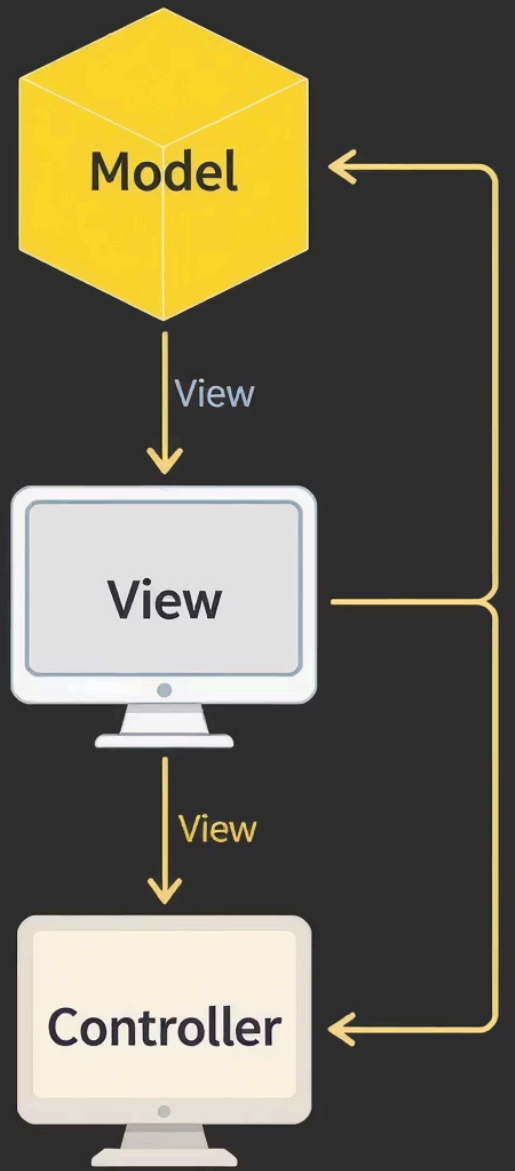
Camada de Persistência: Classes como Frete.java e FreteDAO.java responsáveis por acessar e manipular dados no banco.

CONTROLLER (Controle)

Camada de Negócios: Servlets Java que recebem requisições, invocam a lógica de negócios e direcionam respostas.

VIEW (Visão)

Camada de Aplicação: Interfaces HTML para interação com usuário, geradas dinamicamente pelos Servlets.

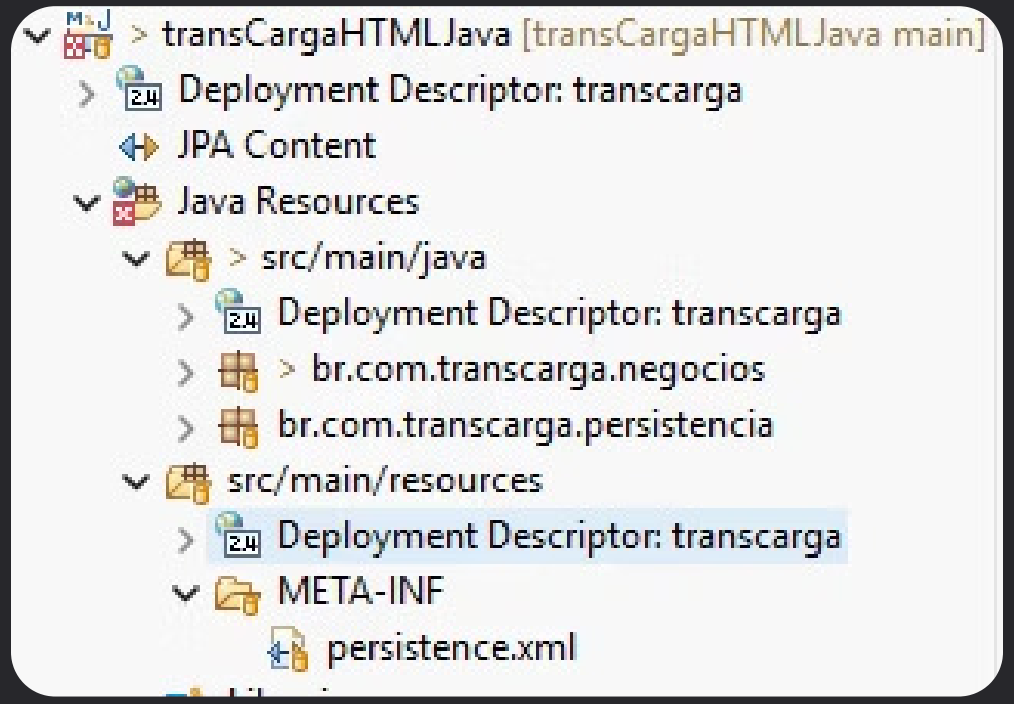


Estrutura do Projeto TransCarga

A organização do projeto segue uma estrutura clara que separa as diferentes camadas da aplicação:

- src/ - Contém os pacotes de negócios e persistência
- WebContent/ - Armazena os arquivos HTML da camada de aplicação
- META-INF/ - Contém o arquivo persistence.xml com definições para persistência

Esta separação facilita a manutenção, escalabilidade e organização do código, seguindo as melhores práticas de desenvolvimento Java EE.



1- Camada de Aplicação:
1.1 Página de Cadastro de Fretes

HTML - cadastrarFrete.html

Esta página permite inserir os dados de um novo frete. O formulário contém campos para:

- Destino (texto)
- Peso em kg (número)
- Transportadora (texto)

Os dados são enviados via método POST para o servlet FreteServlet, que processa as informações e as armazena no banco de dados.


```html
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Cadastrar Frete</title>
</head>
<body>
    <h2>Cadastro de Frete</h2>
    <form action="FreteServlet" method="post" accept-charset="UTF-8">
        Destino: <input type="text" name="destino" required><br>
        Peso (kg): <input type="number" name="peso" required><br>
        Transportadora: <input type="text" name="transportadora" required><br>
        <input type="submit" value="Cadastrar Frete">
    </form>
    <br>
    <a href='index.html'>Home</a>
</body>
</html>
```

1- Camada de Aplicação (view):
1.2 Página de Listagem de Fretes

A página `listarFretes.html` é responsável por exibir todos os fretes cadastrados no sistema TransCarga. Sua estrutura é intencionalmente simples, focando na integração com o servlet para apresentar os dados de forma dinâmica.

Conteúdo Dinâmico

A tabela de fretes é carregada e processada integralmente pelo `FreteServlet`, garantindo dados sempre atualizados.

Integração via Iframe

Um `<iframe>` é utilizado para incorporar o conteúdo gerado pelo servlet, mantendo a responsabilidade de visualização separada da lógica de negócios.

```html
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <title>Listar Fretes</title>
</head>
<body>
    <!-- A tabela será carregada e processada 100% no servlet -->
    <iframe src="/transcarga/FreteServlet"
            style="width:400px; height:auto; border:none;">
    </iframe>
    <br><a href='index.html'>Home</a>
</body>
</html>
```

2- Camada de Negócios (controller):

FreteServlet.java (Servlet)

Método doPost

Responsável pelo cadastro de fretes:

- Recebe parâmetros do formulário HTML
- Cria um novo objeto Frete
- Utiliza o FreteDAO para persistir no banco
- Redireciona para a página de listagem

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    request.setCharacterEncoding("UTF-8");
    response.setContentType("text/html; charset=UTF-8");

    String destino = request.getParameter("destino");
    double peso = Double.parseDouble(request.getParameter("peso"));
    String transportadora = request.getParameter("transportadora");

    Frete frete = new Frete();
    frete.setDestino(destino);
    frete.setPeso(peso);
    frete.setTransportadora(transportadora);

    FreteDAO dao = new FreteDAO();
    dao.cadastrarFrete(frete);
    response.sendRedirect("listarFretes.html");
}
```

Método doGet

Responsável pela listagem de fretes:

- Consulta todos os fretes cadastrados via DAO
- Gera dinamicamente uma tabela HTML
- Exibe ID, destino, peso e transportadora

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    FreteDAO dao = new FreteDAO();
    List<Frete> fretes = dao.listarFretes();
    request.setCharacterEncoding("UTF-8");
    response.setContentType("text/html; charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println(" <h2>Lista de Fretes</h2>");
    out.println("<table border='1'>");
    out.println("<thead>"
            + "<tr>"
            + "<th>ID</th>"
            + "<th>Destino</th>"
            + "<th>Peso</th>"
            + "<th>Transportadora</th></tr></thead>");

    for (Frete f : fretes) {
        out.printf("<tr><td>%d</td><td>%s</td><td>%.2f</td><td>%s</td></tr>",
                f.getId(), f.getDestino(), f.getPeso(), f.getTransportadora());
    }
    out.println("</tbody></table>");
}
```

3- Camada de Persistência (model)

Frete.java (Entidade)

Classe anotada com @Entity que representa a entidade Frete no banco de dados, contendo:

- ID (chave primária com geração automática)
- Destino (String)
- Peso (double)
- Transportadora (String)

Inclui getters e setters para todos os atributos.

```java
@Entity
public class Frete {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String destino;
    private double peso;
    private String transportadora;

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getDestino() { return destino; }
    public void setDestino(String destino) { this.destino = destino; }

    public double getPeso() { return peso; }
    public void setPeso(double peso) { this.peso = peso; }

    public String getTransportadora() { return transportadora; }
    public void setTransportadora(String transportadora) { this.transportadora = transportadora; }
```

3- Camada de Persistência (model)

FreteDAO.java

Classe responsável pelas operações de banco de dados:

- cadastrarFrete(): persiste um novo frete
- listarFretes(): retorna todos os fretes cadastrados

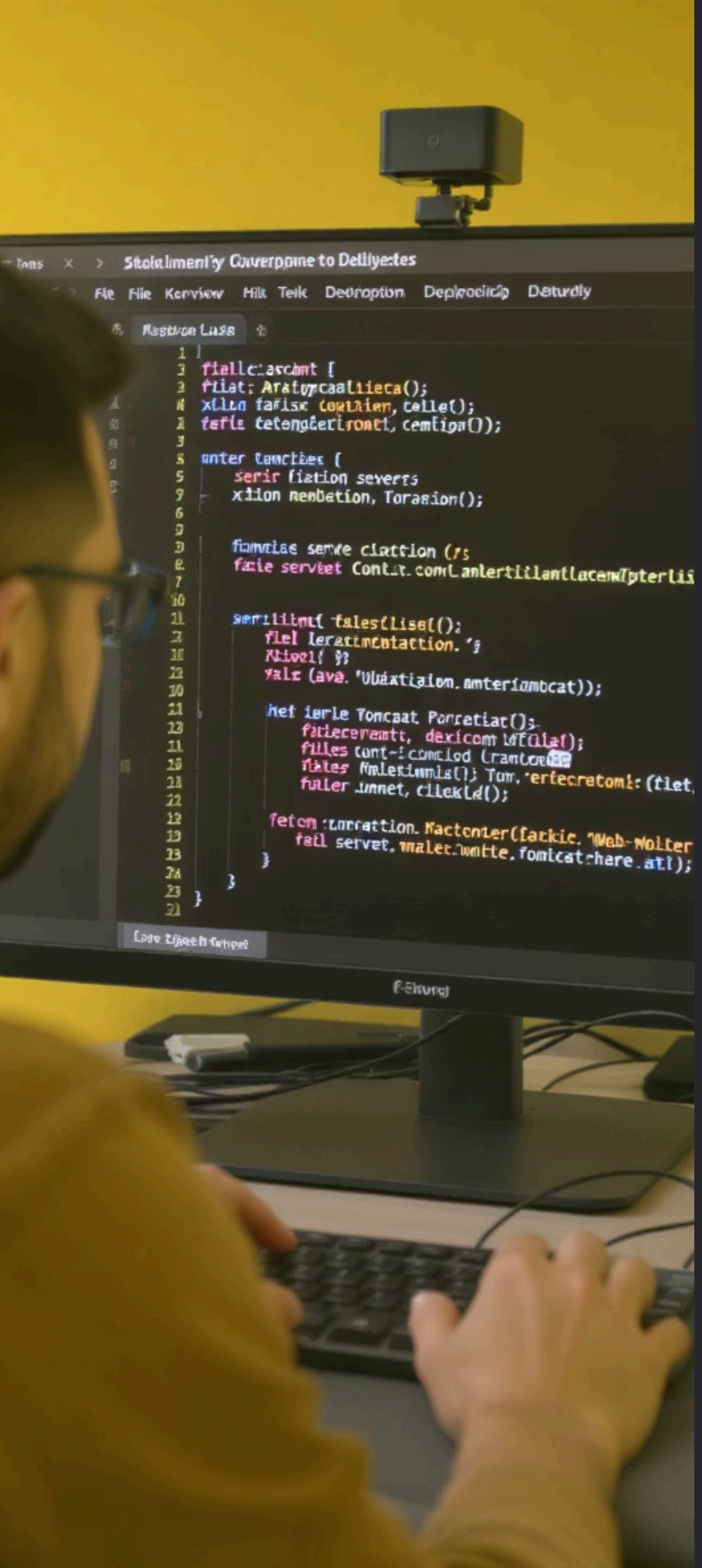
Utiliza JPA com EntityManager para gerenciar as operações de persistência.

```java
public class FreteDAO {

    private EntityManagerFactory emf = Persistence.createEntityManagerFactory("TransCargaPU");

    public void cadastrarFrete(Frete frete) {
        EntityManager em = emf.createEntityManager();
        try {
            em.getTransaction().begin();
            em.persist(frete);
            em.getTransaction().commit();
        } finally {
            em.close();
        }
    }

    public List<Frete> listarFretes() {
        EntityManager em = emf.createEntityManager();
        try {
            return em.createQuery("SELECT f FROM Frete f", Frete.class).getResultList();
        } catch (Exception e) {
            e.printStackTrace();
            return List.of(); // retorna lista vazia em caso de erro
        } finally {
            if (em != null && em.isOpen()) em.close();
        }
    }
}
```



Como Executar o Projeto

Importar o Projeto no Eclipse IDE

No Eclipse, use File->Import-> 'Import existing Maven projects'

Configurar Servidor

Configure um servidor Tomcat e associe ao projeto

Configurar Banco de Dados

Certifique-se que o MariaDB está em execução e configurado no persistence.xml

Acessar o Sistema

Acesse http://localhost:8080/transcarga/index.html

GitHub

Dê uma olhada no código fonte completo em: https://github.com/clistion/transCargaHTMLJava