**Controle Financeiro Pessoal**

**Requisitos**

Contexto, o projeto é um CRUD que visa integrar o Frontend (**HTML**, **CSS** e **JavaScript**) ao backend (**Java** puro e **PostgreSQL**).

Teremos uma tela de login.

Teremos uma segunda tela com as tabelas.

O usuário deverá ser capaz de cadastrar login e senha.

O usuário deverá ser capaz de alterar login de senha.

O usuário deverá ser capaz de cadastrar os valores e textos das tabelas.

O usuário deverá ser capaz de editar os valores e textos das tabelas.

O usuário deverá ser capaz de excluir os valores e textos das tabelas.

O usuário deverá ser capaz de ver o resultado final/parcial da diferença entre custos e renda.

Ao logar na aplicação o usuário deverá encontrar seu os dados das tabelas salvos com as últimas alterações feitas.

O usuário deverá ser capaz de gerar um relatório em PDF sempre que desejar.

O relatório deverá imprimir as tabelas e o resultado final/parcial.

A tabela **CUSTOS** terá 12 linhas e uma linha de **TOTAL** abaixo. O usuário deverá ser capaz de adicionar mais linhas se desejar abaixo da linha 12 e acima da linha **TOTAL.**

A tabela **RENDA** terá 4 linhas e uma linha de **TOTAL** abaixo. O usuário deverá ser capaz de adicionar mais linhas se desejar abaixo da linha 4 e acima da linha **TOTAL.**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Estrutura básica do projeto**  **Backend (Java + JDBC):**   * Projeto **Java** com estrutura **MVC** (Model-View-Controller). * Servlets para lidar com requisições **HTTP**. * **JDBC** para conectar ao **PostgreSQL** e executar comandos **SQL**. * Configurar o **web**.**xml** para mapear meus **servlets**. * **Apache** **Tomcat** * **Maven** para gerenciamento automático de dependências   + Servelet API   + PostgreSQL JDBC Driver   + JSON Parser - Jackson   + PDF Generator - Apache PDFBox   **Frontend (HTML/CSS/JS):**   * Páginas **HTML** com formulários para criar e editar dados. * **CSS** para estilizar. * **JavaScript** para melhorar a experiência do usuário (validações, **AJAX**, etc).   **Exemplo de fluxo CRUD**   1. **Create**: Formulário **HTML** envia dados via **POST** para um servlet. 2. **Read**: **Servlet** consulta o banco e retorna dados para exibir. 3. **Update**: Formulário com dados preenchidos envia via **POST**/**PUT**. 4. **Delete**: Botão que envia requisição para deletar um item.  O que garantir no backend:  * Um **servlet** mapeado para /salvar-dados no web.xml. * Esse servlet deve aceitar requisições **POST**. * Ele deve ler o corpo da requisição como **JSON** e salvar os dados no **PostgreSQL**.   **Planejamento de Classes do Backend**  **model** — *Representa os dados*  Usuario **-** Representa o usuário: id, login, senha, data de cadastro.  ***Validação de senha e login*** *No futuro, quando formos implementar lógica de negócio, será interessante adicionar validações (ex:* ***senha com 6 dígitos, login alfanumérico****). Isso pode ser feito em uma classe auxiliar ou diretamente nos* ***setters****, dependendo da abordagem.*  Transacao - Representa uma transação: id, tipo (RENDA/CUSTO), descrição, valor, data.  ***Validação do campo*** *tipo - Como o campo só aceita "RENDA" ou "CUSTO", podemos pensar em usar um* ***enum*** *no futuro ou aplicar validação no* ***setter*** *ou em uma classe auxiliar.*  **dao** — *Acesso ao banco de dados (****JDBC****)*  UsuarioDAO - **CRUD** de usuários: cadastrar, buscar por login, atualizar senha, etc.  TransacaoDAO - **CRUD** de transações: inserir, listar por usuário, editar, excluir  ConexaoFactory - Gerencia a conexão com o **PostgreSQL** (DriverManager + URL, usuário, senha)  **controller** — ***Servlets*** *que recebem requisições* ***HTTP***  LoginServlet - Autentica o usuário, inicia sessão  CadastroUsuarioServlet - Cadastra novo usuário com validações  TransacaoServlet - Recebe **JSON** com transações e salva no banco (/salvar-dados)  RelatorioPDFServlet - Gera e envia o **PDF** com as tabelas e totais (/gerar-relatorio)  **util** — *Classes auxiliares*  JsonUtil - Converte entre **JSON** e objetos **Java** (**Jackson**)  PDFGenerator - Gera o relatório em **PDF** com os dados formatados (**PDFBox**)  ValidadorUsuario - Valida login e senha conforme regras do front-end  CalculadoraFinanceira - Calcula totais de RENDA, CUSTO e diferença final/parcial  **config** — *Configurações*  web.xml - Mapeia os **servlets** e configura o projeto web  db.properties - Armazena configurações de conexão com o banco (URL, usuário, senha) Fluxo de execução típico  1. Front envia login → LoginServlet → UsuarioDAO → valida e inicia sessão 2. Front envia transações via JSON → TransacaoServlet → TransacaoDAO → salva no banco 3. Front solicita PDF → RelatorioPDFServlet → TransacaoDAO → PDFGenerator → envia PDF   src/  └── main/  └── java/  └── br/  └── com/  └── brunoluna/  └── controlefinanceiro/  ├── model/  ├── dao/  ├── controller/  ├── util/  └── config/ |  |  |
|  |  |  |
| **Frontend - concluído**   Tela de **login** com layout centralizado e botão de cadastro   Tela de **cadastro** com validação:   * Login: só letras e números * Senha: exatamente 6 dígitos numéricos    Tela das **tabelas** com:   * 12 linhas fixas + TOTAL na tabela de CUSTOS * 4 linhas fixas + TOTAL na tabela de RENDA * Botões para adicionar linhas * Edição e exclusão de valores * Cálculo automático da diferença * Botão para gerar PDF |  |  |
|  |  |  |
|  |  |  |
| **Banco de Dados - concluído**  **Etapa 1: Modelagem de tabelas**  Modelar tabelas significa **transformar os dados que seu sistema precisa em estruturas organizadas** dentro do banco. Cada tabela representa uma **entidade** (como usuário, transação, produto) e cada coluna representa um **atributo** dessa entidade.  **Etapa 2: Identificar as entidades do sistema**  No seu projeto de controle financeiro, as principais entidades são:   | **Entidade** | **O que representa** | | --- | --- | | Usuario | Cada pessoa que usa o sistema | | Transacao | Cada entrada de custo ou renda |   **Etapa 3: Definir os atributos de cada entidade**  **Tabela Usuario**   * id → identificador único (chave primária) * login → nome de usuário * senha → senha criptografada * data\_cadastro → quando foi criado   **Tabela Transacao**   * id → identificador único * usuario\_id → referência ao usuário (chave estrangeira) * tipo → RENDA ou CUSTO * descricao → texto explicativo * valor → número decimal * data → data da transação   **Etapa 4: Relacionamentos**   * Um Usuario pode ter **muitas** Transacoes * Cada Transacao pertence a **um único** Usuario   Isso é um relacionamento **1:N** (um para muitos)  **Etapa 5: Tipos de dados**  Você precisa escolher o tipo certo pra cada coluna:   | **Coluna** | **Tipo sugerido** | | --- | --- | | id | SERIAL ou INT | | login | VARCHAR(50) | | senha | VARCHAR(255) | | valor | DECIMAL(10,2) | | data | DATE | | tipo | VARCHAR(10) |   **Etapa 6: Regras e restrições**   * login deve ser **único** * senha não pode ser nula * valor deve ser **positivo** * tipo deve aceitar apenas RENDA ou CUSTO   **SQL**  Banco de Dados: controle\_financeiro  Tabela usuarios: gerencia login e senha  create table usuarios (  id serial primary key,  login varchar(50) unique not null,  senha varchar(255) not null,  data\_cadastro TIMESTAMP default current\_timestamp  );  Tabela transacoes: armazena custos e rendas vinculados ao usuário  create table transacoes (  id serial primary key,  usuario\_id integer not null,  tipo varchar(10) not null check (tipo in('RENDA', 'CUSTO')),  descricao text,  valor decimal(10,2) not null check (valor > 0),  data date not null,  foreign key (usuario\_id) references usuarios(id)  ); |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |