**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.**

**Funcionalidades Esperadas:**

**Autenticação:**

* **Cadastro de usuário (login/senha)**
* **Login com sessão**
* **Alteração de senha**

**Gestão Financeira:**

* **Tabela de Custos (12 linhas fixas + adicionais + TOTAL)**
* **Tabela de Renda (4 linhas fixas + adicionais + TOTAL)**
* **CRUD completo: Criar, Ler, Editar, Excluir transações**
* **Cálculo automático: Totais + Diferença (Renda - Custos)**

**Relatórios:**

* **Geração de PDF com tabelas e resultados**

**Persistência:**

* **Dados salvos por usuário**
* **Recuperação ao logar**

**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
  + BCrypt
  + slf4j api – logback classic

**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**.

src/

└── main/

└── java/

└── br/

└── com/

└── brunoluna/

└── controlefinanceiro/

├── model/

├── dao/

├── controller/

├── util/

└── config/

**Frontend - concluído**

**Tecnologias:**

* **HTML5, CSS3, JavaScript ES6+**
* Arquitetura modular

 Tela de **login** com layout centralizado e botão de cadastro

 Tela de **cadastro** com validação:

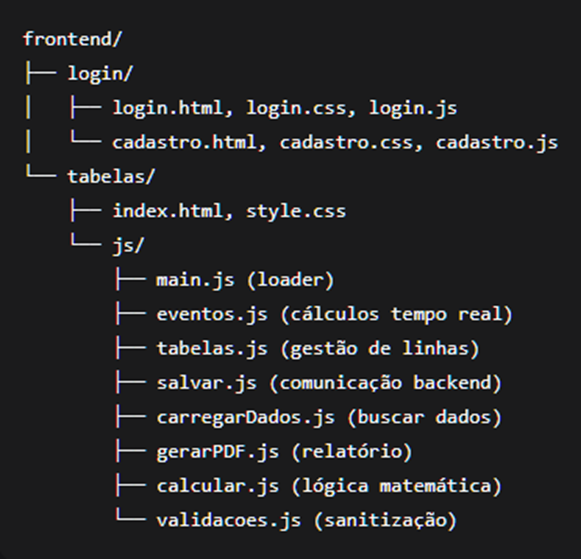
* Login: só letras e números
* Senha: exatamente 6 dígitos numéricos

 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

**Funcionalidades Implementadas:**

* Interface responsiva
* Validações em tempo real
* Sistema de IDs (data-id) para sincronização
* Cálculos automáticos
* Adição/remoção dinâmica de linhas
* Preparado para integração com backend



**Banco de Dados – concluído**

**Modelagem do Banco de Dados - Controle Financeiro Pessoal**

**Arquitetura e Decisões de Design**

A modelagem do banco segue uma abordagem **normalizada** com três entidades principais, refletindo o domínio do negócio de forma clara e objetiva.

**Estrutura das Tabelas**

**1. Tabela usuarios**

* **Função**: Armazenar informações de autenticação e perfil dos usuários
* **Campos**: id (PK), login (único), senha (criptografada), data\_cadastro
* **Propósito**: Garantir isolamento e segurança dos dados entre diferentes usuários

**2. Tabelas custos e rendas**

* **Função**: Armazenar transações financeiras separadas por natureza (despesas vs receitas)
* **Estrutura idêntica**: id (PK), usuario\_id (FK), descricao, valor, data\_criacao
* **Decisão estratégica**: Separar custos e rendas melhora a performance de consultas e simplifica a lógica de negócio

**Relacionamentos e Integridade Referencial**

**Relacionamento 1:N (Um para Muitos)**

* **Um** usuário pode ter **muitos** custos
* **Um** usuário pode ter **muitas** rendas
* Implementado via chave estrangeira usuario\_id

**Política de DELETE CASCADE**

* **Comportamento**: Quando um usuário é excluído, todas suas transações são automaticamente removidas
* **Vantagem**: Mantém a integridade do banco, evitando dados órfãos
* **Adequação**: Ideal para sistemas onde os dados do usuário não precisam ser mantidos após exclusão da conta

**Constraints e Validações**

* CHECK (valor > 0): Garante que valores sejam positivos
* UNIQUE no login: Previte duplicação de usuários
* NOT NULL nos campos essenciais: Mantém consistência dos dados

**Benefícios Desta Modelagem**

1. **Clareza conceitual**: Estrutura reflete diretamente o domínio do problema
2. **Performance**: Consultas específicas por tipo de transação são otimizadas
3. **Manutenibilidade**: Alterações em uma tabela não afetam a outra
4. **Escalabilidade**: Facilita a adição de novos tipos de transações no futuro

Banco de Dados: controle\_financeiro

-- Tabela usuarios

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 custos

create table custos (

id serial primary key,

usuario\_id integer not null,

descricao text not null,

valor decimal(10,2) not null check (valor >= 0),

data\_criacao timestamp default current\_timestamp,

foreign key (usuario\_id) references usuarios(id) on delete cascade

);

-- Tabela renda

create table renda (

id serial primary key,

usuario\_id integer not null,

descricao text not null,

valor decimal(10,2) not null check (valor >= 0),

data\_criacao timestamp default current\_timestamp,

foreign key (usuario\_id) references usuarios(id) on delete cascade

);

**Direto do Postgres**

CREATE TABLE IF NOT EXISTS public.usuarios

(

id integer NOT NULL DEFAULT nextval('usuarios\_id\_seq'::regclass),

login character varying(50) NOT NULL,

senha character varying(255) NOT NULL,

data\_cadastro timestamp without time zone DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT usuarios\_pkey PRIMARY KEY (id),

CONSTRAINT usuarios\_login\_key UNIQUE (login) -- ✅ UNIQUE constraint

)

CREATE TABLE IF NOT EXISTS public.custos

(

id integer NOT NULL DEFAULT nextval('custos\_id\_seq'::regclass),

usuario\_id integer NOT NULL,

descricao text NOT NULL,

valor numeric(10,2) NOT NULL,

data\_criacao timestamp without time zone DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT custos\_pkey PRIMARY KEY (id),

CONSTRAINT custos\_usuario\_id\_fkey FOREIGN KEY (usuario\_id)

REFERENCES public.usuarios (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE, -- ✅ ON DELETE CASCADE

CONSTRAINT chk\_custos\_valor\_nao\_negativo CHECK (valor >= 0::numeric) -- ✅ CHECK constraint

)

CREATE TABLE IF NOT EXISTS public.renda

(

id integer NOT NULL DEFAULT nextval('renda\_id\_seq'::regclass),

usuario\_id integer NOT NULL,

descricao text NOT NULL,

valor numeric(10,2) NOT NULL,

data\_criacao timestamp without time zone DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT renda\_pkey PRIMARY KEY (id),

CONSTRAINT renda\_usuario\_id\_fkey FOREIGN KEY (usuario\_id)

REFERENCES public.usuarios (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE, -- ✅ ON DELETE CASCADE

CONSTRAINT chk\_renda\_valor\_nao\_negativo CHECK (valor >= 0::numeric) -- ✅ CHECK constraint

)

**Backend em andamento**

**model**

* Usuario.java ........................................................................................................ok
* Custos.java ..........................................................................................................ok
* Renda.java ..........................................................................................................ok

**dao**

* ConexaoFactory.java ..........................................................................................ok
  + getConnection()
* UsuarioDAO.java ...............................................................................................ok
  + cadastrarUsuario(Usuario usuario)
  + buscarPorLogin(String login)
  + buscarPorId(int id)
  + atualizarSenha(Usuario usuario)
  + deletarUsuario(Usuario usuario)
* CustosDAO.java .................................................................................................ok
  + cadastrarCustos(Custos custos)
  + carregarCustos(int usuarioId)
  + atualizarCustos(Custos custos)
  + deletarCustos(Custos custos)
* RendaDAO.java ..................................................................................................ok
  + cadastrarRenda(Renda renda)
  + carregarRenda(int usuarioId)
  + atualizarRenda(Renda renda)
  + deletarRenda(Renda renda)

**controller**

**util**

**config**