**ESTÁCIO**

**CAMPUS FORTALEZA PARANGABA**



**SISTEMA DE ADMINISTRAÇÃO DE DESPESAS PESSOAIS**

**RAIMUNDO LUCAS PEREIRA….........................…202202234362**

**ISRAEL MELO DE CASTRO....….........................…202108525189**

**CYTHIA MOREIRA MAIA**

**2023.2**

**FORTALEZA/CEARÁ**

**1. Introdução**

O projeto "Sistema de Administração de Despesas Pessoais" foi desenvolvido como parte de um curso para construir uma aplicação que permite o gerenciamento eficiente de despesas pessoais. Este documento fornece uma visão abrangente sobre a estrutura, funcionalidades e implementação técnica do sistema.

**2. Tecnologias Utilizadas**

* **Flask (Python):** Framework web utilizado para criar o backend da aplicação, oferecendo rotas para manipulação de dados e interação com o frontend.
* **SQLite:** Sistema de gerenciamento de banco de dados relacional utilizado para armazenar informações sobre despesas, condições de pagamento e status.
* **HTML/CSS:** Linguagens utilizadas para criar a interface do usuário e estilizar a aplicação web.

**3. Estrutura do Projeto**

O projeto está organizado em três principais componentes:

* **app.py:** Contém a lógica principal da aplicação Flask, incluindo rotas para manipulação de despesas, condições de pagamento e status.
* **model.py:** Contém a classe **AppBD**, que gerencia a criação de tabelas no banco de dados e realiza operações de CRUD.
* **templates/index.html:** Arquivo HTML que define a estrutura da página web, incluindo formulários para adicionar e editar despesas, bem como listagens para mostrar despesas, condições de pagamento e status.

**4. Estrutura do Banco de Dados**

O banco de dados SQLite possui três tabelas principais:

**4.1. Tabela despesas**

* **Campos:**
  + **id (INTEGER, PRIMARY KEY)**: Identificador único para cada despesa.
  + **descricao (TEXT)**: Descrição da despesa.
  + **data (DATETIME)**: Data da despesa.
  + **valor (REAL)**: Valor da despesa.
  + **id\_condicao (INTEGER, FOREIGN KEY)**: Chave estrangeira referenciando a tabela **codicoes**.
  + **id\_status (INTEGER, FOREIGN KEY)**: Chave estrangeira referenciando a tabela **status**.

**4.2. Tabela codicoes**

* **Campos:**
  + **id (INTEGER, PRIMARY KEY)**: Identificador único para cada condição de pagamento.
  + **descricao (TEXT, UNIQUE)**: Descrição da condição de pagamento.

**4.3. Tabela status**

* **Campos:**
  + **id (INTEGER, PRIMARY KEY)**: Identificador único para cada status de despesa.
  + **descricao (TEXT, UNIQUE)**: Descrição do status da despesa.

**5. Funcionalidades Implementadas**

**5.1. Gerenciamento de Despesas**

* **Adição de Despesas:** Implementada na rota **/gerenciar\_despesas** usando o método POST, permitindo a inserção de novas despesas no banco de dados.
* **Edição de Despesas:** Implementada na rota **/editar\_despesa/<int:despesa\_id>**, permitindo a atualização dos detalhes de uma despesa específica.
* **Exclusão de Despesas:** Implementada na rota **/gerenciar\_despesas** usando o método POST, permitindo a exclusão de despesas do banco de dados com base no seu ID.

**5.2. Gerenciamento de Condições de Pagamento e Status**

* **Adição e Exclusão de Condições e Status:** Implementadas nas rotas **/gerenciar\_condicoes** e **/gerenciar\_status** usando métodos POST, permitindo a adição e exclusão de condições de pagamento e status respectivamente.

## **5.3. Implementação das APIs**

### **5.3.1. API de Condições de Pagamento (/api/condicoes)**

* **GET /api/condicoes:** Retorna todas as condições de pagamento em formato JSON.
* **POST /api/condicoes:** Adiciona uma nova condição de pagamento ao banco de dados. Requer um corpo JSON contendo a descrição da nova condição.
* **PUT /api/condicoes/<id>:** Atualiza uma condição de pagamento existente com base no ID fornecido. Requer um corpo JSON contendo a descrição atualizada.
* **DELETE /api/condicoes/<id>:** Deleta uma condição de pagamento com base no ID fornecido.

### **5.3.2. API de Status (/api/status)**

* **GET /api/status:** Retorna todos os status de despesa em formato JSON.
* **POST /api/status:** Adiciona um novo status ao banco de dados. Requer um corpo JSON contendo a descrição do novo status.
* **PUT /api/status/<id>:** Atualiza um status existente com base no ID fornecido. Requer um corpo JSON contendo a descrição atualizada.
* **DELETE /api/status/<id>:** Deleta um status com base no ID fornecido.

### **5.3.3. API de Despesas (/api/despesas)**

* **GET /api/despesas:** Retorna todas as despesas em formato JSON, incluindo detalhes sobre condição de pagamento e status.
* **POST /api/despesas:** Adiciona uma nova despesa ao banco de dados. Requer um corpo JSON contendo descrição, data, valor, ID de condição e ID de status.
* **PUT /api/despesas/<id>:** Atualiza uma despesa existente com base no ID fornecido. Requer um corpo JSON contendo descrição, data, valor, ID de condição e ID de status atualizados.
* **DELETE /api/despesas/<id>:** Deleta uma despesa com base no ID fornecido.

### **5.3.4. Gerenciamento de Erros**

* Para todas as APIs, em caso de falha na execução de uma operação (por exemplo, tentativa de deletar uma condição de pagamento que está sendo usada por uma despesa), será retornado um código de status HTTP 400 (Bad Request) junto com uma mensagem explicativa no corpo da resposta JSON.

**6. Funções da Classe AppBD**

A classe **AppBD** em **model.py** oferece métodos para interação com o banco de dados:

* **\_\_init\_\_(self):** Inicializa a conexão com o banco de dados SQLite e cria as tabelas se não existirem.
* **inserirCondicoesPadrao(self, descricao\_condicoes):** Insere condições de pagamento padrão na tabela **codicoes**.
* **inserirStatusPadrao(self, descricao\_status):** Insere status padrão na tabela **status**.
* **inserirDespesa(self, descricao, data, valor, id\_condicao, id\_status):** Insere uma nova despesa na tabela **despesas**.
* **atualizarDespesa(self, despesa\_id, descricao, data, valor, id\_condicao, id\_status):** Atualiza os detalhes de uma despesa existente na tabela **despesas**.
* **deletarDespesa(self, despesa\_id):** Deleta uma despesa da tabela **despesas**.
* **inserirCondicao(self, descricao):** Insere uma nova condição de pagamento na tabela **codicoes**.
* **inserirStatus(self, descricao):** Insere um novo status na tabela **status**.
* **deletarCondicao(self, descricao):** Deleta uma condição de pagamento da tabela **codicoes**.
* **deletarStatus(self, descricao):** Deleta um status da tabela **status**.
* **selecionarDespesas(self):** Retorna todas as despesas da tabela **despesas** com detalhes de condição de pagamento e status.
* **selectCondicoes(self):** Retorna todas as condições de pagamento da tabela **codicoes**.
* **selectStatus(self):** Retorna todos os status da tabela **status**.

**7. Conclusão**

O projeto foi implementado com sucesso, oferecendo um sistema completo para o gerenciamento de despesas pessoais. A estrutura do banco de dados, as funcionalidades implementadas e a interação com as APIs RESTful foram detalhadas neste documento para fornecer uma compreensão abrangente do sistema.

**8. Possíveis Melhorias Futuras**

* **Segurança:** Implementar autenticação de usuários para proteger o acesso às funcionalidades do sistema.
* **Visualização de Dados:** Adicionar gráficos e relatórios para visualização estatística das despesas.
* **Experiência do Usuário:** Aprimorar a interface do usuário para tornar a aplicação mais intuitiva e fácil de usar.

**9. Agradecimento e Reconhecimento**

Gostaríamos de expressar nosso profundo agradecimento à professora Cythia Moreira Maia pelas valiosas aulas e orientações fornecidas durante o curso. Sua dedicação ao ensino e sua paixão pela ciência da computação foram inspiradoras.

As lições aprendidas em suas aulas foram fundamentais para o desenvolvimento deste projeto. Agradecemos por compartilhar seu conhecimento e por nos motivar a alcançar a excelência. Sua influência positiva em nossas vidas acadêmicas e profissionais é inestimável.

**10. Referências**

* **Documentação do Flask:** [https://flask.palletsprojects.com/](https://flask.palletsprojects.com/\)
* **Documentação do SQLite:** https://www.sqlite.org/docs.html
* **Thunder Client para VS Code:** <https://www.thunderclient.io/>