Relatório Técnico

Breve relatório técnico criado ao decorrer do desenvolvimento do projeto.

**Aplicação:**   
Sistema web para compra e venda de produtos, com autenticação, proteção contra ataques CSRF e uso de modais dinâmicos para confirmação de transações.

O html é herdado pela pagina base por meio do Jinja2 que permite a delimitação com {% ... %}

A aplicação foi modularizada em:

## **base.html:**

A navbar é um dos elementos herdados pelas outras paginas. Porém dependendo da autenticação ou não do usuário, a visualização e opções são diferentes. Como no caso do usuário não estar logado, existirá a opção Cadastro e Login.

Essa diferença é possível pelo bloco:

{% if current\_user.is\_authenticated %}

Outro ponto a ser mencionado do base.html, é a configuração das flash messages, que recebem sua category como parâmetro.

{% with messages = get\_flashed\_messages(with\_categories=true)%}

{% if messages%}

{% for message in messages %}

…

{{message[1]}}

“Message[1] é o texto e Message[0] é a categoria.“

**Modelos (models.py)**

Utiliza SQLAlchemy como ORM. Cada tabela é representada por uma classe.  
A senha do usuário é armazenada com hash (via bcrypt) usando @property e @setter para encapsulamento da lógica.

Para incluir os models:

{%include 'includes/modal\_produtos.html' %}

**Formulários (forms.py)**

Usa FlaskForm com validators do WTForms para validação no frontend.  
Validações customizadas com métodos na classe com esse nome: validate\_<campo>() e ValidationError para casos como duplicidade.

Funções de validação devem seguir a nomenclatura validate\_<campo> para que sejam detectadas automaticamente durante o validate\_on\_submit().

Toda instancia de formulario devera ter um submitfield com o tratamento em validate\_on\_submit( ).

|  |
| --- |
| **Rotas (routes.py)** Define os endpoints e interações com frontend/backend. |

**/produtos: lista produtos disponíveis e produtos do usuário (via filtro).**

- Para comprar produto:

Usa modal para confirmar compras.

O botão chama o modal passando o ID do item com data-bs-target.  
O modal contém input com name="compra\_produto" cujo value={{item.nome}} dentro de um forms “POST” que ao confirmado, o request.form.get, no backend, procura o elemento com o name “compra\_produto” (o input já mencionado) e retorna o value (item.nome).

Após isso, o backend localiza o item no banco com base no nome, associa ao ID do usuário autenticado e registra a transação.

- Para vender o produto:

Junto com os produtos Disponiveis para compra, será devolvido os produtos adquiridos pelo usuario por meio de um filtro onde devolve uma lista de itens

Lógica semelhante a compra, porém o id recebido pelo Modal é do dono do item.

Ao confirmar o forms “POST” que possui o input com o name=”venda\_produto” utilizado pelo request.form.get para encontra-lo, é retornado o value do input, que é o “dono.id”.

**- Pasta Includes:**

Possui os modais que serão ativados pelos botões no /produto

**- /cadastro ->**

Carrega o forms e passa para o template

Ficará ouvindo a pagina, no caso de receber um post do forms, ele colhe e salva os dados no banco e redireciona para outra page.

cadastro.html tem o {{form.hidden\_tag()}} que gera campos ocultos como o CSRF token, o flask da erro sem ele.

Durante form.validate\_on\_submit(), o Flask-WTF faz:

Verifica se a requisição é POST.

Verifica se o campo csrf\_token foi enviado.

Verifica se o valor do csrf\_token é válido comparando com o token da sessão.

**- /login -> Carrega forms e loga**

Recebera username e senha para log in, pesquisa o username no banco para depois validar com a senha

**\_\_init\_\_.py**

Importação dos módulos, configuração do app e inicialização

- o login\_manager configura mensagem para quando acesso nao autorizado

**BIBLIOTECAS:**

* wtfforms permite abstração dos formularios por meio de classes no Python permitindo tambem validação no front com o "validators"
* FlaskForm é uma classe base de form do flask\_wtf - extensão flask para wtforms (um complementa o outro)
* SQLAchemy permite abstração do banco na forma de ORM
* FLASK-BCRYPT

Bcrypt permite a criptografia de um valor em hash

* FLASK-LOGIN

Gerencia sessões

- login\_user

- logout-user

- login\_required -> valida se esta logado para efetuar uma transação (usado em /produtos)