🧩 1. Estrutura Geral do Projeto

Organize seu projeto mais ou menos assim:

```
yourapp/
│
├── app.py                # ponto de entrada principal da aplicação
├── config.py             # configurações da aplicação
│
├── extensions/
│   └── __init__.py       # inicialização de extensões (Flask-Login, SQLAlchemy
etc.)
│
├── models/
│   ├── __init__.py
│   ├── user.py           # modelo de usuário
│   └── product.py        # modelo de produto
│
├── controllers/
│   ├── __init__.py
│   ├── user_controller.py # rotas e lógica de controle para usuários
│   └── product_controller.py # rotas e lógica de controle para produtos
│
├── auth/
│   ├── __init__.py
│   ├── routes.py         # rotas de login, logout, registro
│   └── utils.py          # funções auxiliares de autenticação
│
└── templates/
    ├── auth/             # templates de login, registro, logout
    ├── users/            # telas relacionadas a usuários
    └── products/         # telas relacionadas a produtos
```

⚙️ 2. Conceito do MVC no Flask

O Flask por si só é minimalista e não impõe um padrão MVC rígido, mas você pode **simular** o padrão assim:

Model (M)

- Representa suas entidades de negócio (User, Product, etc.).
- Fica no módulo `models/`.
- Define atributos e relacionamentos (por exemplo, `User` tem uma relação *one-to-many* com `Product`).
- Também pode conter métodos de negócio simples (ex: `User.verify_password()`).

View (V)

- São suas páginas HTML (em `templates/`) ou respostas JSON se for uma API.
- São “alimentadas” pelos controllers.

Controller (C)

- Fica no módulo `controllers/`.
- É onde ficam as **rotas e lógica de controle**, conectando as *views* com os *models*.
- Um controller não deve ter lógica de persistência nem de autenticação complexa (isso fica nas camadas adequadas).

🔐 3. Módulo de Autenticação (Auth)

O módulo `auth/` será responsável por:

- Gerenciar login e logout.
- Registro de novos usuários.
- Integração com o **Flask-Login**.

Estrutura lógica:

- `auth/routes.py`: define as rotas (`/login`, `/logout`, `/register`).
- `auth/utils.py`: contém funções auxiliares, como carregamento de usuário para o `Flask-Login` e verificação de senha.

Esse módulo é registrado como um **Blueprint**, e o Flask-Login é inicializado na aplicação principal.

🧠 4. Uso do Flask-Login

Você usará **Flask-Login** para gerenciar sessões e autenticação.

- A extensão deve ser inicializada no módulo `extensions/`, junto com o SQLAlchemy.
- No modelo `User`, implemente os métodos necessários (`is_authenticated`, `get_id`, etc.) — geralmente herdando de `UserMixin`.
- No `auth/utils.py`, defina a função `load_user(user_id)` e a registre com o `LoginManager`.

Em resumo:

- `Flask-Login` vai cuidar da sessão e da proteção de rotas.
- O módulo `auth` vai cuidar das rotas e interações (login/logout/registro).
- O `User` model fica isolado em `models/user.py`.

🗃️ 5. Modelagem das Entidades

Você terá **duas entidades principais**:

1. **User**
    - Campos: `id`, `username`, `email`, `password_hash`
    - Relação: `User` tem muitos `Product`s.
2. **Product**
    - Campos: `id`, `name`, `description`, `price`, `user_id`
    - Relação: `Product` pertence a um `User`.

Essas entidades ficam em `models/` e são conectadas via SQLAlchemy.

🕹️ 6. Controllers

Cada controller é um **Blueprint** separado:

- `controllers/user_controller.py` → CRUD de usuários (exceto login/logout, que ficam em `auth`).
- `controllers/product_controller.py` → CRUD de produtos (apenas acessíveis por usuários logados).

Cada controller:

- Importa seus *models* e *forms* (se usar WTForms).
- Usa o `login_required` do Flask-Login nas rotas que precisam de autenticação.
- Renderiza templates (views) ou retorna JSON (se for uma API).

🧰 7. Extensões e Inicialização

No módulo `extensions/`, você centraliza a criação das extensões:

- `db` (SQLAlchemy)
- `login_manager` (Flask-Login)

No `app.py`:

- Cria a instância do Flask.
- Carrega as configurações (`config.py`).
- Inicializa as extensões.
- Registra os *Blueprints* (auth, user_controller, product_controller).
- Cria o banco se necessário.

🌐 8. Views e Templates

- Use pastas separadas dentro de `templates/` para manter organização por módulo.
- Por exemplo:
  - `templates/auth/login.html`
  - `templates/products/list.html`
  - `templates/users/profile.html`

Esses templates são chamados pelos *controllers* via `render_template()`.

☑️ 9. Regras Gerais de Organização

- **Nunca misture lógica de negócio nos controllers.**
  - Controllers coordenam.
  - Models armazenam e manipulam dados.
  - Views exibem resultados.
- **Centralize autenticação** no módulo `auth`.

- **Evite dependências circulares** — use imports relativos e inicie extensões em `extensions/`.
- **Use Blueprints** para modularizar as rotas.