

RELATÓRIO DE ANÁLISE CRÍTICA DO PROJETO - Sistema de Moeda Estudantil (Grupo 01)

Marcus Vinícius Vieira Alves
Filipe Bastos Marra
(Grupo 03)

1. ARQUITETURA DO PROJETO

O projeto segue uma **arquitetura MVC modularizada**, própria dos projetos Django, dividida em:

- **MVC (Model–View–Controller)**

No Django, o padrão é uma variação chamada **MTV (Model–Template–View)**:

Padrão	Equivalente Django	Função
Model	<code>models.py</code>	Estrutura e lógica dos dados
View	<code>views.py</code>	Regras de negócio e retorno HTTP
Controller	URLs + Views	Fluxo da aplicação

Template `templates/.html` Interface com o usuário

Isso está refletido claramente nas apps:

- `accounts` → regras de autenticação, modelos e telas
- `wallet` → lógica de saldo e resgates
- `transactions` → lógica de envio de moedas
- `catalog` → catálogo de vantagens

- Arquitetura Modular (apps independentes)

O projeto usa **várias aplicações internas** dentro da pasta `apps/`, cada uma responsável por um contexto específico (arquitetura por domínios):

- `accounts` → Autenticação e usuários
- `institutions` → Instituições acadêmicas
- `catalog` → Catálogo de vantagens
- `wallet` → Carteiras
- `transactions` → Transações entre usuários
- `partners` → Empresas parceiras
- `notifications` → envio de e-mails
- `core` → utilidades e middlewares

Essa separação gera:

- baixo acoplamento
- alta coesão
- maior facilidade de manutenção e expansão

- Arquitetura Web Tradicional (Server-side Rendering)

O uso de:

- `templates/`
- HTML renderizado no servidor
- CSS / JS estático

Mostra que o projeto segue uma **arquitetura tradicional de servidor**, não SPA, onde:

- o Django renderiza as páginas completas
- não existe framework frontend (React/Vue/etc)
- foco em simplicidade e velocidade

- Arquitetura em Camadas dentro de cada app

Cada app geralmente possui:

- **models** → camada de dados
- **views** → camada de lógica
- **templates** → camada de apresentação
- **forms / services / utils** (se existirem) → camada de serviços

Esse é um padrão clássico de arquitetura em camadas.

1.1. TECNOLOGIAS UTILIZADAS

- Backend

a. Django (Python)

Framework principal do projeto.

Recursos Django utilizados:

- ORM
- Sistema de templates
- Rotas (urls.py)
- Administração (provavelmente)
- Middlewares
- Signals (possivelmente)
- Autenticação padrão do Django

b. Django Templates

Para renderizar HTML no servidor.

c. Python 3

Linguagem base, inferida pelo Django e arquivos .py.

d. SQLite

Banco padrão de desenvolvimento (db.sqlite3).

- Frontend

O projeto utiliza frontend clássico:

- HTML (em templates)
- CSS (em static/css)
- JavaScript simples (em static/js)
- Bootstrap (provável, pela organização — você pode confirmar no HTML)
- HTML para e-mails (templates/emails)

- Infraestrutura / DevOps

a. Shell Scripts (Linux/Mac)

Pasta `scripts/` contém:

- `dev_bootstrap.sh`
- `run_local.sh`
- `seed_demo.py`

Esses scripts ajudam no setup e inicialização.

Não há Docker — isso é **um ponto de melhoria**.

b. Virtualenv (provável)

Recomendado para instalação, porque existe `requirements.txt`.

- Persistência de mídia

a. Django Media Storage

Em:

`media/vantagens/`

Armazena arquivos enviados pelos usuários.

- Documentação

Na pasta `docs/`:

- Diagrama de Classes
- Diagrama de Casos de Uso

- Diagrama de Componentes
- Histórias de Usuário

Isso indica que o projeto utiliza:

- ✓ UML
 - ✓ Modelagem de requisitos
 - ✓ User Stories
-

2. Organização no GitHub

O grupo optou por usar o mesmo repositório para todas as atividades ao longo do semestre de laboratório. Podemos assim pensar que isso poderia trazer uma certa desorganização, certo? Mas muito pelo contrário, o readme inicial está muito explicativo, falando sobre todos os exercícios durante o semestre.

Ao acessar a atividade que estamos analisando em questão, vemos que ela também possui um readme contando um pouco mais especificamente sobre o projeto, de como ele foi feito e sua organização, contendo prints da telas de dentro do software. Porém faltou um passo a passo de como configurar o ambiente.

3. Dificuldade para configuração do ambiente

Tendo em conta a falta de um passo a passo para sabermos como configurar o ambiente, não fui capaz de iniciar o sistema em minha máquina. Porém, o grupo disponibilizou o ambiente online no render (site onde foi feito o deploy).

4. Sugestão de melhorias

Com base na análise da arquitetura, organização do projeto e padrões utilizados, identificamos poucos pontos que podem ser aprimorados para melhorar a parte documental do projeto.

Inicialmente, talvez um passo a passo para iniciar o projeto localmente seria bom para ajudar o usuário a configurar o seu próprio ambiente e testar a aplicação.

Além disso, eu recomendaria o uso de algum linter, que ajudaria na organização do código e ajudaria novos participantes do repositório a seguir padrões do código.

Fora isso, a equipe entregou um projeto muito bem organizado documentalmente, e bem intuitivo para usuários que acessam o repositório poderem entender um pouco mais do que está acontecendo ali.