1 Arquitetura do Software

1.1 Arquitetura Aplicada: Modelo MVC

A arquitetura adotada para o desenvolvimento do *UserHistory Hub* segue o padrão arquitetural **MVC (Model–View–Controller)**, amplamente utilizado em sistemas que exigem organização modular, facilidade de manutenção e escalabilidade. A aplicação dessa arquitetura garante uma separação clara entre a lógica de negócio, as estruturas de dados e as interfaces de interação, promovendo um desenvolvimento mais consistente e extensível.

1.1.1 Visão Geral do MVC

O padrão MVC é composto por três componentes principais:

- **Model (Modelo):** Responsável pela representação dos dados e das regras de negócio. Nesta camada estão as estruturas que mapeiam as entidades do sistema, como *Usuários*, *Projetos* e *Mensagens*. O modelo se comunica diretamente com o banco de dados e implementa validações, transformações e regras de integridade.
- **View (Visão):** Corresponde às interfaces apresentadas ao usuário. No projeto, essa camada é implementada através de páginas dinâmicas desenvolvidas em **Vue.js**, oferecendo navegação intuitiva e reatividade na apresentação das informações. São exemplos de *Views*: a tela de login, a página de dashboard e o painel de projetos.
- **Controller (Controlador):** Atua como intermediário entre a *View* e o *Model*. Responsável por processar requerimentos do usuário, manipular os dados recebidos e chamar os modelos apropriados. No projeto, essa camada é implementada como uma **API REST** construída com **Flask**, contendo rotas como `/login`, `/projeto/[id]`, `/mensagem/insert`, entre outras.

1.2 Aplicação do MVC no UserHistory Hub

O fluxo de execução do sistema baseado na arquitetura MVC funciona da seguinte forma:

1. O usuário interage com a *View* (interface Vue.js), realizando ações como logar, visualizar projetos ou enviar mensagens.
2. A *View* envia requisições HTTP à camada *Controller* (API Flask), que recebe, interpreta e encaminha a ação solicitada.
3. O *Controller* aciona o *Model*, que realiza operações como consultas e inserções no banco de dados PostgreSQL.

4. O resultado retornado pelo *Model* é processado pelo *Controller* e enviado de volta à *View*, que o transforma em elementos visuais.

Esse fluxo assegura que as responsabilidades permaneçam isoladas e bem definidas, permitindo:

- **Escalabilidade:** novas funcionalidades podem ser adicionadas sem interferir diretamente nas demais camadas.
- **Manutenibilidade:** alterações no banco de dados ou nas regras de negócio não exigem mudanças nas interfaces.
- **Reutilização:** componentes do *Model* e do *Controller* podem ser reaproveitados em novas interfaces.
- **Flexibilidade:** possibilita substituição ou evolução da camada de *View* (por exemplo, trocar Vue.js por React) sem reescrever toda a aplicação.

1.3 Estrutura das Camadas no Projeto

A organização final do sistema seguindo MVC pode ser resumida como:

- **Model:** Representado pelas entidades persistidas no banco e pelas funções de lógica de negócio; estruturas como `usuarios`, `projetos` e `mensagens`.
- **View:** Composta pela interface construída em Vue.js, responsável por renderizar listas de projetos, formulários de mensagens, dashboards e a interface de login.
- **Controller:** Representado pela API Flask, com rotas REST que realizam CRUD completo sobre projetos, usuários e mensagens.

1.4 Benefícios da Arquitetura no Projeto

A adoção do padrão MVC trouxe benefícios diretos ao desenvolvimento do *User-History Hub*, tais como:

- Organização mais clara dos componentes do sistema.
- Redução de dependências entre front-end e back-end.
- Maior segurança, já que regras de negócio não ficam expostas na interface.
- Facilidade para testes unitários e validação de módulos individuais.
- Base sólida para expansões futuras, como adição de relatórios avançados, autenticação via OAuth, dashboards interativos, etc.