Sistema de Gestão Ágil

Artefato 5 - Descrição da Arquitetura

1. Objetivos da Arquitetura

Esta seção descreve os objetivos de negócio e qualidade que a arquitetura deve suportar, com base nos requisitos não funcionais (Artefato 3).

A arquitetura deve priorizar:

- **Manutenibilidade:** O código deve ser limpo, modular e fácil de modificar, permitindo que novas funcionalidades (como novos artefatos Scrum) sejam adicionadas no futuro.
- **Segurança:** A arquitetura deve garantir que os dados dos usuários sejam protegidos (via *hashing* de senha) e que as regras de negócio baseadas em papéis (RBAC) sejam estritamente aplicadas.
- **Desempenho:** O sistema deve ser rápido, com tempos de resposta baixos para operações de CRUD (Criar, Ler, Atualizar, Excluir) nos backlogs, conforme definido no Artefato 3.
- **Facilidade de Desenvolvimento:** A arquitetura deve aproveitar ferramentas que geram documentação automaticamente e simplificam a validação de dados.

2. Decisões, Restrições e Justificativas

Esta seção detalha as escolhas tecnológicas e as restrições impostas ao projeto.

Categoria	Decisão / Restrição	Justificativa
Restrição (Backend)	**Python (linguagem)**	Decisão da equipe. É uma linguagem moderna, com vasta biblioteca e forte suporte da comunidade.
Decisão (Framework)	**FastAPI**	Escolhido por sua alta performance, documentação automática (Swagger/ReDoc) e sistema de *Dependency Injection* que facilita testes e código limpo.
Restrição (Frontend)	**HTML puro**	Decisão da equipe para focar na lógica de negócios e na API (backend), mantendo a interface (frontend) simples e funcional.
Decisão (Banco de Dados)	**MySQL**	Um banco de dados relacional robusto, open-source e que suporta tipos de dados avançados, garantindo a integridade dos dados.
Decisão (Persistência)	**SQLAlchemy (ORM)**	Abstrai o banco de dados, permitindo que o código Python interaja com o MySQL de forma segura e eficiente, sem escrever SQL manualmente.
Restrição (Implantação)	**Implantação Local**	O sistema será executado localmente, sem a complexidade de serviços de nuvem, conforme definido no Artefato 3.

3. Perspectivas da Arquitetura

Para descrever a arquitetura, adotamos a **Perspectiva Lógica** em um modelo de **3 Camadas (3-Tier)**.

3.1. Camada de Apresentação (Frontend)

- **Tecnologia:** HTML e CSS.
- **Responsabilidade:** Renderizar as informações recebidas do backend e capturar a entrada do usuário. Esta camada não contém regras de negócio. Ela apenas faz requisições HTTP (ex: `fetch`) para a Camada de Aplicação.

3.2. Camada de Aplicação (Backend / API)

- **Tecnologia:** Python com FastAPI.
- **Responsabilidade:** Esta é o cérebro do sistema.
  - **Endpoints (API):** Expõe os recursos (ex: `/usuarios`, `/projetos`, `/sprints`) para o frontend.
  - **Regras de Negócio:** Aplica a lógica de segurança (autenticação) e as regras de negócio (ex: "Apenas um Product Owner pode criar uma História de Usuário").
  - **Serviços:** Orquestra a lógica entre os *endpoints* e o banco de dados.

3.3. Camada de Dados (Database)

- **Tecnologia:** MySQL.
- **Responsabilidade:** Armazenar, proteger e gerenciar os dados de forma persistente. O acesso a esta camada é controlado exclusivamente pela Camada de Aplicação.

---

4. Mecanismos de Arquitetura

Mecanismos são soluções para problemas comuns que se repetem no sistema.

- **Comunicação Cliente-Servidor:** Será feita via **API RESTful**. O frontend (HTML/JS) enviará e receberá dados no formato **JSON** para os *endpoints* do FastAPI.
- **Autenticação e Autorização:**
  1. **Autenticação:** Na rota `/login`, o usuário envia email e senha. O FastAPI valida e retorna um **Token JWT (JSON Web Token)**.
  2. **Autorização:** Para acessar rotas protegidas (ex: `/projetos`), o frontend envia o Token JWT no *header* da requisição. O FastAPI decodifica o token, identifica o usuário e seu papel (usando o `id_usuario` e a tabela `USUARIO_PROJETO`) e libera ou nega o acesso.
- **Persistência de Dados (Abstração):** Usaremos o **SQLAlchemy ORM (Object-Relational Mapper)**. Isso significa que as tabelas do MySQL (como `USUARIO`) serão representadas como Classes Python no backend. Isso é uma **abstração** (Critério do Roteiro) que facilita a manutenção.

---

5. Impacto das Ferramentas Usadas

- **FastAPI:** O impacto é altamente positivo. O framework "força" o uso de **Modelos Pydantic** para validação de dados, o que significa que os dados JSON recebidos pela API são automaticamente

validados. Além disso, ele **gera automaticamente a documentação interativa (Swagger UI)**, o que economiza tempo de desenvolvimento e facilita os testes (Artefato 8).

- **SQLAlchemy:** O impacto é a **abstração do banco de dados**. A arquitetura não depende do "MySQL" em si, mas sim do SQLAlchemy. Se um dia quisermos trocar para MySQL, apenas a string de conexão mudaria, e o resto do código da Camada de Aplicação permaneceria o mesmo.

---

6. Suposições e Dependências

- **Suposições:**
  - Assumimos que o usuário terá um ambiente Python configurado para a execução local.
  - Assumimos que o volume de dados será pequeno, não exigindo otimizações complexas de banco de dados (ex: *sharding* ou replicação).
- **Dependências:**
  - O backend depende criticamente do `fastapi`, `uvicorn` (servidor), `sqlalchemy` (ORM) e `mysqlclient` (driver do MySQL).
  - O frontend depende de um navegador moderno que execute JavaScript para fazer as chamadas de API (`fetch`).