Documento de Arquitetura de Software (DAS)
Sistema de Gestão de Artefatos Scrum

Arthur Luiz Lima de Araújo
Arthur da Silva Pereira Bispo
Ana Luísa de Souza Paraguassu
Tauã Valentim de A. M. Frade
João Carlos Gonçalves de Oliveira Filho

11 de novembro de 2025

Conteúdo

1 Introdução

Este documento descreve a arquitetura de software do Sistema de Gestão de Artefatos Scrum (SGAS). O objetivo é fornecer uma visão abrangente dos principais elementos arquiteturais, seus relacionamentos e as decisões que moldaram a estrutura do sistema.

O SGAS é concebido como uma **aplicação web de uso interno (Intranet)**. O sistema será hospedado e executado inteiramente dentro da infraestrutura de rede da empresa, utilizando o banco de dados corporativo. O acesso será restrito aos computadores e usuários autorizados na rede local, não sendo acessível publicamente pela internet.

2 Objetivos de Arquitetura

Esta seção detalha os objetivos primários que a arquitetura do SGAS visa alcançar. Estes objetivos são fundamentais para guiar as decisões de design e garantir que o sistema atenda às expectativas de qualidade e funcionalidade.

- **Escalabilidade:** A arquitetura deve suportar um número crescente de usuários e projetos sem degradação significativa de desempenho.
- **Manutenibilidade:** O código deve ser fácil de entender, modificar e estender, minimizando o esforço para implementar novas funcionalidades ou corrigir defeitos.
- **Confiabilidade:** O sistema deve ser robusto e capaz de operar continuamente, minimizando falhas e garantindo a integridade dos dados.
- **Segurança:** A arquitetura deve proteger os dados do usuário e do projeto contra acessos não autorizados e outras ameaças de segurança.
- **Desempenho:** As operações críticas do sistema devem responder dentro de limites de tempo aceitáveis para proporcionar uma boa experiência ao usuário.
- **Reusabilidade:** Componentes e módulos devem ser projetados para serem reutilizáveis em diferentes partes do sistema ou em futuros projetos.

3 Suposições Relativas à Arquitetura

Esta seção lista as suposições feitas durante a fase de design da arquitetura. A validade dessas suposições é crucial para a integridade da arquitetura.

- A equipe de desenvolvimento possui familiaridade com as tecnologias e frameworks selecionados.
- A comunicação entre o frontend e o backend será predominantemente via RESTful APIs.
- Os usuários terão acesso a uma conexão de **rede interna** estável para utilizar o sistema.

4 Dependências Consideradas na Definição da Arquitetura

As dependências externas e internas que influenciaram a definição da arquitetura são descritas aqui.

- **Requisitos de Negócio:** Os requisitos funcionais e não funcionais definidos no Documento de Requisitos de Software (SRS) são a principal dependência.
- **Tecnologias Existentes:** A escolha de linguagens de programação (Python, TypeScript) e frameworks (Django REST Framework, React, Vite) existentes no mercado.
- **Infraestrutura:** A disponibilidade de servidores internos e rede corporativa para hospedagem e banco de dados.
- **Equipe de Desenvolvimento:** As habilidades e experiência da equipe influenciaram a escolha das tecnologias.

5 Requisitos Relativos à Arquitetura

Esta seção detalha os requisitos específicos que a arquitetura deve satisfazer, muitas vezes derivados dos requisitos não funcionais do sistema.

- A arquitetura deve ser baseada em uma arquitetura em camadas bem definida para desacoplamento. *
- Deve haver uma clara separação de responsabilidades (SoC) entre o frontend e o backend.
- O sistema deve ser capaz de integrar-se com sistemas de controle de versão (e.g., Git).
- A arquitetura deve permitir a fácil adição de novos módulos ou funcionalidades sem impactar significativamente os módulos existentes.
- O banco de dados deve ser relacional e otimizado para consultas complexas.

6 Decisões, Restrições e Justificativas Relativas à Arquitetura

Esta seção documenta as principais decisões arquiteturais tomadas, as restrições que as influenciaram e as justificativas para essas escolhas.

6.1 Decisões Arquiteturais

- **Arquitetura em Camadas (Frontend/Backend):** Optou-se por uma arquitetura cliente-servidor com frontend e backend desacoplados.

- **Backend com Django REST Framework:** Escolhido pela robustez, segurança e rapidez no desenvolvimento de APIs RESTful em Python.
- **Frontend com React e Vite:** Selecionado pela performance, ecossistema rico e facilidade de desenvolvimento de interfaces de usuário reativas.
- **Banco de Dados SQLite:** Escolhido por ser o padrão do Django, facilitando a configuração inicial, o desenvolvimento e a implantação local sem a necessidade de um servidor de banco de dados separado.
- **Autenticação Baseada em Tokens (JWT):** Para segurança e escalabilidade em aplicações distribuídas.

6.2 Restrições Arquiteturais

- **Ambiente de Implantação:** O software deve operar exclusivamente na rede interna da empresa, sem dependência de serviços de nuvem externos ou tecnologias de contêiner.
- **Orçamento:** Restrições orçamentárias podem limitar a escolha de ferramentas e serviços de nuvem.
- **Prazo:** Prazos apertados podem influenciar a escolha de frameworks que aceleram o desenvolvimento.
- **Conhecimento da Equipe:** A familiaridade da equipe com certas tecnologias pode ser uma restrição ou um facilitador.

6.3 Justificativas

As decisões foram tomadas visando um equilíbrio entre rapidez de desenvolvimento, manutenibilidade, escalabilidade e segurança, aproveitando as melhores práticas e ferramentas do mercado.

7 Mecanismos de Arquitetura

Esta seção descreve os mecanismos arquiteturais chave que implementam os requisitos não funcionais e fornecem serviços comuns em todo o sistema.

- **Mecanismo de Autenticação e Autorização:** Gerencia o acesso dos usuários e suas permissões.
- **Mecanismo de Persistência de Dados:** Abstrai a interação com o banco de dados.
- **Mecanismo de Comunicação Inter-componentes:** Define como os diferentes módulos do sistema se comunicam (e.g., APIs REST).
- **Mecanismo de Tratamento de Erros e Logs:** Padroniza a forma como erros são capturados, registrados e reportados.

8 Abstrações Relativas à Arquitetura

As principais abstrações e modelos conceituais que simplificam a complexidade do sistema são detalhadas aqui.

- **Modelos de Domínio:** Representações de entidades de negócio (e.g., 'Project', 'UserStory', 'Task').
- **Serviços de Aplicação:** Camadas que orquestram operações de negócio usando os modelos de domínio.
- **APIs RESTful:** Contratos de interface para comunicação entre frontend e backend.
- **Componentes de UI:** Abstrações para elementos visuais reutilizáveis no frontend.

9 Arquitetura Segundo Determinadas Perspectivas

Esta seção descreve a arquitetura do sistema a partir de diferentes pontos de vista, fornecendo uma compreensão mais completa de sua estrutura e comportamento.

9.1 Visão Lógica

A visão lógica organiza o sistema em módulos e componentes que realizam funções específicas.

- **Módulo de Autenticação:** Gerencia registro, login e perfis de usuário.
- **Módulo de Projetos:** Gerencia a criação, edição e visualização de projetos.
- **Módulo de Backlog:** Gerencia itens de backlog do produto e histórias de usuário.
- **Módulo de Sprints:** Gerencia o planejamento e acompanhamento de sprints.
- **Módulo de Membros:** Gerencia a adição e remoção de membros em projetos.

9.2 Visão de Processos

A visão de processos descreve como os componentes interagem em tempo de execução para realizar as funcionalidades do sistema.

- **Fluxo de Autenticação:** Usuário envia credenciais ao backend, que retorna um token JWT. O frontend armazena o token e o usa para requisições futuras.
- **Fluxo de Criação de Projeto:** Usuário preenche formulário no frontend, que envia dados via API REST para o backend. O backend valida, persiste no DB e retorna sucesso.
- **Fluxo de Gerenciamento de Backlog:** Usuário interage com a interface do backlog, realizando operações (criar, editar, mover) que são traduzidas em chamadas de API para o backend.

9.3 Visão de Implantação

A visão de implantação descreve como o sistema é mapeado para o ambiente físico ou virtual do servidor interno da empresa.

- **Servidor Web:** Serve os arquivos estáticos do frontend (construídos/builds) e atua como proxy reverso para o servidor de aplicação backend, gerenciando requisições e balanceamento de carga, se necessário.
- **Servidor de Aplicação Backend:** Executa a aplicação Django, geralmente gerenciada por um serviço de sistema (como **Systemd** ou **Supervisor**) que garante que o processo esteja sempre ativo.
- **Banco de Dados (SQLite):** O banco de dados é um arquivo único (e.g., 'db.sqlite3') armazenado no sistema de arquivos do servidor, junto com a aplicação backend. Não requer um serviço de banco de dados separado, simplificando a implantação.
- **Ambiente de Hospedagem (Servidor Interno):** O sistema será implantado diretamente no hardware ou máquina virtual (VM) fornecida pelo ambiente interno da empresa. A configuração deve incluir o gerenciamento das dependências de **Python** e **Node.js** (para o frontend) no sistema operacional.

10 Impacto das Ferramentas Usadas na Arquitetura

Esta seção detalha como as ferramentas, frameworks e bibliotecas selecionadas impactam a arquitetura do sistema.

10.1 Backend (Python, Django REST Framework)

- **Estrutura MVC/MVT:** Django impõe uma estrutura Model-View-Template (ou Model-View-Controller para APIs), que naturalmente separa a lógica de negócio, apresentação e persistência de dados.
- **ORM (Object-Relational Mapper):** O ORM do Django abstrai a interação direta com o banco de dados, permitindo que os desenvolvedores trabalhem com objetos Python em vez de SQL. Isso facilita a manutenibilidade e a portabilidade do banco de dados.
- **APIs RESTful:** Django REST Framework (DRF) simplifica a criação de endpoints RESTful, serialização de dados e autenticação, impactando diretamente a forma como o frontend e outros serviços interagem com o backend.
- **Segurança Integrada:** Django oferece recursos de segurança como proteção CSRF, XSS e SQL Injection, que são incorporados à arquitetura, reduzindo a superfície de ataque.

10.2 Frontend (TypeScript, React, Vite)

- **Componentização:** React promove uma arquitetura baseada em componentes, onde a UI é dividida em pequenas partes reutilizáveis e independentes. Isso melhora a manutenibilidade e a reusabilidade do código.

- **Gerenciamento de Estado:** Bibliotecas como Redux e Context API do React impactam como o estado da aplicação é gerenciado e compartilhado entre os componentes, influenciando a complexidade e o fluxo de dados.
- **TypeScript:** A tipagem estática do TypeScript melhora a robustez do código, facilita a detecção de erros em tempo de desenvolvimento e melhora a colaboração em equipes grandes.
- **Vite:** Como um bundler e servidor de desenvolvimento, Vite otimiza o processo de build, acelerando o desenvolvimento do frontend.
- **Bibliotecas de UI (e.g. Tailwind CSS):** Impactam a consistência visual e a velocidade de desenvolvimento da interface do usuário, fornecendo componentes pré-construídos e um sistema de design.

10.3 Banco de Dados (SQLite)

- **Simplicidade de Implantação:** Por ser um banco de dados "serverless"(sem servidor), ele é contido em um único arquivo. Isso elimina a necessidade de configurar, gerenciar e manter um serviço de banco de dados separado, alinhando-se à implantação em servidor interno.
- **Facilidade de Desenvolvimento:** É o padrão do Django, permitindo que o desenvolvimento seja iniciado imediatamente (‘manage.py migrate‘) sem configuração de credenciais ou serviços.
- **Limitações de Concorrência:** O SQLite pode apresentar gargalos de desempenho sob alta concorrência de escrita (múltiplos usuários tentando escrever dados simultaneamente), pois ele bloqueia o banco de dados durante as transações. Para o escopo de uma aplicação de intranet com uso moderado, esta é uma restrição aceitável.

10.4 Outras Ferramentas

- **Sistema de Gerenciamento de Serviço (e.g., Systemd, Supervisor):** Ferramenta essencial para **garantir a alta disponibilidade** da aplicação backend, reiniciando automaticamente o processo em caso de falha e gerenciando logs. Isso impacta a visão de implantação e confiabilidade.
- **Git:** O sistema de controle de versão Git é fundamental para a colaboração da equipe e o gerenciamento do histórico de código, impactando o processo de desenvolvimento e a rastreabilidade das mudanças arquiteturais.
- **Gerenciadores de Pacote (pip, npm):** A instalação e gestão das dependências do Python e do JavaScript serão feitas diretamente no servidor através desses gerenciadores, exigindo atenção ao **isolamento de ambiente** (como o uso de ambientes virtuais Python - venv).