DAS - Documentação de Arquitetura de Sistema

Bem-vindo à documentação de arquitetura do sistema **AcadMap**!

O que é um DAS?

O **DAR** (Documento de Arquitetura de Sistema) é um artefato fundamental no processo de desenvolvimento de sistemas. Ele descreve, de forma estruturada, as principais decisões arquiteturais, componentes, módulos, interações e restrições que orientam a construção e evolução do software.

Objetivos do DAS

- **Registrar decisões arquiteturais** tomadas durante o projeto.
- **Comunicar a arquitetura** para todos os stakeholders (desenvolvedores, clientes, gestores, etc).
- **Facilitar a manutenção e evolução** do sistema, servindo como referência para futuras modificações.
- **Apoiar a análise de riscos** e a identificação de pontos críticos do sistema.

Estrutura desta documentação

Esta documentação está organizada em seções que abordam desde a introdução do sistema, identificação de stakeholders, preocupações, pontos de vista arquiteturais, visões específicas (lógica, desenvolvimento, implantação, processo e cenários), até decisões arquiteturais, governança, ferramentas de apoio, riscos, glossário e referências.

Explore o menu lateral para navegar pelos tópicos!

---

1. Introdução

Este documento descreve a arquitetura do sistema **AcadMap**, abordando suas principais decisões estruturais, visões, preocupações dos stakeholders e justificativas técnicas. Ele está baseado nos princípios básicos da ISO/IEC 42010, com foco em clareza, rastreabilidade e evolução futura do sistema.

1.1 Objetivo

Apresentar a estrutura e organização da solução, documentar decisões arquiteturais e comunicar de forma clara entre todos os envolvidos.

1.2 Escopo

Sistema web para permitir o armazenamento, consulta, classificação e geração de relatórios sobre eventos e periódicos científicos, com base em regras atualizadas da CAPES, substituindo a antiga consulta ao Qualis e eliminando a necessidade de verificações manuais, com front-end React, back-end Spring Boot e banco de dados PostgreSQL.

1.3 Definições

- **AcadMap**: Nome do sistema representado nesta arquitetura. Plataforma para análise de produção científica baseada em periódicos e eventos.
- **MVP** (*Minimum Viable Product*): Versão mínima funcional do sistema, com as funcionalidades essenciais para validar a proposta.
- **API REST**: Interface de comunicação baseada em HTTP com operações sobre recursos, utilizando padrões como GET, POST, PUT e DELETE.
- **Stakeholder (ST)**: Pessoa, grupo ou organização com interesse direto ou indireto na definição, uso ou evolução do sistema. Referenciado por códigos `STxx` (ex: ST01 – Desenvolvedor Backend).
- **Preocupação Arquitetural (P)**: Necessidade ou exigência relevante de um ou mais stakeholders que deve ser considerada na arquitetura. Referenciada por `Pxx` (ex: P01 – Clareza de código).

- **Ponto de Vista (PV)**: Perspectiva sob a qual a arquitetura é descrita, tratando um conjunto de preocupações. Referenciado por `PVxx` (ex: PV01 – Visão Lógica).
- **DAS**: Documento de Arquitetura de Software.
- **COMP-BACK-01**, **SEQU-MTRC-02**, etc.: Códigos de identificação dos diagramas utilizados ao longo do documento, seguindo o padrão `[TIPO]-[OBJETO]-[NÚMERO]`.
- **Controller / Service / Repository / Model**: Camadas da arquitetura backend seguindo o padrão MVC, responsáveis respectivamente por controlar requisições, encapsular regras de negócio, acessar dados e representar entidades de domínio.
- **Pages / Components / Services (frontend)**: Diretórios lógicos do frontend React. `pages` contém telas principais, `components` agrupa componentes reutilizáveis, e `services` abstrai chamadas à API.

---

2. Stakeholders

Stakeholders são indivíduos, grupos ou entidades que possuem interesse direto ou indireto na definição, evolução ou operação do sistema. Identificar claramente esses stakeholders é fundamental para garantir que suas preocupações sejam devidamente consideradas durante a elaboração da arquitetura. Isso contribui para a alocação adequada de responsabilidades, melhora a comunicação entre as partes envolvidas e fortalece o alinhamento entre os objetivos de negócio e as decisões técnicas.

A tabela a seguir apresenta os stakeholders identificados para este projeto, descrevendo seus papéis, responsabilidades e as preocupações arquiteturais que representam.

ID	Nome	Papel	Responsabilidades Principais	Preocupações Relacionadas
ST01	Desenvolvedor Back	Desenvolvimento do backend	Implementar lógica de negócio, integrações, persistência	P01, P06, P07, P09
ST02	Desenvolvedor Front	Desenvolvimento do frontend	Criar interfaces, garantir UX, consumir APIs	P01, P02, P09
ST03	Product Owner	Representante do negócio	Definir backlog, validar valor para o usuário, priorizar entregas	P02, P04, P08
ST04	Analista de QA	Garantia da Qualidade	Verificar cobertura de testes, detectar falhas, validar entregas	P03, P06, P09
ST05	Arquiteto de Software	Modelagem e implantação		

	técnica	Projetar a arquitetura do sistema e do banco de dados, realizar implantação em produção	P01, P05, P06, P07, P10

3. Preocupações Arquiteturais

As preocupações arquiteturais representam aspectos, requisitos e desafios críticos que devem ser tratados durante o desenvolvimento do sistema. Elas são derivadas diretamente das necessidades dos stakeholders e impactam diretamente atributos de qualidade como segurança, manutenibilidade, desempenho, testabilidade e usabilidade. O mapeamento claro dessas preocupações permite que a arquitetura seja construída de forma rastreável, atendendo aos objetivos do projeto e assegurando a satisfação dos stakeholders.

A tabela a seguir apresenta as principais preocupações levantadas, associadas aos stakeholders que as expressam e aos pontos de vista arquiteturais que as endereçam.

ID	Preocupação	Stakeholders Relacionados	Pontos de Vista Relacionados
P01	Clareza e qualidade do código	ST01, ST02, ST05	VP02 (Desenvolvimento)
P02	Experiência do usuário (UX)	ST02, ST03	VP01 (Lógico), VP05 (Uso)
P03	Testabilidade e cobertura de testes	ST04	VP02 (Desenvolvimento), VP04 (Processo)
P04	Tempo de entrega e valor ao usuário final	ST03	VP02 (Desenvolvimento), VP05 (Uso)
P05	Alta disponibilidade e desempenho	ST05	VP01 (Lógico), VP03 (Implantação)
P06	Evolutividade e manutenção futura	ST01, ST04, ST05	VP01 (Lógico), VP02 (Desenvolvimento)
P07	Integração com sistemas externos	ST01, ST05	VP01 (Lógico), VP03 (Implantação)

P08	Suporte a diferentes perfis de usuário	ST03	VP05 (Uso), VP01 (Lógico)
P09	Adoção de boas práticas de engenharia	ST01, ST02, ST04	VP02 (Desenvolvimento)
P10	Segurança e privacidade de dados	ST05	VP01 (Lógico), VP03 (Implantação)

4. Pontos de Vista Arquiteturais

Pontos de vista arquiteturais são descrições específicas de aspectos do sistema sob diferentes perspectivas, com o objetivo de abordar um subconjunto das preocupações levantadas pelos stakeholders. Cada ponto de vista adota uma notação e uma forma de representação adequada à natureza das informações que transmite. A adoção de múltiplos pontos de vista facilita a comunicação entre as partes interessadas, permite avaliações segmentadas da arquitetura e aumenta a rastreabilidade das decisões técnicas.

A tabela a seguir descreve os pontos de vista adotados neste projeto, sua finalidade, artefatos utilizados e as principais preocupações que cada um trata.

ID	Ponto de Vista	Objetivo	Notação / Artefato Utilizado	Preocupações Tratadas
PV01	Lógico	Modelar os módulos, componentes e suas responsabilidades	Diagrama de Componentes (UML)	P02, P05, P06, P07, P08, P10
PV02	Desenvolvimento	Representar a organização do código e boas práticas adotadas	Diagrama de pacotes (UML), árvore de arquivos	P01, P03, P04, P06, P09
PV03	Implantação	Modelar a infraestrutura de execução e distribuição do sistema	Diagrama de Deploy (UML)	P05, P07, P10

PV05	Uso	Ilustrar a experiência do usuário e seus fluxos de interação	Cenários, fluxos, protótipos (Figma)	P02, P04, P08

5.1 Visão Lógica

A Visão Lógica descreve a estrutura interna do sistema AcadMap, detalhando os principais pacotes, camadas e dependências entre os componentes de software. Esta visão está organizada em três blocos tecnológicos fundamentais: o frontend, desenvolvido com React e empacotado via Vite; o backend, implementado em Java utilizando o framework Spring Boot sob o padrão MVC (Model-View-Controller); e o banco de dados, estruturado em PostgreSQL.

Cada camada foi desenhada com foco em boas práticas de engenharia de software, promovendo a separação clara de responsabilidades, a reutilização de código, a legibilidade e a facilidade de manutenção.

Os diagramas a seguir ilustram essas estruturas e seus relacionamentos internos e externos, permitindo uma visualização clara da organização lógica do sistema.

5.1.1 Visão Geral do Sistema

O acesso ao sistema se inicia no navegador do usuário, que interage com o frontend da aplicação via protocolo HTTPS. Esse frontend é uma aplicação desenvolvida em React e hospedada em um servidor web Apache, o qual está exposto à Internet. A camada de frontend é responsável por renderizar a interface do usuário e encaminhar as requisições para o backend.

As requisições de dados feitas pelo frontend são encaminhadas ao backend, uma aplicação Java Spring Boot, que atua como servidor de aplicação. A comunicação entre o frontend e o backend se dá via HTTP interno, utilizando uma API REST baseada em JSON.

O backend, por sua vez, realiza operações de leitura e escrita em um banco de dados PostgreSQL, hospedado em um servidor dedicado de banco de dados. Essa comunicação utiliza a especificação JPA (Java Persistence API), implementada com o JDBC, por meio da porta padrão 5432.

Diagrama de Componentes de Alto Nível - COMP-GERAL-01

```
graph TD

  subgraph "Sistema AcadMap"

    subgraph "Frontend"
      FPages[Pages]
      FComponents[Components]
      FServices[FrontendServices]
    end

    subgraph "Backend"
      BController[Controllers]
      BService[BackendServices]
      BRepository[Repository]
      BModel[Models]
    end

    DB[(PostgreSQL)]

    FPages --> FComponents
    FPages --> FServices
    FServices --> BController
    BController --> BService
    BService --> BRepository
    BService --> BModel
    BRepository --> DB

  end
```

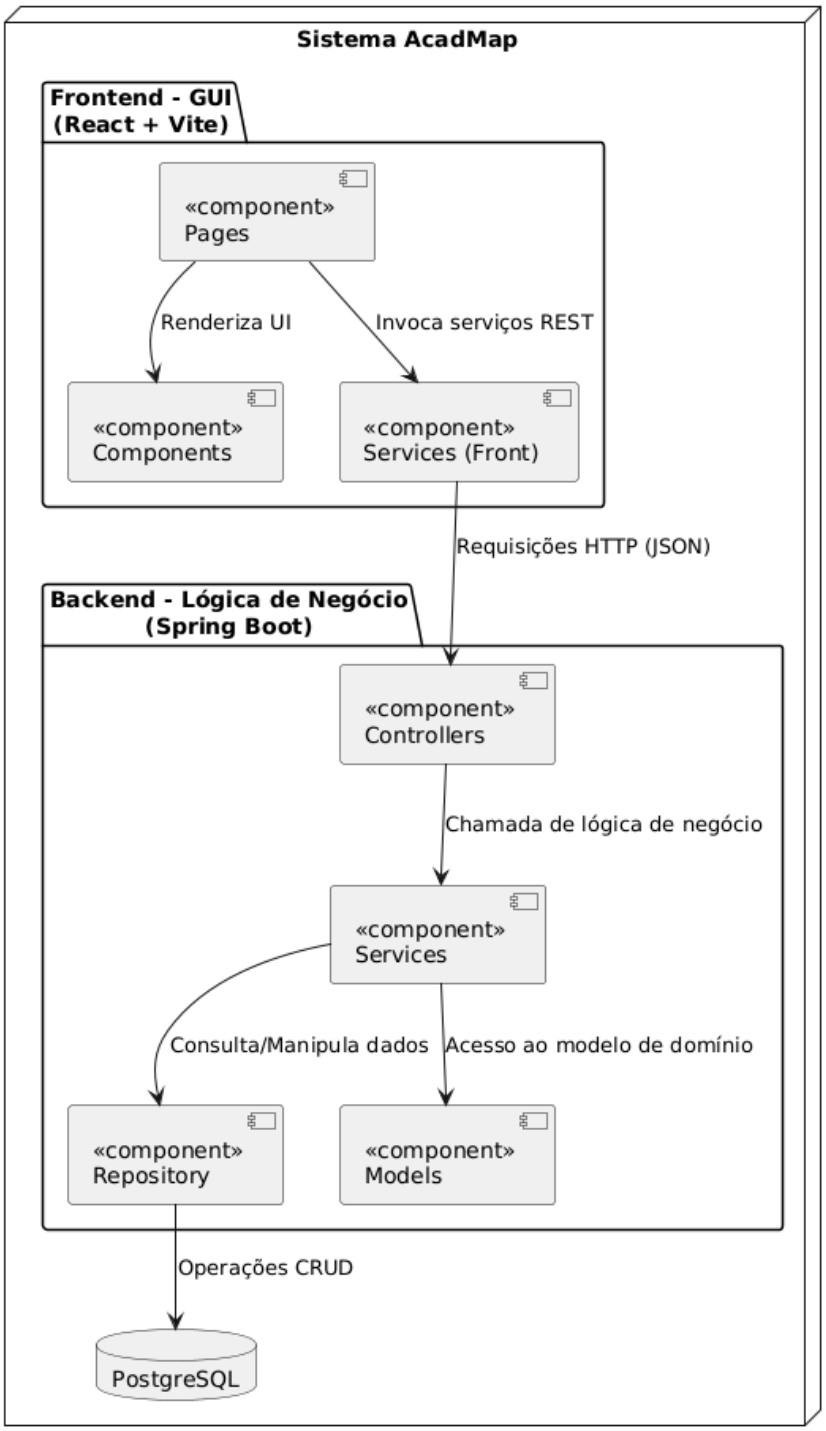
Clique nos blocos abaixo para visualizar versões em outros formatos:

Versão PlantUML

```
@startuml
title COMP-GERAL-01 – Diagrama de Componentes do Sistema AcadMap (Visão de Alto
Nível)

' Container principal
node "Sistema AcadMap" {

  package "Frontend - GUI\n(React + Vite)" {
    [Pages <<component>>] as FPages
    [Components <<component>>] as FComponents
    [Services <<component>>] as FServices
  }

  package "Backend - Lógica de Negócio\n(Spring Boot)" {
    [Controllers <<component>>] as BController
    [Services <<component>>] as BService
    [Repository <<component>>] as BRepository
    [Models <<component>>] as BModel
  }

  database "PostgreSQL" as DB


  ' Interações internas - frontend
  FPages --> FComponents : Renderiza UI
  FPages --> FServices : Invoca serviços REST

  ' Comunicação frontend → backend (API REST)
  FServices --> BController : Requisições HTTP (JSON)

  ' Backend interno
  BController --> BService : Chamada de lógica de negócio
  BService --> BRepository : Consulta/Manipula dados
  BService --> BModel : Acesso ao modelo de domínio
  BRepository --> DB : Operações CRUD
}
@enduml
```

Versão .png

COMP-GERAL-01 - Diagrama de Componentes do Sistema AcadMap (Visão de Alto Nível)



5.1.2 Backend (Spring Boot - MVC)

A camada backend adota uma arquitetura monolítica com padrão **MVC (Model-View-Controller)**, sem modularização por domínio. O código é organizado em pacotes que refletem as camadas de controle (Controller), serviço (Service), persistência (Repository) e modelo de domínio (Model).

Essa organização visa garantir clareza e manutenção simples, permitindo o crescimento gradual do sistema. Cada entidade do domínio é representada por um modelo, e manipulada por seus respectivos controladores, serviços e repositórios.

Diagrama de Componentes do Backend - COMP-BACK-01

> **Nota**
>
> Diagrama desenvolvido antes do recebimento dos insumos necessários para uma versão concreta

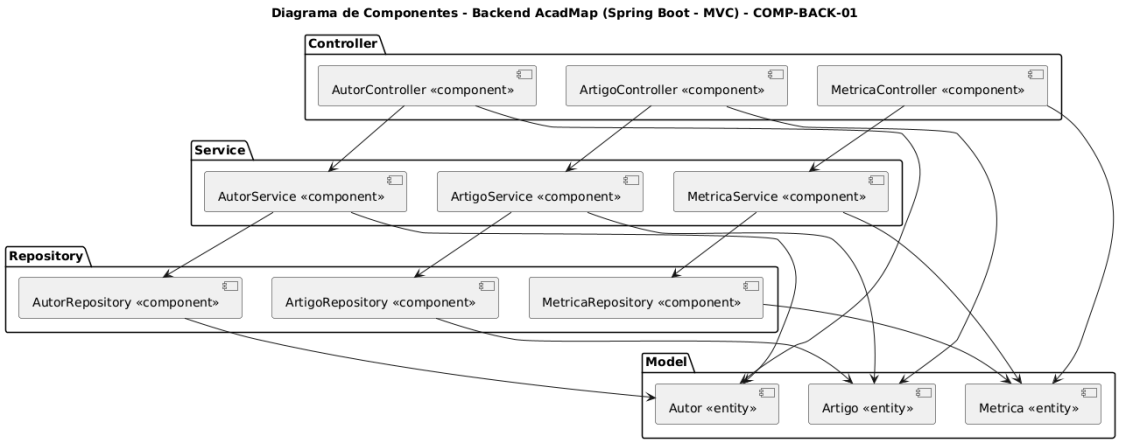
```
graph TD
  subgraph controller
    A[AutorController]
    B[ArtigoController]
    C[MetricaController]
  end

  subgraph service
    D[AutorService]
    E[ArtigoService]
    F[MetricaService]
  end

  subgraph repository
    G[AutorRepository]
    H[ArtigoRepository]
    I[MetricaRepository]
  end

  subgraph model
    J[Autor]
    K[Artigo]
    L[Metrica]
  end

  A --> D
  B --> E
  C --> F

  D --> G
  D --> J
  E --> H
  E --> K
  F --> I
  F --> L

  A --> J
  B --> K
  C --> L

  G --> J
  H --> K
  I --> L
```

Clique nos blocos abaixo para visualizar versões em outros formatos:

Versão PlantUML

```
@startuml
title Diagrama de Componentes - Backend AcadMap (Spring Boot - MVC) - COMP-BACK-01

package "Controller" {
  [AutorController <<component>>] as AC
  [ArtigoController <<component>>] as ARC
  [MetricaController <<component>>] as MC
}

package "Service" {
  [AutorService <<component>>] as AS
  [ArtigoService <<component>>] as ARS
  [MetricaService <<component>>] as MS
}

package "Repository" {
  [AutorRepository <<component>>] as AR
  [ArtigoRepository <<component>>] as ARR
  [MetricaRepository <<component>>] as MR
}

package "Model" {
  [Autor <<entity>>] as A
  [Artigo <<entity>>] as ART
  [Metrica <<entity>>] as M
}

AC --> AS
ARC --> ARS
MC --> MS

AS --> AR
AS --> A
ARS --> ARR
ARS --> ART
MS --> MR
MS --> M

AC --> A
ARC --> ART
MC --> M

AR --> A
ARR --> ART
MR --> M
@enduml
```

Versão .png



5.1.3 Frontend (React + Vite)

A camada frontend é implementada com React e Vite, utilizando o paradigma de componentização funcional. A aplicação é organizada em três principais diretórios lógicos:

- `pages/` : Define as páginas da aplicação vinculadas às rotas principais.
- `components/` : Contém componentes reutilizáveis que compõem visualmente as páginas.
- `services/` : Abstrai a comunicação com a API REST do backend, utilizando axios ou fetch.

Essa organização permite alta reutilização, facilidade de testes e separação de responsabilidades. O fluxo de dados se baseia em props e hooks do React.

Diagrama de Componentes do Frontend - COMP-FRONT-01

Nota

Diagrama desenvolvido antes do recebimento dos insumos necessários para uma versão concreta

```
graph TD
  subgraph pages
    A[DashboardPage]
    B[RelatoriosPage]
    C[AutoresPage]
  end

  subgraph components
    D[CardIndicador]
    E[TabelaArtigos]
    F[FormAutor]
  end

  subgraph services
    G[api.js]
    H[artigoService.js]
    I[autorService.js]
  end

  A --> D
  B --> E
  C --> F

  A --> G
  B --> H
  C --> I

  E --> H
  F --> I
```

Clique nos blocos abaixo para visualizar versões em outros formatos:

Versão PlantUML

```
@startuml
title COMP-FRONT-01 – Diagrama de Componentes do Frontend (React + Vite)

package "Pages" {
  [DashboardPage <<component>>] as DP
  [RelatoriosPage <<component>>] as RP
  [AutoresPage <<component>>] as AP
}

package "Components" {
  [CardIndicador <<component>>] as CI
  [TabelaArtigos <<component>>] as TA
  [FormAutor <<component>>] as FA
}

package "Services" {
  [api.js <<component>>] as API
  [artigoService.js <<component>>] as ARTAPI
  [autorService.js <<component>>] as AUTAPI
}

' Páginas usam componentes
DP --> CI
RP --> TA
AP --> FA

' Páginas consomem serviços
DP --> API
RP --> ARTAPI
AP --> AUTAPI

' Componentes também podem consumir serviços
TA --> ARTAPI
FA --> AUTAPI

@enduml
```

Versão .png

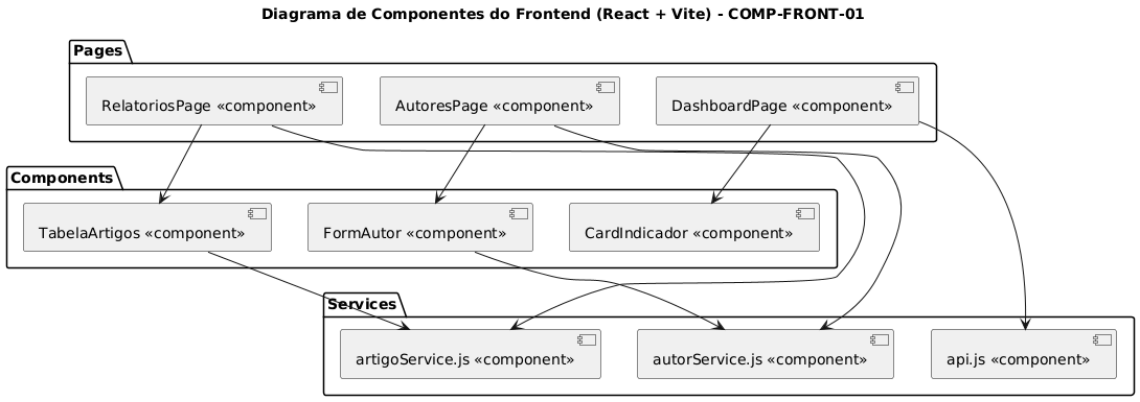


Diagrama de Componentes do Frontend (React + Vite) - COMP-FRONT-01
Pages
RelatoriosPage «component»
AutoresPage «component»
DashboardPage «component»
Components
TabelaArtigos «component»
FormAutor «component»
CardIndicador «component»
Services
artigoService.js «component»
autorService.js «component»
api.js «component»

5.2 Visão de Desenvolvimento

A Visão de Desenvolvimento descreve como o sistema AcadMap está estruturado em termos de diretórios, pacotes e organização do código-fonte. O sistema é composto por dois grandes blocos tecnológicos:

- Backend monolítico com Java + Spring Boot
- Frontend com React + Vite

Ambas as partes são integradas via API REST e armazenam dados em um banco PostgreSQL, conforme representado na Visão Lógica.

5.2.1 Estrutura de Repositórios

O código e a documentação do sistema estão hospedados no GitHub:

1. **Repositório de Desenvolvimento**
2. **Repositório de Documentação**

Todos os repositórios seguem boas práticas de versionamento (`main` , `develop`), commits semânticos (conventional commits), e segue o Git Trunk Flow, com CI/CD configurado via GitHub Actions.

Cada repositório possui:

- Arquivo `README.md` com instruções de uso e contribuição
- Padrão de branches (`feature/*` , `fix/*` , `hotfix/*`)
- Pull Requests com revisão obrigatória

Modelo de Estrutura de Arquivos

```
.
├── backend
│   ├── Dockerfile
│   ├── src
│   │   └── main
│   │       ├── java
│   │       │   └── com
│   │       │       └── acadmap
│   │       │           ├── controller
│   │       │           ├── model
│   │       │           ├── repository
│   │       │           └── service
│   │       └── resources
│   └── tomcat
├── db
├── docker-compose.yml
├── .env
├── frontend
│   ├── Dockerfile
│   ├── public
│   └── src
│       ├── components
│       ├── pages
│       └── services
├── .github
│   └── workflows
│       └── ci.yml
└── README.md
```

5.2.2 Pipeline de CI/CD (RESUMO)

5.2.3 Gerência de Desenvolvimento e Tarefas

O projeto utiliza o recurso **GitHub Projects** para organização e acompanhamento das atividades de desenvolvimento, com os seguintes objetivos:

- Planejamento de entregas por sprint ou fase
- Priorização de funcionalidades e correções
- Acompanhamento do progresso por coluna (Kanban)
- Integração direta com Pull Requests e Issues

Acesso o quadro aqui

```
gitGraph
   commit
   commit
   branch develop
   checkout develop
   commit
   commit
   checkout main
   merge develop
   commit
   commit
```

Visão de Implantação

A Visão de Implantação descreve como o sistema AcadMap é fisicamente distribuído no ambiente de execução, com foco na infraestrutura de contêineres Docker. Essa visão é fundamental para compreender o ambiente operacional do sistema, incluindo a alocação de componentes, fluxos de comunicação, protocolos utilizados e fronteiras de segurança.

O diagrama a seguir apresenta uma visão de alto nível da implantação do sistema em um cenário típico de produção, com os seguintes elementos principais:

- Usuário final, que acessa o sistema por meio de um navegador web;
- Fronteira de rede, que separa o ambiente da internet pública da rede interna de containers;
- Container Apache HTTP Server, responsável por servir os arquivos estáticos do frontend (React + Vite);
- Container Backend (Spring Boot), que implementa a lógica de negócio e expõe as APIs REST em JSON;
- Container Banco de Dados (PostgreSQL), que armazena persistentemente os dados da aplicação;
- Toda a comunicação entre os containers ocorre dentro de uma rede Docker interna (bridge), utilizando protocolos padronizados como HTTPS, HTTP e JDBC.

Esse modelo garante isolamento entre serviços, facilita o escalonamento e simplifica o processo de implantação e manutenção por meio de orquestração baseada em contêineres.

Diagrama de Implantação - DEP-01

```
graph TD
    user["Usuário (Navegador Web)"]

    %% Rede Docker Interna
    subgraph "acadmap-net (intranet)"

        subgraph "Container do Frontend"
            apache["Apache"]

            frontend["Aplicação React (Vite)"]
        end

        subgraph "Container do Backend"
            backend["Aplicação Spring Boot"]
            api["API REST (Java)"]
        end

        subgraph "Container do Banco de Dados"
            db["PostgreSQL"]
        end

    end

    %% Conexões
    user -->|internet - https| apache
    apache -->|HTTP - porta 8080| backend
    backend --> api
    backend -->|JDBC - porta 5432| db
```

Clique nos blocos abaixo para visualizar versões em outros formatos:

Versão PlantUML

```
@startuml
title DEP-01 – Diagrama de Implantação do Sistema AcadMap
actor user as "Usuário"
node browser as "Navegador Web"
' Fronteira entre internet pública e rede interna Docker
boundary "Fronteira de Rede\n(Pública > Interna)" as internet
' Rede interna - Docker
node "Rede Docker Interna (acadmap-net)" as docker_net {
node "Container Apache HTTP Server" as apache {
port p_https
artifact "Frontend (JS + React)" as front
}
node "Container Backend Spring Boot" as backend {
artifact "API REST (Java)" as api
}
node "Container Banco de Dados" as db_container {
database "PostgreSQL" as db
}
}
' Ligações com a fronteira (com indicação clara do ponto de transição)
user --> browser
browser --> internet : HTTPS
internet --> p_https
' Conexões internas da rede Docker
apache --> backend : HTTP (porta 8080)
backend --> db : JDBC (porta 5432)
@enduml
```

Versão .png

▼ Details

5.4 Visão de Processo

5.5 Visão de Cenários

6. Decisões Arquiteturais

7. Governança de Arquitetura

8. Ferramentas de Apoios à Arquitetura

9. Riscos e Mitigações

Glossário

Referências