

DAS - Documento de Arquitetura de Sistema

Bem-vindo ao documento de arquitetura do sistema **AcadMap**!

O que é um DAS?

O **DAS** (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.

Equipes

Gestão de Projeto



GESTÃO DE PROJETO

Arquitetura



ARQUITETURA

Requisitos



REQUISITOS

Desenvolvimento



DESENVOLVIMENTO

Testes



TESTES

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

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	PV02
P02	Experiência do usuário (UX)	ST02, ST03	PV01 , PV05
P03	Testabilidade e cobertura de testes	ST04	PV02 , PV04
P04	Tempo de entrega e valor ao usuário final	ST03	PV02 , PV05
P05	Alta disponibilidade e desempenho	ST05	PV01 , PV03
P06	Evolutividade e manutenção futura	ST01, ST04, ST05	PV01 , PV02

P08	Suporte a diferentes perfis de usuário	ST03	PV05 , PV01
P09	Adoção de boas práticas de engenharia	ST01, ST02, ST04	PV02
P10	Segurança e privacidade de dados	ST05	PV01 , PV03

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
<u>PV01</u>	Lógico	Modelar os módulos, componentes e suas responsabilidades	Diagrama de Componentes (UML)	<u>P02, P05, P06, P07, P08, P10</u>
<u>PV02</u>	Desenvolvimento	Representar a organização do código e boas práticas adotadas	Diagrama de pacotes (UML), árvore de arquivos	<u>P01, P03, P04, P06, P09</u>
<u>PV03</u>	Implantação	Modelar a infraestrutura de execução e distribuição do sistema	Diagrama de Deploy (UML)	<u>P05, P07, P10</u>

fluxos de execução			Diagrama de Sequência (UML)	P03, P04, 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:

```
@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
    }

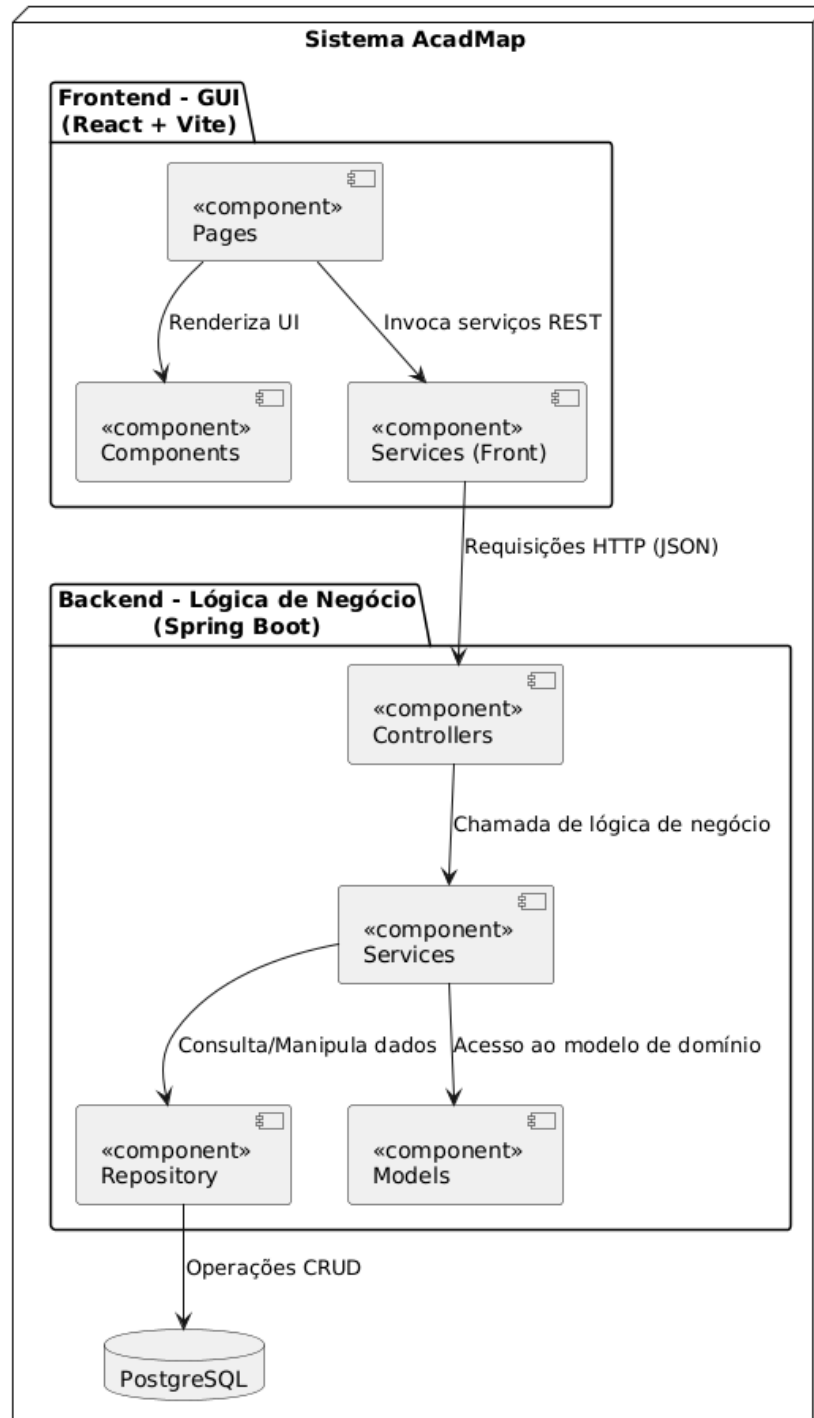
    database "PostgreSQL" as DB

    ' 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
```

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:

```
@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
}

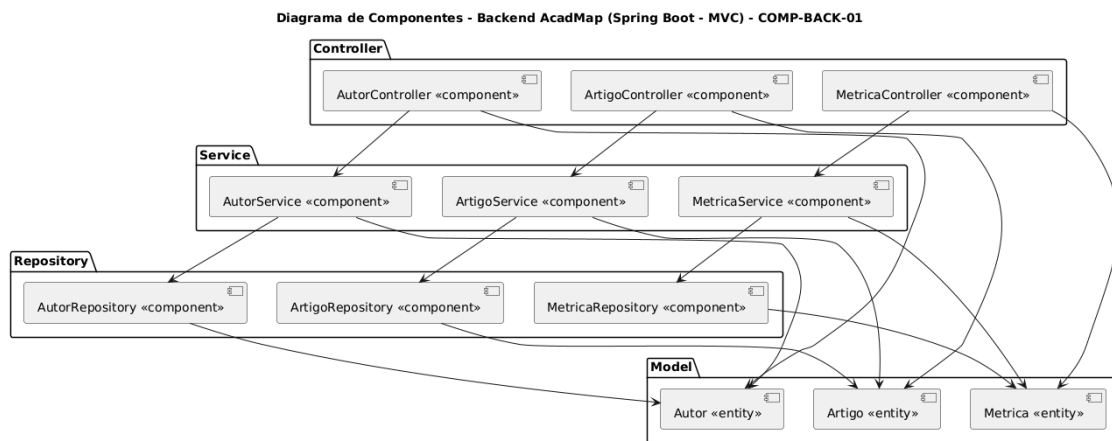
package "Model" {
    [Autor <<entity>>] as A
    [Artigo <<entity>>] as ART
    [Metrica <<entity>>] as M
}

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
```

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:

```
@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
}

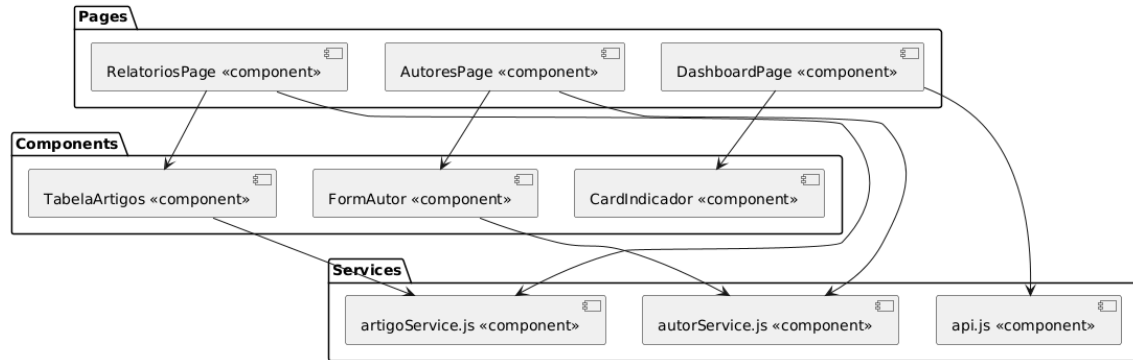
' Páginas usam componentes
DP --> CI
RP --> TA
AP --> FA

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

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

@enduml
```

Diagrama de Componentes do Frontend (React + Vite) - COMP-FRONT-01



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:

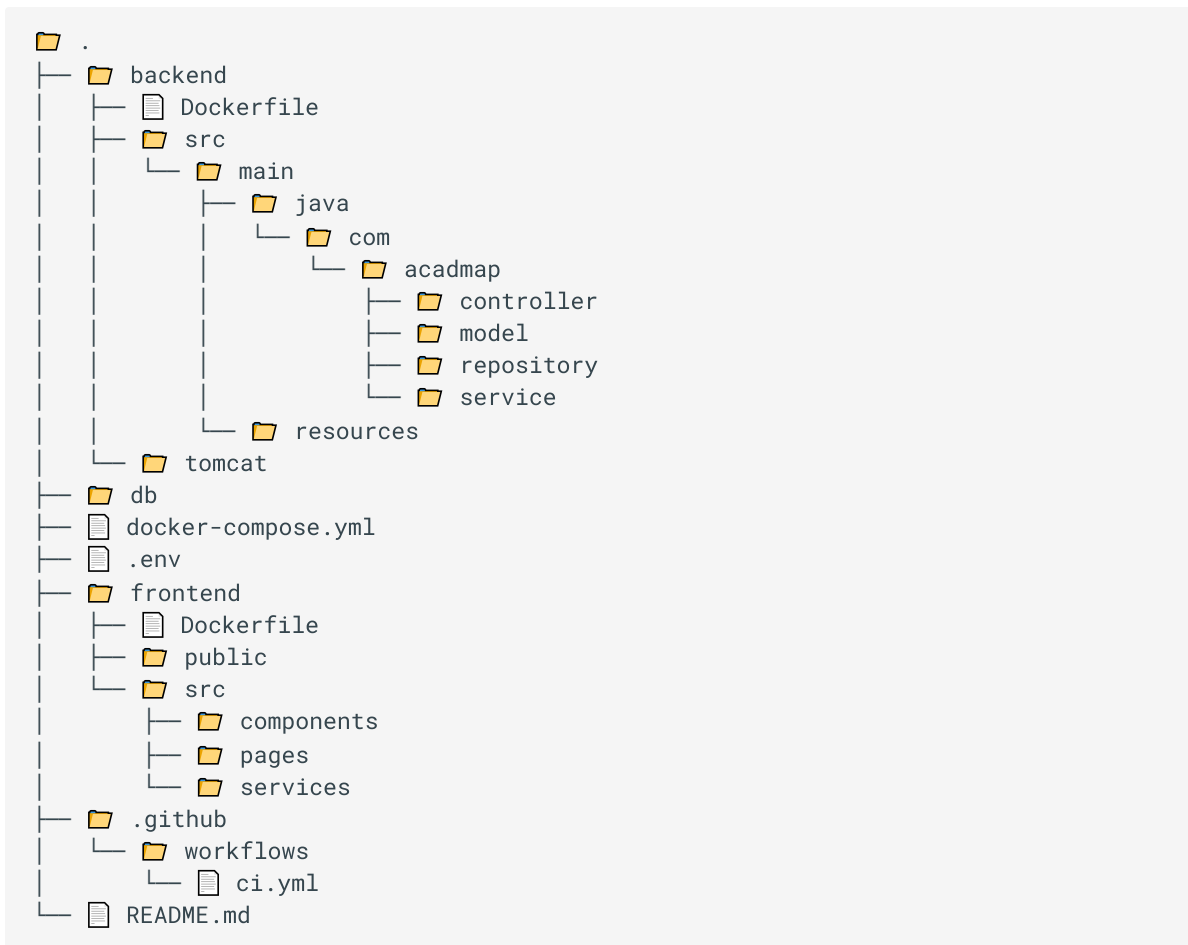
1. [Repositório de Desenvolvimento](#)
2. [Repositório de Documentação](#)

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



5.2.1 Organização de Diretórios (Frontend e Backend)

A estrutura de diretórios do projeto AcadMap segue uma organização baseada em boas práticas de engenharia de software, com separação clara entre responsabilidades tanto no backend quanto no frontend.

Backend (`backend/src/main/java/com/acadmap`)

No backend, implementado com Spring Boot, adota-se o padrão MVC (Model-View-Controller) organizado em quatro diretórios principais:

- **controller/**: contém as classes responsáveis por lidar com as requisições HTTP. Cada classe define os endpoints REST da aplicação, delegando a lógica de negócio para os serviços.
Exemplo: `AutorController.java`, `ArtigoController.java`.
- **service/**: abriga as classes que implementam a lógica de negócio. Os serviços recebem chamadas dos controladores, processam regras, validam dados e orquestram chamadas aos

repositórios. Exemplo: `AutorService.java`.

- **repository/**: inclui as interfaces responsáveis pelo acesso a dados, geralmente estendendo `JpaRepository` ou `CrudRepository`. Elas encapsulam a persistência e possibilitam operações como salvar, buscar e deletar entidades. Exemplo: `AutorRepository.java`.
- **model/**: define as classes que representam as entidades do domínio, mapeadas para as tabelas do banco de dados através de anotações JPA. Exemplo: `Autor.java`, `Artigo.java`.

Essa separação garante coesão, testabilidade e facilidade de manutenção no backend.

Frontend (`frontend/src`)

No frontend, construído com React e Vite, o código é dividido de acordo com a função dos elementos dentro da interface e da lógica da aplicação:

- **pages/**: agrupa os componentes de alto nível que representam as páginas principais do sistema, geralmente associadas às rotas do aplicativo. Cada página é composta por uma combinação de componentes reutilizáveis. Exemplo: `DashboardPage.jsx`, `RelatoriosPage.jsx`.
- **components/**: reúne componentes visuais reutilizáveis, como botões, tabelas, formulários e cartões. Esses elementos são usados em múltiplas páginas para garantir consistência e modularidade na interface. Exemplo: `CardIndicador.jsx`, `TabelaArtigos.jsx`.
- **services/**: contém os módulos responsáveis pela comunicação com a API backend. Esses serviços encapsulam chamadas HTTP usando bibliotecas como `axios` ou `fetch`, centralizando a lógica de consumo da API. Exemplo: `autorService.js`, `artigoService.js`.

5.2.1.1 Workflow de Versionamento

Esta seção tem como objetivo ser um guia para o desenvolvedor. Nele, abordaremos os seguintes assuntos:

- Criar uma nova branch de trabalho
- Como enviar essa branch ao github
- Como resolver conflitos de merge

Clonando o repositório

O passo inicial de todo fluxo de trabalho é fazer o clone do repositório, para isso, utilize o seguinte comando:

```
git clone https://github.com/acadmap-project/AcadMap.git
```

Criando a Branch

Com o repositório clonado, iniciaremos a criação de uma nova branch de funcionalidade. Para fins de exemplificação, chamarei de "feature/teste"

Sendo assim, todo o fluxo de trabalho sempre se iniciará a partir branch Dev, então vamos dar o mudar para ela:

```
git checkout dev
```

Estando na branch correta, temos de atualiza-la com as mudanças mais recentes antes de iniciar o nosso trabalho:

```
git pull
```

Com as atualizações feitas, podemos criar nossa branch nova baseada na dev:

```
git checkout -b feature/teste
```

Neste momento, na nova branch, podemos iniciar o desenvolvimento normalmente, fazendo as modificações necessárias para implementar a feature.



Nota

Lembre-se de manter seu código organizado, estruturado e bem documentado.

COMMITANDO SUAS MODIFICAÇÕES

Com a branch criada localmente e seu trabalho do dia completo, deve-se adicionar as mudanças feitas ao seu repositório local:

```
git add <nome-arquivo>  
git commit -m 'feat: added new button'
```

Enviando suas mudanças ao Github

Com o fim do seu dia de trabalho e seu repositório local contendo as suas últimas mudanças feitas, é importante manda-las para o repositório remoto para mante-lo sempre atualizado:

```
git push origin feature/teste
```

Finalizando o ciclo de vida de uma branch de feature

Ao terminar o desenvolvimento da sua feature, você deve criar um Pull Request no Github, mas para isso, têm alguns passos que devem ser feitos antes:

1. Verificar e atualizar o conteúdo da branch dev.
2. Fazer o merge local.

2.1 Resolver os conflitos localmente.

1. Criar o Pull Request
 2. Encerrar as branch
-

VERIFICANDO E ATUALIZANDO MUDANÇAS FEITAS NA BRANCH DEV

Primeiro, nós atualizamos nossa branch dev local com as mudanças que ocorreram na dev remote.

Mudamos para a branch dev local:

```
git checkout dev
```

Atualizamos ela:

```
git pull origin dev
```

Verificamos a existência de divergências entre a nossa branch de feature e a branch dev atualizada:

```
git log feature/teste..dev --oneline
```

FAZENDO O MERGE LOCAL

Caso existam diferenças, é necessário executar o merge da feature/teste <- dev localmente para garantir que nossa branch esteja atualizada.

Mudamos novamente para nossa branch de feature:

```
git checkout feature/teste
```

Fazemos o merge:

```
git merge dev
```

Resolvendo conflitos

Durante a execução do merge, podem surgir 2 tipos de conflitos diferente:

- Conflito por exclusão de arquivo
- Conflito por mudanças no mesmo conteúdo de um mesmo arquivo

Para garantir a continuação correta do fluxo, precisamos resolve-los para concluir o merge com sucesso.

Conflito por exclusão de arquivo

Esse tipo de conflito ocorre quando um arquivo foi excluído na branch dev, mas ainda está presente e possivelmente foi modificado na sua branch de feature.

O Git não sabe se deve manter o arquivo (por causa das suas alterações) ou excluí-lo (seguindo o que aconteceu na dev). Nesse caso, você precisa decidir manualmente o que deve ser feito.

Para visualizar os arquivos com conflito, utilize:

```
git status
```

Você verá uma mensagem similar a essa:

```
both deleted: nome-do-arquivo
```

A partir desse momento, existem dois caminhos a serem tomados:

- Aceitar a exclusão do arquivo
- Manter o arquivo

Caso você escolha aceitar a exclusão do arquivo:

```
git rm nome-do-arquivo
git commit
```

Caso você julgue que o arquivo é importante e escolha manter o arquivo:

```
git add nome-do-arquivo
git commit
```

Conflito por mudanças no mesmo conteúdo de um mesmo arquivo

Esse tipo de conflito acontece quando o mesmo trecho de um arquivo foi alterado tanto na branch dev quanto na sua branch de feature. O Git não consegue decidir automaticamente qual alteração manter, então você precisa resolver o conflito manualmente.

Ao executar o `git merge dev`, você verá mensagens de conflito como:

```
CONFLICT (content): Merge conflict in nome-do-arquivo.
```

rodando `git status` você verá:

```
both modified: nome-do-arquivo
```

Quando isso ocorrer, você deve abrir o arquivo o qual o conflito ocorreu utilizando seu editor de código de preferência.

Ao abrir o arquivo, você verá algo como:

```
<<<<<< HEAD
// Esta é a versão da sua branch atual (feature/teste)
console.log('Mensagem da sua feature');
=====
// Esta é a versão da branch dev
console.log('Mensagem da dev');
>>>>>> dev
```

Note que o Git nos ajuda marcando o local que ocorreram as diferenças

Agora, você deve editar esse trecho para que o código final reflita o que realmente deve permanecer. Pode ser:

- A versão da sua feature
- A versão da dev
- Uma combinação das duas (mais comum)

Como exemplo:

```
console.log('Mensagem da dev com melhorias da feature');
```

Nota

Depois de resolver todos os conflitos, remova os marcadores <<<<<<, =====, >>>>>>, salve o arquivo e faça o commit com as alterações concluídas:

```
git add nome-do-arquivo
git commit
```

Importante

Sempre converse com outros membros da sua equipe que trabalhou na dev, isso definitivamente ajudará a resolver os conflitos da melhor forma possível

Ao resolver os conflitos e realizar o merge com sucesso, você deve enviar essas alterações para seu repositório remoto:

```
git push origin feature/teste
```

CRIANDO O PULL REQUEST

Com as alterações enviadas a sua branch remota com sucesso, você pode criar o Pull Request pelo site do Github, no seguinte sentido:

```
main <- feature/teste
```

Etapas para criar um Pull Request:

1. Acesse o repositório no GitHub
2. Verifique se a sua branch foi enviada corretamente
3. Clique em "Compare & pull request"
4. Preencha as informações do PR
5. Envie o Pull Request

Com o Pull Request criado, basta esperar a aprovação pelo líder do seu time.



Cuidado

Nunca faça o merge do seu próprio PR sem aprovação.

ENCERRANDO A BRANCH

Após o seu Pull Request ser aprovado e feito o merge na branch dev, você pode deletar sua branch de feature. Isso é importante para manter o repositório limpo e organizado, evitando o acúmulo de branches antigas e já integradas.

- Deletando branch remota:

Caso o merge do seu Pull Request já tenha sido realizado no GitHub, você poderá deletar a branch remota acessando a aba "Pull Requests", clicando em "Closed" e selecionando o PR correspondente à sua branch. Na página do PR, haverá um botão "Delete branch", clique nele para remover a branch do repositório remoto.

- Deletando branch localmente:

Para excluir a branch local (após ter certeza de que o trabalho foi integrado corretamente):

```
git branch -d feature/teste
```

Dessa forma, concluímos o passo-a-passo do fluxo completo de trabalho no repositório de desenvolvimento! Em casos de dúvidas quanto a gerência do repositório, contate o líder da sua equipe ou um Arquiteto!

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:


```
@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

Visão de Processos



Visão de Cenários



Decisões Arquiteturais (DAR)



7. Governança de Arquitetura



A governança da arquitetura do sistema AcadMap segue diretrizes que garantem rastreabilidade, consistência e validação colaborativa das decisões técnicas.

7.1 Decisões Arquiteturais

Todas as decisões arquiteturais relevantes são formalizadas por meio de registros do tipo RDA (Registro de Decisão Arquitetural), numerados sequencialmente e armazenados no repositório `docs/rda`.

7.2 Alterações no DAS

Qualquer alteração significativa no Documento de Arquitetura de Sistema (DAS) deve ser submetida via *pull request*, obrigatoriamente revisada por outro membro da equipe de Arquitetura. Essa prática assegura que as modificações sejam avaliadas quanto à sua conformidade com os princípios arquiteturais e impacto no sistema.

7.3 Sobre o Versionamento

Além disso, as versões do DAS são etiquetadas conforme os ciclos de entrega do projeto, utilizando versionamento semântico e.g. `v1.0`, `v1.1` e assim por diante. Isso permite o controle histórico da evolução da arquitetura e facilita a referência a versões específicas durante auditorias, revisões técnicas e fases de manutenção.

7.4 Ferramentas de Apoio à Arquitetura

A elaboração, manutenção e disseminação da arquitetura do sistema AcadMap são apoiadas por um conjunto de ferramentas que promovem automação, padronização e rastreabilidade. Essas ferramentas estão integradas ao fluxo de desenvolvimento e à governança do DAS, conforme a tabela abaixo:

ID	Ferramenta	Finalidade
001	MkDocs	Geração da documentação navegável em Markdown
002	Mermaid	Criação de diagramas leves em Markdown
003	PlantUML	Modelagem de diagramas estruturados via texto
004	GitHub	Controle de versionamento e colaboração
005	GitHub Actions	Automação de pipelines CI/CD e publicação
006	GitHub Projects	Acompanhamento visual das atividades (Kanban)

8. Riscos e Mitigações

Glossário

Referências

12. Referências

- ISO/IEC/IEEE 42010:2011 – *Systems and software engineering – Architecture description*
<https://www.iso.org/standard/50508.html>
- Spring Boot Reference Documentation
<https://docs.spring.io/spring-boot/docs/current/reference/html/>
- React Official Documentation
<https://react.dev/>
- PostgreSQL Official Documentation
<https://www.postgresql.org/docs/>
- Swagger UI
<https://swagger.io/tools/swagger-ui/>
- SonarQube Documentation
<https://docs.sonarsource.com/sonarqube-community-build/>
- OWASP Dependency-Track
<https://owasp.org/www-project-dependency-track/>
- JCR (Journal Citation Reports – Clarivate)
<https://clarivate.com/webofsciencegroup/solutions/journal-citation-reports/>
- Scopus APIs (Elsevier Developer Portal)
https://dev.elsevier.com/sc_apis.html
- Plataforma SOL da Sociedade Brasileira de Computação
<https://sol.sbc.org.br/>
- Qualis/CAPES antigo (consultas públicas)
<https://sucupira-legado.capes.gov.br/sucupira/>
- Google Scholar
<https://serpapi.com/google-scholar-api>
- Conventional Commits <https://www.conventionalcommits.org/pt-br/v1.0.0-beta.4/>