FACULDADE DE TECNOLOGIA DE DIADEMA LUIGI PAPAIZ
DESENVOLVIMENTO DE SOFTWARE MULTIPLATAFORMA

BÁRBARA CHACON CHAVES
EDUARDO TEBEXERINI ABREU
FELIPE DE SOUZA FERREIRA
GUILHERME COELHO GARNIZET
MATHEUS SILVA CIRIACO
PEDRO ALBERTINI FERNANDES PINTO
PEDRO HENRIQUE DOS SANTOS MARTINES

Saúde++ – Parte 2

Documento de Arquitetura de Software

Versão 1.0

DIADEMA
2025

Histórico de Revisões			
Versão	**Data**	**Descrição**	**Autor**
0.1	2025-08-25	Desenvolvimento do Front-End	Bárbara Chacon Chaves, Eduardo Tebexerini Abreu, Felipe Ferreira
0.2	2025-09-17	Criação do Banco de Dados em MongoDB	Pedro Albertini
0.3	2025-09-17	Desenvolvimento IA Generativa	Guilherme Garnizet
1.0	2025-09-17	Projeto em SpringBoot implementado	Matheus Ciriaco
1.1	2025-09-17	Implementação do PDFBox	Matheus Ciriaco
1.2	2025-09-18	Spring Security + Rotas Protegidas	Matheus Ciriaco, Pedro Albertini

Sumário

Documento de Arquitetura de Software

1. Introdução

Este documento descreve a arquitetura do sistema "Saúde++" — uma plataforma web destinada a profissionais de saúde para consulta e gerenciamento de prontuários médicos com o auxílio da IA. O objetivo é fornecer uma visão clara e estruturada das decisões arquiteturais, componentes principais, integrações e restrições técnicas que sustentam a solução, servindo como referência para desenvolvedores, arquitetos, equipes de infraestrutura e partes interessadas do projeto.

2. Identificação do Projeto

Projeto	Saúde++
Requisitante	FATEC Luigi Papaiz – Curso DSM
Responsável do Projeto	Matheus Silva Ciriaco

3. Representação Arquitetural

A arquitetura do sistema adota o modelo **em camadas (Layered Architecture)**, complementado por um **estilo Cliente-Servidor**.
As visões utilizadas neste documento são:

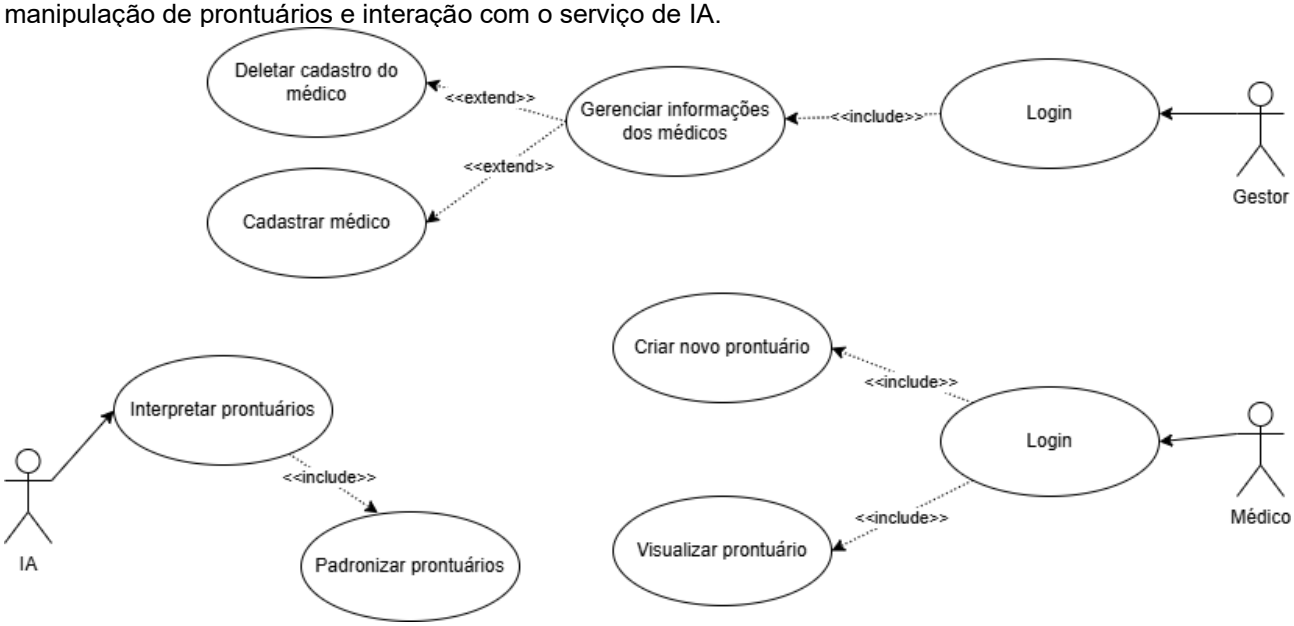
- **Visão de Casos de Uso:** descreve as principais interações entre usuários e sistema.
- **Visão Lógica:** detalha a decomposição em camadas e pacotes (Controllers, Services, Repositories, Domain).
- **Visão de Processos:** representa a execução concorrente e comunicação entre módulos Java e o serviço de IA (FastAPI).
- **Visão de Implantação:** mostra os nós físicos (Servidor Spring Boot, MongoDB e Serviço de IA).
- **Visão de Implementação:** apresenta a estrutura de código-fonte e dependências Maven.
  A arquitetura é representada graficamente por diagramas UML e Mermaid (Component, Deployment e Sequence).

4. Metas e Restrições da Arquitetura

- Desempenho**:** resposta < 3 s por requisição.
- Disponibilidade**:** dependência local (Mongo e IA).
- Segurança**:** autenticação via JWT, dados sensíveis criptografados.
- Portabilidade**:** compatível com ambientes Windows/Linux.
- Restrições**:** uso obrigatório de Spring Boot 3.5.5, MongoDB 6.x, FastAPI em Python 3.10, e JDK 21.

5. Visão de Casos de Uso

Esta seção apresenta os principais **casos de uso arquiteturalmente significativos** do sistema *Saúde++*. Eles representam as funcionalidades centrais relacionadas a autenticação, gerenciamento de médicos, manipulação de prontuários e interação com o serviço de IA.



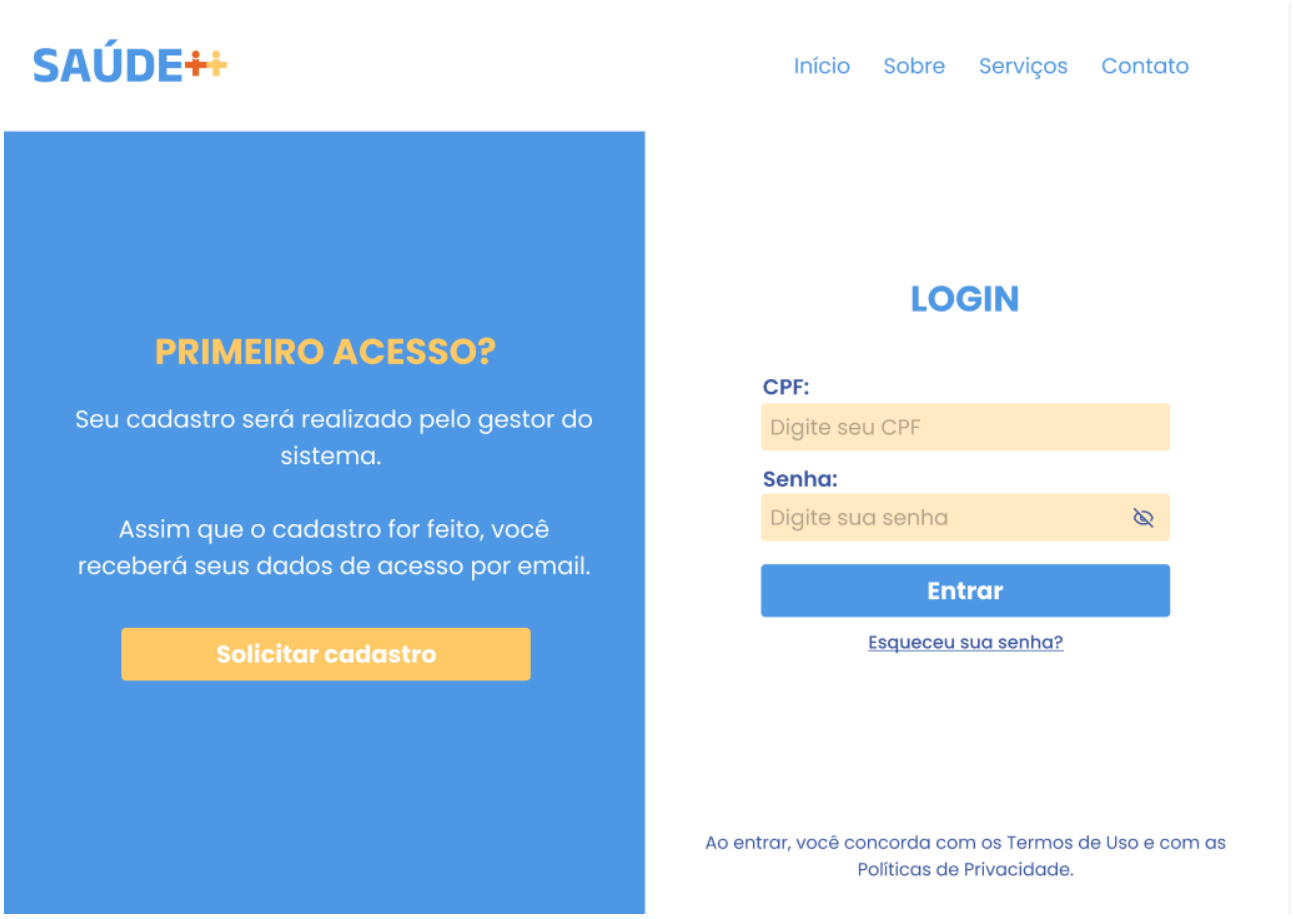
5.1.1. CSU01 – Fazer Login

Sumário:
O sistema autentica o usuário, permitindo o acesso de médicos e gestores conforme o perfil.
Ator Primário: Médico, Gestor
Atores Secundários: —
Pré-condições: O usuário deve possuir cadastro prévio no sistema.
Pós-condições: O usuário é autenticado e tem acesso às funcionalidades correspondentes ao seu perfil.
Fluxo Principal:

1. O caso de uso inicia quando o usuário acessa a tela de login.
2. O gestor informa seu **código de identificação** e senha, ou o médico informa seu **CRM** e senha.
3. O sistema valida as credenciais e identifica o perfil.
4. O sistema gera um **token JWT** e o armazena em cookie.
5. O usuário é redirecionado ao painel correspondente.

Fluxo Alternativo:

- 3a. Se as credenciais forem inválidas, o sistema exibe mensagem de erro e retorna à tela de login.
- 3b. Se o usuário estiver inativo, o sistema bloqueia o acesso e exibe aviso de conta desativada.



5.1.2. CSU02 – Gerenciar Médicos

Sumário:
O gestor administra o cadastro de médicos, podendo criar, editar ou excluir registros.
Ator Primário: Gestor
Pré-condições: O gestor deve estar autenticado.
Pós-condições: As informações dos médicos são atualizadas e persistidas no banco de dados.
Fluxo Principal:

1. O gestor acessa a área de gerenciamento de médicos.
2. O sistema exibe a lista de profissionais cadastrados.
3. O gestor pode:
    - Criar novo médico (<<include>> CSU03)
    - Atualizar dados de médico existente
    - Remover médico (<<include>> CSU04)
4. O sistema valida as alterações e grava as modificações.

Fluxo Alternativo:

- 3a. Caso o médico informado já exista (CRM duplicado), o sistema exibe alerta e impede a gravação.
- 4a. Se ocorrer erro de conexão com o banco, o sistema exibe mensagem de falha e solicita nova tentativa.

5.1.3. CSU03 – Cadastrar Médico

Sumário:
Permite ao gestor cadastrar um novo médico com informações de identificação e acesso.
Ator Primário: Gestor
Pré-condições: O gestor deve estar autenticado e possuir permissão de cadastro.
Pós-condições: O novo médico é criado e pode realizar login.
Fluxo Principal:

1. O gestor acessa a tela de cadastro.
2. Informa nome, CRM, especialidade, e-mail e senha provisória.
3. O sistema valida dados obrigatórios e integridade do CRM.
4. O novo registro é salvo no MongoDB.
5. O sistema exibe mensagem de sucesso.

Fluxo Alternativo:

- 3a. Dados inválidos ou campos obrigatórios ausentes — o sistema destaca os erros e solicita correção.


SAÚDE++
CADASTRO - MÉDICO
1
2
3
Dados Pessoais e de Login
Nome:
Sobrenome:
CPF (para Login):
CPF do Médico:
RG:
Data de Nascimento:
Sexo:
Voltar
Continuar


SAÚDE++
CADASTRO - MÉDICO
1
2
3
Dados Profissionais
CRM:
Especialização:
ID do Gestor:
Voltar
Continuar

5.1.4. CSU04 – Deletar Médico

Sumário:
Remove do sistema o cadastro de um médico inativo ou desligado.
Ator Primário: Gestor
Pré-condições: O gestor deve estar autenticado.
Pós-condições: O médico é removido e perde acesso ao sistema.
Fluxo Principal:

1. O gestor pesquisa o médico a ser removido.
2. O sistema exibe os resultados e solicita confirmação.
3. Após confirmação, o registro é excluído do banco de dados.
4. A listagem de médicos é atualizada.

Fluxo Alternativo:

- 3a. Caso o médico possua prontuários associados, o sistema impede exclusão direta e exibe mensagem: “Remova ou transfira os prontuários antes da exclusão.”

5.1.5. CSU05 – Criar Novo Prontuário

Sumário:
O médico registra um novo prontuário clínico de paciente.
Ator Primário: Médico
Pré-condições: O médico deve estar autenticado.
Pós-condições: O prontuário é criado e vinculado ao médico responsável.
Fluxo Principal:

1. O médico acessa a tela “Novo Prontuário”.
2. Informa dados do paciente, histórico, sintomas e diagnósticos.
3. O sistema valida os campos obrigatórios.
4. Os dados são salvos no MongoDB.
5. O sistema confirma a criação.

Fluxo Alternativo:

- 3a. Caso haja erro de validação, o sistema notifica o usuário e mantém os dados preenchidos para correção.
- 4a. Se a conexão com o banco falhar, o prontuário não é salvo e uma mensagem de erro é exibida.

5.1.6. CSU06 – Visualizar Prontuário

Sumário:
O médico consulta prontuários existentes.
Ator Primário: Médico
Pré-condições: O médico deve estar autenticado e existir pelo menos um prontuário.
Pós-condições: O prontuário é exibido para leitura e possível atualização.
Fluxo Principal:

1. O médico acessa “Listar Prontuários”.
2. O sistema recupera os registros do banco.
3. O médico seleciona um prontuário.
4. O sistema exibe detalhes e histórico do paciente.

Fluxo Alternativo:

- 2a. Caso não haja prontuários, o sistema exibe a mensagem “Nenhum prontuário encontrado.”

5.1.7. CSU07 – Interpretar Prontuário

Sumário:
O médico solicita à IA a interpretação de um prontuário clínico.
Ator Primário: Médico
Ator Secundário: Serviço de IA (FastAPI)
Pré-condições: Deve existir um prontuário registrado.
Pós-condições: O médico visualiza o resultado interpretado (diagnósticos e observações).
Fluxo Principal:

1. O médico acessa um prontuário e clica em “Interpretar”.
2. O sistema envia o conteúdo para a IA (via REST).
3. A IA executa a **padronização de termos** (<<include>> CSU08).
4. Em seguida, aplica modelos de análise semântica.
5. A IA retorna diagnósticos e insights ao sistema.
6. O sistema apresenta o resultado estruturado na interface do médico.

Fluxo Alternativo:

- 2a. Se o serviço IA estiver indisponível, o sistema informa “Serviço temporariamente fora do ar.”
- 4a. Caso o conteúdo do prontuário esteja em formato inválido, a IA retorna erro e o sistema registra o log de falha.

5.1.8. CSU08 – Padronizar Prontuário

Sumário:
A IA uniformiza terminologias médicas em um prontuário.
(*Caso de uso incluído em CSU07 – Interpretar Prontuário*)
Ator Primário: Médico
Ator Secundário: Serviço de IA (FastAPI)
Pré-condições: Deve existir prontuário válido.
Pós-condições: O prontuário padronizado é salvo no banco de dados.
Fluxo Principal:

1. O sistema envia os dados clínicos à IA.
2. A IA aplica modelos de NLP para normalizar termos.
3. Retorna o texto padronizado.
4. O sistema salva o resultado no MongoDB.

5.1.9. CSU09 – Encerrar Sessão

Sumário:
O usuário encerra sua sessão de uso do sistema.
Ator Primário: Médico, Gestor
Pré-condições: O usuário deve estar autenticado.
Pós-condições: O cookie JWT é removido e o acesso é finalizado.
Fluxo Principal:

1. O usuário seleciona a opção “Sair”.
2. O sistema invalida o token JWT armazenado.
3. O cookie é apagado e o usuário é redirecionado à tela de login.

5.2. Realizações de Casos de Uso

Abaixo está uma realização típica, exemplificando o fluxo interno de componentes.

5.2.1. Realização: CSU07 – Interpretar Prontuário

Componentes envolvidos:

- ProntuarioController
- IAService (cliente REST)
- ProntuarioRepository
- saude_mais.py (FastAPI)
- MongoDB

Fluxo:

1. O ProntuarioController recebe a requisição /interpretar/{id}.
2. O IAService envia o conteúdo do prontuário ao endpoint da IA.
3. O saude_mais.py processa o texto, aplica padronização e retorna resultado.
4. O ProntuarioRepository salva a resposta da IA.
5. O sistema retorna o resultado formatado ao frontend.

6. Visão Lógica

A arquitetura adota um modelo em camadas com separação de preocupações:

- **Apresentação (Controllers + Thymeleaf)**: entrega páginas e expõe REST.
- **Serviço (Service)**: regras de negócio, orquestração de chamadas ao MongoDB e à IA.
- **Domínio (Model/DTO)**: entidades e objetos de transferência.
- **Persistência (Repository)**: acesso a dados via Spring Data MongoDB.
- **Integração (Clients)**: clientes HTTP para o FastAPI (saude_mais.py).
- **Infra/Security (auth, config)**: JWT (JJWT), filtros, CORS, política de cookies.

6.1. Visão Geral

A arquitetura do **Saúde ++** adota o **modelo em camadas (Layered Architecture)**, estruturado segundo o padrão MVC (Model-View-Controller) do Spring Boot. Essa organização visa promover **separação de responsabilidades**, **facilidade de manutenção** e **evolução modular** do sistema. O design foi concebido para permitir que o código-fonte cresça de forma controlada, preservando baixo acoplamento entre componentes e alta coesão dentro de cada camada.
O sistema é composto por cinco camadas principais:

1. **Camada de Apresentação** – Responsável pela interface entre usuário e aplicação. Inclui os *controllers* Spring MVC, que tratam requisições HTTP e interagem com as páginas **Thymeleaf** (atualmente em uso) ou com o **frontend React** (em desenvolvimento). Essa camada não contém lógica de negócio, atuando apenas como mediadora entre a interface e os serviços internos.
2. **Camada de Serviço** – Implementa as **regras de negócio** e realiza a orquestração entre *controllers*, repositórios e integrações externas. Abriga componentes como MedicoService, ProntuarioService, IAService e PdfService, responsáveis por validações, transformações de dados e integração com o módulo de IA (FastAPI).
3. **Camada de Domínio** – Reúne as **entidades e DTOs** que representam os conceitos centrais do sistema, como médicos, usuários e prontuários. Essa camada expressa o modelo conceitual do domínio da saúde, independente de tecnologia ou persistência.
4. **Camada de Persistência** – Contém os **repositórios Spring Data MongoDB**, responsáveis pelo acesso e manipulação dos dados armazenados na base **MongoDB** (mongodb://localhost:27017/saude). Essa camada mantém a independência entre o modelo de domínio e o mecanismo de persistência.
5. **Camada de Integração** – Engloba os **clientes HTTP** e componentes de comunicação com sistemas externos, especialmente o **serviço de IA** implementado em Python (FastAPI). Essa camada encapsula as chamadas externas, padronizando formatos e respostas, de modo que falhas de comunicação não afetem o núcleo da aplicação.

Além das camadas funcionais, há dois módulos transversais:

- **auth/config** – engloba autenticação e segurança, com geração e validação de **JWT** via biblioteca JJWT e configuração de filtros do **Spring Security**;
- **pdf** – fornece suporte à criação de relatórios e documentos utilizando a **Apache PDFBox**.

6.2. Pacotes de Design Significativos do Ponto de Vista da Arquitetura

A tabela a seguir descreve os principais pacotes do projeto Saúde++, suas classes encontradas e responsabilidades arquiteturais observadas no código-fonte atual. Observações indicam componentes ausentes ou planejados para versões futuras.

Pacote	Descrição	Classes	Responsabilidades
controller	Controladores MVC/REST — expõem páginas Thymeleaf e endpoints JSON.	AppController, MedicoRestController, PdfController, ProfileController	Mapeamento de rotas; validação de entrada; delegação a serviços; renderização de templates ou respostas JSON. ProntuarioController não identificado no repositório atual.
auth.service	Serviços de autenticação/autorização.	JwtService, CustomUserDetailsService, DatabaseSeeder	Gerar e validar JWTs; carregar detalhes de usuário; inicializar dados de autenticação.
auth.repository	Repositórios Spring Data relacionados à autenticação.	UserRepository	Operações CRUD e consultas na coleção de usuários; outros repositórios de domínio (ex.: MedicoRepository) ausentes.
auth.model	Entidades/DTOs ligados à autenticação.	Usuario	Representa credenciais e papéis de usuário; mapeado para MongoDB.
config	Configurações globais de segurança e filtros.	SecurityConfig, JwtAuthenticationFilter	Define políticas de segurança e encadeamento de filtros JWT. WebConfig e MongoConfig não identificados; Mongo configurado via application.properties.
pdf.controller	Geração/serviço PDF (controlador presente).	PdfController	Serviço de geração de relatórios em PDF (Apache PDFBox). PdfService ausente; provável geração direta no controller.
ia.integration	Serviço de integração com IA (FastAPI).	saude_mais.py	Serviço Python executável (porta 8000). Cliente Java FastApiClient não localizado no repositório.
model.domain	Entidades de domínio (médico, prontuário etc.).	Medico, Prontuario, InterpretacaoIA, PadronizacaoIA	Modelos ausentes no código atual; parte do escopo funcional e previstos para inclusão futura.
repository.domain	Repositórios para entidades de domínio (MongoDB).	MedicoRepository, ProntuarioRepository	Implementar quando entidades correspondentes forem criadas.
service.domain	Serviços de negócio (médico, prontuário, PDF).	MedicoService, ProntuarioService, PdfService, IAService	Camada recomendada para regras de negócio e testes unitários; criação planejada.

client.integration	Clientes HTTP para comunicação com a IA.	FastApiClient	Integração prevista com FastAPI; atualmente não implementada no repositório.

6.3. Camadas

O sistema **Saúde ++** está dividido em **cinco camadas principais**, complementadas por **módulos transversais** de suporte.
Cada camada agrupa subsistemas e componentes com responsabilidades bem definidas, garantindo **baixo acoplamento** e **alta coesão** entre módulos.

6.3.1. Camada de Apresentação

Descrição:
É a camada mais externa da aplicação, responsável por receber e responder às requisições dos usuários. Inclui os controladores Spring MVC, arquivos de template Thymeleaf e scripts estáticos (CSS, JS e HTML).
Responsabilidades principais:

- Expor endpoints REST e páginas web.
- Interagir com o usuário médico e gestor.
- Encaminhar as requisições para os serviços de negócio.
- Renderizar respostas no formato JSON ou HTML.

Principais componentes:

- AppController
- MedicoRestController
- PdfController
- Templates em /templates e assets em /static.

6.3.2. Camada de Serviço

Descrição:
Concentra as **regras de negócio** e a lógica de orquestração.
Aqui são aplicadas validações, tratamentos de exceções e chamadas aos repositórios e integrações externas.
Responsabilidades principais:

- Processar regras de negócio do domínio médico.
- Orquestrar operações entre controladores, repositórios e IA.
- Garantir consistência das transações lógicas.

Principais componentes:

- MedicoService
- ProntuarioService
- PdfService
- IAService

Observação:
Esta camada é o núcleo funcional da aplicação e serve de ponte entre a Apresentação e as demais camadas.

6.3.3. Camada de Domínio

Descrição:
Representa o modelo conceitual da aplicação, contendo as entidades e objetos de transferência de dados (DTOs).
Os elementos dessa camada são independentes de frameworks e visam refletir a realidade do negócio da saúde.
Responsabilidades principais:

- Modelar os conceitos principais (médico, usuário, prontuário).
- Servir como base para persistência e integração.
- Manter consistência sem depender de camadas superiores.

Principais componentes:

- Medico
- Usuario
- Prontuario

- InterpretacaoIA
- PadronizacaoIA

6.3.4. Camada de Persistência

Descrição:
Implementa a comunicação com o **banco de dados MongoDB**, utilizando o **Spring Data MongoDB**. Cada repositório fornece métodos CRUD e consultas derivadas a partir do nome.
Responsabilidades principais:
- Persistir e recuperar documentos do banco.
- Manter integridade das coleções.
- Isolar o acesso ao MongoDB do restante do sistema.

Principais componentes:
- MedicoRepository
- ProntuarioRepository
- UsuarioRepository

6.3.5. Camada de Integração

Descrição:
Responsável pela comunicação com serviços externos, em especial o módulo de **Inteligência Artificial (FastAPI)**.
Essa camada encapsula os detalhes de transporte e formato de dados para evitar impacto no núcleo da aplicação.
Responsabilidades principais:
- Enviar dados de prontuário ao serviço IA (saude_mais.py).
- Receber e tratar respostas padronizadas.
- Implementar políticas de timeout e retries.

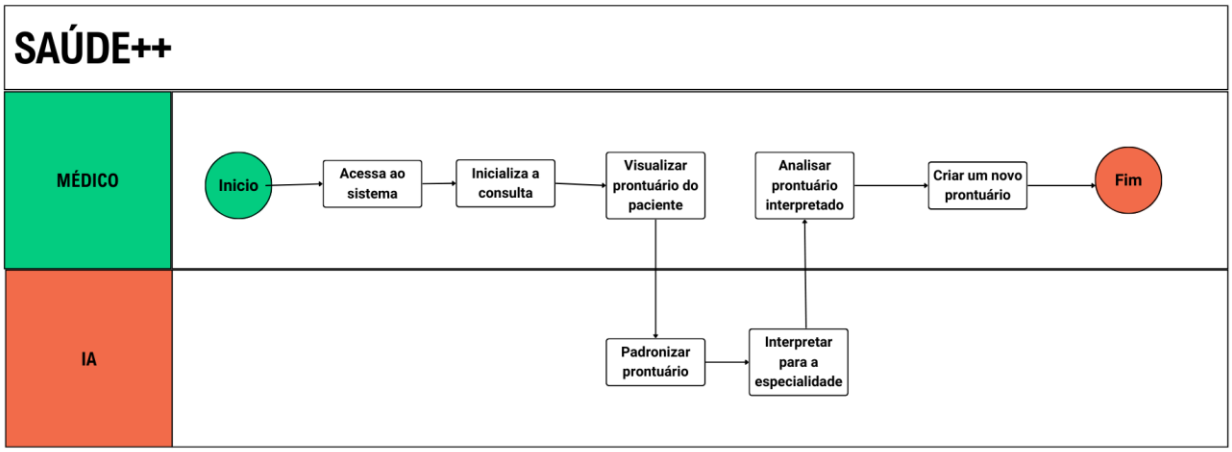
Principais componentes:
- FastApiClient *(previsto)*
- IAService *(atualmente intermediário)*
- saude_mais.py *(componente externo, porta 8000)*

6.3.6. Módulos Transversais

Descrição:
Além das camadas funcionais, existem módulos que atuam de forma transversal, fornecendo serviços compartilhados.
Principais módulos:
1. **auth/config** – gerenciamento de autenticação e autorização via **JWT**, integração com Spring Security e filtros de requisição.
    - Classes: JwtUtil, JwtAuthenticationFilter, SecurityConfig.
2. **pdf** – geração de relatórios e documentos usando **Apache PDFBox**, acessado tanto por controladores quanto por serviços.
    - Classes: PdfController, PdfService.

7. Visão de Processos

O sistema **Saúde ++** é essencialmente **síncrono**, baseado em requisições HTTP tratadas pelo **Spring Boot** com servidor embutido (Tomcat).
Cada requisição é processada em uma thread independente do pool gerenciado pelo framework, garantindo atendimento concorrente e seguro.
Há também comunicação **interprocessual** entre o backend (Java) e o **serviço de IA (FastAPI em Python)**, executado separadamente e acessado via requisições REST.



7.1.1. Processos Principais

- **Processo Web (Spring Boot)**: responsável pelo atendimento das requisições HTTP e geração de respostas (HTML ou JSON).
- **Processo de Persistência**: executa operações de leitura e gravação no MongoDB (porta 27017).
- **Processo de IA (FastAPI)**: realiza padronização e interpretação semântica de prontuários (porta 8000).

7.1.2. Comunicação entre Processos

- **Browser → Spring Boot (5000):** HTTP/HTTPS – troca de dados via formulários e APIs REST.
- **Spring Boot → FastAPI (8000):** HTTP POST – envio de prontuários para análise/retorno JSON.
- **Spring Boot → MongoDB (27017):** conexão TCP – driver oficial do Spring Data MongoDB.

8. Visão de Implantação

O sistema é implantado em ambiente local de desenvolvimento, com execução distribuída entre **três nós principais**: servidor web, serviço de IA e banco de dados.

8.1. Configuração Física e Lógica

Nó	Função	Tecnologia / Porta	Descrição
Cliente Web	Interface do usuário (navegador)	HTTP/HTTPS (80/443)	Acesso à aplicação via navegador.
Servidor Aplicacional	Backend Java (Spring Boot)	5000	Responsável pela lógica de negócio, autenticação e API REST.

Serviço IA	Backend Python (FastAPI)	8000	Processamento de texto e análise semântica.
Banco de Dados	Armazenamento MongoDB	27017	Persiste usuários, médicos e prontuários.

8.2. Mapeamento de Processos

- O **processo web** do Spring Boot é executado no mesmo nó do servidor aplicacional.
- O **processo IA** (FastAPI) roda isolado, comunicando-se via HTTP.
- O **MongoDB** roda em serviço independente, acessado pela camada de persistência.

9. Visão da Implementação

A implementação do **Saúde ++** segue a estrutura de camadas definida na visão lógica, organizada em pacotes conforme o padrão de convenção do Spring Boot.
A modularização busca garantir a rastreabilidade entre requisitos, código e casos de uso.

9.1. Visão Geral

A implementação do sistema Saúde ++ segue uma arquitetura em camadas, organizada por pacotes conforme as convenções do Spring Boot. O objetivo é manter separação clara de responsabilidades, facilitar os testes unitários e permitir rastreabilidade direta entre requisitos, código-fonte e casos de uso.

9.1.1. Camadas principais (nomes de pacotes e propósito)

Camada de Apresentação / Entrada (controller)

Local: src/main/java/.../controller/
Conteúdo: controladores MVC/REST que expõem páginas Thymeleaf e APIs JSON.
Regras de inclusão: classes que tratam requisições HTTP, mapeiam rotas via @Controller ou @RestController, fazem validações superficiais e chamam serviços. NÃO devem conter lógica de negócio nem acesso direto ao banco de dados.
Fronteira: interagem com os clientes (navegadores, aplicações externas) e consomem templates em /templates e recursos em /static.

Camada de Serviço (service)

Local: idealmente src/main/java/.../service/ e, atualmente, src/main/java/.../auth/service/.
Conteúdo: classes que implementam regras de negócio, validações complexas e orquestram chamadas a repositórios e serviços externos (IA, PDF).
Regras de inclusão: lógica de domínio, transações lógicas e coordenação entre múltiplas operações. Pode encapsular chamadas HTTP externas via clientes dedicados.
Fronteira: expõe métodos orientados ao domínio para os controllers e consome repositórios e clientes de integração.

Camada de Persistência / Repositórios (repository)

Local: src/main/java/.../auth/repository/ e futuro src/main/java/.../repository/.
Conteúdo: interfaces Spring Data MongoDB anotadas com @Repository, responsáveis por operações CRUD e consultas derivadas.
Regras de inclusão: acesso direto ao banco de dados, sem lógica de negócio; conversões simples entre documentos e entidades.

Camada de Modelo / Domínio (model / domain)

Local: src/main/java/.../auth/model/ e futuro src/main/java/.../model/.
Conteúdo: entidades persistentes (Document/Entity), DTOs para transporte entre camadas e objetos de valor.

Regras de inclusão: estruturas de dados puras, com validações via anotações, sem lógica de negócio. Boa prática: separar Entities (persistência) de DTOs (API) para evitar exposição de campos sensíveis.

Configuração Transversal (config)

Local: src/main/java/.../config/
Conteúdo: classes de configuração global, filtros de autenticação e segurança (JwtAuthenticationFilter), e políticas CORS e cookies.
Regras de inclusão: configurações, registradores de beans e componentes cross-cutting; não conter lógica de negócio.
Exemplo: SecurityConfig define regras de autenticação e autorização para as rotas da aplicação.

Integrações / Clientes (client / integration)

Local previsto: src/main/java/.../integration/ ou .../client/
Conteúdo: clientes HTTP e adaptadores para serviços externos, como a FastAPI (saude_mais.py), e componentes que tratam retries e timeouts.
Regras de inclusão: encapsular protocolos de comunicação e transformar formatos (JSON ↔ DTO), expondo interfaces simples para os serviços.

Utilitários e Infraestrutura (pdf, util, security helpers)

Local: src/main/java/.../pdf/, .../util/
Conteúdo: classes auxiliares, como geração de relatórios com Apache PDFBox, e funções reutilizáveis sem dependência de regras de negócio.
Regras de inclusão: utilitários genéricos e funções puras de apoio a múltiplas camadas.

9.2. Mapeamento para o Repositório Atual

Status	Elemento / Pacote	Descrição / Observação	Status
Implementado	controller: AppController, MedicoRestController, PdfController, ProfileController	Controladores principais do sistema.	Implementado
Implementado	auth/service: JwtService, CustomUserDetailsService, DatabaseSeeder	Serviços de autenticação e seed inicial.	Implementado
Implementado	auth/repository: UserRepository	Repositório de usuários autenticáveis.	Implementado
Implementado	auth/model: Usuario	Entidade persistente de autenticação.	Implementado
Implementado	config: SecurityConfig, JwtAuthenticationFilter	Configuração de segurança do Spring.	Implementado
Implementado	ia-integration: saude_mais.py (FastAPI)	Serviço Python de IA rodando na porta 8000.	Implementado
Planejado	service de domínio: MedicoService, ProntuarioService, IAService, PdfService	Regras de negócio do domínio médico.	Planejado
Planejado	model de domínio: Medico, Prontuario, InterpretacaoIA, PadronizacaoIA	Estrutura de entidades de negócio.	Planejado
Planejado	repository de domínio: MedicoRepository, ProntuarioRepository	Repositórios das entidades médicas.	Planejado
Planejado	client Java para IA: FastApiClient	Cliente REST para comunicação com o FastAPI.	Planejado

10. Visão de Dados (opcional)

A persistência de dados do sistema Saúde ++ é realizada utilizando o banco MongoDB, acessado através do módulo Spring Data MongoDB. Os dados são organizados em coleções que correspondem às entidades do

domínio (usuários, médicos, prontuários etc.), sendo manipulados por repositórios Java anotados com @Repository. O modelo é orientado a documentos, priorizando flexibilidade e agilidade na gravação e leitura.

10.1. Estrutura Geral

Coleção	Classe Java	Descrição / Finalidade
usuarios	Usuario	Armazena credenciais e perfis de acesso (médico, gestor).
medicos	Medico (planejado)	Contém dados profissionais e de registro CRM.
prontuarios	Prontuario (planejado)	Guarda informações clínicas e observações de pacientes.
interpretacoes	InterpretacaoIA (planejado)	Resultados da análise de prontuários pela IA.
padronizacoes	PadronizacaoIA (planejado)	Versões padronizadas dos textos clínicos.

10.2. Estrutura de Campos

Coleção usuarios:

```
{
  "_id": ObjectId,
  "username": "string",
  "senhaHash": "string (BCrypt)",
  "papel": "enum: ['MEDICO', 'GESTOR']",
  "ativo": true,
  "criadoEm": ISODate,
  "atualizadoEm": ISODate
}
```

Coleção medicos:

```
{
  "_id": ObjectId,
  "nome": "string",
  "crm": "string (único)",
  "especialidade": "string",
  "email": "string",
  "usuarioId": ObjectId,
  "ativo": true
}
```

Coleção prontuarios:

```
{
  "_id": ObjectId,
  "paciente": { "nome": "string", "cpf": "string", "nascimento": "date" },
  "sintomas": "string",
  "diagnostico": "string",
  "prescricoes": ["string"],
  "interpretacaoIA": ObjectId,
  "padronizacaoIA": ObjectId,
  "medicoId": ObjectId,
  "criadoEm": ISODate,
  "atualizadoEm": ISODate
}
```

10.3. Relacionamentos e Regras

• Usuário → Médico: relação 1:1 (cada médico é vinculado a um usuário autenticável).
• Médico → Prontuário: relação 1:N (um médico pode criar múltiplos prontuários).
• Prontuário → IA: relação 1:1 com interpretacaoIA e padronizacaoIA.
• Referências cruzadas são armazenadas via ObjectId em vez de joins relacionais.
• Dados sensíveis devem ser criptografados ou mascarados quando exibidos.

10.4. Índices e Validação

Coleção	Campo(s)	Tipo	Finalidade
usuarios	username	Único	Evitar duplicação de credenciais.
medicos	crm	Único	Garantir unicidade do registro profissional.
prontuarios	medicoId, atualizadoEm	Composto	Otimizar consultas por médico e data.

10.5. Segurança e Privacidade

• Criptografia BCrypt para senhas.
• Campos sensíveis mascarados na visualização.
• Backup automático e replica set em produção.
• Políticas de retenção de dados conforme LGPD (Lei 13.709/2018).

11. Tamanho e Desempenho

11.1. Metas de Desempenho

Métrica	Valor Atual / Esperado	Observação
Tempo médio de resposta (sem IA)	≤ 300 ms	Requisições simples processadas apenas pelo backend Spring Boot.
Tempo médio de resposta (com IA)	~ 60 segundos	A API Python (FastAPI) ainda está em otimização; tempo reflete o processamento do modelo de IA.
Conexões simultâneas	20 usuários (estimado)	Não há testes de carga executados até o momento.
Tempo máximo de autenticação	≤ 1 s	Inclui validação de credenciais e geração de token JWT.

Nota: os testes de desempenho e estresse ainda não foram realizados. As métricas acima são valores observados em ambiente de desenvolvimento.

11.2. Estratégias de Otimização (planejadas)

• Implementar testes de carga e estresse (JMeter ou k6).
• Adotar cache leve em memória por requisição.
• Criar índices MongoDB em crm, medicoId e atualizadoEm.
• Implementar paginação padrão (Pageable) nas listagens.
• Avaliar fila assíncrona (RabbitMQ ou Celery) para IA.
• Configurar timeout e retry no cliente HTTP (70s total, 1 retry).

11.3. Próximos Passos de Validação

1. Teste de Carga: medir throughput e latência P50, P95, P99.
2. Teste de Stress: identificar ponto de saturação do backend e da IA.
3. Teste de Endurance: avaliar estabilidade sob uso contínuo (≥ 4h).
4. Perfilamento: usar Spring Actuator e Prometheus/Grafana para gargalos.

Resumo: O tempo de resposta atual da IA é de aproximadamente 60 segundos, devido ao processamento dos modelos NLP ainda não otimizados. As demais metas permanecem adequadas; os testes de carga serão fundamentais para refinar performance e escalabilidade.

12. Qualidade

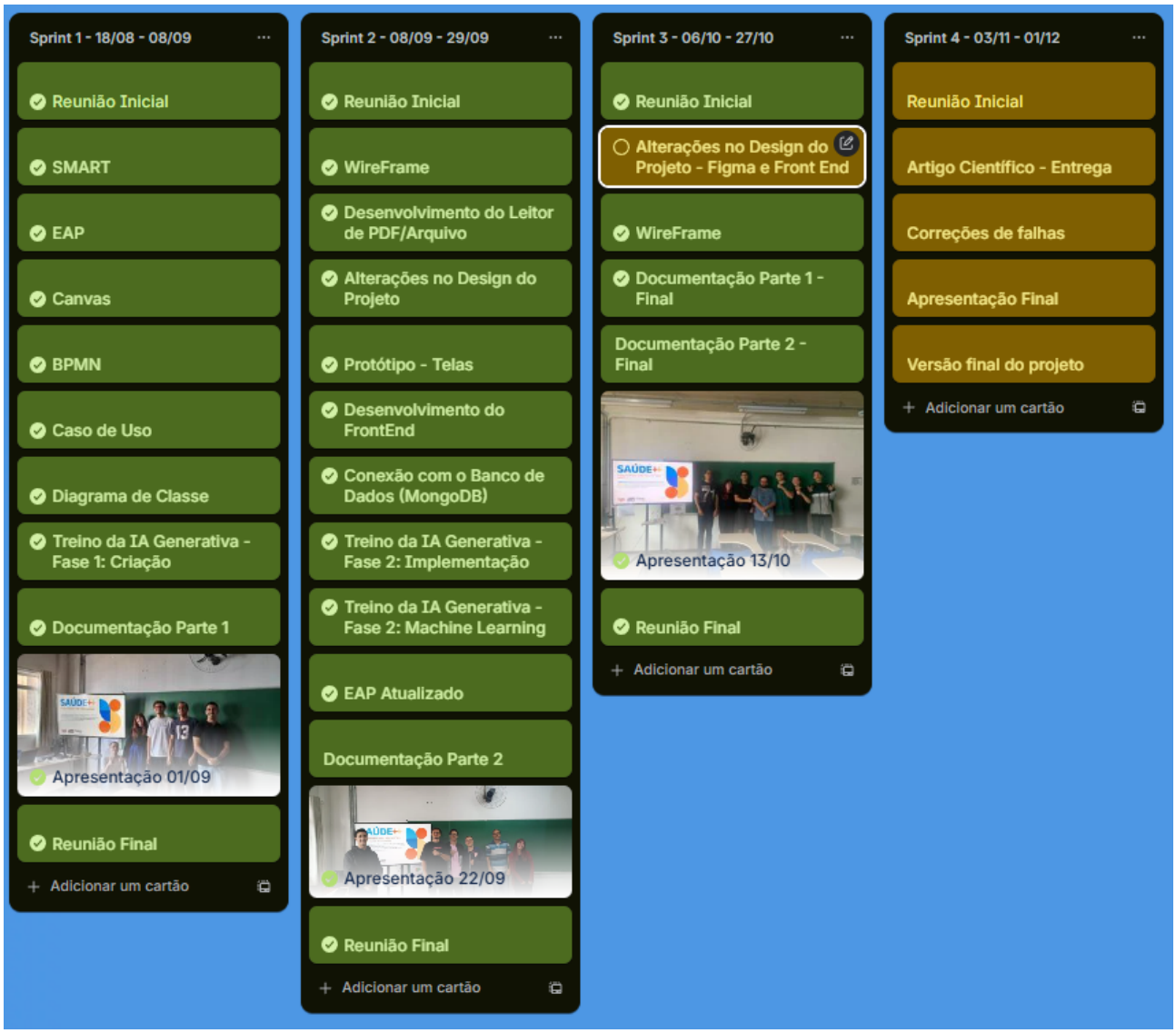
12.1. Atributos de Qualidade e Táticas Arquiteturais

Atributo	Tática Aplicada
Segurança	JWT com assinatura HMAC; cookies HttpOnly, Secure, SameSite; Spring Security; criptografia de senhas (BCrypt).
Confiabilidade	Log estruturado; controle de exceções global; validações antes de persistir.
Manutenibilidade	Estrutura em camadas; baixo acoplamento; uso de Lombok e interfaces claras.
Escalabilidade	Serviço de IA desacoplado (pode escalar horizontalmente).
Portabilidade	Java 21 + Python 3.10; compatível com Windows e Linux.
Privacidade	Dados sensíveis (CPF, email) protegidos por criptografia e mascaramento.
Auditabilidade	Logs de ações e eventos; histórico de alterações em prontuários.

12.2. Padrões de Desenvolvimento

- Padrão **MVC (Spring Boot)** para controle e separação lógica.
- DTOs para comunicação entre camadas.
- Repository **Pattern** com Spring Data.
- Service **Layer Pattern** para isolamento de regras de negócio.
- Integration **Client Pattern** para consumo da API FastAPI.

13. Apêndice



14. Referências

APACHE SOFTWARE FOUNDATION. *Apache PDFBox – A Java PDF Library*. Disponível em: https://pdfbox.apache.org. Acesso em: 30 out. 2025.

BRITO, R. de. *Spring Boot 3 e Spring Security 6: Guia completo*. São Paulo: Alura, 2024.

MONGODB INC. *MongoDB Manual – Version 6.x*. Disponível em: https://www.mongodb.com/docs. Acesso em: 30 out. 2025.

OPENJDK. *JDK 21 Documentation*. Oracle. Disponível em: https://docs.oracle.com/en/java/javase/21. Acesso em: 30 out. 2025.

PYTHON SOFTWARE FOUNDATION. *FastAPI Framework Documentation*. Disponível em: https://fastapi.tiangolo.com. Acesso em: 30 out. 2025.

SPRING. *Spring Boot Reference Documentation – Version 3.5.5*. Disponível em: https://docs.spring.io/spring-boot. Acesso em: 30 out. 2025.

SPRING. *Spring Data MongoDB Reference Documentation*. Disponível em: https://docs.spring.io/spring-data/mongodb. Acesso em: 30 out. 2025.

TIANGOLO, Sebastián Ramírez. *FastAPI: Modern, fast web framework for building APIs with Python 3.10+*. 2024.

LGPD (Lei nº 13.709/2018). *Lei Geral de Proteção de Dados Pessoais.* Disponível em: https://www.planalto.gov.br.

15. Aprovações

Aprovações		
Participante	**Assinatura**	**Data**
Matheus Silva Ciriaco	Matheus Silva Ciriaco	03/11/2025