

Engenharia de Software

Sistema de Gerenciamento Médico

**Discentes: Karolline de Sena, Ketlen Rebeca, Lara Gomes
Paiva, Lucas Moreira, Paloma Vitoria**

Resumo

Este artigo explica o **Sistema de Gerenciamento Médico (SGM)**, uma aplicação focada em centralizar dados de saúde de pacientes com Doenças Crônicas. O SGM foi construído usando uma arquitetura **híbrida**, que combina dois tipos de banco de dados: **MySQL**, para garantir a segurança e a integridade de dados críticos (como prescrições e medições), e **MongoDB**, usado para registrar logs de auditoria de forma rápida e escalável. O sistema segue o padrão **MVC**, com *backend* em **Java (Spring Boot)** e *frontend* em **JavaScript**.

Palavras-chave: Arquitetura Híbrida, Spring Boot, MySQL, MongoDB, MVC, Engenharia de Software, APIs RESTful.

Sumário

1. Introdução	4
2. Objetivos	4
3. Metodologia.....	4
4. Descrição do Sistema	5
4.1. Funcionalidades por Perfil	5
4.2. Tecnologias Utilizadas e Justificativa	5
5. Modelagem do Banco de Dados	7
5.1. Diagrama Entidade-Relacionamento (DER).....	7
5.2. SQL DDL (Data Definition Language)	8
5.3. Justificativa para Mecanismos de Banco de Dados	11
6. Controle de Acesso	12
6.1. Descrição dos Usuários e Grupos	12
7. Uso do Banco NoSQL	13
7.1. O Banco NoSQL Escolhido: MongoDB	13
7.2. Por que o SGM usa o MongoDB?	13
8. Conclusão	13
9. Referências	14

1. Introdução

O acompanhamento de doenças crônicas (DCNTs) exige organização. Atualmente, os dados dos pacientes (medidas, receitas, exames) ficam espalhados, o que dificulta o trabalho do médico e a adesão do paciente ao tratamento. O SGM foi criado para resolver esse problema, oferecendo uma plataforma única e segura para registrar e consultar essas informações.

2. Objetivos

O objetivo principal é criar um sistema de saúde seguro e eficiente. As metas técnicas específicas são:

- **Confiança Total:** Usar o MySQL para garantir as propriedades **ACID** (Atomicidade, Consistência, Isolamento e Durabilidade), assegurando que os dados clínicos nunca sejam corrompidos.
- **Inteligência no Banco:** Usar recursos do SGBD (Triggers, Procedures) para validar dados importantes na origem, antes que sejam salvos.
- **Arquitetura Híbrida:** Explicar por que usar MySQL (para transações) e MongoDB (para logs) juntos é a melhor solução para o sistema.
- **Segurança de Acesso:** Criar um controle de acesso claro, baseado nos perfis (Paciente, Médico, Admin), para proteger a privacidade dos dados.

3. Metodologia

O sistema foi organizado no padrão **MVC (Model-View-Controller)**. Essa escolha separa as responsabilidades do código:

1. **Model (Modelo):** A camada de dados, incluindo as classes Java (Paciente.java, Medico.java) e os DAOs (PacienteDAO.java, etc.) que acessam o banco.

2. **View (Visão):** A interface do usuário, construída com **JavaScript, HTML e CSS.**
3. **Controller (Controlador):** O **Java (Spring Boot)**, que atua como o "cérebro", recebendo requisições da Visão e orquestrando as ações no Modelo.

A modelagem do banco MySQL seguiu a **Terceira Forma Normal (3NF)** para evitar redundância de dados.

4. Descrição do Sistema

4.1. Funcionalidades por Perfil

O sistema oferece funcionalidades específicas para cada perfil, garantindo que ninguém accesse dados que não deveria.

Tabela 4.1 – Perfis de usuário e suas funcionalidades

Perfil	Acesso Permitido	Funcionalidades Principais
Paciente	Acesso aos seus próprios dados de saúde.	Visualizar lembretes, histórico de medições, visualizar seus medicamentos/prescrições e registrar novas medições.
Médico	Acesso aos dados de Pacientes que ele acompanha (relação N:M).	Gerenciar pacientes, registrar medicamentos e prescrições para o paciente, e registrar medições assistidas.
Administrador	Acesso total (CRUD) no sistema.	Gerenciar todos os usuários (Pacientes, Médicos) e criar/editar novos Administradores.

4.2. Tecnologias Utilizadas e Justificativa

Tabela 4.2 – Tecnologias Utilizadas e Justificativas por Camada

Camada	Tecnologia	Justificativa
Frontend	JavaScript (Vanilla JS), HTML5, CSS3	Acessibilidade: Permite criar uma interface que roda em qualquer navegador (web/mobile), oferecendo uma experiência de uso mais intuitiva para o usuário final.
Backend	Java	Ecossistema Robusto: O Spring Boot facilita a criação de APIs RESTful e gerencia as conexões com o banco (JdbcTemplate), permitindo que a lógica de negócio seja o foco principal.
SGBD Relacional	MySQL (via Spring JdbcTemplate)	Confiança e Integridade: Garante as regras ACID . É a escolha certa para dados que não podem ser perdidos ou corrompidos, como prescrições e dados de pacientes.
SGBD NoSQL	MongoDB	Velocidade e Flexibilidade: Ideal para dados de alto volume e baixa complexidade, como logs de auditoria . Ele grava dados muito rápido sem travar o banco principal (MySQL).

5. Modelagem do Banco de Dados

5.1. Diagrama Entidade-Relacionamento (DER)

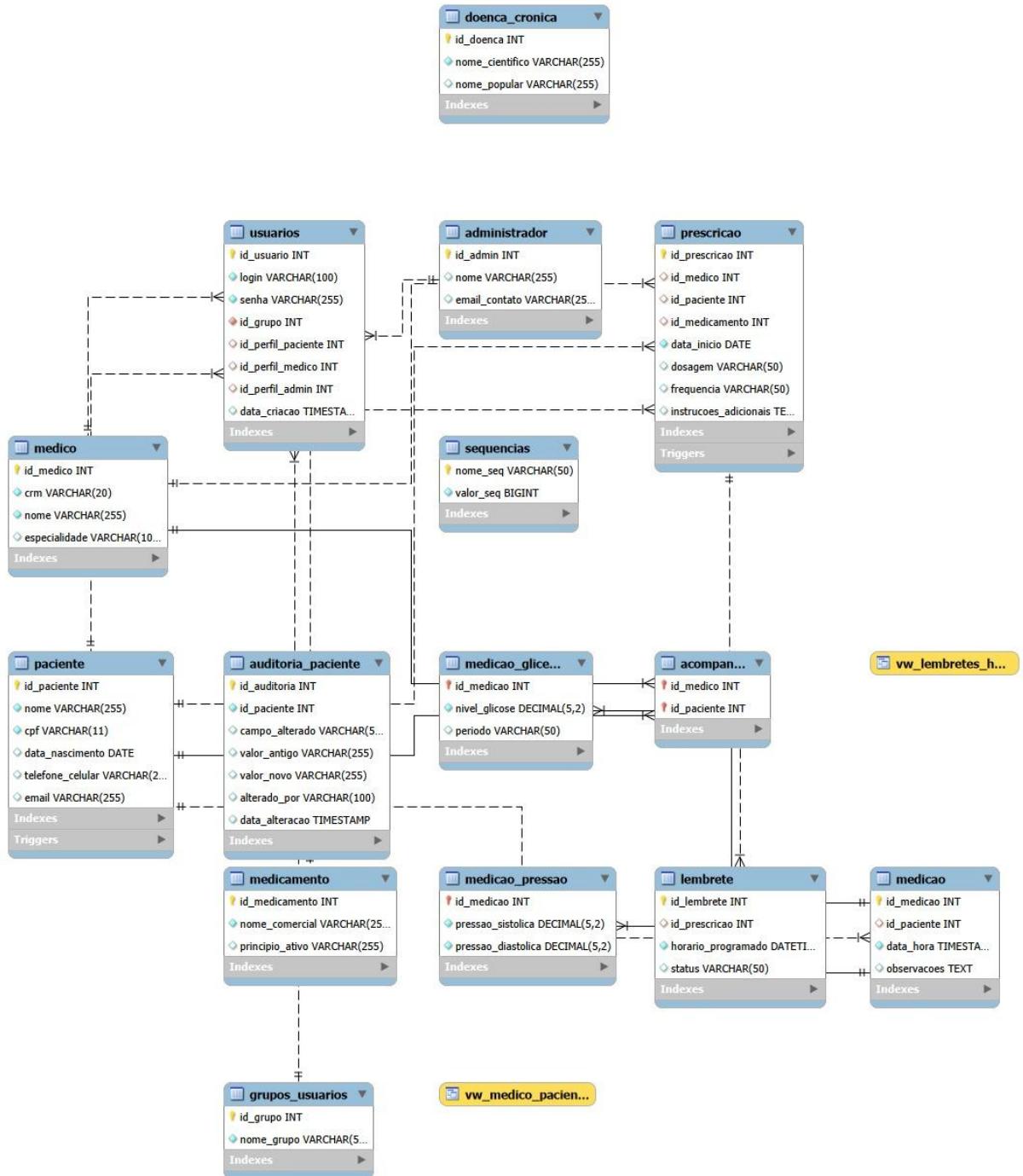


Figura 1 – Diagrama Entidade-Relacionamento do Sistema. Fonte: Lucas Moreira (2025)

O modelo relacional está em 3NF, contemplando a gestão de identificação (paciente, médico), tratamento (prescrição, medicamento) e registros clínicos (medição). A estrutura de autenticação é centralizada nas tabelas *usuarios* e *grupos_usuarios*.

5.2. SQL DDL (Data Definition Language)

O DDL a seguir, define a estrutura lógica implementada no MySQL.

```
-- TABELAS DE AUTENTICAÇÃO E PERMISSÃO (REQUISITO OBRIGATÓRIO)

CREATE TABLE grupos_usuarios (
    id_grupo INT PRIMARY KEY,
    nome_grupo VARCHAR(50) NOT NULL UNIQUE
);

CREATE TABLE usuarios (
    id_usuario INT PRIMARY KEY, -- Usará função de ID customizada
    login VARCHAR(100) NOT NULL UNIQUE,
    senha VARCHAR(255) NOT NULL,
    id_grupo INT NOT NULL,
    id_perfil_paciente INT NULL UNIQUE,
    id_perfil_medico INT NULL UNIQUE,
    id_perfil_admin INT NULL UNIQUE,
    data_criacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_grupo) REFERENCES grupos_usuarios(id_grupo)
);

-- TABELA DE SEQUÊNCIA (REQUISITO: GERAÇÃO DE ID CUSTOMIZADA)

CREATE TABLE sequencias (
    nome_seq VARCHAR(50) PRIMARY KEY,
    valor_seq BIGINT NOT NULL
);

-- TABELAS DE PERFIL E DADOS (SCHEMA PRINCIPAL)

CREATE TABLE paciente (
    id_paciente INT PRIMARY KEY, -- Usará função de ID customizada
    nome VARCHAR(255) NOT NULL,
```

```

        cpf VARCHAR(11) UNIQUE NOT NULL,
        data_nascimento DATE,
        telefone_celular VARCHAR(20),
        email VARCHAR(255)
);

CREATE TABLE medico (
    id_medico INT PRIMARY KEY AUTO_INCREMENT, -- AUTO_INCREMENT
justificado
    crm VARCHAR(20) UNIQUE NOT NULL,
    nome VARCHAR(255) NOT NULL,
    especialidade VARCHAR(100)
);

CREATE TABLE administrador (
    id_admin INT PRIMARY KEY AUTO_INCREMENT, -- AUTO_INCREMENT
justificado
    nome VARCHAR(255),
    email_contato VARCHAR(255)
);

-- Adicionando as FKs de perfil na tabela USUARIOS
ALTER TABLE usuarios
    ADD CONSTRAINT fk_perfil_paciente FOREIGN KEY (id_perfil_paciente)
REFERENCES paciente(id_paciente) ON DELETE CASCADE,
    ADD CONSTRAINT fk_perfil_medico FOREIGN KEY (id_perfil_medico)
REFERENCES medico(id_medico) ON DELETE CASCADE,
    ADD CONSTRAINT fk_perfil_admin FOREIGN KEY (id_perfil_admin)
REFERENCES administrador(id_admin) ON DELETE CASCADE;

-- Tabelas restantes
CREATE TABLE doenca_cronica (
    id_doenca INT PRIMARY KEY AUTO_INCREMENT,
    nome_cientifico VARCHAR(255) UNIQUE NOT NULL,
    nome_popular VARCHAR(255)
);

CREATE TABLE medicamento (
    id_medicamento INT PRIMARY KEY AUTO_INCREMENT,
    nome_comercial VARCHAR(255) NOT NULL,
    principio_ativo VARCHAR(255)
);

CREATE TABLE acompanha (

```

```

id_medico INT,
id_paciente INT,
PRIMARY KEY (id_medico, id_paciente),
FOREIGN KEY (id_medico) REFERENCES medico(id_medico) ON
UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY (id_paciente) REFERENCES paciente(id_paciente) ON
UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE prescricao (
id_prescricao INT PRIMARY KEY AUTO_INCREMENT,
id_medico INT,
id_paciente INT,
id_medicamento INT,
data_inicio DATE NOT NULL,
dosagem VARCHAR(50),
frequencia VARCHAR(50),
instrucoes_adicionais TEXT,
FOREIGN KEY (id_medico) REFERENCES medico(id_medico) ON
UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY (id_paciente) REFERENCES paciente(id_paciente) ON
UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY (id_medicamento) REFERENCES
medicamento(id_medicamento) ON UPDATE CASCADE ON DELETE SET
NULL
);

CREATE TABLE lembrete (
id_lembrete INT PRIMARY KEY AUTO_INCREMENT,
id_prescricao INT,
horario_programado DATETIME NOT NULL,
status VARCHAR(50) DEFAULT 'Pendente',
FOREIGN KEY (id_prescricao) REFERENCES prescricao(id_prescricao)
ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE medicao (
id_medicao INT PRIMARY KEY AUTO_INCREMENT,
id_paciente INT,
data_hora TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
observacoes TEXT,
FOREIGN KEY (id_paciente) REFERENCES paciente(id_paciente) ON
UPDATE CASCADE ON DELETE CASCADE
);

```

```

CREATE TABLE medicao_glicemia (
    id_medicao INT PRIMARY KEY,
    nivel_glicose DECIMAL(5, 2) NOT NULL,
    periodo VARCHAR(50),
    FOREIGN KEY (id_medicao) REFERENCES medicao(id_medicao) ON
    UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE medicao_pressao (
    id_medicao INT PRIMARY KEY,
    pressao_sistolica DECIMAL(5, 2) NOT NULL,
    pressao_diastolica DECIMAL(5, 2) NOT NULL,
    FOREIGN KEY (id_medicao) REFERENCES medicao(id_medicao) ON
    UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE auditoria_paciente (
    id_auditoria INT PRIMARY KEY AUTO_INCREMENT,
    id_paciente INT NOT NULL,
    campo_alterado VARCHAR(50),
    valor_antigo VARCHAR(255),
    valor_novo VARCHAR(255),
    alterado_por VARCHAR(100),
    data.Alteracao TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

5.3. Justificativa para Mecanismos de Banco de Dados

Tabela 5.3 – Justificativa para Mecanismos de Banco de Dados

Mecanismo	Justificativa (Por que é necessário?)
Functions (proximo_id)	Controle de Geração de ID: Cumpre o requisito de IDs personalizados. Em vez de AUTO_INCREMENT (que vaza dados de negócio), o sistema usa a tabela sequencias e a função proximo_id para dar à aplicação controle total sobre a geração de IDs de usuários e pacientes.
Triggers (trg_...)	Integridade de Domínio: São "guardiões" do banco. trg_validar_data_prescricao força regras de negócio (não

	permitir datas passadas) e trg_auditoria_paciente_update cria a trilha de auditoria automaticamente, sem depender do <i>backend</i> .
Views (vw_...)	Otimização de Consultas: Simplificam o trabalho da API. A vw_lembretes_hoje entrega os dados já filtrados e unidos (com <i>JOINS</i>), melhorando a performance e limpando o código Java da camada DAO.
Procedures (sp_...)	Garantia de Atomicidade (ACID): Essencial para operações multi-tabelas, como sp_registrar_novo_paciente (que insere em paciente, usuarios e acompanha). A Procedure garante que a operação ocorra 100% ou seja totalmente desfeita (<i>rollback</i>), mantendo o banco íntegro.
Justificativa do AUTO_INCREMENT	O AUTO_INCREMENT foi mantido apenas em tabelas de "perfil" (como medico e administrador) e em tabelas transacionais (como prescricao), onde o requisito de negócio não exigia um ID customizado, justificando seu uso pela simplicidade na inserção de dados não-críticos.

6. Controle de Acesso

6.1. Descrição dos Usuários e Grupos

O controle de acesso cumpre o requisito de RBAC (Role-Based Access Control) de forma correta e centralizada.

- **Implementação:** O LoginDAO (confirmado pelo código-fonte) implementa a autenticação centralizada. Ele usa uma **consulta única e otimizada** que faz JOIN entre usuarios e grupos_usuarios para validar as credenciais.
- **Segurança:** Esta abordagem é segura, pois o login/senha estão em uma única tabela (usuarios), e flexível, pois os perfis (paciente, medico, admin) são carregados com base no id_grupo.
- **Permissões (GRANTs):** O SGBD também impõe segurança na conexão, usando usuários app_admin, app_medico e app_paciente, cada um com

permissões mínimas necessárias, garantindo que o app_paciente (o mais vulnerável) não possa, por exemplo, executar um DELETE.

7. Uso do Banco NoSQL

7.1. O Banco NoSQL Escolhido: MongoDB

O MongoDB é um SGBD NoSQL orientado a documentos, conhecido pela alta performance de escrita e flexibilidade de esquema (não exige que todos os documentos tenham a mesma estrutura).

7.2. Por que o SGM usa o MongoDB?

O MongoDB foi implementado (confirmado pelo arquivo DiarioMongoDAO.java) para a Trilha de Auditoria e Logs de Eventos.

- **Otimização de Escrita (Write-Heavy):** Logs são dados de "apenas inserção" (*append-only*), gerados em alto volume. O MongoDB é otimizado para essa carga de trabalho, **evitando que o MySQL (focado em transações) sofra com overhead de escrita.**
- **Flexibilidade:** Permite armazenar logs com estruturas variadas (ex: um log de "UPDATE" é diferente de um log de "LOGIN FAILED") sem a necessidade de migrações de esquema, o que é ideal para auditoria.

8. Conclusão

O SGM apresenta uma arquitetura híbrida funcional e robusta. O **MySQL** (acessado via JdbcTemplate) garante a integridade dos dados clínicos. A combinação de **Java/Spring Boot e JavaScript** entrega uma API RESTful e um *frontend* reativo. Finalmente, o uso implementado do **MongoDB** (via DiarioMongoDAO) para auditoria confirma a escalabilidade e rastreabilidade do sistema, atendendo aos requisitos técnicos, funcionais e de segurança do projeto.

9. Referências

ALURA CURSOS ONLINE. **SQL vs NoSQL? | Hipsters Ponto Tech #1.** YouTube, 16 out. 2018. 1 vídeo (48min 10s). Disponível em: https://www.youtube.com/watch?v=a4-Q_g8gX-c. Acesso em: 01 nov. 2025.

BOSON TREINAMENTOS. **MySQL - Triggers - Definição, Sintaxe e Criação - 44.** YouTube, 9 fev. 2014. 1 vídeo (12min 23s). Disponível em: <https://www.youtube.com/watch?v=JOnkvqUaNOU>. Acesso em: 03 nov. 2025.

DEVSUPERIOR. **Introdução a API Web com Java e Spring - Prof. Nelio Alves.** YouTube, 14 out. 2025. 1 vídeo (41min 48s). Disponível em: <https://www.youtube.com/watch?v=wRQSzWTpr6A>. Acesso em: 03 nov. 2025.

GRUPO 6. **Aplicação para Acompanhamento de Doenças Crônicas:** Documento de Visão. Versão 1.0. [S.l.: s.n.], 2025.