# Projeto Integrado – Sistema de Gestão para Consultório Médico

## Introdução

A interseção entre a Engenharia de Software e a Arquitetura de Sistemas constitui o alicerce para a construção de soluções tecnológicas que são não apenas funcionalmente robustas, mas também escaláveis, manteníveis e alinhadas às necessidades do negócio. Este projeto de um Sistema de Gestão para Consultório Médico emerge como um caso prático para a aplicação metodológica dessas disciplinas, traduzindo um problema do mundo real em uma especificação técnica detalhada e viável.

O cenário inicial identificado — marcado por ineficiências no agendamento de consultas, controle manual de estoque e registros clínicos descentralizados — demandava uma abordagem estruturada. Através dos princípios da Engenharia de Software, o projeto percorreu as etapas clássicas do ciclo de vida de desenvolvimento: do levantamento de requisitos e identificação de stakeholders à modelagem de processos e dados. A definição de um Produto Mínimo Viável (MVP) foi crucial para delimitar o escopo, assegurando que as funcionalidades essenciais fossem priorizadas para entregar valor imediato ao consultório.

Paralelamente, a visão da Arquitetura de Sistemas orientou a definição da estrutura técnica que sustentará a solução. A opção por uma arquitetura em camadas, baseada no padrão MVC (Model-View-Controller) e implementada com o ecossistema Spring Boot, reflete uma decisão ponderada que equilibra simplicidade, organização de código e capacidade de evolução futura. Os diagramas UML e o Modelo Entidade-Relacionamento (DER) elaborados não são meras formalidades; eles servem como a espinha dorsal do sistema,

garantindo que todos os envolvidos tenham uma compreensão comum de sua estrutura e comportamento.

Este documento consolida, portanto, o resultado desse processo integrado. Ele apresenta desde a concepção do projeto — com seus objetivos, escopo e critérios de aceitação — até os detalhes de implementação do protótipo, incluindo modelagem, cronograma e estratégia de deploy. A análise aqui contida demonstra como a aplicação rigorosa de fundamentos de engenharia e arquitetura é imprescindível para transformar uma ideia em um sistema software funcional, eficiente e preparado para crescer.

## 1. Definição do Escopo

Nós iniciamos nosso projeto pela definição do escopo, etapa fundamental para entender exatamente o que precisaríamos desenvolver e quais problemas queríamos resolver. Primeiro, identificamos a situação-problema do consultório médico: percebemos que havia longas filas no agendamento de consultas, dificuldade no controle de estoque de medicamentos e materiais, além da falta de registros clínicos centralizados, o que tornava o acompanhamento dos pacientes pouco eficiente.

Com essas observações, conseguimos estabelecer os limites do projeto, ou seja, o que seria tratado pelo nosso sistema de gestão e o que ficaria fora do escopo. Também definimos quais seriam os entregáveis: um sistema capaz de organizar o agendamento de consultas, gerenciar o estoque de maneira eficiente e centralizar os registros clínicos dos Definição dos Objetivos do Sistema

### 1.1 Definição dos Objetivos do Sistema

O sistema deverá permitir o agendamento online de consultas, de forma a reduzir filas e otimizar o tempo dos pacientes e da equipe. Além disso, oferecerá

um controle básico de estoque, permitindo que medicamentos e materiais sejam registrados, monitorados e atualizados de maneira prática.

### 1.2 Definição do MVP (Produto Mínimo Viável)

Depois de estabelecer os objetivos do sistema, nossa próxima etapa foi definir o MVP, ou seja, o Produto Mínimo Viável. O objetivo aqui era identificar as funcionalidades essenciais que precisávamos entregar para que o sistema já fosse utilizável e atendesse às principais necessidades do consultório.

Após discutirmos em grupo, decidimos que o MVP incluiria:

- **o cadastro de pacientes,** para organizar as informações de cada pessoa atendida;
- **o cadastro de profissionais**, permitindo o registro dos médicos e demais colaboradores;
- **o agendamento de consultas**, garantindo que os pacientes pudessem marcar atendimentos de forma organizada;
- **e a geração de dois relatórios**, que dariam uma visão resumida sobre o fluxo de consultas e o controle de estoque.

Definir o MVP nos ajudou a concentrar esforços no que era realmente prioritário, garantindo que, mesmo em uma versão inicial, o sistema já tivesse valor prático para o consultório.

### 1.3 Identificação de Stakeholders e Papéis

Na sequência, nos dedicamos a identificar os stakeholders do sistema e definir os papéis de cada um dentro do contexto do consultório. Para nós, era importante compreender quem seriam os usuários e quais responsabilidades cada um teria para que o sistema atendesse às suas necessidades de forma eficiente.

Identificamos os principais stakeholders:

- **os pacientes**, que fariam o agendamento de consultas e teriam acesso aos seus dados médicos;
- **os recepcionistas**, responsáveis por gerenciar os atendimentos e controlar o fluxo de informações;
- **os médicos**, que precisariam acessar os registros clínicos e atualizar informações durante os atendimentos;
- **o administrador do consultório**, que teria uma visão completa do sistema e poderia gerar relatórios gerenciais;
- **os fornecedores**, que seriam impactados pelo controle de estoque e pelas demandas de materiais.

Com essa definição, conseguimos mapear claramente quem interage com o sistema, quais funções cada usuário teria e como cada grupo seria beneficiado, garantindo que o projeto fosse centrado nas pessoas que realmente utilizariam a ferramenta.

### 1.4 Definição de Critérios de Aceitação e KPIs

Em seguida, avançamos para a definição dos critérios de aceitação do sistema e das perguntas de negócio que ele precisaria responder. Essa etapa foi essencial para garantir que o sistema realmente atendesse às expectativas do consultório e pudesse ser validado de forma objetiva.

Para isso, estabelecemos indicadores de desempenho (KPIs) que seriam monitorados pelo sistema. Por exemplo, queríamos saber qual profissional atende mais pacientes por semana, quantas consultas são realizadas por período, se o estoque está adequado e quais materiais precisam ser repostos com mais urgência.

Esses critérios nos ajudaram a definir o que seria considerado sucesso na entrega do projeto. Com eles, conseguimos medir se o sistema estava

cumprindo seu papel e fornecendo informações relevantes para a tomada de decisão do consultório e pacientes. Essa etapa foi crucial para que todos do grupo tivessem clareza sobre os objetivos e a direção do projeto.

## 2. Levatamento de Requisitos

Os requisitos foram levantados com base nas necessidades do consultório e priorizados de acordo com a relevância.

- **Requisitos Funcionais (RF):** cadastrar pacientes, registrar agendamento, gerar relatórios.
- **Requisitos Não-Funcionais (RNF):** tempo de resposta ≤ 10s, usabilidade, disponibilidade, segurança de dados (LGPD).
- **Priorização:** foi utilizada a técnica MoSCoW.
- **User stories:** “Como recepcionista, quero cadastrar pacientes para organizar o atendimento.”

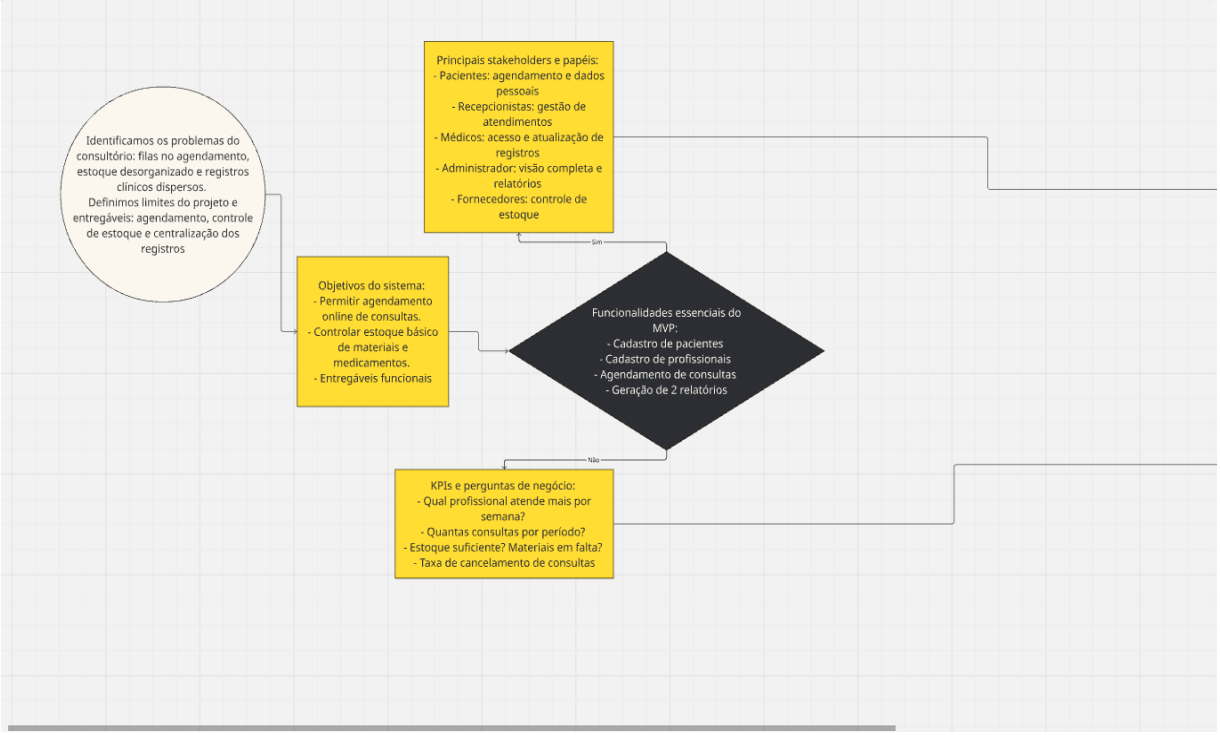


***Figura 1: Definição do escopo e levantamento de requisitos do sistema***

## 3. Modelagem do Sistema

Objetivo: representar graficamente a estrutura de dados e o comportamento do sistema via diagramas (Casos de Uso, Classe, Atividades, DER).

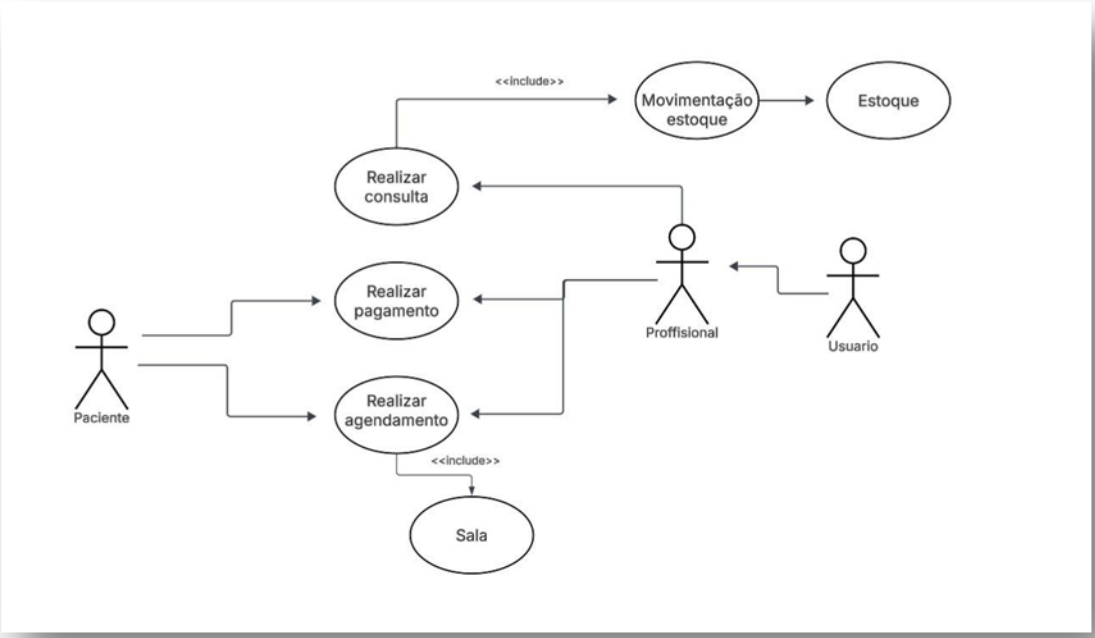


***Figura 2: Diagrama de caso de uso***

O diagrama representa o funcionamento básico de um sistema de consultório. Nele aparecem três atores principais:

- **Usuário**: funcionário ou pessoa de apoio que auxilia nas operações administrativas do consultório.
- **Profissional**: usuário do sistema que exerce funções mais técnicas, podendo ser médico, dentista, enfermeiro ou outro especialista responsável por atender pacientes.
- **Paciente**: pessoa que busca atendimento no consultório e que pode interagir diretamente com o sistema.

## Relações com os Casos de Uso

### Paciente

O paciente pode interagir diretamente com o sistema, sem precisar passar por uma secretária. Ele pode realizar agendamentos online, registrar pagamentos ou simplesmente acessar suas consultas marcadas.

- **Realizar consulta**

Durante a consulta realizada pelo profissional, o sistema pode registrar a utilização de materiais. Isso gera a movimentação de estoque, garantindo o controle de insumos usados no atendimento

- **Realizar agendamento**

  O paciente pode agendar uma consulta online, escolhendo data e horário. Sempre que um agendamento é feito, o sistema precisa reservar uma sala correspondente

- **Realizar pagamento**

  O pagamento pode ser feito diretamente pelo paciente, seja online ou presencial, e registrado no sistema para manter o histórico financeiro do atendimento.

O sistema permite que o paciente tenha autonomia para agendar consultas online e realizar pagamentos, sem depender exclusivamente de um funcionário. O profissional, por sua vez, atua durante as consultas e faz registros que podem impactar o estoque. Já o usuário de apoio auxilia nas tarefas administrativas.

Assim, o diagrama mostra como cada ator se conecta ao sistema para organizar atendimentos, controlar salas, manter registros financeiros e acompanhar o estoque.

## Explicação do Diagrama UML de Classes

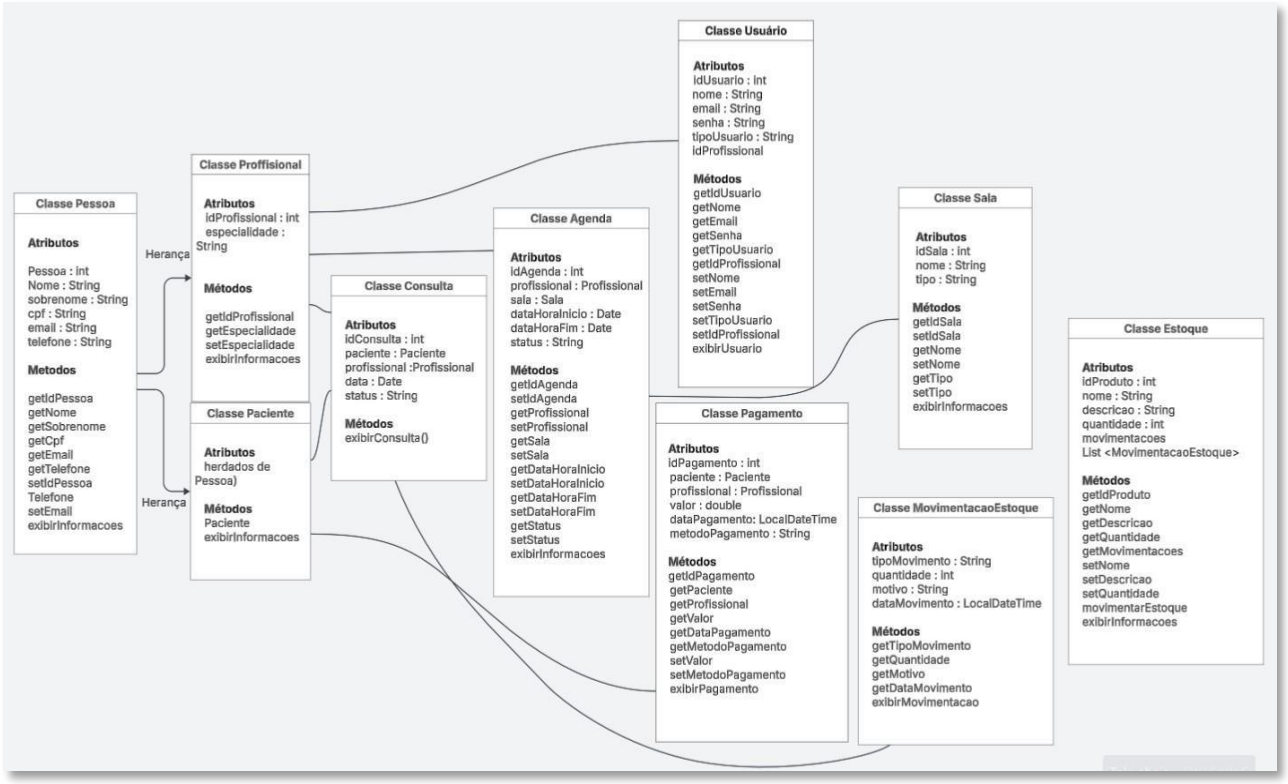


***Figura 3: Diagrama de Classes e Atributos***

O diagrama mostra como o sistema do consultório foi estruturado em várias classes que se relacionam entre si.

**Pessoa**

É a classe mais genérica. Tem atributos como idPessoa, nome, sobrenome, cpf, email e telefone.

A ideia é que tanto Paciente quanto Profissional herdem essas informações, já que todo mundo precisa ter esses dados básicos cadastrados.

**Paciente**

Representa quem busca atendimento. Não adiciona atributos novos além dos herdados de Pessoa, mas pode sobrescrever métodos, como exibirInformacoes.

**Profissional**

Também herda de Pessoa, mas tem atributos próprios como idProfissional e especialidade. Assim, é possível identificar que tipo de profissional ele é (médico, psicólogo, dentista etc.).

**Usuário**

Representa quem acessa o sistema (pode ser recepcionista, administrador ou até um profissional). Tem atributos como idUsuario, nome, email, senha, tipoUsuario e idProfissional (caso o usuário também seja um profissional da clínica).

**Consulta**

É o registro de um atendimento, ligando Paciente e Profissional. Tem atributos como idConsulta, data e status. O método principal mostrado é exibirConsulta

**Pagamento**

Está vinculado à consulta e guarda informações financeiras. Seus atributos são idPagamento, valor, dataPagamento, metodoPagamento, além da referência a paciente e profissional.

**Agenda**

É responsável por organizar os horários das consultas. Possui atributos como idAgenda, dataHoraInicio, dataHoraFim, status, além de vínculos com um Profissionale uma Sala.

**Sala**

Representa o local físico onde ocorre a consulta. Tem atributos simples como idSala, nomee

tipo(por exemplo: consultório, sala de exame, sala de procedimentos).

**Produto**

Refere-se aos itens do estoque, como materiais e medicamentos. Tem atributos como

idProduto, nome, descricao, quantidade e uma lista de movimentacoes. Possui métodos como movimentarEstoque para registrar entradas e saídas.

**MovimentacaoEstoque**

Registra cada alteração no estoque. Seus atributos são tipoMovimento (entrada ou saída), quantidade, motivo e dataMovimento. Isso permite rastrear todo o histórico de uso de produtos.

O paciente (com dados de Pessoa) agenda uma **Consulta** com um **Profissional**, em uma **Sala**, por meio da **Agenda**.

Na consulta, pode haver uso de materiais, que geram **Movimentações de Estoque** em cima dos **Produtos**.

Depois, o paciente realiza um **Pagamento**, que fica vinculado tanto ao profissional quanto ao atendimento.

Tudo isso é controlado pelo **Usuário** do sistema, que pode ter diferentes tipos de acesso.

## Revisão e Modelagem do Banco de dados DER e UML

Nesta etapa, nosso grupo realizou a revisão do modelo de dados e dos diagramas já existentes, com o objetivo de garantir que o DER e os diagramas UML representem fielmente as funcionalidades implementadas no protótipo.

Primeiro, analisamos o diagrama de classes UML já elaborado, que apresenta a herança entre Pessoa, Paciente e Profissional, além das classes Usuário, Consulta, Pagamento, Agenda, Sala, Produto e Movimentação de Estoque. Esse diagrama mostrou-se consistente com as operações previstas no sistema: cadastro de pacientes/profissionais, agendamento de consultas, controle de salas e estoque, e registro de pagamentos.

Em seguida, redesenhamos o DER a partir do banco de dados do protótipo, representando as tabelas e seus relacionamentos. O DER evidencia como as

entidades Paciente, Profissional, Sala, Consulta, Produto e Usuário se conectam. Por exemplo:

Uma Consulta referência obrigatoriamente um paciente e um profissional, e pode estar vinculada a uma sala.

O Produto é movimentado através da entidade Movimentação de Estoque, que registra entradas e saídas.

O Pagamento está relacionado diretamente a uma consulta e ao paciente.

Essa revisão nos permitiu alinhar o modelo conceitual UML com o modelo lógico DER, confirmando que ambos representam as mesmas funcionalidades do protótipo.

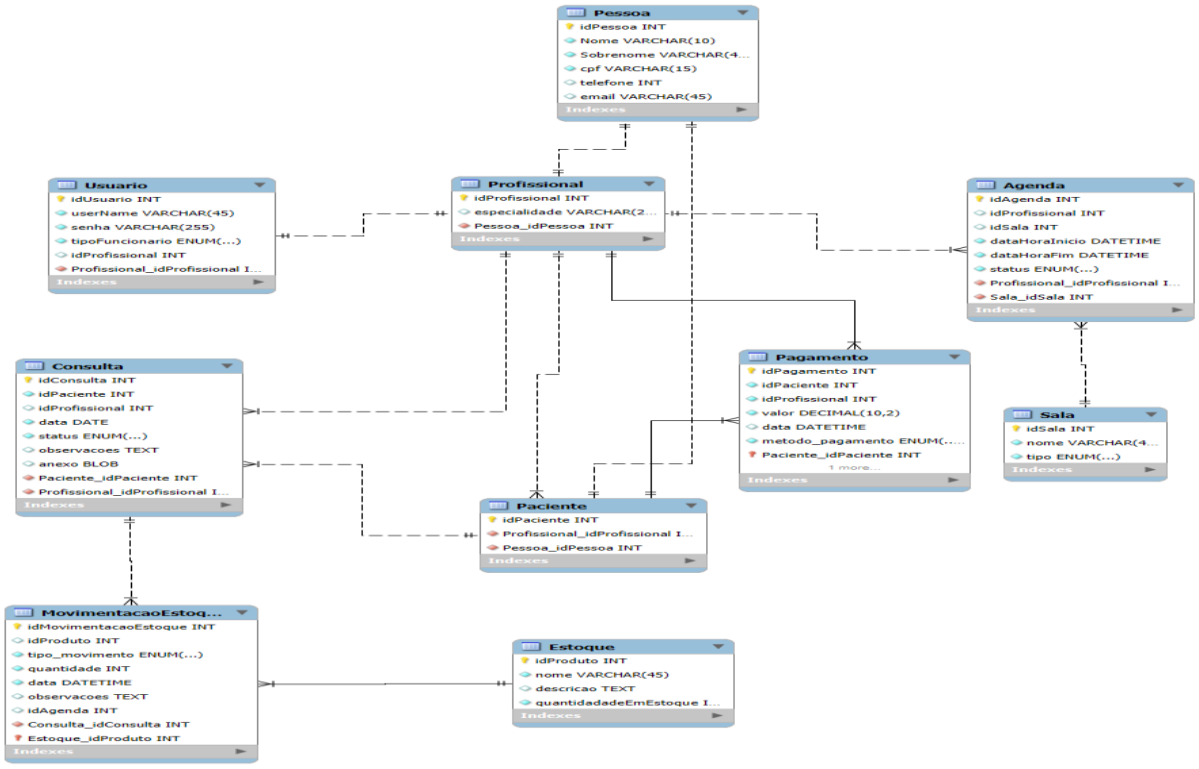


***Figura 4: Modelo DER***

## 4. Arquitetura de Sistemas

A arquitetura escolhida é baseada no padrão MVC (Model-View-Controller). Essa estrutura facilita a manutenção, garante separação de responsabilidades e permite evolução futura para um modelo SaaS.

Para organizar o desenvolvimento, adotamos uma **arquitetura em camadas**, com separação clara de responsabilidades:

- **Frontend (Apresentação):** interface que será utilizada por pacientes, recepcionistas, médicos e administradores. Exibirá telas de cadastro, agendamento e relatórios.
- **API / Controle (Controller):** responsável por receber as requisições do frontend e direcionar para as regras de negócio.
- **Serviço / Negócio:** onde ficam as regras principais do sistema, como lógica de agendamento, atualização de estoque e geração de relatórios.
- **Persistência (DAO/Repository):** camada que se comunica diretamente com o banco de dados para salvar e buscar informações.
- **Banco de Dados (MySQL):** armazenamento de cadastros (pacientes, profissionais, consultas) e registros de estoque.

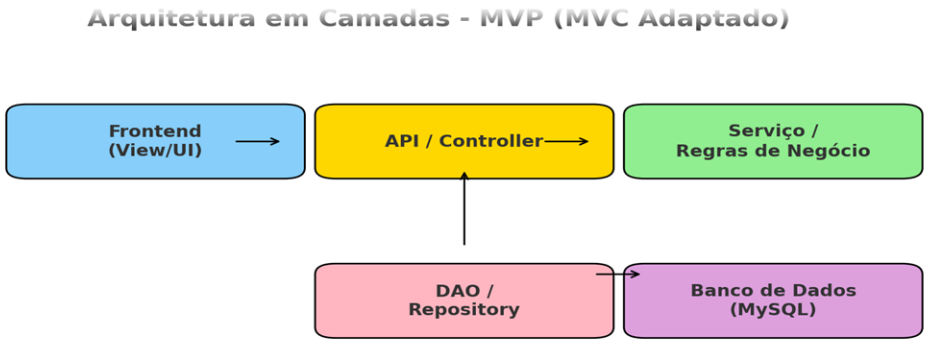


***Figura 5 – Arquitetura em Camadas do MVP (MVC Adaptado)***

**Padrão Arquitetural**

Para o MVP, utilizaremos um monólito modular baseado no padrão MVC (Model-View-Controller)**.**
Essa escolha é adequada porque:

- Simplifica a entrega inicial do sistema.
- Garante boa separação entre interface, regras de negócio e dados.
- Permite evolução futura para uma arquitetura de microserviços, caso o sistema cresça.

**Autenticação e Autorização**

No MVP, será implementado um login básico com senhas criptografadas **(bcrypt)**.
Os papéis definidos serão:

- **Paciente:** pode agendar e visualizar suas consultas.
- **Recepcionista:** pode gerenciar consultas e atualizar cadastros.
- **Médico:** acessa registros clínicos e atualiza informações dos atendimentos.
- **Administrador:** visão geral, incluindo relatórios de estoque e consultas.

**Estratégia de Deploy e Infraestrutura**

- Desenvolvimento inicial em **ambiente local**, utilizando **Docker** para padronizar a execução do backend e banco de dados.
- Organização dos ambientes: **DEV** (desenvolvimento) e **PROD** (produção mínima).
- Infraestrutura planejada para migração futura para nuvem (AWS, Azure ou Railway).

**Tecnologias Definidas**

- **Backend:** Java 17 + Spring Boot, com JPA/Hibernate para persistência.
- **Banco de Dados:** MySQL, administrado via MySQL Workbench
  **Frontend:** Thymeleaf (renderização server-side) no MVP, para simplificar a entrega inicial. Em versões futuras, pode-se evoluir para React (SPA) com gráficos.
- **Ferramentas:** GitHub para controle de versão, Postman para testes de API e Swagger para documentação automática.

## 5. Cronograma detalhado (4 semanas)

- **Semana 1:** Levantamento de requisitos, casos de uso e DER. *Entregáveis:* Documento de requisitos, backlog inicial, DER.
- **Semana 2:** Modelagem detalhada e configuração do ambiente. *Entregáveis:* Diagramas, repositório Git, banco criado, especificação da API.
- **Semana 3:** Implementação parcial (MVP). *Entregáveis:* CRUDs básicos, API de agendamento.
- **Semana 4:** Relatórios, testes e documentação final.

## 6. Desenvolvimento do Protótipo

- **Backend:** desenvolvido em Java puro (sem *Spring Boot*), organizado em classes e pacotes para refletir o padrão MVC.

- **Banco de Dados: MySQL Workbench**, com criação de tabelas a partir de scripts (schema.sql) e manipulação simulada em ambiente local.
- **Frontend:** interface simples implementada em Java, com menus e saídas em terminal (foco acadêmico e funcional).

- **Relatórios:** gerados a partir de consultas simuladas ao banco e exibidos em tela ou exportados em formato simples.

## 7. Testes e Qualidade

Foram realizados testes básicos diretamente no código Java, validando as principais regras de negócio, como:

- Evitar agendamento duplicado no mesmo horário.
- Conferir disponibilidade de salas.
- Validar atualização de estoque após movimentação.

## 8. Deploy e Infraestrutura

O sistema foi projetado para execução local, sem necessidade de servidores externos.

- IDE utilizada: Visual Studio Code.
- Banco de dados**:** MySQL Workbench em ambiente local.
- Versionamento: GitHub para armazenamento do código-fonte e documentação.

## 9. Documentação e entrega final

O pacote de entrega inclui:

- Documento de escopo e requisitos.
- Diagramas UML e DER.
- Scripts SQL.
- Manual do usuário.

- Relatório de testes.
- Plano de melhorias futuras.

**10. Estrutura de pastas**

Projeto_Consultorio/ ├── src/ │ └── main/ │ └── java/ │ └── br/ │ └── com/ │ └── projetoconsultorio/ │ ├── controller/ │ ├── dao/ │ ├── model/ │ └── view/ ├── bin/ # Compilados (.class) ├── lib/ # Bibliotecas externas (opcional) ├── docs/ # Documentação do projeto │ └── documento_teorico.md # Documento único contendo todo o trabalho teórico └── README.md # Este arquivo

## 11. Boas práticas

- Commits pequenos e descritivos.
- Documentar decisões arquiteturais.
- Priorizar os requisitos do MVP.
- Utilizar bibliotecas consolidadas (*Spring*, JPA, Chart.js).

**12. Relatório e Análise de Negócio**

Nesta etapa do projeto, utilizamos o protótipo desenvolvido para simular e ilustrar relatórios essenciais à gestão de um consultório médico. Esses relatórios permitem visualizar o desempenho dos profissionais, o uso das salas disponíveis e o controle do estoque de insumos, fornecendo informações fundamentais para a tomada de decisões estratégicas e operacionais.

Este relatório apresenta a quantidade de consultas realizadas por cada profissional do consultório. Os dados revelam que o Dr. Pedro realizou significativamente mais atendimentos do que a Dra. Ana. Esse cenário pode indicar sobrecarga de trabalho em um único profissional, o que pode levar a cansaço, diminuição na qualidade do atendimento e insatisfação dos pacientes.

Decisão de negócio: Recomenda-se redistribuir os pacientes de forma mais equilibrada entre os profissionais, garantindo melhor aproveitamento da equipe e qualidade no atendimento.

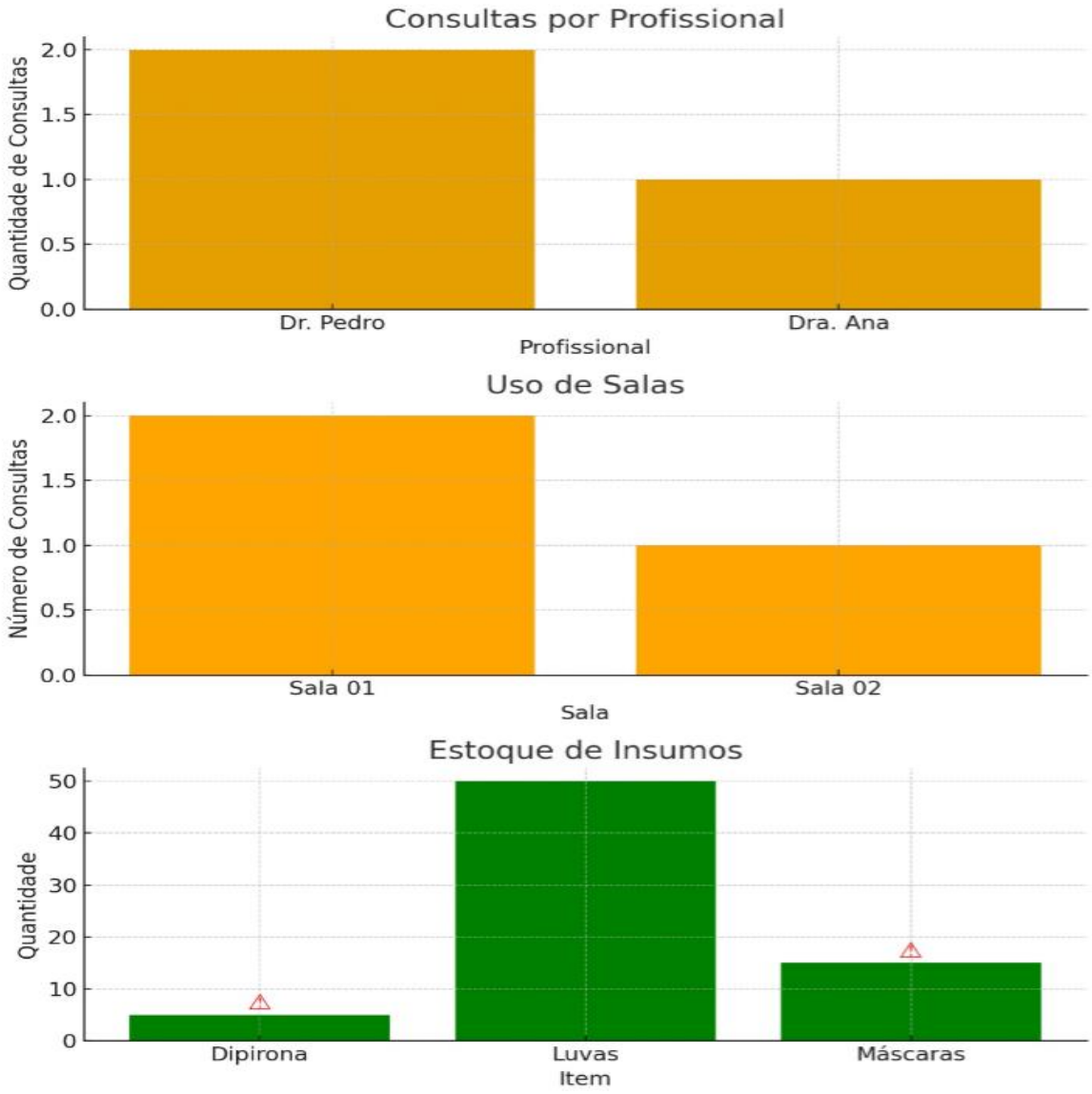


***Figura 3 : Análise gráfica das simulações testes***

Através das simulações observa-se também o nível atual dos principais insumos utilizados no consultório. Verificou-se que alguns itens, como Dipirona e Máscaras, encontram-se abaixo do ponto de reposição definido. A falta de

controle no estoque pode comprometer o atendimento e gerar problemas operacionais.

Decisão de negócio: Recomenda-se repor imediatamente os itens críticos e implementar rotinas periódicas de monitoramento de estoque, garantindo que os materiais essenciais estejam sempre disponíveis.

A partir dos relatórios gerados pelo protótipo, é possível observar a importância da análise de dados na gestão de um consultório médico. O excesso de atendimentos por um profissional sugere a necessidade de redistribuição da carga de trabalho; e a identificação de insumos abaixo do nível mínimo garante que o consultório não enfrente problemas de desabastecimento.

Portanto, a adoção desses relatórios no dia a dia contribui para um processo de tomada de decisão mais assertivo, resultando em maior eficiência operacional, melhor aproveitamento de recursos e aumento na qualidade dos serviços prestados aos pacientes.

## 13. Próximos Passos do Projeto

### 13.1 Consolidar o Protótipo

- Finalizar os **CRUDs** (cadastro, consulta, atualização e exclusão) de todas as entidades: Paciente, Profissional, Consulta, Sala, Estoque e Usuário.
- Garantir que as validações principais estejam implementadas (ex.: evitar choque de horários no agendamento).
- Criar menus simples de navegação para que qualquer pessoa consiga testar.

### 13.2 Melhorar o Banco de Dados

- Ajustar as tabelas no **MySQL Workbench** para refletirem todos os relacionamentos definidos no DER.
- Inserir dados fictícios (pacientes, médicos, insumos) para facilitar os testes e relatórios.
- Testar consultas SQL que correspondam aos relatórios (ex.: consultas por profissional, estoque em falta).

### 13.3 Gerar Relatórios e Simulações

- Implementar no código Java consultas que puxem dados do banco e gerem relatórios simulados em texto ou tabela.
- Relacionar cada relatório a uma decisão de negócio (ex.: “Se um médico está atendendo mais que os outros, é preciso redistribuir pacientes”).

### 13.4 Documentar o Projeto

- Revisar o documento final (escopo, requisitos, modelagem, arquitetura, cronograma).
- Inserir prints do **MySQL Workbench** (tabelas e consultas).
- Inserir prints do código Java em execução (menus, cadastros, relatórios).

## 14. Evoluções Futuras (Roadmap)

- Migrar para interface gráfica (JavaFX ou Swing).
- Criar uma API simples para separar frontend e backend.
- Pensar em deploy em nuvem com Docker.
- Autenticação de usuários com níveis de permissão.

## 15. Conclusão

O sistema de gestão para consultório médico foi planejado com base em princípios de Engenharia de Software, utilizando modelagem, arquitetura em camadas (MVC), levantamento de requisitos e prototipagem. O MVP foi atendido com cadastros, agendamento e relatórios básicos.

O trabalho demonstrou como organizar o desenvolvimento em etapas, integrar ferramentas simples (*Java*, MySQL, Docker, GitHub) e entregar valor prático ao usuário.
A estrutura proposta ainda permite evolução futura para SaaS, relatórios mais avançados e integração em nuvem.