Medical Clinic

Aline Vitória Santana Nunes[2], Alvaro George[2], Beatriz Fernandes[2], Isis Nascimento de Lavor[1], Marcos Grégory Rodrigues Marques[1]

Orientadora: Profa. Dsc. Jacilane de Holanda Rabelo[2]

[1] Ciência da Computação
[2] Engenharia de Software

Universidade Federal do Ceará (UFC) – Ceará – CE – Brasil

{alinevitoria, alvarogeorge, beatrizfernandes, isislavor, marcosvdcdef}@alu.ufc.br

1. Introdução do projeto

O presente projeto tem como objetivo o desenvolvimento de um sistema de gerenciamento de serviços de clínica médica, denominado *Medical Clinic*. A iniciativa busca atender à necessidade de modernização e eficiência nos processos administrativos e operacionais de clínicas, que frequentemente enfrentam dificuldades relacionadas ao controle de atendimentos, organização de agendas, registro de pacientes e acompanhamento de procedimentos. A ausência de sistemas integrados compromete a qualidade do atendimento e aumenta a probabilidade de falhas, como marcações conflitantes, perda de informações relevantes e retrabalho. Tais problemas impactam diretamente a experiência do paciente e a produtividade da equipe, revelando a importância de uma solução tecnológica confiável e eficaz. O sistema *Medical Clinic* será desenvolvido como uma aplicação de gestão clínica, com funcionalidades voltadas para o cadastro de pacientes, controle de agendamentos, gerenciamento de profissionais de saúde, acompanhamento de consultas e geração de relatórios. Suas principais características incluem interface intuitiva, integração de dados e suporte a processos essenciais da rotina clínica. A adoção do *Medical Clinic* proporciona benefícios significativos, como a centralização das informações, a redução de erros administrativos, a otimização do tempo dos profissionais e a melhoria do atendimento ao paciente. Dessa forma, espera-se que o sistema contribua para elevar o padrão de organização e eficiência no gerenciamento de clínicas médicas.

1.2. Objetivos do projeto

O objetivo geral deste projeto é desenvolver um sistema informatizado de gerenciamento de serviços de clínica médica, denominado *Medical Clinic*, destinado a otimizar os processos administrativos e assistenciais, de modo a garantir maior

eficiência, organização e qualidade no atendimento aos pacientes. Como objetivos específicos, destacam-se:

- Automatizar o cadastro e a gestão de informações dos pacientes;
- Implementar um módulo de agendamento de consultas que permita maior controle e organização da agenda dos profissionais;
- Disponibilizar funcionalidades para o gerenciamento de profissionais de saúde e seus atendimentos;
- Proporcionar o acompanhamento de consultas e procedimentos realizados;
- Gerar relatórios administrativos e clínicos que subsidiem a tomada de decisão;
- Reduzir falhas e retrabalho decorrentes de processos manuais ou pouco integrados;
- Melhorar a experiência do paciente por meio de processos mais ágeis e confiáveis.

O conjunto desses objetivos visa assegurar uma solução tecnológica completa para a gestão de clínicas médicas, promovendo maior produtividade, qualidade e eficácia no atendimento em saúde.

1.3. Delimitação do problema e reutilização de software

O presente projeto delimita-se ao desenvolvimento e aperfeiçoamento do sistema *Medical Clinic*, atualmente em estágio de **Produto Mínimo Viável (MVP)**. O sistema contempla, em sua versão inicial, funcionalidades voltadas para dois perfis de usuários principais: **médicos** e **pacientes**. Entre os recursos já implementados, destacam-se o agendamento e marcação de consultas, a prescrição digital de medicamentos e o envio de exames médicos por meio de arquivos anexados.Embora o sistema em sua forma atual já atende a necessidades essenciais do gerenciamento clínico, observa-se que sua abrangência pode ser ampliada a partir da incorporação de novos módulos e da reutilização de software previamente desenvolvido. Assim, busca-se não apenas manter as funcionalidades já consolidadas, mas também promover sua evolução, priorizando a reutilização de componentes existentes sempre que possível, a fim de reduzir custos de desenvolvimento, garantir maior confiabilidade e acelerar a entrega de novas soluções. Dessa forma, o escopo do projeto é delimitado à utilização integral do núcleo já implementado no *Medical Clinic*, com ênfase nos seguintes aspectos:

- Manutenção e consolidação das funcionalidades de cadastro de pacientes, agendamento de consultas, prescrição digital e envio de exames;
- Reutilização de módulos já existentes para dar suporte à expansão do sistema;

- Ampliação das funcionalidades com vistas a aprimorar a experiência de médicos e pacientes, tais como relatórios clínicos, histórico integrado de atendimentos e acompanhamento de tratamentos.

Assim, o ponto central do projeto concentra-se na consolidação do sistema como ferramenta robusta de gestão clínica, utilizando de forma estratégica a reutilização de software para garantir eficiência, confiabilidade e sustentabilidade no desenvolvimento.

1.4. Justificativa da escolha do tema

A escolha do tema justifica-se pela crescente demanda por soluções tecnológicas voltadas à área da saúde, especialmente no que diz respeito ao gerenciamento de serviços em clínicas médicas. O uso de sistemas informatizados neste contexto apresenta-se como fator essencial para a melhoria da qualidade do atendimento, a organização administrativa e a otimização dos recursos disponíveis. Do ponto de vista prático, observa-se que clínicas de pequeno e médio porte frequentemente enfrentam dificuldades na gestão de informações, no controle de consultas e na comunicação entre médicos e pacientes. Essas limitações podem comprometer a eficiência dos serviços prestados e a satisfação dos usuários. O desenvolvimento do *Medical Clinic* surge, portanto, como resposta a essas necessidades, oferecendo uma plataforma capaz de centralizar processos, reduzir falhas e ampliar a confiabilidade do atendimento. Sob a perspectiva acadêmica, o projeto representa a oportunidade de aplicar conhecimentos teóricos adquiridos ao longo da formação, envolvendo áreas como engenharia de software, banco de dados, interfaces de usuário e boas práticas de desenvolvimento orientado a objetos. Além disso, permite explorar metodologias de reúso de software, que se configuram como estratégia relevante na Engenharia de *Software* moderna para redução de custos, incremento da produtividade e aumento da confiabilidade dos sistemas. Dessa forma, a elaboração do *Medical Clinic* encontra justificativa tanto na relevância prática para o setor de saúde quanto na contribuição acadêmica para a consolidação de competências técnicas e científicas, sendo um projeto de impacto significativo em ambas as dimensões.

1.5. Organização do trabalho

O presente documento está estruturado de forma a apresentar, de maneira clara e sistemática, todas as etapas que compõem o desenvolvimento do projeto. Inicialmente, é exposta a **Introdução do Projeto**, na qual se contextualizam os objetivos e a relevância da proposta. Em seguida, a **Descrição Geral do Sistema** fornece uma visão ampla de suas funcionalidades e características principais. A **Documentação de Casos de Uso** detalha os cenários de interação entre os usuários e o sistema, servindo de base para a modelagem. Na sequência, são apresentados o **Diagrama de Classes** e o **Diagrama de Sequência**, que representam, respectivamente, a estrutura estática e o comportamento dinâmico do sistema. As **Conclusões** sintetizam os resultados obtidos e as contribuições do projeto. Complementam o documento a **Ata de reuniões**, que registra o acompanhamento do desenvolvimento, e as **Referências**, que reúnem as fontes bibliográficas e técnicas utilizadas.

2. Descrição geral do sistema

O sistema **MedicalClinic** é um aplicativo desktop (**Java** + **Swing**) que apoia o fluxo ambulatorial de uma clínica: autenticação, agenda de consultas, registro de consultas (com descrição do médico), prontuário do paciente (histórico de consultas), e anexação/"*download*" de exames como arquivos. As telas estão organizadas em login/registro, *home* do paciente (agendar consulta, listar consultas, visualizar detalhes e exames) e home do médico (lista de pacientes, detalhamento do paciente, registro/edição de consultas e exames). A arquitetura segue **MVC** com *controllers* específicos por tela, *views* Swing e modelos para usuários, consultas, exames e prontuário, além de um *SystemModel* central (*in-memory*) e um mecanismo de *Observer* para atualização das telas.

2.1. Principais envolvidos e suas características

2.1.1. Usuários do sistema

Tipo de organização atendida: clínicas médicas ambulatoriais de pequeno e médio porte (multi-especialidades) que precisam controlar consultas e prontuários de forma simples, local (sem servidor) e com anexos de exames. Os seguintes papéis e perfis dos usuários são:

Paciente
• **Necessidades**: agendar consultas com médico e data, consultar histórico, abrir detalhes de uma consulta, acessar/baixar exames anexados. • **Habilidades previsíveis**: uso básico de computador; leitura de telas simples; não precisa conhecer termos técnicos. • **Tarefas principais**: agendar consulta, ver consultas futuras e passadas, visualizar descrição e exames.

Médico
• **Necessidades**: visualizar seus pacientes, abrir o prontuário, **criar/atualizar** consultas (data, descrição, status ativo/inativo), **anexar exames** (arquivo) e disponibilizá-los. • **Habilidades previsíveis**: uso moderado de computador; leitura técnica de dados clínicos; foco em produtividade.

Médico
• **Tarefas principais**: registrar/editar consultas, adicionar exames, revisar histórico do paciente.

Atendente/Recepção
• **Necessidades:** apoiar cadastro inicial e agendamentos. • **Observação:** o sistema, como está, **não** possui telas dedicadas à recepção; o papel pode ser exercido pelo próprio paciente/médico.

2.2. Regras de Negócio

ID	Regra
`RN01`	A autenticação ocorre por login (nome de usuário) + senha existentes no `SystemModel`; se vazios ou inválidos, a entrada é negada com mensagem.
`RN02`	Após autenticação, o usuário é roteado conforme **papel**: `Doctor` → `DoctorHomeView`; `Patient` → `PatientHomeView`.
`RN03`	Cadastro cria Paciente ou Médico; exige campos obrigatórios; proíbe duplicidade de login; para Médico, especialidade é obrigatória.
`RN04`	Paciente agenda consulta escolhendo médico e data no formato `dd/MM/yyyy HH:mm`; data inválida gera erro.
`RN05`	Uma consulta pertence a um paciente e a um médico; ao agendar, o paciente é adicionado à lista de pacientes do médico.
`RN06`	A consulta nasce ativa (`active=true`) e pode ter descrição livre, padrão “Não adicionado pelo médico!” até o médico editar.
`RN07`	O prontuário de um paciente agrega o histórico de consultas; consultas podem ser movidas da lista corrente para o prontuário.
`RN08`	Exames são arquivos anexados a consultas; o *download* sempre substitui o destino se existir e tratar exceções de E/S.
`RN09`	Apenas usuários autenticados acessam as telas internas; logout/fechamento retorna a telas públicas.
`RN10`	Busca de usuários por nome de usuário do paciente e nome (médico) é suportada pelo modelo para ligações de agendamento.

`RN11`	O modelo mantém observadores; alterações no estado devem notificar as views para atualização.
`RN12`	Validação defensiva em *setters* impede estados inválidos (ex.: senha/especialidade vazias, médico nulo na consulta).
`RN13`	A lista de pacientes do médico é incremental por evento de agendamento; (não há deduplicação no MVP).
`RN14`	Formatos e máscaras de data seguem utilitário de data da aplicação; erros geram mensagens amigáveis.
`RN15`	Persistência é *in-memory* (dados não sobrevivem ao fechamento) no MVP.
`RN16`	Um médico não pode visualizar consultas que o paciente realizou com outros médicos.
`RN17`	O sistema não deve permitir agendamentos em horários já ocupados.

2.3 Técnicas Escolhidas/Ciclo de Vida/Divisão da Equipe

Metodologia proposta: scrum enxuto (iterações curtas de 1–2 semanas), pois o sistema tem muitos pontos que se beneficiam de feedback rápido (UI e fluxo clínico).

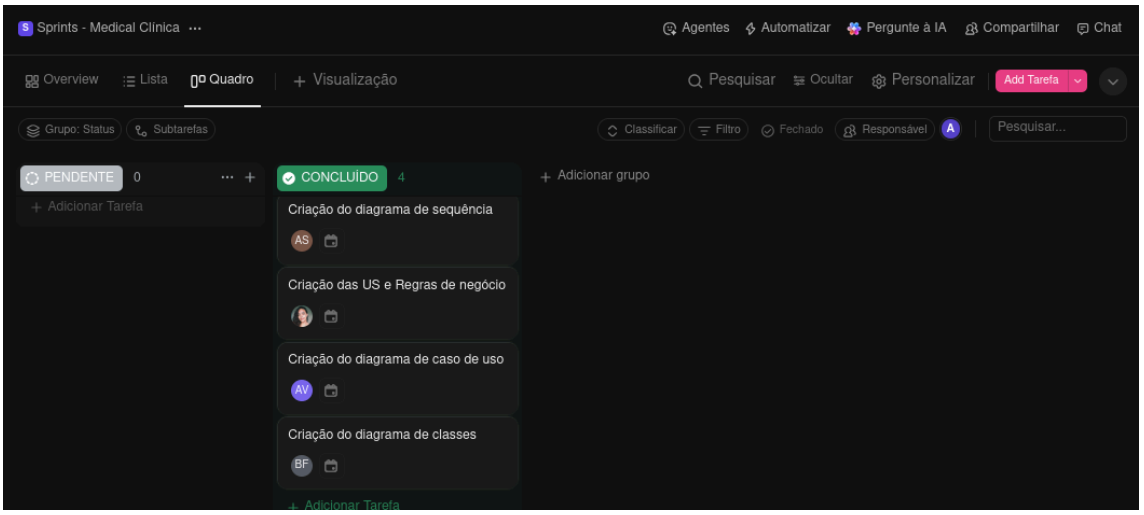


Figura 1. Organização do quadro de atividades no *ClickUp*

- **Artefatos e cerimônias essenciais**
  - **Product Backlog** (US e RF priorizados por valor clínico)
  - **Sprint Planning** (quebrar US em tarefas técnicas)
  - **Daily curta** (15 minutos)
  - **Review** (demonstração para médico/paciente “representantes”)
  - **Retrospectiva** (ajustes de processo)

    - ***Definition of Done***: história implementada, testada (unitária/integração básica), revisada, demonstrável na UI

- **Papéis sugeridos (e responsabilidades)**:
    - ***Product Owner* (PO)**: prioriza backlog, conversa com stakeholders (clínica), define critérios de aceitação clínica
    - ***Scrum Master***: facilita cerimônias, remove impedimentos, zela pelo fluxo
    - **Dev Front (Desktop/Swing)**: telas, controllers, usabilidade
    - **Dev Back/Modelo**: `SystemModel`, entidades, regras e validações, futura persistência
    - **QA/Analista de Testes**: casos de teste (funcionais e de regressão), teste exploratório de UI
    - **UX** (acumulável): fluxos, rótulos, consistência visual

- **Aplicação das Técnicas**
    - **Entrevistas — artefatos gerados**
        - **Roteiro** (amostra):
            - "Como você agenda uma consulta hoje? Quais passos e dificuldades?"
            - "Em uma consulta típica, o que você registra e em que ordem?"
            - "Como você compartilha/recebe exames hoje?"
            - "O que te faria confiar e usar o sistema diariamente?"
        - **Notas de sessão** com necessidades, termos usados e prioridades
        - **Mapas de jornada** simples (Paciente: login → agendar → comparecer → ver resultados; Médico: login → ver pacientes → registrar consulta → anexar exames)
    - **Prototipação — artefatos gerados**
        - **Wireframes das telas**: *Login*/Registro, *home* paciente (cartões de consultas, botão "Agendar"), formulário de agendamento (médico + data), *home* médico (lista de pacientes), detalhe do Paciente (timeline de consultas, ação "Adicionar consulta/Exame"), modal de anexar exame (selecionar arquivo), Tela de Consulta (descrição/status)
        - **Protótipo clicável** com fluxo "feliz" e cenários de erro básicos (data inválida, arquivo ausente)

- **Resultados Obtidos**
    - Agrupar necessidades por jornada (agendar, atender, registrar, anexar, consultar histórico)

- Traduzir necessidades em *User Stories* com critérios de aceite mensuráveis
- Definir MVP com foco em fluxo mínimo completo: login → agendar → registrar consulta → anexar exame → consultar histórico
- Identificar lacunas técnicas (persistência, segurança, auditoria, multiusuário real) para *backlog* futuro

2.4 Requisitos Funcionais

[RF01] – Autenticação de usuário
Descrição: o sistema deve permitir que o usuário (paciente ou médico) se autentique informando para o login: nome de usuário e senha válidos, direcionando-o automaticamente à interface correspondente ao seu perfil. **Atores**: médico e paciente **Prioridade:** ☑ Essencial ☐ Importante ☐ Desejável

[RF02] – Agendamento de consultas
Descrição: o sistema deve permitir que o paciente agende uma consulta selecionando primeiro a especialidade desejada e, em seguida, o nome do médico disponível nessa especialidade. Tudo isso respeitando a `RN17`. **Atores**: Paciente **Prioridade:** ☑ Essencial ☐ Importante ☐ Desejável

[RF03] – Visualizar listagem de consultas
Descrição: o sistema deve permitir que o paciente visualize uma lista contendo o histórico de todas as suas consultas (realizadas, futuras e canceladas). **Atores**: paciente **Prioridade:** ☑ Essencial ☐ Importante ☐ Desejável

[RF04] – Visualizar detalhes da consulta
Descrição: o sistema deve permitir que o paciente, ao selecionar um item da listagem (`RF03`), visualize os dados detalhados daquela consulta específica. Os dados são:

- Data e horário
- Status (Agendada, Realizada, Cancelada)
- Especialidade
- Nome do Médico
- Local (Endereço ou Link da Teleconsulta)
- Tipo (Presencial/Online)
- Observações (se houver)

Atores: paciente
Prioridade: ☑ Essencial ☐ Importante ☐ Desejável

[RF05] – Visualizar agenda de pacientes

Descrição: o sistema deve permitir que o médico visualize uma lista dos pacientes com consultas agendadas (futuras) ou realizadas recentemente, vinculadas ao seu perfil.

Atores: médico
Prioridade: ☐ Essencial ☑ Importante ☐ Desejável

[RF06] – Visualizar detalhes do atendimento

Descrição: o sistema deve permitir que o médico, ao selecionar um paciente da lista, visualize os dados específicos daquela consulta, dados esses especificados em `RF04`.

Atores: médico
Prioridade: ☐ Essencial ☑ Importante ☐ Desejável

[RF07] – Visualizar histórico do paciente

Descrição: o sistema deve permitir que o médico acesse o histórico de consultas passadas daquele paciente. Sendo esse histórico apenas com o médico em questão, assim como especificado em `RN16`.

Atores: médico
Prioridade: ☐ Essencial ☑ Importante ☐ Desejável

[RF08] – Cancelar consulta
Descrição: o sistema deve permitir que o médico e usuário possa alterar o status da consulta (ex: de "Ativo" para "Inativo"). **Atores**: médico e paciente **Prioridade:** ☑ Essencial ☐ Importante ☐ Desejável

[RF09] – Anexar exames médicos
Descrição: o médico deve poder anexar arquivos de exames a uma consulta, vinculando-os ao prontuário do paciente. O sistema deve validar o arquivo antes de salvar. **Atores**: médico **Prioridade:** ☐ Essencial ☑ Importante ☐ Desejável

[RF10] – Download de exames
Descrição: o paciente deve poder realizar o download dos arquivos de exames anexados pelo médico em suas consultas. **Atores**: paciente **Prioridade:** ☐ Essencial ☑ Importante ☐ Desejável

[RF11] – Prontuário do paciente
Descrição: o sistema deve permitir que o médico acesse o prontuário eletrônico do paciente, visualizando o histórico consolidado de consultas e exames registrados nas funcionalidades do sistema, de forma organizada e centralizada. O prontuário deve reunir automaticamente todas as informações cadastradas nos demais requisitos relacionados, funcionando como um histórico clínico acessível ao profissional. **Atores**: médico e paciente **Prioridade:** ☐ Essencial ☑ Importante ☐ Desejável

[RF12] – Cadastro de usuário (Paciente/Médico)
Descrição: o sistema deve permitir o cadastro de novos usuários, diferenciando entre paciente e médico. Para médicos, é obrigatório informar a especialidade. **Atores**: médico e paciente **Prioridade:** ☑ Essencial ☐ Importante ☐ Desejável

2.5 Requisitos não Funcionais

[RNF01] – Usabilidade e clareza
Descrição: a interface do sistema deve apresentar rótulos descritivos e consistentes, com textos compreensíveis para usuários leigos. Sempre que o usuário realizar uma ação inválida (como informar dados fora do formato esperado ou tentar cancelar uma consulta fora do prazo), o sistema deve fornecer feedback visual imediato, contendo mensagens objetivas que indiquem claramente o erro e como corrigi-lo. Um usuário sem treinamento prévio deve conseguir executar as funções essenciais, autenticar-se, agendar uma consulta e visualizar suas consultas, em menos de três minutos, considerando fluxo normal sem erros. **Atores**: todos os usuários **Prioridade**: ☑ Essencial ☐ Importante ☐ Desejável

[RNF02] – Desempenho de operações de arquivo
Descrição: as operações de anexar e baixar exames devem ser realizadas em tempo perceptível ao usuário, preferencialmente abaixo de **2 segundos** para arquivos locais pequenos (definidos como arquivos de até **5MB**). **Atores**: todos os usuários **Prioridade**: ☐ Essencial ☑ Importante ☐ Desejável

[RNF03] – Segurança de senhas
Descrição: as credenciais de acesso devem ser armazenadas com segurança (*hash* e *salt*) quando o sistema passar a ter persistência. **Atores**: todos os usuários **Prioridade**: ☐ Essencial ☑ Importante ☐ Desejável

[RNF04] – Persistência dos dados
Descrição: o sistema deve garantir a persistência das informações (usuários, consultas, prontuários, exames) em banco de dados relacional em futuras versões. **Atores**: sistema **Prioridade**: ☐ Essencial ☑ Importante ☐ Desejável

[RNF05] – Tratamento de exceções e confiabilidade
Descrição: o sistema deve tratar erros de entrada/saída (E/S), especialmente em operações com arquivos, mantendo a aplicação estável e informando o usuário. **Atores**: sistema **Prioridade**: ☑ Essencial ☐ Importante ☐ Desejável

[RNF06] – Padrão de projeto (MVC + *Observer*)
Descrição: a aplicação deve seguir o padrão *Model-View-Controller* com implementação do padrão *Observer* para atualização automática das views. **Atores**: desenvolvedores/sistema **Prioridade**: ☑ Essencial ☐ Importante ☐ Desejável

[RNF07] – Internacionalização e acessibilidade
Descrição: o sistema deve permitir futura tradução de textos e ajuste de contraste para acessibilidade. **Atores**: todos os usuários **Prioridade**: ☐ Essencial ☐ Importante ☑ Desejável

3. Documentação de casos de uso

#	Ator	Definição
1	Médico	Realiza consultas, edita prontuários, anexa exames.
2	Paciente	Agenda e visualiza consultas, faz login e cadastro.
3	Usuário	(Genérico) — faz cadastro inicial, podendo se tornar Médico ou Paciente.

Tabela 3. Atores do Sistema

O sistema possui um modelo de ator com três níveis:

1. **Usuário (Ator genérico)**: representa qualquer indivíduo antes de sua função específica ser determinada. É o ponto de entrada para novos registros.
2. **Paciente (Ator especializado)**: herda do Usuário. É um indivíduo que se registrou para gerenciar suas próprias informações de saúde e agendamentos.
3. **Médico (Ator especializado)**: herda do Usuário. É um profissional de saúde que usa o sistema para gestão clínica.

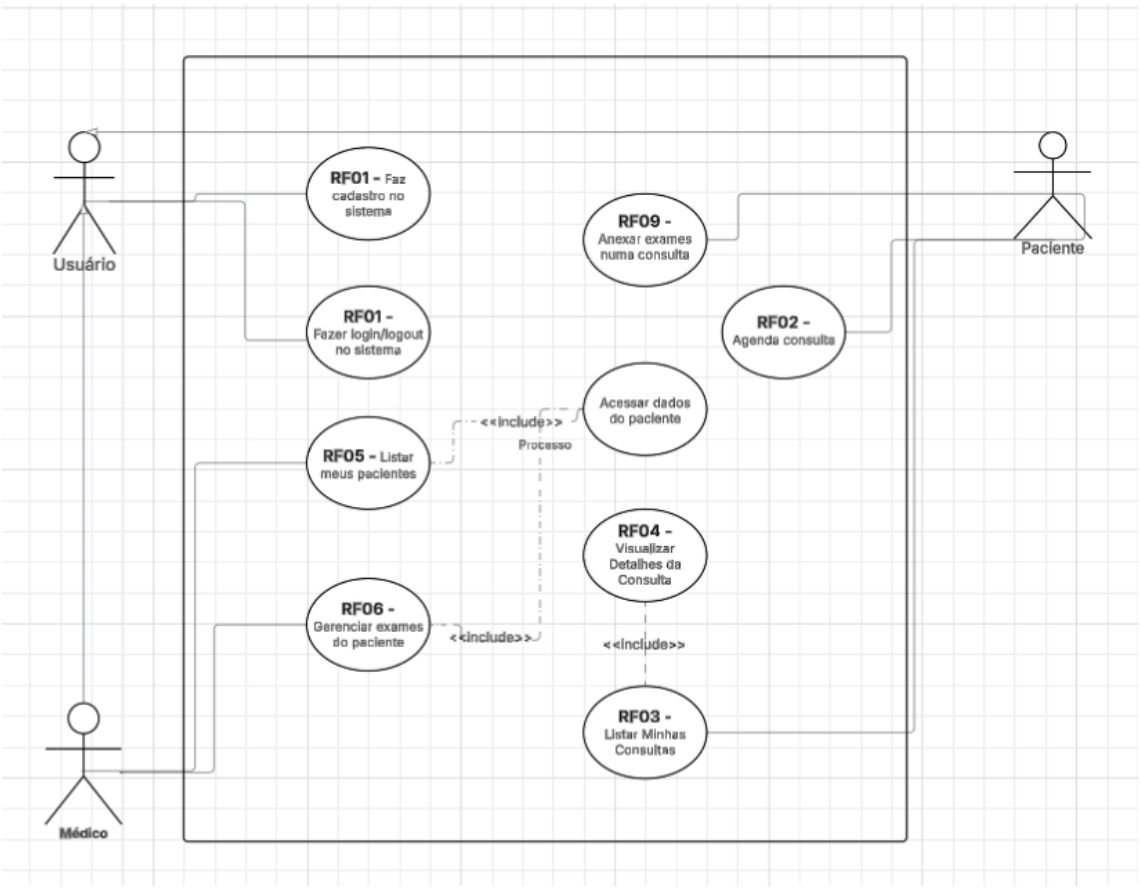
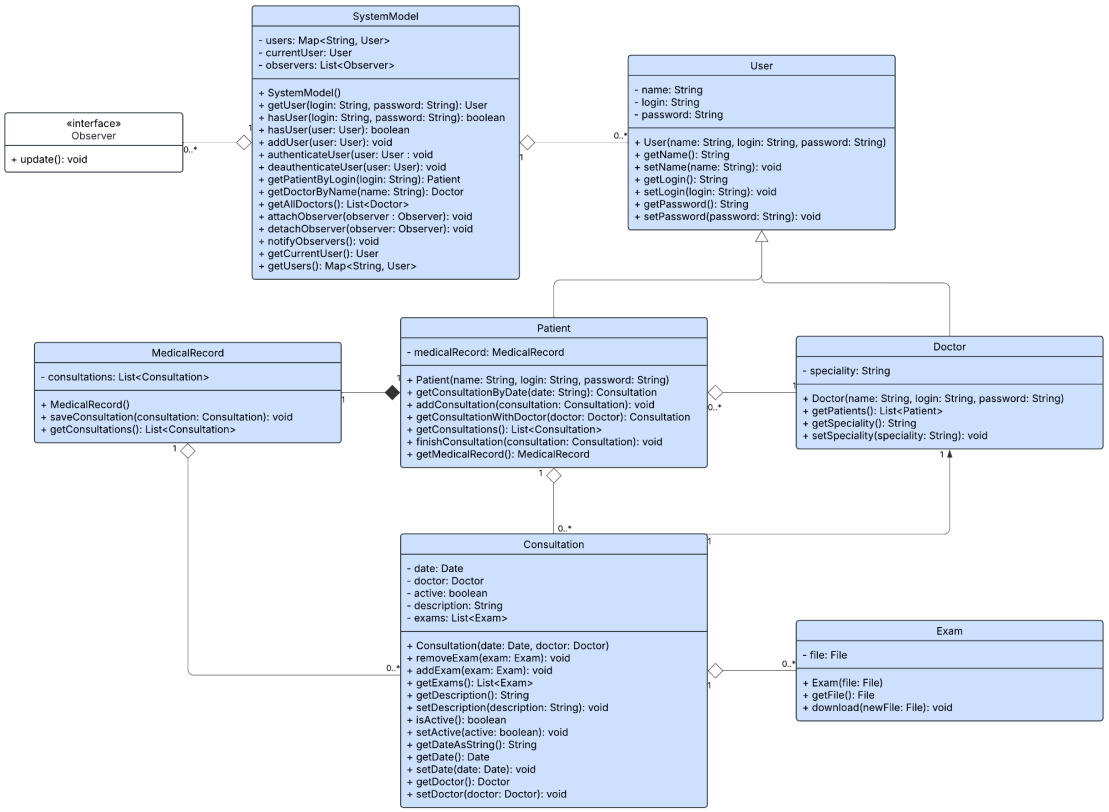


Figura 2. Diagrama de Caso de Uso

4. Diagrama de classe

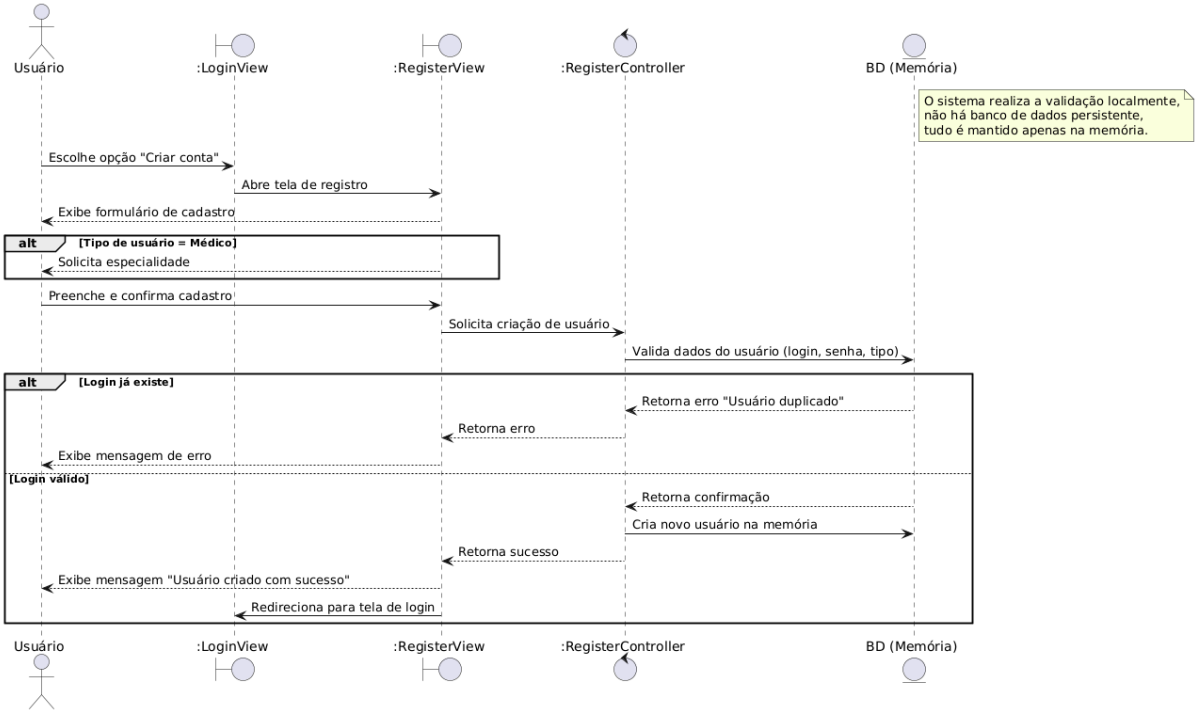
O seguinte diagrama de classe foi feito a partir do sistema escolhido:



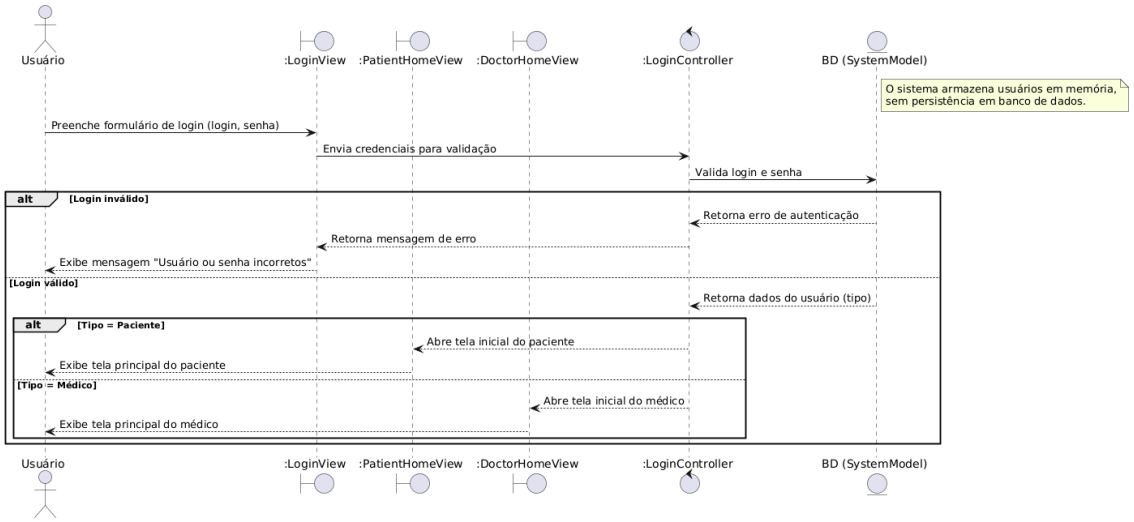
[Link para o diagrama de classes do sistema]

5. Diagrama de sequência

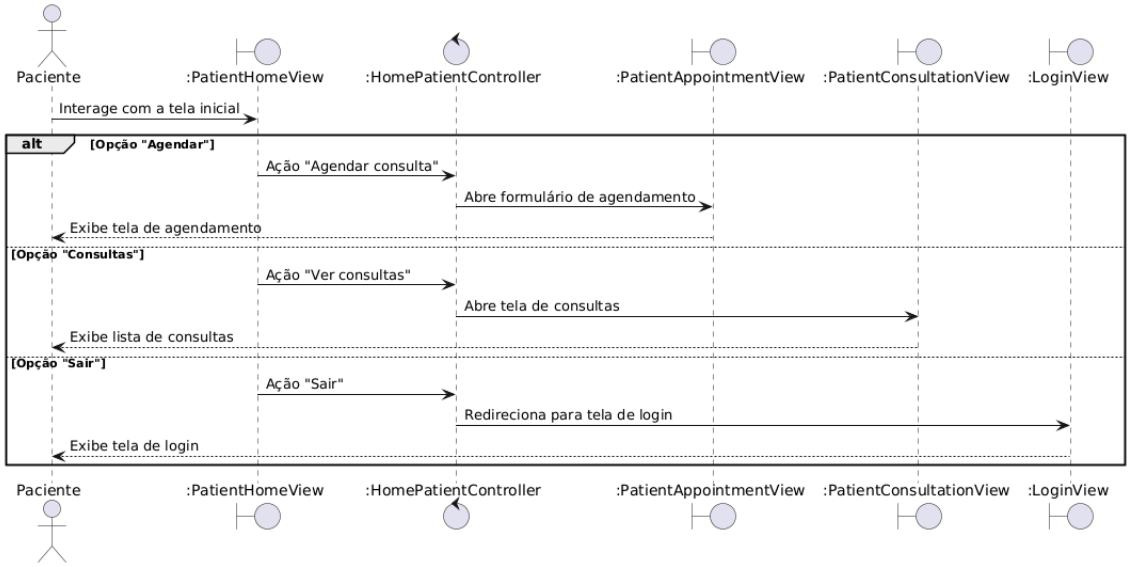
Os diagramas foram criados com *PlantUML*, uma abordagem UML que utiliza código para simplificar a elaboração dos diagramas. Cada imagem possui um *link* para seu endereço, permitindo uma visualização aprimorada no navegador:



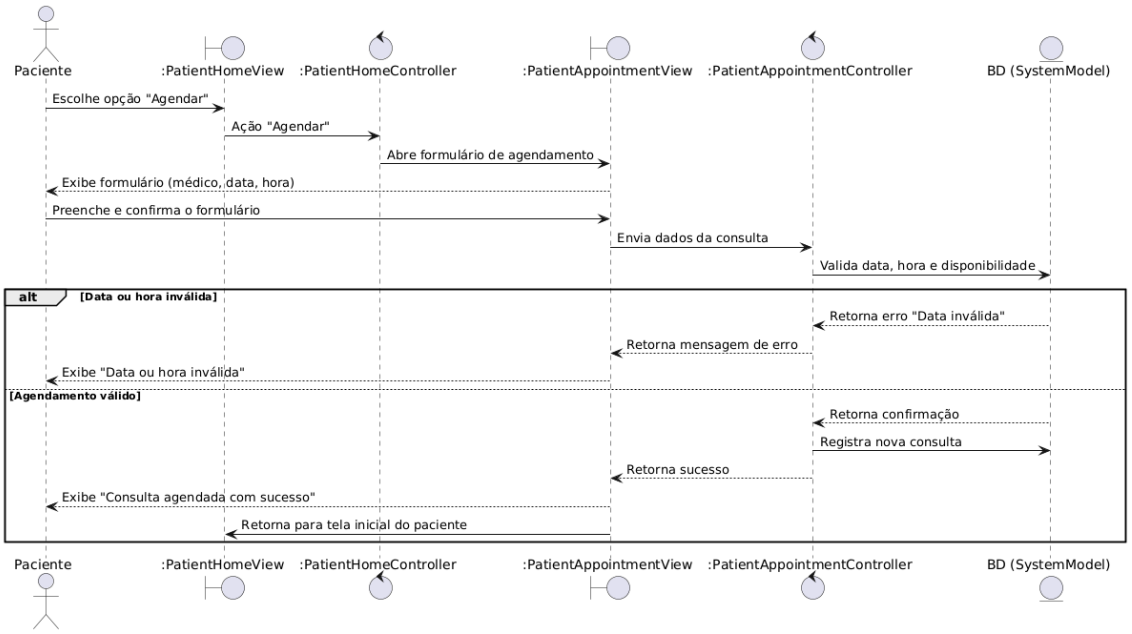
D. Sequência 1. Cadastro de Usuário



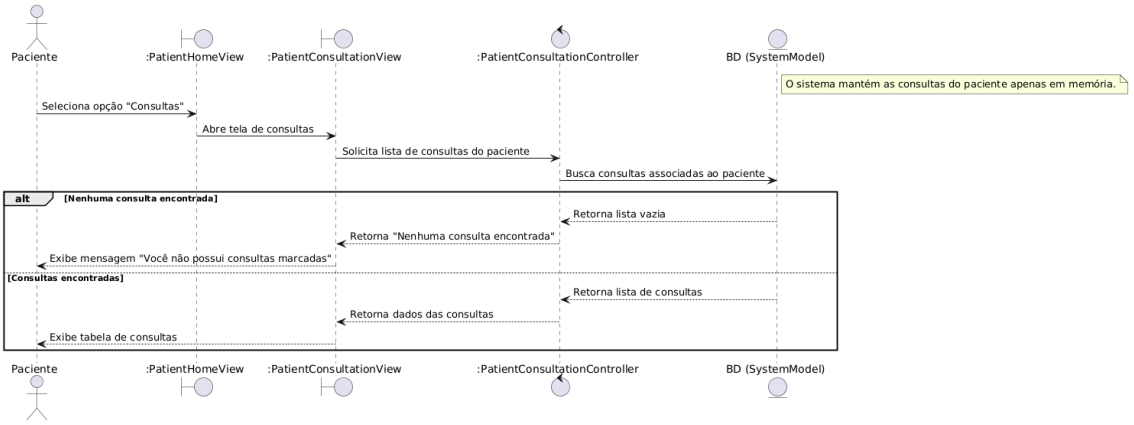
D. Sequência 2. Login de Usuário



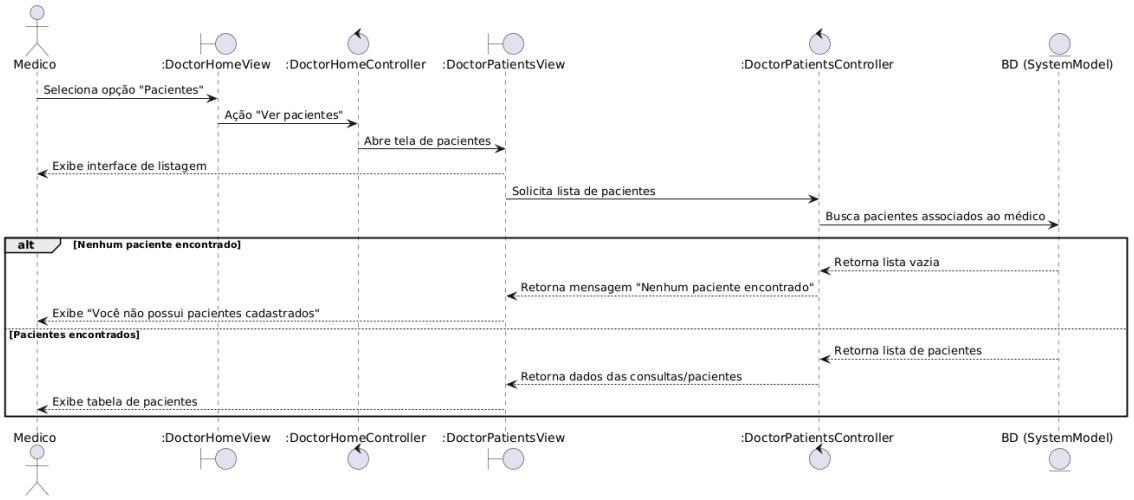
D. Sequência 3. Início Paciente



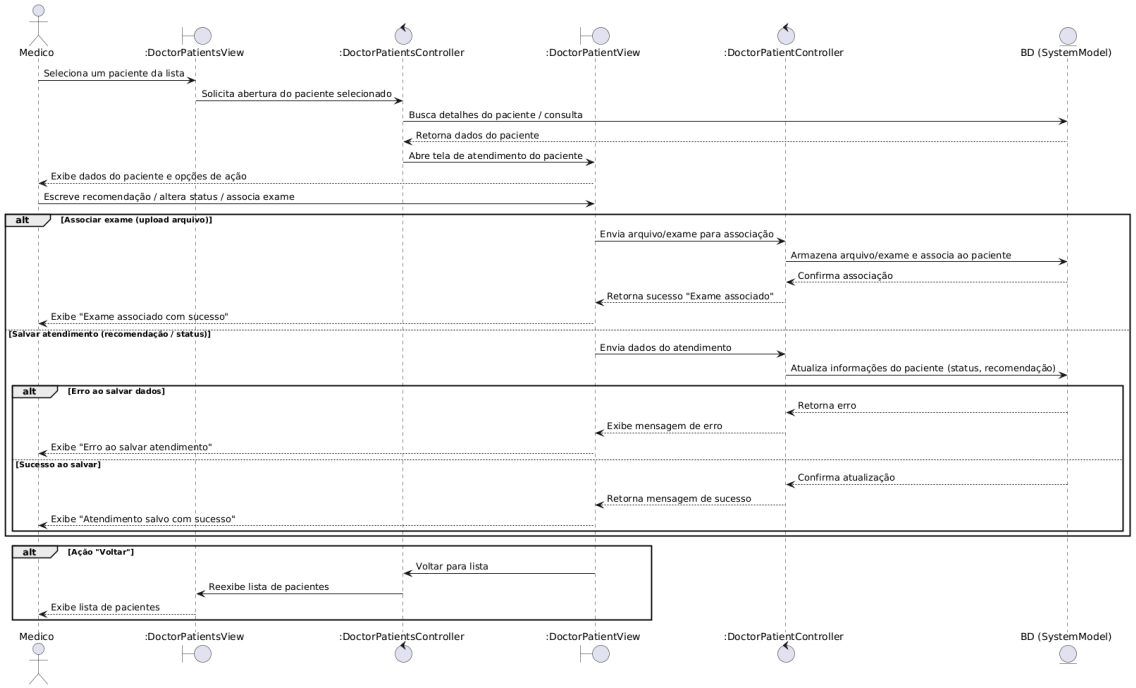
D. Sequência 4. Agendamento do Paciente



D. Sequência 5. Visualizar as consultas (Paciente)



D. Sequência 6. Visualização de Pacientes (Médico)

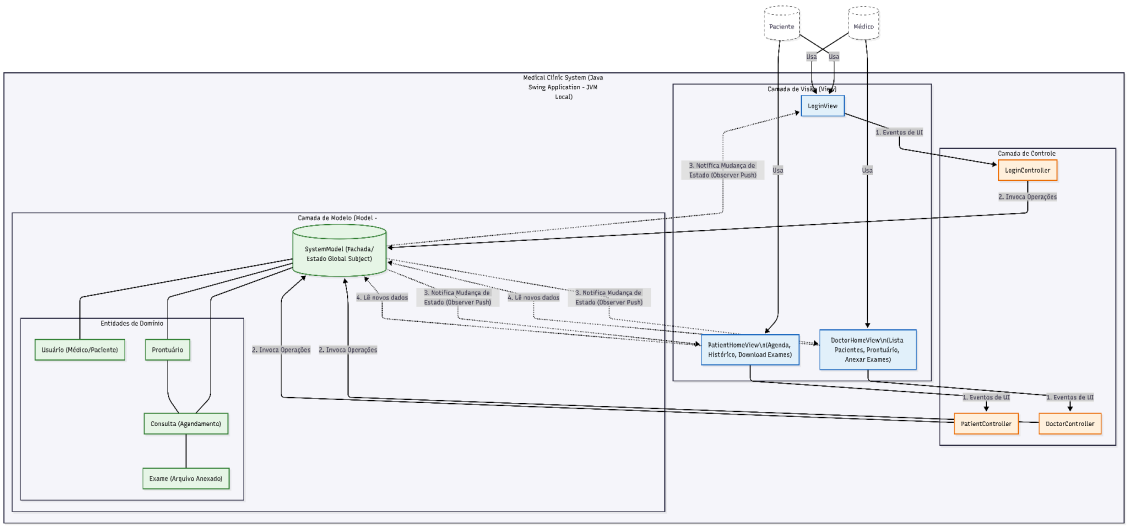


D. Sequência 7. Atendimento (Médico)

6. Arquitetura

6.1 Visão geral

O **Medical Clinic** é um aplicativo *desktop* desenvolvido em *Java* com *Swing*, estruturado segundo o padrão *Model–View–Controller (MVC)*, com um `SystemModel` central em memória responsável por manter usuários, consultas, exames e prontuários. As telas são organizadas em fluxos específicos para paciente e médico, contemplando login/registro, agenda, registro de consultas e anexação/download de exames. A comunicação entre o modelo e as camadas de interface utiliza o padrão *Observer*, garantindo atualização automática das *views* quando o estado interno do sistema é modificado. Na versão atual, a persistência é *in-memory* (os dados não sobrevivem ao fechamento da aplicação), havendo diretriz para futura migração para um banco de dados relacional, conforme os requisitos não funcionais de persistência e segurança.



6.2 Decisões arquiteturais

Para o desenvolvimento do *Medical Clinic*, foram tomadas três decisões arquiteturais fundamentais que definem a estrutura, o comportamento e a implantação do sistema. Abaixo, detalhamos cada decisão, sua justificativa e sua representação visual.

6.2.1 Decisão de arquitetura 1 – estilo monolítico (aplicação *desktop*)

Contexto: o projeto *Medical Clinic* tem como objetivo inicial entregar um Produto Mínimo Viável (MVP) para clínicas ambulatoriais de pequeno porte. O cenário de implantação prevê ambientes com infraestrutura de TI limitada, onde não há garantia de servidores dedicados ou conectividade de rede estável e contínua. A prioridade é validar os fluxos de negócio (agendamento e prontuário) com o menor custo de infraestrutura possível.

Alternativas avaliadas

1. **Arquitetura cliente-servidor (aplicação *web*)**: considerou-se criar um *front-end* (*React/Angular*) e um *back-end* (*API REST*). Foi descartada nesta etapa pois exigiria hospedagem, configuração de servidores e dependência de internet, o que foge do escopo de um MVP local;

2. **Arquitetura distribuída (microsserviços)**: considerou-se separar agendamento e prontuário em serviços distintos. Foi descartada por adicionar complexidade desnecessária (*overengineering*) para um domínio de baixa complexidade e baixo volume de dados;

Decisão tomada: optou-se pela adoção do estilo monolítico em aplicação *desktop*. Todo o sistema (interface gráfica *Swing*, lógica de negócio e armazenamento de dados em memória) reside em um único artefato executável (`.jar`) e roda em um único processo na máquina do usuário.

Consequências

- **Benefícios**
    - **Simplicidade de implantação (*Deployability*)**: o sistema não requer instalação de servidores de aplicação ou bancos de dados externos; basta executar o arquivo *Java*;
    - **Desempenho (*Performance*)**: como não há comunicação de rede (latência zero) e os dados estão em memória, a resposta da interface é imediata;
    - **Autonomia**: o sistema funciona perfeitamente sem conexão com a internet (*offline-first*).
- **Limitações**
    - **Escalabilidade**: não é possível distribuir a carga; o sistema está limitado ao hardware da máquina onde é executado;
    - **Acesso remoto**: o sistema só pode ser acessado localmente; médicos não podem ver a agenda de casa (nesta versão MVP).
    - **Acoplamento**: existe um risco maior de acoplamento entre classes se não houver disciplina na separação de pacotes (mitigado pela Decisão 2 - Camadas).

Impacto em atributos de qualidade

- **(+) Facilidade de construção (*Buildability*)**: alta. Permite foco total na regra de negócio;
- **(+) Desempenho**: alto. Elimina gargalos de I/O de rede;
- **(-) Disponibilidade**: limitada à disponibilidade da máquina física local.

Representação arquitetural

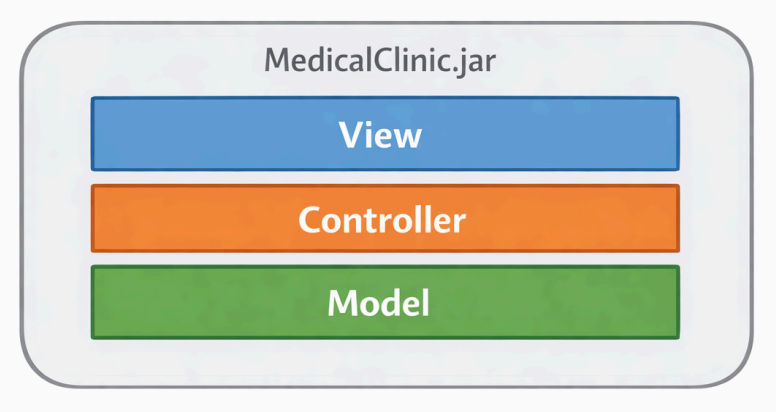


Figura representando visão monolítica em que a interface e modelo residem no mesmo artefato de execução.

6.2.2 Decisão de arquitetura 2 – Arquitetura em camadas (*Layered architecture*)

Contexto: o sistema *Medical Clinic* possui requisitos distintos de interface com o usuário (telas *Swing*), lógica de orquestração (fluxo de navegação) e regras de negócio (validação de agendamentos e prontuários). Era necessário adotar uma estrutura que impedisse a mistura dessas responsabilidades, evitando o anti-padrão "*Smart UI*" (a lógica fica dentro dos eventos de clique dos botões), o que dificultaria a manutenção e a evolução futura para um banco de dados persistente.

Alternativas avaliadas

1. **Tudo na interface (*Smart UI*)**: implementar as regras de negócio diretamente nas classes `JFrame` e `JPanel`. Foi descartada pois tornaria o código difícil de testar e impossível de reutilizar;

2. **Camadas estritas (*Strict Layering*)**: uma arquitetura em que a camada superior só fala com a imediatamente inferior. Foi considerada, mas flexibilizada para permitir que a View receba notificações do *Model* (via *Observer*) para atualização dinâmica;

Decisão tomada: adotou-se a **Arquitetura em camadas**, dividindo o sistema logicamente em três grandes responsabilidades estruturais:

1. **Apresentação (*View*)**: responsável apenas pela exibição e captura de eventos;
2. **Controle/Aplicação (*Controller*):** intermediário que processa as entradas;
3. **Domínio/Dados (*SystemModel*):** núcleo onde residem as entidades e o estado do sistema;

Consequências

- **Benefícios**
  - **Manutenibilidade (*Modifiability*)**: alterações na interface gráfica não impactam as regras de negócio e vice-versa;
  - **Evolutividade**: facilita a substituição futura da persistência (hoje em memória) por um banco de dados, pois o restante do sistema desconhece como os dados são guardados;
  - **Testabilidade**: permite testar as regras de negócio (*Model*) isoladamente, sem precisar instanciar janelas do *Swing*;
- **Limitações**
  - **Verbosidade**: aumenta o número de classes e arquivos, pois cada operação exige trânsito por todas as camadas (*Pass-through*);
  - **Curva de aprendizado**: exige disciplina da equipe para não "furar" as camadas (ex: a *View* acessar dados diretamente sem passar pelo *Controller* quando deveria);

Impacto em atributos de qualidade

- **(+) Manutenibilidade**: alta. O código fica organizado e modular;
- **(+) Testabilidade**: alta. Facilita testes unitários no núcleo do sistema;
- **(-) Desempenho**: leve impacto negativo (imperceptível neste escopo) devido à indireção das chamadas entre camadas;

Representação arquitetural

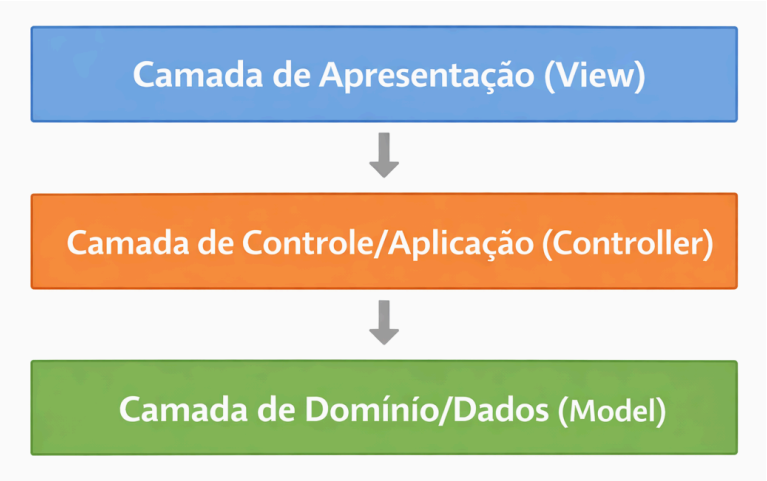


Figura representando arquitetura em camadas do sistema *Medical Clinic*.

6.2.3 Decisão de arquitetura 3 – Padrão MVC (*Model-View-Controller*)

Contexto: o *Medical Clinic* é uma aplicação rica em interfaces gráficas (*GUIs*) construída com *Java Swing*. Um desafio comum nesse tipo de aplicação é o gerenciamento de estados: quando um dado muda (ex: um novo agendamento é criado), várias partes da tela precisam ser atualizadas (a lista de consultas, o calendário, o contador de pacientes). Era necessário uma estratégia que permitisse essas atualizações sem que o código da interface ficasse "poluído" com lógica de sincronização de dados.

Alternativas avaliadas

1. ***Code-Behind* (Lógica na *View*)**: inserir o código de resposta (*listeners*) diretamente dentro das classes de interface. Foi descartada por criar alto acoplamento; mudar um botão exigiria mexer na lógica de negócio;

2. ***Model-View-Presenter* (MVP)**: um padrão em que o *Presenter* atualiza manualmente a *View*. Foi avaliado, mas descartado em favor do MVC com *Observer*, pois queria-se aproveitar o mecanismo de eventos para atualizações automáticas, reduzindo a necessidade de código "*boilerplate*" de atualização de tela;

Decisão tomada: adotou-se o Padrão Arquitetural MVC (*Model-View-Controller*) para a camada de apresentação:

- ***View (Swing)***: captura a interação do usuário e exibe dados. É "burra" (não toma decisões);

- ***Controller***: recebe os eventos da *View* (cliques), interpreta a intenção do usuário e invoca operações no *Model*;

- ***Model (SystemModel)***: contém os dados e a lógica. Ao ser alterado pelo *Controller*, ele notifica a *View* (via mecanismo *Observer*) que seu estado mudou, fechando o ciclo;

Consequências

- **Benefícios**
    - **Separação de interesses (SoC)**: *designers* podem trabalhar no *Swing* (*View*) enquanto desenvolvedores focam nas regras (*Controller/Model*);
    - **Múltiplas visualizações**: o mesmo objeto "Paciente" no *Model* pode ser exibido simultaneamente em uma "Tabela" e em um "Formulário de Detalhes" sem duplicar dados;
    - **Feedback imediato (Usabilidade)**: o uso do mecanismo *Observer* dentro do MVC garante que a interface reflita sempre o estado real dos dados;

- **Limitações**
    - **Complexidade**: para telas muito simples (ex: "Sobre o Sistema"), criar três arquivos (*View*, *Controller*, *Model*) pode ser excessivo (*overkill*);
    - **Indireção**: o fluxo de execução não é linear (*View* → *Controller* → *Model* → volta para *View* assincronamente), o que pode dificultar a depuração (*debugging*) para iniciantes;

Impacto em atributos de qualidade

- **(+) Usabilidade**: alta. Garante consistência visual dos dados apresentados;
- **(+) Manutenibilidade**: alta. Mudanças na aparência não quebram a lógica;
- **(-) Testabilidade da *View***: testar a interface gráfica automatizada é mais difícil que testar o *Model*, mas a separação ajuda a isolar os problemas;

Representação arquitetural

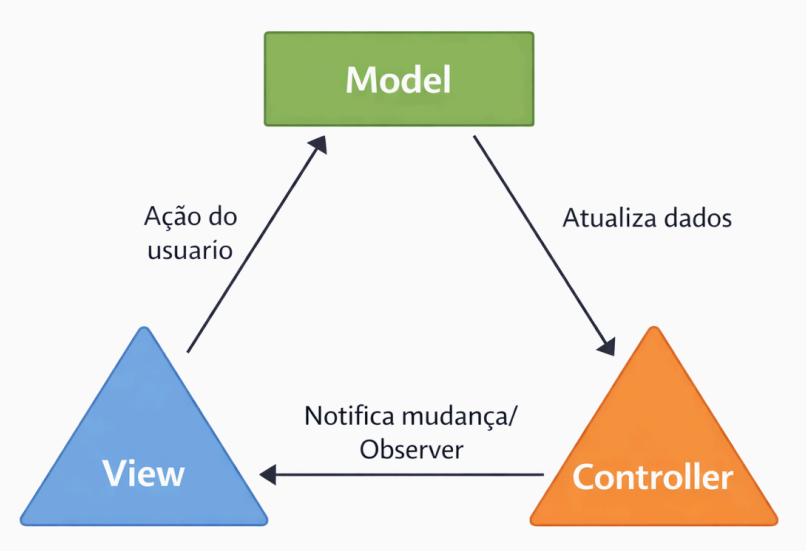


Figura representando o padrão Model-View-Controller (MVC) adotado na interface do sistema.

6.3 Estilos arquiteturais e atributos de qualidade

Esta seção apresenta a relação entre os estilos arquiteturais adotados no projeto *Medical Clinic* e os atributos de qualidade prioritários.

Para a avaliação, utiliza-se a seguinte legenda:

F (Favorece)	O estilo ajuda a atingir o atributo.
P (Prejudica)	O estilo dificulta ou impacta negativamente o atributo.
N (Neutro)	O estilo não possui impacto direto significativo sobre o atributo.

6.3.1 Estilos Analisados

Foram considerados para a matriz os seguintes estilos, alinhados com as decisões da *seção 6.2*:

1. **Aplicação Monolítica *Desktop***: estilo de implantação escolhido (Decisão 1);
2. **Arquitetura em camadas**: estilo estrutural escolhido (Decisão 2);
3. **MVC (*Model-View-Controller*)**: estilo de interação escolhido (Decisão 3);

6.3.2 Matriz de rastreabilidade (Estilos x Atributos)

Estilos arquiteturais	Disponibilidade	Usabilidade	Mutabilidade	Testabilidade	Segurança	Construtibilidade
MVC	**N**	**F**	**F**	**F**	**N**	**N**
Camadas *(Layered)*	**N**	**N**	**F**	**F**	**N**	**F**
Monolítico *Desktop*	**P**	**F**	**P**	**F**	**P**	**F**

6.3.3 Análise da matriz

MVC
Favorece a usabilidade (F): graças ao mecanismo *Observer* (intrínseco ao *MVC* neste projeto), a interface é atualizada automaticamente quando os dados mudam, garantindo *feedback* imediato ao usuário.
Favorece a mutabilidade (F): permite alterar completamente o visual (*View*) ou adicionar novas telas sem precisar reescrever a lógica de negócio (*Model*).
Favorece a testabilidade (F): isola a lógica de controle da visualização, facilitando a automação de testes.

Camadas *(Layered)*
Favorece a mutabilidade (F): a separação clara (*View*, *Controller*, *Model*) permite alterar uma camada (ex: mudar o banco de dados) sem quebrar as outras.
Favorece a testabilidade (F): permite testar as regras de negócio (*Model*) de forma isolada (testes unitários), sem depender da interface gráfica.
Favorece a construtibilidade (F): a organização padronizada dos pacotes facilita o entendimento da equipe e a divisão de tarefas.

Monolítico *desktop*
Prejudica a disponibilidade (P): o sistema reside apenas na máquina local. Se o computador falhar, o sistema fica indisponível (não há redundância em nuvem).
Favorece a usabilidade (F): a resposta da interface é imediata, pois não há latência de rede.
Prejudica a mutabilidade (P): tende a gerar alto acoplamento se não houver rigor, dificultando a separação futura para microsserviços ou web.
Prejudica a segurança (P): os dados ficam salvos localmente ou em memória, sem criptografia robusta de servidor ou controle de acesso centralizado.
Favorece a construtibilidade (F): a construção é simples (apenas um artefato `.jar`), ideal para o MVP.

6.4 Reutilização de software e linha de produtos

6.4.1 Lista de funcionalidades

Funcionalidades mandatória

- Autenticação de usuário
- Cadastro de usuário
- Agendamento de consultas
- Visualizar listagem de consultas
- Visualizar detalhes da consulta
- Cancelamento de consulta

Funcionalidades opcionais

- **Módulo médico**
  - Visualizar agenda de pacientes
  - Visualizar detalhes do atendimento

- Prontuário do paciente
- Visualizar histórico do paciente

- **Módulo de exames**
  - Anexar exames médicos
  - Download de exames

- **Módulo administrativo**
  - Funcionalidades de atendente/recepção
  - Geração de relatórios gerenciais

Funcionalidades alternativas

- **Modalidade de consulta**
  - Consulta presencial
  - Consulta on-line (telemedicina)

Funcionalidades OU

- Notificação via e-mail
- Notificação via WhatsApp

6.4.2 Mapa de Produto

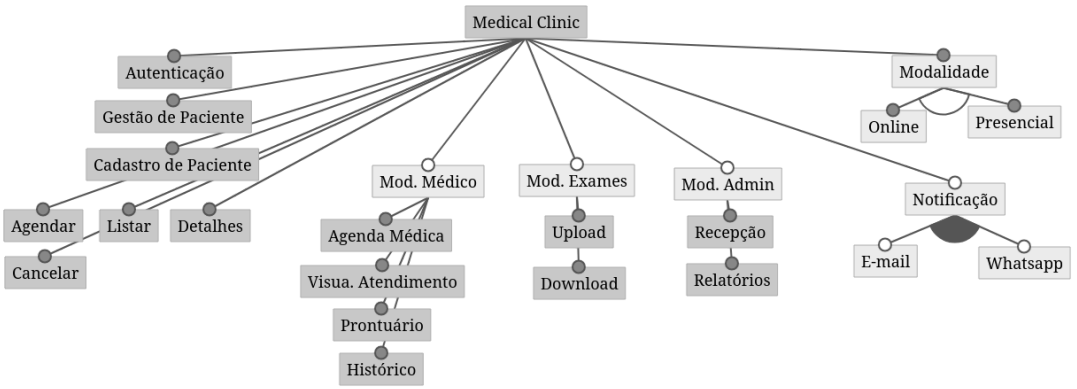
Veja a seguir uma captura do nosso mapa de produto, que também pode ser acessado clicando aqui:

Funcionalidades

Categoria	Funcionalidade (Feature)	Modelo 1 (App Pacier	Modelo 2 (Telemedicina l	Modelo 3 (Clínica Local	Modelo 4 (Enterprise Comp
MANDATÓRIAS	Autenticação de Usuário	X	X	X	X
	Cadastro de Usuário (Paciente)	X	X	X	X
	Agendamento de Consultas	X	X	X	X
	Visualizar Listagem de Consultas	X	X	X	X
	Visualizar Detalhes da Consulta	X	X	X	X
	Cancelamento de Consulta	X	X	X	X
VARIÁVEIS (MÉDICO)	Visualizar Agenda de Pacientes		X	X	X
	Visualizar Detalhes do Atendimer		X	X	X
	Prontuário do Paciente		X	X	X
	Visualizar Histórico do Paciente		X	X	X
VARIÁVEIS (EXAMES)	Anexar Exames Médicos			X	X
	Download de Exames			X	X
VARIÁVEIS (ADMIN)	Funcionalidades de Atendente/Rec				X
	Geração de Relatórios Gerenciais				X
ALTERNATIVAS (MODALIDADE)	Consulta Online (Telemedicina)		X		X
	Consulta Presencial			X	X
PERFIL DE USUÁRIO	Paciente	X	X	X	X
	Médico		X	X	X
	Atendente				X

6.4.3 Modelo F.O.D.A

Veja a seguir uma captura do nosso modelo F.OD.A



Modelo F.O.D.A de Medical Clinic

7. Conclusões

Percepção de Isis Nascimento de Lavor: A experiência de reestruturar e aprimorar um sistema desenvolvido há cerca de dois anos tem sido bastante enriquecedora. O processo de revisitar o projeto original, analisando suas funcionalidades, pontos fortes e limitações, proporcionou uma compreensão mais profunda sobre sua estrutura e funcionamento. Essa reflexão permitiu não apenas identificar oportunidades de melhoria, mas também consolidar aprendizados técnicos e conceituais adquiridos ao longo do tempo, percebendo possibilidades de reutilização e de adaptação dessa primeira versão para algo mais evoluído. Como responsável (junto com o Marcos) principalmente pela documentação inicial do sistema e também as etapas de regras de negócio e requisitos consegui aprender a documentar melhor, pois havia atuado como desenvolvedora nesse sistema inicialmente, no momento em que foi criado, e ele não foi tão bem documentado na época, assim percebo a importância e coloco em prática o que aprendi ao longos dos anos participando de algumas turmas mais voltadas a engenharia de software.

Percepção do Marcos Grégory Rodrigues Marques: o desenvolvimento do projeto *Medical Clinic* foi uma experiência enriquecedora que permitiu aplicar, de forma prática, conceitos de Engenharia de Software, como modelagem orientada a objetos, reutilização de software e arquitetura MVC. A participação no projeto reforçou a importância da documentação de requisitos, dos diagramas UML e da clareza nas regras de negócio, além de destacar a relevância da usabilidade em sistemas voltados à área da saúde. O trabalho em equipe contribuiu para o aprimoramento das habilidades técnicas e de comunicação, enquanto o sistema mostrou grande potencial de evolução com futuras implementações de persistência e banco de dados. De modo geral, o projeto foi relevante e aplicável a contextos reais, representando um avanço significativo no aprendizado e na prática de desenvolvimento de software.

Percepção de Álvaro Santos: O projeto em si é interessante e utilizável, um sistema de clínica não complexo, com um ciclo de ações fáceis de entender. Talvez precise de algumas melhorias na usabilidade e na interface para aperfeiçoar, mas não é complexo que deixe um usuário perdido. Por conta disso, o diagrama de sequência, neste estágio de funções, não é complexo de elaborar.

Percepção de Aline Vitória: Fazer documentação de software sempre é uma atividade muito interessante para mim, então eu sempre acho uma boa experiência e tento ao máximo aprender coisas novas com esse processo e já perceber coisas que podem ser implementadas e melhoradas no software.

Percepção de Beatriz Fernandes: Participar desse projeto foi bem interessante, principalmente pela oportunidade de entender melhor como a arquitetura MVC organiza o sistema de forma prática. Analisar os diagramas e a estrutura das classes me ajudou a perceber como cada componente se relaciona e como isso facilita tanto a manutenção quanto possíveis melhorias futuras. Foi legal ver na prática como um sistema voltado para a área da saúde pode ser estruturado de forma simples, mas funcional.

8. Ata de Reunião

N. da Reunião: 1ª. Reunião	
Data e horário	01/10/2025. Noite (21h)
Tipo	Virtual via Meet
Participantes:	Álvaro, Marcos Gregory, Isis Lavor, Aline Vitoria, Beatriz Fernandes
Atividade:	A equipe se reuniu pela primeira vez para iniciar a documentação e compreender o funcionamento do sistema escolhido para o trabalho.

N. da Reunião: 2ª. Reunião	
Data e horário	12/10/2025
Tipo	Virtual via WhatsApp
Participantes:	Álvaro, Marcos Gregory, Isis Lavor, Aline Vitoria, Beatriz Fernandes
Atividade:	Tirar últimas dúvidas e finalizar trabalho

9. Referências

PLANTUML. *PlantUML – Ferramenta open source para criação de diagramas UML.* Disponível em: https://plantuml.com/

PLANTTEXT. *PlantText – Editor online para PlantUML.* Disponível em: https://www.planttext.com/

Evelance. *Browser-based feature model editor*. May 28, 2020. Disponível em: https://evelance.de/online_feature_model_editor/index.htm. Acesso em: 26 nov. 2025. evelance.de