**INSTITUTO FEDERAL DO PARANÁ**

**CAMPUS TELÊMACO BORBA**

**DEMERSON SEMBRASKI ALVES**

**ERICSON SCHMIDT BICALHO**

**RAPHAEL DE FREITAS MACIEL**

**VICTOR LUIZ DA SILVA**

**Projeto do semestre**

**TELÊMACO BORBA**

**JUNHO, 2025**

**INSTITUTO FEDERAL DO PARANÁ**

**CAMPUS TELÊMACO BORBA**

**DEMERSON SEMBRASKI ALVES**

**ERICSON SCHMIDT BICALHO**

**RAPHAEL DE FREITAS MACIEL**

**VICTOR LUIZ DA SILVA**

**Projeto do semestre**

Atividade apresentada ao componente curricular Engenharia de Software, do curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal do Paraná (IFPR), Campus Telêmaco Borba.

Professor Celso Oliveira

**TELÊMACO BORBA**

**JUNHO, 2025**

**SUMÁRIO**

| [**1 - Objetivo 3**](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.ea7oilgzxzpp)  [1.1 - Requisitos Funcionais e Não-Funcionais 3](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.8qk2iiqbsc23)  [1.2 - Público-Alvo 3](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.xed6a61hkn7u)  [1.3 - Tecnologias Sugeridas: 3](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.cnz5cvo4lg66)  [1.4 - Entrega Mínima Viável (MVP): 3](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.lkqyi7spspql)  [**2 - Definição do escopo do projeto 3**](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.y8d85468ebo)  [2.1 - Escopo do MVP 4](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.wza9u1mb8fjo)  [2.2 - Diagrama de Escopo (Resumo) 6](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.4h8uw3iiz4a4)  [2.3 - Planejamento de cronograma 9](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.59w6ytc7i93e)  [2.4 - Planejamento de recursos 13](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.sx3dl45xgvk5)  [2.5 - Planejamento de riscos 14](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.95a63pp1mxqq)  [2.6 - Planejamento de testes 15](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.66iut1vrm71p)  [**3 - Modelagem do banco de dados: 20**](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.j1nhzsdgwpsc)  [3.1 - Modelagem de casos de uso: 2](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.djn7yt6rcoyf)1  [3.2 - Modelagem do sistema: 22](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.ajmzz83ttwa7)  [3.3 - Definição da arquitetura do sistema: 22](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.bkumhvdpwk4y)  [**4 - Configuração do Ambiente de Desenvolvimento 22**](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.qdw0gtcu6kq7)  [4.1 - Desenvolvimento das Funcionalidades 22](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.uc0vh36hz8gc)  [4.2 - Integração Contínua (CI) 24](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.p4xisyn03qzy)  [4.3 - Testes durante o desenvolvimento: 25](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.hg3x5juuly8s)  [4.4 - Revisão de código e controle de qualidade: 25](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.sw9k0yv8tjrz)  [**5 - Preparação para implantação:**](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.rkeqckga9q7a) **31**  [5.1 - Documentação de entrega:](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.ub2x69lual8q) 33  [5.2 - Implementação no ambiente de produção: 3](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.6ifcc7qy8k9d)4  [5.3 - Ensinando o cliente a usar o sistema 3](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.5rwjqaze5bxw)4  [5.4 - Cuidando do sistema após a entrega](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.42s15gsncdcp) 34  [**Referências 3**](https://docs.google.com/document/d/1p741nNbHFK6sUisajjVNYjGSjfOvSmMonCWjnHl61Uo/edit?pli=1&tab=t.0#heading=h.1ktez3tqfvvl)**5** |
| --- |

================================================================

**Ótica WideVISION**, Projeto para uma Clínica de Optometria

================================================================

| **Equipe:**  Demerson Sembraski Alves  Ericson Schmidt Bicalho  Raphael de Freitas Maciel  Victor Luiz da Silva |
| --- |

### 

# 1 - Objetivo Facilitar o agendamento de consultas, gerenciar pacientes, prontuário eletrônico com histórico de exames, armazenar exames e melhorar a comunicação entre a clínica e os clientes.

## 1.1 - Requisitos Funcionais e Não-Funcionais

[Documentação sobre os Requisitos do Projeto da Clínica de Optometria](https://docs.google.com/document/d/1bwH497-yIMyHnOErpxiAX-F9fowy_FJMJwo23ucA-ao/edit?usp=sharing)

## 1.2 - Público-Alvo Pessoas que buscam aferir sua qualidade visual.

## 1.3 - Tecnologias Sugeridas:

* **Front-end (App):** React Native.
* **Back-end:** Django + DRF (API REST).
* **Banco de Dados:** PostgreSQL.
* **Infraestrutura:** Hospedagem em AWS/Google Cloud (ou VPS).

## 1.4 - Entrega Mínima Viável (MVP):

Painel admin Django + cadastro usuário + agendamento online.

# 2 - Definição do escopo do projeto

Nosso objetivo com o MVP é entregar um sistema funcional, enxuto e bem feito, que atenda às necessidades principais dos usuários sem complicar o desenvolvimento. Vamos focar no que é essencial para que usuários tenham uma experiência fluida e confiável, respeitando prazos e garantindo qualidade. Aqui está o que planejamos para o MVP:

## 2.1 - Escopo do MVP

**Funcionalidades Prioritárias**:

* **Cadastro de Usuários**:

**Pacientes**: Vamos permitir o cadastro com informações básicas (nome, contato, histórico ocular, alergias e cirurgias anteriores). Tudo organizado para facilitar o uso.

**Optometristas e Administradores**: Esses perfis serão gerenciados com cuidado, garantindo controle total sobre quem acessa o sistema.

**Autenticação**: Um login seguro com senha, usando JWT (JSON Web Token) para proteger os acessos.

* **Agendamento Online**:

**Para Pacientes**: Escolher data, horário e optometrista direto pelo app, com uma interface clara e intuitiva.

**Validação no Back-end**: O sistema vai checar automaticamente se não há conflitos de horário, evitando bagunça na agenda.

**Confirmação por E-mail**: Cada agendamento será confirmado com um e-mail direto ao paciente, reforçando a organização e evitando que o mesmo esqueça o seu horário.

* **Painel Administrativo (Django)**:

**Gerenciamento de Usuários**: Administradores poderão visualizar e gerenciar pacientes e optometristas com facilidade.

**Listagem de Agendamentos**: Uma tela prática para ver todos os agendamentos, com filtros por data ou optometrista.

**Relatórios Simples**: Exportação de relatórios básicos, como a ocupação da agenda, em PDF, para facilitar a gestão.

* **Exclusões do MVP**:

Para manter o foco e não sobrecarregar a equipe, decidimos deixar algumas funcionalidades para fases futuras:

* Simulador de lentes (ex.: identificação facial).
* Integração com pagamentos (como PIX ou cartão).
* Gestão de estoque e vendas.
* Lembretes automáticos por SMS ou e-mail (ex.: via Firebase).
* Armazenamento de imagens, como retinografias ou OCTs.
* **Limites do Escopo**:

Queremos que o sistema seja acessível e seguro, então definimos:

**Plataformas**: Um app mobile (React Native para iOS e Android) e um painel web (usando Django Admin).

**Banco de Dados**: PostgreSQL e para simplificar (sem armazenamento de imagens no MVP).

**Segurança**: Criptografia básica para dados sensíveis (nome, contato, histórico) e um termo de consentimento para cumprir as exigências iniciais da LGPD.

**Integrações**: Apenas Google Calendar para sincronizar agendamentos. Integrações mais complexas, como Firebase ou gateways de pagamento, ficam para depois.

* **Critérios de Sucesso**:

\*Cadastro e login funcionando perfeitamente no app.

\*Agendamentos criados e exibidos corretamente no app e no painel administrativo.

\*Relatórios básicos sendo gerados no painel admin.

\*Sistema em conformidade com a LGPD (com criptografia e termo de consentimento).

* **Restrições**:

**Prazo**: Seguiremos o cronograma definido (detalhado em outra seção).

**Equipe**: Estamos trabalhando com uma equipe enxuta (1-3 desenvolvedores, cobrindo back-end, front-end e DevOps).

**Infraestrutura**: O sistema será hospedado na AWS (Elastic Beanstalk + RDS) ou Google Cloud, garantindo escalabilidade e confiabilidade.

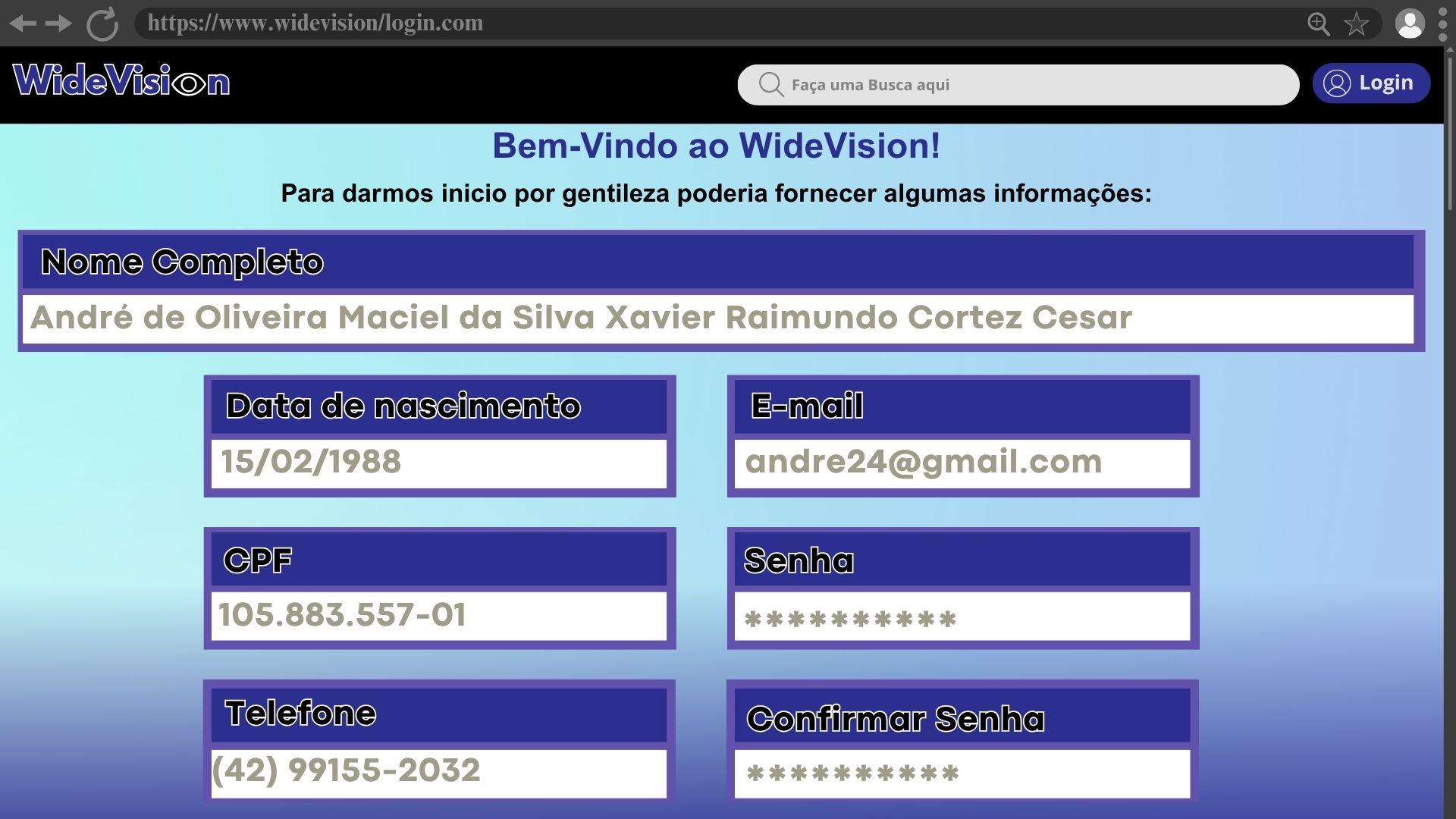
## 2.2 - Diagrama de Escopo (Resumo)

O que está incluído no MVP:

* **Modelos principais**:  
  + Usuário
  + Paciente
  + Agendamento
* **APIs disponíveis**:  
  + Cadastro de usuários (/api/usuarios/)
  + Agendamentos (/api/agendamentos/)
  + Listagem de pacientes (/api/pacientes/)
* **Telas do aplicativo**:  
  + Login



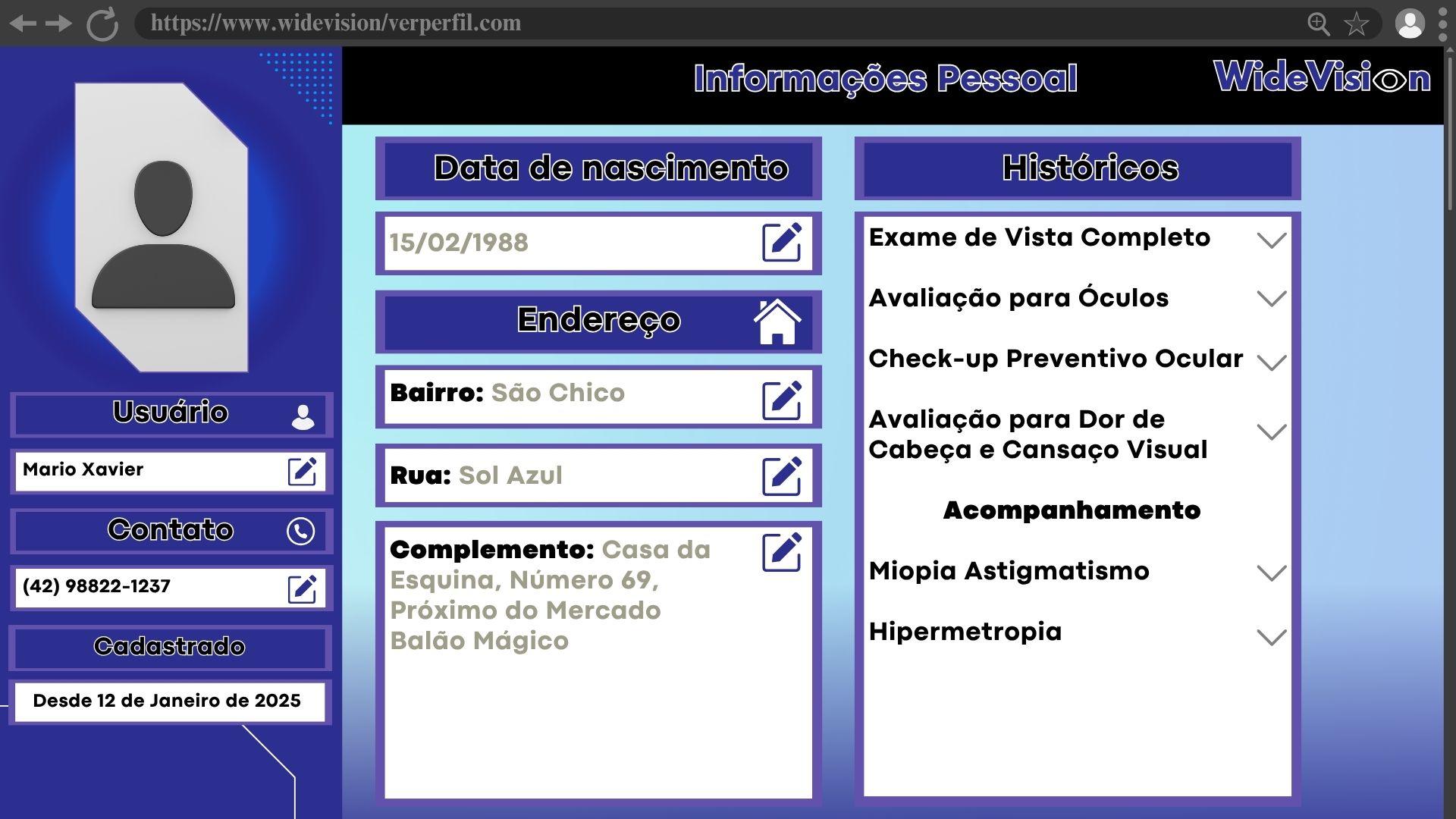
* + Cadastro



* + Agenda



* + Perfil do Usuário



* **Painel administrativo**:  
  + Gerenciamento de usuários
  + Controle de agendamentos

**Funcionalidades que ficam de fora neste momento (escopo futuro):**

* Recursos mais avançados como simulador de lentes, controle de estoque e pagamentos.
* Notificações push (ex.: via Firebase).
* Relatórios financeiros detalhados.

## 2.3 - Planejamento de cronograma

Nosso cronograma está pensado para uma equipe pequena (1-3 desenvolvedores) e um prazo realista de 8 semanas para entregar o MVP, com foco em desenvolvimento, testes e deploy. Ajuste conforme o tamanho e a experiência da equipe, e vamos reservar uma semana extra como margem para imprevistos.

## Semana 1: Pontapé inicial

Vamos começar com tudo! Esta semana é para planejar e preparar o terreno para o desenvolvimento.

O que faremos:

* Escolher a estrutura do sistema: usaremos Django com Django REST Framework para o back-end, React Native para o aplicativo e PostgreSQL para o banco de dados.
* Montar o ambiente de desenvolvimento com Docker e configurar a hospedagem inicial, provavelmente na AWS ou Google Cloud.
* Desenhar o diagrama do banco de dados (ER) usando o site do dbdiagram.io/home, para termos uma visão clara de como os dados vão se conectar.
* Escrever a documentação inicial dos endpoints da API com o Swagger, facilitando a vida de quem for usar ou integrar com ela.

Quem faz: Todo mundo da equipe coloca a mão na massa!  
O que entregaremos:

* Um diagrama ER pronto mostrando a estrutura do banco de dados.
* Repositório no GitHub configurado e pronto para o trabalho.
* Documentação inicial da API, organizada no Swagger.

## Semana 2-3: Construindo o coração do sistema

Hora de trabalhar no back-end, que é a base do nosso projeto.

O que faremos:

* Configurar o projeto Django com Django REST Framework (DRF).
* Criar os modelos principais: User, Paciente e Agendamento.
* Desenvolver os endpoints da API para cadastro, agendamento e listagem de dados.
* Configurar autenticação com JWT (JSON Web Token) para garantir segurança.
* Adicionar criptografia para proteger os dados, seguindo as regras da LGPD, usando o django-cryptography.

Quem faz: Desenvolvedor de back-end.  
O que entregaremos:

* Uma API funcionando com endpoints testados.
* Modelos e serializers prontos no Django.
* Testes unitários feitos com PYtest.

## Semana 4-5: Dando vida ao aplicativo

Agora é hora de construir a interface que os usuários vão ver e usar no aplicativo.

O que faremos:

* Configurar o projeto React Native (usando npx react-native init).
* Criar as telas principais: login, cadastro, agenda e perfil.
* Conectar o aplicativo à API usando a biblioteca axios.
* Configurar a navegação entre telas com react-navigation, para uma experiência fluida.

Quem faz: Desenvolvedor de front-end.  
O que entregaremos:

* Um aplicativo com telas básicas funcionando.
* Integração com a API concluída.
* Testes feitos em emuladores iOS e Android.

## Semana 6: Montando o painel de controle

Vamos criar um painel para facilitar a gestão do sistema, com uma interface prática para administradores.

O que faremos:

* Configurar o Django Admin ou criar um painel personalizado com django-adminlte.
* Mostrar listas de usuários e agendamentos no painel.
* Adicionar filtros para buscar agendamentos por data ou optometrista.
* Criar relatórios em PDF usando a biblioteca reportlab.

Quem faz: Desenvolvedor de back-end.  
O que entregaremos:

* Um painel administrativo funcional.
* Relatórios simples gerados em PDF.

## Semana 7: Conectando e testando

Nesta etapa, conectamos os sistemas e garantimos que tudo funciona corretamente.

O que faremos:

* Integrar a API do Google Calendar para sincronizar agendamentos.
* Testar a integração entre a API e o aplicativo.
* Fazer testes de segurança, verificando a LGPD e a autenticação JWT.
* Corrigir qualquer bug encontrado nos testes.

Quem faz: Todos os integrantes da equipe.  
O que entregaremos:

* Integração com Google Calendar funcionando.
* Relatório com os resultados dos testes de integração e segurança.

## Semana 8: Lançamento do projeto

Chegou a hora de colocar o sistema no ar e testar com usuários reais.

O que faremos:

* Fazer o deploy do back-end no AWS Elastic Beanstalk, com o banco de dados no RDS (ou equivalente no Google Cloud).
* Publicar o aplicativo em ambientes de teste, como TestFlight (iOS) e Google Play Beta (Android).
* Testar com um grupo pequeno de usuários reais e coletar feedback.

Quem faz: DevOps e a equipe completa.  
O que entregaremos:

* Aplicativo e API no ar, em ambiente de produção.
* Feedback inicial dos usuários coletado.

## Observações

* Ferramentas de gestão: Vamos usar Trello ou Jira para organizar as tarefas e GitHub para controle de versão e CI/CD.
* Riscos: Pode haver atrasos na integração com o Google Calendar ou problemas de compatibilidade no React Native entre iOS e Android. Por isso, é bom ter uma semana extra de margem.
* Testes: Vamos incluir testes unitários no back-end (com pytest), testes manuais no aplicativo (em emuladores) e testes de carga na API.
* Revisão: O cronograma será revisado toda semana para acompanhar o progresso e fazer ajustes com base no feedback da equipe.

## 2.4 - Planejamento de recursos

**Tecnologias definidas:**

* **Back-end:** Django + Django REST Framework

Utilizaremos o Django por sua robustez, funcionalidades prontas e, é ideal para MVPs e produtos que necessitam de agilidade para o lançamento.

Portanto utilizaremos também o Django REST Framework, facilitando a criação de APIs, estruturadas e com poucas linhas de código, segurança e customização personalizadas, contendo ótimo nível de escalabilidade e manutenibilidade.

* **Front-end:** React Native (iOS/Android)

A escolha do React, baseia-se na acessibilidade do código, sendo possível compartilhar entre plataformas, ampliando sua performance. Exemplo de empresas que utilizam React Native: Instagram, Tesla, Discord, Facebook etc.

**Equipe necessária:**

* Desenvolvedores (Full-stack, Front-end e Back-end)
* Designer de interface.

## 2.5 - Planejamento de riscos

**Riscos identificados:**

* Integração com sistemas externos (pagamento, notificações) pode gerar atrasos. Por exemplo: o sistema interno pode demorar ou falhar.

Para isso, podemos utilizar ferramentas como Celery (com o próprio Django) para processar posteriormente à realização do pagamento ou compra.

**Estratégias de mitigação:**

Adotar comunicação ativa com o cliente (reuniões regulares) um exemplo é utilizar um calendário de reuniões para definir próximos avanços do sistema, necessidades ou “dores” do cliente.

* Documentação clara das integrações e testes automatizados, atualizar com uma frequência definida pelos desenvolvedores e o cliente, realizar backups frequentemente, verificando os arquivos automaticamente a cada estágio da atualização.

## 

## 2.6 - Planejamento de testes

## Testes Funcionais

### Instalação e Configuração

* **Objetivo**: Validar instalação em diferentes dispositivos móveis
* **Cenários de Teste**:
  + Instalação em Android 8.0+ e iOS 13.0+
  + Primeira abertura do app (tela de boas-vindas)
  + Verificação de permissões (câmera para foto do receituário, notificações para lembretes)
  + Funcionamento offline básico (visualizar consultas já agendadas)
* **Dispositivos**: Simulação em emuladores Android Studio e iOS Simulator

### Sistema de Login

* **Objetivo**: Garantir acesso seguro dos pacientes
* **Cenários de Teste**:
  + **Primeiro acesso**: Cadastro com CPF, nome, email, telefone
  + **Login existente**: Email/CPF + senha
  + **Campos obrigatórios**: Validação de dados vazios
  + **Formato de dados**: CPF válido, email válido, telefone com DDD
  + **Recuperação de senha**: Envio por email
  + **Tentativas inválidas**: Bloqueio após 5 tentativas
* **Dados de Teste**: 10 usuários fictícios com diferentes cenários

### Funcionalidades de Agendamento

* **Objetivo**: Validar o fluxo completo de marcação de consultas
* **Cenários Principais**:
  + **Visualizar agenda**: Horários disponíveis por médico
  + **Marcar consulta**: Selecionar data, horário e tipo (consulta, retorno, exame)
  + **Cancelar consulta**: Até 24h antes do horário
  + **Reagendar consulta**: Alterar data/horário existente
  + **Histórico**: Visualizar consultas passadas
  + **Notificações**: Lembrete 1 dia antes da consulta

### Sistema de Pagamento

* **Objetivo**: Simular pagamento de consultas particulares
* **Métodos de Pagamento**:
  + **PIX**: Gerar QR Code para pagamento
  + **Cartão de débito/crédito**: Integração com gateway de pagamento
  + **Dinheiro**: Marcar como "pagar na clínica"
* **Cenários de Teste**:
  + Pagamento aprovado (consulta confirmada)
  + Pagamento recusado (consulta fica pendente)
  + Cancelamento durante pagamento
  + Comprovante de pagamento por email

## Testes de Performance

### Testes de Carga

* **Objetivo**: Avaliar sistema com múltiplos usuários simultâneos
* **Especificação**: 1000 usuários simultâneos fazendo agendamentos
* **Cenários de Carga**:
  + 100 usuários fazendo login simultaneamente
  + 500 usuários consultando agenda disponível
  + 300 usuários marcando consultas ao mesmo tempo
  + 100 usuários realizando pagamentos
* **Métricas**:
  + Tempo de resposta: < 3 segundos
  + Taxa de erro: < 1%
  + Disponibilidade: 99.5%

### Testes de Integração

* **APIs Externas**:
  + Gateway de pagamento
  + Serviço de notificações (Firebase)
  + Sistema da clínica (agenda dos médicos)

## Ambientes e Ferramentas

### Ambiente de Staging

* **Configuração**: Réplica do ambiente de produção
* **Dados**: Base simulada com:
  + 5 médicos oftalmologistas fictícios
  + 100 pacientes de teste
  + Agenda com 30 dias de antecedência
  + Histórico de 6 meses de consultas

### Pipeline CI/CD

* **Ferramentas**: GitHub Actions ou GitLab CI
* **Processo Automatizado**:
  1. **Build**: Compilação do app React Native
  2. **Testes Unitários**: Validação de funções básicas
  3. **Testes de Interface**: Cypress para fluxos principais
  4. **Deploy Staging**: Ambiente de teste
  5. **Testes Funcionais**: Selenium automatizado
  6. **Aprovação**: Deploy para produção

## Casos de Teste Prioritários

### Alta Prioridade

1. **Agendamento básico**: Paciente marca consulta de rotina
2. **Cancelamento**: Paciente cancela consulta com 48h de antecedência
3. **Pagamento PIX**: Fluxo completo de pagamento
4. **Login/Cadastro**: Primeiro acesso ao sistema

### Média Prioridade

1. **Reagendamento**: Alterar consulta existente
2. **Histórico médico**: Visualizar consultas anteriores
3. **Notificações**: Receber lembretes
4. **Validações**: Campos obrigatórios e formatos

### Baixa Prioridade

1. **Modo offline**: Funcionalidades sem internet
2. **Acessibilidade**: Suporte a leitores de tela
3. **Performance**: Otimização de carregamento

## Cronograma de Execução

**Semana 1**: Testes de instalação e login  
 **Semana 2**: Testes de agendamento e funcionalidades principais  
 **Semana 3**: Testes de pagamento e integração  
 **Semana 4**: Testes de carga e performance

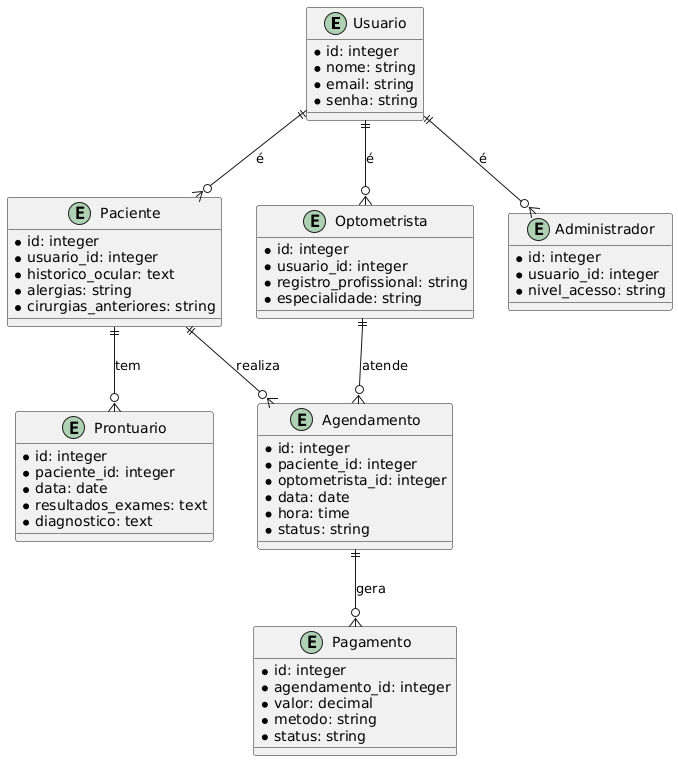
## Critérios de Aceitação

* **99% de sucesso** nos testes funcionais principais
* **Sistema suporta 1000 usuários** simultâneos sem falhas
* **Tempo de resposta < 3 segundos** para operações básicas
* **100% dos fluxos de pagamento** funcionando corretamente
* **Zero bugs críticos** que impeçam agendamentos

## Entregáveis

* Casos de teste documentados
* Relatório de execução
* Log de defeitos encontrados
* Métricas de performance
* Recomendações de melhorias

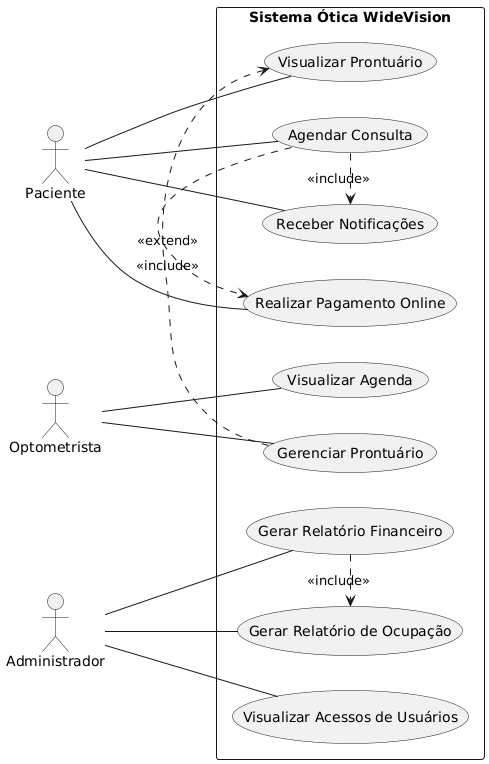
# 3 - Modelagem do banco de dados:

****

## 

## 3.1 - Modelagem de casos de uso:

## Diagrama de Casos de Uso – Visão Geral:



**CASOS DE USO**

| **1 - Agendar Consulta**  **Ator:** Paciente  **Descrição:** O paciente agenda uma consulta via app, escolhendo data, horário e optometrista.  **Passos:**   * Acessar a agenda disponível. * Selecionar horário e optometrista. * Confirmar agendamento e receber notificação.   **2 - Gerenciar Prontuário**  **Ator:** Optometrista  **Descrição:** O optometrista atualiza o prontuário do paciente após a consulta.  **Passos:**   * Acessar o perfil do paciente no painel Django. * Registrar resultados de exames e diagnósticos.   **3 - Gerar Relatório Financeiro**  **Ator:** Administrador **Descrição:** O administrador gera relatórios de pagamentos e consultas realizadas. **Passos:**   * Acessar o painel administrativo Django. * Filtrar dados por período ou optometrista * Exportar relatório em PDF/Excel.   **4 - Realizar Pagamento Online**  **Ator:** Paciente **Descrição:** O paciente paga a consulta via app após o agendamento.  **Passos:**   * Selecionar métodos de pagamento (cartão/PIX). * Confirmar pagamento. * Receber comprovante por e-mail.     **Histórias de Usuário**  **Para Pacientes:**   * Como paciente, quero agendar consultas online para evitar ligações ou deslocamento até a clínica. * Como paciente, quero receber lembretes de consultas para não esquecer meus compromissos. * Como paciente, quero acessar minhas receitas e exames no app para ter tudo organizado digitalmente. * Como paciente, quero pagar consultas via PIX para ter praticidade e segurança.   **Para Optometristas:**   * Como optometrista, quero visualizar a agenda do dia para me preparar para as consultas * Como optometrista, quero atualizar prontuários eletrônicos para manter o histórico do paciente atualizado.   **Para Administradores:**   * Como administrador, quero gerar relatórios de ocupação da agenda para otimizar o calendário de consultas e relatórios financeiros personalizados por data e optometrista. * Como administrador, quero saber quem acessou o app, para campanhas de marketing. |
| --- |

## 

## 3.2 - Modelagem do sistema:

Diagrama de classe:

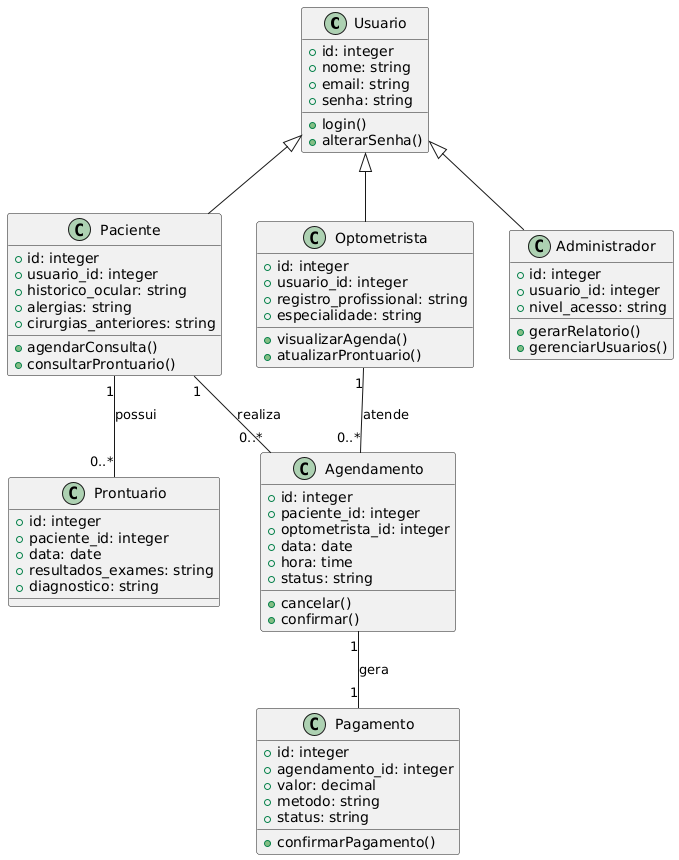


Diagrama de sequência - Agendamento Consulta

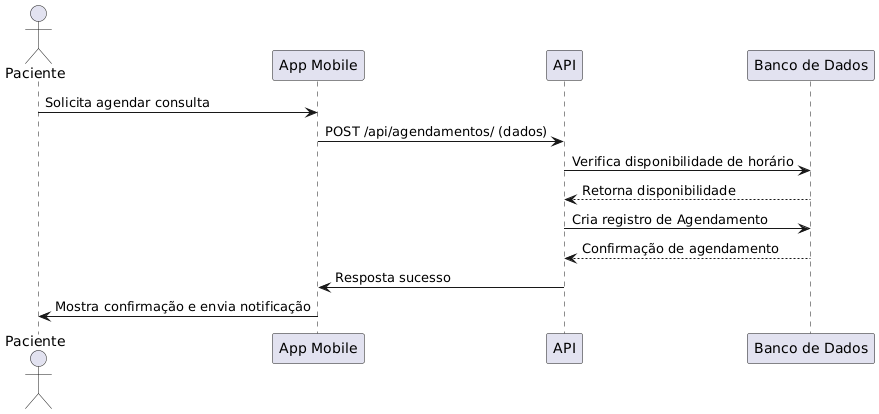
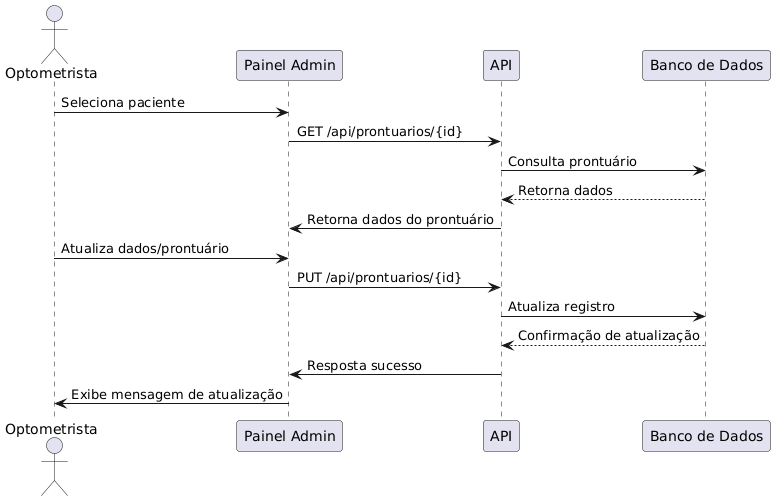
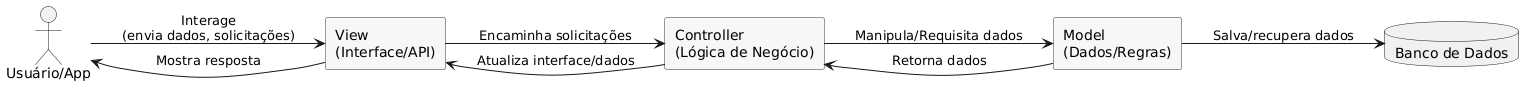


Diagrama de sequência - Atualização do Prontuário



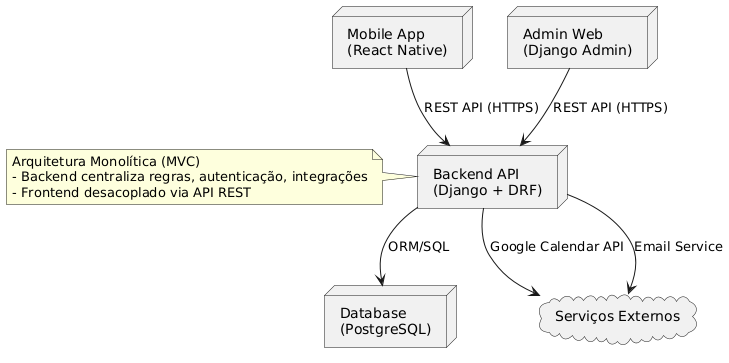
## 3.3 - Definição da arquitetura do sistema:

#### Arquitetura: Monolítica (Model-View-Controller)



* Backend Django (Model-View-Controller)
* Frontend Mobile desacoplado (React Native)
* Relacionamento via API RESTful

Diagrama da arquitetura:



# 4 - Configuração do Ambiente de Desenvolvimento

Nosso projeto consiste em uma base sólida e desenvolvida de ferramentas e organização para se utilizar essas ferramentas para implementar da melhor maneira o projeto.

## 4.1 - Desenvolvimento das Funcionalidades

#### Arquitetura do Projeto

* **Back-end**: Django + Django REST Framework (DRF)
* **Front-end**: React Native (iOS/Android)
* **Banco de Dados**: PostgreSQL
* **Ambiente de Contêineres**: Docker e Docker Compose
* **Controle de Versão**: Git (GitHub)
* **Hospedagem (Staging/Produção)**: AWS (Elastic Beanstalk + RDS) ou Google Cloud
* **Instalação de Dependências**
  + **djangorestframework, django-cors-headers, django-cryptography, djoser** (para autenticação), drf-yasg (documentação Swagger), **psycopg2-binary** (PostgreSQL), PyJWT.
* **Configuração do Docker e Docker Compose**
  + Criar **Dockerfile** para o back-end.
  + Configurar **docker-compose.yml** com containers para back-end, PostgreSQL e pgAdmin.
* **Criação de Modelos**
  + **User, Paciente, Agendamento,** com validações e criptografia em campos sensíveis conforme LGPD.

**Ambiente de Desenvolvimento Mobile**

* + Utilização de emuladores Android Studio e iOS (Xcode) para testes locais.
  + Comunicação com API via Axios, configurando variáveis de ambiente (.env) para URLs de back-end.

**Organização das Telas**

Para o código utilizaremos por exemplo:

* LoginScreen(ambiente de login)
* RegisterScreen(área de registro de usuários)
* ScheduleScreen(área de agendamento de consultas)
* ProfileScreen(área de perfil do usuário).

##### Banco de Dados

* PostgreSQL configurado com migrações gerenciadas via Django ORM.
* Ferramenta **dbdiagram.io** utilizada para gerar o modelo ER e manter documentação visual.

##### Ambiente Staging

* Criar ambiente intermediário para validações, usando banco de dados separado.
* Deploy inicial via Docker em instância EC2, Elastic Beanstalk ou GCP App Engine.

### 

## 4.2 - Integração Contínua (CI)

#### Objetivo da CI

Garantir que cada novo código enviado (commit/push) passe por testes automatizados e validações de qualidade antes de ser mesclado no código principal.

#### 

#### Ferramentas Sugeridas

* **GitHub Actions** ou **GitLab CI/CD**
* **Pytest**: Testes unitários no back-end
* **Jest/React Native Testing Library**: Testes no front-end
* **Prettier/ESLint**: Padronização de código React Native
* **Black/Flake8**: Padronização de código Python/Django

**Benefícios Esperados com CI:**

* Redução de erros na produção.
* Feedback rápido em cada atualização do sistema.
* Garantia de que a API e o app continuam compatíveis durante o desenvolvimento paralelo.

## 4.3 - Testes durante o desenvolvimento:

Os testes durante o desenvolvimento garantem que as funcionalidades do sistema da Clínica de Optometria sejam implementadas corretamente e que os módulos se comuniquem de forma eficiente. Esta seção cobre testes unitários, que validam o comportamento individual de funções e componentes, e testes de integração, que verificam a interação entre módulos (frontend, backend e APIs externas).

**Ferramentas**

**Frontend (React Native)**: Jest (framework de testes) com React Testing Library para testar componentes.

**Backend (Django)**: Pytest ou unittest (bibliotecas nativas do Python) para testar modelos, views e serializers.

**Cobertura**: Cobertura mínima de 90% para funções críticas (agendamento, login, pagamento).

**Escopo**

* Testar funções de validação (ex.: CPF, email).
* Testar componentes de UI (ex.: botão de agendamento, formulário de login).
* Testar lógica de negócios no backend (ex.: regras de agendamento, cálculo de valores de pagamento).

## 4.4 - Revisão de código e controle de qualidade:

**Ferramenta**

GitHub (Pull Requests) ou GitLab (Merge Requests).

**Fluxo**

O desenvolvedor cria um branch para a funcionalidade (ex.: feature/agendamento).

Após concluir o desenvolvimento e testes unitários, o desenvolvedor abre um Pull Request (PR) no GitHub.

O PR é atribuído a pelo menos um revisor (outro desenvolvedor da equipe).

**O revisor verifica:**

**Legibilidade:** Código claro, com nomes de variáveis e funções descritivas.

**Conformidade:** Adesão às convenções de codificação (ex.: PEP 8 para Python, ESLint para JavaScript).

**Funcionalidade:** Alinhamento com os requisitos da história de usuário.

**Segurança:** Identificação de vulnerabilidades (ex.: injeção de SQL, XSS).

**Testes:** Presença de testes com cobertura adequada.

* O revisor deixa comentários no PR, sugerindo melhorias ou aprovando.
* O desenvolvedor faz ajustes, se necessário, e o revisor aprova o PR.
* O PR é mesclado ao branch principal (main) após aprovação e execução bem-sucedida do pipeline CI/CD.
* **Frequência:** Todo código novo ou alterado deve passar por revisão antes da mesclagem.
* **Participantes:** Pelo menos dois desenvolvedores (autor e revisor). Para funcionalidades críticas (ex.: pagamento, login), incluir um desenvolvedor sênior.

**Diretrizes**

Manter revisões focadas (máximo 400 linhas por PR).

Realizar revisões dentro de 24 horas para evitar atrasos.

**Exemplo de Checklist de Revisão:**

**( )** Há testes unitários para as funções/componentes novos?

**( )** As entradas do usuário são validadas (exemplo: CPF, email)?

**( )** Não há vulnerabilidades de segurança (exemplo: senhas expostas, SQL Injection)?

**Métricas:** Tempo médio de revisão: < 24 horas.

# 5 - Preparação para implantação:

**Objetivo**: Confirmar que todas as funcionalidades especificadas estão implementadas, testadas e funcionando conforme os requisitos definidos nas histórias de usuário e casos de uso.

**Revisão dos Requisitos**

Mapear todas as funcionalidades descritas nas histórias de usuário (ex.: agendar consulta, pagar via PIX, gerar relatórios financeiros) e casos de uso (ex.: gerenciar prontuário, realizar login).

Criar uma checklist baseada nos requisitos funcionais e não funcionais (ex.: performance, acessibilidade, modo offline).

**Execução de Testes Automatizados**

Executar testes unitários (Jest para React Native, Pytest para Django) para validar funções e componentes individuais.

Executar testes de integração (Cypress para fluxos end-to-end, Pytest para APIs) para verificar a comunicação entre módulos (frontend, backend, APIs externas como gateway de pagamento e Firebase).

Confirmar cobertura de testes: ≥ 90% para funcionalidades críticas (agendamento, pagamento, login).

**Testes Manuais**

Realizar testes exploratórios em ambiente de staging para validar fluxos completos (ex.: paciente agenda consulta, paga via PIX e recebe comprovante).

Verificar cenários de erro (ex.: tentativa de agendar consulta em horário ocupado, pagamento recusado).

**Validação de Requisitos Não Funcionais**

**Performance**: Tempo de resposta < 3 segundos para operações básicas (ex.: carregar agenda, confirmar agendamento).

**Segurança**: Validar proteção contra injeção de SQL, XSS e armazenamento seguro de senhas (usando Bandit e OWASP ZAP).

**Acessibilidade**: Suporte a leitores de tela com conformidade mínima de 95% com WCAG 2.1 (testado com ferramentas como axe).

**Correção de Defeitos**

Registrar bugs encontrados em ferramentas como Jira ou Trello, com detalhes (passos para reprodução, gravidade, impacto).

Priorizar correções: bugs críticos (ex.: falha no agendamento) resolvidos antes da implantação.

**Documentação**

Atualizar documentação técnica (ex.: README, guias de API) e de usuário (ex.: manual do app).

Gerar relatório de verificação com resultados de testes e status das funcionalidades.

**Checklist de Verificação**

**( )** Login e cadastro funcionando (validação de CPF, email, recuperação de senha).

**( )** Agendamento de consultas (visualizar agenda, marcar, cancelar, reagendar).

**( )** Pagamento (PIX, cartão, dinheiro) com comprovante por email.

**Ferramentas**

**Acessibilidade**: axe, VoiceOver (iOS), TalkBack (Android).

**Gestão de Bugs**: Jira ou Trello.

**Pipeline CI/CD**: GitHub Actions ou GitLab CI para executar testes automaticamente.

**Exemplo de Caso de Teste Manual**

**Funcionalidade**: Agendar Consulta.

**Passos**

* Acessar seção de agendamento.
* Selecionar optometrista, data e horário.
* Confirmar agendamento.

**Resultado Esperado**: Consulta registrada no sistema, notificação enviada, agenda atualizada.

**Cenário de Erro**: Tentar agendar em horário ocupado → sistema exibe mensagem de erro e sugere outros horários.

**Critérios de Aceitação**

* Zero bugs críticos que impeçam o uso de funcionalidades principais.
* Todos os requisitos das histórias de usuário atendidos.
* Relatório de testes gerado com evidências (screenshots, logs).

## 5.1 - Documentação de entrega:

[Documentação de Entrega](https://docs.google.com/document/d/1F5Ta9zWvRkq2A8rUNSyhJoo0JRJO8DY0L5N5QDm40kU/edit?usp=sharing)

## 5.2 - Implementação no ambiente de produção:

Chegou o momento de lançar o sistema para o mundo e vamos garantir que tudo esteja funcionando corretamente no ambiente de produção.

**O que faremos:**

* Fazer o *deploy* do back-end no AWS Elastic Beanstalk, com o banco de dados configurado no RDS (ou equivalente no Google Cloud, se for o caso).
* Configurar os servidores e o ambiente de produção para rodar o aplicativo e a API de forma estável e segura.

## 5.3 - Ensinando o cliente a usar o sistema

Para garantir que os usuários tirem o máximo do nosso sistema, vamos capacitá-los com treinamentos práticos e materiais bem fáceis de entender.

**O que faremos:**

* Organizar sessões de treinamento (presenciais ou online) para mostrar como usar as principais funções do app, como agendamentos, cadastro e o painel administrativo.
* Criar materiais de apoio, como um tutorial passo a passo, um FAQ com as dúvidas mais comuns e, quem sabe, um vídeo rápido explicando o básico.

## 5.4 - Cuidando do sistema após a entrega

Nosso trabalho não acaba com o *deploy*! Vamos ficar de olho para resolver qualquer problema e manter o sistema rodando lisinho.

**O que faremos:**

* Oferecer suporte técnico imediato para corrigir bugs ou dúvidas que aparecerem logo após o lançamento.
* Criar um plano de manutenção, definindo como faremos atualizações, backups e monitoramento para garantir que o sistema continue funcionando bem.

# Referências

**ENCODE OSS LTD.** Instalação, tutoriais e comunidade Django Rest Framework. Disponível em:<https://www.django-rest-framework.org/>. Acesso em: 03 jun. 2025.

**META OPEN SOURCE.** Comunidade, blog e desenvolvimento React Native. Disponível em:<https://reactnative.dev/docs/getting-started>. Acesso em: 03 jun. 2025.

**AMAZON WEB SERVICES.** AWS Elastic Beanstalk: implantação de aplicações web. Disponível em:<https://aws.amazon.com/pt/elasticbeanstalk/>. Acesso em: 05 jun. 2025.

**DJANGO SOFTWARE FOUNDATION.** Django REST Framework: building APIs with Django. Disponível em:<https://www.django-rest-framework.org/>. Acesso em: 05 jun. 2025.