INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ ()
CAMPOS TERESINA CENTRAL ()
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
ENGENHARIA DE SOFTWARE III

VET MANAGER

GABRIEL SILVA, ALEXANDRO, DANIEL VITOR, LAYS EMANUELLY

TERESINA/PI
2025

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ ()
GABRIEL SILVA, ALEXANDRO, DANIEL VITOR, LAYS EMANUELLY

VET MANAGER

Projeto de relatório da disciplina de Engenharia de Software III que visa abordar todas as tecnologias e ferramentas utilizadas, assim como os métodos utilizados pela equipe para entrega dos prazos usando sprints na construção de um aplicativo para clínicas veterinárias.Engenharia de Software III do curso de Análise e Desenvolvimento de Sistemas da Instituto Federal de Educação, Ciência e Tecnologia do Piauí.

Ely Miranda

TERESINA/PI
2025

Lista de Figuras

Lista de Tabelas

Sumário

Páginas

1 Descrição do Projeto

1.1 Apresentação

O VetManager surge como uma solução inovadora, concebida para transformar a gestão da saúde animal. Este sistema integrado representa uma plataforma de conexão robusta e eficiente entre tutores e clínicas veterinárias. Ao facilitar o agendamento online de consultas e centralizar o histórico de saúde dos animais de estimação, o VetManager visa otimizar a experiência de todos os envolvidos.

Em resposta à crescente demanda por modernização no setor veterinário, o VetManager foi desenvolvido para simplificar e aprimorar as interações entre clínicas e seus clientes, elevando a qualidade do atendimento por meio da otimização de processos e comunicação mais fluida.

1.2 Público-alvo

O VetManager destina-se a atender às necessidades de diversos atores no universo da saúde animal:

- **Tutores de animais de estimação:** Indivíduos que valorizam a praticidade e organização no cuidado com seus animais, buscando ferramentas para facilitar esse acompanhamento.
- **Clínicas veterinárias:** Estabelecimentos que desejam modernizar sua gestão e fortalecer o relacionamento com os clientes, otimizando o fluxo de trabalho.
- **Profissionais veterinários:** Veterinários e suas equipes que necessitam de um sistema ágil para acessar rapidamente o histórico médico dos pacientes, garantindo um atendimento mais eficiente.

1.3 Justificativa e Impactos

O VetManager foi impulsionado pela constatação de desafios significativos no setor veterinário:

- **Ineficiência no Agendamento de Consultas:** Processos de agendamento burocráticos geram perda de tempo e frustração para clientes e clínicas.

- **Gestão Desorganizada do Histórico Médico:** Dificuldades no acesso aos prontuários podem resultar em erros de diagnóstico e tratamentos inadequados.
- **Falta de Informações Precisas e Acessíveis:** Informações desatualizadas sobre clínicas dificultam o acesso aos serviços veterinários pelos tutores.

O VetManager busca gerar impactos positivos no setor:

- **Maior Satisfação dos Clientes:** Experiência prática e transparente, aumentando a fidelização.
- **Otimização da Gestão Clínica:** Automatização de tarefas e melhor organização das informações.
- **Acompanhamento da Saúde Animal Mais Eficaz:** Histórico médico acessível e agendamento simplificado.
- **Redução de Erros e Custos Operacionais:** Menos erros manuais, otimização de tempo e recursos.
- **Comunicação Facilitada e Humanizada:** Contato direto entre tutores e veterinários, fortalecendo a confiança.

2 Responsabilidades

2.0.1 Desenvolvedor Back-end

Membros: Gabriel Silva e Francisco Alexandro

Responsabilidades Detalhadas:

- **Desenvolvimento e Manutenção da API REST:**
  - *Criação e Design da API:* Projetar e implementar interfaces de programação de aplicações (APIs) no estilo REST (Representational State Transfer). Isso envolve definir os endpoints (URLs), métodos HTTP (GET, POST, PUT, DELETE) e formatos de dados (JSON, XML) que permitirão a comunicação entre o Front-end e o Back-end. A API REST deve ser bem documentada, clara e eficiente para facilitar a integração e o uso.

- *Manutenção e Evolução da API:* Garantir que a API permaneça funcional, estável e performática ao longo do tempo. Isso inclui corrigir bugs, otimizar o código, adicionar novas funcionalidades, realizar testes de regressão para evitar que novas alterações quebrem funcionalidades existentes, e gerenciar versionamento da API (ex: v1, v2) para permitir atualizações sem quebrar a compatibilidade com versões antigas do Front-end.
- *Documentação da API:* Criar e manter a documentação da API, utilizando ferramentas como Swagger/OpenAPI ou Postman, para que os desenvolvedores Front-end (e outros sistemas, se aplicável) entendam como utilizar a API, quais endpoints estão disponíveis, quais dados enviar e como interpretar as respostas.

- **Modelagem e Gestão do Banco de Dados:**
  - *Design do Esquema do Banco de Dados:* Definir a estrutura do banco de dados, incluindo as tabelas, colunas, tipos de dados, relacionamentos entre tabelas (ex: um-para-um, um-para-muitos, muitos-para-muitos), chaves primárias e estrangeiras, índices para otimizar consultas e garantir a integridade dos dados. A modelagem deve considerar os requisitos do negócio, o volume de dados esperado, a performance e a escalabilidade.
  - *Escolha e Configuração do Banco de Dados:* Selecionar o sistema de gerenciamento de banco de dados (SGBD) mais adequado para o projeto (ex: MySQL, PostgreSQL, MongoDB, SQL Server) com base nos requisitos de performance, escalabilidade, tipo de dados e custos. Configurar e otimizar o banco de dados para garantir segurança, performance e disponibilidade.
  - *Gestão e Manutenção do Banco de Dados:* Realizar tarefas de administração do banco de dados, como backups regulares para evitar perda de dados, otimização de consultas para melhorar o desempenho, monitoramento da saúde do banco de dados, implementação de estratégias de segurança para proteger os dados, e planejamento de escalabilidade para lidar com o crescimento do volume de dados.
  - *Migração de Dados e Esquema:* Gerenciar a migração de dados entre diferentes ambientes (desenvolvimento, teste, produção) e realizar alterações no esquema do banco de dados (adicionar colunas, tabelas, etc.) de forma controlada e segura, garantindo a integridade dos dados durante o processo.

- **Implementação da Lógica de Negócios:**

- *Desenvolvimento das Regras de Negócio:* Traduzir os requisitos do negócio em código. Isso envolve implementar as regras, algoritmos e processos que definem o comportamento da aplicação. Exemplos incluem validações de dados, cálculos complexos, fluxos de trabalho, regras de decisão e orquestração de serviços.
- *Processamento de Dados:* Desenvolver rotinas para processar, transformar e manipular os dados. Isso pode incluir tarefas como validação, limpeza, enriquecimento, agregação e análise de dados. O processamento de dados deve ser eficiente e escalável para lidar com grandes volumes de informação.
- *Integração com Sistemas Externos (se necessário):* Integrar a aplicação com outros sistemas ou serviços externos, como APIs de terceiros, gateways de pagamento, sistemas de CRM, sistemas de ERP, etc. Isso envolve configurar a comunicação, trocar dados e garantir a compatibilidade entre os sistemas.

- **Desenvolvimento de Serviços de Autenticação e Segurança:**
  - *Implementação de Autenticação:* Desenvolver mecanismos para verificar a identidade dos usuários (autenticação), garantindo que apenas usuários autorizados acessem determinadas funcionalidades ou dados. Isso pode envolver a implementação de login e logout, gerenciamento de sessões, autenticação multifator (MFA), integração com provedores de identidade (ex: OAuth 2.0, OpenID Connect).
  - *Implementação de Autorização:* Desenvolver mecanismos para controlar o acesso dos usuários aos recursos da aplicação (autorização), definindo quais permissões cada usuário ou grupo de usuários possui. Isso pode envolver a implementação de controle de acesso baseado em papéis (RBAC), listas de controle de acesso (ACLs) e políticas de segurança.
  - *Implementação de Medidas de Segurança:* Implementar práticas de segurança no desenvolvimento do Back-end para proteger a aplicação contra vulnerabilidades e ataques. Isso inclui a proteção contra injeção de SQL, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), ataques de negação de serviço (DoS), e outras ameaças comuns. Garantir o armazenamento seguro de dados sensíveis, como senhas (utilizando hash e salt) e informações pessoais (utilizando criptografia).
  - *Gestão de Certificados de Segurança (HTTPS):* Configurar e gerenciar certificados SSL/TLS para habilitar HTTPS na API, garantindo a comunicação

criptografada entre o Front-end e o Back-end, protegendo os dados em trânsito contra interceptação.

2.1 Desenvolvedor Front-end

Membros: Lays Emanuelly e Daniel

Responsabilidades Detalhadas:

- **Desenvolvimento da Interface do Usuário (UI):**
  - *Implementação do Design da UI:* Traduzir os designs visuais (mockups, wireframes, protótipos) em código HTML, CSS e JavaScript (ou frameworks/bibliotecas como React, Angular, Vue.js). Isso envolve criar a estrutura da página (HTML), estilizar os elementos visuais (CSS) e adicionar interatividade (JavaScript).
  - *Criação de Elementos de UI:* Desenvolver componentes de interface do usuário reutilizáveis, como botões, formulários, menus, modais, carrosséis, etc. Esses componentes devem seguir o design system da aplicação e ser consistentes em toda a interface.
  - *Design da Experiência do Usuário (UX) - Colaboração:* Colaborar com designers de UX para garantir que a interface seja intuitiva, fácil de usar e proporcione uma boa experiência para o usuário. Isso envolve considerar a usabilidade, a acessibilidade, a arquitetura da informação e o fluxo de navegação.
- **Implementação de Componentes Reutilizáveis:**
  - *Desenvolvimento de Bibliotecas de Componentes:* Criar bibliotecas de componentes de UI reutilizáveis que podem ser utilizados em diferentes partes da aplicação. Isso promove a consistência visual, facilita a manutenção do código e acelera o desenvolvimento.
  - *Utilização de Frameworks/Bibliotecas:* Utilizar frameworks e bibliotecas JavaScript populares (ex: React, Angular, Vue.js) para estruturar o código Front-end, gerenciar o estado da aplicação, criar componentes reutilizáveis e facilitar o desenvolvimento de interfaces complexas.
  - *Manutenção e Evolução da Biblioteca de Componentes:* Manter e evoluir a biblioteca de componentes, corrigindo bugs, adicionando novas funcionalidades, melhorando a performance e garantindo a compatibilidade com as diferentes partes da aplicação.

- **Integração com a API Back-end:**
  - *Consumo da API REST:* Fazer requisições para a API REST desenvolvida pelo Back-end para obter dados, enviar dados e interagir com a lógica de negócios da aplicação. Isso envolve utilizar métodos HTTP (GET, POST, PUT, DELETE), lidar com os formatos de dados (JSON, XML) e processar as respostas da API.
  - *Gestão do Estado da Aplicação (Front-end):* Gerenciar o estado da aplicação Front-end, incluindo os dados recebidos da API, o estado da interface do usuário, e as interações do usuário. Frameworks como React, Angular e Vue.js oferecem mecanismos para gerenciar o estado de forma eficiente.
  - *Tratamento de Erros de API:* Implementar tratamento de erros para lidar com situações em que a API retorna erros (ex: falha na requisição, erro de servidor, dados inválidos). Isso envolve exibir mensagens de erro amigáveis para o usuário, registrar os erros para análise e tentar recuperar a aplicação de forma graciosa.
- **Garantia de Responsividade e Acessibilidade:**
  - *Design Responsivo:* Desenvolver a interface do usuário de forma que ela se adapte a diferentes tamanhos de tela e dispositivos (desktops, tablets, smartphones). Isso envolve utilizar técnicas de CSS como Media Queries e layouts flexíveis (Flexbox, Grid) para garantir que a interface seja visualmente agradável e funcional em qualquer dispositivo.
  - *Acessibilidade Web (WCAG):* Garantir que a interface do usuário seja acessível para pessoas com deficiência, seguindo as diretrizes de acessibilidade web (WCAG - Web Content Accessibility Guidelines). Isso inclui garantir a navegação por teclado, fornecer alternativas textuais para imagens, utilizar contraste de cores adequado, estruturar o conteúdo semanticamente e utilizar tecnologias assistivas (ex: leitores de tela).
  - *Testes em Diferentes Dispositivos e Navegadores:* Testar a interface do usuário em diferentes dispositivos (desktops, smartphones, tablets) e navegadores (Chrome, Firefox, Safari, Edge) para garantir a compatibilidade e identificar e corrigir problemas de layout, funcionalidade ou performance.

3 Artefatos Gerados

Ao longo do desenvolvimento deste projeto, produzimos uma série de artefatos que não apenas guiaram nossas ações, mas também consolidaram nosso entendimento sobre o problema e a solução proposta. Esses artefatos representam o esforço colaborativo da equipe e refletem nosso compromisso em entregar um produto de qualidade. A seguir, detalharemos cada um deles.

3.1 Inception

A fase de *Inception* foi crucial para alinharmos nossas expectativas e definirmos os rumos do projeto. Foi um período de imersão e descobertas, onde buscamos compreender profundamente o mercado, os usuários e as necessidades reais a serem atendidas.

3.1.1 Pesquisa de Mercado

A pesquisa de mercado revelou uma necessidade crescente por uma plataforma integrada que facilite a interação entre clínicas veterinárias e seus clientes. Atualmente, muitos sistemas existentes apresentam limitações, dificultando o agendamento de consultas, o acesso ao histórico médico dos pets e a localização de clínicas.

A evolução dos sistemas para clínicas veterinárias começou nos anos 90, com softwares focados em agendamentos e controle de estoque. Com o avanço da tecnologia, especialmente entre 2010 e 2020, a popularização da internet e dos smartphones impulsionou o desenvolvimento de aplicativos móveis, permitindo agendamentos online, lembretes automáticos e a expansão da telemedicina. Atualmente, as principais tendências tecnológicas apontam para uma adoção ainda maior da telemedicina, o crescimento do uso de aplicativos móveis para gestão de cuidados veterinários e a aplicação de inteligência artificial para otimizar a previsão de demanda e a personalização dos serviços.

Diante dessa análise, conclui-se que há uma demanda clara por uma solução integrada, como o VetManager, que unifique o agendamento, o gerenciamento do histórico médico e a localização de clínicas, oferecendo uma interface moderna e intuitiva.

3.1.2 Descrição de Personas e Mapas de Empatia

Desenvolvemos duas personas, Márcio e João, para representar perfis distintos de usuários que compartilham o interesse em serviços veterinários de qualidade, mas com necessidades e expectativas diferentes

- Márcio, o veterinário: Busca eficiência e praticidade na gestão da clínica, valorizando acesso rápido a informações dos pacientes, ferramentas de organização e atualização profissional. Motivado por oferecer o melhor atendimento e construir sua reputação.
- João, o tutor: Busca praticidade e agilidade no cuidado com seus pets, valorizando agendamentos online, lembretes automáticos e atendimento de qualidade, incluindo teleconsultas. Motivado pelo bem-estar dos animais e otimização de seu tempo.

Suas necessidades direcionam o VetManager, com funcionalidades como prontuários digitais e gestão de consultas para Márcio, e agendamentos online e teleconsultas para João. Considerar ambos garante que o VetManager atenda às demandas do mercado, sendo uma solução completa para veterinários e tutores.

Através de mapas de empatia, conseguimos nos colocar no lugar deles, entendendo suas dores, desejos e motivações. O Dr. Márcio prioriza a gestão interna, buscando otimizar processos como organização de prontuários e redução de tarefas administrativas, para dedicar mais tempo aos cuidados com os animais. Ele valoriza a qualidade do atendimento e a reputação da clínica.

Já João, como cliente, busca praticidade no agendamento de consultas, acesso fácil ao histórico de seus pets e um sistema de lembretes eficiente. Ele também valoriza a confiança e a segurança no armazenamento dos dados dos animais. Ambos reconhecem a tecnologia como essencial para melhorar a eficiência e a experiência.

3.1.3 Visão de Produto

Definimos a visão de produto que serviu como norte para todas as decisões tomadas. Essa visão encapsula o propósito do projeto e o valor que pretendemos entregar aos usuários. Clínicas veterinárias precisam de uma forma mais eficiente de gerenciar agendamentos e acessar o histórico médico dos animais, enquanto os donos de animais buscam uma forma prática de agendar consultas e ter acesso ao histórico de seus pets. O VetManager facilita o agendamento de consultas, organiza o histórico médico dos animais e melhora a comunicação entre a clínica e o cliente, resolvendo essas necessidades e proporcionando uma experiência completa e otimizada para o setor.

3.1.4 Caixa do Produto

Criamos uma representação visual da *caixa do produto*, como se fosse ser vendida em uma prateleira. Esse exercício nos ajudou a sintetizar os principais benefícios e diferenciais do produto. Identificamos os principais benefícios do aplicativo, que incluem:

- acesso rápido à localização de clínicas veterinária
- agendamento simplificado de consultas online
- histórico completo de atendimentos dos pets

Também resumimos as funcionalidades do serviço, como detalhes sobre a localização das clínicas com mapa interativo e cálculo de rotas, a possibilidade de marcar consultas online com visualização de horários disponíveis, e a organização e exibição do histórico de consultas, diagnósticos e tratamentos dos animais

3.1.5 Diagrama de Navegação, Rabisco Frame, Jornada do Usuário

Para proporcionar uma experiência verdadeiramente intuitiva e eficiente, concentramos nossos esforços em desenvolver três elementos essenciais no projeto: o diagrama de navegação, o wireframe (rabisco frame) e a jornada do usuário.

No diagrama de navegação, organizamos cuidadosamente as telas e os fluxos do sistema, mapeando como o usuário transita entre funcionalidades como agendamento de consultas, acesso ao histórico médico e localização de clínicas. Esse processo nos permitiu estruturar a plataforma de maneira clara e lógica, evitando caminhos confusos ou redundantes que poderiam frustrar o usuário.

O wireframe foi criado para esboçar a interface antes mesmo de qualquer linha de código ser escrita. Nele, definimos onde cada elemento ficaria na tela—botões, menus, áreas de conteúdo—garantindo que a disposição fosse intuitiva e a navegação fluida. Essa visualização antecipada nos ajudou a identificar e corrigir possíveis problemas de usabilidade desde o início.

Já na jornada do usuário, nos colocamos no lugar do cliente, simulando cada passo que ele daria dentro da plataforma, desde o primeiro acesso até o agendamento de uma consulta. Esse exercício foi fundamental para identificar dificuldades, antecipar necessidades e otimizar cada interação, tornando o sistema não apenas eficiente, mas também agradável de usar.

3.1.6 Flat Backlog ou Backlog Preliminar

Resumindo nosso flat backlog, sintetizamos as funcionalidades-chave que seriam desenvolvidas no projeto. Focamos no cadastro e autenticação de usuários, gerenciamento de clínicas e pets, além de implementar ferramentas para agendamento de consultas e acesso a históricos médicos. Esse resumo nos forneceu uma visão clara das tarefas a serem

realizadas, auxiliando no planejamento eficiente das próximas etapas e na priorização adequada das atividades.

4 Projeto e Prototipação

Nessa etapa, transformamos nossas ideias em estruturas concretas, preparando o terreno para a implementação.

4.1 Documento Geral de Arquitetura

O VetManager é um aplicativo mobile desenvolvido em Dart, utilizando o Flutter para sua interface de usuário. O backend é baseado em Node.js e Express, com PostgreSQL e Prisma como tecnologias principais para gerenciamento de banco de dados. Este documento descreve a arquitetura geral do projeto, destacando as principais escolhas tecnológicas e as razões por trás delas.

4.1.1 Tecnologias Utilizadas

4.1.2 Backend

- **Node.js**: Escolhido pelo seu desempenho eficiente em operações de I/O não bloqueantes e pelo ecossistema robusto.
- **Express**: Framework minimalista para criação de API REST, oferecendo flexibilidade e suporte a middlewares essenciais:
  - Parsing de JSON
  - Gerenciamento de CORS
  - Compressão de respostas
  - Logging de requisições
- **Zod**: Biblioteca para validação de schemas com tipagem estática, garantindo:
  - Validação em runtime de:
    - Parâmetros de requisição
    - Corpos de POST/PUT
    - Query parameters
  - Mensagens de erro customizáveis

  - Integração com TypeScript

4.1.3 Banco de Dados

- **PostgreSQL**: Sistema de Gerenciamento de Banco de Dados Relacional (SGBD) confiável e robusto, suportando operações complexas.
- **Prisma**: ORM moderno que simplifica a interação com o banco de dados, oferecendo:
  - Type-safe database client
  - Migrações automatizadas
  - Geração automática de tipos
  - Query builder intuitivo
  - Definição clara de entidades via `schema.prisma`

4.1.4 Autenticação e Segurança

- **JWT (JSON Web Tokens)**: Implementação stateless de autenticação, garantindo segurança através de:
  - Tokens assinados com chave secreta HMAC
  - Expiração configurável
  - Middleware de verificação em rotas protegidas
- **Bcrypt**: Para armazenamento seguro de senhas, oferecendo:
  - Salt automático com 12 rounds
  - Funções assíncronas para hashing e verificação
  - Armazenamento seguro no banco de dados

4.1.5 Documentação da API

- **Swagger/OpenAPI**:
  - Documentação interativa gerada automaticamente via `swagger-ui-express`
  - Descrição detalhada dos endpoints
  - Exemplos de requisições e respostas
  - Códigos de status HTTP explicativos

4.1.6 Deploy e Ambiente de Produção

- **Render**: Plataforma cloud para deploy contínuo, oferecendo:
  - Integração com repositórios Git
  - Builds automáticos a cada atualização
  - Gerenciamento de variáveis de ambiente
  - Health checks configurados
  - Escalabilidade horizontal automática
- **CI/CD**:
  - Testes automatizados pré-deploy
  - Execução de migrações Prisma durante o build
  - Rollback automático em caso de falha

4.1.7 Fluxo de Autenticação

1. O usuário envia credenciais para o endpoint `/login`.
2. As credenciais são verificadas no banco de dados.
3. Se corretas, um token JWT é gerado e enviado.
4. Para rotas protegidas, o middleware valida o JWT antes de permitir acesso.

4.1.8 Considerações Finais

O VetManager foi projetado para ser uma aplicação:

- **Robusta e segura**: Com camadas de segurança bem definidas.
- **Escalável**: Preparada para crescer conforme a demanda aumenta.
- **Manutenível**: Com separação clara de responsabilidades e uso de ferramentas que facilitam o desenvolvimento.
- **Bem documentada**: Garantindo uma curva de aprendizado mais suave para novos desenvolvedores.
- **Preparada para o futuro**: Arquitetura modular e flexível para futuras melhorias.

Esse documento serve como guia para a equipe de desenvolvimento e também como referência para futuras expansões do projeto.

4.2 Sprint Backlog

Organizamos o desenvolvimento em três sprints, cada uma focada em funcionalidades-chave que impulsionaram o progresso do projeto:

- **Sprint 1**: Criamos a interface básica para os usuários se cadastrarem, realizarem login e visualizarem informações essenciais. Implementamos o cadastro e a listagem de clínicas, bem como a adição e exibição dos dados básicos dos pets.
- **Sprint 2**: Estabelecemos a conexão do sistema com o banco de dados, assegurando a persistência e integridade das informações. Detalhamos as informações de cada clínica e implementamos as operações de CRUD para os pets. Também desenvolvemos as telas principais da aplicação móvel, aprimorando a experiência do usuário.
- **Sprint 3**: Implementamos a lógica de agendamento de consultas, incluindo a criação de interfaces intuitivas para cadastro e edição de pets. Incorporamos um mapa interativo para exibir as clínicas e desenvolvemos a funcionalidade de histórico, permitindo aos usuários visualizar o histórico completo de seus pets.

Esse planejamento em sprints nos permitiu avançar de forma estruturada, assegurando que as funcionalidades mais importantes fossem priorizadas e entregues com qualidade. Cada sprint contribuiu significativamente para a maturidade do projeto, alinhando-se aos objetivos iniciais e às necessidades dos usuários.

4.2.1 Modelo Entidade-Relacionamento ou Equivalente

Desenhamos o Modelo Entidade-Relacionamento (ER) para representar as entidades e seus relacionamentos no banco de dados. Esse modelo foi fundamental para garantir a integridade e eficiência dos dados.

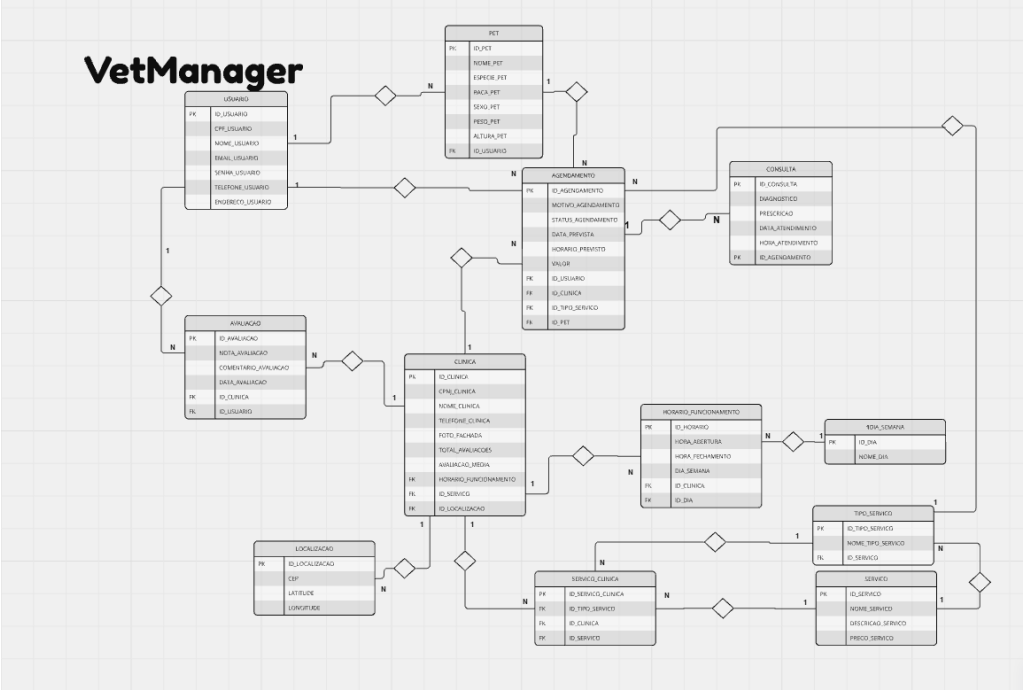


Figura 1: MODELO DE ENTIDADE E RELACIONAMENTOs

4.3 Implementação

4.3.1 Modelo de Banco de Dados Físico ou Esquema

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model Usuario {
  id_usuario    Int           @id @default(autoincrement())
  cpf_usuario   String        @unique
  nome_usuario  String
  email_usuario String        @unique
  senha_usuario String
  foto_usuario  String
  Avaliacao     Avaliacao[]
  Pet           Pet[]
  Agendamento   Agendamento[]
}

model Pet {
  id_pet            Int           @id @default(autoincrement())
  nome_pet          String
  especie_pet       String
  raca_pet          String
  sexo_pet          String
  peso_pet          Float
  altura_pet        Float
  id_usuario        Usuario       @relation(fields: [usuarioId_usuario], references: [id_usuario])
  usuarioId_usuario Int
  Agendamento       Agendamento[]
}

model Clinica {
  id_clinica       Int                   @id
  cnpj_clinica     String                @unique
  nome_clinica     String
  telefone_clinica String
  foto_clinica     String
  avaliacao_clinica Float
  total_avaliacoes  Int
  Avaliacao        Avaliacao[]
  Agendamento      Agendamento[]
  Horarios         HorarioFuncionamento[]
  id_localizacao   LocalizacaoClinica[]
  ServicoClinica   ServicoClinica[]
}

model ServicoClinica {
  id_servico_clinica         Int         @id
  tipoServicoId              TipoServico @relation(fields: [tipoServicoId_tipo_servico], references: [id_tipo_servico])
  clinicaId                  Int
  clinica                    Clinica     @relation(fields: [clinicaId], references: [id_clinica])
  tipoServicoId_tipo_servico Int
  Servico                    Servico?    @relation(fields: [servicoId_servico], references: [id_servico])
  servicoId_servico          Int?
}

model TipoServico {
  id_tipo_servico   Int             @id
  nome_tipo         String
  servicoId_servico Int
  servico           Servico         @relation(fields: [servicoId_servico], references: [id_servico])
  ServicoClinica    ServicoClinica[]
  Agendamento       Agendamento[]
}

model Servico {
  id_servico        Int             @id
  nome_servico      String
  descricao_servico String
  preco_servico     Float
  tipos_servico     TipoServico[]
  Agendamento       Agendamento[]
  ServicoClinica    ServicoClinica[]
}

model LocalizacaoClinica {
  id_localizacao    Int     @id @default(autoincrement())
  latitude          Float
  longitude         Float
  endereco          String
  cidade            String
  estado            String
  cep               String
  clinicaId_clinica Int
  Clinica           Clinica @relation(fields: [clinicaId_clinica], references: [id_clinica])
}

model DiaSemana {
  id_dia   Int                   @id
  nome_dia String                @unique
  Horarios HorarioFuncionamento[]
}

model HorarioFuncionamento {
  id_horario     Int       @id @unique @default(autoincrement())
  horario_inicio String
  horario_fim    String
  id_dia         Int
  dia            DiaSemana @relation(fields: [id_dia], references: [id_dia])
  Clinica        Clinica?  @relation(fields: [clinicaId], references: [id_clinica])
  clinicaId      Int?
}

model Avaliacao {
  id_avaliacao         Int      @id @default(autoincrement())
  nota_avaliacao       Int
  comentario_avaliacao String
  data_avaliacao       DateTime
  id_clinica           Clinica  @relation(fields: [clinicaId_clinica], references: [id_clinica])
  id_usuario           Usuario  @relation(fields: [usuarioId_usuario], references: [id_usuario])
  clinicaId_clinica    Int
  usuarioId_usuario    Int
}

model Agendamento {
  id_agendamento            Int               @id @default(autoincrement())
  data_agendamento          DateTime
  horario_agendamento       String
  status_agendamento        String
  id_tipo_servico           TipoServico       @relation(fields: [tipoServicoId_tipo_servico], references: [id_tipo_servico])
```

4.3.2 Relatório de Bugs

Mantivemos um relatório detalhado de bugs, documentando cada problema encontrado, o impacto e as medidas tomadas para resolvê-lo. Esse processo contínuo de depuração foi essencial para melhorar a qualidade do produto. Este relatório documenta os principais bugs encontrados durante o desenvolvimento do Vet Manager, suas causas, impactos e soluções aplicadas.

4.3.3 Lista de Bugs

ID	Descrição	Impacto	Status
001	Erro ao cadastrar novo usuário (CPF duplicado)	Alto	Resolvido
002	Imagens de pets não carregam corretamente	Médio	Em andamento
003	Agendamento não exibe horário corretamente	Alto	Resolvido
004	Erro 500 ao buscar avaliações de clínicas	Alto	Resolvido
005	Botão de atualização de fotos não funciona	Médio	Em andamento
006	Dados de localização da clínica não são salvos	Alto	Resolvido
007	Lógica para agendamento	Alto	Em andamento

Tabela 1: Bugs identificados no Vet Manager

4.3.4 Descrição Detalhada de Alguns Bugs

4.3.5 Erro ao cadastrar novo usuário (CPF duplicado)

Descrição: O sistema não permitia o cadastro de usuários com CPFs já existentes no banco de dados, mas a validação falhava em alguns casos.

Causa: Conflito entre validação frontend e backend.

Solução: Implementação de uma verificação mais robusta no backend.

4.3.6 Imagens de pets não carregam corretamente

Descrição: Algumas imagens de pets não aparecem na tela do usuário.

Causa: Caminho das imagens gerado incorretamente.

Solução: Ajuste na lógica de geração de URLs de imagens.

4.3.7 Lógica de agendamentos não funciona corretamente

Descrição: O sistema de agendamentos não está registrando ou exibindo os agendamentos de forma precisa, causando falhas na visualização e no gerenciamento dos compromissos.

Causa: Erro na lógica de validação e armazenamento das informações de agendamento, resultando em dados inconsistentes ou em falhas ao salvar as entradas.

Solução: Revisão e correção da lógica de validação de dados e ajustes no processo de armazenamento de agendamentos no banco de dados.

4.3.8 Conclusão

A correção dos bugs listados foi essencial para a estabilidade do sistema. O monitoramento contínuo e a realização de testes garantirão a redução de novos problemas.

4.3.9 Principais Telas do Sistema Funcionando

Desenvolvemos as principais telas do sistema, garantindo funcionalidade e usabilidade. Essas telas passaram por testes rigorosos para assegurar que atendessem às necessidades dos usuários.

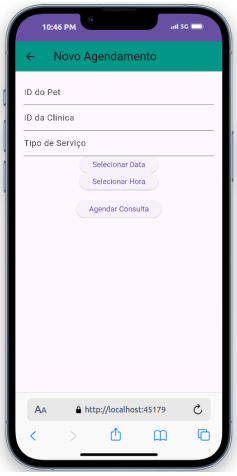


Figura 3: Agendamento

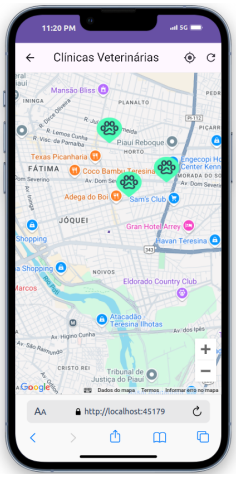


Figura 4: Clínicas

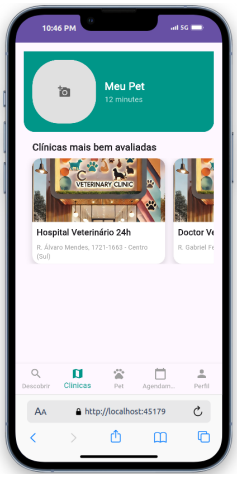


Figura 5: Home

Figura 6: Login

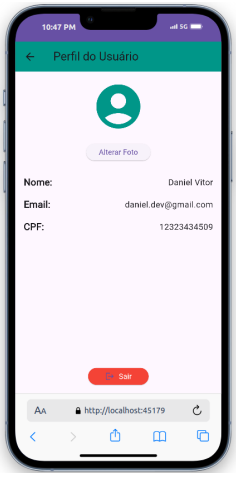


Figura 7: Perfil

Figura 8: Visão geral das telas do sistema.

4.3.10 Link para o Repositório

Todo o código-fonte está versionado e disponível em nossos repositórios oficiais. Isso facilita a colaboração e permite o acompanhamento das evoluções do projeto. Os repositórios podem ser acessados através dos seguintes links:

- **Aplicativo**: <https://github.com/Luna0812y/vet_manager_app>
- **API**: <https://github.com/alexsousadev/vetmanager>

4.4 Considerações Finais

A geração desses artefatos foi mais do que uma obrigação; foi parte de um processo enriquecedor que nos permitiu amadurecer a ideia inicial e transformá-la em uma solução palpável. Cada documento, modelo ou diagrama contribuíram para o alinhamento da equipe e a clareza dos objetivos. Acreditamos que esse conjunto de artefatos será valioso não apenas para a continuidade do projeto, mas também como referência para trabalhos futuros.

5 Detalhamento da Implementação no Back-end

Desenvolver um sistema robusto e eficiente requer mais do que apenas linhas de código; é necessário compreender profundamente as necessidades dos usuários e traduzir essas necessidades em soluções tecnológicas elegantes. A arquitetura do back-end deste projeto foi concebida com esse propósito em mente, adotando princípios modernos de escalabilidade e manutenibilidade. Utilizamos uma combinação estratégica de tecnologias especializadas, sempre com o objetivo de proporcionar a melhor experiência possível. O sistema foi estruturado seguindo o padrão **MVC (Model-View-Controller)**, garantindo uma separação clara de responsabilidades e facilitando a colaboração entre as equipes.

5.1 Arquitetura do Sistema

Para criar uma base sólida, organizamos a estrutura do projeto em camadas distintas, cada uma desempenhando um papel fundamental no funcionamento do sistema:

- **Models**: Aqui definimos as entidades de negócio e o schema do banco de dados. Esta camada representa o coração dos dados, modelando as informações que são cruciais para o sistema.

- **Controllers**: Responsáveis pela lógica de negócios e manipulação de requisições, os controllers atuam como o cérebro do sistema, tomando decisões com base nas interações dos usuários.
- **Routes**: Nesta camada, definimos os endpoints da API e o roteamento. É como o sistema entende para onde cada requisição deve ir, garantindo que as informações cheguem ao destino correto.
- **Services**: Aqui realizamos operações complexas e integrações externas. Os services atuam como os braços do sistema, alcançando outras partes e serviços necessários.
- **Middlewares**: Tratamos a autenticação e validações nesta camada, assegurando que somente usuários autorizados tenham acesso e que os dados sejam confiáveis.

5.2 Tecnologias e Componentes Principais

Optamos por tecnologias que não apenas são reconhecidas no mercado, mas que também ofereciam as melhores soluções para os desafios que enfrentamos.

5.2.1 Node.js e Express

- **Node.js**: Escolhemos o Node.jscomo nosso ambiente de execução server-side por sua eficiência em operações de I/O não bloqueantes e por seu ecossistema robusto. A capacidade de lidar com múltiplas requisições simultâneas de forma eficiente foi crucial.
- **Express**: Utilizamos o Express como framework minimalista para a criação da API REST. Sua flexibilidade nos permitiu configurar middlewares essenciais, tais como:
  - Parsing de JSON: Para interpretar corretamente as requisições dos clientes.
  - Gerenciamento de CORS: Assegurando que nossa API seja acessível de diferentes origens de forma segura.
  - Compressão de respostas: Otimizando o tempo de carregamento e economizando banda.
  - Logging de requisições: Mantendo um registro detalhado das interações para monitoramento e depuração.

5.2.2 Prisma e PostgreSQL

- **PostgreSQL**: Optamos pelo PostgreSQL como nosso Sistema de Gerenciamento de Banco de Dados Relacional devido à sua confiabilidade, robustez e suporte a operações complexas. Sua comunidade ativa também foi um diferencial.
- **Prisma**: O Prisma se destacou como um ORM moderno que simplificou nossa interação com o banco de dados. Entre seus benefícios, destacamos:
  - **Type-safe database client**: Garantindo segurança de tipos em nossas consultas, reduzindo erros em tempo de execução.
  - **Migrações automatizadas**: Facilitando a evolução do schema do banco de dados conforme o projeto crescia.
  - **Geração automática de tipos**: Simplificando o desenvolvimento e mantendo a consistência.
  - **Query builder intuitivo**: Tornando as consultas mais legíveis e fáceis de escrever.
- Criamos um `schema.prisma` contendo a definição completa das entidades e suas relações, funcionando como um mapa detalhado de nossa estrutura de dados.

5.3 Autenticação e Segurança

Segurança não é apenas uma funcionalidade; é um compromisso contínuo com nossos usuários. Implementamos medidas para proteger informações sensíveis e garantir que apenas usuários autorizados tenham acesso.

5.3.1 JWT e Bcrypt

- **JWT (JSON Web Tokens)**: Optamos por uma implementação stateless para autenticação, o que trouxe várias vantagens:
  - Tokens assinados com chave secreta HMAC, aumentando a segurança.
  - Expiração configurável, permitindo controlar a duração das sessões.
  - Middleware de verificação em rotas protegidas, assegurando que cada requisição seja autenticada.
- **Bcrypt**: Para o armazenamento seguro de senhas, utilizamos o Bcrypt, que nos ofereceu:

  - Salt automático com 12 rounds, tornando os hashes mais robustos contra ataques de força bruta.
  - Funções assíncronas para hashing e verificação, melhorando o desempenho.
  - Armazenamento seguro no banco de dados, garantindo a integridade das credenciais dos usuários.

5.4 Validação de Dados

A qualidade dos dados é fundamental para o bom funcionamento do sistema. Erros e inconsistências podem levar a problemas graves, por isso implementamos validações rigorosas.

5.4.1 Zod

- Utilizamos o **Zod** para schema validation com tipagem estática, o que nos permitiu:
  - Realizar validação em runtime de:
    * Parâmetros de requisição: Garantindo que estamos recebendo os dados esperados.
    * Corpos de POST/PUT: Validando os dados enviados pelos usuários antes de processá-los.
    * Query parameters: Assegurando consultas corretas e seguras.
  - Fornecer mensagens de erro customizáveis, melhorando a experiência do desenvolvedor na identificação de problemas.
  - Integrar com o TypeScript para inferência de tipos, mantendo a consistência em todo o código.

5.5 Documentação da API

A documentação é o ponto de contato inicial para muitos desenvolvedores que utilizarão nossa API. Queremos que esse primeiro contato seja claro, eficiente e agradável.

5.5.1 Swagger/OpenAPI

- Geramos uma documentação interativa automaticamente utilizando o **Swagger**, seguindo a especificação **OpenAPI 3.0**.

- Integramos via `swagger-ui-express`, permitindo que os usuários experimentem os endpoints diretamente pelo navegador.
- A documentação fornece uma descrição detalhada de:
  - Endpoints disponíveis: Facilitando a descoberta de funcionalidades.
  - Parâmetros esperados: Especificando exatamente o que cada endpoint necessita.
  - Exemplos de requisições e respostas: Ajudando no entendimento de como interagir com a API.
  - Códigos de status HTTP: Informando sobre possíveis respostas e erros.

5.6 Deploy e Ambiente de Produção

Levar a aplicação do ambiente de desenvolvimento para produção é um passo crucial que requer cuidado e planejamento. Nossa meta foi tornar esse processo o mais suave e automatizado possível.

5.6.1 Render

- Escolhemos o **Render** como plataforma cloud para deploy contínuo, o que nos ofereceu:
  - Integração direta com nosso repositório Git, facilitando o fluxo de trabalho.
  - Builds automáticos a cada atualização, garantindo que o ambiente de produção reflita o estado atual do código.
  - Gerenciamento de variáveis de ambiente, mantendo informações sensíveis seguras.
  - Health checks configurados, monitorando a saúde da aplicação em tempo real.
  - Escalabilidade horizontal automática, permitindo que a aplicação suporte um aumento de demanda sem problemas.
- Implementamos um pipeline de **CI/CD** que abrange:
  - Testes automatizados pré-deploy, para assegurar que novas alterações não introduzam bugs.
  - Execução de migrações do Prisma durante o build, mantendo o banco de dados atualizado.

  - Rollback automático em caso de falha, minimizando o tempo de inatividade.

5.7 Considerações Finais

Ao longo deste projeto, buscamos não apenas construir uma aplicação funcional, mas também criar algo que seja sustentável e que possa evoluir com as necessidades futuras. A combinação das tecnologias e práticas que adotamos resultou em uma API:

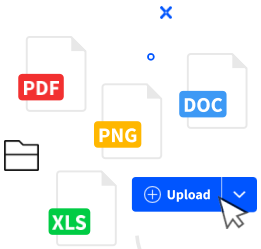
- **Robusta e segura**: Com camadas de segurança implementadas desde o início.
- **Escalável**: Preparada para crescer conforme a demanda aumentar.
- **Manutenível**: Graças à separação clara de responsabilidades e ao uso de ferramentas que facilitam o desenvolvimento.
- **Bem documentada**: Facilitando a vida de outros desenvolvedores que trabalharão com a API.
- **Preparada para o futuro**: Com uma arquitetura que permite a adição de novas funcionalidades sem grandes refatorações.

Referências

Anexo I

Bem-vindo ao Smallpdf

Pronto(a) para levar o gerenciamento de documentos até ao próximo nível?



1

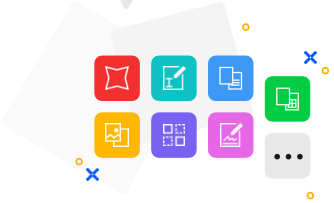
Documentos digitais - tudo em um só lugar

Com a nova experiência Smallpdf, você pode fazer upload, organizar e compartilhar documentos digitais livremente. Quando você ativa a opção "Armazenamento", todos os arquivos processados são salvos nessa área.

2

Melhore documentos em um clique

Ao clicar com o botão direito do mouse em um arquivo, você verá uma série de opções para convertê-lo, comprimi-lo ou modificá-lo.



3

Acesse os arquivos a qualquer hora e em qualquer lugar

Você pode acessar arquivos salvos no Smallpdf em seu computador, celular ou tablet. Também sincronizamos arquivos do Smallpdf Mobile App com nosso portal online.

4

Colabore com outros

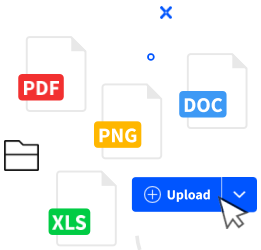
Esqueça as tarefas administrativas mundanas. Com o Smallpdf, você pode solicitar assinaturas eletrônicas, enviar arquivos grandes ou até mesmo ativar o aplicativo Smallpdf G Suite para toda sua organização.

Anexo 9: Há um PDF acima

Anexo II

Bem-vindo ao Smallpdf

Pronto(a) para levar o gerenciamento de documentos até ao próximo nível?



1

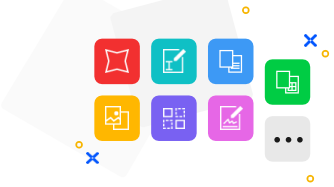
Documentos digitais - tudo em um só lugar

Com a nova experiência Smallpdf, você pode fazer upload, organizar e compartilhar documentos digitais livremente. Quando você ativa a opção "Armazenamento", todos os arquivos processados são salvos nessa área.

2

Melhore documentos em um clique

Ao clicar com o botão direito do mouse em um arquivo, você verá uma série de opções para convertê-lo, comprimi-lo ou modificá-lo.



3

Acesse os arquivos a qualquer hora e em qualquer lugar

Você pode acessar arquivos salvos no Smallpdf em seu computador, celular ou tablet. Também sincronizamos arquivos do Smallpdf Mobile App com nosso portal online.

4

Colabore com outros

Esqueça as tarefas administrativas mundanas. Com o Smallpdf, você pode solicitar assinaturas eletrônicas, enviar arquivos grandes ou até mesmo ativar o aplicativo Smallpdf G Suite para toda sua organização.