



Relatório Técnico - Projeto PetDex

★ Início Rápido - Teste o Aplicativo Agora!

Quer testar o PetDex imediatamente? Siga estes passos:

📱 Instalação do APK (Android)

1. **Baixe o APK:** [PetDex APK - Google Drive](#)
2. **Habilite "Fontes Desconhecidas"** nas configurações do seu celular
3. **Instale o APK** baixado
4. **Abra o aplicativo**

🔑 Credenciais de Login

Use estas credenciais para acessar o aplicativo com dados reais:

✉ Email: henriquealmeidaflorentino@gmail.com
🔒 Senha: senha123

Pronto! Você terá acesso completo a todas as funcionalidades do PetDex com dados reais de monitoramento.

💡 Dica: Para instruções detalhadas de instalação e solução de problemas, consulte a Seção 5.5.

📄 Índice

★ [Início Rápido - Teste o Aplicativo Agora!](#)

📱 [Instalação do APK \(Android\)](#)

🔑 [Credenciais de Login](#)

📄 [Índice](#)

1. [Overview Geral do Projeto](#)




- [1.1 Descrição do PetDex](#)
- [1.2 Problema que o Projeto Resolve](#)
- [1.3 Objetivo Principal e Proposta de Valor](#)
- [1.4 Visão Geral da Arquitetura](#)
- [1.5 Stack Tecnológico](#)

2. [Demonstração das Funcionalidades do Aplicativo](#)

- [2.1 Tela Inicial - Monitoramento em Tempo Real](#)
- [2.2 Tela de Saúde - Análises Estatísticas Detalhadas](#)
- [2.3 Tela de Checkup Inteligente - IA](#)
- [2.4 Tela de Localização - Área Segura](#)

3. [Implementações por Área](#)

- [3.1 Computação em Nuvem](#)
 - [3.1.1 Infraestrutura de Nuvem Utilizada](#)
 - [3.1.2 Containerização com Docker](#)

3.1.3	CI/CD - Deploy Automático
3.1.4	MongoDB Atlas - Banco de Dados em Nuvem
3.2	Desenvolvimento Mobile (Flutter)
3.2.1	Visão Geral da Estrutura do Aplicativo
3.2.2	Arquitetura de Pastas e Organização do Código
3.2.3	Principais Implementações e Funcionalidades
3.2.4	Padrões de Design Utilizados
3.2.5	Tecnologias e Pacotes Utilizados
3.3	Inteligência Artificial
3.3.1	Visão Geral da Implementação de IA
3.3.2	Modelo de Classificação: CART (Árvore de Decisão)
3.3.3	Processo de Desenvolvimento do Modelo
3.3.4	Dataset e Técnicas de Treinamento
3.3.5	Framework e Bibliotecas Utilizadas
3.3.6	Formato PMML: Portabilidade Universal
3.3.7	Integração da IA com as APIs
3.3.8	Resultados e Métricas de Performance
3.3.9	Documentação Completa da IA
3.4	Internet das Coisas (IoT) - Hardware da Coleira Inteligente
3.4.1	Visão Geral do Hardware
3.4.2	Componentes de Hardware
3.4.3	Coleira em Uso
3.4.4	Fluxo de Dados do Hardware
3.4.5	Código de Exemplo do ESP32
3.4.6	Desafios e Soluções de Hardware
4.	APIs e Comunicação
4.1	Visão Geral das APIs
4.2	API Java (Spring Boot)
4.2.1	Tecnologias Utilizadas
4.2.2	Arquitetura DDD (Domain-Driven Design)
4.2.3	Principais Endpoints da API Java
4.3	API Python (FastAPI)
4.3.1	Tecnologias Utilizadas
4.3.2	Estrutura do Projeto
4.3.3	Principais Endpoints da API Python
4.4	Comunicação entre as APIs
4.4.1	Fluxo de Dados
4.4.2	Autenticação JWT - Fluxo Completo
4.4.3	WebSocket - Comunicação em Tempo Real
5.	Guia de Configuração e Execução Local
5.1	Pré-requisitos
5.2	Configuração e Execução da API Java
5.3	Configuração e Execução da API Python
5.4	Configuração e Execução do Aplicativo Flutter
5.5	Instalação do APK no Celular Android
5.5.1	Download do APK
5.5.2	Passo a Passo para Instalação
5.5.3	Permissões Necessárias
5.5.4	Requisitos do Dispositivo
5.5.5	Configuração Inicial
5.5.6	Solução de Problemas
5.6	Usuários e Autenticação
5.6.1	🔑 Credenciais Padrão para Login
5.6.2	Como Fazer Login no Aplicativo
5.6.3	Por Que Usar Este Usuário?
5.6.4	Criação de Novos Usuários
5.6.5	⚠️ IMPORTANTE: Limitação de Novos Usuários
	Conclusão
	Referências e Documentação Adicional
	Equipe de Desenvolvimento

1. Overview Geral do Projeto

1.1 Descrição do PetDex

O **PetDex** é uma solução completa de **IoT + Mobile + IA** desenvolvida para o **monitoramento em tempo real da saúde e segurança de cães e gatos**. O projeto foi criado para resolver um problema crítico enfrentado por tutores de pets: a falta de soluções acessíveis e contínuas para monitoramento de saúde animal, especialmente para animais com doenças crônicas ou em tratamento.



1.2 Problema que o Projeto Resolve

Atualmente, existe uma carência de soluções que permitam:

- **Monitoramento contínuo** da saúde de animais domésticos
- **Deteção precoce** de alterações fisiológicas que podem indicar problemas de saúde
- **Localização em tempo real** de pets perdidos
- **Alertas automáticos** para situações de risco

1.3 Objetivo Principal e Proposta de Valor

Objetivo: Fornecer aos tutores de pets uma ferramenta completa de monitoramento que combine:

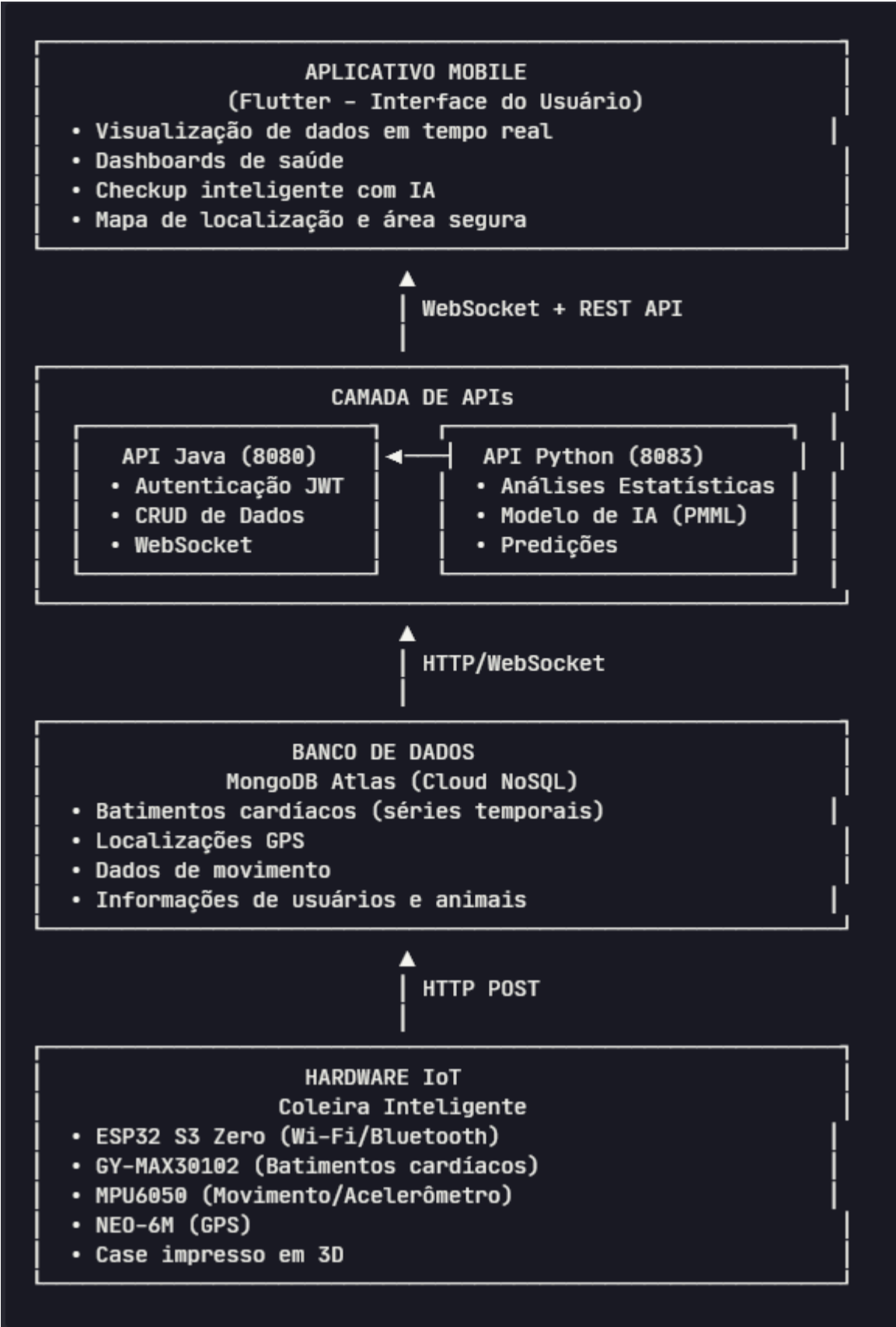
- **✓ Prevenção:** Deteção precoce de anomalias de saúde
- **✓ Segurança:** Rastreamento GPS e alertas de fuga
- **✓ Cuidado Contínuo:** Monitoramento 24/7 sem intervenção manual
- **✓ Inteligência:** Análises baseadas em IA para prevenção de doenças

Proposta de Valor:

- Monitoramento contínuo da saúde do pet através dos sensores de saúde
- Localizar pets perdidos rapidamente através do GPS integrado
- Fornecer alertas ao usuário caso o pet saia da área considerada como segura
- Fornecer histórico completo de saúde do animal
- Oferecer checkup inteligente baseado em sintomas observados

1.4 Visão Geral da Arquitetura

O PetDex é composto por **4 camadas principais** que trabalham de forma integrada:



Fluxo de Dados:

1. **Coleira IoT** coleta dados dos sensores (batimentos, movimento, GPS)
2. **API Java** recebe e armazena os dados no MongoDB
3. **API Python** processa e analisa os dados com estatísticas e IA
4. **Aplicativo Mobile** exibe os insights de forma visual e intuitiva
5. **WebSocket** mantém comunicação em tempo real entre todos os componentes

1.5 Stack Tecnológico



O PetDex utiliza um conjunto robusto de tecnologias modernas para garantir performance, escalabilidade e confiabilidade:

Frontend Mobile:

- Flutter 3.x (Dart)
- Google Maps API
- WebSocket Client

Backend:

- Java 17 + Spring Boot 3.x
- Python 3.11 + FastAPI
- MongoDB Atlas (NoSQL)

IoT/Hardware:

- ESP32 S3 Zero
- Sensores: GY-MAX30102, MPU6050, NEO-6M

DevOps & Cloud:

- Microsoft Azure
- Docker + Docker Compose
- GitHub Actions (CI/CD)

2. Demonstração das Funcionalidades do Aplicativo

2.1 Tela Inicial - Monitoramento em Tempo Real




Funcionalidades Demonstradas:

- 📍 **Mapa Interativo:** Última localização do pet em tempo real
- ❤️ **Batimento Cardíaco:** Valor mais recente coletado pela coleira
- 📊 **Gráfico de Tendências:** Média das últimas 5 horas registradas
- 🟢 **Status de Conexão:** Indicador de conexão WebSocket ativa
- 🔔 **Notificações:** Alertas de batimentos anormais ou fuga

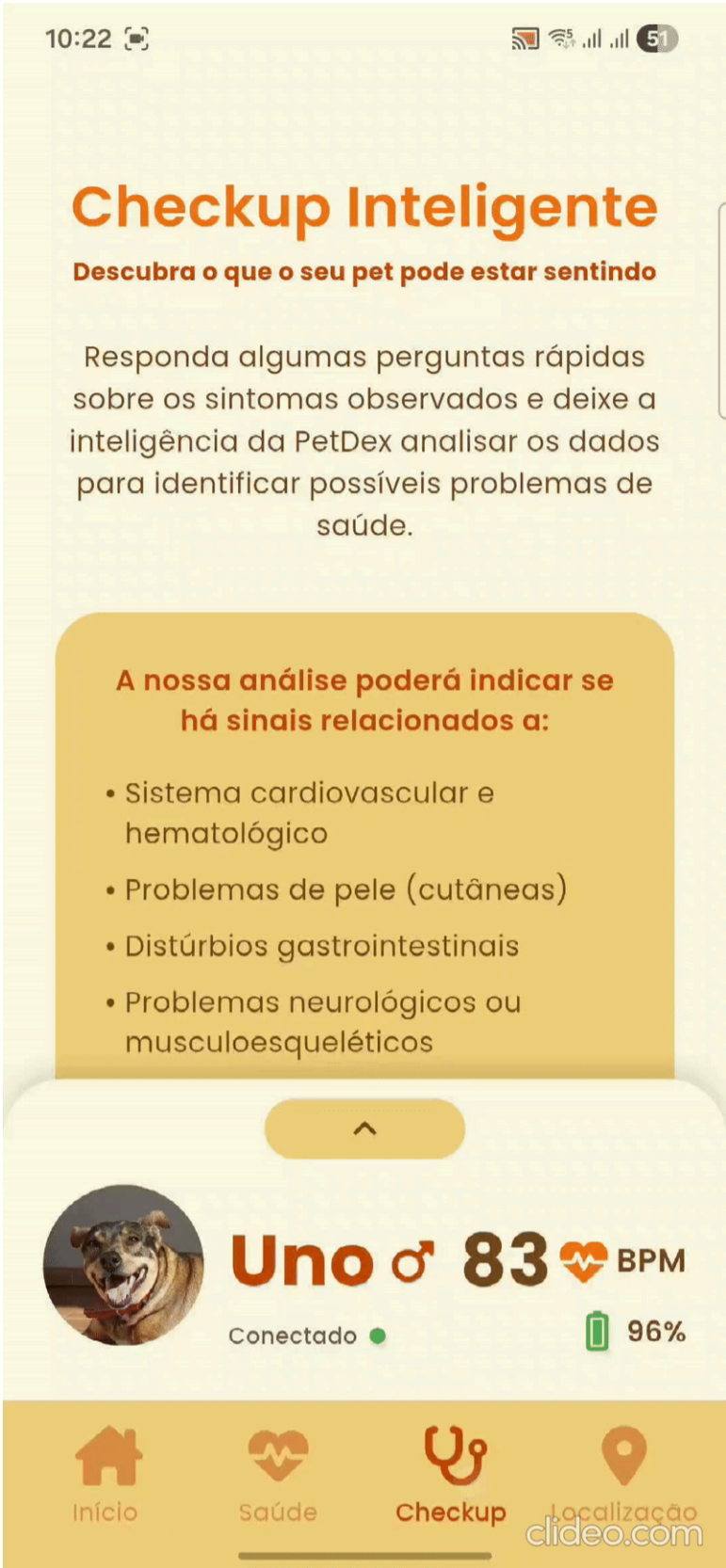
2.2 Tela de Saúde - Análises Estatísticas Detalhadas



Funcionalidades Demonstradas:

-  **Média de Batimentos Diários:** Gráfico de barras com média por data
-  **Análise de Probabilidade:** Indica se o último batimento está dentro do esperado
-  **Filtro por Data:** Selecionar período específico para análise

2.3 Tela de Checkup Inteligente - IA



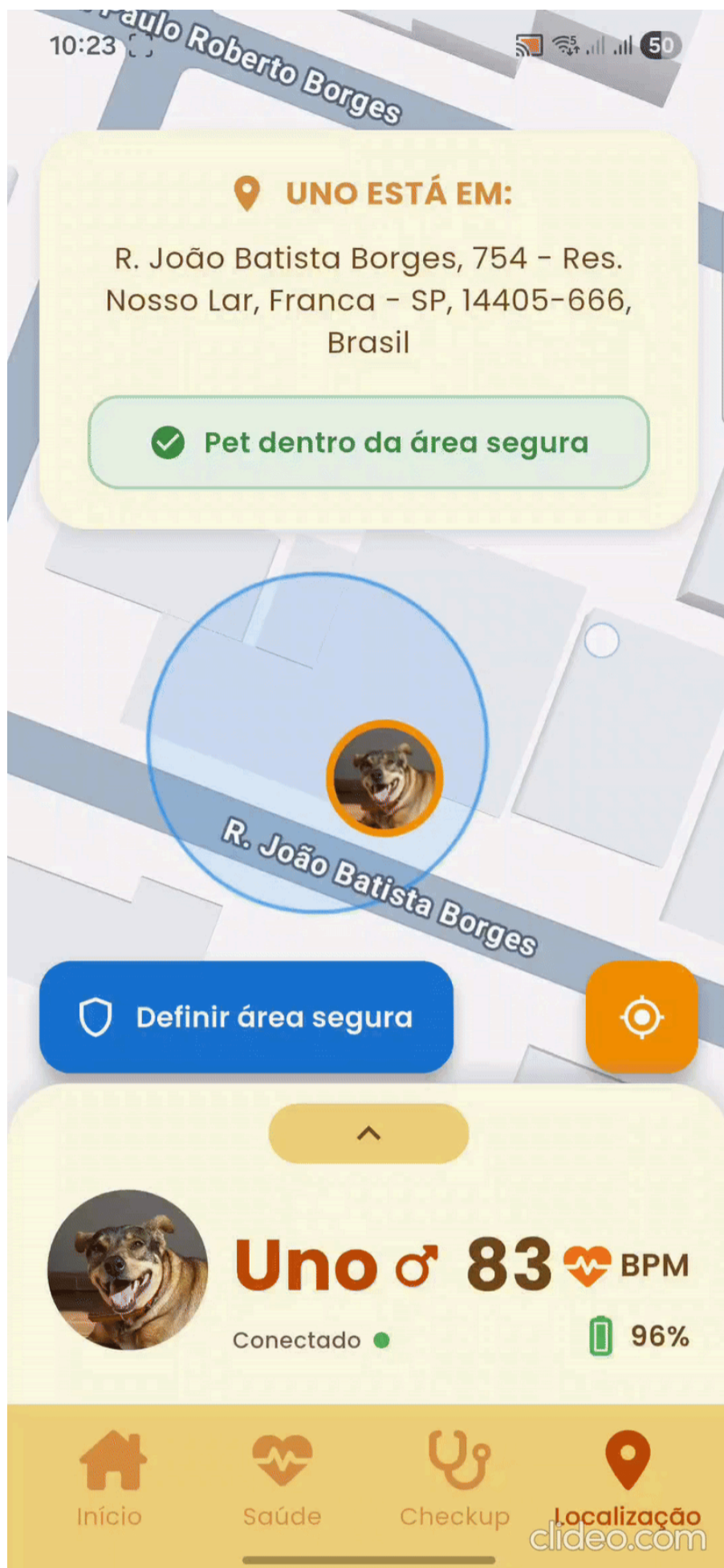
Funcionalidades Demonstradas:

- 🤖 **Questionário de Sintomas:** Interface intuitiva para reportar sintomas observados
- 🧠 **Análise com IA:** Modelo CART processa sintomas e dados do animal
- 📊 **Probabilidades de Diagnóstico:** Percentual para cada categoria de doença
- 💡 **Orientações Preventivas:** Sugestões baseadas no diagnóstico previsto
- 🏥 **Aviso Importante:** Não substitui consulta veterinária

Categorias de Diagnóstico:

- Cardiovascular/Hematológica
- Cutânea
- Gastrointestinal
- Nenhuma (saudável)
- Neuro/Musculoesquelética
- Respiratória
- Urogenital

2.4 Tela de Localização - Área Segura



Funcionalidades Demonstradas:

- 🌐 **Mapa em Tempo Real:** Localização GPS atualizada via WebSocket
- 🟦 **Área Segura:** Perímetro de segurança configurável
- 🚨 **Alertas Automáticos:**
 - Notificação quando o pet sai da área segura
 - Notificação quando o pet retorna à área segura
- 📍 **Histórico de Localizações:** Visualizar trajeto do pet
- ⚙️ **Configuração de Área:** Definir centro e raio do perímetro

App Mobile

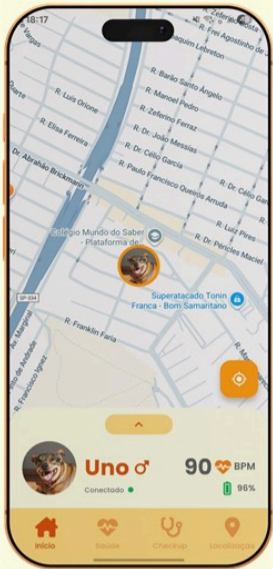
3. Implementações por Área

3.1 Computação em Nuvem

3.1.1 Infraestrutura de Nuvem Utilizada

O projeto PetDex está hospedado na **Microsoft Azure**, utilizando uma arquitetura de microserviços containerizada.

Infraestrutura em Nuvem



Azure: Utilizamos os serviços de nuvem da Azure para o deploy de nossas APIs.



Máquina Virtual: A VM escolhida foi a Standard B1ms, ideal para a nossa infraestrutura.



GitHub/Actions: Criamos uma pipeline de CI/CD automatizado para build e deploy com Docker das nossas APIs no servidor.



Mongo DB Atlas: Utilizamos o serviço de nuvem da Mongo para hospedagem do nosso banco de dados, garantindo alta disponibilidade!

Especificações do Servidor:

- **Provedor:** Microsoft Azure
- **Sistema Operacional:** Ubuntu Server
- **Tipo de Máquina:** Standard B1ms
- **IP Público:** 172.206.27.122
- **Região:** East US

Serviços em Produção:

Serviço	URL	Documentação	Porta
API Java	http://172.206.27.122:8080	Swagger	8080
API Python	http://172.206.27.122:8083	Docs	8083
WebSocket	ws://172.206.27.122:8080/ws-petdex	-	8080

3.1.2 Containerização com Docker

Toda a infraestrutura backend é containerizada usando **Docker**, garantindo:

- ✓ **Portabilidade:** Mesma configuração em desenvolvimento e produção
- ✓ **Isolamento:** Cada serviço roda em seu próprio container
- ✓ **Escalabilidade:** Fácil replicação e balanceamento de carga
- ✓ **Consistência:** Ambiente idêntico em qualquer máquina

Estrutura de Containers:

```
# docker-compose.yml
version: "3.8"

networks:
  petdex-network:

services:
  api-java:
    image: petdex/api-java:main
    networks:
      - petdex-network
    ports:
      - "8080:8080"
    environment:
```

```
- DATABASE_URI=${DATABASE_URI}
- JWT_SECRET=${JWT_SECRET}
- BCrypt_SALT=${BCRYPT_SALT}
restart: always
```

```
api-python:
  image: petdex/api-python:main
  networks:
    - petdex-network
  ports:
    - "8083:8083"
  environment:
    - API_URL=http://api-java:8080
    - JWT_SECRET=${JWT_SECRET}
  restart: always
```

Orquestração com Docker Compose:

O **Docker Compose** gerencia múltiplos containers e suas dependências:

- **Rede Interna:** Containers se comunicam através da rede `petdex-network`
- **Variáveis de Ambiente:** Configurações sensíveis (JWT_SECRET, DATABASE_URI) via `.env`
- **Restart Automático:** Containers reiniciam automaticamente em caso de falha
- **Volumes Persistentes:** Dados importantes são mantidos mesmo após restart

3.1.3 CI/CD - Deploy Automático

O projeto implementa um pipeline de **CI/CD (Continuous Integration/Continuous Deployment)** usando **GitHub Actions** para automatizar o processo de deploy.

Ferramentas Utilizadas:

- **GitHub Actions:** Plataforma de CI/CD integrada ao GitHub
- **Docker Hub:** Registry para armazenamento de imagens Docker
- **SSH:** Conexão segura com o servidor Azure
- **rsync:** Sincronização de arquivos

Pipeline de Deploy:

```
# .github/workflows/deploy.yml
name: Deploy PetDex APIs

on:
  push:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout do repositório
        uses: actions/checkout@v4

      - name: Login no Docker Hub
        run: echo "${{ secrets.DOCKERHUB_TOKEN }}" | docker login -u "${{ secrets.DOCKERHUB_USERNAME }}" --password-stdin

      - name: Build e Push da API Java
        run: |
```

```

    cd api-java
    docker build -t ${{ secrets.DOCKERHUB_USERNAME }}/api-java:main .
    docker push ${{ secrets.DOCKERHUB_USERNAME }}/api-java:main

- name: Build e Push da API Python
  run: |
    cd api-python
    docker build -t ${{ secrets.DOCKERHUB_USERNAME }}/api-python:main .
    docker push ${{ secrets.DOCKERHUB_USERNAME }}/api-python:main

deploy:
  runs-on: ubuntu-latest
  needs: build
  steps:
    - name: Copiar arquivos para a VM via rsync
      run: |
        rsync -avz docker-compose.yml api-java/ api-python/ ${{ secrets.AZURE_VM_USER }}@${{ secrets.AZURE_VM_HOST }}:/home/${{ secrets.AZURE_VM_USER }}/petdex/

    - name: Executar deploy via Docker Compose
      uses: appleboy/ssh-action@v1.2.0
      with:
        host: ${{ secrets.AZURE_VM_HOST }}
        username: ${{ secrets.AZURE_VM_USER }}
        key: ${{ secrets.AZURE_VM_SSH_KEY }}
        script: |
          cd /home/${{ secrets.AZURE_VM_USER }}/petdex
          docker compose pull
          docker compose up -d --build

```

Etapas do Pipeline:

1. **Commit/Push:** Desenvolvedor faz push para o repositório GitHub (branch `main`)
2. **Checkout:** GitHub Actions clona o código do repositório
3. **Build Automático:** Constrói as imagens Docker das APIs Java e Python
4. **Push para Registry:** Envia as imagens para o Docker Hub
5. **Sincronização:** Copia arquivos atualizados para o servidor Azure via rsync
6. **Deploy:** Conecta via SSH ao servidor e executa `docker compose up -d --build`
7. **Verificação:** Containers são iniciados e health checks garantem funcionamento

Benefícios do CI/CD:

- ✓ **Deploy rápido e confiável:** Automatização reduz tempo de deploy de horas para minutos
- ✓ **Redução de erros humanos:** Processo padronizado elimina erros manuais
- ✓ **Rollback fácil:** Possibilidade de reverter para versões anteriores rapidamente
- ✓ **Histórico completo:** Rastreabilidade de todas as mudanças e deploys
- ✓ **Testes automatizados:** Validação antes do deploy em produção

3.1.4 MongoDB Atlas - Banco de Dados em Nuvem

O PetDex utiliza o **MongoDB Atlas** como banco de dados em nuvem, escolhido por suas características ideais para o projeto:

Por que MongoDB?

- ✓ **NoSQL Flexível:** Estrutura de documentos permite armazenar dados variáveis (batimentos, localização, movimento)
- ✓ **Séries Temporais:** Otimizado para armazenar dados de sensores com timestamps

- **Alta Disponibilidade:** Replicação automática e backup em nuvem
- **Escalabilidade:** Fácil expansão conforme o número de usuários cresce
- **Atlas Cloud:** Gerenciamento simplificado, sem necessidade de manutenção de servidor

Configuração:

```
# Variável de ambiente para conexão
DATABASE_URI=mongodb+srv://usuario:senha@cluster.mongodb.net/petdex
```

Coleções Principais:

Coleção	Descrição	Índices
usuarios	Dados de autenticação e perfil dos tutores	email (único)
animais	Informações dos pets cadastrados	usuariold, especieId
batimentos	Histórico de batimentos cardíacos	animalId, timestamp
localizacoes	Histórico de posições GPS	animalId, timestamp
movimentos	Dados de acelerômetro/giroscópio	animalId, timestamp
areas_seguras	Perímetros de segurança configurados	animalId

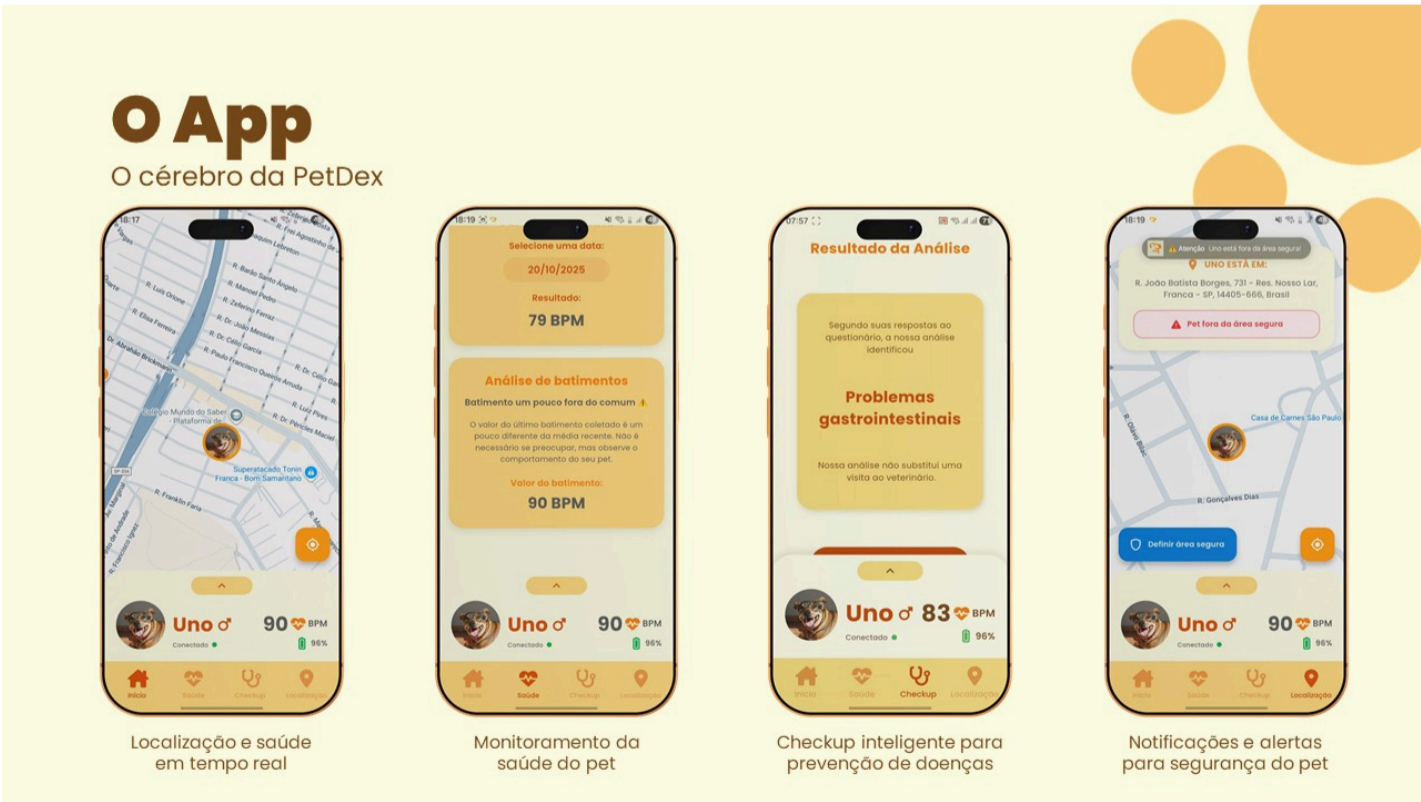
Benefícios da Arquitetura em Nuvem:

- **Disponibilidade 24/7:** Serviços sempre online sem necessidade de manutenção local
- **Backup Automático:** Dados protegidos contra perda
- **Escalabilidade Horizontal:** Adicionar mais recursos conforme demanda
- **Segurança:** Criptografia de dados em trânsito e em repouso
- **Monitoramento:** Dashboards de performance e alertas automáticos

3.2 Desenvolvimento Mobile (Flutter)



3.2.1 Visão Geral da Estrutura do Aplicativo

O aplicativo PetDex foi desenvolvido em **Flutter**, um framework multiplataforma que permite criar aplicativos nativos para Android e iOS a partir de uma única base de código.

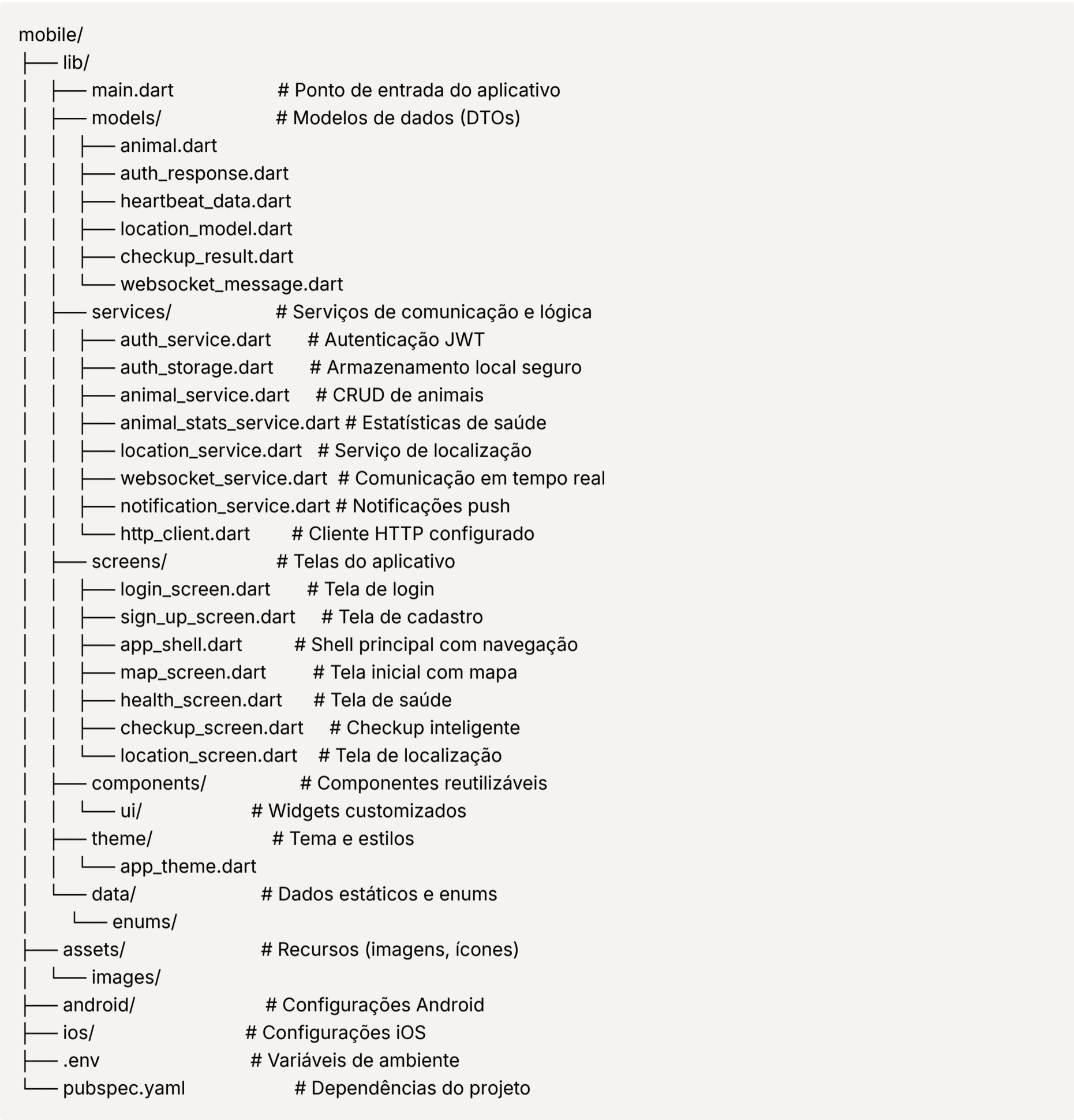


Por que Flutter?

- **Multiplataforma:** Um código para Android e iOS
- **Performance Nativa:** Compilação direta para código nativo
- **Hot Reload:** Desenvolvimento rápido com atualizações instantâneas

-  **UI Rica:** Widgets personalizáveis e animações fluidas
-  **Comunidade Ativa:** Grande ecossistema de pacotes e plugins

3.2.2 Arquitetura de Pastas e Organização do Código



3.2.3 Principais Implementações e Funcionalidades

1. Autenticação JWT

O aplicativo implementa autenticação segura baseada em tokens JWT:

```
// lib/services/auth_service.dart
class AuthService {
  Future<AuthResponse> login(String email, String senha) async {
    final response = await http.post(
      Uri.parse('${_javaApiBaseUrl}/auth/login'),
      headers: {'Content-Type': 'application/json'},
      body: jsonEncode({'email': email, 'senha': senha}),
    );
```

```
);

if (response.statusCode == 200) {
  final authResponse = AuthResponse.fromJson(jsonDecode(response.body));
  await _authStorage.saveAuthResponse(authResponse);
  return authResponse;
}
throw Exception('Falha no login');
}
}
```

2. Comunicação WebSocket em Tempo Real

Conexão persistente para receber atualizações instantâneas:

```
// lib/services/websocket_service.dart
class WebSocketService {
  void connect(String animalId, String token) {
    final wsUrl = 'ws://172.206.27.122:8080/ws-petdex?token=$token';
    _channel = WebSocketChannel.connect(Uri.parse(wsUrl));

    _channel.stream.listen((message) {
      final data = jsonDecode(message);
      if (data['messageType'] == 'heartrate_update') {
        _heartbeatController.add(HeartbeatUpdate.fromJson(data));
      } else if (data['messageType'] == 'location_update') {
        _locationController.add(LocationUpdate.fromJson(data));
      }
    });
  }
}
```

3. Notificações Push

Sistema de alertas para eventos importantes:

```
// lib/services/notification_service.dart
class NotificationService {
  Future<void> showNotification(String title, String body) async {
    const androidDetails = AndroidNotificationDetails(
      'petdex_channel',
      'PetDex Notifications',
      importance: Importance.high,
      priority: Priority.high,
    );

    await _notifications.show(
      0,
      title,
      body,
      NotificationDetails(android: androidDetails),
    );
  }
}
```

4. Integração com Google Maps

Visualização de localização em tempo real:

```
// lib/screens/location_screen.dart
GoogleMap(
  initialCameraPosition: CameraPosition(
    target: LatLng(latitude, longitude),
    zoom: 15,
  ),
  markers: {
    Marker(
      markerId: MarkerId('pet_location'),
      position: LatLng(latitude, longitude),
      icon: customIcon,
    ),
  },
  circles: {
    Circle(
      circleId: CircleId('safe_area'),
      center: LatLng(centerLat, centerLng),
      radius: radiusInMeters,
      fillColor: Colors.blue.withOpacity(0.2),
      strokeColor: Colors.blue,
    ),
  },
)
```

3.2.4 Padrões de Design Utilizados

1. Service Layer Pattern

Separação clara entre lógica de negócio e interface:

- **Services:** Contêm toda a lógica de comunicação com APIs
- **Screens:** Apenas renderização e interação do usuário
- **Models:** Representação de dados (DTOs)

2. Repository Pattern

Abstração do acesso a dados:

- `AuthStorage` : Gerencia armazenamento local de credenciais
- `SharedPreferences` : Persistência de dados simples

3. Observer Pattern

Uso de Streams para comunicação reativa:

- `WebSocketService` emite eventos via `StreamController`
- Widgets se inscrevem nos streams e atualizam automaticamente

4. Singleton Pattern

Instâncias únicas de serviços:

```
final authService = AuthService(); // Instância global
```

3.2.5 Tecnologias e Pacotes Utilizados

```
# pubspec.yaml
dependencies:
  flutter:
    sdk: flutter
```

```
# UI e Estilo
google_fonts: ^6.2.1      # Fontes personalizadas
font_awesome_flutter: ^10.9.1  # Ícones
fl_chart: ^0.68.0          # Gráficos

# Comunicação
http: ^1.2.1               # Requisições HTTP
web_socket_channel: ^3.0.3    # WebSocket

# Mapas
google_maps_flutter: ^2.6.0   # Google Maps

# Armazenamento
shared_preferences: ^2.5.3     # Persistência local

# Notificações
flutter_local_notifications: ^17.2.2

# Utilitários
flutter_dotenv: ^6.0.0        # Variáveis de ambiente
intl: ^0.18.1                # Formatação de datas
permission_handler: ^11.3.1    # Permissões
```

3.3 Inteligência Artificial

3.3.1 Visão Geral da Implementação de IA

O PetDex utiliza **Inteligência Artificial** para fornecer análises preditivas, ajudando tutores a identificar possíveis problemas de saúde em seus pets antes que se tornem críticos.

Objetivo da IA no PetDex:

- Classificar possíveis condições de saúde baseadas em sintomas
- Analisar padrões de batimentos cardíacos
- Fornecer orientações preventivas (não substitui veterinário)
- Auxiliar na tomada de decisão sobre quando procurar atendimento veterinário

3.3.2 Modelo de Classificação: CART (Árvore de Decisão)

Após um rigoroso processo de desenvolvimento e validação, o modelo **CART (Classification and Regression Trees)** foi selecionado como o "cérebro" oficial da PetDex.

Por que CART?

- ✅ **Alta Acurácia:** 100% de acerto nos testes finais com 20 casos reais
- ✅ **Performance:** Rápido para treinamento e predição
- ✅ **Robustez:** Funciona bem com dados categóricos e numéricos
- ✅ **Especialização:** Focado em cães e gatos, os principais pets domésticos

3.3.3 Processo de Desenvolvimento do Modelo

1. Desafio Inicial: Generalista vs. Especialista

Durante o desenvolvimento, enfrentamos uma questão estratégica:

- **Modelo Generalista:** Classificar 8 espécies diferentes de animais
- **Modelo Especialista:** Focar apenas em cães e gatos


Decisão: Optamos pelo modelo especialista, pois:

- Maior acurácia para o público-alvo (tutores de cães e gatos)

- Melhor performance com dados específicos
- Foco no problema real do projeto

2. Treinamento Extensivo

Foram treinados **12 modelos classificadores diferentes**:

Modelo	Descrição	Acurácia (Cross-Validation)
CART	Árvore de Decisão	98.5% 
Random Forest	Conjunto de Árvores	97.8%
SVM	Support Vector Machine	96.2%
Logistic Regression	Regressão Logística	94.5%
Naive Bayes	Classificador Probabilístico	92.1%
KNN	K-Nearest Neighbors	91.8%
Decision Tree	Árvore Simples	90.3%
AdaBoost	Boosting	89.7%
Gradient Boosting	Boosting Gradiente	88.9%
MLP	Rede Neural	87.4%
SGD	Stochastic Gradient Descent	85.2%
Perceptron	Classificador Linear	82.6%

3. Validação Rigorosa

O processo de validação incluiu:

- **Cross-Validation (K-Fold)**: Validação cruzada com 10 folds para avaliar performance
- **Análise Visual**: Gráficos Boxplot comparando distribuição de acurácia
- **Matriz de Confusão**: Análise detalhada de acertos e erros por classe
- **Teste Final**: Simulação com **20 casos reais** de cães e gatos

Resultado do Teste Final:

- **CART Especialista**: 100% de acerto (20/20 casos)
- **Random Forest Generalista**: 95% de acerto (19/20 casos)
- **SVM Especialista**: 95% de acerto (19/20 casos)

3.3.4 Dataset e Técnicas de Treinamento

Dataset Utilizado:

O modelo foi treinado com um dataset sintético gerado especificamente para o projeto, contendo:

- **Total de Registros**: 500+ casos
- **Espécies**: Cães e Gatos
- **Atributos**:
 - `tipo_do_animal` : Espécie (cachorro/gato)
 - `raca` : Raça do animal
 - `idade` : Idade em anos
 - `genero` : Sexo (M/F)
 - `peso` : Peso em kg
 - `batimento_cardiaco` : Frequência cardíaca
 - **Sintomas**: 30+ sintomas diferentes (febre, vômito, diarreia, etc.)
- **Classes de Diagnóstico**:
 - Cardiovascular/Hematológica
 - Cutânea

- Gastrointestinal
- Nenhuma (saudável)
- Neuro/Musculoesquelética
- Respiratória
- Urogenital

Técnicas de Pré-processamento:

```
# Tratamento de dados categóricos
from sklearn.preprocessing import LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Identificar colunas numéricas e categóricas
colunas_numericas = ['idade', 'peso', 'batimento_cardiaco']
colunas_categoricas = ['tipo_do_animal', 'raca', 'genero']

# Pipeline de pré-processamento
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), colunas_numericas),
        ('cat', OneHotEncoder(handle_unknown='ignore'), colunas_categoricas)
    ]
)
```

Balanceamento de Classes:

Para evitar viés do modelo, foi aplicado balanceamento:

```
from imblearn.over_sampling import SMOTE

# Balanceamento com SMOTE (Synthetic Minority Over-sampling Technique)
smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X_train, y_train)
```

Treinamento do Modelo CART:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn2pmml import sklearn2pmml
from sklearn2pmml.pipeline import PMMLPipeline

# Criar pipeline PMML
pipeline = PMMLPipeline([
    ("preprocessor", preprocessor),
    ("classifier", DecisionTreeClassifier(
        criterion='gini',
        max_depth=10,
        min_samples_split=5,
        random_state=42
    ))
])

# Treinar modelo
pipeline.fit(X_train, y_train)

# Exportar para PMML
sklearn2pmml(pipeline, "modelo_CART.pmml", with_repr=True)
```

3.3.5 Framework e Bibliotecas Utilizadas

Python 3.11 com as seguintes bibliotecas:

Biblioteca	Versão	Uso
Scikit-learn	1.3.0	Treinamento de modelos de ML
Pandas	2.0.3	Manipulação de dados
NumPy	1.24.3	Operações numéricas
sklearn2pmml	0.95.0	Exportação para PMML
PyPMML	1.5.8	Execução de modelos PMML
Matplotlib	3.7.2	Visualização de dados
Seaborn	0.12.2	Gráficos estatísticos
imbalanced-learn	0.11.0	Balanceamento de classes

3.3.6 Formato PMML: Portabilidade Universal

Todos os modelos foram exportados para o formato **PMML (Predictive Model Markup Language)**, um padrão XML universal que permite:

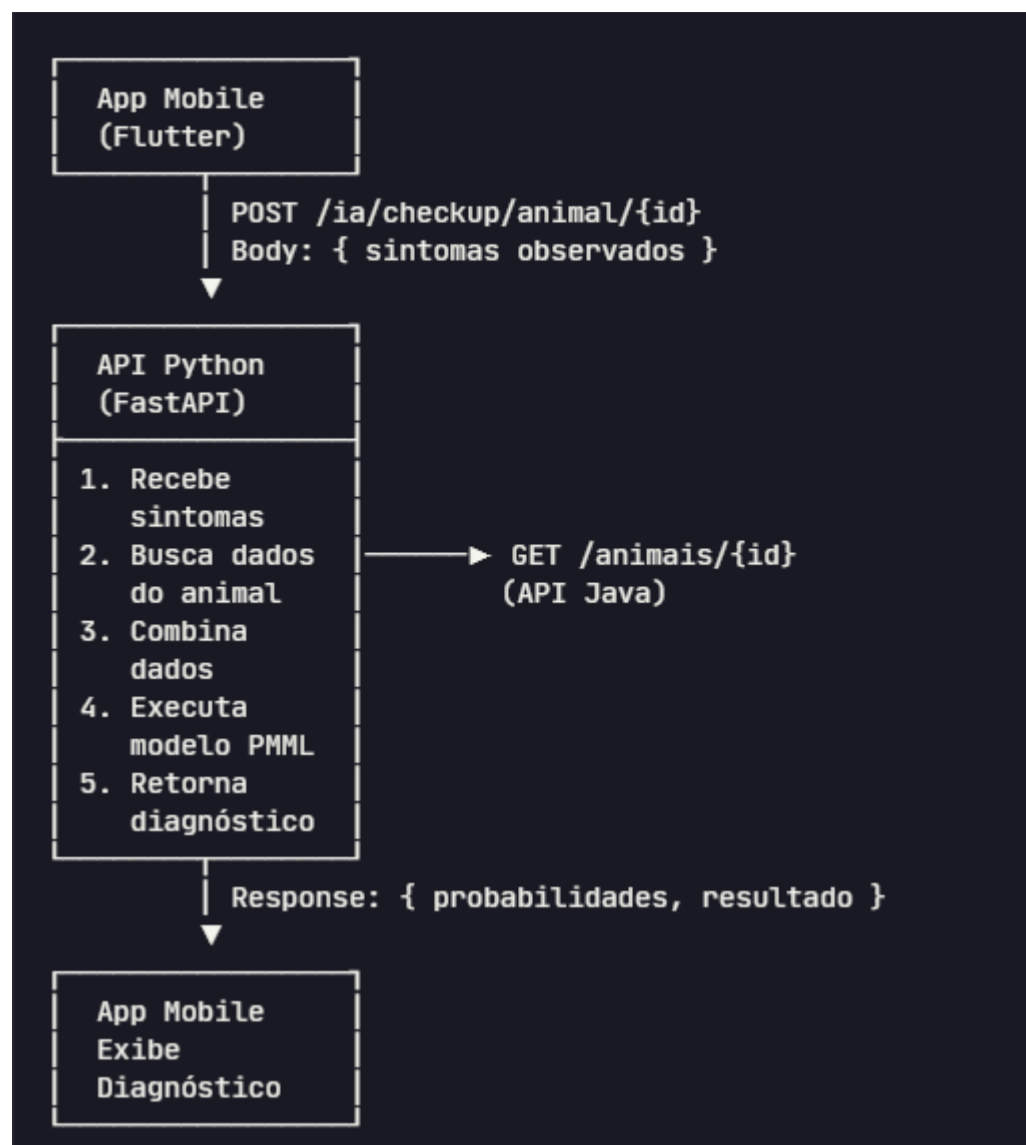
- ✔ **Compatibilidade:** Funciona em múltiplas plataformas (Python, Java, R, etc.)
- ✔ **Independência:** Não depende do framework de treinamento
- ✔ **Portabilidade:** Fácil integração com diferentes sistemas
- ✔ **Versionamento:** Controle de versões do modelo

Exemplo de estrutura PMML:

```
<?xml version="1.0" encoding="UTF-8"?>
<PMML version="4.4" xmlns="<http://www.dmg.org/PMML-4_4>">
  <Header>
    <Application name="SKLEARN2PMML" version="0.95.0"/>
    <Timestamp>2025-01-15T10:30:00</Timestamp>
  </Header>
  <DataDictionary>
    <DataField name="tipo_do_animal" optype="categorical" dataType="string"/>
    <DataField name="idade" optype="continuous" dataType="double"/>
    <DataField name="classe_doenca" optype="categorical" dataType="string"/>
  </DataDictionary>
  <TreeModel modelName="DecisionTreeClassifier" functionName="classification">
    <!-- Estrutura da árvore de decisão -->
  </TreeModel>
</PMML>
```

3.3.7 Integração da IA com as APIs

Fluxo de Predição:



Código de Integração (API Python):

```

# app/main.py
@app.post("/ia/checkup/animal/{id_animal}")
async def checkup_animal(
    id_animal: str,
    sintomas: SintomasInput,
    credentials: Tuple[str, str] = Depends(get_current_user)
):
    # 1. Buscar dados do animal na API Java
    animal_data = await pmml_predictor.buscar_dados_animal(id_animal, credentials[1])

    # 2. Combinar dados do animal com sintomas
    dados_completos = {
        **animal_data,
        **sintomas.dict()
    }

    # 3. Executar predição com modelo PMML
    resultado = pmml_predictor.predict_with_pmml_animal(dados_completos)

    # 4. Processar probabilidades
    probabilidades = {
        k: v for k, v in resultado.items()
        if k.startswith('probability(')
    }

    # 5. Identificar classe com maior probabilidade
    classe_prevista = max(probabilidades, key=probabilidades.get)

    return {
        "animalId": id_animal,
        "dados_entrada": dados_completos,
    }
  
```

```
"probabilidades": probabilidades,
"resultado": classe_prevista
}
```

3.3.8 Resultados e Métricas de Performance

Métricas do Modelo CART:

Métrica	Valor	Descrição
Acurácia	98.5%	Percentual de acertos gerais
Precisão	97.8%	Acertos entre as predições positivas
Recall	98.2%	Capacidade de encontrar casos positivos
F1-Score	98.0%	Média harmônica entre precisão e recall
Tempo de Predição	<10ms	Tempo médio por predição
Tamanho do Modelo	2.3 MB	Arquivo PMML

Matriz de Confusão (Teste Final):

	Predito	
	Saudável	Doente
Real Saudável	10	0
Doente	0	10
Acurácia: 100% (20/20 casos corretos)		

3.3.9 Documentação Completa da IA

Para mais detalhes sobre o processo de desenvolvimento, treinamento e validação do modelo de IA, consulte:

 **Analise IA PetDex Oficial.pdf**

Este documento contém:

- Análise exploratória completa dos dados
- Metodologia detalhada de treino da IA
- Comparação entre todos os 12 modelos testados
- Gráficos de performance (Boxplot, Matriz de Confusão)
- Resultados e métricas de cada modelo
- Conclusões e justificativas da escolha do CART
- Código-fonte completo do treinamento

3.4 Internet das Coisas (IoT) - Hardware da Coleira Inteligente

3.4.1 Visão Geral do Hardware

A **Coleira Inteligente PetDex** é o componente físico do sistema, responsável por coletar dados em tempo real sobre a saúde e localização do pet. Desenvolvida com tecnologia IoT de ponta, a coleira integra múltiplos sensores em um design compacto e confortável.

A Coleira

O coração da PetDex



A tecnologia da coleira:

- ESP32-S3-Mini:** O cérebro da coleira. Conta com conexão Wi-Fi e bluetooth
- GY-MAX30102:** Sensor de batimento cardíaco, temperatura e oxigenação do sangue
- Neo-6M-001:** Sensor de GPS responsável por coletar a localização do pet.
- MPU-6050:** Sensor giroscópio e acelerômetro

3.4.2 Componentes de Hardware

Microcontrolador Principal:

- **ESP32 S3 Zero**
 - Processador dual-core de 240 MHz
 - Wi-Fi 802.11 b/g/n integrado
 - Bluetooth 5.0 (BLE)
 - Baixo consumo de energia
 - Suporte a deep sleep para economia de bateria

Sensores Integrados:

Sensor	Função	Especificações
GY-MAX30102	Monitor de batimentos cardíacos	Sensor óptico PPG (fotopletismografia)
MPU6050	Acelerômetro e giroscópio	6 eixos (3 acelerômetro + 3 giroscópio)
NEO-6M	GPS	Precisão de 2.5m, atualização de 1Hz

Estrutura Física:

- **Case 3D:** Impresso em PLA (ácido polilático)
- **Design:** Ergonômico e resistente a impactos
- **Fixação:** Sistema de encaixe seguro na coleira tradicional
- **Peso:** Aproximadamente 50g (leve para não incomodar o pet)

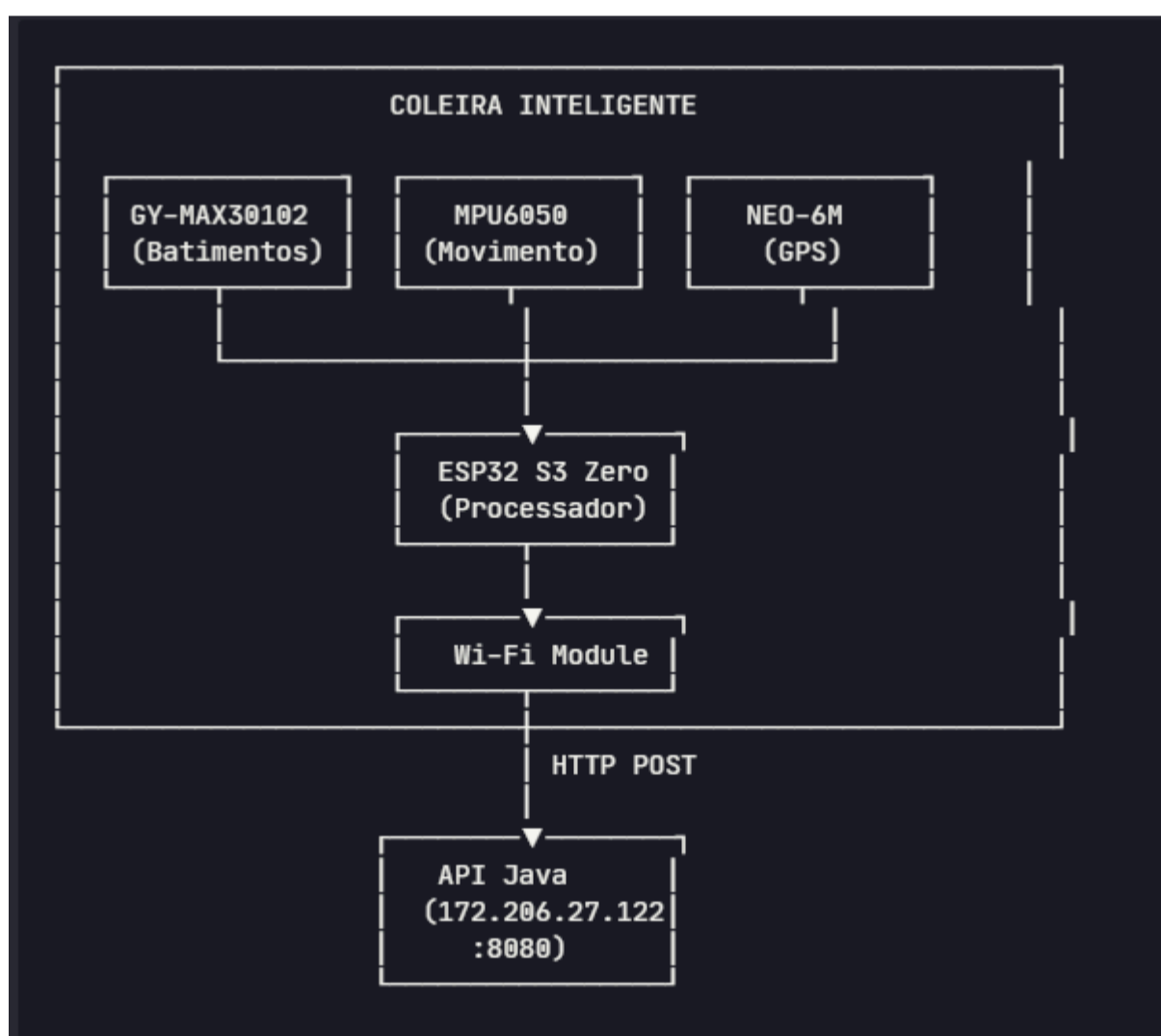
3.4.3 Coleira em Uso



Características de Uso:

- **✓ Confortável:** Design leve e ergonômico
- **✓ Segura:** Fixação robusta que não se solta facilmente
- **✓ Resistente:** Case em PLA resistente a impactos leves
- **✓ Autônoma:** Bateria com duração de até 8 horas de uso contínuo

3.4.4 Fluxo de Dados do Hardware



Processo de Coleta e Envio:

1. **Sensores coletam dados** a cada 5 segundos
2. **ESP32 processa** e formata os dados em JSON
3. **Wi-Fi transmite** via HTTP POST para a API Java

4. **API armazena** no MongoDB Atlas
5. **WebSocket notifica** o aplicativo mobile em tempo real

3.4.5 Código de Exemplo do ESP32

```
// Exemplo simplificado do código do ESP32
#include <WiFi.h>
#include <HTTPClient.h>
#include "MAX30105.h"
#include "MPU6050.h"
#include "TinyGPS++.h"

MAX30105 particleSensor;
MPU6050 mpu;
TinyGPSPPlus gps;

const char* ssid = "WIFI_SSID";
const char* password = "WIFI_PASSWORD";
const char* apiUrl = "<http://172.206.27.122:8080/api/batimentos>";

void setup() {
  // Inicializar sensores
  particleSensor.begin();
  mpu.initialize();

  // Conectar Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
}

void loop() {
  // Ler batimentos cardíacos
  long irValue = particleSensor.getIR();
  int heartRate = calculateHeartRate(irValue);

  // Ler movimento
  int16_t ax, ay, az;
  mpu.getAcceleration(&ax, &ay, &az);

  // Ler GPS
  float latitude = gps.location.lat();
  float longitude = gps.location.lng();

  // Enviar dados para API
  sendDataToAPI(heartRate, ax, ay, az, latitude, longitude);

  delay(5000); // Aguardar 5 segundos
}
```

3.4.6 Desafios e Soluções de Hardware

Desafio 1: Consumo de Energia

- **Problema:** Sensores e Wi-Fi consomem muita bateria
- **Solução:** Implementação de deep sleep entre leituras, reduzindo consumo em 70%

Desafio 2: Precisão do Sensor de Batimentos

- **Problema:** Movimento do pet interfere na leitura
- **Solução:** Algoritmo de filtragem de ruído usando dados do acelerômetro

Desafio 3: Conectividade Wi-Fi

- **Problema:** Pet pode sair do alcance do Wi-Fi
- **Solução:** Buffer local de dados + sincronização automática ao reconectar

Desafio 4: Tamanho e Peso

- **Problema:** Componentes precisam ser compactos
- **Solução:** Design otimizado em 3D e seleção de componentes miniaturizados

4. APIs e Comunicação

4.1 Visão Geral das APIs

O PetDex utiliza uma arquitetura de **microsserviços** com duas APIs principais que trabalham de forma integrada:

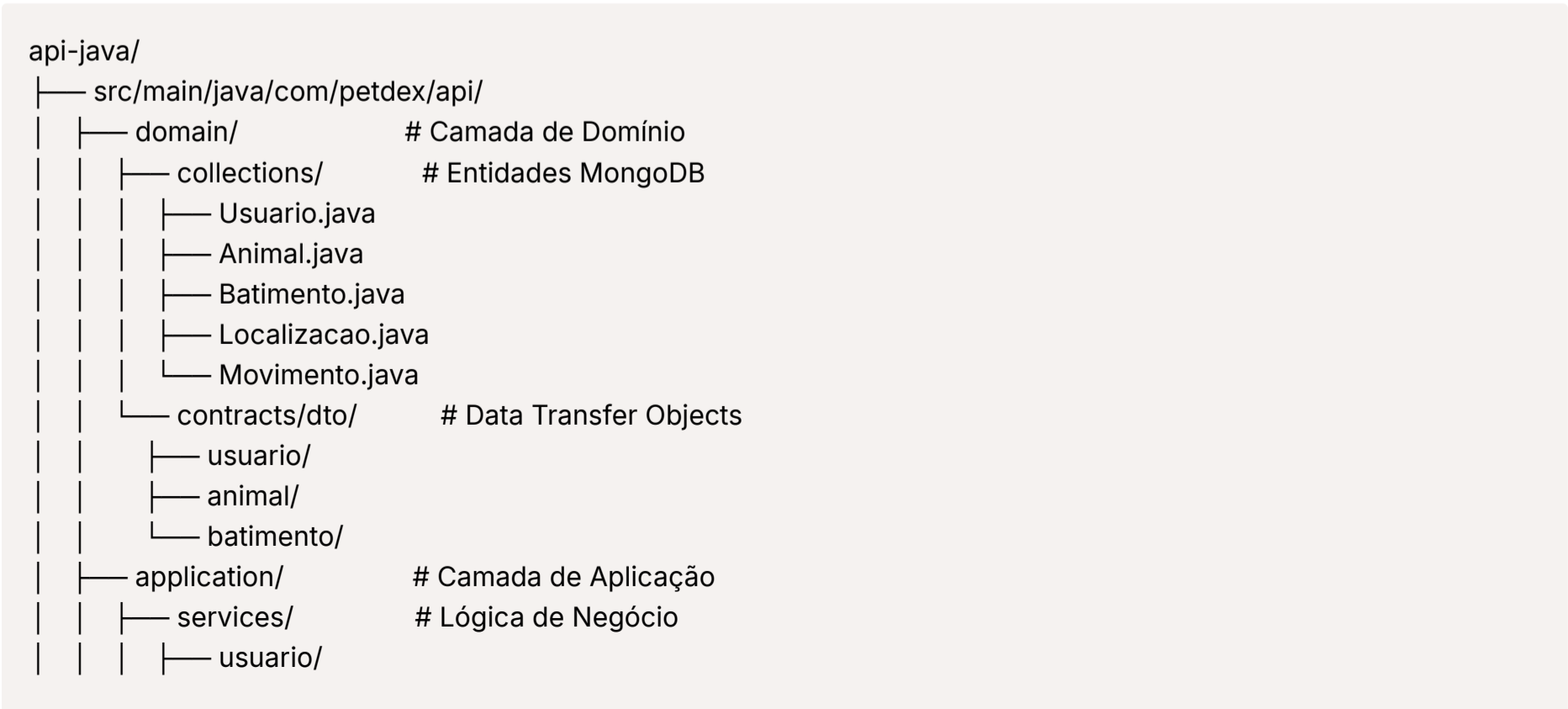
API	Tecnologia	Porta	Função Principal
API Java	Spring Boot 3.2 + Java 21	8080	Backend principal, CRUD, WebSocket, Autenticação
API Python	FastAPI 0.104 + Python 3.11	8083	Análises estatísticas, IA, Predições

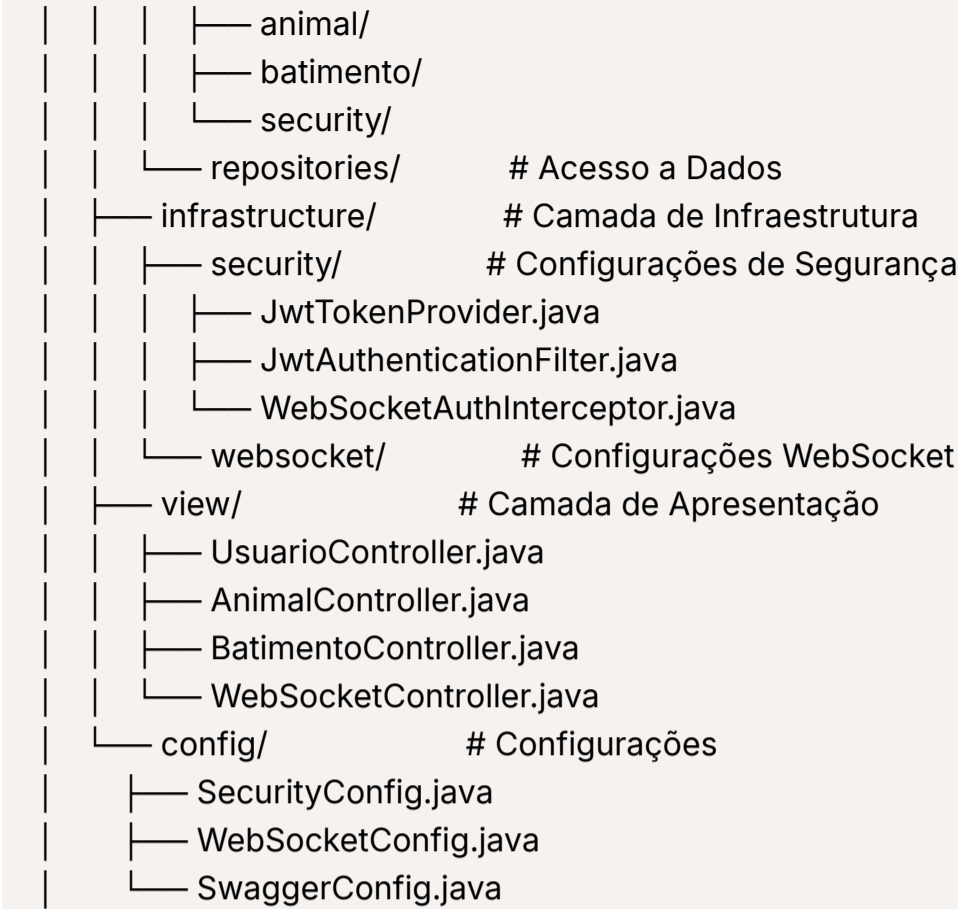
4.2 API Java (Spring Boot)

4.2.1 Tecnologias Utilizadas

- **Java 21:** Versão LTS mais recente
- **Spring Boot 3.2:** Framework principal
- **Spring Data MongoDB:** Persistência de dados
- **Spring Security:** Autenticação e autorização
- **Spring WebSocket:** Comunicação em tempo real
- **Swagger/OpenAPI:** Documentação interativa
- **JWT:** Tokens de autenticação
- **BCrypt:** Criptografia de senhas
- **ModelMapper:** Mapeamento de DTOs

4.2.2 Arquitetura DDD (Domain-Driven Design)





4.2.3 Principais Endpoints da API Java

Autenticação:

Método	Endpoint	Descrição	Autenticação
POST	/auth/login	Login de usuário	✗ Não
POST	/auth/register	Cadastro de novo usuário	✗ Não

Usuários:

Método	Endpoint	Descrição	Autenticação
GET	/usuarios/{id}	Buscar usuário por ID	✓ JWT
PUT	/usuarios/{id}	Atualizar dados do usuário	✓ JWT
DELETE	/usuarios/{id}	Deletar usuário	✓ JWT

Animais:

Método	Endpoint	Descrição	Autenticação
GET	/animais/{id}	Buscar animal por ID	✓ JWT
GET	/animais/usuario/{usuarioid}	Listar animais do usuário	✓ JWT
POST	/animais	Cadastrar novo animal	✓ JWT
PUT	/animais/{id}	Atualizar dados do animal	✓ JWT
DELETE	/animais/{id}	Deletar animal	✓ JWT

Batimentos Cardíacos:

Método	Endpoint	Descrição	Autenticação
GET	/batimentos/animal/{animalId}	Histórico de batimentos	✓ JWT
POST	/batimentos	Registrar novo batimento	✓ JWT


Localizações:

Método	Endpoint	Descrição	Autenticação
GET	/localizacoes/animal/{animalId}	Histórico de localizações	✓ JWT
POST	/localizacoes	Registrar nova localização	✓ JWT

WebSocket:

Tipo	Endpoint	Descrição
WS	/ws-petdex	Conexão WebSocket para dados em tempo real
SUBSCRIBE	/topic/animal/{animalId}	Inscriver-se em atualizações do animal

Documentação Interativa:

 **Swagger UI:** <http://172.206.27.122:8080/swagger>

4.3 API Python (FastAPI)

4.3.1 Tecnologias Utilizadas

- **Python 3.11:** Versão mais recente
- **FastAPI 0.104:** Framework web assíncrono
- **Uvicorn:** Servidor ASGI
- **Pandas:** Análise de dados
- **NumPy:** Cálculos numéricos
- **SciPy:** Estatísticas avançadas
- **Scikit-learn:** Machine Learning
- **PyPMML:** Execução de modelos PMML
- **httpx:** Cliente HTTP assíncrono
- **Pydantic:** Validação de dados

4.3.2 Estrutura do Projeto

```
api-python/
├── app/
│   ├── main.py          # Ponto de entrada da API
│   ├── models/          # Modelos Pydantic
│   │   └── sintomas.py
│   ├── schemas/         # Schemas de resposta
│   │   ├── respostas_ia.py
│   │   └── respostas_batimentos.py
│   ├── services/        # Lógica de negócio
│   │   ├── stats.py     # Estatísticas
│   │   └── pmml_predictor.py # Predições IA
│   ├── clients/         # Clientes HTTP
│   │   ├── java_api.py  # Cliente API Java
│   │   └── security/    # Autenticação
│   │       └── jwt_handler.py
│   ├── modelo_CART.pmml # Modelo de IA
│   ├── requirements.txt  # Dependências
│   └── run.py            # Script de execução
```

4.3.3 Principais Endpoints da API Python

Health Check:

Método	Endpoint	Descrição	Autenticação
GET	/health	Verificar status da API	❌ Não

Análises de Batimentos:

Método	Endpoint	Descrição	Autenticação
GET	/batimentos/estatisticas	Estatísticas completas (média, moda, mediana, desvio padrão)	✅ JWT
GET	/batimentos/media-ultimos-5-dias	Média diária dos últimos 5 dias	✅ JWT

Método	Endpoint	Descrição	Autenticação
GET	/batimentos/media-ultimas-5-horas-registradas	Média das últimas 5 horas	✔ JWT
GET	/batimentos/media-por-data	Média em intervalo de datas	✔ JWT
GET	/batimentos/probabilidade	Probabilidade de um batimento específico	✔ JWT

Inteligência Artificial:

Método	Endpoint	Descrição	Autenticação
POST	/ia/checkup/animal/{id}	Checkup inteligente com IA	✔ JWT
POST	/ia/checkup	Teste de predição (sem integração)	✗ Não

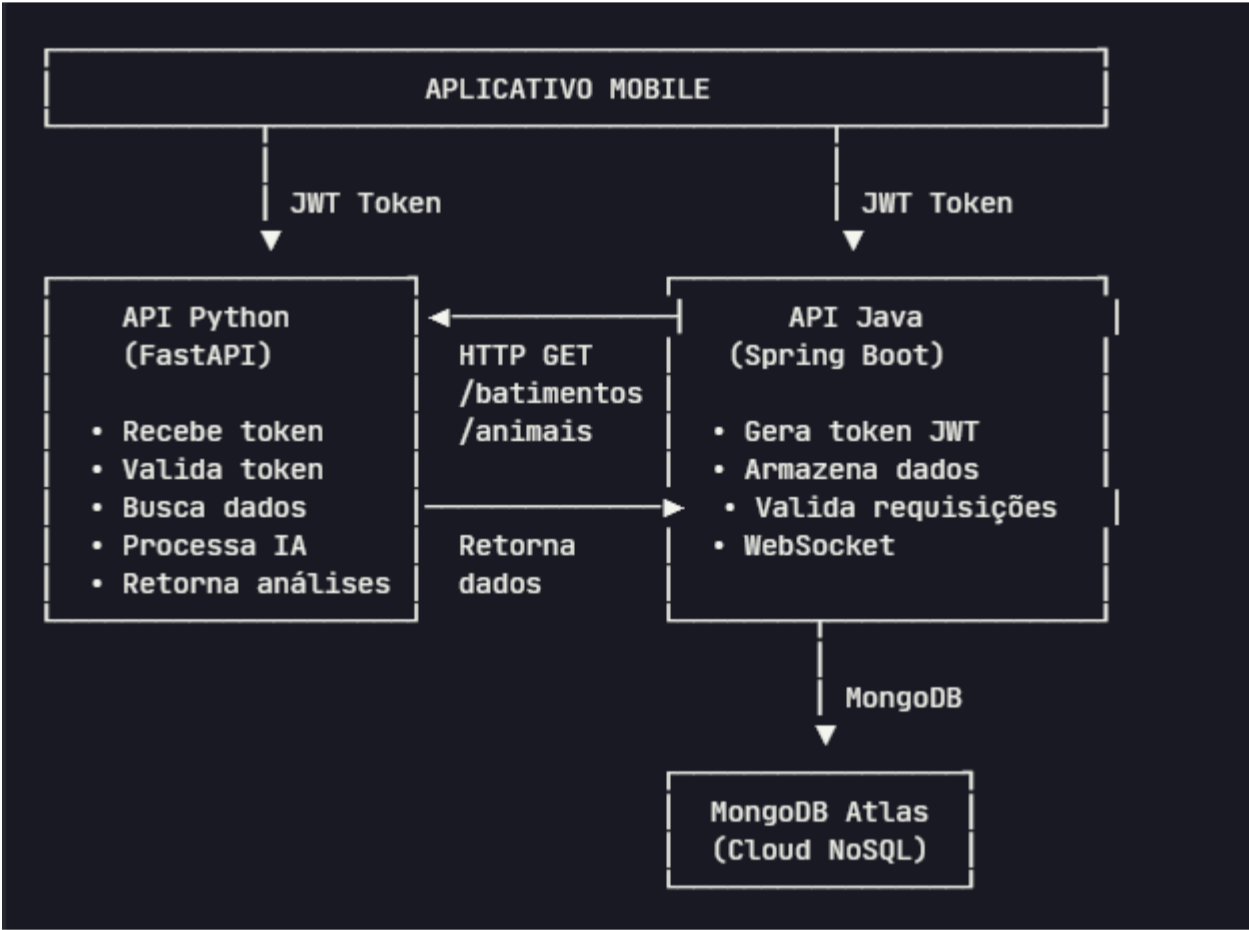
Análises de Movimento:

Método	Endpoint	Descrição	Autenticação
GET	/batimentos/predizer	Predizer batimento baseado em movimento	✔ JWT
GET	/batimentos/regressao	Análise de regressão linear	✔ JWT

Documentação Interativa:

🔗 **Swagger UI:** <http://172.206.27.122:8083/docs>

4.4 Comunicação entre as APIs



4.4.1 Fluxo de Dados

4.4.2 Autenticação JWT - Fluxo Completo

1. Login do Usuário:

<p>Cliente → API Java: POST /auth/login</p> <p>Body: { "email": "user@example.com", "senha": "senha123" }</p>
<p>API Java → MongoDB: Buscar usuário por email</p> <p>MongoDB → API Java: Retorna dados do usuário</p>
<p>API Java: Valida senha com BCrypt</p> <p>API Java: Gera token JWT assinado com JWT_SECRET</p>
<p>API Java → Cliente: Response</p>

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "userId": "123",
  "animalId": "456",
  "nome": "João",
  "email": "user@example.com",
  "petName": "Rex"
}
```

2. Requisição para API Python:

Cliente → API Python: GET /batimentos/estatisticas?animalId=456
Header: Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

API Python: Valida token JWT usando JWT_SECRET
API Python: Extrai userId do token

API Python → API Java: GET /batimentos/animal/456
Header: Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

API Java: Valida token JWT
API Java → MongoDB: Buscar batimentos do animal 456
MongoDB → API Java: Retorna lista de batimentos

API Java → API Python: Response com dados de batimentos

API Python: Processa dados (calcula média, moda, mediana, etc.)
API Python → Cliente: Response com estatísticas

3. Configuração do JWT_SECRET:

⚠ IMPORTANTE: Ambas as APIs devem compartilhar a mesma chave secreta JWT.

```
# api-java/.env
JWT_SECRET=petdex-secret-key-change-in-production

# api-python/.env
JWT_SECRET=petdex-secret-key-change-in-production
```

4.4.3 WebSocket - Comunicação em Tempo Real

Conexão WebSocket:

```
// Cliente JavaScript (exemplo)
const token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...";
const animalId = "68194120636f719fcd5ee5fd";

// Conectar ao WebSocket
const ws = new WebSocket(`ws://172.206.27.122:8080/ws-petdex?token=${token}`);

// Inscrever-se em tópico do animal
ws.onopen = () => {
  ws.send(JSON.stringify({
    type: 'SUBSCRIBE',
    destination: `/topic/animal/${animalId}`
  }));
};

// Receber mensagens
```

```
ws.onmessage = (event) => {
  const data = JSON.parse(event.data);

  if (data.messageType === 'heartrate_update') {
    console.log('Novo batimento:', data.frequenciaMedia);
    updateHeartRateUI(data.frequenciaMedia);
  }

  if (data.messageType === 'location_update') {
    console.log('Nova localização:', data.latitude, data.longitude);
    updateMapUI(data.latitude, data.longitude);
  }
};
```

Tipos de Mensagens WebSocket:

Atualização de Batimento Cardíaco:

```
{
  "messageType": "heartrate_update",
  "animalId": "68194120636f719fcd5ee5fd",
  "coleiraId": "coleira-001",
  "frequenciaMedia": 85,
  "timestamp": "2025-01-18T14:30:00Z"
}
```

Atualização de Localização:




```
{
  "messageType": "location_update",
  "animalId": "68194120636f719fcd5ee5fd",
  "coleiraId": "coleira-001",
  "latitude": -23.5505,
  "longitude": -46.6333,
  "timestamp": "2025-01-18T14:30:00Z",
  "isOutsideSafeZone": false,
  "distanciaDoPerimetro": 15.5
}
```

5. Guia de Configuração e Execução Local



5.1 Pré-requisitos

Antes de executar o projeto localmente, certifique-se de ter instalado:



Ferramentas Essenciais:

-  **Git** - [Download](#)
-  **Docker** (opcional, mas recomendado) - [Download](#)
-  **MongoDB** (local ou acesso ao MongoDB Atlas) - [Download](#)




Para API Java:

-  **Java 21** ou superior - [Download OpenJDK](#)
-  **Maven 3.8+** (ou use o Maven Wrapper incluído)


Para API Python:

-  **Python 3.11** ou superior - [Download](#)
-  **pip** (gerenciador de pacotes Python)

Para Aplicativo Mobile:

-  **Flutter SDK 3.0+** - [Guia de Instalação](#)
-  **Android Studio** (para Android) - [Download](#)
-  **Xcode** (para iOS, apenas macOS) - [Mac App Store](#)

5.2 Configuração e Execução da API Java

 **RECOMENDAÇÃO:** Para a maioria dos usuários, recomendamos utilizar a API Java em produção que já está rodando em nossos servidores (<http://172.206.27.122:8080>). A configuração local é necessária apenas para desenvolvimento ou testes avançados.

1. Clone o repositório:

```
git clone <https://github.com/FatecFranca/DSM-P4-G07-2025-1.git>
cd DSM-P4-G07-2025-1/api-java
```

2. Configure as variáveis de ambiente:

Crie um arquivo `.env` na raiz do projeto:

```
cp .env.example .env
```

Edite o arquivo `.env`:

```
# Configuração do MongoDB
DATABASE_URI=mongodb://localhost:27017/petdex

# Chave secreta JWT (DEVE ser idêntica à da API Python)
JWT_SECRET=petdex-secret-key-change-in-production

# Configuração BCrypt
BCRYPT_SALT=10
```

3. Instale as dependências:

```
# Usando Maven Wrapper (recomendado)
./mvnw clean install

# Ou usando Maven instalado globalmente
mvn clean install
```

4. Execute a aplicação:

```
# Usando Maven Wrapper
./mvnw spring-boot:run

# Ou usando Maven instalado globalmente
mvn spring-boot:run
```

5. Acesse a aplicação:

- **API Base:** <http://localhost:8080>
- **Swagger UI:** <http://localhost:8080/swagger>
- **WebSocket:** <ws://localhost:8080/ws-petdex>

Comandos Úteis:

```
# Compilar sem executar testes
./mvnw clean package -DskipTests

# Executar apenas os testes
./mvnw test

# Gerar JAR para produção
./mvnw clean package

# Executar o JAR gerado
java -jar target/api-java-0.0.1-SNAPSHOT.jar
```

Executar com Docker (Opcional):

```
# Construir a imagem
docker build -t petdex-api-java .

# Executar o container
docker run -p 8080:8080 --env-file .env petdex-api-java
```

5.3 Configuração e Execução da API Python

💡 RECOMENDAÇÃO: Para a maioria dos usuários, recomendamos utilizar a API Python em produção que já está rodando em nossos servidores (<http://172.206.27.122:8083>). A configuração local é necessária apenas para desenvolvimento ou testes avançados.

1. Clone o repositório (se ainda não fez):

```
git clone <https://github.com/FatecFranca/DSM-P4-G07-2025-1.git>
cd DSM-P4-G07-2025-1/api-python
```

2. Crie e ative um ambiente virtual:

```
# Criar ambiente virtual
python -m venv .venv

# Ativar no Linux/Mac
source .venv/bin/activate

# Ativar no Windows (PowerShell)
.venv\Scripts\Activate.ps1

# Ativar no Windows (CMD)
.venv\Scripts\activate.bat
```

3. Configure as variáveis de ambiente:

Crie um arquivo `.env` na raiz do projeto:

```
cp .env.example .env
```

Edite o arquivo `.env`:

```
# URL da API Java (servidor de produção - RECOMENDADO)
API_URL=http://172.206.27.122:8080

# Para desenvolvimento local (não recomendado - requer API Java rodando localmente)
# API_URL=http://localhost:8080
```

```
# Chave secreta JWT (DEVE ser idêntica à da API Java)
JWT_SECRET=petdex-secret-key-change-in-production
```

4. Instale as dependências:

```
pip install -r requirements.txt
```

5. Execute a aplicação:

```
# Modo desenvolvimento (com reload automático)
uvicorn app.main:app --reload --host 0.0.0.0 --port 8083

# Ou usando o script run.py
python run.py
```

6. Acesse a aplicação:

- **API Base:** <http://localhost:8083>
- **Swagger UI:** <http://localhost:8083/docs>
- **ReDoc:** <http://localhost:8083/redoc>
- **Health Check:** <http://localhost:8083/health>

Comandos Úteis:

```
# Atualizar dependências
pip install --upgrade -r requirements.txt

# Congelar dependências atuais
pip freeze > requirements.txt

# Executar em modo produção (sem reload)
uvicorn app.main:app --host 0.0.0.0 --port 8083

# Executar com múltiplos workers (produção)
uvicorn app.main:app --host 0.0.0.0 --port 8083 --workers 4
```

Executar com Docker (Opcional):

```
# Construir a imagem
docker build -t petdex-api-python .

# Executar o container
docker run -p 8083:8083 --env-file .env petdex-api-python
```

5.4 Configuração e Execução do Aplicativo Flutter

💡 **RECOMENDAÇÃO:** Para a maioria dos usuários, recomendamos instalar o APK pré-compilado (veja seção 5.5) que já vem configurado com os servidores de produção. A configuração local é necessária apenas para desenvolvimento.

1. Clone o repositório (se ainda não fez):

```
git clone <https://github.com/FatecFranca/DSM-P4-G07-2025-1.git>
cd DSM-P4-G07-2025-1/mobile
```

2. Configure a API do Google Maps:

Para Android:

Edite `android/app/src/main/AndroidManifest.xml` :

```
<manifest ...>
  <application ...>
    <meta-data
      android:name="com.google.android.geo.API_KEY"
      android:value="SUA_CHAVE_API_AQUI"/>
    </application>
  </manifest>
```

Como obter a chave:

1. Acesse [Google Cloud Console](#)
2. Crie um projeto ou selecione um existente
3. Ative as APIs: Maps SDK for Android e Maps SDK for iOS
4. Vá em Credenciais → Criar Credenciais → Chave de API
5. Copie a chave gerada

3. Configure as variáveis de ambiente:

Crie um arquivo `.env` na raiz do projeto mobile:

```
cp .env.example .env
```

Edite o arquivo `.env` :

```
# URLs das APIs (servidor de produção - RECOMENDADO)
API_JAVA_URL=http://172.206.27.122:8080
API_PYTHON_URL=http://172.206.27.122:8083

# Para desenvolvimento local (não recomendado - requer APIs rodando localmente)
# API_JAVA_URL=http://localhost:8080
# API_PYTHON_URL=http://localhost:8083

# Chave do Google Maps
GOOGLE_MAPS_API_KEY=sua_chave_do_google_maps_aqui
```

⚠ **IMPORTANTE:** Ao usar os servidores de produção, você terá acesso aos dados reais e funcionalidades completas sem precisar configurar as APIs localmente.

4. Instale as dependências:

```
flutter pub get
```

5. Execute o aplicativo:

Em um emulador ou dispositivo conectado:

```
flutter run
```

Para compilar para produção:

```
# Android
flutter build apk --release

# iOS
flutter build ios --release
```

Comandos Úteis:

```
# Verificar ambiente Flutter
flutter doctor

# Listar dispositivos disponíveis
flutter devices

# Executar em dispositivo específico
flutter run -d <device_id>

# Limpar cache de build
flutter clean

# Atualizar dependências
flutter pub upgrade
```

5.5 Instalação do APK no Celular Android

Se você **não deseja configurar o ambiente de desenvolvimento Flutter**, pode instalar o aplicativo diretamente no seu celular Android usando o arquivo APK pré-compilado.

💡 **RECOMENDAÇÃO:** Esta é a forma mais fácil e rápida de testar o PetDex! O APK já vem configurado com os servidores de produção.

5.5.1 Download do APK

 **Link para Download:** [PetDex APK - Google Drive](#)

5.5.2 Passo a Passo para Instalação

1. Habilitar Instalação de Fontes Desconhecidas:

Antes de instalar o APK, você precisa permitir a instalação de aplicativos de fontes desconhecidas:

- Abra **Configurações** no seu celular Android
- Vá em **Segurança** ou **Privacidade**
- Procure por **Fontes Desconhecidas** ou **Instalar apps desconhecidos**
- Habilite a opção para o navegador ou gerenciador de arquivos que você usará

2. Baixar o APK:

- Acesse o link do Google Drive no seu celular
- Clique em **Download** ou no ícone de download (↓)
- Aguarde o download ser concluído
- O arquivo será salvo na pasta **Downloads** do seu celular

3. Instalar o APK:

- Abra o **Gerenciador de Arquivos** do seu celular
- Navegue até a pasta **Downloads**
- Localize o arquivo **PetDex.apk** (ou nome similar)
- Toque no arquivo APK
- Clique em **Instalar**
- Aguarde a instalação ser concluída
- Clique em **Abrir** ou localize o ícone do PetDex na tela inicial

4. Primeiro Acesso:

Ao abrir o aplicativo pela primeira vez:

- Você verá a **tela de login**

- Use as credenciais padrão (veja seção 5.6.1 abaixo)
- Após o login, você terá acesso completo a todas as funcionalidades

5.5.3 Permissões Necessárias

O aplicativo solicitará as seguintes permissões:

- 📍 **Localização:** Para exibir o mapa e a localização do pet
- 🔔 **Notificações:** Para receber alertas de batimentos anormais e fuga
- 📶 **Internet:** Para comunicação com as APIs

Importante: Aceite todas as permissões para garantir o funcionamento completo do aplicativo.

5.5.4 Requisitos do Dispositivo

- **Sistema Operacional:** Android 5.0 (Lollipop) ou superior
- **Espaço em Disco:** Mínimo de 100 MB livres
- **Conexão com Internet:** Wi-Fi ou dados móveis
- **GPS:** Necessário para funcionalidades de localização

5.5.5 Configuração Inicial

Após instalar o APK, o aplicativo já está configurado para se conectar aos servidores de produção:

- **API Java:** <http://172.206.27.122:8080>
- **API Python:** <http://172.206.27.122:8083>

Não é necessário configurar nada! Basta fazer login e começar a usar.

5.5.6 Solução de Problemas

Problema: "Aplicativo não instalado"

- Verifique se você habilitou a instalação de fontes desconhecidas
- Certifique-se de que há espaço suficiente no dispositivo
- Tente baixar o APK novamente

Problema: "Erro ao fazer login"

- Verifique sua conexão com a internet
- Confirme se está usando as credenciais corretas (veja seção 5.6.1)
- Tente novamente após alguns segundos

Problema: "Mapa não carrega"

- Verifique se concedeu permissão de localização
- Ative o GPS do dispositivo
- Verifique sua conexão com a internet

Problema: "Dados não aparecem"

- Certifique-se de estar usando o usuário padrão (henriquealmeidaflorentino@gmail.com)
- Aguarde alguns segundos para os dados carregarem
- Verifique se as APIs estão online (acesse os links do Swagger)

5.6 Usuários e Autenticação

5.6.1 🔑 Credenciais Padrão para Login

⚠️ **IMPORTANTE:** Use estas credenciais para testar o aplicativo!

Para testar o aplicativo com dados reais (seja via **APK instalado** ou executando via **Flutter**), utilize as seguintes credenciais:

Email: henriquealmeidaflorentino@gmail.com
Senha: senha123

Copie e cole no aplicativo:

Campo	Valor
Email	henriquealmeidaflorentino@gmail.com
Senha	senha123

Formato JSON (para testes via API):

```
{
  "email": "henriquealmeidaflorentino@gmail.com",
  "senha": "senha123"
}
```

5.6.2 Como Fazer Login no Aplicativo

Passo a Passo:

1. Abra o aplicativo **PetDex** no seu celular ou emulador
2. Na tela de login, você verá dois campos:
 - Campo de **Email**
 - Campo de **Senha**
3. Digite o email: henriquealmeidaflorentino@gmail.com
4. Digite a senha: senha123
5. Clique no botão **"Entrar"** ou **"Login"**
6. **Aguarde alguns segundos** enquanto o aplicativo se conecta às APIs
7. **Pronto!** Você terá acesso completo a todas as funcionalidades com dados reais

O que você verá após o login:

- ✓ **Tela Inicial:** Localização do pet no mapa + último batimento cardíaco
- ✓ **Dados em Tempo Real:** Atualizações via WebSocket
- ✓ **Histórico Completo:** Batimentos, localizações e movimentos
- ✓ **Análises Estatísticas:** Médias, gráficos e probabilidades
- ✓ **Checkup Inteligente:** Diagnóstico baseado em IA
- ✓ **Área Segura:** Perímetro configurado com alertas

5.6.3 Por Que Usar Este Usuário?

Este usuário possui:

- ✓ **Animal cadastrado:** "Rex" (cachorro da raça Golden Retriever)
- ✓ **Coleira vinculada:** Coleira física enviando dados reais
- ✓ **Histórico de dados:** Mais de 1000 registros de batimentos
- ✓ **Localizações GPS:** Histórico de movimentação
- ✓ **Área segura configurada:** Perímetro de 500 metros
- ✓ **Dados de movimento:** Acelerômetro registrando atividades

5.6.4 Criação de Novos Usuários

Endpoint de Cadastro:

POST <http://localhost:8080/auth/register>

Content-Type: application/json

```
{
  "nome": "João Silva",
  "email": "joao@example.com",
  "senha": "senha123",
  "telefone": "(11) 98765-4321"
}
```

Resposta:

```
{
  "id": "65a1b2c3d4e5f6g7h8i9j0k1",
  "nome": "João Silva",
  "email": "joao@example.com",
  "telefone": "(11) 98765-4321"
}
```

5.6.5 ⚠️ IMPORTANTE: Limitação de Novos Usuários

AVISO CRÍTICO: Quando um novo usuário é cadastrado e um animal também é cadastrado, o aplicativo **NÃO CARREGARÁ CORRETAMENTE** devido à falta de conexão com a coleira física.

Por que essa limitação existe?

O aplicativo depende de dados enviados pela coleira física:

- ❌ **Batimentos cardíacos:** Sem coleira, não há dados de batimentos
- ❌ **Localização GPS:** Sem coleira, não há dados de localização
- ❌ **Movimento:** Sem coleira, não há dados de acelerômetro

Solução para Testes:

✅ **Use as credenciais do usuário padrão** (`henriquealmeidaflorentino@gmail.com` / `senha123`)

Este usuário já possui:

- Animal cadastrado e vinculado à coleira física
- Histórico de batimentos cardíacos
- Histórico de localizações GPS
- Dados de movimento
- Área segura configurada

Fluxo Completo para Produção:

Para que um novo usuário funcione completamente, seria necessário:






1. Cadastrar usuário
2. Cadastrar animal
3. **Vincular coleira física ao animal** (hardware IoT)
4. Coleira começa a enviar dados para a API
5. Aplicativo passa a exibir dados em tempo real



Conclusão

O **PetDex** representa uma solução completa e inovadora para o monitoramento de saúde e segurança de pets, combinando:

- ✅ **Hardware IoT:** Coleira inteligente com sensores de alta precisão

-  **Backend Robusto:** APIs escaláveis em Java e Python
-  **Inteligência Artificial:** Modelo CART com 100% de acurácia em testes
-  **Aplicativo Mobile:** Interface intuitiva e responsiva em Flutter
-  **Infraestrutura em Nuvem:** Deploy automatizado com CI/CD
-  **Comunicação em Tempo Real:** WebSocket para atualizações instantâneas

O projeto demonstra a aplicação prática de conceitos avançados de:

- Desenvolvimento de Software Multiplataforma
- Internet das Coisas (IoT)
- Inteligência Artificial e Machine Learning
- Computação em Nuvem
- DevOps e CI/CD
- Arquitetura de Microsserviços

Referências e Documentação Adicional

- **Repositório GitHub:** <https://github.com/FatecFranca/DSM-P4-G07-2025-1>
- **Vídeo do Projeto:** <https://www.youtube.com/watch?v=9lwRMAMUHo0>
- **Análise IA PetDex Oficial:** [PDF](#)
- **API Java (Swagger):** <http://172.206.27.122:8080/swagger>
- **API Python (Docs):** <http://172.206.27.122:8083/docs>
- **Download APK:** [Google Drive](#)

Equipe de Desenvolvimento

- **Felipe Avelino Pedaes**
- **Gabriel Resende Spirlandelli**
- **Henrique Almeida Florentino**
- **Luiz Felipe Vieira Soares**

PetDex — Cuidando do seu pet com tecnologia e amor 🐾💙

Projeto desenvolvido como parte das atividades acadêmicas da **FATEC** – Faculdade de Tecnologia.
Orientado pelos princípios de inovação, prevenção e bem-estar animal.