Manual de Instalação e Configuração SeePaw

1. Backend

Pré-requisitos

- .NET 8 SDK
- Docker Desktop
- Git
- Visual Studio 2022 ou VS Code (opcional)

Configuração:

1. Clonar o repositório

```
git clone https://github.com/see-paw/backend.git
```

2. Configurar variáveis de ambiente

Criar um ficheiro **.env** na raiz do projeto com o seguinte conteúdo:

```
POSTGRES_DB=seepaw
POSTGRES_USER=seepaw
POSTGRES_PASSWORD=seepawpwd
```

Importante: Este ficheiro **não** é incluído no Git (está listado no .gitignore).

3. Configurar User Secrets (para desenvolvimento local)

Abrir terminal na raíz do projeto de backend: `cd WebAPI`

```
dotnet user-secrets init
dotnet user-secrets set "ConnectionStrings:DefaultConnection"
"Host=localhost;Port=5432;Database=seepaw;Username=seepaw;Password=see
pawpwd"
cd ..
```

4. Iniciar a base de dados

Nota: O Docker Desktop deve estar ativo.

```
docker-compose up -d database
```

O que este comando faz:

- Cria e inicia um container Docker com PostgreSQL.
- O container funciona como um ambiente isolado, sem necessidade de instalar PostgreSQL diretamente no computador.
- O parâmetro -d executa em segundo plano, libertando o terminal.

Verificar o estado do container:

```
docker ps
```

Deve aparecer algo como:

```
CONTAINER ID   IMAGE             STATUS
xxxxxx         postgres:latest   Up X seconds (healthy)
```

5. Executar a aplicação backend

Com terminal aberto na raíz do projeto de backend:

```
dotnet watch --project WebAPI
```

6. Testar

Abrir um browser em:

https://localhost:5001/swagger

Se a interface do Swagger aparecer, está tudo a funcionar!

A partir desta página é possível:

- Consultar todos os endpoints da API
- Ver modelos de request e response
- Enviar pedidos de teste diretamente do browser
- Validar rapidamente se a API está funcional

Fluxo de Desenvolvimento

Criar uma nova migration

Sempre que adicionar ou alterar entidades:

1. Adicionar a entidade na camada Domain
2. Registar no DbContext (Persistence/AppDbContext.cs)
3. Criar a migration

```
cd WebAPI
dotnet ef migrations add NomeDaMigration -p Persistence -s WebAPI
```

4. Aplicar a migration

```
dotnet ef database update update -p Persistence -s WebAPI
```

Atualizar a base de dados após um pull

A aplicação de backend aplica migrations automaticamente ao iniciar! No entanto é necessário repor a Base de Dados local quando é adicionada uma nova migration:

```
cd backend
dotnet ef database drop --project Persistence --startup-project WebAPI --force
dotnet ef database update --project Persistence --startup-project WebAPI
```

Comandos Docker Úteis

Iniciar apenas a base de dados

```
docker-compose up -d database
```

Listar containers em execução

```
docker ps
```

Testar ligação

```
docker exec -it database pg_isready -U seepaw -d seepaw
```

Ver logs da base de dados

```
docker-compose logs database
```

Parar a base de dados

```
docker-compose stop database
```

Parar todos os containers (mantendo os dados)

```
docker-compose down
```

Remover tudo (incluindo dados)

```
docker-compose down -v
```

Atenção: A flag -v remove também os volumes — onde a base de dados guarda a informação.
Todos os dados locais serão **perdidos permanentemente**.

Estrutura do Projeto

```
.env                    # Variáveis de ambiente do Docker (ignorado pelo Git)
docker-compose.yml      # Configuração Docker

Application/            # Lógica de negócio
Domain/                 # Entidades
Persistence/            # Acesso a dados
Tests/                  # Testes unitários
WebAPI/                 # Controllers, middleware e configuração HTTP
Infrastructure/         # Serviços externos e segurança
```

2. Web App

Pré-requisitos

- **Node.js 18+**
- **npm**
- **Git**
- Um editor de código (VS Code recomendado)
- Backend SeePaw já configurado e a correr em: **https://localhost:5001**

---

1. Clonar o repositório

```
git clone https://github.com/see-paw/web-app.git
cd frontend
```

2. Instalar dependências

```
npm install
```

3. Configurar HTTPS (Obrigatório em desenvolvimento)

A aplicação comunica com o backend por HTTPS.
Para evitar erros de ligação, cada membro da equipa deve:

3.1 Gerar e confiar num certificado de desenvolvimento

```
dotnet dev-certs https --trust
```

3.2 Exportar o certificado em dois ficheiros (.crt e .key)

Estes ficheiros serão colocados no frontend para permitir ao Vite servir a aplicação em HTTPS.

Exportar certificado (cert.crt)

```
dotnet dev-certs https -ep cert.crt --format PEM --no-password
```

Exportar chave privada (cert.key)

dotnet dev-certs https -ep cert.key --format PEM --no-password

3.3 Estrutura esperada no frontend

WEB-APP/

└── certs/

  ├── cert.crt

  └── cert.key

4. Criar o ficheiro .env

Na raiz do projeto, criar um ficheiro .env e escrever o seguinte:

`VITE_API_URL=https://localhost:5001/api`

Nota:

- Esta variável diz ao frontend onde está o backend.
- Nunca commitar credenciais nem endpoints privados em .env.
- .env está no .gitignore.

5. Iniciar a Web App

Para fazer build do frontend: `npm run build`

Para iniciar o backend: `npm run start:backend`

Para iniciar o frontend: `npm run start:frontend`

Para iniciar os dois em simultâneo: `npm run start:all`

A aplicação deve abrir em: `https://localhost:3000`

Se os certificados estiverem bem configurados, o browser não dará erros de segurança.

6. Estrutura da Pasta do Projeto

```
src/
├── api/                # Funções de comunicação com o backend
│
├── assets/             # Imagens e ícones
│
├── components/         # Componentes reutilizáveis + layout
│
├── hooks/              # Custom hooks
│
├── lib/                # Libs externas configuradas
│
├── pages/              # Páginas principais da aplicação
│
├── routes/             # Configuração de rotas + loaders
│
├── schemas/            # Schemas Zod para forms e validações
│
├── stores/             # Zustand stores
│
├── styles/             # tokens.css, utilities.css, global.css
│
├── theme/              # Tema global
│
├── types/              # Tipos TypeScript para entidades da aplicação
│
├── utils/              # Funções de utilidade
│
├── App.tsx             # Componente raíz da aplicação
└── main.tsx            # Ponto de entrada React/Vite
```

```
e2e/
├── components/         # Component Objects
│
├── core/               # BasePage e helpers
│
├── fixtures/           # Fixtures customizadas
│
├── pages/              # Page Objects
│
├── test-data/          # Dados mock para testes
│
└── tests/              # Testes E2E
```

3. Android

Pré-Requisitos

- Android Studio (https://developer.android.com/codelabs/basic-android-kotlin-compose-install-android-studio#0)

- Git (https://git-scm.com/install/)

- Projeto de backend configurado.

- Projeto da web app configurado (opcional).

---

1. Clonar o projeto

`git clone https://github.com/see-paw/android-app.git`

2. Abrir o projeto de backend e iniciar o container da base de dados.

Iniciar o container docker: `docker compose up -d database`

Iniciar o projeto: `dotnet run watch –project WebApi`

Deve ser inicializado através do projeto da web app, se for pretendido testar o fluxo de aprovação de Ownerships.

Iniciar o projeto web app + backend: `npm run start:all`

A gestão de *Ownerships* através de administradores de centros de abrigo animal encontra-se na *Web App*. Para que seja possível testar o fluxo de aprovação em tempo real, é aconselhado o arranque através da Web App em simultâneo com o projeto de Android.

3. Caso o objetivo seja correr o projeto no smartphone, para que funcione no smartphone, antes de correr a app escrever no terminal:

Windows

```
C:\Users\<USER_NAME>\AppData\Local\Android\Sdk\platform-tools\adb.exe reverse tcp:5000 tcp:5000
```

➔ substituir <USER_NAME> pelo nome do utilizador

Mac

```
adb reverse tcp:5000 tcp:5000
```

ou

```
~/Library/Android/sdk/platform-tools/adb reverse tcp:5000 tcp:5000
```

Para utilizar a app através do emulador do android studio basta desligar a ligação do smartphone com a máquina, e correr a aplicação.

Sempre que o objetivo for de usar o smartphone, deve-se usar o comando anterior antes de arrancar com a aplicação. Se o smartphone não for desligado do computador (usb) não é necessário voltar a repetir o comando.

É necessário fazer este passo pois o smartphone não consegue aceder ao localhost do PC. Não sabe qual é o endereço usado, `adb reverse tcp:5000 tcp:5000` serve para fazer "Reverse Port Forwarding". Este sistema não é persistente daí ter que ser feito sempre que o objetivo é o de arrancar com a app no smartphone.