Ministério da Cultura - Mapasv7

Documento de Arquitetura de Software

Cédric Lamalle

Versão 1.0.2, 15/03/2024

Índice

Histórico de Revisões

Data	Versão	Descrição	Autor	Revisor
14/12/2023	1.0.0	Criação do documento	Cédric Lamalle	Matheus Carvalho
26/02/2024	1.0.1	Correção no Ambiente de Desenvolvimento	Adilson Balbino	Cédric Lamalle
15/03/2024	1.0.2	Correção no tipo de banco no diagrama de implantação	Cédric Lamalle	Matheus Carvalho

1. Introdução

Este documento visa descrever a arquitetura lógica e física do projeto Mapas Culturais v7.

1.1. Definições, Acrônimos e Abreviações

API

Application Programming Interface: É qualquer interface que expõe as funcionalidades de um sistema.

Leaflet

É uma biblioteca JavaScript de código aberto usada para construir aplicativos web para interagir com mapas.

ORM

Object Relational Mapping Mapeamento Objeto Relacional, técnica que permite mapear tabelas de banco de dados a classes em uma linguagem orientada a objeto.

REST

Representional State Transfer: É um modelo de arquitetura que descreve regras para expor serviços por recursos e determinar ações sobre esses recursos conforme os verbos HTTP.

1.2. Documentos de Referência

- Plano de Implantação do Mapasv7
- Nota Técnica de comparação entre a v4 e a v7 do Mapas

2. Objetivo do Documento

O objetivo deste documento é apresentar a arquitetura do projeto Mapas Culturais.

3. Visões da Arquitetura

3.1. Critérios de Avaliação Arquitetural

Não se aplica.

3.2. Arquitetura imposta pelo cliente

Não se aplica.

3.3. Arquiteturas descartadas

Não se aplica.

3.4. Visão Geral das Camadas Arquiteturais

Em arquitetura de sistemas, damos o nome de camada a uma estrutura lógica de componentes de um software que interagem entre si para cumprir um objetivo específico, responsabilizando-se pela execução de uma dada tarefa ou um conjunto de tarefas semelhantes. Em alguns casos, esse agrupamento pode não ser apenas lógico, mas sim assumir um caráter físico. Isso ocorre quando uma camada possui a habilidade de ser executada separadamente do restante da aplicação.

A arquitetura baseada em múltiplas camadas oferece um modelo flexível e reutilizável para o desenvolvimento de software. Ao se dividir uma aplicação em camadas, evitamos o forte acoplamento entre os componentes, de forma que a maioria das alterações necessárias possam ser feitas de maneira mais isolada e pontual, reduzindo o esforço necessário à manutenção do sistema.

Na figura abaixo, apresentamos uma visão simplificada das camadas que compõem o sistema e como elas se relacionam. Nas seções a seguir, detalhamos conceitualmente cada camada e enumeramos as ferramentas e plataformas que serão utilizadas para sua construção.

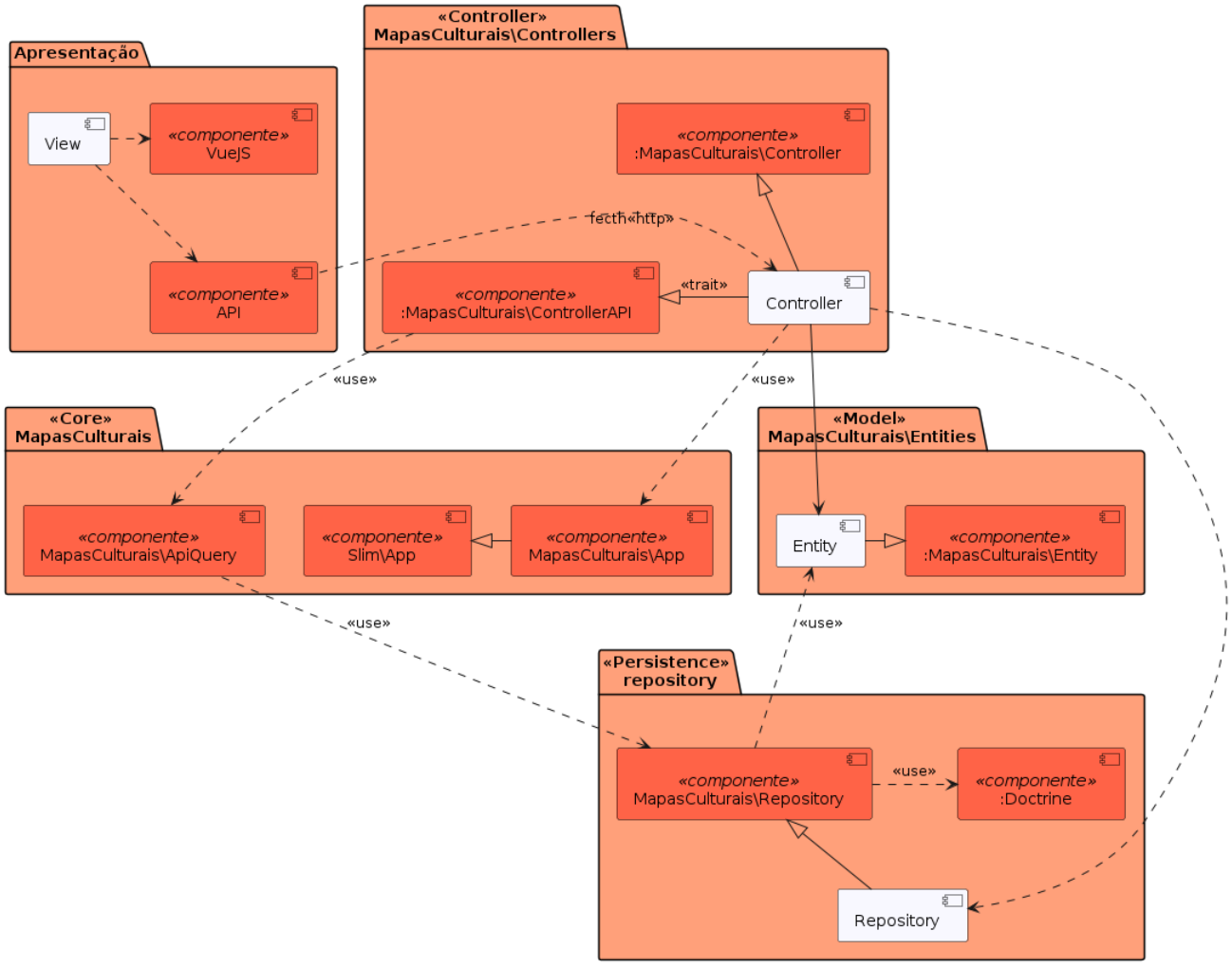

Imagem 1. Visão Geral das Camadas Arquiteturais

3.5. Camada de Apresentação

A camada de apresentação é responsável por fazer a lógica de construção das páginas para serem exibidas pelos usuários, tratar os eventos do navegador, como cliques, e gerenciar o fluxo de execução do sistema. Nesta camada o sistema utiliza a biblioteca VueJS para criar *views* a partir de componentes o misturando com PHP.

3.6. Camada de *Controllers*

O objetivo da camada de controllers é prover um meio de comunicação entre o sistema que vai ser desenvolvido e os demais sistemas que o circundam por meio de *APIs REST*. Os dados recebidos nesta camada, vindo da camada de apresentação ou de interação com outros sistemas são validados e transformados. Por exemplo, os campos de um formulário podem ser mapeados em entidades. A interação com o banco de dados é feita usando o componente APIQuery para as buscas. As operações de criação, alteração e deleção são repassadas para a camada de persistência usando os *Repositories*.

3.7. Camada *Models*

A camada de modelo contém as entidades mapeadas conforme as tabelas do banco de dados usando a técnica de *ORM*.

3.8. Camada *Core*

Esta camada contém os principais componentes, como *ApiQuery* para buscas, *App* o ponto de entrada da aplicação onde são registrados a configuração, os *Controllers*, *Hooks*, *APIs*, grupo de arquivos, etc.

3.9. Camada de Persistência

A camada de persistência é responsável pela lógica de acesso ao banco de dados e pelo mapeamento dos dados em entidades representativas. O objetivo em mapear o banco de dados em entidades representativas ao sistema é diminuir a diferença semântica entre o modelo abstrato do banco e o negócio do sistema.

3.10. Requisitos para Implementação das Camadas

Os itens apresentados a seguir referem-se à lista de tecnologias e requisitos necessários para suportar as camadas de apresentação, negócio e persistência:

- PHP 8.2, com os seguintes módulos:
  - php-gd
  - php-cli
  - php-json
  - php-curl

    - php-pgsql
    - php-apc
- Composer
- Docker
- NodeJS
- Ruby

Bibliotecas a serem utilizadas pela aplicação:

- Slim 4
- Doctrine ORM 2.16
- Mustache PHP 2.11
- VueJS 3.3
- Leaflet 1.7

3.11. Visão Detalhada das Camadas

A seguir, apresentamos uma visão detalhada das camadas do sistema, na qual são ilustrados os principais componentes que compõem cada camada bem como os sistemas externos com o, qual o sistema deverá interagir.

3.12. Visão Geral do Processo

O diagrama de sequência abaixo apresenta o fluxo de informações do sistema e suas interfaces através das camadas arquiteturais.

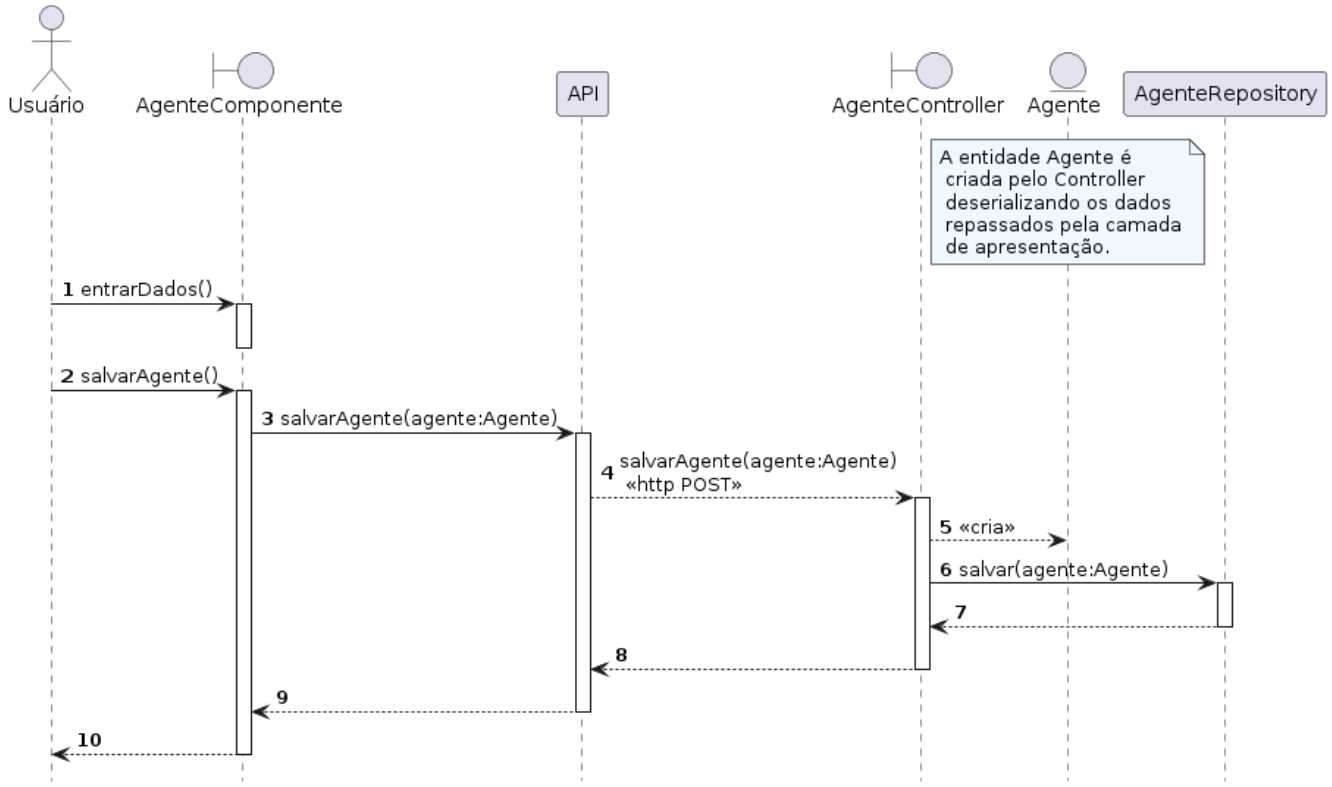

Imagem 2. Visões da arquitetura

3.13. Visão dos Módulos do Sistema

Neste item apresentamos uma visão geral dos módulos do projeto, ilustrando-os graficamente por um diagrama. Em seguida, detalhamos as responsabilidades de cada módulo, em concordância com a especificação funcional.

3.14. Rastreabilidade

Não se aplica.

3.15. Visão de Integração

A visão de integração apresenta as integrações pertinentes ao sistema. Essas integrações podem ser internas ou externas e é de suma importância que os projetistas e desenvolvedores conheçam em detalhes essas integrações, bem como os contratos e protocolos de comunicação entre os sistemas/componentes envolvidos na integração.

O Sistema deve se integrar aos seguintes sistemas a fim de atender os requisitos funcionais e não-funcionais da aplicação:

- CEP Aberto
- Open StreetMap
- Provedor de identidade OpenID

A integração com o CEP permite fazer pesquisa de endereços a partir de CEP ou o contrário. Essa integração é efetuada utilizando a API Rest disponibilizada pelo serviço.

Essa integração necessita uma conta (gratuita) no CEP Aberto para criação de um *Token* de autenticação.

A integração com OpenStreetMap é efetuado no navegador utilizando a biblioteca *Leaflet* que busca as *tiles* via *http* para montar os mapas com quais o usuário pode interagir. Por exemplo, é possível mudar a posição do mapa e efetuar *zooms*.

Em relação ao provedor de identidade OpenID, a integração é opcional e serve a autenticar o usuário no sistema. Um exemplo de provedor OpenID é o *acesso.gov* do Governo Federal.

3.16. Visão de Classes

Nesta seção, estão descritos os diagramas/fluxogramas de classe, sequencia e atividades para os requisitos críticos do sistema.

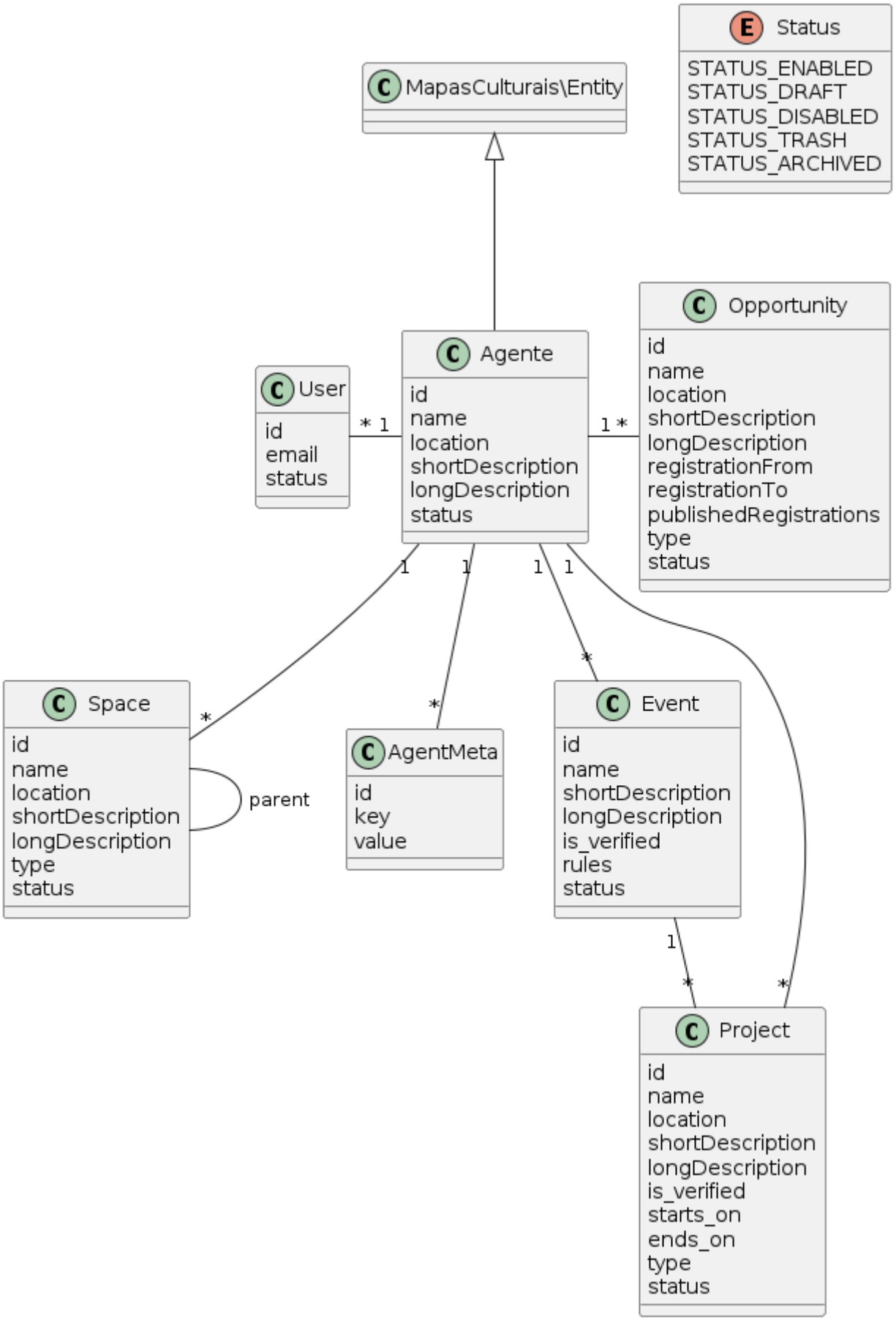

Imagem 3. Diagrama de classes

Todas as entidades herdam da classe `MapasCulturais\\Entity` mas isso não foi representado para não sobrecarregar o diagrama.

4. Ambiente de Desenvolvimento

Ferramentas a serem utilizadas no ambiente de desenvolvimento:

- Eclipse IDE ou PHP Storm
- Docker
- Composer
- PHP 8
- NodeJS
- Ruby

Ferramentas necessárias para dar suporte à aplicação e ao desenvolvimento são:

- DBeaver

A solução prevê acessos às seguintes bases de dados:

- PostgreSQL 10.0 ou superior

A correção dos erros de carregamento da página no Ambiente de Desenvolvimento pode ser resolvidos executando o seguinte comando após a execução do docker compose:

```
docker compose exec mapasculturais sh -c "chmod -R 777 public/assets/"
```

5. Ambiente de Testes

Ferramentas de apoio aos testes e à codificação:

- Jenkins
- Sonar

6. Ambiente de Implantação

6.1. Ambiente de Implantação

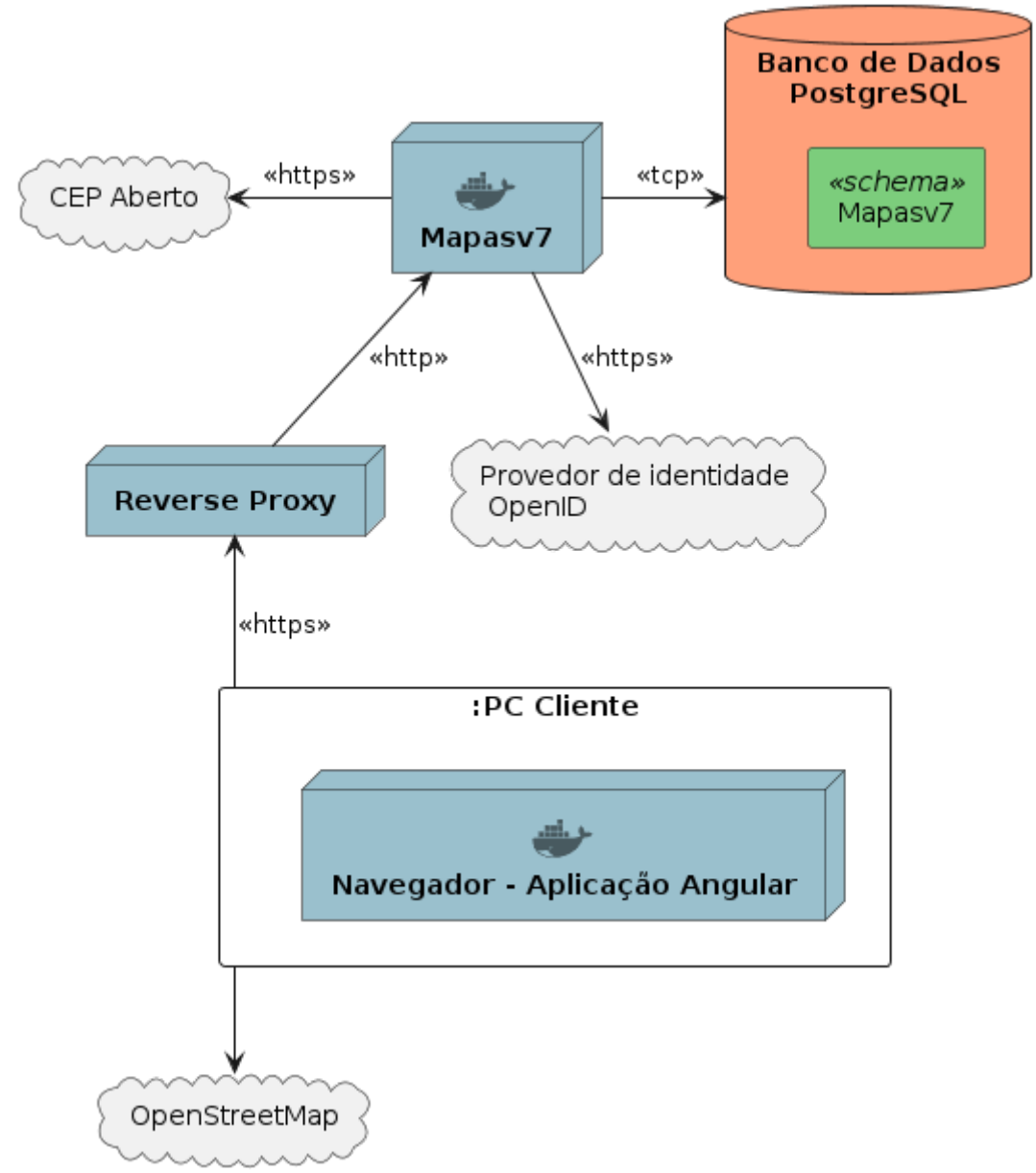

Imagem 4. Diagrama de Implantação