Universidade do Minho
Departamento de Informática

Engenharia Web

Mapa das Ruas de Braga

Grupo *La Familia*:
José Carvalho - A94913
Miguel Filipe Cidade da Silva - A97031
João Gonçalo de Faria Melo - A95085

21 de junho de 2023

Conteúdo

1 Introdução

Neste documento iremos relatar todo o processo de desenvolvimento da aplicação *web* desenvolvida para a unidade curricular de Engenharia Web.

O docente disponibilizou uma lista de possíveis projetos, sendo que também possibilitou que os alunos escolhessem um projeto desde que esse seja validado com ele e que faça sentido tanto para os âmbitos da disciplina como para utilização no mundo real, pois nenhum dos enunciados é um problema académico, sendo todos problemas reais.

Optamos por implementar uma plataforma para o mapa das ruas de Braga.

Escolhemos este tema pois achamos que seria uma ferramenta interessante e de interesse comunitário pois seria uma forma acessível de conhecer e documentar a história das ruas que compõem, aquela que é oficialmente a cidade mais antiga do país. Também, sentimos que este projeto, se for bem implementado seria uma forma excelente de solidificar os conceitos aprendidos durante a unidade curricular.

Relativamente ao esquema do relatório, primeiramente iremos abordar as tecnologias usadas, bem como aspetos de planeamento e estruturação do projeto. De seguida iremos tratar pormenores da implementação, onde falamos como tratamos os dados, as rotas que definimos, as páginas *HTML*, programação do lado do cliente e o processo de autenticação. Por último iremos relatar com organizamos o nosso projeto em *containers Docker*.

Para iniciar a nossa aplicação basta consultar as instruções presentes no ficheiro README.md.

2 Objetivos

Os objetivos do projeto estão descritos formalmente no enunciado, estando sempre aberta a porta para a implementação de funcionalidades extra desde que tenhamos o mínimo implementado.

Assim, o que podemos considerar como metas a alcançar neste projeto é:

- Fazer o tratamento do *dataset* fornecido pelo docente;
- Implementar uma aplicação *web* consistente que nos permita realizar operações CRUD e aceder à informação do *dataset* de forma eficaz e organizada;
- Aplicar o conhecimento da disciplina usando os pacotes e tecnologias abordadas em aula;
- Implementar autenticação;
- Estruturar a aplicação de forma a separar os diferentes serviços da nossa aplicação (Dados, Autenticação e Interface).

3 Ferramentas usadas

Para alcançar os objetivos mencionados na secção prévia, recorremos às seguintes tecnologias.

- *Node.js* - ambiente de desenvolvimento;
- *MongoDB* - motor de base de dados não relacional;
- *Postman* - ferramenta utilizada em certas fases do projeto para testar o serviço de dados;
- *Javascript* - linguagem que predomina no nosso projeto, é com ela que montamos a maioria da aplicação *web*;
- *Python* - linguagem utilizada para fazer o tratamento do *dataset* fornecido de modo a o poder importar para o MongoDB;
- *APIs* externas - usamos a *API OpenStreetMap* para abrir numa nova janela a localização das ruas no mapa;
- *Docker* - De forma a facilitar o processo de manutenção da nossa aplicação, utilizamos *containers Docker* no projeto.

Também optamos por utilizar a *stylesheet* w3.css, que é a mesma que utilizamos ao longo das aulas para montar graficamente o nosso *website*. Esta *stylesheet* pode ser consultada em `https://www.w3schools.com/w3css/defaulT.asp`
Dentro do *node* acabamos por utilizar bastantes tecnologias, dentro das quais destacamos: o *Express*, com o qual geramos a estrutura das aplicações que montamos, tendo a linguagem *Pug* como motor para as páginas do nosso *Website*.
Alguns pacotes relevantes que utilizamos no projeto foram:

- *Mongoose* - estabelece a conexão à base de dados, fornecendo métodos que possibilitam a implementação das operações de CRUD;
- *Axios* - permite fazer pedidos *HTTP* entre os diversos serviços criado por nós;
- *Cookie-Parser* - analisa os *cookies*;
- *Morgan* - *logger* de eventos;
- *http-errors* - cria erros *HTTP* para serem utilizados com o *express*.

4 Planeamento da interface

No que diz respeito à interface, decidimos usar tons de azul seguindo a ideologia dos 60-30-10. Utilizamos consistentemente ao longo das interfaces do *website*

as cores *indigo*, *blue* e *light-blue* disponíveis na *stylesheet* w3.css. Optamos por usar tons de azul devido às cores da bandeira da cidade de Braga serem o azul e o branco.

Decidimos por ter no topo do *website* uma espécie de *navigation tabs*. Nela temos botões que redirecionam para a página inicial, para a lista de ruas, para a lista de referências temporais encontradas no sistema, ou para a lista de entidades presentes nas descrições nas ruas. Também em todo o *website*, temos um *footer* estático que contém os nossos nomes e apontadores para páginas pessoais como *GitHub* ou *LinkedIn*.

Cada rua, data ou entidade tem a sua página individual.

No caso da rua, a página individual apresenta fotografias atuais e antigas, uma descrição da história da rua, as casas presentes nessa rua, uma lista de lugares, datas e entidades mencionados na descrição da rua ou das casas da rua em questão. Cada item destas listas é um *link* para a página individual do respetivo item. No princípio e no fundo da página temos botões com opções de remover a rua do sistema, editar a informação da rua, ou abrir a rua no mapa, via a *API OpenStreetMap*. Para além disso, a página de uma determinada rua, contém uma lista de atualizações, em que é indicado o utilizador que realizou uma determinada ação, o que foi feito e a data.

Nas páginas onde temos as listas de ruas, datas ou entidades, temos sempre uma barra de pesquisa que redireciona para a página individual do item caso esse exista no sistema ou, caso contrário, uma página de erro.

Agora iremos mostrar imagens da interface do *website*, começando com o protótipo inicial.

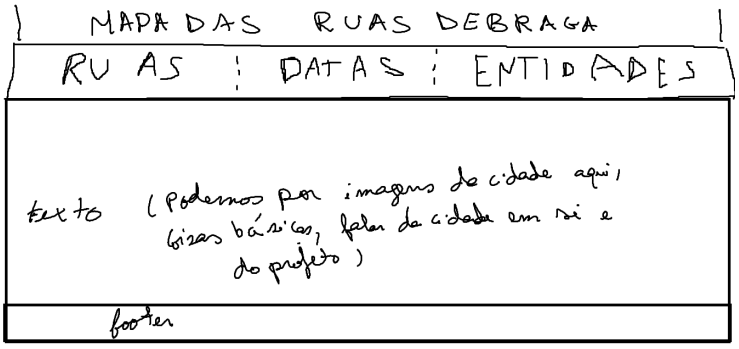

Figura 1: Protótipo Inicial

Este protótipo muito básico não representa a implementação final da aplicação, mas é possível ver a estrutura mencionada com um *header* e *footer* (coloridos com a cor *indigo* da *stylesheet* selecionada após decidirmos as cores).

As imagens finais da nossa aplicação, encontram-se na secção 10.

No nosso sistema para além das imagens fornecidas pelo docente optamos por criar um *favicon* e por utilizar imagens de bancos de imagens públicos para

referenciar os tipos de entidades do sistema.

O nosso *favicon* foi criado usando o website *Photopea*, realizando uma colagem sobrepondo uma imagem "location" de um banco de imagens numa imagem estilizada do contorno do mapa distrito de Braga obtido num **gerador de *favicons* gratuito**.

Figura 2: *Favicon* do nosso *website*

Já no que diz respeito aos tipos de entidades, representamos visualmente quatro possibilidades: pessoa, instituição, empresa e outros (qualquer outro tipo).

Figura 3: Imagem utilizada para definir entidades do tipo pessoa

Figura 4: Imagem utilizada para definir entidades do tipo instituição

Figura 5: Imagem utilizada para definir entidades do tipo empresa

Figura 6: Imagem utilizada para definir entidades de qualquer outro tipo

Estas imagens vem originalmente das seguintes fontes:

- Pessoa - Banco de imagens *Freepik*
- Instituição - Banco de imagens *Freepik*
- Empresa - Banco de imagens *Free ICONS Library*
- Outro - Banco de Imagens *PublicDomainPictures*

5 Arquitetura da aplicação

Relativamente à arquitetura da aplicação, nós dividimos a aplicação em 3 partes distintas. Estas são:

Serviço	Porta
Camada de Interface	7777
Camada de Dados	7776
Camada de Autenticação	7775

Tabela 1: Serviços disponíveis e suas portas

Para além disso contamos com uma pasta chamada *Data* onde colocamos os ficheiros fornecidos pelo docente para alimentar o nosso sistema com dados iniciais, uma *script* em *Python* que faz o tratamento dos dados para um formato usável no motor de base de dados, criando o ficheiro *all_data.json*.

Como a unidade curricular de Engenharia Web não é necessariamente focada em desenvolvimento de *scripts Python* não abordaremos neste documento o código que escrevemos na nossa *script*, no entanto falaremos do ficheiro resultante da sua execução, na secção seguinte.

5.1 Camada de Interface

Esta camada irá ser a principal no projeto. Nela definimos as rotas na aplicação, bem como as páginas *web*. Para ter os dados, esta camada realiza pedidos utilizando o protocolo *HTTP* à camada de dados e à camada de autenticação, quando necessita dos mesmos. Esta camada alberga também os recursos estáticos públicos, nos quais temos o *favicon* e as restantes imagens usadas no *website*.

A comunicação neste serviço é feito na porta 7777 e o código que o implementa está contido na pasta *Website*.

5.2 Camada de Dados

Esta camada, tal como o próprio nome indica, irá ser responsável pelas operações com dados. Podemos encontrá-la na pasta *APIData* e na prática será um serviço muito similar ao *json-server*, mas com funcionalidades específicas para a nossa aplicação e trabalha na porta 7776.

5.3 Camada de Autenticação

Este serviço encontra-se na pasta *APIAutenticacao* e irá ser responsável por verificar se os dados inseridos no *login* ou registo são válidos. Para além disso, este é responsável também por criar *tokens* e verificar se os mesmos são válidos, de forma a termos páginas protegidas. Esta camada recebe e responde a pedidos na porta 7775.

6 Implementação

6.1 Base Dados

6.1.1 Tratamento dos dados

A nossa primeira abordagem ao projeto, foi olhar atentamente para os dados fornecidos pela equipa docente e perceber quais os dados relevantes e como iríamos estruturar a nossa base de dados.

Como nós vamos trabalhar com *MongoDB* neste projeto, tivemos que converter os ficheiros *xml* em ficheiros *json* de forma a adicionar registos diretamente via linha de comando.

Chegamos à conclusão que seria mais fácil meter a informação num único ficheiro e este irá conter toda a informação base da nossa aplicação.

Para gerar esse ficheiro, tivemos recurso ao *Python* usando a biblioteca *xml.etree.ElementTree*.

O ficheiro final em formato *json* irá ser uma lista de dicionários, em que cada um irá representar uma rua.

Por uma questão de organização decidimos separar os lugares, datas e entidades dos parágrafos, porque no final irá no ser útil esta separação trabalhar com as mesmas. Para isso substituímos as respetivas posições com um código identificador. Esse código irá ser #(E|L|D) seguido de um número, em que cada letra irá corresponder a um campo. A letra E é relativo a entidades, a L representa lugares e a D datas. O número seguida da letra irá ser o índice da respetiva entidade, lugar, ou data na lista associada. Por exemplo se tivermos 3 entidades presentes numa rua, a primeira irá ter código #E0, a segunda #E1 e a última

#E2. Com esta associação, a parte de substituir os códigos por texto, não se torna complexa, podendo utilizar módulos de expressões regulares.

Relativamente às figuras, optamos por guardar o caminho das mesmas, em vez de guardar diretamente os ficheiros de vários formatos (*jpg*,*png*,...)

Por último e não menos importante, decidimos adicionar um atributo que irá ser uma lista referente ás últimas atualizações de uma rua. Essas atualizações podem ser, edição de campos, adição de fotografias ou até mesmo a criação das próprias ruas. Uma atualização irá conter quem foi o responsável, o que foi feito na respetiva atualização e a data em que a mesma foi realizada.

6.1.2 Esquema da base de dados

Apresentamos agora como é uma rua é representada:

```
{
    "_id":"nome",
    "figuras_antigas":
    [
        {
            "path" : "path",
            "legenda" : "legenda"
        }
    ],
    "figuras_atuais":
    [
        {
            "path" : "path",
            "legenda" : "legenda"
        }
    ],
    "paragrafos":
    [
        "Exemplo de referencia a uma data #D0",
        "Exemplo de referencia a um lugar #L0",
        "Exemplo de referencia a uma entidade #E0",
        "Exemplo de referencia a uma segunda data #D1"
    ],
    "casas":
    [
        {
            "num":"1",
            "enfiteuta": "Joaquim Alberto",
            "foro": "250 galinhas",
            "desc": [
                 "Exemplo de referencia a uma terceira data #D2",
            ]
        }
    ],
```

```
    "lugares":
    [
        "Rua 1"
    ],
    "datas":
    [
        "1865",
        "2023",
        "sec. XXI"
    ],
    "entidades":
    [
        {
            "nome": "Entidade 1"
            "tipo": "pessoa"
        }
    ]
    "updates" : [
        {
            "username" : "Sistema"
            "message" : "Criacao da rua"
            "date" : "2000-01-01"
        }
    ]
}
```

6.1.3 Operações *CRUD*

A nível de operações de *CRUD*, encontramos-las em duas componentes, temos estas operações na *API* de dados e na *API* de autenticação.

Na *API* de dados quando estamos perante inserção de novas ruas no sistema, edição da informação de ruas já existentes no sistema, remoção de ruas do sistema e também operações de consulta, desde devolver todas as ruas, ou uma rua especifica, ou a lista de entidades, etc.

Já na autenticação temos operações de *CRUD* que permitem criar utilizadores no sistema e também de verificar *tokens*.

6.1.4 Models

Incluímos em cada camada um ficheiro "models", onde estão definidas a estrutura das entidades e respetivos atributos dos objetos/coleções utilizadas no nosso programa. Para isso, foi necessária a utilização da biblioteca *Mongoose*, por forma a criar os "schemas", que representam a estrutura de dados utilizada e trabalhada no programa para realizar operações sobre as informações. Na camada de Dados e de Interface deparamo-nos com o modelo "rua", em que estão definidos a estrutura de dados para figuras, entidades, casas e ruas. Por outro

lado, na camada de Autenticação, foi implementada a estrutura de dados para a definição de um utilizador do sistema, com recurso ao modelo "user".

6.1.5 Controladores

No que toca aos controladores da nossa aplicação, estes são responsáveis pela interação entre o *front-end* e o *back-end*.

6.2 Rotas

Como temos a arquitetura do sistema dividida entre interface, autenticação e *API* de dados acabamos por ter na nossa aplicação vários roteadores, nesta secção iremos abordar as rotas que definimos para aceder às páginas *web* relativas a cada elemento do sistema.

6.2.1 Ruas

Para a coleção de ruas, criamos o ficheiro ruas.js, que apenas conta com uma rota: *ruas/*, na qual implementamos um *GET* de modo a aceder à página com a lista de ruas no sistema. Nessa página, encontra-se uma tabela na qual cada entrada corresponde a uma rua.

Em cada entrada temos o nome da rua, uma parte da sua descrição, e a lista de referências temporais que constam na descrição da rua. Ao clicar no nome da rua o utilizador acede à pagina individual da rua, e ao clicar numa data acede à página individual da data. Devido à complexidade de coisas que implementamos, criamos outro roteador para uma rua especifica, e um roteador para datas.

A nível de páginas *Pug* criadas, apenas temos uma, ruas.pug.

6.2.2 Rua

As rotas para 1 rua específica encontram-se no ficheiros rua.js e são:

6.2.3 Entidades

Decidimos criar também um roteador para as entidades, entidades.js que define duas rotas.

Nas paginas relativas às entidades são apresentadas imagens que representam as entidades, estas imagens aludem ao tipo da entidade. Temos imagens para entidades dos tipos: pessoa, instituição e empresa. Caso a entidade seja de outro tipo aparece a imagem com o caractere '?'.

Na página com a lista de entidades caso o utilizador deseje aceder à página individual da entidade deve clicar no nome da entidade e caso pretenda aceder à pagina de uma das ruas na qual a entidade aparece, deve clicar no nome da rua.

Rota	Método	Objetivo
rua/:id_rua	GET	Retorna página da rua cujo id seja igual a id_rua.
rua/add	GET	Retorna página de criação de uma rua
rua/add	POST	Adicionar a Rua à base de dados.
rua/edit/:id_rua	GET	Retorna página de edição do campos textuais rua
rua/edit/:id_rua	POST	O método POST é submeter as alterações na base de dados.
rua/fotos/:id_rua	GET	Retorna página para adicionar fotos a uma rua, atuais ou antigas.
rua/fotos/:id_rua	POST	O método POST é para adicionar essas fotos à rua em questão na base de dados.

Tabela 2: Rotas de uma rua

Rota	Método	Objetivo
entidades/	GET	Retorna página com a coleção de entidades.
entidade/:entidade	GET	Retorna página com a informação individual de uma entidade.

Tabela 3: Rotas das entidades

A página individual de cada entidade apenas apresenta em maior detalhe a imagem, o nome e tipo da entidade, e a lista de ruas em que a entidade aparece.

6.2.4 Datas

De forma similar ao roteador para as entidades, criamos ainda um roteador para as datas, datas.js que também conta com duas rotas, tal como é representado na tabela 4.

Rota	Método	Objetivo
datas/	GET	Retorna página com a coleção de datas.
datas/data/:data	GET	Retorna página com a informação individual de uma data.

Tabela 4: Rotas das datas

Nas páginas relativas às datas são apresentadas, de forma listada, todas as referências temporais mencionadas na descrição das ruas, incluindo as ruas nas quais a data é referenciada na descrição. A partir desta página, o utilizador pode aceder à página individual da data clicando na mesma.

Na página individual da data é apresentada informação mais específica, nomeadamente, todas as ruas em que a data em questão foi referenciada. O utilizador pode ainda aceder diretamente a uma rua clicando no nome da rua em qualquer uma das páginas.

6.3 Programação do lado do cliente

6.3.1 Rua

Nas páginas relativas a ruas especificas, temos diversos aspetos com programação no lado do cliente.

Para além de aspetos relacionados com a interação com a página, tais como: andar de uma página para outra nas casas na página principal de uma rua, ou de adicionar uma data / entidade ou um lugar na página de edição de uma rua.

O único pormenor a destacar está ligado à página da edição de uma rua. A submissão da edição do cliente, envolve também programação do lado do cliente, pura e exclusivamente para conseguirmos verificar no lado do cliente se a informação que o mesmo inseriu está correta e também para formatar o pacote enviado para o servidor.

6.3.2 Datas, Entidades e Ruas

Nas páginas em que são apresentadas todas as datas, todas as entidades e todas as ruas, temos um pequeno excerto de código que irá simplesmente servir a barra de pesquisa presente nessas páginas, de forma a realizar o redirecionamento para a página de uma data, entidade ou rua específica.

6.4 Autenticação

Em relação à autenticação, não fugimos muito do que aprendemos nas aulas, de forma a simplificar. A estrutura de um utilizador é a seguinte:

```
{
    username: String,
    password: String,
    email: String,
    birth_date: String,
    address: String,
    creation_date: String
}
```

Para o processo de *login*, só temos em conta os campos *username* e *password*. As rotas protegidas com autenticação são todas as rotas que estão relacionadas com a manipulação da informação das ruas na base de dados.

Para gerar *tokens* de autenticação e para validar os mesmos, utilizamos a biblioteca *JWT*, que resumidamente é uma ferramenta que manipulação de *tokens*. Para isso tivemos que definir um segredo, e este tem que ser o mesmo para o processo de criação do *token* e para validar se um *token* é válido. O segredo que usamos para a nossa aplicação é "*RuasBragaEW*".

Para evitar que o utilizador esteja sempre a fazer *login*, os nossos *tokens* tem uma duração de 3600 segundos, ou seja, 1 hora.

7 Docker e Docker Compose

Após o desenvolvimento de uma versão estável e funcional do nosso projeto, pelo menos no que visa os nossos objetivos da disciplina, optamos por utilizar o conhecimento que nos foi passado no final do semestre e compartimentar o nosso programa recorrendo a *containers Docker*.

Para este efeito criamos três *containers*, um para interface, outro para a *API* de dados e por fim um para *API* de autenticação.

Os três *containers* criados comunicam entre si através de uma composição (*Docker Compose*).

O nosso projeto conta com dois ficheiros *shell*, um deles chama-se *rebuild* e o outro *remove*, ao executar os comandos contidos nestes ficheiros reconstruimos ou removemos as imagens criadas para os contentores.

Ao lançar o programa após a primeira composição será de notar que o *MongoDB* não conta com a base de dados necessária então não teremos registos. Para resolver esse problema basta correr os seguintes comandos por ordem um de cada vez, estando localizados na raiz do projeto, ou seja na pasta *ProjetoEW*.

```
sudo docker cp Data/all_data.json ruas-mongodb:/tmp

sudo docker exec -it ruas-mongodb bash

mongoimport -d RuasBragaDB -c ruas --jsonArray /tmp/all_data.json
```

Após correr estes comandos já teremos registos visíveis.

Neste documento não iremos explicar em detalhe as nossas *Dockerfiles*, nem o ficheiro que constrói a composição, pois esses são bastante explícitos, como tal apenas destacamos o facto que usamos as mesmas portas internas e externas no mapeamento. Relembrando que teremos a interface a correr na porta 7777, a *API* de dados na 7776 e a *API* de autenticação na 7776.

8 Conclusão e análise crítica

Analisando o projeto, achamos que conseguimos realizar um bom projeto. Para além dos requisitos do enunciado, implementamos alguns extras. Tentamos ao máximo tornar o código dos nossos servidores de *back-end* o mais simplista e mais eficiente possível. Na interface, dedica-mo-nos a torná-la simples e consistente.

9 Trabalho futuro

Nesta secção iremos abordar alguns extras que poderíamos ter implementado no projeto.

- Autenticação usando serviços tais como *Facebook* e *Gmail*.
- Usar o *Google Maps* em vez do *OpenStreetMap*.
- Aumentar a interação entre o utilizador, implementando um sistema que permita comunicação entre eles para, por exemplo, sugerirem e comunicarem alterações.
- Implementar a remoção de imagens nas paginas individuais das ruas.
- Dar a possibilidade dos utilizadores alterarem os seus dados, por exemplo, a password.

10 Páginas *Web* Criadas

Nesta última secção, em jeito de anexo, deixamos imagens da nossa interface.

Figura 7: Página inicial do nosso *website*

Figura 8: Página com a listagem de todas as ruas do sistema

Figura 9: Página que apresenta a coleção de entidades do sistema

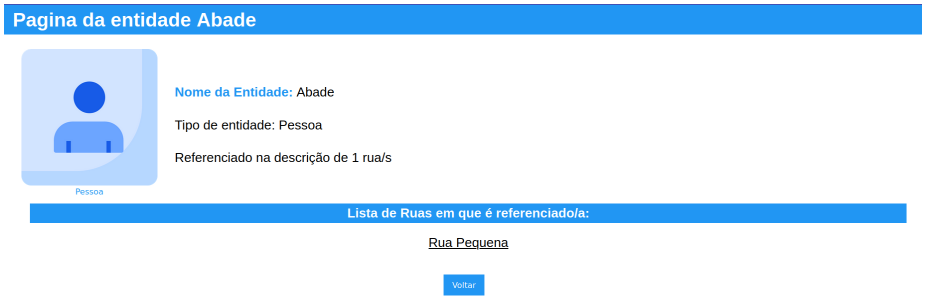

Figura 10: Página que apresenta a informação individual de uma entidade do sistema

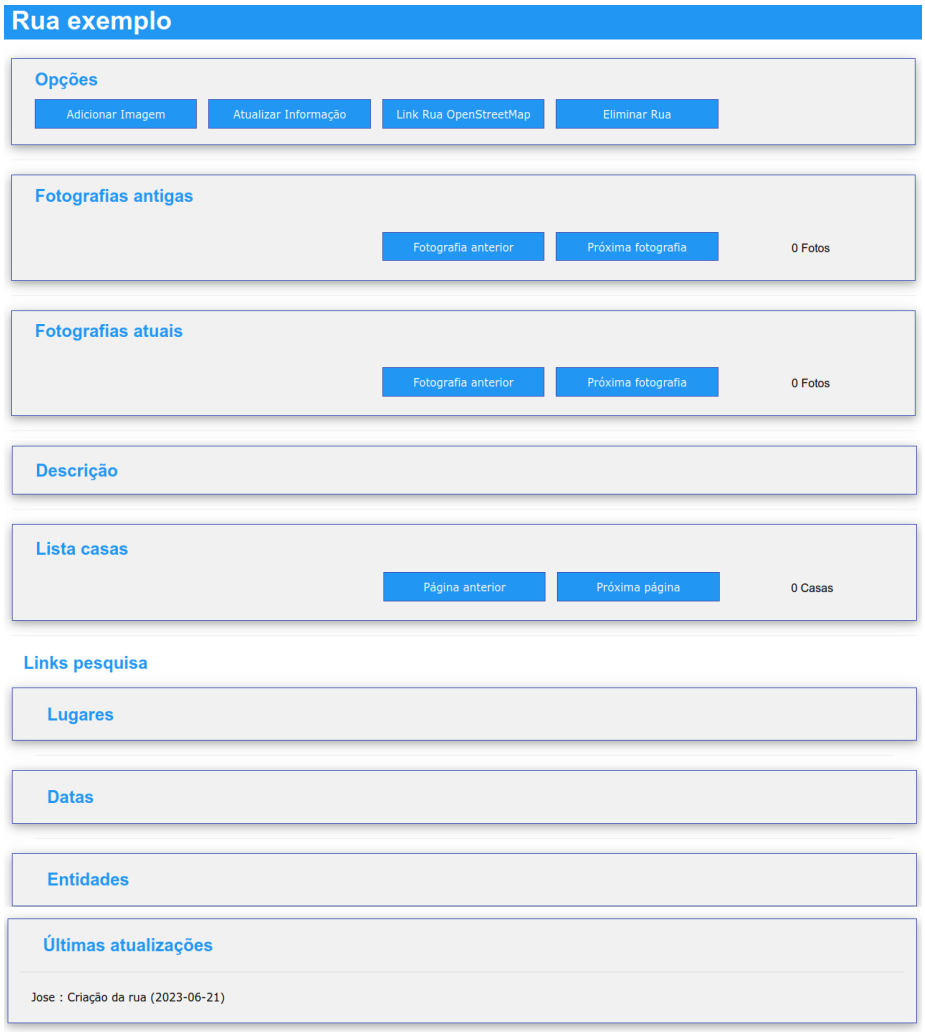

Figura 11: Página de uma rua

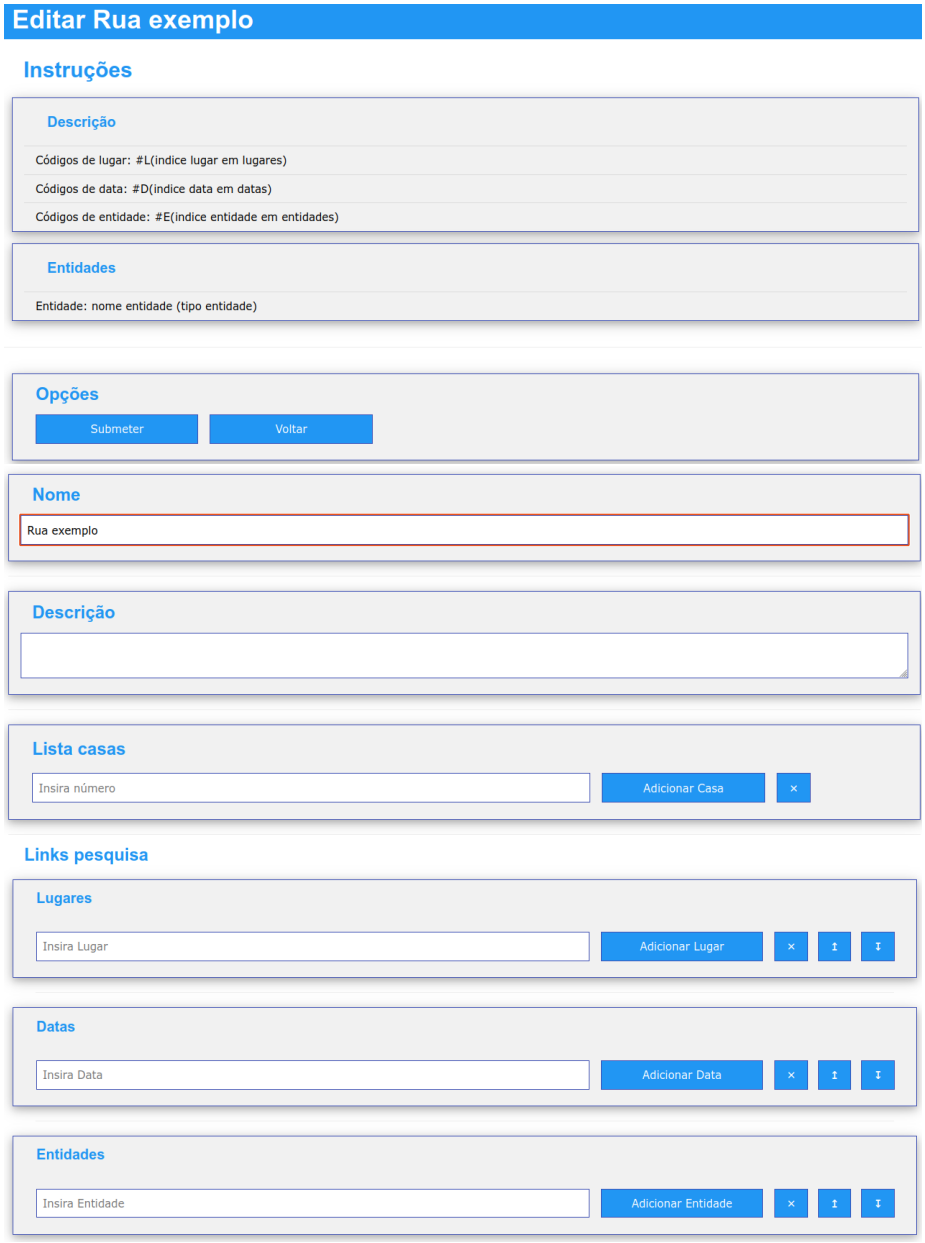

Figura 12: Página de edição de uma rua

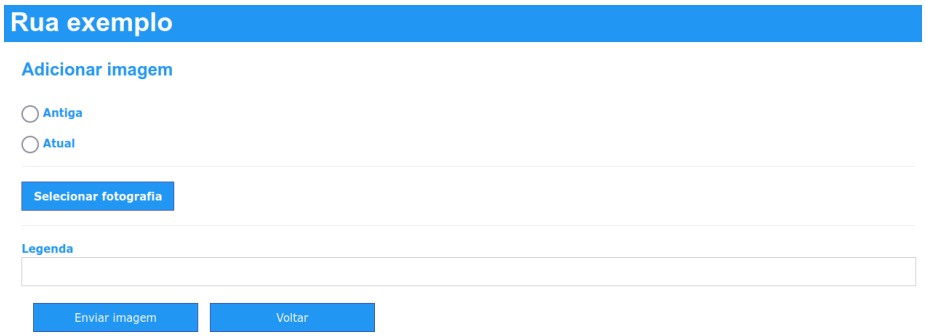

Figura 13: Página para adicionar uma imagem no sistema