

**CENTRO PAULA SOUZA
FACULDADE DE TECNOLOGIA DE FRANCA
“Dr. THOMAZ NOVELINO”**

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

JOSÉ PAULO ARCHETTI CONRADO

CAMPGROUND YELLOW PAGE WEB APP

Sistema para divulgar, localizar e avaliar acampamentos

Trabalho de Engenharia de Software II
apresentado à Faculdade de Tecnologia de
Franca - “Dr. Thomaz Novelino”, como parte dos
requisitos obrigatórios de desenvolvimento de
um sistema.

Orientador: Prof. Me. Fausto Gonçalves Cintra

FRANCA/SP

2022

Sistema para divulgar, localizar e avaliar acampamentos

José Paulo Archetti Conrado¹

Resumo

Acampamento de lazer é uma atividade ao ar livre que envolve dormir fora de casa, na natureza sem abrigo, ou utilizando barraca ou trailer. Tipicamente os integrantes deixam as cidades para desprender seu tempo em áreas ao ar livre numa maneira mais natural, através de atividades de diversão e experiencias educacionais. Uma noite no acampamento vem acompanhado de atividades como trilhas ecológicas, *picnics* e de atividades recreacionais. O aplicativo almeja que os proprietários dos acampamentos encontrem uma maneira fácil de divulgar seus empreendimentos, e que os clientes, amantes da natureza de encontrá-los.

Palavras-chave: avaliação de acampamentos, entretenimento, divulgação de acampamentos, geolocalização, turismo em acampamentos, turismo de aventura.

Abstract

Camping is an outdoor activity involving overnight stays away from home, either without shelter or using basic shelter such as a tent or a recreational vehicle. Typically, participants leave developed areas to spend time outdoors in more natural ones in pursuit of activities providing them enjoyment or an educational experience. The night spent outdoors distinguishes camping from day-tripping, picnicking, and other similarly short-term recreational activities. Being a part of the campground APP network puts a camping site in the hands of campers.

Keywords: adventure tourism, campground industry, campground reviews, campground's location, entertainment, geolocation, finding campgrounds.

¹ Graduando em Análise e Desenvolvimento de Sistemas pela Fatec Dr Thomaz Novelino – Franca/SP.
Endereço eletrônico: ppconrado@yahoo.com.br.

1 Introdução

O presente projeto trata de uma proposta de sistema digital de “páginas amarelas” para o segmento de acampamentos. Um aplicativo Web para amantes da natureza, que apreciam a estadia em barracas, contato com animais, plantas e paisagens naturais. O plano contempla também a integração dos *players* do segmento: os proprietários de acampamentos, aventureiros, famílias, guias de turismo, lojas de equipamento e outros afins.

O mercado de acampamentos e de turismo de aventura pós-pandemia, tem aumentado significantemente. As pessoas estão procurando investir mais, em experiências de vida, em contato com a natureza, sair da rotina “estressante”, para estar ao ar livre com seus familiares e amigos.

“...De acordo com levantamento realizado pela plataforma Hoteis.com, 53% dos viajantes pretendem priorizar passeios com a família e 46% desejam destinos ao ar livre para evitar aglomerações, como cachoeiras e praias. Outro ponto interessante é que 70% dos entrevistados afirmaram que vão passar a valorizar o hábito de viajar e ambicionam aumentar a frequência, principalmente as internas, após o momento de crise. Essas informações corroboram o pensamento do professor da Universidade Victoria de Wellington (Nova Zelândia) e especialista cenários futuros de viagens, *Ian Yeoman...*” (METROPOLES, 2022).

“...O impacto ocasionado pela Covid-19 apresenta um novo normal, um novo mundo, uma nova perspectiva. O turismo e os negócios estão mudando e isso continuará pelos próximos anos. O foco das viagens será voltado mais na simplicidade, na economia e para unir as pessoas...” (METROPOLES, 2022).

“...*Ian Yeoman*, especialista em planejamento de cenários futuros e perspectivas de viagens e professor da Universidade Victoria de Wellington (Nova Zelândia). O relato do especialista foi dado durante a conferência virtual sobre os cenários para o turismo global pós-pandemia, organizada pela Associação Brasileira das Operadoras de Turismo (Braztoa), entidade que representa 90% das viagens de lazer vendidas no Brasil...” (METROPOLES, 2022).

Tendências do turismo

“...O professor Ian Yeoman afirmou que o turismo irá se recuperar, mas dependerá de como o cenário deste ano terminará. “Em qualquer estratégia de recuperação do turismo deve-se analisar os sinais de segurança e os planos de viagem dos locais”, conta. Ele ressaltou, ainda, que a principal procura será mesmo as viagens dentro do mesmo país. O futurologista pontuou possíveis tendências para os próximos anos:

- Prioridade nas viagens familiares.
- Viagens curtas e locais próximos.
- Viagens simples, sem ostentação.
- Procura por viagens com espaços abertos, como natureza e turismo rural.
- Mais atenção às normas de higiene das hospedagens...” (METROPOLES, 2022).

Com a demanda crescente, proprietários de acampamentos se depararam com a dificuldade de divulgação do seu negócio. Ao mesmo tempo, os usuários não conseguem informações, recomendações ou avaliações dos acampamentos existentes. O conceito e proposta do aplicativo pretende atender à demanda do mercado, fazendo a integração dos proprietários dos acampamentos, fornecedores e consumidores do segmento.

2 Viabilidade do projeto

O aplicativo não requer dispositivos ou sistemas avançados, ou de custo elevado. Os usuários do aplicativo precisam de um computador *desktop* ou dispositivo móvel e estar conectados à Internet. Eles podem elaborar a pesquisa da listagem dos acampamentos, sem a necessidade de estarem cadastrados. No entanto, para os proprietários dos acampamentos divulgarem seu negócio, assim como os campistas deixarem suas avaliações, será necessário estar registrado e autenticado (Figuras 15 e 16). Para registrar será solicitado: nome, *e-mail* e senha. A arquitetura do aplicativo será *monorepo*. Ou seja, *front-end* e *back-end* integrados. O *monorepo* utilizara o *framework* NodeJS (subseção 4.1.21). A parte do *back-end* utilizara a ferramenta Express (subseção 4.1.8) e banco de dados MongoDB. O banco de dados e o

monorepo serão instalados separados na nuvem. O gerenciamento das imagens, da geolocalização e dos mapas digitais, dos acampamentos foram delegados também a terceiros. Uma visão geral do funcionamento do aplicativo e fluxo dos dados entre os principais módulos, pode ser visto na Figura 8.1. Apresentamos o *Business Model Canvas* ou modelo de negócio a seguir (Figura1).

Figura 1 – Business Model Canvas - modelo de negócio

Parceiros • Fabricantes de equipamentos de acampamento. • Proprietários de acampamentos • Guias de aventura	Atividades • Entretenimento • Integração social Recursos • Time de desenvolvimento • Time de vendas e marketing	Proposição de Valor • Integração entre clientes, acampamentos, prestadores de serviços e fabricantes de equipamentos. • Facilitar para os usuários de acampamentos, a encontrar novos produtos e serviços.	Relacionamento com o Cliente • Envio de newsletter • Envio de promoções • Divulgação de eventos Canais • Divulgação em redes sociais do segmento • E-mails	Segmentos de Clientes • Aventureiros • Proprietários de acampamentos • Fabricantes de equipamento de acampamentos • Guias de aventura
Estrutura de Custos • Deploy do aplicativo no Heroku. • Deploy do banco de dados no MongoDB Atlas. • Serviço de gerenciamento de fotos Cloudinary.	 • Serviço de gerenciamento de mapas MapBox. • Marketing digital. • Time de desenvolvimento. • Time de marketing e vendas.	Fontes de Receita • Proprietários de acampamentos. • Fabricante de equipamentos. • Lojistas de vestuário e equipamentos para acampamento. • Guias de turismo de aventura. • Divulgação de produtos e serviços.		

Fonte: elaborado pelo autor.

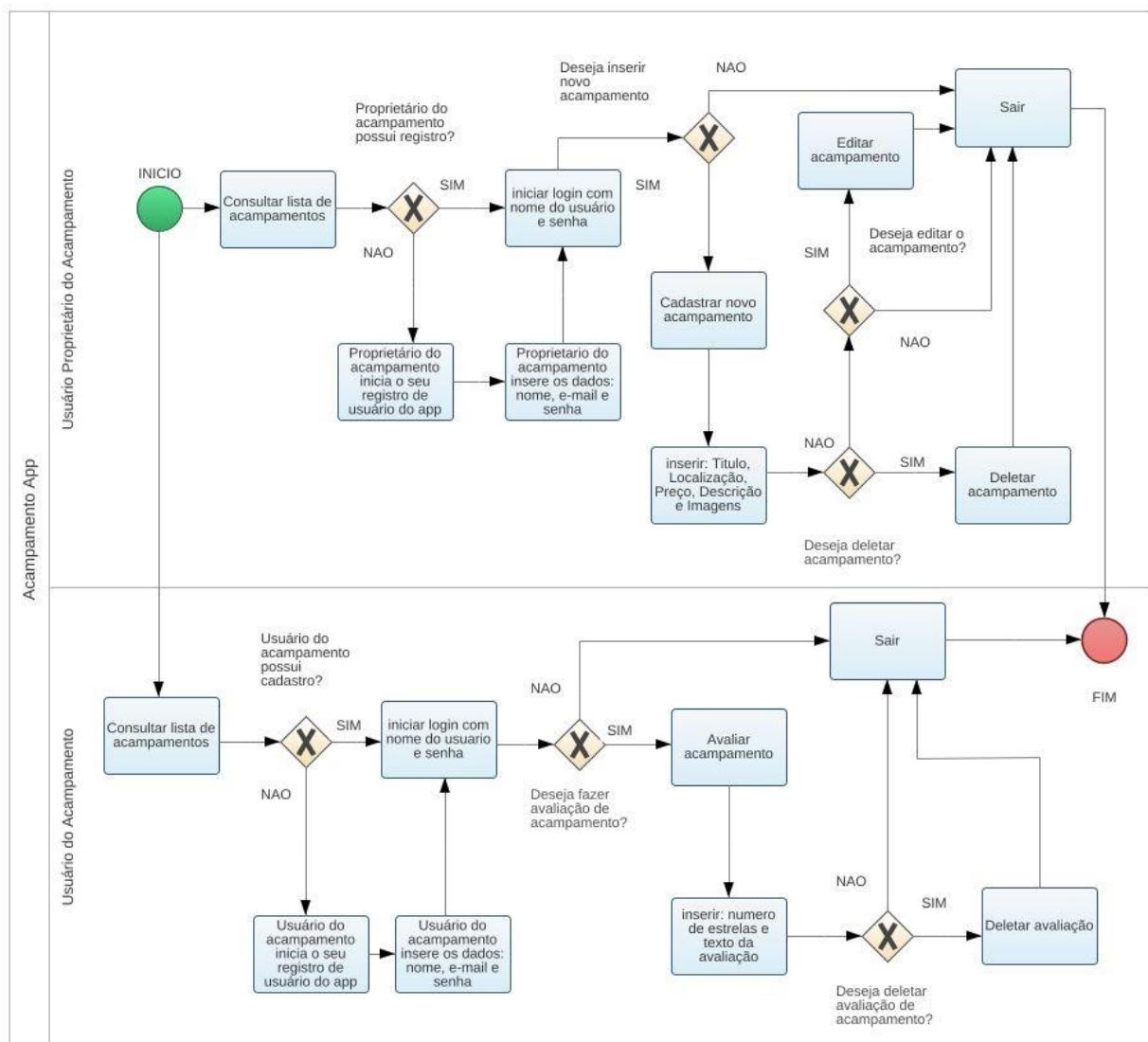
3 Levantamento de Requisitos

3.1 Elicitação e Especificação dos Requisitos

Os requisitos do sistema foram elencados por meio de pesquisas e observação do mercado “páginas amarelas” de acampamentos. O analista de sistemas efetuou pesquisas diárias durante uma semana, para observar os aplicativos existentes no mercado (*benchmark*).

3.2 BPMN

Figura 2 – BPMN - Business Process Model and Notation



Fonte: elaborado pelo autor.

3.3 Requisitos Funcionais

Quadro 1 – Requisitos funcionais do sistema

RF-001 - Mostrar lista de acampamentos disponíveis.	Categoria: <input type="checkbox"/> Oculto <input checked="" type="checkbox"/> Evidente	Prioridade: <input checked="" type="checkbox"/> Altíssima <input type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa
Descrição: O sistema deve disponibilizar a consulta pelo cliente da lista de acampamentos disponíveis.		
RF-002 - Registrar um novo usuário.	Categoria: <input type="checkbox"/> Oculto <input checked="" type="checkbox"/> Evidente	Prioridade: <input checked="" type="checkbox"/> Altíssima <input type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa
Descrição: O sistema deve disponibilizar registrar um novo usuário. Deverá obter os seguintes dados: nome do usuário, e-mail do usuário e senha do usuário.		
RF003 - Autenticar usuário.	Categoria: <input type="checkbox"/> Oculto <input checked="" type="checkbox"/> Evidente	Prioridade: <input checked="" type="checkbox"/> Altíssima <input type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa
Descrição: O sistema deve possibilitar autenticar o usuário solicitando: nome do usuário e senha.		
RF004 – Inserir novo acampamento.	Categoria: <input type="checkbox"/> Oculto <input checked="" type="checkbox"/> Evidente	Prioridade: <input checked="" type="checkbox"/> Altíssima <input type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa
Descrição: O sistema deve disponibilizar registrar um novo acampamento. Deverá obter os seguintes dados: título do acampamento, localização do acampamento, preço por noite no acampamento, descrição do acampamento e possibilitar a inserção de fotos do acampamento.		
RF005 – Avaliar acampamento.	Categoria: <input type="checkbox"/> Oculto <input checked="" type="checkbox"/> Evidente	Prioridade: <input checked="" type="checkbox"/> Altíssima <input type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa
Descrição: O sistema deve permitir o usuário avaliar um acampamento através do critério “Five Star” e um texto da avaliação.		
RF006 – Deletar acampamento.	Categoria: <input type="checkbox"/> Oculto <input checked="" type="checkbox"/> Evidente	Prioridade: <input checked="" type="checkbox"/> Altíssima <input type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa
Descrição: O sistema deve permitir o usuário remover um acampamento da listagem.		
RF007- Deletar avaliação.	Categoria: <input type="checkbox"/> Oculto <input checked="" type="checkbox"/> Evidente	Prioridade: <input checked="" type="checkbox"/> Altíssima <input type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa
Descrição: O sistema deve permitir o usuário remover uma avaliação da listagem.		
RF008 - Mostrar localização do acampamento no mapa.	Categoria: <input type="checkbox"/> Oculto <input checked="" type="checkbox"/> Evidente	Prioridade: <input checked="" type="checkbox"/> Altíssima <input type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa
Descrição: O sistema deve disponibilizar mapa geográfico mostrando a localização do acampamento. Deverá conter ferramenta de zoom para mostrar detalhes no mapa.		

RF009 – Mostrar lista de avaliações por acampamento.	Categoria: <input type="checkbox"/> Oculto <input checked="" type="checkbox"/> Evidente	Prioridade: <input checked="" type="checkbox"/> Altíssima <input type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa
Descrição: O sistema deve disponibilizar lista de avaliações do acampamento.		
RF010 – Editar acampamento.	Categoria: <input type="checkbox"/> Oculto <input checked="" type="checkbox"/> Evidente	Prioridade: <input checked="" type="checkbox"/> Altíssima <input type="checkbox"/> Alta <input type="checkbox"/> Média <input type="checkbox"/> Baixa
Descrição: O sistema deve disponibilizar formulário de edição do acampamento.		

3.4 Requisitos Não Funcionais

Quadro 2 – Requisitos Não Funcionais do Sistema

RNF001- Back-end.	O aplicativo deverá ser desenvolvido em arquitetura <i>monorepo</i> em NodeJs (NODEJS, 2022). Ou seja, <i>front-end</i> e <i>back-end</i> integrados em servidor NodeJs (subseção 4.1.21). Linguagem JavaScript. Este <i>monorepo</i> será instalado em um provedor na nuvem. O banco de dados deverá ser desenvolvido em NoSQL, que possui o conceito de coleções de documentos. O banco de dados deverá ser instalado em provedor na nuvem separado do aplicativo <i>monorepo</i> . O sistema <i>back-end</i> deverá ser desenvolvido em Node.Js e o banco de dados NoSQL MongoDB (MONGODB, 2022), para armazenar os dados contidos, nas requisições trocadas, com o aplicativo e outras APIs. O banco de dados deverá também gerenciar os registros e autenticações do usuário.	Tipo: Arquitetura, Infraestrutura	<input type="checkbox"/> Desejável <input checked="" type="checkbox"/> Obrigatório	<input checked="" type="checkbox"/> Permanente <input type="checkbox"/> Transitório
RNF002 - Banco de Dados.	O sistema do banco de dados deverá ser o NoSQL MongoDB (subseção 4.1.22), utilizando o conceito de coleções de documentos.	Tipo: Arquitetura, Infraestrutura	<input type="checkbox"/> Desejável <input checked="" type="checkbox"/> Obrigatório	<input checked="" type="checkbox"/> Permanente <input type="checkbox"/> Transitório

	O banco de dados MongoDB (MONGODB, 2022) deverá ser mapeado pela ferramenta ODM (<i>Object Data Model</i>) Mongoose (subseção 4.1.14) (MONGOOSE, 2022). Deverá ser instalado no serviço de nuvem MongoDB Atlas Cloud (MONGODB ATLAS, 2022), separado do aplicativo <i>monorepo</i> . Deverá armazenar os dados contidos, nas requisições trocadas com o aplicativo <i>monorepo</i> e outras APIs. Deverá também gerenciar os registros e autenticações dos usuários.			
RNF003 - <i>Front-end.</i>	O aplicativo deverá ser desenvolvido em arquitetura <i>monorepo</i> no NodeJs (NODEJS, 2022). Ou seja, <i>front-end</i> e <i>back-end</i> integrados no servidor NodeJs (subseção 4.1.21). O sistema <i>front-end</i> deverá ser desenvolvido com a ferramenta EJS (subseção 4.1.6) em Node.js. Linguagem JavaScript. O aplicativo será instalado em provedor na nuvem Heroku (HEROKU, 2022). O aplicativo enviará para o <i>browser</i> do usuário, as interfaces, “montadas” (HTML, CSS e JavaScript), pela ferramenta EJS (subseção 4.1.6) (EJS, 2022). As interfaces e os dados solicitados serão enviadas via HTTP para serem renderizadas no <i>browser</i> . Por outro lado, as interações e ações do usuário, nas interfaces no <i>browser</i> também serão enviadas de volta, via HTTP para o aplicativo.	Tipo: Arquitetura, infraestrutura	(<input type="checkbox"/>) Desejável (<input checked="" type="checkbox"/>) Obrigatório	(<input checked="" type="checkbox"/>) Permanente (<input type="checkbox"/>) Transitório

RNF004 – Gerenciamento das Imagens	As imagens serão gerenciadas pelo provedor Cloudinary (CLOUDINARY, 2022), na nuvem (subseção 4.1.2). O aplicativo <i>monorepo</i> envia as imagens inseridas pelo usuário e a API do serviço devolve as URLs das imagens. As URLs das imagens serão armazenadas posteriormente nas coleções do acampamento no banco de dados NoSQL.	Tipo: Produto, Infraestrutura		
RNF005 – Gerenciamento dos mapas	Os mapas serão gerenciados pelo provedor MapBox (subseção 4.1.1) na nuvem (@MPBOX/MAPBOX-SDK, 2022). O endereço do acampamento inserido pelo usuário será enviado a API do serviço e ele devolve o objeto GeoJSON (Figura 9.8), com as coordenadas da localização. O GeoJSON recebido será armazenado na coleção acampamento no banco de dados NoSQL. O serviço vai armazenar as coordenadas dos acampamentos e proverá os mapas a serem exibidos nas interfaces via EJS <i>templates</i> .	Tipo: Produto, Infraestrutura		
RNF006- Instalação do aplicativo <i>monorepo</i>.	O aplicativo <i>monorepo</i> deverá ser instalado no provedor de serviço, Heroku (HEROKU, 2022), na nuvem.	Tipo: Infraestrutura	() Desejável (X) Obrigatório	(X) Permanente () Transitório
RNF007- Sistema <i>online</i>.	O sistema deverá ser <i>online</i> .	Tipo: Usabilidade	() Desejável (X) Obrigatório	(X) Permanente () Transitório
RNF008- Sistema responsivo.	O sistema deverá ser responsivo.	Tipo: Usabilidade	() Desejável (X) Obrigatório	(X) Permanente () Transitório
RNF009- Funcionalidades do usuário.	O usuário cadastrado e autorizado poderá acessar todas as funcionalidades.	Tipo: Segurança	() Desejável (X) Obrigatório	(X) Permanente () Transitório

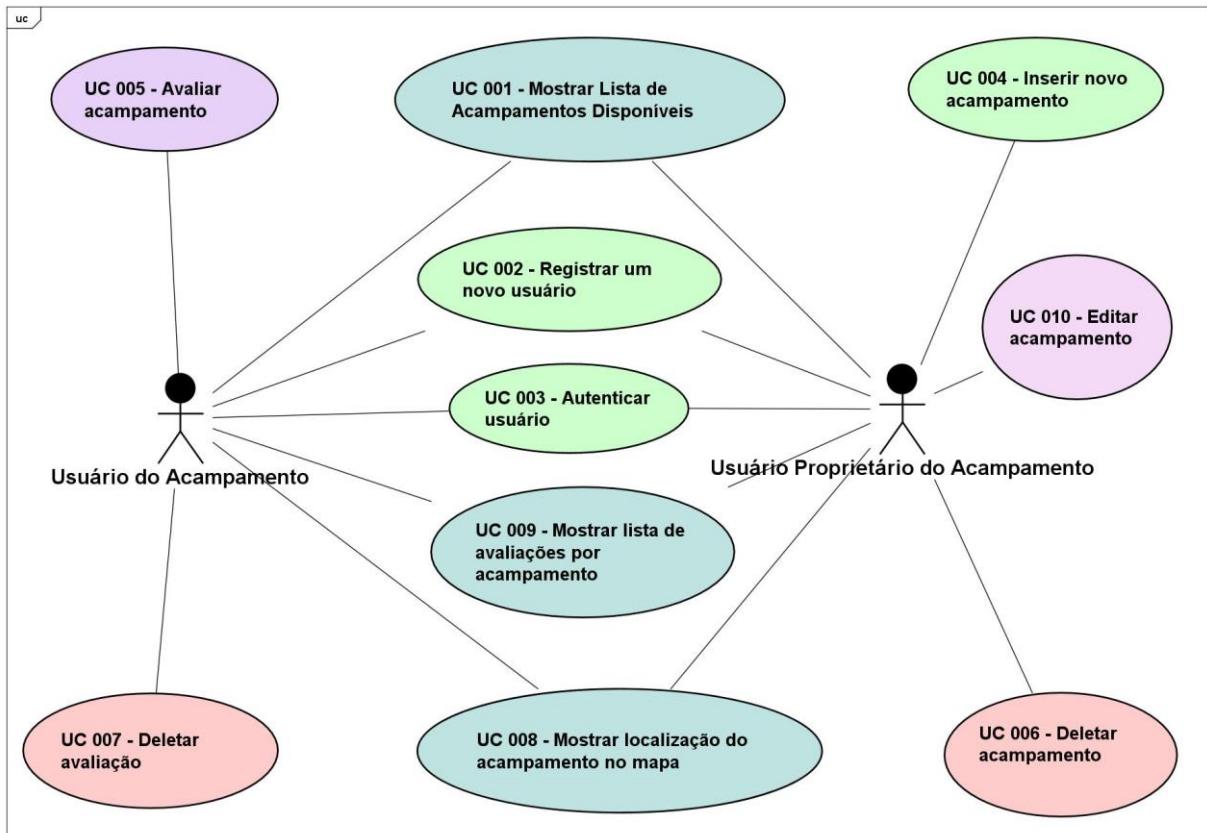
Quadro 3 – Matriz de Rastreabilidade RF x RNF

	RNF1	RNF2	RNF3	RNF4	RNF5	RNF6	RNF7	RNF8	RNF9
RF001							X	X	X
RF002							X	X	X
RF003							X	X	X
RF004							X	X	X
RF005							X	X	X
RF006							X	X	X
RF007							X	X	X
RF008							X	X	X
RF009							X	X	X
RF010							X	X	X

3.5 Casos de Uso

3.5.1 Diagrama de casos de uso

Figura 3 – Diagrama de casos de uso



Fonte: elaborado pelo autor.

3.5.2 Índice de casos de uso (Figura 3):

- UC 001. Mostrar lista de acampamentos disponíveis.
- UC 002. Registrar um novo usuário.
- UC 003. Autenticar usuário.
- UC 004. Inserir novo acampamento.
- UC 005. Avaliar acampamento.
- UC 006. Deletar acampamento.
- UC 007. Deletar avaliação.
- UC 008. Mostrar localização do acampamento no mapa.
- UC 009. Mostrar lista de avaliações por acampamento.
- UC 010. Editar acampamento.

Quadro 4 – Mostrar lista de acampamentos disponíveis.

Caso de Uso – Mostrar lista de acampamentos disponíveis.	
ID	UC 001
Descrição	Este caso de uso tem por objetivo apresentar uma listagem de acampamentos cadastrados pelos usuários.
Autor Primário	Usuário.
Pré-condição	O usuário deverá selecionar o link “ACAMPAMENTOS” localizado no cabeçalho.
Cenário Principal	<ol style="list-style-type: none"> O use case inicia quando o cliente seleciona o link “ACAMPAMENTOS” no cabeçalho da página (Figura 11). O sistema carrega a listagem de acampamentos e renderiza o <i>cluster map</i> na parte superior da interface (Figura 11). O sistema disponibiliza ao usuário trazer novos itens da lista através da barra de rolagem (Figura 11). O usuário escolhe o acampamento desejado, e deverá acionar o botão “VISITAR <nome do acampamento>”(Figura 11).
Pós-condição	Não possui.
Cenário Alternativo	Não possui.

Quadro 5 – Registrar um novo usuário.

Caso de Uso – Registrar um novo usuário.	
ID	UC 002
Descrição	Este caso de uso tem por objetivo registrar um novo usuário do aplicativo.
Autor Primário	Usuário.
Pré-condição	O usuário deverá selecionar o link “REGISTRAR” localizado no cabeçalho.
Cenário Principal	<ol style="list-style-type: none"> O usuário deverá selecionar o link “REGISTRAR” localizado no cabeçalho (Figura 11). O sistema apresenta o formulário de cadastro (Figura 15). O usuário digita o nome, e-mail e senha (Figura 15). O usuário seleciona o botão “REGISTRAR” para finalizar a operação (Figura 15).
Pós-condição	Todos os dados inseridos corretamente e validados.
Cenário Alternativo	Não possui.

Quadro 6 – Autenticar usuário.

Caso de Uso – Autenticar usuário.	
ID	UC 003
Descrição	Este caso de uso tem por objetivo validar a entrada do usuário no sistema.
Autor Primário	Usuário.
Pré-condição	O usuário deverá estar registrado no caso de uso UC 002.
Cenário Principal	<ol style="list-style-type: none"> O usuário deverá selecionar o link “LOGIN” localizado no cabeçalho (Figura 11). O sistema apresenta o formulário de <i>login</i> (Figura 16). O usuário insere o e-mail e senha (Figura 16). O cliente seleciona o botão “LOGIN” (Figura 16), para terminar a operação.
Pós-condição	Todos os dados inseridos corretamente e validados.
Cenário Alternativo	<ol style="list-style-type: none"> O sistema apresenta um <i>banner</i> vermelho na interface, contendo a mensagem “nome do usuário ou senha incorretos”, se os dados de usuário e

	<p>senha não estiverem compatíveis com os dados armazenados no banco de dados.</p> <p>6. Um novo formulário de autenticação será disponibilizado ao usuário (Figura 16).</p> <p>7. Fazer o registro de novo usuário, no caso de uso UC002, caso não esteja registrado (Figura 15).</p>
--	--

Quadro 7 – Inserir novo acampamento.

Caso de Uso – Inserir novo acampamento.	
ID	UC 004
Descrição	Este caso de uso tem por objetivo o usuário proprietário do acampamento registrar seu empreendimento na lista. O proprietário deverá inserir os dados do acampamento.
Autor Primário	Usuário proprietário do acampamento.
Pré-condição	O usuário deverá estar registrado no UC 002 e autenticado no UC 003.
Cenário Principal	<ol style="list-style-type: none"> 1. O usuário seleciona o botão “NOVO ACAMPAMENTO” localizado no cabeçalho (Figura 11). 2. O proprietário deverá inserir os seguintes dados: título (nome do acampamento), localização, preço por noite, descrição do acampamento e fotos do acampamento (Figura 17). 3. Ao concluir o preenchimento, o sistema solicita apertar o botão “ADICIONAR ACAMPAMENTO” (Figura 17).
Pós-condição	Todos os dados inseridos corretamente e validados.
Cenário Alternativo	<ol style="list-style-type: none"> 4. O sistema mostra mensagem em <i>banner</i> vermelho “Você precisa estar logado!”, caso o usuário não esteja autenticado através do caso de uso UC 003.

Quadro 8 – Avaliar acampamento.

Caso de Uso – Avaliar acampamento.	
ID	UC 005
Descrição	Este caso de uso tem por objetivo, o usuário do acampamento deixar sua avaliação.
Autor Primário	Usuário do acampamento.
Pré-condição	O usuário deverá estar registrado no UC 002 e autenticado no UC 003.
Cenário Principal	<ol style="list-style-type: none"> 1. O caso de uso inicia quando o usuário seleciona o botão “VISITAR <nome do acampamento> para entrar e visualizar o acampamento a ser avaliado (Figura 12). 2. O sistema apresenta os detalhes do acampamento (Figura 18). 3. A página do acampamento disponibiliza ao usuário a avaliação do tipo “Five star” para deixar uma nota de 1 a 5 (5 estrelas) (Figura 18). 4. O sistema disponibiliza uma área para deixar o texto da avaliação, denominada “TEXTO DA AVALIAÇÃO” (Figura 18). 5. Ao término da avaliação, o usuário deverá apertar o botão “ENVIAR” (Figura 20).
Pós-condição	Todos os dados inseridos corretamente e validados.
Cenário Alternativo	<ol style="list-style-type: none"> 6. Fazer autenticação no caso de uso UC003, caso não esteja autenticado (Figura 16).

Quadro 9 – Deletar acampamento.

Caso de Uso – Deletar acampamento.	
ID	UC 006
Descrição	Este caso de uso tem por objetivo remover um acampamento.
Autor Primário	Usuário proprietário do acampamento.
Pré-condição	O usuário deverá estar registrado no UC 002 e autenticado no UC 003.
Cenário Principal	<ol style="list-style-type: none"> O use case inicia quando o usuário seleciona o botão “VISITAR <nome do acampamento> para entrar e visualizar o acampamento a ser removido (Figura 11). O sistema apresenta o botão “REMOVER” (Figura 18). O usuário pressiona o botão “REMOVER” (Figura 18). O acampamento será removido da listagem de acampamentos.
Pós-condição	Nenhuma.
Cenário Alternativo	<ol style="list-style-type: none"> Fazer a autenticação no caso de uso UC003, caso não esteja autenticado (Figura 16).

Quadro 10 – Deletar avaliação.

Caso de Uso – Deletar avaliação.	
ID	UC 007
Descrição	Este caso de uso tem por objetivo remover avaliações elaboradas pelo usuário do acampamento.
Autor Primário	Usuário do acampamento.
Pré-condição	O usuário deverá estar registrado no UC 002 e autenticado no UC 003.
Cenário Principal	<ol style="list-style-type: none"> O usuário seleciona o botão “VISITAR <nome do acampamento>” localizado na lista de acampamentos (Figura 11). O usuário escolhe a avaliação a ser removida e aperta o botão “REMOVER” (Figura 18).
Pós-condição	O sistema apresenta um banner verde com a mensagem: “Avaliação removida com sucesso!”
Cenário Alternativo	Não possui.

Quadro 11 – Mostrar localização do acampamento no mapa.

Caso de Uso – Mostrar localização do acampamento no mapa.	
ID	UC 008
Descrição	Este caso de uso tem por objetivo mostrar em mapa a localização específica de um acampamento escolhido.
Autor Primário	Usuário.
Pré-condição	O usuário deverá selecionar o link “VISITAR <nome do acampamento> (Figura 11).
Cenário Principal	<ol style="list-style-type: none"> O use case inicia quando o usuário seleciona o acampamento no link “VISITAR <nome do acampamento> (Figura 11). O mapa será apresentado na página do acampamento escolhido (Figura 14).
Pós-condição	Não possui.
Cenário Alternativo	Não possui.

Quadro 12 – Mostrar lista de avaliações por acampamento.

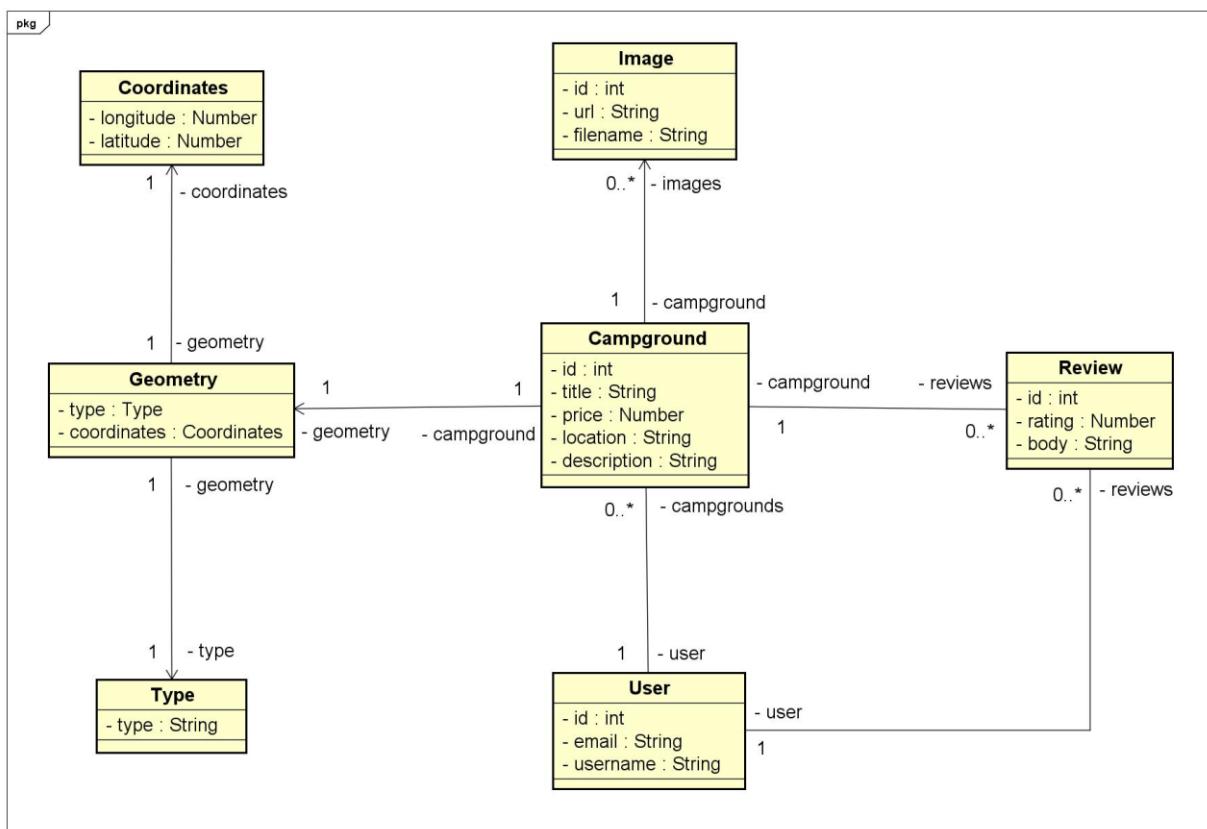
Caso de Uso – Mostrar lista de avaliações por acampamento.	
ID	UC 009
Descrição	Este caso de uso tem por objetivo mostrar as avaliações elaboradas pelos usuários de um determinado acampamento escolhido.
Autor Primário	Usuário
Pré-condição	O usuário deverá selecionar o botão “VISITAR <nome do acampamento> da listagem de acampamentos (Figura 11).
Cenário Principal	<ol style="list-style-type: none"> 1. O uso case inicia quando o usuário seleciona o acampamento no link “VISITAR <nome do acampamento> (Figura 11). 2. O sistema disponibiliza, a listagem das avaliações de todos os usuários de um determinado acampamento (Figura 20).
Pós-condição	Não possui.
Cenário Alternativo	Não possui.

Quadro 13 – Editar acampamento.

Caso de Uso – Editar acampamento.	
ID	UC 0010
Descrição	Este caso de uso tem por objetivo editar os dados de um acampamento existente.
Autor Primário	Usuário proprietário do acampamento.
Pré-condição	O usuário deverá estar registrado no UC 002 e autenticado no caso de uso UC003.
Cenário Principal	<ol style="list-style-type: none"> 1. O caso de uso inicia quando o usuário seleciona o acampamento no link “VISITAR <nome do acampamento> (Figura 11). 2. O sistema disponibiliza na página do acampamento escolhido, o botão “EDITAR” de acesso ao formulário de edição dos dados existentes (Figura 18). 3. O usuário seleciona o botão “EDITAR” (Figura 18). 4. O sistema disponibiliza o formulário de atualizar dados (Figura 19). 5. Editar os dados e apertar botão “ATUALIZAR ACAMPAMENTO”.
Pós-condição	Todos os dados inseridos corretamente e validados.
Cenário Alternativo	Não possui.

3.6 Diagrama de Classes

Figura 4 – Diagrama de classes



Fonte: elaborado pelo autor.

3.6.1 Objeto **campground** na coleção **campgrounds** no *MongoDB Atlas*

Figura 5 – Objeto *campground* na coleção *campgrounds*.

```
_id: ObjectId("628c7aa265d26b0016c01f98")
  geometry: Object
    type: "Point"
    coordinates: Array
      0: -46.8539
      1: -20.3442
  reviews: Array
    0: ObjectId("628c884aa18d4d0016d09a1b")
      1: ObjectId("628ec5b39362f600165a9bd7")
    title: "Trilhas de Minas Pousada Camping"
    location: "Delfinópolis"
    price: 100
    description: "Perto da Cachoeira"
  images: Array
    0: Object
      _id: ObjectId("628c89a0a18d4d0016d09a64")
      url: "https://res.cloudinary.com/ppconrado/image/upload/v1653377439/YelpCamp...)"
      filename: "YelpCamp/r3n33xvm1hl1ombxbattg"
    1: Object
    2: Object
  author: ObjectId("62725615a5d9ba14c4517cd7")
  __v: 7
```

Fonte: elaborado pelo autor.

3.6.2 Objeto **review** na coleção **reviews** no *MongoDB Atlas*

Figura 6 – Objeto *review* na coleção *reviews*.

```
_id: ObjectId("6276f4cc4bdc66001659dc2")
  rating: 5
  body: "Muito divertido "
  author: ObjectId("62725615a5d9ba14c4517cd7")
  __v: 0
```

Fonte: elaborado pelo autor.

3.6.3 Objeto **user** na coleção **users** no *MongoDB Atlas*

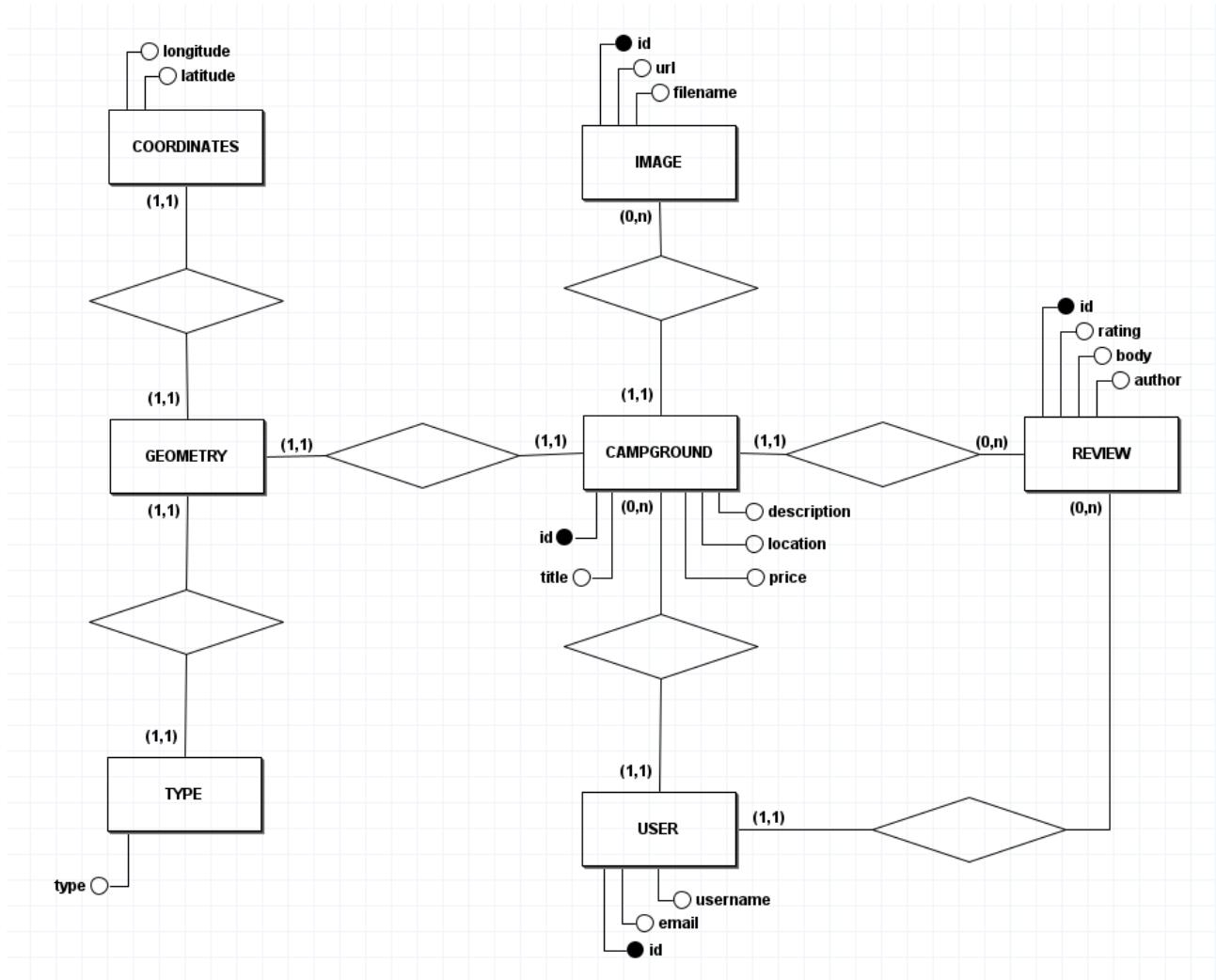
Figura 7 – Objeto *user* na coleção *users*.

```
_id: ObjectId("62725615a5d9ba14c4517cd7")
  email: "jose.conrado@fatec.sp.gov"
  username: "Jose Paulo Archetti Conrado"
  salt: "a92a991cceba0c6f9fad4038c2e23fe42d3acb523221eda7f3fdbfa28c656031"
  hash: "51f089c9d195d1e21ddaab3e94556e039da918eaa7d79a3b71ce3e75391824b2e41da1..."
  __v: 0
```

Fonte: elaborado pelo autor.

3.7 Diagrama Entidade-Relacionamento

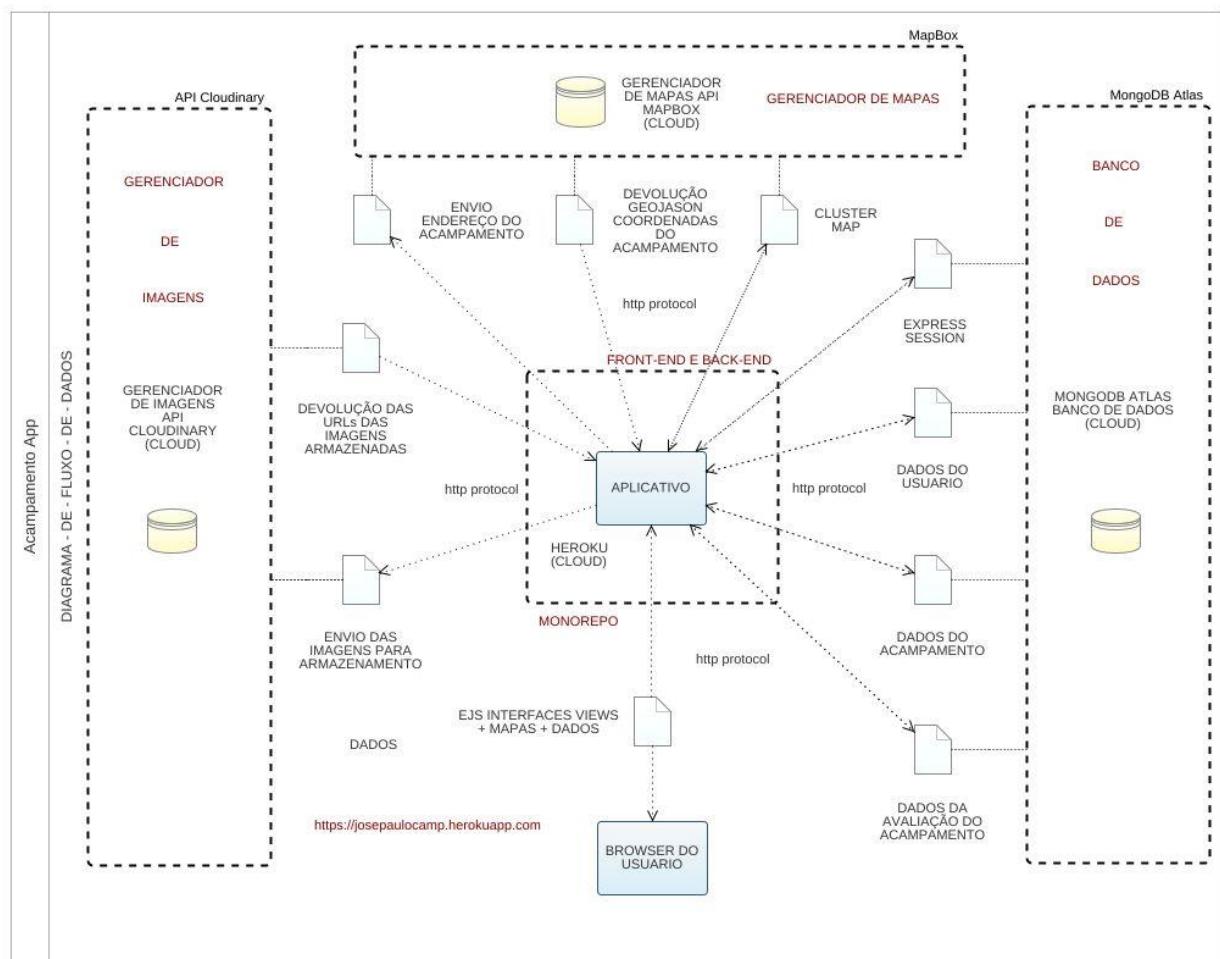
Figura 8 – Diagrama Entidade-Relacionamento



Fonte: elaborado pelo autor.

3.8 Diagrama de Fluxo de Dados

Figura 8.1 – Fluxo de dados do aplicativo (visão geral do projeto).



Fonte: elaborado pelo autor.

4 Ferramentas e Métodos de Desenvolvimento

4.1 Ferramentas

O arquivo **package.json** (figura 9) é um arquivo utilizado para especificar e configurar as dependências do projeto (ferramentas utilizadas) e *scripts* automatizados. Parte fundamental de um projeto em NodeJS (NODEJS, 2022).

Dependências (*dependencies*) são geralmente utilizadas para declarar os pacotes necessários, para executar o projeto em um ambiente de produção. Segue abaixo o detalhamento dos parâmetros do projeto no arquivo:

- *name*: O nome do aplicativo / projeto.
- *version*: a versão do aplicativo. A versão deve seguir as regras de controle de versão semântica.
- *description*: A descrição sobre o aplicativo, objetivo do aplicativo,
- *main*: Este é o ponto de entrada / partida do aplicativo. Ele especifica o arquivo principal do aplicativo que dispara quando o aplicativo é iniciado. O aplicativo pode ser iniciado usando *node start*.
- *scripts*: os scripts que precisam ser incluídos no aplicativo para funcionar corretamente.
- *keywords*: especifica o array de strings que caracteriza o aplicativo.
- *author*: Consiste nas informações sobre o autor, como nome, e-mail e outras informações relacionadas ao autor.
- *license*: A licença para a qual o aplicativo confirma são mencionadas neste par de valor-chave.
- *dependencies*: módulos de terceiros utilizados no projeto. Para definir as versões e atualizações dos pacotes. Utilizamos o gerenciador de pacotes npm.

Figura 9 – Arquivo /package.json - ferramentas do projeto

```
{  
  "name": "josepaulocamp",  
  "version": "1.0.0",  
  "description": "aplicacao web para camping",  
  "main": "app.js",  
  "scripts": {  
    "test": "echo \\"$Error: no test specified\\" && exit 1",  
    "start": "node app.js"  
  },  
  "keywords": [],  
  "author": "Jose Paulo Archetti Conrado",  
  "license": "ISC",  
  "dependencies": {  
    "@mapbox/mapbox-sdk": "^0.11.0",  
    "cloudinary": "^1.23.0",  
    "connect-flash": "^0.1.1",  
    "connect-mongo": "^3.2.0",  
    "dotenv": "^8.2.0",  
    "ejs": "^3.1.5",  
    "ejs-mate": "^3.0.0",  
    "express": "^4.17.1",  
    "express-mongo-sanitize": "^2.0.0",  
    "express-session": "^1.17.1",  
    "helmet": "^4.1.1",  
    "joi": "^17.2.1",  
    "method-override": "^3.0.0",  
    "mongoose": "^5.10.4",  
    "multer": "^1.4.2",  
    "multer-storage-cloudinary": "^4.0.0",  
    "passport": "^0.4.1",  
    "passport-local": "^1.0.0",  
    "passport-local-mongoose": "^6.0.1",  
    "sanitize-html": "^1.27.4"  
  }  
}
```

Fonte: elaborado pelo autor.

4.1.1 @mapbox/mapbox-sdk – API Mapbox

Ferramenta para interagir com API Mapbox (Figura 8.1). Responsável pelos mapas da geolocalização dos acampamentos. Converte dados geoespaciais em um conjunto de vetores de áreas (*tilesets*). Estes conjuntos de tilesets são armazenados nos servidores do Mapbox, para posterior apresentação na interface gráfica do aplicativo (Figuras 8.1, 9.9 e 9.10) (@MPBOX/MAPBOX-SDK, 2022). A utilização de *tilesets* reduz o tamanho dos dados a serem armazenados no banco de dados, convertendo gigabytes de dados geoespaciais em estado bruto em meros quilobytes.

Exige dois dados de entrada:

- *tileset source*: dados geoespaciais dos acampamentos; e
- *tileset recipe*: documento JSON que especifica as opções de configuração de conversão do *tileset source* em *vector tiles*.

4.1.2 Cloudinary

Esta ferramenta é uma solução de gerenciamento de imagens (Figura 8.1), para aplicações *website* e *mobile*. A solução carrega as fotos do acampamento, armazena, manipula e faz otimizações (Figura 9.7). As otimizações inteligentes de imagens foram transferidas para o Cloudinary sem a necessidade de instalar bibliotecas de tratamento complexas no aplicativo. A API do Cloudinary carrega, armazena e devolve as URLs das imagens para o navegador, as quais posteriormente serão armazenados (Figura 9.14), no banco de dados MongoDB Atlas. (CLOUDINARY, 2022).

4.1.3 Connect Flash

Ferramenta responsável pelos avisos e mensagens do sistema para o usuário. Ela apresenta um (*banner*) nas páginas (Figuras 17 e 20). As mensagens geradas são eliminadas depois de serem apresentadas. Aplicada nos *controllers* (Figuras 9.14, 9.15 e 9.16) e *middlewares* (Figura 9.17) (CONNECT-FLASH, 2022).

4.1.4 Connect Mongo

Ferramenta para armazenar uma Express Session (Subseção 4.1.10), no MongoDB. O padrão do armazenamento da ferramenta Express Session é na memória. Configuramos para armazenar no banco de dados. Configuramos *lazy session update* para evitar salvar a cada atualização da página. Um novo armazenamento é realizado a cada 24 horas (Figura 9.24). Passado o período de 14 dias, a Express Session será removida do banco de dados automaticamente (Figuras 8.1 e 9.21) (CONNECT-MONGO, 2022).

4.1.5 Dotenv (.env)

Ferramenta para carregar variáveis de ambiente do arquivo *.env*, armazenando as configurações de produção separadas do código (Figura 9.2) (DOTENV, 2022).

4.1.6 EJS

Ferramenta utilizada para produzir as interfaces do aplicativo (Figuras 9.10 a 9.20). EJS ou (*Embedded JavaScript Templating*) é um motor *templating* usado pelo NodeJs (subseção 4.1.21). O mecanismo de modelo ajuda a criar uma página HTML estilizada com código mínimo. Além disso, ele pode injetar dados no modelo HTML no lado do cliente e produzir o HTML final, que é o conceito do projeto. É uma linguagem de modelos (*templates*) para gerar páginas HTML com JavaScript integrado. A estilização das interfaces do aplicativo é provida pela biblioteca Bootstrap (BOOTSTRAP, 2022). Sua utilização é exemplificada na (Figura 9.18 e 9.24) (EJS, 2022).

4.1.7 EJS Mate

Ferramenta utilizada nas renderizações das interfaces EJS, para biblioteca *NodeJs express 4.x*. apresentada na (Subseção 4.1.6).

Método *layout(view)*

Quando o método *layout* é chamado dentro de um EJS *template* (modelagem da página – Figura 9.18), solicita que a saída deste arquivo passado com argumento do método *layout*, seja passada para a *view ou template*, como *body local - HTML*. Utilizada para especificar *layouts* dentro de uma EJS *template*, o que é recomendado para o Express 3.0 ou mais recente, uma vez que a funcionalidade de *layout* em nível de aplicativo foi removida. Utilizamos para chamar o arquivo de *layout* boilerplate.js (figura 9.18). Inicialização na (Figura 9.24) (EJS-MATE, 2022).

4.1.8 Express

Biblioteca para o Node.js, consistindo em um *framework* para o desenvolvimento de soluções Web. O Express oferece soluções para:

- Gerenciar requisições de diferentes verbos HTTP em diferentes URLs.
- Integrar *view engines* para inserir dados nos templates EJS.
- Definir as configurações comuns de uma aplicação Web, como a porta a ser usada para conexão e a elaboração dos modelos que são usados para renderizar a resposta.
- Adicionar novos processos intermediários na requisição, por meio de *middlewares* (Figura 9.17), em qualquer ponto da "fila". Utilização exemplificada nas rotas (Figuras 9.11, 9.12 e 9.13), nos *controllers* (Figuras 9.14, 9.15, 9.16) e no arquivo inicial app.js (Figuras 9.23, 9.24 e 9.25) (EXPRESS, 2022).

4.1.9 Express Mongoose Sanitize

Ferramenta que procura parâmetros chave (*key params*) em objetos que contenham os símbolos “\$” e “.”. Elabora a procura nos objetos: **req.body**, **req.query** ou **req.params**. Ao encontrar esses símbolos, remove-os e os dados associados a eles (Figura 9.24). Pode também trocar estes símbolos por outros permitidos. O MongoDB utiliza estes símbolos nos seus operadores. Sem está “limpeza” dos dados transportados, o aplicativo fica vulnerável a ataques de injeção de comandos de bancos de dados. *Hackers* podem enviar, por exemplo, objetos com o operador **\$where**, o que poderia alterar o banco de dados (EXPRESS-MONGOOSE-SANITIZE, 2022).

4.1.10 Express Session

Ferramenta para criação de uma *session middleware* para a biblioteca *Express* mencionada na Subseção 4.1.8. Os dados de um objeto de sessão são armazenados no banco de dados (Figura 8.1 e 9.21) por meio da ferramenta Connect Mongo (Subseção 4.1.4). O identificador da sessão é armazenado em um *cookie* do navegador (Figura 9.22) (EXPRESS-SESSION, 2022).

4.1.11 Helmet

Ferramenta que evita o vazamento de dados através de requisições HTTP. Protege o acesso de informações contidas no *header* das requisições (Figura 9.20) (HELMET, 2022).

4.1.12 Joi

Ferramenta para validação de dados baseados em *schemas*. Utilizado na validação de dados dos formulários de entrada (Figuras 9.6). Validamos através do modulo Joi os formulários de registro (Figura 15), de login (Figura 16), de criar novo acampamento (Figura 17), de avaliações (Figura 20) e de atualizar acampamentos (Figura 19) (JOI, 2022).

4.1.13 Method Override

Ferramenta que possibilita a utilização dos verbos HTTP, como PUT ou DELETE, em situações não suportadas pelo cliente HTTP. Necessário nas interfaces EJS *views forms*, para criar, remover e atualizar dados (Figura 9.24) (METHOD-OVERRIDE, 2022).

4.1.14 Mongoose

O banco de dados escolhido, MongoDB, exige escrever códigos de validações e lógicas de negócios. O Mongoose é uma biblioteca de modelagem de dados de objeto (*Object Data Model - ODM*) para o banco de dados NoSQL orientado a

documentos. Facilita o gerenciamento de relacionamentos de dados e fornece validação de esquema para objetos do MongoDB. Foi utilizado para modelar os dados e conectar a um banco de dados do MongoDB (Figura 9.24) (MONGOOSE, 2022).

4.1.15 Multer

Ferramenta para o Node.js para fazer *uploads* de arquivos. A biblioteca possibilita que as requisições com dados em formato JSON, através do *body-parse*, aceite os dados do formulário e possibilite o transporte das imagens do navegador para o servidor. Sem o Multer, ao enviar as imagens através do *request body - object (body)*, este estará vazio.

O Multer foi instalado na rota do acampamento (Figura 9.11). A API do Cloudinary (subseção 4.1.2) carrega, armazena e devolve as URLs das imagens para o navegador (Figura 8.1), que posteriormente serão armazenados no banco de dados (Figura 5) (MULTER, 2022).

4.1.16 Multer Storage Cloudinary

Ferramenta para integrar o aplicativo com ao serviço Cloudinary (Figura 9.14). Foram criadas as variáveis de ambiente no arquivo .env (Figura 9.2) para integrar o navegador com a API do Cloudinary (MULTER-STORAGE-CLOUDINARY, 2022):

- cloud_name: process.env.CLOUDINARY_CLOUD_NAME
- api_key: process.env.CLOUDINARY_KEY
- api_secret: process.env.CLOUDINARY_SECRET

4.1.17 Passport

Passport é uma ferramenta de autenticação para aplicações utilizada como um *middleware* do Express (PASSPORT, 2022). Escolhemos a estratégia de autenticação do tipo local que requer os parâmetros: *username* e *password*. Para utilizar esta estratégia precisamos adicionar também a ferramenta Passport-Local (subseção 4.1.18). As ferramentas serão utilizadas na submissão do formulário de *Login*.

4.1.18 Passport-Local

Ferramenta utilizada em conjunto com a Passport (subseção 4.1.17). Escolhemos a estratégia de autenticação do tipo local que requer os parâmetros: *username* e *password*. Para configurar esta estratégia no Passport precisamos da ferramenta Passport-Local (PASSPORT-LOCAL, 2022). As ferramentas serão utilizadas na submissão do formulário de Login.

4.1.19 Passport Local Mongoose

Ferramenta *plugin* (Figura 25), para biblioteca Mongoose (subseção 4.1.14). Utilizada para simplificar a autenticação através de *username* e *password* em conjunção com as ferramentas apresentadas nas (subseções 4.1.17 e 4.1.18) (PASSPORT-LOCAL-MONGOOSE, 2022).

4.1.20 Sanitize HTML

Ferramenta para especificar e permitir, quais *tags* HTML e atributos poderão ser utilizados. Utilizado em conjunção com a ferramenta de validação Joi, (subseção 4.1.12). Especificamos no projeto para restringir qualquer *tag* HTML (Figura 9.6) (SANITIZE-HTML, 2022).

4.1.21 NodeJS

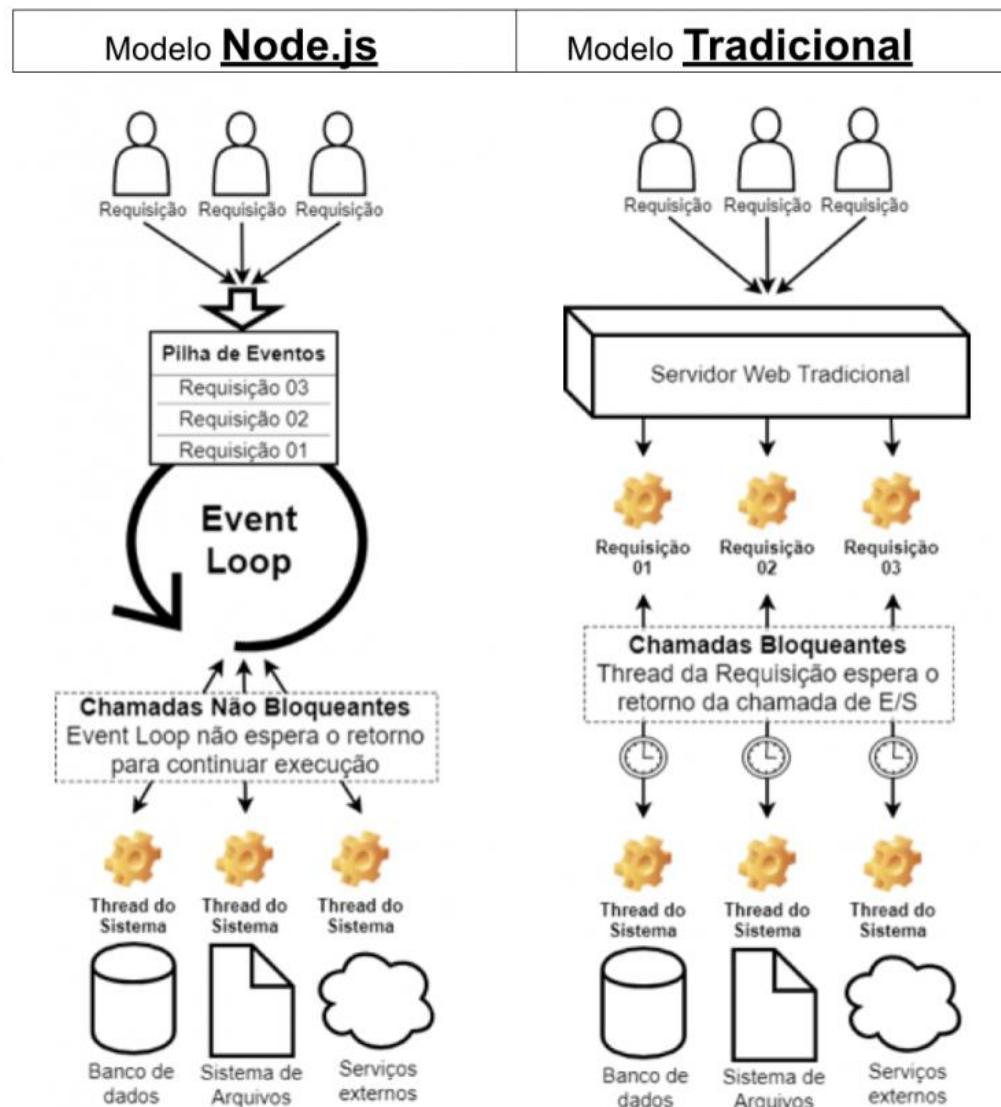
O NodeJS pode ser definido como um ambiente de execução servidor JavaScript. Isso significa que com o NodeJS (NODEJS, 2022) é possível criar aplicações JavaScript para “rodar” como uma aplicação única em uma máquina, não dependendo de um *browser* para a execução, como estamos acostumados.

No NodeJS, apenas um *thread* é responsável por tratar as requisições. Esse *thread* é chamado de *Event Loop*, e leva esse nome pois cada requisição é tratada como um evento. O *Event Loop* fica em execução esperando novos eventos para tratar, e para cada requisição, um novo evento é criado. Apesar de ser *single-threaded*, é possível tratar requisições concorrentes em um servidor NodeJS.

Enquanto o servidor tradicional utiliza o sistema *multi-thread* para tratar requisições concorrentes, o NodeJS consegue o mesmo efeito através de chamadas de E/S (entrada e saída) não-bloqueantes. Isso significa que as operações de entrada e saída (exemplo: acesso a banco de dados e leitura de arquivos do sistema) são assíncronas e não bloqueiam a *thread*. Diferentemente dos servidores tradicionais, a *thread* não fica esperando que essas operações sejam concluídas para continuar sua execução (NODEJS, 2022).

A figura abaixo (Figura 9.1) demonstra a diferença de funcionamento de um servidor web tradicional e um Node.JS:

Figura 9.1 – Diferença entre o servidor NodeJs e Servidor tradicional.



Fonte: banco de imagens internet.

4.1.22 MongoDB

O MongoDB é um banco de dados orientado a documentos que possui código aberto (*open source*) e foi projetado para armazenar uma grande quantidade de dados, além de permitir que se trabalhe de forma eficiente com grandes volumes. Ele é categorizado no banco de dados NoSQL (*not only SQL*) pois o armazenamento e a recuperação de dados no MongoDB, não são feitos no formato de tabelas.

O MongoDB é um servidor de banco de dados, onde as informações são armazenadas, mas o processo é mais fluido, independente, com os elementos tendo identificações únicas. Fornece um servidor de nuvem também (MONGODB ATLAS, 2022), para instalação de vários bancos de dados.

Os itens não são relacionados, obrigatoriamente, e sua hierarquia é totalmente flexível. Por conta de seu banco de dados NoSQL, as informações são armazenadas nas coleções e documentos. As coleções são subpartes do banco geral, independentes.

O MongoDB contém coleções, assim como o banco de dados MySQL contém tabelas. A sua vantagem é a permissão para criar vários bancos de dados e várias coleções dentro do principal.

Na coleção, encontramos os documentos que contêm os dados que vamos armazenar no banco do MongoDB, e uma única coleção pode conter vários documentos. Não existe esquema de tipo, isso significa que não é necessário que um documento seja semelhante ao outro.

Nos documentos, pode-se armazenar dados aninhados. Essa conexão de dados permite criar relações complexas entre eles e armazená-los no mesmo documento, o que torna o trabalho e a busca mais eficientes em comparação com o SQL (MONGODB, 2022). Além disso, podemos citar:

- Fornece grande flexibilidade para os campos nos documentos;
- Trabalha com dados heterogêneos;
- Não requer nenhuma adição ou injeção de SQL;
- Facilidade de criação do esquema através da ferramenta ODM Mongoose (subseção 4.1.14).

4.2 Métodos de Desenvolvimento

O aplicativo foi desenvolvido na arquitetura *monorepo* (*mono repository*), ou seja, *front-end* e *back-end* integrados (Figura 8.1). O *monorepo* foi desenvolvido no *framework* do servidor NodeJs (subseção 4.1.21) na linguagem JavaScript. Para o gerenciamento das requisições e rotas, utilizamos a ferramenta Express (subseção 4.1.8).

Para as interfaces apresentadas no *browser* do usuário (Figuras 10 a 20) utilizamos as ferramentas EJS (subseção 4.1.6) e EJS Mate (subseção 4.1.7). As interfaces são “montadas” no aplicativo *monorepo*, contendo as *markdown tags* **HTML**, estilizações **CSS** e logica **JavaScript** juntas. Elas são enviadas via requisições HTTP para o *browser* do usuário (Figura 8.1). As interações e alterações dos dados, pelo usuário, nas interfaces serão enviadas de volta, para gerenciamento pelo *monorepo*.

Desenvolvemos o banco de dados NoSQL MongoDB (subseção 4.1.22), para o armazenamento dos dados dos acampamentos, usuários e avaliações. Também armazenamos as Express Sessions (subseção 4.1.10), que controlam as autenticações dos usuários. O banco de dados será instalado, separado da aplicação *monorepo*, no provedor MongoDB Atlas (MONGODB ATLAS, 2022).

Utilizamos um gerenciador de imagens na nuvem, Cloudinary (subseção 4.1.2) e o gerenciador de mapas na nuvem, Mapbox (subseção 4.1.1).

O Diagrama de Fluxo de Dados, na subseção 3.8, Figura 8.1, mostra as transações de dados, entre os módulos principais do projeto, descritos acima.

4.2.1 Arquitetura Monorepo

A abordagem monorepo utiliza um único repositório para hospedar todo o código das múltiplas bibliotecas ou serviços que compõem o projeto. Hospedar toda a base de código em um único repositório oferece os seguintes benefícios:

Barreira de Entrada Reduzida

Quando os novos desenvolvedores começam a trabalhar para uma empresa, eles precisam baixar o código e instalar as ferramentas necessárias para começar a

trabalhar em suas tarefas. Suponha que o projeto esteja disperso por muitos repositórios, cada um tendo suas instruções de instalação e ferramentas necessárias. Nesse caso, a instalação inicial será complexa e, na maioria das vezes, a documentação não estará completa, exigindo que esses novos membros da equipe procurem os colegas para obter ajuda. Um monorepo simplifica as coisas. Como há um único local contendo todo o código e documentação, você pode agilizar a configuração inicial.

Gestão de Códigos de Centralizada

Ter um único repositório dá visibilidade de todo o código a todos os desenvolvedores. Ele simplifica o gerenciamento do código, uma vez que podemos usar um único rastreador de problemas para observar todos os problemas ao longo do ciclo de vida do aplicativo. Por exemplo, estas características são valiosas quando um problema abrange duas (ou mais) bibliotecas infantis com o bug existente na biblioteca dependente. Com vários repositórios, pode ser um desafio encontrar o pedaço de código onde o problema acontece. Além disso, precisaríamos descobrir qual repositório usar para recriar o problema e, em seguida, convidar e cruzar os membros de outras equipes para ajudar a resolver o problema. Com um monorepo, porém, tanto a localização de problemas de código como a colaboração para solucionar problemas tornam-se mais simples de alcançar.

Refatorações

Ao criar uma “refatoração” do código no aplicativo, várias bibliotecas serão afetadas. Se estivermos hospedando-as através de múltiplos repositórios, gerenciar todos os diferentes pedidos *pull* para mantê-las sincronizadas umas com as outras pode se revelar um desafio. Um *monorepo* facilita a realização de todas as modificações em todos os códigos para todas as bibliotecas e o envio sob um único pedido *pull*.

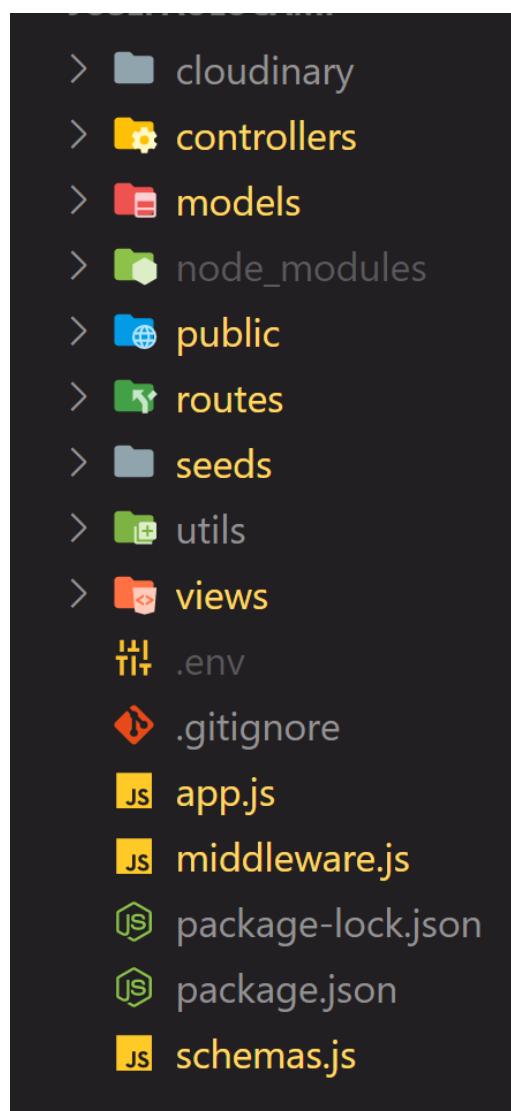
Funcionalidade Adjacente Preservada

Com o monorepo, podemos configurar todos os testes para que todas as bibliotecas funcionem sempre que uma única biblioteca for modificada. Como resultado, a probabilidade de fazer uma alteração em algumas bibliotecas minimizou os efeitos adversos em outras bibliotecas.

4.2.2 Estrutura do projeto NodeJs (pacotes, pastas e arquivos do projeto).

Apresentamos a estrutura do projeto NodeJs *monorepo*, através da disposição dos pacotes, pastas e arquivos. Mostrados na figura abaixo (Figura 9.2).

Figura 9.2 – Arquivo - / (root) - pastas e arquivos do projeto.



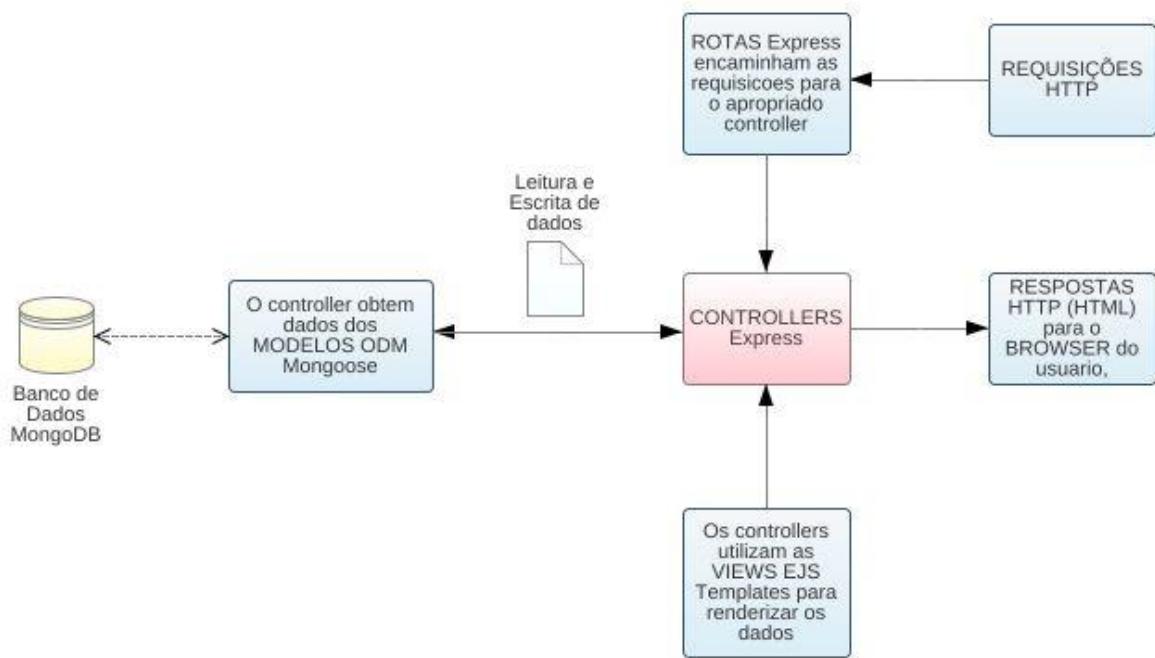
Fonte: elaborado pelo autor.

4.2.3 Servidor NodeJs e Express (*back-end*)

Foi utilizado o Express (subseção 4.1.8), como a principal biblioteca do NodeJS (subseção 4.1.21 e Figura 9.1). Esta ferramenta vai auxiliar nas autenticações, nas rotas (Figuras 9.11, 9.12 e 9.13) e *controllers* (Figuras 9.14, 9.15 e 9.16) das requisições para as APIs utilizadas (Figura 8.1).

No diagrama abaixo (Figura 9.2b) mostramos o fluxo de dados no aplicativo, e os módulos criados no projeto para atender as requisições HTTP. Podemos ver o modulo para visualizações, ou renderização, das páginas (EJS *views templates* – mais detalhes na subseção 4.2.12). Podemos ver também, a rota recebendo as requisições HTTP e encaminhando para o "controlador" (*controller*).

Figura 9.2b – Fluxo de dados e as interações dos componentes com o *controller* Express.



Fonte: elaborado pelo autor.

As rotas (Figuras 9.11, 9.12 e 9.13) encaminham as requisições e os dados contidos nas URLs para o controlador correto (mais detalhes na subseção 4.2.9). O *controller* (Figuras 9.14, 9.15 e 9.16) são métodos que processam as requisições e encaminham as respostas, após concluir a tarefa designada (mais detalhes na subseção 4.2.10). Para criar as páginas HTML que serão visualizadas pelo *browser*

do usuário, o controlador requisita os dados aos modelos Mongoose (subseção 4.1.14), que consequentemente coleta as informações do banco de dados (mais detalhes na subseção 4.2.5). A ferramenta EJS “monta” as páginas contendo (HTML, CSS e JavaScript) para o *controller* renderizar estes dados, no lado do cliente (mais detalhes na subseção 4.2.12).

4.2.4 Banco de Dados

Existem duas abordagens comuns para interagir com um banco de dados:

- Utilizando a linguagem de consulta nativa dos bancos de dados (por exemplo o SQL).
- Utilizando um modelo de dados de objeto ("ODM") ou um modelo relacional de objeto ("ORM").

Um ODM/ORM representa os dados do aplicativo como objetos JavaScript (JSON), que são então mapeados para o banco de dados subjacente.

O melhor desempenho pode ser obtido usando SQL ou qualquer que seja a linguagem de consulta suportado pelo banco de dados. Os ODM (Object Data Model) são muitas vezes mais lentos porque usam o código de tradução para mapear entre objetos e o formato do banco de dados, que podem não usar as consultas de banco de dados mais eficientes (isso é particularmente verdade se o ODM suporta diferentes *back-ends* de banco de dados e deve fazer maiores compromissos em termos de quais recursos de banco de dados são suportados).

O benefício de usar um ODM é que os programadores podem pensar em termos de objetos JavaScript em vez de semântica de banco de dados — isso é particularmente verdade se precisarmos trabalhar com diferentes bancos de dados. Eles também fornecem recursos para realizar a validação de dados. Decidimos utilizar a ODM Mongoose (subseções 4.1.14 e 4.2.5) para mapear o banco de dados NoSQL MongoDB.

O banco de dados NoSQL MongoDB (subseção 4.1.21) foi escolhido considerando a complexidade de elaborar códigos de operações CRUD, em banco de dados relacional. Devido à dificuldade de abstrair e transcrever os objetos desejados, em linhas de código. Com esses obstáculos técnicos, decidiu-se pela utilização de um banco de dados não relacional (NoSQL), que oferece os seguintes benefícios:

- Boa integração com o *framework* Node.js e Express (MERN Stack).
- Boa oferta de ferramentas e soluções para integração do NodeJs, MongoDB Atlas (banco de dados em nuvem - Figura 8.1), Mongoose, Cloudinary (gerenciamento de imagens em nuvem) e Express.
- Flexibilidade, facilidade na implementação da modelagem e validação dos dados dos objetos por meio de esquemas fornecidos pela ODM Mongoose
- Utilização do modelo de documentos, os quais nada mais são do que objetos JSON armazenados em coleções. Esse formato contribui na velocidade das operações para grandes coleções de dados.

O conceito de aplicação “páginas amarelas” apresenta potencial de rápido crescimento da massa de dados armazenada. Os dados dessa aplicação são majoritariamente coleções/listas de dados. Os dados da *express session* (Subseção 4.1.10), dos acampamentos, dos usuários e das avaliações são armazenados no serviço de banco de dados MongoDB Atlas (Figura 8.1). É o serviço de banco de dados em nuvem global para aplicativos (MONGODB ATLAS, 2022). Serviço de DBaaS (Banco de Dados como Serviço) oferecido pela MongoDB (subseção 4.1.21). Podemos usá-lo para armazenar nossos dados de aplicativo em seu servidor. Evitamos a utilização de servidor local devido ao custo de investimento.

Nesta fase do projeto são desejáveis a flexibilidade e o baixo investimento nas soluções implementadas, como o gerenciamento e armazenamento de dados, mapas e imagens. A responsividade de criação e obtenção da listagem dos acampamentos não poderia ser comprometida.

4.2.5 Modelagem do Banco de Dados - ODM Mongoose

O Mongoose (subseção 4.1.14) é uma ferramenta de modelagem de objetos para o MongoDB e projetada para funcionar em um ambiente assíncrono, que é caso do servidor NodeJs (Figura 9.1) escolhido. O Mongoose (MONGOOSE, 2022) categorizada pela sigla ODM (*Object Data Model*) escolhida irá fazer a modelagem de cada classe (Figura 4) ou documento (objeto). Os *controllers* (Figura 9.2b) da ferramenta Express obtêm os dados de cada classe, no banco de dados, com o intermédio do Mongoose. Com a modelagem poderemos com facilidade em poucas linhas de código efetuar as operações de CRUD.

Os modelos são definidos usando a interface *Schema*. O *Schema* permite definir os campos armazenados em cada documento, juntamente com seus requisitos de validação e valores padrão. Além disso, podemos definir métodos estáticos e auxiliares de instância para facilitar o trabalho com as tipagem dos dados, e também propriedades virtuais que podemos usar como qualquer outro campo, mas que não são realmente armazenados no banco de dados (discutiremos um pouco mais abaixo).

Os esquemas (*schemas*) são então "compilados" em modelos usando o método **mongoose.model()**. Uma vez obtido um modelo, podemos usá-lo para buscar, criar, atualizar e excluir objetos de determinada tipagem.

Cada modelo é mapeado para uma coleção de documentos no banco de dados do MongoDB. Os documentos conterão os campos/tipos de esquema definidos no **model.Schema**.

Os modelos são criados a partir de esquemas usando o método, **mongoose.model()**.

Define o esquema (exemplo para **NossoModelo**):

```
const Schema = mongoose.Schema;

const NossoModeloSchema = new Schema ({
    a_string: String,
    a_date: Date,
});
```

Compila o modelo do esquema.

```
const NossoModelo = mongoose.model("NossoModelo", NossoModelo
Schema);
```

O primeiro argumento é o nome singular da coleção que será criada para o modelo. Mongoose (subseção 4.1.14) criará a coleta de banco de dados para o

modelo `NossoModelo`, no exemplo acima), e o segundo argumento é o esquema que desejamos usar na criação do modelo.

Uma vez definida as classes dos modelos, podemos criar, atualizar ou excluir registros e executar consultas para obter todos os registros ou subconjuntos específicos de registros.

Tipagem dos esquemas (campos)

Um esquema pode ter um número arbitrário de campos — cada um representa um campo nos documentos armazenados em MongoDB.

```
const schema = new Schema ({

    name: String,
    binary: Buffer,
    living: Boolean,
    updated: {type: Date, default: Date.now() },
    age: {type: Number, min: 18, max: 65, required: true },
    mixed: Schema.Types.Mixed,
    _somId: Schema.Types.ObjectId,
    array: [ ],
    ofString: [String],
    nested: { stuff: { type: String, lowercase: true, trim: true } },

});
```

A maioria das tipagens (*types*) dos esquemas (*SchemaTypes*), mostrados no exemplo acima são conhecidas. As exceções são:

- **ObjectId**: Representa instâncias específicas de um modelo no banco de dados. Por exemplo, um livro pode usá-lo para representar seu objeto autor. Isso conterá, na verdade, o ID exclusivo () para o objeto especificado. Podemos usar o método para puxar as informações associadas quando necessário. `_idpopulate()`.
- **Mixed**: Um tipo de esquema arbitrário.

- []: Um array de itens. Podemos executar os métodos de arrays do JavaScript nesses modelos (*push*, *pop*, *unshift* etc.).

O código acima também mostra as duas formas de declarar um campo:

- Nome do campo e tipo (como um par de valor-chave) (ou seja, como feito com campos *name*, *binary* e *living*).
- Nome do campo seguido por um objeto que define o tipo (*type*), e quaisquer outras opções para o campo, como: valores padrão, validadores incorporados (por exemplo - valores max/min) e funções de validação personalizadas, se o campo é necessário e se os campos devem ser automaticamente definidos como minúsculas, maiúsculas ou os espaços eliminados (*trim*).

Propriedades virtuais

Propriedades virtuais são propriedades de documentos que podemos obter e definir, mas que não são persistidos ao MongoDB. Os *getters* são úteis para formatar ou combinar campos, enquanto os *setters* são úteis para descompor um único valor em vários valores para armazenamento. Utilizada no modelo do acampamento (Figura 9.5), para formatar as URLs das imagens e para obter os *pop-ups* (*link* do acampamento), no mapa de aglomerados (Figura 13).

Métodos e auxiliares de consulta

Um esquema também pode ter métodos de instância, métodos estáticos e auxiliares de consulta. Os métodos de instância e estática são semelhantes, mas com a diferença óbvia de que um método de instância está associado a um registro específico e tem acesso ao objeto atual. Os auxiliares de consulta permitem que ampliemos a API do construtor de consultas (por exemplo, permitindo que você adicione uma consulta "byName" além do e dos métodos). *find()*, *findOne()* e *findById()*.

Usando modelos

Depois de criar um esquema, podemos usá-los para criar os modelos. O modelo representa uma coleção de documentos no banco de dados que podemos pesquisar, enquanto as instâncias do modelo representam documentos individuais que podemos salvar e recuperar.

Criação e modificação de documentos

Para criar um registro, podemos definir uma instância do modelo e, em seguida, chamar. No exemplo acima assumem que **NossoModelo** é um modelo (com um único "nome" de campo) que criamos a partir do nosso esquema.

Modelagem

Os modelos foram baseados no diagrama de Entidade-Relacionamento (Figura 8) e Diagrama de Classes (Figura 4). Os objetos (documentos) armazenados nas (coleções) do banco de dados podem ser vistas para o acampamento (Figura 5), para a avaliação (Figura 6) e para usuário (Figura 7).

Esquemas do Mongoose

Os esquemas dos modelos do projeto foram alocados em três arquivos e apresentados a seguir: esquema do modelo usuário (Figura 9.3), esquema do modelo avaliação (Figura 9.4) e esquema do modelo acampamento (Figura 9.5).

Esquema do modelo do usuário (Figura 9.3).

Figura 9.3 – Arquivo - /models/user.js – modelo do usuário.

```
> [JS] user.js > [ej] UserSchema
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const passportLocalMongoose = require('passport-local-mongoose');

const UserSchema = new Schema({
  email: {
    type: String,
    required: true,
    unique: true
  }
});

UserSchema.plugin(passportLocalMongoose);

module.exports = mongoose.model('User', UserSchema);
```

Fonte: elaborado pelo autor.

Esquema do modelo da avaliação (Figura 9.4).

Figura 9.4 – Arquivo - /models/review.js – modelo da avaliação.

```
> JS review.js > [🔗] reviewSchema > 🔑 author
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

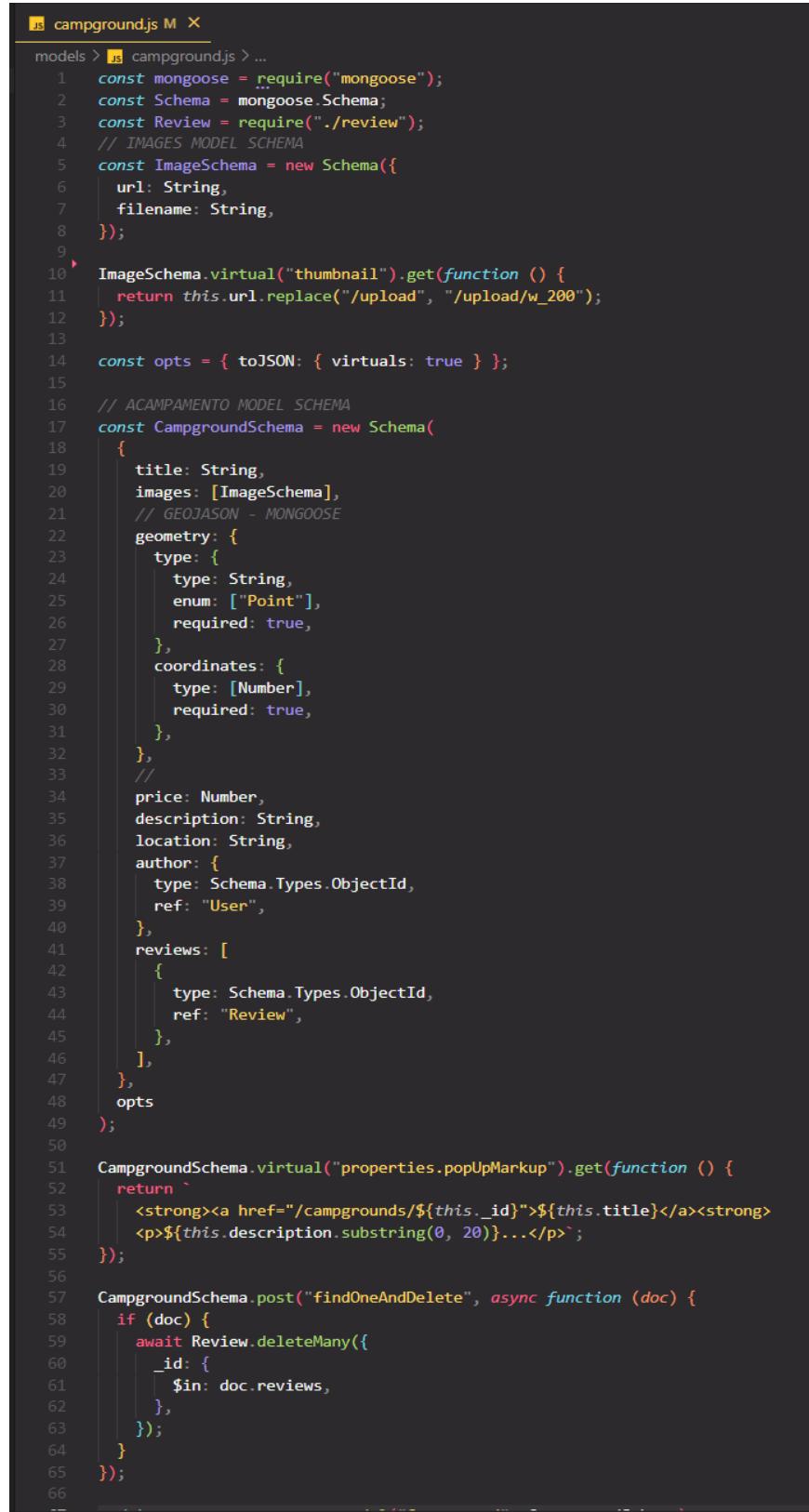
const reviewSchema = new Schema({
  body: String,
  rating: Number,
  author: {
    type: Schema.Types.ObjectId,
    ref: 'User'
  }
});

module.exports = mongoose.model("Review", reviewSchema);
```

Fonte: elaborado pelo autor.

Esquema do modelo do acampamento (Figura 9.5).

Figura 9.5 – Arquivo - /models/campground.js – modelo do acampamento.



```

1  const mongoose = require("mongoose");
2  const Schema = mongoose.Schema;
3  const Review = require("./review");
4  // IMAGES MODEL SCHEMA
5  const ImageSchema = new Schema({
6    url: String,
7    filename: String,
8  });
9
10 ImageSchema.virtual("thumbnail").get(function () {
11   return this.url.replace("/upload", "/upload/w_200");
12 });
13
14 const opts = { toJSON: { virtuals: true } };
15
16 // ACAMPAMENTO MODEL SCHEMA
17 const CampgroundSchema = new Schema(
18 {
19   title: String,
20   images: [ImageSchema],
21   // GEOJSON - MONGOOSE
22   geometry: {
23     type: {
24       type: String,
25       enum: ["Point"],
26       required: true,
27     },
28     coordinates: {
29       type: [Number],
30       required: true,
31     },
32   },
33   //
34   price: Number,
35   description: String,
36   location: String,
37   author: {
38     type: Schema.Types.ObjectId,
39     ref: "User",
40   },
41   reviews: [
42     {
43       type: Schema.Types.ObjectId,
44       ref: "Review",
45     },
46   ],
47 },
48 opts
49 );
50
51 CampgroundSchema.virtual("properties.popUpMarkup").get(function () {
52   return `
53     <strong><a href="/campgrounds/${this._id}">${this.title}</a><strong>
54     <p>${this.description.substring(0, 20)}...</p>
55   `);
56
57 CampgroundSchema.post("findOneAndDelete", async function (doc) {
58   if (doc) {
59     await Review.deleteMany({
60       _id: {
61         $in: doc.reviews,
62       },
63     });
64   }
65 });
66

```

Fonte: elaborado pelo autor

4.2.6 Validação de dados de entrada do formulário - ferramenta Joi.

Os dados inseridos nos formulários são validados pela ferramenta Joi (subseção 4.1.12). Podemos ver na Figura 9.6 os campos de cada formulário e os critérios de validação. Neste arquivo utilizamos também a ferramenta Sanitize HTML (subseção 4.1.20), para restringir qualquer *tag* HTML.

Figura 9.6 – Arquivo - /schemas.js – validação dos formulários (ferramenta - Joi).

```

js schemas.js > ...
  4   const extension = (joi) => ({
  5     type: "string",
  6     base: joi.string(),
  7     messages: {
  8       "string.escapeHTML": "{{#label}} must not include HTML!",
  9     },
10     rules: {
11       escapeHTML: {
12         validate(value, helpers) {
13           const clean = sanitizeHtml(value, {
14             allowedTags: [],
15             allowedAttributes: {},
16           });
17           if (clean !== value)
18             return helpers.error("string.escapeHTML", { value });
19           return clean;
20         },
21       },
22     },
23   });
24
25   const Joi = BaseJoi.extend(extension);
26
27 // VALIDACAO DOS DADOS DE ENTRADA DO ACAMPAMENTO
28
29 module.exports.campgroundSchema = Joi.object({
30   campground: Joi.object({
31     title: Joi.string().required().escapeHTML(),
32     price: Joi.number().required().min(0),
33     location: Joi.string().required().escapeHTML(),
34     description: Joi.string().required().escapeHTML(),
35   }).required(),
36   deleteImages: Joi.array(),
37 });
38
39 // VALIDACAO DOS DADOS DE ENTRADA DAS AVALIACOES
40
41 module.exports.reviewSchema = Joi.object({
42   review: Joi.object({
43     rating: Joi.number().required().min(1).max(5),
44     body: Joi.string().required().escapeHTML(),
45   }).required(),
46 });
47

```

Fonte: elaborado pelo autor.

4.2.7 Gerenciamento de Imagens

Delegamos a ferramenta Cloudinary (Subseção 4.1.2), a funcionalidade de coletar, “tratar”, armazenar, excluir imagens do acampamento (Figura 8.1). Optou-se por armazenar as imagens dos acampamentos no serviço Cloudinary principalmente, para eliminar a instalação de algoritmos complexos de tratamento de imagens, padronização e mudança de formatos (Figura 9.7). Outra vantagem é a transferência do espaço de armazenamento das imagens para terceiros. A ferramenta Multer (Subseção 4.1.15) faz a análise das imagens, construindo dois objetos, um contendo texto (dados do formulário) e o outro, um vetor de imagens. Tais objetos são enviados através das requisições (Figura 8.1) para o servidor do Cloudinary (Subseção 4.1.2). A ferramenta Multer Storage Cloudinary (Subseção 4.1.16) envia as imagens para o servidor e recebe o retorno das URLs das imagens armazenadas (Figura 8.1).

Figura 9.7 – Arquivo - /cloudinary/index.js – configuração da ferramenta Cloudinary.

```
ary > js index.js > ...
const cloudinary = require('cloudinary').v2;
const { CloudinaryStorage } = require('multer-storage-cloudinary');

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_KEY,
  api_secret: process.env.CLOUDINARY_SECRET
});

const storage = new CloudinaryStorage({
  cloudinary,
  params: {
    folder: 'YelpCamp',
    allowedFormats: ['jpeg', 'png', 'jpg']
  }
});

module.exports = {
  cloudinary,
  storage
}
```

Fonte: elaborado pelo autor.

4.2.8 Mapas e Geolocalização

A ferramenta @mapbox/mapbox-sdk (Subseção 4.1.1) apresentou a solução para os desafios mostrados abaixo:

- Configurar e apresentar a localização dos acampamentos em mapas.
- Converter um endereço de uma localização geográfica (*string*) em latitude e longitude.
- Aumentar a responsividade do aplicativo devido ao fluxo de dados de imagens de mapas.

Escolhemos o serviço Mapbox (@MAPBOX/MAPBOX-SDK, 2022), para gerenciamento dos mapas (Figura 8.1). A plataforma oferece a API Mapbox Tiling Service (MTS), para renderização de mapas e a API Geocoding, para obter as coordenadas da localização do acampamento. O *tileset* (conjunto de fragmentos de áreas do mapa) gerado fica armazenado no servidor da Mapbox. O conceito de *tileset* é bastante útil para visualizar grandes quantidades de dados apresentados em um determinado mapa, já que o navegador tem limitação para carregar mais de 100 Mb de dados de uma única vez. O serviço traz flexibilidade na criação e *design* dos mapas, adicionando camadas (*layers*) de informações da região, de colorização e estilizações. Enviamos o endereço do acampamento e a API Mapbox devolve as coordenadas em objeto **GeoJSON** (Figura 9.8) para ser armazenado com as informações do acampamento.

A funcionalidade de apresentar a geolocalização dos acampamentos pode impactar na velocidade do aplicativo, devido ao fluxo de dados dos mapas, e na renderização deles. A solução desenvolvida foi de receber no *input*, de um *form* de um novo acampamento, uma *string* do endereço. O método **forwardGeocode**, utilizado no *controller createCampground* (Figura 9.14) envia a *string* do endereço inserido através do *end-point* abaixo (Figura 8.1):

https://api.mapbox.com/geocoding/v5/{endpoint}/{search_text}.json

A Geocoding API Mapbox recebe a *string* e devolve o objeto GeoJSON (Figura 9.8), que contém o *array* - coordinates: [longitude, latitude].

Figura 9.8 – Objeto geometry – tipo GeoJSON.

```
// GEOJSON - MONGOOSE
geometry: {
  type: {
    type: String,
    enum: ["Point"],
    required: true,
  },
  coordinates: {
    type: [Number],
    required: true,
  },
}
```

Fonte: elaborado pelo autor.

A API Mapbox Tiling Service (MTS) coleta todas as localizações dos acampamentos da listagem e renderiza-os em um mapa de aglomerados (Figura 13) indicando em um ponto, a localização de cada acampamento. Cada ponto no mapa foi determinado pela sua coordenada, longitude e latitude. Cada endereço inserido na inclusão de um novo acampamento será incluído neste mapa. O mapa de aglomerados (Figura 13) renderizado, com todas as localizações da lista de acampamentos será enviada ao aplicativo quando solicitamos a listagem dos acampamentos.

O serviço de renderização de mapas, e o de geolocalização (gerador de coordenadas) Mapbox reduziu espaço de armazenamento no banco de dados do aplicativo. O conceito apresentado acima acelera a velocidade na troca de dados e na renderização dos mapas, pelo *browser* (Figura 9.2).

Configuramos a seguir (Figuras 9.9 e 9.8), o mapa de aglomerados contido na interface de listagem (Figuras 11 e 13). Definimos a fonte de dados, as camadas de informações, o zoom e posição central inicial, estilizações etc.

Figura 9.9 – Arquivo - /public/javascripts/clusterMaps.js – configuração do mapa de aglomerados.

```
> javascripts > clusterMap.js > map.on("load") callback
mapboxgl.accessToken = mapToken;
// CRIAR NOVO MAPA CLUSTER
const map = new mapboxgl.Map({
  container: "cluster-map",
  style: "mapbox://styles/mapbox/light-v10",
  center: [-47.4039, -20.5352], // coordenadas do centro do mundo : FRANCA
  zoom: 3,
});

map.addControl(new mapboxgl.NavigationControl());

map.on("load", function () {
  // Add a new source from our GeoJSON data and
  // set the 'cluster' option to true. GL-JS will
  // add the point_count property to your source data.
  map.addSource("campgrounds", {
    type: "geojson",
    // Point to GeoJSON data. This example visualizes all M1.0+ earthquakes
    // from 12/22/15 to 1/21/16 as logged by USGS' Earthquake hazards program.
    data: campgrounds,
    cluster: true,
    clusterMaxZoom: 14, // Max zoom to cluster points on
    clusterRadius: 50, // Radius of each cluster when clustering points (defaults to 50)
  });

  map.addLayer({
    id: "clusters",
    type: "circle",
    source: "campgrounds",
    filter: ["has", "point_count"],
    paint: {
      // Use step expressions (https://docs.mapbox.com/mapbox-gl-js/style-spec/#expressions-step)
      // with three steps to implement three types of circles:
      // * Blue, 20px circles when point count is less than 100
      // * Yellow, 30px circles when point count is between 100 and 750
      // * Pink, 40px circles when point count is greater than or equal to 750
      "circle-color": [
        "step",
        ["get", "point_count"],
        "#00BCD4",
        10,
        "#2196F3",
        30,
        "#3F51B5",
      ],
      "circle-radius": ["step", ["get", "point_count"], 15, 10, 20, 30, 25],
    },
  });

  map.addLayer({
    id: "cluster-count",
    type: "symbol",
    source: "campgrounds",
    filter: ["has", "point_count"],
    layout: {
      "text-field": "{point_count_abbreviated}",
      "text-font": ["DIN Offc Pro Medium", "Arial Unicode MS Bold"],
      "text-size": 12,
    },
  });
});
```

Fonte: elaborado pelo autor

Figura 9.10 – Arquivo - /public/javascripts/clusterMaps.js – configuração do mapa de aglomerados.

```

javascrips > js clusterMap.js > map
});

map.addLayer({
  id: "unclustered-point",
  type: "circle",
  source: "campgrounds",
  filter: ["!", ["has", "point_count"]],
  paint: {
    "circle-color": "#11b4da",
    "circle-radius": 4,
    "circle-stroke-width": 1,
    "circle-stroke-color": "#fff",
  },
});

// inspect a cluster on click
map.on("click", "clusters", function (e) {
  const features = map.queryRenderedFeatures(e.point, {
    layers: ["clusters"],
  });
  const clusterId = features[0].properties.cluster_id;
  map
    .getSource("campgrounds")
    .getClusterExpansionZoom(clusterId, function (err, zoom) {
      if (err) return;

      map.easeTo({
        center: features[0].geometry.coordinates,
        zoom: zoom,
      });
    });
});

// When a click event occurs on a feature in
// the unclustered-point layer, open a popup at
// the location of the feature, with
// description HTML from its properties.
map.on("click", "unclustered-point", function (e) {
  const { popUpMarkup } = e.features[0].properties;
  const coordinates = e.features[0].geometry.coordinates.slice();

  // Ensure that if the map is zoomed out such that
  // multiple copies of the feature are visible, the
  // popup appears over the copy being pointed to.
  while (Math.abs(e.lngLat.lng - coordinates[0]) > 180) {
    coordinates[0] += e.lngLat.lng > coordinates[0] ? 360 : -360;
  }

  new mapboxgl.Popup().setLngLat(coordinates).setHTML(popUpMarkup).addTo(map);
});

map.on("mouseenter", "clusters", function () {
  map.getCanvas().style.cursor = "pointer";
});
map.on("mouseleave", "clusters", function () {
  map.getCanvas().style.cursor = "";
});
});

```

Fonte: elaborado pelo autor.

4.2.9 Rotas para os *controllers* Express

A rota é uma seção do código Express que associa um verbo HTTP (GET, PUT, DELETE, POST), um *end-point* da URL e uma função que é chamada para direcionar a requisição para o determinado *controller* (Figura 9.2b).

```
app.get("/", function (req, res) {  
    res.send("Hello World!");  
});
```

Criamos as rotas dos *controllers* da ferramenta Express (seção 4.1.8), referente as requisições em três arquivos: Rotas para o *controller* do acampamento (Figura 9.11), rotas para o *controller* do usuário (Figura 9.12) e rotas para o *controller* da avaliação (Figura 9.13). Os códigos apresentam comentários descrevendo cada rota.

Rotas para os *controllers* do acampamento (Figura 9.11).

Figura 9.11 – Arquivo - /routes/campgrounds.js – rotas para os *controllers* do acampamento.

```
routes > JS campgrounds.js > [o] router
  1  const express = require("express");
  2  const router = express.Router();
  3  const campgrounds = require("../controllers/campgrounds");
  4  const catchAsync = require("../utils/catchAsync");
  5  const { isLoggedIn, isAuthor, validateCampground } = require("../middleware");
  6  const multer = require("multer");
  7  const { storage } = require("../cloudinary");
  8  const upload = multer({ storage });
  9  const Campground = require("../models/campground");
 10
 11 // 1 - ROUTE ROOT "/" - GET e POST
 12 router
 13   .route("/")
 14   .get(catchAsync(campgrounds.index))
 15   .post(
 16     isLoggedIn,
 17     upload.array("image"),
 18     validateCampground,
 19     catchAsync(campgrounds.createCampground)
 20   );
 21
 22 // 2 - ROUTE "/new" - GET - (view new.ejs)
 23 router.get("/new", isLoggedIn, campgrounds.renderNewForm);
 24
 25 // 3 - ROUTE("/:id" - GET, PUT e DELETE
 26 router
 27   .route("/:id")
 28   .get(catchAsync(campgrounds.showCampground))
 29   .put(
 30     isLoggedIn,
 31     isAuthor,
 32     upload.array("image"),
 33     validateCampground,
 34     catchAsync(campgrounds.updateCampground)
 35   )
 36   .delete(isLoggedIn, isAuthor, catchAsync(campgrounds.deleteCampground));
 37
 38 // 4 - ROUTE "/:id/edit" - GET
 39 router.get(
 40   "/:id/edit",
 41   isLoggedIn,
 42   isAuthor,
 43   catchAsync(campgrounds.renderEditForm)
 44 );
 45
```

Fonte: elaborado pelo autor

Rotas para os *controllers* do usuário (Figura 9.12).

Figura 9.12 – Arquivo - /routes/users.js – rotas para os *controllers* do usuário.

```
routes > js users.js > ...
  1 const express = require("express");
  2 const router = express.Router();
  3 const passport = require("passport");
  4 const catchAsync = require("../utils/catchAsync");
  5 const User = require("../models/user");
  6 const users = require("../controllers/users");
  7
  8 // 1- ROUTE "/register" - GET e POST (view register.ejs) - REGISTRAR USUARIO
  9 router
 10   .route("/register")
 11     .get(users.renderRegister)
 12     .post(catchAsync(users.register));
 13
 14 // 2 - ROUTE "/Login" - GET e POST + PASSPORT" (view Login.ejs) - AUTENTICAR USUARIO
 15 router
 16   .route("/login")
 17     .get(users.renderLogin)
 18     .post(
 19       passport.authenticate("local", {
 20         failureFlash: true,
 21         failureRedirect: "/login",
 22       }),
 23       users.login
 24     );
 25
 26 // 3 - ROUTE "/Logout" - GET - (view home.ejs) - SAIR
 27 router.get("/logout", users.logout);
 28
 29 module.exports = router;
 30
```

Fonte: elaborado pelo autor.

Rotas para os *controllers* da avaliação (Figura 9.13).

Figura 9.13 – Arquivo - /routes/reviews.js – rotas para os *controllers* da avaliação.

```
routes > js reviews.js > ...
1  const express = require("express");
2  const router = express.Router({ mergeParams: true });
3  const { validateReview, isLoggedIn, isReviewAuthor } = require("../middleware");
4  const Campground = require("../models/campground");
5  const Review = require("../models/review");
6  const reviews = require("../controllers/reviews");
7  const ExpressError = require("../utils/ExpressError");
8  const catchAsync = require("../utils/catchAsync");
9
10 // 1 - ROUTE "/" - POST - CRIAR AVALIACAO
11 router.post("/", isLoggedIn, validateReview, catchAsync(reviews.createReview));
12
13 // 2 - ROUTE "/:reviewId" - REMOVER AVALIACAO
14 router.delete(
15   "/:reviewId",
16   isLoggedIn,
17   isReviewAuthor,
18   catchAsync(reviews.deleteReview)
19 );
20
21 module.exports = router;
22
```

Fonte: elaborado pelo autor.

4.2.10 Controllers Express

Criamos os *controllers* (Figura 9.2b) da ferramenta Express (seção 4.1.8), das requisições, também em três arquivos: *controllers* do acampamento (Figura 9.15), do usuário (Figura 9.16) e de avaliações (Figura 9.17). Os códigos apresentam comentários de sua funcionalidade, para cada *controller*.

Controllers do acampamento (Figura 9.14).

Figura 9.14 – Arquivo - /controllers/campgrounds.js – controllers do acampamento.

```
ers > campgrounds.js > ...
// CONTROLLER - VIEW DE NOVO ACAMPAMENTO
module.exports.renderNewForm = (req, res) => {
  res.render("campgrounds/new");
};

// CONTROLLER - CRIAR NOVO ACAMPAMENTO
module.exports.createCampground = async (req, res, next) => {
  const geoData = await geocoder
    .forwardGeocode({
      query: req.body.campground.location,
      limit: 1,
    })
    .send();
  const campground = new Campground(req.body.campground);
  campground.geometry = geoData.body.features[0].geometry;
  campground.images = req.files.map((f) => ({
    url: f.path,
    filename: f.filename,
  }));
  campground.author = req.user._id;
  await campground.save();
  console.log(campground);
  req.flash("success", "Novo Camping criado com sucesso!");
  res.redirect(`#/campgrounds/${campground._id}`);
};

// CONTROLLER - MOSTRAR ACAMPAMENTO
module.exports.showCampground = async (req, res) => {
  const campground = await Campground.findById(req.params.id)
    .populate({
      path: "reviews",
      populate: {
        path: "author",
      },
    })
    .populate("author");
  if (!campground) {
    req.flash("error", "Não foi possível encontrar o camping!");
    return res.redirect("/campgrounds");
  }
  res.render("campgrounds/show", { campground });
};

// CONTROLLER - EDITAR ACAMPAMENTO
module.exports.renderEditForm = async (req, res) => {
  const { id } = req.params;
  const campground = await Campground.findById(id);
  if (!campground) {
    req.flash("error", "Não foi possível encontrar este camping!");
    return res.redirect("/campgrounds");
  }
  res.render("campgrounds/edit", { campground });
};

// CONTROLLER - ATUALIZAR ACAMPAMENTO
module.exports.updateCampground = async (req, res) => {
  const { id } = req.params;
  console.log(req.body);
  const campground = await Campground.findByIdAndUpdate(id, {
    ...req.body.campground,
  });
  const imgs = req.files.map((f) => ({ url: f.path, filename: f.filename }));
  campground.images.push(...imgs);
  await campground.save();
  if (req.body.deleteImages) {
    for (let filename of req.body.deleteImages) {
      await cloudinary.uploader.destroy(filename);
    }
    await campground.updateOne({
      $pull: { images: { filename: { $in: req.body.deleteImages } } },
    });
  }
  req.flash("success", "O camping foi atualizado com sucesso!");
  res.redirect(`#/campgrounds/${campground._id}`);
};

// CONTROLLER - REMOVER ACAMPAMENTO
module.exports.deleteCampground = async (req, res) => {
  const { id } = req.params;
  await Campground.findByIdAndDelete(id);
  req.flash("success", "O camping foi removido com sucesso!");
  res.redirect("/campgrounds");
};
```

Fonte: elaborado pelo autor.

Controllers do usuário (Figura 9.15).

Figura 9.15 – Arquivo - /controllers/users.js – *controllers* do usuário.

```
lers > JS users.js > ...
const User = require("../models/user");
// CONTROLLER - VIEW DE ENTRADA DOS DADOS DE AUTORIZACAO
module.exports.renderRegister = (req, res) => {
  res.render("users/register");
};

// CONTROLLER - CRIAR USUARIO
module.exports.register = async (req, res, next) => {
  try {
    const { email, username, password } = req.body;
    const user = new User({ email, username });
    const registeredUser = await User.register(user, password);
    req.login(registeredUser, (err) => {
      if (err) return next(err);
      req.flash("success", "Bem Vindo ao Jose Paulo Camp!");
      res.redirect("/campgrounds");
    });
  } catch (e) {
    req.flash("error", e.message);
    res.redirect("register");
  }
};

// CONTROLLER - VIEW DO LOGIN DO USUARIO AO APLICATIVO
module.exports.renderLogin = (req, res) => {
  res.render("users/login");
};

// CONTROLLER - VIEW DE SUCESSO NA AUTORIZACAO DE ACESSO DO USUARIO
module.exports.login = (req, res) => {
  req.flash("success", "Bem vindo! Estamos felizes com seu retorno!");
  const redirectUrl = req.session.returnTo || "/campgrounds";
  delete req.session.returnTo;
  res.redirect(redirectUrl);
};

// CONTROLLER - VIEW DE SAIDA DO USUARIO DO APLICATIVO
module.exports.logout = (req, res) => {
  req.logout();
  // req.session.destroy();
  req.flash("success", "Até a próxima aventura!");
  res.redirect("/campgrounds");
};
```

Fonte: elaborado pelo autor.

Controlleres de avaliação (Figura 9.16).

Figura 9.16 – Arquivo - /controllers/reviews.js – controllers de avaliação.

```
ers > js reviews.js > ...
const Campground = require("../models/campground");
const Review = require("../models/review");
// CONTROLLER - CRIAR NOVA AVALIAÇÃO
module.exports.createReview = async (req, res) => {
  const campground = await Campground.findById(req.params.id);
  const review = new Review(req.body.review);
  review.author = req.user._id;
  campground.reviews.push(review);
  await review.save();
  await campground.save();
  req.flash("success", "Nova avaliação criada com sucesso!");
  res.redirect(`~/campgrounds/${campground._id}`);
};

// CONTROLLER - REMOVER AVALIAÇÃO
module.exports.deleteReview = async (req, res) => {
  const { id, reviewId } = req.params;
  await Campground.findByIdAndUpdate(id, { $pull: { reviews: reviewId } });
  await Review.findByIdAndDelete(reviewId);
  req.flash("success", "Avaliação removida com sucesso!");
  res.redirect(`~/campgrounds/${id}`);
};
```

Fonte: elaborado pelo autor.

4.2.11 Middlewares Express

O *Middleware* é usado extensivamente em aplicativos Express, para tarefas desde o serviço de arquivos estáticos até o tratamento de erros, até a compressão de respostas HTTP. Considerando que as funções de rota encerram o ciclo de solicitação-resposta HTTP retornando alguma resposta ao cliente via HTTP, as funções de *middleware* normalmente realizam alguma operação na solicitação ou resposta e, em seguida, chamam a próxima função na "pilha", que pode ser mais *middleware* ou um manipulador de rota. A ordem em que o *middleware* é chamado cabe ao desenvolvedor do aplicativo.

Algumas rotas do projeto passaram por métodos intermediários (*middlewares*), através do arquivo middleware.js (Figura 9.17). Criamos métodos (*controllers*) para o controle de segurança, para os usuários autenticados terem acesso ao privilégio de

utilizar funcionalidades. Os Métodos de segurança: autenticar usuário, permitir usuário editar dados do acampamento, permitir o autor das avaliações removê-las. Também foram adicionados neste arquivo as validações dos dados inseridos nos formulários. Métodos de validação: validação dos dados de entrada do acampamento e validação dos dados de entrada das avaliações (Figura 9.6). Os dois últimos utilizam a ferramenta Joi (subseção 4.1.12). Apresentamos a seguir (Figura 9.17), o código comentado com a funcionalidade, de cada função intermediaria ou *middleware*. Observe o método **next()**, no final da expressão. Este método dá seguimento as requisições, caso validadas e aprovadas.

Figura 9.17 – Arquivo - /middleware.js

```
// MIDDLEWARE - AUTENTICAÇÃO DO USUÁRIO
module.exports.isLoggedIn = (req, res, next) => {
  if (!req.isAuthenticated()) {
    req.session.returnTo = req.originalUrl;
    req.flash("error", "Você precisa estar logado!");
    return res.redirect("/login");
  }
  next();
};

// MIDDLEWARE - VALIDAÇÃO DOS DADOS DE ENTRADA DO ACAMPAMENTO (Ferramenta Joi)
module.exports.validateCampground = (req, res, next) => {
  const { error } = campgroundSchema.validate(req.body);
  console.log(req.body);
  if (error) {
    const msg = error.details.map((el) => el.message).join(",");
    throw new ExpressError(msg, 400);
  } else {
    next();
  }
};

// MIDDLEWARE - PERMISSÃO PARA O AUTOR DO ACAMPAMENTO EDITAR
module.exports.isAuthor = async (req, res, next) => {
  const { id } = req.params;
  const campground = await Campground.findById(id);
  if (!campground.author.equals(req.user._id)) {
    req.flash("error", "Você não tem permissão para fazer isto!");
    return res.redirect(`campgrounds/${id}`);
  }
  next();
};

// MIDDLEWARE - PERMISSÃO PARA O AUTOR DAS AVALIAÇÕES EDITAR
module.exports.isReviewAuthor = async (req, res, next) => {
  const { id, reviewId } = req.params;
  const review = await Review.findById(reviewId);
  if (!review.author.equals(req.user._id)) {
    req.flash("error", "Você não tem permissão para fazer isto!");
    return res.redirect(`campgrounds/${id}`);
  }
  next();
};

// MIDDLEWARE - VALIDAÇÃO DOS DADOS DE ENTRADA DAS AVALIAÇÕES (Ferramenta Joi)
module.exports.validateReview = (req, res, next) => {
  const { error } = reviewSchema.validate(req.body);
  if (error) {
    const msg = error.details.map((el) => el.message).join(",");
    throw new ExpressError(msg, 400);
  } else {
    next();
  }
};
```

Fonte: elaborado pelo autor.

4.2.12 Interfaces EJS Template

O EJS é uma ferramenta para gerar páginas da web que podem incluir dados dinâmicos e podem compartilhar peças modeladas com outras páginas da web (como cabeçalhos e rodapés). Não é uma estrutura de *front-end*. Embora o EJS possa ser usado pelo JavaScript do lado do cliente para gerar HTML, ele é normalmente usado no *back-end* para gerar páginas HTML (*views* do Express - subseção 4.1.8) em resposta a requisições de URLs. O EJS não é uma estrutura do lado do cliente (*client-side*), como React ou Angular e não determina qual *framework* do lado do cliente será utilizado. No nosso aplicativo, basta o usuário ter um *browser* para exibir as interfaces.

EJS utiliza modelos (*templates*). Define páginas HTML na sintaxe EJS e específica para onde os dados irão na página. A ferramenta combina dados com o modelo (*template*) e "renderiza" uma página HTML completa, onde o EJS coleta os dados no banco de dados e os insere na página da web de acordo com a forma que definimos no modelo (HTML, CSS e JavaScript).

No projeto utilizamos documentos (objetos) devido a escolha do banco de dados NoSQL. Podemos obter uma coleção de documentos de dados dinâmicos do banco de dados e o EJS gera uma listagem, de acordo com as regras de exibição do modelo (*template*). Economizamos código para gerar dinamicamente o HTML com a base em dados.

O EJS é compatível com o Express para uso no *back-end*, pois se conecta à arquitetura do mecanismo *view* que o Express (subseção 4.1.8) fornece e permite o aplicativo renderize páginas da web para o *browser client-side* através do método Express **res.render()**.

Existem dezenas de sistemas de modelos concorrentes para uso em JavaScript. O EJS foi escolhido com base em recursos que correspondem às nossas necessidades, como a linguagem de *layout* que facilita o desenvolvimento. O EJS permite gerar páginas HTML completas.

Estilização das Interfaces

As bibliotecas de estilos CSS, utilizados nas interfaces do aplicativo foram importadas via *tag <link>* e *<script>* do arquivo HTML (Figura 9.18). Bootstrap (BOOTSTRAP, 2022) foi utilizado na formatação de estilo das interfaces (Figuras 10

a 20). O gerenciador de mapas, Mapbox (@MPBOX/MAPBOX-SDK, 2022) forneceu os estilos do mapa de aglomerados (*cluster map*) (Figura 11) e o mapa de localização na página do acampamento (Figura 14). Bootstrap (BOOTSTRAP, 2022) foi utilizado na estilização do banner das mensagens ao usuário (Figuras 17 e 20), via Connect Flash (CONNECT-FLASH, 2022).

Figura 9.18 – Arquivo - /views/layouts/boilerplate.js – importação dos estilos CSS.

```
<title>Jose Paulo Camp</title>
<!-- Bootstrap -->
<link
  rel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/bootstrap/5.0.0-alpha1/css/bootstrap.min.css"
  integrity="sha384-r4NyP46KrjDleawBgD5tp8Y7UzmLA05oM1iAEQ17CSuDqnUK2+k9luXQ0FXJCJ4I"
  crossorigin="anonymous"
/>
<!-- BOOTSTRAP PLUGIN -->
<script src="https://cdn.jsdelivr.net/npm/bs-custom-file-input/dist/bs-custom-file-input.js"></script>
<!-- MAPBOX GL JS-->
<script src="https://api.mapbox.com/mapbox-gl-js/v1.12.0/mapbox-gl.js"></script>
<!-- MAPBOX CSS STYLE -->
<link
  href="https://api.mapbox.com/mapbox-gl-js/v1.12.0/mapbox-gl.css"
  rel="stylesheet"
/>
<!-- APP.JS MAPS DYSPLAY CSS -->
<link rel="stylesheet" href="/stylesheets/app.css" />
</head>

<!-- NAVBAR, FLASH MESSAGES, FOOTER -->
<body class="d-flex flex-column vh-100">
  <!-- NAVBAR -->
  <% include('../partials/navbar')%>
  <!-- FLASH MESSAGES-->
  <main class="container mt-5">
    <% include('../partials/flash')%> <% body %>
  </main>
  <!-- FOOTER -->
  <% include('../partials/footer')%>
  <!-- FLASH MESSAGES -->
  <script
    src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
    integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
    crossorigin="anonymous"
  ></script>
  <!-- Bootstrap -->
  <script
    src="https://stackpath.bootstrapcdn.com/bootstrap/5.0.0-alpha1/js/bootstrap.min.js"
    integrity="sha384-oesi62h0L fzrys4LxRF630JCxDxDipiYWBnvT19Y9/TRlw5x1KIEHpnvDShgf/"
    crossorigin="anonymous"
  ></script>
</body>
```

Fonte: elaborado pelo autor.

4.2.13 Segurança e Autenticação

Foi mandatório no projeto, a segurança ao aplicativo, assim como dos dados dos usuários. As ferramentas, Passport (subseção 4.1.17), Passport-Local (subseção 4.1.18) e Passport-Local-Mongoose (subseção 4.1.19) facilitaram a implementação da funcionalidade de autenticação do usuário e liberação de acesso a determinadas

rotas. O controle das rotas foi implementado no arquivo middleware.js (Figura 9.17). Ao registro de um novo usuário, a ferramenta Passport-Local-Mongoose converte a senha inserida em dois *strings*: **hash** e **salt**. O objeto *user* será armazenado no banco de dados, com os parâmetros: **e-mail**, **username**, **salt** e **hash** (Figura 7). Uma autenticação futura compara a senha inserida na página, *login* (Figura 16) com a versão única, **criptografada** contida no banco de dados (Figura 7). O algoritmo *salt* reforça a segurança das senhas de ataques cibernéticos. As ferramentas Sanitize-Html (Subseção 4.1.19) e Joi (Subseção 4.1.12), implementaram a segurança, de ataques tipo **cross-site scripting** (Figura 34). A ferramenta Helmet (Subseção 4.1.11), implementou a segurança das informações contidas nos **headers** das requisições (Figura 9.20).

Figura 9.20 – Arquivo - /app.js – configuração da ferramenta Helmet.

```
// HELMET configuration - protege ao acesso de dados das requisições
const fontSrcUrls = [];
app.use(
  helmet.contentSecurityPolicy({
    directives: {
      defaultSrc: [],
      connectSrc: ["'self'", ...connectSrcUrls],
      scriptSrc: ["'unsafe-inline'", "'self'", ...scriptSrcUrls],
      styleSrc: ["'self'", "'unsafe-inline'", ...styleSrcUrls],
      workerSrc: ["'self'", "blob:"],
      childSrc: ["blob:"],
      objectSrc: [],
      imgSrc: [
        "'self'",
        "blob:",
        "data:",
        "https://res.cloudinary.com/ppconrado/", // CONTA do CLOUDINARY!
        "https://images.unsplash.com",
      ],
      fontSrc: ["'self'", ...fontSrcUrls],
    },
  })
);
```

Fonte: elaborado pelo autor.

4.2.14 Comunicação ao Usuário

A implementação da ferramenta Connect-Flash (subseção 4.1.3), facilitou a comunicação, dos alertas e avisos das requisições (processada pelos métodos dos *controllers* – Figuras 9.14, 9.15 e 9.16), bem-sucedidas e das que apresentaram falhas. O método reproduz um *banner* (Figuras 17 e 20), nas interfaces do aplicativo e codificada em cores pelo tipo de aviso. Bem-sucedido: **verde** e as malsucedidas: **vermelho**. A mensagem pode ser fechada, se o usuário desejar

4.2.15 Express Sessions

A solução MERN, (MongoDB + Express + Connect Mongo), implementou o controle de acesso do aplicativo. As *sessions* são criadas (Figura 23) e armazenadas no banco de dados (Figura 8.1 e 9.21), após a autenticação via *login* (Figura 16). Os dados são recuperados, no futuro acesso do usuário.

Figura 9.21 – Express Session armazenada no MongoDB Atlas.

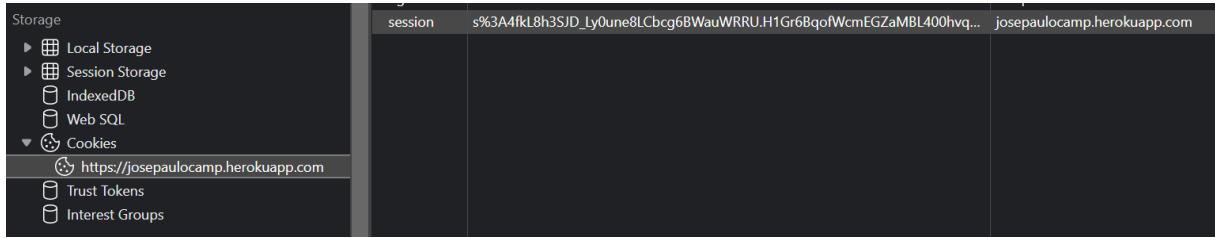
```
_id: "ofM6jmNwFh-w1AGS5qdZ_UGCKzUypLNy"
expires: 2022-10-18T01:02:39.483+00:00
lastModified: 2022-10-04T01:02:39.483+00:00
session: "{\"hmac\":\"d3d5066c554a97203410f978cc13b131bc569c4a79164e543133f9c09...\""}"
```

Fonte: elaborado pelo autor.

4.2.16 Cookies

O *browser* do usuário armazena, um **cookie** (Figura 9.22), recebido pela requisição HTTP, ao acessar pela primeira vez o aplicativo (Figura 9.21). O *cookie* converte requisições, *stateless* em *statefull*. O aplicativo identifica o usuário, pelo *cookie* que é gerenciado pela ferramenta Express-Session (Subseção 4.1.10).

Figura 9.22 – Cookie armazenado no browser do usuário.



Fonte: elaborado pelo autor.

4.2.17 Inicialização do aplicativo (*front-end e back-end*) (Figuras 9.23, 9.24 e 9.25).

Inicialização do MongoDB, do ODM Mongoose e criação das variáveis de conexão das bibliotecas (de uso global) Node.js, no arquivo inicial - app.js (Figura 9.23).

Figura 9.23 – Arquivo - /app.js – início do aplicativo.

```
js app.js M X
js app.js > ...
1  if (process.env.NODE_ENV !== "production") {
2  |   require("dotenv").config();
3  }
4  //
5  const express = require("express");
6  const path = require("path");
7  const mongoose = require("mongoose");
8  const ejsMate = require("ejs-mate");
9  const session = require("express-session");
10 const flash = require("connect-flash");
11 const ExpressError = require("./utils/ExpressError");
12 const methodOverride = require("method-override");
13 const passport = require("passport");
14 const LocalStrategy = require("passport-local");
15 const User = require("./models/user");
16 const helmet = require("helmet");
17 const mongoSanitize = require("express-mongo-sanitize");
18 const userRoutes = require("./routes/users");
19 const campgroundRoutes = require("./routes/campgrounds");
20 const reviewRoutes = require("./routes/reviews");
21
22 // CONEXAO PARA CRIACAO DE UMA EXPRESS SESSION (MERN)
23
24 const MongoDBStore = require("connect-mongo")(session);
25
26 // BANCO DE DADOS (dev e prod)
27
28 const dbUrl = process.env.DB_URL || "mongodb://localhost:27017/yelp-camp";
29
30 // CONECTANDO MONGOOSE
31
32 mongoose.connect(dbUrl, {
33   // Connect Mongoose e OPTIONS
34   useNewUrlParser: true,
35   useCreateIndex: true,
36   useUnifiedTopology: true,
37   useFindAndModify: false,
38 });
39
40 // Logica de confirmacao da conexao com o DB.
41
42 const db = mongoose.connection;
43 db.on("error", console.error.bind(console, "erro de conexao:"));
44 db.once("open", () => {
45   console.log("Banco de Dados conectado");
46 })
```

Fonte: elaborado pelo autor

Inicialização do servidor e EJS views. Criação e armazenamento da Express Session no banco de dados, de acordo com as configurações do objeto: **sessionConfig** (Figura 9.24).

Figura 9.24 – Arquivo - /app.js – Express Session e EJS views.

```
js app.js > ...
48 // EXPRESS
49
50 const app = express();
51
52 // EJS
53
54 app.engine("ejs", ejsMate);
55 app.set("view engine", "ejs");
56 app.set("views", path.join(__dirname, "views"));
57
58 app.use(express.urlencoded({ extended: true })); // express parse the body da requisicao
59 app.use(methodOverride("_method"));
60 app.use(express.static(path.join(__dirname, "public")));
61 app.use(
62   mongoSanitize({
63     replaceWith: "_",
64   })
65 );
66 const secret =
67   process.env.SECRET || "Cuidado com a exposicao da senha de acesso!";
68
69 const store = new MongoDBStore({
70   url: dbUrl,
71   secret,
72   touchAfter: 24 * 60 * 60,
73 });
74
75 // SESSION CONFIGURACAO
76
77 store.on("error", function (e) {
78   console.log("ERRO NO ARMAZENAMENTO DA SESSION no DB", e);
79 });
80
81 const sessionConfig = {
82   store,
83   name: "session",
84   secret,
85   resave: false,
86   saveUninitialized: true,
87   cookie: {
88     httpOnly: true,
89     // secure: true,
90     expires: Date.now() + 1000 * 60 * 60 * 24 * 7,
91     maxAge: 1000 * 60 * 60 * 24 * 7,
92   },
93 }
```

Fonte: elaborado pelo autor

Inicialização das rotas padrão do servidor (Figura 9.25).

Figura 9.25 – Arquivo - /app.js – inicialização das rotas.

```

js app.js > ...
152 |     res.locals.error = req.flash(error);
163 |     next();
164 });
165
166 // ROTAS express
167
168 app.use("/", userRoutes);
169 app.use("/campgrounds", campgroundRoutes);
170 app.use("/campgrounds/:id/reviews", reviewRoutes);
171
172 // home
173
174 app.get("/", (req, res) => {
175     res.render("home");
176 });
177
178 app.all("*", (req, res, next) => {
179     next(new ExpressError("Pagina não encontrada!", 404));
180 });
181
182 app.use((err, req, res, next) => {
183     const { statusCode } = err;
184     if (!err.message) err.message = "Oh No, Alguma coisa deu errado!";
185     res.status(statusCode).render("error", { err });
186 });
187
188 // API dev e prod - port config
189
190 const port = process.env.PORT || 3000;
191 app.listen(port, () => {
192     console.log(`Serviço disponível na PORTA ${port}`);
193 });
194

```

Fonte: elaborado pelo autor.

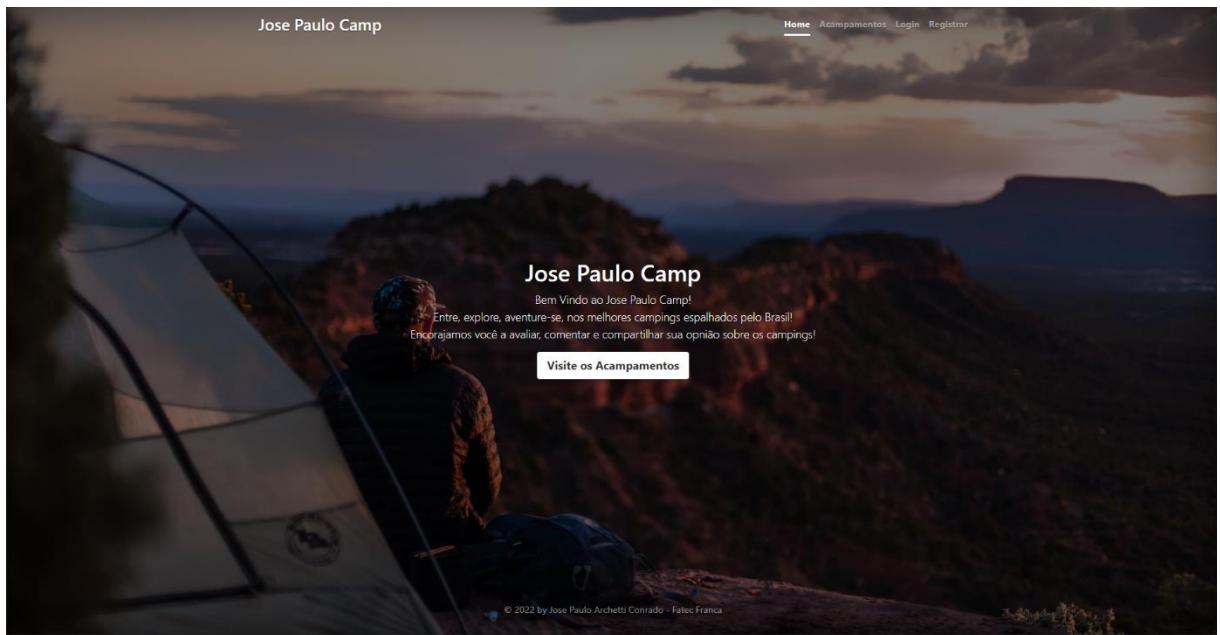
5 Resultados e Discussão

5.1 Interfaces EJS templates – *front-end* (Figuras 10 a 20).

5.1.1 Página inicial com *links* de navegação.

Iniciamos a aplicação pela interface *Home* (Figura 10). Na parte central encontramos o botão “Visite os Acampamentos” e no cabeçalho da página os *links* “Home”, “Acampamentos”, “Login” e “Registrar”. Ao clicar no botão, “Visite os Acampamentos” ou no *link* “Acampamentos” no cabeçalho, o sistema abre uma nova interface que contém: uma listagem dos acampamentos cadastrados, um mapa com as localizações (pontos pretos) e o cabeçalho. O usuário não precisa estar registrado ou autenticado para acessar esta listagem. (Figuras 11 e 12).

Figura 10 – Página inicial *Home* com os *links* de navegação.



Fonte: elaborado pelo autor.

5.1.2 Página contendo o mapa com as localizações (pontos/*pop-up*), listagem dos acampamentos e cabeçalho.

Interface de listagem dos acampamentos cadastrados pelos usuários (Figuras 11 e 12). O mapa de aglomerados (*cluster map*), mostra a localização, marcados por pontos pretos, de todos os acampamentos contidos na listagem.

O mapa contém ferramentas de *zoom* e rotação (localizado no canto direito superior do mapa). Ambas manuseadas com *mouse* ou *touch*. Um novo *link* “Novo Acampamento” é adicionado ao cabeçalho

Figura 11 – Página contendo o mapa de aglomerados das localizações (pontos pretos), da listagem dos acampamentos e do cabeçalho.

The screenshot shows a web-based application for managing campsite locations. At the top, there's a navigation bar with links for 'Home', 'Acampamentos', and 'Novo Acampamento'. On the right side of the header, there are 'Login' and 'Registrar' buttons. The main content area features a map of South America, specifically focusing on Brazil and its neighbors. Numerous black dots are scattered across the map, representing the locations of registered campsites. Below the map, a section titled 'Visite os Acampamentos' is displayed. It includes two examples: 'Trilhas de Minas Pousada Camping' (with a photo of a yellow tent in a field) and 'Montanha do Lobo Pousada Camping' (with a photo of a campsite near a body of water). Each example includes the campsite name, location details, and a blue 'Visitar Trilhas de Minas Pousada Camping' button.

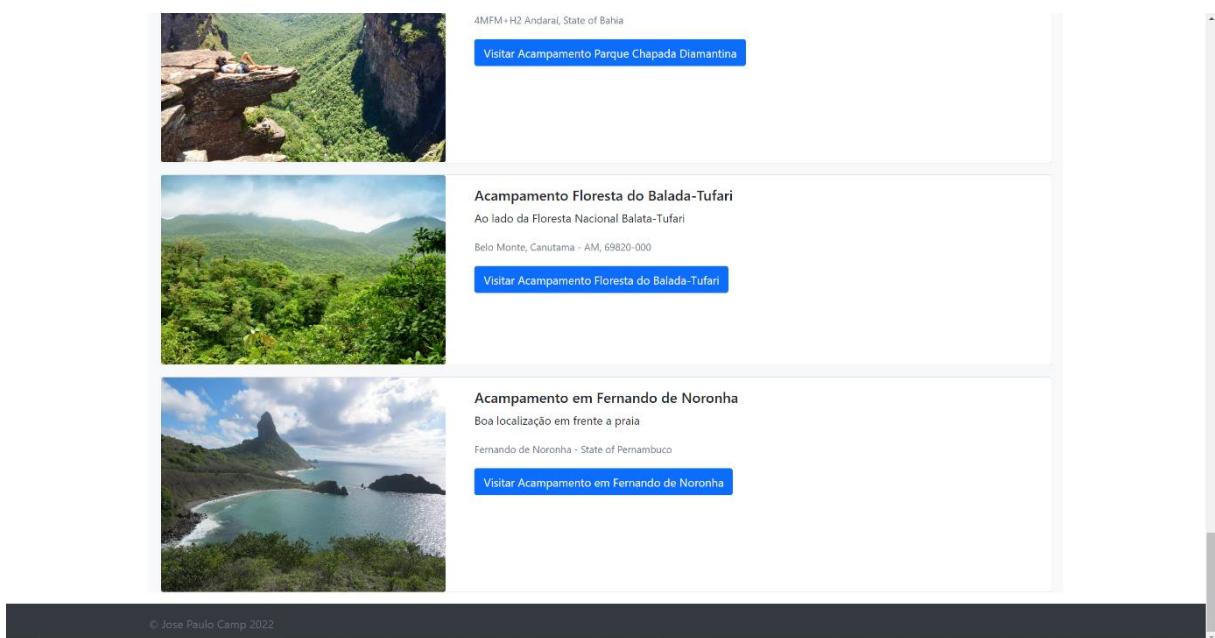
Fonte: elaborado pelo autor.

5.1.3 Página de listagem scroll dos acampamentos e rodapé.

Para cada item da lista dos acampamentos, encontra-se o botão dinâmico azul “Visitar <NOME DO ACAMPAMENTO>”. O usuário escolhe o acampamento. Para ter

acesso a página do acampamento o usuário precisa clicar neste botão. O usuário não precisa estar registrado ou autenticado para visitar a página do acampamento.

Figura 12 – Página de listagem dos acampamentos e rodapé.



Fonte: elaborado pelo autor.

5.1.4 Página mostrando o *pop-up* do acampamento (*mouse click*) no ponto preto, no mapa de aglomerados.

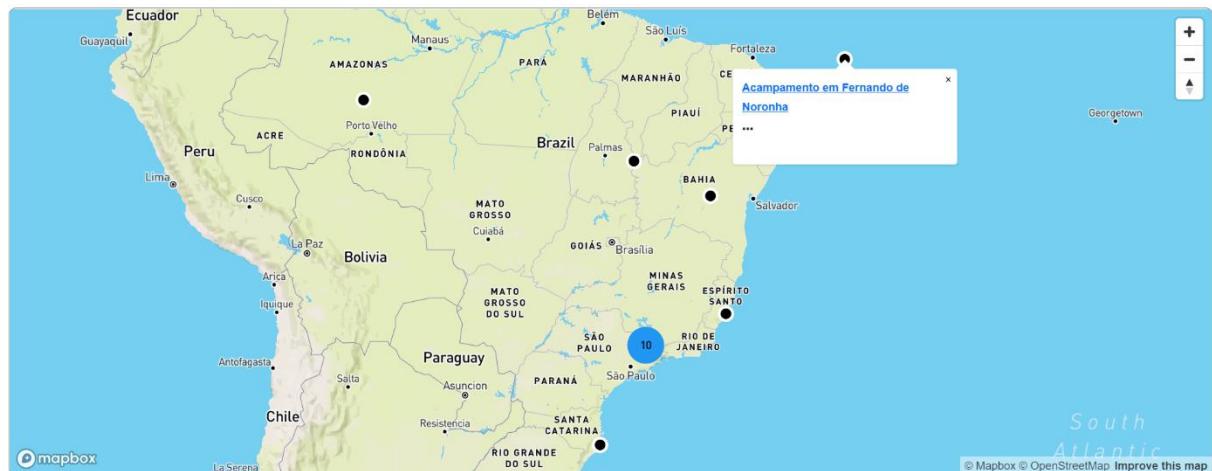
O mapa de aglomerados (*cluster map*) apresenta nos pontos pretos a localização dos acampamentos da listagem. O usuário pode clicar no ponto desejado para abrir um *pop-up* branco contendo o *link* da página do acampamento (Figura 13).

O mapa também contém no lado superior direito, as ferramentas de *zoom* e rotação. Podem ser utilizados com o *mouse* ou toque (*touch*).

Podemos ver também o círculo azul com o número 10 (Figura 13). Estes círculos são criados pelo gerenciador de mapas, Mapbox, quando na região do mapa

existem vários acampamentos. Neste caso significa que existem dez acampamentos na região (mapa de aglomerados).

Figura 13 – Página mostrando a localização e o *pop-up* do acampamento (*mouse click*) no ponto preto, no mapa aglomerados (*cluster map*).



Fonte: elaborado pelo autor.

5.1.5 Página apresentando o acampamento selecionado na listagem.

Interface apresentando as informações do acampamento. A página apresenta as informações como título, localização, preço por noite, descrição e nome do usuário que cadastrou o acampamento. As imagens são apresentadas em *loop* de carrossel (BOOTSTRAP, 2022), ou intercambiadas com click na flecha branca nas laterais da foto.

Encontramos também a lista de avaliações dos usuários que viajaram ao acampamento.

Um mapa menor centralizado e com *zoom* bem aproximado, apresenta a localização do acampamento. Este também possui ferramentas de *zoom* e rotação, como visto no mapa de aglomerados contido na página da listagem (Figura 13).

Figura 14 – Página apresentando o acampamento selecionado na listagem.

The screenshot shows a web page from the 'Jose Paulo Camp' website. At the top, there's a navigation bar with links for 'Home', 'Acamparamentos', 'Novo Acampamento', 'Login', and 'Registrar'. The main content area features a large image of a coastal landscape with a prominent rock formation. Below the image, the title 'Acampamento em Fernando de Noronha' is displayed, followed by the subtitle 'Boa localização em frente a praia'. A map of Fernando de Noronha is shown with a pin indicating the campsite location. Two reviews are listed: one from 'Flavia' (5 stars, 'Excelente') and one from 'Jose Paulo Archetti Conrado' (5 stars, 'Ótima localização, Boa infraestrutura, Preço: Muito caro'). The review from Flavia includes a snippet of text: 'Avaliação: Ótima localização, Boa infraestrutura. Preço: Muito caro.' At the bottom of the page, a footer bar contains the text '© Jose Paulo Camp 2022'.

Fonte: elaborado pelo autor.

5.1.6 Página de registro do novo usuário.

Página contendo o formulário de registro de novo usuário. Para acessar o formulário, basta clicar no link “Registrar” no cabeçalho.

Figura 15 – Página de registro do novo usuário.

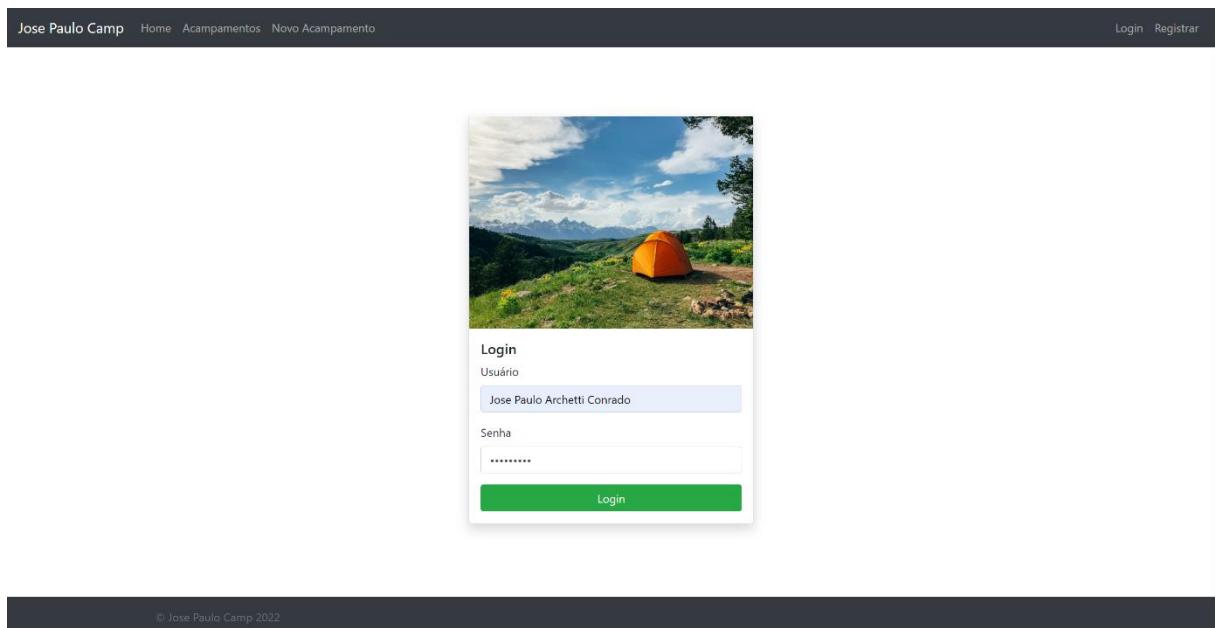
The screenshot shows a registration form on the 'Jose Paulo Camp' website. The top navigation bar includes 'Home', 'Acamparamentos', 'Novo Acampamento', 'Login', and 'Registrar'. The main form area has a background image of a tent in a scenic, hilly landscape. The form itself is titled 'Registrar' and contains fields for 'Usuário' (with placeholder 'Jose Paulo Archetti Conrado'), 'Email' (with placeholder 'jose.conrado@fatec.sp.gov.br'), and 'Senha' (with a masked input field). A green 'Registrar' button is at the bottom right of the form. A footer bar at the bottom of the page contains the text '© Jose Paulo Camp 2022'.

Fonte: elaborado pelo autor.

5.1.7 Página de login para autenticação do usuário.

Página contendo o formulário de *login* do usuário. Para acessar o formulário, basta clicar no *link* “Login” no cabeçalho.

Figura 16 – Página de login para autenticação do usuário.



Fonte: elaborado pelo autor.

5.1.8 Página para criar um acampamento.

Página contendo o formulário de cadastro de um novo acampamento. O usuário precisa estar registrado e autenticado para ter acesso a esta funcionalidade. Para acessar o formulário, após a autenticação, basta clicar no *link* “Novo Acampamento” localizado no cabeçalho.

Figura 17 – Página para criar um acampamento.

Bem vindo! Estamos felizes com seu retorno!

Novo Acampamento

Titulo
Trilhas de Minas Pousada Camping

Localização
Delfinópolis, State of Minas Gerais, 37910-000

Preço do Camping
R\$/noite 200

Descrição
Bem localizado. Ótima Infraestrutura.

camping.jpg, free-camping-1024x682.jpg, photo-1530488345268-51e61... Procurar

[Adicionar Acampamento](#)

[Acampamentos Disponíveis](#)

© Jose Paulo Camp 2022

Fonte: elaborado pelo autor.

A Figura 17, mostra também o *banner* verde com mensagem de autenticação do usuário bem-sucedida. Para as mensagens de operações malsucedidas a cor do banner será vermelha

5.1.9 Página apresentando os botões de edição e avaliação do acampamento.

A página do acampamento abaixo mostra o botão azul de “Editar” e vermelho de “Remover” na seção que mostra as informações do acampamento. Os botões somente aparecem para o usuário, que cadastrou o acampamento. Para atualizar as informações e imagens do acampamento, basta clicar no botão “Editar”. Para remover o acampamento da listagem, basta clicar no botão “Remover”.

Também mostramos o botão verde “Enviar” e vermelho “Remover” da seção de avaliações. O usuário que viajou ao acampamento precisa estar logado para deixar uma ou mais avaliações. O botão vermelho “Remover” somente aparece ao usuário que criou a avaliação.

Figura 18 – Página apresentando os botões de edição e avaliação do acampamento.

Fonte: elaborado pelo autor.

Para remover a avaliação, basta clicar no botão. Para avaliar o acampamento, o usuário escolhe uma nota de 0 a 5, passando o mouse e preenchendo as estrelas amarelas. No campo Texto da Avaliação, o usuário digita o texto da avaliação. Ao terminar, basta clicar no botão verde “Enviar”. Uma mensagem confirma a operação bem-sucedida (Figura 20).

5.1.10 Página para atualizar dados do acampamento.

Página contendo o formulário de atualização das informações e imagens de um acampamento. O usuário precisa estar registrado e autenticado para ter acesso a esta funcionalidade. Para acessar o formulário, após a autenticação, basta clicar no botão “EDITAR” localizado na página do acampamento. O botão somente ficará acessível para o usuário que efetuou o cadastro do acampamento (Figura 18).

Figura 19 – Página para atualizar dados do acampamento.

The screenshot shows a web-based form titled "Atualizar os dados do acampamento". The form has several input fields:

- Título: Trihas de Minas Pousada Camping
- Localização: P072+MG7 Babilônia, Delfinópolis - State of Minas Gerais
- Preço do Acampamento: R\$/noite 100
- Descrição: Perto da Cachoeira

Below these fields is a section for adding images:

- A text input field: "Adicione mais imagem(s)..."
- A "Procurar" (Search) button.
- Three images are listed:
 - A yellow tent in a field with a "Remove Image?" checkbox.
 - A blue tent by a river with a "Remove Image?" checkbox.
 - A person sitting near a green tent with a "Remove Image?" checkbox.

At the bottom of the form is a green "Atualizar Acampamento" (Update Campsite) button.

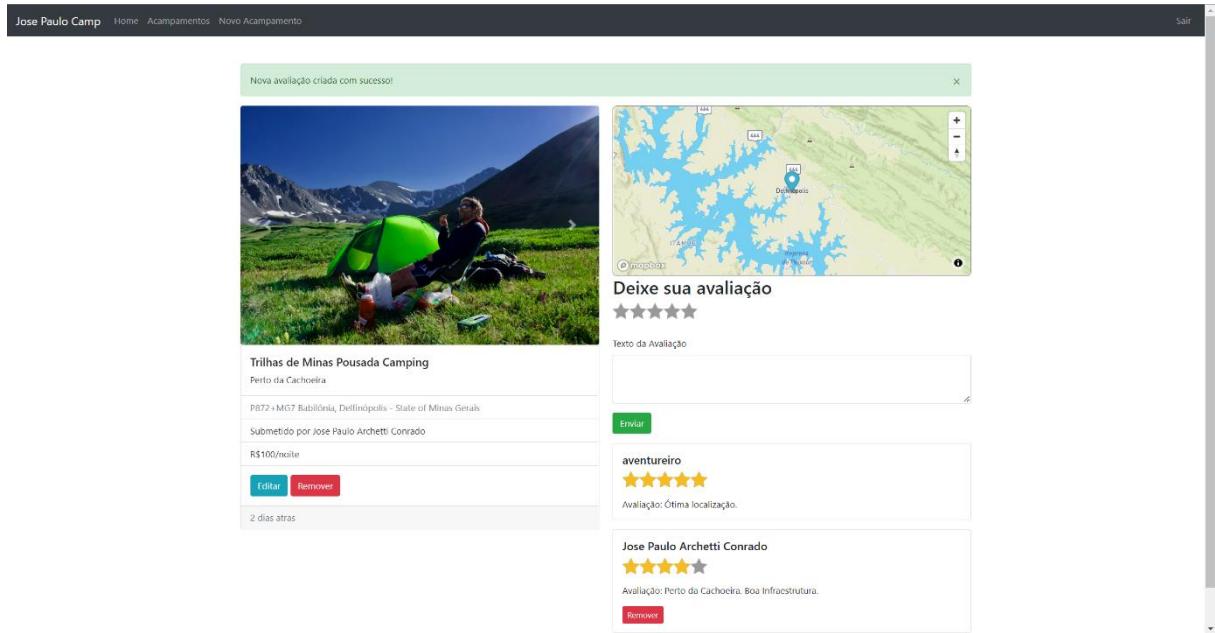
Fonte: elaborado pelo autor.

5.1.11 Página mostrando o *banner* de nova avaliação criada.

Página mostrando o *banner* de cor verde (Figura 20), com mensagens de operações bem-sucedidas. Neste caso mostramos a mensagem com a finalização do

envio de uma avaliação bem-sucedido. Caso ocorra uma mensagem malsucedida a mensagem será apresentada em um *banner* de cor vermelha.

Figura 20 – Página mostrando nova avaliação criada.



Fonte: elaborado pelo autor.

Para sair do ambiente de usuários autenticados, basta clicar no *link* “Sair”, localizado no canto direito superior do cabeçalho (Figura 20). O usuário pode continuar vendo a página de listagens dos acampamentos, mas perde o privilégio de acessar os serviços de inclusão e alteração de informações do aplicativo.

O usuário que não sair da área de autenticação (*link* “Sair”) e fechar o browser, não precisara fazer uma nova autenticação, se o prazo da Express Session (Figura 9.21) não estiver expirado (validade 14 dias). O usuário é identificado pelo *cookie* armazenado no browser (Figura 9.22).

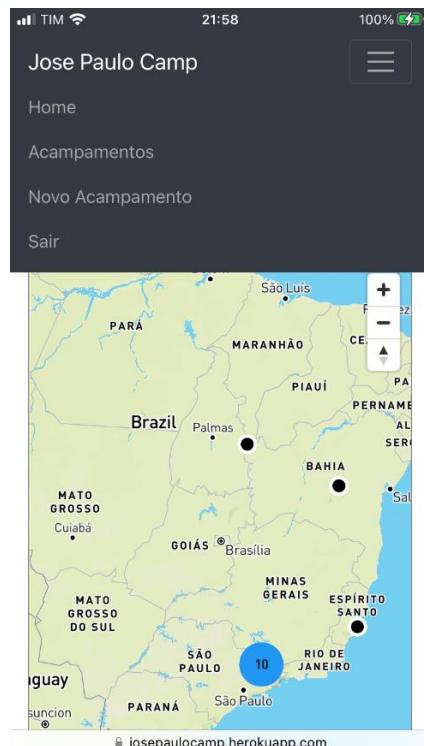
5.1.12 Interfaces renderizadas em browser do smartphone

Figura 21 – Página Home no browser do smartphone.



Fonte: elaborado pelo autor.

Figura 22 – Página listagem, menu e mapa de aglomerados no browser do smartphone.



Fonte: elaborado pelo autor.

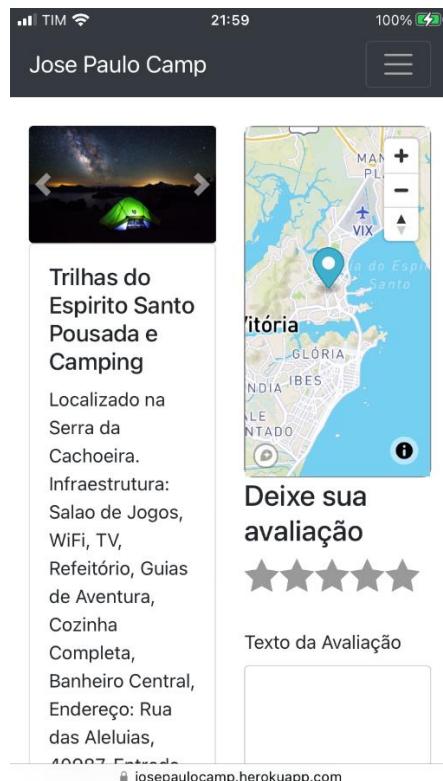
Figura 24 – Página listagem mostrando um acampamento no *browser do smartphone*.



josepaulocamp.herokuapp.com

Fonte: elaborado pelo autor.

Figura 23 – Página do acampamento no *browser do smartphone*.



Fonte: elaborado pelo autor.

Considerações finais

Desafios e Aprendizado

Os maiores desafios do projeto foram gerenciar e armazenar imagens, renderizar mapas e a geolocalização dos acampamentos. Tivemos obstáculos, de como armazenar as imagens, e da conversão da *string* do endereço em coordenadas geográficas. O desenvolvimento foi bem complexo, pois envolveu várias ferramentas, arquitetura *monorepo* e comunicação com diversas APIs. O projeto ajudou no aprendizado do funcionamento, dos conceitos de uma aplicação *web*.

Custos de Infraestrutura

Tivemos uma preocupação nos custos de infraestrutura. Planejamos usar os serviços que ofereceram o conceito de custo proporcional ao volume utilizado. Fizemos a instalação (*deploy*) do aplicativo, na plataforma Heroku (HEROKU, 2022), no qual ofereceu custo zero de implantação. O mesmo aconteceu com o banco de dados na nuvem (MONGODB ATLAS, 2022). Os custos da infraestrutura serão majorados proporcionalmente pela demanda de acesso ao aplicativo (Figura 8.1).

Melhorias e Planos Futuros

Para melhorar a responsividade do *front-end* planejamos, a conversão dos EJS *templates* (Figuras de 10 a 20), para componentes ReactJs. Para implementação de um e-commerce (projeto futuro) vamos precisar de um banco de dados relacional, para atender as regras de negócio. Temos um desafio técnico futuro para melhorar a experiência do usuário (UX), nas inserções de endereço, no formulário de criação de um novo acampamento (Figura 17). O método de conversão *geoespacial* atual, da ferramenta `@mapbox/mapbox-sdk` (subseção 4.1.1), **forwardGeocoding** não apresentou a precisão desejada (Figuras 11, 13 e 14). Pretendemos desenvolver a implementação do método **reverseGeocoding**, da mesma ferramenta, para que o usuário assinale no mapa a localização, e a interface converta em coordenadas precisas.

Pretendemos analisar os *feedbacks* dos *players do mercado*, para implementar melhorias e serviços adicionais. Desenvolvemos o embrião do conceito. Esperamos que o aplicativo atinja os objetivos de fomentar o segmento da indústria de turismo de aventura, dos fornecedores, dos acampamentos e principalmente dos consumidores dos serviços.

Referências

@MPBOX/MAPBOX-SDK, **@mapbox/mapbox-sdk**. Disponível em:

<https://www.npmjs.com/package/@mapbox/mapbox-sdk/v/0.11.0>

<https://www.mapbox.com/>

<https://docs.mapbox.com/mapbox-gl-js/guides/>

<https://www.mapbox.com/mts/>

<https://docs.mapbox.com/mapbox-tiling-service/guides/>

<https://docs.mapbox.com/mapbox-tiling-service/examples/basic-recipe-using-zoom-levels/>

<https://docs.mapbox.com/api/overview/>

Acesso em 01.set.22.

BOOTSTRAP, **Bootstrap**. Disponível em:

<https://getbootstrap.com/docs/5.0/getting-started/introduction/>

Acesso em 01.set.22.

CLOUDINARY, **Cloudinary**. Disponível em:

<https://www.npmjs.com/package/cloudinary/v/1.23.0>

<https://cloudinary.com/>

https://cloudinary.com/documentation/node_quickstart

Acesso em 01.set.22.

CONNECT-FLASH, **Connect-Flash**. Disponível em:

<https://www.npmjs.com/package/connect-flash/v/0.1.1>

Acesso em 01.set.22.

CONNECT-MONGO, **Connect-Mongo**. Disponível em:

<https://www.npmjs.com/package/connect-mongo/v/3.2.0>

Acesso em 01.set.22.

DOTENV, **Dotenv**. Disponível em:

<https://www.npmjs.com/package/dotenv/v/8.2.0>

Acesso em 01.set.22.

EJS, **Ejs**. Disponível em:

<https://www.npmjs.com/package/ejs/v/3.1.5>

<https://ejs.co/>

<https://www.digitalocean.com/community/tutorials/how-to-use-ejs-to-template-your-node-application>

Acesso em 01.set.22.

EJS-MATE, **Ejs-Mate**. Disponível em:

<https://www.npmjs.com/package/ejs-mate/v/3.0.0>

Acesso em 01.set.22.

EXPRESS, **Express**. Disponível em:

<https://www.npmjs.com/package/express/v/4.17.1>

<http://expressjs.com/>

Acesso em 01.set.22.

EXPRESS-MONGO-SANITIZE, **Express-Mongo-Sanitize**. Disponível em:

<https://www.npmjs.com/package/express-mongo-sanitize/v/2.2.0>

Acesso em 01.set.22.

EXPRESS-SESSION, **Express-Session**. Disponível em:

<https://www.npmjs.com/package/express-session/v/1.17.1>

Acesso em 01.set.22.

HEROKU, **Heroku**. Disponível em:

<https://www.heroku.com/students>

Acesso em 01.set.22.

HELMET, **Helmet**. Disponível em:

<https://www.npmjs.com/package/helmet/v/4.1.1>

<https://www.veracode.com/blog/secure-development/fasten-your-helmetjs-part-1-securig-your-express-http-headers>

Acesso em 01.set.22.

JOI, **Joi**. Disponível em:

<https://www.npmjs.com/package/joi/v/17.2.1>

<https://joi.dev/>

Acesso em 01.set.22.

METHOD-OVERRIDE, **Method-Override**. Disponível em:

<https://www.npmjs.com/package/method-override/v/3.0.0>

Acesso em 01.set.22.

METRÓPOLES, **Turismo pós-pandemia será impulsionado por viagens curtas e mais simples**. Disponível em:

[Turismo pós-pandemia será impulsionado por viagens curtas e mais simples \(metropoles.com\)](https://metropoles.com/turismo-pos-pandemia-sera-impulsionado-por-viagens-curtas-e-mais-simples)

Acesso em 01.set.22.

MONGODB, **Mongodb**, Disponível em:

<https://www.mongodb.com/>

Acesso em 01.set.22.

MONGODB ATLAS, **Mongodb** Atlas. Disponível em:
<https://www.mongodb.com/developer/products/atlas/>
Acesso em 01.set.22.

MONGOOSE, **Mongoose**. Disponível em:
<https://www.npmjs.com/package/mongoose/v/5.10.4>
<https://mongoosejs.com/>
Acesso em 01.set.22.

MULTER, **Multer**. Disponível em:
<https://www.npmjs.com/package/multer>
<https://expressjs.com/en/resources/middleware/multer.html>
Acesso em 01.set.22.

MULTER-STORAGE-CLOUDINARY, **Multer-Storage-Cloudinary**. Disponível em:
<https://www.npmjs.com/package/multer-storage-cloudinary/v/4.0.0>
https://medium.com/@joeeeasy/_/uploading-images-to-cloudinary-using-multer-and-expressjs-f0b9a4e14c54
Acesso em 01.set.22.

NODEJS, **Nodejs**, Disponível em:
<https://nodejs.org/en/>
Acesso em 01.set.22.

PASSPORT, **Passport**. Disponível em:
<https://www.npmjs.com/package/passport/v/0.4.1>
<https://www.passportjs.org/>
Acesso em 01.set.22.

PASSPORT-LOCAL, **Passport-Local**. Disponível em:
<https://www.npmjs.com/package/passport-local/v/1.0.0>
<https://www.passportjs.org/packages/passport-local/>
Acesso em 01.set.22.

PASSPORT-LOCAL-MONGOOSE, **Passport-Local-Mongoose**. Disponível em:
<https://www.npmjs.com/package/passport-local-mongoose/v/6.0.1>
Acesso em 01.set.22.

POPPER JS, **Popper Js**, Disponível em:
<https://popper.js.org/>
Acesso em 01.set.22.

SANITIZE-HTML, **Sanitize-Html**. Disponível em:
<https://www.npmjs.com/package/sanitize-html/v/1.27.4>
Acesso em 01.set.22.

SOMMERVILLE, Ian, tradução Ivan Bosnic e Kalinka G.
de O. Gonçalves; revisão técnica Kechi Hirama.
Engenharia de Software. — 9. ed. São Paulo, Pearson Prentice Hall, 2011.

SCHUSTER, Ethel; LEVKOWITZ, Haim; OLIVEIRA Jr., Osvaldo N.

Writing Scientific Papers in English Successfully: Your Complete Roadmap. São Carlos, Compacta, 2014. p. (192).