

1TINOR-2022

Enterprise Connection – Pitch Gulliver

Abigail Fernanda Madanello Victoria de Lima
RM96351

Gilberto Gonçalves de Lima
RM96769

Kaique Ewerton Bernardo
RM96395

Material complementar



[Acesse o protótipo
funcional no Figma](#)



[Acesse o repositório de
conteúdo no Github](#)

Link Docker DB

https://github.com/abigailvlima/traveller-final/tree/main/docker_traveller

API

<https://github.com/abigailvlima/traveller-final/tree/main/api>

Front

<https://github.com/abigailvlima/traveller-final/tree/main/front>

Modelo lógico Insert

https://github.com/abigailvlima/traveller-final/blob/main/modelo_logico_insert.js

Modelo API Postman

<https://github.com/abigailvlima/traveller-final/tree/main/api/postman>

Protótipo navegável

TRAVELLER

Ajuda Cadastre-se Entrar

Hotéis Lugares

Localização Entrada Saída Adultos Crianças

Buscar

O que os hóspedes estão falando sobre as acomodações
Mais de 400.000 avaliações com uma média de 4,8 ⚡ 5 estrelas

5 estrelas
Excelente localização e estrutura. Tivemos uma ótima estadia.
Gilberto (Brasil)

4 estrelas
Recomendo visitar é um lugar lindo.

5 estrelas
Lugar incrível, se você nunca veio ao Rio de Janeiro é uma obrigação. A melhor maneira de ver o Cristo Redentor é pelo trenzinho permitir uma vista linda.

Kaleque (Brasil)

Reserve no Traveller e viaje com tranquilidade

Proteção com o Traveller
A proteção mais ampla para suas estadas. Sempre inclusa, sempre gratuita.

Opções de cancelamento flexíveis
As opções de cancelamento facilitam a remanejamento de reservas se seu planejamento muda.

Atendimento ao cliente 24h
Fale com nossa equipe de atendimento em qualquer lugar do mundo, a qualquer hora do dia.

© 2022 Traveller, Inc. All rights reserved

f i t

TRAVELLER

Ajuda Cadastre-se Entrar

Hospedagens

Localização Entrada Saída Adultos Crianças

Buscar

Traveller > Hotéis > Macaé

Sobre Jatiúca Hotel & Resort

O Jatiúca Hotel & Resort possui 62 mil m² e está localizado em Macaé, a 25 minutos de carro do Aeroporto Internacional Zumbi-dos-Palmares e em frente à praia. Oferece jardins e piscinas adulto e infantil.

O hotel possui quadra de tênis, sala de jogos e serviço de massagem, por um custo extra.

O hotel de Macaé é servido diariamente com pães, frutas e bolos, além de café, leite e sucos. O Jatiúca Hotel & Resort também dispõe de 2 restaurantes, sendo que um delas possui menu a la carte.

Outras atrações da vila mais próximas:

- A 3 km do parque comercial de Macaé
- A 4 km da Praia das Laranjeiras, Macaé
- A 6 km da catedral metropolitana

O hotel conta com quadra de tênis, sala de jogos e serviço de massagem, por um custo extra.

Fotos Jatiúca Hotel & Resort

Reservar já

© 2022 Traveller, Inc. All rights reserved

TRAVELLER

Ajuda Cadastre-se Entrar

Hospedagens

Localização Entrada Saída Adultos Crianças

Buscar

Traveller > Hotéis > Macaé

Ofertas de hotéis populares
Mais de 100 resultados para sua busca

Jatiúca Hotel & Resort
A 25 min do centro

Preço por noite R\$ 686,00

Hotel Ponta Verde Macaé
A 4,6 km do centro

Preço por noite R\$ 443,00

Hotel Brisa Suites
A 2,27 km do centro

Preço por noite R\$ 400,00

Hotel Brisa Suites
A 2,27 km do centro

Preço por noite R\$ 400,00

Hotel Brisa Suites
A 2,27 km do centro

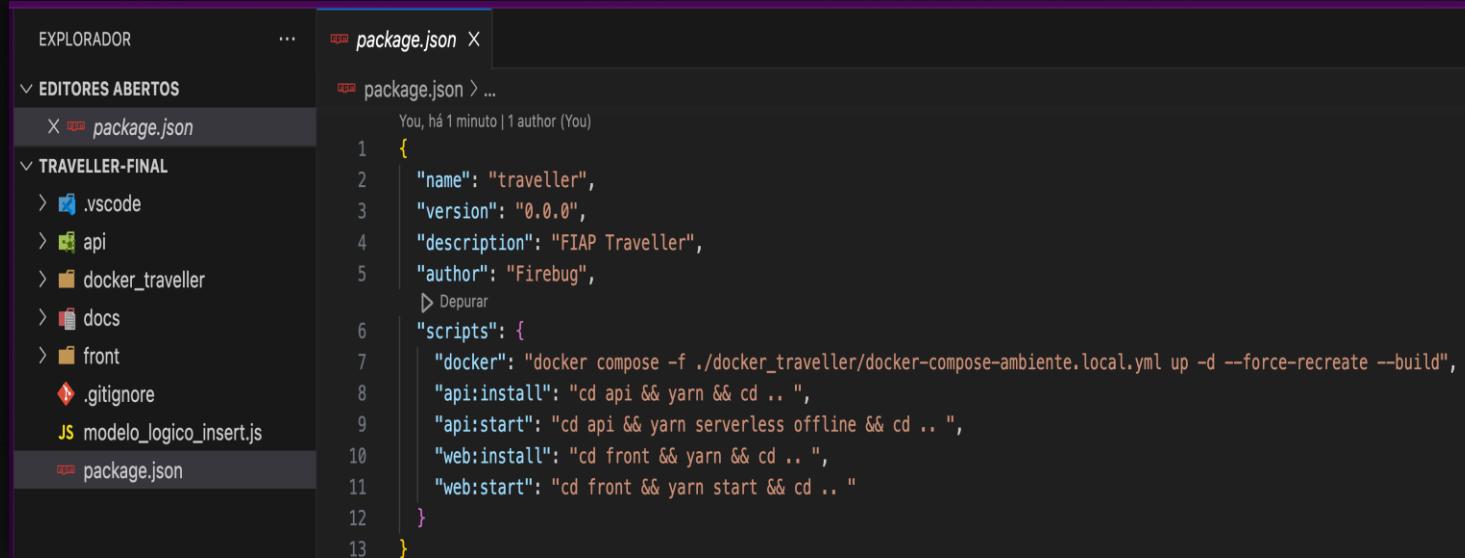
Preço por noite R\$ 400,00

Ver mais ofertas

© 2022 Traveller, Inc. All rights reserved

f i t

Configurações Execução



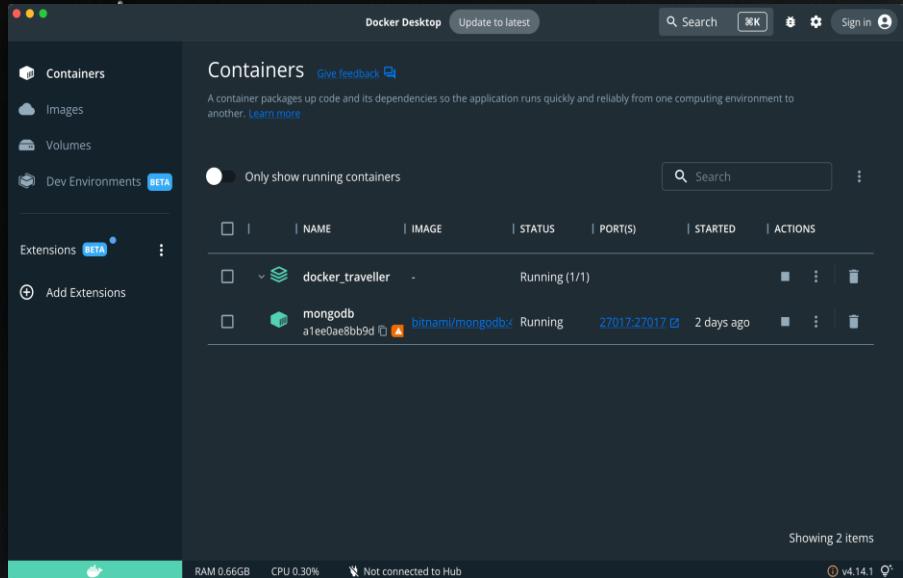
The screenshot shows the VS Code interface with the following details:

- EXPLORADOR**: Shows the project structure:
 - EDITORES ABERTOS**: package.json
 - TRAVELLER-FINAL**: .vscode, api, docker_traveller, docs, front, .gitignore, modelo_logico_insert.js, package.json
- EDITOR DE CÓDIGO**: package.json (active tab)
 - Content:

```
You, há 1 minuto | 1 author (You)
1 {
2   "name": "traveller",
3   "version": "0.0.0",
4   "description": "FIAP Traveller",
5   "author": "Firebug",
6   "scripts": {
7     "docker": "docker compose -f ./docker_traveller/docker-compose-ambiente.local.yml up -d --force-recreate --build",
8     "api:install": "cd api && yarn && cd ..",
9     "api:start": "cd api && yarn serverless offline && cd ..",
10    "web:install": "cd front && yarn && cd ..",
11    "web:start": "cd front && yarn start && cd .."
12  }
13 }
```

Docker NoSQL

Evidência do mongodb rodando no docker



The screenshot shows a VS Code editor window with an open file named 'JS modelo_logico_insert.js'. The code is a Node.js script using Mongoose to insert documents into MongoDB collections 'usuario' and 'locations'. It includes logic to generate ObjectIds and insert user and location data. The file also contains a comment about adjusting files after creation.

```
You, há 1 segundo | 1 author (You)
1 const { ObjectId } = require("mongoose/lib/types");
2
3 db = db.getSiblingDB("traveller");
4 db.createCollection("usuario");
5 db.createCollection("locations");
6
7 db.usuario.insert([
8   {
9     _id: new ObjectId("6466032822b9a867fd065db5"),
10    nome: "Abigail Lima",
11    email: "abigaillima@gmail.com",
12    senha: "Abg102030",
13    data: "2023-05-18T10:51:20.809Z",
14  },
15  {
16    _id: new ObjectId("6466032822b9a867fd065db6"),
17    nome: "Kaique Bernardo",
18    email: "kaiquebernardo@gmail.com",
19    senha: "Abg102030",
20    You, há 17 minutos + ajustando arquivos ...
21    data: "2023-05-18T10:51:20.809Z",
22  },
23
24 db.locations.insert([
25   /* 1 createdAt: 09/06/2023 15:19:41 */
26   {
27     _id: ObjectId("64836d3d1c194c4cca844a6b"),
28     id: "T02",
29     type: 2,
30     bookNow: false,
31     title: "Passeio Noturno Zoológico",
32     stars: 5,
33     distance: "A 4,26 km do centro",
34     pricePerNight: 886,
35     summary: ...
36   }
37 ])
```

Script para inserir e registro de localidades: "Hotéis, restaurantes, pontos turísticos..."

NosQL Client

Collection de localidades

The screenshot shows a MongoDB client interface with two tabs: "traveller:usuario@greenlight" and "traveller.locations@greenlight". The second tab is active, displaying a list of documents in the "locations" collection.

Query in the top-left pane:

```
1 db.locations.find({})
2   .projection({})
3   .sort({_id:-1})
4   .limit(100)
```

Table below the query pane:

| Key | Value | Type |
|------------------------------|---|----------|
| _id | T02 | String |
| type | 2 | Int32 |
| bookNow | false | Bool |
| title | Passeio Noturno Zoológico | String |
| stars | 5 | Int32 |
| distance | A 4,26 km do centro | String |
| pricePerNight | 886 | Int32 |
| summary | Nessa experiência inédita, você poderá caminhar pelo Zoo SP por meio de um trajeto único e delimitado para conhecer os animais que possuem h... | String |
| text | Como os grandes e pequenos felinos, lobo-guará, corujas, serpentes, entre outros. Esses animais receberão itens especiais de en... erão aguçar | String |
| convenience | { wifi : false, pool : false, parking : false, academy : false } (4 fields) | Object |
| assessments | Array[1] | Array |
| photo | data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAD/2wCEAAyEBQYFBAYGBQYHbwYIChAKCgkJChQODwWQFxQYGBcUFhYaHSUIG... | String |
| photos | Array[1] | Array |
| attributes | { title : "Como chegar" } (2 fields) | Object |
| (2) 64836d3d1c194c4cca844a6a | (15 fields) | Document |
| _id | 64836d3d1c194c4cca844a6a | ObjectId |
| id | T01 | String |
| type | 2 | Int32 |
| bookNow | false | Bool |
| title | Catavento Cultural | String |
| stars | 5 | Int32 |
| distance | A 4,26 km do centro | String |
| pricePerNight | 886 | Int32 |
| summary | Do átomo ao maior planeta do sistema solar; do menor inseto aos maiores animais da terra; das leis da física às transformações q ... proteção am... | String |
| text | O Museu Catavento, foi inaugurado em março de 2009 com a missão de aproximar crianças, jovens e adultos do mundo científico, des ... leticida... | String |
| convenience | { wifi : false, pool : false, parking : false, academy : false } (4 fields) | Object |
| assessments | Array[1] | Array |
| photo | data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAAUDBAQEAwUEBAQFBQUGBwwIBwcHBwsLCwkMEQ8SEhEPERETFhwXE... | String |
| photos | Array[1] | Array |

NosQL Client

Usuários

The screenshot shows the NoSQLBooster for MongoDB interface. The left sidebar displays the database structure under the 'greenlight' database, including 'admin', 'config', 'local', 'traveller', 'locations', and 'usuario'. The 'usuario' collection is selected and highlighted in blue. The main panel shows a query editor with the following code:

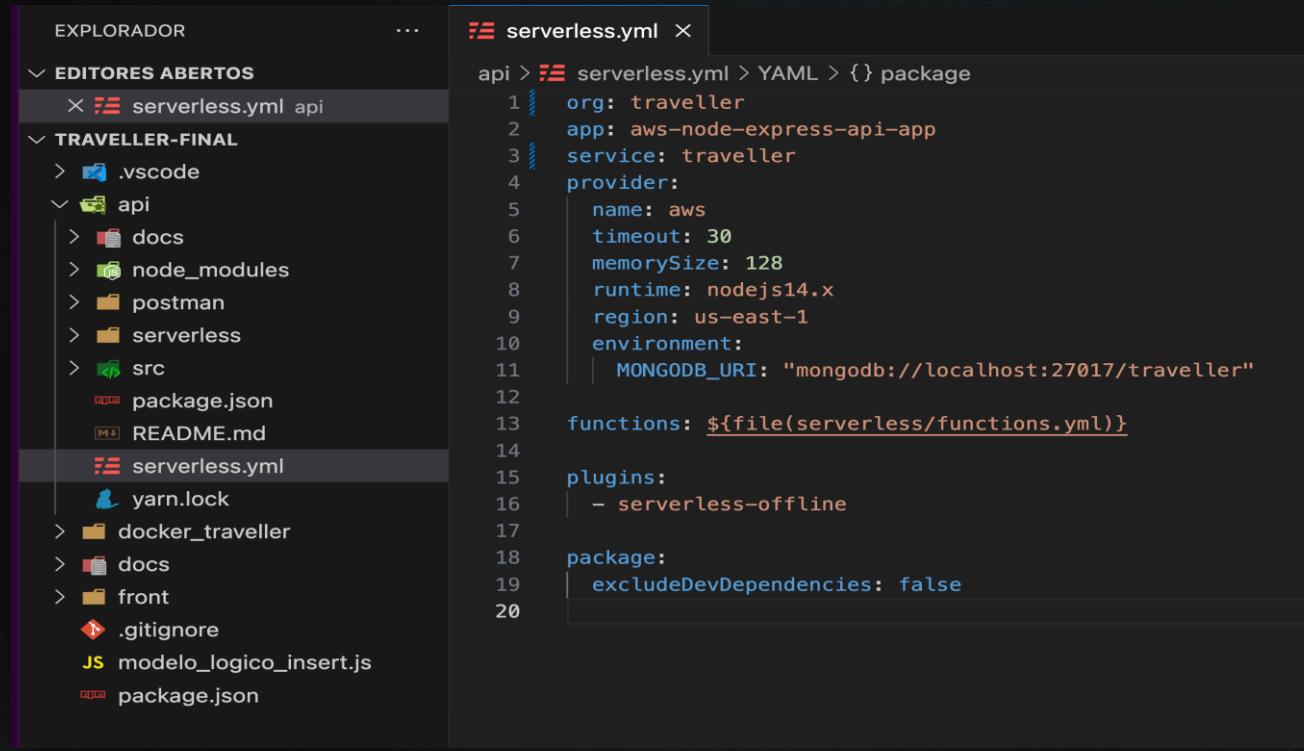
```
1 db.usuario.find({})
2   .projection({})
3   .sort({_id:-1})
4   .limit(100)
```

Below the query, the results are displayed in a table:

| Key | Value | Type |
|--------------------------|---|----------|
| 6466032822b9a867fd065db6 | { email : "kaiquebernardo@gmail.com" } (5 fields) | Document |
| _id | 6466032822b9a867fd065db6 | ObjectId |
| nome | Kaique Bernardo | String |
| email | kaiquebernardo@gmail.com | String |
| senha | Ab@102030 | String |
| data | 2023-05-18T10:51:20.809Z | String |
| 6466032822b9a867fd065db5 | { nome : "Abigail Lima", email : "abigaillima@gmail.com", senha : "Ab@102030", data : "2023-05-18T10:51:20.809Z" } (5 fields) | Document |
| _id | 6466032822b9a867fd065db5 | ObjectId |
| nome | Abigail Lima | String |
| email | abigaillima@gmail.com | String |
| senha | Ab@102030 | String |
| data | 2023-05-18T10:51:20.809Z | String |

API servless.yml e deploy

Configuração serviless.yml para publicação, deploy na AWS



The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor pane on the right.

EXPLORADOR

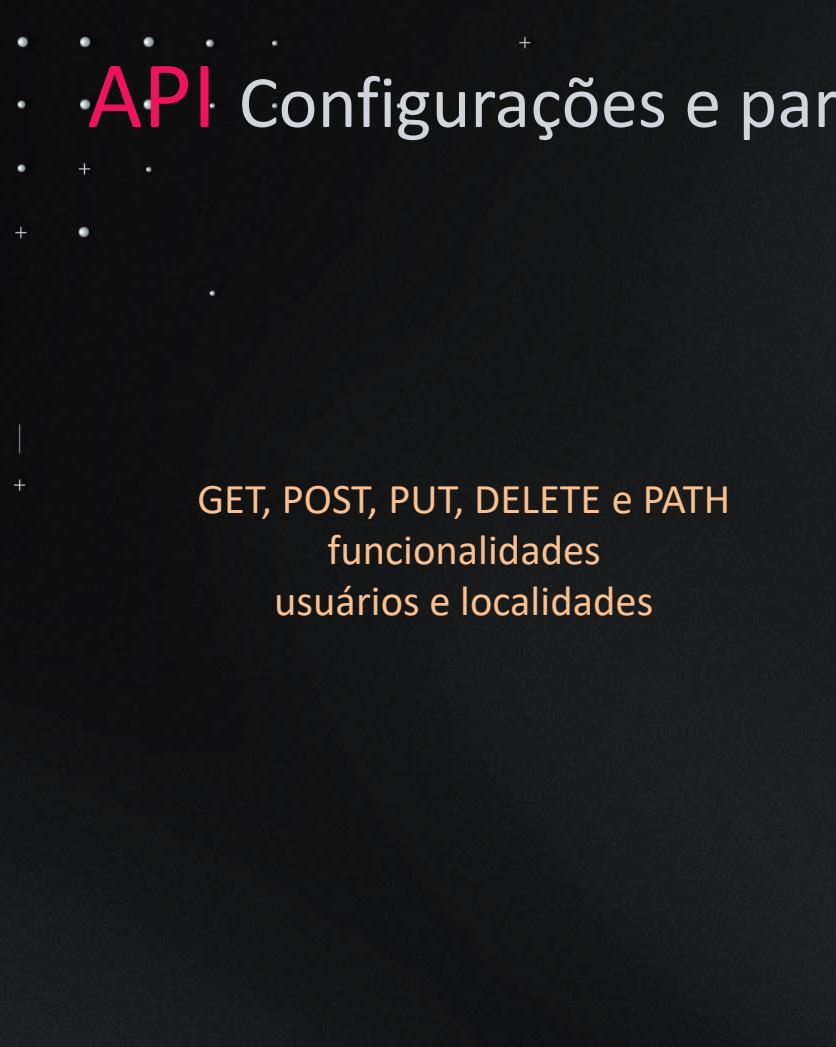
- EDITORES ABERTOS
 - serverless.yml api
- TRAVELLER-FINAL
 - .vscode
 - api
 - docs
 - node_modules
 - postman
 - serverless
 - src
 - package.json
 - README.md
 - serverless.yml
 - yarn.lock
 - docker_traveller
 - docs
 - front
 - .gitignore
 - modelo_logico_insert.js
 - package.json

serverless.yml

```
api > serverless.yml > YAML > {} package
1 org: traveller
2 app: aws-node-express-api-app
3 service: traveller
4 provider:
5   name: aws
6   timeout: 30
7   memorySize: 128
8   runtime: nodejs14.x
9   region: us-east-1
10  environment:
11    MONGODB_URI: "mongodb://localhost:27017/traveller"
12
13  functions: ${file(serverless/functions.yml)}
14
15  plugins:
16    - serverless-offline
17
18  package:
19    excludeDevDependencies: false
20
```

API Configurações e parâmetros

GET, POST, PUT, DELETE e PATH
funcionalidades
usuários e localidades



The screenshot displays the VS Code interface with the following details:

- EXPLORADOR:** Shows the project structure with the following files:
 - EDITORES ABERTOS: serverless.yml, api, functions.yml
 - TRAVELLER-FINAL: .vscode, api (docs, node_modules, postman, serverless), src (package.json, README.md, serverless.yml, yarn.lock), docker_traveller, docs, front (.gitignore, modelo_logico_insert.js, package.json)
- EDITOR:** Shows the contents of the serverless.yml file, which defines various API endpoints (findEvent, loginUser, refreshUser, findUser, insertUser, updateUser) using the Serverless Framework.
- TERMINAL:** Shows the command "Serverless IDE > {} findEvent > []" being typed.

API Modelo de api / postman

The screenshot shows a code editor with several tabs open. The left sidebar lists project files and folders:

- EXPLORADOR
- EDTORES ABERTOS
 - serverless.yml api
 - functions.yml api/serverless
 - API-Traveller.postman_collection.json
- TRAVELLER-FINAL
 - .vscode
 - api
 - docs
 - node_modules
 - postman
 - API-Traveller.postman_collection.json
 - serverless
 - src
 - package.json
 - README.md
 - serverless.yml
 - yarn.lock
- docker_traveller
- docs
- front
- .gitignore
- modelo_logico_insert.js
- package.json

The main editor area has three tabs:

- serverless.yml
- functions.yml
- API-Traveller.postman_collection.json

The content of the API-Traveller.postman_collection.json tab is as follows:

```
api > postman > {} API-Traveller.postman_collection.json > [ ] item > {} 0 > [ ] item
1 {
  "info": {
    "_postman_id": "9d1e75cc-5675-4bba-947b-fb015cc8e1b8",
    "name": "API",
    "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json",
    "_exporter_id": "3890866"
  },
  "item": [
    {
      "name": "Localidades",
      "item": [
        {
          "name": "Consultar",
          "request": {
            "method": "GET",
            "header": [],
            "url": {
              "raw": "{{server}}/locations?limit=3",
              "host": ["{{server}}"],
              "path": ["usuario"],
              "query": [
                {
                  "key": "limit",
                  "value": "3"
                }
              ]
            },
            "response": []
          }
        },
        {
          "name": "Usuarios",
          "item": [
            {
              "name": "Consultar Usuario por Codigo",
              "request": {
```

API AWS - Lambda

- A API é configurada usando o framework Serverless.yml para definir os recursos necessários.

É utilizado o Node.js como ambiente de execução e o TypeScript para fornecer recursos de tipagem estática ao código. A API se conecta ao MongoDB, configurando a conexão com o banco de dados e utilizando o Mongoose como biblioteca para interagir com o MongoDB.

As rotas da API são definidas para lidar com requisições HTTP, como GET, POST, PUT e DELETE, realizando operações correspondentes no banco de dados.

A API manipula as requisições, processando-as e chamando as funções adequadas para lidar com cada solicitação. Ela gera respostas adequadas, formatando os dados em JSON, enviando códigos de status HTTP apropriados e cabeçalhos de resposta relevantes.

Após o desenvolvimento e teste local, a API pode ser implantada em um provedor de nuvem compatível com o Serverless Framework, como a AWS Lambda, para ser executada sem servidor e escalada automaticamente.

API AWS - Lambda

The image shows two side-by-side instances of the Microsoft Visual Studio Code (VS Code) code editor, both displaying code for AWS Lambda functions.

Left Window:

- Title Bar:** JS locations.js
- Status Bar:** api > src > lambda > JS locations.js > ... You, há 40 minutos | 1 author (You)
- Code:**

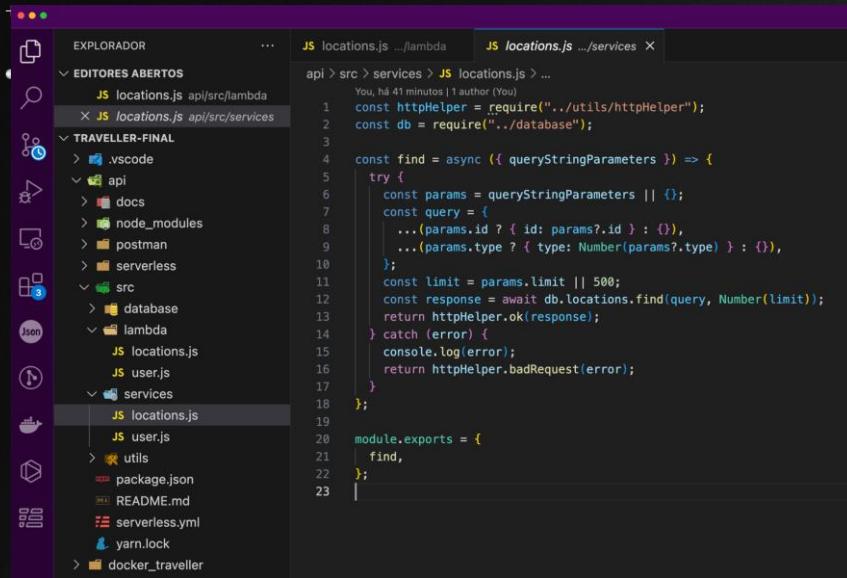
```
1 const { find } = require("../services/locations");
2
3 module.exports = {
4   find,
5 };
6
```
- Explorer View:** Shows the project structure:
 - EDITORES ABERTOS: JS locations.js (highlighted), JS user.js
 - TRAVELLER-FINAL: .vscode, api, docs, node_modules, postman, serverless, src, database, lambda
 - JS locations.js (highlighted)
 - JS user.js
 - services
 - utils
 - package.json
 - README.md
 - serverless.yml
 - yarn.lock
 - docker_traveller

Right Window:

- Title Bar:** JS user.js
- Status Bar:** api > src > lambda > JS user.js > ... You, há 40 minutos | 1 author (You)
- Code:**

```
1 const {
2   find,
3   login,
4   insert,
5   update,
6   remove,
7   refresh,
8   updatePhoto,
9 } = require("../services/user");
10
11 module.exports = {
12   find,
13   login,
14   refresh,
15   insert,
16   update,
17   remove,
18   updatePhoto,
19 }; You, há 40 minutos * ajustando arquivos
```
- Explorer View:** Shows the project structure:
 - EDITORES ABERTOS: JS user.js (highlighted), JS locations.js
 - TRAVELLER-FINAL: .vscode, api, docs, node_modules, postman, serverless, src, database, lambda
 - JS locations.js
 - JS user.js (highlighted)
 - services
 - utils
 - package.json
 - README.md
 - serverless.yml
 - yarn.lock
 - docker_traveller

API AWS - Lambda

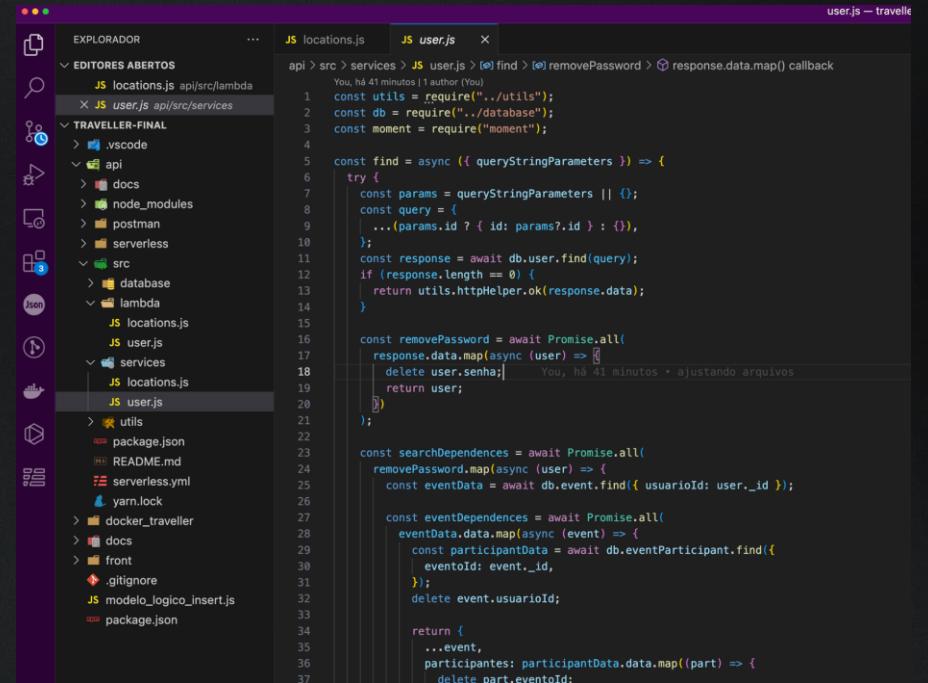


The screenshot shows the Visual Studio Code interface with the following details:

- Explorador:** Shows the project structure with folders like .vscode, api, docs, node_modules, postman, serverless, src, database, lambda, services, and docker_traveller.
- Editores Abertos:** Shows two files: locations.js (api/src/lambda) and locations.js (api/src/services).
- Visualizar:** Shows a preview of the locations.js code.
- Code:** The locations.js file content is displayed:

```
api > src > services > JS locations.js .../services > ...
You, há 41 minutos | author (You)

1 const httpHelper = require("../utils/httpHelper");
2 const db = require("../database");
3
4 const find = async ({ queryStringParameters }) => {
5   try {
6     const params = queryStringParameters || {};
7     const query = {
8       ...(params.id ? { id: params?.id } : {}),
9       ...(params.type ? { type: Number(params?.type) } : {}),
10      };
11    const limit = params.limit || 500;
12    const response = await db.locations.find(query, Number(limit));
13    return httpHelper.ok(response);
14  } catch (error) {
15    console.log(error);
16    return httpHelper.badRequest(error);
17  }
18};
19
20 module.exports = {
21   find,
22};
```



The screenshot shows the Visual Studio Code interface with the following details:

- Explorador:** Shows the project structure with the same set of folders as the first screenshot.
- Editores Abertos:** Shows two files: locations.js (api/src/lambda) and user.js (api/src/services).
- Visualizar:** Shows a preview of the user.js code.
- Code:** The user.js file content is displayed:

```
api > src > services > JS user.js > ...
You, há 41 minutos | author (You)

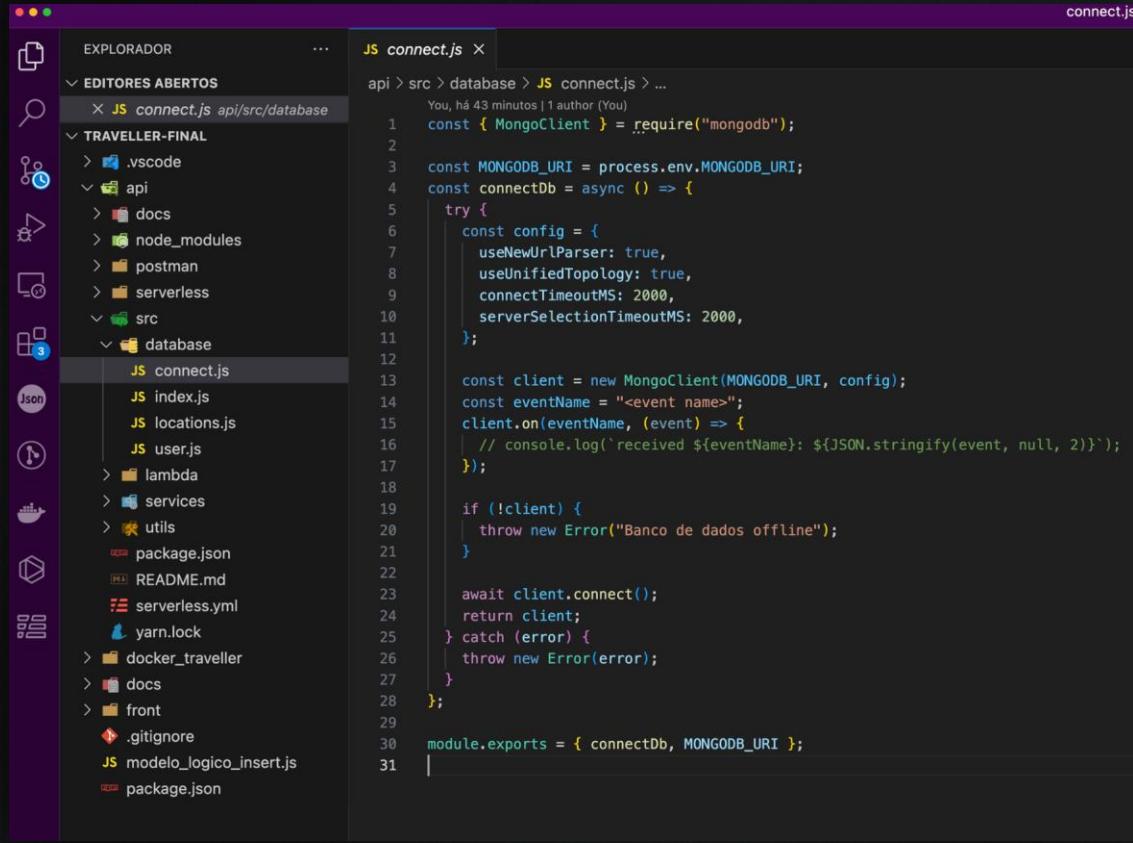
1 const utils = require("../utils");
2 const db = require("../database");
3 const moment = require("moment");
4
5 const find = async ({ queryStringParameters }) => {
6   try {
7     const params = queryStringParameters || {};
8     const query = {
9       ...(params.id ? { id: params?.id } : {}),
10      };
11    const response = await db.user.find(query);
12    if (response.length == 0) {
13      return utils.httpHelper.ok(response.data);
14    }
15
16    const removePassword = await Promise.all(
17      response.data.map(async (user) => [
18        delete user.senha,
19        return user,
20      ]);
21
22
23    const searchDependences = await Promise.all(
24      removePassword.map(async (user) => {
25        const eventData = await db.event.find({ usuarioId: user._id });
26
27        const eventDependences = await Promise.all(
28          eventData.data.map(async (event) => {
29            const participantData = await db.eventParticipant.find({
30              eventId: event._id,
31            });
32            delete event.usuarioId;
33
34            return {
35              ...event,
36              participantes: participantData.data.map((part) => {
37                delete part.eventoId;
38              });
39            };
40          });
41        });
42
43        return {
44          ...event,
45          participantes: participantData.data.map((part) => {
46            delete part.eventoId;
47          });
48        };
49      });
50    });
51
52    return utils.httpHelper.ok(response);
53  } catch (error) {
54    return utils.httpHelper.badRequest(error);
55  }
56}
```

API AWS - Lambda

The screenshot shows a dark-themed interface of the Visual Studio Code code editor. On the left is the Explorer sidebar, which lists several files and folders:

- EDITORES ABERTOS:
 - JS locations.js
 - JS user.js
- TRAVELLER-FINAL:
 - .vscode
 - api
 - docs
 - node_modules
 - postman
 - serverless
 - src
 - database
 - lambda
 - JS locations.js
 - JS user.js
 - services
 - JS locations.js
 - JS user.js
 - utils
 - package.json
 - README.md
 - serverless.yml
 - yarn.lock
 - docker_traveller
 - docs
 - front
 - .gitignore
 - JS modelo_logico_insert.js
 - package.json

API Conexão com Banco de Dados NoSQL



The screenshot shows a dark-themed instance of Visual Studio Code. On the left, the Explorer sidebar displays a project structure under 'EDITORES ABERTOS'. The 'database' folder is expanded, showing files like 'connect.js', 'index.js', 'locations.js', 'user.js', 'lambda', 'services', 'utils', 'package.json', 'README.md', 'serverless.yml', 'yarn.lock', 'docker_traveller', 'docs', 'front', '.gitignore', 'modelo_logico_insert.js', and 'package.json'. The 'connect.js' file is currently selected and open in the main editor area. The code in 'connect.js' is as follows:

```
const { MongoClient } = require("mongodb");
const MONGODB_URI = process.env.MONGODB_URI;
const connectDb = async () => {
  try {
    const config = {
      useNewUrlParser: true,
      useUnifiedTopology: true,
      connectTimeoutMS: 2000,
      serverSelectionTimeoutMS: 2000,
    };

    const client = new MongoClient(MONGODB_URI, config);
    const eventName = "<event name>";
    client.on(eventName, (event) => {
      // console.log(`received ${eventName}: ${JSON.stringify(event, null, 2)})`);
    });

    if (!client) {
      throw new Error("Banco de dados offline");
    }

    await client.connect();
    return client;
  } catch (error) {
    throw new Error(error);
  }
};

module.exports = { connectDb, MONGODB_URI };
```

API Utilitários do Lambda

The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORADOR (Explorer):** Shows the project structure:
 - EDITORES ABERTOS (Open Editors): JS index.js (api/src/utils)
 - TRAVELLER-FINAL (Traveller-Final): .vscode, api, docs, node_modules, postman, serverless, src
 - database, lambda, services, utils
 - dataController.js, db.js, httpHelper.js, index.js (highlighted), lambda.js, returnDb.js
- EDITOR (Editor): JS index.js**

```
api > src > utils > JS index.js > ...
You, há 43 minutos | 1 author (You)
1 const httpHelper = require("./httpHelper");
2 const returnDb = require("./returnDb");
3 const lambda = require("./lambda");
4 const db = require("./db");
5 const dataController = require("./dataController");
6
7 module.exports = {
8   dataController,
9   httpHelper,
10  returnDb,
11  lambda,
12  db,
13};
14
```

API Execução

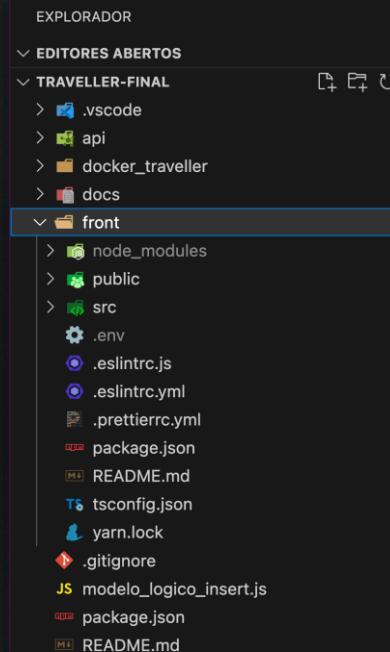
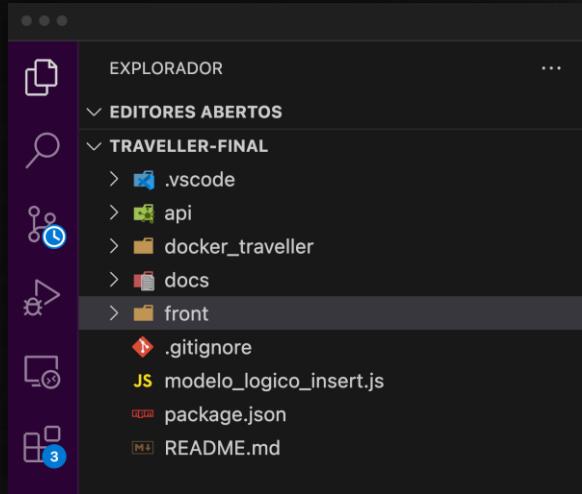
```
Starting Offline at stage dev (us-east-1)

Offline [http for lambda] listening on http://localhost:3002
Function names exposed for local invocation by aws-sdk:
  * findEvent: traveller-dev-findEvent
  * loginUser: traveller-dev-loginUser
  * refreshUser: traveller-dev-refreshUser
  * findUser: traveller-dev-findUser
  * insertUser: traveller-dev-insertUser
  * updateUser: traveller-dev-updateUser
  * uploadPhotoUser: traveller-dev-uploadPhotoUser
  * removeUser: traveller-dev-removeUser

GET  | http://localhost:3000/dev/locations
POST | http://localhost:3000/2015-03-31/functions/findEvent/invocations
POST | http://localhost:3000/dev/usuario/acesso
POST | http://localhost:3000/2015-03-31/functions/loginUser/invocations
PUT  | http://localhost:3000/dev/usuario/acesso
POST | http://localhost:3000/2015-03-31/functions/refreshUser/invocations
GET  | http://localhost:3000/dev/usuario
POST | http://localhost:3000/2015-03-31/functions/findUser/invocations
POST | http://localhost:3000/dev/usuario
POST | http://localhost:3000/2015-03-31/functions/insertUser/invocations
PUT  | http://localhost:3000/dev/usuario
POST | http://localhost:3000/2015-03-31/functions/updateUser/invocations
PUT  | http://localhost:3000/dev/usuario/foto
POST | http://localhost:3000/2015-03-31/functions/uploadPhotoUser/invocations
DELETE | http://localhost:3000/dev/usuario
POST | http://localhost:3000/2015-03-31/functions/removeUser/invocations
```

Server ready: http://localhost:3000 🚀

Front estrutura do código



Front sobre

- Um front-end feito em React e TypeScript é uma aplicação web que utiliza o React, uma biblioteca JavaScript de código aberto, e o TypeScript, uma linguagem de programação que adiciona tipagem estática ao JavaScript. O React é utilizado para criar interfaces de usuário interativas e reativas, enquanto o TypeScript traz recursos adicionais de verificação de tipos durante o desenvolvimento.

Criado front-end em React e TypeScript, dividimos a aplicação em componentes reutilizáveis. Os componentes são blocos de construção da interface de usuário e podem variar em tamanho e complexidade. Eles podem conter elementos como botões, campos de formulário, imagens e outros elementos visuais.

O TypeScript é utilizado para definir tipos de dados e interfaces, o que ajuda a evitar erros comuns de programação e facilita o trabalho em equipe, permitindo que os desenvolvedores tenham uma compreensão clara da estrutura e das propriedades dos componentes.

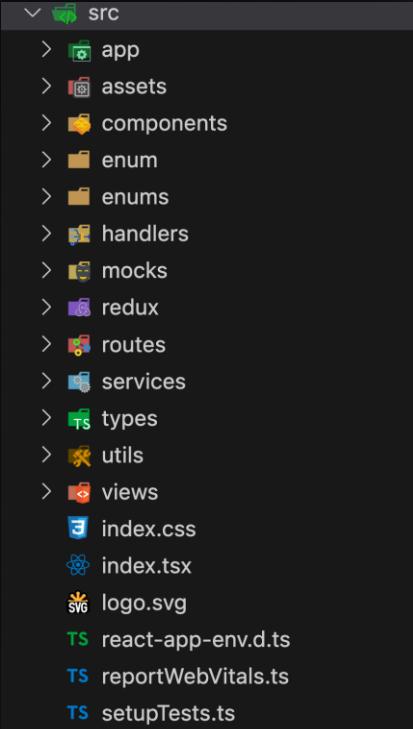
O React trabalha com o conceito de estado, que representa a forma como os dados são armazenados e manipulados pela aplicação. O estado pode ser atualizado e, quando isso acontece, o React atualiza automaticamente a interface de usuário para refletir essas alterações.

Além disso, o React utiliza uma abordagem chamada de "Virtual DOM" (Documento Objeto Virtual) para atualizar eficientemente apenas as partes da interface de usuário que foram modificadas, melhorando o desempenho da aplicação.

Com React e TypeScript, é possível utilizar bibliotecas de terceiros e ferramentas para adicionar recursos adicionais à aplicação, como gerenciamento de estado avançado, roteamento, chamadas de API e muito mais.

No geral, um front-end feito em React e TypeScript combina a poderosa biblioteca React com os recursos de verificação de tipos do TypeScript, permitindo a criação de interfaces de usuário modernas, reativas e escaláveis.

Front estrutura do código



```
index.tsx x
front > src > views > inicioView > index.tsx > default > useEffect() callback
17 |
18 export default () => {
19   const dispatch = useDispatch();
20   const searchState: TSearchState = useSelector((s: RootState) => s.search);
21   const [listCards, setListCards] = useState<TLocation[]>([]);
22
23   useEffect(() => {
24     dispatch(
25       actions.search.execute({
26         limit: 3,
27       })
28     );
29   }, []);
30
31   useEffect(() => {
32     if (searchState.loading) return;
33     if (!searchState.loaded) return;
34     setListCards(searchState.response?.data || []);
35   }, [searchState]);
36
37   return (
38     <ControlePagina>
39       <MenuSuperior items={[{ label: 'Ajuda' }, { label: 'Cadastre-se' }, { label: 'Entrar' }]} />
40       <St.FiltroPesquisaCarousel>
41         <FiltroPesquisaCarousel />
42       </St.FiltroPesquisaCarousel>
43       <St.Row>
44         <TituloPadrao
45           title={'0 que os hóspedes estão falando sobre as acomodações'}
46           subTitle={'Mais de 490.000 avaliações com uma média de 4.8 de 5 estrelas'}
47         />
48       </St.Row>
49       <St.Cards>
50         {listCards.slice(0, 3).map((card) => (
51           <CardHome {...card} />
52         )));
53       </St.Cards>
54       <InfoSeguranca />
55       <Rodape />
56     </ControlePagina>
57   );
58 }
59
```

Front páginas

Página inicio

```
front > src > views > inicioView > index.tsx > default > useEffect() callback
17 |
18 export default () => {
19   const dispatch = useDispatch();
20   const searchState: TSearchState = useSelector((s: RootState) => s.search);
21   const [listCards, setListCards] = useState<TLocation>([]);
22
23   useEffect(() => {
24     dispatch(
25       actions.search.execute({
26         limit: 3,
27       })
28     );
29   }, []);
30
31   useEffect(() => {
32     if (searchState.loading) return;
33     if (!searchState.loaded) return;
34     setListCards(searchState.response?.data || []);
35   }, [searchState]);
36
37   return (
38     <ControlePagina>
39       <MenuSuperior items={[{ label: 'Ajuda' }, { label: 'Cadastre-se' }, { label: 'Entrar' }]} />
40       <St.FiltroPesquisaCarousel>
41         <FiltroPesquisaCarousel />
42       </St.FiltroPesquisaCarousel>
43       <St.Row>
44         <TituloPadrao
45           title="O que os hóspedes estão falando sobre as acomodações"
46           subTitle="Mais de 490.000 avaliações com uma média de 4.8 de 5 estrelas"
47         />
48       </St.Row>
49       <St.Cards>
50         {listCards.slice(0, 3).map((card) => (
51           <CardHome {...card} />
52         )));
53       </St.Cards>
54       <InfoSegurança />
55       <Rodape />
56     </ControlePagina>
57   );
58 }
```

Página Pesquisa

```
front > src > views > pesquisaView > index.tsx > default
22 export default () => {
23   const history = useHistory();
24   const searchState: TSearchState = useSelector((s: RootState) => s.search);
25   const navigateState: TNavigateState = useSelector((s: RootState) => s.navigate);
26   const [listCards, setListCards] = useState<TLocation>([]);
27   const [type, setType] = useState<EType>();
28
29   useEffect(() => {
30     if (searchState.loading) return;
31     if (!searchState.loaded) return;
32     setListCards(searchState.response?.data || []);
33   }, [searchState]);
34
35   useEffect(() => {
36     if (!navigateState.data) return;
37     const data: any = navigateState.data;
38     setType(data.type || '');
39   }, [navigateState]);
40
41   return (
42     <ControlePagina>
43       <St.FlexRow>
44         <MenuSuperior items={[{ label: 'Ajuda' }, { label: 'Cadastro' }]} type={type || EType.hoteis} />
45       <St.Row>
46         <Breadcrumbs
47           data={[
48             {
49               title: 'Traveller',
50               onClick: () => {
51                 history.push('/');
52               }
53             },
54             {
55               title: 'Pesquisa',
56             }
57           ]}
58         />
59       </St.Row>
60       <St.Row>
61         <TituloPadrao
62           title="Ofertas de hotéis populares"
63           subTitle="Mais de 100 resultados para sua busca"
64         />
65       </St.Row>
66     </ControlePagina>
67   );
68 }
```

Página detalhes

```
front > src > views > detalhesView > index.tsx > default > shuffle
18 export default () => {
19   const history = useHistory();
20   const navigateState: TNavigateState = useSelector((s: RootState) => s.navigate);
21   const [item, setItem] = useState<TLocation>();
22
23   function shuffle(array: string[]) {
24     let currentIndex = array.length;
25     let temporaryValue, randomIndex;
26
27     while (currentIndex != 0) {
28       randomIndex = Math.floor(Math.random() * currentIndex);
29       currentIndex -= 1;
30
31       temporaryValue = array[currentIndex];
32       array[currentIndex] = array[randomIndex];
33       array[randomIndex] = temporaryValue;
34     }
35
36     return array;
37   }
38
39   useEffect(() => {
40     if (!navigateState.data) return;
41     const itemTemp: TLocation = navigateState.data;
42     const photos = shuffle(itemTemp.photos || []);
43     setItem({ ...itemTemp, photosRandom: photos });
44   }, [navigateState]);
45
46   return (
47     <ControlePagina>
48       <St.FlexRow>
49         <MenuSuperior items={[{ label: 'Ajuda' }, { label: 'Cadastre-se' }, { label: 'Entrar' }]} />
50         <FiltroPesquisa redirectRoute={true} type={item.type || EType.hoteis} />
51       <St.Row>
52         <Breadcrumbs
53           data={[
54             {
55               title: 'Traveller',
56               onClick: () => {
57                 history.push('/');
58               }
59             },
60             {
61               title: 'Detalhes'
62             }
63           ]}
64         />
65       </St.Row>
66     </ControlePagina>
67   );
68 }
```

Front páginas

Integração dos serviço de pesquisa

The screenshot shows a dark-themed code editor interface. On the left is the Explorer sidebar, which lists project files and folders. In the center is the main editor area displaying the content of a file named `serviceSearch.ts`. The code is a TypeScript file that performs a search operation, likely using Elasticsearch, by sending a POST request to a service endpoint. The code includes imports for `HttpOptions`, `HttpResponse`, `TSearchRequest`, `TSearchResponse`, and `utils`. It defines a function `buscarDados` that takes a `TSearchRequest` object and returns a `Promise<TSearchResponse>`. The function constructs the URL path based on the request type and limit, prepares the parameters, sends the request, and handles the response. A note at the bottom right indicates the file was last modified 59 minutes ago.

```
front > src > services > TS serviceSearch.ts > buscarDados > params > <unknown>
You, há 69 minutos | 1 author (You)
1 import cardsHome from '../mocks/cardsHome';
2 import { HttpOptions, HttpResponse } from '../types/THtt';
3 import { TSearchRequest, TSearchResponse } from '../types/TSearch';
4 import { getRequest } from './utils/service.https';
5
6 export const buscarDados = async (request: TSearchRequest): Promise<TSearchResponse> => {
7   // const mockBool = Boolean(process.env.REACT_APP_MOCK);
8   // if (mockBool == true)
9   //   return {
10   //     err: false,
11   //     data: cardsHome().filter((f) => f.type == request.type),
12   //   };
13
14   const options: HttpOptions = {
15     path: `${process.env.REACT_APP_API_URL}/locations`,
16   };
17
18   const params = {
19     ...[request.type ? { type: request.type } : {}],
20     ...[request.limit ? { limit: request.limit } : {}],
21   };
22
23   const response: HttpResponse = await getRequest(options, params);
24
25   if (response.err) {
26     return {
27       err: true,
28       data: undefined,
29     };
30   }
31
32   const data = response.data;
33   return data || [];
34 };
35
```

Front páginas

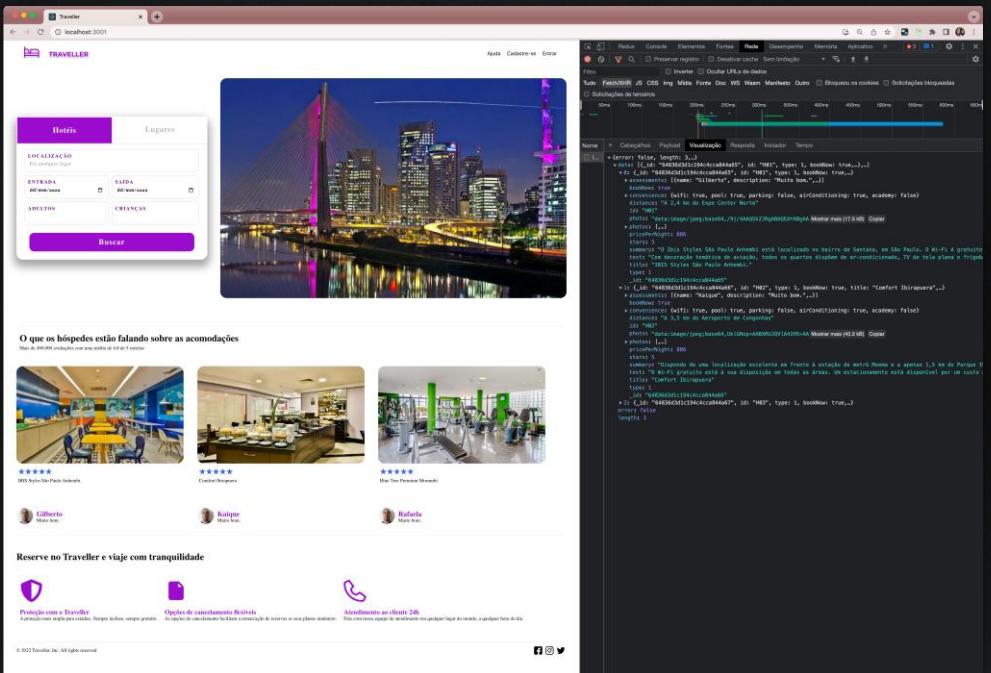
Declaração de Rotas

```
index.tsx .../inicioView index.tsx .../routes X
front > src > routes > index.tsx > ...
You, há 4 dias | 1 author (You)
1 import { Switch, Route } from 'react-router';      You, semana passada • adicionado
2 import { ConnectedRouter } from 'connected-react-router';
3 import { history } from '../app/history';
4
5 import HomeView from '../views/inicioView';
6 import DetalhesView from '../views/detalhesView';
7 import PesquisaView from '../views/pesquisaView';
8
9 export default () =>
10   <ConnectedRouter history={history}>
11     <Switch>
12       <Route exact={false} path="/detalhes" component={DetalhesView} />
13       <Route exact={false} path="/pesquisa" component={PesquisaView} />
14       <Route exact={false} path="/" component={HomeView} />
15       <Route path="*" exact={true} component={HomeView} />
16     </Switch>
17   </ConnectedRouter>
18 );
19
```

Front execução e integração com API

Página Home

Nela o usuário pode fazer a pesquisa de hotéis ou lugares como ponto turístico e restaurantes.



Front execução e integração com API

Página Pesquisa Hotéis

Na página de hotéis, serão exibidos os resultados correspondentes à localidade e datas especificadas na busca. Acima desses resultados, o usuário terá a opção de realizar uma nova busca caso queira alterar alguma informação ou localidade.

The screenshot shows a search results page for hotels in São Paulo. At the top, there's a navigation bar with links for 'Hotéis', 'Hotéis', 'Hotéis', 'Hotéis', 'Hotéis', 'Hotéis', and a search button. Below the navigation is a search form with fields for 'Cidade' (São Paulo), 'Data de saída' (20/08/2018), 'Data de chegada' (21/08/2018), 'Número de pessoas' (2), and a 'Buscar' button. A section titled 'Ofertas de hotéis populares' displays five hotel cards:

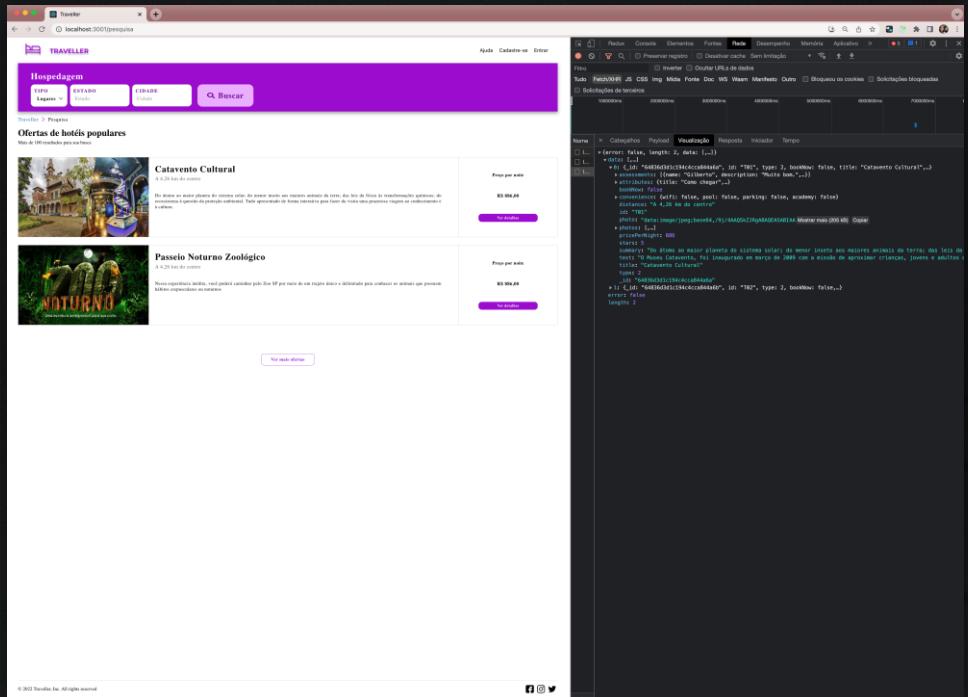
- IBIS Styles São Paulo Anhembi.** Located in the heart of São Paulo, it's a 4-star hotel with a swimming pool and free Wi-Fi. Price per night: R\$ 399,00.
- Comfort Ibirapuera.** Located in the Ibirapuera neighborhood, it's a 3-star hotel with a swimming pool and free Wi-Fi. Price per night: R\$ 399,00.
- Blue Tree Premium Morumbi.** Located in the Morumbi neighborhood, it's a 4-star hotel with a swimming pool and free Wi-Fi. Price per night: R\$ 399,00.
- Mercure São Paulo Pampulha.** Located in the Jardim Paulista neighborhood, it's a 4-star hotel with a swimming pool and free Wi-Fi. Price per night: R\$ 399,00.
- Sheraton São Paulo WTC Hotel.** Located near the World Trade Center, it's a 5-star hotel with a swimming pool and free Wi-Fi. Price per night: R\$ 399,00.

At the bottom of the page, there's a footer with the text "© 2012 Traveler, Inc. All rights reserved." and social media icons for Facebook, Twitter, and LinkedIn. To the right of the main content, there's a developer tools panel showing the network tab with several requests for hotel data, each with a status of "OK".

Front execução e integração com API

Página pesquisa Ponto Turístico

Na página de pontos turísticos, serão exibidos os resultados correspondentes à localidade especificada na busca. Acima desses resultados, o usuário terá a opção de realizar uma nova busca caso queira alterar alguma informação ou localidade.



Front execução e integração com API

Página detalhes do Ponto Turístico

Na página de detalhes do ponto turístico, você encontrará informações sobre como chegar ao local. Além disso, logo acima, há uma opção de busca caso o usuário deseje alterar alguma informação ou localidade.

The screenshot displays a web browser window with the URL `localhost:3001/detalhes`. The page is titled "TRAVELLER" and shows a search interface for "Catavento Cultural". The search bar contains the text "Catavento Cultural". Below the search bar are several buttons: "Hospedagem", "TIPO", "LOCALIZAÇÃO", "ENTRADA", "SAÍDA", "ADULTOS", "CRIANÇAS", and a "Buscar" button. The main content area features a large image of a building at night and a smaller image of an interior exhibit. To the right of the image, there is descriptive text about the museum's history and exhibits. Below the images, there are two sections: "Como chegar" and "De metrô". The "Como chegar" section includes a map and driving directions. The "De metrô" section provides a link to a map. At the bottom of the page, there is a footer with copyright information and social media links. On the right side of the browser window, the developer tools' Network tab is open, showing a list of requests and responses. One request is highlighted, showing a JSON response body containing details about the "Catavento Cultural" point of interest, including its title, address, and nearby landmarks like the "Museu da Cidade".

Front execução e integrado com API

Página detalhes do Hotél

Na página de detalhes do hotel, você encontrará fotos, informações adicionais sobre o hotel e a opção de fazer uma reserva. Além disso, logo acima, há uma opção de busca caso o usuário deseje alterar alguma informação ou localidade.

The screenshot shows a web browser displaying the Hotel Traveller website at localhost:3001/detalhes. The page is for the IBIS São Paulo Anhembi hotel. At the top, there's a navigation bar with links for 'Home', 'Busca', 'Localização', 'Entrada', 'Saída', 'Adultos', 'Crianças', and a search button. Below the header, there's a form for 'Hospedagem' with dropdowns for 'Turista' and 'Básico', and input fields for 'Localização', 'Entrada', 'Saída', 'Adultos', and 'Crianças'. A 'Q. Buscar' button is also present. The main content area features a large image of a restaurant interior with blue walls and yellow tables, accompanied by text describing its features. Below this, there's a section titled 'Fotos' with a grid of smaller images showing various parts of the hotel, including rooms and a breakfast spread. On the far right, the browser's developer tools are visible, specifically the Network tab, which lists several requests made by the page, such as 'IBIS STYLES São Paulo Anhembi', 'IBIS STYLES São Paulo Anhembi', 'IBIS STYLES São Paulo Anhembi', and 'IBIS STYLES São Paulo Anhembi'.

Obrigado !

Estamos prontos para maiores esclarecimentos

FIAF