
**CENTRO UNIVERSITÁRIO FACENS**
**ENGENHARIA DA COMPUTAÇÃO**

ANTÔNIO LUÍS CANNO DE ARAÚJO - 190929 - CP406TIN3
EDUARDO DE OLIVEIRA ALMEIDA - 190963 - CP406TIN3
JHONATAN JOSÉ GOMES DO AMARAL - 190932 - CP406TIN2
LUCAS DE ABREU FURTADO GARCIA - 190581 - CP406TIN3
DANIEL DE LIMA FERREIRA - 200010 - CP406TIN1
ENRICO VENTURINI COSTA - 210901 - CP406TIN1

# Programação Distribuída

**SOROCABA**
**2023**

# Sumário

# Lista de Figuras

# 1 Introdução

Webservice e Webapp são dois conceitos fundamentais no desenvolvimento de aplicações web. Um webservice é uma solução que permite a comunicação entre diferentes sistemas através da internet, utilizando protocolos padrão como HTTP, XML e JSON (W3C, 2004). Ele fornece uma interface que permite que os clientes acessem e interajam com os serviços disponibilizados pelo servidor. Já um webapp, ou aplicativo web, é uma aplicação que roda em um navegador web e pode ser acessada pelos usuários através da internet. Ele utiliza tecnologias como HTML, CSS e JavaScript para criar interfaces interativas e dinâmicas. Tanto os webservices quanto os webapps desempenham um papel crucial na construção de soluções web modernas, permitindo a troca de informações e a criação de experiências interativas na web (PRESSMAN, 2010).

# 2 Descrição do projeto

O grupo decidiu desenvolver uma API que irá permitir a listagem de hotéis com base em diferentes filtros, ou seja, o modelo proposto: WebApp Hotéis. Os quartos disponíveis podem ser de três tipos: com uma cama simples para uma pessoa, com uma cama dupla para duas pessoas ou com duas camas simples também para duas pessoas. Além disso, os hotéis serão classificados em três categorias: baratos, econômicos e luxuosos. Essa API proporcionará aos usuários a possibilidade de encontrar opções de hospedagem que atendam às suas necessidades e preferências, oferecendo uma experiência personalizada e completa.

## 2.1 Objeto do hotel

A interface presente na figura 2, define o tipo "Hotel"em TypeScript.

Figura 1: Interface do hotel.

```
export type Hotel = {
  id: string;
  name: string;
  address: Address;
  starsQuantity: number;
  description: string;
  hasBreakfast: boolean;
  hasLunch: boolean;
  hasDinner: boolean;
  category: HotelCategory;
  parkingLotsQuantity?: number;
  roomCategories: RoomCategory[];
  rentedRooms: RoomRent[];
};
```

Fonte: criado pelo autor

A interface descreve as propriedades que um objeto do tipo Hotel pode ter. Aqui está uma explicação de cada uma das propriedades:

- id: Uma string que representa o identificador único do hotel;
- name: Uma string que representa o nome do hotel;
- address: Um objeto do tipo "Address"que contém informações sobre o endereço do hotel;
- starsQuantity: Um número que representa a quantidade de estrelas do hotel, indicando seu nível de qualidade;
- description: Uma string que contém uma descrição do hotel;
- hasBreakfast: Um valor booleano que indica se o hotel oferece café da manhã;
- hasLunch: Um valor booleano que indica se o hotel oferece almoço;
- hasDinner: Um valor booleano que indica se o hotel oferece jantar;
- category: Um valor do tipo "HotelCategory"que representa a categoria do hotel;
- parkingLotsQuantity: Um número opcional que representa a quantidade de vagas de estacionamento disponíveis no hotel;
- roomCategories: Um array de objetos do tipo "RoomCategory"que descreve as categorias de quartos disponíveis no hotel;

- rentedRooms: Um array de objetos do tipo "RoomRent"que demonstra os quartos que já foram alugados no hotel, sejam ativos ou inativos;

Essa interface define a estrutura de um objeto do tipo Hotel, indicando quais propriedades são necessárias e quais são opcionais. Isso ajuda a garantir que os dados sejam consistentes e facilita a manipulação desses objetos em um programa TypeScript.

## 2.2 Objeto do endereço

A interface presente na figura 2, define o tipo "Endereço"em TypeScript, que é um objeto que está em toda instância de Hotel.

Figura 2: Interface do endereço.

```
export type Hotel = {
  id: string;
  name: string;
  address: Address;
  starsQuantity: number;
  description: string;
  hasBreakfast: boolean;
  hasLunch: boolean;
  hasDinner: boolean;
  category: HotelCategory;
  parkingLotsQuantity?: number;
  roomCategories: RoomCategory[];
  rentedRooms: RoomRent[];
};
```

Fonte: criado pelo autor

Segue a explicação de cada uma das propriedades:

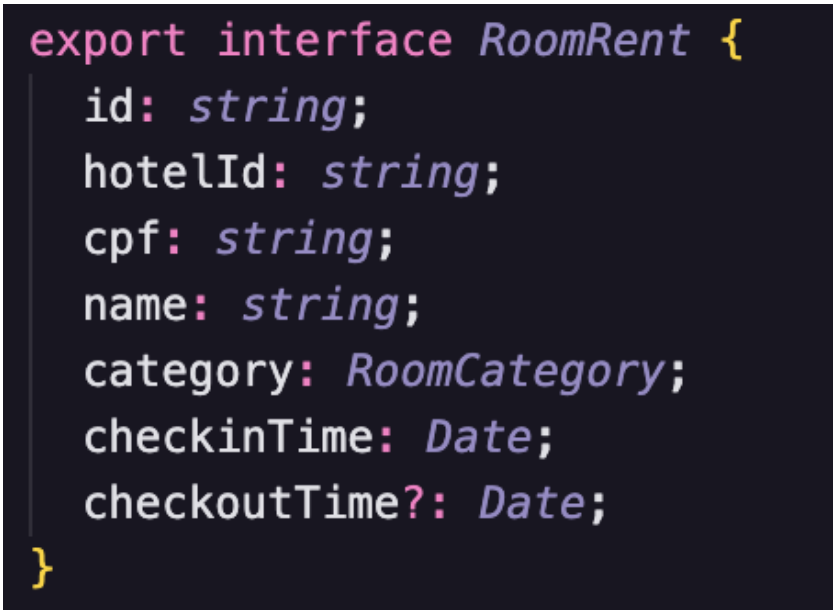
- city: Uma string que representa o nome da cidade do endereço;
- street: Uma string que representa o nome da rua do endereço;
- zipNumber: Uma string que representa o número do CEP do endereço;
- number: Um número opcional que representa o número da residência no endereço;
- neighborhood: Uma string que representa o bairro do endereço;
- country: Uma string que representa o país do endereço;
- state: Uma string que representa o estado do endereço;

- complement: Uma string opcional que representa informações adicionais sobre o endereço;

## 2.3 Objeto do aluguel de quarto

A interface presente na figura 3, define o tipo "Aluguel de quarto"em TypeScript, que é um objeto utilizado dentro da interface Hotel.

Figura 3: Interface do aluguel de quarto.




Fonte: criado pelo autor

Segue a explicação de cada uma das propriedades:

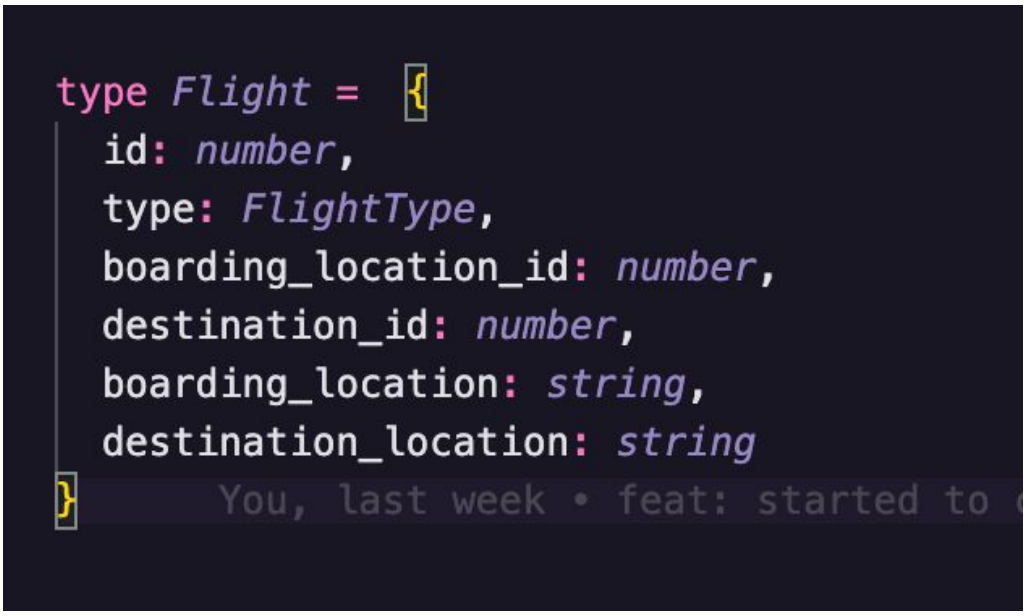
- id: Uma string que representa o identificador único do aluguel do quarto;
- hotelId: Uma string que representa o identificador único do hotel;
- name: Uma string que representa o nome do cliente que alugou o quarto;
- cpf: Uma string que representa o cpf do cliente que alugou o quarto;
- starsQuantity: Um número que representa a quantidade de estrelas do hotel, indicando seu nível de qualidade;
- category: Uma string do tipo "RoomCategory"que descreve as categorias de quartos disponíveis no hotel, e especificamente a categoria do quarto alugado;
- checkinTime: Uma data que representa o momento em que o cliente que alugou o quarto;

- checkoutTime: Uma data que representa o momento em que o cliente que deixou o quarto;

## 2.4 Voo

A figura 4 apresenta a definição do tipo "Voo" em TypeScript, o qual representa um modelo de dados para informações de voos na aplicação.

Figura 4: Dados de voo.




Fonte: criado pelo autor

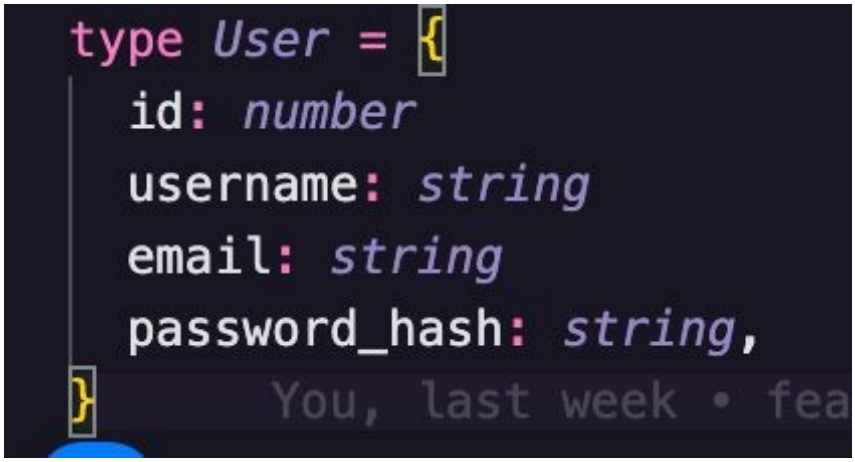
Segue a descrição de cada propriedade do tipo `Flight`:

- id: Um número que serve como identificador único do voo.
- type: Um tipo "FlightType" que categoriza o voo.
- boarding_location_id: Um número que identifica unicamente o local de embarque.
- destination_id: Um número que identifica unicamente o local de destino.
- boarding_location: Uma string que descreve o local de embarque, como o nome do aeroporto ou cidade.
- destination_location: Uma string que descreve o local de destino do voo.

## 2.5 Modelo de Usuário

A Figura 5 mostra a definição do tipo "Usuário" em TypeScript, que é utilizado para modelar informações de usuário em uma aplicação.

Figura 5: Entidade do usuário logado a aplicação.




Fonte: criado pelo autor

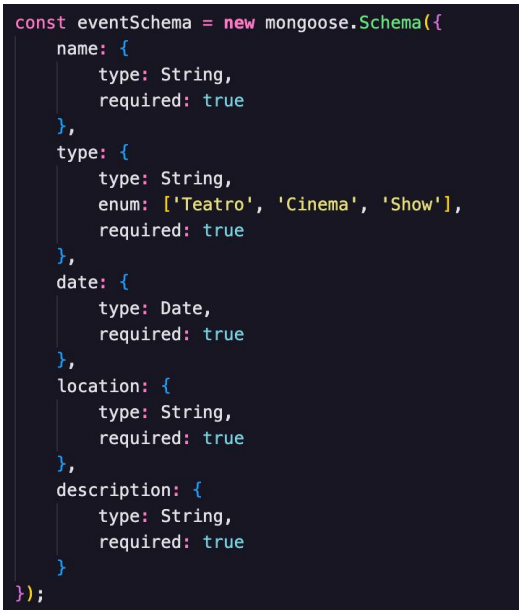
Segue a descrição de cada propriedade do tipo "User":

- id: Um número que serve como identificador único para o usuário.
- username: Uma string que representa o nome de usuário escolhido.
- email: Uma string que armazena o endereço de email do usuário.
- password_hash: Uma string que contém o hash da senha do usuário para segurança.

## 2.6 Entidade do Evento

A Figura 6 ilustra a definição de "eventSchema" utilizando a biblioteca Mongoose em JavaScript, que é usada para modelar a estrutura de dados de eventos em uma aplicação.

Figura 6: Estrutura de dados do evento.



```
const eventSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    type: {
        type: String,
        enum: ['Teatro', 'Cinema', 'Show'],
        required: true
    },
    date: {
        type: Date,
        required: true
    },
    location: {
        type: String,
        required: true
    },
    description: {
        type: String,
        required: true
    }
});
```

Fonte: criado pelo autor

Segue-se a descrição de cada campo do esquema “eventSchema”:

- name: Uma string que é obrigatória e representa o nome do evento.
- type: Uma string que também é obrigatória e só pode assumir os valores ’Teatro’, ’Cinema’, ou ’Show’.
- date: Uma data que é obrigatória e representa a data do evento.
- location: Uma string que é obrigatória e descreve o local do evento.
- description: Uma string que é obrigatória e fornece uma descrição do evento.

## 2.7 Pacote de Turismo

A Figura 7 apresenta a definição do tipo “Pacote” em TypeScript, que é projetado para representar pacotes turísticos em um sistema de gestão de viagens.

Figura 7: Entidade do Pacote de Turismo.


Fonte: criado pelo autor

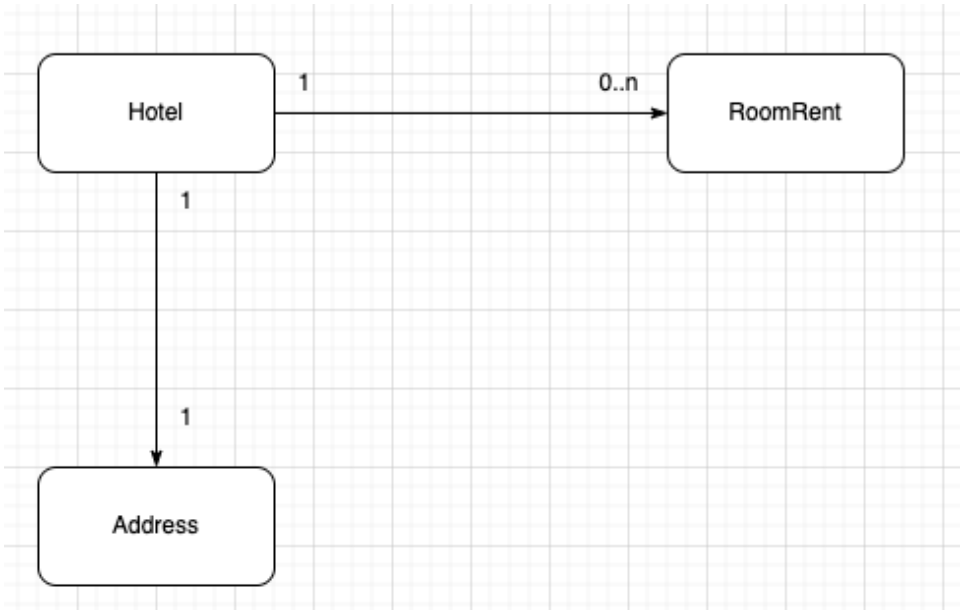
Segue-se a descrição de cada campo do tipo "Pacote":

- id: Uma string que serve como identificador único do pacote turístico.
- cpf: Uma string que armazena o CPF do cliente responsável pela compra do pacote.
- title: Uma string que representa o título ou nome do pacote turístico.
- eventID: Uma string que contém o identificador do evento associado ao pacote, se houver.
- hotelID: Uma string que contém o identificador do hotel associado ao pacote.
- flightID: Uma string que contém o identificador do voo associado ao pacote.

## 2.8 Comunicação entre os objetos

De acordo com as considerações previamente abordadas, constata-se que o objeto hotel é intrinsecamente vinculado a um endereço, e, reciprocamente, o endereço está diretamente associado a um hotel. No que tange ao aluguel de quartos, é possível observar que um determinado hotel pode apresentar uma variação que pode oscilar entre a ausência de aluguéis e a potencialidade de ter um número infinito deles. Essa relação entre o hotel, o endereço e o aluguel de quartos evidencia a interdependência existente entre esses elementos, ressaltando assim a relevância de se considerar tais aspectos para uma compreensão adequada do objeto de estudo em questão, valide a partir da figura 8.

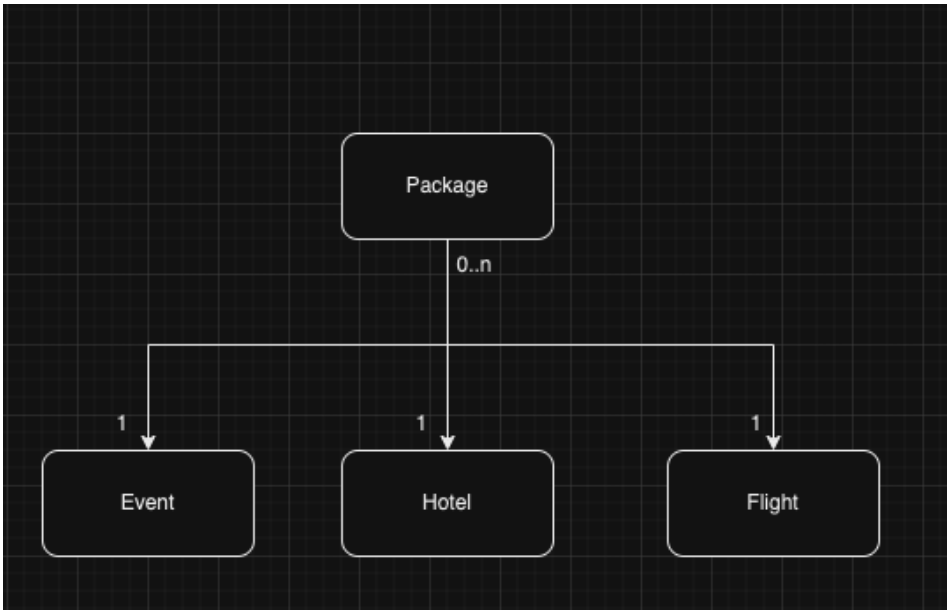
Figura 8: Comunicação entre os objetos principais da API de hotel.




Fonte: criado pelo autor

Ademais, para a elaboração de um pacote turístico, é necessário considerar três elementos essenciais: a ocorrência de um evento específico, a reserva de um voo e a escolha de um hotel localizado na mesma cidade de destino. Esses fatores estão interligados de forma a garantir uma experiência turística completa e satisfatória para o viajante. O diagrama a ser apresentado na figrua 9 evidenciará uma relação em que cada pacote terá apenas uma entidade de cada tipo, mas a entidade pode estar associada a nenhum ou infinitos pacotes.

Figura 9: Comunicação entre os objetos para construção de um pacote.




Fonte: criado pelo autor

# 3 Desenvolvimento

É essencial o entendimento dos contratos e da IDL na criação de uma API de listagem de hotéis, utilizando a ferramenta Swagger. Os contratos estabelecerão as regras, responsabilidades e interfaces necessárias para garantir a comunicação efetiva entre os diferentes componentes envolvidos. Através da IDL, a estrutura será definida e os detalhes da interface da API, especificando as operações disponíveis, os parâmetros requeridos, os tipos de dados envolvidos e os resultados esperados. É de grande importância ressaltar que utilizou-se o Swagger para documentar e visualizar de forma clara e intuitiva a API, facilitando a integração e o entendimento por parte dos desenvolvedores. Assim, a utilização dessas tecnologias de contrato e IDL, em conjunto com o Swagger, será fundamental para garantir a consistência e a interoperabilidade da API de listagem de hotéis.

## 3.1 Entendendo as APIs

Durante o desenvolvimento da aplicação, não foi necessário realizar modificações nas APIs consumidas. Em vez disso, foi criado um novo endpoint para salvar as entidades selecionadas por cada usuário da aplicação. Para facilitar a listagem e montagem dos pacotes turísticos, foram criadas entidades que possuíam a mesma localização tanto para o evento, o local de destino do voo e o local do hotel. Essa abordagem permitiu uma integração mais eficiente e coerente entre as diferentes informações, garantindo uma experiência de usuário mais fluida e organizada, e já dispondo de todas as opções possíveis entre as APIs.

Caso uma localização não tenha uma das três entidades, então não será adicionado um pacote turístico para o local.

## 3.2 Contrato

Pode-se dividir os contratos em duas seções: crud do hotel, e checkin e checkout de um cliente.

### 3.2.1 Hotel

Para adicionar um novo hotel na aplicação, é preciso seguir algumas regras:

- O nome do estabelecimento deve possuir mais de três letras e o endereço deve ser fornecido corretamente, sendo opcional apenas o envio do número e complemento;
- A quantidade de estrelas deve ser um número entre 0 e 5;
- Embora a quantidade de vagas no estacionamento não seja um requisito obrigatório, quando fornecida, deve ser maior ou igual a zero;
- As categorias dos quartos devem ser apresentadas em uma lista contendo no máximo três campos, podendo ser apenas "1_single_bed", "2_single_bed"ou "1_couple_bed";
- A categoria do hotel deve ser selecionada entre as opções "barato", "econômico"ou "luxuoso".

Já para atualizar o hotel, as regras são as mesmas, tendo apenas uma a mais:

- É preciso passar o identificador do hotel que será editado, caso contrário não será possível atualizar nenhuma instância.

Para obter as informações de apenas um hotel, é preciso passar apenas o identificador do hotel já encontrado, contudo para listar mais de uma entidade, é possível fazer alguns tipos de filtros:

- O nome do estabelecimento pode ser passado, não é necessário passar o nome completo, apenas uma parte é o suficiente. Caso não seja passado, não será aplicado nenhuma lógica no retorno de hotéis;
- As categorias dos quartos devem ser apresentadas em uma lista contendo no máximo três campos, podendo ser apenas "1_single_bed", "2_single_bed"ou "1_couple_bed". Caso não seja passado, não será aplicado nenhuma lógica no retorno de hotéis;
- A categoria do hotel deve ser selecionada entre as opções "barato", "econômico"ou "luxuoso". Caso não seja passado, não será aplicado nenhuma lógica no retorno de hotéis.

Por fim, para deletar um hotel, é necessário passar apenas o identificador para a requisição, mas caso o identificador não seja válido ele retornará a mensagem da mesma maneira.

### 3.2.2 Checkin e Checkout

Para realizar o checkin em um quarto de hotel, ou seja alugar um dos quartos, é preciso:

- O identificador do hotel em que se está fazendo o checkin, além de ser obrigatório, ele precisa existir entre os hotéis mapeados no software, gerando um erro ao passar um que não exista na lista;
- O cpf do cliente que está fazendo o checkin, obrigatoriamente;
- O nome do cliente que está fazendo o checkin, obrigatoriamente;
- O momento em que o cliente fez o checkin no hotel, no formato ISOString, obrigatoriamente, e não deve ser no futuro, ou seja, tem que ser menor que o horário da requisição à API;
- A categoria do quarto do quarto alugado deve se apresentado contendo um dos três campos: "1_single_bed", "2_single_bed"ou "1_couple_bed". É importante ressaltar que caso o hotel não tenha essa categoria disponível, será retornado um erro.

Já para o checkout, que seria sair do hotel alugado e devolver as chaves, é preciso:

- O identificador do hotel em que se está fazendo o checkin, além de ser obrigatório, ele precisa existir entre os hotéis mapeados no software, gerando um erro ao passar um que não exista na lista;
- O momento em que o cliente fez o checkout no hotel, no formato ISOString, obrigatoriamente, e não deve ser no futuro, ou seja, tem que ser menor que o horário da requisição à API.

## 3.3 IDL

Nesse capítulo, será abordado a IDL para cada requisição na API de hotéis, sendo dividido em: busca unitária de hotéis, listagem de hotéis, adição de hotéis, atualização de hotéis, remoção de hotéis, checkin e checkout.

É importante ressaltar, que os pontos descritos estão disponíveis, também, no swagger da aplicação.

### 3.3.1 Busca unitária de hotéis

Para realizar a requisição, é preciso chamar a requisição com o método GET, no endpoint: /hotels/id. É importante ressaltar que id deve ser substituído pelo identificador do hotel.

O retorno da API segue o formato conforme a figura 10. Esse retorno segue o seguinte padrão: todo conteúdo que o cliente realmente buscou está dentro do atributo data, já a mensagem que dirá o status da requisição está no atributo message, caso seja uma mensagem positiva. Em cenários de erro, um objeto error é criado, e dentro dele terá a mensagem e, caso esteja acordado, um código que facilitará o dê para entre o cliente e o servidor.

Figura 10: Retorno da busca de um hotel.

```
{
  "data": {
    "id": "string",
    "name": "string",
    "address": {
      "city": "string",
      "street": "string",
      "zipNumber": "string",
      "number": 0,
      "neighborhood": "string",
      "country": "string",
      "state": "string",
      "complement": "string"
    },
    "starsQuantity": 0,
    "description": "string",
    "hasBreakfast": true,
    "hasLunch": true,
    "hasDinner": true,
    "category": "string",
    "parkingLotsQuantity": 0,
    "roomCategories": [
      "string"
    ]
  },
  "message": "string",
  "error": {
    "code": "string",
    "message": "string"
  }
}
```

Fonte: criado pelo autor

### 3.3.2 Listagem de hotéis

Para realizar a requisição, é preciso chamar a requisição com o método GET, no endpoint: /hotels. Além disso, é possível passar três tipos de query params: name, que filtra pelo nome do hotel; category, que filtra pela categoria do hotel; roomCategories, que filtra pela tipo de quarto do hotel;

O retorno da API segue o formato conforme a figura 11. Esse retorno segue o seguinte padrão: todo conteúdo que o cliente realmente buscou está dentro do atributo data, e é uma lista de hotéis, já a mensagem que dirá o status da requisição está no atributo message, caso seja uma mensagem positiva. Em cenários de erro, um objeto error é criado, e dentro dele terá a mensagem e, caso esteja acordado, um código que facilitará o dê para entre o cliente e o servidor.

Figura 11: Retorno da busca de um hotel.


```
{
  "data": [
    {
      "id": "string",
      "name": "string",
      "address": {
        "city": "string",
        "street": "string",
        "zipNumber": "string",
        "number": 0,
        "neighborhood": "string",
        "country": "string",
        "state": "string",
        "complement": "string"
      },
      "starsQuantity": 0,
      "description": "string",
      "hasBreakfast": true,
      "hasLunch": true,
      "hasDinner": true,
      "category": "string",
      "parkingLotsQuantity": 0,
      "roomCategories": [
        "string"
      ]
    }
  ],
  "message": "string",
  "error": {
    "code": "string",
    "message": "string"
  }
}
```


Fonte: criado pelo autor

### 3.3.3 Adição de hotéis

Para realizar a requisição, é preciso chamar a requisição com o método POST, no endpoint: /hotels. Além disso, é preciso enviar no body da requisição o json conforme a figura 12.

Figura 12: Retorno da busca de um hotel.


```
{
  "name": "string",
  "address": {
    "city": "string",
    "street": "string",
    "zipNumber": "string",
    "number": 0,
    "neighborhood": "string",
    "country": "string",
    "state": "string",
    "complement": "string"
  },
  "starsQuantity": 0,
  "description": "string",
  "hasBreakfast": true,
  "hasLunch": true,
  "hasDinner": true,
  "category": "string",
  "parkingLotsQuantity": 0,
  "roomCategories": [
    "string"
  ]
}
```


Fonte: criado pelo autor

O retorno da API segue o mesmo formato conforme a figura 10, além dos padrões propostos na requisição de GET.

### 3.3.4 Atualização de hotéis

Para realizar a requisição, é preciso chamar a requisição com o método PATCH, no endpoint: /hotels/id. É importante ressaltar que id deve ser substituído pelo identificador do hotel. Além disso, é preciso enviar no body da requisição o json conforme a figura 13.

Figura 13: Retorno da busca de um hotel.


Fonte: criado pelo autor

O retorno da API segue o mesmo formato conforme a figura 10, além dos padrões propostos na requisição de GET.

### 3.3.5 Remoção de hotéis

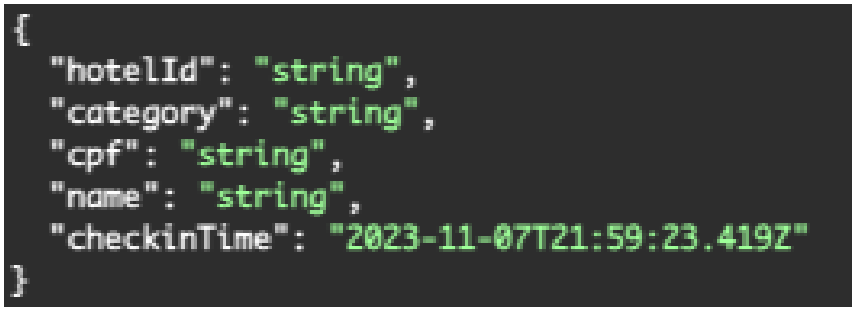
Para realizar a requisição, é preciso chamar a requisição com o método DELETE, no endpoint: /hotels/id. É importante ressaltar que id deve ser substituído pelo identificador do hotel.

O retorno da API segue o mesmo formato conforme a figura 10, além dos padrões propostos na requisição de GET

### 3.3.6 Checkin

Para realizar a requisição, é preciso chamar a requisição com o método POST, no endpoint: /hotels/checkin. Já como corpo da requisição, deve-se enviar conforme a figura 14.

Figura 14: Payload de envio para o checkin em um hotel.




Fonte: criado pelo autor

O retorno da API segue conforme a figura 15, além dos padrões já propostos nos outros endpoints.

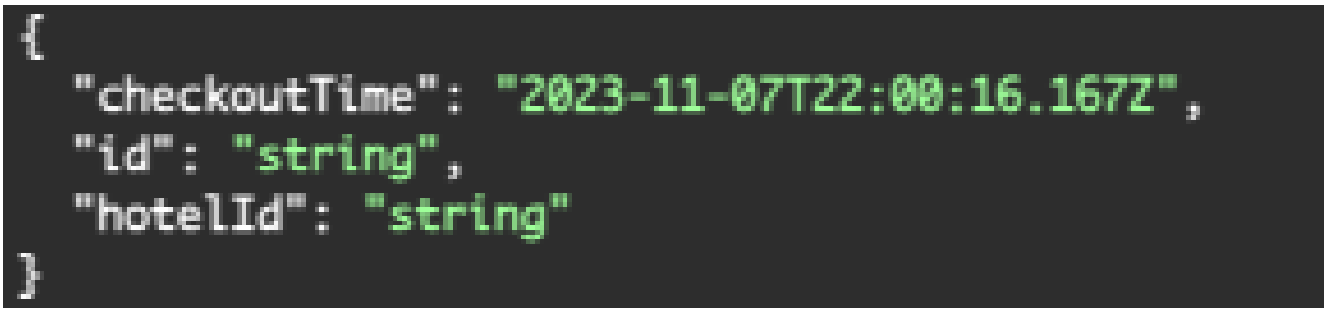
Figura 15: Retorno do checkin em um hotel.


Fonte: criado pelo autor

### 3.3.7 Checkout

Para realizar a requisição, é preciso chamar a requisição com o método POST, no endpoint: /hotels/checkout. Já como corpo da requisição, deve-se enviar conforme a figura 16.

Figura 16: Payload de envio para o checkin em um hotel.




Fonte: criado pelo autor

O retorno da API segue conforme a figura 15, além dos padrões já propostos nos outros endpoints.

### 3.3.8 Listar pacotes reservados

Para realizar a requisição, é preciso chamar a requisição com o método GET, no endpoint: /packages. Já como corpo da requisição, deve-se enviar como query string o cpf ou id do usuário.

O retorno da API segue conforme a figura 17, além dos padrões já propostos nos outros endpoints.

Figura 17: Retorno da listagem de pacotes.


Fonte: criado pelo autor

### 3.3.9 Reservar pacote turístico

Para realizar a requisição, é preciso chamar a requisição com o método POST, no endpoint: /packages. O payload de envio segue o padrão mostrado na figura 18.

Figura 18: Payload de envio para reservar um pacote turístico.


Fonte: criado pelo autor

O retorno da API segue conforme a figura 17, além dos padrões já propostos nos outros endpoints.

### 3.3.10 Cancelar pacote reservado

Para realizar a requisição, é preciso chamar a requisição com o método DELETE, no endpoint: /packages/:id. Em que o id seria o identificador do pacote turístico reservado.

O retorno da API segue conforme a figura 17, além dos padrões já propostos nos outros endpoints.

# 4 Notas de implantação e consumo

Para consumir a aplicação, primeiramente é necessário clonar o repositório no seguinte domínio: https://github.com/DuduAlmeida/facens.sistema.operacionais.embarcados. Essa clonagem pode ser realizada utilizando o Git e o comando "git clone https://github.com/DuduAlmeida/facens.sistema.operacionais.embarcados.git". Ao clonar o repositório, uma cópia local do projeto será obtida em seu computador.

Uma vez que o repositório foi clonado, é preciso navegar até a pasta do projeto no terminal. Para isso, abra um terminal ou prompt de comando e utilize o comando "cd af_jardini/webapp/impl/"para acessar a pasta correta onde o projeto foi clonado. Essa etapa é importante para garantir que você esteja no diretório adequado para executar os comandos subsequentes.

Após navegar até a pasta correta, é necessário executar o comando "yarn && yarn dev"no terminal. Esse comando iniciará o servidor local e permitirá que a aplicação seja executada no ambiente de desenvolvimento. Certifique-se de ter o gerenciador de pacotes Yarn instalado em seu sistema para poder executar esse comando corretamente.

Uma vez que o servidor local esteja em execução, será possível acessar a aplicação através do domínio http://localhost:4321/api em seu navegador. Ao acessar esse domínio, você poderá verificar as idls (Interface Description Languages) da aplicação. Essas idls fornecem informações sobre a interface da aplicação, como os métodos disponíveis, os parâmetros esperados e as respostas esperadas.

Contudo, para fazer as requisições de utlização, basta chamar o domínio http://localhost:4321 como base, e adicionar os endpoints mapeados na IDL.

Para facilitar os testes, o grupo preparou uma collection no postman, que ao instalar o aplicativo é possível clicar em importar e colocar o json da collection feita pelo grupo. A collecion encontra-se ao seguir o caminho: "af_jardini/webapp/idl/Hotel Api.postman_collection.json"

# Referências

PRESSMAN, Roger S. **Engenharia de software**. [S.l.]: McGraw Hill, 2010.

W3C. **Web Services Architecture**. 2004. Disponível em: ¡https://www.w3.org/TR/ws-arch/¿.