

Microarquitetura Back - End

A microarquitetura apresentada é uma aplicação back-end construída usando o framework Flask. A aplicação gerencia jornadas do herói e suas missões, permitindo a criação, atualização, exclusão e recuperação desses dados por meio de uma API RESTful.

Organização dos Componentes Principais

1. Flask App

Flask é criado na variável `app` utilizando a classe `Flask` e passando `__name__` como argumento. Este é o ponto de entrada da aplicação web e onde todas as rotas, configurações e funcionalidades são definidas.

```
from flask import Flask
app = Flask(__name__)
```

2. Routes

As rotas são URLs específicas que o aplicativo Flask pode manipular. No código, as rotas são definidas usando o decorador `@app.route()`. Cada rota define um ponto de extremidade de API que pode receber solicitações **HTTP GET, POST, PUT e DELETE**. As rotas principais para manipulação de jornadas e suas missões são:

`/journeys` (GET, POST): Para obter todas as jornadas e criar novas jornadas.

`/journeys/<journey_id>` (PUT, DELETE): Para atualizar e excluir jornadas específicas com base no ID.

`/journeys/<journey_id>/missions` (POST): Para adicionar uma nova missão a uma jornada específica.

`/journeys/<journey_id>/missions/<mission_id>` (PUT, DELETE): Para atualizar e excluir missões específicas em uma jornada.

```
@app.route('/journeys', methods=['GET', 'POST'])
def get_or_create_journeys():
    # Implementação para obter ou criar jornadas
    pass
```

```

@app.route('/journeys/<journey_id>', methods=['PUT', 'DELETE'])
def update_or_delete_journey(journey_id):
    # Implementação para atualizar ou excluir jornadas
    pass

@app.route('/journeys/<journey_id>/missions', methods=['POST'])
def create_mission(journey_id):
    # Implementação para adicionar uma nova missão a uma jornada
    pass

@app.route('/journeys/<journey_id>/missions/<mission_id>', methods=['PUT', 'DELETE'])
def update_or_delete_mission(journey_id, mission_id):
    # Implementação para atualizar ou excluir uma missão em uma jornada
    pass

```

3. Manipulação de Dados

As funções `load_db()` e `save_db()` são responsáveis por carregar e salvar os dados no arquivo JSON que atua como o banco de dados da aplicação. A função `get_next_id()` é utilizada para obter o próximo ID disponível para atribuir a novos elementos de dados.

```
db_file = './db.json'
```

```

def load_db():
    if os.path.exists(db_file):
        with open(db_file, 'r') as f:
            data = json.load(f)
    else:
        data = {'journeys': [], 'setted_journey': None}
    return data

def save_db(data):
    with open(db_file, 'w') as f:
        json.dump(data, f)

def get_next_id(entity_list):
    if not entity_list:
        return '1'
    else:
        return str(int(max(entity_list, key=lambda x: int(x['id']))['id']) + 1)

```

Armazenamento de dados

As jornadas e suas missões são armazenadas em um arquivo JSON (db.json), com as jornadas sendo uma lista de dicionários dentro de um dicionário principal. Além disso, foi adicionado um campo "setted_journey" para manter o controle da jornada definida.

Exemplo de estrutura de dados no arquivo db.json

```
{
  "journeys": [
    {
      "id": "1",
      "name": "Journey 1",
      "missions": [
        {"id": "1", "name": "Mission 1"},
        {"id": "2", "name": "Mission 2"}
      ]
    },
    {
      "id": "2",
      "name": "Journey 2",
      "missions": [
        {"id": "1", "name": "Mission 1"}
      ]
    }
  ],
  "setted_journey": "1"
}
```

Validações

O sistema realiza algumas validações, se concentrando principalmente em garantir que as jornadas e missões tenham os campos necessários e que as operações de criação e atualização sejam realizadas corretamente..

```
# Função para validar se uma jornada possui pelo menos uma missão
def validate_journey_payload(payload):
    if 'missions' not in payload or len(payload['missions']) == 0:
        abort(400, {'message': 'A journey must have at least one mission.'})
```

```
@app.route('/journeys', methods=['POST'])
def post_journey():
    data = load_db()
    new_journey = request.get_json()
    validate_journey_payload(new_journey)
    # Restante da implementação para criar uma nova jornada
```

Tratamento de Erros

O sistema trata erros retornando respostas apropriadas com códigos de status HTTP correspondentes e mensagens explicativas.

```
# Tratamento de erro para jornada não encontrada
@app.errorhandler(404)
def not_found(error):
    return jsonify({'error': 'Not found'}), 404
```

```
# Tratamento de erro para payload inválido
@app.errorhandler(400)
def bad_request(error):
    return jsonify({'error': 'Bad request'}), 400
```

Rotas (CRUD)

1. Create (Criar)

- Rota:

- *POST /journeys*: Cria uma nova jornada do herói.
- *POST /journeys/<journey_id>/missions*: Cria uma nova missão em uma jornada específica.

- Retorno:

- Objeto JSON com uma mensagem de sucesso e o ID da jornada criada para */journeys*.

- Objeto JSON com uma mensagem de sucesso e o ID da missão criada para */journeys/<journey_id>/missions*.

- Descrição Geral:**

- As rotas de criação permitem adicionar novas jornadas do herói e missões associadas ao sistema.

2. READ(Ler)

- Rota:**

- *GET /journeys*: Retorna todas as jornadas do herói existentes.
 - *GET /journeys/<journey_id>*: Retorna os detalhes de uma jornada específica do herói.
 - *GET /journeys/<journey_id>/missions*: Retorna todas as missões de uma jornada específica.
 - *GET /setted_journey*: Retorna a jornada atualmente definida.

- Retorno:**

- Objeto JSON contendo as jornadas do herói para */journeys*.
 - Objeto JSON contendo os detalhes da jornada solicitada para */journeys/<journey_id>*.
 - Objeto JSON contendo as missões da jornada solicitada para */journeys/<journey_id>/missions*.
 - Objeto JSON contendo os detalhes da jornada atualmente definida para */setted_journey*.

- Descrição Geral:**

- As rotas de leitura permitem visualizar as jornadas do herói, suas missões associadas e a jornada atualmente definida no sistema.

3. Update (Atualizar):

- Rota:**

- *PUT /journeys/<journey_id>*: Atualiza os detalhes de uma jornada existente.
 - *PUT /journeys/<journey_id>/missions/<mission_id>*: Atualiza os detalhes de uma missão existente em uma jornada específica.
 - *PUT /setted_journey/<journey_id>*: Define uma nova jornada como a jornada atualmente definida.

- Retorno:**

- Objeto JSON com uma mensagem de sucesso e os detalhes da jornada atualizada para */journeys/<journey_id>*.

- Objeto JSON com uma mensagem de sucesso e os detalhes da missão atualizada para */journeys/<journey_id>/missions/<mission_id>*.
- Objeto JSON com uma mensagem de sucesso ao definir a nova jornada para */setted_journey/<journey_id>*.

- Descrição Geral:

- As rotas de atualização permitem modificar as informações das jornadas do herói, suas missões associadas e a jornada atualmente definida no sistema.

4. Delete (Excluir)

- Rota:

- *DELETE /journeys/<journey_id>*: Exclui uma jornada do herói existente.
- *DELETE /journeys/<journey_id>/missions/<mission_id>*: Exclui uma missão específica de uma jornada existente.

- Retorno:

- Objeto JSON com uma mensagem de sucesso ou uma mensagem de erro se a jornada não for encontrada para */journeys/<journey_id>*.
- Objeto JSON com uma mensagem de sucesso ou uma mensagem de erro se a missão não for encontrada para */journeys/<journey_id>/missions/<mission_id>*.

- Descrição Geral:

- As rotas de exclusão permitem remover jornadas do herói, suas missões associadas e a jornada atualmente definida do sistema.