Microarquitetura Back - End

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

Organização dos Componentes Principais

1. **Flask App**
   O aplicativo Flask é criado na variável `app` usando `Flask(__name__)`. Ele é a base do aplicativo web e é onde todas as rotas, configurações e funcionalidades são definidas.
2. **Routes**
   As rotas são URLs específicas que o aplicativo Flask pode manipular. No código fornecido, 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 tarefas são:

- `/task` (GET, POST): Para obter todas as tarefas e criar novas tarefas.
- `/task/<id>` (PUT, DELETE): Para atualizar e excluir tarefas específicas com base no ID.
- As rotas para manipulação de jornadas do herói são:
- `/jornadas` (GET, POST): Para obter todas as jornadas e criar novas jornadas.
- `/jornadas/<nome_jornada>` (PUT, DELETE, GET): Para atualizar, excluir e obter jornadas específicas com base no nome.

3. **Manipulação de Dados**

Existem várias funções definidas para manipular os dados das tarefas e jornadas:

- `load_db()`: Carrega os dados do banco de dados JSON.
- `save_db()`: Salva os dados no banco de dados JSON.
- `initialize_tasks()`: Inicializa as tarefas se o banco de dados estiver vazio.

- `validate_jornada_payload(payload)`: Valida se uma jornada tem pelo menos uma tarefa.

Já o banco de dados é representado por um arquivo JSON localizado em `./db.json`. Ele armazena todas as tarefas e jornadas do herói. As funções `load_db()` e `save_db()` são responsáveis por carregar e salvar os dados neste arquivo.

Armazenamento de dados

As tarefas e jornadas são armazenadas em um arquivo JSON (`db.json`), que atua como um banco de dados simples.

linhas 11 - 23

```
def load_db():
    if os.path.exists(db_file):
        with open(db_file, 'r') as f:
            tasks = json.load(f)
        if not isinstance(tasks, dict):
            tasks = {'disponiveis': [], 'emAndamento': [], 'concluidas': [], 'jornadas': {}}
    else:
        tasks = {'disponiveis': [], 'emAndamento': [], 'concluidas': [], 'jornadas': {}}
    return tasks

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

Validações

O sistema realiza algumas validações, como garantir que uma jornada do herói tenha pelo menos uma tarefa ao ser criada.

Também verifica se uma tarefa ou jornada a ser atualizada ou excluída existe antes de realizar a operação.

linhas 25 - 28

```
def validate_jornada_payload(payload):
    if 'tarefas' not in payload or not payload['tarefas']:
```

abort(400, {'message': 'Uma jornada do herói deve ter pelo menos uma tarefa.'})

Tratamento de Erros

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

```
@app.errorhandler(404)
@app.errorhandler(400)
def error_handler(error):
    return jsonify({'error': error.description}), error.code
```

Rotas (CRUD)

1. **Create (Criar):**
   Rota:
   - *POST /task*: Cria uma nova tarefa.
   - *POST /jornadas*: Cria uma nova jornada do herói.
   - Função:
   - *criar_task()*: Cria uma nova tarefa no sistema.
   - *criar_jornada()*: Cria uma nova jornada do herói no sistema.
   Retorno:
   - Objeto JSON com uma mensagem de sucesso e o ID da tarefa criada para /task.
   - Objeto JSON com uma mensagem de sucesso e o nome da jornada criada para /jornadas.
   Descrição Geral:
   - As rotas de criação permitem adicionar novas tarefas e jornadas do herói ao sistema.

2. Read (Ler):
   Rota:
   - *GET /task*: Retorna todas as tarefas existentes.
   - *GET /jornadas:* Retorna todas as jornadas do herói existentes.
   - *GET /jornadas/<nome_jornada>*: Retorna os detalhes de uma jornada específica.
   Função:

- *get_tasks()*: Retorna todas as tarefas existentes no sistema.
- *get_jornadas()*: Retorna todas as jornadas do herói no sistema.
- *get_jornada()*: Retorna os detalhes de uma jornada específica do herói.

Retorno:

- Objeto JSON contendo as tarefas organizadas por status para /task.
- Objeto JSON contendo as jornadas do herói para /jornadas.
- Objeto JSON contendo os detalhes da jornada solicitada para */jornadas/<nome_jornada>*.

Descrição Geral:

- As rotas de leitura permitem visualizar as tarefas e as jornadas do herói existentes no sistema.

3. Update (Atualizar):

Rota:

- *PUT /task/<id>*: Atualiza os detalhes de uma tarefa existente.
- *PUT /jornadas/<nome_jornada>*: Atualiza as tarefas de uma jornada existente.

Função:

- *update_task()*: Atualiza os detalhes de uma tarefa existente no sistema.
- *update_jornada()*: Atualiza as tarefas de uma jornada existente no sistema.

Retorno:

- Objeto JSON com uma mensagem de sucesso e os detalhes da tarefa atualizada para */task/<id>*.
- Objeto JSON com uma mensagem de sucesso e as tarefas atualizadas da jornada para */jornadas/<nome_jornada>*.

Descrição Geral:

- As rotas de atualização permitem modificar as informações das tarefas e das jornadas do herói existentes no sistema.

4**. Delete (Excluir):**

Rota:

- *DELETE /task/<id>*: Exclui uma tarefa existente.
- *DELETE /jornadas/<nome_jornada>*: Exclui uma jornada do herói existente.

Função:

- *delete_task()*: Exclui uma tarefa do sistema.
- *delete_jornada()*: Exclui uma jornada do herói do sistema.

Retorno:

- Objeto JSON com uma mensagem de sucesso ou uma mensagem de erro se a tarefa não for encontrada para */task/<id>*.

- Objeto JSON com uma mensagem de sucesso ou uma mensagem de erro se a jornada não for encontrada para */jornadas/<nome_jornada>*.

Descrição Geral:

- As rotas de exclusão permitem remover tarefas e jornadas do herói do sistema.