Documentação Técnica

1. Introdução geral ao sistema

O sistema é composto por um backend em Node.js, utilizando o Express como framework para a construção de uma API RESTful. Para o armazenamento persistente de dados, é empregado o banco de dados NoSQL MongoDB. A aplicação é encapsulada e executada em containers usando Docker, garantindo um ambiente controlado para desenvolvimento e produção.

1.1 Arquitetura

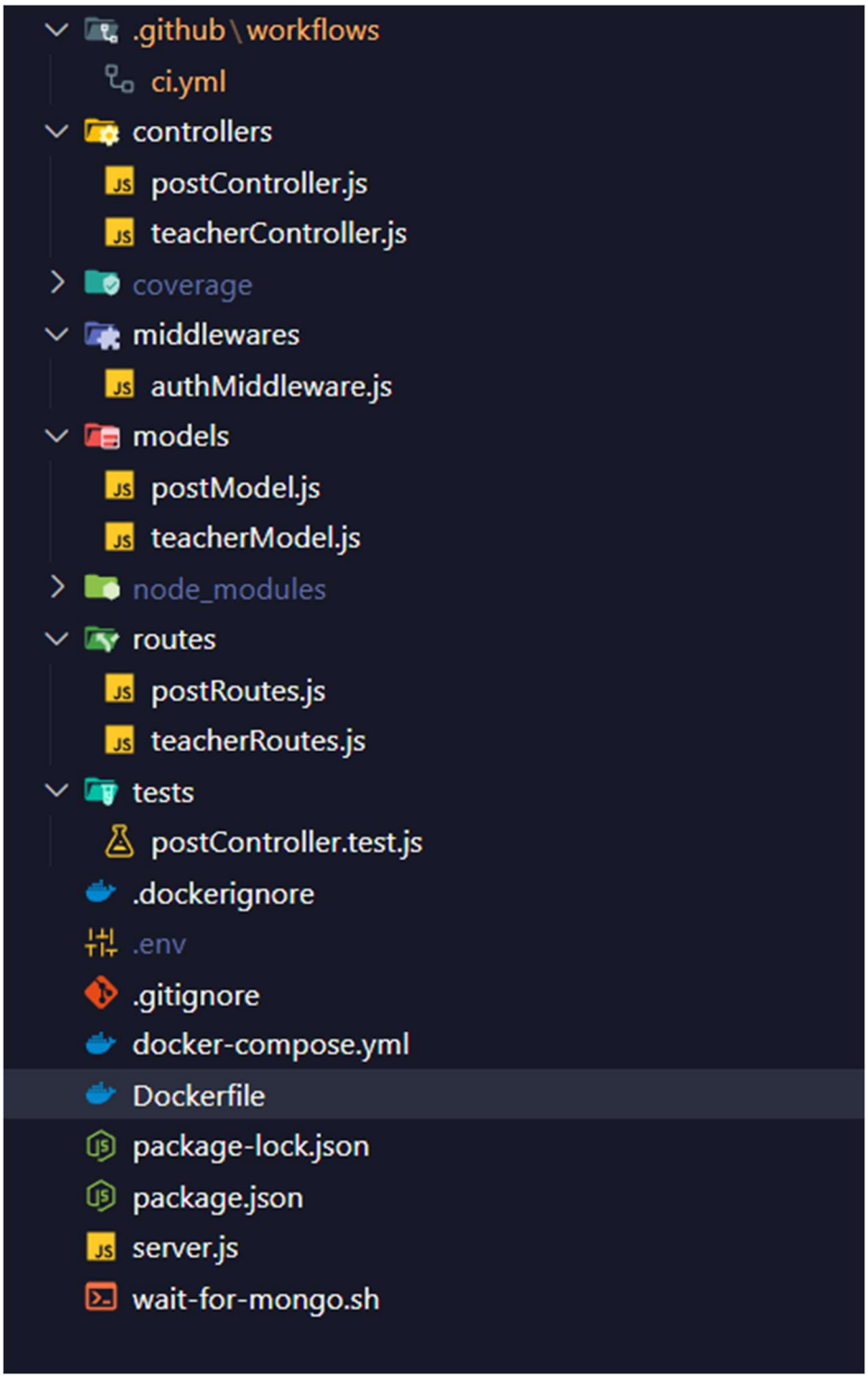
- **Backend**
  - **Node.js + Express:** Utilizados para implementação do servidor e das APIs;
  - **JWT:** JASON Web Token implementado para garantir autenticação e autorização de usuários, controlando o acesso a recursos da API;
  - **MongoDB:** Responsável pela persistência de dados.
- **Infraestrutura**
  - **Docker:** Permite criar um ambiente isolado e controlado para desenvolvimento e produção;
  - **Docker Compose**: Utilizado para gerenciar os containers do backend e do banco de dados, garantindo escalabilidade e facilidade de configuração.

2. Ambiente de execução

A aplicação está configurada para rodar em um ambiente de containers. O Docker compose é utilizado para orquestrar os serviços, incluindo o banco de dados MongoDB e o container da aplicação, facilitando o gerenciamento e a interação entre os componentes do sistema.

2.1 Estrutura de diretórios

A estrutura do projeto:



2.2 Dockerfile

```
# Especifica a versão do node
FROM node:20.12.2

# Defini o diretório de trabalho
WORKDIR /blog-api

# Copia o package.json e package-lock.json
COPY package*.json ./

# Instala as dependências do projeto
RUN npm install

# Instala o netcat (nc)
RUN apt-get update && apt-get install -y netcat-openbsd

# Copia todo o código da aplicação
COPY . .

# permissão de execução ao script wait-for-mongo.sh
RUN chmod +x wait-for-mongo.sh

# Porta que a aplicação vai rodar
EXPOSE 3000

# Comando para rodar o servidor com o nodemon
CMD ["node", "server.js"]
```

2.3 Docker compose

O arquivo docker-compose.yml é responsável pela configuração dos containers. Exemplo:

```yaml
version: '3.8'

services:
  mongo:
    image: mongo
    container_name: mongo_db
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
    command: ["mongod", "--auth"]
    environment:
      MONGO_INITDB_ROOT_USERNAME: usuario1
      MONGO_INITDB_ROOT_PASSWORD: 123456
      MONGO_LOG_LEVEL: error

  app:
    build:.
    container_name: blog_api
    ports:
      - "3000:3000"
    depends_on:
      - mongo
    environment:
      MONGO_URI: mongodb://usuario1:123456@mongo_db:27017/blog?authSource=admin
      JWT_SECRET: chaveSecreta1245Boa
    volumes:
      - .:/blog-api
      - ./wait-for-mongo.sh:/wait-for-mongo.sh
    entrypoint: ["sh", "/wait-for-mongo.sh", "node", "server.js"]

volumes:
  mongo_data:
```

- **Banco de dados:** MongoDB armazena os dados em (mongo_data), para a persistência dos mesmos, inicializando na porta 27017;
- **Aplicação:** Constrói a imagem a partir do Dockerfile e inicia na porta 3000;
- **Dependência:** Só permite que aplicação inicie após o banco de dados ter inicializado.

3. APIs

As APIs são construídas utilizando o Express e permitem a manipulação de dados dos usuários e posts. Exemplos técnicos:

3.1 Criar novo usuário

POST http://localhost:3000/teacher/register

Body:

```
{
    "email": "teste@gmail.com",
    "password": "teste123"
}
```

Retorno:

```
{
    "message": "Professor registrado com sucesso"
}
```

3.2 Login de usuário

POST http://localhost:3000/teacher/register/login

Body:

```
{
    "email": "teste@gmail.com",
    "password": "teste123"
}
```

Retorno:

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3MGRhOWRlYzNmYzk0MGQ4ZGMxNzk1NyIsImlhdCI6MTcyODk0ODc2NywiZXhwIjoxNzI4OTUyMzY3fQ.87xj5VUQn2PkUPAPMRDmqlLZE1HrcZID9PIVugkUO5Y"
}
```

3.3 Busca de posts

GET http://localhost:3000/posts/

Retorno:


```
[
    {
        "_id": "67055941fdf1d018f9381435",
        "title": "Noa pode",
        "content": "Funcionar",
        "author": "vai?",
        "createdAt": "2024-10-08T16:09:37.566Z",
        "updatedAt": "2024-10-08T16:09:37.566Z",
        "__v": 0
    }
]
```


3.4 Busca de post específico

GET http://localhost:3000/posts/search?q={título ou conteúdo}

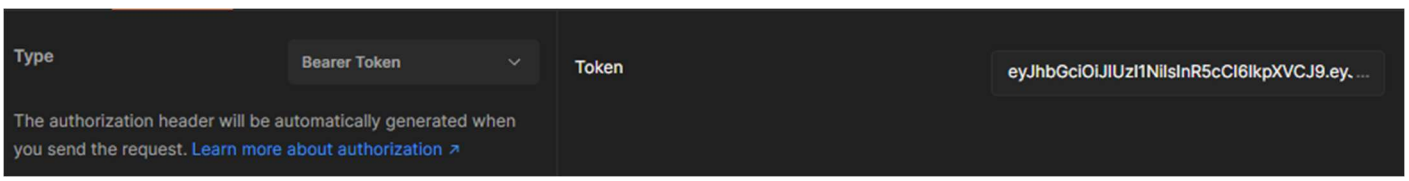
Retorno:


```
[
    {
        "_id": "670dabf4c3fc940d8dc1795a",
        "title": "Testando, alterado",
        "content": "Funcionar, atulizando",
        "author": "Eu, novo",
        "createdAt": "2024-10-14T23:40:36.110Z",
        "updatedAt": "2024-10-14T23:44:54.883Z",
        "__v": 0
    }
]
```


3.5 Criação de posts

POST http://localhost:3000/posts/

Header:



Body:

```
{
    "title": "Testando",
    "content": "Funcionar",
    "author": "Eu"
}
```

3.6 Alteração de posts

PUT http://localhost:3000/posts/:id

Header:

Body:

```
{
    "title": "Testando, alterado",
    "content": "Funcionar, atulizando",
    "author": "Eu, novo"
}
```

3.7 Deletar post

DELETE http://localhost:3000/posts/:id

Header:

Retorno:

```
{
    "message": "Post excluido com sucesso"
}
```

4. Autenticação e Segurança

A autenticação é baseada em JWT. Quando um usuário (professor) faz login, um token é gerado e enviado como resposta. Este token deve ser utilizado nas próximas requisições para garantir que o usuário tenha permissão para acessar recursos protegidos.

4.1 Middleware de autenticação

```
5. const jwt = require('jsonwebtoken');
6.
7. const secret = process.env.JWT_SECRET;
8.
9. function authMiddleware(req, res, next){
10.         const authHeader = req.header('Authorization');
11.         if(!authHeader) return res.status(401).json({ error: 'Acesso
   negado. Token não fornecido'});
12.         const token = authHeader.replace('Bearer ', '');
13.
14.         try{
15.             const decoded = jwt.verify(token, secret);
16.             req.teacherId = decoded.id;
17.             next();
18.
19.         }catch(err){
20.             res.status(400).json({ error: 'Token invalido'});
21.         }
22.     };
23.
24.     module.exports = authMiddleware;
```