material de apoio

Site: Geração Tech
Curso: Formação em Desenvolvedor Web - Online
Livro: material de apoio

Impresso por: JOÃO VITOR DE MELO FREITAS
Data: sábado, 17 ago. 2024, 14:57

Índice

1. Iniciando os Testes com Jest

Introdução

Professor, os testes só podem ser realizados com um projeto pronto?

Não! Na verdade, toda vez que começamos um projeto e vamos modularizando, fazendo o primeiro CRUD, já podemos iniciar os testes com o Jest. Dessa forma, o Jest já fica configurado desde o início, e vamos executando os testes conforme finalizamos as integrações.

Normalmente, existe um processo de entrega incremental com o cliente, onde vamos entregando cada pedaço do projeto à medida que fica pronto. Por exemplo, entregamos primeiro a tela de login, depois a funcionalidade de troca de produto, cadastro de estoque, etc. Nesse projeto, vamos configurar o Jest desde o início e começar a configurar os testes para todo o projeto.

1.1. Configurando o Jest no Projeto

1. **Instalação do Jest**:
   - No console, rode o comando:

     bash
     Copiar código

     ```
     npm install jest --save-dev
     ```

   - Isso vai instalar o Jest como uma dependência de desenvolvimento.
2. **Configuração do Script de Teste**:
   - No arquivo `package.json`, substitua o script de teste padrão pelo Jest:

     json
     Copiar código

     ```
     "scripts": { "test": "jest" }
     ```

3. **Criando a Estrutura de Testes**:
   - Crie uma pasta chamada `tests` no projeto. É nessa pasta que todos os arquivos de teste serão armazenados.

Testando o Servidor

Agora, vamos testar se nosso servidor está funcionando corretamente. Primeiro, certifique-se de que o projeto está rodando com `npm start`. Ele deve estar acessível na porta 3000.

Vamos criar um arquivo de teste para verificar se o servidor está rodando como esperado.

javascript
Copiar código

```
// tests/server.test.js test('Teste de servidor', async () => { const response = await fetch('http://localhost:3000');
expect(response.status).toBe(200); // Verifica se o status retornado é 200 (OK) });
```

Rodando `npm test`, o Jest deve executar esse teste e verificar se o servidor está retornando o status 200, indicando que ele está rodando corretamente.

Verificando Respostas do Servidor

Podemos também testar o conteúdo da resposta do servidor. Por exemplo, podemos verificar se o nome que aparece na resposta é o que esperamos:

javascript
Copiar código

```
test('Verifica o nome na resposta', async () => { const response = await fetch('http://localhost:3000'); const text = await
response.text(); expect(text).toBe('Nome Esperado'); // Substitua 'Nome Esperado' pelo nome que você espera });
```

Aqui, estamos capturando o texto da resposta e verificando se ele corresponde ao valor esperado.

Testando Autenticação e CRUDs

Nosso projeto tem rotas públicas e privadas. Por exemplo, temos uma rota de login que espera `username` e `password` e, se tudo estiver correto, retorna um token JWT. Podemos criar testes para verificar se a autenticação está funcionando corretamente:

javascript
Copiar código

```
// tests/auth.test.js test('Teste de login válido', async () => { const response = await
fetch('http://localhost:3000/login', { method: 'POST', headers: { 'Content-Type': 'application/json' }, body:
JSON.stringify({ username: 'usuario', password: 'senha' }) }); const data = await response.json();
expect(response.status).toBe(200); expect(data.token).toBeDefined(); // Verifica se o token foi retornado }); test('Teste de
login inválido', async () => { const response = await fetch('http://localhost:3000/login', { method: 'POST', headers: {
'Content-Type': 'application/json' }, body: JSON.stringify({ username: 'usuario_invalido', password: 'senha_errada' }) });
expect(response.status).toBe(401); // Verifica se o status é 401 (não autorizado) });
```

Com esses testes, garantimos que o sistema de autenticação está funcionando corretamente, tanto para logins válidos quanto para logins inválidos.

Organizando os Testes

Podemos organizar os testes em arquivos separados para cada funcionalidade. Por exemplo:

- `tests/user.test.js` para testar o CRUD de usuários.
- `tests/product.test.js` para testar o CRUD de produtos.
- `tests/order.test.js` para testar o CRUD de pedidos.

Cada arquivo conterá os testes específicos para aquela funcionalidade. Isso torna o processo de teste mais organizado e facilita a manutenção dos testes ao longo do desenvolvimento do projeto.

Conclusão

Neste ponto, já configuramos o Jest no nosso projeto e começamos a escrever testes básicos para verificar se o servidor está rodando, se a autenticação está funcionando e se as respostas estão corretas.

A partir daqui, vamos continuar criando mais testes para cobrir todas as funcionalidades do nosso backend.

Estudem a documentação do Jest e continuem praticando. Até a próxima aula!

1.2. Vídeo Aula

dia 46 jest p1 converted

1.3. Revisando o que já foi feito

Na última aula, configuramos o Jest e criamos um teste para verificar se o servidor está rodando corretamente. Nosso servidor está funcionando e agora vamos avançar para testar a parte de usuários.

Utilizando Axios para Requisições

Neste projeto, vamos utilizar o Axios, uma biblioteca que facilita a realização de requisições HTTP. Apesar de ser mais comum no frontend, também podemos usá-la no backend. Vamos instalá-la e configurá-la:

1. **Instalando Axios**:

   bash
   Copiar código

   ```
   npm install axios
   ```

2. **Configurando Axios**: Vamos criar um arquivo `api.js` para configurar o Axios:

   javascript
   Copiar código

   ```
   const axios = require('axios'); const api = axios.create({ baseURL: 'http://localhost:3000', }); module.exports = api;
   ```

   Isso nos permitirá utilizar `api` em nossos testes sem precisar repetir a URL base em cada requisição.

Testando a Autenticação

Agora, vamos configurar um teste para autenticação de usuários:

1. **Importando a API**: No arquivo `tests/user.test.js`, importe a configuração da API:

   javascript
   Copiar código

   ```
   const api = require('../api');
   ```

2. **Criando o Teste de Autenticação**: Vamos testar se o login funciona corretamente:

   javascript
   Copiar código

   ```
   test('Autenticação de usuário', async () => { const response = await api.post('/login', { username: 'Max', password:
   '1234' }); expect(response.data.message).toBe('Login e senha corretos'); expect(response.data.token).toBeDefined(); //
   Verifica se o token foi gerado });
   ```

   Ao executar o comando `npm test`, o Jest deve mostrar que o teste passou, indicando que a autenticação está funcionando corretamente.

Testando Rotas Protegidas

Agora que temos o token após a autenticação, podemos utilizá-lo para testar rotas protegidas:

1. **Testando Acesso Não Autorizado**: Primeiro, vamos verificar se o acesso a uma rota protegida sem o token retorna um erro:

   javascript
   Copiar código

```
test('Acesso não autorizado a /users sem token', async () => { try { await api.get('/users'); } catch (error) {
expect(error.response.status).toBe(401); // Verifica se o status retornado é 401 (não autorizado) } });
```

2. **Testando Acesso Autorizado**: Depois, usamos o token gerado na autenticação para acessar uma rota protegida:

javascript
Copiar código

```
test('Acesso autorizado a /users com token', async () => { const loginResponse = await api.post('/login', { username:
'Max', password: '1234' }); const token = loginResponse.data.token; const response = await api.get('/users', { headers:
{ Authorization: `Bearer ${token}` } }); expect(response.status).toBe(200); // Verifica se o acesso foi autorizado
expect(response.data).toBeDefined(); // Verifica se os dados dos usuários foram retornados });
```

Criando um Novo Usuário

Vamos criar um teste para adicionar um novo usuário ao banco de dados:

1. **Teste para Criação de Usuário**: Criamos um novo teste para verificar se a criação de usuário funciona corretamente:

javascript
Copiar código

```
test('Criação de novo usuário', async () => { const response = await api.post('/users', { username: 'Jack', password:
'1234', email: 'jack@example.com' }); expect(response.data.message).toBe('Usuário criado com sucesso'); });
```

Novamente, rodamos `npm test` e verificamos se o Jest retorna tudo verde, indicando que os testes passaram.

Considerações Finais

Lembre-se de descomentar qualquer código de segurança que tenha sido comentado para testes, como a verificação de token nas rotas. Isso é importante para garantir que a aplicação estará protegida quando for implantada.

Os testes automatizados são fundamentais para garantir a qualidade e o funcionamento correto da aplicação. Com esses testes, podemos identificar e corrigir erros antes que eles cheguem ao cliente.

Continuem praticando a configuração e criação de testes com Jest. Isso vai enriquecer bastante o conhecimento de vocês e garantir que as aplicações entregues tenham alta qualidade.

Até pessoal! Valeu!

1.4. Vídeo Aula

dia 46 jest p2 converted