Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Unidade Curricular: Engenharia de Software II

Relatório Relativo ao Épico 2

Tema: Testes Black Box e White Box

Realizado por: João Laranjeiro– 23904

Rodrigo Almeida– 23095

Gonçalo Fonseca– 23911

Leandro Dias– 23028

Viseu, 2024

Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Relatório relativo ao Épico 2

Curso de Licenciatura em Engenharia Informática

Unidade Curricular de Engenharia de Software

Testes Black Box e White Box

Ano Letivo 2023/24

Viseu, 2024

ÍNDICE

Índice de Figuras

Índice de tabelas

1. Introdução

No âmbito do Épico 2 da Unidade Curricular de Engenharia de Software 2, foi essencial realizar testes Black Box e White Box para verificar e testar as funcionalidades do código. Começouse por implementar uma API, assegurando que esta atendesse a todos os requisitos e funcionalidades necessárias para a execução adequada dos testes.

Visando garantir uma abordagem eficiente, este trabalho estava separado em 3 *sprints,* sendo estes semanais. Cada *sprint* representava uma parte do trabalho, permitindo assim a realização do trabalho passo a passo, garantindo uma análise minuciosa e uma cobertura de testes abrangente.

Para isto foi utilizado um documento Excel para documentar todos os testes realizados. Este serviu como base para visualizar todas as etapas e resultados dos testes realizados, facilitando toda a análise e acompanhamento do processo de verificação das funcionalidades.

2. API em Node.js

Este capítulo explora a criação de uma API em Node.js durante o Sprint 1. A API foi desenvolvida utilizando a biblioteca Express.js, uma framework *web* minimalista para receber e rotear pedidos HTTP.

A API foi projetada para lidar com a gestão de utilizadores e pedidos (*orders*). Para atender a estes requisitos, foram criados dois controladores separados: um para utilizadores e outro para pedidos. Além disso, foram implementadas várias rotas para cada entidade, conforme detalhado na Tabela 1.

Método	**Rota**	**Descrição**
POST	/user	Criar de utilizador
POST	/login	Login
GET	/user	Obter dados do utilizador atual
PUT	/user	Atualizar dados do utilizador atual
POST	/order	Criar de pedido
GET	/order/{id}	Obter dados de um pedido

Table 1 Rotas para cada entidade

2.1. Controlo de Acesso Baseado em Funções

Cada utilizador pode ter um cargo específico, e cada cargo possui determinadas permissões para acessar os diferentes endpoints. Para definir essas permissões, foi seguido o padrão de RBAC (*Role-Based Access Control*).

2.2. Autenticação com *JWT*

Para a autenticação, foi utilizado um sistema de JWT (*JSON Web Tokens*) com uma duração máxima de 14 dias. Esse mecanismo garante a segurança das operações realizadas na API, permitindo que apenas utilizadores autenticados possam acessar os recursos protegidos.

2.3. Validação de dados de entrada

A API foi implementada para retornar dados apenas no formato JSON. Por esse motivo, o cabeçalho Accept está a ser validado para garantir que apenas pedidos com o cabeçalho Accept adequado sejam processados corretamente.

Além disso, todos os *endpoints* que recebem dados no corpo do pedido estão a ser validados com a biblioteca Zod, que é uma biblioteca de validação de esquemas que oferece uma abordagem declarativa e baseada em esquemas para validar dados de entrada. Algumas vantagens da utilização do Zod incluem:

- Validação de dados robusta e segura;
- Suporte para tipos complexos e aninhados;
- Mensagens de erro informativas;
- Desempenho otimizado.

Ao utilizar o Zod, a API garante que apenas dados válidos sejam processados, o que melhora a segurança e a confiança do sistema.

3. Testes *Black Box*

Este capítulo aborda o desenvolvimento de testes *black box* em função do Sprint 2. Estes testes, também conhecidos como testes de caixa preta, são uma abordagem de teste de software onde se avalia a funcionalidade do sistema sem conhecimento prévio sobre a estrutura interna do código.

O foco está nas entradas e saídas do *software*, verificando se o sistema responde corretamente a uma variedade de condições de entrada.

3.1. Implementação

Na sequência destes conceitos, os testes *black box* foram implementados para várias funcionalidades do sistema, conforme ilustrado na imagem fornecida. A imagem mostra a estrutura de diretórios de testes, incluindo diferentes endpoints da API que foram testados.

Aqui está uma breve descrição dos principais componentes testados:

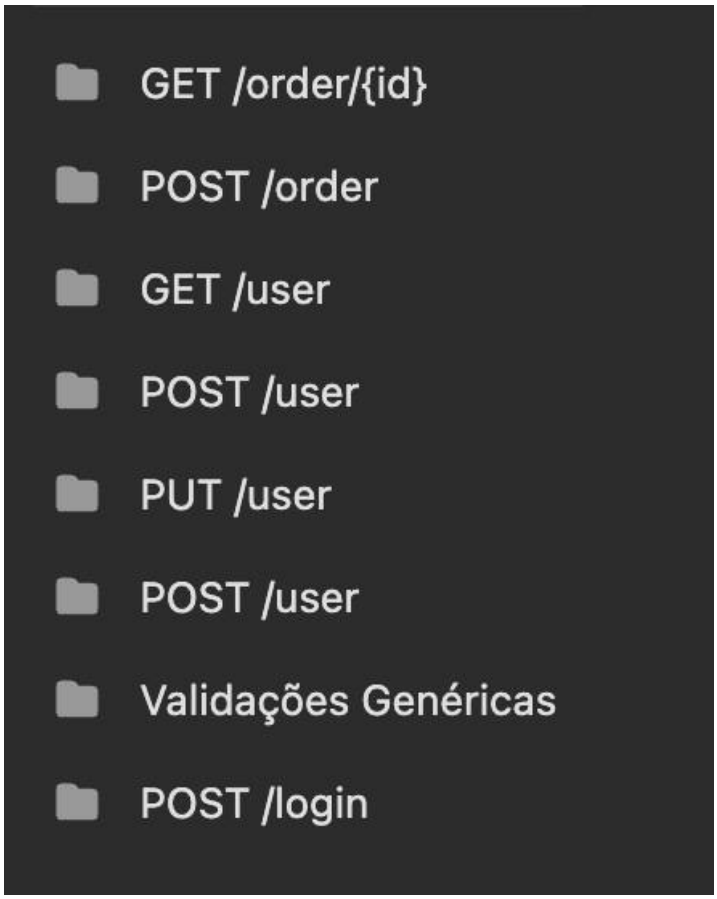

Figura 3-1 Componentes Testados

GET /order/{id}:

- TestBlackBoxGetOrderInvalidID: Teste para verificar a resposta do sistema quando um ID inválido é fornecido para obter uma ordem.
- TestBlackBoxGetOrderNotFound: Teste para verificar a resposta quando o ID fornecido não corresponde a nenhuma ordem existente.
- TestBlackBoxGetOrderFromOtherUserAdmin: Teste para verificar se um usuário administrador pode obter ordens de outros usuários.
- TestBlackBoxGetOrderFromOtherUserNotAdmin: Teste para verificar se um usuário não administrador não pode obter ordens de outros usuários.
- TestBlackBoxGetOrderSuccess: Teste para verificar a resposta bem-sucedida ao obter uma ordem existente com um ID válido.

POST /order:

- TestBlackBoxCreateOrderMissingField: Teste para verificar a resposta quando um campo obrigatório está faltando ao criar uma nova ordem.
- TestBlackBoxCreateOrderNullParameter: Teste para verificar a resposta quando um parâmetro nulo é fornecido ao criar uma nova ordem.
- TestBlackBoxCreateOrderInvalidParameter: Teste para verificar a resposta quando um parâmetro inválido é fornecido ao criar uma nova ordem.
- TestBlackBoxCreateOrderSuccess: Teste para verificar a resposta bem-sucedida ao criar uma nova ordem com dados válidos.

GET /user:

- TestBlackBoxGetUserSuccess: Teste para verificar a resposta bem-sucedida ao obter informações de um usuário autenticado.
- TestBlackBoxGetUserNotLoggedIn: Teste para verificar a resposta quando um usuário não autenticado tenta obter informações de usuário.
- TestBlackBoxGetUserTokenExpired: Teste para verificar a resposta quando o token de autenticação do usuário expirou.

POST /user (Criação de Usuários):

- TestBlackBoxCreateUserMissingBodyField: Teste para verificar a resposta quando um campo obrigatório está faltando ao criar um novo usuário.
- TestBlackBoxCreateUserNullParameter: Teste para verificar a resposta quando um parâmetro nulo é fornecido ao criar um novo usuário.
- TestBlackBoxCreateUserInvalidParameter: Teste para verificar a resposta quando um parâmetro inválido é fornecido ao criar um novo usuário.
- TestBlackBoxCreateUserUsernameTaken: Teste para verificar a resposta quando o nome de usuário fornecido já está em uso.
- TestBlackBoxCreateUserSuccess: Teste para verificar a resposta bem-sucedida ao criar um novo usuário com dados válidos.

PUT /user (Atualização de Usuários):

- TestBlackBoxUpdateUserSamePassword: Teste para verificar a atualização de informações do usuário quando a nova senha é igual à senha antiga.
- TestBlackBoxUpdateUserNullParameter: Teste para verificar a resposta quando um parâmetro nulo é fornecido ao atualizar um usuário.
- TestBlackBoxUpdateUserInvalidParameter: Teste para verificar a resposta quando um parâmetro inválido é fornecido ao atualizar um usuário.
- TestBlackBoxUpdateUserUsernameTaken: Teste para verificar a resposta quando o novo nome de usuário fornecido já está em uso por outro usuário.
- TestBlackBoxUpdateUserSuccess: Teste para verificar a resposta bem-sucedida ao atualizar informações de um usuário com dados válidos.

POST /login (Autenticação):

- TestBlackBoxLoginMissingBodyField: Teste para verificar a resposta quando um campo obrigatório está faltando ao fazer login.
- TestBlackBoxLoginUserNotFound: Teste para verificar a resposta quando o nome de usuário fornecido não existe.
- TestBlackBoxLoginNullParameter: Teste para verificar a resposta quando um parâmetro nulo é fornecido ao fazer login.
- TestBlackBoxLoginInvalidParameter: Teste para verificar a resposta quando um parâmetro inválido é fornecido ao fazer login.

- TestBlackBoxLoginSuccess: Teste para verificar a resposta bem-sucedida ao fazer login com credenciais válidas.
- TestBlackBoxLoginWrongPassword: Teste para verificar a resposta quando uma senha incorreta é fornecida.

Validações Genéricas:

- TestBlackBoxLoginMethod: Teste relacionado à validação do método HTTP utilizado para login.
- TestBlackBoxLoginSmallPassword: Teste para verificar a validação de comprimento mínimo de senha.
- TestBlackBoxLoginSmallUsername: Teste para verificar a validação de comprimento mínimo do nome de usuário.
- TestBlackBoxLoginToBigPassword: Teste para verificar a validação de comprimento máximo de senha.
- TestBlackBoxLoginToBigUsername: Teste para verificar a validação de comprimento máximo de nome de usuário.
- TestBlackBoxUnexistentRoute: Teste para verificar a resposta do sistema quando uma rota inexistente é acessada.
- TestBlackBoxHeaderAcceptAny: Teste relacionado à validação do cabeçalho "Accept" com valor "*/*".
- TestBlackBoxHeaderAcceptJSON: Teste relacionado à validação do cabeçalho "Accept" com valor "application/json".
- TestBlackBoxUnsupportedAcceptHeader: Teste para verificar a resposta do sistema quando um cabeçalho "Accept" não suportado é enviado.

3.2. Execução dos Testes

A implementação foi dividida em duas partes. Os testes descritos acima foram feitos em coleções no Insomnia, uma ferramenta para teste de APIs. Além disso, foram realizados testes usando a funcionalidade de "*tests*" do Insomnia, que permite a criação e execução de scripts de teste automáticos diretamente dentro dos pedidos. Isso inclui verificações de respostas HTTP, validações de dados retornados e simulações de diferentes condições de entrada.

Essas abordagens combinadas garantiram uma cobertura abrangente das funcionalidades do sistema e uma validação robusta de suas operações.

4. Testes *White Box*

Este capítulo aborda o desenvolvimento de testes *white box* em função do Sprint 3. O teste *white box*, também conhecidos como testes estruturais, são uma metodologia de verificação interna do funcionamento dos softwares, que exige um conhecimento detalhado do código.

4.1. Análise dos Testes

Foram desenvolvidos cinco testes de white box, dos quais quatro focados no no *authMiddleware* e um no *routeValidationMiddleware*.

No *authMiddleware* :

1. Teste para Rotas não autenticadas

Descrição: Avalia a capacidade do *middleware* de permitir a progressão imediata para o próximo componente via função "next()" em rotas como "/login" que não exigem autenticação.

Importância: Garante a usabilidade e eficiência, permitindo acesso a funcionalidades básicas sem procedimentos de segurança adicionais, evitando impacto negativo na experiência do utilizador e desempenho do sistema.

2. Teste para requisições sem token

Descrição: Examina cenários onde requisições que necessitam de autenticação são feitas sem *token*, resultando em erro 401 para indicar token inválido ou expirado.

Importância: Fundamental para proteger rotas sensíveis e prevenir acessos não autorizados, mantendo a integridade e confidencialidade dos dados

3. Teste com o *token* válido, mas utilizador inexistente

Descrição: Verifica a resposta do sistema a tokens válidos que não correspondem a qualquer utilizador ativo, emitindo erro 401

Importância: Previne acessos com *tokens* desatualizados ou de utilizadores removidos, reforçando a segurança e integridade da autenticação

4. Teste de autenticação bem sucedida

Descrição: Confirma que, com um *token* válido e um utilizador existente, o acesso é corretamente autorizado e a requisição é encaminhada sem erros.

Importância: Essencial para verificar a eficácia do middleware em permitir acesso seguro e sem obstáculos às funcionalidades apropriadas, crucial para a operacionalidade e segurança da aplicação

No *routeValidateionMiddleware* :

5. Teste de Aceitação do header

Descrição: Avalia se o middleware rejeita requisições para rotas que devem retornar conteúdo em *JSON* quando o cabeçalho "Accept" indica preferência por outro formato, como *XML*. Nesse caso, um erro com status 406 é emitido.

Importância: Garante o correto manejo dos cabeçalhos *HTTP* pela *API*, assegurando a entrega de conteúdos no formato apropriado. Esta verificação é crucial para prevenir erros do lado do cliente e confusões relativas aos formatos de dados suportados pela *API*

A imagem abaixo ilustra os resultados obtidos no terminal de testes de white box realizados nos middlewares "authMiddleware"e "routeValidationMiddleware". Todos os cinco testes planejados foram executados com sucesso, sem registar quaisquer falhas.

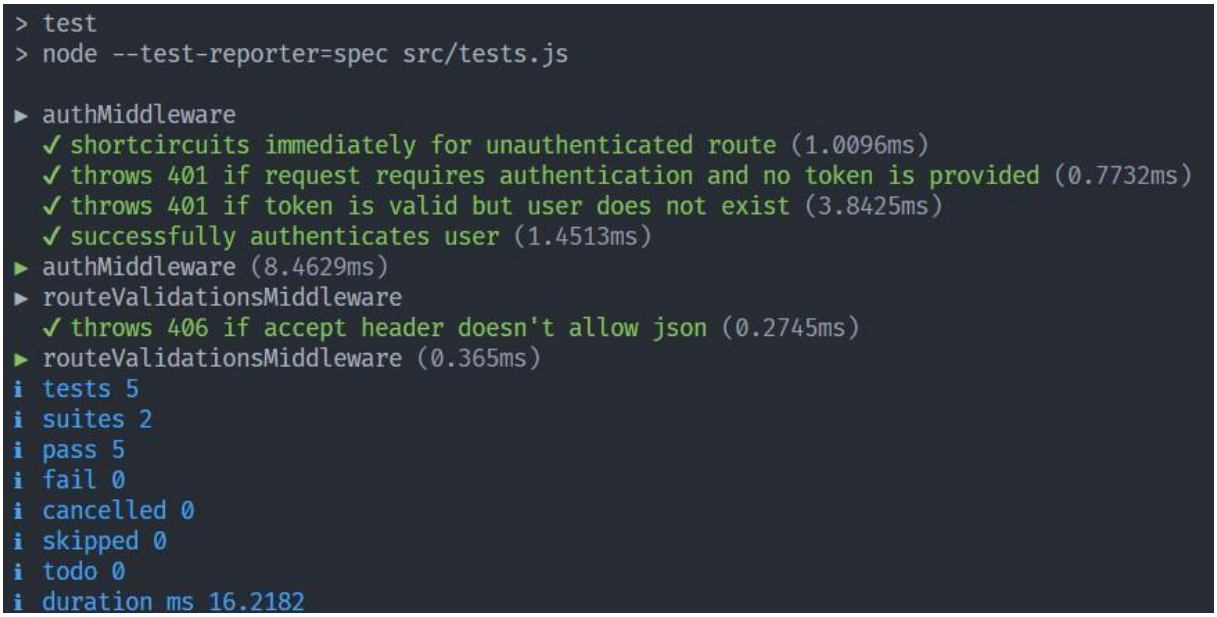

Figura 4-1 Resultados obtidos dos testes *White Box*

5. Conclusões

Ao longo deste projeto foi possível aplicar e consolidar os conceitos lecionados na Unidade Curricular de Engenharia de Software 2. A experiência adquirida não apenas permitiu uma melhor compreensão dos métodos de teste, mas também fortaleceu as habilidades de desenvolvimento e resolução de problemas. O resultado alcançado reflete não apenas a qualidade do trabalho realizado, mas também a capacidade de enfrentar desafios e entregar soluções eficazes.

Conclui-se assim que o projeto foi concluído com sucesso, atendendo aos requisitos estabelecidos e demonstrando o empenho na unidade curricular.