

INSTITUTO POLITÉCNICO DE SETÚBAL
ESCOLA SUPERIOR DE TECNOLOGIAS DE SETÚBAL



Qualidade de Software
Mestrado em Engenharia de Software

Daniel Singh - 201901071
Rafael Marçalo - 201900456

19 de novembro de 2023

Conteúdo

1	Introdução	4
2	Análise do Caso de Estudo	5
2.1	Análise de Requisitos	5
3	Ferramentas de Análise	7
4	Linting	8
4.1	Resultados	9
4.1.1	Análise simples	9
4.1.2	Análise simples (Client-side)	9
4.1.3	Análise simples (Server-side)	10
4.1.4	Análise detalhada	10
4.1.5	Análise dos resultados obtidos	12
5	Normas ISO	13
5.1	ISO 5055	13
5.1.1	Definição	13
5.1.2	Importância	13
5.1.3	Aplicação	14
6	Testes Unitários	15
6.1	Metodologias e Práticas	15
6.1.1	Test-Driven Development	15
6.1.2	Behavior-Driven Development	17
6.1.3	TDD e BDD	18
6.2	Resultados	19
6.2.1	Testes sem Sucesso	19
7	Anexos	20

Lista de Figuras

1	Estatísticas de Download de Linters	8
2	Fluxograma Desenvolvimento Orientado a Testes(TDD)	15
3	Fluxograma TDD com BDD	18

Lista de Tabelas

1	Análise de requisitos entidade cliente	5
2	Análise de requisitos entidade utilizador	5
3	Análise de requisitos entidade serviço	6
4	Análise de requisitos entidade mensagem	6
5	Análise de requisitos entidade sistema	6
6	Análise de Lint Simples	9
7	Análise de Lint Simples (Client-side)	9
8	Análise de Lint Simples (Server-side)	10
9	Análise de Lint Detalhada	11
10	Tabela ISO 5055	14

1 Introdução

Este relatório tem como objetivo a análise da qualidade de código de um projeto realizado por outros alunos. Para esta análise iremos recorrer a ao teste de funcionalidades e requisitos do projeto. Em seguida iremos testar várias ferramentas e livrarias na análise da qualidade/consistência do código. De seguida iremos em breve apresentar não só as funcionalidades destas ferramentas, mas também os resultados que obtivemos com elas.

2 Análise do Caso de Estudo

Sendo esta a primeira fase desta pequena investigação, transferi-mos o projeto caso de estudo do Moodle¹, importámos para o nosso repositório GitHub² e instalámos nas nossas máquinas para análise. Desta forma, após instalarmos e configurarmos o projeto nos nossos computadores, fizemos um breve teste das suas funcionalidades e diagnosticámos alguns problemas que possam afetar a qualidade do projeto, desta forma, para simplificar o processo, elaborámos umas tabelas com o requisitos funcionais que detetamos.

2.1 Análise de Requisitos

ID	Descrição	Implementado
RF1	O sistema deverá permitir inserir novos clientes	Sim
RF2	O sistema deverá permitir pesquisar pelos clientes	Sim
RF3	O sistema deverá permitir aplicar filtros às pesquisas dos clientes	Parcialmente ³
RF4	O sistema deverá permitir atualizar a informação dos clientes	Sim
RF5	O sistema deverá permitir eliminar clientes	Sim
RF6	O sistema deverá efetuar a validação de dados ao criar um cliente	Parcialmente ⁴
RF7	O sistema deverá limpar os campos prévios dos formulários, após sua utilização	Não

Tabela 1: Análise de requisitos entidade cliente

ID	Descrição	Implementado
RF8	O sistema deverá permitir inserir novos utilizadores	Sim
RF9	O sistema deverá permitir pesquisar pelos utilizadores	Sim
RF10	O sistema deverá permitir aplicar filtros às pesquisas dos utilizadores	Parcialmente ²
RF11	O sistema deverá permitir atualizar os seus dados pessoais	Não
RF12	O sistema deverá permitir não atualizar os dados pessoais de utilizadores com as mesmas roles ou com hierarquia superior	Não
RF13	O sistema deverá permitir atualizar a informação de outros utilizadores com hierarquia inferior	Sim
RF14	O sistema deverá permitir eliminar outros utilizadores	Sim
RF15	O sistema deverá efetuar a validação de dados ao criar um utilizador	Parcialmente ³
RF16	O sistema deverá limpar os campos prévios dos formulários, após sua utilização	Não

Tabela 2: Análise de requisitos entidade utilizador

¹<https://moodle.ips.pt/2324/>

²https://github.com/MES-ES/QS_Lab1

³Utiliza biblioteca externa

⁴Apenas no lado do cliente

ID	Descrição	Implementado
RF17	O sistema deverá permitir criar serviços	Sim
RF18	O sistema deverá permitir pesquisar pelos serviços	Sim
RF19	O sistema deverá permitir realizar pesquisas filtrados aos serviços	Sim
RF20	O sistema deverá permitir aplicar filtros às pesquisas dos serviços	Sim
RF21	O sistema deverá permitir atualizar a informação dos serviços	Sim
RF22	O sistema deverá de validar os dados ao criar um serviço	Sim
RF23	O sistema deverá permitir reabrir um serviço concluído	Sim
RF24	O sistema deverá de mostrar qual o index do serviço relativamente à sua prioridade	Parcialmente ⁵
RF25	O sistema deverá limpar os campos prévios dos formulários, após sua utilização	Não

Tabela 3: Análise de requisitos entidade serviço

ID	Descrição	Implementado
RF26	O sistema deverá permitir enviar mensagens para outros utilizadores em tempo real	Sim
RF27	O sistema deverá permitir escolher para qual utilizador enviar mensagem	Sim
RF28	O sistema deverá permitir visualizar o histórico de mensagens	Sim
RF29	O sistema deverá permitir mensagens de grupo	Não

Tabela 4: Análise de requisitos entidade mensagem

ID	Descrição	Implementado
RF30	O sistema deverá permitir fazer login	Parcialmente ⁴
RF31	O sistema deverá permitir fazer logout	Parcialmente ⁴
RF32	O sistema deverá de verificar permissões no routing	Não

Tabela 5: Análise de requisitos entidade sistema

⁵Está implementado mas não funciona corretamente

3 Ferramentas de Análise

Após um diagnóstico de requisitos do projeto, avançamos para uma análise a nível de possíveis problemas de código, para esta tarefa, efetuamos uma pesquisa de ferramentas que nos possam ajudar com este trabalho, das quais, achamos importante destacar:

- **ESLint** é uma ferramenta de análise de código estático que verifica o código JavaScript procurando diversos problemas comuns, como erros de sintaxe, problemas de formatação, violações de estilo de código e possíveis bugs.
- **Prettier Code Formatter** é um formatador de código. Impõe um estilo consistente analisando o código e reescrevendo-o conforme um conjunto de regras que levam em consideração diversos parâmetros como (comprimento máximo de linha, agrupando o código quando necessário, etc...). Ao utilizarmos esta ferramenta conseguimos garantir a uniformização do código desenvolvido ao longo do projeto.
- **Jest** é uma framework para testes de código compatível com bastantes projetos (Babel, TypeScript, Node, React, Angular, Vue e mais...). Suporta mocking, gera reports de code coverage.
- **Mocha** é outra framework de testes para Node capaz de correr no browser. Esta framework contém “interfaces” de testes que podem ser utilizados para vários tipos de desenvolvimento(TDD, BDD, exports, qunit, require, etc...). O Mocha também permite o fácil teste de funções assíncronas e de promises.
- **Chai**, livreria de testes especificada para testes do tipo BDD/TDD, esta livreria geralmente é integrada noutras frameworks específicas de testes(como o Mocha, por exemplo).

4 Linting

Para a análise de linting do código usamos a ferramenta ESLint, devido à sua alta configurabilidade e por ser uma das ferramentas de linting mais utilizadas, como podemos observar no esquema abaixo:

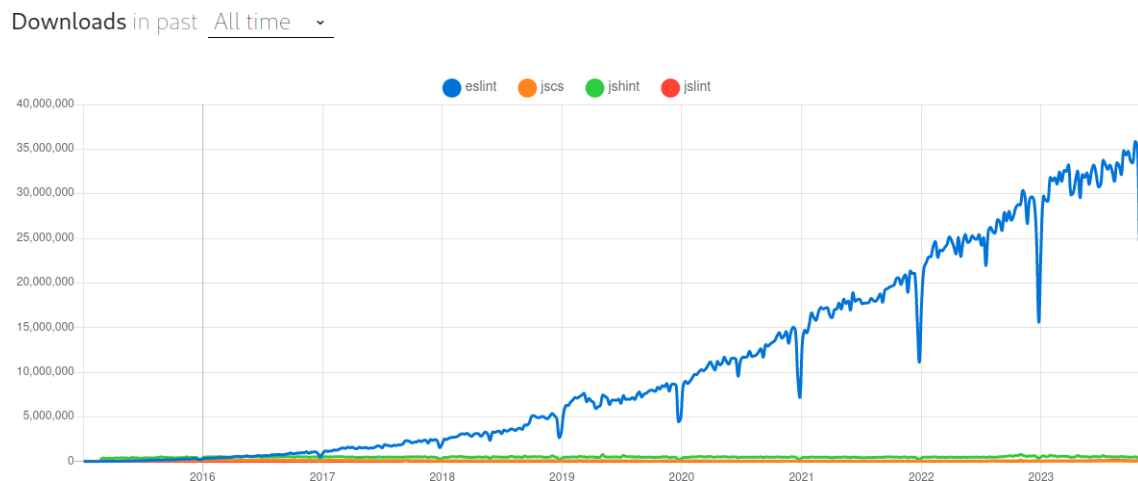


Figura 1: Estatísticas do download dos Linters ao longo do tempo⁶

A nossa experiência a utilizar o **ESLint** tem sido bastante benéfica, ajudou-nos a identificar vários problemas no código analisado. Devido a configurabilidade da ferramenta, esta permitiu-nos identificar e relatar diversos problemas desde erros de sintaxe a problemas mais complexos relacionados com a qualidade e a manutenibilidade do código **JavaScript**. Adicionalmente podemos também referir que a habilidade de definir regras personalizadas ajudaram-nos a adaptar a ferramenta aos requisitos do projeto.

Com a ajuda deste [custom formatter](#) foi nos possível organizar os resultados obtidos do output do **ESLint**.

⁶Imagem obtida através de <https://npm trends.com/eslint-vs-jscs-vs-jshint-vs-jslint>

4.1 Resultados

Através desta ferramenta, foi nos possível efetuar várias auditorias ao código utilizando várias configurações, permitindo-nos assim destacar as estatísticas

4.1.1 Análise simples

Para esta análise, configuramos o **ESLint** para apenas reportar erros de sintaxe e alguns problemas de código em geral (variáveis inutilizadas, valores *undefined*, *espaços* misturados com *tabs*, etc...).

Problema Anotado	Total
Variáveis Inutilizadas	40
Valores <i>undefined</i>	189
<i>Espaços</i> Misturados com <i>Tabs</i>	2

Tabela 6: Análise de Lint Simples

4.1.2 Análise simples (Client-side)

Nesta análise decidimos fazer distinção entre problemas de código de cliente obtendo assim os seguintes resultados:

Problema Anotado	Total
Variáveis Inutilizadas	23
Valores <i>undefined</i>	189
<i>Espaços</i> Misturados com <i>Tabs</i>	2

Tabela 7: Análise de Lint Simples (Client-side)

4.1.3 Análise simples (Server-side)

Nesta análise decidimos fazer distinção entre problemas de código de servidor obtendo assim os seguintes resultados:

Problema Anotado	Total
Variáveis Inutilizadas	17

Tabela 8: Análise de Lint Simples (Server-side)

4.1.4 Análise detalhada

Para a análise detalhada, configuramos o **ESLint** para reportar os problemas da análise simples, assim como os problemas de estilo de código. Para esta análise, escolhemos o estilo de código *Standard* onde pudemos observar os seguintes resultados:

Problema Anotado	Total
Utilização de <i>aspas</i>	772
Utilização de <i>ponto e vírgula</i>	1302
<i>Espaço</i> a terminar linha	171
Linhas em branco	26
Indentação incorreta	2062
<i>Espaço</i> inserido várias vezes de seguida	11
<i>Espaço</i> inserido antes de comentário	50
<i>Espaço</i> em falta antes do <i>parênteses</i> da <i>função</i>	121
<i>Espaço</i> em falta antes de blocos de código	67
Operador ternário desnecessário	1
Trocar variável para <i>const</i>	1
Usar <i>var</i>	11
<i>Espaço</i> antes de uma palavra-chave	2
Estilo das <i>chavetas</i>	93
<i>Espaço</i> entre as <i>=></i>	85
<i>Objeto</i> abreviado	6
Caractere terminal de ficheiro	5
<i>Espaço</i> dentro da definição de objeto	5
Variáveis Inutilizadas	11
<i>Espaços padding</i> blocos de código	7
<i>Hardcoded callback</i>	24
Valores <i>undefined</i>	189
Propriedades sem <i>aspas</i>	6
<i>Tabs</i> vazios	2
Nova linha após <i>parênteses</i> de objeto	9
Nova linha após propriedade de objeto	9
Falta de <i>espaços</i> entre operadores lógicos e aritméticos	22
<i>Vírgula</i> em propriedades finais de objetos	2
<i>Espaços</i> entre parênteses	2
<i>Espaços</i> múltiplos	2

Tabela 9: Análise de Lint Detalhada

4.1.5 Análise dos resultados obtidos

Observando a análise de código detalhada podemos constatar que muitos dos problemas apontados pela ferramenta, poderiam ser evitados através da utilização de uma ferramenta de formatação de código (*ex: **Prettier Code Formatter***).

O **Prettier** é uma ferramenta de formatação de código que analisar o código e de seguida, o imprime de novo num estilo consistente. Ao contrário dos linters tradicionais que se focam em identificar e corrigir erros de codificação específicos, o principal objetivo do **Prettier** é impor um estilo de código consistente e opinativo em todo o código-fonte.

Um dos principais benefícios do **Prettier** é a sua simplicidade e configuração *barebones*. Ele pretende minimizar a necessidade dos programadores tomarem decisões sobre o estilo de código, fornecendo um conjunto padrão de regras sendo estas facilmente configuradas.

Os programadores podem integrar o **Prettier** no seu *workflow* usando-o diretamente a partir da linha de comandos ou integrando-o diretamente em editores de código e *IDEs*.

5 Normas ISO

As normas ISO em desenvolvimento de software são padrões internacionais que permitem aos programadores de todo o mundo estruturar melhor o seu código. Existem várias certificações ISO aplicáveis aos programadores de software, e cada uma delas estabelece diretrizes e boas práticas para a criação de software. A conformidade com as normas ISO pode melhorar a segurança dos dados, a confiabilidade para os clientes e a clareza dos procedimentos e segurança do ciclo de vida do desenvolvimento do software.

As normas ISO mais comuns na engenharia de software são:

- *ISO/IEC 9126*;
- *ISO/IEC 25010*;
- *ISO/IEC 15504*;
- *ISO/IEC 12207*;
- *ISO 5055*;
- *ISO 9001*;
- *ISO/IEC 27001*;
- *ISO/IEC 20000-1*.

5.1 ISO 5055

5.1.1 Definição

A norma que escolhemos para este projeto foi a norma *ISO 5055* que mede a qualidade e integridade de um sistema de software analisando a sua construção de forma a detetar falhas estruturais. Esta norma baseia-se em 4 fatores: segurança, confiabilidade, eficiência e manutenibilidade.

5.1.2 Importância

Esta norma é importante porque ajuda a determinar a confiabilidade, a dependabilidade e a resiliência de um software a nível de sistema e componente.

5.1.3 Aplicação

Caraterística	Causa	Consequência
Segurança	<ul style="list-style-type: none">• Passwords não <i>Hashed</i>• Inexistência de verificações do lado do servidor	<ul style="list-style-type: none">• Informação sensível exposta a <i>hackers</i>• Acesso indevido a componentes e partes administrativas do sistema• Cross-site scripting
Confiabilidade	<ul style="list-style-type: none">• Baixo nível de gestão de erros⁷	<ul style="list-style-type: none">• Exceções durante as transações com a Base de Dados
Eficiência	<ul style="list-style-type: none">• Consultas inteiras à Base de Dados• Carregamento de recursos excessivos a cada atualização de página	<ul style="list-style-type: none">• Alto tráfego de dados• Processamento excessivo
Manutenibilidade	<ul style="list-style-type: none">• Repetição de código• Não cumprimento das convenções de codificação	<ul style="list-style-type: none">• Alto acoplamento• Problemas de interpretação de código

Tabela 10: Tabela ISO 5055

⁷Apesar de existirem poucas verificações, os módulos e funções que foram escolhidos ao longo do projeto têm alguns mecanismos fail-safe.

6 Testes Unitários

Os testes unitários são uma prática fundamental no desenvolvimento de software. Estes, envolvem o teste de unidades/componentes individuais do código-fonte por forma a garantir a boa funcionalidade das suas operações. Estas unidades/componentes podem variar entre funções, métodos ou objetos. O objetivo principal destes testes é certificar cada uma destas unidades/componentes por forma a verificar se os retornos estão corretos, consoante a variedade de inputs possíveis. Ao se isolar e testar pequenas porções do código, os programadores mais facilmente conseguem identificar e corrigir bugs no ciclo de desenvolvimento.

6.1 Metodologias e Práticas

O Desenvolvimento Orientado a Testes (TDD) e o Desenvolvimento Orientado ao Comportamento (BDD) são duas metodologias essenciais no campo do desenvolvimento e teste de software. Tanto o TDD quanto o BDD desempenham um papel crucial na garantia da qualidade e confiabilidade de aplicações de software.

6.1.1 Test-Driven Development

O TDD é uma metodologia de desenvolvimento de software na qual testes unitários são usados para orientar o desenvolvimento da aplicação. Com o TDD, os programadores escrevem os testes antes de escrever o código. Com isto os programadores conseguem ter a certeza de que o código que escreveram faz o trabalho pretendido e os testers podem executar os seus testes tendo a certeza de que o código funcionará corretamente.

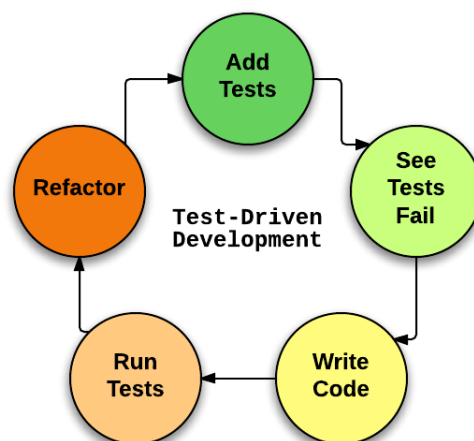


Figura 2: Fluxograma Desenvolvimento Orientado a Testes(TDD)⁸

⁸Imagem obtida através de https://www.clipartmax.com/middle/m2i8m2m2K9K9N4i8_bdd-flow-diagram-images-gallery-diagram/

Processo Test Driven Development

- Escrever um teste que define uma nova função ou melhoria;
- Escrever a quantidade mínima de código para passar no teste;
- Refatorar o código para remover duplicações e melhorar seu design.

Benefícios Test Driven Development

- Tempo de trabalho reduzido: O TDD não permite que novo código seja escrito, a menos que o código existente seja testado com sucesso e sem falhas. Até que as falhas sejam completamente resolvidas e removidas, o processo de escrita do código é interrompido;
- Rápido feedback: Como os testes focam-se em secções específicas do código, os programadores podem receber feedback mais rapidamente permitindo-lhes implementar alterações de maneira mais eficiente;
- Maior produtividade de desenvolvimento: Com o TDD, o foco está na produção de código funcional, e não no design de casos de teste;
- Código mais flexível e de fácil manutenção: Como cada parte do código é testada antes de passar para a próxima parte do processo de desenvolvimento de software, o código mantém a funcionalidade e é adaptável no futuro.

6.1.2 Behavior-Driven Development

O Desenvolvimento Orientado ao Comportamento (BDD) é uma extensão do TDD que reforça testar o comportamento da aplicação do ponto de vista do utilizador final. O BDD envolve a criação de especificações executáveis que definem o comportamento da aplicação. Este, baseia-se em testar comportamentos em vez de detalhes de implementação, facilitando a colaboração entre programadores, testers e utilizadores. Os cenários de BDD geralmente são descritos num formato mais legível e centrado ao utilizador, permitindo que utilizadores não programadores possam compreender os testes.

Processo Behavior-Driven Development As etapas para o Desenvolvimento Orientado ao Comportamento são bastante simples:

- O comportamento é descrito normalmente utilizando uma user story, que permite que a equipa discuta exemplos concretos da nova funcionalidade e as expectativas do seu comportamento;
- A ação é então escrita, transformando os exemplos em documentação de forma que possa ser automatizada;
- O teste é executado para auxiliar e orientar os programadores no desenvolvimento do código;
- O código é então criado, para realizar esta funcionalidade tornando assim o código funcional.

Benefícios Behavior-Driven Development Existem vários benefícios em usar BDD para desenvolvimento de software, incluindo:

- Incorporação da experiência do utilizador(UX): o BDD foca-se na experiência do utilizador, como tal, permite que a equipa desenvolva uma perspetiva mais ampla e observe lacunas no seu design;
- Custo-benefício: Como o BDD estabelece prioridades para utilizadores, programadores e investidores, este, permite que os recursos sejam utilizados de forma otimizada no desenvolvimento de software;
- Testes simples entre browsers: o BDD concentra-se no comportamento de utilizador, o que significa que oferece uma estrutura ideal para testes entre browsers.

6.1.3 TDD e BDD

Em resumo, o TDD é uma prática de desenvolvimento que se foca em testar o código de forma isolada, enquanto o BDD é uma metodologia de equipa que se foca em testar o comportamento da aplicação do ponto de vista do utilizador. Embora partilhem semelhanças, eles têm algumas diferenças-chave na finalidade, no tipo de colaboração, na linguagem e nos casos de teste.

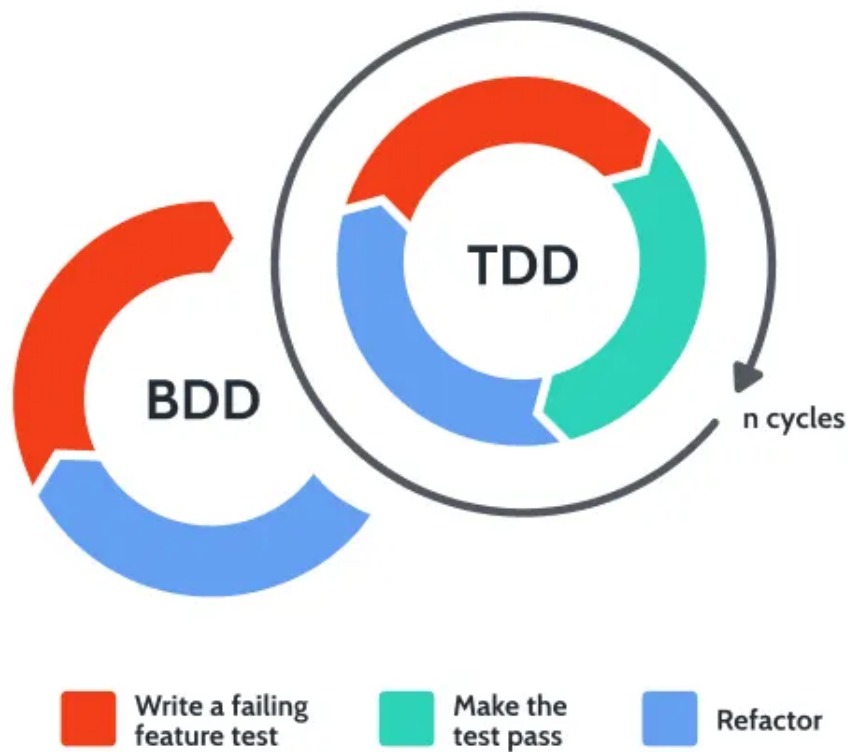


Figura 3: Fluxograma TDD com BDD⁹

⁹Imagem obtida através de <https://www.outsystems.com/blog/posts/using-bdd-tdd/>

6.2 Resultados

Após efetuarmos os testes unitários ao código-fonte, correremos o comando `npx jest --coverage` e verificamos que conseguimos obter uma code coverage de cerca 95%.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	91.34	78.2	95.12	91.34	
scripts	86.88	76.78	90.24	86.88	
authentication-handlers.js	100	100	100	100	
clients-handlers.js	100	100	100	100	
globalHandlers.js	100	100	100	100	
jobs-handlers.js	85.84	67.85	100	85.84	40-48,94-96,101,121-126,154-156,165
messaging-handlers.js	74.28	66.66	66.66	74.28	57-78
users-handlers.js	82.97	80	80	82.97	77-86
tests	95.85	79	100	95.85	
classes-code.js	100	100	100	100	
localStorage-code.js	89.47	76.47	100	89.47	29,43,58,72,87,101,116,130
requests-code.js	98.4	80.3	100	98.4	94-95

Podemos observar que relativamente ao ficheiro `localStorage-code.js` as linhas não cobertas estão relacionadas com verificações à propriedade `localStorage`, que no ambiente de testes esteve *"mocked"*, por forma a não mexer com dados originais. Ainda no ficheiro `request-code.js`, as linhas não cobertas estão relacionadas com o insucesso do pedido de `logout` pois num ambiente de produção o servidor retornava sempre o resultado *status: 200*.

Desta forma, deixamos nos anexos o report do resultado individual de cada teste unitário.

6.2.1 Testes sem Sucesso

De seguida podemos analisar os testes que falharam, de uma forma breve iremos explicar o porquê.

- Delete User tests(handle an unsuccessful user delete because of missing data) - Este teste falhou pois não é efetuado a validação do parâmetro ID;
- getUsersAjax function tests (With unsuccessful request and new users saved in localStorage)
- Este teste falhou devido ao mau tratamento de dados. Quando o servidor consegue ler os dados da base de dados, este envia um objeto JSON para o cliente e este é devolvido. Quando o servidor não consegue ler os dados da base de dados, a request vai buscar os dados guardados no localStorage, mas o localStorage guarda os dados em String e a função de request devolve os dados sem fazer o parse para objeto;
- getClientsAjax function tests (With unsuccessful request and new users saved in localStorage)
- Este teste falhou devido ao mau tratamento de dados. Quando o servidor consegue ler os dados da base de dados, este envia um objeto JSON para o cliente e este é devolvido. Quando o servidor não consegue ler os dados da base de dados, a request vai buscar os dados guardados no localStorage, mas o localStorage guarda os dados em String e a função de request devolve os dados sem fazer o parse para objeto.

7 Anexos

Test Report

Started: 2023-11-18 22:47:24

Suites (9)

7 passed

2 failed

0 pending

Tests (98)

95 passed

3 failed

0 pending

v C:\Users\danie\Desktop\QS\QS_Lab1\tests\authentication-handlers.test.js			4.382s
login function	handle the successful login	passed	0.098s
login function	handle an unsuccessful login with incorrect credentials	passed	0.005s
login function	should handle an error during the login process	passed	0.003s
v C:\Users\danie\Desktop\QS\QS_Lab1\tests\messaging-handlers.test.js			4.338s
loadWebSocketSettings tests	handle a successful query and return settings	passed	0.092s
loadWebSocketSettings tests	should handle an unsuccessful query and return an empty array	passed	0.004s
messagingInsertNew tests	handle a successful insert	passed	0.004s
messagingInsertNew tests	handle a message request with missing data	passed	0.003s
messagingInsertNew tests	should handle an unsuccessful query and return -1	passed	0.002s
loadWebSocketMessages tests	handle a successful query and return the messages	passed	0.005s
loadWebSocketMessages tests	should handle an unsuccessful query and return an empty array	passed	0.003s
v C:\Users\danie\Desktop\QS\QS_Lab1\tests\classes.test.js			4.589s

Validate Client Class	<i>constructor without values</i>	passed	0.062s
Validate Client Class	<i>constructor with values</i>	passed	0.006s
Validate User Class	<i>constructor without values</i>	passed	0.002s
Validate User Class	<i>constructor with values</i>	passed	0.009s
Validate JobInfo Class	<i>constructor without values</i>	passed	0.004s
Validate JobInfo Class	<i>constructor with values</i>	passed	0.01s
Validate JobTypeRequest Class	<i>constructor without values</i>	passed	0.001s
Validate JobTypeRequest Class	<i>constructor with values</i>	passed	0.004s
Validate MessageSocket class	<i>constructor without values</i>	passed	0.003s
Validate MessageSocket class	<i>constructor with values</i>	passed	0.004s

<div> <div></div> <div>C:\Users\danie\Desktop\QS\QS_Lab1\tests\clients-handlers.test.js</div> </div>			4.48s
getClients function	<i>handle a successful query and return clients</i>	passed	0.094s
getClients function	<i>handle an error during the query and return an empty array</i>	passed	0.003s
editClient function	<i>handle a successful update and return success: true</i>	passed	0.002s
editClient function	<i>should handle an unsuccessful update and return success: false</i>	passed	0.002s
deleteClient function	<i>handle a successful delete and return 200 status</i>	passed	0.002s
deleteClient function	<i>handle an unsuccessful delete and return 500 status</i>	passed	0.004s
createClient function	<i>handle a successful insert and return 200 status</i>	passed	0.002s

createClient function	<i>handle an unsuccessful insert and return 500 status</i>	passed	0.002s
v C:\Users\danie\Desktop\QS\QS_Lab1\tests\localStorage.test.js			4.278s
SaveUserDetails function test	<i>SaveUserDetails with user to localStorage</i>	passed	0.044s
SaveUserDetails function test	<i>SaveUserDetails with no user to localStorage</i>	passed	0.005s
SaveUserDetails function test	<i>SaveUserDetails with null to localStorage</i>	passed	0.003s
getUserDetails function test	<i>getUserDetails with user in localStorage</i>	passed	0.003s
getUserDetails function test	<i>getUserDetails with no user in localStorage</i>	passed	0.003s
saveClientsLocalStorage function test	<i>saveClientsLocalStorage with clients to localStorage</i>	passed	0.002s
saveClientsLocalStorage function test	<i>saveClientsLocalStorage with no clients to localStorage</i>	passed	0.001s
saveClientsLocalStorage function test	<i>saveClientsLocalStorage with null value to localStorage</i>	passed	0.002s
getClientsLocalStorage function test	<i>getClientsLocalStorage with user in localStorage</i>	passed	0.009s
getClientsLocalStorage function test	<i>getClientsLocalStorage with no user in localStorage</i>	passed	0.003s
saveUsersLocalStorage function test	<i>saveUsersLocalStorage with users to localStorage</i>	passed	0.002s
saveUsersLocalStorage function test	<i>saveUsersLocalStorage with no users to localStorage</i>	passed	0.002s
saveUsersLocalStorage function test	<i>saveUsersLocalStorage with null value to localStorage</i>	passed	0.001s
getUsersLocalStorage function test	<i>getUsersLocalStorage with user in localStorage</i>	passed	0.004s

Get User Tests	handle a get all user requests with multiple users	passed	0.001s
Get User Tests	handle a get all users with no users	passed	0.004s
Get User Tests	handle a get all users when a database error occurred	passed	0.006s

getUserInfo initState function	<i>should handle an unsuccessful query and return an empty array</i>	passed	0.002s
editJobInfo function	<i>handle a successful update and return 200 status</i>	passed	0.003s
editJobInfo function	<i>handle an unsuccessful update and return 500 status</i>	passed	0.002s
createJob function	<i>create a job and return 200 status on success</i>	passed	0.003s
createJob function	<i>handle an error during job creation and return 500 status</i>	passed	0.003s
reopenJob function	<i>handle a successful update and return 200 status</i>	passed	0.002s
reopenJob function	<i>handle an unsuccessful update and return 500 status</i>	passed	0.002s
editOrderPriority function	<i>handle a successful update and return 200 status</i>	passed	0.001s
editOrderPriority function	<i>handle an unsuccessful update and return 500 status</i>	passed	0.001s