
# Trabalho Prático 1

## Qualidade de Software 2023/2024

**ALUNOS**

Andrei Balcan, nº 202002359

Bernardo Fortaleza, nº201700228

**PROFESSOR**

Rui Rodrigues

# Índice

# Índice de Figuras

# Índice de Tabelas

# Introdução

O presente relatório visa documentar a análise de código de um projeto pré-existente e realizar a sua melhoria mediante técnicas lecionadas na unidade curricular de Qualidade de Software.

O projeto analisado encontra-se no repositório privado do *GitHub* [https://github.com/201700228/Trabalho-Laboratorio-1], esta plataforma para além de auxiliar no desenvolvimento também ajuda a desenvolver práticas de *pair programming*, que são valorizadas no mercado laboral e académico.

O projeto em causa foi desenvolvido utilizando as tecnologias e linguagens de programação *HTML*, *CSS* e *JavaScript* e trata-se de uma solução *Web* de um sistema de informação que permite a gestão/organização de reparações de componentes eletrónicos.

Este relatório está dividido em três capítulos essenciais: extração de requisitos, ferramentas para melhoria de código e normas ISO.

No primeiro capítulo, foi feita uma análise detalhada do código existente para extrair os requisitos funcionais do sistema, usando uma técnica chamada '*reverse engineering*'.

O segundo capítulo centrou-se na seleção cuidadosa e na aplicação sistemática de algumas ferramentas para melhorar e testar a qualidade do código-fonte, ferramentas essas que foram fundamentais para uma avaliação detalhada dos resultados obtidos.

Por fim, no último capítulo realizou-se uma pesquisa sobre as normas *ISO* existentes para gerir e assegurar a qualidade do software. Após uma análise comparativa, selecionou-se a norma mais apropriada e completa para aplicação no projeto, escolha esta que permitiu uma análise crítica das funcionalidades existentes, identificando tanto os pontos fortes quanto as lacunas do sistema em relação aos requisitos estabelecidos.

## Extração de Requisitos

A qualidade particular de um *software* é fortemente determinada pela precisão e clareza dos requisitos estabelecidos durante o processo de conceção. Esta fase crítica não apenas forma a base para o desenvolvimento do sistema, mas também influência diretamente a experiência do utilizador, a segurança do sistema e a eficiência operacional. No âmbito do projeto em análise, identificou-se a imperativa necessidade de refinar os requisitos, visando não apenas a correção de imprecisões, mas também a maximização da usabilidade e eficácia do sistema.

A etapa de identificação de requisitos desempenha um papel crucial ao traduzir as necessidades dos utilizadores em funcionalidades tangíveis do sistema classificando-as conforme a sua prioridade e relevância.

Tabela *1* - Requisitos da página de *login*

| Requisitos de l*ogin* | Descrição | Prioridade |
|---|---|---|
| **R1: Registo de utilizadores** | Os utilizadores devem ter a capacidade de se registar na aplicação. | MUST HAVE |
| **R2: Autenticação de utilizadores** | Os utilizadores registados devem poder fazer login utilizando um nome de utilizador e senha. | MUST HAVE |
| **R3: Autenticação segura** | O sistema deve garantir uma autenticação segura, protegendo as informações do utilizador. | MUST HAVE |
| **R4: Recuperação de senha** | Os utilizadores devem ter a possibilidade de alterar as suas senhas caso se esqueçam. | SHOULD HAVE |
| **R5: *Logout*** | Os utilizadores devem conseguir fazer logout da aplicação. | MUST HAVE |
| **R6: Autenticação multifatores (*Two Factor Authentication*)** | O sistema deve suportar autenticação multifatores para reforçar a segurança. | SHOULD HAVE |

Tabela *2* - Requisitos da página de clientes

| Requisitos dos clientes | Descrição | Prioridade |
|---|---|---|
| **R7: Registo de clientes** | Os utilizadores autorizados devem poder criar registos de clientes, incluindo informações como nome, endereço, e número de telefone. | MUST HAVE |
| **R8: Gestão de informações de clientes** | Os utilizadores autorizados devem poder visualizar, editar e excluir informações de clientes existentes. | MUST HAVE |
| **R9: Dados nos perfis de clientes** | Os perfis de clientes devem incluir informações detalhadas, como nome, endereço, número de telefone, etc. | MUST HAVE |
| **R10: Criação de utilizadores** | Os perfis autorizados devem poder criar utilizadores. | SHOULD HAVE |
| **R11: Atribuição de níveis de permissão** | Os utilizadores autorizados devem poder atribuir níveis de permissão a cada utilizador. | SHOULD HAVE |

| | | |
|---|---|---|
| **R12: Gestão de utilizadores** | Os utilizadores autorizados devem poder visualizar, editar e excluir informações de utilizadores existentes. | SHOULD HAVE |
| **R13: Pesquisa de clientes** | Os utilizadores devem conseguir pesquisar clientes existentes através de uma caixa de pesquisa. | SHOULD HAVE |
| **R14: Seleção de número de clientes** | Os utilizadores devem conseguir selecionar o número de clientes a serem exibidos no ecrã. | COULD HAVE |

Tabela *3* - Requisitos da página de serviços

| Requisitos dos serviços | Descrição | Prioridade |
|---|---|---|
| **R15: Registo de serviços** | Os utilizadores autorizados devem poder criar registos de serviços atribuídos a clientes. | MUST HAVE |
| **R16: Associação de serviços a clientes** | Os utilizadores autorizados devem poder associar serviços a clientes específicos. | MUST HAVE |
| **R17: Gestão de informações de serviços** | Os utilizadores autorizados devem poder visualizar, editar e excluir informações de serviços. | MUST HAVE |
| **R18: Níveis de prioridade para serviços** | Os serviços devem ter níveis de prioridade atribuídos, como alta, média e baixa. | MUST HAVE |
| **R19: Atribuição e modificação de prioridades** | Os utilizadores autorizados devem poder atribuir e modificar os níveis de prioridade dos serviços. | SHOULD HAVE |
| **R20: Pesquisa de serviços** | Os utilizadores devem conseguir pesquisar serviços existentes através de uma caixa de pesquisa. | SHOULD HAVE |
| **R21: Filtro por estado dos serviços** | Os utilizadores devem conseguir filtrar os serviços pelo seu estado. | SHOULD HAVE |
| **R22: Filtro por pessoa responsável** | Os utilizadores devem conseguir filtrar os serviços pela pessoa a que estão atribuídos. | COULD HAVE |
| **R23: Seleção de número de serviços** | Os utilizadores devem conseguir selecionar o número de serviços a serem exibidos no ecrã. | COULD HAVE |

Tabela *4* - Requisitos da funcionalidade *chat*

| Requisitos do chat | Descrição | Prioridade |
|---|---|---|
| **R24: Comunicação através de chat** | Os utilizadores autorizados devem poder comunicar através de um sistema de chat de mensagens. | MUST HAVE |
| **R25: Início de conversas** | Os utilizadores devem poder iniciar conversas com outros utilizadores ou grupos. | MUST HAVE |
| **R26: Suporte a mensagens de texto** | O chat deve suportar o envio de mensagens de texto. | MUST HAVE |
| **R27: Notificações de novas mensagens** | O chat deve notificar os utilizadores sobre novas mensagens recebidas. | SHOULD HAVE |

Tabela *5* - Requisitos dos operadores

| Requisitos do operador | Descrição | Prioridade |
|---|---|---|
| **R28: Restrição de acesso à página de utilizadores** | Os operadores não deverão poder aceder à página de Utilizadores, mesmo através do URL. | MUST HAVE |

# Ferramentas para Melhoria de Código

Neste capítulo, foram identificadas diversas ferramentas de *code linting* e testes unitários, destinadas a detetar potenciais erros de sintaxe, problemas de formatação de código e outras questões relevantes.

Após uma análise detalhada, foi escolhida uma ferramenta para cada categoria e, posteriormente, foram aplicadas no projeto para avaliar a sua eficácia na identificação de erros.

## *Code Linters*

Os *code linters* são ferramentas de análise estática utilizadas para detetar erros de programação, problemas estilísticos, *bugs* e práticas de desenvolvimento que possam prejudicar o desempenho do código da aplicação, além de serem essenciais para garantir a qualidade e a consistência do código-fonte.

### Principais Funcionalidades

De seguida serão apresentadas algumas funcionalidades e vantagens dos *code linters*:

- **Deteção de Espaçamento Incorreto:** Os *linters* analisam o código na procura de espaçamentos inconsistentes, promovendo não apenas a legibilidade, mas também prevenindo potenciais erros de execução decorrentes de formatação inadequada.
- **Identificação de Falta de Ponto e Vírgula (;):** A ausência de ponto e vírgula, um erro comum, é destacada pelos *linters*, evitando assim ambiguidades no código e potenciais problemas de execução.
- **Harmonização entre Aspas Simples e Duplas:** Ao verificar a uniformidade no uso de aspas, os *linters* ajudam a garantir que as aspas sejam usadas de forma consistente, o que melhora a aparência do código e torna mais fácil ler e manter.
- **Alertas sobre Importações Descontinuadas:** Os *linters* são úteis para encontrar e eliminar importações antigas, evitando possíveis conflitos e garantindo que o código esteja sempre atualizado com as versões mais recentes das bibliotecas utilizadas.
- **Deteção de Variáveis Declaradas e Não Utilizadas:** Os *linters* apontam variáveis que foram declaradas, mas não têm um propósito no código, incentivando uma abordagem mais eficiente.

## Exemplos de *Code Linters*

De seguida serão apresentadas algumas das ferramentas populares para identificar *code linters*.

Tabela *6 - ESLint*

| | | |
|---|---|---|
| **Linguagens Suportadas** | *JavaScript* e *JSX* (com vários *plugins* e configurações). |  |
| **Recursos Principais** | O ***ESLint*** é altamente configurável e extensível, permitindo a aplicação de padrões de codificação, deteção de erros e problemas de estilo. | |
| **Comunidade** | O ***ESLint*** tem uma comunidade grande e ativa, com recursos online, configurações e *plugins* abundantes para atender a necessidades específicas de projetos. | |

Tabela *7 - Sonarlint*

| | | |
|---|---|---|
| **Linguagens Suportadas** | Uma variedade de linguagens, incluindo *Java*, *JavaScript*, *Python*, *C#*. |  |
| **Recursos Principais** | O ***SonarLint*** faz parte do ecossistema maior do *SonarQube*, que se concentra na qualidade e segurança do código.<br>O ***SonarLint*** identifica e ajuda a corrigir problemas relacionados à qualidade do código, vulnerabilidades de segurança e problemas de código. Integra-se com vários *IDEs* e faz parte do pipeline de *CI/CD*. | |
| **Conjunto de Regras** | Aplica um conjunto específico de regras ou padrões de codificação com base na configuração do projeto | |

Tabela *8 - JSLint*

| | | |
|---|---|---|
| **Linguagens Suportadas** | *JavaScript* |  |
| **Recursos Principais** | O ***JSLint*** é um *linter JavaScript* simples e rígido que aplica um conjunto rigoroso de padrões de codificação. É conhecido por sua rigidez e simplicidade, o que pode ser uma vantagem ou limitação, dependendo dos requisitos do projeto.<br><br>É frequentemente usado para fins educacionais e como ponto de partida para padrões de codificação. | |
| **Comunidade** | Possui uma comunidade menor em comparação com o *ESLint*, o que pode limitar a sua flexibilidade e extensibilidade. | |

## Escolha da Ferramenta

Após analisarmos todas as ferramentas mencionadas acima, decidimos escolher a ferramenta ***ESLint*** para aplicar ao nosso projeto, tendo em conta as seguintes **vantagens**:

- É uma ferramenta gratuita.
- É fácil de configurar e utilizar.
- Comparado com as outras ferramentas, torna claro quais os problemas presentes no código.
- É a biblioteca mais popular para *code linting* dentro da comunidade *JavaScript*.

## Aplicação da Ferramenta no Projeto

A análise concentrou-se de maneira específica em duas pastas do projeto: "***scripts***" e "***www***", sendo que cada uma delas desempenha um papel crucial no desenvolvimento.

A pasta "***scripts***" está direcionada para a lógica e manipulação de dados do lado do servidor (***backend***), enquanto a pasta ***"www"*** engloba os elementos fundamentais para a interface do utilizador e a apresentação visual do lado do cliente (***frontend***).

A utilização do *ESLint* identificou uma quantidade significativa de problemas, sublinhando a importância de uma intervenção imediata.

## Resultados Obtidos

### Pasta *Scripts*

Durante a análise detalhada desta pasta foram identificados vários problemas relacionados com erros de sintaxe e de *code styling*.

**Erros de Sintaxe:**

Em 12 casos específicos de erros de sintaxe, foi identificado um problema recorrente: a declaração de múltiplas variáveis que acabaram por não serem utilizadas (`no-unused-vars`). Essa situação cria redundância no código, prejudicando a sua eficiência e legibilidade.

***Code Styling***:

Encontrámos 673 ocorrências, sendo que a mais comum estava associada a erros de indentação, destacando-se a importância de manter um alinhamento consistente no código.

Além disso, foram identificados outros erros:

- Problemas relacionados com o ponto e vírgula (`semi`).
- Espaçamento entre chavetas (`object-curly-spacing`).
- Preferência por constantes (`prefer-const`).
- Estilo de chavetas (`brace-style`).
- Espaços excedentes no final de linhas (`no-trailing-spaces`).
- Comentários com espaços (`spaced-comment`).
- A prática de evitar literais de callback (`no-callback-literal`).

Tabela *9* - Pasta *Scripts (*Erros de Sintaxe)

| Especificação | Nº de Ocorrências | Ficheiros | Exemplos |
|---|---|---|---|
| `no-unused-vars` | 12 | clients-handlers.js, globalHandlers.js jobs-handlers.js, users-handlers.js | `'rows', 'mysql', 'options', 'results'` |

Tabela *10* - Pasta *Scripts* (Erros de *Code Styling*)

| Nº de Ocorrências | Ficheiros | Exemplos |
|---|---|---|
| 673 | clients-handlers.js,<br>globalHandlers.js<br>jobs-handlers.js,<br>users-handlers.js | `semi`,<br>`object-curly-spacing`,<br>`indent`,<br>`prefer-const`,<br>`brace-style`,<br>`no-trailing-spaces`,<br>`spaced-comment`,<br>`n/no-callback-literal` |

### Pasta *"www"*

Durante a análise dessa pasta, identificámos diversos desafios relacionados a erros de sintaxe e questões de *code styling*.

Ao contrário da pasta "*scripts*", encontrámos uma quantidade significativa de erros, o que evidencia a complexidade da componente *frontend* do nosso projeto.

**Erros de Sintaxe**:

Além do problema mencionado anteriormente (`no-unused-vars`), também identificámos a presença de inconsistências na utilização de espaços e tabs (`no-mixed-spaces-and-tabs`).

**Code Styling**:

Encontramos 4383 ocorrências de várias naturezas, sendo os erros de indentação os mais frequentes. No entanto, também identificámos outros tipos de erros, incluindo:

- Existência de referências a variáveis que não foram previamente declaradas (`no-undef`).
- Erros de espaçamento em declarações de setas das arrow functions (`arrow-spacing`).
- Preferência por `let` e `const` em vez de `var` (`no-var`).
- Utilização de sintaxe shorthand para propriedades de objetos (`object-shorthand`).
- Preferência por espaços em vez de tabs (`no-tabs`).
- Presença de uma nova linha no final do ficheiro (`eol-last`).

Tabela *11* - Pasta *www* (Erros de Sintaxe)

| Especificação | Nº de Ocorrências | Ficheiros | Exemplos |
|---|---|---|---|
| no-unused-vars | 23 | users.js<br>home.js<br>helper.js<br>clients.js | `'_', '__', 'e', 'settings', 'switchUserMessaging', 'getBrowser', 'Client', 'User', 'JobInfo', 'JobTyperequest', 'getClientsLocalStorage', 'getUsersLocalStorage', 'getUserJobsLocalStorage'` |
| no-undef | 193 | clients.js<br>helper.js<br>home.js<br>login.js<br>users.js | `'$', 'Client', 'saveClientsLocalStorage', 'getClientsLocalStorage', 'JobInfo', 'userDetails', 'JobTyperequest', 'saveUserJobsLocalStorage',` |

| | | | |
|---|---|---|---|
| | | | 'getUserJobsLocalStorage',<br>'resetLocalStorage',<br>'getBrowser',<br>'SaveUserDetails' |
| no-mixed-spaces-and-tabs | 2 | clients.js<br>users.js | Mixed spaces and tabs |

Tabela *12* - Pasta *www* (Erros de *Code Styling*)

| Nº de Ocorrências | Ficheiros | Exemplos |
|---|---|---|
| 4383 | clients.js<br>helper.js<br>home.js<br>login.js<br>users.js | Indent,<br>Semi,<br>space-before-function-paren,<br>prefer-const,<br>no-undef,<br>quotes,<br>arrow-spacing,<br>space-before-blocks,<br>space-infix-ops,<br>n/no-callback-literal,<br>brace-style,<br>object-curly-spacing,<br>object-shorthand,<br>no-multiple-empty-lines,<br>spaced-comment,<br>no-var,<br>no-trailing-spaces,<br>no-mixed-spaces-and-tabs,<br>no-tabs,<br>eol-last |

### Análise dos Resultados Obtidos

Em síntese, os *code linters* não se limitam a ser meros corretores de erros no código, eles além de identificar e corrigir erros, conseguem também identificar práticas estilísticas e estruturais.

Os *code linters* são os arquitetos da qualidade, que ajudam um código a ser robusto, coeso e duradouro, corrigindo e garantindo a qualidade a longo prazo.

## Testes Unitários

Neste capítulo, vamos explorar e apresentar algumas ferramentas disponíveis para a realização de testes unitários.

Depois, iremos explicar a nossa escolha entre essas ferramentas, detalhando os critérios que influenciaram essa decisão. Além disso, forneceremos exemplos práticos de como esses testes unitários foram implementados e executados no projeto, esclarecendo a importância e o funcionamento dos mesmos.

### Principais Funcionalidades

As *frameworks* para testes unitários são essenciais no processo de desenvolvimento de *software*, pois possibilitam a verificação automatizada e sistemática do comportamento de partes específicas do código, além de fornecerem uma estrutura que simplifica a criação, execução e manutenção dos testes, assegurando a qualidade do software produzido.

## Exemplos de Frameworks

Neste relatório, serão exploradas três *frameworks* populares para testes unitários em *JavaScript*: o *Jest*, o *Mocha* e o *Unit.js*.

Cada uma dessas *frameworks* tem características distintas que contribuem para a eficiência na criação e execução de testes.

Tabela *13 - Unit.js*

| | | |
|---|---|---|
| **Vantagens** | Sintaxe clara e expressiva<br>Extensibilidade<br>Integração Simples |  |
| **Desvantagens** | Comunidade pequena<br>Poucos recursos integrados | |
| **Aspetos Técnicos** | Sintaxe fluente<br>Asserções encadeadas | |

Tabela *14 - Mocha*

| | | |
|---|---|---|
| **Vantagens** | Altamente configurável<br>Flexível para adaptar-se a necessidades<br>Suporte a vários *browsers* |  |
| **Desvantagens** | Configuração inicial complicada<br>*API* pouco descritiva | |
| **Aspetos Técnicos** | Requer instalação de *plugins* para recursos<br>Uso de funções de *callback* para execução<br>Utilização de *JavaScript* puro | |

Tabela *15 - Jest*

| | | |
|---|---|---|
| **Vantagens** | Configuração simples<br>Rápido e poderoso<br>Fácil integração com *frameworks* como o *React* |  |
| **Desvantagens** | Complexidade elevada<br>Integração limitada com *browsers* | |
| **Aspetos Técnicos** | Suporte nativo para testes de *snapshot*<br>Utilização de uma *API Jest* própria | |

## Escolha da Ferramenta

Optámos pelo *Jest*, uma *framework open-source* projetada para trabalhar especificamente com aplicações *web,* pelo facto do *Jest* permitir iniciar os testes sem exigir alterações extensivas nas configurações prévias, tornando a execução dos testes mais ágil.

## Aplicação da Ferramenta no Projeto

Para garantir a integridade e o funcionamento adequado do projeto, foram realizados vários testes unitários nos arquivos **authentication-handlers.js**, **globalHandlers.js**, **clients-handlers.js**, **jobs-handlers.js**, **messaging-handlers.js** e **users-handlers.js,** todos localizados na pasta "***scripts***", que trata da lógica e manipulação de dados do servidor.

- **Authentication-handlers.js:** É responsável por autenticar os utilizadores no sistema. Os testes foram concebidos para validar o processo de *login*, verificando as credenciais e assegurando o correto armazenamento em sessão após a autenticação.
- **GlobalHandlers.js:** É o ficheiro que contém a função de *logout* do utilizador no sistema.
- **Clients-handlers.js:** É o ficheiro que contém as funções gerem os clientes no sistema e os testes unitários realizados validam operações como obtenção, edição, remoção e criação de clientes, garantindo a execução correta das operações e o tratamento adequado em diferentes cenários.
- **Jobs-handlers.js:** É responsável por gerir os *jobs* no sistema. Os testes realizados validam operações como listagem, edição, criação, reabertura e ordenação de prioridades, assegurando a execução correta das operações e uma gestão eficaz dos *jobs*.
- **Messaging-handlers.js**: Tem funções que são cruciais para a troca de mensagens e comunicação em tempo real. Os testes foram concebidos para garantir a manipulação adequada das mensagens, verificando a inserção e recuperação na base de dados, mantendo a integridade do sistema de comunicação.
- **User-handlers.js**: Gere os utilizadores no sistema e os testes unitários realizados validam operações como obtenção, criação, edição e exclusão de utilizadores, além de verificar as configurações de página para diferentes tipos de utilizadores.

## Resultados Obtidos

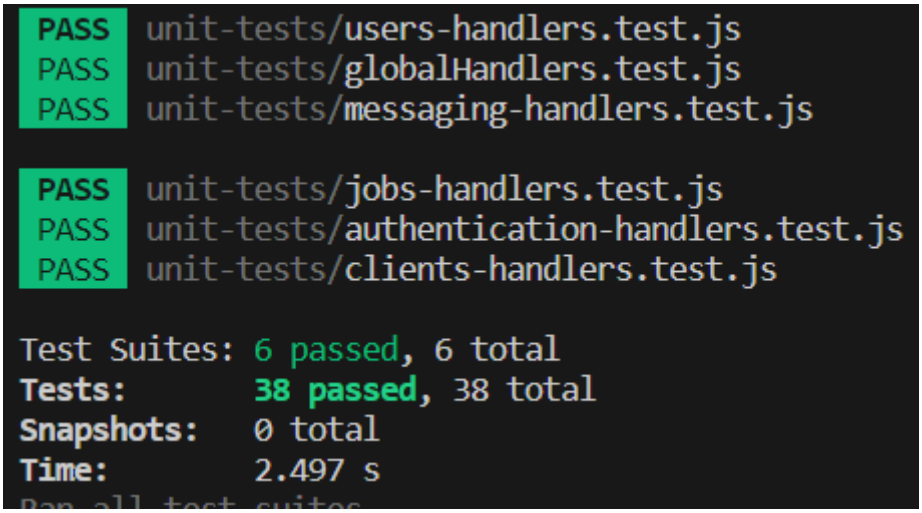


Figura *1* - Resultados obtidos nos testes unitários

Os resultados da figura 1 demonstram a execução bem-sucedida de todos os conjuntos de testes associados aos *handlers* do sistema. Cada um dos ficheiros de teste evidenciou um bom desempenho, confirmando a solidez e a correta implementação das funcionalidades, mas também a fiabilidade e a estabilidade do sistema como um todo.

Num total de seis conjuntos, todos eles foram concluídos sem falhas, apresentando um resultado de 38 testes individuais bem-sucedidos em 38 testes no total, com um tempo de execução total de 2.497 segundos, o que revela uma performance satisfatória, considerando o número de testes que foram realizados.

# Normas *ISO*

As normas *ISO*, são um conjunto de diretrizes e requisitos estabelecidos pela *International Organization for Standardization* (*ISO*) para garantir uma entrega de *software* de alta qualidade.

Essas normas definem critérios específicos para características de qualidade, métricas e procedimentos de avaliação em ambientes de desenvolvimento de *software, além de* permitirem o estabelecimento de padrões consistentes, aprimorando continuamente os processos de desenvolvimento e demonstrando a conformidade com critérios reconhecidos globalmente.

## Escolha da Norma

Após uma extensa pesquisa sobre as normas *ISO* disponíveis, concluímos que a norma ***ISO/IEC* 25000** seria a mais apropriada para o projeto em causa.

Esta norma também conhecida como *SQuaRE* (*Systems and Software Quality Requirements and Evaluation*), visa estabelecer uma estrutura para a avaliação da qualidade de produtos de *software*.


*Figura 2 - SQuARE ISO/IEC 25000*

## Divisões da ISO/IEC 25000

Esta norma está dividida em 4 partes, contribuindo cada uma para o objetivo final de ter um projeto com qualidade.

***ISO/IEC* 2501n – Divisão da Qualidade**

- ***ISO/IEC* 25000**: Ajuda a avaliar a qualidade do *software,* explicando os documentos das normas e oferecendo orientações para garantir que o *software atende* aos padrões de qualidade esperados.
- ***ISO/IEC* 25001**: Oferece diretrizes para planear e gerir a definição de requisitos e avaliação de *software, estabelecendo* uma base sólida para garantir que os critérios de qualidade sejam compreendidos e seguidos ao longo do desenvolvimento.

**ISO/IEC 2501n – Divisão do Modelo de Qualidade**

Este modelo determina quais serão as características de qualidade que vão ser consideradas para avaliar as propriedades de um produto de software.

- **ISO/IEC 25010**: Descreve o modelo em 8 características e sub-características para a qualidade do *software:*

    1. **Adequação Funcional**: O grau em que o sistema fornece funções que atendem às necessidades declaradas e implícitas quando utilizadas em condições específicas.
        - Completude Funcional.
        - Correção Funcional.
        - Adequação Funcional.
    2. **Desempenho**: Quantidade de recursos usados nas condições declaradas.
        - Comportamento Temporal.
        - Utilização de Recursos.
        - Capacidade.
    3. **Compatibilidade**: Avalia se o produto pode trocar informações com outros produtos ou sistemas.
        - Coexistência.
        - Interoperabilidade.
    4. **Usabilidade**: Avalia se o produto pode ser usado por utilizadores específicos para atingir metas especificadas com eficácia, eficiência e satisfação.
        - Reconhecimento de Adequabilidade.
        - Aprendizagem.
        - Operacionalidade.
        - Proteção contra erros de utilizador.
        - Estética da interface.
        - Acessibilidade.
    5. **Confiabilidade**: Avalia o grau para o qual o sistema executa funções.
        - Maturidade.
        - Disponibilidade.
        - Tolerância a falhas.
        - Recuperabilidade.
    6. **Segurança**: Avalia se o sistema protege informações e dados. Avalia também os níveis de autorização para os utilizadores.
        - Confidencialidade.
        - Integridade.
        - Não Repúdio.
        - Autenticidade
        - Responsabilidade
    7. **Manutenção**: Avalia se o sistema pode ser modificado facilmente com o fim de melhorá-lo, corrigi-lo ou adaptá-lo a mudanças no ambiente e nos requisitos.
        - Modularidade.
        - Reutilização.
        - Analisabilidade.
        - Modificabilidade.

        - Testabilidade.
    8. **Portabilidade**: Avalia se um sistema pode ser transferido de um hardware, software ou de outro ambiente operacional para outro.
        - Adaptabilidade.
        - Capacidade de instalação.
        - Substituição.
- **ISO/IEC 25012**: Concentra-se na qualidade dos dados como parte de um sistema e define características de qualidade para dados de destino.
    1. **Qualidade Inerente de Dados**
        - Precisão Sintática
        - Precisão Semântica
        - Completude
        - Consistência
        - Credibilidade
        - Atualidade
    2. **Qualidade de Dados inerente e Dependente do Sistema**
        - Acessibilidade.
        - Conformidade.
        - Confidencialidade.
        - Eficiência.
        - Precisão.
        - Rastreabilidade.
        - Compreensibilidade.
    3. **Qualidade de Dados dependente do sistema**
        - Disponibilidade.
        - Portabilidade.
        - Recuperabilidade.

**ISO/IEC 2502n – Divisão de Medição de Qualidade**

Esta ISO inclui modelos de referência de medição da qualidade do produto, definições matemáticas de medidas de qualidade e orientação prática para a sua aplicação.

- **ISO/IEC 25020**: Guia e modelo de referência de medição.
- **ISO/IEC 25021**: Elementos de medida de qualidade.
- **ISO/IEC 25022**: Medição da qualidade em uso.
- **ISO/IEC 25023**: Medição de sistema e qualidade do produto de software.
- **ISO/IEC 25024**: Medição de qualidade de dados.

**ISO/IEC 2503n – Divisão de Requisitos de Qualidade**

Define o padrão que forma a divisão dos requisitos de qualidade e ajuda a especificá-los. Esses requisitos de qualidade podem ser então utilizados como entrada para um processo de avaliação.

- **ISO/IEC 25030**: Fornece os requisitos e orientações para o processo usado para desenvolver requisitos de qualidade.

**ISO/IEC 2504n – Divisão de Avaliação de Qualidade**

São os padrões que formam os requisitos, recomendações e diretrizes para avaliação do software.

- **ISSO/IEC 25040**: Fornece uma estrutura para avaliar a qualidade do produto de software.
    1. **Estabelecer o propósito da avaliação**
        1. Obter os requisitos de qualidade do produto de software.
        2. Identificar as peças do produto a serem incluídas na avaliação.
        3. Definir o rigor da avaliação.
    2. **Especificar a avaliação**
        1. Selecionar as medidas de qualidade.
        2. Definir critérios de decisão para medidas de qualidade.
        3. Definir critérios de decisão para avaliação.
    3. **Projetar a avaliação**
        1. Planear atividades de avaliação.
    4. **Executar a avaliação**
        1. Fazer medições.
        2. Aplicar critérios de decisão para medidas de qualidade.
        3. Aplicar critérios de decisão para avaliação.
    5. **Concluir a avaliação**
        1. Rever o resultado.
        2. Criar um relatório de avaliação.
        3. Rever a avaliação da qualidade e fornecer feedback.
        4. Realizar disposição de dados de avaliação.

Como foi possível verificar, esta é uma norma que consegue alcançar quase tudo o que diz respeito à qualidade de software, sendo esse o principal motivo que fundamenta a nossa escolha.

## Críticas ao Projeto

Embora a norma ISO selecionada seja complexa, é importante considerar o objetivo inicial do projeto, que pode não ter sido concebido para atender a todas as normas de qualidade de software, mas sim para explorar conceitos básicos de programação web.

Por essa razão, optaremos por selecionar apenas algumas características desta norma ISO que consideramos justas e pertinentes para avaliar no projeto.

### Segurança

Apesar de se tratar de um projeto pequeno, temos que ter sempre em conta a segurança que a plataforma proporciona aos seus utilizadores.

Segundo a norma **ISO/IEC 25010 > Segurança > Confidencialidade**, percebemos que este projeto não tem confidencialidade, visto que os *developers* conseguem aceder aos scripts de criação da base de dados que contém todos os dados dos utilizadores, bem como o facto das passwords na base de dados não se encontrarem encriptadas, o que faz com que qualquer pessoa possa consultar a base de dados e visualize as passwords de qualquer utilizador.

## Confiabilidade

Neste projeto, notámos uma ausência de arquivos (*readme.md*) que ajudassem os de*velopers* na configuração inicial do projeto. Esta escassez faz com que com o tempo, a manutenção e implementação de alterações se tornem desafiadoras.

Por ser um projeto pequeno, essa falta de documentação compromete a **Confiabilidade** e especialmente a **Maturidade**, da norma **ISO/IEC 25010** desde o início.

## Usabilidade

Dentro da norma Isso **ISO/IEC 25010 > Usabilidade** temos vários pontos que demonstram falhas.

**Reconhecimento de Adequabilidade**

Neste projeto existe uma funcionalidade que é a **Prioridade do Serviço**, tal como pode ser observado na figura 3.

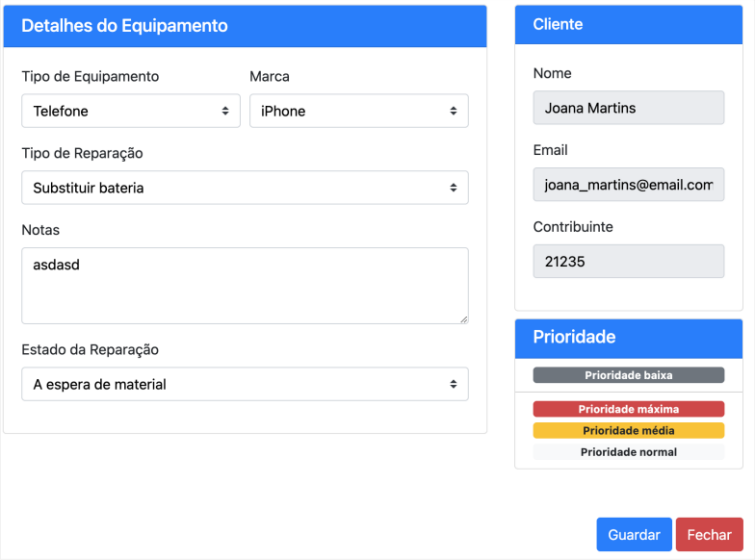


Figura *3* - Prioridade do Serviço

Ao criar um serviço, a prioridade é inicialmente definida como baixa. No entanto, caso o utilizador deseje modificar essa prioridade, depara-se com um *drag & drop*.

Este comportamento, exclusivo desta página, gera confusão para o utilizador e estraga a experiência do mesmo, já que não é intuitivo para definir uma prioridade.

Além de não atender à norma de **Reconhecimento de Adequabilidade**, isso também afeta a norma de **Manutenção > Reutilização**.

## Acessibilidade

Na plataforma, notámos a ausência de suporte ao utilizador para a conclusão de tarefas.

Para fazer face a esse problema, sugerimos o uso de ícones informativos (i), detalhando cada campo ou funcionalidade, ajudando assim os utilizadores ao fornecer informações específicas sobre seu significado e uso.

Além disso, não foram adotadas cores adequadas para pessoas com daltonismo. Existem várias ferramentas disponíveis que geram paletas de cores adaptadas para atender a esse público específico, e no nosso entender ao incorporar essas paletas iriamos melhorar a acessibilidade para todos os utilizadores.

## Adequação Funcional

O projeto tem uma funcionalidade de *logout* que não funciona como é suposto.

Ao pressionar o botão de *logout*, não somos redirecionados para a página de *login*, o que não nos permite iniciar a sessão com outro utilizador.

O facto dessa funcionalidade não funcionar significa que não estamos a cumprir com a **Adequação Funcional** desta norma ISO.

## Divisão de Requisitos de Avaliação de Qualidade

Para este projeto não foram proporcionados nenhuns requisitos de qualidade que nos permitisse então realizar a avaliação dos mesmos para garantir que estava tudo conforme o esperado e planeado.

Sendo assim, não será possível cumprir as normas **ISO 2503n** e **ISO 2504n**.

# Conclusão

Este projeto representou um desafio significativo que nos proporcionou valiosas lições. Durante o processo, através da análise do código, extraímos requisitos e aplicámos ferramentas modernas de teste para avaliar tanto aspetos técnicos quanto de usabilidade.

Ao longo do desenvolvimento, enfrentámos desafios em diferentes áreas, desde a identificação de problemas de código até questões de conformidade com as normas *ISO* para qualidade de *software*. Esta experiência destacou o quão importante é ter uma documentação detalhada, testes abrangentes e a adesão a padrões de qualidade reconhecidos.

Através deste projeto, aprimorámos igualmente as nossas habilidades de análise, resolução de problemas e compreensão das melhores práticas de desenvolvimento, que serão importantes para o futuro com vista a desenvolver soluções mais robustas e alinhadas com os padrões de qualidade exigidos.