Licenciatura em Engenharia Informática

**Relatório do Trabalho prático Linguagens Script 2021/2022**

Jorge Ricardo Marques Duarte – a2021110042

João Alexandre Caravela Marques – a2021146383

Rodrigo Delgado da Cruz – a2019122799

# Introdução

Este relatório surgiu no âmbito da cadeira de Linguagens Script com o objetivo de especificar todas as funcionalidades implementadas bem como as soluções utilizadas no desenvolvimento da aplicação, especificações de componentes e outros elementos utilizados durante o desenvolvimento.

# Equipa de trabalho

O trabalho prático foi realizado pelos seguintes alunos:

* Jorge Ricardo Marques Duarte
* João Alexandre Caravela Marques
* Rodrigo Delgado da Cruz

# Ambiente de desenvolvimento

O programa foi desenvolvido em Visual Studio Code, um editor de código otimizado para criar e fazer depuração de aplicações modernas da *Web* e *Cloud*.

Foi criado um repositório privado no Github, permitindo o armazenamento e o desenvolvimento da aplicação através de *commits* e *pushs para a branch* principal. Isto demonstrou ser muito útil na parte de organização e gestão de código entre o grupo.

# Tecnologias e ferramentas utilizadas

Para facilitar o desenvolvimento, termos uma forma de programação concisa e organizada e ainda aumentar a produtividade, foram adicionadas ao projeto várias ferramentas, apresentadas em seguida.

Começámos por utilizar o *“Create React App”* que tem um conjunto de pré configurações e não é necessária a instalação de outros packages como Babel ou Webpack para a criação do projeto.

A primeira decisão tomada na realização do projeto foi que todo o desenvolvimento seria realizado na linguagem de programação Typescript, sendo esta uma linguagem fortemente tipada, o que facilitou imenso na depuração da aplicação e prevenção de bugs durante o seu desenvolvimento.

Além da adição de Typescript, fizemos a adição de um package importante ao projeto: o Google's Type Script style guide (gts). Este *package* é um conjunto de configurações predefinidas da google com estilos de código e regras para o *eslint* e *prettier*.

Para o *eslint* funcionar corretamente e para que as regras sejam aplicadas automaticamente durante a gravação de um ficheiro, foi adicionada a extensão *eslint* do *marketplace*, permitindo ao Visual Studio Code a utilização destas regras presentes no projeto.

Para garantirmos que todos os *commits* fossem sem problemas para a *branch* no Github, adicinámos o Husky Hooks, outro *package* que nos permite fazer pré validações de *commits* e *pushs* para a respetiva *branch*.

Além destas configurações e já com a base do projeto pronta para começarmos o desenvolvimento, foi adicionado o Docker ao projeto. Isto permite a criação de imagens para que, por sua vez, sejam utilizadas em contentores num servidor Linux.

Para que a *build* da solução seja efetuada corretamente, estamos a importar a versão do node 16 e a fazer a sua instalação para a criação da imagem.

No ficheiro docker-compose foram definidos os respetivos serviços utilizados. Neste caso específico que apenas tem uma aplicação web, contem um serviço com o nome isec\_react\_soup\_letter que utiliza a imagem do próprio projeto.

Além disto, fizemos uso do “Github workflows” para as publicações automáticas da aplicação para um servidor Linux, que por sua vez tem o projeto atual hospedado no domínio <https://isec-react.overwrite.pt>.

Por último, adicionámos ao projeto o package “Redux” e as suas respetivas dependências, permitindo assim ter apenas uma *store* de estado global que facilita bastante a gestão de estados de componentes.

# Funcionalidades da aplicação

## Estrutura de pastas

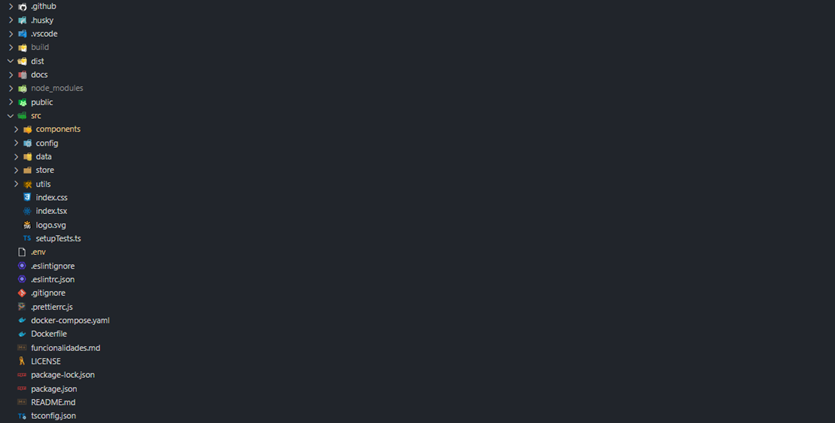
O projeto foi organizado de modo a facilitar a criação de imagens e ter a separação de gestão de dados da *store* e dos componentes, assim como pastas para ficheiros de configuração, pastas para ficheiros de documentação e pasta para ficheiros utilitários.

Figura - Apresentação da estrutura de pastas da aplicação

## Componentes principais

### Componente Start

O componente Start, é considerado o componente principal pois é através dele que, dependendo dos dados que estão na *store*, são apresentados os componentes seguintes recorrendo a operadores ternários para verificar o estado do jogo. Se este estiver finalizado apresenta o componente final com as classificações, caso este seja inicializado apresenta o jogo propriamente dito ou a interface para escolha de dificuldade e nome de utilizador.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Componente principal para apresentação de outros componentes

### Componente Board

O componente Board é o componente com mais complexidade da aplicação e no qual está contida toda a lógica de jogo, desde a escolha de palavras até à renderização das palavras no tabuleiro de jogo. Recorremos à utilização do estado do componente (useState) e da *store* da aplicação de forma a perceber se a seleção efetuada foi em linha, coluna ou na diagonal uma vez que o utilizador apenas indica a primeira e última posição onde está contida a palavra.

Este componente está a fazer uso de múltiplos useEffects, pois quando o utilizador faz a escolha da primeira e da última letra, estamos a chamar a função “VerifyFoundWord()” de forma a verificar se a palavra selecionada está presente ou não na lista de palavras a encontrar.

Por defeito, sempre que o componente é renderizado, estamos a utilizar o *hook* useEffect para chamar a função “startWordRenderization()”, que vai renderizar as palavras na matriz de forma aleatória.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Apresentação geral do componente Board

### Componente Congratulations

O componente Congratulations é bastante importante para o projeto uma vez que é o único na aplicação a fazer uso da *localStorage*. Esta *localStorage* é utilizada para guardar no lado do cliente, ou seja, no *browser*, a classificação do utilizador. O *useEffect* responsável por guardar a classificação só é despoletado caso o tempo de jogo não tenha sido esgotado e o utilizador tenha conseguido encontrar todas as palavras presentes na lista de palavras. Uma imagem com texto

Descrição gerada automaticamente

Figura - Apresentação geral do componente Congratulations

Com base no resultado do jogo é apresentada a respetiva informação, ou seja, se o utilizador ganhou é apresentado um ecrã a felicitá-lo com a informação da classificação que o mesmo obteve, o tempo utilizado e uma lista com as 10 melhores pontuações de todos os jogos ocorridos. Caso o tempo de jogo esgote é apresentado um ecrã a informar o utilizador deste acontecimento.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Apresentação do useEffect que guarda a classificação em localStorage

## Store da aplicação

Com a utilização do Redux criámos uma *store* principal com o nome *“gameboard”* onde existe um estado inicial “*initialState”* que contém todas as propriedades utilizadas na aplicação. Estas propriedades são utilizadas para definir se determinados componentes devem ou não serem apresentados, para ter especificações da dificuldade de jogo selecionada tais como a quantidade de linhas, a quantidade de colunas, quantas palavras o utilizador deve encontrar, o número de palavras encontradas, o estado do jogo, entre outras propriedades que podem ser observadas na Figura 6.Uma imagem com texto

Descrição gerada automaticamente

Figura 6 – Estado inicial da store

A *store* tem, por sua vez, vários *extra reducers* que vão modificar os estados definidos no nosso *initialState* ao longo da aplicação, tendo funções como definição de palavras, atualização de matrizes, adição de palavras encontradas, adição de nome de utilizador, alterações de estados de jogo entre outras.

Para a correta funcionalidade dos nossos *extra reducers*, foi criado um ficheiro chamado “board-actions.ts”, onde está contida toda a lógica de alteração de dados e que é posteriormente enviado para os *extra reducers* na *store* de forma a fazer a alteração dos estados.

## Configurações

O projeto em termos de configurações é bastante simples, apenas existe a necessidade de garantir a existência de um ficheiro “.env” no projeto uma vez que está a ser utilizado o *docker* para a criação de imagens e por sua vez é feita a sua publicação num contentor no servidor Linux. Neste ficheiro tem de estar definida a porta que queremos abrir e utilizar internamente na aplicação para evitar problemas na publicação, uma vez que a porta 3000 predefinida pelo React poderá já estar em utilização.

# Conclusão

Considerámos este trabalho bastante interessante pois foi possível aplicar os conceitos aprendidos nas aulas teóricas, dados através de apresentações práticas de exercícios ao longo do semestre. O desenvolvimento desta aplicação foi essencial na consolidação e aplicação dos conhecimentos obtidos e permitiu-nos ter a perceção de como é feita a implementação de uma aplicação em React, na sua totalidade. Na nossa opinião a abordagem de React na disciplina permite maior facilidade de entrada no mundo de trabalho tendo em conta a procura e oferta para a utilização desta libraria.