



UNIVERSIDADE
LUSÓFONA

BoxIt

Frontend

Relatório Final

Bernardo Barardo | 21807838

Trabalho Final de Curso | Licenciatura em Engenharia Informática | 23/07/2021

Orientado por: Miguel Tavares

www.ulusofona.pt

Direitos de cópia

BoxIt - Frontend, Copyright de Bernardo Barardo, ULHT.

A Escola de Comunicação, Arquitectura, Artes e Tecnologias da Informação (ECATI) e a Universidade Lusófona de Humanidades e Tecnologias (ULHT) têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objectivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

Resumo

O BoxIt é uma plataforma que começou a ser desenvolvida no ano passado, tem por objetivo virtualizar redes de computadores, com o propósito de auxiliar no desenvolvimento de sistemas distribuídos. Durante o desenvolvimento destes sistemas é comum ocorrerem certos problemas, tais como testar o sistema em várias máquinas (quer físicas quer virtuais), a configuração necessária para que tal aconteça, entre outros. A virtualização levanta um outro problema, que está relacionado com os recursos necessários para a virtualização das várias componentes do sistema. É aqui que o BoxIt tenta mitigar este problema, recorrendo a uma técnica de virtualização mais otimizada.

Como referido anteriormente, já existe trabalho desenvolvido em prol desta ideia, trabalho este que é um *backend* desenvolvido e praticamente concluído, este trabalho tem como objetivo principal criar um *frontend* para auxiliar a interação com esta plataforma, ou seja, integrar processos como sistema de *login*, *upload* de ficheiros, automatização de certos procedimentos e acesso remoto a máquinas virtuais.

Abstract

BoxIt is a platform that started to be developed last year, it aims to virtualize computer networks, with the purpose of assisting in the development of distributed systems. During the development of these systems, it is common to experience certain problems, such as testing the system on several machines (either physical or virtual), the necessary configuration for this to happen, among others. Virtualization raises another problem, which is related to the resources necessary for the virtualization of the various components of the system. It is here that BoxIt tries to mitigate this problem, using a more optimized virtualization technique.

As previously mentioned, there is already work developed in support of this idea, this work is a backend developed and practically completed, this project has as main objective to create a frontend to assist the interaction with this platform, that is, to integrate processes such as login system, file upload, automation of certain procedures and remote access to virtual machines.

Índice

1	Identificação do Problema	6
1.1	Contexto Geral / descrição do problema.....	6
1.2	Exemplo de um caso real	6
1.3	O que já foi feito	9
1.4	O que se vai fazer.....	10
2	Levantamento e análise dos Requisitos.....	11
3	Viabilidade e Pertinência	15
4	Solução Desenvolvida	15
4.1	Arquitetura	15
4.2	Tecnologias:.....	16
4.2.1	Docker	16
4.2.2	React	16
4.2.3	Redux.....	19
4.2.4	Axios.....	20
4.2.5	Node Package Manager (NPM).....	20
4.2.6	Secure Socket Shell (SSH)	20
4.3	Arquitetura do <i>frontend</i>	21
4.4	Implementação	23
5	<i>Benchmarking</i>	24
6	Método e Planeamento	27
6.1	Calendário	27
6.1.1	Descrição de certos pontos:	27
6.1.2	Cumprimento do calendário	29
7	Resultados	29
7.1	Interface	29
7.2	<i>Deployment</i>	35
8	Conclusão e trabalhos futuros	37
	Bibliografia.....	38
	Anexos.....	40
	Anexo 1 – Levantamento de requisitos.....	40

Requisitos Funcionais:	40
Requisitos Não Funcionais:.....	45
Anexo 2 – Manual de instruções do Boxit (<i>Frontend</i>)	47
Glossário.....	49

Lista de Figuras

Figura 1 - Arquitetura Cliente-Servidor [1]	6
Figura 2 - Arquitetura do projeto de Redes de Computadores	7
Figura 3 - Erro ao iniciar segundo cliente.....	7
Figura 4 - Erro de ligação do segundo cliente	8
Figura 5 - Utilização de um segundo computador como solução.....	8
Figura 6 - Utilização de máquina virtual como solução	9
Figura 7 - Docker container vs. Máquina Virtual [2]	10
Figura 8 - Arquitetura da solução	15
Figura 9 - Exemplo de uma componente React.js [16].....	17
Figura 10 - Focando o estado do exemplo. [16]	18
Figura 11 - Fluxo dos dados em React.js. [17]	18
Figura 12 - Estrutura em árvore dos componentes [17].....	19
Figura 13 - Estrutura com a adição da store do Redux. [17]	20
Figura 14 - Arquitetura do frontend.....	21
Figura 15 - Estrutura de ficheiros.	22
Figura 16 - Erro de CORS no browser	23
Figura 17 - Não obtenção de resposta do backend	23
Figura 18 - Resultados do benchmark de tempo de inicialização.....	26
Figura 19 - Comparação de funcionalidades entre frameworks	27
Figura 20 - BoxIt home page	30
Figura 21- BoxIt signup page	30
Figura 22 - Email de confirmação	31
Figura 23- BoxIt login page.....	31
Figura 24 - BoxIt projects page	32
Figura 25 - BoxIt project page	33
Figura 26 - BoxIt new project page	34
Figura 27 - Página de confirmar conta (antes).....	35
Figura 28 - Página de confirmar conta (depois).....	36
Figura 29 - Página conta já ativada (antes).....	36
Figura 30 – Página conta já ativada (depois).....	36

Lista de Tabelas

Tabela 1 - Calendário Proposto	28
--------------------------------------	----

1 Identificação do Problema

1.1 Contexto Geral / descrição do problema

O desenvolvimento de sistemas distribuídos levanta diversos desafios que um sistema dito normal não levanta devido ao menor grau de complexidade.

Tendo isto em conta, foi criada uma plataforma que virtualiza redes, através de máquinas virtuais, com os diversos componentes de um sistema distribuído que se queira testar. Contudo a plataforma no seu estado atual não é muito “*user friendly*”, pois não tem nenhum *frontend*, sendo apenas possível de interagir através de uma *Application Programming Interface* (API) REST.

Este trabalho tem como objetivo desenvolver um *frontend* de forma a quebrar esta barreira.

1.2 Exemplo de um caso real

Na unidade curricular de redes de computadores da Universidade Lusófona de Humanidades e Tecnologias os alunos tiveram de desenvolver um projeto que seguia uma arquitetura cliente-servidor, isto é, um projeto onde existe um servidor que recebe pedidos de um ou mais clientes e que responde aos mesmos pedidos, como mostra a figura 1.

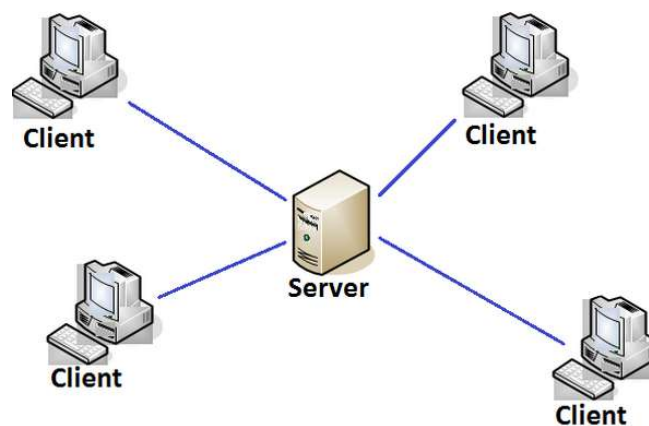


Figura 1 - Arquitetura Cliente-Servidor [1]

Este projeto consistia na criação de um servidor, onde o servidor que gerenciava os pedidos dos clientes e de um cliente que poderia pedir uma série de informação, tal como obter uma lista de utilizadores *online* ou enviar mensagens a um utilizador específico.

Para fins educativos, era necessário utilizar dois protocolos de comunicação o *Transmission Control Protocol* (TCP) e *User Datagram Protocol* (UDP) [3], onde o primeiro seria utilizado pelos clientes para enviar o pedido ao servidor, e o segundo seria a resposta do servidor, como apresenta a figura 2.

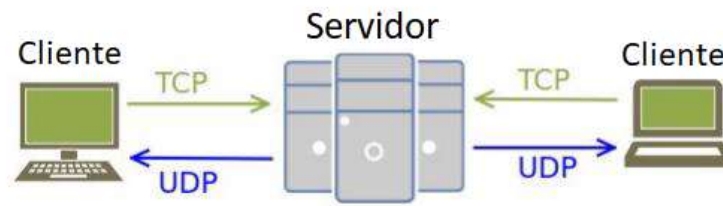


Figura 2 - Arquitetura do projeto de Redes de Computadores

Durante os testes que implicavam mais do que um cliente, os alunos utilizaram sempre o mesmo computador. Desta forma, depararam-se com o erro apresentado na figura 3.

```
java.net.BindException: Address already in use: Cannot bind
    at java.net.DualStackPlainDatagramSocketImpl.socketBind(Native Method)
    at java.net.DualStackPlainDatagramSocketImpl.bind0(DualStackPlainDatagramSocketImpl.java:84)
    at java.net.AbstractPlainDatagramSocketImpl.bind(AbstractPlainDatagramSocketImpl.java:93)
    at java.net.DatagramSocket.bind(DatagramSocket.java:392)
    at java.net.DatagramSocket.<init>(DatagramSocket.java:242)
    at java.net.DatagramSocket.<init>(DatagramSocket.java:299)
    at java.net.DatagramSocket.<init>(DatagramSocket.java:271)
    at Cliente3.<clinit>(Cliente3.java:67)
Exception in thread "Thread-0" java.lang.NullPointerException Create breakpoint
    at Cliente3.reciveEcho(Cliente3.java:113)
    at Cliente3$ConstantReciver.run(Cliente3.java:90)
    at java.lang.Thread.run(Thread.java:748)
```

Figura 3 - Erro ao iniciar segundo cliente

Este erro ocorria quando os alunos tentavam executar um segundo cliente no mesmo computador e está relacionado com a já ocupação do porto do protocolo UDP tal como ilustrado na figura 4.

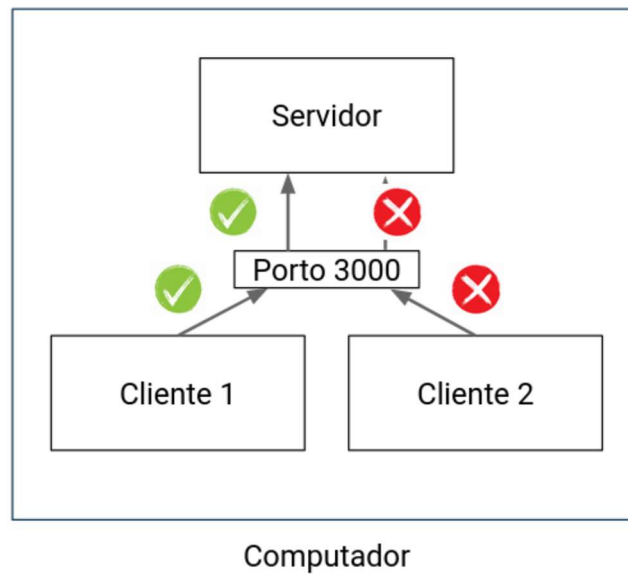


Figura 4 - Erro de ligação do segundo cliente

A solução mais imediata e óbvia para este problema talvez seja a utilização de um segundo computador, como demonstra a figura 5.

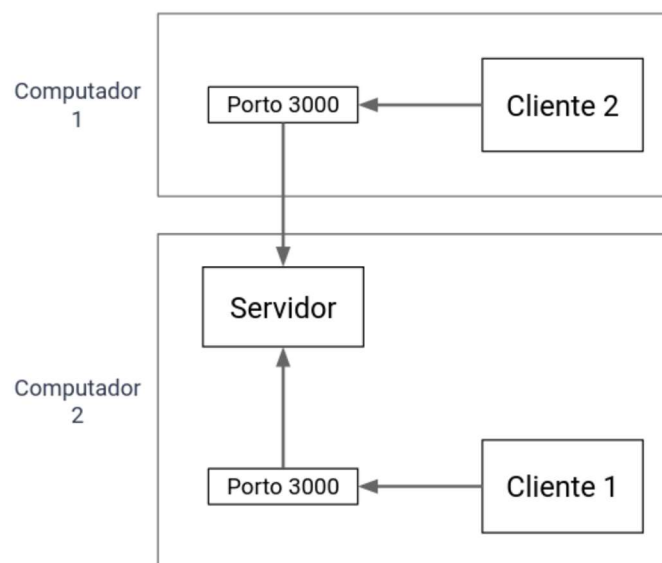


Figura 5 - Utilização de um segundo computador como solução

Esta opção pode não ser a mais viável para a maior parte dos estudantes uma vez que estes geralmente não possuem computadores suficientes.

Outra Solução seria a utilização de máquinas virtuais como demonstra a figura 6.

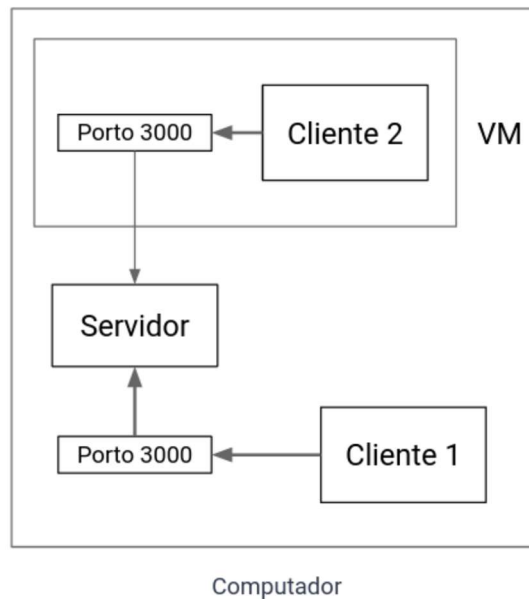


Figura 6 - Utilização de máquina virtual como solução

O problema desta solução é por vezes as configurações necessárias da própria máquina virtual, ou então a capacidade do computador a ser utilizado.

O BoxIt tem como objetivo criar uma resposta mais eficiente, isto é uma plataforma que não requeira tantos recursos de *hardware* como comparando com as máquinas virtuais tradicionais, e que permita a estes programadores testar o seu sistema distribuído em apenas um computador.

1.3 O que já foi feito

No contexto deste trabalho já foi desenvolvido no ano passado um *backend* para esta plataforma, esta foi construída utilizando as tecnologias:

- Node.js
- Docker
- Node Package Manager (NPM)
- JSON Web Token (JWT)

O *backend* serve para automatizar o processo de virtualização de componentes de projetos, passando a explicar; ele recebe estas componentes e de seguida cria Docker *containers* com um sistema operativo Linux básico que irão executar estas componentes. Desta forma fica-se com vários Docker *containers* a executar as diversas componentes do sistema distribuído. Um projeto no contexto desta aplicação é o conjunto de todas as componentes do sistema distribuído, voltando ao exemplo dos alunos da unidade curricular de redes de computadores, o servidor que recebia pedidos do cliente é uma componente e o cliente que envia pedidos (para enviar mensagens, saber os utilizadores *online*, etc.) é outra componente.

Este *backend* está disponível publicamente num servidor, cuja documentação pode ser acedida em: <https://boxit-edu.duckdns.org/api-docs/>. Junto com o servidor também se encontra em execução a base de dados.

O *backend* disponibiliza uma API REST e foi desenvolvido em Node.js. Tem como funções registar novos utilizadores, criar projetos, entre outras.

Com o uso do Docker é possível virtualizar máquinas utilizando menos recursos relativamente às virtualizações convencionais, tais como, por exemplo, o VirtualBox. Isto porque enquanto uma máquina virtual convencional cria uma virtualização tanto do sistema operativo como do *hardware*, já o Docker partilha o sistema operativo do *host* não virtualizando o *hardware*, tornando desnecessária a existência do *Hypervisor*, tal como ilustrado na figura 7.

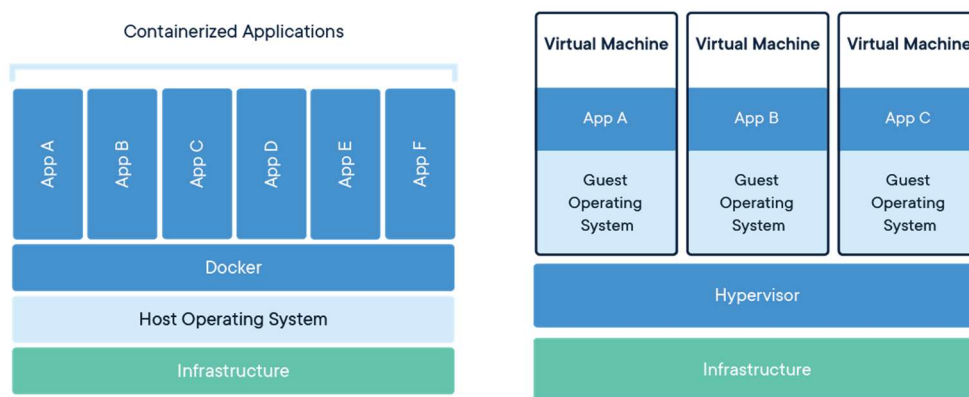


Figura 7 - Docker container vs. Máquina Virtual [2]

1.4 O que se vai fazer

Quanto à criação do *frontend* esta será feita utilizando a *framework* React.js que irá utilizar a API disponibilizada pelo *backend*.

O *frontend* terá todas as opções necessária para receber do utilizador e consequentemente enviar ao *backend* todos os dados necessários para a execução dos projetos.

2 Levantamento e análise dos Requisitos

Em anexo [Anexos 0] segue a listagem original dos requisitos para este trabalho e após alguma análise chegou-se à conclusão de que existem requisitos de maior complexidade como é o caso do RF4 pois, o *backend* não dispõe de um *endpoint* que permita o *reset* de uma password, assim como no requisito RF9_UI2, o *backend* não disponibiliza funcionalidades para poder manusear componentes individuais de um projeto, entre outros.

Existem requisitos que foram mais complexos do que inicialmente se expetava, pois o *backend* ainda não cumpria certas condições. Isto fez com que fosse necessário alterar/criar mais código no *backend* do que se pensou inicialmente. Nos requisitos em que este foi o caso será explicado com mais detalhe.

Dito isto segue a listagem do nome de todos os requisitos assim como o seu estado e/ou devidas justificações ou comentários:

RF1 - Registo do utilizador

Estado: Implementado

Uma vez na página inicial o utilizador consegue navegar até a página de registo e de seguida realizar o seu registo na plataforma e em caso de erro, é apresentada uma mensagem ao utilizador.

RF2 – Acesso via autenticação

Estado: Implementado

Na altura em que o utilizador se encontre na página inicial este consegue navegar até a página de *login* e de seguida entrar na plataforma, em caso de erro tanto no *frontend* como no *backend*, estes são apresentados dependendo de onde provém o erro (*frontend* ou *backend*) este é apresentado de forma diferente.

RF3 - Acesso sem autenticação (convidado)

Estado: Implementado

Foi adicionado um *endpoint* no backend o qual retorna o modo em que este executa (*single-user* ou *multi-user*), com esta informação o frontend pede ou não a autenticação do utilizador para aceder as funcionalidades da plataforma.

RF4 – Recuperar *password*

Estado: Não implementado

Infelizmente o *backend* não fornece as ferramentas necessárias para poder implementar este requisito, e o mesmo não se conseguiria implementar apenas com o *frontend* pois exigiria alterar dados da base de dados.

RF5 - Criar Projetos

Estado: Implementado

O utilizador após a autenticação pode ir a página dos projetos e criar um novo projeto, fornecendo o zip com as componentes do projeto, e os nomes e argumentos das diversas componentes, assim como o número de instâncias que o projeto irá conter.

RF6 - Criar componentes do projeto

Estado: Parcialmente implementado

Não é possível modificar ou criar componentes individuais de um projeto, o *backend* não fornece essa funcionalidade. Mas como é possível criar as diversas componentes no momento de criação do projeto.

RF7 - Administrar contas

Estado: Não implementado

Dado que a validação de utilizadores é feita através de um email que é enviado para o utilizador e não existe forma de obter uma lista de todos os utilizadores (existe sim forma de obter informação de um utilizador dado um email), a implementação deste requisito exigiria mais uma reestruturação do código para ser implementado.

RF8 - CRUD (*Create, Read, Update e Delete*) de projetos e qualquer tipo de componente dos mesmos (ex. JAR, zip).

Estado: Parcialmente implementado

RF8.1 - Criar

Estado: Implementado

Como referido no RF5 e RF6 é possível criar Projetos assim com as suas componentes.

RF8.2 - Read

Estado: Implementado

Na página dos projetos é possível obter a lista completa dos projetos do utilizador, onde é possível verificar algumas informações sobre os mesmos, e se clicar em algum dos projetos navegará para a página do mesmo onde obtém mais informação sobre o projeto assim como informações das suas componentes (estado, id, ip, entre outros).

Importante de se referir é que neste requisito o *endpoint* que o *backend* fornecia para obtenção de todas estas informações apenas devolvia o id dos projeto e datas de modificação (Informação provinda da base de dados), com isto nada se consegue fazer (aceder a *containers* assim como iniciar e parar).

Logo foi criado um novo *endpoint* (`/projects/state/{uuid}`) que devolve muito mais informação direta do Docker para o *frontend*. Este *endpoint* se não receber UUID devolve

a informação de todos os projetos do utilizador, caso receba UUID apenas devolve a informação do respetivo projeto.

RF8.3 - Update

Estado: Não implementado

Dado que o *backend* não suporta esta funcionalidade como se esperava inicialmente, este requisito ainda não se encontra implementado.

RF8.4 - Delete

Estado: Parcialmente implementado

Em diversas zonas da aplicação é possível eliminar projetos que o utilizador deseje, na página dos projetos e na página de cada projeto, aqui uma vez mais o *backend* não fornecia funções para eliminar componentes individualmente, sendo esta a parte que falta para este requisito ficar totalmente implementado.

RF9_UI1 - Executar múltiplas instâncias de cada componente do projeto

Estado: Implementado

Durante a criação do projeto é pedido ao utilizador que introduza o número de instâncias que deseja que corram da componente selecionada.

Para a implementação deste requisito também foram necessárias algumas alterações não convencionais para “enganar” o *backend*. O mesmo passa por altear o formData que lhe é enviado para ele criar as diversas instâncias que o utilizador desejou.

RF9_UI2 – Trabalhar com apenas certas componentes do projeto.

Estado: Implementado

A partir do ecrã de um projeto é possível parar e inicializar instâncias individuais.

É de se referir que estas funcionalidades não se encontravam disponíveis no *backend*, tendo sido necessária a criação de 4 novos *endpoints* no *backend* para estes objetivos, 2 *endpoints* para iniciar ou para projetos inteiros, e outros 2 para componentes individuais.

RF10 - Validação de campos de texto

Estado: Implementado

Na página de login e registo existe validações dos campos de acordo com as características dos requisitos, na página de criação dos projetos a mesma validação não existe também porque no caso dos argumentos pode haver necessidade de escrever caracteres mais estranhos que iram passar para o programa em si, e a existência de validações nestes campos poderia originar conflitos.

RF11 – Download de projetos

Estado: Implementado

Na página de um projeto foi adicionado um botão, o qual quando acionado executa o download do projeto para a máquina do utilizador.

Este foi um novo requisito que não se encontrava na lista inicial de requisitos.

RNF 1 – Acesso às instâncias executadas através do protocolo SSH (*Secure Socket Shell*)

Estado: Não implementado

Antes da realização deste TFC era espectável que esta operação fosse possível. No entanto a mesma não foi possível o acesso as instâncias via SSH.

Modificou-se e/ou adicionou-se novas configurações para que o acesso fosse exequível, mas o mesmo não se verificou.

RNF2 - HTTPS

Estado: Implementado

O servidor onde corre o *backend* do Boxit, executa toda a comunicação com o *frontend* via canal seguro, HTTPS e vice-versa.

RNF 3 - Docker

Estado: Implementado

Junto com o *source-code* do *frontend* encontram-se os ficheiros “Dockerfile” e “docker-compose.yml” que juntos facilitam imenso o *deployment* da aplicação num servidor.

RNF 4 - Comunicação entre *frontend* e *backend*

Estado: Implementado

Todas as comunicações entre o *frontend* e o *backend* são feitas de acordo com a sua documentação

RNF 5 – Instanciação dos novos *containers* ao mesmo nível do *backend*

Estado: Implementado

Quando o *backend* inicializa as novas instâncias, estas não são criadas dentro da sua máquina, mas sim dentro da máquina do *Host*. Desta forma toda as instâncias são *Guests* do mesmo *Host*.

Este foi mais um requisito que não estava presente na lista inicial, mas foi preciso de adicionar para resolver problemas do *backend*.

3 Viabilidade e Pertinência

Como descrito anteriormente, a plataforma já desenvolvida (BoxIt), tem a sua pertinência na área de desenvolvimento de aplicações. No entanto, no estado atual, é complicada de ser utilizada visto que não tem uma interface gráfica. Assim sendo, é necessário enviar todos os pedidos via REST com um programa do género Postman [15].

Pretende-se desenvolver um *frontend* que quebre esta barreira entre o utilizador e o BoxIt, tornando assim o seu acesso muito mais eficaz.

Pretende-se também disponibilizar o BoxIt num servidor *online*, onde retiramos problemas de configuração e instalação, tornando-o assim num sistema acessível por todos.

4 Solução Desenvolvida

4.1 Arquitetura

A arquitetura do BoxIt será composta por 3 camadas, o *frontend*, *backend* e a base de dados. Esta arquitetura encontra-se ilustrada no diagrama abaixo:

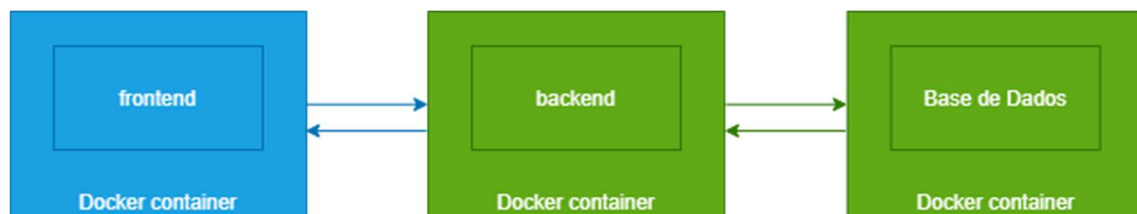


Figura 8 - Arquitetura da solução

Da figura 8 o que se encontra a cor verde é parte da solução que já foi desenvolvida e a azul é o que se irá desenvolver neste projeto.

1. **Frontend:** Esta camada permite que o utilizador interaja com o UI do sistema, permitindo-o executar tarefas como, registo, autenticação, criação / execução de projetos, entre outras;
2. **Backend:** Esta camada recebe e processa os pedidos provenientes de uma API REST que ela própria expõe. Trata-se do núcleo do sistema, é no *backend* onde por exemplo os projetos são criados e executados.
3. **Base de dados:** O *backend* guarda nesta camada toda a sua informação, como dados dos utilizadores e os seus projetos.

É de referir que a comunicação entre o *backend* e também é feita no formato de JSON, e em certos casos em form-data.

De forma a facilitar as tarefas de *deployment* assim como a gestão do BoxIt, cada componente desta arquitetura é executada dentro de um Docker *container*.

4.2 Tecnologias:

Os próximos subcapítulos explicam as principais tecnologias que iram ser utilizadas no desenvolvimento deste trabalho em suma as razões pelas quais estas foram escolhidas foras as seguintes:

- **Docker** – Facilitação de *deploy* da solução
- **React.js** – Foi ponderada a utilização de outra *framework*, nomeadamente Angular, devido ao facto de já ter sido lecionado em programação web, unidade curricular de LEI em 2019/2020, e portanto, já existir conhecimento prévio, mas eu quero tentar uma nova *framework* e vejo isto um pouco como um desafio.
- **Redux** – Facilitar o fluxo de informação dentro da aplicação.
- **Axios** – Facilitar a comunicação com o *backend*.
- **NPM** – Gestão de pacotes instalados no *frontend*.
- **SSH** – Protocolo de comunicação com as máquinas virtuais a executar no *backend*.
- **Gitlab CICD** – Facilitar o Deploy do frontend para a PaaS Heroku.

4.2.1 Docker

Docker é uma ferramenta desenhada para facilitar a criação, o *deploy* e o fluxo das aplicações utilizando *containers*. Estes *containers* permitem o desenvolvedor a agrupar uma aplicação e todas as suas componentes como bibliotecas e outras dependências e fazer *deploy* como um pacote único.

4.2.2 React

Para o desenvolvimento do *frontend*, será utilizada a tecnologia React.js desenvolvida pela Facebook [12]. React.js [5] é uma biblioteca de JavaScript que constrói páginas web através da técnica *single-page application*.

Estas aplicações, consistem em apenas uma página, para tal efeito, todo o código necessário (HTML, CSS, JavaScript e outros) é obtido através de um único carregamento de página ou então os recursos vão sendo apropriadamente carregados dinamicamente e adicionados à página conforme necessário, geralmente como resposta a ações do utilizador.

Tudo o que é visível em React.js é considerado um componente: um botão, uma tabela, ou algo mais complexo como um formulário, mesmo que não se utilize esta

característica todo o código fará parte de um componente “*App*” (que é a “*root*” da aplicação).

```
1 import React, { Component } from 'react'
2
3 class App extends Component {
4   constructor(props) {
5     super(props)
6
7     this.state = {
8       list: [1, 2, 3],
9     }
10  }
11
12  render() {
13    return (
14      <div>
15        <ul>
16          {this.state.list.map((item) => (
17            <li key={item}>{item}</li>
18          ))}
19        </ul>
20      </div>
21    )
22  }
23 }
24
```

Figura 9 - Exemplo de uma componente React.js [16]

A figura 9 mostra uma implementação de um pequeno componente que apenas mostra uma lista de acordo com o seu estado. Na rotina `render` tem-se o código responsável pela sua apresentação na página. Este componente em si não tem nenhuma lógica implementada, por exemplo, outras rotinas. No entanto, contém um pequeno estado, que como se pode observar é uma lista de tamanho 3.

Esta forma de estruturar o código agiliza o desenvolvimento da aplicação e permite uma maior facilidade em compartimentar código, ficando assim mais organizado. Em caso de alteração é apenas necessário alterar o código de uma componente em vez de várias páginas HTML.

Uma outra vantagem do React.js é o facto de trabalhar com virtual DOM, de forma curta o que isto faz é em vez de renderizar a página toda quando um elemento da mesma se altera. O React.js renderiza apenas os componentes que se modificam, isto para aplicações mais complexas oferece rapidez, onde um programador disciplinado pode tirar muito partido desta característica e o ganho em termos de desempenho é notório. É importante referir que caso seja mal implementado corre-se o risco de renderizar página toda perdendo assim toda esta eficácia.

Existem dois tipos principais de componentes em React.js:

1. **Functional Components:**

Um componente funcional, no fundo é uma função de JavaScript, este tipo de componentes é utilizado quando sabemos que o componente não irá interagir com outros componentes.

2. **Class Components:**

Um componente de classe é muito semelhante ao anterior, mas é utilizado quando se quer que os componentes interajam uns com os outros, partilhando estados, total ou parcialmente.

```
this.state = {  
  list: [1, 2, 3],  
}
```

Figura 10 - Focando o estado do exemplo. [16]

O estado de um componente é o conjunto de atributos que se define para o componente, mas a forma de manusear este estado pode ficar complexa pois, os dados em React.js apenas fluem em uma direção, do componente pai para o componente filho como mostra a seguinte figura:

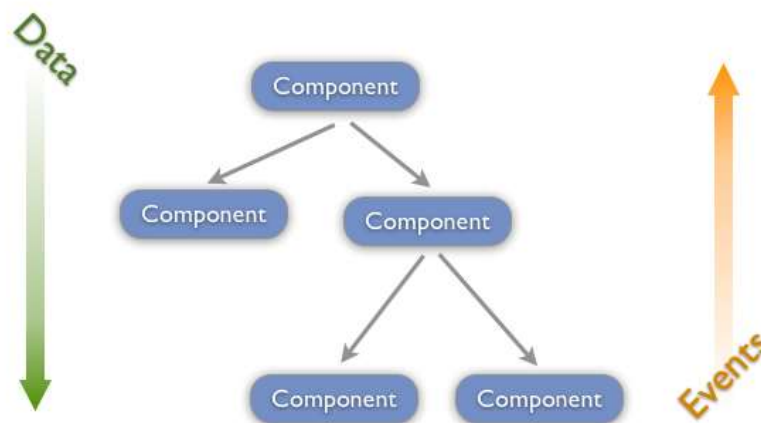


Figura 11 - Fluxo dos dados em React.js. [17]

A informação que flui pode causar mudança nestes “filhos” e estas mudanças podem até mesmo ser observadas pelos “pais”, mas um componente filho não consegue enviar informação para outros componentes que não “filhos” do mesmo, isto torna mais complexo a partilha de informação que é geral à aplicação.

Para a partilha de informação de um componente num nível diferente da outra, seria necessário observar mudanças desde um nível muito atrás da árvore de componentes, observando a figura.

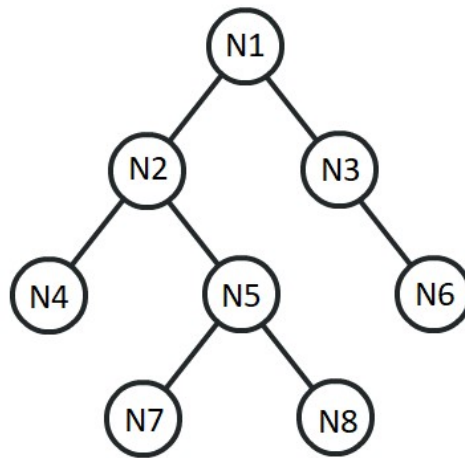


Figura 12 - Estrutura em árvore dos componentes [17]

Para que uma mudança no componente N6 influencie o N7 ter-se-ia de implementar no N1 alguma forma de observar todas as mudanças abaixo dele, neste caso N3 e em N3 observar N6 para tais mudanças, isto tudo para que N1 seja notificado e posteriormente enviar a informação para N2 e por aí fora até chegar ao N7. Para facilitar esta partilha de informação e o manuseamento do estado geral desta *frontend* ir-se-á utilizar a ferramenta Redux.

4.2.3 Redux

O Redux surge no sentido de centralizar os estados partilhados entre os diversos componentes, permitindo assim que estes sejam renderizados sempre que este estado se altere.

Para tal, Redux utiliza lojas (*stores*), geralmente apenas uma, e redutores (*reducers*), onde a *store* guarda o estado da aplicação e os *reducers* efetuam operações nele. Desta forma simplifica-se o processo de partilha de dados entre componentes e torna assim desnecessário o processo anteriormente mencionado.

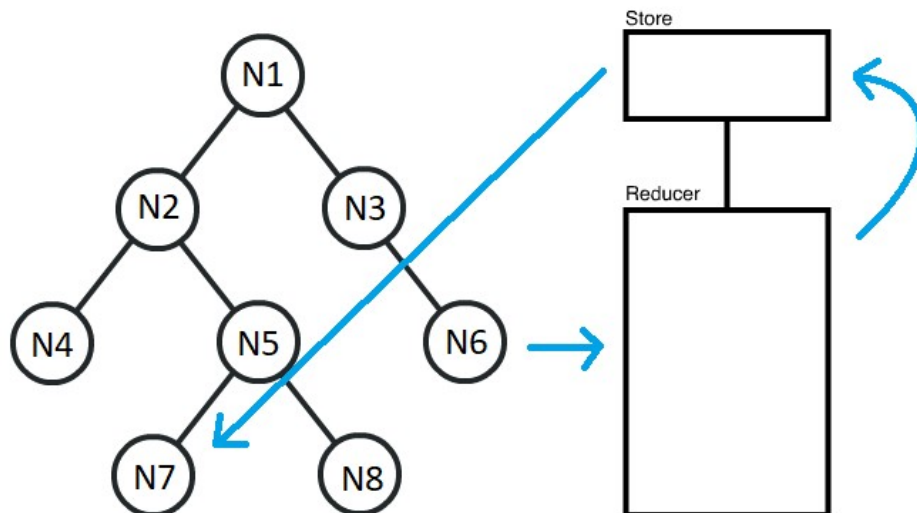


Figura 13 - Estrutura com a adição da *store* do Redux. [17]

Recorrendo novamente ao exemplo anterior em que a mudança que N6 efetua afeta o N7, N6 utilizaria um *reducer* que manipula o estado necessário em N7, este *reducer* efetua as alterações necessárias na *store*, e N7 como está subscrito a este estado, sempre que este estado será alterado a componente também será alterada de acordo com a nova informação.

4.2.4 Axios

Uma vez que o *backend* disponibiliza uma API REST para comunicação, o *frontend* irá utilizar a biblioteca Axios [18] para tal efeito. Axios é uma biblioteca que permite realizar operações HTTP elementares como POST, GET, entre outras.

4.2.5 Node Package Manager (NPM)

Uma vez que o projeto utiliza Node.js, foi utilizado o NPM como gestor de pacotes da aplicação, ou seja, o NPM instala desinstala ou atualiza todos os pacotes necessários para executar a aplicação.

4.2.6 Secure Socket Shell (SSH)

SSH é um protocolo de rede criptográfico para a operação de serviços de rede de forma segura sobre redes inseguras. O exemplo mais comum é acesso remoto de um utilizador num computador. Este também permite, após o *login*, a execução de comandos remotamente, isto permite aos utilizadores aceder e gerenciar servidores via Internet, portanto um excelente protocolo para comunicar com as virtualizações.

4.3 Arquitetura do *frontend*

Todas estas tecnologias culminam numa arquitetura como mostra a seguinte figura:

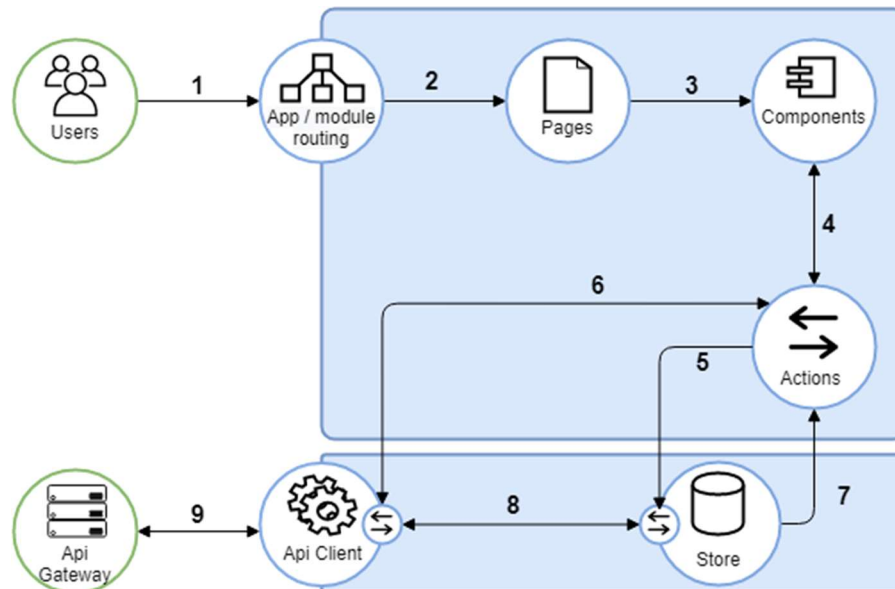


Figura 14 - Arquitetura do *frontend*.

1. O Utilizador interage com o módulo principal App.
2. O módulo principal App “direciona” o utilizador para as páginas corretas, apresentando-as no seu “corpo”.
3. As páginas são compostas por diversas componentes, logo estas terão de ir buscar a informação de cada uma.
4. As componentes que partilhem estado terão de ter acesso a *actions* para modificar estados e aceder aos mesmos.
5. As *actions* modificam o conteúdo da *store*.
6. As *actions* podem interagir diretamente com a *API client* do *frontend* para *fetch* de informação.
7. A *Store* devolve informação às *actions* (para consequentemente chegar aos componentes como em 4)
8. A *Store* pode também interagir com o *API client* do *frontend* para *fetch* de informação.
9. O *API client* do *frontend* Comunica com o *API Gateway* do *backend*.

Em termos práticos é esperado uma estrutura de ficheiros semelhante a seguinte:

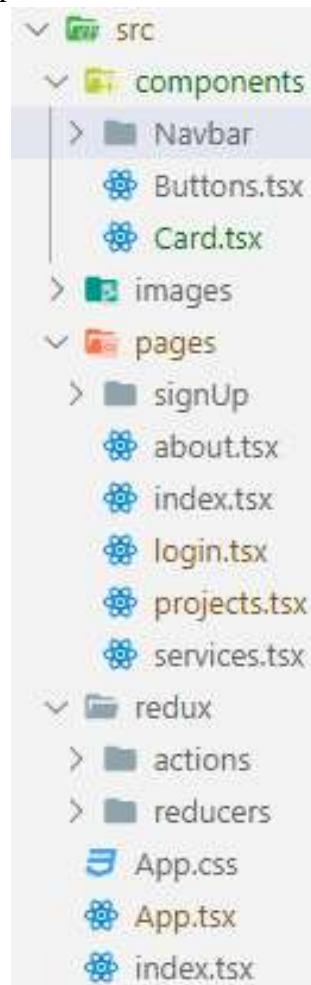


Figura 15 - Estrutura de ficheiros.

Como é possível visualizar na figura 15 tem-se os diversos componentes da *frontend* devidamente separados, os componentes, ou peças num sítio, as páginas também individualizadas, onde estas páginas no seu corpo delas irão “chamar” os diversos componentes necessários. O Redux também está individualizado, separando as ações dos *reducers*.

Desta forma torna-se mais fácil escalar esta aplicação se assim for necessário ao longo do tempo.

4.4 Implementação

Para o desenvolvimento do *frontend* foi implementado o *backend* (<https://boxit-edu.duckdns.org/api-docs/>), e o próprio *frontend* de momento já se encontra também disponível no mesmo servidor através do link: <https://boxit-edu.duckdns.org/>.

Na fase inicial (solução que foi apresentada na defesa), onde apenas estavam implementados o registo de novos utilizadores e o *login* dos mesmos. Nesta mesma fase foi necessário resolver um problema de *cross-origin resource sharing* (CORS):

<input type="checkbox"/> login	CORS err...	xhr	xhr.js:177	0 B	267 ms			
<input type="checkbox"/> login	204	preflight	Preflight	0 B	264 ms			

Figura 16 - Erro de CORS no browser

Este problema surgiu, pois o *backend* não aceitava pedidos provindos de outras fontes e consequentemente não respondia a esses mesmos pedidos:

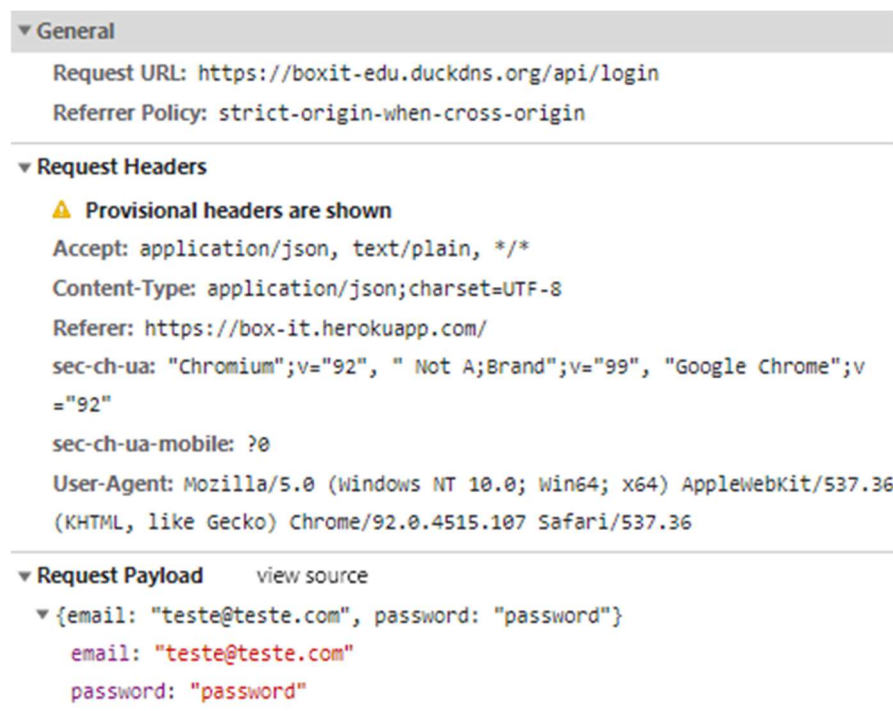


Figura 17 - Não obtenção de resposta do *backend*

Foi também necessário alterar a resposta que o *endpoint* do *login* fornecia para o *frontend* receber também o nome do utilizador (para além dos *tokens* de autenticação) de forma a poder mostrar o mesmo na página inicial como objetivo a demo que foi feita na primeira defesa do TFC.

Como nesta fase inicial o nível de aprendizagem era imenso, muitas funcionalidades não estavam implementadas da melhor forma, pois o processo de aprendizagem deu-se diretamente dos componentes que necessitavam da funcionalidade, o que originaria código de difícil manutenção e compreensão, como por exemplo a lógica do registo e o *login* estavam implementadas nos próprios componentes utilizando o Redux.

Nesta altura todo o projeto estava implementado utilizando o superconjunto de JavaScript, o TypeScript. Esta linguagem foi escolhida com o objetivo de poupar tempo reduzindo a possibilidade de existência de bugs dada a sua extrema tipificação. Esta decisão teve de ser alterada, pois estava a ter o efeito contrário, estava-se a perder demasiado tempo em pesquisa sobre a linguagem e os melhores métodos de implementar certas funcionalidades, tendo em conta que ao mesmo tempo se pesquisava sobre outras tecnologias como é o caso do Redux.

Devido a estas duas últimas razões houve necessidade de tomar uma decisão. Decisão esta que foi refazer o *frontend* de início e utilizar apenas JavaScript. Não se perdeu muito tempo pois o projeto também não ia muito avançado e muito código era reutilizável devido as semelhanças entre TypeScript e JavaScript. Isto permitiu separar melhor a lógica dos componentes que acelerou muito o processo de desenvolvimento do que já estava implementado e das novas funcionalidades.

Quanto à arquitetura da plataforma em si, esta tem vindo a ser cumprida uma vez que se tem o *frontend* em execução independente de todos os outros componentes, deu-se apenas alguma alteração na organização interna de ficheiros devido à reestruturação que existiu.

5 Benchmarking

No caso deste trabalho, em que se está a criar uma *frontend* para uma plataforma específica e desconhecida (não existe muita informação sobre a mesma), torna-se complicado pesquisar o que seria melhor ou pior para o desenvolvimento desta *frontend*.

O que se sabe é que se terá de utilizar tecnologias para o desenvolvimento de *frontend*, tais como HTML, CSS, JavaScript, para facilitar o uso destas ferramentas ir-se-á utilizar uma *framework*, como referido anteriormente vai-se utilizar React, mas existem outras tais como Angular ou Vue. [6]

Virtual dom.

O DOM virtual (VDOM) [7] [8] é um conceito de programação onde uma representação ideal, ou “virtual” de uma IU é mantida na memória e sincronizada com o DOM “real” por uma biblioteca como o ReactDOM. Este processo é chamado de “*reconciliation*” (reconciliação). Com este processo consegue-se melhorias de *performance* pois quando um elemento é modificado o React.js apenas modifica no DOM “real” os objetos que foram modificados, em vez de atualizar o DOM “real” todo como é feito sem a utilização desta tecnologia.

Não é apenas o React.js que utiliza esta tecnologia das *frameworks*, anteriormente faladas, o Vue também a utiliza.

Server-side rendering

Server-Side Rendering [9] ou SSR é o processo de pegar em todos os ficheiros JavaScript, CSS de um site que, geralmente é carregado para o *browser (client-side)*, e fazer render dos mesmos como estáticos do lado do servidor.

Com esta operação é possível obter um site com tempo de carregamento mais reduzido e isto é importante se se tiver o objetivo de indexar as páginas em motores de busca como o Google.

De estes dois pontos gostaria de dar ênfase ao primeiro (Virtual DOM) pois em termos de *performance* será o que terá mais impacto, pois o de componentes reutilizáveis todos têm e terá mais impacto na fase de desenvolvimento da aplicação, e o de *Server-Side Rendering*, não será implementado, mas vale sempre a pena saber que esta disponível no caso de se querer alargar a rede de utilizadores.

Em termos de desempenho vai-se agora comparar estas 3 *frameworks* com a ajuda de um pequeno teste realizado pela LogRocket[6], onde se tem as seguintes tabelas das quais se pode retirar algumas conclusões:

Startup metrics (lighthouse with mobile simulation)

Name	angular-v8.0.1-keyed	react-v16.8.6-keyed	vue-v2.6.2-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	2,703.7 ± 0.4 (1.24)	2,353.2 ± 0.5 (1.08)	2,177.5 ± 0.3 (1.00)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	52.0 ± 70.5 (3.25)	16.0 ± 0.0 (1.00)	16.0 ± 0.0 (1.00)
total kilobyte weight network transfer cost (post-compression) of all the resources loaded into the page.	295.5 ± 0.0 (1.40)	260.7 ± 0.0 (1.24)	211.0 ± 0.0 (1.00)
slowdown geometric mean	1.78	1.10	1.00

Figura 18 - Resultados do benchmark de tempo de inicialização

Como se pode verificar, Vue.js é o vencedor absoluto neste caso. O tamanho notavelmente pequeno desta estrutura ajuda muito quando se trata de tempo de inicialização.

Vemos resultados muito semelhantes para React neste caso, pode-se ver tem bons resultados aqui. Angular, no entanto, sofre com sua estrutura mais pesada, como se verifica claramente. Este peso pode ser justificado pois o Angular tem muitas funcionalidades que o restante não tem:

Feature	Angular	React	Vue
UI / DOM Manipulation	✓	✓	✓
State Management	✓	✓	✓
Routing	✓	✗	✓
Form Validation & Handling	✓	✗	✗
Http Client	✓	✗	✗

Figura 19 - Comparação de funcionalidades entre frameworks

Em suma, o React.js tem resultados bons, ou intermédios comparativamente com os outros dois competidores, e dada a complexidade envolvida com o website em si, julgo não ser uma má escolha como *framework* deste projeto.

6 Método e Planeamento

6.1 Calendário

6.1.1 Descrição de certos pontos:

Modelação primaria da Web App:

Com esta fase pretende-se fazer um *mockup* inicial básico do que será a aplicação, e depois desenvolver o esqueleto da aplicação para depois se implementar o sistema de *login*.

Modelação secundaria da Web App:

Com esta fase pretende-se continuar o processo primário, tendo em conta a criação de projetos e suas componentes.

Criar projetos / importar projetos:

Com esta fase pretende-se interligar a *frontend* com a *backend* na criação de projetos e as suas componentes.

Operações com projetos:

Com esta fase pretende-se implementar as funções de *start*, *stop* e *reset* dos Docker containers na *backend* a partir do *frontend*.

Tabela 1 - Calendário Proposto

Período de Desenvolvimento	Objetivo
23 de novembro a 29 de novembro	Análise de requisitos
30 de novembro a 13 de dezembro	Modelação primaria da Web App
14 de dezembro a 27 de dezembro	Implementar Sistema de autenticação
28 de dezembro a 10 de janeiro	Modelação secundaria da Web App / Deploy no servidor
11 de janeiro a 17 de janeiro	Criar projetos / importar projetos (formato JAR)
18 de janeiro a 24 de janeiro	Default dockerfiles
25 de janeiro a 7 de fevereiro	Operações com projetos
8 de fevereiro a 14 de fevereiro	Acesso via browser SSH
15 de fevereiro a 21 de fevereiro	Escrever relatório intermedio
22 de fevereiro a 21 de março	Correção de bugs
22 de março a 4 de abril	Importar projetos em formato ZIP
5 de abril a 16 de abril	Importar projetos do utilizador de repositórios Git
17 de abril a 23 de abril	Escrever relatório intercalar
17 de abril a 23 de abril	Deixar pronto para testes com utilizadores
24 de abril a 30 de abril	Testes com utilizadores
1 de maio a 7 de maio	Aplicar sugestões de acordo com a opinião dos utilizadores
8 de maio a 14 de maio	Correção de bugs
15 de maio a 28 de maio	Criação de Dockerfiles para <i>frontend</i> e backend
29 de maio a 18 de junho	Criação de mais dockerfiles para outras linguagens de programação (Ex. Kotlin)
19 de junho a 25 de junho	Escrever relatório final

6.1.2 Cumprimento do calendário

Após a criação deste calendário tentou-se cumpri-lo, existem algumas tarefas que levavam duas semanas a serem feitas onde se calhar não seria necessário tanto tempo, pois aparecem sempre imprevistos pelo caminho e isto deixava alguma flexibilidade caso aparecesse um.

Foi criada uma lista de tarefas ao longo do desenvolvimento. Esta lista de tarefas era atualizada sempre que uma dessas tarefas era concluída ou caso surgisse algum imprevisto como um *bug*.

Apesar deste planeamento, o calendário não tem sido cumprido na totalidade. Devido ao número de tecnologias utilizados neste projeto e também à falta de funcionalidades no *backend*, o tempo de aprendizagem tem sido muito superior ao previsto, além de toda a aprendizagem necessária com as tecnologias também exige um esforço para saber o funcionamento do *backend* para se poder efetuar todas as modificações que lá foram feitas.

7 Resultados

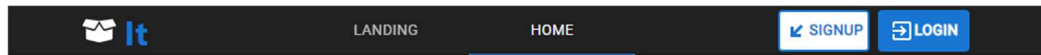
7.1 Interface

A interface encontra-se num estado que permite ao utilizador realizar algumas operações essenciais para utilizar a plataforma.

Desta forma o utilizador consegue fazer *login/logout*, criar projetos e as suas componentes, dizer quantas instâncias de cada uma ele deseja, e realizar algumas operações as instâncias tais como parar e iniciar cada uma individualmente assim como todas as do projeto se esse for o seu objetivo.

Dado que a plataforma demorou tempo a chegar a este estado, e também ainda não é possível o acesso aos *containers* do Docker via browser, não foram executados ainda testes com os utilizadores.

Passo a apresentar algumas figuras do *frontend*:



Home

Figura 20 - BoxIt home page

O home screen e Langing page, apenas existem caso no futuro se queira adicionar alguma informação sobre o BoxIt.

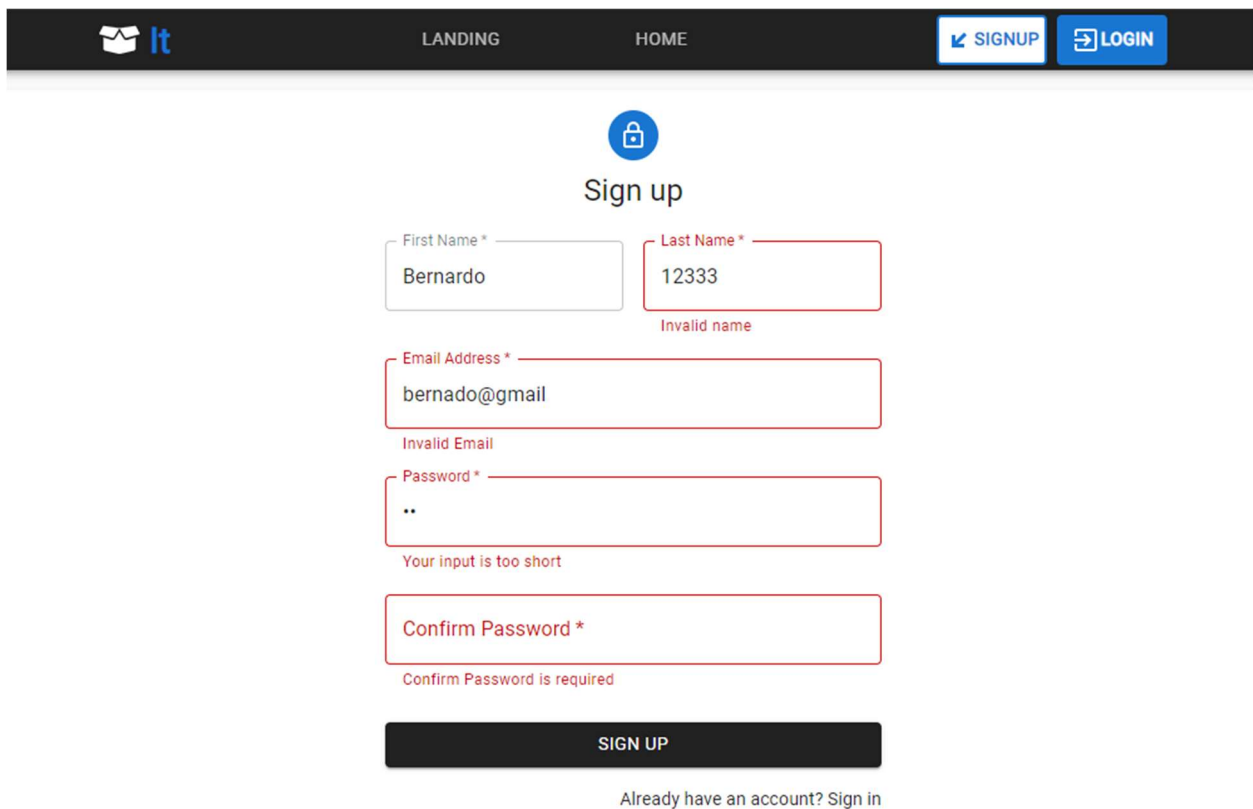
A sign-up form with a dark blue header bar. The header contains a logo, 'LANDING' and 'HOME' links, and 'SIGNUP' and 'LOGIN' buttons. The main content area has a blue lock icon and the text 'Sign up'. Below this are four input fields: 'First Name *' with the value 'Bernardo', 'Last Name *' with the value '12333' and a red error message 'Invalid name', 'Email Address *' with the value 'bernado@gmail' and a red error message 'Invalid Email', and 'Password *' with the value '..' and a red error message 'Your input is too short'. Below these is a 'Confirm Password *' field with a red error message 'Confirm Password is required'. At the bottom is a black 'SIGN UP' button and a link 'Already have an account? Sign in'.

Figura 21- BoxIt signup page

Nesta página o utilizador pode realizar o registo na plataforma. Todos os campos de texto executam validação de texto conforme o que se quer obter (*password, e-mail...*) como demonstra a figura.

You have register into Boxit

Please press the following button to activate your account:

Verify Account

Figura 22 - Email de confirmação

Após o registo, o utilizador receberá o email que a figura 22 mostra, onde terá de clicar no botão para verificar a conta e depois será redirecionado para a página que a figura Figura 28] mostra.

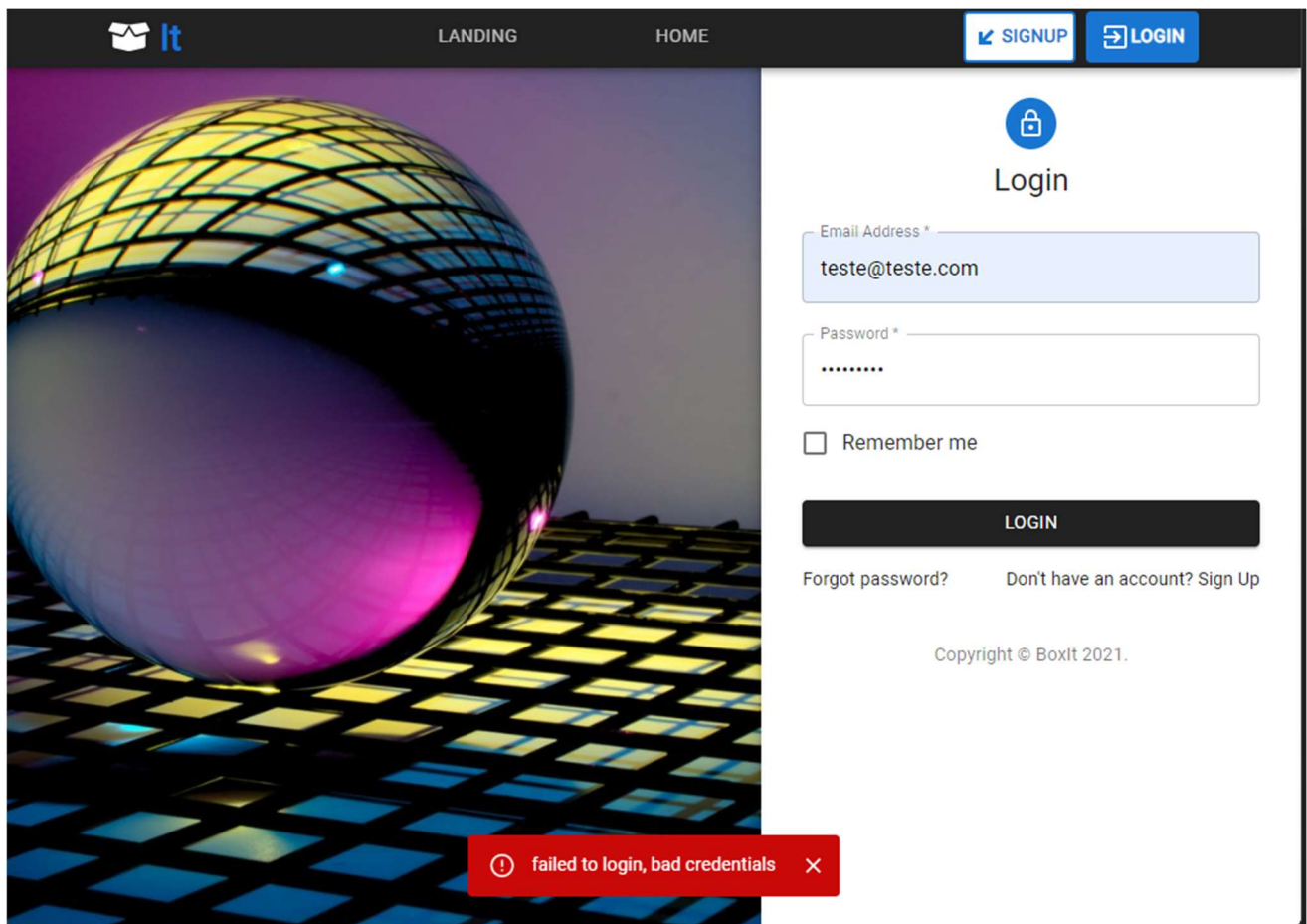


Figura 23- BoxIt login page

A *Login Page* permite o utilizador autenticar-se para entrar na plataforma, estes campos fazem as mesmas validações que a página anterior e ainda tem uma adição que é a apresentação da mensagem de erro (através de um *toast/snackbar*) que provém diretamente do *backend*.

Após o *login* o utilizador ira-se deparar com uma página de boas-vindas que mostra o nome do mesmo.

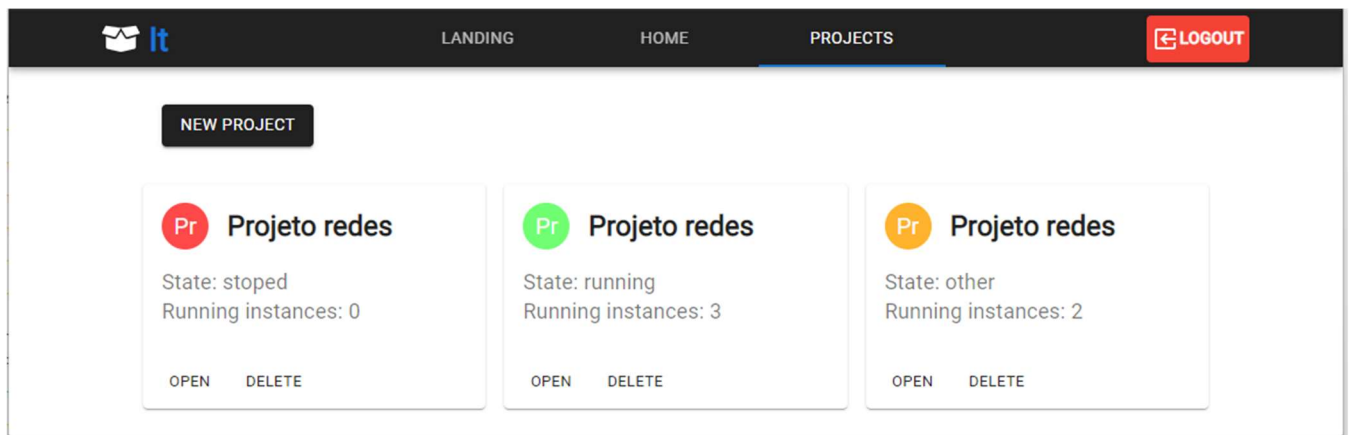



Figura 24 - BoxIt projects page

Uma vez autenticado o utilizador pode navegar até a página de projetos que apresenta ao utilizador todos os seus projetos, assim como alguma informação como por exemplo o estado do projeto e o número de instâncias que estão ligadas.

Existe uma associação de cores ao estado do projeto, se estiver parado o avatar do projeto está a vermelho, se algumas estiverem paradas esta a laranja e se estiverem todas em execução estará a verde.

As letras que se encontram no avatar são as duas primeiras letras do nome do projeto.

LANDINGHOMEPROJECTSLOGOUT

Projeto redes

Uuid: e97ad9a9-a126-4216-8154-338e844a7a83

state: other

Created At: 2021-07-21T17:23:32.000Z

Last Updated at: 2021-07-21T17:23:32.000Z

	Name	State	IP	Id
^	client0	running	172.23.0.3	e2a0e544e8a6e8b7ac2b883920384a0f991bb9320701ac535a2baa000403b992
Operations				
<div>▶ START INSTANCE</div> <div>■ STOP INSTANCE</div>				
∨	client2	running	172.23.0.5	abfc119b3d3f07b385f1ac1565d8db4b527207aad0af01fc4bd4d1dd3741c80d
^	client1	exited		600592ae649f8535e93b6661fc4f5edb8f37dec1e50a30837e67dfc278dfde27
Operations				
<div>▶ START INSTANCE</div> <div>■ STOP INSTANCE</div>				
∨	server0	exited		0bcd1ceb20dd1271d9827bf0413399508de9049f675368bc136e07f3978f81a2

↓ DOWNLOAD PROJECT

DELETE PROJECT


■ STOP PROJECT

▶ START PROJECT

Figura 25 - BoxIt project page

Na página de um projeto o utilizador pode aceder às informações mais importantes do projeto e suas instâncias assim como o estado de cada uma, é aqui que o utilizador pode iniciar e parar instâncias individualmente, note-se que se uma instância já se encontrar em execução, o botão de *start* encontra-se *disabled* assim como o do *stop* caso a instância esteja parada.

Aqui também é possível executar o *download* e o *delete* de um projeto.

LANDINGHOMEPROJECTSLOGOUT

New Project

Project Title *

Trabalho de Redes

Drag 'n' drop the Project zip, or click to select it

File: Projeto redes.zip - 432612 bytes

Instance Name *	File Name *	Arguments *
server	server.jar	
Depends On: ▾	Number of instances: *	
	1	
Instance Name *	File Name *	Arguments *
client	client.jar	
Depends On:	Number of instances: *	
server ▾	10	

+ ADD NEW INSTANCEDOWN SUBMIT

Figura 26 - BoxIt new project page

A página de *New Project* permite ao utilizador criar o seu projeto, dando-lhe um título, selecionando o zip ou fazendo *drag and drop* na zona definida.

Inicialmente terá apenas um formulário que permitirá a inserção de uma componente, mas poderá adicionar quantas desejar através do botão *Add New Instance*.

Como aqui poderá haver necessidade de adicionar muitas componentes os botões encontram-se de baixo do formulário, para que o utilizador não tenha que fazer *scroll* até ao topo da página.

7.2 Deployment

No início deste projeto foi-me disponibilizado um servidor já com HTTPS a funcionar com um certificado letsencrypt [20] o que é bastante bom, pois desta forma toda a comunicação entre os clientes e servidor é cifrada protegendo assim a informação. Neste servidor foi de imediato implementado através de docker *container* a solução do *backend* do trabalho realizado no ano letivo passado (<https://boxit-edu.duckdns.org/api-docs/>) e assim este é o ambiente de produção do projeto, que de momento também já contem o ambiente de produção do *frontend* (<https://boxit-edu.duckdns.org/>).

Para ambiente de testes e desenvolvimento, é utilizado a minha própria máquina que corre o *backend* numa distribuição de Linux (Ubuntu) [21] através de uma tecnologia relativamente recente, *Windows Subsystem for Linux* (WSL) [22], que me permite executar todo o projeto utilizado o Windows 10 e ter as regalias do Linux e o *frontend* corre dentro do próprio Windows. Também se encontra disponível o *frontend* (através de CI/CD) na *platform as a service* (PaaS) Heroku [19] que foi o primeiro sítio de *deployment* deste *frontend* (<https://box-it.herokuapp.com/>), este pode estar desatualizado, dado que já não é o URL principal.

Com este processo já foram captados alguns problemas que não estavam indicados na documentação do *backend* tal como o problema *cross-origin resource sharing* (cors) e também uma diferença no método de comunicação para o *backend*, normalmente em todos os pedidos via API a informação é enviada através de um JSON, mas o serviço que cria os projetos a informação tem de ser enviado através de data-form, não existia forma de efetuar operações cruciais tais como iniciar ou parar instâncias.

O *backend* foi modificado também nas páginas que ele fornece quando um utilizador se regista:

```
{"type": "success", "message": "Account activated successfully"}
```

Figura 27 - Página de confirmar conta (antes)

Account activated successfully
You may now [Login](#) in BoxIt

Figura 28 - Página de confirmar conta (depois)

```
{"type":"error","message":"account already activated","error":{"status":400,"errorType":"bad request"}}
```

Figura 29 - Página conta já ativada (antes)

Account already activated

Figura 30 – Página conta já ativada (depois)

As novas páginas são muito simples pois não se tem acesso a todas as ferramentas que o React oferece, como estas páginas são servidas pelo *backend* esta resposta é dada fazendo o *hardcode* de uma página no próprio código.

Foi gravado um pequeno vídeo para facilitar a visualização do funcionamento da plataforma:

<https://youtu.be/hJ-6XvmjQKg>

8 Conclusão e trabalhos futuros

Em suma, o desenvolvimento deste projeto foi muito mais desafiante do que qualquer outro projeto das outras cadeiras do curso. Necessitou de um planeamento mais detalhado, tomadas de decisões críticas, exigiu uma grande capacidade de adaptação dos conceitos adquiridos ao longo do curso, que culminaram numa maior facilidade de aprender novas tecnologias como o Docker e React.JS, assim como aprofundar outros conceitos já obtidos tanto na área de redes como Linux obtidos no processo de *deployment*.

Estando o *frontend* feito e já capaz de resolver a maior parte dos problemas a que se propôs, este ainda não se encontra no estado desejado, para tal ainda é necessário completar os restantes requisitos.

A não total finalização de todos os requisitos deveu-se ao facto de o *backend* não estar no estado em que se esperava quando se iniciou este projeto, houve diversas áreas que necessitaram de correções, muitas delas, áreas que se dava por garantido que funcionava tais como a utilização de *Docker-compose* para iniciar o *backend*, o manuseamento individual ou total de instâncias, *refresh* de *tokens*, obtenção dos dados das instancias do Docker.

O desafio a que me propus de utilizar uma framework que nunca tinha utilizado foi cumprido, após este projeto sinto-me mais confiante a utilizar React.js.

A circunstância de ter aceitado um trabalho que já tinha começado, pessoalmente diria que é uma mais-valia, pois foi necessário lidar com código de um colega, e ter de o estudar para saber como efetuar todas as alterações que foram necessárias, e sei que no mercado de trabalho lida-se com esta situação todos os dias.

O *frontend* ainda pode ser melhorado, dando indicações aos utilizadores de quando algumas tarefas são concluídas (criação de projetos, *start* e *stop* de instâncias), fornecer o acesso as instâncias via browser, criar o *dashboard* de administrador seria uma boa oportunidade utilizar esta plataforma num âmbito educacional, quando o *frontend* se encontrar finalizado.

Bibliografia

- [1] “Arquiteturas de Aplicação em Redes,” [Online]. Available: <https://www.programacaoprogressiva.net/2019/02/Arquiteturas-de-Aplicacao-em-Redes-Cliente-Servidor-P2P.html> [Acedido em 20 11 2020].
- [2] “What is a Container?” [Online]. Available: <https://www.docker.com/resources/what-container> [Acedido em 20/11/2020].
- [3] “Uma introdução a TCP, UDP e Sockets” [Online]. Available: <https://www.treinaweb.com.br/blog/uma-introducao-a-tcp-udp-e-sockets/>. [Acedido em 15 Julho 2020].
- [4] “[JWT] [Online]. Available: <https://jwt.io/> [Acedido em 20/11/2020].
- [5] “React (JavaScript)” [Online]. Available: <https://reactjs.org/>. [Acedido em 15 Julho 2020].
- [6] “Angular vs. React vs. Vue: A performance comparison” [Online]. Available: <https://blog.logrocket.com/angular-vs-react-vs-vue-a-performance-comparison/> [Acedido em 27 Julho 2020].
- [7] “Virtual DOM and Internals” [Online]. Available: <https://reactjs.org/docs/faq-internals.html> [Acedido em 27 Julho 2020].
- [8] “React: The Virtual DOM” [Online]. Available: <https://www.codecademy.com/articles/react-virtual-dom> [Acedido em 27 Julho 2020].
- [9] “What is server-side rendering?” [Online]. Available: <https://www.educative.io/edpresso/what-is-server-side-rendering> [Acedido em 27 Julho 2020].
- [10] ULHT – Redes de Computadores: Projeto Prático “Ambiente Cliente-Servidor”.
- [11] Francisco Silva “Boxit - Plataforma Geradora de Redes de Computadores Virtuais”.
- [12] React.js github repository from [Online]. Available: <https://github.com/facebook/react> [Acedido em 21 de Janeiro de 2021].
- [13] “Internet Message Format” [Online]. Available: <https://tools.ietf.org/html/rfc2822> [Acedido em 21 de Janeiro de 2021].
- [14] “Types of React Components you should know” [Online]. Available: <https://medium.com/weslony-team/types-of-react-components-you-should-know-251cceacd8ac> [Acedido em 21 de Janeiro de 2021].
- [15] Postman [Online]. Available: <https://www.postman.com/> [Acedido em 15 de Janeiro de 2021].
- [16] “Everything you need to know about setState()” [Online]. Available: <https://medium.com/hootsuite-engineering/everything-you-need-to-know-about-setstate-8233a7042677> [Acedido em 21 de Janeiro de 2021].
- [17] “One Way Data Flow” [Online]. Available: <https://tkssharna.gitbook.io/react-training/day-01/react-js-3-principles/one-way-data-flow> [Acedido em 21 de Janeiro de 2021].
- [18] Axios [Online]. Available: <https://www.npmjs.com/package/axios> [Acedido em 21 de Janeiro de 2021].

- [19] heroku [Online]. Available: <https://www.heroku.com/> [Acedido em 22 de Abril de 2021].
- [20] letsencrypt [Online]. Available: <https://letsencrypt.org/> [Acedido em 22 de Abril de 2021].
- [21] ubuntu [Online]. Available: <https://ubuntu.com/> [Acedido em 22 de Abril de 2021].
- [22] Windows Subsystem for Linux [Online]. Available: <https://docs.microsoft.com/en-us/windows/wsl/> [Acedido em 22 de Abril de 2021].

Anexos

Anexo 1 – Levantamento de requisitos

Requisitos Funcionais:

RF1 - Registo do utilizador

Pré-Condição:

O utilizador deve ter uma conta de email e acesso à mesma, estar no ecrã de registo (*Sign up*).

Requisito:

No ecrã de registo o utilizador deve poder inserir o seu email, *password*, confirmar *password*, primeiro e último nome de forma a registar-se na plataforma.

O *frontend* verifica se o e-mail e nomes inseridos são válidos e se as *passwords* coincidem e em caso de erro deverá ser visualizada uma mensagem que indica o mesmo.

Estes campos serão validados da forma especificada no RF10.

Em caso de erro do *backend* a mensagem de erro fornecida pelo mesmo deverá ser apresentada no ecrã notificando assim o utilizador do sucedido. (Ex. email já existente)

Pós-Condição:

O *frontend* verificou corretamente as informações, notificou bem os erros (caso existam) e passou a informação ao *backend* de acordo com a documentação da mesma.

Em caso de erro do *backend* apresentou o mesmo no ecrã.

Estado:

Implementado

RF2 – acesso via autenticação

Pré-Condição:

O sistema deverá estar configurado no modo multiutilizador, o utilizador deve estar registado para se poder autenticar e estar no ecrã de *login*.

Requisito:

O *frontend* deverá conseguir distinguir se o *backend* se encontra em modo de multiutilizador ou de utilizador único.

No ecrã de *login* o utilizador deve poder inserir o seu email e *password* para efetuar o *login* na plataforma. Estes campos de texto serão validados conforme especificado no RF10.

Pós-Condição:

O *frontend* envia as credenciais inseridas ao *backend*, caso a resposta seja que o utilizador está autenticado, o *frontend* guarda o Token JWT [4] que recebeu para os

próximos pedidos. Caso a resposta seja que não foi possível autenticar o utilizador deverá ser apresentada a mensagem de erro enviada pelo *backend*.

Estado:

Implementado

RF3 - acesso sem autenticação (convidado)

Pré-Condição:

O sistema deverá estar configurado no modo "*single user*".

Requisito:

O *frontend* deverá conseguir distinguir se o *backend* se encontra em modo de multiutilizador ou de utilizador único.

Pós-Condição:

Quando carrega a página inicial da aplicação esta deverá ser apresentada conforme o estado de execução do *backend*.

Por exemplo, se estiver em modo de utilizador único, não deverá aparecer opções para o registo e *login*.

Estado:

Não implementado

RF4 – Recuperar *password*

Pré-Condição:

O utilizador deve estar registado para poder recuperar a *password* e estar na página de recuperar *password*.

Requisito:

No ecrã de recuperar *password* o *frontend* deve apresentar um campo de texto que permita ao utilizador inserir o email da conta da qual o utilizador pretende recuperar a *password*, este campo de texto é validado através do RF10.

Se a conta existir, o sistema notifica o utilizador com os passos, caso não exista apresenta uma mensagem de erro a indicar o sucedido.

Pós-Condição:

O *frontend* envia essa informação ao *backend* no formato indicado pela documentação do mesmo e notifica o utilizador através de uma mensagem ou um novo ecrã se a operação foi bem-sucedida ou não.

Estado:

Não implementado

RF5 - Criar Projetos

Pré-Condição:

O utilizador deve estar autenticado ou aceder via convidado, abrir ou estar na página dos projetos e carregar no botão para criar um novo projeto.

Requisito:

O sistema deve permitir ao utilizador criar projetos, onde cada projeto tem um nome, e as suas diversas componentes (programas para serem executados) que são adicionadas via *upload* de ficheiros em formato zip (caso projeto completo) ou inicialmente JAR (idealmente aceitar mais extensões para outras linguagens de programação).

Neste passo apenas é obrigatório o projeto ter um nome e este requisito está interligado com o RF7.

Pós-Condição:

O sistema cria um projeto e guarda todas as informações relativas ao projeto (nome, componentes, dono).

Estado:

Implementado

RF6 - Criar componentes do projeto

Pré-Condição:

O utilizador deve estar autenticado ou aceder via convidado, ter um projeto já criado e ter a componente pronta a ser enviada junto o devido *dockerfile* relativo à mesma componente.

Requisito:

O *frontend* deve permitir ao utilizador importar os componentes do projeto (Ex: JAR) através de *upload* de ficheiros presentes no computador.

O sistema deve permitir ao utilizador importar dockerfiles relativos ao componente ou então definir um *dockerfile* a utilizar disponibilizado pela plataforma, este *dockerfile* estará interligado com uma componente.

Estas importações são feitas via *upload* de ficheiros (também mencionado no RF6).

Pós-Condição:

O *frontend* enviou os componentes do projeto do utilizador para o *backend*.

Estado:

Parcialmente implementado

RF7 - Administrar contas

Pré-Condição:

O administrador deve estar autenticado e encontrar-se no ecrã de gestão de contas

Requisito:

O ecrã de gestão de contas deve permitir ao administrador validar ou bloquear contas apresentando a lista de utilizadores e fornecendo formas de interação como botões para administrar tais contas.

Pós-Condição:

As operações (validar bloquear utilizador) devem ser registadas corretamente

Estado:

Não implementado

RF8 - CRUD (*Create, Read, Update e Delete*) de projetos e qualquer tipo de componente dos mesmos (ex. JAR, zip).

RF8.1 - Criar

Pré-Condição:

O utilizador deve estar autenticado ou aceder via convidado.

Requisito:

O sistema deve permitir aos utilizadores criar projetos na página de projetos através de um botão que leva o utilizador para uma nova página onde este poderá adicionar componentes e nomes, e outros.

Pós-Condição:

O sistema fica com os projetos e componentes guardados.

Estado:

Implementado

RF8.2 - Read

Pré-Condição:

O utilizador deve estar autenticado ou aceder via convidado e deve existir um projeto ou componentes (dependendo da situação)

Requisito:

O sistema deve poder mostrar aos utilizadores os projetos existentes do próprio utilizador, assim como os componentes existentes dos projetos.

Pós-Condição:

O sistema mostra aos utilizadores os projetos e componentes existentes do próprio utilizador.

Estado:

Não implementado

RF8.3 - Update

Pré-Condição:

O utilizador deve estar autenticado ou aceder via convidado e deve existir um projeto ou componentes (dependendo da situação).

Requisito:

O utilizador deve a partir do sistema conseguir modificar as componentes do projeto assim como nome e outros elementos do mesmo através de botões para confirmar e áreas de texto assim como novos uploads de ficheiros para substituir componentes já existentes.

Pós-Condição:

O sistema guarda as alterações efetuadas pelo utilizador no projeto.

Estado:

Não implementado

RF8.4 - Delete

Pré-Condição:

O utilizador deve estar autenticado ou aceder via convidado e deve existir um projeto ou componentes (dependendo da situação).

Requisito:

O sistema deve permitir que o utilizador elimine componentes através de botões na página do projeto e também deve permitir eliminar projetos através de um botão na página do projeto.

Pós-Condição:

O sistema elimina toda a informação selecionada pelo utilizador, perdendo assim toda essa informação.

Estado:

Não implementado

RF9_UI1 - Executar múltiplas instâncias de cada componente do projeto

Pré-Condição:

O utilizador deve estar autenticado ou aceder via convidado, ter o projeto pronto a ser executado, e estar na página do projeto.

Requisito:

O sistema deve permitir ao utilizador selecionar o número de instâncias de diversas componentes de um projeto que pretende executar e pode mandar executá-las todas com apenas um botão.

Pós-Condição:

O sistema envia esta informação para o *backend* e este executa o número de instâncias selecionadas pelo utilizador a partir de apenas um click no *frontend*.

Estado:

Não implementado

RF9_UI2 – Trabalhar com apenas certas componentes do projeto.

Pré-Condição:

O utilizador deve estar autenticado ou aceder via convidado, ter o projeto pronto a ser executado e estar na página do projeto em questão.

Requisito:

O utilizador pode iniciar o número de instâncias que desejar apenas de certas componentes de um projeto assim como parar ou reiniciar a partir de um botão referente a esse componente.

Pós-Condição:

O sistema executa as operações que o utilizador definiu a partir do botão.

Estado:

Não implementado

RF10 - Validação de campos de texto

Pré-Condição:

Existe um campo de texto para ser validado, nomeadamente campo de email, nome, password.

Requisito:

O *frontend* deve validar todos os campos de texto de email no registo e *login* da plataforma de acordo com a especificação no RFC2822 [13] (3.4.1. *Addr-spec specification*) e nos restantes verificar a existência de caracteres inválidos tais como ”{[]}=?”.

Após esta verificação em caso de erro deverá ser apresentada uma mensagem a apontar a existência do mesmo.

Pós-Condição:

O sistema verifica corretamente a existência dos erros e apresenta a mensagem de erro caso este exista.

Estado:

Implementado

Requisitos Não Funcionais:

RNF 1 – Acesso às instâncias executadas através do protocolo SSH (*Secure Socket Shell*)

O utilizador deve ter acesso a cada uma das componentes individualmente via browser, através do protocolo SSH.

Estado:

Não implementado

RNF2 - HTTPS

A comunicação com o *backend* deve ser feita via um canal seguro, HTTPS.

Estado:

Implementado

RNF 3 - Docker

Utilizar Docker para facilitar *deploys* da própria *frontend*.

Estado:

Em desenvolvimento

RNF 4 - Comunicação entre *frontend* e *backend*

O *frontend* deve comunicar com o *backend* de acordo com o que está protocolado e documentado pela API em <https://boxit-edu.duckdns.org/api-docs/>

Estado:

Tem-se consultado sempre a documentação para a implementação das funcionalidades.

Anexo 2 – Manual de instruções do Boxit (*Frontend*)

Boxit Frontend

Requirements:

- NodeJs
- NPM (usually installed with NodeJs)
- Docker (if you want to run the frontend inside a container)

Instructions:

Change the API URL on the `/src/index.js` file if needed

```
axios.defaults.baseURL = "http://localhost:5000/api/";
```

1. Development

1.1 Install dependencies

```
npm install
```

1.2 Start the project

```
npm start
```

2. Production/Deploy

2.1. Using Docker containers

2.1.1 run with docker-compose

```
docker-compose up
```

2.2 Native

2.2.1 Install dependencies

```
npm install
```

2.2.2 Install serve

```
npm install -g serve
```

2.2.3 Build the project

```
npm run build
```

2.2.2 Serve the application

```
serve -s build
```

Glossário

TFC	Trabalho Final de Curso
NPM	Node Package Manager
JWT	JSON Web Token
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Máquina Virtual
SSH	Secure Socket Shell
UI	User Interface
REST	Representational State Transfer
API	Application Programming Interface
HTTPS	HyperText Transfer Protocol Secure
HTML	HyperText Markup Language
DOM	Document Object Mode
CSS	Cascading Style Sheets
UI	User Interface
UX	User Experience
URL	Uniform Resource Locator
CI/CD	Continuous Integration / Continuous Delivery
CVS	Version Control System
CORS	Cross-Origin Resource Sharing
UUID	Universally Unique Identifier