



**Universidade do Minho**  
Escola de Engenharia

# Laboratório de Engenharia Informática

## **Mantis**

### *Plataforma de Gestão de Equivalências*

Diogo Araújo

**A78485**

Diogo Nogueira

**A78957**

Miguel Solans

**PG41841**

## **Resumo**

Este trabalho prático é fruto de um desafio de projeto da Unidade Curricular de Laboratório de Engenharia Informática e destina-se essencialmente à explicação prática da criação duma *webapp* para gestão de equivalências do ensino superior, recorrendo a tecnologias de desenvolvimento moderno para a Web.

# Conteúdo

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introdução</b>  | <b>3</b>  |
| 1.1      | Estrutura do relatório . . . . .                               | 3         |
| <b>2</b> | <b>Arquitetura do Sistema</b>                                  | <b>4</b>  |
| <b>3</b> | <b>Backend</b>   | <b>5</b>  |
| 3.1      | Tecnologias Utilizadas . . . . .                               | 5         |
| 3.2      | Arquitetura do Servidor . . . . .                              | 5         |
| 3.3      | Modelo de Dados . . . . .                                      | 7         |
| 3.3.1    | <i>Collection</i> Users . . . . .                              | 7         |
| 3.3.2    | <i>Collection</i> Equivalências . . . . .                      | 7         |
| 3.4      | Endpoints da API . . . . .                                     | 8         |
| 3.4.1    | Utilizadores . . . . .   | 9         |
| 3.4.2    | Universidades . . . . .  | 11        |
| 3.4.3    | Processos . . . . .  | 12        |
| <b>4</b> | <b>Frontend</b>  | <b>16</b> |
| 4.1      | Design e <i>wireframing</i> da <i>User Interface</i> . . . . . | 16        |
| 4.2      | <i>Framework</i> Vue.js . . . . .                              | 18        |
| 4.3      | Arquitetura . . . . .  | 26        |
| <b>5</b> | <b>Deployment</b>  | <b>27</b> |
| 5.1      | <i>Deployment</i> de Serviços Iterativamente . . . . .         | 27        |
| 5.1.1    | Development . . . . .  | 27        |
| 5.1.2    | Production . . . . .   | 28        |
| 5.2      | Virtualização Utilizando Docker . . . . .                      | 28        |
| 5.3      | Variáveis de Ambiente . . . . .                                | 29        |
| 5.3.1    | Backend . . . . .  | 29        |
| 5.3.2    | VueJS . . . . .  | 30        |
| <b>6</b> | <b>Conclusão</b>   | <b>31</b> |

# Lista de Figuras

|   |    |
|---|----|
| 2.1 Orquestração de Serviços . . . . .  | 4  |
| 3.1 Arquitetura do servidor Backend . . . . .   | 6  |
| 4.1 Wireframe da página de login . . . . .  | 17 |
| 4.2 Wireframe da página de processo de equivalência . . . . .   | 17 |
| 4.3 Wireframe da página principal com os processos de equivalência . . . . .                              | 18 |
| 4.4 Página de login dos utilizadores . . . . .  | 19 |
| 4.5 Página principal (dashboard) da aplicação . . . . .   | 20 |
| 4.6 View do Student/Process na tab da informação do processo em causa . . . . .                           | 21 |
| 4.7 View do Student/Process na tab da informação das equivalências efetuadas . . . . .                    | 22 |
| 4.8 View do Student/Process na tab da criação duma nova equivalência . . . . .                            | 23 |
| 4.9 View do Student/Process na tab de exportação de documentos PDF . . . . .                              | 23 |
| 4.10 View da página das Settings com a gestão individual dos próprios dados. (Modo Normal) . . . . .      | 24 |
| 4.11 View da página das Settings com a gestão dos dados das universidades. (Modo Administrador) . . . . . | 25 |
| 4.12 View da página das Settings com a gestão dos dados dos utilizadores. (Modo Administrador) . . . . .  | 25 |
| 4.13 Arquitetura do Cliente . . . . .   | 26 |

# Capítulo 1

## Introdução

No âmbito da Unidade Curricular de Laboratório em Engenharia Informática, inserida no plano de estudos de Mestrado em Engenharia Informática, foi desenvolvido um *Software* que visa auxiliar a gestão de processos de equivalências de alunos provenientes de outros cursos e na sua equivalência entre a Universidade do Minho e/ou faculdades externas.

### 1.1 Estrutura do relatório

Inicialmente, o presente relatório incidirá sobre as tecnologias utilizadas em cada serviço, sejam estas bases de dados ou servidores da aplicação, seguindo por expor as funcionalidades pretendidas e incorporadas na aplicação final.

O capítulo 2 incidirá exclusivamente sobre a arquitetura da Solução desenvolvida. Será dada a conhecer uma arquitetura generalizada de todo o sistema, apresentando o fluxo de pedidos e troca de informação entre micro-serviços. De forma a aprofundar a forma como cada micro-serviço foi implementado, serão apresentadas e discutidas as arquiteturas de cada serviço de forma pormenorizada.

Seguidamente, serão abordadas as tecnologias utilizadas para concretizar o sistema, designado por **Mantis**, dando ênfase, inicialmente, no *Backend*, aprofundando as tecnologias e bibliotecas utilizadas e informações relativamente à API de dados fornecida por este servidor (capítulo 3). Posto isto, será de seguida apresentada a aplicação final, a parte visual à qual o utilizador deverá utilizar para executar as diversas funções do sistema, explicando como esta funciona (capítulo 4), recorrendo a diversas imagens, servindo também como manual de utilização da mesma.

Acontecerá ainda um capítulo que demonstrará as várias formas de efetuar um *deployment* desta aplicação com uma atenção ao detalhe mas simples de entender por forma a toda a gente conseguir correr este *software* da melhor maneira possível e que lhe traga menos dificuldades iniciais.

Em forma de desfecho será apresentada uma conclusão retrospectiva em relação a todo o trabalho desenvolvido na plataforma.

## Capítulo 2

# Arquitetura do Sistema

O Sistema desenvolvido foi decomposto em vários micro-serviços, sendo que cada um representará diferentes papéis no contexto do sistema, destacando-se o servidor Backend de API de dados e o Cliente.

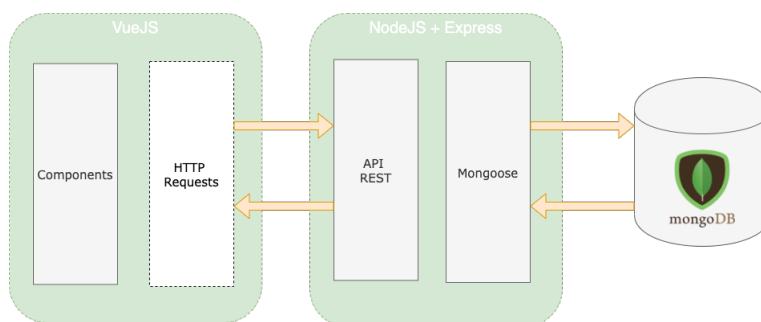


Figura 2.1: Orquestração de Serviços

Como sistema de base de dados, foi optado o armazenamento em MongoDB, que será acessado através de um Servidor desenvolvido em NodeJS com recurso ao Mongoose. Esta API de dados, desenvolvida com a *Framework ExpressJS*, fornece uma interface REST, que será discutida ao longo do Capítulo 3.

Por fim, de forma a representar a informação e a auxiliar a gestão dos processos, foi desenvolvido um Cliente em VueJS que, recorrendo ao *Axios*, comunicará com a API através de pedidos HTTP GET, PUT, POST, DELETE e, em alguns casos PATCH.

# Capítulo 3

## *Backend*

De forma a que o projeto fosse desenvolvido de forma consistente, por imposição e de forma a escalar, deu-se prioridade ao desenvolvimento do *Backend* recorrendo ao **NodeJS**. Como tal, foram utilizadas diversas tecnologias de forma a efectuar operações **CRUD** na base de dados de sistema NoSQL e de forma a oferecer uma interface **REST** para o Cliente/*Frontend*.

### 3.1 Tecnologias Utilizadas

Para cumprir com todos os requisitos funcionais estipulados, foram utilizadas diversas bibliotecas no servidor API do sistema, entre os quais:

- **ExpressJS**: Framework utilizada para o desenvolvimento do servidor com interface REST;
- **Mongoose**: Conexão com o servidor de base de dados, modelação e manipulação de dados;
- **BcryptJS**: utilizado para a encriptação de passwords, antes de serem armazenadas na base de dados;
- **PDFmake**: biblioteca para a geração de ficheiros PDF

A conjugação das *Frameworks* **Mongoose** e **ExpressJS** permitiu construir um servidor que segue um padrão arquitetural em MVC (*Model-View-Controller*). Essa mesma arquitetura do servidor será o tópico da secção seguinte.

### 3.2 Arquitetura do Servidor

Tal como foi dado a conhecer, a utilização das *frameworks* **ExpressJS** e **Mongoose** permitem construir um servidor com um padrão arquitetural MVC, *Model-View-Controller*, em que os modelos são construídos com auxílio do **Mongoose**, a *View* é a resposta aos pedidos em formato JSON e os *Controllers* que controlam em forma de *Query* dos dados.

Foi optado este padrão arquitetural pela organização do código, o que permite a futuras alterações ou adições de funcionalidades de forma mais simples e sem grande esforço. Na figura 3.1, representa-se a arquitetura nuclear deste micro-serviço criado.

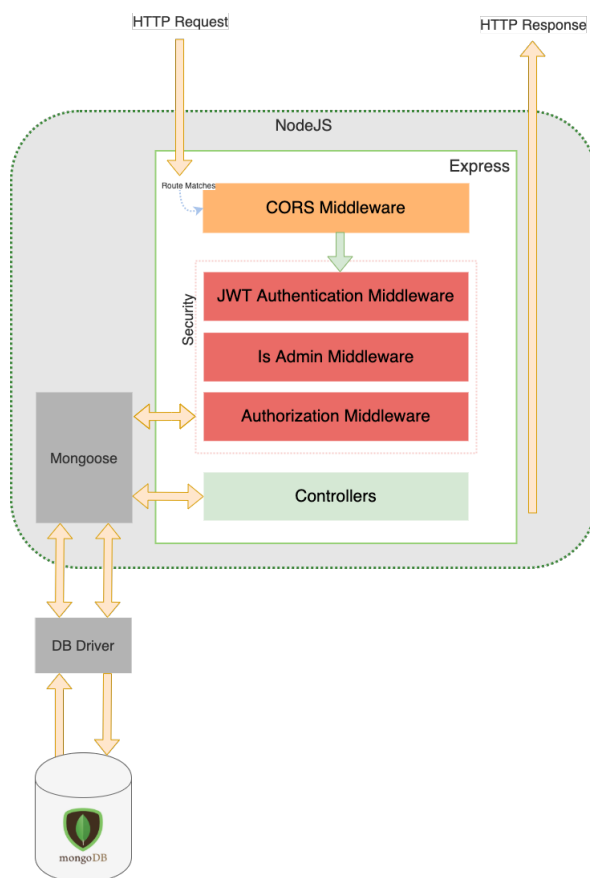


Figura 3.1: Arquitetura do servidor Backend

Quando são efetuados pedidos a este serviço, estes passam sempre por uma camada *Middleware*, mais conhecida por *CORS*, que permite bloquear pedidos por clientes desconhecidos. Neste caso, como foi optado pelo desenvolvimento de micro-serviços, o URL do cliente desenvolvido em **VueJS** é conhecido, permitindo aceder às demais rotas fornecidas pelo sistema.

Dada a sensibilidade de alguns dados, é extremamente necessário a autenticação dos utilizadores ao sistema. A verificação é feita através de uma *Middleware* que verifica o estado do *Token* fornecido. Em alguns casos, é também verificado, caso o *Token* seja válido, se pertence a uma conta de administrador e, se afirmativo, a rota que se pretende aceder será desbloqueada. Caso estas verificações falhem, o acesso será barrado nas camadas de *Middleware*.



## 3.3 Modelo de Dados

Os dados são armazenados num Sistema de Base de Dados *NoSQL*, pelo que são ficheiros em formato de JSON em diferentes *Collections*. Nesta secção, é dado a conhecer todas as *Collections* que compõem o nosso sistema, informação sobre esses dados e o tipo de informação armazenada nos mesmos.

### 3.3.1 *Collection* Users

A *Collection* Users permite armazenar as seguintes informações sobre os utilizadores da plataforma.

---

```
1 {
2   "_id": "OBJECT ID, atribuido pelo MongoDB",
3   "email": "STRING, com EMAIL DO UTILIZADOR",
4   "fullName": "STRING, com NOME DO UTILIZADOR",
5   "password": "STRING, encriptada",
6   "admin": "BOOLEAN, administrador se TRUE"
7 }
```

---

Desta forma, é possível identificar o Utilizador e desbloquear o acesso à plataforma **Mantis** e, ainda, atribuir cargos de administração.

### 3.3.2 *Collection* Equivalências

A *Collection* de equivalências é composta por três partes. Em primeiro lugar, um objeto com informações gerais relativamente ao processo. Dentro deste mesmo objeto, a propriedade equivalências é um *Array* com vários objetos que definem as unidades curriculares equivalentes do processo. Para além disso, são armazenadas informações sobre a documentação gerada para este processo. Assim sendo:

#### Base da *Collection*

---

```
1 {
2   "_id": "OBJECT ID, atribuido pelo MongoDB",
3   "processo": "STRING, ID atribuido ao Processo",
4   "idAluno": "STRING, identificador do Aluno",
5   "initiatedBy": "STRING, username de quem iniciou o projecto",
6   "nomeAluno": "STRING, Nome completo do Aluno",
7   "instProv": "STRING, Instituicao que o Aluno e proveniente",
8   "cursoProv": "STRING, Curso proveniente do Aluno",
9   "data": "DATA de criacao do processo",
10  "equivalencias": "ARRAY de objetos de EQUIVALENCIA",
```

```
11   "documentation": "ARRAY de objetos de DOCUMENTATION"
12 }
```

---

## Equivalencias

---

```
1 {
2   "ucRealizada": "STRING, nome da UC realizada",
3   "credits": "NUMBER, numero de creditos da UC realizada",
4   "nota": "NUMBER, de 0~20, com a nota da UC realizada",
5   "anoLetivo": "STRING, formato YYYY/YYYY, ano letivo da UC realizada",
6   "percent": "NUMBER, percentagem",
7   "ucEquiv": "STRING, nome da UC que sera dada equivalencia",
8   "semUcEquiv": "STRING, formato Numero Semestre, da UC equivalente",
9   "createdBy": "STRING, nome de utilizador de quem concedeu a equivalencia",
10  "createdAt": "DATA de insercao da equivalencia"
11 }
```

---

## Documentation

---

```
1 {
2   "_id": "OBJECT ID, atribuido pelo MongoDB",
3   "generatedAt": "DATA em que o ficheiro foi gerado",
4   "generatedBy": "STRING, username da conta que gerou o ficheiro",
5   "filename": "STRING, nome atribuido ao ficheiro",
6 }
```

---

## 3.4 Endpoints da API

Visto que o sistema foi implementado recorrendo a desenvolvimento de micro-serviços, o servidor *Backend* possui uma interface HTTP com padrão REST para cada *Collection* supra mencionada.

Ainda que não aconselhável, o utilizador final da plataforma poderá usufruir desta interface. Como tal, serão apresentados os diferentes *endpoints* desta API, bem como formulação de pedidos, *body*, *headers* e parâmetros.

### 3.4.1 Utilizadores

Os *endpoints* da API de utilizadores permitem obter informações sobre utilizadores registados, criar novas contas, efectuar atualizações de dados e autenticação na plataforma. Pedidos HTTP são respondidos nesta interface na rota */user*, em que possui os seguintes tipos de pedidos:

#### POST /login

Permite a autenticação no serviço. É gerado um *JSON Web Token*, que deve ser armazenado apenas durante uma hora. O corpo do pedido HTTP, deverá ser o seguinte:

|                        |   |
|------------------------|---|
| username<br>(required) | Username usado no registo na plataforma |
| password<br>(required) | Password da Conta                       |

#### PUT /update

Esta rota permite atualizar as informações da conta em que foi iniciada sessão. Como tal, é importante enviar no Cabeçalho do pedido HTTP e o *JSON Web Token* obtido aquando o *login*. É importante mencionar, o campo *admin* será apenas atualizado caso a conta em que foi iniciada sessão, tenha sido previamente registada como administrador.

| Headers | userToken              | JSON Web Token              |
|---------|------------------------|-----------------------------|
| Body    | fullName<br>(required) | Nome do utilizador da conta |
|         | email<br>(required)    | Email do utilizador         |
|         | admin<br>(reserved)    | True se administrador       |

## PATCH /username/:username

Permite atualizar o nome de utilizador de uma determinada conta, em que *:username* uma parâmetro de nome de utilizador referente à conta a atualizar. De forma a barrar a utilização desta rota, apenas serão aceites pedidos de administradores do sistema.

|         |                        |                         |
|---------|------------------------|-------------------------|
| Headers | userToken              | JSON Web Token          |
| Body    | username<br>(required) | Novo nome de utilizador |

## POST /register

Esta rota permite registar novas contas na plataforma. Dada a sensibilidade dos dados, apenas administradores do sistema poderão registar novos utilizadores.

|         |                        |   |
|---------|------------------------|---|
| Headers | userToken              | JSON Web Token  |
| Body    | username<br>(required) | Nome de utilizador da conta   |
|         | password<br>(required) | Password da conta, será encriptada à posteriori   |
|         | email<br>(required)    | E-mail do utilizador da conta   |
|         | fullName<br>(required) | Nome completo do utilizador da conta  |
|         | admin<br>(optional)    | Se omitido, a conta será standard. Para conta de administrador, deverá ser preenchido como true |

## PUT /password

Atualização da *password* da conta atual. É necessário enviar a *password* actual, juntamente com a nova *password*, de forma a confirmar a atualização. Caso a *password* antiga não corresponda à *password* registada no sistema, a atualização é ignorada.

|         |                           |                |
|---------|---------------------------|----------------|
| Headers | userToken                 | JSON Web Token |
| Body    | oldPassword<br>(required) | Password atual |
|         | newPassword<br>(required) | Nova password  |

### **DELETE /:username**

Elimina a conta do utilizador de parâmetro `/:username`. Esta rota está protegida e reservada apenas a administradores do sistema.

### **GET /**

Obtém a lista de utilizadores registados na plataforma. Tal como a rota anterior, apenas administradores poderão aceder a esta rota.

## **3.4.2 Universidades**

Os *endpoints* da API de universidades encontram-se protegidos por funções *middleware*, que verificam a autenticação dos utilizadores. Conforme o papel desempenhado, permitem diversas operações CRUD sobre a *Collection* de Universidades. Pedidos HTTP são respondidos nesta interface na rota `/universidade`, e fornece os seguintes tipos de pedidos:

### **GET /**

Listagem de todas as Universidades registadas na plataforma. É possível listar a informação sobre as faculdades e cursos existentes, fazendo uso das diferentes *Query Strings*.

| Headers | userToken  | JSON Web Token  |
|---------|--|---|
| Query   | university<br>(optional)                               | Retorna os nomes dos cursos da universidade                   |
|         | course<br>(optional [university required])             | Retorna o nome das UC registadas de um Curso                  |
|         | subject<br>(optional [university and course required]) | Retorna equivalencias identicas dada a Universidade e o Curso |

### **POST /**

Permite criar uma nova faculdade. Apenas os Administradores poderão registar novas Universidades.

### **DELETE /:id**

Elimina uma Faculdade do Sistema com o código `:id`. Apenas os Administradores poderão eliminar faculdades.

|         |                          |   |
|---------|--------------------------|---|
| Headers | userToken                | JSON Web Token                            |
| Body    | codInstit<br>(required)  | Código da Instituição atribuído pela DGES |
|         | nomeInstit<br>(required) | Nome da Instituição de Ensino             |

### PUT /:id

Atualiza as informações de uma Faculdade com código :id. Apenas os Administradores poderão atualizar dados das faculdades.

|         |                          |   |
|---------|--------------------------|---|
| Headers | userToken                | JSON Web Token                            |
| Params  | codInstit                | Código da instituição                     |
| Body    | codInstit<br>(required)  | Código da Instituição atribuído pela DGES |
|         | nomeInstit<br>(required) | Nome da Instituição de Ensino             |

### 3.4.3 Processos

Os *endpoints* da API de Processos permitem obter informações e criar novos processos, atribuir equivalências e gerar documentação. Os pedidos HTTP são respondidos nesta interface na rota */processo*, em possui os seguintes tipos de pedidos:

#### GET /

O pedido GET na raiz da rota, permite obter informações gerais de processos registados, como por exemplo identificador do processo, nome do aluno, numero mecanográfico, instituição e curso proveniente. Tal como todas as rotas dos processos, encontra-se protegida por autenticação.

|         |                    |  |
|---------|--------------------|--|
| Headers | userToken          | JSON Web Token                                   |
| Query   | year<br>(optional) | Obtém os processos iniciados num determinado ano |

## GET /:id

Obtém todas as informações relativamente a um determinado processo com o ID descrito em /:id

|         |                  |   |
|---------|------------------|---|
| Headers | userToken        | JSON Web Token                                      |
| Params  | id<br>(required) | Obtém informações relativamente a um único processo |

## POST /:id/generate

Gera documentação em PDF para um processo cujo o id é /:id.

|         |                  |   |
|---------|------------------|---|
| Headers | userToken        | JSON Web Token                          |
| Params  | id<br>(required) | Gera documentação em PDF do processo ID |

## POST /

Cria um novo processo respeitando os seguintes parâmetros do *Body*.

|         |                         |                                       |
|---------|-------------------------|---------------------------------------|
| Headers | userToken               | JSON Web Token                        |
| Body    | processo<br>(required)  | Identificador do Processo             |
|         | idAluno<br>(required)   | Identificador do Aluno                |
|         | nomeAluno<br>(required) | Nome do Aluno                         |
|         | instProv<br>(required)  | Instituição que o aluno é proveniente |
|         | cursoProv<br>(required) | Curso proveniente                     |

## **PUT /:id**

Atualiza as informações do processo cujo o ID respeita o parâmetro /:id. O corpo do pedido deverá respeitar a seguinte estrutura.

| Headers | userToken               | JSON Web Token                        |
|---------|-------------------------|---------------------------------------|
| Body    | processo<br>(required)  | Identificador do Processo             |
|         | idAluno<br>(required)   | Identificador do Aluno                |
|         | nomeAluno<br>(required) | Nome do Aluno                         |
|         | instProv<br>(required)  | Instituição que o aluno é proveniente |
|         | cursoProv<br>(required) | Curso proveniente                     |

## **GET /:id/files**

Lista toda a documentação gerada para o processo cujo o ID respeita o parâmetro /:id.

| Headers | userToken        | JSON Web Token |
|---------|------------------|----------------|
| Param   | id<br>(required) | ID do Processo |

## **GET /:id/file/:filename**

Descarrega a documentação em PDF do processo cujo o ID respeita o parâmetro /:id e o nome do ficheiro /:filename.



| Headers | userToken              | JSON Web Token   |
|---------|------------------------|------------------|
| Param   | id<br>(required)       | ID do Processo   |
|         | filename<br>(required) | Nome do ficheiro |

### **POST /:id/subject**

Permite atribuir novas equivalências a um processo cujo o ID respeita o parâmetro /:id. O corpo do pedido HTTP deverá respeitar a seguinte estrutura.

| Headers | userToken                 | JSON Web Token  |
|---------|---------------------------|---|
| Body    | semUcEquiv<br>(required)  | Semestre da UC equivalente (1º Semestre ou 2º Semestre) |
|         | ucEquiv<br>(required)     | Nome da UC equivalente                                  |
|         | anoLetivo<br>(required)   | Ano letivo de conclusão da UC Equivalente               |
|         | percent<br>(required)     | Percentagem da equivalencia                             |
|         | nota<br>(required)        | Nota atribuída à equivalencia, entre 0 e 20             |
|         | ects<br>(required)        | Número de créditos atribuídos à UC Equivalente          |
|         | ucRealizada<br>(required) | Nome da UC já realizada                                 |

# Capítulo 4

## *Frontend*

Com o *backend* desenvolvido, com as funcionalidades e rotas necessárias para o funcionamento arquitectural da aplicação como um todo, começou-se o trabalho visual e conceptual sobre como a aplicação devia parecer, funcionar e proporcionar todas as suas capacidades prometidas ao utilizador final.

Numa primeira fase iremos explorar alguns *wireframes* que foram a conceção inicial do *User Experience (UX)* e numa última fase a sua implementação utilizando a biblioteca de componentes **Vuetify** e a sua integração com a **JavaScript framework** intitulada de **Vue.js**.

### 4.1 Design e *wireframing* da *User Interface*

Utilizando a aplicação **Balsamiq** efetuamos uns *wireframes* para cada *view* pensada de raiz, ou seja, existiria uma página de *login*, uma página para o processo escolhido e a trabalhar e a página principal com a lista de processos presentes na base de dados central.

Desta forma, a nossa ideia para toda a *Webapp* foi que seria simples, funcional, responsiva para os vários tipos e tamanhos de ecrã, mas também com a acessibilidade em mente desde de a raiz e conceção do projeto, como deve ser qualquer produto de *software* atual.

A página de *login* não foge a esse *ethos* pensado pela equipa e dessa forma o seu design foi pensado desta maneira, como demonstra a figura 4.1.

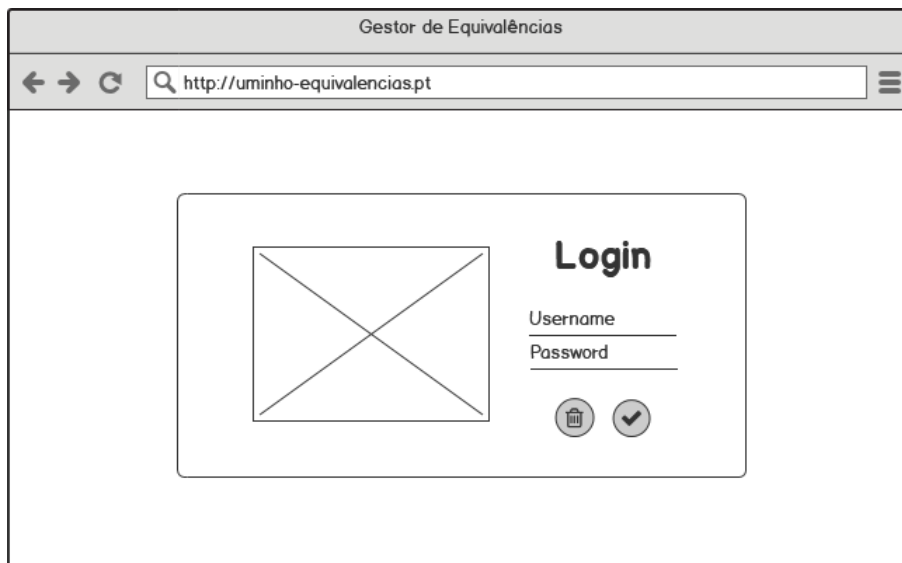


Figura 4.1: Wireframe da página de *login*

Por conseguinte, a página de gestão do processo de equivalência foi concebido desta maneira, como demonstra a figura 4.2.

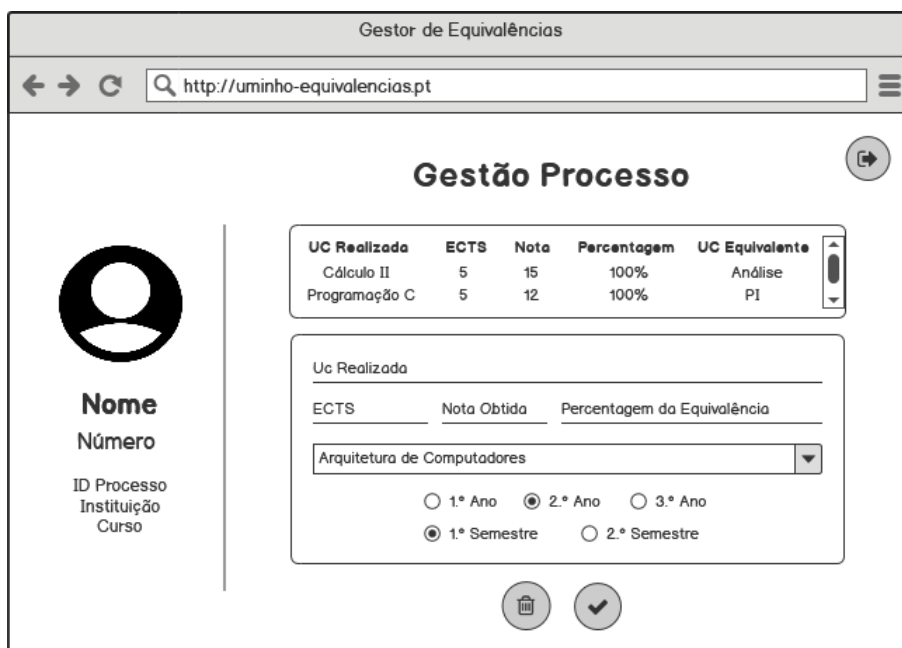


Figura 4.2: Wireframe da página de processo de equivalência

Por fim, temos a página principal, que teve como pensamento duas colunas, uma com a lista de processos em trabalho e outra para novos processos de equivalência que os utilizadores deste *software* possam criar. Assim, a versão do UI foi pensada e demonstrada na figura 4.3.

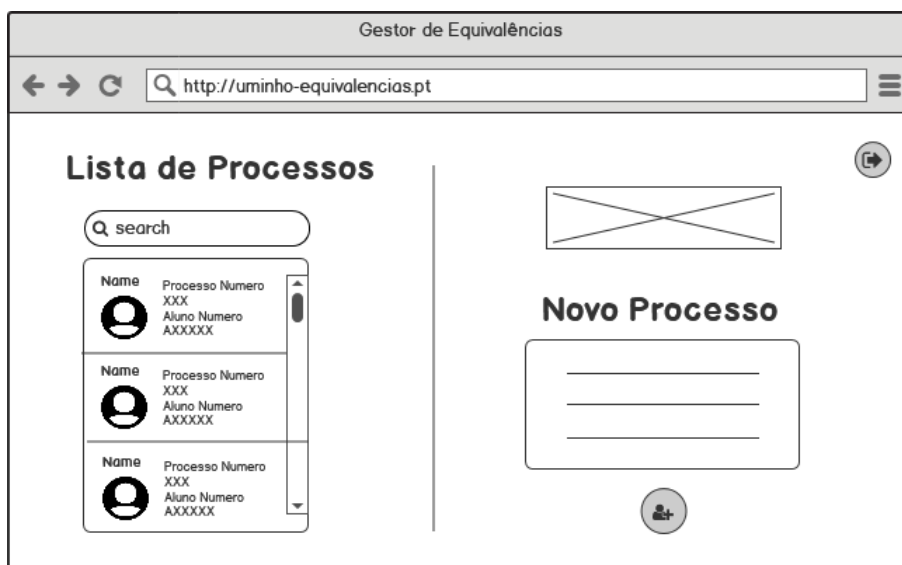


Figura 4.3: Wireframe da página principal com os processos de equivalência

## 4.2 Framework Vue.js

Para a concretização de todo a conceção inicial demonstrada na secção anterior, que começa na página 16, foi consenso da equipa a utilização da ferramenta **Vue.js**, uma *framework* em **JavaScript** totalmente *open-source* que é utilizada para criar *user interfaces* e aplicações de página única com uma reatividade célere e sempre com o modelo de *MVVM*, ou seja, tem como base a plataforma arquitetural de *model-view-viewmodel*, separando assim o *backend/model* da parte visual (*view*).

Toda a aplicação **Vue.js** tem como base alguns componentes extra utilizados, tanto *first-party*, como o **Vue Router**, **Vue Cookies**, **Vuex** mas também utilizamos *plugins* de terceiros como a panóplia de ícones da *Font Awesome* e o mais importante, a biblioteca **Vuetify** que nos trouxe os componentes baseados no *Material Design* como fundação basilar do design e funcionalidade das nossas páginas a desenvolver.

A nossa aplicação **Vue.js** está dividida por quatro *views*, em que as mesmas têm uma variedade de componentes para uma modularização mais segura e simples de entender. Assim as nossas quatro *views* são:

- Login Page
- Dashboard Page
- Student/Process Page
- Settings Page

Como início, houve a criação dum *state*, utilizando o **Vuex** para o *middleware* do *login* e a sua parte consequente das *cookies* quando a pessoa efetivamente entrava no sistema. Assim através do *parse* da resposta JSON dada pelo *backend* (explicado em detalhe no capítulo 3), o utilizador e o sistema confirmam a sua sessão e é criada uma *cookie* de sessão aceite para depois entrar.

Para isto tudo acontecer, foi criada uma página/*view* de *login*, com apenas um componente como demonstra a figura 4.4.



Figura 4.4: Página de *login* dos utilizadores

Como é possível na figura acima (4.4), foi criado um componente **Vue** com os campos de texto, botões e alertas fornecidos pela biblioteca **Vuetify**, de forma a criar um *design*, não só similar ao decidido na secção 4.1, mas também funcional, responsivo aos vários tipos de ecrã e acessível.

Com a sessão iniciada, temos a *view* da *Dashboard*, que é dividida em duas colunas, uma com a lista de processos de equivalência na base de dados do sistema e a outra com um componente para criar um novo processo de equivalência pelo utilizador da aplicação.

The screenshot shows the Mantis application dashboard. At the top is a dark navigation bar with the 'mantis' logo, a 'Página Inicial' link with a home icon, a 'Definições' link with a gear icon, and a 'Sair' link with a door icon. Below the navigation bar, the dashboard is split into two columns. The left column features a search bar labeled 'Procurar Processo Equivalência' and a scrollable list of four process entries. Each entry includes a circular icon with initials, the name, the process name, and the student number. The right column is titled 'Registar Novo Processo Equivalência' with a note that asterisks indicate required fields. It contains four input fields: 'Identificador do Processo', 'Identificador do Aluno \*', 'Nome Completo do Aluno \*', and 'Instituição Proveniente \*'. Below these are two dropdown menus for 'Curso Proveniente \*'. At the bottom of the right column is a toggle switch for 'Curso Inexistente', a green 'Criar Processo' button, and a green circular icon with a document symbol.

| Initials | Name                    | Process                | Student Number      |
|----------|-------------------------|------------------------|---------------------|
| CS       | Cecília Soares          | Processo INT-SAUM/2... | Aluno Número A34900 |
| IB       | Ivo Alexandre Pereir... | Processo INT-SAUM/2... | Aluno Número A86579 |
| JR       | José Carlos Ramalho     | Processo jcr/teste     | Aluno Número A4140  |
| JC       | João Coutinho           | Processo INT-SAUM/2... | Aluno Número A86272 |

Figura 4.5: Página principal (*dashboard*) da aplicação

Na figura acima (4.5), podemos ver então a implementação do *wireframe* pensado, com listas, barras de pesquisa, campos de texto, *sliders* e botões, que em conjunto trouxeram a ideia e funcionalidade de toda a página e a sua fruição para o de cima.

É de salientar que existe um componente que é sempre fixo e presente em todas as *views* mencionadas aqui (excetuando o *Login*) que é a *Navbar*, demonstrada também na figura 4.5 que contém os botões para aceder às *views Dashboard* e *Settings* demonstradas nas figuras 4.5 e 4.10 respetivamente. Tem ainda um botão para terminar a sessão que elimina a *cookie* criada e retira do *state* as informações desse utilizador até uma próxima sessão seja criada pelo mesmo.

Aquando a escolha dum processo na lista de processos da *dashboard*, o utilizador é redirecionado para a *view* do *Student/Process* que é um conjunto de quatro componentes fulcrais cada um colocado numa *tab* fornecida pela biblioteca *Vuetify* mais uma vez.

**mantis** Página Inicial Definições Sair

## Processo Equivalência

Número INT-SAUM/2018/4424

Aluno(a) Cecília Soares  
Número A34900

INFORMAÇÕES GERAIS LISTA EQUIVALÊNCIAS FORMULÁRIO NOVA EQUIVALÊNCIA DOCUMENTOS

\* Campos de preenchimento obrigatório

Identificador do Processo \*  
INT-SAUM/2018/4424

Identificador do Aluno \*  
A34900

Nome do Aluno \*  
Cecília Soares

Instituição Proveniente \*  
Universidade do Minho

Curso Proveniente \*  
Direito

☐ Modo de Leitura ☐ Curso Inexistente Guardar Alterações

Figura 4.6: *View* do *Student/Process* na *tab* da informação do processo em causa

É nesta *tab* (demonstrada na figura 4.6) que podemos alterar as informações gerais do processo como o identificador, o número, nome, instituição e curso proveniente do aluno em causa. Esta edição está desativada por defeito, tendo sido colocado um *slider* para os casos que seja realmente necessário alterar.

Alguns campos de texto presentes no nosso projeto têm colocado um *auto-complete* para os cursos, unidades curriculares, ou até mesmo uma pequena inteligência/algoritmia para completar equivalências já feitas anteriormente. Dessa forma, o campo de texto da instituição proveniente aqui demonstrado têm como base os dados da Direção Geral de Ensino Superior, mas caso seja necessário a inserção manual para os casos do curso ser inexistente, a aplicação está preparada também para tal.

## Processo Equivalência

Número INT-SAUM/2018/4424

Aluno(a) Cecília Soares  
Número A34900

INFORMAÇÕES GERAIS

LISTA EQUIVALÊNCIAS

FORMULÁRIO NOVA EQUIVALÊNCIA

DOCUMENTOS

| UC Equivalente                  | Nota UC | ECTS | UC Realizada          |
|---------------------------------|---------|------|-----------------------|
| Opção Uminho (2ºano-2ºsem MIEI) | 17      | 6    | Direitos Fundamentais |
| Opção Uminho (2ºano-2ºsem MIEI) | 20      | 1    | Direitos Fundamentais |



Figura 4.7: View do Student/Process na tab da informação das equivalências efetuadas

Nesta segunda *tab*, demonstrada na figura 4.7 temos uma tabela com paginação que demonstra as equivalências já efetuadas e guardadas na base de dados. As informações das Unidades Curriculares são desde o nome, à nota, os créditos ECTS e qual a UC que foi dita como equivalente à realizada.

Na terceira *tab*, visionada na figura 4.8 temos um formulário criado em **Vuetify** para a introdução de uma nova equivalência no processo em questão. Dessa forma, como filtragem e análise prévia da introdução de dados, foram utilizadas regras com expressões regulares para aceitar apenas o ano letivo de maneira correta, a nota obtida dentro dos valores aceites e também a utilização dum *auto-complete* para as unidades curriculares que possam já ter tido equivalência em processos anteriores, colocando assim uma pequena inteligência artificial no projeto, de forma a facilitar todo o algoritmo e por consequência a experiência final do utilizador.

Na quarta e última *tab* deste componente, ilustrada na figura 4.9, temos o local onde serão colocados todos os documentos gerados para validar este processo de equivalência, mas também a possibilidade de efetuar o descarregamento do mesmo ou então a geração dum novo, sempre com o *timestamp* e o nome do utilizador da aplicação responsável por esta nova geração.



# Processo Equivalência

Número INT-SAUM/2018/4424

Aluno(a) Cecília Soares  
Número A34900



INFORMAÇÕES GERAIS
LISTA EQUIVALÊNCIAS
FORMULÁRIO NOVA EQUIVALÊNCIA
DOCUMENTOS

\* Campos de preenchimento obrigatório

Regras: Ex. Ano Letivo: 2019/2020; Ex. Percentagem: 50

UC Realizada \*

Direitos Fundamentais

X

UC Equivalente \*

Opção Uminho (2ºano-2ºsem MIEI)


Ano Letivo de Conclusão \*

Nota Obtida \*

ECTS \*

Percentagem da ...

Figura 4.8: View do *Student/Process* na *tab* da criação duma nova equivalência




Página Inicial
Definições
Sair

Processo Equivalência











Número INT-SAUM/2018/4424

Aluno(a) Cecília Soares
Número A34900



INFORMAÇÕES GERAIS
LISTA EQUIVALÊNCIAS
FORMULÁRIO NOVA EQUIVALÊNCIA
DOCUMENTOS

Gerar Novo Documento do Processo

|   |  |  |
|---|--|--|
|  | Data e Hora de Criação: 18/07/2020, 00:35<br>Utilizador Responsável: Carlos Miguel Rebelo Solans | Download  |
|  | Data e Hora de Criação: 18/07/2020, 00:35<br>Utilizador Responsável: Administrador do Serviço    | Download  |
|  | Data e Hora de Criação: 18/07/2020, 00:35<br>Utilizador Responsável: Administrador do Serviço    | Download  |
|  | Data e Hora de Criação: 18/07/2020, 00:34<br>Utilizador Responsável: Administrador do Serviço    | Download  |
|  | Data e Hora de Criação: 18/07/2020, 00:32<br>Utilizador Responsável: Administrador do Serviço    | Download  |

<
1
2
>

Figura 4.9: View do *Student/Process* na *tab* de exportação de documentos PDF

Por fim, a última página/view criada acaba por ser imperativa e necessária para a manipulação dos utilizadores da aplicação ou até mesmo para uma alteração por parte dos administradores às listas de universidades ou utilizadores da *app*.

Dessa forma, foi criada a *view* das *Settings* que é dividida em duas partes implícitas:

1. **O modo normal:** para alteração dos dados do utilizador com sessão iniciada.
2. **O modo administrador:** para poder-se alterar a lista de utilizadores da aplicação e/ou introduzir novos utilizadores no sistema, dado que a aplicação não têm uma página de registo por opção superior de forma a criar uma segurança mais rigorosa. Este modo ainda oferece a possibilidade de introduzir/alterar as universidades presentes na base de dados do sistema.

The screenshot shows the Mantis application interface. At the top is a dark navigation bar with the 'mantis' logo, 'Página Inicial' with a home icon, 'Definições' with a gear icon, and 'Sair' with a logout icon. Below this is a white header area with 'Definições' on the left, and on the right, the user information: 'Utilizador(a) Administrador do Serviço @admin Administrador' next to a green circular profile icon. A horizontal menu below the header contains three items: 'GESTÃO CONTA E SEGURANÇA' (highlighted with a green underline), 'GESTÃO UTILIZADORES', and 'GESTÃO INSTITUIÇÕES'. The main content area is a white card titled 'Informações Gerais' with a subtitle 'Gestão do Nome, Username, Email e Tipo de Utilizador ^'. It includes a note '\* Campos de preenchimento obrigatório'. There are four input fields: 'Nome do Utilizador \*' (containing 'Administrador do Serviço'), 'Username do Utilizador \*' (containing 'admin'), 'E-mail do Utilizador \*' (containing 'admin@mantis.com'), and a checkbox for 'Administrador \*' which is checked. A 'Guardar Alterações' button is to the right of the checkbox. At the bottom of the card is a toggle switch for 'Modo de Leitura' which is turned on. Below the main card is a secondary section titled 'Segurança' with a subtitle 'Alteração da Password v'.

Figura 4.10: View da página das *Settings* com a gestão individual dos próprios dados. (**Modo Normal**)

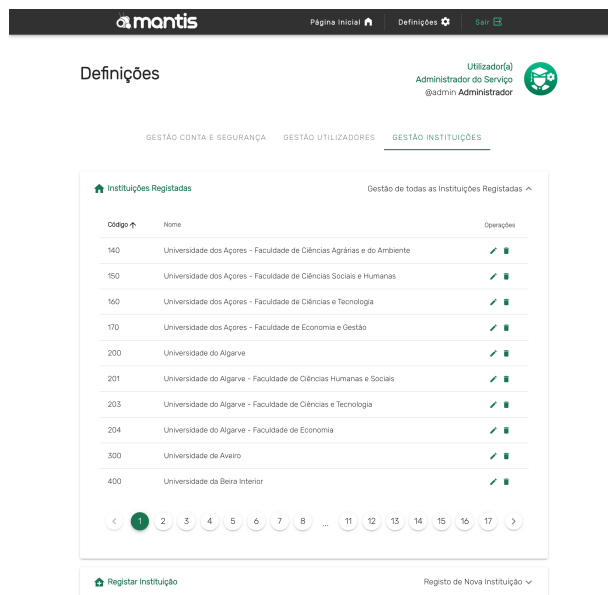


Figura 4.11: View da página das *Settings* com a gestão dos dados das universidades. (**Modo Administrador**)

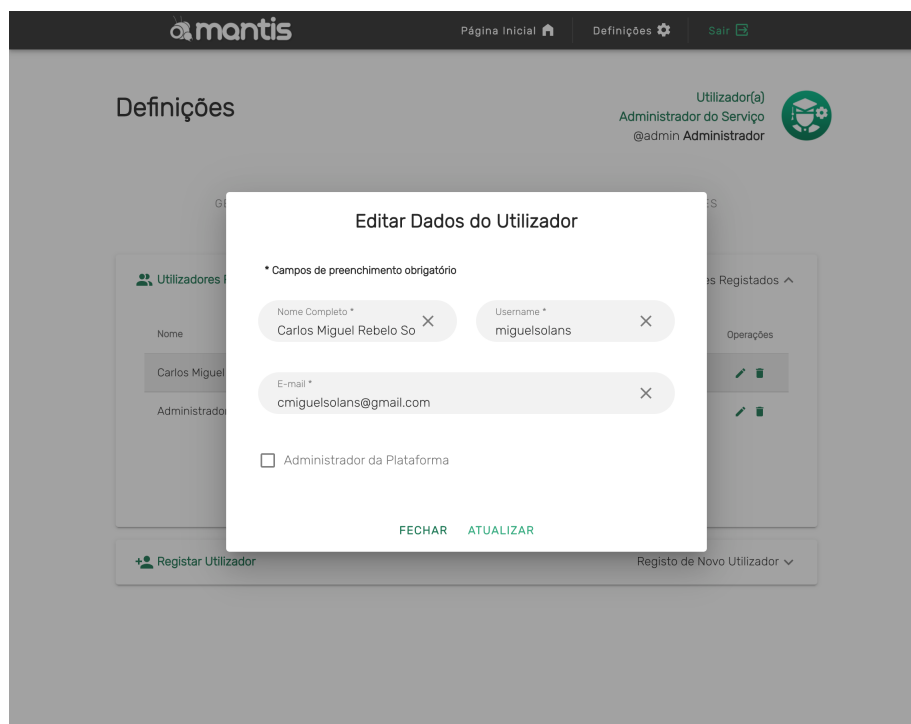


Figura 4.12: View da página das *Settings* com a gestão dos dados dos utilizadores. (**Modo Administrador**)

Assim, como demonstrado na figura 4.11 os administradores do sistema podem alterar as informações das universidades existentes, desde o código da mesma ao nome e também in-

introduzir novas universidades à base de dados central caso necessitem.

Por fim, no componente da gestão dos utilizadores, os administradores podem criar novos utilizadores, gerar novas credenciais para os mesmos e também alterar dados de utilizadores existentes como por exemplo dar privilégios de administrador a um outro utilizador, como é possível se ver na figura 4.12.

Com todas estas *views*, componentes e variadas funcionalidades explicadas, temos uma aplicação não só com o intuito pedido a funcionar, como elevamos e fortificamos ideias de forma a ter uma aplicação mais coesa, simples e fácil de manusear e alterar num futuro próximo.

## 4.3 Arquitetura

Uma vez que todas as funcionalidades do sistema necessitam que os utilizadores se encontrem autenticados, foi importante incorporar **Vuex** na aplicação. O **Vuex** permite armazenar o estado da autenticação e aplicar algumas mutações. Todas as *endpoints* deste serviço foram registadas num *Router* da aplicação que, com auxílio do *state* armazenado em **Vuex**, permite o barramento das páginas e informação que os utilizadores pretendem aceder. Caso não se encontrem autenticados, ou a sessão tenha expirado, serão sempre, automaticamente, redireccionados para a página de *login* explicada e demonstrada anteriormente (figura 4.4).

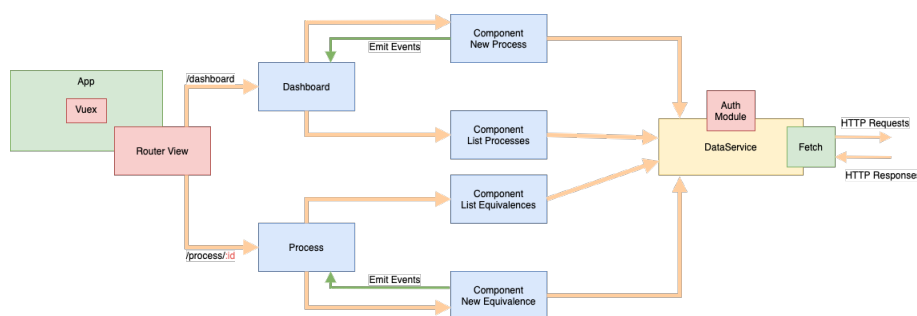


Figura 4.13: Arquitetura do Cliente

# Capítulo 5

## Deployment

Os diversos micro-serviços podem ser iniciados de diferentes formas. Uma vez que ambos têm como base um servidor baseado em **NodeJS**, é aconselhável a instalação da mais recente versão estável do **NodeJS**. Posto isto, os micro-serviços podem ser iniciados via terminal, um de cada vez, ou recorrendo à tecnologia **Docker**.

Nos sub-capítulos necessários, são explicados os passos de *Deployment* para diferentes formas.

### 5.1 *Deployment* de Serviços Iterativamente

Tal como foi exposto, tratando-se micro-serviços baseados em **NodeJS**, é necessário proceder à instalação do *Software*. Assim que este tenha sido instalado com sucesso, é ainda importante proceder à instalação do sistema de armazenamento de dados *NoSQL*, o **MongoDB** e iniciar o serviço por fim. Instaladas as *frameworks* supracitadas, os micro-serviços *Backend* e *Frontend* podem ser iniciados em diferentes ambientes, tal como *Development* ou *Production*.

#### 5.1.1 Development

Em modo *Development*, o *Backend* é iniciado recorrendo ao **Nodemon**. Quer isto dizer, sempre que são efetuadas alterações ao código fonte, o serviço é reiniciado.

Por defeito, o *Backend* inicia na Porta 3030, mas tal pode ser alterada nas variáveis ambiente - discutidas no sub-capítulo "Variáveis Ambiente" na página 29.

Para iniciar o serviço desta forma, devem ser executados os seguintes comandos:

---

```
1 $ npm install
2 $ npm test
```

---

Tal como o *Backend*, é também possível iniciar o serviço *Frontend* em modo *Development*, reiniciando igualmente sempre que são aplicadas alterações ao código fonte da aplicação.

Por defeito, a Porta atribuída para o micro-serviço *Frontend*, será a porta 8080. Contudo, se esta não estiver disponível, será alterada automaticamente para a próxima Porta disponível e comunicada ao utilizador na sessão do terminal.

Para serviço *Frontend* em modo *Development*, devem ser executados os seguintes comandos:

---

```
1 $ npm install
2 $ npm run serve
```

---

### 5.1.2 Production

Em modo *Production*, a aplicação será otimizada para correr com a finalidade de ser usada em contexto real. Querendo isto dizer, sempre que forem aplicadas alterações ao código fonte, ambos os servidores não irão reiniciar, no caso do *Backend* os *logs* da consola serão escondidos e no *Frontend* a aplicação será compilada.

Como tal, o *Backend* pode ser iniciado da seguinte forma:

---

```
1 $ npm install
2 $ npm start
```

---

E o *Frontend* da seguinte maneira:

---

```
1 $ npm install
2 $ npm run build
```

---

## 5.2 Virtualização Utilizando Docker

É ainda possível instalar todos os serviços de forma mais ágil e rápida recorrendo à tecnologia de orquestração de serviços em **Docker**. Para que tal seja possível, é necessário a instalação da aplicação disponível no *website* oficial do **Docker**.

Instalada a aplicação, basta introduzir o seguinte comando, na raiz do projecto do Git, onde se encontra o ficheiro **Docker Compose**.

---

```
1 $ docker-compose up --build
```

---

O comando supracitado deverá obter imagens do **NodeJS** e **MongoDB**, pelo que não será necessária a instalação prévia do *Software*. Aquando a instalação do **MongoDB**, serão ainda importados os dados contidos na pasta *Data* para as respectivas *Collections*: *Universidades* e *Users*. Por defeito, será criado um utilizador cujo o *username* é *admin* e a *password* *admin*.

## 5.3 Variáveis de Ambiente

De forma a que todos os serviços sejam instalados dependendo do contexto e recursos disponíveis, todos os serviços suportam variáveis de ambiente, contidas no ficheiro *.dotenv* e, para o sistema orquestrado em **Docker**, na pasta *Environment*. Todas elas poderão ser alteradas conforme as preferências, contudo é importante que nenhuma seja eliminada para o correcto funcionamento dos serviços.

### 5.3.1 Backend

No excerto de código que se segue, apresenta-se as variáveis de ambiente necessárias para o correcto funcionamento da aplicação. Em comentário, o contexto em que cada um se insere e um breve exemplo.

---

```
1 APP_PORT= # Listening TCP binding Port, exp.: 7332
2 API_URL= # API URL, for a 3030 Port - http://localhost:7332/api
3 MONGO_CONNECTION= # MongoDB Connection String
4 AUTH_SECRET= # JWT and Encryption Secret, exp.: anticonstitucionalissimamente
5 AUTH_TOKEN_TIMETOLIVE= # Cookie maximum lifetime span, exp.: 1h
6 AUTH_TOKEN_ALGORITHM= # Encryption algorithm, exp.: RS256
7 FRONTEND_SERVER= # Frontend URL for CORS middleware, exp.: http://localhost:8080
8 # Course details for PDF outputs
9 COURSE_DEGREE="Mestrado Integrado"
10 COURSE_NAME="Engenharia Informatica"
```

---

É de extrema necessidade que o URL de *FRONTEND\_SERVER* coincida com o URL da aplicação **VueJS** para que os pedidos sejam aprovados pela função *middleware CORS*.

As variáveis *COURSE\_DEGREE* e *COURSE\_NAME* serão consultadas aquando a geração de documentação e atribuição de equivalências, sendo desta forma possível estender o *Software* a qualquer curso.

Por defeito, se for optado pelo *Deployment* em **Docker**, o *Backend* encontra-se a responder na porta 7332.

### 5.3.2 VueJS

No micro-serviço, por imposição da *Framework*, todas as variáveis de ambiente necessitam de iniciar como *VUE\_APP*. Ao contrário do *Backend*, a aplicação Cliente possui apenas uma variável local, para que seja possível efetuar pedidos ao serviço da API. Desta forma, o URL deve coincidir com o URL atribuído ao *Backend*

---

1 VUE\_APP\_API\_SERVER=http://127.0.0.1:7332

---

Por defeito, se for optado pelo *Deployment* em **Docker**, o *Backend* encontra-se a responder na porta 7333.



# Capítulo 6

## Conclusão

No início do trabalho prático começou-se por identificar todos os micro-serviços, como estes se deveriam dispor na arquitectura e como estes interagem entre si. Desta forma, surge uma arquitectura final, onde se pode ver que se contornou um dos principais problemas das arquitecturas de micro-serviços, a dependência de uma única base de dados, o que não acontece aqui dada a existência de uma base de dados para cada serviço.

Em seguida passou-se ao desenvolvimento dos vários serviços e componentes da arquitectura onde foram sentidas as principais dificuldades. Estas surgem, sobretudo, na utilização de novas ferramentas, **Vue.js** e as suas variadas *frameworks* que utilizamos por cima. Com estas, não existia qualquer tipo de experiência prévia e, por isso, houve a necessidade de perceber o funcionamento geral das mesmas. Quanto à interface do sistema, o objectivo principal era que esta fosse simples e de fácil aprendizagem para que o utilizador final não tenha de perder muito tempo a familiarizar-se com o sistema. Surgiram também algumas dificuldades no que toca ao *deployment* através do *Docker*, isto é, a existência de vários micro-serviços tornou mais complicada esta actividade uma vez que existem várias dependências entre eles e era necessário garantir que estes conseguiam comunicar entre si, e só iniciam após os serviços de que dependem estarem ativos e operacionais.