

UNIVERSIDADE DO MINHO

PROCESSAMENTO E REPRESENTAÇÃO DE INFORMAÇÃO
GRAMÁTICAS NA COMPREENSÃO DE SOFTWARE

Informatics Social Network

Bruno Sousa - A78997

Rafael Silva - A74264

Ricardo Pereira - A77045

30 de Janeiro de 2020

Conteúdo

1	Introdução	3
2	Gramática	4
2.1	Propósito	4
2.2	Ficheiro de <i>Input</i>	4
2.3	<i>Lexer</i>	5
2.4	<i>Parser</i>	6
2.5	<i>Visitor</i>	7
2.6	Resultado	8
3	Aplicação Web	9
3.1	Arquitetura da aplicação	9
3.2	Funcionalidades	9
3.3	Dependências	10
3.4	Modelos	10
3.4.1	Evento	10
3.4.2	Publicação	11
3.4.3	Utilizador	11
3.5	<i>Controllers</i>	12
3.5.1	Evento	12
3.5.2	Publicação	12
3.5.3	Utilizador	12
3.6	Rotas	13
3.6.1	Rotas da API	13
3.6.2	Rotas do <i>Front-End</i>	13
3.7	Views	13
4	Conclusão	17

Lista de Figuras

2.1	Formulário para Registo	5
2.2	<i>Lexer</i>	5
2.3	<i>Parser</i>	6
2.4	<i>Visitor</i>	7
2.5	Exemplo	8
2.6	Resultado Final	8
3.1	Homepage sem autenticação	14
3.2	Página de registo de utilizador	14
3.3	Página de Login de utilizador	15
3.4	Homepage de aluno autenticado	15
3.5	Página de perfil de docente	16
3.6	Página de registo de publicação	16

Capítulo 1

Introdução

O presente relatório visa descrever o desenvolvimento do trabalho prático do perfil de Processamento de Linguagens e de Conhecimento, inserido no 4º ano do Mestrado Integrado de Engenharia Informática da Universidade do Minho. Este trabalho é conjunto das UC's de Processamento e Representação de Informação e de Gramáticas na Compreensão de Software. Assim sendo, iremos abordar as opções tomadas em todos os momentos, fazendo referência ao conteúdo estudado em cada uma das Unidades Curriculares ao longo do semestre. Neste TP foi-nos proposto o desenvolvimento de uma aplicação web, idêntica a uma rede social como o *Twitter*, por exemplo. Para isso utilizamos *NodeJS* para desenvolver a parte web e *ANTLR* para desenvolver a gramática inicial.

Capítulo 2

Gramática

Neste capítulo explicamos como criamos a gramática, e qual o seu propósito.

2.1 Propósito

A aplicação desenvolvida dá ao utilizador duas formas de se registar na mesma. Tem a opção de preencher um formulário online disponibilizado na aplicação. Por outro lado tem a opção de fazer o *download* de um ficheiro de texto que poderá preencher com os seus dados e posteriormente fazer o *upload* do mesmo na aplicação e assim concluir o seu registo. Para que fosse possível então o registo do utilizador através do *upload* de um ficheiro seria necessário efetuar o *parsing* desse mesmo ficheiro e a sua transformação num documento *JSON* para ser inserido na base de dados. Para isso, utilizamos o ANTLR e linguagem *Java* para o desenvolvimento de uma gramática que fosse capaz, através do padrão *Visitor*, de receber o ficheiro de texto preenchido pelo utilizador com os seus dados e devolver os dados em formato *JSON*.

2.2 Ficheiro de *Input*

O ficheiro de input terá as instruções necessárias para que o utilizador preencha corretamente os campos que lhe são apresentados. O ficheiro segue a seguinte estrutura:

```

----- Registo -----

Para efetuar o registo devidamente, preencha os campos substituindo os espacos para o efeito.
No final faca upload do ficheiro na plataforma.

----- Registo -----

Nome: _____
Email: _____
Cidade: _____
Curso: _____
Password: _____
Tipo de Utilizador (aluno ou docente): _____
Genero (masculino ou feminino): _____

```

Figura 2.1: Formulário para Registo

2.3 *Lexer*

A gramática capaz de reconhecer os simbolos terminais da gramática é o *RegisterLexer* e podemos ver na seguinte figura a sua estrutura:

```

lexer grammar RegisterLexer;

BLOCK      :  [\-]+ ' '[a-zA-Z]+' ' [\-]+          ;
TEXT       :  [^\-]                                ;
NOME       :  ([a-zA-Z]+)(' '[a-zA-Z]+)*           ;
EMAIL      :  ([a-zA-Z]|[\-_.]|[\0-9])+[@]([a-z]+)[.]( [a-z]+) ;
PASSWORD   :  ([a-zA-Z]|[\-_@#!*]|[\0-9])+         ;
ENUNCIADO  :  ([a-zA-Z]|' '|[(,')+[:]              ;
WS         :  [ \r\n\t] -> skip

```

Figura 2.2: *Lexer*

Aqui podemos observar as expressões regulares para reconhecimento de texto no ficheiro de input. O símbolo *BLOCK* identifica as linhas que delimitam o Cabeçalho do formulário e o símbolo *TEXT* apanha o texto do Cabeçalho. O símbolo *NOME* é usado para identificar os seguintes campos preenchidos pelo utilizador: Nome, Cidade, Curso, Tipo de Utilizador e Género. Uma vez que todos seguem a mesma estrutura, é usado o mesmo símbolo para os identificar.

2.4 *Parser*

A gramática que contém as derivações dos símbolos não terminais apenas estrutura e espelha a forma como a informação aparece no ficheiro que será recebido. O axioma da gramática é *registo* e este deriva nos símbolos não terminais: *cabecalho* e *info*. O *cabecalho* identifica a parte do ficheiro em que estão contidas as informações de preenchimento do mesmo. O *info* identifica a parte do formulário onde estão as informações do utilizador. Tanto o *cabecalho* como a *info* derivam de seguida em símbolos terminais que já foram explicados na seção anterior. Podemos então observar este *parser* na figura seguinte:

```
parser grammar RegisterParser;  
  
options {  
    tokenVocab=RegisterLexer;  
}  
  
registo  
    : cabecalho info  
    ;  
  
cabecalho  
    : BLOCK TEXT BLOCK  
    ;  
  
info  
    : ENUNCIADO NOME ENUNCIADO EMAIL ENUNCIADO NOME ENUNCIADO NOME ENUNCIADO PASSWORD ENUNCIADO NOME ENUNCIADO NOME  
    ;
```

Figura 2.3: *Parser*

2.5 Visitor

Como dito anteriormente foi utilizado o padrão *Visitor* para desencadear ações na gramática. O objetivo seria então a criação de um ficheiro com os campos do utilizador em formato *JSON*. Para isso, apenas precisamos da informação que obtemos ao visitar o símbolo não terminal *info*. Uma vez no reconhecimento deste símbolo, vamos concatenando os dados do utilizador no formato pretendido. Podemos ver na imagem seguinte o *Visitor*:

```
public class Visitor extends RegisterParserBaseVisitor<Integer>{
    private String res;

    public Visitor() { this.res = "{ \"nome\": \" \" ; } }

    public String getJson() { return this.res; }

    @Override
    public Integer visitRegisto(RegisterParser.RegistoContext ctx){
        return visit(ctx.info());
    }

    @Override
    public Integer visitInfo(RegisterParser.InfoContext ctx) {
        this.res += "\"" + ctx.NOME( 0 ) + "\", \"email\": ";
        this.res += "\"" + ctx.EMAIL() + "\", \"cidade\": ";
        this.res += "\"" + ctx.NOME( 1 ) + "\", \"curso\": ";
        this.res += "\"" + ctx.NOME( 2 ) + "\", \"password\": ";
        this.res += "\"" + ctx.PASSWORD() + "\", \"tipo\": ";
        this.res += "\"" + ctx.NOME( 3 ) + "\", \"genero\": ";
        this.res += "\"" + ctx.NOME( 4 ) + "\" }";

        return 1;
    }
}
```

Figura 2.4: *Visitor*

2.6 Resultado

Para que pudéssemos testar a gramática e verificar que realmente satisfaz o seu propósito realizamos alguns exemplos de teste e apresentamos de seguida um deles para ilustrar os resultados finais. Podemos ver na imagem seguinte o formulário devidamente preenchido e pronto a ser consumido pela gramática:

```
----- Registo -----  
  
Para efetuar o registo devidamente, preencha os campos substituindo os espaços para o efeito.  
No final faça upload do ficheiro na plataforma.  
  
----- Registo -----  
  
Nome: Ricardo Pereira  
  
Email: ricardo@hotmail.com  
  
Cidade: Braga  
  
Curso: MIEI  
  
Password: sdf*A342-@Dad!#  
  
Tipo de Utilizador (aluno ou docente): aluno  
  
Genero (masculino ou feminino): masculino
```

Figura 2.5: Exemplo

O resultado seria então o pretendido, o *JSON* com os dados do utilizador pronto a ser inserido na base de dados:

```
{ "nome": "Ricardo Pereira", "email": "ricardo@hotmail.com", "cidade": "Braga", "curso": "MIEI", "password": "sdf*A342-@Dad!#", "tipo": "aluno", "genero": "masculino" }  
  
Process finished with exit code 0
```

Figura 2.6: Resultado Final

Capítulo 3

Aplicação Web

3.1 Arquitetura da aplicação

Como já foi dito anteriormente, a nossa app foi desenvolvida em *NodeJS* sendo que optamos por utilizar o *MongoDB* para armazenar os dados. Fizemos uso do *Mongoose* para estabelecer a conexão entre o *NodeJS* e o *MongoDB*. A aplicação está organizada segundo o padrão do *NodeJS*, com modelos, respetivos controladores, rotas e *views*.

Nos modelos estão definidos os esquemas das coleções presentes na base de dados, ao passo que nos controladores desenvolvemos as *queries* que vamos enviar à base de dados para obter a informação que pretendemos em cada momento. Com as rotas, fizemos uma divisão entre as que interagem diretamente com os controladores e as que interagem com as *views*. Assim sendo, temos as rotas da API e as rotas responsáveis por fazer a ponte entre o *Back-End* e *Front-End*. Por fim, temos as *Views* onde definimos todas as páginas Web que vamos disponibilizar na nossa aplicação.

3.2 Funcionalidades

A nossa aplicação foi desenvolvida tendo por base dois tipos de utilizadores: o utilizador dito "normal" que vai usufruir das funcionalidades da aplicação e o admin, que gere e tem total controlo sobre toda a aplicação.

Assim sendo, como utilizadores normais temos alunos e docentes. Esta distinção é importante fazer para definir a visibilidade de cada publicação ou evento inserido na aplicação. Posto isto, temos 4 tipos de visibilidade. '0' significa que a publicação/e-

vento é pública e qualquer pessoa pode ter acesso a ela, mesmo sem estar registado e autenticado na aplicação. '1' corresponde a conteúdos visíveis apenas para alunos, '2' apenas para docentes e '3' para ambos. A diferença entre a visibilidade '0' e a '3' reside no facto de, ao contrário da '0', na '3' é necessário que o utilizador esteja registado e autenticado para poder visualizar o conteúdo da publicação/evento.

Posto isto, os alunos e docentes podem registar eventos ou publicações, adicionar ficheiros a ambos e ainda atualizar ou remover eventos ou publicações criados por si. Para todas as publicações a que tem acesso, este tipo de utilizador pode ainda colocar ou remover o gosto ou um comentário das publicações. No que diz respeito aos eventos, pode responder se vai ou não ao mesmo. Por fim, pode ainda fazer uma pesquisa de publicações ou eventos, podendo aplicar diversos filtros. Essas filtrações são feitas por título, data, visibilidade, UC (apenas em eventos), curso e tipo. Estes utilizadores têm também a possibilidade de atualizar a foto e a informação contida no seu perfil e fazer a exportação dos seus dados.

No caso do admin, este tem total controlo sobre a aplicação. Com isto, pode ativar e desativar utilizadores que estejam registados e ver todas as publicações e eventos que foram inseridas, tendo ainda capacidade para remover qualquer conteúdo que achar conveniente.

3.3 Dependências

Todas as dependências necessárias para o funcionamento da aplicação estão contidas no ficheiro *package.json*. De entre todas elas, destacamos cinco das que nos parecem ser mais importantes. O *antlr4* foi usado para converter a gramática para *javascript*; o *axios* permite-nos estabelecer a ligação entre a API de dados do *back-end* e as páginas web do *front-end*; o *bcrypt* foi utilizado na encriptação das passwords guardadas na base de dados; e, por fim, o *mongoose* que estabelece a conexão com o *MongoDB*.

3.4 Modelos

3.4.1 Evento

O modelo do evento é constituído por um esquema principal (*EventoSchema*) com 13 campos, sendo que dois deles são sub-esquemas do esquema principal (*UserSchema* e *FicheiroSchema*). Começando pelo principal, é formado por tipo, título, data, local,

descrição, UC, duração, hora, email do utilizador, ID do utilizador, anexos (*FicheiroSchema*), visibilidade e utilizadores que vão ao evento (*UserSchema*). De todos estes atributos, apenas UC, duração, anexos e utilizadores não são obrigatórios. Focando agora nos sub-esquemas, no *FicheiroSchema* temos 5 campos, todos eles obrigatórios. São eles data, name, *path* (caminho para o ficheiro), *mimetype* e *size* (tamanho do ficheiro). Por fim, o *UserSchema* é composto por email e id de utilizador, obrigatórios e por um id.

3.4.2 Publicação

Tal como no modelo do Evento, o da Publicação é composto por um esquema principal (*PublicaçãoSchema*) com 10 atributos, onde dois deles são sub-esquemas. Destes sub-esquemas, um é o *FicheiroSchema* já detalhado anteriormente. Assim sendo, os 10 campos que compõe a publicação são o email e o id do utilizador, título, curso, data, descrição, gostos na publicação, comentários (*ComentarioSchema*), anexos (*FicheiroSchema*) e visibilidade. De todos estes atributos, apenas o número de gostos, comentários e anexos não são exigidos obrigatoriamente. O esquema dos comentários é composto 4 atributos, todos eles obrigatórios. O id e email do utilizador, a descrição, que é o texto do comentário, e a data.

3.4.3 Utilizador

No modelo do utilizador utilizamos 2 esquemas para guardar a informação do perfil de cada utilizador. Temos o esquema principal (*UtilizadorSchema*) e, mais uma vez, o sub-esquema dos Ficheiros que surge nos dois modelos anteriores. Quanto ao principal, surgem 10 campos. Nome, curso, local, email, tipo de utilizador, género e *password*, todos obrigatórios, e o campo ativo, fotografia e anexos, facultativos. Estes dois últimos são identificados pelo sub-esquema dos Ficheiros, sendo que nos anexos podemos ter uma lista de vários ficheiros, enquanto na fotografia é permitido apenas um. É ainda importante referir que, neste modelo do utilizador, utilizamos o *Bcrypt* para fazer a validação da *password* do utilizador.

3.5 *Controllers*

3.5.1 Evento

No controlador do evento temos várias funções que interagem com a base de dados, fazendo uso do modelo correspondente. Aqui temos funções de validação, listagem, atualização, inserção ou remoção de eventos. Passando a descrever as mais importantes, começamos pela de validação (*validateEvento*) que testa se todos os campos obrigatórios foram corretamente preenchidos. De seguida temos várias funções de listagem, utilizando todas elas diferentes critérios de procura, como por exemplo a função *listarAlunos* que apresenta a listagem de todos os eventos disponíveis para os alunos. Por fim, temos as funções de inserção (*inserir*) para inserir um novo evento, de remoção (*remover*) para remover um evento e de atualização de dados (*addAnexos*, *addUtilizador* e *removeUtilizador*) para fazer update da lista de anexos e adicionar ou remover utilizadores de um determinado evento, respetivamente.

3.5.2 Publicação

Tal como o controlador dos eventos, o das publicações foi contruído de forma idêntica. Temos portanto uma função de validação dos dados (*validatePublicacao*) seguida de várias funções de listagem com diferentes filtros, consoante o caso a que serão aplicadas. Por fim, temos as tais funções de manipulação de dados. Para atualizar os dados de uma publicação existente, temos as funções *addGosto* e *addComentario*, para adicionar um gosto ou um comentário a determinada publicação. Em sentido inverso surgem as funções *removeLike* e *removeComentario* que fazem o oposto das anteriores. Por fim, temos ainda a função *addAnexos* que atualiza a lista de anexos de uma publicação. Para finalizar temos as funções *inserir* e *remover* para, respetivamente, inserir uma nova publicação ou remover uma outra já existente.

3.5.3 Utilizador

De forma análoga aos dois controladores já especificados previamente, surge o *controller* dos utilizadores. Contém também ele uma função de validação de dados mas, ao contrário dos outros dois, tem menos funções de listagem e mais de atualização de dados. Nas listagens aparecem, portanto, apenas 3 funções. Uma para listar todos os utilizadores da base de dados (*listar*), uma para consultar a informação de um determinado utilizador através do email (*consultar*) e, por fim, uma listagem de utilizadores de acordo com o tipo especificado (*listarUtilizadorTipo*). No que diz respeito à inserção,

temos 3 funções para inserir utilizadores. *Inserir*, que insere um utilizador normal. *InserirNoFotografia* para inserir um utilizador através de um ficheiro e *InserirAdmin* que insere um administrador. Em sentido contrário, temos a função *remove* que elimina um determinado utilizador da base de dados. Para finalizar este controlador, surgem mais 3 funções para fazer atualização de dados. A *atualizaEstado* que dá *update* ao estado do utilizador, *renamePathImageProfile* para atualizar o caminho para a foto de perfil do utilizador e a função *atualiza* que procura um determinado utilizador e atualiza toda a sua informação.

3.6 Rotas

3.6.1 Rotas da API

Estas são as rotas que fazem a ligação entre a aplicação e a base de dados. Estas rotas invocam os controladores para fazer uso das *queries* necessárias para obter a informação da BD e transportam essa informação para o *Front-End*. Neste capítulo temos 3 rotas: evento, publicação e utilizador. Como é fácil perceber, cada uma destas rotas interage com o respetivo controlador para obter a informação da coleção correspondente.

3.6.2 Rotas do *Front-End*

Aqui estão as rotas que são invocadas pelas páginas, fazendo a ponte entre os dados obtidos pela API e *Front-End*. Ou seja, é através destas rotas que é feita a ligação que permite mostrar a informação da base de dados nas páginas Web da aplicação. Neste caso temos 4 rotas distintas: admin, aluno, docente e index. As três primeiras são invocadas consoante o tipo do utilizador autenticado. A última é responsável por toda página inicial da aplicação.

3.7 Views

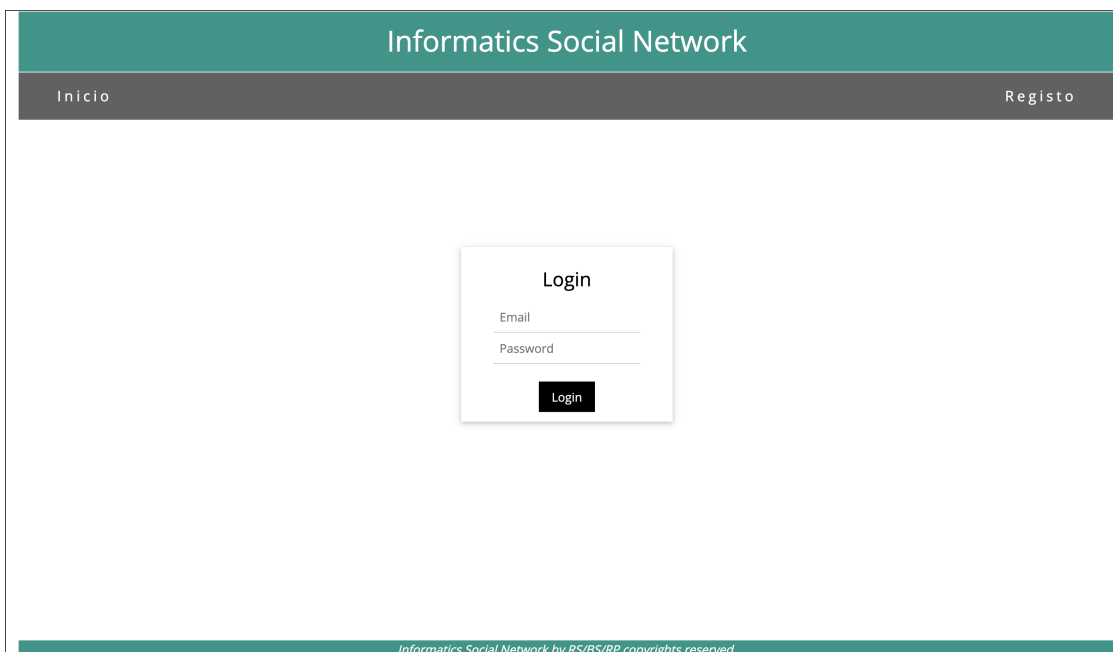
Nesta secção vamos apresentar o *design* da nossa aplicação, ilustrando com alguns *print's* de páginas que podem ser consultadas através da aplicação.



Figura 3.1: Homepage sem autenticação

The screenshot shows the user registration page of the Informatics Social Network. At the top, there is a teal header with the text "Informatics Social Network". Below this is a dark grey navigation bar with links: "Inicio" and "Login". The main content area has a grey header with "Registo de Utilizador". Below this, there is a form with the following fields: "Nome", "Email", "Cidade", "Curso", and "Password". There are two dropdown menus for "Tipo Utilizador" (Aluno, Docente) and "Género" (Feminino, Masculino). There is a section for "Foto de perfil" with a button "Escolher ficheiro" and the text "Nenhum ficheiro selecionado". There is a "Registar" button. There is a section for "Registo via ficheiro" with a button "Escolher ficheiro" and the text "Nenhum ficheiro selecionado". There is a "Link do formulário" link. At the bottom, there is a teal footer with the text "Informatics Social Network by RS/BS/RP copyrights reserved."

Figura 3.2: Página de registo de utilizador



Informatics Social Network

Inicio Registo

Login

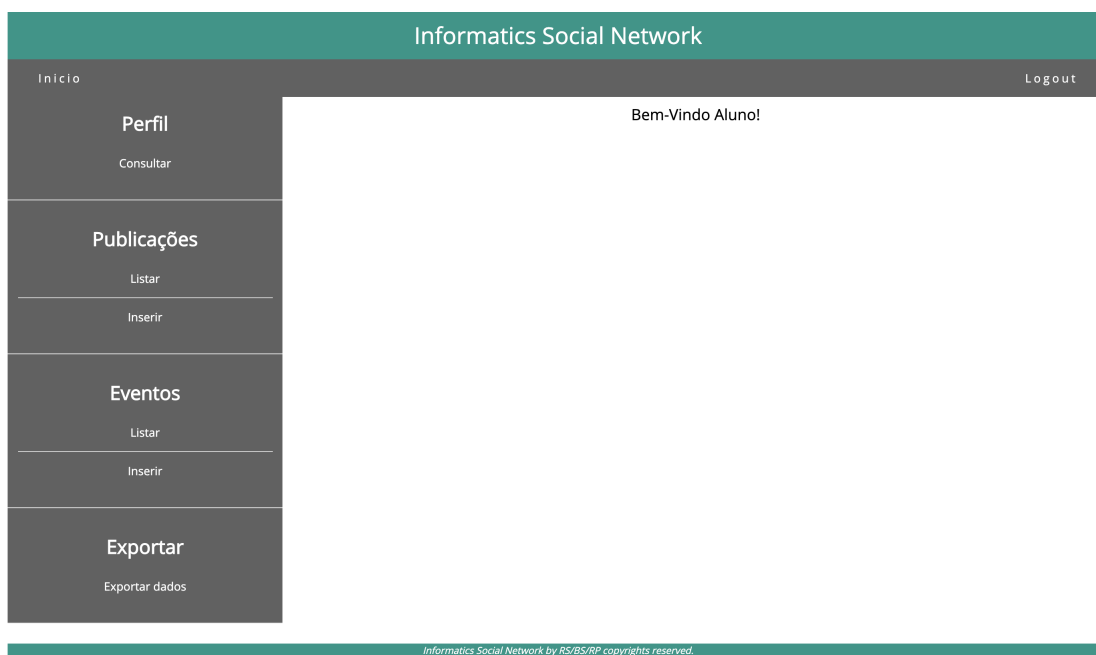
Email

Password

Login

Informatics Social Network by RS/BS/RP copyrights reserved.

Figura 3.3: Página de Login de utilizador



Informatics Social Network

Inicio Logout

Bem-Vindo Aluno!

Perfil

Consultar

Publicações

Listar

Inserir

Eventos

Listar

Inserir

Exportar

Exportar dados

Informatics Social Network by RS/BS/RP copyrights reserved.

Figura 3.4: Homepage de aluno autenticado

Perfil Individual

Foto de perfil

Escolher ficheiro Nenhum ficheiro selecionado

Atualizar foto de perfil

jcr@gmail.com

Jose Ramalho

LEI

Braga

.....

Docente

Masculino

Atualizar

Informatics Social Network by RS/BS/RP copyrights reserved.

Figura 3.5: Página de perfil de docente

Informatics Social Network

Início Logout

Registo de Publicação

Titulo

Curso

Descrição

Visibilidade

Pública ☐ Privada (para alunos) ☐ Alunos e Docentes ☐

Ficheiros

Adicionar ficheiro

Registrar

Informatics Social Network by RS/BS/RP copyrights reserved.

Figura 3.6: Página de registo de publicação

Capítulo 4

Conclusão

Dado por terminado o trabalho efetuado, podemos constatar que a aplicação desenvolvida cumpre com todos os requisitos propostos. Podemos afirmar que a aplicação desenvolvida é fiável, robusta e fluida.

As principais dificuldades encontradas ao longo do desenvolvimento do projecto, foram em primeiro lugar, a gestão do tempo face ao elevado número de funcionalidades que eram preciso ser implementadas. Em segundo lugar, a necessidade de autenticação no sistema de forma a estar devidamente protegido estabeleceu-se como outra barreira a ultrapassar. Por último, a implementação de grupos de utilizadores foi também uma tarefa difícil mas que conseguimos concretizar com sucesso.

Como trabalho futuro, poderíamos utilizar outras ferramentas como *ReactJS* ou *VueJS* para melhorar a interface uma vez que apesar da ferramenta utilizada servir para o efeito, não é a mais apelativa.

Em suma, esta aplicação contém todas as ferramentas e funcionalidades necessárias e bem implementadas para satisfazer utilizadores reais.