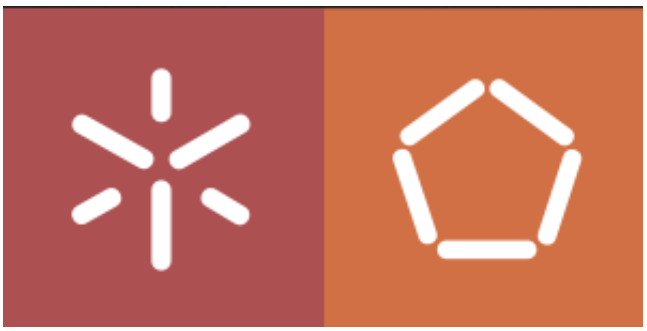

Universidade do Minho

Mestrado em Engenharia Informática

Processamento de Linguagens e Conhecimento

ALUMNIO

Membros da Equipa 3:

João Leal	A75569
João Vieira	A76516
Manuel Monteiro	A74036

31 de Janeiro de 2020

Conteúdo

1 Introdução

Este relatório é referente a uma aplicação Web desenvolvida no âmbito do perfil de Processamento de Linguagens e Conhecimento. A aplicação deve funcionar como uma plataforma com vertente social direcionada a alunos.

O objetivo do trabalho passou por desenvolver uma aplicação, que dentro da temática do projeto, estaria dividida por um servidor com uma API de dados, e uma interface que permitisse fornecer as funcionalidades sobre essa API. Sendo assim, a resolução do nosso grupo passa por implementar em *NodeJS* o servidor *backend*, com o auxílio da ferramenta de processamento de gramáticas *Antlr*. Quanto ao *frontend*, o seu desenvolvimento passou por utilizar uma *framework* de interfaces reativas com *Vue.js*.

As próximas alíneas explicam toda a arquitetura e lógica pensada para o funcionamento da aplicação.

2 Opções de Desenvolvimento

Em termos de tecnologias utilizadas o grupo optou por aquelas que foram leccionadas nas aulas. Sendo assim, o *frontend* foi desenvolvido em *Vue.js* com recurso ao *plugin Vuetify*. Já o *backend* foi implementado em *Node.js + Express.js* com persistência de dados em *Mongodb*. Para a gramática foi utilizada a ferramenta *Antlr*.

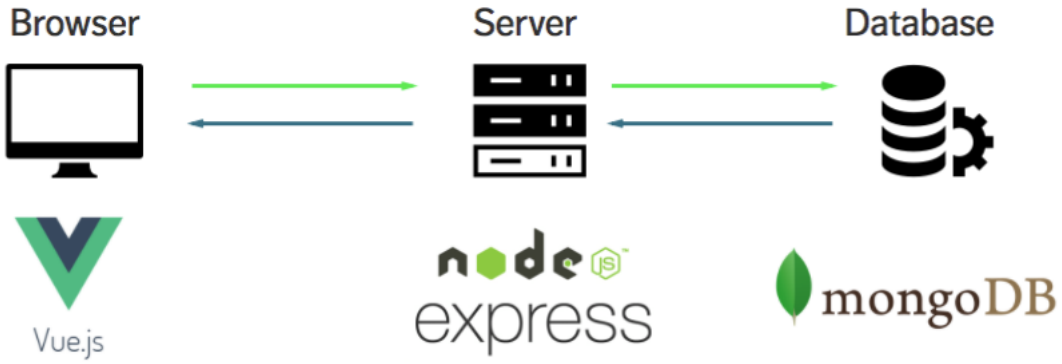

Figura 1: Esquema da Arquitetura da Aplicação

3 Persistência de Dados

A persistência de Dados desta aplicação foi implementada em *MongoDB*. Foram definidos diferentes modelos guardados em diferentes colecções e os respectivos controladores.

3.1 Caracterização dos modelos

- **USERS** - Representa os utilizadores da aplicação
  - name

  - email
  - password
  - feed
  - friends
  - events
  - pending
- **POSTS** - Representa as publicações
  - author
  - text
  - date
  - comments
  - category
- **EVENTS** - Representa os eventos
  - author
  - name
  - description
  - date
  - time
  - local
  - category
- **GROUPS** - Representa os grupos
  - name
  - description
  - admin
  - members
  - events
  - feed
  - pending
- **CHATS** - Representa os *chat's* entre utilizadores
  - members
  - messages
- **MESSAGES** - Representa as mensagens de cada *chat*
  - author
  - content
  - date

4 Autenticação

No que diz respeito à autenticação, utilizamos *Json Web Token* (JWT) com o algoritmo RSA256, que utiliza um par de chaves pública e privada. O servidor tem uma chave privada para gerar a assinatura do *token*, que após autenticação, o cliente recebe e este é sempre validado através de uma chave pública. Quaisquer futuros pedidos por parte do cliente, terão que incluir no *Authorization Header*, o *token* de modo a aceder aos recursos do servidor. O tempo de expiração de um *token* na nossa aplicação é de uma hora, em que após este tempo o cliente tem que voltar a realizar a autenticação.

5 Rotas

Quanto às rotas, foram criados um roteador da API de dados por cada um dos modelos referidos anteriormente, *users*, *posts*, *events* e *groups*. Para assegurarmos a segurança e limitarmos o acesso à nossa API de dados, protegemos todas as rotas exceto a autenticação e o registo, através de uma função que verifica se o *token* enviado nos pedidos é válido.

6 Gramática e Visitor

No âmbito da U.C. de Gramáticas na Compreensão de Software, foi nos proposto integrar neste projeto um método de inserção de utilizadores através da temática estudada. O objetivo passava por fornecer um ficheiro de *input* que seria posteriormente processado por uma gramática em ANTLR, e retornava ao servidor da aplicação esses utilizadores de modo a serem inseridos na nossa base de dados.

6.1 Gramática ANTLR

Como tal, desenvolvemos uma simples gramática que tendo tem conta um ou mais utilizadores fornecidos no *input*, processa a informação pessoal de cada um.

Essa informação pessoal consiste no nome, email, *password*, biografia, curso e ano curricular que frequenta. Cada um destes atributos é precedido de um terminal em texto que identifica qual dos atributos se trata. Os restantes símbolos terminais da gramática, processados lexicamente por expressões regulares, são STR, NUM, EMAIL, PASS e WS. O símbolo STR identifica uma palavra e o NUM um número, o EMAIL identifica a estrutura de um endereço de email e o PASS processa uma *password*, e o último WS apenas serve para ignorar espaços, *tabs* e mudanças de linha.

```
grammar users;

//Lexer
STR : [a-zA-Z]+;
NUM: [0-9]+;
EMAIL: ([a-zA-Z0-9._-]+[@][a-zA-Z0-9._-]+[.][a-zA-Z]+);
```

```
PASS: [a-zA-Z0-9._-]+;
WS : [ \t\n\r]+ -> skip ;

//Parser
users : user+
      ;

user : 'NAME:' name 'EMAIL:' email 'PASSWORD:' password 'BIOGRAPHY:'
biography 'COURSE:' course 'YEAR:' year
     ;

name : STR+
     ;

email : EMAIL
      ;

password : PASS
         ;

biography : STR+
          ;

course : STR+
       ;

year : NUM
     ;
```

6.2 Linguagem e Visitor

Após explicitar como é processado o ficheiro de utilizadores com a nossa gramática, apresentamos a estrutura desse mesmo *input* com dois utilizadores, a título de exemplo:

```
NAME: Joao Leal
EMAIL: a75569@alunos.uminho.pt
PASSWORD: 123pass
BIOGRAPHY: Gosto muito de programar
COURSE: Mestrado Integrado em Engenharia Informatica
YEAR: 4

NAME: Joao Vieira
EMAIL: a76516@alunos.uminho.pt
PASSWORD: qwerty123
BIOGRAPHY: Gosto de fotografia e viajar
COURSE: Mestrado Integrado em Engenharia Informatica
YEAR: 4
```

De seguida, de modo a realizarmos as ações semânticas necessárias e traduzir o *input* para o tipo de ficheiro pretendido, em JSON, foi criado um *Visitor*. O nosso *usersVisitor* percorre cada regra da gramática e imprime em sintaxe JSON as informações de cada contexto diferente. Apresentamos a seguir os *visitor's* mais relevantes:

```
// Visit a parse tree produced by usersParser#users.
usersVisitor.prototype.visitUsers = function(ctx) {

  let code = ''
  for (let i = 0; i < ctx.getChildCount(); i++) {
    if(i==ctx.getChildCount()-1)
      code += '{'+this.visit(ctx.getChild(i))+'}';
    else
      code += '{'+this.visit(ctx.getChild(i))+'},';
  }
  code = code.replace(/:,/g,'":')

  return '['+code+']';
};


// Visit a parse tree produced by usersParser#user.
usersVisitor.prototype.visitUser = function(ctx) {
  return this.visitChildren(ctx);
};


// Visit a parse tree produced by usersParser#name.
usersVisitor.prototype.visitName = function(ctx) {

  let start = ctx.start.start
  let stop = ctx.stop.stop

  return '"'+ctx.start.getInputStream().getText(start,stop)+'"'
};
```

O *visitUsers* vai iterar todos os utilizadores e faz *visit* a cada um deles, imprimindo os símbolos de JSON. O *visitUser* apenas faz visit ao seu contexto. No fim todos os *visitor's* restantes que são relativos a terminais, apenas retornam esse mesmo símbolo na sintaxe JSON correta, como no exemplo do *visitName*.

O texto retornado em formato JSON pelo *Visitor*, será o seguinte:

```
[{
"name": "Joao Leal",
"email": "a75569@alunos.uminho.pt",
"password": "123pass",
"biography": "Gosto muito de programar",
```

```
"course": "Mestrado Integrado em Engenharia Informatica",
"year": "4"
},
{
"name": "Joao Vieira",
"email": "a76516@alunos.uminho.pt",
"password": "qwerty123",
"biography": "Gosto de fotografia e viajar",
"course": "Mestrado Integrado em Engenharia Informatica",
"year": "4"
}]
```


Para ser realizado o registo destes utilizadores na aplicação, apenas é necessário fazer um **POST** para a rota **/api/users**, em que fornecemos o ficheiro de *input* no *body*.

7 Conclusão

Chegado o fim do desenvolvimento do projeto, podemos afirmar que as competências adquiridas durante as aulas do perfil de PLC, foram melhoradas e permitiu-nos alargar algum deste conhecimento. O trabalho também permitiu que tivessemos uma visão muito mais clara do que é o desenvolvimento e a *stack* de uma aplicação Web, visão essa que não era tão clara durante a licenciatura.

Relativamente aos requisitos propostos, achamos que foram implementados com sucesso em termos de *backend*, mas houve alguma dificuldade para disponibilizar essas funcionalidades na interface em *Vue.js*, devido ao pouco conhecimento do grupo com esta ferramenta. Conseguimos implementar o registo dos utilizadores com uma gramática no servidor *backend*, o que nos permitiu também explorar uma forma diferente de povoamento para a aplicação.

Em suma, o projeto apesar de ter um grau de complexidade um pouco elevado, por ter muitas funcionalidades como solução final, tornou-se um desafio o que nos fez aumentar bastante o conhecimento na área de Processamento de Linguagens e Conhecimento.