Universidade Nova de Lisboa

Mestrado Integrado em Engenharia Informática

Conceção e Implementação de Aplicações para a Internet

CIAI Projecto

Autores:
41772 - Pedro Simão
41626 - André Correia
41915 - Cristiano Martins

December 16, 2016

Índice

1 Introdução

O objetivo deste projeto é desenhar e implementar um aplicação web que consiga gerir cursos, as respetivas cadeiras, os alunos inscritos e toda a informação necessária para que ambos alunos e professores consigam gerir a informação relacionada com o contexto em que se encontram. Para desenvolver a aplicação é necessário estabelecer alguns parâmetros primeiro. Cada curso é descrito pelo seu número total de ECTS, número de anos que demora a ser realizado e um conjunto de cadeiras. Cada cadeira tem um identificador único, um nome, uma descrição e um conjunto de professores. Para além disso, uma cadeira pode ter várias edições, cada edição corresponde a um semestre e a um ano letivo, e diferentes professores em cada edição da cadeira. Um estudante é identificado por um número e tem ainda um conjunto de atributos, como por exemplo, e-mail pessoal, e-mail da faculdade, morada, fotografia, as edições em que se encontra inscrito e as edições concluídas. Cada estudante deve, numa determinada cadeira em que se encontre inscrito, ter notas associadas a avaliações (testes, exames e projetos). Os Professores são identificados por nome de utilizador, e tem um conjunto de atributos, como o email pessoal e uma fotografia, e podem estar associados a várias cadeiras desempenhando diferentes funções: professor ou assistente.

Em relação a funcionalidades necessárias para os professores temos:

- Apresentar toda a informação pessoal do mesmo, assim como possibilitar a alteração de alguns parâmetros.
- Apresentar as edições em que está associado.
- Para uma dada edição deve ser possível ver todas as atividades agendadas, como testes, exames, aulas práticas, assim como possibilitar a alteração, adição e remoção das mesmas.
- Para uma dada edição deve ainda ser possível adicionar e remover alunos, assim como atribuir notas aos mesmos (caso seja professor e não assistente dessa edição).
- Para uma dada edição deve ser possível adicionar novos professores (caso seja professor e não assistente dessa edição)

Para estudantes são necessárias as seguintes funcionalidades:

- Apresentar toda a informação pessoal do mesmo, assim como possibilitar a alteração de alguns parâmetros.
- Apresentar notificações e futuros eventos.
- Apresentar as edições em que se encontra inscrito e as que já concluiu, assim como os ECTS dessa cadeira e a nota final.
- Para edições que já tenha concluído deve ser possível visualizar as notas que o aluno teve em testes, exames e trabalhos.

2 Tecnologias usadas

Neste capitulo apresentamos as ferramentas que foram usadas para o desenvolvimento do projeto. Para além das mencionadas pelos docentes no âmbito da cadeira, decidimos usar outras que nos facilitaram no desenvolvimento do projeto. Em baixo segue uma descrição de cada uma das ferramentas usadas e em que contexto foram usadas.

2.1 *Client-side*

Na implementação do código para o cliente, foram usadas algumas ferramentas para além das recomendadas. Estas ferramentas não têm impacto direto na interface, mas sim no ambiente de desenvolvimento do código.

2.1.1 NPM

NPM é uma ferramenta utilizada para gerir pacotes de Javascript. Estes pacotes podem ser consultados no repositório do NPM. Uma vez instalada a dependência de um pacote no projeto, esta é referenciada num ficheiro *package.json*, previamente criado através do comando *npm init*, executado para definir algumas propriedades do projeto (nome, descrição, autor, dependências). Uma das vantagens desta ferramenta para além da reutilização de código por parte de outros *developers*, é os membros de um projeto poderem instalar as novas dependências através do comando *npm install*.

Algumas das dependências que utilizamos neste projeto:

- **bootstrap**, *framework* de HTML, CSS e Javascript que ajuda no desenvolvimento de sites *responsive*. Disponibiliza um variado leque de classes e componentes já predefinidos.
- **jquery**, biblioteca de Javascript que facilita a manipulação de elementos HTML, animações, gestão de eventos e simplifica o uso de Ajax.
- **react**, biblioteca de Javascript que ajuda a desenvolver interfaces interativas. Permite a implementação de componentes com um estado e uma uma *view* próprios. Sendo alterado o estado de um componente, este encarrega-se de renderizar a *view* dos componentes associados.
- **react-router**, biblioteca de Javascript para gerir rotas. Mantém a interface em sincronização com um URL específico.

2.1.2 Webpack

Webpack é uma ferramenta que avalia todas as dependências entre módulos instalados num projeto (a partir do NPM) e gera um ficheiro estático que representa esses módulos. A vantagem desta ferramenta é que disponibiliza um desenvolvimento por módulos. Podemos criar vários módulos que representam

um determinado componente do site, simplificando a estrutura geral do projeto. Os módulos que foram instalados por NPM também vão ser encupidos no ficheiro gerado. Este ficheiro Javascript será o ponto de entrada no site. Uma vez que instalamos os módulos correspondentes ao bootstrap, react e jquery então deixa de ser necessário importar diretemente estas bibliotecas. Para o webpack funcionar é preciso criar um ficheiro de configuração *webpack.config.js* que contém algumas informações como também o ficheiro de ponto de entrada para o Webpack. A partir do ficheiro de entrada (app.js) o webpack começa por ver os módulos dependentes. Avaliando recursivamente as expressões *require(module name)* presentes nos ficheiros. Para cada dependência o webpack verifica se a mesma existe no caminho especificado.

2.1.3 Gulp

Gulp é uma ferramenta para automação de tarefas. Usamos como complemento ao webpack, mas não sendo por si uma ferramenta importante no contexto deste projeto. Existe apenas uma tarefa designada para o Gulp, executar o webpack, e uma vez compilado o ficheiro pelo webpack, este é copiado para uma pasta destino (/public/resources/assets/js).

2.1.4 Babel

Babel é um compilador de Javascirpt. Uma vez que o código em React não segue as notações puras do Javascript, o Babel compila o código em React para código nativo de Javascript. Utilizamo-lo como módulo instalado através de NPM. O Babel podia ser importado diretamente no browser e compilar o codigo react diretamente no browser, desde que o ficheiro que contém o codigo react seja importado com o atributo *type="text/babel"*. Uma vez que fizemos esta transformação localmente então deixa de ser necessário acrescentar este atributo. Tendo ainda como vantagem o facto de a compilação do código ser feita localmente e não diretamente no browser. Isto resulta também numa redução do tempo de espera de loading da página.

2.1.5 Sass

Syntactically Awesome StyleSheets (SASS) é uma extensão ao CSS que fornece algumas vantagens à linguagem base do CSS. Permite-nos usar variáveis, regras em cadeia e importar outras folhas de estilo. Ajuda também a manter o código das folhas de estilo mais organizado. Como estas folhas de estilo não são interpretadas diretamente pelo browser, é necessário um compilador. Para compilar usamos Compass que instalamos através de NPM. As folhas de estilo são importadas através do javascript como módulos (requeire('stylefile.scss')) e o webpack, uma vez mais, fica encarregue de compilar este código com auxílio do módulo Compass, sendo gerada uma folha de estilo em linguagem CSS básico.

2.2 *Server-side*

Na implementação do código para o servidor foi usada a framework Spring e algumas dependências. Uma vez que se trata de um projeto em *Maven*, as dependências usadas foram as que nos permitiram ter um ambiente simples de uma aplicação inicial em Spring e que nos permitissem realizar uma aplicação para o contexto do projeto.

2.2.1 Spring

Spring é uma framework para implementação de sistemas e aplicações em Java. Utilizamos esta framework para o desenvolvimento do Servidor. As dependências usadas foram as seguintes:

- **Spring Boot Starter Web**, dependência para construir a estrutura base de uma aplicação web.
- **Spring Boot Starter Thymeleaf**, dependência para *view engine*. Possibilita a criação de vistas em HTML com algumas anotações para embutir código, e passar variáveis do servidor que são renderizadas pela vista.
- **Spring Boot Starter Data JPA**, dependência para poder usar JPA com Hibernate. Esta ferramenta é usada para mapear os objetos em Java para objetos relacionais (neste caso em SQL). Esta ferramenta é útil pois abstrai o código SQL, sendo gerado o código de consultas, inserções e atualizações através das anotações usadas nas classes em Java.
- **Mysql Connector**, dependência para ligação à base de dados.
- **Jackson**, dependência que transforma um objeto Java em JSON.
- **Spring Security Web**, dependência usada para camada de segurança.

3 Desenho e Arquitetura

Neste capitulo é apresentado o modelo IMFL e de base de dados. Ambos foram desenhados com o objetivo de fornecer uma visão mais geral do problema. Os componentes e entidades que eram necessários e as relações entre eles.

3.1 Modelo IMFL

No desenvolvimento da interface da aplicação foi feito um modelo usando a ferramenta IFML. Esta ferramenta permite ter uma visão geral dos componentes usados para desenhar a interface assim como as relações entre eles. Embora tenha sido útil fazer este modelo inicialmente para servir de ponto de partida para o desenvolvimento dos componentes em React, ao desenvolver a aplicação algumas alterações foram surgindo.

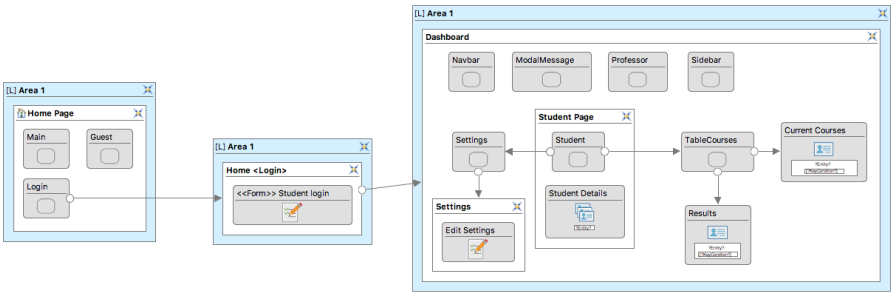

Figure 1: Modelo IFML

3.2 Modelo de Base de Dados

O modelo de base de dados inicialmente foi desenhado pela ferramenta MYSQL-Workbrench, para termos uma noção de que entidades seriam necessárias e as relações entre elas. Uma vez que o modelo estava de acordo com o contexto do projeto, começamos por usar as anotações disponíveis em JPA para conseguir ter como resultado um modelo de base de dados igual ao que tínhamos pensado. Durante o desenvolvimento do projeto foram surgindo algumas alterações, que se mostraram simples de resolver com o uso das anotações disponíveis em JPA.

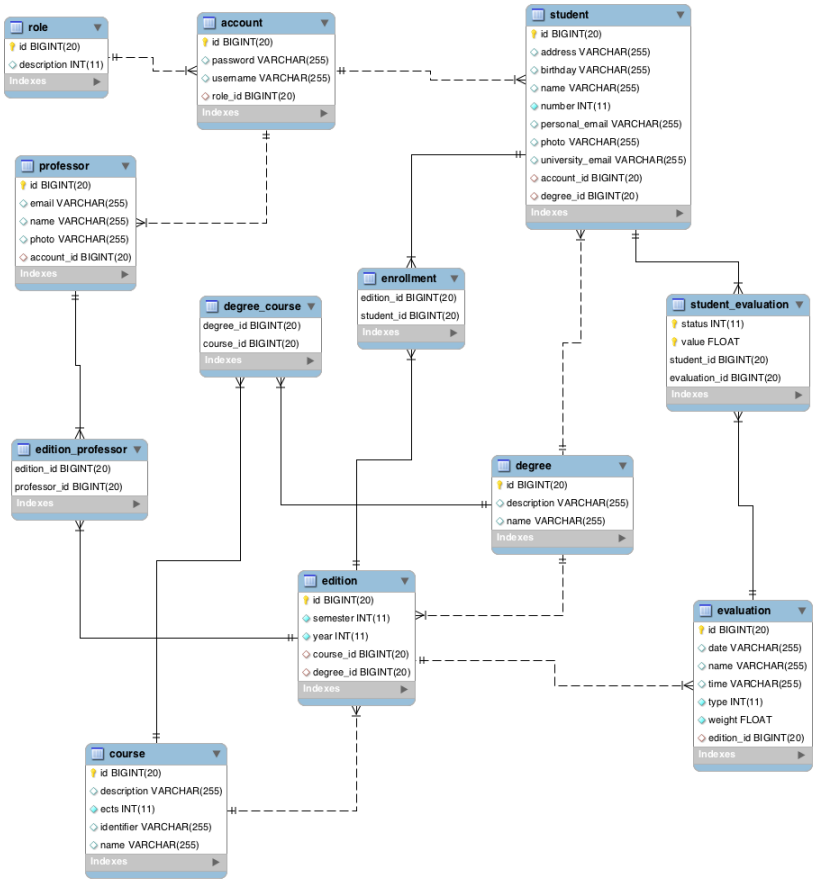

Figure 2: Modelo Base de dados

4 Arquitetura

Neste capitulo apresentamos uma descrição detalhada da arquitetura do nosso sistema. Na implementação do servidor, decidimos separar o máximo possível por componentes. Não só para facilitar na própria organização e estrutura, mas também para o reutilização do código.

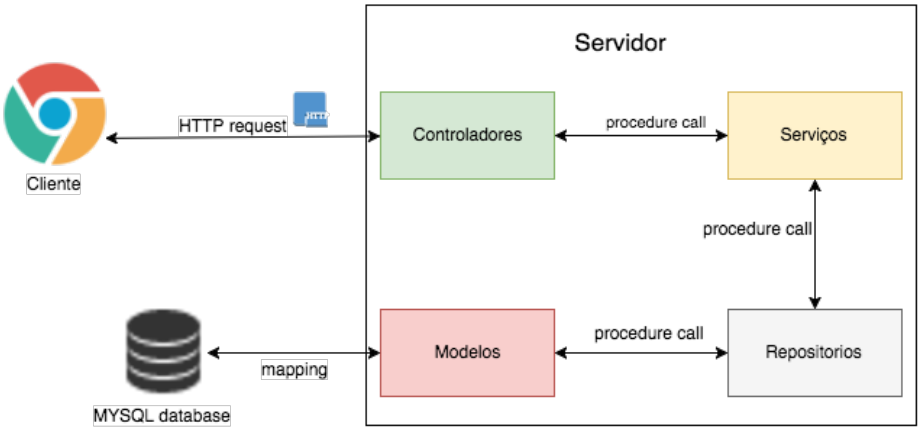

Figure 3: Arquitetura geral do sistema

O servidor está dividido nos seguintes componentes:

- **Serviços** - Os serviços são componentes internos ao servidor. São encarregues por disponibilzar as operações necessárias pelos controladores. No projeto temos vários componentes do tipo serviço. O *AppService* é o componente principal, que serve de ponte entre os controladores e os outros componentes que são necessários. Ou seja sempre que qualquer um dos controladores necessita de alguma operação interna, este chama uma operação previamente definida no *AppService*. Ficando ao cargo deste componente saber que outros serviços executar para dar a resposta pretendida.

- **Controladores** - Os controladores são as classes que se encarregam de fazer a ligação entre o cliente e servidor. Existem vários tipos de controladores disponíveis, cada um encarregue de um contexto em especifico. Os controladores recebem pedidos pelo cliente e tratam de chamar métodos que estão defendidos no componente (*AppService*).

- **Repositórios** - Os repositórios são interfaces com a definição de alguns métodos. Estes componentes são usados pelos serviços sempre que é necessário manipular informação existente nos modelos.

- **Modelos** - Os modelos são as entidades do sistema. É onde estão definidos as propriedades de cada entidade e as relações entre cada uma delas. São nestes componentes que são usadas as anotações de JPA para fazer o mapeamento para o MYSQL.
- **Segurança** - Os componentes de segurança são serviços mas com a responsabilidade de garantir que os recursos não são acedidos por entidades que não têm permissão.

5 Implementação

Neste capitulo descrevemos alguns aspetos que achamos mais relevantes a nível de implementação. Tanto a nível de cliente, como de servidor, foram surgindo alguns desafios e aqui descrevemos alguns deles e a abordagem tomada em cada caso.

5.1 Cliente

No cliente, para além das ferramentas usadas detalhadas em cima, os pontos mais relevantes é a reutilização de componentes. Sempre que algum componente é utilizado para mostrar dados semelhantes, estes foram adaptados de maneira a que possam ser usados em vários contextos. Como exemplo mostramos o componente *TablueCourses*, que é utilizado para mostrar as edições atuais de um aluno, os resultados de um aluno e as edições atuais de um professor.

```
var TableCourses = React.createClass({

  getInitialState:function(){
    return {
      open: false,
      data: null,
      type: null,
      course_data: null
    }
  },

  componentDidUpdate: function(){
    var type = this.props.list_type;

    if(this.props.student != null && this.state.data != null && (type == null ||
        type != this.state.type)){
      if(type == "currentCourses")
        this.getCurrentEdition(this.props.student.number);
      else if(type == "finishedCourses")
        this.getCoursesResults(this.props.student.number);
    } else if(this.props.professor != null && this.state.data != null && (type
        == null || this.state.type)){
      this.getProfessorEditions(this.props.professor.id);
    }

  },

  ....

  getCoursesResults: function(number){
    var self = this;
    $.get('/student/view/'+number+'/results', function(data) {
      self.setState({data:data, type:self.props.list_type});
    });
  },

  getCurrentEdition: function(number){
```

```
    var self = this;
    $.get('/student/view/'+number+'/editions',function(data){
      self.setState({data:data,type:self.props.list_type});
    });
  },

  getProfessorEditions: function(number){
    $.get("/professor/view/"+number+"/editions", function (data) {
      this.setState({data:data});
    }.bind(this));
  },

  ...
}
```

Listing 1: ToubleCourse

5.2 Servidor

No servidor, o desafio maior foi a parte referente à segurança. Como tínhamos um formulário de Login construido em React que executava um POST por ajax, a abordagem foi um pouco diferente da exemplificada nas aulas.

Na classe *SecurityConfig.java* adicionamos algumas operações necessárias e implementamos os métodos de duas classes (AuthenticationSuccessHandler e *AuthenticationFailureHandler*)

A ideia foi definir dois *handlers* para o caso de sucesso ou falha na autenticação. Nesta configuração também são protegidas as rotas consoante se podem ser acedidas por professores, estudantes ou assistentes.

```
 @EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  private UserDetailsService users;
  @Autowired
  AuthenticationFailure customAuthFailure;
  @Autowired
  AuthenticationSuccess customAuthSucc;

  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http
      .csrf().disable()
      .authorizeRequests()
        .antMatchers("/img/**", "/js/**", "/").permitAll()
        .antMatchers("/student/**").hasRole("STUDENT")
        .antMatchers("/edition/**").hasAnyRole("PROFESSOR","STUDENT")
        .antMatchers("/professor/**").hasAnyRole("PROFESSOR","ASSISTANT").and()
      .formLogin()
        .permitAll()
        .loginPage("/login")
        .successHandler(customAuthSucc)
```

```
        .failureHandler(customAuthFailure).and()
      .logout().logoutSuccessUrl("/").and()
      .exceptionHandling().accessDeniedPage("/").and();
  }

  @Autowired
  public void configureGlobal(AuthenticationManagerBuilder auth) throws
       Exception {
    auth.userDetailsService(users).passwordEncoder(new BCryptPasswordEncoder());
  }
}
```

Listing 2: SecurityConfig classe

Nesta classe reimplementamos o método que é chamado em caso da autenticação ser sucedida. Como temos vários tipos de conta, fazer a autenticação não chega, pois embora um utilizador seja autenticado, temos que garantir que está a ser autenticado no local autorizado. Neste método é verificado que a conta existe e se a entidade que fez autenticação é um estudante ou um professor. Caso a conta exista e seja do tipo professor ou estudante então é colocado um parâmetro no *reponse header* com o nome de *redirect*. Este parâmetro é acedido na resposta no lado do cliente para sabermos para que pagina deverá ser reencaminhado depois da autenticação.

```
@Component
public class AuthenticationSuccess extends SimpleUrlAuthenticationSuccessHandler
     {

  @Autowired
  AppService service;

  @Override
  public void onAuthenticationSuccess(HttpServletRequest request,
       HttpServletResponse response,
      Authentication authentication) throws IOException, ServletException {

    String username = request.getParameter("username");
    Account ac = service.getAccountByUsername(username);

    if (ac != null) {
           AccountRole accRole = ac.getRole().getDescription();
           if (accRole.equals(AccountRole.STUDENT)) {
       Student s = service.getStudentByAccount(ac);
       response.addHeader("redirect", "/student/" + s.getNumber());
     } else if (accRole.equals(AccountRole.PROFESSOR) ||
         accRole.equals(AccountRole.ASSISTANT)) {
       Professor p = service.getProfessorByAccount(ac);
       response.addHeader("redirect", "/professor/" + p.getId());
     }
       } else {response.sendError(500, "Account doesn't exist");}
  }
}
```

Listing 3: AuthenticationSuccessHandler interface

Esta classe é o inverso da descrita a cima. Quando a autenticação falha é enviado *status code 500* para o cliente.

```
@Component
public class AuthenticationFailure extends SimpleUrlAuthenticationFailureHandler{

  @Override
  public void onAuthenticationFailure(HttpServletRequest request,
       HttpServletResponse response,
      AuthenticationException exception) throws IOException, ServletException {

    response.sendError(500, "Authentication Failed");
  }

}
```

Listing 4: AuthenticationFailureHandler interface

Foi necessário a implementação destas classes, pois a autenticação usada foi por *request* em ajax, e consoante a resposta do servidor o cliente decide o que fazer. O comportamento de autenticação por formulário acaba por redirecionar automaticamente para uma pagina (de erro ou principal) e não era esse o comportamento desejado.

De notar, que cada entidade (Student, Professor) no nosso modelo de base de dados tem uma conta (Account) e que cada conta está associada a um tipo de role (PROFESSOR, STUDENT, ASSISTANT). Sendo que cada role tem permissões diferentes. Para controlar estas permissões foram criadas anotações aos pedidos nos controladores.

Como por exemplo, um estudante editar a sua informação pessoal. Partimos do princípio que apenas o próprio estudante pode editar as suas informações pessoais.

```
@RequestMapping(value = "/edit/{number}", method = RequestMethod.POST)
  @PreAuthorize("hasRole('STUDENT') and @securityService.isStudentPage(#number)")
  public @ResponseBody ResponseEntity<?> editStudent(@PathVariable int number,
       @RequestParam Map<String, String> requestParams){...}
```

Listing 5: Exemplo de anotação de segurança

Finalmente neste capítulo, existem algumas anotações que achamos relevantes comentar. Como por exemplo, confirmar se a entidade pode aceder a um determinado pedido verificando o seu tipo de conta. Mas também se, mesmo que o tipo de conta seja válido, essa entidade tem responsabilidade sobre o recurso pretendido. Como é o caso de um professor só poder dar notas se a entidade autenticada for do tipo Professor e caso seja professor dessa edição. Estas verificações são feitas na classe *SecurityService*.

```
@Service
public class SecurityService {

  @Autowired
  AppService service;

  public boolean isStudentPage(final int number) {
      User user = (User)
           SecurityContextHolder.getContext().getAuthentication().getPrincipal();
      try {
      Student student = service.getStudentByNumber(number);
      Account a = student.getAccount();
           if (a.getUsername().equals(user.getUsername()))
        return true;
           return false;
        } catch (Exception e) {return false;}
  }

  public boolean isProfessorOfEdition(final Long eid){

    User user = (User)
         SecurityContextHolder.getContext().getAuthentication().getPrincipal();

    try{
      Account a = service.getAccountByUsername(user.getUsername());
      Professor prof = service.getProfessorByAccount(a);
      for(Edition e : prof.getEditions()){
        if(e.getId() == eid)
          return true;
      }
      return false;
    }catch(Exception e){
      return false;
    }
  }
  ...
}
```

Listing 6: Exemplo de verificação de segurança

6 Resultado

Neste capitulo decidimos mostrar o resultado final do projeto. Em baixo seguem algumas imagens e uma breve explicação da informação que é apresentada e as funcionalidades que permitem.

A imagem a baixo retrata a pagina de entrada do site. É possível ver alguma informação relevante sobre a faculdade, mais em baixo existe uma lista de contactos e ainda a lista de cursos e as cadeiras de cada um. Existe ainda o botão de LOGIN, para que o utilizador possa ter acesso ao formulário de autenticação ou registo.

Figure 4: Página inicial do site - Informação

Continuação da página de entrada do site referente a lista de cursos e cadeiras. Aqui é possível ver informação da oferta educativa e as cadeiras de cada um dos cursos.

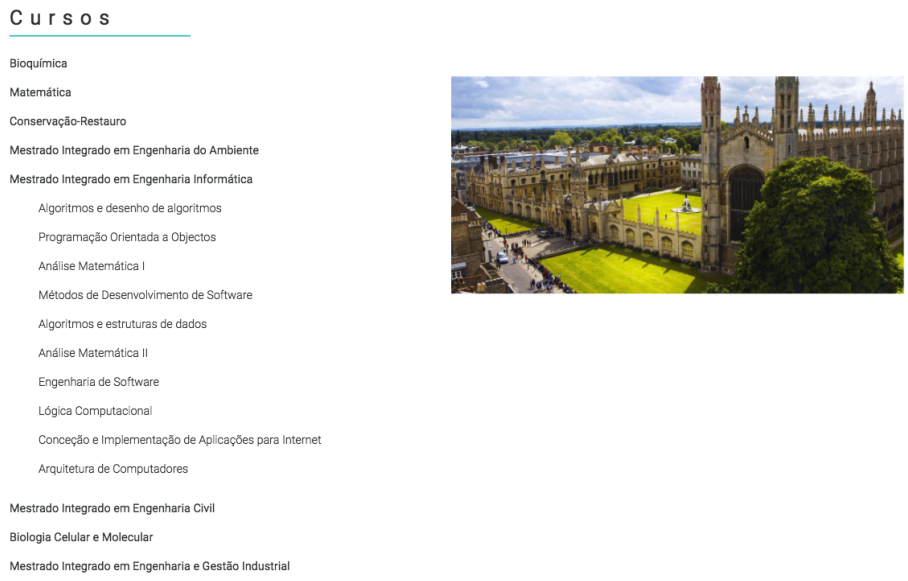

Figure 5: Página inicial do site - Oferta educativa

De seguida apresentamos duas imagens respetivas à parte de autenticação. A primeira representa o formulário de login de estudante, sendo que o do professor é semelhante. E a segunda representa o formulário de registo, para criar uma nova conta.

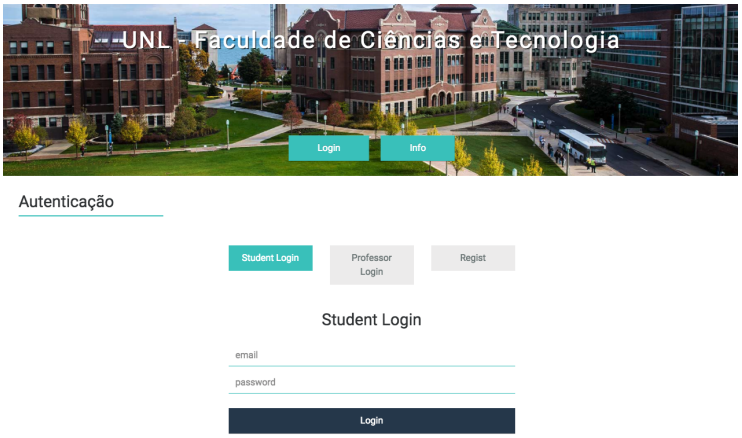

Figure 6: Autenticação - Formulário de Login

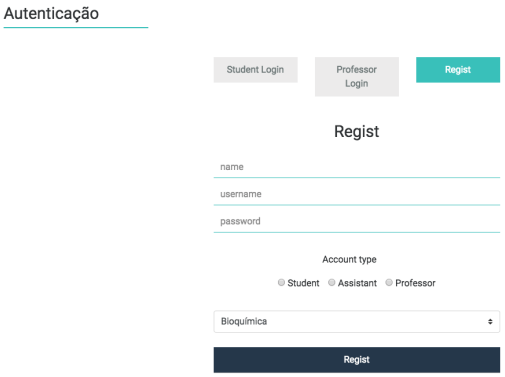

Figure 7: Autenticação - Formulário de registo

Em ambos os formulários é apresentado uma mensagem de erro caso tenha ocorrido um erro ao criar uma nova conta. Este erro pode ocorrer por campos inválidos, por já existir um utilizador com o nome da nova conta ou, no caso do login, por o nome de utilizador e a password estarem incorretos.

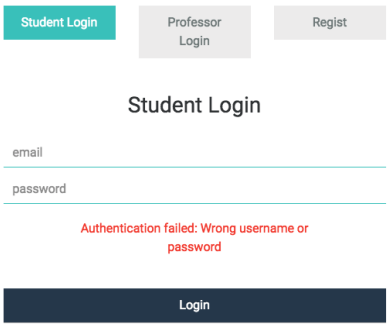

Figure 8: Autenticação - Formulário de Login em caso de erro

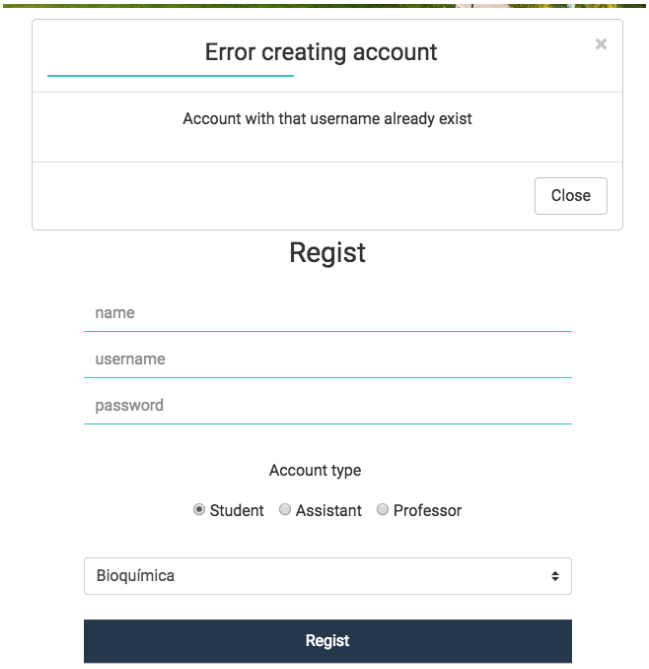

Figure 9: Autenticação - Formulário de registo em caso de erro

As próximas imagens retratam a área pessoal de um estudante. A página de entrada, na área pessoal do estudante, mostra algumas informações gerais do estudante e é apresentado um menu do lado esquerdo com as opções de navegação possíveis.

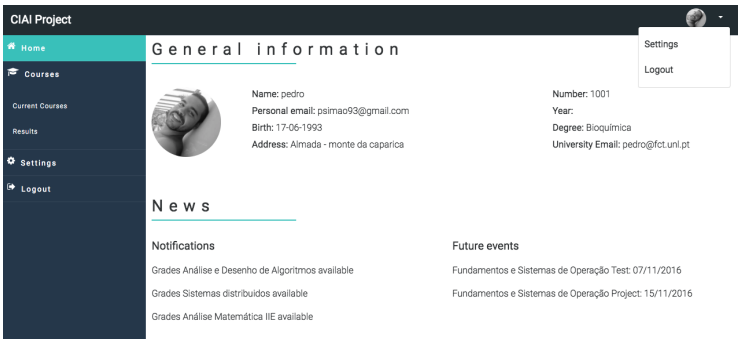

Figure 10: Estudante - Página inicial

Na imagem em baixo é apresentada uma página de definições, onde o estudante pode editar algumas informações como o email pessoal, a foto de perfil, a morada e a sua data de nascimento.

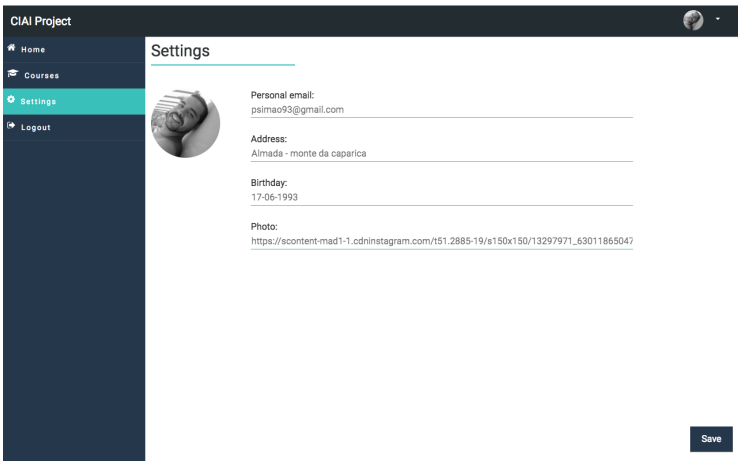

Figure 11: Estudante - Página de definições

Cada estudante tem acesso aos seus resultados e às informações das edições de cadeiras que estão concluídas ou às que está inscrito. Em baixo é apresentada a lista de edições de cadeiras que o estudante concluiu e o respetivo resultado. Para a lista de edições em que está inscrito, a lista é semelhante com a diferença de que não é apresentada a nota final. Em cada cadeira apresentada é possível clicar para obter mais informações sobre a mesma. Para uma cadeira que tenha, ou não, sido concluída, a vista é igual com a diferença que nas concluídas aparece a nota em cada avaliação.

Figure 12: Estudante - Página de resultados

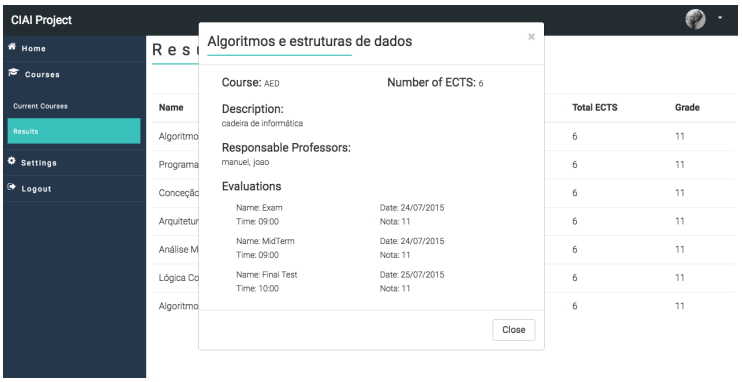

Figure 13: Estudante - Informação da edição de uma cadeira

De seguida apresentamos as imagens da área pessoal de um professor. A página de entrada na área pessoal do professor mostra algumas informações gerais do professor e é apresentado um menu do lado esquerdo com as opções de navegação possíveis. A interface apresentada, seja para um professor ou para um assistente, é igual, embora alguns botões não estejam disponíveis na interface consoante o tipo de conta autenticada. Para controlar também pelo lado do cliente as ações que cada tipo de professor pode executar. De qualquer maneira estas ações são controladas no lado do servidor consoante o tipo de conta que está autenticada. A página de entrada na área do professor e da página de definições são semelhantes à do estudante, com a diferença nos dados que podem ser editados. Na lista de cadeiras é apresentada uma lista de edições de cadeiras em que o professor é regente ou assistente. A lista é igual à apresentada aos estudantes, mas só é mostrado o nome da cadeira. Para aceder à página de gestão de uma cadeira basta clicar na respetiva cadeira.

Em baixo é apresentada a imagem correspondente à pagina principal de gestão de uma cadeira. Esta página está divida em 4 secções (um separador por secção). Em baixo é explicado o conteúdo de cada um dos separadores e as funcionalidades respetivas.

Na primeira secção é apresentado a lista de estudantes inscritos na edição da cadeira. Sendo a página de um professor (não assistente), é possível remover e adicionar novos estudantes à edição. Ambos podem pesquisar por um determinado aluno seja por nome ou número.

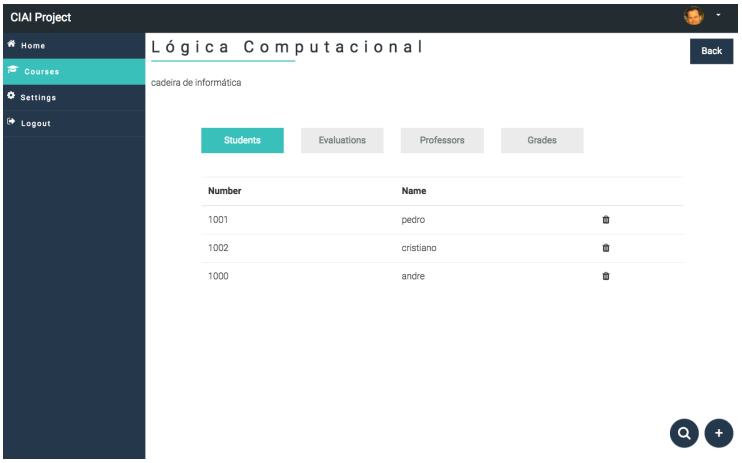

Figure 14: Professor - Estudantes inscritos numa edição

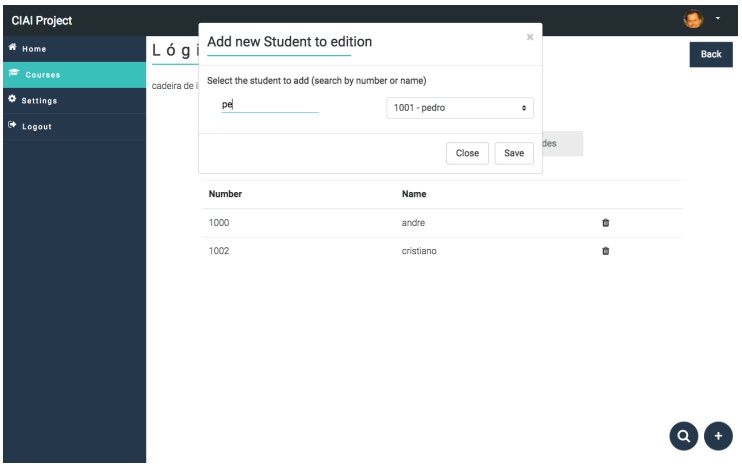

Figure 15: Professor - Adicionar estudante à edição

Na segunda secção é apresentada uma tabela com as avaliações da edição. Também neste caso sendo um professor, é possível editar a informação de uma avaliação, apagar ou criar uma nova. Ambos podem pesquisar por uma avaliação através do nome ou da data.

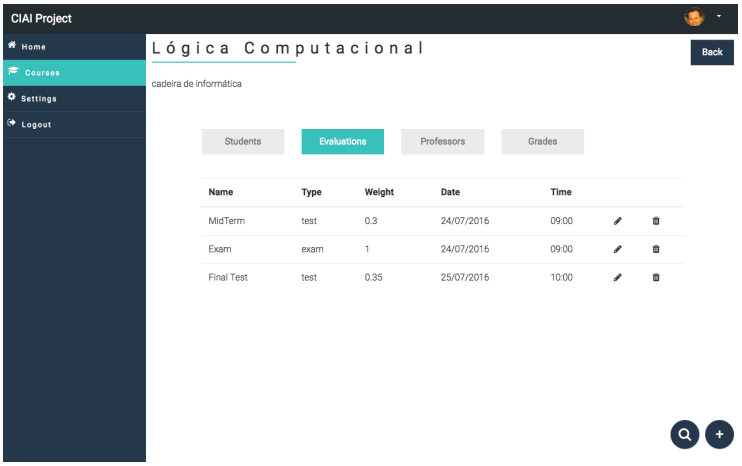

Figure 16: Professor - Lista de avaliações de uma edição

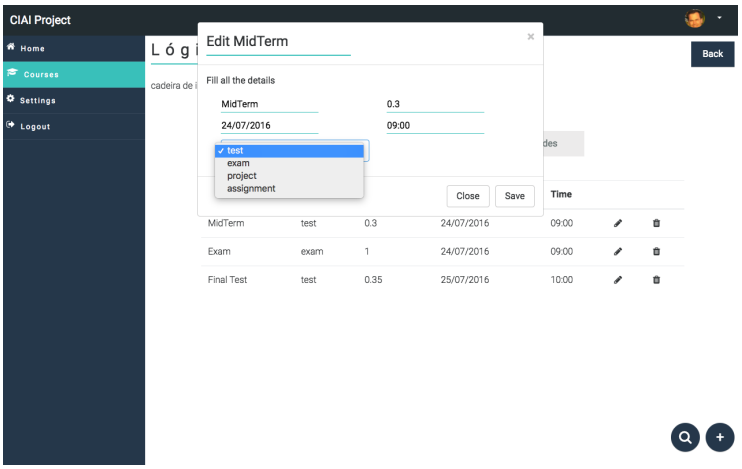

Figure 17: Professor - Editar detalhes de uma avaliação

Na terceira secção é apresentada uma lista dos professores atuais de uma edição. Um professor pode adicionar ou remover novos professores. Um assistente só pode consultar.

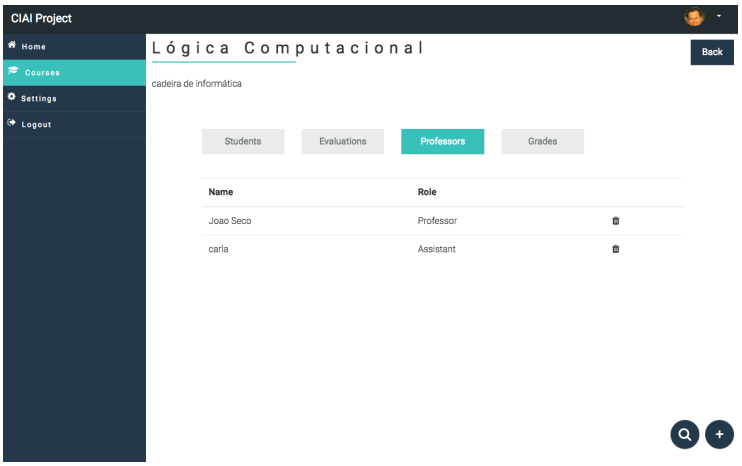

Figure 18: Professor - Lista de professores de uma edição

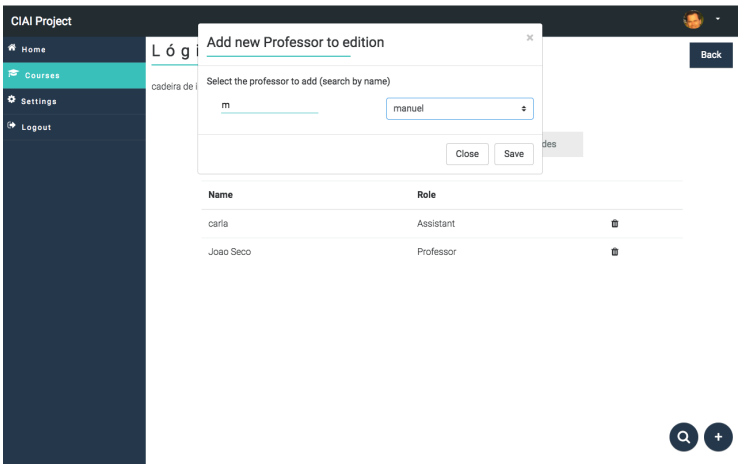

Figure 19: Professor - Adicionar professor a uma avaliação

Na ultima secção é apresentada uma tabela com as respetivas notas dos estudantes da edição. Ambos professores e assistentes podem mudar as notas dos estudantes.

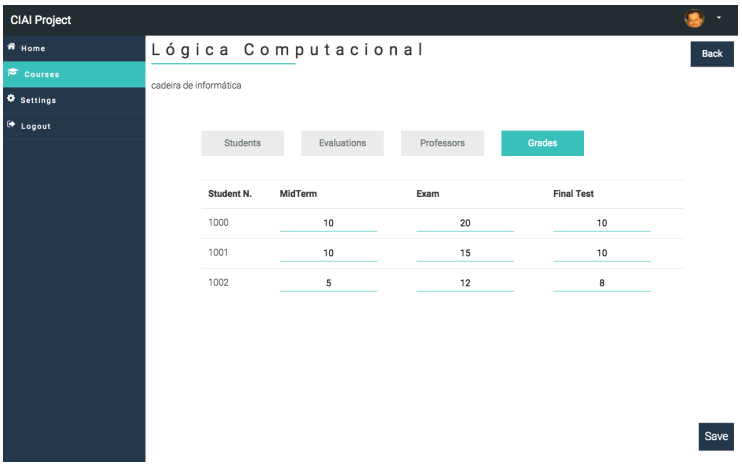

Figure 20: Professor - Adicionar professor a uma avaliação

Em cada secção sempre que se trata de uma operação de adicionar, editar ou remover são apresentadas mensagens de sucesso ou de erro. Caso se trate de um erro é mostrada uma explicação de qual poderá ter sido a causa.

7 Conclusão

Na elaboração deste trabalho foram surgindo várias questões. Inicialmente decidimos montar um ambiente de desenvolvimento um pouco diferente do que foi falado nas aulas da cadeira. Como já tínhamos algum conhecimento prévio de algumas ferramentas como o Webpack, Gulp e NPM decidimos que eram uma boa opção para este contexto.

Na primeira fase do projeto o proposto era implementar apenas o lado do cliente com o uso do React. Para adiantar algum trabalho e preparar a estrutura do cliente foi decidido que iríamos implementar uma base de dados em memória do lado do servidor e implementar um controlador para podermos obter informação que viria a ser apresentada na interface.

Na segunda parte do projeto, uma vez que já tínhamos grande parte das classes criadas foi mais rápido fazer a transição. Depois de decidirmos qual era o modelo de dados mais adequado passamos por fazer as anotações em JPA para que fosse de acordo com o planeado.

Concluídas as entidades necessárias, o passo seguinte passou por criar os controladores necessários e os respetivos repositórios. Decidimos que primeiro íamos deixar o lado do servidor funcional e só depois fazer as alterações necessárias do lado do cliente para ficarem consistentes.

À medida que os pedidos iam sendo feitos, recorremos a ferramenta Postman para irmos testando se estavam de acordo com o que era necessário no cliente.

Uma vez concluídas algumas das funcionalidades mais importantes, começamos por fazer as devidas alterações no lado do cliente. O que se mostrou ser razoavelmente rápido devido ao trabalho feito na primeira fase.

No desenrolar do projeto, como a quantidade de código e classes já existiam em número razoavelmente grande decidimos parar e estruturar todo o servidor. Daqui surgiu a arquitetura apresentada no capitulo 4. Separámos melhor os componentes, dando contexto e funcionalidades diferentes a cada um.

Neste momento surgiu também um outro problema. As respostas ao cliente nem sempre eram aquilo que era pretendido. Ou seja, em caso de tudo funcionar corretamente a resposta apresentada era o esperado mas em caso de erro não havia consistência no que era enviado nos diversos pedidos, em alguns casos retornava *null* , noutros era retornado uma exceção. Surgiu então a necessidade de criar vários tipos de exceção. Isto para sabermos do lado do cliente o que tinha falhado e assim poder apresentar uma mensagem consistente com o erro ocorrido. Estas mensagens são enviadas pelo servidor no caso de alguma exceção ter ocorrido.

O maior desafio no projeto foi a incorporação da camada de segurança na aplicação. Primeiro a questão da autenticação por termos feito um formulário em React e não através da *view engine* usada no projeto. Em segundo por não estarmos a conseguir implementar as medidas de segurança que à partida tínhamos em mente. Para resolver estes problemas fizemos alguma consulta pela documentação do Spring e algums fóruns, para ter uma ideia de como iríamos abordar este problema. Como solução chegámos ao que foi descrito no capitulo 5.

Embora tenhamos tido desafios e problemas no desenrolar do projeto, aprendemos sobre uma nova tecnologia. E apesar de já existirem algumas ferramentas semelhantes ao Spring, que nos proporcionam o desenvolvimento de aplicações com esta estrutura e o uso da técnica ORM (*Object Relational Model*) como é o caso do *Laravel* ou *NodeJS + MongoDB*, gostámos de aprender sobre Spring e achamos que é uma boa adição ao nosso *skill set*.

8 Bibliografia

Na realização do projeto para além dos slides disponibilizados e de alguns exemplos realizados na aula, os seguintes sites foram consultados à medida que iam surgindo algumas dúvidas.

- JPA Entity Relationships - `https://www.tutorialspoint.com/jpa/jpa_entity_relationships.htm`
- JPA Entity Primary Key - `http://www.objectdb.com/java/jpa/entity/id#Composite_Primary_Key_`
- JPA Many-To-Many Relationship Mapping - `https://hellokoding.com/jpa-many-to-many-relationship-mapping-example-with-spring-boot-maven-and-mysql/`
- JSON infinite recursion Stackoverflow - `http://keenformatics.blogspot.pt/2013/08/how-to-solve-json-infinite-recursion.html`
- Infinite Recursion with Jackson JSON and Hibernate JPA issue - `http://stackoverflow.com/questions/3325387/infinite-recursion-with-jackson-json-and-hibernate`
- Spring Web security - `http://tuhrig.de/expression-based-security-with-spring-security/`
- Spring documentação AuthFailure - `http://docs.spring.io/autorepo/docs/spring-security/4.1.0.RELEASE/apidocs/org/springframework/security/web/authentication/AuthenticationFailureHandler.html`
- Redirecionamento de paginas depois de Login - `http://www.baeldung.com/spring_redirect_after_login`