universidade de aveiro

Mestrado Comunicação e Multimédia

Ramo Multimédia Interativo

1º Ano | 1º Semestre

Projeto Altice Labs

Look&Learn

Share&Learn

Relatório Final PLB – P2

TDI – Tecnologias Dinâmicas para a Internet

Professor Carlos Santos
Professor Nuno Ribeiro

Grupo:
José Dias | 70832
Maria Inês Almeida | 73123
Nuno Soares | 58390

06.02.2017

Grupo G1: José Dias nº70832 | Maria Inês Almeida nº73123 | Nuno Soares nº58390

Índice

Sinopse do projeto 2
Introdução 3
Estratégias gerais para a implementação 4
Arquitetura global da aplicação final 5
Modelo de base de dados 7
Implementação server-side 8
Implementação client-side 12
Principais desafios tecnológicos resolvidos 15
Principais obstáculos não foram ultrapassados 19
Ferramenta de gestão de projeto utilizada 21
Funcionalidades acrescentadas 22
Conclusão 23
Webgrafia 24

Grupo G1: José Dias nº70832 | Maria Inês Almeida nº73123 | Nuno Soares nº58390

Sinopse do projeto

O nosso projeto divide-se em duas partes (a plataforma Share&Learn e o second screen Look&Learn).

A plataforma **Share&Learn** tem como objetivo fornecer um website a pais e educadores (professores) de forma a possibilitar a partilha e também a criação conteúdos educativos (jogos, vídeos, quizzes, palavras-cruzadas).

Por outro lado, **Look&Learn** é uma aplicação para um dispositivo second screen onde o nosso público-alvo (crianças dos seis aos dez anos) pode visualizar os conteúdos que foram publicados na plataforma Share&Learn enquanto assistem a um episódio de uma série televisiva do MeoKids na iTV (televisão interativa). Assim será possível associar diretamente um episódio do MeoKids com a aplicação second screen e, por sua vez, com o conteúdo que foi produzido para esse episódio em concreto.

Concluindo, a plataforma Share&Learn pretende facilitar a circulação de informação, referente à produção de conteúdos para utilização em dispositivos second screen. Permite ainda visualizar todos os conteúdos publicados pelos vários utilizadores, bem como ter uma conta pessoal onde estão presentes os próprios conteúdos do utilizador alvo. Para além disso, dá a conhecer vários developers deste tipo de aplicações, permitindo partilha de ideias, matérias, entre outros.

Já a Look&Learn é uma aplicação mais "lúdica" feita a pensar nas crianças que muitas vezes recorrem a séries televisivas apenas como forma de lazer e distração, podendo assim obter alguma aprendizagem informal desses "desenhos animados".

Grupo G1: José Dias nº70832 | Maria Inês Almeida nº73123 | Nuno Soares nº58390

Introdução

Todo o trabalho apresentado neste relatório foi desenvolvido na disciplina de Tecnologias Dinâmicas para a Internet e integra-se no projeto (PBL – Project Based Learning) que o grupo está a elaborar em conjunto com as disciplinas de DCM (Design e Comunicação Multimédia) e STI (Serviços e Tecnologias nas Instituições) e em parceria com a empresa Altice Labs.

Todo o trabalho elaborado ao longo desta disciplina serviu para desenvolver a parte técnica para as nossas duas aplicações (Share&Learn e Look&Learn). Nesta disciplina desenvolvemos a parte *backend* (utilizando a framework do PHP, designada Laravel) e *frontend* (utilizando a framework de javascript designada react e ainda a arquitetura reflux de implementação do Flux) do projeto, aprendendo novas tecnologias de implementação que nos ajudaram a conseguir atingir os nossos objetivos.

No caso da disciplina de Design e Comunicação Multimédia começámos por desenvolver o estado de arte, procurando aplicações já existentes semelhantes à nossa. Definimos a estratégia de cor da aplicação, desenvolvendo uma paleta de cores, a adequação tipográfica e as funcionalidades. Desenvolvemos a navegação dos layouts, desenhado inicialmente os mockups no https://uaveiro.mybalsamiq.com e posteriormente no *illustrator*. Por fim, desenvolvemos o logótipo para ambas as aplicações e ainda um protótipo de baixa fidelidade, onde aprendemos e utilizámos tecnologias inovadores como o https://www.invisionapp.com/ .

Já a disciplina de Serviços e Tecnologias para a Internet serviu para avaliar a user experience da nossa aplicação. Desenvolvemos um plano de testes, com instrumentos de recolha de dados e posterior análise, testámos a usabilidade das nossas aplicações e conseguimos ir presencialmente a um ATL de crianças, para estar com o público-alvo da nossa aplicação para os conseguir ver a interagir.

Assim, todas as unidades curriculares envolvidas serviram para a implementação do nosso projeto, contribuindo de diferentes formas.

Este relatório serve como complemento do projeto e está estruturado da seguinte forma. Começámos pela sinopse do projeto. Posteriormente referimos quais as estratégias gerais para a implementação, qual a arquitetura global final, o modelo da base de dados. De seguida, demonstramos aquilo que foi utilizado na implementação do server-side e do client-side. Também fizemos referência aos principais desafios tecnológicos resolvidos e aos obstáculos que não conseguimos ultrapassar. Falámos acerca da ferramenta de gestão de projeto utilizada para o acompanhamento de todo o trabalho pelos vários membros do grupo, bem como pelos docentes da disciplina que nos auxiliaram em todo o processo. Mencionámos as funcionalidades que foram acrescentadas e concluímos com a Webgrafia utlizada para nos ajudar no desenvolvimento.

Como foi referido na sinopse, o nosso projeto baseia-se em duas partes e é composto por duas aplicações e, desta forma, todo o trabalho foi um desafio constante. No entanto o grupo esforçou-se para conseguir entregar um MVP consistente e melhorado em relação ao protótipo entregue inicialmente.

Grupo G1: José Dias nº70832 | Maria Inês Almeida nº73123 | Nuno Soares nº58390

Estratégias gerais para a implementação

As aplicações Share&Learn e Look&Learn foram desenvolvidas com o objetivo de serem uma "modern web app", ou seja, uma plataforma que aglomere tecnologias recentes, por forma a obter ganhos consideráveis na fase de produção, relativamente aos tópicos de segurança, velocidade de renderização no browser, otimização de código e organização geral na implementação. O processo de desenvolvimento iniciou-se com a programação de páginas estáticas em HTML e CSS, e usando o Bootstrap, por forma a facilitar o processo de adaptação da nossa plataforma em vários dispositivos. Seguiu-se o protótipo elaborado na unidade curricular de Design de Comunicação Multimédia, como referência do desenvolvimento para as páginas estáticas. Uma vez que o público-alvo da aplicação Share&Learn era dedicada para professores e desenvolvedores de aplicações em second-screen, a otimização de CSS foi ajustada para tablet e desktop, sendo ignorada a interface para mobile. Para a aplicação Look&Learn, foi elaborado o desenvolvimento para tablet e desktop.

Por forma a agilizar o processo de desenvolvimento nesta fase, foram feitas algumas alterações visuais por forma a simplificar o processo, uma vez que o número de páginas do protótipo, eram bastante elevadas.

Após o fim desta fase de desenvolvimento, iniciou-se o processo de converter todo o HTML e CSS para o React. Nesta fase eliminaram-se as repetições de código, e isolou-se o HTML em vários componentes isolados e que posteriormente eram invocados quando necessário. Por fim, iniciaram-se os trabalhos de desenvolvimento da API em Laravel, e com a respetiva documentação, por forma a facilitar o processo de desenvolvimento do client-side. À medida que a API ia sendo implementada, foram feitos os módulos de programação do lado do React, por forma a integrar um fluxo de dados dinâmico entre as componentes client-side e server-side. Por forma a garantir vários programadores a alterar uma plataforma, os lados de client-side e server-side foram submetidos no github, garantindo assim que os ficheiros estão sincronizados entre todos os elementos.

Grupo G1: José Dias nº70832 | Maria Inês Almeida nº73123 | Nuno Soares nº58390

Arquitetura global da aplicação final

A aplicação Share&Learn e Look&Learn estão divididas em dois módulos diferentes, e que têm em comum um fluxo de comunicação entre eles. Esse processo está exemplificado na seguinte figura:

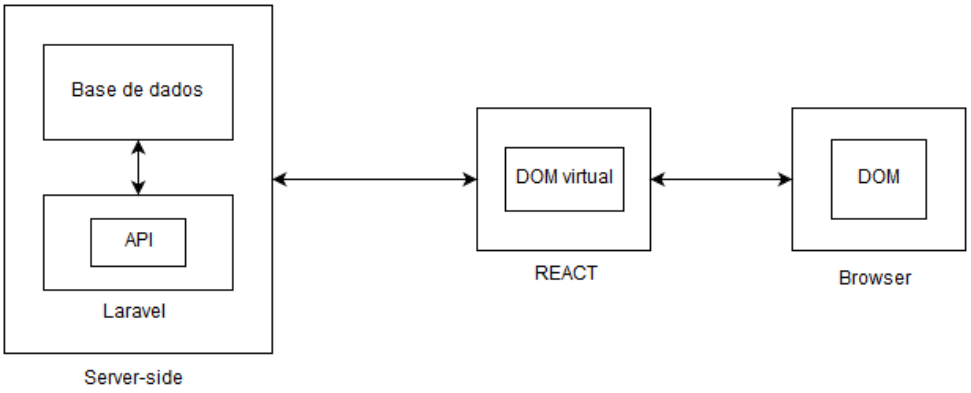

Figura 1 - Arquitetura server-side e client-side

O server-side é responsável por interagir com a base de dados, e gerar um JSON pela API. Esta comunicação é efetuada entre o *Laravel* e o MySQL sendo que são feitos pedidos à base de dados e esta retorna o resultado da query feita. A API representa os resultados que o *Laravel* foi buscar, e são do formato JSON. O formato JSON tem sido adotado para *RESTful API's*, ao passo que as SOAP *API's* usam XML. Desenvolveu-se uma API em REST, visto que é uma forma rápida de produção, e mais orientada para a web. Para além disso, os resultados em JSON são mais fáceis de ler.

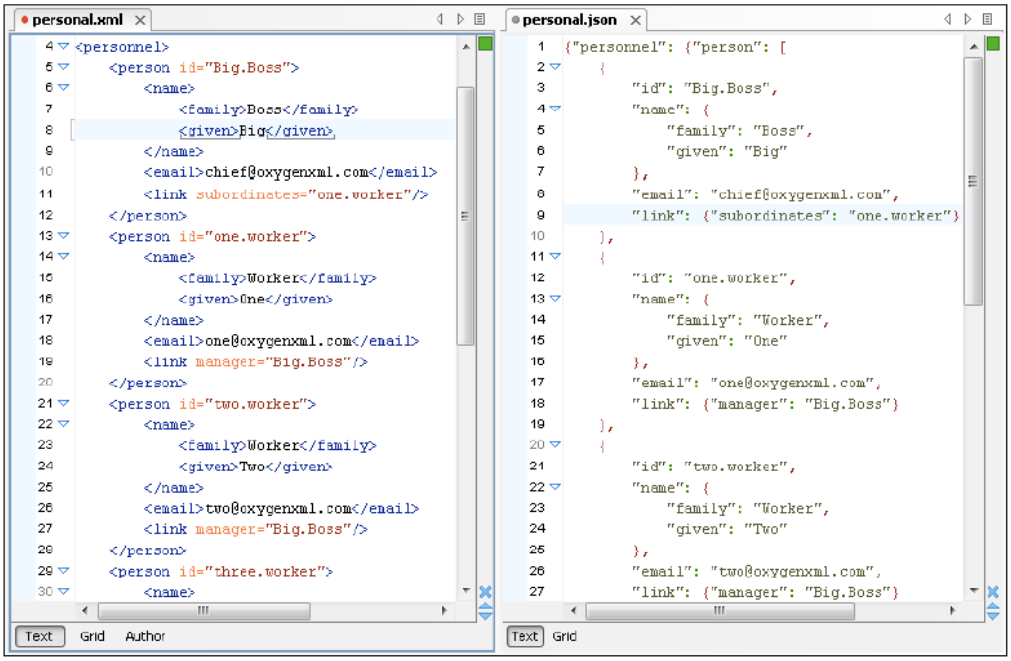

Figura 2 - JSON vs XML

De acordo com a matéria teórica lecionada conseguimos perceber que o JSON é superior ao XML por várias razões. Contudo, o XML oferece duas enormes vantagens como uma linguagem de representação de dados. É baseado em texto e é independente de posição. Juntos (XML e JSON) incentivaram um maior nível de independência de aplicações do que outros formatos de intercâmbio de dados. Infelizmente, o XML não é adequado para o intercâmbio de dados pois não corresponde ao modelo de dados da maioria das linguagens de programação. Existe, no entanto, o JavaScript Object Notation (JSON). O JSON é processado mais facilmente porque a sua estrutura é mais simples. Assim, sendo uma notação mais simples, precisa de software muito menos especializado. Por outro lado, o JSON é um formato de troca de dados melhor, enquanto que o XML é um melhor formato para troca de documentos.

O **REST** é um padrão de arquitetura, tendo uma relação cliente-servidor sem qualquer estado. Isto significa que não existem sessões guardadas do lado do servidor. Cada pedido tem todas as informações necessárias para o servidor autenticar o utilizador. A tecnologia REST tira partido dos métodos HTTP, e usando os seguintes métodos:

GET – solicitação de leitura;
POST – insere dados novos;
PUT/PATCH – modifica os dados existentes;
DELETE – elimina dados;

Do lado do cliente, temos o **REACT**, que é uma arquitetura responsável por gerar as *views* no browser. O REACT tem embutido no seu sistema uma DOM virtual, que é alterada consoante as atualizações de dados que são recebidas, sendo posteriormente feito o *update* na representação visual. O REACT difere do Javascript, na medida em que o Javascript para escrever qualquer alteração, tem de ler o código todo, e gerar uma nova representação da *view*, ao passo que o REACT apenas atualiza um componente, e sem recurso a uma nova leitura de toda a página HTML. A DOM virtual é corrida e compilada, de forma que o browser consiga perceber o desenvolvimento que foi feito em REACT ("JSX files").

Fluxo de dados:

Os protocolos de comunicação entre o client-side e o server-side são estabelecidos via HTTP. Existe uma comunicação entre cliente e servidor através de mensagens. O cliente envia uma mensagem de requisição de um recurso, e o servidor envia uma mensagem de resposta ao cliente com a solicitação.

Grupo G1: José Dias nº70832 | Maria Inês Almeida nº73123 | Nuno Soares nº58390

Modelo de base de dados

Relativamente aos desafios propostos pelo grupo de trabalho, foi necessário fazer uma modelação do problema que tínhamos em mãos. Os problemas foram identificados e modelados numa base de dados constituída por múltiplas entidades, sendo que estas têm relações entre si. Estas relações foram traduzidas na base de dados relacional que se pode ver em baixo.

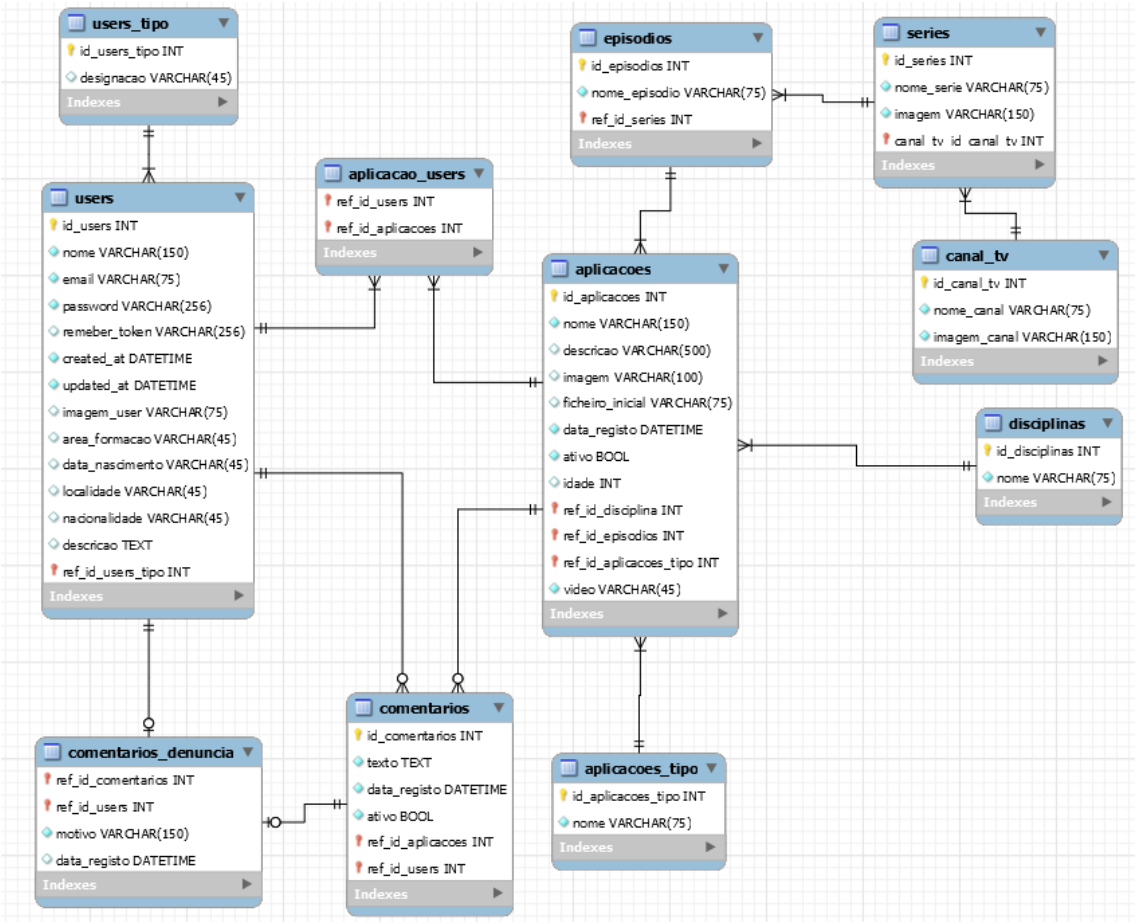

Figura 3 - Modelo de Base de Dados

Por forma a evitar erros de introdução de dados nos campos, já existem valores possíveis de ser usados pelo utilizador.

Depois foi garantida a integridade referencial, garantindo que existem chaves estrangeiras na tabela de chegada e *primary keys* na tabela de partida.

Por último, foi efetuado o processo de normalização, de forma a reduzir redundâncias e dependências de dados.

.

Grupo G1: José Dias nº70832 | Maria Inês Almeida nº73123 | Nuno Soares nº58390

Implementação server-side

A implementação server-side está baseado na framework de Laravel. Foi feita uma instalação de raiz e com recurso a um gerador de documentação, que lista todas as *routes* criadas e vai buscar os comentários de todos os controladores criados.

https://github.com/mpociot/laravel-apidoc-generator

Ao terminar a tarefa descrita acima, foi feito o *upload* de todos os ficheiros para um servidor da Amazon, previamente configurado nas aulas práticas da unidade curricular de Tecnologias Dinâmicas para a Internet. Foram feitas as configurações necessárias por forma a automatizar a pasta *Github*, *localhost* e server, e configurado o *virtualhost* com o caminho de acesso à pasta de projeto.

1. Criação de tabelas

Foram criadas todas as tabelas representadas na base de dados na pasta *migrations.* Cada coluna tem um tipo de dados adequado.

```
public function up()
{
    Schema::create('series', function (Blueprint $table) {
        $table->increments('id_series');
        $table->string('nome_series');
        $table->string('imagem_series');
        $table->timestamps();
    });

    Schema::table('series', function(Blueprint $table) {
        $table->integer('ref_id_canal')->unsigned();
        $table->foreign('ref_id_canal')->references('id_canal_tv')->on('canal_tv')->onDelete('cascade');
    });
```

Figura 4 - Criação de tabelas

2. Seeds

Por forma a tirar partido das potencialidades do migrate, foram usadas seeds que preenchessem a base de dados com dados essenciais para as aplicações, como as séries existentes, tipos de utilizadores, entre outros.

Na tabela utilizadores, por forma a facilitar o processo de testes, é gerado automaticamente vários utilizadores com *username* e email fictícios.

```
public function run() {
$faker = Faker\Factory::create();

    static $password;

for($i = 0; $i < 50; $i++) {
    App\User::create([
        'username' => $faker->userName,
        'email' => $faker->email,
        'password' => $password ?: $password = bcrypt('secret'),
        'remember_token' => str_random(60),
        'imagem_user' => 'https://app.awolacademy.com/img/avatar.png',
        'ref_id_users_tipo' => 1
    ]);
}
```

Figura 5 - Gerar utilizadores random

3. Models

A definição do tipo de relações existentes entre tabelas ocorre no *models do laravel*. Cada tabela é definida por um modelo, sendo que existem funções que definem as ligações entre tabelas, podendo ser de muitos para muitos, um para muitos, ou de um para um.

4. Controladores

Os controladores são responsáveis pela lógica de programação. Do ponto de vista da lógica geral de cada controlador, existe um *construct* que define os *headers* do CORS, permitindo assim que existam pedidos à API de fontes externas. Outra componente do construct, é o middleware da API, fornecido pelo *passport do laravel.*

Um controlador é definido por uma *route*. Usou-se o *route:resource*, por forma a usar as funções *index, store, show, update e destroy.* Esta estrutura permite realizar todas as necessidades da API, relativa à logica dos dados. Cada resposta da API relativa a uma query efetuada vem com uma *string* de sucesso ou insucesso.

5. Passport

O passport é um sistema de autentificação da API, por parte de aplicações externas. Após terminado o processo de instalação, é possível emitir um *token* temporário, que deverá estar definido no header de um pedido POST, PUT ou DELETE. Em alternativa é possível associar um pedido de autorização a um domínio, tendo assim acesso à manipulação de dados. O passport está integrado no sistema de login, e permite retornar o ID da sessão do utilizador, tornando assim as nossas páginas dinâmicas e privadas, sendo que só são permitidas manipulações nos dados referentes ao utilizador que tem feito o login.

6. Querys por Url

A página da biblioteca geral permite filtrar os resultados das aplicações existentes na base de dados, e que estão relacionadas uma série existente na plataforma. Essa seleção é feita através de um pedido GET por url, com a seguinte sintaxe:

http://josedias.zapto.org/api/biblioteca-geral?nome_series=Dexter

```
$series = (new Series)->newQuery();

if($request->exists('series')){
    $series->orderBy('nome_series', 'asc');
}

if($request->has('nome_series')){
    $series->where('nome_series', $request->nome_series);


}
```

Figura 6 - Querys por Url

O index de biblioteca geral retorna dois arrays distintos (Data1 e Data2). Data1 retorna todas as aplicações existentes na base de dados e Data2 retorna o valor da query introduzida por url. A injeção do URL ocorre do lado do REACT.

7. Querys gerais de índex:

```
$series2 = Series::with('apps')->get();
```

Esta query relaciona todas as aplicações pertencentes a uma determinada série, e lista todos os resultados.

8. Alterar foto perfil:

A função alterar perfil, espera por um ficheiro em form-data, guarda os dados numa pasta do servidor e grava o caminho para aceder à imagem numa coluna da base de dados dedicada. São retornadas mensagens de sucesso ou insucesso.

```
if ($user) {

    if ($request->hasFile('imagem_user')) {

        $file = $request->file('imagem_user');

        $path = $file->hashname();
        $file->move(public_path('images/profile'), $path);

        $user->imagem_user = 'images/profile/' . $path;
        $user->save();

        return $this->_result('Imagem alterada com sucesso');

    } else {
        return $this->_result('Erro de submisao');
    }
}
return $this->_result('User invalid', 404, 'Not Found');
```

Figura 7 - Guardar imagens no servidor

9. Validators

O validator é usado quando é suposto receber inputs do lado do client-side. É feita uma verificação do tipo de dados, e é retornado uma mensagem. Caso o *validator* dê erro, o resto do código da função não é retornado. Esta função é bastante útil para garantir o tipo de dados que são introduzidos pelo utilizador e pelo que é esperado no sistema.

10. Inserção de dados

Cada campo dos formulários no laravel está à espera de receber um valor, de forma a que exista um PATCH na base de dados. Assim os utilizadores não têm de inserir dados em todos os campos para alterar os seus dados pessoais e de perfil.

```
if ($request->has('area_formacao'))
```

11. Stores

Todas as funções de store, que guardam novos dados na aplicação Share&Learn necessitam de invocar o modelo respetivo. No seguinte exemplo estamos a guardar um novo utilizador na base de dados:

```
User::create(
    [
        'username' => $data['username'],
        'email' => $data['email'],
        'password' => bcrypt($data['password']),
        'ref_id_users_tipo' => 1,
    ]
);
```

Figura 8 - Inserção de dados na base de dados

12. Headers Postman

Todos os controladores implementados foram testados no postman, e com os headers respetivos. Este processo permitiu fazer um debugging do lado da API.

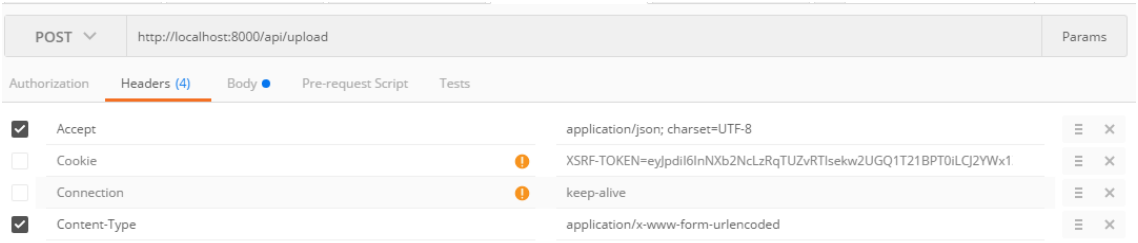

Figura 9 - Headers Postman

No caso dos POSTS, PUT e DELETE, foi necessário desativar o content-type, nos casos em que era necessário fazer o envio de um form-data. Para envio exclusivo de strings, usou-se o x-www-form-urlencoded, e o header contente-type ativo, como ilustra a imagem 9.

Grupo G1: José Dias nº70832 | Maria Inês Almeida nº73123 | Nuno Soares nº58390

Implementação client-side

Optou-se por usar o REACT ES5, uma vez que todos os elementos do grupo estavam mais familiarizados com essa sintaxe. Assim tornou-se mais fácil o contributo de programação, por parte de todos.

A página **public/index.html** é a página destino de toda a programação elaborada, sendo constituída por uma única div. Dentro da pasta src, existe um ficheiro main.jsx responsável por linkar todas as routes entre páginas e chamar todos os componentes necessários. No entanto, os packs instalados no REACT permitem que o código em main.jsx seja compilado automaticamente para a pasta **public/js/main.js**. Este ficheiro depois de compilado, está pronto para ser consumido pelo browser. A pasta src também contem todos os componentes necessários para a renderização da DOM virtual no REACT.

Para puxar os dados da API, foi instalado o seguinte pacote fetch que permite substituir o uso de XMLHttpRequest

https://github.com/github/fetch

A implementação do fetch no desenvolvimento do Share&Learn e Look&Learn, só permite usar pedidos POST, PUT e DELETE. No entanto do lado da API usou-se o PATCH, pois este pedido permite alterar apenas as colunas solicitadas. Utilizou-se então o PUT no REACT, sendo que não foram encontrados quaisquer problemas do lado da API, até ao momento da elaboração deste relatório. Para cada input de formulário, foi necessário fazer um pedido PUT. Por exemplo na componente de editar dados pessoais, é necessário fazer um PUT para cada inserção de dados (username, email, password).

Face a restrições de tempo no período de implementação, optou-se por cortar algumas páginas que foram analisadas pelos docentes na fase de prototipagem em Invision, nomeadamente:

- Biblioteca Pessoal
- Biblioteca pessoal – detalhe
- Landing page
- Cronologia
- Parametrização – substituída por um vídeo, uma vez que existiam quatro tipologias diferentes para uma aplicação
- Recuperar password

Sendo assim a página principal passou a ser a biblioteca geral, que lista todas as aplicações existentes. Ao selecionar uma aplicação, é visto o detalhe da aplicação.

A página relativa a informações pessoais do utilizador é consultada no separador “sobre”. Neste caso em concreto, foram feitos maps, dentro de maps, por forma a chamar os valores que estavam dentro da string JSON, uma vez que estavam a ser feitas relações entre tabelas, com outputs de arrays, dentro de arrays.

Na listagem de séries, devido ao facto de a função onClick, não estar a funcionar para o dropdown, alterou-se o dropdown, para botões dinâmicos. Estes botões são ativos com a função onSubmit. Apenas foi desenvolvido a seleção para a série Dexter.

O menu lateral é invocado pelo css, sendo que este é expandido e eliminado da view através de classes. Para essa classes, foi criado em estado this.state no modo ativo e não ativo.

Ao fazer um onClick no ícone, os estados são alterados. Dentro do menu, é possível aceder a vários urls com funcionalidades da aplicação.

Para comutar entre o menu de login e registo inicialmente foi desenvolvido um plugin em jquery, que animava e permitia a transição entre painéis. Ao transpor esta lógica para REACT, foi eliminado o Jquery e substituído por estados que permitem que as classes de CSS fiquem ativas ou não-ativas.

A página de publicar foi simplificada por forma a apenas submeter uma nova aplicação na base de dados, retirando assim alguma lógica na sua programação.

a) Reflux

Inicialmente as duas aplicações estavam a utilizar apenas React no *frontend* a comunicar com uma api no *backend* em Laravel. Contudo verificámos que não era a maneira mais eficiente, uma vez que em certas páginas estavam a ser feitos quatro pedidos em simultâneo à API para receber os mesmos dados. Para resolver este problema começámos a utilizar o Reflux implementando assim a arquitetura Flux, passando apenas a ser feito um pedido nessas mesmas páginas.

Ao utilizar o Reflux foi possível reduzir as linhas de código, e seguir de forma mais eficiente o principio DRY (Don't Repeat Yourself) facilitando a manutenção do código das aplicações e reutilizando os elementos sem ser necessário estar sempre a repetir o mesmo código. Em vez de ser preciso mudar um parâmetro em vários sítios, passou a ser apenas necessário mudar num local, e a leitura de código passou a ser mais fácil e simples. É muito importante separar a parte logica da parte gráfica.

As *stores* na arquitetura Flux contêm o estado e lógica da aplicação. O seu papel é semelhante ao model (o M) do MVC. No entanto, estas são responsáveis pelo estado de vários objetos sendo diferente dos modelos ORM.

No Reflux passámos a utilizar varias *stores* que são responsáveis por fazer pedidos a Api de informação, e para enviar dados à Api. Foram implementadas Actions referentes a cada uma das operações básicas da *store*, facilitando a reutilização do código, uma vez que, um mesmo tipo de ação pode ser referenciado em qualquer parte da aplicação, e a store fica a saber de forma imediata o que necessita fazer.

As *stores* ficam à escuta de pedidos de actions, que são ativadas nas views, e as views estão sempre à escuta de alterações nas *stores* para garantir a atualização dos dados sem ser necessário fazer *refresh* na página.

Exemplo: Temos uma *store* responsável por retornar os dados referentes ao utilizador logado, e por atualizar os dados referentes a esse utilizador. Quando o utilizador alterar os seus dados a view emite uma *Action* que é posteriormente executada pela *store*. Após a alteração efetuada na *store,* ela avisa que existem atualização dos dados, e a view atualiza os dados sem refresh, uma vez que, a view está à escuta de alterações na *store.*

Para se fazer os pedidos utilizou-se a biblioteca fetch, que facilita a interação com a Api, e é amplamente utilizada por uma grande parte da comunidade de programadores. Contudo detetamos que a mesma não permite o método Patch, dai ter sido utilizado vários métodos Put para efetuar as alterações dos dados dos utilizadores, por exemplo.

b) Alterar dados e foto de perfil

```
var InputImage = React.createClass({
  UpdateImagemUsera:function(){
    console.log("hey");

    this.props.onSubmit(this.refs.imagem_user);

  },
  onChange: function(){
    this.props.onChanger(this.refs.imagem_user);
  },
    render: function() {
```

Figura 10 - Alteração de estado da imagem em front-end

A função **UpdateImagemUsera** espera por alterações e renderiza o novo estado, sem fazer um refresh à página do browser.

c) Injetar dados no servidor

```
fetch('http://josedias.zapto.org/api/editar-perfil/1', {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      // email : this.refs.emailNovo.value,
      area_formacao: this.refs.area_formacao.value,
      // password: this.refs.passwordNova.value
      // data: data
    })
  }).then(function(response) {
      Actions.getSobre()
  });
```

Figura 11 - Fetch

O fetch permite definir um método do tipo GET, POST, PUT, DELETE e um tipo de conteúdo que neste caso é em JSON.

Como exemplo na figura 11, está a ser enviado o campo referente à área de formação de um utilizador para a base de dados.

Grupo G1: José Dias nº70832 | Maria Inês Almeida nº73123 | Nuno Soares nº58390

Principais desafios tecnológicos resolvidos

a) Implementação de comentários associados a uma aplicação do lado da API

```
$aplicacoes = Aplicacao::get();
$apps = [];
foreach ($aplicacoes as $app)
{
    $application = array('app' => $app);

    $comentarios = Comentario::with('comentarios_do_user')

        ->whereRefIdAplicacao($app->id)
        ->get();

    foreach ($comentarios as $comment)
    {
        $application['coments'][] = $comment;
    }

    $apps[] = $application;
}
```

Figura 12 - App com comentários associados

Esta query faz uma listagem de todas as aplicações existentes, e lista os comentários associados a cada aplicação. Tendo em conta as relações entre tabelas existentes, foi necessário colocar a query relativa aos comentários associados a uma app num array. O processo de escrita é escrito, consoante o número de chaves estrangeiras correspondentes à chave primária.

b) Alterar imagem em editar-perfil na API, substituindo um patch por um POST

A função de submeter um PATCH em form-data deu grandes problemas na questão dos headers do Postman, uma vez que o content-type estava sempre a ser forçado a texto/html. Neste caso pretendia-se submeter uma imagem do tipo "file". Reformulou-se a função, forçando os dados a serem submetidos via POST, e com consequente alteração nos dados da coluna pretendida.

c) Querys para o Look&Learn

Existiram situações onde as funções do laravel não nos podiam auxiliar na relação entre tabelas. Para o caso do Look&Learn, é necessário receber um parâmetro de input, que pode ser uma série ou um nome de uma aplicação, tendo de ser retornado todas as apps associadas, caso seja inserido o nome de uma série.

Figura 13 - Opção "search bar" com o retorno de todas as aplicações associadas a uma série

Para resolver este problema, foi feito SQL à moda antiga, por forma a perguntar à base de dados se tinham nomes iguais aos inputs, como mostra a **figura 14:**

```
$data = $request->all();

if ($request->has('input')) {

    $keyword = $data['input'];

        $apps = DB::table('aplicacaos')->select('aplicacaos.id', 'aplicacaos.nome', 'aplicacaos.ima
            ->join('series', 'aplicacaos.ref_id_episodios', '=' , 'series.id_series')
            ->where('series.nome_series', '=' , $keyword)
            ->orWhere('aplicacaos.nome', $keyword)
            ->get();

    if(empty($apps[0])){
        return $this->_result('Essa procura não existe');
    }

    else{
        $output = json_encode(array('data' => $apps));
        return $output;
    }
```

Figura 14 - Sistema de filtragem por séries existentes ou por aplicação

Caso o array retornado esteja vazio, o utilizador é informado do resultado da pesquisa. Caso existam resultados é retornado um array com toda a informação necessária.

Também foram adicionados atalhos de retorno, sempre que é selecionada uma série no look&learn.

d) Strings JSON em arrays

O output de dados na API foi estruturado por forma a estar dentro de arrays, como é possível ver na figura abaixo.

```
$output = json_encode(array('data' => $apps));
return $output;
```

Figura 15 - Output de dados

Este processo permitiu fazer os processos de chamada no lado do React, de uma forma mais organizada.

e) Publicar aplicações com imagem e vídeo

O formulário para publicar aplicações é composto por campos que recebem texto, imagens e vídeo. Para enviar dados do react para a API, existe um header que define os tipos de dados que vão ser enviados, no entanto não foi possível aferir a impossibilidade de enviar todos os tipos de dados num único POST do lado do React. Para solucionar este problema, o processo de envio foi separado em duas componentes, sendo que a primeira componente é direcionada para um url responsável por lidar com o texto e a segunda componente contém um url para lidar com o vídeo e com as imagens. Resumidamente é feito um primeiro POST que preenche todos os campos, inclusivamente os referentes à imagem e ao vídeo, com dados fictícios. O segundo POST vai fazer um update à última linha inserida na base de dados, com os caminhos de acesso às imagens e aos vídeos e posterior upload dos dados para a pasta public do laravel.

f) CORS

Existem dois servidores ao serviço do grupo de trabalho, sendo que um é responsável pela API e o outro é responsável pelo front-end. Este facto obriga a que esteja ativo o Cross-Origin-

Resource Sharing, ou seja a troca de informação entre 2 servidores. Foi utilizador o seguinte pacote pré-configurado:
https://github.com/barryvdh/laravel-cors

A instalação deste pacote não foi pacífica. Numa primeira fase a instalação do pacote não estava a ser feita com sucesso no servidor, o que obrigou o grupo de trabalho a passar todo o projeto para um único servidor, e com posterior alteração de todas as linkagens nos pedidos à API. Apesar de esta solução evitar o uso do CORS, todo o login, registo, e passport no laravel ficaram com as suas linkagens destruídas, impedindo assim o desenvolvimento de autenticação na API.

Após muitas tentativas falhadas em restaurar as linkagens no passport, destruiu-se a API no servidor, e foi criada uma nova instalação. Este processo permitiu incorporar o pacote de CORS, e restaurar o login, registo e passport. Como consequência, o front-end teve de ser novamente restaurado nas linkagens ao servidor.

g) Login - Passport

O processo de autenticação foi desenvolvido por forma a tirar as potencialidades do laravel 5.3, nomeadamente o passport. Sempre que é feito um pedido de autenticação, é emitido um token que é associado ao id do user em sessão. Esta correspondência permite obter o id do utilizador, sempre que é feito um pedido à API.

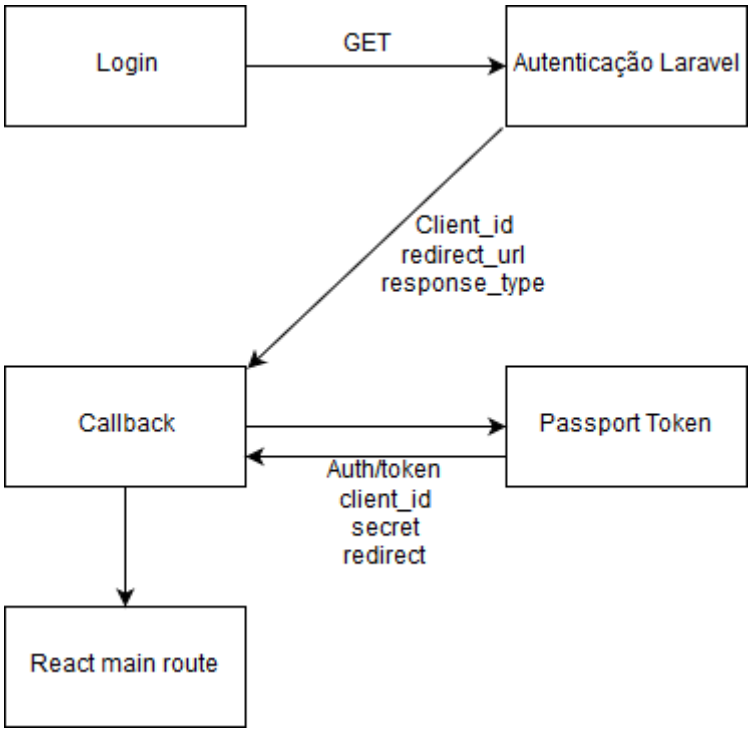

Figura 16 - Esquema de autenticação do Share&Learn

Resumidamente, existe uma página de login, a redirecionar o cliente para a página de autenticação do laravel. Ao digitar os dados corretos, é feito um redirect para uma página de callback, onde é recebido um token de acesso e os dados do utilizador. Esses dados são armazenados através da função localstorage, que guarda todos os dados no browser. Ao terminar

este processo, somos redirecionados para a página principal da aplicação Share&Learn. Para processo de logout, elimina-se o token da localstorage e reencaminha-se o utilizador para a função Auth:logout().

O ficheiro Callbackreact.jsx é uma adaptação de uma chamada em AJAX, que permite obter o token de acesso emitido pelo Passport e guardá-lo no cliente.

A aplicação Look&Learn não tem sistema de autenticação, uma vez as aplicações submetidas devem ser acedidas por todas as crianças que estejam a navegar na plataforma.

h) Login – Callback

Inicialmente foi testado um ficheiro em javascript que recebia o token emitido pelo passport, no entanto a tradução de código para o React implicou algumas mudanças. O react usa a # como um referencial para geral todas as routes existentes, sendo que para gerar o path em url, é necessário contar o número de caracteres posteriores ao "#" e até ao "?code". Foi necessário adaptar o código através da declaração location.hash, contornando o problema inicial, sendo que se tornou possível obter o token de acesso e posteriores dados do utilizador.

Principais obstáculos não foram ultrapassados

a) Comentários na Landing page

Foram implementados com sucesso o POST de comentários do lado da API e do REACT, no entanto a escrita dos comentários, estava a ser associada sempre ao último ID da aplicação. Existe um problema na passagem dos ID's por parte do REACT. Não sendo possível regularizar esta situação a tempo da entrega, optou-se por deitar abaixo a landing page inicialmente proposta no protótipo.

É de referir que a implementação dos comentários tem duas modalidades, sendo que é possível listar todas as aplicações e seus comentários associados a um utilizador, ou então os comentários de uma única aplicação.

b) Passagem de um vídeo dinâmico no Look&Learn

Foram feitos vários testes para correr um vídeo do youtube, numa primeira fase de forma estática e posteriormente de forma dinâmica. O link do vídeo estava a ser passado dinamicamente, mas depois o render não retorna o componente pretendido. Por esse motivo, optou-se por colocar um vídeo nas pastas de desenvolvimento do REACT, com leitura direta do player de vídeo por parte do servidor e de forma estática.

c) GOOGLE - Login

Devido a restrições temporais não foi possível implementar do lado do client-side, a autenticação através de redes sociais. No entanto, essa implementação está pronta do lado do servidor, e com retorno de uma string com todos os dados do utilizador, como se pode ver na figura 17.

Figura 17- Dados de retorno da rede social Google+

Sempre que o utilizador pede acesso via rede social, é feito um pedido de autorização por parte do cliente, e em caso afirmativo, retorna os dados do user.

Esta implementação derivou do seguinte pacote no github.

https://github.com/laravel/socialite

Este pacote necessitou de algumas adaptações nas componentes de segurança do servidor, nomeadamente permissões no SSL.

d) Bootstrap TOUR

Para o utilizador final se adaptar mais facilmente à aplicação Share&Learn, foi implementado em Javascript o código necessário para produzir *popups* descritivos das várias funcionalidades da aplicação. No entanto não foi possível adaptar o script para react, sendo que foram testadas duas abordagens:

A primeira era colocar o script na página compilada do react, provocando um ciclo infinito na página inicial.

A segunda abordagem foi criar um ficheiro jsx e com o script incorporado num render, seguindo o mesmo conceito do ajax no nosso login, no entanto também não foi possível correr o código. Como alternativa, o grupo de trabalho optou por **melhorar a "user experience"** do utilizador, com **feedbacks**, **hovers** nas aplicações, **retorno de animações de botões**, entre outros, de forma a facilitar o processo de adaptação e navegação na nossa plataforma.

WEBSITE de apoio - Por outro lado, como pretendemos também melhorar a plataforma pensando no utilizador e na sua experiência, optámos por desenvolver um **website de apoio**. Este website será utilizado na apresentação do projeto como apoio para a explicação do conceito das plataformas. Neste website é abordado o conceito das duas aplicações, as funcionalidades, as tecnologias utilizadas, uma pequena galeria com algumas fotos do Share&Learn, um Sobre que explica como surgiu a ideia desde projeto, o processo de desenvolvimento, a equipa de *developers,* factos interessantes sobre os testes realizados no user experience, e ainda ideias futuras que não foram implementadas mas que no futuro são uma sugestão a desenvolver. Por fim, existe uma seção que contém dois botões que redirecionam para o nosso projeto: plataforma Share&Learn e aplicação Look&Learn. URL – (http://shareandlearnapp.zapto.org/)

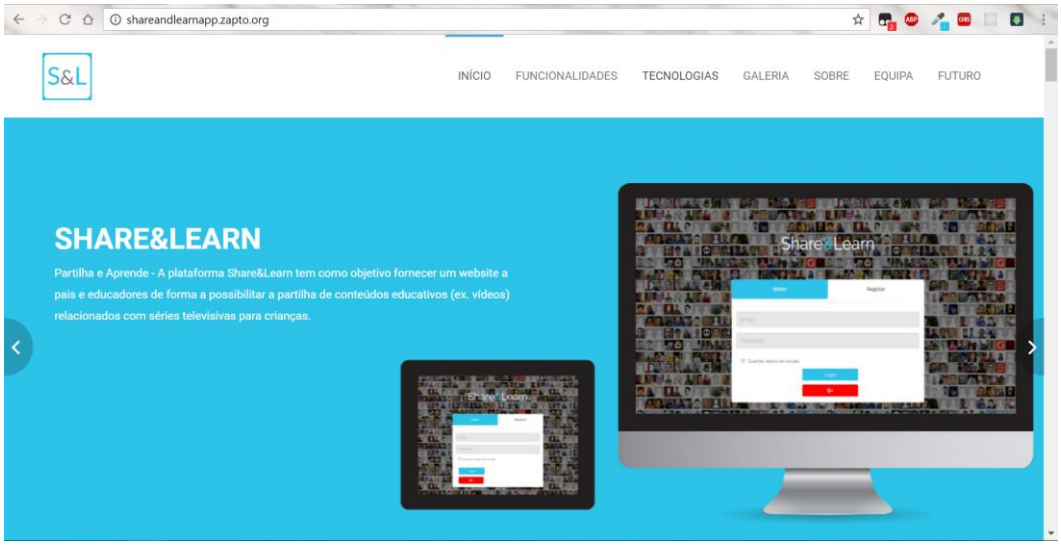

Figura 18 - Website Share&Learn

Ferramenta de gestão de projeto utilizada

a) Trello

Trello é uma aplicação para a web que serve para a gestão de projetos, onde se podem monitorizar tarefas a ser realizadas a curto, médio e longo prazo. É um serviço gratuito, que pode ser utilizada para uma diversidade de trabalhos, incluindo a gestão de projetos de software, planeamento de aulas, gestão de casos de escritório de advocacia, entre outros.

Esta ferramenta foi muito útil para a gestão do desenvolvimento do nosso trabalho. Criámos uma conta para o projeto Share&Learn e posteriormente várias colunas para designar as várias fases do projeto, nomeadamente "Para Desenvolver" (aquilo pretendíamos implementar mas ainda se encontrava a aguardar a conclusão de outras tarefas), "Em Desenvolvimento" (o que se estava a desenvolver no momento), "Para Testar" (aquilo que já tinha sido desenvolvido e se encontrava em fase de testes, na eventualidade de encontrar algumas falhas e possíveis erros a corrigir), "Feito" (o que já estava desenvolvido e testado), "Para mais tarde" (aquilo que não chegou a ser implementado.

Por outro lado, o Trello foi bastante útil na medida em que os professores podiam ir dando o feedback sobre as tarefas desenvolvidas, melhoramentos, e novas tarefas a executar.

Esta ferramenta permitiu ao grupo estar a par de todas as tarefas que tinha de realizar, melhorando a comunicação entre os membros e ajudando a recordar o que faltava ser implementado.

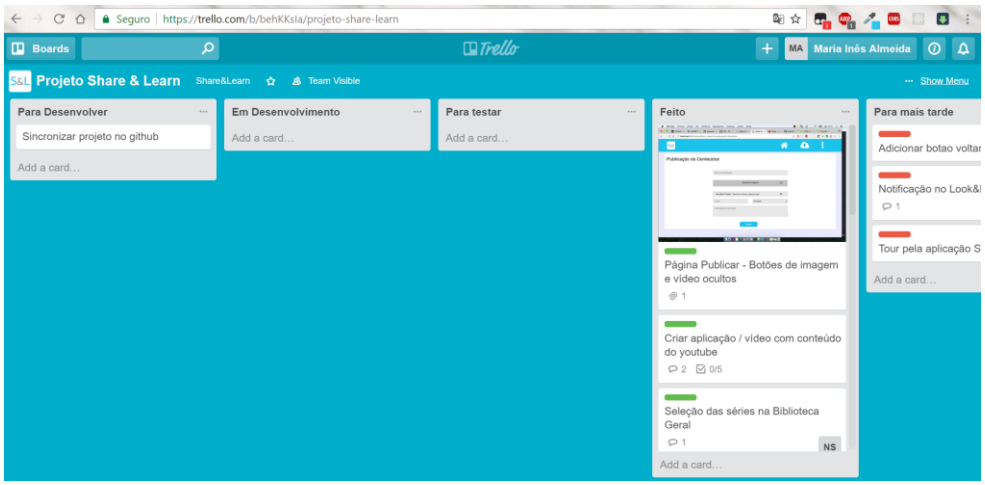

Figura 19 - Ferramenta Trello (https://trello.com/b/behKKsIa/projeto-share-learn)

Funcionalidades acrescentadas

1. Apagar Conteúdos

Na Biblioteca de Conteúdos, ao clicar no conteúdo específico vai para a página do Detalhe de Conteúdo. Nessa página é possível o utilizador apagar a aplicação que está a visualizar. Neste caso deveria ser o próprio utilizador a puder eliminar a sua própria aplicação. No entanto, essa implementação não foi bem conseguida e, dessa forma, colocámos na mesma a funcionalidade de apagar a aplicação embora disponível para todos os utilizadores.

Figura 20 - Página Detalhe de Conteúdo (apagar conteúdo)

2. Página do Autor

Foi criada uma página do autor da aplicação, para que outros utilizadores conseguissem ver quem criou o conteúdo e detalhes sobre essa pessoa. Assim, nesta página estão disponíveis informações pessoais sobre o autor dos conteúdos, bem como os conteúdos já partilhados pelo próprio.

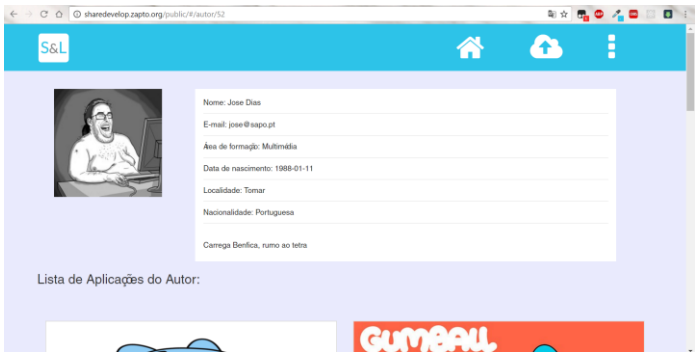

Figura 21 - Página do Autor (informação pessoal)

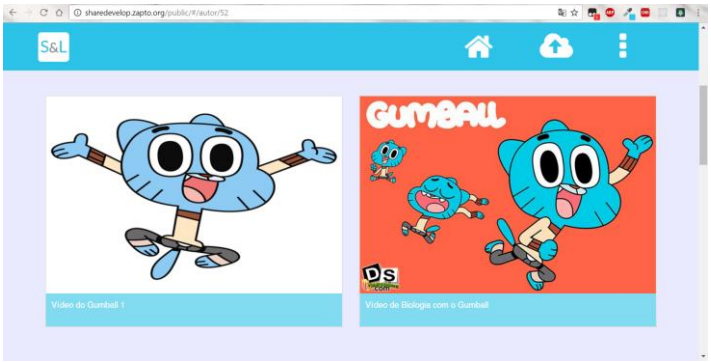

Figura 22 - Página do Autor (conteúdos partilhados)

Grupo G1: José Dias nº70832 | Maria Inês Almeida nº73123 | Nuno Soares nº58390

Conclusão

O trabalho produzido até agora e demonstrado neste relatório foi realizado na disciplina de Tecnologias Dinâmicas para a Internet e integra-se no projeto (PLB) que o grupo está a elaborar em conjunto com as disciplinas de STI e DCM e em parceria com a empresa Altice Labs. Todo o trabalho elaborado ao longo desta disciplina serviu para desenvolver a parte técnica do nosso projeto.

Foi bastante enriquecedor conhecer novas tecnologias que nos auxiliarem no desenvolvimento do projeto, nomeadamente o funcionamento do laravel, react e reflux.

A disciplina de TDI permitiu-nos aprofundar conhecimentos e essencialmente foi um grande desafio para a implementação de um MVP. Este projeto contribuiu para colocarmos em prática tudo o que fomos aprendendo a nível técnico ao longo das primeiras semanas de aulas.

Apesar do nosso projeto ter sofrido várias alterações ao longo do tempo, passando por diversas mudanças de funcionalidades e até mesmo de conceito, conseguimos ultrapassar várias dificuldades de implementação e tentar obter um MVP viável para apresentar.

Todo o trabalho envolvente permitiu-nos conhecer novas tecnologias de implementação o que exigiu um esforço superior por parte dos elementos do grupo.

Nesse sentido a nossa ideia inicial de “rede social” foi completamente modificada, mas o nosso MVP da plataforma (Share&Learn) de certa forma, tenta ir buscar essa ideia, permitindo que os utilizadores consigam publicar um vídeo, (um conteúdo), mas também ver informações associadas ao autor dessa partilha. Assim, é possível ver conteúdos partilhados por outros utilizadores, bem como informações pessoais sobre o mesmo.

Por fim, consideramos que esta disciplina foi fundamental essencialmente para o desenvolvimento técnico do nosso projeto.

Queremos agradecer todas as sugestões que foram feitas pelos professores que sempre nos acompanharam neste processo longo e trabalhoso.

De acordo com as nossas espectativas, consideramos ter conseguido obter um MVP viável bem conseguido.

Webgrafia

TAYLOR OTWELL. (2016). Laravel. 25 de Novembro 2016, from https://laravel.com/docs/5.3/authentication

Trello. (2017). Trello. 18 de Janeiro 2017, from https://trello.com/

GitHub. (2017). GitHub. 20 de Novembro 2017, https://github.com/mpociot/laravel-apidoc-generator

GitHub. (2017). GitHub. 28 de Novembro 2017, https://github.com/github/fetch

GitHub. (2017). GitHub. 31 de Novembro 2017,https://github.com/barryvdh/laravel-cors

GitHub. (2017) from https://www.nextofwindows.com/how-to-get-rid-of-cfakepath-in-ie-when-uploading-a-file-to-the-website-fix

GitHub. (2017). GitHub. 14 de Novembro 2017,https://github.com/laravel/socialite

GitHub. (2017). GitHub. 10 de Janeiro 2017,https://github.com/reflux/refluxjs

GitHub. (2016) from http://deviq.com/don-t-repeat-yourself/

Facebook. (2014 / 2015) 1 de Fevereiro 2017 from https://facebook.github.io/flux/docs/in-depth-overview.html#content

Facebook. (2014 / 2015) 1 de Fevereiro 2017 from https://facebook.github.io/react/docs/installation.html

Bootstrap. (2016) Bootstrap. 20 de Novembro 2017 from http://getbootstrap.com/components/

http://josedias.zapto.org/api/biblioteca-geral?nome_series=Dexter

http://shareandlearnapp.zapto.org/

https://mcmm-mi.slack.com/messages/tdi_api_solutions/