INTRODUÇÃO AO DESENVOLVIMENTO MODERNO PARA A WEB

Do *front-end* ao *back-end*: uma visão global!

HTML
CSS
BOOTSTRAP
JAVASCRIPT
HTTP(S)
FETCH
REST
NODE.JS
EXPRESS
SEQUELIZE
MYSQL
PASSPORT
GITHUB
HEROKU

Filipe Portela
Ricardo Queirós

FCA

EDIÇÃO
FCA – Editora de Informática, Lda.
Av. Praia da Vitória, 14 A – 1000-247 Lisboa
Tel: +351 213 511 448
fca@fca.pt
www.fca.pt

DISTRIBUIÇÃO
Lidel – Edições Técnicas, Lda.
Rua D. Estefânia, 183, R/C Dto. – 1049-057 Lisboa
Tel: +351 213 511 448
lidel@lidel.pt
www.lidel.pt

LIVRARIA
Av. Praia da Vitória, 14 A – 1000-247 Lisboa
Tel: +351 213 511 448 * Fax: +351 213 522 684
livraria@lidel.pt

ISBN edição impressa: 978-972-722-897-3
1.ª edição impressa: novembro 2018

Impressão e acabamento: Tipografia Lousanense, Lda. – Lousã
Depósito Legal n.º 447579/18
Capa: José M. Ferrão – *Look-Ahead*

ÍNDICE

OS AUTORES

Filipe Portela (www.filipeportela.com) – Licenciado em Tecnologias e Sistemas de Informação (2007), mestre em Engenharia e Gestão de Sistemas de Informação (2009) e doutorado em Tecnologias e Sistemas de Informação (2013) pela Universidade do Minho (UM). Professor Auxiliar convidado do Departamento de Sistemas de Informação da Escola de Engenharia da UM, onde tem lecionado e supervisionado vários alunos de mestrado, e Professor Adjunto convidado na Escola Superior de Media Artes e Design (ESMAD) do Politécnico do Porto (P.PORTO), sendo responsável por unidades curriculares na área da Programação Web.

Investigador Integrado do Centro de Investigação Algoritmi, onde desenvolveu o seu trabalho de investigação de pós-doutoramento sob o tópico Sistemas de Apoio à Decisão Inteligentes e *Pervasive*. A nível científico, possui diversas publicações indexadas nos seguintes tópicos de investigação: Sistemas de Informação *Pervasive*, Apoio à Decisão, Sistemas Inteligentes, *Big Data*, *Data Science*, Inteligência Artificial, *Business Intelligence*, *Data Mining* e *Knowledge Discovery*. Coeditor de diversos livros e revistas, coorganizador de várias conferências e *workshops* e revisor/membro dos comités de reconhecidas revistas, livros e conferências.

A sua experiência e o seu conhecimento têm potenciado o convite/participação em diversos eventos, projetos e júris técnico-científicos, contribuindo assim para a disseminação do conhecimento junto da sociedade e dos cidadãos. Este facto teve o ponto alto em janeiro de 2018, altura em que fundou a *startup* tecnológica IOTech – Innovation on Technology (www.iotech.pt).

Ricardo Queirós (www.ricardoqueiros.com) – Doutorado em Ciências de Computadores pela Faculdade de Ciências da Universidade do Porto (FCUP). Docente na Escola Superior de Media Artes e Design (ESMAD) do Politécnico do Porto (P.PORTO), sendo responsável por disciplinas na área da Programação de Computadores, focada para o desenvolvimento de aplicações e serviços para os ambientes Web e os dispositivos móveis. Membro efetivo do *Center for Research in Advanced Computing Systems* (CRACS), uma unidade de investigação do laboratório associado INESC TEC, onde desenvolve atividade científica nas áreas de Integração de Sistemas, Interoperabilidade entre Sistemas de E-learning e Gamificação em Ambientes de Aprendizagem de Programação de Computadores. Membro da Comissão Científica da Unidade de investigação da ESMAD (uniMAD).

A nível de produção científica, destaca-se o trabalho em vários projetos nacionais e internacionais e mais de uma centena de publicações. Integra várias comissões científicas e de revisão de várias conferências e revistas de especialidade, bem como a equipa de júris do Concurso Internacional de Desenvolvimento de Aplicações Móveis (IEEEmadC), promovido pelo *Institute of Electrical and Electronics Engineers* (IEEE).

Autor dos seguintes livros relacionados com programação de aplicações para dispositivos móveis e para a Web, publicados pela FCA: *Android – Introdução ao Desenvolvimento de Aplicações*, *Desenvolvimento de Aplicações Profissionais em Android*, *Introdução ao Desenvolvimento de Jogos em Android* (em coautoria), *Android – Bases de Dados e Geolocalização, Android – Desenvolvimento de Aplicações com o Android Studio, Criação Rápida de Sites Responsivos com o Bootstrap* e *Android Profissional – Desenvolvimento Moderno de Aplicações*.

PREFÁCIO

Inesperadamente, a minha primeira sensação ao começar a ler este livro foi de nostalgia. O "moderno" no seu título fez-me tomar consciência de que já se passaram quase 25 anos desde que estive, como o leitor, avidamente à procura de saber mais sobre programação na World Wide Web. Na altura, ainda não era, simplesmente, a Web; na melhor das hipóteses, seria WWW, sigla propositadamente tão impronunciável como o nome, que apenas simplificava a escrita. O melhor que se conseguia eram algumas páginas da própria WWW, com descrições muito concisas, de introdução ao HTML, para formatar páginas no cliente, ou sobre a ligação de programas ao servidor usando o *Common Gateway Interface*. Linguagens como o CSS (*Cascading Style Sheets*) ou o JavaScript, para não falar de livros bem escritos, organizados, abrangentes, aprofundados e em português, eram ainda uma promessa.

Sem dúvida que este livro é moderno. Não apenas pelas tecnologias que cobre e pelas versões que usa como referência, mas também pela abordagem integrada que delas faz. É importante lembrar que a Web não é um sistema homogéneo, criado por um único grupo de pessoas de uma empresa ou instituição; a Web, tal como a conhecemos hoje, é um ecossistema formado por linguagens, protocolos e programas, que evoluíram em conjunto. Se, aos olhos dos utilizadores, formam um todo harmonioso numa aplicação é porque foram bem orquestradas por quem as desenvolveu. Por isso, este livro começa por dar essa visão de conjunto, propondo uma aplicação concreta – o projeto WebConference –, no qual o leitor tem oportunidade de começar a compreender o nicho de cada um dos habitantes desse ecossistema.

Ao longo do seu crescimento, a Web deixou de ser a "velhinha" WWW: diferenciou-se e especializou-se. As linguagens que operam nos navegadores e os clientes Web relacionam-se mais entre si, e outro tanto acontece nos servidores Web. Hoje em dia, é habitual um programador Web apresentar-se como sendo de *front-end* ou de *back-end* consoante o grupo de tecnologias que domina. Com a especialização, surgem os especialistas, como é o caso dos autores deste livro. O conhecimento especializado de cada um numa destas áreas permite que se complementem para nos darem uma visão da Web moderna, que, apesar de abrangente, não deixa de ser aprofundada.

Já se percebeu que a nostalgia durou apenas um instante. Rapidamente, deu lugar ao entusiasmo de seguir o Filipe e o Ricardo nesta proposta de (re)descobrir o desenvolvimento Web, tendo-os como guias. A si, que está agora a dar os primeiros passos nesta forma de programação, peço-lhe que fixe bem o momento em que começa a ler este livro. É bem possível, e é o que lhe desejo, que daqui a 25 anos também se lembre dele com um pouco de nostalgia.

José Paulo Leal

Professor Auxiliar no Departamento de Ciências de Computadores
da Faculdade de Ciências da Universidade do Porto

0 INTRODUÇÃO

Nos últimos anos, a Web tem sofrido uma evolução considerável. De facto, hoje em dia, a programação de uma aplicação Web envolve o conhecimento e a manipulação de um número considerável de linguagens, *frameworks* e ferramentas. O que será algo normal para quem trabalha, no dia a dia, neste contexto poderá ser um fator inibidor para quem pretende (re)entrar neste maravilhoso mundo da Web. Este livro tem um duplo objetivo: por um lado, organiza de uma forma lógica toda esta panóplia de tecnologias que orbitam à volta da Web; por outro, desmistifica a entrada neste mundo a todos aqueles que se querem iniciar na criação de aplicações para a Web ou a todos os que já programaram para a Web, mas que agora sentem dificuldades em regressar.

0.1 O QUE POSSO ENCONTRAR NESTE LIVRO?

O livro tem como principal objetivo (re)introduzir os leitores no mundo do desenvolvimento para a Web. Neste livro, são apresentados os princípios básicos associados ao desenvolvimento para a Web, divididos em *front-end* e *back-end*.

Na parte do *front-end* (Capítulos 3 a 5), são introduzidos os conceitos de estruturação, estilização e interação através das suas principais linguagens, respetivamente, HTML, CSS e JavaScript. Na parte do *back-end* (Capítulos 6 a 8), é feita uma introdução aos servidores Web, às bases de dados e aos serviços na Web. De forma a consolidar todos os conceitos teóricos, é implementado um projeto prático completo (Capítulos A e B).

0.2 PÚBLICO-ALVO

Este livro foi escrito tendo como público-alvo todos aqueles que, com conhecimentos básicos de programação, pretendem (re)entrar no mundo da Web, nomeadamente:

- Professores e alunos de cursos/disciplinas de Informática, Ciência de Computadores e Tecnologias Web;
- Profissionais de empresas tecnológicas (p. ex., programadores Web, gestores de projetos, Web designers).

É também uma obra fundamental para todos aqueles que pretendem colocar-se rapidamente a par de todas as novidades introduzidas nos últimos anos a nível do desenvolvimento Web, incluindo todos os que programaram para a Web há mais de 10 anos e que queiram reciclar os seus conhecimentos.

Apesar de a obra partir do pressuposto de que o leitor já tem conhecimentos básicos na programação para a Web, tal não é obrigatório para a sua leitura. Na realidade, são introduzidos, de uma forma sustentada, todos os conceitos necessários para a criação básica de aplicações Web.

0.3 CONVENÇÕES

Ao longo deste livro, optou-se por seguir um conjunto de convenções que facilitam a interpretação do texto e do código apresentados. Assim, todos os excertos de código são apresentados no formato seguinte:

```
let name = "Ricardo"
console.log(`Olá ${name}!`)
```

Por sua vez, as notas ou observações importantes poderão ser encontradas no interior de uma secção semelhante à seguinte:

Nota importante

Esta é uma nota ou observação importante.

0.4 ORGANIZAÇÃO DO LIVRO

Este livro está organizado em três partes:

- **Parte I: Programação Web** – introdução às tecnologias Web (Capítulo 1) e apresentação do projeto prático WebConference (Capítulo 2);
- **Parte II: *Front-end*** – introdução das principais linguagens de desenvolvimento *front-end*, nomeadamente a linguagem de marcação HTML (Capítulo 3), a linguagem de estilização CSS (Capítulo 4) e a linguagem de programação JavaScript (Capítulo 5);
- **Parte III: *Back-end*** – introdução dos principais componentes do lado do servidor, nomeadamente o servidor Web (Capítulo 6), as bases de dados (Capítulo 7) e os serviços na Web (Capítulo 8).

As Partes II e III são complementadas por um capítulo de implementação, respetivamente, do *front-end* (Capítulo A) e do *back-end* (Capítulo B), onde é aplicado – projeto prático WebConference – todo o conhecimento assimilado das linguagens e dos componentes explanados nos capítulos anteriores (Figura 0.1).

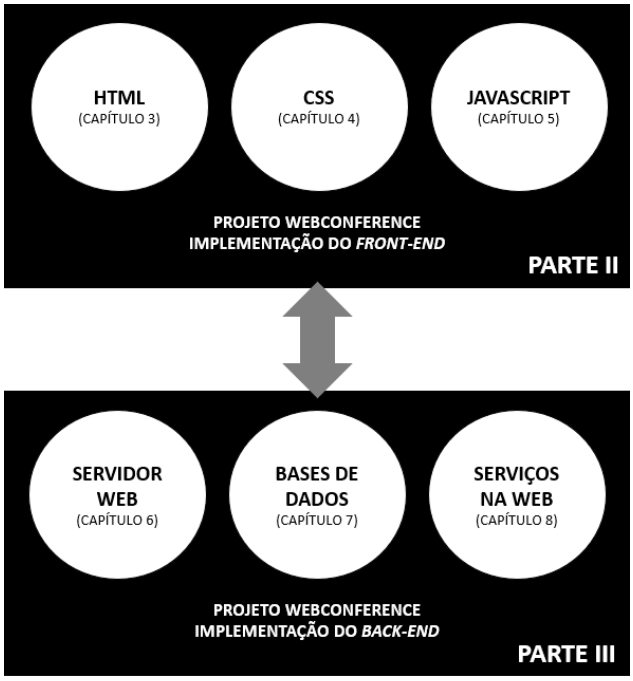

FIGURA 0.1 – Estrutura das Partes II e III da obra

Os capítulos podem ser lidos sequencialmente ou, se o leitor assim o preferir, de forma alternada (isto é, sem respeitar a ordem de capítulos apresentada), dado que as poucas dependências entre os mesmos estão devidamente identificadas. O leitor poderá ler apenas uma parte da obra (*front-end* ou *back-end*) ou, até mesmo, um capítulo de uma determinada linguagem ou tecnologia, sem prejuízo da sua compreensão.

0.5 SUPORTE

Caso o leitor encontre informação que lhe pareça incorreta ou tenha sugestões em relação ao conteúdo de alguma secção do livro, não hesite em enviar aos autores um *e-mail* com as suas questões e/ou considerações: sobre a parte de *front-end* para ricardo.queiros@gmail.com; e sobre a parte de *back-end* para filipeportela@iotech.pt. Eventuais alterações e uma errata serão publicadas na página do livro no *site* da editora, em www.fca.pt. Aqui, na área Downloads, poderá também aceder a todo o código-fonte do projeto WebConference.

```
      <article>
        <h3>Curso de Jersey</h3>
        <p>Neste curso ...</p>
      </article>
    </section>
  <footer>
    <p>Ricardo Queirós. Todos os direitos reservados.</p>
  </footer>
</body>
```

Segue-se uma breve explicação do trecho de código anterior. Logo no início, é definido o cabeçalho da página, no qual se coloca o título principal. Depois, são criadas várias secções (elementos `section`) e dentro de cada uma delas poderá haver vários elementos do tipo `article`. Aqui, usámos `section` para delimitar a secção de cursos referentes a cada linguagem e `article` para definir cada curso. No rodapé da página podemos adicionar informações sobre autoria e direitos autorais, *links* adicionais, etc. O resultado final é apresentado na Figura 3.6.

FIGURA 3.6 – Página HTML de exemplo construída com elementos semânticos

Para finalizar esta secção, deixamos um conselho: lembre-se de que quando escrevemos uma página Web estamos a preparar um conteúdo que será consumido por diferentes "clientes", entre os quais outros programadores, *browsers*, algoritmos de indexação de motores de busca e, até mesmo, ferramentas de mineração de texto. Portanto, ao escrever código HTML, preocupe-se também com a semântica dos elementos que nele existem. O uso correto de elementos semânticos é fundamental para a construção de páginas Web modernas e fáceis de compreender e manter.

3.3 FORMULÁRIOS

Um formulário é usado para colecionar dados introduzidos pelo utilizador e para os enviar, por norma, a um servidor. Para definir um formulário em HTML, recorremos ao elemento `<form>`:

```
<form>
   <!-- Elementos de formulário -->
</form>
```

O elemento `<form>` pode incluir vários elementos de formulário:

- **`<input>`** — um campo de entrada de dados;
- **`<textarea>`** — uma caixa de texto alargada;
- **`<select>`** — uma caixa de listagem com múltiplos itens;
- **`<button>`** — um botão para desencadear ações;
- **`<label>`** — um rótulo que pode ser associado a um elemento de formulário;
- **`<fieldset>`** — um contentor para agrupar elementos relacionados.

O elemento mais importante de um formulário é o `<input>`, que é responsável pela entrada de dados ou por desencadear ações específicas, como a submissão de um formulário. Para definir a sua natureza, é necessário incluir o atributo `type`, caso contrário, será um campo básico de entrada de texto. São suportados vários tipos de dados de entrada (p. ex., texto, números, datas).

O próximo exemplo mostra como criar um formulário para receber, de parte do utilizador, um *e-mail* e uma *password*:

```
<form>
   <input type=”email”><br>
   <input type=”password”><br>
</form>
```

O excerto anterior é bastante limitado e pouco legível, uma vez que não fornece rótulos que, normalmente, acompanham os elementos `<input>`. O elemento `<label>` define um rótulo para um elemento `<input>` e fornece uma melhoria de usabilidade para quem utilize o rato, já que, ao clicar no texto incluído no elemento `<label>`, o foco passa para o elemento `<input>`:

```
<form>
  <label for="myEmail">E-mail:</label>
  <input id="myEmail" type="email"><br>
  <label for="myPassword">Password:</label>
  <input id="myPassword" type="password">
</form>
```

Neste exemplo, a regra CSS usa um seletor por tipo de elementos que seleciona o elemento `<body>` da página HTML e aplica a cor de fundo vermelha.

4.2.2.3 INTEGRAÇÃO EXTERNA

Para definir regras externas é necessário criar um ficheiro novo com extensão `.css`. Depois, devem ser incluídas nesse ficheiro todas as regras desejadas. Por fim, falta associar o ficheiro HTML com o ficheiro de estilos criado. Para tal, inclua um elemento `<link>` dentro do elemento `<head>` da página HTML e use o atributo `href` para referenciar o ficheiro com as regras CSS:

```
<head>
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
```

Devem ser usadas regras externas, de forma a promover a reutilização de declarações entre documentos e a consistência em *sites* multipágina.

Outra das grandes vantagens é a de separar o que é estrutural (HTML) do que é formatação (CSS). Esta separação de responsabilidades em ficheiros independentes promove a legibilidade, a manutenção e a modularização de todo o código de um *site*.

Outro aspeto a reter é a flexibilidade desta abordagem. Se tiver vários ficheiros HTML, poderá colocá-los todos a apontar para o mesmo ficheiro CSS e, assim, mudar a aparência de um *site* inteiro, alterando apenas um ficheiro.

4.3 CASCATA

Um das partes mais importantes no funcionamento das CSS é perceber o conceito de cascata. Aliás, a palavra *cascading*, no nome da linguagem, está lá por alguma razão. Em primeiro lugar, é importante reter que, caso não sejam definidas quaisquer regras CSS, o *browser* aplicará as suas próprias regras em todos os elementos HTML. Por outro lado, se o programador definir regras CSS, estas terão precedência sobre as regras definidas por predefinição pelo *browser*.

Como vimos na secção 4.2, as regras CSS podem estar dispersas num *site* HTML (integração local, interna ou externa). Sendo assim, o que acontece quando duas regras selecionam o mesmo elemento? Se as propriedades dessas regras forem diferentes, estas serão agregadas; se forem iguais, serão aplicadas as regras em **cascata**.

Apresenta-se um exemplo para o primeiro caso:

```
<head>
  <style>
    p {
      color:red;
```

```
    }
  </style>
</head>
<body>
  <p style="background-color: yellow">ESMAD</p>
</body>
```

Neste exemplo, temos duas regras associadas de forma diferente (integrações interna e local), mas que selecionam o mesmo elemento `<p>`. Como as suas declarações têm propriedades diferentes, ambas são aplicadas ao elemento, ficando o parágrafo com cor de fundo amarela e cor do texto vermelha.

Se as propriedades forem iguais, a aplicação das regras seguirá um critério mais complexo, chamado **cascata**. Atente ao próximo exemplo:

```
<head>
  <style>
    body p {
      color:blue;
    }
    p {
      color:red;
    }
  </style>
</head>
<body>
  <p>ESMAD</p>
</body>
```

Como ambas as regras internas selecionam o mesmo elemento e possuem a mesma propriedade, qual é a regra que deve ser aplicada? Neste caso, são aplicadas as seguintes regras em cascata, de forma a resolver a ambiguidade:

1) **Cálculo de especificidade** – as mais específicas prevalecem.
2) **Ordem de especificação** – se forem iguais, valerá a última.
3) **Anotação** `!important`.

4.3.1 CÁLCULO DE ESPECIFICIDADE

O cálculo da especificidade é feito quando duas regras selecionam o mesmo elemento e têm definida a mesma propriedade.

Não se esqueça de que se a comparação for feita entre as propriedades definidas pelo programador e as que existem por predefinição no *browser*, prevalecerão as do programador.

No cálculo de especificidade, as regras são ordenadas pelo grau de especificidade do seletor. As regras que se aplicam a vários tipos são gerais e as que se aplicam a um só tipo são específicas, prevalecendo em relação às gerais. A especificidade de um seletor é dividida em quatro níveis constituintes, ilustrados na Figura 4.2.

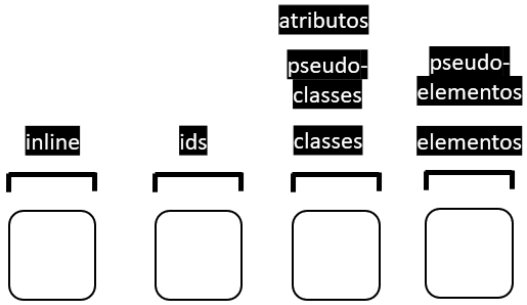

FIGURA 4.2 – Níveis constituintes para cálculo do grau de especificidade de um seletor

Relembrando as regras às quais queremos calcular a especificidade, temos:

```
<style>
  body p {
    color:blue;
  }
  p {
    color:red;
  }
</style>
```

No primeiro nível (N1), e como ambas as regras não são locais (*inline*), atribuímos o valor 0 (Tabela 4.3).

REGRAS	N1	N2	N3	N4
`body p { color:blue; }`	**0**	-	-	-
`p { color:red; }`	**0**	-	-	-

TABELA 4.3 – Tabela de níveis para cálculo de especificidade do N1

No segundo nível (N2), contabilizámos o número total de seletores por identificador. Como não temos esse tipo de seletores, atribuímos o valor 0 a ambas as regras (Tabela 4.4).

REGRA	N1	N2	N3	N4
`body p { color:blue; }`	0	**0**	-	-
`p { color:red; }`	0	**0**	-	-

TABELA 4.4 – Tabela de níveis para cálculo de especificidade do N2

5 JAVASCRIPT

O JavaScript é, hoje em dia, uma das linguagens de programação mais populares a nível mundial. Inicialmente projetado para o desenvolvimento Web *front-end*, começou, na última década, a alargar o seu âmbito para o *back-end*, com o aparecimento do Node.js, sendo já usado na construção de aplicações *desktop* e móveis. Este capítulo apresenta a sintaxe do JavaScript e destaca os conceitos fundamentais desta linguagem, tais como funções, *arrays*, objetos, *Document Object Model* (DOM) e comunicação com o servidor.

5.1 INTRODUÇÃO

O JavaScript foi criado por Brendan Eich, em 1995. Na altura, esta linguagem de *scripting* foi desenvolvida para integrar o *browser* Netscape Navigator 2. Inicialmente, chamava-se LiveScript, mas como a linguagem Java era muito popular naquela época, foi decidido posicionar a nova linguagem como uma "irmã mais nova" do Java, tendo-se mudado o seu nome para JavaScript. Contudo, à medida que esta foi evoluindo, tornou-se uma linguagem totalmente independente, não tendo agora qualquer relação com o Java.

Em 1997, linguagem JavaScript tornou-se uma especificação da *European Computer Manufacturers Association* (ECMA-262), denominada ECMAScript (ou ES). Desde então, a Ecma Internacional tem publicado várias edições da especificação, conforme ilustra a Figura 5.1.

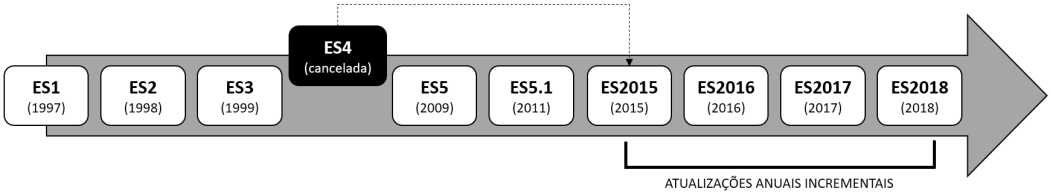

FIGURA 5.1 – Evolução da especificação ECMAScript

Convém frisar que a evolução da especificação nem sempre foi linear. Por exemplo, a quarta edição foi abandonada devido a diferenças políticas relativamente à complexidade da linguagem. Nessa quarta edição, muitas das funcionalidades propostas foram descartadas, ao contrário de outras que foram incorporadas na sexta edição, em 2015.

A partir desta edição, a ECMAScript mudou o seu nome de ES6 para ES2015, já que a Ecma International decidiu lançar edições anuais da especificação, passando a nomeá-las com base no ano em que são lançadas. A última edição da ECMAScript é a nona, tendo sido lançada em (junho de) 2018[1].

O JavaScript é uma linguagem de *scripting* que está em conformidade com a especificação ECMAScript, ou seja, é uma implementação desta.

A ECMAScript possui outras implementações, tais como o JScript e o ActionScript.

O JavaScript popularizou-se como uma linguagem para navegadores Web, dada a sua robustez e a total integração com o HTML e o CSS. Hoje em dia, a linguagem é usada noutros ambientes – designados por JavaScript *runtimes* –, em que o código JavaScript é executado por um motor (*engine*).

O *runtime* fornece os objetos programáveis nos quais o JavaScript pode operar e trabalhar. Tipicamente, os motores JavaScript estão incluídos em navegadores Web (p. ex., V8, no Chrome; SpiderMonkey, no Firefox; Chakra, no Edge), mas, hoje em dia, podem ser encontrados noutros *runtimes*, como no Node.js, que também usa o motor V8.

5.2 SINTAXE BÁSICA

Nesta secção, faz-se uma breve apresentação da sintaxe do JavaScript, nomeadamente a definição de variáveis, os tipos de dados, as conversões, os operadores e as estruturas cíclicas e condicionais.

A maior parte dos exemplos apresentados neste capítulo poderá ser testada através da consola de um *browser*. Por exemplo, abra o Chrome e pressione a tecla **F12** (ou **Ctrl+Shift+I**) para aceder às ferramentas do programador. Depois, no separador **Console**, escreva os exemplos de código e confirme os resultados (Figura 5.2).

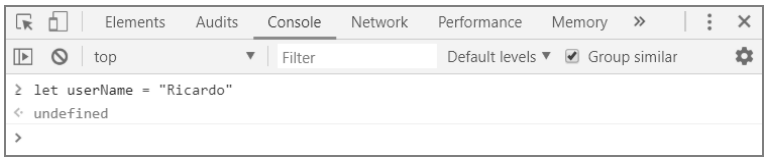

FIGURA 5.2 – Consola do *browser* Chrome

[1] *Link*: http://www.ecma-international.org/publications/standards/Ecma-262.htm.

A IMPLEMENTAÇÃO DO PROJETO: *FRONT-END*

Neste capítulo, é descrita a implementação da parte *front-end* do projeto WebConference (apresentado na generalidade no Capítulo 2). Este projeto tem como objetivo gerir e apresentar toda a informação relativa a uma conferência fictícia sobre desenvolvimento Web que vai decorrer em Portugal. A parte de *front-end* é composta por duas páginas: uma *Single Page Application*, com acesso público, que exibe toda a informação relativa à conferência, desde a possibilidade de se registar no evento até à informação sobre oradores e *sponsors*; e uma página de *back-office*, de acesso condicionado, que vai permitir a um administrador gerir toda a informação respeitante a participantes, oradores e *sponsors*.

A.1 INTRODUÇÃO

O *site* da aplicação WebConference é composto por duas partes: *front-end* e *back-end*. O *front-end* é responsável por apresentar os dados da conferência ao utilizador. O *back-end* terá como missão processar os pedidos feitos pelos utilizadores a partir do *front-end* e persistir a informação numa base de dados (a implementação em *back-end* será desenvolvida no Capítulo B, na Parte III do livro). Este capítulo descreve toda a construção do *front-end*, que poderá ser acedido por dois tipos de utilizadores: o utilizador "normal" e o administrador (Figura A.1).

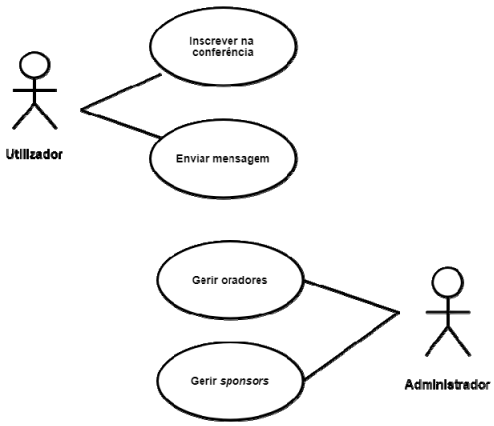

FIGURA A.1 – Diagrama de casos de uso da aplicação WebConference

O utilizador "normal" terá acesso a uma *Single Page Application* (SPA) e a toda a informação do evento, no qual se poderá inscrever e enviar mensagens. O administrador terá as mesmas permissões, mas, além disso, poderá aceder a uma área reservada na qual irá gerir os oradores e os *sponsors* da conferência.

Para compreender melhor todos os componentes do *front-end* e o respetivo fluxo de pedidos, é apresentado, na Figura A.2, um diagrama geral do projeto WebConference.

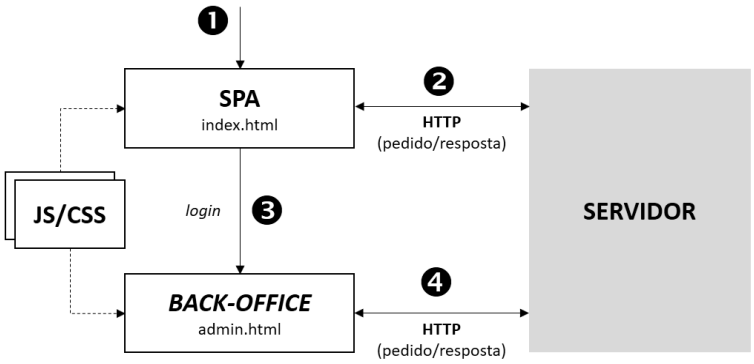

FIGURA A.2 – Componentes do *front-end* e fluxo de pedidos

Como pode ser observado, a parte de *front-end* é composta por duas páginas HTML e vários ficheiros auxiliares (JavaScript – JS e *Cascading Style Sheets* – CSS). As duas páginas HTML vão permitir a interação com os utilizadores, da qual resultarão pedidos ao servidor e as respetivas respostas.

O fluxo é simples: o utilizador entra na página principal (**1**, na Figura A.2); ao carregar a página no *browser*, vários pedidos são feitos automaticamente ao servidor (**2**, na Figura A.2), entre os quais obter a listagem dos oradores e dos *sponsors* que estejam armazenados na base de dados. O utilizador poderá também entrar numa área reservada, mediante credenciais de administrador, onde irá aceder a uma nova página de administração (**3**, na Figura A.2). Nesta página, o administrador poderá gerir os oradores e *sponsors* do evento, bem como definir configurações gerais. Esta gestão irá obrigar a novos pedidos ao servidor de forma a que a informação fique persistida na base de dados (**4**, na Figura A.2) e seja refletida nas interações posteriores dos utilizadores.

Os pedidos ao servidor serão baseados na API WebConference (que será desenvolvida no Capítulo B). A Tabela A.1 apresenta os principais *endpoints* a serem usados pela SPA e pelo *back-office*. O *Uniform Resource Locator* (URL) base dos pedidos é https://fcawebbook.herokuapp.com.

A base de dados no servidor foi desenhada para suportar várias conferências. Uma vez que esta aplicação é específica para a WebConference, não faz sentido que a sua identificação seja dinâmica. Por essa razão, em diversos surge *endpoints* o valor 1, que corresponde ao identificador numérico da WebConference na base de dados.

6.4 *FRAMEWORK* EXPRESS

O Express e o Node.js deram ao JavaScript uma nova funcionalidade de *back-end*. Pela primeira vez, os programadores podem criar software no lado do servidor utilizando JavaScript. A combinação de ambos permite a construção de uma aplicação completamente baseada naquela linguagem.

As aplicações são desenvolvidas do lado do servidor com Node.js e, em seguida, publicadas com o Express. Como o Node.js não foi criado para desenvolver aplicações, o Express é capaz de desenvolver uma camada na estrutura interna e publicar as funções necessárias para construir um *site* ou uma aplicação.

O Express apresenta-se como:

- ***Framework* do lado do servidor e aplicação *mobile*** – permite criar aplicações de uma página, multipágina, híbridas, *mobile* e Web, bem como desenvolver funcionalidades de *back-end* para aplicações Web e API;
- **Linguagem** – usa o JavaScript;
- ***Templating*** – contém dois motores de modelos, o Jade e o EJS, que facilitam o fluxo de dados numa estrutura de *site* e permitem utilizar outros modelos;
- **MVC** – suporta a arquitetura MVC;
- **Plataforma** – usa o Node.js;
- **Sistemas operativos** – é multiplataforma, por isso não se limita a um sistema operativo;
- **Gerador de código Express** – permite criar rapidamente aplicações complexas.

Entre as características desta *framework*, destacam-se as seguintes:

- Código minimalista;
- *Routing* robusto;
- Facilmente integrável com os principais *template engines* (motores de *template*);
- Trabalha com o conceito de *middleware* (secção 6.4.1);
- Focado em elevada *performance*;
- Adota padrões e boas práticas de serviços *Representational State Transfer* (REST);
- Permite *content negotiation* (RESTFul).

Content negotiation é um mecanismo HTTP que permite aceder a diversas versões de documentos com o mesmo *Uniform Resource Identifier* (URI).

6.4.1 *MIDDLEWARE*

O *middleware* é representado por um conjunto de funções que são invocadas pela camada de *routing* do Express antes de serem manipuladas. Como demonstrado na Figura 6.9, o *middleware* é colocado no meio de um pedido inicial (cru) e da rota pretendida.

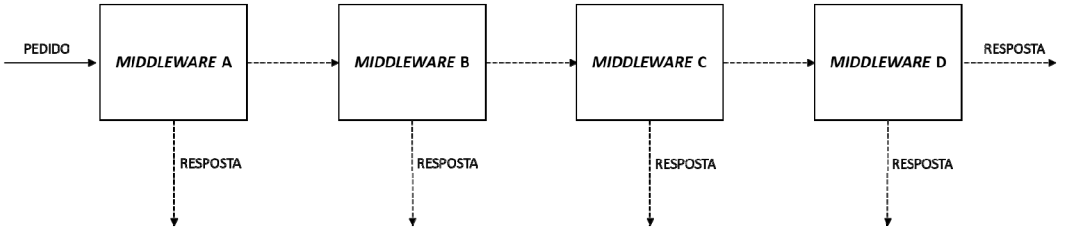

FIGURA 6.9 – *Middleware*

Estas funções são, muitas vezes, definidas como ***middleware stack*** (pilha de *middleware*), uma vez que são sempre invocadas pela ordem em que são adicionadas (***first in, first out*** – **FIFO**). O *middleware* funciona como um filtro das requisições efetuadas: ao passarem pelo *middleware*, os pedidos podem ser modificados antes de serem entregues ao processo seguinte. Trata-se da ideia central por detrás do processo de requisição e *routing* do Express.

Compreendendo o funcionamento do *middleware*, é possível criar aplicações mais fáceis de manter e com menos código. No momento em que um pedido é recebido por uma aplicação do Express, este aciona várias funções referidas como *middleware*.

É fundamental perceber que, embora a `app.use()` seja chamada para cada método/verbo de HTTP, o *middleware* é processado após receber a ordem enviada pela manipulação de rota. Nesse sentido, é importante salientar os seguintes aspetos:

- **Ordem dos pedidos** – o `GET` é executado antes do *middleware* e o `POST` após o *middleware*;
- **Esquecimento do `next()`** – se a solução não atualiza ou não está a responder, então o código não está a chamar o `next();`
- **Sobreposição de propriedades** – os argumentos `request` e `response` correspondem à mesma instância para todos os *middlewares* e rotas. Tal significa que existem dois *middlewares* que modificam as propriedades do objeto de formas diferentes. Estas alterações podem causar erros na aplicação, pelo que é importante estar-se atento às modificações que o *middleware* faz nas propriedades.

A Tabela 7.2 apresenta os comandos de SQL mais utilizados.

OPERADOR/COMANDO	DESCRIÇÃO	EXEMPLO
`BETWEEN`	Permite especificar um intervalo de valores, sendo inclusivo.	`SELECT * FROM professor WHERE codprof BETWEEN 1 AND 3`
`IN`	Permite especificar um conjunto de valores.	`SELECT * FROM professor WHERE codprof IN (1,2,4);`
`LIKE`	Permite filtrar *strings*. O elemento `%` representa um ou mais caracteres e o elemento _ um caráter.	`SELECT * FROM professor WHERE nome LIKE '_nome%';`
`AND` e `OR`	`AND` (e) tem maior precedência sobre `OR` (ou).	`SELECT * FROM professor WHERE codprof BETWEEN 1 AND 2 AND (ativo = 1 OR ativo = 2);`
`ORDER BY -`	Permite ordenar os resultados da *query* por ordem ascendente (`asc`) ou descendente (`desc`).	`SELECT column(s) FROM table WHERE predicate(s) ORDER BY column(s) [ASC | DESC]`
`GROUP BY -`	Permite agrupar os resultados da *query* por um determinado campo. Aplica-se, geralmente, às funções (p. ex., `max`, `min`, `sum`).	`SELECT column(s), function FROM table WHERE predicate(s) GROUP BY column(s)`

TABELA 7.2 – Comandos de SQL mais usados

7.4 MAPEAMENTO DE OBJETOS RELACIONAIS

A nível de implementação e utilização das BD, existem duas abordagens: a relacional e a orientada a objetos.

Na **abordagem relacional**, prevalecem os princípios matemáticos e a finalidade de armazenar e gerir corretamente os dados. A linguagem SQL é utilizada para dizer à BD "o que" fazer e não "como" fazer. Por seu lado, na **abordagem orientada a objetos** trabalha-se com classes e métodos, ou seja, opera-se com os fundamentos e os princípios da Engenharia de Software, que nos dizem "como" fazer.

O mapeamento de objetos relacionais (***object-relational mapping*** – **ORM**) é, justamente, a ponte entre estes dois mundos, ou seja, é o que vai permitir que os dados sejam armazenados na BD. Para tal, é necessário efetuar um mapeamento dos objetos para as tabelas da BD.

Sendo este um livro introdutório, iremos recorrer à linguagem tradicional do SQL. No entanto, um dos exemplos do projeto utilizará uma ferramenta ORM (sistema de *login*).

7.4.1 *OBJECT-RELATIONAL MAPPING* (ORM)

É uma técnica de mapeamento de objetos relacionais que permite fazer uma relação destes com os dados que representam. O ORM converte os dados provenientes de sistemas incompatíveis em BD relacionais utilizando as linguagens de programação orientadas a objetos, permitindo criar BD de objetos virtuais que podem ser utilizadas a partir daquelas.

Como aspeto positivo, o ORM reduz a quantidade necessária de código escrito, tornando o software mais robusto (menos linhas de código e menos erros num programa). Quanto a aspetos negativos, algumas ferramentas ORM não permitem o processamento de dados em massa.

Os procedimentos armazenados têm melhor desempenho, mas são fixos. Imagine um cliente que tem uma conta na BD. Utilizando o ORM, essa conta estará diretamente relacionada com o cliente. Assim, será possível utilizar o cliente e a conta como dois objetos interligados. A Figura 7.6 faz o mapeamento ORM entre a tabela **Cliente** e o objeto **Cliente**.

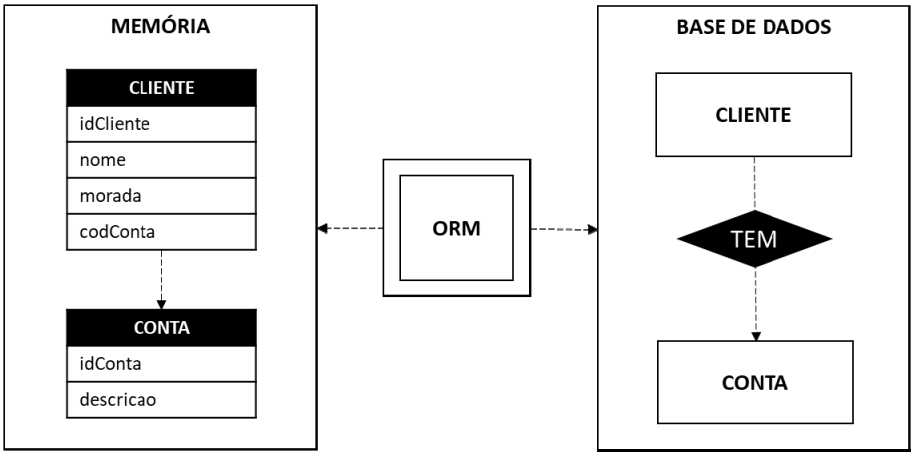

FIGURA 7.6 – Exemplo de mapeamento ORM

Os excertos de código seguintes, que recorrem ao exemplo já apresentado, fazem o mapeamento entre SQL e ORM, ou seja, ambas as instruções permitem imprimir os dados do professor com o código 2. O código ORM cria um objeto `p` do tipo `professor`, utilizando o método `get` para passar a condição de seleção. O repositório representa a ligação à BD. No final, a *string* `name` fica com o nome do professor.

O código SQL será:

```
"SELECT * FROM professor WHERE codprof = 2"
DbCommand cmd = new DbCommand (conexao, sql);
Resultado res = cmd.Execute (); String name = res [0] ["Nome"];
```

8 SERVIÇOS NA WEB

Neste capítulo, são introduzidos os conceitos associados à utilização de serviços disponibilizados na Web. É apresentado o paradigma principal CRUD (acrónimo de ***c**reate*, *read*, ***u**pdate*, ***d**elete*) e demonstrada a forma como este pode ser aplicado através de Web Services, de *Application Programming Interfaces* (API) e do protocolo de comunicação *Representational State Transfer* (REST). São também ilustrados dois exemplos de utilização de API.

8.1 PARADIGMA CRUD (*CREATE, READ, UPDATE, DELETE*)

A utilização do paradigma CRUD é fundamental para a construção de aplicações Web robustas, pois fornece uma estrutura bem definida e transversal a todos os programadores, que, por isso, conhecem os métodos disponíveis. Num ambiente HTTP, os pedidos CRUD correspondem, respetivamente, aos métodos `post`, `get`, `put` e `delete`. A Figura 8.1 exemplifica um pedido CRUD efetuado pelo cliente ao servidor.

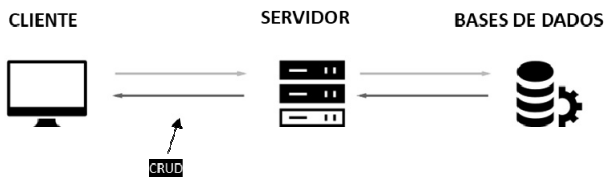

FIGURA 8.1 – Pedido CRUD

O CRUD caracteriza-se por utilizar um método HTTP através de uma rota específica e fazer uma determinada operação na base de dados. Os programadores devem enviar sempre uma mensagem como resposta sobre o o sucesso da operação.

A Tabela 8.1 demonstra o comportamento do CRUD, sendo de referir que os pedidos *read* (apenas um resultado), *update* e *delete* devem conter um ID específico. O CRUD pode ser formalizado através de ***Web Services*** (secção 8.2) ou ***Application Programming Interfaces*** (**API** – secção 8.3).

PEDIDO	ROTA DE PEDIDO	OPERAÇÃO NA BD	RESPOSTA (*BODY*)	RESPOSTA (CÓDIGO DE SUCESSO)
Create	`POST /data`	Inserir dados enviados no pedido (*body*)	`{"success": "Registo Criado"}`	201
Read (todos os dados)	`GET /data`	Nenhuma (apenas lê)	`{"success": [Array contendo todos os registos ]}`	200
Read (dados específicos de um ID)	`GET /data/:id`	Nenhuma (apenas lê)	`{"success": [Array contendo os registos de um determinado ID]}`	200
Update	`PUT /classes/:id`	Alterar dados enviados no pedido (*body*) para determinado ID	`{"success": "Os dados foram atualizados com sucesso"}`	200
Delete	`DELETE classes/:id`	Remover dados de determinado ID	Nenhuma	204

TABELA 8.1 – Comportamento do CRUD

8.2 *WEB SERVICES*

Os *Web Services* existem há alguns anos, mas inicialmente não eram suficientemente rápidos. Permitem converter as soluções existentes em aplicações e publicar as "suas funcionalidades" para "o mundo", sendo componentes de software que comunicam através de protocolos abertos, autocontidos e autodescritíveis, ou seja, são descritos, publicados, descobertos e invocados via Web e através de mensagens *standard* de XML (*Extensible Markup Language*), que constitui a sua base, e do protocolo de comunicação SOAP (*Simple Object Access Protocol*).

Na Figura 8.2, é possível verificar a iteração entre os três principais componentes de um *Web Service*: *broker*, *provider* e *requester*. O serviço de registo *Universal Description, Discovery, and Integration* (UDDI) cria e gere toda a informação do *Web Service*. Por seu lado, o *Web Services Description Language* (WSDL) contém a estrutura base XML do *Web Service* e é o local onde se descreve a sua interface deste. Por último, a comunicação é efetuada através do protocolo SOAP, que fornece uma forma de comunicação entre aplicações executadas em sistemas operacionais diferentes, com tecnologias e linguagens de programação igualmente distintas.

Os *Web Services* foram evoluindo e as tecnologias atualizadas, criando-se uma nova era na qual o foco é a utilização de **API** (secção 8.3) e de **protocolos do tipo *Representational State Transfer*** (**REST**; secção 8.4).

B.5 PASTA CONTROLLERS

A pasta **controllers** (controladores) contém as operações das rotas que se traduzem em operações na BD. O primeiro ficheiro (**auth.controller.js**) está relacionado com as rotas do Passport e os outros controladores estão associados às restantes rotas da aplicação.

A Tabela B.2 faz um breve resumo das operações presentes em cada rota e em cada tabela. Por exemplo, a rota `conferences` tem um conjunto de funções que permitem ler (`get`), inserir (`post`) e apagar (`delete`) nas tabelas **conference**, **conf_sponsor**, **conf_speaker** e **conf_participant**.

ROTA	TABELA(S)	OPERAÇÃO(ÕES)
`/auth`	**user**	`sequelize`

`/conferences`	**conference**
	conf_speaker
	conf_sponsor
	conf_participant
		`get()`
		`post()`
		`delete()`

`/sponsors`	**sponsor**	`get()`
		`post()`
		`put()`
		`delete()`

`/speakers`	**speaker**	`get()`
		`post()`
		`put()`
		`delete()`

TABELA B.2 – RESTful: rotas, tabelas e operações

B.5.1 CONTROLADOR AUTH.CONTROLLER.JS

O primeiro controlador contém as regras do Passport, sendo indiferente a forma como é definido, uma vez que utiliza o Sequelize. Inclui uma chamada da pasta que contém as mensagens JSON (**/assets/jsonMessages/**) e o ficheiro de *login*. Em seguida, e de modo a que os módulos sejam chamados pela `auth.route`, é necessário exportá-los. Cada módulo envia uma mensagem de acordo com a chamada da rota. Por exemplo, se um pedido de registo tiver sido efetuado com sucesso, a rota `/signupSuccess` chama o controlador `signupSuccess` que, por sua vez, acede ao ficheiro de *login* e envia a mensagem que se encontra no objeto `user` e no método `signupSuccess`:

```
const jsonMessagesPath = __dirname + "/../assets/jsonMessages/";
const jsonMessages = require(jsonMessagesPath + "login");
var exports = module.exports = {};
exports.signup = function(req, res) {
     res.status(jsonMessages.user.duplicate.status).send(jsonMessages.use
     r.duplicate);
};
exports.signupSuccess = function(req, res) {
```

```
      res.status(jsonMessages.user.signupSuccess.status).send(jsonMessages
      .user.signupSuccess);
};
exports.signin = function(req, res) {
      res.status(jsonMessages.user.invalid.status).send(jsonMessages.user.
      invalid);
};
exports.signinSuccess = function(req, res) {
      res.status(jsonMessages.user.signinSucces.status).send(jsonMessages.
      user.signinSucces);
};
```

A chamada do *logout* permite terminar e destruir a sessão, assim como enviar a respetiva mensagem:

```
exports.logout = function(req, res, err) {
    req.session.destroy(function(err) {
        if (err) {
            console.log(err);
      res.status(jsonMessages.user.logoutError.status).send(jsonMessages.u
      ser.logoutError);
        }
      res.status(jsonMessages.user.logoutSuccess.status).send(jsonMessages
      .user.logoutSuccess);
    });
};
```

B.5.2 CONTROLADOR CONFERENCES

O controlador **conferences** permite efetuar operações de BD na tabela **conferences**. Em primeiro lugar, é necessário incluir a ligação à BD (`connectMySQL`) que se encontra na pasta **config**. Após a sua inclusão, a ligação passa a estar acessível através da constante `connect`. No início, é também necessário incluir a pasta (`jsonMessages`) que contém as mensagens JSON (`bd`):

```
const connect = require('../config/connectMySQL');
const jsonMessagesPath = __dirname + "/../assets/jsonMessages/";
const jsonMessages = require(jsonMessagesPath + "bd");
```

Em seguida, terão de ser criadas as funções que permitam alterar a BD de acordo com o pedido da rota. A primeira função (`readConference`) permite ler todos os dados da conferência. A função utiliza a ligação (`con`) para efetuar uma *query* à BD. Esta *query* permite devolver o ID da conferência (`idConference`), o acrónimo, o nome, a descrição, o local e a data da conferência. Por forma a fazer *debugging* da *query* e poder validá-la diretamente na BD, é importante fazer o respetivo `console.log` (`query.sql`):

```
function readConference(req, res) {
    const query = connect.con.query('SELECT idConference, acronimo, nome,
      descricao,local, data FROM conference order by data desc',
      function(err, rows, fields) {
        console.log(query.sql);
```