Laboratórios de Informática 2014/2015
DETI - Universidade de Aveiro

Docentes
João Paulo Barraca <jpbarraca@ua.pt>
Diogo Gomes <dgomes@ua.pt>
João Manuel Rodrigues <jmr@ua.pt>
Mário Antunes <mario.antunes@ua.pt>

TEMA 20

Aplicações Móveis Web

Objetivos:
- Conceitos sobre aplicações móveis
- Conceitos sobre HTML para ambientes móveis
- Interacção com Camera e GPS
- Comunicação com serviços externos

20.1 Introdução

Os ambientes móveis, nomeadamente os focados em *smartphones* e *tablets* possuem métodos próprios para o desenvolvimento de aplicações, sobre a forma de Software Development Kits (SDKs). Estes ambientes possibilitam aceder às capacidades do dispositivo, nomeadamente as Application Programming Interfaces (APIs) para acesso à câmara fotográfica, captura de som, localização, ou sensores. As aplicações desenvolvidas podem igualmente trocar informação com servidores, permitindo a implementação de aplicações ricas de mensagens, troca de imagens, ou mesmo jogos.

Uma limitação desta aproximação é a de que as três plataformas principais actualmente em utilização (*Windows Phone*, *Android*, *iOS*) apresentam APIs completamente diferentes. Além disso, todas estas plataformas exigem que as aplicações sejam desenvolvidas em linguagens diferentes. A plataforma *Android* utiliza *Java*, a plataforma *Windows*

Phone utiliza *C#*, e a plataforma *iOS* utiliza *ObjectiveC*. Desenvolver uma aplicação para as três plataformas obriga a que ela seja desenvolvida três vezes, o que apresenta vários problemas para o planeamento e manutenção do código produzido, assim como para o planeamento do modelo de interacção.

Felizmente, o navegador presente nas variadas plataformas possui capacidades bastante avançadas de processamento, permitindo o desenvolvimento de aplicações utilizando plataformas *Web*. A grande vantagem é a de que uma aplicação apenas necessita de ser implementada uma única vez, numa linguagem de fácil acesso como *JavaScript*. Ainda para mais, plataformas como o *Twitter Bootstrap*, consideram prioritária a representação correta das páginas *Web* nas plataformas móveis, possibilitando ainda que uma mesma página *Web* possa ser corretamente visualizada em dispositivos móveis ou computadores portáteis. Um bom exemplo da capacidade de escalabilidade das páginas atuais é a página do serviço *Facebook*, que se adapta de forma bastante correta ao dispositivo cliente.

Como desvantagens destas aplicações considera-se que a velocidade de execução pode ser inferior, o aspeto da aplicação não será igual ao aspeto nativo do dispositivo, e a execução depende da existência de uma ligação de dados. Ainda, pode não ser possível aceder a todos os recursos disponíveis nativamente.

Para o desenvolvimento de aplicações é comum utilizarem-se sistemas como o *PhoneJS* ou *PhoneGap*. No entanto, a utilização destes sistemas requer conhecimentos mais avançados. Deste modo, este guião foca-se na utilização das funcionalidades através de uma variante da biblioteca *Twitter Bootstrap* denominada de *Ratchet* que pode ser obtida em `http://goratchet.com/`.

A documentação desta biblioteca pode ser obtida em `http://goratchet.com/components/`. O que se aconselha que seja realizado antes da execução deste guião.

20.2 Uma aplicação simples

As aplicações não são mais do que páginas *Web* em que a lógica é implementada em *JavaScript* e um conjunto de páginas de estilos criam a ilusão de se tratar de uma aplicação real. Antes de iniciar este ponto, aceda à documentação da biblioteca *Ratchet* e verifique que componentes estão disponíveis.

O desenvolvimento inicia-se obtendo a biblioteca *Ratchet* e colocando os ficheiros necessários nos directórios corretos (directório **`dist`**). Tem também de existir um ficheiro **`index.html`** que representará a aplicação.

O exemplo seguinte resulta numa aplicação apenas com uma barra de título com o texto **LABI**.

```
<html>
  <head>
    <meta charset="utf-8">
    <title>LABI</title>
    <meta name="viewport" content="initial-scale=1, maximum-scale=1, user-scalable=no, minimal-ui">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="apple-mobile-web-app-status-bar-style" content="black">

    <link rel="stylesheet" href="css/ratchet.css">
    <link rel="stylesheet" href="css/ratchet-theme-ios.css">
    <link rel="stylesheet" href="css/app.css">
    <script src="js/ratchet.js"></script>
  </head>
  <body>
    <header class="bar bar-nav">
      <h1 class="title">LABI</h1>
    </header>

    <div class="bar bar-footer">
    </div>

    <div class="content">
        <ul class="table-view">
          <li class="table-view-cell">Hello World</li>
        </ul>
     </div>
  </body>
</html>
```

Exercício 20.1

Crie um directório e coloque nele um ficheiro **index.html** com o conteúdo apresentado anteriormente. Obtenha a biblioteca *Ratchet* e copie o conteúdo do directório **dist** para o local onde se encontra o ficheiro **index.html**.

Para verificar a aplicação num telemóvel, pode enviar o código para o servidor **xcoa.av.it.pt** e depois aceder à página *Web* a partir do telemóvel.

Uma alternativa é activar um servidor HyperText Transfer Protocol (HTTP)[1] a servir o conteúdo do diretório atual. Pode fazer isto executando **python -m SimpleHTTPServer**. No entanto, na rede da UA, não será possível o acesso a partir de outros dispositivos.

Exercício 20.2

Verifique que ao modificar a referência do tema para `ratchet-theme-android.css`, a aplicação irá mudar de aspeto de forma a se tornar semelhante ao sistema android.

Exercício 20.3

Adicione um componente adicional à aplicação desenvolvida.

20.3 Localização

Frequentemente os dispositivos móveis possuem um equipamento de Global Positioning System (GPS) que permite determinar a localização com bastante exactidão. No entanto é possível determinar a localização de outras formas, tal como através das redes sem fios disponíveis. Os diversos meios de determinação de localização foram assim englobados numa mesma API permitindo obter a localização, independente do método utilizado. Um aspeto importante é que o acesso à localização tem de ser explicitamente autorizado pelo utilizador.

O método de acesso assume que se pede acesso à localização, sendo ativada uma função quando o resultado estiver disponível. Esta função é frequentemente denominada de *callback*.

Na aplicação cliente será assim necessário a inclusão de um elemento HyperText Markup Language (HTML)[2] e de código *JavaScript*. Recomenda-se a utilização da biblioteca *jQuery* que tem de ser incluída no directório `js` e importada para a página (no `<head>`).

```
function showPosition(position){
    $('#location').html(position.coords.latitude+'  '+position.coords.longitude);
}

$(document).ready(function() {
    navigator.geolocation.getCurrentPosition(showPosition);
});
```

```
<div class="content">
   <div id="location"></div>
</div>
```

Exercício 20.4

Componha o exercício que permite obter a localização do cliente que acede a uma página. Pode testar localmente ou enviar a página para o servidor `xcoa.av.it.pt`.

Exercício 20.5

Utilizando *LeafletJS* adicione um mapa centrado na posição atual do dispositivo.

20.4 Comunicação com serviços externos

A comunicação com serviços externos é importante pois permite que uma aplicação possa trocar informação com serviços, ou mesmo com outros utilizadores. Aplicações de *chat*, de visualização do estado do tempo, ou de partilha de outra informação (ex, imagens), podem ser rapidamente implementadas. Acima de tudo, este tipo de aplicações é util para demonstrar que as aplicações *Web*, mesmo em ambientes móveis, podem ser dinâmicas e interagir com outros serviços externos.

Um exemplo simples é o de obter informação de um servidor remoto tal como a data e hora. Para isto serão necessários os seguintes componentes:

- Um botão para iniciar o processo (pode ser substituído por um *timer* caso de pretenda actualizar a informação de forma periódica.)
- Código *JavaScript* que obtenha a informação do servidor remoto.
- Uma aplicação que implemente o serviço pedido.
- Um elemento HTML para armazenar o resultado.

Os dois elementos HTML são adicionados no elemento de classe `content`:

```
....
<div class="content">
    <button id="refresh" class="btn btn-block btn-primary">Refresh</button>
    <div id="clock" style="width:100%; text-align:center;"></div>
</div>
```

Sendo que o código *JavaScript* é adicionado num ficheiro que tipicamente se denomina **app.js**. Neste caso, o código espera que seja enviado um elemento JavaScript Object Notation (JSON)[3] com dois atributos: **date** e **time**. O pedido é activado quando o elemento com **id refresh** tiver um evento de **click**.

```
function refresh(){
    $.get("/time",function(response){
        var text="<h2>"+response.date+"</h2><br /><h2>"+response.time+"</h2>";
        $('#clock').html(text);
    });
}

$( document ).ready(function() {
    $('#refresh').on('click',refresh);
});
```

Do lado do serviço, é necessária a implementação de um método que devolva a data e hora, por exemplo:

```
import cherrypy
import time

class HelloWorld(object):
    @cherrypy.expose
    def index(self):
        # Método serve_file tb poderia ser utilizado
        f = open("index.html")
        data = f.read()
        f.close()
        return data

    @cherrypy.expose
    def time(self):
        cherrypy.response.headers['Content-Type'] = 'application/json'
        return time.strftime('{"date":"%d-%m-%Y", "time":"%H:%M:%S"}')

cherrypy.server.socket_port = 8080
cherrypy.server.socket_host = "0.0.0.0"
cherrypy.tree.mount(HelloWorld(),"/","app.config")
```

```
cherrypy.engine.start()
cherrypy.engine.block()
```

De notar que alguns ficheiros são considerados estáticos, enquanto outros são gerados dinamicamente. Os ficheiros estáticos encontram-se nos directórios **`css`** e **`js`**. Desta forma é necessário criar um ficheiro chamado **`app.config`** com o seguinte conteúdo:

```
[/]
tools.staticdir.root ="/home/utilizador/directorio-do-servico"

[/css]
tools.staticdir.on: True
tools.staticdir.dir: 'css'

[/js]
tools.staticdir.on: True
tools.staticdir.dir: 'js'
```

Exercício 20.6

Construa um exemplo que demonstre a comunicação entre uma aplicação Web e um serviço, através de JSON.

Exercício 20.7

O exemplo anterior pode ser modificado de forma a devolver algo mais útil, tal como a distância para o estádio do SL Benfica (38.752667, -9.184711).

Considerando que a função seguinte devolve a distância entre duas coordenadas. Componha um exemplo que envie a localização actual do cliente para o servidor, sendo que este devolve a distância para o estádio.

```
from math import radians, cos, sin, asin, sqrt
...

def distance(lat, lon):
    lat1 = 38.752667
    lon1 = -9.184711
    # graus para radianos
    lon, lat, lon1, lat1 = map(radians, [lon, lat, lon1, lat1])

    # formula de haversine
    dlon = lon - lon1
    dlat = lat - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))

    # 6367 km is the radius of the Earth
    km = 6367 * c

    cherrypy.response.headers['Content-Type'] = 'application/json'
    cherrypy.response.body='{"distance": %d}' % km

...
```

20.5 Acesso a imagens

Através de APIs específicas é igualmente possível aceder às imagens armazenadas num dispositivo móvel, ou mesmo activar a câmara e obter uma imagem. Estas podem depois ser processadas localmente e mesmo enviadas para servidores.

Este método recorre a um elemento **<input>** especificando que se pretende aceder a fotografias.

```
...
<div class="content">
    <input type="file" accept="image/*"></input>
</div>
...
```

O resultado será que o dispositivo irá pedir para que selecione o método de entrada, podendo este ser a câmara ou os documentos já existentes. Se se especificar o atributo **`capture="camera"`**, a camera será activada imediatamente para que possa capturar uma imagem.

Exercício 20.8

Integre o exemplo anterior numa aplicação e verifique o que acontece quando activa o input criado.

Exercício 20.9

Adicione as classes **`btn`**, **`btn-primary`**, **`btn-block`** e **`button-input`**. Considere depois o seguinte excerto de Cascading Style Sheets (CSS)[4].

```
.button{
  text-align: center;
  color: #007aff;
  width: 137px;
}
.button:before{;
  color: white;
  content: 'Usar Imagem';
  padding-right:40px;
  margin-top: -20px;
  padding-left: 10px;
}
```

Componha a aplicação e teste.

Depois de estar selecionada a imagem, seria interessante poder visualizar a mesma. Isto pode ser feito se for activada uma função depois da fotografia ter sido seleccionada e existir um elemento para onde a apresentar.

O elemento, neste caso será um **<canvas>**. Este elemento HTML é semelhante a um **<img>**, com a diferença que é possível desenhar para ele em tempo real. O elemento **<img>** apresenta uma imagem estática. O novo HTML seria:

```
...
<input type="file" accept="image/*" onchange="updatePhoto(event);"></input>
<canvas id="photo" width="530" height="400">
...
```

O código *JavaScript* necessário corresponde à implementação da função **updatePhoto()**, e irá aplicar a imagem capturada ao elemento **<canvas>**:

```
function updatePhoto(event){
    var reader = new FileReader();
    reader.onload = function(event){

        //Criar uma imagem
        var img = new Image();
        img.onload = function(){

            //Colocar a imagem no ecrã
            canvas = document.getElementById("photo");
            ctx = canvas.getContext("2d");
            ctx.drawImage(img,0,0,img.width,img.height,0,0,530, 400);
        }
        img.src = event.target.result;
    }

    //Obter o ficheiro
    reader.readAsDataURL(event.target.files[0]);
    sendFile(event.target.files[0]);

}
```

Exercício 20.10

Componha uma aplicação que replique o exemplo anterior.

Eventualmente a imagem pode ser enviada para um servidor remoto. Será necessário criar código *JavaScript* para construir um pedido de envio (**POST**):

```
function sendFile(file) {
    var data = new FormData();
    data.append('myFile', file);
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "upload");
    xhr.upload.addEventListener("progress", updateProgress, false);

    xhr.send(data);
}

function updateProgress(evt){
    if(evt.loaded == evt.total)
        alert("OK!");
}

function updatePhoto(evt){
    ....

    sendFile(image[0]);

    //Libertar imagem seleccionada
    windowURL.revokeObjectURL(picURL);

}
```

Do lado do servidor será depois necessário receber o ficheiro. Os ficheiros enviados são fornecidos ao serviço, sendo que é necessário copiar a informação para um local mais permanente. Isto porque os dados serão apagados assim que o método **upload** terminar. O código de exemplo será o seguinte:

```
@cherrypy.expose
def upload(self, myFile):
    fo = open(os.getcwd()+ '/uploads/'' + myFile.filename, 'wb')
    while True:
        data = myFile.file.read(8192)
        if not data:
            break
        fo.write(data)

    fo.close()
```

Exercício 20.11

Crie um conjunto de aplicações que permita o envio de imagens para o servidor.

Exercício 20.12

Altere o exemplo de forma a enviar outros ficheiros além de imagens.

Glossário

API	Application Programming Interface
CSS	Cascading Style Sheets
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
SDK	Software Development Kit

Referências

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach e T. Berners-Lee, *Hypertext transfer protocol – http/1.1*, RFC 2616 (Draft Standard), Updated by RFCs 2817, 5785, 6266, Internet Engineering Task Force, jun. de 1999.

[2] W3C. (1999). Html 4.01 specification, endereço: `http://www.w3.org/TR/1999/REC-html401-19991224/`.

[3] E. T. Bray, *The javascript object notation (json) data interchange format*, RFC 7159, Internet Engineering Task Force, mar. de 2014.

[4] W3C. (2001). Cascading style sheets level 2 revision 1 (css 2.1) specification, endereço: `http://www.w3.org/TR/2011/REC-CSS2-20110607/`.