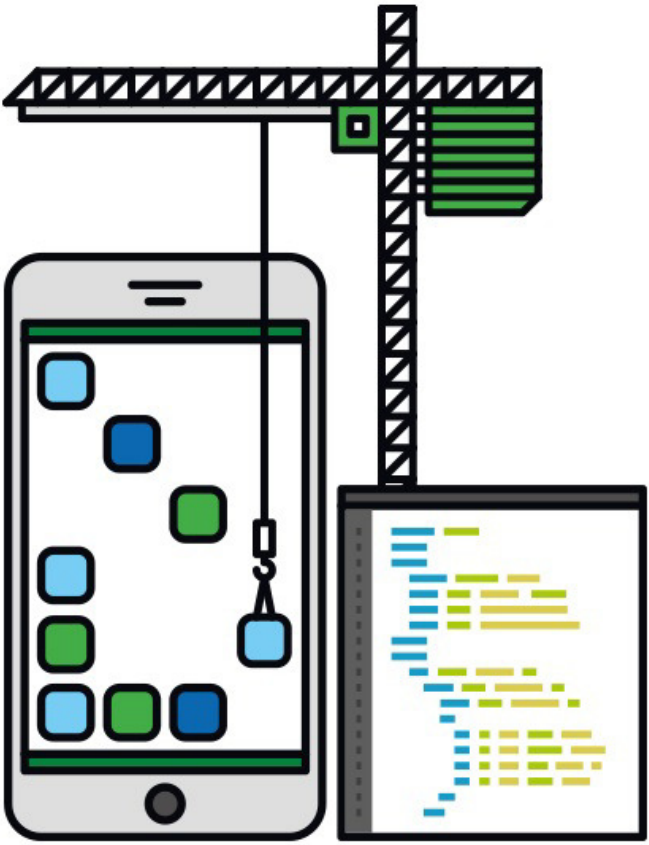
jQuery Mobile

Desenvolva interfaces para múltiplos dispositivos



Casa do Código

DANIEL SCHMITZ

© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

Edição

Adriano Almeida
Vivian Matsui

Revisão

Bianca Hubert
Vivian Matsui

[2017]

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

www.casadocodigo.com.br

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

SOBRE O GRUPO CAELUM

Este livro possui a curadoria da Casa do Código e foi estruturado e criado com todo o carinho para que você possa aprender algo novo e acrescentar conhecimentos ao seu portfólio e à sua carreira.

A Casa do Código faz parte do Grupo Caelum, um grupo focado na educação e ensino de tecnologia, design e negócios.

Se você gosta de aprender, convidamos você a conhecer a Alura (www.alura.com.br), que é o braço de cursos online do Grupo. Acesse o site deles e veja as centenas de cursos disponíveis para você fazer da sua casa também, no seu computador. Muitos instrutores da Alura são também autores aqui da Casa do Código.

O mesmo vale para os cursos da Caelum (www.caelum.com.br), que é o lado de cursos presenciais, onde você pode aprender junto dos instrutores em tempo real e usando toda a infraestrutura fornecida pela empresa. Veja também as opções disponíveis lá.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

PREFÁCIO

Esta obra tem como objetivo abordar o uso da biblioteca jQuery Mobile, que é amplamente utilizada para o desenvolvimento de websites que são visualizados em um dispositivo móvel, geralmente um celular ou *tablet*. Apesar de termos ferramentas para o desenvolvimento nativo, como *Android* e iOs, existem sites e sistemas que devem ser visualizados no browser, e este livro aborda esse processo como criar sistemas para serem exibidos no navegador.

Como biblioteca escolhemos o jQuery Mobile que possui recursos suficientes para a criação de sistemas, além de ser mantido pela mesma equipe que contribuiu com um dos maiores avanços na web nos dias de hoje, que é o jQuery.

O livro possui como principal público-alvo os desenvolvedores que desejam criar websites e aplicações para dispositivos mobile, utilizando as linguagens HTML e JavaScript para o desenvolvimento.

Como pré-requisitos, o leitor deve conhecer um pouco de HTML, CSS e JavaScript. Alguns programas também serão utilizados ao longo desta obra, como o Node e o npm, então é necessário que você tenha um pouco de conhecimento no console do Windows/Linux/Mac.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

SOBRE O AUTOR

Daniel Schmitz trabalha com programação desde 1995, quando utilizava `Basic` para criar programas que faziam cálculos matemáticos. Em 1997, conheceu a internet e os programas `cgi`, juntamente com a linguagem PHP. Desde então, vem acompanhando o lançamento de novas linguagens e tecnologias, tendo como hobby a criação de conteúdo em português para diversos portais na internet, sendo um deles o iMasters.com.br.

A partir de 2007, começou a escrever livros sobre programação, a princípio publicando-os pelas editoras Brasport e Novatec. Em 2009, tornou-se autor independente, publicando livros no portal leanpub.com/u/danielschmitz.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

AGRADECIMENTOS

Meus sinceros agradecimentos a toda a equipe da Casa do Código, pelo auxílio no desenvolvimento deste livro. Agradeço também à minha esposa Jândria, por todo o apoio.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Sumário

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Versão: 20.8.24

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Capítulo 1

INTRODUÇÃO

A cada dia, o desenvolvimento para dispositivos móveis vem ganhando um grande mercado, e uma das provas deste crescimento é a criação de diversas tecnologias para este fim. Assim como tivemos o desenvolvimento web em uma ascensão muito boa nos últimos anos, temos agora o início do desenvolvimento mobile, que compreende *tablets* e celulares.

Como na web, existem duas vertentes que devem ser consideradas no desenvolvimento de um projeto: ou você está criando um site, ou você está criando um sistema. Veja que existe uma diferença enorme entre estes dois caminhos, e é através deles que você começa a escolher as tecnologias que vai usar.

Um detalhe importante no desenvolvimento mobile é que existe uma certa carência de recursos e componentes disponíveis, e talvez por isso a regra de *quanto mais simples melhor* seja aplicada com tanta perfeição. No desenvolvimento mobile, não é recomendado inserir uma tabela com diversas colunas, um formulário muito extenso. Recursos como arrastar e soltar, comuns no desenvolvimento web, são bem restritivos também.

1.1 O QUE PODEMOS FAZER COM MOBILE?

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Seguindo a ideia de que menos é mais, podemos criar interfaces ricas para o mobile com apenas um conjunto limitado de componentes. Como isso é possível? Por meio de um fluxo "natural" de informações, o seu sistema mobile pode se tornar atraente ao usuário, mesmo com poucos recursos. Lembre-se de que a ausência de componentes não compromete a sua aplicação (ou site) se, e somente se, o usuário se sentir confortável em usá-la.

Ou seja, uma tabela com muitas colunas deve ser otimizada para exibir somente o necessário. Um formulário muito extenso pode ser dividido em dois ou três menores. Toda essa simplicidade, apesar de sugerir que a sua aplicação ficará pobre, está na verdade contribuindo para uma aplicação melhor. Quando somos obrigados a simplificar, nós melhoramos um sistema. E isso está trazendo ao Mobile aplicações simples, mas que fluem com o que o usuário precisa.

1.2 E O QUE PODEMOS FAZER COM JQUERY MOBILE?

O assunto específico desta obra é jQuery Mobile, então, se você a adquiriu, você sabe do que se trata. Não vamos falar o que é jQuery Mobile, muitos menos da sua história. Estamos aqui para a prática! Com jQuery Mobile, podemos criar páginas compatíveis com dispositivos mobile, sejam eles *tablets* ou celulares.

O melhor do jQuery Mobile é que ele é HTML puro, você não precisa aprender nada novo ou nada complicado, basta apenas conhecer alguns conceitos fundamentais para começar a criar a sua aplicação. Mesmo sendo puramente HTML, é possível criar tanto páginas quanto aplicações mobile, usando recursos do HTMl 5,

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

que veremos ao longo deste livro.

E o que podemos criar com jQuery Mobile?

- **Páginas HTML:** claro, este é o principal propósito. Cada página contém um título, um conjunto de componentes e um rodapé. A transição entre páginas é realizada automaticamente utilizando Ajax.
- **Barra de botões:** já conhecidas pelos usuários mobile, são indispensáveis para uma fácil navegação no site.
- **Formulários e seus controles:** todos os controles de um formulário, acrescidos de controles específicos do mobile.
- **Listas:** as listas formam a principal forma de visualização de dados de uma aplicação ou página para dispositivos mobile.

1.3 NAVEGADOR WEB

Pode-se utilizar o Google Chrome ou o Mozzila Firefox para acessar as páginas jQuery Mobile. Em http://jquerymobile.com/gbs/, tem-se acesso a quais navegadores são compatíveis com o jQuery Mobile.

1.4 INDO ALÉM DO JQUERY MOBILE

Como todo desenvolvedor *front-end*, o uso de apenas uma tecnologia é algo que não existe em sua rotina de trabalho. Estamos sim trabalhando com jQuery Mobile, mas temos de compreender diversas outras tecnologias e termos que se adéquam ao

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

desenvolvimento como um todo.

Neste livro, nós vamos nos estender um pouco além do jQuery Mobile e apresentar uma tecnologia que ganhou muito destaque no desenvolvimento front-end, que é o Node.js, uma ferramenta muito útil que trouxe a programação JavaScript para o lado do servidor. Através do Node, poderemos realizar diversas tarefas extras que automatizam o nosso aprendizado, e por isso a sua abordagem é obrigatória em qualquer obra front-end disponível no mercado.

1.5 CÓDIGO-FONTE

Todo o código-fonte desta obra está disponível no GitHub: https://github.com/danielschmitz/jquerymobile-codigos.

Lembre-se de que a maioria dos códigos depende de uma atualização do Node pelo comando `npm install`, além da geração da biblioteca pelo comando `gulp`. Caso não conheça estes dois comandos, veja o próximo capítulo, sobre a instalação do jQuery Mobile, para maiores informações.

1.6 DISCUTA SOBRE ESTE LIVRO

Caso tenha alguma dúvida ou sugestão sobre esta obra, escreva para o nosso grupo de discussão, em:

https://groups.google.com/d/forum/livro-jqm

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Capítulo 2

INSTALAÇÃO E PREPARAÇÃO DO AMBIENTE

Neste capítulo, vamos abordar todos os passos necessários para instalar o jQuery Mobile no seu sistema. Perceba que dominar esta instalação não significa apenas referenciar a biblioteca jQuery Mobile no seu documento HTML, mas sim compreender que existem diversas tecnologias envolvidas neste processo – e, como desenvolvedor front-end, você deve conhecê-las.

Com a evolução do JavaScript nos últimos anos, outra tecnologia ganhou destaque no desenvolvimento web, que é o Node.js, o qual vamos chamar simplesmente de **node**. Node é uma plataforma para executar JavaScript no lado do servidor, construída sobre o motor JavaScript do Google Chrome.

2.1 QUE EDITOR DE TEXTOS DEVO USAR?

Aqui deixamos livre a sua escolha por um editor de textos ou uma IDE. Lembre-se de que todo o nosso desenvolvimento é focado em HTML e JavaScript. Ou seja, você não precisará de algo "poderoso" para aprender jQuery Mobile, apenas algo que

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

complemente o código JavaScript/HTML já está ótimo.

Tanto esta obra quanto todo código foram criados usando o Sublime Text 2, http://www.sublimetext.com/, então nos sentimos confortáveis em recomendá-lo.

2.2 QUE SISTEMA OPERACIONAL DEVO USAR?

Você pode utilizar qualquer sistema operacional para aprender a usar o jQuery Mobile. Todas as tecnologias envolvidas têm suporte a Windows, Linux e Mac.

Mas tanto o Linux como o Mac possuem uma otimização melhor quanto a linha de comando e a instalação de programas, enquanto no Windows isso pode parecer um pouco mais complexo. Observe que, nesta obra, vamos abordar a palavra *linha de comando* como sendo o *terminal* do Linux/Mac, ou o *command* do Windows. Para abrir um terminal no Windows, você deve ir em `Iniciar` > `Executar` > `cmd` .

Toda esta obra, bem como seus exemplos e imagens, foi criada usando soluções livres. O sistema operacional utilizado foi o Ubuntu 15.04. Nós recomendamos o uso do Linux no seu desenvolvimento web, mesmo que para isso você tenha de instalar o Ubuntu em uma máquina virtual como o Virtual Box. Experimente novas tecnologias e aprimore o seu aprendizado.

Como padrão para esta obra, toda referência ao diretório `~/` é a referência para o diretório do usuário, como por exemplo, `/home/fulano` no Linux ou `c:\Usuarios\Fulano` no Windows.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

2.3 NODE.JS

Como já foi comentado no capítulo anterior, usaremos extensivamente o `node` para realizar as mais diferenciadas tarefas. Além disso, também temos o `npm` (*Node Package Manager*), que vamos utilizar para instalar bibliotecas do Node no nosso sistema. Siga os procedimentos a seguir para instalar o Node.JS em seu ambiente de desenvolvimento.

Instalando Node no Windows

Se você utiliza Windows, instale o Node pelo link http://www.nodejs.org. Faça o download, instale-o e deixe selecionado a opção `npm` , que é o seu gerenciador de pacotes, conforme a figura a seguir. Além do Node, é útil instalar também o GIT, disponível em https://git-scm.com/.

Instalando Node no Linux

Podemos usar o gerenciador de pacotes `apt-get` do Linux para instalar tudo o que precisamos, bastando apenas executar o seguinte comando:

```
$ sudo apt-get install git npm
```

Após instalar todos os pacotes necessários, é preciso criar um link simbólico para a palavra `node` , conforme o comando:

```
$ sudo ln -s /usr/bin/nodejs /usr/bin/node
```

2.4 SERVIDOR WEB

Mesmo que nosso projeto jQuery Mobile contenha somente

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

arquivos HTML e JavaScript, será preciso um servidor web para que se possa realizar algumas tarefas específicas de servidor. Quando pensamos em servidor web, o Apache é o primeiro da lista que citamos, pois ele é um dos mais conhecidos no mundo. Nós não podemos descartá-lo, principalmente se o seu sistema utiliza o PHP para persistir dados e realizar tarefas comuns, como enviar e-mails, gerenciar sessão etc.

Instalando o Apache no Windows

Se você utiliza Windows, poderá instalar o **Wamp Server**, disponível em http://www.wampserver.com/en/. Faça o download da versão mais recente e instale o Wamp na configuração padrão.

Após instalado, você poderá incluir arquivos na seguinte pasta `C:\wamp\www` , e poderá utilizar o Apache, PHP, MySql e outros utilitários acessando http://localhost/.

Instalando o Apache no Linux

Se usa Linux e uma versão derivada do debian, pode usar o gerenciador de pacotes `apt-get` para instalar tudo para você. O comando a seguir vai instalar o Apache, o PHP e o MySQL.

```
$ sudo apt-get install apache2 apache2-utils
$ sudo apt-get install php5 php5-mysql php-pear php5-mcrypt
$ sudo apt-get install mysql-server
```

Servidor Web Express

Perceba que nosso desenvolvimento front-end está diretamente ligado a duas linguagens: HTML e JavaScript. Veja também que estaremos utilizando constantemente o Node em

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

nossos exemplos e aplicações, sendo que o uso de outras tecnologias (como o PHP) não será abordado.

Em termos didáticos, não faz sentido usarmos o Apache como servidor web, uma vez que podemos expandir nosso conhecimento para aprender tecnologias que estão diretamente ligadas ao desenvolvimento front-end. Uma dessas tecnologias é o framework `express`, que utiliza o Node como base para criar um servidor web totalmente customizável.

Servidor Web http-server

O servidor web `http-server` é semelhante ao `express`, só que mais simples. Ao ativá-lo, a pasta na qual o comando `http-server` é executado torna-se uma pasta virtual, acessada por meio de uma URL e uma porta.

Para instalar o `http-server`, use o `npm`:

```
$ npm install -g http-server
```

Se houver algum problema na instalação, certifique-se de abrir a janela de comando como administrador (Windows), ou use o `sudo` (Linux). Após a instalação, basta executar o comando `http-server` em qualquer pasta que se deseje expor para a web. A resposta após executar este comando é semelhante ao texto a seguir.

```
Starting up http-server, serving ./
Available on:
  http:127.0.0.1:8080
```

2.5 PREPARANDO O PRIMEIRO EXEMPLO

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

COM JQUERY MOBILE

Vamos criar o primeiro exemplo com jQuery Mobile, chamado de `jqm-starter`. Por meio dele, veremos dezenas de detalhes que são pertinentes ao desenvolvimento em si, deixando assim um projeto inicial (que pode ser copiado para outros projetos do livro) como se fosse um projeto base.

Criando o projeto starter

O primeiro passo é criar o diretório `~/jqm-starter`, e iniciar o gerenciador de pacotes, conforme o exemplo a seguir.

```
$ mkdir ~/jqm-starter
$ cd ~/jqm-starter
$  npm init
```

O comando `npm init` apresenta uma das muitas utilidades do Node, que é criar um arquivo que contém informações sobre o projeto. Ao executar o comando `npm init`, diversas informações são requisitadas ao usuário, que podem ser preenchidas conforme a sua necessidade. Após este processo, o arquivo `package.json` é criado, e contém tudo aquilo que você informou.

Este arquivo conterá todas as informações necessárias para que o seu projeto funcione perfeitamente com o Node. Você pode editá-lo quando quiser, adicionando mais informações.

Primeiro, devemos preparar o servidor web, utilizando o `express`. Para instalá-lo usando o Node Package Manager, o `npm`, siga o comando adiante:

```
$ sudo npm install express -g
```

Este comando vai instalar o `express` globalmente no seu

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

sistema, por isso a necessidade do `sudo` para Linux – no Windows, basta abrir a janela de comando no modo administrador. Isso é necessário porque o `express` é uma biblioteca que será utilizada em todos os nossos projetos, então é melhor instalar globalmente primeiro e depois localmente.

Após a instalação global do `express`, vamos instalar localmente no nosso projeto. Para isso, faça:

```
$ npm install express --save
```

Perceba que a instalação foi muito mais rápida, já que o `express` está instalado globalmente. Além disso, usamos o parâmetro `--save` na instalação. Isso diz ao `npm` que atualize o arquivo `package.json` incluindo a referência ao `express`. De fato, ao verificar novamente o arquivo `package.json`, você perceberá a existência da chave `dependencies`, contendo a versão do `express`. Este procedimento de instalação de pacotes do `npm` será uma rotina neste capítulo, então comentaremos apenas algum novo comando além do `npm install`.

Para criar o servidor web, precisamos criar um arquivo que contém alguns comandos do Express. Podemos chamar este arquivo de `server.js`, e ele terá o seguinte código:

```
var express = require("express");
var app = express();

app.use(express.static('public'));

var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;
  console.log('Webserver executando em http://%s:%s', host, port)
;
});
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Inicialmente, usamos o método `require` para utilizar a biblioteca do express. Este comando é exclusivo do Node, e não um comando nativo do JavaScript. Criamos também a variável `app`, que é uma instância do express.

Então, temos o seguinte comando:

```
app.use(express.static('public'));
```

O método `app.use`, em conjunto com o método `express.static`, define um diretório chamado `public` como o diretório que conterá os arquivos estáticos do jQuery Mobile, como arquivos de imagem, JavaScript, CSS e outros. Ainda não criamos este diretório, vamos fazer isso após compreender o arquivo `server.js`.

Continuando, temos mais uma instrução do express, que é a criação do servidor por meio do `app.listen`. Este método informa que o servidor web estará "escutando" o servidor web através da porta `3000`. Além disso, criamos uma mensagem dizendo pelo `console.log` qual o host e porta o express está escutando.

Para que possamos testar o servidor, precisamos agora criar o diretório `public`. Após criá-lo, crie também o arquivo `index.html`, contendo o HTML básico conforme o exemplo a seguir.

```
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
    <meta name="viewport" content="width=device-width, initial-sc
ale=1">
</head>
<body>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
    Hello World
</body>
</html>
```

Com tudo pronto, podemos executar o arquivo `server.js` para iniciar o servidor web. Para isso, abra outro terminal, navegue até o diretório `jqm-starter` e digite:

```
$ node server.js
```

Este terminal trará a resposta `Webserver executando em http://0.0.0.0:3000` e ficará preso. Para cancelar o servidor, basta utilizar `ctrc+c` ou fechar o terminal. Com o servidor executando, a URL `0.0.0.0:3000` aponta para a pasta `public` do nosso projeto (graças ao `express.static`) e, ao abrir este endereço no navegador, temos a seguinte resposta:

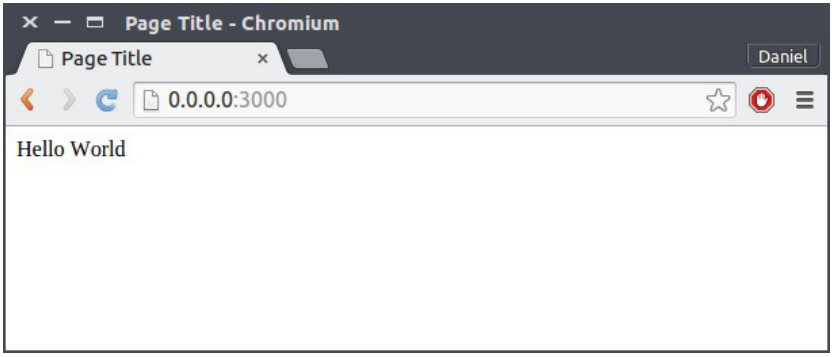


Figura 2.1: Resposta do servidor ao navegador

Com isso, o nosso servidor está pronto. Caso tenha alguma dúvida nas alterações que fizemos nesta parte, acesse este link http://bit.ly/jqm001 para ver no GitHub o que foi alterado e o que foi incluído.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

2.6 INSTALANDO O JQUERY MOBILE

Para instalar o jQuery Mobile, considere duas opções distintas na qual você pode escolher. Cada uma delas tem suas vantagens e desvantagens. A primeira delas, e a mais fácil, é adicionar no seu documento HTML os arquivos JavaScript e CSS necessários para habilitar a biblioteca, como no exemplo a seguir:

```
<!DOCTYPE html>
<html>
<head>
    <title> jqm </title>
    <meta name="viewport" content="width=device-width, initial-sc
ale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.
4.5/jquery.mobile-1.4.5.min.css" />
<script src="http://code.jquery.com/jquery-1.11.1.min.js"></scrip
>
<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.
4.5.min.js"></script>
</head>
<body>
    Hello World jQuery Mobile!
</body>
</html>
```

A segunda forma de instalação do jQuery Mobile é pelo `npm`, simplesmente digitando o seguinte comando:

```
$ npm install jquery-mobile --save
```

O último passo para instalar o jQuery Mobile na pasta `jqm-starter/public` é criar um processo no qual vamos juntar todos os arquivos JavaScript e CSS em um só, e copiá-lo para as pastas `jqm-starter/public/js` e `jqm-starter/public/css`, respectivamente. Esse passo é necessário para que a nossa estrutura fique compatível com futuras atualizações da biblioteca.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Além dos arquivos da biblioteca jQuery e jQuery mobile, criaremos mais dois arquivos que serão usados na aplicação. O primeiro arquivo JavaScript é chamado de `events.js`, e ele é usado para manipular eventos do jQuery Mobile. Este arquivo deve ser inserido após a inclusão da biblioteca jQuery, e antes da inclusão da biblioteca jQuery Mobile. O segundo arquivo da aplicação é chamado de `app.js`, e deve ser inserido após a inclusão da biblioteca jQuery Mobile. Estes arquivos podem ser criados no diretório `jqm-starter`.

Como temos de criar um deploy contendo diversas modificações no projeto, precisamos utilizar um automatizador de tarefas, como o `gulp`.

2.7 UTILIZANDO O GULP PARA AUTOMATIZAR TAREFAS

Como vimos no HTML anterior, temos dois JavaScripts que serão adicionados ao documento HTML. Em um projeto real, pode-se chegar a mais de dez arquivos, contando com código JavaScript e CSS. Cada arquivo deste é uma requisição que o browser faz ao servidor. E em ambiente de produção, temos de nos preocupar com esta "carga". O ideal, neste aspecto, é juntar todos os arquivos JavaScript em um só, fazendo com que somente uma requisição seja feita para carregar todo o JavaScript da página.

Além disso, também temos de retirar todos os espaços em branco do arquivo e reduzir o nome das variáveis e funções para 1 ou duas letras, de forma a diminuir o tamanho dos arquivos envolvidos. Este processo é caracterizado pelo nome *minify*, e você pode ter uma noção exata de como é um arquivo *minificado*

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

acessando este link: http://bit.ly/jqm002.

Então, resta a você, a cada alteração de versão ou a cada inclusão de um novo arquivo `js`, juntá-lo em um único arquivo. Neste ponto, devemos concordar que esta tarefa é muito tediosa, e possivelmente resultará em erros se for realizada manualmente. Neste contexto, entram os automatizadores de tarefas, como o `grunt` ou o `gulp`. Como o `gulp` é mais rápido e fácil de escrever, vamos fazer uma pequena abordagem sobre ele. Lembrando de que o `gulp` nao é o objeto principal de nosso livro, por isso vamos criar algo básico apenas para demonstrar a ferramenta.

O `gulp` é uma biblioteca gerenciada pelo `npm`, então podemos instalá-la da seguinte forma:

```
$ sudo npm install gulp -g
```

Após instalar o `gulp` globalmente, é preciso dizer ao projeto `jqm-starter` que usaremos esta biblioteca. Mas antes disso, vamos analisar um ponto importante na sua instalação. Perceba que o `gulp` é uma ferramenta que vai automatizar alguns processos que nós, desenvolvedores, deveríamos ter feito, como unir vários arquivos JavaScript em um único. Em nenhum momento o `gulp` será usado pelo navegador, pelos nossos sistemas e pelos clientes que usam o nosso sistema. Isso torna o `gulp` uma biblioteca que deve ser usada somente no desenvolvimento do sistema, e não em produção.

Com isso, precisamos dizer ao projeto que ele é usado apenas no desenvolvimento, e isso é feito através do atributo `--save-dev`, que será utilizado substituindo o parâmetro `--save`. Então, para instalar o `gulp` em nosso projeto, faremos:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
$ npm install gulp --save-dev
```

Após executar este comando, ele será instalado no diretório `node_modules`, e o arquivo `package.json` será atualizado adicionando uma referência a esta biblioteca.

Além do `gulp`, também precisamos do `gulp-concat`, pois será ele que vai concatenar os arquivos JavaScript e CSS. Para instalá-lo, faça:

```
$ sudo npm install gulp-concat -g
$ npm install gulp-concat --save-dev
```

Com todas as bibliotecas prontas, vamos iniciar a rotina para juntar todos os arquivos. Isso é feito por meio de um arquivo chamado `gulpfile.js`, que deve estar na raiz do projeto `jqm-starter`.

A princípio, escreva o seguinte código no arquivo `jqm-starter/gulpfile.js`:

```
var gulp = require('gulp');
var concat = require('gulp-concat');

//Cria a tarefa default
gulp.task('default',function(){

});
```

Neste arquivo, criamos duas variáveis, `gulp` e `concat`, e então usamos o método `task` da variável `gulp` para criar uma tarefa, que inicialmente não faz nada. Perceba que a criação da variável `gulp` usa o método `require`, que não é um método do JavaScript, mas sim do Node.

Salve o arquivo e abra o terminal, e então digite o seguinte comando:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
$ gulp
[22:56:25] Using gulpfile ~/jqm-starter/gulpfile.js
[22:56:25] Starting 'default'...
[22:56:25] Finished 'default' after 57 µs
```

O comando `gulp` automaticamente executará a tarefa `default` criada. Agora, vamos concatenar todos os JavaScripts e CSSs para gerar um novo arquivo, que será chamado de `script.min.js` e `style.css`. Altere o arquivo `gulpfile.js` para o seguinte código.

```
var gulp = require('gulp');
var concat = require('gulp-concat');

var js  = [
    './node_modules/jquery-mobile/node_modules/jquery/dist/jquery
.js',
    './events.js',
    './node_modules/jquery-mobile/dist/jquery.mobile.js',
    './app.js',
];

var css = [
    './node_modules/jquery-mobile/dist/jquery.mobile.min.css'
]

var fodlers = [
        {from:'./node_modules/jquery-mobile/dist/images',to:'./pu
blic/css/'}
]

gulp.task('default',function(){

    gulp.src(js)
        .pipe(concat('script.js'))
        .pipe(gulp.dest('./public/js/'));

    gulp.src(css)
        .pipe(concat('style.css'))
        .pipe(gulp.dest('./public/css/'));

    gulp.src(['./node_modules/jquery-mobile/dist/images/**/*.*'])
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
        .pipe(gulp.dest('./public/css/images'));
});
```

Agora inserimos alguns comandos na tarefa `default`, sendo o primeiro deles o `gulp.src`, que indica um array com a fonte de arquivos que será trabalhada. Após o `src`, usamos o método `pipe` para conectar outro comando, o `concat`, que vai juntar tudo aquilo que o `src` obteve. Depois, usamos novamente o `pipe` em conjunto com o comando `gulp.dest` que salvará tudo que foi processado na pasta `public/js`. O mesmo acontece com o arquivo CSS.

Além de juntar e copiar os arquivos JavaScript e CSS, também copiamos a pasta `images` para o mesmo diretório do CSS, pois o jQuery Mobile utiliza algumas imagens em sua estrutura.

Após executar novamente o comando `gulp`, teremos então a criação do arquivo `public/js/script.js` e `public/css/style.css`.

2.8 CRIANDO A VERSÃO FINAL DO PROJETO STARTER

Agora que as bibliotecas estão prontas, chegamos à versão definitiva do HTML inicial a ser criado, veja:

```
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
    <meta name="viewport" content="width=device-width, initial-sc
ale=1">
    <link rel="stylesheet" type="text/css" href="css/style.css">
</head>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
<body>
    Hello World
<script type="text/javascript" src="js/script.js"></script>
</body>
</html>
```

Para testar o arquivo `index.html`, execute o servidor pelo comando `node server.js` e acesse `0.0.0.0:3000`. A página será carregada, mas ainda sem uma formatação padronizada para um dispositivo mobile.

Vamos então ajustar esta formatação e finalizar o projeto `jqm-starter`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
    <meta name="viewport" content="width=device-width, initial-sc
ale=1">
    <link rel="stylesheet" type="text/css" href="css/style.css">
</head>
<body>

<div data-role="page">
    <div data-role="header">
        <h1> Título </h1>
    </div>

    <div role="main" class="ui-content">
        <p> Hello Word !!</p>
    </div>

    <div data-role="footer">
        <h4>Rodapé</h4>
    </div>
</div>

<script type="text/javascript" src="js/script.js"></script>
</body>
</html>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

A resposta para o código HTML anterior é semelhante à figura a seguir:

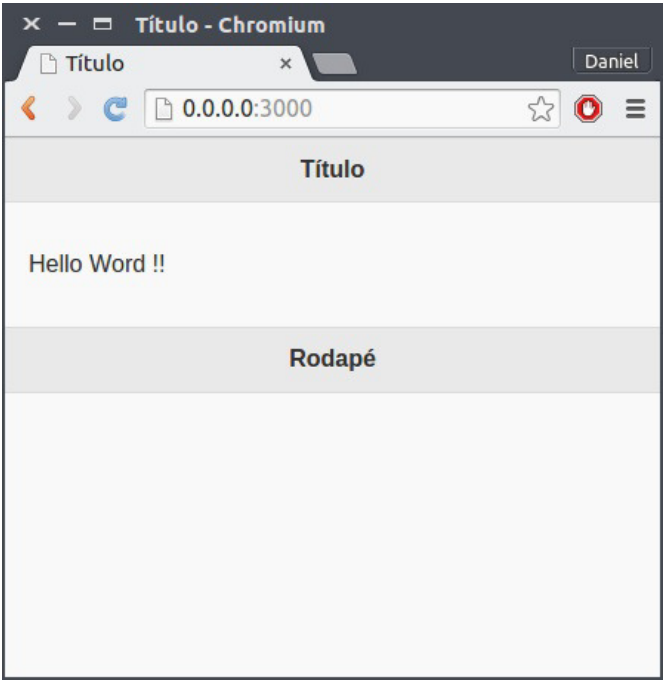


Figura 2.2: Projeto starter pronto para uso, com uma página padrão do jQuery Mobile

Este processo, apesar de um pouco trabalhoso, é recomendado para que possamos otimizar o código JavaScript no dispositivo mobile. Ainda existem mais dois passos importantes para melhorar a tarefa de automatização, que serão descritas a seguir.

2.9 UTILIZANDO O GULP PARA DIMINUIR O TAMANHO DO JAVASCRIPT

Este processo, chamado de *minify*, corresponde em retirar

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

todos os espaços do arquivo, deixando todo o código em uma única linha. Isso é bom porque o arquivo será menor.

Primeiro, precisamos instalar o pacote `gulp-minify`, da seguinte forma:

```
$ sudo npm install gulp-minify -g
$ npm install gulp-minify --save-dev
```

Após instalar a biblioteca, altere o arquivo `gulp.js` incluindo a biblioteca `gulp-minify` e adicionando o método `minify` na concatenação dos arquivos JavaScript, da seguinte forma:

```
var gulp = require('gulp');
var concat = require('gulp-concat');
var minify = require('gulp-minify');

var js  = [
    './node_modules/jquery-mobile/node_modules/jquery/dist/jquery
.js',
    './events.js',
    './node_modules/jquery-mobile/dist/jquery.mobile.js',
    './app.js',
];

var css = [
    './node_modules/jquery-mobile/dist/jquery.mobile.min.css'
]

var fodlers = [
        {from:'./node_modules/jquery-mobile/dist/images',to:'./pu
blic/css/'}
]

gulp.task('default',function(){

    gulp.src(js)
        .pipe(concat('script.js'))
        .pipe(minify())
        .pipe(gulp.dest('./public/js/'));

    gulp.src(css)
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
        .pipe(concat('style.css'))
        .pipe(gulp.dest('./public/css/'));

    gulp.src(['./node_modules/jquery-mobile/dist/images/**/*.*'])
        .pipe(gulp.dest('./public/css/images'));

});
```

Consulte agora o diretório `jqm-starter/public/js` e verifique se existem dois arquivos, `script.js` e `script-min.js` . Compare o tamanho em KB de cada um deles e seu conteúdo. Em ambiente de desenvolvimento, usa-se o arquivo `script.js` , pois será possível debugar o código JavaScript e ver mensagens de erro mais claras. Em ambiente de produção, quando o sistema estiver pronto e funcionando no servidor, usa-se o arquivo `script.min.js` .

2.10 TORNANDO O GULP AUTOMÁTICO

Quando escrevemos a tarefa de juntar todos os arquivos JavaScript em um único arquivo, criamos um pequeno problema que pode ser facilmente resolvido. No ambiente de desenvolvimento, você vai alterar os arquivos `events.js` e `app.js` várias vezes ao dia. Um inconveniente nisso é que, toda vez que você alterar o arquivo `events.js` , terá de executar o comando `gulp` para que esta alteração reflita no diretório `public/js` .

Quando isso acontece, usamos novamente o `gulp` para resolver esse problema, por meio do código a seguir, que deverá ser inserido no final do arquivo `gulpfile.js` :

```
// início do arquivo gulpfile.js

gulp.watch(js,['default']);
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Esta única linha utiliza o método `watch` para que, em vez de terminar a execução do gulp, ele fique escutando alterações nos arquivos que estão no array `js` e, caso estas alterações ocorram o método `default`, seja executado novamente.

Com todas estas tarefas implementadas, temos o projeto `jqm-starter` funcional para que possa ser usado como modelo para demais projetos em jQuery Mobile.

2.11 VENDO OS EXEMPLOS DESTE LIVRO ONLINE

Podemos também utilizar o recurso de construir páginas em jQuery Mobile no próprio navegador, através de um site chamado **jsfiddle**. Ao acessar este site, pelo endereço www.jsfiddle.net, você verá uma página semelhante à exibida a seguir, na qual pode-se escolher uma versão do jQuery a ser carregada, juntamente com a versão do jQuery Mobile. Neste livro, estaremos utilizando o jQuery 2.1.0 e o jQuery Mobile 1.4.2.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

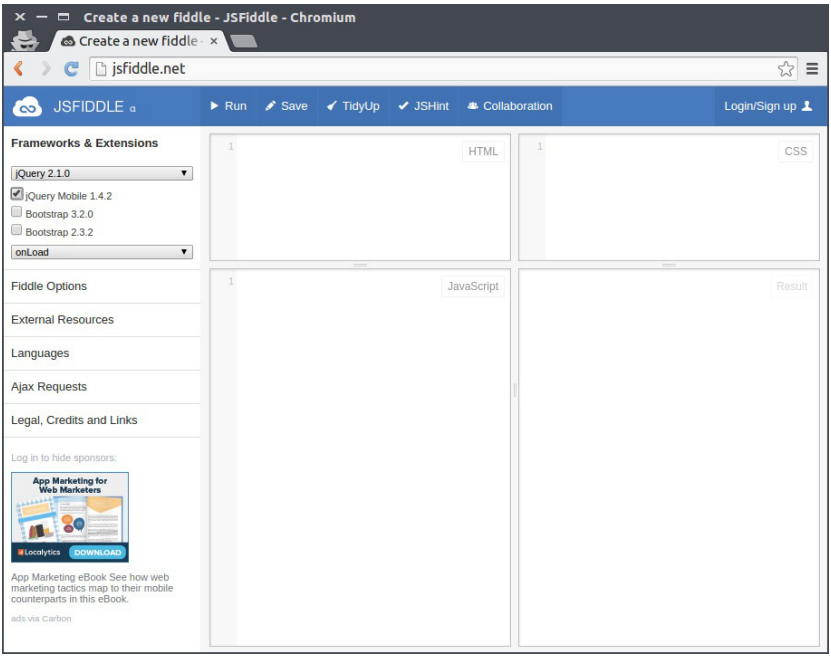


Figura 2.3: Página jsfiddle pronta para o jQuery Mobile

Sempre que possível, haverá um link após o código com o texto `Ver online`. Clique nele para ver o exemplo em ação.

2.12 CONCLUSÃO

Neste capítulo, nós aprendemos a instalar o jQuery Mobile em nosso sistema por meio do Node e do NPM. Também utilizamos técnicas avançadas para juntar todos os arquivos JavaScript em um, provendo assim um projeto no qual usaremos como base para todos os próximos projetos desta obra.

No próximo capítulo, vamos aprender todos os detalhes do jQuery Mobile, aplicados ao nosso projeto.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Capítulo 3

CONHECENDO O JQUERY MOBILE

Agora que criamos uma estrutura básica para os projetos com jQuery mobile, podemos começar a compreender como é o seu funcionamento. Para testá-los, recomendo copiar e colar o projeto `jqm-starter` para `jqm-basico` (ou um outro nome que você desejar), e escrever os códigos que serão apresentados a seguir, testando-os no navegador ou no dispositivo mobile.

O primeiro conceito importante do jQuery Mobile é a página, ou *page*. Uma página é definida pelo elemento que contém o atributo `data-role='page'` . Geralmente, este elemento é uma `<div>` , e ela possui como elementos filhos um cabeçalho, conteúdo e rodapé. Este molde é definido pelo código a seguir:

```
<div data-role="page">
    <div data-role="header">
        <h1> Título </h1>
    </div>
    <div role="main" class="ui-content">
        <p> Conteúdo </p>
    </div>
    <div data-role="footer">
        <h4>Rodapé</h4>
    </div>
</div>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Ver online: http://bit.ly/jqm004.

No exemplo a seguir, ilustraremos um exemplo simples de lista:

```
<div data-role="page">
    <div data-role="header">
        <h1>Lista Simples</h1>
    </div>
    <div data-role="content">
        <ul data-role="listview">
            <li>Item 1</li>
            <li>Item 2</li>
            <li>Item 3</li>
            <li>Item 4</li>
            <li>Item 5</li>
        </ul>
    </div>
</div>
```

Ver online: http://bit.ly/jqm005

Esse exemplo produzirá o seguinte resultado:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

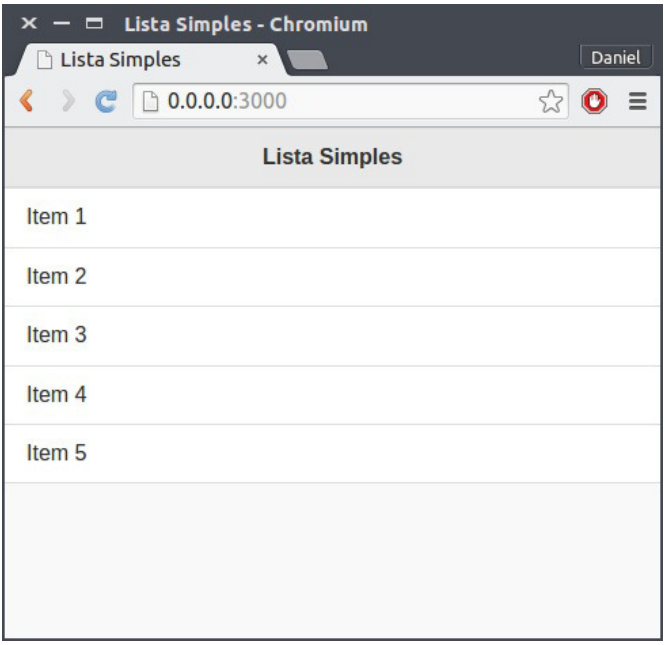


Figura 3.1: Exemplo de lista com jQuery Mobile

A lista é definida através do atributo `data-role='listview'`. Perceba que, quando estamos trabalhando com jQuery Mobile, estamos criando HTML. O principal conceito do desenvolvimento de aplicações neste framework é que ele é construído com HTML, contendo elementos `data-` que definem os seus componentes.

Saber jQuery Mobile é saber quais tags HTML e quais `data-roles` usar, o que pode ser facilmente consultado na documentação. Veja que não é nenhuma tarefa complexa conhecer estes tipos, já que conhecemos amplamente o HTML e sua estrutura. Sabemos que devemos criar tags encadeadas, sabemos como criar atributos para as tags, entre outras particularidades, que já são comuns a nós, desenvolvedores web.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

3.1 PÁGINAS

Já vimos que uma página é formada pelo elemento `data-role='page'` . É possível criar quantas páginas forem necessárias em um único arquivo HTML. O exemplo a seguir ilustra este processo:

```
<div data-role="page" id=”pag1”>
    <div data-role="header">Título 1</div>
    <div data-role="content">Conteúdo Página 1</div>
    <div data-role="footer">Rodapé</div>
</div>
<div data-role="page" id=”pag2”>
    <div data-role="header">Título 2</div>
    <div data-role="content">Conteúdo Página 2</div>
    <div data-role="footer">Rodapé</div>
</div>
<div data-role="page" id=”pag3”>
    <div data-role="header">Título 3</div>
    <div data-role="content">Conteúdo Página 3</div>
    <div data-role="footer">Rodapé</div>
</div>
```

Estas três páginas não serão exibidas de uma vez só. Inicialmente, é exibida a primeira página, enquanto as outras duas páginas estão escondidas, e podem ser chamadas através de um simples link, como no exemplo a seguir:

```
<div data-role="page" id="pag1">
    <div data-role="header">
        <h1>Página 1</h1>
    </div>
    <div data-role="content">
        <ul data-role="listview">
            <li><a href="#pag2">Página 2</a>
            </li>
            <li><a href="#pag3">Página 3</a>
            </li>
            <li><a href="padrao.html">Página Padrão</a>
            </li>
        </ul>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
    </div>
</div>
<div data-role="page" id="pag2">
    <div data-role="header">
        <a href='#pag1' data-role='button'
        data-direction='reverse'>Voltar</a>
        <h1>Página 2</h1>
    </div>
    <div data-role="content">
        <p>Esta é a página 2</p>
    </div>
</div>
<div data-role="page" id="pag3">
    <div data-role="header">
        <h1>Página 3</h1>
    </div>
    <div data-role="content">
        <p>Esta é a página 3</p>
    </div>
</div>
```

Ver online: http://bit.ly/jqm008

Neste código HTML, temos um novo atributo, o `data-direction='reverse'` , que executa a animação de voltar a página, criando uma transição no formato reverso.

3.2 TRANSIÇÕES

Por padrão, o jQuery Mobile aplica o efeito de slide na transição entre as páginas. Mas existem outros efeitos que podem ser aplicados, e são determinados pelo atributo `data-transition` . Os efeitos são:

- `pop`
- `slideup`
- `fade`
- `slidedown`

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

- `flip`
- `turn`
- `flow`
- `slidefade`
- `none`

Para adicionar o efeito, basta incluir o atributo no link da página:

```
<a href="#pag2" data-transition="slide">Página 2</a>
```

3.3 CAIXAS DE DIÁLOGO (POP-UPS)

É muito comum em aplicações para web/desktop criar caixas pop-up para exibir informações ou perguntar algo ao usuário. No jQuery Mobile, usamos o atributo `data-rel="dialog"` para adicionar este efeito, como no exemplo a seguir:

```
<div data-role="page" id="pag1">
    <div data-role="header">
        <h1>Um objeto qualquer</h1>
    </div>
    <div data-role="content">
        <p>Página de um objeto qualquer</p>
        <a href="#confirmeDelete" data-role="button" data-rel="di
alog">Apagar</a>
    </div>
</div>

<div data-role="page" id="confirmeDelete">
    <div data-role="header">
        <h1>Confirmação</h1>
    </div>

    <div data-role="content">
        <p>Tem certeza?</p>
        <a href="#deletar" data-role="button">Sim</a>
        <a href="#" data-role="button" data-rel="back">Não</a>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
    </div>
</div>

<div data-role="page" id="deletar">
    <div data-role="header">
        <h1>Foi...</h1>
    </div>

    <div data-role="content">
        <p>Apagado com sucesso!</p>
    </div>
</div>
```

Ver online: http://bit.ly/jqm009

Execute este exemplo, e navegue pelo botão apagar e pelos botões `Sim` e `Não`. Veja que, quando criamos o botão `apagar`, adicionarmos o `data-role="button"`, o que torna o link um botão. O `data-rel="dialog"` torna a página de destino (`href`) um pop-up, que assumirá o efeito de pop.

A página `confirmDelete` possui dois botões, sendo que o botão `Sim` apenas direciona para outra página (lembre-se, página jQuery Mobile, e não outro arquivo HTML). O botão `Não` possui `data-rel="back"`, que simula o efeito de "back" do dispositivo, voltando a quem o chamou.

3.4 TOOLBARS

As toolbars podem estar no cabeçalho ou no rodapé da página. A posição depende apenas se o botão está no `data-role="header"` ou no `data-role="footer"`.

Um botão da toolbar é definido por meio de um link a `href=""`, onde é possível usar um atributo chamado `data-icon`, que define um ícone padrão. O exemplo a seguir pode ser usado

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

para um formulário de dados:

```
<div data-role="page" id="pag1">
    <div data-role="header">
        <a href="#">Cancelar</a>
        <h1>Um objeto qualquer</h1>
        <a href="save.html" data-icon="check">Salvar</a>
    </div>
    <div data-role="content">
        <p>Página com um formulário qualquer.</p>
    </div>
</div>
```

Ver online: http://bit.ly/jqm010

Para fixar o cabeçalho ou rodapé na pagina, use o atributo `data-position="fixed"`. Desta forma, a rolagem da página mantém o cabeçalho dela.

3.5 ÍCONES

Todos os botões do jQuery Mobile (que são links `a href=""`) podem ter ícones atribuídos através da propriedade `data-icon`. Os ícones disponíveis estão neste link: http://bit.ly/jqm011.

Os ícones são atribuídos pelo atributo `data-icon`, como por exemplo, `data-icon="arrow-r"`. Pode-se alterar a posição do ícone em relação ao texto do botão por meio do atributo `data-iconpos="posição"`. A posição pode ser `top`, `bottom`, `left` e `right` e, caso deseje retirar o label do botão, pode-se utilizar `notext`.

3.6 NAVBARS

Uma NavBar é composta por botões que preenchem 100% a

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

tela, de forma a simular abas para uma aplicação mobile. Para criar uma Navbar, basta usar o atributo `data-role="navbar"` seguido de itens `ul` e `li`, conforme o exemplo a seguir:

```
<div data-role="page" id="pag1">
    <div data-role="header">
        <h1>Uma página qualquer</h1>
        <div data-role="navbar">
            <ul>
                <li><a href="#" class="ui-btn-active">Item 1</a>
                </li>
                <li><a href="#">Item 2</a>
                </li>
                <li><a href="#">Item 3</a>
                </li>
                <li><a href="#">Item 4</a>
                </li>
            </ul>
        </div>
    </div>
    <div data-role="content">
        <p>Página com um conteúdo qualquer.</p>
    </div>
</div>
```

Ver online: http://bit.ly/jqm015

3.7 BOTÕES

A maioria dos botões no jQuery Mobile conduz a outras páginas. Lembre-se de que uma página jQuery Mobile não é necessariamente um arquivo HTML, mas uma seção `data-role="Page"`.

Mas nem todos os botões são formados pela tag `<input type="Button">`. A maioria deles são links com o atributo `data-role="Button"`, conforme o exemplo:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
<a href="index.html" data-role="button">Link button</a>
```

Da mesma forma que adicionamos ícones nas toolbars e nas navbars, podemos usar o atributo `data-icon` para adicionar ícones aos botões. No exemplo anterior, repare que o botão ficou com a largura total da página e, caso adicione dois botões, um ficará abaixo do outro. Para evitar este comportamento, usamos o atributo `data-inline="true"` fazendo com que o botão fique no seu tamanho natural, por exemplo:

```
<a href="index.html" data-role="button" data-inline="true">Cancel
</a>
<a href="index.html" data-role="button" data-inline="true" data-t
heme="b">Save</a>
```

Ver online: http://bit.ly/jqm016

Pode-se agrupar botões facilmente através da criação de uma `<div>` com o atributo `data-role="controlgroup"`, conforme vemos a seguir:

```
<div data-role="controlgroup">
     <a href="index.html" data-role="button">Yes</a>
     <a href="index.html" data-role="button">No</a>
     <a href="index.html" data-role="button">Maybe</a>
</div>
```

Ver online: http://bit.ly/jqm017

3.8 FORM

Formulários são usados para a entrada de dados, sendo que, como estamos usando HTML para criar páginas, nada mais normal do que usar os mesmos controles do HTML para os formulários jQuery Mobile. Além de o jQuery Mobile estilizar os controles e botões para os dispositivos mobile, ele também

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

adiciona novos controles, como o `slider` e o `flip`, que são muito comuns neste tipo de aplicação.

Assim como no HTML, começamos um formulário jQuery Mobile com a tag `form`, incluindo os atributos `action` e `method`. Depois, podemos adicionar diversos componentes, enumerados a seguir.

TextInput

É o principal componente, usado para inclusão de texto.

```
<input type="text" name="name" id="name" value="" placeholder="Se
 nome" />
```

TextArea

Idêntico ao HTML comum.

```
<textarea cols="40" rows="8" name="textarea" id="textarea"></text
area>
```

Flip

Usado para alterar um valor booleano. É utilizado no mobile por ser mais fácil de ser alterado em relação a um *checkBox* comum. Ele é caracterizado pelo atributo `data-role="slider"`.

```
<div data-role="fieldcontain">
    <select name="slider2" id="slider2" data-role="slider">
        <option value="off">Off</option>
        <option value="on">On</option>
    </select>
</div>
```

Ver online: http://bit.ly/jqm019

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Slider

O Slider é usado para selecionar um valor numérico dado um valor mínimo e máximo.

```
    <input type="range" name="slider" id="slider" value="0" min="
0" max="100"  />
```

Ele é caracterizado pelo `type="range"` , e você precisa fornecer os atributos `min` e `max` .

Control Group com check/radio box

O `controlgroup` é um componente para agrupar os campos `CheckBox` e `RadioBox` . Pode-se agrupá-los na vertical ou na horizontal, conforme os exemplos a seguir:

```
    <div data-role="fieldcontain">
         <fieldset data-role="controlgroup">
         <legend>Choose as many snacks as you'd like:</legend>
         <input type="checkbox" name="checkbox-1a" id="checkbox-1
a" class="custom" />
         <label for="checkbox-1a">Cheetos</label>
         <input type="checkbox" name="checkbox-2a" id="checkbox-2
a" class="custom" />
         <label for="checkbox-2a">Doritos</label>
         <input type="checkbox" name="checkbox-3a" id="checkbox-3
a" class="custom" />
         <label for="checkbox-3a">Fritos</label>
         <input type="checkbox" name="checkbox-4a" id="checkbox-4
a" class="custom" />
         <label for="checkbox-4a">Sun Chips</label>
        </fieldset>
    </div>
```

Ver online: http://bit.ly/jqm020

Control Group com checkbox horizontal

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Pode-se agrupar as opções do `controlgroup` na forma horizontal, pelo atributo `data-type=horizontal` .

```
<div data-role="fieldcontain">
   <fieldset data-role="controlgroup" data-type="horizontal">
      <legend>Font styling:</legend>
      <input type="checkbox" name="checkbox-6" id="checkbox-6"  c
lass="custom" />
      <label for="checkbox-6">b</label>
      <input type="checkbox" name="checkbox-7" id="checkbox-7" cl
ass="custom" />
      <label for="checkbox-7"><em>i</em></label>
      <input type="checkbox" name="checkbox-8" id="checkbox-8" cl
ass="custom" />
      <label for="checkbox-8">u</label>
  </fieldset>
</div>
```

Ver online: http://bit.ly/jqm021

Select

Um select é usado para escolher um valor de uma lista, mas a lista aparece em um pop-up quando clicamos no controle. Ele é criado da mesma forma que o `select` do HTML:

```
<select name="select-choice-0" id="select-choice-1">
   <option value="standard">Standard: 7 day</option>
   <option value="rush">Rush: 3 days</option>
   <option value="express">Express: next day</option>
   <option value="overnight">Overnight</option>
</select>
```

3.9 LISTAS

Um dos componentes mais usados nas aplicações mobile é a lista. Seja para criar um menu para exibir configurações ou o resultado de alguma informação, a lista é uma das melhores

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

alternativas para a exibição em telas com o tamanho reduzido. Ela substitui os conhecidos *DataGrids* da versão desktop.

Como você já sabe, o jQuery Mobile baseia-se no modelo HTML, utilizando o atributo `data-role` para diferenciar a visualização de informações na tela. No caso da lista, usamos `data-role="listview"`, em uma lista `ul li`.

```
<div data-role="page">
    <div data-role="header">
        <h1>Lista Simples</h1>
    </div>
    <div data-role="content">
        <ul data-role="listview">
            <li>Item 1</li>
            <li>Item 2</li>
            <li>Item 3</li>
            <li>Item 4</li>
            <li>Item 5</li>
        </ul>
    </div>
</div>
```

Ver online: http://bit.ly/jqm022

Listas com filtro

É muito simples adicionar um filtro na sua lista, bastando apenas adicionar o atributo `data-filter="true"` na definição do `data-role="listview"`.

Listas com margem

Para adicionar uma margem em uma lista, basta usar o atributo `data-inset="true"`.

Lista com tópicos

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Para criar o tópico, usamos o atributo `data-role="list-divider"` no item `li` da lista.

```
<div data-role="content">
    <ul data-role="listview">
        <li data-role="list-divider">Sudeste</li>
        <li>Minas Gerais</li>
        <li>São Paulo</li>
        <li>Rio de Janeiro</li>
        <li>Espírito Santo</li>
        <li data-role="list-divider">Sul</li>
        <li>Paraná</li>
        <li>Santa Catarina</li>
        <li>Rio Grande do Sul</li>
        <li data-role="list-divider">Centro-Oeste</li>
        <li>Mato Grosso</li>
        <li>Goiás</li>
        <li>Mato Grosso do Sul</li>
        <li data-role="list-divider">Nordeste</li>
        <li>Bahia</li>
        <li>Sergipe</li>
        <li>....</li>
        <li>Maranhão</li>
        <li data-role="list-divider">Norte</li>
        <li>Amazonas</li>
        <li>Acre</li>
        <li>Pará</li>
        <li>Amapá</li>
        <li>Roraima</li>
        <li>Tocantins</li>
    </ul>
</div>
```

Ver online: http://bit.ly/jqm023

Listas com textos extras

Uma lista de itens não está limitada a conter somente itens. É possível adicionar diversos itens a mais nela, sendo que cada item é corretamente renderizado no dispositivo.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Incluindo texto no lado direito do item

Ao usar a seguinte tag:

```
<span class="ui-li-count">...</span>
```

Pode-se adicionar um texto na parte direita do item da lista. Por exemplo:

```
<ul data-role="listview" data-inset="true">
    <li data-role="list-divider">Sudeste
        <span class="ui-li-count">4</span>
    </li>
    <li>Minas Gerais</li>
    <li>São Paulo</li>
    <li>Rio de Janeiro</li>
    <li>Espírito Santo</li>
</ul>
```

Ver online: http://bit.ly/jqm024

Texto dentro do item

Use a tag `h3` para criar um título no item da lista, e a tag `p` para adicionar algum texto. Por exemplo:

```
<ul data-role="listview" data-filter="true">
   <li data-role="list-divider">Sudeste
       <span class="ui-li-count">4</span>
   </li>
   <li>
    <h3>Minas Gerais</h3>
    <p>População: X.XXX.XXX</p>
</li>
<li>
    <h3>São Paulo</h3>
    <p>População: xx.xxx.xxx</p></li>
    <li>Rio de Janeiro</li>
    <li>Espírito Santo</li>
</ul>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Ver online: http://bit.ly/jqm025

Incluindo texto ao lado direito de cada item

Pode-se também adicionar texto ao lado direito do item, pelo atributo `class=ui-li-aside` . Veja o exemplo:

```
<li>
    <h3>Minas Gerais</h3>
    <p class="ui-li-aside">Capital: Belo Horizonte</p>
    <p>População: X.XXX.XXX</p>
</li>
```

3.10 TEMAS

O jQuery Mobile possui na versão 1.4 dois temas prontos que podem ser usados por meio do atributo `data-theme="a"` ou `data-theme="b"` . Também é possível criar o próprio tema através da URL: http://jqueryui.com/themeroller/

3.11 CONCLUSÃO

Neste capítulo, abordamos os componentes principais do jQuery Mobile. É por eles que criaremos páginas para a interação com o usuário. Além de componentes, também precisamos adicionar um pouco de lógica de programação nestas páginas, e faremos isso por meio do JavaScript, abordado no próximo capítulo.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

CAPÍTULO 4

UTILIZANDO JAVASCRIPT NO JQUERY MOBILE

A utilização de JavaScript em uma página jQuery Mobile segue o mesmo conceito de uma página HTML comum. Pode-se adicionar JavaScript na página por meio do elemento `<script>`, sendo que, para o jQuery Mobile, ainda existem métodos e eventos adicionais, que serão abordados a seguir.

4.1 EVENTOS

O jQuery Mobile possui eventos que podem ser usados em todo o ciclo de formação da página HTML. Lembre-se de que, no projeto `jqm-starter`, os eventos devem ser associados ao arquivo `events.js`, pois ele é inserido após a inclusão da biblioteca `jQuery` e antes da `jquery.mobile`. Esta é uma característica do framework e deve ser respeitada para que os eventos funcionem.

Mobileinit

Um dos primeiros eventos que podemos utilizar é o `mobileinit`, que é executado quando o framework está devidamente carregado, mas as páginas ainda não foram

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

desenhadas. Um exemplo que podemos fazer neste evento é alterar o tema de todas as páginas que serão carregadas, sem a necessidade de usar o `data-role`, conforme o código a seguir.

```
$( document ).on( "mobileinit", function() {
  console.log("mobileinit");
  $.mobile.page.prototype.options.theme = "b";
});
```

Neste exemplo, usamos o método `on` do jQuery para capturar o evento `mobileinit` que o jQuery Mobile disparará. Quando isso acontece, além de exibir uma mensagem no log do navegador, usamos a variável `$.mobile` para alterar o comportamento da página, por meio de *prototype*.

orientationchange

Este evento é disparado quando o dispositivo muda de orientação (retrato/paisagem). Existem dezenas de eventos que podem ser consultados na API do jQuery Mobile, em https://api.jquerymobile.com/category/events/.

Eventos touch

Além de eventos relacionados à inicialização, criação e mudança de páginas, o jQuery Mobile possui eventos relacionados ao touch do dispositivo, como por exemplo, o ato de deslizar o toque para a direita (`swiperight`), ou para a esquerda (`swipeleft`).

Vamos a um exemplo muito comum em listas, que é a ação de deslizar para a direita com o propósito de excluir um item. Para exemplificar este processo, vamos criar o arquivo `jqm-starter/public/swipe-delete.html` , que, além do código

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

comum de uma página HTML, contém a seguinte lista:

```
<ul id="list" data-role="listview">
  <li >
    <span>Banana</span>
  </li>
  <li >
    <span>Maçã</span>
  </li>
  <li>
    <span >Pera</span>
  </li>
  <li >
    <span >Laranja</span>
  </li>
  <li >
    <span >Morango</span>
  </li>
</ul>
```

No final do arquivo `swipe-delete`, após incluir o `script.js` e antes do elemento `<body>`, vamos adicionar o código capaz de retirar o elemento da lista. De início, podemos ter o seguinte código:

```
<script type="text/javascript" src="js/script-min.js"></script>
<script type="text/javascript">
    $("#list").on("swiperight",">li",function(event){

    });
</script>
</body>
```

Aqui usamos o seletor jQuery para pegar o elemento cuja id é `list`, e usamos o método `on` para relacionar um evento, neste caso, o `swiperight`. Perceba que usamos o filtro `">li"` como segundo parâmetro do método, para que possamos obter somente a linha em que o evento touch está acontecendo. Neste caso, a linha é o próprio `<li>`. Como terceiro parâmetro, passamos a

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

função que será chamada quando o evento acontecer, que é exibida a seguir.

```
$("#list").on("swiperight",">li",function(event){

  var li = $(this);
  var span = li.children();
  console.log("Excluindo " + span.html());

    $(this).animate({marginLeft: parseInt($(this).css('marginLeft
),10) === 0 ? $(this).outerWidth() : 0 }).fadeOut('fast',function
(){li.remove()});

});
```

Na função, obtemos o `<li>` como teste, por meio do `$(this)` e o `<span>`, que contém o texto do item da lista. Fazemos isso para exibir uma mensagem no console. O método que retira a linha é realizado pelo comando `animate`. Após o evento de animar a saída da linha, usamos `li.remove()` para remover o item realmente da lista `<ul>`. Neste ponto, pode-se chamar uma função que removeria os dados através de um acesso ao servidor. Verifique o código completo deste exemplo em http://bit.ly/jqm030.

O evento de arrastar o toque para a direta ou esquerda é compreendido pelo dispositivo como o movimento de 30 pixels, em menos de um segundo. Este é um valor padrão que pode ser alterado caso haja necessidade. Essa alteração envolve quatro propriedades adicionais do evento `swipe`, descritas a seguir.

- `.event.special.swipe.scrollSupressionThreshold` – Com o valor padrão de 10 pixels, configura a altura ou comprimento máximo que o toque no dispositivo

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

pode ser realizado. Por exemplo, em um swipe horizontal, há uma margem de 10 pixels de altura que o dedo do usuário pode percorrer. Se esta altura for ultrapassada, o evento de swipe não será disparado. Isso evita, por exemplo, fazer um swipe na diagonal da tela do dispositivo.

- `$.event.special.swipe.durationThreshold` – É a duração em milissegundos em que o toque deve percorrer uma determinada distância para que o evento de swipe seja considerado válido. O valor padrão é 1000ms, ou seja, 1 segundo.

- `$.event.special.swipe.horizontalDistanceThreshold` – É a distância na horizontal que o toque deve percorrer antes do tempo indicado pelo `durationThreshold`. O valor padrão é 30px, mas um valor ideal seria 100px. Pode-se editar este valor alterando-o no arquivo `events.js` do projeto `jqm-starter`, conforme o exemplo a seguir.

```
$( document ).on( "mobileinit", function() {
    console.log("mobileinit");
    $.event.special.swipe.horizontalDistanceThreshold = 100;
});
```

- `$.event.special.swipe.verticalDistanceThreshold` – O mesmo comportamento do item anterior, só que na vertical.

Existem mais dois eventos relacionados ao toque do dispositivo mobile. Veremos estes a seguir:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

- `Tap` – Ele é usado quando o usuário rapidamente toca na tela do dispositivo. O evento é semelhante ao `click` de um mouse, e pode ser usado para realizar alguma ação específica em qualquer objeto da tela. Não é preciso utilizar este evento para um link ou um item de menu, pois eles já possuem essa funcionalidade.

- `TapHold` – Ele é caracterizado quando o usuário toca na tela e a mantém pressionada. Este evento é útil caso queira exibir algum menu quando o usuário "segura" um objeto na tela. Entretanto, a configuração deste processo envolve, além de JavaScript, a instalação de bibliotecas extras como o `jQuery UI Touch Punch`, o que não será abordado neste livro. Nossa recomendação é que utilize menus dropdown para a tarefa de abrir um menu de opções.

4.2 ALETRANDO PÁGINAS VIA JAVASCRIPT

Algumas vezes será necessário alterar uma página do jQuery Mobile via JavaScript. Para realizar esta tarefa, usamos o método `change()` do widget `Pagecontainer`. O `Pagecontainer` é um objeto que representa a página ativa, que pode ser obtido por meio do seletor do jQuery.

Por exemplo, se a sua página possui o id `pagina1`, pode-se obter uma instância do `pagecontainer` da seguinte forma:

```
$('#pagina1').pagecontainer();
```

Para evocar o método `change()`, usamos:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
$('#pagina1').pagecontainer("change","nova_pagina.html");
```

Se houver a necessidade de alterar a página para uma página modal, devemos usar:

```
$('#pagina1').pagecontainer("change","nova_pagina.html",{role:"
dialog"});
```

4.3 CONCLUSÃO

Percebe-se o pouco uso de JavaScript nas páginas jQuery Mobile, graças ao framework que possui uma arquitetura focada em requisições Ajax como padrão. No próximo capítulo, veremos um exemplo de como essa arquitetura funciona e de como podemos nos beneficiar com ela.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Capítulo 5

AJAX

Requisições Ajax são realizadas no jQuery Mobile através do uso da variável `$.ajax()` do próprio jQuery, cuja API pode ser encontrada em http://bit.ly/ajaxapi. É válido lembrar de que o uso do `$.ajax()` será minimizado tanto quanto possível, já que todas as nossas páginas em jQuery Mobile já têm um suporte ao carregamento de páginas com Ajax, realizado automaticamente.

5.1 TRANSIÇÃO ENTRE PÁGINAS COM AJAX

No exemplo a seguir, vamos ilustrar este processo. Primeiro, crie o diretório `jqm-ajax-exemplo`, e crie os seguintes arquivos: `index.html`, `pagina2.html` e `pagina3.html`. O arquivo `index.html` é o principal do projeto, e contém as bibliotecas do jQuery Mobile via CDN, conforme o código:

```
<html>
<head>
    <title> jQuery Mobile </title>
    <meta name="viewport" content="width=device-width, initial-sc
ale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.
4.5/jquery.mobile-1.4.5.min.css" />
    <script src="http://code.jquery.com/jquery-1.11.1.min.js"></s
cript>
    <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobil
e-1.4.5.min.js"></script>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
</head>
<body>
    <div data-role="page">
        <div data-role="header">
            <h1> Exemplo Ajax </h1>
        </div>
        <div data-role="content">
            <ul data-role="listview" data-inset="true">
                <li><a href="pagina2.html">Página 2</a></li>
            </ul><br>
        </div>
    </div>
</body>
</html>
```

Veja que o arquivo `index.html` possui uma lista que contém um link para `pagina2.html`. Vamos agora criar este arquivo e adicionar o seguinte HTML:

```
<div data-role="page">
    <div data-role="header">
        <a href="index.html" class="ui-btn" data-rel="back">Volta
</a>
        <h1>Página 2</h1>
        <a href="pagina3.html" class="ui-btn">Opções</a>
    </div>
    <div data-role="main" class="ui-content">
        <p>Esta é a página 2</p>
    </div>
</div>
```

Neste arquivo, não inserimos as bibliotecas do jQuery Mobile, nem criamos os elementos `html`, `head` e `body`. Isso acontece porque todas as bibliotecas já foram carregadas no arquivo `index.html` e, quando clicamos no link do `index.html`, o jQuery Mobile intercepta este clique e usa Ajax para carregar a `pagina2.html`. Por usar Ajax, é necessário executar este exemplo em um servidor web, podendo ser o Apache, o *express* ou o *http-server*.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Para testar este exemplo, abra o terminal no diretório `jqm-ajax-exemplo` e digite `http-server` para iniciar o servidor web. Acesse a URL e a porta (provavelmente é 127.0.0.1:8080) para ver o arquivo `index.html`. Clique no item da lista e perceba que a `pagina2.html` é carregada com a formatação do jQuery Mobile.

O que aconteceu aqui foi que, quando você clicou no link, o jQuery Mobile interceptou este clique, cancelou a ação padrão de carregar uma página, iniciou o Ajax para obter o conteúdo HTML do arquivo `pagina2.html` e, após obter este conteúdo, adicionou-o na DOM do `index.html`. Por este motivo, não é necessário implementar Ajax utilizando o `$.ajax` do jQuery.

No cabeçalho da página 2, temos dois botões. O primeiro é o botão `Voltar`, que usa a diretiva `data-rel='back'` para que o jQuery Mobile possa realizar o efeito de voltar uma página. O segundo botão é o `Opções`, que possui um link para `pagina3.html`, descrito a seguir:

```
<div data-role="page" data-dialog="true">
    <div data-role="header">
        <h1>Página 3</h1>
    </div>
    <div data-role="main" class="ui-content">
        <p>Esta é a página 3</p>
    </div>
    <div data-role="footer">
        <h4>Rodapé</h4>
    </div>
</div>
```

Como diretiva adicional no `pagina3.html`, temos o `data-dialog='true'`, que formata a página para ser exibida como uma caixa de diálogo. Mesmo como caixa de diálogo, o comportamento Ajax do jQuery Mobile é o mesmo.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Perceba que, ao acessar `pagina3.html` diretamente pelo navegador, a página está totalmente desconfigurada, pois ela depende do jQuery Mobile para ser carregada. Desta forma, sempre inicie a sua aplicação pelo `index.html` e, por meio dele, faça o carregamento das outras páginas.

5.2 REALIZANDO UMA CHAMADA AJAX

Pode-se usar jQuery para realizar uma chamada Ajax na página. No exemplo a seguir, incluímos um botão na `index.html` que vai simular o acesso a uma URL. Neste caso, estaremos acessando `foo.json` , que retorna uma informação no formato JSON.

```
<div data-role="page">
        <div data-role="header">
            <h1> Exemplo Ajax </h1>
        </div>
        <div data-role="content">
            <ul data-role="listview" data-inset="true">
                <li><a href="pagina2.html">Página 2</a></li>
            </ul><br>
            <button id="testAjax" class="ui-btn">Click To Test</b
utton>
        </div>
    </div>
    <script type="text/javascript">
    $("#testAjax").click(function(){
        $.ajax({url: "foo.json", success: function(result){
            console.log(result);
            $("#testAjax").html("Hello " + result.name);
        }});
    });
    </script>
```

Neste exemplo, criamos um botão com o id `testAjax` , e usamos jQuery para inicialmente adicionar um *listener* ao evento

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

`click` , através do `$("#testAjax").click` . Então, usamos `$.ajax` , nativo do jQuery, para realizar uma requisição à URL `foo.json` , que contém o seguinte conteúdo:

```
{name:"daniel"}
```

Após realizar esta requisição, alteramos o HTML do botão adicionando uma mensagem de boas-vindas, concatenando com o retorno do `json` na propriedade `result.name` .

5.3 CONCLUSÃO

Neste capítulo, vimos como o jQuery Mobile trata as requisições entre as páginas com Ajax, possibilitando a criação de páginas HTML somente com o seu conteúdo principal. Este tipo de processamento é muito útil para a criação de uma aplicação em jQuery Mobile, que veremos no próximo capítulo.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

CAPÍTULO 6

EXEMPLO COM PHP

Neste capítulo, vamos criar um pequeno sistema de tarefas utilizando Apache, PHP e MySQL. Para que possamos nos concentrar somente no jQuery Mobile, e não no PHP, vamos a princípio realizar o download de um projeto básico, que já contém códigos para criar o banco de dados e as tabelas.

6.1 OBTENDO OS ARQUIVOS INICIAIS

Acesse a seguinte URL: https://github.com/danielschmitz/jquerymobile-codigos/archive/php-tasker-1.1.zip. Depois, faça o download do arquivo `zip`, descompactando-o logo em seguida. Copie a pasta `jqm-php-tasker` para o *document root* do servidor. No Windows, temos o `c:\wamp\www` e, no Linux, temos `/var/www`.

6.2 INSTALANDO O BANCO DE DADOS

Após copiar a pasta, acesse o seguinte endereço `http://localhost/jqm-php-tasker/install.php` para abrir a página de instalação do banco de dados. Inicialmente, clique em `Testar conexão com banco de dados`, para que seja feito um teste de conexão com o banco.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Esse teste usa as configurações do arquivo `config.php`, que deve ser alterado de acordo com o seu banco. No Wamp Server, o usuário é `root` e a senha é em branco. No Linux, o usuário e senha são pedidos na instalação do MySQL.

Se você obtiver a mensagem `Conectou!`, pode partir para a segunda parte da instalação, que é criar o banco de dados `tasker` e as tabelas. Para isso, clique no link `Criar tabelas`. Após a criação das tabelas, surge a mensagem `Tabelas criadas!` e o ambiente está pronto para iniciarmos o desenvolvimento da aplicação mobile.

6.3 CRIANDO A TELA INICIAL DE LOGIN

Retornando ao arquivo `index.php`, temos o HTML padrão da aplicação, conforme o código a seguir:

```
<html>
<head>
    <title> jQuery Mobile </title>
    <meta name="viewport" content="width=device-width, initial-sc
ale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.
4.5/jquery.mobile-1.4.5.min.css" />
    <script src="http://code.jquery.com/jquery-1.11.1.min.js"></s
cript>
    <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobil
e-1.4.5.min.js"></script>
</head>
<body>
    <div data-role="page">
        <div data-role="header">
            <h1> Exemplo Php </h1>
        </div>
        <div data-role="content">
    Hello World
        </div>
    </div>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
</body>
</html>
```

Nesse HTML, temos uma página com um simples *Hello World*. A primeira tarefa será alterar essa página e criar um formulário de login, além do botão para realizar o cadastro de usuários. Qualquer pessoa que acessa a página pode se cadastrar. Primeiro, alteramos o cabeçalho:

```
<div data-role="header">
    <h1> Tasker </h1>
    <a href="signup.php" class="ui-btn ui-btn-right">Cadastrar</a

</div>
```

Neste cabeçalho, incluímos um link para `signup.php` com as classes `ui-btn`, que transforma o link em um botão, e `ui-btn-right`, que adiciona o botão na parte direita do cabeçalho.

No corpo da página, criamos um formulário de login, conforme o código:

```
 <div data-role="content">
          <form action="doLogin.php" method="POST">
             <div class="ui-field-contain">
                 <label for="email">Email:</label>
                 <input type="email" name="email" id="email" valu
="<?php echo $_POST['email']?>">
             </div>
             <div class="ui-field-contain">
                 <label for="password">Password:</label>
                 <input type="password" name="password" id="passw
ord" value="<?php echo $_POST['password']?>">
             </div>
             <input type="submit" value="Logar">
         </form>
     </div>
```

Este código produz o seguinte resultado:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

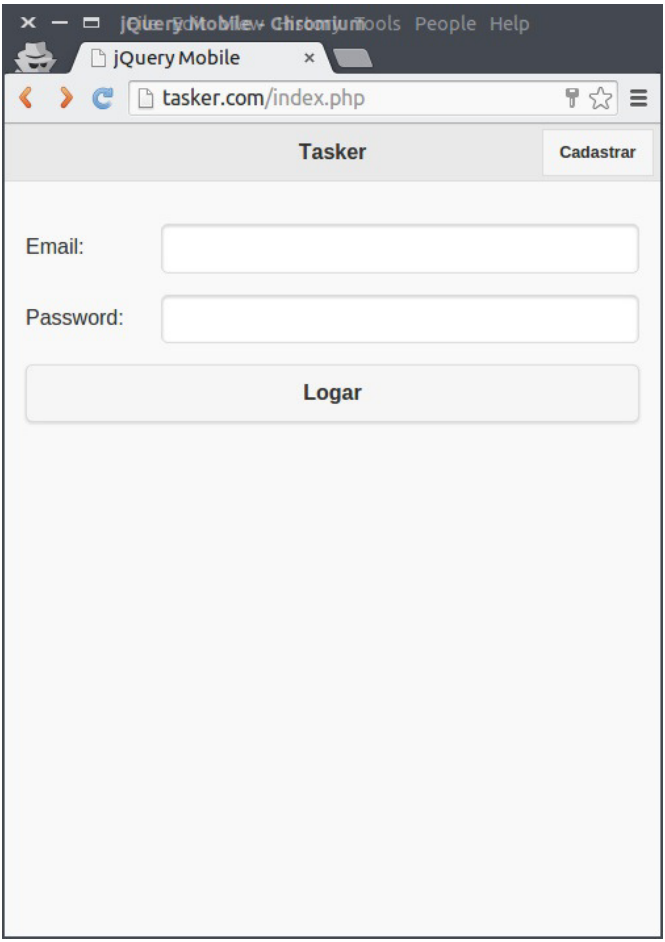


Figura 6.1: Tela inicial de login

Perceba que o submit do formulário aponta para o arquivo `doLogin.php` . Este arquivo contém o código PHP que vai verificar o login do usuário. Mas antes de fazer essa verificação, devemos criar a tela para cadastrar usuários.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

6.4 CADASTRO DO USUÁRIO

Antes de darmos prosseguimento ao login do usuário, vamos criar a tela para o seu cadastro. No HTML anterior, existe um botão chamado `Cadastrar`, que aponta para a página `signup.php`. Inicialmente, esta tela possui os três campos da tabela `user`, para que o usuário possa se registrar. O HTML inicial desta tela é apresentado a seguir:

```
<div data-role="page">
    <div data-role="header">
        <a href="index.php" class="ui-btn" data-rel="back">Voltar
</a>
        <h1> Cadastro </h1>
    </div>
    <div data-role="content">
        <i style="color:red"><?php echo $errorMessage ?></i>
        <form action="signup.php" method="POST">
           <div class="ui-field-contain">
               <label for="name">Nome:</label>
               <input type="text" name="name" id="name" value="<?
php echo $_POST['name']?>">
           </div>
           <div class="ui-field-contain">
               <label for="email">Email:</label>
               <input type="email" name="email" id="email" value=
<?php echo $_POST['email']?>">
           </div>
            <div class="ui-field-contain">
               <label for="password">Password:</label>
               <input type="password" name="password" id="passwor
d" value="<?php echo $_POST['password']?>">
           </div>
            <input type="submit" value="Enviar">
       </form>
   </div>
</div>
```

Esse HTML produz o seguinte resultado:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

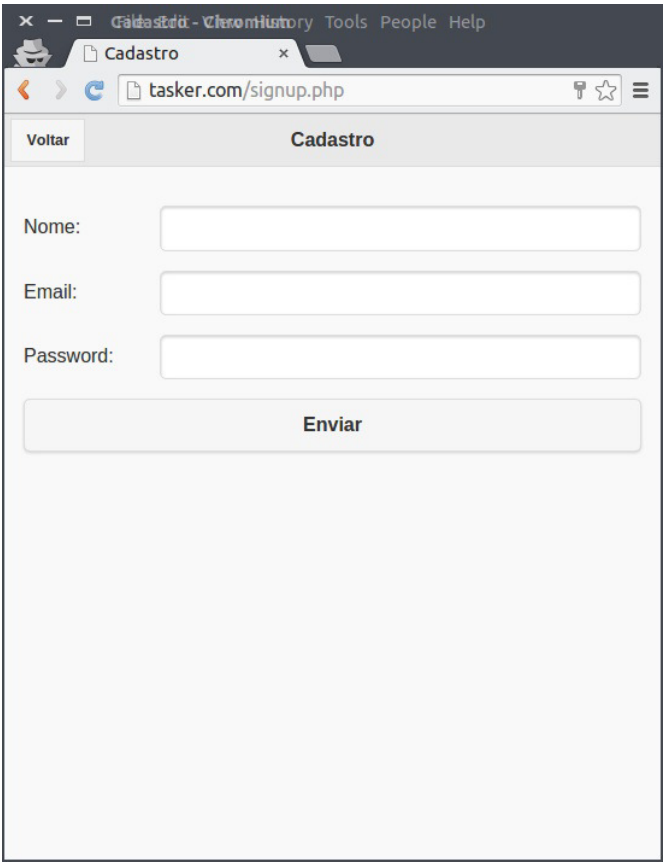


Figura 6.2: Tela de cadastro de usuários

Nele temos um formulário simples, tendo como `action` a sua própria página `signup.php`. Isso significa que, quando o usuário preencher os dados do cadastro e clicar no botão `Cadastrar`, o formulário será submetido para a própria página.

Como a página é submetida para ela mesma, o código PHP que faz a inclusão do usuário estará na própria página que contém o formulário. Talvez este não seja o melhor caminho no

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

desenvolvimento PHP, já que o ideal é termos os arquivos HTML separados dos arquivos PHP, mas lembre-se de que nosso foco é o jQuery Mobile, e não as linguagens que executam no lado do servidor.

Quando o usuário clica no botão `cadastrar`, inicia-se o código PHP, a princípio incluindo alguns outros arquivos para dar suporte ao cadastro. Ou seja, logo no início do arquivo `signup.php` temos:

```
<?php
include 'config.php';
include 'functions.php';
include 'DB.php';
include 'User.php';
?>
```

Os arquivos `config.php` e `DB.php` já são conhecidos. O primeiro informa as configurações de acesso ao banco, como usuário e senha. O segundo é uma camada de acesso PDO (*PHP Data Objects*) nativo do PHP. O arquivo `functions.php` conterá diversas funções úteis no desenvolvimento do sistema, como por exemplo, a função `ShowDialog` exibida a seguir:

```
<?php
function ShowDialog($caption,$text,$page=""){

    $link = "";
    if ($page != "")
        $link = "<a href='{$page}' class='ui-btn'>Ok</a> ";

    return "
    <div data-role='page' data-dialog='true'>
            <div data-role='header'>
                <h1> {$caption} </h1>
            </div>
            <div data-role='content'>
                    {$text}
                    {$link}
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
            </div>
        </div>
    ";
}
?>
```

Nesta função, retornamos um texto que exibe uma caixa de diálogo em jQuery Mobile, contendo um título, um texto e um link para redirecionar a página quando o usuário clicar no botão `Ok`. Nós usaremos essa função para exibir uma mensagem de sucesso ao usuário, assim que ele se cadastrar, conforme a figura:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

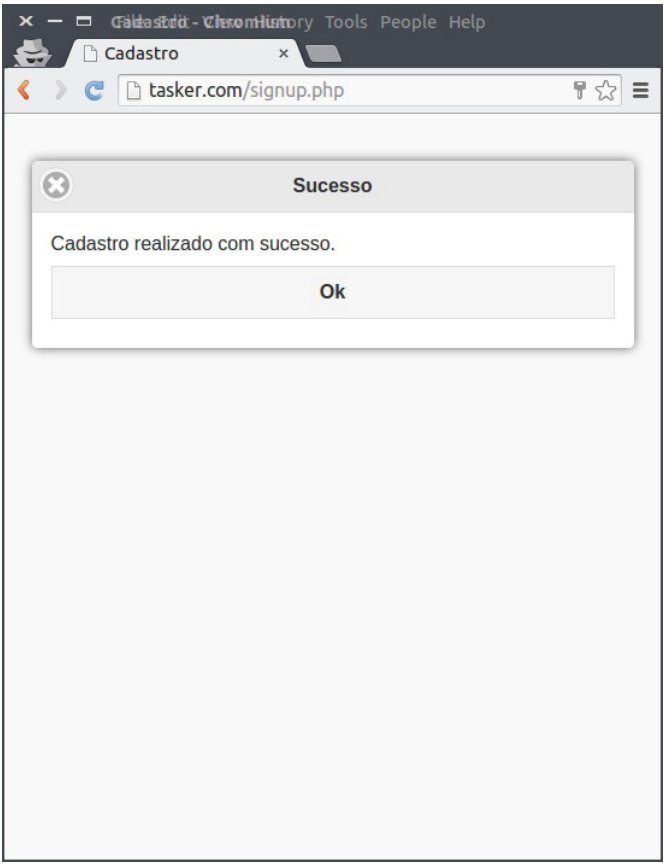


Figura 6.3: Popup da função ShowDialog

Também incluímos o arquivo `User.php` que contém métodos para acesso à tabela de usuários. Qualquer código que manipula dados dos usuários estará neste arquivo. A princípio, temos o seguinte conteúdo:

```
<?php
class User{

    public function create($name,$email,$password){
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
        $this->checkEmail($email);
        $sql = "INSERT INTO users (name,email,password) VALUES (:
name,:email,:password)";
        $stmt = DB::prepare($sql);
        $stmt->bindParam("name", $name);
        $stmt->bindParam("email", $email);
        $stmt->bindParam("password", $password);
        $stmt->execute();
        $id_user = DB::lastInsertId();
        return $id_user;
    }

    public function checkEmail($email){
        $sql = "SELECT id FROM users WHERE email=:email";
        $stmt = DB::prepare($sql);
        $stmt->bindParam("email", $email);
        $stmt->execute();
        $user = $stmt->fetch();
        if ($user)
            throw new \Exception("Email já está cadastrado");
    }

}
?>
```

Na classe `User`, temos dois métodos. O primeiro, `create`, usa PDO para realizar o comando `INSERT` na tabela `users`. Perceba que, antes de incluir, o método `checkEmail` é chamado para verificar se o e-mail do usuário está cadastrado. Se já estiver na tabela, disparamos uma exceção e, desta forma, o método `create` não será concluído.

Voltando ao arquivo `signup.php`, após incluir os arquivos PHP necessários para incluir um usuário, temos:

```
<?php
include 'config.php';
include 'functions.php';
include 'DB.php';
include 'User.php';
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
$errorMessage = "";

if ( !empty($_POST['name'])
        && !empty($_POST['email'])
        && !empty($_POST['password'])
    ){

    $name = $_POST['name'];
    $email = $_POST['email'];
    $password = $_POST['password'];

    try {
        $u = new User();
        $u->create($name,$email,$password);
        $html = ShowDialog("Sucesso","Cadastro realizado com suce
sso.","index.php");
        echo $html;
    } catch (\Exception $e) {
        $errorMessage = $e->getMessage();
    }
}
?>
```

Veja que temos um `if` verificando se os campos `$_POST` estão preenchidos. Todos eles (`name` , `email` , `password`) devem estar preenchidos para que possamos incluir o usuário na tabela. No PHP, nós usamos `$_POST['name']` para obter o conteúdo digitado no campo de nome do usuário. Fizemos isso nos três campos do formulário, e criamos as variáveis `$name` , `$email` e `$password` .

Para cadastrar o usuário, usamos a classe `User` , previamente inserida no `include` do arquivo `user.php` . Usamos também o `try..catch` para exibir alguma mensagem de erro, caso exista. Se não houver nenhum erro, a função `ShowDialog` é executada exibindo a mensagem de que o usuário foi cadastrado.

A seguir, veja o arquivo `signup.php` completo:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
<?php
include 'config.php';
include 'functions.php';
include 'DB.php';
include 'User.php';

$errorMessage = "";

if ( !empty($_POST['name'])
        && !empty($_POST['email'])
        && !empty($_POST['password'])
    ){

    $name = $_POST['name'];
    $email = $_POST['email'];
    $password = $_POST['password'];

    try {
        $u = new User();
        $u->create($name,$email,$password);
        $html = ShowDialog("Sucesso","Cadastro realizado com suce
sso.","index.php");
        echo $html;
    } catch (\Exception $e) {
        $errorMessage = $e->getMessage();
    }
}
?>
<div data-role="page">
    <div data-role="header">
        <a href="index.php" class="ui-btn" data-rel="back">Voltar
</a>
        <h1> Cadastro </h1>
    </div>
    <div data-role="content">
        <i style="color:red"><?php echo $errorMessage ?></i>
        <form action="signup.php" method="POST">
           <div class="ui-field-contain">
               <label for="name">Nome:</label>
               <input type="text" name="name" id="name" value="<?
php echo $_POST['name']?>">
           </div>
           <div class="ui-field-contain">
               <label for="email">Email:</label>
               <input type="email" name="email" id="email" value=
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
"<?php echo $_POST['email']?>">
           </div>
            <div class="ui-field-contain">
               <label for="password">Password:</label>
               <input type="password" name="password" id="passwor
d" value="<?php echo $_POST['password']?>">
           </div>
                <input type="submit" value="Enviar">
       </form>
   </div>
</div>
```

6.5 REALIZANDO O LOGIN

Como vimos no capítulo anterior, o submit do formulário de login carrega o arquivo `doLogin.php`, que vai verificar os dados do usuário. Este arquivo tem a função de verificar o e-mail e senha que o usuário digitou, e caso os encontre na tabela `users`, realizar o login. O login é feito por meio da sessão do PHP, que armazena os dados do usuário.

Como de costume, o arquivo `doLogin.php` começa com a inclusão das bibliotecas que precisamos utilizar:

```
include 'config.php';
include 'functions.php';
include 'DB.php';
include 'User.php';
```

A próxima etapa consiste em verificar se o usuário digitou o e-mail e a senha. Caso algum deles esteja vazio, devemos exibir uma mensagem de erro.

```
if ( !empty($_POST['email'])
    && !empty($_POST['password'])
    )
{
   //login
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
}
else
{
   $html = ShowDialog("Erro","Preencha o email/senha","index.php"
);
   echo $html;
}
```

Veja que usamos a função `ShowDialog` para exibir a mensagem de erro, e redirecionamos o fluxo da requisição para o arquivo `index.html`. Se o usuário preencher o e-mail e a senha, podemos então verificar o login, conforme o código a seguir:

```
$email = $_POST['email'];
$password = $_POST['password'];

$u = new User();
if ($user = $u->checkLogin($email,$password)){
      $_SESSION['user_id'] = $user->id;
      $_SESSION['user_name'] = $user->name;
      $_SESSION['user_email'] = $user->email;
      $html = ShowDialog("Sucesso","Olá {$user->name}, você logou
 com sucesso ","tasks.php");
      echo $html;
  }else{
      $html = ShowDialog("Erro","Login senha incorretos","index.p
hp");
      echo $html;
  }
```

Na verificação do login, usamos novamente a classe `User` e um novo método, chamado `checkLogin`. Se este retornar o usuário, usamos `$_SESSION` para registrar as informações do usuário que vieram do banco, e exibimos uma mensagem de sucesso. Se o usuário errar o e-mail ou a senha, `$user` será nulo e exibiremos a mensagem de erro.

O código completo do arquivo `doLogin.php` é exibido a seguir.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
<?php
include 'config.php';
include 'functions.php';
include 'DB.php';
include 'User.php';
if ( !empty($_POST['email'])
    && !empty($_POST['password'])
    )
{
    $email = $_POST['email'];
    $password = $_POST['password'];
    $u = new User();
    if ($user = $u->checkLogin($email,$password)){
        $_SESSION['user_id'] = $user->id;
        $_SESSION['user_name'] = $user->name;
        $_SESSION['user_email'] = $user->email;
        $html = ShowDialog("Sucesso","Olá {$user->name}, você log
ou com sucesso ","tasks.php");
        echo $html;
    }else{
        $html = ShowDialog("Erro","Login senha incorretos","index
.php");
        echo $html;
    }
}
else
{
   $html = ShowDialog("Erro","Preencha o email/senha","index.php"
);
   echo $html;
}
```

Com o usuário logado, a mensagem de sucesso redireciona para o arquivo `tasks.php` , que exibiremos a seguir.

```
<?php
include 'config.php';
include 'functions.php';
include 'DB.php';
include 'Task.php';
$t = new Task();
$tasks = $t->getByStatus("0");
$tasks_list = "";
foreach ($tasks as $task) {
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
    $tasks_list .= "<li id='{$task->id}'>{$task->title}</li>";
}
?>
<div data-role="page">
    <div data-role="header">
        <h1> Tarefas </h1>
        <a href="taskAdd.php" class="ui-btn ui-btn-right">Adicion
ar</a>
    </div>
    <div data-role="content">
        <ul id="list" data-role="listview">
            <?php echo $tasks_list?>
        </ul>
    </div>
    <script type="text/javascript">
    $("#list").on("swiperight",">li",function(event){

      var li = $(this);
      var span = li.children();
      console.log("Excluindo " + span.html());
      var idTask = li.attr('id');
      $(this).animate({marginLeft: parseInt($(this).css('marginLe
ft'),10) === 0 ? $(this).outerWidth() : 0 }).fadeOut('fast',funct
ion(){li.remove()});
       //completa a tarefa
       $.ajax({url: "doCompleteTask.php?taskId="+idTask, success:
 function(result){
            console.log(result);
        }});
  });
    </script>
</div>
```

6.6 ADICIONANDO UMA TAREFA

Após o login, o usuário é redirecionado para a página `tasks.php` , que contém a lista de tarefas cadastradas pela tabela `tasks` e um botão para adicionar uma nova tarefa. O cabeçalho da página possui o seguinte código HTML:

```
<div data-role="header">
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
  <h1> Tarefas </h1>
  <a href="taskAdd.php" class="ui-btn ui-btn-right">Adicionar</a>
</div>
```

O arquivo `tasksAdd.php` contém o seguinte formulário para adicionar uma tarefa:

```
<div data-role="page">
  <div data-role="header">
    <a href="index.php" class="ui-btn" data-rel="back">Voltar</a>
    <h1> Adicionar Tarefa </h1>
  </div>
  <div data-role="content">
    <form action="taskAdd.php" method="POST">
     <div class="ui-field-contain">
       <label for="title">Título:</label>
       <input type="text" name="title" id="title" value="<?php ec
ho $_POST['title']?>">
     </div>
     <input type="submit" value="Enviar">
   </form>
 </div>
</div>
```

O formulário faz o post para a própria página, onde devemos adicionar o código PHP para incluir a tarefa na tabela `tasks`:

```
include 'config.php';
include 'functions.php';
include 'DB.php';
include 'Task.php';
```

Nos `includes` que precisamos fazer, além dos já conhecidos `config`, `functions` e `DB`, agora temos `Task.php`, que contém inicialmente o método para fazer a inclusão de dados na tabela `tasks`.

A primeira verificação que faremos no arquivo é analisar se o usuário está logado no sistema, avaliando se a variável `user_id` está na sessão:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
if (empty($_SESSION['user_id'])){
  echo ShowDialog("Erro","Necessário Login.","index.php");
 }
```

Se `$_SESSION['user_id']` for vazio, o usuário não está logado e não pode continuar. Criamos então uma mensagem de erro redirecionando a página para o `index.php` . Se o usuário estiver logado, procedemos com a criação da tarefa:

```
  if (empty($_SESSION['user_id'])){
  echo ShowDialog("Erro","Necessário Login.","index.php");
 }
 else
 {
    if (!empty($_POST['title'])){
      $t = new Task();
      $t->create($_POST['title']);
      $html = ShowDialog("Sucesso","Tarefa criada com sucesso","t
asks.php");
      echo $html;
    }
 }
```

Usamos o método `create` da classe `Task` para adicionar o registro. Este é apresentado a seguir:

```
 <?php
class Task{

     public function create($title){

        if (empty($_SESSION['user_id']))
            throw new Exception("Necessário login.");

        $status = '0'; //incompleta
        $user_id = $_SESSION['user_id'];
        $created = date('Y-m-d');

        $sql = "INSERT INTO tasks (title,status,created,user_id)
VALUES (:title,:status,:created,:user_id)";
        $stmt = DB::prepare($sql);
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
        $stmt->bindParam("title", $title);
        $stmt->bindParam("status", $status);
        $stmt->bindParam("created", $created);
        $stmt->bindParam("user_id", $user_id);
        $stmt->execute();

        $id_task = DB::lastInsertId();
        return $id_task;
    }

}
?>
```

O método `create` verifica novamente se o usuário está logado e, em caso positivo, realiza a inserção da tarefa na tabela. Alguns campos extras são necessários, como o status (`0` indica que a tarefa não está concluída), o usuário logado e a data de criação da tarefa.

Com a tarefa incluída, uma mensagem de sucesso é apresentada, retornando o fluxo para o arquivo `tasks.php` novamente, onde devemos exibir as tarefas criadas, exibidas no tópico a seguir.

Listando tarefas

Para listar as tarefas, precisamos selecionar todas as tarefas do usuário que não estão completas (`status=0`). Para isso, criamos um método no arquivo `Task.php` chamado `getByStatus`, apresentado a seguir:

```php
<?php
class Task{

  //outros métodos

public function getByStatus($status="0"){

    if (empty($_SESSION['user_id']))
        throw new Exception("Necessário login.");

    $user_id = $_SESSION['user_id'];

    $sql = "SELECT * FROM tasks WHERE status=:status and user_id=
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
:user_id";
    $stmt = DB::prepare($sql);
    $stmt->bindParam("status", $status);
    $stmt->bindParam("user_id", $user_id);
    $stmt->execute();
    return $stmt->fetchAll();
}
}
?>
```

Após criar o método, podemos voltar ao arquivo `tasks.php` e criar a listagem de tarefas do usuário.

```
<?php
include 'config.php';
include 'functions.php';
include 'DB.php';
include 'Task.php';

$t = new Task();
$tasks = $t->getByStatus("0");

$tasks_list = "";
foreach ($tasks as $task) {
    $tasks_list .= "<li id='{$task->id}'>{$task->title}</li>";
}

?>
```

Veja que chamamos o método `getByStatus` repassando o status `"0"` e usamos o laço `foreach` para criar uma string contendo uma lista do elemento `<li>`. Essa lista será inserida no elemento `<ul>` que criaremos a seguir, no código HTML que desenha a lista de tarefas.

```
<div data-role="page">
    <div data-role="header">
        <h1> Tarefas </h1>
        <a href="taskAdd.php" class="ui-btn ui-btn-right">Adicion
ar</a>
    </div>
    <div data-role="content">
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
        <ul id="list" data-role="listview">
            <?php echo $tasks_list?>
        </ul>
    </div>
</div>
```

Veja que o arquivo `tasks.php` contém uma lista de elementos cujo os dados são fornecidos pela variável `$tasks_list` . Até o momento, ela apresenta a seguinte interface:

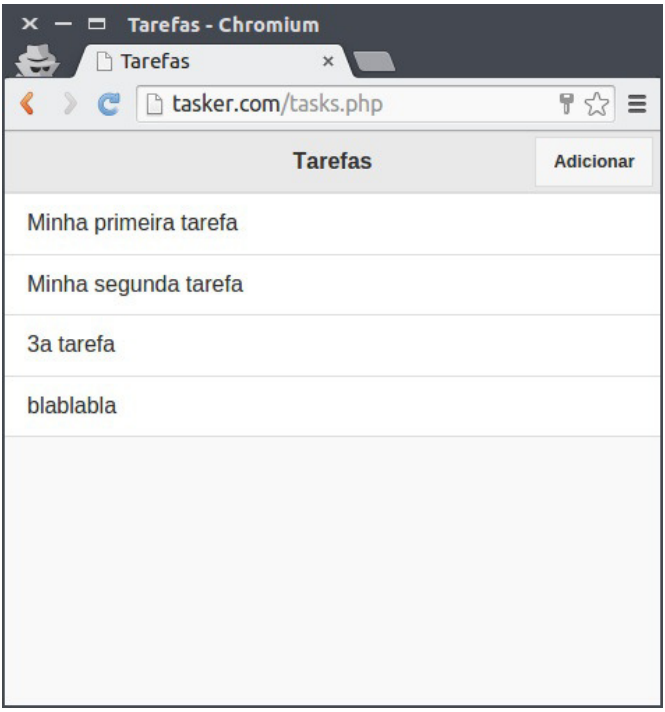


Figura 6.4: Tarefas

Para finalizarmos a aplicação, vamos utilizar o evento `swiperight` aprendido no capítulo *Utilizando JavaScript no jQuery Mobile*, e fazer com que, quando o usuário realizar o swipe na tarefa, ela seja marcada como completada.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Primeiro, vamos adicionar esta funcionalidade na lista, da seguinte forma:

```
<div data-role="page">
    <div data-role="header">
        <h1> Tarefas </h1>
        <a href="taskAdd.php" class="ui-btn ui-btn-right">Adicion
ar</a>
    </div>
    <div data-role="content">
        <ul id="list" data-role="listview">
            <?php echo $tasks_list?>
        </ul>
    </div>
    <script type="text/javascript">
    $("#list").on("swiperight",">li",function(event){

      var li = $(this);
      var span = li.children();
      console.log("Excluindo " + span.html());
      var idTask = li.attr('id');

      $(this).animate({marginLeft: parseInt($(this).css('marginLe
ft'),10) === 0 ? $(this).outerWidth() : 0 }).fadeOut('fast',funct
ion(){li.remove()});

       //completa a tarefa
       $.ajax({url: "doCompleteTask.php?taskId="+idTask, success:
function(result){
            console.log(result);
        }});

  });
    </script>
</div>
```

Perceba que adicionamos um `script` JavaScript no final da página HTML, usando o seletor do jQuery `$("#list")` para obter a lista, e adicionando o evento `swiperight` . Quando esse evento ocorrer, usamos alguns artifícios para realizar o movimento do item se movimentando para a direita. Além disso, também

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

usamos `$.Ajax` para realizar uma requisição Ajax ao servidor, informando que a tarefa deve ser alterada. Essa requisição chama o arquivo `doCompleteTask.php` repassando o id da tarefa por meio da variável `taskId`. O arquivo `doCompleteTask.php` é exibido a seguir:

```
<?php
include 'config.php';
include 'functions.php';
include 'DB.php';
include 'Task.php';

$id = $_GET["taskId"];

header('Content-Type: application/json');

try {

    $t = new Task();
    $t->completeTask($id);
    header("HTTP/1.0 200 Ok");
    echo "{result:'ok'}";

    } catch (Exception $e) {

    header("HTTP/1.0 500 Error");
    echo "{error:'{$e->getMessage()}'}";

}
?>
```

Neste arquivo, obtemos o id da tarefa e repassamos para o método `completeTask` da classe `Task`, que realizará a alteração no banco de dados. Perceba que, como fizemos uma chamada Ajax a este arquivo, ele pode responder no formato JSON, caso seja necessário.

O método `completeTask` é exibido a seguir:

```
<?php
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
class Task{

public function completeTask($id){

     if (empty($_SESSION['user_id']))
        throw new Exception("Necessário login.");

    $user_id = $_SESSION['user_id'];

    $sql = "SELECT * FROM tasks WHERE id=:id and user_id=:user_id
";
    $stmt = DB::prepare($sql);
    $stmt->bindParam("id", $id);
    $stmt->bindParam("user_id", $user_id);
    $stmt->execute();
    $task = $stmt->fetch();

    if ($task!=null){

        $sql = "UPDATE tasks SET status='1' where id=:id";
        $stmt = DB::prepare($sql);
        $stmt->bindParam("id", $id);
        $stmt->execute();
        return true;

    }else
         throw new Exception("Tarefa não existe");
 }
}
?>
```

Neste método, verificamos inicialmente se a tarefa existe para aquele usuário, e em caso positivo, realizamos o update no registro. Desta forma, quando o usuário carregar novamente a lista de tarefas, ela não será mais exibida na lista.

6.7 CONCLUSÃO

Neste capítulo, criamos uma simples aplicação com jQuery Mobile que gerencia tarefas. Vimos como criar várias páginas e

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

prover a interação entre elas, o que é o mais importante neste contexto. Quando estamos trabalhando com jQuery Mobile, estamos na verdade criando código HTML e usando um pouco de JavaScript para prover alguma interação com o usuário.

Independente da linguagem de servidor que você escolher, o processo para criar páginas em jQuery Mobile é o mesmo, como mostrado neste exemplo. É possível prover mais funcionalidades ao jQuery Mobile pelo framework PhoneGap, que será visto no próximo capítulo.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Capítulo 7

PHONEGAP E JQUERY MOBILE

O PhoneGap é um framework usado para criarmos aplicações nativas (Android, iOS, Windows Phone) utilizando como fonte uma aplicação HTML/JavaScript pura. Ou seja, primeiro criamos uma aplicação usando HTML para desenhar a interface, e então usamos JavaScript para adicionar alguma lógica na aplicação.

A partir disso, usamos o PhoneGap para gerar o arquivo de instalação no dispositivo, como por exemplo, um arquivo com a extensão `.apk` para dispositivos Android. Ele pode ser copiado para o seu dispositivo mobile e instalado. Desta forma, podemos criar uma aplicação com jQuery Mobile sendo executada nativamente no dispositivo, sem a necessidade de se utilizar o navegador.

Uma aplicação criada com o PhoneGap pode realizar chamadas a um servidor web. Mas nosso interesse neste momento é criar uma aplicação que utilize um banco de dados interno ao dispositivo, de forma que ela possa ser executada em um celular ou tablet que não esteja conectado à internet.

7.1 INSTALAÇÃO DO PHONEGAP

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Apesar de haver uma interface visual para Windows/Mac, vamos instalar a versão em modo texto que possui opções mais avançadas em relação à interface gráfica.

Usaremos o gerenciador de pacotes `npm` para instalar o PhoneGap, pelo seguinte comando:

```
sudo npm install -g phonegap
```

A diretiva `-g` do comando indica que o PhoneGap será instalado globalmente no sistema, e desta forma pode-se usá-lo em qualquer diretório.

7.2 CRIANDO UMA APLICAÇÃO VIA LINHA DE COMANDO

Abordamos no início deste capítulo que o PhoneGap pode ser utilizado em uma aplicação HTML + JavaScript. Para criar uma aplicação inicial com uma estrutura de arquivos pronta, podemos usar o próprio PhoneGap através do seguinte comando:

```
$ phonegap create helloPhoneGap
```

Este comando criará a estrutura de uma aplicação em PhoneGap, semelhante à estrutura a seguir:

```
helloPhoneGap
   |
   | - config.xml
   | - hooks
   | - plataforms
   | - plugins
   | - www
```

Nela temos como principal diretório o `www`, onde se encontram os arquivos da aplicação HTML, que será exportada

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

para o dispositivo mobile. É por esta estrutura inicial que iniciaremos o projeto.

Executando a aplicação via linha de comando

Após criar a aplicação, pode-se utilizar o seguinte comando:

```
$ phonegap run android
```

Ele vai preparar tudo o que é necessário para executar a sua aplicação em um dispositivo Android. Para que a aplicação seja exibida no dispositivo mobile, este deve estar conectado ao computador via USB com o modo "Depuração USB" ativado.

Testando a aplicação via linha de comando

Pode-se testar a aplicação diretamente no navegador, por meio de um minisservidor que o PhoneGap cria. O comando para testar a aplicação é exibido a seguir.

```
$ phonegap serve
...
[phonegap] starting app server...
[phonegap] listening on 127.0.0.1:3000
[phonegap]
[phonegap] ctrl-c to stop the server
```

Na resposta do comando é informado o endereço o qual o dispositivo mobile (ou o navegador) pode acessar.

7.3 EXPORTANDO A APLICAÇÃO PARA UMA PLATAFORMA VIA PHONEGAP BUILD

O *PhoneGap Build* é um serviço adicional do PhoneGap para a geração da aplicação nativa. Isto é, pode-se gerar o arquivo

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

instalador para Android ou iOS, sem a necessidade de maiores instalações no seu sistema operacional.

Para usá-lo, deve-se criar uma conta no site https://build.phonegap.com/plans, onde é possível criar gratuitamente uma aplicação com até 50MB de espaço. Após acessar o site do PhoneGap, clique no botão `Completely free` e crie uma conta.

Após criar a conta, pode-se gerar a aplicação nativa para o dispositivo. A geração pode ser feita pelo comando:

```
$ phonegap remote build android
```

Será requisitado o login e a senha, e após o processamento, o arquivo `apk` estará disponível para download no site, semelhante à figura a seguir.

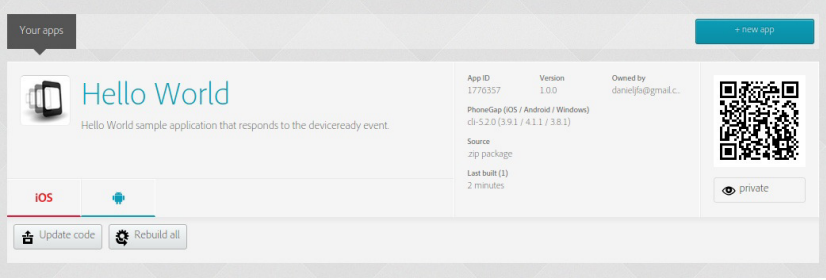


Figura 7.1: Aplicação HelloWorld no PhoneGap Build

Pelo QR code da página, é possível instalar a aplicação. O arquivo `.apk` será baixado para o dispositivo e poderá ser instalado para testes. Perceba que, para instalar o `apk` diretamente, é necessário ativar o item *"Permitir fontes desconhecidas"* no dispositivo seu Android.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

7.4 PREPARANDO UMA APLICAÇÃO JQUERY MOBILE PARA O PHONEGAP

O uso do jQuery Mobile juntamente com o PhoneGap é muito comum e, pela linha de comando, é possível criar uma aplicação com a biblioteca de forma automática através de um template estabelecido.

Para verificar os templates existentes do PhoneGap, use o seguinte comando:

```
$ phonegap template list
blank                   A blank and empty PhoneGap app.
hello-world             Default hello world app for PhoneGap.
hello-cordova           Default hello world app for Cordova.
jquery-mobile-starter   Starter PhoneGap project using jQuery Mob
ile.
```

No último template, temos o jQuery Mobile. Para criar uma aplicação baseada neste template, usamos o comando:

```
$ phonegap create helloPhoneGapjqm --template jquery-mobile-start
er
```

Nele, a aplicação `helloPhoneGapjqm` é criada, juntamente com todas as bibliotecas do jQuery Mobile devidamente instaladas. Ao analisarmos o arquivo `helloPhoneGapjqm\www\index.html`, temos:

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
  <meta name="viewport" content="width=device-width, initial-scal
e=1">
  <link rel="stylesheet" href="lib/jquery.mobile-1.4.5.min.css" />

  <script src="lib/jquery-2.1.1.min.js"></script>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
  <script src="js/app.js"></script>
  <script src="lib/jquery.mobile-1.4.5.min.js"></script>
</head>

<body>

<div data-role="page">

</div>

<script src="cordova.js"></script>
</body>
</html>
```

Perceba que o template segue o padrão que criamos ao longo deste livro. Ele inclui o jQuery, juntamente com o arquivo `app.js` e com uma página mobile no elemento `<body>`.

No final do `<body>`, o arquivo `cordova.js` é incluído, mas ao analisarmos o código-fonte, este arquivo não existe. Isso acontece porque existe um arquivo `cordova.js` para cada tipo de sistema mobile (Android, iOS, Windows phone etc.), e ele é adicionado durante o seu build, e usado para acessar algumas funcionalidades do dispositivo mobile.

> **DICA**
>
> Adicione `<meta charset="utf-8">` no cabeçalho do documento HTML para que a acentuação seja exibida corretamente na aplicação do PhoneGap.

Todos os recursos do jQuery Mobile aprendidos até este momento podem ser utilizados, inclusive o uso de várias páginas

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

HTML, que serão carregadas via Ajax (internamente ao PhoneGap).

Também foi criado o arquivo `www/js/app.js` que conterá o código JavaScript para manipulação da aplicação, como por exemplo, obter dados do usuário, persistir em banco de dados, obter informações do dispositivo, câmera, lista de contatos etc.

7.5 UTILIZANDO PLUGINS

O PhoneGap possui diversos plugins que podem ser adicionados a sua aplicação, de forma a prover acesso nativo ao dispositivo mobile, como acesso a câmera, lista de contatos, notificações etc.

Como exemplo, vamos criar uma aplicação que mostra os detalhes do dispositivo mobile que está usando. Primeiro, crie a aplicação `myDevice` através do seguinte comando:

```
$ phonegap create myDevice --template jquery-mobile-starter
```

Após criar a aplicação, deve-se prepará-la para ser executada em dispositivos Android, pelo comando:

```
$ phonegap run android
```

Com a aplicação pronta, alteramos o arquivo `www/index.html` para o seguinte conteúdo:

```
<!DOCTYPE html>
<html>
<head>
  <title>MyDevice</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scal
e=1">
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
  <link rel="stylesheet" href="lib/jquery.mobile-1.4.5.min.css" />

  <script src="lib/jquery-2.1.1.min.js"></script>
  <script src="js/app.js"></script>
  <script src="lib/jquery.mobile-1.4.5.min.js"></script>
</head>

<body>

<div data-role="page" id="pageone">
  <div data-role="header">
    <h1>myDevice</h1>
  </div>
  <div data-role="main" class="ui-content">
    <ul data-role="listview">
        <li>Cordova version: <span id="cordovaVersion" class="ui-
li-count"></span></li>
        <li>Model: <span id="model" class="ui-li-count"></span></
li>
        <li>Platform: <span id="platform" class="ui-li-count"></s
pan></li>
        <li>UUID: <span id="uuid" class="ui-li-count"></span></li

        <li>Version: <span id="version" class="ui-li-count"></spa
></li>
    </ul>
  </div>
</div>

<script src="cordova.js"></script>
</body>
</html>
```

No arquivo `index.html`, criamos uma página com algumas informações. Elas serão obtidas através do plugin `cordova-plugin-device`, que está pronto para uso. Primeiro, é necessário adicionar o plugin no projeto `myDevice`, e isso é realizado por meio do seguinte comando:

```
$ phonegap cordova plugin add cordova-plugin-device
```

Após incluir o plugin, é preciso alterar o arquivo `config.xml`

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

indicando que esse plugin será usado. Isso é realizado adicionando a seguinte entrada `<plugin name="cordova-plugin-device" spec="1" />` no arquivo `config.xml`, que pode ser adicionada logo após o plugin `whitelist`. Aproveite também para alterar as informações como o nome e a descrição do projeto.

O arquivo `config.xml` ficará semelhante ao exibido adiante:

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="com.phonegap.helloworld" version="0.0.1" xmlns="http:
//www.w3.org/ns/widgets" xmlns:cdv="http://cordova.apache.org/ns/
1.0">
    <name>myDevice</name>
    <description>
        Uma aplicação de exemplo que mostra detalhes do dispositi
vo mobile
    </description>
    <author email="danieljfa@gmail.com" href="https://github.com/
danielschmitz/jquerymobile-codigos">
        Daniel Pace Schmitz
    </author>
    <content src="index.html" />
    <plugin name="cordova-plugin-whitelist" spec="1" />
    <plugin name="cordova-plugin-device" spec="1" />
    <access origin="*" />
    <allow-intent href="http://*/*" />
    <allow-intent href="https://*/*" />
    <allow-intent href="tel:*" />
    <allow-intent href="sms:*" />
    <allow-intent href="mailto:*" />
    <allow-intent href="geo:*" />
    <platform name="android">
        <allow-intent href="market:*" />
    </platform>
    <platform name="ios">
        <allow-intent href="itms:*" />
        <allow-intent href="itms-apps:*" />
    </platform>
    <engine name="android" spec="~4.1.1" />
</widget>
```

Após incluir as informações necessárias no `config`, podemos

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

utilizar o PhoneGap Build para construir a aplicação, pelo seguinte comando:

```
$ phonegap remote build android
[phonegap] compressing the app...
[phonegap] uploading the app...
[phonegap] building the app...
[phonegap] Android build complete
```

Após a construção da aplicação, pode-se acessar o site `build.phonegap.com` e verificar se o plugin device foi devidamente instalado, conforme a figura:

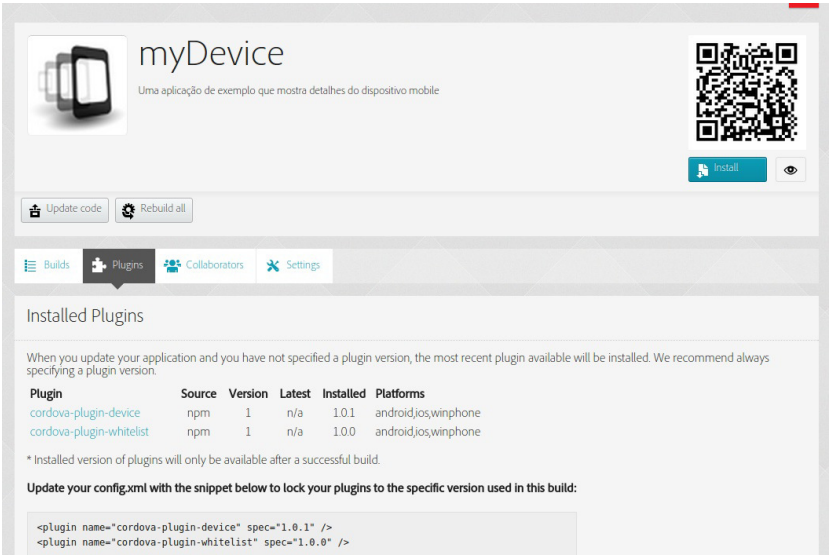


Figura 7.2: Aplicação myDevice no PhoneGap Build

Após confirmar que o plugin está devidamente instalado, podemos usar JavaScript para obter estas informações. Isso será realizado no arquivo `www/js/app.js` , no método `init()` que já está criado, conforme o código a seguir.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
var deviceReadyDeferred = $.Deferred();
var jqmReadyDeferred = $.Deferred();

$(document).on("deviceready", function() {
  deviceReadyDeferred.resolve();
});

$(document).on("mobileinit", function () {
  jqmReadyDeferred.resolve();
});

$.when(deviceReadyDeferred, jqmReadyDeferred).then(init);

function init() {
    $("#cordovaVersion").text(device.cordova);
    $("#model").text(device.model);
    $("#platform").text(device.platform);
    $("#uuid").text(device.uuid);
    $("#version").text(device.version);
}
```

Quando você abrir o arquivo `app.js` pela primeira vez, o método `init()` estará vazio. A alteração está adicionando várias informações obtidas pelo objeto `device`, que é nativo do PhoneGap - graças ao plugin `cordova-plugin-device`. Veja que usamos jQuery para obter o elemento HTML em questão, e o método `text` para sobrescrever o conteúdo deste elemento.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

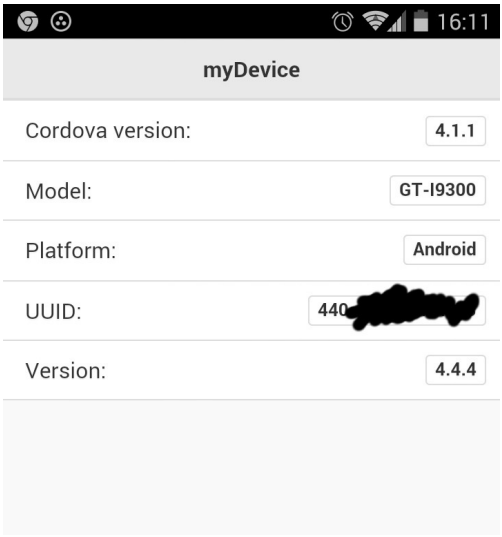


Figura 7.3: Aplicação myDevice executando no Samsung Galaxy S3

7.6 CONCLUSÃO

Neste capítulo, aprendemos a utilizar o PhoneGap para criar aplicações nativas ao dispositivo mobile. Ou seja, a aplicação é instalada no dispositivo, e não acessada através do navegador.

Para compilar a aplicação, usamos um serviço web chamado PhonGap Build, que pode ser usado livremente. Entretanto, em um ambiente de desenvolvimento, o ideal é executar a aplicação diretamente no dispositivo, que é o que abordaremos no próximo capítulo.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

CAPÍTULO 8

INTEGRANDO O ANDROID AO PHONEGAP

No capítulo anterior, criamos a aplicação `myDevice`. Foi possível testá-la por meio do site *PhoneGap Build*, usando o comando `phonegap remote build android`. Neste capítulo, vamos instalar o Android SDK para que possamos executar as aplicações diretamente no dispositivo, sem a necessidade de utilizar o site.

Uma das formas de instalar o Android SDK é pela instalação do Android Studio, uma IDE completa que possui tudo o que é necessário para criar e executar aplicações no dispositivo Android.

8.1 INSTALANDO O ANDROID STUDIO NO WINDOWS

Acesse o seguinte endereço http://developer.android.com/sdk/index.html, e clique no botão `Download Android Studio for Windows`. Aceite a licença e clique no botão `Download Android Studio for Linux`.

Após baixar o instalador, execute-o para iniciar o processo de instalação. Caso necessário, faça a instalação do `Java`

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Development Kit 7 . A verificação do JDK e o link para a instalação é realizada pelo próprio instalador do Android Studio . Na instalação, use a configuração padrão, conforme as figuras a seguir.

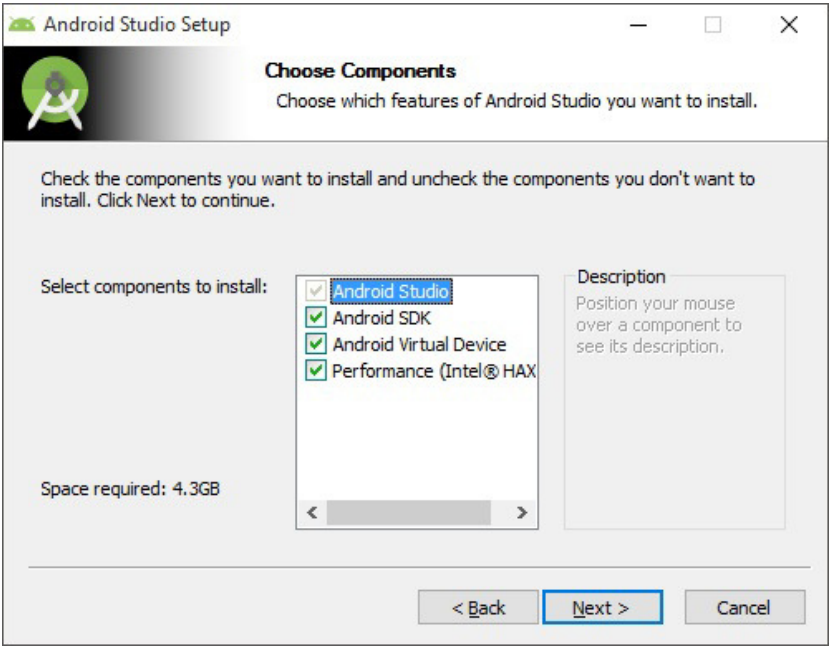


Figura 8.1: Instalação do Android Studio – passo 1

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

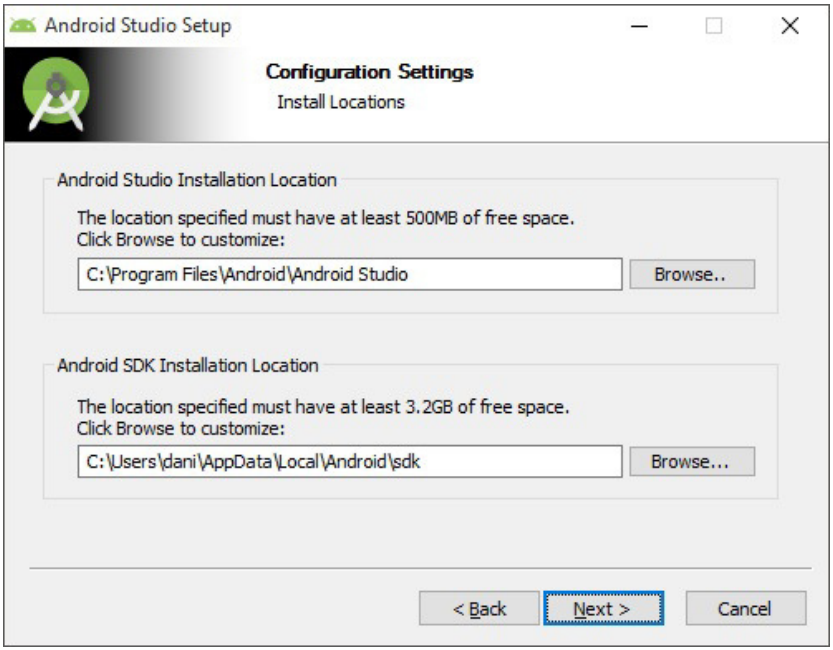


Figura 8.2: Instalação do Android Studio – passo 2

Após a instalação do `Android Studio`, execute-o para que algumas configurações extras sejam realizadas. Na tela `Android Studio Setup Wizard`, escolha o item `Configure`, e depois em `SDK Manager`, conforme a figura a seguir, na qual o Android SDK é exibido.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

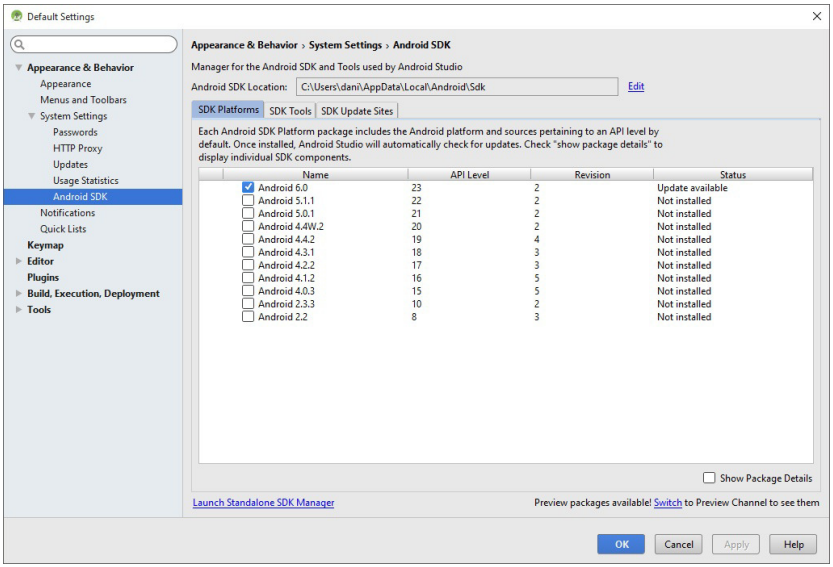


Figura 8.3: Instalação do Android Studio – passo 3

Para o PhoneGap, precisamos do Android 5.0.1 cuja API é a 22, então marque esta opção e clique no botão `Aplly` para que o SDK seja instalado, conforme a figura a seguir.

Figura 8.4: Instalação do SDK Android 5.0.1

8.2 VERIFICANDO A CONEXÃO COM O DISPOSITIVO MOBILE PELO ANDROID DEVICE MONITOR

Após conferir a versão do SDK, juntamente com o diretório de

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

instalação, acesse o diretório onde o SDK está instalado e execute a aplicação `monitor.bat`, localizada no diretório `Android\sdk\tools`. Se o seu dispositivo estiver com o modo de depuração habilitado, e conectado ao computador através de um cabo USB, ao abrir o `monitor`, surgirá uma mensagem no dispositivo requerendo permissão de acesso para que ele possa ser visível ao `monitor`, conforme a figura a seguir.

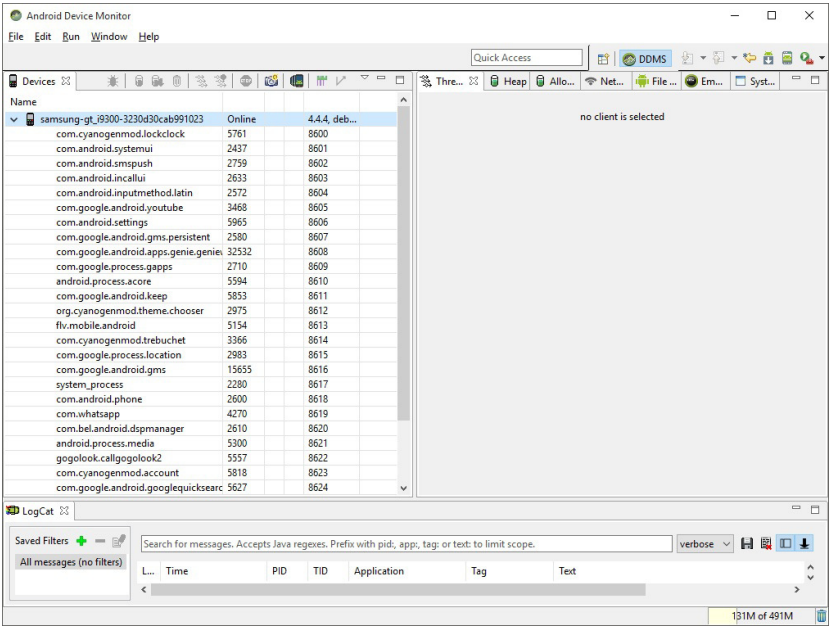


Figura 8.5: Dispositivo sendo exibido no monitor

> Se você não utiliza Linux, pule para a seção *Executando a aplicação Android diretamente no dispositivo*.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

8.3 INSTALANDO O ANDROID STUDIO NO LINUX

Acesse o seguinte endereço http://developer.android.com/sdk/index.html, e clique no botão `Download Android Studio` . Aceite a licença e clique no botão `Download Android Studio for Linux` .

Após o download, descompacte o arquivo e execute o arquivo `android-studio/bin/studio.sh` , conforme o comando a seguir.

```
$ unzip android-studio-ide-141.2422023-linux.zip
$ cd android-studio/bin
android-studio/bin $  ./studio.sh &
```

Após executar o Android Studio, surge um assistente de instalação, semelhante à figura:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

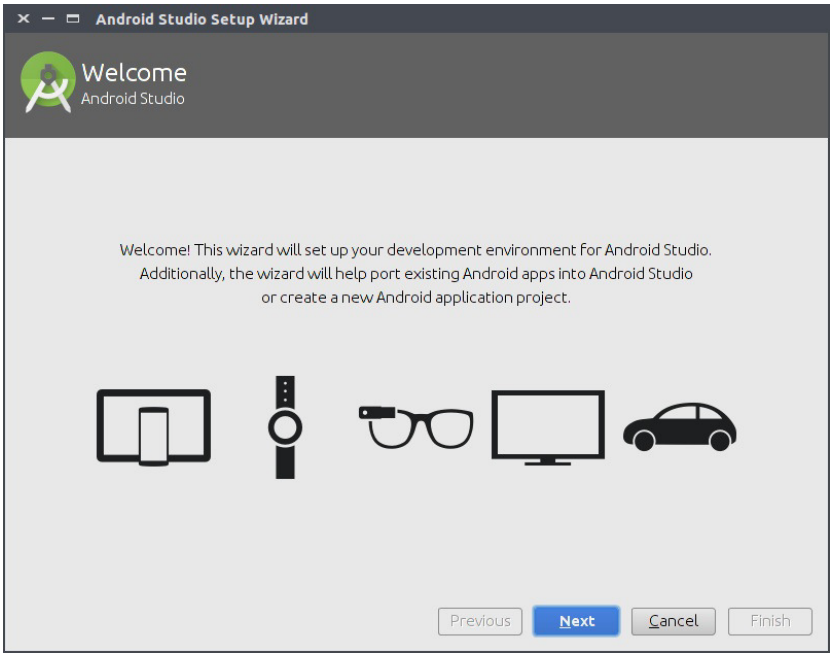


Figura 8.6: Android Studio

Clique no botão `Next`, escolha a versão `Standard`, depois clique em `Next` até a finalização da instalação. O SDK do Android será instalado em `~/android/sdk`. A palavra `SDK` vem de *Software Development Kit*, e é o kit de desenvolvimento de software para Android que o PhoneGap precisa para poder exibir a sua aplicação no seu dispositivo, ou em um dispositivo emulado.

Após a instalação, é apresentado um novo assistente para que se possa criar um novo projeto em Android. Neste caso, o projeto é nativo da linguagem Android, e não será abordado nesta obra. Nosso interesse neste momento é conferir se o SDK foi devidamente instalado. Para isso, clique no item *Configure*, e depois em `SDK Manager`. Deverá surgir uma tela semelhante à

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

figura a seguir.

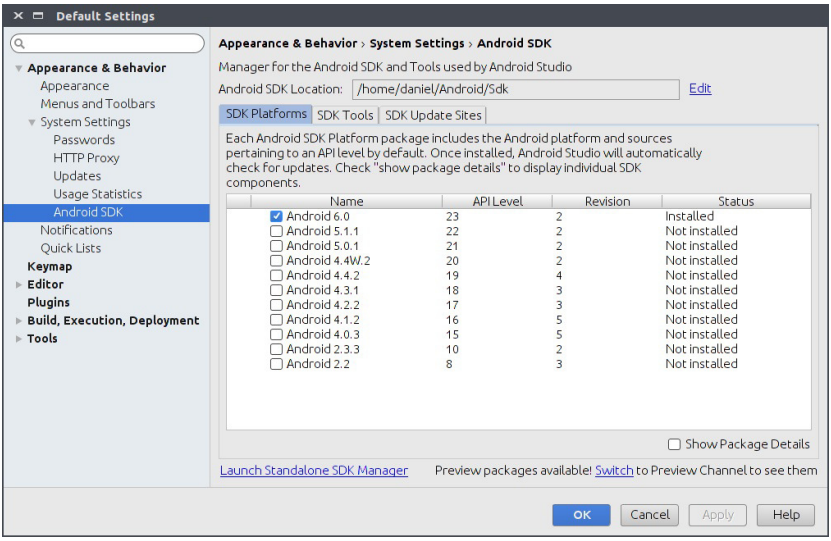


Figura 8.7: SDK Manager

8.4 VERIFICANDO A CONEXÃO COM O DISPOSITIVO MOBILE PELO ANDROID DEVICE MONITOR NO LINUX

Após conferir a versão do SDK, juntamente com o diretório de instalação, acesse o diretório onde o SDK está instalado e execute a aplicação `monitor` . Certifique-se de estar com o dispositivo mobile conectado via USB e que ele permita depuração USB em suas configurações.

```
$ cd ~/Android/sdk/
$ ./tools/monitor
```

Após abrir o monitor e permitir a depuração pelo dispositivo, surge a confirmação de que ele está conectado, semelhante à figura

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

a seguir.

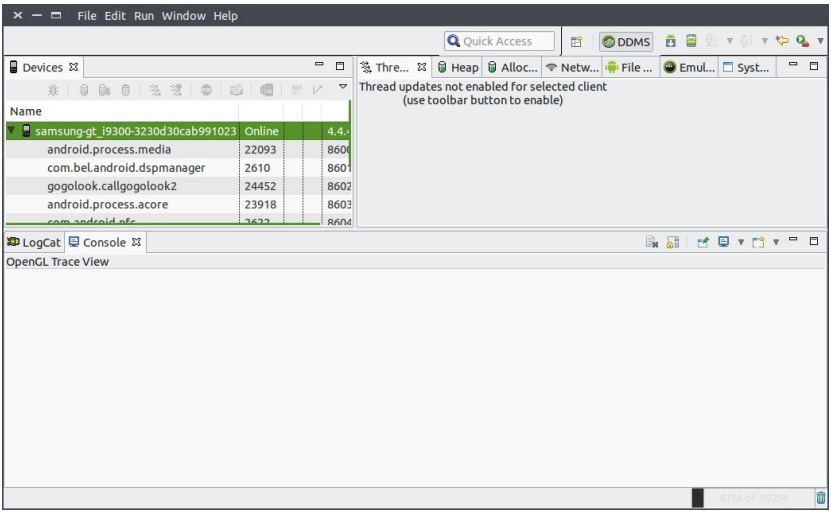


Figura 8.8: SDK Manager

8.5 COMPILANDO A APLICAÇÃO PHONEGAP PARA ANDROID NO LINUX

O comando para criar uma aplicação em Android é exibido a seguir. Ele vai gerar um arquivo com a extensão `.apk`, que poderá ser executado no dispositivo.

Abra um novo terminal, navegue até o diretório onde a aplicação `myDevice` está criada, e digite:

```
$ phonegap build android
```

Como o SDK foi instalado recentemente, um erro surgirá com o texto `Failed to find 'ANDROID_HOME' environment variable.`. Se este erro ocorrer, deve-se adicionar manualmente o Android SDK no *path* do sistema.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

No Linux, abra o arquivo `~/.bashrc` e adicione a seguinte configuração em seu final:

```
export ANDROID_HOME=~/Android/Sdk
```

Esta configuração vai adicionar a variável `ANDROID_HOME` no sistema, apontando para `~/Android/Sdk`, que é o SDK instalado pelo Android Studio. Após salvar o arquivo `.bashrc`, abra um novo terminal, navegue até o diretório onde a aplicação `myDevice` está criada, e tente novamente:

```
$ phonegap build android
```

Após executar novamente o comando, pode surgir o seguinte erro:

```
[Error: Please install Android target: "android-22".

Hint: Open the SDK manager by running:
~/Android/Sdk/tools/android

You will require:
1. "SDK Platform" for android-22
2. "Android SDK Platform-tools (latest)
3. "Android SDK Build-tools" (latest)]
```

Este erro informa que deve ser instalada uma versão do Android diferente da encontrada. O erro sugere abrir o Android SDK Manager e instalar o `SDK Platform for android-22`. Para abrir o SDK Manager, execute o seguinte comando:

```
$ cd ~/Android/Sdk/tools
$ ./android
```

E então, selecione o item `SDK Platform` cuja API é 22, conforme a figura:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

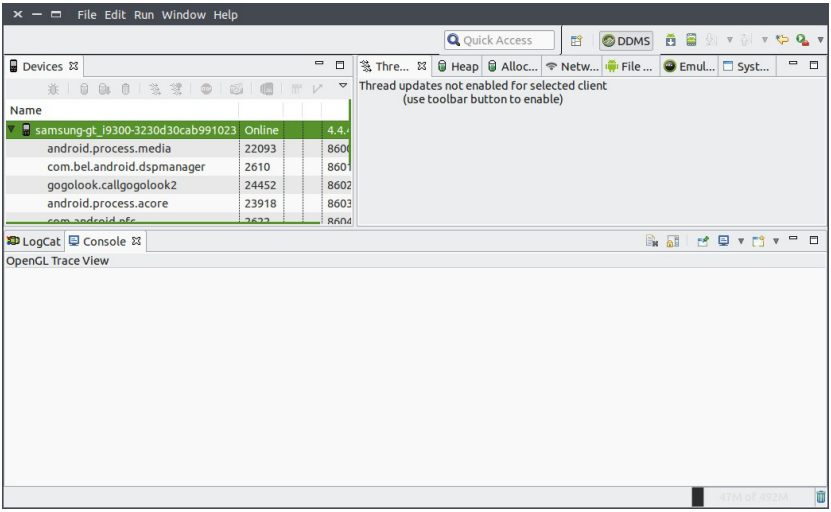


Figura 8.9: SDK Manager com SDK Platform 22

Após selecionar o SDK, clique no botão `Install Packages` , aceite as licenças relacionadas e clique em `Install` .

Após essa instalação, acesse o diretório onde a aplicação `myDevice` está e, para criarmos uma build "limpa", apague o diretório `myDevice\platforms\android` . Depois, execute o seguinte comando:

```
$ phonegap build android
[phonegap] executing 'cordova platform add --save android'...
[phonegap] completed 'cordova platform add --save android'
[phonegap] executing 'cordova build android'...
[phonegap] completed 'cordova build android'
```

8.6 EXECUTANDO A APLICAÇÃO ANDROID DIRETAMENTE NO DISPOSITIVO

Isso criará o build para o dispositivo Android e, como nosso

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

interesse é executar a aplicação Android `myDevice` diretamente no dispositivo, podemos utilizar o seguinte comando:

```
$ phonegap run android
```

A aplicação `myDevice` deverá ser exibida no dispositivo Android. Caso ocorra algum erro, use a opção `--verbose` para que a mensagem de erro seja exibida. Certifique-se de desinstalar a aplicação `myDevice` no dispositivo, caso ela tenha sido instalada pelo site PhoneGap Build.

Se a aplicação não funcionar no dispositivo, crie uma nova aplicação pelo comando `phonegap create HelloWorld` , e depois execute-a através do comando `phonegap run android` . Caso funcione, existe algo errado no `myDevice` , cujo parâmetro `--verbose` mostrará.

8.7 TESTANDO NO NAVEGADOR

Para testar a aplicação no navegador Chrome, execute primeiro o seguinte comando.

```
& phonegap build android
```

Este comando vai gerar uma compilação do PhoneGap para o Android. Depois disso, deve-se copiar todo o conteúdo do diretório `platforms/android/platform_www` para `www` , e deve-se alterar o arquivo `www/index.html` incluindo a seguinte tag HTML após o `<title>` .

```
  <meta http-equiv="Content-Security-Policy" content="default-src
 *; style-src 'self' 'unsafe-inline'; script-src 'self' 'unsafe-i
nline' 'unsafe-eval'">
```

Por este procedimento, pode-se testar uma aplicação

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

PhoneGap diretamente no navegador.

8.8 CONCLUSÃO

Neste capítulo, aprendemos a executar a aplicação PhoneGap diretamente no dispositivo, através de um cabo USB. Desta forma, podemos testar a aplicação mobile de forma mais rápida, sem a necessidade de utilizar o site do PhoneGap Build.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

CAPÍTULO 9

PLUGINS DO PHONEGAP

Neste capítulo, veremos algumas formas de acessar os recursos do dispositivo mobile, como câmera, banco de dados, acelerômetro etc.

9.1 MANIPULANDO A CÂMERA DO DISPOSITIVO MOBILE

Se usarmos apenas o jQuery Mobile no navegador web do dispositivo, essa funcionalidade não poderá ser utilizada. É possível acionar a câmera do dispositivo através de plugins do PhoneGap, que veremos a seguir.

Criando o projeto myCamera

Para criar o projeto pelo PhoneGap, use o seguinte comando:

```
$ phonegap create myCamera --id "com.example.myCamera" --name "my
Camera" --template jquery-mobile-starter
```

Neste exemplo, criamos a aplicação `myCamera` , incluindo o ID do aplicativo, que é `com.example.myCamera` . Em um caso real, o ID é um conjunto de palavras que definem uma identificação única ao seu aplicativo, por isso costuma-se usar o domínio de sua empresa em ordem reversa. Por exemplo, suponha que seu site de

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

desenvolvimento seja `www.fulanodev.com`, então o ID da sua aplicação será `com.fulanodev.nomedaapp`.

Após a criação do projeto, pode-se executá-lo diretamente no dispositivo pelo comando `phonegap run android`, ou utilizar o `Phonegap Builder` pelo comando `phonegap remote build android`.

Adicionando o plugin ao projeto

Com o projeto criado, é preciso adicionar o plugin para prover acesso à câmera. Isso é realizado por meio do seguinte comando:

```
$ phonegap cordova plugin add cordova-plugin-camera
Fetching plugin "cordova-plugin-camera" via npm
Installing "cordova-plugin-camera" for android
```

É necessário configurar as permissões de acesso que o programa `myCamera` terá para cada tipo de dispositivo (Android, iOS, Windows Phone). Isso é realizado no arquivo de configuração específico de cada plataforma, na pasta `myCamera/platforms`.

No caso do Android, o arquivo é o `myCamera/platforms/android/AndroidMainfest.xml`. Nele temos um XML com várias configurações, sendo uma delas `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />`, que permitirá salvar um arquivo no dispositivo. Esta configuração é necessária para que possamos utilizar a câmera.

Criando a interface

Neste programa, teremos um botão que vai acionar a câmera do dispositivo. Após tirar a foto, ela é exibida abaixo do botão.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Alteramos o `<body>` do arquivo `myCamera/www/index.html` para:

```
<div data-role="header">
    <h1>myCamera</h1>
  </div>
  <div data-role="main" class="ui-content">
    <input type="button" value="Tirar Foto" onClick="getPhoto()">
    <image id="cameraImage"  style="width:100px;height:100px;"/>
  </div>
</div>
```

Nesta página, criamos um botão que, quando clicado, executa o método `getPhoto()` . Abaixo do botão, usamos o `<image>` para exibir a imagem que será capturada pela câmera.

Acessando a câmera do dispositivo

O método `getPhoto` é responsável por acessar a câmera, e ele está localizado no arquivo `myCamera/www/js/app.js` . Adicione-o logo abaixo do método `init()` , conforme o código a seguir.

```
function getPhoto(){
    navigator.camera.getPicture(onSuccess, onFail, { quality: 50,
    destinationType: Camera.DestinationType.DATA_URL
    });
    function onSuccess(imageData) {
        var image = document.getElementById('cameraImage');
        image.src = "" + imageData;
    }
    function onFail(message) {
        alert('Failed because: ' + message);
    }
}
```

O método `getPhoto()` usa a API do PhoneGap (pois instalamos o plugin `cordova-plugin-camera`) através do método `navigator.camera.getPicture` . Este tem 3 parâmetros, sendo o primeira o método que será chamado assim que a foto for

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

tirada com sucesso, o segundo o método `onFail` (chamado caso a câmera retorne com algum erro), e o terceiro configurações que podem ser repassadas ao plugin.

Nessas configurações temos o `destinationType` , que mostra o destino da imagem que a câmera tirou. Neste caso, temos `DATA_UTL` que indica uma string codificada em base64, que pode ser lida pelo `<image>` . Existem diversas configurações neste ponto, e uma consulta a API em https://github.com/apache/cordova-plugin-camera possui mais informações.

9.2 BATERIA

Para verificar o nível da bateria do dispositivo, usamos o evento `batterystatus` que fica disponível após adicionarmos o plugin `cordova-plugin-battery-status` . Para exemplificar este processo, crie o projeto `myBattery` , de acordo com o seguinte comando.

```
$ phonegap create myBattery --id "com.example.myBattery" --name "
mybattery" --template jquery-mobile-starter
```

Após criar o projeto, devemos adicionar o plugin `cordova-plugin-battery-status` pelo seguinte comando.

```
$ cd mybattery
$ phonegap cordova plugin add cordova-plugin-battery
```

Após criar o projeto e adicionar o plugin, vamos alterar o arquivo `www\index.html` para incluir um título e adicionar o texto que vai exibir o status da sua bateria.

```
<!DOCTYPE html>
<html>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
<head>
  <meta charset="utf-8">
  <title>Page Title</title>
  <meta name="viewport" content="width=device-width, initial-scal
e=1">
  <link rel="stylesheet" href="lib/jquery.mobile-1.4.5.min.css" /

  <script src="lib/jquery-2.1.1.min.js"></script>
  <script src="js/app.js"></script>
  <script src="lib/jquery.mobile-1.4.5.min.js"></script>
</head>
<body>
<div data-role="page">
   <div data-role="header">
    <h1>Bateria</h1>
  </div>
  <div data-role="main" class="ui-content">
    <p>Sua bateria está <span id='bateria'>??</span>% carregada</
>
  </div>
</div>
<script src="cordova.js"></script>
</body>
</html>
```

Neste HTML, criamos na página um `<span>` com o id `bateria`, que será alterado pelo JavaScript, no arquivo `app.js` exibido a seguir.

```
var deviceReadyDeferred = $.Deferred();
var jqmReadyDeferred = $.Deferred();

$(document).on("deviceready", function() {
  deviceReadyDeferred.resolve();
});

$(document).on("mobileinit", function () {
  jqmReadyDeferred.resolve();
});

$.when(deviceReadyDeferred, jqmReadyDeferred).then(init);

function init() {
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
    window.addEventListener("batterystatus", onBatteryStatus, fal
se);
}

function onBatteryStatus(info) {
    $("#bateria").text(info.level);
}
```

No método `init()`, adicionamos por meio do `window.event` um callback para o evento `batterystatus`, no qual o método `onBatteryStatus` será chamado. Nele usamos jQuery para pegar o `<span id='bateria'>` e atualizar o seu texto. A propriedade `info.level` mostra o nível da bateria, que é a porcentagem que o dispositivo está carregado.

O resultado deste código é exibido na figura:

Figura 9.1: Bateria do dispositivo

9.3 ACELERÔMETRO

O acelerômetro é uma funcionalidade que informa a posição

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

`x`, `y` e `z` do dispositivo. É muito usado em jogos e está disponível para o PhoneGap através do plugin `cordova-plugin-device-motion`. Crie um novo projeto pelo seguinte comando:

```
$ phonegap create myMotion --id "com.example.myMotion" --name "my
motion" --template jquery-mobile-starter
```

Adicione o plugin por meio do comando:

```
$ phonegap cordova plugin add cordova-plugin-device-motion
```

Após adicionar o plugin, precisamos alterar o arquivo `www/index.html` para adicionar algumas informações sobre o acelerômetro. A página HTML pode ser configurada da seguinte forma:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Acelerometro</title>
  <meta name="viewport" content="width=device-width, initial-scal
e=1">
  <link rel="stylesheet" href="lib/jquery.mobile-1.4.5.min.css" />

  <script src="lib/jquery-2.1.1.min.js"></script>
  <script src="js/app.js"></script>
  <script src="lib/jquery.mobile-1.4.5.min.js"></script>
</head>

<body>

<div data-role="page">
  <div data-role="header">
    <h1>Acelerômetro</h1>
  </div>

  <div data-role="main" class="ui-content">
    <ul data-role="listview">
        <li>x: <span id="coord_x" class="ui-li-count"></span></li>

        <li>y: <span id="coord_y" class="ui-li-count"></span></li>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
        <li>z: <span id="coord_z" class="ui-li-count"></span></li>

    </ul>
  </div>
</div>
</div>

<script src="cordova.js"></script>
</body>
</html>
```

Neste HTML, criamos uma página com as três coordenadas que o acelerômetro pode mapear. Alguns dispositivos não exibem a coordenada `z`.

Para exibir os dados do acelerômetro, temos dois métodos principais: `getCurrentAcceleration`, que exibe as coordenadas no momento em que o método foi chamado; e `watchAcceleration`, que obtém as coordenadas em um intervalo de tempo determinado.

Para adicionarmos esta funcionalidade, altere o arquivo `www/js/app.js` para o seguinte código:

```
var deviceReadyDeferred = $.Deferred();
var jqmReadyDeferred = $.Deferred();

$(document).on("deviceready", function() {
  deviceReadyDeferred.resolve();
});

$(document).on("mobileinit", function () {
  jqmReadyDeferred.resolve();
});

$.when(deviceReadyDeferred, jqmReadyDeferred).then(init);

function init() {
    var options = { frequency: 200 }; //.2 segundos
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
    navigator.accelerometer.watchAcceleration(accelerometerSucces
s, accelerometerError,options);
}

function accelerometerSuccess(acceleration) {
     $("#coord_x").text(acceleration.x);
     $("#coord_y").text(acceleration.y);
     $("#coord_y").text(acceleration.z);
}

function accelerometerError() {
    alert('onError!');
}
```

Criamos o método `init` pelo método `navigator.accelerometer.watchAcceleration`, que vai executar o método `accelerometerSuccess` a cada 200 milissegundos – o que foi definido na variável `options`.

No método `accelerometerSuccess`, usamos um pouco de jQuery para selecionar os elementos correspondentes às coordenadas, e utilizamos a variável `acceleration` para obter estes valores.

9.4 CAIXAS DE DIÁLOGO

Apesar do jQuery Mobile possuir este recurso, o PhoneGap possui o plugin `cordova-plugin-dialogs`, que habilita quatro tipos de caixa de diálogo que serão nativas ao dispositivo mobile. Com os comandos a seguir, criamos o projeto `myDialog` e adicionamos o plugin `cordova-plugin-dialogs`.

```
$ phonegap create myDialog --id "com.example.myDialog" --name "my
Dialog" --template jquery-mobile-starter
$ phonegap cordova plugin add cordova-plugin-dialogs
```

Após criar o projeto, crie 4 botões em `www/index.html`,

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

conforme o código:

```
<div data-role="page" id="pageone">
  <div data-role="header">
    <h1>Dialogs</h1>
  </div>
  <div data-role="main" class="ui-content">
        <button onClick='btnAlertClick()' class="ui-btn">Alert</b
utton>
        <button onClick='btnConfirmClick()' class="ui-btn">Confir
</button>
        <button onClick='btnPromptClick()' class="ui-btn">Prompt<
button>
        <button onClick='btnBeepClick()' class="ui-btn">Beep</but
ton>
  </div>
</div>
```

O primeiro botão é o `Alert`, que exibe uma mensagem na tela. No método `btnAlertClick()`, adicionado no arquivo `www/js/app.js`, podemos usar a seguinte configuração:

```
function btnAlertClick(){
    navigator.notification.alert("Mensagem", onButtonClick, "Títu
lo", "Botão");
}

function onButtonClick(){
    // executado após clicar no botão
}
```

O resultado do `navigator.notification.alert` é exibido a seguir.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

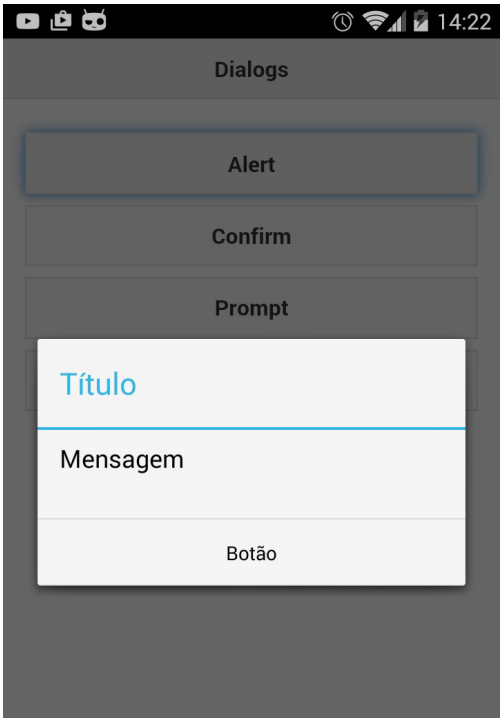


Figura 9.2: Mensagem alert nativa do dispositivo Android

A segunda caixa de diálogo é o `confirm`, que exibe uma mensagem de confirmação, conforme o código:

```
function btnConfirmClick(){
    navigator.notification.confirm(
    'Deseja excluir isso?',
     onConfirm,
    'Registro XYZ',
    ['Sim','Não']);
}

function onConfirm(buttonIndex) {
    alert('Você escolheu ' + buttonIndex);
}
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Outra caixa de diálogo é o `prompt`, no qual é possível adicionar um texto de entrada na caixa de diálogo. O código a seguir mostra como criar uma caixa de diálogo no tipo `prompt`.

```
function btnPromptClick(){
    navigator.notification.prompt(
        "Mensagem",
        onPromptCallback,
        "Título",
        ['Ok','Sair'],
       'Texto Padrão');
}

function onPromptCallback(results) {
    alert("Botão " + results.buttonIndex + " e texto: " + results
.input1);
}
```

O último comando disponível do plugin `cordova-plugin-dialogs` é o `beep`, que emite um *beep* no dispositivo – o mesmo sinal sonoro de uma notificação.

```
navigator.notification.beep(1);
```

9.5 GEOLOCATION

É possível utilizar o GPS do dispositivo para obter informações de localização, tais como latitude e altitude. O plugin que trabalha com *geoclocation* é o `cordova-plugin-geolocation`. Para criar o projeto e adicionar o plugin, execute os comandos a seguir.

```
$ phonegap create myGeolocation --id "com.example.myGeolocation"
--name "myGeolocetion" --template jquery-mobile-starter
$ phonegap cordova plugin add cordova-plugin-geolocation
```

No arquivo `www/index.html`, adicionamos as informações referente ao GPS, como latitude e longitude.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
<div data-role="page" id="pageone">
  <div data-role="header">
    <h1>Geolocation</h1>
  </div>
  <div data-role="main" class="ui-content">
    <ul data-role="listview">
        <li>Latitude: <span id="latitude" class="ui-li-count"></s
pan></li>
        <li>Longitude: <span id="longitude" class="ui-li-count"><
span></li>
        <li>Altitude: <span id="altitude" class="ui-li-count"></s
pan></li>
        <li>Altitude Accuracy: <span id="altitudeAccuracy" class=
"ui-li-count"></span></li>
        <li>Heading: <span id="heading" class="ui-li-count"></spa
></li>
        <li>Speed: <span id="speed" class="ui-li-count"></span></
li>
        <li>Timestamp: <span id="timestamp" class="ui-li-count"><
span></li>
    </ul>
  </div>
</div>
```

Estas informações são adicionadas a uma lista do jQuery Mobile, pela tag `<ul data-role="listview">`. Para exibir o valor de cada item, usa-se a tag `<span class="ui-li-count">`.

No arquivo `www/js/app.js`, adicionamos o método `navigator.geolocation.getCurrentPosition` que obterá as informações de GPS do dispositivo.

```
function init() {
    navigator.geolocation.getCurrentPosition(onGeolocationSuccess
,onGeolocationError);
}

function onGeolocationSuccess(position){
    $("#latitude").text(position.coords.latitude);
    $("#longitude").text(position.coords.longitude);
    $("#altitude").text(position.coords.altitude);
    $("#accuracy").text(position.coords.accuracy);
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
    $("#altitudeAccuracy").text(position.coords.altitudeAccuracy)
;
    $("#heading").text(position.coords.heading);
    $("#speed").text(position.coords.speed);
    $("#timestamp").text(position.coords.timestamp);
}

function onGeolocationError(error) {
    alert('code: '    + error.code    + '\n' +
          'message: ' + error.message + '\n');
}
```

Neste código, usamos o método `getCurrentPosition` no `init()` da aplicação, que é executada quando todo o código JavaScript está carregado. O método `onGeolocationSuccess` usa a variável `position` para obter os valores das coordenadas, como por exemplo, `position.coords.latitude` obtém a latitude do dispositivo.

9.6 INAPPBROWSER

Este plugin abre uma URL no navegador web padrão do dispositivo. Para instalá-lo, use o seguinte comando.

```
$ cordova plugin add cordova-plugin-inappbrowser
```

O código a seguir exibe um exemplo de utilização do `InAppBrowser`.

```
document.addEventListener("deviceready", onDeviceReady, false);
function onDeviceReady() {
    window.open = cordova.InAppBrowser.open(url, target, options)
;
}
```

Neste caso, o parâmetro `url` define a página que será aberta no navegador. O atributo `target` pode assumir três valores:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

- `_self` – A URL será aberta na própria aplicação;
- `_blank` – A URL será aberta em uma nova janela, mas na mesma instância do InAppBrowser;
- `_system` – A URL é aberta no navegador nativo do dispositivo mobile.

Em `options`, temos diversas configurações que podem ser usadas para personalização da página a ser aberta, como por exemplo, `hidden`, que determina se a página deve ser aberta e exibida ao usuário. Estas configurações podem ser consultadas em https://www.npmjs.com/package/cordova-plugin-inappbrowser.

Para abrir URLs de outros domínios, é preciso configurar um plugin que já vem instalado nativamente, chamado de *whitelist*. Para isso, adicione a URL que deseja acessar no arquivo `config.xml` do projeto. Suponha que a sua aplicação deseja abrir uma URL para o site `google.com`, então deve-se adicionar a seguinte entrada no arquivo `config.xml`:

```
<allow-navigation href="http://google.com/*" />
```

9.7 CONEXÃO

É possível verificar se o dispositivo está conectado em alguma rede wifi ou pacote de dados, através do plugin `cordova-plugin-network-information`. Para instalá-lo, use o seguinte comando.

```
$ cordova plugin add cordova-plugin-network-information
```

Após instalar o plugin, é possível obter as informações sobre a conexão através da propriedade `navigator.connection`, que pode assumir os seguintes valores:

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

- `Connection.UNKNOWN`
- `Connection.ETHERNET`
- `Connection.WIFI`
- `Connection.CELL_2G`
- `Connection.CELL_3G`
- `Connection.CELL_4G`
- `Connection.CELL`
- `Connection.NONE`

A função a seguir exibe um exemplo do uso do `Connection`.

```
function checkConnection() {
    var networkState = navigator.connection.type;
    var states = {};
    states[Connection.UNKNOWN]  = 'Unknown connection';
    states[Connection.ETHERNET] = 'Ethernet connection';
    states[Connection.WIFI]     = 'WiFi connection';
    states[Connection.CELL_2G]  = 'Cell 2G connection';
    states[Connection.CELL_3G]  = 'Cell 3G connection';
    states[Connection.CELL_4G]  = 'Cell 4G connection';
    states[Connection.CELL]     = 'Cell generic connection';
    states[Connection.NONE]     = 'No network connection';
    alert('Connection type: ' + states[networkState]);
}

function init(){
checkConnection();
}
```

Neste código, o método `init()` chamará o método `checkConnection`, que usa o objeto `navigator.connection.type` para obter o tipo de conexão atual do dispositivo. O array `states` armazena estes tipos, que podem ser as mais variadas fontes, como uma conexão WiFi ou conexão 4G. No final deste método, usamos o comando `alert` juntamente com o array `states` para exibir a conexão.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

9.8 VIBRATION

Este plugin fará o dispositivo vibrar. Para instalá-lo, execute o seguinte comando.

```
$ cordova plugin add cordova-plugin-vibration
```

Após a instalação do plugin, pode-se ativá-lo por meio do seguinte comando:

```
function init(){
  navigator.vibrate(time)
}
```

`time` é o tempo de duração em milissegundos. Ou seja, `navigator.vibrate(3000)` fará o dispositivo vibrar por 3 segundos.

9.9 LISTA DE CONTATOS

É possível obter a lista de contatos do dispositivo pelo plugin `cordova-plugin-contacts`, que pode ser instalado pelo comando a seguir:

```
$ phonegap cordova plugin add org.apache.cordova.contacts
```

Os contatos são obtidos através do método `navigator.contacts.find`, e estão sempre associados a algum tipo de filtro.

Obtendo todos os contatos

Para obter todos os contatos do dispositivo, é necessário criar um filtro vazio, pelo método `ContactFindOptions`, conforme o exemplo a seguir.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
var options = new ContactFindOptions();
options.filter = "";
var filter = ["displayName", "addresses"];
navigator.contacts.find(filter, onSuccess, onError, options);

function onSuccess(contacts) {
  alert(contacts.length + " contatos encontrados");
}

function onError(contactError) {
    alert('onError!');
};
```

Neste código, perceba que o campo `filter` configura quais as informações serão retornadas, como o nome que é exibido na tela e os endereços que estão associados ao contato. A lista exibe os tipos de informação que cada contato pode possuir.

- `id` – O identificador global do contato no telefone;
- `displayName` – O nome do contato;
- `name` – Um objeto contendo todo o nome do contato, como nome principal, sobrenome, nome do meio;
- `nickname` – Um apelido que o contato pode possuir;
- `phoneNumbers` – Um array com todos os telefones associados ao contato;
- `emails` – Um array com todos os e-mails do contato;
- `addresses` – Um array com todos os endereços do contato;
- `ims` – Um array com todos os *Instant Messengers* do contato, como o Jabber, Skype, WhatsApp etc.;
- `organizations` – Um array com todas as organizações que o contato pode ter;
- `birthday` – A data de aniversário do contato;
- `note` – Uma nota que o contato pode ter;

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

- `photos` – Um array com as fotos do usuário;
- `categories` – Um array com todas as categorias a que o contato possa estar associado;
- `urls` – Um array com as páginas web a que o contato possa estar associado.

Utilizando filtros

Caso deseje fazer pesquisas no contato, o campo `options.field` deverá ser informado, conforme o exemplo a seguir.

```
// Obtém todos os contatos que tenham "Bob" em qualquer um dos ca
mpos
var options      = new ContactFindOptions();
options.filter   = "Bob";
options.multiple = true;
options.hasPhoneNumber = true;
var fields       = [navigator.contacts.fieldType.displayName, nav
igator.contacts.fieldType.name];
navigator.contacts.find(fields, onSuccess, onError, options);
```

Criando um contato

Para criar o contato, usa-se o método `contact.save`, de acordo com o exemplo seguinte.

```
function onSuccess(contact) {
    alert("Save Success");
};

function onError(contactError) {
    alert("Error = " + contactError.code);
};

// cria o contato
var contact = navigator.contacts.create();
contact.displayName = "Plumber";
contact.nickname = "Plumber";
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
// adiciona alguns campos
var name = new ContactName();
name.givenName = "Jane";
name.familyName = "Doe";
contact.name = name;

// salva o contato no dispositivo
contact.save(onSuccess,onError);
```

Para adicionar números de telefone ao contato durante a sua criação, usa-se o método `ContactField` , de acordo com o exemplo.

```
var contact = navigator.contacts.create();
var phoneNumbers = [];
phoneNumbers[0] = new ContactField('work', '212-555-1234', false)
;
phoneNumbers[1] = new ContactField('mobile', '917-555-5432', true
);
phoneNumbers[2] = new ContactField('home', '203-555-7890', false)
;
contact.phoneNumbers = phoneNumbers;
contact.save();
```

9.10 CONCLUSÃO

Neste capítulo, podemos adicionar ao jQuery Mobile as funcionalidades de acesso nativo ao dispositivo, graças ao framework PhoneGap. Vimos também que, com o PhoneGap, o jQuery Mobile não é mais acessado pelo navegador do dispositivo, pois as páginas HTML/JavaScript tornam-se uma aplicação nativa ao mobile, podendo ser instaladas e executadas pelo menu do aparelho.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Capítulo 10

PERSISTINDO DADOS COM JQUERY MOBILE E PHONEGAP

No exemplo que apresentamos em jQuery Mobile e PHP, usamos a linguagem de servidor PHP para persistir os dados no banco de dados MySQL. Esta aplicação, ao ser executada em um navegador mobile, necessita de conexão com a internet.

O uso do PhoneGap + jQuery Mobile tem duas principais vantagens em relação ao uso do jQuery Mobile no navegador:

- O PhoneGap provê acesso nativo ao dispositivo mobile, isto é, por meio dos plugins que abordamos no capítulo anterior, podemos acessar diversas funcionalidades do dispositivo mobile, como câmera, acelerômetro, GPS, eventos etc.
- O PhoneGap provê acesso offline a sua aplicação, já que o banco de dados pode estar localizado no próprio dispositivo.

Neste capítulo, estaremos focados no desenvolvimento na persistência de dados de uma aplicação HTML/JavaScript. Quando

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

criamos uma aplicação neste nível, onde não é preciso utilizar PHP e MySQL para persistir dados, podemos usar o PhoneGap para gerar uma aplicação web que seja 100% offline. Ou seja, temos uma aplicação jQuery Mobile que funciona nativamente no dispositivo, persiste dados e não necessita de internet.

Existem diversas implementações relativas à persistência de dados presentes no HTML 5. Dentre elas, podemos destacar duas.

LocalStorage

É a mais simples e usada para persistir dados com base em um conjunto de chave/valor. Não existem tabelas, transações ou operações mais complexas. Um exemplo simples deste processo é exibido a seguir:

```
window.localStorage.setItem('usuario', 'daniel');
```

Neste exemplo, usamos o objeto `window`, que é a instância do documento HTML, e a propriedade `localStorage`, que é a referência a API do local *Storage*. Nesta API, temos o método `setItem` que possui duas propriedades, sendo a primeira a chave do registro e a segunda o seu valor.

Para obter um registro que foi previamente criado, usamos o método `getItem("chave")`. Para remover um item, usamos `removeItem("chave")` e, finalmente, podemos usar `document.localStorage.clear()` para remover todo o conjunto de dados que foram armazenados.

Pode-se usar o `localStorage` para armazenar pequenas informações, por exemplo, se o usuário deixou selecionado o item "lembrar" de um formulário de login.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

IndexedDB

Esta opção é uma das mais utilizadas atualmente, e possui uma boa aceitação relativa aos dispositivos mobile. Atente-se à versão do Android, pois o IndexedDB é compatível somente com dispositivos na versão 4.2 ou superior. O IndexedDB é mantido pela W3C, que o escolheu como API de persistência e descontinuou o *WebSQL*, sobre o qual ainda podem-se encontrar referências de uso em tutoriais da internet.

Vamos criar um pequeno exemplo de manipulação com IndexedDB. Primeiro, criamos o documento HTML e inserimos uma verificação para analisar se o `indexedDB` está funcionando.

```
<html>
<head>
    <title>IndexedDB Ex 01</title>
</head>
<body>
<script type="text/javascript">
document.addEventListener("DOMContentLoaded", function(event) {
    if (!window.indexedDB)
            window.alert("Banco de dados incompatível");
    else
            console.log("Banco de dados compatível");
});
</script>
</body>
</html>
```

Abra este arquivo no navegador e verifique se a mensagem `Banco de dados compatível` surge no console do Google Chrome (`F12`).

Com o objeto `indexedDB` funcional no navegador, podemos usá-lo para criar o banco de dados. Isso é realizado pelo método `open` , conforme o exemplo a seguir.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
var request = window.indexedDB.open("banco_de_dados");
```

O método `open` vai abrir o banco de dados `banco_de_dados`. Como este banco não existe ainda no dispositivo, o evento `onupgradeneeded` será disparado e podemos, através dele, criar uma tabela.

```
var request = window.indexedDB.open("banco_de_dados");
request.onupgradeneeded = function() {
    console.log("onupgradeneeded");
     var db = request.result;
     var store = db.createObjectStore("todo", {keyPath: "id",auto
Increment : true});
     var title = store.createIndex("title", "title", {unique: tru
});
     var status = store.createIndex("status", "status");
     //Adiciona uma tarefa de teste
      store.put({title: "Tarefa teste", status: 1});
}
request.onsuccess = function() {
  console.log("onsuccess");
  db = request.result;
}
```

O evento `onupgradeneeded` é disparado somente se o banco de dados `banco_de_dados` não existir. Desta forma, podemos adicionar tabelas por meio do comando `createObjectStore`, como fizemos no código anterior, onde criamos a tabela `todo` com a chave primária `id`. No `indexedDB`, chamamos uma tabela de *store*. Após a criação da tabela, criamos dois campos utilizando o método `createIndex`. No final do evento `onupgradeneeded`, adicionamos um registro à tabela, usando JSON.

O evento `onsuccess` sempre é executado quando o banco de dados é aberto ou criado com sucesso. A partir da variável `db`, podemos inserir, recuperar ou apagar registros, conforme os exemplos a seguir.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
var tx = db.transaction("todo", "readonly");
var store = tx.objectStore("todo");
var response = store.get(1);
response.onsuccess = function () {
    console.log(response.result);
};
```

Neste exemplo, abrimos uma transação na tabela `todo` em modo de somente leitura. Então, usamos o método `objectStore` para obter a tabela em si (tecnicamente, é retornado um objeto que corresponde ao store `todo`, mas suponha que seja uma tabela). O método `store.get` retornará um objeto que corresponde à chave-primária da tabela. Como passamos o valor `1`, que é a chave, o primeiro objeto é retornado, aquele mesmo cujo o campo título é *"Tarefa teste"*, conforme a figura a seguir.

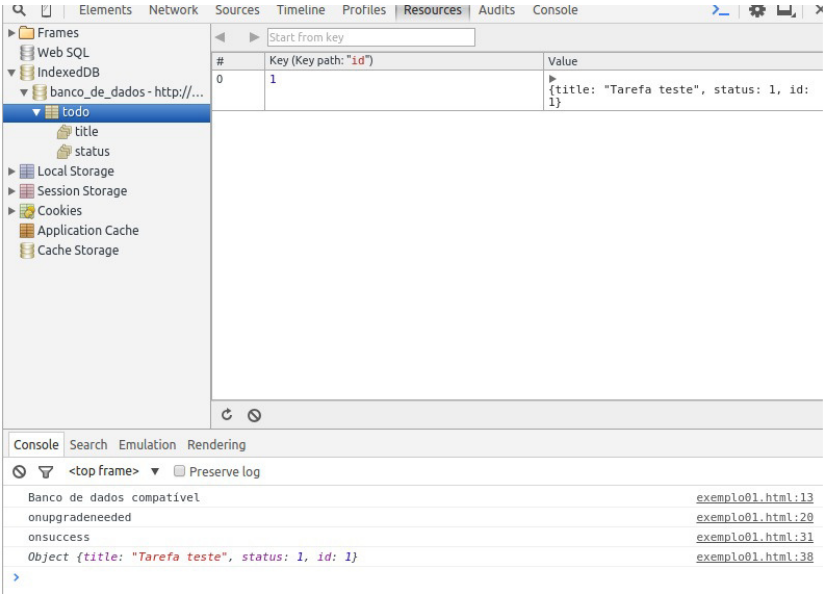


Figura 10.1: Exemplo com IndexedDB

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

10.1 CRIANDO A APLICAÇÃO TASKER

Vamos criar uma aplicação para gerenciar tarefas, chamada de *tasker*. Usaremos PhoneGap e IndexedDB. Primeiro, crie a aplicação pelo seguinte comando:

```
$ phonegap create tasker --id "com.example.tasker" --name "tasker
 --template jquery-mobile-starter
```

Após criar a aplicação, devem-se inicialmente criar as telas do gerenciador de tarefas. A tela inicial vai exibir as tarefas ativas, que possuem o `status=0`. Existirá um botão para adicionar uma tarefa, e nesta outra tela o usuário poderá adicionar uma nova tarefa.

Inicialmente, o arquivo `www/index.html` contém o seguinte código:

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
  <meta name="viewport" content="width=device-width, initial-scal
e=1">
  <link rel="stylesheet" href="lib/jquery.mobile-1.4.5.min.css" />

  <script src="lib/jquery-2.1.1.min.js"></script>
  <script src="js/app.js"></script>
  <script src="lib/jquery.mobile-1.4.5.min.js"></script>
</head>
<body>
<div data-role="page">
</div>
<script src="cordova.js"></script>
</body>
</html>
```

Adicione o `charset` antes da tag `<title>`, conforme o código a seguir.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Page Title</title>
  ..... continua.....
```

Isso é necessário para que a página HTML possa exibir os acentos corretamente. Na tag <body> , criaremos duas páginas. A primeira é usada para exibir as tarefas.

```
  <div id="tasksPage" data-role="page">
    <div data-role="header">
        <h1> Tarefas </h1>
        <a href="#addTask" class="ui-btn ui-btn-right">Adicionar<
a>
    </div>
    <div data-role="content">
        <ul id="listTasks" data-role="listview">
            <li> Sem tarefas... </li>
        </ul>
    </div>
</div>
```

Nesta página, criamos no cabeçalho um botão para adicionar tarefas, e criamos no conteúdo da página uma lista que vai conter as tarefas criadas. Por enquanto, exibimos apenas uma mensagem de que não existem tarefas.

A página que exibe o formulário para cadastrar uma tarefa é exibida a seguir.

```
<div id="addTask" data-role="page">
  <div data-role="header">
    <a href="#tasksPage" class="ui-btn" data-rel="back">Voltar</a

    <h1> Adicionar Tarefa </h1>
  </div>
  <div data-role="content">
     <div class="ui-field-contain">
       <label for="title">Título:</label>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
      <input type="text" name="title" id="title" value="">
    </div>
    <button onclick=" ">Enviar</button>
</div>
</div>
```

Nesta página, temos um botão "voltar" que retornará a página `tasksPage` . Também temos um campo de texto para que o usuário possa escrever o título da tarefa e depois clicar no botão `enviar` .

Até o presente momento, a página `index.html` possui o seguinte código HTML:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta http-equiv="Content-Security-Policy" content="default-src
 *; style-src 'self' 'unsafe-inline'; script-src 'self' 'unsafe-i
nline' 'unsafe-eval'">
  <title>Page Title</title>
  <meta name="viewport" content="width=device-width, initial-scal
e=1">
  <link rel="stylesheet" href="lib/jquery.mobile-1.4.5.min.css" />

  <script src="lib/jquery-2.1.1.min.js"></script>
  <script src="js/app.js"></script>
  <script src="lib/jquery.mobile-1.4.5.min.js"></script>
</head>

<body>

<div id="tasksPage" data-role="page">
    <div data-role="header">
        <h1> Tarefas </h1>
        <a href="#addTask" class="ui-btn ui-btn-right">Adicionar</
a>
    </div>
    <div data-role="content">
        <ul id="listTasks" data-role="listview">
            <li> Sem tarefas... </li>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
        </ul>
    </div>
</div>

<div id="addTask" data-role="page">
  <div data-role="header">
    <a  id="_tasksPage" href="#tasksPage" class="ui-btn" data-rel
"back">Voltar</a>
    <h1> Adicionar Tarefa </h1>
  </div>
  <div data-role="content">
     <div class="ui-field-contain">
       <label for="title">Título:</label>
       <input type="text" name="title" id="title" value="">
     </div>
    <button onclick="btnAddTask_click()">Enviar</button>
 </div>
</div>

<script src="cordova.js"></script>
</body>
</html>
```

10.2 INICIALIZANDO O BANCO DE DADOS

Como foi visto anteriormente, quando inicializamos o banco de dados com o IndexedDB, o método `onupgradeneeded` é executado, no qual podemos criar a tabela de tarefas, conforme o código a seguir do arquivo `www/js/app.js` .

```
var deviceReadyDeferred = $.Deferred();
var jqmReadyDeferred = $.Deferred();
var db = null;

$(document).on("deviceready", function() {
  deviceReadyDeferred.resolve();
});

$(document).on("mobileinit", function () {
  jqmReadyDeferred.resolve();
});
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
$.when(deviceReadyDeferred, jqmReadyDeferred).then(init);

function init() {
  dbInit();
}

function dbInit(){
    if (!window.indexedDB)
        window.alert("Banco de dados incompatível");
    else{
        var request = window.indexedDB.open("taskerdb");
        request.onupgradeneeded = function() {
            console.log("newDB");
            var newDB = request.result;
            var store = newDB.createObjectStore("todo", {keyPath:
"id",autoIncrement : true});
            var title = store.createIndex("title", "title", {uniq
ue: true});
            var status = store.createIndex("status", "status");
     }
     request.onsuccess = function (e) {
        console.log("openDB");
        db = e.target.result;
      };
    }
}

function btnAddTask_click(){
    console.log("btnAddTask_click");
}
```

No método `init` do PhoneGap, criamos o método `dbInit()` que tem como responsabilidade iniciar o banco de dados. Neste método, usamos `!window.indexedDB` para verificar se o `IndexedDB` está operacional, e então usamos o método `window.indexedDB.open` para abrir o banco de dados `taskerdb`. Se o banco de dados não está criado, o método `onupgradeneeded` é executado e podemos adicionar a tabela `todo`, com os campos `id`, `title` e `status`.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Veja também que criamos uma variável global chamada `db`, que é uma referência ao banco de dados em toda a aplicação.

10.3 CRIANDO UMA TAREFA

Na tela para criar uma nova tarefa, temos o campo de texto `title` e um botão. O botão chama o método `btnAddTask_click()`, que usa *IndexedDB* para salvar a tarefa no banco de dados.

```
function btnAddTask_click(){
    var tx = db.transaction(["todo"], "readwrite");
    var store = tx.objectStore("todo");
    var task = {title:$("#title").val(),status:0}
    var request = store.add(task);
    request.onsuccess = function (e) {
        console.log("adicionado " + JSON.stringify(task));
        $("#title").val("");
        $("#_tasksPage").trigger("click");
        refreshTasks();
    };
    request.onerror = function (e) {
        console.log("Erro:", e.target.error.name);
    };
}
```

Nesta função, iniciamos uma transação de acordo com a documentação do IndexedDB. A transação indica o seu nome e o modo de abertura (neste caso, `readwrite`) para leitura e escrita.

A variável `store` obtém a tabela `todo` pelo método `objectStore`. Após obter a tabela, criamos o objeto `task`, que é um JSON, formado pelo título que usamos jQuery para obter o campo `title`, e informamos o `status` da tarefa como `0`, como se ela ainda não estivesse concluída.

Após incluir o registro por meio do método `store.add`, o

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

evento `onsuccess` é executado e nele podemos voltar à tela principal da aplicação, a qual exibe todas as tarefas cadastradas. Para que a nova tarefa apareça nesta tela, chamamos o método `refreshTasks` que exibe todas as tarefas abertas.

10.4 VISUALIZANDO AS TAREFAS

Quando o usuário adiciona uma tarefa, o método `refreshTasks` é executado. Este é exibido a seguir:

```
function refreshTasks(){
    $("#listTasks").empty();

    var tx = db.transaction(["todo"], "readonly");
    var objectStore = tx.objectStore("todo");
    var index = objectStore.index("status");
    var cursor = index.openCursor(IDBKeyRange.only(0));

    cursor.onsuccess = function () {
        var row = cursor.result;
        if (row) {
            task = row.value;
            $("#listTasks").append("<li data-id='"+task.id+"'>"+t
ask.title+"</li>");
            row.continue();
        }

        $("#listTasks").listview('refresh');

     // "deletar" a tarefa
     $("#listTasks").on("swiperight",">li",function(event){
      var li = $(this);
      var span = li.children();
      var idTask = $(this).attr("data-id");
      $(this).animate({marginLeft: parseInt($(this).css('marginLe
ft'),10) === 0 ? $(this).outerWidth() : 0 }).fadeOut('fast',funct
ion(){li.remove();changeStatus(idTask);});
  });
 };
}
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

O processo para atualizar as tarefas consiste em apagar todos os `<li>` da lista e recriá-los. Usamos `$("#listTasks").empty()` para apagar as linhas, e depois usamos `$("#listTasks").append` para adicioná-las novamente. Para obter somente os registros cujo `status=0`, abrimos o índice `status` e usamos o método `IDBKeyRange.only(0)` para obter somente registros cujo o valor é `0`.

Após adicionar todos os `<li>` novamente, é preciso redesenhar a lista pelo comando `$("#listTasks").listview('refresh');`. No final do método, incluímos uma forma de remover os registros da lista, através do `swiperight`, que já foi abordado em um capítulo anterior. O *Swipe* deslizará a linha para a direita e, no final do movimento, o método `changeStatus(idTask)` será chamado, para que a tarefa em si tenha o status alterado para 1.

10.5 ALTERANDO O STATUS DA TAREFA

No método `refreshTasks`, usamos a animação do `swiperight` para remover a tarefa, que na verdade realiza um update na tabela `todo`, alterando o status da tarefa para `1`. Veja o método descrito:

```
function changeStatus(idTask){
    var tx = db.transaction(["todo"], "readwrite");
    var store = tx.objectStore("todo");
    var request = store.get(Number(idTask));
    request.onsuccess = function(e) {
        var todo = e.target.result;
        todo.status = 1;
        store.put(todo);
    }
}
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Nesta função, usamos a chave-primária da tarefa para obter o registro através do `store.get`. Ao obtê-lo, alteramos o status para `1` e atualizamos novamente a tabela por meio do `store.put`.

O arquivo `app.js` completo é exibido a seguir:

```
var deviceReadyDeferred = $.Deferred();
var jqmReadyDeferred = $.Deferred();
var db = null;

$(document).on("deviceready", function() {
  deviceReadyDeferred.resolve();
});

$(document).on("mobileinit", function () {
  jqmReadyDeferred.resolve();
});

$.when(deviceReadyDeferred, jqmReadyDeferred).then(init);

function init() {
  dbInit();
}

function dbInit(){
    if (!window.indexedDB)
        window.alert("Banco de dados incompatível");
    else{
        var request = window.indexedDB.open("taskerdb");
        request.onupgradeneeded = function() {
            console.log("newDB");
            var newDB = request.result;
            var store = newDB.createObjectStore("todo", {keyPath:
"id",autoIncrement : true});
            var title = store.createIndex("title", "title", {uniq
ue: true});
            var status = store.createIndex("status", "status");
        }
        request.onsuccess = function (e) {
            console.log("openDB");
            db = e.target.result;
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
            refreshTasks();
        };
    }
}

function btnAddTask_click(){
    console.log("btnAddTask_click");

    var tx = db.transaction(["todo"], "readwrite");
    var store = tx.objectStore("todo");
    var task = {title:$("#title").val(),status:0}
    var request = store.add(task);

    request.onsuccess = function (e) {
        console.log("adicionado " + JSON.stringify(task));
        $("#title").val("");
        $("#_tasksPage").trigger("click");
        refreshTasks();
    };

    request.onerror = function (e) {
        console.log("Erro:", e.target.error.name);
    };

}


function refreshTasks(){
    console.log("refreshTasks");

    $("#listTasks").empty();

    var tx = db.transaction(["todo"], "readonly");
    var objectStore = tx.objectStore("todo");
    var index = objectStore.index("status");
    var cursor = index.openCursor(IDBKeyRange.only(0));

    cursor.onsuccess = function () {
        var row = cursor.result;
        if (row) {
            task = row.value;
            $("#listTasks").append("<li data-id='"+task.id+"'>"+t
ask.title+"</li>");
            row.continue();
        }
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
        $("#listTasks").listview('refresh');

      // "deletar" a tarefa
      $("#listTasks").on("swiperight",">li",function(event){
       var li = $(this);
       var span = li.children();
       var idTask = $(this).attr("data-id");
       $(this).animate({marginLeft: parseInt($(this).css('marginLe
ft'),10) === 0 ? $(this).outerWidth() : 0 }).fadeOut('fast',funct
ion(){li.remove();changeStatus(idTask);});
  });
 };
}

function changeStatus(idTask){
    console.log("changeStatus: " + idTask);
    var tx = db.transaction(["todo"], "readwrite");
    var store = tx.objectStore("todo");
    var request = store.get(Number(idTask));
    request.onsuccess = function(e) {
        var todo = e.target.result;
        todo.status = 1;
        store.put(todo);
    }
}
```

10.6 CONCLUSÃO

Como podemos ver neste exemplo, é perfeitamente possível manipular dados através do IndexedDB. Também, com o PhoneGap, pode-se criar uma aplicação que esteja instalada no dispositivo mobile, de forma que ela possa funcionar sem acesso à internet.

Quando criamos um site ou aplicação em jQuery Mobile, é possível escolher entre estes dois caminhos. Também pode-se criar uma aplicação em conjunto com uma linguagem de servidor, onde o jQuery Mobile seria executado no navegador do cliente e

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

necessitaria de acesso à internet, ou pode-se criar uma aplicação com PhoneGap, sendo instalada no dispositivo.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Capítulo 11

CONCLUSÃO

Chegamos ao final deste livro. Vamos fazer uma recapitulação sobre o que aprendemos até aqui, como também apontar possíveis caminhos que o leitor pode seguir para ampliar o seu conhecimento em jQuery Mobile.

Nos capítulos iniciais, vimos que a instalação do jQuery Mobile é realizada de diferentes formas, sendo por meio do `npm`, `cdn` ou realizando o download das bibliotecas necessárias. A instalação do jQuery Mobile na página sempre é feita referenciando dois arquivos JavaScript essenciais, o `jquery.XXX` e o `jquery.mobile.XXX`, onde `XXX` é a versão atual da biblioteca. Também é necessário adicionar o arquivo `CSS` responsável em adicionar a folha de estilos à página. O exemplo a seguir ilustra este processo:

```
<!DOCTYPE html>
<html>
<head>
    <title> Página jQuery Mobile </title>
    <meta name="viewport" content="width=device-width, initial-sc
ale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.
4.5/jquery.mobile-1.4.5.min.css" />
<script src="http://code.jquery.com/jquery-1.11.1.min.js"></scrip
>
<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.
4.5.min.js"></script>
```


E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

```
</head>
<body>
    Hello World jQuery Mobile!
</body>
</html>
```

Também vimos um pouco do automatizador de tarefas `gulp`, que pode compactar todos os arquivos JavaScript do projeto em um só, tornando o seu tamanho menor e, consequentemente, melhorando a performance da página. Quando usamos o `gulp`, nossa página HTML pode ser resumida a seguir.

```
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
    <meta name="viewport" content="width=device-width, initial-sc
ale=1">
    <link rel="stylesheet" type="text/css" href="css/style.css">
</head>
<body>
    Hello World jQuery Mobile!
    <script type="text/javascript" src="js/script.js"></script>
</body>
</body>
</html>
```

Neste exemplo, vimos que a chamada `<script>` está imediatamente antes do `</body>`, e esta é a melhor forma de adicionar arquivos JavaScript ao documento HTML.

Após a instalação, conhecemos um pouco mais sobre o jQuery Mobile e concluímos que esta biblioteca trabalhada diretamente com HTML, onde cada componente pode ser representado através de uma `tag` HTML e, na maioria das vezes, pelos atributo `class` e `data-role`. Por exemplo, uma página é definida por `<div data-role="page">`, enquanto uma lista é definida por `<ul data-role="listview">`.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

Mas não somente de HTML o jQuery Mobile é construído. Para dar lógica à aplicação, usamos JavaScript, que é fundamental para acessar recursos do dispositivo, como a câmera, GPS, lista de contatos etc. Estes recursos são acessados pelo framework PhoneGap, que tornam o jQuery Mobile uma aplicação nativa ao dispositivo.

É possível também integrar o jQuery Mobile a linguagens de servidor, como o PHP ou Java, e obter dados destes servidores por meio de Ajax. Você poderá escolher entre usar uma linguagem de servidor para persistir dados, ou então usar o próprio dispositivo, através de uma tecnologia nativa aos navegadores web chamada *IndexedDB*.

Independente da forma como você escolheu trabalhar com jQuery Mobile, existe uma vasta documentação para que você possa aprofundar o seu conhecimento. Nos links a seguir, exibimos alguns sites que podem ser consultados para se obter mais conhecimento, veja:

- http://jquerymobile.com/ – Site oficial do jQuery Mobile, onde existe o link *API Documentation*, onde podemos conhecer mais sobre a API do framework, em sua versão atual.
- http://themeroller.jquerymobile.com/ – Themeroller é um gerenciador de temas para a sua aplicação. Neste site pode-se definir todas as cores e estilos necessários para a sua aplicação, podendo ser baixada e instalada na sua página.
- http://jquerymobile.com/download-builder/ – Você pode escolher quais os componentes serão usados na

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br

sua aplicação através do Download Builder. Ao escolher cada componente, pode-se diminuir o tamanho final do arquivo JavaScript a ser importado no seu projeto.

- http://w3schools.com/jquerymobile/ – A w3schools é um site de aprendizado que contém diversos exemplos úteis no desenvolvimento de aplicações em jQuery Mobile.
- http://phonegap.com/ – Site oficial do PhoneGap, com diversas informações sobre o framework.

Além destes sites, você pode participar do Fórum da Casa do Código, no qual você poderá enviar dúvidas e sugestões ao autor. O endereço é: http://forum.casadocodigo.com.br

Fique à vontade para escrever para o autor, pelo e-mail `danieljfa@gmail.com` , e conheça mais dos seus livros no site http://www.casadocodigo.com.br e no site do autor http://www.danielschmitz.com.br.

E-book gerado especialmente para Caixa Econômica Federal - gedti04@caixa.gov.br