PhoneGap & jQuery Mobile

Lesson 6

Thomas Mak

makzan@42games.net

Source Codes

https://github.com/makzan/PhoneGap-Course-Examples

Today

- Using Form
- Saving data in MongoLab
- Fetching data from MongoLab

Continue from last lesson

- We will continue the Great Limited project from last lesson.
  - We did: Facebook, Twitter, Map, News
  - This time: Contact / Feedback Form

Feedback Form

In this section, we will create a feedback form that collects feedbacks from users and save to database.

Feedback Form

```
<!-- ////////// Feedback Page -->
<article data-role='page' id='feedback'>
  <header data-position='fixed' data-role='header'>
    <h1>Feedback</h1>
    <a data-icon='arrow-l' data-rel='back'>About</a>
  </header>
  <section data-role='content'>
    <p>Your feedback is valuble to us.</p>
    <label for='feedback-name'>Your Name</label>
    <input type='text' id='feedback-name'>
    <label for='feedback-email'>Email</label>
    <input type='text' id='feedback-email'>
    <label for='feedback-message'>Message</label>
    <textarea id='feedback-message'></textarea>
    <input id='feedback-submit' type='submit' value='send'>
  </section>
</article>
```

first, we prepare the HTML of the feedback form.

Feedback Form

```
<!-- ////////// Feedback Page -->
<article data-role='page' id='feedback'>
  <header data-position='fixed' data-role='header'>
    <h1>Feedback</h1>
    <a data-icon='arrow-l' data-rel='back'>About</a>
  </header>
  <section data-role='content'>
    <p>Your feedback is valuble to us.</p>
    <label for='feedback-name'>Your Name</label>
    <input type='text' id='feedback-name'>
    <label for='feedback-email'>Email</label>
    <input type='text' id='feedback-email'>
    <label for='feedback-message'>Message</label>
    <textarea id='feedback-message'></textarea>
    <input id='feedback-submit' type='submit' value='send'>
  </section>
</article>
```

we assign id to each form element because we'll need it in jQuery logic.

Feedback Form

```
<a href='#feedback' data-role='button'>Leave feedback</a>
```

then we link to the feedback page from about page.

Feedback Form

Question: Where do we save the feedback data?

Feedback Form

Question: Where do we save the feedback data?

Possible Answers:

- Send email
- Post to a server to stores data in database
- Directly send to database via web service

Feedback Form

Post to a server to stores data in database.

- We need a middle ware between the client and database.

- Traditionally we have PHP, .NET or Ruby on Rails.

- Recently we can use Node.JS

Feedback Form

Post to a server to stores data in database.

- We need to write our API for querying and updating the data between backend storage and client.

Feedback Form

Directly send to database via web service.

MongoLab

- a web service to serve MongoDB.

- the service provide an API for client to access the database.

Feedback Form

MongoLab – MongoDB Hosting | Cloud Hosted MongoDB

https mongolab.com/databases/great_limited_db#collections

mongolab

products | pricing | blog | support | logout

{ user: "makzan", account: "makzan" }

Home

Database: great_limited_db

To connect using the shell:

```
% mongo ds037087.mongolab.com:37087/great_limited_db -u <user> -p <password>
```

To connect using a driver via the standard URI (what's this?):

```
mongodb://<user>:<password>@ds037087.mongolab.com:37087/great_limited_db
```

mongod v2.0.7

question ?

Collections | Users | Stats | Tools

Collections — Open API view — Add

[None at this time]

System Collections

NAME		
system.indexes		
system.namespaces		
system.users	1	0.09 KB

Connect to your database

Use the connection information on the top of the "Collections" tab of this page with either a:

(1) MongoDB interactive shell

Quick links to MongoDB Client v2.0.2 (64-bit): Mac OS X, Linux, Windows, Solaris 64

For the full and most up-to-date release, visit MongoDB's download page here. Learn about the shell here.

(2) MongoDB driver

To learn about and download driver(s), visit MongoDB's drivers page here.

sign up and create a database in mongolab.com

Feedback Form

MongoLab – MongoDB Hosting | Cloud Hosted MongoDB

https mongolab.com/databases/great_limited_db#collections

mongolab

products | pricing | blog | support | logout

{ user: "makzan", account: "makzan" }

Home

Database: great_limited_db

To connect using the shell:

```
% mongo ds037087.mongolab.com:37087/great_limited_db -u <user> -p <password>
```

To connect using a driver via the standard URI (what's this?):

```
mongodb://<user>:<password>@ds037087.mongolab.com:37087/great_limited_db
```

mongod v2.0.7

question ?

Collections | Use

Add new collection

Collection name*

feedbacks

show advanced options

Cancel | Create

Collections

[None at this time]

System Collections

NAME	DOCUMENTS	SIZE
system.indexes	2	0.17 KB
system.namespaces	5	
system.users	1	

Connect to your database

Use the connection information on the top of the "Collections" tab of this page with either a:

(1) MongoDB interactive shell

Quick links to MongoDB Client v2.0.2 (64-bit): Mac OS X, Linux, Windows, Solaris 64

For the full and most up-to-date release, visit MongoDB's download page here. Learn about the shell here.

(2) MongoDB driver

To learn about and download driver(s), visit MongoDB's drivers page here.

create a collection named **feedbacks**.

Feedback Form

API Key

The resources in your MongoLab account can be accessed via the MongoLab REST API.

In order to use the API, you must code your clients to present an API Key to the server on each API request. The key should be presented using an HTTP query parameter called 'apiKey' as in the following example:

```
https://api.mongolab.com/api/1/databases?apiKey=<your-api-key>
```

Each user of your MongoLab account has a separate API Key. The current key for this user is displayed below. You may regenerate the key for this user at any time.

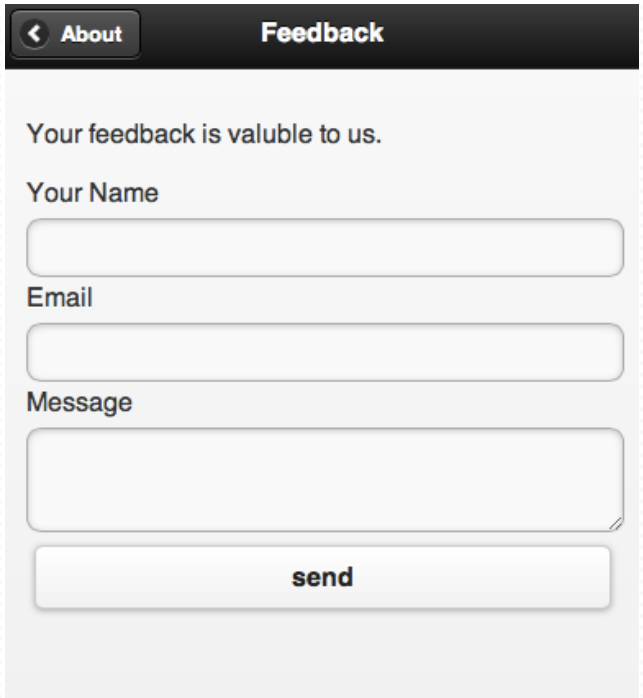
Current key: 502d0 [redacted] 35e

Regenerate API key

create a collection named **feedbacks**.

Feedback Form

```
<!-- ////////// Feedback Page -->
<article data-role='page' id='feedback'>
  <header data-position='fixed' data-role='header'>
    <h1>Feedback</h1>
    <a data-icon='arrow-l' data-rel='back'>About</a>
  </header>
  <section data-role='content'>
    <p>Your feedback is valuable to us.</p>
    <label for='feedback-name'>Your Name</label>
    <input type='text' id='feedback-name'>
    <label for='feedback-email'>Email</label>
    <input type='text' id='feedback-email'>
    <label for='feedback-message'>Message</label>
    <textarea id='feedback-message'></textarea>
    <input id='feedback-submit' type='submit' value='send'>
  </section>
</article>
```



we add a Feedback page to HTML.

Feedback Form

```
<a href='#feedback' data-role='button'>Leave feedback</a>
```

don’t forget to link to the page from somewhere.

Feedback Form

we will have two classes for the Feedback function.

Feedbacks - the model to query and create data entry to MongoLab

FeedbacksView - the view logic, mostly jQuery stuff, to post and show feedbacks.

MV* Pattern

So we are now using **MV*** pattern

Someone may refer this as

MVC, Model-View-Controller

Usually in JavaScript, it is more like a

MVP, Model-View-Presenter

Feedback Form

```
// Feedbacks Module
(function() {
  var Feedbacks = (function(){
    function Feedbacks(){
      // logic here later.
    }
    Feedbacks.prototype.post = function (name, email, message, callback) {
      // logic here later.
    }

    return Feedbacks;

  })();

  // export the Feedbacks to global scope
  if (!this.greatLtd) this.greatLtd = {}
  this.greatLtd.Feedbacks = Feedbacks;

}).call(this);
```

the Feedback module skeleton.

Feedback Form

```
// Feedbacks Module
(function() {
  var Feedbacks = (function(){
    function Feedbacks(){
      // logic here later.
    }
    Feedbacks.prototype.post = function (name, email, message, callback) {
      // logic here later.
    }

    return Feedbacks;

  })();

  // export the Feedbacks to global scope
  if (!this.greatLtd) this.greatLtd = {}
  this.greatLtd.Feedbacks = Feedbacks;

}).call(this);
```

we will put logics in constructor and post method

Feedback Form

```
function Feedbacks(){
  this.url = 'https://api.mongolab.com/api/1/databases/great_limited_db/collections/
feedbacks?apiKey=502d0b59e4b07320d21ab85e'
}
```

the constructor stores the MongoLab service URL

Feedback Form

```
Feedbacks.prototype.post = function (name, email, message, callback) {
  $.ajax({
    url: this.url,
    type: "POST",
    contentType: "application/json",
    crossDomain: true,
    dataType: "json",
    data: JSON.stringify({
      name: name,
      email: email,
      message: message,
    }),
    success: function (data) {
      console.log('saved to mongo, response:', data);
      // call the callback function
      if (callback != undefined)
      {
        callback(data);
      }
    }
  })
}
```

the logic in post method

Feedback Form

```
Feedbacks.prototype.post = function (name, email, message, callback) {
  $.ajax({
    url: this.url,
    type: "POST",
    contentType: "application/json",
    crossDomain: true,
    dataType: "json",
    data: JSON.stringify({
      name: name,
      email: email,
      message: message,
    }),
    success: function (data) {
      console.log('saved to mongo, response:', data);
      // call the callback function
      if (callback != undef
      {
        callback(data);
      }
    }
  })
}
```

the method takes parameters of the Feedback content. And also a callback.

Feedback Form

when we design a method that will work on something asynchronously, we usually provide a *callback* parameter for the caller to implement logic after the operation completed.

Feedback Form

```
Feedbacks.prototype.post = function (name, email, message, callback) {
  $.ajax({
    url: this.url,
    type: "POST",
    contentType: "application/json",
    crossDomain: true,
    dataType: "json",
    data: JSON.stringify({
      name: name,
      email: email,
      message: message,
    }),
    success: function (data) {
      console.log('saved to mongo, response:', data);
      // call the callback function
      if (callback != undefined)
      {
        callback(data);
      }
    }
  })
}
```

mongolab API takes JSON formatted parameter when creating data entry.

Feedback Form

```
Feedbacks.prototype.post = function (name, email, message, callback) {
  $.ajax({
    url: this.url,
    type: "POST",
    contentType: "application/json",
    crossDomain: true,
    dataType: "json",
    data: JSON.stringify({
      name: name,
      email: email,
      message: message,
    }),
    success: function (data) {
      console.log('saved to mongo, response:', data);
      // call the callback function
      if (callback != undefined)
      {
        callback(data);
      }
    }
  })
}
```

after we post the entry to mongolab, we invoke the callback method.

Feedback Form

```
// Feedbacks View Module
(function() {
  var FeedbacksView = (function(){
    // expected data is array of object with name, email, message as key in each object.
    function FeedbacksView(){
    }

    // posting view
    FeedbacksView.prototype.handlePostButton = function() {
      // logic after pressing post button
    }
    return FeedbacksView;
  })();

  // export the FeedbacksView to global scope
  if (!this.greatLtd) this.greatLtd = {}
  this.greatLtd.FeedbacksView = FeedbacksView;
}).call(this);
```

the FeedbacksView skeleton

Feedback Form

```
// posting view
FeedbacksView.prototype.handlePostButton = function() {
  var Feedbacks = greatLtd.Feedbacks;
  $('#feedback-submit').click(function(){
    var name = $('#feedback-name').val();
    var message = $('#feedback-message').val();
    var email = $('#feedback-email').val();
    (new Feedbacks()).post(name, email, message, function(){
      $('#feedback-name').val('');
      $('#feedback-email').val('');
      $('#feedback-message').val('');
      alert('Feedback sent. Thanks.');
    });
  })
}
```

the logic when pressing the post button

Feedback Form

```
// posting view
FeedbacksView.prototype.handlePostButton = function() {
  var Feedbacks = greatLtd.Feedbacks;
  $('#feedback-submit').click(function(){
    var name = $('#feedback-name').val();
    var message = $('#feedback-message').val();
    var email = $('#feedback-email').val();
    (new Feedbacks()).post(name, email, message, function(){
      $('#feedback-name').val('');
      $('#feedback-email').val('');
      $('#feedback-message').val('');
      alert('Feedback sent. Thanks.');
    });
  })
}
```

clear the form on callback.

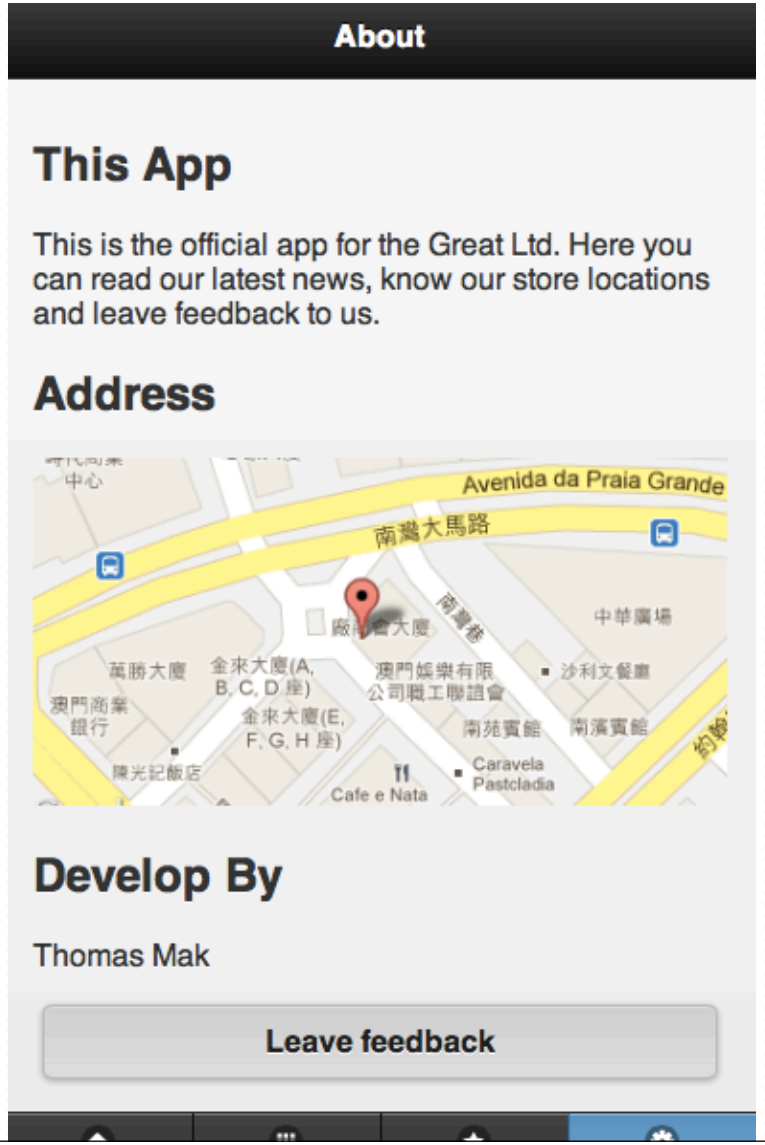
the logic when pressing the post button

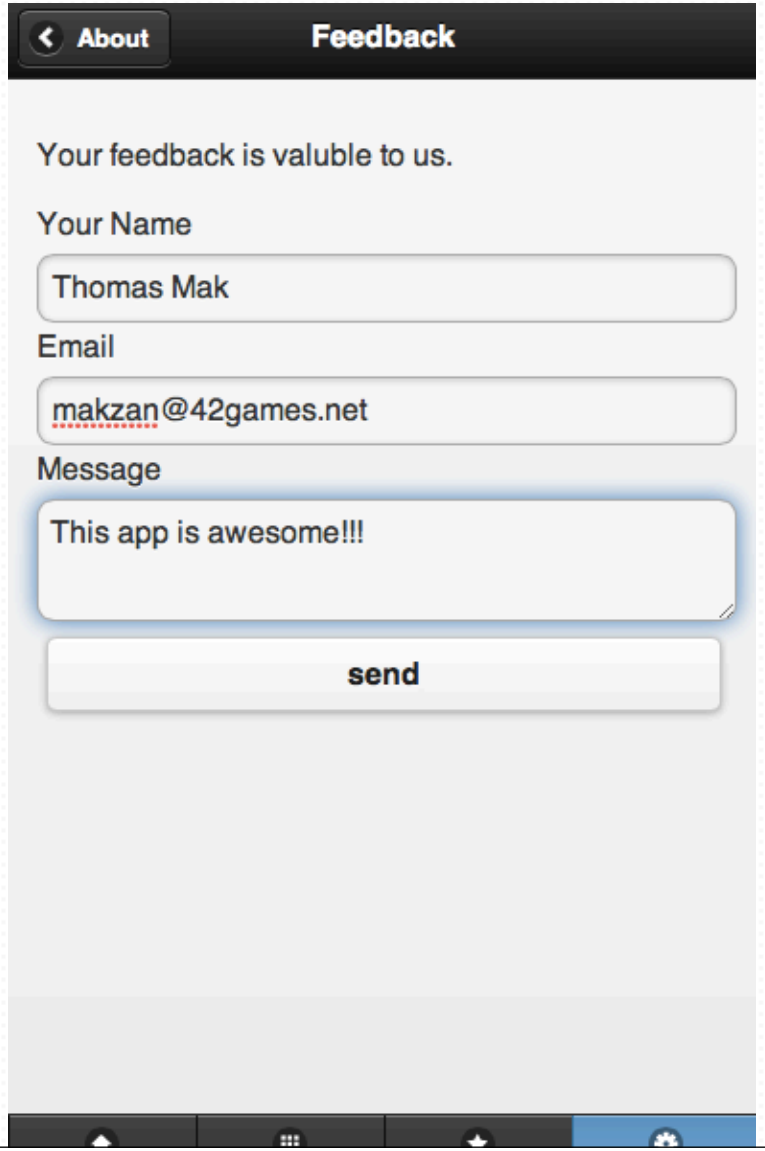
Feedback Form

```
// init the FeedbacksView to handle the feedback submit button.
(new FeedbacksView()).handlePostButton();
```

at last, we init the post button handling inside jQuery ready function.

Feedback Form Result





Feedback Form Result

Home : {db : "great_limited_db"}

Collection: feedbacks (rename)

Documents | Indexes | Stats | Tools

Documents / Objects

Open API view | Delete all | Add

--- Start new search ---

All Documents

Display mode: list table (edit table view)

widen column ⇒

records / page 10

[1 - 9 of 9]


```
{
    "_id": {
        "$oid": "502d0ddbe4b073874ff8ad4c"
    },
    "name": "Hello"
}
```

```
{
    "_id": {
        "$oid": "502d0defe4b073874ff8ad4d"
    },
    "name": "Hello",
    "email": "makzan@gmail.com"
}
```

```
{
```


and we can check if the data is stored into the mongolab.

An issue on jQuery Mobile and PhoneGap 1.9.0

An known issue on jQuery Mobile and PhoneGap 1.9.0 on Android:

space and some characters cannot be input into the input field or text area.

Solution: use PhoneGap version other than 1.9.0

List Feedbacks

In this section, we will create a page to list all the saved feedbacks.

List Feedbacks

```
Feedbacks.prototype.fetch = function(callback) {
  $.ajax({
    url: this.url,
    type: "GET",
    success: function (data) {
      callback(data);
    }
  })
}
```

add a fetch method to our Feedbacks model class.
note that we have callback parameter.

List Feedbacks

```
// listing the data
FeedbacksView.prototype.list = function(data, element) {
  $(element).empty();
  for (var i=0, len=data.length; i<len; i++)
  {
    var feedback = data[i];
    var listItem = "<li><h2>" + feedback.name + "</h2><p>" + feedback.email + "</p><p>" +
feedback.message + "</p>";
    $(element).append(listItem);
  }
  $(element).listview('refresh');
}
```

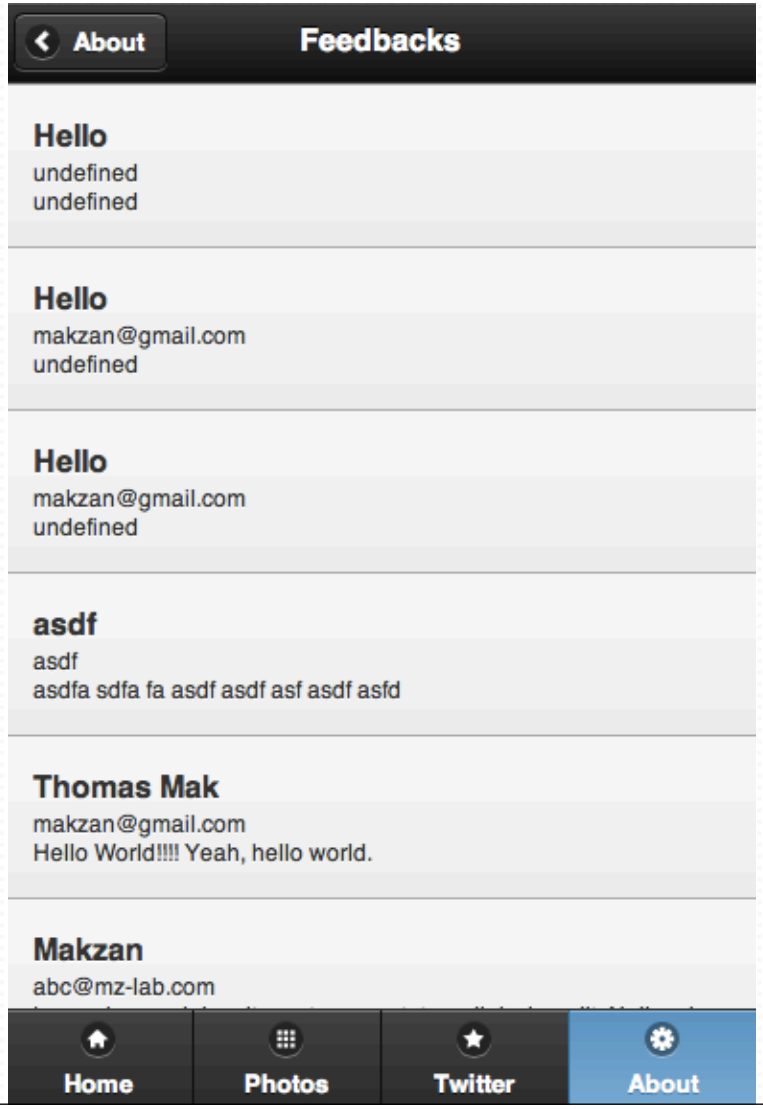
and now we have one more view, add a *list* method to FeedbacksView class to list the feedbacks.

List Feedbacks

```
case 'list-feedback':
  (new Feedbacks()).fetch(function(data){
    (new FeedbacksView()).list(data, '#feedbacks-list');
  });
```

at last, we fetch the feedbacks and display it when the list-feedback page is shown.

List Feedbacks Result



More Form Components

In this section, we will explore several more form components that worth checking in jQuery Mobile.

More Form Components

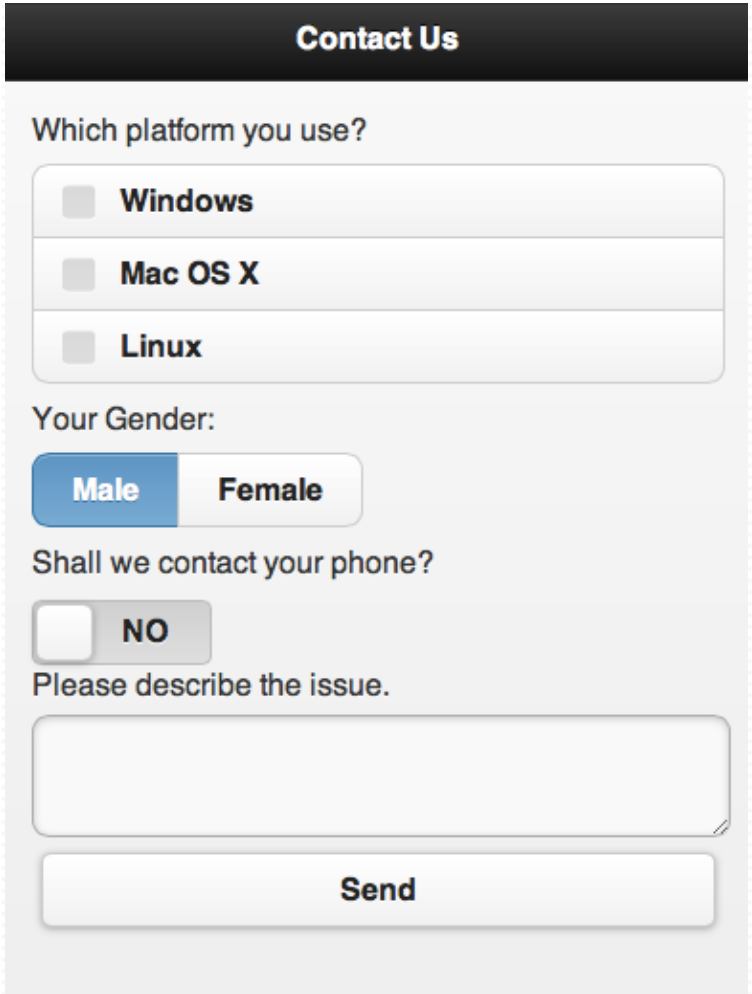
HTML5 introduces several more types:

<input type=‘number’>

<input type=‘email’>

<input type=‘url’>

More Form Components



Most of the form element works by default.

jQuery Mobile provide few more controls.

More Form Components

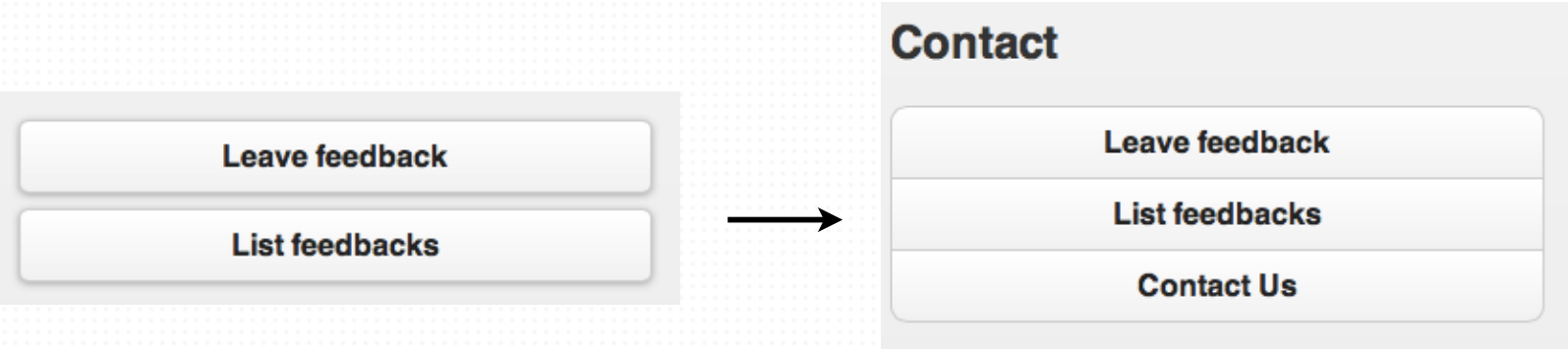
```
<!-- ////////// Contact Us Page -->
<article data-role='page' id='contact-us'>
  <header data-position='fixed' data-role='header'>
    <h1>Contact Us</h1>
  </header>
  <section data-role='content'>

    <!-- more form components here -->

  </section>
</article>
```

let's create a contact page to play around more different form components.

More Form Components



Previously we have two separated buttons.

In mobile, we tend to group buttons into list-like look.

we can do it by putting elements into *controlgroup*

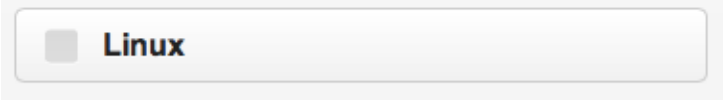
More Form Components

```
<h2>Contact</h2>
  <div data-role='controlgroup'>
    <a href='#feedback' data-role='button'>Leave feedback</a>
    <a href='#list-feedback' data-role='button'>List feedbacks</a>
    <a href='#contact-us' data-role='button'>Contact Us</a>
  </div>
</section>
```

add a link to the contact page and put the links into *controlgroup*.

More Form Components

```
<fieldset data-role="controlgroup">
  <legend>Which platform you use?</legend>
  <input type="checkbox" name="contact-windows" id="contact-windows">
  <label for="contact-windows">Windows</label>
  <input type="checkbox" name="contact-mac" id="contact-mac">
  <label for="contact-mac">Mac OS X</label>
  <input type="checkbox" name="contact-linux" id="contact-linux">
  <label for="contact-linux">Linux</label>
</fieldset>
```



The checkbox works when the label is associated with the input.

More Form Components

```
<fieldset data-role="controlgroup" data-type="horizontal">
  <legend>Your Gender:</legend>
  <input type="radio" name="contact-gender" id="radio-male" value='m' checked="checked">
  <label for="radio-male">Male</label>
  <input type="radio" name="contact-gender" id="radio-female" value='f'>
  <label for="radio-female">Female</label>
</fieldset>
```

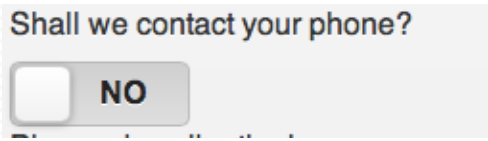
data-type="vertical"

data-type="horizontal"

same as checkbox, the radio works when the label is associated with the input.

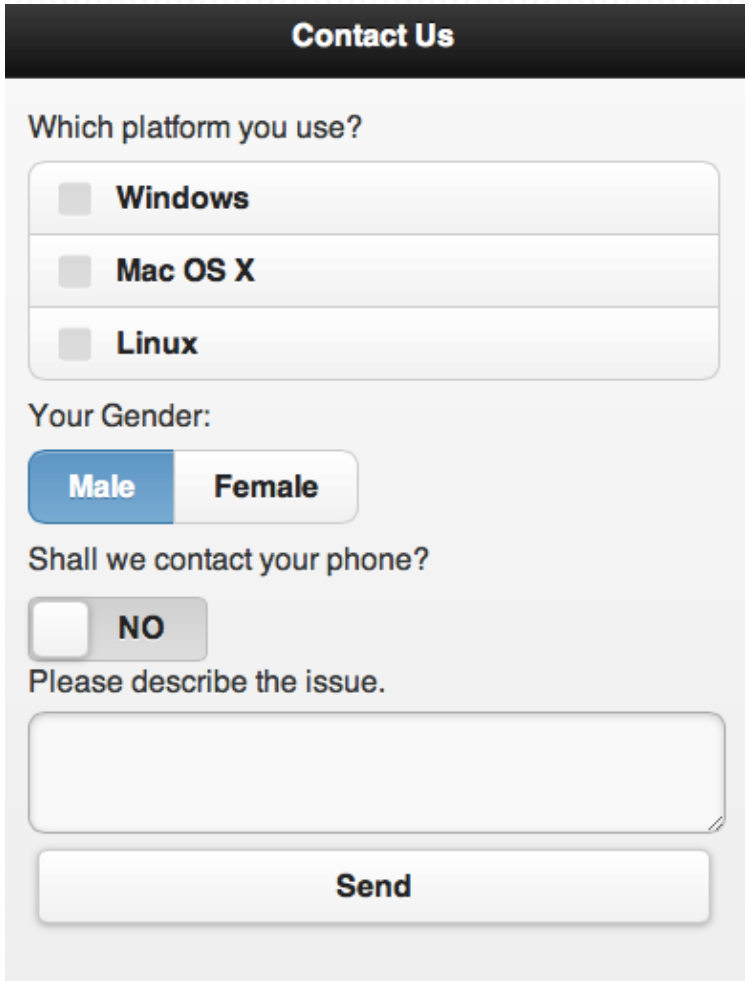
More Form Components

```
<fieldset>
  <label for="contact-phone">Shall we contact your phone?</label>
  <select name="contact-phone" id="contact-phone" data-role="slider">
    <option value="off">NO</option>
    <option value="on">YES</option>
  </select>
</fieldset>
```



alternative to checkbox and radio, we can use slider with `data-role="slider"`

More Form Components



This is the UI that we have in contact page.

More Form Components

For the logic part, we first need to create a *contacts* collection in MongoLab.

More Form Components

```
this.url = 'https://api.mongolab.com/api/1/databases/great_limited_db/collections/
contacts?apiKey=502d0b59e4b07320d21ab85e'
```

time for the Contacts class, it is the same as Feedbacks class except the URL we use.

More Form Components

```
(function(){
  var ContactView = (function(){
    function ContactView() {}

    // posting view
    ContactView.prototype.handlePostButton = function() {
      $('#contact-submit').click(function(){
        var windows = $('#contact-windows').is(':checked');
        var mac = $('#contact-mac').is(':checked');
        var linux = $('#contact-linux').is(':checked');
        var gender = $('input[name=contact-gender]:checked').val();
        var phone = $('#contact-phone').val();
        var message = $('#contact-message').val();
        (new greatLtd.Contact().post(windows, mac, linux, gender, phone, message, function(){
          alert("We received your contact. Thanks.");
          $.mobile.changePage('#about');
        }));
      })
    }

    return ContactView;
  })();

}).call(this);
```

ContactView Class

More Form Components

```
(function(){
  var ContactView = (function(){
    function ContactView() {}

    // posting view
    ContactView.prototype.handlePostButton = function() {
      $('#contact-submit').click(function(){
        var windows = $('#contact-windows').is(':checked');
        var mac = $('#contact-mac').is(':checked');
        var linux = $('#contact-linux').is(':checked');
        var gender = $('input[name=contact-gender]:checked').val();
        var phone = $('#contact-phone').val();
        var message = $('#contact-message').val();
        (new greatLtd.Contact().post(windows, mac, linux, gender, phone, message, function(){
          alert("We received your contact. Thanks.");
          $.mobile.changePage('#about');
        }));
      })
    }

    return ContactView;
  })();

}).call(th
```

for checkbox, we need to use `is(':checked')`

More Form Components

```
(function(){
  var ContactView = (function(){
    function ContactView() {}

    // posting view
    ContactView.prototype.handlePostButton = function() {
      $('#contact-submit').click(function(){
        var windows = $('#contact-windows').is(':checked');
        var mac = $('#contact-mac').is(':checked');
        var linux = $('#contact-linux').is(':checked');
        var gender = $('input[name=contact-gender]:checked').val();
        var phone = $('#contact-phone').val();
        var message = $('#contact-message').val();
        (new greatLtd.Contact().post(windows, mac, linux, gender, phone, message, function(){
          alert("We received your contact. Thanks.");
          $.mobile.changePage('#about');
        }));
      })
    }

    return ContactView;
  })();

}).call(this)
```

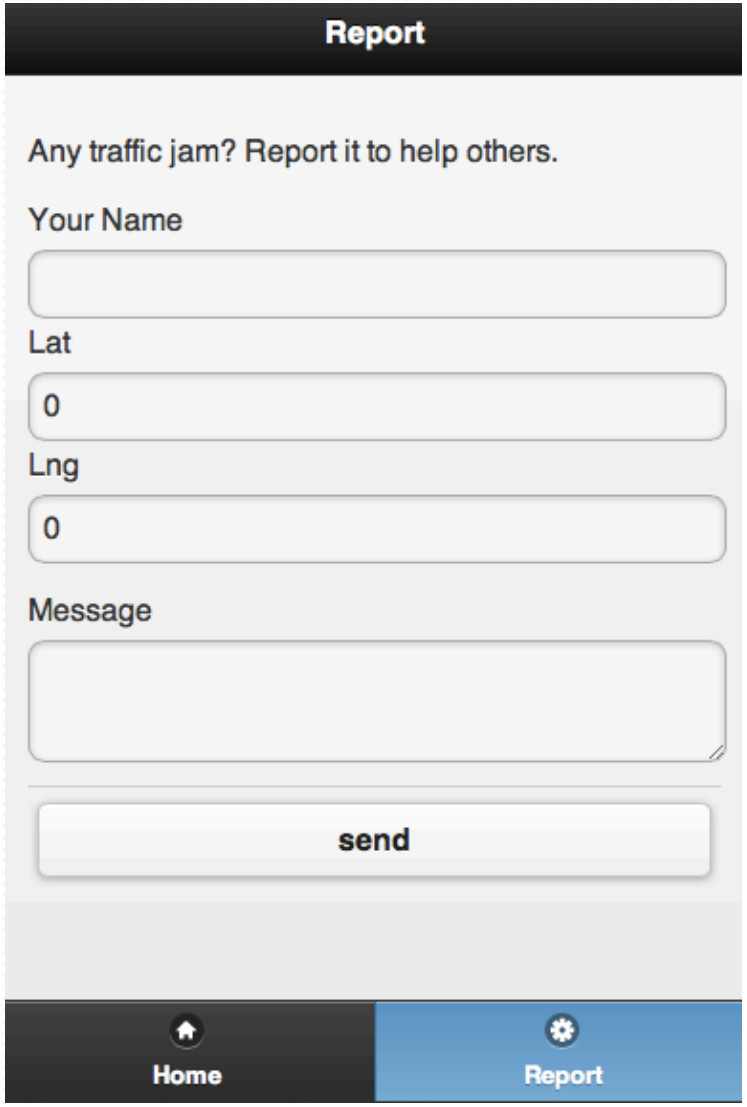
for radio, we get the value of the checked one.

More Form Components


```
{
    "_id": {
        "$oid": "502dbe6ae4b0e3348c741dfe"
    },
    "windows": false,
    "mac": false,
    "linux": false,
    "gender": "m",
    "phone": "off",
    "message": "abcde"
}
```


when testing the app, check the MongoLab to see if we saved the object into the database.

Exercise



Use the techniques in this section to create an app that drivers can report traffic jam of the current location. Others can view and vote up the reports.