Mobile Applications Professional Education Series

Brian Davidson
Research Network Operations Center

WEB FRAMEWORKS
Mobile Applications Professional Education Series

Brian Davidson
Research Network Operations Center

WEB APP MVC
Visiting a Website

• Files are sent from server disk after processing
• Rendered on client
• Javascript runs on client to update the rendering
Model-View-Controller

The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the application, it defines the way objects communicate with each other. Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries.
Model

• Models represent knowledge. A model could be a single object (rather uninteresting), or it could be some structure of objects.

• A model object can have to-one and to-many relationships with other model objects, and so sometimes the model layer of an application effectively is one or more object graphs.

• There should be a one-to-one correspondence between the model and its parts on the one hand, and the represented world as perceived by the owner of the model on the other hand.
View

- A view is a (visual) representation of its model. It would ordinarily highlight certain attributes of the model and suppress others. It is thus acting as a presentation filter.

- A view is attached to its model (or model part) and gets the data necessary for the presentation from the model by asking questions. It may also update the model by sending appropriate messages. All these questions and messages have to be in the terminology of the model, the view will therefore have to know the semantics of the attributes of the model it represents.
Controllers

- A controller is the link between a user and the system. It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen. It provides means for user output by presenting the user with menus or other means of giving commands and data. The controller receives such user output, translates it into the appropriate messages and pass these messages on to one or more of the views.
Model In Web (HTML & JS Variables)
Controller in Web (Javascript & Browser)
MVC in Web (Client)

- The HTML on the client represents the model of information we want to convey to the user.
- The CSS on the client updates the model so it renders it in a pleasing manner.
- The Javascript on the client gets triggered by the user interacting with the HTML (based on CSS positioning) and updates the HTML model.

---

Copyright 2014 Russell J Clark
## Model in Web (Database)

<table>
<thead>
<tr>
<th><strong>First Name</strong></th>
<th><strong>Last Name</strong></th>
<th><strong>Address</strong></th>
<th><strong>City</strong></th>
<th><strong>Age</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mickey</td>
<td>Mouse</td>
<td>123 Fantasy Way</td>
<td>Anaheim</td>
<td>73</td>
</tr>
<tr>
<td>Bat</td>
<td>Man</td>
<td>321 Cavern Ave</td>
<td>Gotham</td>
<td>54</td>
</tr>
<tr>
<td>Wonder</td>
<td>Woman</td>
<td>987 Truth Way</td>
<td>Paradise</td>
<td>39</td>
</tr>
<tr>
<td>Donald</td>
<td>Duck</td>
<td>555 Quack Street</td>
<td>Mallard</td>
<td>65</td>
</tr>
<tr>
<td>Bugs</td>
<td>Bunny</td>
<td>567 Carrot Street</td>
<td>Rascal</td>
<td>58</td>
</tr>
<tr>
<td>Wiley</td>
<td>Coyote</td>
<td>999 Acme Way</td>
<td>Canyon</td>
<td>61</td>
</tr>
<tr>
<td>Cat</td>
<td>Woman</td>
<td>234 Purrfect Street</td>
<td>Hairball</td>
<td>32</td>
</tr>
<tr>
<td>Tweety</td>
<td>Bird</td>
<td>543</td>
<td>Itotltaw</td>
<td>28</td>
</tr>
</tbody>
</table>
View in Web (Data Formats, json / xml / html)

```json
{
    "arguments": { "number": 10 },
    "url": "http://localhost:8080/restty-tester/collection",
    "method": "POST",
    "header": {
        "Content-Type": "application/json"
    },
    "body": [
        {
            "id": 0,
            "name": "name 0",
            "description": "description 0"
        },
        {
            "id": 1,
            "name": "name 1",
            "description": "description 1"
        }
    ],
    "output": "json"
}
```
Controller in Web (Backend Languages)
MVC in Web (Server)

• The JSON represents the view that the user consumes, html, json, xml
• The DB represents the model which stores the data
• The Backend languages (PHP, Python, Javascript, etc.) works with the DB to generate the View.
MVC in Web Apps

- The HTML on the client represents the model of information we want to convey to the user.
- The CSS on the client updates the model so it renders it in a pleasing manor.
- The Javascript on the client gets triggered by the user interacting with the HTML (based on CSS positioning) and updates the HTML model.

- The JSON represents the view that the user consumes, html, json, xml.
- The DB represents the model which stores the data.
- The Backend languages (PHP, Python, Javascript, etc.) works with the DB to generate the View.
MVC in Web Apps

- The Client represents the view
- The Server represents the model which stores the data
- The AJAX with Client & Server Processing represent the controller
Rendering Languages

• Model: Hypertext Markup Language (HTML): used for writing web pages
• View: Cascading Style Sheets (CSS): stylistic info for web pages
• Controller: JavaScript: interactive and programmable web pages
Communication Languages and Technologies (MODEL)

- Asynchronous JavaScript and XML (Ajax): accessing data for web applications
- eXtensible Markup Language (XML): metalanguage for organizing data
- Javascript object notation (json): metalanguage for organizing data
Backend Languages

- PHP Hypertext Processor (PHP): dynamically create content on a standard web server
- Python: dynamically create content on a standard web server
- Javascript (Node.js): Create your own webserver and content that responds on the webserver
- Ruby (on Rails): Use default rails webserver (WEBrick) and dynamically content that is server
Mobile Applications Professional Education Series

Brian Davidson
Research Network Operations Center

RESTFUL APIS
RESTful APIs

RESTful APIs is a collection of resources, with four defined aspects:

- The base URL for the web service
- The Internet media type of the data supported by the web service (e.g., JSON)
- The CRUD set of operations supported by the web service using HTTP methods (i.e., POST, GET, PUT or DELETE)
- The API must be hypertext driven

Collections contain elements of the collection
RESTful APIs

And elements themselves can contain collections themselves which is how you can accomplish more complicated representations.

So let's foreshadow a bit and say that maybe we also want to represent which users have made the comments, and find comments made only by specific users.
RESTful APIs

Let's pretend we are making an application to store comments.

For simplicity, let's continue and say we wouldn't store them in a database but just as flat files we could have them as:

- ~/comments/
- ~/comments/76.txt => Hey Brian
- ~/comments/78.txt => Hello
- .....
RESTful APIs

Lets continue by saying we also want to store information about the users of the application

If we were to pretend these users were files we could have

- ~/users/
- ~/users/bdavidson8.txt => Brian Davidson
- ~/users/lfreil3.txt => Larry Freil
- .....
RESTful APIs

If we were to pretend these comments made by users and we want to track which comments comments by which user

• ~/users/
• ~/users/bdavidson8/comments/78.txt => Hello
• ~/users/bdavidson8/comments/80.txt => What Up
• ~/users/lfreil3/comments/76.txt => Yo Brian
• ~/users/lfreil3/comments/79.txt => Nothing
• ......
RESTful APIs

Isn't the folder comments, redundant in ever user folder couldn't we get rid of it

• ~/users/
• ~/users/bdavidson8/78.txt => Hello
• ~/users/bdavidson8/80.txt => What Up
• .....  

Technically but if we later want to add in a collections of friends how do we represent that

• ~/users/bdavidson8/lfreil3.txt => Larry Freil
• ~/users/bdavidson8/rc57.txt => Russ Clark

That's will cause confusion since the collections are mingling
RESTful APIs

It would be better to represent them with fuller longer URLs having elements inside collections inside elements inside collections .......

• ~/users/
• ~/users/bdavidson8/comments/78.txt => Hello
• ~/users/bdavidson8/comments/80.txt => What Up
• ..... 
• ~/users/bdavidson8/friends/lfreil3.txt => Larry Freil
• ~/users/bdavidson8/friends/rc57.txt => Russ Clark
• .....
CRUD

• In computer programming, create, read, update and delete (CRUD) are the four basic functions of persistent storage. Each letter is implemented in relational database applications mapping to a standard SQL statement:

• Operation   SQL
• Create       INSERT
• Read (Retrieve) SELECT
• Update (Modify) UPDATE
• Delete (Destroy) DELETE
CRUD

• Optionally they can also map to HTTP methods, which along with other constraints, is considered "RESTful".

<table>
<thead>
<tr>
<th>Operation</th>
<th>HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>PUT / POST</td>
</tr>
<tr>
<td>Read (Retrieve)</td>
<td>GET</td>
</tr>
<tr>
<td>Update (Modify)</td>
<td>PUT / PATCH</td>
</tr>
<tr>
<td>Delete (Destroy)</td>
<td>DELETE</td>
</tr>
</tbody>
</table>
RESTful APIs (cont’d)

One-to-one mapping between CRUD (Create, Read, Update, and Delete) operations and HTTP methods:

<table>
<thead>
<tr>
<th>CRUD Operation</th>
<th>HTTP/REST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a resource on the server</td>
<td>POST</td>
</tr>
<tr>
<td>Retrieve a resource</td>
<td>GET</td>
</tr>
<tr>
<td>Change the state of a resource or update it</td>
<td>PUT</td>
</tr>
<tr>
<td>Remove or delete a resource</td>
<td>DELETE</td>
</tr>
</tbody>
</table>

RESTful Web service is stateless
Restful API CRUD

So back to our CRUD what kind of actions would we like to commonly do

• CREATE
  • Create a new entry in the collection. The new entry's URL is assigned automatically and is usually returned by the operation.
  • Collection URI (http://example.org/api/comments)
  • HTTP POST (RESTful POST)
Restful API CRUD

• READ A LIST OF COMMENTS
  • List the URIs and perhaps other details of the collection's members.
  • Collection URI (http://example.org/api/comments)
  • HTTP GET (RESTful INDEX)

• READ A SINGLE COMMENTS
  • Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.
  • Element URI (http://example.org/api/comments/1)
  • HTTP GET (RESTful GET)
Restful API CRUD

• UPDATE A SINGLE COMMENTS
  • Replace the addressed member of the collection, or if it doesn't exist, create it.
  • Element URI (http://example.org/api/comments/1)
  • HTTP PUT (RESTful PUT)
Restful API CRUD

• DELETE A SINGLE COMMENTS
  • Delete the addressed member of the collection.
  • Element URI (http://example.org/api/comments/1)
  • HTTP DELETE (RESTful DELETE)
WEB FRAMEWORKS
( SERVER )