Controller/server communication

Mendel Rosenblum
Controller's role in Model, View, Controller

- Controller's job to fetch model for the view
  - May have other server communication needs as well (e.g. authentication services)
- Browser is already talking to a web server, ask it for the model
- Early approach: have the browser do an HTTP request for the model
  - First people at Microsoft liked XML so the DOM extension got called: XMLHttpRequest
- Allowed JavaScript to do an HTTP request without switching page
- Widely used and called AJAX - Asynchronous JavaScript and XML
- Since it is using an HTTP request it can carry XML or anything else
  - More often used with JSON
**XMLHttpRequest**

**Sending a Request**

```javascript
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

**Event handling**

```javascript
function xhrHandler() {
    if (this.readyState != 4) { // DONE
        return;
    }
    if (this.status != 200) { // OK
        // Handle error ...
        return;
    }
    ...
    var text = this.responseText;
    ...
```
XMLHttpRequest event processing

- Event handler gets called at various stages in the processing of the request

  0  UNSENT          open() has not been called yet.
  1  OPENED          send() has been called.
  2  HEADERS_RECEIVED send() has been called, and headers and status are available.
  3  LOADING         Downloading; responseText holds partial data.
  4  DONE            The operation is complete.

- Response available as:
  - raw text - responseText
  - XML document - responseXML

- Can set request headers and read response headers
Traditional AJAX uses patterns

- Response is HTML
  ```javascript
  elem.innerHTML = xhr.responseText;
  ```
- Response is JavaScript
  ```javascript
  eval(xhr.responseText);
  ```

Neither of the above are the AngularJS way

- Response is model data (JSON frequently uses here)
  ```javascript
  JSON.parse(xhr.responseText);
  ```
Fetching models with XMLHttpRequest

- Controller needs to communicate in the request what model is needed
- Can encode model selection information in request in:

  URL path:   `xhr.open("GET","userModel/78237489/fullname");`

  Query params: `xhr.open("GET","userModel?id=78237489&type=fullname");`

  Request body:

  ```javascript
  xhr.open("POST", url);
  xhr.setRequestHeader("Content-type",
                      "application/x-www-form-urlencoded");
  xhr.send("id=78237489&type=fullname");
  ```
REST APIs

- REST - representational state transfer
- Guidelines for web app to server communications
- 2000 PhD dissertation that was highly impactful
  - Trend at the time was complex Remote Procedure Calls (RPCs) system
  - Became a must have thing: Do you have a REST API?
- Some good ideas, some not so good
  - Doesn't work for everything
Some RESTful API attributes

- Server should export **resources** to clients using unique names (**URIs**)
  - Example: http://www.example.com/photo/ is a collection
  - Example: http://www.example.com/photo/78237489 is a resource

- Keep servers "stateless"
  - Support easy load balancing across web servers
  - Allow caching of resources

- Server supports a set of HTTP methods mapping to **Create**, **Read**, **Update**, **Delete** (CRUD) on resource specified in the URL
  - GET method - Read resource (list on collection)
  - PUT method - Update resource
  - POST method - Create resource
  - DELETE method - Delete resource
REST API design

- Define the **resources** of the service and give them unique names (URIs)
  - Example: Photos, Users, Comments, ...

- Have clients use a CRUD operations using HTTP methods

- Extend when needed (e.g. transaction across multiple resources)
Angular accessing RESTful APIs

- $http - Send an arbitrary HTTP request ($http.get, $http.post)
- $resource - Interact with RESTful server-side data sources

Define a REST resource $resource

```
var resource = $resource(resourceURL, parameters);
```

Perform REST method on the resource

```
resource.get(parameters, callback);
resource.save(parameters, callback);
(query, delete as well)
```
Angular $resource service example - Fetch model

```javascript
var PhotoListOfUser = $resource('/photos/:id', {id: '@id'}, {
  get: {method: 'get', isArray: true}
});

PhotoListOfUser.get({id: userId}, function(userPhotos) {
  console.log('userPhotos', userPhotos);
});
```

Generates a HTTP GET to the URL and returns the model (an array of Photo Models)
Angular $resource service example - Store model

var AddComment = $resource('/commentsOfPhoto/:id', {id: photoId});

AddComment.save({commentText: 'New Comment!'}, function (comment) {
    console.log('Added comment', comment);
});

Generates a HTTP POST (rest create) to the URL and the model created
Other Transports: HTML5 WebSockets

- Rather than running over HTTP, HTML5 brings sockets to the browser
  - TCP connection from JavaScript to backend Web Server
- Event-based interface like XMLHttpRequest:
  ```javascript
  var socket = new WebSocket("ws://www.example.com/socketserver");
  socket.onopen = function (event) {
    socket.send(JSON.stringify(request));
  };
  
  socket.onmessage = function (event) {
    JSON.parse(event.data);
  };
  ```
Remote Procedure Call (RPC)

- Traditional distributed computation technology supporting calling of a function on a remote machine.
  - Browser packages function's arguments into a message to the web server.
  - Function is invoked with the arguments on the server.
  - Function's return value is sent back to the browser.

- Allows arbitrary code to be run on server - handles complex, multiple resource operations
  - Reduces number of round trip messages and makes failure handling easier.

- Can result in more complex to use interface compared to REST
  - Need to document the API (i.e. functions and external calling sequence)

- RPC can be done over HTTP (e.g. POST) or WebSockets
GraphQL

● Standard protocol for backends from Facebook
  ○ Like REST, server exports resources that can be fetched by the web app
  ○ Unlike REST
    ■ Server exports a "schema" describing the resources and supported queries.
    ■ Client specifies what properties of the resource it is interested in retrieving.
    ■ Can fetch from many different resources in the same request (i.e. entire model in one query).

● Update operations specified in the exported schema
  ○ Allows an RPC-like interface

● Gaining in popularity particularly compared to REST