THE UNIVERSITY OF WESTERN AUSTRALIA

Topic 14: Client-Side Rendering

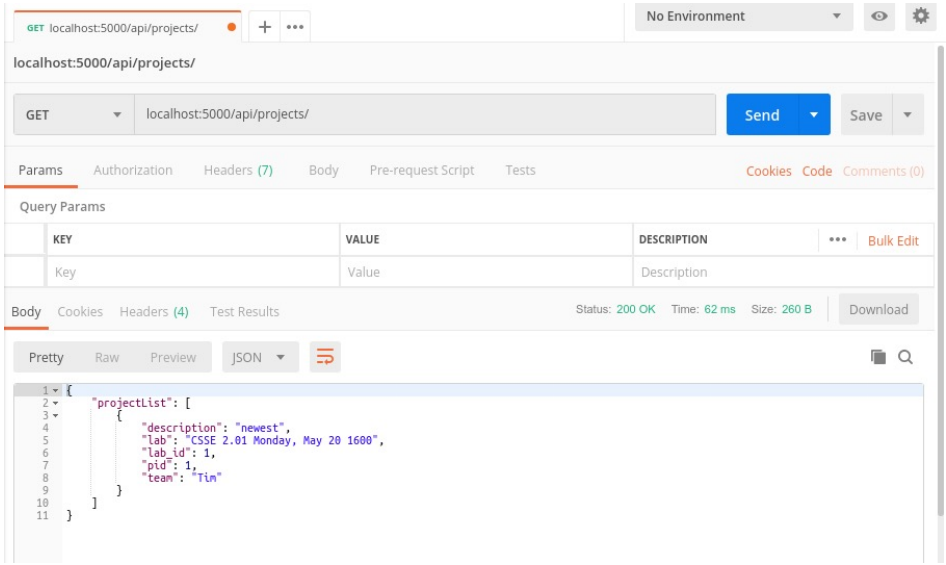
CITS3403 Agile Web Development

Reading: The Flask Mega-Tutorial, part 14
Miguel Grinberg
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-xiv-ajax

Semester 1, 2022

Accessing a REST API

- A REST API takes your application from the *web* to the *internet*. Any device with TCP/IP can interact with the application through HTTP requests.
- We can interact with a REST API through a number of mediums: command line, Postman, or a web browser.
- These applications create and send http requests to the REST API and receive http responses.
- Postman can also be used for mocking APIs and automated testing.




```
(virtual-environment) drtnf@drtnf-ThinkPad:$ http GET http://localhost:5000/api/projects/
getting projects
127.0.0.1 - - [15/May/2019 12:53:05] "GET /api/projects/ HTTP/1.1" 200 -
HTTP/1.0 200 OK
Content-Length: 113
Content-Type: application/json
Date: Wed, 15 May 2019 04:53:05 GMT
Server: Werkzeug/0.14.1 Python/3.6.7

{
    "projectList": [
        {
            "description": "newest",
            "lab": "CSSE 2.01 Monday, May 20 1600",
            "lab_id": 1,
            "pid": 1,
            "team": "Tim"
        }
    ]
}
```

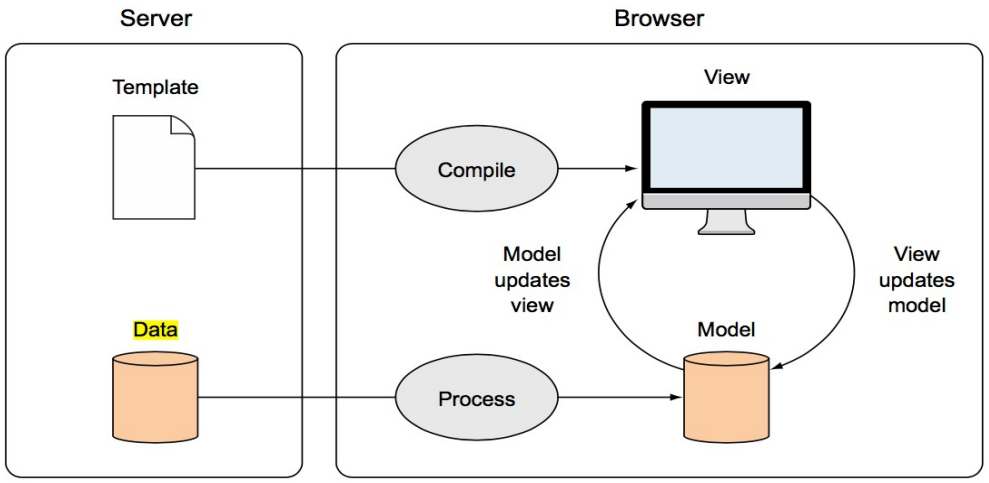
Javascript and DOM

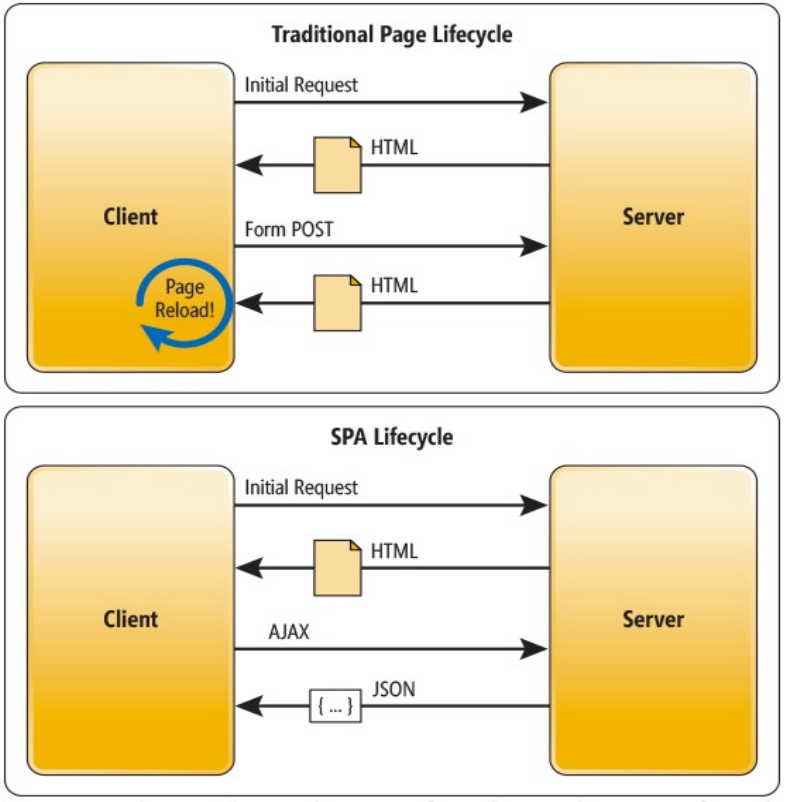
THE UNIVERSITY OF WESTERN AUSTRALIA

- As a simple example of consuming a REST API we will look at writing a low level single page application that interacts directly with the API.
- It will use AJAX to send and receive requests from the server.
- It will use Javascript and DOM to update the web page.
- We will (redundantly) include it with an existing server-side rendering app.

Single Page Applications

- Single Page Applications have the browser/client do the heavy lifting in a web application: The server just provides the data while the client does the logic and rendering

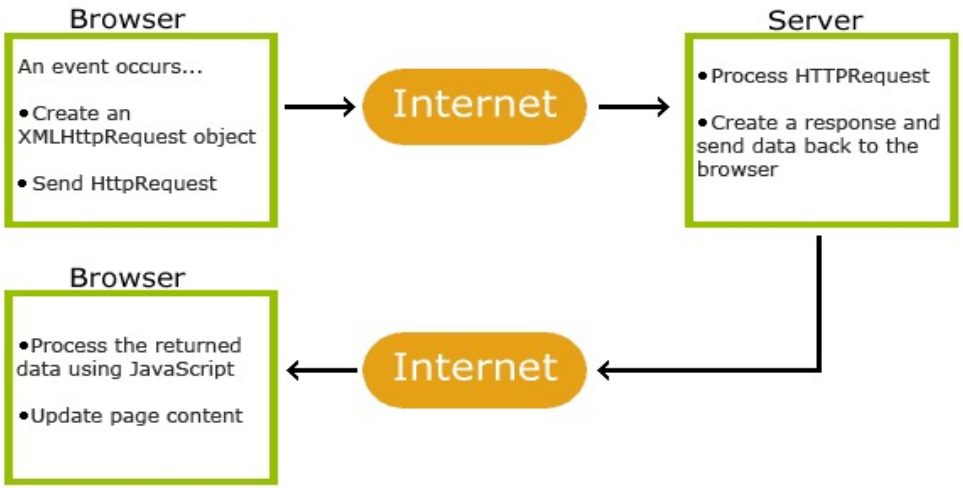




AJAX

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

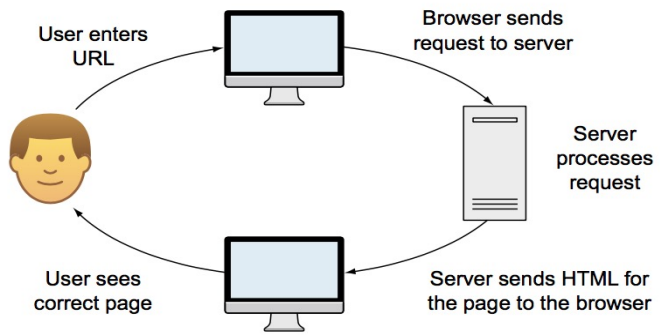
- AJAX is not a programming language.
- AJAX just uses a combination of:
  - A browser built-in XMLHttpRequest object (to request data from a web server)
  - JavaScript and HTML DOM (to display or use the data)
- AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.
- AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.



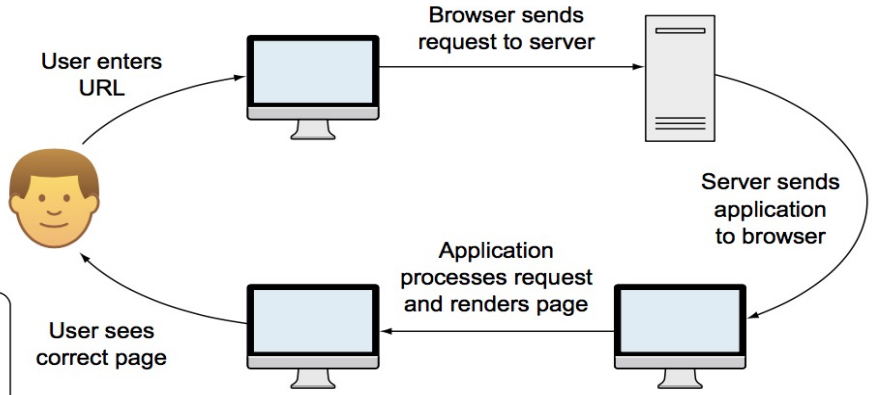
Application Architectures

THE UNIVERSITY OF WESTERN AUSTRALIA

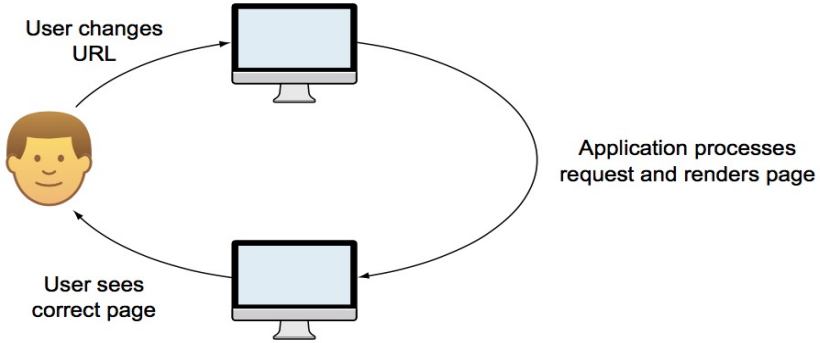
1. Server application request loop



2. SPA initial request loop



3. SPA subsequent request loop



Pros and Cons

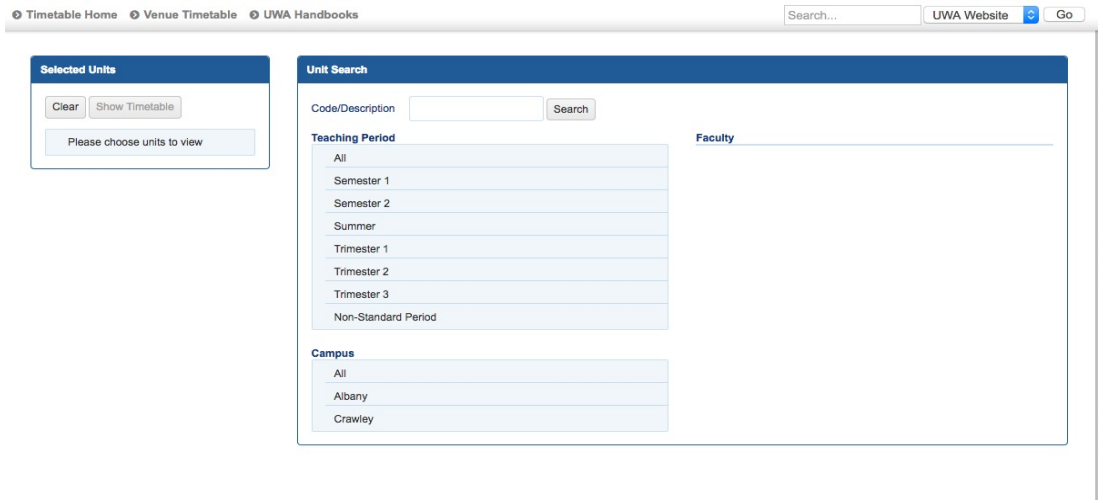
THE UNIVERSITY OF WESTERN AUSTRALIA

Pros

- Less load on the server, able to respond to more clients.
- A more responsive client. No need to wait for server responses.
- Genuine separation between content and presentation.

Cons

- Longer load time, first up. A lot of JS has to be transferred.
- SEO can be a problem. Robots won't crawl js.
- Navigation can be an issue.



Design a Single Page Application in Flask

- We will consider a very lightweight single page application in flask.
- We use the static directory, which is used to store the static resources used by your application.
- In this directory we will include one file `spa.html` to contain the html, and one file `pairup.js` to contain the javascript.
- It is common to use a number of javascript files to orgainse client side models, and enhance reuse.

Building the HTML

- We can use the hidden attribute to populate the HTML with all the attributes we will require, and hide them.
- We create an id attribute for most elements so we can reference them in the DOM.
- We create templates and divs for any future views.

```
<html>
  <head>
    <title>Pair-Up!</title>
      <link rel="stylesheet" media="screen" href="bootstrap.min.css">
      <link rel="stylesheet" href="bootstrap-theme.min.css">
      <script src="jquery-3.4.1.min.js"></script>
      <script src="bootstrap.min.js"></script>
      <script src="pairup.js"></script>
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>

  <body onload='setUp()'>
    <div class='container'>
      <div class='col-sm-4'><!-- empty space --></div>
      <div class='col-sm-4'><h1>Pair Up!</h1></div>
      <div class='col-sm-4' id='menu-panel'>
        <button id='log' value='Login' style='float-right'>
          Login
        </button>
        <button id='project' value='Edit Project' style='float-left' hidden=true>
          New Project
        </button>
      </div>
    </div>

    <div class='jumbotron'>
      <h3>CITS3403 group allocation tool, and flask sample project.</h3>

      <p> The jQuery, AJAX client-side rendering version!</p>
    </div>
      <div class='container' id='login-panel' hidden=True>
          Student Number: <input type='text' id='snum' type='text' size=8>
          Pin: <input id='pin' type ='password' size=4>
          <button name="logSubmit", onclick="login();">Submit</button>
      </div>

      <div class='container' id='project-panel' hidden=true>
          <h4 id='title'>Project Details</h4>
          Partner's Student Number: <input type='text' id='partnerNum' type='text' size=8><br>
          Project Description: <input type='text' id='projectDesc' type = 'text' size=64><br>
          Demonstration Lab: <select id='labs' type = 'selection'></select><br>
          <button id='delete' hidden=true onclick='deleteProject();'>Delete</button>
          <button id="projectSubmit", onclick='updateProject();'>Submit</button>
      </div>

      <div class='container' id='project-list'>
         <table class='table table-striped table-bordered' id='projectTable' hidden=true>
           <tr id='tableHeader'><th>Num</th><th>Team</th><th>Project Description</th><th>Demo Lab</th></tr>
         </table>
      </div>
  <!-- Footer -->
  <footer class="page-footer font-small blue pt-4">
    <!-- Copyright -->
    <div class="footer-copyright text-center py-3">Written by Tim, 2019, for
      <a href="http://teaching.csse.uwa.edu.au/units/CITS3403/index.php?fname=projects&project=yes">_CITS3403_
3403-pair-up">github</a>.
    </div>
    <!-- Copyright -->

  </footer>
```

app/static/spa.html

Using Javascript and DOM

- The javascript will do several things. It will maintain client side models.
- Here we have just declared variables for student, project and authToken, which will be populated by AJAX.
- However we could (should) build more comprehensive models to wrap up these AJAX functions.
- We also create references to the DOM elements we will need to populate.

```
1  //script for java script functions for SPA pair-up app
2  //
3
4  ///Data
5  var authToken=null;//or store in a cookie?
6  var snum=null;//or store in a cookie?
7  var student=null;
8  var url=location.hostname; //use navigator to compute current url for requests
9  var project = null;
10
11 ///DOM elements
12 var loginButton, projectButton, loginPanel, projectPanel, projectTable, snum, pin, partnerNum, projectDesc, labSelect;
13
14
```

Client-side Models

- The client side models are typically different to the server side models (e.g. we do not store passwords etc)
- Remember everything that is sent to the client is fully accessible by the client, and anyone who has access to the client.
- These variable hold the data to be displayed visually (text fields, dates), plus possible some data that will be used in requests (such as primary keys for entities).

```
80    //getlabs with callback
81    //expected format {available_labs:[{labid:3,lab:'lab2.01, Monday 3pm'}
```


```
{"available_labs":[{"lab":"CSSE 2.01 Monday, May 20 1605","lab_id":2},{"lab":"CSSE 2.01 Monday, May 20 1610","lab_id":3},{"lab":"CSSE 2.01 Monday, May 20
1615","lab_id":4},{"lab":"CSSE 2.01 Monday, May 20 1620","lab_id":5},{"lab":"CSSE 2.01 Monday, May 20 1625","lab_id":6},{"lab":"CSSE 2.01 Monday, May 20
1630","lab_id":7},{"lab":"CSSE 2.01 Monday, May 20 1635","lab_id":8},{"lab":"CSSE 2.01 Monday, May 20 1640","lab_id":9},{"lab":"CSSE 2.01 Monday, May 20
1645","lab_id":10},{"lab":"CSSE 2.01 Monday, May 20 1650","lab_id":11},{"lab":"CSSE 2.01 Monday, May 20 1655","lab_id":12},{"lab":"CSSE 2.01 Monday, May 20
1700","lab_id":13},{"lab":"CSSE 2.01 Monday, May 20 1705","lab_id":14},{"lab":"CSSE 2.01 Monday, May 20 1710","lab_id":15},{"lab":"CSSE 2.01 Monday, May 20
1715","lab_id":16},{"lab":"CSSE 2.01 Monday, May 20 1720","lab_id":17},{"lab":"CSSE 2.01 Monday, May 20 1725","lab_id":18},{"lab":"CSSE 2.01 Monday, May 20
1730","lab_id":19},{"lab":"CSSE 2.01 Monday, May 20 1735","lab_id":20},{"lab":"CSSE 2.01 Monday, May 20 1740","lab_id":21},{"lab":"CSSE 2.01 Monday, May 20
1745","lab_id":22},{"lab":"CSSE 2.01 Monday, May 20 1750","lab_id":23},{"lab":"CSSE 2.01 Monday, May 20 1755","lab_id":24},{"lab":"CSSE 2.03 Wednesday, May 22
```


```
177 //Assumes data format {'projectList': [p1,p2,p3]}
178 //where pi = {pid:23, team:'Tim & Friends', description:'Project', lab_id:3, lab='2.01 Mon 3pm'}
179 function getProjectList(){
```

Sending requests

THE UNIVERSITY OF WESTERN AUSTRALIA

- Requests are sent through at XMLHttpRequest object.
- The object is initialised, and then *open*s a connection to a server. The *send* method sends the request to the server, and when the server responds, the *status* and *response* can be accessed as properties.
- Browsers only handle GET and POST requests.

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(*method,url,async,user,psw*)	Specifies the request *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

Preparing a Request

- The requests use a callback (a function that waits for a response before running.
- For authentication the user data is included as authentication fields, and an auth token is received.
- For subsequent requests, that auth token can be included in the header of the request.
- Note, we do not use forms, since we are manually creating the requests.

```
111 //Expected data format
112 //{token:"HASHef+HASH', expiry:'2019-5-30T12:00'}
113 function login(){
114   if(authToken!=null) logout();
115   else{
116     var xhttp = new XMLHttpRequest();
117     xhttp.onreadystatechange = function(){
118       if(this.readyState==4 && this.status==200){
119         responseData = JSON.parse(this.responseText);
120         authToken = responseData['token'];
121         loginButton.innerHTML='Logout';
122         loginPanel.hidden=true;
123         projectButton.hidden=false;
124         getStudent(snum.value);
125       }
126       else if(this.readyState==4)
127         alert(this.statusText);
128     };
129     xhttp.open('POST','/api/tokens',true,user=snum.value, psw=pin.value);
130     xhttp.send();
131   }
132 }
134 function logout(){
135   if(authToken==null) return;
136   var xhttp = new XMLHttpRequest();
137   xhttp.onreadystatechange = function(){
138     if(this.readyState==4 && this.status==204){
139       authToken=null;
140       student=null;
141       project=null;
142       document.getElementById('log').value='Login';
143       loginPanel.hidden=true;
144       projectPanel.hidden=true;
145       projectButton.hidden=true;
146       renderTable([]);
147     }
148     else{
149       alert(this.statusText);
150     }
151   }
152   xhttp.open('DELETE','/api/tokens',true);
153   xhttp.setRequestHeader("Authorization","Bearer "+authToken);
154   xhttp.send();
155 }
```

Callbacks and Asynchrony

- Callbacks allow the browser to run asynchronously. We cannot get user data, until the login is complete, we cannot get project data until we have the student number etc...
- This is often refered to as callback hell, and can make testing and debugging difficult.
- Good callback design requires a knowledge of functional programming.
- In Node, asynchrony is used server side as well.

```
158 //Assumes data format {id:19617810, name:'Tim'}
159 function getStudent(id){
160   if(authToken==null) return;
161   var xhttp = new XMLHttpRequest();
162   xhttp.onreadystatechange = function(){
163     if(this.readyState==4 && this.status==200){
164       responseData = JSON.parse(this.responseText);
165       student = responseData;
166       getProject();
167     }
168     else if(this.readyState==4 && this.status!=404){
169       alert(this.statusText);
170     }
171   }
172   xhttp.open('GET','/api/students/'+id,true);
173   xhttp.setRequestHeader("Authorization","Bearer "+authToken);
174   xhttp.send();
175 }
```

```
199 //Assumes data format
200 //{pid:23, team:'Tim & Friends', description:'Project', lab_id:3, lab='2.01 Mon 3pm'}
201 function getProject(){
202   if(authToken==null) return;
203   var xhttp = new XMLHttpRequest();
204   xhttp.onreadystatechange = function(){
205     if(this.readyState==4 && this.status==200){
206       responseData = JSON.parse(this.responseText);
207       project = responseData;
208       renderProject();
209       getProjectList();
210     }
211     else if(this.readyState==4 && this.status!=404){
212       alert(this.statusText);
213     }
214   }
215   xhttp.open('GET','/api/students/'+student['id']+'/project',true);
216   xhttp.setRequestHeader("Authorization","Bearer "+authToken);
217   xhttp.send();
218 }
```

```
199 //Assumes data format
200 //{pid:23, team:'Tim & Friends', description:'Project', lab_id:3, lab='2.01 Mon 3pm'}
201 function getProject(){
202   if(authToken==null) return;
203   var xhttp = new XMLHttpRequest();
204   xhttp.onreadystatechange = function(){
205     if(this.readyState==4 && this.status==200){
206       responseData = JSON.parse(this.responseText);
207       project = responseData;
208       renderProject();
209       getProjectList();
210     }
211     else if(this.readyState==4 && this.status!=404){
212       alert(this.statusText);
213     }
214   }
215   xhttp.open('GET','/api/students/'+student['id']+'/project',true);
216   xhttp.setRequestHeader("Authorization","Bearer "+authToken);
217   xhttp.send();
218 }
```

Populating the DOM

- Once we have received JSON from the server we can populate the HTML elements using javascript and DOM.
- The setup function is ran when the document first loads and initialises the objects.

```
function renderProject(){
  if(project==null){
    projectPanel.title.innerHTML='New Project';
    partnerNum.hidden=False;
    projectDesc.value='';
    projectPanel.delete.hidden=true;
  }
  else{
    document.getElementById('title').innerHTML=project['team']+"'s Project";
    partnerNum.hidden=true;
    projectDesc.value=project['description'];
    document.getElementById('delete').hidden=false;
  }
  //getlabs with callback
  //expected format {available_labs:[{labid:3,lab:'lab2.01, Monday 3pm'},...]
  if(authToken==null) return;
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function(){
    if(this.readyState==4 && this.status==200){
      responseData = JSON.parse(this.responseText);
      availableLabs = responseData['available_labs'];
      if(project!=null){
        availableLabs.unshift({'lab_id': project['lab_id'], 'lab': project['lab_name']});
      }
      labSelect.innerHtml='';
      for(var i = 0; i<availableLabs.length; i++){
        opt = document.createElement('OPTION');
        opt.value = availableLabs[i]['lab_id'];
        opt.innerText = availableLabs[i]['lab'];
        labSelect.appendChild(opt);
      }
    }
    else if(this.readyState==4){
      alert(this.statusText);
    }
  }
  xhttp.open('GET','/api/labs/',true);
  xhttp.setRequestHeader("Authorization","Bearer "+authToken);
  xhttp.send();
}

```

```
function renderTable(projectList){
  tableHeader=document.getElementById('tableHeader');
  projectTable.innerText='';
  projectTable.appendChild(tableHeader);
  for(var i =0; i<projectList.length; i++){
    tr = document.createElement('TR');
    if(project!=null && project['pid']==projectList[i]['pid']) tr.setAttribute('bg-color','green');
    td=document.createElement('TD');
    td.innerHTML=i+1;
    tr.appendChild(td);
    td=document.createElement('TD');
    td.innerHTML=projectList[i]['team'];
    tr.appendChild(td);
    td=document.createElement('TD');
    td.innerHTML=projectList[i]['description'];
    tr.appendChild(td);
    td=document.createElement('TD');
    td.innerHTML=projectList[i]['lab'];
    tr.appendChild(td);
    projectTable.appendChild(tr);
  }
  projectTable.hidden=false;
}

```

```
////////////////HTML Rendering Functions/////////////
function setUp(){
  loginButton = document.getElementById('log');
  projectButton = document.getElementById('project');
  loginPanel = document.getElementById('login-panel');
  snum = document.getElementById('snum');
  pin = document.getElementById('pin');
  projectPanel = document.getElementById('project-panel');
  partnerNum = document.getElementById('partnerNum');
  projectDesc = document.getElementById('projectDesc');
  labSelect = document.getElementById('labs');
  projectTable = document.getElementById('projectTable');
  loginButton.onclick = function(){
    if(authToken==null)
      loginPanel.hidden=!loginPanel.hidden;
    else{
      logout();
      loginButton.innerHTML='Login';
      loginPanel.hidden=true;
      projectPanel.hidden=true;
      renderTable([]);
    }
  };
  projectButton.onclick=function(){
    projectPanel.hidden=!projectPanel.hidden;
  };
}
```

Posting Data

- To POST or PUT data we extract user entered data from the input elements and create javascript objects.
- We include the JSON as a parameter of the send function.
- We must set the content type to JSON so that the flask API will accept the data.
- To DELETE data we expect a different response type.

```
220 //sends data {partnerNUmber:'19617810', description:'Project', lab_id:3}
221 function newProject(){
222   //POST request with new fields for project
223   if(authToken==null) return;
224   var xhttp = new XMLHttpRequest();
225   xhttp.onreadystatechange = function(){
226     if(this.readystate==4 && this.status==201){
227       getProject();
228     }
229     else if(this.readystate==4){
230       alert(this.statusText);
231     }
232   }
233   var project={};
234   project['partnerNumber']=partnerNum.value;
235   project['description']=projectDesc.value;
236   project['lab_id']=labSelect.value;
237   xhttp.open('POST','/api/students/'+student['id']+'/project',true);
238   xhttp.setRequestHeader("Authorization","Bearer "+authToken);
239   xhttp.setRequestHeader('Content-Type','application/json');
240   xhttp.send(JSON.stringify(project));
241 }
```

```
243 //sends data {description:'Project', lab_id:3}
244 function updateProject(){
245   //PUT request with new fields for project
246   if(project==null) newProject();
247   else{
248     if(authToken==null) return;
249     var xhttp = new XMLHttpRequest();
250     xhttp.onreadystatechange = function(){
251       if(this.readystate==4 && this.status==200){
252         renderProject();
253       }
254       else if(this.readystate==4){
255         alert(this.statusText);
256       }
257     }
258     project['description']=projectDesc.value;
259     project['lab_id']=labSelect.value;
260     xhttp.open('PUT','/api/students/'+student['id']+'/project',true);
261     xhttp.setRequestHeader("Authorization","Bearer "+authToken);
262     xhttp.setRequestHeader('Content-Type','application/json');
263     xhttp.send(JSON.stringify(project));
264   }
265 }
```

```
267 function deleteProject(){
268   if(authToken==null) return;
269   var xhttp = new XMLHttpRequest();
270   xhttp.onreadystatechange = function(){
271     if(this.readystate==4 && this.status==204){
272       project = null;
273       renderProject();
274     }
275     else if(this.readystate==4){
276       alert(this.statusText);
277     }
278   }
279   xhttp.open('DELETE','/api/students/'+student['id']+'/project',true);
280   xhttp.setRequestHeader("Authorization","Bearer "+authToken);
281   xhttp.send();
282 }
```

Using jQuery

- jQuery offers some low-level methods to make these operations more succinct

```
$( "#dtr" ).click(function() {
    $.ajax({
       url: '{{ url('employees/profile/dtr/data?id=').$profile->fempidno }}',
       dataType: 'json',
       success: function (data) {
           console.log(data);
           $('#datatable tr').not(':first').not(':last').remove();
           var html = '';
           for(var i = 0; i < data.length; i++){
               html += '<tr>'+
                           '<td>' + data[i].famin + '</td>' +
                           '<td>' + data[i].famout + '</td>' +
                           '<td>' + data[i].fpmin + '</td>' +
                           '<td>' + data[i].fpmout + '</td>' +
                       '</tr>';
               }
           $('#datatable tr').first().after(html);
       },
       error: function (data) {
       }
   });
});
```

June 2016 Go

DATE	#	AM IN	AM OUT	PM IN	PM OUT
Wed	1	07:35	12:07	12:35	6:19
Thu	2	07:46	12:25	12:45	5:18
Fri	3	07:31	12:12	12:37	7:10

```
<div class="row">
    <div class="col-md-8">
        <div class="portlet-body">
            <div class="">
                    <select id="months">
                        <option value="1">January</option>
                        <option value="2">February</option>
                        <option value="3">March</option>
                        <option value="4">April</option>
                        <option value="5">May</option>
                        <option value="6">June</option>
                        <option value="7">July</option>
                        <option value="8">August</option>
                        <option value="9">September</option>
                        <option value="10">October</option>
                        <option value="11">November</option>
                        <option value="12">December</option>
                    </select>
                    <select id="years"></select>
                    <button id="dtr" class="btn btn-sm btn-primary"> Go </button>

                <table class="table table-striped table-bordered table-hover" id="datatable">
                        <tr>
                            <th>AM IN</th>
                            <th>AM OUT</th>
                            <th>PM IN</th>
                            <th>PM OUT</th>
                        </tr>
                </table>
            </div>
        </div>
    </div>
</div>
```

AngularJS

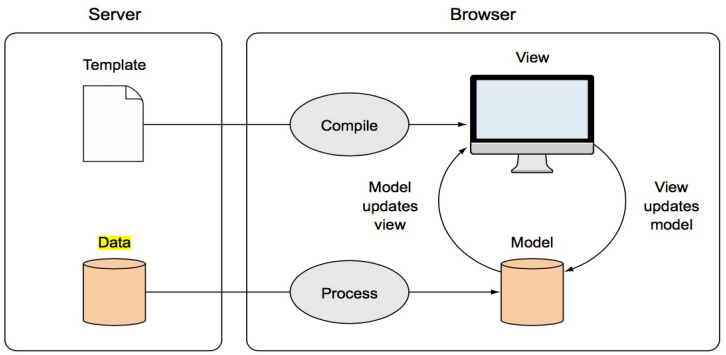
THE UNIVERSITY OF WESTERN AUSTRALIA

Angular is a MVC Javascript Framework by Google for Rich Web Application Development

"Other frameworks deal with HTML's shortcomings by either abstracting away HTML, CSS, and/or JavaScript or by providing an imperative way for manipulating the DOM. Neither of these address the root problem that HTML was not designed for dynamic views".

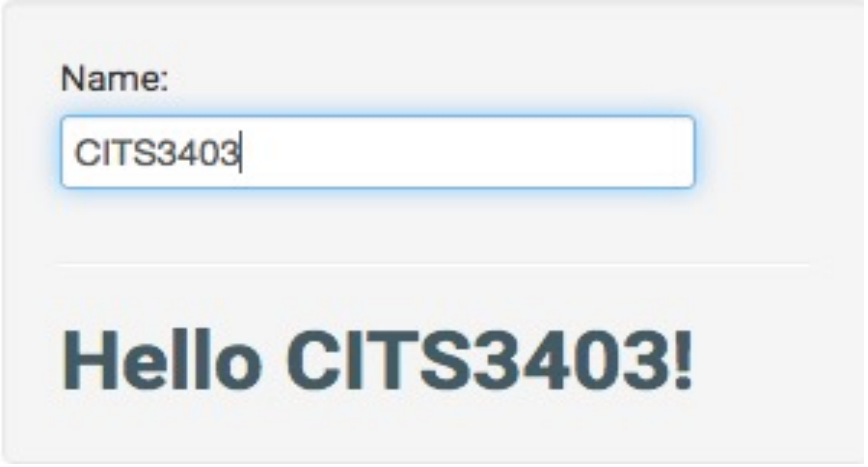
- Structure, Quality and Organization
- Lightweight (< 36KB compressed and minified)
- Free
- Separation of concern
- Modularity
- Extensibility & Maintainability
- Reusable Components

- Two-way Data Binding – Model as single source of truth
- Directives – Extend HTML
- MVC
- Dependency Injection
- Testing
- Deep Linking (Map URL to route Definition)
- Server-Side Communication



Data Binding

```
1.  <!doctype html>
2.  <html ng-app>
3.    <head>
4.      <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/
    angular.min.js"></script>
5.    </head>
6.    <body>
7.      <div>
8.        <label>Name:</label>
9.        <input type="text" ng-model="yourName" placeholder="Enter a name here"
    >
10.       <hr>
11.       <h1>Hello {{yourName}}!</h1>
12.     </div>
13.   </body>
14. </html>
```

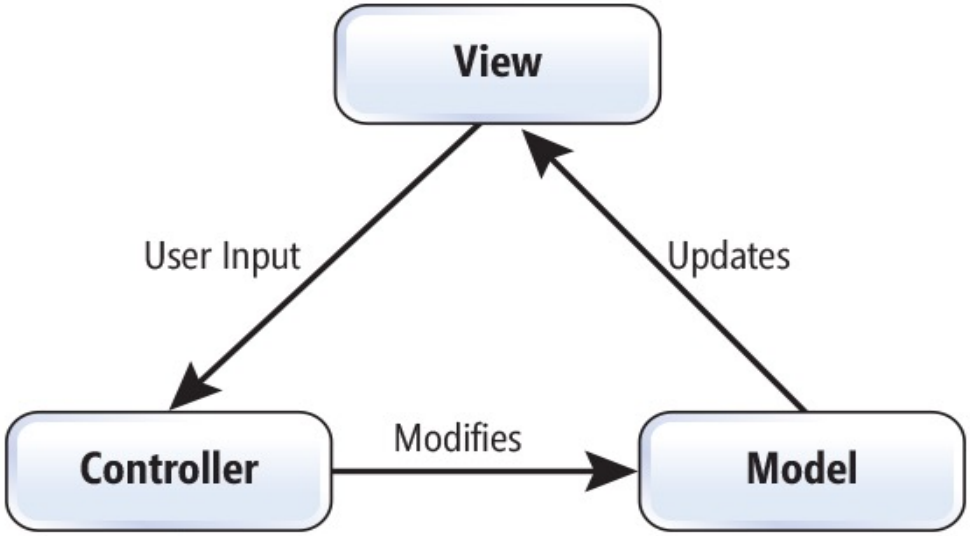


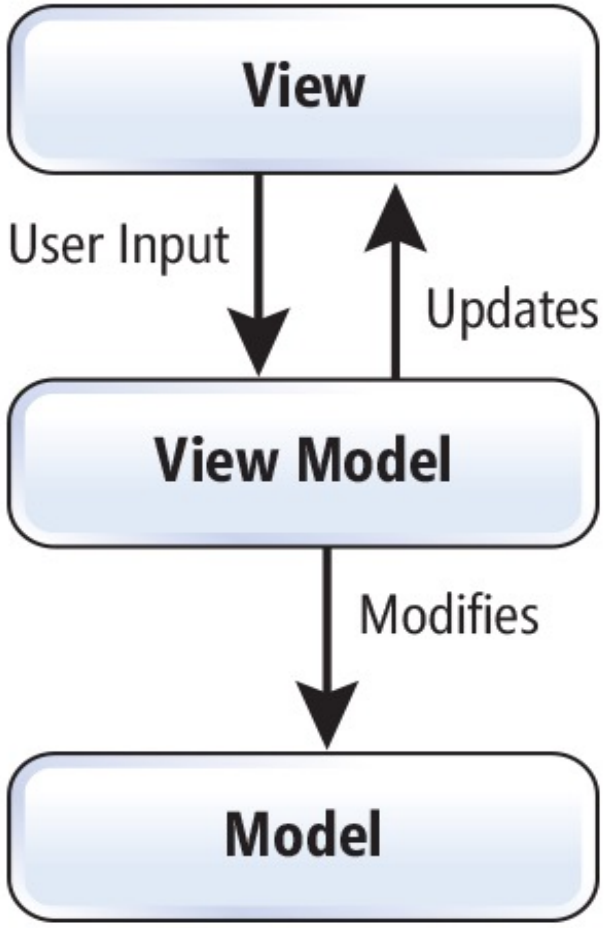
Angular Concepts

THE UNIVERSITY OF WESTERN AUSTRALIA

Template	HTML with additional markup used to describe what should be displayed
Directive	Allows developer to extend HTML with own elements and attributes (reusable pieces)
Scope	Context where the model data is stored so that templates and controllers can access
Compiler	Processes the template to generate HTML for the browser
Data Binding	Syncing of the data between the Scope and the HTML (two ways)
Dependency Injection	Fetching and setting up all the functionality needed by a component
Module	A container for all the parts of an application
Service	A way of packaging functionality to make it available to any view

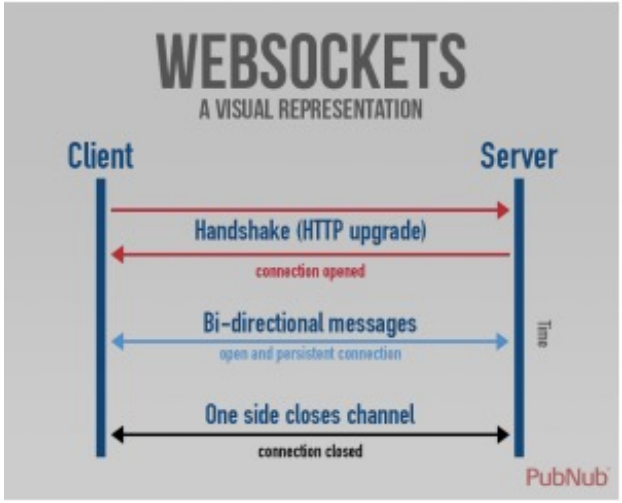
MVC vs MVVM

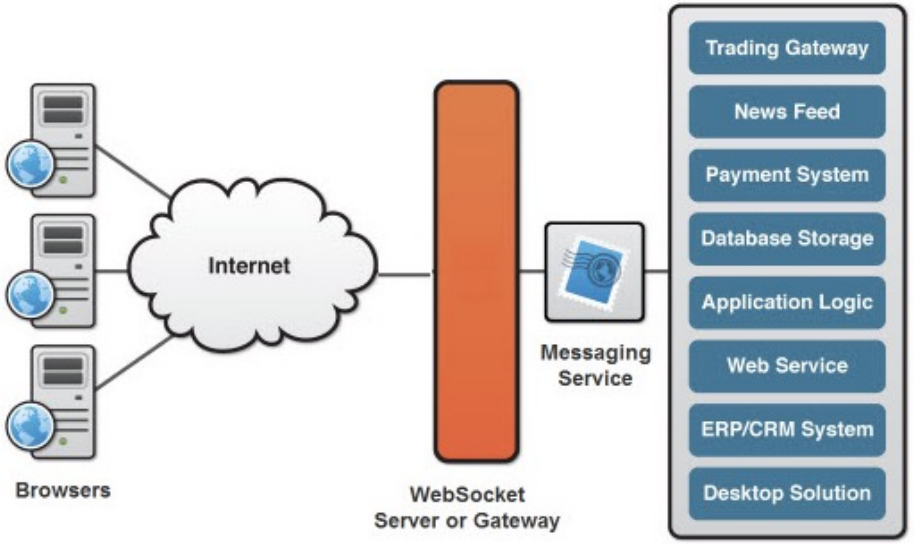

View
User Input
Updates
Controller
Modifies
Model


View
User Input
Updates
View Model
Modifies
Model

Websockets

- HTTP requests satisfy the 6 REST fundamentals, but many web applications depend on real time interaction.
- Websockets were standardise in 2011 as a means to provide full duplex communication.
- WebSockets allow your client-side JavaScript to open a persistent connection (stream) to the server.
- This allows real time communication in the application without having to send HTTP requests.





SocketIO

- Websockets are supported in Flask via the package flask-socketIO (see https://flask-socketio.readthedocs.io/en/latest/)
- SocketIO is good for message passing chat or distributed games.
- For direct video and audio, WebRTC can be used (peer-to-peer).
- Clients can connect to a socket on a server, and then the server can push messages to clients.
- The client has a *listener* architecture so it will respond to the push immediately.

Sockets in a Flask Project

THE UNIVERSITY OF WESTERN AUSTRALIA

- Sockets mirror the routes architecture of a Flask project, but instead of listening for requests, they listen for messages and actions, and broadcast to all listening clients.
- The server works as a common blackboard for the session (or room) and the clients implement a listening architecture via jQuery.
- The socketIO architecture maintains rooms that users/processes can subscribe to.
- Clients and server interact by emitting events including *join, status, message,* and *leave.* You can also create customised events for clients to create and receive.
- We will follow a simple demonstration from Miguel Grinberg taken from: https://github.com/miguelgrinberg/Flask-SocketIO-Chat

Setting up the server

- We use a similar architecture. A main folder called *main*, containing a *forms.py* for registration, *routes.py* for handling login, and a *events.py* file for handling the socket events.
- The socketio includes a decorator to match incoming messages with python methods.
- We don't use models, as there is no persistence here.

```python
from flask import session
from flask_socketio import emit, join_room, leave_room
from .. import socketio


@socketio.on('joined', namespace='/chat')
def joined(message):
    """Sent by clients when they enter a room.
    A status message is broadcast to all people in the room."""
    room = session.get('room')
    join_room(room)
    emit('status', {'msg': session.get('name') + ' has entered the room.'}, room=room)


@socketio.on('text', namespace='/chat')
def text(message):
    """Sent by a client when the user entered a new message.
    The message is sent to all people in the room."""
    room = session.get('room')
    emit('message', {'msg': session.get('name') + ':' + message['msg']}, room=room)


@socketio.on('left', namespace='/chat')
def left(message):
    """Sent by clients when they leave a room.
    A status message is broadcast to all people in the room."""
    room = session.get('room')
    leave_room(room)
    emit('status', {'msg': session.get('name') + ' has left the room.'}, room=room)

```

Implementing the front-end

- We use jQuery to send events to the server, listen for events coming from the server, and update the DOM accordingly.


```html
<html>
    <head>
        <title>Flask-SocketIO-Chat: {{ room }}</title>
        <script type="text/javascript" src="//code.jquery.com/jquery-1.4.2.min.js"></script>
        <script type="text/javascript" src="//cdnjs.cloudflare.com/ajax/libs/socket.io/1.3.6/socket.io.min.js"></script>
        <script type="text/javascript" charset="utf-8">
            var socket;
            $(document).ready(function(){
                socket = io.connect('http://' + document.domain + ':' + location.port + '/chat');
                socket.on('connect', function() {
                    socket.emit('joined', {});
                });
                socket.on('status', function(data) {
                    $('#chat').val($('#chat').val() + '<' + data.msg + '>\n');
                    $('#chat').scrollTop($('#chat')[0].scrollHeight);
                });
                socket.on('message', function(data) {
                    $('#chat').val($('#chat').val() + data.msg + '\n');
                    $('#chat').scrollTop($('#chat')[0].scrollHeight);
                });
                $('#text').keypress(function(e) {
                    var code = e.keyCode || e.which;
                    if (code == 13) {
                        text = $('#text').val();
                        $('#text').val('');
                        socket.emit('text', {msg: text});
                    }
                });
            });
            function leave_room() {
                socket.emit('left', {}, function() {
                    socket.disconnect();

                    // go back to the login page
                    window.location.href = "{{ url_for('main.index') }}";
                });
            }
        </script>
    </head>
    <body>
        <h1>Flask-SocketIO-Chat: {{ room }}</h1>
        <textarea id="chat" cols="80" rows="20"></textarea><br><br>
        <input id="text" size="80" placeholder="Enter your message here"><br><br>
        <a href="#" onclick="leave_room();">Leave this room</a>
    </body>
</html>
```

Other applications for sockets

- Sockets can be used for distributing real time events such as real-time scoreboards, stock prices, or weather.
- Implementing user-ids and sessions (next lecture) can allow you to have private chats between two users.
- Socket.io allows you to group sockets into namespaces and rooms, which allows you to control who can access and post messages.

```
from flask_socketio import join_room, leave_room

@socketio.on('join')
def on_join(data):
    username = data['username']
    room = data['room']
    join_room(room)
    send(username + ' has entered the room.', room=room)

@socketio.on('leave')
def on_leave(data):
    username = data['username']
    room = data['room']
    leave_room(room)
    send(username + ' has left the room.', room=room)
```