THE UNIVERSITY OF WESTERN AUSTRALIA

Topic 14: Client-Side Rendering

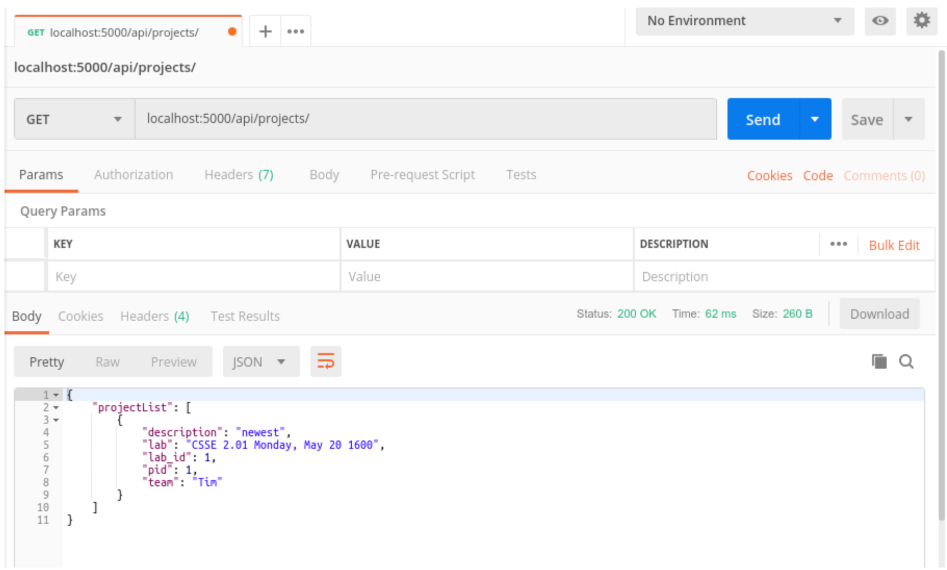
CITS3403 Agile Web Development

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

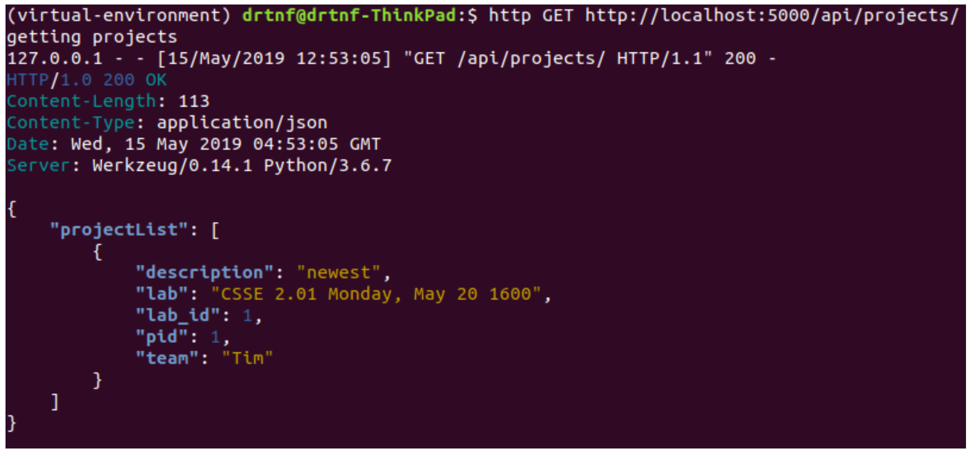
Semester 1, 2023

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.



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



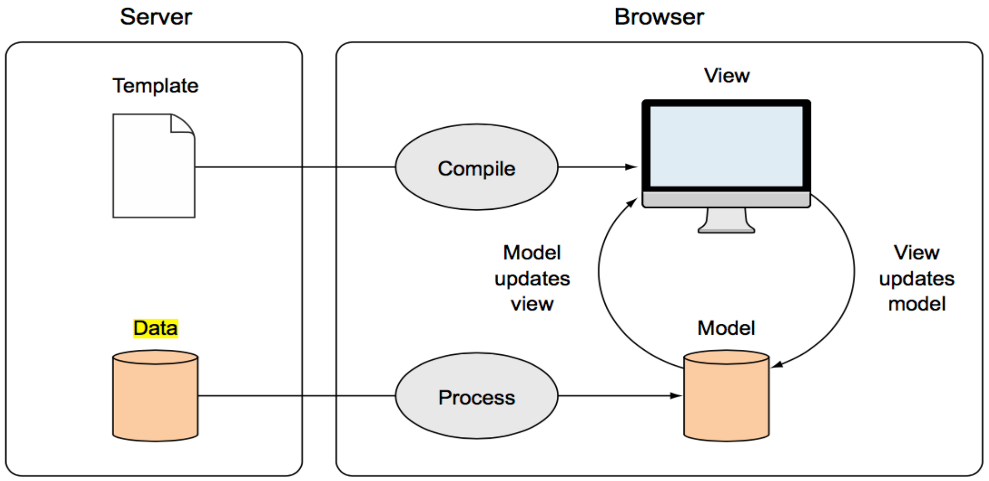
```
(virtual-environment) drtnf@drtnf-ThinkPad:$ http GET http://localhost:5000/api/projects/
getting projects
127.0.0.1 - - [15/May/2019 12:53:05] "GET /api/projects/ HTTP/1.1" 200 -
HTTP/1.0 200 OK
Content-Length: 113
Content-Type: application/json
Date: Wed, 15 May 2019 04:53:05 GMT
Server: Werkzeug/0.14.1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

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

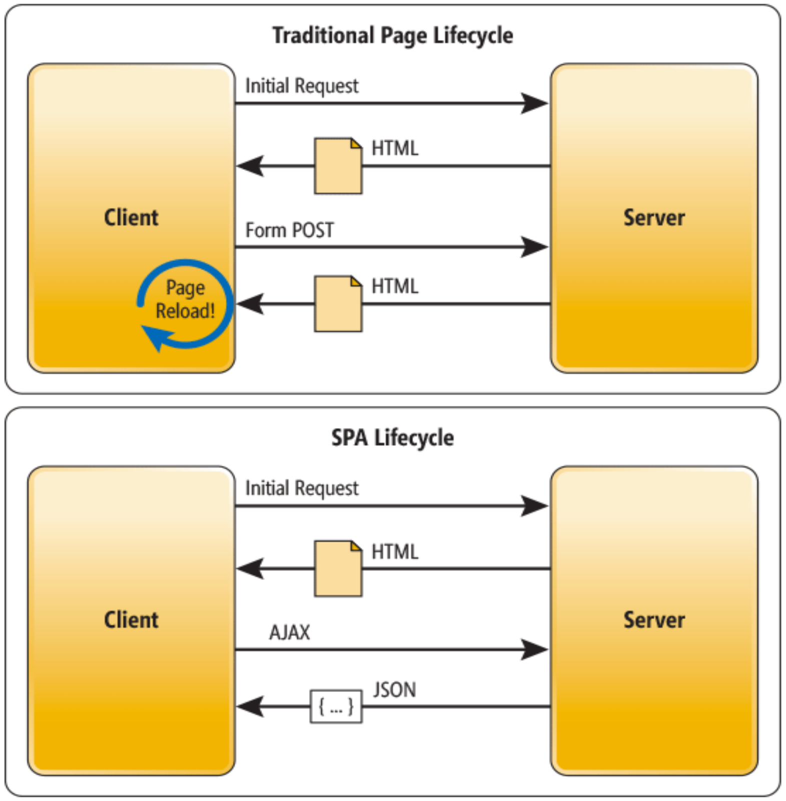
Javascript and DOM

- 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

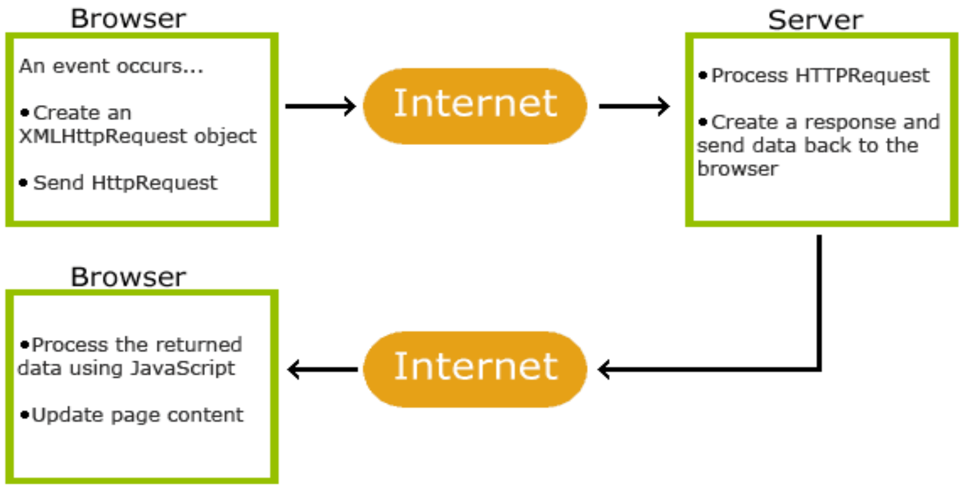




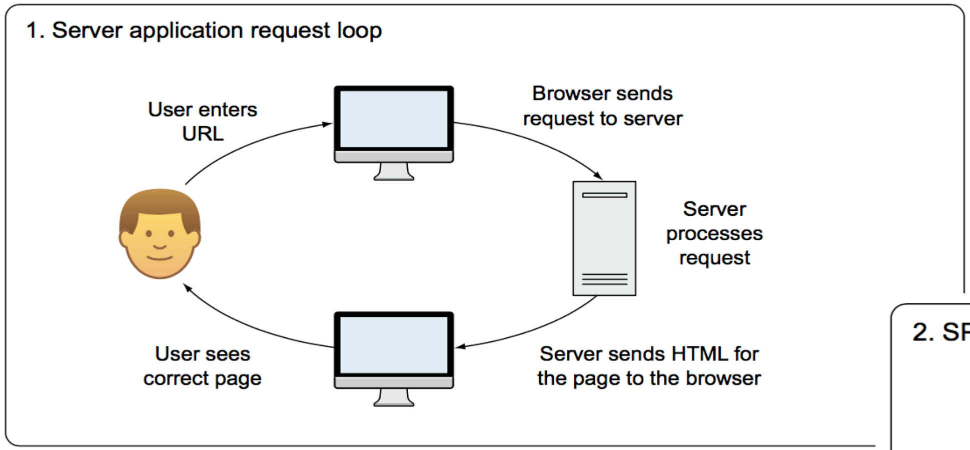
AJAX

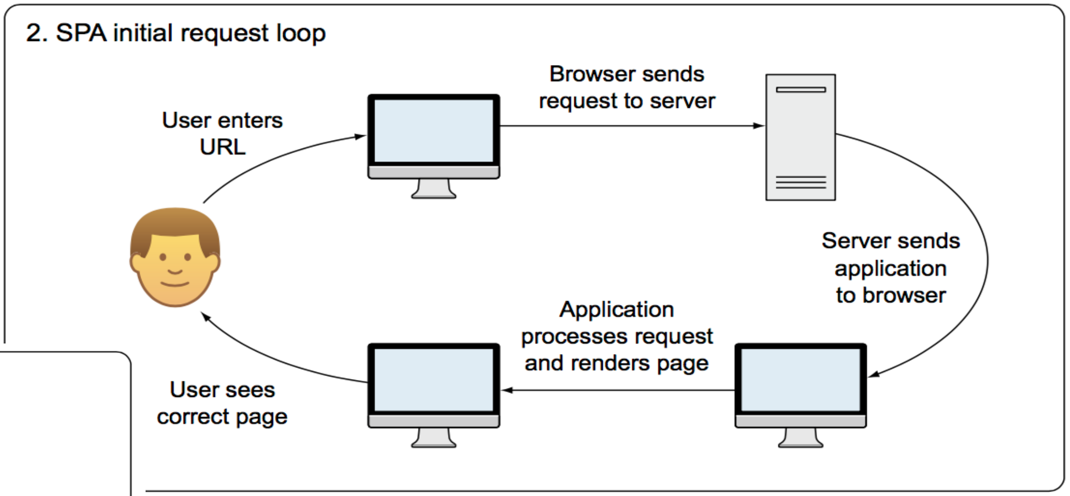
AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

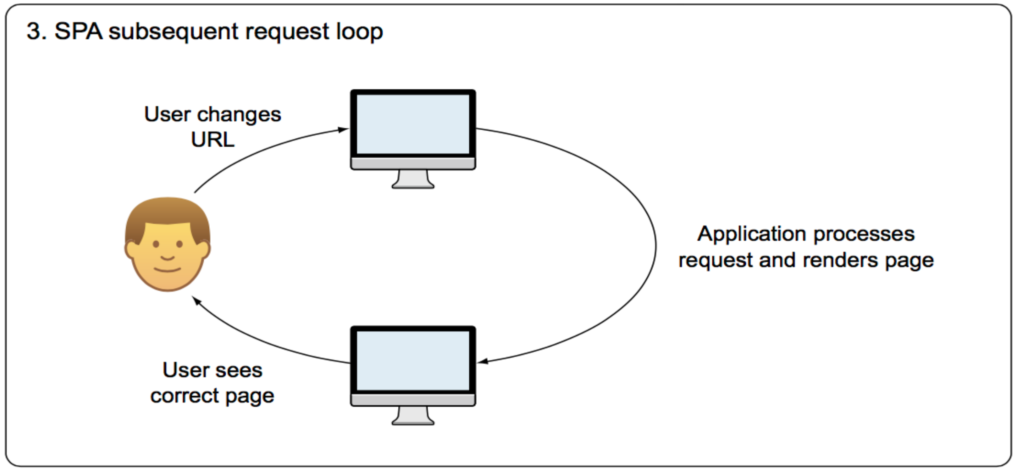
- AJAX is not a programming language.
- AJAX just uses a combination of:
  - A browser built-in XMLHttpRequest object (to request data from a web server)
  - JavaScript and HTML DOM (to display or use the data)
- AJAX is a misleading name. AJAX applications might use XML to transport data, but it is 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.


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Internet
Browser
•Process the returned data using JavaScript
•Update page content

Application Architectures


1. Server application request loop
User enters URL
Browser sends request to server
Server processes request
Server sends HTML for the page to the browser
User sees correct page


2. SPA initial request loop
User enters URL
Browser sends request to server
Server sends application to browser
Application processes request and renders page
User sees correct page


3. SPA subsequent request loop
User changes URL
Application processes request and renders page
User sees correct page

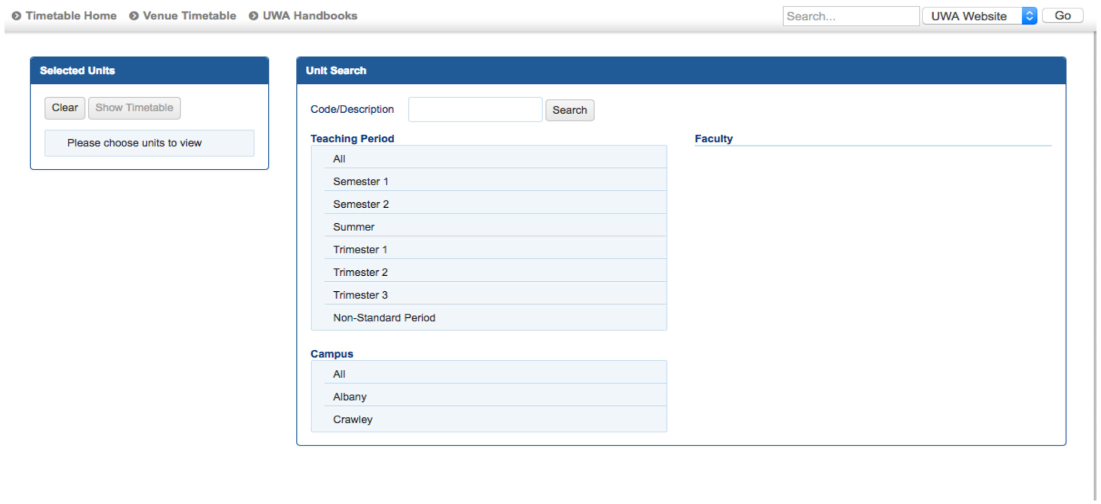
Pros and Cons

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.

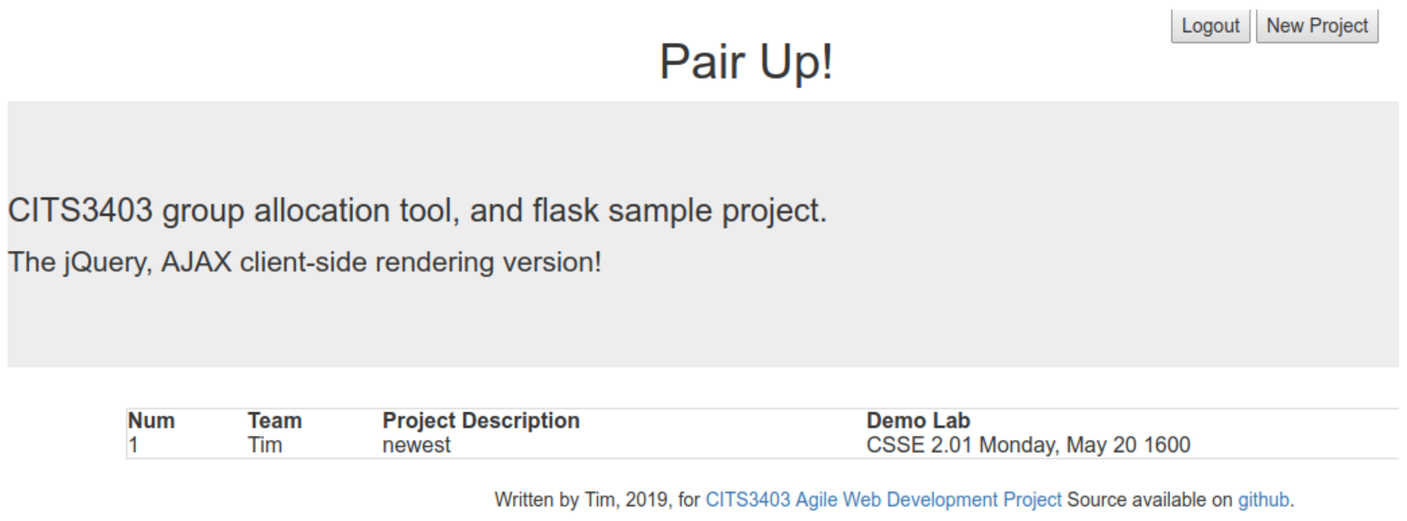
Cons

- 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.



Pair Up!

CITS3403 group allocation tool, and flask sample project.

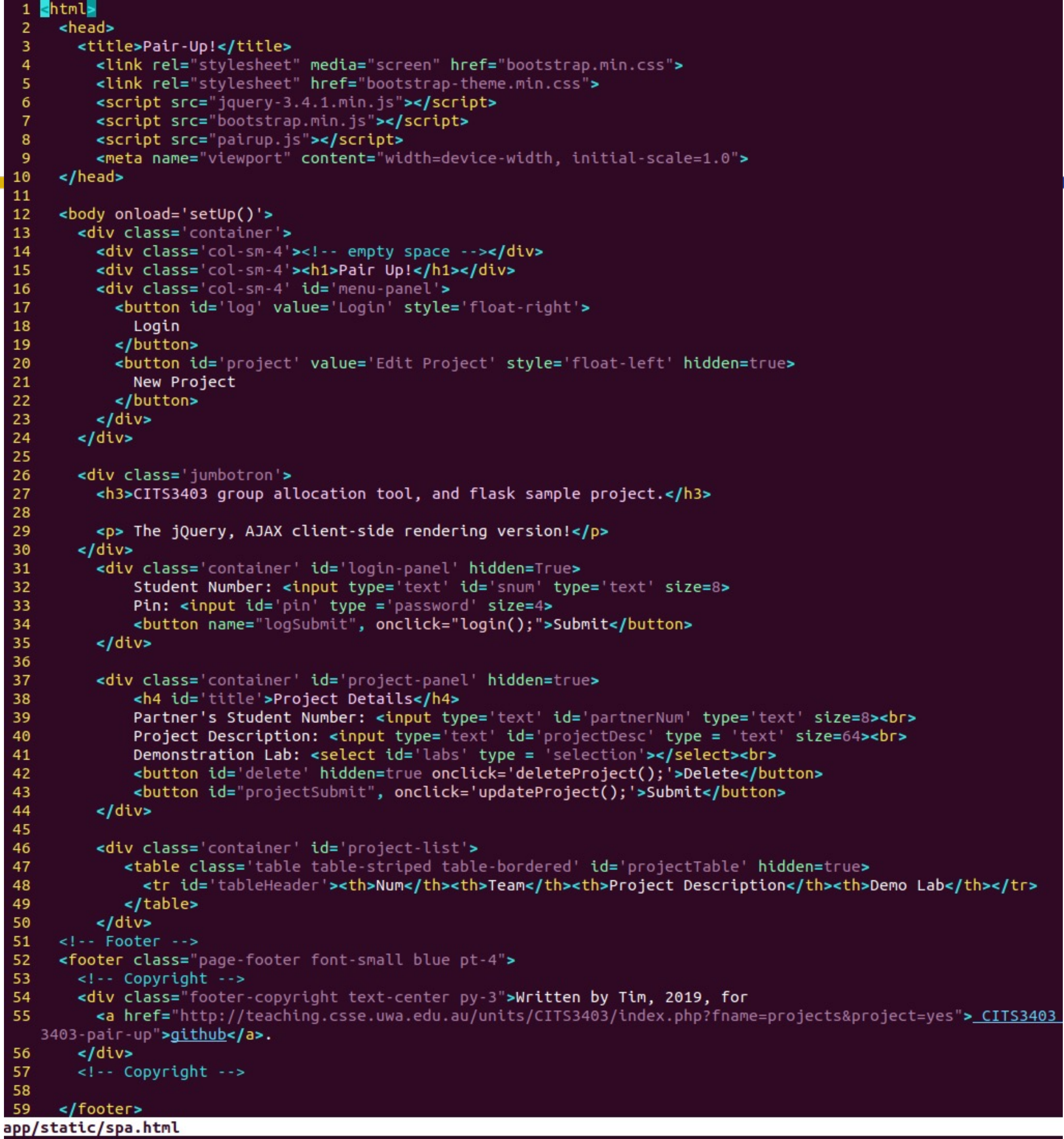
The jQuery, AJAX client-side rendering version!

Num	Team	Project Description	Demo Lab
1	Tim	newest	CSSE 2.01 Monday, May 20 1600

Written by Tim, 2019, for CITS3403 Agile Web Development Project Source available on github.

Building the HTML

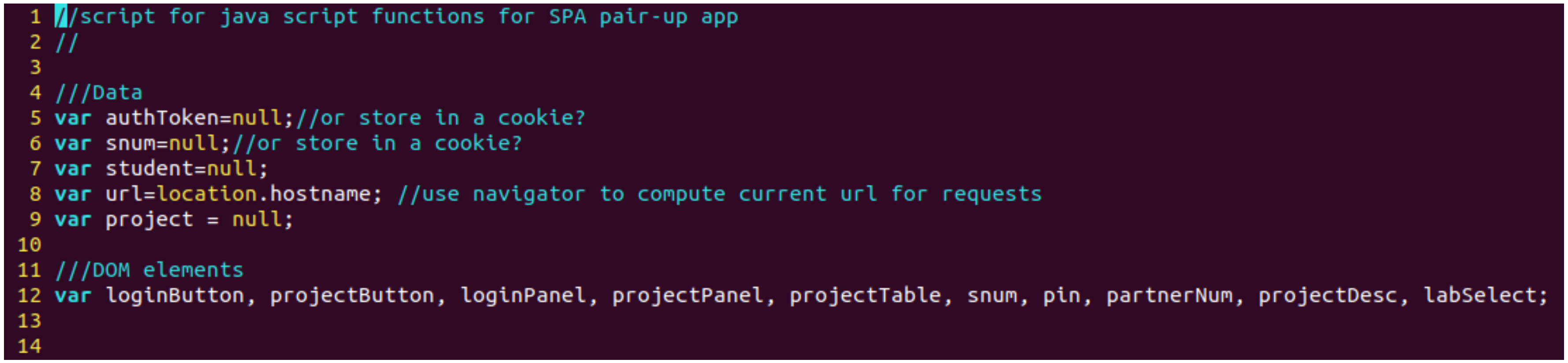
- We can use the hidden attribute to populate the HTML with all the attributes we will require, and hide them.
- We create an id attribute for most elements so we can reference them in the DOM.
- We create templates and divs for any future views.



```
<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

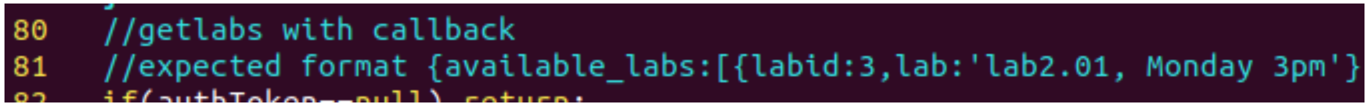
- The javascript will do several things. It will maintain client side models.
- Here we have just declared variables for student, project and authToken, which will be populated by AJAX.
- However we could (should) build more comprehensive models to wrap up these AJAX functions.
- We also create references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



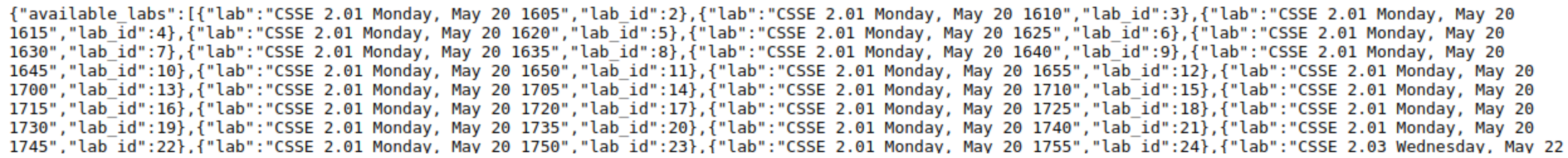
```
//script for java script functions for SPA pair-up app
//

///Data
var authToken=null;//or store in a cookie?
var snum=null;//or store in a cookie?
var student=null;
var url=location.hostname; //use navigator to compute current url for requests
var project = null;

///DOM elements
var loginButton, projectButton, loginPanel, projectPanel, projectTable, snum, pin, partnerNum, projectDesc, labSelect;


```

Client-side Models

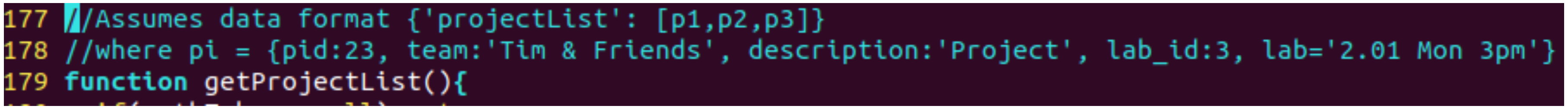
- The client side models are typically different to the server side models (e.g. we do not store passwords etc)
- Remember everything that is sent to the client is fully accessible by the client, and anyone who has access to the client.
- These variable hold the data to be displayed visually (text fields,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



```
//getlabs with callback
//expected format {available_labs:[{labid:3,lab:'lab2.01, Monday 3pm'}
if(authToken==null) return;
```



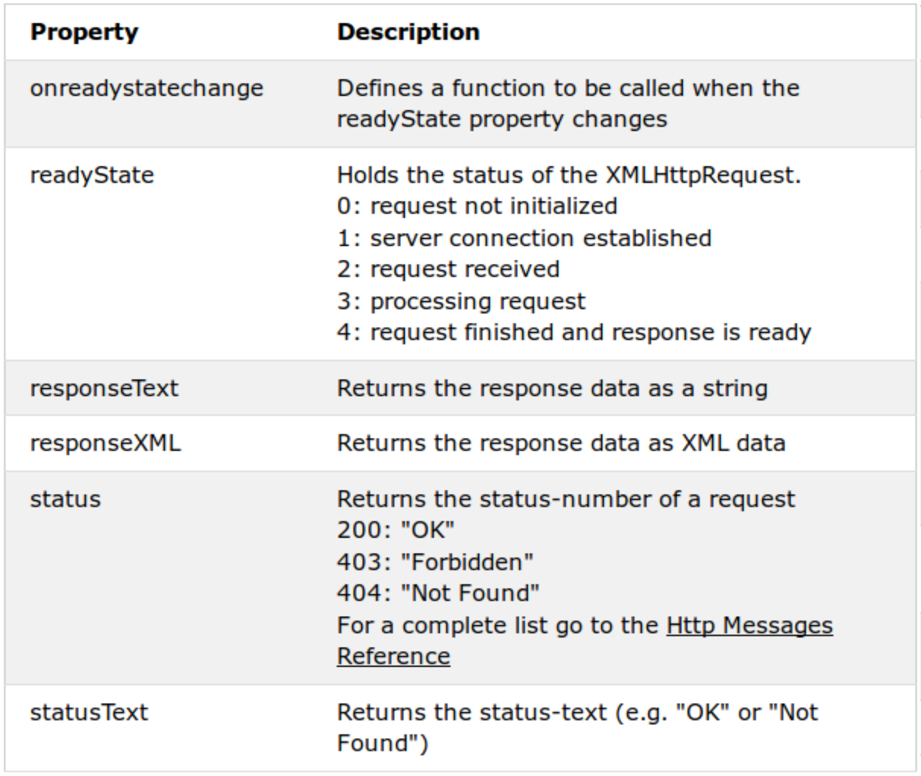
```
{"available_labs":[{"lab":"CSSE 2.01 Monday, May 20 1605","lab_id":2},{"lab":"CSSE 2.01 Monday, May 20 1610","lab_id":3},{"lab":"CSSE 2.01 Monday, May 20
1615","lab_id":4},{"lab":"CSSE 2.01 Monday, May 20 1620","lab_id":5},{"lab":"CSSE 2.01 Monday, May 20 1625","lab_id":6},{"lab":"CSSE 2.01 Monday, 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
```



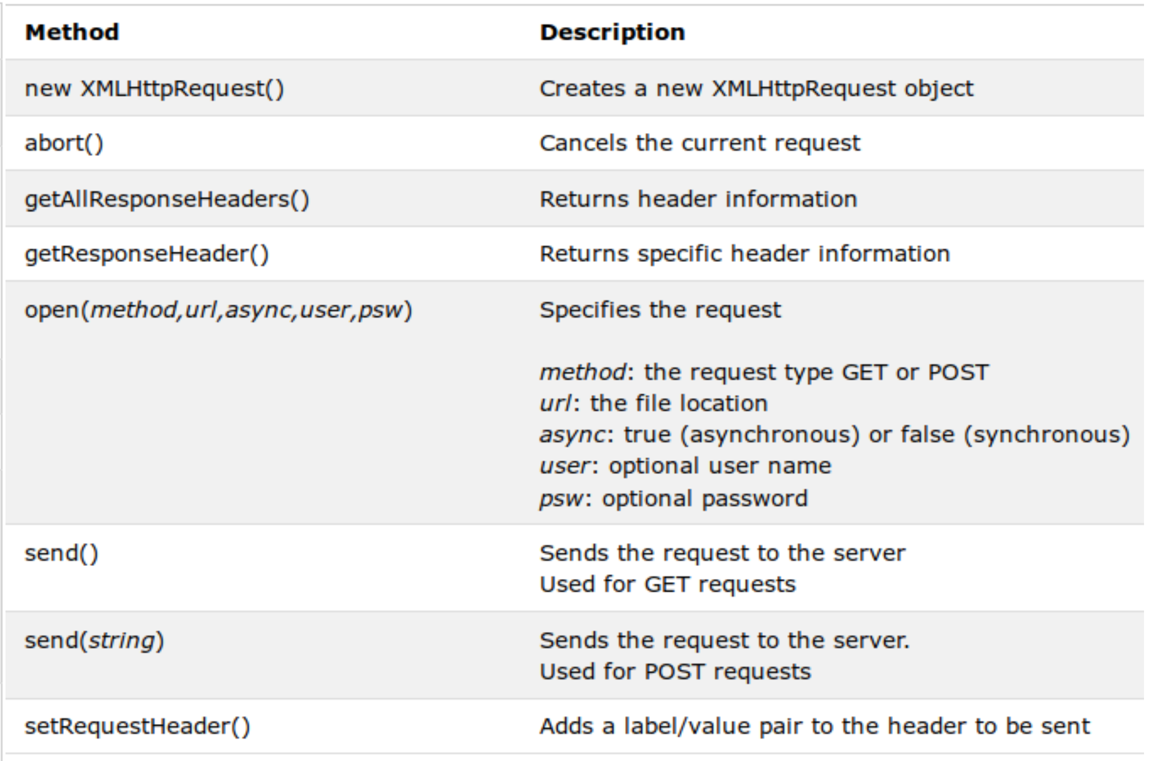
```
//Assumes data format {'projectList': [p1,p2,p3]}
//where pi = {pid:23, team:'Tim & Friends', description:'Project', lab_id:3, lab='2.01 Mon 3pm'}
function getProjectList(){
```

Sending requests

- Requests are sent through at XMLHttpRequest object.
- The object is initialised, and then *opens* 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.



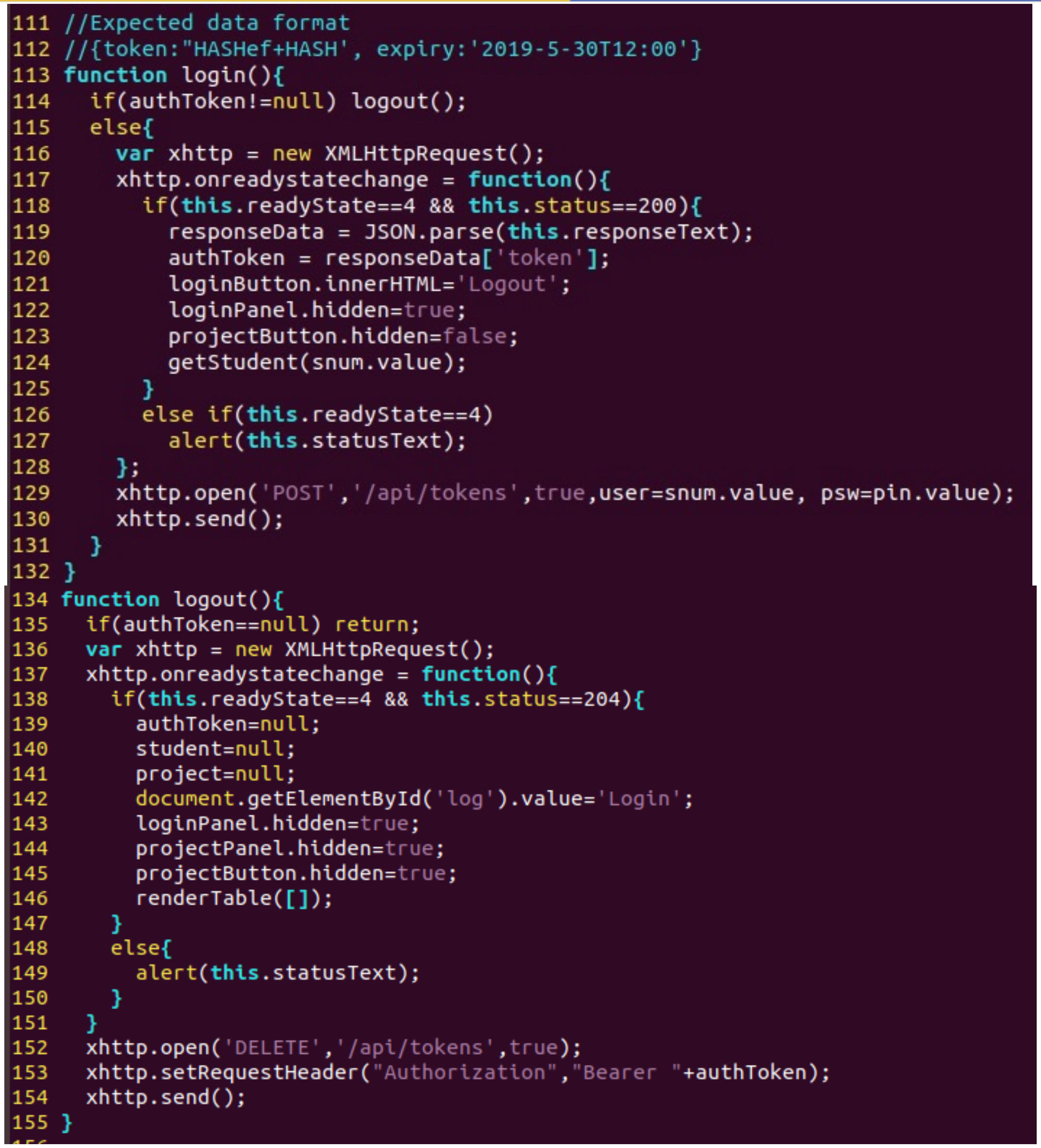
Property	**Description**
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is 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.

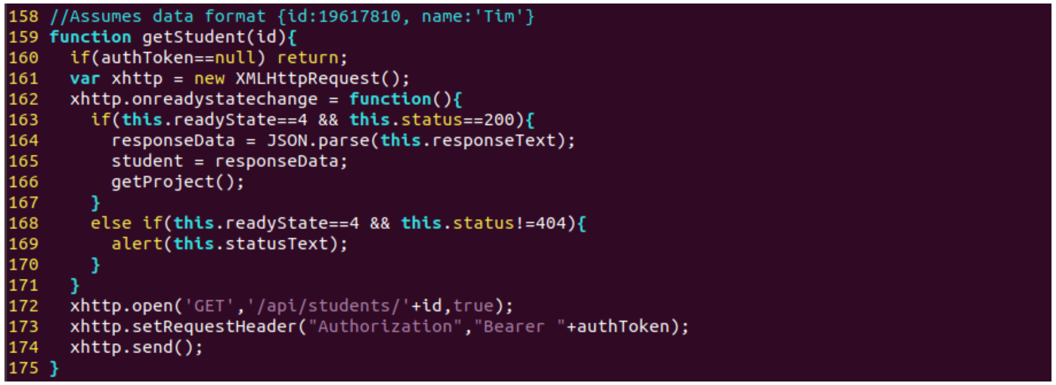


```
//Expected data format
//{token:"HASHef+HASH', expiry:'2019-5-30T12:00'}
function login(){
  if(authToken!=null) logout();
  else{
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function(){
      if(this.readyState==4 && this.status==200){
        responseData = JSON.parse(this.responseText);
        authToken = responseData['token'];
        loginButton.innerHTML='Logout';
        loginPanel.hidden=true;
        projectButton.hidden=false;
        getStudent(snum.value);
      }
      else if(this.readyState==4)
        alert(this.statusText);
    };
    xhttp.open('POST','/api/tokens',true,user=snum.value, psw=pin.value);
    xhttp.send();
  }
}
```

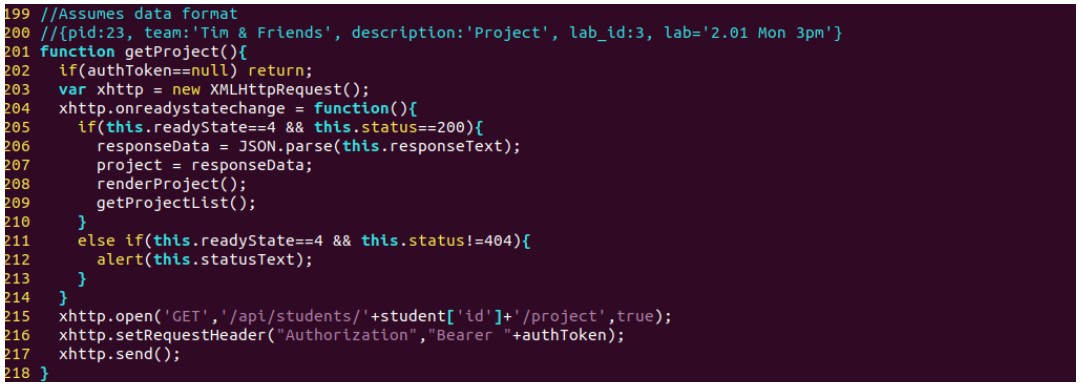
```
function logout(){
  if(authToken==null) return;
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function(){
    if(this.readyState==4 && this.status==204){
      authToken=null;
      student=null;
      project=null;
      document.getElementById('log').value='Login';
      loginPanel.hidden=true;
      projectPanel.hidden=true;
      projectButton.hidden=true;
      renderTable([]);
    }
    else{
      alert(this.statusText);
    }
  }
  xhttp.open('DELETE','/api/tokens',true);
  xhttp.setRequestHeader("Authorization","Bearer "+authToken);
  xhttp.send();
}
```

Callbacks and Asynchrony

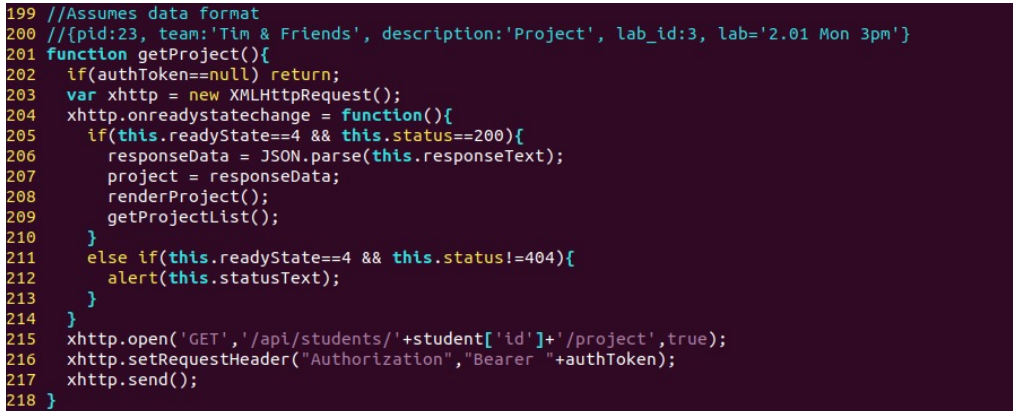
- Callbacks allow the browser to run asynchronously. We cannot get user data, until the login is complete, we cannot get project data until we have the student number etc...
- This is often refered to as callback hell, and can make testing and debugging difficult.
- Good callback design requires a knowledge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



```
//Assumes data format {id:19617810, name:'Tim'}
function getStudent(id){
  if(authToken==null) return;
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function(){
    if(this.readyState==4 && this.status==200){
      responseData = JSON.parse(this.responseText);
      student = responseData;
      getProject();
    }
    else if(this.readyState==4 && this.status!=404){
      alert(this.statusText);
    }
  }
  xhttp.open('GET','/api/students/'+id,true);
  xhttp.setRequestHeader("Authorization","Bearer "+authToken);
  xhttp.send();
}
```



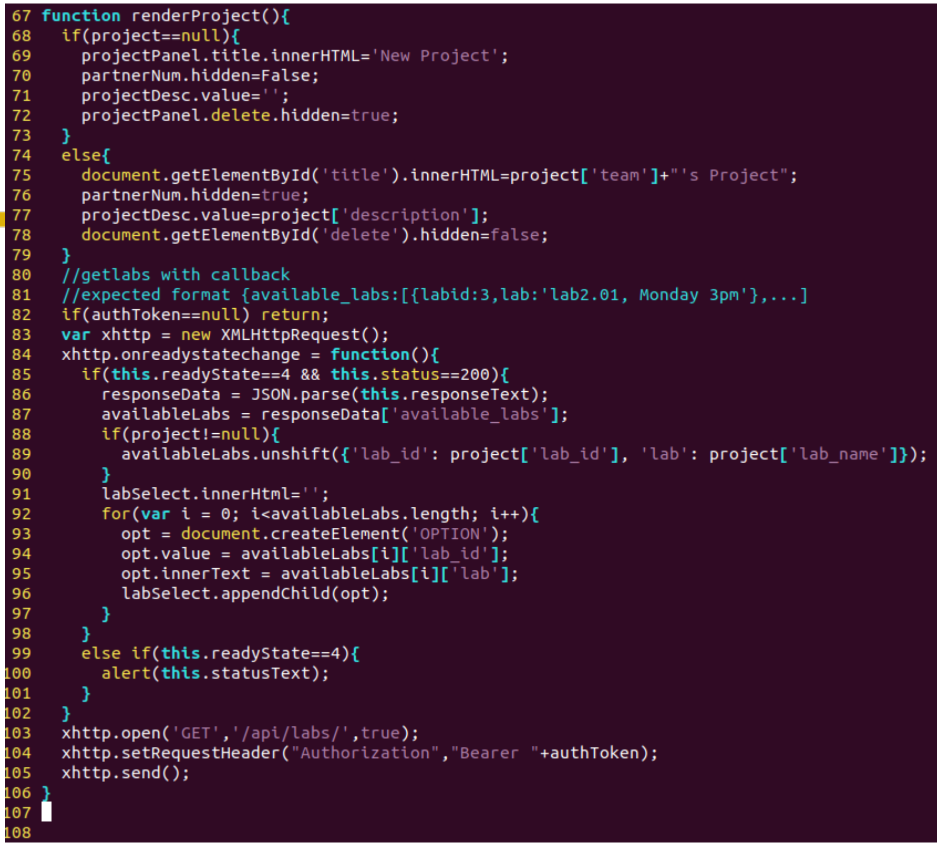
```
//Assumes data format
//{pid:23, team:'Tim & Friends', description:'Project', lab_id:3, lab='2.01 Mon 3pm'}
function getProject(){
  if(authToken==null) return;
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function(){
    if(this.readyState==4 && this.status==200){
      responseData = JSON.parse(this.responseText);
      project = responseData;
      renderProject();
      getProjectList();
    }
    else if(this.readyState==4 && this.status!=404){
      alert(this.statusText);
    }
  }
  xhttp.open('GET','/api/students/'+student['id']+'/project',true);
  xhttp.setRequestHeader("Authorization","Bearer "+authToken);
  xhttp.send();
}
```



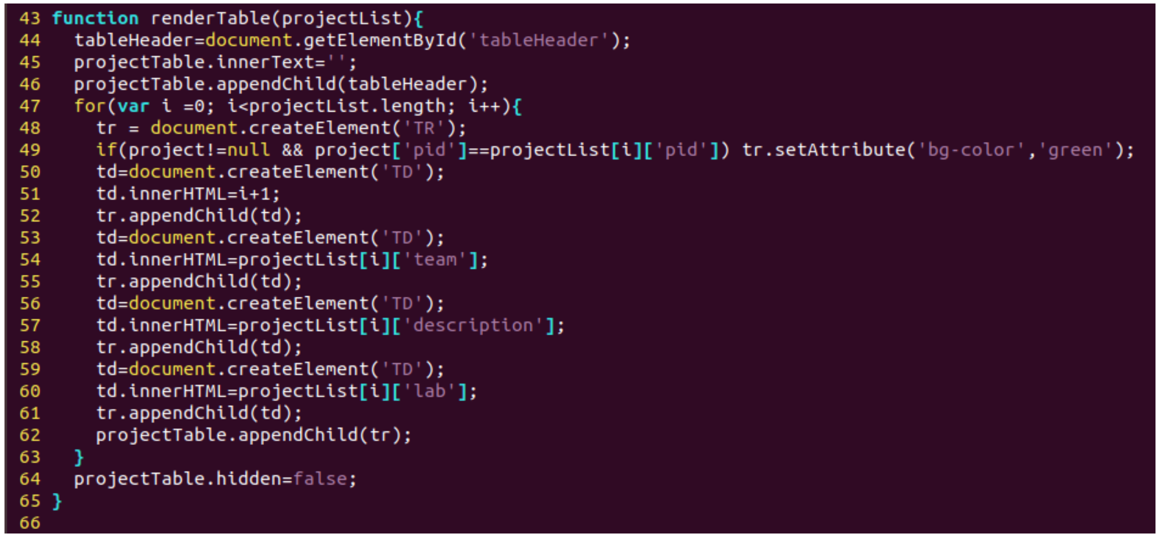
```
//Assumes data format
//{pid:23, team:'Tim & Friends', description:'Project', lab_id:3, lab='2.01 Mon 3pm'}
function getProject(){
  if(authToken==null) return;
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function(){
    if(this.readyState==4 && this.status==200){
      responseData = JSON.parse(this.responseText);
      project = responseData;
      renderProject();
      getProjectList();
    }
    else if(this.readyState==4 && this.status!=404){
      alert(this.statusText);
    }
  }
  xhttp.open('GET','/api/students/'+student['id']+'/project',true);
  xhttp.setRequestHeader("Authorization","Bearer "+authToken);
  xhttp.send();
}
```

Populating the DOM

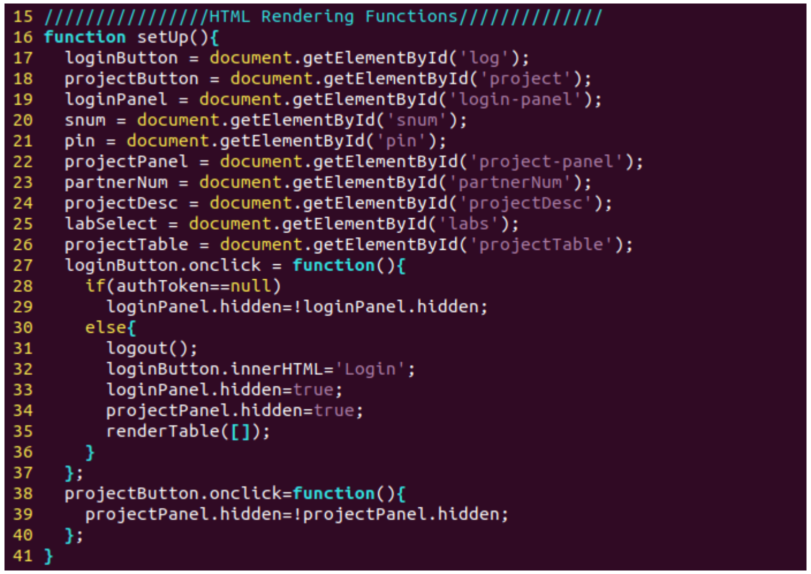
- Once we have received JSON from the server we can populate the HTML elements using javascript and DOM.
- The setup function is ran when the document first loads and initialises the objects.



```javascript
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();
}
```



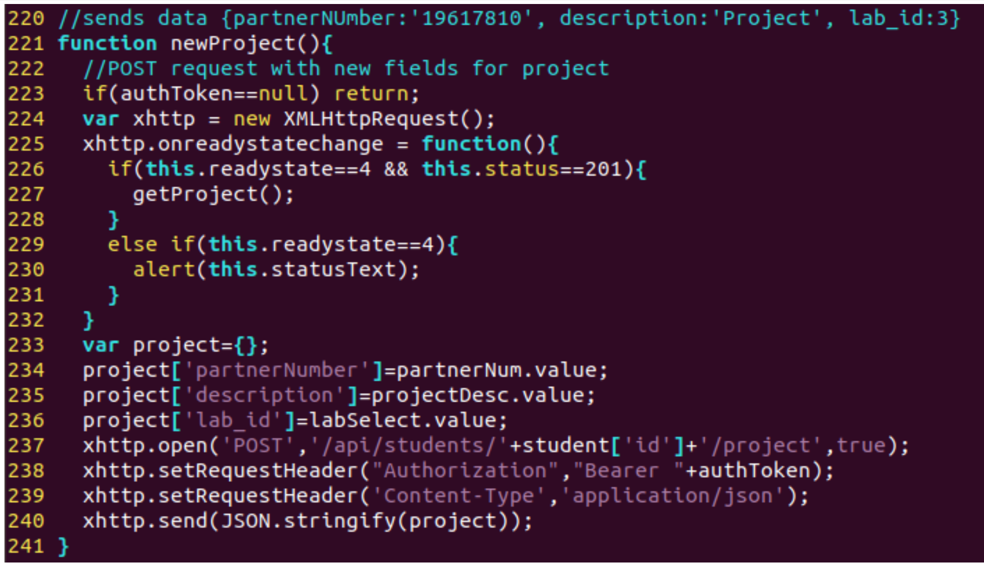
```javascript
function renderTable(projectList){
  tableHeader=document.getElementById('tableHeader');
  projectTable.innerText='';
  projectTable.appendChild(tableHeader);
  for(var i =0; i<projectList.length; i++){
    tr = document.createElement('TR');
    if(project!=null && project['pid']==projectList[i]['pid']) 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;
}
```



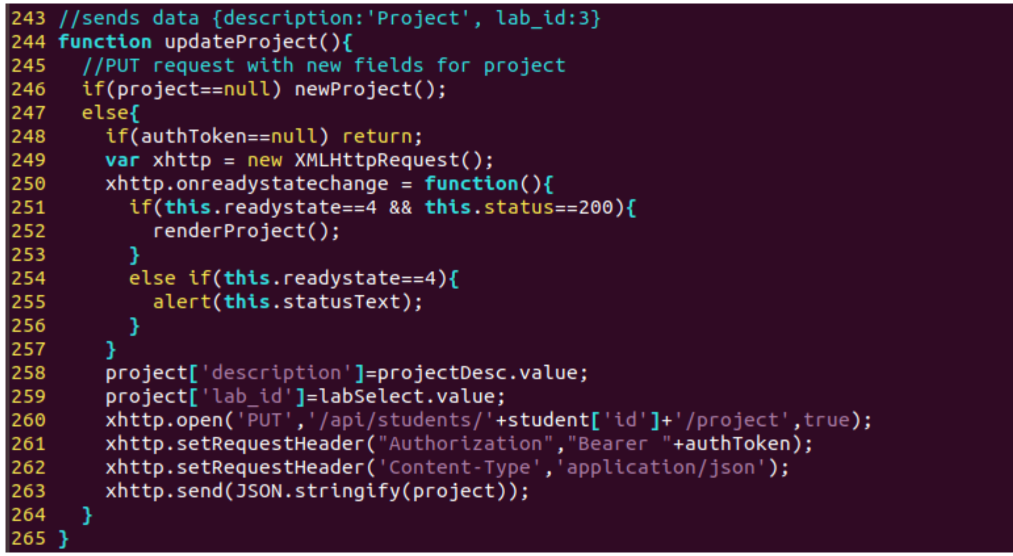
```javascript
///////////////HTML Rendering Functions/////////////
function setUp(){
  loginButton = document.getElementById('log');
  projectButton = document.getElementById('project');
  loginPanel = document.getElementById('login-panel');
  snum = document.getElementById('snum');
  pin = document.getElementById('pin');
  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

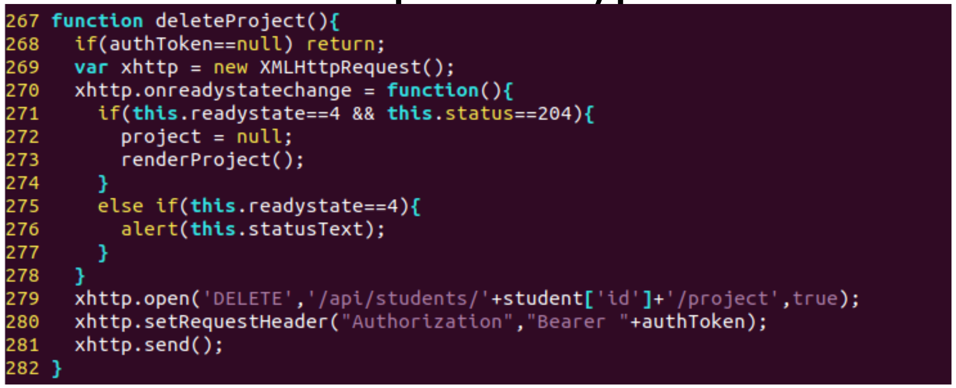
- To POST or PUT data we extract user entered data from the input elements and create javascript objects.
- We include the JSON as a parameter of the send function.
- We must set the content type to JSON so that the flask API will accept the data.
- To DELETE data we expect a different response type.



```javascript
//sends data {partnerNUmber:'19617810', description:'Project', lab_id:3}
function newProject(){
  //POST request with new fields for project
  if(authToken==null) return;
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function(){
    if(this.readystate==4 && this.status==201){
      getProject();
    }
    else if(this.readystate==4){
      alert(this.statusText);
    }
  }
  var project={};
  project['partnerNumber']=partnerNum.value;
  project['description']=projectDesc.value;
  project['lab_id']=labSelect.value;
  xhttp.open('POST','/api/students/'+student['id']+'/project',true);
  xhttp.setRequestHeader("Authorization","Bearer "+authToken);
  xhttp.setRequestHeader('Content-Type','application/json');
  xhttp.send(JSON.stringify(project));
}
```



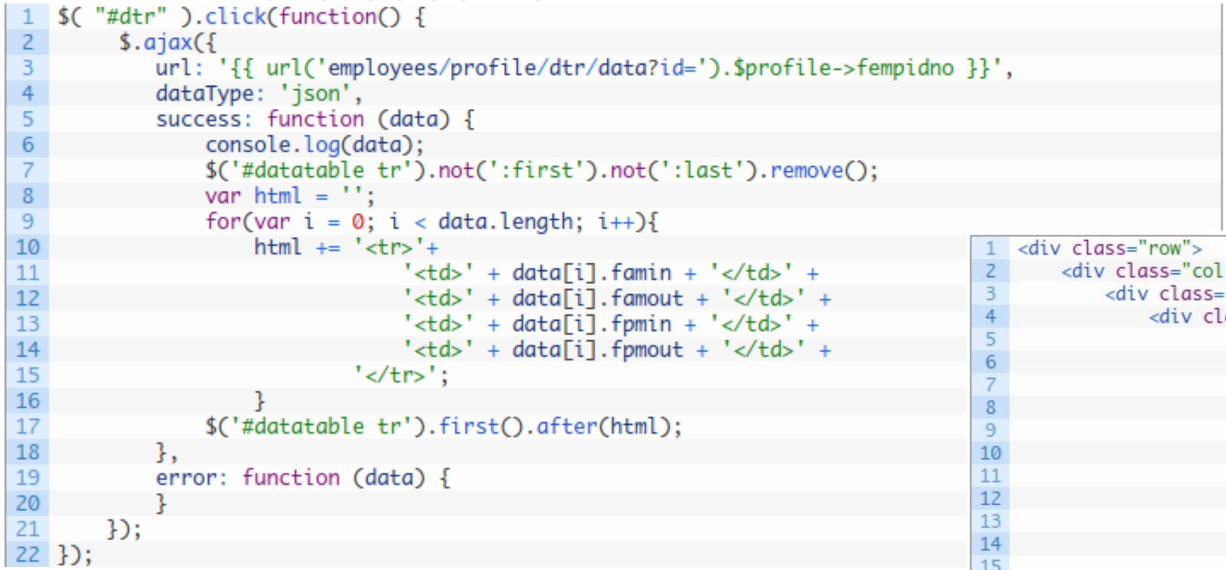
```javascript
//sends data {description:'Project', lab_id:3}
function updateProject(){
  //PUT request with new fields for project
  if(project==null) newProject();
  else{
    if(authToken==null) return;
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function(){
      if(this.readystate==4 && this.status==200){
        renderProject();
      }
      else if(this.readystate==4){
        alert(this.statusText);
      }
    }
    project['description']=projectDesc.value;
    project['lab_id']=labSelect.value;
    xhttp.open('PUT','/api/students/'+student['id']+'/project',true);
    xhttp.setRequestHeader("Authorization","Bearer "+authToken);
    xhttp.setRequestHeader('Content-Type','application/json');
    xhttp.send(JSON.stringify(project));
  }
}
```



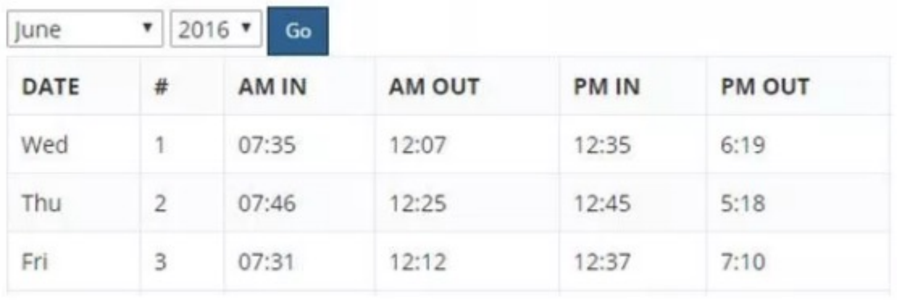
```javascript
function deleteProject(){
  if(authToken==null) return;
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function(){
    if(this.readystate==4 && this.status==204){
      project = null;
      renderProject();
    }
    else if(this.readystate==4){
      alert(this.statusText);
    }
  }
  xhttp.open('DELETE','/api/students/'+student['id']+'/project',true);
  xhttp.setRequestHeader("Authorization","Bearer "+authToken);
  xhttp.send();
}
```

Using jQuery

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

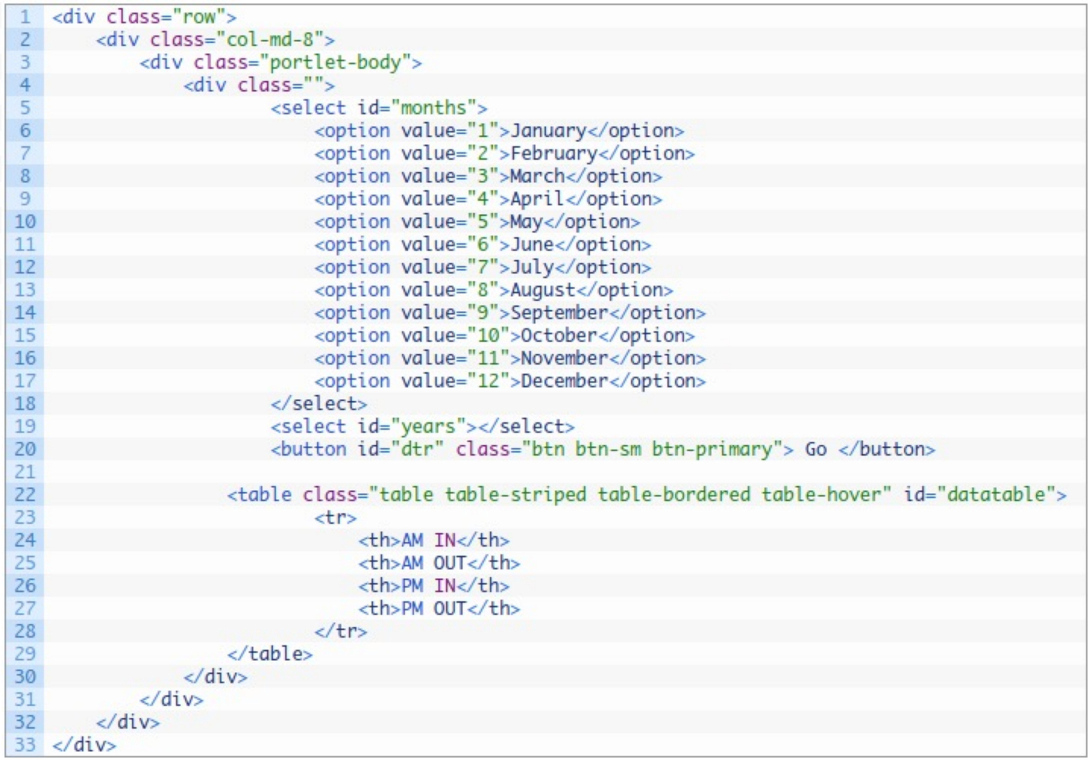


```
$( "#dtr" ).click(function() {
  $.ajax({
    url: '{{ url('employees/profile/dtr/data?id=').$profile->fempidno }}',
    dataType: 'json',
    success: function (data) {
      console.log(data);
      $('#datatable tr').not(':first').not(':last').remove();
      var html = '';
      for(var i = 0; 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>
```

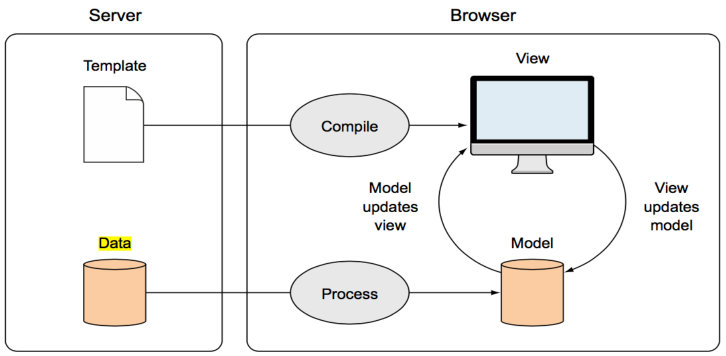
AngularJS

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

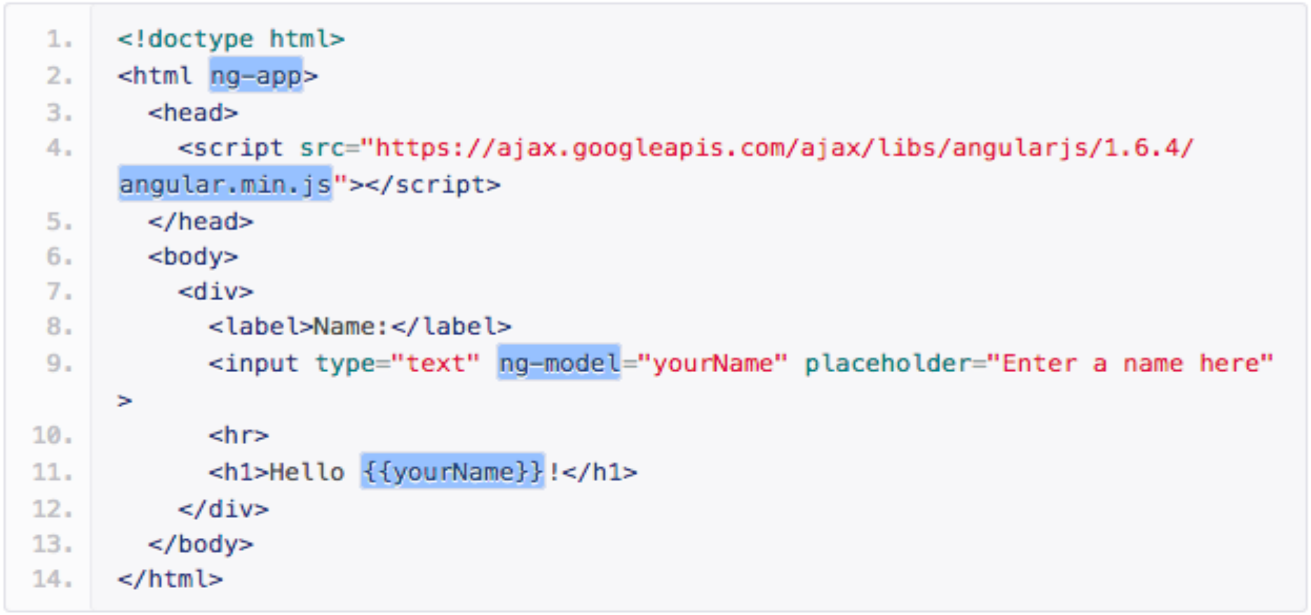
“Other frameworks deal with HTML’s shortcomings by either abstracting away HTML, CSS, and/or JavaScript or by providing an imperative way for manipulating the DOM. Neither of these address the root problem that HTML was not designed for dynamic views”.

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

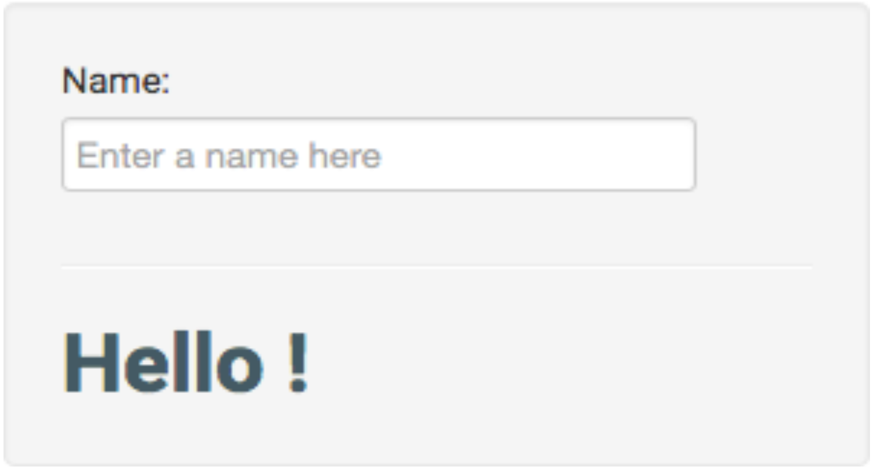
- Two-way Data Binding – Model as single source of truth
- Directives – Extend HTML
- MVC
- Dependency Injection
- Testing
- Deep Linking (Map URL to route Definition)
- Server-Side Communication



Data Binding



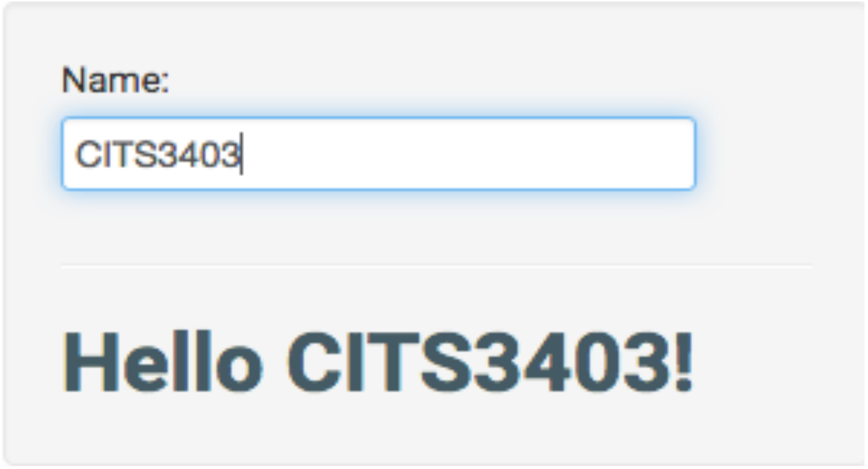
```
<!doctype html>
<html ng-app>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/
angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here"
>
      <hr>
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>
```



Name:

Enter a name here

Hello !



Name:

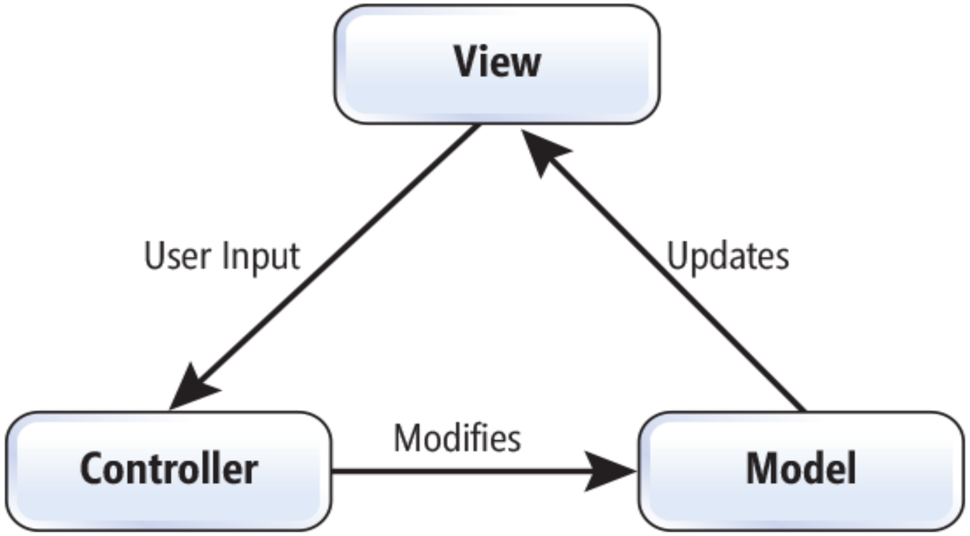
CITS3403

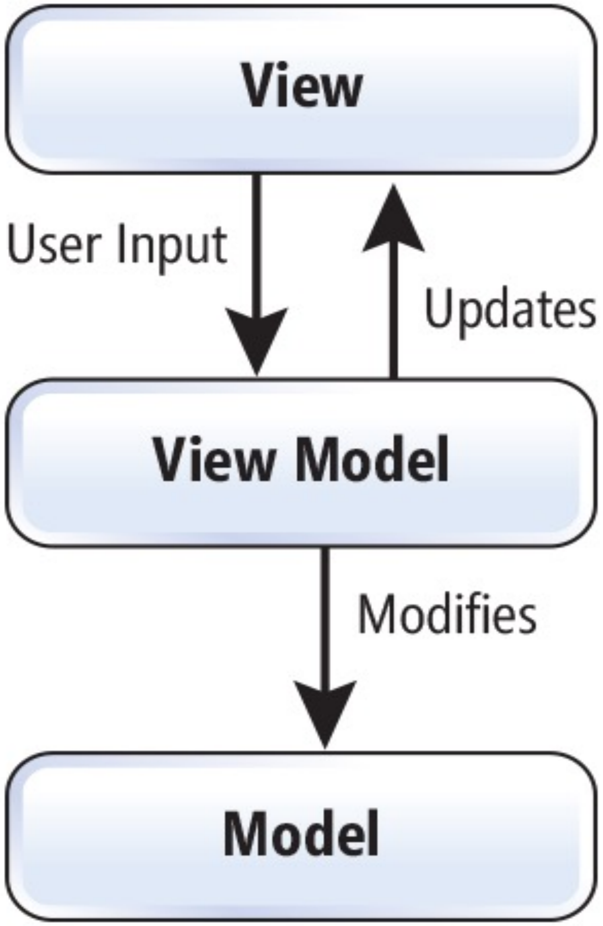
Hello CITS3403!

Angular Concepts

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