COMP721 Web Development

AUT UNIVERSITY

Week 11: Web Services and JavaScript APIs

Misc

- Assignment 2 due next week…

- SPEQ survey is online…

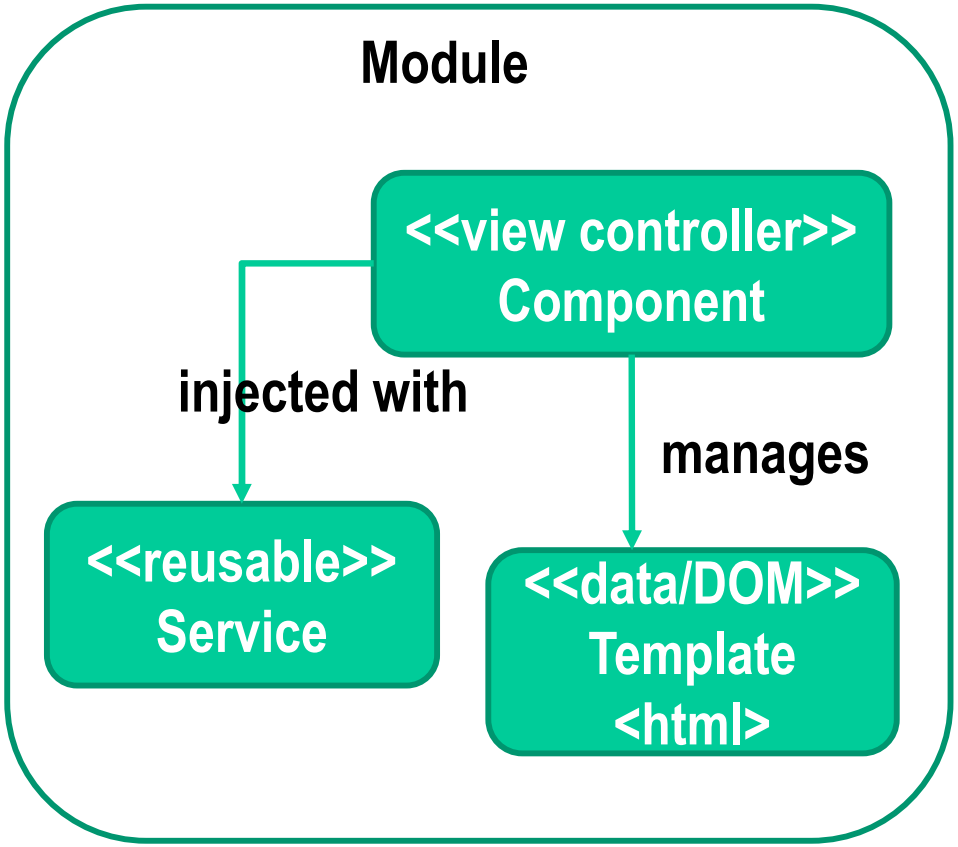
https://surveys.aut.ac.nz

- please spend a few minutes on it now or fill it afterwards…

Agenda

- Week 10 review
- What are web services
- Consuming web services
- Web services and Ajax
- Building RESTful web services using Node.js + MongoDB
- Using JavaScript APIs: Google map example

Week 10 Review: Angular framework



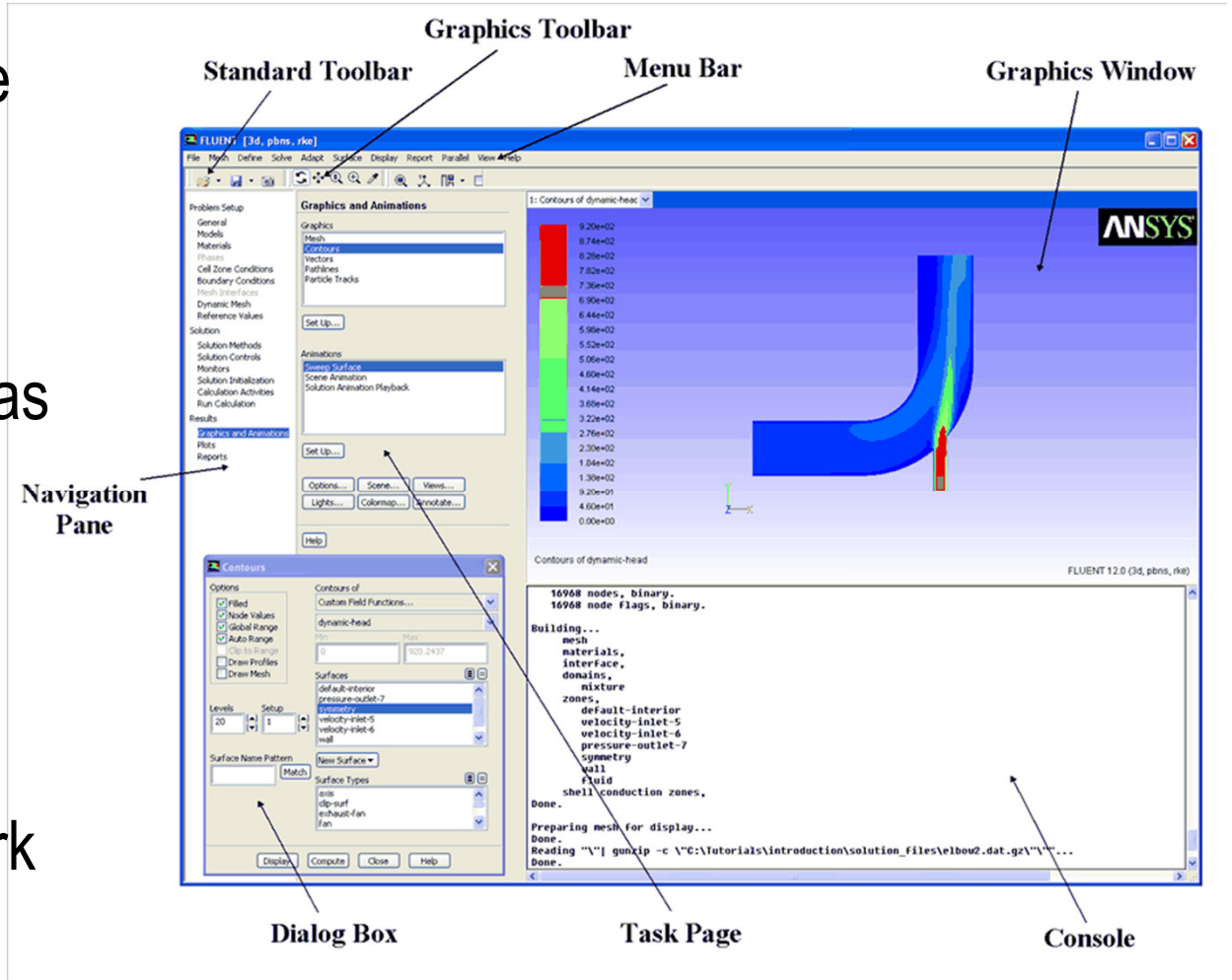
Angular is an OO framework

You write Angular applications by:

- composing **HTML templates/GUI objects/HTML element objects** with Angularized markup,
- writing **components** to manage those templates,
- adding application logic in **services (reusable)**, like business objects (order, shopping cart…)
- and boxing components and services in **modules**.

Design and structure your Angular project first

- Design the GUI
- A window could be a module
- Every GUI component is an Angular component
- Every Angular component has a view, which is the HTML template
- Put reusable functions into services (e.g., business service, data service, network service…)



Angular Key features

- two-way data binding: Two-way binding **binds an HTML Form input element to the property of an object**, so that user can update object properties through GUI (which means the property is editable)
- Interpolation binding: binds HTML content to JS variables
- Templates: HTML view is extended to contain instructions on how the model should be projected into the view (The HTML templates are parsed into the DOM), e.g.,
  - special markups/tags, loop rendering, conditional rendering, conditional styling, render two-way bound data…

Web Services

Note: The terms "Web services" and "Web APIs" are used interchangeably in the community

Using Others' Provided Functionality

- So far we have discussed building web-based systems where we have provided both the server-side and the client-side functionality
- Now we look at using services provided by others – web services
- Thus we need to look at systems that involve three levels of provision: services on third-party servers, services on our own server, and user interfaces on the client

Characteristics

- A Web Service is **accessible over the Web**.
- Web Services communicate using **platform-independent and language-neutral** Web protocols.
- A Web Service provides an **interface** that can be called from another program.
- A Web Service can be **registered and be located** through a Web Service Registry.
- Web Services support **loosely coupled** connections between systems.
- Web Services promote componentization of common functions.
- Web services ease your web programming effort – mostly developers consume rather than create Web Services

Consuming web services

Consuming a Web Service

- Weather API:
- https://openweathermap.org/api

Returned JSON string - openweathermap

http://api.openweathermap.org/data/2.5/weather?q=Auckland&APPID=2f4d83e3d50672cf2009fc34611903f3


```
{    "coord":{"lon":174.77,"lat":-36.85},
     "weather":[{"id":804,"main":"Clouds","description":
     "overcast clouds","icon":"04d"}],
     "base":"stations",
     "main":{"temp":291.15,"pressure":1012,"humidity":63
     ,"temp_min":291.15,"temp_max":291.15},
     "visibility":10000,
     "wind":{"speed":4.6,"deg":270},
     "clouds":{"all":88},
     "dt":1526779800,
     "sys":{"type":1,"id":8276,"message":0.0038,
     "country":"NZ","sunrise":1526757347,"sunset":
     1526793526},
     "id":2193733,
     "name":"Auckland",
     "cod":200
}
```

Returned JSON string – stormglass.io

- https://api.stormglass.io/v2/weather/point?lat=36.8509&lng=174.7645&params=waterTemperature&key=


```
{"hours":[{"time":"2022-05-23T00:00:00+00:00","waterTemperature":{"meto":15.28,"noaa":16.38,"sg":15.28}},{
{"meto":15.31,"noaa":16.57,"sg":15.31}},{"time":"2022-05-23T02:00:00+00:00","waterTemperature":{"meto":15.
23T03:00:00+00:00","waterTemperature":{"meto":15.36,"noaa":16.93,"sg":15.36}},{"time":"2022-05-23T04:00:00
{"time":"2022-05-23T05:00:00+00:00","waterTemperature":{"meto":15.42,"noaa":16.27,"sg":15.42}},{"time":"20
{"meto":15.44,"noaa":15.94,"sg":15.44}},{"time":"2022-05-23T07:00:00+00:00","waterTemperature":{"meto":15.
23T08:00:00+00:00","waterTemperature":{"meto":15.48,"noaa":15.67,"sg":15.48}},{"time":"2022-05-23T09:00:00
{"time":"2022-05-23T10:00:00+00:00","waterTemperature":{"meto":15.52,"noaa":15.54,"sg":15.52}},{"time":"20
{"meto":15.53,"noaa":15.55,"sg":15.53}},{"time":"2022-05-23T12:00:00+00:00","waterTemperature":{"meto":15.
23T13:00:00+00:00","waterTemperature":{"meto":15.56,"noaa":15.55,"sg":15.56}},{"time":"2022-05-23T14:00:00
{"time":"2022-05-23T15:00:00+00:00","waterTemperature":{"meto":15.59,"noaa":15.55,"sg":15.59}},{"time":"20
{"meto":15.6,"noaa":15.62,"sg":15.6}},{"time":"2022-05-23T17:00:00+00:00","waterTemperature":{"meto":15.61
23T18:00:00+00:00","waterTemperature":{"meto":15.62,"noaa":15.75,"sg":15.62}},{"time":"2022-05-23T19:00:00
{"time":"2022-05-23T20:00:00+00:00","waterTemperature":{"meto":15.62,"noaa":15.76,"sg":15.62}},{"time":"20
{"meto":15.62,"noaa":15.77,"sg":15.62}},{"time":"2022-05-23T22:00:00+00:00","waterTemperature":{"meto":15.
23T23:00:00+00:00","waterTemperature":{"meto":15.62,"noaa":15.83,"sg":15.62}},{"time":"2022-05-24T00:00:00
{"time":"2022-05-24T01:00:00+00:00","waterTemperature":{"meto":15.02,"noaa":15.87,"sg":15.02}},{"time":"20
{"meto":15.03,"noaa":15.88,"sg":15.03}},{"time":"2022-05-24T03:00:00+00:00","waterTemperature":{"meto":15.
24T04:00:00+00:00","waterTemperature":{"meto":15.04,"noaa":15.88,"sg":15.04}},{"time":"2022-05-24T05:00:00
```

Making Use of a Web Service

- Whilst what we did on the last few slides enables us to get weather info, it is not really very useful
- What is useful is that by invoking the web service we can get a chunk of JSON data returned, and we can then manipulate this to include in an application
- So we need a way to integrate a “call” to a web service within a web-hosted program, and then arrange for manipulation of the data returned.

The SOAP Approach

- A SOAP (Simple Object Access Protocol) XML document is used for request/response
  - A SOAP document contains a <SOAP:Envelope> element
  - The <SOAP:Envelope> specifies several namespaces for XML Schema, instance and SOAP
  - The <SOAP:Envelope> contains a <SOAP:Header> element (optional) and a <SOAP:Body> element
- In a request SOAP document, the method to be called together with parameters are constructed within <SOAP:Body>
- In a response SOAP document, the response and the returned result are constructed within <SOAP:Body>
- Extra work is needed to construct and encode the SOAP documents

The REST Approach

- REST: REpresentational State Transfer
- Use either HTTP-GET or HTTP-POST, e.g., http://www.webservicex.net/StockQuote.asmx/GetQuote?symbol=GOOG
- It's just like what we have already been doing when "calling" a PHP file on the server
  - Send the method name ()
  - Send the parameters (q=Auckland)
    - Pros: gets message stored in JSON, uses HTTP (GET, POST) to transfer, uses URIs for identification, can use HTTP authentication for security, easy to use (with AJAX), easy to create mashups
- Cons: less secure than SOAP, the length of URI is limited while using GET

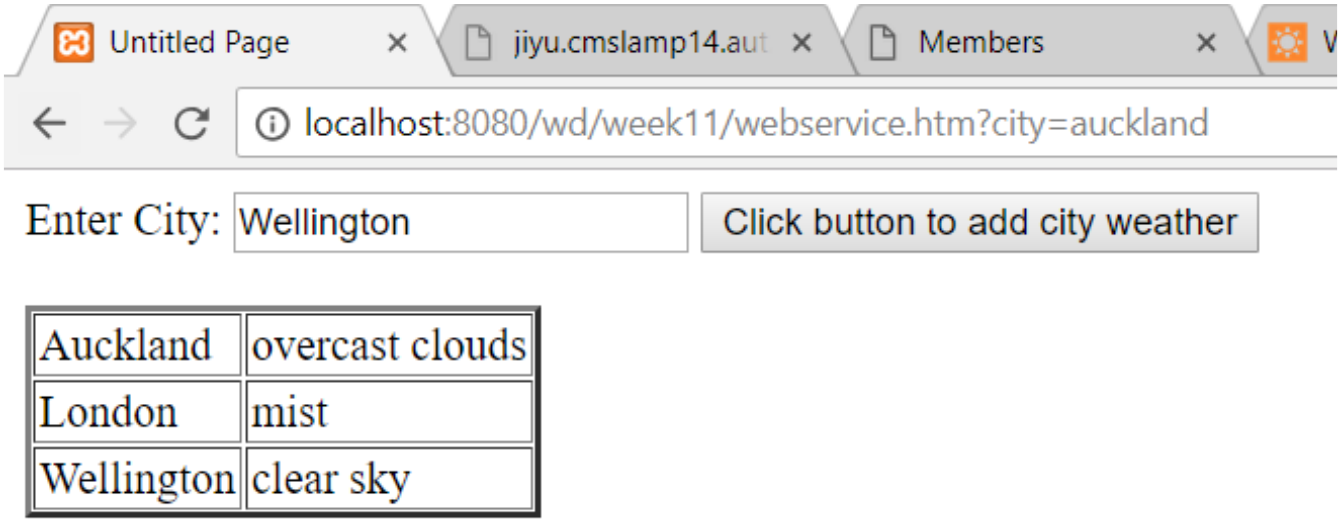
Web services and Ajax

Four-Step Process of a Web Service

- The client (a browser of a certain type) calls the web service over a protocol (normally HTTP, could be SMTP or HTTPS though)
- The client selects a method and sends the method to the server with instructions - parameters and methods of transmission (HTTP-GET, HTTP-POST, or SOAP)
- The server returns value (in XML or JSON) and/or acknowledgement
- The client gets the result and acts on the received information.

Example: Weather Web Service

- Create an Ajax application to get city weather
- 3 files (plus Web service) working together
  - Webservice.htm : (drives the application)
  - Webservice.js : (client-side processing of data)
  - ApplicationProxy.php : (server-side script to access the web service and forward data to the client)



Consuming a Web Service via a Proxy

webservice.htm

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Untitled Page</title>
<script type="text/javascript" src=”xhr.js"></script>
<script type="text/javascript" src="webservice.js"></script>
</head>
<body>
<form id="form1" name="form1">
Enter Symbol: <input type="text" name=“city" id=“city" />
<input type="button" onclick="sendData()" value="Click button to add quote" />
<br /><br />
<table id="table1">
</table>
</form>
</body>
</html>
```

JSON will be formatted and displayed here

webservice.js

```
var xhr = null;
xhr = createRequest();
function sendData()
{ xhr.abort(); // abort any prior activity on xhr
  var url = "applicationproxy.php?symbol=" + document.form1.city.value;
  xhr.open("GET", url, true);
  xhr.onreadystatechange = getData;
  xhr.send(null);
}
```

This is new. It just ensures that since it uses a globally accessible XHR variable, any previous computation using that variable is aborted.

Callback function

ApplicationProxy.php

- Our example

```
<?php
  $url =
"http://api.openweathermap.org/data/2.5/weather?APPID=2f4d83e3d50672cf
2009fc34611903f3&q=";
 $qs = $_GET["city"];
 $url = $url.$qs;
 $curl = curl_init();
  curl_setopt($curl, CURLOPT_URL, $url);
  curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
  curl_setopt($curl, CURLOPT_PROXY, 'cache.aut.ac.nz:3128');
  $pageContent = curl_exec($curl);
  curl_close($curl);
echo $pageContent;
?>
```

Curl (could be useful when using an http proxy server)

- Libcurl (the multiprotocol file transfer library)
  - library that allows you to connect and communicate to many different types of servers with many different types of protocols
  - libcurl currently supports the http, https, ftp, gopher, telnet, dict, file, and ldap protocols. libcurl also supports HTTPS certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, cookies, and user+password authentication.
- PHP supports libcurl (& it is installed on mercury)
  - http://www.php.net/manual/en/ref.curl.php

webservice.js callback function

```
function getData(){
                if ((xmlhttp.readyState == 4) &&( xmlhttp.status == 200)){
                 alert(xmlhttp.responseText); // easier for debugging
……
//see BB example code for details
```

webservice.js (Cont'd)

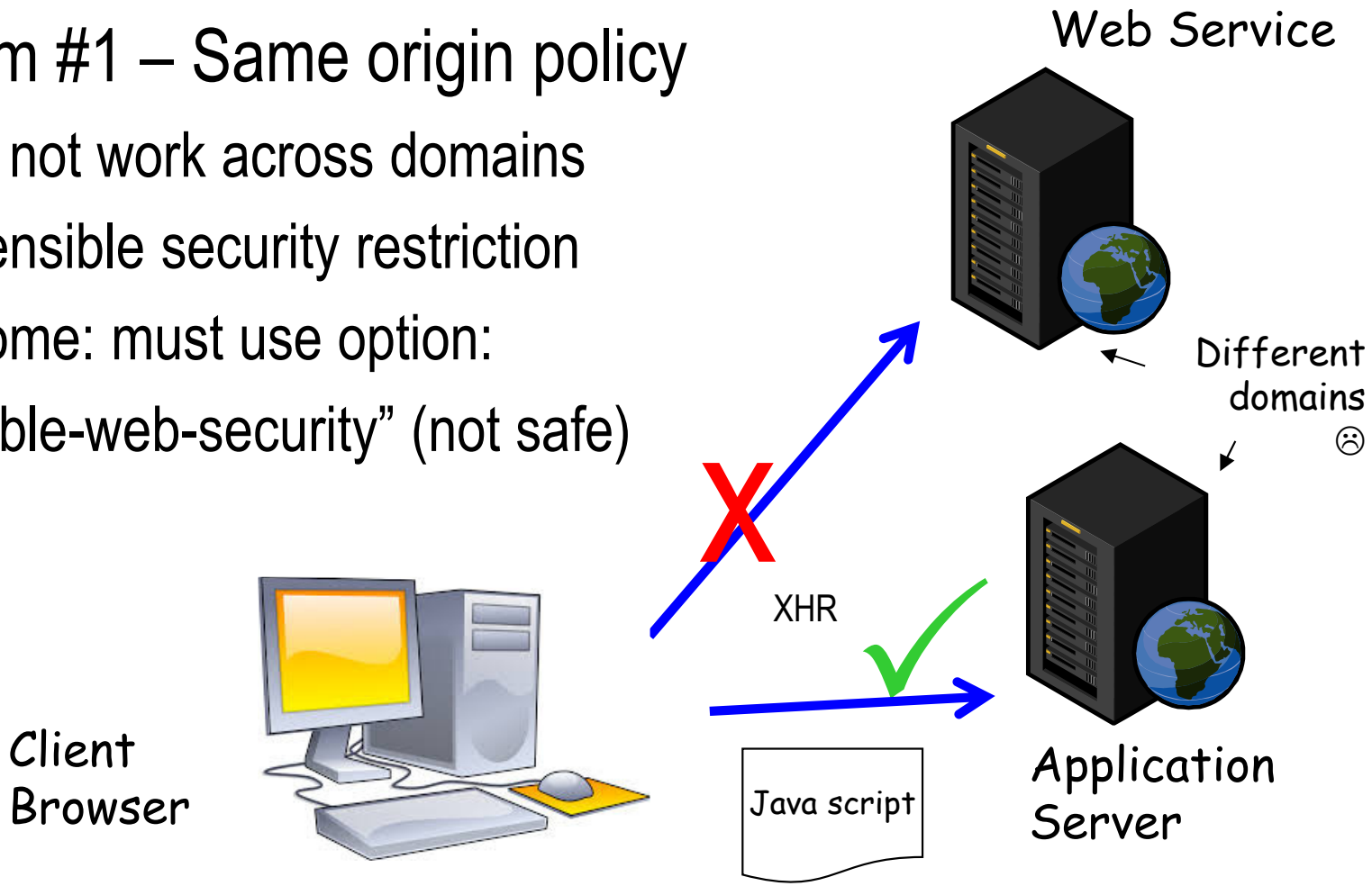
```
…
var table = document.getElementById("table1");
    var row = table.insertRow(table.rows.length);
    var cell1 = row.insertCell(row.cells.length);
    cell1.appendChild(getText("Name",xmlobject));
    var cell2 = row.insertCell(row.cells.length);
    cell2.appendChild(getText("Last",xmlobject));
    var cell3 = row.insertCell(row.cells.length);
    cell3.appendChild(getText("Date",xmlobject));
    table.setAttribute("border", "2");
  }
}
function getText(tagName, xmlobject)
{ var tags = xmlobject.getElementsByTagName(tagName);
  var txtNode = null;
  if (window.ActiveXObject) { txtNode = document.createTextNode(tags[0].firstChild.text);}
  else { txtNode = document.createTextNode(tags[0].firstChild.textContent);}
  return txtNode; }
```

Format data retrieved from XML and add to the end of the table in the HTML doc

Integrating a Web Service into an Application using Ajax and REST

- Passing web service's endpoint in the *open* method of an *XHR* object
  - Works in the same domain but …
- Problem #1 – Same origin policy
  - May not work across domains
    - sensible security restriction
  - Chrome: must use option:

" --disable-web-security" (not safe)



Same Origin Policy

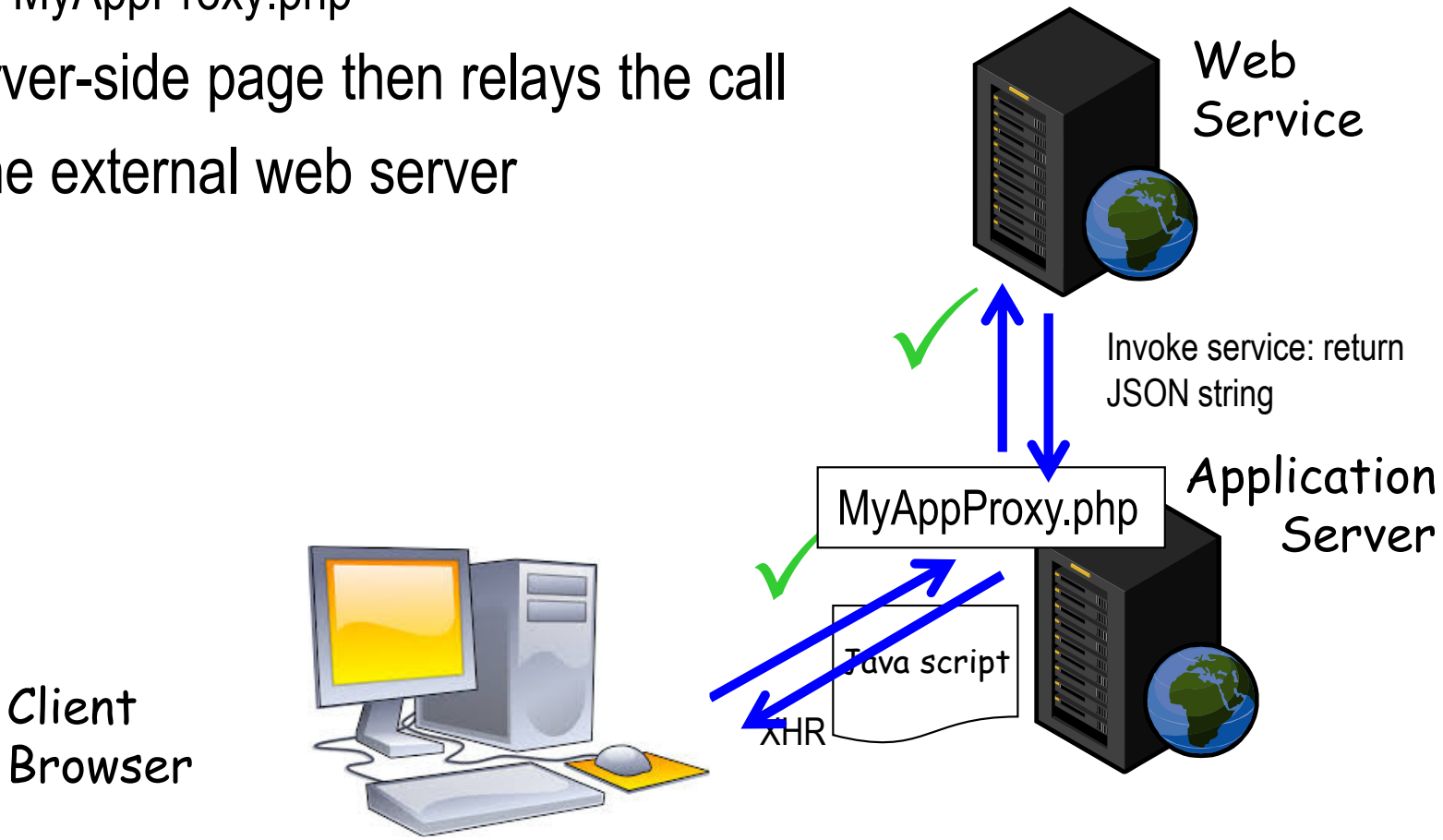
- http://www.mozilla.org/projects/security/components/same-origin.html
- Mozilla considers two pages to have the same origin if *protocol*, *port* (if given), and *host* (*domain*) are the same.
- Example: Javascript from http://store.company.com/dir/page.html.

URL	Outcome	Reason
http://store.company.com/dir2/other.html	Success	
http://store.company.com/dir/inner/another.html	Success	
https://store.company.com/secure.html	Failure	Different protocol
http://store.company.com:81/dir/etc.html	Failure	Different port
http://news.company.com/dir/other.html	Failure	Different host

Footnote: HTML5 can do cross domain messaging. (postMessage)

Main Solution to Same Source Restriction

- Create an "Application Proxy"
  - Create a proxy page on the server
  - The XHR object calls the page on the server
    - E.g. MyAppProxy.php
  - This server-side page then relays the call on to the external web server

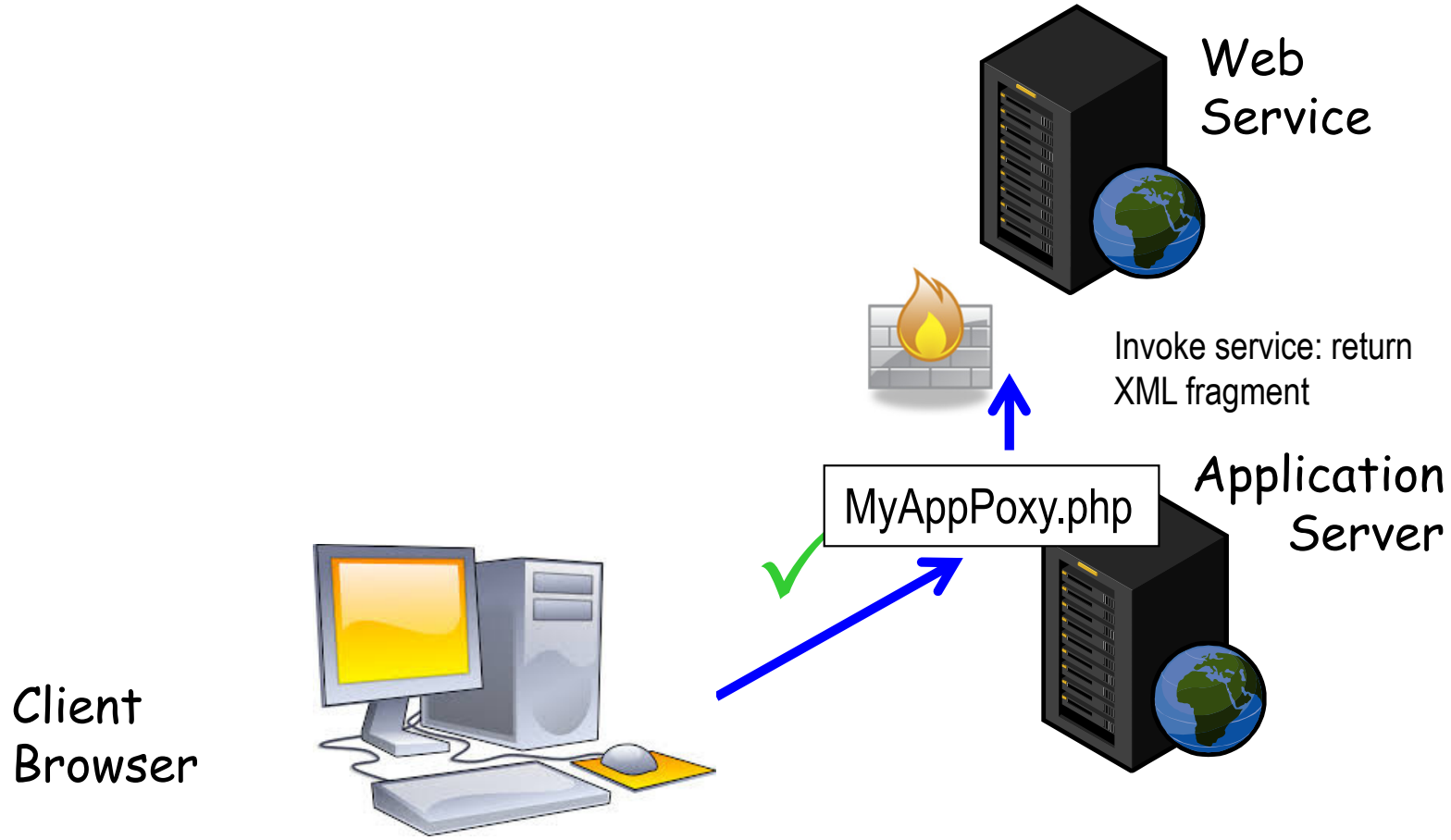


Main Solution to Same Source Restriction

- The main issue is that for security reasons, the client is not permitted to directly access a resource that is not from the same domain as the client-side page came from – this is a universal restriction of the WWW
- But there is no universal restriction placed on what a server can access – although, as we will see, a server's administrator can provide protection through use of a firewall, for example
- Hence the programming pattern is for the client to get the server to go off to the external resource, get data as required, and then relay it back to the client.

An aside: Web services and Firewall

- Web services can be transmitted over HTTP. This makes it easy for Web services to bypass network firewalls.



Building RESTful web services using Node.js + MongoDB

RESTful API design

- POST is used to send data to a server—Create
- GET is used to fetch data from a server—Read
- PUT is used to send and update data—Update
- DELETE is used to delete data—Delete

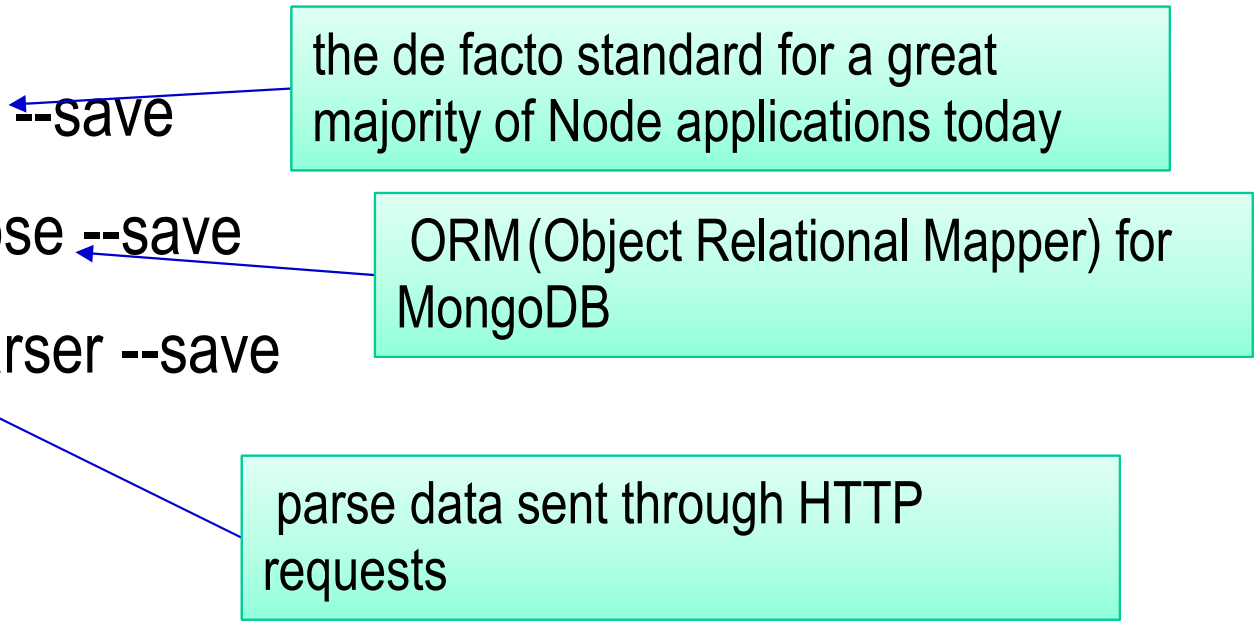
The key is to map/route an URL to a function implemented, which means ROUTING is a major concern in Restful API/WS design

Routing example

- Routing design for a user management module
  - HTTP GET Root request ‘/’ goes to index.html
  - HTTP POST ‘/users’ request goes to user.createUsers()
  - HTTP GET ‘/users’ request goes to user.seeResults()
  - HTTP DELETE ‘users/<id>’ goes to user.delete()

A user management app with CRUD functions

- First create a directory for this app in terminal
- Go inside the dir and Install modules for this app:
  - npm init
  - npm install express --save
  - npm install mongoose --save
  - npm install body-parser --save



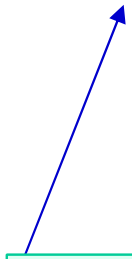
db.js: managing connection to MongoDB

```
// db.js
var mongoose = require('mongoose');

//use cloud DB
//mongoose.connect('
mongodb://admin:admin@cluster0-shard-00-00-syy3n.mongodb.net:270
17,cluster0-shard-00-01-syy3n.mongodb.net:27017,cluster0-shard-0
0-02-syy3n.mongodb.net:27017/test?ssl=true&replicaSet=Cluster0-s
hard-0&authSource=admin');

//use local DB
mongoose.connect('mongodb://localhost/');

console.log('MongoDB connected');
```

Users.js

- First create the user data model/schema using Mongoose

```
// User.js
var mongoose = require('mongoose');
var UserSchema = new mongoose.Schema({
    name: String,
    age: Number,
    email: String
});
var User = mongoose.model('User', UserSchema);
```



Collection ‘user’ in MongoDB

User.js – cont'd

- Then define a function to create a user document on MongoDB
- Note that the function parameter: req is HTTP request, res is HTTP response

```
module.exports = {
  createUsers: function (req, res) {
    var person = req.body;
    new User({ name: person.name, age: person.age, email:
    person.email })
      .save(function (err) {
        if (err) {
          res.status(504);
          res.end(err);
        } else {
          console.log('user saved');
          res.end();
        }
      });
  },
```

User.js – cont'd

- Next define a function to retrieve all users
- Note that the '`next`' param is the callback function that can be invoked after executing the current function; will discuss in detail in the next slides…

```
seeResults: function (req, res, next) {
  User.find({}, function (err, docs) {
    if (err) {
      res.status(504);
      res.end(err);
    } else {
      for (var i = 0; i < docs.length; i++) {
       console.log('user:', docs[i].name);
      }
      res.end(JSON.stringify(docs));
    }
  });
},
```

User.js – cont'd

- Finally define a function to delete a users
- This function takes the user id as a parameter

```
delete: function( req, res, next) {
  console.log(req.params.id);
  User.find({ _id: req.params.id}, function(err) {
    if(err) {
      req.status(504);
      req.end();
      console.log(err);
    }
  }).remove(function (err) {
    console.log(err);
    if (err) {
      res.end(err);
    } else {
      res.end();
    }
  });
```

Routing using the Express module

- Routing refers to how an application's endpoints (URIs) respond to client requests
- A **route** associate an URI/URL with one or more functions defined in the backend
- Node.js Express module is usually used to manage routing

app.js

- First we require body-parser module, and the self-defined db.js and user.js modules
- We also require the express module and create an instance called app of it

```
// app.js
const bodyParser= require('body-parser');

var db = require('./db');
var user = require('./user');

var express = require('express');
var app = express();
```

app.js – cont'd

- Next we process the request body: any functions included in `app.use()` will be executed every time the app receives a request

```
app.use(bodyParser.urlencoded({extended: true}));
app.use(bodyParser.json());
```

app.js – cont'd

- Next we define the routes for URIs:
  - **HTTP GET Root request '/' goes to index.html**
  - **HTTP POST '/users' request goes to user.createUsers()**
  - **HTTP GET '/users' request goes to user.seeResults()**
  - **HTTP DELETE 'users/<id>' goes to user.delete()**

```
app.get('/', function (req, res) {
  res.sendFile(__dirname + '/index.html');
});

app.post('/users', user.createUsers);
app.get('/users', user.seeResults);
app.delete('/users/:id', user.delete);
```

About ‘next’

- An URI can be mapped to multiple functions, so we can use next() to chain these functions
- In the following example, if next() is not invoked in the first function, then the next function will not be invoked…

```
var app = require("express")();

app.get("/", function(httpRequest, httpResponse, next){
    httpResponse.write("Hello");
    next(); //remove this and see what happens
});

app.get("/", function(httpRequest, httpResponse, next){
    httpResponse.write(" World !!!");
    httpResponse.end();
});

app.listen(8080);
```

app.js – cont'd

- Finally we let the server listen on port 3000

```
app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

- To launch the server, type the following in the terminal

```
node app.js
```

Start mongo db server (needs admin permission):

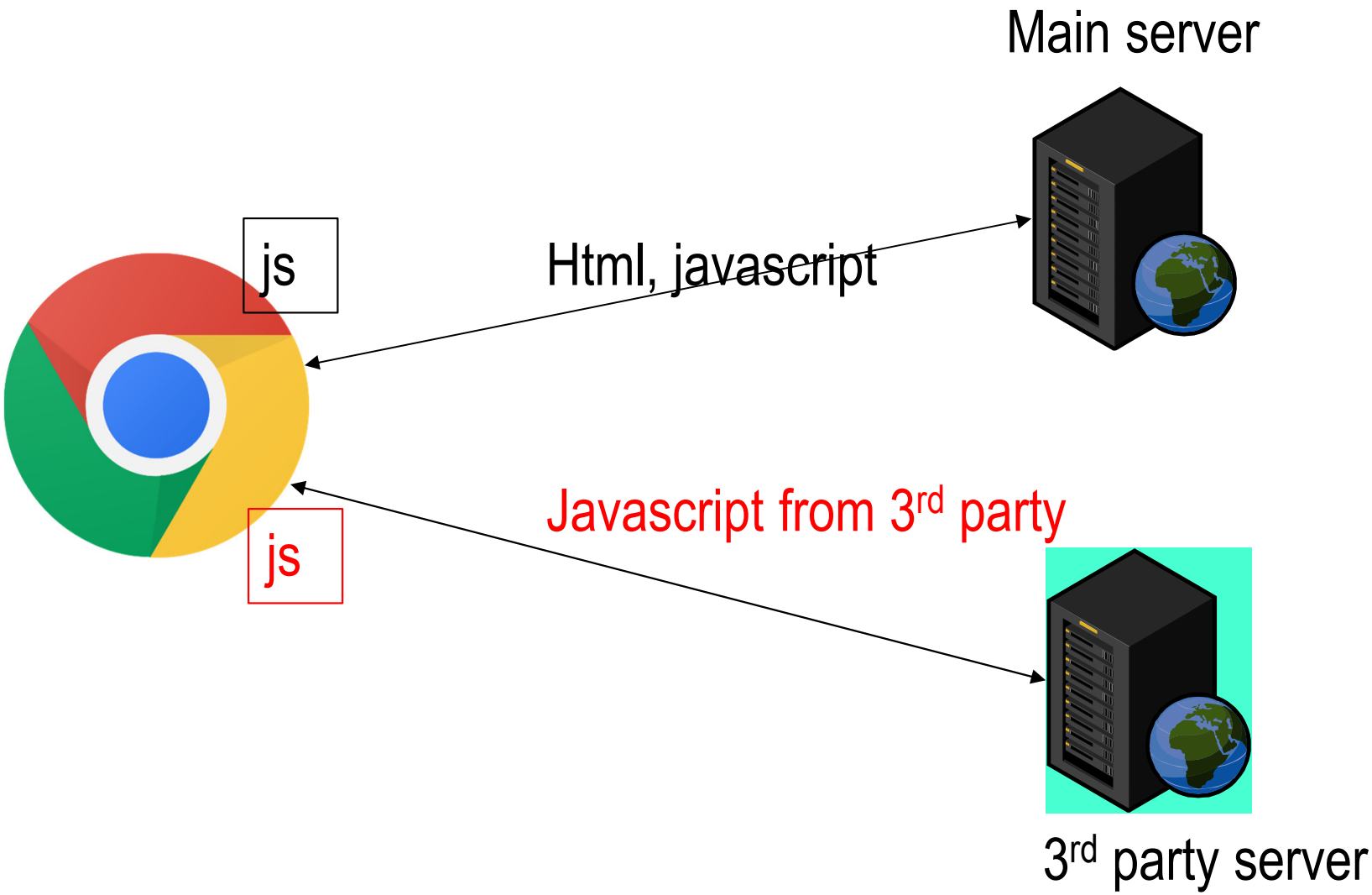
- In command line enter C:\Program Files\MongoDB\Server\5.0\bin
- Run: mongod -f .\mongod.cfg

Interact with the server

- Use the ‘postman’ tool (https://www.getpostman.com/)
- Use a simple HTML interface (to send GET, POST requests)

Using JavaScript APIs
example: google maps API

Why JS APIs do not break the “Same Origin Policy”?



Javascript from 3rd party is not allowed to access the main server

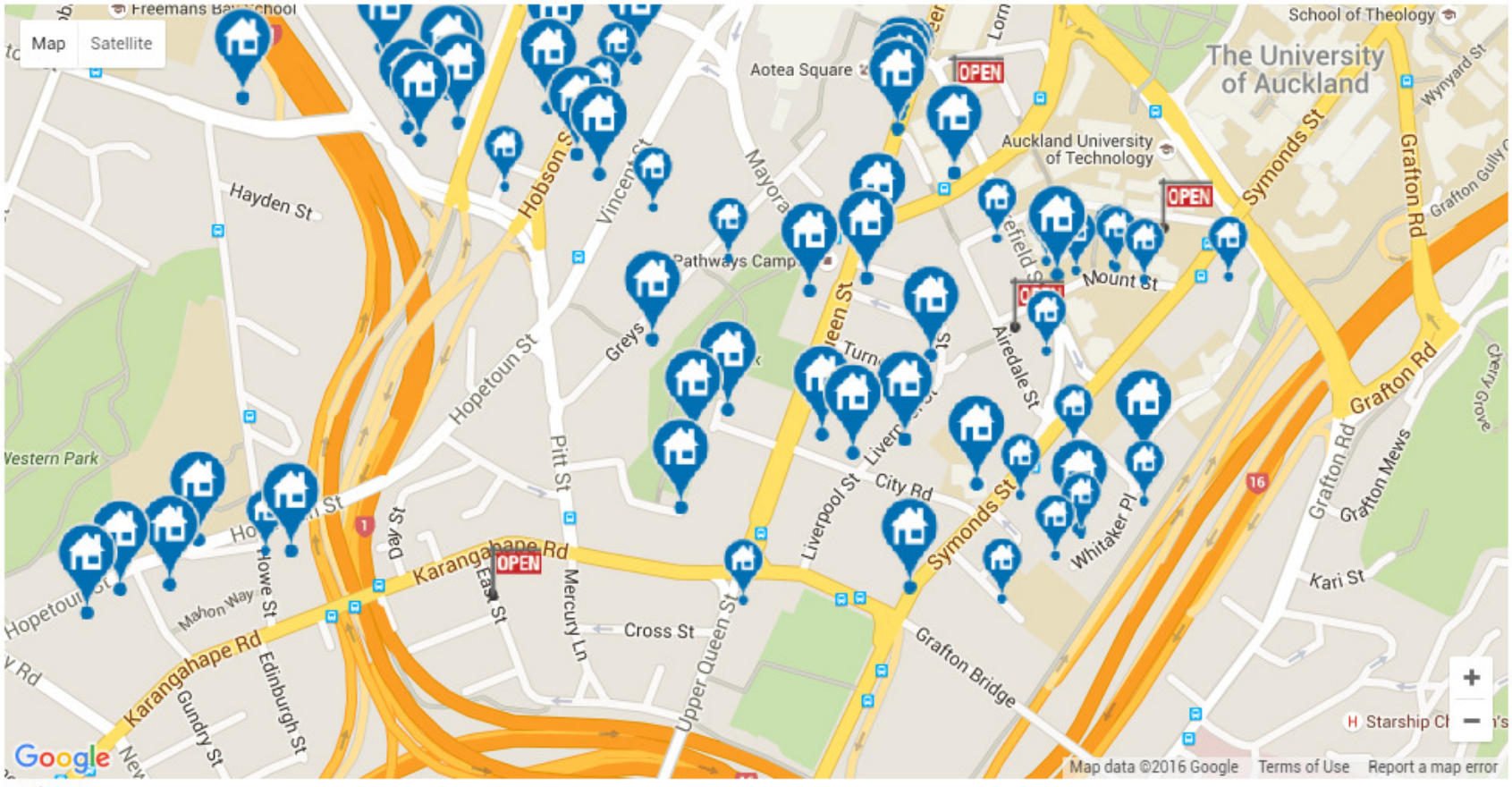
Most Common API Examples

Registry: www.programmableweb.com

- Facebook, LinkedIn
- Flickr (www.flickr.com/services)
- YouTube (www.youtube.com/dev)
- Google Maps (www.google.com/apis/maps)

Example: A Mashup web app that combines your own data and Google Maps data

- In widespread commercial use: e.g. Real Estate



Index.html

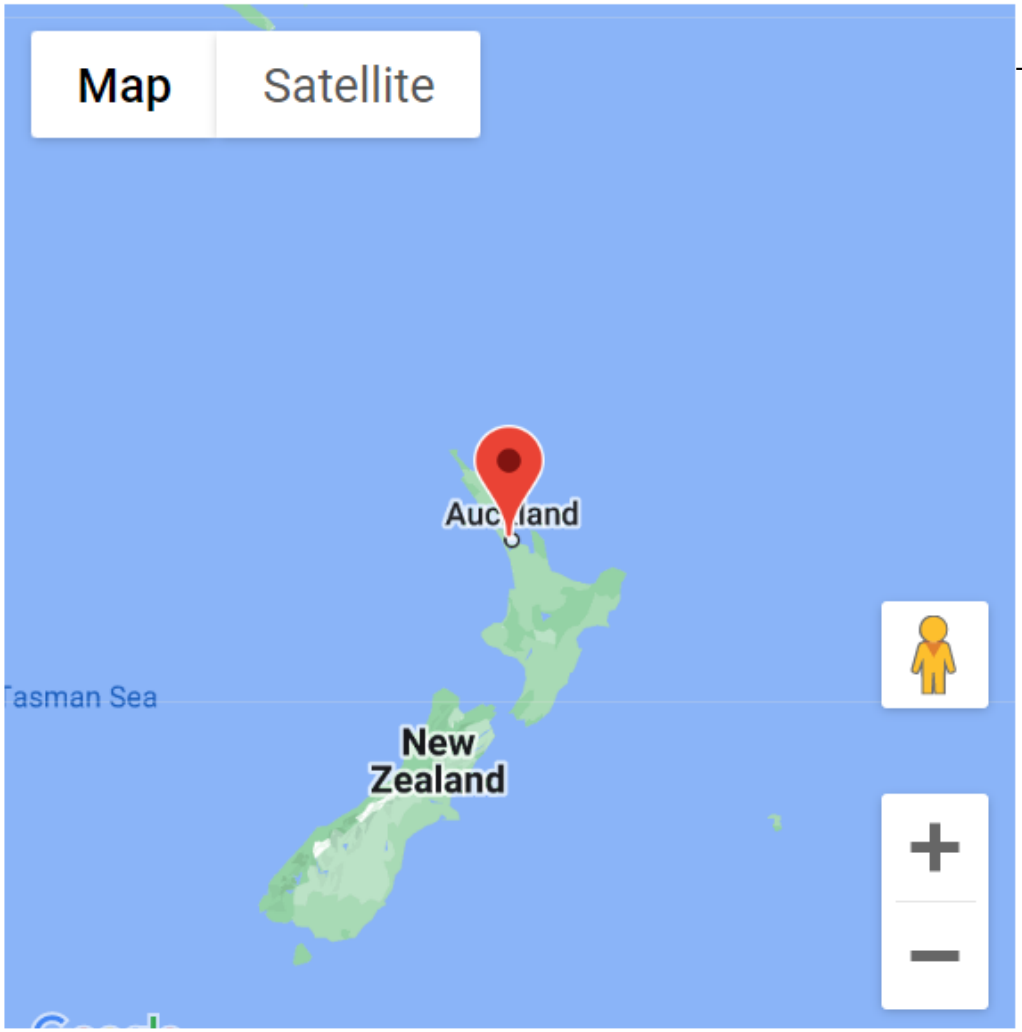
```
<html>
  <head>
    <title>Add Map</title>
    <script src="https://polyfill.io/v3/polyfill.min.js?features=default"></script>
    <link rel="stylesheet" type="text/css" href="./style.css" />
    <script type="module" src="./index.ts"></script>
  </head>
  <body>
    <h3>My Google Maps Demo</h3>
    <div id="map"></div>
<script
      src="https://maps.googleapis.com/maps/api/js?key=AIzaSyB41DRUbKWJHPxaFjMAwdrzWzbVKartNGg&callback=initMap&v=weekly"
      defer></script></body></html>
```

Initialize the map and put a marker in Auckland

```
// Initialize and add the map
function initMap(): void {
  // The location of Auckland
  const auckland = { lat: -36.8, lng: 174.7 };
  // The map, centered at Auckland
  const map = new google.maps.Map(
    document.getElementById("map") as HTMLElement,
    {
      zoom: 4,
      center: auckland,
    }
  );

  // The marker, positioned at
  const marker = new google.maps.Marker({
    position: auckland,
    map: map,
  });
```

```
declare global {
  interface Window {
    initMap: () => void;
  }
}
window.initMap = initMap;
export {};
```

My Google Maps Demo



Test run: https://stackblitz.com/github/googlemaps/js-samples/tree/sample-add-map?file=index.ts