

# **JavaScript Frameworks**

# Outline

- Node.js
- Angular

# Node.js

1. Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.
2. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.
3. Node.js allows the creation of Web servers and networking tools using JavaScript and a collection of "modules" that handle various core functionality.
4. Modules handle file system I/O, networking (DNS, HTTP, TCP, TLS/SSL, or UDP), binary data (buffers), cryptography functions, data streams and other core functions.



# Basic functionality

## HTTP

- To use the HTTP server and client one must **require('http')**.
- The HTTP interfaces in Node.js are designed to support many features of the protocol which have been traditionally difficult to use. In particular, large, possibly chunk-encoded, messages. The interface is careful to never buffer entire requests or responses--the user is able to stream data.

### **http.createServer([requestListener])**

- Returns a new instance of `http.Server`.
- The `requestListener` is a function which is automatically added to the 'request' event.

### **http.request(options[, callback])**

- Node.js maintains several connections per server to make HTTP requests. This function allows one to transparently issue requests.
- `options` can be an object or a string. If `options` is a string, it is automatically parsed with `url.parse()`.

### **http.get(options[, callback])**

- Since most requests are GET requests without bodies, Node.js provides this convenience method. The only difference between this method and `http.request()` is that it sets the method to GET and calls `req.end()` automatically.

# Basic functionality

## File System

- File I/O is provided by simple wrappers around standard POSIX functions. To use this module do **require('fs')**. All the methods have asynchronous and synchronous forms..

### **fs.readFile(file[, options], callback)**

- Asynchronously reads the entire contents of a file.
- The callback is passed two arguments (err, data), where data is the contents of the file.
- If no encoding is specified, then the raw buffer is returned.

### **fs.readFileSync(file[, options])**

- Synchronous version of fs.readFile. Returns the contents of the file.
- If the encoding option is specified then this function returns a string. Otherwise it returns a buffer.

.

# Basic functionality

## Buffer

- Prior to the introduction of TypedArray in ECMAScript 2015 (ES6), the JavaScript language had no mechanism for reading or manipulating streams of binary data. The Buffer class was introduced as part of the Node.js API to make it possible to interact with octet streams in the context of things like TCP streams and file system operations.
- The Buffer class is a **global** within Node.js, making it unlikely that one would need to ever use `require('buffer')`.

## Buffers and Character Encodings

- Buffers are commonly used to represent sequences of encoded characters such as UTF8, UCS2, Base64 or even Hex-encoded data. It is possible to convert back and forth between Buffers and ordinary JavaScript string objects by using an explicit encoding method.

# Example usage – Buffer Class

buffer.js:

```
const buf = new Buffer('hello world', 'ascii');

console.log(buf.toString('hex'));
// prints: 68656c6c6f20776f726c64
console.log(buf.toString('base64'));
// prints: aGVsbG8gd29ybGQ=
```

# Example usage – ‘fs’Read File

fs-readFile.js:

```
var fs = require('fs');
fs.readFile('./intro.txt', function (err, data) {
  if (err) throw err;
  console.log(data);
});
```

Intro.txt:

JsApp.US is a hosting platform for node.js applications.  
It is setup to be a platform to coddle to quick, weekend hack like projects.

Run command:

\$ nodejs fs-readFile.js

```
ubuntu@ip-172-31-12-48:~/nodejs$ nodejs fs-readFile.js
CSCI571 focuses on the phenomenon known as the World Wide Web (WWW or Web).
It's focus is to present many of the core technologies that the Web is based upon.
```



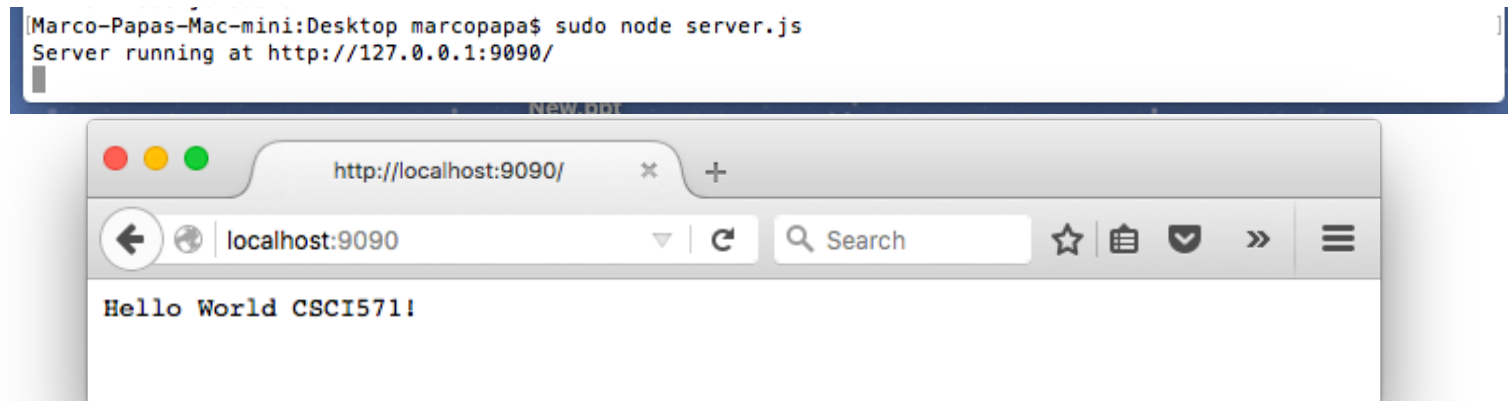
# Example usage – ‘http’ Run Web Server

server.js:

```
const http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello World CSCI571!\n');
}).listen(9090, '127.0.0.1', function () {
  console.log(`Server running at http://localhost:9090/`);});
```

Run command:

\$ sudo node server.js



# Node.js Modules

- **npm**: package manager for JavaScript.  
The **npm** command-line tool is bundled with Node.js. If you have it installed, then you already have **npm** too. (see <https://www.npmjs.com>)
- **nodemon**: monitor script for use during development of a Node.js app. Will watch files in the directory in which nodemon was started. If any files change, it will restart your node app. (see <https://nodemon.io>)
- **jshint**: a static analysis tool to detect errors and potential problems in JavaScript code and to enforce your team's coding conventions. (see <http://jshint.com>)

# Node.js Modules (cont'd)

- **express:** a fast, minimalist web framework. It provides small, robust tooling for HTTP servers, making it a great solution for single page applications, web sites, hybrids, or public HTTP APIs. (see <https://expressjs.com>)
- **cors:** a Node.js package for providing a Connect/Express middleware that can be used to enable CORS with various options. (see <https://github.com/expressjs/cors>)
- **body-parser:** body-parser extracts the entire body portion of an incoming request stream and exposes it on req.body as something easier to interface with. (see <https://github.com/expressjs/body-parser>)
- **request:** simplified HTTP request client. It supports HTTPS and follows redirects by default. (see <https://github.com/request/request>)

# Node.js Modules (cont'd)

- **xml2js**: a simple XML to JavaScript object converter. (see <https://www.npmjs.com/package/xml2js>)
- **async**: a utility module which provides straight-forward, powerful functions for working with asynchronous JavaScript. (see <https://caolan.github.io/async/>)
- **q**: a library for promises. A promise is an object that represents the return value or the thrown exception that the function may eventually provide. (see <https://github.com/krisowal/q>)
- **underscore**: a JavaScript library that provides a whole set of useful functional programming helpers without extending any built-in objects. (see <http://underscorejs.org>)
- **socket.io**: a Node.js real-time framework server. It enables real-time bidirectional event-based communication. (see <https://socket.io/docs/>)
- **minimist**: a module used to parse command line arguments. (see <https://www.npmjs.com/package/minimistx>)

## Node.js Modules (cont'd)

- **mocha**: a simple, flexible, fun JavaScript test framework for Node.js and the browser. (see <https://mochajs.org>)
- **http-server**: a simple, zero-configuration command-line http server. Powerful enough for production usage, but it's simple and hackable enough to be used for testing, local development, and learning. (see <https://github.com/indexzero/http-server>)
- **mongodb**: the official MongoDB driver for Node.js. It provides a high-level API on top of mongodb-core that is meant for end users. (see <https://github.com/mongodb/node-mongodb-native>)
- **mongoose**: a MongoDB object modeling tool designed to work in an asynchronous environment. (see <http://mongoosejs.com>)

# Node.js Modules (cont'd)

- **node-fetch**: a light-weight module that brings window.fetch to Node.js. Server version of client-side window.fetch compatible API (see <https://www.npmjs.com/package/node-fetch>)
- **axios**: a promise-based HTTP client for the browser and node.js. Make [XMLHttpRequests](#) from the browser. Make [http](#) requests from node.js. supports the [Promise](#) API. (see <https://www.npmjs.com/package/axios>)
- **Hapi**: a powerful and robust framework that is used for developing APIs. It was first introduced by Eran Hammer 2011 at Walmart while trying to handle the traffic on black Friday.. (see <https://hapi.dev/>)
- **passport**: a unique authentication module for Node.js devs. Hundreds of authentication methods to choose from, starting from internal ones, all the way up to external ones like Google, Facebook, and others. (see <https://github.com/jaredhanson/passport>)

# express Module Example

```
var express = require('express');  
var app = express();  
  
//respond with "hello world" when a GET request is made  
to the homepage  
app.get('/', function (req, res) {  
    res.send('Hello World')  
});  
  
//server listening on port 3000  
app.listen(3000);
```

# **cors Module Example**

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

app.use(cors());

app.get('/products/:id', function (req, res, next) {
  res.json({msg: 'This is CORS-
enabled for all origins!'})
});

app.listen(80, function () {
  console.log('CORS-
enabled web server listening on port 80')
});
```



# xml2js Module Example

```
var parseString = require('xml2js').parseString;  
var xml = "<root>Hello xml2js!</root>";  
  
// "result" is the result of parsing xml  
parseString(xml, function (err, result) {  
    console.dir(result);  
});
```

# Node.js on AWS

- Create Ubuntu Micro EC32 instance
- ssh to the AWS ubuntu server
- Download node.js using 'wget'
- Execute the binaries, make, and install:  
./configure && make && sudo make install

```
ssh -i "aws-csci571.pem" ubuntu@ec2-52-79-54-82.ap-northeast-2.compute.amazonaws.com
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-74-generic x86_64)

* Documentation:  https://help.ubuntu.com/

System information as of Wed Feb  3 23:39:49 UTC 2016

System load:  0.0               Processes:    101
Usage of /:   14.2% of 7.74GB   Users logged in:  0
Memory usage: 10%              IP address for eth0: 172.31.12.48
Swap usage:   0%

Graph this data and manage this system at:
https://landscape.canonical.com/

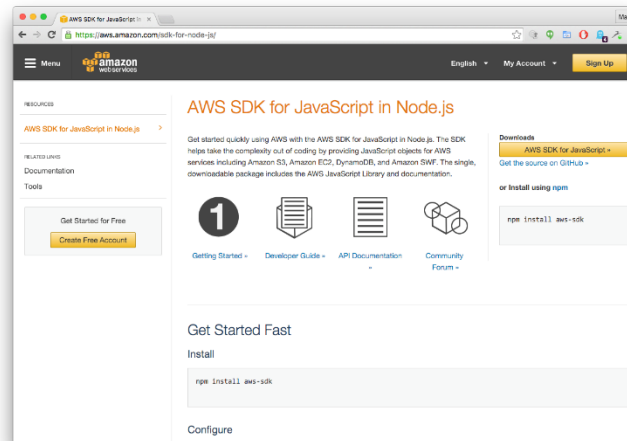
Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

Last login: Wed Feb  3 23:39:49 2016 from usc-secure-wireless-088-117.usc.edu
ubuntu@ip-172-31-12-48:~$ ls
nodejs
```

- Alternatively, use **AWS Elastic Beanstalk** and select the Node.js Preconfigure Platform when creating Environment
- 64bit Amazon Linux running Node.js 10.16.3
- nginx Proxy server
- Supports static file mappings and gzip compression
- See Homework #7 slides

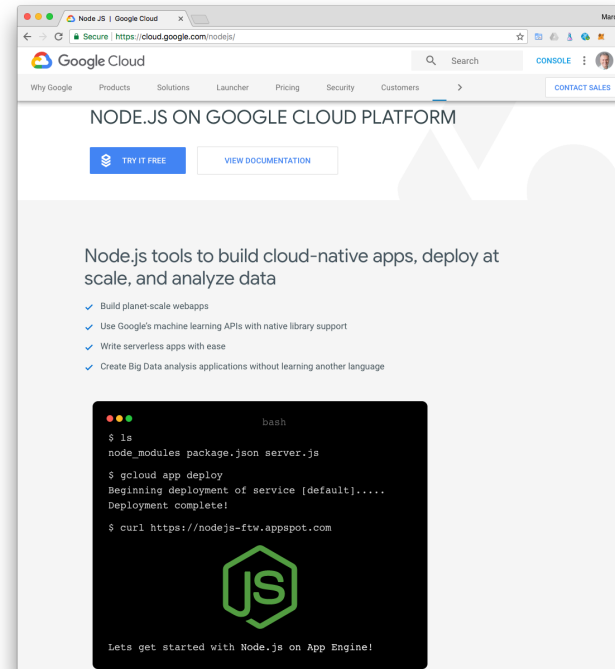
# Node.js on AWS (cont'd)

- AWS SDK for JavaScript in Node.js
- Provides JavaScript objects for AWS services including Amazon S3, Amazon EC2, DynamoDB, Amazon Elastic Beanstalk and many more.
- Single, downloadable package includes the AWS JavaScript Library and documentation
- See: <https://aws.amazon.com/sdk-for-node-js/>



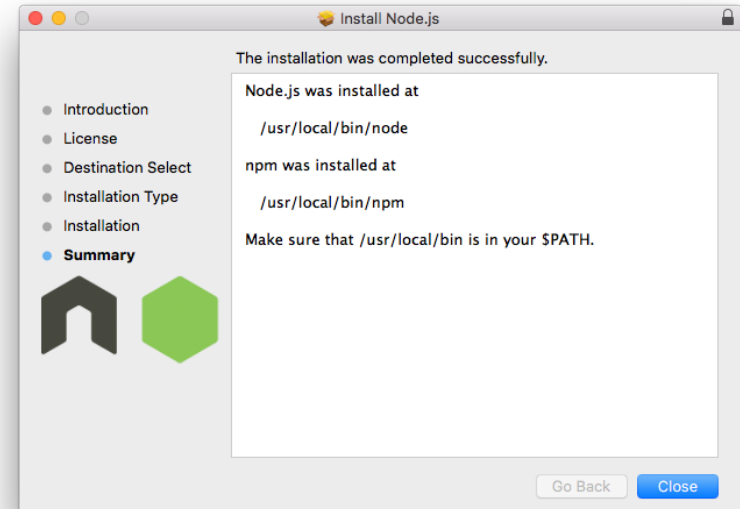
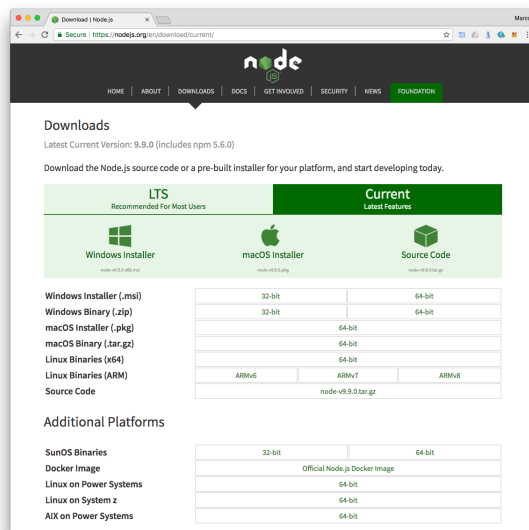
# Node.js on Google Cloud Platform

- Node.js app deployed using Google App Engine Managed VMs
- Scales to serve millions of requests
- Supports structures and binary data, authentication, logging, events
- See: <https://cloud.google.com/nodejs>
- See Homework #7 slides



# Node.js on macOS and Windows

- Download macOS (64bit) and Windows package at <https://nodejs.org/en/>
- Learn Node.js on you local MacBook or Windows PC
- Latest versions are 12.13.0 LTS and 13.0.1 Current



# Related URLs

- **Node.js website:** <https://nodejs.org/>
- **Node.js on Github:** <https://github.com/nodejs/node>
- **NPM:** <https://www.npmjs.com/>
- **Learn Node.js in terminal:** <https://github.com/workshopper/learnyounode>
- **Tools:** <http://gruntjs.com/>

# Introduction

- AngularJS is a complete JavaScript-based open-source front-end web application framework.
- Its mainly maintained by Google and some community of individuals.
- It provides a framework for client-side [model-view-controller](#) (MVC) and [model-view-viewmodel](#) (MVVM) architectures.
- AngularJS is the frontend part of the **MEAN stack**, consisting of MongoDB database, Express.js web application server framework, Angular.js itself, and Node.js runtime environment.



# Angular.js

## Why AngularJS?

- HTML is great for declaring static documents, but it falters when we try to use it for declaring **dynamic views** in web-applications. AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop.

## Alternatives

- 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**.

## Extensibility

- AngularJS is a toolset for building the framework most suited to your application development. It is fully extensible and works well with other libraries. Every feature can be modified or replaced to suit your unique development workflow and feature needs. Read on to find out how.



# Basic functionality

## Control of the app

### Data Binding

- Data-binding is an automatic way of **updating the view** whenever the **model changes**, as well as **updating the model** whenever the **view changes**. This is awesome because it eliminates DOM manipulation from the list of things you have to worry about.

### Controller

- Controllers are the behavior behind the DOM elements. AngularJS lets you express the behavior in a clean readable form without the usual boilerplate of updating the DOM, registering callbacks or watching model changes.

### Plain JavaScript

- Unlike other frameworks, there is no need to inherit from proprietary types in order to wrap the model in accessors methods. Angular models are plain old JavaScript objects. This makes your code easy to test, maintain, reuse, and again free from boilerplate.

# Basic functionality (cont'd)

## Wire up a Backend

### Deep Linking

- A deep link reflects where the user is in the app, this is useful so users can bookmark and email links to locations within apps. Round trip apps get this automatically, but AJAX apps by their nature do not. AngularJS combines the benefits of deep link with desktop app-like behavior.

### Form Validation

- **Client-side form validation** is an important part of great user experience. AngularJS lets you declare the validation rules of the form without having to write JavaScript code. Write less code, go have beer sooner.

### Server Communication

- AngularJS provides built-in services on top of XHR (XMLHttpRequest) as well as various other backends using third party libraries. Promises further simplify your code by handling asynchronous return of data. (Alternative to jQuery's AJAX)

# Basic functionality (cont'd)

## Create Components

### Directives

- Directives is a unique and powerful feature available only in Angular. Directives let you invent new HTML syntax, specific to your application.

### Reusable Components

- We use directives to create reusable components. A component allows you to hide complex DOM structure, CSS, and behavior. This lets you focus either on what the application does or how the application looks separately.

### Localization

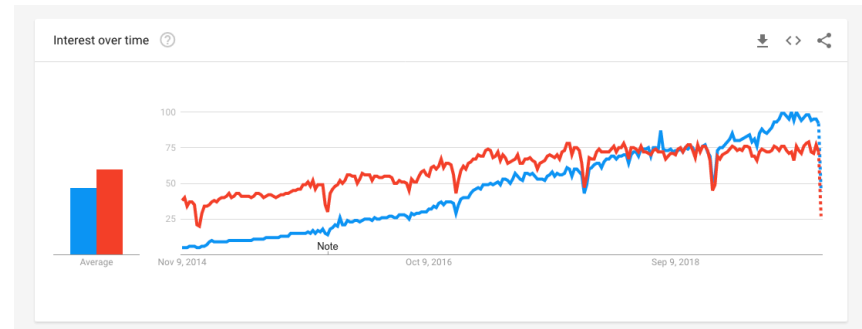
- An important part of serious apps is localization. Angular's locale aware filters and stemming directives give you building blocks to make your application available in all locales.

# Companies that Use Angular JS



There are approximately 12,000 other sites out of 1 million tested in May 2017 that use Angular JS  
Companies that use Angular JS [1] - <http://libscore.com/?#angular> (React is only on 1,455 sites)

React rising (95) and Angular (77) on Google Trends:  
<https://trends.google.com/trends/explore?date=today%205-y&q=%2Fm%2F01211vxv,Angular>



# Goals

AngularJS de-emphasizes explicit DOM manipulation with the goal of improving testability and performance.

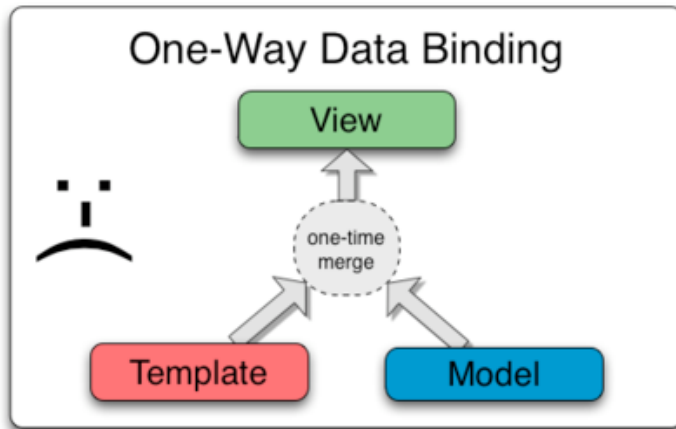
- Design goals are
  - It decouples DOM manipulation from application logic
  - It decouple the client side of an application from the server side.
  - It provides structure for building an application
    - Designing the UI
    - Writing the business logic
    - Testing

# Goals (cont'd)

- Angular JS framework adapts and extends traditional HTML.
- It supports dynamic content through two-way data-binding.
- Two-way data-binding allows for the automatic synchronization of models and views.
- The tasks in angular bootstrapper occur in 3 phases
  - Creation of a new Injector
  - Compilation of the directives that decorate the DOM
  - Linking of all directives to scope

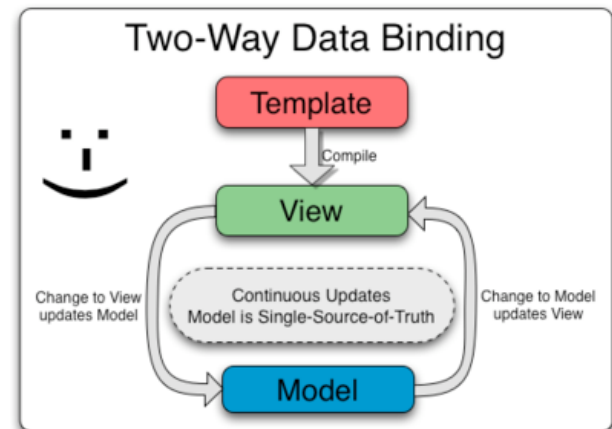
# Data Binding

Classical



Any changes that the user makes to the view are not reflected in the model

Angular



The view is just a projection of the model. So there is automatic refresh of the data between view and model

# AngularJS Directives

- AngularJS directives allows to specify custom reusable HTML-like elements and attributes.
- Some of the Angular Directives are
  - ng-app
  - ng-controller
  - ng-bind
  - ng-model
  - ng-class
  - ng-repeat



# AngularJS Directives (Cont'd)

- **ng-app**
  - Declares the root element of an AngularJS application.
- **ng-controller**
  - Specifies a JavaScript controller class that evaluates HTML expressions.
- **ng-bind**
  - Sets the text of a DOM element to the value of an expression.
- **ng-model**
  - Similar to ng-bind, but establishes a two-way data binding between the view and the scope.
- **ng-repeat**
  - Instantiate an element once per item from a collection.

# Code Snippet – AngularJS Instantiation

```
<script>
var app = angular.module("myApp", []);
app.controller("myController", function($scope,$http) {
    // $scope holds your model (metadata) for your application
    $scope.topic = "CSCI 571";
});
</script>
```

```
<body ng-app="myApp" ng-controller="myController">
</body>
```

- An AngularJS module defines an application.
- The module is a container for controllers
- Controllers always belong to a module.

# AngularJS Data Binding - Example

```
<div ng-app="formExample" ng-controller="ExampleController">
  <form class="form container simple-form">
    <div class="form-group">
      <label>Name :</label> <input class="form-control" type="text" ng-model="user.name" />
    </div>
    <div class="form-group">
      <label>E-mail:</label> <input class="form-control" type="email" ng-model="user.email" /><br />
    </div>
    <div class="form-group">
      <label>School:</label> <input class="form-control" type="text" ng-model="user.school" /><br />
    </div>
    <div class="form-group">
      <label>Level:</label>
      <select class="form-control" ng-model="user.level">
        <option value=""></option>
        <option value="G">Graduate</option>
        <option value="UG">Under Graduate</option>
        <option value="PhD">Doctoral</option>
      </select><br />
    </div>
    <input class="btn btn-default" type="button" ng-click="reset()" value="Reset" />
    <input class="btn btn-primary" type="submit" ng-click="update(user)" value="Save" />
  </form>
  <br/>
  <div class="container">
    <pre>user = {{user | json}}</pre>
    <pre>master = {{master | json}}</pre>
  </div>
</div>
```

```
angular.module('formExample', [])
  .controller('ExampleController', ['$scope', function($scope) {
    $scope.master = {};

    $scope.update = function(user) {
      $scope.master = angular.copy(user);
    };

    $scope.reset = function() {
      $scope.user = angular.copy($scope.master);
    };

    $scope.reset();
  }])
```

A simple example which shows how AngularJS process two-way data binding using **ng-model**.

<http://csci571.com/examples/Angular/binding/index.html>

# AngularJS Repeat with data from static array

```
<div class="row">
  <div class="col-md-6 col-sm-12">
    <h3>Loading Data from Array</h3>
    <h4>Web Tech Producer</h4>
    <table class="table table-striped">
      <tr>
        <th>#</th>
        <th>Name</th>
        <th>Office Hours</th>
        <th>Location</th>
      </tr>
      <tr ng-repeat="x in producers track by $index">
        <td>{{ $index + 1 }}</td>
        <td>{{ x.Name }}</td>
        <td>{{ x.Office }}</td>
        <td>{{ x.Location }}</td>
      </tr>
    </table>
  </div>
</div>
```

**ng-repeat** works like a for loop and replicates the template to the number of rows in the model

```
var app = angular.module("data", []);
app.controller("data", function ($scope, $http) {
  $scope.producers = [
    {
      Name: "Producer 1",
      Office: "10-11 AM",
      Location: "Leavey Library (LVL) 201"
    },
    {
      Name: "Producer 2",
      Office: "9-10 AM",
      Location: "Leavey Library (LVL) 202"
    },
    {
      Name: "Producer 3",
      Office: "4-5 PM",
      Location: "Leavey Library (LVL) 203"
    },
    {
      Name: "Producer 4",
      Office: "2-3 PM",
      Location: "Leavey Library (LVL) 204"
    },
    {
      Name: "Producer 5",
      Office: "5-6 PM",
      Location: "Leavey Library (LVL) 201"
    },
    {
      Name: "Producer 6",
      Office: "10-11 AM",
      Location: "Leavey Library (LVL) 209"
    },
    {
      Name: "Producer 7",
      Office: "10-11 AM",
      Location: "Leavey Library (LVL) 202"
    }
  ];
});
```

[http://csci571.com/examples/Angular/populating\\_data/index.html](http://csci571.com/examples/Angular/populating_data/index.html)

# AngularJS Repeat from http request

```
<table class="table table-responsive table-striped" ng-app="myapp" ng-controller="myapp">
  <tr>
    <th>Name</th>
    <th>City</th>
    <th>Country</th>
  </tr>
  <tr ng-repeat="x in rows">
    <td>{{x.Name}}</td>
    <td>{{x.City}}</td>
    <td>{{x.Country}}</td>
  </tr>
</table>
```

```
angular.module('myapp', [])
  .controller('myapp', function($scope, $http) {
    $scope.rows = {};

    $http.get("http://www.w3schools.com/angular/customers.php").then(function(response){
      $scope.rows = response.data.records;
    });
  });
```

\$http holds the xml http request handler in Angular.

Name	City	Country
Alfreds Futterkiste	Berlin	Germany
Ana Trujillo Emparedados y helados	México D.F.	Mexico
Antonio Moreno Taquería	México D.F.	Mexico
Around the Horn	London	UK
B's Beverages	London	UK
Berglunds snabbköp	Luleå	Sweden
Blauer See Delikatessen	Mannheim	Germany

[http://csci571.com/examples/Angular/populating\\_data/index.html](http://csci571.com/examples/Angular/populating_data/index.html)

# AngularJS Sort and Search

```
<table class="table table-responsive table-striped" ng-app="myapp" ng-controller="myapp">
  <tr>
    <th>Name</th>
    <th>City</th>
    <th>Country
      <input type="text" class="form-control pull-right" style="width:40%;" placeholder="Search" ng-
model="search" />
    </th>
  </tr>
  <tr ng-repeat="x in rows | orderBy:Name | filter: search">
    <td>{{x.Name}}</td>
    <td>{{x.City}}</td>
    <td>{{x.Country}}</td>
  </tr>
</table>
```

```
angular.module('myapp', [])
.controller('myapp', function($scope, $http) {
  $scope.rows = {};

  $http.get("http://www.w3schools.com/angular/customers.php").then(function(response){
    $scope.rows = response.data.records;
  });
});
```

**orderBy:** sort the Column ascending

*orderBy:* <column>: <reverse>

*<column>* - the column you want to sort

*<reverse>* - true-descending, false-ascending

**Filter:** search the rows in the model

*Filter:* <searchstring> e.x. filter: search

Or

*Filter:* <column\_based\_search> e.x.

*filter:* {<column>:search}

Name	City	Country
Alfreds Futterkiste	Berlin	Germany
Ana Trujillo Emparedados y helados	México D.F.	Mexico
Antonio Moreno Taquería	México D.F.	Mexico
Around the Horn	London	UK
B's Beverages	London	UK
Berglunds snabbköp	Luleå	Sweden
Blauer See Delikatessen	Mannheim	Germany

[http://csci571.com/examples/Angular/sort\\_and\\_search/index.html](http://csci571.com/examples/Angular/sort_and_search/index.html)

# AngularJS External UI Components

```
<table class="table table-responsive table-striped" ng-app="myapp" ng-controller="myapp">
  <tr>
    <th>Name</th>
    <th>City</th>
    <th>Country
      <input type="text" class="form-control pull-right" style="width:40%;" placeholder="Search" ng-
model="search" />
    </th>
  </tr>
  <tr dir-paginate="x in rows | orderBy:Name:false | filter: search | itemsperpage: 10" pagination-
id="example">
    <td>{{x.Name}}</td>
    <td>{{x.City}}</td>
    <td>{{x.Country}}</td>
  </tr>
</table>
<dir-pagination-controls max-size="10" boundary-links="true" direction-links="true" max-size="10"
pagination-id="example"></dir-pagination-controls>
```

External components need to be added to the angular application.

```
angular.module('myapp',
[<external_components>])
```

```
angular.module('myapp', ['angularUtils.directives.dirPagination'])
.controller('myapp', function($scope, $http) {
  $scope.rows = {};

  $http.get("http://www.w3schools.com/angular/customers.php").then(function(response){
    $scope.rows = response.data.records;
  });
});
```

[http://csci571.com/examples/Angular/external\\_plugins/index.html](http://csci571.com/examples/Angular/external_plugins/index.html)

# AngularJS Remove and Insert DOM Element

```
<div ng-app="myApp">
  <div ng-controller="AppCtrl">
    <input type="checkbox" ng-click="toggleShowDiv()"/>
    <label for="showDiv">Toggle DIV</label>
    <div id="my-div" ng-if="showDiv">New Div</div>
  </div>
</div>
```

Using *ng-if* to insert/remove the dom.

```
angular.module('myApp', ['ngAnimate'])
  .controller('AppCtrl', function ($scope) {
    $scope.showDiv = false;

    $scope.toggleShowDiv = function(){
      console.log('fired');
      $scope.showDiv = !$scope.showDiv;
    }
  });
```

[http://csci571.com/examples/Angular/manipulate\\_dom/index.html](http://csci571.com/examples/Angular/manipulate_dom/index.html)



# About Angular 2+

- AngularJS is the name for all AngularJS version 1.x.
- Angular 2+ stands for the Angular 2 and later, including Angular 2,4,5,6,7 and 8.
- Angular 2+ is a ground-up rewrite of AngularJS and has many unique qualities.
- AngularJS 1.7 was released on July 1, 2018 and it enters a 3 years long-term support. It is “retired” now.
- There are few upgrade possibilities of AngularJS to Angular 2+ and in most cases the only suitable possible option is to rewrite the application in Angular 2+.
- Starting with Angular 2, a major release is published every 6 months. There are some improvements between each version.
- The latest stable Angular version is Angular 9 and was released in November 2019. <https://angular.io/guide/releases>

# Differences between versions

	AngularJS	Angular 2+
<b>Architecture</b>	Based on MVC	Based on service/component
<b>Javascript vs TypeScript</b>	Uses JavaScript	Uses TypeScript (a superset of JavaScript ) and RxJS (A reactive programming library for JavaScript).
<b>Component-based UI</b>		Can split application features into various components and call required UI, which increases reusability and flexibility
<b>Mobile Support</b>		Native Script or Ionic
<b>SEO (Search Engine Optimization) Friendly</b>	Difficult to develop search engine friendly Single Page Applications	Possible to build Single Page Applications SEO friendly by rendering plain HTML at the server side. It released Angular Universal.

# Features of Angular 4

- **Smaller and Faster**
- **View engine with less code**
  - The view engine is introduced in Angular 4 where the produced code of components can be reduced up to 60%. The bundles are reduced to thousands of KBs.
- **Improved \*ngIf directive**
  - A new “else” statement is added
- **Animation**
  - Animations are pulled from the Angular core and set in their own package
- **TypeScript 2.1 and 2.2 Compatibility**
- **Source Maps for Templates**
  - Now whenever there's an error caused by something in one of the templates, source maps are created which provide a meaningful context concerning the original template.

# Features of Angular 6

- **ng update**
  - Using **ng update** to update your Angular app from Angular 2, 4 or 5.
- **ng add**
  - Using **ng add** to add new dependencies and invoke an installation script.
- **Angular Material + CDK Components**
- **CLI Workspaces**
  - CLI projects will now use angular.json for build and project configuration.
- **Animations Performance Improvements**
- **RxJS v6**

# Features of Angular 7

- **Application Performance**
  - Optimize polyfills.ts
- **Angular Material & the CDK**
  - Virtual Scrolling
  - Drag and Drop
- **Partner Launches**
  - NativeScript: A single project that builds for both web and installed mobile with NativeScript
  - StackBlitz 2.0 supports multipane editing and the Angular Language Service
- **Dependency Updates**
  - Typescript 3.1
  - RxJS 6.3
  - Node 10

# Features of Angular 8

- **Requires TypeScript 3.4**
  - Required. You will need to upgrade.
- **Ivy support**
  - New compiler / runtime for Angular
- **Bazel support**
  - Build tool developed and built by Google.
  - Build back-end and front-end with same tool
- **Router**
  - Lazy-loading with `import()` syntax
- **Location**
  - Location services to help migrating from AngularJS

# Features of Angular 9

- **Requires TypeScript 3.7**
  - Required. You will need to upgrade.
- **Ivy support**
  - Angular compiles by Ivy by default
- **APIs**
  - Lots on new removed and deprecated APIs
- **Migration**
  - CLI handles many migrations automatically

# RxJS

- Reactive programming is an asynchronous programming paradigm concerned with data streams and the propagation of change (Wikipedia)
- RxJS is a library for reactive programming. ([RxJS Docs](#))
- Angular2+ use RxJS for asynchronous operation. ([Angular RxJS](#))
- RxJS uses [Observables](#) for asynchronous or callback-based code.
  - Converting existing code for async operations into observables
  - Iterating through the values in a stream
  - Mapping values to different types
  - Filtering streams
  - Composing multiple streams
- See example “Repeat from http request” on page 48.



# Angular 2+ Data Binding Component Example

```
<div>
  <form class="form container simple-form">
    <div class="form-group">
      <label>Name:</label> <input class="form-control" type="text" [(ngModel)]="user.name" name="name" />
    </div>
    <div class="form-group">
      <label>E-mail:</label> <input class="form-control" type="email" [(ngModel)]="user.email" name="email" /><br />
    </div>
    <div class="form-group">
      <label>School:</label> <input class="form-control" type="text" [(ngModel)]="user.school" name="school" /><br />
    </div>
    <div class="form-group">
      <label>Level:</label>
      <select class="form-control" [(ngModel)]="user.level" name="level">
        <option value=""></option>
        <option value="G">Graduate</option>
        <option value="UG">Under Graduate</option>
        <option value="PhD">Doctoral</option>
      </select><br />
    </div>
    <input class="btn btn-default" type="button" (click)="reset()" value="Reset" />
    <input class="btn btn-primary" type="submit" (click)="update()" value="Save" />
  </form>
  <br />
  <div class="container">
    <pre>user = {{user | json}}</pre>
    <pre>master = {{master | json}}</pre>
  </div>
</div>
```

form.component.html

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent implements OnInit {
  user = {};
  master = {};

  constructor() { }

  ngOnInit() {
  }

  update() {
    this.master = Object.assign({}, this.user);
  }

  reset() {
    this.user = Object.assign({}, this.master);
  }
}
```

form.component.ts

[http://csci571.com/examples/Angular/binding\\_angular2\\_and\\_4/index.html](http://csci571.com/examples/Angular/binding_angular2_and_4/index.html)

# Angular 2+ Repeat with data from static array

```
<div class="row">
  <div class="col-md-6 col-sm-12">
    <h3>Loading Data from Array</h3>
    <h4>Web Tech Producer</h4>
    <table class="table table-striped">
      <tr>
        <th>#</th>
        <th>Name</th>
        <th>Office Hours</th>
        <th>Locaton</th>
      </tr>
      <tr *ngFor="let x of producers; index as i; trackBy: trackByFn">
        <td>{{i + 1}}</td>
        <td>{{x.Name}}</td>
        <td>{{x.Office}}</td>
        <td>{{x.Location}}</td>
      </tr>
    </table>
  </div>
</div>
```

staff.component.html

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-staff',
  templateUrl: './staff.component.html',
  styleUrls: ['./staff.component.css']
})
export class StaffComponent implements OnInit {

  producers = [
    {
      Name: 'Producer 1',
      Office: '10-11 AM',
      Location: 'Leavey Library (LVL) 201'
    },
    {
      Name: 'Producer 2',
      Office: '9-10 AM',
      Location: 'Leavey Library (LVL) 202'
    },
    {
      Name: 'Producer 3',
      Office: '4-5 PM',
      Location: 'Leavey Library (LVL) 203'
    },
    {
      Name: 'Producer 4',
      Office: '2-3 PM',
      Location: 'Leavey Library (LVL) 204'
    },
    {
      Name: 'Producer 5',
      Office: '5-6 PM',
      Location: 'Leavey Library (LVL) 201'
    },
    {
      Name: 'Producer 6',
      Office: '10-11 AM',
      Location: 'Leavey Library (LVL) 209'
    },
    {
      Name: 'Producer 7',
      Office: '10-11 AM',
      Location: 'Leavey Library (LVL) 202'
    }
  ];

  constructor() {}

  ngOnInit() {}

  trackByFn(index, item) {
    return index;
  }
}
```

staff.component.ts

[http://csci571.com/examples/Angular/populating\\_data\\_static\\_angular\\_2\\_and\\_4/index.html](http://csci571.com/examples/Angular/populating_data_static_angular_2_and_4/index.html)

# Angular 2+ Repeat from http request

```
<table class="table table-responsive table-striped">
  <tr>
    <th>Name</th>
    <th>Email</th>
    <th>Phone</th>
    <th>Website</th>
  </tr>
  <tr *ngFor="let x of rows">
    <td>{{x.name}}</td>
    <td>{{x.email}}</td>
    <td>{{x.phone}}</td>
    <td>{{x.website}}</td>
  </tr>
</table>
```

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';

@Injectable()
export class DataService {

  constructor(private http: Http) { }

  getCustomer() {
    return this.http.get('http://jsonplaceholder.typicode.com/users');
  }
}
```

```
import { DataService } from './../../services/data.service';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-customer',
  templateUrl: './customer.component.html',
  styleUrls: ['./customer.component.css']
})
export class CustomerComponent implements OnInit {

  rows = [];

  constructor(private dataService: DataService) {
    dataService.getCustomer().subscribe(response => {
      this.rows = response.json();
    });
  }

  ngOnInit() {
  }
}
```

Using **observable** subscription to handle  
asynchronized http request  
(sorry example broken)

[http://csci571.com/examples/Angular/populating\\_data\\_external\\_angular\\_2\\_and\\_4/index.html](http://csci571.com/examples/Angular/populating_data_external_angular_2_and_4/index.html)

# Angular 2+ Sort and Search

- Angular 2+ doesn't support *FilterPipe* or *OrderByPipe* mainly because they are expensive operations and they have often been abused in AngularJS apps.
- To learn more about why *FilterPipe* and *OrderByPipe* are not supported and what the alternatives are, see this page:

<https://angular.io/guide/pipes#appendix-no-filterpipe-or-orderbypipe>.

# Angular 2+ External UI Components

```
<table class="table table-responsive table-striped">
  <tr>
    <th>Name</th>
    <th>Email</th>
    <th>Phone</th>
    <th>Website</th>
  </tr>
  <tr *ngFor="let x of rows | paginate: { itemsPerPage: 5, currentPage: p }">
    <td>{{x.name}}</td>
    <td>{{x.email}}</td>
    <td>{{x.phone}}</td>
    <td>{{x.website}}</td>
  </tr>
</table>

<pagination-controls (pageChange)="p = $event"></pagination-controls>
```

```
import { DataService } from './services/data.service';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpModule } from '@angular/http';
import { NgxPaginationModule } from 'ngx-pagination';

import { AppComponent } from './app.component';
import { PaginationComponent } from './components/pagination/pagination.component';

@NgModule({
  declarations: [
    AppComponent,
    PaginationComponent
  ],
  imports: [
    BrowserModule,
    HttpModule,
    NgxPaginationModule
  ],
  providers: [DataService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```
import { DataService } from './../../services/data.service';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-pagination',
  templateUrl: './pagination.component.html',
  styleUrls: ['./pagination.component.css']
})
export class PaginationComponent implements OnInit {

  p: number = 1;
  rows = [];

  constructor(private dataService: DataService) {
    dataService.getCustomer().subscribe(response => {
      this.rows = response.json();
    });
  }

  ngOnInit() {
  }
}
```

(sorry example broken)

[http://csci571.com/examples/Angular/external\\_plugins\\_angular\\_2\\_and\\_4/index.html](http://csci571.com/examples/Angular/external_plugins_angular_2_and_4/index.html)

# Angular 2+ Remove and Insert DOM Element

```
<div>
  <div>
    <input type="checkbox" (click)="toggleShowDiv()"/>
    <label for="showDiv">Toggle DIV</label>
    <div id="my-div" *ngIf="showDiv">New Div</div>
  </div>
</div>
```

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-manipulation',
  templateUrl: './manipulation.component.html',
  styleUrls: ['./manipulation.component.css']
})
export class ManipulationComponent implements OnInit {

  showDiv = false;

  constructor() { }

  ngOnInit() {
  }

  toggleShowDiv() {
    console.log('fired');
    this.showDiv = !this.showDiv;
  }
}
```

[http://csci571.com/examples/Angular/manipulate\\_dom\\_angular\\_2\\_and\\_4/index.html](http://csci571.com/examples/Angular/manipulate_dom_angular_2_and_4/index.html)

# TypeScript

- Open source programming language developed and maintained by Microsoft
- Syntactical superset of JavaScript
- Developed by Anders Hejlsberg, C# Architect and creator of Turbo Pascal
- First made public in October 2012 (version 0.8)
- Built-in support for TypeScript in Visual Studio 2013+
- TypeScript 3.0 released July 2018
- TypeScript program can seamlessly consume JavaScript
- TypeScript compiler written in TypeScript
- See: <http://www.typescriptlang.org/>

# TypeScript Features

- Extensions to ECMAScript 5<sup>th</sup> Ed.
  - Type annotations and compile-time type checking
  - Type inference
  - Type erasure
  - Interfaces
  - Enumerated type
  - Mixin
  - Generic
  - Namespaces
  - Tuple
  - Await
- Backported features from ECMAScript 2015
  - Classes
  - Modules
  - Arrow syntax for anonymous functions
  - Optional and default parameters



# TypeScript and Angular

- Angular IDE, optimized for Angular 2+
- Commercial product from Webclipse  
<https://www.genuitec.com/products/angular-ide/>
- TypeScript 3.x validation and debugging
- Angular HTML Template Intelligence
  - Validation
  - Detection of misdefined element tags
  - HTML elements auto-complete
  - TypeScript expressions auto-complete
- Angular-CLI Integration
- Angular Source Navigation
- TypeScript Debugging
- Live Preview
- Free (for 45 days) download for Windows, MacOS and Linux at:  
<https://www.genuitec.com/products/angular-ide/download/>

# Example usage & Related URLs

## Examples

- **Hello World:** <https://angularjs.org/#the-basics>
- **Todo List:** <https://angularjs.org/#add-some-control>
- **Advanced Single Page App:** <https://angularjs.org/#wire-up-a-backend>

## Related URLs

- **Angular.js website:** <https://angularjs.org>
- **Angular.js on Github:** <https://github.com/angular/angular.js>
- **Tutorial:** <https://docs.angularjs.org/tutorial>
- **Angular.js Course**  
<http://campus.codeschool.com/courses/shaping-up-with-angular-js/level/1/section/1/creating-a-store-module>
- **Angular 2+:** <https://angular.io/>
- **Angular 2+ docs:** <https://angular.io/docs>
- **Angular 2+ IDEs, Tools, Libraries, UI Components:** <https://angular.io/resources>