
Construction of User Interfaces (SE/ComS 319)

Ali Jannesari



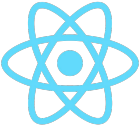


Jinu Susan Kabala

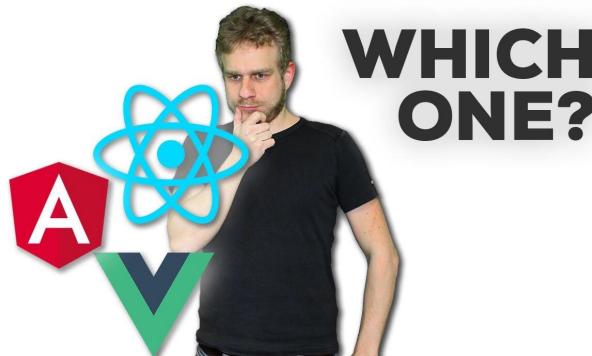
Department of Computer Science

Iowa State University, Spring 2021

FRAMEWORKS AND API

Libraries and frameworks for Node.js & JavaScript

- Chrome DevTools 
- NPM is a package manager for Node.js packages/modules 
- Libraries/frameworks:
 - **React.js**: JavaScript library for building user interfaces by Facebook 
 - **Angular/Angular.js**: TypeScript-based/Javascript framework by Google 
 - **Vue.js**: rapidly growing JS framework 



See this video for an answer: <https://www.youtube.com/watch?v=KMX1mFEmM3E&app=desktop>

JavaScript framework

REACT

PART II

JavaScript framework – React

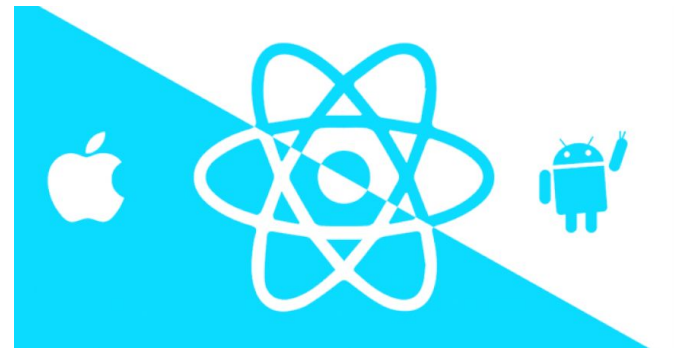
- **React** is a JavaScript library for building fast and interactive user interfaces for the web.
- It allows developers to write highly efficient JavaScript for the purpose of rendering a UI.
- It is an **open source**, reusable **component-based** front-end library.
- Traditional JavaScript will re-render the entire DOM during a state change
 - But React will only render the parts of the DOM that have changed.
- In a model-view-controller architecture, React is the ‘view’ which is responsible for how the app looks.

Beyond ReactJS

- React goes beyond simple UI and has many extensions for complete application support.
- It provides server-side rendering.
- Supports mobile app development
- Extended with Flux and Redux.

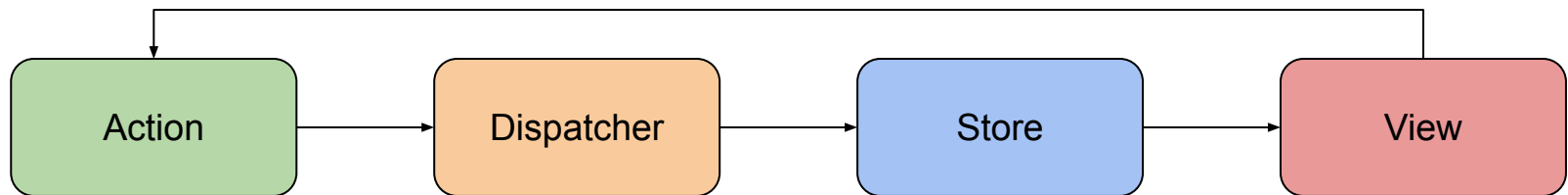
React Native

- React Native lets your build mobile apps using only JavaScript.
- A React Native app is a REAL mobile app and not just a web app running on mobile.



FLUX

- Flux is the application architecture that Facebook uses for building web applications.
- Flux Implements unidirectional flow which makes it easier to deduce what's going on.

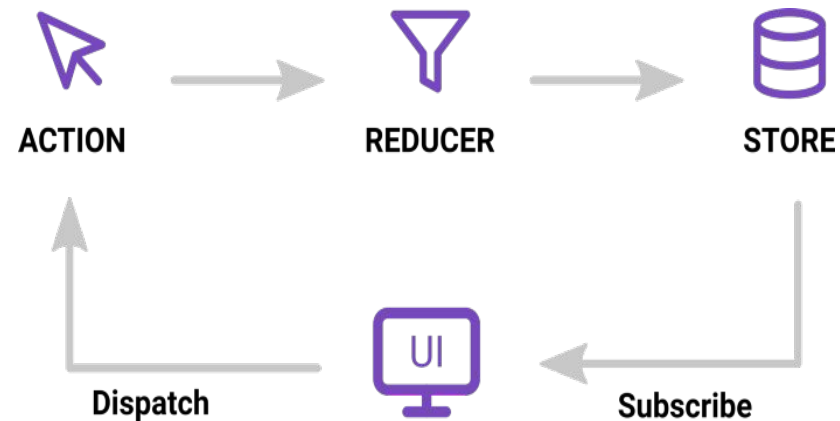


Redux

- Redux is a popular JavaScript library for managing the state of your application.
- Redux stores the whole state of the app in an immutable object tree.

The three building blocks of Redux

1. **Store:** holds the state of the application.
2. **Actions:** JavaScript objects that describe what has happened
3. **Reducers:** functions that define how the app state changes.



Design pattern – Singleton in React

- **Singleton** is prevalent
- React **components** could use Singletons
 - Once React constructs an object, the same instance is reused throughout your app (there are never ever two instances!)
 - Great candidates to share application data across multiple components
- **Example**
 - After a successful login: you'll need to store the login status to be used in all other components
 - Store the status in an object: you can just inject it into your controller/service and check it whenever you need!

Design pattern - Singleton

What is Singleton?

- Singleton synchronizes state between all instances of a component as if that component were a singleton object.
- If any instance updates its state, all other instances will be updated automatically.
- This is how singleton enables us to encapsulate data in a component and still have it available throughout the application via the interface.

Singleton – Design pattern

- **Purpose**

- Ensure that a class has exactly one instance and provide a global access point to it

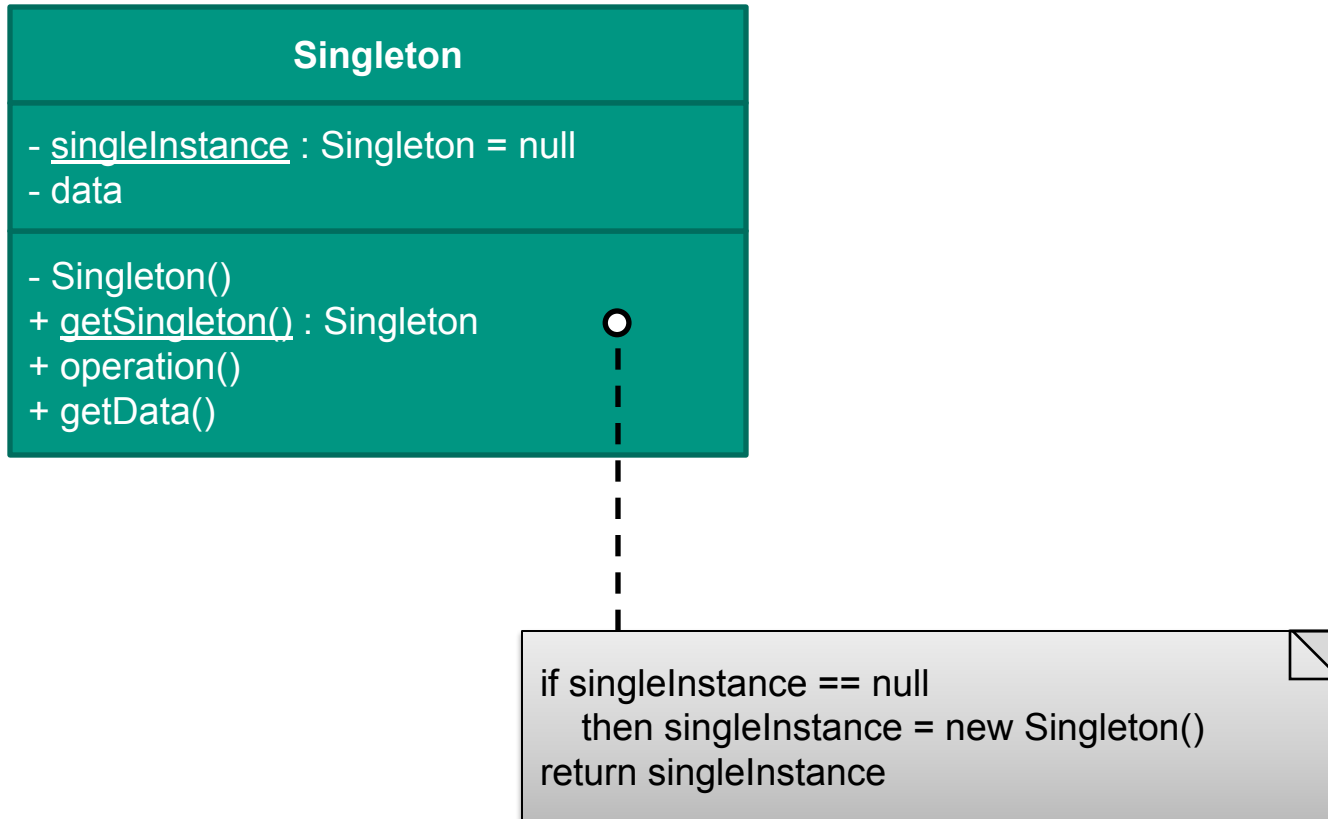
- **Motivation**

- The class is responsible for managing its only copy. By intercepting commands to create new objects, the class can ensure that no additional instance is created.

- **When to Use Singleton**

- It is better to use singleton to manage the shared data - without using the *provider pattern* or exposing state.

Singleton: Structure



Singleton – Example

- **Logger Classes:** The Singleton pattern is used in the design of logger classes (more than one logger instance the log file is a total chaos).
- **Configuration Classes:** The Singleton pattern is used in classes which provides the configuration settings for an application.
- **Java Singleton**
 - Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine.
 - Singleton pattern is used for logging, drivers objects, caching and thread pool.

Singleton: Applicability

- If there is **only one instance** of a class and this instance is to be made accessible to the client in a known place.
- If it is difficult or impossible to determine which part of the application is creating the first instance.
- If the only instance should be extensible by sub-classing and the clients should use it without changing their source code.

Singleton in JavaScript

```
var Singleton = (function () {  
    var instance;  
  
    function createInstance() {  
        var object = new Object("I am the instance");  
        return object;  
    }  
  
    return {  
        getInstance: function () {  
            if (!instance) {  
                instance = createInstance();  
            }  
            return instance;  
        }  
    };  
})();  
  
function run() {  
  
    var instance1 = Singleton.getInstance();  
    var instance2 = Singleton.getInstance();  
  
    alert("Same instance? " + (instance1 === instance2));  
}
```

React Singleton: When to Use

- While working on an application that has a dozen teams working on it across dozens of repositories. Each team manages different pages of the application and we all write shared components for other teams to use on their pages.
- *Provider* and *broadcast* patterns couple the composition of individual components to the full application.
 - This is a major hurdle (full of frustration, lost time, and wasted money) every time a team wants to alter how it manages its data.
- With a shared global Redux store, teams also have the ability to carry that on top of Redux actions, even though those actions can change at any time in the future.
- **Solution:** *Singleton* pattern

React Singleton: How to Use

- We can use Singleton: the encapsulated, simple, readable way to provide shared information across a multitude of components.
- For example, we can use it to manage the logged-in user account in an application.
 - Many parts of the app need to know if a user is logged in, who that user is, and what preferences they have configured.
- Any component throughout the app can import a singleton state hook.

```
npm install --save-dev https://github.com/peterbee/react-singleton.git
```

```
import createSingleton from '@peterbee/react-singleton';

const [useMyData, updateMyData] = createSingleton('initial value');

function MyComponent() {
  const myData = useMyData();

  return myData;
}
```

- Install a React Singleton [1] module with React hook* that synchronizes state across all instances of a component.

** Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.*

React Singleton: How to Use (2)

- We can use Singleton: the encapsulated, simple, readable way to provide shared information across a multitude of components.
- For example, we can use it to manage the logged-in user account in an application.
 - Many parts of the app need to know if a user is logged in, who that user is, and what preferences they have configured.
- Any component throughout the app can import a singleton state hook

```
npm install --save-dev https://github.com/peterbee/react-singleton
```

```
import createSingleton from '@peterbee/react-singleton';

const [useMyData, updateMyData] = createSingleton('initial value');

function MyComponent() {
  const myData = useMyData();

  return myData;
}
```

- When *updateMyData* is called, every component that uses the hook, re-renders with the updated value.
- No need of tracing through complex call stacks and data stores to figure out which action needs to be dispatched or which listeners are subscribed to which channels.

** Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.*

React Singleton: How to Use

- Create the hook in a file that owns account data:

```
// useAccount.js
import createSingleton from '@peterbee/react-singleton';

const [useAccount, updateAccount] = createSingleton(accountObject);

export default useAccount;
```

- Use the hook in any component that needs account data:

```
import useAccount from './useAccount';

function Profile() {
  const account = useAccount();

  return (
    <div>Hello {account.first_name}!</div>
  );
}

export default Profile;
```

Literature – React

- <https://www.w3schools.com/react/default.asp>
- <https://reactjs.org/docs/getting-started.html>
- <https://react-tutorial.app/>
- <https://www.tutorialspoint.com/reactjs/index.htm>

OTHER FRAMEWORKS

Angular

- Angular JS is on long term support.
- It's been replaced with **Angular**



- Angular (currently Angular v7) is a reboot of AngularJS. It's a complete makeover, and not just a following iteration.
- AngularJS long term support ends on june 2021. Until then, Google will only be fixing bugs and making sure AngularJS doesn't break with new browsers/libraries.

Language

AngularJS

- Javascript
 - JS is a very powerful web language.
 - Because of its popularity, it introduces a lesser learning curve.

```
var name = "Susan",  
    age = 25,  
    hasCode = true;
```

Angular

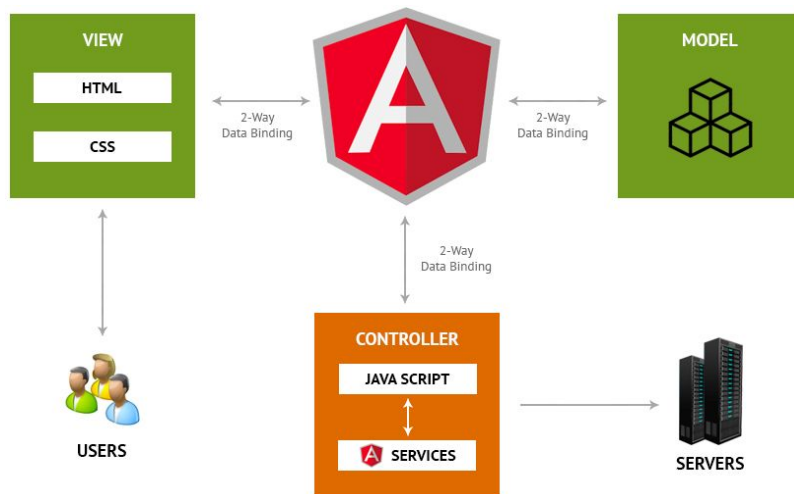
- Typescript
 - Typescript is a JS superset.
 - It introduces static typing.
 - It builds on JS in terms of supporting OOP.
 - It is compiled (transpiled) into JS before running in the browser.
 - Steeper learning curve

```
let name: string = "Susan",  
    age: number = 25,  
    hasCode: boolean = true;
```

Architecture

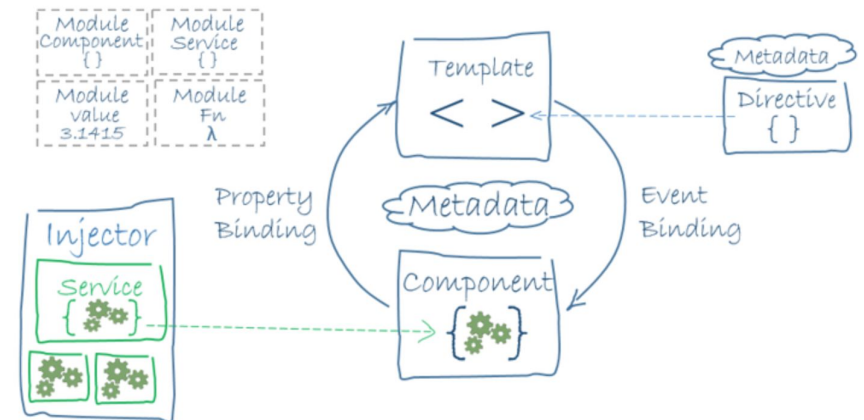
AngularJS

- MVC (Model-View-Controller)



Angular

- Component Based architecture.



Literature – Angular

- <https://angular-templates.io/tutorials/about/learn-angular-from-scratch-step-by-step>
- <https://www.tektutorialshub.com/angular-tutorial/>
- <https://www.edureka.co/blog/angular-tutorial/>
- <https://stackblitz.com/edit/angular-xhjhsd>

Laravel – The PHP Framework

- What is it?
 - A framework
- Why learn it? (PHP users)
 - Organize code into appropriate folders (modular development)
 - Use conventions for quick development + maintenance
 - Access libraries and utilities, authentication, etc.
- Documentation
 - <https://laravel.com/docs/5.3/> (incl. installation)
 - <https://laracasts.com/series/laravel-5-from-scratch> (there are 18 screencasts walking you thru the materials)

Laravel – MVC Framework

1. Submit User Request

2. Route to appropriate Laravel Controller

Routing

Controller

- Most **PHP** frameworks are based on the MVC pattern
- An **index.php** script gets the URL and starts the **routing** process.

3. Interact with Data Model

4. Controller invokes results View

View

Model

5. Render view in users browser



Database

Laravel – Main files and locations

./.env

./app/Http/routes.php

1 - routes

./app/Http/Controllers/Auth/AuthController.php

./app/Http/Controllers/Auth/PasswordController.php

./app/Http/Controllers/Controller.php

./app/Http/Controllers/MyController.php

2 -
controllers

./app/TestTable.php

./app/User.php

3 - model

./config/app.php

./config/database.php

./config/view.php

4 - config

./public/.htaccess

./public/index.php

./public/web.config

5 - website

./resources/views/errors/503.blade.php

./resources/views/welcome.blade.php

6 - views

./tests/ExampleTest.php

./tests/TestCase.php

7 - tests

Literature – LARAVEL

- Documentation
 - See <https://laravel.com/docs/5.3/> (incl. installation)
- BEST RESOURCE FOR LEARNING LARAVEL
 - <https://laracasts.com/series/laravel-5-from-scratch> (there are 18 screencasts walking you thru the materials)

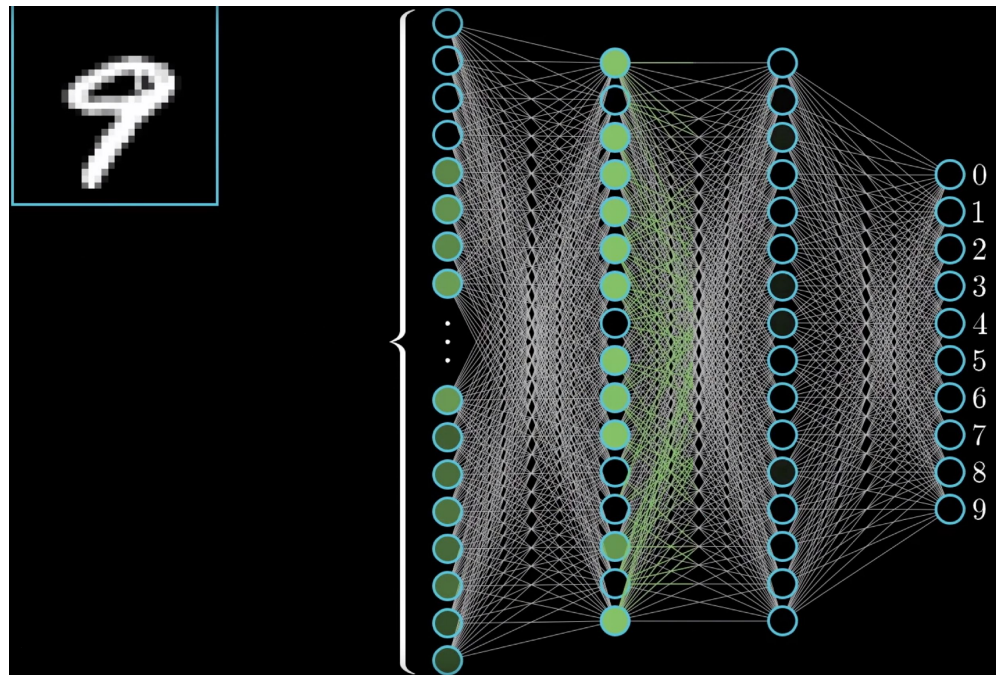
Bootstrap – Popular front-end component library

- Popular front-end component library
- Open source framework for developing with HTML, CSS, and JavaScript.
- Includes HTML and CSS based design templates
 - E.g. typography, forms, buttons, tables, navigation, image, etc.
- Provides supports for **JavaScript** plugins
- Build your app with prebuilt components and plugins
- <https://getbootstrap.com/>



Tensorflow.js

- Machine Learning For Front-End Developers With **Tensorflow.js**
 - Using machine learning in the browser

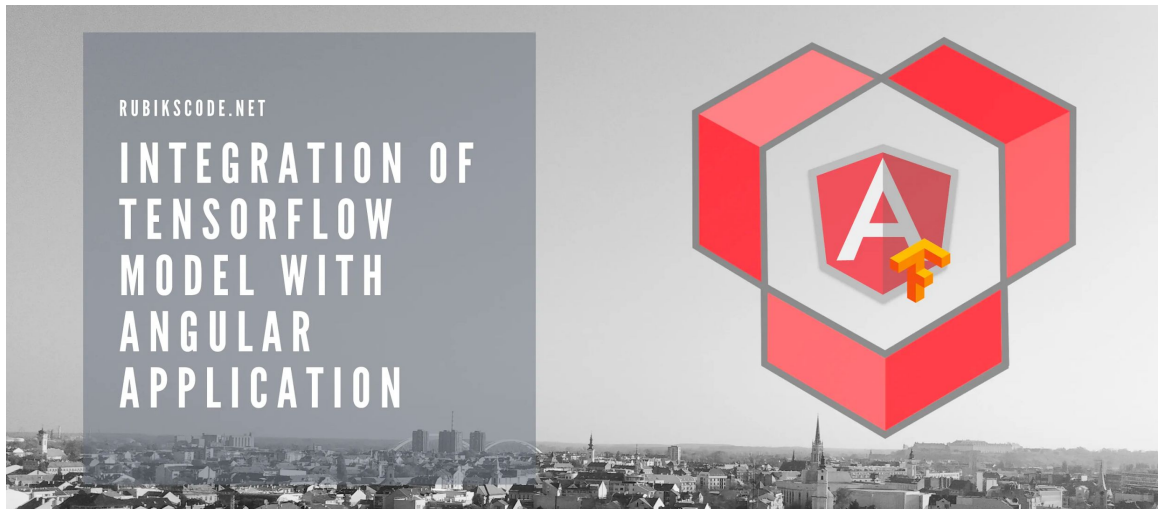


Tensorflow.js

- Build and train **models** in the browser and/or in the *Node.js*:
 - <https://www.smashingmagazine.com/2019/09/machine-learning-front-end-developers-tensorflowjs/>
- Artificial Neural Networks for Total Beginners:
 - <https://towardsdatascience.com/artificial-neural-networks-for-total-beginners-d8cd07abaae4>

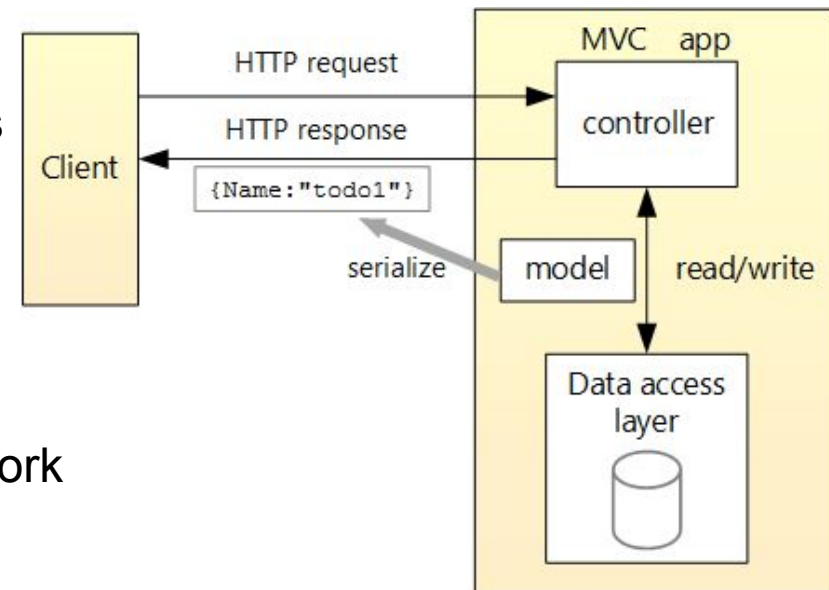
How to Integrate TensorFlow Model in Angular Application?

- <https://rubikscore.net/2019/09/09/integration-of-tensorflow-model-into-angular-application/>



.NET Core

- .NET Core is an **open-source**, **general-purpose** development platform
 - Microsoft and the .NET community
- A **cross-platform**: Windows, macOS, and Linux
 - Build device, cloud, and IoT applications
 - C# applications
- Web API with ASP.NET Core MVC
 - ASP.NET Core is an open-source and cross-platform framework
 - Building web apps and services



Source: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-2.2&tabs=visual-studio>

Visual Studio Code

- A source code editor
 - By Microsoft for Windows, Linux and macOS
 - <https://code.visualstudio.com>
- Support for
 - Debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring
 - Almost **every major programming language**
 - Comparable with with IntelliJ and Eclipse!
- .NET Core
 - A once hopefully future...

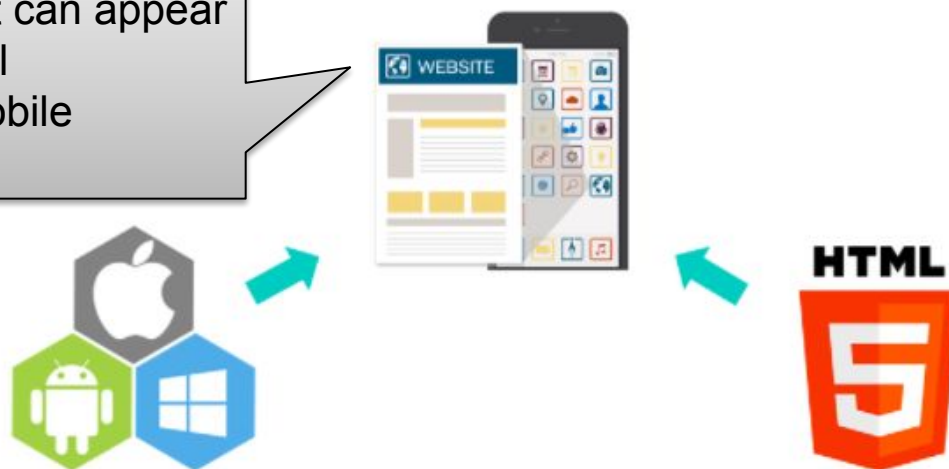


Visual Studio Code

Next Step: Progressive Web App – PWA (1)

- Develop across all platforms
 - Progressive Web App (PWA)
 - Turn websites into native phone and desktop applications

Web applications that are regular web pages/websites, but can appear to the user like traditional applications or native mobile applications!



Next Step: Progressive Web App – PWA (2)

- **PWAs are web applications** that can appear to the user like **traditional applications** or **native mobile applications!**
 - Combines features offered by browsers with the benefits of a mobile experience
 - Let users upgrade web apps to progressive web applications in their native OS
- Native Apps: coded in a programming language like Java
- Traditional Web Apps: coded in standard HTML, CSS, and JavaScript

Next Step: Progressive Web App – PWA (3)

- **PWAs:** visit in a browser tab, no install required!
 - Visit the site, add to home screen,
 - Go to home screen and open site, use the app!
 - Supported by Google Chrome and Mozilla Firefox

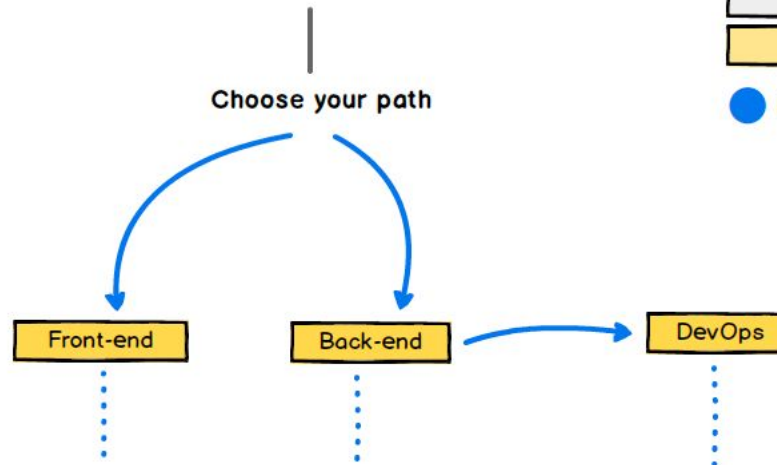


The Web Developer Roadmap (1)

Required for any path

Git - Version Control
SSH
HTTP/HTTPS and APIs
Basic Terminal Usage
Learn to Research
Data Structures & Algorithms
Character Encodings
Design Patterns
GitHub
Create a profile. Explore relevant open source projects. Make a habit of looking under the hood of projects you like. Create and contribute to open source projects.

Web Developer in 2018

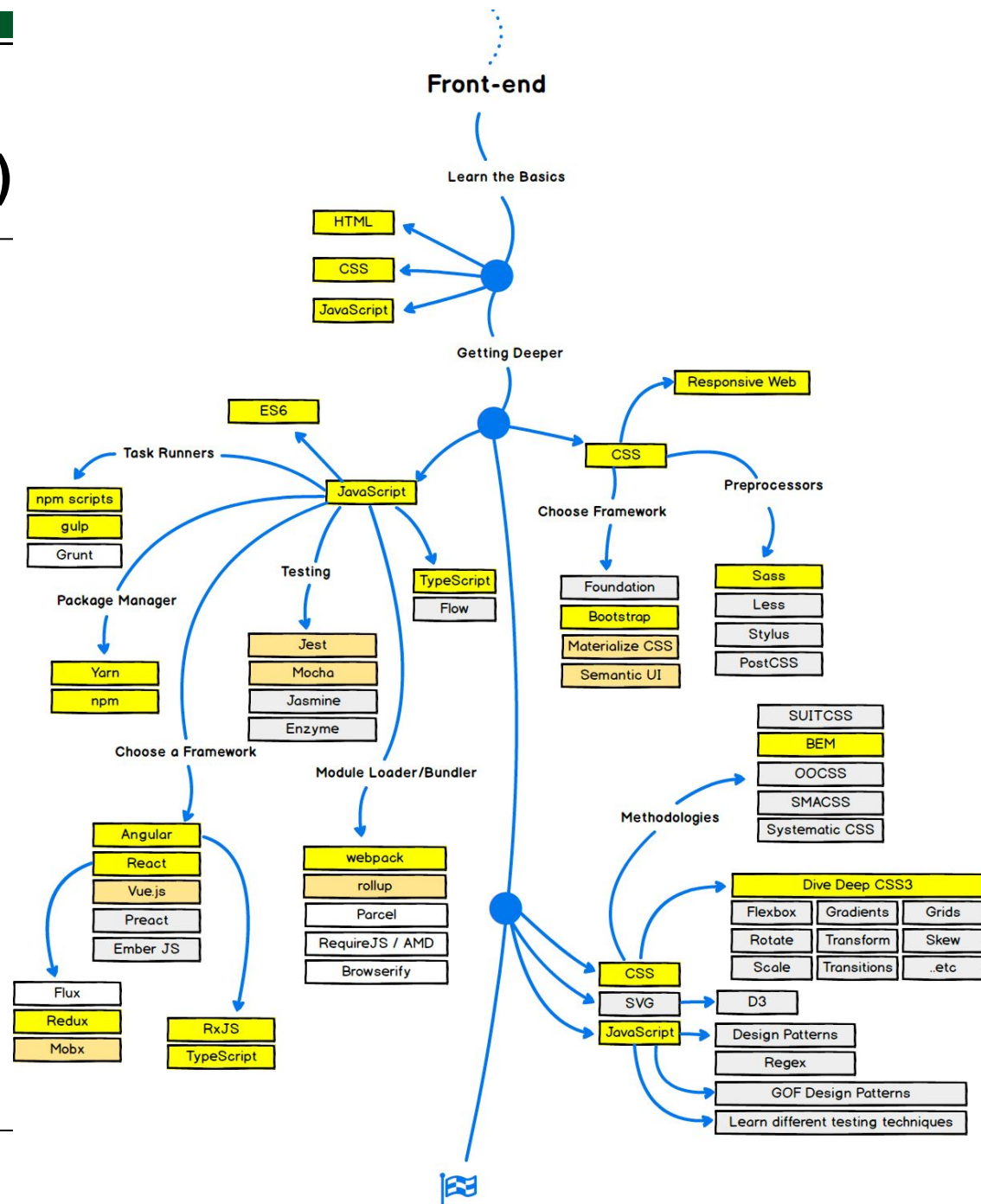


Legends

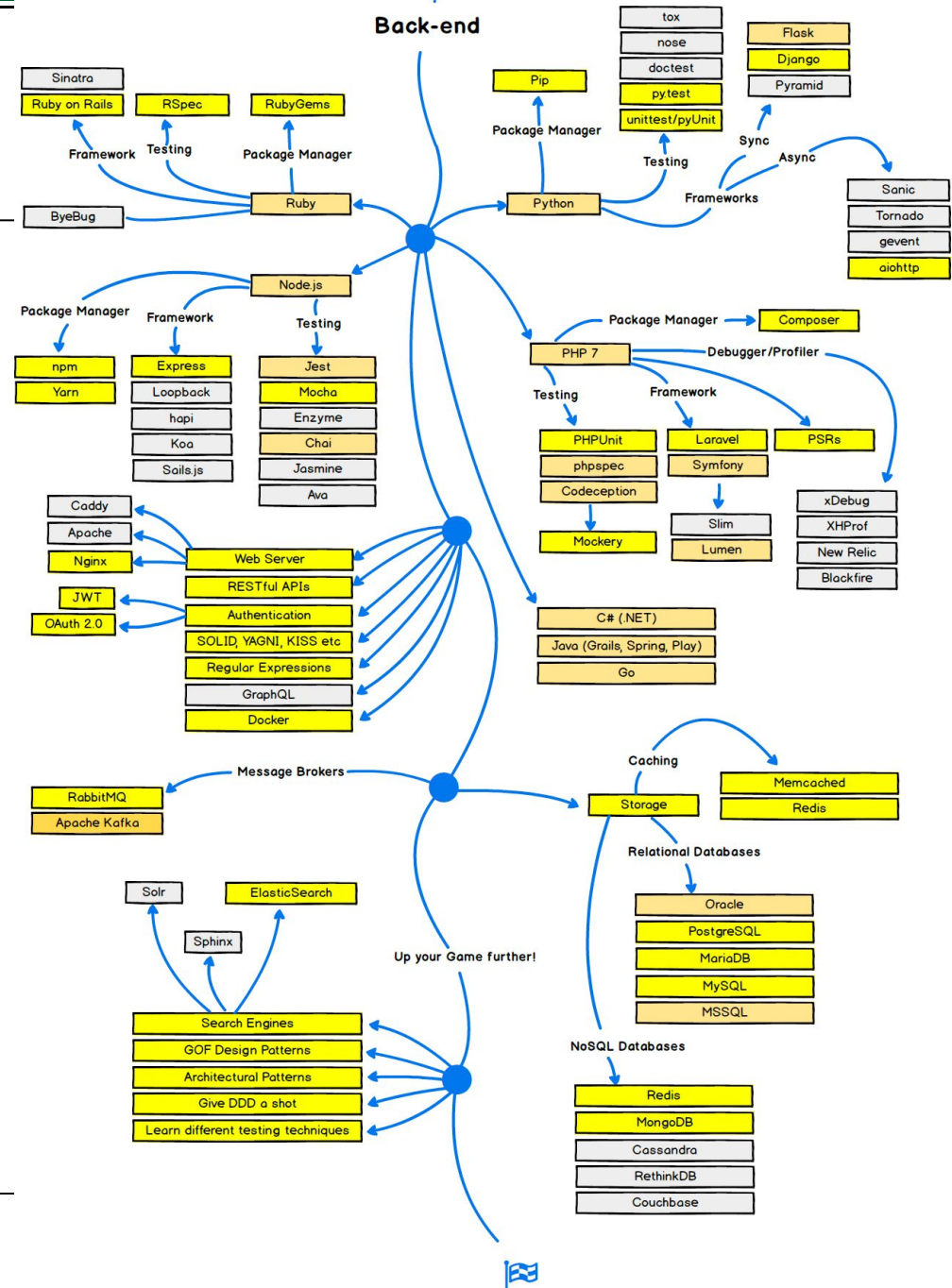
Personal Recommendation!
Possibilities
Pick any!
● Now build something

Source: <https://codeburst.io/the-2018-web-developer-roadmap-826b1b806e8d>
Check roadmap 2020: <https://github.com/kamranahmedse/developer-roadmap>

The Web Developer Roadmap (2)



The Web Developer Roadmap (3)

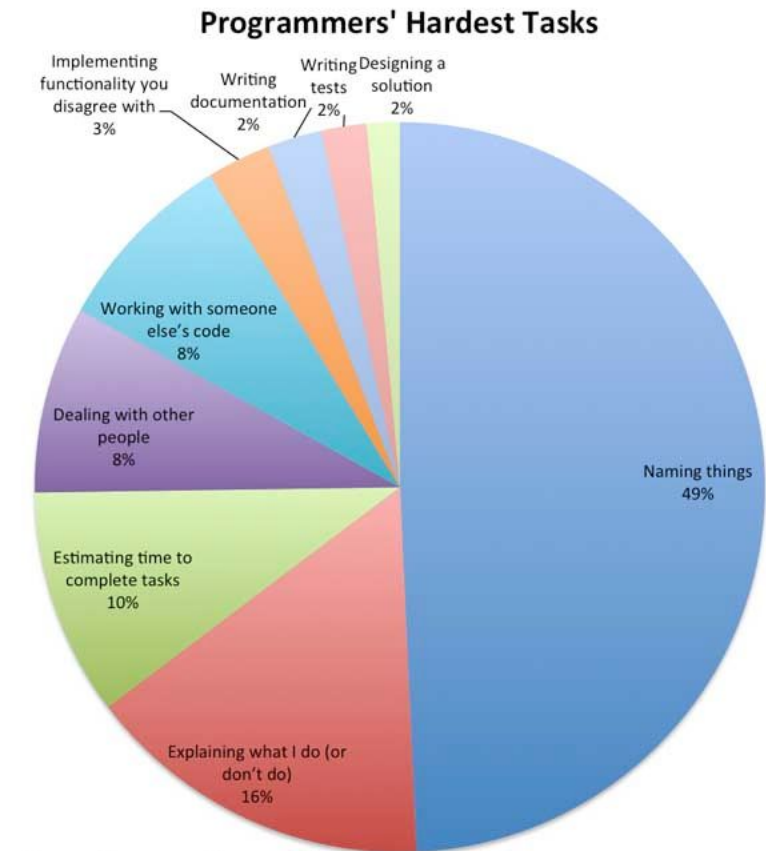


The Future of Web Development

- To have robust and modular apps!
- To never have to write the same code twice!
- To be able to deal with code in an easier and more friendly way!
- To focus more on logic and composition, rather than on the implementation and integrations details of what we're building!
- Writing code is going to get a lot easier, **but**
- **It's our responsibility as software developers to anticipate what's to come and to deliver long term solutions accordingly!**

Final note...

- For **teamwork** consider programmers' harder tasks...



Data Source: Quora/Ubuntu Forums
Total Votes: 4,522



Summary

- Frameworks and API
 - React
 - Singleton pattern in React
- Other frameworks and APIs
- Web development roadmap



Self-study [not relevant for the exam]

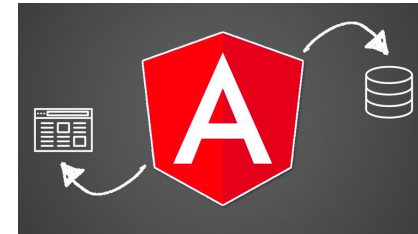
- Following sections/slides are self-study [not relevant for the exam]

JavaScript framework
ANGULARJS



JavaScript framework – AngularJS

- A JavaScript open source framework for developing single page web application (SPAs) using **MVC**
 - Model: Managing application data
 - View: Presenting data in particular format
 - Controller: Handling user input and interacting on the data model objects
- AngularJS extends HTML attributes by **directives**
 - Directives in Angular start with “**ng-**”
 - Directives are used to bind data from application to HTML elements
- Basics of AngularJS: directives, expressions, filters, modules, controllers



AngularJS – Directives

```
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
</script>
<body>
  <div ng-app="app1">
    <p>Name: <input type="text" ng-model="name"></p>
    <p ng-bind="name"></p>
  </div>
</body>
</html>
```

Define module

Define binding data

Define model

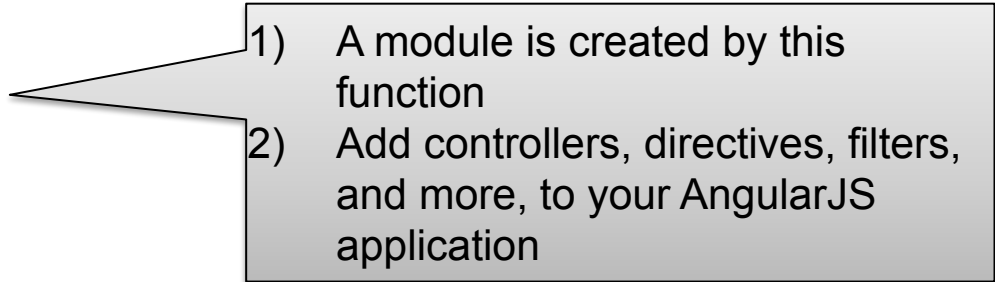
AngularJS – Directives

- **ng-app:** defines an AngularJS application
- **ng-model:** binds the value of HTML controls (e.g. input, select, textarea) to application data
- **ng-bind:** binds application data to the HTML view
- **ng-init:** initializes an AngularJS Application data
- **ng-repeat:** repeats HTML elements for each item in a collection
- Directives can be custom-built from developers

AngularJS – Controller

- Controllers **control the data** of AngularJS applications
- **ng-controller** defines the application controller in HTML
- A controller is a JavaScript object
 - Controllers are created using JavaScript constructors
 - Allocating **properties**' values and **methods**
- To interact with controller, we need to load the application by the following statement:

- *angular.module("myApp", []);*

- 
- 1) A module is created by this function
 - 2) Add controllers, directives, filters, and more, to your AngularJS application

AngularJS – Controller

```
<!DOCTYPE html>
```

```
...
```

```
<div ng-app="myApp" ng-controller="myCtrl"> {{ firstName + " " + lastName }}</div>
```

```
<script>
```

```
var app = angular.module("myApp", []);
```

```
app.controller("myCtrl", function($scope) {
```

```
  $scope.firstName = "John";
```

```
  $scope.lastName = "Doe";
```

```
});
```

```
</script>
```

```
...
```

Controller creates two properties (variables):
firstName and **lastName**

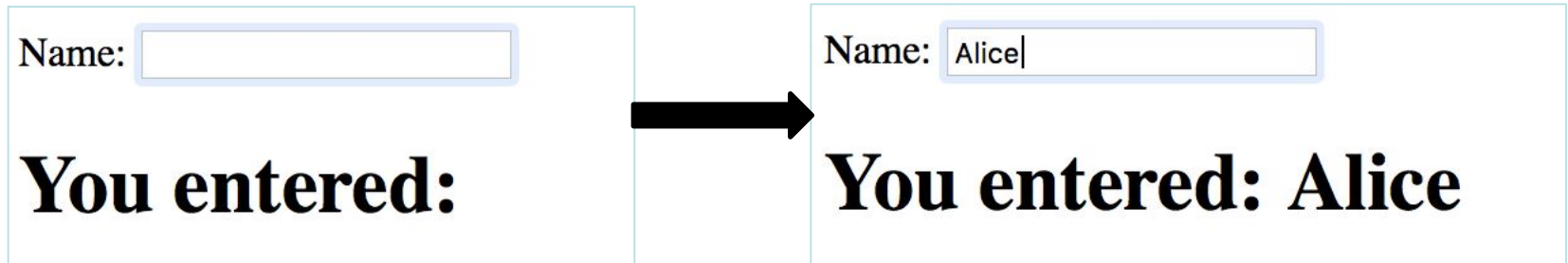
The "myApp" parameter refers to the HTML element in which the application will run

Application object – the owner of application variables and functions

AngularJS – Model

- Provide a mechanism to bind the input value from HTML input tags to variables created by AngularJS
 - **Two way binding:** Both changes of values from input tags and from JavaScript handling are reflected immediately

```
<input ng-model="name">  
<h1>You entered: {{name}}</h1>
```



AngularJS – View

AngularJS defines extension to HTML in:

- *Expression*
- *HTML DOM*
- *HTML Event*
- *Representing List*

AngularJS – Expression

- Writing inside HTML page:
 - Syntax 1: double braces: **{{expression }}**
 - Syntax 2: using directive: **ng-bind="expression"**
- AngularJS expression are much like JavaScript expression in fundamental expressions of JavaScript String, number, array and object
 - However, AngularJS expression **doesn't** support **condition, loop and exception!**
- AngularJS expression supports **filters** (while Javascript doesn't support)

<p>The name is **{{ lastName | uppercase }}**</p>

AngularJS – HTML DOM

- Provide a list of directives to represent popular properties in HTML:
 - **ng-hide** and **ng-show**: alter the visibility of an HTML object
 - **ng-disabled**: disable the input from users in textbox, textarea, etc.
- AngularJS allows to add extended properties for validating the input HTML:
 - Defines a set of **states** for each inputs.

```
<form name="myForm">  
  <input name="myInput"  
    ng-model="myInput" type="email">  
</form>
```

- Valid email type
- Show the **valid** state of input

```
<p>The input's valid state is:</p>  
<h1>{{myForm.myInput.$valid}}</h1>
```


AngularJS – HTML Event

Directives for covering fundamental events in Web programming:

- **ng-click:** defines AngularJS code that will be executed when an element is clicked

```
<button ng-click="count = count + 1">Click me!</button>
```

- **Mouse events:** Support mouse enter, mouse over, mouse move and mouse leave

```
<h1 ng-mousemove="count = count + 1">Mouse over me!</h1>
```

- The event itself can be parsed into JavaScript functions as an argument using **\$event** object

AngularJS – Representing List

- In HTML, there are some elements for representing a list of items:
 - Table
 - Select (ComboBox).
 - AngularJS supports for representing list by **ng-repeat** and **ng-options**
 - **ng-repeat**: use for displaying table's rows, with utilized functions:
 - Ordering by **orderBy** keyword
 - Filtering data
- ```
<table>
 <tr ng-repeat="x in names">
 <td>{{ x.Name }}</td>
 <td>{{ x.Country }}</td>
 </tr>
</table>
```

- **Ng-options**: suitable when we want to bind the selected option as an **object**, instead of displayed value like ng-repeat (has to be a string)!

# AngularJS – Representing List (Example)

ng-repeat:

```
<table>
 <tr ng-repeat="x in names">
 <td>{{ x.Name }}</td>
 <td>{{ x.Country }}</td>
 </tr>
</table>
```

Output:

Alfreds Futterkiste	Germany
Ana Trujillo Emparedados y helados	Mexico
Antonio Moreno Taquería	Mexico
Around the Horn	UK
B's Beverages	UK

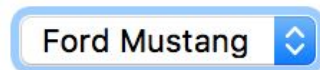
# AngularJS – Representing List

ng-options:

```
<select ng-model="selectedCar"
ng-options="x.model for x in cars">
</select>
<h1>You selected: {{selectedCar.model}}</h1>
<p>Its color is: {{selectedCar.color}}</p>
```

Output:

Select a car:

A screenshot of a web browser showing a dropdown menu. The menu is titled 'Select a car:'. The dropdown list is open, showing 'Ford Mustang' as the selected option. The dropdown button is blue with a white arrow pointing down.

**You selected: Ford Mustang**

# AngularJS Services

---

- A **service** is a function/object
  - Available for, and limited to the AngularJS application
  - There are about 30 built-in services:
  - `$location`, `$http`, `$timeout`, etc.
- **Example:** Built-in `$location` service to get the absolute url of the page:

```
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope,
 $location) {
 $scope.myUrl = $location.absUrl();
});
```

- You can create custom services!

## Services – Example

---

- \$http service requests a page on the server
- The response is set as the value of the "myWelcome" variable:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
 $http.get("welcome.htm").then(function (response
) {
 $scope.myWelcome = response.data;
 });
});
```

## Services – Example: Customized services

---

- Create your own service:

```
app.service('hexafy', function() {
 this.myFunc = function (x) {
 return x.toString(16);
 }
});
```

- Use the service:

```
app.controller('myCtrl', function($
scope, hexafy) {
 $scope.hex = hexafy.myFunc(255);
});
```

# Design pattern – Singleton in AngularJS

---

- **Singleton** is prevalent in AngularJS
- AngularJS **services** are always Singletons
  - Once AngularJS constructs a service object, the same instance is reused throughout your app (there are never ever two service instances!)
  - Great candidates to share application data across multiple components
- **Example**
  - After a successful login: you'll need to store the login status to be used in all other components
  - Store the status in a service: you can just inject it into your controller/service and check it whenever you need!



# Singleton – Design pattern

---

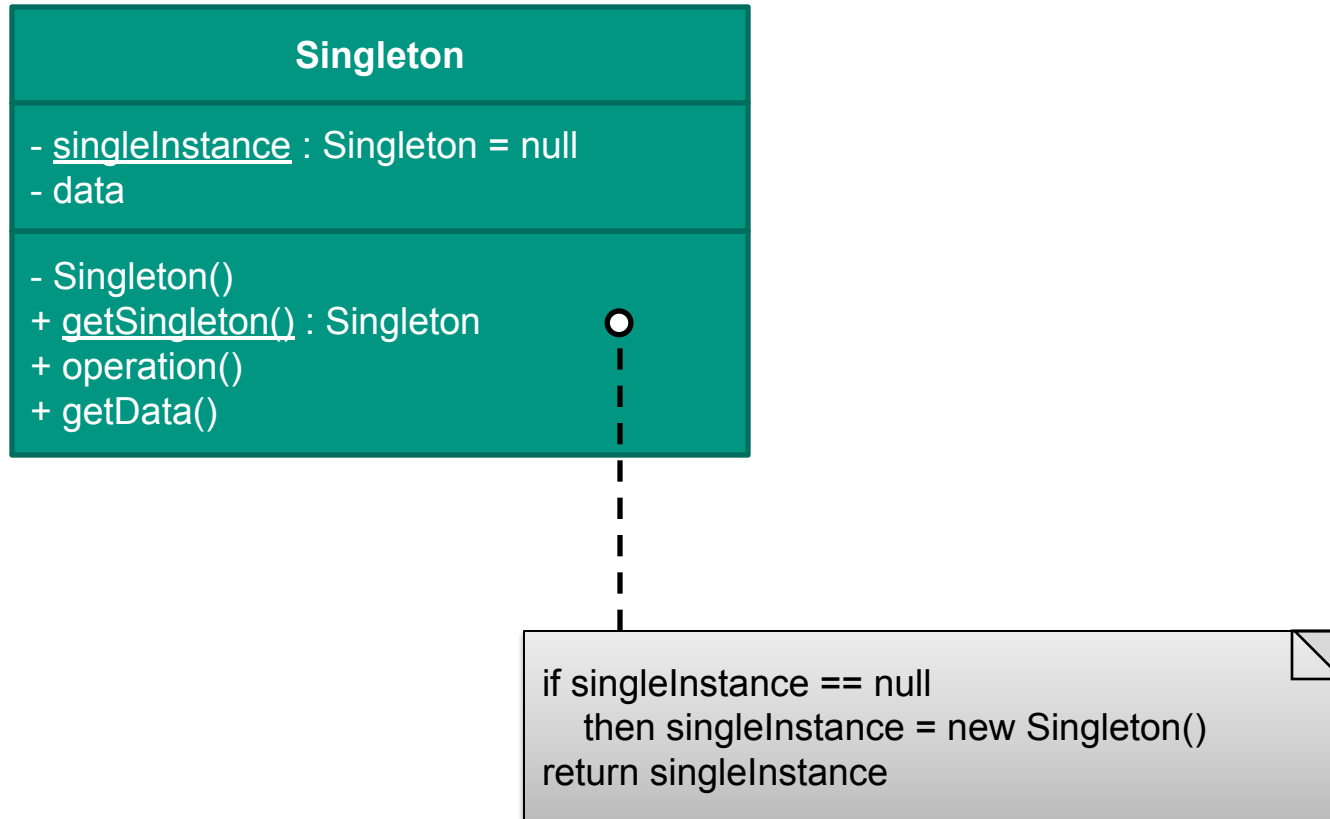
- **Purpose**

- Ensure that a class has exactly one instance and provide a global access point to it

- **Motivation**

- The class is responsible for managing its only copy. By intercepting commands to create new objects, the class can ensure that no additional instance is created.

# Singleton: Structure



# Singleton – Example

---

- **Logger Classes:** The Singleton pattern is used in the design of logger classes (more than one logger instance the log file is a total chaos).
- **Configuration Classes:** The Singleton pattern is used in classes which provides the configuration settings for an application.
- **Java Singleton**
  - Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine.
  - Singleton pattern is used for logging, drivers objects, caching and thread pool.
- A **service** in AngularJS is a singleton
  - AngularJS instantiates the service object only once and all other components share the same instance

# Singleton: Applicability

---

- If there is **only one instance** of a class and this instance is to be made accessible to the client in a known place.
- If it is difficult or impossible to determine which part of the application is creating the first instance.
- If the only instance should be extensible by sub-classing and the clients should use it without changing their source code.

# Singleton in AngularJS

---

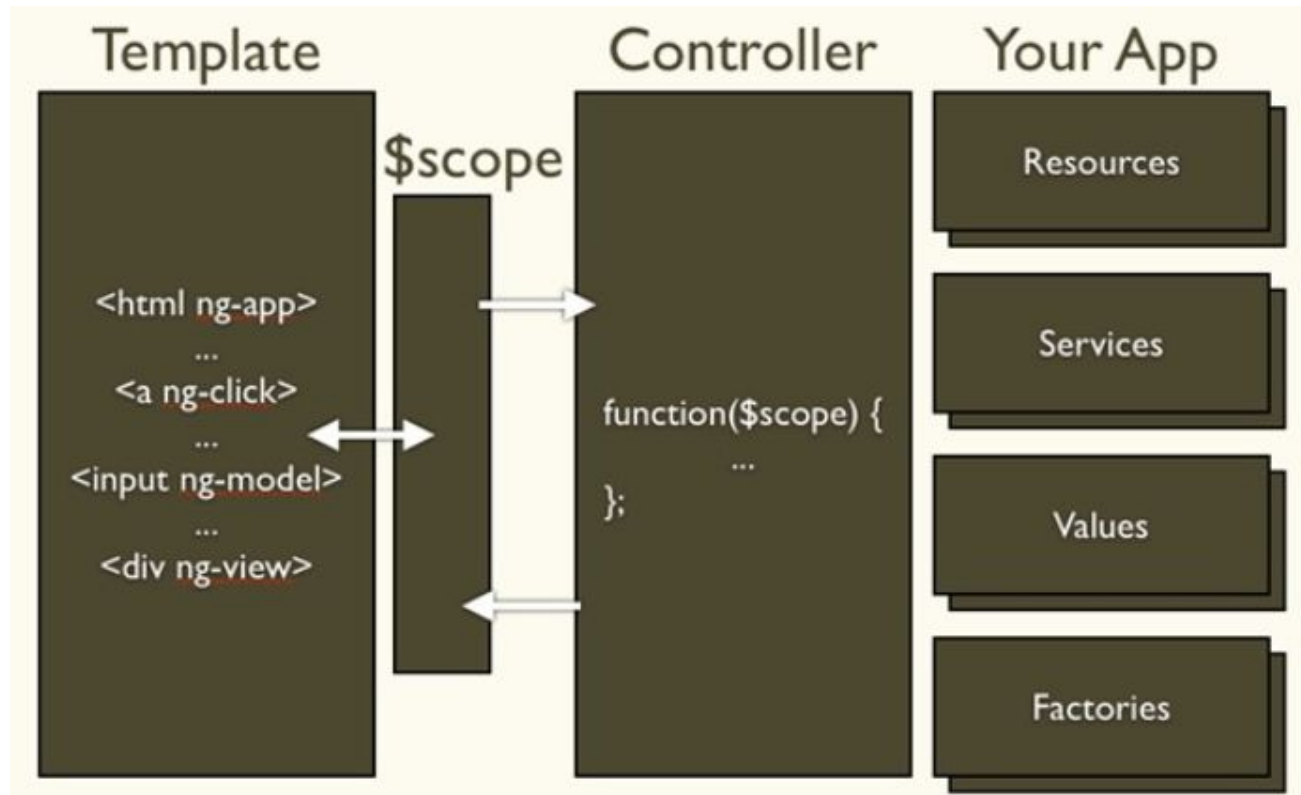
- A service in AngularJS is a singleton JavaScript object which contains a set of functions:

```
function MyService() {
 this.value = 5;
 this.dolt = function() {
 console.log("done "+this.value);
 }
}

var myModule = angular.module("myModule", []);
myModule.service("myService", MyService);
myModule.controller("MyController",
 function($scope, myService) {
 myService.dolt(); // will print "done 5"
 });
```

- AngularJS does it internally: `var theService = new MyService();`

# AngularJS – Layered architecture



# AngularJS – Analytics

## Top Websites Using AngularJS

 google.com	1 TRAFFIC RANK	43.8B MONTHLY VISITS
 youtube.com	2 TRAFFIC RANK	24.6B MONTHLY VISITS
 google.com.br	15 TRAFFIC RANK	3.1B MONTHLY VISITS
 google.co.in	17 TRAFFIC RANK	3B MONTHLY VISITS
 doubleclick.net		2.8B MONTHLY VISITS
 google.co.uk	18 TRAFFIC RANK	2.2B MONTHLY VISITS
 google.co.jp	22 TRAFFIC RANK	1.7B MONTHLY VISITS
 google.de	24 TRAFFIC RANK	1.6B MONTHLY VISITS
 google.ru	29 TRAFFIC RANK	1.6B MONTHLY VISITS
 google.fr	28 TRAFFIC RANK	1.5B MONTHLY VISITS

**325,521 additional websites are using AngularJS**

<https://www.bacancytechnology.com/blog/angular-statistics-infographic>

## Top Website Categories

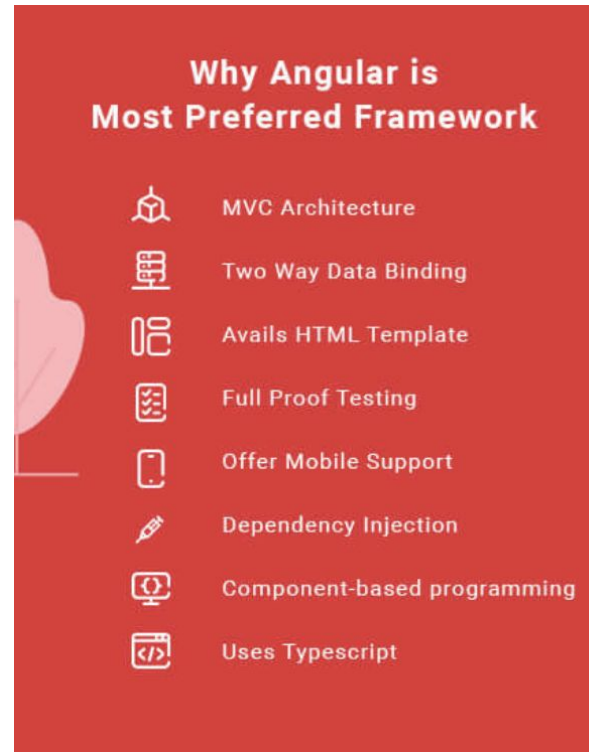
Website categories where Angular is being used



# Why Angular?

---

- MVC
- Two easy binding
- HTML templates



<https://www.bacancytechnology.com/blog/angular-statistics-infographic>



# Literature – AngularJS

---

- <https://www.w3schools.com/angular/default.asp>
- <https://docs.angularjs.org/tutorial>
- <https://plnkr.co/edit/?p=preview>
- <https://www.guru99.com/angularjs-tutorial.html>

---

**Angular JS** is on long term support.

It's been replaced with **Angular**.

**Angular**



# Angular

---

- Angular (currently Angular v7) is a reboot of AngularJS. It's a complete makeover, and not just a following iteration.
- AngularJS long term support ends on june 2021. Until then, Google will only be fixing bugs and making sure AngularJS doesn't break with new browsers/libraries.

# Language

## AngularJS

- Javascript
  - JS is a very powerful web language.
  - Because of its popularity, it introduces a lesser learning curve.

```
var name = "Susan",
 age = 25,
 hasCode = true;
```

## Angular

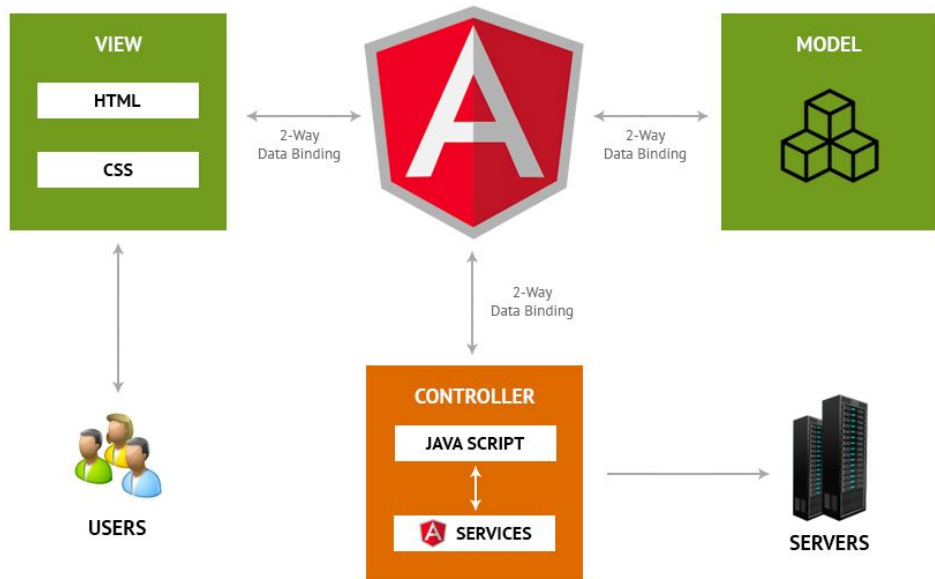
- Typescript
  - Typescript is a JS superset.
  - It introduces static typing.
  - It builds on JS in terms of supporting OOP.
  - It is compiled (transpiled) into JS before running in the browser.
  - Steeper learning curve

```
let name: string = "Susan",
 age: number = 25,
 hasCode: boolean = true;
```

# Architecture

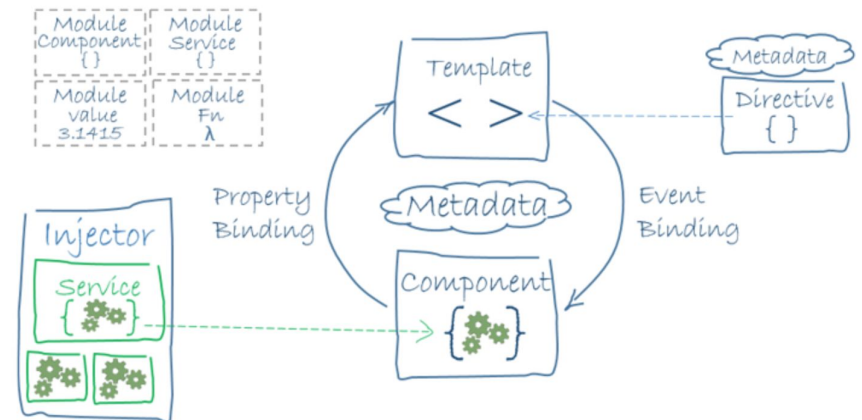
## AngularJS

- MVC (Model-View-Controller)



## Angular

- Component Based architecture.



# Architecture

---

## AngularJS

- Controllers + \$scope

## Angular

- Component and directives replaced controllers + \$scope
- Components are directives + template. A component is usually made up of Typescript, html and css files.

# Template

---

## AngularJS

- HTML + Angular code attributes, markups, filters

## Angular

- Similar in that Angular markup extends html markup
- However each component has its **own template**.
- Uses 3 kinds of directives. Component, attribute and structural.
- Changes have been made to template binding syntax.

# Dependency Injection

---

## AngularJS

- DI is a mechanism where certain functionality (service) is implemented outside the class/function, and then injected into the class/function.
- AngularJS makes use of DI for its controllers.
- Services, directives and filters can be injected by using a factory method.

## Angular

- Similar to AngularJS controllers, components in Angular use DI.
- Newer DI syntax.
- DI is more efficient.



# Extra

---

## AngularJS

- Easier learning curve
- Not built for mobile support
- Slower than Angular
- Third party CLI (Command Line interface)

## Angular

- Steeper learning curve
- Built for mobile support
- Faster than AngularJS
- Own CLI
  
- Scale better (Recommended esp for projects that scale)

## Angular – Self-study [not relevant for the exam]

---

- NgModules - basic building blocks of an Angular application
  - provide a compilation context for components
  - An Angular app is defined by a set of NgModules
- NgModules group components, directives, pipes, and services, which are related to the application.
- Every Angular app has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. An app typically contains many functional modules.

# Angular

NgModule decorator marks the class below as an NgModule

Imported NgModules

```
@NgModule({
 imports: [
 BrowserModule,
 ReactiveFormsModule,
 RouterModule.forRoot([
 { path: '', component: ProductListComponent },
 { path: 'products/:productId', component:
 ProductDetailsComponent },
])
],
 declarations: [
 AppComponent,
 TopBarComponent,
 ProductListComponent,
 ProductAlertsComponent,
 ProductDetailsComponent
],
 bootstrap: [AppComponent]
})
export class AppModule { }
```

- NgModules are importable and exportable. This snippet for instance has imported the *Router* NgModule

Components in this module

Appmodule (Module that is generated by default)


# Angular

---

- **Components** define *views*, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.
- A component consists of three things:
  - A **component class** that handles data and functionality.
  - An **HTML template** that determines the UI.
  - **Component-specific styles** that define the look and feel.

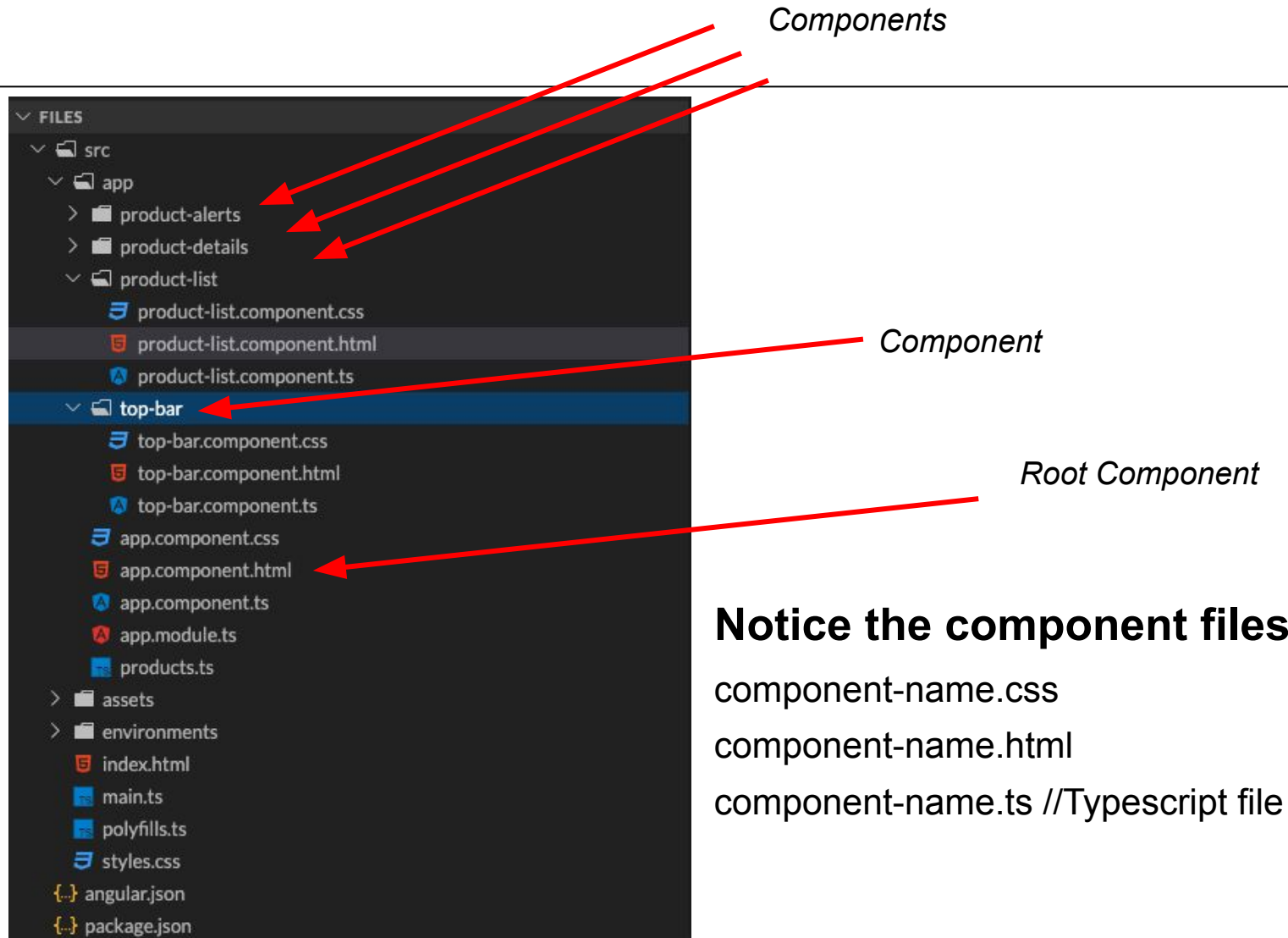
# Angular

- The `@Component()` decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.



```
5 @Component({
6 selector: 'app-product-list',
7 templateUrl: './product-list.component.html',
8 styleUrls: ['./product-list.component.css']
9 })
10 export class ProductListComponent {
11 products = products;
12
13 share() {
14 window.alert('The product has been shared!');
15 }
16
17 onNotify() {
18 window.alert('You will be notified when the product goes on sale');
19 }
20 }
```

# Angular



# Angular

---

- **Template** combines HTML with Angular markup that can modify HTML elements before they are displayed. i.e. A template introduces additional syntax to your HTML.
- Template ***directives*** provide program logic, while ***binding markup*** connects your application data and the DOM.
- There are two types of data binding:
  - ***Event binding*** lets your app respond to user input in the target environment by updating your application data.
  - ***Property binding*** lets you interpolate values that are computed from your application data into the HTML.

# Angular

*Structural directive  
\*ngFor*

```
<div *ngFor="let product of products">
 <h3>
 <a [title]="product.name + ' details'"
 {{ product.name }} $ {{ product.price}}

 <p *ngIf="product.description">
 Description: {{ product.description }}
 </p>

 <button (click)="share()">
 Share
 </button>

 <app-product-alerts
 [product]="product"
 (notify)="onNotify()"
 </app-product-alerts>

 </h3>
</div>
```

- Template syntax extends html tags.

*two-way data binding with  
interpolation symbol {{}}*

*Event binding*

*Child Component*



# Angular

---

- There are **four** forms of data binding **markups**. Each form has a direction: to the DOM, from the DOM, or both.
  - `{{item.name}}` - Display item.name (Direction : to DOM)
  - `[property] = value` - Property binding (Direction : to DOM)
  - `(event) = "handler"` - Event binding (Direction : from DOM)
  - `<input [(ngModel)]="hero.name">` - Combines property + event binding (Direction : Both)
- **Pipes** let you declare display-value transformations (formats) in your template HTML. (Eg.) Use the `|` symbol for piping.
  - `<!-- Default format: output 'Jun 15, 2015'-->`

```
<p>Today is {{today | date}}</p>
```

# Angular

---


- **Directives** are instructions in the DOM. They specify how to place your components and business logic in the Angular.
- Directives are js class and declared as `@directive`. There are 3 directives in Angular.
  - **Component Directives:** Component directives are used in main class. They contain the detail of how the component should be processed, instantiated and used at runtime.
  - **Structural Directives:** Structural directives start with a `*` sign. These directives are used to manipulate and change the structure of the DOM elements. Eg. `*ngIf` and `*ngFor`.
  - **Attribute Directives:** Attribute directives are used to change the look and behavior of the DOM elements. For example: `ngModel` , `ngClass`, `ngStyle` etc.

# Angular

- **Routing** allows navigating between views (components).

```
@NgModule ({
 imports: [
 BrowserModule,
 AppRoutingModule
],
 declarations: [
 AppComponent,
 HeroListComponent,
 CrisisListComponent,
 PageNotFoundComponent
],
 bootstrap: [AppComponent]
})
```

Add the AppRoutingModule module in  
app.module.ts file



```
export class AppModule { }
```

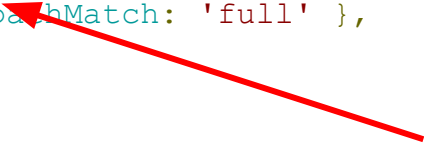
# Angular

---

```
const appRoutes: Routes = [
 { path: 'hero-detail', component: HeroDetailComponent },
 { path: 'heroes', component: HeroListComponent },
 { path: '', redirectTo: '/heroes', pathMatch: 'full' },
];
```

```
@NgModule({
 imports: [
 RouterModule.forRoot(
 appRoutes,
 { enableTracing: true } // <-- debugging purposes only
)
],
 exports: [
 RouterModule
]
})
```

Add routes in the app-routing.module file



```
export class AppRoutingModule {}
```

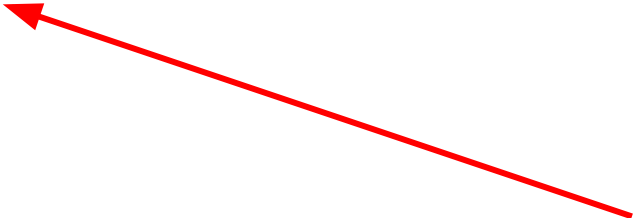
# Angular

---

Notice the routes



```
<h1>Angular Router</h1>
<nav>
 Hero Detail
 Heroes
</nav>
<router-outlet></router-outlet>
```



Finally add router-outlet directive in the template as a spot where the new view will occupy. i.e. the view that comes from clicking the links above.

# Angular

---

- **Services** can help extend the functionality of components.
  - Services are reusable classes/functions/values that deal with specific problems, eg. server communication, form validation, etc.
- Separating Components and Services allows you to further modularize your code. Your components become lean and capable of plugging in and comparing services that do the same job.
  - Services can further depend on other services modularizing your code even more

# Angular

---

- **Dependency injection** (DI), is an important application design pattern.
- DI is a coding pattern in which a class asks for dependencies from external sources rather than creating them itself.
- You can tell Angular to inject a dependency (service) in a component's constructor by specifying a constructor parameter with the dependency type.

# Angular

---

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

```
@Injectable({
```

```
 providedIn: 'root',
```

```
})
```

```
export class HeroService {
```


```
 getHeroes() { return
```

```
 [{ id: 11, isSecret: false, name: 'Dr Nice' },
```

```
 { id: 12, isSecret: false, name: 'Narco' },
```

```
 { id: 13, isSecret: false, name: 'Bombasto' }]]; }
```

```
}
```

 @Injectable decorator used to mark the class under it as a service.



# Angular

---

Component class that  
uses dependency/service

```
export class HeroListComponent {

 heroes: Hero[];

 constructor(heroService: HeroService) {

 this.heroes = heroService.getHeroes();

 }

}
```

Injecting the  
dependency/service

## Literature – Angular self-study

---

- <https://angular-templates.io/tutorials/about/learn-angular-from-scratch-step-by-step>
- <https://www.tektutorialshub.com/angular-tutorial/>
- <https://www.edureka.co/blog/angular-tutorial/>
- <https://stackblitz.com/edit/angular-xhjhsd>



# Thread-based vs. Event-based (Node.js)

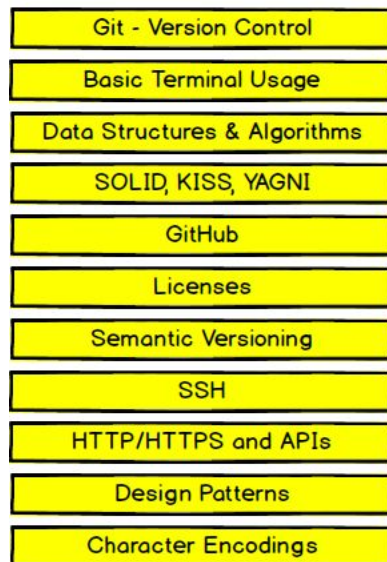
Threads	Asynchronous Event-driven
Lock application / request with listener-workers threads, difficult to program	only one thread, which repeatedly fetches an event
Using incoming-request model	Using queue and then processes it
multithreaded server might block the request which might involve multiple events	manually saves state and then goes on to process the next event
Using context switching	no contention and no context switches
Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock	Using asynchronous I/O facilities (callbacks, not poll/select or O_NONBLOCK) environments

## Conclusion:

- Use threads for performance critical applications (kernels)
- Use events for GUI and distributed systems

# The 2019 Web Developer Roadmap (1)

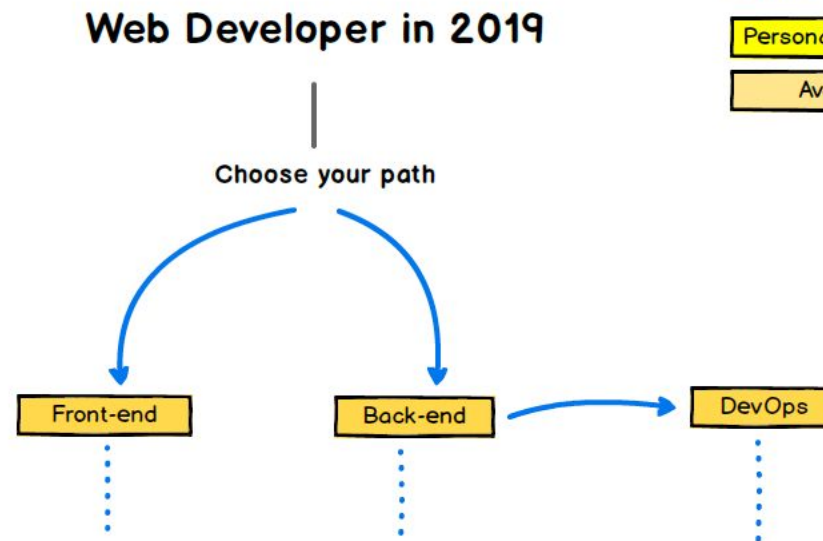
## Required for any path



## Legends

Personal Recommendation!

Available Options



Source: <https://github.com/kamranahmedse/developer-roadmap>

---

# Laravel

---

# Routes

---

Way to redirect the request (or URL) from the client.

In the below example, requests to home will return a welcome view.

```
<?php
Route::get('/', function () {
 return view('welcome');
});
```

1

```
Route::get('/myStuff', 'MyController@getTableData');
```

2

# Controllers

---

1. This is where the main work takes place
2. It may make calls to the database
3. It may make calls to some processing modules
4. It may make calls to create some views
5. and then returns the results back to the client.



## Controllers – Example

---

```
<?php

namespace App\Http\Controllers;

use Illuminate\Foundation\Bus\DispatchesJobs;
use Illuminate\Routing\Controller as BaseController;
use Illuminate\Foundation\Validation\ValidatesRequests;
use Illuminate\Foundation\Auth\Access\AuthorizesRequests;

class MyController extends BaseController
{
 public function getTableData(){
 echo json_encode(\App\testTable::all());
 }
}
```

# Eloquent ORM – Object Relational Mapper

---

- Provides a simple ActiveRecord implementation for working with a database.
- Each database table has a corresponding "Model" which is used to interact with that table
  - Models data
  - Simple access to data

## Model – Example

---

```
<?php

namespace App;

use Illuminate\Foundation\Auth\User as Authenticatable;

class TestTable extends Authenticatable
{
 protected $table = 'testTable';
 public $timestamps = false;
}
```

# Configuration file (.env file) – Example

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=2uUDtMdi5ldsgHNQjdHQAyUwRHgWfjmW
APP_URL=http://localhost
```

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=testSchema
DB_USERNAME=testuser
DB_PASSWORD=testuser
```

```
CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync
```

```
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
```

```
MAIL_DRIVER=smtp
MAIL_HOST=mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
```

**in config/database.php**

```
'default' => env('DB_CONNECTION', 'mysql'),
```

# Additional Tools

---

- Blade: UI templating tool
- Artisan: Utility tool to help developer
  - Tinker: Tool to help developer run and try methods on server side
  - Laravel artisan's tinker: a **repl (read-eval-print loop)**
    - An interactive language shell
    - It takes in a single user input, evaluates it, and returns the result to the user
    - Quick and easy way to see the data in your database:

<https://scotch.io/tutorials/tinker-with-the-data-in-your-laravel-apps-with-php-artisan-tinker>

```
// find a specific user and see their attributes
App\User::where('username', 'samuel')->first();
```

```
// see the count of all users
App\User::count();
```

- Composer: Dependency manager for php (similar to make tool)

# Literature – LARAVEL

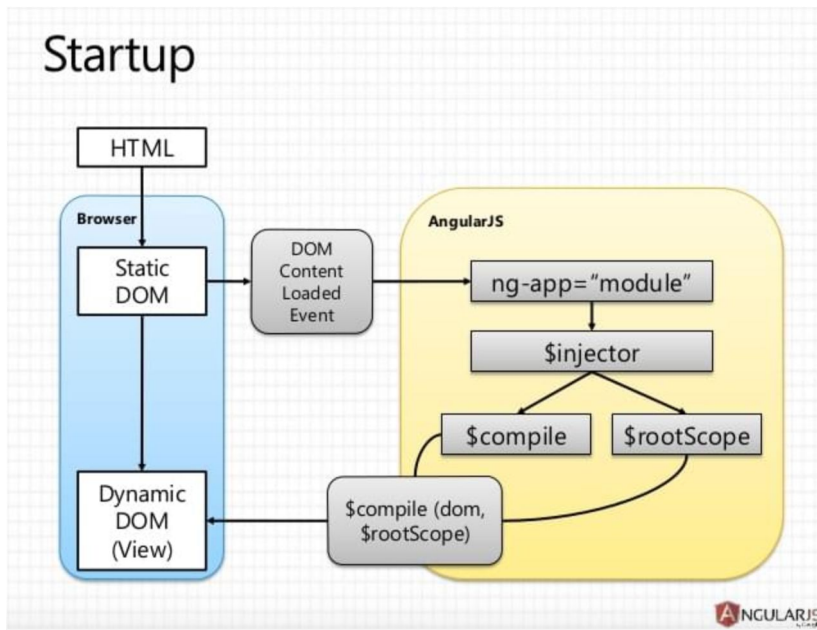
---

- Documentation
  - See <https://laravel.com/docs/5.3/> (incl. installation)
- BEST RESOURCE FOR LEARNING LARAVEL
  - <https://laracasts.com/series/laravel-5-from-scratch> (there are 18 screencasts walking you thru the materials)

# Architecture

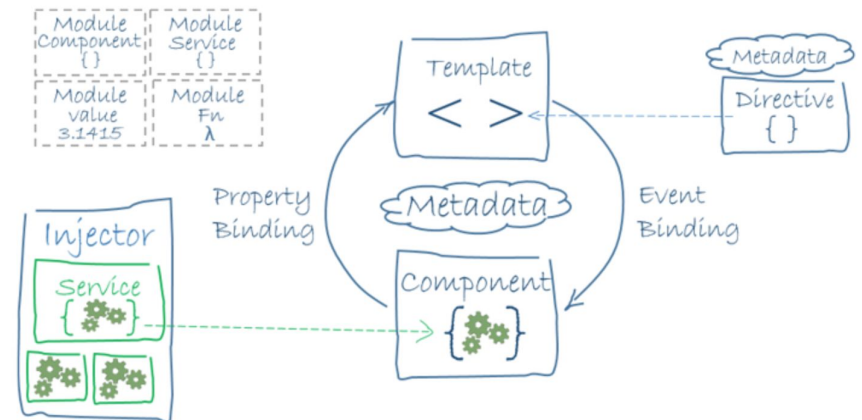
# AngularJS

- MVC (Model-View-Controller)  
or MVVM  
(Model-View-View-Model)



# Angular

- Component Based architecture.

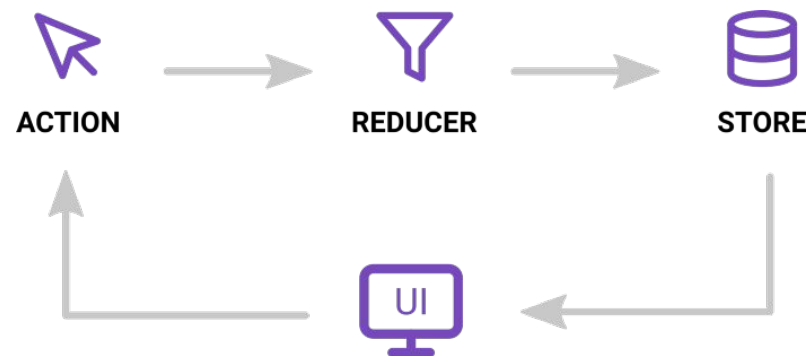


# Redux

- Redux is a popular JavaScript library for managing the state of your application.
- Redux stores the whole state of the app in an immutable object tree.

## The three building blocks of Redux

1. Store
2. Actions
3. Reducers





# Store

---

- The store holds the state of the application.
- It is form of a JavaScript object.
- We only have one store per application.
- It is possible to subscribe to events whenever the store updates.

# Actions

---

- Actions are JavaScript objects that describe what has happened.
- We send (dispatch) the actions to the Store instance when we want to update the state of our application.
- The dispatched actions is handled by the reducers.
- Redux requires action objects to contain a **type** field to describe the type of action we are dispatching.

```
{
 type: ADD_NEW_NOTE,
 title: 'Note Title',
 content: 'The content of the note'
}
```

Imagine we are developing a note taking app. This could be the action that will be dispatched when the user clicks on the 'Add note' button.

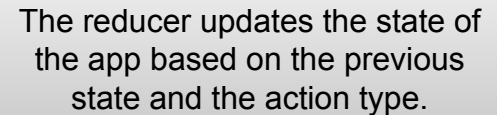
# Reducers

---

- Reducers are functions that define how the app state changes.
- Whenever we dispatch an action to our store, the action gets passed to the reducer.

The reducer function takes two parameters:

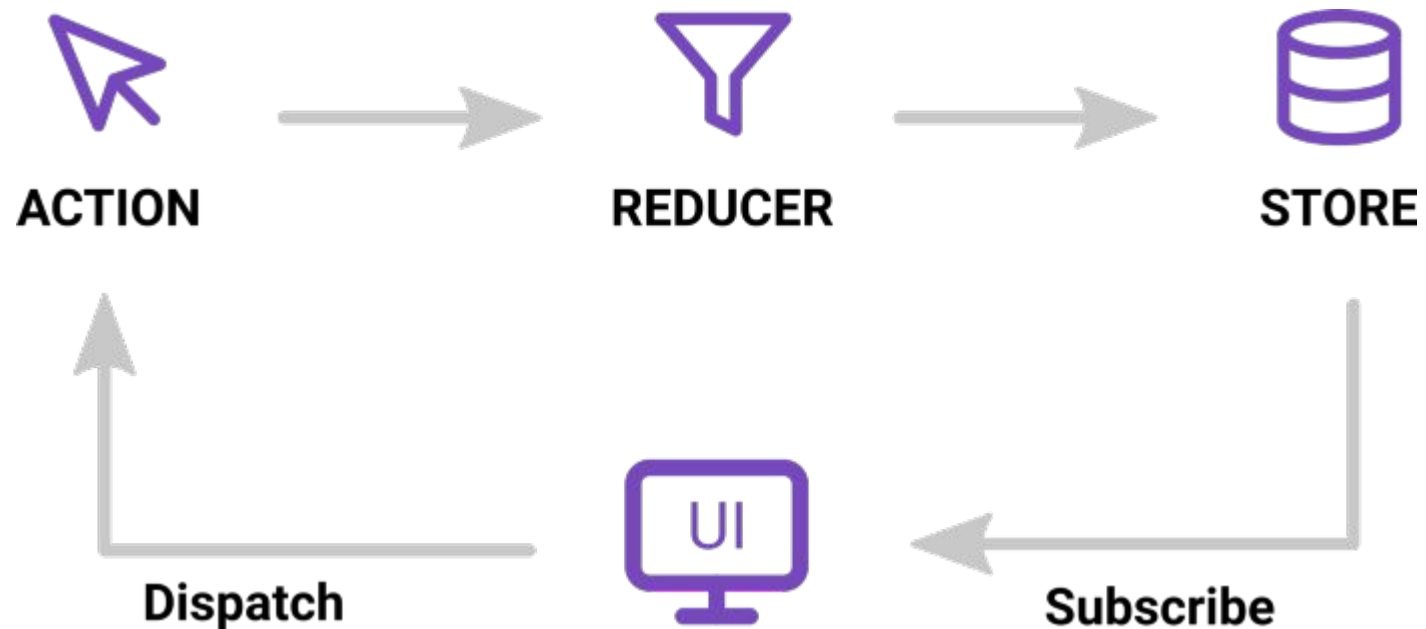
- The previous app state
- the action being dispatched



The reducer updates the state of the app based on the previous state and the action type.

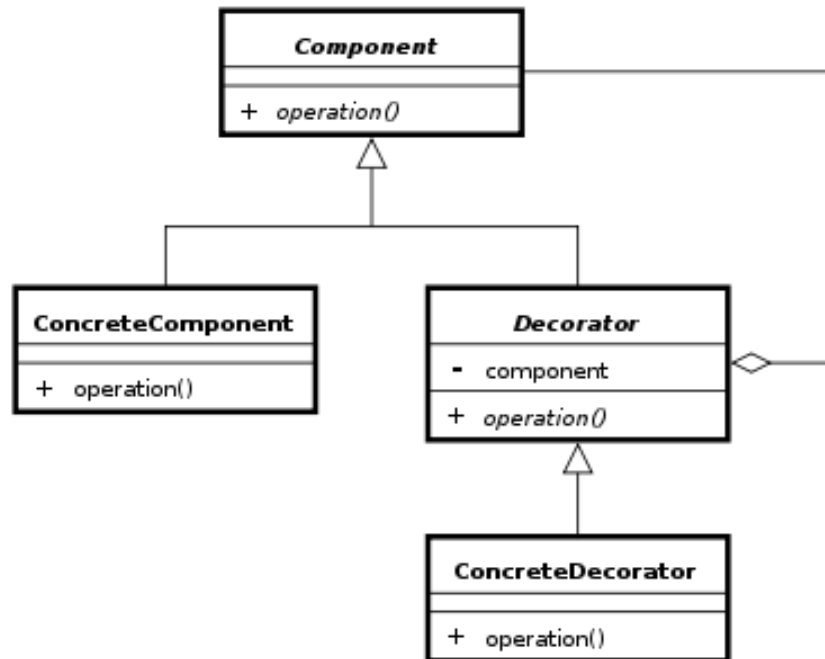
```
(previous State, action) => newState
```

# The Data Flow of Redux



# Design Pattern - Decorator

Decorator pattern allows to attach additional responsibilities or behavior to an object dynamically without affecting the behavior of other objects from the same class.



# An Example of Decorator Pattern in React

Decorator  
function

calling the function  
of the original  
component

The new behaviour  
that we want to add

The component that  
we want to  
decorate it

```
function deprecatedFn(target, name, descriptor) {
 const newDescriptor = { ...descriptor };
 const originalFunction = descriptor.value;
 function deprecatedFunctionWarning(args) {
 originalFunction.call(this, ...args);
 if(__DEV__) {
 console.warn("This function is about to be Deprecated");
 }
 }
 newDescriptor.value = deprecatedFunctionWarning;
 return newDescriptor;
}

class SomeOldCodeBaseClass extends React.Component {
 @deprecatedFn
 someFunction() { // Some deprecated logic }
}
```

## React Singleton: When to Use

---

- While working on an application that has a dozen teams working on it across dozens of repositories. Each team manages different pages of the application and we all write shared components for others teams to use on their pages.
- Provider and broadcast patterns couple the composition of individual components to the full application; this is a major hurdle (full of frustration, lost time, and wasted money) every time a team wants to alter how it manages its data.
- With a shared global Redux store, teams also have the ability to carry that on top of Redux actions, even though those actions can change at any time in the future.

# React Singleton: How to Use

- We can use Singleton: encapsulated, simple, readable way to provide shared information across a multitude of components.
- The way to install a React Singleton [1] module with React hook\* that synchronizes state across all instances of a component:

```
npm install --save-dev https://github.com/peterbee/react-singleton.git
```

- When *updateMyData* is called, every component that uses the hook, rerenders with the updated value. It does not need to trace through complex call stacks and data stores to figure out which action needs to be dispatched or which listeners are subscribed to which channels.

```
import createSingleton from '@peterbee/react-singleton';

const [useMyData, updateMyData] = createSingleton('initial value');

function MyComponent() {
 const myData = useMyData();

 return myData;
}
```

*\* Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.*



# React Singleton: How to Use

- For instance, we can use React Singleton to manage the logged-in user account in an application.
- Some portions of the application require to know if a user is logged in, who that user is, and what preferences they have configured. Any component throughout the application can import a singleton state hook.
- A way to create the hook in a file that owns account data:

```
// useAccount.js
import createSingleton from '@peterbee/react-singleton';

const [useAccount, updateAccount] = createSingleton(accountObject);

export default useAccount;
```

- A way to use the hook in any component that needs account data:

```
import useAccount from './useAccount';

function Profile() {
 const account = useAccount();

 return (
 <div>Hello {account.first_name}!</div>
);
}

export default Profile;
```