AngularJS Tutorial

What every Java developer should know about AngularJS

Edgar Mueller, Maximilian Koegel

eclipsecon Europe 2016

Setup 1/5: Copy/download required files

- **Java:** If you do **not** have Java version 7 or higher on your computer:
  - Copy/Download a Java Installer for your OS
  - Please do **not** install now, but continue with next step
- **NodeJS:** If you do **not** have NodeJS version 4.x or higher installed on your computer:
  - Copy/Download NodeJS installer for your OS
  - Please do **not** install now, but continue with next step
- **Eclipse:** Copy/Download an Eclipse zip for your OS
  - Please do **not** use your own existing Eclipse installation
  - Please do **not** unzip now, but continue with next step
- **Browser:** If you do not have a current version of Chrome or Firefox installed:
  - Copy/Download a Chrome Installer for your OS
- **Code Examples:**
  - Copy/Download the TutorialExamples.zip
  - Please do **not** unzip now, but continue with next step
- You have all required files now: **Before** you continue, give back or pass on the USB stick so others can also use it, if applicable :)

Setup 2/5: Tooling Installation

- Install Java if required
- Extract and launch Eclipse (installing closer to a root folder is better on Windows OS)
- *OSX users*: open via context-menu (since the provided Eclipse instance is unsigned)
- Extract TutorialExamples.zip into a folder of your choice, which we refer to as the tutorial folder
- Install Node if required
  - Only on Ubuntu/Debian:
    - If node command points to wrong executable, either :
      - Install nodejs-legacy
      - Or run after installation: `sudo ln -s /usr/bin/nodejs /usr/bin/node`
      - Or look at this SO question
- Activate native refresh hooks in Eclipse:
  - Window → Preferences → General → Workspace → Refresh using native hooks or polling

Setup 3/5: Import example

- Import first project into workspace from `ng-tutorial-exercise-1.zip`
  - *Do **not** select File → Import → Archive…*
  - ***but** select File → Import → General → Existing Projects… → Select archive file → Finish*

- Adapt Karma Test Runner configuration
  - Open `karma.conf.js`
  - Change the value of the `browsers` property to your Browser
    - Chrome: `browsers:  ['Chrome']`
    - Firefox: `browsers:  ['Firefox']`

Setup 4/5: Check build and test tooling

- The good news: Eclipse JS Tooling has been improved a lot recently 😄 👍
- However: It’s still WIP and we might need to use the CLI
- If everything works as intended:
    - Open console (Project Context Menu → Show In → Terminal)
    - On Linux/Mac you might need to run: `chmod -R +x ./node_modules/`
    - To test
        - Run: `npm run test`
        - If the previous command failed, run alternatively:

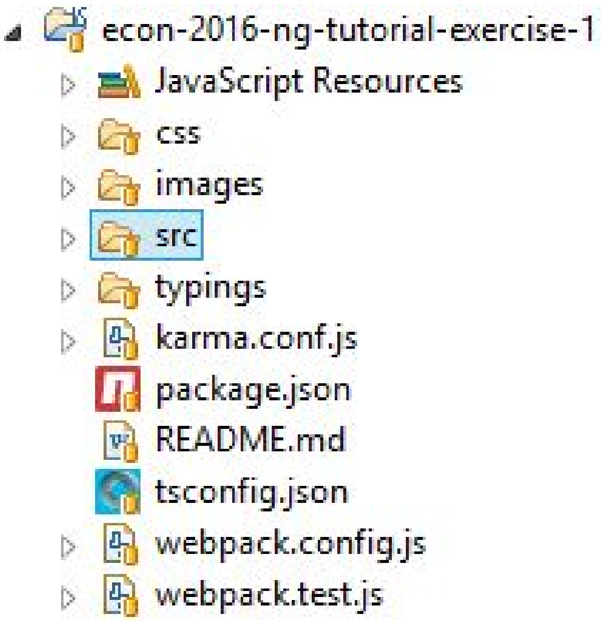
          ```
          ./node_modules/.bin/karma start
          ```

        - output should look like:

          ```
          Executed 3 of 3 (1 FAILED) (0.036 secs / 0.023 secs)
          ```

Setup 5/5: Final check

- Eclipse is running and your workspace should look like screenshot on the right (no errors):
  - If you see Errors, hit F5 to refresh and wait for rebuild



```
econ-2016-ng-tutorial-exercise-1
  JavaScript Resources
  css
  images
  src
  typings
  karma.conf.js
  package.json
  README.md
  tsconfig.json
  webpack.config.js
  webpack.test.js
```

AngularJS Tutorial

If you followed the preparation steps:

- You are all done :)

Otherwise:

- Raise your hand and get a USB Stick
- Copy only “Slides.pdf” from the Stick onto your computer
- Open it and follow the instructions on Setup starting on slide 2 (Setup 1/5)

AngularJS Tutorial

What every Java developer should know about AngularJS

Edgar Mueller, Maximilian Koegel

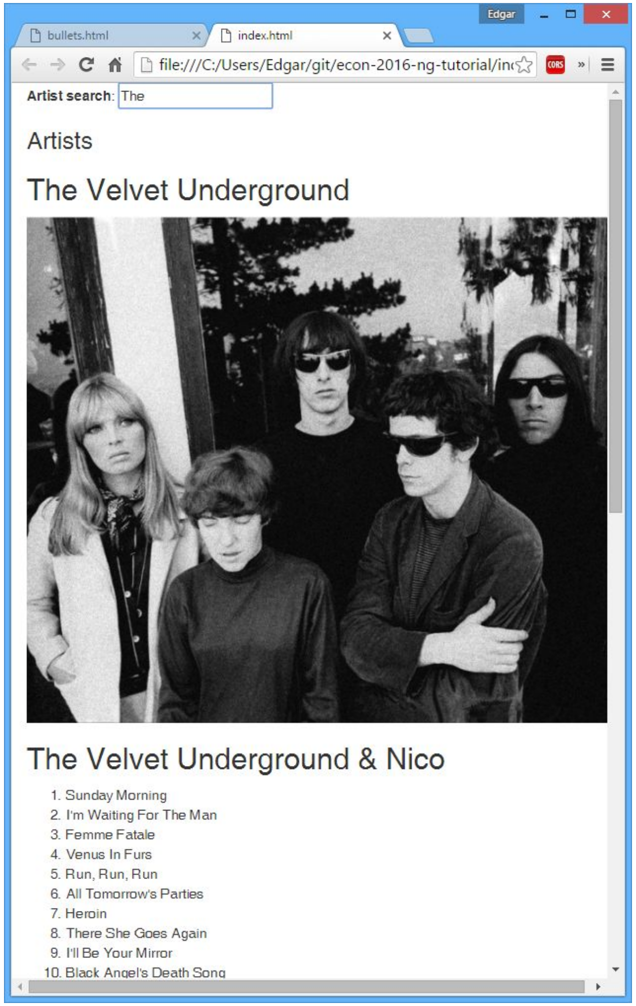
Agenda

1. General Introduction
2. Typescript
   a. Introduction
   b. Exercise 1
3. AngularJS
   a. Introduction
   b. Exercise 2
   c. Exercise 3

Introduction

What we’re about to build

- A sample application for displaying artists and their albums
- We’ll iterate multiple times to introduce new features



Artist search: The

Artists

The Velvet Underground

The Velvet Underground & Nico

1. Sunday Morning
2. I'm Waiting For The Man
3. Femme Fatale
4. Venus In Furs
5. Run, Run, Run
6. All Tomorrow's Parties
7. Heroin
8. There She Goes Again
9. I'll Be Your Mirror
10. Black Angel's Death Song

Building the Demo Application

- Exercise 1: Backend
  - Implement a findByName method in Repository
  - Can be used to find artists by name

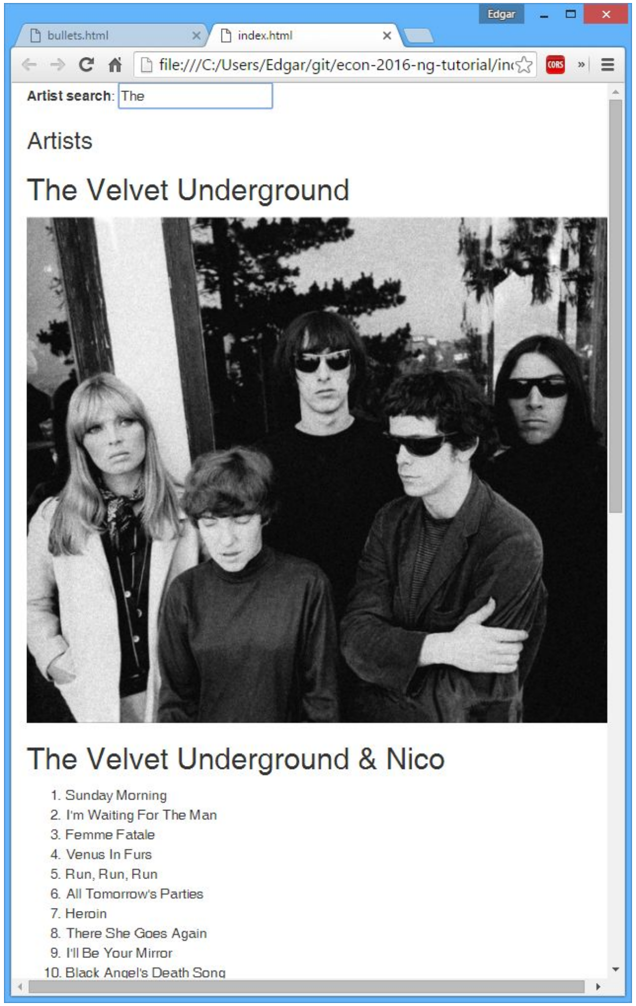
- Exercise 2 - Artist List
  - Show a list of all artists
  - Show each artist with name, image and album list

- Exercise 3 - Searchable Artist List
  - Add search box on top of artists-list where a search string can be entered
  - Change artist-list directive to show only artists containing the current search string

Why dynamic HTML in the browser?

- Users expect desktop-app like experience
  - responsiveness
  - rich user experience
- Platform independence

  → the browser is your VM
- No client deployment



Artist search: The

Artists

The Velvet Underground

The Velvet Underground & Nico

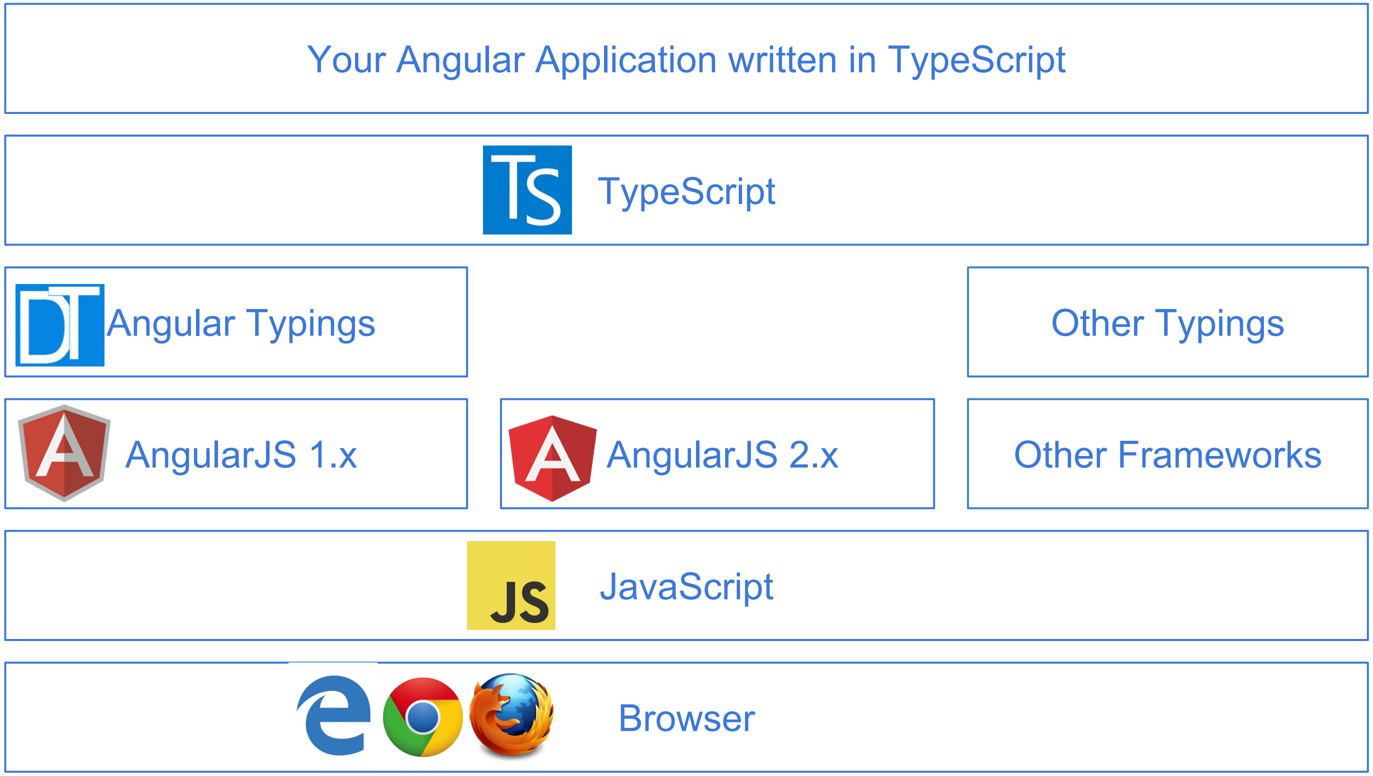
1. Sunday Morning
2. I'm Waiting For The Man
3. Femme Fatale
4. Venus In Furs
5. Run, Run, Run
6. All Tomorrow's Parties
7. Heroin
8. There She Goes Again
9. I'll Be Your Mirror
10. Black Angel's Death Song

Why AngularJS?

- Framework for declaring expressive dynamic views in HTML
- Framework with broad functionality → RCP for web applications
- Super-mature framework (for JavaScript Standards)
- Broad user-base with many deployments
- Good documentation
- Superheroic :)

https://commons.wikimedia.org/wiki/File:Superhero.svg

Stack



Your Angular Application written in TypeScript

TypeScript

Angular Typings

Other Typings

AngularJS 1.x

AngularJS 2.x

Other Frameworks

JavaScript

Browser

Tools

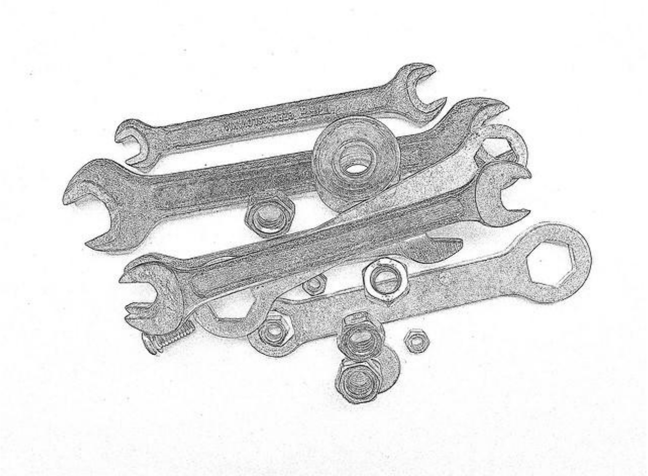
- Jasmine:
  - Testing
  - Comparable to JUnit
- Webpack
  - Bundling
  - Usage of NPM packages in frontend code
  - Builds ‘product’
- NPM (Node.js Package Manager)
  - Node.js is a runtime for JavaScript Applications
  - NPM is a package management tool
  - Resolves ‘Target Platform’
  - NPM Build Scripts

Tooling for Eclipse

Obviously: Eclipse IDE for JS and Web Developers

Typescript

- **Palantir TypeScript**
- http://typecsdev.com/
- Typescript.java



Angular:

- AngularJS Eclipse
- Angular 2 Eclipse

TypeScript

Why JavaScript is strange (sometimes)?

https://www.destroyallsoftware.com/talks/wat

`[] + []`: empty Array + empty Array

→ `""`: (empty) string

`[] + {}`: empty Array + empty Object

→ `"[object Object]"`: string

`{} + []`: empty Object + empty Array

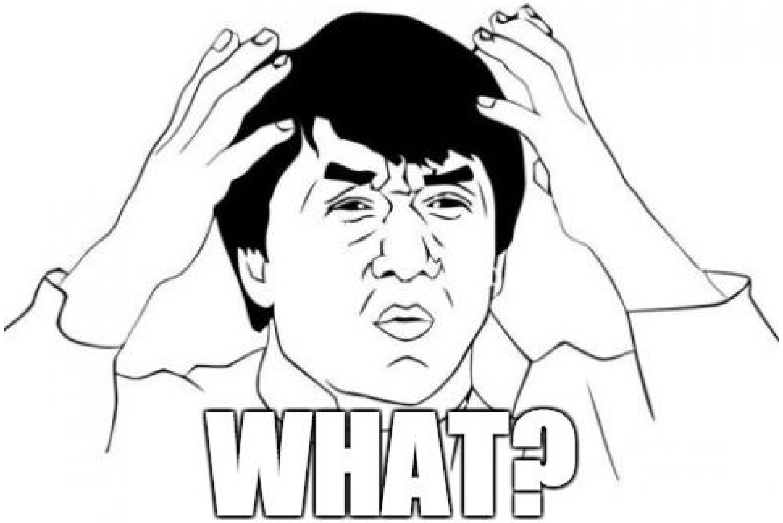
→ `0`: number

`{} + {}`: empty Object + empty Object

→ `NaN`: not a number

`Array(16).join("wat" - 1) + "a Batman!"`

→ `"NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNa Batman!"`: Batman Jingle



What's wrong with JavaScript?

- Dynamic typing
- Implicit type conversion
- Prototype inheritance: No polymorphic object model
- Messed up scope concept
- No built-in modularity concepts (not even packages)
- No good semantics of "this"
- No access modifiers
- ....

→ Many Java Devs don't like JavaScript

Typescript Features 1/2

- Obviously: Types
  - type annotations
  - compile-time type checking
  - type inference

```
let bar: boolean = true
bar = 3// fails
let bar = true
```

- String interpolation

```
let world = "EclipseCon";
console.log(`Hello ${world}`)
```

- Interfaces
- Generics

```
interface IRepository<T extends CommonEntity>
```

- Default values and optional types

```
findById(id: number = 42) { … }
initAlbum(artistId: number, name: string, year?: number) { … }
```

Typescript Features 2/2

- Lambdas: Fat arrow syntax
  - `this` keyword bound properly
  - implicit return
  - like Java 8 Lambdas but with “`=>`” instead of “`->`”

```
[1,2,3,4].map(n => n + 1) // returns [2,3,4,5]
```

- Namespaces/Modules (via external module loader)
- Superset of JavaScript → any JavaScript is valid TypeScript
- Typescript is compiled to JavaScript but output is still quite readable

→ feels way more natural for Java developers than native JavaScript

Typescript Classes

```
class Album {

    constructor(
     artistId: number,
     name:     string,
     tracks:   string[],
     year?:    number) { }
}
```

Typescript Classes

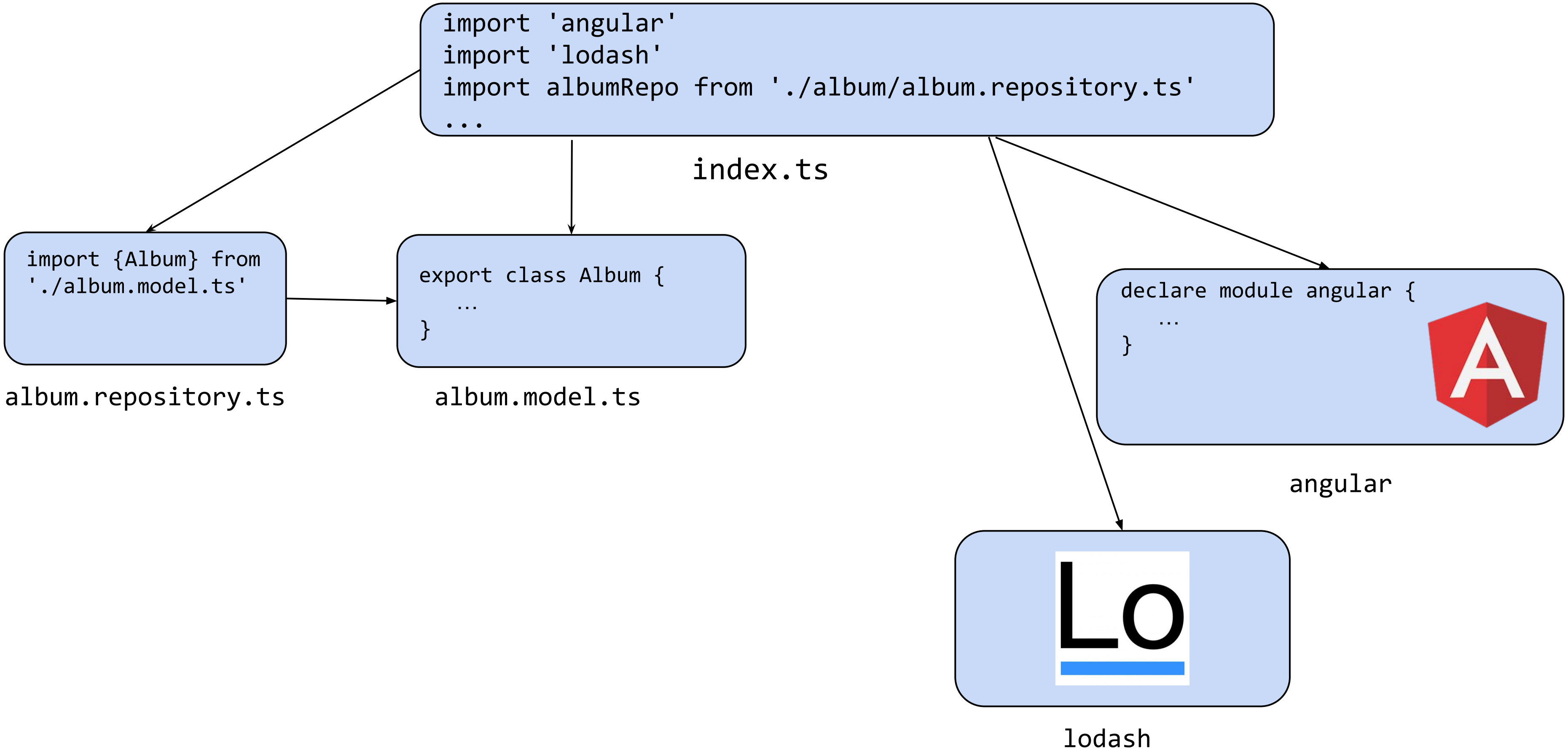
- constructor parameters with access modifiers become class members

```
class Artist {
     constructor(public id: number, private name: string) { }
}
```

- `this` keyword is always mandatory to access class members

```
class Artist {
  constructor(public id: number, private name: string) { }
  getName() {
    return this.name;
  }
}
```

Code organization



```
import 'angular'
import 'lodash'
import albumRepo from './album/album.repository.ts'
...
```

index.ts

```
import {Album} from
'./album.model.ts'
```

album.repository.ts

```
export class Album {
   …
}
```

album.model.ts

```
declare module angular {
   …
}
```

angular

lodash

General Remarks for exercises

- Slides before the exercise contain all necessary concepts and information
- Solution is always on the next slide: Try to do it on your own!
- Before doing the next exercise, we load the sample solution to avoid follow-up bugs
- Raise your hand, if you have questions or need help:
    - We will try to support you directly
    - We can address the question/problem for everyone if it is more generic

Exercise 1

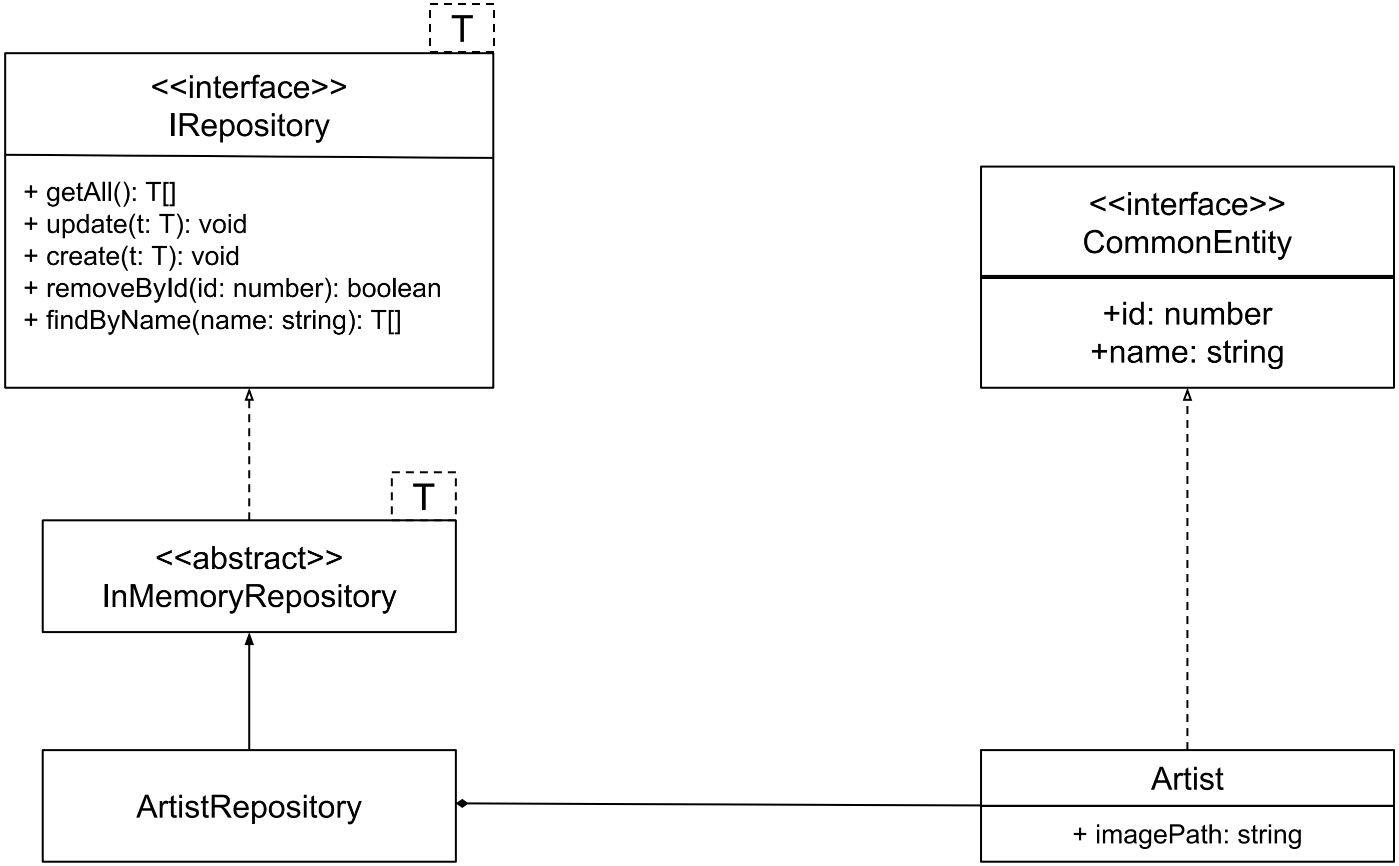
Initial project layout

- *src* folder :
  - contains the actual application with all subcomponents
  - like Java src folder
- *node_modules* folder (not visible):
  - materialization of all required modules
  - like target platform (as binary)
- *typings* folder:
  - typescript definitions 3rd party libs
  - like collection of Java Interfaces
- *dist* folder:
  - compiled and bundled application
  - *dist/index.html* file: application entry point
  - like Java bin folder
- *package.json* + *webpack.*.js* + *karma-conf.js*:
  - build and test configuration

Test Cases with Jasmine

- Tools:
  - Jasmine: JUnit for JavaScript
  - Karma: Test runner like JUnit Runner
- Programming Testcases with Jasmine
  - JUnit `assertEquals(a,b)` becomes `expect(a).toBe(b)`
- Execute Jasmine Testcases with Karma:
  - Open Terminal (Project Context Menu → Show In → Terminal)
  - Run: `npm run test`
  - If this fails, alternatively try: `./node_modules/.bin/karma start`

Entity and Repository



Repository interface

```
interface CommonEntity {
     id?: number;
     name: string;
}

interface IRepository<T extends CommonEntity > {
     getAll(): T[]
     update(t: T): void
     findById(id: number): T
     create(t: T): void
     removeById(id: number): boolean
     findByName(name: string): T[] // we need to implement this
}
```

Demo Application

- Exercise 1: Backend
  - Implement findByName function in Repository
  - Can be used to find artists by name

- Exercise 2 - Artist List
  - Show a list of all artists
  - Show each artist with name, image and album list

- Exercise 3 - Searchable Artist List
  - Add search box on top of artists-list where a search string can be entered
  - Change artist-list directive to show only artists containing the current search string

Checking input values, filtering arrays

- Checking input values:
  - given a string name
  - `if (name !== undefined && name !== "")`
- Filtering Arrays:
  - Use function on array: `Array.filter(element => boolean)`
  - Parameter of `filter` function is a function itself (like lambdas in Java 8)
    - it takes an element of the array as input
    - it returns a boolean
  - `filter()` applies parameter function to all elements in the array
  - `filter()` returns only array elements where parameter function returned true
  - Example: `[1,2,3].filter(element => (element % 2)!=0)` returns `[1,3]`

Finding substrings

- Finding Substrings:
  - Use function on string: `String.indexOf(subString: string)`
  - Parameter of `indexOf` function is a string
  - If parameter string is found as substring in the string the start index is returned
  - If it is not, `indexOf` returns -1
  - Example: `"EclipseCon".indexOf("Con")` returns `7`

Your turn: Exercise 1 - Typescript

- Execute Jasmine Testcases with Karma:
  - Open Console
  - Run: `npm run test`
  - Alternatively, run `./node_modules/.bin/karma start`
- Find test case in src/`artist/artist.repository.spec.ts` with `//FIXME`
- Implement method `findByName(string): T[]` in class `InMemoryRepository` (`src/common/in-memory-repository.ts`) to fix test case
  - read JSDoc
  - use `filter()` method on the entities array
  - use `.indexOf(subString)` method on entity name to implement filter lambda
  - Important corner case: passing in `""` or `undefined` shall return all entities
- Don't forget about `this` when referencing class attributes
- Careful: Solution is on next slide :)

Exercise 1 - Solution

```
findByName(name: string): ENTITY[] {
    if (name === undefined || name === "") {
        return this.getAll();
    } else {
        return this.getAll().filter(element => element.name.indexOf(name) > -1);
    }
}
```

ANGULARJS
presented by EclipseSource

What is it and why do we need it?

We want to build Single Page Applications (SPA), because

- no page loading
- fluid and responsive user experience

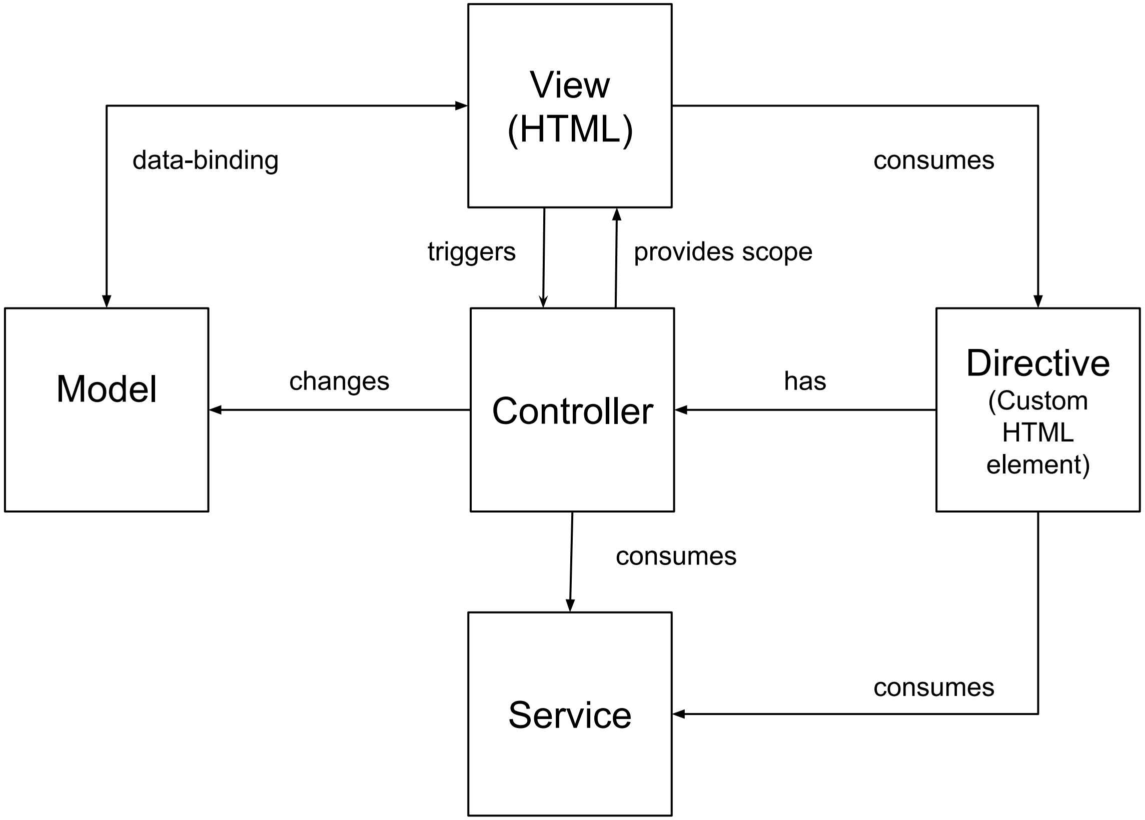
→ hence, we need dynamic views, but HTML is static and has a fixed vocabulary

AngularJS is built for this scenario and allows each application to:

- manipulate the HTML DOM (“AST for HTML”) without page reload
- extend the HTML vocabulary with custom HTML elements (called directives)

Important Angular Concepts

- Controllers
- DataBinding
- Services
- Directives



Controllers

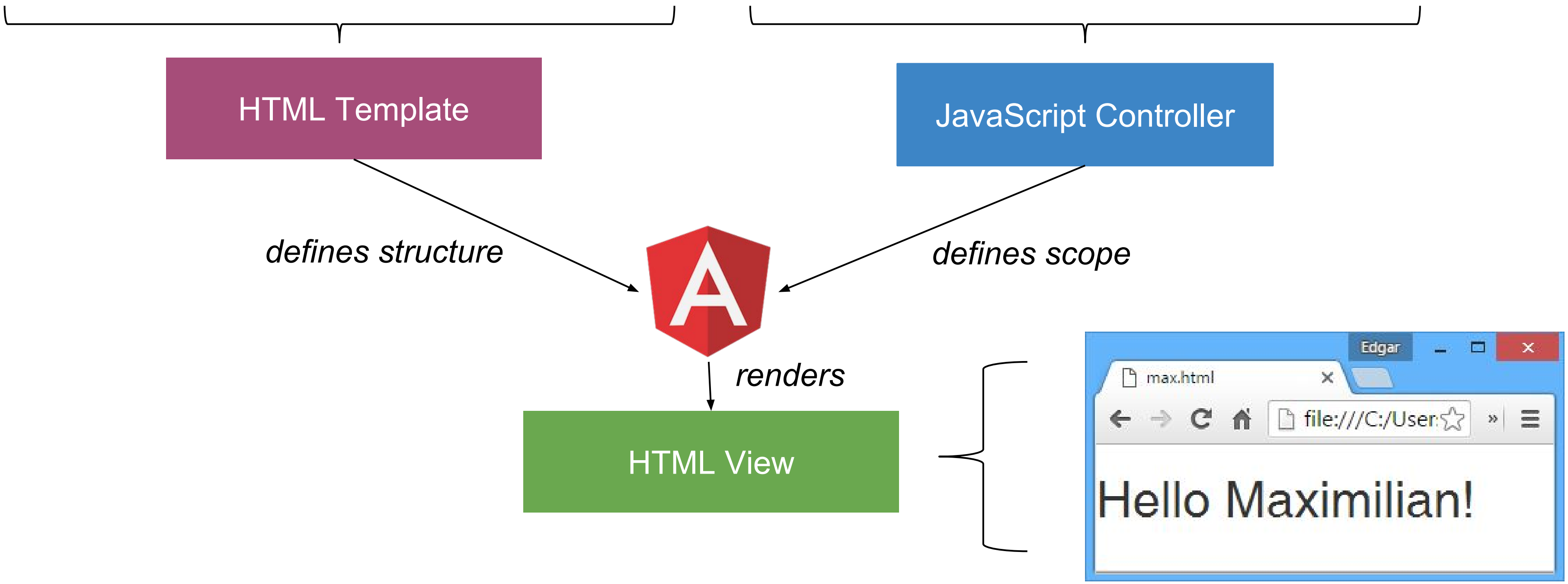
Controllers define the application behaviour, by:

- acting as glue between HTML and JavaScript
- providing properties or methods which can be accessed in the HTML Template (a.k.a. “scope”)
- adding behaviour to your app (business logic)

Controllers in action

```
<div ng-controller="SomeCtrl as ctrl">
  <h1>Hello {{ctrl.yourName}}!</h1>
</div>
```

```
class SomeCtrl {
  public yourName: string = "Maximilian";
}
```



Hello Maximilian!

→ **Angular acts as a template engine**

What controllers should not be used for

Do **not** use controllers for:

- Sharing code or state across controllers
  → Use Services (upcoming)

- Manipulating the DOM
  → Use Directives or Databinding (both upcoming)

- Filtering and formatting value of Angular expressions (typically within template)
  → Filters (not covered here)

No controllers in Angular 2

- Angular 2 uses components which essentially are a combination of
  - a regular TypeScript class
  - TypeScript class properties become the scope/viewmodel
  - a HTML template
  - a selector

→ hence we try to avoid controllers in this tutorial

→ we use Angular 1 Directives

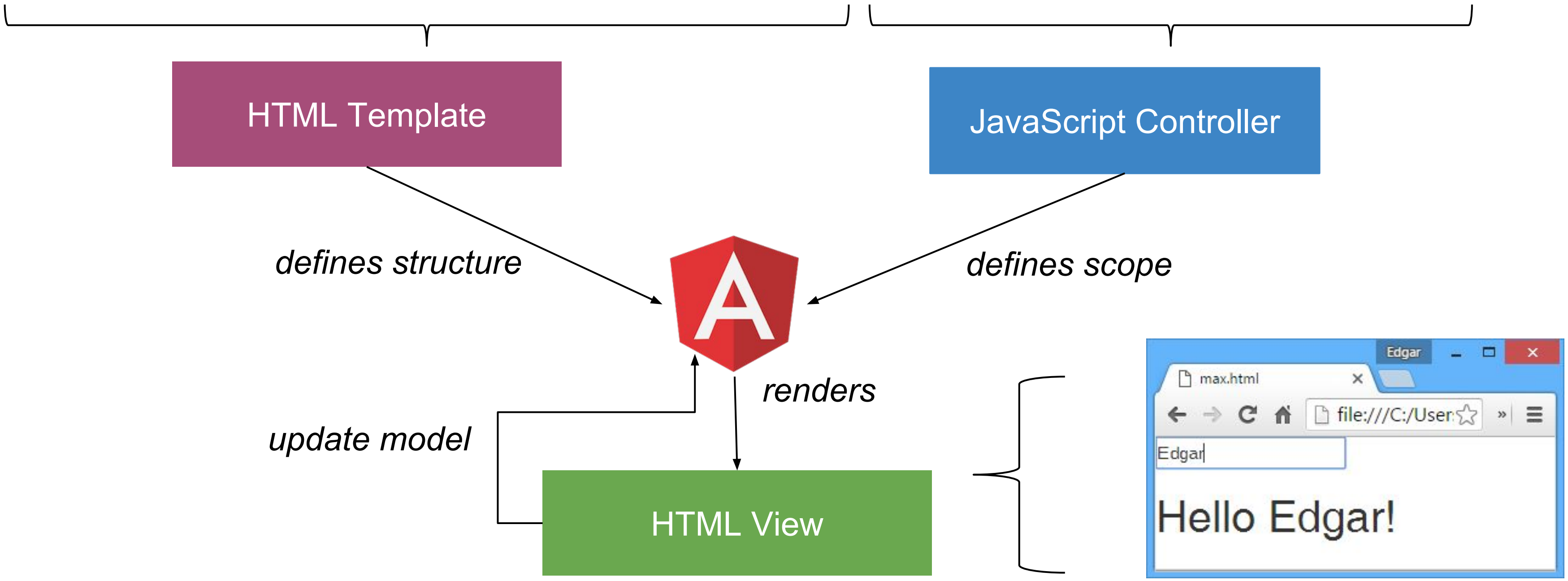
→ eases migration to Angular 2 Components

As of Angular 1.5, components (alongside other A2 features like life cycle hooks) are also available in Angular 1, but they basically are just syntactic sugar for the concepts we are going to cover

Databinding

```
<div ng-controller="SomeCtrl as ctrl">
  <input type="text" ng-model="ctrl.yourName">
  <h1>Hello {{ctrl.yourName}}!</h1>
</div>
```

```
class SomeCtrl {
  public yourName: string = "Maximilian";
}
```



Edgar

Hello Edgar!

Services

- Singletons
- Lazy Initialization
- Primarily used for sharing data and functions across different controllers/directives

```
angular.module('econTutorial')
        .service('ArtistRepository', ArtistRepository);
```

Service Identifier, like
OSGi Service Interface

Class to be registered, like
OSGi service implementation

Consuming Services

Service Interface needs
to be explicitly stated

```
angular.module('econTutorial')
        .controller('ArtistListDirectiveController',
            ['ArtistRepository', (artistRepo: ArtistRepository) =>
                new ArtistListDirectiveController(artistRepo)])
```

Service Identifier

Inject instance into
controller constructor

Directives

- Allow you to create custom HTML elements (and attributes)
- Add application specific semantics to your html
- Improve modularity
- Angular ships with some built-in directives:
    - start with `ng-`prefix
    - `e.g. ng-repeat`

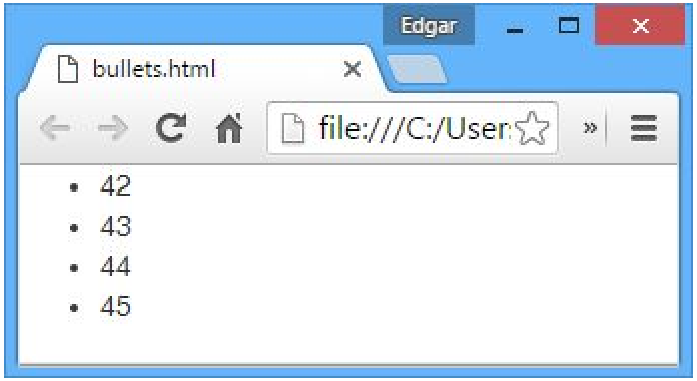
ng-repeat Example

HTML Template:

```
<ul>
  <li ng-repeat="n in [42, 43, 44, 45]">
    {{n}}
  </li>
</ul>
```

Resulting HTML:

```
<ul>
  <li>42</li>
  <li>43</li>
  <li>44</li>
  <li>45</li>
</ul>
```



- 42
- 43
- 44
- 45

Exercise 2

Defining custom Directives

```
class MyDirective implements ng.IDirective {

  restrict: string = 'E';

  controller: string = 'MyDirectiveController';

  controllerAs: string = 'vm';

  template: string = '<h1>{{vm.text}}</h1>';
}
```

Typescript class implementing IDirective

Directive type, 'E' for Element

Name of controller

Controller alias in template

defines the HTML template that the directives expands to

```
class MyDirectiveController {

  public text: string = 'some text';
}
```

Controller “POJO”

Property which is accessible in template

Registering and using custom Directives

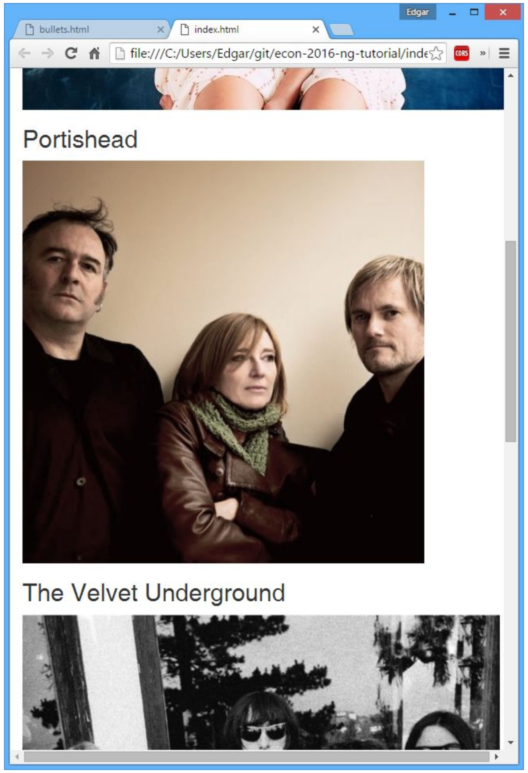
```
angular.module('econTutorial')
       .directive('myElement', () => new MyDirective());

<div>
    <my-element>
    </my-element>
</div>
```

→ Camel-case in code → dashed name in HTML (`my-element`)

Exercise 2 Preparation

- Import project *econ-2016-ng-tutorial-exercise-2* into workspace from: `econ-2016-ng-tutorial-exercise-2.zip`
  - *Do **not** select File → Import → Archive…*
  - ***but** select File → Import → General → Existing Projects… → Select archive file → Finish*
  - On Linux go into the project folder and run: `chmod -R +x ./node_modules/`

Demo Application



Portishead

The Velvet Underground

- Exercise 1: Backend
    - Implement a findByName function in Repository
    - Can be used to find artists by name

- Exercise 2 - Artist List
    - Show a list of all artists with their name and picture
    - Optional: Also show their album list

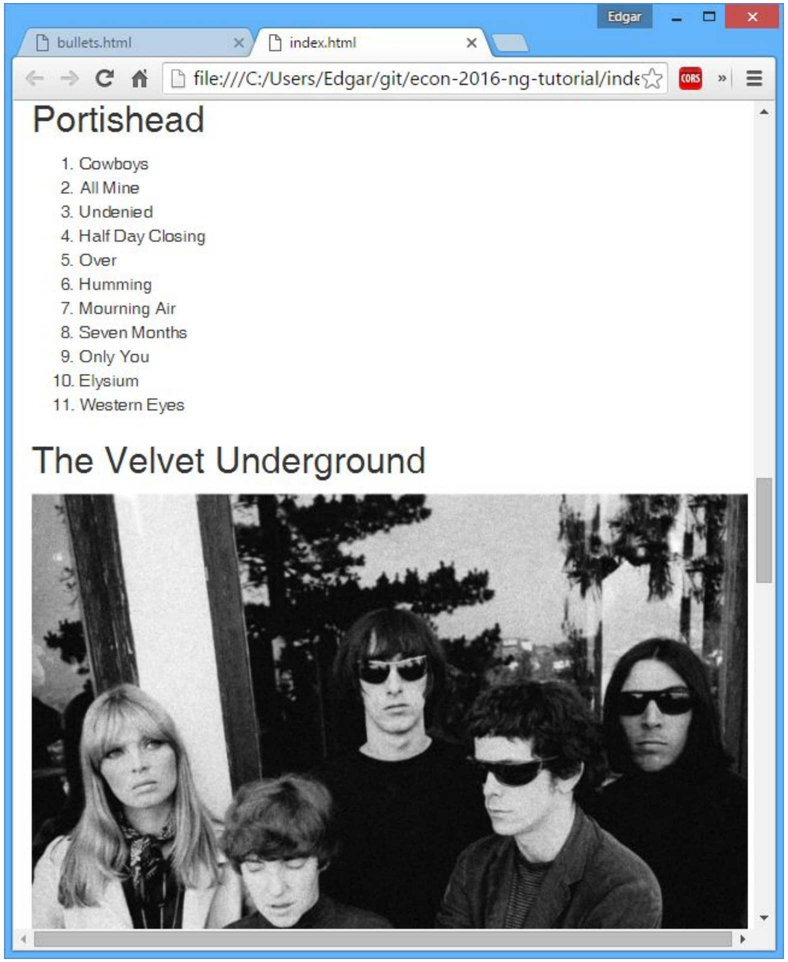
- Exercise 3 - Searchable Artist List
    - Add search box on top of artists-list where a search string can be entered
    - Change artist-list directive to show only artists containing the current search string

Your turn: Exercise 2

- Implement the `template` property of the `ArtistListDirective` `(src/artist/artist-list.directive.ts):`
  - use `ng-repeat` with the `getArtists()` method from the controller
  - Display an artist's name
  - Display an artist's image using `<img ng-src="{{???}}"/>`
  - Remember to use angular expressions: `{{vm.<property/method>}}`
- Build via webpack:
  - Open console (Project Context Menu → Show In → Terminal)
  - Run: `npm run build`
  - Alternatively, run: `node_modules/.bin/webpack --config webpack.config.js`
- Open dist/`index.html` with your Browser to view the result
- Optionally, if you are really good: Take Exercise 2.5 on next slide

Your turn: Optional Exercise 2.5

- We also provide a second directive `<album-list artist-id="<ID>">`
  - It receives an artist id as parameter
  - It renders the albums of that specific artist provided as parameter
- Try to also include this directive in your template property of the `ArtistListDirective`



Portishead

1. Cowboys
2. All Mine
3. Undenied
4. Half Day Closing
5. Over
6. Humming
7. Mourning Air
8. Seven Months
9. Only You
10. Elysium
11. Western Eyes

The Velvet Underground

Solution

Directive:

```
class ArtistListDirective implements ng.IDirective {

    restrict = 'E';

    controller = 'ArtistListDirectiveController';

    controllerAs = 'vm';

    template = `<div ng-repeat="artist in vm.getArtists()">
      <h1>{{artist.name}}</h1>
      <img ng-src="{{artist.imagePath}}"/>
      <album-list artist-id="{{artist.id}}"></album-list>
    </div>`;

}
```

Usage in `index.html`

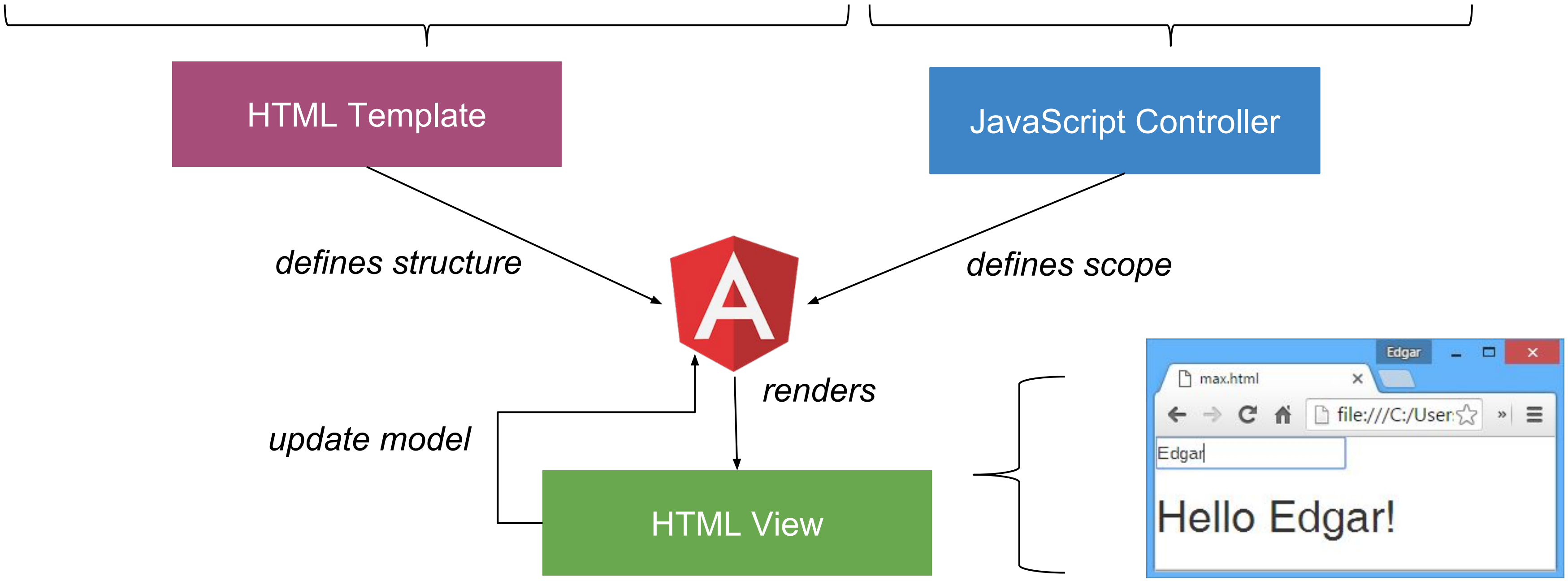
```
<body ng-app='econTutorial'>
  <artist-list></artist-list>
</body>
```

Exercise 3

Recap: Databinding

```
<div ng-controller="SomeCtrl as ctrl">
  <input type="text" ng-model="ctrl.yourName">
  <h1>Hello {{ctrl.yourName}}!</h1>
</div>
```

```
class SomeCtrl {
  public yourName: string = "Maximilian";
}
```



Edgar

Hello Edgar!

Adding attributes to directives

Pass in value by value

```
<some-element name='Max'></some-element>
```

```
class SomeDirective implements ng.IDirective {
  restrict = 'E';
  controller = 'SomeController';
  controllerAs = 'ctrl';
  bindToController = {
    fullName: '@name'
  };
  template: '<div>{{ctrl.fullName}}</div>'
}

class SomeController {
  public fullName: string;
}
```

Binds value of name attribute from directive to controller property `fullName`:

`@`: String-Binding (call-by-value)
=: Two-way-binding (call-by-reference)

```
 angular.module('myApp')
        .directive('someElement', () => new SomeDirective());
```

Exercise 3 Preparation

- Import project *econ-2016-ng-tutorial-exercise-3* into workspace from: `econ-2016-ng-tutorial-exercise-3.zip`
  - *Do **not** select File → Import → Archive…*
  - ***but** select File → Import → General → Existing Projects… → Select archive file → Finish*
  - On Linux go into the project folder and run: `chmod -R +x ./node_modules/`

Demo Application

- Exercise 1: Backend
  - Implement a findByName function in Repository
  - Can be used to find artists by name
- Exercise 2 - Artist List
  - Show a list of all artists
  - Show each artist with name, image and album list
- Exercise 3 - Searchable Artist List
  - Add search box on top of artists-list where a search string can be entered
  - Change artist-list directive to show only artists containing the current search string

Exercise 3

- Update your `index.html`:
  - Add input text box
  - Bind textbox against `SearchController.searchString` property with `ng-model`

    ```
    <input type="text" ng-model="???"/>
    ```
  - Update artist-list element to bind to same property

    ```
    <artist-list filter-by-name="searchCtrl.searchString"></artist-list>
    ```
- Add a `bindToController` property to `ArtistListDirective:`
  - Declare a parameter `filterByName` to be passed in via '=' (Two-way-binding)
  - Bind it to a property called `searchString`
- Adapt `ArtistListDirectiveController.getArtists()` to filter the returned artists to the artists matching the value of the bound `searchString` property
  - Create a string property `searchString in ArtistListDirectiveController`
  - Use method `findByName(this.searchString)` on artist repo to filter

Solution Exercise 3 - Template

```
<body ng-app="econTutorial">
<div ng-controller="SearchController as searchCtrl">
    <h1>Artists</h1>
    <input type="text" ng-model="searchCtrl.searchString">
    <artist-list filter-by-name="searchCtrl.searchString"></artist-list>
</div>
</body>
```

Solution Exercise 3 - Directive

```
class ArtistListDirective implements ng.IDirective {
    restrict = 'E';
    controller = 'ArtistListDirectiveController';
    controllerAs = 'vm';
    bindToController = {
      searchString: '=filterByName'
    };
    template = `<div ng-repeat="artist in vm.getArtists()">
      <h1>{{artist.name}}</h1>
      <img ng-src="{{artist.imagePath}}"/>
      <album-list artist-id="{{artist.id}}"></album-list>
    </div>`;
}
```

Solution Exercise 3 - Controller

```
class ArtistListDirectiveController {

    searchString: string;

    constructor(private repo: ArtistRepository) { }

    getArtists() {
     return this.repo.findByName(this.searchString);
    }
}
```

Angular 1.5

Before Angular 1.5

```
app.directive('list', () => {
     return {
       bindToController: {
         items: '='
       },
       templateUrl: 'list.html',
       controller: function ListCtrl() {},
       controllerAs: 'ctrl'
     }
   });
```

With Angular 1.5

```
app.component('list', {
  bindings: {
    items: '='
  },
  templateUrl: 'list.html',
  controller: function ListCtrl() {},
  controllerAs: 'ctrl',
});
```

→ `bindings` defines input and outputs

→ simpler, less boilerplate

→ see docs for more info

Resources

Angular

- https://angularjs.org/
- https://angular.io/
- http://angular.codeschool.com/
- https://thinkster.io/a-better-way-to-learn-angularjs
- https://egghead.io/technologies/angularjs
- http://blog.thoughtram.io/categories/angular/

Tooling

- https://marketplace.eclipse.org/content/angularjs-eclipse
- https://marketplace.eclipse.org/content/tern-eclipse-ide
- https://github.com/palantir/eclipse-typescript
- http://typecsdev.com/
- https://github.com/angelozerr/typescript.java
- https://github.com/angelozerr/jsbuild-eclipse

Thank you!

Evaluate the Sessions

Sign in and vote at **eclipsecon.org**

- 1 0 + 1

eclipsecon Europe 2016

OSGi™ Community Event 2016

Evaluate ~~the~~ this Session~~s~~

Sign in and vote at eclipsecon.org

WITH

- ~~1~~ ~~0~~ +1