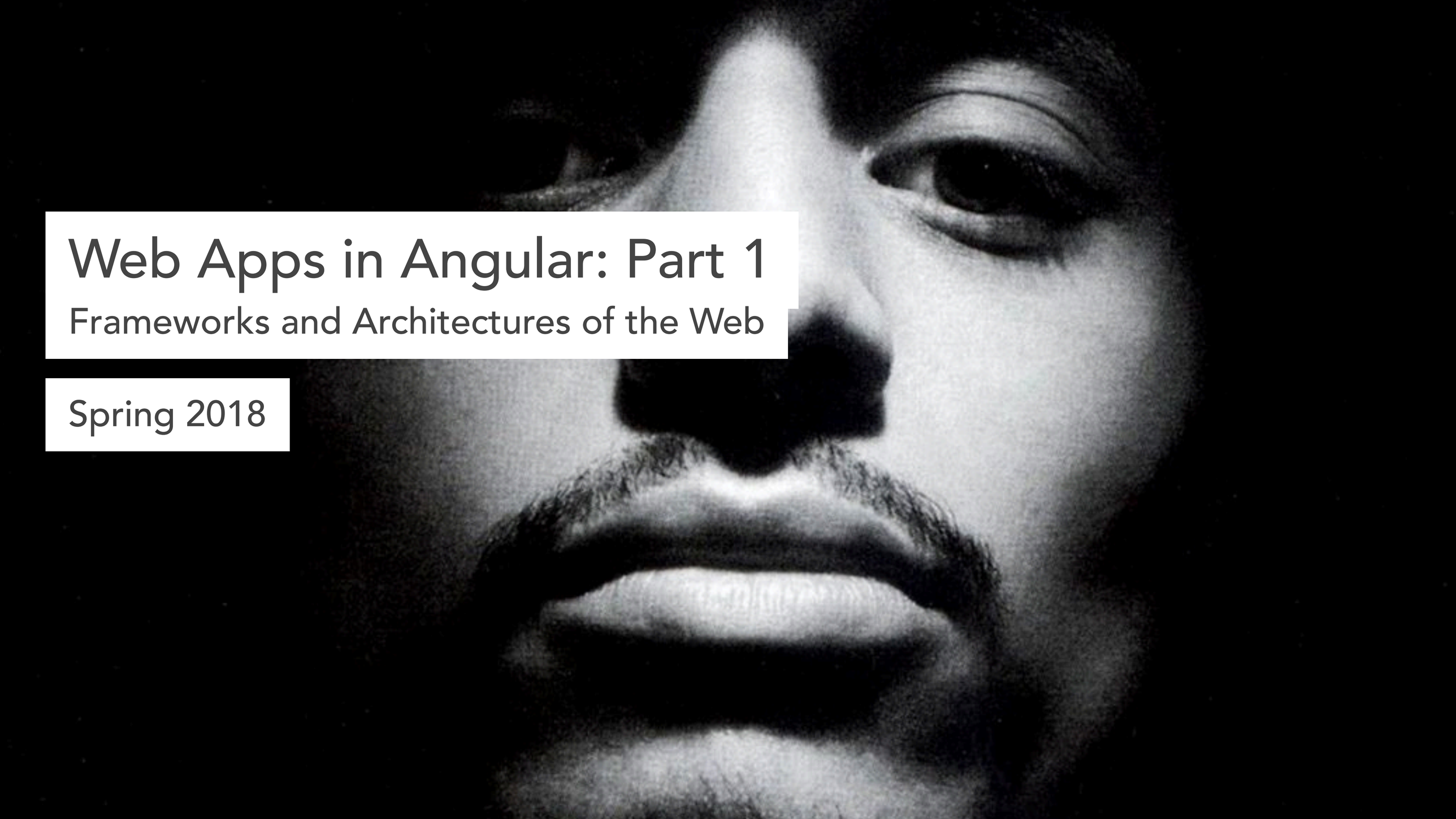


Web Apps in Angular: Part 1

Frameworks and Architectures of the Web

Spring 2018

Today’s Program

Introduction to Angular: Core Principles

Setting up an Angular Project

The Enspire App Walkthrough: Templates, Bindings and Directives

Break

Consultation and TA Assistance

The Dreamer
José James
The Dreamer

Course Outline

Frameworks

Week 13	Week 14	Week 15	Week 16	Week 17	Week 18	Week 19	Week 20
No Lecture (Easter Holiday)	Web Apps in React (Part 1)	Web Apps in React (Part 2)	TypeScript and Object-Oriented Programming	Web Apps in Angular (Part 1)	Web Apps in Angular (Part 2)	Augmented Reality in the Browser	No Lecture (End of Semester)

Creative Implementation of a Web App

Web Project - Implementation

Image source: https://cdn.worldvectorlogo.com/logos/angular-icon-1.svg

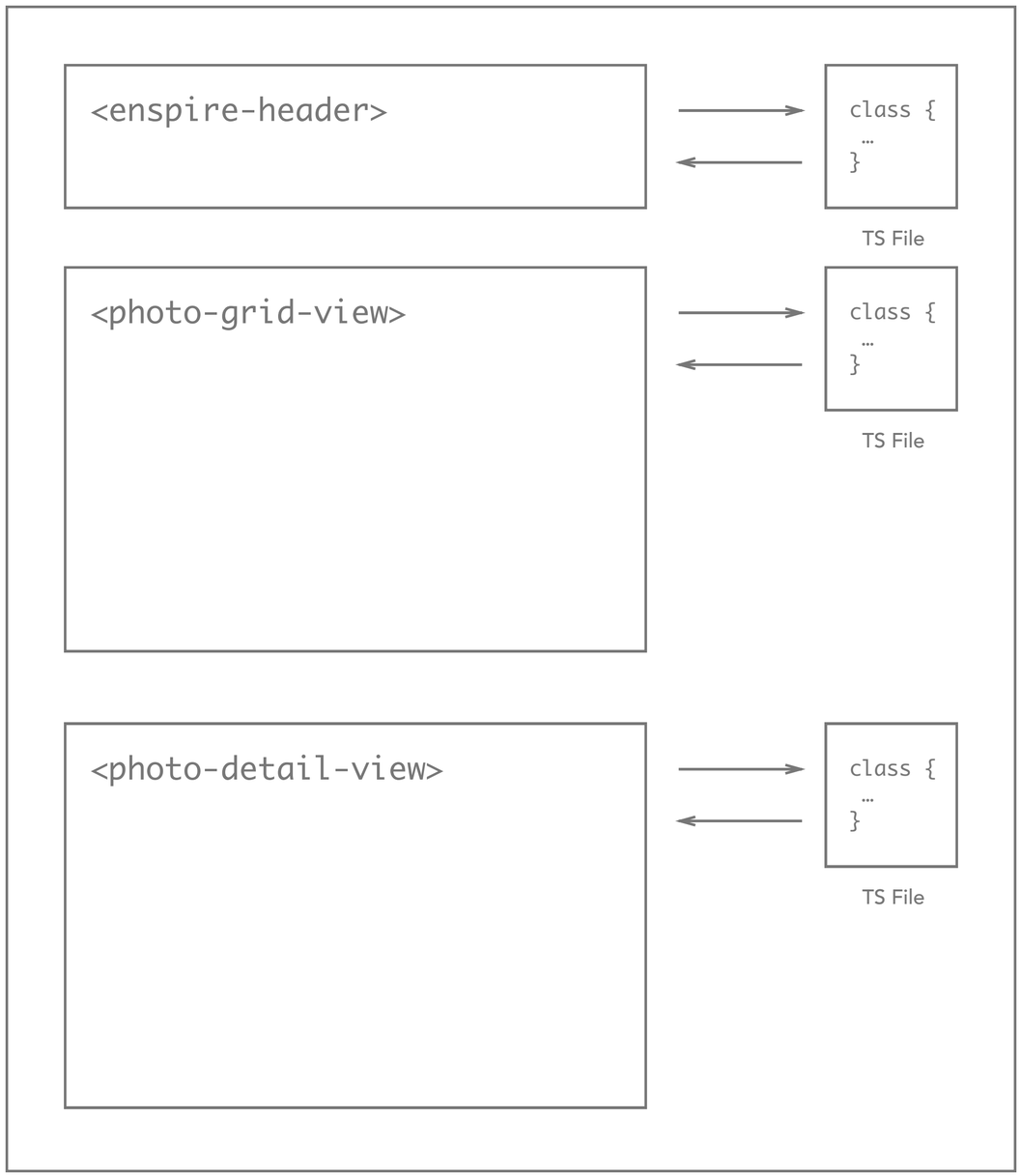
What is Angular?

Angular is a front-end Web application framework.

It provides a set of tools and workflows for developing complex Web applications.

Angular apps consist of components. A component could be a section of an app, such as a header or a list view, or a custom user interface component, such as an image gallery.

Components are intended to be independent, testable and reusable. This allows you to develop complex applications **with scale**.



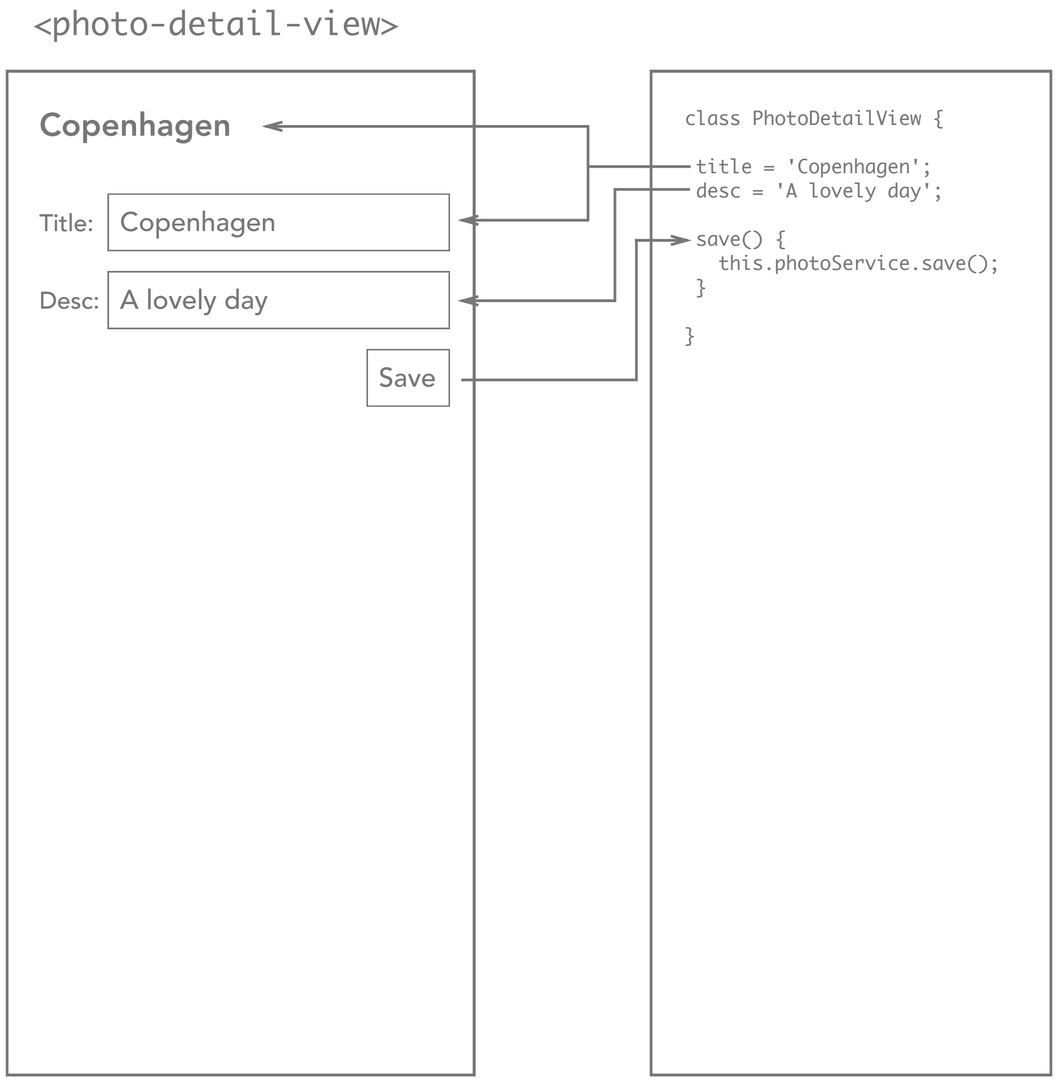
Components

A component could be best thought of as a ‘piece’ of an application.

Each component has a template, which is a view, and a TypeScript file, which is a controller, and its own independent stylesheet.

In plain JavaScript, you need to bind events to elements and update the DOM manually. In Angular, the component displays and synchronises the data contained within its controller.

In Angular, components are commonly nested, and communicate via events.

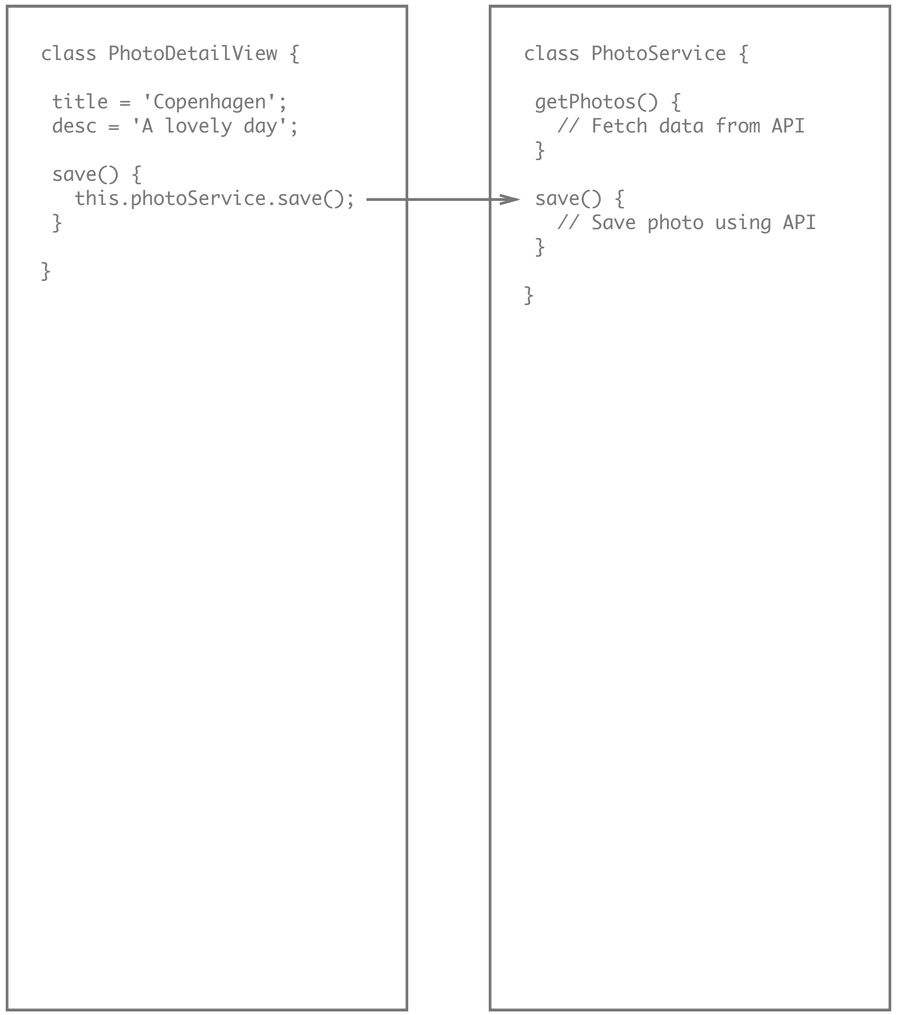


Data Binding

Templates provide the ‘view’ for the component. The data contained within this view is synchronised with its controller.

Within the template, you specify how it displays the data contained within its controller. You can interpolate data to display it, and you can bind it to DOM properties. Angular updates the changes between its view and controller automatically.

You can also bind events on your elements so your users can interact with your app.



Services

Components use services to perform common pieces of functionality that are not related to the view of the component.

Services are commonly used, for example, to handle data storage, or as an interface to an external API.

Services provide an abstraction for components to perform common and shared tasks within the application.

Dependency Injection

Angular applications are highly modular. For each component, you'll need to explicitly declare its model classes and its services.

The Angular framework itself is highly modular. If you need to use any piece of functionality that's not part of the "core" framework, you'll need to explicitly declare it and inject it into your components and services.

```
npm install -g @angular/cli

ng new enspire-app

cd enspire-app

ng serve
```

Set up an Angular Project

Download NodeJS for your platform, if you haven't done so already: https://nodejs.org/en/download/

In your terminal, run the commands on the left. These commands:

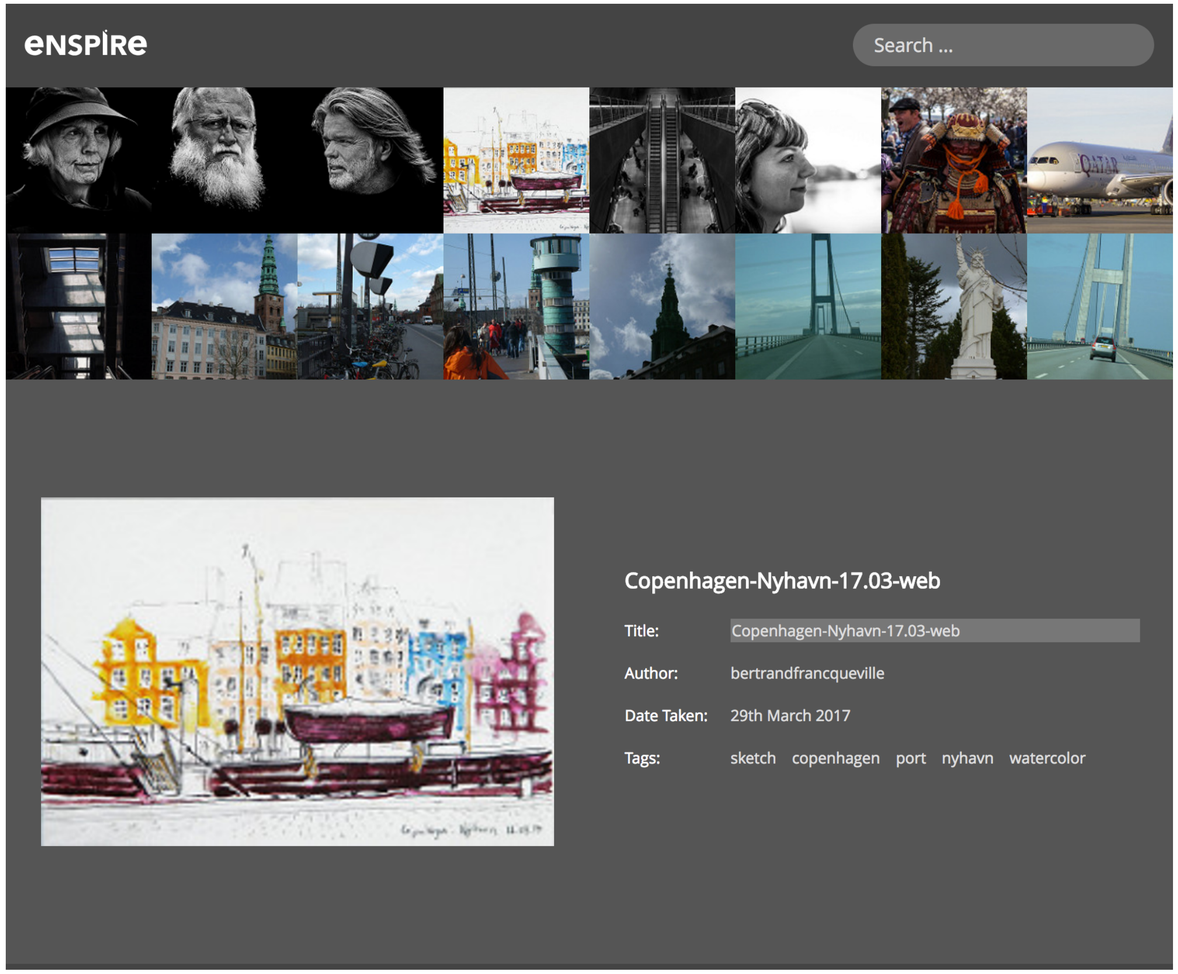
- Install Angular CLI.
- Create a new Angular project called enspire-app.
- Switch the current directory to your app directory.
- Compile your app, start a server, and watch for changes.

Your Build Tools

`ng serve` creates a server for you to run your app on. View your app on `localhost:4200` (or whatever it says in your console.).

These build tools watch your changes in your code, recompile your code using Webpack, and automatically display the changes in your browser.

It's usually best to have the following windows open:
- Your IDE (such as Atom).
- A web browser displaying your application.
- Your terminal, to watch for compilation errors.



The Enspire App

Today, we'll be building a simple photo browser app in Angular. This will be based on an existing static HTML / CSS template that we have created for you.

Download the 'Enspire App Static Template' from the LearnIT site. Open these files as a seperate project in your IDE.

Welcome to app!

Here are some links to help you start:

- Tour of Heroes
- CLI Documentation
- Angular blog

Your Angular App

If everything is all set-up correctly, you should see this welcome screen.

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Copenhagen';
}
```

The App Component

The app component is the “root” component of your application. Today, we’ll be working with this component.

Open `app.component.ts` within your `app/src` folder. Change the title string to ‘Copenhagen’. What happens?

app.component.html

```
<h1>
  {{title}}
</h1>
<p>
  {{description}}
</p>
```

The Template

When you hit “save” in your editor, Angular automatically compiles your changes and updates your browser.

Now, navigate to `app.component.html`. Replace the code that you see there with the code on the left.

Our app will display a title and a description that are bound to the properties of our App Component.

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Copenhagen';
  description = 'A very nice city.';
}
```

Add Another Property

Then, back within the app component's TypeScript file, we'll add the description property.

The template is used to display the public properties and fields within the app's controller.

Let's expand on this example and display information about a single photograph.

app.component.html

```
<section class="image-view-detail">
  <!-- … -->
  <div class="image-view-detail-image-container">
    <figure class="image-view-detail-image" style="background-
image: url('http://farm5.staticflickr.com/
4177/34380247115_efff2849ee_m.jpg)')">
      <img src="http://farm5.staticflickr.com/
4177/34380247115_efff2849ee_m.jpg" alt="Kormoran im
Hafenbecken" />
    </figure>
  </div>
  <div class="image-view-detail-meta">
    <h2>Copenhagen-Nyhavn-17.03-web</h2>
    <div class="meta-group">
      <label class="meta-label" for="">Title:</label>
      <span class="meta-data">Copenhagen-Nyhavn-17.03-web</span>
      <!-- … -->
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Author:</label>
      <span class="meta-data">bertrandfrancqueville</span>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Date Taken:</label>
      <span class="meta-data">29th March 2017</span>
    </div>
    …
</section>
```

Update the Template

Copy this portion of code from the `index.html` file of the Enspire App Static Template and paste it into `app.component.html`.

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Copenhagen';
  author = 'John Smith';
  dateTaken = new Date('1 April 2017');
}
```

Update the Properties

In our component's TypeScript file, we'll update our properties so that our photo displays a title, author and the date that it was taken.

app.component.html

```
<section class="image-view-detail">
  <!-- … -->
  <div class="image-view-detail-image-container">
    <figure class="image-view-detail-image" style="background-
image: url('http://farm5.staticflickr.com/
4177/34380247115_efff2849ee_m.jpg)')">
      <img src="http://farm5.staticflickr.com/
4177/34380247115_efff2849ee_m.jpg" alt="Kormoran im
Hafenbecken" />
    </figure>
  </div>
  <div class="image-view-detail-meta">
    <h2>{{title}}</h2>
    <div class="meta-group">
      <label class="meta-label" for="">Title:</label>
      <span class="meta-data">{{title}}</span>
      <!-- … -->
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Author:</label>
      <span class="meta-data">{{author}}</span>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Date Taken:</label>
      <span class="meta-data">{{dateTaken}}</span>
    </div>
    …
</section>
```

Bind Using Interpolation

Then, we'll bind the values of these properties to the template using curly braces ({{}}).

This is one way of binding data to a template, and it's called **interpolation**.

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Copenhagen';
  author = 'John Smith';
  dateTaken = new Date('1 April 2017');
  imageURL = 'http://farm5.staticflickr.com/
4177/34380247115_efff2849ee_m.jpg';
  URL = 'http://www.flickr.com/photos/itzick/33571103773/';
}
```

Add Another Two Properties

Next, we'll add another two properties that display both the image URL of the photo, and its source URL as it appears on Flickr.

app.component.html

```
<section class="image-view-detail">
  <!-- … -->
  <div class="image-view-detail-image-container">
    <figure class="image-view-detail-image"
            [style.backgroundImage]="'url(' + imageURL + ')'">
      <img [src]="imageURL" [alt]="title" />
    </figure>
  </div>
  <div class="image-view-detail-meta">
    <h2>{{title}}</h2>
    <div class="meta-group">
      <label class="meta-label" for="">Title:</label>
      <span class="meta-data">{{title}}</span>
      <!-- … -->
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Author:</label>
      <span class="meta-data">{{author}}</span>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Date Taken:</label>
      <span class="meta-data">{{dateTaken}}</span>
    </div>
    …
</section>
```

Bind to Properties: One-Way

You can also bind to the properties of a DOM element using the [] syntax.

Within the [], you specify the name of the property you would wish to bind to, and then as part of the attribute's value, you can specify either the name of the property from your controller, or an expression.

These bindings are **one-way**: the controller updates these bindings if any of its properties change. Conversely, these properties only "receive" the values that are provided by the controller.

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { NgModule } from '@angular/core';


import { AppComponent } from './app.component';


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

Adding the FormsModule

Angular is highly modular, meaning that when you build apps with Angular, you explicitly indicate only the functionality that you require.

For the next part of the walkthrough, we'll be dealing with Angular's form handling capabilities. To do this, we'll need to add the `FormsModule` to our `app.module.ts` file.

app.component.html

```
<section class="image-view-detail">
  <!-- … -->
  <div class="image-view-detail-image-container">
    <figure class="image-view-detail-image"
            [style.backgroundImage]="'url(' + imageURL + ')'">
      <img [src]="imageURL" [alt]="title" />
    </figure>
  </div>
  <div class="image-view-detail-meta">
    <h2>{{title}}</h2>
    <div class="meta-group">
      <label class="meta-label" for="">Title:</label>
      <span class="meta-data">{{title}}</span>
      <div class="meta-data">
        <input type="text" name="title" [(ngModel)]="title" />
      </div>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Author:</label>
      <span class="meta-data">{{author}}</span>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Date Taken:</label>
      <span class="meta-data">{{dateTaken}}</span>
    </div>
    …
</section>
```

ngModel: Two-Way Binding

You can also bind using `[()]`. `[()]` means two-way binding — data can travel in both directions.

In this example, we've added an input field. The input field has a special attribute called `[(ngModel)]`. `[(ngModel)]` indicates that it will be bound to the `title` property of the controller: the value of the input field will be set to the value of the `title` property.

Likewise, when the input field changes, it will **update** the value within the controller, and update all instances of that property within the template. Try it yourself!

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  title = 'Copenhagen';
  author = 'John Smith';
  dateTaken = new Date('1 April 2017');
  imageURL = 'http://farm5.staticflickr.com/
4177/34380247115_efff2849ee_m.jpg';
  URL = 'http://www.flickr.com/photos/itzick/33571103773/';

  isEditing: boolean = false;
}
```

Adding Interactivity

Let's add some interactivity to our application. We want to make it so that a user can edit it's title by clicking on it.

Within our component's TypeScript file, we'll add an additional field called `isEditing` and we'll set it to false.

app.component.html

```
<section class="image-view-detail">
  <!-- … -->
  <div class="image-view-detail-image-container">
    <figure class="image-view-detail-image"
            [style.backgroundImage]="'url(' + imageURL + ')'">
      <img [src]="imageURL" [alt]="title" />
    </figure>
  </div>
  <div class="image-view-detail-meta">
    <h2>{{title}}</h2>
    <div class="meta-group">
      <label class="meta-label" for="">Title:</label>
      <span *ngIf="!isEditing" class="meta-data">
        {{title}}
      </span>
      <div *ngIf="isEditing" class="meta-data">
        <input type="text" name="title" [(ngModel)]="title" />
      </div>
      <div class="meta-data">
        <small>Click to edit</small>
      </div>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Author:</label>
      <span class="meta-data">{{author}}</span>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Date Taken:</label>
      <span class="meta-data">{{dateTaken}}</span>
    </div>
    …
</section>
```

ngIf

Next, we'll use a special attribute called *ngIf to toggle the display of certain elements based on the value of an expression. You can think of *ngIf as like an 'if statement' in your HTML.

In this example, the `span` of the title field is displayed if `isEditing` is set to `false`, and the `input` field is displayed if `isEditing` is set to `true`.

`*ngIf` is what's known as a **structural directive** as it used to alter the structure of your HTML template

app.component.html

```
<section class="image-view-detail">
  <!-- … -->
  <div class="image-view-detail-image-container">
    <figure class="image-view-detail-image"
            [style.backgroundImage]="'url(' + imageURL + ')'">
      <img [src]="imageURL" [alt]="title" />
    </figure>
  </div>
  <div class="image-view-detail-meta">
    <h2>{{title}}</h2>
    <div class="meta-group">
      <label class="meta-label" for="">Title:</label>
      <span *ngIf="!isEditing" class="meta-
data" (click)="isEditing = true">
        {{title}}
      </span>
      <div *ngIf="isEditing" class="meta-data">
        <input type="text"
name="title" [(ngModel)]="title" (blur)="isEditing = false" />
      </div>
      <div class="meta-data">
        <small>Click to edit</small>
      </div>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Author:</label>
      <span class="meta-data">{{author}}</span>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Date Taken:</label>
      <span class="meta-data">{{dateTaken}}</span>
    </div>
    …
</section>
```

One Way Binding from Events

Next, we can bind events on our DOM elements to update the properties of our component. We do this by using the `()` syntax.

In this example, when the span element is clicked on, it sets `isEditing` to `true`. Likewise, when the input text field loses its focus (i.e, it “blurs”), `isEditing` is set to `false`.

As the `isEditing` field gets updated via these event bindings, our application automatically toggles the display of the `span` and `input` element because of `ngIf`. Event-bindings are one way: they activate in response to events and ‘update’ the properties of the component.

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  title = 'Copenhagen';
  author = 'John Smith';
  dateTaken = new Date('1 April 2017');
  imageURL = 'http://farm5.staticflickr.com/
4177/34380247115_efff2849ee_m.jpg';
  URL = 'http://www.flickr.com/photos/itzick/33571103773/';
  tags = ['sketch', 'copenhagen', 'port', 'nyhavn'];

  isEditing: boolean = false;
}
```

Adding Complex Data

Let’s add some complex data to our application. In this example, we’ll display an array of tags that are associated with each photo. We’ll add this array to our component.

app.component.html

```
<section class="image-view-detail">
  <!-- … -->
  <div class="image-view-detail-image-container">
    <figure class="image-view-detail-image"
            [style.backgroundImage]="'url(' + imageURL + ')'">
      <img [src]="imageURL" [alt]="title" />
    </figure>
  </div>
  <div class="image-view-detail-meta">
    <h2>{{title}}</h2>
    …
    <div class="meta-group">
      <label class="meta-label" for="">Author:</label>
      <span class="meta-data">{{author}}</span>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Date Taken:</label>
      <span class="meta-data">{{dateTaken}}</span>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Tags:</label>
      <span class="meta-data">
        <span *ngFor="let tag of tags" class="tag">
          <a href>{{tag}}</a>
        </span>
      </span>
    </div>
  </div>
</section>
```

ngFor

Then, we'll use another structural directive called ngFor to actually display the tags. You would use *ngFor if you needed to display elements in repitition, such as lists.

*ngFor is like a for loop for your HTML. The expression `let tag of tags` takes the `tags` array from our component and iterates over that array. It creates multiple copies of the element. Within it, *ngFor creates a locally scoped variable called `tag` which can be used within the content of the element that will be repeated.

Try it yourself! Add and remove some tags from the `tags` array in your component.

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  ...
  isEditing: boolean = false;
  newTagName: string;
  newTagErrorText: string;

  addTag() {
    if (this.newTagName) {
      if (this.tags.indexOf(this.newTagName) === -1) {
        this.tags.push(this.newTagName);
        this.newTagName = '';
        this.newTagErrorText = '';
      } else {
        this.newTagErrorText = 'Tag name already exists';
      }
    } else {
      this.newTagErrorText = 'Please enter a tag name.';
    }
  }
}
```

Adding a Method

Now we can add some more sophisticated functionality to our application. Let's implement the ability for a user to add a tag.

First, we'll define two additional properties: `newTagName` and `newTagErrorText`. These will store the name of the new tag, and the error to be displayed to the user (if any).

Next, we'll add a public method called `addTag()` that will check if the tag name is non-empty and if it's not in the array. If these conditions are true, it will add the new tag.

app.component.html

```
<section class="image-view-detail">
  <!-- … -->
  <div class="image-view-detail-meta">
    <h2>{{title}}</h2>
    …
    <div class="meta-group">
      <label class="meta-label" for="">Tags:</label>
      <span class="meta-data">
        <span *ngFor="let tag of tags" class="tag">
          <a href>{{tag}}</a>
        </span>
      </span>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Add Tag:</label>
      <span class="meta-data">
        <input type="text"
               name="new-tag-name" [(ngModel)]="newTagName" />
        <input type="button" (click)="addTag()" value="Add">
      </span>
      <span *ngIf="newTagErrorText" class="meta-data">
        <small>{{ newTagErrorText }}</small>
      </span>
    </div>
  </div>
</section>
```

Update the Template

Next, we'll add our input HTML elements and bind them properties and events.

We'll also use `*ngIf` to conditionally display an error (if any).

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  ...
  isEditing: boolean = false;
  newTagName: string;
  newTagErrorText: string;

  addTag() {
    if (this.newTagName) {
      if (this.tags.indexOf(this.newTagName) === -1) {
        this.tags.push(this.newTagName);
        this.newTagName = '';
        this.newTagErrorText = '';
      } else {
        this.newTagErrorText = 'Tag name already exists';
      }
    } else {
      this.newTagErrorText = 'Please enter a tag name.';
    }
  }

  removeTag(e, idx) {
    e.preventDefault();
    this.tags.splice(idx, 1);
  }
}
```

Adding another Method

We'll add another method that allows the user to remove one of the tags from the list. We'll call this method `removeTag()`.

app.component.html

```
<section class="image-view-detail">
  <!-- … -->
  <div class="image-view-detail-meta">
    <h2>{{title}}</h2>
    …
    <div class="meta-group">
      <label class="meta-label" for="">Tags:</label>
      <span class="meta-data">
        <span *ngFor="let tag of tags; let i = index"
              class="tag">
          <a href (click)="removeTag($event, i)">{{tag}}</a>
        </span>
      </span>
    </div>
    <div class="meta-group">
      <label class="meta-label" for="">Add Tag:</label>
      <span class="meta-data">
        <input type="text"
               name="new-tag-name" [(ngModel)]="newTagName" />
        <input type="button" (click)="addTag()" value="Add">
      </span>
      <span *ngIf="newTagErrorText" class="meta-data">
        <small>{{ newTagErrorText }}</small>
      </span>
    </div>
  </div>
</section>
```

Update the Template

We'll also update the template. We induce another locally scope variable from *ngFor, called `index`. Index keeps track of which tag we're looking at, which we pass into the method of the `(click)` binding, `removeTag()`.

styles.css

```
@font-face {
    font-family: 'open_sansregular';
    src: url('assets/fonts/opensans-regular-webfont.woff2') format('woff2'),
         url('assets/fonts/opensans-regular-webfont.woff') format('woff');
    font-weight: normal;
    font-style: normal;
}

html, body {
  min-height: 100%;
  background-color: #444444;
  color: #FFFFFF;
  margin: 0;
  padding: 0;
}

body {
  font-family: 'open_sansregular', Helvetica, Arial, sans-serif;
  position: relative;
}
```

Add Global Styles

Finally, we'll add styling to our application. From the `style.css` of the Enspire App Static template, copy these styles only into `src/styles.css`.

These are global styles which would apply to all components within the application.

You'll also need to add the fonts folder to your project, and copy the font files from the Enspire App Static template.

Add Component Styles

Then, we'll add the remaining styles from the style.css file to our app's component in `src/app/app.component.css`.

Copenhagen

Title: Copenhagen
Click to edit

Author: John Smith

Date Taken: Sat Apr 01 2017 00:00:00 GMT+0200 (CEST)

Tags: port københavn pictures

Add Tag:

Add