

A silhouette of a person's head in profile, facing left, set against a warm, orange-toned background. The silhouette is dark and detailed, showing the hair and facial features. The background has a subtle gradient, with the orange being more vibrant on the left and darker on the right.

Web Apps in Angular: Part 2

Frameworks and Architectures of the Web

Spring 2018

Today's Program


From Last Week's Tutorial

Model Classes, Services and HTTP / JSONP

Breaking Our App into Components

Break

Feedback of Deliverables

 **Smoke & Mirrors**
Agnes Obel
Aventine

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							

Continuing from Last Week

We'll pick up right where we left off from last week. There are three ZIP files on the LearnIT site:

- **Enspire Angular App from Week 17:** The current state of the Angular demo from last week.
- **Enspire Static App Template:** Static HTML, CSS and TypeScript files that we'll continue to use in our angular app.
- **Angular Components and Modules:** Complete TypeScript examples of components and modules.

```
cd enspire-app-from-week17
```

```
npm install
```

```
ng serve
```

Setup Last Week's Project

We'll start by running last week's project. We'll assume at this point that you'll already have NodeJS and Angular CLI installed.

Change the directory (cd) to your project folder, then `ng serve`.

Alternatively, you can also work with your project from the StackBlitz online editor:
<https://stackblitz.com/edit/enspire-app-demo>



København

Title: København

Author: Tim Wray

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

Tags: one two three four harbour

Add:

Add

LAST WEEK

Introduction to Angular

Build Tools & Environment

Template - Component Relationship

Interpolation, Data Binding

Structural Directives (*ngIf, *ngFor)

THIS WEEK

Model Classes and Data Sources

Services

HTTP / JSONP

Components and Component Communication

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'København';
  author = 'Tim Wray';
  dateTaken = new Date('1 April 2017');
  imageURL = 'http://farm5.staticflickr.com/4177/34380247115_ffff2849ee_m.jpg';
  URL = 'http://www.flickr.com/photos/bertrandsketch/34380247115/';
  tags = ['one', 'two', 'three', 'four', 'harbour'];

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

  ...
}
```

Creating a Model Class

Currently, everything related to the data of the application is arbitrarily stored within our app component.

We would like to formalise the type of data we'll be dealing with, and separate the concerns of **data retrieval** and **view logic**.

The component should only handle the logic contained within the view (the template), whereas data storage / retrieval should be handled separately within a service.

This will allow us to expand and easily maintain our application.

photo.ts

```
export class Photo {  
  title: string;  
  URL: string;  
  imageURL: string;  
  author: string;  
  dateTaken: Date;  
  tags: string[];  
}
```

Creating a Model Class

In your src/app folder, create a new file called photo.ts.

This will be our TypeScript class that defines a Photo.

app.component.ts

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

```
import { Photo } from '../photo'
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'København';
  author = 'Tim Wray';
  dateTaken = new Date('1 April 2017');
  imageURL = 'http://farm5.staticflickr.com/
4177/34380247115_eff2849ee_m.jpg';
  URL = 'http://www.flickr.com/photos/bertrandsketch/
34380247115/';
  tags = ['one', 'two', 'three', 'four', 'harbour'];

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

  ...
}
```

Import the Photo Class

Next, import the Photo class so that your app component recognises it.

app.component.ts

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

import { Photo } from '../photo'

const photos: Photo[] = [{ ... }];

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'København';
  author = 'Tim Wray';
  dateTaken = new Date('1 April 2017');
  imageURL = 'http://farm5.staticflickr.com/4177/34380247115_ffff2849ee_m.jpg';
  URL = 'http://www.flickr.com/photos/bertrandsketch/34380247115/';
  tags = ['one', 'two', 'three', 'four', 'harbour'];

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

  ...
}
```

Add Static Data

Now, we'll be adding static data to our project. From the Enspire Static App Template project, copy the entire array of `const photos: Photo[] = [{ ... }]` from `photo.ts`, and add it to your app component.

This is a large static array of photographs. Make sure you paste this array before the `@Component` decorator, but after your import statements.

app.component.ts

...

```
export class AppComponent {
```

```
  photo: Photo = photos[0];
```

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

```
  addTag() {
    if (this.newTagName) {
      if (this.photo.tags.indexOf(this.newTagName) === -1) {
        this.photo.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.photo.tags.splice(idx, 1);
  }
}
```

```
}
```

Reference a Single Photo

Our component will now reference a single Photo object from the const photos array.

Update the component accordingly, including any references to the photo's data.

app.component.html

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

Update the Template

We'll also need to update the component's template, so that it now references the photo field within the component (a partial demonstration is shown here).

app.component.ts

...

```
export class AppComponent {
```

```
  photo: Photo = photos[5];
```

```
  isEditing: boolean = false;
```

```
  newTagName: string;
```

```
  newTagErrorText: string;
```

```
  addTag() {
```

```
    if (this.newTagName) {
```

```
      if (this.photo.tags.indexOf(this.newTagName) === -1) {
```

```
        this.photo.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.photo.tags.splice(idx, 1);
```

```
  }
```

```
}
```

Update the Photo's Reference

Try updating the photo's reference within the app component. It now references a different photo from the array.

photo.service.ts

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

```
@Injectable()  
export class PhotoService { }
```

Create a PhotoService

It would be much cleaner to separate the logic that handles the storage and retrieval of photos from the components that require them.

For this reason, we'll create a PhotoService that can be used by any component to retrieve and display photos.

In your `src/app` folder, create a new file called `photo.service.ts` and add the following code.

photo.service.ts

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

```
const photos: Photo[] = [{ ... }];
```

```
@Injectable()  
export class PhotoService { }
```

Add the photos array

Next, we'll add

`const photos: Photo[] = [{ ... }]`
from `app.component.ts` to the service, so
that the data is stored and contained within the
service.

photo.service.ts

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

const photos: Photo[] = [{ ... }];

@Injectable()
export class PhotoService {

  getPhotos(): Promise<Photo[]> {
    return Promise.resolve(photos);
  }

}
```

Add getPhotos() method

Then, we'll create a method called `getPhotos()` that returns a resolved Promise, which consists of the array of photos instantiated by `const photos`.

app.module.ts

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

import { PhotoService } from '../photo.service';

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

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [PhotoService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Add it to the app.module

So now we've created a service that can be used to retrieve some photos for us. This service can be used by any component within our application. To actually import and use this service, we'll need to do a few things.

First of all, open the app.module.ts file. This is the application module, and it contains everything Angular needs to know about our application, including its dependencies and services.

Import and add the PhotoService to the app module, as shown.

app.component.ts

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

import { Photo } from '../photo';

import { PhotoService } from '../photo.service';

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

  photo: Photo;

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

  constructor(
    private photoService: PhotoService
  ) { }

  ngOnInit() {
    this.photoService.getPhotos().then(photos => this.photo =
      photos[0]);
  }

  ...

}
```

Import / call PhotoService

In our app component, we'll import the service and inject it via its constructor method. The constructor method creates a private variable called `photoService` that will allow the component to reference the service within the app.

We want to retrieve and display the photos when the component is initialised, to this end, we implement the `OnInit` interface via the `ngOnInit()` method. The `ngOnInit()` method is called as part of the component's life-cycle, and the method gets called when the component is first initialised.

app.component.html

```
<section class="image-view-detail">
  <div *ngIf="!photo" class="message-container">
    <p class="status-message">No image selected.</p>
  </div>
  <div *ngIf="photo" class="image-view-detail-image-container">
    <figure class="image-view-detail-
image" [style.backgroundImage]='url(' + photo.imageUrl + ')'>
      <img [src]="photo.imageUrl" [alt]="title" />
    </figure>
  </div>
  <div *ngIf="photo" class="image-view-detail-meta">
    ...
  </div>
</section>
```

Add *ngIf guards

Since we will be dealing with data asynchronously, it's a good idea to add guards to your template so that certain portions of your template would only be rendered if a value is present.

In this example, if the component's photo field is `null` or uninitialised, it will display a 'no image selected' message to the user.

app.component.html

```
<section class="image-view-detail">
  ***
  <div *ngIf="photo" class="image-view-detail-meta">
    <h2>{{photo.title}}</h2>
    ***
    <div class="meta-group">
      <label class="meta-label" for="">Date Taken:</label>
      <span class="meta-data">
        {{photo.dateTaken | date:'dd MMMM yyyy'}}
      </span>
    </div>
    ***
  </div>
</section>
```

Format the Date using Pipes

Next, we'll continue along and build on the functionality of our application.

Angular allows you to transform and format data within a template via the use of pipes. Pipes are commonly used, for example, to format dates and currency.

Currently, the 'date Taken' field within our template doesn't look very nice. We'll use the date pipe to format our dateTaken property (which is a JavaScript Date object) into a format that's a little more readable.

app.component.html

```
<section class="image-view-master">
  <figure class="thumbnail" style="background-image:
url(&quot;http://farm3.staticflickr.com/
2839/33571103773_46a66bbcbf_m.jpg&quot;);">
    
  </figure>
  <figure class="thumbnail" style="background-image:
url(&quot;http://farm3.staticflickr.com/
2873/34250459111_8a9094a638_m.jpg&quot;);">
    
  </figure>
  . . .
</section>
<section class="image-view-detail">
  . . .
</section>
```

Add the Image Gallery

Now, we'll add some functionality to our application that allows us to browse and select photos via an image gallery.

From the **Enspire Static App Template**, copy and paste the entire `<section>` with the class name of `image-view-master`. Add it to your app component's template.

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Photo } from '../photo';
import { PhotoService } from '../photo.service';

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

  photos: Photo[] = [];
  photo: Photo;

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

  constructor(
    private photoService: PhotoService
  ) { }

  ngOnInit() {
    this.photoService.getPhotos().then(photos => this.photos =
    photos.splice(0, 16));
  }

  ...

}
```

Add a photos field

Next, we'll add an array, called photos to our component that displays the list of photos.

We'll update the service call within our ngOnInit() to initialise and assign the results from the PhotoService to the photos array. We'll use the splice() operator to ensure that at most, only the first 16 photos are displayed.

app.component.html

```
<section class="image-view-master">
  <figure *ngFor="let photoInList of photos"
    class="thumbnail"
    [style.backgroundImage]='url('+photoInList.imageURL+')'
    [class.selected]="photo === photoInList"
    (click)="photo = photoInList">
    <img [src]="photoInList.imageURL" [alt]="photoInList.title">
  </figure>
  ...
</section>
<section class="image-view-detail">
  ...
</section>
```

app.component.css

```
section.image-view-master .thumbnail.selected {
  opacity: 0.7;
}
```

Update the Template

Now we'll update the template to display the photo gallery. We'll use an `*ngFor` to display each photo.

We'll also add a `click` event handler so that, when each photo is clicked on, the component's `photo` property is assigned to the photo that was clicked on.

Finally, we'll use Angular's class binding to conditionally display the `selected` class (and show that the photo is highlighted) when the photo in the list is set as the `photo` property on the component. We'll also add this class to our stylesheet.

app.module.ts

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

import { PhotoService } from '../photo.service';

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

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    JsonpModule
  ],
  providers: [PhotoService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Retrieve Photos from Flickr

Next, we might actually want to be able to retrieve photos from an external API.

We'll use the Flickr API to do this, and we'll use JSONP to retrieve the data cross-domain.

To use JSONP, we'll need to import and declare this functionality within our app module.

photo.service.ts

```
import { Injectable } from '@angular/core';
import { Jsonp, URLSearchParams } from '@angular/http';
import { Photo } from '../photo';

@Injectable()
export class PhotoService {

  private flickrAPIURL: string = 'https://api.flickr.com/services/feeds/
photos_public.gne';

  constructor(
    private jsonp: Jsonp
  ) { }

  getPhotos(search: string): Promise<Photo[]> {
    return new Promise((resolve, reject) => {
      let params = new URLSearchParams();
      params.set('tags', search);
      params.set('tagmode', 'any');
      params.set('format', 'json');
      params.set('jsoncallback', 'JSONP_CALLBACK');
      this.jsonp.get(this.flickrAPIURL, {'search':
params}).subscribe(data => {
        let photos: Photo[] = [];
        let photosJSON = data.json().items.slice(0, 16);
        photosJSON.map(photoJSON => {
          photos.push({
            title: photoJSON.title,
            URL: photoJSON.link,
            imageURL: photoJSON.media.m,
            author: photoJSON.author,
            dateTaken: new Date(photoJSON['date_taken']),
            tags: photoJSON.tags.split(' ').slice(0, 10)
          });
        });
        resolve(photos);
      });
    });
  }
}
```

Retrieve Photos from Flickr

Next, we'll update the PhotoService class so that it now does the following:

- Imports the Jsonp and URLSearchParams classes
- Calls the Flickr API using a search parameter that can be passed into getPhoto().
- Formats the result as an array of Photos.
- Returns the result as a Promise.

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Photo } from '../photo';
import { PhotoService } from '../photo.service';

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

  photos: Photo[] = [];
  photo: Photo;

  isSearchingPhotos: boolean = true;
  isEditing: boolean = false;
  newTagName: string;
  newTagErrorText: string;

  constructor(
    private photoService: PhotoService
  ) { }

  ngOnInit() {
    this.photoService.getPhotos('copenhagen').then(photos => {
      this.photos = photos.splice(0, 16);
      this.isSearchingPhotos = false;
    });
  }
  ...
}
```

Retrieve Photos from Flickr

Within the app component, we'll update our method call so that it passes in the string 'copenhagen' as our search parameter.

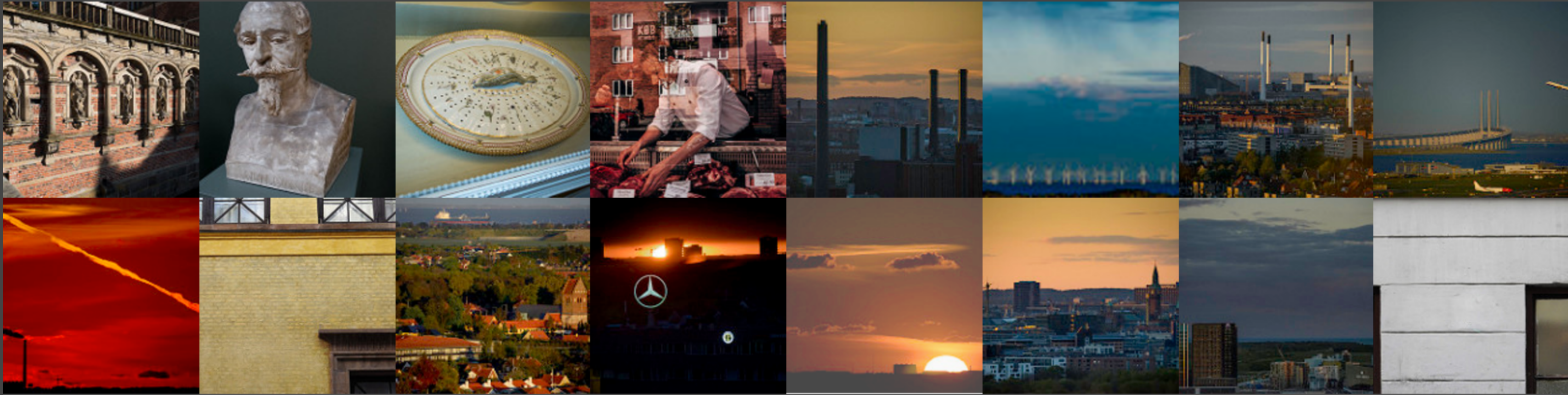
We'll also add another state variable called `isSearchingPhotos` that keeps track of whether the application is waiting for a response from the Flickr API.

app.component.html

```
<section class="image-view-master">
  <div *ngIf="isSearchingPhotos" class="message-container">
    <p class="loading-message">Searching for images ...</p>
  </div>
  <div *ngIf="!isSearchingPhotos">
    <figure *ngFor="let photoInList of photos"
      class="thumbnail"
      [style.backgroundImage]="url('+photoInList.imageURL+')"
      [class.selected]="photo === photoInList"
      (click)="photo = photoInList">
      <img [src]="photoInList.imageURL"
        [alt]="photoInList.title">
    </figure>
  </div>
  ...
</section>
<section class="image-view-detail">
  ...
</section>
```

Retrieve Photos from Flickr

Finally, within our template, we'll conditionally display a loading message if `isSearchingPhotos === true`. Once `isSearchingPhotos` is set to `false`, then the gallery would display.



Copenhagen Sunset | 170508-0745-jikatu

Title: Copenhagen Sunset | 170508-0745-jikatu

Author: nobody@flickr.com ("jikatu")

Date Taken: 09 May 2017

Tags: 150600mm copenhagen cph d800
danish dinamarca dk jikatu nikon
sigma

Add:

Add

Creating Components

Next, we'll split our app up into components. The app will consist of:

- A photo-master component, that loads the photos and displays the image gallery.
- A photo-detail component, that displays the detail of a single photograph.
- The app-component, which serves as a container for the above two components.

ng g component photo-detail

photo-detail Component

First, we'll create the photo-detail component. This can be done via the command line, which will set up our component for us, create the necessary files, and add it to the app's module.

photo-detail.component.html

```
<section class="image-view-detail">
  <div *ngIf="!photo" class="message-container">
    <p class="status-message">No image selected.</p>
  </div>
  <div *ngIf="photo" class="image-view-detail-image-container">
    <figure class="image-view-detail-
image" [style.backgroundImage]='url(' + photo.imageUrl + ')'>
      <img [src]="photo.imageUrl" [alt]="title" />
    </figure>
  </div>
  <div *ngIf="photo" class="image-view-detail-meta">
    <h2>{{photo.title}}</h2>
    ...
  </div>
</section>
```

photo-detail Component

Within the template of this component, copy the entire `<section>` element with class `image-view-detail` from `app.component.html`.

Within `photo-detail.component.css`, copy all styles under the `/* Image View Detail */` heading from `app.component.css`.

photo-detail.component.ts

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

import { Photo } from '../photo';

@Component({
  selector: 'photo-detail',
  templateUrl: './photo-detail.component.html',
  styleUrls: ['./photo-detail.component.css']
})
export class PhotoDetailComponent {

  @Input() photo: Photo;

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

  addTag() {
    ...
  }

  removeTag(e, idx) {
    ...
  }
}
```

photo-detail Component

Add the relevant logic to the photo-detail component.

Notice the `@Input()` decorator. This specifies that we can one-way bind a photo attribute to this component, as we'll see in the next section.

app.component.html

```
<section class="image-view-master">
  ...
</section>
<photo-detail [photo]="photo"></photo-detail>
```

Update app-component

Within the app component, we'll add an instance of the photo-detail component to the template, and we will one-way bind the photo attribute using [].

Any updates to photo within the app component will automatically be reflected within the photo-detail component.

ng g component photo-master

photo-master component

Next, we'll create the photo-master component.

The photo-master component will load and display images from the API.

Whereas the photo-detail component receives input via the [photo] binding, the photo-master component will output a selected photo via the (select) binding.

To create the component, execute the Angular command on the left.

photo-master.component.html

```
<section class="image-view-master">
  <div *ngIf="isSearchingPhotos" class="message-container">
    <p class="loading-message">Searching for images ...</p>
  </div>
  <div *ngIf="!isSearchingPhotos">
    <figure *ngFor="let photoInList of photos"
      class="thumbnail"
      [style.backgroundImage]="url('+photoInList.imageUrl+')"
      [class.selected]="selectedPhoto === photoInList"
      (click)="selectPhoto(photoInList)">
      <img [src]="photoInList.imageUrl"
        [alt]="photoInList.title">
    </figure>
  </div>
</section>
```

photo-master Component

Within the template of this component, copy the entire `<section>` element with class `image-view-master` from `app.component.html`. Notice that we've update the `(click)` event binding for the photo to call a method called `selectPhoto()`.

Within `photo-master.component.css`, copy all styles under the `/* Image View Master */` heading from `app.component.css`.

photo-master.component.ts

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
import { Photo } from '../photo';
import { PhotoService } from '../photo.service';

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

  photos: Photo[] = [];
  selectedPhoto: Photo;
  isSearchingPhotos: boolean = true;

  @Output() select: EventEmitter<Photo> = new EventEmitter<Photo>();

  constructor(
    private photoService: PhotoService
  ) { }

  ngOnInit() {
    this.photoService.getPhotos('copenhagen').then(photos => {
      this.photos = photos.splice(0, 16);
      this.isSearchingPhotos = false;
    });
  }

  selectPhoto(photo: Photo): void {
    this.selectedPhoto = photo;
    this.select.emit(this.selectedPhoto);
  }
}
```

photo-master Component

Add the relevant logic to the photo-master component. We can see that it is now the photo-master component which calls the PhotoService to retrieve and display the photos.

Notice the @Output() decorator. This means we can bind an event on select. The event can be triggered when the user clicks on a photo. Within the selectPhoto() method, the event is fired. Our app component can 'listen' to this event and act accordingly.

app.component.html

```
<photo-master (select)="photoSelect($event)"></photo-master>  
<photo-detail [photo]="photo"></photo-detail>
```

app.component.ts

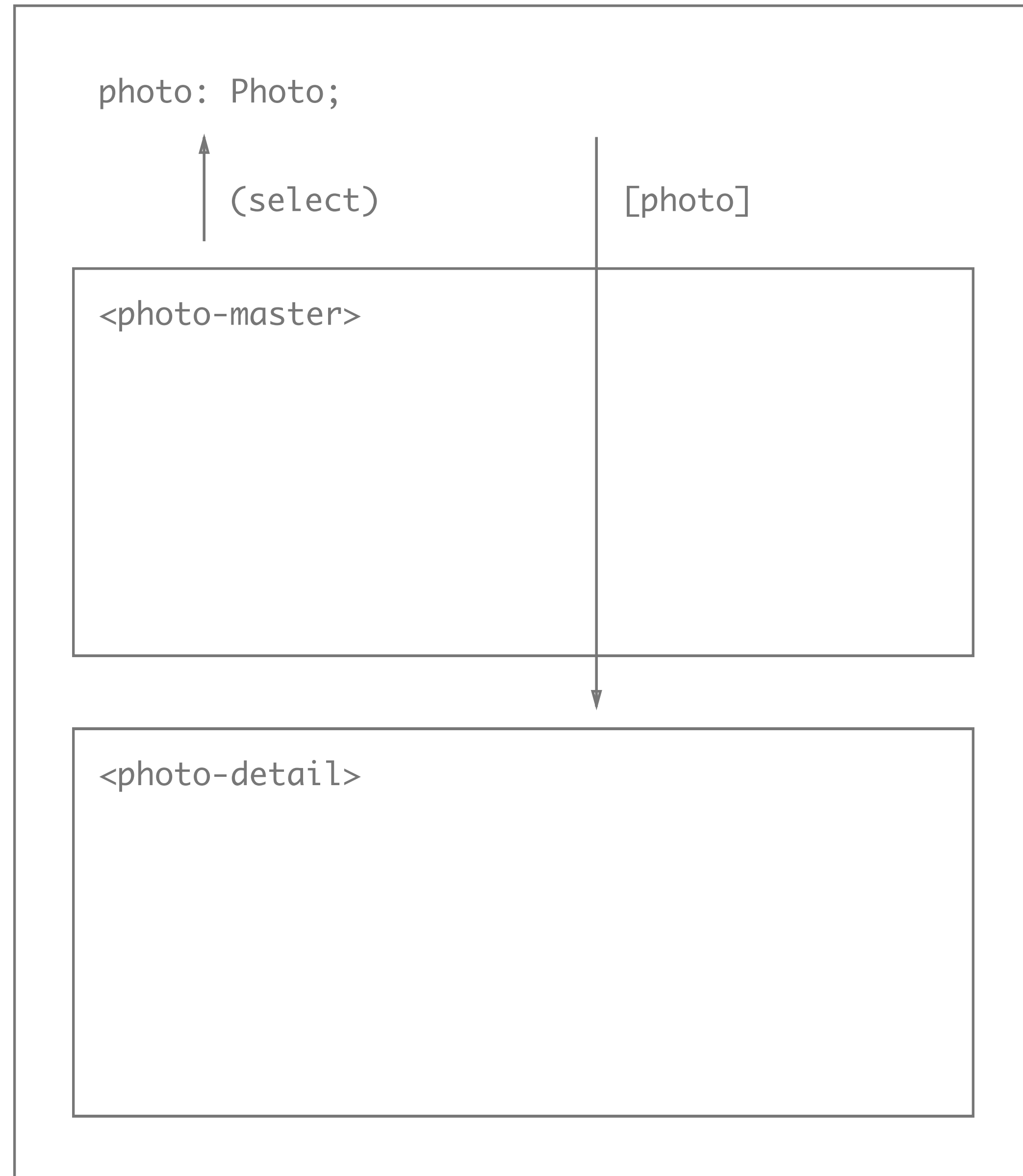
```
import { Component } from '@angular/core';  
import { Photo } from '../photo';  
  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  
  photo: Photo;  
  
  photoSelect(photo: Photo) {  
    this.photo = photo;  
  }  
  
}
```

Update app-component

Our app component is now a container for the photo-master and photo-detail components. All it does is hold these components and binds a single photo property.

The photo property is updated when the select event is fired from the photo-master component. This property is bound to the photo-detail component, so when it gets updated, the corresponding photo is displayed in photo-detail.

<app-root>



Component Communication

Components allow you to modularise, reuse and individually test portions of your Angular app. Components exist in a hierarchy, where they can be nested within other components.

Components pass data to child components via the use of property bindings `[]`.

Likewise, components pass data to parent components via the use of event bindings `()`.

Thank You!