Single-Page Applications

1. Overview of SPAs
2. Creating components
3. Angular routing

```
Demo app:    AngularDev/Demos/02-SinglePageApps/DemoApp
To install:  npm install
To run:      ng serve
```

Section 1: Overview of SPAs

- What is an SPA?
- Demo application
- Creating an application with routing

What is an SPA?

- According to Wiki:

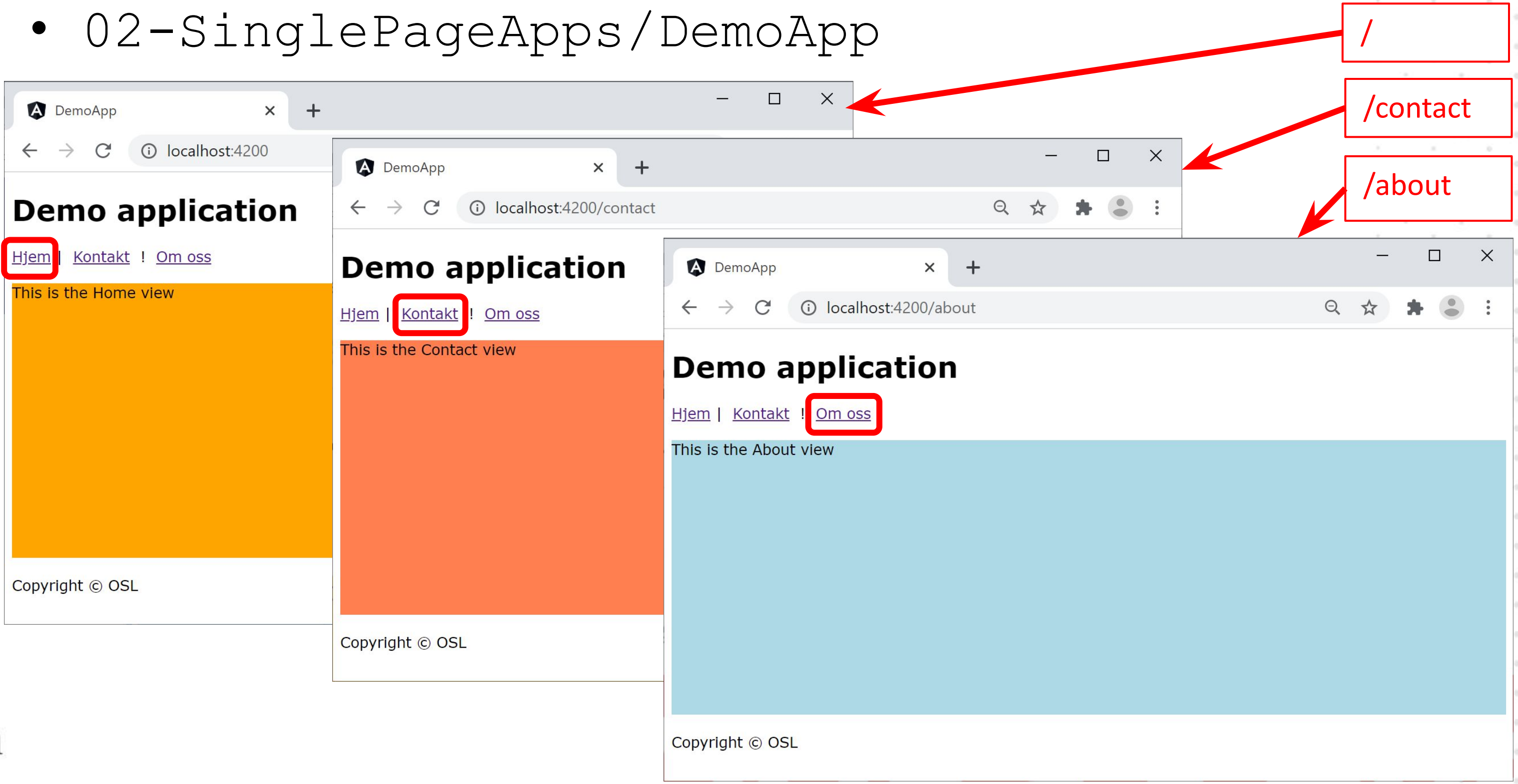
A single-page application is a web app that fits on a single web page with the goal of providing a more fluent user experience similar to a desktop application.

In an SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load, or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions. The page does not reload at any point in the process, nor does control transfer to another page.

Interaction with the SPA often involves dynamic communication with the web server behind the scenes.

Demo Application (1 of 2)

- Angular has good support for SPAs, see this demo:
  - `02-SinglePageApps/DemoApp`



Demo application

Hjem | Kontakt | Om oss

This is the Home view

Copyright © OSL

Demo application

Hjem | Kontakt | Om oss

This is the Contact view

Copyright © OSL

Demo application

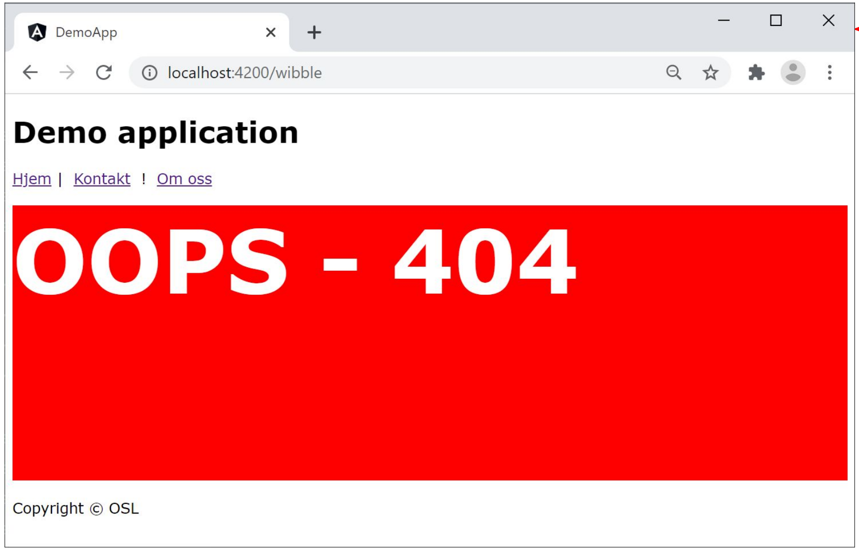
Hjem | Kontakt | Om oss

This is the About view

Copyright © OSL

Demo Application (2 of 2)

- The application also has a "page not found" component for unrecognised URLs



Demo application

Hjem | Kontakt ! Om oss

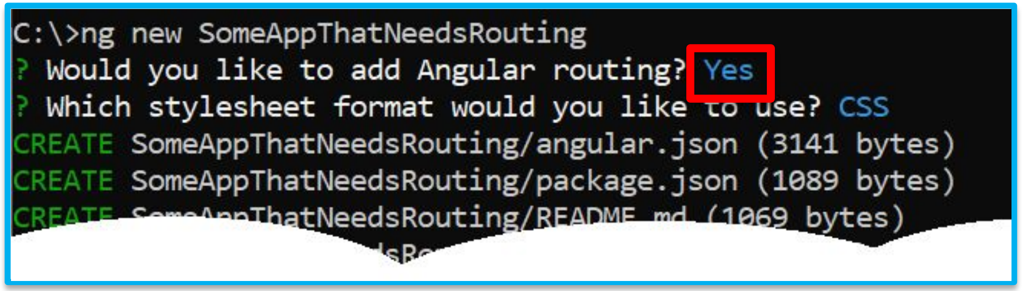
OOPS - 404

Copyright © OSL

/wibble

Creating an Application with Routing

- If you're using Angular CLI to create your application...
  - Then you should answer "yes" when it asks if you'd like to add Angular routing to the app



```
C:\>ng new SomeAppThatNeedsRouting
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE SomeAppThatNeedsRouting/angular.json (3141 bytes)
CREATE SomeAppThatNeedsRouting/package.json (1089 bytes)
CREATE SomeAppThatNeedsRouting/README.md (1069 bytes)
```

- We'll explain all about routing later in this chapter...
  - First we'll see how to create new components

Section 2: Creating Components

- Overview
- Creating artifacts using Angular CLI
- How to create a component
- Reviewing the component
- Summarizing our application's components
- Reviewing the module code
- Aside: Defining global styles

Overview

- An Angular application typically has many components:
  - A root component (typically named `AppComponent`)
  - Plus many additional components

- Our demo application has 4 additional components:
  - `HomeComponent`
  - `ContactComponent`
  - `AboutComponent`
  - `PagenotfoundComponent`

Creating Artifacts using Angular CLI

- You can use Angular CLI to create artifacts in your app

Type of artifact	How to generate using Angular CLI
Component	`ng g component my-new-component`
Directive	`ng g directive my-new-directive`
Pipe	`ng g pipe my-new-pipe`
Service	`ng g service my-new-service`
Class	`ng g class my-new-class`
Interface	`ng g interface my-new-interface`
Module	`ng g module my-module`
Web worker	`ng g web-worker my-web-worker`

How to Create a Component

- This is how we created the *home* component in our demo application, using Angular CLI

```
ng g component home
```

```
CREATE src/app/home/home.component.html (19 bytes)
CREATE src/app/home/home.component.spec.ts (585 bytes)
CREATE src/app/home/home.component.ts (267 bytes)
CREATE src/app/home/home.component.css (0 bytes)
UPDATE src/app/app.module.ts (467 bytes)
```

- Angular CLI adheres to the Angular style guide:
  - https://angular.io/guide/styleguide

Reviewing the Component (1 of 3)

- Angular CLI created a new folder for the component
  - `src/app/home`

- Angular CLI created 4 files for the component, using a standardized file-naming convention:
  - `home.component.ts`
  - `home.component.html`
  - `home.component.css`
  - `home.component.spec.ts`

Pearson

Reviewing the Component (2 of 3)

- Here's the code in `home.component.ts`:
  - We'll discuss `constructor()` and `ngOnInit()` later

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

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

  constructor() {}

  ngOnInit(): void {}
}
```

src/app/home/home.component.ts

Reviewing the Component (3 of 3)

- Here's our `home.component.css`:
  - We've defined simple styles for this component

```
div {
  background-color: orange;
  height: 300px;
  margin: 20px 0px;
}
```

`src/app/home/home.component.css`

Summarizing our Application's Components

- Here's a summary of the components we generated in our app via Angular CLI:

```
ng g component home
```

See `src/app/home/`

```
ng g component contact
```

See `src/app/contact/`

```
ng g component about
```

See `src/app/about/`

```
ng g component pagenotfound
```

See `src/app/pagenotfound/`

Pearson

Reviewing the Module Code

- When you create components using Angular CLI, it updates your module to incorporate the components:

```
import {HomeComponent}         from './home/home.component';
import {ContactComponent}      from './contact/contact.component';
import {AboutComponent}        from './about/about.component';
import {PagenotfoundComponent} from './pagenotfound/pagenotfound.component';
…

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent, ContactComponent, AboutComponent, PagenotfoundComponent
  ],
  …
})
export class AppModule { }                                  src/app/app.module.ts
```

Pearson

Aside: Defining Global Styles

- If you want to define global styles that apply across the entire website, define them here:
  - `src/styles.css`

```
body {
  font-family: Verdana, Geneva, Tahoma, sans-serif;
}
```
src/styles.css

- Note: The location of global stylesheets is defined in `angular.json`, via the following property:
  - `projects.XXX.architect.build.options.styles`

Pearson

Section 3: Angular Routing

- Angular routing dependencies
- Defining a routing table
- Defining a base href
- Defining a router outlet
- Defining router links
- Running the application

Angular Routing Dependencies

- If you want your Angular app to support routing, you must have this dependency in `package.json`:

```
{
  "dependencies": {
    "@angular/router": "^14.1.0",
    …
  },
  …
}
```

`package.json`

- Note:
  - Angular CLI adds this dependency automatically, if you select the "add routing" option

Pearson

Defining a Routing Table (1 of 2)

- You must define a *routing table* for your application
  - Maps relative URLs to components in your app

Relative URL	Map the relative URL to this component…
(none)	`HomeComponent`
contact	`ContactComponent`
about	`AboutComponent`
(anything else)	`PagenotfoundComponent`

- When the user navigates to one of these URLs:
  - Angular creates an instance of the relevant component, and renders it in a *router outlet* (see later)

Defining a Routing Table (2 of 2)

- Here's the routing table for our application:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component'
import { AboutComponent } from './about/about.component'
import { ContactComponent } from './contact/contact.component'
import { PagenotfoundComponent } from './pagenotfound/pagenotfound.component'

const routes: Routes = [
  { path: '',        component: HomeComponent },
  { path: 'contact', component: ContactComponent },
  { path: 'about',   component: AboutComponent },
  { path: '**',      component: PagenotfoundComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

src/app/app-routing.module.ts

Pearson

Defining a Base href

- You must also define a base `href` for your application
  - All router URLs are considered relative to this base `href`

- You define the base `href` as follows in `index.html`:

```
<!doctype html>
<html lang="en">
<head>
  <base href="/">
  …
</head>
<body>
  <app-root></app-root>
</body>
</html>                                                src/index.html
```

Pearson

Defining a Router Outlet

- Somewhere in your UI (typically the root component), define a `<router-outlet>` where components appear

```
SOME STUFF ALWAYS RENDERED HERE …

<router-outlet></router-outlet>

SOME MORE STUFF ALWAYS RENDERED HERE …          src/app/app.component.html
```

- Angular will display the appropriate component in the router outlet, depending on the current URL

Defining Router Links

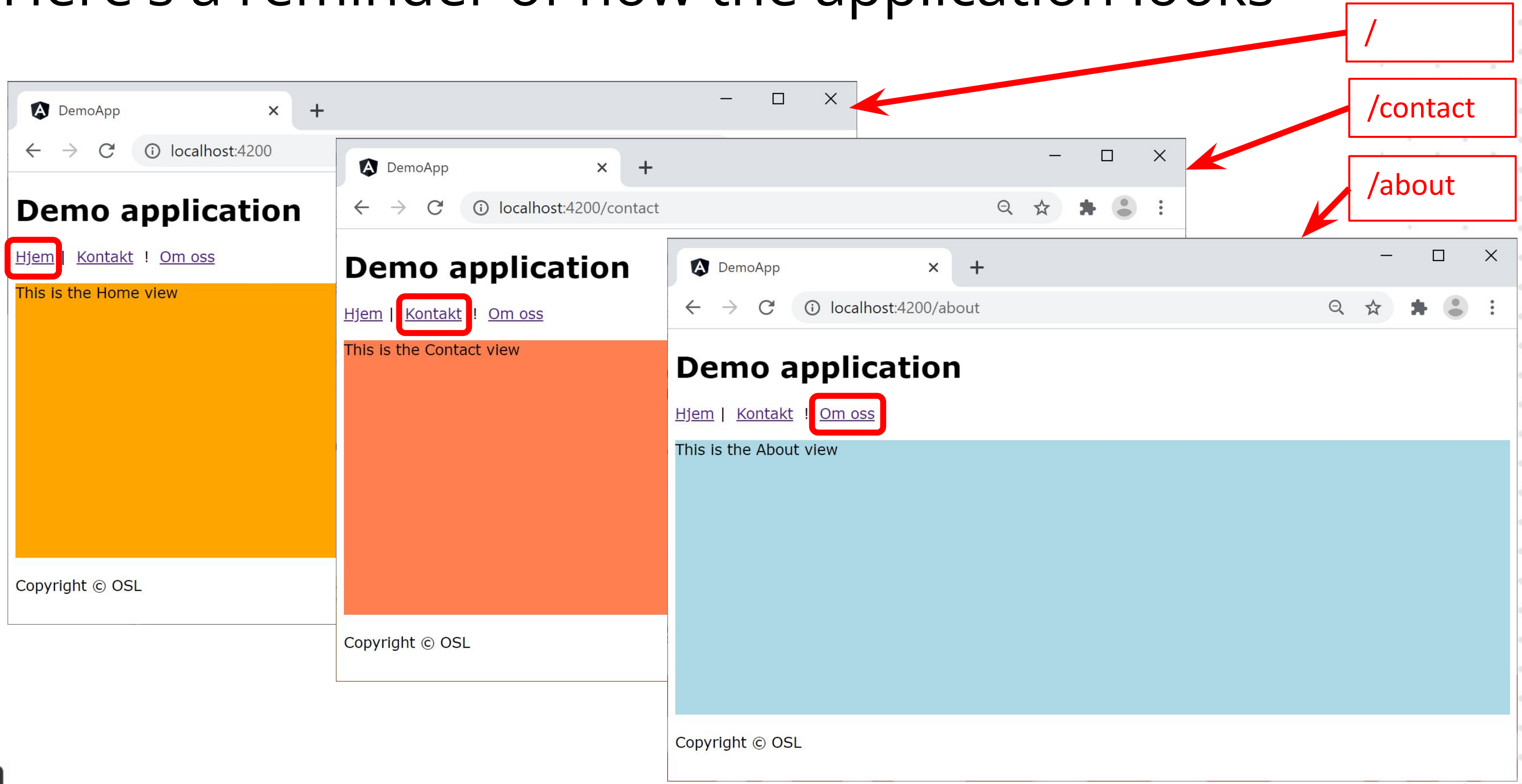
- You can define *router links* in your web pages, to help the user navigate between components
  - Router links refer to routes in the routing table, prefixed by the base `href`

```
<h1>Demo application</h1>

<nav>
  <a routerLink="/">Hjem</a>&nbsp;|&nbsp;
  <a routerLink="/contact">Kontakt</a> &nbsp;|&nbsp;
  <a routerLink="/about">Om oss</a>
</nav>

<router-outlet></router-outlet>

<p>Copyright &copy; OSL</p>
```

src/app/app.component.html

Running the Application

- Here's a reminder of how the application looks



Demo application

Hjem | Kontakt ! Om oss

This is the Home view

Copyright © OSL

Demo application

Hjem | Kontakt ! Om oss

This is the Contact view

Copyright © OSL

Demo application

Hjem | Kontakt ! Om oss

This is the About view

Copyright © OSL

Summary

- Overview of SPAs
- Creating components
- Angular routing