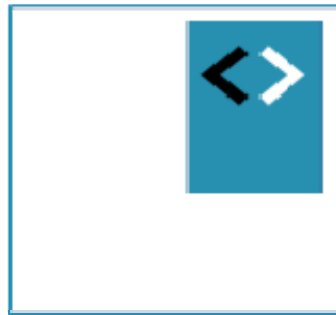




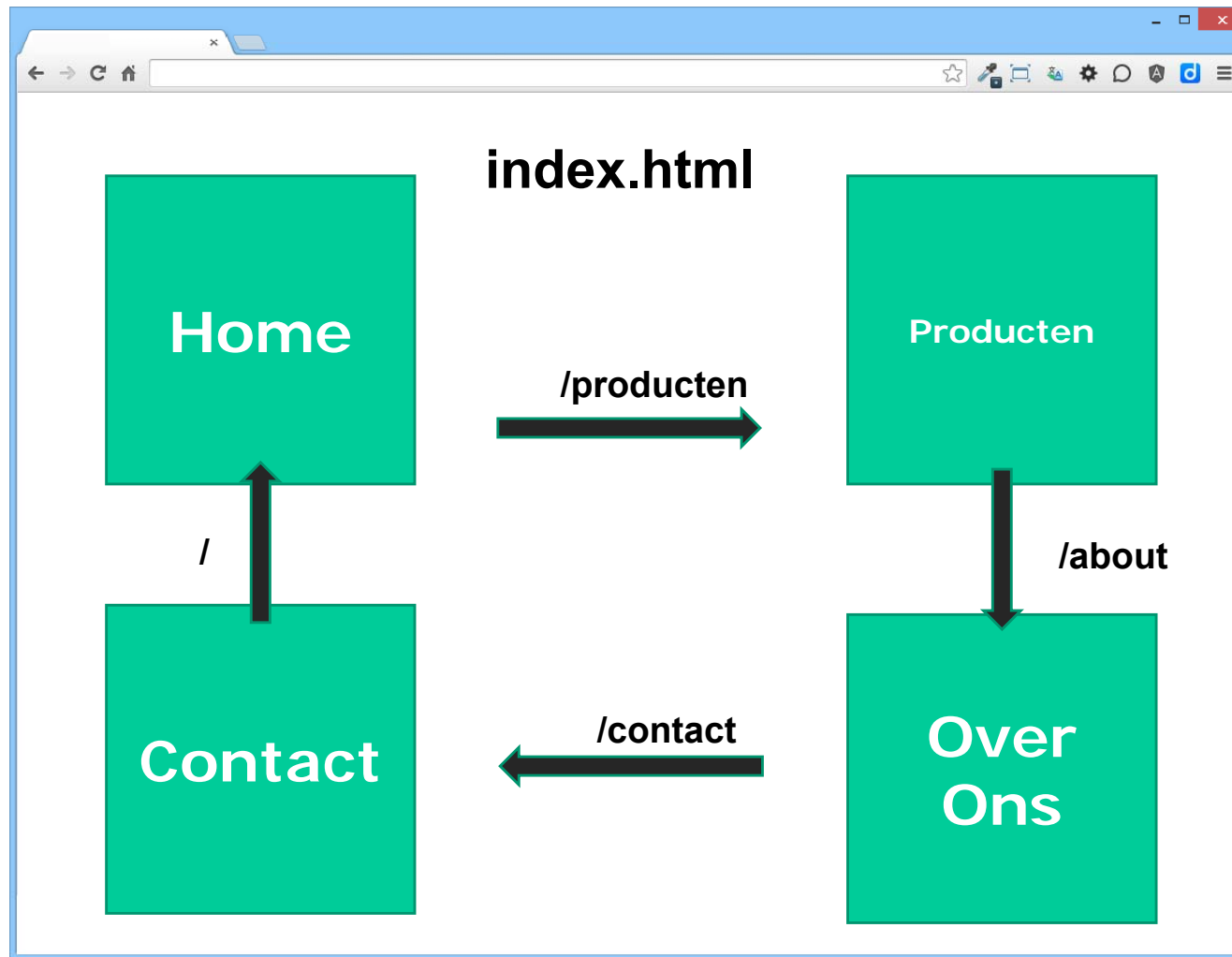
Angular Fundamentals

Module - Routing



Peter Kassenaar –
info@kassenaar.com

Routing architecture and goal



- Make use of SPA principle
- Making deep links possible

Angular 1: ng-route, of ui-router

1. `<script src="js/vendor/angular/angular-route.min.js"></script>`
2. `<div ng-view></div>`
3. `var app = angular.module('myApp', ['ngRoute']);`

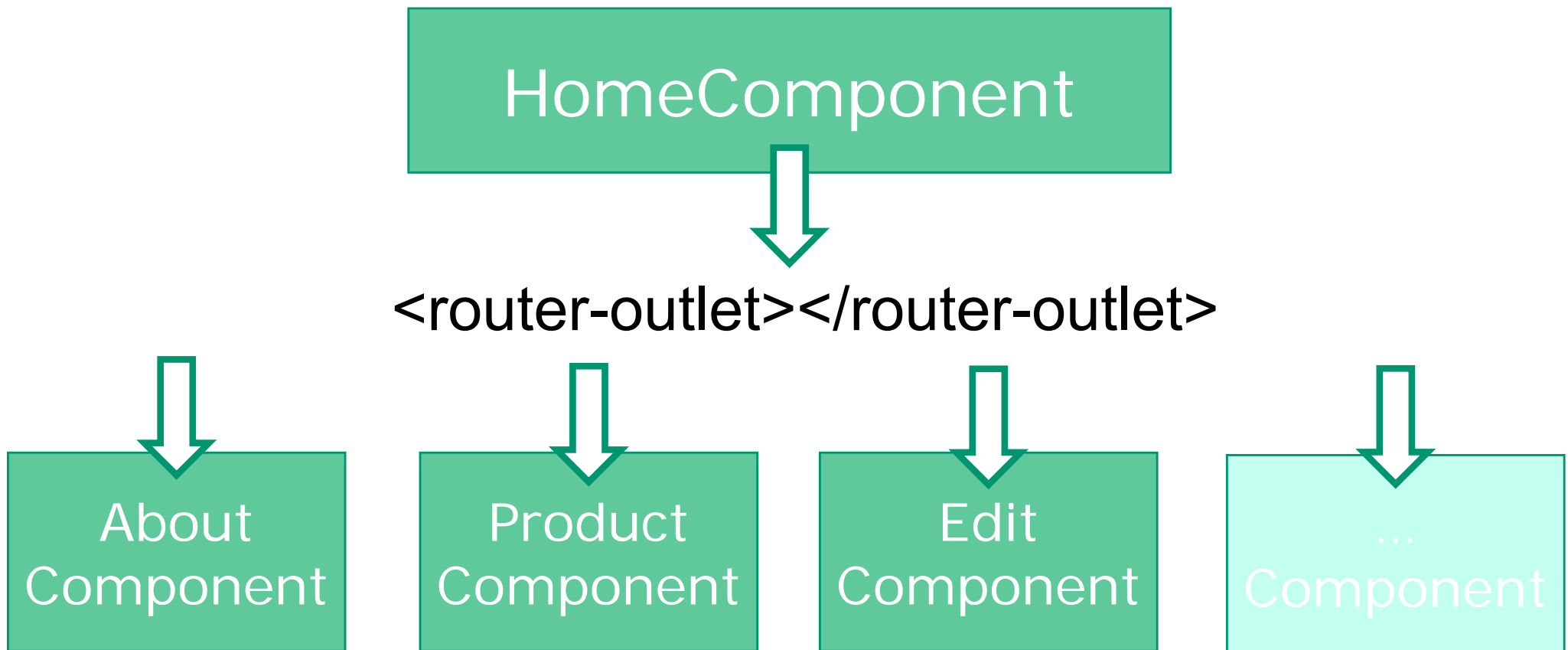
Then: configure `$routeProvider` (or `$stateProvider` with ui-router)

Angular: Component Router

- NOT available for AngularJS 1.4+
- Also not available: ui-router

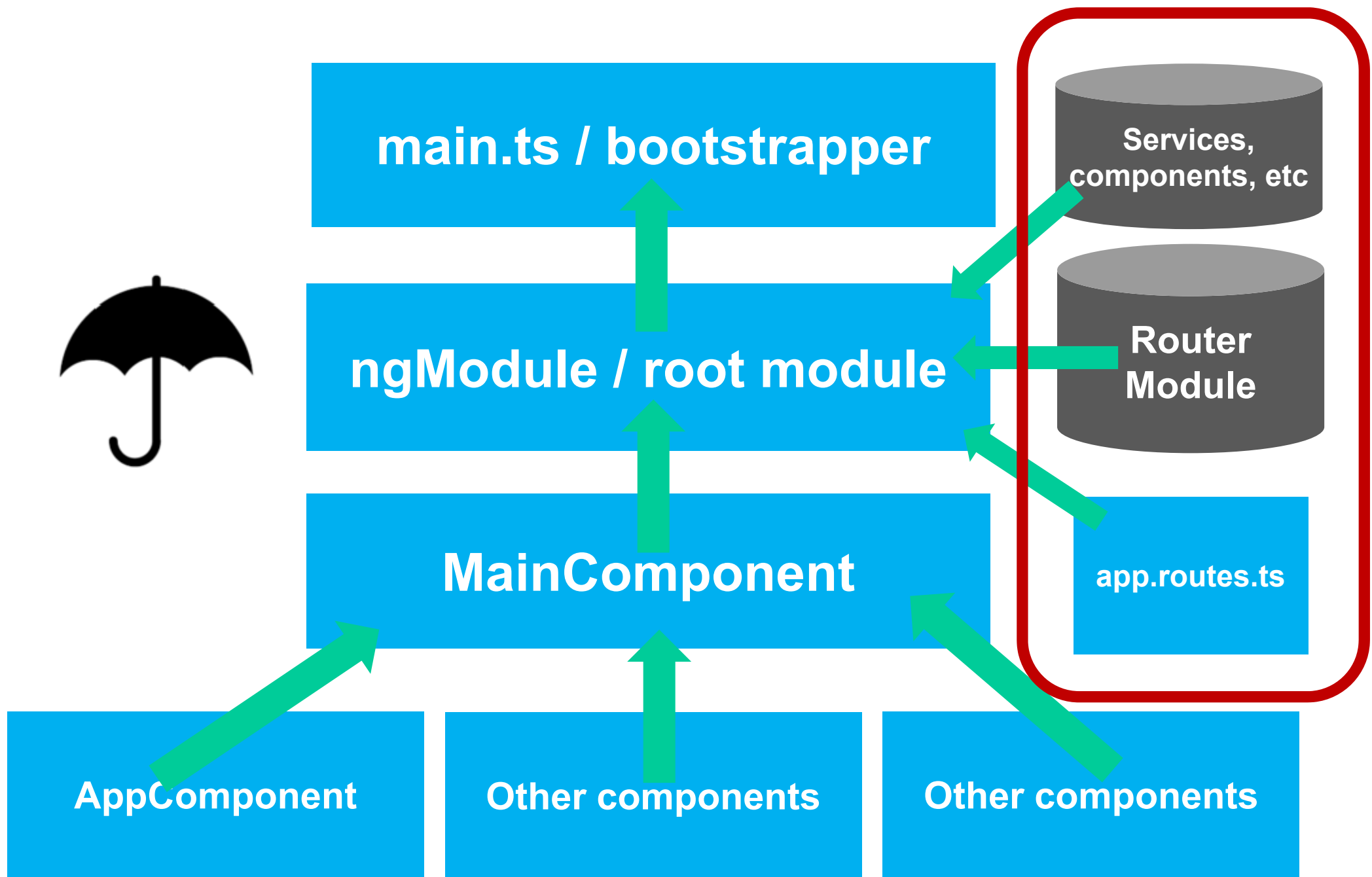
Routing – every route is a Component

- HomeComponent (or: RootComponent, whatever) with main menu
- Components are injected in `<router-outlet></router-outlet>`



Routing with Angular CLI

- Default: no routing in CLI-projects
- Add routing from the start?
 - `ng new myProject --routing`
- This creates `app.routing.module` in project
- (a little) different than the approach in this module
 - We add routing later on –you'll learn what components are used



Routing steps

1. Add base href in header of index.html (!)

`<base href="/">`

- There *can* be multiple routes per module. Each component can configure its own `ChildRoutes` – to be discussed.
- Angular-CLI adds this automatically for you

2. Add routes. Convention: `app.routes.ts` or `app.routing-module.ts`.

```
// app.routes.ts
import {Routes} from '@angular/router';
import {AppComponent} from './app.component';
import {CityAddComponent} from './city.add.component';

export const AppRoutes: Routes = [
  {path: '', component: AppComponent},
  {path: 'home', component: AppComponent},
  {path: 'add', component: CityAddComponent}
];
```

Some people or tools use different notation on declaring routes

3. Making routes available in Module

- Import RouterModule in applicatie
- Import ./app.routes in applicatie

```
...  
// Router  
import {RouterModule} from '@angular/router';  
import {AppRoutes} from './app.routes';
```

Import Router stuff

```
// Components  
import {MainComponent} from './MainComponent';
```

New!
MainComponent.
To be created

```
...  
@NgModule({  
  imports      : [  
    BrowserModule, HttpClientModule,  
    RouterModule.forRoot(AppRoutes)  
  ],  
  declarations: [  
    MainComponent,  
    AppComponent,  
    CityAddComponent  
  ],  
  bootstrap    : [MainComponent]  
})  
export class AppModule {  
}
```

Configure
RouterModule.forRoot()

MainComponent is now
bootstrapped

4. MainComponent met Routing maken

- New component with main menu and `<router-outlet>`

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

```
@Component({
  selector: 'main-component',
  template: `
    <h1>Pick your favorite city</h1>
    <!-- Static 'main menu'. Always visible-->
    <!-- Add routerLink directive. Angular replaces this with correct <a href="..."> -->
    <a routerLink="/home" class="btn btn-primary">List of cities</a>
    <a routerLink="/add" class="btn btn-primary">Add City</a>
    <hr>
    <!-- Dynamically inject views here -->
    <router-outlet></router-outlet>
    <!-- Static footer here. Always visible-->
  `
})
export class MainComponent implements OnInit {
  constructor() { }
  ngOnInit() { }
}
```

**"Hoofdmenu".
Notice routerLink**

<router-outlet>

Empty Component

5. Edit index.html

- if `MainComponent` has a different selector, update `index.html`

```
<div class="container">  
  <main-component>  
    Loading...  
  </main-component>  
</div>
```

6. Nieuwe component(en) maken en importeren

Every component is a rout

```
// city.add.component.ts
import { Component } from '@angular/core';
```

```
@Component({
  selector: '...',
  template: `...`
})

export class CityAddComponent {
  ...
}
```

```
// city.edit.component.ts
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'edit-city',
  template: `<h1>Edit City</h1>`
})

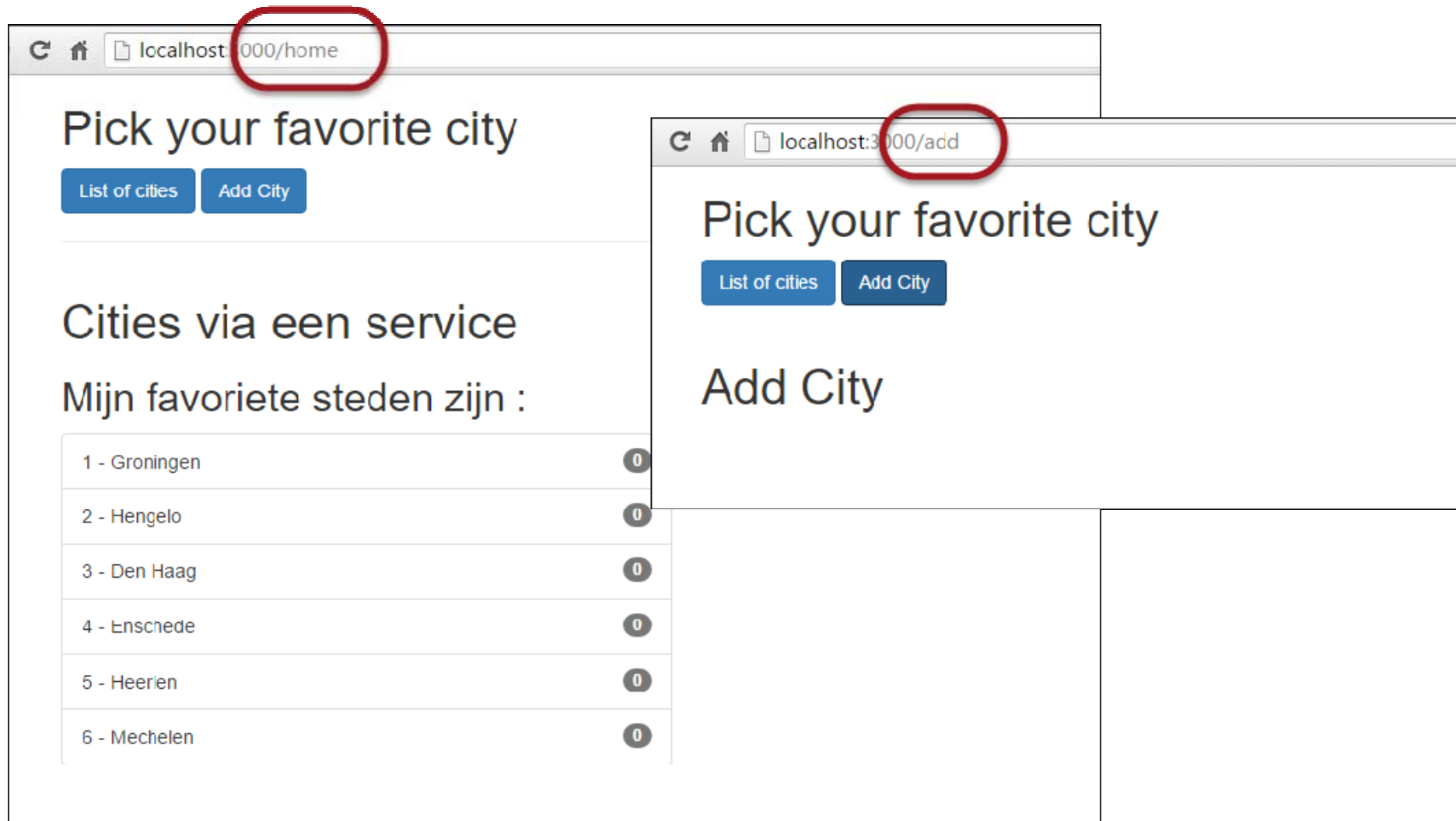
export class CityEditComponent {
  ...
}
```

```
// city.detail.component.ts
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'detail-city',
  template: `<h1>Detail City</h1> ...`
})

export class CityDetailComponent {
  ...
}
```

7. Run the application



Catch-all routes

```
6 export const AppRoutes: Routes = [  
7   {path: '', component: AppComponent},  
8   {path: 'home', component: AppComponent},  
9   {path: 'add', component: CityAddComponent},  
10  {  
11    // catch all route  
12    path: '**',  
13    redirectTo: 'home'  
14  },  
15 ];
```

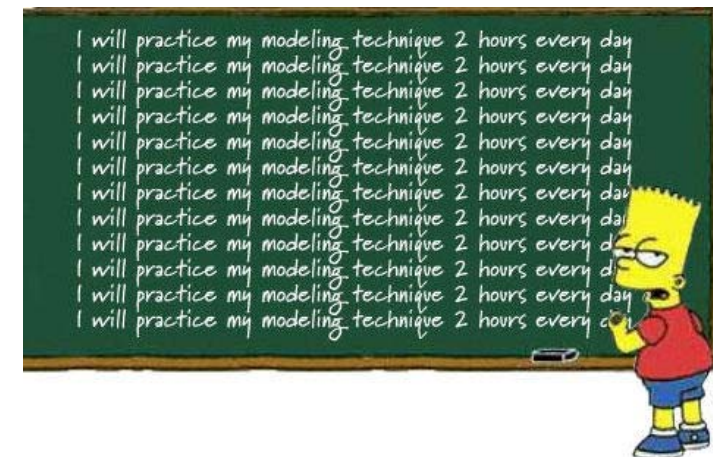
Use `**` as a catch-all route:

- `redirectTo`: route you want to show in address bar.
- The component is mentioned in the route that is pointed at.

Checkpoint

- Routes are created on module level (Angular 1: app level). Every Module can have it's own routes.
- Components can have child routes.
- Follow these steps. Remember to inject RouterModule, create `app.routes.ts` en `<base href="/">` and so on.
- Example: `/400-routing`
- Exercise: 7a). Optional: 7b)
- Official docs:

<https://angular.io/guide/router>





Route parameters

Master-Detail views and –applications

Dynamische routes maken

- Goal: Single detail page for customers, products, services, etc.
- Readable routes like: `/cities/5`, or `products/apple/iphone`, and so on
- Method:
 1. Edit `app.routes.ts` and hyperlinks on the page.
 2. Use `route:ActivatedRoute` in detail component
 3. Write hyperlinks like `<a [routerLink]=[...]>` with parameter

1. Edit app.routes.ts

```
// app.routes.ts
import {Routes} from '@angular/router';
import {AppComponent} from './app.component';
import {CityAddComponent} from './city.add.component';
import {CityDetailComponent} from './city.detail.component';

export const AppRoutes: Routes = [
  {path: '', component: AppComponent},
  {path: 'home', component: AppComponent},
  {path: 'add', component: CityAddComponent},
  {path: 'detail/:id', component: CityDetailComponent}
];
```



2. Create Detail Component

```
// city.detail.component.ts
...
// import {RouteParams} from "@angular/router"; // OLD way
import {ActivatedRoute} from '@angular/router';

@Component({
  selector: 'city-detail',
  template: `<h1>City Detail</h1>
    <h2>Details voor city: {{ id }}</h2>
  `
})

export class CityDetailComponent implements OnInit, OnDestroy {
  id: string;
  currentCity: City;

  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    this.route.params
      .subscribe((id: any) => {
        this.id = id;
      });
  }
}
```



ActivatedRoute

2a. DetailComponent - variants

Using router snapshots

```
// OR:  
// Work via Router-snapshot:  
// Sometimes we're not interested in future changes of a route parameter.  
// All we need the id and once we have it, we can provide the data we want to provide.  
// In this case, an Observable can bit a bit of an overkill.  
// A *snapshot* is simply a snapshot representation of the activated route.  
this.id = this.route.snapshot.params['id'];  
this.name = this.route.snapshot.params['name'];
```

2b. DetailComponent - variants

```
ngOnInit() {  
  // NEW:  
  this.sub = this.route.params  
    .subscribe((params: any) => {  
    this.id = params['id'];  
    this.name = params['name'];  
  });  
}
```

```
ngOnDestroy() {  
  // If subscribed, we must unsubscribe before Angular destroys the component.  
  // Failure to do so could create a memory leak.  
  this.sub.unsubscribe();  
}
```



.unsubscribe()

3. Add Detail component to Module

```
// app.module.ts
...
// Components
import {CityDetailComponent} from './city.detail.component';

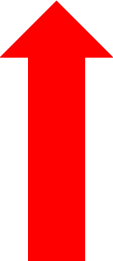
@NgModule({
  imports      : [
    ...
  ],
  declarations: [
    ...
    CityDetailComponent
  ],
  providers    : [CityService],
  bootstrap    : [MainComponent]
})
export class AppModule {
}
```



Component

Edit App Component ('Master View')

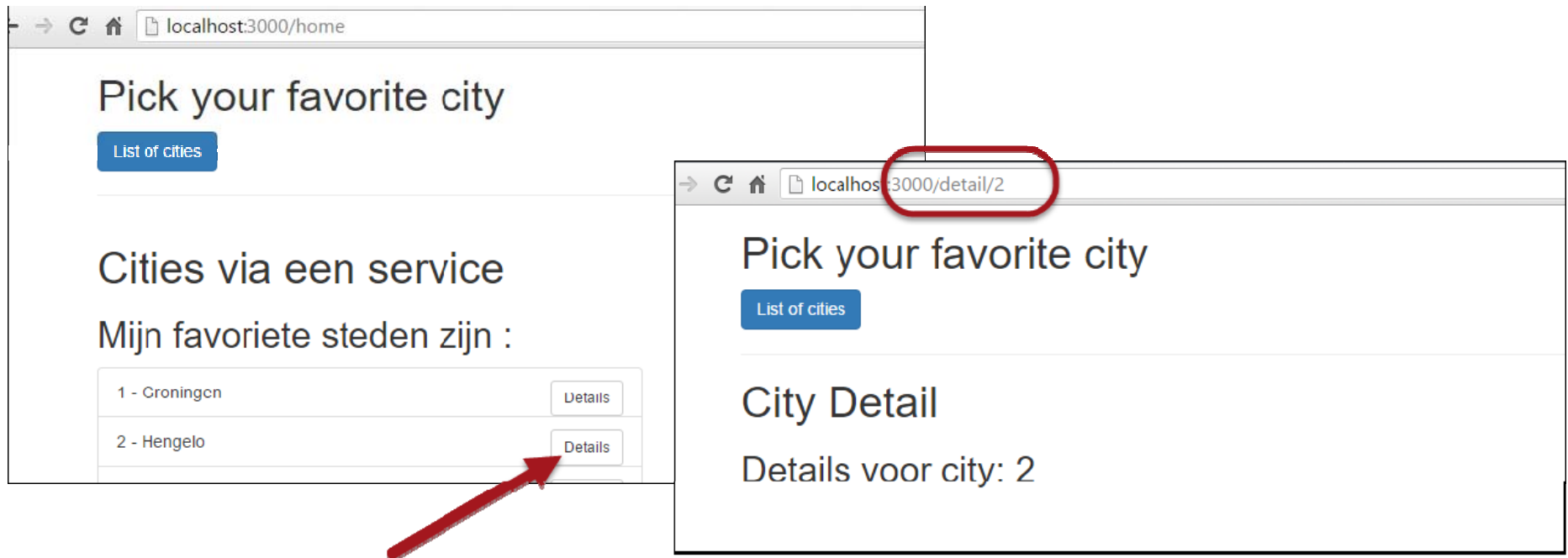
```
<li *ngFor="let city of cities" class="list-group-item">
  <a [routerLink]="['/detail', city.id]">
    {{ city.id }} - {{ city.name }}
  </a>
</li>
```



Remember that `[routerLink]` should now be calculated dynamically and thus should be written with `[...]` for attribute binding

Meegeven van parameters

- You pass an *array of parameters* to [routerLink]
- Parameters are matched on position. Not on name.
- Optional : extend service to return specific product/item



Optional parameters : [queryParams]

HTML

```
<a [routerLink]="['/detail', city.id, city.name]"  
  [queryParams]="{province:city.province, population:180000}">  
  {{ city.id }} - {{ city.name }}  
</a>
```

Class

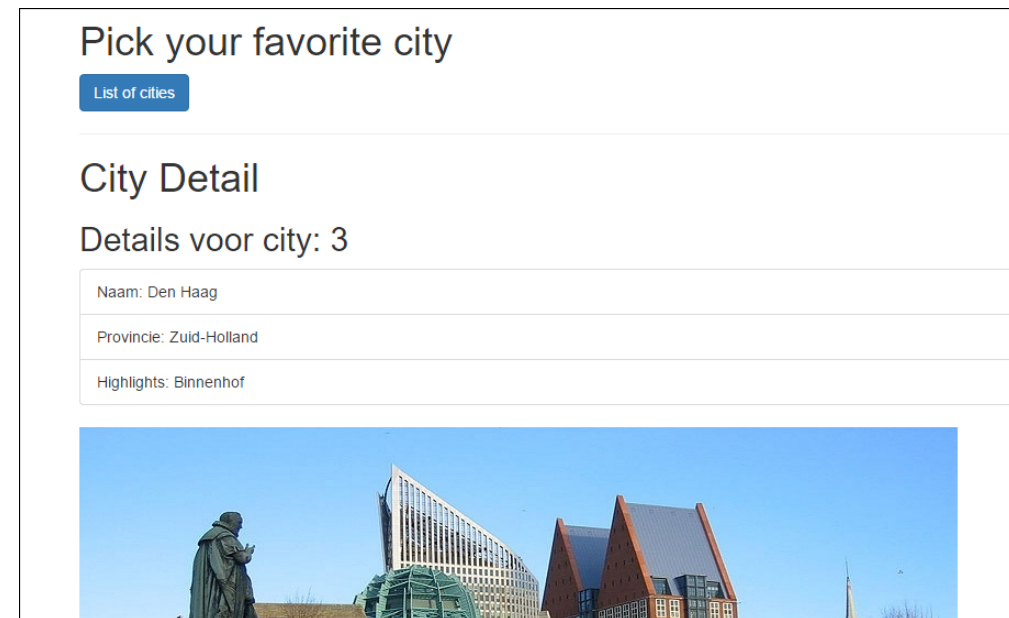
```
this.route.queryParams.subscribe((params: any) => {  
  this.province = params.province;  
})
```

Next up – details via Service

- Make sure to add a method like `.getCity(id)` that returns a city, based on id.

// NEW, with fetching details via Service:

```
this.sub = this.route.params
    .map(params => params['id'])
    .switchMap(id => this.cityService.getCity(id))
    .subscribe((city:City) => {
        this.currentCity = city;
    });
```



In city.service.ts:

- Edit/add a method to return a specific city

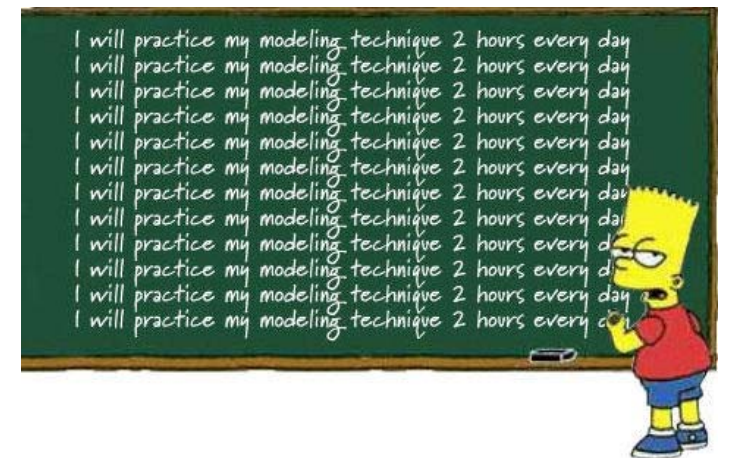
// return a city, based on Id

```
getCity(id: string): City[] {  
    return this.http.get<City>('app/cities.json').pipe(  
        map(cities => cities.filter((city: City) => {  
            return city.id === parseInt(id);  
        }))  
    )  
}
```

Checkpoint

- RouteParameters are set with `:parameterName` in `app.routes.ts`.
- Remember to inject `ActivatedRoute` in component.
- Use the property `.params` to retrieve the passed in values.
- Example: `\401-route-parameter`
- Exercise 7c)

Exercise....



More on routing

- Router Guards – Secure parts of your application, based on Auth-logic
- Child Routes
- Named Router Outlets
 - <http://onehungrymind.com/named-router-outlets-in-angular-2/>
- Router resolvers
 - <https://blog.thoughttram.io/angular/2016/10/10/resolving-route-data-in-angular-2.html>
- Lazy Loading – Split app in Modules and load *on demand*
 - <https://angular.io/guide/router#lazy-loading-route-configuration>



Route Guards

Secure parts of you application with Guards

Guard Types

- Four types of guards:
 - `CanActivate` – decides if a route can be activated
 - `CanActivateChild` – decides if children of a route can be activated
 - `CanDeactivate` – decides if a route can be deactivated
 - `CanLoad` – decides if a module can be loaded lazily

Defining Guards

- Multiple ways (as functions or as classes)
- Regardless, it needs to return a
 - `Observable<boolean>`,
 - `Promise<boolean>` or
 - `boolean`.
- Defined in `@NgModule`, or as a separate class

Guards as a function

- Define a token and a guard function. For example in `app.module.ts`.

```
// app.module.ts
```

```
...
```

```
@NgModule({
```

```
...
```

```
  providers : [
```

```
    CityService,
```

```
    {
```

```
      provide : 'CanAlwaysActivateGuard',
```

```
      useValue: () => {
```

```
        console.log("Route requested");
```

```
        return true; // do validation or other stuff here
```

```
      }
```

```
    }
```

```
  ],
```

```
  ...
```

```
})
```

```
export class AppModule {}
```



Token



Function

Use the guard token in app.routes

```
// app.routes.ts
```

```
...
```

```
export const AppRoutes: Routes = [
```

```
  ...
```

```
  {
```

```
    path: 'home',
```

```
    component: AppComponent,
```

```
    canActivate: ['CanAlwaysActivateGuard'] // Defined in app.module.ts
```

```
  },
```

```
  ...
```

```
];
```



(re)use of string
token

You *can* have multiple tokens/functions, guarding your route

Guards as a class

- Used: when the guard needs Dependency Injection
- Common use: with some kind of Authentication Service.
- All about Implementing interfaces!
 - `canActivate()`
 - `canActivateChild()`
 - `canDeactivate()`

canActivateViaAuthGuard.ts

```
// canActivateViaAuthGuard.ts
```

```
import { Injectable } from '@angular/core';  
import { CanActivate } from '@angular/router';  
import { AuthService } from '../auth.service';
```

```
@Injectable()
```

```
export class CanActivateViaAuthGuard implements CanActivate {
```

```
  constructor(private authService: AuthService) {}
```

```
  canActivate() {  
    return this.authService.isLoggedIn();  
  }
```

```
}
```

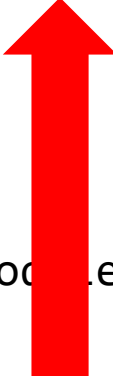
Class/Guard name

Auth Service

**Interface
implementation**

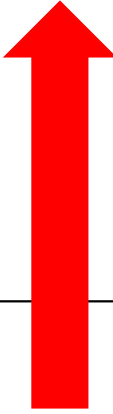
Register Guard class on module and routes

```
// app.module.ts
...
@NgModule({
  ...
  providers : [
    ...,
    AuthService,
    CanActivateViaAuthGuard
  ],
  ...
})
export class AppModule {
}
```



```
// app.routes.ts
...
import {CanActivateViaAuthGuard} from './canActivateViaAuthGuard';

export const AppRoutes: Routes = [
  ...
  {
    path      : 'add',
    component : CityAddComponent,
    canActivate: [CanActivateViaAuthGuard]
  },
  ...
];
```




Deactivating routes

- Called when navigating *away* from a route
- Same approach as CanActivate route

```
// canDeactivateGuard.ts
import {Injectable} from '@angular/core';
import {CanDeactivate} from '@angular/router';
import {CanDeactivateComponent} from "./canDeactivate.component";

@Injectable()
export class CanDeactivateGuard implements CanDeactivate<CanDeactivateComponent> {

  canDeactivate(target:CanDeactivateComponent) {
    // Can the user deactivate the route? Test for changes here!
    // For now, return Yes/Nope from the browser confirm dialog.
    if (target.hasChanges()) {
      return window.confirm('Do you really want to cancel? There might be unsaved changes');
    }
    return true;
  }
}
```



Add guard to routes

```
// app.routes.ts
```

```
...
```

```
import {CanDeactivateComponent} from "./canDeactivate.component";
```

```
import {CanDeactivateGuard} from "./canDeactivateGuard";
```

```
export const AppRoutes: Routes = [
```

```
...
```

```
{
```

```
  path      : 'deactivate',
```

```
  component : CanDeactivateComponent,
```

```
  canDeactivate: [CanDeactivateGuard]
```

```
},
```

```
...
```

```
];
```




Create DeactivateComponent

- Add implementation of `.hasChanges()`!

```
// ...
export class CanDeactivateComponent implements OnInit {
  // Properties voor de component/class
  myForm: FormGroup = new FormGroup({
    txtInput: new FormControl()
  });

  constructor(private route: Router) { }
  ngOnInit() {}

  moveAway() {
    this.route.navigate(['/home']);
  }
  hasChanges(){
    return this.myForm.dirty; // return state of the form
  }
}
```





More info

More background information on routing

More on routing

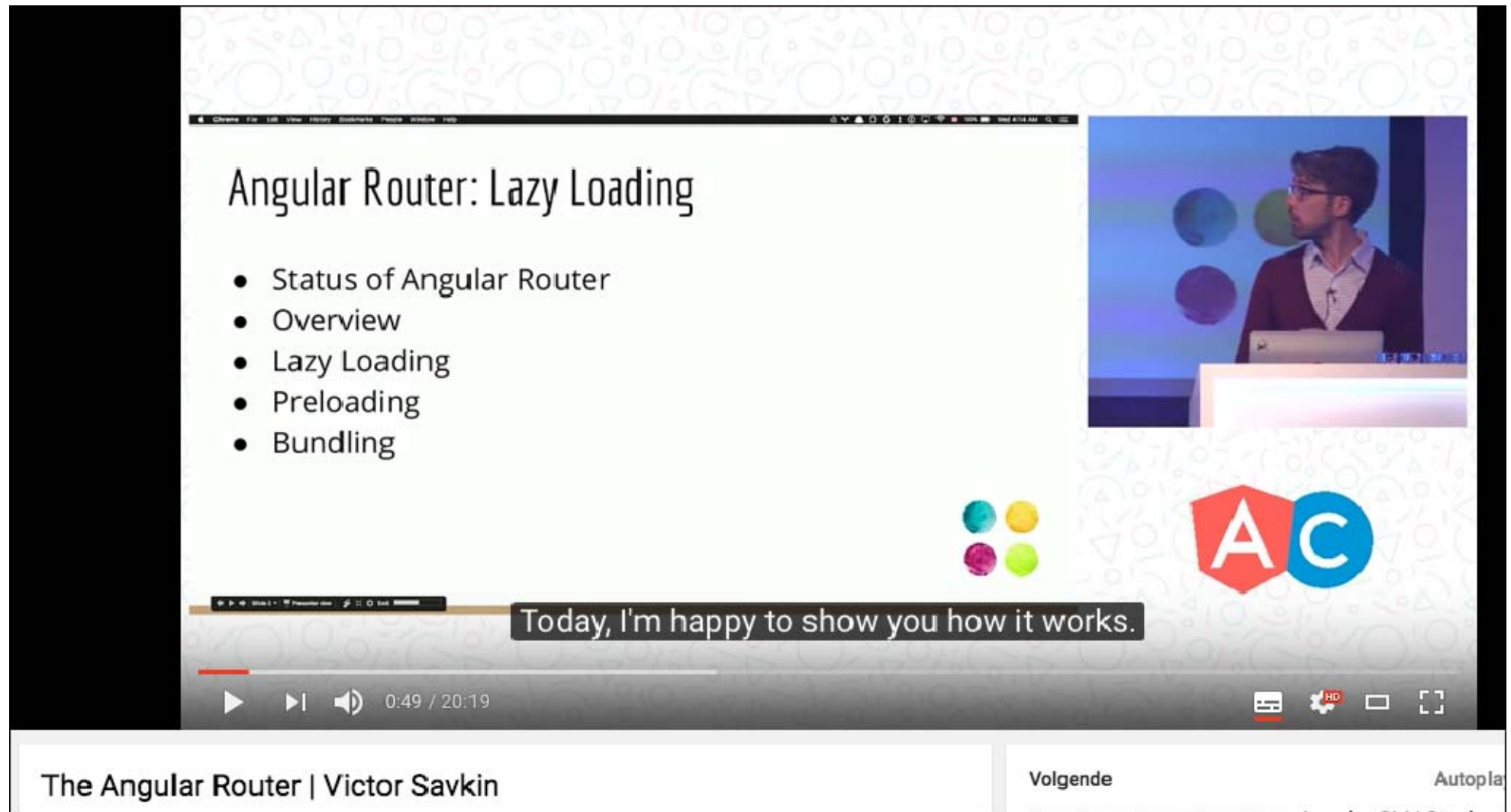
- <https://angular.io/docs/ts/latest/guide/router.html>
- <http://blog.thoughttram.io/angular/2016/06/14/routing-in-angular-2-revisited.html>
- <http://blog.thoughttram.io/angular/2016/07/18/guards-in-angular-2.html>
- <https://vsavkin.com/>
- https://angular-2-training-book.rangle.io/handout/routing/child_routes.html

Victor Savkin (=creator of the router)



<https://leanpub.com/router>

<https://www.youtube.com/watch?v=QLns6s02O48>



The video player shows a presentation slide titled "Angular Router: Lazy Loading". The slide lists the following topics:

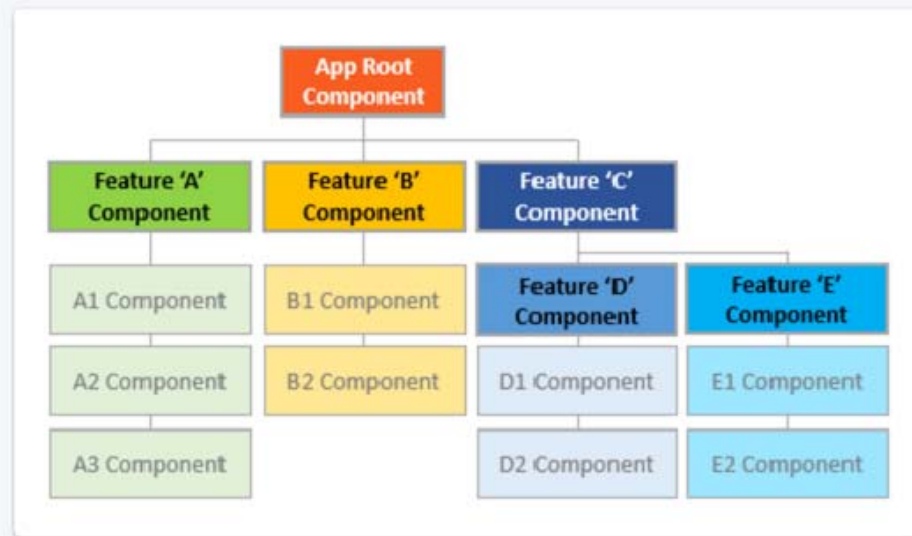
- Status of Angular Router
- Overview
- Lazy Loading
- Preloading
- Bundling

A small inset video in the top right corner shows the presenter, Victor Savkin, at a podium. The Angular logo (a red shield with a white 'A' and a blue circle with a white 'C') is visible in the bottom right of the slide. A subtitle at the bottom of the slide reads: "Today, I'm happy to show you how it works."

The video player interface includes a progress bar at 0:49 / 20:19, a play button, a volume icon, and a settings icon. The video title "The Angular Router | Victor Savkin" is displayed at the bottom left, and the word "Volgende" (Next) is at the bottom right.

Advanced routing

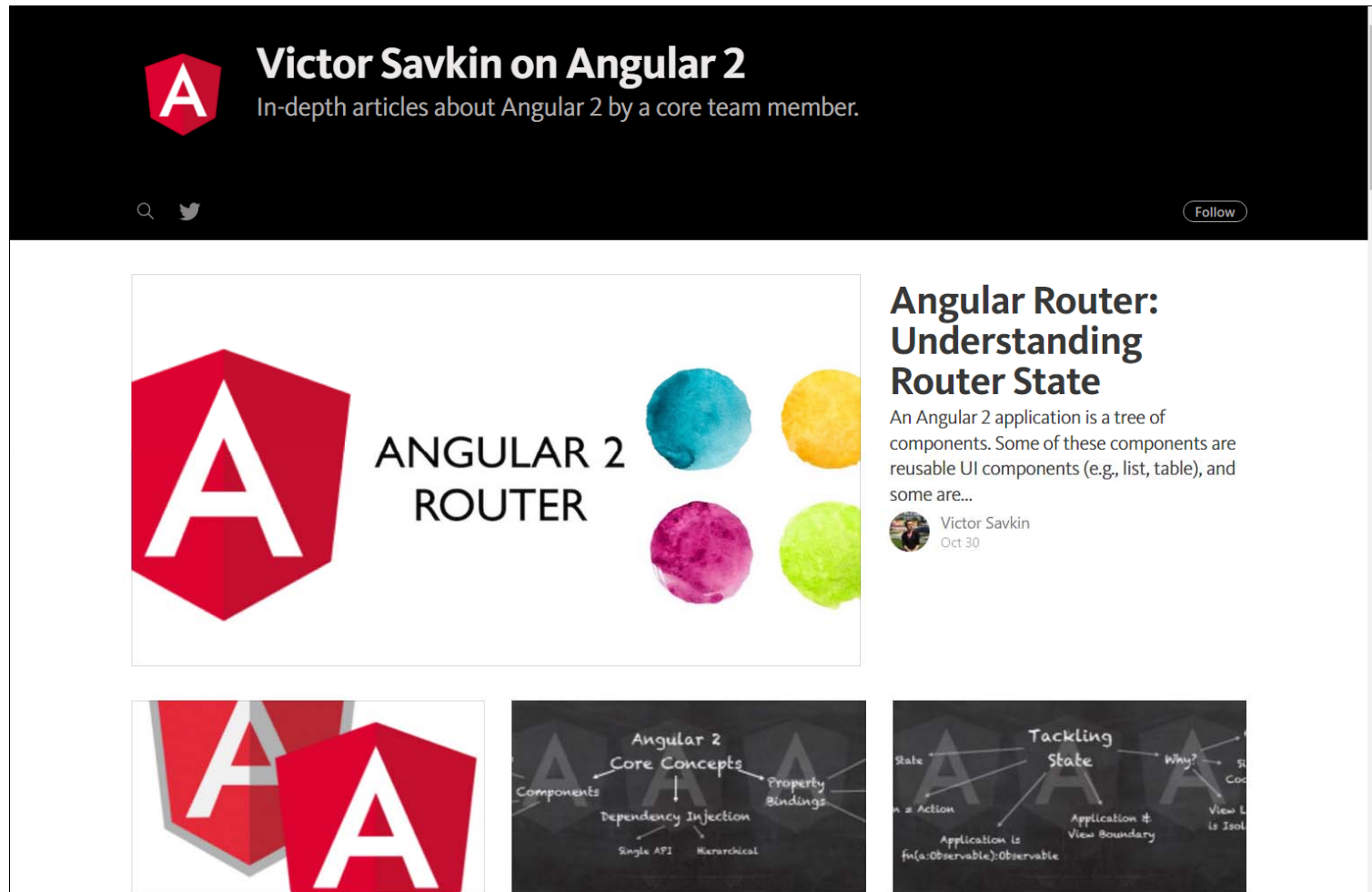
It's looking good as a general pattern for Angular applications.



- each feature area in its own module folder
- each area with its own root component
- each area root component with its own router-outlet and child routes
- area routes rarely (if ever) cross

<https://angular.io/docs/ts/latest/guide/router.html>

Victor Savkin on Routing



<https://vsavkin.com/>