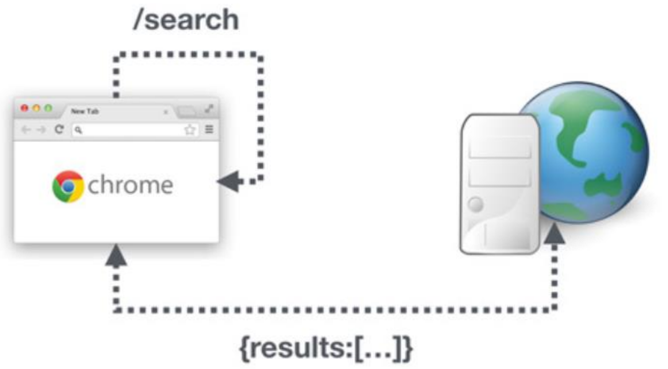
7 Angular Routing

Client-side routing, Router module, Navigation and Guards

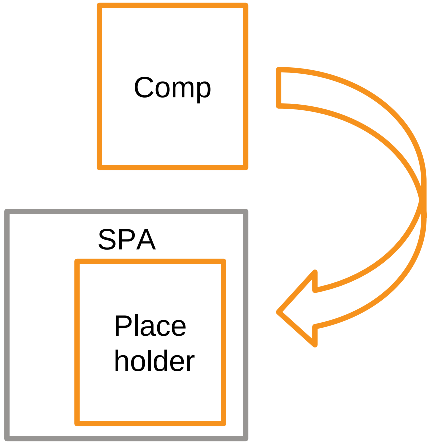
Reminder (SPA)

- A Single Page Application is a web application that fits on one web page
  - Much fluent user experience
  - Page never reloads
  - Many things are prefetched: html, css, js
  - Only data is loaded from server
  - Relies on client-side routing



Client side routing

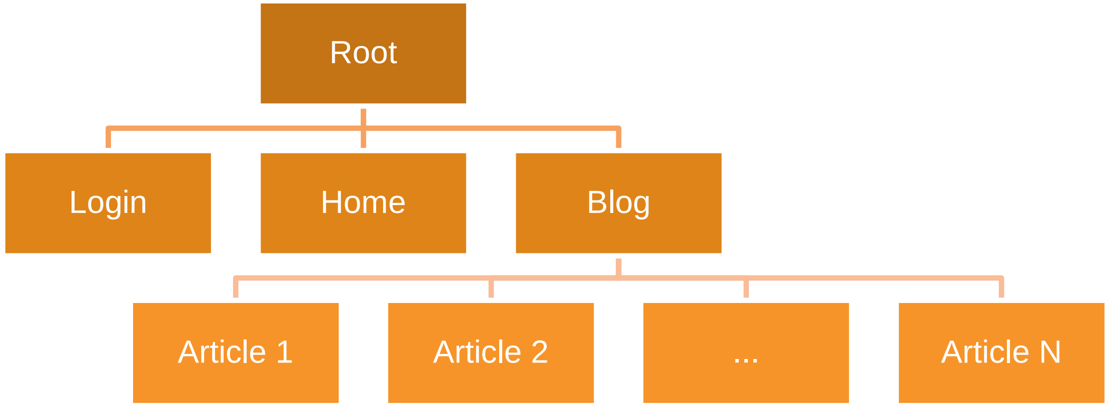
- In Angular a page is a component that is composed from smaller components
- Angular Router module
    - Match an **URL** with a **Component**
    - Manage the tree (and subtree) of components in the app
    - Inject components dynamically



Router configuration

1. **Design the routing tree state of your application**

Example:

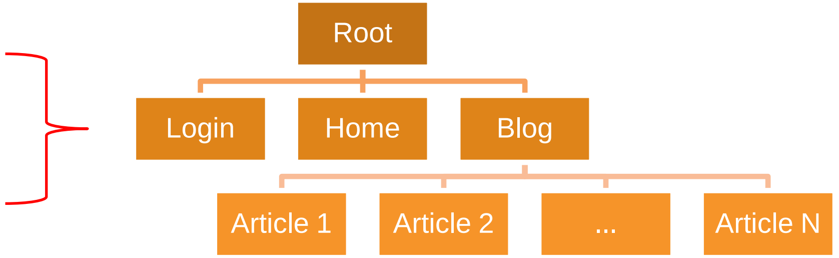


Router configuration

2. Create a definition for each route

- Use "**Route**" Interface
  - `path`: Path name, part of the URL
  - `component` : Angular component to load if the path match

```
routes = [{
    path: "login",
    component : LoginComponent
}]
```



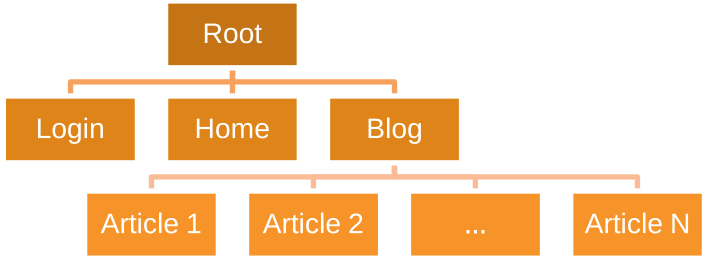
Router configuration

2. Create a definition for each route

- Configure dynamic routes

```
{
    path: "article1",
    component: Article1Component
},
{
    path: "article2",
    component: Article2Component
},

...
{
    path: "articleN",
    component: ArticleNComponent
}
```

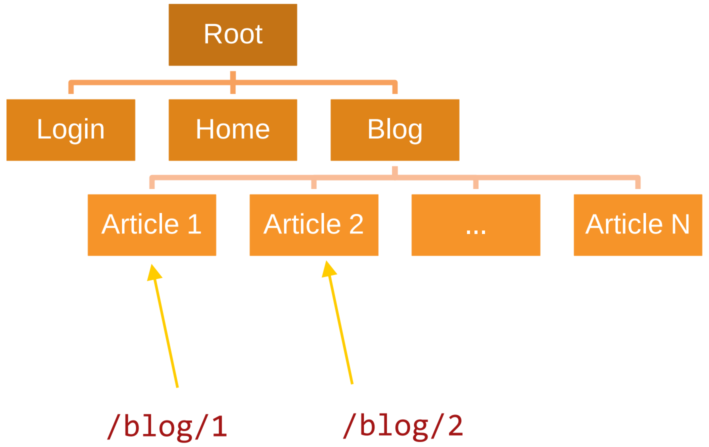


Router configuration

2. Create a definition for each route

- Define dynamic parameter using **:param**

```
{
   path: "blog/:id",
   component: ArticleComponent
},
```



Non-parameterized routes always takes priority over parameterized ones.

Router configuration

3. Default route

- Use an empty string (“”) to match the root
  - Redirect to the **default** route
  - pathMath: How to check the URL
    - `prefix`: matches if the url starts with the given path
    - `full`: matches if the url equal the given path

```
{ path: "", redirectTo : "login", pathMatch: "full"},
```

Router configuration

4. Wildcard (Catch-all route)

- Use the wildcard (**) to match anything
  - Used generally for 404 not found page

```
{ path: "**", redirectTo : "home"},
//OR
{ path: "**", component : NotFoundComponent},
```

The order of routes is important, wildcard route is always in the end.

Router configuration

5. Activating routes

- Import the router module into the app module
- Pass in the configured routes

```
const routes = [
{
    path: "", redirectTo:"home"
}
//...
]

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

Router configuration

6. Displaying route content

- Where to inject components when route changes?
- Use router-outlet element

```
//app.component.html
<div>
    <span><a href="/login">Login</a></span>
    <span><a href="/home">Home</a></span>
</div>
<router-outlet></router-outlet>
```

Navigation

Navigation (from HTML)

- Router module provides **routerLink** directive
- Used generally with *<a>* tag

```
<nav>
  <a routerLink="/home">Home</a>
  <a routerLink="/articles">Articles</a>
</nav>
```

Navigation

Navigation (from TS)

- Use **Router** service
- Navigate method accepts an array
  - First parameter: the path
  - Other parameters: route parameters

```
import {Router} from '@angular/router';

export class AppComponent {
  constructor(private router: Router) { }

  readArticle(articleId) {
    this.router.navigate(['/blog’, articleId]);
  }
}
```

Reading Route data

Use ActivatedRoute service to read params

- Provides us with all information about the current route
- Provides **paramMap** for path
- Provides **queryParamMap** for query string

```
this._route.snapshot.paramMap['id'];

this._route.snapshot.queryParamMap['search'];
```

- Read the parameters in the **ngOnInit** hook

Guards

- A Guard is an **interceptor** of the navigation
- Used to control the navigation
  - Securing access to pages
  - Unsaved data
  - Do something before navigating
- Returns
  - True: continues navigation
  - False: keeps use on the same view
- Can override the navigation
  - Redirect to another route

Guards

- **CanActivate**
  - Mediate navigation to a route
  - Eg. Is user allowed to navigate to this component ? If not send to /login route
- **CanDeactivate**
  - Is user allowed to navigate away from the page ?
  - Eg. Don’t allow navigation when unsaved changes
- **Resolve**
  - Fetch data before the route loads

CanActivate (Example)

```
@Injectable({
   providedIn: 'root'
})
export class AuthGuard implements CanActivate {

   constructor(private userService: UserService,private router:Router) {
   }

   canActivate(Route: ActivatedRouteSnapshot, state: RouterStateSnapshot)  {

      if (this.userService.isAuthenticated) {
         return true
      } else {
          this.router.navigate(["/login"])
          return false
      }

   }
}
```

CanActivate (Example)

```
{
    path: "login",
    component: LoginComponent
},
{
    path: "home",
    canActivate: [AuthGuard]
    component: HomeComponent
},

...
{
    path: "blog/:id",
    canActivate: [AuthGuard]
    component: ArticleComponent
}
```

Feature areas

- Sepration of concerns
  - Splitting into multiple components
  - Splitting routes configuration

- Each feature area has its own
  - Folder
  - Root component
  - Router configuration

```
app
├── app.component.ts
├── app.module.ts
├── app.routes.ts
├── games
│   ├── game.module.ts
│   ├── game-detail.component.ts
│   ├── games.service.ts
│   ├── game-list.component.ts
│   └── game.routes.ts
└── tournaments
    ├── tournament.module.ts
    ├── tournament-detail.component.ts
    └── tournament.routes.ts
```

Feature areas

Use RouterModule.forRoot() once (in AppModule)

- Make router services available for entire app
- Makes router components available in AppModule

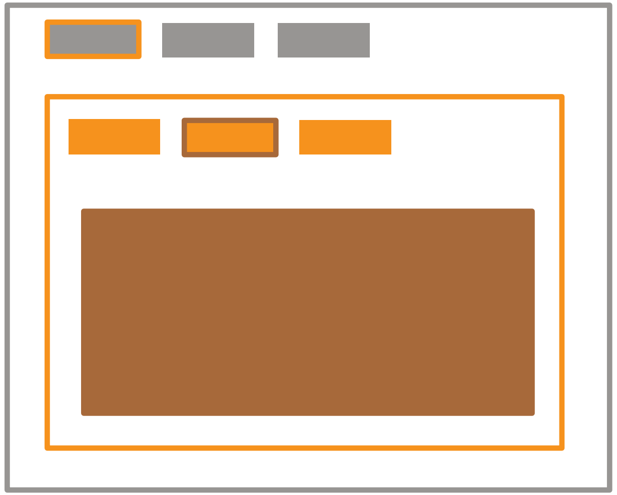
Use RouterModule.forChild() in each module

- Makes router components available in FA
- Does NOT expose router services again

Nested navigation

Multiple levels of navigation

- Uses the router-outlet element to render the nested view
- Leads to nested router-outlets
- Leads to nested route configuration
- Still one URL determines the content of all router-outlets



https://localhost:4200/settings/profile

Nested navigation

Child routes

- Routing happens per feature area
- No need to know about ancestors
- Use **children** property

```
const settingsRoutes: Routes = [{
  path: 'settings',
  component: SettingsComponents,
  children: [
    { path: '',component: GeneralSettingsComponent },
    { path: 'profile',   component: ProfileSettingsComponent     }
  ]
}];
```

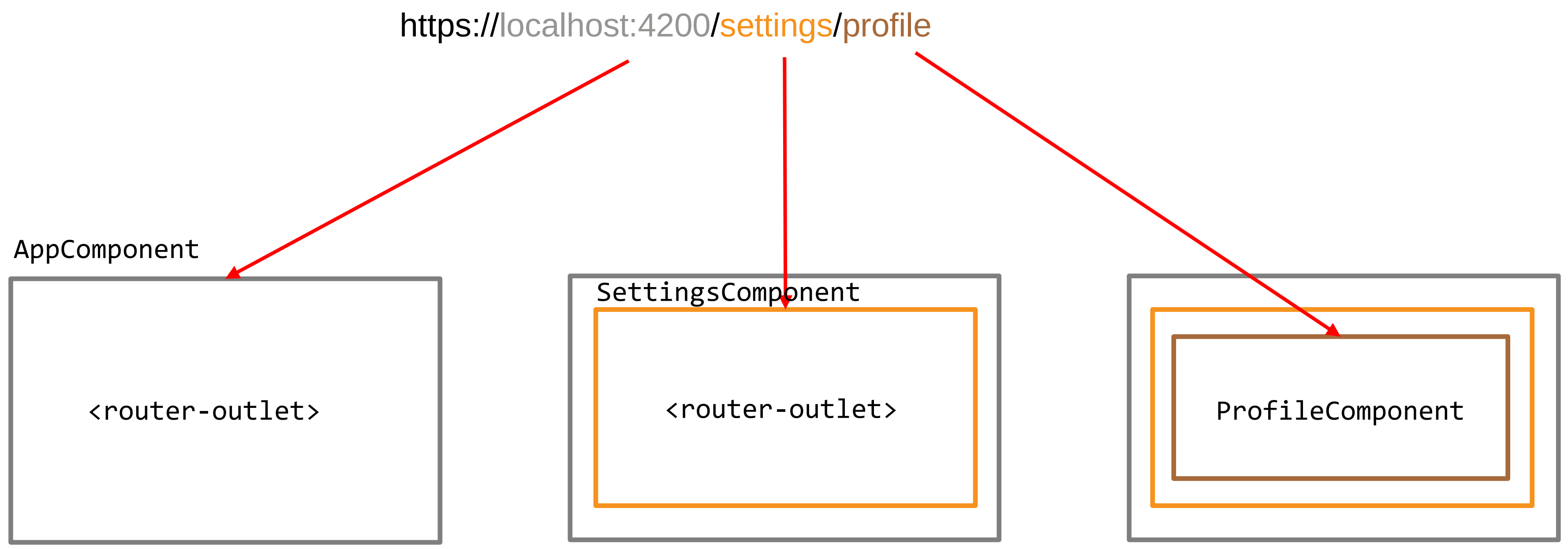
Nested navigation

Feature area root component

- Usually one per feature area
- Uses router-outlet to render the nested view

```
@Component({
    template: `
    <h2>Settings</h2>
    <router-outlet></router-outlet>
  `
})
export class SettingsComponent { }
```

Nested navigation (Example)



LAB 7

Use Angular Router Module