Workshop

Angular Routing

Routing

No Single Page Application without routing

Why routing

- A website consists of multiple pages for displaying various content.
- Angular applications are single-page applications (SPAs), meaning they typically have only one HTML page.
- Routing is the mechanism used to navigate within an Angular app without physically leaving or reloading the page.

Basic routing

Basic Routing

- Based on browser location and history
- Map of url to content
- Special package: @angular/router

Basic Routing

- Define routes per feature
- An extra file for route configuration:

feature-name.routes.ts

Basic Routing

<code>

Define routes

```
// app.routes.ts
import { Routes }  from '@angular/router'

export const routes: Routes = [{ /** **/ }];
```

Basic Routing

<code>

Register routes

```
// app.config.ts
import { provideRouter } from '@angular/router';
import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [provideHttpClient(), provideRouter(routes)]
};
```

Basic Routing

<code>

Defining a route - without leading ‘/’!

```
export const appRoutes: Routes = [{
  path: 'books',
  component: BookComponent
}];
```

Default routes

Routing wildcard

Set a wildcard to handle all not defined routes

```
{
  path: '**',
  component: PageNotFoundComponent
}
```

Routing Redirection

Redirect to default router

```
export const appRoutes: Routes = [{
  path: '',
  redirectTo: '/books',
  pathMatch: 'full' // checks if full url matches path!
}, {
  ...
}];
```

Displaying routes

Basic Routing

- No connection between DOM and route, yet
- Router needs to know where he should append the component
- Special component: RouterOutlet

Basic Routing

<code>

Routing components are available through the routing import

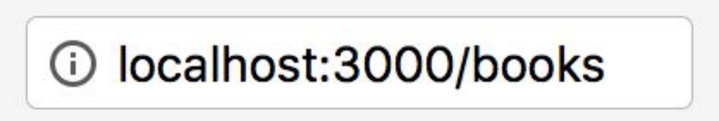
```
// app.component.ts
@Component({
  //...
  imports: [RouterOutlet],
  //...
})
export class AppComponent {}

// app.component.html
<router-outlet></router-outlet>
```

Basic Routing

1. Url in browser matches against route path
2. Information of connected route are evaluated
3. Information are used to show correct component in routerOutlet

Basic Routing



localhost:3000/books

```
Routes = [{ path: 'books', component: BookComponent }, …];
```

```
<router-outlet></router-outlet>
<app-book>...</app-book>
```

routerLink

RouterLink example

```
<a routerLink="/books">
```

+

```
{ path: 'books', component: BookComponent }
```

=

```
<a href="/books">
```

RouterLink import

<code>

```
// app.component.ts
@Component({
  //...
  imports: [RouterLink],
  //...
})
export class AppComponent {}
```

Task

Add basic routing

Routing with parameters

Routing with parameters

- You need dynamic routes very often, e.g. Detail Views
- Content of a component is configurable
- You need additional data in your component

Routing with parameters

<code>

Add parameter placeholders with a leading “:”

```
// app.routes.ts
const routes: Routes = [
  { path: 'books/detail/:isbn', component: BookDetailComponent }
];
```

routerLink with params

RouterLink with params example

```
<a [routerLink]=" ['/books', '/detail', 1] ">
```

+

```
{ path: 'books/detail/:isbn', component: BookDetailComponent }
```

=

```
<a href="/books/detail/1">
```

Retrieve route params in a component class

Route params

<code>

Inject `ActivatedRoute` service and subscribe params observable.

```
@Component(…)
export class BookDetailComponent implements OnInit {
  private readonly route = inject(ActivatedRoute)

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

Why an Observable?

Route params

- Angular has some caching mechanisms
- Current component and components on the same level in the tree are cached for faster navigation
- Components are not instantiated again
- But parameters could have changed, e.g. paging

Simple approach with snapshots

Route params - Snapshots

- Snapshots are images of the current state
- **ActivatedRoute** gives access to the current router state
- Can be used if not future changes expected

Route params - Snapshots

The params of a route are stored in a snapshot object.

```
@Component(...)
export class BookDetailComponent implements OnInit {
  private readonly route = inject(ActivatedRoute)


  ngOnInit () {
    const bookIsbn = this.route.snapshot.paramMap.get('isbn');
  }
}
```

Navigate with Router Injectable

Router Service

<code>

Trigger navigation from Component Class

```
@Component({ /* ... */})
class BookComponent {
  private readonly router = inject(Router)
  private readonly bookApi = inject(BookApiService)

  goToBookDetails(book: Book) {
    this.router.navigate(['books', 'detail', book.isbn]);
  }
 }
```

Task

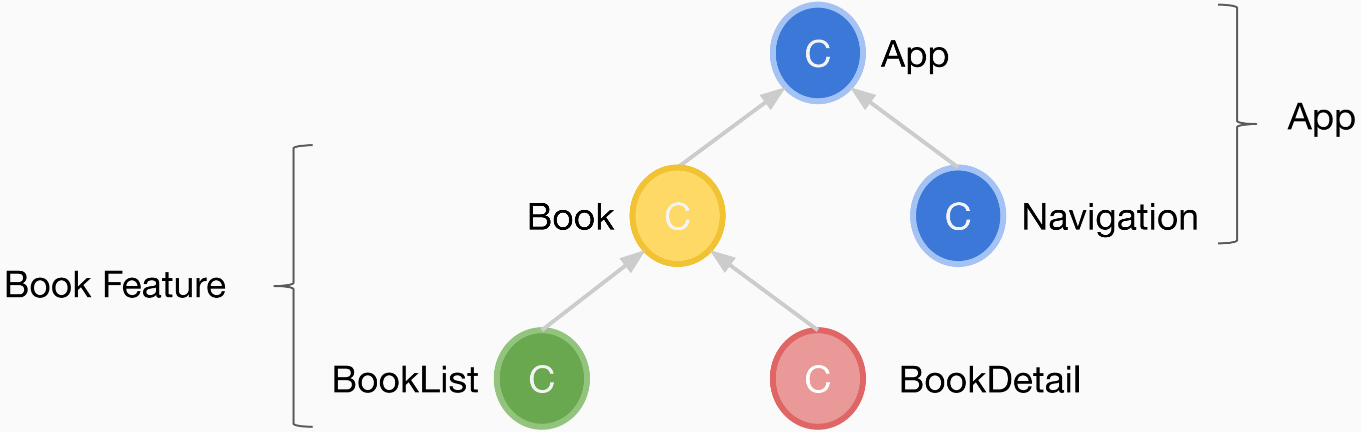
Add BookDetail Route

Nested & child routes

Nested routes

- An app / feature can have sub features with own components
- Each (sub) feature can manage its own routes
- No need to change root routing

Think in Features & Components



workshops.de

Nested routes

<code>

Routes of a book feature with a root book component

```
// book.routes.ts
export const bookRoutes: Routes = [
  {
    path: 'books',
    component: BookComponent
  }
];
```

Child routes

<code>

HTML with child route - book.component.html has its own routerOutlet

```
<app-root>
  <router-outlet></router-outlet>
  <book>
    <router-outlet></router-outlet>
    ...
  </book>
</app-root>
```

Child routes

- A route can have children
- Each child gets its parent path as base path
- Child route will be displayed in the RouterOutlet of its parent

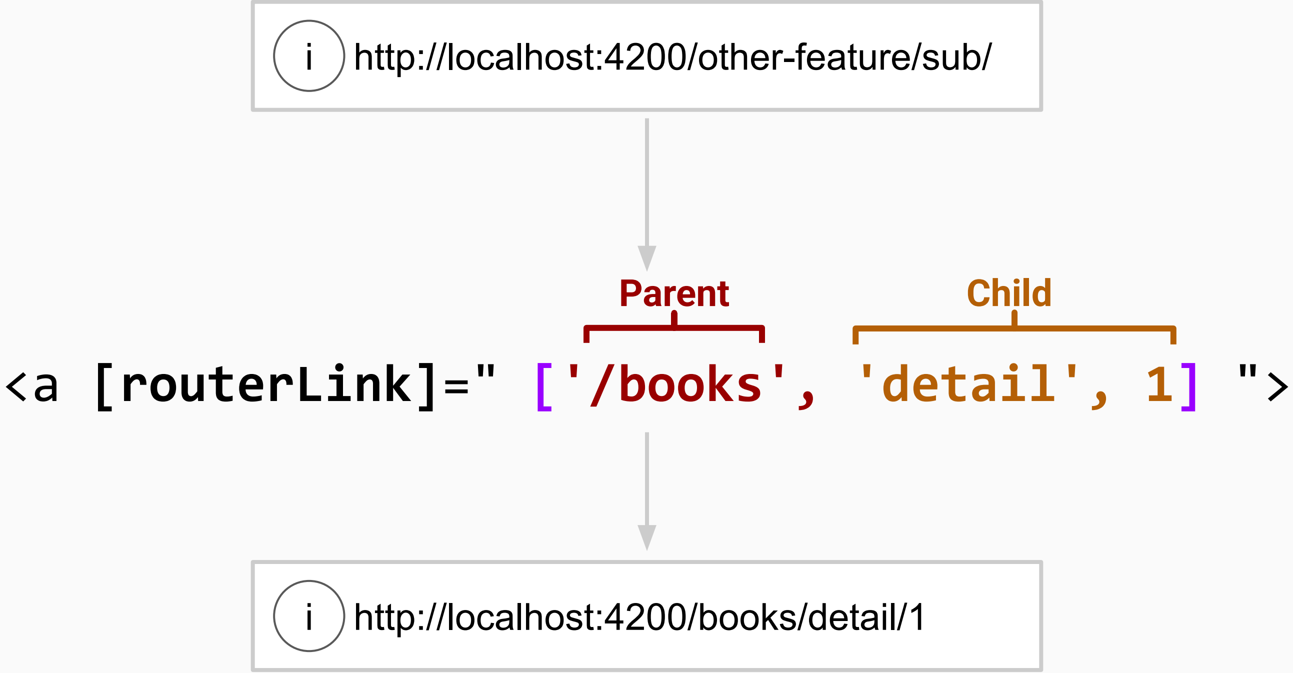
Child routes

<code>

BookList and BookDetail as route children under its parent Book

```
// book.routes.ts
{
   path: 'books',
   component: BookComponent,
   children: [
     {
       path: '', component: BookListComponent
     },
     {
       path: 'detail/:isbn', component: BookDetailComponent
     }
   ]
```

Routes - absolute links

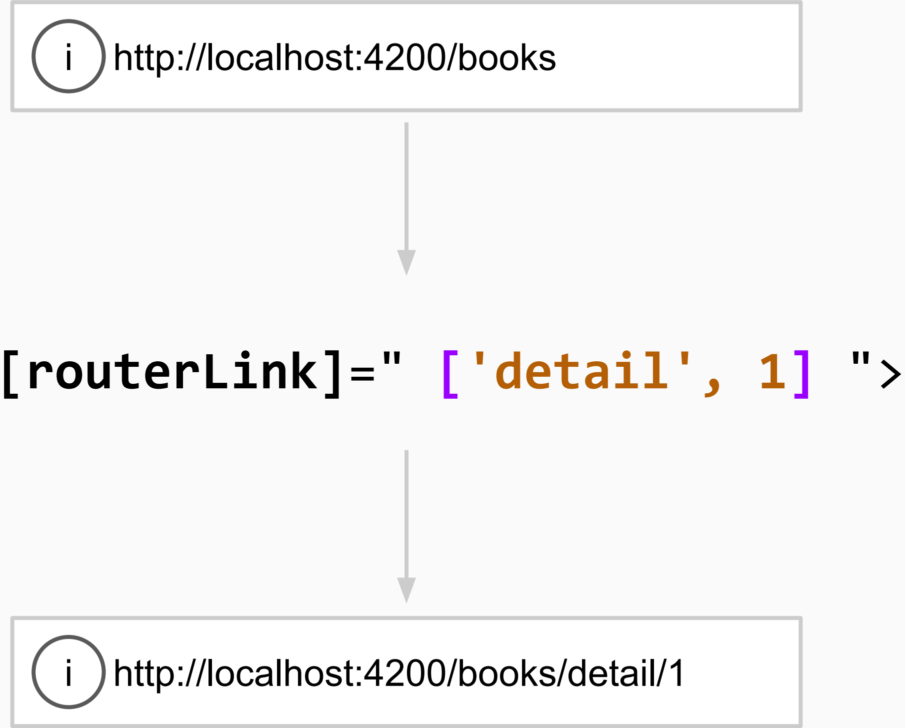


i
http://localhost:4200/other-feature/sub/

```
<a [routerLink]=" ['/books', 'detail', 1] ">
```

i
http://localhost:4200/books/detail/1

Routes - relative links



i http://localhost:4200/books

```
<a [routerLink]=" ['detail', 1] ">
```

i http://localhost:4200/books/detail/1

Lazy Loading

Load Routes and Features only if they are needed

Lazy Loading

- You do not want to load everything at once
- Split up your app in smaller parts → load them when needed
- Smaller initial bundle size → faster initial loading
- Routes with complex code or many dependencies but mostly not opened → add lazy loading

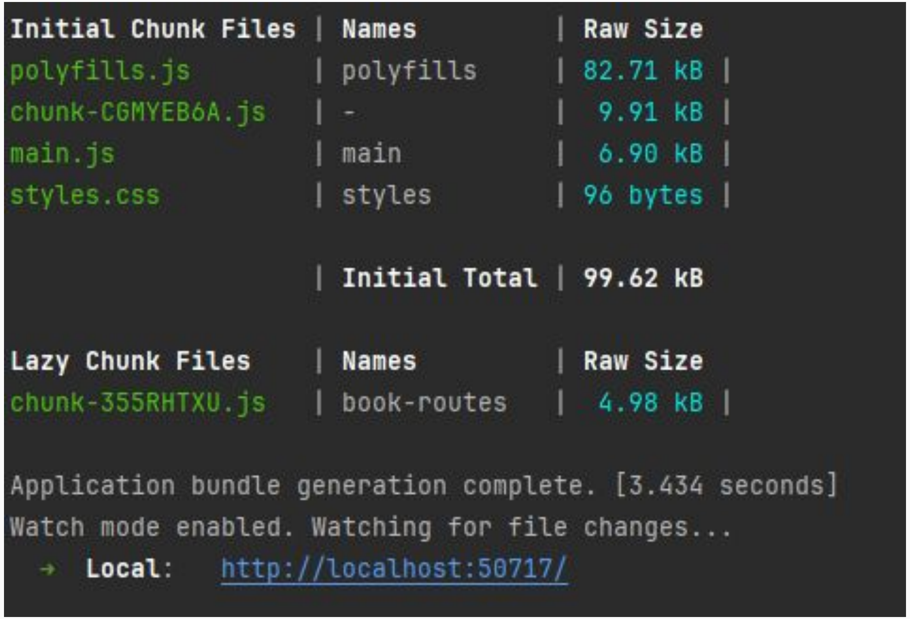
Lazy Loading

Request Books feature if needed - app.routes.ts

```
export const routes: Routes = [
  // ...
  {
    path: 'books',
    loadChildren: () => import('./book/book.routes')
                          .then(mod => mod.bookRoutes)
  }
];
```

Book components imports in other app.*.ts are NOT needed anymore!

Lazy Loading Compiler



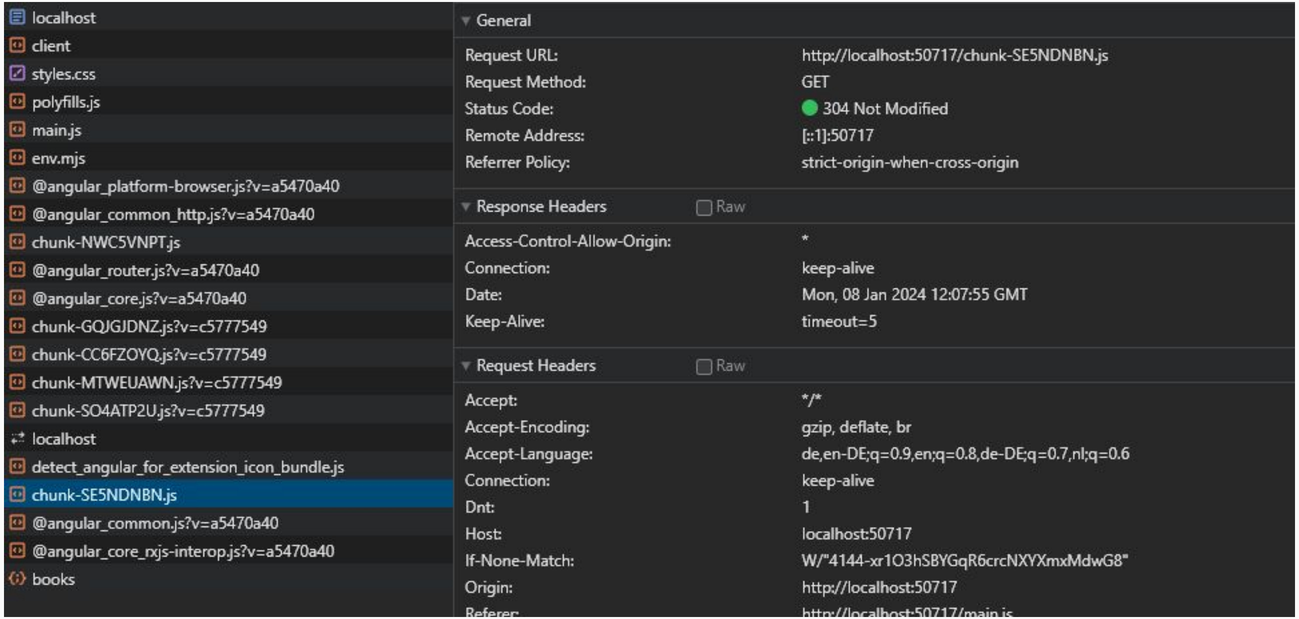
```
Initial Chunk Files | Names         | Raw Size
polyfills.js        | polyfills     | 82.71 kB |
chunk-CGMYEB6A.js   | -             |  9.91 kB |
main.js             | main          |  6.90 kB |
styles.css          | styles        | 96 bytes |

                    | Initial Total | 99.62 kB

Lazy Chunk Files    | Names         | Raw Size
chunk-355RHTXU.js   | book-routes   |  4.98 kB |

Application bundle generation complete. [3.434 seconds]
Watch mode enabled. Watching for file changes...
  ➜  Local:   http://localhost:50717/
```

Lazy Loading Browser



- localhost
- client
- styles.css
- polyfills.js
- main.js
- env.mjs
- @angular_platform-browser.js?v=a5470a40
- @angular_common_http.js?v=a5470a40
- chunk-NWC5VNPT.js
- @angular_router.js?v=a5470a40
- @angular_core.js?v=a5470a40
- chunk-GQJGJDNZ.js?v=c5777549
- chunk-CC6FZOYQ.js?v=c5777549
- chunk-MTWEUAWN.js?v=c5777549
- chunk-SO4ATP2U.js?v=c5777549
- localhost
- detect_angular_for_extension_icon_bundle.js
- chunk-SE5NDNBN.js
- @angular_common.js?v=a5470a40
- @angular_core_rxjs-interop.js?v=a5470a40
- books

▼ General

Request URL:	http://localhost:50717/chunk-SE5NDNBN.js
Request Method:	GET
Status Code:	● 304 Not Modified
Remote Address:	[::1]:50717
Referrer Policy:	strict-origin-when-cross-origin

▼ Response Headers □ Raw

Access-Control-Allow-Origin:	*
Connection:	keep-alive
Date:	Mon, 08 Jan 2024 12:07:55 GMT
Keep-Alive:	timeout=5

▼ Request Headers □ Raw

Accept:	*/*
Accept-Encoding:	gzip, deflate, br
Accept-Language:	de,en-DE;q=0.9,en;q=0.8,de-DE;q=0.7,nl;q=0.6
Connection:	keep-alive
Dnt:	1
Host:	localhost:50717
If-None-Match:	W/"4144-xr1O3hSBYGqR6crcNXYXmxMdwG8"
Origin:	http://localhost:50717
Referer:	http://localhost:50717/main.js

http://localhost:4200/books

`chunk-*.js` is loaded

Task

Use Lazy Loading for Book feature

Route Guards

Why guards?

Why guards?

- You want to protect your routes against unwanted access
- Sometimes you may have restricted permissions
  - User have to be signed in to see the content
- Protect the user
  - Notify him about unsaved changes, before leaving the route

Route Guards

- Angular defines Function Types
- Guards have to return boolean, URLTree or RedirectCommand
- Guards functions can return static values or async values (Promise or Observables)
- Possibility to have asynchronous guard functions, e.g. authorization check with an API

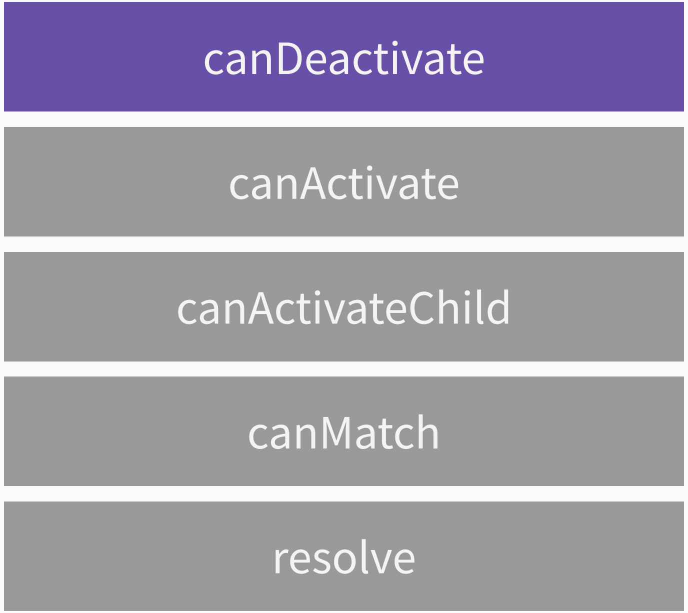
Route Guards

- 5 kinds of route guards
- Implement a function and use it on multiple guards
- Guards are route based and not component based

Route Guards

- canDeactivate
- canActivate
- canActivateChild
- canMatch
- resolve

Route Guards



canDeactivate

canActivate

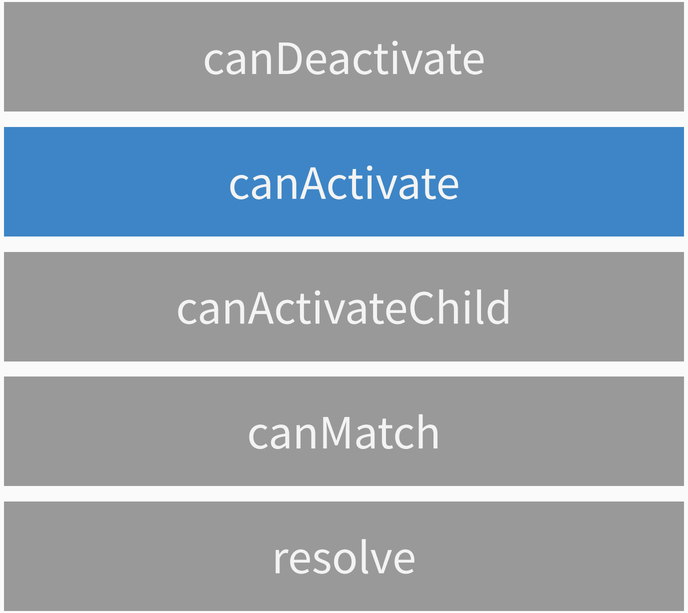
canActivateChild

canMatch

resolve

- Is it permissible for users to exit a route?
- Verify if the data has been successfully saved.
- Receive notifications when leaving the route.

Route Guards



canDeactivate

canActivate

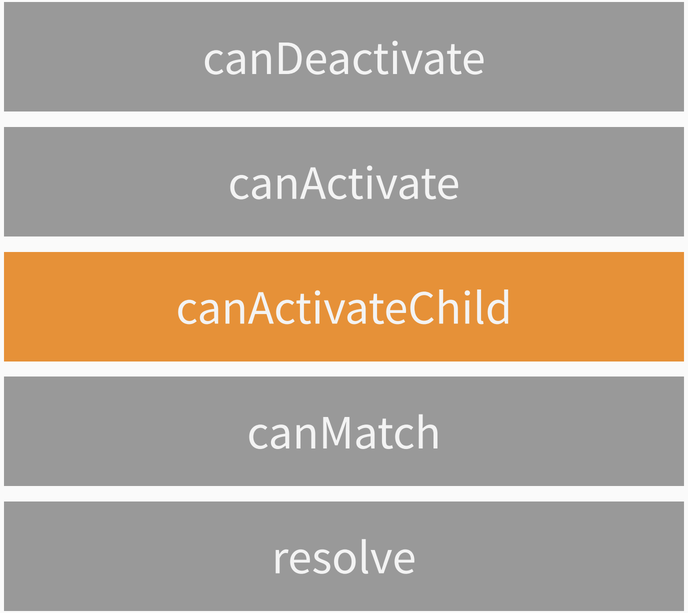
canActivateChild

canMatch

resolve

- Verify whether the route can be activated.
- Confirm the user's authentication status.
- Validate the user's access rights.

Route Guards



canDeactivate

canActivate

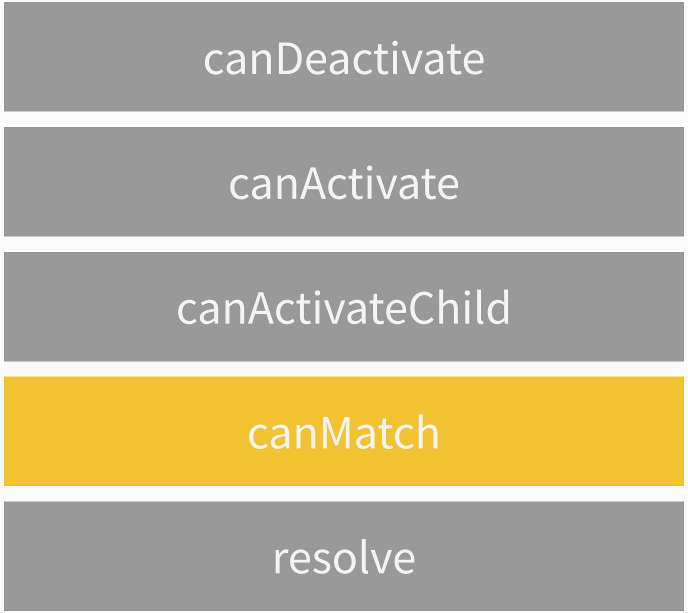
canActivateChild

canMatch

resolve

- A route may have subordinate routes.
- Verify whether subordinate routes can be activated.
- If all child routes share the same "canActivate" function, you can implement a single check for all of them.

Route Guards



canDeactivate

canActivate

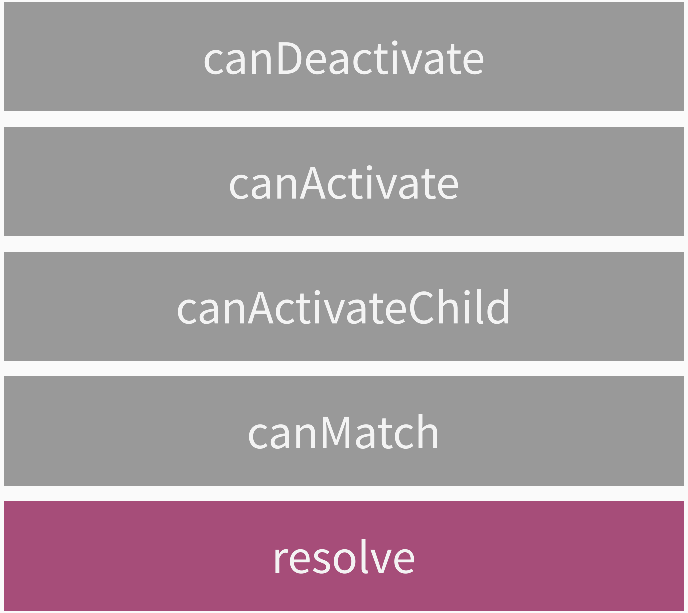
canActivateChild

canMatch

resolve

- Avoid to match the current path
- Avoid to load the content of the route (lazy loading)
- Multiple declarations of same path are possible
- Route recognition will not be aborted

Route Guards



canDeactivate

canActivate

canActivateChild

canMatch

resolve

- Fetch data prior to component loading.
- Able to handle any return value, such as Observables.

Guards as functions

Guards as functions

- Return types are exported by @angular/router
- Type names:
  - canActivate guard → CanActivateFn type
  - canDeactivate guard → CanDeactivateFn type
  - …

Guards as functions

- One guard have a generic option
  - You might want access to the component, their information and current state
  - `function confirmLeaveGuard:CanDeactivateFn<BookDetailComponent>`
- All others not:
  - `function hasAccessGuard: CanActivateFn`

Guards as functions

Simple guard function

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

export const hasAccessGuard: CanActivateFn =
  (route: ActivatedRouteSnapshot, state: RouterStateSnapshot) => {
    return true;
  };
```

Guards as functions

<code>

Connect a guard with a route

```
{
  path: 'books',
  component: BookComponent,
  canActivate: [hasAccessGuard]
}
```

Guards as classes (deprecated)

- An Angular Service
- A class that implements the guard interfaces

Guards as classes (deprecated)

<code>

Simple guard service

```
@Injectable({
  providedIn: 'root'
})
export class CanActivateViaServiceGuard implements CanActivate {
  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    return true;
  }
}
```

Task

Build a simple canDeactivate guard

Stateful guard functions

Stateful Route Guards

- Functions can use existing services to be stateful
- **inject** function can be used in this context

Stateful Route Guards

inject service

```
import { inject } from '@angular/core';
import { ServiceA } from './service-a';

export const hasAccessGuard: CanActivateFn =
  (route: ActivatedRouteSnapshot, state: RouterStateSnapshot) => {
    const service = inject(ServiceA);
    // ...
```

Task

Build a guard with state

Automatic Parameter Binding

Automatic Component Input Binding

- Router can bind inputs from parameters / data automatically
  - Additional feature of the router itself
- **input()** binding name must match path parameter / data name

Router feature ComponentInputBinding

<code>

Activate feature

```
// app.config.ts
provideRouter(routes, withComponentInputBinding())
```

Router feature ComponentInputBinding

Use feature

```
{
  path: 'detail/:isbn',
  component: BookDetailComponent,
}


export class BookDetailComponent {

  isbn = input.required<string>()
```

Task

Use ComponentInputBinding

symetics