Routing

Single Page Applications (SPAs)

A web application usually consists of more than one view:

- A list of cars
- Details of a single car
- Edit car information

For a SPA, this means that either...

- All views must be on that one HTML page
- Views must be loaded dynamically through AJAX

Angular routes

Angular supports routing by loading views in dynamically through AJAX. To implement routing:

1. Define routes based on the `Route` interface

```
export interface Route {
    path?: string;
    component?: Type<any>;
    ...
}
```

2. Call `RouterModule.forRoot([...])` to supply all the routes
3. Place an `<router-outlet></router-outlet>` in the HTML of your root component. This is the element where the different views are loaded beside.

Angular routes

Provide routes during bootstrap:

```
import { Route } from '@angular/router';
import { CarsPage } from './cars.page';
import { CarDetailsPage } from './car-details.page';

let routes: Route[] = [
    { path: 'cars', component: CarsPage },
    { path: 'cars/:id', component: CarDetailsPage }
]; // you can split routes up in different files as well

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

Accessing route information

Inject `ActivatedRoute`:

```
import { Component } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Car } from './car';
import { CarService } from './car.service';

@Component({
    templateUrl: 'car-details.page.html'
})
export class CarDetailsPage {
    car: Car;

    constructor(
        private carService: CarService,
        private route: ActivatedRoute) {
        this.route.params
            .pipe(
                map(params => +params['id']),
                switchMap(id => this.carService.getContact(id))
            )
            .subscribe(car => this.car = car);
        }
```

Define where the views are shown

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

@Component({
    selector: 'my-app',
    templateUrl: 'my-app.component.html'
})
export class AppComponent {}
```

app/my-app.component.html

```
<h1>Welcome to the car app!</h1>
<router-outlet></router-outlet>
```

Link to another route

You can use the `routerLink` directive to link to another route

```
<a routerLink="/cars">Back to the cars overview</a>
```

We can also use an Angular expression

```
<a [routerLink]="['/cars', 14]">View details of this amazing car</a>
```

Accessing route parameters

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { map } from 'rxjs/operators';

@Component({
    selector: 'car-details',
    templateUrl: 'car-details.page.html'
})
export class CarDetailsPage implements OnInit {
    constructor(private route: ActivatedRoute) { }

    ngOnInit() {
        this.route.params
            .pipe(map(params => params['id']))
            .subscribe(id => console.log('Retrieve some car data:', id));
    }
}
```

Or, if you're not interested in parameter changes:

```
ngOnInit() {
    console.log('Retrieve some car data:', this.route.snapshot.params.id);
}
```

A route can have child routes

```
export const adminRoutes: Routes = [
  {
    path: 'admin',
    component: AdminPage,
    children: [
      {
        path: 'users',
        children: [
          { path: '', component: AdminUsersPage }
          { path: ':id', component: AdminUserDetailPage },
        ]
      }
    ]
  }
];
```

Pre-fetching data

Angular routes have a **resolve** mechanism to pre-fetch data.

```
export const routes: Route[] = [
    {
        path: 'cardetails/:id',
        component: CardetailsPage,
        resolve: {
            car: CarResolver
        }
    }
];
```

Pre-fetching data

The resolver itself looks as follows:

```
@Injectable()
export class CarResolver implements Resolve<Car> {
    constructor(
        private router: Router,
        private carApi: CarApi) { }

    resolve(route: ActivatedRouteSnapshot,
            state: RouterStateSnapshot) {
        let id = +route.paramMap.get('id');
        return this.carApi.get(id).pipe(map(car => {
            if (car) { return car; }
            this.router.navigate(['/start']);
            return null;
        }));
    }
```

Pre-fetching data

To get that fetched data in your component, check `data` of your `ActivatedRoute`.

```
constructor(private route: ActivatedRoute) { }

ngOnInit() {
    this.route.data
        .subscribe((data: { car: Car }) => {
            console.log('found car:', this.car);
        });
}
```

Control route access

By placing a **guard** on a route:

```
export const routes: Route[] = [
    {
        path: 'my-profile',
        component: MyProfilePage,
        canActivate: [AuthGuard]
    },
    ...
];
```

```
@NgModule({
    providers: [
        AuthGuard,
        ...
    ],
    bootstrap: [AppComponent]
})
export class AppModule { }
```

Control route access

The guard itself looks like this:

```
@Injectable()
export class AuthGuard implements CanActivate {
    constructor(private router: Router,
                private auth: AuthService) { }

    canActivate(route: ActivatedRouteSnapshot,
                state: RouterStateSnapshot) {
        if (!this.auth.isLoggedIn) {
            this.router.navigate(['/login']);
            return false;
        }

        return true;
    }
}
```

Lazy load routes

Angular supports lazy loading routes through the `loadChildren` property, thus shortening initial load time.

```
export const routes: Routes = [
    {
        path: 'admin',
        loadChildren: () => import('./admin/admin.module').then(m => m.AdminMod
        canLoad: [AdminAuthGuard]
    },
];
```

and define the subroutes using `RouterModule.forChild()`.

```
@NgModule({
    imports: [
        RouterModule.forChild(adminRoutes),
    ],
})
export class AdminModule { }
```

Lazy loud routes

If configured correctly, running a build with `ng build` should also generate a separate chunk.

```
chunk {scripts} scripts.fe0becbb6bce177076a6.js (scripts) 203 kB  [rendered]
chunk {0} 0.518b5c028a0e1bceefc6.js () 58.2 kB  [rendered]
chunk {1} runtime.542f3d4c29f3e62da136.js (runtime) 1.84 kB [entry] [rendered]
chunk {2} styles.1927224d725878310e0b.css (styles) 62.7 kB [initial] [rendered]
chunk {3} polyfills.713f759b4376e0518969.js (polyfills) 59.6 kB [initial] [rendered]
chunk {4} main.8c8cd98a88a14a100c83.js (main) 913 kB [initial] [rendered]
```

Recap

- Angular supports routing through the `RouterModule`
- Use `<router-outlet>` to define where views are loaded into
- Link to other views with `routerLink`
- Child routes are supported with `children`
- Protect routes with guards
- Lazy load routes using `loadChildren`

Lab time!

Create a second page where you can invite all your contacts to some sort of fun party