Routing & Navigation

Routing

- Routing means splitting the application into different areas usually based on rules that are derived from the current URL in the browser.

- Defining routes in our application is useful because we can:
  - Separate different areas of the app
  - Maintain the state in the app
  - Protect areas of the app based on certain rules

Client-Side vs Server-Side Routing

- For Server-Side routing, the server accepts a request and routes to a controller and the controller runs a specific action, depending on the path and parameters.

```
var express = require('express');
var router = express.Router();
router.get('/home', function(req, res){
        res.send('Homepage');
});
```

- Client-side routing is very similar in concept but different in implementation. With client-side routing we're not necessarily making a request to the server on every URL change. We call this **"Single Page Apps" (SPA)** because our server only gives us a single page and it's JavaScript that renders all the different pages.

Changing the URL?

- Because our app is client-side, it's not technically required that we change the URL when we change pages (no HTTP requests). But what would be the consequences of using the same URL for all pages?
  - You wouldn't be able to **refresh** the page and keep your location within the app
  - You wouldn't be able to **bookmark** a page and come back to it later
  - You wouldn't be able to **share the URL** of that page with others

Changing URL Techniques

- Use the **anchor tag** as the client-side URL (fragments): take the anchor tags and use them to represent the routes within the app by formatting them as paths.
- With the introduction of **HTML5**, browsers acquired the ability to programmatically create new browser history entries that change the displayed URL without the need for a new request. This is achieved using the `history.pushState` method that exposes the browser's navigational history to JavaScript.

Angular Routing Components

- The Angular Router enables navigation from one view to the next as users perform application tasks.

- There are 3 main components which used to configure routing in Angular:
  - Routes: describes the routes the application supports. Will be passed to RouterModule and imported in NgModule.
  - RouterOutlet: a placeholder component that get expanded to each route's content.
  - RouterLink: a directive which is used to link to routes so browser won't refresh when change routes

New way to add Angular routing

- Since Angular 8, before proceeding to generate the project, the CLI will prompt you if:
- **Would you like to add Angular routing?** The default answer is NO, if you type y:
  - CLI installs the `@angular/router` package in the project
  - Generate a `src/app/app-routing.module.ts` file
  - Add a <router-outlet> in the `src/app/app.component.html`
- To our Angular Application.
- You have to do it manually in previous versions.

Understanding what CLI automatically did for us
Adding `<base href>`

- This tag is traditionally used to tell the browser where to look for images and other resources declared using relative paths.
- Angular Router also relies on this tag to determine how to construct its routing information.
  - If we have a route with a path of `/hello` and our base element declares `<base href="/app">`, the application will use `/app/hello` as the concrete path.
- The `<base href>` tag is not specific to Angular Router. It's an HTML tag which specifies the base URL for all relative URLs in the page.

Understanding what CLI automatically did for us
Creating a Routing Module

- Create a routing module inside the root application module
  - `ng generate module app-routing --flat`

- Import the Router and Setting up Routing

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/
router';
const routes: Routes = [];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Understanding what CLI automatically did for us

- Adding the Router-Outlet
  - Adding `<router-outlet>` to src/app/app.component.html
    - `<router-outlet></router-outlet>`

- Importing the Routing Module in the Main Application Module

```
import { AppRoutingModule } from './app-routing.module’;

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Router Module – Hello World Demo

```
import { Routes, RouterModule } from '@angular/router';
const routes: Routes = [
    { path: '', redirectTo: 'home', pathMatch: 'full' },
    { path: 'home', component: HomeComponent },
    { path: 'aboutus', component: AboutUsComponent },
    { path: 'contact', component: ContactUsComponent }
];
@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
export class AppRoutingModule { }
```

src/app/app.component.html

```
<nav>
    <ul>
        <li><a [routerLink]="['home']">Home</a></li>
        <li><a [routerLink]="['aboutus']">About</a></li>
        <li><a [routerLink]="['contact']">Contact us</a></li>
    </ul>
</nav>

<router-outlet></router-outlet>
```

Routes and Paths

A `route` is an object (instance of `Route`) that provides information about which component maps to a specific path.

A collection of routes defines the router configuration which is an instance of `Routes`.

A `path` is the fragment of a URL that determines where exactly is located the resource(or page) you want to access.

`redirectTo` is the URL fragment which you will be redirected to if a route is matched.

```
const routes: Routes = [
    { path: '', redirectTo: 'home', pathMatch: 'full’ },

    { path: 'home', component: HomeComponent },

    { path: 'aboutus', component: AboutUsComponent },

    { path: 'contact', component: ContactUsComponent }
];
```

Routing Matching Strategies

- The built-in matching strategies are `prefix`(default) and `full`.
- When the matching strategy of a route is **prefix**, the router will simply check if the start of the browser's URL is prefixed with the route's path. If that's the case, it will render the related component.
- This is not always the wanted behavior. In some scenarios, you want the router to match the full path before rendering a component. You can set the full strategy using the pathMatch property of a route. For example:

```
{ path: '', redirectTo: 'home', pathMatch: 'full' }
```

- Wildcard string (**) – match by the router if the visited URL doesn't match any paths in the router configuration.

```
{ path: '**', redirectTo: 'home' }
```

RouterOutlet <router-outlet>

- When we change routes, we want to keep our outer layout template and only substitute the inner section of the page with the route's component.
- In order to describe to Angular **where** in our page we want to render the contents for each route, we use the **router-outlet** directive.
- We are going to use our `AppComponent` as a layout which contains all our `RouterLink` and `RouterOutlet` directives which exported by `RouterModule`.

RouterLink [routerLink]

- If we might try linking to the routes directly using pure HTML will result links when clicked they trigger a **page reload** (not SPA):
  - `<a href="/home">Home</a>`

- To solve this problem, we will use the `RouterLink` directive:

src/app/app.component.html

```
<nav>
    <ul>
        <li><a [routerLink]="['home']">Home</a></li>
        <li><a [routerLink]="['aboutus']">About</a></li>
        <li><a [routerLink]="['contact']">Contact us</a></li>
    </ul>
</nav>

<router-outlet></router-outlet>
```

Route Parameters – Mandatory Params

- Dynamic routes are often used in web applications to pass data (parameters) or state to the application or between various components and pages.

```
{ path: 'products/:id', component: ProductDetailComponent }
```

- In order to read route parameters, we need to first import `ActivatedRoute` Service and we inject it into the constructor of our component:

```
export class ProductDetailComponent {
  product: Product;

  constructor(private productService: ProductService,
                      private route: ActivatedRoute) {
    route.params.subscribe(params =>
        { this.product = this.productService.getById(+params['id']); });
  }
}
```

Query Parameters – Optional Params

- Query parameters can be applied to all routes without specifying them in the routes list.
- In order to read query parameters, we need to first import ActivatedRoute Service and we inject it into the constructor of our component:

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

constructor(private route: ActivatedRoute) {
        route.queryParams.subscribe( params => { this.id = params['id']; });
}
```

route.queryParams is an observable

Set Parameters Examples

Template

```
<a [routerLink]="['users', 'update', id.value]">Update</a>
// users/update/1
<a [routerLink]="['users', 'update']" [queryParams]="{id: id.value}">Update</a>
// users/update?id=1
<a [routerLink]="['home']" [fragment]="top">Go to top</a>
// home#top
```

Component Class

```
this.router.navigate(['users', 'update'], { queryParams: { id: id.value} })
// users/update?id=1
this.router.navigate(['home'], fragment: 'section1'})
// home#section1
```

Fragments

- Fragments allow us to jump to specific place on our page, they represent everything after the # symbol in the URL.
- In order to read fragments, we need to first import ActivatedRoute Service and we inject it into the constructor of our component:

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

constructor(private route: ActivatedRoute) {
        route.fragment.subscribe( fragment => { this.id = fragment; });
}
```

route.fragment is an observable

Observable Subscription

- When subscribing to the observable we generate a Subscription, after we finish from the component, this subscription will stay alive and will cause memory leak. You should always destroy your observable subscription.
- **You have to `unsubscribe()` from the Observable in the `ngOnDestroy` method/lifecycle hook**

```
import { Subscription } from 'rxjs';

export class ProductDetailComponent implements OnDestroy {
  product: Product;

  private subscription: Subscription;
  constructor(private productService: ProductService, private route: ActivatedRoute) {
    this.subscription = route.params.subscribe(params => {
         this.product = this.productService.getById(+params['id']); });
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}
```

Styling Routes

- To make our link have an extra CSS style when its route is being activated we use `routerLinkActive` Directive:

```
<a class="nav-link" [routerLink]="['home']" routerLinkActive="active">Home</a>
```

Imperative Routing

- You can also navigate to a route imperatively (in your code), you need to inject the **`Router`** service then you may call **`navigate()`** like this:

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

constructor(private router: Router) {}

this.router.navigate(['home'])
```

Guards

- Guards are useful Services which allow you to control access to and from a Route/Component.
- Here are the 4 types of routing guards available:
  - `CanActivate`: Controls if a route can be activated.
  - `CanActivateChild`: Controls if children of a route can be activated.
  - `CanLoad`: Controls if a route can even be loaded. This becomes useful for feature modules that are lazy loaded. They won't even load if the guard returns false.
  - `CanDeactivate`: Controls if the user can leave a route. Note that this guard doesn't prevent the user from closing the browser tab or navigating to a different address. It only prevents actions from within the application itself.

CanActivate

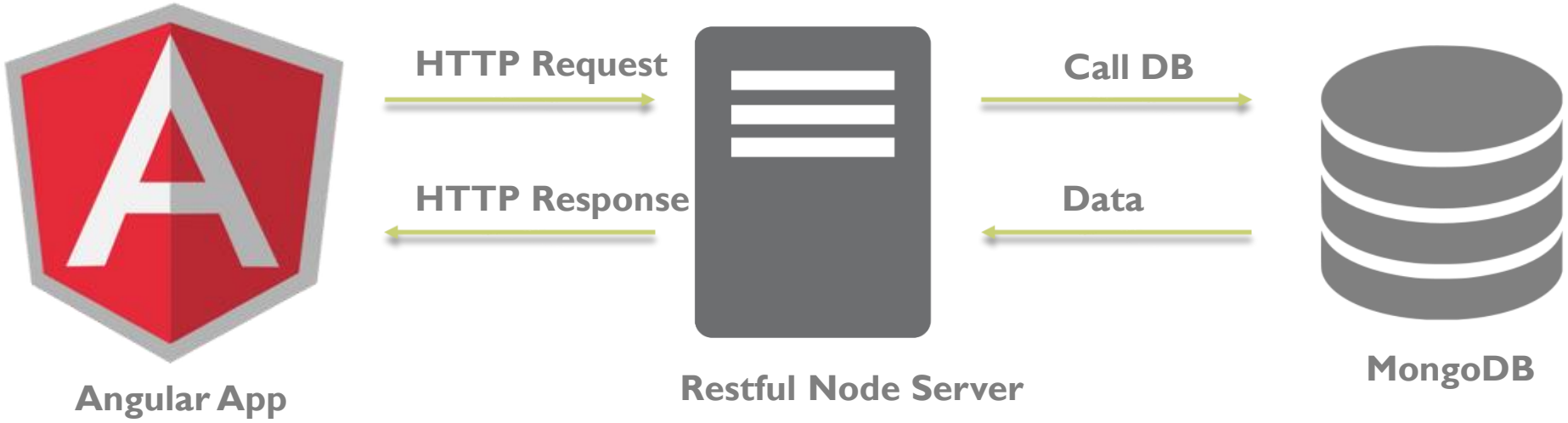
- Route guards can return a boolean, Promise<boolean> or Observable<boolean> (asynchronous boolean objects) to tell the router if the route can be activated or not.
- Since Angular 7.1, you can also return an UrlTree variable which provides the new router state (route) that should be activated.

```
@Injectable({
  providedIn: 'root'
})
export class AdminGuard implements CanActivate {
  constructor(private authService: AuthService, private router: Router) { }
  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    if (this.authService.isUserAuthenticated()) {
      return true;
    } else {
      return this.router.parseUrl("/contact");
    }
  }
}
```

HTTP

Making HTTP requests from Angular

- The Angular HTTP Module simplifies application programming with the XHR and JSONP APIs. All async requests return an `Observable`.



What is Angular HttpClient?

- Front end applications, built using frameworks like Angular communicate with backend servers through REST APIs (which are based on the HTTP protocol) using either the `XMLHttpRequest` interface or the `fetch()` API.

- Angular `HttpClient` makes use of the `XMLHttpRequest` interface that supports both modern and legacy browsers.

- The `HttpClient` is available from the `@angular/common/http` package.

Why Angular HttpClient?

- The HttpClient builtin service provides many advantages to Angular developers:
  - HttpClient makes it easy to send and process HTTP requests and responses,
  - HttpClient has many built-in features for implementing test units,
  - HttpClient makes use of RxJS Observables for handling asynchronous operations instead of Promises which simplify common web development tasks such as
    - The Listening for the progression of download and upload operations,
    - Easy error handling,
    - Retrying failed HTTP requests, etc.

Use HttpClientModule

- We add in Root Module, so it's available in entire application

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

API Service

```
@Injectable({
  providedIn: 'root'
})
export class ApiService {

  constructor(private http: HttpClient) { }
  baseUrl: string = 'http://localhost:3000/users/';

  getUsers(): Observable<ApiResponse> {
    return this.http.get<ApiResponse>(this.baseUrl);
  }

  getUserById(id: string): Observable<ApiResponse> {
    return this.http.get<ApiResponse>(this.baseUrl + id);
  }

  createUser(user: User): Observable<ApiResponse> {
    return this.http.post<ApiResponse>(this.baseUrl, user);
  }

  updateUser(user: User): Observable<ApiResponse> {
    return this.http.put<ApiResponse>(this.baseUrl + user._id, user);
  }

  deleteUser(id: string): Observable<ApiResponse> {
    return this.http.delete<ApiResponse>(this.baseUrl + id);
  }
}
```

```
export class ApiResponse {

  status: number;
  message: string;
  result: any;
}
```

Display All Users

```
@Component({
  selector: 'app-list-user',
  templateUrl: './list-user.component.html',
  styles: []
})
export class ListUserComponent implements OnInit {

  users: User[];

  constructor(private router: Router, private apiService: ApiService) { }

  ngOnInit() {
    this.apiService.getUsers()
      .subscribe(data => {
        this.users = data.result;
      });
  }
}
```

Delete a User

```
@Component({
  selector: 'app-list-user',
  templateUrl: './list-user.component.html',
  styles: []
})
export class ListUserComponent implements OnInit {

  users: User[];

  constructor(private router: Router, private apiService: ApiService) { }


  deleteUser(user: User): void {
    console.log('user', user);
    this.apiService.deleteUser(user._id)
      .subscribe(data => {
        this.users = this.users.filter(u => u !== user);
      })
  }
```

Add a User

```
export class AddUserComponent implements OnInit {

  constructor(private formBuilder: FormBuilder, private router: Router, private apiService: ApiService) { }

  addForm: FormGroup;

  ngOnInit() {
    this.addForm = this.formBuilder.group({
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      birthDate: ['', Validators.required],
      role: ['', Validators.required]
    });
  }

  onSubmit() {
    this.apiService.createUser(this.addForm.value)
      .subscribe(data => {
        this.router.navigate(['list-user']);
      });
  }
}
```

Add a User

```
<div class="col-md-6 user-container">
    <h2 class="text-center">Add User</h2>
    <form [formGroup]="addForm" (ngSubmit)="onSubmit()">

        <div class="form-group">
            <label for="firstName">First Name:</label>
            <input formControlName="firstName" placeholder="First Name" name="firstName" id="firstName">
        </div>

        <div class="form-group">
            <label for="lastName">Last Name:</label>
            <input formControlName="lastName" placeholder="Last name" name="lastName" id="lastName">
        </div>

        <div class="form-group">
            <label for="birthDate">BirthDate:</label>
            <input type="date" formControlName="birthDate" placeholder="birthDate" name="birthDate" id="birthDate">
        </div>

        <div class="form-group">
            <label for="role">Role:</label>
            <input formControlName="role" placeholder="role" name="role" id="role">
        </div>

        <button class="btn btn-success">Add</button>
    </form>
</div>
```

Edit a User

```
export class EditUserComponent implements OnInit {

  user: User;
  editForm: FormGroup;
  pipe = new DatePipe('en-US');

  constructor(private formBuilder: FormBuilder, private router: Router, private apiService: ApiService) { }

  ngOnInit() {
    let userId = window.localStorage.getItem("editUserId");
    if (!userId) {
      alert("Invalid action.")
      this.router.navigate(['list-user']);
      return;
    }
    this.editForm = this.formBuilder.group({
      _id: [''],
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      birthDate: ['', Validators.required],
      role: ['', Validators.required]
    });
    this.apiService.getUserById(userId)
      .subscribe(data => {
        let temp = data.result.birthDate;
        data.result.birthDate = this.pipe.transform(temp, 'yyyy-MM-dd');
        delete data.result.__v;
        this.editForm.setValue(data.result);
      });
  }
```

```
onSubmit() {
   this.apiService.updateUser(this.editForm.value)
     .pipe(first())
     .subscribe(
       data => {
         if (data.status === 200) {
           alert('User updated successfully.');
           this.router.navigate(['list-user']);
         } else {
           alert(data.message);
         }
       },
       error => {
         alert(error);
       });
 }
```

Edit a User

```
<form [formGroup]="editForm" (ngSubmit)="onSubmit()">
        <div class="hidden">
            <input type="hidden" formControlName="_id" placeholder="id" name="_id" class="form-control" id="id">
        </div>
        <div class="form-group">
            <label for="firstName">First Name:</label>
            <input formControlName="firstName" placeholder="First Name" name="firstName" class="form-control" id="firstName">
        </div>

        <div class="form-group">
            <label for="lastName">Last Name:</label>
            <input formControlName="lastName" placeholder="Last name" name="lastName" class="form-control" id="lastName">
        </div>

        <div class="form-group">
            <label for="birthDate">BirthDate:</label>
            <input type="date" formControlName="birthDate" placeholder="birthDate" name="birthDate" id="birthDate">
        </div>

        <div class="form-group">
            <label for="role">Role:</label>
            <input formControlName="role" placeholder="role" name="role" class="form-control" id="role">
        </div>

        <button class="btn btn-success">Update</button>
    </form>
```

Reference

- HTTP
  - https://www.techiediaries.com/angular-http-client/

- Router
  - https://angular.io/guide/router