

# Building a Site

## Bootstrap

- One of the most useful tools for front end design is Bootstrap. It can be installed with:  
`npm install bootstrap jquery popper.js`
- The file `Angular.json` contains where to find certain elements of the application, such as the Bootstrap file.
- For example:

```
{}..  
  "assets": [  
    "src/favicon.ico",  
    "src/assets"  
  ],  
  "styles": [  
    "src/styles.css",  
    "node_modules/bootstrap/dist/css/bootstrap.min.css"  
  ],  
  "scripts": [  
    "node_modules/jquery/dist/jquery.min.js",  
    "node_modules/popper.js/dist/umd/popper.min.js",  
    "node_modules/bootstrap/dist/js/bootstrap.min.js"  
  ]  
},  
{} ...
```

This would apply the css in `styles.css` to and node modules every page in the application

## Routing Basics

- In `app.module.ts`:

```

const routes: Routes = [
  // The extension of the URL, and what component or action should be loaded
  { path : 'admin/users', component : UsersComponent }
];

@NgModule({
  declarations: [
    AppComponent,
    MenuComponent,
    CalendarComponent,
    RoomsComponent,
    UsersComponent
  ],
  imports: [
    BrowserModule,
    // It is necessary to add this module with the routes
    // It never changes, ceremonial code.
    RouterModule.forRoot(routes)
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Then in app.component.html:

```

<div class="container">
  <app-menu></app-menu>
  <router-outlet></router-outlet>
</div>

```

router-outlet will be replaced with whatever component it finds in the router module with the given URL

- The wild card symbol in routing is `**`. So to set up a 404 page use:

```

// Note: It's usually production standard to put these in a separate file
// called app.routing.module.ts
const routes = [
  { path : '', component : HomeComponent },
  { path : '404', component : PageNotFoundComponent },

```

```
{ path : "**", redirectTo : '/404' }  
};
```

The wild card must always come at the end

# Making Links work with Routing

- The problem with the above is that the links perform a GET request on the server, reloading the entire Angular application every time.
  - Angular is intended to be a single page application
- Solution: Remove all href tags that link to a server and replace them with a click event function.
  - This requires the use of the Routing package from @angular/router, and is added to the constructor In the menu.html:

```
...  
<a class="dropdown-item" (click)="navigateToRoomsAdmin()">Rooms</a>  
...
```

In the menu.component.ts class:

```
@Component({  
  selector: 'app-menu',  
  templateUrl: './menu.component.html',  
  styleUrls: ['./menu.component.css']  
})  
export class MenuComponent implements OnInit {  
  
  constructor(private router: Router) { }  
  
  ngOnInit(): void {  
  }  
  
  navigateToRoomsAdmin() {  
    // navigate to /admin/rooms  
    this.router.navigate(['admin','rooms']);  
  }  
  
}
```

# Routing for Sub-Components

- To have a url that changes within an application we use yet another library named `ActiveRouting`

```
rooms: Array<Room>;
selectedRoom: Room;

constructor(private dataService: DataService,
private route: ActivatedRoute) { }

ngOnInit(): void {
  this.rooms = this.dataService.rooms;
  // inspect the URL to see if there is a parameter on the path
  this.route.queryParams.subscribe((params) => {
    const id = params['id'];
    if (id) {
      // cast a variable to a number using +
      this.selectedRoom = this.rooms.find( room => room.id === +id);
    }
  })
}
```

# Models and Views

- If we want to store data so users can see it later, it must be connected to a backend
  - Spring, ExpressJS, RubyOnRails, etc.
- When the data arrives from the backend (the Observable) we create an event to contain it in an array.
- For the above reason we use getters and setters of type Observable

```
private rooms: Array<Room>;
private users: Array<User>;

getRooms(): Observable<Array<Room>> {
  return of(this.rooms);
}
```

```
}

getUsers(): Observable<Array<User>> {
  return of(this.users);
}
```

Then in the ngOnInit function we subscribe to that event

```
ngOnInit(): void {
  this.dataService.getRooms().subscribe(next => this.rooms = next );
}
```

# Pipes

Pipes allow us to change the way something is displayed, such as a date

```
selectedDate = new Date();
```

```
<p>The selected date is {{selectedDate | date:'yyyy-MM-dd'}}</p>
```

Check out some other pipe usages in the [documentation](#)

The alternative to the above would be

```
ngOnInit(): void {
  const date: string = formatDate(this.selectedDate, 'yyyy-MM-dd', 'en-US');
}
```

You could also specify the locale this way. But pipes are a pretty cool feature of Angular.

---

Revision #1

Created 16 April 2022 23:25:17 by Elkip

Updated 16 April 2022 23:40:55 by Elkip