UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

WEB TECHNOLOGIES — LECTURE 22

ANGULAR: TO-DO LIST APP

Luigi Libero Lucio Starace, PhD

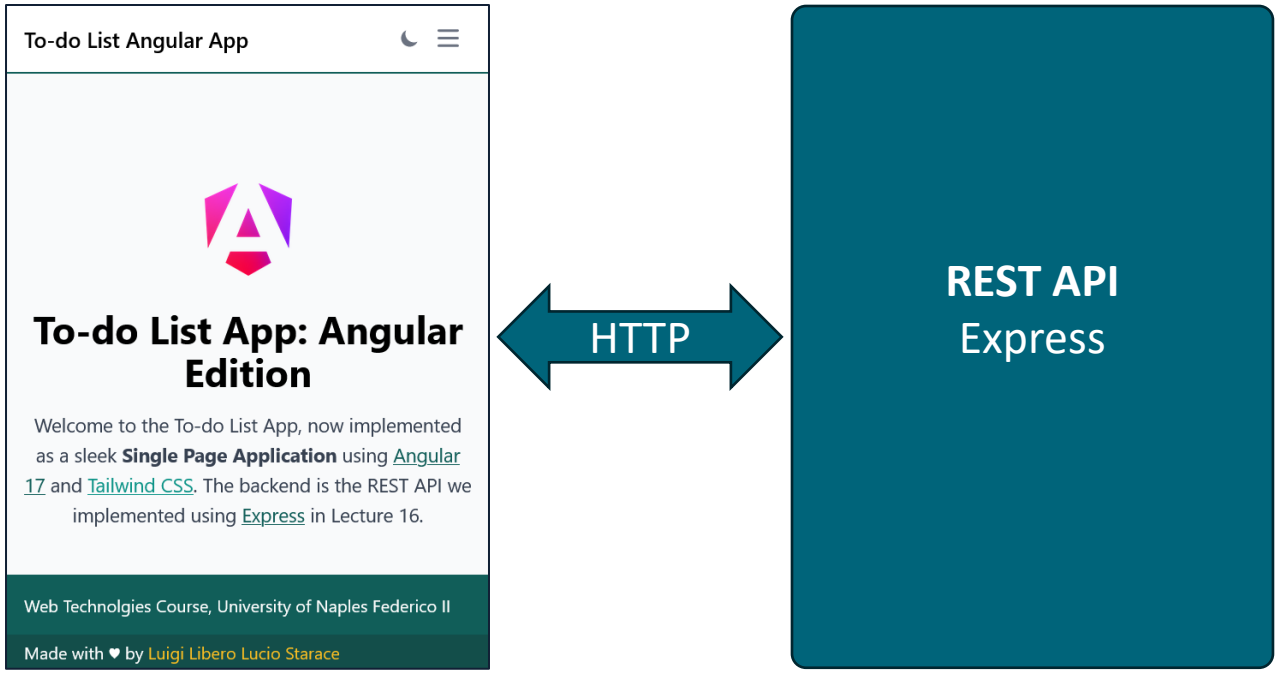
luigiliberolucio.starace@unina.it

https://luistar.github.io

https://www.docenti.unina.it/luigiliberolucio.starace

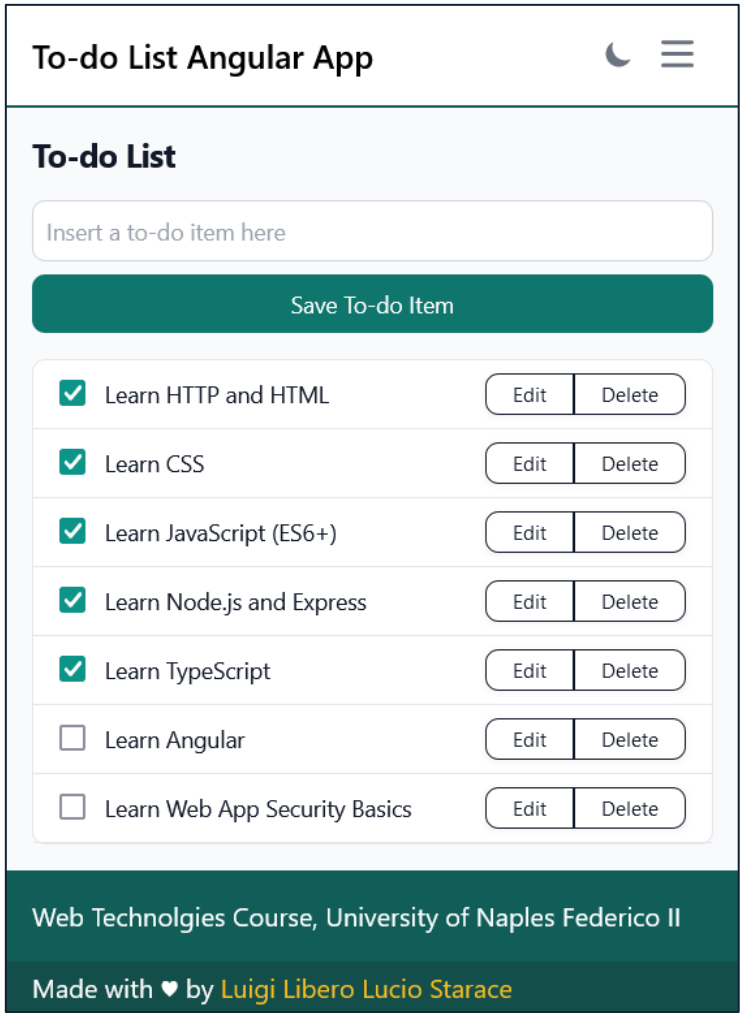
TO–DO LIST WEB APP: SPA EDITION

- Today, we'll implement the good ol' To-do List App as a modern SPA
- We will use the REST backend developed back in **Lecture 15**
- And add a nice Angular 17 frontend on top of it



TO–DO LIST WEB APP: SPA EDITION

- During today's lecture, we'll check out the Angular To-do List SPA the teacher implemented
- The source code is available in the course materials on Teams
- In the remainder of these slides, we'll go over some of the interesting parts for reference



TO–DO LIST SPA: LIVE DEMO TIME

TO–DO LIST SPA: ROUTES

```
/* Nothing much is going on here */
export const routes: Routes = [{
    path: "home", component: HomepageComponent,
    title: "To-do List Angular App"
  }, {
    path: "login", component: LoginComponent,
    title: "Login | To-do List Angular App"
  }, { /*some routes were omitted for the sake of brevity*/
    path: "todos", component: TodoPageComponent,
    title: "To-do List | To-do List Angular App",
    canActivate: [authGuard]
  }, {
    path: "todos/:id", component: TodoDetailComponent,
    title: "To-do Detail | To-do List Angular App",
    canActivate: [authGuard]
  }
];
```

Notice that two routes are guarded by authGuard

Notice that this route is parametric!

TO–DO LIST SPA: AUTHGUARD

```
export const authGuard: CanActivateFn = (route, state) => {
  const authService = inject(AuthService);
  const toastr = inject(ToastrService);
  const router = inject(Router);
  if(authService.isUserAuthenticated()){
    return true;
  } else {
    toastr.warning("Please, login to access this feature", "Unauthorized!");
    return router.parseUrl("/login"); //return a UrlTree
  }
};
```

TO–DO LIST SPA: LOGIN COMPONENT

```
export class LoginComponent {
  toastr = inject(ToastrService);
  router = inject(Router);
  restService = inject(RestBackendService);
  authService = inject(AuthService);
  submitted = false;
  loginForm = new FormGroup({
    user: new FormControl('', [Validators.required]),
    pass: new FormControl('', [
      Validators.required,
      Validators.minLength(4),
      Validators.maxLength(16)])
  });

  handleLogin() {/* handles login submit */}
}
```

TO–DO LIST SPA: LOGIN COMPONENT

```
handleLogin() {
  this.submitted = true;
  if(this.loginForm.invalid){
    this.toastr.error("Invalid data!", "Oops! Invalid data!");
  } else {
    this.restService.login({
      usr: this.loginForm.value.user as string,
      pwd: this.loginForm.value.pass as string,
    }).subscribe({
      next: (token) => { this.authService.updateToken(token as string); },
      error: (err)  => { this.toastr.error("Invalid", "Oops! Invalid credentials");},
      complete: ()  => {
        this.toastr.success(`Welcome`,`Welcome ${this.loginForm.value.user}!`);
        this.router.navigateByUrl("/todos");
      }
    })
  }
}
```

TO–DO LIST SPA: LOGIN TEMPLATE

- Interesting bit from the LoginComponent template
- We conditionally show input errors only when needed
  - We use the new Angular 17 control flow syntax **@if**!

```
<label for="pass" class="classes-omitted">Password</label>
<input type="password" formControlName="pass" id="pass" class="omitted" required="">
@if(submitted && loginForm.controls.pass.errors){
  @if(loginForm.controls.pass.errors['required']){
    <p class="form-error">Password is required.</p>
  }
  @if(loginForm.controls.pass.errors['minlength']){
    <p class="form-error">Password should contain at least 4 characters</p>
  }
}
```

TO-DO LIST SPA: AUTH SERVICE

```
type AuthState = {
  user: string | null,
  token: string | null,
  isAuthenticated: boolean
}

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

  authState: WritableSignal<AuthState> = signal<AuthState>({
    user: this.getUser(),
    token: this.getToken(), //get token from localStorage, if there
    isAuthenticated: this.verifyToken(this.getToken()) //verify it's not expired
  })
```

TO-DO LIST SPA: AUTH SERVICE

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

  //continues from the previous slide

  user = computed(() => this.authState().user);
  token = computed(() => this.authState().token);
  isAuthenticated = computed(() => this.authState().isAuthenticated);
```

TO–DO LIST SPA: AUTH SERVICE

```
@Injectable({ providedIn: 'root'})
export class AuthService {
 //continues from the previous slide
 constructor(){
   effect( () => { //we use an effect to keep data aligned with localStorage
     const token = this.authState().token;
     const user = this.authState().user;
     if(token !== null)
       localStorage.setItem("token", token);
     else
       localStorage.removeItem("token");
     if(user !== null)
       localStorage.setItem("user", user);
     else
       localStorage.removeItem("user");
   });
 }
```

TO-DO LIST SPA: REST BACKEND SERVICE

```
@Injectable( { providedIn: 'root' } )
export class RestBackendService {
  url = "http://localhost:3000"
  constructor(private http: HttpClient) {}
  httpOptions = {
    headers: new HttpHeaders({
      'Content-Type': 'application/json'
    })
  };

  login(loginRequest: AuthRequest){
    const url = `${this.url}/auth`;
    return this.http.post<string>(url, loginRequest, this.httpOptions);
  }
  //rest of the service omitted for the sake of brevity
}
```

```
export interface AuthRequest {
  usr: string,
  pwd: string
}
```

TO-DO LIST SPA: AUTH INTERCEPTOR

```
function authInterceptor(request: HttpRequest, next: HttpHandlerFn) {
  const authService = inject(AuthService);
  const token = authService.getToken();

  if (token) {
    // Clone the request and add the Authorization header with the token
    request = request.clone({
      setHeaders: {
        Authorization: 'Bearer ' + token
      }
    });
  }

  return next(request); // Continue with subsequent interceptors, if any
}
```

TO-DO LIST SPA: TO-DO PAGE COMPONENT

```
export class TodoPageComponent {
  createTodoSubmitted = false;
  restService = inject(RestBackendService);
  toastr = inject(ToastrService);
  router = inject(Router);
  todos: TodoItem[] = []; //array of TodoItems

  newTodoForm = new FormGroup({
    todo: new FormControl('', [Validators.required])
  });

  ngOnInit() { this.fetchTodos(); }
  fetchTodos(){ /*...*/ }
  handleTodoSubmit(){ /*...*/ }
  handleDelete(id: number | undefined){ /*...*/ }
}
```

TO–DO LIST SPA: TO–DO TEMPLATE

- Interesting bit shown below
- Note that `<app-todo-item>` is a separate component, taking care of displaying a single To-do Item in the list
- The TodoItem to show is passed as a property
- Note that the component can also emit `delete` events

```
<ul class="classes-omitted">
  @for(item of todos; track item.id){
    <app-todo-item [todoItem]=item (delete)="handleDelete($event)"/>
  } @empty {
    <li class="classes-omitted">No to-do items to show.</li>
  }
</ul>
```

THE TO-DO ITEM COMPONENT

```
export class TodoItemComponent {
  @Input({ required: true}) todoItem: TodoItem;
  @Output() delete: EventEmitter<number | undefined> = new EventEmitter();
  editLink = "";
  restBackend = inject(RestBackendService);
  toastr = inject(ToastrService);


  ngOnInit(){ this.editLink = "/todos/"+this.todoItem?.id; }

  handleToggleDoneStatus(){ /*...*/ }
  handleDelete(){ /*...*/ }
}
```