8 - Seguridad y validación de datos

8.1. Formularios para recogida segura de datos

8.1.1 Formularios reactivos

```
ng g c routes/auth/register --type=form
```

```html
<form [formGroup]="form" (submit)="onSubmit()">
  <label for="username">
    <span>Username</span>
    <small>{{ form.controls['username'].errors | json }}</small>
    <input
      id="username"
      type="text"
      formControlName="username"
      [attr.aria-invalid]="form.controls['username'].invalid" />
  </label>
  <label for="email">
    <span>Email</span>
    <small>{{ form.controls['email'].errors | json }}</small>
    <input id="email" type="email" formControlName="email" [attr.aria-
invalid]="form.controls['email'].invalid" />
  </label>
  <label for="password">
    <span>Password</span>
    <small>{{ form.controls['password'].errors | json }}</small>
    <input
      id="password"
      type="password"
      formControlName="password"
      [attr.aria-invalid]="form.controls['password'].invalid" />
  </label>
  <label for="confirm">
    <span>Confirm password</span>
    <small>{{ form.controls['confirm'].errors | json }}</small>
    <input
      id="confirm"
      type="password"
      formControlName="confirm"
      [attr.aria-invalid]="form.controls['confirm'].invalid" />
  </label>
  <label for="terms">
```

```
    <span>Accept the terms and conditions</span>
    <input id="terms" type="checkbox" formControlName="terms" [attr.aria-
invalid]="form.controls['terms'].invalid" />
  </label>
  <button type="submit" [disabled]="form.invalid">Register</button>
</form>
```

```
export class RegisterForm {
  register = output<Register>();

  form: FormGroup = new FormGroup(
    {
      username: new FormControl("A", Validators.required),
      email: new FormControl("a@b.c", [Validators.required, Validators.email]),
      password: new FormControl("1234", [Validators.required,
Validators.minLength(4)]),
      confirm: new FormControl("123", [Validators.required,
Validators.minLength(4)]),
      terms: new FormControl(false, Validators.requiredTrue),
    },
    {
      validators: matchValidator("password", "confirm"),
    }
  );

  onSubmit() {
    if (this.form.valid) {
      // eslint-disable-next-line @typescript-eslint/no-unused-vars
      const { confirm, ...register } = this.form.value;
      this.register.emit(register);
    }
  }
}
```

8.1.2 Validaciones personalizadas

```
/** Match validator
 * @param controlName The first form control
 * @param matchingControlName The second form control
 * @returns The validator function that checks if both values are equal
 */
export function matchValidator(controlName: string, matchingControlName:
string): ValidatorFn {
  // Main function is a factory that returns the validator function for the
current form
  return (form: AbstractControl): ValidationErrors | null => {
    const first = form.get(controlName);
    const second = form.get(matchingControlName);

    if (first && second && first.value !== second.value) {
      const validationErrors: ValidationErrors = { dataMismatch: true };
      second.setErrors(validationErrors);
      return validationErrors;
    }

    return null;
  };
}
```

```
<article>
  <header>
    <h2>Register</h2>
  </header>
  <main>
    <lab-register (register)="onRegister($event)" />
  </main>
  <footer>
    <a [routerLink]="['/auth', 'login']">Login if already have an account</a>
  </footer>
</article>
```

```
export default class RegisterPage {
  onRegister(register: Register) {
    console.log("Register", register);
  }
}
```

8.1.3 Control de presentación

```
ng g c shared/ui/control
```

```
export class ControlComponent {
  /** The form control name to bind to */
  controlName = input.required<string>();
  /** The label to display */
  labelDisplay = input.required<string>();
  /** The errors to display if any */
  errors = input<unknown>();
}
```

```
<label [for]="controlName()">
  <span>{{ labelDisplay() }}</span>
  @if (errors()) {
  <small>{{ errors() | json }}</small>
  }
  <ng-content />
</label>
```

8.2. Interceptores de comunicaciones y guardias de navegación

8.2.1 Envío de credenciales y almacenaje de Token

```
ng g s shared/api/auth-repository
```

```
@Injectable({
  providedIn: "root",
})
export class AuthRepository {
  #apiUrl = "http://localhost:3000";
  #http = inject(HttpClient);
  #authStore = inject(AuthStore);
  postRegister$(register: Register): Observable<UserAccessToken> {
    return this.#http
      .post<UserAccessToken>(`${this.#apiUrl}/register`, register)
  }
      .pipe(tap((userAccessToken) =>
this.#authStore.setState(userAccessToken)));
  postLogin$(login: Login): Observable<UserAccessToken> {
    return this.#http
      .post<UserAccessToken>(`${this.#apiUrl}/login`, login)
      .pipe(tap((userAccessToken) =>
this.#authStore.setState(userAccessToken)));
  }
}
```

```
export default class RegisterPage {
  authRepository: AuthRepository = inject(AuthRepository);

  onRegister(register: Register) {
    this.authRepository.postRegister$(register).subscribe();
  }
}
```

```
ng g s shared/state/auth-store
```

```
@Injectable({
  providedIn: "root",
})
export class AuthStore {
  #localRepository: LocalRepository = inject(LocalRepository);
  #state: WritableSignal<UserAccessToken> = signal<UserAccessToken>
(NULL_USER_ACCESS_TOKEN);
  isAuthenticated: Signal<boolean> = computed(() => this.#state().accessToken
!== "");
  isAnonymous: Signal<boolean> = computed(() => this.#state().accessToken ===
"");
  userId: Signal<string> = computed(() => this.#state().user.id);
  setState(userAccessToken: UserAccessToken): void {
    this.#state.set(userAccessToken);
  }
}
```

8.2.2 Interceptores de comunicaciones

```
ng g interceptor core/auth
```

```
export const authInterceptor: HttpInterceptorFn = (req: HttpRequest<unknown>,
next: HttpHandlerFn) => {
  const authStore = inject(AuthStore);
  const accessToken: string = authStore.accessToken();
  const router: Router = inject(Router);
  req = req.clone({
    setHeaders: {
      Authorization: accessToken ? `Bearer ${accessToken}` : "",
    },
  });
  return next(req).pipe(
    catchError((error) => {
      if (error.status === 401) {
        authStore.setState(NULL_USER_ACCESS_TOKEN);
        router.navigate(["/auth", "login"]);
      }
      return throwError(() => error);
    })
  );
};
```

```
export const appConfig: ApplicationConfig = {
  providers: [
    provideClientHydration(),
    provideHttpClient(withFetch(), withInterceptors([authInterceptor])),
    provideRouter(routes, withComponentInputBinding()),
  ],
};
```

8.2.3 Guardias de navegación

```
ng g g core/auth --implements=CanActivate
```

```
export const authGuard: CanActivateFn = () => {
  const authStore = inject(AuthStore);
  if (authStore.isAuthenticated()) {
    return true;
  }
  const router = inject(Router);
  return router.createUrlTree(["/auth", "login"]);
};
```

```
export const routes: Routes = [
  {
    path: "bookings/:slug",
    loadComponent: () => import("./routes/bookings/bookings.page"),
    canActivate: [authGuard],
    resolve: {
      activity: activityResolver,
    },
  },
];
```

```
ng g r routes/bookings/activity
```

```
export const activityResolver: ResolveFn<Activity> = (route:
ActivatedRouteSnapshot) => {
  const slug: string = route.paramMap.get("slug") || "";
  const bookingsService = inject(BookingsService);
  return bookingsService.getActivityBySlug$(slug);
};
```

```
export default class BookingsPage {
  #route = inject(ActivatedRoute);
  #resolvedActivity: Activity = this.#route.snapshot.data["activity"];
  activity: Signal<Activity> = signal(this.#resolvedActivity);
}
```

8.3. Presentación de feedback al usuario

8.3.1 Feedback de operaciones

```
export type FeedbackStatus = "idle" | "busy" | "success" | "error";
export type Feedback = { status: FeedbackStatus; message: string };
```

```
ng g c shared/ui/feedback
```

```
export class FeedbackComponent {
  feedback: InputSignal<Feedback> = input<Feedback>({ status: "idle", message:
"" });
  status: Signal<FeedbackStatus> = computed(() => this.feedback().status);
  message: Signal<string> = computed(() => this.feedback().message);
}
```

```
<div>
  @switch (status()) { @case ('busy') {
  <fieldset role="group">
    <input disabled [value]="message() || 'Busy'" />
    <button disabled aria-busy="true" class="outline">.</button>
  </fieldset>
  } @case ('success') {
  <input disabled aria-invalid="false" [value]="message() || 'Success'" />
  } @case ('error') {
  <input disabled aria-invalid="true" [value]="message() || 'Error'" />
  } }
</div>
```

```
export default class RegisterPage {
  #activityService = inject(ActivityService);
  feedback: WritableSignal<Feedback> = signal<Feedback>({ status: "idle",
message: "" });
  onRegister(register: Register) {
    this.feedback.set({ status: "busy", message: "Registering..." });
    this.authRepository.postRegister$(register).subscribe({
      next: () => this.feedback.set({ status: "success", message: "Register ok,
thanks for join." }),
      error: () => this.feedback.set({ status: "error", message: "Failed to
register. Review your data." }),
    });
  }
}
```

```
<lab-register (register)="onRegister($event)" />
<lab-feedback [feedback]="feedback()" />
```

8.3.2 Notificaciones de errores

`ng g s shared/state/notifications-store`

```
export type Notification = { message: string; type: "info" | "error" };
@Injectable({
  providedIn: "root",
})
export class NotificationsStore {
  #state: WritableSignal<Notification[]> = signal<Notification[]>([]);

  notifications: Signal<Notification[]> = this.#state.asReadonly();
  count: Signal<number> = computed(() => this.#state().length);

  addNotification(notification: Notification): void {
    this.#state.update((current) => [...current, notification]);
  }
  clearNotifications(): void {
    this.#state.set([]);
  }
}
```

`ng g class core/error.service`

```
export class ErrorService implements ErrorHandler {
  #notificationsStore: NotificationsStore = inject(NotificationsStore);
  handleError(error: any): void {
    const notification: Notification = { message: "An error occurred", type:
"error" };
    if (error instanceof HttpErrorResponse) {
      notification.message = error.message;
    } else {
      notification.message = error.toString();
    }
    this.#notificationsStore.addNotification(notification);
  }
}
export const appConfig: ApplicationConfig = {
  providers: [
    provideClientHydration(),
    provideHttpClient(withFetch(), withInterceptors([authInterceptor])),
    provideRouter(routes, withComponentInputBinding()),
    { provide: ErrorHandler, useClass: ErrorService },
  ],
};
```

`ng g c shared/ui/notifications`

```
export class NotificationsComponent {
  notifications: InputSignal<Notification[]> = input<Notification[]>([]);
  close = output();
}
```

```
<dialog open>
  <article>
    <header>
      <h2>Notifications</h2>
    </header>
    @for (notification of notifications(); track notification) { @if
(notification.type === 'error') {
    <input disabled aria-invalid="true" [value]="notification.message" />
    } @else {
    <input disabled aria-invalid="false" [value]="notification.message" />
    } }
    <footer>
      <button (click)="close.emit()">Close</button>
    </footer>
  </article>
</dialog>
```

```
export class FooterWidget {
  #notificationsStore: NotificationsStore = inject(NotificationsStore);
  showNotification: WritableSignal<boolean> = signal<boolean>(false);

  notifications: Signal<Notification[]> =
this.#notificationsStore.notifications;
  notificationsCount: Signal<number> = this.#notificationsStore.count;
  hasNotifications: Signal<boolean> = computed(() => this.notificationsCount() >
0);

  toggleNotifications(): void {
    this.showNotification.update((current) => !current);
  }
  onNotificationsClose(): void {
    this.showNotification.set(false);
    this.#notificationsStore.clearNotifications();
  }
}
```

```
<footer>
  <nav>
    <span>
      <a [href]="author.homepage" target="_blank">© {{ getYear() }} {{
author.name }}</a>
    </span>
    @if (hasNotifications()) {
    <button [attr.data-tooltip]="notificationsCount()"
(click)="toggleNotifications()" class="outline">🔥</button>
    }
    <span>
      @switch (cookiesStatus()) { @case ('pending') {
      <lab-cookies (cancel)="cookiesStatus.set('rejected')"
(accept)="cookiesStatus.set($event)" />
      } @case ('rejected') {
      <small data-tooltip="No cookies applied">🍪 ❌</small>
      } @case ('essentials') {
      <small data-tooltip="Essential cookies applied">🍪 ✅</small>
      } @case ('all') {
      <small data-tooltip="All cookies applied">🍪 ✅ ✅</small>
      } }
    </span>
  </nav>
</footer>
@if (showNotification()) {
<lab-notifications [notifications]="notifications()"
(close)="onNotificationsClose()" />
}
```