# **Error Handling Strategy Document**

## **For Angular + NgRx E-Commerce Application**

## **🎯 Objective**

To define a consistent, centralized, and user-friendly strategy for handling and displaying errors across the application, covering:

* HTTP/API errors
* NgRx effects and reducers
* Form and validation errors
* UI-level notifications

## **📌 Key Principles**

1. **Centralized handling**: Avoid redundant error handling in each service/component.
2. **User-friendly messages**: Never expose raw backend errors directly.
3. **Consistent state**: Errors should be tracked in NgRx for global access.
4. **Visual feedback**: Use alert, toast, or modal to inform users.

Component → Store (Action) → Effect (API call) → CatchError

↓

Dispatch Failure Action

↓

Reducer updates state

↓

Component UI subscribes to error$

**Global HTTP Error Interceptor**

Create HttpErrorInterceptor:

@Injectable()

export class HttpErrorInterceptor implements HttpInterceptor {

constructor(private store: Store) {}

intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

return next.handle(req).pipe(

catchError((error: HttpErrorResponse) => {

const message = this.mapError(error);

this.store.dispatch(setGlobalError({ error: message }));

return throwError(() => error);

}) ); }

private mapError(error: HttpErrorResponse): string {

if (error.status === 0) {

return 'Network error. Please check your connection.';

} else if (error.status === 401) {

return 'Unauthorized. Please login again.';

} else if (error.status === 404) {

return 'Resource not found.';

} else if (error.status === 500) {

return 'Server error. Please try later.';

}

return error.error?.message || 'Unexpected error occurred.';

}

}

**Example: Product Update**

updateProduct$ = createEffect(() =>

this.actions$.pipe(

ofType(updateProduct),

switchMap(({ product }) =>

this.productService.updateProduct(product).pipe(

map(updated => updateProductSuccess({ product: updated })),

catchError(error => of(updateProductFailure({ error: this.mapError(error) })))

)

)

)

);

**Global Error State**

**Actions**

export const setGlobalError = createAction('[App] Set Global Error', props<{ error: string }>());

export const clearGlobalError = createAction('[App] Clear Global Error');

**Reducer**

export const errorReducer = createReducer(

'',

on(setGlobalError, (\_, { error }) => error),

on(clearGlobalError, () => '')

);

**Selector**

export const selectGlobalError = createSelector(appState, state => state.error);

**UI Display Strategy**

this.store.select(selectGlobalError).subscribe(error => {

if (error) {

this.toastService.show(error, { classname: 'bg-danger text-light' });

this.store.dispatch(clearGlobalError());

}

});

Or

Bootstrap alert

<div \*ngIf="(globalError$ | async) as error" class="alert alert-danger mt-2">

{{ error }}

</div>

| **Layer** | **Responsibility** | **Strategy** |
| --- | --- | --- |
| Interceptor | Catch HTTP errors | Map errors & dispatch global error |
| Effects | Catch API failures | Dispatch specific failure actions |
| Reducers | Store error in state | Update state slice |
| Components | Show error messages | Subscribe to error$ selectors |
| Forms | Handle input validation | Angular reactive form validators |

## **Optional Enhancements**

* Add retry strategies with retryWhen
* Use ErrorBoundaryComponent for critical view errors
* Automatically log out on 401 Unauthorized
* Localize error messages (i18n)