Toggle Action Processor - Complete Documentation

Afficher l'image

Afficher l'image

Afficher l'image

Afficher l'image

Enterprise-grade HTTP request processor for toggle actions with loading states, success/error notifications, and comprehensive error handling. Framework-agnostic and battle-tested in production environments.

Table of Contents

API Documentation

- processToggleAction()
- processToggleActionSafe()
- showLoadingDialog()
- showSuccessDialog()
- showErrorDialog()

Configuration

- Configuration Options
- Dialog Customization
- HTTP Handler Configuration

Framework Integration

- Vanilla JavaScript
- jQuery
- React
- Vue.js
- Angular

Advanced Topics

- Error Handling
- Retry Logic
- Best Practices

Reference

- TypeScript Types
- Default Configurations
- Troubleshooting

Installation

NPM / Yarn

```bash
```

```
npm install @wlindabla/form_validator sweetalert2
# or
yarn add @wlindabla/form_validator sweetalert2
```

Dependencies

- **SweetAlert2** (for modals)
- **HTTP Handler** (your custom fetch/axios wrapper)

Import

```typescript
// ES6 Module
import {
    processToggleAction,
    showLoadingDialog,
    showSuccessDialog,
    showErrorDialog
} from '@wlindabla/form_validator';

// CommonJS
const {
    processToggleAction
} = require('@wlindabla/form_validator');
```

Basic Usage

Complete Example

```typescript
```

```
import { processToggleAction } from '@wlindabla/form_validator';
import Swal from 'sweetalert2';
import { httpFetchHandler } from './http-handler';

// 1. Listen for toggle confirmation event
document.addEventListener('account:toggle:confirmed', async (event: CustomEvent) => {
    try {
        const response = await processToggleAction({
            eventDetail: event.detail,
            httpHandler: httpFetchHandler,
            dialogHandler: Swal.fire,
            httpMethod: 'PATCH',
            retryCount: 2,

            onSuccess: async (data) => {
                console.log('Toggle successful:', data);
                // Reload page after 2 seconds
                setTimeout(() => location.reload(), 2000);
            },

            onError: (error) => {
                console.error('Toggle failed:', error);
            }
        });

        console.log('Response:', response);
    } catch (error) {
        console.error('Process failed:', error);
    }
});
```

HTTP Handler Example

typescript

```typescript
// http-handler.ts
export async function httpFetchHandler(options: {
  url: string;
  data: unknown;
  methodSend: string;
  retryCount: number;
}): Promise<HttpResponse<unknown>> {
  const { url, data, methodSend, retryCount } = options;

  for (let attempt = 0; attempt < retryCount; attempt++) {
    try {
      const response = await fetch(url, {
        method: methodSend,
        headers: {
          'Content-Type': 'application/json',
          'X-CSRF-Token': getCsrfToken()
        },
        body: JSON.stringify(data)
      });

      const result = await response.json();

      return {
        status: response.status,
        data: result,
        headers: response.headers
      };
    } catch (error) {
      if (attempt === retryCount - 1) throw error;
      await delay(1000 * (attempt + 1)); // Exponential backoff
    }
  }

  throw new Error('Max retries exceeded');
}

function getCsrfToken(): string {
  return document.querySelector<HTMLMetaElement>('meta[name="csrf-token"]')?.content || '';
}

function delay(ms: number): Promise<void> {
  return new Promise(resolve => setTimeout(resolve, ms));
}
```

Why Toggle Listener?

✨ Key Features

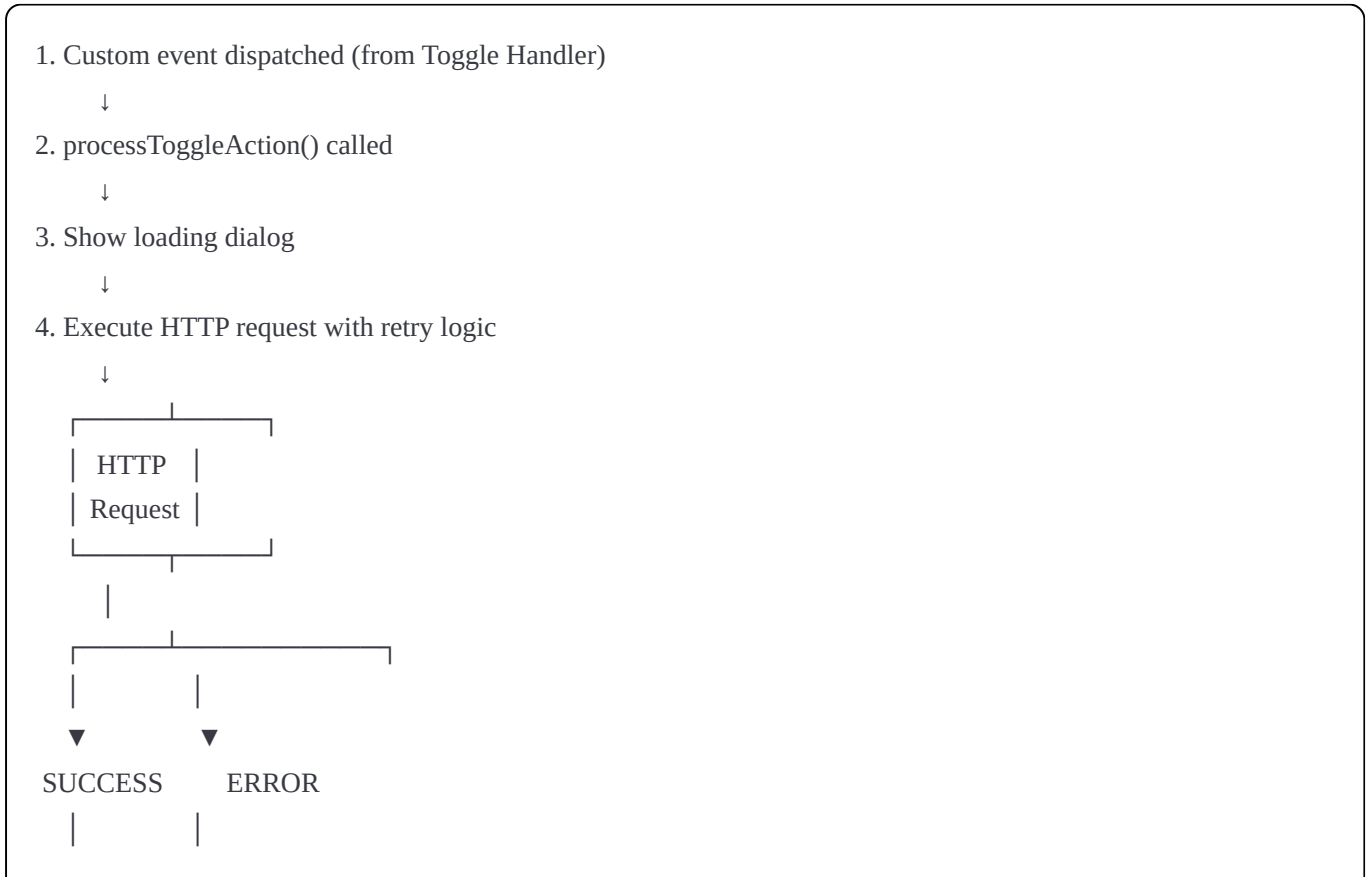
✅ **Automatic Loading States** - Shows loading dialog during HTTP requests
✅ **Success/Error Handling** - Displays appropriate notifications
✅ **Retry Logic** - Configurable retry count for failed requests
✅ **Framework-Agnostic** - Works with any JavaScript framework
✅ **Type-Safe** - Full TypeScript support
✅ **Extensible** - Callbacks for success/error/complete
✅ **Translation Ready** - Built-in i18n support
✅ **Error Detection** - Intelligent HTTP status mapping
✅ **Production Ready** - Battle-tested with comprehensive error handling

🎯 Use Cases

- ✅ Account activation/deactivation requests
- ✅ Feature enable/disable API calls
- ✅ Status change submissions
- ✅ Any POST/PATCH/PUT request requiring user feedback
- ✅ Long-running operations requiring loading indicators

How It Works

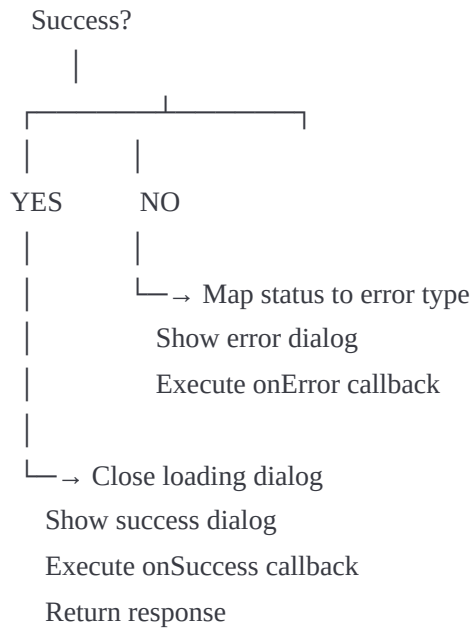
Process Flow



```
│              └─→ Show error dialog
│                  Call onError callback
│                  Throw error
│
└─→ Show success dialog
    Call onSuccess callback
    Return response
      │
      ▼
Call onComplete callback (always)
```

Detailed Flow Diagram

```
┌──────────────────────────────┐
│ Toggle Confirmed             │
│ Custom Event                 │
│                              │
│ event.detail = {             │
│   data,                      │
│   url_action_confirm         │
│ }                            │
└──────────────┬───────────────┘
               │
               ▼
┌──────────────────────────────┐
│ Close Previous Modal         │
└──────────────┬───────────────┘
               │
               ▼
┌──────────────────────────────┐
│ Show Loading Dialog          │
│  - Spinner                   │
│  - Progress bar              │
│  - "Processing..."           │
└──────────────┬───────────────┘
               │
               ▼
┌──────────────────────────────┐
│ Execute HTTP                 │
│ Request                      │
│                              │
│  - URL from detail           │
│  - Data from detail          │
│  - Retry logic               │
└──────────────┬───────────────┘
               │
```



Server Requirements

CRITICAL: Server-Side Configuration

Your server **MUST** meet these requirements:

1. Accept XMLHttpRequest

```php
// Symfony example
if (!$request->isXmlHttpRequest()) {
    return new JsonResponse(['error' => 'Invalid request'], 400);
}
```

2. Always Return JSON

```php
```

```php
// ✅ CORRECT - Always return JSON
public function toggleAction(Request $request): JsonResponse
{
    try {
        // ... toggle logic ...

        return new JsonResponse([
            'title' => 'Success',
            'message' => 'Account activated successfully'
        ], 200);

    } catch (\Exception $e) {
        // Even on error, return JSON
        return new JsonResponse([
            'title' => 'Error',
            'message' => $e->getMessage()
        ], 500);
    }
}

// ❌ WRONG - Don't return HTML
public function toggleAction(Request $request): Response
{
    return $this->render('success.html.twig'); // Won't work!
}
```

3. Standard Response Format

typescript

```
// Expected response format
interface ServerResponse {
    title: string;    // Dialog title
    message: string;  // Dialog message
    // ... additional fields optional
}

// Success response (2xx status)
{
    "title": "Success",
    "message": "Account has been activated successfully"
}

// Error response (4xx, 5xx status)
{
    "title": "Error",
    "message": "Failed to activate account: User not found"
}
```

4. Example Implementations

Symfony:

```
php
```

```php
namespace App\Controller\Admin;

use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;

class UserController
{
    public function toggleAction(Request $request, int $id): JsonResponse
    {
        // Validate XMLHttpRequest
        if (!$request->isXmlHttpRequest()) {
            return new JsonResponse([
                'title' => 'Invalid Request',
                'message' => 'Only AJAX requests are allowed'
            ], 400);
        }

        try {
            $user = $this->userRepository->find($id);

            if (!$user) {
                return new JsonResponse([
                    'title' => 'Not Found',
                    'message' => 'User not found'
                ], 404);
            }

            // Toggle logic
            $newStatus = !$user->isEnabled();
            $user->setEnabled($newStatus);
            $this->entityManager->flush();

            return new JsonResponse([
                'title' => 'Success',
                'message' => sprintf(
                    'Account %s successfully',
                    $newStatus ? 'activated' : 'deactivated'
                )
            ], 200);

        } catch (\Exception $e) {
            return new JsonResponse([
                'title' => 'Server Error',
                'message' => 'An error occurred: ' . $e->getMessage()
            ], 500);
        }
```

```
    }
}
```

Laravel:

```php
```

```php
namespace App\Http\Controllers\Admin;

use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;

class UserController extends Controller
{
    public function toggle(Request $request, int $id): JsonResponse
    {
        // Validate AJAX request
        if (!$request->ajax()) {
            return response()->json([
                'title' => 'Invalid Request',
                'message' => 'Only AJAX requests are allowed'
            ], 400);
        }

        try {
            $user = User::findOrFail($id);

            // Toggle logic
            $user->enabled = !$user->enabled;
            $user->save();

            return response()->json([
                'title' => 'Success',
                'message' => sprintf(
                    'Account %s successfully',
                    $user->enabled ? 'activated' : 'deactivated'
                )
            ], 200);

        } catch (ModelNotFoundException $e) {
            return response()->json([
                'title' => 'Not Found',
                'message' => 'User not found'
            ], 404);

        } catch (\Exception $e) {
            return response()->json([
                'title' => 'Server Error',
                'message' => 'An error occurred: ' . $e->getMessage()
            ], 500);
        }
```

```
    }
}
```

Express.js:

```javascript
```

```
const express = require('express');
const router = express.Router();

router.post('/users/:id/toggle', async (req, res) => {
    // Validate AJAX request
    if (!req.xhr) {
        return res.status(400).json({
            title: 'Invalid Request',
            message: 'Only AJAX requests are allowed'
        });
    }

    try {
        const user = await User.findById(req.params.id);

        if (!user) {
            return res.status(404).json({
                title: 'Not Found',
                message: 'User not found'
            });
        }

        // Toggle logic
        user.enabled = !user.enabled;
        await user.save();

        res.status(200).json({
            title: 'Success',
            message: `Account ${user.enabled ? 'activated' : 'deactivated'} successfully`
        });

    } catch (error) {
        res.status(500).json({
            title: 'Server Error',
            message: `An error occurred: ${error.message}`
        });
    }
});

module.exports = router;
```

API Documentation

processToggleAction()

Main function that processes toggle actions with HTTP requests.

```typescript
async function processToggleAction(
  params: ProcessToggleActionParams
): Promise<HttpResponse<unknown>>
```

Parameters:

```typescript
interface ProcessToggleActionParams {
  eventDetail: ToggleEventDetail;        // Required
  httpHandler: HttpHandler;              // Required
  dialogHandler: DialogHandler;          // Required (Swal.fire)
  translator?: Translator | null;        // Optional
  loadingConfig?: Partial<SweetAlertOptions>;   // Optional
  successConfig?: Partial<SweetAlertOptions>;   // Optional
  errorConfig?: Partial<SweetAlertOptions>;     // Optional
  httpMethod?: string;                   // Default: 'PATCH'
  retryCount?: number;                   // Default: 2
  onSuccess?: (data, eventDetail) => void | Promise<void>;  // Optional
  onError?: (error, eventDetail) => void | Promise<void>;   // Optional
  onComplete?: (eventDetail) => void | Promise<void>;       // Optional
}
```

Returns: `Promise<HttpResponse<unknown>>`

Throws: Re-throws error after showing error dialog

Example:

```typescript
```

```
try {
    const response = await processToggleAction({
        eventDetail: event.detail,
        httpHandler: fetchHandler,
        dialogHandler: Swal.fire,

        // Optional: Translation
        translator: (key, error, lang) => {
            if (error) {
                return errorTranslator.translate(error.name, error, lang);
            }
            return translations[key] || key;
        },

        // Optional: Custom dialogs
        loadingConfig: {
            background: '#fff',
            color: '#333'
        },

        successConfig: {
            timer: 3000,
            toast: true
        },

        errorConfig: {
            showConfirmButton: true
        },

        // Optional: HTTP config
        httpMethod: 'PATCH',
        retryCount: 3,

        // Optional: Callbacks
        onSuccess: async (data, eventDetail) => {
            console.log('Success:', data);

            // Update UI
            updateButton(eventDetail.sourceElement, data.status);

            // Analytics
            analytics.track('toggle_success');

            // Reload after delay
            setTimeout(() => location.reload(), 2000);
        },
```

```
    onError: (error, eventDetail) => {
      console.error('Error:', error);

      // Send to error tracking
      errorTracker.captureException(error);

      // Analytics
      analytics.track('toggle_error');
    },

    onComplete: (eventDetail) => {
      console.log('Process complete');

      // Cleanup
      removeLoadingIndicators();
    }
  });

  console.log('Response:', response);

} catch (error) {
  // Error already shown to user
  console.error('Toggle action failed:', error);
}
```

---

processToggleActionSafe()

Safe version that doesn't throw errors.

typescript

```
async function processToggleActionSafe(
  params: ProcessToggleActionParams
): Promise<{
  success: boolean;
  response?: HttpResponse<unknown>;
  error?: Error | HttpResponse<unknown>;
}>
```

Example:

typescript

```
const result = await processToggleActionSafe({
    eventDetail: event.detail,
    httpHandler: fetchHandler,
    dialogHandler: Swal.fire
});

if (result.success) {
    console.log('Response:', result.response);
} else {
    console.error('Error:', result.error);
}
```

---

showLoadingDialog()

Shows a loading dialog with progress indication.

```typescript
function showLoadingDialog(
    options: ShowLoadingDialogOptions
): TimerWrapper
```

Parameters:

```typescript
interface ShowLoadingDialogOptions {
    dialogHandler: DialogHandler;
    translator?: Translator | null;
    config?: Partial<SweetAlertOptions>;
}
```

Returns: `{ timerInterval?: NodeJS.Timeout }`

Example:

```typescript
```

```
const { timerInterval } = showLoadingDialog({
    dialogHandler: Swal.fire,
    translator: (key) => translations[key],
    config: {
        title: 'Custom Loading...',
        timer: 30000
    }
});

// Later: cleanup
if (timerInterval) clearInterval(timerInterval);
```

---

showSuccessDialog()

Shows a success dialog.

typescript

```
function showSuccessDialog(
    options: ShowSuccessDialogOptions
): Promise<SweetAlertResult>
```

Parameters:

typescript

```
interface ShowSuccessDialogOptions {
    dialogHandler: DialogHandler;
    title: string;
    message: string;
    config?: Partial<SweetAlertOptions>;
}
```

Example:

typescript

```typescript
await showSuccessDialog({
    dialogHandler: Swal.fire,
    title: 'Success!',
    message: 'Operation completed successfully',
    config: {
        timer: 3000,
        toast: true,
        position: 'top-end'
    }
});
```

---

showErrorDialog()

Shows an error dialog.

```typescript
function showErrorDialog(
    options: ShowErrorDialogOptions
): Promise<SweetAlertResult>
```

Parameters:

```typescript
interface ShowErrorDialogOptions {
    dialogHandler: DialogHandler;
    title: string;
    message: string;
    config?: Partial<SweetAlertOptions>;
}
```

Example:

```typescript
await showErrorDialog({
    dialogHandler: Swal.fire,
    title: 'Error',
    message: 'Operation failed. Please try again.',
    config: {
        confirmButtonText: 'Retry'
    }
});
```

Configuration Options

Default Loading Dialog

typescript

```typescript
const DEFAULT_LOADING_DIALOG_CONFIG = {
    icon: "info",
    allowOutsideClick: false,
    showConfirmButton: false,
    timer: 50000,
    timerProgressBar: true,
    customClass: {
        loader: "spinner-border text-info",
        timerProgressBar: "bg-info"
    }
};
```

Default Success Dialog

typescript

```typescript
const DEFAULT_SUCCESS_DIALOG_CONFIG = {
    icon: "success",
    timer: 40000,
    timerProgressBar: true,
    showConfirmButton: false,
    showCloseButton: true
};
```

Default Error Dialog

typescript

```typescript
const DEFAULT_ERROR_DIALOG_CONFIG = {
    icon: "error",
    timer: 30000,
    showConfirmButton: true,
    confirmButtonText: "OK",
    showCloseButton: true
};
```

Framework Integration

Vanilla JavaScript Integration

typescript

```typescript
import { processToggleAction } from '@wlindabla/form_validator';
import Swal from 'sweetalert2';
import { httpFetchHandler } from './http-handler';

// Initialize listener
document.addEventListener('DOMContentLoaded', () => {
    initToggleListener();
});

function initToggleListener() {
    document.addEventListener('account:toggle:confirmed', async (event: CustomEvent) => {
        try {
            await processToggleAction({
                eventDetail: event.detail,
                httpHandler: httpFetchHandler,
                dialogHandler: Swal.fire,
                httpMethod: 'PATCH',
                retryCount: 2,
                onSuccess: async (data) => {
                    console.log('Toggle successful');
                    setTimeout(() => location.reload(), 2000);
                }
            });
        } catch (error) {
            console.error('Toggle failed:', error);
        }
    });
}
```

jQuery Integration

typescript

```
import { processToggleAction } from '@wlindabla/form_validator';
import Swal from 'sweetalert2';

jQuery(document).ready(function($) {
    $(document).on('account:toggle:confirmed', async function(event) {
        const detail = event.originalEvent.detail;

        try {
            await processToggleAction({
                eventDetail: detail,
                httpHandler: jQueryAjaxHandler,
                dialogHandler: Swal.fire,
                httpMethod: 'POST',
                onSuccess: (data) => {
                    console.log('Success:', data);
                    setTimeout(() => location.reload(), 2000);
                }
            });
        } catch (xhr) {
            console.error('Failed:', xhr);
        }
    });
});

// jQuery AJAX handler
async function jQueryAjaxHandler(options) {
    const { url, data, methodSend, retryCount } = options;

    for (let attempt = 0; attempt < retryCount; attempt++) {
        try {
            const response = await jQuery.ajax({
                url,
                method: methodSend,
                data: JSON.stringify(data),
                contentType: 'application/json',
                dataType: 'json'
            });

            return {
                status: 200,
                data: response
            };
        } catch (xhr) {
            if (attempt === retryCount - 1) throw xhr;
            await new Promise(resolve => setTimeout(resolve, 1000 * (attempt + 1)));
        }
```

```
  }
}
```

React Integration

typescript

```tsx
import { useEffect } from 'react';
import { processToggleAction } from '@wlindabla/form_validator';
import Swal from 'sweetalert2';

export function useToggleListener(eventName: string, onSuccess?: () => void) {
    useEffect(() => {
        const handleToggleEvent = async (event: Event) => {
            const customEvent = event as CustomEvent;

            try {
                await processToggleAction({
                    eventDetail: customEvent.detail,
                    httpHandler: fetchHandler,
                    dialogHandler: Swal.fire,
                    onSuccess: async (data) => {
                        console.log('Success:', data);
                        if (onSuccess) onSuccess();
                    }
                });
            } catch (error) {
                console.error('Toggle failed:', error);
            }
        };

        document.addEventListener(eventName, handleToggleEvent);

        return () => {
            document.removeEventListener(eventName, handleToggleEvent);
        };
    }, [eventName, onSuccess]);
}

// Component usage
function UserManagement() {
    const [users, setUsers] = useState([]);

    useToggleListener('account:toggle:confirmed', () => {
        // Refresh user list after successful toggle
        fetchUsers();
    });

    return <UserList users={users} />;
}
```

Vue.js Integration

vue

```
<script setup lang="ts">
import { onMounted, onUnmounted } from 'vue';
import { processToggleAction } from '@wlindabla/form_validator';
import Swal from 'sweetalert2';

const emit = defineEmits<{
    success: [];
}>();

let cleanup: (() => void) | null = null;

onMounted(() => {
    const handleToggleEvent = async (event: Event) => {
        const customEvent = event as CustomEvent;

        try {
            await processToggleAction({
                eventDetail: customEvent.detail,
                httpHandler: fetchHandler,
                dialogHandler: Swal.fire,
                onSuccess: () => {
                    emit('success');
                }
            });
        } catch (error) {
            console.error('Toggle failed:', error);
        }
    };

    document.addEventListener('account:toggle:confirmed', handleToggleEvent);

    cleanup = () => {
        document.removeEventListener('account:toggle:confirmed', handleToggleEvent);
    };
});

onUnmounted(() => {
    if (cleanup) cleanup();
});
</script>
```

Angular Integration

typescript

```
import { Injectable, OnDestroy } from '@angular/core';
import { processToggleAction } from '@wlindabla/form_validator';
import Swal from 'sweetalert2';
import { Subject } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ToggleListenerService implements OnDestroy {
  private toggleSuccess$ = new Subject<any>();
  private toggleError$ = new Subject<any>();
  private boundHandler: ((event: Event) => void) | null = null;

  constructor() {
    this.initListener();
  }

  private initListener(): void {
    this.boundHandler = async (event: Event) => {
      const customEvent = event as CustomEvent;

      try {
        await processToggleAction({
          eventDetail: customEvent.detail,
          httpHandler: fetchHandler,
          dialogHandler: Swal.fire,
          onSuccess: (data) => {
            this.toggleSuccess$.next(data);
          },
          onError: (error) => {
            this.toggleError$.next(error);
          }
        });
      } catch (error) {
        this.toggleError$.next(error);
      }
    };

    document.addEventListener('account:toggle:confirmed', this.boundHandler);
  }

  get onToggleSuccess$() {
    return this.toggleSuccess$.asObservable();
  }

  get onToggleError$() {
```

```
    return this.toggleError$.asObservable();
  }

  ngOnDestroy(): void {
    if (this.boundHandler) {
      document.removeEventListener('account:toggle:confirmed', this.boundHandler);
    }
    this.toggleSuccess$.complete();
    this.toggleError$.complete();
  }
}
```

Error Handling

HTTP Response Errors

typescript

```
try {
  await processToggleAction({
    eventDetail: event.detail,
    httpHandler: fetchHandler,
    dialogHandler: Swal.fire
  });
} catch (error) {
  if (error instanceof HttpResponse) {
    console.error(`HTTP ${error.status}:`, error.data);
    // Error already shown to user via dialog
  }
}
```

Network Errors

typescript

```
try {
    await processToggleAction({
        eventDetail: event.detail,
        httpHandler: fetchHandler,
        dialogHandler: Swal.fire,
        translator: (key, error, lang) => {
            // Use FetchErrorTranslator for network errors
            if (error) {
                return fetchErrorTranslator.translate(error.name, error, lang);
            }
            return translations[key];
        }
    });
} catch (error) {
    if (error instanceof Error) {
        // Network error (timeout, abort, etc.)
        // Already translated and shown to user
        console.error('Network error:', error);
    }
}
```

Best Practices

1. Always Handle Callbacks

typescript

```
// ✅ Good
await processToggleAction({
    eventDetail: event.detail,
    httpHandler: fetchHandler,
    dialogHandler: Swal.fire,

    onSuccess: async (data) => {
        // Update UI, reload, etc.
        await updateUI(data);
    },

    onError: (error) => {
        // Log, track, notify
        errorTracker.capture(error);
    },

    onComplete: () => {
        // Cleanup
        cleanup();
    }
});

// ❌ Bad - No callbacks
await processToggleAction({
    eventDetail: event.detail,
    httpHandler: fetchHandler,
    dialogHandler: Swal.fire
});
```

2. Use Translation Function

typescript

```
// ✅ Good
await processToggleAction({
    eventDetail: event.detail,
    httpHandler: fetchHandler,
    dialogHandler: Swal.fire,
    translator: (key, error, lang) => {
        if (error) {
            return fetchErrorTranslator.translate(error.name, error, lang);
        }
        return translations[key] || key;
    }
});
```

3. Implement Retry Logic

```typescript
// ✅ Good
await processToggleAction({
    eventDetail: event.detail,
    httpHandler: fetchHandler,
    dialogHandler: Swal.fire,
    retryCount: 3 // Try 3 times before failing
});
```

TypeScript Types

```typescript
```

```typescript
// Event detail
interface ToggleEventDetail {
    data: {
        status: boolean;
        [key: string]: unknown;
    };
    url_action_confirm: string;
    sourceElement?: HTMLElement;
    timestamp?: string;
}

// HTTP handler
type HttpHandler = (options: {
    url: string;
    data: unknown;
    methodSend: string;
    retryCount: number;
}) => Promise<HttpResponse<unknown>>;

// Dialog handler
type DialogHandler = typeof Swal.fire & {
    close?: () => void;
    getPopup?: () => HTMLElement | null;
    getTimerLeft?: () => number;
};

// Translator
type Translator = (
    key: string,
    error?: Error | null,
    language?: string
) => string;
```

Troubleshooting

Issue: "Server returns HTML instead of JSON"

Solution: Ensure your server always returns JSON. See Server Requirements.

Issue: "Loading dialog doesn't close"

Solution: Ensure `dialogHandler.close()` is available or catch errors properly.

Issue: "Success dialog shows wrong message"

Solution: Ensure server returns `{ title, message }` format.

Made with ❤ by AGBOKOUDJO Franck

⬆ Back to Top