SignaturGruppen

Frontend guide.

for service providers

Version 0.2.1 2020

Table of Contents

Introduction

This document describes a best practice example on initiating Nets eID Broker as redirects and pop-ups.

It is intended as a ‘getting started guide’ and not the one and only way of implementing it on your platform.

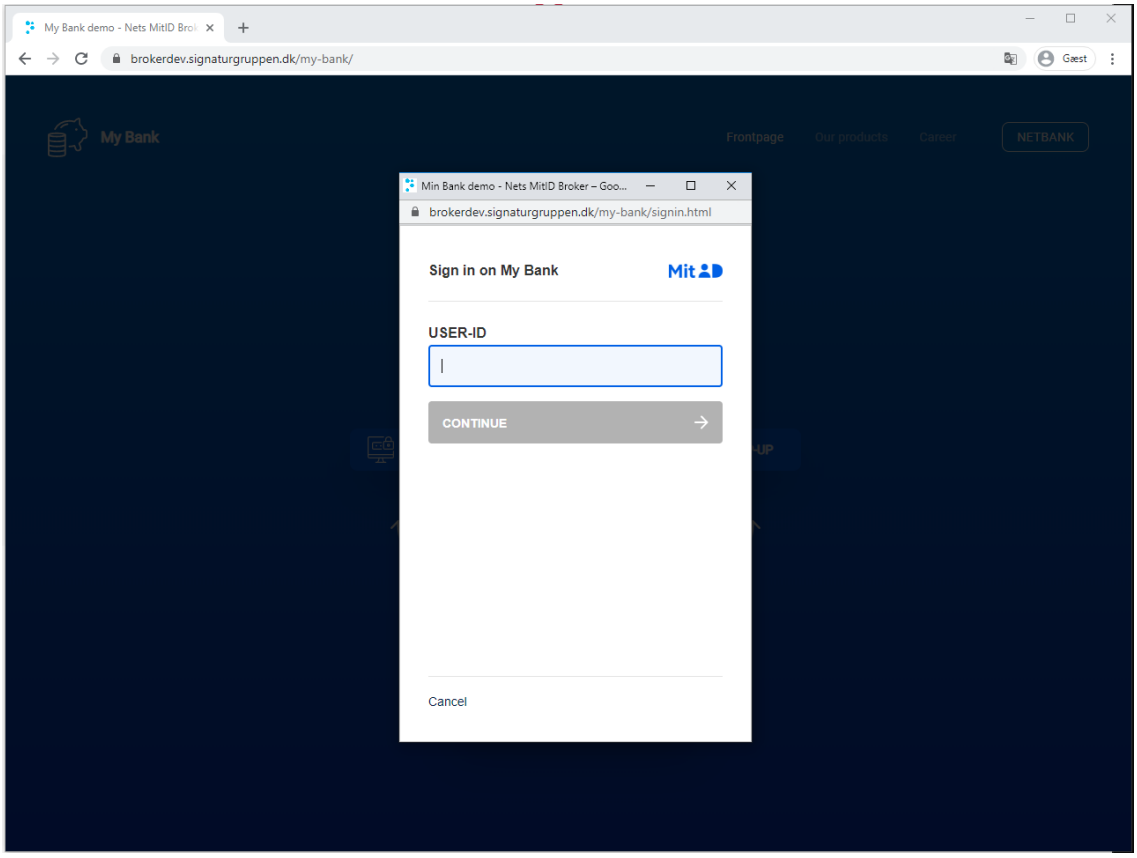
It’s possible to display the Nets eID Broker to the user with either a pop-up or redirecting the browser to Nets eID Broker. They are equally secure ways to authenticate the end-user. The user experience is quite different, and the decision is up to you.

The intended audiences are frontend developers and UX designers.

Pop-up UX assessment

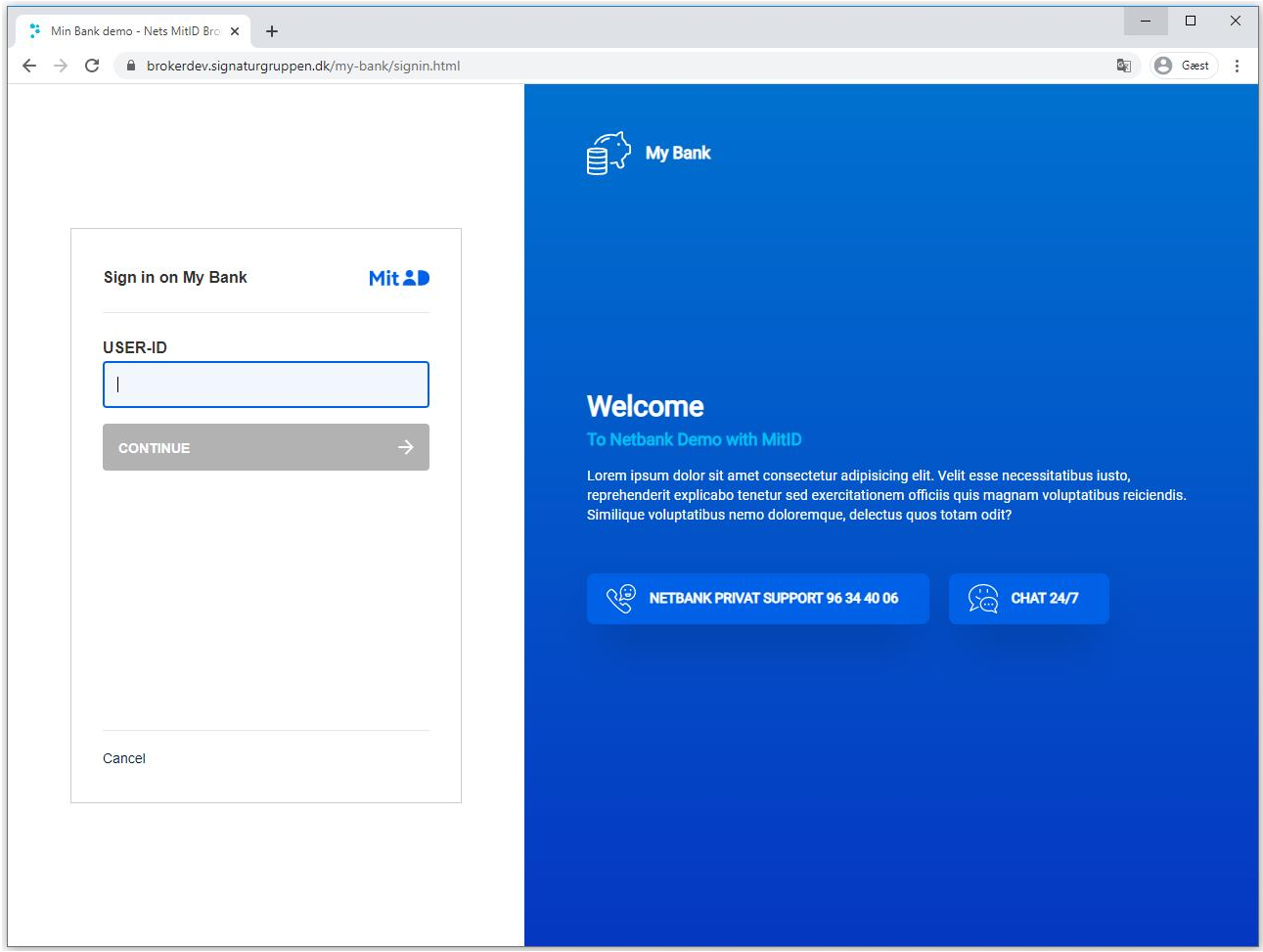
- Your existing page branding is visible behind the pop-up when displayed on PC. When used on mobile devices (including iPad) the popup opens in a new tab.
- If you are using it on a SPA (single page application), you can ensure your JavaScript state is intact, since the browser doesn't need to navigate away from the SPA.

  With the pop-up the user has an extra browser window which incidentally can be closed or set the focus to the parent page. It's recommended to handle those scenarios also to ensure a great user experience.
- Opening the pop-up with default size, might cause scrollbars if users with low vision resizes the text size.
- Some flows might require more space and are not suited for pop-ups.



Redirect UX assessment

- The user can bookmark the page and go to that page directly to make a login.
- It can be initialized automatically on a page load. This makes it possible to have an absolute URL on your domain e.g. mydomain.dk/login that redirects to Nets eID Brokers login. With the pop-up you need the onclick event, to ensure the pop-up is opened correct.
- Easier implementation with less JavaScript event handling.



Comparing the NemID and the MitID user experience

To introduce the Nets eID Broker client user experience, we start by viewing the existing NemID solution.

The NemID JavaScript client is integrated with the Service Provider's page using an <iframe> element, which enables a web page to allocate a segment of its area to another page.

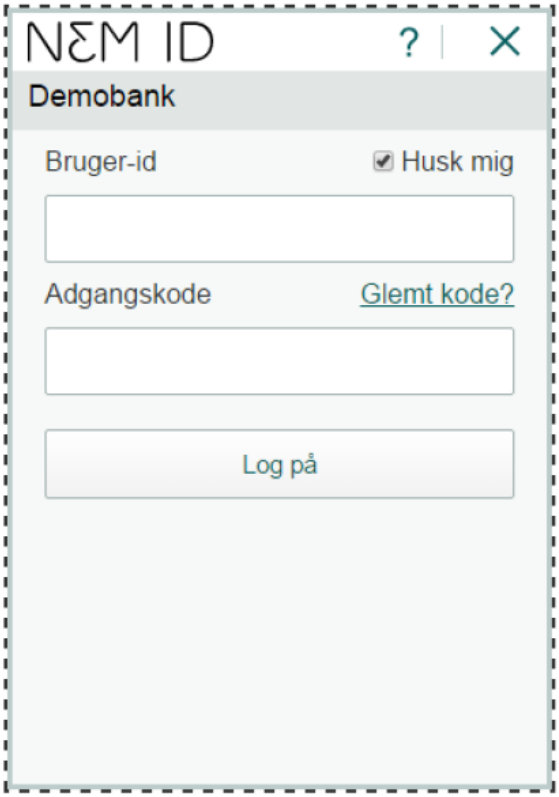


Figure 1 - The NemID JavaScript client in limited mode 320*460 with the recommended option of "Remember userid". The dashed line indicates the extent of the iframe.

The Nets eID Broker client

The Nets eID Broker has a large margin of freedom to customize the user experience. NEB will tailor the experience to the needs of the customers to the largest extends in bilateral dialogue between the partners.

As an example, it is possible to have the Service Provider send in user ID and preferred logon device as part of the initialization and by this help to adapt the user experience to the SPs users and their knowledge of their users possible means of identifying themselves.

The UI can be customized with colors, fonts, logos, help-text and other visual elements. The UI is responsive, and scales well on both small and large displays.

The image below illustrates the logon concept where the UI is shown as a pop–up over the service provider's web application:

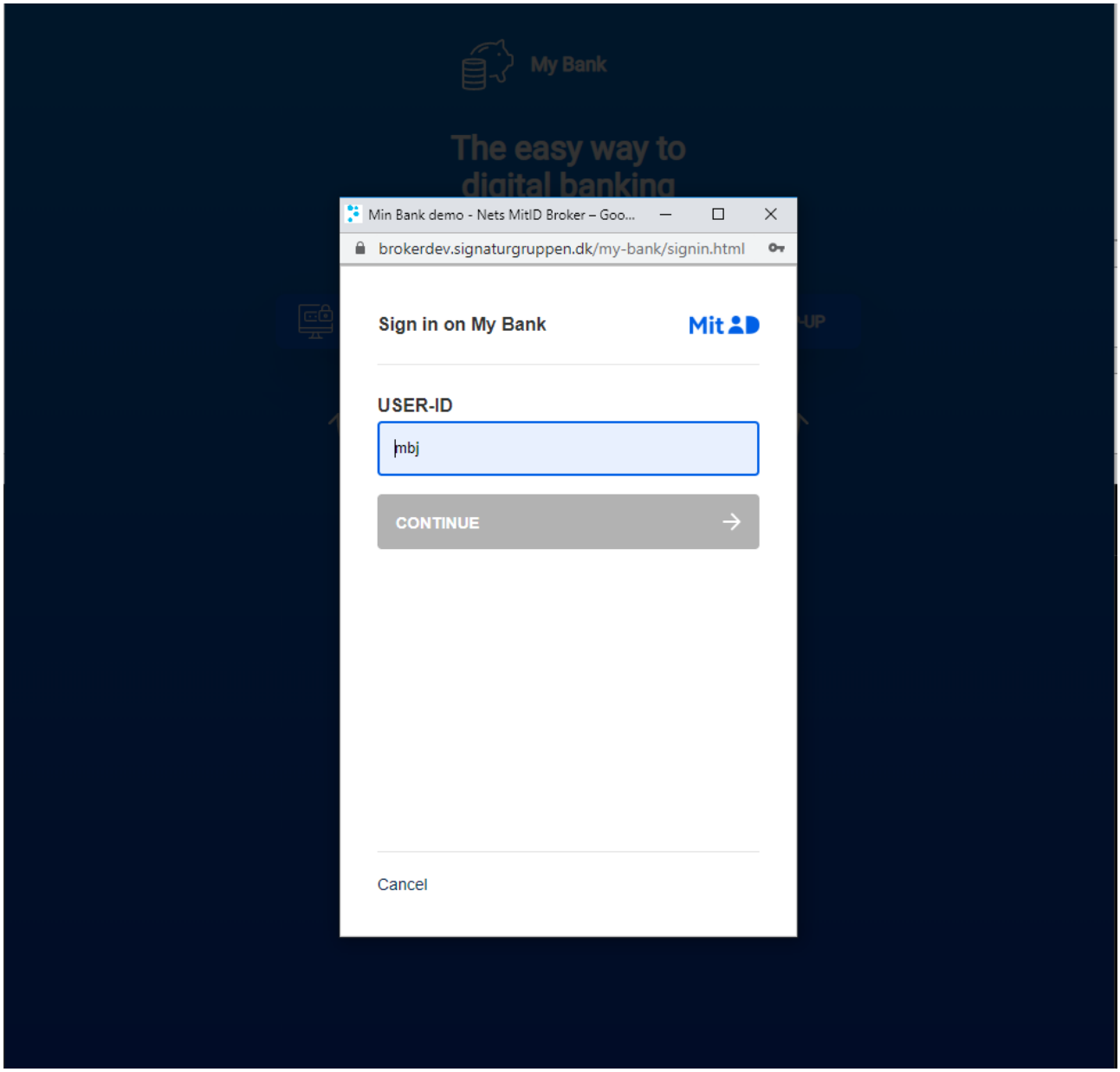


Figure 2 - The Nets eID Broker client exemplified as a pop-up overlay.

The Service Provider can customize the user experience through individual configuration and by setting parameters in the actual online authentication request.

Outside the MitID Secure Frame where will be high degree of flexibility on UI. Within the MitID Secure Frame the guidelines in MitID UX scheme will define the look and feel of MitID, but there will be some flexibility with reference text. Other elements of flexibility known from the NemID JS Client will be continued in the Nets eID Broker Client.

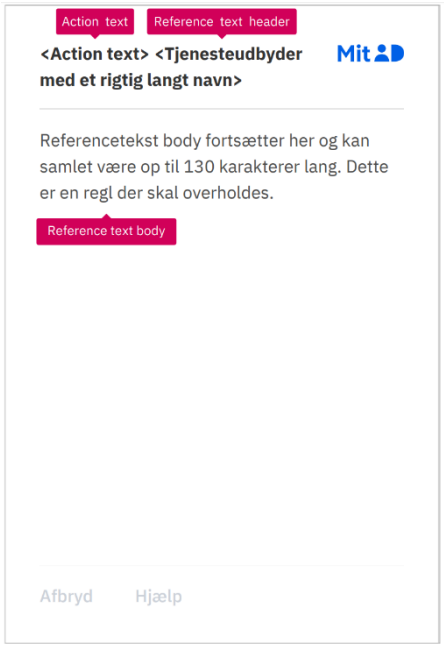


Figure 3: Reference texts of the Nets eID Broker client.

For the purpose of dialogue, an example is given below of how a transaction signing might be experienced by the customer. Further descriptions of these flows are given in Appendix A and Appendix B.

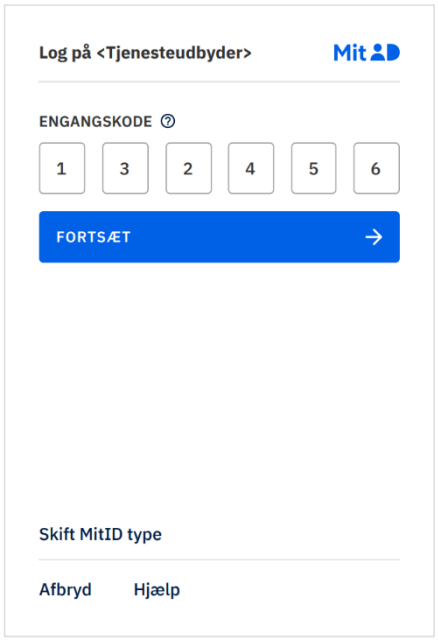


Figure 4: Example transaction signing in the Nets eID Broker client

Mobile app integration

Integrating with NEB from mobile apps (iOS and Android) is done using the OpenID Connect protocol in the same manner, as when integrating with standard web applications.

This section describes the technical integration from mobile apps as well as explains how to integrate the OIDC flow and standard browser flows into mobile apps while maintaining security, user-flow control and desired user-experience (UX).

When authenticating the user with NEB from a mobile app, the systems default browser should be used. This can be done with a tight and seamless way where switching to the browser and back to the app is handled in a natural way.

In broad terms, the following is implemented:

- iOS Universal Links and Android App Links
- In-app browser tab pattern for both iOS and Android
- OpenID Connect authorization Code Flow + PKCE
- App backend handles all OIDC flows directly with the NEB API endpoints using a secure client

iOS Universal Links and Android App Links

Universal Links and App Links is the way the iOS and Android platforms allow apps to register and control specific URLs at system level and allowing seamless switching between app and browser.

When a browser on iOS and Android redirects the user to an app-controlled URL by these mechanisms – the user is automatically moved to the controlling app, allowing flows to switch between browser and app.

In-app browser tab pattern for both iOS and Android

This feature allows the app to start the system default browser using an in-app browser tab supporting all the required features needed to complete the authorization flow (WebAuthn, user display of address bar etc.).

Both iOS and Android support this in a way, that lets the app integrate the browser experience in their app and allows the “look and feel” to stay in the app – thus creating a user experience of not leaving the app.

There is equal control over UX and visual integration into the app using in-app browser tabs vs. using a webview. Using the in-app browser tab approach the app gains performance, better support for various frameworks like WebAuthn (needed for MitID FIDO and other FIDO security keys, not available using Webviews).

OpenID Connect Authorization Code flow + PKCE

Utilizing OIDC authorization Code Flow + PKCE allows a strong and secure integration to OIDC on mobile app platforms.

PKCE is an extension to the Authorization Code flow which ensures that only the initiating application, will be able to exchange the received authorization code to tokens.

Read more here: https://openid.net/2015/05/26/enhancing-oauth-security-for-mobile-applications-with-pkse/.

App backend handles all OIDC flows directly with the NEB API endpoints using a secure client

Only allowing the app backend to request the signed tokens and access the APIs on the NEB platform using a secure client strengthens the security of the setup. It is highly recommended, that both web- and app applications utilizes this approach to have no OIDC client configured client-side, but have the backend setup and handle all the specific integrations (except opening the browser etc.)

App switching to MitID app

App switching the MitID app in the UX flows is handled by the web flow using a combination and JavaScript detection and redirecting the user. It is expected that the MitID app supports control over specific URLs to handle switching this way. If the MitID app does not support this, but only supports app-to-app switching – support for this will be added and described.

Vanilla JavaScript pop-up handling

To create a pop-up, you need to implement some JavaScript functions. We have made our examples with pure vanilla JavaScript to accommodate most use cases. We encourage you to transform the code examples to your specific needs and technology stack.

Some users might have disabled JavaScript. So, you need a strategy on how to handle those users: either ask your users to enable it or make an noscript failover solution.

Initiating the pop-up

```
function popupwindow(url, title, w, h) {
  //adding the selector for the css backdrop
  document.body.classList.add("popup-active");
  //automatic centering of the pop-up.
  let y = window.outerHeight / 2 + window.screenY - h / 2;
  let x = window.outerWidth / 2 + window.screenX - w / 2;
  wref = window.open(
    url,
    title,
    "toolbar=no, location=no, directories=no, status=no, menubar=no, scrollbars=no, resizable=no,
 copyhistory=no, width=" +
      w +
      ", height=" +
      h +
      ", top=" +
      y +
      ", left=" +
      x
  );
  //ensuring focus of the new pop-up
  wref.focus();
}
```

channelmode=yes|no|1|0	Whether or not to display the window in theater mode. Default is no. IE only
directories=yes|no|1|0	Obsolete. Whether or not to add directory buttons. Default is yes. IE only
fullscreen=yes|no|1|0	Whether or not to display the browser in full-screen mode. Default is no. A window in full-screen mode must also be in

	theater mode. IE only
height=pixels	The height of the window. Min. value is 100
left=pixels	The left position of the window. Negative values not allowed
location=yes|no|1|0	Whether or not to display the address field. Opera only
menubar=yes|no|1|0	Whether or not to display the menu bar
resizable=yes|no|1|0	Whether or not the window is resizable. IE only
scrollbars=yes|no|1|0	Whether or not to display scroll bars. IE, Firefox & Opera only
status=yes|no|1|0	Whether or not to add a status bar
titlebar=yes|no|1|0	Whether or not to display the title bar. Ignored unless the calling application is an HTML Application or a trusted dialog box
toolbar=yes|no|1|0	Whether or not to display the browser toolbar. IE and Firefox only
top=pixels	The top position of the window. Negative values not allowed
width=pixels	The width of the window. Min. value is 100

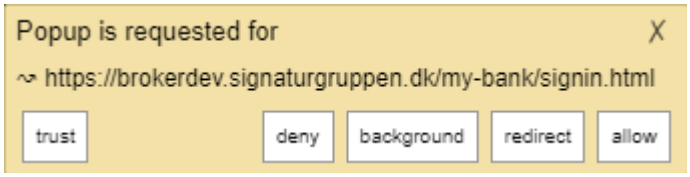
Pop-up blocker compliance

Most browsers block pop-ups if they are called outside of user-triggered event handlers like onclick. Therefor it's important to initiate the pop-up with onclick event. Some very aggressive pop-up blockers can still catch the pop-up.

We have tested this implementation with the following setups.

Windows 10	Chrome	Built-in pop-up blocker
Windows 10	Chrome	Popup blocker (strict), Chrome Extension*
Windows 10	Chrome	Popper blocker, Chrome Extension
Windows 10	Edge	Built-in pop-up blocker
Windows 10	Internet Explorer 11	Default settings
Windows 10	Firefox	Built-in pop-up blocker
Windows 10	Opera	Built-in pop-up blocker
Windows 10	Yandex	Built-in pop-up blocker
MacOS Catalina	Safari 13	Built-in pop-up blocker
MacOS Mojave	Safari 12.1	Built-in pop-up blocker
MacOS Mojave	Chrome	Built-in pop-up blocker
MacOS Mojave	Firefox	Built-in pop-up blocker
MacOS Mojave	Edge	Built-in pop-up blocker

** with Chrome Extension, Popup blocker (strict), the user gets this dialog prompt.*



Failover

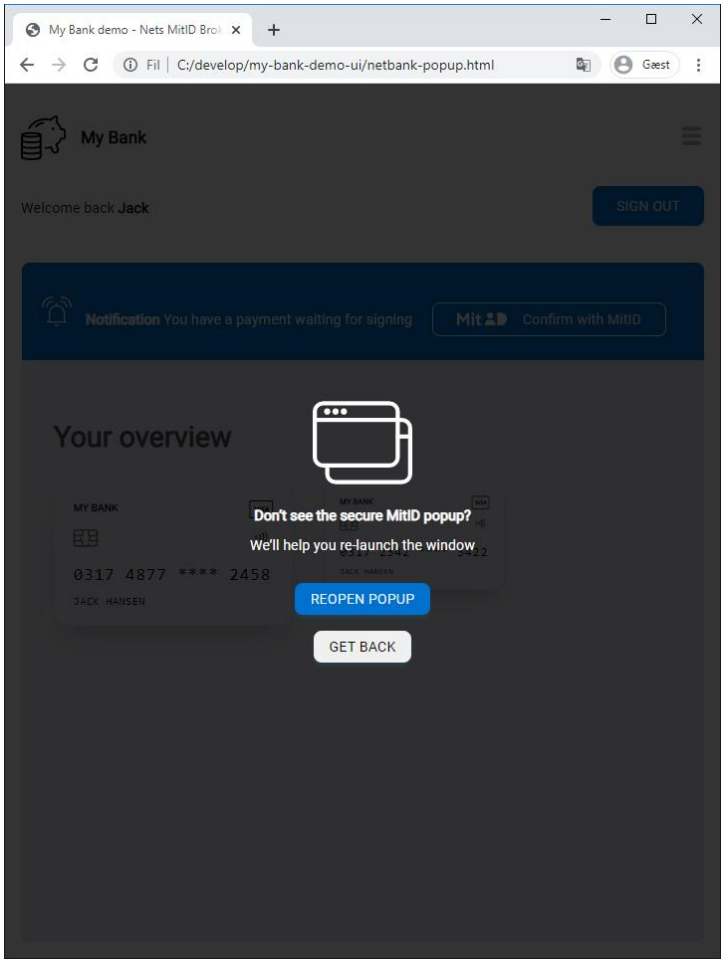
We recommend making a failover strategy: Either inform the user that the pop-up couldn't be loaded properly or automatically switch to fullscreen redirect.

Example of failover handling

```
window.setTimeout(function () {
    if(!wref || wref.closed || typeof wref.closed=='undefined'){
        //Your popup is blocked. Do the failover action
    }
}, 1000);
```

Background behind the pop-up screen

To ensure the best user experience, we encourage you to make a pop-up backdrop on your webpage, that put focus to the pop-up. It could be an transparent color overlay or a graphical backdrop. It's also recommended to implement support-text behind the pop-up, if the user minimizes or closes the pop-up unintended.



Pop-up backdrop with support-text example

HTML example

```
<div class="popup-dialog">
  <div class="popup-background"></div>
  <div class="popup-content">
    <figure class="popup"></figure>
    <strong>Don't see the secure MitID popup?</strong>
    <p>We'll help you re-launch the window</p>
    <button
      class="btn btn-main"
      onclick="popupwindow('signin.html', 'Sign in','400','588')"
    >
      Reopen popup
    </button>
    <button class="btn" onclick="closeWindow()">Get back</button>
  </div>
</div>
```

CSS example

```
.popup-dialog .popup-background {
    transition: 1s opacity; /* Makes animation on the fade down */
    opacity: 0;
  }
  .popup-dialog .popup-content {
    display: none;
  }

  /*The 'popup-active' class is added to a DOM element higher eg. <body>, when pop-up is open */
  .popup-active .popup-dialog {
    display: flex;
    align-items: center;
    justify-content: center;
    width: 100vw;
    height: 100vh; /* Ensures full hight of the viewport, even if the page is smaller than the
                      screen resolution*/
    position: absolute;
    left: 0;
    top: 0;
    z-index: 1000; /* Ensures breakout of depth. If you have higher Z-
                      indexes set, you need to raise this number*/
  }
  .popup-active .popup-dialog .popup-content {
    display: flex;
    flex-direction: column;
    width: 400px; /* minimum width of MidID, to ensure the popover hides the content*/
    height: 588px;
    align-items: center;
    justify-content: center;
    color: white;
    z-index: 1100;
    text-align: center;
  }
  .popup-active .popup-dialog .popup-content h3 {
    color: white;
  }
  .popup-active .popup-dialog .popup-content .btn-main {
    margin: 1rem;
  }
  .popup-active .popup-dialog .popup-background {
    opacity: 0.8; /* Controls the opacity of the backdrop */
    background: black;
    width: 100%;
    height: 100%;
    position: absolute;
  }
```

Keeping the pop-up in focus

If the user clicks away from the pop-up on the underlaying page, we advise you to give the pop-up window focus back again.

Code example of how you can do that

```
window.onfocus = function () {
    //wref is your object from containing the binding to the open window
    if (wref != null && typeof wref.window === 'object') {
        //when focus is set on parent page,
        //set the focus back to pop-up if it's open. The setTimeout is necessary to
        //avoid timing conflicts
        window.setTimeout(function () {
            wref.focus();
        }, 100);
    }
};
```

Please note: The default setting in Firefox called *dom.disable_window_flip, ignores the focus script.*

Eventlistener

To keep a connection between the pop-up and the parent frame, you need to setup an eventlistener, that can manage the events emitted from the pop-up.

Please note: The events emitted from the Nets eID Broker is not yet defined. A complete list of events will be posted later.

Code example on eventlistener

```
window.addEventListener("message", receiveMessage, false);

function receiveMessage(event) {
  if (event.data === "CLOSE") {
    // closing the pop-up
    closeWindow();
  } else if (event.data === "REDIRECT") {
    // other actions
    window.location.replace("some-url.html");
  }
}
```

Pop-up on mobile devices

Pop-up on mobile devices have been an UX challenge in the past. But up to date devices, has proven to handle it better than in the pat. But not all devices may handle it in the same way. Most devices handle the pop-up as a new tab. We encourage you to do comprehensive testing of your implementation on real mobile devices.

On iPad the pop-up is also opened as a new tab, which either makes a lot of whitespace or room for branding.

We have tested this pop-up solution on the following devices:

Android:

Samsung Galaxy S10e	Android 10	Chrome
Samsung Galaxy S10	Android 9	Chrome Firefox Samsung Base Browser UC Browser

Samsung Galaxy S9	Android 8	Chrome Firefox Samsung Base Browser UC Browser
Google Pixel 2	Android 7.1	Chrome Firefox
Google Nexus 6	Android 6	Chrome Firefox
Google Nexus 6	Android 5	Chrome Firefox
Google Nexus 5	Android 4.4	Chrome Firefox

IOS:

iPhone 11	IOS 13	Chrome Safari
iPhone XS	IOS 12	Chrome Safari
iPhone 8 Plus	IOS 11	Chrome Safari
iPhone 6S plus	IOS 9	Chrome Safari
iPhone 6	IOS 8	Chrome Safari
iPad Pro 12.9	IOS 13	Chrome Safari

User Interface elements

Accessibility

If the button or link for opening the Nets eID Broker is anything else than an <a> or <button> element, you should add `role="link" tabindex="0"` to the DOM element, to make it accessible for keyboard navigation and screen readers according to the WCAG 3.0 standard.

We advise you to put helping text on functions that opens in a new window. You can use either of these two ways:

```
<!-- option 1 aria-label -->
<button aria-label="MitID sign in. Opens in a new window">Sign in</button>

<!-- option 2 visually hidden text -->
<button>Sign in <span class="visually-hidden">Opens in a new window</span></button>
<!-- visually-hidden class containing visibility:hidden in css -->
```