



# **Secure Authentication in a Micro Frontend World**

# TABLE OF CONTENTS

---

Micro frontends

01



04

Kubernetes config

Module federation

02



05

Live application

OIDC authentication

03





**Peter Eijgermans**

**CodeSmith / Architect  
Full-Stack**



# Context: ProRail & the Dutch Railways

---





# **Spoorviewer 2.0**

---

**Demo time!**

# Spoorviewer our future ready app!



Spoorviewer

\*tab1

\*Werkblad2

tab3

default

default

default

BET

GVC

**ZL / AML / Almelo (PPLG)**  
Servertijd: 17:11:48

1653	A	16:24	-1	Aml	16:20	WA	202B	
1693	V	16:25	+1	Aml	16:20	202B	AH	
7951	A	16:29	+1	Aml	16:26	WA	202B	
7960	A	16:30	0	Aml	16:26	HH	204A	
7960	V	16:31	0	Aml	16:26	204A	AW	
7951	V	16:30	+1	Aml	16:26	202B	AH	
7053	A	16:36	0	Aml	16:32	WA	203B	m
+ 1660	A	16:35	+4	Aml	16:35	HH	202A	
+ 1660	V	16:36	+5	Aml	16:35	202A	WA	
31062	A	16:43	-1	Aml	16:39	GA	204A	m
+ 144	D	16:19	+25	Aml	16:41	HH	AW	1
31053	V	16:48	0	Aml	16:47	204A	GA	m
! 49729	D	16:50	+1	Aml	16:48	WA	AH	1
7062	V	16:53	0	Aml	16:52	203B	AW	m
! 1755	A	16:54	0	Aml	16:52	WA	202B	
! 1755	V	16:55	+1	Aml	16:52	202B	AH	
7953	A	16:59	0	Aml	16:56	WA	202B	
7953	V	17:00	0	Aml	16:56	202B	AH	
! 7962	A	17:00	+2	Aml	16:58	HH	204A	
! 7962	V	17:01	+2	Aml	16:58	204A	AW	

Ah-Ww

Laatste update: 17:11:45

**Ah 20026 (-2)**

3656

I

Ahp

I

Va

I

IJbww

I

30955 (+1) Wtv

I

Dvn

I

Zv

I

Did

**AH / AH / Arnhem (PPLG)**  
Servertijd: 17:11:48

3156	V	16:17	0	Ah	16:16	10	AK	m
! 3689	A	16:19	0	Ah	16:16	EV	7	
7655	A	16:20	0	Ah	16:17	AE	3	
! 124	A	16:27	-5	Ah	16:19	EV	9	
31147	A	16:23	0	Ah	16:20	AE	4B	m
! 3659	V	16:23	0	Ah	16:22	7	BE	
3056	A	16:26	+1	Ah	16:22	AE	11	
7655	V	16:25	0	Ah	16:24	3	FV	
! 3055	A	16:28	0	Ah	16:25	AX	8	
! 124	V	16:29	0	Ah	16:28	9	AK	
! 89982	A	16:33	-2	Ah	16:28	AX	7	
! 3056	V	16:32	+1	Ah	16:31	11	AK	
+ 7658	A	16:36	0	Ah	16:31	EV	9	
! 3654	A	16:37	0	Ah	16:33	AE	3	
! 3055	V	16:35	0	Ah	16:34	8	BE	
+ 408943	R	16:43			16:39	27	8	m
! 3654	V	16:41	0	Ah	16:40	3	FV	
3157	A	16:43	+1	Ah	16:41	AX	10	m
+ 7658	V	16:41	+2	Ah	16:41	9	BE	
20091	V	16:46	+1	Ah	16:41	6A	EV	1

Ut-Brn

Laatste update: 17:11:51

661 (-1) Ut

I

Uto

I

Blw

I

5561

Blo

I

Bloa

I

11761 (+1) Bhv

I

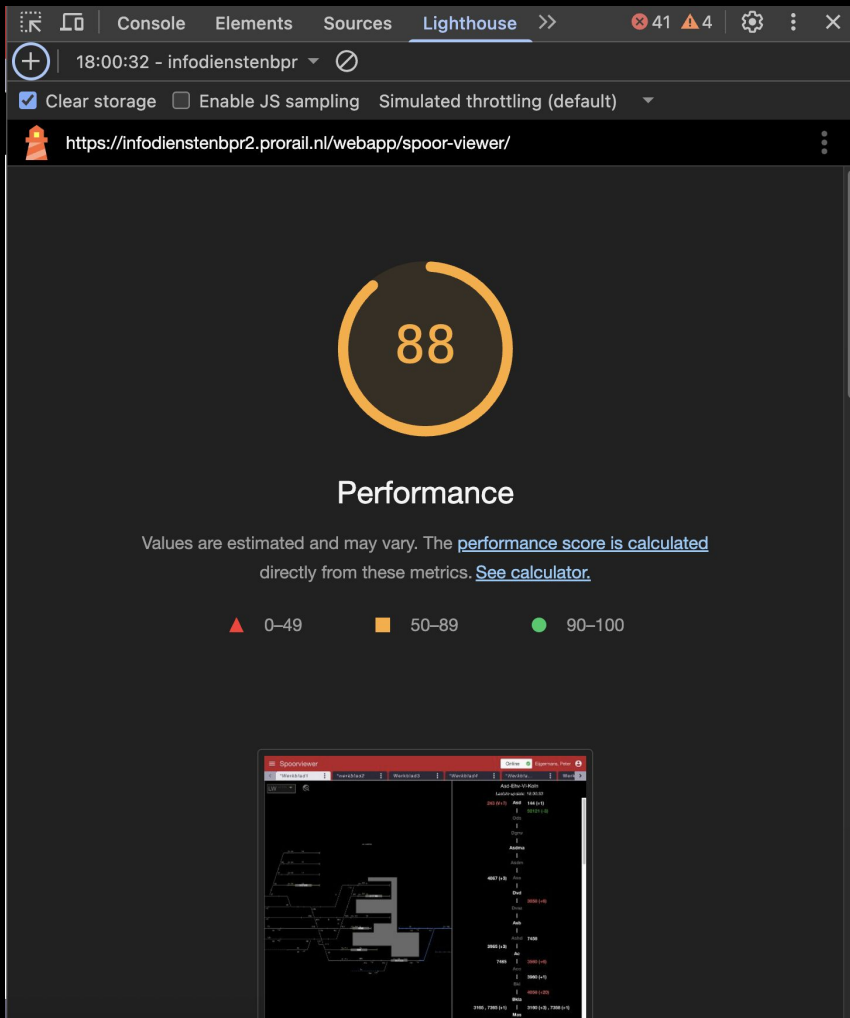
5556, 556 (+7)

Did

I

Str

Het is niet toegestaan om Spoorviewer te gebruiken voor veiligheidsdoeleinden, veiligheidsfuncties en veiligheidsbehandelingen.



## Benefits:

- ✓ Faster frontend – big performance gains
- ✓ Better team collaboration – alignment across teams
- ✓ Easier maintenance & scaling
- ✓ Faster feedback cycles – quicker delivery, happier users

# Performance Micro frontends



Looks good, right?

Now... let's rewind to *2019!*





# Meet Petra - Train Controller in 2019



6:00 AM: Needs real-time train positions

6:15 AM: Emergency response required

6:30 AM: System takes 45 seconds to load each screen

6:31 AM: Frustration level: MAXIMUM



# 8:47 AM - Amsterdam Station Chaos

---



**3,000 railway workers locked  
out**

**Authentication system:  
CRASHED**

**Rush hour:  
IN PROGRESS**

**Punchline:**

This is why we're here—  
to **prevent** such **disasters**

# Show of Hands Time

---

 "Who's afraid to deploy on Friday afternoon?"

 "Who's maintaining multiple authentication systems?"

 "Who's dealing with a monolith that takes forever to build?"

# The Railway Monolith **2019**: A Beautiful Disaster

---



NOT MAINTAINABLE

SLOW DEVELOPMENT/DEPLOYMENT

HARD TO SCALE

SINGLE POINT OF FAILURE

"This HUGE app is like one giant box of  
LEGO **that nobody can organize.**

If one part of the app breaks? The whole thing  
crashes.

If you want to add something new? You have to  
rebuild the whole box."

# Monolith

---



***Massive*** Application





The diagram illustrates a monolithic architecture. It features a large, dark blue rounded rectangle with an orange border. Inside this rectangle, there is a large orange rounded rectangle at the top containing the text '***Massive*** Application'. Below this, there is a smaller, light gray rounded rectangle containing the text 'Database'. The orange rectangle is positioned above the gray rectangle, and both are centered within the dark blue container.

Database




# What If We Could..

---

## Architecture: How to get from Monolith to Modular speed ?!

-  Deploy features independently — anytime, without fear
-  Let teams move fast — without stepping on each other's toes
-  Connect multiple apps seamlessly
-  Use Module Federation to share code and scale easily

## Authentication & Security:

-  Login once — access everything, seamlessly
-  Share authentication state across all apps
-  Auto-refresh tokens & secure WebSockets for real-time data



**These are the dreams of every team:**

First, we want to build and improve fast—  
with smaller parts, easy to connect, no more crashes.

Then, we want users to log in just once,

Today, I'll walk through these points—  
and we'll check them off

**Let's go!**

# Architecture: From Monolith to Modular Speed !

---



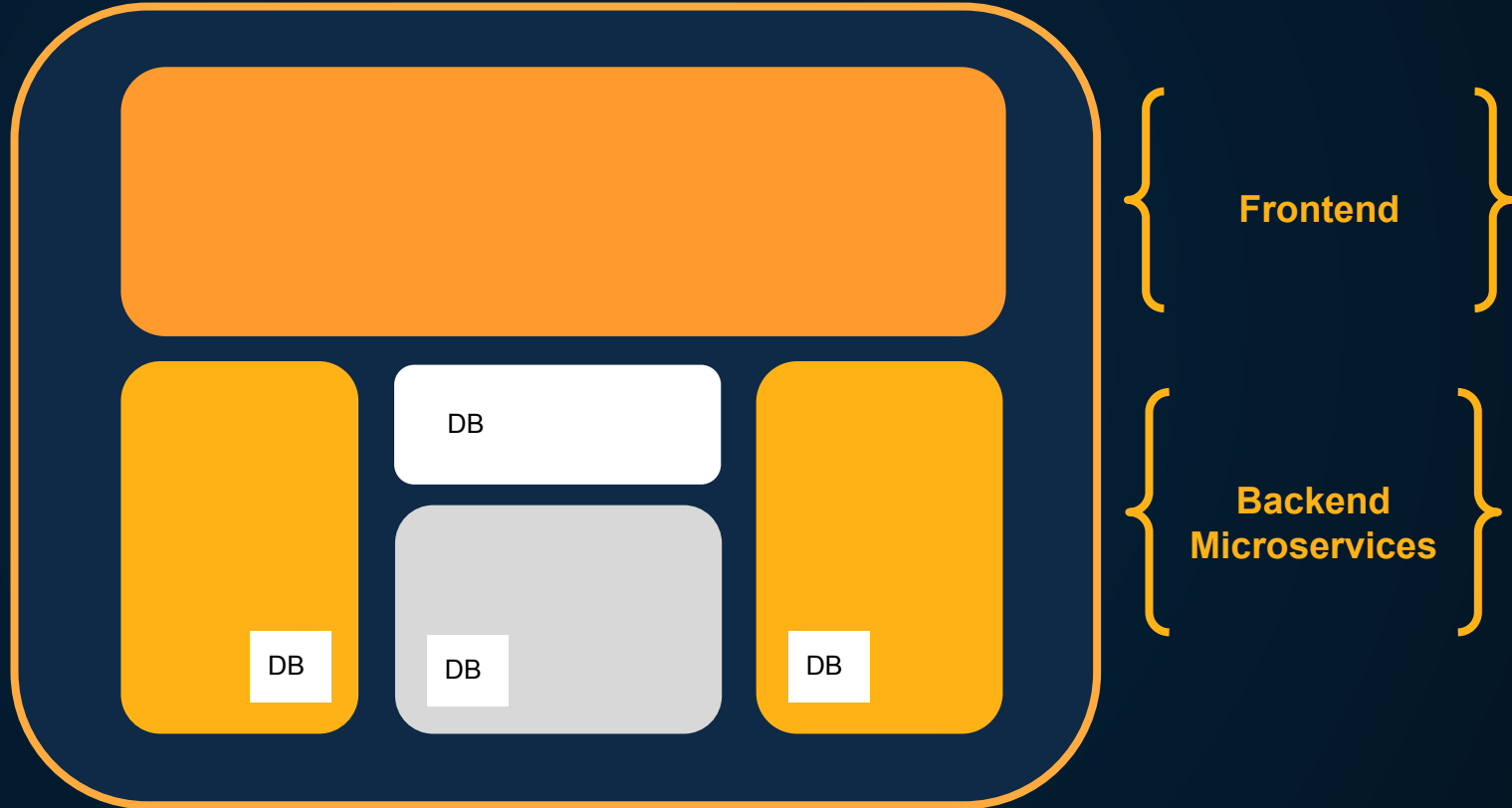
# Step 1 - Split frontend and backend

---



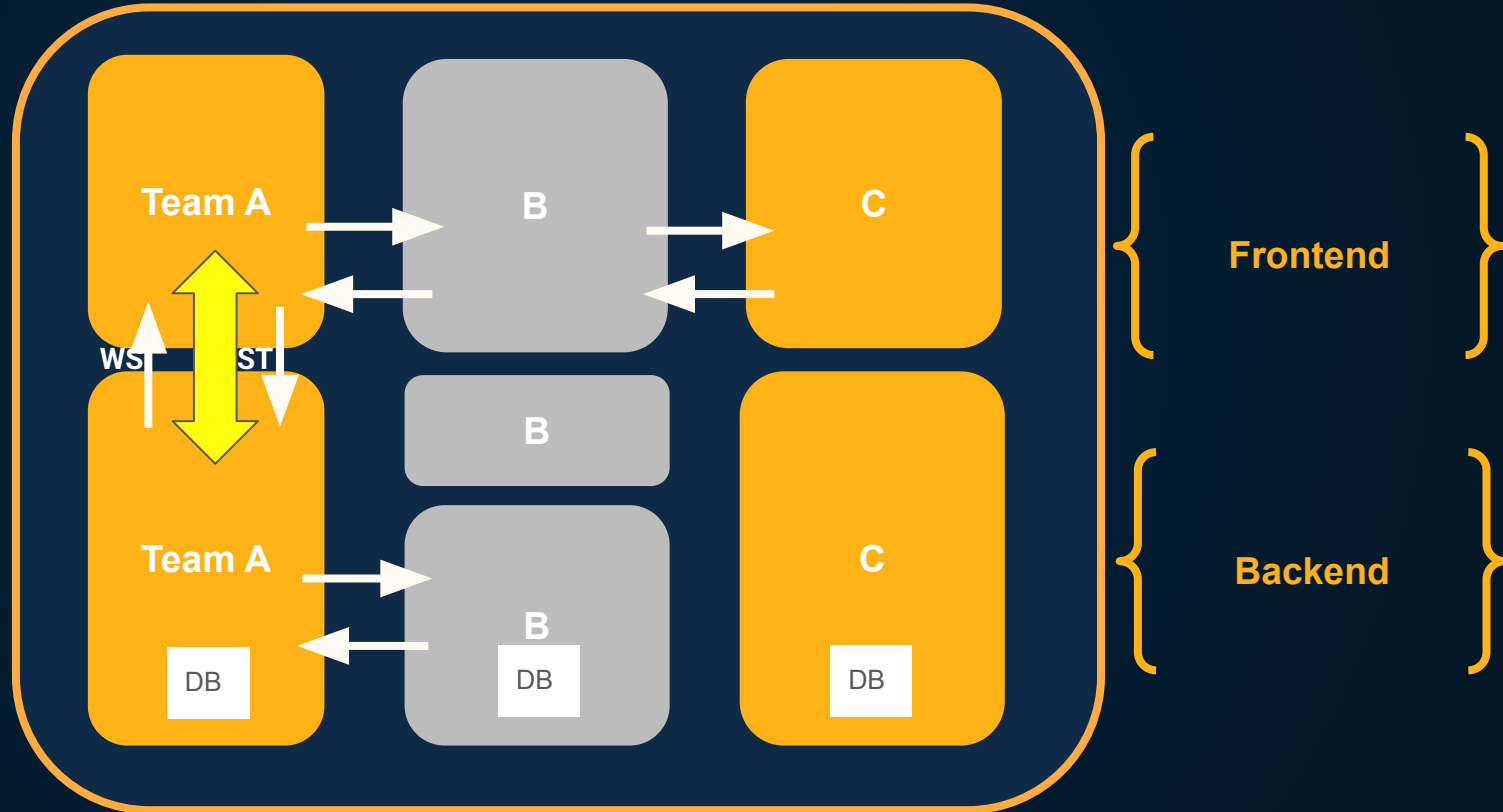
## Step 2 - Split backend as micro services

---



# Step 3 - Split frontend -> Verticals

pairing



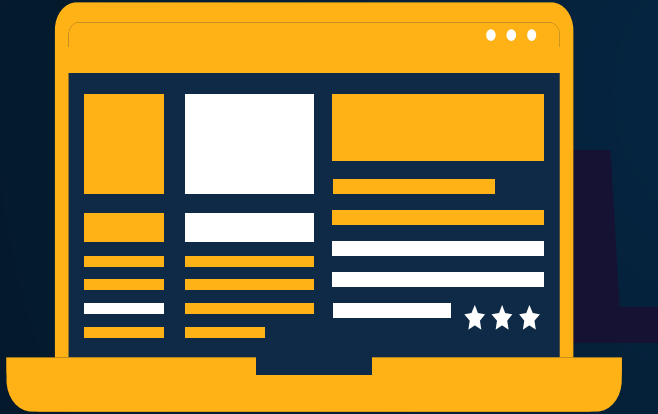


*"An architectural style where  
independently deliverable  
**frontend and backend**  
applications are composed into  
a greater whole"*

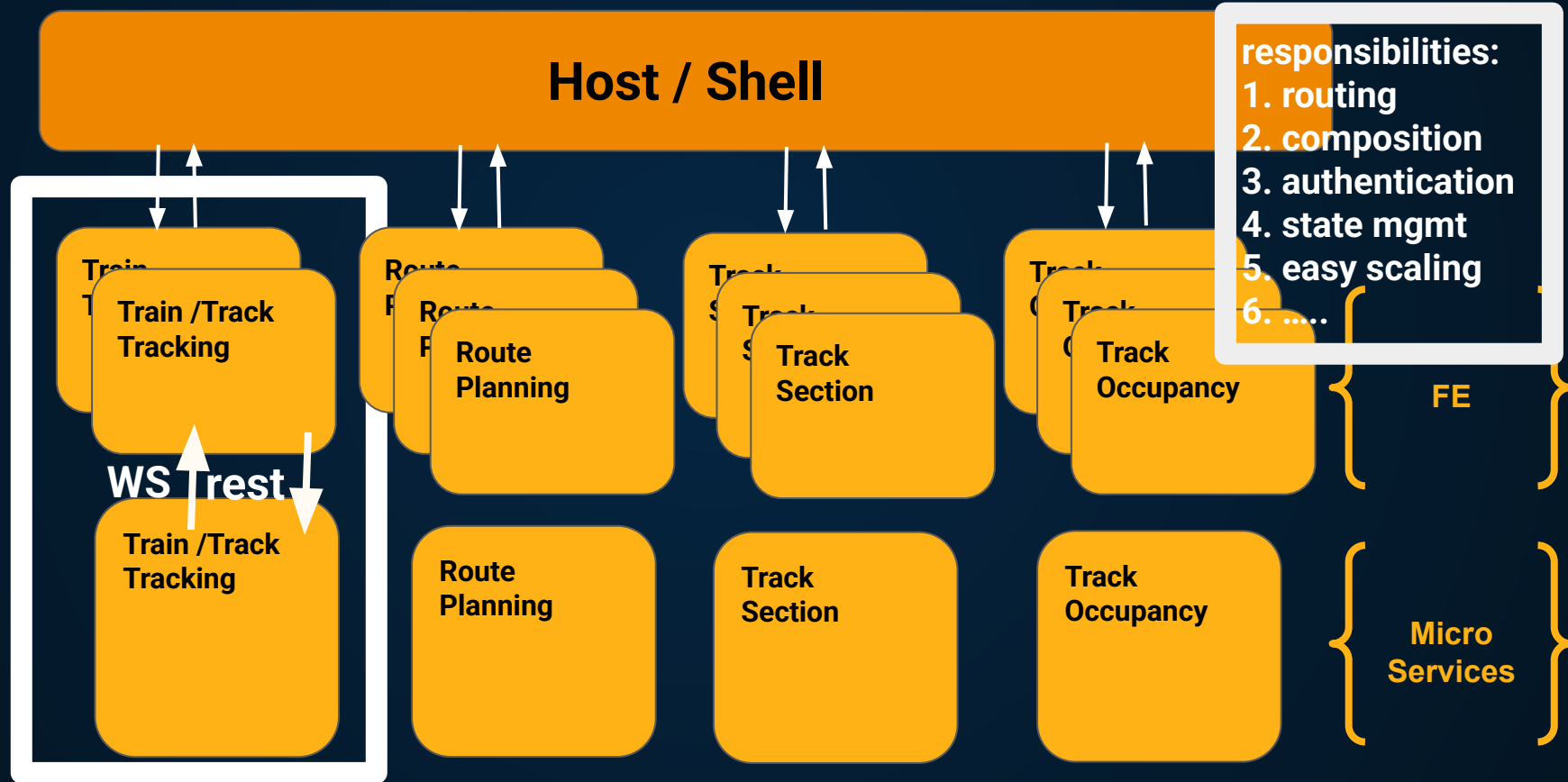
**Martin Fowler**

# Micro Frontends to the Rescue

---



# Micro frontend architecture Dutch Railways





Spoorviewer

HOST / SHELL

OnlinePeter Eijgmans

\*tab1

\*Werkblad2

tab3

default

default

default

BE1

MF train tracking

GVC

ZL / AML / Almelo (PPLG)

Servertijd: 17:11:48

1653	A	16:24	-1	Aml	16:20	WA	202B	
1653	V	16:25	0	Aml	16:20	WA	202B	
7951	A	16:29	+1	Aml	16:20	WA	202B	
7960	A	16:30	0	Aml	16:26	HH	204A	
7960	V	16:30	0	Aml	16:26	HH	204A	
7951	V	16:30	+1	Aml	16:26	202B	AH	
+	1660	A	16:35	+4	Aml	16:35	HH	202A
+	1660	V	16:36	+5	Aml	16:35	202A	WA
31062	A	16:43	-1	Aml	16:39	GA	204A	m
+	144	D	16:19	+25	Aml	16:41	HH	AW 1
31053	V	16:48	0	Aml	16:47	204A	GA	m
!	49729	D	16:50	+1	Aml	16:48	WA	AH 1
7062	V	16:53	0	Aml	16:52	203B	AW	m
!	1755	A	16:54	0	Aml	16:52	WA	202B
!	1755	V	16:55	+1	Aml	16:52	202B	AH
7953	A	16:59	0	Aml	16:56	WA	202B	
7953	V	17:00	0	Aml	16:56	202B	AH	
!	7962	A	17:00	+2	Aml	16:58	HH	204A
!	7962	V	17:01	+3	Aml	16:58	204A	AW

AH / AH / Arnhem (PPLG)

Servertijd: 17:11:48

3156	V	16:17	0	Ah	16:16	10	AK	m
!	3659	A	16:19	0	Ah	16:16	EV 7	
7655	A	16:20	0	Ah	16:17	AE 3		
!	124	A	16:27	-5	Ah	16:19	EV 9	
31147	A	16:23	0	Ah	16:20	AE 4B		m
!	3659	V	16:23	0	Ah	16:22	7	BE
3056	A	16:26	+1	Ah	16:22	AE 11		
7655	V	16:25	0	Ah	16:24	3	FV	
!	3055	A	16:28	0	Ah	16:25	AX 8	
!	124	V	16:29	0	Ah	16:28	9	AK
!	89982	A	16:33	-2	Ah	16:28	AX 7	
!	3056	V	16:32	+1	Ah	16:31	11	AK
+	7658	A	16:36	0	Ah	16:31	EV 9	
!	3654	A	16:37	0	Ah	16:33	AE 3	
!	3055	V	16:35	0	Ah	16:34	8	BE
+	408943	R	16:43			16:39	27	8 m
!	3654	V	16:41	0	Ah	16:40	3	FV
3157	A	16:43	+1	Ah	16:41	AX 10		m
+	7658	V	16:41	+2	Ah	16:41	9	BE
20031	V	16:45	+1	Ah	16:44	6A	EV 1	m

AH-VW

Laatste update: 17:11:45

Ah 20026 (-2)

30955 (+1) Wtv

Dvn

Zv

Did

Het is niet toegestaan om Spoorviewer te gebruiken voor veiligheidsdoeleinden, veiligheidsfuncties en veiligheidshandelingen.

**But...**

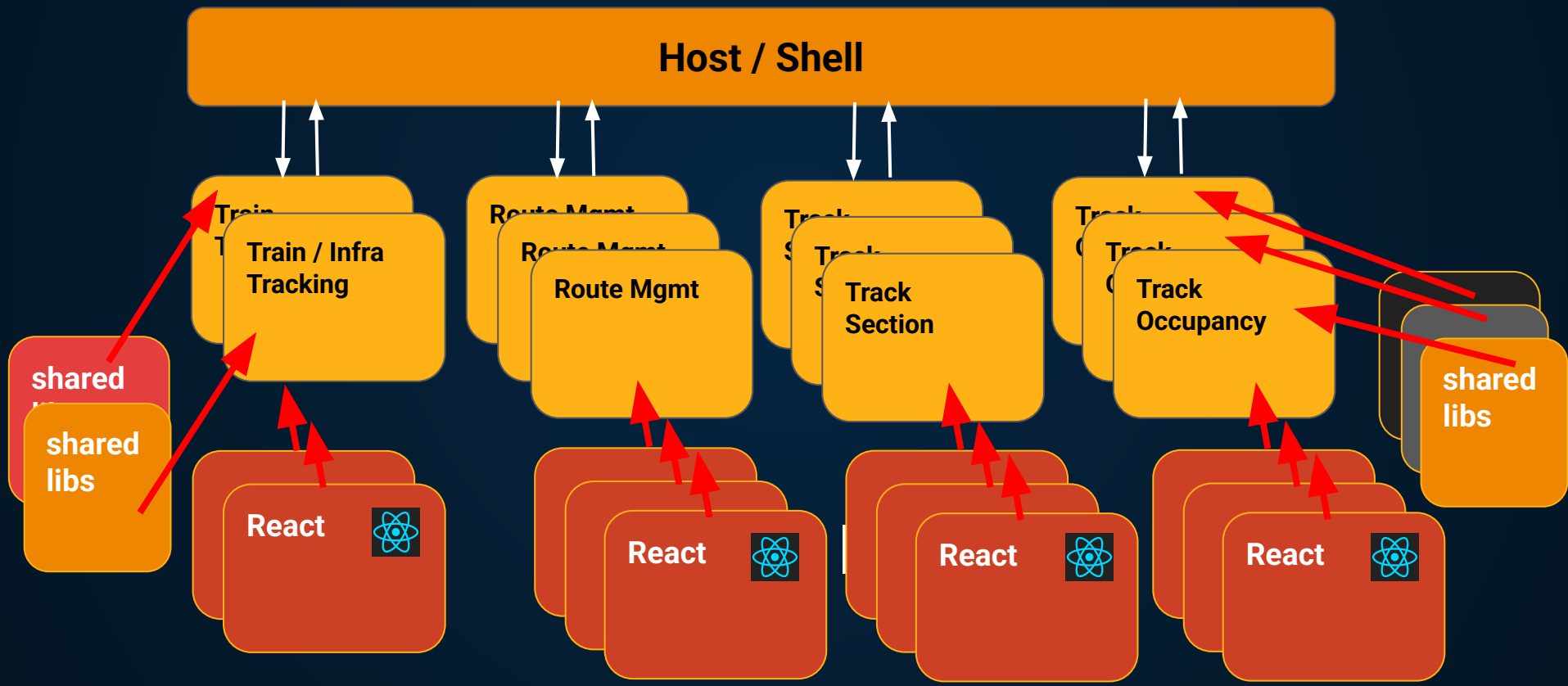
**How to build the  
Frontend efficiently?**

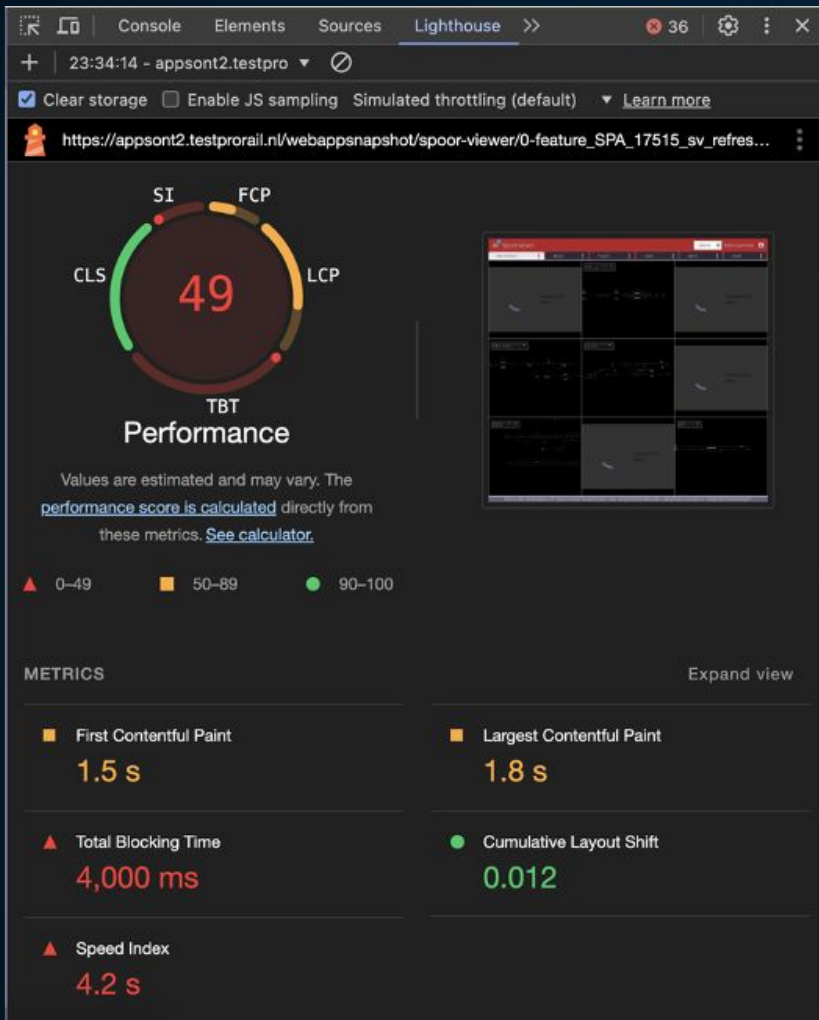
# Module Federation

---



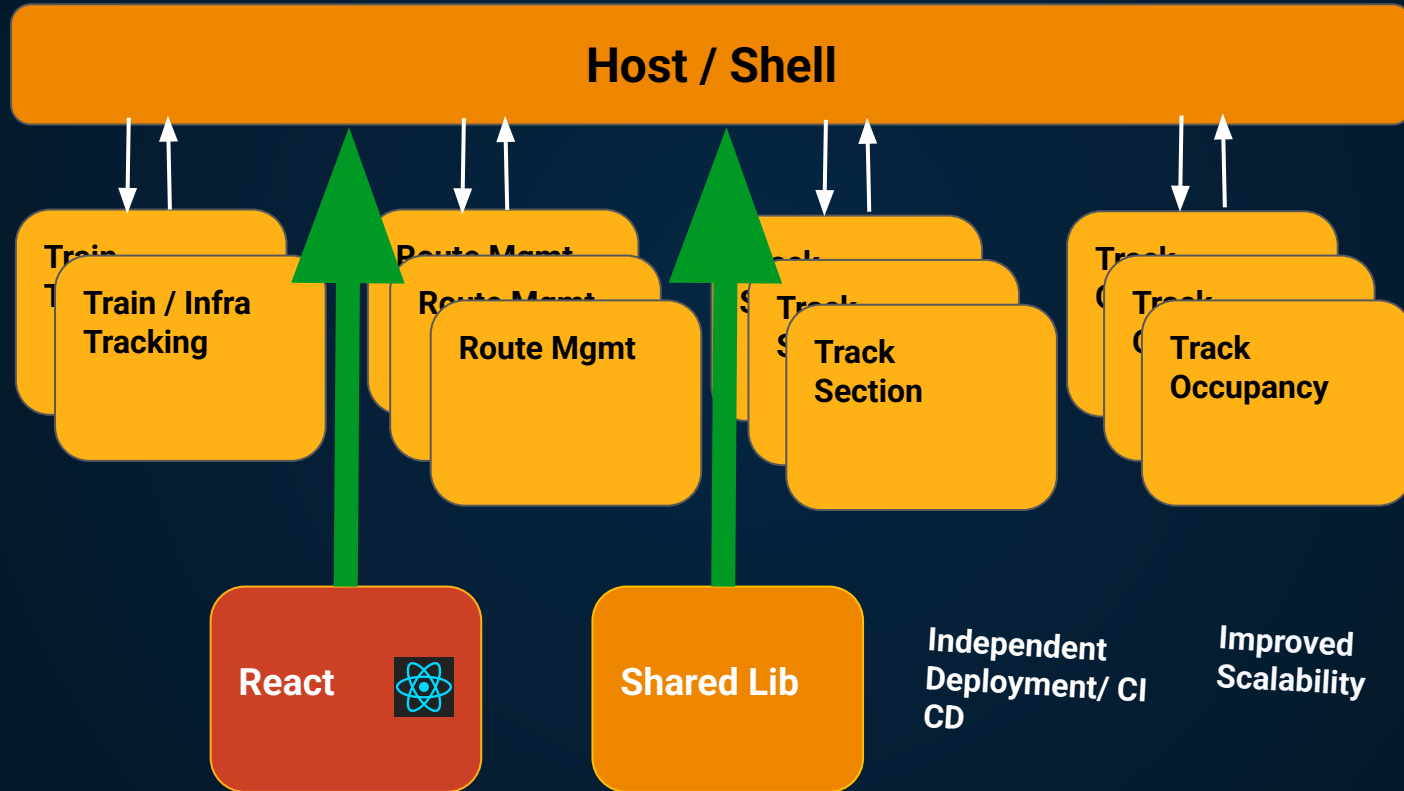
# 2022 *WebComponents*





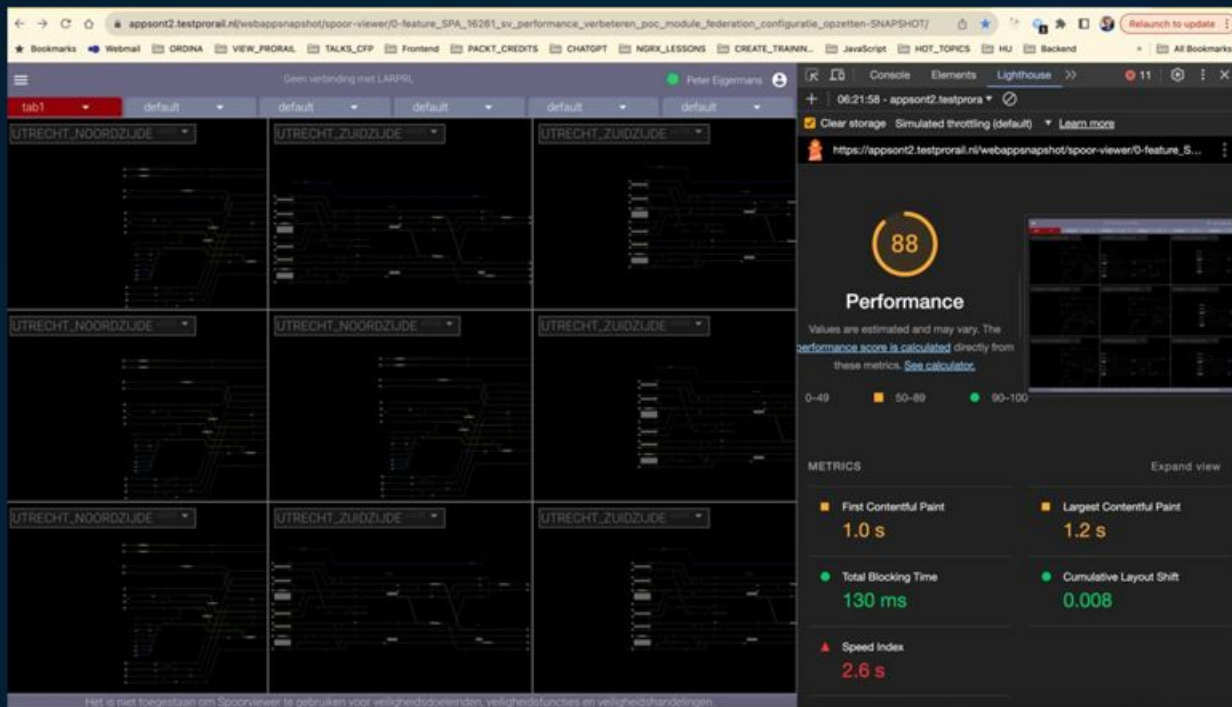
 50s load time  
→ Petra misses emergencies

# 2025 Module Federation for speed !!!



Technology Agnostic





# Performance Module Federation

**How to config?**



# Two Roles !

---

**Host / Shell**

**Federation.config**

**Remote 1**

**Federation.config**

**Remote 2**

**Federation.config**

# Config Host app

---



```
const { withNativeFederation, shareAll } = require('@softarc/native-federation/build');

module.exports = withNativeFederation({
  name: 'host',
  shared: shareAll(),
  skip: ['react-dom/server', 'react-dom/server.node', 'vite-react-microfrontends'],
});
```

# Config Remote app

---



```
const { withNativeFederation, shareAll } = require('@softarc/native-federation/build');

module.exports = withNativeFederation({
  name: 'remote',
  exposes: {
    './remote-app': './src/App.tsx',
  },
  shared: shareAll(),
  skip: ['react-dom/server', 'react-dom/server.node', 'vite-react-microfrontends'],
});
```

# Config specific versions to share !



```
const { withNativeFederation, shareAll } = require('@softarc/native-federation/build');

module.exports = withNativeFederation({
  name: 'remote',
  exposes: {
    './remote-app': './src/App.tsx',
  },
  shared: {
    react: {
      singleton: true,
      eager: true,
      requiredVersion: "18.2.0", // 📁 specific version
    },
    "react-dom": {
      singleton: true,
      eager: true,
      requiredVersion: "18.2.0",
    },
  },
  skip: ['react-dom/server', 'react-dom/server.node', 'vite-react-microfrontends'],
});
```

How does the *Host*  
load  
the *Remotes*?

✓ src

✓ assets

{ } manifest.local.json

{ } manifest.prod.json

{ } manifest.test.json







```
{  
  "mfe1": "https://prod.nl/mfe1/remoteEntry.json",  
  "mfe2": "https://prod.nl/mfe2/remoteEntry.json"  
}
```

# Mid-talk Checkpoint

---

## Architecture & Speed (Micro Frontends & Module Federation):

- ✓  Deploy features independently — anytime, without fear
- ✓  Let teams move fast — without stepping on each other's toes
- ✓  Connect multiple apps into one seamless experience
- ✓  Use Module Federation to share code and scale easily





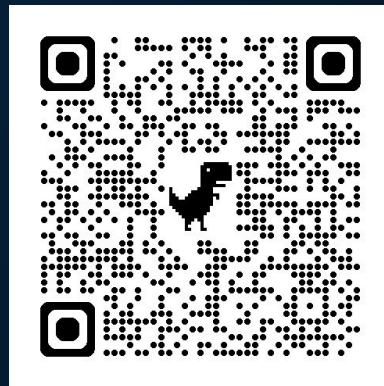
**Grap** the Code!  
Go **Module**  
**Federation** !

---

"Breaking the system apart helped speed up development—but but we still had a major problem left."

# Authentication

---



**WHEN YOUR SECURITY GATE**



**IS A LADDER**

# The Multi-App Authentication Nightmare

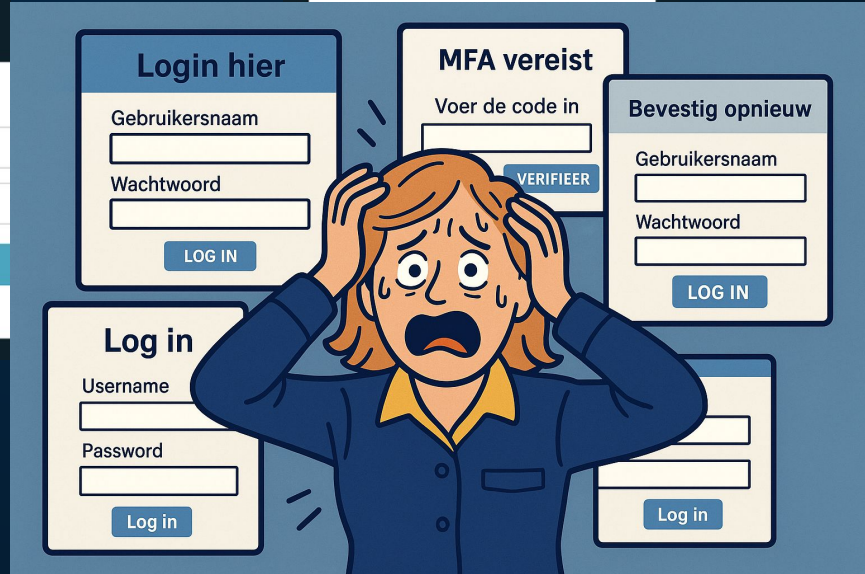
Multiple logins

Token Chaos

CORS issues

Redirect loops

Slow or broken SSO





# The 2 Heroes of Modern Authentication

## OpenID Connect (OIDC): Identity Layer

"OIDC is like showing your ID at the entrance"

## OAuth 2.1: The Delegation Protocol

"OAuth is like getting a festival wristband"



### Key Players:

- Authorization Server (**EntraID** - The Bouncer)
- Resource Server (**Your API** - The Bartender checks wristband)
- Client Application (**React App** - You, The Customer)





# OIDC Flow with **MSAL** !

---

"Now let's see exactly how this  
OIDC flow works step by step.

Starting with what happens when a  
user clicks 'Login'..."

# Flow 1: Initial Login & Authorization

---



# FIRST: needed *msalConfig*

```

● ● ●

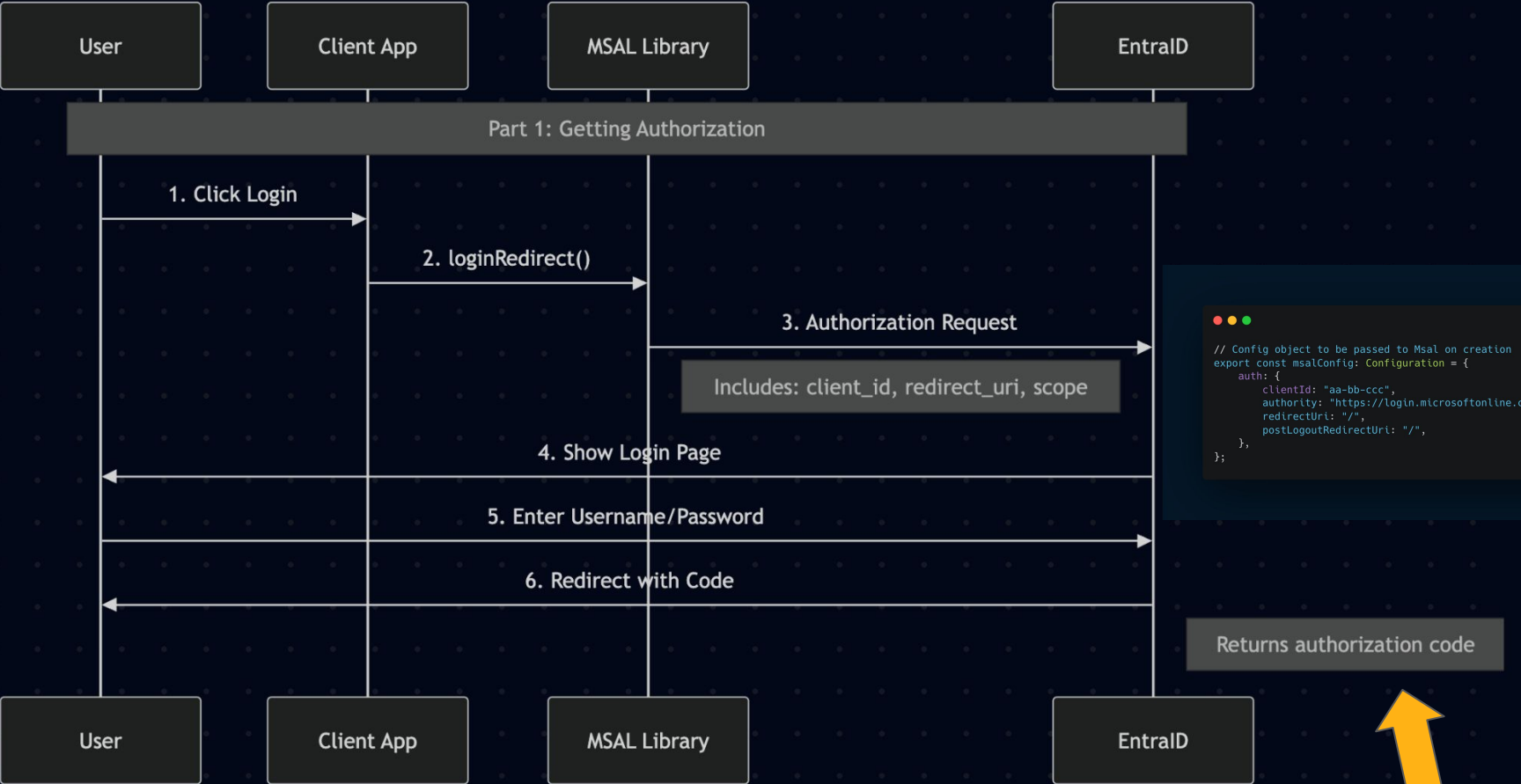
// Config object to be passed to Msal on creation
export const msalConfig: Configuration = {
  auth: {
    clientId: "b5c2e510-4a17-4feb-b219-e55aa5b74144",
    authority: "https://login.microsoftonline.com/<TENANT-ID>",
    redirectUri: "/",
  },
};

export const loginRequest: PopupRequest = {
  scopes: ["User.Read"],
};

export const protectedResources = {
  todoListApi: {
    endpoint: "YOUR_API_ENDPOINT_URL",
    scopes: ["api://YOUR_CLIENT_ID/access_as_user"],
  },
};

```

# Authorization Server (EntraID - The Bouncer)



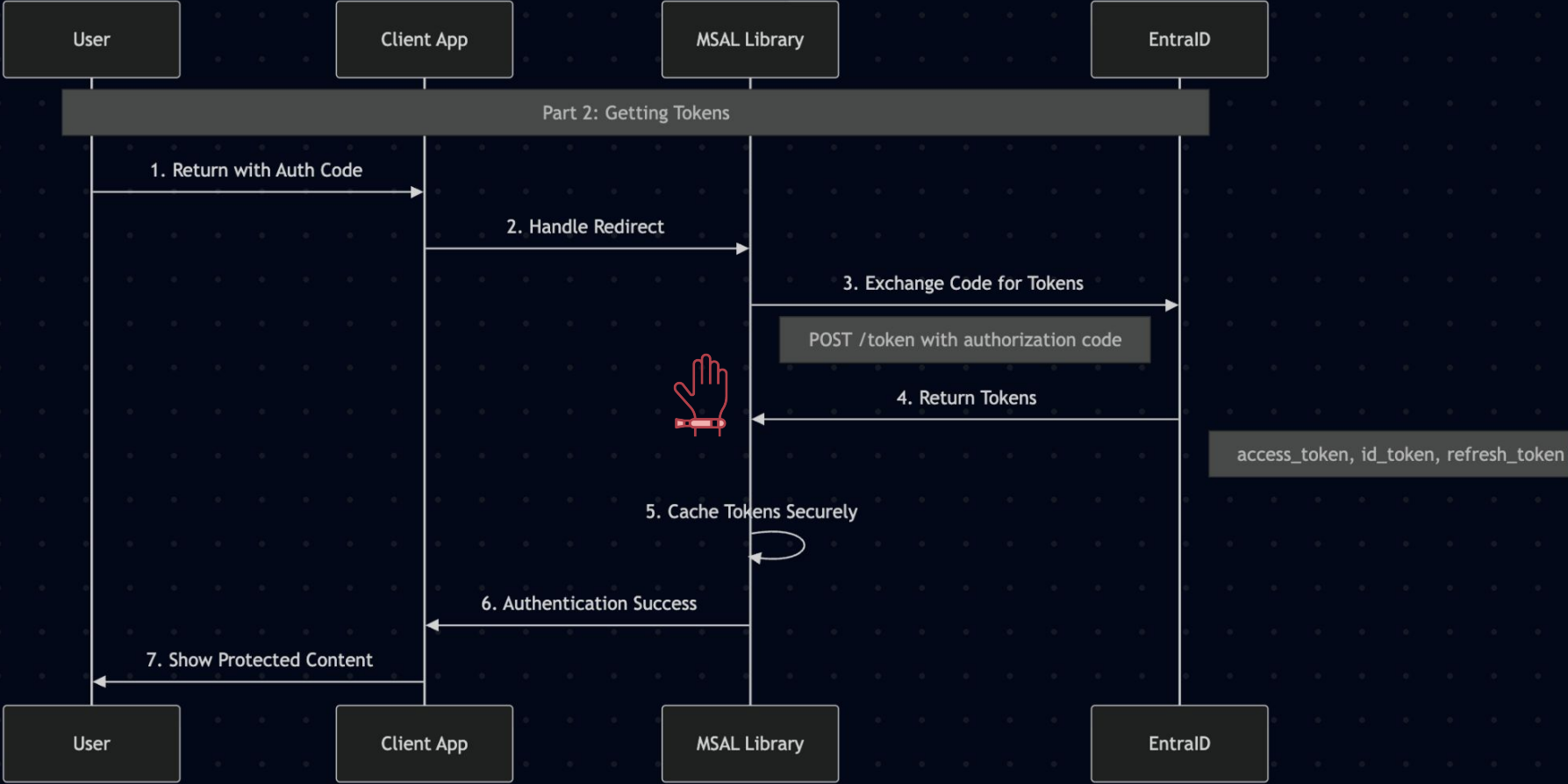
Part 1 → Part 2: *"Great!*  
*We have an authorization code.*  
*Now let's see how we exchange it for tokens..."*

## Flow 2: Token Exchange & Caching

---






Authorization Server (EntraID - The Bouncer)



The Scope is a **URL** for specific **permissions** for an **API**



<i>Token Types</i>	<i>Purpose</i>	<i>Where Used</i>	<i>Format</i>	<i>Lifetime</i>	<i>Example Scope Needed</i>
 <b>Access Token</b>	Access to <b>protected APIs</b>	Sent in Authorisation headers as <b>Bearer</b>	JWT	~1 hour	<code>api://xyz/.default</code>
 <b>ID Token</b>	User identity info for <b>authentication</b>	Used by frontend	JWT	~1 hour	
 <b>Refresh Token</b>	Get new tokens without login. <b>Enables SSO!</b> 🎯	Used internally	hidden	Long-lived	

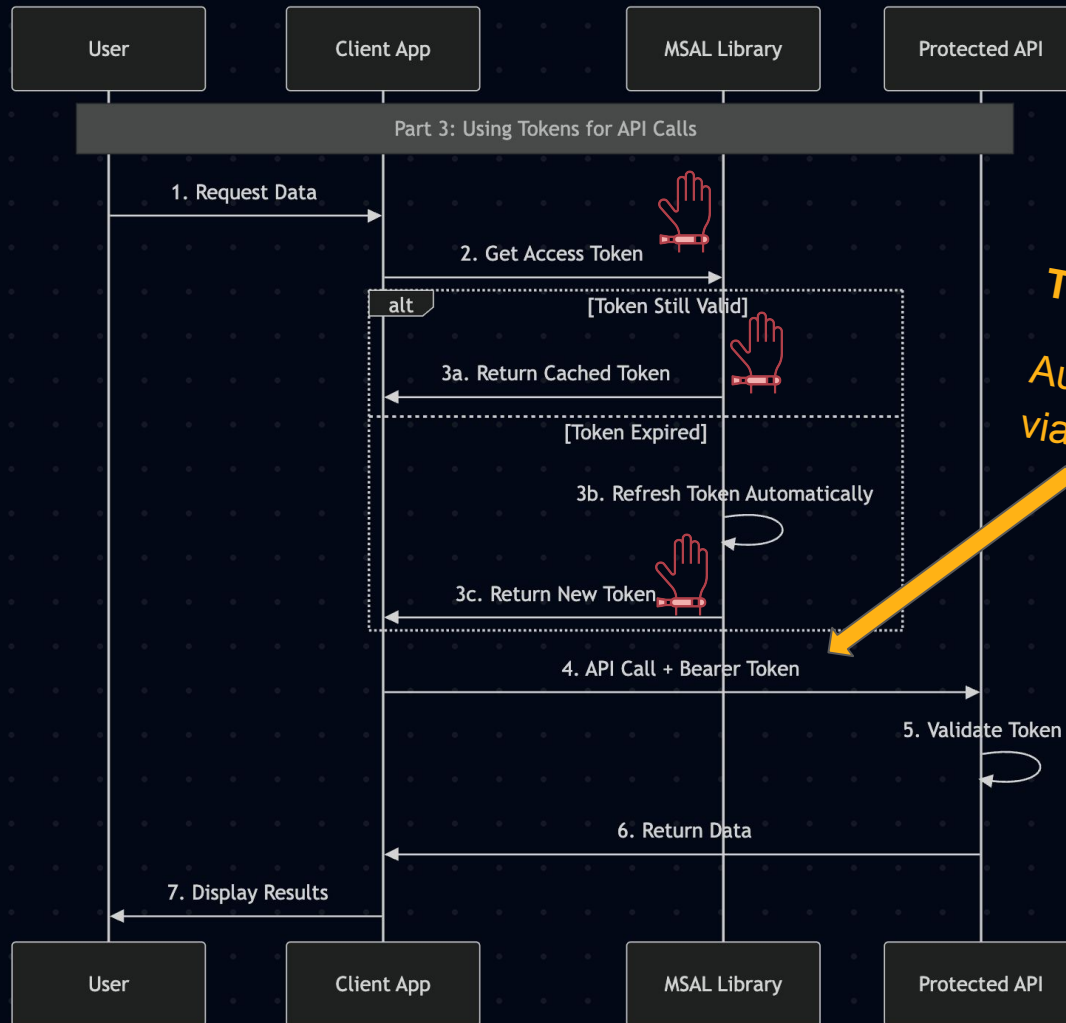
Part 2 → Part 3: *"Perfect! We have our tokens  
cached in the browser! → **local or session scope***

*Now let's see how our application uses them..."*

# Flow 3: API Calls & Token Management

---





**The Magic:**  
Automatic token injection  
via HTTP interceptors!

# *MSAL Token injection* ==> Bearer token

---



# OIDC Flow - Key Takeaways!

---

## Login

User → EntraID → Authorization Code

## Exchange

Authorization Code → 3 Tokens:

- **Access:** API calls
- **ID:** User info
- **Refresh:** New tokens + **SSO**

## Use

Access token → API calls (auto-refresh)

# Oops! Someone Stole Your Auth Code

---



If someone steals that code → they can get access tokens and act as your app!



# Quiz: How Would You Fix This?

---

**How can we stop a stolen code from being useful?**

- A) Use HTTPS
- B) Make codes expire faster
- C) Tie the code to the original app somehow
- D) Just trust users not to get hacked 😊

# PKCE to the rescue!

PKCE adds a “proof” step:

App proves it's the same one that



How PKCE

1

An



Why it's secure

If someone steals the authorization code, they **don't** have the **code\_verifier**,  
so they can't exchange it for tokens.

2

5

6

S

match

→

OK

→

tokens issued





# MSAL for OIDC

---

**Key Benefit:** MSAL abstracts away the complexity of token management, token renewal and token injection.



# Install MSAL libs

---

**npm install @azure/msal-browser**

**npm install @azure/msal-react**

# Now Let's See How MSAL React Handles This

---

## MSAL AuthenticationResult Interface:

```
interface AuthenticationResult {  
  
    accessToken: string;    // "eyJ0eXAiOiJKV1QiLCJhbGci..."  
  
    idToken: string;        // Identity proof  
  
    refreshToken: string;   // Renewal ticket  
  
    expiresOn: Date;        // Expiration time  
  
    scopes: string[];       // Permissions granted  
  
    account: AccountInfo;    // User profile information  
  
}
```



# MSAL Architecture

---



# React MSAL Architecture Overview

*// Core MSAL Angular Components*

MSAL Angular

**MsalService**

*// Core service*

MsalGuard

*// Route protection*

MsalInterceptor

*// HTTP token injection*

MsalBroadcastService

*// Event notifications*

MsalRedirectComponent

*// Handle redirects*

**Transition : "Now that we understand MSAL architecture, let's see how this looks in a real React component..."**

# Real-World Login Component

---





# Make MSAL Context available to all Components



```
// App.js
import React from 'react';
import { MsalProvider } from '@azure/msal-react';
import { PublicClientApplication } from '@azure/msal-browser';
import { msalConfig } from './msalConfig';
```

```
const msalInstance = new PublicClientApplication(msalConfig);
```

```
function App() {
```

```
  return (
```

```
    <MsalProvider instance={msalInstance}> ← MSAL Context
      /* Your application's components go here */
```

```
    <Home />
```

```
  </MsalProvider>
```

```
);
```

```
}
```

```
export default App;
```



# Building Your First AuthButtons.jsx Component

```
// components/AuthButtons.jsx
import React from 'react';
import { useMsal, AuthenticatedTemplate, UnauthenticatedTemplate } from '@azure/msal-react';

const AuthButtons = () => {
  const { instance } = useMsal();

  const handleLogin = () => {
    // You can choose loginPopup() or loginRedirect()
    instance.loginRedirect().catch(e => {
      console.error(e);
    });
  };

  const handleLogout = () => {
    // You can choose logoutPopup() or logoutRedirect()
    instance.logoutRedirect().catch(e => {
      console.error(e);
    });
  };

  return (
    <div>
      <AuthenticatedTemplate>
        <button onClick={handleLogout}>Sign Out</button>
      </AuthenticatedTemplate>
      <UnauthenticatedTemplate>
        <button onClick={handleLogin}>Sign In</button>
      </UnauthenticatedTemplate>
    </div>
  );
};
```





# Use the AuthButtons.jsx

```
// components/PageLayout.jsx
import React from 'react';
import { AuthenticatedTemplate, UnauthenticatedTemplate } from '@azure/msal-react';
import AuthButtons from './AuthButtons';

const PageLayout = (props) => {
  return (
    <>
      <header>
        <h1>My App</h1>
        <AuthButtons />
      </header>
      <main>
        <UnauthenticatedTemplate>
          <p>You need to sign in to view the content.</p>
        </UnauthenticatedTemplate>
        <AuthenticatedTemplate>
          {props.children}
        </AuthenticatedTemplate>
      </main>
    </>
  );
};

export default PageLayout;
```

**Acquiring**  
***Access Tokens***  
**for**  
***API Calls !***



# Get Access Token for API-Call

```
const CallCustomApi = ({ apiName }) => {
  const { instance, accounts } = useMsal();
  const account = accounts[0];
  const resource = protectedResources[apiName];

  const getAndUseToken = async () => {
    if (!account || !resource) return;

    const tokenRequest = {
      scopes: resource.scopes, // e.g., ["api://<client-id>/read_access"]
      account: account,
    };

    try {
      const response = await instance.acquireTokenSilent(tokenRequest);
      const accessToken = response.accessToken;

      const apiResponse = await fetch(resource.endpoint, {
        headers: {
          'Authorization': `Bearer ${accessToken}`, // Crucial header for API access
        },
      });

      const data = await apiResponse.json();
      console.log(`API response for ${apiName}:`, data);
    } catch (error) {
      console.warn("Silent token acquisition failed. Initiating redirect.");
      instance.acquireTokenRedirect(tokenRequest).catch(e => console.error(e));
    }
  };

  return (
    <button onClick={getAndUseToken}>
      Call {apiName} API
    </button>
  );
};
```

**Protecting routes with**  
**with**  
*MsalAuthenticationTemplate*



# Protecting pages !

```
// pages/ProtectedPage.jsx
import React from 'react';
import { MsalAuthenticationTemplate } from '@azure/msal-react';
import { InteractionType } from '@azure/msal-browser';
import ProfileData from '../components/ProfileData';
```

```
const ProtectedPage = () => {
  const authRequest = {
    scopes: ['User.Read'],
  };
};
```

```
  return (
    <MsalAuthenticationTemplate
      interactionType={InteractionType.Redirect}
      authenticationRequest={authRequest}
    >
      <ProfileData />
    </MsalAuthenticationTemplate>
  );
};
```

```
export default ProtectedPage;
```

← triggers login if user is not logged in!



# React MSAL Essentials

---

## React MSAL Essentials



### AuthenticatedTemplate

- Shows content **only if logged in**
- Use for **conditional UI**



### MsalAuthenticationTemplate

- Shows content **and triggers login if needed**
- Use for **protecting full pages**



### useMsal()

- Gives access to **MSAL instance + accounts + status**

# Handling Msal Event



# Handling MSAL events



```
import { useEffect } from 'react';
import { useMsal } from '@azure/msal-react';
import { EventType } from '@azure/msal-browser';

const MsalEventLogger = () => {
  const { instance } = useMsal();

  useEffect(() => {
    const callbackId = instance.addEventCallback((message) => {
      // Handle the event
      if (message.eventType === EventType.LOGIN_SUCCESS) {
        console.log("Login successful! Welcome.");
      }
    });

    return () => {
      if (callbackId) {
        instance.removeEventCallback(callbackId);
      }
    };
  }, [instance]);

  return null;
};
```



**But...**

**How do we share authentication  
across micro-frontends?**



# Challenge

## ⚠ Key Risks:

Different Login Status

Multipl

sync

Goal: SSO



# Challenge

## ! Key Risks:

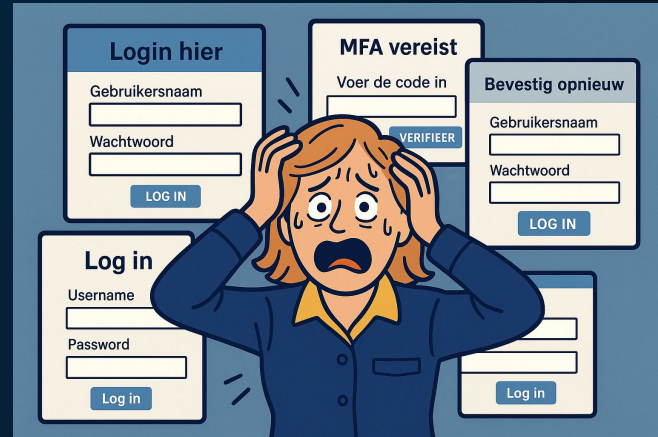
Different Login Status

Multiple Sync

Across Different Domains

Security Gaps

Goal: SSO



**AUTH BYPASS**





# **The Solution: Share MSAL Packages**

---

# Module Federation Solution

## Sharing MSAL Packages in config **HOST**

**Key Concept:**  
Shared MSAL packages  
ensure **single instance** !

```
const { withNativeFederation, shareAll } = require('@softarc/native-federation/build');

module.exports = withNativeFederation({
  name: 'host',

  shared: {
    react: { singleton: true, strictVersion: true },
    "react-dom": { singleton: true, strictVersion: true },
    "@azure/msal-react": { singleton: true, strictVersion: true },
    "@azure/msal-browser": { singleton: true, strictVersion: true }
  },
});
```

# Module Federation Solution

## Sharing MSAL Packages in **REMOTE** config

Key Concept:

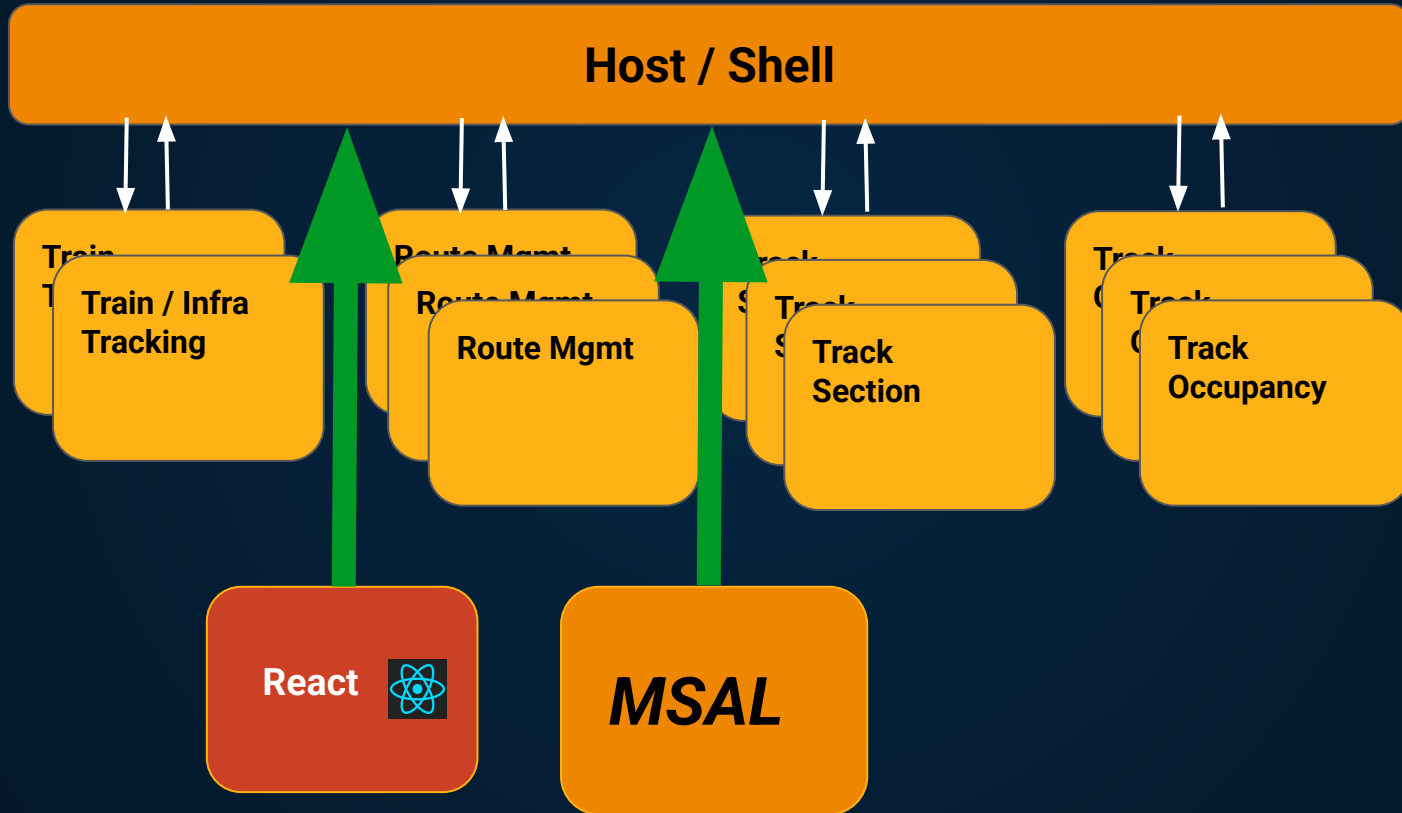
Shared MSAL packages  
ensure **single instance** !



```
const { withNativeFederation, shareAll } = require('@softarc/native-federation/build');

module.exports = withNativeFederation({
  name: 'remote',
  exposes: {
    './remote-app': './src/App.tsx',
  },
  shared: {
    react: { singleton: true, strictVersion: true },
    "react-dom": { singleton: true, strictVersion: true },
    "@azure/msal-react": { singleton: true, strictVersion: true },
    "@azure/msal-browser": { singleton: true, strictVersion: true }
  },
});
```

# Module Federation *solved multiple login !*







**Username: admin**  
**password: admin**



**Username: KoLpVXriw**  
**password: l\*\$j">?ui\$5**



# How to config SSO?

---



# Same auth-config for all Micro frontends !

```

● ● ●

// Config object to be passed to Msal on creation
export const msalConfig: Configuration = {
  auth: {
    clientId: "b5c2e510-4a17-4feb-b219-e55aa5b74144",
    authority: "https://login.microsoftonline.com/<TENANT-ID>",
    redirectUri: "/",
  },
};

export const loginRequest: PopupRequest = {
  scopes: ["User.Read"],
};

export const protectedResources = {
  todoListApi: {
    endpoint: "YOUR_API_ENDPOINT_URL",
    scopes: ["api://YOUR_CLIENT_ID/access_as_user"],
  },
};

```



# Same OIDC Module - The Foundation for OIDC

```
@NgModule({ declarations: [], imports: [MsalModule], providers:  
[provideHttpClient(withInterceptorsFromDi())] })  
export class OidcModule {
```

```
  static forRoot(): ModuleWithProviders<OidcModule> {  
    return {  
      ngModule: OidcModule,  
      providers: [  
        {  
          provide: HTTP_INTERCEPTORS,  
          useClass: MsalInterceptor,  
          multi: true,  
        },  
        {  
          provide: MSAL_INSTANCE,  
          useFactory: MSALInstanceProvider,  
          deps: [APP_CONFIG],  
        },  
        {  
          provide: MSAL_GUARD_CONFIG,  
          useFactory: MSALGuardConfigProvider,  
          deps: [APP_CONFIG],  
        },  
        {  
          provide: MSAL_INTERCEPTOR_CONFIG,  
          useFactory: MSALInterceptorConfigProvider,  
          deps: [APP_CONFIG],  
        },  
        MsalService,  
      ],  
    };  
  }
```



# Same MSAL Instance config!

```
export function MSALInstanceProvider(config: AppConfig): IPublicClientApplication {
  const redirectUri = `${location.origin}${location.pathname}`;
  return new PublicClientApplication({
    auth: {
      clientId: config.getMsalConfig().clientId,
      authority: config.getMsalConfig().authority,
      redirectUri: redirectUri,
      postLogoutRedirectUri: redirectUri,
    },
    cache: {
      cacheLocation: BrowserCacheLocation.LocalStorage,
      storeAuthStateInCookie: isIE,
    },
  });
}
```

# HTTP Interceptor Configuration

---



Automatic Token Injection:

```
export function MSALInterceptorConfigProvider(config: AppConfig): MsalInterceptorConfiguration {
  const protectedResourceMap = new Map<string, Array<string>>();
  protectedResourceMap.set("https://testprorail.nl/users", [config.getMsalConfig().scopeUri]);
  protectedResourceMap.set("https://testprorail.nl/orders", [config.getMsalConfig().scopeUri]);
  return {
    interactionType: InteractionType.Redirect,
    protectedResourceMap,
  };
}
```

**Magic:** Automatically adds Authorization: Bearer <token> header

# SSO Checklist !

---

 Same auth config

 Same Identity Provider (EntraID)

 Shares MSAL Package and storage

 Single Token Refresh Strategy

# **Demo Login**

## **Micro frontends !**





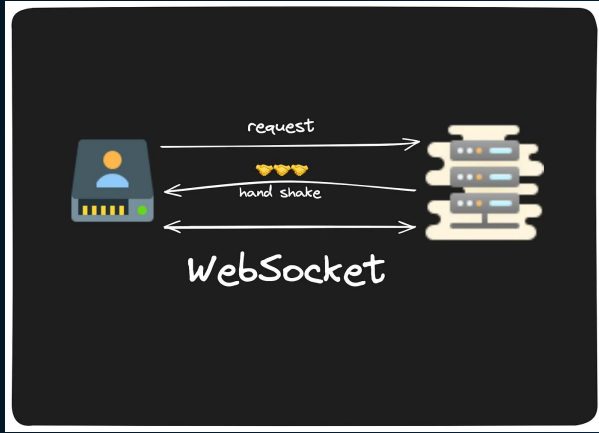
# Websockets for real time data

---



# What are Websockets ?

---



**Network protocol**

**Two-way communication**

**Data is pushed to client**

**Stays Open/no reconnect**



# WebSocket Authentication Challenge #3

---

**Problem → Websockets do not support HTTP headers for tokens !**



```
// WebSockets don't support HTTP headers
const ws = new WebSocket('wss://api.example.com/socket');
// How do we authenticate this connection?
```



# WebSocket Authentication Challenge *Solved!*

---

**Solution → Custom authentication handling !**

```
const token = await instance.acquireTokenSilent(tokenRequest);  
const protocols = ['v10.stomp'];  
if (token) {  
    protocols.push('Bearer', token.accessToken);  
}  
new WebSocket(baseUrl, protocols);
```

# Final Checkoff

---

## Authentication & Security (MSAL & Azure AD B2C):

- ✓  Login once — access everything securely
- ✓  Share authentication across all apps
- ✓  Auto-encrypt data — use secure WebSockets for real-time data

**Mission Completed!**

# ***Final Challenge :***

**Kubernetes** Config

— One App, Many Environments?

---



# MSAL app-config example

```

● ● ●

// Config object to be passed to Msal on creation
export const msalConfig: Configuration = {
  auth: {
    clientId: "b5c2e510-4a17-4feb-b219-e55aa5b74144",
    authority: "https://login.microsoftonline.com/<TENANT-ID>",
    redirectUri: "/",
  },
};

export const loginRequest: PopupRequest = {
  scopes: ["User.Read"],
};

export const protectedResources = {
  todoListApi: {
    endpoint: "YOUR_API_ENDPOINT_URL",
    scopes: ["api://YOUR_CLIENT_ID/access_as_user"],
  },
};

```



# Kubernetes ConfigMap

---

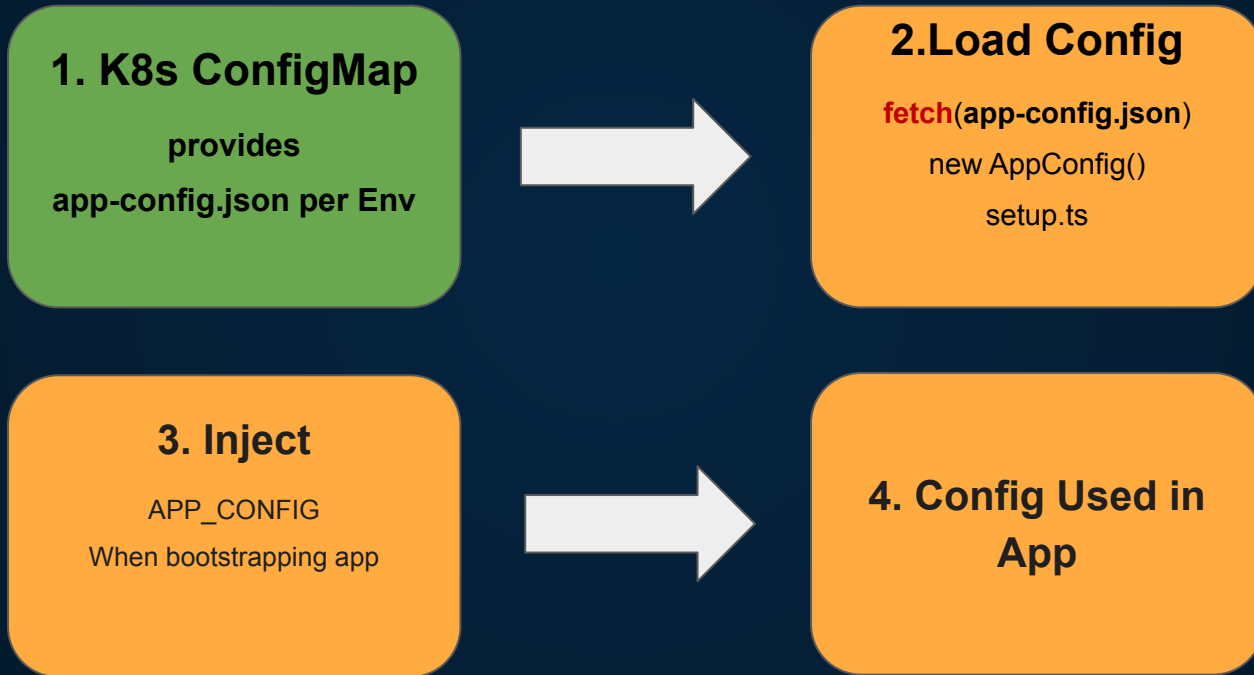


```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  app-config.json: |
    { "env": "production", "clientId": "XYZ" }
```



# Runtime Config Flow (High-level)

---



# Kubernetes advantages!

---



One app on different environments



Settings in Kubernetes, not in the app

change config without rebuilding

**One app, many environments — no rebuilds**



# REAL WORLD APP

---

Trackviewer is micro frontend-based app for the Dutch Railway. With Trackviewer you can see at a glance where exactly trains are, what the delay is and from which track they will disappear.

# Remember Petra ?

---



## “Call to Action” :

---

Claim **one whole day**,

Try **Module Federation**  
and implement **MSAL** for seamless  
*OIDC* flows

## “Call to Action” :

---

Claim **one** whole

Try **M** **ation**

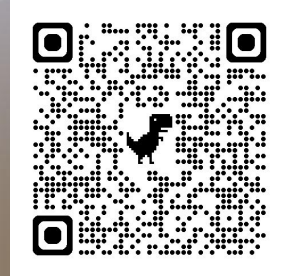
and in **SAL** for seamless  
OIDC flows

**Thank you !**



**Peter Eijgermans**

**CodeSmith / Architect  
Full-Stack**



**Peter Eijgermans**







# Bootstrapping

---





# Bootstrapping the Shell Application

```

// app.module.ts - Shell Application
@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    OidcModule.forRoot(), // Our custom OIDC module
    AppRoutingModule,
    // Module Federation remotes loaded here
  ],
  providers: [
    // MSAL providers configured in OidcModule
  ],
  bootstrap: [
    AppComponent,
    MsalRedirectComponent // Handle OAuth redirects
  ]
})
export class AppModule {}
```

**Key Point:**  
MsalRedirectComponent handles OAuth  
callback



# Silent First, Interactive Fallback

```
// Silent token acquisition (preferred)
try {
  return await firstValueFrom(
    this.authService.acquireTokenSilent({...authRequest, scopes, account})
  );
} catch (error) {
  // Interactive token acquisition (fallback)
  await this.waitForInteractionToComplete();
  return await this.acquireTokenInteractively(authRequest, scopes);
}

private async waitForInteractionToComplete(): Promise<void> {
  return new Promise((resolve) => {
    this.msalsBroadcastService.inProgress$
      .pipe(
        filter((status: InteractionStatus) => status === InteractionStatus.None),
        take(1)
      )
      .subscribe(() => resolve());
  });
}
```

**Strategy:**  
Always try silent first, fallback to interactive



# Security Best Practices

---



## What We're Doing Right:

- **Secure Token Storage:** LocalStorage with proper cache management
- **Scope Validation:** Backend validates audience claims
- **Token Expiration Handling:** Automatic refresh with fallback
- **HTTPS Everywhere:** Non-negotiable for production



## Watch Out For:

- **Token Leakage:** Never log tokens or put them in URLs params
- **Scope Creep:** Request minimal necessary permissions
- **Cross-Origin Issues:** Proper CORS configuration
- **Token Sharing:** Don't share tokens between domains
- **Refresh Token Rotation:** Handle refresh token expiration

# Common Problems & Solutions:

---

## 1. "MSAL instance not initialized"

```
// Ensure initialization in BearerTokenService  
  
await this.authService.instance.initialize();
```

## 2. Module Federation version conflicts

```
// Use singleton and strict versions  
  
shared: {  
  
  '@azure/msal-react': { singleton: true, strictVersion: true }  
  
}
```

# Common Problems & Solutions:

---

## 3. WebSocket authentication failures

typescript

```
// Check token expiration before sending

if (tokenResult.expiresOn < new Date()) {

    tokenResult = await this.bearerTokenService.acquireToken();

}
```

## 4. Redirect loops

typescript

```
// Ensure proper redirect URI configuration

redirectUri: `${location.origin}${location.pathname}`
```

# Key takeaways

---

 **Security:** MSAL simplifies the complexity of implementing OAuth/OIDC

 **Scalability:** Module Federation enables distributed auth

 **Performance:** Shared packages, single MSAL instance

 **Flexibility:** *Custom services* for WebSocket auth

 **Maintainability:** Environment-specific configurations in K8s



# Custom Token Service for WebSockets

```

@Inject({ providedIn: 'root' })
export class BearerTokenService {
  constructor(
    private readonly authService: MsalService,
    private readonly appConfigService: AppConfigService,
    private readonly msalBroadcastService: MsalBroadcastService,
    @Inject(MSAL_GUARD_CONFIG) private readonly msalGuardConfig: MsalGuardConfiguration
  ) {}

  public async acquireToken(): Promise<AuthenticationResult> {
    await this.authService.instance.initialize();
    const authRequest = this.getAuthRequest();
    const scopes = [this.appConfigService.getSettings('scopeUri')];
    const account = this.authService.instance.getAllAccounts()[0];

    try {
      // Try silent token acquisition first
      return await firstValueFrom(
        this.authService.acquireTokenSilent({...authRequest, scopes, account})
      );
    } catch (error) {
      // Fall back to interactive if silent fails
      return await this.acquireTokenInteractively(authRequest, scopes);
    }
  }
}
```



Grants access to **protected APIs**  
Grants access to **protected APIs**

# OIDC Flow: 📄 Getting the golden ticket

---

We have 3 Types of Tokens:

- 📄 **Access Token**: "Grants access to protected APIs" - Short-lived, contains permissions/scopes (JWT format)
- ID **ID Token**: Contains user identity claims / info
- ↺ **Refresh Token**: "I can get new tickets" - Long-lived, used to get new Access and ID tokens




# The Magic of Injection Tokens

---

The `APP_CONFIG` injection token acts as a bridge between your external configuration (from K8s) and Angular's DI system

# The Magic of Injection Tokens

## Step-by-step flow:

1.  **Kubernetes ConfigMap** → Provides `app-config.json`
  - Contains environment-specific config (clientId, authority, scopes)
2.  **Docker Container Setup**
  - ConfigMap mounted as volume into nginx container
  - Path: `/usr/share/nginx/html/assets/app-config.json`
3.  **Browser Runtime**
  - Angular app loads in browser
  - Fetches `./assets/app-config.json` via HTTP
4.  **Angular Injection**
  - Config loaded into `APP_CONFIG` injection token
  - MSAL receive config via dependency injection



# Protecting Routes with MSAL Guard

```
export function MSALGuardConfigProvider(config: AppConfigService): MsalGuardConfiguration {
  return {
    interactionType: InteractionType.Redirect,
    authRequest: {
      // Scopes define what permissions we're requesting
      // Format: api://clientID/scope-name
      scopes: [config.getSettings('scopeUri')], // api://client-id/.default
    },
  };
}

// Usage in routing
{
  path: 'dashboard',
  component: DashboardComponent,
  canActivate: [MsalGuard] // Automatically redirects if not authenticated
}
```

# MSAL coding - Key Takeaways!

---

✓ **Share MSAL packages with *Module Federation***

✓ **OIDC Module configures MSAL for React**

✓ **MSALInterceptorConfigProvider contains list protected resources**

✓ **Custom Token Service for WebSockets**

# Challenge 4: How to Solve the Multi-Environment Authentication ?

---



## **Problem:**

**Hard-coded authentication  
configuration**

Different environments need  
different MSAL settings

## **Traditional Approach:**

**Build per Environment  
Hard-coded config  
Deployment complexity**

## **Solution:**

**Single build artifact  
Dynamic configuration  
*Kubernetes Configmaps*  
Type-safe injection**



# Micro frontends - Key Takeaways!

---



Understand the Micro frontend Architecture



Host vs. Remote applications



Configure and *Share* resources with *Module Federation*



klossnet

"WELL, THEY BANNED PASSWORD RE-USE.  
WHAT DO YOU EXPECT ME TO DO?"

### Step 1: Kubernetes ConfigMap

- 📄 Stores the app's config
  - 📄 app-config.json
- 🔑 Settings: clientId, authority, scopes

### Step 2: Docker Container

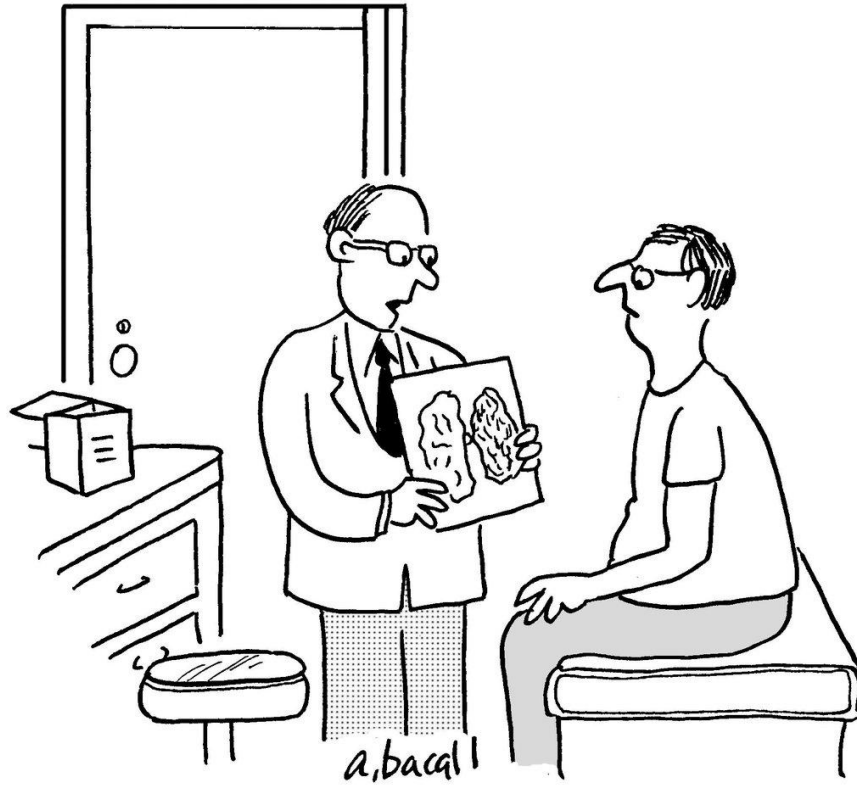
- 📁 Config file placed inside container
  - 📍 Location:  
/usr/share/nginx/html/assets/app-config.json

### Step 3: Browser Loads App

- 🌐 User opens the app
- 📄 App fetches config file
  - 📁 From: ./assets/app-config.json

### Step 4: Angular Uses Config

- ⚙️ Angular reads & stores config
- 🔑 MSAL library uses config
- 🔑 Handles authentication



**"I have the MRI scan of your brain. The right hemisphere is clogged with computer passwords."**