

# From Zero to SSO Hero 🦸

OIDC, Authorization, and Keycloak Workshop

# Agenda

- **Introduction** (5min)
- **OIDC & OAuth2 Fundamentals** (45min)
- **System Setup Overview** (10min)
- **Hands-on Session** (60min)
- **Bonus Challenge: Hack the Super-Secret Backend** (???)
- **Discussion & Networking**

# Why

Let's face it - most authentication setups in our company are inherited, rarely revisited, and often become siloed knowledge over time. This makes them harder to understand, improve, and modernize, potentially creating gaps in security or user experience. 🗝️

# About This Workshop

This workshop will:

- Demystify authentication and authorization concepts
- Provide hands-on experience with Keycloak and OIDC
- Guide you through implementing secure applications

**By the end:** You'll understand how to implement and secure applications using industry-standard protocols.

# Part 1: Fundamentals

## Authentication

- Verifies **WHO** you are
- "Are you really who you claim to be?"
- Examples: Username/password, biometrics, MFA

## Authorization

- Determines **WHAT** you can do
- "Are you allowed to access this resource?"
- Examples: Role-based access, permissions

## OAuth 2.0:

- Authorization framework for third-party apps
- Uses tokens (not credentials)
- Separates authentication from authorization
- Multiple "grant types" (flows)

## OIDC (OpenID Connect)

- Identity layer built on OAuth 2.0
- Adds authentication & user profile info
- Introduces the **ID Token**
- Enables Single Sign-On (SSO)



# [INTERACTIVE MOMENT]

## JWT Live Demo

Let's paste a token into [jwt.io](https://jwt.io)

- What info can you see?
- What can't you change?

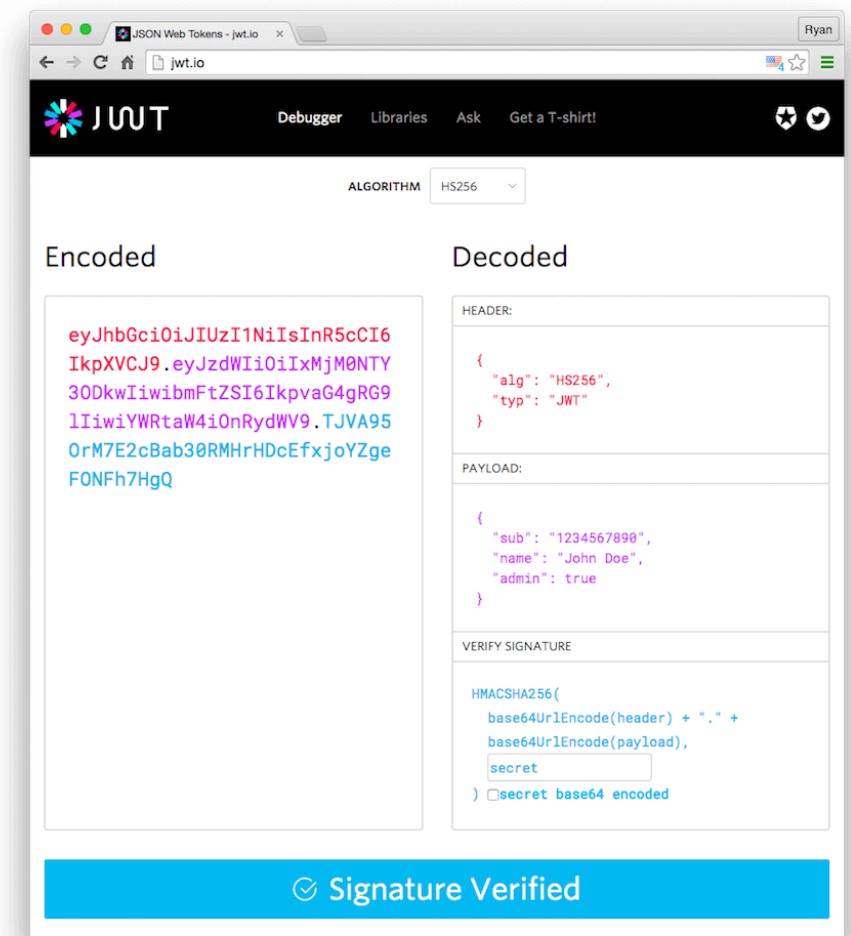
# JWT: Not Just JSON

JSON Web Token (JWT) consists of 3 parts:

- **Header:** Algorithm & token type
- **Payload:** Claims (data)
- **Signature:** Ensures integrity

## Why secure?

- Signed using a secret key from IDP
- Can be verified by the client without contacting the IDP using the IDPs JSON Web Key Set (JWKS) public keys
- Tamper evident due to signature











# What is Keycloak?

- Open-source **Identity and Access Management** by Red Hat
- Provides SSO, User management, Social login, Multifactor authentication, Fine grained authorization, Admin console, REST API, ...
- in OIDC: Keycloak is an identity provider (IDP), which provides user authentication functions to other apps called relying parties (RP)



# Keycloak Terminology

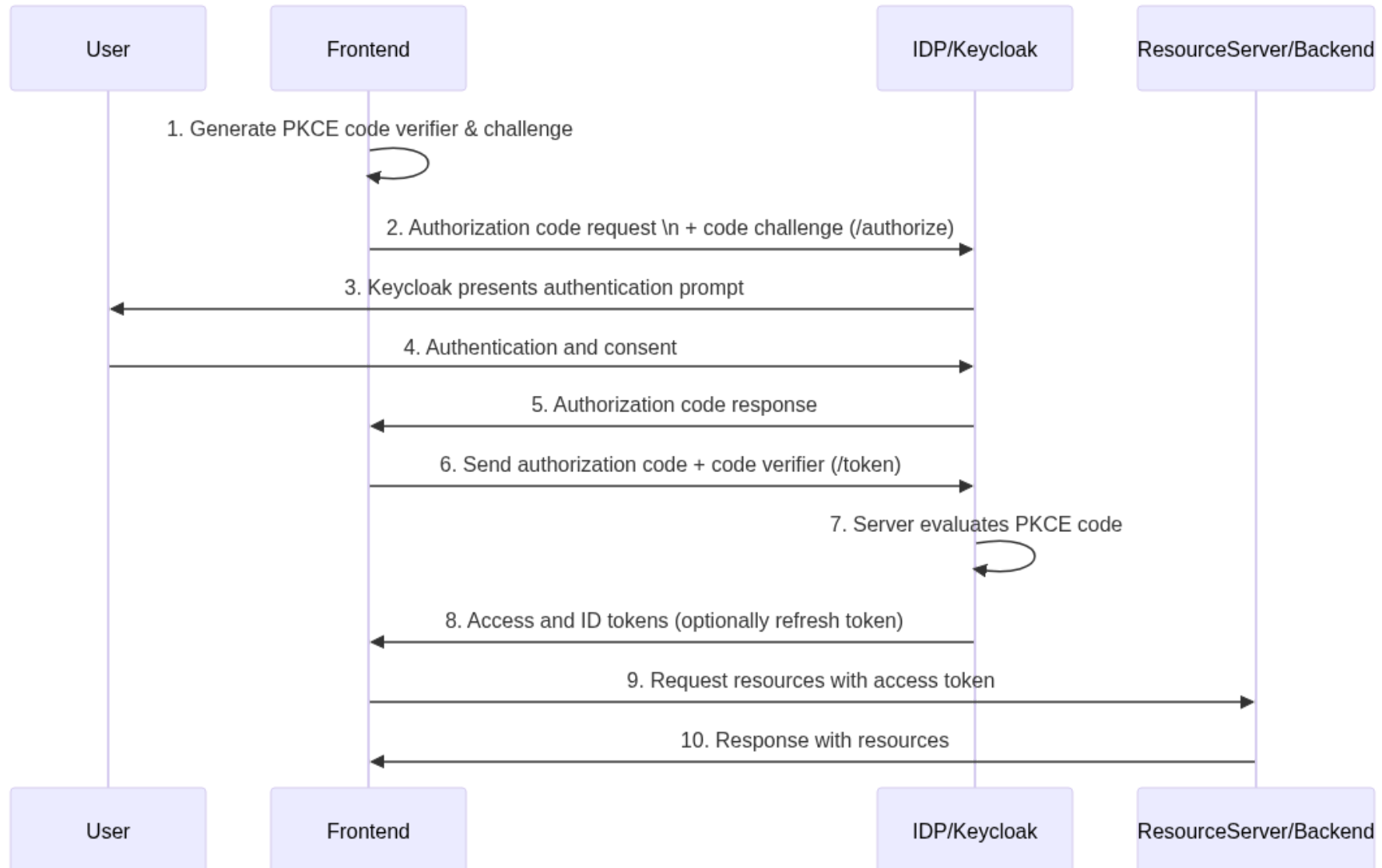
-  **Realm:** A security domain (isolated user base)
-  **Client:** An application that uses Keycloak for auth
-  **Client Roles:** Permissions specific to an application
-  **Users:** People and other Entities that can authenticate
-  **Groups:** Collections of users
-  **Identity Providers:** External auth sources (Google, Facebook)

# OIDC Flows: Authorization Code with PKCE

## PKCE (Proof Key for Code Exchange)

- Enhanced security for public clients
- Prevents authorization code interception
- Uses a code verifier and code challenge
- Recommended for all clients, especially SPAs and mobile apps

# Authorization Code Flow with PKCE



# [INTERACTIVE MOMENT]

## Authorization Code Flow Roleplay

*Assign participants:*

- User
- Frontend
- Keycloak
- Backend

And walk through the sequence!

# Redirect URIs and Web Origins

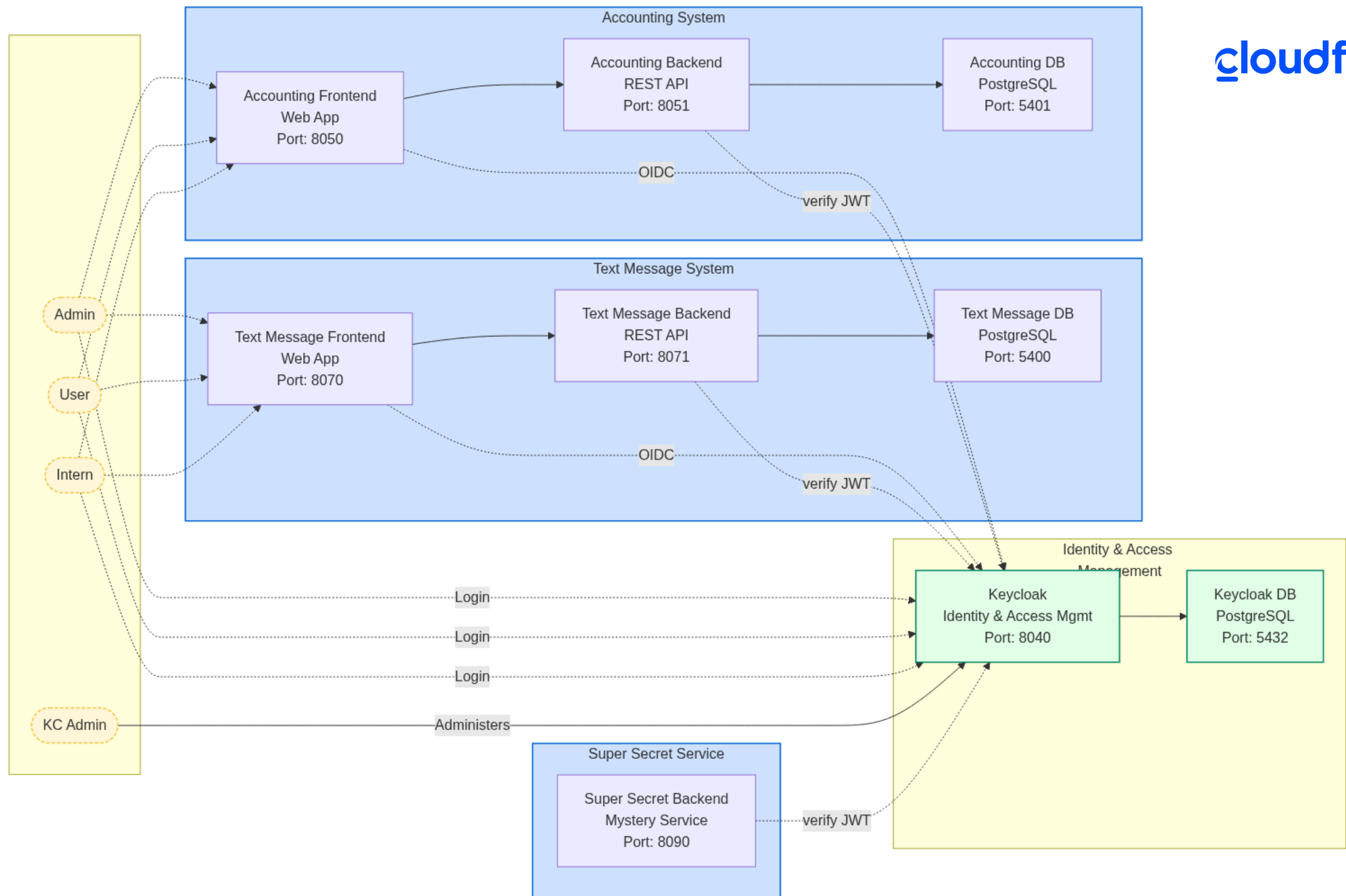
## **Redirect URIs**

- Where Keycloak sends the user after authentication
- Must be preregistered for security
- Must match exactly (including trailing slashes, KC supports wildcards)

## **Web Origins (CORS)**

- Domains allowed to make AJAX requests to Keycloak
- Prevents cross site request forgery
- Manifests as CORS errors when misconfigured

## Part 2: System Overview





# [INTERACTIVE MOMENT]

## Repository Overview

*Walkthrough:*

- Repository Structure
- Show `readme.md` and `docs`
- Docker Compose Setup and started Containers
- Accounting Backend and Frontend are missing Implementations, look for `TODD` s

# Implementation Approaches

## ✅ Best Practice

- Use established libraries:
  - `keycloak.js` for JS Frontend
  - `oauth2ResourceServer` for Spring Backend
- many Benefits: Silent refresh, Token management, Standardized security, low Maintenance

## 😞 Our Workshop Approach

- Low-level implementation with `oauth4webapi`
- Custom `SecurityFilter` in backend
- Helps understand the underlying mechanisms

# Endpoint Security vs. Method Security

## Endpoint Security

- Secures entire API endpoints
- Coarse grained control
- Implemented at the web layer

## Method Security

- Secures individual service methods
- Additional control
- Implemented at the service layer

## Part 3: Hands-on Session

### Your Mission

Complete the accounting service implementation:

- Set up Keycloak realm, client, roles and assign them
- Connect the frontend to Keycloak
- Implement security in the backend

**Success Criteria:** Admin can add account entries, users can view them, and interns have no access.

# Keycloak Setup Tasks

1. Add a new client in the e-corp realm
2. Add client roles (need to be exact, or you need to change the impl.):
  - VIEW\_ACCOUNT\_INFO
  - EDIT\_ACCOUNT\_INFO
3. Assign roles to users:
  - Admin: EDIT and VIEW roles
  - User: VIEW only
  - Intern: No roles
4. Configure client settings:
  - Valid redirect URIs
  - Web origins

# Backend Implementation Tasks

1. Complete the `JWTSecurityFilter` :

- Validate tokens
- Extract user roles
- Set security context

2. Register the filter in `WebSecurityConfig`

3. Secure endpoints and service methods:

- Data modification → WRITE rights
- Data retrieval → READ rights

**TIP:** Look for `TODO` comments in the code

# Frontend Implementation Tasks

1. Complete the initial login request
2. Register and implement the logout callback
3. Complete session handling in `loadSession()`
4. Fix the `config.json` file

**TIP:** Look for `TODO` comments in the code

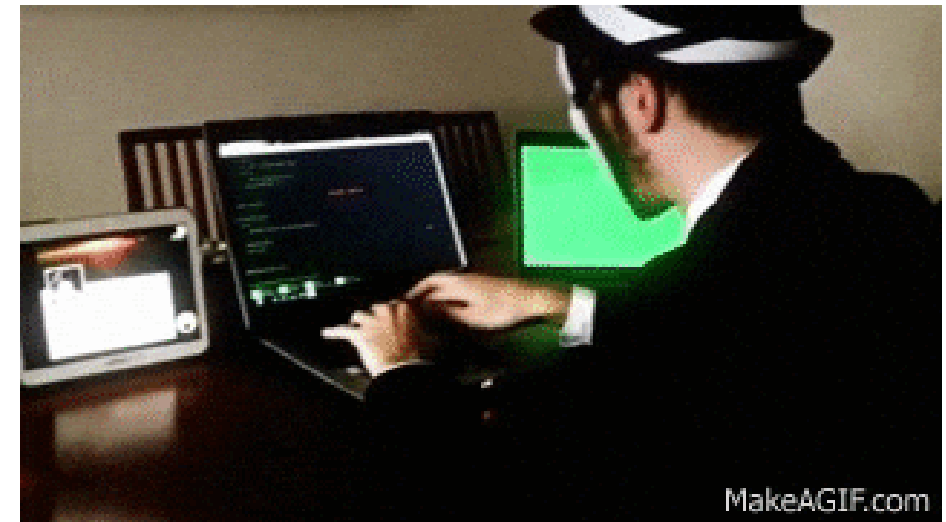
## Implementation Help

- All tests should pass when you're done
- If you are stuck, use the text message service as a reference
- Take a look at the `readme.md`



## Bonus Challenge: Hack the Super-Secret Backend

- The super-secret backend runs on port `8090`
- Your goal: Successfully authenticate against its root endpoint
- Hint: It uses our Keycloak's JWKS for JWT verification



## Discussion & Wrap-up

- What challenges did you face?
- How would you implement this in your own projects?
- What security considerations are most important?
- How does this compare to other auth systems you've used?

# Extra Slides

# Common Pitfalls: Domain Names

## The Problem:

- Browser uses `localhost` to access Keycloak
- Container services use internal names (e.g., `keycloak`)
- JWT issuer URL must match exactly for verification

## Solutions:

- Configure proper issuer URL in Keycloak
- Use consistent naming across environments
- Configure token verification to handle different issuer URLs