TECHNICAL REPORT ON DESIGN AND EXTENSION OF OHDSI ATLAS LOGIN SERVICE FOR INSPIRE MENTAL HEALTH PROJECT.

DEVELOPED BY
WLADEK AIRO (Consultant)

REVIEWED BY
MICHAEL OCHOLA (Data Scientist @Data Science Programs)

Contents

1.0 Background of the Project

The Implementation Network for Sharing Population Information from Research Entities (INSPIRE) recruited an experienced and qualified java developer to implement a robust login system using Apache Shiro framework.
The primary goal was to streamline the login process, and alleviate the team from the experienced authentication challenges. The intention was to seamlessly integrate this login system into the existing OHDSI Atlas application running on AWS cloud, augmenting functionalities related to user authentication, authorization, and access control. The design currently supports:

- User sign up and validation
- Ensures username and passwords are encrypted on transit
- Supports federated access via Google OAuth2
- Follows security best practices

2.0 OHDSI Web Tools set-up and Implementation

2.1 Set-up OHDSI Tools stack using independent software tools

2.1.1 Installing Java

Step-by-Step Guide

1. Download Java:
    - Go to the official Java SE Development Kit (JDK) download page.
    - Choose the appropriate version for your operating system (Windows, Linux).
2. Install Java:
    - Windows:
        - Run the downloaded .exe file and follow the installation prompts.
        - Add the JDK bin directory to the system PATH.
    - Linux:
        - Use the package manager for your distribution (e.g., sudo apt install openjdk-11-jdk for Ubuntu).
3. Verify Installation:
    - Run the command: java -version
    - Ensure it displays the installed Java version.
4. Set JAVA_HOME Environment Variable:
    - Windows:
        - Open System Properties > Advanced > Environment Variables.
        - Add a new system variable JAVA_HOME pointing to the JDK installation directory.

- Linux:
  - Add the following line to your shell configuration file (~/.bashrc, ~/.zshrc, etc.):

2.1.2 Install Git

Step-by-Step Guide

1. Download Git:
   - Go to the official Git website.
2. Install Git:
   - Windows:
     - Download the installer and run it.
     - Follow the setup instructions, leaving default options unless specific configurations are needed.
   - Linux:
     - Use the package manager for your distribution:
     - sudo apt update && sudo apt install git # Debian/Ubuntu

3. Verify installation git --version

2.1.3 Install Apache Maven

Step-by-Step Guide

1. Download Maven:
   - Go to the Apache Maven download page.
   - Choose the binary zip archive or tar.gz archive suitable for your operating system.
2. Extract the Archive:
   - Windows:
     - Extract the downloaded zip file to a directory of your choice.
   - Linux : *tar xzvf apache-maven-3.x.y-bin.tar.gz*
3. Set Environment Variables
   - Windows
     - Add the MAVEN_HOME environment variable pointing to the Maven directory.
     - Add %MAVEN_HOME%\bin to the system PATH.
   - Linux

        - Add the following to your shell configuration file (~/.bashrc, ~/.zshrc, etc.):
        - export MAVEN_HOME=/path/to/apache-maven export
        - PATH=$MAVEN_HOME/bin:$PATH
        - Reload as : source ~/.bashrc

2.1.4 Install Apache Tomcat

Step-by-Step Guide

1. Download Tomcat:
    - Go to the Apache Tomcat download page.
    - Choose the binary distribution suitable for your operating system (e.g., tar.gz for Unix/Linux or zip for Windows).
2. Extract the Archive:
    - Windows:
        - Extract the downloaded zip file to a directory of your choice.
    - Linux:
        - Extract : tar xzvf apache-tomcat-9.x.xx.tar.gz
3. Set Environment Variables:
    - Windows**:**
        - Add the CATALINA_HOME environment variable pointing to the Tomcat directory.
    - Linux:
        - export CATALINA_HOME=/path/to/apache-tomcat export
        - PATH=$CATALINA_HOME/bin:$PAT
        - Reload : source ~/.bashrc
4. Start Tomcat:
    - Windows:
        - Run startup.bat located in the bin directory.
    - Linux:
        - Run startup.sh located in the bin directory.
5. Verify Installation
    - http://localhost:8080.

2.1.5 Install Postgres

1. Install PostgreSQL:
    - Windows:

        - Download the installer from the PostgreSQL website.
        - Run the installer and follow the installation wizard.
    - Linux:
        - sudo apt update && sudo apt install postgresql postgresql-contrib
        - sudo service postgresql initdb # or sudo postgresql-setup initdb
        - sudo systemctl start postgresql
        - sudo systemctl enable postgresql

2.1.6 Configure Environment variables

1. Edit Shell Configuration File:
    - Linux :
        - For Bash, edit ~/.bashrc or ~/.bash_profile.
        - export JAVA_HOME=/path/to/java
        - export MAVEN_HOME=/path/to/maven
        - export CATALINA_HOME=/path/to/tomcat
        - export PATH=$JAVA_HOME/bin:$MAVEN_HOME/bin:$CATALINA_HOME/bin:$PATH
        - source ~/.bashrc

2.1.7 Database Preparation

Step-by-Step Guide

1. Switch to PostgreSQL User:
    - sudo -i -u postgres
2. Create a New Database:
    - createdb ohdsi_db
3. Create a new user
    - createuser ohdsi_user
    - ALTER USER ohdsi_user WITH ENCRYPTED PASSWORD 'yourpassword';
    - GRANT ALL PRIVILEGES ON DATABASE ohdsi_db TO ohdsi_user;
4. Create schema
    - psql -d ohdsi_db
    - CREATE SCHEMA cdm;
    - CREATE SCHEMA results;

    - CREATE SCHEMA vocab;
5. Load CDM
6. Configure datasource in WebAPI:

2.1.8 WebAPI Installation and Set-up

Step-by-Step Guide

1. Clone the WebAPI Repository:
    - git clone https://github.com/OHDSI/WebAPI.git
    - cd WebAPI
2. Configure Database Connection:
    - datasource.driverClassName=org.postgresql.Driver
    - datasource.url=jdbc:postgresql://localhost:5432/ohdsi_db
    - datasource.username=ohdsi_user
    - datasource.password=yourpassword
3. Build the project
    - mvn clean package
4. Deploy to Tomcat – Copy the generated WebAPI.war file from target/ to the Tomcat webapps directory:
    - cp target/WebAPI.war $CATALINA_HOME/webapps/
5. Configure Tomcat – Edit conf/server.xml in Tomcat to include a new context:
    - <Context docBase="WebAPI" path="/WebAPI" reloadable="true" />
6. Start Tomcat:
    - $CATALINA_HOME/bin/startup.sh
7. Verify Deployment
    - $CATALINA_HOME/bin/startup.sh

2.1.9 CDM configuration

Step-by-Step Guide:

1. Install SQL Scripts for CDM:
    - Download the CDM version-specific SQL scripts from the OHDSI GitHub repository.
2. Load CDM Structure:
    - Execute the SQL scripts to create the CDM schema and tables. Connect to the PostgreSQL database using a client like psql or any other SQL client tool.

- psql -U ohdsi_user -d ohdsi_db -f /path/to/cdm_structure.sql

3. Load Vocabulary Data:pg_bulkload /path/to/vocabulary_csv_files
4. Laod sample data : psql -U ohdsi_user -d ohdsi_db -f /path/to/sample_data.sql
5. Configure CDM source in WebAPI
    - INSERT INTO source (source_id, source_name, source_key, source_connection, source_dialect)
    - VALUES (1, 'CDM PostgreSQL', 'cdm_postgresql', 'jdbc:postgresql://localhost:5432/ohdsi_db', 'postgresql');
    -
    - INSERT INTO source_daimon (source_daimon_id, source_id, daimon_type, table_qualifier)
    - VALUES (1, 1, 0, 'cdm'),
    - (2, 1, 1, 'vocab'),
    - (3, 1, 2, 'results');
6. Restart WebAPI tomcat
    - $CATALINA_HOME/bin/shutdown.sh
    - $CATALINA_HOME/bin/startup.sh

2.1.10 ATLAS installation and set-up

Step-by-Step Guide:

1. Install Node.js and npm:
    - ATLAS requires Node.js and npm (Node Package Manager) for installation. Download and install them from the official Node.js website.
2. Windows:
    - Download and run the Node.js installer from the Node.js website.
    - Follow the installation prompts and ensure npm is installed along with Node.js.
3. Linux:
    - sudo apt update
    - sudo apt install nodejs npm
    - node -v
    - npm -v
    - git clone https://github.com/OHDSI/Atlas.git
    - cd Atlas
    - npm install
    - cp config-local.js.sample config-local.js
        - define([], function () {
        - var configLocal = {};
        -

        - configLocal.webAPIRoot = 'http://localhost:8080/WebAPI/';
        -
        - return configLocal;
        - });
    - Build ATLAS
        - npm install -g grunt-cli
        - grunt
    - Deploy
        - cp -r dist/* /var/www/html/
    - Configure Web Server
        - Apache

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        <Directory /var/www/html>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

        - Nginx

```
server {
        listen 80;
        server_name localhost;

        location / {
        root /var/www/html;
        index index.html index.htm;
        try_files $uri $uri/ =404;
        }
}
```

    - Verify ATLAS
        - http://localhost/

2.1.11 Automated set-up pipeline using Makefile

The above setup files were automated to run using a makefile as shown below.

```
# Variables
JAVA_VERSION=11
POSTGRES_VERSION=13
TOMCAT_VERSION=9.0.54
MAVEN_VERSION=3.8.3
CATALINA_HOME=/opt/tomcat
JAVA_HOME=/usr/lib/jvm/java-$(JAVA_VERSION)-openjdk-amd64

# Paths
WEBAPI_REPO=https://github.com/OHDSI/WebAPI.git
ATLAS_REPO=https://github.com/OHDSI/Atlas.git

# Database Variables
DB_NAME=ohdsi_db
DB_USER=ohdsi_user
DB_PASS=yourpassword

.PHONY: all install_java install_git install_maven install_tomcat install_postgresql configure_env
cdm_setup webapi_setup atlas_setup

all: install_java install_git install_maven install_tomcat install_postgresql configure_env cdm_setup
webapi_setup atlas_setup

install_java:
    # Install Java
    sudo apt update
    sudo apt install -y openjdk-$(JAVA_VERSION)-jdk
    echo "export JAVA_HOME=$(JAVA_HOME)" >> ~/.bashrc
    echo "export PATH=\$$JAVA_HOME/bin:\$$PATH" >> ~/.bashrc
    source ~/.bashrc

install_git:
```

```
    # Install Git
    sudo apt update
    sudo apt install -y git

install_maven:
    # Install Maven
    wget https://downloads.apache.org/maven/maven-3/$(MAVEN_VERSION)/binaries/apache-maven-$(MAVEN_VERSION)-bin.tar.gz
    sudo tar xzvf apache-maven-$(MAVEN_VERSION)-bin.tar.gz -C /opt
    sudo ln -s /opt/apache-maven-$(MAVEN_VERSION) /opt/maven
    echo "export MAVEN_HOME=/opt/maven" >> ~/.bashrc
    echo "export PATH=\$$MAVEN_HOME/bin:\$$PATH" >> ~/.bashrc
    source ~/.bashrc

install_tomcat:
    # Install Tomcat
    wget https://downloads.apache.org/tomcat/tomcat-9/v$(TOMCAT_VERSION)/bin/apache-tomcat-$(TOMCAT_VERSION).tar.gz
    sudo tar xzvf apache-tomcat-$(TOMCAT_VERSION).tar.gz -C /opt
    sudo ln -s /opt/apache-tomcat-$(TOMCAT_VERSION) $(CATALINA_HOME)
    echo "export CATALINA_HOME=$(CATALINA_HOME)" >> ~/.bashrc
    echo "export PATH=\$$CATALINA_HOME/bin:\$$PATH" >> ~/.bashrc
    source ~/.bashrc

install_postgresql:
    # Install PostgreSQL
    sudo apt update
    sudo apt install -y postgresql-$(POSTGRES_VERSION) postgresql-contrib
    sudo -u postgres psql -c "CREATE DATABASE $(DB_NAME);"
    sudo -u postgres psql -c "CREATE USER $(DB_USER) WITH ENCRYPTED PASSWORD '$(DB_PASS)';"
    sudo -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE $(DB_NAME) TO $(DB_USER);"
    sudo -u postgres psql -c "CREATE SCHEMA cdm;"
    sudo -u postgres psql -c "CREATE SCHEMA results;"
    sudo -u postgres psql -c "CREATE SCHEMA vocab;"
```

```
configure_env:
    # Configure Environment Variables
    echo "export JAVA_HOME=$(JAVA_HOME)" >> ~/.bashrc
    echo "export MAVEN_HOME=/opt/maven" >> ~/.bashrc
    echo "export CATALINA_HOME=$(CATALINA_HOME)" >> ~/.bashrc
    echo "export PATH=\$$JAVA_HOME/bin:\$$MAVEN_HOME/bin:\$$CATALINA_HOME/bin:\$$PATH"
>> ~/.bashrc
    source ~/.bashrc

cdm_setup:
    # Load CDM Structure and Vocabulary
    sudo -u postgres psql -d $(DB_NAME) -f /path/to/cdm_structure.sql
    # Replace with actual vocabulary loading commands
    # Example: pg_bulkload /path/to/vocabulary_csv_files

webapi_setup:
    # Clone and Build WebAPI
    git clone $(WEBAPI_REPO)
    cd WebAPI && mvn clean package
    # Configure Database Connection
    echo "datasource.driverClassName=org.postgresql.Driver" >>
src/main/resources/spring/application.properties
    echo "datasource.url=jdbc:postgresql://localhost:5432/$(DB_NAME)" >>
src/main/resources/spring/application.properties
    echo "datasource.username=$(DB_USER)" >> src/main/resources/spring/application.properties
    echo "datasource.password=$(DB_PASS)" >> src/main/resources/spring/application.properties
    # Deploy WebAPI to Tomcat
    cp target/WebAPI.war $(CATALINA_HOME)/webapps/
    # Configure Tomcat
    echo "<Context docBase=\"WebAPI\" path=\"/WebAPI\" reloadable=\"true\" />" >>
$(CATALINA_HOME)/conf/server.xml
    # Start Tomcat
    $(CATALINA_HOME)/bin/startup.sh

atlas_setup:
```

```
# Install Node.js and npm
sudo apt update
sudo apt install -y nodejs npm
# Clone and Build ATLAS
git clone $(ATLAS_REPO)
cd Atlas && npm install && grunt
# Deploy ATLAS
sudo cp -r dist/* /var/www/html/
# Configure Web Server (if using Apache or Nginx)
# Apache example provided in the text above
# Restart web server
sudo systemctl restart apache2
```

2.2 Set-up OHDSI Tools stack using Docker Technology

Broadsea runs the core OHDSI technology stack using cross-platform Docker container technology.Currently the 3.5 version supports ATLAS setup with user authentication and authorisation. For Broadsea 3.5, you will need Docker version 1.27.0+.

Step-by-step Guide

- Install Docker version 1.27.0+
- git clone https://github.com/OHDSI/Broadsea.git
- docker compose --profile default up -d
- More options supported for advanced customisation

Link to Broadsea Github. We are running the Broadsea within APHRC LAN and only accessible outside the center using a VPN link. We intend to use this approach for a majority of data harmonization use cases. We shall only migrate to AWS for collaborative research purposes in order to be sustainable.

2.3 Set-up OHDSIonAWS using Amazon Web Services (AWS) Stack

AWS supports deployment of OMOP instances using two variation:

i) OHDSI in a Box

ii) OHDSI on AWS

Here we use the production variant that is OHDSIonAWS implementation

We deployed an isolated, three-tier Amazon Virtual Private Cloud (Amazon VPC) with ability to access from the public internet.

Set up has OMOP CDM with vocabulary data on Redshift, Atlas, WebAPI, Achilles, RStudio with PatientLevelPrediction, CohortMethod, and many other R libraries. It supports role-based access control for Atlas and RStudio. WebAPI data is stored on Postgres RDS instance.

It uses data-at-rest and in-flight encryption to respect the requirements of HIPAA. It uses managed services from AWS; OS, middleware, and database patching and maintenance is largely automatic.

It creates automated backups for operational and disaster recovery and the build process happens just within a few hours.

The deployment supports both horizontal and vertical scaling with high availability and geographical redundancy.

3.0 OAuth2 and SSL Setup

Setup SSL first as it is a requirement for OAuth2.0 to work

Step-by-Step Guide for SSL

1. Request an SSL/TLS Certificate from AWS Certificate Manager (ACM)

1. Open the ACM console:
    - Sign in to the AWS Management Console and open the ACM console at https://console.aws.amazon.com/acm/home.
2. Request a certificate:
    - Choose "Request a certificate."
    - Select "Request a public certificate" and choose "Next."
    - Enter your domain name (e.g., yourdomain.com) and any additional names (e.g., www.yourdomain.com).
    - Choose "Next."
3. Validate the domain:
    - Choose a validation method (DNS validation is recommended).
    - Follow the instructions to add a CNAME record to your domain's DNS settings.

    - After the DNS record is added, choose "Continue" and then "Review" and "Request."
4. Wait for validation:
    - ACM will validate the domain. This may take a few minutes. Once validated, the certificate status will change to "Issued."

2. Configure Elastic Load Balancer (ELB) to Use the SSL Certificate

1. Open the EC2 console:
    - Sign in to the AWS Management Console and open the EC2 console at https://console.aws.amazon.com/ec2/.
2. Navigate to Load Balancers:
    - In the left navigation pane, under "Load Balancing," choose "Load Balancers."
3. Select your Load Balancer:
    - Select the load balancer that you want to configure to use SSL.
4. Listeners:
    - Under the "Listeners" tab, choose "View/edit listeners."
5. Add Listener:
    - Choose "Add listener."
    - Select "HTTPS" as the protocol, and set the port to 443.
    - For the "Default SSL certificate," choose "From ACM" and select the certificate you requested.
    - Configure the listener's default action to forward to your target group.
    - Save the configuration.
6. Security Groups:
    - Ensure that the security groups associated with your load balancer allow inbound traffic on port 443.

Step-by-Step Guide for OAuth2 with Google

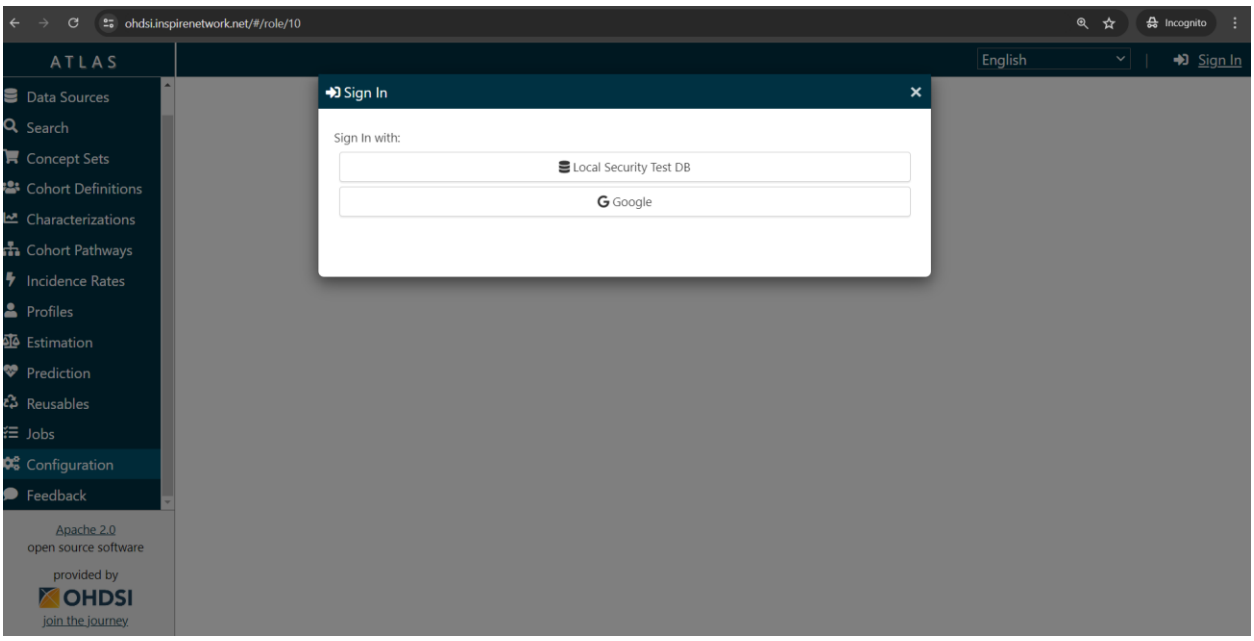
1. Set Up Google OAuth2 Credentials

1. Go to Google Cloud Console:
    - Open the Google Cloud Console at https://console.cloud.google.com/.
2. Create a new project:
    - In the top-left corner, click the project drop-down and select "New Project."
    - Give your project a name and click "Create."
3. Enable OAuth2 APIs:
    - In the left-hand menu, go to "APIs & Services" > "Library."
    - Search for "Google+ API" and "Google Identity Platform" and enable them for your project.
4. Configure OAuth consent screen:
    - Go to "APIs & Services" > "OAuth consent screen."

- Select "External" and click "Create."
- Fill out the required fields, including the application name, support email, and authorized domains.
- Click "Save and Continue" and configure the scopes if necessary.

5. Create OAuth2 credentials:
    - Go to "APIs & Services" > "Credentials."
    - Click "Create Credentials" and select "OAuth 2.0 Client IDs."
    - Choose "Web application" and provide a name for your OAuth2 client.
    - Under "Authorized redirect URIs," add the URI where your application will handle the OAuth2 callback. For example, https://yourdomain.com/oauth2/callback.
    - Click "Create."
6. Save the client ID and client secret:
    - After creation, you will receive a client ID and client secret. Save these as you will need them to configure your application.

4.0 ATLAS Supporting Two Login Modules

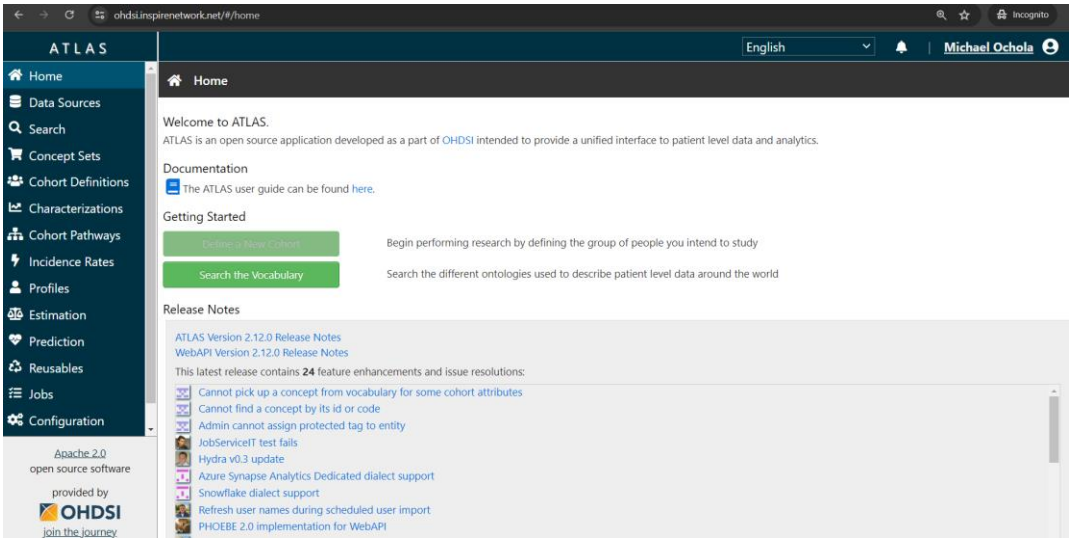


With the above configurations done and tested on AWS the login module now is fully operational with support for

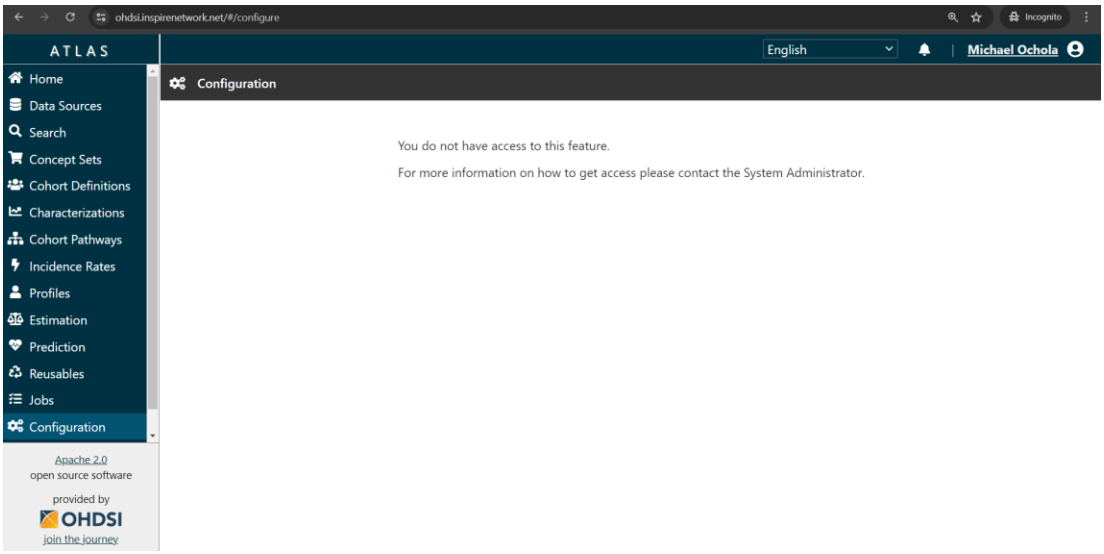
- Local security Test DB for creation of users with a special script from the backend.
- Google federated login that allows anyone with a google account to login to the ATLAS tool but with no access to any data. Therefore the access can now be granted based on needed dataset

Managing Users and Roles

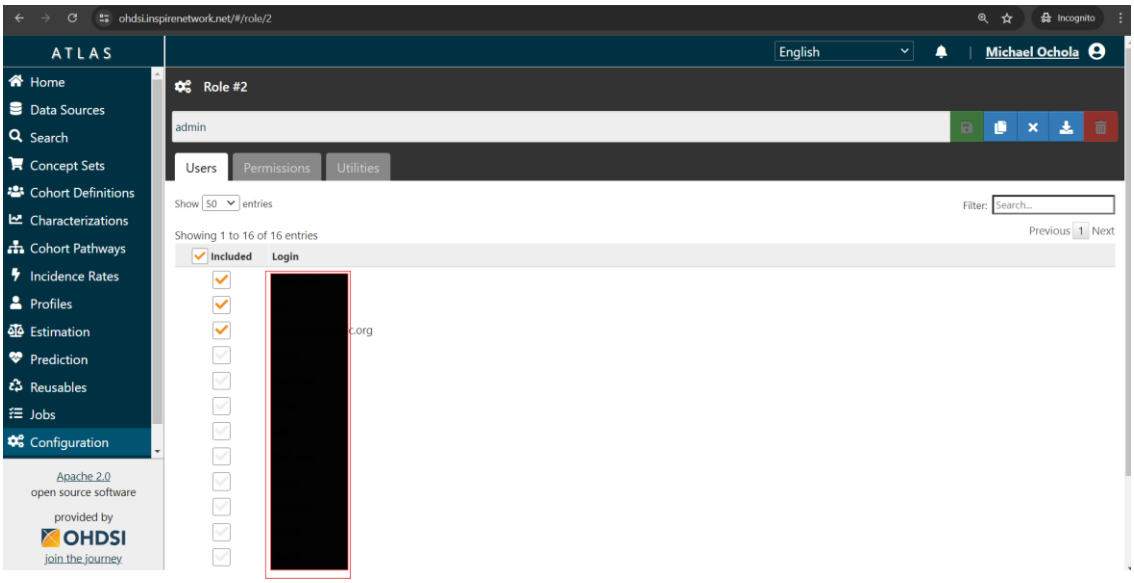
1. **Access ATLAS:**
    - Open ATLAS in your web browser (Using our test domain : https://ohdsi.inspirenetwork.net) and login using Google OAuth2.



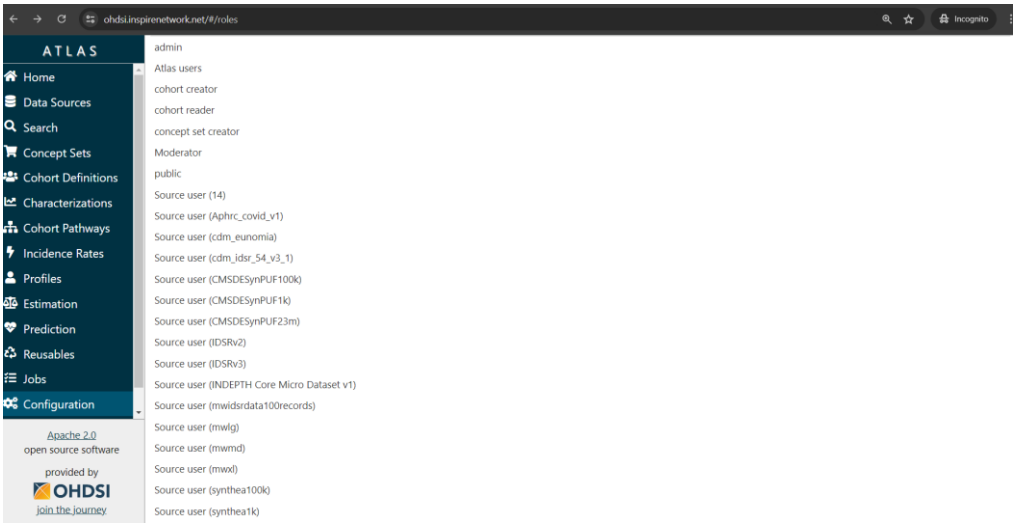
    - Depending on privileges navigate to the user management section otherwise request for additional privileges to continue.



2. **Navigate to the User Management Section:**
    - Once logged in, navigate to the Configuration menu, and select "User Management."



3. **Add and Manage Users:**
    - You can add new roles to newly signed up users by specifying either Atlas and give specific common data model access
    - Assign roles to users based on their responsibilities. The available roles typically include:
        - **Admin**: Full access to all features.
        - **User**: Access to data analysis and viewing but no administrative privileges.



4. **Assign Permissions:**
    - Assign specific permissions to each role. Permissions control access to various features like cohort definitions, pathway analysis, and reporting.