Developer Documentation-SSO System with Kong API Gateway

1. System Overview

This SSO (Single Sign-On) system provides centralized authentication using **SAML 2.0 with Google Workspace**. It uses **Kong API Gateway** for routing and security and is designed to support multiple client applications with a shared authentication state.

Architecture Components:

- **SAML Auth Service (saml-auth-service)**: A Spring Boot application that handles the SAML authentication protocol with Google Workspace, session management, and JWT generation.
- **Kong API Gateway (kong)**: The central entry point for all traffic. It manages routing, JWT validation, cookie handling, and enforcing access control on protected routes.
- **Custom Kong Plugin (`jwt-cookie-auth-redirect`)**: A custom Lua plugin that enforces authentication on protected frontend routes.
- **Client Applications (saml-auth-client, second-client-app)**: Two separate React applications that consume the SSO service.

Authentication & Routing Flows

Protected Route Access Flow

1. A user attempts to access a protected frontend route (e.g., `/dashboard`).
2. The request hits a Kong route tagged as `protected-route` .
3. The custom Kong plugin (`jwt-cookie-auth-redirect`) triggers, checking for a valid `access_token` cookie.
4. **If the cookie is invalid or missing**:
   - The plugin stores the user's original destination URL in a `return_after_auth` cookie.
   - The user is redirected to `/auth` to begin the login process.
5. **If the cookie is valid**: The request is allowed to proceed to the frontend service.

Full Login & Redirect Flow

1. A user clicks a "Login" button, which directs them to `http://localhost:8000/auth` .

2. Kong's `login-referer-handler` plugin intercepts the request, captures the `Referer` header, and saves it in the `return_after_auth` cookie.
3. The request is forwarded to the backend **SAML Auth Service**.
4. The backend service initiates the SAML flow, redirecting the user to Google for authentication.
5. After successful Google login, the user is redirected back to the backend service.
6. The backend validates the SAML response and generates a JWT `access_token`.
7. The backend redirects the user back to Kong's root URL with the token in a query parameter: `http://localhost:8000/?token=...`
8. Kong's `token-handler` plugin detects the `?token=` parameter, sets the `access_token` as a secure, `HttpOnly` cookie, and clears the temporary redirect cookie.
9. Finally, the `token-handler` reads the `return_after_auth` cookie to find the user's original destination and redirects them there.

Logout Flow

1. A user clicks a "Logout" button, which directs them to `http://localhost:8000/custom-logout`.
2. The request is forwarded to the backend **SAML Auth Service**.
3. The backend invalidates the user's `JSESSIONID` server-side.
4. The backend then redirects the user to Kong's dedicated cookie-clearing endpoint: `http://localhost:8000/kong-clear-cookies`.
5. A plugin on this Kong route receives the request, clears the `access_token` and `JSESSIONID` cookies from the browser, and performs the final redirect.

---

2. Backend Service Setup (saml-auth-service)

The backend service is a standard Spring Boot application.

2.1 Key Components

- **Configuration**:
  - application.properties: Contains all environment-specific settings for JWT, SAML, and server ports.
  - SamlSecurityConfig.java: Configures Spring Security to define public vs. protected endpoints and integrate the SAML protocol.
- **Controllers**:

    - EnterpriseAuthController.java: Handles the initial `/auth` request to start the SAML flow.
    - AuthController.java: Manages the `/custom-logout` endpoint.
    - TokenApiController.java: Provides the protected `/api/userinfo` endpoint.
- **Services**:
    - JwtService.java: Handles JWT generation after a successful SAML authentication.
    - CustomSamlAuthenticationSuccessHandler.java: Defines the logic to execute upon successful login, namely generating a JWT and redirecting back to Kong.

2.2 Environment Configuration

All settings are located in application.properties. For the full file, see the project repository.

```
# JWT Configuration (must match kong.yml)
jwt.secret=your_strong_secret_key_at_least_32_chars_long
jwt.expiration=86400000

# SAML Configuration (replace with your Google Workspace SAML App values)
saml.entity-id=https://.../saml2/service-provider-metadata/google
saml.acs-url=https://.../login/saml2/sso/google
# ... more SAML properties
```

---

3. Kong Gateway Configuration (kong.yml)

Kong's configuration is declarative. For the full file, see the project repository.

3.1 Core Plugins

- **JWT Plugin**: Attached to `/api` routes. It validates the `access_token` cookie on every API request.
- **CORS Plugin**: A global plugin that allows the frontend applications to make requests to the gateway.

3.2 Custom `pre-function` Plugins

These are small, inline Lua scripts in kong.yml that provide critical logic for the SSO flows.

- **Token Handler** (on `root-route`): Sets the `access_token` cookie when a `?token=` parameter is present in the URL.

- **Login Referer Capture** (on `auth-route`): Saves the user's original location in a cookie before redirecting to login.
- **Logout Cookie Clearer** (on `kong-clear-cookies-route`): Deletes authentication cookies from the browser.

3.3 Custom `jwt-cookie-auth-redirect` Plugin

This is the primary mechanism for protecting frontend routes. It is a custom Lua plugin that must be manually created.

- **Function**: Enforces authentication on any route it's attached to.
- **Logic**: It extracts the `access_token` cookie, validates it, and either allows access or redirects to the login flow.

---

4. Adding a New Frontend Application

Follow these steps to integrate a new application (e.g., `third-app` on port `3002`).

Step 1: Add a New Service in kong.yml

```yaml
# In services:
- name: third-app-service
  url: http://host.docker.internal:3002
```

Step 2: Add New Routes in kong.yml

Use the `protected-route` tag for any route that requires authentication.

```yaml
# In routes:
- name: third-app-root
  service: third-app-service
  paths: ["/app3", "/app3/"]
  strip_path: true

- name: third-app-dashboard
  service: third-app-service
  paths: ["/app3/dashboard"]
  strip_path: true
  tags: ["protected-route"] # This route is now protected
```

Step 3: Apply the Protection Plugin in kong.yml

The `jwt-cookie-auth-redirect` plugin is configured to automatically apply to any route with the `protected-route` tag. No extra plugin configuration is needed.

Step 4: Configure the New React App

1. In the new app's package.json, set the homepage: `"homepage": "/app3"`
2. Rebuild the app (`npm run build`) and serve it on the correct port (`3002`).

---

5. Local Development & Setup Guide

This guide provides detailed instructions for setting up the complete system locally.

5.1. Step 1: Set Up the Custom Kong Plugin

1. **Create the Plugin Directory**: In your project root, create a folder named exactly `jwt-cookie-auth-redirect` . The folder name **must** match the plugin name.
2. **Create Plugin Files**: Inside this new folder, create two files: handler.lua (the logic) and schema.lua (the definition). You can find the source code for these files in the project's GitHub repository.

5.2. Step 2: Run the System Components

A. Start the Kong Gateway

1. Run the following command from your project root to start the Kong container. This command mounts your local kong.yml file, enables the custom plugin, and exposes the necessary ports.

```
docker run -d --name kong2 `
  -v "C:/Users/Rithika/OneDrive/Desktop/sso-
kong/kong/kong.yml:/usr/local/kong/declarative/kong.yml" `
  -e "KONG_DATABASE=off" `
  -e "KONG_DECLARATIVE_CONFIG=/usr/local/kong/declarative/kong.yml" `
  -e "KONG_PLUGINS=bundled,jwt-cookie-auth-redirect" `
  -e "KONG_PROXY_ACCESS_LOG=/dev/stdout" `
  -e "KONG_ADMIN_ACCESS_LOG=/dev/stdout" `
  -e "KONG_PROXY_ERROR_LOG=/dev/stderr" `
  -e "KONG_ADMIN_ERROR_LOG=/dev/stderr" `
  -e "KONG_ADMIN_LISTEN=0.0.0.0:8001" `
  -e "KONG_LOG_LEVEL=debug" `
  -p 8000:8000 `
  -p 8001:8001 `
  kong:latest
```

2. Copy your custom plugin code into the running container. The `KONG_PLUGINS` variable tells Kong to *load* the plugin, but the files must physically exist inside the container.

```
docker cp jwt-cookie-auth-redirect
kong2:/usr/local/share/lua/5.1/kong/plugins/
```

3. Restart the container to force Kong to recognize and load the newly copied plugin files.

```
docker restart kong2
```

B. Start the Backend Service

1. Navigate to the saml-auth-service directory.
2. Run the Spring Boot application:

```
./mvnw spring-boot:run
```

C. Start the Frontend Applications

Main App (saml-auth-client)

1. Navigate to the saml-auth-client directory.
2. Run the development server:

```
npm start
```

Second App (second-client-app)

This app is served from a subpath (`/app2`), which requires a production build to work correctly.

1. **Why is a production build needed?** The development server (`npm start`) serves files from the root (`/`). When Kong tries to serve the app from `/app2` , the app's internal links to its own JavaScript and CSS files break. The production build process reads the `"homepage": "/app2"` setting in package.json and automatically fixes all these paths.

2. **Build the app**: You only need to run this command once, or whenever you change the code for this app.

   ```
   npm run build
   ```

3. **Serve the build folder**: Use a simple static server to serve the production files.

   ```
   serve -s -l 3001 build
   ```

   (If you don't have `serve`, install it globally with `npm install -g serve`).

Your entire system is now running. Access all applications through the Kong gateway at `http://localhost:8000`.