SAML Authentication System Documentation

Overview

This is a robust authentication system built using Spring Boot for the backend. It leverages SAML-based Single Sign-On (SSO) with Google Workspace as the Identity Provider (IdP) and issues JWT tokens for stateless API authentication. The system is designed to provide secure authentication and seamless integration for client applications.

Project Structure

Backend (service folder)

```
service/
│
├── src/main/java/com/example/flutto/
│   ├── config/
│   │   ├── SamlConfig.java           # SAML configuration and certificate
setup
│   │   └── SecurityConfig.java       # Spring Security configuration
│   ├── controller/
│   │   └── AuthApiController.java    # REST endpoints for client apps
│   ├── filter/
│   │   └── JwtAuthenticationFilter.java  # JWT validation filter
│   ├── service/
│   │   └── JwtService.java           # JWT generation and validation
│   └── FluttoApplication.java        # Main application class
│
├── src/main/resources/
│   ├── application.properties        # Configuration properties
│   └── google-idp.crt                # Google IdP certificate
│
├── pom.xml                           # Maven dependencies
└── .gitignore                        # Git ignore rules
```

Frontend (client folder)

```
client/
│
├── src/
│   ├── App.js                    # Main React component
│   ├── ProtectedPage.js          # Protected route component
│   ├── index.js                  # React entry point
│   ├── App.css                   # Styling for the app
│   ├── index.css                 # Global styles
│   └── reportWebVitals.js        # Performance reporting
│
├── public/
│   ├── index.html                # HTML template
│   ├── manifest.json             # Web app manifest
│   ├── favicon.ico               # Favicon
│   └── logo*.png                 # App logos
│
├── .env                          # Environment variables
├── package.json                  # Node.js dependencies
└── README.md                     # Project documentation
```

Backend Components

1. SamlConfig.java

Configures the SAML Service Provider (SP) settings for Google Workspace integration:

- **Key Features**:
  - Loads Google IdP metadata and certificate.
  - Sets up SP entity ID and ACS URL.
  - Generates self-signed certificates for SAML request signing.
  - Creates SAML metadata endpoints.

```
@Bean
public RelyingPartyRegistrationRepository relyingPartyRegistrationRepository()
{
    // Creates SAML SP configuration with Google as IdP
```

```
    // Generates self-signed certificate for signing SAML requests
    // Loads Google IdP certificate for response verification
}
```

---

2. SecurityConfig.java

Configures Spring Security for SAML and JWT authentication:

- **Key Features**:
  - Protects routes using Spring Security.
  - Configures SAML login flow (`/saml2/authenticate/google`).
  - Adds JWT authentication filter for API endpoints.
  - Defines success handler for SAML login to generate JWT tokens.

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .csrf(csrf -> csrf.disable()) // Disable CSRF for now
        .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class)
        .authorizeHttpRequests(authz -> authz
            .requestMatchers("/api/auth/custom-login",
"/api/auth/validate").permitAll()
            .anyRequest().authenticated()
        )
        .saml2Login(saml2 -> saml2.successHandler(samlSuccessHandler()))
        .logout(logout ->
logout.logoutUrl("/logout").invalidateHttpSession(true));
}
```

---

3. JwtService.java

Handles JWT token generation, validation, and parsing:

- **Key Features**:
  - Generates JWT tokens with user claims (username, email, roles).
  - Validates JWT tokens for API requests.
  - Extracts claims from tokens.

```
public String generateToken(Authentication authentication) {
    // Creates JWT with username, email, roles
}
```

---

4. JwtAuthenticationFilter.java

Intercepts API requests to validate JWT and set up security context:

- **Key Features**:
  - Extracts JWT from cookies.
  - Validates JWT using `JwtService`.
  - Sets up Spring Security context for authenticated requests.

```
@Override
protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) {
    // Extracts JWT from cookies and validates it
    // Sets up security context if valid
}
```

---

5. AuthApiController.java

Provides REST API endpoints for client applications:

- **Endpoints**:
  - `/api/auth/custom-login`: Stores redirect URI and initiates SAML login.
  - `/api/auth/validate`: Validates JWT and returns user info.
  - `/api/auth/custom-logout`: Handles logout and redirects to client app.

```
@GetMapping("/custom-login")
public void customLogin(@RequestParam String redirectUri, HttpServletRequest
request, HttpServletResponse response) {
    // Stores redirect URI in session and redirects to SAML login
}
```

Frontend Components

1. App.js

The main React component that handles routing and authentication:

- **Key Features**:
  - Implements login flow using `/api/auth/custom-login`.
  - Validates JWT via `/api/auth/validate`.
  - Protects routes using `ProtectedRoute` logic.

```
const ProtectedRoute = ({ children }) => {
    if (!user) return <Navigate to="/" replace />;
    return children;
};
```

---

2. ProtectedPage.js

A sample protected route that displays content for authenticated users:

```
export default function ProtectedPage() {
    return (
        <div>
            <h1>🔒 Protected Page</h1>
            <p>Welcome to the protected page! You are authenticated.</p>
        </div>
    );
}
```

---

3. Login Flow

The frontend initiates the login flow by calling `/api/auth/custom-login`:

```
const handleLogin = () => {
    window.location.href = `${API_BASE}/api/auth/custom-login?
redirectUri=${encodeURIComponent(window.location.origin)}`;
};
```

4. JWT Validation

The frontend validates the JWT token after login:

```
useEffect(() => {
    if (jwt) {
        fetch(`${API_BASE}/api/auth/validate?token=${jwt}`)
            .then(res => res.json())
            .then(data => {
                if (data.valid) setUser(data);
                else {
                    setUser(null);
                    setJwt(null);
                    localStorage.removeItem("jwt");
                }
            });
    }
}, [jwt]);
```

Authentication Flow

1. Login Flow

1. **Frontend**:
   - Calls `/api/auth/custom-login` with the redirect URI.
   - Redirects the user to `/saml2/authenticate/google`.
2. **Backend**:
   - Stores the redirect URI in the session.
   - Initiates the SAML login flow.
3. **Google Workspace**:
   - Authenticates the user.
   - Sends the SAML response to the backend.
4. **Backend**:
   - Processes the SAML response.
   - Generates a JWT token.
   - Redirects the user to the stored redirect URI with the JWT.
5. **Frontend**:

- Extracts the JWT from the URL.
- Validates the JWT via `/api/auth/validate`.
- Displays authenticated content.

---

2. Protected Route Flow

1. **Frontend**:
   - Navigates to `/protected`.
   - Checks if the user is authenticated using `ProtectedRoute`.
2. **Backend**:
   - Validates the JWT for API requests.
   - Returns the requested data if authenticated.

Security Considerations

1. **JWT Secret**:
   - Use a strong, unique secret key for signing JWTs.
2. **HTTPS**:
   - Always use HTTPS for all endpoints in production.
3. **Token Storage**:
   - Store JWT tokens securely (prefer HTTP-only cookies).
4. **CORS**:
   - Configure CORS to allow only trusted origins.
5. **Session Management**:
   - Ensure sessions are invalidated on logout.

Example API Calls

1. Custom Login

```
GET /api/auth/custom-login?redirectUri=https://your-client-app.com
```

2. Validate JWT

```
GET /api/auth/validate?token=<JWT_TOKEN>
```

3. Custom Logout

```
GET /api/auth/custom-logout?redirectUri=https://your-client-app.com
```

Steps to perform to set it up in your app:

- Configure the acs url and entity url in application.properties and in admin console(SP DETAILS)
- Ensure the Idp metadata is correct
    - Get the Idp metadata form the admin console.(check the google sso url and google entity id in application.properties and google-idp-certificate in src/main/resources)