Low‑Level Technical Specification: Authentication & Authorization

## **1. Overview**

Provide secure, standards‑based authentication (AuthN) and fine‑grained authorization (AuthZ) using Keycloak (OpenID Connect/OAuth2). Integrate with backend APIs and frontend apps via JWT tokens.

## **2. Identity Provider: Keycloak Setup**

* **Version**: Keycloak 21.x (quarkus distribution, Apache‑2.0)
* **Realm**: scrum-poker-realm
* **Clients**:  
  + **web-client**
    - Client ID: scrum-poker-web
    - Access Type: public
    - Redirect URIs: https://app.example.com/\* (matching production/dev)
    - Valid Redirect URIs: http://localhost:5173/\*
    - Web Origins: \*
    - Standard Flow Enabled: true (authorization code)
    - Implicit Flow Disabled
    - Service Accounts Disabled
  + **mobile-client**
    - Client ID: scrum-poker-mobile
    - Access Type: public
    - Redirect URIs: exp://\* (Expo deep links)
    - Standard Flow Enabled: true
* **Realm Roles**:  
  + Developer
  + ScrumMaster
  + ProductOwner
  + Observer
* **User Federation / LDAP**: Optional; use Keycloak internal user store.
* **Password Policy**: length(8) and digits(1) and upperCase(1)
* **OTP**: Disabled for MFA (optional later)
* **Token Settings** (Realm Settings → Tokens):  
  + Access Token Lifespan: 1h
  + Refresh Token Lifespan: 7d
  + Client Session Idle: 30m
  + Tokens Signed With: RS256

## **3. OAuth2 / OIDC Token Flow**

1. **Login Redirect**: Frontend redirects unauthenticated user to Keycloak /protocol/openid-connect/auth with params: client\_id, redirect\_uri, scope=openid profile email, response\_type=code, code\_challenge (PKCE), state.
2. **Consent & Authentication**: Keycloak renders login page, authenticates user, returns authorization\_code to redirect\_uri.
3. **Token Exchange**: Frontend calls /protocol/openid-connect/token with grant\_type=authorization\_code, code, redirect\_uri, code\_verifier. Keycloak returns:  
   * access\_token (JWT)
   * refresh\_token
   * id\_token (JWT with user claims)
4. **Session Establishment**: Frontend stores tokens in memory (or secure storage for mobile).
5. **API Calls**: Frontend includes Authorization: Bearer <access\_token> header on each API/WebSocket call.
6. **Token Refresh**: On 401 or near expiration, frontend calls /protocol/openid-connect/token with grant\_type=refresh\_token, refresh\_token. Updates tokens.

## **4. JWT Configuration & Claims**

* **Signing Algorithm**: RS256 with realm’s RSA key pair.
* **Claims**:  
  + iss → https://auth.example.com/realms/scrum-poker-realm
  + sub → user UUID
  + aud → client IDs (scrum-poker-web, scrum-poker-mobile)
  + exp, iat, auth\_time
  + azp → authorized party (client)
  + realm\_access.roles → array of assigned realm roles
  + resource\_access.<client>.roles → client‑specific roles (if any)
* **Token Size**: keep minimal to avoid large headers (include only necessary claims).

## **5. Backend Middleware (Express.js)**

* **Library**: express-jwt v8.x (MIT) or keycloak-connect (Apache‑2.0)

**Initialization**:  
  
 import jwt from 'express-jwt';

import jwksRsa from 'jwks-rsa';

const checkJwt = jwt({

secret: jwksRsa.expressJwtSecret({

cache: true,

rateLimit: true,

jwksRequestsPerMinute: 10,

jwksUri: `${KEYCLOAK\_URL}/realms/scrum-poker-realm/protocol/openid-connect/certs`

}),

audience: 'scrum-poker-web',

issuer: `${KEYCLOAK\_URL}/realms/scrum-poker-realm`,

algorithms: ['RS256']

});

**Role Middleware**:  
  
 function requireRole(role) {

return (req, res, next) => {

const roles = req.user.realm\_access?.roles || [];

if (roles.includes(role)) next(); else res.sendStatus(403);

};

}

**Route Protection**:  
  
 app.post('/api/sessions', checkJwt, requireRole('ScrumMaster'), createSession);

app.post('/api/sessions/:id/vote', checkJwt, requireRole('Developer'), submitVote);

**WebSocket Auth**:  
  
 io.use((socket, next) => {

const token = socket.handshake.auth.token;

jwt.verify(token, publicKey, options, (err, decoded) => {

if (err) next(new Error('Authentication error')); else {

socket.user = decoded;

next();

}

});

});

## **6. Frontend Integration**

* **Library**: oidc-client-ts v2.x (MIT) for web; react-native-app-auth for mobile

**Configuration**:  
  
 const oidcConfig = {

authority: 'https://auth.example.com/realms/scrum-poker-realm',

client\_id: 'scrum-poker-web',

redirect\_uri: 'https://app.example.com/callback',

response\_type: 'code',

scope: 'openid profile email',

post\_logout\_redirect\_uri: 'https://app.example.com/',

automaticSilentRenew: true,

filterProtocolClaims: true,

loadUserInfo: true

};

* **Storage**: Use in-memory store or SecureStore (Expo) for refresh tokens; avoid localStorage for refresh token to reduce XSS risk.
* **Auth Context**: React Context providing user, login(), logout(), getToken(), isAuthenticated.

## **7. Roles & Permissions Matrix**

| **Action** | **Developer** | **ScrumMaster** | **ProductOwner** | **Observer** |
| --- | --- | --- | --- | --- |
| Create Session |  | ✅ |  |  |
| Add Story | ✅ | ✅ | ✅ |  |
| Submit Vote (real‑time) | ✅ | ✅ | ✅ |  |
| Reveal Votes |  | ✅ |  |  |
| Terminate Session |  | ✅ |  |  |
| View Session (read only) | ✅ | ✅ | ✅ | ✅ |
| Access AI Reports | ✅ | ✅ | ✅ |  |

## **8. Security Considerations**

* **CSRF Protection**: Not required for pure API (JWT in header), but protect OIDC callback endpoint against forged requests.
* **CORS**: Restrict to known origins (app.example.com, localhost:5173).
* **Secure Cookies**: If using cookies for tokens, set HttpOnly, Secure, SameSite=Strict.
* **PKCE**: Enforce Proof Key for Code Exchange on public clients.
* **Replay Protection**: Validate nonce in ID token; reject reused code.
* **Audit**: Log authentication events (login, logout, token\_refresh) to audit\_logs table.

## **9. Testing & Validation**

* **Integration Tests**:  
  + Mock Keycloak JWKS endpoint via nock.
  + Validate protected routes return 401 without token, 403 without role.
* **E2E Tests**:  
  + Simulate login via OIDC flow using Cypress plugin (cypress-oauth), assert session persists.
* **Security Scans**:  
  + OWASP ZAP targeting OIDC endpoints.
  + Static analysis (e.g., npm audit, snyk).

*End of Authentication & Authorization Specification*