Authentication and Security

Authentication Flow

The ACT app uses **Authentik** as its identity provider (IdP) via **OpenID Connect (OIDC)**. Below is a step-by-step overview of the flow:

1. **User Clicks Authenticate Button**
   The user initiates authentication by clicking the “Authenticate with Authentik” button, which redirects to Authentik’s OIDC authorization endpoint.
2. **OIDC Authorization Request**
   The app uses the configured OIDC_AUTHORIZATION_ENDPOINT, passing the following parameters:
   a. client_id – the registered Authentik client ID
   b. redirect_uri – e.g., http://localhost:3000/auth/callback
   c. response_type=code – for authorization code flow
   d. scope=openid profile email
3. **Authentik Login Page**
   The user is taken to Authentik's default authentication flow. Upon successful login, Authentik issues an authorization code.
4. **Token Exchange**
   The app exchanges the code for tokens by POSTing to the OIDC_TOKEN_ENDPOINT. The response includes:
   a. id_token
   b. access_token
   c. refresh_token (optional)
5. **User Session Created**
   The tokens are stored in an HTTP-only cookie on the frontend. The session is now authenticated and the user can interact with the app.
6. **Silent Authorization**
   For future requests, the token stored in the cookie is sent with every call, maintaining the user’s authenticated state.

Authorization Rules

Role-based access control (RBAC) is configured inside Authentik. At present, the ACT app uses a **single client with global access**, meaning:

- All authenticated users have the same access level within the application.
- More granular roles (e.g., Admin, Viewer) can be configured in Authentik and enforced using scopes or claim-based logic.

Future development could enhance authorization logic by mapping OIDC claims to UI/feature-level permissions.

Secrets & Key Management

Secrets used in the application are stored in the local .env file, which is passed into the Docker containers at runtime.

Key items stored in .env include:

- OIDC_CLIENT_SECRET – for exchanging authorization codes
- OIDC_COOKIE_SECRET – used to sign secure session cookies
- OPENAI_API_KEY – used to call the GPT model
- POSTGRES_PASSWORD, COUCHDB_PASSWORD – database access

Current strategy:

- Environment variables are **not committed to version control**
- .env.example is included for structure, but real secrets must be injected locally

Future recommendations:

- Use Docker secrets or a vaulting solution like **HashiCorp Vault**, **AWS Secrets Manager**, or **Docker Swarm Secrets**
- Implement secret rotation policies and access control

🛡️ Security Considerations

Concern	Current Handling	Recommendation
Token Storage	Stored in HTTP-only cookie	✓ Secure and recommended
HTTPS (TLS)	Not enabled by default in local setup	Enable TLS via reverse proxy (e.g., Traefik/Nginx)
Role-based Access Control	Not fully enforced in frontend/backend	Integrate scopes or claims in Authentik
CSRF/XSS Protection	Cookies are HTTP-only	Ensure input sanitation and consider CSRF tokens
Session Expiration & Refresh	Depends on Authentik configuration	Configure token expiration and idle timeout