**F. Software Security Concerns**

**F1. Authentication (2 pts)**

* **Password handling:** Registration hashes passwords with bcryptjs (configurable cost factor) and stores only the hash, role, salts, and metadata. Salts are reused to derive vault encryption keys, ensuring no plaintext secrets are persisted.
* **Session strategy:** Successful login issues an HTTP-only, same-site cookie backed by express-session plus a short-lived JWT containing the user id and role. Tokens include issuer/audience checks and expire after one hour; refreshes occur only after reauthentication.
* **Logout and account deletion:** The logout endpoint destroys the session, clears cookies, and invalidates the JWT immediately. Right-to-be-forgotten deletion removes the user record and encrypted vault file from disk, ensuring no stale credentials remain.
* **Error hygiene:** Authentication endpoints return generic failure messages to avoid leaking whether an email exists, reducing account enumeration risk.

**F2. Role Management (1 pt)**

* **Middleware enforcement:** A dedicated authorizeRoles helper ensures admin-only routes (such as progress summaries) are accessible solely to privileged accounts. Routes early-return with 403 Forbidden when a JWT lacks the required role claim.
* **Principle of least privilege:** Admin dashboards expose only aggregated counts and milestone statistics, not individual vault data, so elevated users cannot accidentally view private journal entries.

**F3. Threats & Countermeasures (1 pt)**

**Data injection vulnerabilities**

* **Threat:** Unvalidated request bodies could inject malicious payloads that pollute user profiles, coerce prompt templates, or cause persistent cross-site scripting once echoed in the UI.
* **Countermeasures:** Server endpoints normalize and validate numeric inputs, cap text payload sizes (chat and Socket.IO), and store only the minimal data required. Password hashes are generated with bcryptjs to resist offline cracking attempts.
* **Next steps:** Introduce schema validation with libraries like Zod or Joi and escape user-supplied fields on render paths to harden against residual injection risks.

**Session hijacking prevention**

* **Threat:** Stolen cookies or replayed JWTs could let attackers impersonate users or steal progress data.
* **Countermeasures:** Sessions use HTTP-only, same-site cookies with short lifetimes, and JWTs embed role claims with issuer/audience validation. Logout explicitly revokes tokens and clears session data. Admin-only APIs enforce role checks.
* **Next steps:** Enable secure cookies in production, rotate secrets regularly, and consider refresh-token rotation or device fingerprinting as usage grows.

**API rate limiting needs**

* **Threat:** Unbounded chat calls can lead to denial-of-service or brute-force probing.
* **Countermeasures:** /api/chat is protected by express-rate-limit, throttling bursts with helpful error feedback. Socket messages mirror the same length limits to block spam payloads. Additional strategic rate limits can be layered per route if abuse is detected.

**Personal data exposure risks**

* **Threat:** Email addresses, hashed credentials, and progress logs represent personal data subject to regulation.
* **Countermeasures:** User store persists only hashes, role, and optional display name. Aggregated admin views expose metrics without identifiers. Sanitisation removes sensitive fields from API responses, and encrypted vault storage keeps personal content confidential.
* **Next steps:** Define a formal data-retention policy, add subject erasure workflows to operational runbooks, and enforce disk encryption or managed secrets for production deployments.

**Implemented safeguards summary**

* Input validation across assessment, progress, and chat endpoints.
* Rate limiting and payload caps to deter abuse.
* Secure session management with logout revocation.
* HTTPS deployment guidance, including HSTS once TLS is enforced.