QA Plan

1 Objectives

- Ensure code quality, functional correctness, and reliability of core features, including expense aggregation, spending visualization, and goal setting/tracking.
- Detect defects early via automated checks and disciplined reviews.
- Define clear acceptance/exit criteria for the sprint and final release.

2 Scope

- Core modules: Authentication & Account Recovery (security questions), Expense CRUD, Category logic, Goal set/track, Alerts, Dashboard.
- API/DB integration (PostgreSQL), data validation, and permissions.
- Cross-browser sanity (latest Chrome/Edge/Firefox).

3 Quality Approach (Code Quality)

- **Standards:** ESLint + Prettier; commit message convention (e.g., Conventional Commits).
- **Branching:** main (protected) / develop / feature branches (feature/<name>).
- **Code Review:** ≥1 reviewer; review checklist (logic correctness, null/edge handling, accessibility, tests updated, performance).
- **Static Analysis:** ESLint (errors block merge), TypeScript strict mode (if used).
- **Pre-commit Hooks:** Husky + lint-staged for lint/format/test on changed files.
- **CI Gates (GitHub Actions):** install → lint → build → unit tests → coverage check → (optional) integration tests on PR.

4 Test Levels & Methods

- **Unit Tests** (Dev-owned): pure functions, reducers, validators, services.
  Tools: Jest + (React Testing Library for components).
- **Integration Tests** (Dev/QA): API ↔ DB (via test DB), UI component ↔ service contracts.
- **End-to-End (E2E)** (QA-owned): critical flows (login, record expense, set goal, alert trigger).
- **Non-functional (lightweight):**
  - **Performance:** page load under X seconds (local baseline), API typical responses < 300ms for simple reads (mock/stub acceptable).
  - **Security (basic):** no secrets in repo, OWASP checks for input validation, rate limit for recovery endpoint.
  - **Accessibility:** basic ARIA roles, keyboard navigation for primary flows.

5 Test Data & Environments

- **Environments:** dev (local), test (CI), staging (optional), same schema as prod.

- **DB:** isolated test database with seed scripts; migrations version-controlled (e.g., Drizzle).
- **Data Management:** deterministic seeds for repeatability; anonymized sample data only.

6 Defect Management

- **Severity:**
  S1—Blocker (feature unusable); S2—Major (no workaround); S3—Minor (workaround exists); S4—Trivial (UI copy).
- **Workflow:** New → In Progress → Fixed → In Review → Verified → Closed.
- **SLAs (coursework-practical):** S1 fix within 24h; S2 within 48h; S3/S4 within sprint.

7 Test Coverage Targets

- **Unit:** ≥80% lines/branches on core logic (validators, calculators, reducers).
- **Integration:** key API routes and DB paths covered (create/read/update expense; goal set/track).
- **E2E:** happy paths + at least one edge path per epic (e.g., over-budget alert).

8 Acceptance / Exit Criteria

- **Per User Story (Definition of Done)**
  - Code reviewed and merged with passing CI.
  - Unit tests for new/changed logic with coverage maintained.
  - Acceptance criteria verified (manual or automated).
  - No open S1/S2 defects; documentation/README updated.
- **Sprint Exit**
  - All committed stories meet DoD.
  - Regression suite green; no S1/S2 open; ≤3 S3/S4 with agreed deferral.
- **Release Exit**
  - Critical flows pass E2E:
  - Login → Dashboard
  - Add expense → visible in list → totals update
  - Set goal → progress shown
  - Cross 80% → warning; cross 100% → over-budget alert
  - Password reset via security questions
  - Backup/restore tested on test DB; deployment checklist completed.