**Test Plan • FastAPI Full Stack Template**

**1. Project picked and why**

I chose the FastAPI Full Stack Template because it provides a complete but lean stack with a React TypeScript frontend, a FastAPI backend, SQLModel with PostgreSQL, authentication, and built-in testing scaffolds. It runs locally with Docker Compose and has clear docs and recent releases. This lets me focus on test design and automation rather than heavy setup. (GitHub, FastAPI)

**2. Scope and objectives**

- Validate core user journeys end to end through the UI
- Confirm auth and session behavior
- Verify CRUD flows on at least one entity in the sample app
- Exercise client to server integration and API error surfacing
- Produce a small but maintainable TypeScript automation suite

**Out of scope**

- Performance and load
- Security pen tests
- Multi locale and visual regression beyond basic checks

**3. Test approach**

**Manual**

- Short exploratory passes to map flows, states, and edge cases
- Targeted checks for input validation and navigation

**Automated**

- Tooling: Playwright with TypeScript
- Strategy: page object light helpers, data builders, fixtures for auth
- Coverage: three critical flows with positive and negative paths
- CI ready: headless run and concise HTML report

**4. Environment**

- Local Docker Compose as provided by the template
- Frontend on localhost, backend on localhost with interactive docs at docs and redoc, database via Adminer, email testing via MailCatcher, reverse proxy via Traefik

- Test user seeded from environment variables for first superuser Reference. (GitHub)

**Assumptions**

- Fresh containers and volumes before runs when needed
- Stable ports and default seed data available

**5. Test data**

- One admin user created from env values
- One regular user created via UI during tests
- Synthetic records for CRUD flows created and cleaned per test

**6. Entry and exit criteria**

**Entry**

- App builds and all services report healthy
- Test user credentials available

**Exit**

- Planned scenarios executed
- Zero open critical or high blockers in covered areas
- Automation green in headless mode

**7. Risks and mitigations**

- Flaky selectors due to dynamic UI. Mitigation: role based and text based locators and explicit waits
- Data coupling across tests. Mitigation: isolated users and id based cleanup
- Auth state leakage between tests. Mitigation: per test context and storage state reset

**8. Prioritization**

**High**

- Sign up, login, logout
- Protected route access control
- CRUD happy path on a primary entity

**Medium**

- Validation errors and API error surfacing

- Session persistence and expiry

**Low**

- Non critical UI details and layout

**9. Test cases for critical flows**

**TC-01Login with valid credentials**

Precondition: first superuser exists

Steps: open login, enter valid email and password, submit

Expected: redirected to dashboard and user name visible

Priority: High

**TC-02 Login with invalid password**

Steps: valid email, wrong password, submit

Expected: clear error message, no session created

Priority: High

**TC-03 Sign up new user**

Steps: open sign up, enter unique email and strong password, submit, then login

Expected: account created and login succeeds

Priority: High

**TC-04 Access protected page without auth**

Steps: open a protected route in a fresh context

Expected: redirected to login or see access denied

Priority: High

**TC-05 Logout clears session**

Steps: login, trigger logout, navigate back to protected route

Expected: redirected to login

Priority: High

**TC-06 Create entity happy path**

Steps: login, open create form, fill valid fields, submit

Expected: success toast, record appears in list and detail

Priority: High

**TC-07 Create entity with invalid input**

Steps: submit empty or invalid fields

Expected: field level errors and no record created

Priority: Medium

**TC-08 Update entity**

Steps: open existing record, edit a field, save

Expected: success toast and persisted change in list and detail

Priority: Medium

**TC-09 Delete entity with confirm**

Steps: delete existing record and confirm

Expected: record removed from list, 404 on old detail link

Priority: Medium

**TC-10 API error surfaced to UI**

Steps: simulate backend failure for create or update if possible or use invalid payload

Expected: friendly error state and no crash

Priority: Medium

**TC-11 Navigation and deep links**

Steps: use in app links and direct URL entries for main areas

Expected: consistent routing and expected guards

Priority: Low

**TC-12 Password recovery path**

Steps: request reset for existing email, confirm email captured in MailCatcher, complete flow if enabled

Expected: reset email visible in MailCatcher, password change effective

Priority: Low

**10. Automation plan**

Framework and language

- Playwright with TypeScript as required by the assessment

Suite outline

- tests auth login positive and negative
- tests entities create update delete
- tests access control and direct route guard

Design

- Reusable selectors and helpers under utils
- Storage state fixture for authenticated runs where appropriate
- Test data builders to keep inputs consistent

Reporting

- Standard Playwright HTML report plus junit xml for CI

**11. Defect reporting**

Fields

- Title and unique id
- Environment and build
- Steps to reproduce
- Expected result and actual result
- Severity and priority with justification
- Screenshots or video and console or network logs
- Suggested area to investigate or probable cause

Severity model

- Critical: stops core flow or data loss

- High: blocks primary path without workaround
- Medium: affects secondary path or confusing behavior
- Low: minor issue or cosmetic

**12. Deliverables**

- Test plan document
- Separate repository with Playwright TypeScript tests and README that explains setup, how to run tests, and assumptions
- Three detailed bug reports with reproduction steps and evidence as requested in the assessment.