Automaspec — Front-End Architecture & Development Report

Date: 2025-12-18

0) Project Summary

Automaspec is a web application for managing test specifications, requirements, and tests with a tree-based navigation UI, status tracking, and Vitest report synchronization. The product includes authentication, organizations (workspaces), and a dashboard UI.

Front-end stack (based on `README.md` and `package.json`):

- Next.js (App Router) + React 19 + TypeScript
- Tailwind CSS v4 + UI components (shadcn-style in `components/ui/*`)
- TanStack Query (global server-state) + TanStack React Form (forms)
- oRPC (type-safe contracts) + OpenAPI handler (single entry point `/rpc`)
- Vitest + React Testing Library
- Oxlint + Oxfmt + Lefthook (engineering culture)

1) SPA Requirement and Routing

The project uses the Next.js App Router (`app/*`) and provides an SPA-like user experience:

- client-side navigation via `next/link` / `next/navigation` without full page reloads
- key interactive areas are client components (`'use client'`) with local state and async API calls

Key routes (3-4+ routes requirement is met):

- `/` — landing (`app/page.tsx`)
- `/login` — authentication (`app/login/page.tsx`)
- `/dashboard` — main application (tree + details) (`app/dashboard/page.tsx`)
- `/profile` — profile/export/delete account (`app/profile/page.tsx`)
- `/ai` — AI assistant page (`app/ai/page.tsx`)
- `/create-organization`, `/choose-organization`, `/invitations` — organization flows (`app/(organizations)/*`)

2) Architecture Diagram (Modules, Components, Routes, State)

2.1 Logical diagram

```
[UI: app/* + components/*]
  |
  |  (TanStack Query / TanStack Form)
  v
[API client: lib/orpc/orpc.ts]
  |
  |  fetch + cookies, OpenAPI link
  v
[/rpc handler: app/(backend)/rpc/[...all]/route.ts]
  |
  |  oRPC router + middleware (auth/org)
  v
[Server routes: orpc/routes/*]
  |
  v
[DB: drizzle + libsql]   (not FE, but defines contracts)
```

2.2 Physical project structure (front-end relevant)

```
app/
  layout.tsx              - root layout + Providers
  providers.tsx           - Theme + TanStack Query + Toaster
  page.tsx                - landing
  login/page.tsx          - forms + validation
  dashboard/*             - feature (page + components + hooks)
  profile/page.tsx        - profile actions
  (organizations)/*       - org flows
```

```
components/
  ui/*                    - reusable UI elements
  theme-provider.tsx      - theming
  theme-toggler.tsx       - theme switcher
lib/
  orpc/*                  - oRPC context/client
  query/*                 - QueryClient (+ optional hydration)
  shared/*                - auth client, form helpers
  constants.ts            - centralized constants
  types.ts                - Zod schemas + TS types
__tests__/
  components/*            - unit/component tests (RTL)
  integration/*           - integration flows
```

3) Architectural Approach

3.1 Component-based architecture

- Pages/containers: `app/**/page.tsx` assemble features and connect UI to data.
- UI components: `components/ui/*` provide primitives (buttons, dialogs, tabs, skeleton, toast, etc.).
- Feature components: `app/dashboard/components/*` implement dashboard-specific functionality.

3.2 State management

- Local state: `useState`, `useEffect` for selection, dialogs, forms, UI toggles, AI progress, etc.
- Global/cross-cutting state:
  - server-state: TanStack Query (`@tanstack/react-query`) provided via `app/providers.tsx`
  - session/organization: `authClient` (Better Auth hooks): `useSession`, `useActiveOrganization`, `useListOrganizations`
- Theming: `next-themes` via `ThemeProvider` (`app/providers.tsx`)

3.3 Services / hooks

- Centralized API layer: `lib/orpc/orpc.ts`
  - `safeClient` for calls returning `{ data, error }`
  - `orpc` TanStack Query utilities (`queryOptions`, `key(...)`)
- Server context for middleware: `lib/orpc/context.ts`
- Query client configuration + serialization: `lib/query/client.ts`, `lib/query/serializer.ts`

4) API Documentation (Contracts, Methods, Examples)

4.1 How the client calls the API

Single API entry point: `/rpc` (Next.js route handler)

- `app/(backend)/rpc/[...all]/route.ts` runs the OpenAPI handler with logging and CORS.
- The client uses `fetch(..., { credentials: 'include' })` so session cookies are included automatically (`lib/orpc/orpc.ts`).

OpenAPI endpoints:

- `/rpc/spec` — OpenAPI spec
- `/rpc/docs` — reference UI (via `OpenAPIReferencePlugin`)

4.2 Authentication and organization

- oRPC middleware:
  - `orpc/middleware.ts`: `authMiddleware` (requires `session`) and `organizationMiddleware` (requires `activeOrganizationId`)
- If there is no session, the `/rpc` handler redirects to `/login`.

4.3 Core API methods (tests domain)

Contract file: `orpc/contracts/tests.ts`

- Folders:
  - `GET /test-folders/{id}`

  - `GET /test-folders` (optional `parentFolderId`)
  - `GET /test-folders/{folderId}/children?depth=0..5`
  - `GET /test-folders/find?name=...`
  - `POST /test-folders/{id}` (upsert)
  - `DELETE /test-folders/{id}`
- Specs:
  - `GET /test-specs/{id}`
  - `GET /test-specs` (optional `folderId`)
  - `PUT /test-specs/{id}` (upsert)
  - `DELETE /test-specs/{id}`
- Requirements:
  - `GET /test-requirements` (optional `specId`)
  - `PUT /test-requirements/{id}` (upsert)
  - `DELETE /test-requirements/{id}`
- Tests:
  - `GET /tests` (optional `requirementId`)
  - `PUT /tests/{id}` (upsert)
  - `DELETE /tests/{id}`
  - `POST /tests/sync-report` — sync Vitest report into DB state
  - `GET /tests/report` — read Vitest report

Types and schemas:

- Zod schemas: `lib/types.ts` (`test*SelectSchema`, `test*InsertSchema`, `vitestReportSchema`, etc.)
- Status constants: `lib/constants.ts` (`TEST_STATUSES`, `SPEC_STATUSES`, `STATUS_CONFIGS`)

Example: fetch folder children (tree)

```
GET /rpc/test-folders/root/children?depth=2
```

Response example (shortened):

```
[
  {
    "id": "folder-1",
    "name": "Auth",
    "type": "folder",
    "children": [{ "id": "spec-1", "name": "Login", "type": "spec" }]
  },
  { "id": "spec-2", "name": "Smoke", "type": "spec" }
]
```

5) User Flows / Navigation

5.1 Main flows

1) Sign in:

- `/` → `/login`
- `Sign in` (email/password validation) → `/dashboard`

2) Organization onboarding:

- if there is no active organization:
  - organizations exist → `/choose-organization` → set active → `/dashboard`
  - no organizations → `/create-organization` → create + set active → `/dashboard`

3) Dashboard work:

- `/dashboard`
  - folder/spec tree navigation
  - create folders/specs, delete
  - view requirements and tests for the selected spec
  - report sync via `Sync Tests` button

4) Profile:

- `/profile` → export data / sign out / delete account

6) UI and Reusability

- Layout: `app/layout.tsx` + `app/providers.tsx`
- Reusable UI: `components/ui/*` (button, dialog, dropdown, tabs, skeleton, toast, etc.)
- Dashboard feature UI: `app/dashboard/components/*`

7) Forms and Validation

- `/login`: TanStack React Form + Zod schemas (`SignInSchema`, `SignUpSchema`), user-facing validation messages, submitting states.
- `/create-organization`: TanStack React Form + sync/async validators (including slug check via `authClient.organization.checkSlug`).

8) Async Operations, Error Handling, Loading States

8.1 Error handling

- UI layer: toast notifications via `sonner` (`toast.success`, `toast.error`) in core flows (login, org, dashboard, profile).
- API layer: oRPC handler `onError(...)` logs errors and validation issues (`app/(backend)/rpc/[...all]/route.ts`).
- Auth handling: no session → redirect to `/login` at `/rpc`.

8.2 Loading states

- Shared loader component: `components/loader.tsx`
- Per-flow states: `isPending` (session/org hooks), `isLoading` (AI), `isSyncing` (sync report)
- Tree UI shows per-item loading indicators.

9) Testing

Tooling:

- Vitest (`pnpm test`)
- React Testing Library (jsdom, setup in `__tests__/setup.ts`)

Coverage in repo:

- Unit/component tests: `__tests__/components/*`, `__tests__/lib/*`, `__tests__/db/*`, `__tests__/orpc/*`
- Integration tests: `__tests__/integration/*` (README notes they require a reachable `NEXT_PUBLIC_DATABASE_URL`)

10) Stack Rationale

- Next.js + React: strong DX, routing, code splitting, mature ecosystem.
- TypeScript + Zod: strong typing and boundary validation.
- oRPC + OpenAPI: contract-first approach with shared types and OpenAPI spec generation; integrates well with TanStack Query.
- TanStack Query: caching, invalidation, refetching, predictable server-state management.
- Tailwind + shadcn-style UI: fast development and consistent design system patterns.
- Vitest + RTL: fast component tests and integration-style flows.

11) Checklist Self-Assessment (What is covered / what to improve)

Already covered:

- SPA-like navigation and routing (many routes).
- Separation into pages/feature components/UI components/services.
- Strong typing (TypeScript + Zod).
- Centralized API layer (`lib/orpc/orpc.ts`).
- Error handling + user-friendly messages (toasts).
- Forms + validation.
- Data lists (tree UI, requirements/tests lists).
- Unit + integration tests.
- Linting/formatting/hooks (Oxlint/Oxfmt/Lefthook).

Potential gaps for strict checklists:

- A dedicated user-facing “search/filter/sort” control in the dashboard is not clearly present (there is status grouping and list rendering, but not an explicit search/filter UI).
- No explicit `app/error.tsx`, `app/not-found.tsx`, `app/*/loading.tsx` pages (if the reviewer expects Next.js error boundaries / 404 page / route-level loading UI).

12) Build PDF on Windows (pandoc + xelatex)

Run from repository root:

```
cd docs
pandoc --pdf-engine=xelatex -V lang=en-US -V mainfont="Times New Roman" -V monofont="Consolas" --variable=geometry:ma
```