Automaspec — Front-End Architecture & Development Report

Project: Automaspec
Version: 0.1.0
Date: 2026-01-03

1. Summary

Automaspec is a web application for managing test specifications, requirements, and tests in a foldered hierarchy. The UI supports authentication and organizations (workspaces), a dashboard for managing specs, an analytics page, and an AI assistant page.

2. Technology Stack

- Next.js (App Router) + React 19 + TypeScript
- Tailwind CSS v4 + reusable UI primitives in `components/ui/*`
- TanStack Query (server-state) for oRPC calls
- oRPC (OpenAPI handler) exposed under `/rpc`
- Recharts for analytics visualizations
- Better Auth for auth + organizations

3. Routes and Navigation (SPA-like UX)

The app uses Next.js App Router (`app/*`). Navigation is client-side (`next/link`, `next/navigation`) and behaves like an SPA for most flows.

Key routes:

- `/` landing (`app/page.tsx`)
- `/login` authentication (`app/login/page.tsx`)
- `/dashboard` main UI (`app/dashboard/page.tsx`)
- `/analytics` analytics dashboard (`app/analytics/page.tsx`)
- `/ai` AI assistant (`app/ai/page.tsx`)
- `/profile` profile + export + API keys (`app/profile/page.tsx`)
- Organization flows under `app/(organizations)/*`

4. UI Architecture

High-level layering:

1. **Pages** (`app/**/page.tsx`) assemble features and wire data fetching.
2. **Feature components** (e.g. `app/dashboard/components/*`, `app/analytics/components/*`) implement screens and interactions.
3. **UI primitives** (`components/ui/*`) provide reusable building blocks (buttons, dialogs, tabs, tables, etc.).

5. State Management

- Local UI state: React `useState` for selection, dialogs, inputs, loading flags.
- Server state: TanStack Query caches RPC responses and handles invalidation.
- Auth/org context: Better Auth client hooks (session + active organization) are used for gating and scoping UI behavior.

6. API Integration

6.1 RPC entry point

- The server exposes the OpenAPI handler at `/rpc` (`app/(backend)/rpc/[...all]/route.ts`).
- OpenAPI spec and docs:
  - `/rpc/spec`
  - `/rpc/docs`

6.2 Client calls

- The front-end calls RPC methods via `safeClient` / `orpc` from `lib/orpc/orpc.ts`.
- Cookies are included so the session is available on the server.

7. Key User Flows

1. Sign in:
   - User goes to `/login`
   - On success, user lands in `/dashboard`
2. Organization onboarding:
   - If no active organization is selected, the user is redirected into the organization selection/creation flow.
3. Dashboard work:
   - Navigate a folder/spec tree
   - Create folders/specs, rename and delete
   - Manage requirements in the “Functional Requirements” tab
   - Manage/update generated code in the “Code” tab
   - Sync test status via report sync endpoints (RPC and/or webhook)
4. Analytics:
   - View aggregated organization metrics and charts
   - Switch between 7/30/90-day windows
5. AI assistant:
   - Send prompts to the assistant
   - Use provider/model selection
   - Reset session chat with “New chat”

8. Loading, Errors, and UX

- Loading states use a shared loader (`components/loader.tsx`) and per-feature placeholders.
- Error states show user-facing messages and avoid blank screens.
- Toast notifications (`sonner`) are used for user feedback in core mutations.

9. Testing

Tooling:

- Unit/component/workflow tests: Vitest + React Testing Library in `__tests__`
- E2E: Playwright under `e2e`

Common commands:

- `pnpm test`
- `pnpm test:coverage`
- `pnpm test:e2e`

10. Build PDF (Pandoc)

Run from repository root:

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