

# AutomaSpec

## AI-Powered Test Specification Management System

**Student:** Aliaksandr Samatyia

**Group:** Js

**Supervisor:** Volha Kuznetsova

**Date:** 2026

# The Problem: Testing Fragmentation

## Who suffers?

QA Engineers, Developers, and Product Managers in fast-paced teams.

## The Reality:

- **✗ Disconnected Workflows:** Requirements live in docs, tests live in code. Links are manual and fragile.
- **✗ Visibility Black Holes:** Stakeholders cannot verify if a specific requirement is actually covered by a passing test.
- **✗ Stale Documentation:** Test cases often lag behind code changes, leading to false confidence.
- **✗ Manual & Slow:** meaningful reporting requires manual spreadsheet updates.

*"We don't know if we broke the feature until users tell us."*

# The Solution: Unified Test Lifecycle

## How AutomaSpec solves it:

AutomaSpec acts as the **central nervous system** for quality assurance, syncing code, tests, and requirements.

## Key Capabilities:

-  **Deep Integration:** Automatically syncs Playwright & Vitest execution results to requirements.
-  **Live Traceability:** Requirement ↔ Test Spec ↔ Execution Result. All linked.
-  **AI Assistant:** Chat with your test suite to generate cases or explain failures.

## Why it's different:

Unlike erratic spreadsheets or siloed Jira plugins, AutomaSpec represents the **state of truth directly from CI/CD**.

# Demo: Core Workflow

## 1. Define Requirements:

Users create requirements linked to specs.

## 2. Sync Execution:

CI pipeline pushes results; coverage updates instantly.

## 3. Trace & Audit:

Drill down from a business goal to the specific test.

The screenshot displays the Automaspec Org application interface. On the left, a sidebar shows a file tree with a folder named 'Test8' expanded, containing subfolders 'Test7' and 'Test9', and files 'New Test' and 'New Test'. Other collapsed items include 'test723', 'test123', and 'test21'. At the top right, there are buttons for 'Analytics', a search icon, and a refresh icon. Below the sidebar, the main area is titled 'Test8' and shows a 'Statistics' card with the following data:

Subfolders	Test Specs	Passed	Failed
1	2	0	0

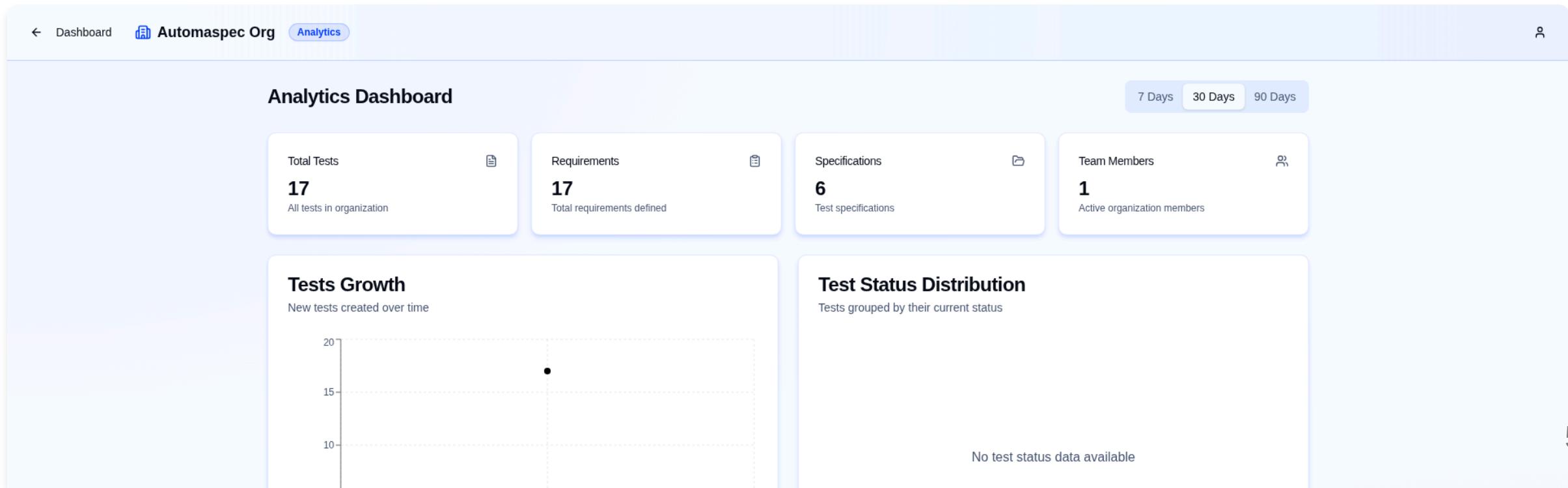
Below the statistics are cards for 'Skipped' (0) and 'Pending' (0). At the bottom, there is a section for 'Test Specs' with a button to '+ New Spec'.

# Demo: Analytics Dashboard

## Real-Time Insights:

Comprehensive metrics and visualizations for test coverage and execution trends.

- **Coverage Metrics:** Track requirement coverage over time.
- **Execution Trends:** Visualize test pass/fail rates.
- **Period Selection:** Analyze performance across different timeframes.



# Demo: Main Dashboard

## Centralized Test Management:

Organized view of projects, folders, and test specifications.

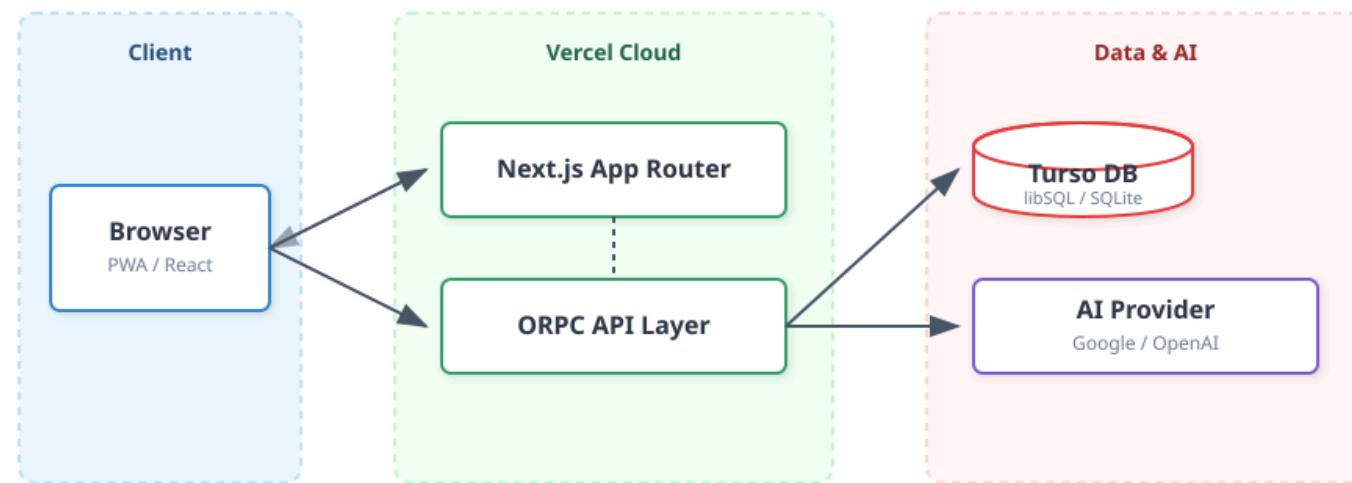
- **Hierarchical Structure:** Navigate through organizations and projects.
- **Quick Access:** Direct links to requirements and test specs.
- **Status Overview:** Visual indicators for test execution status.

The screenshot displays the Automaspec Org main dashboard. On the left, there's a sidebar with navigation links for 'Components' (Test Details Panel, Tree Component, Dashboard Tree View) and 'Database'. The main area is divided into several sections: 'Components' (with a 'Edit' and 'Delete' button), 'Statistics' (Subfolders: 0, Test Specs: 3, Passed: 0, Failed: 0, Skipped: 0, Pending: 7), 'Test Specs' (Test Details Panel, tree-display.test.ts, 0 tests), and 'Tree Component' (tree-display.test.ts, 0 tests). A blue button '+ New Spec' is located at the bottom right of the 'Test Specs' section. The top right corner shows a small number '6'.

# High-Level Architecture

## Key Components:

- **Frontend:** Next.js 16 (React 19), Tailwind CSS, Framer Motion.
- **Backend:** Serverless Functions via Vercel, ORPC for type-safe contracts.
- **Database:** Distributed SQLite (Turso) managed via Drizzle ORM.
- **AI Integration:** Vercel AI SDK into Google/OpenAI.



# Technology Stack

Category	Technology	Purpose
Framework	<a href="#">Next.js 16</a>	Full-stack React framework with App Router
Language	<a href="#">TypeScript</a>	Strict type safety across full stack
Database	<a href="#">Turso (LibSQL)</a>	Edge-compatible distributed SQLite
ORM	<a href="#">Drizzle ORM</a>	Type-safe SQL builder and schema management
API	<a href="#">ORPC</a>	End-to-end type-safe API contracts
Testing	<a href="#">Playwright + Vitest</a>	E2E and Unit testing frameworks
AI	<a href="#">Vercel AI SDK</a>	Integration with LLM providers (Google/OpenAI)

# Front-End Architecture

## WHY:

Needed a scalable, SEO-friendly SPA with robust server integration for a complex dashboard.

## WHAT:

- **App Router:** Hierarchical routing for Organizations/Projects.
- **Server State:** TanStack Query for caching & optimistic updates.
- **Type Safety:** End-to-end typed API calls via oRPC.
- **Components:** Modular UI using Radix Primitives.

## TECH:

Next.js 16, React 19, TanStack Query, Radix UI

```
// Type-safe reactive data fetching with TanStack Query
const [period] = useState<AnalyticsPeriod>('30d')
const { data } = useQuery(orpc.analytics.getMetrics.queryOptions({
  input: { period }
}))
```

# Adaptive User Interface

## WHY:

To provide a seamless experience for QA engineers across Desktop (4K), Tablet, and Mobile devices.

## WHAT:

- **Mobile-First:** Styles defined for small screens, scaling up via breakpoints ( `sm` , `md` , `lg` ).
- **Responsive Navigation:** Sidebar on desktop -> Drawer on mobile.
- **Theme Support:** System-aware Dark/Light mode integration.
- **Accessibility:** WCAG 2.1 AA compliance via Radix UI.

**TECH:** Tailwind CSS v4, Lucide Icons, next-themes

*Verified support for 16:9, 21:9, and mobile portrait aspect ratios.*

# API Documentation

## WHY:

Ensure external integrations and developers have an accurate source of truth.

## WHAT:

- **Auto-Generated:** Docs derived from Zod schemas.
- **Interactive:** Scalar UI for in-browser testing.
- **OpenAPI:** Exports valid 3.0 spec.
- **Zero Drift:** Docs update with code.

TECH: oRPC, Scalar UI, Zod, OpenAPI

The screenshot displays the Automaspec API documentation and testing interface. At the top, there's a search bar and a sidebar with a tree view showing categories like 'ai' (Chat with AI, tests, folders, specs, requirements, account, analytics), 'POST' method, and version 'v1.0.0 OAS 3.1.1'. Below this is the 'Automaspec API' title, a 'Download OpenAPI Document' button, and sections for 'Server' (URL: https://automaspec.vercel.app/rpc) and 'Client Libraries' (Shell, Ruby, Node.js, PHP, Python, More). The main content area shows an 'ai' section with an 'Operations' table for a 'POST /ai/chat' operation. Below this is a 'Chat with AI' section with a description, headers (Accept: Example, Content-Type: Example), and a 'curl' command example for the POST /ai/chat endpoint. The curl command is as follows:

```
POST /ai/chat
1 curl https://automaspec.vercel.app/rpc/ai/chat \
2 --request POST \
3 --header 'Content-Type: application/json' \
4 --data '{
5   "messages": [
6     {
7       "role": "user",
8     }
9   ]
}'
```

# CI/CD Pipeline

## WHY:

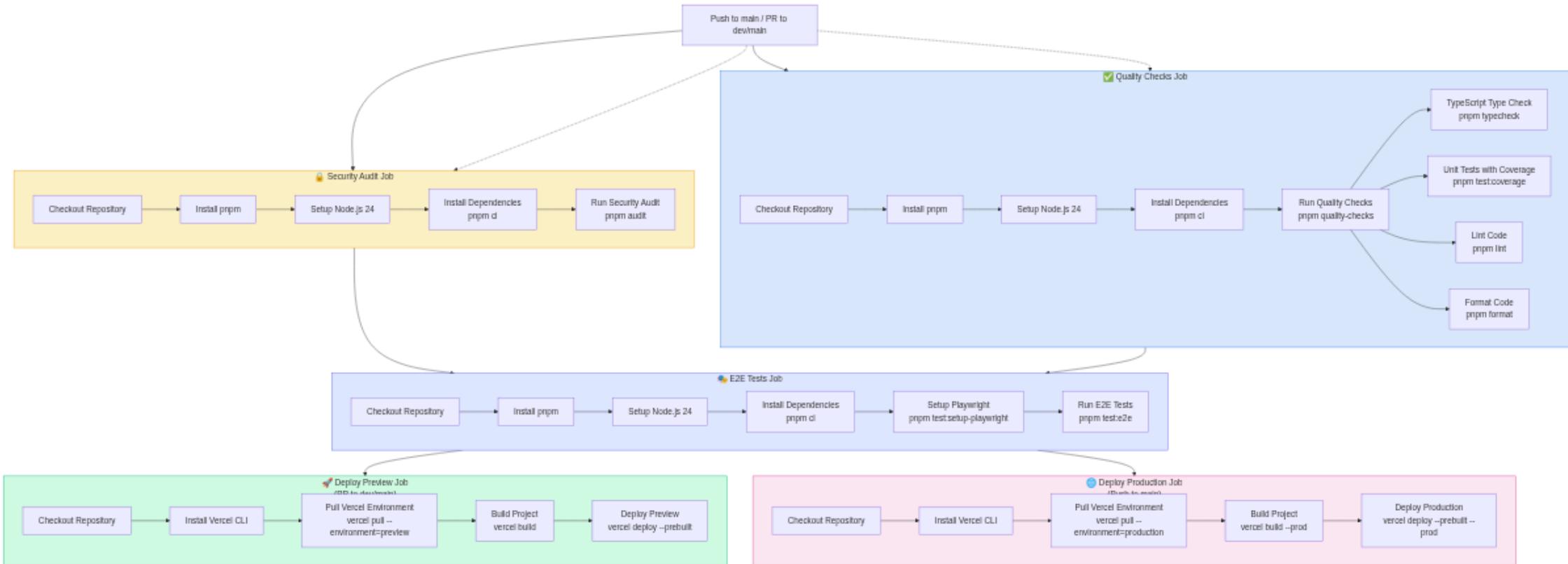
To automate quality control and ensure safe, frequent deployments to production.

## WHAT:

1. **Quality Gate:** Lint (`oxlint`), Format, Typecheck before merge.
2. **Security:** Automated `pnpm audit` for dependencies.
3. **Test Automation:** Unit (Vitest) + E2E (Playwright) execution.
4. **Delivery:** Auto-deploy to Vercel (Preview/Prod).

**TECH:** GitHub Actions, Vercel CLI, Docker, Lefthook

# CI/CD Pipeline: Diagram



# Containerization

## WHY:

To guarantee environment consistency ("works on my machine") and enable portability.

## WHAT:

- **Multi-Stage Build:** `deps` → `builder` → `runner` (Optimized layers).
- **Standalone Mode:** Trims `node_modules` for ~100MB final image.
- **Security:** Runs as non-root user (`nextjs`).
- **Orchestration:** Docker Compose profiles for Dev vs. Prod.

## TECH:

Docker, Docker Compose, node-alpine

```
# Final Stage
FROM base AS runner
USER nextjs
COPY --from=builder /app/.next/standalone ./
CMD ["node", "server.js"]
```

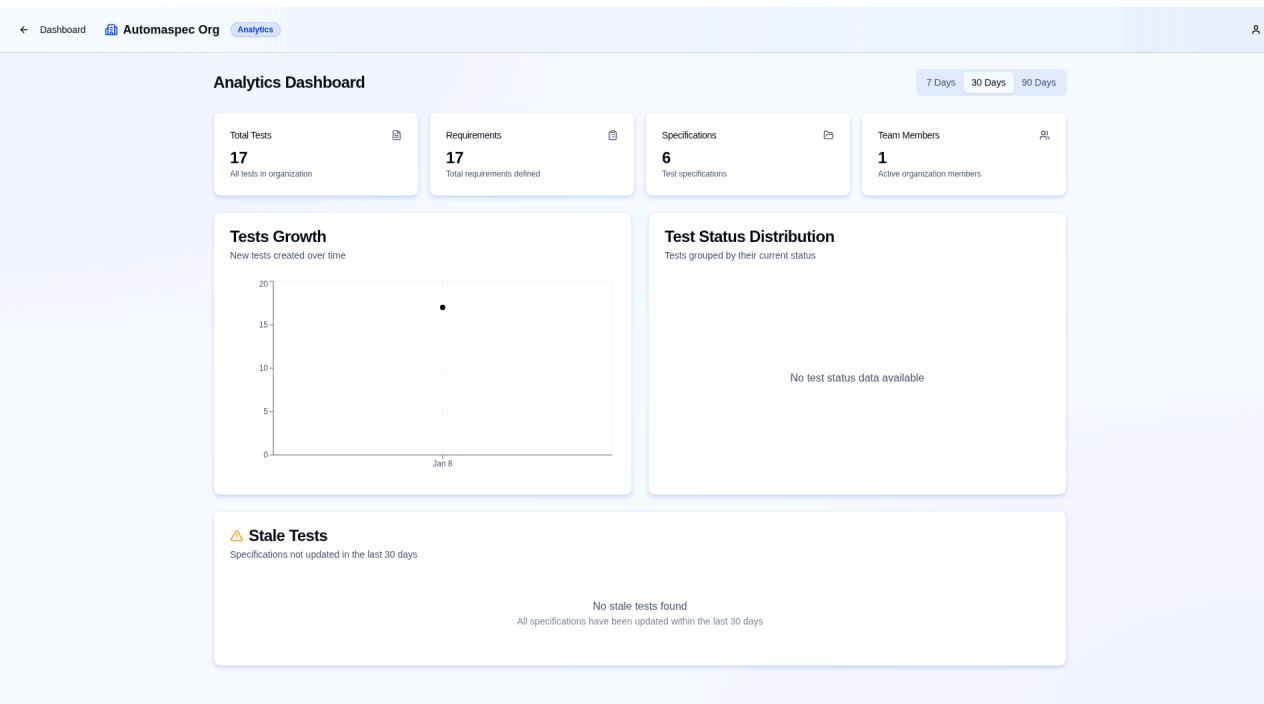
# Challenges & Solutions

Challenge	Solution
<b>Vercel vs Docker</b>	<p><i>Problem:</i> Vercel doesn't run Docker.</p> <p><i>Fix:</i> Used Hybrid strategy—Docker for local dev/testing reliability, Vercel for scalable Serverless production.</p>
<b>Type Synchronization</b>	<p><i>Problem:</i> Keeping API and Frontend types in sync.</p> <p><i>Fix:</i> Implemented <b>oRPC</b> to infer frontend types directly from backend Zod schemas.</p>
<b>Complex State</b>	<p><i>Problem:</i> Managing real-time spec updates.</p> <p><i>Fix:</i> Utilized <b>TanStack Query</b> for efficient server-state caching and optimistic UI updates.</p>

# Results

## Project Checklist

- [x] **Core MVP:** Requirement management & Test syncing.
- [x] **Architecture:** Scalable Next.js 16 + Serverless setup.
- [x] **Quality:** CI/CD pipeline with 100% E2E critical flow coverage.
- [x] **Documentation:** Auto-generated API Reference.



# Q&A

**Production:** [automaspec.vercel.app](https://automaspec.vercel.app)

**Repository:** [github.com/qweered/automaspec](https://github.com/qweered/automaspec)

**Documentation:** [/rpc/docs](#) (Scalar)

**Thank You!**

**Student:** Aliaksandr Samatyia

**Contact:** [aliaksandr.samatyia@stud.ehu.lt](mailto:aliaksandr.samatyia@stud.ehu.lt)