Software Testing Documentation

Project: Amrita Placement Tracker (APT)

Date: February 9, 2026

1 Introduction

This document outlines the software testing strategies, tools, and results for the Amrita Placement Tracker (APT) application. The testing process aims to ensure the reliability, functionality, and performance of both the backend (Node.js/Express) and frontend (React/Vite) components.

2 Testing Strategy

A comprehensive unit testing strategy was employed to verify individual components and modules in isolation.

2.1 Backend Strategy

- **Unit Testing:** Focused on testing controllers, services, middleware, and models independently.
- **Mocking:** External dependencies such as databases (MongoDB via Mongoose), authentication services (JWT, bcrypt), and third-party APIs (EmailJS, AI services) were mocked to isolate the code under test.
- **Coverage:** Targeted critical paths including authentication, user management, drive management, reports, and AI services.

2.2 Frontend Strategy

- **Component Testing:** Verified the rendering and behavior of React components in a simulated DOM environment using JSDOM.
- **Event Simulation:** User interactions such as clicks, form inputs, and navigation were simulated to test event handlers.
- **State Validation:** Assertions were made on the DOM state changes resulting from user actions and asynchronous operations.

3 Tools Used

The following tools were selected for their robustness and compatibility with the project stack.

Component	Tool	Reason for Selection
Backend Testing	**Jest**	Industry standard for JavaScript testing. Built-in test runner, assertion library, and powerful mocking capabilities ideal for Node.js environments.
Frontend Testing	**Vitest**	Native test runner for Vite projects. Offers faster execution (HMR) and seamless integration with Vite configuration compared to Jest.
DOM Simulation	**JSDOM**	Simulates a browser-like environment in Node.js for testing React components without a browser.
React Utilities	**RTL**	(React Testing Library) Encourages testing best practices by querying the DOM as a user would (e.g., by text or role).

4 Test Execution Methodology

Tests were executed in the local development environment using standard npm scripts.

Steps Performed:

1. **Environment Setup:** Installed dependencies (`jest`, `vitest`, `jsdom`) and configured test environments (`jest.config.js`, `vite.config.js`).
2. **Code Analysis:** Analyzed source code to identify testable units (controllers, components).
3. **Test Creation:** Created test files mirroring the source structure (e.g., `src/controllers/auth.js` → `tests/jest/controllers/auth.test.js`).
4. **Mock Implementation:** Implemented mocks for database calls (`User.findOne`), API calls (`axios.get`), and context hooks (`useAuth`).
5. **Execution:** Ran tests via `npm run test:jest` (backend) and `npx vitest` (frontend).
6. **Refinement:** Analyzed failures, fixed bugs (e.g., `PlacementCountdown` timer logic), and refined test cases.

5 Testing Metrics & Results

5.1 Summary

Suite	Total Tests	Passed	Failed/Skipped
Backend (Jest)	78	77	1 (Skipped)
Frontend (Vitest)	24	24	0
Total	**102**	**101**	**1**

5.2 Detailed Breakdown

Backend Units (Passed: 77)

- **Authentication:** 6 tests (Login, Registration, Password Hashing)
- **User Management:** ∼12 tests (Profile retrieval, Updates, Role verification)
- **Drives:** ∼15 tests (CRUD operations, Application handling)
- **Resources & Reports:** ∼18 tests (File generation, Data export)
- **Alumni & AI:** ∼14 tests (Directory listing, Resume analysis)
- **Middleware:** ∼12 tests (Auth protection, Role-based access)

Frontend Units (Passed: 24)

- **Core UI Components:** 15 tests
  - `CompanyLogo`: Styling, fallback rendering.
  - `PlacementCountdown`: Timer logic, zero-state handling.
  - `SkillProgress`: Visual rendering based on props.
  - `Navbar`: Role-based links, Theme toggle, Logout.
- **Authentication Pages:** 9 tests
  - `Login.jsx`: Form rendering, Input handling, API integration (mocked), Navigation.
  - `Register.jsx`: Multi-step form flow, Validation (Email domain), API interaction.

6 Conclusion

The testing phase has achieved high coverage across critical modules of the APT application.

- **Reliability:** Core flows (Auth, Drives) are strictly verified.
- **Maintainability:** The separation of concerns in testing (Frontend vs Backend) allows for independent scaling of test suites.
- **Quality Assurance:** 101 passing tests provide a strong safety net for future development.