**Test Plan — Smart Traffic Automation (mini project)**

**Test Plan Identifier**

ST-TP-001

**References**

* Project SRS, Architecture Diagram, High-Level & Low-Level Design
* Backend API spec: /api/auth, /api/roads, /api/bookings
* Frontend: React App (Login/Register/Admin/Booking)
* ML predictor: backend/ml/predict.py

**Introduction**

This plan defines test objectives and test cases for the Smart Traffic Automation web app. Purpose: verify correctness of core flows (auth, booking, admin), UI interactions (graph selector), backend integration (Dijkstra + ML), and basic non-functional checks.

**Test Items**

* User registration & login (JWT)
* Admin login & dashboard
* Route booking (frontend → backend → ML → DB)
* Roads API and road status update
* Graph-based source/destination selection and path highlighting
* Booking visibility rules (user vs admin)
* Basic security and input validation
* Basic performance/availability sanity

**Software Risk Issues**

* ML process failure (predictor crashes)
* DB connectivity issues
* JWT expiry/misuse
* Graph click interactions inconsistent across browsers
* Concurrency: overlapping bookings causing incorrect occupancy

**Features to be Tested**

* Register new user
* Login (valid / invalid)
* Booking creation and response (path + recommended speed)
* Admin viewing all bookings and exporting CSV
* Toggling road status (open/closed)
* Graph selector: click/Ctrl+click to set source/destination; highlight path returned
* Booking visibility (users see own bookings only)
* Predictor fallback (heuristic when model unavailable)

**Features not to be Tested**

* Full ML model training pipeline (only inference/fallback tested)
* Load testing beyond basic responsiveness
* Email verification, payment, or external integrations

**Approach**

* Functional manual tests executed locally (dev server + local MongoDB).
* A small set of automated unit tests (optional) for backend utils (dijkstra).
* Use browser DevTools and Postman/curl for API verification.
* Record results in a simple spreadsheet or markdown table.

**Item Pass/Fail Criteria**

* **Pass**: Actual result matches expected result exactly (or within defined tolerance for numeric values like recommended speed ±5 km/h).
* **Fail**: Any mismatch in expected behavior, unhandled errors, or crashes.

**Suspension Criteria and Resumption Requirements**

* Suspend testing if backend is down, DB connection failing, or ML script consistently throws unrecoverable errors.
* Resume after service restored and smoke test passes (health endpoint responds and DB accessible).

**Test Deliverables**

* Test cases (this document)
* Test execution log (pass/fail with notes)
* Bug reports for failed cases (steps to reproduce, console logs)
* Optional: small unit test scripts for Dijkstra and auth

**Remaining Test Tasks**

* Add automated tests for critical backend functions (auth, booking, dijkstra) later.
* Cross-browser testing (Chrome, Firefox, Edge) if time allows.

**Environmental Needs**

* Local MongoDB instance (or Atlas) reachable by backend
* Node.js environment for backend (port 4000)
* Vite React dev server for frontend (port 5173)
* Python (for ML predictor) in PATH
* Postman or curl for API tests

**Staffing and Training Needs**

* 1–2 developers/testers who know how to run backend/frontend locally and use DevTools/Postman.

**Responsibilities**

* Tester: execute test cases, record results.
* Dev: fix issues found, provide logs/stack traces.

**Schedule**

* Smoke tests + auth + registration: 0.5 day
* Booking flows + admin features: 1 day
* UI/UX graph checks + regression: 0.5 day
* Buffer / bug fixes: 1 day

**Planning Risks and Contingencies**

* If ML predictor fails, use heuristic fallback and record as reduced functionality.
* If DB seeding fails, run seed script again; keep backup graph.json.

**Approvals**

* Project lead / instructor to approve final test report.

**Glossary**

* ML: Machine Learning
* DB: Database (MongoDB)
* JWT: JSON Web Token