I’m a non‑developer. You (Claude) have direct access to the mivton directory on my desktop. This is a Node.js + Express + PostgreSQL app deployed on Railway with a custom domain. We completed **Phases 1–3** from the “🚀 Mivton Development Plan – Complete Roadmap” (auth, dashboard/UI, friends system basics). We now need an **automated testing + fix pass** that you run locally and also against the live site, then fix issues one by one, committing changes after each fix. The project structure and phases are in that plan; keep your work consistent with it. After Phases 1–3 are clean, set the baseline for **Phase 4 (Socket.IO messaging)** tests.

**Tech stack (for your setup):** Node.js/Express, PostgreSQL (Railway), vanilla JS front‑end, planned Socket.IO, OpenAI translation later. Directory layout is as in the plan (server.js, routes/, middleware/, database/, public/, etc.).

**Environment details you can assume:**

* Local dir: ~/Desktop/mivton
* Prod URL (replace if different): https://www.mivton.com
* Railway Postgres is configured via env vars in production. For **local tests**, create a **separate local test DB**.

**Your objectives (do these in order)**

1. **Install a lightweight test toolchain (dev‑only):**
   * Add **Jest** for unit tests, **Supertest** for API tests, **Playwright** for end‑to‑end UI checks, and **ESLint** for quick static checks.
   * Update package.json with scripts (create any missing files as needed):
     + "lint": "eslint ."
     + "test:unit": "jest --runInBand"
     + "test:api": "jest --runInBand tests/api"
     + "test:e2e:headless": "playwright test --reporter=list"
     + "test:all": "npm run lint && npm run test:unit && npm run test:api && npm run test:e2e:headless"
   * If TypeScript isn’t used, stay with JS (optionally add JSDoc types).
2. **Create safe local test envs:**
   * Add .env.test with a **separate** Postgres DB (local or a Railway fork dedicated to testing).
   * Provide npm script hooks so tests bootstrap and tear down the test DB.
   * Add database/schema.sql seeding for **minimal users/friends data** plus teardown scripts; never touch prod data. (Use the schema paths from our plan.)
3. **Write Phase‑aligned test suites for what exists today (Phases 1–3):**
4. **Phase 1 – Auth & Sessions**

**API tests (Supertest)**

* + POST /auth/register happy path: creates user, hashed password, session created; rejects weak/duplicate email; email verification placeholder doesn’t break flow.
  + POST /auth/login success + failure (bad pass, unknown user), session cookie set, rate‑limit guard present (if implemented).
  + Session persistence: authenticated route returns 200 when logged in, 401 when not.

**Unit tests (Jest)**

* + Password hashing/validation helpers.
  + Input validation middleware rejects invalid payloads.
  + DB connection utilities handle errors gracefully.

**Phase 2 – Dashboard/UI Components**

**E2E (Playwright) – Local and Prod**

* + Landing loads with no console errors; main assets 200 OK.
  + Login flow via UI, redirect to dashboard, sidebar navigation works, responsive layout below 420px width.
  + Profile panel loads, toggles work, language selector visible and persists in localStorage/session.

**Phase 3 – Friends System (what’s implemented so far)**

**API tests (Supertest)**

* + POST /friends/request, POST /friends/accept, POST /friends/decline reflect correct rows in friend\_requests/friends tables, protect against duplicates and invalid states.
  + Blocking/unblocking endpoints (if implemented) refuse requests from blocked users.
  + Pagination/search work and are sanitized.

**E2E (Playwright)**

* + User A sends request to User B; B sees notification; B accepts; both see each other in list.
  + Blocking flow (if present) hides user and prevents messaging attempts.

1. **Add a quick Production smoke test suite (non‑destructive):**
   * Use Playwright to hit https://www.mivton.com:
     + GET / returns 200, main scripts/styles load, no severe console errors.
     + Login page loads; form validation messages appear for empty/invalid inputs (don’t actually register new accounts repeatedly—mock or reuse a throwaway testing account named test+e2e@mivton.com).
     + Basic nav clicks don’t produce 4xx/5xx.
   * Collect **console errors**, network failures, and JS exceptions; export a short HTML/text report in /test-reports.
2. **Run everything and produce a single failure list:**
   * npm run test:all locally.
   * npm run test:e2e:headless against **prod** with a separate config (e.g., playwright.config.prod.ts).
   * Aggregate failures by **area and severity** (Security > Data‑loss > Broken core flow > UI polish). Save to /test-reports/summary.md.
3. **Fix issues one by one, smallest‑blast‑radius first:**
   * For each failure:

a) Reproduce with a focused test.

b) Patch code.

c) Add/adjust tests to prevent regressions.

d) Run only that spec, then the full suite.

e) Commit with a clear message: fix(auth): handle duplicate emails in register etc.

* + Keep DB migrations backward‑compatible. If you must alter schema, write a migration and update seeds.

1. **Prepare groundwork for Phase 4 (Socket.IO) tests (don’t implement chat yet):**
   * Add a **placeholder** socket/handlers.test.js with:
     + connects/disconnects with auth token middleware mock;
     + refuses unauthenticated sockets;
     + reserves test helpers for message delivery/receipt ack assertions to be filled in during Phase 4.
   * Add Playwright helpers to open two browser contexts (User A/B) for future real‑time chat tests.
2. **Developer ergonomics & CI (optional but recommended):**
   * Add eslint --fix pre‑commit hook with Husky.
   * (Optional) Add a GitHub Action (node + Postgres service) running npm run test:all on push.

**Acceptance criteria (mark these off as you go)**

* Dev dependencies installed; scripts added.
* .env.test + isolated test DB working; seed + teardown in place.
* Jest + Supertest tests covering Phase 1–3 APIs and utilities.
* Playwright E2E covers auth, dashboard, and friends flows (local & prod smoke).
* /test-reports/summary.md generated with failures grouped by severity and area, plus a prod smoke report.
* All **critical** and **high** issues fixed with corresponding tests.
* Phase 4 test scaffolding created (socket auth/connect placeholders, dual-context helper).
* Lint passes; tests green locally; prod smoke shows no blocking errors.

**Commands to run (you execute them; confirm each step)**

1. Open the project folder and initialize test tooling:
   * cd ~/Desktop/mivton
   * npm i -D jest supertest @types/jest playwright eslint eslint-config-standard eslint-plugin-import eslint-plugin-n promise eslint-plugin-nod
   * npx playwright install --with-deps
   * Create/update package.json scripts as specified.
2. Create .env.test (you generate safely).
3. Create tests/unit, tests/api, tests/e2e with the suites listed above.
4. Add DB seed/teardown scripts for tests in database/.
5. Run: npm run test:all and then prod smoke via Playwright prod config.
6. Produce /test-reports/summary.md, then begin **fix‑and‑commit** cycle until all critical/high issues are resolved.

**Important:** After each fix, re‑run only the impacted tests, then the full suite. Keep your messages short and specific. If you need credentials for a prod smoke test account, ask me once; otherwise run read‑only checks.

**Scope boundaries**

* Do **not** implement Phase 4 features yet—just the **test scaffolding**.
* Do **not** spam prod with registrations; use one reusable test account or mock.
* All changes must be local to the mivton directory and compatible with Railway deployment.

**Deliverables for me**

* /test-reports/summary.md (with clear list of bugs found both locally and on prod, linked to failing tests).
* A short changelog of fixes with file paths.
* Instructions (if any) for me to redeploy to Railway via CLI.

If anything in the plan conflicts with the “🚀 Mivton Development Plan – Complete Roadmap,” keep the plan as the source of truth and note the conflict in the report before proceeding.