TigerTix — Sprint 2 Testing Strategy, Execution, and Evidence

Project: TigerTix (Admin Service, Client Service, LLM Service, React/Vite Frontend)
Scope: Automated unit/integration tests + frontend tests + manual exploratory checks (NL/Voice, A11y, Concurrency)

1) Overall Testing Strategy

- **Unit tests:** validate handlers, input parsing/validation, and small helpers.
- **Integration tests:** exercise HTTP routes via Supertest against an isolated SQLite DB (temp file initialized from `init.sql`); external HTTP calls are mocked.
- **Frontend tests:** React Testing Library (RTL) + Vitest on JSDOM validate UI flows (EventList purchase, ChatAssistant propose→confirm).
- **Manual tests:** short scripts to verify voice beeps, aria-live announcements, keyboard focus, and DB transaction safety under concurrent requests.

2) Test Environments & Tooling

Component	Tooling / Notes
Backend runner	**Jest** (`testEnvironment: "node"`) + **Supertest**
HTTP mocking	**axios** mocked (LLM service) to avoid real OpenAI/Client calls
Database	**SQLite** temp file per suite via `process.env.DB_PATH`, schema from `init.sql`
Frontend runner	**Vitest** + **JSDOM** + **@testing-library/react/user-event/jest-dom**
Voice	Voice helpers (STT/TTS/beeps) **mocked** to avoid audio and browser APIs

3) Automated Tests (By Component)

3.1 Admin Service (Event Creation)

Location: `backend/admin-service/tests/admin.routes.test.js`
Notes: ESM enabled for Jest; API key header set; Zod (or equivalent) validates payload.

ID	Case	Expected Result
A1	`POST /api/admin/events` with invalid payload	**400** Bad Request (validation failure)
A2	`POST /api/admin/events` with valid payload	**201** Created + event payload
A3	`POST /api/admin/events` duplicate (name+date)	**409** Conflict (or idempotent **201** if allowed)

3.2 Client Service (List & Purchase with Concurrency)

Location: `backend/client-service/test/client.routes.test.js`
DB: temp file created before import; schema applied from `../shared-db/init.sql`; cleaned after tests.
Routes: `/api/events`, `/api/events/:id/purchase` (defaults to 1 ticket if body omitted).

ID	Case	Expected Result
C1	`GET /api/events`	**200** + JSON array
C2	`POST /api/events/:id/purchase` sequential (stock=2)	**200**, **200**, then **409** on third purchase
C3	`POST /api/events/:id/purchase` concurrent (stock=1)	One **200**, one **409**; **no oversell**

3.3 LLM Service (Parse & Confirm, mocked)

Location: `backend/llm-service/tests/llm.routes.test.js` (CommonJS)
Server: `server.js` exports `app` and only `listen()` when run directly.
Mocks: `jest.mock('axios')` for all external calls.

ID	Case	Expected Result

L1	`POST /api/llm/parse` ("book 2 for Jazz Night")	**200**; `intent="propose_booking"`, `tickets=2`, `requiresConfirmation=true`
L2	`POST /api/llm/confirm` (resolves ID + books via client)	**200**; `{ eventId, tickets }` from downstream purchase
L3	`POST /api/llm/confirm` (unknown event)	**404** Not Found

3.4 Frontend (React/Vite)

Runner: Vitest + RTL on JSDOM
Locations: `src/components/tests/EventList.test.jsx`, `src/components/tests/ChatAssistant.test.jsx`
Notes: `global.fetch` mocked by **URL** (not call order) to avoid flakes; voice helpers mocked; `App.test.js` renamed to `.jsx` so JSX parses.

ID	Case	Expected Result
F1	**EventList** renders and buys one ticket	Live region announces purchase; events refetch shows decremented availability
F2	**ChatAssistant** propose → confirm booking	Proposal text appears (visible + aria-live); on confirm, success message displayed

4) Database Isolation & Initialization

- Tests create a **temporary SQLite file** (outside `shared-db/`) and set `DB_PATH` **before importing** service routes/models so the connection targets the temp DB.
- The schema from `init.sql` is applied at suite start; rows are seeded per test case.
- Temp DB is removed in `afterAll`, ensuring **no side effects** and **deterministic** behavior.

5) Manual Tests (Exploratory, for Demo Video)

A. Natural Language (Text)

1. Type **“show events”** → chat shows list.
2. Type **“book 2 for Jazz Night”** → assistant shows a **proposal** (no auto-book).
3. Click **Confirm** → success; EventList count decreases accordingly.

B. Voice + Beeps

1. Click **Talk** → short **start beep**; say *“book one ticket for Jazz Night”*.
2. Click **Stop** (or wait for end) → **double beep**; recognized text fills input; **Send** → proposal → **Confirm**.

C. Accessibility

1. **Keyboard**: Tab through skip link & controls; visible focus ring.
2. **Screen reader**: **aria-live** region announces purchases or errors (EventList).
3. **Landmarks**: main banner, main, buttons have accessible names.

D. Concurrency (DB Transactions)

1. Seed event with **1** ticket.
2. In two terminals, POST `/api/events/:id/purchase` concurrently.
3. Expect exactly **one 200** and **one 409**; tickets never below zero.

6) Notable Findings & Fixes

- **Jest vs ESM**:
    - Admin/Client: enabled ESM for Jest (or used CommonJS tests).
    - LLM: exported `app` and used CommonJS tests; avoided server auto-listen under tests.

- **Client-service DB**: `DB_PATH` must be set **before** importing routes/models; apply `init.sql` immediately.

- **Frontend**:

    - `App.test.jsx` (JSX extension) fixed JSX parse error.

    - `ChatAssistant` test uses **URL-based fetch mocks** (parse, confirm, events) and non-order-dependent assertions; proposal appears in both chat log and aria-live (assert with `findAllByText` or scope to chat log).

    - Voice helpers mocked so tests don't access audio APIs.

7) Commands — How to Run

- **Admin Service**
  ```
  cd backend/admin-service && npm test
  ```

- **Client Service**
  ```
  cd backend/client-service && npm test
  ```

- **LLM Service**
  ```
  cd backend/llm-service && npm test
  ```

- **Frontend**
  ```
  cd frontend && npm run test
  ```