TigerTix – Sprint 3 Report

CPSC 3720 – November 18, 2025

Group Members: Isabella John, Levi Bates, Price Medlin

---

1. Introduction

Sprint 3 focused on strengthening TigerTix in three main areas: accessibility, security, and testing. We built a complete authentication flow using bcrypt and JWTs, added multi-level testing (unit, integration, E2E), and updated the UI to be usable with a keyboard and screen reader. These changes move TigerTix from a basic prototype to something more secure, reliable, and inclusive.

2. Accessibility for Visually Impaired Users

2.1 - Frontend Accessibility Improvements

We updated the frontend so visually impaired students can navigate and book tickets without a mouse. Some key additions include:

- **Semantic HTML** (<header>, <main>, <nav>) so screen readers understand the layout.
- **Skip link** at the top to jump directly to the main content.
- **Tab-friendly navigation** with consistent focus outlines and Enter/Space activation.
- **Meaningful ARIA labels** for icon-only buttons, especially the voice input button (aria-label, aria-pressed).
- **Live regions** (aria-live="polite") so updates—like booking confirmations—are announced automatically.
- **WCAG-compliant contrast** for buttons and text.

2.2 - Accessibility Testing

We used keyboard-only navigation and VoiceOver to ensure headings, buttons, and status messages were read correctly and in a logical order.

3. Security: Hashed Passwords & Token Authentication

3.1 - Bcrypt password hashing

Passwords are never stored in plaintext. On registration, the password is hashed with bcrypt (cost = 12). On login, bcrypt.compare() checks the password against the stored hash.

Why this matters

- A DB leak doesn't expose real passwords.
- Bcrypt salts each password automatically.
- The cost factor slows brute-force attacks.

3.2 - JWT-based authentication

After login, the auth service issues a JWT containing the user id and email. The token lasts ~30 minutes and is sent either as an HTTP-only cookie or a Bearer token. All other microservices use the same secret to validate it.

Why JWT fits our architecture

- Works cleanly across multiple microservices.
- Stateless—no shared session store required.
- Short expiration improves security.

3.3 - Remaining vulnerabilities

- No rate limiting → brute-force risk.
- Possible XSS issues if tokens ever end up in sessionStorage.
- CSRF protection should be strengthened.
- JWT secret rotation not yet implemented.

4. Testing: Unit, Integration, and End-to-End

4.1 - Unit Tests

Small, isolated tests for:

- Password/validation helpers
- JWT utilities

- React components (focus behavior, labels, rendering)

They catch logic errors but not cross-service issues.

4.2 - Integration Tests

Test real service interactions:

- Register → login → protected route
- Event creation and retrieval
- Ticket purchase updates SQLite correctly

These find API/middleware/database issues unit tests can't detect.

4.3 - End-to-End (E2E) Tests

Simulate full user flows:

- Register → login → buy ticket
- LLM booking flow → confirmation → purchase

They verify the system works correctly for real users.

4.4 - Why All Three Are Needed

Unit tests can pass even while:

- The frontend isn't sending tokens
- Middleware is misconfigured
- Transactions fail under concurrency

Integration + E2E tests catch these real-world problems.

5. Code Quality Standards

5.1 - Naming & Structure

- **camelCase** for variables/functions
- **PascalCase** for React components
- **UPPER_SNAKE_CASE** for constants
- Backend uses routes/, controllers/, models/, __tests__/

- Frontend uses pages/, components/, auth/, __tests__/

5.2 - Style & Maintainability

- Early returns for cleaner flow
- Small, single-purpose functions
- Comments focus on *why* important logic exists

6. Updated Architecture & Flow

6.1 - Architecture Overview

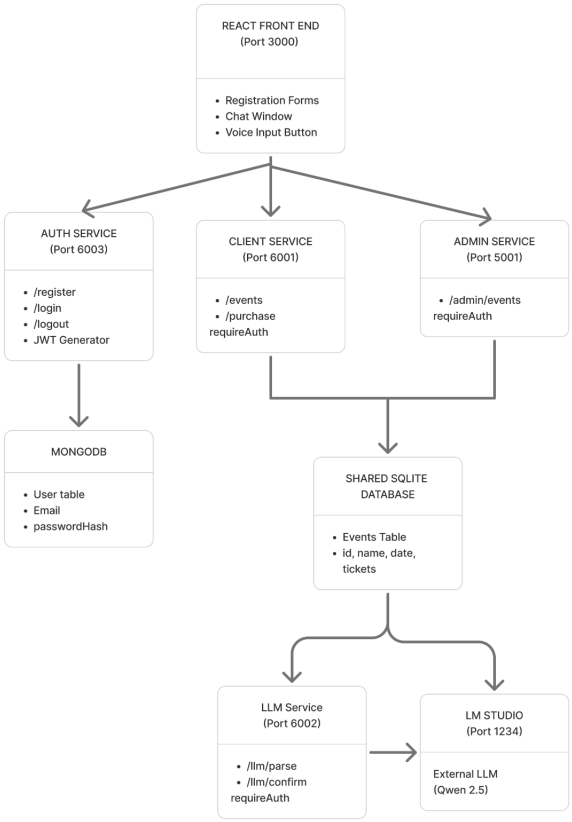
Components:

- React frontend
- Admin, Client, LLM, Auth microservices
- SQLite DB for events/tickets
- Separate user store for authentication

6.2 - Typical Request Flows

- **Registration/Login**: Frontend → Auth → bcrypt hash → JWT returned
- **Protected Action**: Frontend sends token → route validated → DB transaction runs
- **LLM Booking**: User message → LLM parses → confirmation → token-protected purchase

The Auth service now anchors all authentication, and other services validate tokens the same way.



7. Conclusion

Sprint 3 delivered strong improvements to TigerTix:

- Full accessibility support for visually impaired users
- Secure password hashing and JWT authentication
- Layered testing for reliability
- Clean, consistent code quality
- Updated architecture with unified authentication

Combined, these upgrades make TigerTix much more secure, usable, and production-ready.