TigerTix – Smart Event Ticketing System

TigerTix is a microservice-based event ticketing application that supports event creation, ticket purchasing, secure user authentication, and natural-language booking via an LLM integration. The system is designed following distributed software architecture principles and includes full regression testing, accessibility support, and modular service boundaries.

Project Overview

TigerTix allows users to:

- View and purchase event tickets
- Create and manage events (Admin users)
- Register and log in securely using JWT authentication
- Interact with the system using natural language (LLM-driven booking)
- Access protected routes through token-based authentication
- Maintain consistent booking and ticket counts via shared database access

The project was developed as part of CPSC 3720 – Software Engineering, applying microservices, testing methodologies, accessibility standards, and secure coding practices.

Tech Stack

Frontend

- **React (JavaScript)**
- React Router

- Fetch API for backend communication

Backend Microservices

Service	Description	Tech
Admin Service (Port 5001)	Create + edit events	Node.js, Express, SQLite
Client Service (Port 6001)	View events + purchase tickets	Node.js, Express, SQLite
User Authentication Service (Port 4000)	Registration, Login, JWT handling	Node.js, Express, bcryptjs, jsonwebtoken, cookie-parser
LLM Booking Service (Port 7001)	Parse natural-language booking requests	Node.js, Express, OpenAI API

Database

- **SQLite** (shared `events` database)
- **SQLite authentication DB** (users)

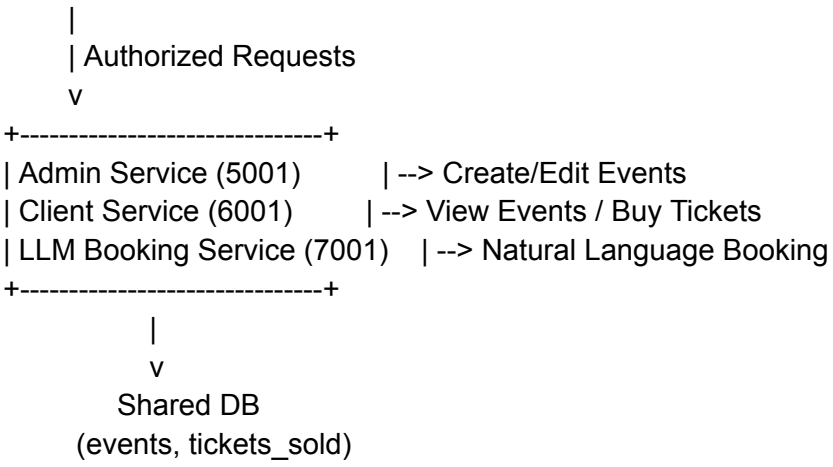
LLM API

- OpenAI API (or compatible local LLM)

Architecture Summary

TigerTix uses a multi-service distributed architecture with a shared database layer.

Browser / Frontend (React)
    |
    | Register / Login / Logout
    v
User Authentication Service (4000)
    | issues JWT
    v
--- HTTP-only Cookie ---



Key Flows

- JWT is verified by each protected route
- Admin + Client + LLM microservices communicate with the shared DB
- Authentication microservice stores hashed passwords only
- LLM service interprets text but *never books automatically* without confirmation

---

Installation & Setup Instructions

1. Clone the Repository

git clone https://github.com/quad0605/TigerTix.git
cd TigerTix

---

2. Install Dependencies for Each Microservice

Admin Service

cd backend/admin-service
npm install

Client Service

```
cd ../client-service
npm install
```

Auth Service

```
cd ../user-authentication
npm install
```

LLM Booking Service

```
cd ../llm-booking
npm install
```

---

3. Install Frontend

```
cd ../../frontend
npm install
```

---

4. Set Up Databases

The project auto-creates SQLite files if they do not exist:

```
shared-db/database.sqlite   → Events + ticket counts
auth.db                     → User accounts
```

---

5. Run All Services

Admin Service

```
cd backend/admin-service
npm start
```

Client Service

```
cd ../client-service
npm start
```

Authentication Service

cd ../user-authentication
npm start

LLM Booking Service

cd ../llm-booking
npm start

Frontend

cd frontend
npm start

Environment Variables

Create a `.env` file inside **each microservice** where needed.

Minimum Required Config

User Authentication Service

JWT_SECRET=your_secret_key
JWT_EXPIRES_IN=30m
COOKIE_SECURE=false
DB_FILE=./shared-db/auth.db

LLM Booking Microservice

OPENAI_API_KEY= whatever your key is
MODEL=gpt-4o-mini

Frontend

REACT_APP_ADMIN_URL=http://localhost:5001
REACT_APP_CLIENT_URL=http://localhost:6001
REACT_APP_AUTH_URL=http://localhost:4000
REACT_APP_LLM_URL=http://localhost:7001

Running Regression Tests

Backend Tests

Tests use Jest + Supertest + in-memory SQLite.

Run all tests:

npm test

Run only authentication tests:

cd backend/user-authentication
npm test

What Tests Cover

- User Registration
- Login & hashed password verification
- JWT issuance & cookie handling
- Protected routes (middleware)
- Logout & token clearing
- Admin event creation
- Client ticket purchasing
- LLM request parsing
- End-to-end user flows

Team Members & Roles

Member	Role
Sam Smith	Full-stack development, authentication implementation, LLM integration, testing
James Tann	Full-stack development, authentication implementation, LLM integration, testing

---

License

Final Notes

TigerTix demonstrates:

- Microservice architecture
- Secure authentication

- LLM-driven interaction
- Distributed testing strategy
- Accessibility-first front-end design