Sprint 2 Report

Team Name: Schizos Resurrection
Team Members: Rodrigo Villalobos, Ronak Bhattal and David Misyuk

Testing Strategy

TigerTix was tested through **unit, integration, end-to-end, accessibility, and concurrency** methods across all microservices and the frontend.

- **Admin Service:** Event creation, validation, and date/ticket rules.
- **Client Service:** Event retrieval, ticket purchase, transaction rollback on failure.
- **LLM Service:** Natural-language parsing, intent detection, JSON validation, booking confirmation.
- **Frontend:** Event rendering, LLM integration, voice recognition, narrator accessibility, error handling.

Automated Testing

Frontend (Jest + RTL):

- Loads events from backend; displays cards correctly.
- Buy/Sold-Out button logic.
- Mocks axios for LLM responses.
- Mocks SpeechRecognition + speechSynthesis for voice and narrator.
- Accessibility: ARIA labels, semantic roles, keyboard focus.

Backend (Jest + Supertest):

- Admin: valid/invalid event creation and validation errors.
- Client: retrieve events, purchase tickets, handle sold-out cases.
- LLM: parses booking requests, handles missing data, confirms bookings.
- Concurrency tests prevent multiple users from overselling tickets.

End-to-End (Selenium):

Simulated full user flow:
Open site → open AI assistant → send natural-language booking → confirm → verify UI & DB ticket decrement.

Manual Testing

- Natural-language booking (text + voice)
- Narrator (speech synthesis)
- Keyboard-only navigation
- Screen-reader labels
- Sold-out purchase behavior
- LLM event/date lookup
- UI updates after backend changes
- Race conditions with simultaneous purchases

All manual tests passed successfully.

Results & Coverage

All areas tested:

- Admin + Client microservices

- LLM-driven booking
- Voice interface
- Accessibility
- Database transactions & concurrency
- UI integration + E2E flow

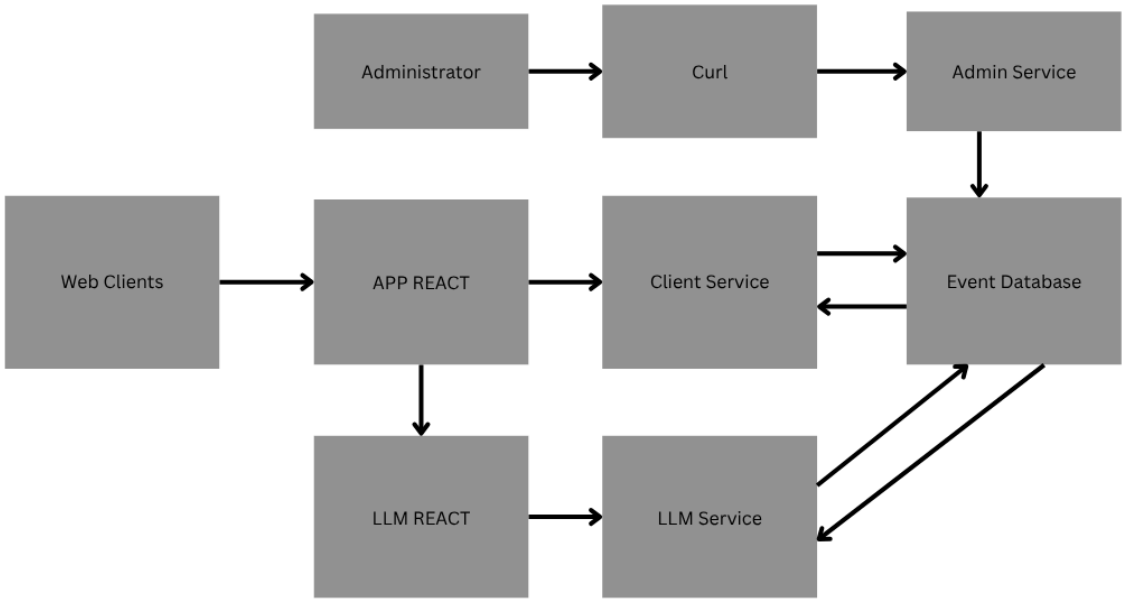
Final Assessment

TigerTix demonstrates complete coverage across backend logic, frontend UI, natural-language and voice features, and database concurrency behavior with no known bugs or defects.

Architecture

Web users use the **React app**, which communicates with the **Client microservice** for event data and the **LLM React app** which communicates with the **LLM microservice** for natural-language booking. The **Client microservice** and the **LLM microservice** connect to a shared database.

Admins use **Postman/cURL** to access the **Admin microservice**, which updates the shared database.



Accessibility Features of the LLM

Voice Interaction

The LLM assistant supports **voice input** through the browser's SpeechRecognition API. This allows users to:

- Speak natural language commands, such as "Book 2 tickets for Tiger Football Game."
- Avoid reliance on keyboard input, which improves accessibility for users with mobility impairments.
- Receive real-time feedback while speaking, with visual indicators displaying [Listening...] to confirm the assistant is active.

Narrator Feature

The LLM includes a **narrator toggle** that uses the browser's speech synthesis capabilities to read AI responses aloud. Key benefits:

- Supports **visually impaired users** by vocalizing ticket details and AI guidance.
- Can be turned on or off according to user preference.
- Integrates seamlessly with the text-based chat interface, ensuring consistent user experience.

Semantic Structure and ARIA

Accessibility is reinforced through **semantic HTML and ARIA labels**:

- Buttons have **descriptive ARIA labels**, e.g., "Start voice input" or "Buy ticket for Tiger Football Game."
- The chatbot panel and main interactive elements are **properly labeled**, ensuring screen readers can navigate the interface efficiently.
- Focus management allows keyboard users to interact with the chatbot without losing context.

UI Considerations

- **Visual indicators** complement voice features to confirm system state.
- **Error messages** are displayed clearly and read aloud when the narrator is enabled.
- Users can fully interact with the AI assistant using **voice, keyboard, or a combination**, accommodating diverse accessibility needs.

AI Chatbot feature

- **User Choice:** The AI can be toggled on by the user by clicking the lower right corner
- **Booking tickets:** The AI can book tickets in advance for a customer, and prompts the user to confirm. The database is updated to display any tickets purchased.
- **Narrator choice:** Any disabled users can toggle on a narrator feature where the AI will read back all messages
- **Event and date information:** The AI can inform the user on any events scheduled on specific dates, or inform the user on certain categories of events i.e (basketball games)

Code Quality and Standards

Function Documentation and Comments

- Function headers and in-line comments were implemented using JSDoc style in all associated components, describing the purpose, inputs, and outputs of each function.

Variable Naming and Code Readability

- All variables are clearly named to reflect purpose and lines of code are kept under 100 characters where possible. Indentation and spacing are consistent throughout code improving readability.

Modularization

- Code is modular and reusable. The LLM , voice recognition and testing was done following our already decided MVC pattern.

Error Handling

- Error handling is implemented in network requests, speech recognition, and ticket purchase logic. Try/Catch blocks are used to handle fetch failures and AI response errors. Speech recognition errors are logged and handled.
  Ex. lines 63 to 68 in llmControler.js, lines 143 to 173 in llm.js, etc

Consistent Formatting

- Indentation is consistent within all files, blanks are used to separate functions and similar statements are logically grouped.

Input and Output Handling

- Inputs and outputs are clearly defined where they are expected. Event data and AI responses are validated. Async operations and API calls are handled with async/await to ensure proper sequencing and error handling
  Ex. line 173 in llm.js, line 67, 8, 96 in llmController, etc