Technical Plan

1. Overview of Technology Stack

Front-End:

- **Framework**: **React.js**
    - To build a web-based dashboard that displays real-time events.
- **Real-Time Data Handling**: **WebSocket**
    - WebSocket will push real-time event data from the back-end to the front-end.
- **UI Components**: **Material-UI**
    - Provides a responsive and modern design framework for the user interface.
- **Data Visualization**: **Chart.js** or **D3.js**
    - To render real-time graphs and charts based on event data.

Back-End:

- **Framework**: **Spring Boot (Java)**
    - Spring Boot will be used to develop the REST API and WebSocket endpoints to receive and process real-time event data.
- **Database**: **MySQL**
    - MySQL will store event data. It's known for its robustness and is aligned with your learning objectives. MySQL will handle both real-time and historical data storage.
- **ORM**: **Hibernate (JPA)**
    - Hibernate will be used for object-relational mapping, allowing easy interaction between Java objects and the MySQL database.

Deployment & CI/CD:

- **Containerization**: **Docker**
    - Docker will be used to containerize the back-end, front-end, and database for consistency and easier deployment.
- **CI/CD Pipeline**: **GitLab CI/CD**

    - CI/CD pipelines will automate the testing, building, and deployment processes.
- **Optional Cloud Deployment**: **AWS (Stretch Goal)**
    - If the stretch goal is pursued, Dockerized services can be deployed on AWS for scalability.

2. System Components

Back-End Components:

1. **Event Receiver (API and WebSocket)**:
    - This API will receive event data via WebSocket or REST API endpoints. The events will be validated according to the provided schema and processed accordingly.
2. **Event Processor**:
    - The back-end will process and validate the event data before storing it in MySQL for real-time and historical access.
3. **Database Integration (MySQL)**:
    - Hibernate (JPA) will abstract database interactions, allowing for seamless data persistence in MySQL.

Front-End Components:

1. **Real-Time Dashboard**:
    - Built using React.js, the front-end will allow users to monitor real-time events as they are received from the back-end via WebSocket.
2. **Visualisation**:
    - Dynamic charts and graphs will be generated using Chart.js or D3.js to visualise the event data.

Deployment & Infrastructure:

- **Docker**:
    - Both the front-end and back-end will be containerized using Docker, simplifying deployment across different environments.
- **CI/CD Pipeline**:

- A CI/CD pipeline will be established to automate building, testing, and deploying the system, ensuring a smooth development process.

3. CI/CD Pipeline and Testing Strategy

To maintain code quality and ensure consistent functionality, each code update will undergo structured testing in the CI/CD pipeline, enhancing reliability and stakeholder trust in the system's integrity.

Detailed CI/CD Testing Stages:

- **Stage 1: Unit Testing**:
  - **Purpose**: Validate individual code units, such as functions and components.
  - **Tools**: **JUnit** for back-end services (Spring Boot) and **Jest** for front-end components (React).
  - **Automation**: Automatically runs on each code commit within the CI pipeline.
  - **Goal**: Target 70-80% coverage, especially for critical components, providing immediate feedback on isolated code.
- **Stage 2: Integration Testing**:
  - **Purpose**: Confirm interaction between components, such as API endpoints, database, and WebSocket layer.
  - **Tools**: **Spring Boot Test**, **Postman** for API calls, **Mockito** for dependency mocking.
  - **Automation**: Integrated into the CI pipeline, with selective manual checks for complex scenarios.
  - **Scope**: Covers data flow, API communication, and database operations to ensure cohesive system behavior.
- **Stage 3: End-to-End (E2E) Testing**:
  - **Purpose**: Validate entire user flows, simulating real user interactions from front-end to back-end.
  - **Tools**: **Cypress** or **Selenium** for browser-based E2E testing.
  - **Automation**: Primarily automated for reliability; runs with major updates to detect regression issues.
  - **Scope**: Ensures functionality in workflows like real-time event updates, filtering, and report generation.

- **Stage 4: System Testing and Performance Benchmarks**:
    - **Purpose**: Assess overall system performance, including response time, throughput, and concurrency.
    - **Tools**: **JMeter** for load testing, **SonarQube** for code quality checks.
    - **Automation**: Performance benchmarks run periodically, with results logged for review.
    - **Scope**: Measures event throughput (target: 10,000 events/hour), response times (target: under 2 seconds), and concurrent user handling (target: 100 users).
- **Stage 5: Acceptance Testing**:
    - **Purpose**: Confirm that system functionalities meet the project requirements as defined by stakeholders.
    - **Tools**: Manual tests guided by acceptance criteria; feedback sessions with stakeholders.
    - **Automation**: Mostly manual; scheduled review meetings with stakeholders for feedback and validation.
    - **Outcome**: Ensures the final system is aligned with user needs and stakeholder expectations.

Continuous Testing with CI/CD:

- **Implementation**: Each code commit triggers automatic testing stages (unit, integration, and system tests) in the CI pipeline, ensuring ongoing validation.
- **Monitoring**: Logs and dashboards provide visibility into test results, error rates, and performance metrics, helping to maintain code quality throughout development.