# M5
# Logic Legends


## Team members:-

Sarabroop Aulukh (24102220)
Shreya Saxena (41969981)
Raghav Bhagria (65827354)
Lluis Escolano (27196567)


## Video Walkthrough (10%)

**Youtube link: Video**

## (2%) Update the requirement document, including:

***A brief description of the software you are building:***

This event ticketing platform aims to help customers buy tickets and allows event organizers to host and manage events efficiently. The platform will also include a customer service ticket system where customers(customers who buy tickets, and event organizers who host events) can get their concerns addressed,  ensuring a smooth and user-friendly experience for all parties.

***A list of user groups for your software, along with an example scenario for each user group of how they will use the software***

- **Customer**:  Users buying event tickets. They create accounts, browse events, purchase tickets, manage their profile, and contact support if needed

  *Example Scenario*: A user logs in, explores events, purchases tickets for a concert, views purchased tickets in their profile, and updates their profile information.

- **Event Organizer** Users hosting events. They create, edit and manag events, awaiting admin approval for listing on the platform.

  *Example Scenario*: An organizer logs in, creates a new event, awaits admin approval, edits event details, and views pending events in their profile.

- **Admin**: Users with backend access for platform management. They approve or reject events for listing.

  *Example Scenario*: An admin logs in, reviews pending events, approves/rejects listings, and manages platform content.

***The final list of requirements of the software that was built. Incorporate feedback from M2.***

These are the user requirements that are done and implemented.

- User Registration and Profile Management

  Functionality: Users can register, create, and manage their profiles on the platform. This includes updating personal information.

- Event Discovery and Search

  Functionality: The platform enables users to discover and search for events.

- Ticket Booking and Management

  Functionality: Users can book tickets for events and manage their bookings directly from their profiles.

- Event Creation and Management for Organizers

  Functionality: Event organizers, who have special permissions, can create and manage their events. Event status is managed through admin approval processes.

- Customer Support and Assistance(**Bonus requirement**)

  Functionality: The platform includes a "Contact Us" feature. This allows users to address and resolve issues through direct communication, facilitated by backend integration with EmailJS for managing inquiries.

These requirements were implemented considering the TA's feedback (provided in person) but had already been discussed before M2. Hence our M2 had very little to no feedback regarding requirements.

Following are the nonfunction requirements we aimed for:

- Performance: Handle 1000 concurrent users, average page load time within 2 seconds.
- Scalability: Support database for 10 thousand events.
- Reliability: System uptime of at least 98%, recover within 15 minutes.
- Availability: Accessible 24/7, compatibility across devices and browsers.
- Security: Encrypt important data.
- Usability: User-friendly interface for all technical levels.
- Maintainability: Well-documented codebase, deploy updates with minimal disruption.
- Compatibility: Compatible with major operating systems, browsers, and devices.

## (70%)Status of the software implementation

***-How many of your initial requirements that your team set out to deliver did you actually deliver (a checklist/table would help to summarize)? Were you able to deliver everything or are there things missing? Did your initial requirements sufficiently capture the details needed for the project?***

## Summary of Initial Requirements vs. Actual Delivery

Our project aimed to develop a comprehensive event ticketing platform, and here's a detailed summary table showing the extent to which initial requirements were met, including any changes made during the development process:

| Requirement | Planned | Delivered | Notes |
|---|---|---|---|
| User Authentication | Yes | Yes | Secure login implemented for users and admins |
| Differentiation Between Roles | Yes | Yes | Distinct pathways and permissions for users and event organizers |
| Event Listing and Management | Yes | Yes | Comprehensive management capabilities for admins |
| Shopping Cart and Checkout System | Yes | Yes | Real-time updates and local storage for user sessions |

| | | | |
|---|---|---|---|
| Dynamic Navigation Bar | Yes | Yes | Adjusts based on user role and authentication status |
| Detailed Event Pages | Yes | Yes | Included event details with options to buy or add to cart |
| Customer Support Agent Interaction | Yes | No | Replaced with a 'Contact Us' page using EmailJS |
| Contact Us Page( Bonus requirement) | No | Yes | Added to facilitate direct user inquiries |
| Admin Login and Dashboard | Yes | Yes | Special access for admins to manage platform and events |
| Status Tracking for Admin Approvals | Yes | Yes | Enabled admins to track and update event status |
| Edit Events | Yes | Yes | Provided functionality for organizers to modify event details |
| View and Edit User Profiles | Yes | Yes | Users can view and update their profile details |

Status on Nonfunctional requirements:

- Performance: The website has minimal load time and is responsive to user interaction.
- Scalability: We are using MySQL as backed so we have the ability to support well over 10 thousand events.
- Reliability and Availability: Since we were not able to deploy our project yet so these requirements could not be met.
- Security: We are encrypting important data like passwords using bcrypt.

- Usability: The website has a user-friendly interface for all technical levels.
- Compatibility: The website is compatible with major operating systems, browsers, and devices

## Reflection on Project Delivery

Our team successfully delivered on all the planned initial requirements, with a notable adaptation involving the customer support mechanism. The original plan included a direct interaction feature through a customer support agent. However, this was strategically replaced with a more streamlined and manageable 'Contact Us' page, leveraging EmailJS to handle user queries directly. This adjustment was made to simplify the support process while still providing an effective channel for user communication.

## Evaluation of Requirement Sufficiency

**1. User Authentication**
Planned and Delivered: Yes
Details: Implemented secure login processes for both regular users and admins. Authentication mechanisms ensure that user data is protected and that access is granted according to user roles.

**2. Differentiation Between Roles**
Planned and Delivered: Yes
Details: Successfully created distinct pathways and permissions for different user types, including customers, event organizers, and admins.

**3. Event Listing and Management**
Planned and Delivered: Yes
Details: Provided comprehensive tools for admins to manage event listings, including approvals and rejections. Event organizers have the ability to create their events, which after admin approval, appear to the public.

**4. Shopping Cart and Checkout System**
Planned and Delivered: Yes
Details: Implemented a real-time shopping cart and checkout system that updates dynamically. Utilizes local storage to enhance user experience by reducing server requests and speeding up load times.

**5. Dynamic Navigation Bar**
Planned and Delivered: Yes
Details: The navigation bar adjusts based on the user’s authentication status and role, providing a tailored interface that facilitates easy navigation through the platform's various features.

**6. Detailed Event Pages**
Planned and Delivered: Yes
Details: Each event page includes detailed descriptions, pricing options, and the ability to directly add tickets to the shopping cart or proceed to immediate purchase. These pages are crucial for user decision-making and enhance the overall user experience.

**7. Customer Support Agent Interaction()**
Planned: Yes
Delivered: No
Details: Originally planned to include a direct interaction feature with customer support agents. However, this was replaced by a more manageable 'Contact Us' page using EmailJS, facilitating efficient communication without real-time agent interaction.

**8. Contact Us Page( as a bonus requirement )**
Planned: No
Delivered: Yes
Details: Added as an enhancement to address direct user inquiries effectively. This feature serves as a critical support tool for users, enabling them to communicate any issues or questions directly to the platform administrators.

**9. Admin Login and Dashboard**
Planned and Delivered: Yes
Details: Special login and dashboard for admins, providing them with the tools needed to manage the platform, approve or reject events.

**10. Status Tracking for Admin Approvals**
Planned and Delivered: Yes
Details: Implemented a status tracking system that allows admins to efficiently manage and update the status of event listings, ensuring that the event management process is transparent and efficient.

**11. Edit Events**
Planned and Delivered: Yes
Details: Event organizers can modify details of their events. This functionality is essential for keeping event information up to date and allows for flexibility in event management.

**12. View and Edit User Profiles**
Planned and Delivered: Yes

Details: Users can view and edit their profiles, which is essential for maintaining accurate user information and personalization of the user experience on the platform.

The initial requirements were sufficiently detailed and structured to guide the project effectively. They encompassed the essential functionalities needed to build a robust event ticketing platform, such as user authentication, role differentiation, event management, and administrative features. These requirements not only facilitated a focused development process but also allowed for adaptive changes without compromising the project's goals or functionality.

The requirements sufficiently captured the necessary details for system architecture and user interaction, evidenced by the successful implementation of core features and the seamless integration of additional functionalities like the 'Contact Us' page. The ability to differentiate between user roles and provide tailored experiences for general users, event organizers, and admins reflects the comprehensive planning and foresight in the project's initial phase.

## Conclusion

Overall, the project was executed with a high level of adherence to the planned requirements, with proactive adjustments made in response to practical considerations. The decision to modify the support component from a customer support agent to a contact form was a testament to our team's flexibility and responsiveness to project needs. This successful delivery and adaptive project management underscore our team's ability to navigate complex development landscapes and deliver a functional, user-friendly event ticketing platform.

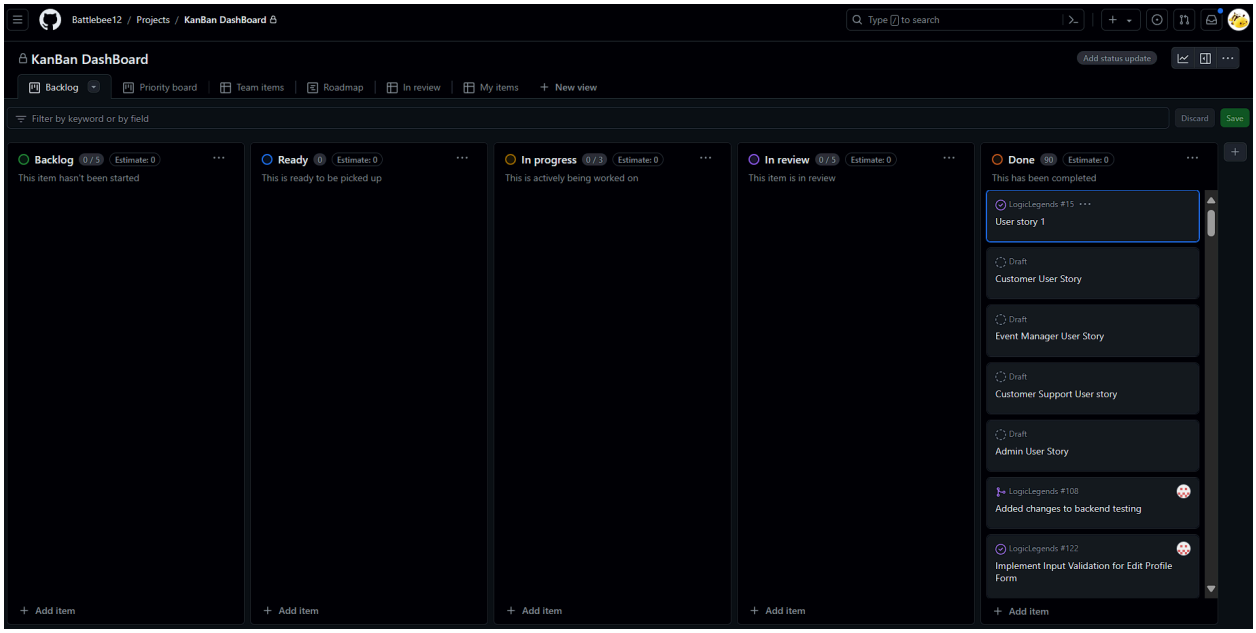
***-If you have any requirements unimplemented, you will want to include an extra section showing those requirements have not been implemented. -***

Instead of the live agent support system, we introduced a 'Contact Us' page. This decision was made to better align with the available resources and to simplify the support mechanism within the platform. A Contact Us page simplifies the process of reaching out for help, making it accessible at any time without the need for immediate human presence. It allows users to describe their issues in detail, which can be addressed systematically.

***-If you have a requirement that is partially working but feel that there's enough you don't want to call the whole requirement as incomplete, then break it up into two requirements, with one shown as tested and working, while the other is shown as not working.*** - None.

***-How many tasks are left in the backlog?***

The project backlog is empty. There is an attached image of the Kanban Dashboard.




In the context of backend testing for our event ticketing platform, we have conducted a total of four tests to evaluate various functionalities. Currently, two of these tests have successfully passed, confirming that specific backend operations are performing as expected. The remaining two tests are still under review to address and resolve issues that were identified during testing. This approach ensures that we continue refining our backend processes, even though these tests are not classified as minimum requirements for the project.

***-What is the architecture of the system? What are the key components? What design patterns did you use in the implementation? This needs to be sufficiently detailed so that the reader will have an understanding of what was built and what components were used/created. You will need to reflect on what you planned to build vs what you built.***

# System Architecture and Key Components

## Architecture Overview

The architecture of our event ticketing platform leverages a modern web application framework, utilizing React for the front-end, Node.js for the backend, and SQL for database management. This architecture provides a robust, scalable, and flexible system capable of efficiently handling dynamic content updates and user interactions.

## Front-End

- React: Used for building the user interface with components managing their own state, enabling complex UIs that are easy to debug due to React's declarative nature.
- Tailwind: Used to beautify the look and appearance of the website.
- React Router: Manages navigation between different parts of the application, such as viewing event details, managing cart operations, and handling user authentication.
- Local Storage: Utilized for storing user and cart data locally to reduce server requests, enhancing user experience by making the interface responsive.

## Back-End

- Node.js: Provides the runtime environment for our server-side logic, used to create and manage server requests, handle business logic, and interact with our SQL database.
- Express.js: A web application framework for Node.js, designed for building web applications and APIs, it aids in routing and middleware functionality.
- SQL Database: Manages all data storage for the application, including user data, event details, and transaction records, allowing for efficient data retrieval and storage operations.

## Key Components

- Event Details Component: Manages the display of individual events, allowing users to view details, select quantities, and either add to cart or buy now. This component fetches event data from the backend based on the event ID.
- Login and Admin Components: Handles user authentication and administrative actions such as approving or rejecting event listings.
- Cart Component: Manages shopping cart operations, including adding and removing items, and calculates total cost dynamically.
- Navbar Component: Provides navigation links and reflects user authentication status dynamically, adjusting displayed options based on the user’s role (organizer or attendee).

# Design Patterns Used in Implementation

**Model-View-Controller (MVC) Pattern Implementation:**

The MVC pattern serves as the foundational architectural design pattern for our online ticketing application, providing a structured approach to separating concerns and enhancing maintainability and scalability. Here's how we've implemented it:

**1. Model:**

- The Model component represents the data and business logic of the application.

- In our ticketing app, the Model encompasses functionalities related to user information, event details, transaction data, and backend operations.
- We have a dedicated backend folder containing the Model part of our application. This folder houses all backend functionalities, including backend calls, data fetching and updating, and convenient functions to manage backend operations.

- Additionally, our backend folder includes the Data Definition Language (DDL) file, which defines the database schema and structure.

**2. View:**

- The View component is responsible for presenting the user interface and interacting with users.
- In our ticketing app, the View encompasses various components used to render different aspects of the user interface, such as event browsing, seat selection, ticket purchasing, event creation, and management.
- Components are organized into separate folders within our project structure, facilitating modularization and reusability. For example, the eventCat component is utilized on the homepage, while the navbar component can be easily inserted into any page.
- Each component serves as an independent unit that can be seamlessly integrated into different parts of the application as needed.

**3. Controller:**

- The Controller acts as an intermediary between the Model and View components, facilitating communication and managing user interactions.
- In our implementation, the Controller functionality is distributed across various parts of the application, effectively handling user input, processing requests, and updating the Model accordingly.
- While we don't have a distinct Controller component isolated within our project structure, the Controller logic is embedded within different modules and functionalities throughout the application.

- Every module, page, or feature within our application functions as a Controller, orchestrating interactions between the backend (Model) and frontend (View) components.

**Another minor design pattern:**

1. **Observer Pattern**

   We are employing the Observer pattern through state and props, where components re-render in response to state changes. Enhanced our project by using hooks to react to changes in local storage and manage user sessions dynamically.

## Reflection on Planned vs. Actual Build

Besides the table above that goes in depth on the planned vs build features, initially, the plan focused on creating a streamlined ticketing platform with basic functionalities. As the project developed, it expanded to include features such as detailed event pages, user authentication, and a sophisticated cart system, requiring the integration of more complex React features and design patterns to manage state and side effects effectively. These enhancements were not only in response to user feedback but also due to the iterative nature of agile development practices that our team adopted.
This detailed description encapsulates the technical structure and thought process behind your application, highlighting both the initial intentions and the evolutionary adaptations of your project. It provides a comprehensive understanding of the architecture, components, and design patterns, suitable for a detailed project reflection or evaluation.

***-What degree and level of re-use was the team able to achieve and why?***

As we have already explained above we based our project implementation on the MVC design pattern and we also tried to have as many components in our projects that were independent and could be used in other places too. For instance, as we already mentioned above we have a separate folder for components that does this. This is because we wanted to create code that was modular and could be reused in future.

The View component is structured into reusable UI components that render different aspects of the user interface. The navbar component is designed to be reusable across pages, dynamically adjusting to the user's role and authentication status.
Modular Design: The modularization of UI components facilitated their reuse across different views without the need to duplicate code. This not only speeds up development

but also reduces errors and improves maintainability. The reusability of components has made maintenance and upgrades simpler and less error-prone, as changes need to be made in fewer places and can be propagated easily across the application.

***-Indicate any known bugs in the software.***

Event Visibility Issue: When an event organizer creates a new event, it becomes visible to all other event organizers before it has been approved or rejected by an admin. This behavior is unintended as per the initial design, which aims to make events waiting for approval or rejection be visible to only the organizer creating them.

***-Explain how these bugs can be fixed.***

- Adjust the SQL query used to fetch events so that it includes a check both for the approval status and the identity of the organizer. This means the query should only return events that are either approved or belong to the organizer who is logged in.
- Update the frontend to handle the possibility that the list of events might include both approved events and the user's own pending events.
- Verify that access controls are in place to prevent unauthorized access to event data. This involves reviewing and potentially enhancing security measures around user authentication and authorization.

## (15%) A step-by-step guide for handing over the project that includes:

**Installation details needed to deploy the project:**

**Running Eventify Web App**

To run the Eventify web application on your machine, follow these steps:

- Clone Repository

  Clone the Eventify repository to your local machine:

- git clone https://github.com/Battlebee12/LogicLegends.git

**Install Dependencies Navigate to the cloned repository directory in the terminal and run:**

- npm install

**Set Up MySQL Database Open a new terminal window and run MySQL:**

- mysql -u root -p

Enter the password for your MySQL root user when prompted, then execute the SQL commands located in the ddl folder to create the necessary tables. Ensure that the SQL script contains all the required tables as specified.

Configure Backend Navigate to the backend folder in the repository. In the index.js file, update the password field with your MySQL root user password.

**Start Backend Server In the same terminal window, navigate to the backend folder and run:**

- node index.js

**The backend server should now be running.**

- Start Frontend

**Start the frontend of the application by running:**

- npm start

and the website should be ready to be tested

**Dependencies needed to deploy the project**

Following the steps above you would be able to install all the dependencies. Additionally following are the dependencies that are needed for the project:

Any maintenance issues required to run the project (e.g., account information)

If you need to login as an admin then you need the following email and password:

- testuser@test.com
- testuser

## (3%)Reflections (Comment on the following items as a team):

### -How did your project management work for the team? What was the hardest thing, and what would you do the same/differently the next time you plan to complete a project like this?

Our project management strategy proved highly effective, characterized by regularly scheduled meetings and robust participation from all team members. This structured approach facilitated clear communication and timely progress updates. The most challenging aspect was dealing with technical issues, such as hardware failures and software glitches on individual team members' devices, which were unforeseen and caused delays. To enhance our future project outcomes, we plan to begin the project setup much earlier than previously done. Starting earlier will allow us more flexibility to address these technical challenges and ensure that all team members are adequately prepared with the necessary tools and environment setups before diving deep into development tasks.

### -Do you feel that your initial requirements were sufficiently detailed for this project? Which requirements did you miss or overlook?

Our project started with a well-detailed set of initial requirements, which guided the early stages of the platform's development effectively. As the project progressed, we encountered additional functional and non-functional requirements that were initially unaccounted for, such as advanced security features and user experience enhancements. This realization highlighted the dynamic nature of software development where initial plans often evolve. While we managed to expand our scope to incorporate these new insights, a lesson learned was to maintain a more adaptive approach to requirement gathering from the outset, possibly through iterative feedback loops with potential users.

### -What did you miss in your initial planning for the project (beyond just the requirements)?

In hindsight, our initial planning was heavily skewed towards the development aspects without adequate attention to execution and deployment strategies. This oversight led to some inefficiencies in how we managed project resources and timelines. Going forward, we intend to

place greater emphasis on the execution phase during our planning process. This will include detailed discussions on deployment strategies, resource allocation, and risk management plans to ensure smoother transitions from development to production.

**- Would you (as a team) deal with testing differently in the future?**

Our approach to testing started later than ideal, which in retrospect, contributed to a condensed and stressful final phase of the project. Recognizing the importance of early and continuous testing, we would revise our strategy to incorporate testing phases concurrently with development phases. This adjustment would help identify and resolve issues earlier in the process, potentially saving time and reducing last-minute debugging efforts. Adopting a test-driven development (TDD) approach could also be considered to ensure that new features are robust and well-integrated from the start.

**-If you were to estimate the efforts required for this project again, what would you consider? (Really I am asking the team to reflect on the difference between what you thought it would take to complete the project vs what it actually took to deliver it).**

Reflecting on our effort estimations, the project demanded considerably more time and resources than initially anticipated. Original estimates did not account for the additional requirements identified during the development process nor the time required for comprehensive testing and revisions based on feedback from our teaching assistants. This underestimation led to an average workload of about 8-10 hours per person per week, which was higher than expected. For future projects, we would consider including a more substantial contingency factor when estimating efforts and also engage in more thorough upfront planning to better anticipate the demands of the project.

**-What did your team do that you feel is unique or something that the team is especially proud of (was there a big learning moment that the team had in terms of gaining knowledge of a new concept/process that was implemented).**

One of the unique aspects of our project was the rapid acquisition and application of new technologies, specifically React and Node.js, which none of the team members had substantial experience with prior to this project. The ability to learn and effectively implement these technologies under project constraints is something we are particularly proud of. This experience has not only enhanced our technical capabilities but also fostered a resilient and adaptive team dynamic that can confidently tackle new challenges. Additionally, the successful collaboration

and problem-solving exhibited by our team in learning these new technologies from scratch and applying them effectively was a significant accomplishment.