GETTICK

ARCHITECTURAL MODEL PLAN

05190000081 - Safiye Defne Kargı
05200000034 - Muhammet Şancı
05190000104 - Hazal Karataş
05200000761 - Berat Duran

INTRODUCTION

In the dynamic and evolving landscape of event planning and management, the need for a robust, scalable, and user friendly digital system is paramount. Our project, aimed at revolutionizing the way events are organized, managed, and experienced, leverages cutting-edge technology to offer an unparalleled solution.

This document presents the architectural model of our state-of-the-art event planning and ticketing system. We have adopted the Model-View-Controller (MVC) architecture, a proven pattern that ensures a clean separation of concerns, enhanced scalability, and ease of maintenance.

This architecture is tailored to meet the intricate demands of event management, including event creation, ticketing, attendee management, and real-time analytics. The ollowing sections will detail the architecture, explaining its components, their interrelationships, and the rationale behind our architectural choices.

TABLE OF CONTENTS

PROJECT ARCHITECTURE

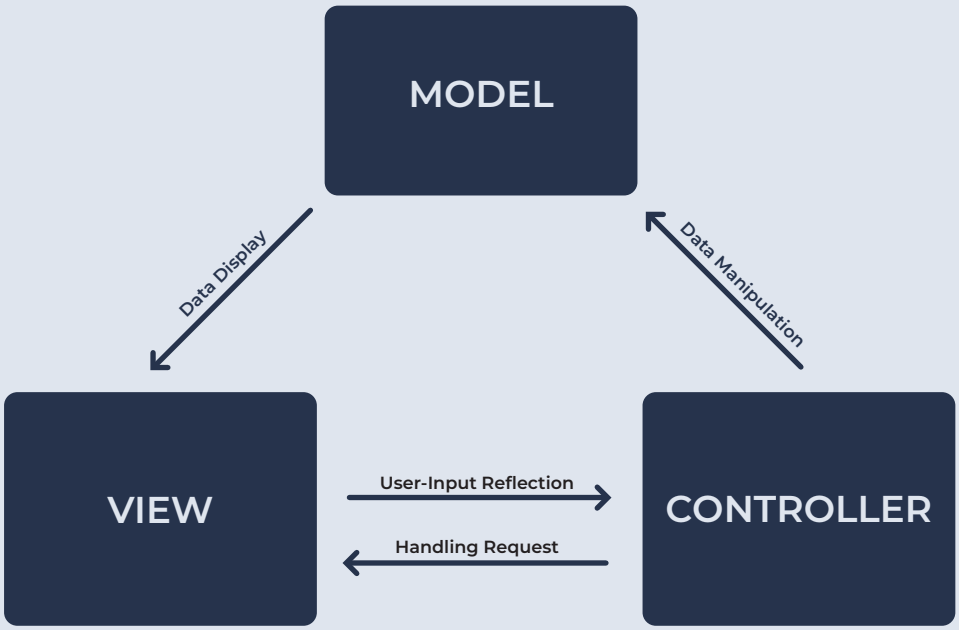
WHY MODEL-CONTROLLER-VIEW ARCHITECTURE?

Separation of Concerns:
MVC allows for a clear separation of concerns - **the Model handles data and business logic, the View manages the user interface and user interactions, and the Controller bridges the two, handling user input and system output.** This separation is ideal for your system as it deals with complex functionalities like event management, ticket purchasing, and attendee tracking.

Flexibility and Scalability:
MVC architecture supports flexibility and scalability, which are essential for adapting to various event types and sizes, and handling large volumes of ticket transactions and user interactions.

Ease of Maintenance:
With MVC, updates and maintenance can be done more efficiently, as changes in one component (e.g. the user interface) do not require changes in the logic or data layers.

STRUCTURE OF THE ARCHITECTURE



PROJECT ARCHITECTURE

MAIN COMPONENTS & SUBSYSTEMS

MODEL

M1 Account Management

Manages user accounts, including registration, login, profile management, and security features. It's essential for personalizing user experience and securing user data.

M2 Ticketing System

Manages ticket inventory, including ticket creation, allocation, and tracking. It's crucial for handling the availability and distribution of tickets for events.

M3 Event Data

Handles the storage, retrieval, and management of event-related information such as event details, schedules, and locations. Backbone for organizing and displaying event information.

M4 Payment Processing

Facilitates financial transactions for ticket purchases. It includes integration with payment gateways, secure processing of payments, and transaction logging.

VIEW

V1 Event Listing Interface

The user interface element where events are displayed. It allows users to browse and select from available events.

V2 Ticket Purchase Interface

The user interface for the ticket buying process. It includes selection of seats, viewing pricing, and proceeding to payment.

V3 Attendee Registration Interface

The interface where attendees register for events, providing necessary information and preferences.

V4 Real-Time Update Display

Shows up-to-date information on events, such as ticket availability or schedule changes, enhancing the user experience with current data.

PROJECT ARCHITECTURE

MAIN COMPONENTS & SUBSYSTEMS - CONT.

CONTROLLER

C1 Input Validation

Ensures the correctness and security of user input, preventing errors and security vulnerabilities in processes like ticket purchasing or account creation.

C2 Event Create-Edit Controls

Tools and functionalities that allow event organizers to create, modify, and manage event details, ensuring accurate and up-to-date event information.

C3 Ticket Purchase Controls

Handles the logic and workflows involved in purchasing tickets, from seat selection to payment confirmation.

C4 Attendee Management Controls

Includes functionalities for managing attendee data, tracking registration, and check-ins, and generating reports on attendee engagement and behavior.

RELATIONSHIPS BETWEEN THE COMPONENTS

Model-View Interaction:

- **Data Display:** The View requests information from the Model to display current data to the user (e.g., available events, seating arrangements).

View-Controller Interaction:

- **User Input Reflection:** When a user makes changes (like buying a ticket), the View sends these inputs to the Controller.
- **Handling Requests:** The Controller receives user input from the View, interprets it, and determines the necessary operations.
- **Updating View:** After processing input, the Controller updates the View with new or modified data.

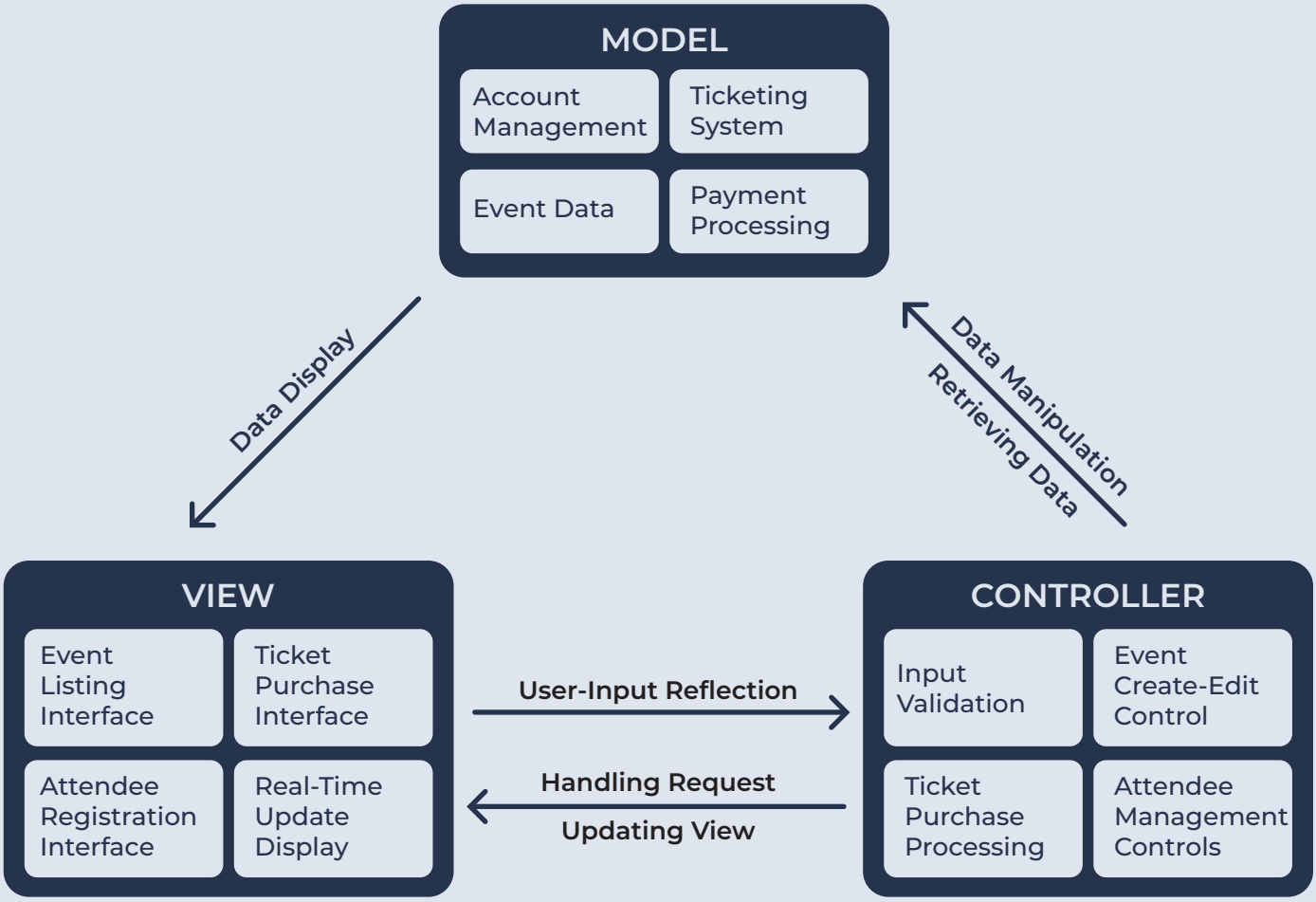
Controller-Model Interaction:

- **Data Manipulation:** The Controller updates the Model based on user input (e.g., adding a new event, processing a purchase).
- **Retrieving Data:** The Controller retrieves data from the Model to provide the View with the information needed to display to the user.

BLOCK DIAGRAM

Architectural block diagram of the project is as follows.

Explanation of every component, subsytem and interaction is told earlier at the previous pages.



PREFERENCES

Why We Didn't Choose Layered Architecture?

- **Tight Coupling Between Layers:** In a layered architecture, each layer directly interacts only with the layer directly below it. This can lead to tight coupling and a lack of flexibility, especially if the application needs more dynamic interaction between various components.

- **Performance Concerns:** Layered architectures can sometimes introduce latency, as requests must pass through multiple layers. For a system requiring real-time updates, like ticket availability and analytics, this could be a hindrance.

- **Less Suitable for Rich User Interfaces:** Layered architecture is often server-centric and might not offer the best structure for developing complex, interactive user interfaces that are crucial for your event management platform.

Why We Didn't Choose Repository Architecture?

- **Centralized Data Management Focus:** The repository architecture emphasizes centralized data management, which can be limiting for applications requiring distributed processing and flexibility in data handling.

- **Potential Bottlenecks:** Having a single repository can become a bottleneck, especially in systems with heavy data traffic and numerous transactions, such as ticket sales and attendee management.

- **Less Emphasis on User Interface:** This architecture is more data-centric and might not provide the best framework for developing a highly interactive and user-friendly interface.

PREFERENCES

Why We Didn't Choose Client-Server Architecture?

- **Scalability Concerns:** While client-server architecture is suitable for distributed systems, it can face scalability issues under heavy load, which is a common scenario in event ticketing systems, especially during peak sale times.

- **Less Flexible for Rapid Changes:** In a dynamic event management scenario, where changes are frequent, the client-server model might not be as agile as MVC in deploying rapid updates across both client and server components.

- **Security Risks:** This architecture can sometimes pose additional security challenges, especially in a context where sensitive financial transactions are involved.

Why We Didn't Choose Pipe & Filter Architecture?

- **Centralized Data Management Focus:** The repository architecture emphasizes centralized data management, which can be limiting for applications requiring distributed processing and flexibility in data handling.

- **Potential Bottlenecks:** Having a single repository can become a bottleneck, especially in systems with heavy data traffic and numerous transactions, such as ticket sales and attendee management.

- **Less Emphasis on User Interface:** This architecture is more data-centric and might not provide the best framework for developing a highly interactive and user-friendly interface.

CONCLUSION

In conclusion, the architectural model presented for our event planning and ticketing system embodies a harmonious blend of functionality, user experience, and technological innovation. By choosing the MVC architecture, we have laid a solid foundation that supports flexibility, scalability, and ease of maintenance.

This architecture not only addresses the current needs of event management but is also versatile enough to adapt to future advancements and changes in the industry. The careful consideration in not choosing alternative architectures further reinforces our commitment to providing a system that is both robust and agile. We believe that this architectural blueprint will pave the way for a new era in event management, offering users a seamless, efficient, and enjoyable experience.

GETTICK

NEW ERA OF
TICKET & EVENT
MANAGEMENT