TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

SAGARMATHA COLLEGE OF SCIENCE AND TECHNOLOGY

A

PROJECT REPORT

ON

FREELANCE ARTIST BOOKING SYSTEM

(KalaConnect)

BY

GREETIKA BAJRACHARYA(28954/078)

KRISHMA MAHARJAN(28958/078)

MANSHI KANDU(28960/078)

In partial fulfillment of the requirements for the degree of bachelor of science in computer science and information technology awarded by tribhuvan university.

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

SANEPA, LALITPUR, NEPAL

September, 2025

SUPERVISOR RECOMMENDATION

I hereby recommend that this project prepared under my supervision by **Greetika Bajracharya, Krishma Maharjan** and **Manshi Kandu** entitled **"Freelance Artist Booking System"** in partial fulfillment of the requirements for the degree of Bachelor of Science (B.Sc.) in Computer Science and Information and Technology be processed for the evaluation.

.....

Mr. Manish Aryal

Project Supervisor

Department of Computer Science and Information Technology

Sagarmatha College of Science and Technology

APPROVAL LETTER

The project work entitled **"Freelance Artist Booking System"**, submitted by **Greetika Bajracharya, Krishma Maharjan, Manshi Kandu** in partial fulfillment of the requirement for the award of the degree of **"Bachelor of Computer Science and Information Technology"** has been accepted as a bonafide record of work independently carried out by team in the department.

.....

Mr. Manish Aryal

Supervisor

Sagarmatha College of Science and Technology

.....

Mr. Bishnu Khadka

Head of Department

Sagarmatha College of Science and Technology

.....

Mr. Binod Kumar Adhikari

External Examiner

IOST, Tribhuvan University

STUDENT'S DECLARATION

We hereby declare that the project report entitled **"Freelance Artist Booking System"** which is being submitted to the Sagarmatha College of Science and Technology, IOST, Tribhuvan University, in the partial fulfillment of the requirements for the Bachelor's Degree in Computer Science and Information Technology, is our original work. The material contained in this report has not been submitted to any University or Institution for the award of any degree.

.....

Greetika Bajracharya 28954/078

.....

Krishma Maharjan 28958/078

.....

Manshi Kandu 28960/078

ACKNOWLEDGEMENT

We would like to express our deepest appreciation to all those who provided us with the possibility to complete this report. We express our profound gratitude and deep regards to our project supervisor **Mr.Manish Aryal** for monitoring us, providing constant encouragement throughout the completion of this project and guiding towards right direction. We are also thankful to our Department head **Mr.Bishnu Khadka** for providing us with a work space where we could easily work in a group. We are obliged to our class friends for their full effort in guiding and supporting us in achieving the goal. We would like to thank all the teaching and non teaching staffs and senior for their support, motivation and encouragement which helped us achieving our goal.

Finally, an honorable mention goes to our families for their understanding and supports on us in completing this project. Without the help of the particular that mentioned above, we would not have been able to complete this project. We would highly appreciate and heartily welcome suggestion for further improvement if any.

Greetika Bajracharya 28954/078

Krishma Maharjan 28958/078

Manshi kandu 28960/078

ABSTRACT

Freelance Artist Booking System aims to make hiring performers easier by connecting event planners with artists through a well-organized, feature-complete platform. The system connects clients with various performers like DJs, dancers, singers, emcees, comedians, and photographers through categorized listings, comprehensive portfolios, and live availability tracking. It employs algorithms like a Recommendation System to suggest suitable artists to users according to their taste, location, as well as prevent double booking and optimize artist schedules. With additional features like location-based suggestions, in-system messaging, contract generation, and secure payment processing, the system addresses key issues in the current informal and decentralized booking process. With the inclusion of review mechanisms, client get oversight, and promotion of new and underappreciated artists, trust, fairness, and opportunity are further enhanced. This solution enables professionalism, efficiency, and inclusiveness within the entertainment and events industry.

Keywords: Freelance Artists, Online Platform, Artist Marketplace, Performer Scheduling

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
DFD	Data Flow Diagram
ER	Entity Relationship
JS	JavaScript
JWT	JSON Web Token
JSON	JavaScript Object Notation
MERN	MongoDB ExpressJS React NodeJS
UI	User Interface
UX	User Experience

CHAPTER 1

INTRODUCTION

1.1 Introduction

The entertainment industry has seen a significant change in recent years towards freelance and gig-performances. Events such as weddings, corporate seminars, restaurant or cafe gigs, local festivals, clubs, and private parties often require entertainers like DJs, musicians, singers, dancers, emcees (presenters), stand-up comedians.The "Freelance Artist Booking System" aim to create a centralized digital platform that seamlessly connects artists and event organizers for efficient bookings. It ensures professionalism through verified profiles, real-time communication, contracts, and secure payment integration.

1.2 Problem Statement

In the current scenario of freelancing services in entertainment field, both event organizers and freelancers are facing challenges due to the absence of centralized and professional booking platform.

The following are the most common problems faced:

For Users / Event Organizers

- **Scattered and Informal Booking Process :** Relying on social media, the referral is unreliable and it lacks the standardization for finding and booking artists.
- **Difficulty in finding right talent :** In the context of Nepal, there is no effective way to filter artists by categories such as musician, dancer, Djs, emcee, photographers etc. and also according to different genre such as Cultural western or modern etc.
- **No Real Time Availability :** No real time availability results in frequent scheduling conflicts, as users or event organizers cannot check the current artists availabil-

ity before booking.

- **No Legal Safeguards :** Absence of formal contracts may leads to misunderstanding and disputes between Artists and event organizers.
- **Unsecure Payments :** Informal payment method can also cause issues such as fraud.

For Freelance Artists

- **Limited Recognition:** The artists visibility and recognition is only limited to local networks.
- **No organized or structured portfolio:** Lack of organized portfolio with videos and history of work.
- **No Legal Security:** Lack of Legal Security may lead to risk of cancellations and create misunderstanding and disputes between Artists and event organizers.
- **Unverified Client Requests and No Legal Security:** Risk of cancellations or exploitative deals.
- **Delayed or unsecure Payments:** There is no assurance of timely and secure payments.

1.3 Objectives

The objectives of this project is:

- To empower local and emerging freelance artists by providing them a platform to include their detailed profiles featuring images, videos, location, experience, genre, and pricing and also to ensure reliable and smooth financial transactions for both clients and artists.

- To enable personalized artist discovery based on user preferences, event types, and geographic proximity and enable real-time communication and also publish detailed posts outlining their event needs, preferred talent type, location, budget, and date. Artists can respond directly to these posts, making the hiring process more targeted and efficient.

1.4 Scope and Limitation

1.4.1 Scope

The scope of this project includes the following:

1. **Artist Registration and Profiles:** Artists can sign up, create profiles, add bios, select categories (e.g., singer, DJ, dancer), and upload media (images/videos) .

2. **Availability Management:** Artists can manage their availability using a calendar.

3. **Artist Search and Filter:** Clients can browse and filter artists by category, location, budget, genre, language.

4. **Booking System:** Clients can send booking requests, and artists can accept or reject them. All event details including date, time, venue, and fees are captured.

1.4.2 Limitation

While the application aims to provide a comprehensive solution for booking system, there are some limitations to consider:

1. **Manual Artist Approval Process:** Artist profile verification and approval require manual review by admins, which can slow down onboarding.

2. **AI Recommendation:** Features such as AI-based artist suggestions and smart search are planned but not implemented in this version.

3. **Limited Advanced Analytics:** The admin panel lacks detailed data analysis tools such as booking trends, artist performance metrics, and client insights.

4. **No Offline Booking Support:** All bookings and interactions must happen online within the platform, with no provision for offline agreements or updates.

1.5 Development Methodology

For the development of the Freelance Artist Booking System, Agile methodology is a modern approach to software development that emphasizes flexibility, collaboration, and rapid delivery of functional software. Unlike traditional models, Agile breaks the development process into smaller, manageable units called iterations or sprints. Each iteration typically lasts from one to four weeks and results in a working product increment. Agile encourages constant communication among team members and stakeholders, allowing for quick feedback and continuous improvement.

- **Planning**

  Each sprint focused on high-priority features with clearly defined goals, timelines, and resources. This approach allowed the team to adjust quickly based on progress and feedback.

- **System Design**

  Initial sketches and prototypes are created for the booking form, artist profiles, dashboard, and calendar workflows. As development progresses, designs evolve based on user testing and stakeholder input.

- **Implementation**

  Features such as artist search, bookings, notifications, and payment integration were developed incrementally. Each component was coded, tested, and integrated in small working pieces to ensure smooth functionality.

- **Testing**

  Each feature is verified as it is developed. Bugs are fixed immediately within the sprint, ensuring the system remains stable and user-ready at all times.

- **Deployment and Maintenance**

  Functional features were deployed incrementally, allowing early user access. The system was continuously monitored and updated with improvements such as advanced filters, notifications, and calendar enhancements.

- **Review and Feedback**

  At the end of each sprint, stakeholders reviewed the delivered functionality. Feedback was added to the backlog to guide future development, ensuring the system evolved according to real user needs.

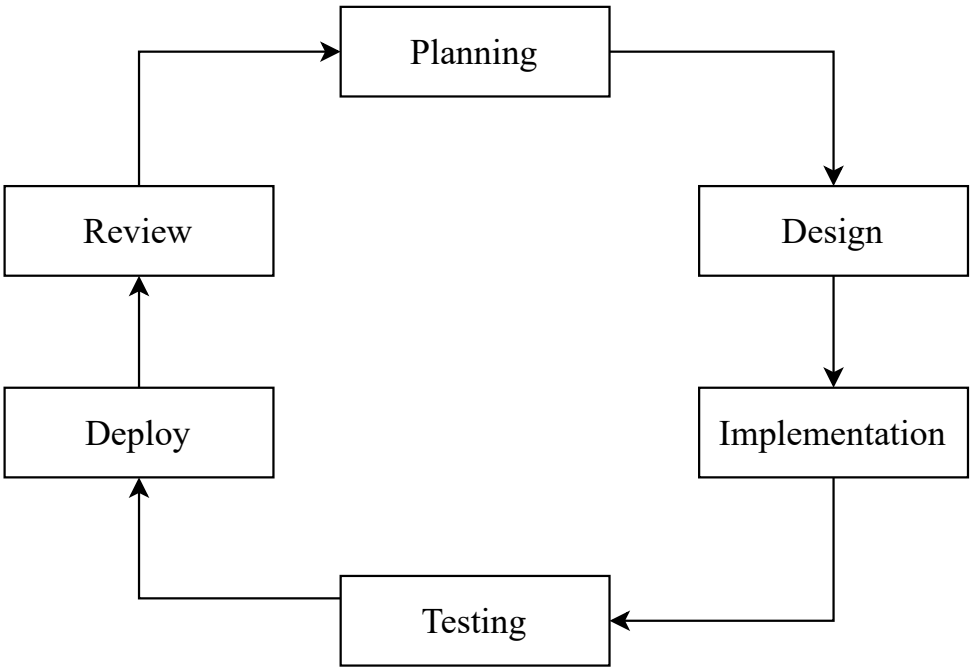

Figure 1.1: Agile methodology

1.6 Report Organization

The report is divided into 6 chapters. Each chapter is further divided into different headings.

- **Chapter 1: Introduction**

  In Chapter 1, we have provided an overview of the project, highlighting its purpose and the problem it addresses. The objectives of the project were clearly defined, and we have outlined the scope and limitations. We also described the development methodology used and gave an overview of the report organization, explaining how each chapter is structured.

- **Chapter 2: Background Study and Literature Review**

  Chapter 2, covers the background and relevant theories, explaining the fundamental concepts and terminologies. We have reviewed related projects and research to establish the context for our work, highlighting existing solutions and identifying gaps that our project aims to fill.

- **Chapter 3: System Analysis**

  In Chapter 3, we have conducted a thorough system analysis, detailing both functional and non-functional requirements. We performed a feasibility analysis from technical, operational, economic, and schedule perspectives. The chapter also includes an analysis of the system using either structured or object-oriented approaches, with data and process modeling.

- **Chapter 4: System Design**

  Chapter 4 focuses on the design of the system. We have presented a procedural structured design, using ER modeling, database design, and DFDs to define data flow, structure, and process logic for efficient and organized system development.

- **Chapter 5: Implementation and Testing**

  In Chapter 5, we have described the tools, programming languages, and platforms used for development. We also provided an overview of the system modules and testing phases, including unit and system testing. The test cases were executed successfully, and we analyzed the results to ensure the system met the required specifications.

- **Chapter 6: Conclusion and Future Recommendations**

  Chapter 6 concludes the project, summarizing the outcomes and achievements. We reflected on the success of the project in meeting its objectives and discussed its potential impact. We also provided recommendations for future work and improvements that could enhance the system further.

CHAPTER 2

BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

The freelance artist industry faces several challenges, including lack of centralized booking systems, inconsistent communication between clients and artists, and limited access to verified portfolios. According to recent industry analyses, many artists struggle to find reliable platforms that ensure secure payments, proper agreements, and professional engagement opportunities. These issues emphasize the need for a dedicated system that can streamline bookings, manage artist profiles, and build trust between parties.

Unorganized scheduling, informal contract handling, and inefficient communication are common trends that disrupt the workflow of freelance artists. Clients often face difficulty verifying an artist's credibility, while artists lack visibility and access to consistent job opportunities. Additionally, with increasing demand for diverse performers at events, the absence of a centralized, structured platform limits potential collaborations and market reach.

2.1.1 Key Concepts and Terminologies

2.1.1.1 User Roles

The system identifies two main user roles: **Artists** and **Clients**. Artists are performers who offer services, while clients are individuals or organizations looking to book performers for events. Both roles require verified registration and profile setup.

2.1.1.2 Booking Request

A booking request is initiated by a client, specifying event details such as date, time, location, performance category, and budget. The system matches available artists based

on their calendar and preferences.

2.1.1.3 Profile and Portfolio

Each artist maintains a digital profile showcasing their bio, past work, media uploads (images, videos), performance categories, reviews, and availability calendar. This builds trust and helps clients make informed decisions.

2.1.1.4 Contract and Payment Integration

The system auto-generates contracts upon mutual confirmation of a booking. Payment integration allows clients to pay securely through the platform, with support for partial or advance deposits.

2.1.1.5 Real-time Communication

A built-in chat feature facilitates smooth interaction between clients and artists before, during, and after booking. This feature reduces delays and misunderstandings, supporting a professional experience.

2.1.2 System Design Approach

The system is developed using a **Procedural structured design** approach. This includes the use of **Entity-Relationship (ER)** diagrams to define entities like Artists, Clients, Bookings, Payments, and Chats. A normalized **database design** ensures efficient data storage and retrieval. **Data Flow Diagrams (DFDs)** are used to visualize how data moves through the system for booking and other processes.

2.1.3 Data Collection

1. **Industry Literature:** Reviewed articles and reports on the freelance and event management industries to understand standard practices, pricing trends, and legal

frameworks for service agreements.

2.2 Literature Review

The Seek N Book web application, designed for seeking gigs and booking performers. The proposed web application in this study enables the users, which is the organizer and the musician, to create a job and communicate with each other. The design shows its effectiveness in creating a platform specifically in connecting organizers and musicians. [2]

The documentation is focused on the design of a freelancing application using modern frameworks. In the thesis, the main objective is to introduce a new mechanism for user booking functionality. The future implementation level includes what kind of suggestion that can be implemented. [1]

Pierre Michel Menger's study provides insights into the dynamics of artistic labor markets and career structures, offering a foundational perspective on how freelance artists navigate job opportunities, uncertainty, and competition in their careers. This supports the relevance of a dedicated booking system tailored for freelance artists. [3]

The purpose of this study also compares the MVC Laravel and Slim framework architecture with a performance comparison method by implementing the MVC (Model View Controller) architecture model with the PHP Laravel and Slim framework. [4]

The impact of technology on talent management emphasizes how emerging digital tools and platforms optimize recruitment, training, and employee engagement. [5]

2.3 Existing System

- **Gigsalad**

  GigSalad is an online platform that connects entertainers and service providers with individuals or organizations looking to hire them for events. GigSalad is headquartered

in Springfield, Missouri, USA. It operates as a marketplace for booking talent and services for a wide range of occasions such as weddings, parties, corporate events, and community functions.

- **System One Software**

  System One Software provides advance booking and scheduling platform which is designed for large scale events. System One Software includes features like artist calendar management and artists performance tracker which is suitable for corporate events planners or venue managers. However, its complexity can be confusing and hard to use for individual artists or small event organizers. System One Software lacks an integrated search and filter function for clients to browse artists by genre, location etc. Additionally, in this system direct communication between artists and event organizers is limited.

- **CueUp.io (Kathmandu DJ Booking)**

  CueUp.io is a Kathmandu based DJ booking platform that makes easy to find and book DJs using location based search. It is simple and useful for just booking for a DJs as CueUp.io doesn't include other artist as it only focuses on providing DJs and this system lacks real-time calendar to check availability and there is no option for legal contracts. Our platform solves all these problems by offering more features and supporting many types of artists or different kinds of event.

CHAPTER 3

SYSTEM ANALYSIS

3.1 System Analysis

To develop a successful and user-friendly freelance artist booking platform, the Freelance Artist Booking System requires a comprehensive analysis of both functional and non-functional requirements. These requirements ensure that the system meets the needs of all users, including artists, clients, and administrators, by supporting smooth booking processes, secure transactions, and reliable communication.

3.1.1 Requirement Analysis

3.1.1.1 Functional Requirements

Functional requirements define the visible and specific behaviour or the act of the systems. The requirements informs us about what the system is essential to do:

User Registration and Profile Management

Artist Profiles

- Categorized Listings such as dancers, singers, emcees, musician etc.
- Comprehensive Portfolio such as gallery, videos, past work samples
- Skills, Specializations and Event Types in weddings, corporate events, etc.
- Pricing based on hourly, per-event, custom quotes etc.

Client Profiles

- Booking History such as past hires, booking details.
- Event Details such as date, type, location, budget etc.

Searching and Filtering Artists

- Advanced Filters by using categories, location, price range and ratings.
- Location based suggestion using nearby artists for local events.
- Recommendation System

Booking System

- Real-Time artists check
- Instant Booking and Request Approval with automated confirmations.
- Rescheduling and Cancellation Policies with penalties if applicable.

Payment Integration

- Integrated Payment Gateway such as Paypal, etc.
- Contract Generation
- Artists earning Dashboard

Admin Dashboard

- Admin Dashboard for user management, fraud detection, analytics
- Content Moderation for fake profile/review prevention
- Support for Disputes and Refunds

Reviews and Trust Building

- Post-Event Ratings and Reviews for clients.
- Verified Artist Badges for background checks and portfolio validation.

- Recommendation of Underappreciated Artists.

Use Case Diagram

A Use Case Diagram is a visual representation that depicts the interactions between users (actors) and a system. It outlines the functional requirements of a system by showing various use cases, which are the actions or services that the system provides to fulfill user needs.

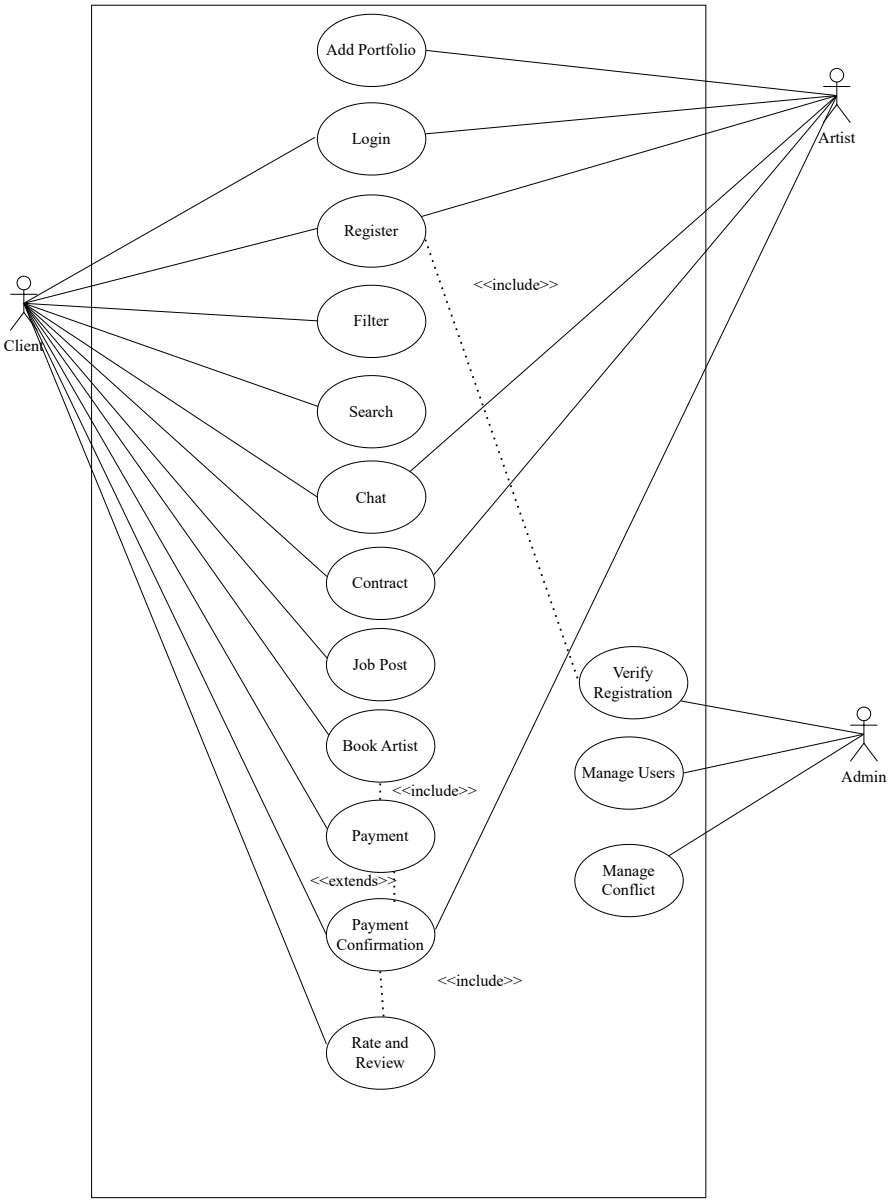

Figure 3.1: Use Case Diagram

3.1.1.2 Non-Functional Requirements

Non-functional requirements define the overall performance, qualities, attributes, and what the system includes.

- **Performance**: This system will ensure that the boking and search functionalities load within a few second.
- **Scalability**: It must support a growing numbers of users and bookings.
- **Usability**: The system should be simple, user-friendly UI for both artist and Clients.
- **Security**: The system should provide secure login , payment encryption and data privacy.
- **Maintainability**: The system is easy to maintain and update, with clear documentation and support for future modifications.

3.1.2 Feasibility Study

The feasibility study for the project was conducted based on 4 parameters and evaluated its practicality across technical, schedule, and operational dimensions.

3.1.2.1 Technical Feasibility

The freelance Artist Booking system is viable and can be implemented with stable and widely used technologies. The MERN stack(MongoDB , Express.js, React.js and Node.js) that allows seamless integration between frontend and backend portion. These tools are well documents and supported , ensuring smooth development and deployment processes.

3.1.2.2 Operational Feasibility

Operationally , the system is feasible due to increasing demand of freelance artists such as DJs, musicians, dancers, MCs. It offers one stop centralized platform where user have access and book entertainers, artists and allows artists to upload their portfolios, handle booking and update calendar. This reduces the hassle of manual booking and promotes professionalism in the freelance sector. The simplicity of use, which enables performers and clients to access the platform anywhere, enhancing user satisfaction and engagement.

3.1.2.3 Economic Feasibility

From the financial point of view, the initial development expense of the system experiences design, development, cloud hosting and third-part services integration fees. Monetization is achievable through various streams such as charging a commission for each successful booking, offering premium listings features for those artists who want additional visibility.

3.1.2.4 Schedule Feasibility

From a timeline point of view, the project is feasible to be finished in a well planned 12 week schedule. The first one week are spent on brainstorming, planning, requirement gathering and a week for designing, weeks four to eleven will be utilized for core frontend and backend development. In this period complex functionalities such as real time messaging, review, payment integration, search and filter function according to the location. Testing and bug fixing would happen between weeks ten and twelve. With clear milestones and steady progress this schedule is feasible and realistic.

Table 3.1: Schedule Table

Task	Start Date	Days to Complete
Planning	4/5/2025	8 days
Requirement Analysis	11/5/2025	8 days
System Design	18/5/2025	10 days
Coding	28/5/2025	40 days
Testing	10/6/2025	34 days
Documentation	4/5/2025	84 days

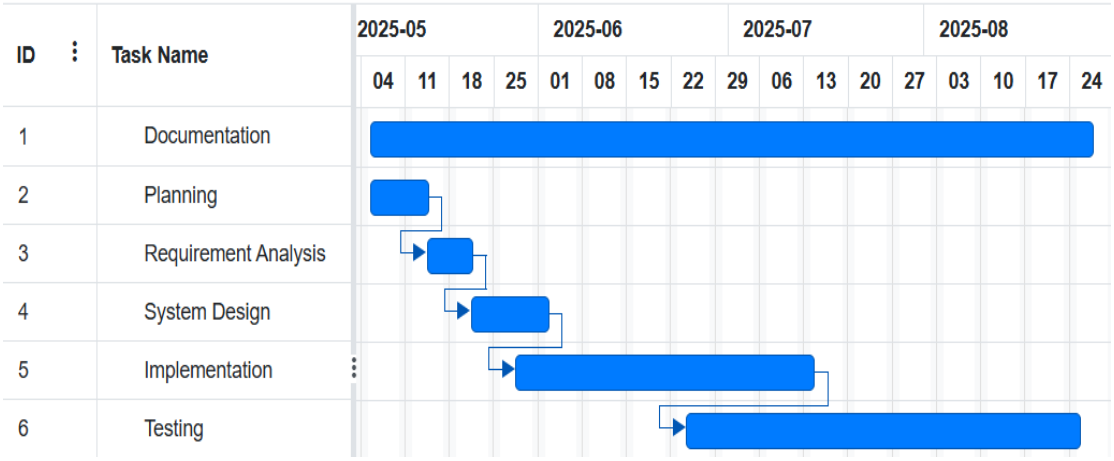

Figure 3.2: Gantt Chart of Project Schedule

3.1.3 Analysis

The Freelance Artist Booking System is evaluated based on its technical, operational, and schedule feasibility. It is developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), ensuring scalability, high performance, and efficient development. The chosen technologies enable real-time booking, secure user authentication, and robust

data management. The platform serves both artists and clients, providing features such as profile creation, booking management, integrated messaging, and secure payment options to ensure smooth communication and transactions. With a user-friendly interface, the system enhances visibility and opportunities for freelance artists.The development timeline is realistic, with key milestones and thorough testing planned, ensuring the system will be delivered on time.

3.1.4 Data modeling using ER Diagram

An Entity-Relationship (ER) diagram is a visual representation used to model the structure of a database. It illustrates the relationships between entities and their attributes. ER diagrams use symbols like rectangles to represent entities, diamonds for relationships, and ovals for attributes, helping to design and communicate the data structure clearly before actual implementation.

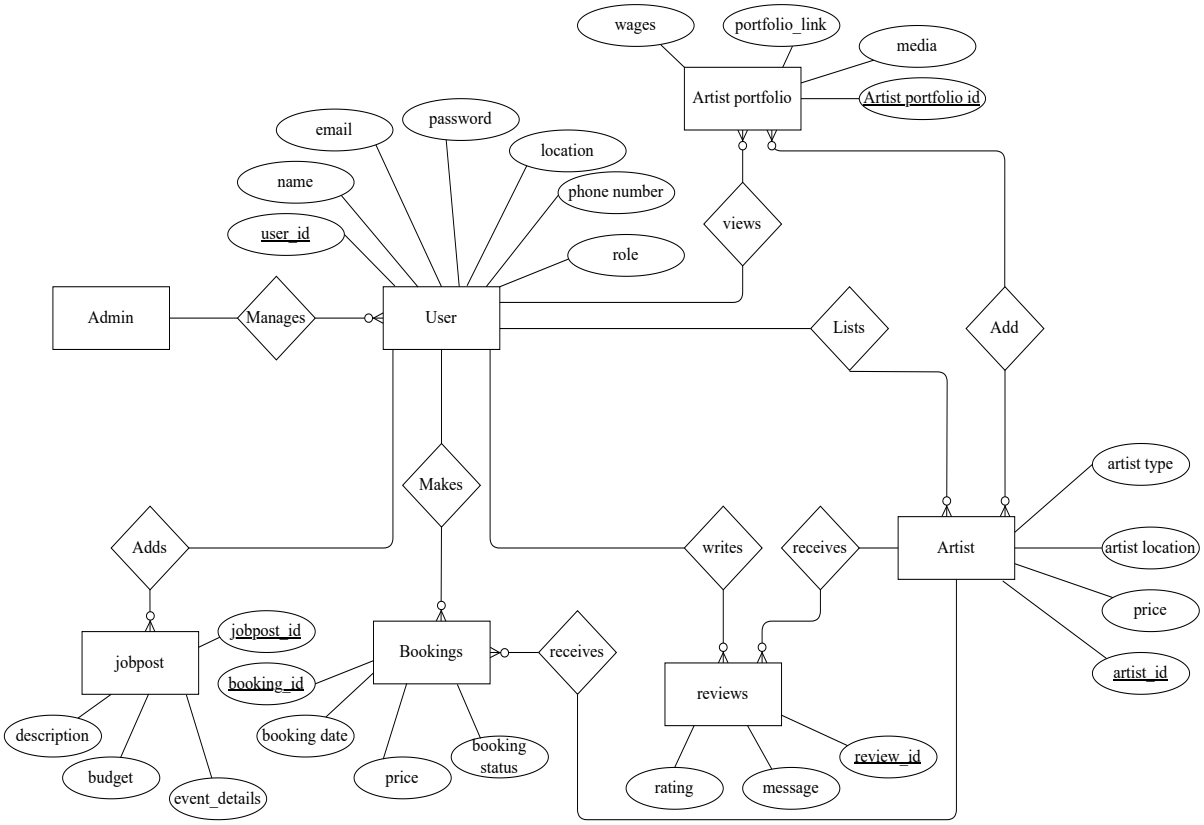

Figure 3.3: ER Diagram

In the above ER diagram, the main entities, their attributes, and relationships are described as follows:

1. **Artist Portfolio**: It contains additional media and links that showcase the artist's work. It is associated with a specific artist. The attributes include wages, portfolio_link, media, and artist_portfolio_id.

2. **Booking**: It records event bookings made by users for artists. The key attributes include booking_id, booking_date, price, and booking_status. Each booking links a User and an Artist.

3. **Review**: A user can write reviews for artists after a booking. The attributes include review_id, rating, and message.

4. **Job Post**: It allows users to create custom event requests with details such as jobpost_id, event_details, budget, and description. Artists can view and apply for these job posts.

Relationships:

1. A User can make multiple Bookings with different Artists.
2. A Booking is linked to one User and one Artist.
3. A User can write Reviews for an Artist they booked.
4. An Artist can have one or more Portfolios, which contain media showcasing their work.
5. A User can create Job Posts, which can be listed for artists.

3.1.5 Process modeling using DFD

The process modeling of the Freelance Artist Booking System is done using Data Flow Diagrams (DFD). DFDs show how data moves through the system and the interactions between Client, Artist, admins, and different system components. It illustrates processes like registration, artist profile listing, booking, management, payment handling, and reviews, along with the data inputs, outputs, and storage. DFDs help in understanding the system's functions and identifying areas for improvement.

1. Level 0 DFD

Level 0 DFD for Freelance Artist Booking System represents the highest-level view showing the entire system as a single process. It illustrates how the system interacts with external entities such as Client, Artist and Admin without showing any inter process or data stores. It's essentially a bird's eye view showing data flowing in and out of the system boundary.

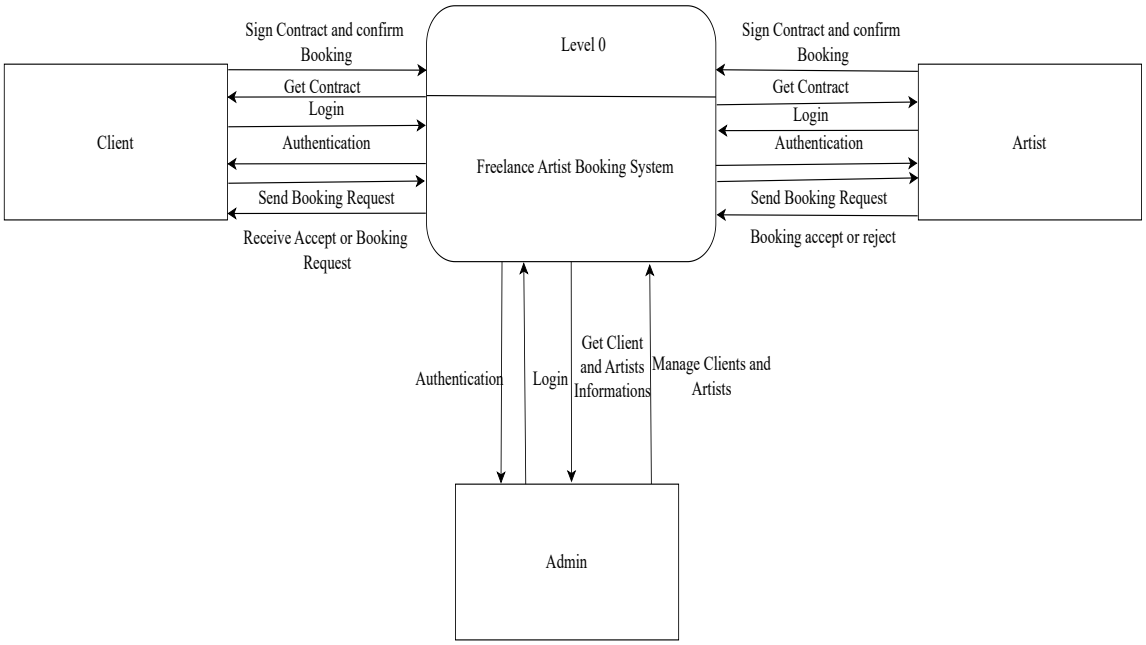

Figure 3.4: DFD Level 0 diagram

2. Level 1 DFD of Client

Level 1 DFD for Client and Artist breaks down their interactions into detailed processes. Clients can register/login, search for artists, view their profile, portfolio, and make bookings, write reviews, and make Payments.

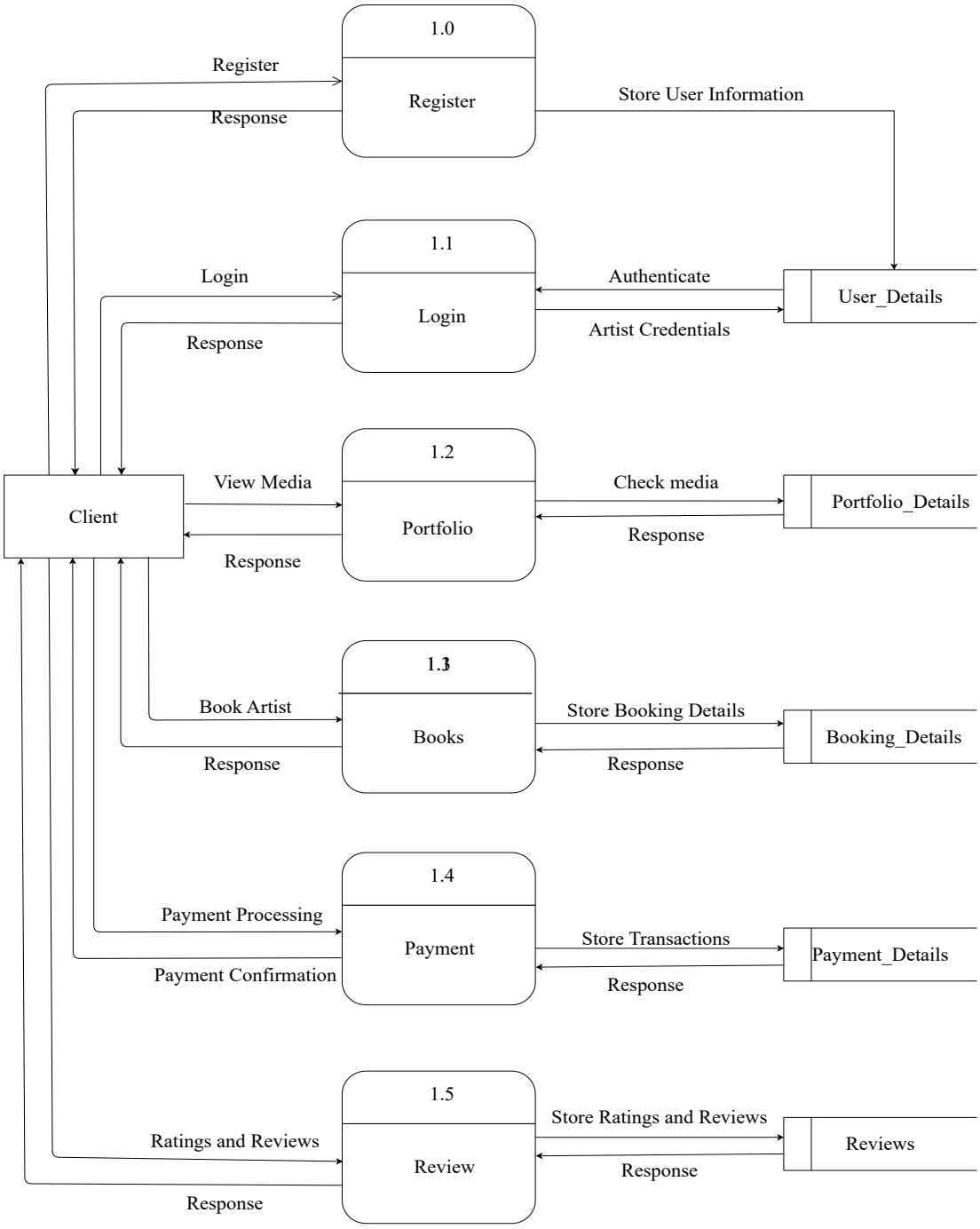

Figure 3.5: DFD Level 1 diagram of Client

2. Level 1 DFD of Artist

Level 1 DFD for Artist illustrates detailed interactions where artists can register/login, manage and update their profiles and portfolios, set Availability, receive and respond to booking requests, view client reviews, and track payments and earnings.

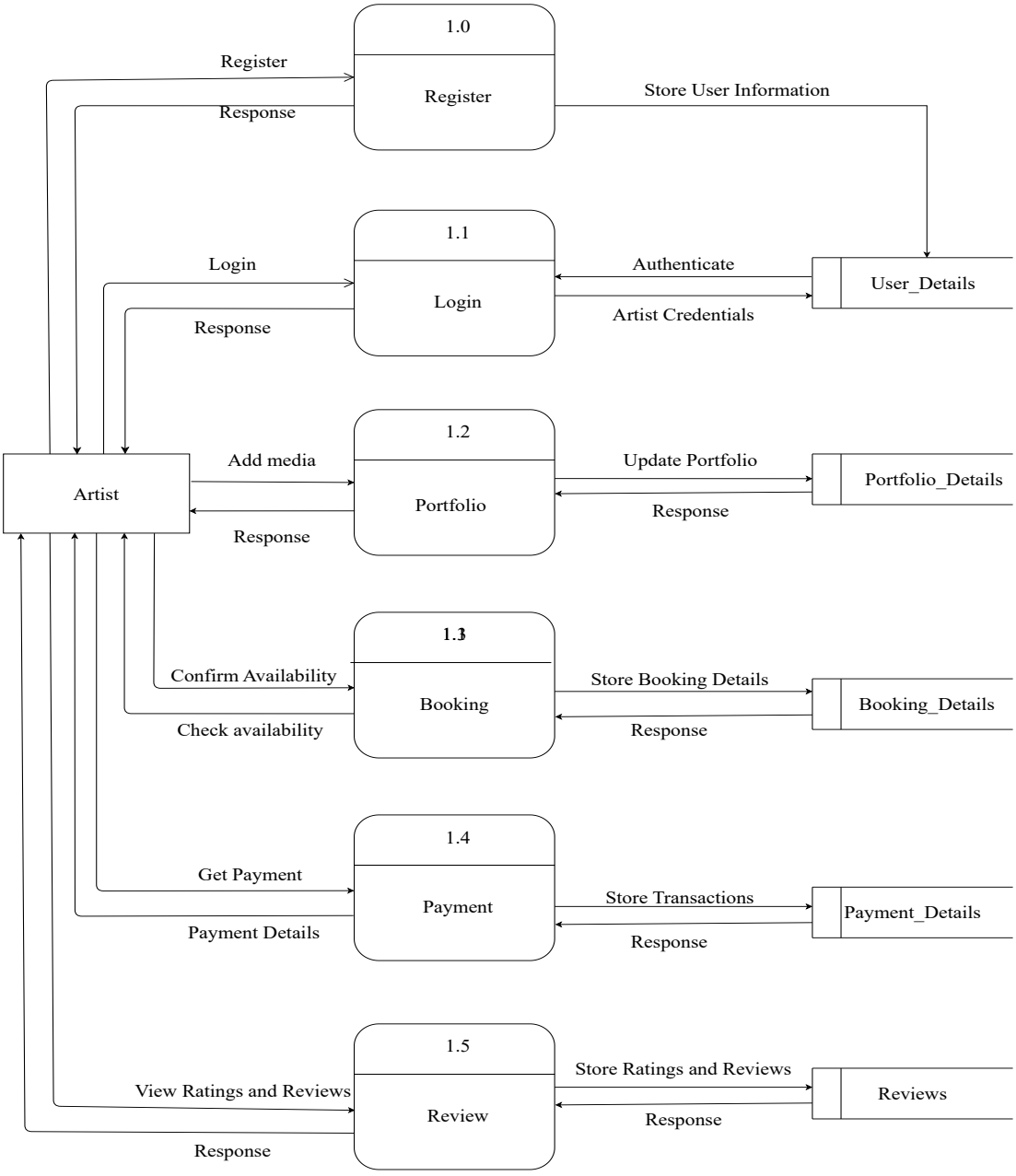

Figure 3.6: DFD Level 1 diagram of Artist

CHAPTER 4

SYSTEM DESIGN

4.1 Design

System design defines the architecture, components, modules, interfaces, and data structures of the Freelance Artist Booking System to meet the platform's functional and non-functional requirements. It shows how everything works together like features, user roles and how they communicate with each other. This design turns user needs and business goals into a clear technical plan. It includes both the big picture like how users (clients) find artists and make bookings and the small details like managing profiles, handling contracts, and making payments. This plan helps developers build and launch the system smoothly. This ensures a well-organized, efficient, and scalable solution for the Freelance Artist Booking System.

4.1.1 Database Design

The database integrated into our system is MongoDB which is a document-oriented NoSQL database designed for storing, retrieving, and managing large volumes of un-structured or semi-structured data.
The main parts of the database includes:

- **Categories:** It defines the different types of artists available on the platform. Each artist belongs to one or more categories based on the kind of service they provide. This helps users filter and search for the right artist for their event.
- **User detail:** It stores information about everyone using the system (both clients and artists), with roles to tell who is who.
- **Bookings**: Keeps track of who booked which artist, when, and the status of the booking.

- **Portfolio details:** It stores the media and information artists share to showcase their work. It includes photos, videos descriptions, and links that help clients understand the artist's style and skills before booking them.

- **Jobpost:** It manages any event or job listings posted by clients. It contains details like event date, location, type, and specific requirements so artists can see available opportunities and apply or be booked accordingly.

- **Chat:** It stores messages exchanged between clients and artists. It supports real-time communication for discussing bookings and clarifying details.

- **Contract:** It hold the agreements between artists and clients. They include terms of service, payment conditions, event details, and signatures to ensure both parties understand their responsibilities.

- **Payment:** It tracks all financial transactions. It records advance payments, payment confirmations, refunds, and the payment status related to bookings, ensuring secure and transparent money handling.

- **Reviews:** It collect feedback from clients after an event or booking. Clients can rate artists and write reviews about their performance. This helps build trust and lets future clients make informed decisions.

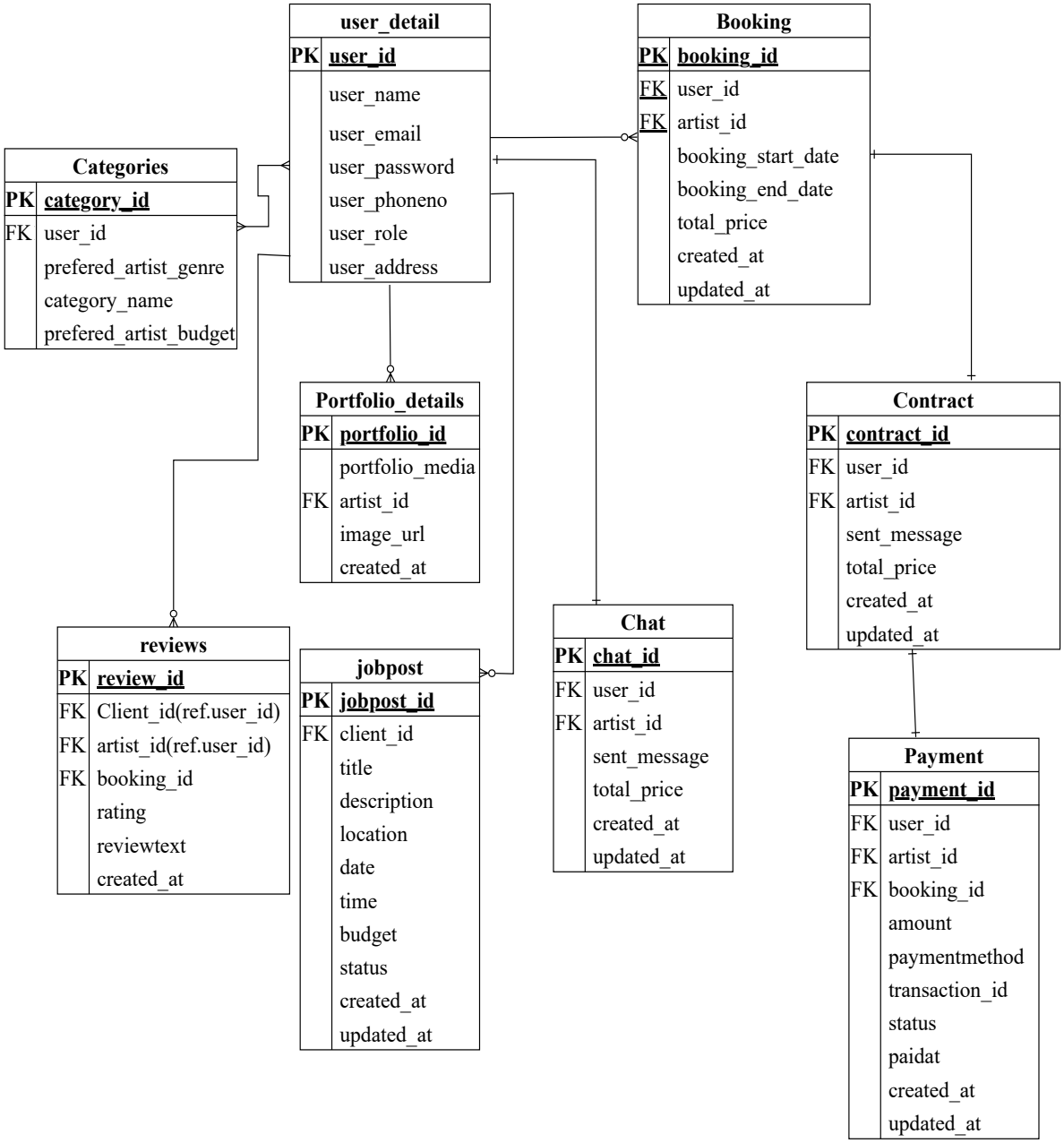

Figure 4.1: Database design

4.1.2 System Design

The Freelance Artist Booking System is designed to provide a seamless and user-friendly experience. The user interface is built with React JS, Tailwind CSS, and Daisy

UI, ensuring visually appealing design and intuitive navigation. The authentication and authorization module handles tasks like user registration, login, and password management using JWT for token-based authentication and Bcrypt for secure password hashing. The server-side application is powered by Node.js and Express.js, efficiently managing the backend processes and business logic. MongoDB serves as the primary database for structured data storage, and Cloudinary is used for handling media.

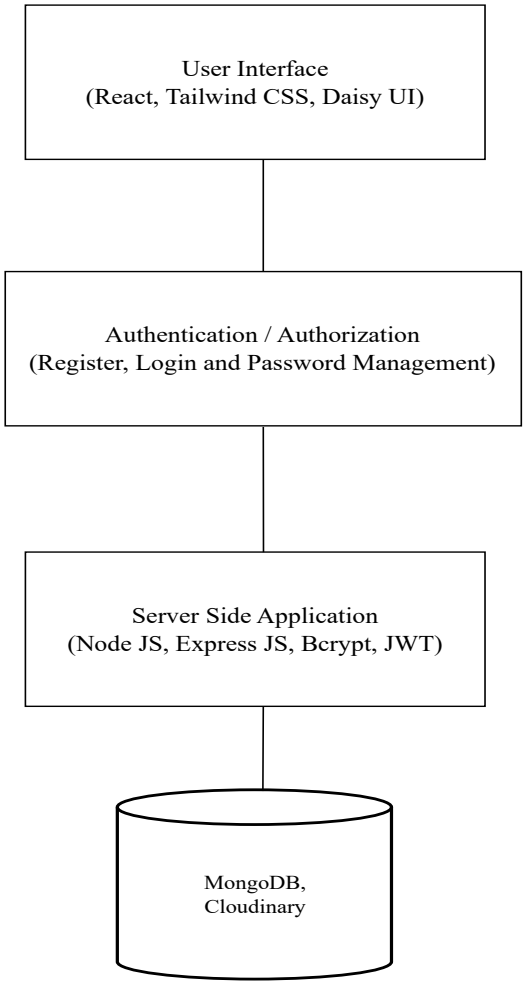

Figure 4.2: System Design

4.1.3 Flowchart

The flowchart of the Freelance Artist Booking System illustrates the process flow among its primary users: Client, Artist, and Admin. Clients can search for artists, view artist

details, make bookings, and complete payment. Artists can register, add their portfolio, accept or reject bookings, and receive their payments. Clients and artists can connect through the chat application. Both parties confirm bookings through signing contracts. Admins verify registrations, and manage users.The flowchart visually connects these actions, showing how data moves between users and the system to ensure seamless functionality.

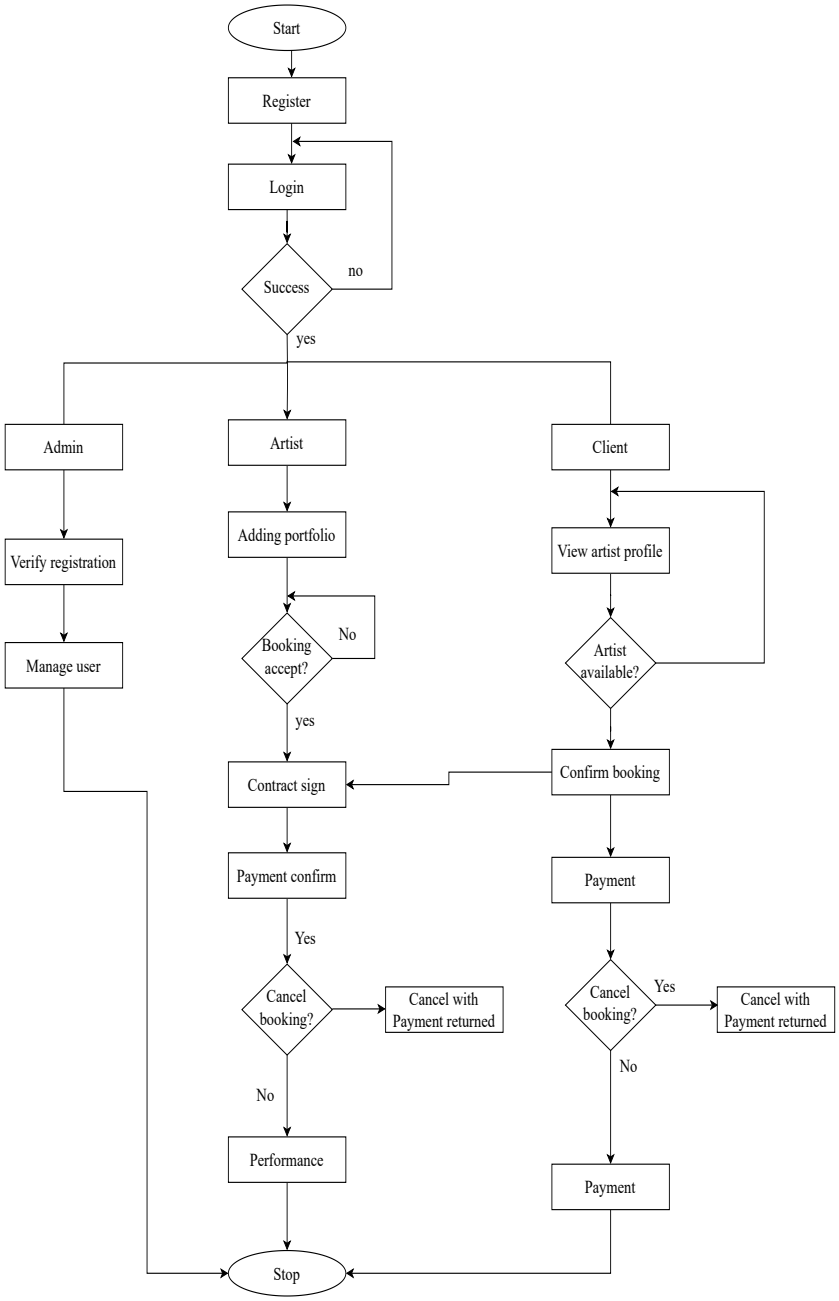

Figure 4.3: Flowchart

4.1.4 Forms and Report Design

1. **Register User Form:** This form will be used by new users to create an account. It should include fields for username, email, password, and potentially other relevant information.

2. **Login Form:** This form allows existing users to log in to the system. It will require the user to enter their username or email and password.

3. **Forgot Password Form:** This form allows users to recover their passwords if they have forgotten them. It typically requires the user to enter their registered email address, and the system will send a password reset link to their email.

4. **Reset Password Form:** This form is accessed via the password reset link and allows the user to create a new password for their account.

4.2 Algorithm Details

The system implements algorithms such as the Hybrid recommendation algorithm , Haversine algorithm, and the filtering algorithm. This is for users to get recommendations and also to search as per their needs.

4.2.1 Filtering Algorithm

This algorithm ranks the artist according to budget, category, location, genre, specialties, and language.

- **Cosine similarity**

  Matches preferences between client preferences and artist tags (semantic match).

- **Geographic distance**

  It is done by the Haversine Algorithm

- **Pricing compatibility**

  It is done by comparing the budget.

4.2.1.1 Steps in Filtering Algorithm

1. **Input from Client**

   The request body includes:

   - userLat, `userLng`: Client's coordinates.
   - maxBudget: Maximum hourly rate they can pay.
   - preferences: { genres, specialties, languages }
   - categoryName: e.g., "singer"

2. **Define Tags and Create Vectors**

   A fixed list of all possible tags is defined as:

   ```
   TAGS = ["musician", "dj", "singer", ..., "wedding"]
   ```

   The user's preferences and the artist's tags are converted into binary vectors using the createVector() function.

   ```
   userVector = [0, 0, 0, 0, 0, 1, 0, ..., 0, 1]  // 1 for matching tags
   ```

3. **Cosine Similarity**

   The cosine similarity between the userVector and each artistVector is computed. This measures how similar the client and artist are in terms of tags.

   - A score of 1 indicates a perfect match.
   - A score of 0 indicates no similarity.

4. **Distance Calculation (Haversine)**

   For artists with valid coordinates:

   ```
   const distance = haversine(userLat, userLng, artistLat, artistLng)
   ```

This gives the physical distance in kilometers. It's then normalized:

```
normalizedDistance = Math.min(distance / 50, 1)
```

- 0 km $\rightarrow$ 0
- 50+ km $\rightarrow$ 1

(Closer = higher score)

5. **Budget Fit**

   The algorithm checks how well the artist's price fits the client's `maxBudget`:

   - If the artist is within budget: `priceScore = 1`
   - If over budget: score decreases linearly

6. **Final Weighted Score**

   All three factors are combined using a weighted linear formula:

```
finalScore =
  0.6 * cosineScore +
  0.3 * (1 - normalizedDistance) +
  0.1 * priceScore
```

   Weights:

   - 60% weight to tag similarity (cosine match)
   - 30% weight to geographic proximity
   - 10% weight to budget compatibility

7. **Labeling and Ranking**

   Based on the final score, artists are labeled:

   - ≥ 0.9: **Best Match**

- ≥ 0.75: **Excellent Match**
- ≥ 0.5: **Good Match**

Artists are then:

- **Filtered** (to remove budget-incompatible or invalid entries)
- **Sorted** by descending final score, and by ascending distance in case of ties

4.2.2 Haversine Algorithm

The Haversine formula is used to calculate the shortest distance between two points on a sphere using their latitude and longitude coordinates. It is used to identify nearby artists relative to the users location.

4.2.2.1 Key Terms

Latitude(ϕ)

Latitude is the angular distance of a point north or south of the Earth's equator, measured in degrees. In the Haversine formula, latitude values are converted to radians for accurate trigonometric calculations.

Longitude(λ)

Longitude is the angular distance of a point east or west of the Prime Meridian, also measured in degrees. Like latitude, it must be converted to radians before being used in the Haversine formula.

Δ Latitude($\Delta\lambda$)

The difference in latitude between two geographic points. Calculated as:

$$\Delta\phi = \phi_2 - \phi_1$$

Δ Longitude$(\Delta\lambda)$

The difference in longitude between two geographic points. Calculated as:.

$$\Delta\lambda = \lambda_2 - \lambda_1$$

Radius of the Earth(r)

A constant representing the Earth's average radius, typically taken as 6,371 km. This value is used to convert the angular distance into a physical distance.

Haversine Function

The Haversine function is defined as:

$$\text{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right)$$

It is used in the formula to calculate great circle distances on a sphere.

4.2.2.2 Steps in Haversine Algorithm

1. **Input Coordinates**

   Obtain the latitude and longitude of the two points

   - **Point 1:** ϕ_1, λ_1
   - **Point 2:** ϕ_2, λ_2

2. Convert degrees to radians

$$\phi = \phi \times \frac{\pi}{180}, \quad \lambda = \lambda \times \frac{\pi}{180}$$

3. Calculate differences

$$\Delta\phi = \phi_2 - \phi_1$$

$$\Delta\lambda = \lambda_2 - \lambda_1$$

4. Apply the Haversine formula

   The distance d between two points on the Earth's surface is calculated as:

$$d = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

   Where:

   - r is the radius of the Earth (approximately 6371 km),
   - ϕ_1, ϕ_2 are the latitudes of the two points in radians,
   - $\Delta\phi = \phi_2 - \phi_1$ is the difference in latitudes,
   - $\Delta\lambda = \lambda_2 - \lambda_1$ is the difference in longitudes.

4.2.2.3 Haversine Algorithm Integration

Distance Calculation Using Haversine Algorithm

The calculateDistance function is responsible for determining the geographical distance between two locations, most likely the client's location and the artist's location. This is important for a freelance artist booking platform that may need to:

- Suggest nearby artists to clients based on proximity.
- Calculate travel cost or assess the feasibility of booking depending on the distance.

How It Works

1. **Get Coordinates from Browser (Artist Location)**

   The application uses the Geolocation API to retrieve the artist's current latitude and longitude:

```
navigator.geolocation.getCurrentPosition(...)
```

These coordinates are then stored in the artistPosition state variable.

2. **Visual Routing on the Map (Leaflet Algorithm)**

   The route between the artist and the booking location is drawn on a map using the Leaflet Routing Machine library:

```
const routingControl = L.Routing.control({
  waypoints: [
    L.latLng(artistPosition[0], artistPosition[1]),
    L.latLng(bookingLocation[0], bookingLocation[1]),
  ],
  ...
}).addTo(map);
```

   This component uses OpenStreetMap data to generate and render a route visually on the map. It helps the artist understand the direction and distance without calculating the actual value.

3. **Precise Distance Calculation (Backend Haversine API)**

   To calculate the real-world distance, the application sends both coordinates to the backend API using a POST request:

```
const res = await fetch("http://localhost:5000/api/dis/distance", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({
    lat1: latitude,
    lon1: longitude,
    lat2: bookingLocation[0],
    lon2: bookingLocation[1],
  }),
```

```
});
```

The backend server, using the Haversine formula, calculates the great-circle distance between the two points and returns the value in kilometers.

4. **Display Result**

   Finally, the application displays the computed distance to the user:

```
<p className="mt-2 text-center text-sm text-gray-700">
  Distance to booking: <strong>{distance} km</strong>
</p>
```

   This provides a user-friendly way for the artist to understand how far they are from the booking location.

The calculateDistance function is responsible for determining the geographical distance between two locations, typically the client's location and the artist's location. This is important for a freelance artist booking platform because it may be used to:

- Suggest nearby artists to clients based on proximity.
- Calculate travel cost or assess the feasibility of booking depending on the distance.

4.2.3 Linear Search Algorithm

Linear Search is a simple algorithm that checks each element in a list one by one until the target is found. It is implemented in the search functionality of the system for small datasets, where users search for artists based on keywords. Each item in a dataset is scanned sequentially to calculate a match score against a given query. It operates in O(n) time for filtering and O(n log n) for final sorting, making it effective for datasets with a manageable size.

4.2.3.1 Steps in the Linear Search Algorithm

1. **Input Query**

   Accept a search term (query) from the user.

2. **Initialize Score for Each Item**

   Assign a score of 0 to each item in the dataset (e.g., artist profiles).

3. **Match Attributes and Calculate Score**

   - If the artist's name starts with the query: +3 points
   - If the name contains the query (but not at the start): +2 points
   - If the category contains the query: +1 point

4. **Filter Non-Matching Items**

   Remove items whose total score is zero (i.e., they are not relevant to the search query).

5. **Sort by Score**

   Arrange the remaining items in descending order based on their score to rank their relevance.

6. **Display Results**

   Render the sorted list in the user interface for the user to view.

4.2.3.2 Mathematical Formula

Let n be the number of total items (e.g., artist profiles). The time complexity of the algorithm is:

$$T(n) = a \cdot n + b \cdot n + c \cdot n \log n$$

Where:

- $a \cdot n$: time for scoring using map()
- $b \cdot n$: time for filtering using filter()
- $c \cdot n \log n$: time for sorting using sort()

After removing constants, the final time complexity is:

$$T(n) = O(n \log n)$$

4.2.3.3 Linear Search Algorithm Integration

The linear search filtering algorithm is applied in the search functionality of an artist booking system. When a user inputs a search query such as "dan", the system evaluates each artist profile as follows:

- If the artist's name starts with "dan", it receives a high score (+3).
- If the artist's name contains "dan" (not at the start), it receives a moderate score (+2).
- If the artist's category contains the query (e.g., "dancer"), it receives a lower score (+1).

Only artists with a non-zero total score are considered relevant and are sorted by score to prioritize the best matches.

4.2.4 Hybrid Recommendation System

4.2.4.1 Steps in Hybrid Recommendation System

1. **Identify User Type**

   Check if the user is new (no prior bookings) or returning (has booking history).

2. **Compute Content-Based Scores (for returning users)**

   Compare previously booked artists with others using category, genres, location, and ratings. Sum similarity scores to get a content-based score.

3. **Retrieve Collaborative Scores**

   Calculate scores for all artists based on booking count, completion rate, and ratings to reflect popularity and reliability.

4. **Combine Scores**

   Merge content-based and collaborative scores using weighted averaging (70% content-based, 30% collaborative).

5. **Filter and Sort**

   Remove already booked artists, sort by final score, and return the top recommendations.

4.2.5 Linear Weighted Scoring Algorithm

This is a custom scoring algorithm used to rank or prioritize bookings based on their status, contract progress, and recency. It's a mathematical method for ranking or scoring items by assigning each input a weight and a value, then combining them linearly (i.e., via multiplication and addition).

4.2.5.1 Steps in Linear Weighted Scoring Algorithm

It assigns a numerical score to each booking by combining:

$$\text{Score} = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n$$

1. **Booking status**

   - Different statuses like completed, booked, pending are assigned weights (e.g., completed = 100).

- These represent how valuable or important the status is.

2. **Contract status (contractScore)**

   - It defines whether the contract signed, generated, or none.
   - It indicates how far the contract process has progressed

3. **Recency (recencyScore):**

   - It measures how recent the last update was (more recent = higher score).
   - Diminishes over time (linearly drops to 0 after 50 hours).

   The final score is calculated with weighted importance:

   Booking Score = 0.5 * statusScore + 0.3 * contractScore + 0.2 * recencyScore

   This means:

   - Status is the most important (50 percent weight).
   - Contract state matters (30 percent weight).
   - Recency also contributes (20 percent weight).

CHAPTER 5

IMPLEMENTATION AND TESTING

5.1 Implementation

System implementation is a key phase in the development of our project, where the system design is transformed into a working and operational solution. This phase ensures that the system meets the required quality standards and functions as intended.

5.1.1 Tools Used

5.1.1.1 Frontend Development

1. **Figma:** We are using figma for designing UI/UX mockups and prototypes.

2. **JavaScript:**: JavaScript served as the core programming language for frontend development. It was used to add interactivity and dynamic behavior to the application, enabling features such as form validation, animations, and real-time data fetching.

3. **React:** We used React to build the application's user interface. It enabled the creation of reusable components and efficient updates to the DOM, which improved application speed, maintainability, and scalability.

4. **React Router DOM:** This library was integrated to handle client-side routing within the React application. It allowed seamless navigation between different views or components without full page reloads, contributing to a smoother and faster user experience.

5. **Vite:** Implemented as the build tool to ensure fast and efficient development server, enhancing productivity.

6. **Zustand:** Chosen for global state management due to its simplicity and performance. Zustand provides a lightweight and flexible solution for managing application state without the overhead of more complex libraries

UI Frameworks and Libraries

1. **Tailwind CSS:** Tailwind CSS is a utility-first CSS framework that was used to style the user interface of the application. It provides low-level utility classes that allow developers to build custom designs directly in the markup without writing traditional CSS. This approach made it faster and more efficient to create a responsive and consistent layout across different components.

2. **DaisyUI:** DaisyUI is a Tailwind CSS based component library that extends Tailwind with a set of pre-designed, accessible UI components. It simplifies frontend development by providing ready to use elements such as buttons, forms etc. It allows rapid styling and easy theme customization while maintaining consistency in design. In this project, DaisyUI was used for designing the chat application interface, enabling the development of components like message input fields, chat bubbles and layout containers.

5.1.1.2 Backend Development

1. **Node.js:** Node.js is a JavaScript runtime environment that allows developers to run JavaScript on the server side.Node is used extensively for server-side programming, making it possible for developers to use JavaScript for client-side and server-side code without needing to learn an additional language.

2. **Express.js:** Express.js is a minimal and flexible Node.js web application framework. It is designed for building robust web applications and APIs by providing a set of features for handling HTTP requests, routing, middleware management, and more.

3. **Socket.io:** Socket.io is a javascript library that enables real-time bidirectional communication between users and servers. Socket.io is used for the real-time communication between the client-server which enables chat functionality in chat application. Socket.io was used on both frontend and backend to implement real-time communication. The socket.io-client library allowed the frontend to send and receive messages instantly, while the server-side socket.io handled message broadcasting and socket event handling.

5.1.1.3 Database

1. **Mongo DB**: MongoDB is a NoSQL, document-oriented database designed for high performance, scalability, and flexibility.It stores data in JSON-like documents with dynamic schemas, making it ideal for applications that require rapid development and changing data structures.

2. **Cloudinary**: Cloudinary is an end-to-end image and video management solution for websites. Cloudinary is used to handle image uploads and manage media storage. It provides secure APIs to upload and retrieve media, with support for automatic transformations like resizing, cropping, and format conversion

5.1.1.4 Version Control and Project Management

1. **GitHub:** Serves as the centralized repository for code collaboration and version control. GitHub Projects is employed for task assignment and progress tracking.

5.1.1.5 Documentation

1. **Overleaf:** Used for creating and managing project documentation in LaTeX. Overleaf provides a collaborative, cloud-based platform for writing, editing, and compiling LaTeX documents efficiently.

2. **Draw.io:** Utilized for designing ER diagrams, DFD diagram,Use Case diagram,offering an intuitive interface for creating clear and professional visual representations of system architecture, workflows, and processes.

3. **Trello:** We used Trello as our project management tool to plan, track, and manage the development of the Freelance Artist Booking System. It provided a visual kanban board where we organized tasks into categories such as To Do, In Progress, and Completed.

5.1.1.6 Other Tools

- **Visual Studio Code**: It is used for efficient code development with extensions for debugging, version control, and language support.

- **Postman**:we used Postman for testing and debugging APIs.

5.1.2 Implementation Details of Modules

5.1.2.1 Filtering Module

```
const getMatchLabel = (score) => {
  if (score >= 0.9) return "Best Match";
  if (score >= 0.75) return " Excellent Match";
  if (score >= 0.5) return " Good Match";
  return null;
};

const getArtistMatches = async (req, res) => {
  try {
    const { userLat, userLng, maxBudget, distanceLimit = 50, preferences
    ↪  = {}, categoryName } = req.body;
```

```
if (!userLat || !userLng) {
  return res.status(400).json({ error: "User coordinates are
  ↪ required" });
}

const categoryFilter = categoryName
  ? { category: categoryName.toLowerCase() }
  : {};

const artists = await Artist.find(categoryFilter).lean();

if (artists.length === 0) {
  return res.status(200).json([]);
}

// Collect preference tags from user request
const preferenceTags = [
  ...(preferences.genres || []),
  ...(preferences.specialties || []),
  ...(preferences.languages || []),
  categoryName || ""
].map(t => t.toLowerCase());

const userVector = createVector(preferenceTags);

const scored = artists
  .filter(a => a.location?.coordinates?.length === 2)
  .map((artist) => {
    const [lng, lat] = artist.location.coordinates;
    const distance = haversine(userLat, userLng, lat, lng);
    return {
      ...artist,
      distance: Number(distance.toFixed(2))
    };
```

```
  })
  .filter(artist => artist.distance <= distanceLimit);

if (scored.length === 0) {
  return res.status(200).json([]);
}

const finalScored = scored.map((artist) => {
  const tags = [
    ...(artist.genres || []),
    ...(artist.specialties || []),
    ...(artist.languages || []),
    ...(artist.eventTypes || []),
    artist.category || ""
  ].map(t => t.toLowerCase());

  const artistVector = createVector(tags);
  const cosineScore = cosineSimilarity(userVector, artistVector);

  const normalizedDistance = Math.min(artist.distance /
  ↪  distanceLimit, 1);

  const wage = artist.ratePerHour || artist.wage || 0;
  let priceScore = 1; // default if no budget

  if (maxBudget) {
    if (wage <= maxBudget) {
      const priceDiff = Math.abs(wage - maxBudget);
      priceScore = 1 - (priceDiff / maxBudget) * 0.3;
    } else {
      priceScore = Math.max(0, 1 - (wage - maxBudget) / maxBudget);
    }
  }
```

```
    const priceWeight = maxBudget ? 0.25 : 0.1;
    const contentWeight = maxBudget ? 0.45 : 0.6;
    const distanceWeight = 0.3;

    const finalScore = contentWeight * cosineScore
                     + distanceWeight * (1 - normalizedDistance)
                     + priceWeight * priceScore;

    return {
      ...artist,
      score: Number(finalScore.toFixed(4)),
      matchLabel: getMatchLabel(finalScore)
    };
  });

  const ranked = finalScored
    .filter(a => a.score > 0)
    .sort((a, b) => {
      if (b.score !== a.score) return b.score - a.score;
      return a.distance - b.distance;
    });
```

Artist Matching Algorithm Implementation

- The system receives user input including location (userLat, userLng), maxBudget, categoryName, and preference tags (genres, specialties, languages).
- Both the user's preferences and each artist's profile data are converted into binary tag vectors using the createVector() function, based on a fixed tag list.
- Cosine similarity is calculated between the user and artist vectors to measure semantic relevance.
- The geographical distance between the user and artist is computed using the Haversine formula and normalized to a 0–1 scale assuming a 50km max relevance radius.
- A price score is calculated: artists within the user's budget receive full score; those over budget are penalized smoothly.
- A final weighted score is computed using the formula:

  $$\text{Final Score} = 0.6 \times \text{Cosine Similarity} + 0.3 \times (1 - \text{Normalized Distance}) + 0.1 \times \text{Price Score}$$

- Artists are labeled (e.g., "Best Match") based on score thresholds and returned after filtering and sorting by score and proximity.

5.1.2.2 Booking Module

```
export const statusWeights = {
  completed: 100,
  booked: 80,
  accepted: 60,
  pending: 30,
  rejected: 10,
  cancelled: 5,
```

```
};

export const contractWeights = {
  signed: 50,
  generated: 30,
  none: 0,
};

export function getRecencyWeight(timestamp) {
  const hoursAgo = (Date.now() - new Date(timestamp)) / (1000 * 60 *
  ↪ 60);
  return Math.max(0, 50 - hoursAgo); +
}

export function calculateBookingScore(booking) {
  const statusScore = statusWeights[booking.status] || 0;
  const contractScore = contractWeights[booking.contractStatus ||
  ↪ "none"] || 0;
  const recencyScore = getRecencyWeight(booking.lastActionTime ||
  ↪ booking.updatedAt);

  return (
    0.5 * statusScore +
    0.3 * contractScore +
    0.2 * recencyScore
  );
}
```

Booking Score Calculation

- Assigns weights to booking status, contract status, and recency using predefined lookup tables.
- Computes how recent a booking is (within 50 hours) to influence score.

- Calculates a final weighted score:

$$\text{Score} = 0.5 \times \text{Status} + 0.3 \times \text{Contract} + 0.2 \times \text{Recency}$$

- Used to rank or prioritize bookings for dashboards or analytics.

5.1.2.3 Linear Search Algorithm

```
router.get("/search", async (req, res) => {
  try {
    const query = req.query.query?.toLowerCase() || "";

    if (!query) return res.json([]);

    const artists = await Artist.find({
      $or: [
        { username: { $regex: query, $options: "i" } },
        { category: { $regex: query, $options: "i" } },
        { "location.city": { $regex: query, $options: "i" } },
      ],
    });

    const scored = artists
      .map((artist) => {
        let score = 0;
        const name = artist.username?.toLowerCase() || "";
        const category = artist.category?.toLowerCase() || "";
        const locationCity = artist.location?.city?.toLowerCase() || "";

        if (name.startsWith(query)) score += 3;
        else if (name.includes(query)) score += 2;

        if (category.includes(query)) score += 1;
        if (category.includes(query) || locationCity.includes(query))
        ↪  score += 1;
```

```
        console.log(`Artist: ${artist.username}, Score: ${score}`);
        return { ...artist.toObject(), score };
      })
      .filter((a) => a.score > 0)
      .sort((a, b) => b.score - a.score);

    res.json(scored);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Search failed" });
  }
});
```

Linear Search Filtering Algorithm Implementation

- Applied in the search functionality of the artist booking system.
- Iterates through all artist profiles linearly for each search query.
- Assigns a score based on matching criteria:
  - +3 if artist's name starts with the query.
  - +2 if artist's name contains the query (not at the start).
  - +1 if artist's category contains the query.
- Only artists with a total score > 0 are considered relevant.
- Filters out non-matching artists and sorts results by descending score.
- Enables real-time, keyword-based filtering for small to medium datasets.

5.1.2.4 Hybrid Recommendation Algorithm

```
const computeArtistSimilarity = (a, b) => {
  let score = 0;

  if (a.category && b.category && a.category === b.category) score +=
  ↪ 0.5;
  if (a.genres && b.genres) {
    const overlap = a.genres.filter(g => b.genres.includes(g)).length;
    score += 0.2 * overlap;
  }
  if (a.location?.city && b.location?.city && a.location.city ===
  ↪ b.location.city) {
    score += 0.2;
  }
  if (a.weightedRating != null && b.weightedRating != null) {
    const avgRating = (a.weightedRating + b.weightedRating) / 2;
    const normalized = avgRating / 5;
    score += 0.1 * normalized;
  }
  return score;
};

const getCollaborativeRecommendations = async (limit = 10) => {
  const allBookingStats = await Booking.aggregate([
    { $match: { status: { $in: ['accepted', 'completed'] } } },
    { $group: { _id: '$artist', bookingCount: { $sum: 1 },
    ↪ completedCount: { $sum: { $cond: [{ $eq: ['$status',
    ↪ 'completed'] }, 1, 0] } } } },
    { $lookup: { from: 'artists', localField: '_id', foreignField:
    ↪ '_id', as: 'artist' } },
    { $unwind: '$artist' },
```

```
    { $project: { artistId: '$_id', bookingCount: 1, completedCount: 1,
    ↪ weightedRating: '$artist.weightedRating', totalRatings:
    ↪ '$artist.totalRatings' } }
  ]);

  const maxBookings = Math.max(...allBookingStats.map(i =>
  ↪ i.bookingCount));
  const maxCompleted = Math.max(...allBookingStats.map(i =>
  ↪ i.completedCount));
  const maxTotalRatings = Math.max(...allBookingStats.map(i =>
  ↪ i.totalRatings || 1));

  const scoredArtists = allBookingStats.map(item => {
    const bookingScore = (item.bookingCount / maxBookings) * 0.4;
    const completionScore = (item.completedCount / maxCompleted) * 0.3;
    const ratingScore = ((item.weightedRating || 0) / 5) *
    ↪ (item.totalRatings / maxTotalRatings) * 0.3;
    return { artistId: item.artistId.toString(), score: bookingScore +
    ↪ completionScore + ratingScore, algorithm: 'trending' };
  });

  scoredArtists.sort((a, b) => b.score - a.score);
  return scoredArtists.slice(0, limit);
};

const getContentBasedRecommendations = async (clientId, limit = 10) => {
  const clientBookings = await Booking.find({ client: clientId
  ↪ }).select('artist').lean();
  const bookedArtistIds = [...new Set(clientBookings.map(b =>
  ↪ b.artist.toString()))];
  if (bookedArtistIds.length === 0) return [];

  const bookedArtists = await Artist.find({ _id: { $in: bookedArtistIds
  ↪ } }).lean();
```

```
  const otherArtists = await Artist.find({ _id: { $nin: bookedArtistIds
   ↪ } }).lean();

  const scores = [];
  for (const other of otherArtists) {
    let totalScore = 0;
    for (const booked of bookedArtists) totalScore +=
     ↪ computeArtistSimilarity(booked, other);
    scores.push({ artistId: other._id.toString(), score: totalScore,
     ↪ algorithm: 'for-you' });
  }

  scores.sort((a, b) => b.score - a.score);
  return scores.slice(0, limit);
};

const combineRecommendations = (contentRecs, collaborativeRecs) => {
  const weightedContent = contentRecs.map(r => ({ ...r, weight: 0.7,
   ↪ finalScore: r.score * 0.7 }));
  const weightedCollaborative = collaborativeRecs.slice(0, 3).map(r =>
   ↪ ({ ...r, weight: 0.3, finalScore: r.score * 0.3 }));
  const combined = [...weightedContent, ...weightedCollaborative];

  const uniqueRecs = new Map();
  combined.forEach(rec => {
    const existing = uniqueRecs.get(rec.artistId);
    const score = rec.finalScore;
    if (!existing || existing.finalScore < score)
     ↪ uniqueRecs.set(rec.artistId, { ...rec, finalScore: score });
  });

  return Array.from(uniqueRecs.values()).sort((a, b) => b.finalScore -
   ↪ a.finalScore).slice(0, 10);
};
```

Hybrid Recommendation Algorithm Implementation

- Combines content-based and collaborative filtering algorithms to generate artist recommendations.

- ContentScore(a, b) = $\begin{cases} +0.5, & \text{if } a_{\text{category}} = b_{\text{category}} \\ +0.2 \times \text{GenreOverlap}(a,b) & \\ +0.2, & \text{if } a_{\text{city}} = b_{\text{city}} \\ +0.1 \times \left(\dfrac{a_{\text{rating}} + b_{\text{rating}}}{2 \times 5} \right) & \end{cases}$

- CollaborativeScore = $\left(\frac{\text{bookings}}{\text{maxBookings}} \times 0.4 \right) + \left(\frac{\text{completed}}{\text{maxCompleted}} \times 0.3 \right) + \left(\frac{\text{rating}}{5} \times \frac{\text{totalRatings}}{\text{maxRatings}} \times 0.3 \right)$

  $\text{FinalScore} = \begin{cases} 0.7 \times \text{ContentScore}, & \text{if using content-based match} \\ 0.3 \times \text{CollaborativeScore}, & \text{if using trending data} \end{cases}$

- Uses client's booking history to find similar artists based on category, genres, location, and ratings (content-based).
- Utilizes global popularity metrics such as booking counts, completed bookings, and weighted ratings to identify trending artists (collaborative filtering).
- For new users without booking history, relies solely on collaborative (trending) recommendations.
- For existing users, blends recommendations with weighted scores: 70% content-based similarity and 30% popularity-based trends.

5.1.2.5 Bayesian Average Rating Algorithm

```
export const updateArtistBayesianRating = async (artistId) => {
  const m = 10;
```

```
const artistStats = await Review.aggregate([
  {
    $match: {
      artistId: new mongoose.Types.ObjectId(String(artistId)),
      status: "confirmed"
    }
  },
  {
    $group: {
      _id: "$artistId",
      avgRating: { $avg: "$rating" },
      totalRatings: { $sum: 1 },
    },
  },
]);

const globalStats = await Review.aggregate([
  { $match: { status: "confirmed" } },
  {
    $group: {
      _id: null,
      globalAvg: { $avg: "$rating" },
    },
  },
]);

const R = artistStats[0]?.avgRating || 0;
const v = artistStats[0]?.totalRatings || 0;
const C = globalStats[0]?.globalAvg || 0;
const WR = ((v / (v + m)) * R) + ((m / (v + m)) * C);

await Artist.findByIdAndUpdate(artistId, {
  weightedRating: WR.toFixed(2),
  rawAverageRating: R.toFixed(2),
```

```
    totalRatings: v,
  });
};
```

Bayesian Average Rating Implementation

- When a new client review is confirmed, the system recalculates the artist's rating using both their own reviews and global data.
- Aggregates the artist's confirmed reviews to compute the artist's average rating, the global average rating.
- Uses a predefined threshold representing the minimum reliable number of reviews.
- Calculates the weighted rating with the formula:

$$\text{WR} = \frac{v}{v+m} \cdot R + \frac{m}{v+m} \cdot C$$

Where:

WR : Weighted Rating (Bayesian average)

R : Average rating of the artist

v : Number of ratings for the artist

m : Minimum number of reviews required (threshold)

C : Mean rating across all artists (global average)

- This formula balances the artist's own rating with the global average, weighting more toward the global average if the artist has few reviews.

5.2 Testing

5.2.1 Test Cases for Unit Testing

Table 5.1: Test Table for User Login

Test 1	
Objective	To test the user login
Action	User login credentials was inserted, and the login was initiated
Expected Result	The user gets into the system and into the user dashboard
Actual Result	The user got into the system and into the user dashboard
Test 2	
Action	Wrong user credentials are inserted, and login was initiated
Expected Result	The user should not get into the system, throw an error
Actual Result	The user did not get into the system, error is displayed
Test Successful	

Table 5.2: Test Table for Client Registration

Test 1	
Objective	To test the Client registration
Action	User registration data was inserted, and the registration was initiated
Expected Result	The user should be redirected to Client Page
Actual Result	The Client was redirected to Client Page
Test 2	
Objective	To test the Client registration with an existing email
Action	User registration data with an existing email was inserted, and the registration was initiated
Expected Result	The system should display an error message indicating that the email is already in use
Actual Result	The system displayed an error message indicating that the email is already in use
Test Successful	

Table 5.3: Test Table for Artist Registration

Test 1	
Objective	To test the Artist registration
Action	User registration data was inserted, and the registration was initiated
Expected Result	The user should be redirected to Artist Home Page
Actual Result	The user was redirected to Artist Home Page
Test 2	
Objective	To test the Artist registration with an existing email
Action	User registration data with an existing email was inserted, and the registration was initiated
Expected Result	The system should display an error message indicating that the email is already in use
Actual Result	The system displayed an error message indicating that the email is already in use
Test Successful	

Table 5.4: Test Table for User Verification

Test 1	
Objective	To test the user verification
Action	Correct Token for verifying user was inserted, and verification was initiated
Expected Result	The user should be redirected to login page with success message displayed
Actual Result	The user was redirected to login page with a success message displayed
Test 2	
Action	Incorrect Token for verifying user was inserted, and verification was initiated
Expected Result	The user gets error message and should retry entering token
Actual Result	The user got an error message and retry was initiated
Test Successful	

Table 5.5: Test Table for Booking Artist

Test 1	
Objective	To test the artist booking functionality
Action	Client selects artist, date, time, location, Event Details and confirms booking
Expected Result	Booking should be saved and confirmation notification should appear
Actual Result	Booking was saved and confirmation notification appeared
Test 2	
Action	Client selects a time slot that is already booked
Expected Result	Booking should be rejected and error should display
Actual Result	Booking was rejected and error message was displayed
Test Successful	

Table 5.6: Test Table for Viewing Bookings

Test 1	
Objective	To test viewing upcoming bookings
Action	Logged in artist views their dashboard
Expected Result	List of upcoming bookings should be shown
Actual Result	List of upcoming booking was shown
Test 2	
Action	Artist filters booking by specific date
Expected Result	Only bookings from that date should be displayed
Actual Result	Only the bookings from the selected date were displayed
Test Successful	

Table 5.7: Test Table for Add Job Post

Test 1	
Objective	To test adding a new job post
Action	Client fills in job title, description, budget and deadline, then submits the form
Expected Result	Job post should be created and success message should be displayed
Actual Result	Job post was successfully created and success message displayed
Test 2	
Action	Client submits job form without entering required fields.
Expected Result	Job should not be submitted and validation error shoud be shown
Actual Result	Form submission was blocked and validation error messages were shown
Test Successful	

Table 5.8: Test Table for View Job Post

Test 1	
Objective	To test viewing posted job
Action	Artist navigates to the job listings page
Expected Result	All active job posts should be listed
Actual Result	List of all active job posts was displayed
Test 2	
Objective	To test viewing jobs by filter
Action	Artist applies filter
Expected Result	Only job posts matching the filter should be shown
Actual Result	Job post matching the filter were displayed
Test 3	
Objective	To test viewing job post detail
Action	Full job details should be displayed
Expected Result	Complete job details were shown successfully
Actual Result	Full job details should be displayed
Test Successful	

Table 5.9: Test Table for real-time Chat

Test 1	
Objective	To test real-time chat between client and artist
Action	Client Sends a message, Artist receives and replies
Expected Result	Both users should see each others messages instantly
Actual Result	Messages were delivered and displayed in real-time
Test Successful	

Table 5.10: Test Table for Contracts

Test 1	
Objective	To test contract creation after booking
Action	System generates a contract automatically after booking is accepted
Expected Result	A contract should be created linking client, artist and job details
Actual Result	Contract was created successfully with correct booking and user details
Test 2	
Objective	To test opening the contract from the booking
Action	User clicks on "View Contract" inside booking details
Expected Result	Full contract terms related to that booking should be shown
Actual Result	Contract opened successfully with correct job and user details
Test Successful	

Table 5.11: Test Table for Payment Processing

Test 1	
Objective	To test successful payment transaction
Action	Client confirms a booking, signs a contract and proceeds to payment using valid payment details
Expected Result	Payment should be processed successfully and confirmation message should appear
Actual Result	Payment was processed successfully and confirmation message was shown
Test 2	
Objective	To test remaining payment success after completion
Action	Client confirms the completion of performance then proceeds to payment using valid payment details
Expected Result	Payment should be processed successfully and confirmation message should appear
Actual Result	Payment successful and confirmation message shown

Table 5.12: Test Table for Admin Dashboard and Reports

Test 1	
Objective	To test admin view of user statistics
Action	Admin login and accesses dashboard
Expected Result	Dashboard should show user stats and booking data
Actual Result	Dashboard displayed user stats and booking data accurately
Test Successful	

5.2.2 Test Cases for System Testing

5.2.2.1 Test Case: Hybrid Artist Recommendation System

To test the proper functioning of the recommendation module, we assume a test case as follows. A new user ClientD is registered with required credentials. Before registering ClientD, We already have three users ClientA, ClientB, ClientC who have booked and rated various artists in the system. These bookings and ratings are used for relevance scoring in the recommendation engine.

Table 5.13: Review Table before Inserting ClientD

S.N	Client ID	Artist Id	Artist Name	Rating
1	ClientA	A2	Priya	5
2	ClientA	A4	Sushant	4
3	ClientB	A2	Priya	4
4	ClientB	A1	Ashma	4
5	ClientC	A3	Roneeshma Shrestha	5
6	ClientC	A7	Sanjay Gupta	4.5
7	ClientB	A6	DJ Chetas	4
8	ClientB	A9	Sushant	4.5
9	ClientC	A8	Elements	5

We also define artist information required for filtering and scoring based on location, category, and price.

Table 5.14: Artist Details

Artist ID	Artist Name	Category	Location	Genre/Type
A1	Ashma	Dancer	Lalitpur	Hip-hop, Folk
A2	Priya	Dancer	Bhaktapur	pop, rock
A3	Roneeshma Shrestha	MC	Lalitpur	Concert, Corporate
A4	Sushant Khatri	Dancer	Lalitpur	Pop
A5	DJWomen	DJ	Bhaktapur	rock
A6	DJ Chetas	DJ	Lalitpur	Hip-hop, Pop
A7	Sanjay Gupta	MC	Bhaktapur	Concert, Corporate, Private
A8	Elements	Band	Kathmandu	Hip-hop, Pop
A9	Karma	Band	Lalitpur	Pop, rock
A10	Albatross	Band	Lalitpur	Hip-hop, rock, pop

After registering ClientD, the user logs into the system. Initially, ClientD receives recommendation using collaborative, as they have not set any preferences or performed bookings. As a result, the homepage recommends the artist that were previously booked by other clients. After ClientD books the following artist.

- **Artist A1 (Ashma)** Category: Dancer, Location: Lalitpur

The system then uses this booking data to show similar artists on the homepage using a **hybrid recommendation system**, which combines:

1. **Content-Based Filtering:**
    - Same or similar **Category** (e.g., Dancer)
    - Same or nearby **Genre**

- Same or nearby **Location** (e.g., Lalitpur)
- Based on **Rating and Review** similarity

2. **Collaborative Filtering:**
    - Based on user behavior such as **bookings count**, **completed count**, and **Rating Score**

Table 5.15: Recommendation Output for ClientD based on booking history

S.N	Artist ID	Artist Name	Score	Category	Location	Genre	Avg. Rating
1	A10	Albatross	0.615	Dancer	Lalitpur	Concert, Festival	3
2	A2	Priya	0.479	Dancer	Bhaktapur	Pop, Rock	4

Result Analysis

Hence, In the Hybrid Artist Recommendation System test case, a new user, ClientD, is registered after three existing users ClientA, ClientB, and ClientC have booked and rated various artists. These prior bookings and ratings form the basis for relevance scoring. When ClientD first logs in, recommendations are primarily collaborative, showing artists previously booked by other clients. After booking Artist A1 (Ashma), a dancer in Lalitpur, the system updates recommendations using a hybrid approach. Content-based filtering considers category, genre, location, and rating/review similarity, while collaborative filtering incorporates bookings, completed events, and rating scores. Each recommended artist is assigned a similarity score, reflecting both attribute relevance and community behavior, providing ClientD with personalized suggestions.

Conclusion: Test Passed

The homepage recommendation module successfully provides relevant artist suggestions based on historical bookings. It uses simple content-based and collaborative filtering by matching artist to the client's past interactions, supporting personalization even without explicit preference input.

CHAPTER 6

CONCLUSION AND FUTURE RECOMMENDATIONS

6.1 Conclusion

The Freelance Artist Booking System successfully addresses the demand for a centralized, efficient,and user-friendly platform that connects artists with clients. Through its intuitive user interface and secure backend, the system enables clients to register, search for artists by category or location, book services, and provide reviews. Artists can create and manage their profiles, handle booking requests, and communicate with clients, while administrators oversee the overall verification process, ensuring compliance and smooth operations.The system is developed using modern technologies such as the MERN stack, the system adheres to industry best practices in terms of performance, security, and scalability.Overall, the platform delivers a comprehensive solution for freelance artist bookings, promoting convenience, transparency, and trust for all users.

6.2 Future Recommendations

As we move forward with the project,there is significant potential for enhancements that could further elevate its usability and market competitiveness :

1. **Mobile Application:** Develop dedicated mobile apps for Android and iOS to provide artists and clients with on-the-go access, real-time notifications, and a smoother user experience.

2. **AI-Powered Recommendations:** Integrate artificial intelligence to personalize artist suggestions for clients based on booking history, preferences, and feedback, thereby improving artist discoverability and booking efficiency.

3. **Geographic Expansion within Nepal:** Extend the platform's availability to additional regions and cities across Nepal, making the service more accessible to both urban and rural users and supporting local talent nationwide.

REFERENCES

[1] Gamege Danushka. Design a freelance application evaluating a full stack for scalable and user centric development. Jamk University of Applied Sciences, Finland, 2025.

[2] Elijah Lowell Calip Pauline Andrea Vivero Eric Blancaflor, Jeanne Bernaldo. Seek n book: A web application for seeking gigs and booking performers. Proceedings of Eighth International Congress on Information and Communication Technology, 2023.

[3] Pierre Michel Menger. Artistic labor markets and careers. Procedia computer science, Volume publication date August 2000.

[4] Andri Sunardi. A comparative study between laravel framework in freelancer project monitoring system web basede. Procedia computer science, 2019.

[5] Samarth Tripathi. Research paper on impact of technology on talent management. *International Journal of Research Publication and Reviews*, 5(5):2113–2117, May 2024. Scholar Amity Business School, Amity University Uttar Pradesh, Lucknow Campus.

APPENDIX

Project User Interface

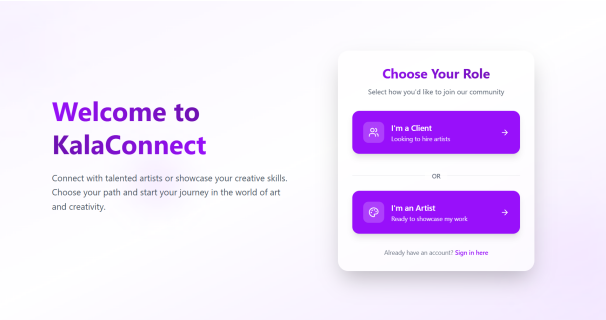

Figure 1: User Role Selection

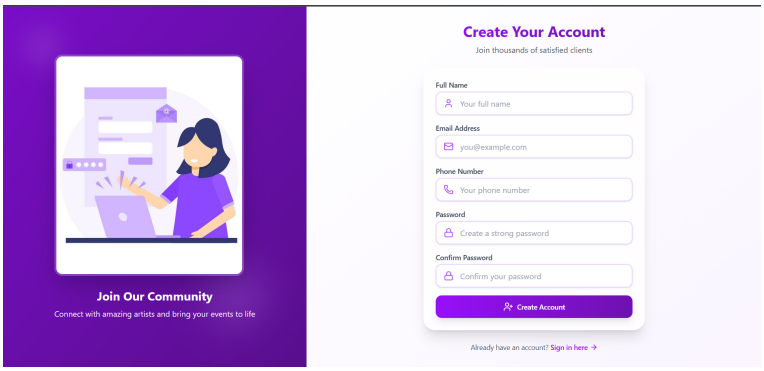

Figure 2: Register UI

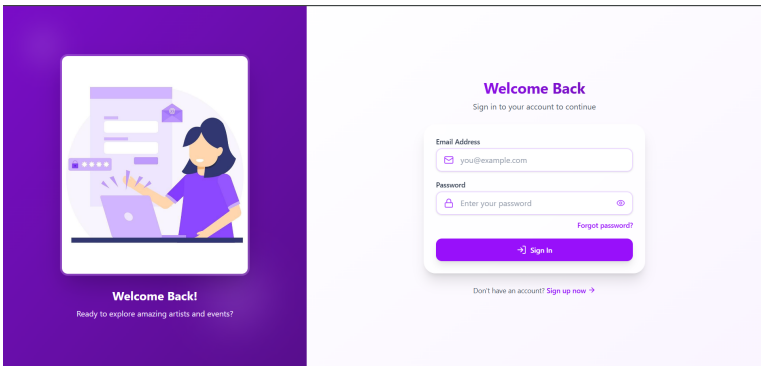

Figure 3: Login UI

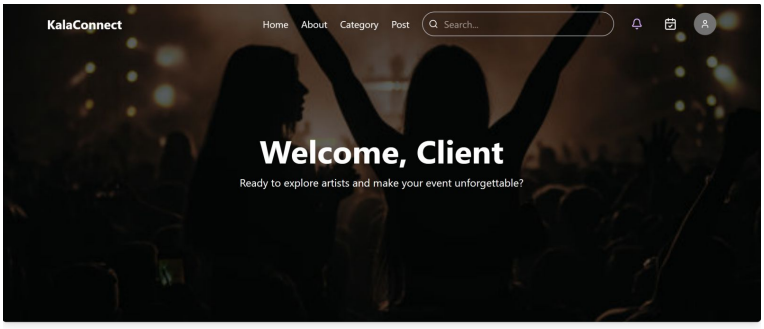

Figure 4: HomePage

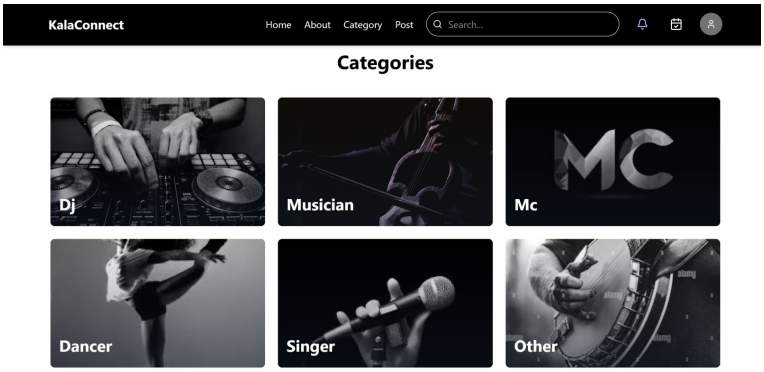

Figure 5: Category

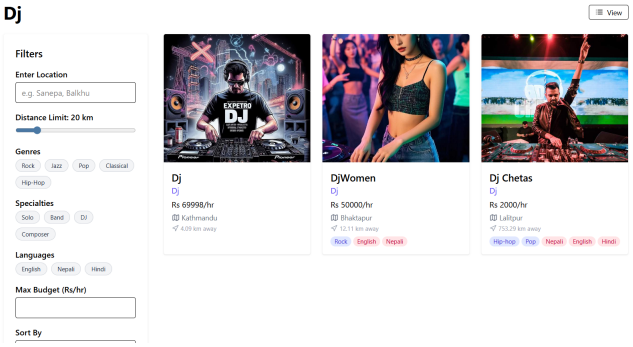

Figure 6: Artist List

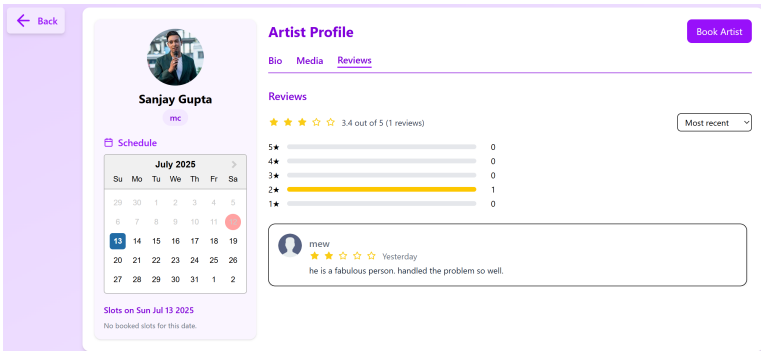

Figure 7: Artist profile page

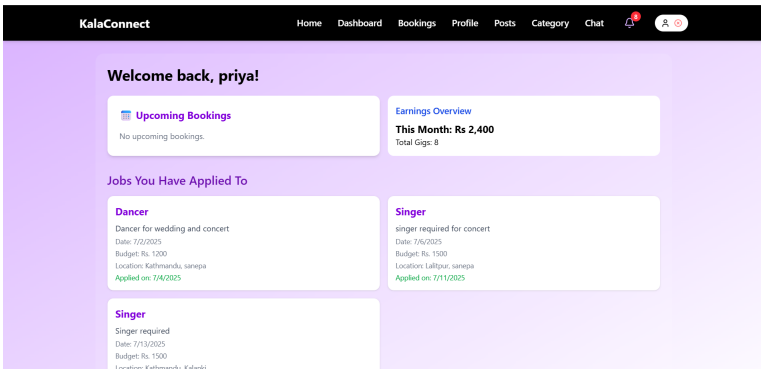

Figure 8: Artist Dashboard

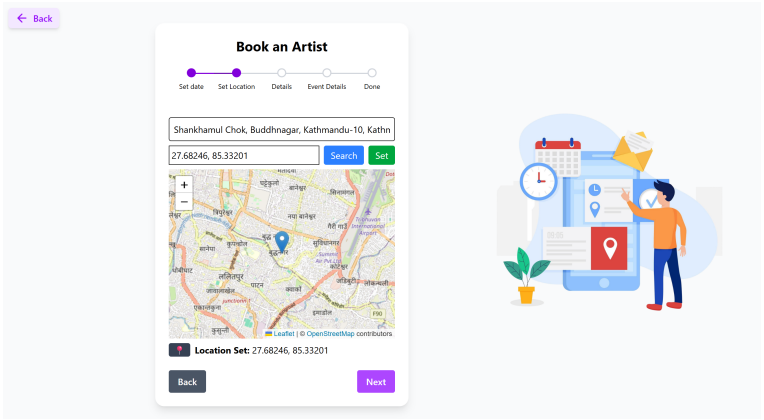

Figure 9: Booking

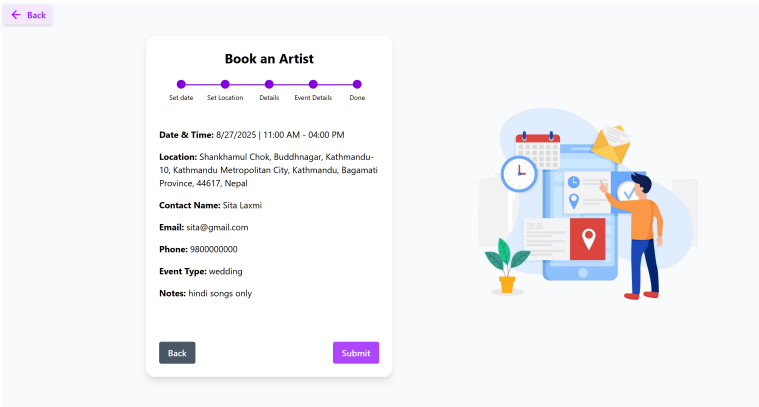

Figure 10: Booking

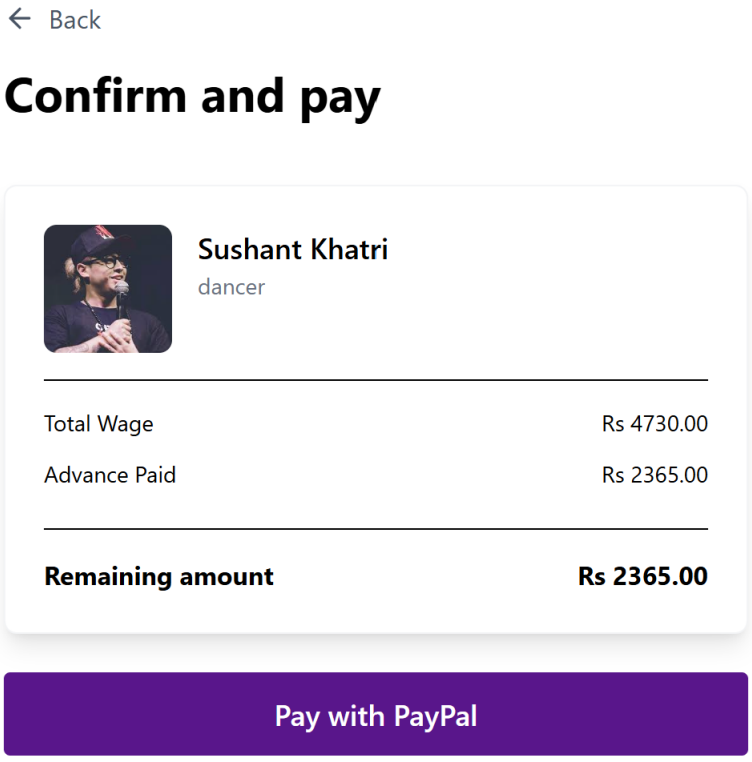

Figure 11: Payment

Project Admin Interface

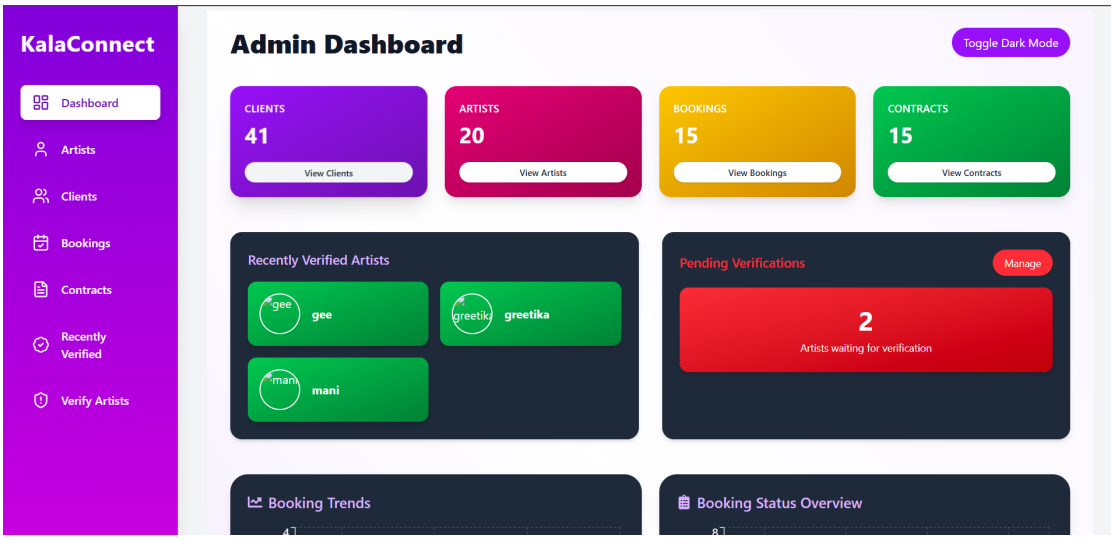

Figure 12: Admin Dashboard Overview UI

Sagarmatha College of Science and Technology

Sanepa, Lalitpur

Supervisor Name: **Manish Aryal**

PROJECT LOGBOOK

Date	**Meeting Discussion**	**Supervisor Signed**	**Remarks**
24th Baisakh, 2082	Project title discussion, Method of approachable model for development		
25th Baisakh, 2082	Reviewed project proposal and suggestion on requirement analysis		
3rd Jestha, 2082	Discussion on algorithms		
3rd Ashar, 2082	System Analysis and System Design		
16th Ashar, 2082	Updated about the system progress and discussion on additional feature implementation		
26th Ashar, 2082	Discussion on User Validations, template model and payment integration		
28th Ashar, 2082	Update on project		
29th Ashar, 2082	Report sent and reviewed through email		
30th Ashar, 2082	Final internal report signature		