DREAM KNOT

Wedding Planning Management System

Mini Project Report

Submitted by

Fathima P.S

Reg. No.: AJC23MCA-2027

In Partial fulfillment for the Award of the Degree of

MASTER OF COMPUTER APPLICATIONS (MCA)

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

[Approved by AICTE, Accredited by NAAC, Accredited by NBA.
Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2024-2025

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report, "**DREAM KNOT"** is the bonafide work of **FATHIMA P.S (Regno: AJC23MCA-2027)** in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications under **Amal Jyothi College of Engineering Autonomous, Kanjirappally** during the year 2024-25.

Mr. T.J Jobin
Internal Guide

Mr. Binumon Joseph
Coordinator

Rev. Fr. Dr. Rubin Thottupurathu Jose
Head of the Department

DECLARATION

I hereby declare that the project report **"DREAM KNOT"** is a bona fide work done at Amal Jyothi College of Engineering, towards the partial fulfilment of the requirements for the award of the Master of Computer Applications (MCA) from Amal Jyothi College of Engineering Autonomous during the academic year 2024-2025.

Date: 07/11/2024 **FATHIMA P.S**
KANJIRAPPALLY **Reg: AJC23MCA-2027**

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Director (Administration) **Rev. Fr. Dr. Roy Abraham Pazhayaparampil** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude towards our Head of the Department **Rev. Fr. Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Mr. Binumon Joseph** for his valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Mr. T J Jobin** for his inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

FATHIMA P.S

ABSTRACT

Dream Knot is an innovative web-based application designed to streamline the wedding planning process by connecting couples with various wedding service providers. This comprehensive platform offers a suite of features to simplify the complex task of organizing a wedding. *Dream Knot* features an easy-to-use interface, customizable options, and robust data security to meet the needs of both couples and wedding professionals. With *Dream Knot*, couples can bring their dream wedding to life, and planners can efficiently handle multiple events while providing exceptional client service.

Dream Knot offers a comprehensive suite of features designed to simplify the wedding planning process. The platform supports multiple user roles couples, vendors, and administrators each with tailored functionalities to enhance their experience. Couples benefit from a personalized wedding timeline, ensuring they stay on track as their big day approaches, while a customizable task management system helps them organize predefined and custom tasks. The vendor marketplace allows couples to easily search, filter, and book wedding service providers, while the integrated e- Invitations helps to send invitation to guests. Additionally, couples can manage their bookings directly through the platform, and a review and rating system encourages trust by allowing users to rate vendors after their wedding. Administrators have access to a powerful dashboard to oversee user management, monitor platform activities, and ensure system integrity, making Dream Knot an all-in-one solution for wedding planning.

Dream Knot leverages modern web technologies, including Django framework and MYSQL, to deliver a responsive and secure user experience. The platform's modular design allows for easy scalability and feature expansion, making it adaptable to evolving wedding planning trends and user needs.

By centralizing various aspects of wedding planning into a single, user-friendly platform, Dream Knot aims to reduce the stress and complexity typically associated with wedding preparations, offering a valuable tool for couples, vendors, and wedding professionals alike.

CONTENT

List of Abbreviations

• UML - Unified Modelling Language
• ORM - Object-Relational Mapping
• MVT - Model-View-Template
• MVC - Model-View-Controller
• RDBMS - Relational Database Management System
• 1NF - First Normal Form
• 2NF - Second Normal Form
• 3NF - Third Normal Form
• IDE - Integrated Development Environment
• HTML - Hypertext Markup Language
• JS - JavaScript
• CSS - Cascading Style Sheets
• AJAX - Asynchronous JavaScript and XML
• JSON - JavaScript Object Notation
• API - Application Programming Interface
• UI - User Interface
• HTTP - Hypertext Transfer Protocol
• URL - Uniform Resource Locator
• PK - Primary Key
• FK - Foreign Key
• SQL - Structured Query Language
• CRUD - Create, Read, Update, Delete

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

Dream Knot is a comprehensive and innovative web-based platform designed to transform the wedding planning experience. By offering an all-in-one solution for couples, wedding vendors, and administrators, Dream Knot centralizes the entire wedding planning process. The platform enables users to manage tasks, connect with service providers, handle RSVPs, book vendors, and much more, all through a secure and user-friendly interface.

The platform caters to the needs of couples by providing personalized planning tools and a streamlined process to discover, book, and review vendors. Vendors, on the other hand, benefit from a marketplace where they can showcase their services, manage bookings, and receive feedback from customers. Additionally, administrators have access to an advanced dashboard that allows them to manage users and oversee platform activities, ensuring smooth operation and security.

Key Roles:

- Couples: Manage guest lists, book services, track tasks, and plan their weddings with ease.
- Service Providers: Showcase services (e.g., catering, makeup, decoration), update availability, and handle bookings.
- Administrators: Ensure the smooth operation of the platform, oversee user activities, and maintain data integrity.

1.2 PROJECT SPECIFICATION

Dream Knot aims to simplify wedding planning by offering a unified platform that integrates various tools and services into a seamless experience. It is designed to:

- Help couples manage their wedding preparations.
- Connect wedding vendors with potential clients.
- Provide administrative tools to manage the platform's operations.

Key Features: a. **User Management:**

- Multi-role system: Supports couples, vendors, and administrators.
- Secure authentication with email verification.
- Detailed profile management for both users and vendors.

b. **Wedding Planning Tools:**

- Wedding Countdown: Personalized countdown to the wedding day.
- Task Management: Customizable wedding-related to-do lists.
- E-Invitation: Efficient guest list and invitation management.

c. **Vendor Marketplace:**

- Searchable and filterable directory of wedding service providers.
- Comprehensive service listings with images, descriptions, and pricing.
- Vendor booking management system with calendar integrations.

d. **User Interaction:**

- Rating and review system for vendor services.
- Favorite vendors tracking.
- Option for direct messaging between couples and vendors (future feature).

e. **Admin Functions:**

- User account management: Activation, deactivation, and permission settings.
- Vendor and service oversight.
- Platform analytics, reporting, and monitoring tools.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

System analysis is a critical initial phase in the development of the Dream Knot platform, as it lays the foundation for understanding the system's goals, challenges, and solutions. The main objective of this stage is to gather and evaluate data comprehensively to identify issues in the current wedding planning process and propose practical solutions. Effective communication between system users—such as engaged couples, wedding planners, vendors, and administrators—and developers is central to this phase, ensuring that the platform will meet user needs and expectations.

In the system analysis phase, the system analyst acts as an investigator, studying the effectiveness of current wedding planning methods, whether through traditional means or other digital solutions, and identifying areas for improvement. This involves examining the inputs (such as vendor data, booking details, and user requirements) and outputs (such as booking confirmations, task reminders, and vendor profiles) and understanding the relationship between activities and the outcomes that Dream Knot aims to achieve.System analysis provides a clear roadmap for the development of Dream Knot, ensuring that every feature—from the booking system to the event management and planning tools—addresses actual user pain points and enhances the overall user experience.

2.2 EXISTING SYSTEM

2.2.1 NATURAL SYSTEM STUDIED

A natural system in this context refers to the traditional or organically evolved methods people use to plan weddings. This includes the following:

- Manual Planning: Couples often create paper checklists, spreadsheets, or handwritten notes to manage tasks, timelines, and budgets. Physical or digital calendars are used to track deadlines, while email or phone communication is the primary way of contacting vendors and guests.
- Vendor Discovery: Couples typically search for vendors through personal recommendations, online reviews, and social media platforms. However, this process is unorganized, and there is no single platform to compare multiple vendors, manage bookings, or handle payments in one place.
- Task and Timeline Management: Couples manage their wedding timelines and task lists

manually using either a notebook or general-purpose digital tools like Trello or Asana. These tools are not optimized for wedding-specific needs and may miss important aspects, such as automatic reminders for upcoming tasks.

- Payment and Booking: Payments are often made directly between couples and vendors via bank transfers, checks, or cash, leading to a lack of centralized tracking for both parties. Booking details and contracts are managed through email exchanges or paper contracts, which can become difficult to organize.

The natural systems studied highlight the disjointed and manual processes in traditional wedding planning. While each method may work on its own, there is a clear lack of integration, leading to duplicated efforts, missed deadlines, and stress for both couples and vendors

2.2.2 DESIGNED SYSTEM STUDIED

The **designed system** refers to the planned solution for addressing the inefficiencies of the natural system by integrating all wedding planning functions into a single, user-friendly platform—Dream Knot. The goal is to replace the fragmented tools and manual processes with a streamlined, comprehensive system that connects couples with vendors, simplifies communication, manages tasks, and automates guest tracking and booking.

2.3 DRAWBACKS OF EXISTING SYSTEM

- Fragmented Planning Tools: Couples rely on multiple platforms for task management, budgeting, and vendor communication, resulting in a disorganized process.
- Manual Task Tracking: Wedding tasks and timelines are often tracked manually through notebooks or general-purpose apps, leading to missed deadlines or duplicated efforts.
- Inefficient Vendor Discovery: The current vendor discovery process is time-consuming, relying on personal recommendations, scattered online reviews, and social media. There is no centralized platform for easy vendor comparison and booking.
- No Centralized Digital Invitation System: Invitations are often managed manually or through non-integrated digital tools.
- No Integrated Payment System: Payments to vendors are handled through separate channels, often via bank transfers or cash, without a central system to track transactions or manage contracts.

- Lack of Vendor Accountability: Without a centralized review system, it's difficult for couples to assess vendor reliability and service quality in advance.
- Stressful Coordination: The reliance on multiple disconnected tools and manual processes increases stress for couples, leading to potential oversights and delays in planning.

2.4 PROPOSED SYSTEM

The **proposed system**—Dream Knot—is a centralized, web-based wedding planning platform designed to address the inefficiencies of traditional planning processes. It integrates various tools and features into one comprehensive platform, allowing couples to manage every aspect of their wedding planning in a structured and streamlined manner.

Key features of the proposed system:

- User Role Management: Customizable dashboards for couples, vendors, and administrators, with features tailored to each user type.
- Digital Invitation Management: Instead of traditional RSVPs, Dream Knot offers an integrated digital invitation system, allowing couples to send and manage guest invitations seamlessly within the platform.
- Vendor Marketplace: A searchable and filterable directory of wedding vendors, allowing couples to easily discover, compare, and book services from verified providers.
- Task and Timeline Management: Offers a wedding-specific task management tool, providing automatic reminders and customizable to-do lists based on key milestones.
- Booking and Payment Integration: Couples can book services directly through the platform and use integrated payment gateways to manage secure transactions with vendors.
- Review and Rating System: Allows couples to leave feedback on vendors, fostering a trustworthy and accountable vendor ecosystem.
- Admin Dashboard: Administrators have access to tools for managing platform users, monitoring system performance, and overseeing vendor listings.

2.5 ADVANTAGES OF PROPOSED SYSTEM

- All-in-One Platform: Dream Knot consolidates all wedding planning tools, vendor management, and guest communication into one place, reducing the need for multiple disconnected platforms.
- Efficient Digital Invitations: Couples can manage digital invitations directly on the platform, streamlining guest communication and tracking responses in real-time.
- Simplified Vendor Discovery: The vendor marketplace allows couples to easily search for vendors based on preferences and budget, with detailed service descriptions and reviews to make informed decisions.
- Centralized Task Management: The customizable task list and timeline system ensures that couples stay on track with their wedding plans, with reminders for upcoming tasks and deadlines.
- Seamless Booking and Payment: Integrated booking and payment features eliminate the need for external payment methods, simplifying financial transactions and providing a clear record of all bookings.
- Trustworthy Vendor Ecosystem: With a built-in review and rating system, couples can rely on verified feedback from other users, promoting high-quality service among vendors.
- Improved Communication: All vendor interactions and bookings are managed within the platform, reducing the need for fragmented communication methods like emails and phone calls.
- Enhanced Platform Control: Administrators have a comprehensive view of platform activities, ensuring smooth operation, resolving issues, and maintaining system integrity.
- Availability Checking: The platform provides real-time availability for vendors, allowing couples to confirm whether their preferred services are available for their chosen wedding dates before booking.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

The primary purpose of conducting a feasibility study is to comprehensively assess whether the proposed project can successfully meet the organization's objectives in terms of available resources, labor, and time. This crucial study allows the developers and decision-makers to gain valuable insights into the project's potential viability and prospects. By carefully examining various aspects of the proposed system, such as its impact on the organization, its ability to fulfill user requirements, and the optimal utilization of resources, a feasibility study helps in determining the project's feasibility and potential success.

The assessment of the proposed project's feasibility involves multiple dimensions, each playing a critical role in the decision-making process.

- Technical Feasibility
- Operational Feasibility
- Economic Feasibility

A well-conducted feasibility study provides valuable insights to decision-makers, allowing them to make informed judgments about the project's potential success. It assists in identifying potential risks, challenges, and opportunities associated with the proposed endeavor, enabling stakeholders to devise effective mitigation strategies

3.1.1 Economical Feasibility

Economic feasibility assesses the financial implications of the proposed system to ascertain if the benefits will outweigh the costs.

Cost of Development is a crucial factor, involving initial expenses such as software development, hosting services, and domain registration. Additionally, ongoing operational costs, including server maintenance, updates, and user support, must be considered.

On the revenue generation side, Dream Knot can explore multiple streams. Subscription fees can be charged to vendors for listing their services, providing them with visibility to engaged couples. Additionally, the platform can implement commissions on bookings made through it, creating a steady revenue stream. Advertising revenue can also be pursued by collaborating with other wedding-related businesses that wish to promote their services on the platform.

3.1.2 Technical Feasibility

Technical feasibility evaluates whether the necessary technology to implement the system is available and if the development team possesses the requisite skills.

The proposed technology stack includes Python and the Django framework for backend development, while PostgreSQL will be used for database management. For the frontend, technologies like HTML, CSS, and JavaScript will be utilized to create a responsive and engaging user interface.

The infrastructure for hosting the application can leverage cloud services, ensuring scalability and reliability. Security measures will also be paramount, particularly in protecting user data and financial transactions.

Evaluating the development team's skills is crucial for successful implementation. The team should consist of skilled developers experienced in the chosen technologies and proficient in project management methodologies, such as Agile, to facilitate effective collaboration.

Another critical factor is the system's integration capabilities. The platform must seamlessly integrate with payment gateways, communication tools, and other third-party services to enhance functionality and user experience.

3.1.3 Behavioral Feasibility

Assessing the behavioral feasibility of Dream Knot involves evaluating how well the platform aligns with the needs of engaged couples, wedding vendors, and administrators. We establish a clear outline of system requirements, focusing on user needs, functional requirements, and the necessary inputs and outputs. Identifying key functionalities—such as user role management, vendor marketplace, and booking integration—is essential for meeting user expectations. We also detail the programs and procedures required for smooth operation, alongside the necessary technical infrastructure to support the platform. Once the design is finalized, we prioritize user-friendly interfaces and develop support materials, like tutorials, to aid adaptation. Conducting user testing sessions will gather feedback from couples and vendors to refine the platform further. This feedback will ensure that Dream Knot meets user expectations and provides a positive wedding planning experience. Ultimately, by addressing these aspects, we can confirm that Dream Knot is designed with user needs at its core.

3.1.4 Feasibility Study Questionnaire

- How do you prioritize tasks during wedding planning?

  Answer: We use a detailed task management system to categorize tasks by priority (high, medium, low) and deadline. Regular meetings with the couple ensure that their most important needs are met first.Managing

- What strategies do you use to manage multiple marriages at once?

  Answer: We use project management software that allows us to create separate projects for each wedding. This helps to track progress, assign tasks, and ensure no detail is overlooked. It is also essential to assign responsibilities to team members based on their expertise

- How do you ensure effective communication between the couple, vendors, and planners?

  Answer: We use a built-in messaging system in our scheduling software to centralize all communications. Regular updates and scheduled check-ins with couples and vendors help keep communication clear and consistent.

- What are the key metrics you track to measure the success of a wedding event?

  Answer: Key metrics include client satisfaction scores, budget adherence, timely completion of tasks, and vendor performance. Post-event client feedback is also a key metric.

- How do you handle last-minute changes or emergencies during wedding events?

  We have a contingency plan for each event, including backup vendors and flexible scheduling. Effective communication and a calm, proactive approach help to resolve last-minute issues smoothly.

- What is your approach to budgeting and managing wedding expenses?

  Answer: We use budget tracking software to monitor expenses in real-time. This includes setting budget limits for different categories and tracking actual spending against the planned budget. Regular financial reviews with our spouse help to control spending.

- How do you select and evaluate wedding vendors?

  Answer: Suppliers are selected based on reputation, past performance and customer reviews. We conduct thorough assessments, including site visits and sample reviews, to ensure they meet our standards.

- What tools or software do you currently use for wedding planning?

Answer: We currently use a combination of project management tools (like Trello or Asana), budget tracking software (like Excel or dedicated wedding planning apps), and communication platforms (like Slack or integrated messaging systems).

- How do you manage the expectations and preferences of the couple and their families?

  Answer: It is essential to understand the couple's vision and preferences. We conduct a detailed initial consultation and create a mood board or concept presentation to match expectations. Regular updates and flexibility in adapting to changes also help manage expectations

- What feedback mechanisms have you implemented to improve your wedding planning services?.

  Answer: Post-event surveys and feedback are sent to customers to gather their opinions. We also hold debriefing sessions with our team to discuss what went well and what could be improved. Continuous feedback loops help us improve our service.

3.1 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor - Intel i7

RAM - 16.0 GB

Hard disk - 500 GB

3.2.2 Software Specification

Front End	- HTML, JS, CSS, jQuery, Ajax, Bootstrap
Back End	- Python-Django
Database	- Sqlite
Client on PC	- Windows 7 and above.
Technologies used	- JS, HTML5, AJAX, jQuery, CSS

3.3 SOFTWARE DESCRIPTION

3.3.1 DJANGO

Django is a popular and powerful open-source web framework written in Python, designed to facilitate rapid development and maintainable web applications. It follows the Model-View Template (MVT) architectural pattern, which is similar to the Model-View-Controller (MVC) pattern. Django provides a structured and efficient way to build web applications, offering several key components and features.

At its core, Django includes a robust Object-Relational Mapping (ORM) system that simplifies database interactions, allowing developers to work with Python objects instead of raw SQL queries. It also includes a URL dispatcher for mapping URLs to view functions, an automatic admin interface for managing application data, and a templating engine for creating dynamic and reusable user interfaces.

Django places a strong emphasis on security, with built-in features to protect against common web vulnerabilities. It offers authentication and authorization systems, middleware support for global request and response processing, and compatibility with various databases.

The framework's scalability, extensibility, and a vibrant community of developers make it a popular choice for building web applications, from simple websites to complex, high-traffic platforms. Django's extensive documentation and ecosystem of third-party packages further enhance its appeal for web developers.

3.3.2 MySQL

MySQL is an open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) for accessing and managing the data. It is widely used for web applications and is a crucial part of many software stacks, including the popular LAMP (Linux, Apache, MySQL, PHP) stack. MySQL is known for its reliability, speed, and flexibility, making it an excellent choice for various applications, including Dream Knot.

CHAPTER 4

SYSTEM DESIGN

4.1INTRODUCTION

The initial stage of developing any engineered product or system is the design phase, which involves a creative approach. A well-crafted design plays a critical role in ensuring the successful functioning of a system. Design is defined as the process of employing various techniques and principles to define a process or system in enough detail to enable its physical realization. This involves using different methods to describe a machine or system, explaining how it operates, in sufficient detail for its creation. In software development, design is a crucial step that is always present, regardless of the development approach. System design involves creating a blueprint for building a machine or product. Careful software design is essential to ensure optimal performance and accuracy. During the design phase, the focus shifts from the user to the programmers or those working with the database. The process of creating a system typically involves two key steps: Logical Design and Physical Design.

4.2UML DIAGRAM

UML, which stands for Unified Modeling Language, is a standardized language used for specifying, visualizing, constructing, and documenting the elements of software systems. The Object Management Group (OMG) is responsible for the creation of UML, with the initial draft of the UML 1.0 specification presented to OMG in January 1997. Unlike common programming languages such as C++, Java, or COBOL, UML is not a programming language itself. Instead, it is a graphical language that serves as a tool for creating software blueprints.

UML is a versatile and general-purpose visual modeling language that facilitates the visualization, specification, construction, and documentation of software systems. While its primary use is in modeling software systems, UML's applications are not limited to this domain. It can also be employed to represent and understand processes in various contexts, including non-software scenarios like manufacturing unit processes.

It's important to note that UML is not a programming language, but it can be utilized with tools that generate code in different programming languages based on UML diagrams. UML encompasses nine core diagrams that aid in representing various aspects of a system.

- Class diagram
- Object diagram
- Activity diagram
- State chart diagram
- Deployment diagram
- Component diagram

4.2.1 USE CASE DIAGRAM

A use case is a tool for understanding a system's requirements and organizing them, especially in the context of creating or using something like a product delivery website. These tools are represented using "use case" diagrams within the Unified Modeling Language, a standardized way of creating models for real-world things and systems.

A use case diagram consists of these main elements:

- The boundary, which delineates the system and distinguishes it from its surroundings.
- Actors, representing individuals or entities playing specific roles within the system.
- The interactions between different people or elements in specific scenarios or problems.
- The primary purpose of use case diagrams is to document a system's functional specifications. To create an effective use case diagram, certain guidelines must be followed:
- Providing clear and meaningful names for use cases and actors.
- Ensuring that the relationships and dependencies are well-defined.
- Including only necessary relationships for the diagram's clarity.
- Utilizing explanatory notes when needed to clarify essential details.

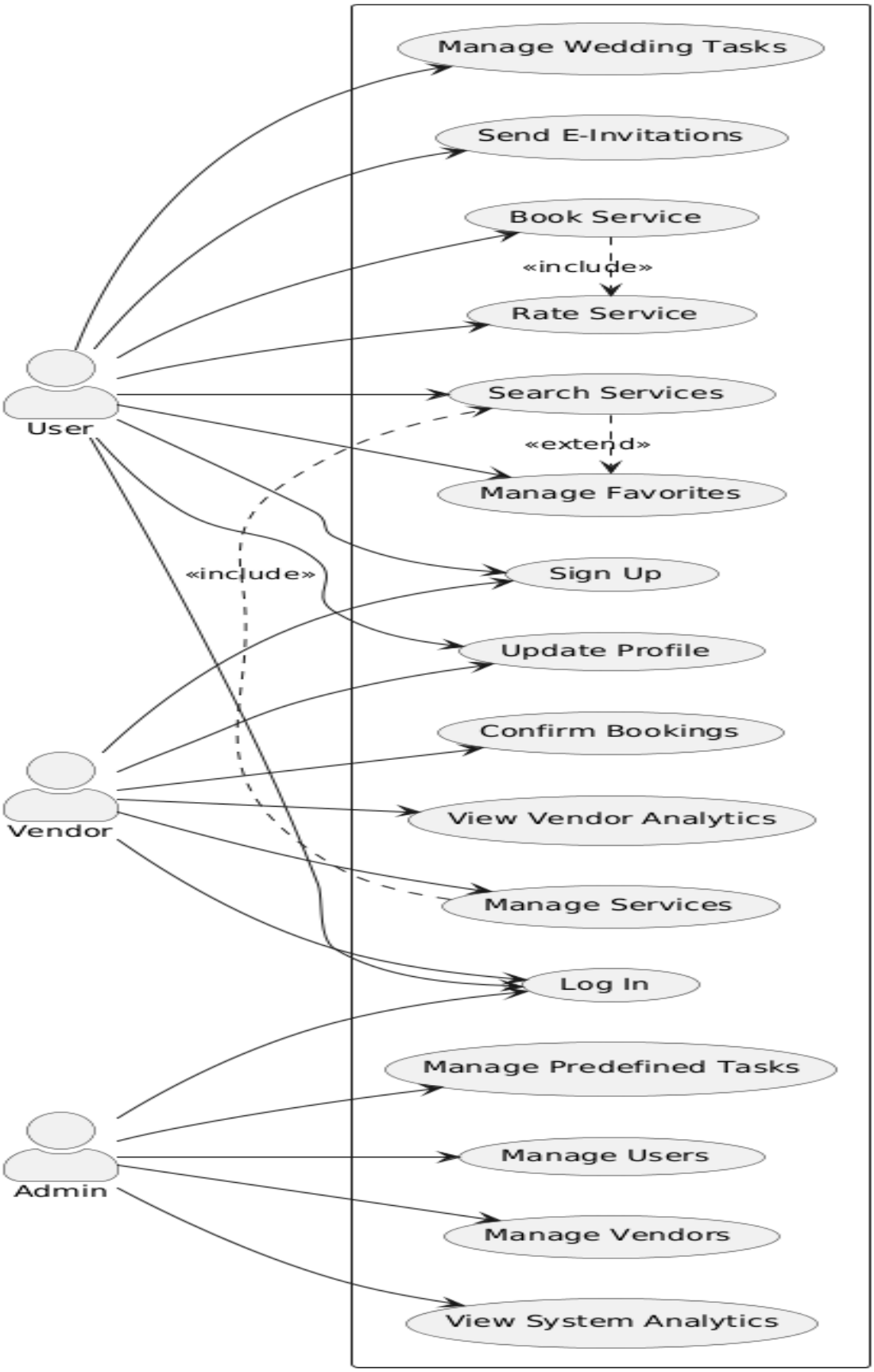


Fig 4.2.1: Use Case Diagram

4.2.2 SEQUENCE DIAGRAM

A sequence diagram illustrates the specific order in which objects interact with each other, showcasing the sequential flow of events. This type of diagram is also known as event diagrams or event scenarios. Sequence diagrams serve the purpose of elucidating how various components of a system collaborate and the precise sequence in which these actions occur. These diagrams find frequent application among business professionals and software developers, aiding in the understanding and depiction of requirements for both new and existing systems.

Sequence Diagram Notations:

i. Actors - Within a UML diagram, actors represent individuals who utilize the system and its components. Actors are not depicted within the UML diagram as they exist outside the system being modeled. They serve as role-players in a story, encompassing people and external entities. In a UML diagram, actors are represented by simple stick figures. It's possible to depict multiple individuals in a diagram that portrays the sequential progression of events.

ii. Lifelines - In a sequence diagram, each element is presented as a lifeline, with lifeline components positioned at the top of the diagram.

iii. Messages - Communication between objects is achieved through the exchange of messages, with messages being arranged sequentially on the lifeline. Arrows are employed to represent messages, forming the core structure of a sequence diagram.

iv. Guards - Within the UML, guards are employed to denote various conditions. They are used to restrict messages in the event that specific conditions are met, providing software developers with insights into the rules governing a system or process.

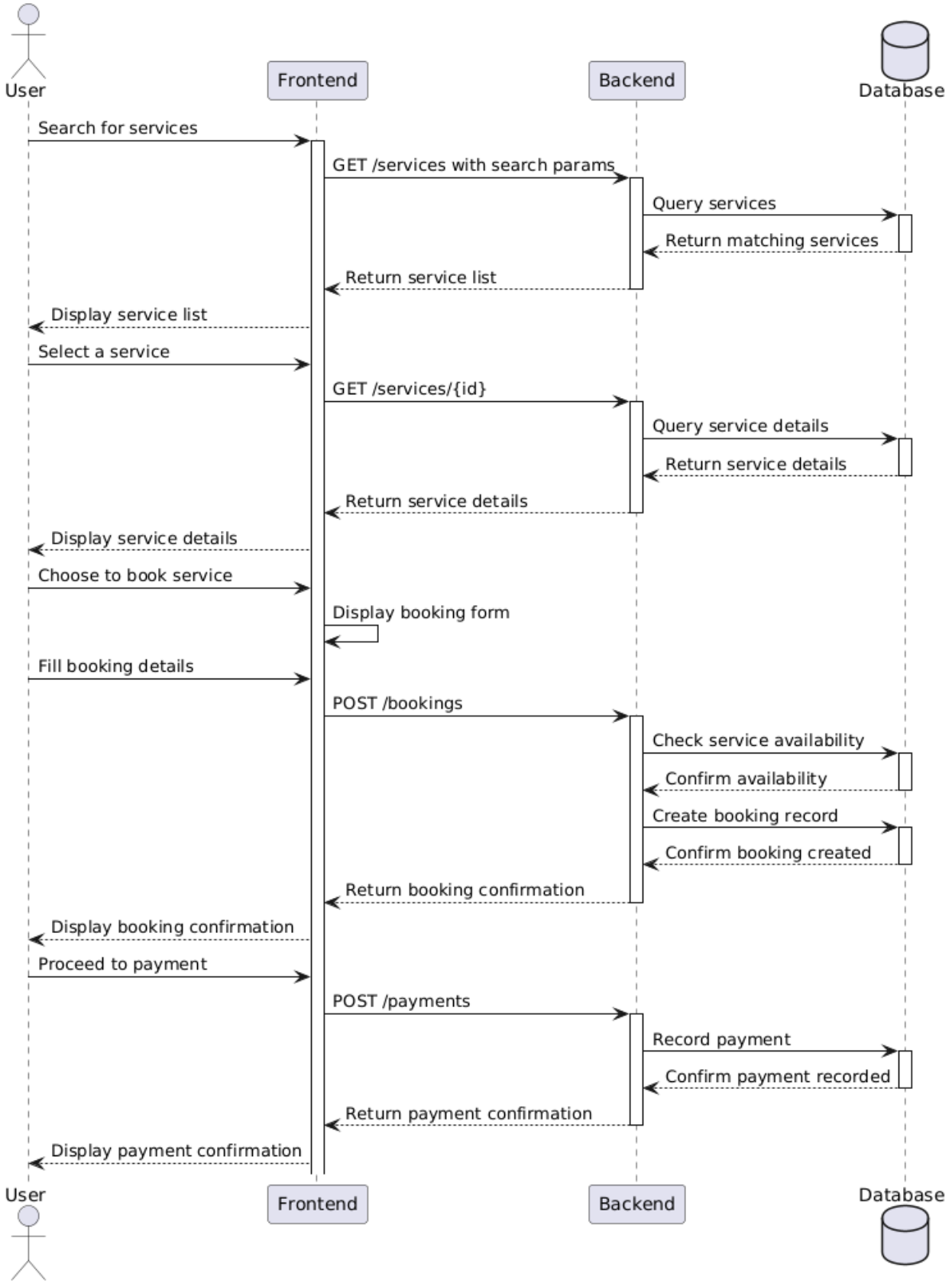


Fig4.2.2: Sequence Diagram

4.2.3 State Chart Diagram

A state machine diagram, also known as a state chart, visually represents the various states an object undergoes within a system and the sequence in which these states are traversed. It serves as a record of the system's behavior, illustrating the collaborative functioning of a group of entities, whether it's a team, a collection of students, a large assembly, or an entire organization. State machine diagrams are a valuable method for depicting the interactions of diverse components within a system, outlining how objects evolve in response to events and elucidating the diverse conditions that each entity or component can inhabit.

Notations within a state machine diagram encompass:

- Initial state: Symbolized by a black circle, this signifies the commencement of a process.
- Final state: Representing the conclusion of a process, this is denoted by a filled circle within another circle.
- Decision box: Shaped like a diamond, it aids in decision-making by considering the evaluation of a guard.
- Transition: Whenever a change in authority or state occurs due to an event, it is termed a transition. Transitions are depicted as arrows with labels indicating the triggering event.
- State box: It portrays the condition or state of an element within a group at a specific moment. These are typically represented by rectangles with rounded corners.

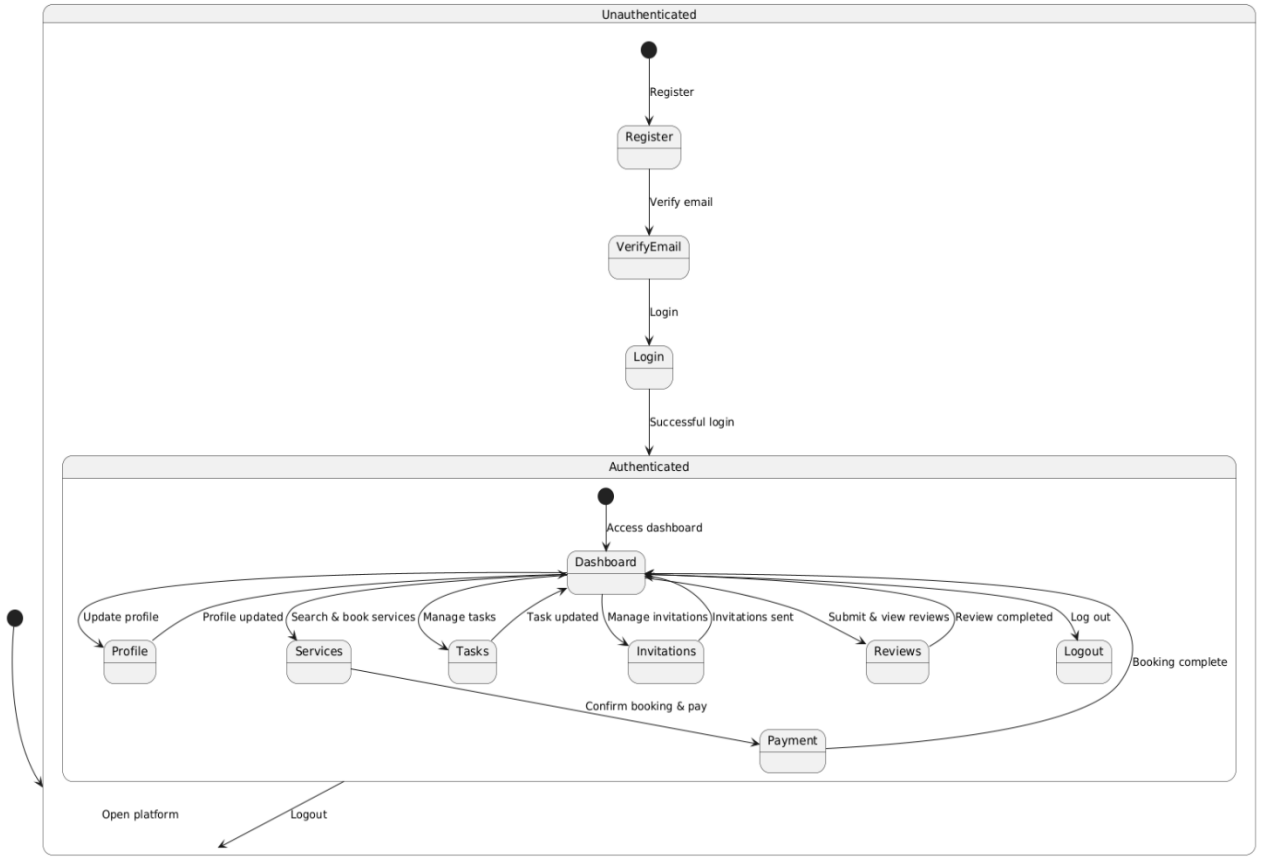


Fig4.2.3: State Chart Diagram

4.2.4 Activity Diagram

An activity diagram is a visual representation of how events unfold simultaneously or sequentially. It aids in understanding the flow of activities, emphasizing the progression from one task to the next. Activity diagrams focus on the order in which tasks occur and can depict various types of flows, including sequential, parallel, and alternative paths. To facilitate these flows, activity diagrams incorporate elements such as forks and join nodes, aligning with the concept of illustrating the functioning of a system in a specific manner.

Key Components of an Activity Diagram include:

a) Activities: Activities group behaviors into one or more actions, forming a network of interconnected steps. Lines between these actions outline the step-by-step sequence of events. Actions encompass tasks, controlling elements, and resources utilized in the process.

b) Activity Partition/Swim Lane: Swim lanes are used to categorize similar tasks into rows or columns, enhancing modularity in the activity diagram. They can be arranged vertically or horizontally, though they are not mandatory for every activity diagram.

c) Forks: Fork nodes enable the simultaneous execution of different segments of a task. They represent a point where one input transforms into multiple outputs, resembling the diverse factors influencing a decision.

d) Join Nodes: Join nodes are distinct from fork nodes and employ a Logical AND operation to ensure that all incoming data flows converge to a single point, promoting synchronization.

Notations in an Activity Diagram include:

- Initial State: Depicting the beginning or first step in the process.

- Final State: Signifying the completion of all actions with no further progress.

- Decision Box: Ensuring that activities follow a specific path.

- Action Box: Representing the specific tasks or actions to be performed in the process.

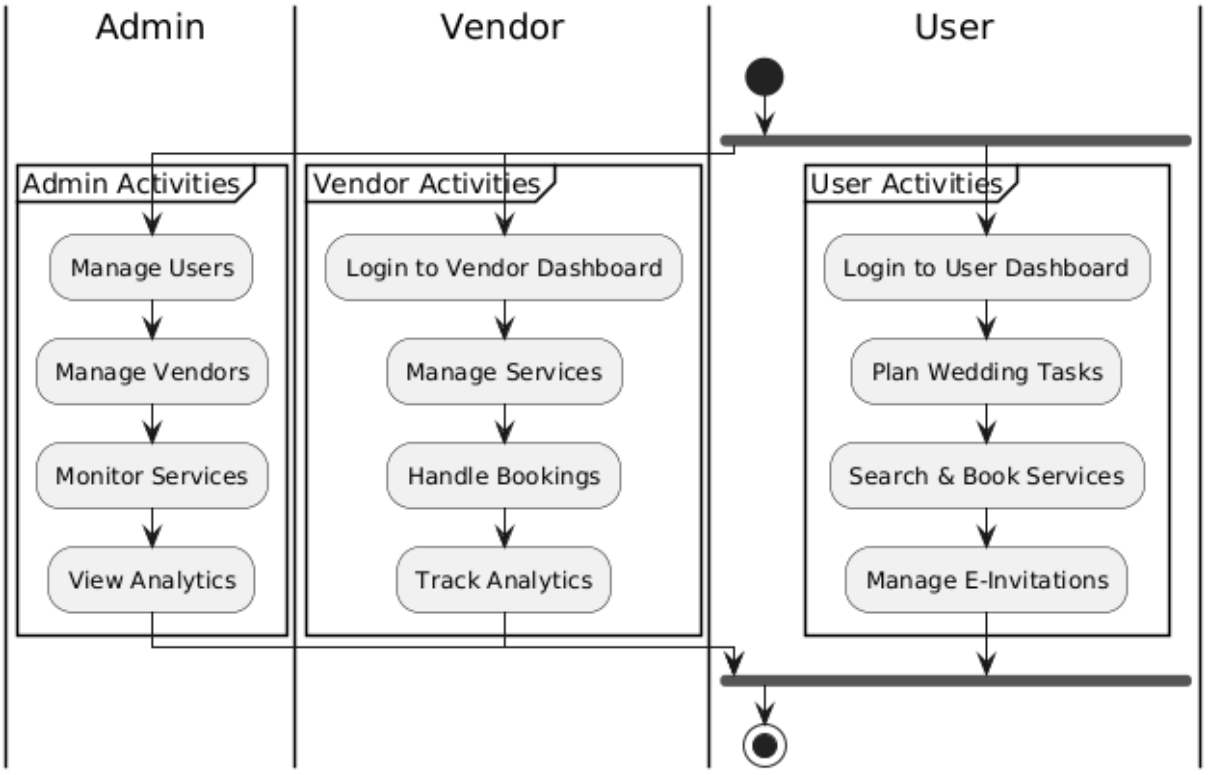


Fig 4.2.4: Activity Diagram

4.2.5 Class Diagram

A class diagram serves as a static blueprint for an application, illustrating its components and their relationships when the system is in a dormant state. It provides insights into the system's structure, showcasing the various elements it comprises and how they interact. In essence, a class diagram acts as a visual guide for software development, aiding in the creation of functional applications.

Key Aspects of a Class Diagram:

- System Overview: A class diagram offers a high-level representation of a software system, presenting its constituent parts, associations, and collaborative dynamics. It serves as an organizational framework for elements such as names, attributes, and methods, simplifying the software development process.
- Structural Visualization: The class diagram is a structural diagram that combines classes, associations, and constraints to define the system's architecture.

Components of a Class Diagram:

The class diagram comprises three primary sections:

- Upper Section: This top segment features the class name, representing a group of objects that share common attributes, behaviors, and roles. Guidelines for displaying groups of objects include capitalizing the initial letter of the class name, positioning it in the center, using bold lettering, and employing slanted writing style for abstract class titles.
- Middle Section: In this part, the class's attributes are detailed, including their visibility indicators, denoted as public (+), private (-), protected (#), or package (~).
- Lower Section: The lower section elaborates on the class's methods or operations, presented in a list format with each method on a separate line. It outlines how the class interacts with data.

In UML, relationships within a class diagram fall into three categories:

- Dependency: Signifying the influence of one element's changes on another.
- Generalization: Representing a hierarchical relationship where one class acts as a parent, and another serves as its child.
- Association: Indicating connections between elements.
- Multiplicity: Defining constraints on the number of instances allowed to possess specific

characteristics, with one being the default value when not specified.

- Aggregation: An aggregation is a group that is a part of a relationship called association.
- Composition: "Composition" is like a smaller part of "aggregation.". This describes how a parent and child need each other, so if one is taken away, the other won't work anymore.

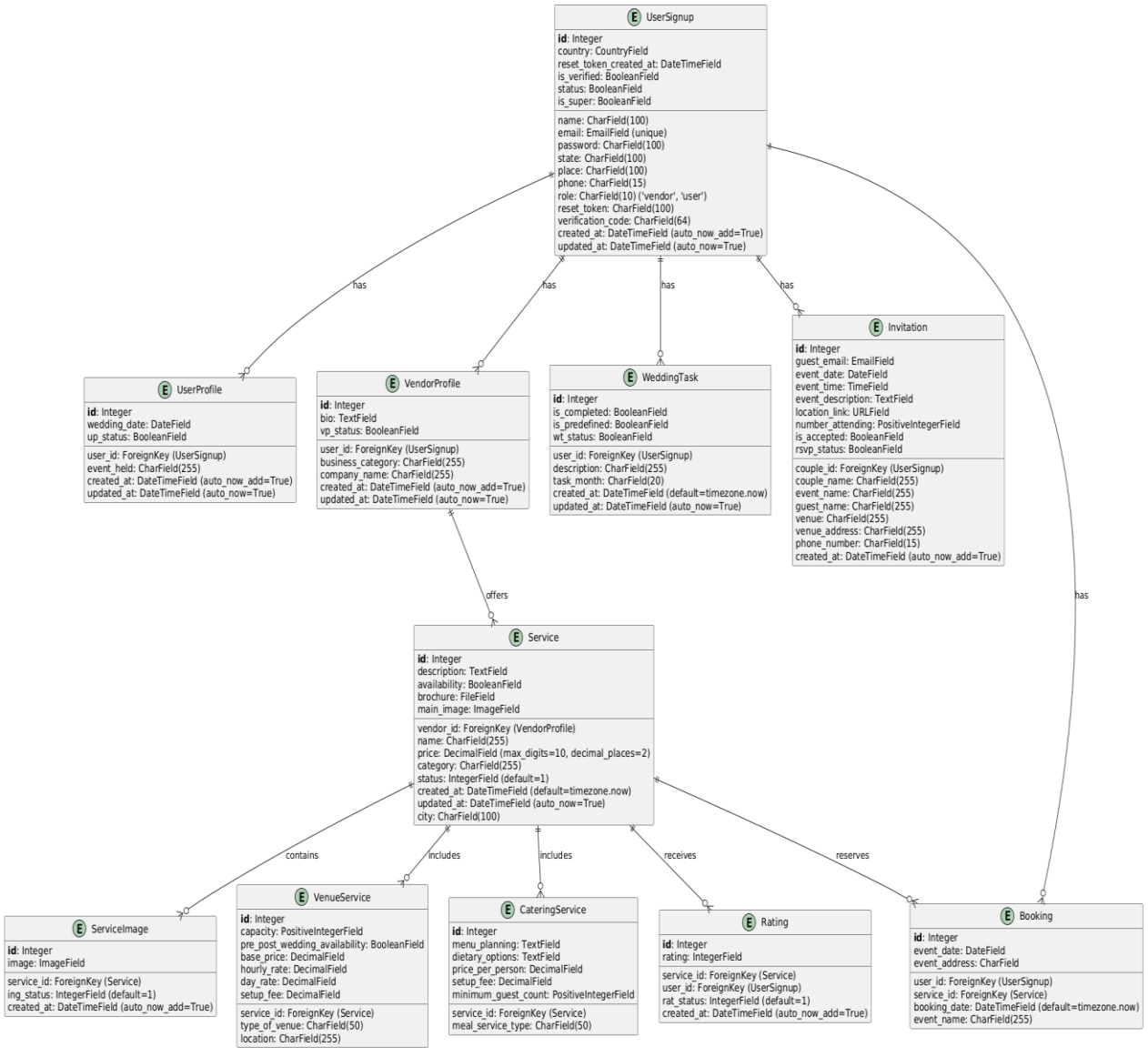


Fig 4.2.5: Class Diagram

4.2.6 Object Diagram

Object diagrams are derived from class diagrams and rely on them to provide a visual representation. They offer an illustration of a collection of objects related to a particular class. Object diagrams provide a snapshot of objects in an object-oriented system at a specific point in time.

Object diagrams and class diagrams share similarities, but they also have distinctions. Class diagrams are more generalized and do not portray specific objects. This abstraction in class diagrams simplifies the comprehension of a system's functionality and structure

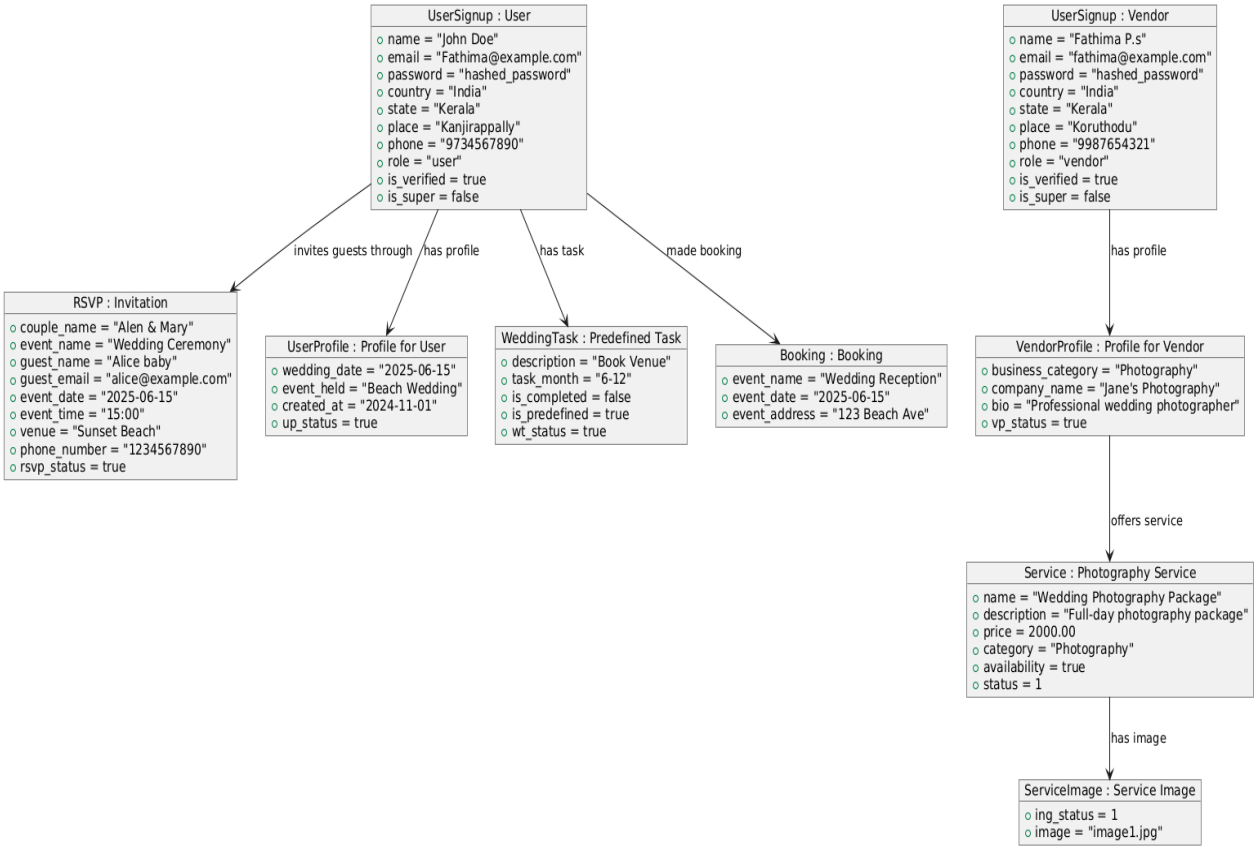


Fig 4.2.6: Object Diagram

4.2.7 Component Diagram

A component diagram serves the purpose of breaking down a complex system that utilizes objects into more manageable segments. It offers a visual representation of the system, showcasing its internal components such as programs, documents, and tools within the nodes.

This diagram elucidates the connections and organization of elements within a system, resulting in the creation of a usable system.

In the context of a component diagram, a component refers to a system part that can be modified and operates independently. It retains the secrecy of its internal operations and requires a specific method to execute a task, resembling a concealed box that functions only when operated correctly.

Notation for a Component Diagram includes:

- A component
- A node

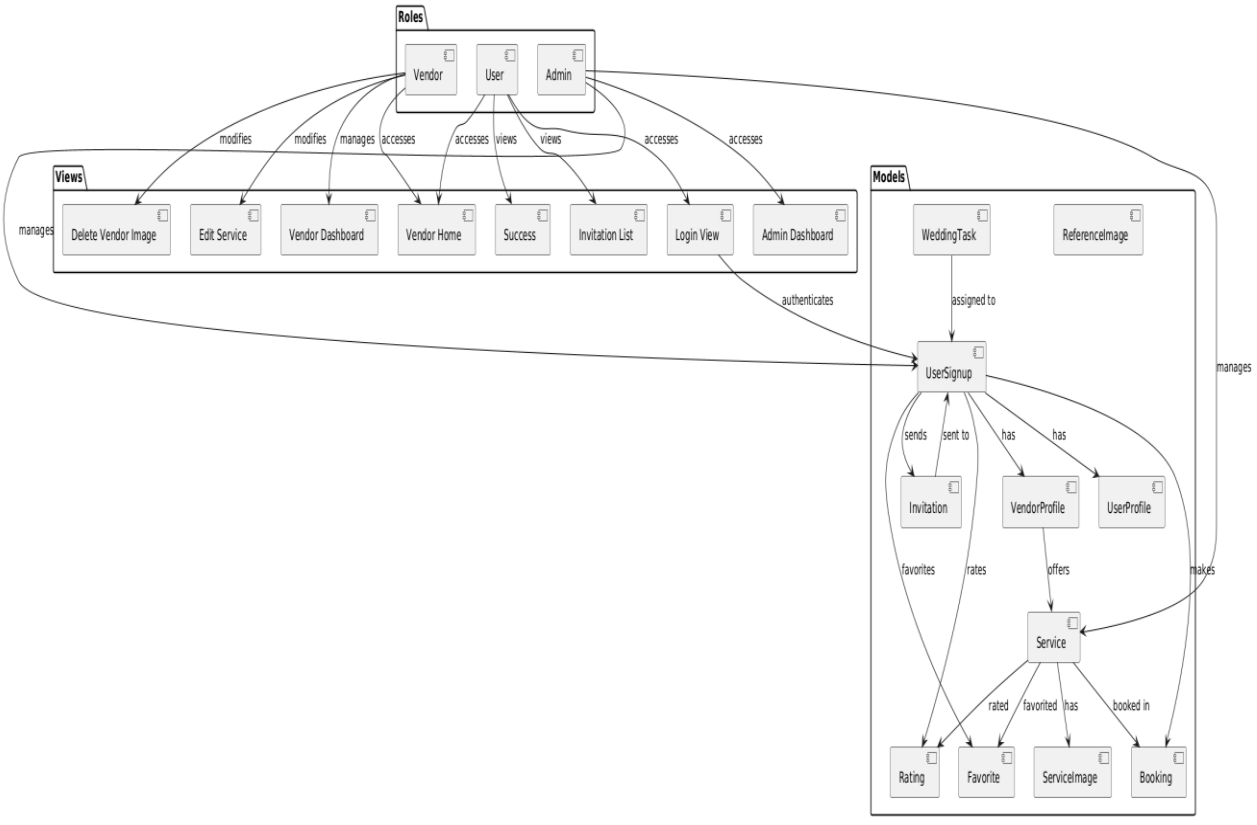


Fig 4.2.7: Component Diagram

4.2.8 Deployment Diagram

A deployment diagram provides a visual representation of how software is positioned on physical computers or servers. It depicts the static view of the system, emphasizing the arrangement of nodes and their connections.

This type of diagram delves into the process of placing programs on computers, elucidating how software is constructed to align with the physical computer system.

Notations in a Deployment Diagram include:

- A component
- An artifact
- An interface
- A node

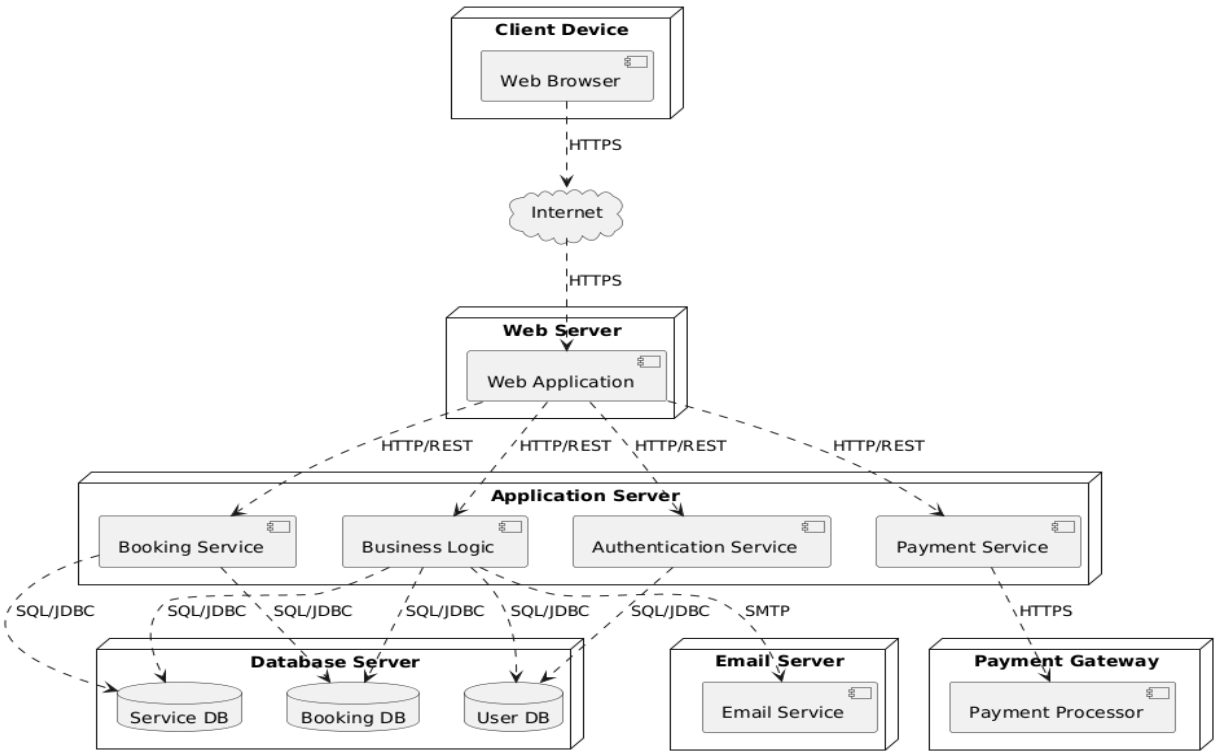


Fig 4.2.8: Deployment Diagram

4.2.9 Collaboration Diagram

A communication diagram, also known as a collaboration diagram, is a type of UML interaction diagram that shows how objects in a system interact with each other to accomplish a specific task or scenario. In this case, we're illustrating the booking process in the Online Hair Salon system.

Key elements of the communication diagram:

- Objects: Represented by boxes (Client, Web Interface, Authentication System, etc.)
- Links: Lines connecting objects, showing their associations
- Messages: Arrows on the links, showing the flow of communication
- Sequence numbers: Numbers preceding the messages, indicating the order of interactions

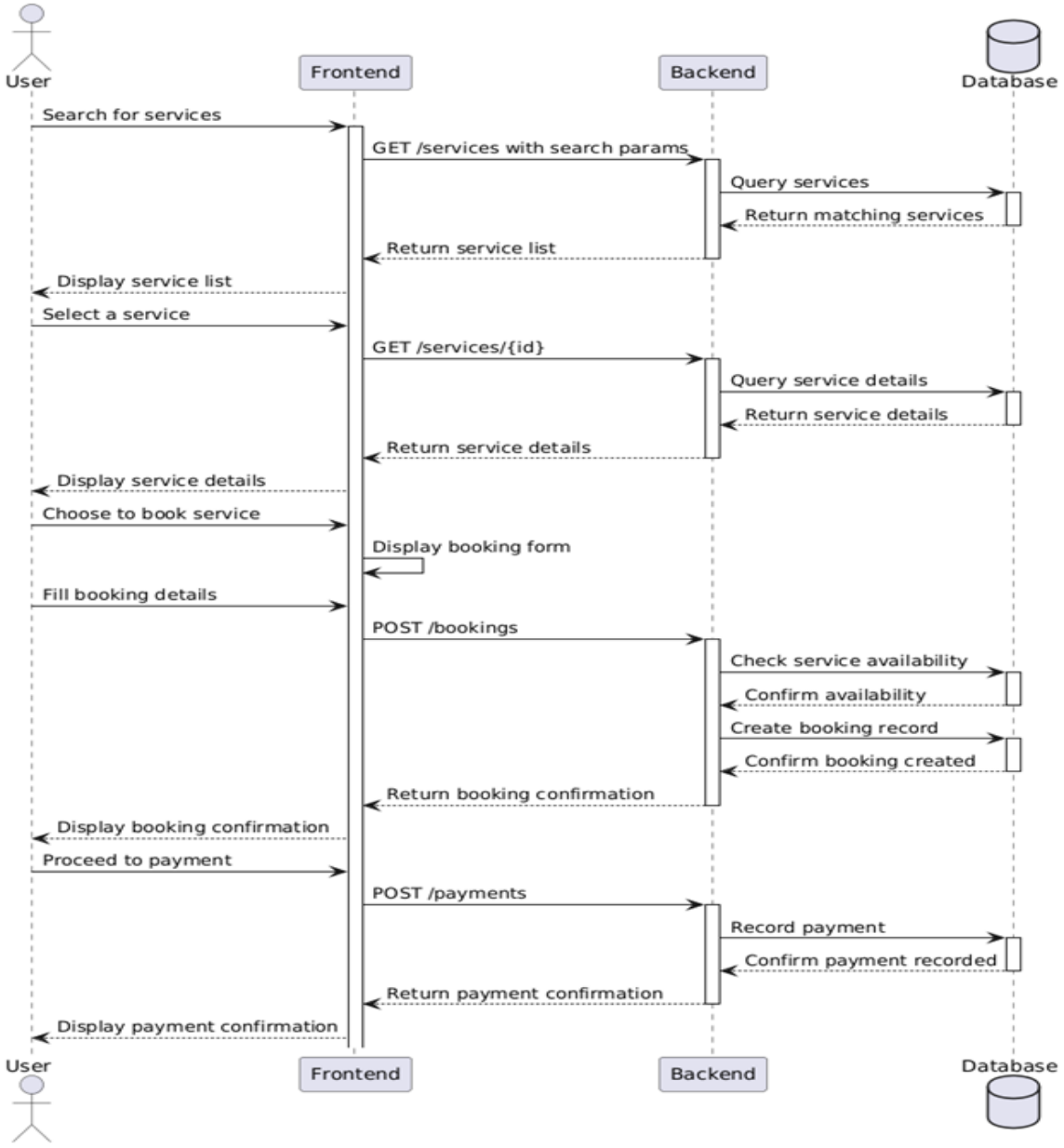


Fig 4.2.9: Collaboration Diagram

4.3 USER INTERFACE DESIGN USING FIGMA

4.3.1 Form Design: Home page

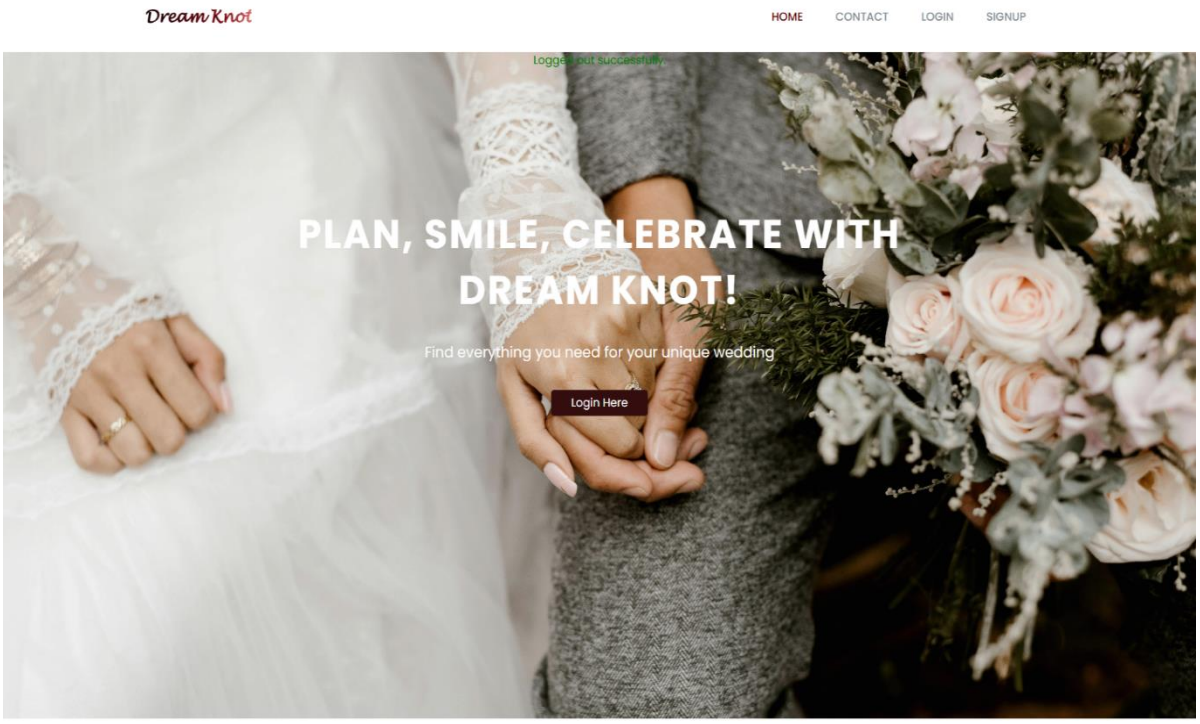


Figure 4.3.1: Home Page

4.3.2 Form Design: Login Page

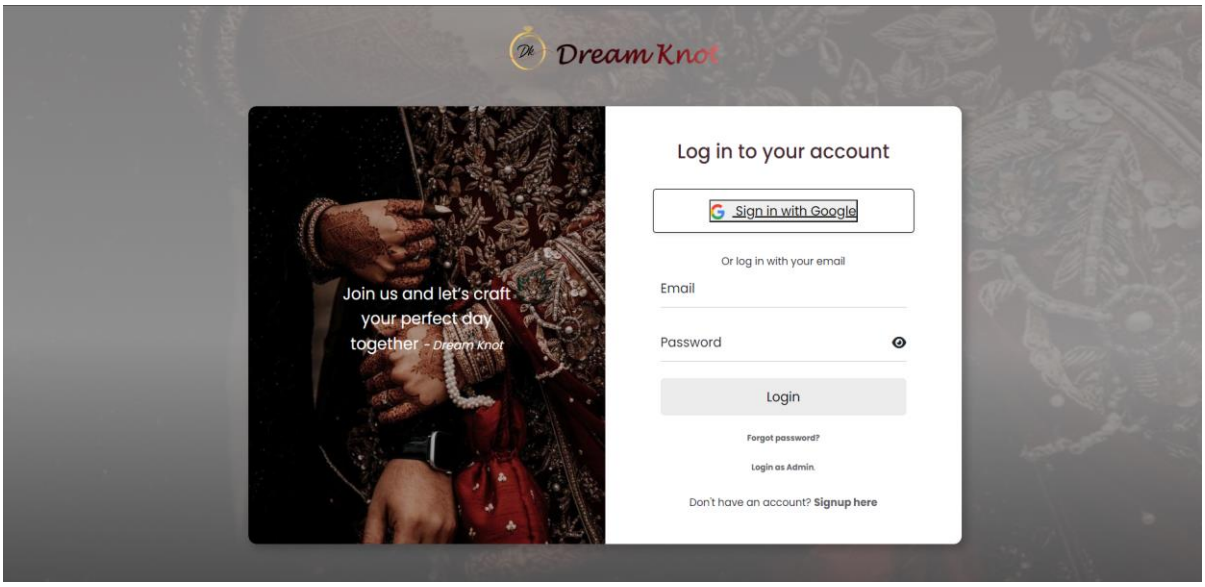


Figure 4.3.2: Login Page

4.3.3 Form Design: Registration Page

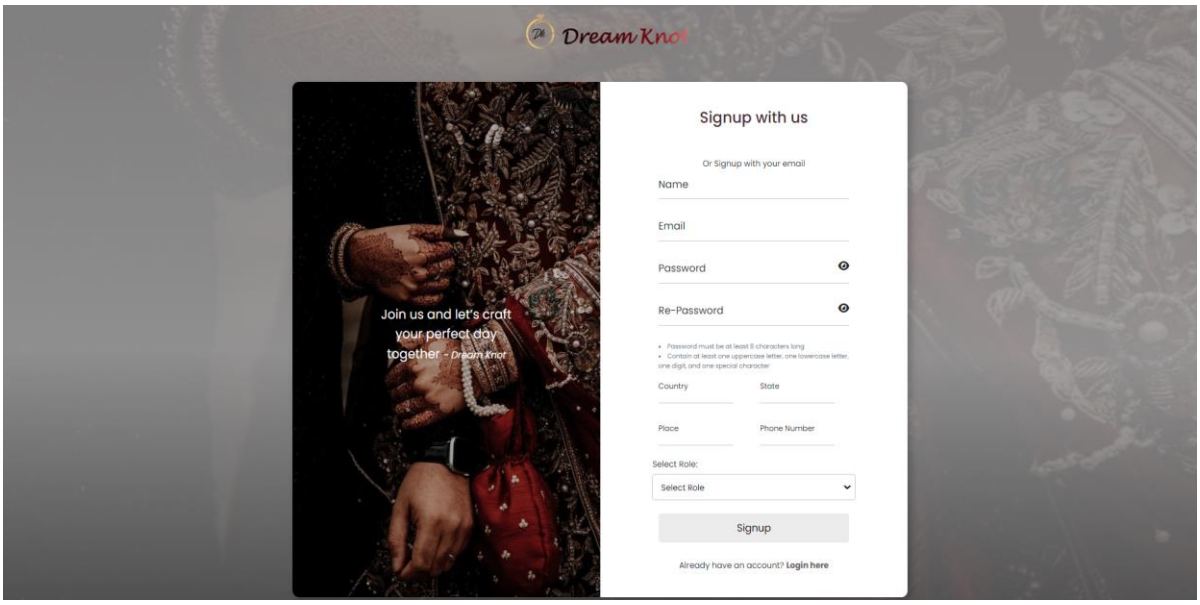
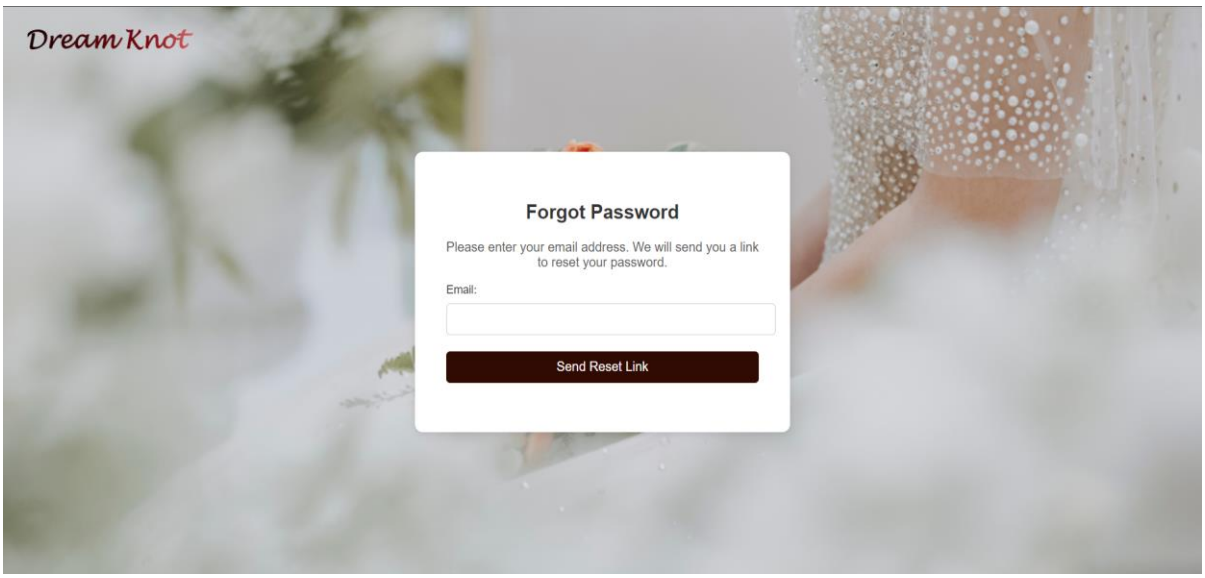


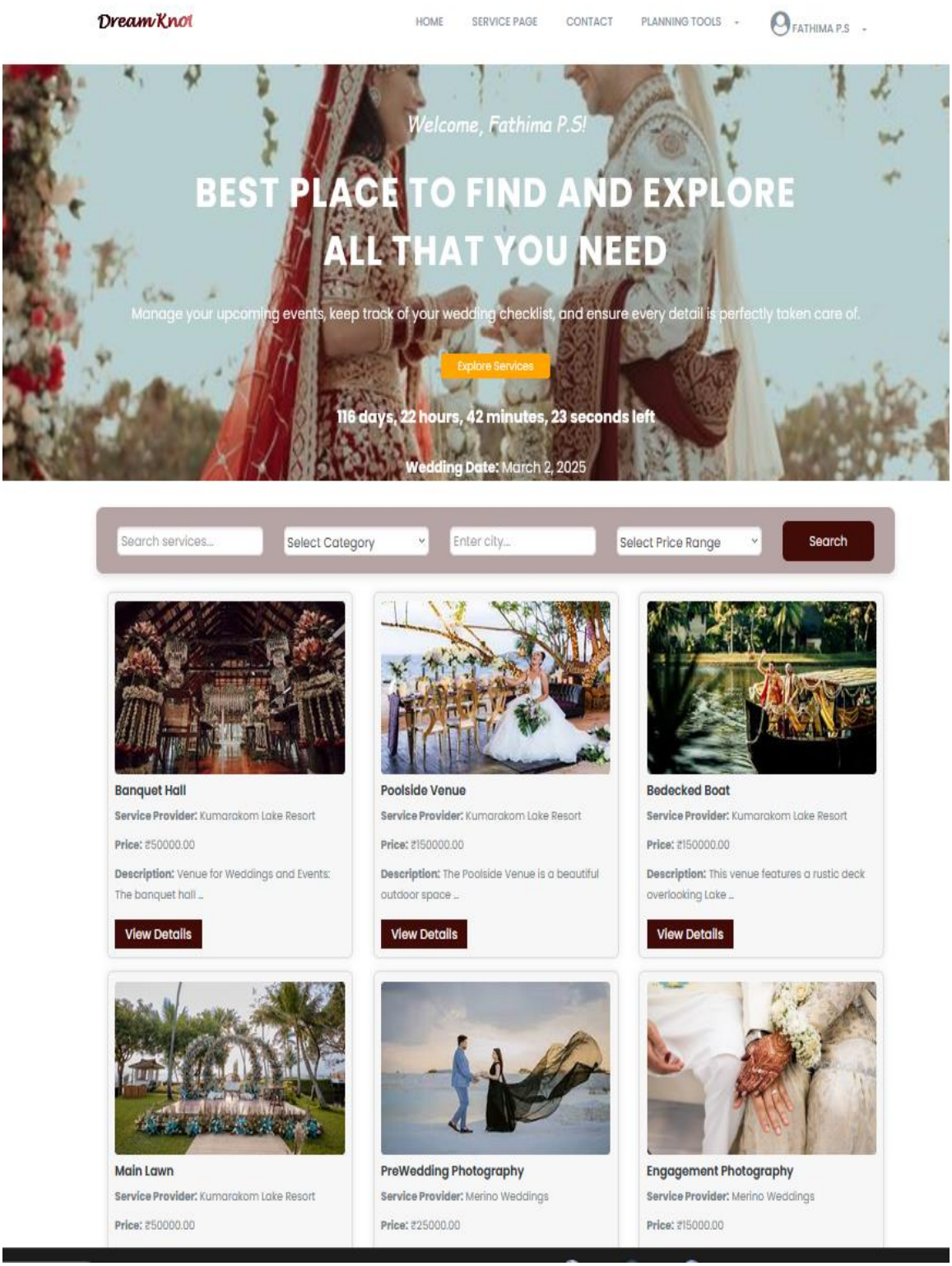
Figure 4.3.2:Registration Page

4.3.4 Form Design: Forgot Password



4.3.3 Forgot Password

4.3.5 Form Design: Index Page



4.3.5 Figure Index Page

4.4 DATABASE DESIGN

4.4.1 Relational Database Management System (RDBMS)

A database is a system designed to store and facilitate the retrieval and utilization of information. The integrity and security of the data within a database are of paramount importance.

Creating a database involves two essential steps. Initially, it is crucial to understand the user's requirements and establish a database structure that aligns with their needs. The first step entails devising an organizational plan for the information, which is not reliant on any particular computer program.

Subsequently, this plan is employed to craft a design tailored to the specific computer program that will be employed to construct the database system. This phase is centered on determining how data will be stored within the chosen computer program.

Key aspects of database development include:

- Data Integrity: Ensuring the accuracy and consistency of data stored within the database.
- Data Independence: The ability to modify the data storage structure without affecting the application's access to the data.

4.4.2 Normalization

A way of showing a database as a group of connections is called a relational model. Every relationship is like a chart of information or a group of data. In the formal way of talking about tables, a row is called a tuple, a column header is called an attribute, and the table is called a relation. A bunch of tables with special names make up a relational database. A row in a story shows a group of things that are connected.

Relations, Domains & Attributes

Tables serve as containers for organized and related information. Within a table, the rows are referred to as tuples, and a tuple is essentially an ordered list containing n items. Columns in a table are synonymous with characteristics, each representing a specific aspect of the data.

In a relational database, tables are interconnected, ensuring coherence and meaningful associations between different sets of data. This relational structure forms the foundation of a well-structured database.

A domain can be thought of as a collection of fundamental values with a shared source or data type. Naming domains is a helpful practice as it aids in comprehending the significance of the values they

encompass. It's essential to recognize that values within a domain are indivisible.

- Table relationships are established using keys, with the primary key and foreign key being the primary types. Entity Integrity and Referential Integrity are fundamental principles that guide these relationships.
- Entity Integrity mandates that no Primary Key should contain null values, reinforcing the unique and integral nature of primary keys in database relationships.

Normalization is the process of organizing data to ensure its efficient storage and to minimize redundancy while maintaining accuracy and reliability. This approach involves eliminating unnecessary columns and breaking down large tables into smaller, more manageable parts. Normalization helps prevent errors when adding, removing, or modifying data. In data modeling, two key concepts are integral to its normal form: keys and relationships.

Keys serve as unique identifiers to distinguish one row from all others in a table. There are two main types of keys: the primary key, which groups similar records within a table, and the foreign key, which acts as a special code to identify specific records in another table.

- **First Normal Form (1NF):** This rule dictates that an attribute can only contain a single value that cannot be further divided. Each value in a tuple must match the attribute's type. In 1NF, tables cannot contain other tables or have tables as part of their data. This form ensures that values are indivisible and represent a single entity. Achieving 1NF often involves organizing data into separate tables, with each table having a designated key based on project requirements. New relationships are established for various data categories or related groups, eliminating redundant information. A relationship adhering to primary key constraints alone is termed the first normal form.
- **Second Normal Form (2NF)**: In simpler terms, 2NF stipulates that no additional information should be associated with only part of the primary information used for data organization. This involves breaking down the information and creating separate groupings for each part along with its related details. It's essential to maintain a connection between the original primary key and any data dependent on it. This step helps remove data that relies on only a portion of the key. When a set of information has a primary means of identifying each item, and all other details depend solely on that primary means, it is considered to be in the second normal form.
- **Third Normal Form (3NF)** is a critical concept in database normalization, aiming to achieve table independence and control over attributes. In 3NF, a table should not contain columns that depend on other non-key columns, ensuring that each column is functionally

dependent only on the primary key. It breaks down relationships involving non-key attributes to eliminate dependencies on attributes not part of the primary key. To meet the criteria for 3NF, data must already be in the Second Normal Form (2NF), and non-key attributes should not rely on other non-key attributes within the same table, avoiding transitive dependencies. This process enhances data integrity, reduces redundancy, and optimizes database structure and organization.

4.4.3 Sanitization

Django incorporates various in-built mechanisms for data sanitization. To begin with, it provides a diverse range of field types that come with automatic validation and sanitization capabilities. For instance, the CharField performs automatic validation and cleaning of text input, while the EmailField ensures that email addresses adhere to the correct format. These field types serve as protective measures to prevent the storage of malicious or invalid data in the database.

Moreover, Django strongly advocates for the utilization of form validation to sanitize user input. Django's forms are equipped with predefined validation methods and validators that can be applied to form fields. These validators execute a range of checks and cleansing operations, such as confirming that numerical values fall within specified ranges or verifying that uploaded files adhere to specific formats. These combined features in Django promote data integrity and security, making it a reliable choice for web application development.

4.4.4 Indexing

The index keeps track of a certain piece of information or group of information, arranged in order of its value. Arranging the index entries helps find things quickly and easily that match exactly or are within a certain range. Indexes make it easy to find information in a database without having to search through every record every time the database is used. An index is like a roadmap for finding information in a database. It helps you look up data quickly and also makes it easy to find records that are in a certain order. It can be based on one or more columns in the table.

4.5 TABLE DESIGN

4.5.1.Tbl_UserSignup

Primary key: **user_id**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	user_id	INT (Primary Key, Auto Increment)	Primary Key	Unique identifier for each user
2	name	VARCHAR (100)		Name of the user
3	email	VARCHAR (100)	Unique	Email of the user
4	password	VARCHAR (100)		User’s password
5	country	VARCHAR (50)		User's country
6	state	VARCHAR (100)		User's state
7	place	VARCHAR (100)		User's city/place
8	phone	VARCHAR (15)		User's phone number
9	role	ENUM		Role of the user: 'vendor' or 'user'
10	reset_token	VARCHAR (100)	Nullable	Token for password reset
11	reset_token_created_at	DATETIME	Nullable	Timestamp for token creation
12	is_verified	BOOLEAN		Status of email verification
13	verification_code	VARCHAR (64)	Nullable	Code for email verification
14	created_at	DATETIME		Account creation timestamp
15	updated_at	DATETIME		Last update timestamp
16	status	BOOLEAN		Account active status
17	is_super	tinyint(1)		Indicator if user is a superuser

4.5.2.Tbl_vendorprofile

Primary key: **vendor_id**

Foreign key: **vendor_id** references table **Tbl_usersignup**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	vendor_id	INT (Primary Key, Auto Increment)	Primary Key	Unique identifier for each vendor profile
2	User_id	INT	Foreign Key (UserSignup)	Reference to the UserSignup
3	business_category	VARCHAR (255)	NotNull	Category of business
4	company_name	VARCHAR (255)	NotNull	Company name of the vendor
5	bio	TEXT	NotNull	Bio or description of the vendor
6	created_at	DATETIME	Nullable	Creation timestamp
7	updated_at	DATETIME	Nullable	Last update timestamp
8	vp_status	BOOLEAN	Nullable	Active status of the vendor profile

4.5.3.Tbl_userprofile

Primary key: **couple_ld**

Foreign key: **user_id** references table **Tbl_usersignup**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	couple_id	INT (Primary Key, Auto Increment)	Primary Key	Unique identifier for each profile
2	user_id	INT	Foreign Key (UserSignup)	Reference to the UserSignup
3	wedding_date	DATE	NotNull	Date of wedding
4	event_held	VARCHAR (255)	NotNull	Place where event is held
5	created_at	DATETIME	Nullable	Creation timestamp
6	updated_at	DATETIME	Nullable	Last update timestamp
7	up_status	BOOLEAN	Nullable	Active status of the profile

4.5.4 Tbl_ WeddingTask

Primary key: **WeddingTask_id**

Foreign key: **user_id** references table **Tbl_usersignup**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Weddingtask_id	INT (Primary Key, Auto Increment)	Primary Key	Unique identifier for each wedding task
2	user_id	INT	Foreign Key (UserSignup)	Reference to the UserSignup
3	description	VARCHAR (255)	NotNull	Description of the task
4	task_month	ENUM	NotNull	Timeline for task
5	is_completed	BOOLEAN	Nullable	Task completion status
6	is_predefined	BOOLEAN	Nullable	Indicates if the task is predefined
7	created_at	DATETIME	Nullable	Creation timestamp
8	updated_at	DATETIME	Nullable	Last update timestamp
9	wt_status	BOOLEAN	Nullable	Active status of the task

4.5.5 Tbl_RSVPInvitation

Primary key:**invitation_ld**

Foreign key: **couple_id** references table **Tbl_usersignup**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Invitation_id	INT (Primary Key, Auto Increment)	Primary Key	Unique identifier for each RSVP invitation
2	couple_id	INT	Foreign Key (UserSignup)	Reference to the couple in UserSignup
3	couple_name	VARCHAR (255)	NotNull	Name of the couple
4	event_name	VARCHAR (255)	NotNull	Name of the event
5	guest_name	VARCHAR (255)	NotNull	Name of the guest
6	guest_email	VARCHAR (255)	NotNull	Email of the guest

7	event_date	DATE	NotNull	Date of the event
8	event_time	TIME	NotNull	Time of the event
9	event_description	TEXT	Nullable	Optional event description
10	venue	VARCHAR (255)	NotNull	Venue name of the event
11	venue_address	VARCHAR (255)	NotNull	Address of the event venue
12	phone_number	VARCHAR (15)	NotNull	Contact number
13	location_link	VARCHAR (255)	Nullable	URL link for event location
14	number_attending	INT	Nullable	Number of attendees
15	is_accepted	BOOLEAN	Nullable	Indicates if the invitation is accepted
16	created_at	DATETIME	Nullable	Creation timestamp
17	rsvp_status	BOOLEAN	Nullable	Active status of the RSVP invitation

4.5.6.Tbl_Service

Primary key: **service_ld**

Foreign key: **vendor_id** references table **Tbl_vendorprofile**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Service_id	INT	Primary Key, Auto Increment	Unique identifier for each service
2	vendor_id	INT	Foreign Key (VendorProfile)	Reference to the VendorProfile
3	name	VARCHAR (255)	Not Null	Name of the service
4	description	TEXT	Not Null	Description of the service
5	price	DECIMAL (10, 2)	Not Null	Price of the service
6	category	VARCHAR (255)	Not Null	Category of the service (e.g., Photography, etc.)
7	availability	BOOLEAN	Default True	Availability status of the service
8	status	INT	Default 1	Status of the service (0 for Inactive, 1 for Active)

9	created_at	DATETIME	Nullable	Creation timestamp
10	updated_at	DATETIME	Nullable	Last update timestamp
11	city	VARCHAR (100)	Nullable	City where the service is offered
12	brochure	FILE	Nullable	Brochure file for the service
13	main_image	IMAGE	Nullable	Main image for the service

4.5.7 Tbl_ServiceImage

Primary key: **Serviceimage_ ld**

Foreign key: **service_id** references table **Tbl_service**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Serviceimage_id	INT	Primary Key, Auto Increment	Unique identifier for each service image
2	service_id	INT	Foreign Key (Service)	Reference to the Service model
3	image	IMAGE	Not Null	Image associated with the service
4	ing_status	INT	Default 1	Status of the image (0 for Inactive, 1 for Active)
5	created_at	DATETIME	Primary Key, Auto Increment	Creation timestamp

4.5.8 Tbl_VenueService

Primary key: **venueservice_ld**

Foreign key: **service_id references table Tbl_service**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Venueservice_id	INT	Primary Key, Auto Increment	Unique identifier for the venue service
2	service_id	INT	Foreign Key (Service)	Reference to the Service model
3	type_of_venue	VARCHAR (50)	Not Null	Type of venue (Indoor, Outdoor, Destination)
4	location	VARCHAR (255)	Not Null	Location of the venue

5	capacity	INT	Not Null	Capacity of the venue
6	pre_post_weddin g_availability	BOOLEAN	Default True	Availability before and after the wedding
7	base_price	DECIMAL (10, 2)	Not Null	Base price for the venue service
8	hourly_rate	DECIMAL (10, 2)	Default 0.00	Hourly rate for the venue service
9	day_rate	DECIMAL (10, 2)	Not Null	Day rate for the venue service
10	setup_fee	DECIMAL (10, 2)	Default 0.00	Setup fee for the venue service

4.5.9 Tbl_ CateringService

Primary key: **Cateringservice_id**

Foreign key: **service_id references table Tbl_service**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	Cateringservice_id	INT	Primary Key, Auto Increment	Unique identifier for the catering service
2	service_id	INT	Foreign Key (Service)	Reference to the Service model
3	menu_planning	TEXT	Not Null	Menu planning details
4	meal_service_type	VARCHAR (50)	Not Null	Type of meal service (Buffet, Plated, etc.)
5	dietary_options	TEXT	Not Null	Dietary options available
6	price_per_person	DECIMAL (10, 2)	Not Null	Price per person for catering
7	setup_fee	DECIMAL (10, 2)	Default 0.00	Setup fee for catering services
8	minimum_guest_cou nt	INT	Default 1	Minimum guest count for catering

4.5.10 Tbl_PhotographyService

Primary key: **photographyService _ld**

Foreign key: **service_id references table Tbl_service**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field

1	photographyService_id	INT	Primary Key, Auto Increment	Unique identifier for the photography service
2	service_id	INT	Foreign Key (Service)	Reference to the Service model
3	package_duration	VARCHAR (50)	Not Null	Duration of the photography package
4	styles	TEXT	Not Null	Styles offered for photography
5	engagement_shoots	BOOLEAN	Default False	Availability of engagement shoots
6	videography_options	BOOLEAN	Default False	Availability of videography options
7	base_price	DECIMAL(10, 2)	Not Null	Base price for photography services
8	hourly_rate	DECIMAL(10, 2)	Default 0.00	Hourly rate for photography services

11.Tbl_MusicEntertainmentService

Primary key: **MusicEntertainmentService_ld**

Foreign key: **service_id references table Tbl_service**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	MusicEntertainmentService_id	INT	Primary Key, Auto Increment	Unique identifier for the music entertainment service
2	service_id	INT	Foreign Key (Service)	Reference to the Service model
3	entertainment_options	TEXT	Not Null	Options for entertainment services
4	sound_system_setup	BOOLEAN	Default False	Availability of sound system setup
5	multiple_entertainment_acts	BOOLEAN	Default False	Availability of multiple entertainment acts
6	emcee_services	BOOLEAN	Default False	Availability of emcee services
7	playlist_customization	BOOLEAN	Default False	Availability of playlist customization
8	base_price	DECIMAL (10, 2)	Not Null	Base price for music entertainment services
9	hourly_rate	DECIMAL (10, 2)	Default 0.00	Hourly rate for music entertainment services

12.Tbl_MakeupHairService

Primary key: **MakeupHairService _ld**

Foreign key **:service_id references table Tbl_service**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	MakeupHairService _id	INT	Primary Key, Auto Increment	Unique identifier for the makeup & hair service
2	service_id	INT	Foreign Key (Service)	Reference to the Service model
3	grooming_services	TEXT	Not Null	Description of grooming services
4	trial_sessions	BOOLEAN	Default False	Availability of trial sessions
5	high_end_products	BOOLEAN	Default False	Use of high-end products
6	base_price	DECIMAL(10, 2)	Not Null	Base price for makeup & hair services
7	hourly_rate	DECIMAL(10, 2)	Default 0.00	Hourly rate for makeup & hair services

13. Tbl_RentalsService

Eg.Primary key: **RentalsService_ld**

Eg.Foreign key: **service_id references table Tbl_service**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	RentalsService_id	INT	Primary Key, Auto Increment	Unique identifier for the rentals service
2	service_id	INT	Foreign Key (Service)	Reference to the Service model
3	rental_items	TEXT	Not Null	Description of rental items
4	setup_services	BOOLEAN	Default False	Availability of setup services
5	rental_price_per_item	DECIMAL(10, 2)	Not Null	Price per item for rental services

14.Tbl_RentalsService

Eg.Primary key: **RentalsService_ld**

Eg.Foreign key: **service_id references table Tbl_service**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	RentalsService_id	INT	Primary Key, Auto Increment	Unique identifier for the rentals service
2	service_id	INT	Foreign Key (Service)	Reference to the Service model
3	rental_items	TEXT	Not Null	Description of rental items
4	setup_services	BOOLEAN	Default False	Availability of setup services
5	rental_price_per_item	DECIMAL(10, 2)	Not Null	Price per item for rental services

14.Tbl_Rating

Primary key: **Rating_ld**

Foreign key: **service_id references table Tbl_service**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	rating_id	INT	PRIMARY KEY, AUTO_INCREMENT	Primary key for each rating record
2	service_id	INT	FOREIGN KEY	References the Service ID
3	user_id	INT	FOREIGN KEY	References the UserSignup ID
4	rating	INT		Rating value given by the user (1-5)
5	rat_status	TINYINT(1)	DEFAULT 1	Status of the rating (0: Inactive, 1: Active)
6	created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Timestamp when the rating was created

15.Tbl_Booking

Primary key: **user_id**

Foreign key: **service_id references table Tbl_service and user_id** references table **Tbl_usersignup**

No:	Field name	Datatype (Size)	Key Constraints	Description of the field
1	booking_id	INT	PRIMARY KEY, AUTO_INCREMENT	Primary key for each booking record
2	user_id	INT	FOREIGN KEY	References the UserSignup ID
3	service_id	INT	FOREIGN KEY	References the Service ID
4	booking_date	DATETIME	Not Null	Date and time when the booking was made
5	event_name	VARCHAR (255)	DEFAULT 1	Name of the event
6	event_date	DATE	DEFAULT CURRENT_TIMESTAMP	Scheduled date of the event
7	event_address	VARCHAR (255)		Address of the event location
8	user_address	VARCHAR (255)		Address of the user
9	num_days	INT		Number of days for the booking
10	additional_req	TEXT		Additional requirements for the event
11	vendor_confirmed_at	DATETIME		Timestamp when the vendor confirmed the booking
12	canceled_by_user	TINYINT (1)		Indicates if the booking was canceled by user
13	cancellation_reason	TEXT	PRIMARY KEY, AUTO_INCREMENT	Reason for cancellation
14	book_status	TINYINT (1)	FOREIGN KEY	Status of booking (0: Pending, 1: Confirmed, 2: Completed, 3: Canceled)
15	total_amount	DECIMAL (10,2)	FOREIGN KEY	Total amount for the booking
16	booking_amount	DECIMAL (10,2)	DEFAULT CURRENT_TIMESTAMP	Amount paid at the time of booking

17	terms_and_conditions	TEXT	NULLABLE	Terms and conditions of the booking
18	user_agreed_to_terms	TINYINT (1)		Indicates if the user agreed to the terms
19	user_agreement_date	DATETIME	NULLABLE	Date when the user agreed to the terms
20	razorpay_order_id	VARCHAR (255)	NULLABLE	Order ID from Razorpay for tracking payment
21	razorpay_payment_id	VARCHAR (255)	DEFAULT 1, MINIMUM 1	Payment ID from Razorpay
22	razorpay_signature	VARCHAR (255)	NULLABLE	Signature from Razorpay to validate payment

CHAPTER 5
SYSTEM TESTING

5.1 INTRODUCTION

Software testing is an essential procedure employed to verify if a computer program functions as intended. It is conducted to ensure that the software performs its designated tasks accurately and complies with the prescribed requirements and standards. Validation is the process of inspecting and assessing software to confirm its adherence to the specified criteria. Software testing is a method for assessing the performance of a program, often in conjunction with techniques like code inspection and program walkthroughs. Validation ensures that the software aligns with the user's expectations and requirements.

Several principles and objectives guide the process of software testing, including:

1. Testing is the practice of executing a program with the primary aim of identifying errors.
2. An effective test case is one that has a high likelihood of uncovering previously undiscovered errors.
3. A successful test is one that exposes previously undiscovered errors.

When a test case operates effectively and accomplishes its objectives, it can detect flaws within the software. This demonstrates that the computer program is functioning as intended and is performing well. The process of evaluating a computer program encompasses three primary aspects:

1. Correctness assessment
2. Evaluation of implementation efficiency
3. Examination of computational complexity

5.2 TEST PLAN

A test plan serves as a comprehensive set of instructions for conducting various types of tests. It can be likened to a map that outlines the steps to follow when evaluating a computer program. Software developers create instructions for both using and organizing the necessary information for the program's proper functionality. They ensure that each component of the program performs its intended functions. To ensure the thorough testing of the software, a group known as the ITG (Information Technology Group) is responsible for verifying its functionality, instead of relying solely on the software's creators for testing.

The objectives of testing should be clearly defined and measurable. A well-structured test plan should encompass details about the frequency of failures, the associated repair costs, the occurrence rate of issues, and the time required for complete testing. The levels of testing typically include:

- Unit testing
- Integration testing
- Data validation testing
- Output testing

5.2.1 Unit Testing

Unit testing checks the smallest part of a software design - the software component or module. Testing important control paths within a module using the design guide to find errors. This means how difficult the tests are for each small part of a program and what parts of the program haven't been tested yet. Unit testing is a type of testing that looks at how the code works inside and can be done at the same time for different parts of the program.

Before starting any other test, we need to check if the data flows correctly between different parts of the computer program. If the information doesn't move in and out correctly, all other checks are pointless. When designing something, it's important to think about what could go wrong and make a plan for how to deal with those problems. This can mean redirecting the process or stopping it completely.

The Sell-Soft System was tested by looking at each part by itself and trying different tests on it. Some mistakes in the design of the modules were discovered and then fixed. After writing the instructions for different parts, each part is checked and tried out separately. We got rid of extra code and made sure everything works the way it should.

5.2.2 Integration Testing

Integration testing is a critical process in software development that involves constructing a program while simultaneously identifying errors in the interaction between different program components. The primary objective is to utilize tested individual parts and assemble them into a program according to the initial plan. This comprehensive testing approach assesses the entire program to ensure its proper and correct functionality.

As issues are identified and resolved during integration testing, it's not uncommon for new problems to surface, leading to an ongoing cycle of testing and refinement. After each individual component of the system is thoroughly examined, these components are integrated to ensure they function

harmoniously. Additionally, efforts are made to standardize all programs to ensure uniformity rather than having disparate versions.

5.2.3 Validation Testing or System Testing

The final phase of testing involves a comprehensive examination of the entire system to ensure the correct interaction of various components, including different types of instructions and building blocks. This testing approach is referred to as Black Box testing or System testing.

Black Box testing is a method employed to determine if the software functions as intended. It assists software engineers in identifying all program issues by employing diverse input types. Black Box testing encompasses the assessment of errors in functions, interfaces, data access, performance, as well as initialization and termination processes. It is a vital technique to verify that the software meets its intended requirements and performs its functions correctly.

5.2.4 Output Testing or User Acceptance Testing

System testing is conducted to assess user satisfaction and alignment with the company's

requirements. During the development or update of a computer program, it's essential to maintain a connection with the end-users. This connection is established through the following elements:

- Input Screen Designs.
- Output Screen Designs.

To perform system testing, various types of data are utilized. The preparation of test data plays a

crucial role in this phase. Once the test data is gathered, it is used to evaluate the system under investigation. When issues are identified during this testing, they are addressed by following established procedures. Records of these corrections are maintained for future reference and improvement. This process ensures that the system functions effectively and meets the needs of both users and the organization.

5.2.5 Automation Testing

Automated testing is a method employed to verify that software functions correctly and complies with established standards before it is put into official use. This type of testing relies on written instructions that are executed by testing tools. Specifically, UI automation testing involves the use of specialized tools to automate the testing process. Instead of relying on manual interactions where individuals click through the application to ensure its proper functioning, scripts are created to automate these tests for various scenarios. Automating testing is particularly valuable when it is necessary to conduct the same test across multiple computers simultaneously, streamlining the testing process and ensuring consistency.

5.2.6 Selenium Testing

Selenium is a valuable and free tool designed for automating website testing. It plays a crucial role for web developers as it simplifies the testing process. Selenium automation testing refers to the practice of using Selenium for this purpose. Selenium isn't just a single tool; it's a collection of tools, each serving distinct functions in the realm of automation testing. Manual testing is a necessary aspect of application development, but it can be monotonous and repetitive. To alleviate these challenges, Jason Huggins, an employee at ThoughtWorks, devised a method for automating testing procedures, replacing manual tasks. He initially created a tool named the JavaScriptTestRunner to facilitate automated website testing, and in 2004, it was rebranded as Selenium.

Test Case 1

Code

```
package Definitions;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import io.cucumber.java.en.And;

import io.cucumber.java.en.Given;

import io.cucumber.java.en.Then;

import io.cucumber.java.en.When;

public class stepdefinition {

		WebDriver driver=null;

		@Given("browser is open")

		public void browser_is_open() {

		 System.out.println("Inside step-Browser is open");


System.setProperty("webdriver.gecko.marionette","E:\\Testing\\src\\test\\resources
\\drivers\\geckodriver.exe");

		 driver=new FirefoxDriver();

		 driver.manage().window().maximize();


		}


		@And("user is on login page")

		public void user_is_on_login_page() throws Exception {

		 driver.navigate().to("http://127.0.0.1:8000/login");

		 Thread.sleep(2000);


		}
```

```
            @When("user enters username and password")
           public void user_enters_username_and_password() throws Throwable{

driver.findElement(By.id("email")).sendKeys("fathimaps521050@gmail.com");
             driver.findElement(By.id("password")).sendKeys("Fathima@123");
            }


            @When("user clicks on login")
            public void user_clicks_on_login() {
             driver.findElement(By.id("login")).click(); }



}
```

Screenshot

```
Scenario: Check login is successful with valid credentials # src/test/resources/Features/Login.feature:3
Inside step-Browser is open
  Given browser is open                                    # Definitions.stepdefinition.browser_is_open()
  And user is on login page                                # Definitions.stepdefinition.user_is_on_login_page()
  When user enters username and password                   # Definitions.stepdefinition.user_enters_username_and_password()
  And user clicks on login                                 # Definitions.stepdefinition.user_clicks_on_login()

1 Scenarios (1 passed)
4 Steps (4 passed)
0m9.834s
```

Test Report

Test Case 1

Project Name: Dream Knot					
Login Test Case					
Test Case ID: Test_1			**Test Designed By: Fathima P.S**		
Test Priority(Low/Medium/High): High			**Test Designed Date: 30-10-2024**		
Module Name: Login Module			**Test Executed By : Mr. T J Jobin**		
Test Title : User Login			**Test Execution Date: 30-10-2024**		
Description: User has a valid email/password					
Pre-Condition : User has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	pass
2	Provide valid email	fathimaps521050@gmail.com	User should be able to login	User logs in	pass
3	Provide valid password	Fathima@123			
4	Click on login button				
Post-Condition: User is validate with database and successfully logs into Home page					

Test Case 2:

Code

```
package Definitions;


import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import io.cucumber.java.en.And;
```

```
import io.cucumber.java.en.Given;

import io.cucumber.java.en.Then;

import io.cucumber.java.en.When;

public class AddToFavoritesSteps {

              WebDriver driver=null;

              @Given("browser is opens")

              public void browser_is_open() {

               System.out.println("Inside step-Browser is open");


System.setProperty("webdriver.gecko.marionette","E:\\Testing\\src\\test\\resources\\dr
ivers\\geckodriver.exe");

               driver=new FirefoxDriver();

               driver.manage().window().maximize();


              }


              @And("user is on login pages")

              public void user_is_on_login_page() throws Exception {

               driver.navigate().to("http://127.0.0.1:8000/login");

               Thread.sleep(2000);


              }


              @When("user enters username and passwords")

             public void user_enters_username_and_password() throws Throwable{


driver.findElement(By.id("email")).sendKeys("fathimaps521050@gmail.com");

               driver.findElement(By.id("password")).sendKeys("Fathima@123");

              }


              @And("user clicks on logins")
```

```
            public void user_clicks_on_login() {

             driver.findElement(By.id("login")).click();
             }

            @Then("user clicks on view details")

            public void user_clicks_on_view_details() {

             driver.findElement(By.id("viewdetails")).click();
             }

            @And("user clicks on addToFavorite")

            public void user_clicks_on_addToFavorite() {

             driver.findElement(By.id("addToFavorite")).click();
             }


            @Then("user clicks on account")
            public void user_clicks_on_account() {
                driver.findElement(By.id("account")).click();  // Click on the
account dropdown
            }

            @And("user clicks on favorite list")
            public void user_clicks_on_favorite_list() {
                driver.findElement(By.id("favourite")).click();  // Click on the
favorite list link

            }


}
```

Screenshot

```
Scenario: User logs in and adds a service to favorites # src/test/resources/Features/add_to_favorites.feature:3
Inside step-Browser is open
  Given browser is opens                              # Definitions.AddToFavoritesSteps.browser_is_open()
  And user is on login pages                          # Definitions.AddToFavoritesSteps.user_is_on_login_page()
  When user enters username and passwords             # Definitions.AddToFavoritesSteps.user_enters_username_and_password()
  And user clicks on logins                           # Definitions.AddToFavoritesSteps.user_clicks_on_login()
  Then user clicks on view details                    # Definitions.AddToFavoritesSteps.user_clicks_on_view_details()
  And user clicks on addToFavorite                    # Definitions.AddToFavoritesSteps.user_clicks_on_addToFavorite()
  Then user clicks on account                         # Definitions.AddToFavoritesSteps.user_clicks_on_account()
  And user clicks on favorite list                    # Definitions.AddToFavoritesSteps.user_clicks_on_favorite_list()

1 Scenarios (1 passed)
8 Steps (8 passed)
0m8.873s
```

Test report

Test Case 2

Project Name: Dream Knot					
Login Test Case					
Test Case ID: Test_2			**Test Designed By: Fathima P.S**		
Test Priority(Low/Medium/High): High			**Test Designed Date: 30-10-2024**		
Module Name: **User Management**			**Test Executed By : Mr. T J Jobin**		
Test Title : Add Service to Favorites			**Test Execution Date: 30-10-2024**		
Description: User with valid credentials adds a service to their favorites list					
Pre-Condition : User is logged in and navigates to a service details page					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	pass
2	Provide valid email	fathimaps521050@gmail.com	User should be able to login	User logs in	pass
3	Provide valid password	Fathima@123			
4	Click on login button				
5	Click on view details for a service		Service details page should be displayed	Service details page is displayed	pass
6	Click on add to favorites		Service should be added to favorites	Service added to favorites	pass
7	Click on account		Account dropdown should be displayed	Account dropdown is displayed	pass
8	Click on favorite list		Favorite list should show the added service	Service appears in favorite list	pass
Post-Condition: User is validated with database and successfully logs into Homepage and the user successfully adds a service to their favorites list. This can be verified by navigating to the "Favorite List" section under the user's account, where the selected service is now displayed.					

Test Case 3:

Code

```
package Definitions;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class invitation {

    WebDriver driver = null;

    @Given("browser is openss")
    public void browser_is_open() {
        System.out.println("Inside step - Browser is open");

System.setProperty("webdriver.gecko.marionette","E:\\Testing\\src\\test\\resources\\drivers\\geckodriver.exe");
        driver = new FirefoxDriver();
        driver.manage().window().maximize();
    }

    @And("user is on login pagess")
    public void user_is_on_login_page() throws Exception {
        driver.navigate().to("http://127.0.0.1:8000/login");
        Thread.sleep(2000);  // Wait for the page to load
    }

    @When("user enters username and passwordss")
    public void user_enters_username_and_password() throws Throwable {
        driver.findElement(By.id("email")).sendKeys("fathimaps521050@gmail.com");
        driver.findElement(By.id("password")).sendKeys("Fathima@123");
    }

    @And("user clicks on loginss")
    public void user_clicks_on_login() {
        driver.findElement(By.id("login")).click();
    }

    @Then("user clicks on Planning")
    public void user_clicks_on_Planning() {
        driver.findElement(By.id("Planning")).click();  // Ensure the ID matches your HTML
    }

    @And("user clicks on Invitation")
    public void user_clicks_on_Invitation() {
        driver.findElement(By.id("Invitation")).click();  // Ensure the ID matches your HTML
    }
```

```
    @When("user enters event name, event date, event time, venue, venue address, phone
number, location link,guestname,guestemail, and event description")
    public void user_enters_event_details() throws Throwable {
        driver.findElement(By.id("event_name")).sendKeys("Wedding");
        driver.findElement(By.id("event_date")).sendKeys("2024-12-20");  // Use YYYY-
MM-DD format
        driver.findElement(By.id("event_time")).sendKeys("17:19");
        driver.findElement(By.id("venue")).sendKeys("KMA Hall");
        driver.findElement(By.id("venue_address")).sendKeys("Padipurackal (H),
Edakkunnam P.O, Kanjirappally");
        driver.findElement(By.id("phone_number")).sendKeys("9744685155");

driver.findElement(By.id("location_link")).sendKeys("https://maps.app.goo.gl/a67tBVPPw
63MUMdu7");
        driver.findElement(By.id("event_description")).sendKeys("We heartily welcome
one and all");
        driver.findElement(By.id("guest_name")).sendKeys("fathima P.S");

driver.findElement(By.id("guest_email")).sendKeys("fathimaps521050@gmail.com");
    }

    @And("user clicks on send invitation")
    public void user_clicks_on_send_invitation() {
        driver.findElement(By.id("submit")).click();  // Ensure the ID matches your
HTML
    }
}
```

Screenshot

```
Scenario: User logs in and sends an invitation
Inside step - Browser is open
  Given browser is openss
  And user is on login pagess
  When user enters username and passwordss
  And user clicks on loginss
  Then user clicks on Planning
  And user clicks on Invitation
  When user enters event name, event date, event time, venue, venue address, phone number, location link,guestname,guestemail, and event description
  And user clicks on send invitation

1 Scenarios (1 passed)
8 Steps (8 passed)
0m32.087s
```

Test report

Test Case 3

Project Name: Dream Knot	
Login Test Case	
Test Case ID: Test_2	**Test Designed By: Fathima P.S**
Test Priority(Low/Medium/High): High	**Test Designed Date: 30-10-2024**
Module Name:	**Test Executed By : Mr. T J Jobin**

Invitation Module					
Test Title : Send Invitation			**Test Execution Date: 30-10-2024**		
Description: User has a valid email/password			User with valid credentials sends an event invitation		
Pre-Condition : User is logged in and navigates to the Invitation section					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	pass
2	Provide valid email	fathimaps521050 @gmail.com	User should be able to login	User logs in	pass
3	Provide valid password	Fathima@123			
4	Click on login button				
5	Click on Planning section		Planning section is displayed	Planning section is displayed	pass
6	Click on Invitation		Invitation form should be displayed	Invitation form is displayed	pass
7	Enter event details (name, date, time, venue, VenueAddress.,phone,location link,guest name,guest email, description)	Event details: Wedding, 2024-12-20, 17:19, KMA Hall, Padipurackal (H),Edakunnam, 9744685155, Google Maps Link, fathima P.S, fathimaps521050 @gmail.com, "We heartily welcome one and all"	User fills in invitation details	Invitation details filled	pass
8	Click on send invitation		Invitation should be sent	Invitation sent	pass

Post-Condition: The user successfully logs into the system, navigates to the Planning section, accesses the send Invitation, and the invitation is successfully sent, and the recipient should receive an email with the event details, confirming that the invitation functionality works as expected.

Test Case 4:

Code

```
package Definitions;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class addtask {

    WebDriver driver = null;

    @Given("browser is opensss")
    public void browser_is_open() {
        System.out.println("Inside step - Browser is open");

System.setProperty("webdriver.gecko.marionette","E:\\Testing\\src\\test\\resources\\dr
ivers\\geckodriver.exe");
        driver = new FirefoxDriver();
        driver.manage().window().maximize();
    }

    @And("user is on login pagesss")
    public void user_is_on_login_page() throws Exception {
        driver.navigate().to("http://127.0.0.1:8000/login");
        Thread.sleep(2000);  // Wait for the page to load
    }

    @When("user enters username and passwordsss")
    public void user_enters_username_and_password() throws Throwable {
        driver.findElement(By.id("email")).sendKeys("fathimaps521050@gmail.com");
        driver.findElement(By.id("password")).sendKeys("Fathima@123");
    }

    @And("user clicks on loginsss")
    public void user_clicks_on_login() {
        driver.findElement(By.id("login")).click();
    }

    @Then("user clicks on Planningsss")
    public void user_clicks_on_Planning() {
        driver.findElement(By.id("Planning")).click();
    }

    @And("user clicks on to-do list")
    public void user_clicks_on_todolist() {
        driver.findElement(By.id("addtask")).click();
```

```
    }

    @Then("user clicks on complete")
    public void user_clicks_on_complete() {
        driver.findElement(By.cssSelector(".btn-success")).click();
    }
}
```

Screenshot

```
Scenario: User logs in and adds a task      # src/test/resources/Features/addtask.feature:3
Inside step - Browser is open
  Given browser is opensss                  # Definitions.addtask.browser_is_open()
  And user is on login pagesss              # Definitions.addtask.user_is_on_login_page()
  When user enters username and passwordsss # Definitions.addtask.user_enters_username_and_password()
  And user clicks on loginsss               # Definitions.addtask.user_clicks_on_login()
  Then user clicks on Planningsss           # Definitions.addtask.user_clicks_on_Planning()
  And user clicks on to-do list             # Definitions.addtask.user_clicks_on_todolist()
  Then user clicks on complete              # Definitions.addtask.user_clicks_on_complete()

1 Scenarios (1 passed)
7 Steps (7 passed)
1m12.196s
```

Test report

Test Case 4

Project Name: Dream Knot					
Task Completion					
Test Case ID: Test_4			**Test Designed By: Fathima P.S**		
Test Priority(Low/Medium/High): High			**Test Designed Date: 30-10-2024**		
Module Name: Login Module			**Test Executed By : Mr. T J Jobin**		
Test Title : Task Completion			**Test Execution Date: 30-10-2024**		
Description: User has a valid email/password					
Pre-Condition : User has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	pass
2	Provide valid email	fathimaps521050@gmail.com	User should	User logs in	pass

3	Provide valid password	Fathima@123	be able to login		
4	Click on login button				
5	Click on Planning section		Navigates to the planning section.	Open list in navbar	pass
6	Click on to-do list		To-do list page should be displayed	To-do list displayed	pass
7	Click on complete button		Task should be marked as complete	Task marked complete	pass
Post-Condition: The user successfully logs into the system, navigates to the Planning section, accesses the To-Do List, and marks a task as complete. The system should update the task's status to reflect its completion.					

CHAPTER 6

IMPLEMENTATION

6.1 INTRODUCTION

Implementation is the stage where a planned system transforms into a tangible and operational entity. Building trust and instilling confidence in users is of paramount importance for the success of the system. This is a critical phase that places a strong emphasis on user training and the creation of informative materials. The actual transition often occurs during or after user training. Implementation signifies the act of putting a new system into action after its design phase, turning a conceptual design into a functional one.

Implementation involves the process of transitioning from the old method of operation to the new one, whether it replaces the old system entirely or makes incremental changes. Ensuring that the process is executed accurately is vital to establish a system that aligns well with the organization's requirements. System implementation encompasses the following tasks:

- Meticulous planning.
- Assessment of the existing system and its constraints.
- Designing methods to facilitate the transition

6.2 IMPLEMENTATION PROCEDURES

Software implementation involves the installation of software in its intended location and ensuring that it functions as intended. In some organizations, an individual who is not directly involved in using the software may oversee and approve the project's development. Initially, there may be some skepticism regarding the software, and it's essential to address these concerns to prevent strong resistance.

To ensure a successful transition, several key steps are important:

- Communicating Benefits: Users need to understand why the new software is an improvement over the old one, instilling trust in its capabilities.
- User Training: Providing training to users is crucial to make them feel confident and comfortable using the application.

To evaluate the outcome, it is essential to verify that the server program is running on the server. If the server is not operational, the expected outcomes will not be achieved.

6.2.1 User Training

User training is a critical component of ensuring that individuals know how to use and adapt to a new system. It plays a vital role in fostering user comfort and confidence with the system. Even when a system becomes more complex, the need for training becomes even more apparent. User training covers various aspects, including inputting information, error handling, querying databases, and utilizing tools for generating reports and performing essential tasks. It aims to empower individuals with the knowledge and skills needed to effectively interact with the system.

6.2.2 Training on the Application Software

Once the fundamental computer skills have been covered, the next step is to instruct individuals on using a new software program. This training should provide a comprehensive understanding of the new system, including navigation through screens, accessing help resources, error management, and resolution procedures. The training aims to equip users or groups with the knowledge and skills necessary to effectively utilize the system or its components. It's important to note that training may be tailored differently for various groups of users and individuals in different roles within the organization to cater to their specific needs and requirements.

6.2.3 System Maintenance

Maintaining a system's functionality is a complex challenge in software development. In the maintenance phase of the software life cycle, the software performs critical functions and operates smoothly. Once a system is operational, it requires ongoing care and maintenance to ensure its continued optimal performance. Software maintenance is a crucial aspect of the development process as it ensures that the system can adapt to changes in its environment. Maintenance involves more than just identifying and correcting errors in the code. It encompasses various tasks that contribute to the system's stability and effectiveness.

6.2.4 Hosting

The Dream Knot platform is hosted on Render, a reliable and scalable cloud hosting provider that meets the project's deployment and operational requirements.

- Reasons for Choosing Render:
    - Ease of Deployment: Render offers a streamlined deployment process, allowing easy connection to GitHub repositories. This simplifies application setup, continuous updates, and deployment directly from the codebase.
    - Scalability: With Render's scalable hosting options, Dream Knot can smoothly handle traffic growth and resource demand as the platform expands.

- Backend Support for Django and HTML: While Dream Knot is built with Django (backend) and HTML (frontend), Render efficiently supports this setup, providing a seamless environment for both the application server and static files.

Render

Render is a cloud hosting service that provides a streamlined platform for deploying web applications, APIs, static sites, and databases. It simplifies the process of hosting applications by automating tasks such as server setup, scaling, and load balancing, which allows developers to focus on building their applications rather than managing infrastructure. Render supports various languages and frameworks, including Django, and offers seamless integration with Git for continuous deployment.

Procedure for hosting a website on Render:

Step 1: Login to your render account

Step 2: Create a new project

Step 3: Create a requirements.txt file which will include all your dependencies to be installed.

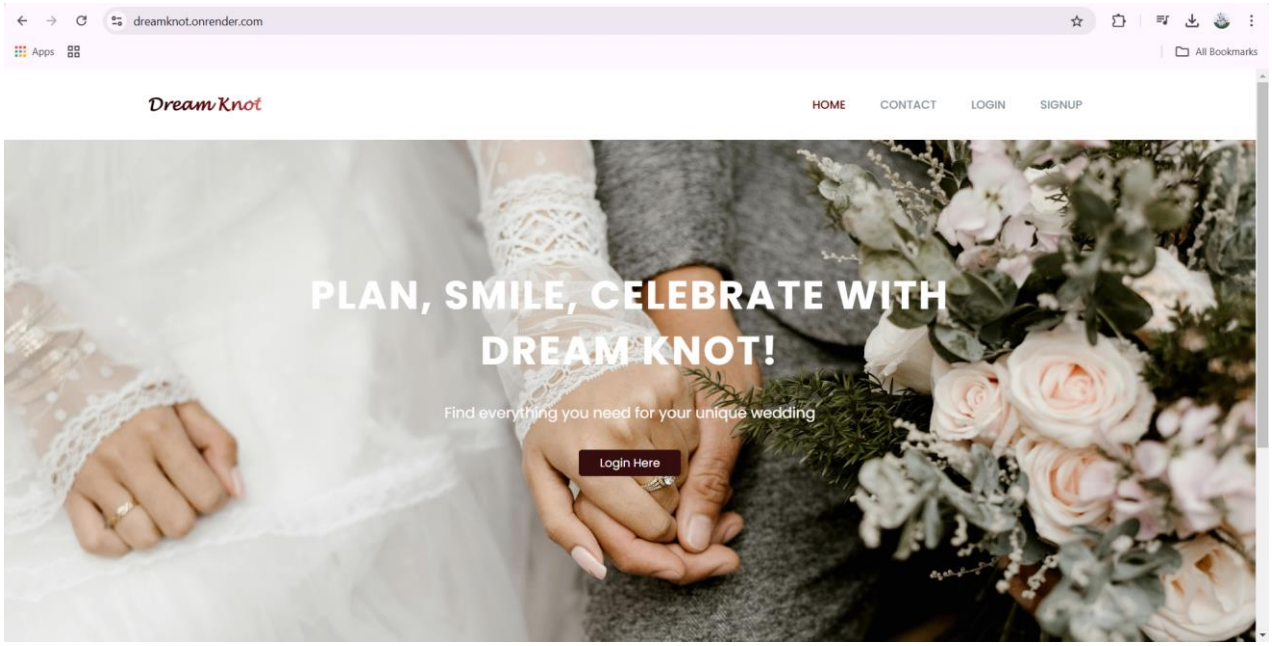
Step 4: Upload the content which is to be hosted into GitHub

Step 5: Change the setting of your project to include the host also and then deploy.**Hosted Website:**

Hosted Link: https://dreamknot.onrender.com

Hosted QR Code:

Screenshot


dreamknot.onrender.com
Apps
All Bookmarks
DreamKnot
HOME
CONTACT
LOGIN
SIGNUP
PLAN, SMILE, CELEBRATE WITH DREAM KNOT!
Find everything you need for your unique wedding
Login Here

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

wedding planning experience for couples and vendors alike. By integrating various functionalities—such as user management, service listings, booking systems, and task management—Dream Knot offers a seamless solution that simplifies the intricacies of organizing a wedding. The platform enhances communication between couples and service providers, facilitates efficient booking processes, and ensures users stay organized with customizable tools and features.

As an all-in-one wedding planning solution, Dream Knot not only empowers couples to manage their wedding journey with confidence but also provides vendors with the necessary tools to effectively showcase their services and engage with clients. The platform's emphasis on usability, security, and modular design positions it as a valuable resource in the wedding planning industry.

7.2 FUTURE SCOPE

Future Prospects Augmented Reality (AR) **Planning Tools**: Incorporate AR functionalities that enable couples to visualize their wedding venue and decor selections in real-time. Users would have the capability to position virtual items, such as floral arrangements, table settings, and decorations, within their actual venue to assess how these elements harmonize. **Virtual Wedding Planning Assistants:** Deploy chatbots or virtual assistants that assist users throughout the wedding planning journey, addressing inquiries, providing task reminders, and recommending vendors based on user preferences and prior interactions. **Personalized Wedding Websites:** Enable couples to design customized wedding websites directly through the platform. These websites could feature RSVP management, event information, and even a blog to document their planning experiences for guests. **Crowdsourced Inspiration Gallery:** Establish a community-driven inspiration gallery where users can contribute their wedding ideas, photographs, and experiences. This feature would promote engagement and offer innovative concepts for couples in the planning stages. **Loyalty and Referral Programs**: Develop a loyalty program that rewards users for securing services through the platform. Introduce referral incentives that allow couples to earn discounts or credits for recommending friends to utilize Dream Knot. **Integration with Local Services**: Collaborate with local businesses to offer couples exclusive deals on related services, such as catering, floral arrangements, or transportation. This could also encompass local experiences for pre-wedding events.

CHAPTER 8
BIBLIOGRAPHY

REFERENCES:

- "HTML and CSS: Design and Build Websites" by Duckett, Jon..
- "Django for Beginners: Build Websites with Python and Django" by Vincent, William S.
- "JavaScript and jQuery: Interactive Front-End Web Development" by Duckett, Jon.
- "Scrum: The Art of Doing Twice the Work in Half the Time" by Sutherland, Jeff.
- "The Principles of Beautiful Web Design" by Beaird, Jason.
- Johnson, Emily. "The Impact of Social Media on Wedding Planning: Trends and Insights." *Journal of Wedding Studies*, vol. 15, no. 2, 2021, pp. 45-62.

WEBSITES:

- The Knot: https://www.theknot.com
- WeddingWire: https://www.weddingwire.com
- Bridal Guide: https://www.bridalguide.com
- WedMeGood: https://www.wedmegood.com
- Bridebook: https://www.bridebook.co.uk

CHAPTER 9

APPENDIX

9.1 Sample Code

Login Page

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!-- CSS Files -->
    <link rel="stylesheet" href="{% static 'css/login.css' %}">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-
beta3/css/all.min.css">

    <title>Login Page - Dream Knot</title>
</head>
<body style="background-image: url('{% static 'images/bg1.png' %}');">

    <!-- Header Section -->
    <header>
        <div class="logo">
            <img src="{% static 'images/logo1.png' %}" alt="Dream Knot Logo">
        </div>
        <nav class="navbar1">
            <a href="#">Dream Knot</a>
        </nav>
    </header>
```

```
<!-- Main Wrapper -->
<div class="wrapper">
    <div class="container main">
        <div class="row">

            <!-- Left Side Image -->
            <div class="col-md-6 side-image" style="background-image: url('{% static
'images/logincp.jpg' %}');">
                <div class="text">
                    <p>Join us and let’s craft your perfect day together <i>- Dream Knot</i></p>
                </div>
            </div>

            <!-- Right Side Form -->
            <div class="col-md-6 right">
                <div class="input-box">
                    <h4>Log in to your account</h4>

                    <!-- Display Messages -->
                    {% if messages %}
                    <div class="alert alert-danger">
                        {% for message in messages %}
                            <p>{{ message }}</p>
                        {% endfor %}
                    </div>
                {% endif %}

                    <form method="post">
                        {% csrf_token %}

                        <!-- Social Login Buttons -->
                        <div class="social-login">
                            <!-- <button type="button" class="social-btn fb-btn">
                                <img src="{% static 'images/facebook.png' %}" alt="Facebook"
```

```
style="height: 18px; margin-right: 10px;">
                        Sign in with Facebook
                    </button> -->
                    <!-- Google login -->
                <a href="{% url 'social:begin' 'google-oauth2' %}" class="social-btn google-btn">
                    <button type="button" id="googleSignInButton">
                        <img src="{% static 'images/google.png' %}" alt="Google" style="height:
18px; margin-right: 10px;">
                        Sign in with Google
                    </button>
                </a>
                </div>
                <!-- Divider Text -->
                <div class="signin">
                    <span>Or log in with your email</span>
                </div>

                <!-- Email Input -->
<div class="input-field">
    <input type="email" class="input" id="email" name="email" value="{{ email }}" required >
    <label for="email">Email</label>
</div>
                <!-- Password Input -->
<div class="input-field">
    <input type="password" class="input" id="password" name="password" value="{{ password
}}" required>
    <label for="password">Password</label>
    <i class="fa fa-eye" id="togglePassword" style="cursor: pointer; position: absolute; right: 10px;
top: 15px;"></i>
</div>
                <!-- Submit Button -->
                <div class="input-field">
                    <input type="submit" class="submit" id="login" value="Login">
                </div>
```

```
                </form>
                <!-- Forgot Password Link -->
                <div class="forgetpswd">
                    <span><a href="{% url 'forgotpass' %}">Forgot password?</a></span>
                </div>
                <div class="forgetpswd">
                    <span><a href="{% url 'admin:login' %}" class="admin-login-button">Login as
Admin</a>.</span>
                </div>
                <!-- Signup Link -->
                <div class="signin">
                    <span>Don't have an account? <a href="{% url 'signup' %}">Signup
here</a></span>
                </div>
            </div>
        </div>
    </div>
</div>

<!-- Bootstrap JS and Dependencies -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
cVKXjLU5BNTuUJK2JEMLwtNCLuyPmWHnoAhjtTHeR7oAZ1hZGWuiTimKfkYAUs2L"
crossorigin="anonymous"></script>
<script>
    const togglePassword = document.querySelector('#togglePassword');
    const password = document.querySelector('#password');
    togglePassword.addEventListener('click', function () {
        // Toggle the type attribute between 'password' and 'text'
        const type = password.getAttribute('type') === 'password' ? 'text' : 'password';
        password.setAttribute('type', type);

        // Toggle between eye and eye-slash icons
```

```
        this.classList.toggle('fa-eye-slash');
    });
  </script>
</body>
</html>
```

Views function

```
from django.contrib.auth.hashers import check_password
from django.shortcuts import redirect, render
from django.contrib import messages
from .models import UserSignup

def login_view(request):
  if request.method == 'POST':
    email = request.POST.get('email')
    password = request.POST.get('password')
    try:
      user = UserSignup.objects.get(email=email)
      print("User found:", user.name)
      print("User Status:", user.status)  # Debugging

      if not user.status:
        messages.error(request, "Your account is deactivated. Please contact the admin.")
        print("Deactivated user attempted to log in.")
        return redirect('login')

      # Check if the user is verified
      if not user.is_verified:
        messages.error(request, "Please verify your email before logging in. Check your inbox for the verification link.")
        return redirect('login')

      if check_password(password, user.password):
```

```
            request.session['user_id'] = user.id
            request.session['user_role'] = user.role
            request.session['user_name'] = user.name

            messages.success(request, "Login successful!")
            if user.role == 'admin':
                return redirect('admin_dashboard')
            elif user.role == 'vendor':
                return redirect('vendor_home')
            else:
                return redirect('user_home')
        else:
            messages.error(request, "Invalid email or password.")
            print("Incorrect password for user:", user.name)  # Debugging
            return redirect('login')

    except UserSignup.DoesNotExist:
        messages.error(request, "Invalid email or password.")
        print("User not found for email:", email)  # Debugging
        return redirect('login')

return render(request, 'dreamknot1/login.html')
```

Other view functions from user module

```
def update_user_profile(request):
    if not request.session.get('user_id'):
        return redirect('login')
    user = UserSignup.objects.get(id=request.session['user_id'])
    user_profile, created = UserProfile.objects.get_or_create(user=user)

    if request.method == 'POST':
        name = request.POST.get('name')
        email = request.POST.get('email')
        phone = request.POST.get('phone')
```

```
wedding_date = request.POST.get('wedding_date')
event_held = request.POST.get('event_held')
country = request.POST.get('country')
state = request.POST.get('state')
place = request.POST.get('place')
new_password = request.POST.get('new_password')
confirm_password = request.POST.get('confirm_password')
errors = {}
# Validate email
if not re.match(r'^[a-zA-Z0-9_.+-]+@gmail\.com$', email):
    messages.error(request, "Please enter a valid Gmail address.")
    errors['email'] = "Invalid Gmail address."
# Validate wedding_date
if wedding_date:
    wedding_date_parsed = date.fromisoformat(wedding_date)
    if wedding_date_parsed < date.today():
        errors['wedding_date'] = "Wedding date cannot be in the past."

if not name:
    errors['name'] = 'Name is required.'
if not phone or not re.match(r'^\+?1?\d{10}$', phone):
    errors['phone'] = 'Enter a valid phone number with 10 digits.'

if new_password or confirm_password:
    if new_password != confirm_password:
        errors['password'] = 'Passwords do not match.'
    if len(new_password) < 8:
        errors['password_length'] = "Password must be at least 8 characters long."
    if not re.search(r'[A-Z]', new_password):
        errors['password_uppercase'] = "Password must contain at least one uppercase letter."
    if not re.search(r'[a-z]', new_password):
        errors['password_lowercase'] = "Password must contain at least one lowercase letter."
    if not re.search(r'[0-9]', new_password):
        errors['password_digit'] = "Password must contain at least one digit."
```

```
        if not re.search(r'[!@#$%^&*(),.?":{}|<>]', new_password):
            errors['password_special'] = "Password must contain at least one special character."

    if errors:
        return render(request, 'dreamknot1/update_user_profile.html', {
            'profile': user_profile,
            'errors': errors,
            'name': name,
            'email': email,
            'phone': phone,
            'wedding_date': wedding_date,
            'event_held': event_held,
            'country': country,
            'state': state,
            'place': place,
            'countries': countries,  # Pass list of countries for dropdown
        })

    # Update user details
    user.name = name
    user.email = email
    user.phone = phone
    user.country = country
    user.state = state
    user.place = place

    if new_password and confirm_password and new_password == confirm_password:
        user.password = make_password(new_password)

    user.save()

    # Update user profile
    user_profile.wedding_date = wedding_date
    user_profile.event_held = event_held   # Save the event_held status
```

```
        user_profile.status = True
        user_profile.updated_at = timezone.now()
        user_profile.save()

        messages.success(request, "Profile updated successfully!")
        return redirect('user_home')

    return render(request, 'dreamknot1/update_user_profile.html', {
        'profile': user_profile,
        'name': user.name,
        'email': user.email,
        'phone': user.phone,
        'wedding_date': user_profile.wedding_date,
        'event_held': user_profile.event_held,  # Pre-populate event_held status
        'country': user.country,  # Pre-populate country
        'state': user.state,      # Pre-populate state
        'place': user.place,      # Pre-populate place
        'countries': countries,   # Pass list of countries
    })


# current month todo list
# user view for current month todo list
def current_month_todolist(request):
    user_id = request.session.get('user_id')
    if not user_id:
        messages.error(request, "You must be logged in to view your tasks.")
        return redirect('login')

    user_instance = get_object_or_404(UserSignup, id=user_id)
    user_profile = get_object_or_404(UserProfile, user=user_instance)
    user_name = user_instance.name  # Assuming the username is stored in the 'name' field

    if not user_profile.wedding_date:
```

```
        messages.warning(request, "Please set your wedding date to view tasks.")
        return redirect('profile_update')

    today = timezone.now().date()
    wedding_date = user_profile.wedding_date
    remaining_days = (wedding_date - today).days

    if remaining_days > 180:
        current_month = '6-12'
        next_month = '4-6'
        days_until_next_month = remaining_days - 180
    elif 120 < remaining_days <= 180:
        current_month = '4-6'
        next_month = '2-4'
        days_until_next_month = remaining_days - 120
    elif 60 < remaining_days <= 120:
        current_month = '2-4'
        next_month = '1-2'
        days_until_next_month = remaining_days - 60
    elif 30 < remaining_days <= 60:
        current_month = '1-2'
        next_month = '1-2 Weeks'
        days_until_next_month = remaining_days - 30
    elif 14 < remaining_days <= 30:
        current_month = '1-2 Weeks'
        next_month = 'Final Days'
        days_until_next_month = remaining_days - 14
    else:
        current_month = 'Final Days'
        next_month = 'Wedding Day'
        days_until_next_month = remaining_days

    # Get user-specific tasks for the current month
    user_tasks = WeddingTask.objects.filter(user=user_instance, task_month=current_month)
```

```
]
    # Prepare the HTML email content
    html_message = render_to_string('dreamknot1/email_invitation.html', {
        'couple_name': couple.name,
        'event_name': event_name,
        'event_date': event_date,
        'event_time': event_time,
        'event_description': event_description,
        'venue': venue,
        'venue_address': venue_address,
        'phone_number': phone_number,
        'location_link': location_link,
        'guest_name': guest_name,  # Customize the email for each guest
    })

    # Send the HTML email to each guest
    send_mail(
        f"Invitation to {event_name} from {couple.name}",
        '',
        'noreply@dreamknot.com',
        [guest_email],
        html_message=html_message,
        fail_silently=False,
    )

# User Dashboard - View Vendor Services, Book, Rate, Favorite
def user_dashboard(request):
    user_name = request.session.get('user_name', 'user')
    if not user_name:
        messages.error(request, "You must be logged in to view this page.")
        return redirect('login')

    # Fetch the user from session
```

```
try:
    user = UserSignup.objects.get(name=user_name)
except UserSignup.DoesNotExist:
    return HttpResponse("User not found.")

# Fetch active services
services = Service.objects.filter(status=1, availability=True)

# Filter services by category
if 'category' in request.GET:
    category = request.GET['category']
    services = services.filter(category=category)

# Search services
if 'search' in request.GET:
    search_query = request.GET['search']
    services = services.filter(name__icontains=search_query)

# Get vendors and related services
vendor_services = {}
for service in services:
    vendor = service.vendor
    vendor_services.setdefault(vendor, []).append(service)

# Fetch favorite services for the logged-in user
favorites = Favorite.objects.filter(user=user).select_related('service')

# Fetch bookings for the logged-in user
bookings = Booking.objects.filter(user=user).select_related('service')

return render(request, 'dreamknot1/user_dashboard.html', {
    'vendor_services': vendor_services,
    'user_name': user_name,
    'favorites': favorites,
```

```
        'bookings': bookings,
    })

def service_detail(request, service_id):
    user_name = request.session.get('user_name', '')
    if not user_name:
        messages.error(request, "You must be logged in to view this page.")
        return redirect('login')

    service = get_object_or_404(Service, id=service_id)
    vendor_phone = service.vendor.user.phone

    # Get category-specific details
    category_details = None
    if service.category == 'Venue':
        category_details = VenueService.objects.filter(service=service).first()
    elif service.category == 'Catering':
        category_details = CateringService.objects.filter(service=service).first()
    elif service.category == 'Photography':
        category_details = PhotographyService.objects.filter(service=service).first()
    elif service.category == 'MusicEntertainment':
        category_details = MusicEntertainmentService.objects.filter(service=service).first()
    elif service.category == 'MakeupHair':
        category_details = MakeupHairService.objects.filter(service=service).first()
    elif service.category == 'Rentals':
        category_details = RentalsService.objects.filter(service=service).first()
    elif service.category == 'MehendiArtist':
        category_details = MehendiArtistService.objects.filter(service=service).first()
    elif service.category == 'Decoration':
        category_details = DecorationService.objects.filter(service=service).first()

    context = {
        'service': service,
        'vendor_phone': vendor_phone,
```

```
        'category_details': category_details,
        'user_name': user_name
    }

    return render(request, 'dreamknot1/service_detail.html', context)

# user view for booking calendar
# View to render the calendar and form page
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def booking_calendar_view(request, service_id):
    service = Service.objects.get(id=service_id)
    return render(request, 'dreamknot1/booking_calendar.html', {'service': service})

# API view to fetch booked and available slots for FullCalendar
from django.http import JsonResponse
from django.utils.timezone import now, timedelta

def get_booking_slots(request, service_id):
    # Fetch bookings for the given service
    bookings = Booking.objects.filter(service_id=service_id)
    events = []

    # Define status-to-color mapping
    status_color_mapping = {
        0: ('Pending', 'orange'),  # Vendor has not confirmed
        1: ('Confirmed', 'green'),  # Vendor confirmed the booking
        2: ('Completed', 'blue'),  # Booking is marked as completed
        3: ('Canceled', 'red'),  # Booking was canceled
    }

    # Add booked slots to the events list
    for booking in bookings:
        event_status, color = status_color_mapping.get(booking.book_status, ('Unknown', 'gray'))
        events.append({
```

```
            'title': f'{event_status}',
            'start': booking.event_date.isoformat(),  # ISO format for date compatibility
            'color': color,
            'status': event_status
        })

    # Optionally: Add available slots logic (e.g., next 30 days)
    # Assuming "available" means dates with no bookings in the next 30 days
    today = now().date()
    future_dates = [today + timedelta(days=i) for i in range(365)]  # Next 30 days
    booked_dates = bookings.values_list('event_date', flat=True)  # List of already booked dates

    for date in future_dates:
        if date not in booked_dates:
            events.append({
                'title': 'Available',  # Mark as available
                'start': date.isoformat(),
                'color': 'lightgray',  # Use light gray for available slots
                'status': 'Available'
            })

    return JsonResponse(events, safe=False)
# user view for booking details
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def user_booking_details(request):
    user_name = request.session.get('user_name')

    if not user_name:
        return redirect('login')

    user_signup = get_object_or_404(UserSignup, name=user_name)
    bookings = Booking.objects.filter(user=user_signup).select_related('service')

    today = timezone.now().date()
```

```
for booking in bookings:
    booking.days_until_event = (booking.event_date - today).days
    booking.is_refundable = booking.days_until_event > 30

if request.method == 'POST':
    booking_id = request.POST.get('booking_id')
    cancellation_reason = request.POST.get('cancellation_reason')

    booking = get_object_or_404(Booking, id=booking_id, user=user_signup)

    # Check if cancellation is within 30 days of the event
    days_until_event = (booking.event_date - today).days

    if days_until_event > 30:
        # Cancellation is more than 30 days before the event
        booking.book_status = 3
        booking.canceled_by_user = True
        booking.cancellation_reason = cancellation_reason
        booking.vendor_confirmed_at = None
        booking.save()

        # Send email to service provider
        send_mail(
            subject=f"Booking Cancellation: {booking.service.name}",
            message=f"""
            Dear Service Provider,

            A booking for your service has been canceled.

            Details:
            - Service: {booking.service.name}
            - Event Date: {booking.event_date}
            - Cancellation Reason: {cancellation_reason}
```

The booking was canceled more than 30 days before the event, so a full refund will be processed.

```
        Best regards,
        Dream Knot Team
        """,
        from_email=settings.DEFAULT_FROM_EMAIL,
        recipient_list=[booking.service.vendor.user.email],
        fail_silently=False,
    )

    messages.success(request, "Your booking has been canceled successfully. A full refund will be processed.")
else:
    # Cancellation is within 30 days of the event
    booking.book_status = 3
    booking.canceled_by_user = True
    booking.cancellation_reason = cancellation_reason
    booking.vendor_confirmed_at = None
    booking.save()

    # Send email to service provider
    send_mail(
        subject=f"Booking Cancellation: {booking.service.name}",
        message=f"""
        Dear Service Provider,
        A booking for your service has been canceled.
        Details:
        - Service: {booking.service.name}
        - Event Date: {booking.event_date}
        - Cancellation Reason: {cancellation_reason}
```

The booking was canceled within 30 days of the event, so the booking amount is non-

refundable.

Best regards,
Dream Knot Team
""",
from_email=settings.DEFAULT_FROM_EMAIL,
recipient_list=[booking.service.vendor.user.email],
fail_silently=False,
)

messages.warning(request, "Your booking has been canceled. However, as it's within 30 days of the event, the booking amount is non-refundable.")

return redirect('user_booking_details')

return render(request, 'dreamknot1/user_booking_details.html', {
'bookings': bookings,
'user_name': user_name,
'today': today,
})

user view for book service
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def book_service(request, service_id):
user_id = request.session.get('user_id')

if not user_id:
messages.error(request, "You must be logged in to book a service.")
return redirect('login')

try:
service = get_object_or_404(Service, id=service_id)
user = UserSignup.objects.get(id=user_id)
user_name = user.name # Fetch the username (assuming it's stored in the 'name' field)

```
    except UserSignup.DoesNotExist:
        messages.error(request, "User profile not found. Please log in again.")
        return redirect('login')
    except Service.DoesNotExist:
        messages.error(request, "The requested service does not exist.")
        return redirect('user_dashboard')

    try:
        user_profile = UserProfile.objects.get(user=user)
    except UserProfile.DoesNotExist:
        user_profile = None

    if request.method == "POST":
        # Retrieve form data
        event_name = request.POST.get('event_name')
        event_date = request.POST.get('event_date')
        event_address = request.POST.get('event_address') if service.category != 'Venue' else
service.city
        user_address = request.POST.get('user_address')
        num_days = int(request.POST.get('num_days', 1))
        additional_requirements = request.POST.get('additional_requirements')
        agreed_to_terms = request.POST.get('agreed_to_terms') == 'on'


        # Check if there's an existing booking for the same service on the same day
        existing_booking = Booking.objects.filter(
            Q(service=service) &
            Q(event_date=event_date) &
            ~Q(book_status=3)  # Exclude cancelled bookings
        ).exists()

        if existing_booking:
            messages.error(request, "This date is already booked for this service. Please choose
another date.")
```

```
            return redirect('book_service', service_id=service_id)

        # Create new booking
        booking = Booking(
            user=user,
            service=service,
            event_name=event_name,
            event_date=event_date,
            event_address=event_address,
            user_address=user_address,
            num_days=num_days,
            additional_requirements=additional_requirements,
            user_agreed_to_terms=agreed_to_terms,
            user_agreement_date=timezone.now() if agreed_to_terms else None

        )
        booking.save()  # This will trigger the save method in the Booking model

        # Handle reference images
        reference_images = request.FILES.getlist('reference_images')
        for image in reference_images:
            ref_image = ReferenceImage.objects.create(image=image)
            booking.reference_images.add(ref_image)

        if booking.user_agreed_to_terms:
            messages.success(request, f"Booking submitted successfully! Total amount:
₹{booking.total_amount}")
        else:
            messages.warning(request, f"Booking submitted, but terms were not agreed to. Total
amount: ₹{booking.total_amount}")

        return redirect('user_booking_details')
    # For GET requests
    selected_date = request.GET.get('selected_date')
```

```
# For GET requests, render the booking form
context = {
    'service': service,
    'user': user,
    'user_profile': user_profile,
    'terms_and_conditions': Booking().get_default_terms_and_conditions(),
    'user_name': user_name,
    'selected_date': selected_date,  # Add this line
    'is_venue': service.category == 'Venue',
    'venue_city': service.city if service.category == 'Venue' else None,
}
return render(request, 'dreamknot1/book_service.html', context)
```

9.2 Screen Shots

9.2.1 Landing Page

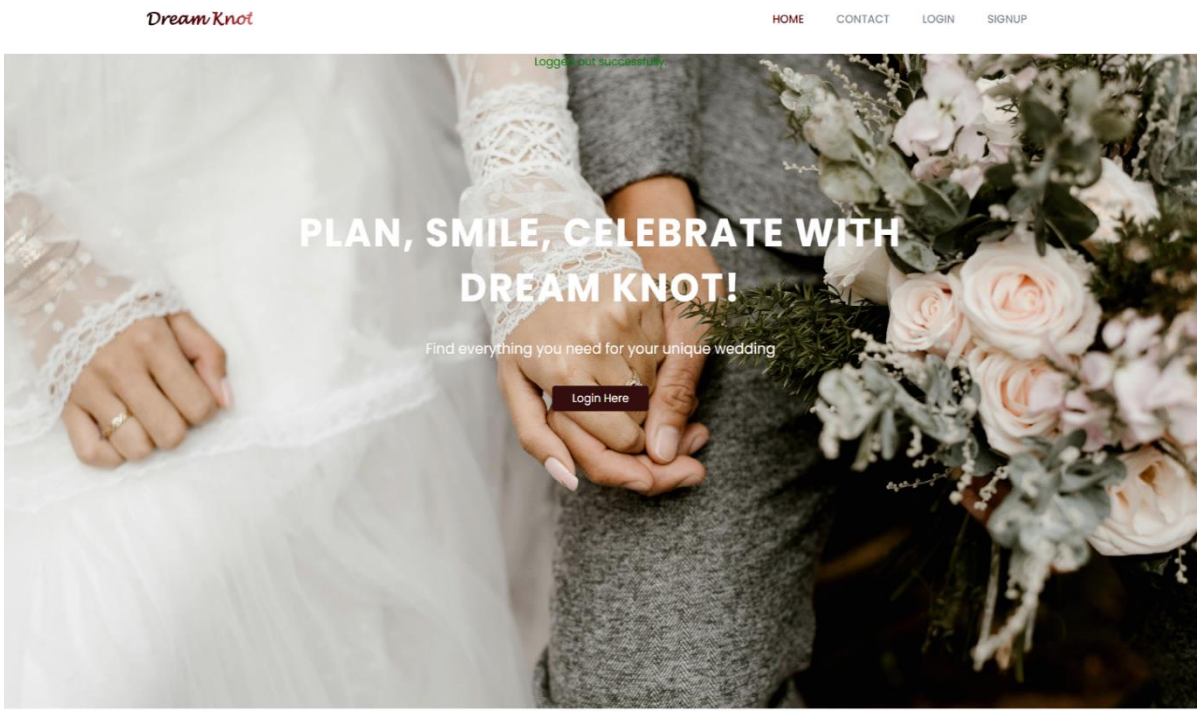


Figure 9.2.1 Login page

9.2.2 Registration Page

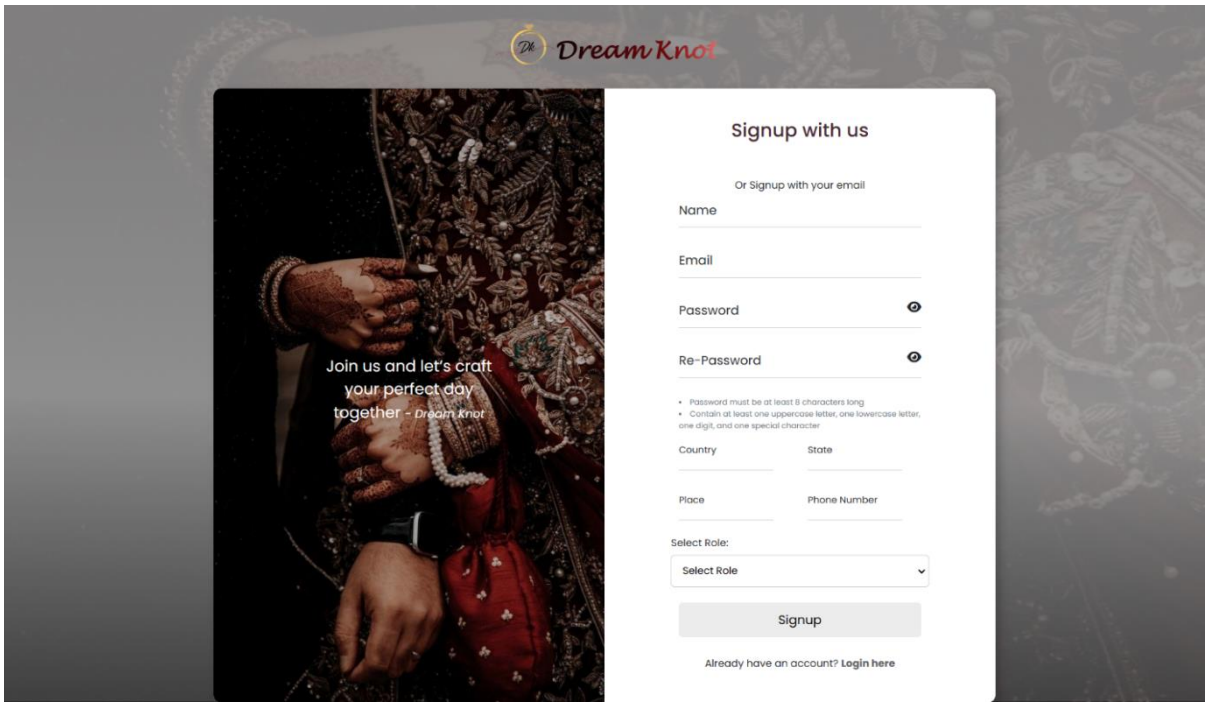


Figure 9.2.2 Registration Page

9.2.3 Login Page

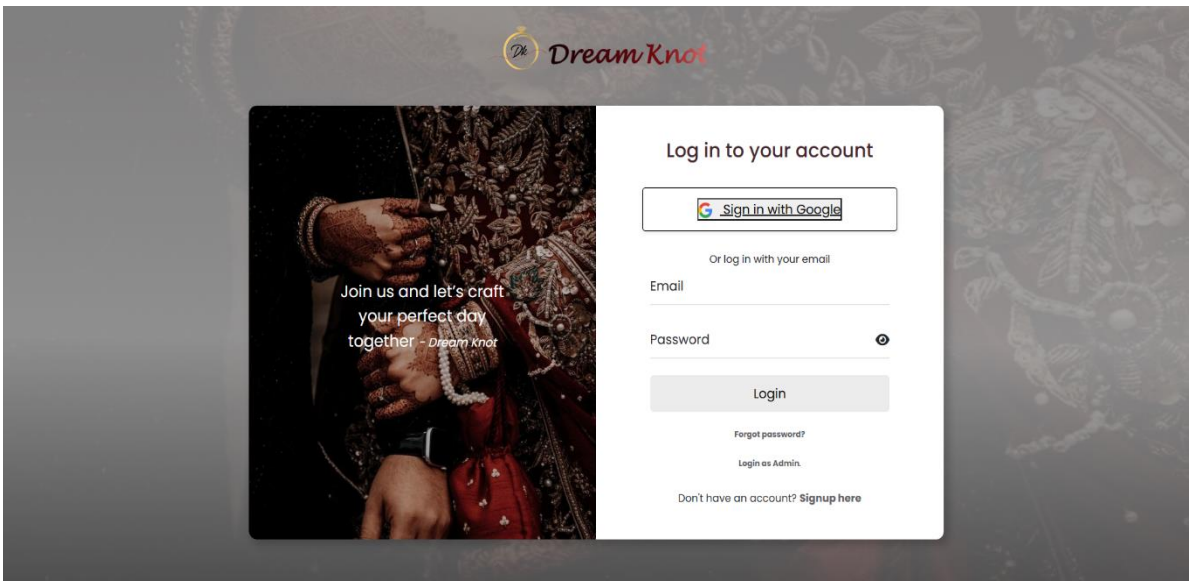
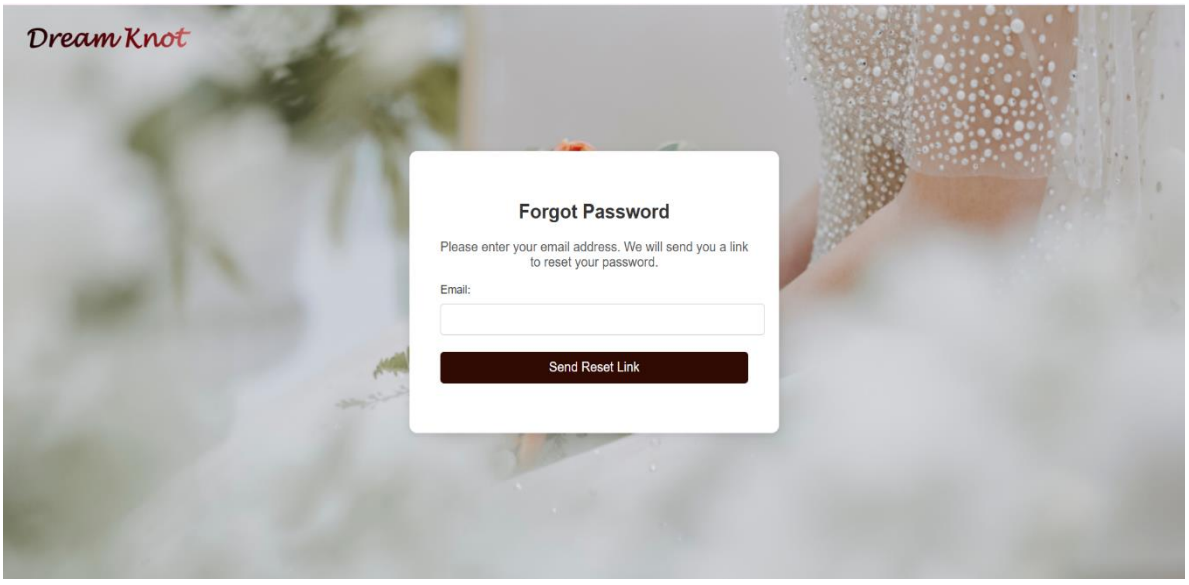


Figure 9.2.3 Login page

9.2.4 Forget password Page



9.2.4 Forget password Page

9.2.5 Home Page

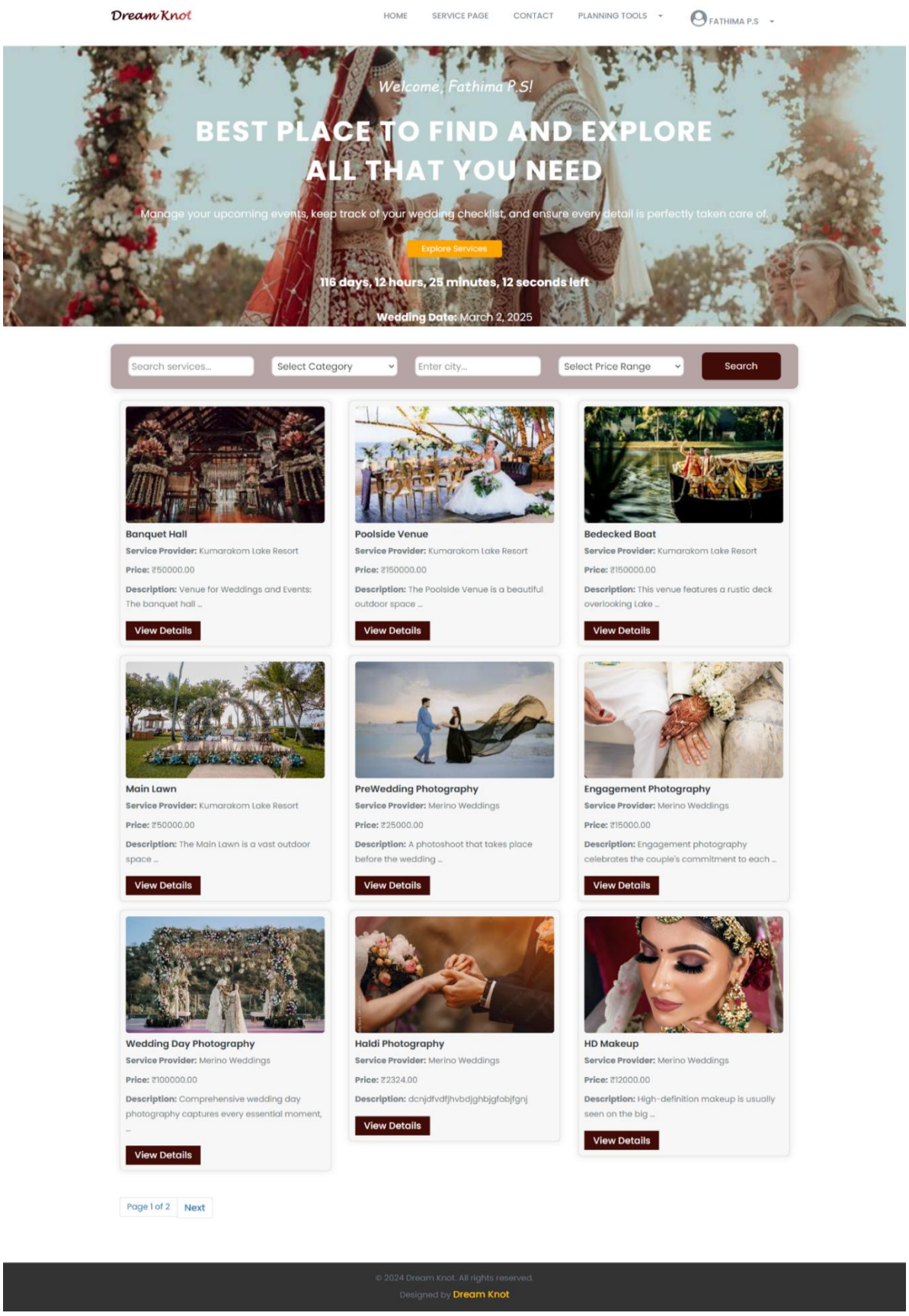


Figure 9.2.5 Home Page

9.2.6 Service Page

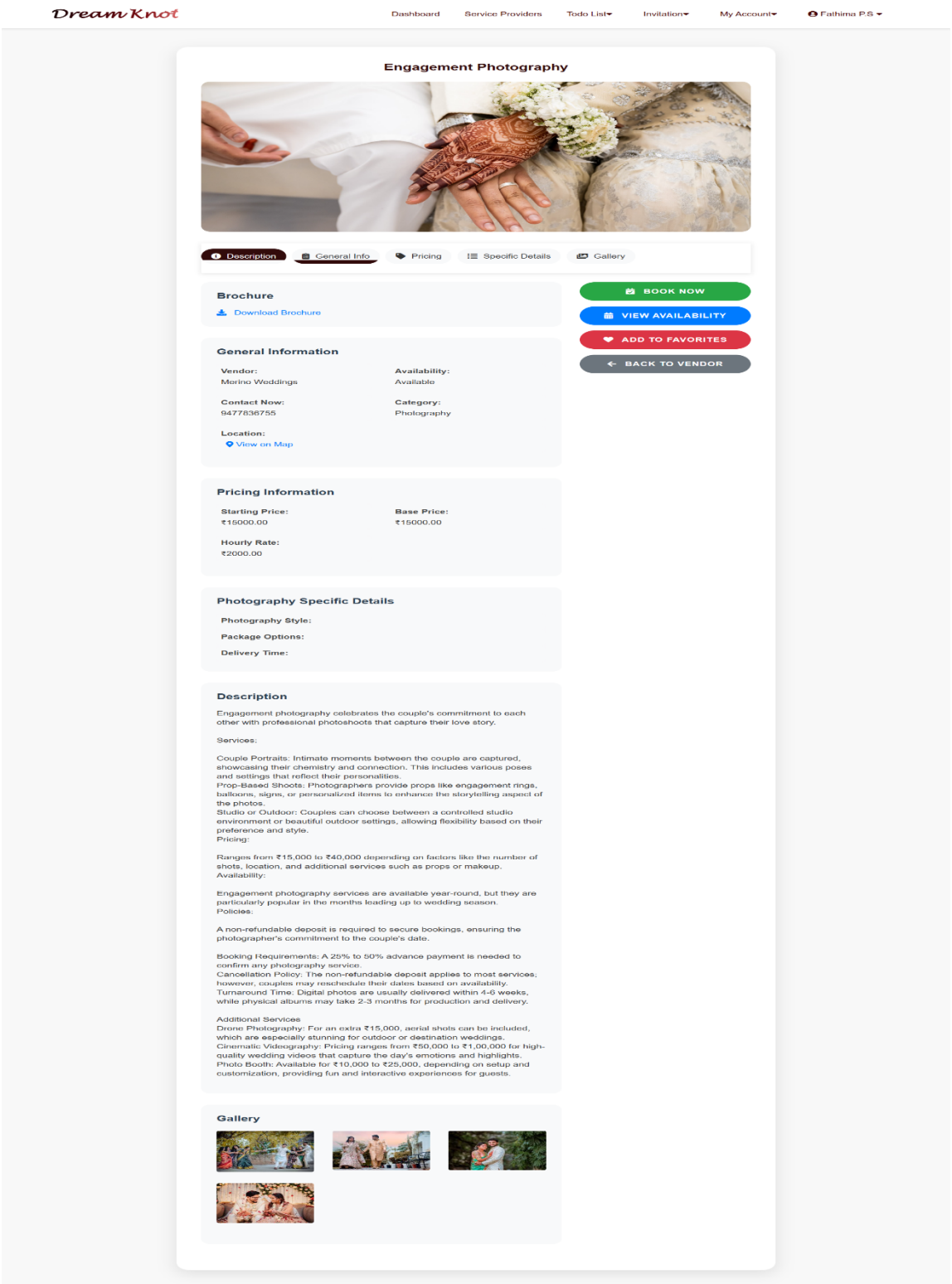


Figure. 9.2.6 Service Page

9.2.7 Booking Page

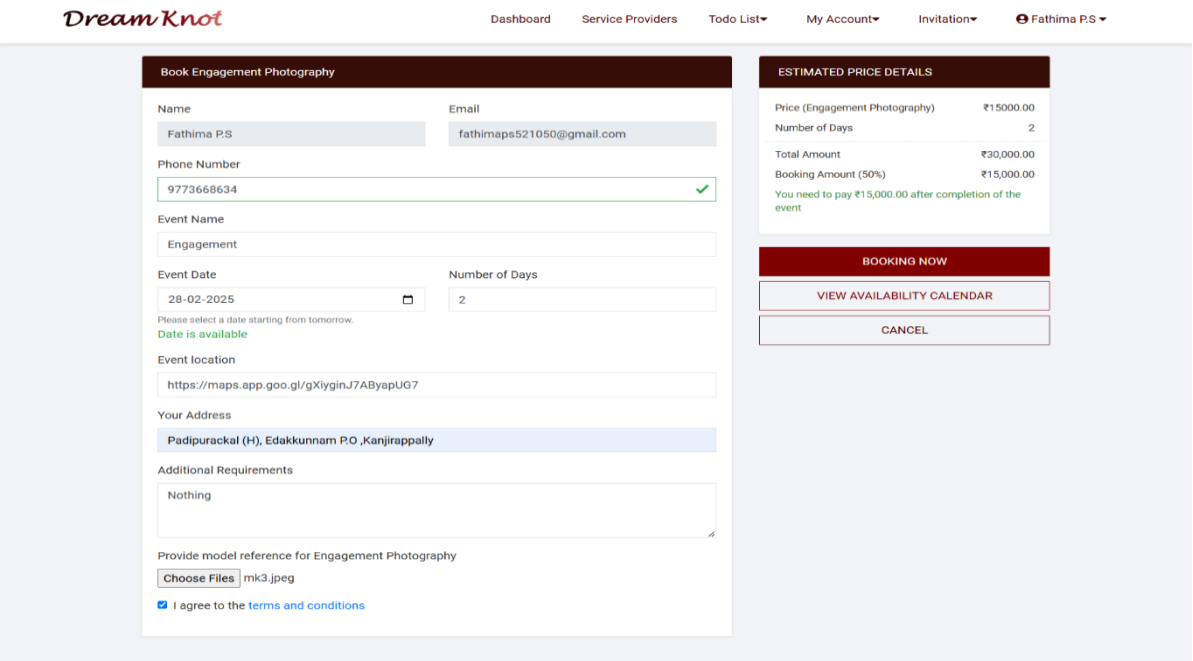


Figure 9.2.7 Booking Page

9.2.8 Availability calender view Page

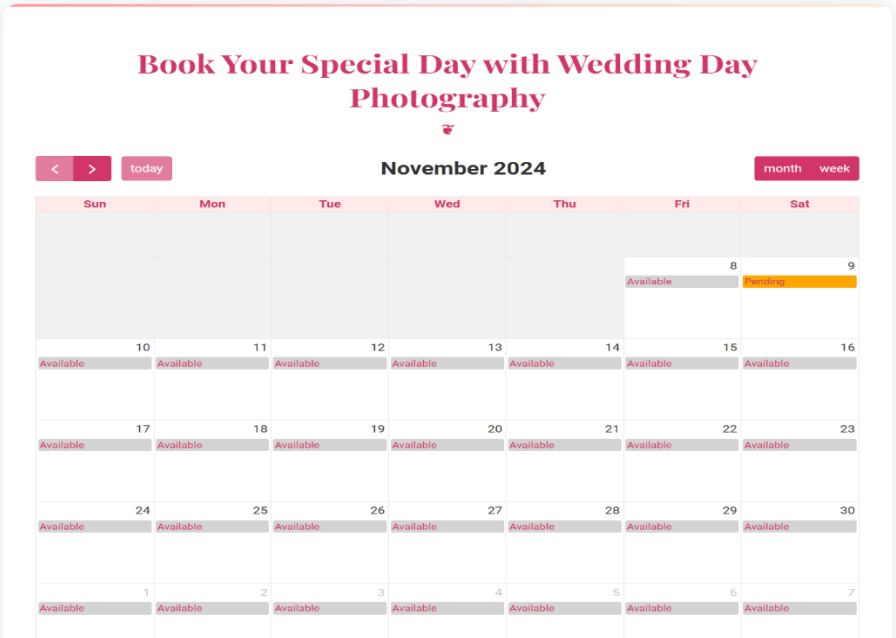


Figure 9.2.8 Availability calender view Page

9.2.9 Checkout Page

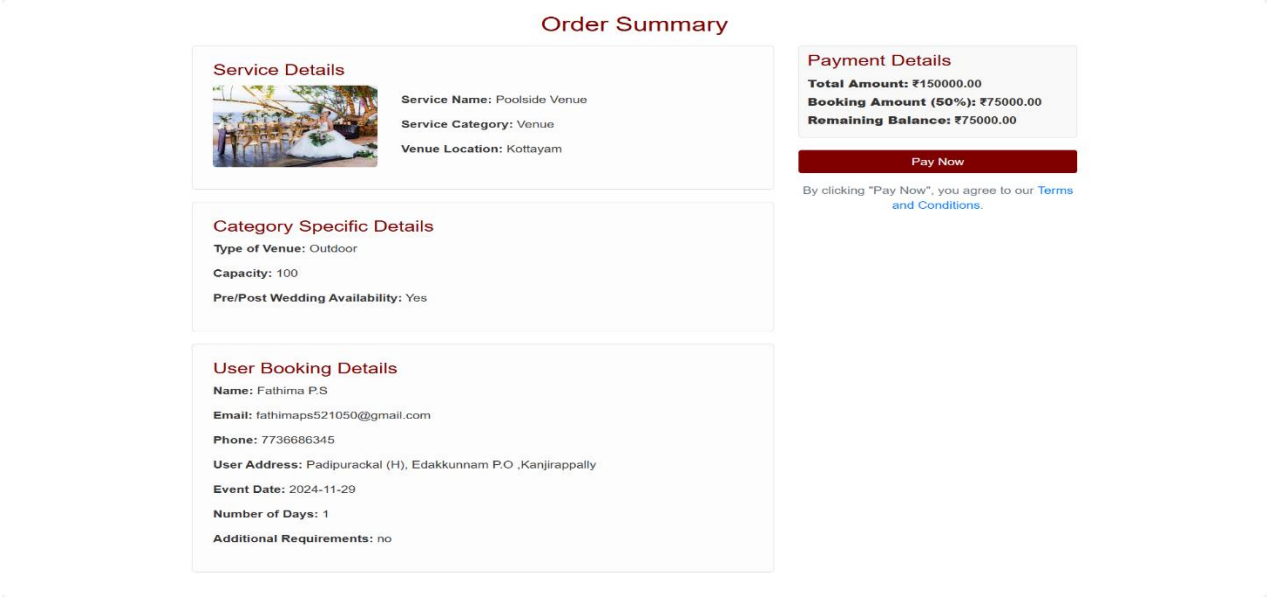


Figure 9.2.9 Checkout Page

9.2.10 Payment Page

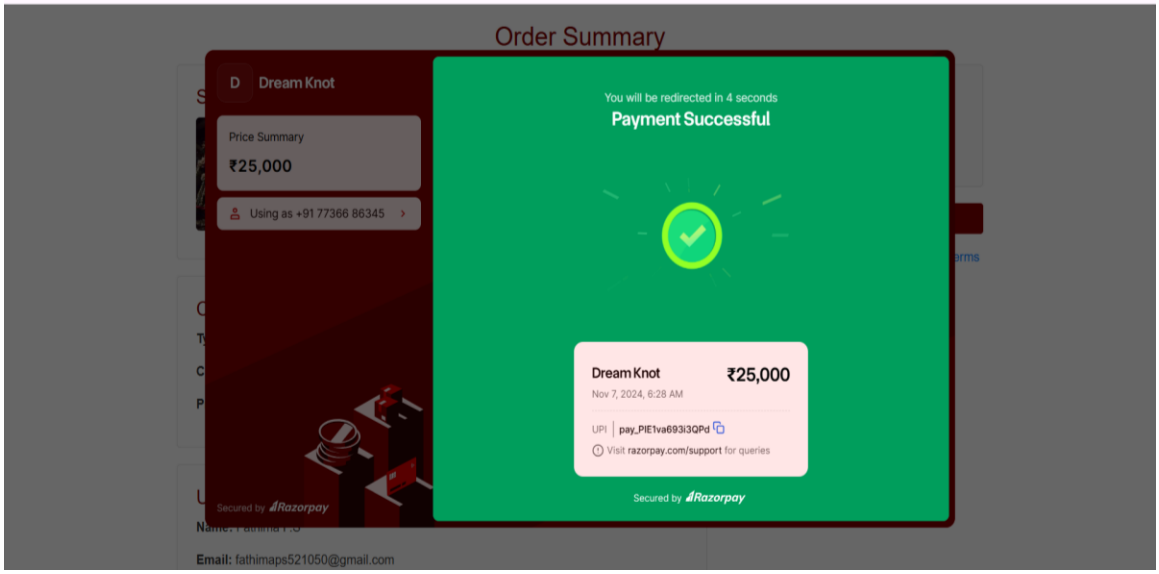


Figure 9.2.10 Payment Page

9.2.11 Current Mont task list Page

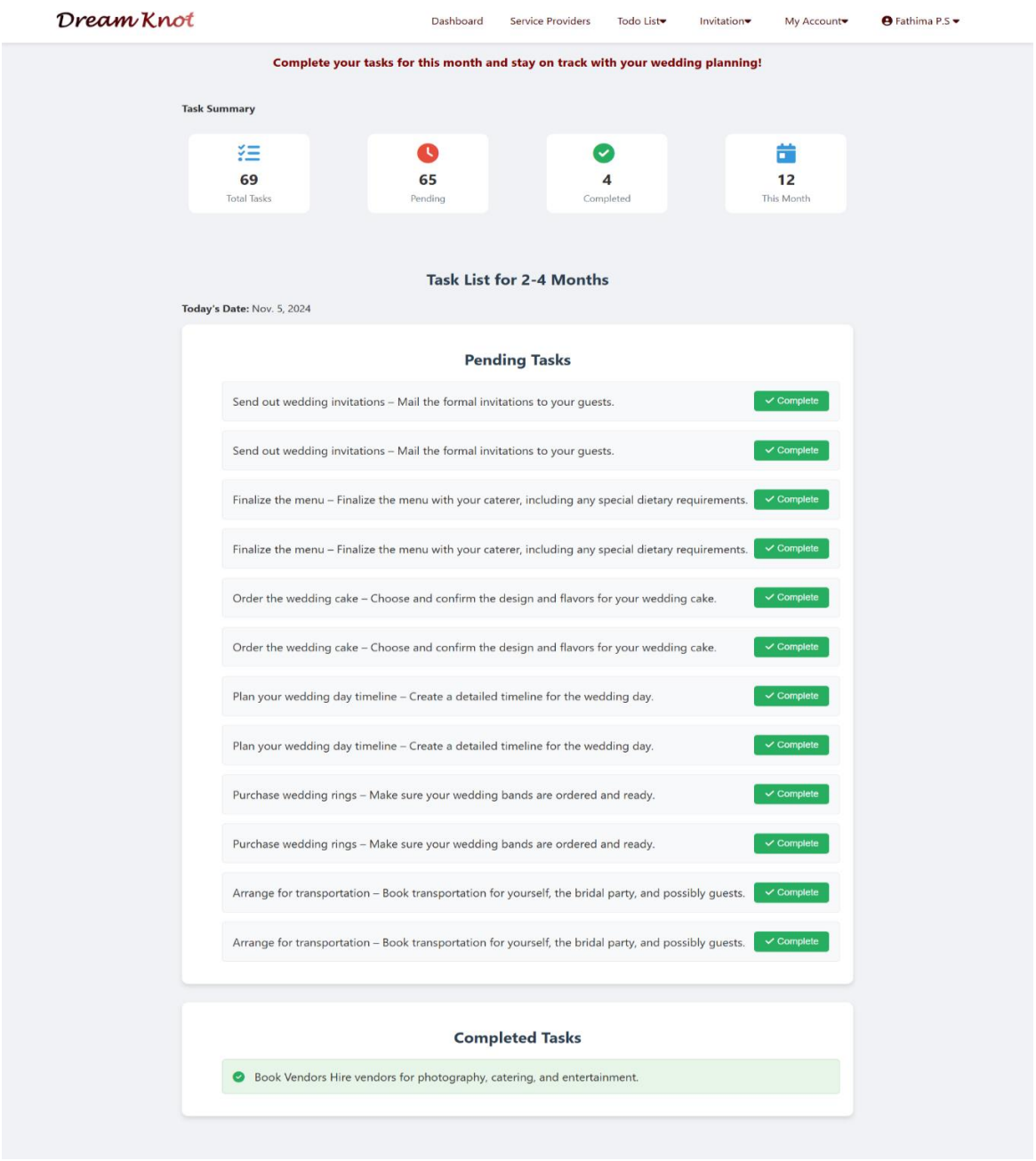
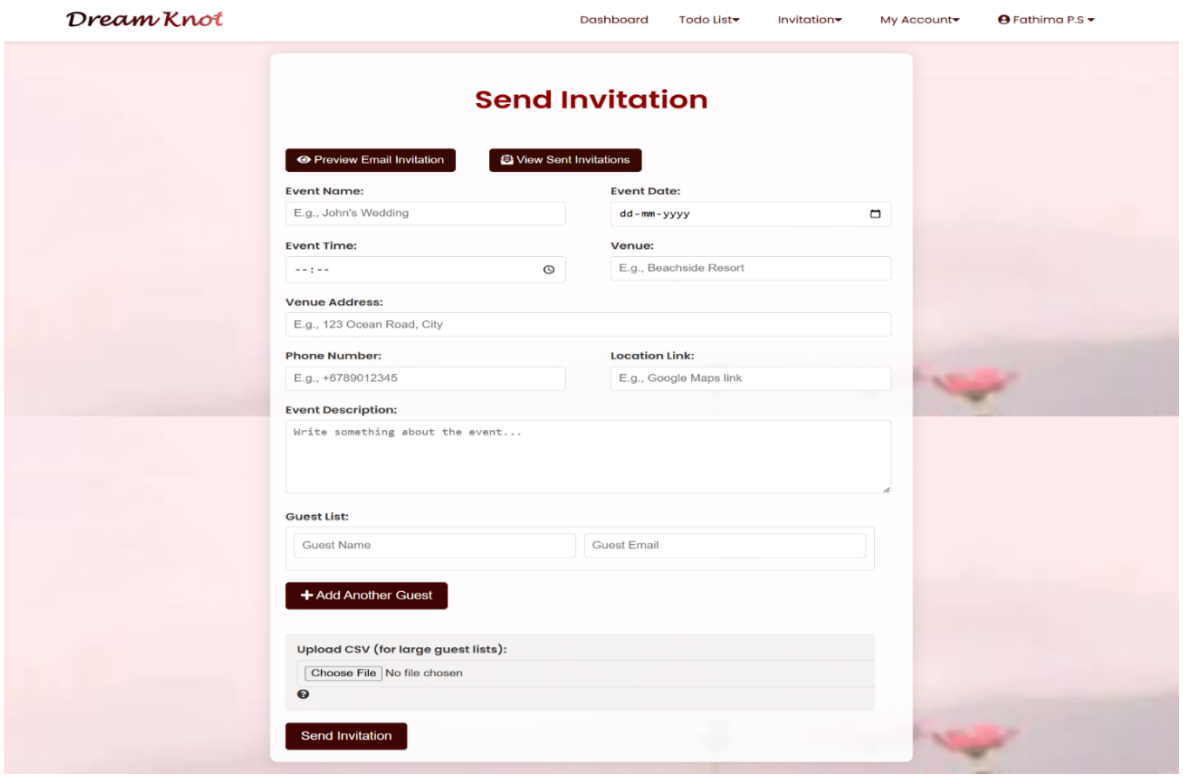


Figure 9.2.11 Current Mont task list Page

9.2.12 E-Invitation Page



9.2.12 E-Invitation Page

9.2.13 Favorite list Page

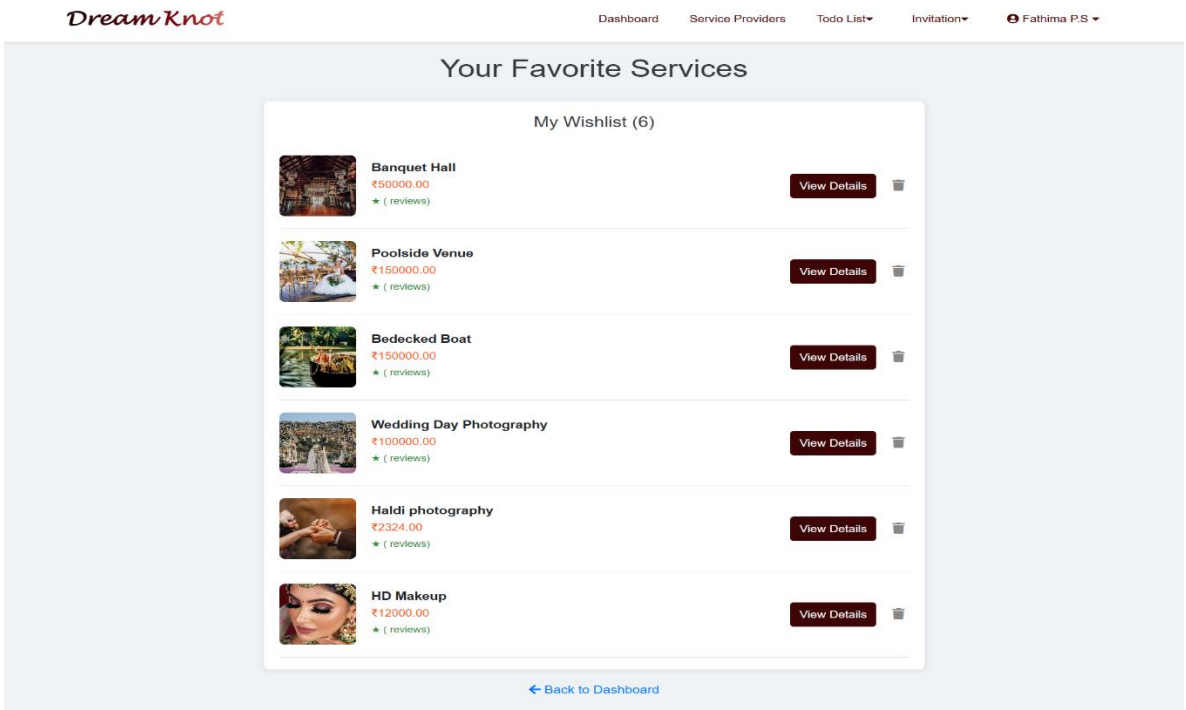


Figure 9.2.13 Favorite list Page