**Requirements Analysis Design**

**Hire A Chef**

|  |  |  |
| --- | --- | --- |
| **Industry Partner** |  | NA |
| **Primary Instructor** |  | Anjana Shah |
| **Team Member** |  | Gokul Jinu |
| **Team Member** |  | Ans Joju |
| **Team Member** |  | Basil Shaji |
| **Team Member** |  | Nithiyan Annapoorani Valluvan |
| **Team Member** |  |  |

Document Revision History

|  |  |
| --- | --- |
| Revision # | Date |
| 1 | 2024-11-08 |
|  |  |

## **1.0 Introduction**

This document outlines the software requirements and design for the Hire-A-Chef platform. It serves as a roadmap for the development process by identifying the required features, constraints, assumptions, and design frameworks. The purpose of this analysis is to ensure the platform meets business needs and provides an efficient, user-friendly service for hiring private chefs.

## **1.1 Purpose**

This document defines the software requirements from a high-level perspective, focusing on what the system will do to meet the needs of its users. It is intended for stakeholders including developers, clients, chefs, and project managers. This analysis ensures that the system aligns with business objectives and user expectations without going into detailed technical implementation.

## **1.2 Scope**

The Hire-A-Chef platform is designed to connect clients with private chefs for various events. It will allow users to search for chefs, make bookings, manage payments, and provide feedback. Core functionalities include:

* User registration for chefs, clients, and admins.
* Chef profiles with availability and menu offerings.
* Booking management for clients and chefs.
* In-app messaging between chefs and clients.
* Secure payments via integrated payment gateway.

**What the platform will not do**:

* It will not provide in-house delivery or catering services.
* It will not handle event planning logistics outside chef booking.

## **2.0 System Overview**

## **2.1 Project Perspective**

The Hire-A-Chef platform is a new self-contained system designed to streamline the booking of private chefs. It addresses the fragmented nature of the private chef industry by offering a centralized, user-friendly solution.

## **2.2 System Context**

The platform supports three key user roles: Clients, Chefs, and Admins. Clients search for chefs based on location, budget, and preferences. Chefs manage their profiles and accept or decline bookings, while Admins oversee the platform’s operations. Third-party services like payment gateways and SMS notifications ensure smooth operation and communication.

## **2.3 General Constraints**

* **Time Constraint:** MVP launch planned by March 2025.
* **Budget Constraint:** Limited to free or low-cost services for hosting and communication.
* **Technology Stack Constraint:** Development will rely on JavaScript, Node.js, and MongoDB.

## **2.4 Assumptions and Dependencies**

* **Assumptions**:
  + Chefs will update their availability regularly.
  + Users will provide accurate information during registration and booking.
* **Dependencies**:
  + Integration with third-party payment providers (e.g., Stripe or PayPal).
  + Reliance on communication services (e.g., Twilio for SMS).
  + Availability of cloud hosting services (AWS or Azure).

## **3.0 Functional Requirements**

## This section describes the specific functionalities required for the "Hire a Chef" platform, allowing users (clients, chefs, admins) to interact with the system effectively.

## **3.1 Feature #1: Chef Search and Selection**

## **Introduction:** This feature allows clients to search for chefs based on filters such as location, budget, dietary preferences, and cuisine type. This feature helps clients find suitable chefs for their events efficiently.

## **Inputs:**

## Location, budget, and dietary preferences entered by the client.

## Optional filters for cuisine type, event type, and chef ratings.

## **Processing:**

## The system retrieves chef profiles that match the client's search criteria.

## Filters are applied to sort chefs based on ratings, availability, and relevance to the client’s preferences.

## **Outputs:**

## A list of chefs that match the client's criteria.

## Detailed profiles for each chef, including experience, specialties, ratings, and availability.

## **3.2 Feature #2: Chef Booking and Confirmation**

## **Introduction**: Enables clients to book a chef for a specific date and time, with the chef confirming the booking. This feature helps streamline the booking process for both clients and chefs.

## **Inputs**:

## Selected chef profile, event date, time, and additional booking details provided by the client.

## **Processing**:

## System verifies the chef's availability for the selected date.

## A booking request is generated and sent to the chef for confirmation.

## Chef either accepts or declines the booking request.

## **Outputs**:

## Booking confirmation notification for the client if the chef accepts.

## Updated booking status in the system.

## **3.3 Feature #3: Payment Processing**

## **Introduction**: Allows clients to pay for chef services securely through integrated payment gateways. Ensures payment is processed before the booking is confirmed.

## **Inputs**:

## Booking details, total amount, and client’s payment information.

## **Processing**:

## System calculates the total cost based on the booking details (e.g., chef rate, additional charges).

## Payment details are sent to the payment processor for authorization.

## Once authorized, payment is confirmed and recorded in the system.

## **Outputs**:

## Confirmation of successful payment.

## Receipt of payment sent to the client and chef.

## Updated booking status to "Confirmed" in the system.

## **3.4 Feature #4: Chef Availability Management**

## **Introduction:** Allows chefs to manage and update their availability, ensuring that clients can book them only when they are available.

## **Inputs:**

## Dates and times marked as available or unavailable by the chef.

## **Processing:**

## The system updates the chef's availability in real-time.

## Any conflicts with existing bookings are flagged, and the chef is notified.

## **Outputs:**

## Updated availability status for the chef.

## Clients see only the updated availability when searching for chefs.

## **3.5 Feature #5: User Registration and Authentication**

## **Introduction**: Allows new users (clients, chefs, admins) to register and existing users to log in securely.

## **Inputs**:

## Registration information (name, email, password, role) or login credentials (email, password).

## **Processing**:

## System validates inputs and verifies user identity.

## For new users, a verification process (e.g., email confirmation) is conducted.

## **Outputs**:

## Successful account creation or login confirmation.

## Access to the platform based on the user's role (e.g., client, chef, admin).

## **3.2 Use Cases**

## **3.2.1 Use Case #1: Book a Chef**

## **Description**: Allows a client to search for available chefs, view their profiles, and book a chef for a specific date and event.

## **Actors**: Client, Chef, Hire-A-Chef System, Payment Processor

## **Preconditions**:

## The client is logged into their account.

## The chef's availability has been updated in the system.

## **Main Success Scenario**:

## Client searches for available chefs based on date, location, budget, and dietary preferences.

## System displays a list of chefs matching the criteria.

## Client selects a chef and views the chef’s profile, including experience, specialties, and ratings.

## Client selects a date and time and submits a booking request.

## System processes the booking and sends a notification to the chef.

## Chef accepts or declines the booking request.

## If accepted, the system confirms the booking and prompts the client for payment.

## Client submits payment, which is processed by the Payment Processor.

## System updates the booking status to "Confirmed" and notifies both the client and the chef.

## **Postconditions**:

## A confirmed booking is recorded in the system.

## Payment is processed and recorded.

## Chef’s availability is updated accordingly.

## **Exceptions**:

## Chef declines the booking: System notifies the client and allows them to choose another chef.

## Payment failure: System notifies the client to retry or use a different payment method.

## 3.2.2 Use Case #2: Manage Chef Availability

## **Description**: Allows chefs to manage and update their availability on the platform.

## **Actors**: Chef, Hire-A-Chef System, Admin

## **Preconditions**:

## The chef is logged into their account.

## The chef has an active profile on the platform.

## **Main Success Scenario**:

## Chef logs into their account and navigates to the "Manage Availability" section.

## Chef selects specific dates and times they are available or unavailable.

## Chef submits the updated availability.

## System updates the chef’s availability data and makes it accessible for clients to view during searches.

## If any conflicts arise (e.g., previously confirmed bookings on the updated dates), the system notifies the chef and suggests alternatives or prevents the update.

## **Postconditions**:

## Chef’s availability is updated in the system.

## Clients searching for chefs see only available options based on the updated information.

## **Exceptions**:

## Chef tries to mark unavailable dates with existing bookings: System prompts the chef to contact the admin for conflict resolution.

## 3.3 Data Modelling and Analysis

* **Normalized Data Model Diagram**:

A computer screen shot of a computer

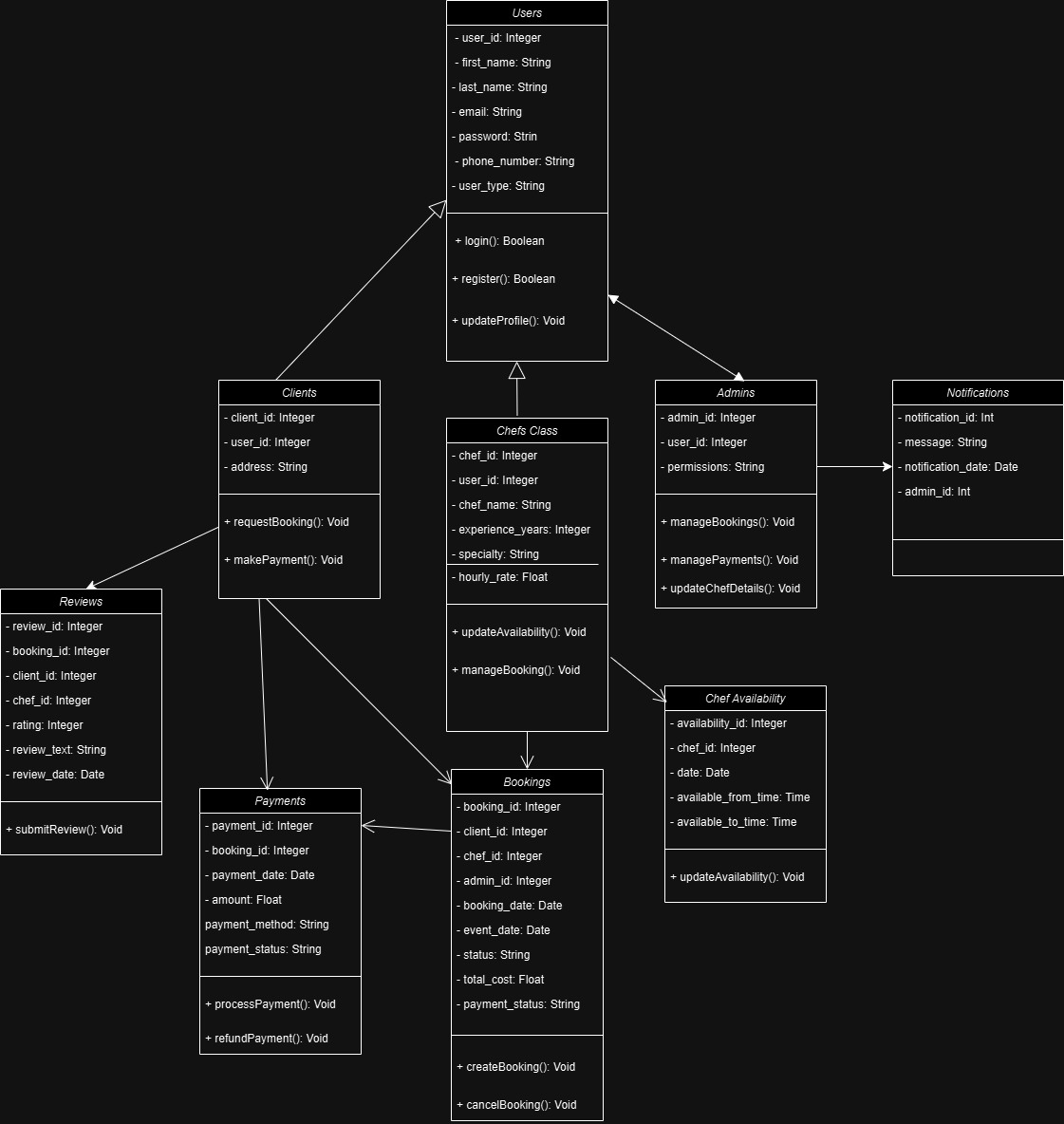
Description automatically generated

**Activity Diagram**: A diagram of a company structure

Description automatically generated

* A diagram of a restaurant

  Description automatically generated**Sequence Diagram**:
* **UML Class Diagram**:



## 3.4 Process Modelling

* **Data Flow Diagram (DFD)**:

A diagram of a software system

Description automatically generated.

## 4.0 Non-Functional Requirements

This section describes the non-functional requirements of the "Hire a Chef" platform. These requirements focus on how the system should operate, ensuring that it provides a smooth, secure, and reliable experience for users.

4.1 Performance

* **Requirement**: The system should be able to handle at least 1,000 concurrent users without significant degradation in performance.
* **Response Time**: 95% of user interactions, such as chef searches and profile views, should load in less than 2 seconds.
* **Transaction Processing**: Payment transactions should be processed within 3 seconds to ensure a seamless booking experience.
* **Scalability**: The system must be scalable to accommodate increasing user loads, with the ability to add more server resources as needed.

4.2 Reliability

* **Requirement**: The platform should maintain consistent functionality without failures during normal operations.
* **Error Tolerance**: System errors or crashes should not exceed 0.01% of all user interactions.
* **Data Integrity**: User data, including booking details and payment information, should remain consistent and accurate, even during peak usage or unexpected system issues.
* **Error Recovery**: In the event of a system failure, the platform should automatically recover within 5 seconds, with minimal disruption to the user experience.

4.3 Availability

* **Requirement**: The system should maintain an uptime of 99.9%, translating to less than 8.77 hours of downtime per year.
* **System Downtime**: Planned maintenance or updates should be scheduled outside peak hours and should not exceed 15 minutes at a time.
* **Redundancy**: Implement server redundancy to ensure that if one server fails, another server can take over without affecting availability.

4.4 Security

* **Requirement**: The system must ensure that all user data, especially sensitive information like payment details, is securely protected.
* **Data Encryption**: All data in transit (e.g., during payment processing) should be encrypted using SSL/TLS.
* **User Authentication**: Multi-factor authentication (MFA) should be implemented for chefs and admins to enhance account security.
* **Access Control**: Role-based access control (RBAC) should be applied to restrict access to sensitive features and data based on user roles (client, chef, admin).
* **Vulnerability Management**: Regular vulnerability scans and security audits should be conducted quarterly to identify and mitigate any potential security risks.

4.5 Maintainability

* **Requirement**: The system should be easily maintainable and adaptable to future changes or feature additions.
* **Code Modularity**: The codebase should be modular and well-documented, allowing for easy updates or replacements of specific components.
* **Bug Fix Time**: Critical bugs should be resolved within 24 hours of identification, while minor issues should be addressed within a week.
* **Logging and Monitoring**: Real-time logging and monitoring systems should be in place to help identify performance issues and errors for quick resolution.

4.6 Portability

* **Requirement**: The system should be deployable across multiple environments, such as cloud servers and on-premises data centers, to support flexibility in deployment.
* **Cross-Platform Compatibility**: The user interface should be compatible with major web browsers (e.g., Chrome, Firefox, Safari) and mobile devices.
* **Database Portability**: The platform’s database should be portable, enabling migration to other database management systems if needed without extensive code changes.

## 5.0 Logical Database Requirements

The "Hire a Chef" platform will use a MongoDB database to manage core data entities like user accounts, chef profiles, bookings, payments, and reviews. Below are the key requirements to ensure effective database management and integrity:

5.1 Database Overview

* **Database:** MongoDB, a NoSQL database, will handle flexible and scalable data storage for user accounts, chef profiles, booking records, payments, and client reviews.

5.2 Data Structure and Format

* **Data Collections:**
  + **Users:** Stores client and chef profiles, including names, contact details, addresses, and preferences.
  + **Bookings:** Records booking details (client ID, chef ID, date, time, event details, booking status).
  + **Payments:** Tracks transaction information (booking ID, client ID, payment amount, date, status).
  + **Reviews:** Stores client ratings and feedback on chefs, linked by chef ID.

5.3 Data Retention

* **Transaction Data:** Transaction records, including payments and booking details, will be retained for a minimum of 12 months. This allows time for review and follow-up inquiries while complying with storage limits.
* **Inactive Accounts:** User accounts that remain inactive for over 18 months will be flagged for review, with data archived if necessary.

5.4 Data Integrity

* **Unique Constraints:**
  + Enforce unique constraints on email addresses to prevent duplicate user accounts.
  + Apply unique booking IDs to ensure each booking is tracked distinctly.
* **Data Validation:** Validation rules will ensure essential fields (e.g., email, payment status) are complete and accurate.
* **Relationship Integrity:** Relationships between users, bookings, and payments will be maintained through MongoDB’s reference fields, linking relevant data across collections.

5.5 Backup Policy

* **Automated Weekly Backups:** Weekly backups will be automatically created and stored in cloud storage (e.g., AWS S3) to ensure data is protected against loss.
* **Backup Retention:** Backups will be retained for three months, after which they will be rotated to maintain storage efficiency.
* **Recovery Testing:** Backup recovery tests will be conducted quarterly to ensure data can be restored in case of system issues.

## 6.0 Other Requirements

These additional requirements address compliance, accessibility, development practices, and recovery needs.

6.1 Compliance and Security

* **Data Privacy:** The platform will implement data encryption at rest and in transit for sensitive information, especially for user credentials and payment details.
* **Authentication:** Secure user authentication with password hashing will be implemented. Optional two-factor authentication (2FA) will be available for enhanced security, particularly for chefs and admins.
* **Access Control:** Role-based access control will restrict admin functions to authorized users, limiting permissions based on user role (e.g., client, chef, admin).

6.2 User Interface Accessibility

* **Responsive Design:** The interface will be optimized for accessibility on both desktop and mobile devices, supporting a broad user demographic.
* **Language Support:** A basic multilingual support feature will allow users to view content in two to three languages as needed by the target audience.

6.3 Environment Requirements

* **Development and Version Control:** GitHub will be used for source code version control, with a simple CI/CD pipeline configured to run automated tests and deploy updates efficiently.
* **Hosting:** The platform will be deployed on a cloud service provider (e.g., AWS or Azure) for scalability and managed infrastructure.

6.4 Backup and Recovery

* **Backup Frequency:** Weekly backups, with automated processes for cloud storage, will ensure data is consistently protected.
* **Disaster Recovery Plan:** A recovery plan will outline procedures for data restoration, backup testing, and service continuity to minimize downtime.

## 7.0 Approval

| **Project Role** | **Name** | **Signature** | **Date** |
| --- | --- | --- | --- |
| Team Member | Nithiyan Annapoorani Valluvan | Nithiyan Annapoorani Valluvan | \_\_\_2024/11/08\_\_\_\_\_ |
| Team Member | Ans Joju | \_\_Ans Joju\_\_\_\_\_\_\_\_\_ | \_\_\_2024/11/08\_\_\_ |
| Team Member | Basil Shaji | \_\_Basil Shaji \_\_\_\_\_\_\_\_\_ | \_\_\_2024/11/08\_\_\_\_\_ |
| Team Member | Gokul Jinu | \_\_\_Gokul Jinu \_\_\_\_\_\_\_\_ | \_\_\_2024/11/08\_\_\_ |