Hamdard University

Department of Computing

Final Year Project

Digital Chef Sync

Software Design Specifications

Submitted by

Anas Sabih Sarwar (2600-2021)

Abu Bakar (2167-2021)

Hamza Rafique (2195-2021)

Supervisor

Sir Iqbal Uddin Khan

Spring 2020

Document Sign off Sheet

1.1.1 Document Information

Project Title	Digital Chef Sync
Project Code	FYP-001/SP25
Document Name	Software Design Specifications
Document Version	1.0
Document Identifier	FYP-001/SP25-SDS
Document Status	Final
Author(s)	Anas Sabih Sarwar, Abu Bakar, Hamza Rafique
Approver(s)	Sir Iqbal Uddin Khan
Issue Date	01-July-2025

Name	Role	Signature	Date
Anas Sabih Sarwar	Team Lead		
Abu Bakar Balouch	Team Member 2		
Hamza Rafique	Team Member 3		
Sir Iqbal Uddin Khan	Supervisor		
Sir Prem Sagar	Project Coordinator		

Revision History

Date	Version	Description	Author
19-June-2025	0.1	First Draft	Anas Sabih , Hamza
26-June-2025	0.2	Revised Version	Sir Iqbal Uddin Khan
01-July-2025	1.0	Final Report	Anas Sabih

Definition of Terms, Acronyms, and Abbreviations

TERM	DESCRIPTION
Scrum	An agile project management methodology emphasizing iterative development and continuous feedback.
UI/UX	User Interface/User Experience design, focused on usability, simplicity, and accessibility for both job seekers and recruiters.
MongoDB	A flexible NoSQL database used for scalable and schema-less data storage in real-time applications.
React Native	A JavaScript framework used to build cross-platform mobile applications for Android with a native feel.
Frontend	The mobile app interface developed using React Native, allowing users to interact with job features, wallet, and notifications.
Backend	Server-side logic and APIs developed using Node.js and Express.js, handling authentication, job data, payments, and more.

2 Table of Contents

3 Introduction

The **Digital Chef Sync** project is designed to modernize how the food and hospitality industry handles workforce recruitment and shift management. It addresses the ongoing challenges faced by restaurants, cafes, and event staffing teams in sourcing skilled workers efficiently and managing staff assignments in real time. Instead of relying on manual hiring methods or fragmented communication channels, this platform introduces an integrated and automated solution that simplifies the hiring process from start to finish.

The core pillars of the system are **reliability**, **security**, **scalability**, and **ease of use**, ensuring that both recruiters (such as restaurant owners and event managers) and job seekers (including chefs, servers, and kitchen staff) experience a smooth and transparent process. The development process follows the **Scrum methodology**, ensuring agile, iterative progress with continuous stakeholder involvement and feedback at every stage.

3.1 Purpose of Document

The purpose of this **Software Design Specification (SDS)** document is to present a detailed technical blueprint of the system's architecture, modules, and functionalities. It describes how the system components will be developed, integrated, and maintained to meet project goals.

This document serves as a communication bridge between all stakeholders, helping ensure that the development team, supervisors, and end-users have a shared understanding of how the platform is being built. It also provides a foundation for future upgrades by maintaining clear documentation aligned with **object-oriented design** principles and best practices for **modular and scalable development**.

3.2 Intended Audience

This document is specifically intended for the following audience:

- **Project Supervisors** – To track the system's design decisions and verify if project development aligns with the original goals and deadlines.

- **Developers & UI/UX Designers** – To follow a consistent design pattern, coding guidelines, and system architecture.
- **Recruiters / Business Users** – To understand how their hiring and management needs are being addressed through the system.
- **Job Seekers (Chefs, Staff)** – While not directly using this document, their requirements influence the platform's user-facing design and functionality.

3.3 Document Convention

Font Style: Calibri

Font Size: 12pt for main content, 14pt or higher for headings

Terminology:

- "Seeker" refers to the job applicant.
- "Recruiter" refers to the hiring entity or business.
- "Platform" or "System" refers to the Digital Chef Sync web application.

3.4 Project Overview

Digital Chef Sync is a specialized workforce management platform developed to streamline hiring and shift scheduling for restaurants, cafes, and event-based businesses. The system is designed to automate the end-to-end process of connecting recruiters with skilled hospitality professionals while ensuring smooth communication, secure transactions, and real-time updates.

Key functionalities include:

- **Job Posting and Application Tracking:** Recruiters can post available positions, and seekers can apply and monitor their application status through a user-friendly dashboard.
- **Automated Payment Verification:** Secure wallet-based system ensures that shift payments and job bookings are accurately processed and verified without delays.
- **Smart Notifications and Status Updates:** Real-time alerts keep both parties informed about job confirmations, cancellations, and shift start times.
- **Shift Scheduling and Staff Matching:** Based on skill tags, location, and availability, the system assists recruiters in selecting the most suitable candidates efficiently.

The platform is built using **ReactJS for the frontend**, **Node.js and Express.js for backend services**, and **MongoDB for scalable data management**. Agile practices through **Scrum methodology** ensure that feedback is regularly implemented and development stays aligned with evolving business needs.

3.5 Scope

The scope of **Digital Chef Sync** is focused on transforming the traditional restaurant staffing model by introducing automation, accountability, and flexibility.

The system will:

- **Automate Job Matching:** Ensure that recruiters are matched with the most relevant candidates based on job role, location, and availability.
- **In-App Wallet Integration:** Enable secure payment processing, job booking, and fund transfers through the system's wallet.
- **Real-Time Status Tracking:** Allow seekers to view job status, notifications, and application outcomes instantly.
- **Shift & Role Management:** Let recruiters manage shifts, assign roles, and monitor completion—all from one centralized platform.

Out of Scope:

- **Employee performance evaluation or disciplinary tracking** beyond shift attendance and status.
- **Integration with third-party payroll systems** or government regulatory platforms for tax/legal compliance.
- **Support for non-hospitality industries** outside of food service and event staffing.

4 Design Considerations

This section highlights the key assumptions, dependencies, and potential risks that may impact the overall system design for **Digital Chef Sync**. These considerations help ensure that the platform remains flexible, scalable, and ready to accommodate future updates while maintaining operational stability.

4.1 Assumptions and Dependencies

The design and development of Digital Chef Sync are based on the following assumptions and system dependencies:

- **Reliable Internet Connectivity:**
  It is assumed that both recruiters and job seekers will have consistent access to the internet, as the platform is App based and requires online interactions for all major functionalities.
- **Technology Stack Availability and Compatibility:**
  The system relies on proven tools such as **ReactJS** for the frontend, **Node.js and Express.js** for backend services, and **MongoDB** for handling data operations. These technologies are expected to function smoothly together without critical compatibility issues.
- **Basic Digital Literacy:**
  It is assumed that users (both seekers and recruiters) have a basic understanding of using mobile application. Minimal training may be provided for onboarding users unfamiliar with digital tools.
- **Stakeholder Cooperation:**
  Timely collaboration from all stakeholders—particularly restaurant managers and hiring staff—is essential for successful requirement gathering, testing, and deployment phases.
- **Compliance with Privacy Standards:**
  The platform is expected to comply with applicable data protection regulations, especially when handling user identity, transaction records, and communication data.

4.2 Risks and Volatile Areas

While the project is structured to be stable, several unpredictable factors may affect the system's performance or development flow. Anticipating these risks allows for better design decisions and effective mitigation planning.

1. Evolving Feature Requests

- **Risk:** Recruiters or management teams might request additional features post-deployment (e.g., analytics dashboards).
- **Mitigation:** Design the platform with **modular architecture** so that new components can be added without affecting the core system.

2. Rapid Technological Advancements

- **Risk:** Tools or libraries currently in use may become outdated or unsupported in the near future.
- **Mitigation:** Use **widely adopted open-source technologies** with active community support and maintain **flexible coding standards** for easier upgrades.

3. High User Load or Scalability Issues

- **Risk:** Sudden spikes in user traffic (e.g., during job fairs or seasonal events) may affect system responsiveness.
- **Mitigation:** Optimize backend logic and implement **database indexing and load balancing** techniques to manage concurrent requests efficiently.

4. Data Security and User Privacy

- **Risk:** As the platform deals with sensitive job profiles and payment information, there's a risk of data breaches or misuse.
- **Mitigation:** Apply **encryption protocols (SSL)**, enforce **strong password policies**, and include **role-based access control (RBAC)** for added security.

5. External System Dependencies

- **Risk:** Integration with third-party payment services or email APIs may be interrupted due to outages or changes.
- **Mitigation:** Include **manual override workflows** or fail-safes to handle temporary external service disruptions.

6. Team Collaboration and Development Delays

- **Risk:** Inadequate communication or task mismanagement within the development team could delay milestones.
- **Mitigation:** Follow the **Scrum framework** with daily stand-ups, clear task distribution, and sprint planning sessions to keep progress on track.

7. User Resistance to Change

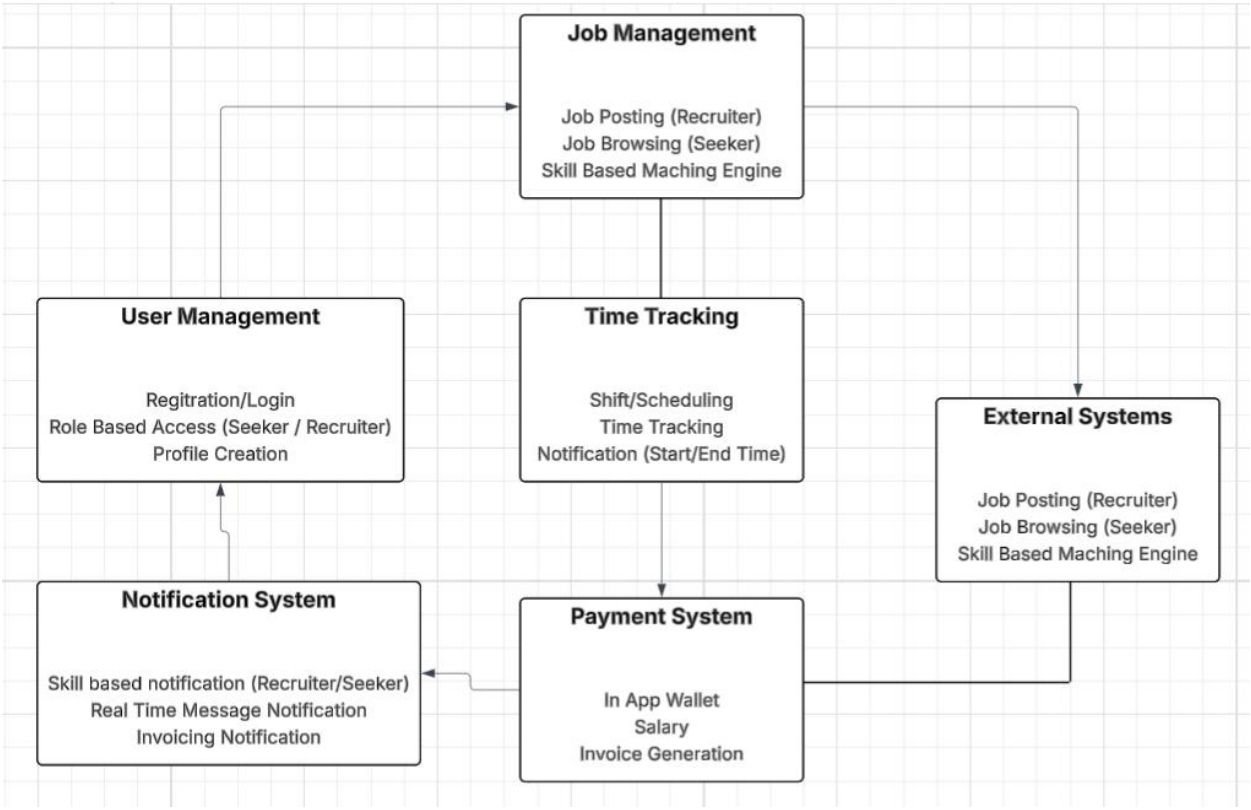
- **Risk:** Restaurant managers and workers may be hesitant to switch from traditional or manual hiring methods.
- **Mitigation:** Provide **easy-to-follow guides**, onboarding videos, and **in-app assistance** to improve user confidence and acceptance.

5 System Architecture

The architecture of ***Digital Chef Sync*** is organized into clearly defined subsystems that support modular development, efficient staffing operations, and seamless user interaction between job seekers and recruiters. This Android-only platform is purpose-built for restaurants, cafes, and event management businesses to streamline hiring and workforce management without relying on external communication or payment platforms.

5.1 System Level Architecture

- **User Management Subsystem**
  Handles user registration, login, profile management, and role identification (seeker or recruiter). This module controls access to functionalities based on the user type, ensuring personalized navigation throughout the app.
- **Job Application & Matching Subsystem**
  Enables recruiters to post job listings and seekers to view and apply based on skills, availability, and location. The system uses structured matching logic to pair the right candidate with the right job, ensuring efficient hiring.
- **In-App Notification Subsystem**
  Delivers real-time alerts to both seekers and recruiters. Seekers are notified about new jobs, application status, or gig confirmations. Recruiters receive alerts for incoming applications and responses. No email or SMS integrations are used.
- **Shift & Time Tracking Subsystem**
  Facilitates shift scheduling by recruiters and enables seekers to check-in/out for tracking work hours. Shift data is logged for both parties and linked to the payment system.
- **In-App Wallet & Payment Subsystem**
  Manages payments within the application. After shifts are completed and verified, recruiters can release funds into the seeker’s in-app wallet. Users can later request withdrawals manually. No external payment APIs or gateways are integrated.



5.2 Software Architecture

The software architecture of **Digital Chef Sync** is built using a layered design to ensure maintainability, scalability, and clear separation of responsibilities across the system. This modular architecture helps streamline communication between components and simplifies future enhancements.

Architectural Layers

1. Presentation Layer (Android User Interface)

- This layer is responsible for delivering the user experience for both **job seekers** and **recruiters**.
- Designed using **Android SDK** and **XML layouts**, it handles user input, screen navigation, and real-time updates via **in-app notifications**.
- It communicates with the backend services through **secure API calls** and reflects changes such as new job postings, shift assignments, or wallet updates.

2. Application Logic Layer

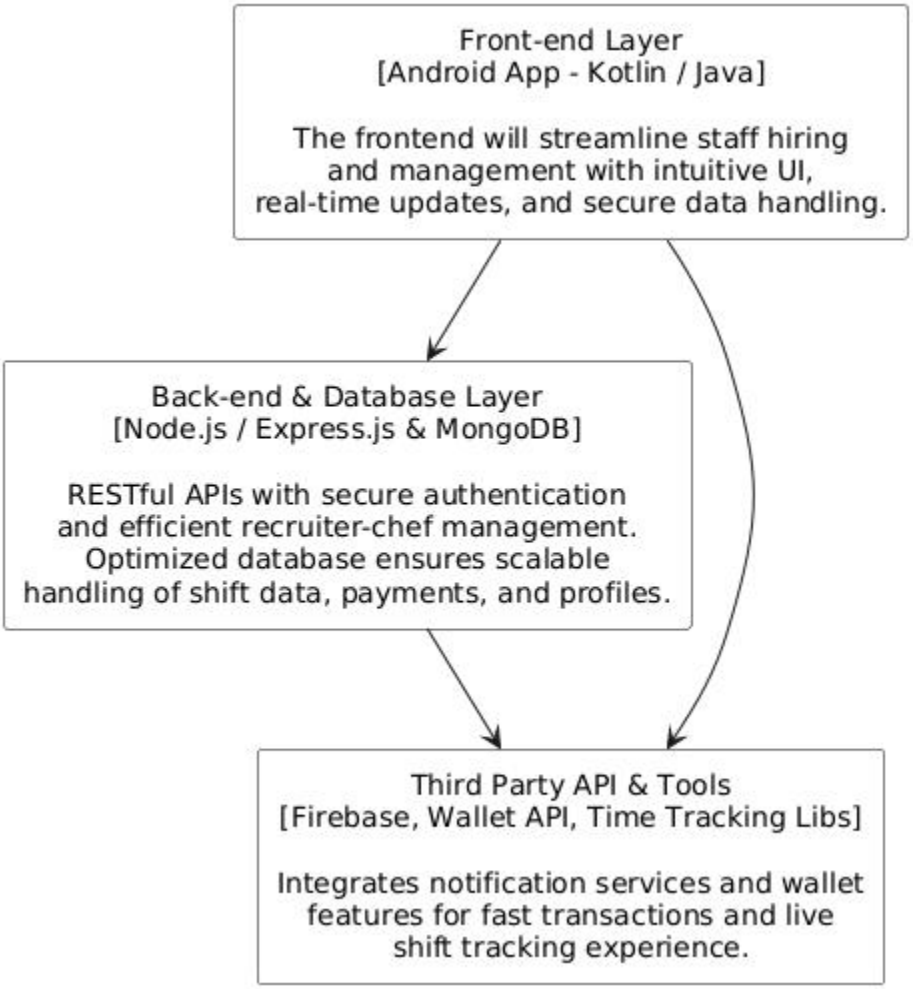
- This middle layer handles the **core functionality** of the app — including job matching, shift tracking, wallet management, and access control.
- Built using **Node.js** and **Express.js**, it defines how the system processes data and enforces business rules (e.g., only verified seekers can apply for jobs or track shifts).
- This layer validates requests, processes transactions, and routes them to the appropriate database or service endpoints.

3. Data Access Layer

- This layer manages all interactions with the **backend database**, including storing and retrieving user profiles, job posts, shift logs, and wallet transactions.
- Uses **MongoDB** for its schema flexibility and ability to handle dynamic, document-based data.
- Ensures that queries are secure, efficient, and optimized to support real-time app performance, especially during high traffic such as during mass hiring events.

Interactions Between Layers

- The **Presentation Layer** sends requests and receives data from the **Application Logic Layer** using **RESTful APIs**.
- The **Application Logic Layer** performs input validation, enforces business rules, and then forwards database operations to the **Data Access Layer**.
- All in-app actions — such as applying for a job, scheduling a shift, or requesting wallet withdrawal — follow this structured flow, maintaining a clean and efficient architecture.


Front-end Layer
[Android App - Kotlin / Java]
The frontend will streamline staff hiring and management with intuitive UI, real-time updates, and secure data handling.
Back-end & Database Layer
[Node.js / Express.js & MongoDB]
RESTful APIs with secure authentication and efficient recruiter-chef management. Optimized database ensures scalable handling of shift data, payments, and profiles.
Third Party API & Tools
[Firebase, Wallet API, Time Tracking Libs]
Integrates notification services and wallet features for fast transactions and live shift tracking experience.

6 Design Strategy

The design approach for Digital Chef Sync is centered around building a reliable, modular, and scalable Android application that meets the fast-paced demands of the food service and event staffing industry. The system is built to ensure performance, flexibility, and user-friendliness for both recruiters and job seekers. It is structured to support current staffing operations while allowing room for future enhancements.

6.1 Future System Extension or Enhancement

- **Modular Structure:**
  The system is broken down into logical modules such as **User Authentication**, **Job Posting**, **Shift Assignment**, **In-App Wallet**, and **Profile Management**. Each module is designed independently so new features (e.g., rating system or chat support) can be added later without overhauling the existing app.
- **Scalability:**
  Although currently designed for a limited region or set of businesses, the system architecture allows scale-up in terms of both **user base** and **feature expansion**, supporting more restaurants, cities, or service categories.
- **Integration Ready:**
  The app is built in a way that can later be integrated with third-party services, such as **external payment providers**, **HR platforms**, or **identity verification tools**, using REST APIs without major structural changes.

6.2 System Reuse

Reusable Logic Components:
Business rules such as role validation, wallet transaction logic, and job filtering are written as reusable functions for easy maintenance and use in other modules.

Shared UI Elements:
Components like buttons, input fields, card layouts, and modals are designed as reusable widgets, maintaining UI consistency and reducing redundant code.

Standard Data Models:
Collections in the MongoDB database follow a well-documented schema (e.g., users, shifts, jobs, walletLogs) to promote reuse in any similar staffing or gig-matching apps in the future.

6.3 User Interface Paradigms

- **Mobile-First Interface:**
  As this is an Android-only app, the entire UI is designed natively for mobile interaction using **Jetpack libraries** and **Material Design principles**, ensuring a smooth and responsive user experience.
- **User Role-Based Layouts:**
  Job Seekers and Recruiters are presented with distinct dashboards. Seekers can view and apply for jobs, track shifts, and manage wallets, while Recruiters can post vacancies, monitor applicants, and confirm shift attendance.
- **Simple and Interactive Navigation:**
  Bottom navigation bars, filters, and search components are used to keep the app accessible even for users with minimal technical knowledge.

6.4 Data Management

Centralized NoSQL Storage:
MongoDB is used to store user profiles, job posts, applications, and transaction logs. This provides flexibility in handling dynamic and nested data like multiple shift preferences or payment histories.

Security & Encryption:
User data and wallet records are protected using token-based authentication and encrypted communication between the frontend and backend (e.g., HTTPS & hashed tokens).

Auto Backup and Recovery:
The database is configured for regular backups and supports easy recovery to avoid any data loss during downtime or app crashes.

Scalable Collections:
The database schema supports dynamic data growth — e.g., more job categories or feedback logs — without altering existing data structures.

6.5 Concurrency and Synchronization

Asynchronous API Handling:
The backend handles API calls using asynchronous logic via **Node.js**, allowing the app to perform background tasks such as wallet updates or job application status changes without delays.

Concurrent Job Applications:
The system supports multiple seekers applying for the same job post in real-time, with conflict management in place to prevent overbooking.

Optimistic Data Updates:
To avoid data collisions when multiple users perform actions simultaneously, updates are handled using optimistic concurrency where newer entries replace outdated data after confirmation.

6.6 Reasoning and Trade-Offs

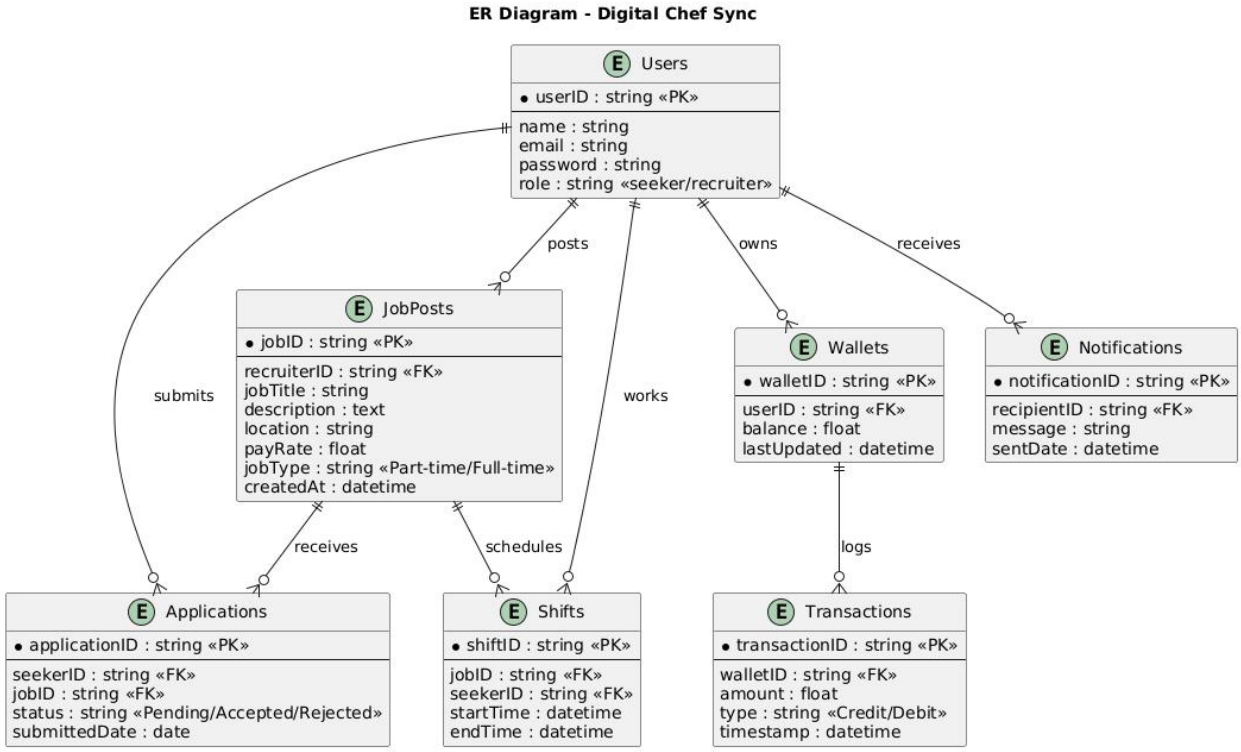
Strategic Design Decisions:
The choice of a modular, role-based architecture aligns with the project's core goals of flexibility and rapid deployment. This separation ensures that the codebase remains clean and each module is easily testable and upgradeable.

Trade-Offs:

- Using **Node.js and MongoDB** speeds up development but requires developers to be familiar with JavaScript across the stack.
- **Android-only focus** limits user reach but allows deeper optimization and faster updates for the primary user base.
- In-app notifications are used instead of SMS/email alerts to reduce operational costs and simplify integration, although it may reduce reach for users without regular app access.

6.1 Database Design

6.1.1 ER Diagram



6.1.2 Data Dictionary

6.1.2.1 Data 1

User						
Name	Users					
Alias	Seeker / Recruiter					
Where-used/how-used	Used for login, job posting (by recruiter), job application (by seeker), notifications, wallet management, and shift allocation.					
Content description	Stores all registered users on the platform, including their credentials and user type (seeker or recruiter).					
Column Name	**Description**	**Type**	**Length**	**Null able**	**Default Value**	**Key Type**
userID	Unique identifier for the user	*string*	*36*	*No*	*None*	*PK*
name	*Full name of the user*	*string*	*255*	*No*	*None*	
email	*User's email address*	*string*	*255*	*No*	*None*	
password	*Encrypted password*	*string*	*255*	*No*	*None*	
role	*Type of user (seeker or recruiter)*	*string*	*20*	*No*	*seeker*	

6.1.2.2 Data 2

JobPosts	
Name	JobPosts
Alias	Jobs
Where-used/how-used	Created by recruiters and linked to applications and shifts.
Content description	Stores job listings posted by recruiters including job type, location, and compensation.

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
jobID	Unique identifier for the job	string	36	No	None	PK
recruiterID	Recruiter who posted the job	string	36	No	None	FK
jobTitle	Title of the job	string	100	No	None	
description	Job description	text	-	Yes	NULL	
location	Location of the job	string	100	No	None	
payRate	Pay offered	float	-	No	0.0	
jobType	Job category (e.g., Part-time)	string	50	No	Part-time	
createdAt	Job posting timestamp	datetime	-	No	current_time	

6.1.2.3 Data 3

Applications	
Name	Applications
Alias	Job Applications
Where-used/how-used	Submitted by seekers; linked to job posts and test scheduling.
Content description	Captures applicant interest, status, and submission details.

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
applicationID	Unique ID for the application	string	36	No	None	PK
seekerID	User ID of the applicant	string	36	No	None	FK
jobID	Job to which the user applied	string	36	No	None	FK
status	Application status	string	20	No	Pending	
submittedDate	Date of application submission	date	-	No	current_date	

6.1.2.4 Data 4

Shifts	
Name	Shifts
Alias	Work Slots
Where-used/how-used	Assigned to selected seekers after application acceptance.
Content description	Stores scheduled shift information for staff working under accepted job posts.

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
shiftID	Unique identifier for the shift	string	36	No	None	PK
jobID	Associated job ID	string	36	No	None	FK
seekerID	Assigned seeker	string	36	No	None	FK
startTime	Start of the shift	datetime	-	No	None	
endTime	End of the shift	datetime	-	No	None	

6.1.2.5 Data 5

Wallets						
Name	Wallets					
Alias	In-app Wallet					
Where-used/how-used	Used for handling user payments and receiving wages.					
Content description	Maintains balance and records for each user's wallet.					
Column Name	**Description**	**Type**	**Length**	**Nullable**	**Default Value**	**Key Type**
walletID	Unique ID for wallet	string	36	No	None	PK
userID	Wallet owner's user ID	string	36	No	None	FK
balance	Current balance in the wallet	float	-	No	0.0	
lastUpdated	Timestamp of last update	datetime	-	No	current_time	

6.1.2.6 Data 6

Notifications						
Name	Notifications					
Alias	In-app Alerts					
Where-used/how-used	Displayed to users in-app to update them about their application, shift, or payments.					
Content description	Stores messages sent to users within the app notification system.					
Column Name	**Description**	**Type**	**Length**	**Nullable**	**Default Value**	**Key Type**
notificationID	Unique ID for notification	string	36	No	None	PK
recipientID	User receiving the message	string	36	No	None	FK
message	Message content	string	255	No	None	
sentDate	When the	datetime	-	No	current_time	

	message was sent					

6.1.2.7 Data 7

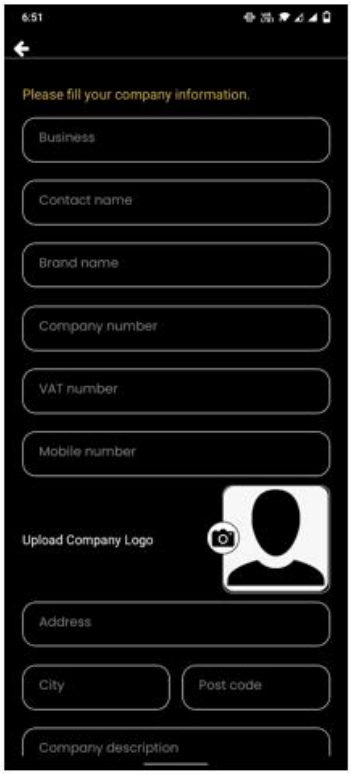
Transactions	
Name	Transactions
Alias	Wallet Transactions / Payment History
Where-used/how-used	Used for recording all in-app wallet activity such as crediting recruiter funds or paying seekers after job completion. Also used for viewing transaction logs and resolving disputes.
Content description	This table logs each financial transaction between users and the system wallet. It includes amounts, transaction types (credit/debit), and timestamps.

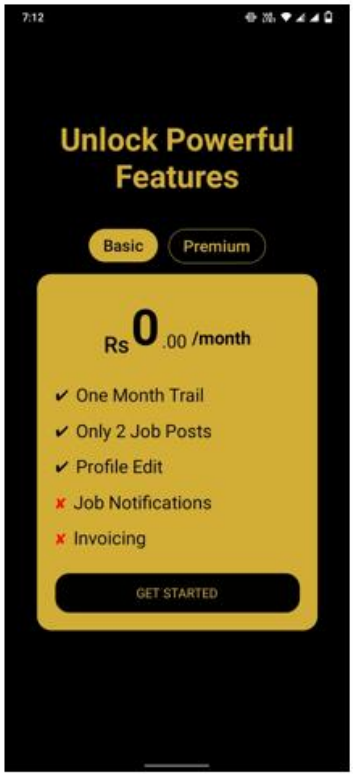
Column Name	Description	Type	Length	Nullable	Default Value	Key Type
transactionID	Unique identifier for each transaction	string	36	No	None	PK
walletID	ID of the wallet involved in transaction	string	36	No	None	FK
amount	Amount of money credited/debited	float	-	No	0.0	
type	Type of transaction (credit/debit)	string	10	No	debit	
description	Short description or remarks	string	255	Yes	NULL	
timestamp	Date and time of transaction	datetime	-	No	current_time	

6.2 GUI Design

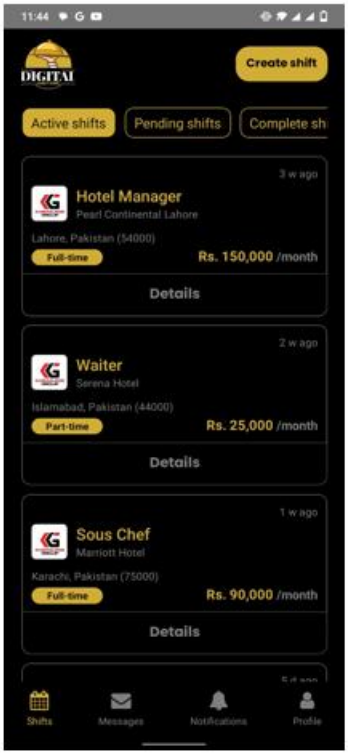
6.2.1 Business View

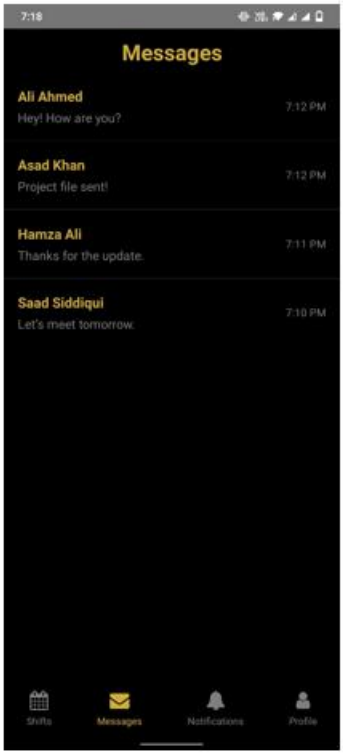
6.2.2 Business View



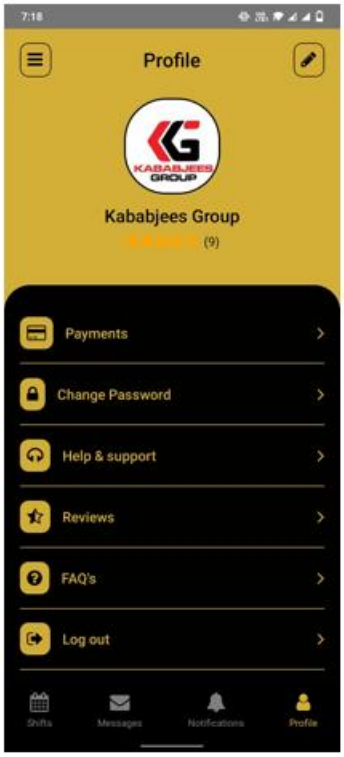


6.2.3 Business View

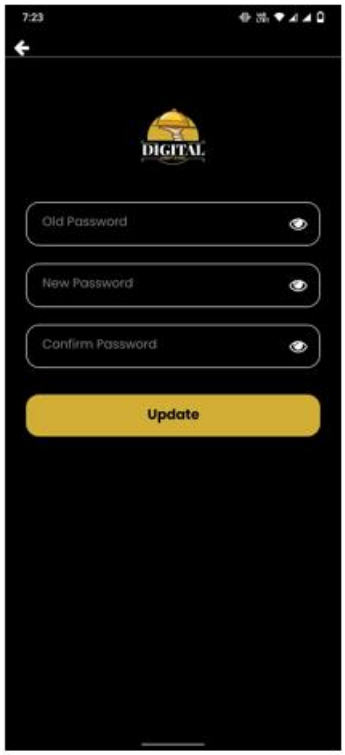


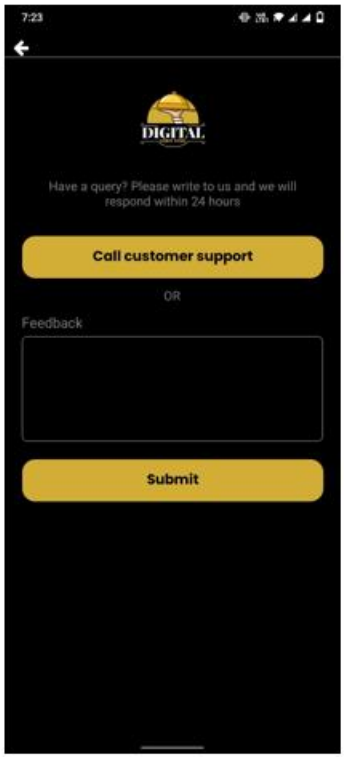


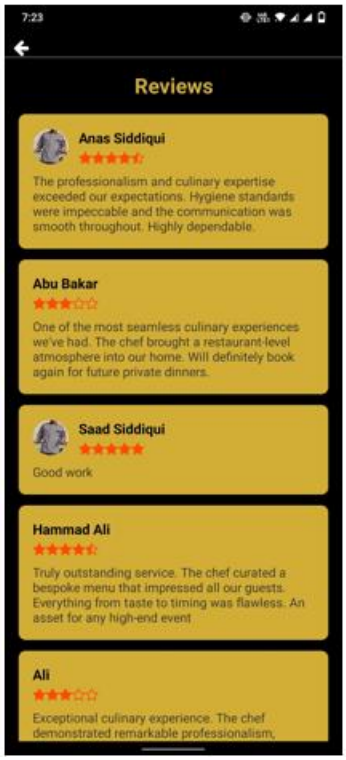
6.2.4 Business View



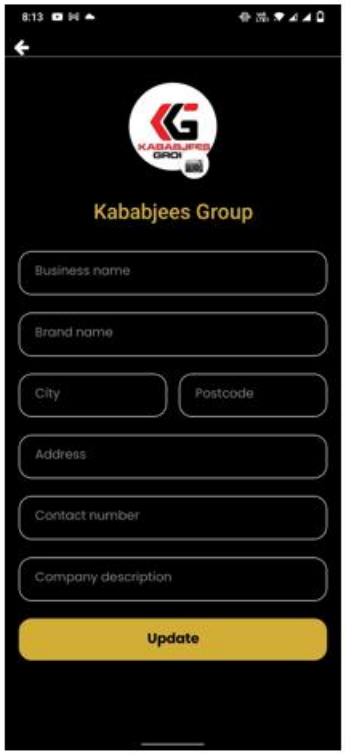
6.2.5 Business View

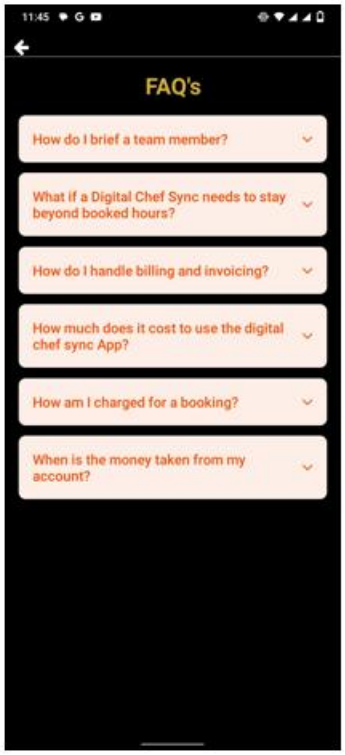




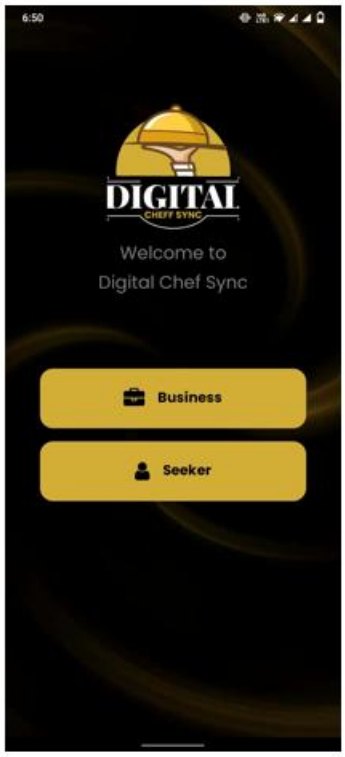


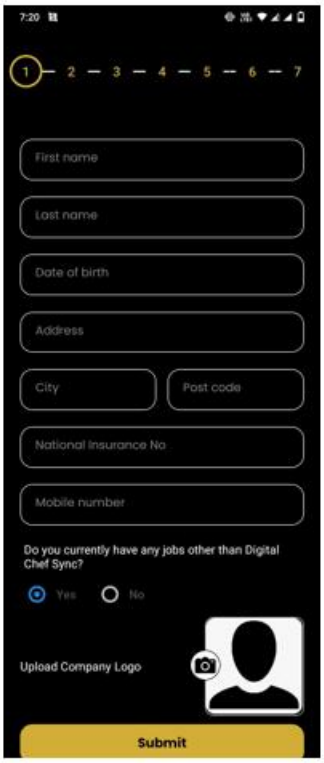
6.2.6 Business View



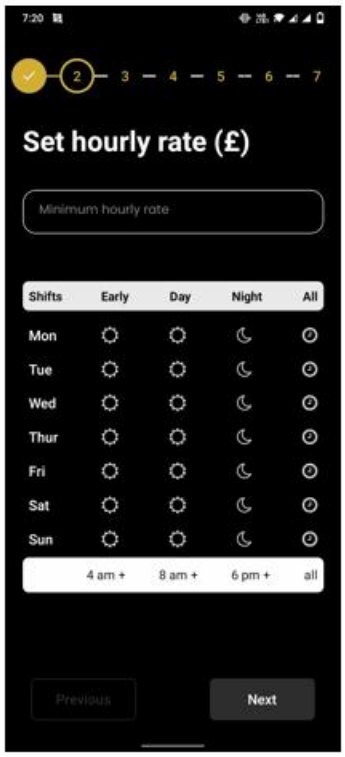


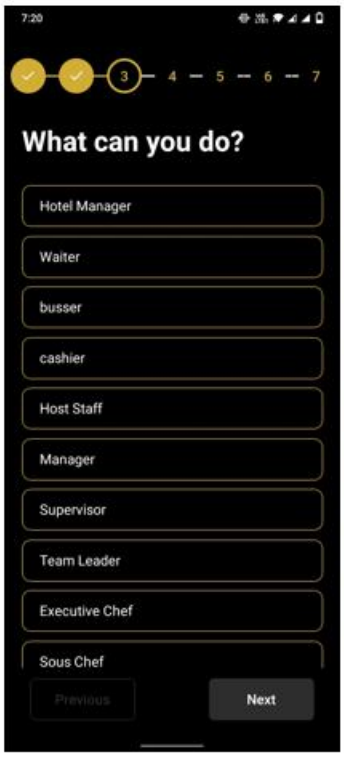
6.2.7 Seeker View

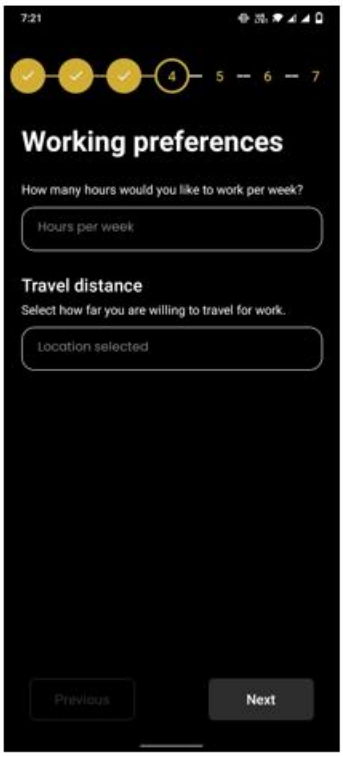




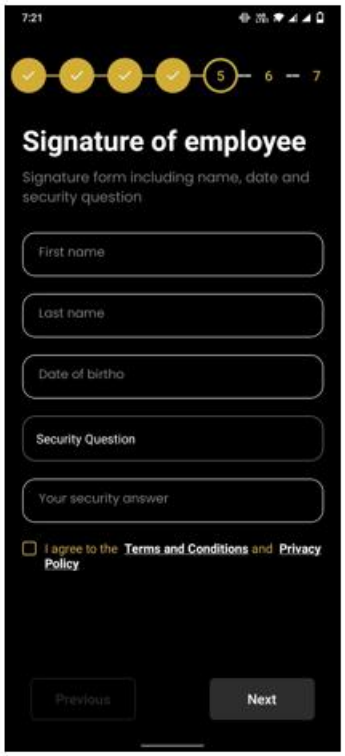
6.2.8 Seeker View

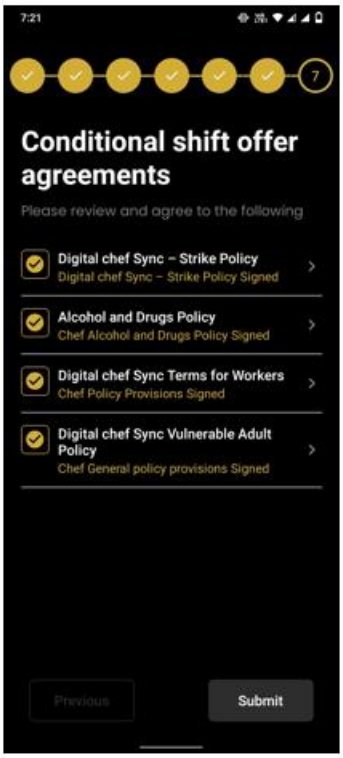




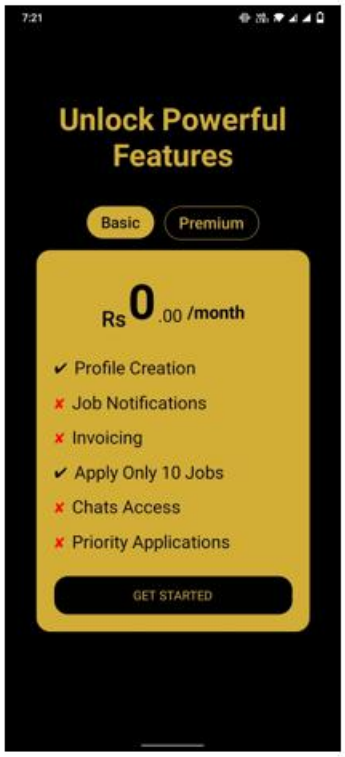


6.2.9 Seeker View

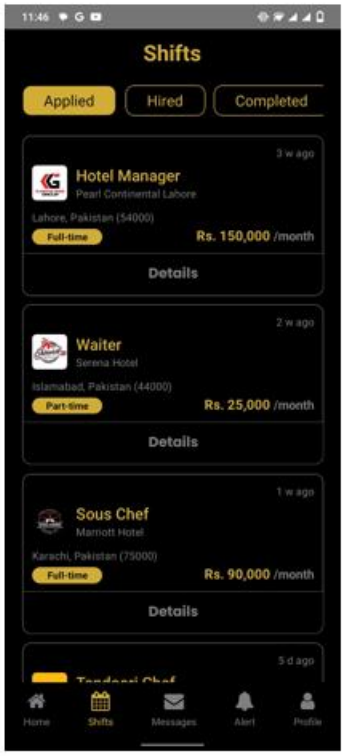


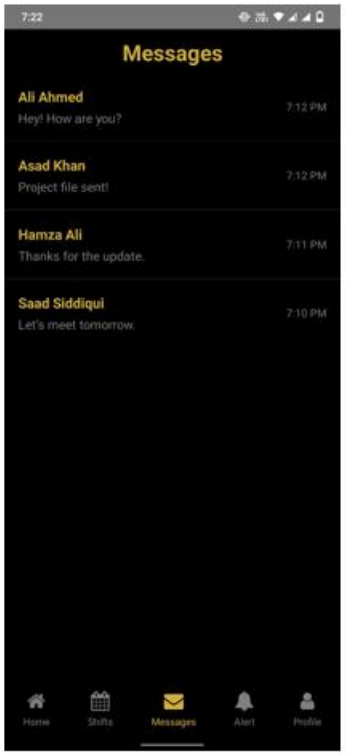


6.2.10 Seeker View

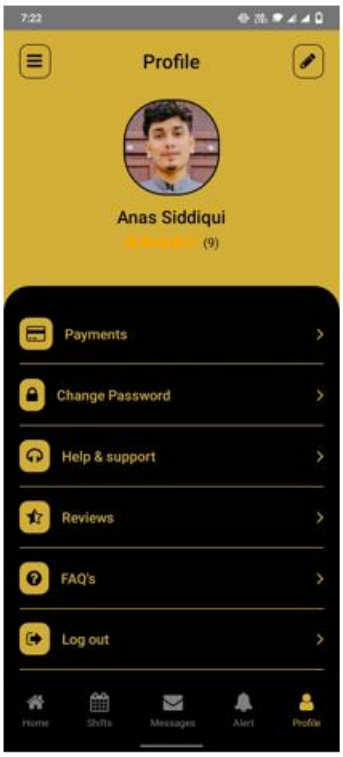


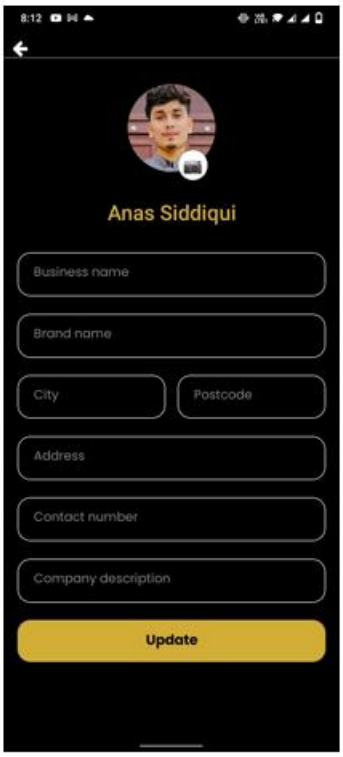
6.2.11 Seeker View





6.2.12 Seeker View





References

- Workstream. (n.d.). ***Restaurant Hiring Software***. Retrieved from https://www.workstream.us/blog/restaurant-hiring-software
- QSR Magazine. (2023). ***How Tech Is Changing Restaurant Staffing***. Retrieved from https://www.qsrmagazine.com/emerging-tech/how-tech-changing-restaurant-staffing
- Toast. (n.d.). ***How to Hire Restaurant Staff (The Right Way)***. Retrieved from https://pos.toasttab.com/blog/on-the-line/hiring-restaurant-staff
- Scrum Guides. (2020). ***The Scrum Guide™ – The Definitive Guide to Scrum***. Retrieved from https://scrumguides.org
- Lucidchart. (n.d.). ***How to Make a Sequence Diagram***. Retrieved from https://lucidchart.com/blog/sequence-diagram-tutorial
- Firebase. (n.d.). ***Firebase Documentation***. Retrieved from https://firebase.google.com/docs
- MongoDB. (n.d.). ***MongoDB Documentation***. Retrieved from https://www.mongodb.com/docs

Appendices