## **1 Introduction**

### **1.1 Purpose**

This document outlines all the specific requirements and scope of FastBite, a fast food ordering platform. It intends to identify the key features and requirements of the platform, along with the corresponding space, time, and data constraints. The document also defines the design constraints for the platform through UML and Use Case Diagrams.

This document provides developers, stakeholders, and testers with a comprehensive reference to all functionality and constraints. In case of ambiguity or missing details, this document should be the first point of reference before contacting the project team.

### **1.2 Scope**

FastBite will support two synchronized portals—**Customer Portal** and **Restaurant Portal**—both connected to a shared database and real-time event stream.

* **In Scope**: Account creation/login, menu browsing, cart & checkout, real-time order tracking, restaurant order management, driver assignment, inventory updates, sales reporting, and notifications.
* **Out of Scope**: Delivery logistics beyond driver assignment (e.g., route optimization), third-party integrations not specified, and financial record-keeping outside order payments.

### **1.3 Definitions**

* **Customer** - End-user placing food orders.
* **Manager** - Employee(s) managing incoming orders, inventory, and drivers.
* **Driver** - Delivery personnel assigned by restaurant staff to fulfill orders.
* **Order Lifecycle** - Defined state transitions: *New → Accepted → In Prep → Ready → Out for Delivery → Delivered*.
* **Cart** - List of menu items placed by the customer.
* **Order** -List of menu items that are to be delivered by the delivery personnel.
* **Dashboard** - Restaurant interface for live order management.

### **1.4 References**

* IEEE 830-1993: Recommended Practice for Software Requirements Specifications
* MongoDB Atlas Documentation
* React, Express.js, and WebSocket developer documentation

### **1.5 Overview of SSR**

This document is structured as follows:

* **Section 2: Overall Description** - High-level system design, interfaces, constraints, assumptions.
* **Section 3: Specific Requirements** - Functional and non-functional requirements organized by feature, database constraints, and system attributes.
* **Section 4: Supporting Information** - Incremental development plan, team contributions, index.

## **2 Overall Description**

Don’t have the time to drive or wait in line? FastBite offers you the convenience of ordering food at your fingertips. Our user-friendly web application allows you to easily order and customize your food, seamlessly pay for your order, and track your delivery in real-time!

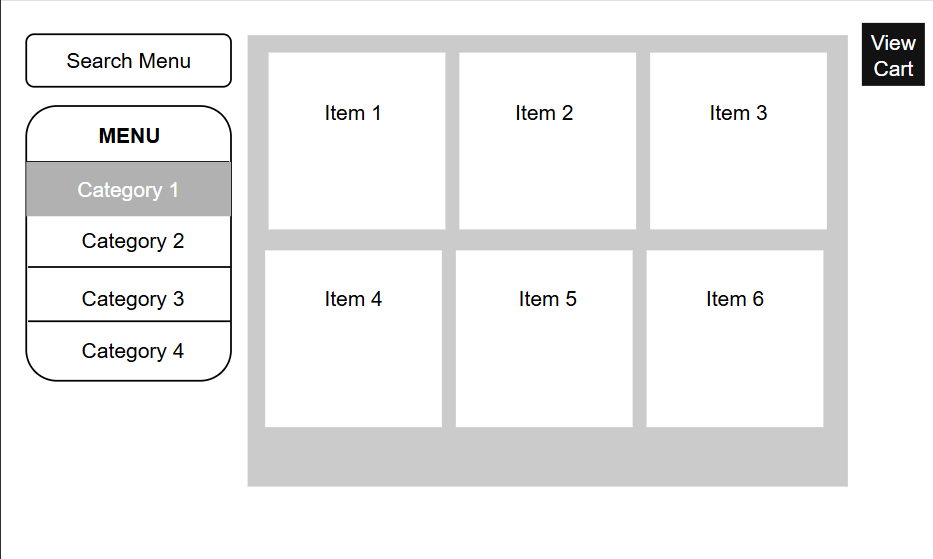
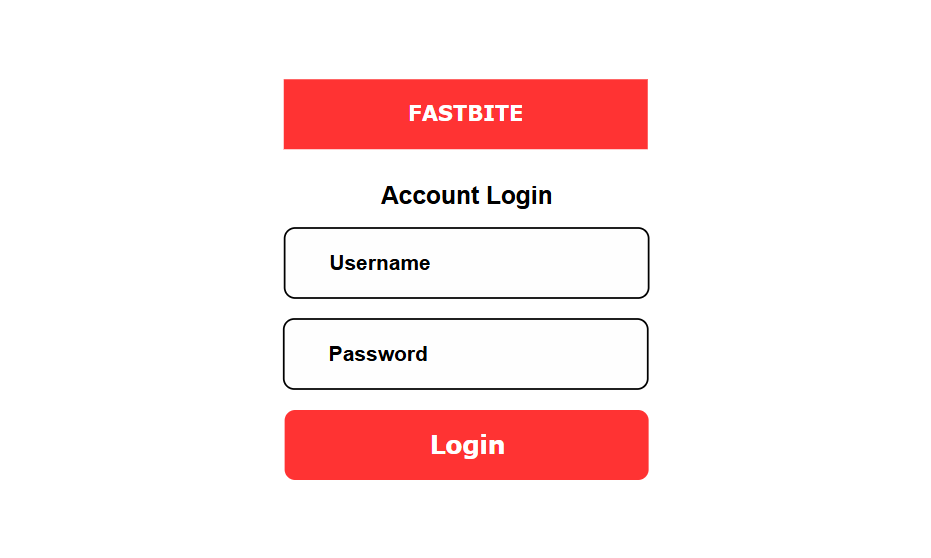
### **2.1 Product Perspective**

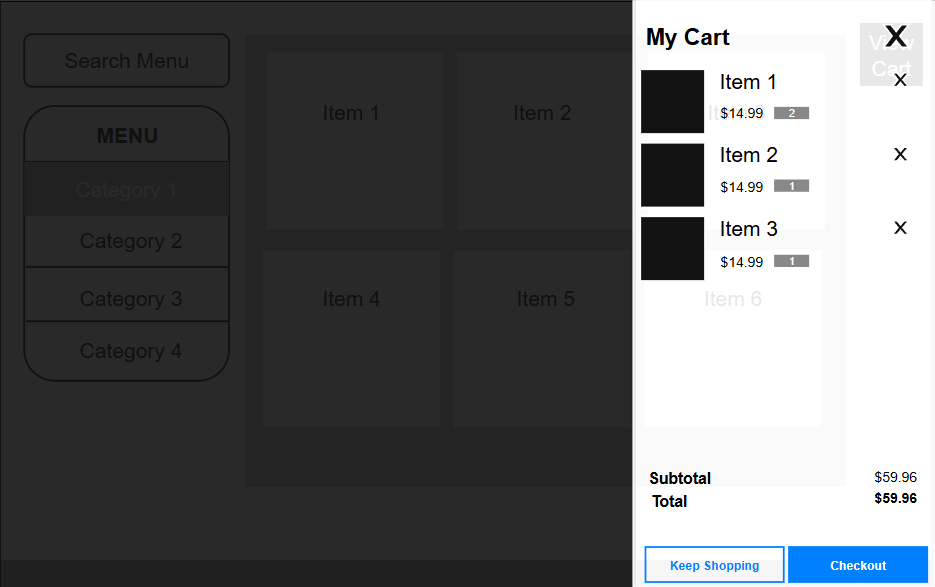
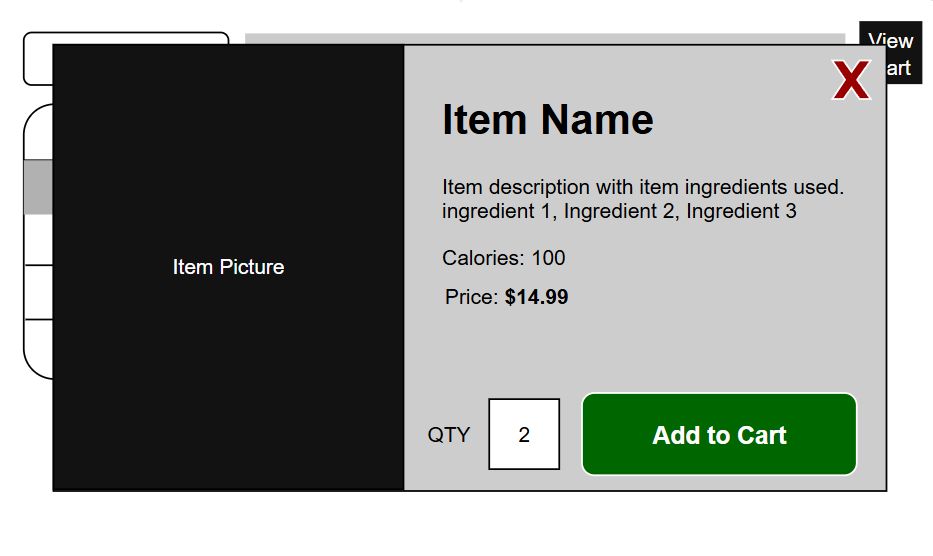
FastBite is a new web-based application built with **React (frontend)**, **Express.js (backend)**, and **MongoDB (database)**. It will feature two portals with synchronized real-time updates via WebSockets.

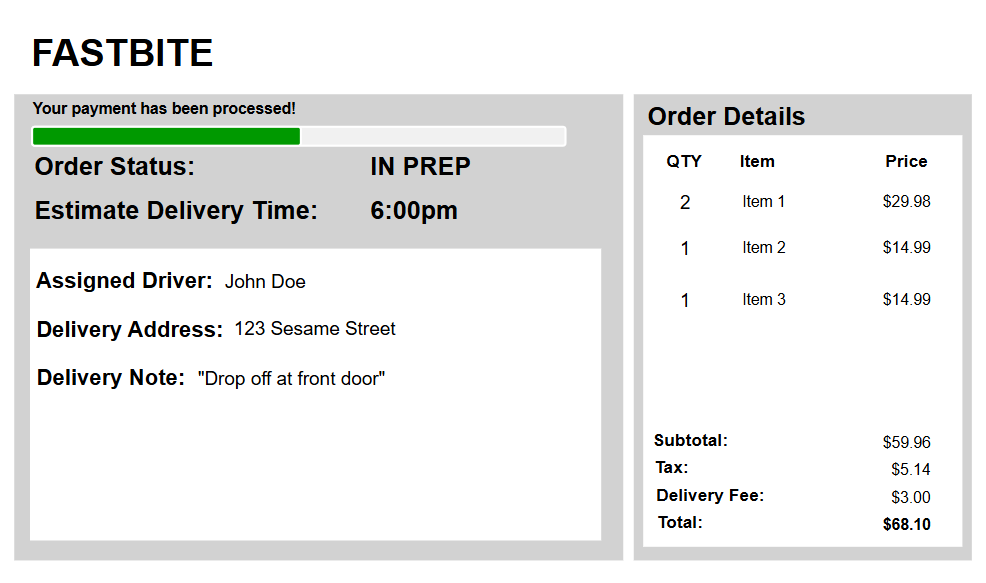
#### **2.1.1 System Interfaces**

* **Frontend**: Web browsers on desktop and mobile.
* **Backend**: REST APIs + WebSocket channels.
* **Database**: MongoDB Atlas accessed via Mongoose.

#### **2.1.2 User Interfaces**







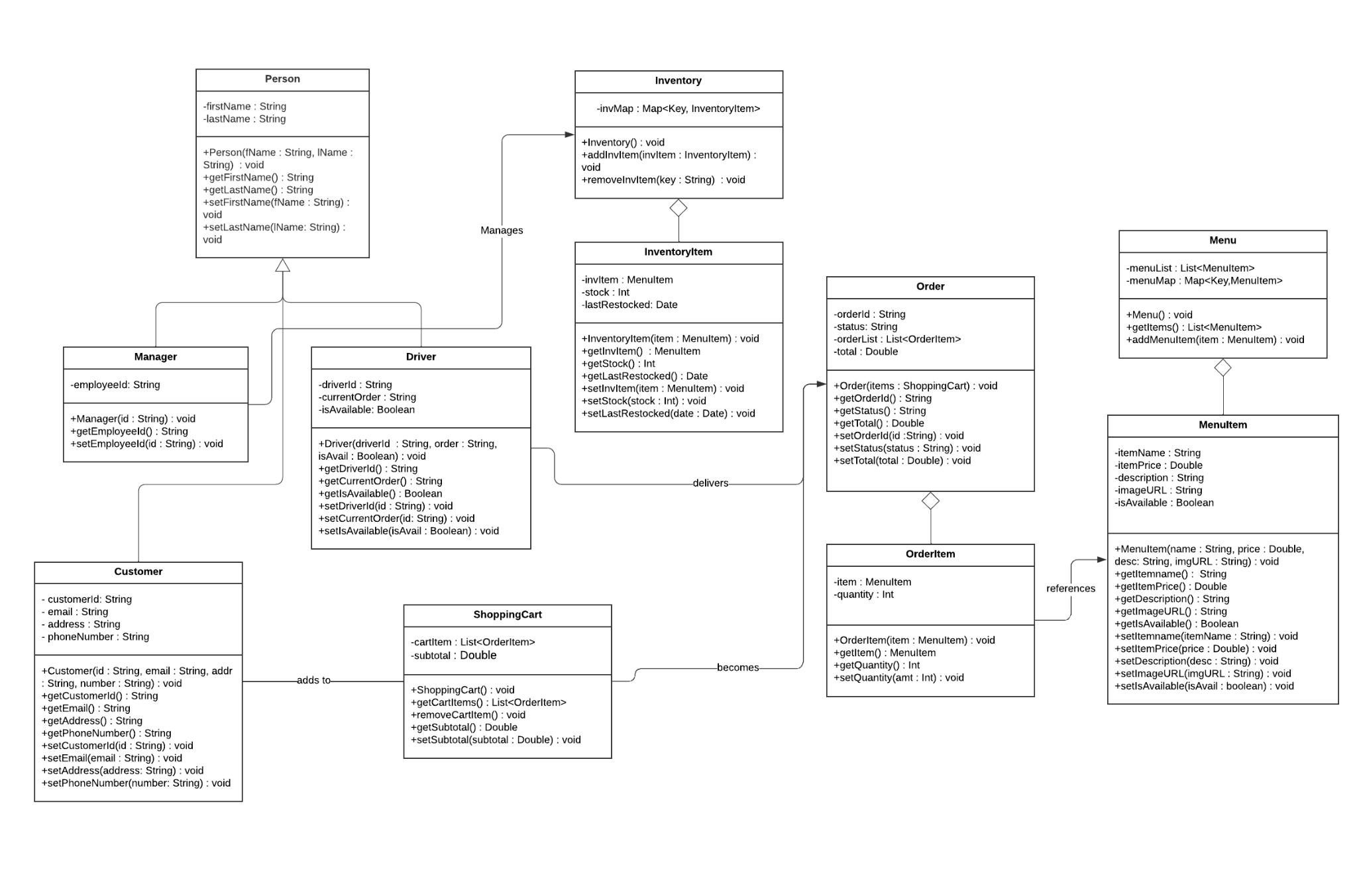
**Customer Portal**

* Home/Menu with categories, search, and item details
* Cart & checkout with modifiers, taxes, and notes
* Order tracker with progress bar and ETA
* Profile for saved addresses and payment methods

**Restaurant Portal**

* Orders dashboard with live queue
* Menu builder (CRUD operations)
* Driver assignment panel
* Inventory and analytics views

#### **2.1.2.1 UML Class Diagram**



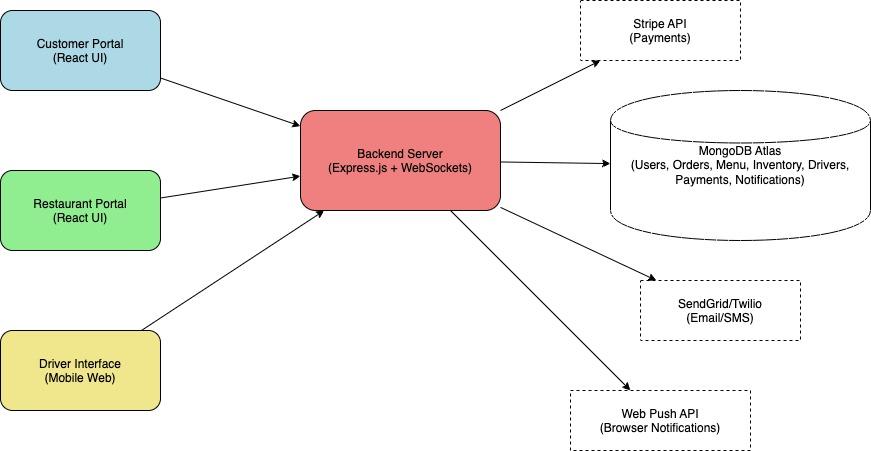
#### **2.1.3 System Structure Diagram and Descriptions**

FastBite has these main subsystems:

* **Customer Portal (Frontend)**
* **Restaurant Portal (Frontend)**
* **Backend Server (APIs + WebSocket)**
* **Database (MongoDB Atlas)**
* **Payment Service (Stripe -external)**
* **Notification Service (SendGrid/Twilio/Web Push -external)**

All of these systems will communicate with each other primarily through the **Backend Server**.

* **Customer Portal (React Frontend)**Lets customers browse menus, add items to cart, checkout, track orders, and manage profiles. Communicates with the backend via REST + WebSocket.
* **Restaurant Portal (React Frontend)**Used by staff to manage orders, assign drivers, update inventory, and view analytics. Real-time updates via WebSocket.
* **Backend Server (Express.js)**The core application layer. Handles authentication, order processing, driver assignment, notifications, and exposes REST + WebSocket APIs.
* **Database (MongoDB Atlas)**Stores all persistent data: users, menu items, orders, drivers, payments, and notifications.
* **Payment Service (Stripe)**External API for secure online payment processing (Phase 2).
* **Notification Service (Twilio/SendGrid/Web Push)** External APIs for sending SMS, email, and browser push notifications.

System Structure Diagram

#### **2.1.4 Hardware Interfaces**

No specific hardware beyond standard desktop/mobile devices.

#### **2.1.5 Software Interfaces**

* **Stripe** for online payments (Phase 2).
* **SendGrid/Twilio** for email/SMS notifications.
* **Web Push API** for browser notifications.

#### **2.1.6 Communication Interfaces**

* HTTPS for REST APIs.
* Secure WebSockets (WSS) for real-time order updates.

#### **2.1.7 Memory Constraints**

Minimal; application must avoid memory leaks. MongoDB Atlas autoscaling will ensure adequate storage.

#### **2.1.8 Operations**

The system will support concurrent customer orders and live updates to both customer and restaurant portals.

#### **2.1.9 Site Adaptation Requirements**

No special requirements beyond cloud hosting (MongoDB Atlas, Node.js server).

### **2.2 Product Functions**

* Customers can browse the menu, customize items, add to cart, and place orders.
* Customers can pay (cash or online via Stripe) and track orders in real time.
* Restaurant staff can view, accept, or reject orders, update inventory, and assign drivers.
* Drivers can update status when picking up and delivering orders.
* Notifications are sent at key order stages.
* Analytics reports summarize sales, top items, and delivery times.

### **2.3 User Characteristics**

* **Customers**: Tech-savvy, familiar with online ordering.
* **Restaurant Staff**: Require simple dashboards with clear workflows.
* **Drivers**: Minimal interaction—primarily update delivery milestones.

### **2.4 Constraints**

* Budget limited to free or student versions of tools.
* PCI-DSS compliance required for payment handling.
* Web applications must be responsive.

### **2.5 Assumptions and Dependencies**

* Reliable internet access is assumed.
* Stripe availability for online payments.
* SMS/Email delivery dependent on external providers.

### **2.6 Apportioning of Requirements**

Phase 1 focuses on Minimal Viable Product (MVP) features (cash-on-delivery payments, core ordering). Advanced features like Stripe payments and analytics will be developed in later phases.

## **3 Specific Requirements**

### **3.1 External Interface Requirements**

* **Keyboard/Touch Input**: Text entry for login, address, and notes.
* **Monitor/Mobile Screen**: Responsive UI displays all interfaces.
* **Notifications**: Push, email, or SMS messages delivered through third-party APIs.

### **3.2 Functional Requirements (by Feature)**

#### **3.2.1 Customer Account**

* Register/Login/Logout using JWT authentication.
* Manage profile (addresses, saved payment methods).
* View past orders and reorder.

#### **3.2.2 Menu & Inventory**

* Browse menu categories, search, and filter items.
* Item detail supports modifiers (size, toppings, spice level).
* “Sold Out” items dynamically hidden or marked.
* Staff can add/edit/delete menu items and categories.

#### **3.2.3 Cart & Checkout**

* Add/edit/remove items in cart.
* Apply coupons (Phase 2).
* Checkout with Cash on Delivery (Phase 1) or Stripe payment (Phase 2).

#### **3.2.4 Order Tracking**

* Real-time status updates through WebSocket.
* Order lifecycle: New → Accepted → In Prep → Ready → Out for Delivery → Delivered.
* Progress bar and ETA visible to customers.

#### **3.2.5 Driver Management**

* Staff assign/unassign drivers.
* Drivers mark status updates (picked up, delivered).

#### **3.2.6 Notifications**

* Email/SMS/Web Push for order confirmation, updates, and delivery.

#### **3.2.7 Analytics**

* Sales summaries, top-selling items, cancellation rates.
* Delivery lead time metrics.

#### **3.2.8 Chatbox/Help**

* FAQ/help widget for customers.
* Optional live chat handoff to staff in Phase 3.

### **3.3 Non-Functional Requirements**

#### **3.3.1 Space Requirement**

The app must remain lightweight (<250 MB deployment size).

#### **3.3.2 Time Requirements**

* API latency: ≤ 300 ms for reads, ≤ 600 ms for writes.
* WebSocket updates within 2 seconds.

#### **3.3.3 Reliability**

Graceful handling of failures; WebSocket fallback to polling.

#### **3.3.4 Availability**

99.5% uptime during business hours.

#### **3.3.5 Security**

* TLS 1.2+ for all communications.
* Passwords hashed; JWT with expiry/refresh.
* No raw credit card storage.

#### **3.3.6 Maintainability**

Clean modular code with TypeScript on backend, ESLint/Prettier enforced.

#### **3.3.7 Portability**

Runs on all modern browsers; mobile responsive.

### **3.4 Logical Database Requirements**

* Collections: Users, MenuItems, Inventory, Orders, Drivers, Payments, Notifications.
* Relationships: Orders reference Users, MenuItems, and Drivers.
* Integrity: Transactions ensure consistent order/payment states.

### **3.5 Design Constraints**

* Stripe integration requires external API keys and secure webhook handling.
* MongoDB Atlas used as the primary database.

### **3.6 Software System Attributes**

* **Reliability**: Error rate < 1% requests.
* **Scalability**: Support up to 200 concurrent orders.
* **Usability**: Accessible UI (WCAG 2.1 AA).
* **Observability**: Logging and monitoring via structured logs and metrics.

## **4 Supporting Information**

### **4.1 Table of Contents**

(Will mirror Example Templates: Introduction, Overall Description, Specific Requirements, Supporting Information, Incremental Plan, Contributions, Index.)

### **4.2 Index**

Keywords: Account, Menu, Orders, Driver, Notifications, Stripe, Inventory, Dashboard.

### **4.3 Appendices**

#### **4.3.1 Incremental Development Plan**

| **Feature** | **Phase** | **Task Description** | **Technologies** |
| --- | --- | --- | --- |
| Customer Account | 1 | Implement registration/login/logout with JWT | React, Express, Mongo |
| Menu & Inventory | 1 | CRUD operations for menu and inventory | React, MongoDB |
| Cart & Checkout | 1 | Add/remove items, CoD payment | React, Express |
| Orders Dashboard | 1 | Real-time order updates for restaurant | WebSockets, Express |
| Stripe Payments | 2 | Integrate Stripe Checkout + webhook handling | Express, Stripe API |
| Notifications | 2 | Email/SMS/Web Push at order lifecycle events | Twilio/SendGrid, WebPush |
| Driver Management | 2 | Drivers can update pickup/delivery status | React, MongoDB |
| Analytics Reports | 3 | Generate sales, top items, cancellations, SLA metrics | MongoDB Aggregations |
| Chatbox/Help | 3 | Implement live chat widget for customer support | React, WebSockets |

#### 

#### **4.3.2 Team Contributions**

| **Member** | **Contribution Description** | **%** |
| --- | --- | --- |
| Neel Panajkar | System Structure Diagram and Descriptions | 20 |
| Kaksh Patel | Requirements documentation, functional specs | 20 |
| Tyler Quach | UI design and mockup | 20 |
| Roger Lin | UML Diagram, intro and overall description, doc organization | 20 |
| Aayush Niroula | System Specification Document Outline and Planning | 20 |