**Team Deliverable 3 – Project Phase I**

Course: ITIS‑3300

Team: Lab Rats | Date: October 28, 2025

Project: FastByte – Food Ordering Platform

**Introduction**

FastBite is a full-stack web-based fast-food ordering system designed to simplify food ordering for both customers and restaurants. In this first implementation phase (Phase I), our team focused on developing the foundational functionality of the system, which includes menu browsing and search, cart management (add/edit/remove), and MongoDB database integration. These elements form the Minimum Viable Product (MVP) core that ensures customers can explore the menu, configure items in their cart, and that all data interactions are properly stored and retrieved in real time through MongoDB.

**1. Requirements (Phase 1)**

**1.1 Functional Requirements Implemented (Phase 1)**

* **User & Auth**: allows both customers and employees to securely sign up and log in, using hashed passwords for security.
* **Menu & Inventory**: CRUD for categories and menu items; item availability flag; image URL field optional.
* **Health & Meta:** Provides / and /api/ping endpoints for API status checks and version information, with CORS configured to allow requests from http://localhost:3000 during development.

**1.2 Deferred/Out‑of‑Scope in Phase 1**

* Driver assignment and live delivery tracking (Phase 2).
* Realtime updates over WebSocket (Phase 2).
* Inventory management (Phase 2).
* Frontend-Backend API Integration (Phase 2).
* JWT Authentication (Phase 2).
* Cart System and Checkout System (Phase 2).

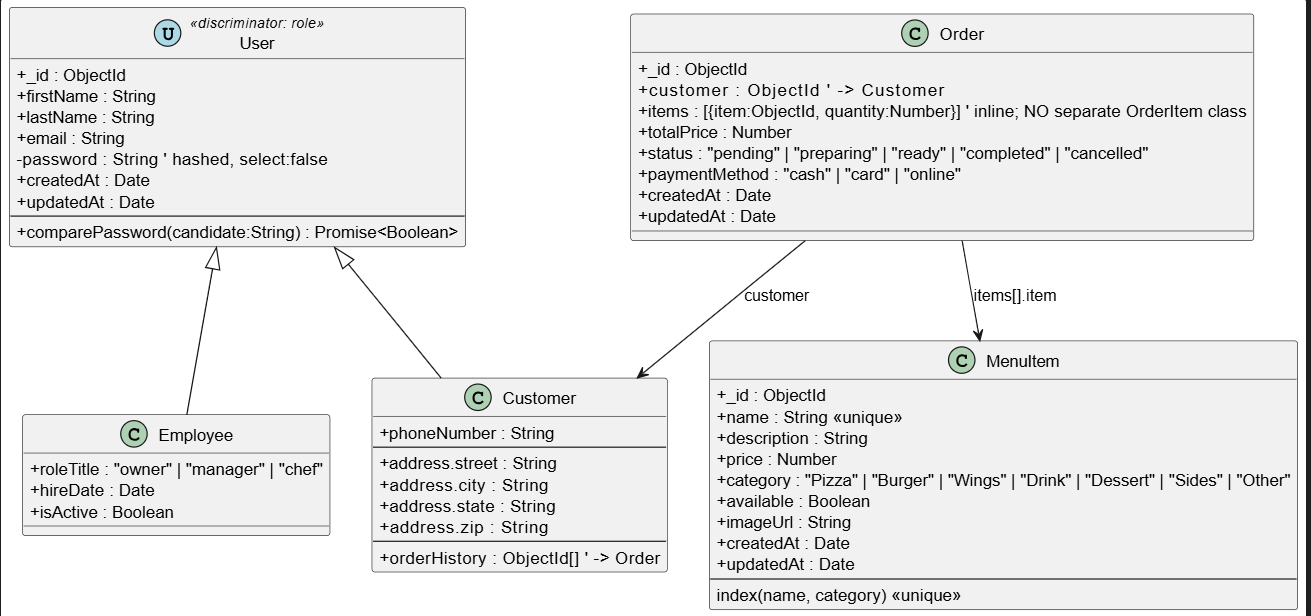
**1.3 Changes since Deliverable 2**

No change in overall scope. Implementation is staged: API finished; frontend temporarily hardcoded to expedite UI layout sign‑off. Live integration is planned for Phase 2.

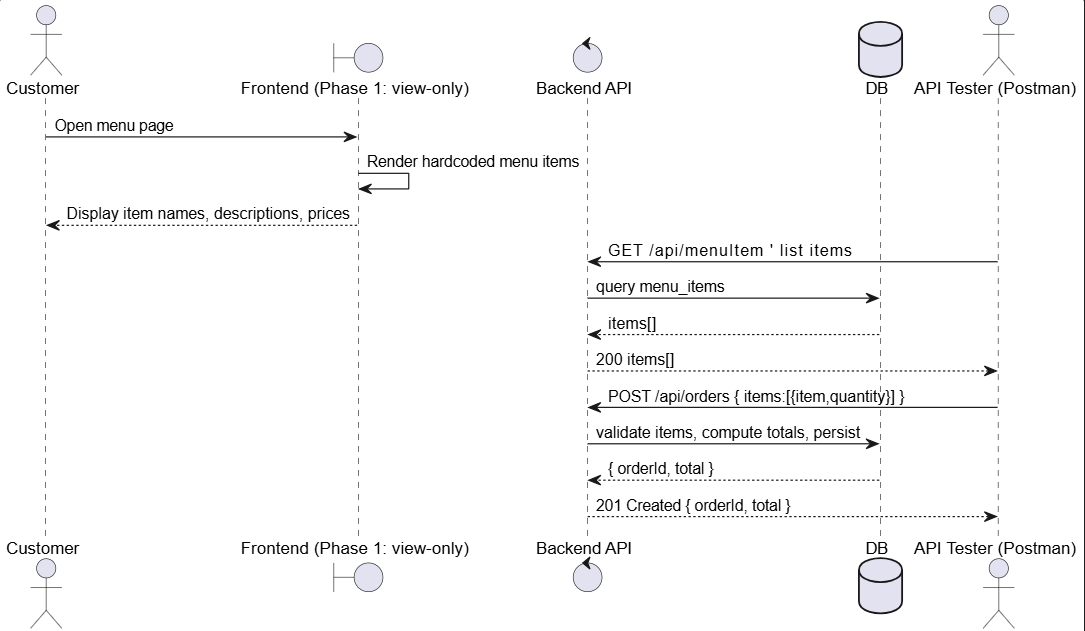
**2. UML Design (Phase 1)**

Below are concise UML artifacts for the Phase 1 backend domain. (Front‑end is mocked/hardcoded.)

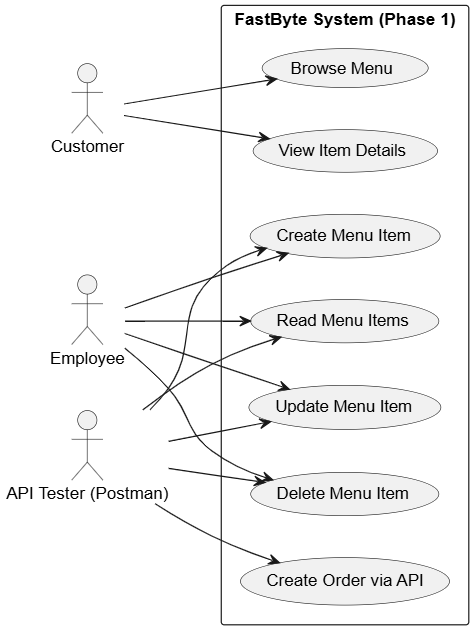
**2.1 Class Diagram**



**2.2 Sequence Diagram (Place Order)**



**2.3 Use Case Diagram**



**3. Test Cases**

All tests executed with Postman and curl against the local backend. Frontend pages are static; no integration tests yet.

| Name | Endpoint | Expected Result |
| --- | --- | --- |
| Health Root | GET / | 200 OK — returns "FastByte API is running" text. |
| Ping | GET /api/ping | 200 OK — JSON { ok: true, msg: "pong" }. |
| Signup Customer | POST /api/customers/signup { firstName, lastName, email, password } | 201 Created — JSON with user ID; duplicate email → 409. |
| Login Customer | POST /api/customers/login { email, password } | 200 OK — JSON with token; invalid credentials → 401. |
| Create Menu Item | POST /api/menuItem { name, description, price, category, available } | 201 Created — new item added; missing fields → 400. |
| List Menu Items | GET /api/menuItem | 200 OK — returns JSON array of menu items. |
| Get Menu Item | GET /api/menuItem/:id | 200 OK — single item data; 404 if not found. |
| Update Menu Item | PUT /api/menuItem/:id { ... } | 200 OK — updated item; invalid data → 400; not found → 404. |
| Delete Menu Item | DELETE /api/menuItem/:id | 204 No Content — on success; 404 if not found. |
| Create Order | POST /api/orders { items:[{item,quantity}] } | 201 Created — order stored with calculated total; unavailable item → 409; invalid body → 400. |

**4. User Manual (End‑User)**

For Phase 1, the FastByte application includes a static, hardcoded frontend that represents the customer’s interaction flow without connecting to the backend API. The goal of this phase is to demonstrate the intended user experience and layout while backend functionalities are tested separately using Postman.

When launching the frontend, users can navigate through the menu pages to explore available food categories and items. Each item displays sample details such as name, description, price, and availability. Users can add items to a simulated cart, proceed to a checkout page, and click “Place Order” to experience the visual order flow.

The checkout confirmation screen provides a static success message confirming that the order has been placed; however, no real order data is sent to or stored in the database at this stage. All interactions occur locally within the browser.

This prototype allows end-users to understand how browsing, cart management, and checkout will look and feel once API integration is completed in Phase 2, where actions such as placing orders, logging in, and managing accounts will become fully functional.

**5. Compilation / Run Instructions**

Prerequisites:- [Node.js](http://node.js) v.18 or higher, npm, MONGO\_URI

1. In the root folder, create .env.local and attach the MONGO\_URI = <String>
2. Open terminal on the root folder and run”npm install”
3. In the same terminal run “npm run dev”
4. To access the web-app go to <https://localhost:3000>
5. To access the back-end, Open testing application like POSTMAN and run <https://localhost:3000/api/ping>

**6. Reflection**

We completed the backend API and validated core flows with Postman. Keeping the frontend hardcoded lets us finalize layout and content quickly. Next, we will wire the UI to live endpoints, add payments and notifications, and introduce realtime updates with proper error handling.

| **Member** | **Contribution Description** | **Overall %** | **Notes** |
| --- | --- | --- | --- |
| **Neel Panajkar** | UML diagram creation | 15 | — |
| **Kaksh Patel** | Cart Functionalty | 15 | — |
| **Roger Lin** | Frontend Development; UI page; layout and styling | 35 | — |
| **Aayush Niroula** | Backend Development, Database Integration, controllers and API Logic | 35 | — |