# Team Deliverable 5 – Project Phase III (Final Product)

**Project:** FastByte – Full-Stack Food Ordering & Restaurant Management System
**Course:** ITIS 3300
**Team:** Lab Rats
**Members:** Kaksh Patel, Aayush Niroula, Neel Panajkar, Roger Lin
**Date:** December 2, 2025

## 1. Requirements for Phase III

### 1.1 Project Overview Across Phases

FastByte is our team's web-based food ordering and restaurant management system. Throughout the semester, we built it step-by-step. In Phase I we focused mainly on the backend, setting up MongoDB and building the basic API routes for signing up, logging in, viewing menu items, and creating orders. At that point the frontend was mostly static.

In Phase II we focused more on connecting the backend to the frontend, and building the first version of the restaurant dashboard. Customers were able to browse a basic menu, log in, and add things to a cart.

Now, in Phase III, we turned everything into a complete full-stack Next.js application. We polished both portals, finished the cart system, improved the dashboard, and made the app feel more like a real product. Overall, this phase was about pulling everything together into something functional and presentable.

### 1.2 Phase III Detailed Requirements

#### Customer Portal

- Customers can browse the menu, which is organized by categories like Burgers, Sides, Drinks, and Desserts.
- There is a search/filter option so customers can quickly find items.
- Customers can add things to the cart, change quantities, or remove items.
- The cart shows totals.
- The cart is saved using localStorage, so refreshing the page keeps the items.

- A Track Order screen shows a visual timeline of an order's progress after checkout.

**Restaurant Portal**

- Staff can open a dashboard that displays all active orders in a queue.
- They can see order details such as items, order number, timestamps, and status.
- Staff can filter based on categories like All Orders, Kitchen Queue, Delivery, and Dine-In.
- Staff can update the order status step-by-step.
- They can also delete old or test orders.

**Backend & Data**

- The backend uses Next.js API routes.
- MongoDB stores menu items and orders.
- Validation checks make sure requests include required data.
- Some routes check for x-customer-id headers for cart operations.

**Testing & Quality**

- We used Postman to test the backend.
- We also performed manual testing on the frontend.
- We documented test cases for both the API and UI.

### 1.3 Scope Changes from Earlier Phases

Even though our overall idea stayed the same, we changed what we prioritized:

**Things we completed successfully:**

- CRUD for menu items with MongoDB.
- Customer menu browsing and cart that actually works.
- Restaurant dashboard with order management.
- Basic authentication.

**Things we reduced or postponed:**

- Full role-based authentication (Customer vs Staff vs Admin) is not fully enforced in this version.
- No payment system like Stripe yet.
- No real-time updates (we originally planned WebSockets).
- No complete Admin panel for inventory and menu updates.

These choices helped us finish a stable product on time.

## 2. System Design and UML

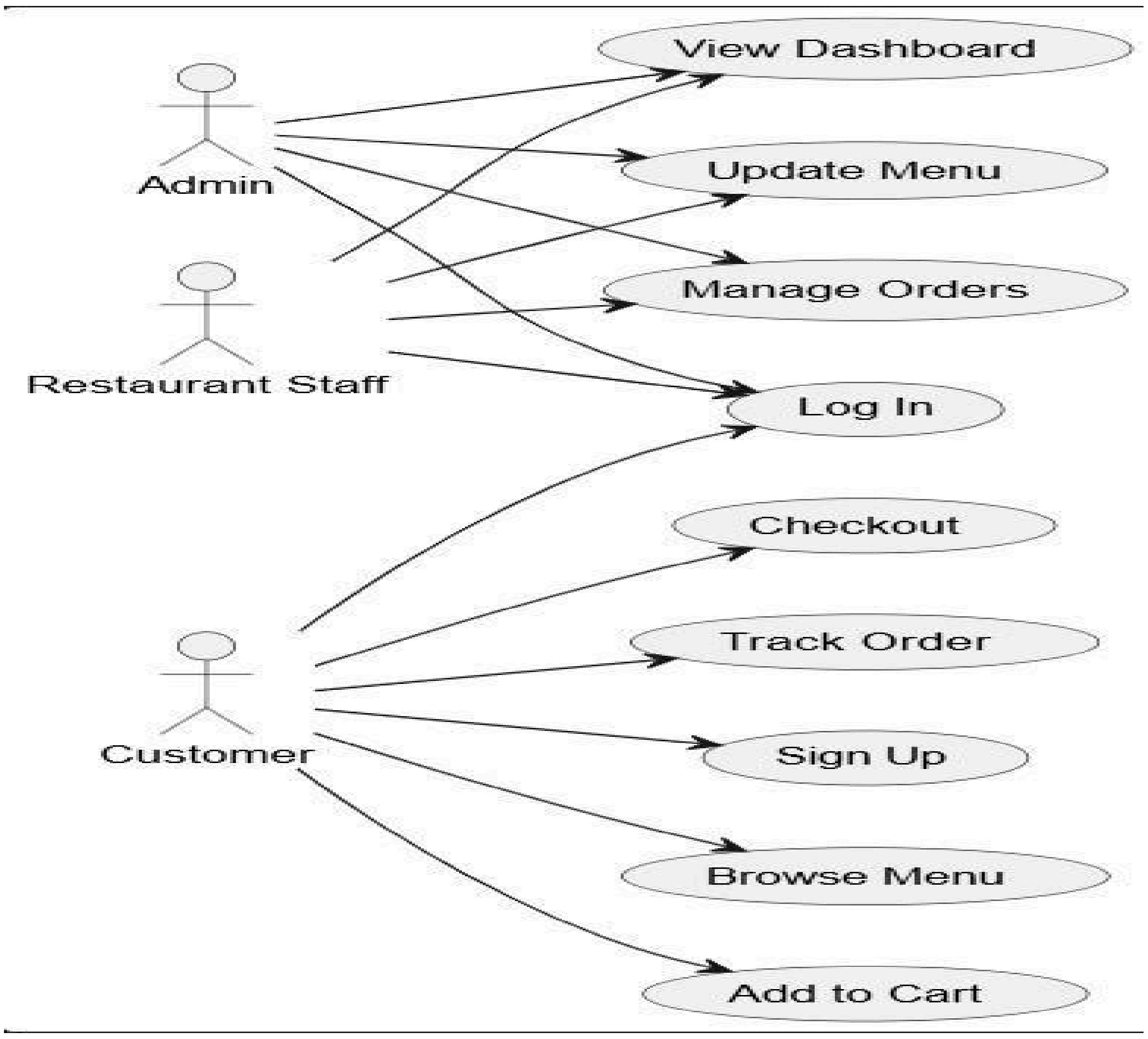
We reused the diagrams from Phase II and updated them where needed.

### 2.1 Use Case Diagram (Summary)

The system has three main actors:

- **Admin** – manages menu and global settings
- **Restaurant Staff –** handles orders
- **Customer –** browses menu, adds to cart, checks out, and tracks orders

Major use cases include browsing the menu, adding items to the cart, managing orders, and updating statuses.



**2.2 Class Diagram (Summary)**

The backend mostly revolves around controllers and MongoDB models.

**Middleware**

- Protects routes if needed.
- Checks cookies and user sessions.
- Redirects or denies access when required.

**Controllers**

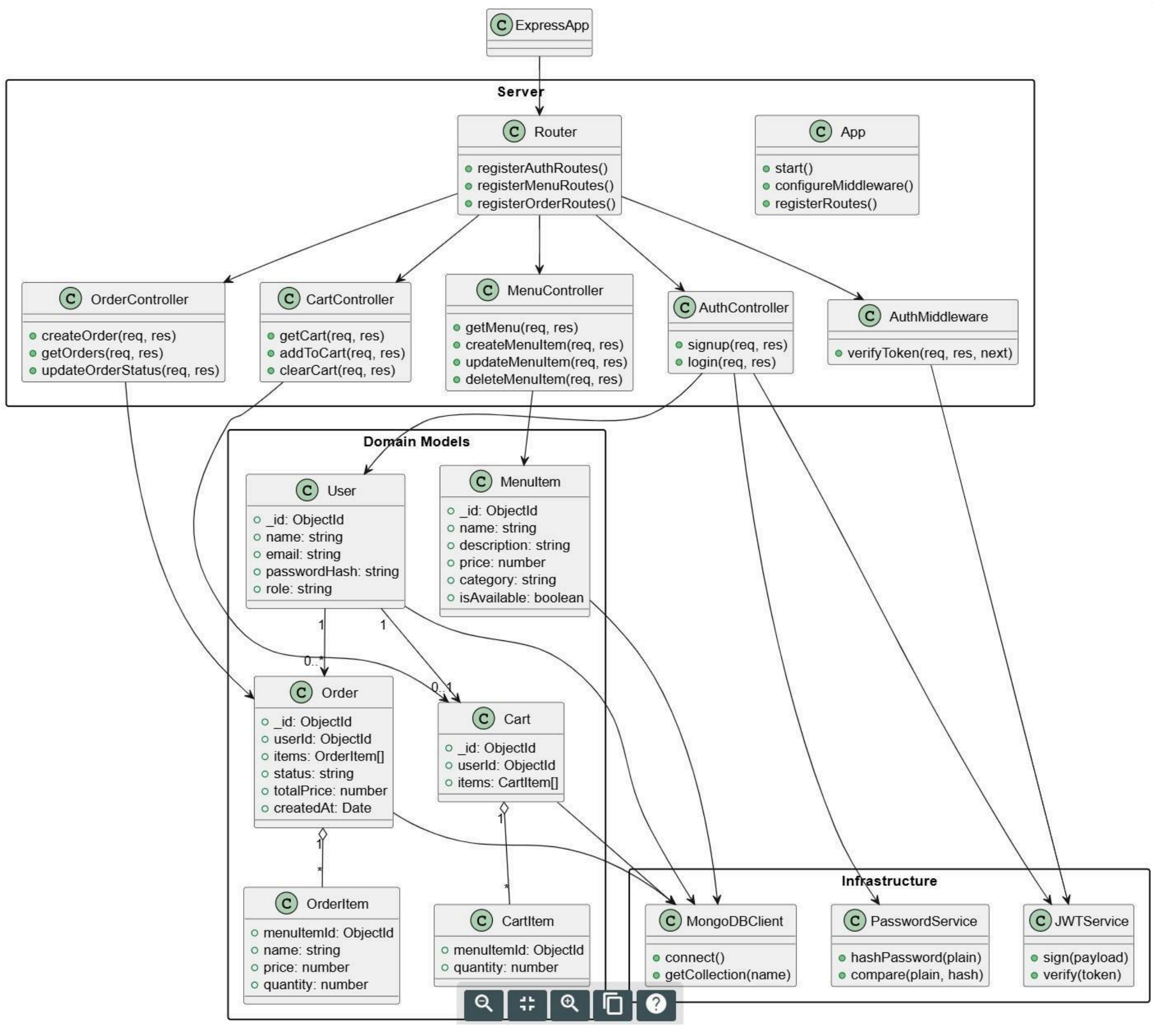
- **Cart Controller:** loads, updates, and clears carts.
- **Menu Controller:** lists, gets, creates, updates, and deletes menu items.
- **Order Controller:** creates orders, updates statuses, gets filtered order lists, and assigns drivers.

**MongoDB Models**

- **MenuItem** — holds name, description, price, category, availability, etc.
- **Cart** — tied to a customer and stores selected items.
- **Order** — stores items, status, timestamps, driver info, etc.

**Utilities**

- **connectDB()** for MongoDB connection.
- **decrypt()** for session cookies.

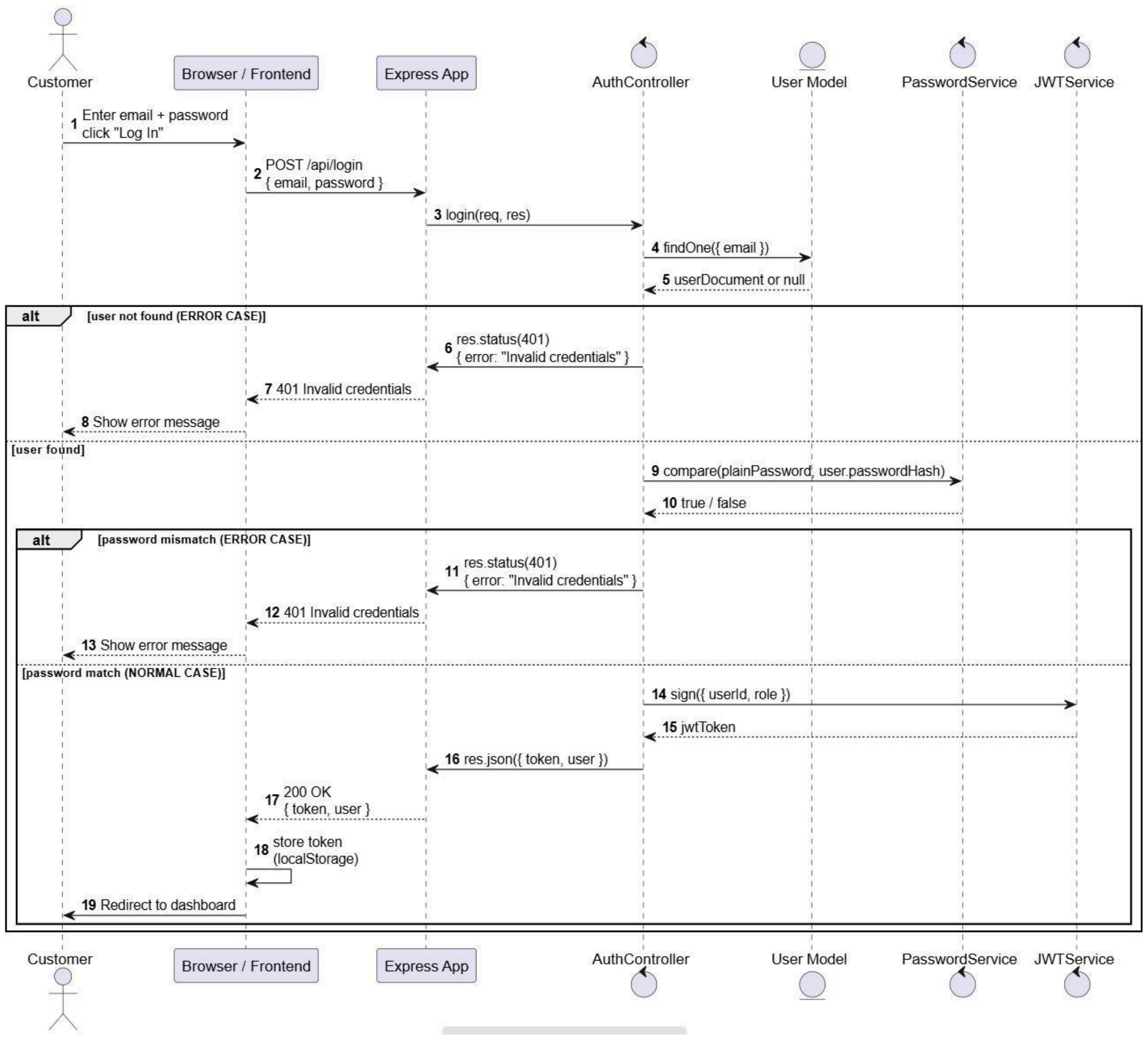


## 2.3 Sequence Diagram (Login Flow)

The login process works like this:

1. Customer enters email and password.
2. Browser sends request to /api/login.
3. Backend checks if the email exists.
4. If not, it returns an error.
5. If it exists, it checks the password.
6. Wrong password returns an error.

7. Correct password generates JWT.
8. The browser stores a token and the user is redirected.



Customer
Browser / Frontend
Express App
AuthController
User Model
PasswordService
JWTService
1 Enter email + password click "Log In"
2 POST /api/login { email, password }
3 login(req, res)
4 findOne({ email })
5 userDocument or null
alt [user not found (ERROR CASE)]
6 res.status(401) { error: "Invalid credentials" }
7 401 Invalid credentials
8 Show error message
[user found]
9 compare(plainPassword, user.passwordHash)
10 true / false
alt [password mismatch (ERROR CASE)]
11 res.status(401) { error: "Invalid credentials" }
12 401 Invalid credentials
13 Show error message
[password match (NORMAL CASE)]
14 sign({ userId, role })
15 jwtToken
16 res.json({ token, user })
17 200 OK { token, user }
18 store token (localStorage)
19 Redirect to dashboard

## 3. User Manual

### 3.1 Introduction

FastByte lets customers browse menu items and order food, while restaurant staff manage and track those orders. This manual explains how to use both sides of the system.

### 3.2 Accessing the Application

1. Start the application (instructions in Section 4).
2. Visit **http://localhost:3000/**.
3. You will land on the Customer Portal.
4. Staff can access **/restaurant** for the dashboard.

### 3.3 Customer Portal Instructions

#### 1. Browsing the Menu

- The menu is pulled from the backend.
- Items are grouped by categories.

#### 2. Searching / Filtering

- Use the search bar or category filters.

#### 3. Adding Items to Cart

- Click “Add to Cart”.
- The cart is saved using localStorage.

#### 4. Viewing / Editing Cart

- Open the cart page.
- Increase/decrease quantity or remove items.
- Totals update instantly.

**5. Checkout & Tracking**

- After checkout, the Track Order screen appears.
- It shows the stage of the order (visual only for now).

### 3.4 Restaurant Portal Instructions

**1. Open the Dashboard**

Navigate to /restaurant.

**2. Filter Orders**

Tabs include:

- All Orders
- Kitchen Queue
- Delivery
- Dine-In

**3. Managing Orders**

- Click into an order to see its details.
- Change order status using the buttons.
- Staff may delete orders to clean up the queue.

### 3.5 Logout Behavior

Logout is not fully implemented.
Users may clear the token from localStorage if needed.

### 3.6 Current Limitations

- Authentication is not enforced everywhere.

- Track Order is mostly visual.
- Dashboard does not auto-refresh.
- No payment integration.

## 4. Compilation and Run Instructions

### Prerequisites

- Install Node.js (version 18 or higher).
- Have a MongoDB connection string (Atlas or local).
- Install Git on your computer.

### Setup

- Clone the project:
    - git clone https://github.com/neelp3145/ITIS-3300
    - cd fastbyte
- Create an .env.local file in the project root.
- Add these environment variables:
    - MONGODB_URI=<your-mongo-uri>
    - JWT_SECRET=<your-secret>
- Install project dependencies:
    - npm install

### Running the Application

- Start the development server:
    - npm run dev
- Open your browser and go to: http://localhost:3000/

## 4.4 Testing

We used Postman to test backend routes such as:

- /api/ping
- /api/signup
- /api/login
- /api/menuItem
- /api/cart

We also relied on manual UI testing.

## 5. Test Cases

Below is the summarized test case table, rewritten in a simple student style:

| # | Functionality | Input / Scenario | Expected Result | Level |
|---|---|---|---|---|
| 1 | Root Health Check | GET / | “FastByte API is running” | API |
| 2 | Ping | GET /api/ping | { ok: true, msg: "pong" } | API |
| 3 | Signup Success | Valid body | 201 Created, user stored | API |
| 4 | Signup Duplicate | Email already used | Error returned | API |
| 5 | Login Success | Correct credentials | Token returned | API |
| 6 | Login Wrong Password | Wrong password | 401 error | API |
| 7 | List Menu Items | GET /api/menuItem | List of items | API |
| 8 | Add to Cart | Add an item | Cart updates | System |
| 9 | Update Cart Quantities | Change quantity | Totals update | System |
| 10 | Missing Header | GET /api/cart without header | 401 error | API |
| 11 | Restaurant View Loads | Visit /restaurant | Orders load | System |
| 12 | Update Order Status | Staff clicks status button | Status updates | System |

## 6. Implemented Features, Limitations & Future Plans

### 6.1 Implemented Features

**Customer Side**

- Menu categories
- Filtering and search
- Working cart with quantity controls
- Cart persistence (localStorage)
- Track Order visual page

**Restaurant Side**

- Dashboard for managing orders
- Tabs for filtering
- Status updating system
- Ability to delete orders

**Backend**

- Next.js API routes
- MongoDB integration
- Signup/login with hashing
- Basic validation and error responses

### 6.2 Limitations

- Authentication isn’t enforced system-wide
- No payment processing
- No real-time updates

- Dashboard uses mock/partial data
- Limited automated tests

### 6.3 Future Work

- Full role-based authentication
- Stripe payments
- WebSockets for real-time updates
- Admin management tools
- Customer order history and analytics for restaurant

## 7. Team Reflection

Throughout all three phases, we learned a lot about how to build a real full-stack web application. The project started small, but each phase added more structure and responsibilities. By the end, we built a functional system that handles both customer ordering and restaurant management.

**What we did well**

- Divided work evenly
- Used Postman to keep backend stable
- Communicated and avoided merge conflicts

**What could be improved**

- Start authentication earlier
- Plan real-time features ahead of time
- Add automated tests instead of relying mainly on manual testing

Overall, FastByte helped us practice teamwork, backend logic, frontend integration, and project planning in a realistic way.

## 8. Team Member Contributions

| Member | Contribution | % |
|---|---|---|
| **Neel Panajkar** | Project management, backend logic (menu, orders, cart), MongoDB setup, UML updates | 25% |
| **Aayush Niroula** | Backend controllers/routes, environment setup, final demo support | 25% |
| **Roger Lin** | Customer portal, cart UI, frontend styling, and testing | 25% |
| **Kaksh Patel** | Requirements, user manual, slides, dashboard layout support | 25% |

-