🍰 BakeBuddy — Online Bakery Ordering Platform (MERN)

A modern full-stack bakery ordering app where customers browse desserts, add items to cart, place orders, and track status — while the admin manages products & orders. The project includes a secure user login system, enabling customers to save their order history and preferences, and enabling admins to access management features.

Status: 🚧 In Development
Tech: **MongoDB + Express + React + Node (MERN)** | Auth | Admin Panel | Cart | Order Tracking

---

🎯 Features

👨‍🍳 Customer Portal

- Browse bakery menu (cakes, pastries, buns, cookies)
- View product details (price, calories, ingredients)
- Add items to cart
- Place order & track status
- View order history
- View personal order history after login

🛠️ Admin Dashboard

- Login as admin
- Add / edit / delete bakery products
- Upload images
- Manage orders (Placed → Preparing → Ready → Delivered)
- View customer list
- Update order status

💻 Objectives

- Deliver production-grade backend using Node.js + Express
- Implement MongoDB database with secure schema
- Provide role based (Customer / Admin) authentication & authorization using JWT
- Include ordering system with tracking
- Create minimal, clean React UI for customer & admin

📽️ Project Scope

In-Scope:

- User Authentication (Customer + Admin)
- Product catalog (CRUD)
- Cart & Checkout
- Order history & status updates
- Admin management dashboard

- User Registration & Login system to allow customers to save order history and preferences

Out of Scope:

- Real online payments (may add later)
- Third-party delivery systems

🖼️ UI Mockups (Concept Design)

Real screenshots will replace these once built

✅ **Customer Menu Page**

✅ **Product Detail + Add to Cart**

✅ **Cart + Checkout Page**

✅ **Admin Dashboard (Products + Orders)**

🧠 System Architecture

```
React (Frontend)
↓ Axios
Express API (Node.js)
↓ Mongoose
MongoDB Atlas
```

🗃️ Database Models

Model	Fields
User	name, email, passwordHash, **role: "customer" | "admin"**, address {street, city, state, zip}, createdAt
Product	name, price, category, calories, description, imageUrl, inStock, createdAt, updatedAt
Cart	userId, items [{productId, qty, priceAtAdd}], updatedAt
Order	userId, items [{productId, qty, unitPrice, subtotal}], total, address, **status: Placed → Preparing → Ready → Delivered**, paymentStatus, createdAt, updatedAt

✅ Detailed JSON-style Model Format

User Model

Handles both customer & admin accounts using role field

```
User {
  name: String
  email: String (unique)
  passwordHash: String
  role: "customer" | "admin"    // default: "customer"

  address: {
    street: String
    city: String
    state: String
    zip: String
  }

  createdAt: Date
}
```

Product Model

```
Product {
  name: String (required)
  price: Number (required)
  category: String
  calories: Number
  description: String
  imageUrl: String
  inStock: Boolean (default: true)

  createdAt: Date
  updatedAt: Date
}
```

Cart Model

```
Cart {
  userId: ObjectId(User)
  items: [
    {
      productId: ObjectId(Product)
      qty: Number (min 1)
      priceAtAdd: Number
    }
  ]

  updatedAt: Date
}
```

Order Model

```
Order {
  userId: ObjectId(User)
  items: [
    {
      productId: ObjectId(Product)
      qty: Number
      unitPrice: Number
      subtotal: Number
    }
  ]

  total: Number
  address: String

  status: "Placed" | "Preparing" | "Ready" | "Out for Delivery" | "Delivered" |
"Cancelled"
  paymentStatus: "Pending" | "Paid" | "Refunded"

  createdAt: Date
  updatedAt: Date
}
```

✅ Key Notes (paste this too)

- Only **one User model** is used — admin and customer are identified via role
- Improves scalability & security (#industry practice)
- Makes authentication & session management easier
- Supports multiple admins if needed later

🛠️ Tech Stack

Category	Tech
Frontend	React, Tailwind/Bootstrap, Axios
Backend	Node.js, Express
Database	MongoDB Atlas, Mongoose
Auth	JWT, bcrypt
Deployment	Vercel (client) + Render (server)
Tools	GitHub, VsCode,

🚀 API Endpoints

Auth

- POST /api/auth/register → Create Account (Customer / Admin)
- POST /api/auth/login → Generate Login Token (Role Based)

Products

- GET /api/products (*customer*)
- POST /api/products *(admin)*
- PUT /api/products/:id *(admin)*
- DELETE /api/products/:id *(admin)*

Cart

- GET /api/cart
- POST /api/cart
- PUT /api/cart/: itemId
- DELETE /api/cart/: itemId

Orders

- POST /api/orders
- GET /api/orders/me
- GET /api/orders *(admin)*
- PATCH /api/orders/:id/status *(admin)*

NOTE - Since both customers and admins share many attributes, the system uses a single User model with a role field, instead of multiple tables. This improves scalability and follows modern MERN-stack architecture practices.

⚙️ Installation & Run

Backend

```
cd server
npm install
npm run dev
```

Frontend

```
cd client
npm install
npm start
```

🧪 Testing

- Postman collection included *(coming soon)*
- Frontend UI test flow

"How does the backend magically know if this user is admin or customer? Am I supposed to type admin somewhere?"

Answer:

✅ Yes — we store a field called role in the user in database.

That's how backend knows.

There is **NO magic.**

You tell the system once when creating the user.

📦 When someone registers

Regular customer signup 👉 role = "customer"

This happens **automatically** during normal registration.

Example incoming signup data

```
{
  "name": "Yash",
  "email": "yash@example.com",
  "password": "123456"
}
```

Backend assigns role: "customer" by default:

```
role: "customer"
```

So, this gets stored in MongoDB:

```
{
  "name": "Yash",
  "email": "yash@example.com",
  "role": "customer"
}
```

👑 But what about the Admin?

Admins are **not created by normal signup.**

2 ways to make admin:

✅ Method 1: **Manually in database (most common for projects)**

You insert a record in MongoDB like:

```
{
  "name": "Bakery Owner",
  "email": "admin@bakery.com",
  "passwordHash": "xxxxx",
  "role": "admin"
}
```

or update an existing user role in MongoDB Compass/Atlas:

role: "customer" → role: "admin"

✅ Method 2: **Admin Creation via backend route** (Optional)

You (developer) call a special route once:

Example:

```
POST /api/auth/create-admin
```

With body:

```
{"email": "owner@bakery.com", "password": "admin123"}
```

And backend sets:

```
role: "admin"
```

✅ Method 3: Simple default logic for class project

We assign admin in code:

```
if (email === "admin@bakery.com") {
   role = "admin";
}
else {
   role = "customer";
}
```

So, people who login using this email become admin.

🧠 Summary

Who decides the role?	How
Customer	Automatically on signup
Admin	Manually set in DB OR by specific admin route OR by predefined email rule

---

👇 In simple words:

When someone signs up normally → they become **customer** automatically.

You create admin user separately and set their role to **admin**.

Backend reads that role from database every time user logs in.

📅 Timeline

Week	Milestone
1	Database + Auth + Product APIs
2	Cart + Orders + Admin APIs
3	Frontend UI + Test + Deployment

💡 Future Enhancements

- Stripe payments
- Real-time notifications
- Delivery tracking map
- Discounts / Coupons

👨‍💻 Author

Javeed Quadri Mohammad

Email: jmohamma@aum.edu

Student ID: S00459899