# Single-File Backend for Food Delivery App

```
const express = require('express');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const cors = require('cors');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

dotenv.config();
const app = express();
app.use(express.json());
app.use(cors());

mongoose.connect(process.env.MONGO_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => console.log("MongoDB Connected"))
.catch(err => console.error("MongoDB Connection Error:", err));

const UserSchema = new mongoose.Schema({
    name: String,
    email: { type: String, unique: true },
    password: String,
    role: { type: String, enum: ['customer', 'restaurant'], default: 'customer' }
});
const User = mongoose.model('User', UserSchema);

const RestaurantSchema = new mongoose.Schema({
    name: String,
    address: String,
    owner: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }
});
const Restaurant = mongoose.model('Restaurant', RestaurantSchema);

const OrderSchema = new mongoose.Schema({
    customer: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
    restaurant: { type: mongoose.Schema.Types.ObjectId, ref: 'Restaurant' },
    items: [{ name: String, quantity: Number }],
    status: { type: String, enum: ['pending', 'completed'], default: 'pending' }
});
const Order = mongoose.model('Order', OrderSchema);

app.post('/api/register', async (req, res) => {
    try {
        const { name, email, password, role } = req.body;
        const hashedPassword = await bcrypt.hash(password, 10);
        const user = new User({ name, email, password: hashedPassword, role });
        await user.save();
        res.status(201).json({ message: "User registered successfully" });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
});

app.post('/api/login', async (req, res) => {
    try {
        const { email, password } = req.body;
        const user = await User.findOne({ email });
        if (!user) return res.status(404).json({ message: "User not found" });

        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) return res.status(400).json({ message: "Invalid credentials" });
```

```
        const token = jwt.sign({ id: user._id, role: user.role }, process.env.JWT_SECRET, { expiresIn: '1d' });
        res.json({ token, user });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
});

const authMiddleware = (req, res, next) => {
    const token = req.headers.authorization;
    if (!token) return res.status(401).json({ message: "Access Denied" });

    try {
        const decoded = jwt.verify(token, process.env.JWT_SECRET);
        req.user = decoded;
        next();
    } catch (error) {
        res.status(400).json({ message: "Invalid Token" });
    }
};

app.post('/api/restaurants', authMiddleware, async (req, res) => {
    if (req.user.role !== 'restaurant') return res.status(403).json({ message: "Unauthorized" });

    const { name, address } = req.body;
    const restaurant = new Restaurant({ name, address, owner: req.user.id });
    await restaurant.save();
    res.status(201).json(restaurant);
});

app.get('/api/restaurants', async (req, res) => {
    const restaurants = await Restaurant.find();
    res.json(restaurants);
});

app.post('/api/orders', authMiddleware, async (req, res) => {
    if (req.user.role !== 'customer') return res.status(403).json({ message: "Unauthorized" });

    const { restaurantId, items } = req.body;
    const order = new Order({ customer: req.user.id, restaurant: restaurantId, items });
    await order.save();
    res.status(201).json(order);
});

app.get('/api/orders', authMiddleware, async (req, res) => {
    const orders = await Order.find({ customer: req.user.id }).populate('restaurant');
    res.json(orders);
});

app.put('/api/orders/:orderId', authMiddleware, async (req, res) => {
    if (req.user.role !== 'restaurant') return res.status(403).json({ message: "Unauthorized" });

    const { status } = req.body;
    await Order.findByIdAndUpdate(req.params.orderId, { status });
    res.json({ message: "Order status updated" });
});

app.get("/", (req, res) => res.send("Food Delivery API Running"));

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```