College Canteen Order System — Backend Code Package

This PDF contains the backend code (Node.js + Express + SQLite) for the College Canteen Order System MVP. Copy files into a backend/ folder and follow the instructions to run locally.

1) package.json

```
{
  "name": "canteen-backend",
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "bcrypt": "^5.1.0",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.0",
    "sqlite3": "^5.1.6",
    "body-parser": "^1.20.2"
  },
  "devDependencies": {
    "nodemon": "^2.0.22"
  }
}
```

2) .env (example)

```
PORT=5000
JWT_SECRET=your_secret_key_here
DB_FILE=./canteen.db
```

3) models/db.js

```
import sqlite3 from "sqlite3";
import { open } from "sqlite";
import dotenv from "dotenv";
dotenv.config();

const dbFile = process.env.DB_FILE || "./canteen.db";

export async function openDB() {
  const db = await open({
    filename: dbFile,
    driver: sqlite3.Database
  });
  return db;
}
```

4) sql/init.sql

```
PRAGMA foreign_keys = ON;

CREATE TABLE IF NOT EXISTS users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT,
  email TEXT UNIQUE,
  phone TEXT,
  role TEXT DEFAULT 'student',
  password TEXT
);

CREATE TABLE IF NOT EXISTS categories (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT
);

CREATE TABLE IF NOT EXISTS items (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT,
  category_id INTEGER,
  price INTEGER,
  description TEXT,
  image TEXT,
  available INTEGER DEFAULT 1,
  FOREIGN KEY (category_id) REFERENCES categories(id)
);

CREATE TABLE IF NOT EXISTS orders (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  user_id INTEGER,
  total_amount INTEGER,
  status TEXT DEFAULT 'placed',
  created_at TEXT DEFAULT (datetime('now')),
  pickup_time TEXT,
  payment_id INTEGER,
  FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE TABLE IF NOT EXISTS order_items (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  order_id INTEGER,
  item_id INTEGER,
  qty INTEGER,
  price INTEGER,
  FOREIGN KEY (order_id) REFERENCES orders(id),
  FOREIGN KEY (item_id) REFERENCES items(id)
);

CREATE TABLE IF NOT EXISTS payments (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  order_id INTEGER,
  method TEXT,
  amount INTEGER,
  status TEXT DEFAULT 'pending',
  tx_ref TEXT,
  FOREIGN KEY (order_id) REFERENCES orders(id)
);

CREATE TABLE IF NOT EXISTS inventory (
  item_id INTEGER PRIMARY KEY,
  stock_qty INTEGER,
  FOREIGN KEY (item_id) REFERENCES items(id)
);
```

5) server.js

```
import express from "express";
import cors from "cors";
import bodyParser from "body-parser";
import dotenv from "dotenv";
dotenv.config();

import { openDB } from "./models/db.js";
import authRoutes from "./routes/auth.js";
import itemsRoutes from "./routes/items.js";
import ordersRoutes from "./routes/orders.js";
import paymentsRoutes from "./routes/payments.js";

const app = express();
app.use(cors());
app.use(bodyParser.json());

// attach db to req context
app.use(async (req, res, next) => {
  req.db = await openDB();
  next();
});

app.use("/auth", authRoutes);
app.use("/menu", itemsRoutes);
app.use("/orders", ordersRoutes);
app.use("/payments", paymentsRoutes);

app.get("/", (req, res) => res.json({ msg: "Canteen API Running" }));

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

6) routes/auth.js

```
import express from "express";
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
const router = express.Router();

const JWT_SECRET = process.env.JWT_SECRET || "secret";

router.post("/register", async (req, res) => {
  const db = req.db;
  const { name, email, phone, password, role } = req.body;
  if (!email || !password) return res.status(400).json({ error: "email & password required" });
  try {
    const hash = await bcrypt.hash(password, 10);
    const result = await db.run(
      `INSERT INTO users (name,email,phone,role,password) VALUES(?,?,?,?,?)`,
      [name, email, phone, role || "student", hash]
    );
    const user = await db.get("SELECT id,name,email,phone,role FROM users WHERE id = ?", [result.lastID]
    res.json({ user });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

router.post("/login", async (req, res) => {
  const db = req.db;
  const { email, password } = req.body;
  if (!email || !password) return res.status(400).json({ error: "email & password required" });
  try {
    const user = await db.get("SELECT * FROM users WHERE email = ?", [email]);
    if (!user) return res.status(401).json({ error: "Invalid credentials" });
    const ok = await bcrypt.compare(password, user.password);
    if (!ok) return res.status(401).json({ error: "Invalid credentials" });
    const token = jwt.sign({ id: user.id, role: user.role }, JWT_SECRET, { expiresIn: "8h" });
    res.json({ token, user: { id: user.id, name: user.name, email: user.email, role: user.role }});
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

export default router;
```

7) routes/items.js

```
import express from "express";
const router = express.Router();

// GET /menu  => categories + items
router.get("/", async (req, res) => {
  const db = req.db;
  try {
    // return items with category name
    const rows = await db.all(`
      SELECT i.*, c.name as category_name
      FROM items i LEFT JOIN categories c ON i.category_id = c.id
      WHERE i.available = 1
    `);
    // group by category (optional)
    res.json(rows);
  } catch (err) { res.status(500).json({ error: err.message }); }
});

// GET /menu/:id
router.get("/:id", async (req, res) => {
  const db = req.db;
  try {
    const row = await db.get("SELECT * FROM items WHERE id = ?", [req.params.id]);
    res.json(row);
  } catch (err) { res.status(500).json({ error: err.message }); }
});

export default router;
```

8) routes/orders.js

```
import express from "express";
const router = express.Router();

// POST /orders/checkout
// expected body: { user_id, items: [{item_id, qty}], pickup_time, payment_method }
router.post("/checkout", async (req, res) => {
  const db = req.db;
  try {
    const { user_id, items, pickup_time, payment_method } = req.body;
    if (!user_id || !items || !items.length) return res.status(400).json({ error: "invalid payload" });

    // compute total
    let total = 0;
    for (const it of items) {
      const row = await db.get("SELECT price FROM items WHERE id = ?", [it.item_id]);
      if (!row) return res.status(400).json({ error: `item ${it.item_id} not found` });
      total += row.price * (it.qty || 1);
    }

    // create order
    const result = await db.run("INSERT INTO orders (user_id,total_amount,pickup_time,status) VALUES (?,
      [user_id, total, pickup_time || null, "placed"]);
    const orderId = result.lastID;

    // insert order_items
    for (const it of items) {
      const row = await db.get("SELECT price FROM items WHERE id = ?", [it.item_id]);
      await db.run("INSERT INTO order_items (order_id,item_id,qty,price) VALUES (?,?,?,?)",
        [orderId, it.item_id, it.qty || 1, row.price]);
    }

    // create payment record (pending)
    const pay = await db.run("INSERT INTO payments (order_id,method,amount,status) VALUES (?,?,?,?)",
      [orderId, payment_method || "pay_on_pickup", total, payment_method ? "paid" : "pending"]);

    res.json({ orderId, total });
  } catch (err) { res.status(500).json({ error: err.message }); }
});

// GET /orders/:userId
router.get("/:userId", async (req, res) => {
  const db = req.db;
  try {
    const orders = await db.all("SELECT * FROM orders WHERE user_id = ? ORDER BY created_at DESC", [req.
    res.json(orders);
  } catch (err) { res.status(500).json({ error: err.message }); }
});

// PATCH /orders/:id/status
router.patch("/:id/status", async (req, res) => {
  const db = req.db;
  const { status } = req.body;
  try {
    await db.run("UPDATE orders SET status = ? WHERE id = ?", [status, req.params.id]);
    res.json({ success: true });
  } catch (err) { res.status(500).json({ error: err.message }); }
});

export default router;
```

9) routes/payments.js

```
import express from "express";
const router = express.Router();

router.post("/webhook", async (req, res) => {
  const db = req.db;
  // example: { order_id, status, tx_ref }
  const { order_id, status, tx_ref } = req.body;
  try {
    await db.run("UPDATE payments SET status = ?, tx_ref = ? WHERE order_id = ?", [status, tx_ref, order
    if (status === "paid") {
      await db.run("UPDATE orders SET status = ? WHERE id = ?", ["paid", order_id]);
    }
    res.json({ ok: true });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

export default router;
```

10) README & Run Instructions

```
How to initialize and run locally:

1. Create project folder and files exactly as shown in this PDF, or download the code files into a folde

2. Install dependencies:
   npm install

3. Initialize the SQLite database:
   - If you have sqlite3 CLI: sqlite3 canteen.db < sql/init.sql
   - Or run a small node script that opens the DB and executes the init.sql file (example below).

4. Create a .env file in project root with:
   PORT=5000
   JWT_SECRET=your_secret_key_here
   DB_FILE=./canteen.db

5. Start server:
   node server.js
   (Or for development with auto-restart: npm run dev)

6. Test API endpoints with Postman or curl:
   - GET http://localhost:5000/menu
   - POST http://localhost:5000/auth/register  (body: JSON { name, email, password })
   - POST http://localhost:5000/auth/login     (body: JSON { email, password })
   - POST http://localhost:5000/orders/checkout (body: JSON { user_id, items: [{item_id, qty}], pickup_t

Optional: Add seed data by inserting categories and items into the items/categories tables using sqlite3

Security note: This is a starter project. For production, add input validation, secure environment varia
```