EGOTECHWORLD PVT LTD

BAKERY

Management System

A Step-by-Step Beginner's Tutorial

Build a complete bakery system with billing, stock control, online ordering, user login, and an admin dashboard.

HTML 5	Bootstrap 5	Node.js	Express.js	SQLite

Tutorial Level:	Beginner
Estimated Time:	4 - 6 hours
Project Type:	Full-Stack Web Application
Database:	SQLite (file-based, no setup)
Published by:	EGOTECHWORLD - egotechworld.com

Table of Contents

TIP: How to read this tutorial

Follow each step in order. Every step shows you the exact code to type, what it does, and a preview of the screen. Do not skip ahead - each step builds on the previous one. If something does not work, re-read the step carefully and check for typos.

1. Introduction & What You Will Build

Welcome to this beginner-friendly tutorial from **EGOTECHWORLD**. By the end of this guide, you will have built a complete, working **Bakery Management System** that runs on your own computer. The system handles everything a small bakery needs: customers can browse products and place orders online, the shop owner can generate bills, track stock, and manage everything from a clean admin dashboard.

What you will learn

- How to set up a Node.js project from scratch.
- How to design and create a SQLite database.
- How to build pages with HTML and Bootstrap 5.
- How to write Express routes for login, orders, and billing.
- How to protect admin pages with sessions.
- How to test your software properly.

Features included

Module	What it does
User Login	Customers and admin can register and sign in.
Online Ordering	Customers add products to a cart and checkout.
Billing	Generate a printable bill for each order.
Stock	Auto-reduce stock when orders are placed.
Admin Panel	View orders, manage products, see total sales.

NOTE: Who this tutorial is for

This tutorial is written for absolute beginners. You only need basic HTML and a willingness to follow instructions. We will explain every step. If you can copy and paste code into a file, you can build this.

2. System Flow Chart

Before we write any code, let's understand how the bakery system works as a whole. The flow chart below shows the journey a user takes - from opening the website to data being saved in the database.

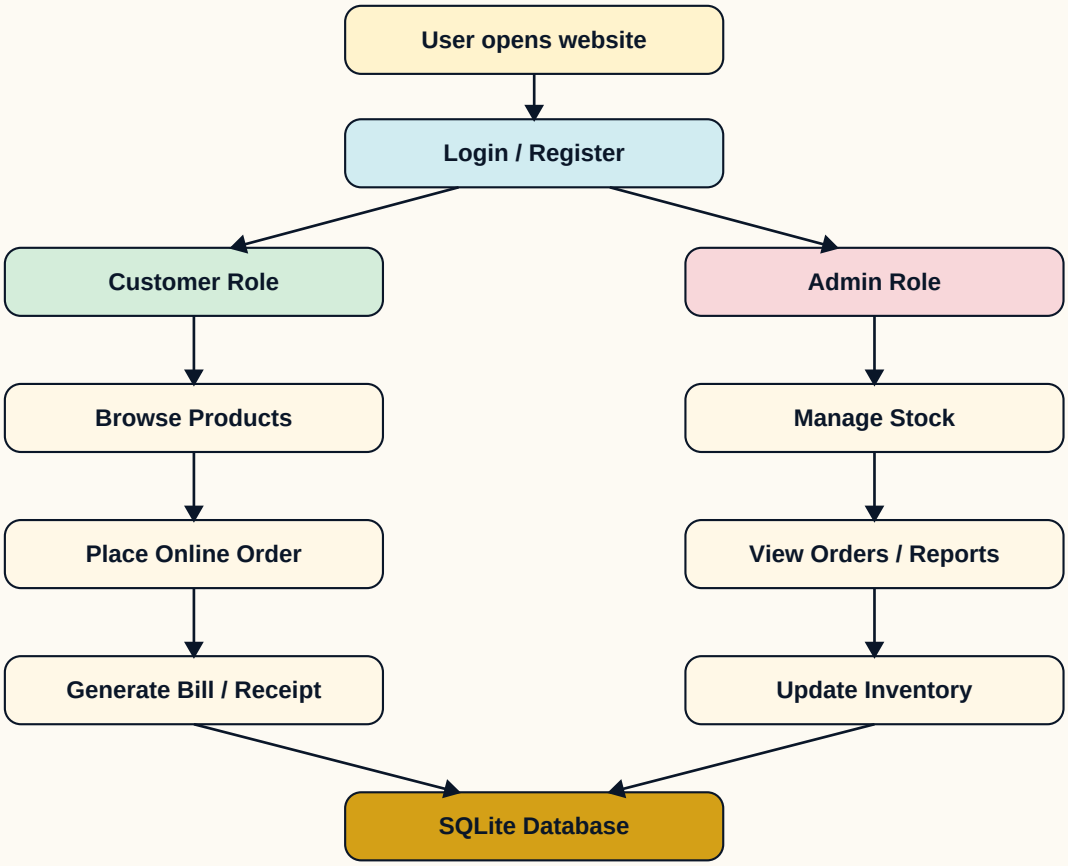

Figure 2.1 - High level flow of the Bakery Management System

Reading the flow chart

- Every visitor lands on the home page first.
- They must log in - the system checks their role.
- Customers can browse products and place online orders.
- Admins can manage stock and view all orders/sales.
- All data ends up in one SQLite database file.

3. Software Documentation

3.1 System Overview

The Bakery Management System is a full-stack web application. The **front end** is built with HTML and Bootstrap 5 - this is what users see. The **back end** is Node.js with the Express framework - this handles requests and talks to the database. The **database** is SQLite, a single-file database that needs no server installation.

3.2 Architecture

Layer	Technology	Responsibility
Presentation	HTML, Bootstrap 5, CSS	Display pages, forms, buttons
Application	Node.js + Express	Routing, business logic, sessions
Data	SQLite (better-sqlite3)	Persist users, products, orders

3.3 Folder Structure

FOLDERS

```
bakery-app/
  |-- node_modules/          (created by npm)
  |-- public/                (CSS, JS, images for browser)
  |   |-- css/style.css
  |   |-- images/
  |-- views/                 (HTML pages)
  |   |-- index.html
  |   |-- login.html
  |   |-- register.html
  |   |-- products.html
  |   |-- cart.html
  |   |-- bill.html
  |   |-- admin.html
  |-- bakery.db              (SQLite database file)
  |-- server.js              (Express backend)
  |-- package.json           (project metadata)
```

Figure 3.1 - Project folder layout

3.4 User Roles

Role	Permissions
Customer	Browse products, place orders, view own bills.
Admin	Add/edit products, manage stock, view all orders, see totals.

3.5 Database Tables

The system uses three simple tables:

Table	Columns	Purpose
users	id, name, email, password, role	Stores customers & admins
products	id, name, price, stock, image	Bakery items for sale
orders	id, user_id, items, total, date, status	Each order placed

3.6 Main API Routes

Method	Path	What it does
GET	/	Home page
GET / POST	/login	Show login page / process login
GET / POST	/register	Sign up new customer
GET	/products	List bakery products
POST	/cart/add	Add item to cart
GET / POST	/checkout	Place an online order
GET	/bill/:id	Printable receipt for an order
GET	/admin	Admin dashboard (protected)
POST	/admin/stock	Update product stock

4. Installation Instructions

Before writing any code, you need three things installed on your computer. Each one is free and easy to set up.

4.1 Install Node.js

- Go to **https://nodejs.org** in your browser.
- Download the **LTS version** (the green button on the left).
- Run the installer, click Next on every screen, then Finish.
- Open **Command Prompt** (Windows) or **Terminal** (Mac/Linux).
- Type **node -v** and press Enter. You should see a version like *v20.10.0*.

> **TIP: Why Node.js?**
>
> Node.js lets you run JavaScript on your computer (not just inside a browser). It is the foundation of our backend server.

4.2 Install a Code Editor

We recommend **Visual Studio Code** - it is free and easy. Download from **https://code.visualstudio.com** and install it. It works on Windows, Mac, and Linux.

4.3 Verify everything works

BASH

```bash
# Check Node.js version
node -v

# Check npm (comes with Node.js)
npm -v

# Both commands should print a version number
```

> **DONE: Pre-flight checklist**
>
> You are ready to continue if: (1) Node.js shows a version, (2) npm shows a version, (3) you have VS Code installed. If any of these fail, re-run the installer before moving on.

4.4 What npm packages we will use

Package	What it does
`express`	The web server framework.
`better-sqlite3`	Talks to our SQLite database file.
`bcryptjs`	Encrypts user passwords safely.
`express-session`	Remembers logged-in users.

`body-parser`	Reads form data from POST requests.
`ejs`	Lets us inject data into HTML pages.

Don't worry - we will install all of these in Step 1.

STEP 1 Project Setup

We will create the project folder, initialize npm, and install every package we need. Take it slow - copy each command exactly.

1.1 Create the folder

BASH

```
# Open your terminal and run:
mkdir bakery-app
cd bakery-app
```

What just happened? mkdir creates a folder called bakery-app and cd moves you inside it. Every command from now on runs from this folder.

1.2 Start a new Node project

BASH

```
npm init -y
```

This creates a **package.json** file - a list of your project's details and which packages it uses. The -y flag accepts all the defaults.

1.3 Install the packages

BASH

```
npm install express better-sqlite3 bcryptjs express-session body-parser ejs
```

Wait until the installation finishes. You will see a **node_modules** folder appear - that holds all the packages we just installed.

1.4 Create the file structure

BASH

```
# Inside bakery-app, create these files & folders:
mkdir public views
mkdir public/css public/images
type nul > server.js          # On Windows
# touch server.js             # On Mac/Linux
```

NOTE: Use VS Code instead

It is easier to right-click in the VS Code file explorer and choose 'New File' or 'New Folder' to build the structure shown in section 3.3.

1.5 First server file

Open **server.js** in VS Code and paste this:

JS

```
// server.js - the heart of our bakery app
const express = require('express');
const path = require('path');

const app = express();
const PORT = 3000;

// Serve files from the 'public' folder
app.use(express.static(path.join(__dirname, 'public')));

// Test home route
app.get('/', (req, res) => {
  res.send('<h1>Welcome to the Bakery!</h1>');
});

app.listen(PORT, () => {
  console.log('Bakery server running on http://localhost:' + PORT);
});
```

1.6 Run the server

BASH

```
node server.js
```

Open your browser and go to **http://localhost:3000**. You should see *Welcome to the Bakery!*. Press **Ctrl + C** in the terminal to stop the server when needed.

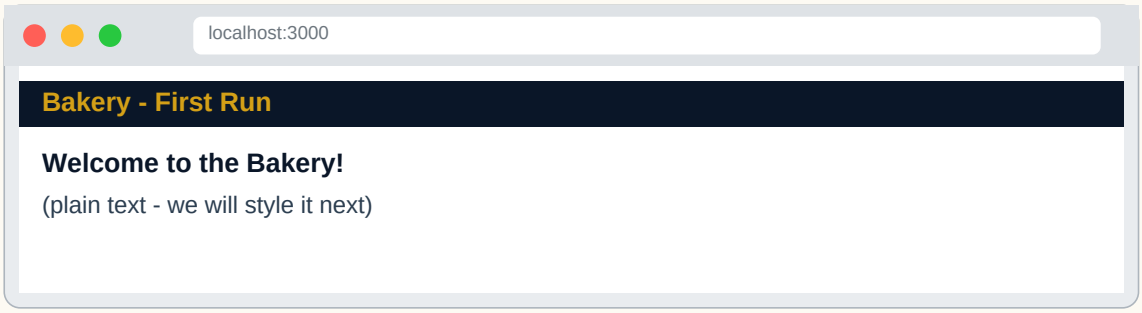

Figure 5.1 - First successful run on localhost:3000

DONE: Step 1 complete

Your project is set up, packages are installed, and you have a working server. You are now ready to design the database.

STEP 2 Database Design (SQLite)

SQLite stores everything in one file - **bakery.db**. We will create that file and add three tables: users, products, and orders.

2.1 Create db.js

Make a new file called **db.js** in the project root and paste:

JS

```
// db.js - sets up SQLite and creates tables
const Database = require('better-sqlite3');
const db = new Database('bakery.db');

// USERS table
db.exec(`
  CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL,
    role TEXT DEFAULT 'customer'
  )
`);

// PRODUCTS table
db.exec(`
  CREATE TABLE IF NOT EXISTS products (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    price REAL NOT NULL,
    stock INTEGER DEFAULT 0,
    image TEXT
  )
`);

// ORDERS table
db.exec(`
  CREATE TABLE IF NOT EXISTS orders (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    items TEXT NOT NULL,
    total REAL NOT NULL,
    status TEXT DEFAULT 'pending',
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
  )
`);

module.exports = db;
```

Reading the SQL: CREATE TABLE IF NOT EXISTS only creates the table the first time. PRIMARY KEY AUTOINCREMENT means SQLite assigns each new row a unique number automatically.

2.2 Add starter products & admin

Create a file **seed.js** to fill the database with sample data:

JS

```
// seed.js - run once to add starter data
const db = require('./db');
const bcrypt = require('bcryptjs');

// Add an admin (password: admin123)
const hash = bcrypt.hashSync('admin123', 10);
db.prepare(
  'INSERT OR IGNORE INTO users (name, email, password, role) VALUES (?, ?, ?, ?)'
).run('Bakery Admin', 'admin@bakery.com', hash, 'admin');

// Add bakery products
const products = [
  { name: 'Chocolate Cake',    price: 1500, stock: 10 },
  { name: 'Vanilla Cupcake',   price: 250,  stock: 40 },
  { name: 'Strawberry Donut',  price: 300,  stock: 25 },
  { name: 'Butter Croissant',  price: 350,  stock: 30 },
  { name: 'Cinnamon Roll',     price: 400,  stock: 20 },
  { name: 'Garlic Bread',      price: 450,  stock: 15 }
];
const insert = db.prepare(
  'INSERT INTO products (name, price, stock) VALUES (?, ?, ?)'
);
products.forEach(p => insert.run(p.name, p.price, p.stock));

console.log('Database seeded successfully!');
```

Run the seed

BASH

```
node seed.js
```

WARNING: Run seed only once

If you run seed.js twice it will add duplicate products. The INSERT OR IGNORE for the admin user is safe, but products will repeat. To start fresh, delete **bakery.db** and run seed.js again.

DONE: Step 2 complete

You have a SQLite database with three tables and sample data. Login: admin@bakery.com / Password: admin123

STEP 3 Express Server & Routes

Now we expand **server.js** to support sessions, EJS templates (so we can inject data into HTML), and form data.

JS

```
// server.js - full version
const express = require('express');
const session = require('express-session');
const bodyParser = require('body-parser');
const path = require('path');

const db = require('./db');
const app = express();
const PORT = 3000;

// View engine - lets us use .ejs files in /views
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Read POST form data and serve static files
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, 'public')));

// Sessions remember who is logged in
app.use(session({
  secret: 'bakery-secret-egotechworld',
  resave: false,
  saveUninitialized: false
}));

// Make the logged-in user available in every view
app.use((req, res, next) => {
  res.locals.user = req.session.user || null;
  next();
});

// Home page
app.get('/', (req, res) => {
  res.render('index');
});

app.listen(PORT, () => {
  console.log('Bakery running on http://localhost:' + PORT);
});
```

TIP: Rename .html to .ejs

EJS files are just HTML files with a different extension. Inside them you can write **<%= variableName %>** to print data from your routes. We will use this on every page.

3.1 Create the home page

Create **views/index.ejs**:

HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Sweet Crust Bakery</title>
  <link rel="stylesheet" href="/css/bootstrap.min.css">
  <link rel="stylesheet" href="/css/style.css">
</head>
<body>

  <!-- Top navigation -->
  <nav class="navbar navbar-expand-lg navbar-dark"
       style="background-color:#5a3a22;">
    <div class="container">
      <a class="navbar-brand fw-bold" href="/">Sweet Crust Bakery</a>
      <div>
        <a href="/products" class="btn btn-warning me-2">Shop</a>
        <% if (user) { %>
          <a href="/logout" class="btn btn-light">Logout</a>
        <% } else { %>
          <a href="/login" class="btn btn-light">Login</a>
        <% } %>
      </div>
    </div>
  </nav>

  <!-- Hero section -->
  <div class="text-center py-5"
       style="background:#fff8e7;">
    <h1 class="display-4 fw-bold" style="color:#5a3a22;">
      Fresh Baked Daily
    </h1>
    <p class="lead">Cakes, cupcakes, donuts &amp;amp; more, made every morning.</p>
    <a href="/products" class="btn btn-lg btn-warning">Order Online</a>
  </div>
</body>
</html>
```

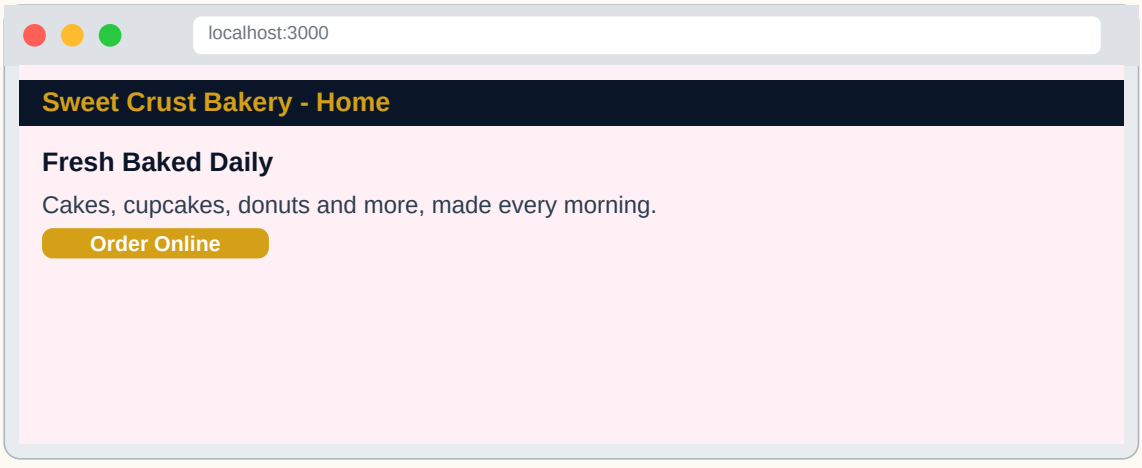

Figure 7.1 - Preview of the home page

DONE: Step 3 complete

Your server now supports HTML pages, sessions, and form data. The home page is live.

STEP 4 User Login & Registration

Users need accounts. We will create routes to register new customers and log existing users in. Passwords are hashed with bcrypt - we never store the original.

4.1 Add auth routes to server.js

JS

```
// At the top of server.js
const bcrypt = require('bcryptjs');

// Register
app.get('/register', (req, res) => res.render('register', { error: null }));
app.post('/register', (req, res) => {
  const { name, email, password } = req.body;
  const hash = bcrypt.hashSync(password, 10);
  try {
    db.prepare(
      'INSERT INTO users (name, email, password) VALUES (?, ?, ?)'
    ).run(name, email, hash);
    res.redirect('/login');
  } catch (e) {
    res.render('register', { error: 'Email already used' });
  }
});

// Login
app.get('/login', (req, res) => res.render('login', { error: null }));
app.post('/login', (req, res) => {
  const { email, password } = req.body;
  const user = db.prepare(
    'SELECT * FROM users WHERE email = ?'
  ).get(email);
  if (user && bcrypt.compareSync(password, user.password)) {
    req.session.user = {
      id: user.id, name: user.name, role: user.role
    };
    return res.redirect(user.role === 'admin' ? '/admin' : '/products');
  }
  res.render('login', { error: 'Wrong email or password' });
});

// Logout
app.get('/logout', (req, res) => {
  req.session.destroy(() => res.redirect('/'));
});
```

4.2 Create views/login.ejs

HTML

```
<!DOCTYPE html>
<html><head>
<title>Login - Sweet Crust</title>
<link rel="stylesheet" href="/css/bootstrap.min.css">
</head>
<body class="bg-light">
  <div class="container py-5" style="max-width:420px;">
    <div class="card shadow-sm">
      <div class="card-body p-4">
        <h3 class="text-center mb-4" style="color:#5a3a22;">Welcome Back</h3>
        <% if (error) { %>
          <div class="alert alert-danger"><%= error %></div>
        <% } %>
        <form method="POST" action="/login">
          <div class="mb-3">
            <label class="form-label">Email</label>
            <input type="email" name="email" class="form-control" required>
          </div>
          <div class="mb-3">
            <label class="form-label">Password</label>
            <input type="password" name="password" class="form-control" required>
          </div>
          <button class="btn btn-warning w-100">Login</button>
          <p class="text-center mt-3">
            New user? <a href="/register">Register here</a>
          </p>
        </form>
      </div>
    </div>
  </div>
</body></html>
```

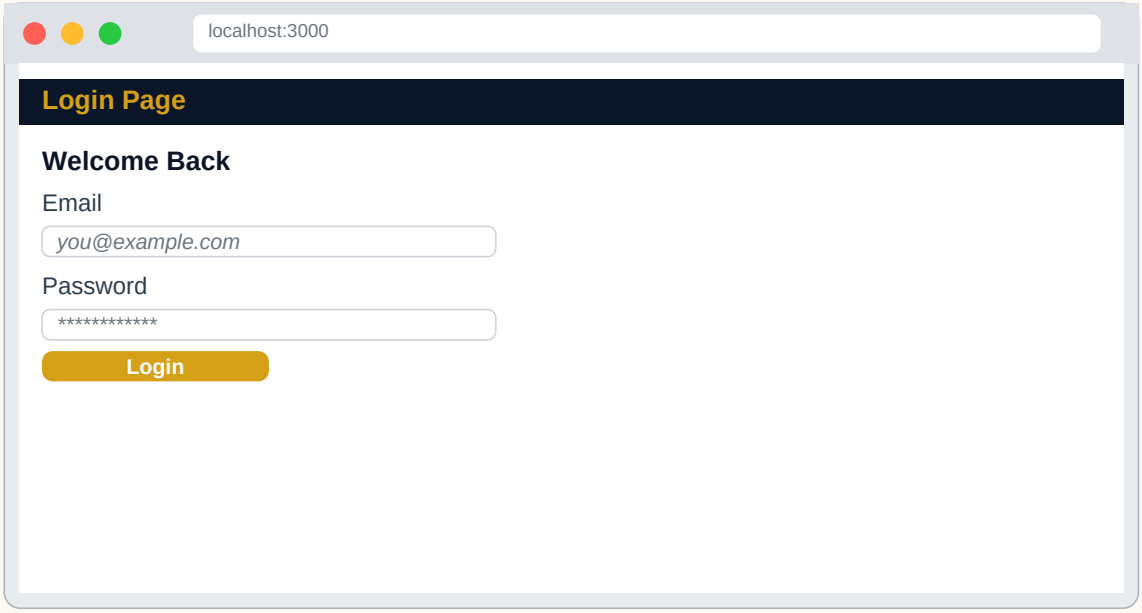

Figure 8.1 - Login page preview

> **NOTE: Build register.ejs the same way**
>
> Copy login.ejs, change the title to 'Create Account', add a 'Name' input, and change the form action to **/register**. That's it.

STEP 5 Customer Front Page (Products)

After login, customers see a beautiful product grid. We will fetch products from the database and pass them to a Bootstrap card layout.

5.1 Add the /products route

JS

```
// in server.js
app.get('/products', (req, res) => {
  if (!req.session.user) return res.redirect('/login');
  const products = db.prepare(
    'SELECT * FROM products WHERE stock > 0'
  ).all();
  res.render('products', { products });
});
```

5.2 Create views/products.ejs

HTML

```
<!DOCTYPE html>
<html><head>
<title>Our Bakery</title>
<link rel="stylesheet" href="/css/bootstrap.min.css">
</head>
<body class="bg-light">
  <nav class="navbar navbar-dark" style="background:#5a3a22;">
    <div class="container">
      <a class="navbar-brand" href="/">Sweet Crust Bakery</a>
      <div>
        <a href="/cart" class="btn btn-warning btn-sm">Cart</a>
        <a href="/logout" class="btn btn-light btn-sm">Logout</a>
      </div>
    </div>
  </nav>

  <div class="container py-4">
    <h2 class="mb-4" style="color:#5a3a22;">Today's Menu</h2>
    <div class="row g-4">
      <% products.forEach(p => { %>
        <div class="col-md-4">
          <div class="card h-100 shadow-sm">
            <div class="card-body">
              <h5 class="card-title"><%= p.name %></h5>
              <p class="text-muted small">In stock: <%= p.stock %></p>
              <h4 style="color:#b8860b;">Rs. <%= p.price %></h4>
              <form method="POST" action="/cart/add">
                <input type="hidden" name="id" value="<%= p.id %>">
                <button class="btn btn-warning w-100">Add to Cart</button>
              </form>
            </div>
          </div>
        </div>
      <% }) %>
    </div>
  </div>
</body></html>
```

NOTE: About /css/bootstrap.min.css

We reference Bootstrap from **/css/bootstrap.min.css** in our public folder. You can either download bootstrap.min.css from **getbootstrap.com** and save it to **public/css/**, or replace the link in every page with the CDN URL: https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css

5.3 Preview

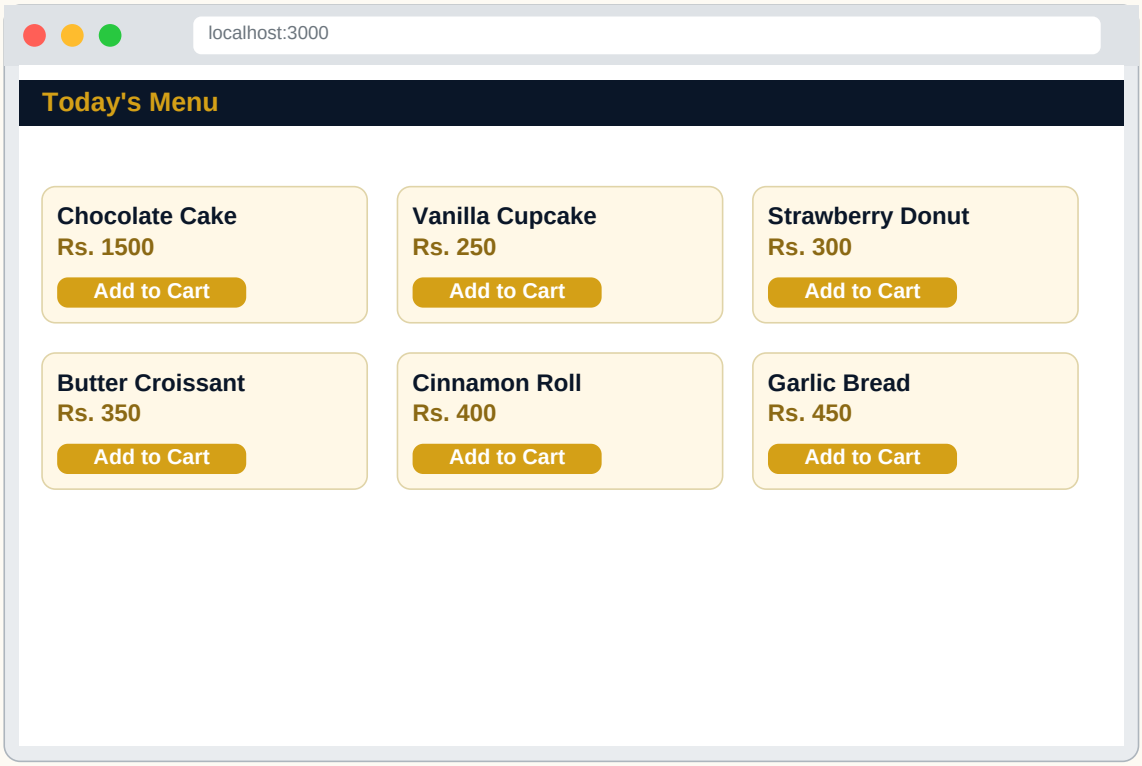

Figure 9.1 - Product cards rendered with Bootstrap

TIP: Make it your own

Replace product names with whatever your bakery sells. Add a real image: upload it to **public/images/** and store the filename in the *image* column.

DONE: Step 5 complete

Customers can see your full bakery menu. Next, we let them add items to a cart and check out.

STEP 6 Online Ordering & Cart

We use the user's session to hold their cart. Each cart is a list of **{ id, name, price, qty }** objects. When they check out, the cart becomes a row in the orders table.

JS

```
// in server.js

// Add to cart
app.post('/cart/add', (req, res) => {
  if (!req.session.user) return res.redirect('/login');
  const product = db.prepare(
    'SELECT * FROM products WHERE id = ?'
  ).get(req.body.id);
  if (!product) return res.redirect('/products');

  req.session.cart = req.session.cart || [];
  const existing = req.session.cart.find(i => i.id === product.id);
  if (existing) {
    existing.qty += 1;
  } else {
    req.session.cart.push({
      id: product.id, name: product.name,
      price: product.price, qty: 1
    });
  }
  res.redirect('/cart');
});

// View cart
app.get('/cart', (req, res) => {
  if (!req.session.user) return res.redirect('/login');
  const cart = req.session.cart || [];
  const total = cart.reduce((s, i) => s + i.price * i.qty, 0);
  res.render('cart', { cart, total });
});

// Place order (checkout)
app.post('/checkout', (req, res) => {
  if (!req.session.user) return res.redirect('/login');
  const cart = req.session.cart || [];
  if (cart.length === 0) return res.redirect('/products');
  const total = cart.reduce((s, i) => s + i.price * i.qty, 0);

  // Save order
  const result = db.prepare(
    'INSERT INTO orders (user_id, items, total) VALUES (?, ?, ?)'
  ).run(req.session.user.id, JSON.stringify(cart), total);

  // Reduce stock for every product in the cart
  const updateStock = db.prepare(
    'UPDATE products SET stock = stock - ? WHERE id = ?'
  );
  cart.forEach(i => updateStock.run(i.qty, i.id));

  req.session.cart = [];
  res.redirect('/bill/' + result.lastInsertRowid);
});
```

6.1 Create views/cart.ejs

HTML

```
<!DOCTYPE html>
<html><head>
<title>Your Cart</title>
<link rel="stylesheet" href="/css/bootstrap.min.css">
</head>
<body class="bg-light">
  <div class="container py-4">
    <h2 style="color:#5a3a22;">Your Cart</h2>

    <% if (cart.length === 0) { %>
      <p>Your cart is empty. <a href="/products">Browse products</a></p>
    <% } else { %>
      <table class="table bg-white shadow-sm">
        <thead style="background:#fff8e7;">
          <tr><th>Item</th><th>Qty</th><th>Price</th><th>Total</th></tr>
        </thead>
        <tbody>
          <% cart.forEach(i => { %>
            <tr>
              <td><%= i.name %></td>
              <td><%= i.qty %></td>
              <td>Rs. <%= i.price %></td>
              <td>Rs. <%= i.price * i.qty %></td>
            </tr>
          <% }) %>
        </tbody>
      </table>
      <h3 class="text-end">Total: Rs. <%= total %></h3>
      <form method="POST" action="/checkout" class="text-end">
        <button class="btn btn-lg btn-warning">Place Order</button>
      </form>
    <% } %>
  </div>
</body></html>
```

NOTE: Why JSON.stringify?

SQLite cannot store an array directly. We turn the cart into a string with **JSON.stringify** to save it, and parse it back with **JSON.parse** when we display the bill.

STEP 7

Billing System

After checkout the user is redirected to **/bill/:id** - a printable receipt page showing every item, the total, the date, and the order number.

JS

```
// in server.js
app.get('/bill/:id', (req, res) => {
  if (!req.session.user) return res.redirect('/login');
  const order = db.prepare(
    'SELECT * FROM orders WHERE id = ? AND user_id = ?'
  ).get(req.params.id, req.session.user.id);
  if (!order) return res.send('Order not found');
  order.items = JSON.parse(order.items);
  res.render('bill', { order });
});
```

7.1 Create views/bill.ejs

HTML

```
<!DOCTYPE html>
<html><head>
<title>Bill #<%= order.id %></title>
<link rel="stylesheet" href="/css/bootstrap.min.css">
<style>
  @media print { .no-print { display:none; } }
  body { background:#fff8e7; }
</style>
</head>
<body>
<div class="container py-5" style="max-width:600px;">
  <div class="card shadow">
    <div class="card-body p-4">
      <h2 class="text-center" style="color:#5a3a22;">Sweet Crust Bakery</h2>
      <p class="text-center text-muted small">Thank you for your order!</p>
      <hr>
      <p>Order #<%= order.id %><br>
         Date: <%= order.created_at %></p>
      <table class="table">
        <thead><tr><th>Item</th><th>Qty</th><th>Total</th></tr></thead>
        <tbody>
          <% order.items.forEach(i => { %>
            <tr>
              <td><%= i.name %></td>
              <td><%= i.qty %></td>
              <td>Rs. <%= i.price * i.qty %></td>
            </tr>
          <% }) %>
        </tbody>
      </table>
      <h4 class="text-end">Grand Total: Rs. <%= order.total %></h4>
      <div class="no-print text-center mt-4">
        <button class="btn btn-warning" onclick="window.print()">
          Print Bill
        </button>
        <a href="/products" class="btn btn-outline-dark">Order More</a>
      </div>
    </div>
  </div>
</div>
</body></html>
```

Figure 11.1 - Printable bill preview

TIP: Print-friendly CSS

The @media print rule hides the buttons when the user prints. The bill comes out clean on paper or as a saved PDF (Save as PDF in browser print dialog).

DONE: Step 7 complete

Customers can now place an order and immediately see a printable bill.

STEP 8 Stock Management

Stock already decreases automatically every time a customer checks out (we did that in Step 6). Here we add the admin's ability to **add new products** and **refill stock**.

8.1 Admin-only middleware

JS

```
// in server.js - add this helper
function adminOnly(req, res, next) {
  if (req.session.user && req.session.user.role === 'admin') {
    return next();
  }
  res.status(403).send('Admins only');
}
```

8.2 Stock routes

JS

```
// Update stock
app.post('/admin/stock', adminOnly, (req, res) => {
  const { id, stock } = req.body;
  db.prepare('UPDATE products SET stock = ? WHERE id = ?')
    .run(stock, id);
  res.redirect('/admin');
});

// Add new product
app.post('/admin/product', adminOnly, (req, res) => {
  const { name, price, stock } = req.body;
  db.prepare(
    'INSERT INTO products (name, price, stock) VALUES (?, ?, ?)'
  ).run(name, price, stock);
  res.redirect('/admin');
});

// Delete product
app.post('/admin/product/delete', adminOnly, (req, res) => {
  db.prepare('DELETE FROM products WHERE id = ?').run(req.body.id);
  res.redirect('/admin');
});
```

NOTE: Why low-stock matters

When a product's stock reaches 0, it no longer appears in the customer's product list (we filtered with **WHERE stock > 0**). An admin can refill stock from the dashboard at any time.

STEP 9

Admin Dashboard

The admin dashboard is a single page with three sections: summary cards (today's sales), a list of all orders, and a stock manager with one row per product.

9.1 The /admin route

JS

```
app.get('/admin', adminOnly, (req, res) => {
  const products = db.prepare('SELECT * FROM products').all();
  const orders = db.prepare(
    'SELECT o.*, u.name AS customer FROM orders o ' +
    'JOIN users u ON o.user_id = u.id ORDER BY o.id DESC'
  ).all();
  const totalSales = db.prepare(
    'SELECT SUM(total) AS total FROM orders'
  ).get().total || 0;
  res.render('admin', { products, orders, totalSales });
});
```

9.2 views/admin.ejs (key parts)

HTML

```
<div class="container py-4">
  <h2 style="color:#5a3a22;">Admin Dashboard</h2>

  <!-- Summary cards -->
  <div class="row g-3 my-3">
    <div class="col-md-4">
      <div class="card text-center p-3" style="background:#fff8e7;">
        <h6>Total Products</h6>
        <h3><%= products.length %></h3>
      </div>
    </div>
    <div class="col-md-4">
      <div class="card text-center p-3" style="background:#fff8e7;">
        <h6>Total Orders</h6>
        <h3><%= orders.length %></h3>
      </div>
    </div>
    <div class="col-md-4">
      <div class="card text-center p-3" style="background:#fff8e7;">
        <h6>Total Sales (Rs.)</h6>
        <h3><%= totalSales %></h3>
      </div>
    </div>
  </div>

  <!-- Stock manager -->
  <h4 class="mt-4">Stock</h4>
  <table class="table bg-white">
    <thead><tr><th>Product</th><th>Price</th><th>Stock</th><th>Update</th></tr></thead>
    <tbody>
      <% products.forEach(p => { %>
        <tr>
          <td><%= p.name %></td>
          <td>Rs. <%= p.price %></td>
          <td><%= p.stock %></td>
          <td>
            <form method="POST" action="/admin/stock" class="d-flex gap-2">
              <input type="hidden" name="id" value="<%= p.id %>">
              <input type="number" name="stock" value="<%= p.stock %>"
                     class="form-control form-control-sm" style="max-width:80px;">
              <button class="btn btn-sm btn-warning">Save</button>
            </form>
          </td>
        </tr>
      <% }) %>
    </tbody>
  </table>
</div>
```

Figure 13.1 - Admin dashboard preview

DONE: Step 9 complete

Your bakery now has a working admin panel. Log in with admin@bakery.com / admin123 to try it.

STEP 10

Testing the Application

Testing means making sure the system actually works. We will do two kinds of testing: **manual testing** (you click around) and **basic automated testing** using a script.

10.1 Manual Test Plan

#	Test Case	Expected Result	Status
1	Open localhost:3000	Home page loads	
2	Click Register, fill form	Redirected to login	
3	Login with new account	Redirected to /products	
4	Click Add to Cart on a product	Redirected to /cart with 1 item	
5	Add same product again	Quantity becomes 2	
6	Press Place Order	Bill page shown	
7	Check stock on /products	Stock decreased	
8	Login as admin	Redirected to /admin	
9	Update stock to 0	Item disappears from /products	
10	Logout, try /admin	Should be blocked	

How to use this: Print this page or copy the table. Walk through each test on your running app and put a tick (or **PASS** / **FAIL**) in the Status column.

10.2 Simple automated test

Create **test.js** at the project root. This script tests that a customer can register and login programmatically.

JS

```
// test.js - tiny end-to-end check
// Run with: node test.js  (server must be running)
const http = require('http');

function post(path, data) {
  return new Promise((resolve) => {
    const body = new URLSearchParams(data).toString();
    const req = http.request({
      host: 'localhost', port: 3000, path: path,
      method: 'POST',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
        'Content-Length': body.length
      }
    }, res => resolve(res.statusCode));
    req.write(body);
    req.end();
  });
}

(async () => {
  console.log('TEST 1: Register a new user');
  const r1 = await post('/register', {
    name: 'Test User',
    email: 't' + Date.now() + '@a.com',
    password: 'pass123'
  });
  console.log(r1 === 302 ? 'PASS' : 'FAIL', r1);

  console.log('TEST 2: Login with admin');
  const r2 = await post('/login', {
    email: 'admin@bakery.com', password: 'admin123'
  });
  console.log(r2 === 302 ? 'PASS' : 'FAIL', r2);
})();
```

Run the tests

BASH

```
# Make sure the server is running first
node server.js

# In a SECOND terminal:
node test.js
```

TIP: Going further with testing

When you are comfortable, look at the **Jest** and **Supertest** npm packages. They let you write proper unit tests for each route. Tutorials available on egotechworld.com.

11. Running, Debugging & Deployment

11.1 Daily run

```bash
# Start the bakery system
cd bakery-app
node server.js

# Open browser:
http://localhost:3000
```

11.2 Auto-restart with nodemon

Tired of stopping and starting the server every time you save a file? Install **nodemon** - it watches your files and restarts automatically.

```bash
npm install -g nodemon
nodemon server.js
```

11.3 Common errors & fixes

Error message	Fix
`EADDRINUSE port 3000`	Another app is using port 3000. Change PORT in server.js to 3001.
`Cannot find module 'express'`	Run npm install again inside the project folder.
`SqliteError: no such table`	Run node seed.js first to create the tables.
`TypeError: Cannot read property of undefined`	A variable is missing. Check the route passes all data to res.render.

11.4 Going live

When you are ready to publish online, options include **Render**, **Railway**, and **Glitch** - all have free tiers for small Node.js apps. Push your code to GitHub, connect the host, set the start command to node server.js and you are live.

12. Conclusion & Next Steps

Congratulations - you have built a full-stack bakery management system! You used HTML, Bootstrap, Node.js, Express, and SQLite to create something real and useful. Most importantly, you now understand how the pieces fit together: the front end shows pages, the server handles requests, and the database remembers everything.

What you can build next

- Add product photos and a category filter.
- Send order confirmation emails using nodemailer.
- Add a dashboard chart showing weekly sales.
- Build a delivery tracking module.
- Create a mobile-friendly Progressive Web App version.

TIP: Get more tutorials & tools

Visit **egotechworld.com** for more beginner tutorials, free coding tools, software showcase, and full-stack project templates. Built and maintained by EGOTECHWORLD PVT LTD - helping you learn by building.

End of tutorial - Happy baking and happy coding!