🔹 Frontend (what users see)

- **JavaScript (React.js + Tailwind CSS)** → Best balance of speed, responsiveness, and ecosystem.
- Alternatives (if React feels heavy):
  - **Vanilla JS + Bootstrap** (simpler, but less impressive).
  - **Angular/Vue** (ok, but React is more common in student projects).

👉 **Why JS/React?**

- Easy to deploy on **Vercel/Netlify** (free).
- Big library ecosystem (product cards, modals, forms).
- Responsive design is straightforward with Tailwind.

🔹 Backend + API

- **JavaScript (Node.js + Express)** → Natural choice if you're already using JS for frontend.
- Alternatives:
  - **Python (Flask / Django)** → Great, but deployment on free tiers may be trickier.
  - **PHP (Laravel)** → Possible, but not as modern/flexible for APIs.

👉 **Why Node/Express?**

- Same language across frontend + backend (JS).
- Deploys easily on **Render/Railway/Fly.io** free tiers.
- Many tutorials/examples for e-commerce flows.

🔹 Database

- **MongoDB (NoSQL)** → pairs perfectly with Node/Express. Free tier on **MongoDB Atlas**.
- Alternatives:
  - **PostgreSQL (SQL)** → more structured, free hosting via Supabase/Neon.
  - **MySQL** → common, but less beginner-friendly than Mongo.

👉 **Why MongoDB?**

- JSON-like documents → easy to store products, users, orders.
- Quick to integrate with Node.js using Mongoose.

🔹 Payment Gateway (custom simulator)

- **Backend language = Node.js/Express** (or whatever backend you choose).
- Just write custom endpoints that mimic how real payment APIs work (validate card, mark order paid).

🔹 Free Hosting Recommendations

- **Frontend (React)** → Vercel / Netlify.
- **Backend (Node)** → Railway / Render / Fly.io.
- **Database** → MongoDB Atlas free cluster.

✅ **Best combo for your assignment (time + marks + simplicity):**
React (frontend) + Node/Express (backend & payment gateway) + MongoDB Atlas (database)

Frontend = React
Backend = Node.js/Express
Payment Gateway = Node.js (custom simulation)
Database = MongoDB

🗓️ Two-Week Project Plan (14 Days)

Week 1 – Building the Core

Day 1 – Setup

- Install Node.js, MongoDB (local or Atlas), Git.
- Create GitHub repo.
- Initialize backend (`npm init`, install Express, Mongoose, dotenv, cors).
- Initialize frontend with React (Vite or Create React App) + Tailwind CSS.

Day 2 – Database & Products API

- Design MongoDB schemas: `Product`, `User`, `Order`, `Transaction`.
- Implement MongoDB connection.
- Create `GET /products` API route to fetch products.
- Add a `seed.js` script to insert sample pillows & candles.

Day 3 – Frontend Product Pages

- Build product listing page (grid of products).
- Build product detail page (image, description, price, Add to Cart button).

- Make it responsive (mobile & desktop).

Day 4 – Cart Functionality

- Implement cart using React Context or localStorage.
- Add "View Cart" page with update quantity/remove.
- Show subtotal and checkout button.

Day 5 – Auth Basics

- Backend: add `/auth/register` and `/auth/login` (JWT + bcrypt).
- Frontend: create login/register forms.
- Store token in localStorage for authenticated requests.

Day 6 – Orders

- Backend: `POST /orders` to create an order with items + shipping info.
- Frontend: checkout form → call API to create order.

Day 7 – Custom Payment Gateway

- Backend:
  - `/gateway/initiate` → generate paymentToken.
  - `/gateway/charge` → simulate card validation, mark order `paid` or `failed`.
- Frontend: integrate with checkout → collect mock card details → send to backend.
- Test flow: add product → cart → checkout → pay → order marked `paid`.

Week 2 – Polish, Test & Deploy

Day 8 – Admin/Product Management

- **Add `/admin/products` (protected by JWT, role = admin).**
- **Simple page to add products manually.**
- **Or just keep using seed data if short on time.**

Day 9 – UI & Responsiveness

- **Polish UI with Tailwind (cards, buttons, forms).**
- **Ensure mobile responsiveness.**

Day 10 – Security & Validation

- **Sanitize inputs.**
- **Validate forms (email format, password length, card number length).**
- **Protect routes with JWT middleware.**

Day 11 – Testing

- **Test flows:**
  - **Guest checkout.**
  - **Login + checkout.**
  - **Payment success & failure.**
  - **Cart updates persist correctly.**

Day 12 – Deploy Backend

- **Push backend to GitHub.**
- **Deploy on Railway or Render.**
- **Connect to MongoDB Atlas free cluster.**

- Test APIs live with Postman.

Day 13 – Deploy Frontend

- Deploy React frontend on Vercel or Netlify.
- Point frontend to backend API URL.
- Test full flow live (products → cart → checkout → pay).

Day 14 – Final Checks & Submission

- Write clear README.md:
  - Demo link.
  - Test card numbers (`4242...` success, `4000...` fail).
  - Setup instructions (clone repo, run locally).
- Test on phone (responsiveness).
- Take screenshots or record a short demo video.
- Submit GitHub link + deployed app link.

🗓️ Two-Week Learning + Building Plan

Week 1 – Learning Core Skills + Building Core Features

Day 1 – Setup & Basics

- 📚 Learn:

    - What Node.js is.
    - What Express.js does.
    - Basic Git/GitHub workflow.
- 🛠️ Build:
    - Create GitHub repo.
    - Initialize backend (Node + Express).
    - Initialize frontend (React + Tailwind).

Day 2 – Database + API Basics

- 📚 Learn:
    - What MongoDB is.
    - How to define a schema with Mongoose.
    - REST API basics (GET, POST).
- 🛠️ Build:
    - Connect to MongoDB Atlas.
    - Create `Product` model.
    - Implement `GET /products` route.
    - Seed sample products.

Day 3 – React Basics + Product Listing

- 📚 Learn:

    - JSX syntax (HTML in React).
    - `useState` & `useEffect`.
    - Mapping arrays to components.
- 🛠️ Build:
    - Fetch `/products` API.
    - Display products in a grid.
    - Product detail page with image + description.

Day 4 – State Management + Cart

- 📚 Learn:
    - React Context API OR localStorage.
    - Event handling (onClick, forms).
- 🛠️ Build:
    - Add-to-cart button.
    - Cart page → update quantity, remove items.
    - Show subtotal.

Day 5 – Authentication

- 📚 Learn:
    - What JWT (JSON Web Token) is.
    - Password hashing with bcrypt.

    - React forms + handling API responses.
- 🛠️ Build:
    - Backend: `/auth/register`, `/auth/login`.
    - Frontend: login & signup forms.
    - Store JWT in localStorage.

Day 6 – Orders

- 📚 Learn:
    - How to create an order API.
    - Passing data from frontend forms to backend.
- 🛠️ Build:
    - Backend: `POST /orders` → saves cart + shipping info.
    - Frontend: checkout form → call API.

Day 7 – Payment Gateway (Simulated)

- 📚 Learn:
    - How real payment gateways work.
    - How to design your own “fake” gateway API.
- 🛠️ Build:
    - Backend: `/gateway/initiate` + `/gateway/charge`.
    - Frontend: payment form (mock card number, expiry, CVV).

- Test full flow: Add → Cart → Checkout → Pay → Success/Fail.

Week 2 – Polish, Security, Deployment

Day 8 – Admin Panel (Optional if time is tight)

- 📚 Learn:
  - Route protection with JWT (role = admin).
- 🛠️ Build:
  - Simple admin page to add products OR stick with seed data.

Day 9 – UI & Responsiveness

- 📚 Learn:
  - Tailwind responsive classes (`md:`, `lg:`).
  - Flexbox/Grid basics.
- 🛠️ Build:
  - Polish product cards, cart, forms.
  - Ensure mobile + desktop views are clean.

Day 10 – Security + Validation

- 📚 Learn:
  - Input validation (check email, password length).
  - Middleware in Express.

- 🛠️ **Build:**
  - **Validate auth + checkout forms.**
  - **Sanitize inputs before saving to DB.**

Day 11 – Testing

- 📚 **Learn:**
  - **Testing APIs with Postman/Thunder Client.**
  - **Manual testing strategies.**
- 🛠️ **Build:**
  - **Test full e-commerce flow.**
  - **Test payment success + failure.**
  - **Fix bugs.**

Day 12 – Backend Deployment

- 📚 **Learn:**
  - **How to deploy Node.js apps on Railway or Render.**
  - **Environment variables (Mongo URI, JWT secret).**
- 🛠️ **Build:**
  - **Deploy backend.**
  - **Connect live API to MongoDB Atlas.**
  - **Test `/products` endpoint online.**

Day 13 – Frontend Deployment

- 📚 **Learn:**
  - **How to deploy React apps on Vercel/Netlify.**
  - **How to set frontend API URL to deployed backend.**
- 🛠️ **Build:**
  - **Deploy frontend.**
  - **Connect frontend → backend.**
  - **Test full live flow.**

Day 14 – Final Checks & Submission

- 📚 **Learn:**
  - **How to write a good README (project description, setup, test instructions).**
- 🛠️ **Build:**
  - **Write README (with demo links, test cards, admin credentials if needed).**
  - **Test responsiveness on your phone.**
  - **Record screenshots/video demo.**
  - **Submit GitHub + live demo link.**

Gold/Yellow: The flame-shaped design is in a shiny gold tone.

Light Beige/Cream: The background is a soft beige or cream color.

Option 3: Modern & Luxurious

- **Background:** Beige (#eadbba)
- **Primary Accent:** Gold (#c9a227)
- **Secondary Accent:** Deep Navy (#1c2a39) or Emerald Green (#046307)
- **Text:** White (#ffffff) on dark areas, Charcoal (#222222) on light areas

👉 Strong contrast + luxury feel = **premium product branding**.

Keep **2–3 main colors** (background, primary, secondary) and use **gold/cream sparingly as highlights** so the site doesn't look overwhelming.

Add **lots of white space** for a premium clean look.