**BrightCart: E-commerce Data Structures Demo**

**Overview**

BrightCart is a lightweight, proof-of-concept Flask backend designed to demonstrate the practical application and manipulation of fundamental computer science data structures within a simulated e-commerce environment. The application exposes RESTful endpoints for real-time interaction with these structures.

**Core Implementations & Algorithms**

The table below maps each simulated e-commerce component to the underlying data structure and the key algorithm used to drive its functionality.

|  |  |  |
| --- | --- | --- |
| **Component** | **Primary Data Structure** | **Key Algorithm / Concept** |
| **Product Carousel** | **Doubly Linked List** | **Floyd's Tortoise and Hare** (Cycle Detection and Repair) |
| **Shopping Cart** | **Stack** (LIFO) | Last-In, First-Out (LIFO) operation enabling **Undo** feature |
| **Order Processor** | **Deque / Queue** (FIFO) | Priority Insertion Logic for **VIP Orders** |
| **Customer Lookup** | **Binary Search Tree (BST)** | Efficient Binary Search Tree Traversal (O(log n) lookup) |
| **Customer Promotion** | **BST** | **Tree Rotations** (Left & Right) to promote "hot" customers (Splay-Tree concept) |
| **Product Recommendations** | **Graph** (Adjacency Map) | Weighted Graph Traversal and ranking based on co-purchase frequency |

**User Interface (Frontend)**

The frontend is a single-file, responsive HTML/JavaScript application (frontend.html) designed purely as a **Prototype Controller** to interact with the Flask API endpoints.

* **Architecture:** A lightweight, single-page application (SPA) model where all UI logic and presentation are contained in one file.
* **Interaction:** Uses plain **JavaScript's fetch API** to send and receive JSON data to the backend, demonstrating real-time updates to the data structures.
* **Styling & Theme:** Styled using **Tailwind CSS** with a custom "Girly Theme" (fuchsia/pink/violet palettes) for high visual contrast and modern aesthetics.
* **Client Resilience:** Implements an **exponential backoff** mechanism within the apiCall function to handle transient network or rate-limiting errors.
* **Accessibility:** Includes keyboard shortcuts, notably **Ctrl/Cmd + Z** to trigger the Cart's **Undo Stack** operation.

**Technology Stack**

* **Backend Framework:** Python **Flask**
* **Data Structures:** Custom Python classes for Linked List, BST, Stack, Deque/Queue, and Graph
* **Cross-Origin Handling:** flask-cors
* **Frontend UI:** **HTML5**, **JavaScript**, **Tailwind CSS**

# SkillBridge LMS

**Introduction**

SkillBridge LMS (Learning Management System) is a modern, responsive, and efficient platform designed to make online learning seamless for students and instructors. It provides an interactive experience with user-friendly dashboards, secure authentication, and real-time course progress tracking — all in one place.

**Tech Stack**

Frontend:

* **Next.js 14** (React framework for Server-Side Rendering (SSR) and performance optimization)
* **Tailwind CSS** (for modern, responsive styling)
* **ShadCN/UI** (for clean and reusable UI components)
* **Framer Motion** (for animations and transitions)

Backend & API:

* **Node.js + Express** (for building robust RESTful APIs)
* **MongoDB with Mongoose** (for data storage and schema modeling)
* **JWT Authentication** (for secure user login and role-based access control)

Tools & Libraries:

* **Axios** (for efficient API communication)
* **React Hook Form** (for streamlined form validation)
* **Radix UI** (for accessible UI primitives)

**Features**

* 📚 **Course Management:** Instructors can create, update, and manage courses easily.
* 👨‍🎓 **Student Dashboard:** Track progress, view enrolled courses, and seamlessly continue learning.
* 🔐 **Authentication:** Secure login/signup using **JWT tokens**.
* 🎥 **Video Integration:** Stream and watch course videos with adaptive layouts.
* 🧾 **Progress Tracking:** Save user progress automatically and reliably.
* ⚙️ **Responsive Design:** Works perfectly on mobile, tablet, and desktop.
* 💬 **Announcements & Notifications:** Keep learners informed about updates and new content.

**ShopEase Point-of-Sale (POS) System**

**Overview**

ShopEase is a two-part application designed for fast and accurate processing of retail sales. It features a modern, single-page frontend that communicates with a robust **FastAPI backend** to handle critical business logic, including real-time cart total calculation (subtotal, tax, and final total).

**Tech Stack**

|  |  |  |
| --- | --- | --- |
| **Component** | **Technology** | **Rationale / Detail** |
| **Backend** | **Python** (3.8+) | Primary language for server-side logic. |
| **Web Framework** | **FastAPI** | Chosen for high performance and modern asynchronous features, replacing an assumed legacy Flask application. |
| **Data Handling** | **Pydantic** | Used for data validation and modeling (implied by pip install pydantic). |
| **Styling** | **Tailwind CSS** (via CDN) | Used for rapid, utility-first styling of the POS interface. |
| **Frontend Logic** | **Vanilla JavaScript fetch API** | Used to manage asynchronous POST requests and handle the API contract. |
| **Frontend Markup** | **HTML** | Standard structure for the single-page application. |

**Key Features**

1. **Dynamic Cart Calculation (Critical Feature):** The application uses the **POST /cart/calculate** API endpoint to dynamically determine the subtotal, tax\_amount, and final total in real-time.
2. **Two-Tier Architecture:** The system is split into a standalone HTML/JS frontend and a separate FastAPI backend.
3. **Real-Time UI Update:** The frontend logic ensures that when products are added, the UI elements (subtotal-display, total-display, and checkout-btn) are updated immediately using data returned from the backend.
4. **Robust Connectivity:** Includes necessary **CORS Middleware** configuration on the backend to allow the frontend to connect without security issues.
5. **Health Check:** Provides a simple GET / endpoint to confirm API connectivity.