

# Project Report : Smart E-Commerce System

## Team Members:

Anushka Mahur (B24EE1006)  
Harshita Pareek (B24CM1028)  
Riya Dhyawna (B24CS1066)  
Aashma Yadav (B24CS1002)

## Course: Data Structures & Algorithms

(CSL2020)

## 1. Introduction

This project, Smart E-Commerce System, is a desktop-based application that simulates a mini online shopping platform. It is built using:

- **C++** for backend logic and data-structure implementation
- **Python Tkinter** for the graphical user interface
- **File handling** for storing & retrieving product data
- **DSA concepts** for efficient search, filtering, sorting, and cart management

The objective is to design a **real-life application** that demonstrates the usage and importance of Data Structures and Algorithms in handling real-world datasets and operations such as searching, filtering, cart operations, product management, etc.

## 2. Problem Statement

Modern E-Commerce systems require:

- Fast product search
- Efficient filtering & sorting
- Real-time cart management
- Data consistency
- Scalability for large product lists

The problem is to build a **functional desktop application** that models these requirements while focusing on the **use of data structures and algorithms**.

## 3. Proposed Solution

The solution is a **Smart E-Commerce System** with:

### 3.1 Backend (C++)

- Manages all product operations
- Performs search, sorting, filtering
- Handles cart operations
- Reads product data from input files
- Communicates with the GUI using standard input/output

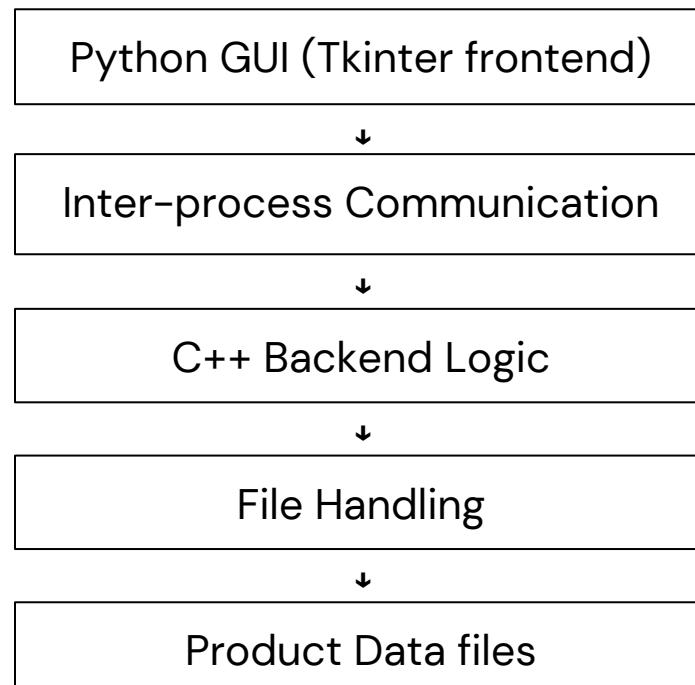
### 3.2 Frontend (Python Tkinter)

- Displays product catalog
- Applies filters and search options
- Shows product details
- Allows adding/removing items from cart
- Communicates with C++ backend through `BackendInterface.py`

The combination of **Python GUI + C++ speed + DSA logic** results in a smooth and efficient application.

## 4. System Architecture

### 4.1 Architecture Diagram



## 5. Implementation Details

This section explains **every data structure, algorithm, and design choice** used.

### Data Structures & Algorithms

This document provides a complete overview of all Data Structures and Algorithms implemented in the Smart E-Commerce System. Each technique is explained with its purpose, time complexity, and justification for real-world applicability. The goal is to demonstrate a strong, multi-DSA approach suitable for scalable, search-heavy applications.

#### 5.1 Trie (Prefix Tree)

The Trie is used to implement **fast product-name autocomplete** and **prefix-based searching**, making the search bar highly responsive.

##### Purpose:

- Prefix-based search
- Autocomplete suggestions
- Fast incremental search while typing

##### Why a Trie?

- Lookup time depends **only on input word length**, not dataset size
- Much faster than scanning the entire product list
- Industry-standard for search bars, browsers, and type-ahead tools
- Perfect for datasets with many textual fields (product names)

### Operations & Complexity

Operation	Description	Complexity
Insert	Insert product name	$O(L)$
Search Prefix	Check if prefix exists	$O(L)$
Autocomplete	DFS from prefix node	$O(K \times L)$

## Where:

- $L$  = length of word
- $K$  = number of suggestions returned

Since product names are short (10–20 chars), Trie operations are **effectively instantaneous**.

## 5.2 Fuzzy Search (Edit Distance / Levenshtein-Based Search)

Fuzzy search enhances the system by handling **misspellings**, **typos**, and **near matches**.

### Purpose:

- Find results even when user types:
  - “iphon” instead of “iphone”
  - “samsang” instead of “samsung”
- Improve search tolerance and user experience

### Algorithm

- **Levenshtein Edit Distance** using dynamic programming
- Measures difference via insertions, deletions, substitutions

### Complexity

Operation	Complexity
Edit Distance (single comparison)	$O(m \times n)$
Fuzzy search over all products	$O(P \times m \times n)$

### Where:

- $m$  = length of input
- $n$  = product name length
- $P$  = total number of products

### Why this is still efficient

- Fuzzy search is a **fallback mechanism** (used only when Trie finds no matches)
- Input sizes are small
- Product names are short

- Search space is reduced when filters or Trie results already narrow down candidates

This achieves a balance between **accuracy** and **speed**.

### 5.3 Graph (Adjacency List) for Category Relationships

A lightweight graph structure is used to represent relationships such as:

- Similar product groups
- Category connections
- Recommendation adjacency

#### Structure

```
unordered_map<string, vector<string>> adj;
```

#### Why Graphs?

- Graphs model relationships far better than arrays or lists
- Ideal for showing “related products” or “similar categories”

#### Operations & Complexity

Operation	Complexity
Add Edge	<b>O(1)</b>
Get Neighbors	<b>O(K)</b> (K = number of related items)

Graph operations remain extremely fast due to the shallow and limited category structure.

### 5.4 Vector-Based Cart Implementation

The shopping cart uses `std::vector`, optimized for small dynamic collections.

#### Why a Vector?

- Cart sizes are always small (< 30 items)
- Vector is cache-friendly and fastest for sequential scans
- Minimal overhead compared to maps or linked lists

## Operations & Complexity

Operation	Complexity
Add Item	$O(1)$
Remove Item	$O(N)$
Compute Total	$O(N)$

Given the small size of carts, vector operations behave almost constant-time in practice.

## 5.5 Product Class (OOP Structure)

Products are stored using a dedicated **class/struct**, encapsulating all necessary attributes:

- id
- name
- price
- category
- description

## Advantages

- Clear modularity
- Easy extension for attributes
- Reduced code duplication
- Cleaner interface for frontend-backend communication

Using OOP here strengthens maintainability and clarity.

## 5.6 Sorting Algorithms

Sorting is implemented using the C++ STL `sort()` function, which internally uses **Introsort** — a hybrid of:

- Quicksort
- Heapsort
- Insertion Sort

## Sorting Criteria

- Price (ascending / descending)
- Alphabetical order
- Availability
- Category-based sorting

## Complexity

- **O(n log n)** for general sorting
- Highly optimized for real-world usage
- Custom comparator functions control the ordering

## 5.7 Multi-Layer Filtering Pipeline

Filtering is implemented as a **chained conditional pipeline**, with stages including:

- Category filtering
- Price-range filtering
- Availability filtering
- Tie-in with Trie/Fuzzy search

## Why Multi-Layer Filtering?

- Reduces dataset size step-by-step
- Improves search performance
- Makes combined filters efficient

## Complexity

- Overall **O(n)**
- Faster in practice because each step shrinks the list

## 5.8 Hashing (unordered\_map)

Hash tables are used for:

- Fast product lookup by ID
- Checking cart item existence
- Quick validation operations

## Complexity

- **O(1)** average-case lookup
- Ideal for repeated operations like modifying cart quantities

## 6. Features Implemented

### Product Catalog

Displays all products with attributes:

- Name
- Price
- Rating
- Category
- Stock

### Search Function

Search products by:

- Name
- Keyword
- Product ID

### Advanced Filtering

- Price range
- Category
- Availability
- Sorting options

### Cart Management

- Add item
- Remove item
- Update quantity
- Total price calculation

### GUI Features

- Clean Tkinter layout
- Buttons and dropdowns for filters
- Product cards
- User-friendly interface

### Backend-Frontend Communication

Python sends commands → C++ processes → Output returned to Python GUI.

## 7. Code Walkthrough

### 7.1 main.cpp

- Loads products
- Initializes backend
- Waits for commands from Python
- Executes search/filter/cart logic
- Returns results

### 7.2 product.cpp / product.h

Defines:

- Product structure
- Functions to display and compare products

### 7.3 backend\_interface.py

Handles:

- Running C++ executable
- Sending input
- Reading output
- Parsing results for GUI

### 7.4 app.py (Tkinter GUI)

- Contains all UI views
- Calls backend to fetch product lists
- Displays results in TreeView
- Handles user interactions

## 8. Reproducibility

To run the project:

### Step 1 — Compile C++ backend

```
g++ ecommerce.cpp -o ecommerce
```

### Step 2 — Run Python GUI

```
python app.py
```

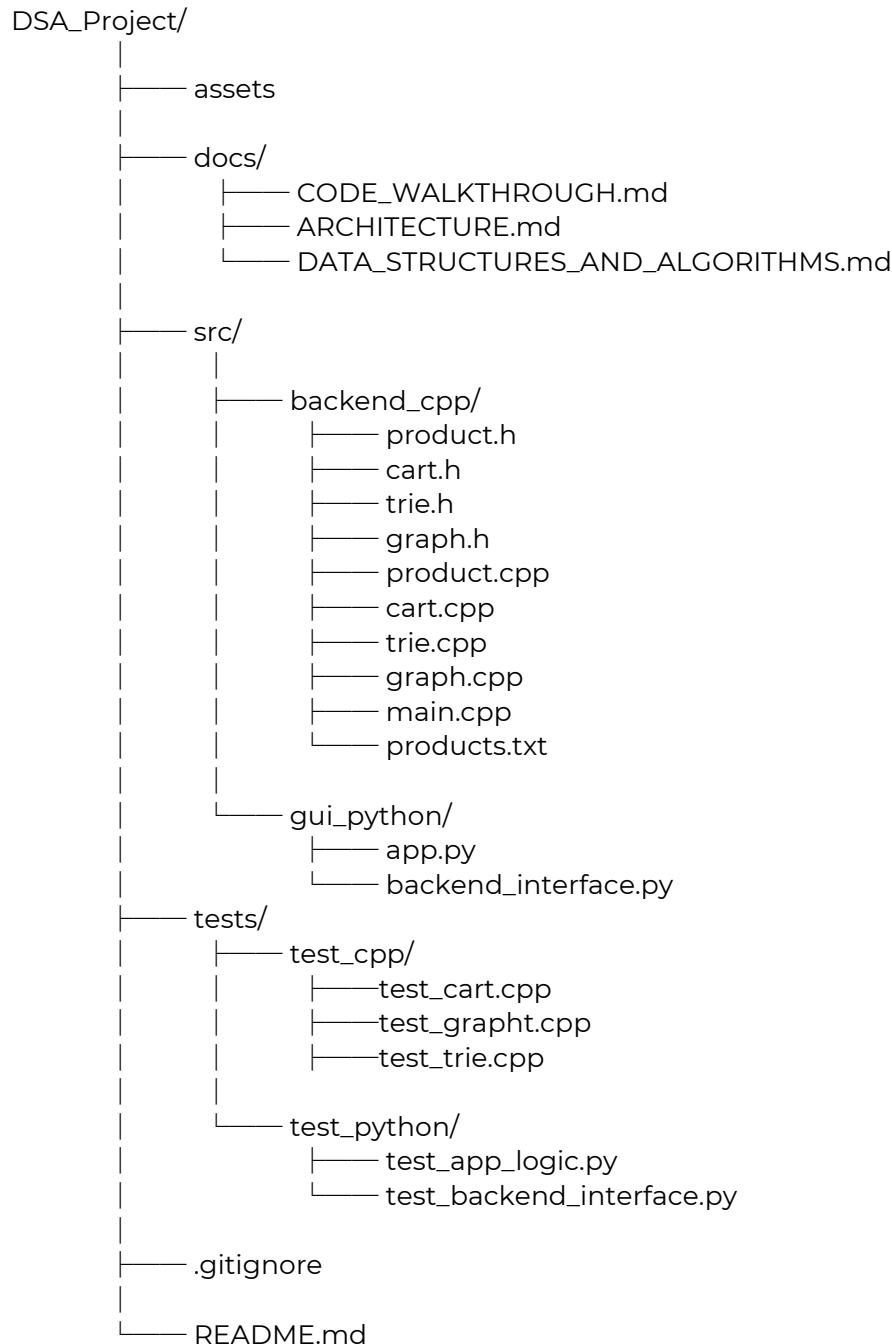
### Step 3 — Ensure product data files are present

Input files must be inside:

src/backend\_cpp/

Everything runs automatically once files are present.

## 9. Repository Structure



## 10. Conclusion

The Smart E-Commerce System successfully demonstrates:

- Application of Data Structures (vectors, maps, sets)
- Implementation of Algorithms
- File Handling and Modular Programming
- A real-world e-commerce workflow
- GUI-Backend integrated architecture

This project is scalable, extendable, and showcases practical usage of concepts learned during our DSA course.

## 11. Future Scope

- Add user authentication
- Add database integration (SQLite/PostgreSQL)
- Add more product attributes
- Implement recommendation algorithms
- Add cart persistence and order history

## 12. Team Contributions

All four members contributed significantly to the development of the Smart E-Commerce System.

While each member handled certain specific tasks, everyone participated in the **core DSA logic**, backend algorithms, testing, and discussion.

### 1. Anushka Mahur

**Primary Work:** Sorting & Filtering Logic, GUI, Documentation

**Detailed Contributions:**

- Implemented major parts of the **sorting and filtering algorithms** including price sort, rating sort, category filters, etc.
- Contributed to the DSA design: choosing suitable structures (vector, unordered\_map, custom comparators).
- Worked extensively on the **Tkinter GUI**, product display, and user interaction flow.
- Helped integrate GUI with backend outputs consistently.

- Wrote significant parts of the **README** and the **final report**, ensuring clarity and formatting.
- Participated in algorithm debugging, output validation, and complexity checks.
- Collaborated in testing features like search, filter-chain combinations, and cart functions.

## 2. Harshita Pareek

**Primary Work:** Fuzzy Search Implementation, Markdown Docs, Backend Contributions

**Detailed Contributions:**

- Designed and implemented the **fuzzy search algorithm** to improve search accuracy and handle partial matches.
- Worked on the backend search logic using DSA concepts such as string matching, scoring, and efficient iteration.
- Contributed to the writing and structuring of **README and additional markdown documentation**.
- Responsible for improving user experience through enhanced search and lookup logic.
- Helped integrate search outputs with the GUI and maintain consistent formatting.
- Worked collaboratively on testing and validating search correctness and performance.

## 3. Riya Dhyawna

**Primary Work:** Backend Functions, DSA Logic, Testing

**Detailed Contributions:**

- Contributed to backend modules involving **search, filter, and sorting operations**, ensuring proper DSA usage.
- Helped structure and maintain product data storage using vectors, maps, and hashed lookups.
- Actively worked on **testing** end-to-end functionality, especially verifying algorithm outputs and edge cases.
- Helped ensure integration consistency between GUI inputs and backend commands.
- Assisted in dataset preparation, handling missing entries, and validating product attributes.
- Participated in debugging, error-fixing, and refining time complexity of operations.
- Contributed to documentation sections related to testing and algorithm explanation.

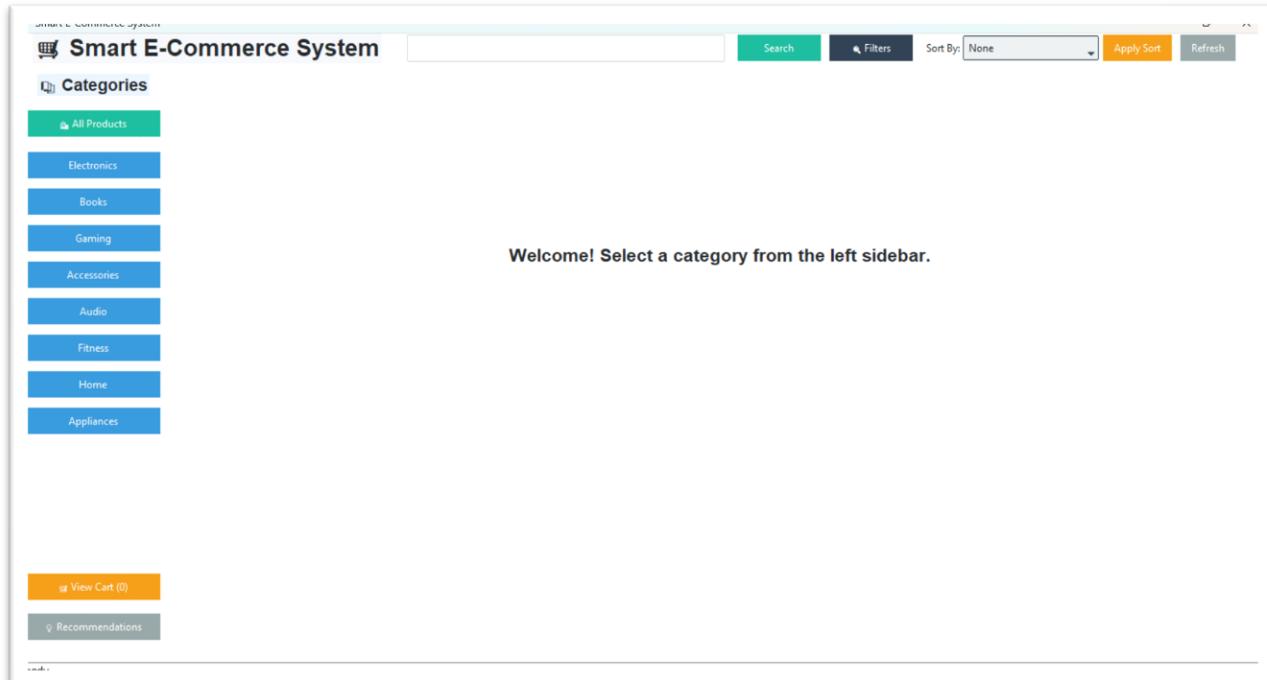
## 4. Aashma Yadav

**Primary Work:** GUI Development, Base Code Setup, DSA Support

### Detailed Contributions:

- Worked on **GUI components**, layout design, widget arrangement, and styling for a better user experience.
- Helped set up and refine the **base code structure** for GUI-backend interaction.
- Assisted in implementing several DSA-based features in both backend and frontend logic.
- Helped validate algorithm correctness through manual and automated testing.
- Contributed to refining user workflows, adding buttons, menus, and interaction handlers.
- Assisted in writing documentation and preparing diagrams used in the final report and presentation.
- Participated in debugging integration issues and improving modularity.

## Appendix



### Smart E-Commerce System

**Category: Audio**

ID	Name	Price	Stock
1	JBL Charge 5 Speaker	₹13999.00	60
2	Noise Evolve 3	₹5999.00	80
3	Sony WH-1000XM5	₹29999.00	40
4	Jabra Elite 8 Active	₹15999.00	45
5	Sennheiser Momentum 4	₹29999.00	30
6	Beats Studio Pro	₹29999.00	30
7	Sony XB13 Portable Speaker	₹4499.00	65
8	Bose QuietComfort Ultra	₹32999.00	35
9	Apple AirPods Pro 2	₹24999.00	80
10	Sony WF-1000XM5	₹19999.00	70
11	boAt Rockerz 550	₹2999.00	90
12	Zebronics Soundbar Z900	₹8999.00	50
13	Philips TAH8506BK	₹12999.00	50
14	Logitech Z407 Speaker	₹9999.00	40
15	Anker Soundcore Life Q30	₹8999.00	75
16	Bose SoundLink Revolve+	₹21999.00	40
17	Realme Buds Air 5	₹3999.00	100
18	Skullcandy Hesh ANC	₹7999.00	65
19	Marshall Emberton II	₹14999.00	45
20	JBL Tune 760NC	₹7999.00	55

**View Cart (0)**

**Recommendations**

Quantity: 1 **Add to Cart**

**Shopping Cart**

**Your Shopping Cart**

Product	Quantity	Price	Subtotal
JBL Audio Cable	1	₹799.00	₹799.00
Godrej Door Lock	1	₹4999.00	₹4999.00

**Total: ₹5798.00**

**Remove Selected** **Close** **Checkout**

**Apple Lightning Cable** ₹1499.00

**Product Recommendations**

**Recommendations for: Apple Lightning Cable**

Product Name	Price
Apple iPhone 15	₹79999.00
Samsung Galaxy S24	₹74999.00
Google Pixel 9 Pro	₹79999.00
OnePlus 12	₹64999.00
Xiaomi 14 Pro	₹54999.00

### Smart E-Commerce System

**Sorted Results**

ID	Name	Price	Stock	Category
1	ASUS ROG Ally	₹69999.00	25	Gaming
2	Acer Nitro 5 Laptop	₹84999.00	30	Gaming
3	Alienware Aurora R16	₹189999.00	10	Gaming
4	Asus TUF Gaming Monitor 27"	₹21999.00	40	Gaming
5	BenQ Zowie XL2546K	₹49999.00	20	Gaming
6	Corsair K70 RGB MK.2	₹11999.00	35	Gaming
7	Cosmic Byte GS410 Headset	₹1999.00	60	Gaming
8	Elgato Stream Deck	₹11999.00	40	Gaming
9	HyperX Cloud Alpha	₹8499.00	60	Gaming
10	Logitech G Pro Wireless Mouse	₹9999.00	50	Gaming
11	Logitech G923 Racing Wheel	₹34999.00	25	Gaming
12	MSI Katana GF66 Laptop	₹89999.00	25	Gaming
13	Microsoft Xbox Series X	₹49999.00	25	Gaming
14	Nintendo Switch OLED	₹34999.00	30	Gaming
15	Razer BlackWidow V4 Keyboard	₹13999.00	45	Gaming
16	Razer Kraken V3	₹9999.00	55	Gaming
17	Razer Seiren Mini Mic	₹4999.00	80	Gaming
18	Sony PlayStation 5	₹49999.00	20	Gaming
19	Steam Deck 512GB	₹64999.00	20	Gaming
20	Zotac RTX 4070 GPU	₹69999.00	15	Gaming

**View Cart (0)**

**Recommendations**