**Inventory Management System:**   
Inventory Management System Built with React JS, Node JS, Express JS, MongoDB, and Tailwind CSS. The application manages products, purchases, sales, and stores for a user. The MVC architecture is evident in the separation of concerns across different modules: product, purchase, sales, and store.

Notable Patterns or Practices Used

It follows a basic MVC structure, separating concerns to enhance maintainability and scalability. Each module (product, purchase, sales, store) encapsulates its own logic, promoting code organization.

Database Schema Design

The application uses MongoDB as the database. Below is a brief overview of the database schema:

1. Products Collection

\_id: Object ID (automatically generated)

userID: ID of the user who owns the product.

storeID: ID of the store where the product belongs.

name: Name of the product

manufacturer: Manufacturer of the product

stock: Current stock quantity

description: Description of the product

createdAt: Timestamp of when the product was created.

updatedAt: Timestamp of when the product was last updated.

\_\_v: Version key for handling updates

2. Purchases Collection

\_id: Object ID (automatically generated)

userID: ID of the user who made the purchase.

storeID: ID of the store where the purchase was made.

productID: ID of the product purchased.

quantityPurchased: Quantity of the product purchased.

purchaseDate: Date of the purchase

totalPurchaseAmount: Total amount spent on the purchase.

createdAt: Timestamp of when the purchase was created.

updatedAt: Timestamp of when the purchase was last updated.

\_\_v: Version key for handling updates

3. Sales Collection

\_id: Object ID (automatically generated)

userID: ID of the user who made the sale.

productID: ID of the product sold.

storeID: ID of the store where the sale occurred.

stockSold: Quantity of the product sold.

saleDate: Date of the sale

totalSaleAmount: Total amount earned from the sale.

createdAt: Timestamp of when the sale was created.

updatedAt: Timestamp of when the sale was last updated.

\_\_v: Version key for handling updates

4. Stores Collection

\_id: Object ID (automatically generated)

userID: ID of the user who owns the store.

name: Name of the store

category: Category of the store

address: Address of the store

city: City where the store is located.

createdAt: Timestamp of when the store was created.

updatedAt: Timestamp of when the store was last updated.

\_\_v: Version key for handling updates

5. Users Collection

\_id: Object ID (automatically generated)

firstName: First name of the user

lastName: Last name of the user

email: Email address of the user

password: Encrypted password of the user

phoneNumber: Phone number of the user

imageUrl: URL of the user's profile image

\_\_v: Version key for handling updates

API Design

The application exposes several RESTful APIs for managing products, purchases, sales, and stores. The main endpoints are as follows:

1. **Product Module**

Purpose: Manages the creation, retrieval, update, and deletion of products.

Controller: Handles HTTP requests related to products.

Add Product (/api/product/add): Adds a new product to the database.

Get All Products (/api/product/get/:userId): Retrieves all products associated with a specific user.

Delete Selected Product (/api/product/delete/:id): Deletes a selected product based on its ID.

Update Selected Product (/api/product/update): Updates information for a selected product.

Search Product (/api/product/search): Searches for products based on a provided search term.

2. **Purchase Module**

Purpose: Handles the recording and retrieval of purchase transactions.

Controller: Manages HTTP requests related to purchases.

Add Purchase (/api/purchase/add): Records a new purchase transaction.

Get All Purchases (/api/purchase/get/:userId): Retrieves all purchase transactions associated with a specific user.

Get Total Purchase Amount (/api/purchase/get/:userId/totalpurchaseamount): Calculates and returns the total purchase amount for a user.

3. **Sales Module**

Purpose: Manages the recording and retrieval of sales transactions.

Controller: Handles HTTP requests related to sales.

Add Sales (/api/sales/add): Records a new sales transaction.

Get All Sales (/api/sales/get/:userId): Retrieves all sales transactions associated with a specific user.

Get Monthly Sales (/api/sales/getmonthly): Retrieves sales data for the current month.

Get Total Sales Amount (/api/sales/get/:userId/totalsaleamount): Calculates and returns the total sales amount for a user.

4. **Store Module**

Purpose: Manages the creation, retrieval, update, and deletion of stores.

Controller: Handles HTTP requests related to stores.

Add Store (/api/store/add): Adds a new store to the database.

Get All Stores (/api/store/get/:userId): Retrieves all stores associated with a specific user.

Update Selected Store (/api/store/update): Updates information for a selected store.

Delete Selected Store (/api/store/delete/:id): Deletes a selected store based on its ID.

**Workflow Overview**

**Products**: Users can add, view, update, and delete products. Searching for products based on a search term is also supported.

**Purchases**: Users can record purchases, view their purchase history, and calculate the total amount spent on purchases.

**Sales**: Users can record sales transactions, view their sales history, get monthly sales data, and calculate the total amount earned from sales.

**Stores**: Users can add, view, update, and delete stores. Stores are associated with products and purchases/sales.