

# StockSync

## Description

**StockSync - Inventory Management System** is a comprehensive MERN stack-based solution designed to streamline inventory tracking, supplier management, order processing, and sales monitoring for businesses. It enables seamless stock management by reducing manual errors and improving operational efficiency. The system supports real-time stock updates, user authentication with role-based access, and automated order tracking, ensuring smooth and secure business operations. Admins have access to all functionalities, while regular users can interact with essential features like the shop page.

The platform also offers intelligent alerts for low-stock levels and order status changes, along with a robust dashboard that delivers detailed insights into sales, purchases, and inventory performance. By integrating modern web technologies, StockSync simplifies inventory workflows, enhances supply chain transparency, and ultimately drives higher productivity and better decision-making for businesses of all sizes.

## Purpose

The purpose of StockSync is to provide an easy-to-use and automated inventory management system that helps businesses manage their stock, process orders, and coordinate with suppliers efficiently. The system supports two roles: Admin, who can access all features like dashboard, inventory, suppliers, and analytics; and User, who can view products on the shop page and place orders. These orders are then sent to the admin for processing. StockSync aims to:

- **Reduce Manual Errors:** Track inventory automatically to avoid mistakes.
- **Improve Business Efficiency:** Make order management, supplier tracking, and sales handling simpler.
- **Support Better Decisions:** Show real-time data with easy-to-read dashboards.
- **Ensure Secure Access:** Use role-based login and keep data safe.
- **Manage Stock Better:** Send alerts for low stock and order updates.

With modern web technologies, StockSync helps businesses keep their inventory accurate, prevent overstocking or running out of stock, and improve overall workflow for both users and admins.

## Getting Started

### Prerequisites

Before setting up and running the application, ensure the following software and tools are installed on your system:

- **Node.js and npm**  
Download and install from Node.js official website.  
npm comes bundled with Node.js.
- **MongoDB**  
Install and setup MongoDB locally or use a cloud-hosted solution like MongoDB Atlas.
- **IDE**  
Use a development environment like Visual Studio Code for editing and debugging.

### Installation

Follow these steps to set up the project:

- **Clone the Repository**  
git clone [https://github.com/springboardmentor106/InventoryManagement\\_Infosys\\_Internship\\_Feb2025\\_Team\\_03.git](https://github.com/springboardmentor106/InventoryManagement_Infosys_Internship_Feb2025_Team_03.git)
- **Install Dependencies**  
npm install

## Running the Application

- **Start the Backend**

Navigate to the backend directory:

```
cd server
```

Run the backend server(ensure the backend runs on a specific port, e.g., 5000):

```
npm run dev
```

- **Start the Backend**

Navigate to the frontend directory:

```
cd client
```

Run the development server(ensure the frontend runs on a specific port, e.g., 5173):

```
npm run dev
```

## Accessing the Application

- **Frontend:** Visit <http://localhost:5173> (or the port specified during the setup).
- **Backend:** Accessible at <http://localhost:5000> for testing endpoints.

## Environmental Setup for StockSync

To run the project seamlessly, you need to set up environment variables for both the backend and frontend. Create a .env file in the root directory of the backend project and add the required configurations as shown below.

### Required Environment Variables

Variable Name	Description	Example Value
PORT	Port number on which the backend server runs	5000
DB_CONN_STRING	MongoDB connection string for database access	mongodb://127.0.0.1:27017/team3
JWT_SECRET	Secret key used for signing JSON Web Tokens (JWT)	abc123@
CLIENT_URL	URL of the frontend application	http://localhost:5173
MY_EMAIL	Email ID used for authentication or notifications	iimtthree@gmail.com
MY_MAIL_PASSWORD	App-specific email password (should be kept secure)	***** (Never expose this publicly!)

### Example .env Configuration

Here's an example .env file for your project:

```
PORT=5000
```

```
DB_CONN_STRING=mongodb://127.0.0.1:27017/practice_mern
```

```
JWT_SECRET=abc123@
```

```
CLIENT_URL=http://localhost:5173
```

```
MY_EMAIL=your-email@example.com
```

```
MY_MAIL_PASSWORD=password
```

### Steps to Set Up the Environment

- Create a .env file in the root of your backend directory.
- Copy the example configuration above and paste it into the .env file.
- Replace placeholder values (like your-email@example.com and password) with your actual configuration.

## Additional Notes

- For cloud-hosted MongoDB (e.g., MongoDB Atlas), replace `mongodb://localhost:27017/` with your cluster's connection string.
- Keep the `.env` file secure and do not commit it to version control. Add `.env` to your `.gitignore` file.

## Folder Structure

A well-organized folder structure helps maintain clarity and efficiency in development. Below is a suggested folder structure for your Right Resource Fit project, leveraging the MERN stack.

```
StockSync/
├── client/
│   ├── node_modules/
│   ├── public/
│   ├── src/
│   │   ├── api/
│   │   ├── assets/
│   │   ├── components/
│   │   ├── context/
│   │   ├── pages/
│   │   │   ├── admin/
│   │   │   ├── auth/
│   │   │   └── shop/
│   │   ├── routes/
│   │   ├── utils/
│   │   ├── App.jsx
│   │   ├── appwrite.config.js
│   │   ├── index.css
│   │   └── main.jsx
│   ├── .env
│   ├── .gitignore
│   ├── eslint.config.js
│   ├── index.html
│   ├── package-lock.json
│   ├── package.json
│   ├── README.md
│   └── vite.config.js
└── server/
    ├── controllers/
    ├── models/
    ├── node_modules/
    ├── public/
    ├── routes/
    ├── utils/
    ├── .env
    ├── .gitignore
    ├── package-lock.json
    ├── package.json
    └── server.js
```

## Key Features

- **User Authentication & Management:** Secure registration and login using JWT-based authentication, with password reset functionality for both Admin and User roles.
- **Role-Based Access Control:** Admin has full access to all system modules, while Users have limited access (e.g., Shop page and order placement).
- **Inventory Management:** Admin can add, update, delete, and track products in real-time, with low-stock alerts to prevent stockouts.
- **Supplier Management:** Admin can maintain supplier records, monitor deliveries, and manage supplier-related data.
- **Order Processing & Tracking:** Admin can view and manage all user orders with status updates such as Confirmed, Delivered, Returned, and Delayed.
- **Shop & Order Placement:** Users can browse available products on the Shop page and place orders, which are reflected to the admin for further processing.
- **Sales Management:** Admin can record sales, track category-wise turnover, and automatically update stock levels.
- **Dashboard & Analytics:** Real-time graphical insights on sales, purchases, inventory levels, and performance metrics.
- **Notifications & Alerts:** Automated alerts for low stock, delayed deliveries, and order status changes.

## Project Components

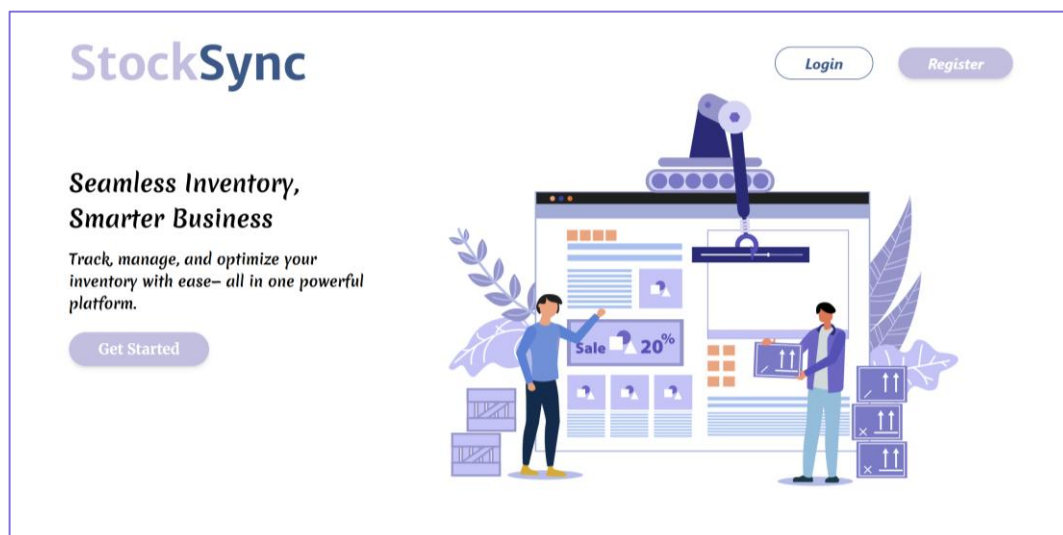
Here's a list of key components in the StockSync project, along with brief descriptions of their functionality:

### Landing Page Component

Serves as the main entry point for the StockSync application, showcasing the platform's value proposition and directing users toward account creation or login.

#### **Key Features:**

- **Hero Section:** Features the tagline "Seamless Inventory, Smarter Business" with supporting text explaining the platform's purpose.
- **Visual Illustration:** Includes an engaging illustration demonstrating inventory management with interactive elements.
- **Call-to-Action Buttons:** Prominently displays "Get Started" button to drive user conversion.
- **Authentication Access:** Provides clear pathways to both login and registration functions.
- **Brand Establishment:** Presents the StockSync logo and color scheme to build brand recognition.

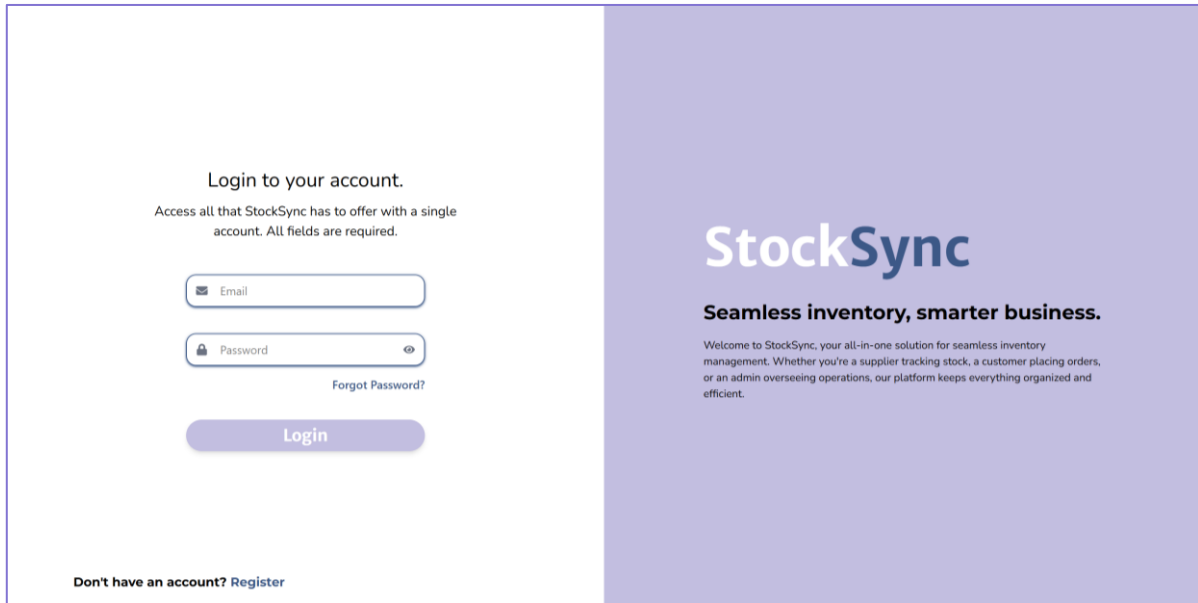


## Login Component

Manages the user login process, handling inputs for email and password.

### Key Features:

- Password Visibility Toggle: Allows users to show/hide their password for easy input.
- 'Remember Me' Functionality: Saves user login credentials (using cookies or localStorage) for subsequent visits without needing to re-enter credentials.
- Form Validation: Ensures that the user provides valid credentials before submission.
- Error Handling: Displays error messages in case of incorrect credentials.



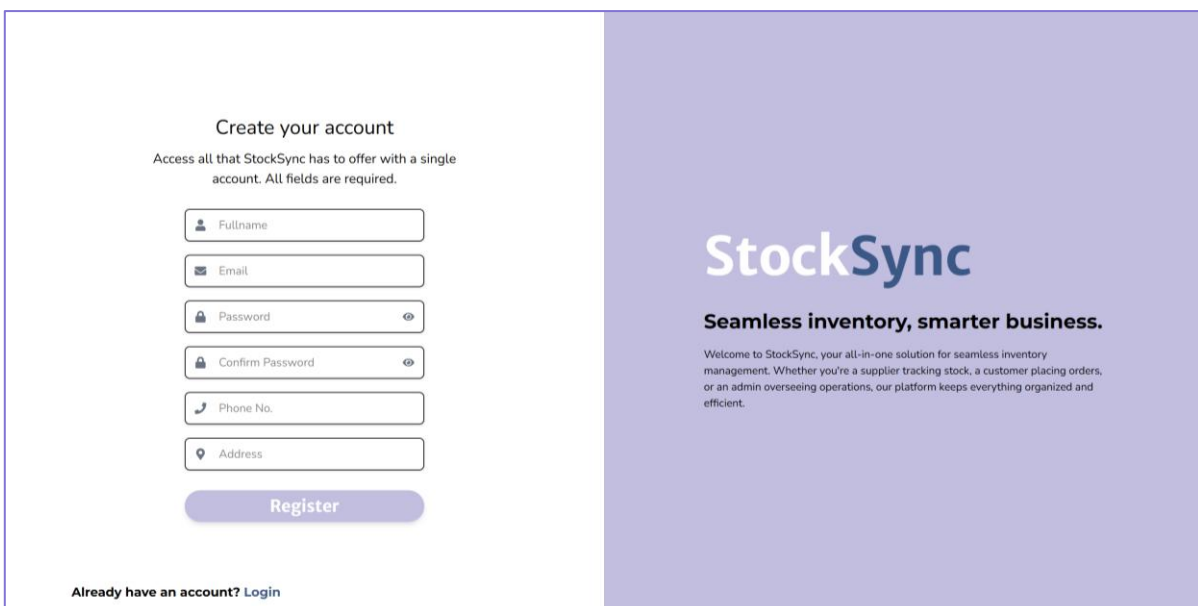
The login component features a clean, modern design. On the left, a white card contains the login form. It starts with the heading "Login to your account." followed by a subtext: "Access all that StockSync has to offer with a single account. All fields are required." Below this are two input fields: "Email" with an envelope icon and "Password" with a lock icon and a toggle eye icon. A "Forgot Password?" link is positioned below the password field. A prominent purple "Login" button is at the bottom of the form. At the very bottom of the card, a link reads "Don't have an account? Register". The right side of the component has a solid purple background. It features the "StockSync" logo in white, the tagline "Seamless inventory, smarter business." in bold white text, and a paragraph of welcome text: "Welcome to StockSync, your all-in-one solution for seamless inventory management. Whether you're a supplier tracking stock, a customer placing orders, or an admin overseeing operations, our platform keeps everything organized and efficient."

## Registration Component

Facilitates new user account creation with a comprehensive form for collecting user information.

### Key Features:

- Multiple Field Collection: Gathers fullname, email, password, phone number, and address.
- Password Strength Verification: Ensures users create secure passwords with confirmation field.
- Field Validation: Validates all input fields to ensure proper formatting and completeness.
- Account Linking: Provides option to navigate to login for users who already have accounts.



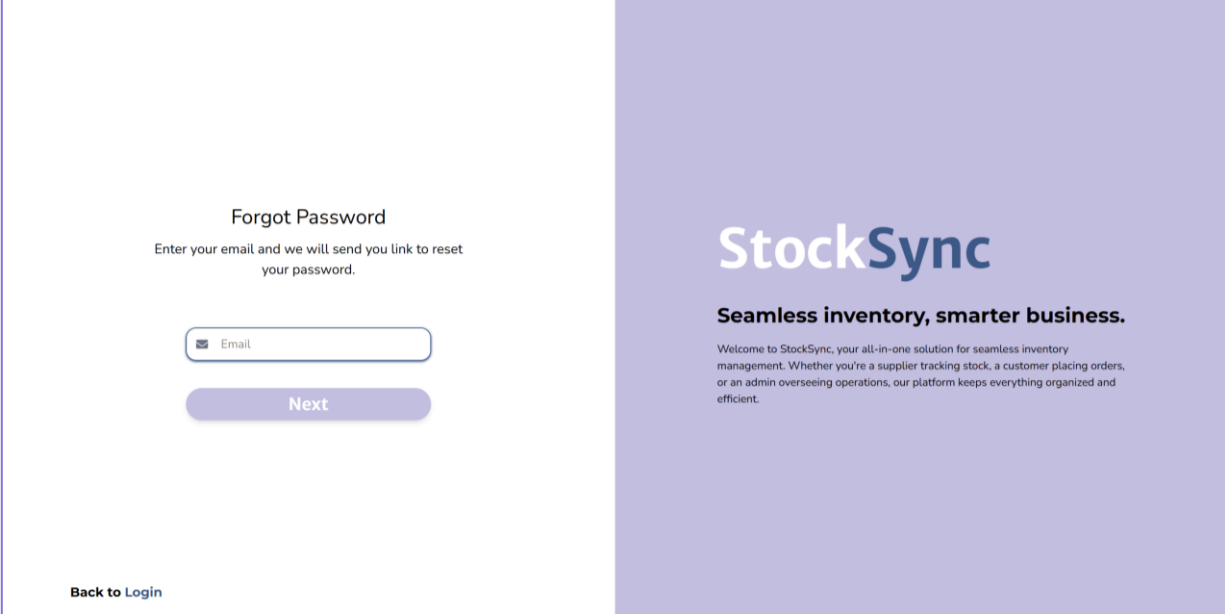
The registration component mirrors the login component's layout. The left white card is titled "Create your account" with the same subtext: "Access all that StockSync has to offer with a single account. All fields are required." The form includes six input fields: "Fullname" (with a person icon), "Email" (with an envelope icon), "Password" (with a lock icon and toggle eye), "Confirm Password" (with a lock icon and toggle eye), "Phone No." (with a phone icon), and "Address" (with a location pin icon). A purple "Register" button is at the bottom of the form. A link at the bottom of the card reads "Already have an account? Login". The right purple background section contains the "StockSync" logo, the tagline "Seamless inventory, smarter business.", and the same welcome text as the login component.

## Forgot Password Component

Manages the initial step of the password recovery process for users who cannot access their accounts.

### Key Features:

- Email Submission: Collects user's registered email address for verification.
- Recovery Link Generation: Triggers system to send password reset instructions to user's email.
- User Guidance: Provides clear instructions on the password recovery process.
- Navigation Options: Includes link to return to the login page if recovery is not needed.



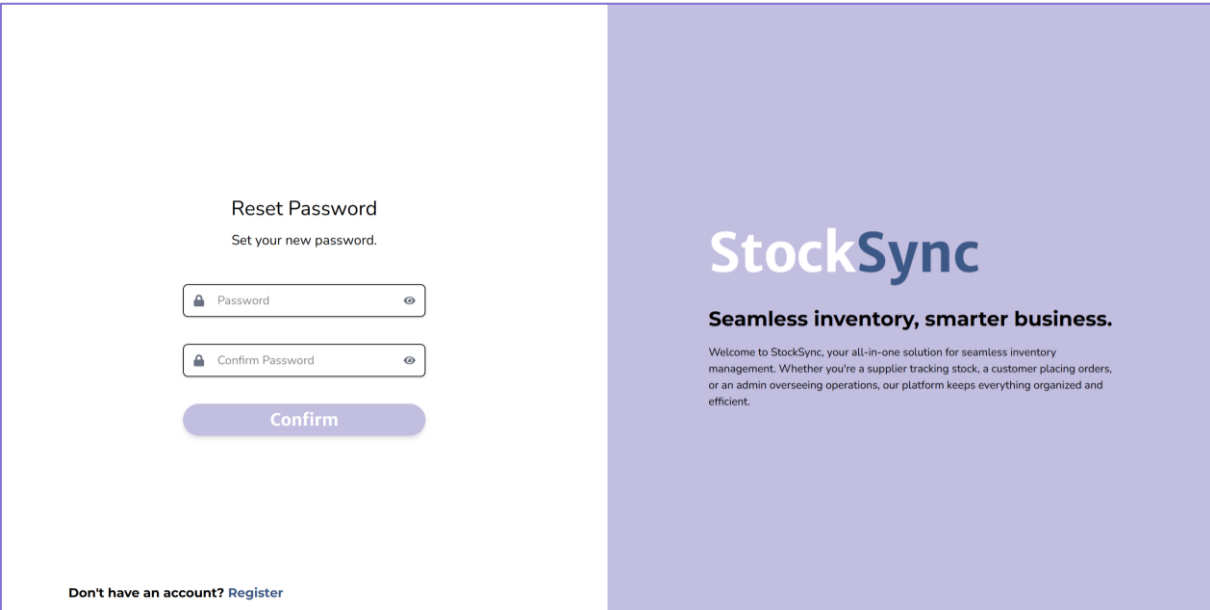
The image shows a web form for the 'Forgot Password' component. The form is split into two main sections. The left section is white and contains the following elements: a title 'Forgot Password', a subtitle 'Enter your email and we will send you link to reset your password.', an email input field with a placeholder 'Email', a 'Next' button, and a 'Back to Login' link at the bottom left. The right section is a solid purple color and contains the 'StockSync' logo, the tagline 'Seamless inventory, smarter business.', and a welcome message: 'Welcome to StockSync, your all-in-one solution for seamless inventory management. Whether you're a supplier tracking stock, a customer placing orders, or an admin overseeing operations, our platform keeps everything organized and efficient.'

## Reset Password Component

Handles the final step of the password recovery process, allowing users to create new credentials.

### Key Features:

- New Password Creation: Provides fields for entering and confirming new password.
- Security Validation: Ensures token authenticity before allowing password changes.
- Password Requirements: Enforces password strength and matching confirmation.
- Completion Handling: Processes the password change and redirects users appropriately.



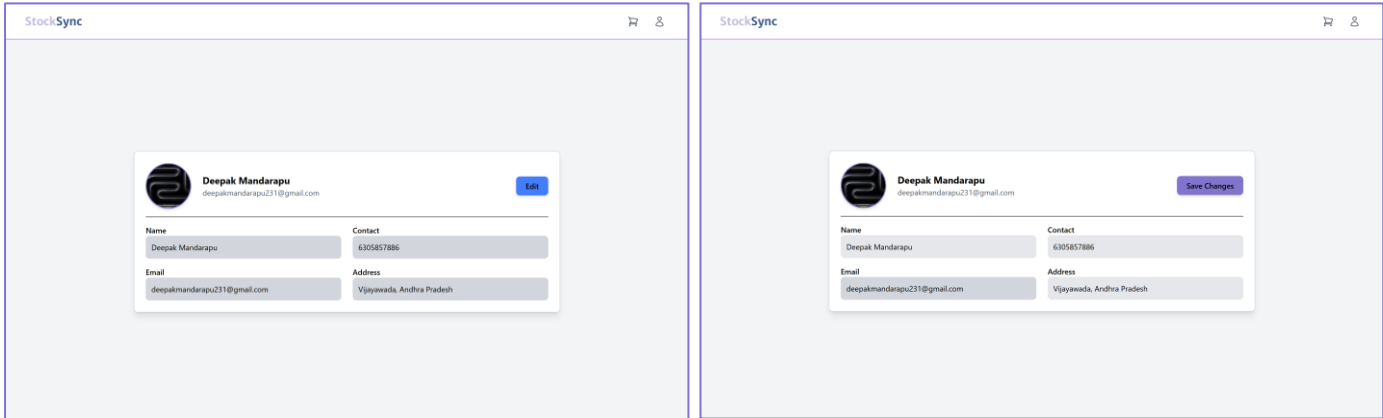
The image shows a web form for the 'Reset Password' component. The form is split into two main sections. The left section is white and contains the following elements: a title 'Reset Password', a subtitle 'Set your new password.', two password input fields labeled 'Password' and 'Confirm Password', a 'Confirm' button, and a link 'Don't have an account? Register' at the bottom left. The right section is a solid purple color and contains the 'StockSync' logo, the tagline 'Seamless inventory, smarter business.', and the same welcome message as the previous form: 'Welcome to StockSync, your all-in-one solution for seamless inventory management. Whether you're a supplier tracking stock, a customer placing orders, or an admin overseeing operations, our platform keeps everything organized and efficient.'

Profile Component

Allows users to view and update their profile details efficiently.

Key Features:

- User Information Display: Shows name, email, contact, and address in a structured format.
- Edit Mode: Users can update their details by clicking the "Edit" button.
- Save Changes: After modifications, users can save their updates using the "Save Changes" button.
- User Avatar: Displays a profile picture for easy identification.

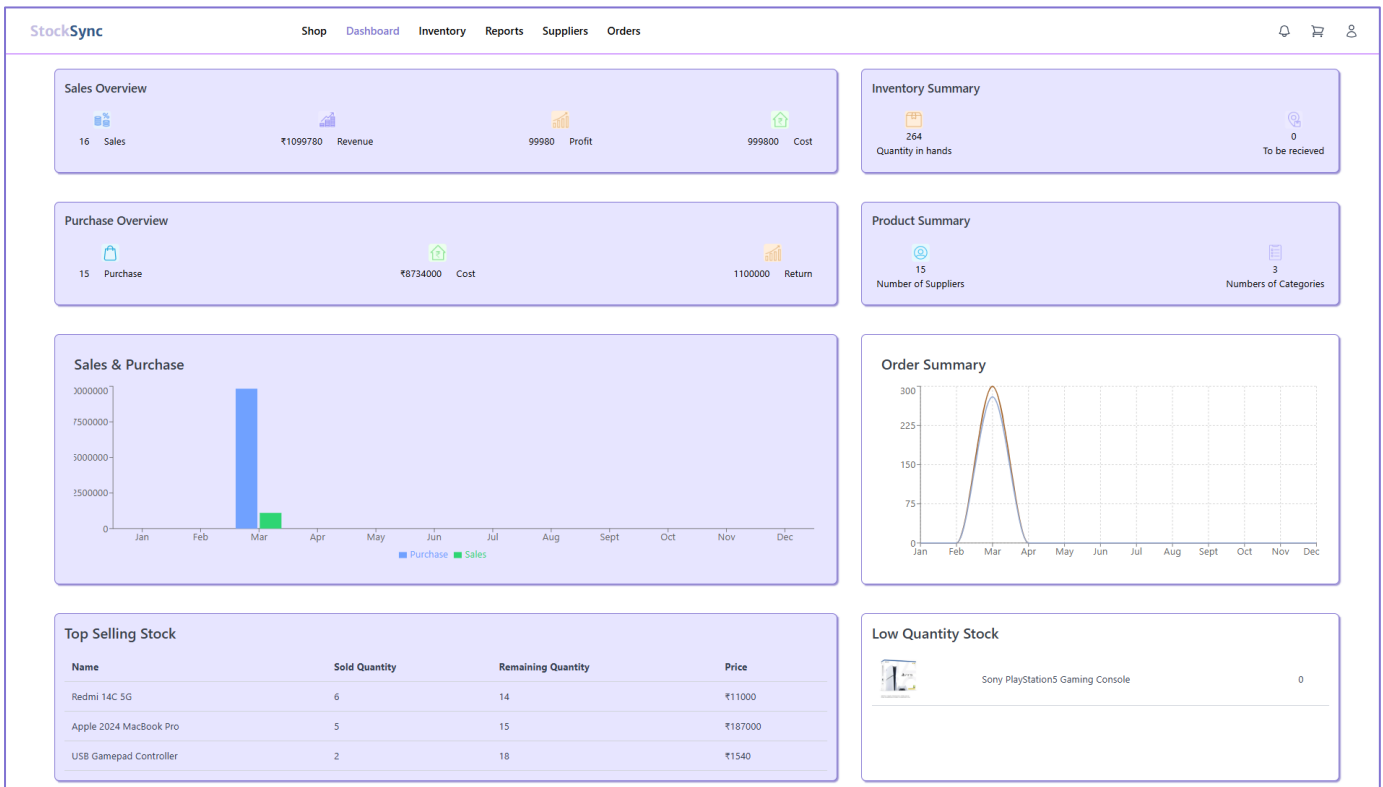


Dashboard Component

Gives a quick view of sales, purchases, inventory, and order summaries to help manage stock efficiently.

Key Features:

- Sales & Purchase Overview: Shows total sales, revenue, profit, and purchase data.
- Order Summary: Tracks order trends across each month.
- Inventory & Product Summary: Displays current stock, supplier count, and categories.
- Visual Insights: Includes charts and graphs for easy analysis.
- Top & Low Stock: Highlights best-selling and low-quantity items.



## Inventory Component

Manages and tracks product stock levels, categories, and availability efficiently.

### Key Features:

- Overall Inventory Overview: Displays total product count, revenue, top-selling items, and low-stock alerts.
- Product Details: Lists products along with cost, quantity, threshold values, expiry dates, and availability status.
- Pagination: Users can navigate through products using "Prev" and "Next" buttons.
- Search Functionality: Allows quick lookup of products in inventory.

The screenshot shows the 'Inventory' tab in the StockSync application. At the top, there's a navigation bar with 'Shop', 'Dashboard', 'Inventory' (active), 'Reports', 'Suppliers', and 'Orders'. Below the navigation bar, there's a summary section titled 'Overall Inventory' with five cards: 'Categories' (3), 'Total Products' (15), '9607400' (Revenue), 'Top Selling' (5), '1099780' (Revenue), and 'Low Stocks' (1). Below this is a 'Products' table with columns: Product, Cost, Quantity, threshold value, Expiry Date, and Availability. The table lists 10 products, including Redmi 14C 5G, FRONTECH 22 Inch HD LED Monitor, Samsung Galaxy S25 5G, USB Gamepad Controller, Apple 2024 MacBook Pro, ASUS Marshmallow Md100 USB Mouse, ZEBRONICS igloo 1, 2.0 USB Computer Speakers, CORSAIR K57 RGB Wireless Gaming Keyboard, OnePlus Nord CE4, and IQOO 13 5G. Each product has its cost, quantity, threshold value, expiry date, and availability status (all are 'In-Stock'). At the bottom of the table are 'Prev' and 'Next' buttons.

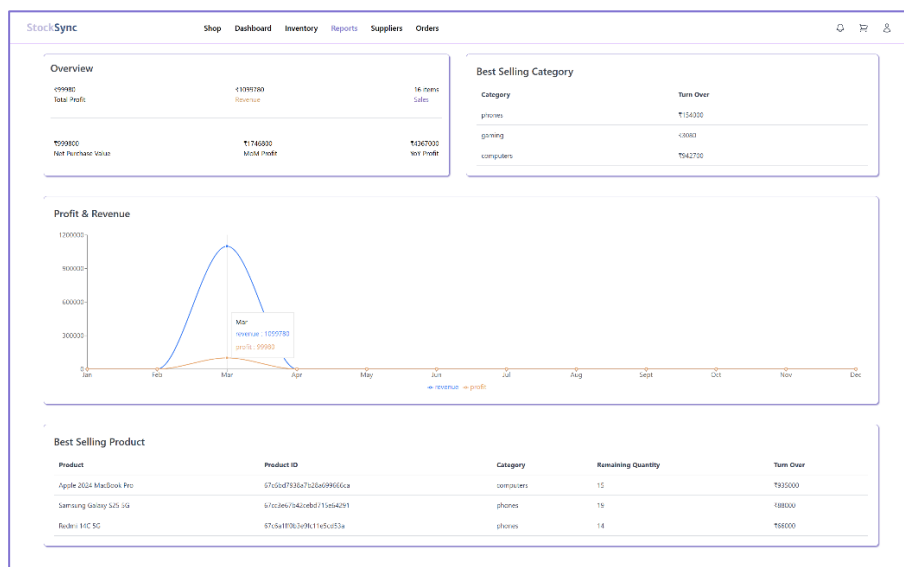
Product	Cost	Quantity	threshold value	Expiry Date	Availability
Redmi 14C 5G	10000	14	10	2027-03-04	In-Stock
FRONTECH 22 Inch HD LED Monitor	3500	18	10	2027-03-08	In-Stock
Samsung Galaxy S25 5G	80000	19	10	2027-03-08	In-Stock
USB Gamepad Controller	1400	18	10	2027-03-08	In-Stock
Apple 2024 MacBook Pro	170000	15	10	2027-03-04	In-Stock
ASUS Marshmallow Md100 USB Mouse	1000	20	10	2027-03-08	In-Stock
ZEBRONICS igloo 1, 2.0 USB Computer Speakers	400	20	10	2027-03-08	In-Stock
CORSAIR K57 RGB Wireless Gaming Keyboard	10000	20	10	2027-03-08	In-Stock
OnePlus Nord CE4	22000	20	10	2027-03-08	In-Stock
IQOO 13 5G	60000	20	10	2027-03-08	In-Stock

## Reports Component

Provides detailed insights into profit, revenue, best-selling products, and category-wise performance.

### Key Features:

- Overview Dashboard – Displays key metrics like Total Profit, Revenue, Net Purchase Value, MoM and YoY Profit, and Total Sales Items.
- Best Selling Category – Highlights top-performing categories based on turnover.
- Profit & Revenue Chart – Visual representation of monthly revenue and profit trends.
- Best Selling Products – Lists top-selling products with details like Product ID, Category, Remaining Quantity, and Turnover.





## Suppliers Component

Manages supplier details, allowing users to view, add, and update supplier information efficiently.

### Key Features:

- **Supplier Overview:** Displays a list of suppliers along with their product, contact details, email, type (Premium/Regular), and on-the-way orders.
- **Search & Navigation:** Allows users to search suppliers and navigate through the list.
- **Add Supplier Form:** Enables users to add new suppliers by entering supplier name, product, category, contact, email, and type (Taking/Not taking returns).
- **Supplier Management:** Provides options to submit or discard new supplier entries with a user-friendly interface.

StockSync

ShopDashboardInventoryReportsSuppliersOrders

Suppliers

Add Supplier

Supplier	Product	Contact	Email	Type	On the way
Alexander	OnePlus 13	9963365944	alexander@gmail.com	taking return	0
Benjamin	Samsung Galaxy S25 5G	9968365944	benjamin@gmail.com	taking return	0
Caleb	FRONTECH 22 Inch HD LED Monitor	9969369758	caleb@gmail.com	taking return	0
Oliver	ZEBRONICS Igloo 1, 2.0 USB Computer Speakers	9969369746	oliver@gmail.com	taking return	0
Samuel	ASUS Marshmallow Md100 USB Mouse	9969369748	samuel@gmail.com	taking return	0
Jameson	CORSAIR K57 RGB Wireless Gaming Keyboard	9969365745	jameson@gmail.com	taking return	0
Daniel	OnePlus Nord CE4	9969365744	daniel@gmail.com	taking return	0
Nathaniel	IQOO 13 5G	9969365944	nathaniel@gmail.com	taking return	0
Smith	PSS Storage Bag for PSS Slim	8963365935	smith@gmail.com	taking return	0
James	U & I Entertainment Black Myth: Wukong PSS-CD	8966665235	james@gmail.com	taking return	0

PrevNext

New Supplier

Supplier Name

Enter supplier name

Product

Enter product name

Category

Enter product category

Contact

Enter contact no.

Email

Enter email

Type

Taking returns

Not taking returns

Discard

Submit

## Orders Component

Manages the ordering process, allowing users to place, track, and manage orders efficiently.

### Key Features:

- **Order Overview:** Displays total orders, received, returned, and ongoing deliveries.
- **Order Listing:** Shows order details, including product, order value, quantity, order ID, expected delivery, and status.
- **New Order Form:** Enables users to create new orders by entering product details.
- **Image Upload:** Allows users to upload product images for better record-keeping.
- **Order Submission:** Provides options to submit or discard an order with form validation system.

StockSync

ShopDashboardInventoryReportsSuppliersOrders

Overall Orders

Total Orders

15

Total

Total Received

14

Total

8/34200

Received

Total Returned

1

Total

1100000

Cost

Being Delivered

0

Total

0

Cost

Orders

Place Order

Product	Order Value	Quantity	Order ID	Expected Delivery	Status
OnePlus 13	1400000	20	67cc3dcd842cebd715e64287	2025-03-09	Delivered
Samsung Galaxy S25 5G	1600000	20	67cc3e67b42cebd715e6428e	2025-03-09	Delivered
FRONTECH 22 Inch HD LED Monitor	70000	20	67cc10b7b42cebd715e642b8	2025-03-09	Delivered
ZEBRONICS Igloo 1, 2.0 USB Computer Speakers	8000	20	67cc3f9ab42cebd715e642aa	2025-03-09	Delivered
ASUS Marshmallow Md100 USB Mouse	20000	20	67cc455b42cebd715e642b1	2025-03-09	Delivered
CORSAIR K57 RGB Wireless Gaming Keyboard	200000	20	67cc3d99b42cebd715e642a3	2025-03-09	Delivered
OnePlus Nord CE4	440000	20	67cc3d77b42cebd715e6429e	2025-03-09	Delivered
IQOO 13 5G	1200000	20	67cc3ec9b42cebd715e64295	2025-03-09	Delivered
PSS Storage Bag for PSS Slim	44000	20	67cc3894b7054dbbe65c11c5	2025-03-09	Delivered
U & I Entertainment Black Myth: Wukong PSS-CD	110000	20	67cc379057054dbbe65c11be	2025-03-09	Delivered

PrevNext

New Order

Upload Product Image

Browse image

Product Name

Enter Product name

Supplier ID

Enter product supplier ID

Category

Enter product category

Quantity

0

Unit

Eg: pcs

Buying Price

0

Date of Delivery

dd-mm-yyyy

Discard

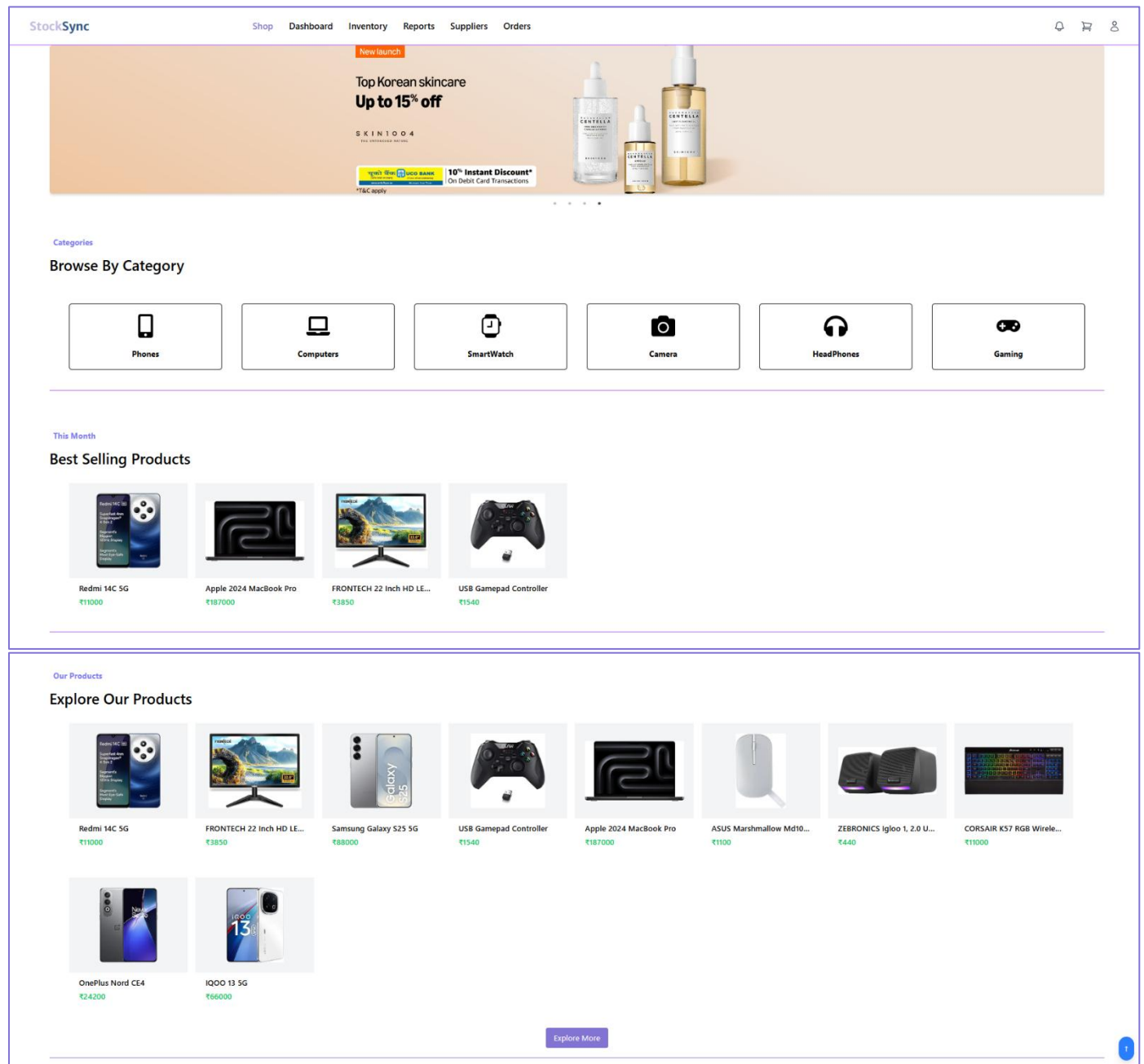
Submit

## Shop Page Component

Lets users explore and buy products by browsing categories or checking best-selling items.

### Key Features:

- Category Browsing: Easy navigation through categories like Phones, Computers, Cameras, etc.
- Best Sellers: Highlights top-selling products of the month.
- Product Exploration: Displays a wide range of available products with prices.
- Visual Display: Includes product images and clean layout for better user experience.
- Explore More: Button to view additional products.



## Product Page Component:

Displays detailed information about a selected item with options to purchase.

### Key Features:

- Product Info: Shows product name, price, and image.
- Add to Cart: Button to quickly add the item to the cart.
- Offers Section: Lists special deals like cashback, EMI options, and free delivery.

- Clean Layout: Simple design for easy navigation and readability.

StockSync

Shop

Dashboard

Inventory

Reports

Suppliers

Orders

Redmi 14C 5G

Superfast 4nm Snapdragon® 4 Gen 2

Segment's Biggest 120Hz Display

Segment's Most Eye-Safe Display

Redmi 14C 5G

₹11000

Add to Cart

Offers:

• 10% Cashback on UPI

• EMI Options Available

• Free Delivery for Prime Members

**Cart Page Component:**

Displays all selected products with options to update quantity, remove items, and proceed to checkout.

**Key Features:**

- Product List: Shows product image, name, price, quantity selector, and subtotal.
- Remove Option: Allows users to delete items from the cart with a single click.
- Price Summary: Clearly lists subtotal, shipping fee, and total cost.
- Checkout Button: Directs users to the payment and order confirmation page.
- Clean UI: Organized layout for a smooth and user-friendly experience.

StockSync

Shop

Dashboard

Inventory

Reports

Suppliers

Orders

Product	Price	Quantity	Subtotal	Action
<div><div><div></div></div><div>Redmi 14C 5G</div></div>	₹11000	<div>2</div>	₹22000.00	<div>Remove</div>
<div><div><div></div></div><div>Apple 2024 MacBook Pro</div></div>	₹187000	<div>1</div>	₹187000.00	<div>Remove</div>

Cart Total

Subtotal:

Shipping Fee:

Total:

₹209000

₹0

₹209000

Proceed to Checkout

## Checkout Page Component:

Captures user billing details and displays the final list of products before order confirmation.

### Key Features:

- **Billing Form:** Collects customer information – name, address, phone number, and email.
- **Order Summary:** Lists selected products with names, prices, and total cost.
- **Cost Breakdown:** Displays subtotal, shipping fee, and grand total.
- **Place Order Button:** Finalizes the purchase and submits the order.
- **Minimal & Clean Design:** Ensures user-friendly experience during checkout.

StockSync

ShopDashboardInventoryReportsSuppliersOrders


Billing Details

First Name

Address


Phone Number

Email Address



Redmi 14C 5G

₹22000.00



Apple 2024 MacBook Pro

₹187000.00

Subtotal:

₹209000

Shipping Fee:

₹0

Total:

₹209000

Place Order

## API Components

### User Authentication & Management

Base Url: /api/users/

#### 1. Register a User

- **Endpoint:** POST /register
- **Description:** Registers a new user.

#### Request Body:

```
{
  "fullname": "John Doe",
  "email": "johndoe@example.com",
  "password": "SecurePass123",
  "phoneNumber": "1234567890",
  "address": "123 Street, City, Country"
}
```

#### Response:

```
{
  "message": "User created"
}
```

#### 2. Login

- **Endpoint:** POST /login
- **Description:** Authenticates a user and returns a token.

**Request Body:**

```
{
  "email": "johndoe@example.com",
  "password": "SecurePass123"
}
```

**Response:**

```
{
  "success": true,
  "user": {
    "fullname": "John Doe",
    "email": "johndoe@example.com",
    "phoneNumber": "1234567890",
    "address": "123 Street, City, Country"
  }
}
```

**3. Forgot Password**

**Endpoint:** POST /forgot-password

**Description:** Sends a password reset email.

**Request Body:**

```
{
  "email": "johndoe@example.com"
}
```

**Response:**

```
{
  "message": "Email Sent Successfully!"
}
```

**4. Reset Password**

- **Endpoint:** POST /reset-password/:token
- **Description:** Resets the user's password.

**Request Body:**

```
{
  "password": "NewSecurePass123"
}
```

**Response:**

```
{
  "message": "Password reset successful"
}
```

**5. Edit Profile**

- **Endpoint:** POST /editProfile
- **Description:** Updates user profile (requires authentication).
- **Headers:** Authorization: Bearer <token>

**Request Body:**

```
{
  "fullname": "John Updated",
  "phoneNumber": "9876543210",
}
```

```
"address": "456 New Street, City, Country"
}
```

**Response:**

```
{
  "message": "User updated",
  "token": "new-jwt-token"
}
```

## **6. Get User Details**

- **Endpoint:** GET /getUser
- **Description:** Retrieves user details (requires authentication).
- **Headers:** Authorization: Bearer <token>

**Response:**

```
{
  "fullName": "John Doe",
  "email": "johndoe@example.com",
  "phoneNumber": "1234567890",
  "address": "123 Street, City, Country",
  "admin": false
}
```

## **User Schema**

```
{
  "fullName": "String (min 3 characters)",
  "email": "String (unique)",
  "password": "String",
  "salt": "String",
  "profilePic": "String (default: '/images/man_5-1024.webp')",
  "phoneNumber": "String",
  "address": "String",
  "resetPasswordToken": "String",
  "resetPasswordTimeout": "Date",
  "admin": "Boolean (default: false)"
}
```

# **Suppliers Management**

**Base URL:** /api/supplier/

## **1. Get Suppliers**

- **Endpoint:** GET /getSuppliers
- **Description:** Retrieves a paginated list of suppliers (Admin only).
- **Headers:** Authorization: Bearer <token>

**Query Parameters:**

- page (optional, default: 1) - The page number.
- limit (optional, default: 10) - Number of suppliers per page.

**Response:**

```
{
  "suppliers": [
```

```
{
  "name": "Supplier Name",
  "product": "Product Name",
  "category": "Category Name",
  "contact": "1234567890",
  "email": "supplier@example.com",
  "type": "Type",
  "onTheWay": 0,
  "supplierImage": "image_url",
  "supplierId": "abcd1234"
}
],
"currentPage": 1,
"totalPages": 5,
"hasNextPage": true,
"hasPrevPage": false
}
```

## **2. Add Supplier**

- **Endpoint:** POST /addSupplier
- **Description:** Adds a new supplier (Admin only).
- **Headers:** Authorization: Bearer <token>

### **Request Body:**

```
{
  "name": "Supplier Name",
  "product": "Product Name",
  "category": "Category Name",
  "contact": "1234567890",
  "email": "supplier@example.com",
  "type": "Type",
  "supplierImage": "image_url"
}
```

### **Response:**

```
{
  "message": "Supplier added!",
  "supplier": {
    "name": "Supplier Name",
    "product": "Product Name",
    "category": "Category Name",
    "contact": "1234567890",
    "email": "supplier@example.com",
    "type": "Type",
    "supplierImage": "image_url",
    "supplierId": "abcd1234"
  }
}
```

## **Supplier Schema**

```
{
  "name": "String (required)",
  "product": "String",
  "category": "String",
  "contact": "String (required)",
  "email": "String (unique, required)",
  "type": "String (required)",
  "onTheWay": "Number (default: 0)",
  "supplierImage": "String",
  "supplierId": "String (unique, required)"
}
```

## Orders Management

**Base URL:** /api/order

### 1. Place an Order

- **Endpoint:** POST /placeOrder
- **Description:** Places a new order (Admins only)

**Request Body:**

```
{
  "name": "Product Name",
  "supplierId": "supplier1234",
  "category": "Electronics",
  "quantity": 10,
  "unit": "pcs",
  "orderValue": 5000,
  "expectedDelivery": "2025-03-20",
  "productImage": "url-to-image"
}
```

**Response:**

- `201 Created`: Order placed successfully
- `400 Bad Request`: Supplier not found or invalid data
- `500 Internal Server Error`: Something went wrong

### 2. Get Orders

- **Endpoint:** GET /getOrders
- **Description:** Fetches the details of orders (Admins only)

**Query Parameters:**

- page (optional, default: `1`)
- limit (optional, default: `10`)

**Response:**

```
{
  "orders": [ {...orderData} ],
  "currentPage": 1,
  "totalPages": 5,
  "hasNextPage": true,
  "hasPrevPage": false
}
```



### Status Codes:

- 200 OK: Returns list of orders
- 500 Internal Server Error: Issue retrieving orders

### 3. Get Overall Order Statistics

- **Endpoint:** GET /overAllOrders
- **Description:** Gives the details of all orders (Admins only)

### Response:

```
{
  "totalOrders": 100,
  "totalReceived": 80,
  "totalReturned": 5,
  "totalOnTheWay": 15,
  "totalReceivedCost": 400000,
  "totalReturnedCost": 25000,
  "totalOnTheWayCost": 75000
}
```

### Status Codes:

- 200 OK: Returns order statistics
- 500 Internal Server Error: Issue retrieving data

### Order Schema

```
{
  "name": "Product Name",
  "supplierId": "supplier1234",
  "category": "Electronics",
  "quantity": 10,
  "unit": "pcs",
  "orderValue": 5000,
  "expectedDelivery": "2025-03-20",
  "status": "Confirmed | Delivered | Returned | Delayed"
}
```

## Product Management

**Base URL:** /api/product

### 1. Get All Products

- **Endpoint:** GET /getproducts
- **Description:** Fetch all products.

### Response:

```
[
  {
    "_id": "product_id",
    "name": "Product Name",
    "category": "Category Name",
    "cost": 100,
    "sellingPrice": 120,
    "unit": "kg",
  }
]
```

```
"productImage": "image_url",
"qtySold": 50,
"qtyRemaining": 20
}
]
```

## **2. Get Top Selling Products**

- **Endpoint:** GET /getTopSellingProducts
- **Description:** Get the top 4 best-selling products.

## **3. Get Products By Category**

- **Endpoint:** GET /getTopSellingProducts
- **Query Params:** category
- **Description:** Fetch products by category.

## **4. Get Limited Products**

- **Endpoint:** GET /getLimitedProducts
- **Description:** Fetch up to 8 products.

## **5. Search Product**

- **Endpoint:** GET /searchProduct
- **Query Params:** search
- **Description:** Search for products by name or category.

## **6. Get Inventory Stats**

- **Endpoint:** GET /inventoryStats
- **Description:** Fetch inventory statistics.

## **7. Product Pagination**

- **Endpoint:** GET /productPagination
- **Query Params:** page, limit
- **Description:** Fetch paginated products.

## **8. Get Products By ID List**

- **Endpoint:** GET /getProductsByIdsList
- **Description:** Fetch multiple products by their IDs.

### **Request Body:**

```
{
  "productIds": ["id1", "id2"]
}
```

## **9. Get Single Product**

- **Endpoint:** GET /getProduct/:id
- **Description:** Fetch a single product by ID.

### **Response:**

```
{
  "_id": "product_id",
  "name": "Product Name",
```

```
"category": "Category Name",
"cost": 100,
"sellingPrice": 120,
"unit": "kg",
"productImage": "image_url",
"qtySold": 50,
"qtyRemaining": 20
}
```

### Error Handling

- 400 - Bad Request (Invalid parameters or request body)
- 500 - Internal Server Error (Database or server issues)

### Schema - Product Model:

```
{
  "name": "string",
  "category": "string",
  "cost": "number",
  "sellingPrice": "number",
  "unit": "string",
  "productImage": "string",
  "qtySold": "number",
  "qtyRemaining": "number",
  "thresholdValue": "number"
}
```

## Notification Management

**Base URL:** /api/product

### 1. Get Notifications

- **Endpoint:** GET /api/notifications/getNotifications
- **Authorization:** Admin Required

#### Request

GET /api/notifications/getNotifications HTTP/1.1

Host: yourdomain.com

Authorization: Bearer <your-token>

#### Response (Success - 200)

```
[
  {
    "_id": "66234abcd1234ef567890xyz",
    "orderId": {
      "_id": "66123abcde4567ef890xyz",
      "name": "Laptop",
      "category": "Electronics",
      "quantity": 10,
      "orderValue": 50000,
      "expectedDelivery": "2025-03-15",
      "supplierId": "6609876543210abcd"
    },
  },
]
```

```
"createdAt": "2025-03-10T12:00:00Z",
"updatedAt": "2025-03-10T12:00:00Z"
}
]
```

#### **Response (Error - 500)**

```
{
  "message": "Internal server error"
}
```

### **2. Mark Order as Delivered**

- **Endpoint:** POST /api/notifications/delivered
- **Authorization:** Admin Required

#### **Request**

POST /api/notifications/delivered HTTP/1.1

Host: yourdomain.com

Authorization: Bearer <your-token>

Content-Type: application/json

```
{
  "_id": "66234abcd1234ef567890xyz",
  "orderId": {
    "_id": "66123abcde4567ef890xyz",
    "supplierId": "6609876543210abcd"
  }
}
```

#### **Response (Success - 200)**

```
{
  "success": true
}
```

#### **Response (Error - 404)**

```
{
  "message": "Order not found"
}
```

#### **Response (Error - 500)**

```
{
  "message": "Internal server error"
}
```

### **3. Mark Order as Delayed**

- **Endpoint:** POST /api/notifications/delay
- **Authorization:** Admin Required

#### **Request**

POST /api/notifications/delay HTTP/1.1

Host: yourdomain.com

Authorization: Bearer <your-token>

Content-Type: application/json

```
{
  "orderId": {
    "_id": "66123abcde4567ef890xyz"
  }
}
```

```
}  
}  
Response (Success - 200)  
{  
  "success": true  
}  
Response (Error - 404)  
{  
  "message": "Order not found"  
}  
Response (Error - 500)  
{  
  "message": "Internal server error"  
}
```

#### **4. Mark Order as Returned**

- **Endpoint:** POST /api/notifications/returned
- **Authorization:** Admin Required

##### **Request**

POST /api/notifications/returned HTTP/1.1

Host: yourdomain.com

Authorization: Bearer <your-token>

Content-Type: application/json

```
{  
  "_id": "66234abcd1234ef567890xyz",  
  "orderId": {  
    "_id": "66123abcde4567ef890xyz",  
    "name": "Laptop",  
    "category": "Electronics",  
    "quantity": 10,  
    "supplierId": "6609876543210abcd"  
  }  
}
```

##### **Response (Success - 200)**

```
{  
  "success": true  
}
```

##### **Response (Error - 500)**

```
{  
  "message": "Internal server error"  
}
```

## **Sales Management**

**Base URL:** /api/sales

**Middleware:** authenticateUser - Ensures only authenticated users can access sales routes.

### **1. Record Sales**

- **Endpoint:** POST /api/sales/recordSales
- **Description:** Records a list of sales transactions, updates product stock, category turnover, and sales records.

**Request Body:**

```
{
  "sales": [
    {
      "_id": "product_id",
      "category": "category_name",
      "name": "product_name",
      "quantity": 5
    }
  ]
}
```

**Response:**

- **200 OK**

```
{
  "success": true,
  "message": "Sales recorded successfully"
}
```
- **400 Bad Request** (Invalid or missing sales data)
 

```
{
        "success": false,
        "message": "Invalid sales data"
      }
```
- **500 Internal Server Error**

```
{
  "success": false,
  "message": "Server error",
  "error": "Error details"
}
```

```
}
```

**Sales Schema**

```
const salesSchema = new Schema({
  product: {
    type: String,
    required: true
  },
  turnover: {
    type: Number,
    default: 0,
  },
  preTurnOver: {
    type: Number,
    default: 0
  }
})
```

```
}  
, { timestamps: true }));
```

### Functionality

- Deducts sold quantity from Product.qtyRemaining and increases Product.qtySold.
- Updates Category.turnOver and Sales.turnOver.
- Ensures previous turnover (preTurnOver) is stored before updating current turnover.

### Notes

- If a product or category does not exist, a new record is created.
- qtyRemaining is never negative; it stops at 0 if sales exceed available stock.
- Logs missing products instead of failing the entire transaction.

## Dashboard Management

**Base URL:** /api/dashboard

### Authentication & Authorization

- Requires authentication (authenticateUser middleware)
- Requires admin authorization (authorizeAdmin middleware)

#### 1. Sales Overview

- **Endpoint:** GET /salesOverview
- **Description:** Provides an overview of total sales, revenue, profit, and incurred costs.

#### Response:

```
{  
  "totalSales": 1500,  
  "totalRevenue": 500000,  
  "totalProfit": 100000,  
  "totalCostIncurred": 400000  
}
```

#### 2. Inventory Summary

- **Endpoint:** GET /InventorySummary
- **Description:** Provides an overview of stock quantities in hand and pending incoming orders.

#### Response:

```
{  
  "quantityInHand": 1200,  
  "quantityToBeReceived": 500  
}
```

#### 3. Purchase Overview

- **Endpoint:** GET /purchaseOverview
- **Description:** Summarizes purchase details, including the total number of purchases, cost of delivered and returned orders.

#### Response:

```
{  
  "totalPurchases": 200,  
  "totalCostDelivered": 150000,
```

```
"totalCostReturned": 20000
}
```

#### **4. Product Summary**

**Endpoint:** GET /productSummary

**Description:** Provides the count of unique suppliers and product categories.

**Response:**

```
{
  "totalUniqueSuppliers": 30,
  "totalUniqueCategories": 15
}
```

#### **5. Sales and Purchases (Monthly Data)**

**Endpoint:** GET /salesAndPurchases

**Description:** Returns monthly sales and purchase amounts.

**Response:**

```
[
  { "month": "Jan", "Sales": 50000, "Purchase": 30000 },
  { "month": "Feb", "Sales": 45000, "Purchase": 35000 }
]
```

#### **6. Order Summary (Monthly Data)**

- **Endpoint:** GET /orderSummary
- **Description:** Returns monthly ordered and delivered quantities.

**Response:**

```
[
  { "month": "Jan", "Ordered": 200, "Delivered": 180 },
  { "month": "Feb", "Ordered": 250, "Delivered": 240 }
]
```

#### **7. Top Selling Stock**

- **Endpoint:** GET /topSellingStock
- **Description:** Fetches the top 3 best-selling products.

**Response:**

```
[
  { "name": "Product A", "qtySold": 500, "qtyRemaining": 200, "sellingPrice": 100 },
  { "name": "Product B", "qtySold": 450, "qtyRemaining": 150, "sellingPrice": 120 }
]
```

#### **8. Low Quantity Stock**

**Endpoint:** GET /lowQuantityStock

**Description:** Fetches the top 3 products with the lowest remaining stock ( $\leq 10$  units).

**Response:**

```
[
  { "name": "Product X", "qtyRemaining": 5, "productImage": "image_url" },
  { "name": "Product Y", "qtyRemaining": 8, "productImage": "image_url" }
]
```



## Error Responses

For all endpoints, common error responses include:

### 400 Bad Request:

```
{  
  "message": "Invalid request data"  
}
```

### 404 Not Found:

```
{  
  "message": "Resource not found"  
}
```

### 500 Internal Server Error:

```
{  
  "message": "Internal server error"  
}
```

## Error Handling and Troubleshooting

Effective error handling and troubleshooting ensure the smooth operation of the **Inventory Management System**. Below are common errors users and developers may encounter, along with steps for resolving them.

### Email Notification Failures

**Error Description:** In the forgotPassword functionality, email notifications are critical for sending password reset links. If the SMTP service fails (e.g., due to incorrect credentials, connectivity issues, or Gmail restrictions), the user won't receive the reset email, leaving them unable to reset their password.

#### Troubleshooting Steps:

##### 1. Check SMTP Configuration:

- Host: smtp.gmail.com
- Port: Use 465 for SSL (secure: true) or 587 for TLS (secure: false).
- Transporter Config: Ensure the nodemailer.createTransport configuration matches Gmail's requirements.

##### 2. Gmail Account Security Settings:

- Enable access for less secure apps (for test accounts only, not recommended for production).
- Go to Google Account Security Settings and verify settings.

##### 5. Network/Firewall Checks:

- Ensure your server or development environment is not blocking SMTP ports (465, 587).

### Port Conflicts

**Error Description:** The server may fail to start if the configured port (e.g., 5000) is already in use.

#### Troubleshooting Steps:

##### 1. Identify Conflicting Process:

- Linux/macOS: `lsof -i :5000`
- Windows: `netstat -ano | findstr :5000`

##### 2. Terminate Process:

- Linux/macOS: kill -9 <PID>
- Windows: taskkill /PID <PID> /F

### 3. Change Port:

- Modify the port number in the project's .env file to something like 3001 or 8080.

## **Database Connection Failures (MongoDB)**

**Error Description:** Failure to connect to the MongoDB database due to incorrect credentials or service downtime.

### **Troubleshooting Steps:**

#### 1. Check MongoDB URI: Ensure it matches the format:

```
mongodb+srv://<USERNAME>:<PASSWORD>@<CLUSTER_NAME>.mongodb.net/<DATABASE_NAME>?retryWrites=true&w=majority&appName=<APP_NAME>
```

#### 2. Ensure MongoDB is Running:

- Local: Confirm MongoDB service is active.
- Cloud (Atlas): Check cluster status and network access settings.

#### 3. Permissions: Verify that the database user has read/write access.

#### 4. Firewall and IP Whitelist: Add the appropriate IP to the whitelist in MongoDB Atlas.

## **API Request Failures**

**Error Description:** Backend API requests may fail due to misconfigured routes or missing headers.

### **Troubleshooting Steps:**

#### 1. Validate Endpoints: Confirm that client requests match server route definitions.

#### 2. Authentication: Ensure JWT tokens or session IDs are included in request headers if needed.

#### 3. Analyze Response Codes:

- 404 Not Found: Incorrect route or URL typo.
- 500 Internal Server Error: Backend crash; check server logs.
- 401 Unauthorized: Missing/invalid credentials.

## **File Upload Failures (Product Images)**

**Error Description:** Errors may occur while uploading images for products.

### **Troubleshooting Steps:**

#### 1. File Size Limit: Ensure files don't exceed backend limits (e.g., 5MB).

#### 2. Allowed File Types: Accept only .jpg, .png, .jpeg.

#### 3. Storage Issues: Ensure the upload directory has proper write permissions and sufficient space.

## **Authentication Errors (JWT or Session)**

**Error Description:** Issues with user login or session persistence may arise due to expired JWTs or incorrect credentials.

### **Troubleshooting Steps:**

#### 1. Check Token Expiration: Ensure the token has not expired.

#### 2. Verify Credentials: Confirm user login inputs and backend authentication logic.

#### 3. Token in Requests: Ensure tokens are included in Authorization headers as Bearer <token>.

## **Frontend Rendering Issues**

**Error Description:** The React-based frontend may fail to render due to state errors, missing props, or API failures.

### **Troubleshooting Steps:**

#### 1. Check Console for Errors: Use DevTools to identify missing props, undefined variables, etc.

2. Validate Data Flow: Confirm the structure of API responses and the props being passed to components.
3. Install Missing Packages: `npm install`
4. Check CORS Errors: Ensure the backend allows frontend-origin requests.

## **General Troubleshooting Tips**

1. Clear Cache: Clear browser or application cache.
2. Check Logs:
  - Frontend: Browser console (F12).
  - Backend: Server console or log files.
3. Use Debuggers: Step through code using VS Code debugger or Chrome DevTools.
4. Run in Safe Mode: Disable plugins or extensions that may interfere.

## **Maintenance and Support**

### **1. Database Backup**

- Use MongoDB Atlas' automated backups or run `mongodump` for local backups.
- Store backups in secure cloud services (e.g., Google Cloud Storage, AWS S3).

### **2. API Keys & Environment Variables**

- Use `.env` files for storing secrets.
- Rotate sensitive keys periodically.
- Use Dotenv, AWS Secrets Manager, or Azure Key Vault for secure key management.

## **Version Control and Updates**

### **1. Git & Branching**

- Use `main` for production.
- Create `feature/` or `bugfix/` branches as needed.
- Tag stable versions (e.g., `v1.0.0`).

### **2. Safe Updates & Migrations**

- Use a staging branch or environment for testing.
- Run database migration scripts using Mongoose migration tools.
- Always test thoroughly before pushing updates live.

## **Glossary**

- **Admin:** A user role with full access to all system functionalities including dashboard, inventory management, supplier management, and analytics.
- **CORS (Cross-Origin Resource Sharing):** A security feature that restricts web pages from making requests to a different domain than the one that served the original page.
- **Dotenv:** A module that loads environment variables from a `.env` file into the Node.js process.
- **Inventory Management:** The process of tracking, organizing, and controlling stock levels of products.
- **Nodemailer:** A module for Node.js applications to send emails, used for password reset functionality.
- **Pagination:** The process of dividing content into discrete pages, used in StockSync's product, supplier, and order listings.
- **User:** A standard role with limited access to system features, primarily the shop page and order placement.

## **Frequently Asked Questions (FAQs)**

### **1. How do I set up the MongoDB connection?**

In your .env file, set the DB\_CONN\_STRING variable to your MongoDB connection URL. For a local MongoDB instance, use mongodb://127.0.0.1:27017/<name>. For MongoDB Atlas, use the connection string provided in your Atlas dashboard.

### **2. How do I create an admin account?**

Admin accounts must be created directly in the database. By default, all accounts created through the registration form are regular user accounts.

### **3. How do I set up low-stock alerts?**

When adding or editing a product, set the "Threshold Value" field to the minimum quantity you want to maintain. When the stock level falls to or below this value, the system will generate a low-stock alert.

### **4. Can I track orders from specific suppliers?**

Yes, the system tracks orders by supplier. In the Suppliers section, you can view the "On the Way" count for each supplier, which shows pending deliveries.

### **5. The server won't start due to a port conflict. What should I do?**

Change the PORT value in your .env file to an available port (e.g., 3001, 8080). Alternatively, identify and terminate the process using the conflicting port.

### **6. The frontend isn't connecting to the backend. What should I check?**

Ensure both frontend and backend servers are running. Check the CLIENT\_URL in your backend .env file matches your frontend URL. Look for CORS errors in the browser console and ensure CORS is properly configured. Verify that your API request URLs are correctly formatted.

### **7. How does StockSync handle data security?**

Password hashing with salt using modern cryptographic methods. JWT-based authentication with limited token lifetime. Environment variables for sensitive information. HTTPS recommended for production deployments. Role-based access control to restrict feature access.

## **Future Enhancements**

- **Predictive Inventory Management:** Implement machine learning algorithms to forecast inventory needs based on historical data and seasonal trends.
- **Native Mobile Applications:** Develop dedicated iOS and Android apps for on-the-go inventory management.
- **Barcode/QR Scanner Integration:** Allow for mobile scanning of products for quick stock checks and updates
- **Automated Purchase Orders:** Generate and send purchase orders automatically when stock reaches threshold levels.
- **Personalized Product Recommendations:** Suggest products based on customer purchase history.
- **Onboarding Wizard:** Create interactive tutorials for new users