Store Management

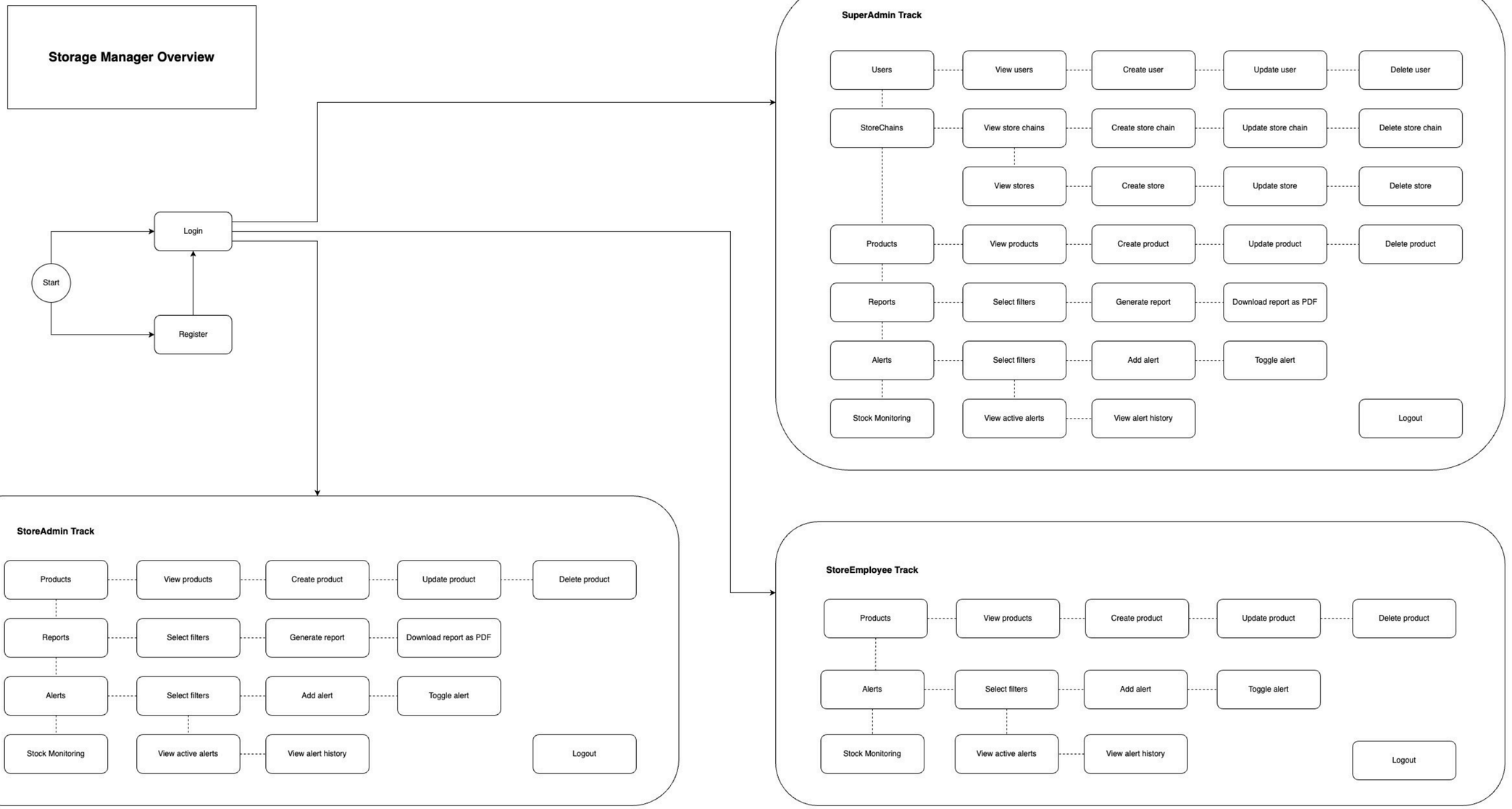
Team 13
Banilean Alexandru
Besliu Radu
Nechita Ilinca
Pasare Francisca
Stanciu Carol

About the application

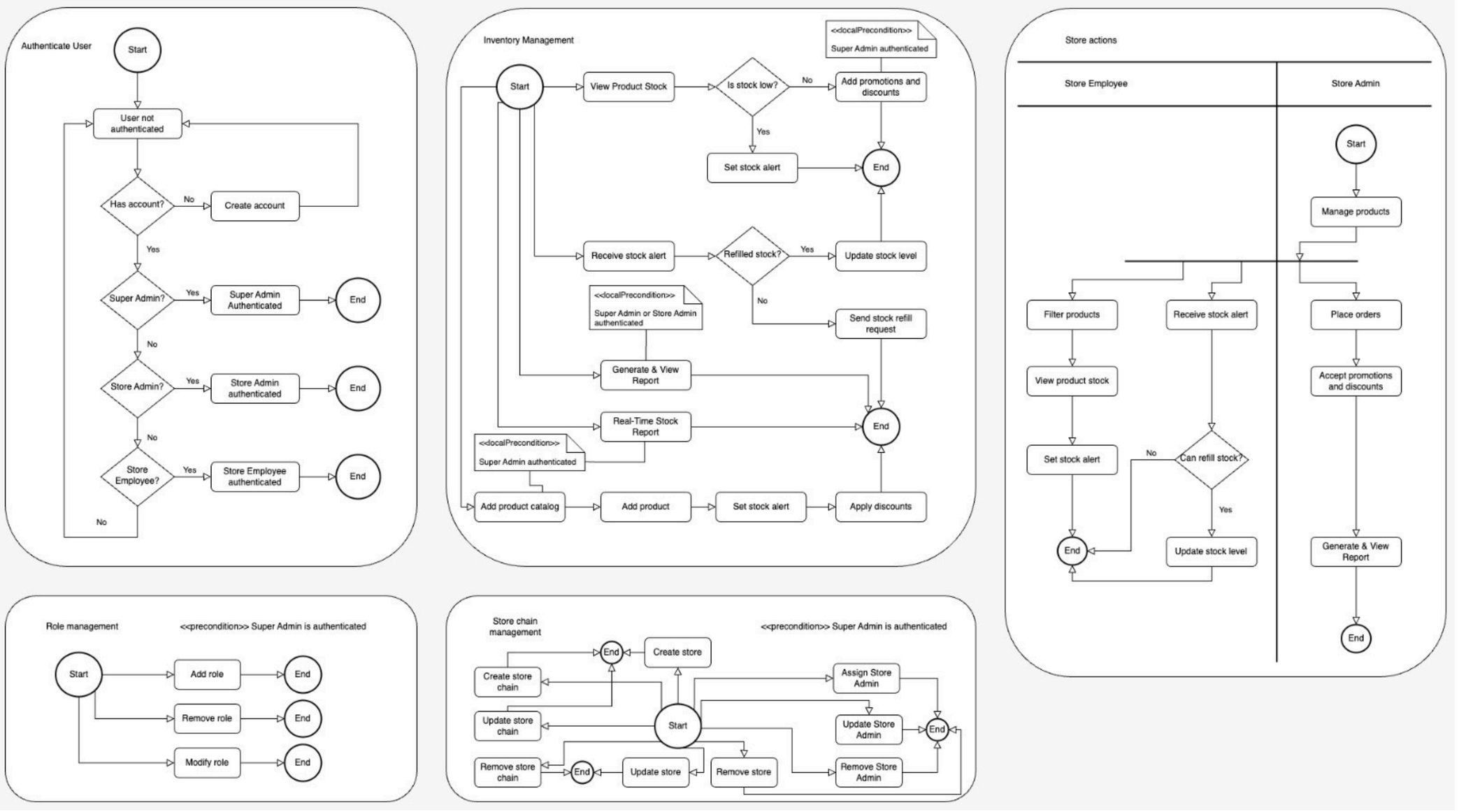
This application is designed to facilitate store operations by automating inventory management, optimizing ordering processes, easily generating detailed reports, and supporting marketing campaigns through promotions and discounts.

It is a web platform that supports multiple user roles and allows the execution of various activities specific to each role, from Super Admin to Employee.

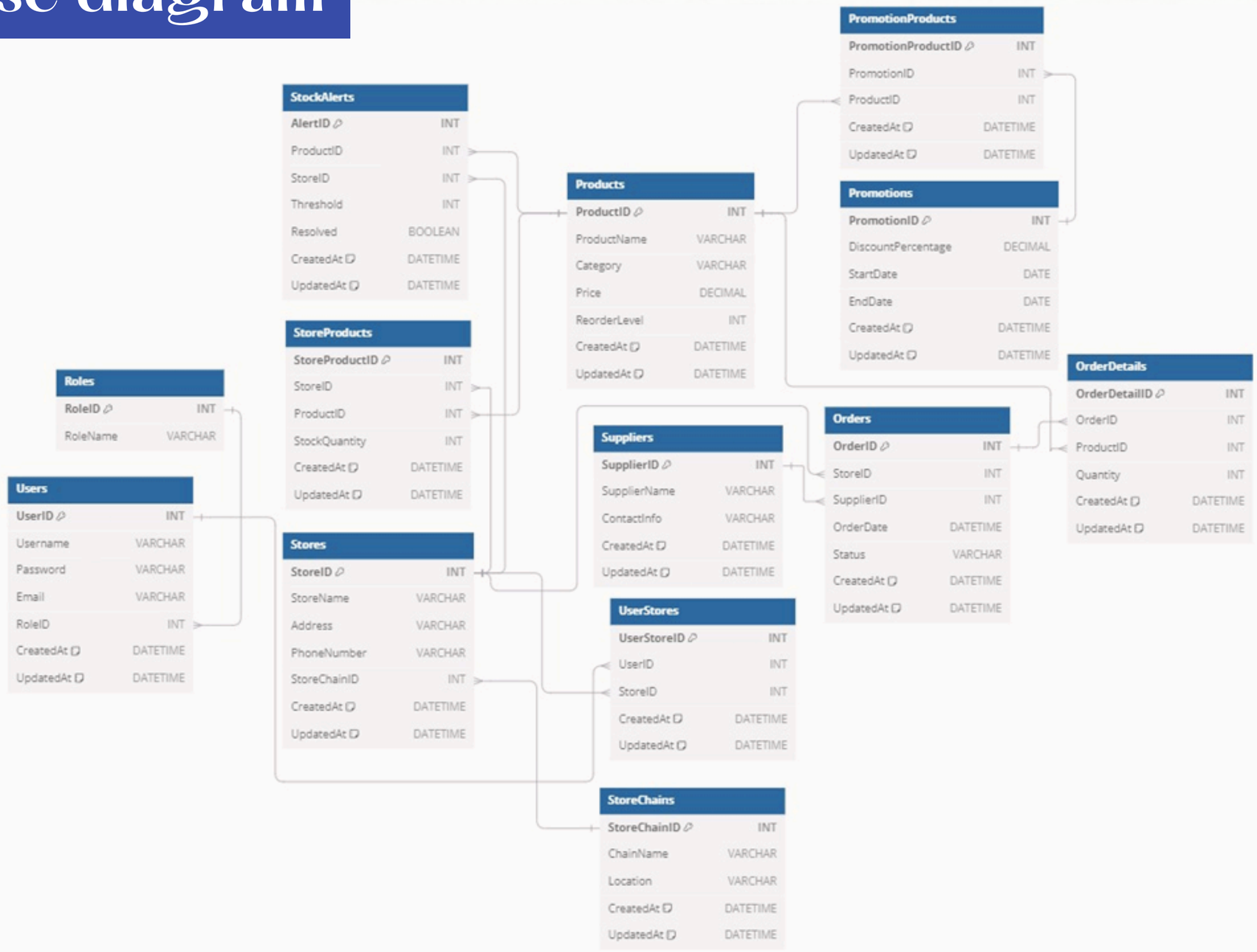
Overview diagram



Workflow diagram

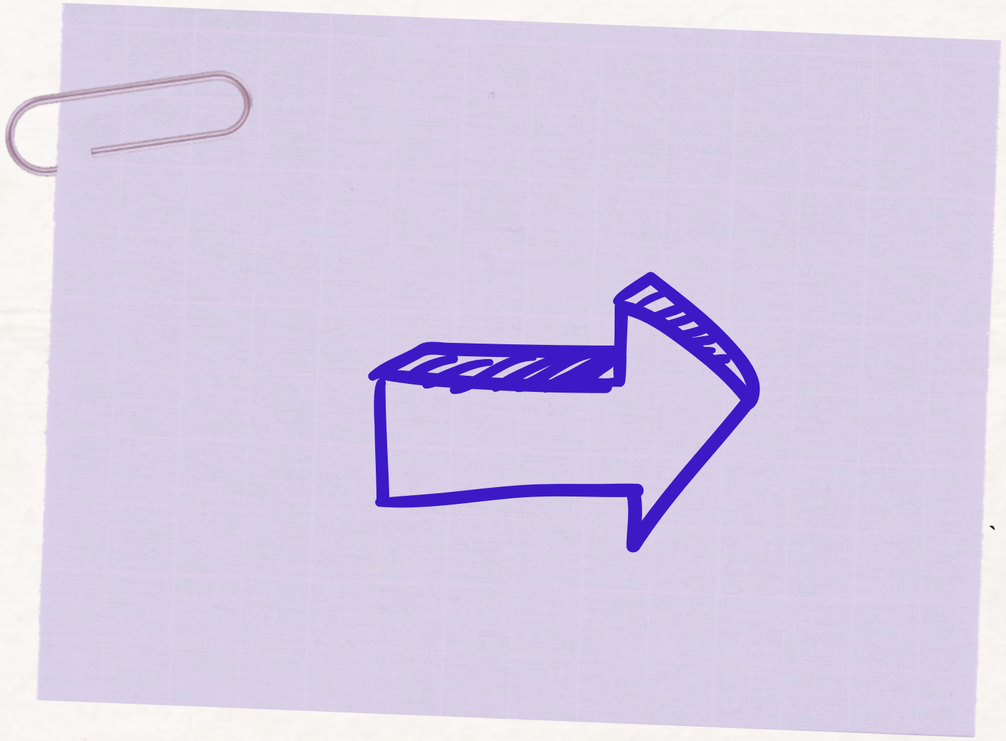


Database diagram



Activity diagrams

The workflow of this application can be divided according to the components we have to operate: stores, products, promotions, log in.

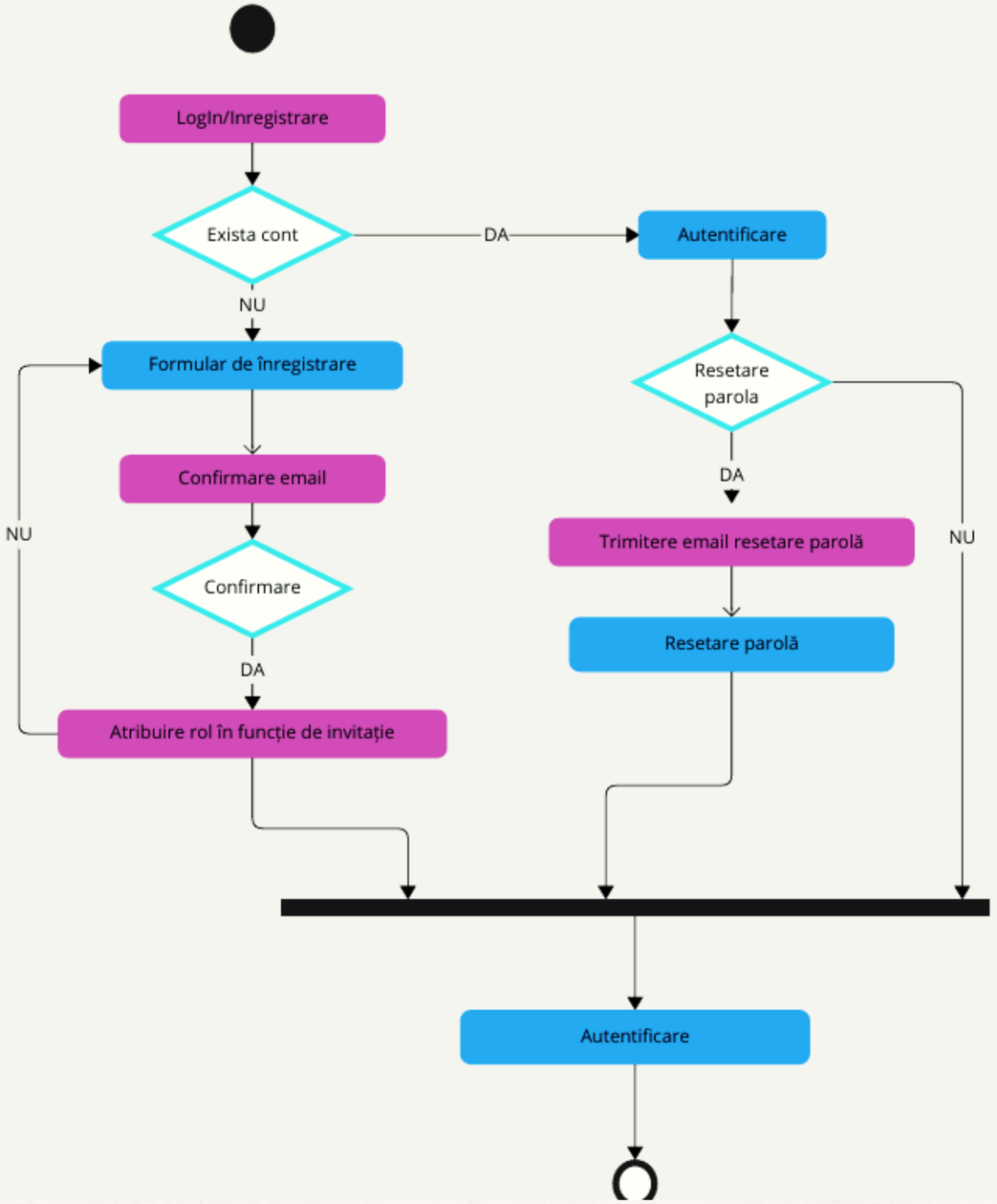


Authentication

Login/Sign Up
→ Choose to log in or create a new account.

Account Exists?
→ Yes: Proceed to login.
→ No: Fill in the registration form.

Email Confirmation
→ Verify the email address.
→ Yes: Confirm registration.
→ No: Resend confirmation.

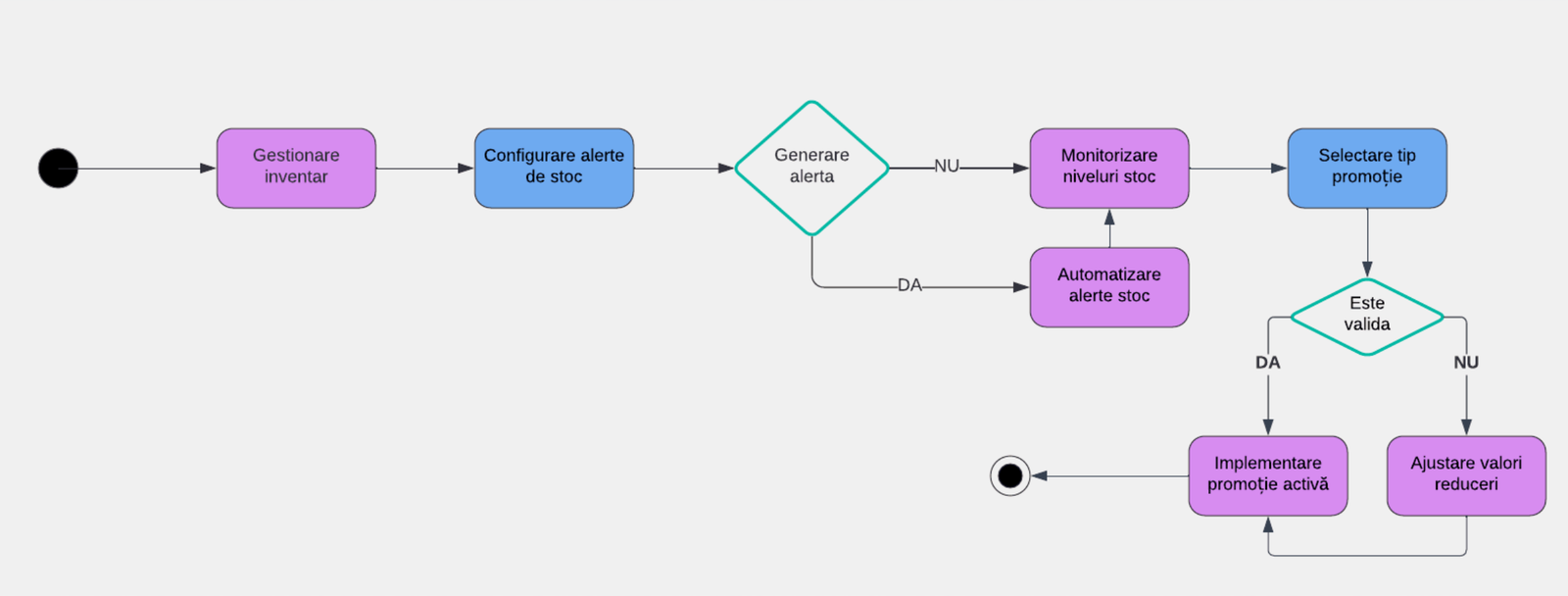


Forgot Password?
→ Yes: Send password reset email.
→ No: Continue with login.

Password Reset
→ Change the password and log in.

Role Assignment
→ Assign user roles based on the invitation.

Stock management



Inventory Management
→ Manage and update stock levels.

Set Stock Alerts
→ Configure alerts for low stock.

Generate Alert?
→ Yes: Trigger automated alerts.
→ No: Continue monitoring stock.

Monitor Stock Levels
→ Keep track of inventory in real time.

Automate Alerts
→ Automatically notify when stock is low.

Adjust Discounts

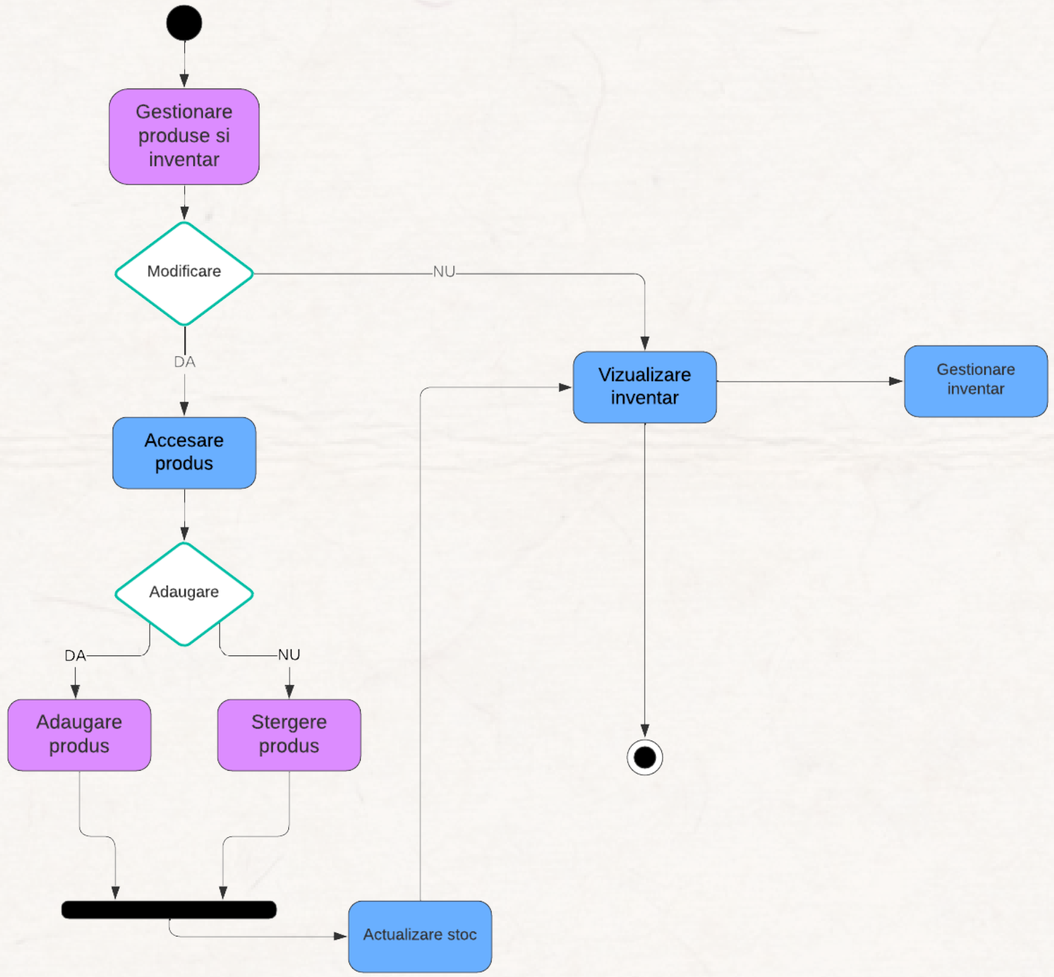
→ Fine-tune discounts for better results.

Select Promotion Type
→ Choose the right promotion for products.

Is the Promotion Valid?
→ Yes: Apply the promotion.
→ No: Adjust discount values.

Activate Promotion
→ Launch the promotion for customers.

Product management



Manage Products & Inventory
→ Start managing product data.

Modify Product?
→ Yes: Access product details.
→ No: View inventory.

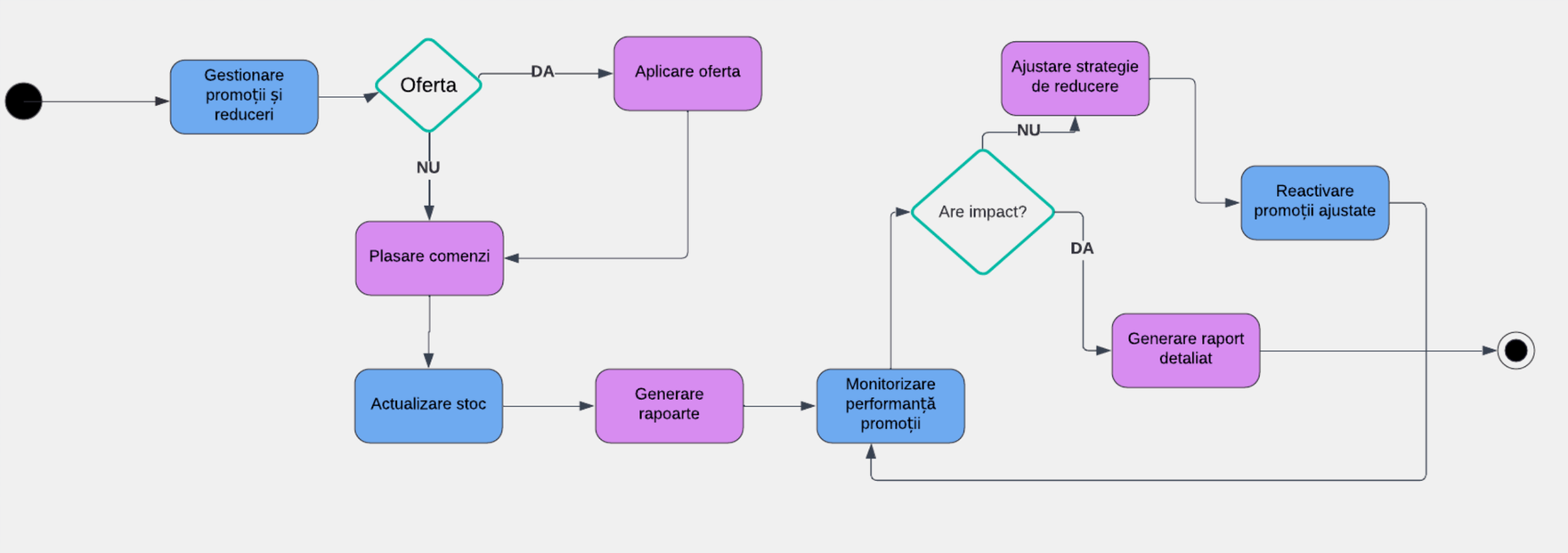
Add or Delete Product?

→ Yes: Add a new product.
→ No: Delete an existing product.

Update Inventory
→ Reflect product changes in stock levels.

View Inventory
→ Monitor and manage existing stock.

Promotions



Manage Promotions & Discounts
→ Set up and organize offers.

Offer Decision (Has discount?)
→ Yes: Apply the offer.
→ No: Place product orders.
Update Inventory
→ Adjust stock after orders.

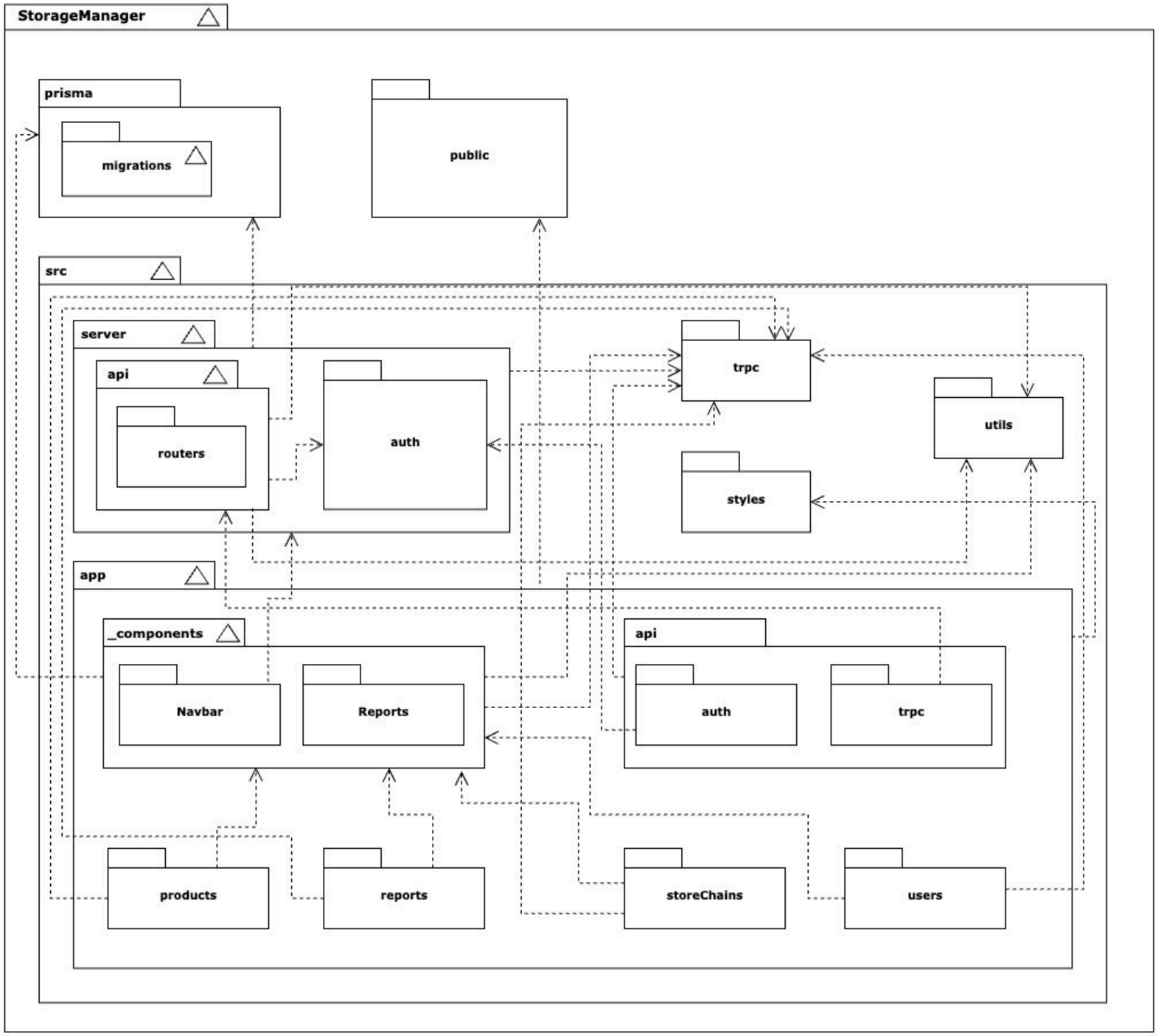
Generate Reports
→ Create reports for performance tracking.

Monitor Promotion Impact
→ Check if the promotion is effective.
Impact Check (has Impact?)
→ Yes: Generate a detailed report.
→ No: Adjust the discount strategy.

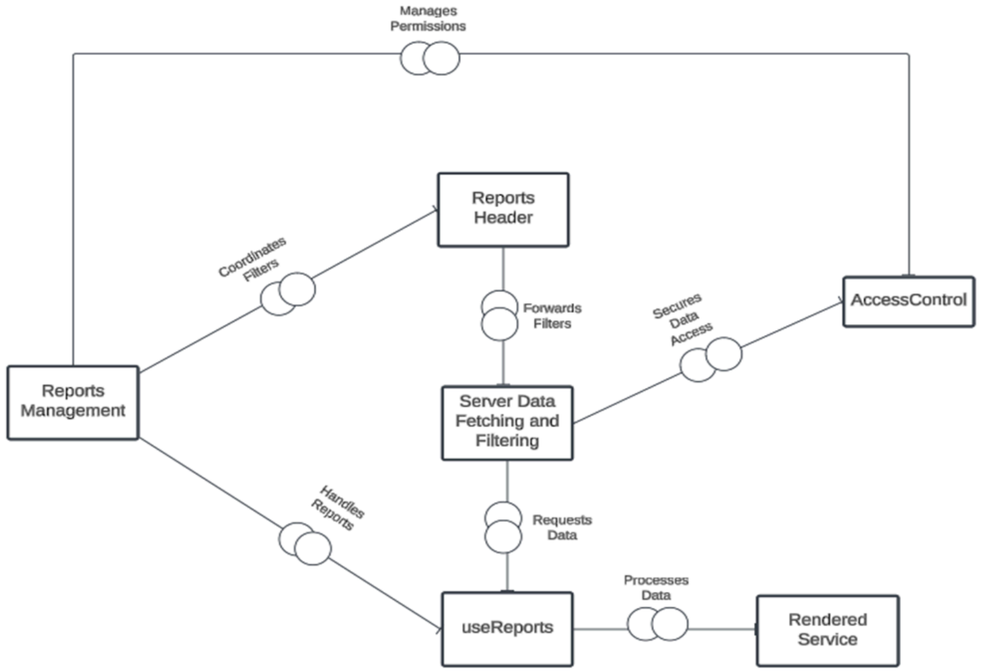
Adjust Strategy

→ Refine the discount for better results.
Reactivate Promotion
→ Relaunch the updated promotion.

Package diagram



Reports management



Reports Management

Central Hub: This core component manages the overall report flow in the system. It interacts with other components to handle and filter data for reports.
Manages Permissions: Controls data access permissions, ensuring secure and compliant access to reports.
Coordinates Filters: Handles data filters applied to reports, ensuring the correct information is retrieved for analysis and presentation.
Handles Reports: Manages report creation and usage, coordinating how final reports are used and displayed

Reports Header

Manages Headers: Responsible for managing report headers, including titles and other formatting elements.
Forwards Filters: Sends filters set by Reports Management to the data fetching and filtering component.

useReports

Processes Data: Processes data from Server Data Fetching and Filtering and generates reports based on it.
Delivers Reports: Sends completed reports to the Rendered Service for user presentation.

Rendered Service

Delivers final reports to users through a user interface, ensuring information is accessible and easy to understand.

Server Data Fetching and Filtering

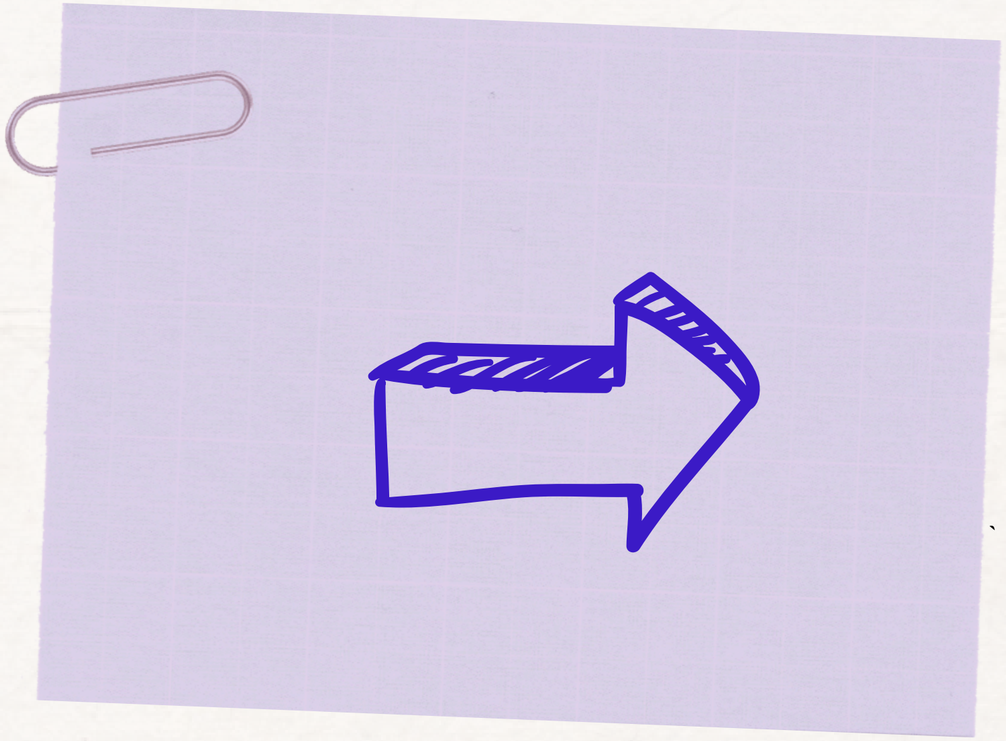
Secures Data Access: Works with AccessControl to secure data access during retrieval.
Requests Data: Sends filtered data to useReports for final processing, preparing it for complete report generation.

Access Control

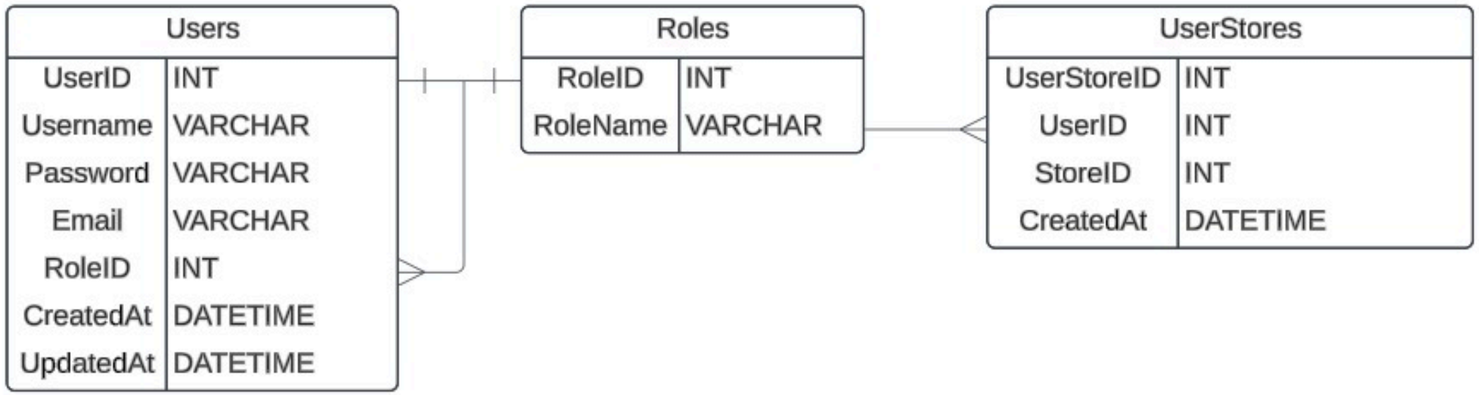
Controls report and data access security, ensuring confidentiality and integrity standards are always met.

Structural diagrams

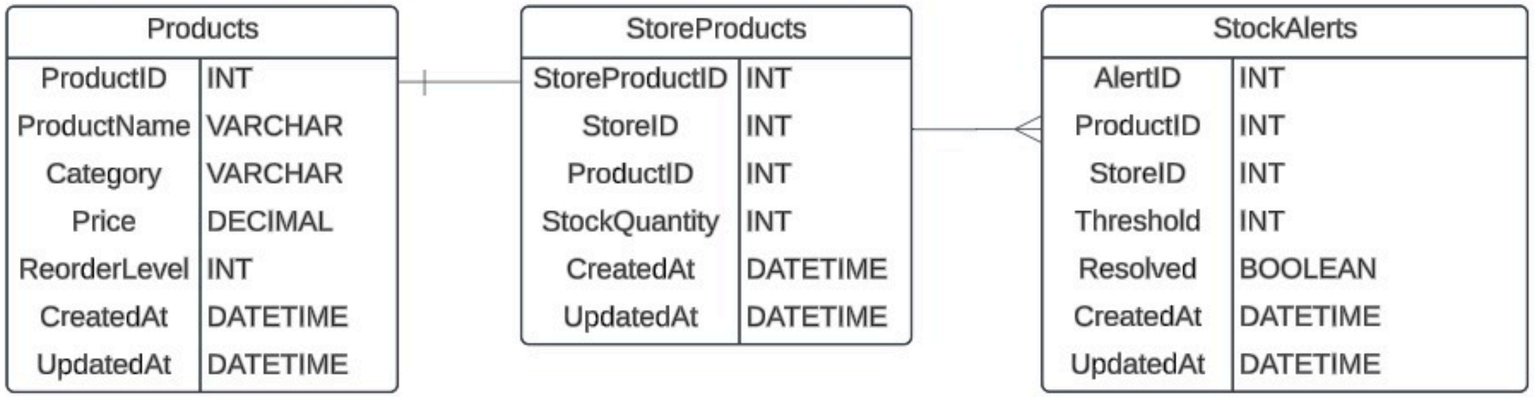
The application is connected to a relational database, so we chose to represent the classes involved in the application's workflow.



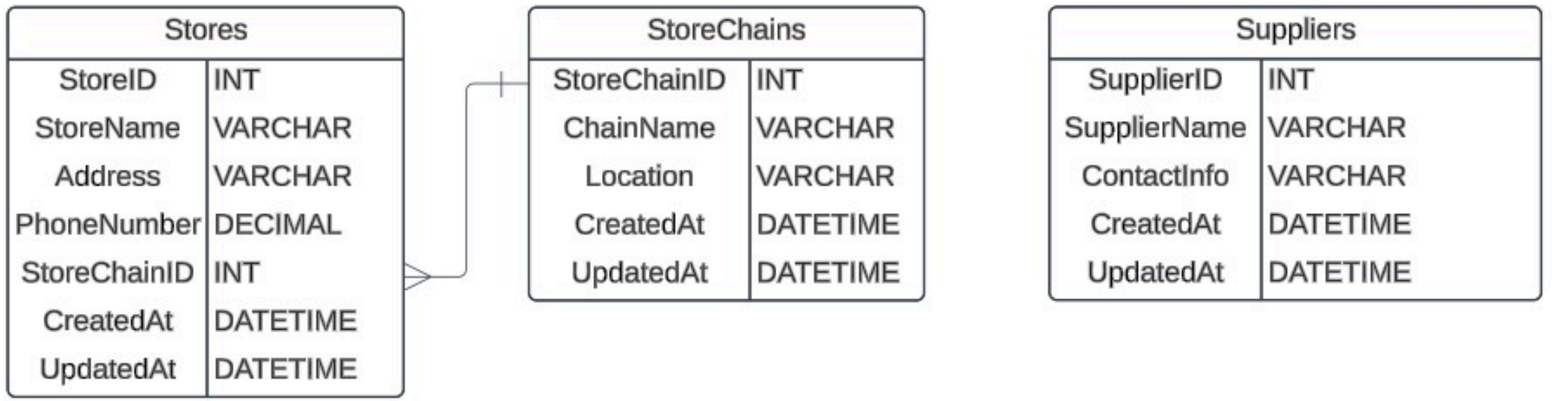
Application User



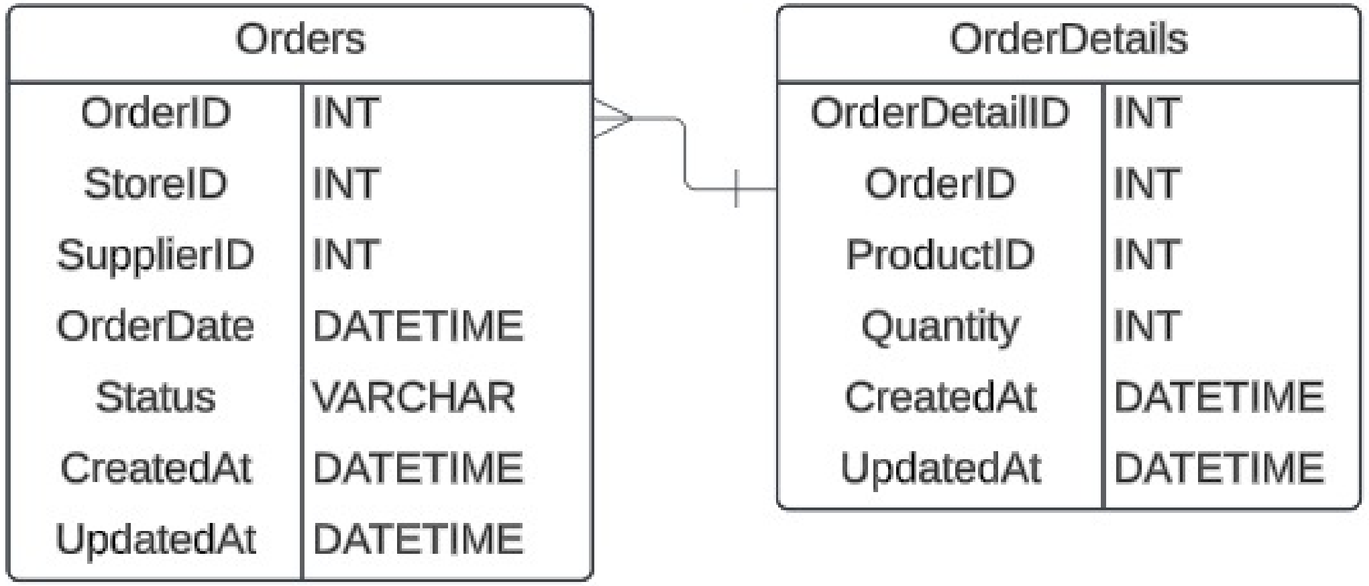
Product management



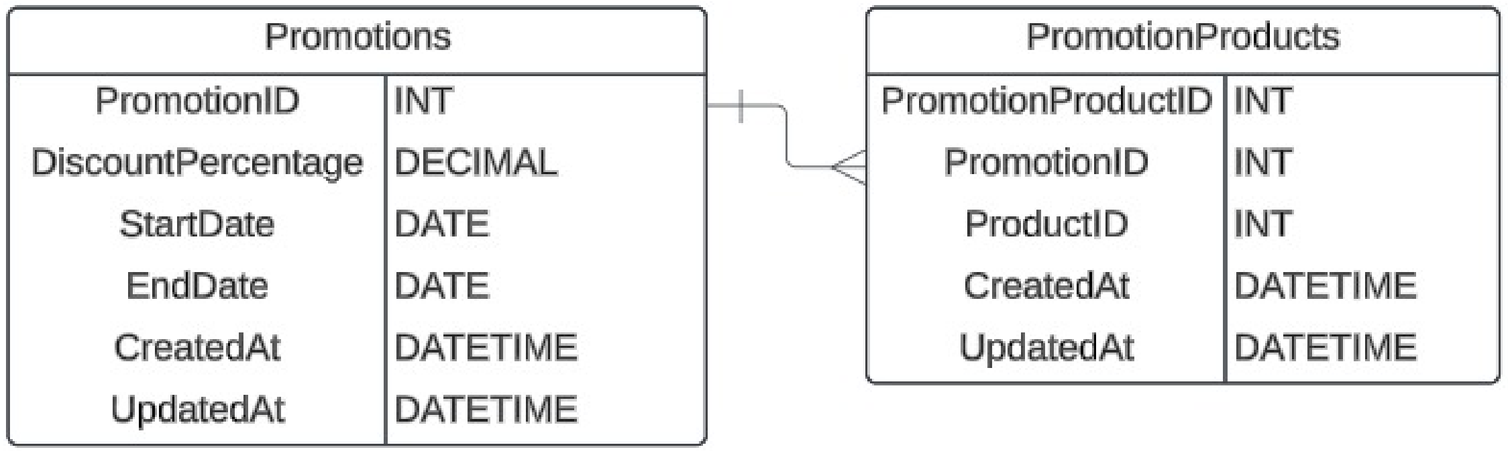
Store management



Placing an order



Applying promotions



Functional Requirements

Management of the Stores:

- Super Admin can add and remove stores from the network.
- The interface allows entering store details (name, address, phone).
- Confirmation when deleting stores.

Administration of Users and Roles:

- Super Admin: assigns, modifies and removes roles for users.
- Store Admin: manages products and updates the store's inventory, views and generates performance reports. Creates stock alerts and monitors inventory.
- Store Employee: filters products and supports inventory management.

Product and Inventory Management:

- Add, update and remove products (price, stock, description).
- Automatic inventory update after sales/deliveries (store employees).

Functional Requirements

Placing Orders to Suppliers:

- Initiate and manage orders to suppliers(store admin).
- Automatic inventory confirmation and update after orders are placed.

Stock Alerting and Monitoring:

- Configures and automates alerts for low stock levels.
- Sends notifications when stocks reach critical thresholds.

Reporting and Analysis:

- Generates detailed reports on sales, inventory, and performance.
- Real-time reports presented in graphical format.

Promotion and Discount Management:

- Configures and manages promotions and discounts.
- Automatically sends offers to store admins.

Non-functional Requirements

1. **Performance**: the system responds quickly to complex queries and support large data volumes.

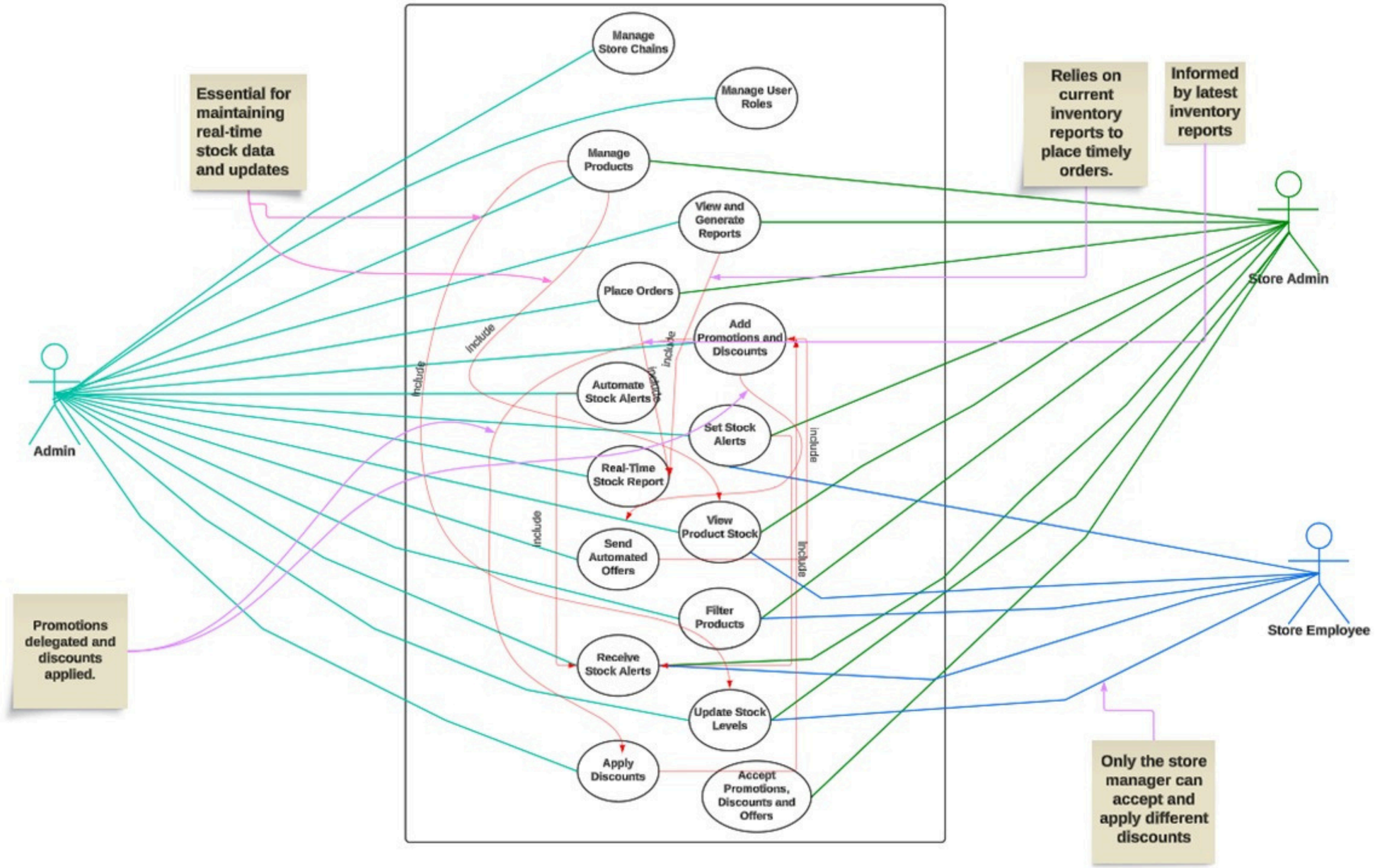
2. **Accessibility**: The interface is intuitive and adaptable across various devices (desktop, tablet, mobile).

3. **Usability**: an user-friendly interface, easy to navigate for all user categories.

4. **Scalability**: the platform is scalable, allowing the addition of new stores and users.

5. **Security**: secure user authentication and data encryption.

Use Case diagram



User stories

1. Super Admin
2. Store Admin
3. Store Employee

Super admin

1. Manage Stores: Add or remove stores in the network.
2. Control Access: Assign, change, or delete user roles.
3. Manage Products: Keep the product list updated and accurate.
4. View Reports: Generate reports to make better decisions.
5. Order Stock: Place orders with suppliers to keep shelves full.
6. Create Promotions: Add and manage discounts to boost sales.
7. Set Stock Alerts: Get notified when products are low in stock.
8. Automate Alerts: Set automatic stock notifications.
9. Live Stock Reports: See real-time stock updates.
10. Send Offers: Share promotions with store admins.
11. Check Stock Levels: Monitor product availability.
12. Quick Search: Filter products for fast access.
13. Instant Alerts: Get stock alerts.
14. Update Inventory: Keep stock levels accurate after sales.
15. Apply Discounts: Offer attractive deals to customers.

Store admin and store employee

Store Admin

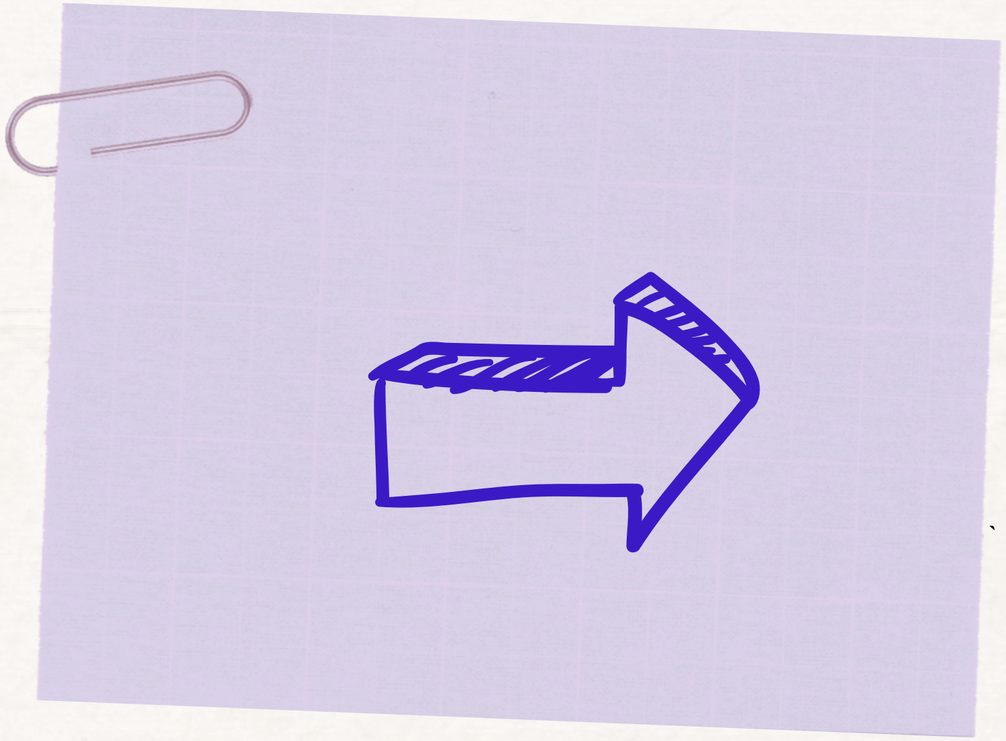
- Manage Inventory: Keep products updated and organized.
- Generate Reports: Check store performance with reports.
- Order Supplies: Restock by ordering from suppliers.
- Stock Notifications: Set alerts for low stock.
- Monitor Stock: Keep an eye on inventory levels.
- Quick Product Search: Find products easily.
- Update Stock: Adjust inventory after sales and deliveries.
- Approve Discounts: Accept and apply promotions.

Store Employee

- Set Alerts: Create stock alerts to avoid shortages.
- Watch Inventory: Monitor stock levels daily.
- Find Products Fast: Use filters for quick searches.
- Act on Alerts: Respond to stock notifications.
- Update Stock: Keep inventory correct after sales and deliveries.

Design Patterns

This section will present the design patterns used in our implementation.



1

Factory Method

Context: The Navbar component dynamically returns a role-specific navbar (e.g., SuperAdminNavbar, StoreEmployeeNavbar) based on the user's role.

How it works:
- A single Navbar component gets the user role from the session.
- Depending on the role, the component instantiates and returns the appropriate specialized navbar.

Benefits:
- Encapsulates role-specific logic into separate components.
- Facilitates easy addition of new roles and navbars.

```
const Navbar = async () => {
 const session = await auth();

 switch (user.role) {
  case Role.SUPER_ADMIN:
   return <SuperAdminNavBar />;
  case Role.STORE_ADMIN:
   return <StoreAdminNavBar />;
  case Role.STORE_EMPLOYEE:
   return <StoreEmployeeNavBar />;
  default:
   return <></>;
 }
};
```

2 Facade

Context: The useReports hook abstracts the complexity of fetching and processing data for different report types and provides a unified interface for rendering reports.

How it works:
- The hook fetches all required data and exposes a renderReports function.
- The function dynamically delegates the rendering process to the appropriate rendering logic based on the report type.

Benefits:
- Simplifies the interface for rendering reports.
- Hides the complexity of data fetching and processing.

```
const useReports = () => {
  const { data: saleReport } =
api.router.getSaleReport.useQuery();
  const { data: priceChangeReport } =
api.router.getSaleReport.useQuery();

  const renderReports = (reportType) => {
   switch (reportType) {
    case Event.SALE:
     return renderSaleReport(saleReport);
    case Event.PRICE_CHANGE:
     return
renderPriceChangeReport(priceChangeReport);
    default:
     return null;
   }
  };

  return { renderReports };
 };
```

3

Decorator

Context: The TRPC query client enhances API calls with additional features like caching, invalidation, and error handling.

How it works:
- API calls are wrapped with functionalities such as optimistic updates.
- Example of invalidation in the right.

Benefits:
- Enhances core functionality (e.g., optimistic updates, error handling) without altering the main logic.
- Promotes reusable and modular enhancements.

```
const doDeleteProduct =
api.product.delete.useMutation({
  onSuccess: async () => {
    await utils.product.getInfinite.invalidate();
    await utils.product.get.invalidate();
  },
  onError: (error) => {
    console.error(error);
  },
 });
```

4

Observer

Context: The Observer pattern is used in the product infinite scrolling implementation. Whenever new product data is needed, useInView (from react-intersection-observer) observes the scrolling behavior to fetch the next page.

How it works:
- useInView is used to detect when a user scrolls near the bottom of the product list.
- When the observer triggers, `fetchNextPage` from useInfiniteQuery is called to fetch more products.

Benefits:
- Efficient and seamless data fetching triggered by user actions.
- Reduces resource usage by loading data incrementally instead of all at once.

```
const {
 data: productsData,
 hasNextPage,
 fetchNextPage,
 isFetchingNextPage,
} = api.product.getInfinite.useInfiniteQuery(
 {
  limit: 20,
  storeId: session.data?.user?.storeId ?? undefined,
 },
 {
  getNextPageParam: (lastPage) => lastPage.nextCursor,
  refetchOnWindowFocus: false,
 },
);

const { ref, inView } = useInView();

useEffect(() => {
 if (inView && hasNextPage) {
 fetchNextPage();
 }
}, [inView, fetchNextPage, hasNextPage]);
```

5 Strategy

Context: The renderReports function dynamically selects the rendering logic based on the report type.

How it works:
- Each report type is associated with a distinct rendering strategy.
- The function maps the report type to the appropriate rendering function and invokes it.

Benefits:
- Encapsulates rendering logic for different report types.
- Simplifies the addition of new report types.

```
const useReports = () => {
  const { data: saleReport } =
api.router.getSaleReport.useQuery();
  const { data: priceChangeReport } =
api.router.getSaleReport.useQuery();

  const renderReports = (reportType) => {
    switch (reportType) {
      case Event.SALE:
        return renderSaleReport(saleReport);
      case Event.PRICE_CHANGE:
        return
renderPriceChangeReport(priceChangeReport);
      default:
        return null;
    }
  };

  return { renderReports };
};
```

6

Continous Scrolling

Context: useInfiniteQuery is used for implementing continuous scrolling, enabling seamless data fetching for products.

How it works:
- Data is fetched incrementally in pages as the user scrolls.
- useInfiniteQuery provides utilities for fetching the next page when needed.

Benefits:
- Improves user experience by loading data on demand.
- Reduces initial load time and resource usage.

```
const {
 data: productsData,
 hasNextPage,
 fetchNextPage,
 isFetchingNextPage,
} = api.product.getInfinite.useInfiniteQuery(
 {
  limit: 20,
  storeId: session.data?.user?.storeId ??
undefined,
 },
 {
  getNextPageParam: (lastPage) =>
lastPage.nextCursor,
  refetchOnWindowFocus: false,
 },
);
```


Thank
you!