الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria Ministry of higher education and scientific research 8 Mai 1945 - Guelma University
Faculty of Mathematics, Computer Science and sciences of matter
Department of Computer Science

Second year computer engineering

Developing a

Hypermarket management system

Team members:

1. DJEGHBAL Sidali
2. BOUTOUATOU Hichem
3. CHERFA Mohammed Akram
4. HALITIM Amin
5. ZOUAOUI Djasser
6. TOUALBIA Taki Eddine

Mentored by: HANNOUSSE Abdelhakim

2024/2025

Table of Contents

List of Figures

Acknowledgments

We would like to express our deep gratitude to those who helped and supported us throughout this Project.

Our sincere thanks go to Mr. Benabdellah Ahcène Youcef, the multidisciplinary project manager, for his constant efforts, patience, and valuable advice throughout our project.

We express our heartfelt gratitude to our supervisor, Mr. Abdelhakim Hannousse, for his invaluable advice and guidance throughout our project.

Finally, and not to be forgotten, a big thank you to the entire family of the computer science department: all the teachers and students who contributed to the success of this project.

Introduction

In today's retail landscape, effectively managing operations is paramount for business owners. This often involves overseeing a complex interplay of sales, inventory, and personnel to ensure smooth daily functioning and sustained success. Historically, these tasks relied heavily on manual processes, but with the increasing pace of commerce and the sheer volume of transactions, retail managers are constantly seeking more efficient ways to maintain control and gain insights into their businesses.

Problem Statement

Managing modern hypermarkets presents several significant operational challenges. Firstly, manual sales and inventory management often leads to errors and considerable inefficiencies, hindering smooth operations. Secondly, there's a critical lack of real-time visibility into stock levels across different branches, making it difficult to optimize inventory and prevent shortages or overstock. Thirdly, the difficulty in generating accurate and timely reports significantly impacts effective decision-making. Fourthly, managing employees and cash registers across multiple locations becomes a complex and time-consuming task. Lastly, the absence of a centralized system for all commercial operations means data is often fragmented and difficult to consolidate, creating inefficiencies and hindering overall business intelligence.

Objectives

This system aims to provide a robust solution for managing retail operations within hypermarkets. The specific objectives include:

1. **Automation of Sales Processes:** Automate sales processes across multiple supermarket locations to enhance speed and reduce manual errors.
2. **Real-time Inventory Management:** Track inventory in real-time and efficiently manage transfers of goods between different locations, ensuring optimal stock levels.
3. **Role-Based Access Control:** Implement a secure role-based access control system for different types of users, ensuring data integrity and operational security.
4. **Automated Report Generation:** Automatically generate daily sales reports, providing timely and accurate data for informed decision-making.
5. **Unified Operational Platform:** Provide a centralized and unified platform for all retail operations, streamlining management and improving overall efficiency.

This project is designed to optimize the daily operations of hypermarkets, addressing the critical need for efficient and integrated management in today's competitive retail environment.

Chapter I: Analysis

1. Presentation of the Hypermarket Context

Shopping Center represents a prominent hypermarket chain, operating across multiple branches in various regions. This structure signifies a large-scale retail operation designed to cater to thousands of customers daily. This diverse product offering, combined with the high volume of daily transactions, inherently presents significant operational challenges.

2. Existing Hardware and Current Processes

The current hardware infrastructure at Shopping Center reflects an older generation of retail technology, characterized by localized solutions and limited interconnectivity.

1. Hardware Resources:

- **Traditional Cash Registers with Limited Functionality:** Each sales point is equipped with a traditional cash register. These devices are primarily designed for basic transaction processing – recording sales, calculating totals, and printing receipts. Their limited functionality implies a lack of advanced features such as direct integration with inventory systems, real-time sales data transmission to a central server, or sophisticated reporting capabilities.

- **Desktop Computers for Administrative Staff:** Administrative personnel in each branch utilize standard desktop computers. These machines are likely used for general office tasks, local record-keeping, and potentially for rudimentary data entry related to sales or inventory.

- **Barcode Scanners for Inventory:** Barcode scanners are present, indicating an understanding of basic inventory tracking. However, their effectiveness is heavily dependent on the software they are integrated with and the processes in place.

2. Current Processes:

The operational methodologies employed at Shopping Center are predominantly manual and reactive, leading to various inefficiencies and limitations.

1. **Sales Process:**

   o **Manual Recording of Sales on Independent Cash Registers:** Each sale transaction is recorded manually by cashiers using independent cash registers. This means that the act of inputting product codes or prices often relies on manual keying or limited barcode scanning without immediate synchronization to a central database. The independent nature of these registers signifies that they operate as standalone units, not constantly communicating with a larger system.

- **Manual Reconciliation of Sales at the End of the Day:** At the close of each business day, the sales data from every independent cash register must be manually reconciled. This laborious process involves tallying receipts, comparing them against register totals, and consolidating figures. This manual reconciliation is prone to human error, consumes significant administrative time, and delays the availability of comprehensive sales figures for the entire hypermarket or chain.
- **Delayed Updating of Stock Levels:** A direct consequence of manual sales recording and reconciliation is the significant delay in updating stock levels. Products sold at the cash register are not immediately deducted from the central inventory count. Instead, inventory updates typically occur after the manual sales reconciliation process is complete, often at the end of the day or even the next morning.

2. **Inventory Management:**

Inventory management at Shopping Center is reactive and paper-based, lacking the precision and foresight needed for optimal stock control.

- **Periodic Physical Inventory to Verify Stock Levels:** The primary method of inventory control involves conducting periodic physical inventories.
- **Stock Transfers Between Stores Documented on Paper:** When stock needs to be moved between different branches of Shopping Center, these transfers are documented manually on paper.
- **Supplier Orders Based on Approximate Estimates:** The process of ordering new stock from suppliers appears to be based on approximate estimates rather than precise, data-driven analysis.

3. **Reporting:**

- **Sales Reports Manually Generated from Data Extractions:** Sales reports are not automatically generated in a ready-to-use format. Instead, they require manual extraction of data from various sources.
- **Significant Delay in Obtaining Consolidated Information:** Due to the manual nature of data extraction and report generation, there is a significant delay in obtaining consolidated information across all branches.
- **Limited Analysis for Decision Making:** The manually generated reports offer only limited analysis. This implies that the reports are primarily descriptive, providing raw numbers rather than insightful analytics.

Chapter II: Modeling

1. Use Case Diagram:

Actors	Use Cases
Cashier	Authenticate into system (login) Scan/Add products to cart Finalize sale (select payment, complete transaction) Print/generate receipt Start/end shifts on cash registers
Manager	Authenticate into system (login) Manage Cashiers accounts (create, edit, delete) View branch reports Manage staff schedules
Admin	Authenticate into system (login) Manage Managers accounts (create, edit, delete) Manage products (add, modify, remove) Manage transfers (approve, track) Manage suppliers Access all system reports

Figure 2.1 shows the UML use-case diagram describing the developed and automated system:

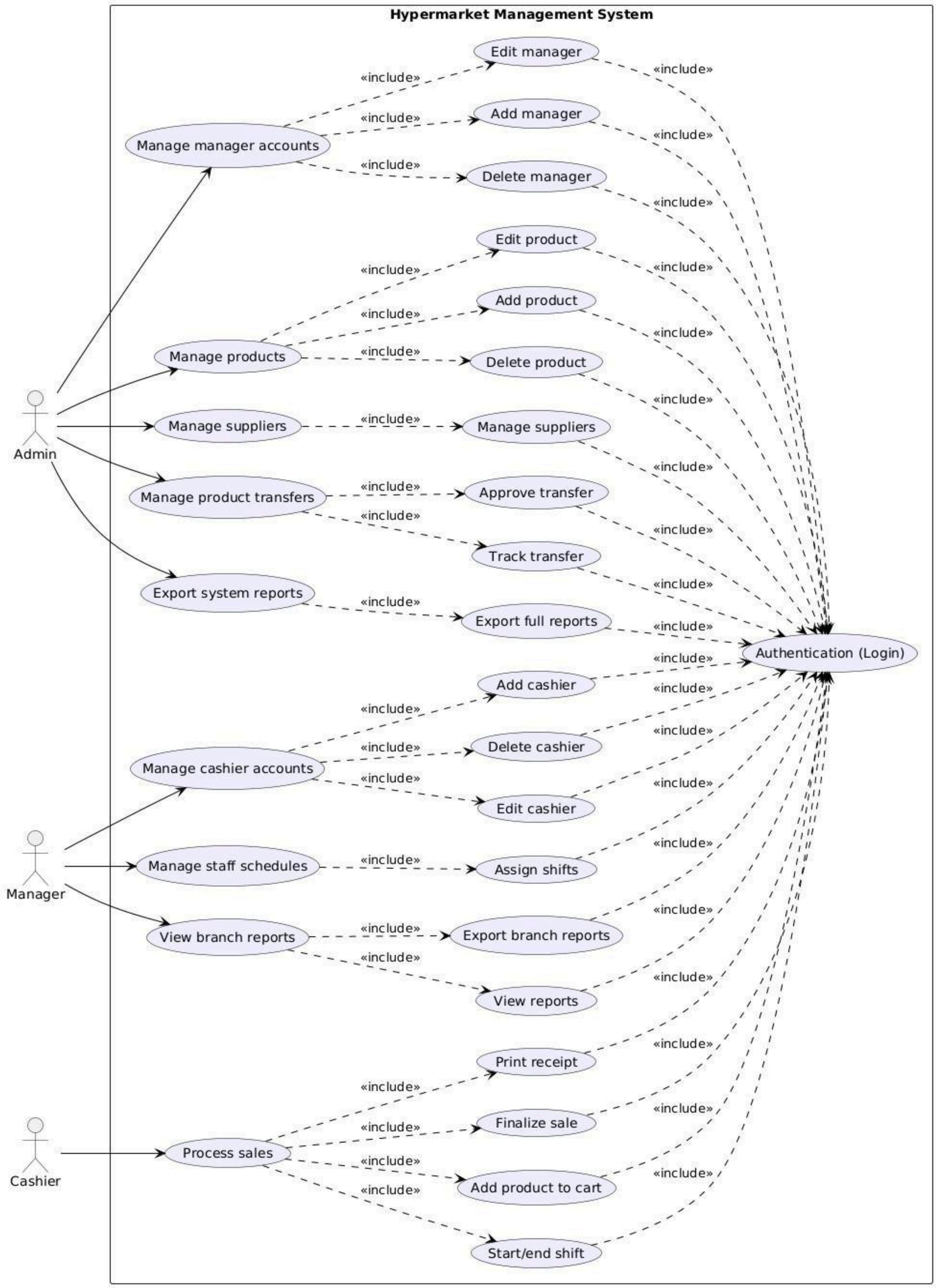


Figure 2.1 - Use case diagram

2. Entity-Relation Diagram

Entities:

1. **Supermarket:** The `Supermarket` entity represents individual supermarket branches, it has the following attributes:
   `ID`: Unique identifier for the supermarket.
   `name`: Name of the supermarket.
   `created_at`: Timestamp of creation.
   `updated_at`: Timestamp of last update.
   `manager_id`: Reference to the user who manages this supermarket.

2. **Locations:** Stores geographical and address-related information of supermarkets, it has the following attributes:
   `ID`: Unique identifier for the location.
   `supermarket_id`: Foreign key linking to a supermarket.
   `street_name`: Street name of the location.
   `state`: State of the location.
   `latitude`: Latitude coordinate.
   `longitude`: Longitude coordinate.
   `create_at`: Timestamp of creation.
   `updated_at`: Timestamp of last update.

3. **Cash_Register:** Represents a register in a supermarket used for processing sales, it has the following attributes:
   `ID`: Unique identifier for the cash register.
   `supermarket_id`: Foreign key linking to a supermarket.
   `created_at`: Timestamp of creation.
   `updated_at`: Timestamp of last update.

4. **Users:** Represents staff members of the supermarket system, it has the following attributes:
   `ID`: Unique identifier for the user.
   `name`: Name of the user.
   `password`: Encrypted password.
   `role`: Role of the user (`admin`, `manager`, `casher`).
   `created_at`: Timestamp of creation.
   `updated_at`: Timestamp of last update.

5. **Sale:** Represents a transaction made at a cash register, it has the following attributes:
   `ID`: Unique identifier for the sale.
   `cash_register_id`: Foreign key linking to a cash register.
   `Payment_method`: Mode of payment (cash, card).
   `created_at`: Timestamp of the sale.
   `updated_at`: Timestamp of last update.

6. **Shifts:** Tracks the working hours of users at specific cash registers, it has the following attributes:
   `ID`: Unique identifier for the shift.

`user_id`: Foreign key linking to a user.
`cash_register_id`: Foreign key linking to a cash register.
`start_at`: Start time of the shift.
`end_at`: End time of the shift (nullable).

7. **Sale_Items:** Tracks products involved in each sale; it has the following attributes:
   `ID`: Unique identifier.
   `sale_id`: Foreign key linking to a sale.
   `product_id`: Foreign key linking to a product.
   `quantity`: Quantity of the product sold.
   `created_at`: Timestamp of creation.
   `updated_at`: Timestamp of last update.

8. **Product:** Represents products available in the supermarket system, it has the following attributes:
   `ID`: Unique identifier for the product.
   `name`: Name of the product.
   `barcode`: Barcode identifier.
   `price`: Price of the product.
   `categorie_id`: Foreign key linking to the product category.
   `supplier_id`: Foreign key linking to the supplier.
   `created_at`: Timestamp of creation.
   `updated_at`: Timestamp of last update.

9. **Categorie:** Stores information about product suppliers, it has the following attributes:
   `ID`: Unique identifier for the category.
   `name`: Name of the category.
   `created_at`: Timestamp of creation.
   `updated_at`: Timestamp of last update.

10. **Supplier:** Represents a transaction made at a cash register, it has the following attributes:
    `ID`: Unique identifier for the supplier.
    `name`: Name of the supplier.
    `phone_number`: Contact number.
    `created_at`: Timestamp of creation.
    `updated_at`: Timestamp of last update.

11. **Stock:** Tracks the stock levels of products at each supermarket, it has the following attributes:
    `ID`: Unique identifier.
    `supermarket_id`: Foreign key linking to a supermarket.
    `product_id`: Foreign key linking to a product.
    `quantity`: Quantity in stock.
    `created_at`: Timestamp of creation.
    `updated_at`: Timestamp of last update.

12. **Transfers:** Tracks movement of products between supermarkets, it has the following attributes:
    `ID`: Unique identifier.

`product_id`: Foreign key linking to a product.
`from_supermarket_id`: Source supermarket.
`to_supermarket_id`: Destination supermarket.
`quantity`: Quantity being transferred.
`status`: Status of transfer (pending, in_transit, delivered).
`created_at`: Timestamp of creation.
`updated_at`: Timestamp of last update.

13. **Supplier_order**: Tracks orders placed with suppliers for specific supermarket.
    attributes:
    ‘ID’:Unique identifier
    ‘product_id’:Foreing key linking to a product(cascade on delete)
    ‘supplier_id’:Foreing key linking to a supplier(cascade on delete)
    ‘supermarket_id’:Foreign key linking to a supermarket (cascade on delete)
    ‘quantity_ordered’:Quantity of the product ordered
    ‘status’:Order status (pending,ordered,shipped,received,cancelled)
    ‘notes’:Optinal notes or comments
    ‘created_at’:timestamp of creation
    ‘updated_at’:timestamp of last update

14. **SaleReports**: Stores metadata about generated sales reports.
    attributes:
    ‘ID’:Unique indentifier
    ‘file_path’:Path to the report file
    ‘report_date’:Date the report was generated
    ‘created_at’:timestamp of creation
    ‘ updated_at’:timestamp of last update

The following schema supports our supermarket management system, providing robust capabilities for tracking inventory, handling sales transactions, managing users and shifts, and overseeing product movements between locations.

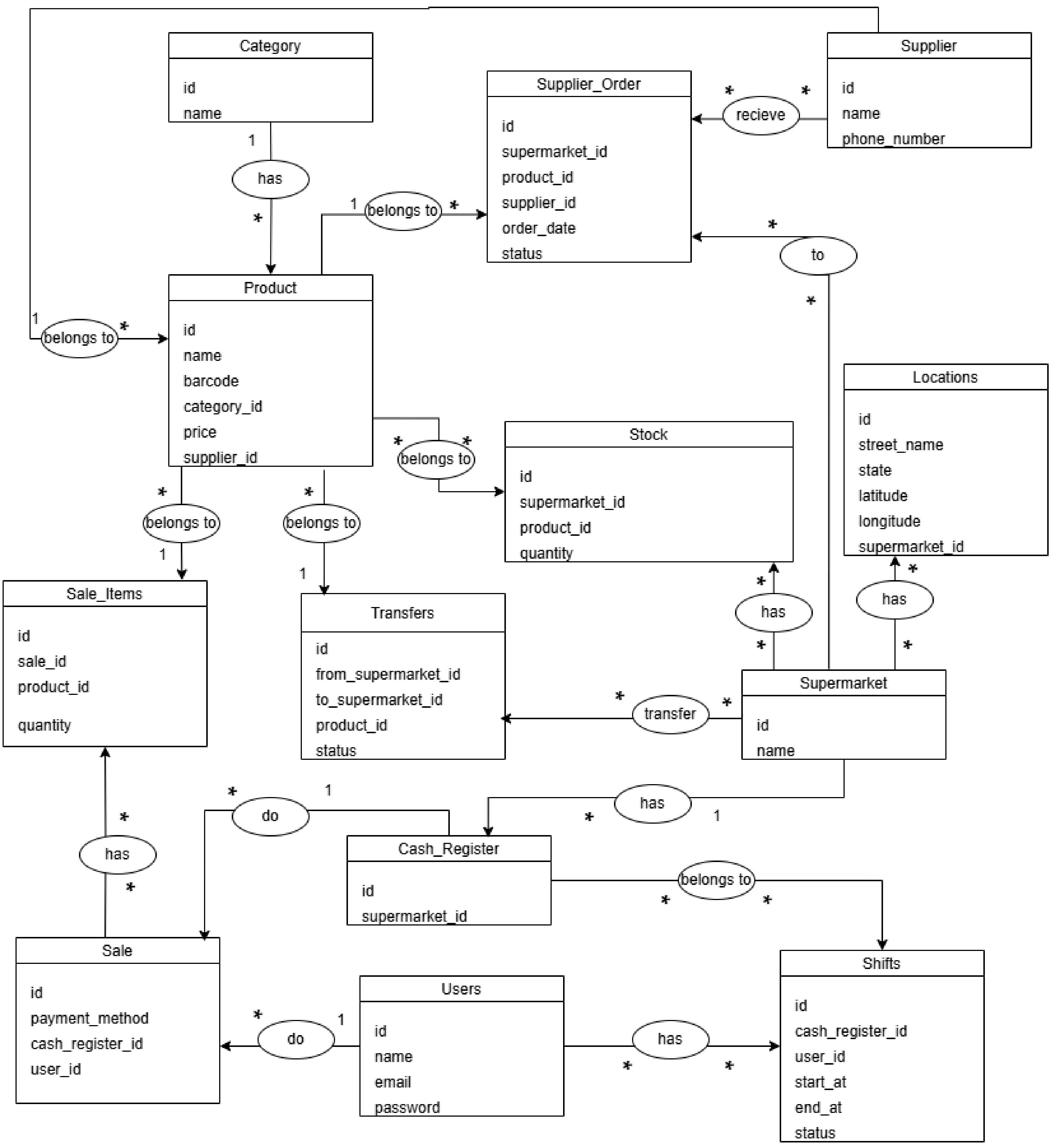


Figure 2.2 - Entitiy/Relation Diagram of the system

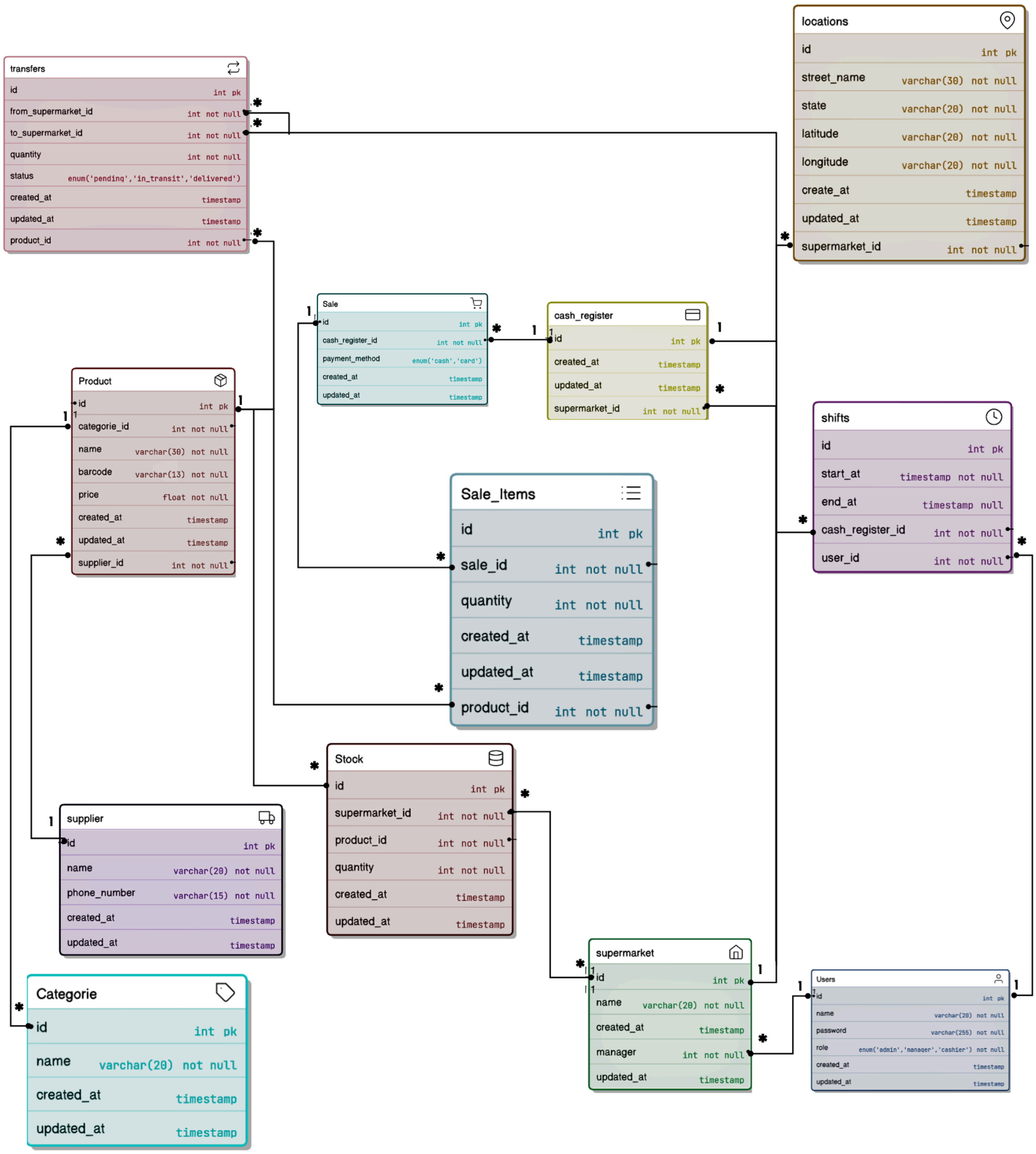


Figure 2.3 - Database description diagram

Chapter III: Proposed system

1. System Architecture

Figure 3.1 describes the architecture of the proposed Hypermarket Sales Management System. The system is based on the MVC (Model-View-Controller) pattern and leverages modern technologies to deliver a scalable, modular, and secure platform. The architectural components are outlined below:

1.1. Administration

Admin Panel (Filament + Laravel)

The administration panel is developed using *Filament*, a *Laravel-based* admin interface framework. It provides admins and managers with powerful tools to manage critical system resources such as *users*, *inventory*, *products*, *suppliers*, *transfers*, and *reports*. Key features include:

- Role assignment and user management
- Resource control (products, stock, categories)
- Audit logging (based on best practices)
- Access to dashboards and analytical widgets

1.2. Sales Management (Point of Sale System)

Cashier Interface (Next.js + React)

A dedicated cashier interface enables rapid and efficient transaction processing through:

- Product registration using barcode scanning
- Cart management, total price calculation
- Receipt generation
- Shift tracking per cashier

Transaction Handling

Each sale:

- Is recorded in the database using the *Sale* and *SaleItems* models
- Triggers real-time inventory updates
- Is securely communicated via RESTful APIs protected by *Laravel Sanctum*

1.3. Inventory Management Real-Time Stock Updates

Inventory levels are instantly updated upon each sale or stock transfer, ensuring system-wide consistency.

Stock Transfers Between Branches

A structured stock transfer system includes:

- Creation of transfer requests

- Manager approvals
- Delivery tracking via status fields (pending, in_transit, delivered)

Product & Supplier Management

Admins can manage:

- Product details (name, barcode, category, price)
- Supplier data (name, contact, linked products)
- Categories for inventory organization

Low Stock Alerts

The system monitors inventory and triggers alerts when product quantities drop below a defined threshold.

1.4. User Management

Role-Based Access Control

User access is governed by roles with strict permission boundaries:

- **Admin:** Full control, user/resource management,approves stock transfers.
- **Manager:** Supervises operations, managing shifts.
- **Cashier:** Handles transactions & printing receipts..

User Lifecycle Management

Admins can:

- Create, edit, or delete user accounts
- Assign roles and associate managers to specific supermarkets
- Ensure secure password storage using hashing

1.5. Reports and Analytics Dashboards (Filament Widgets)

Admins and managers have access to a visual dashboard displaying:

- Daily sales summaries
- Sales trends via line charts
- Top-selling products by volume

Automated Daily Reports

- A scheduled Laravel command (GenerateDailyReport) generates daily sales reports
- Reports are stored in the database and emailed to managers using background jobs

Data Export Capabilities

Filament resources allow exporting data (sales, inventory, user activity) in various formats for further analysis.

1.6. System Communication and Flow RESTful API (Laravel)

All client-server communication uses secure RESTful APIs, which handle:

- User authentication
- Sales transactions
- Inventory queries
- Report generation

Asynchronous Operations (Laravel Queue)

Tasks like report generation and email dispatch are processed asynchronously to improve system performance and frontend responsiveness.

1.7. Desktop Application (Electron Integration)

To ensure platform independence and maximize accessibility across various operating systems, our Hypermarket Management System includes a cross-platform desktop application built using Electron. This component plays a crucial role in enabling local deployment and streamlined access to core functionalities without relying on a web browser.

Key Features of the Desktop App:

- **Unified Access Point:** The Electron-based desktop app serves as a centralized access point for all user roles—cashiers, managers, and administrators. It packages the web-based cashier interface (React + Next.js) and admin panel (Filament + Laravel) into a standalone executable application.
- **Offline-Ready Frontend:** Through local caching and service workers, the Electron app is designed to support offline mode for basic operations such as scanning products, cart management, and generating receipts. Data synchronization is handled once the connection is restored.
- **System Tray and Auto-Launch:** For convenience, the application minimizes to the system tray and supports auto- launch on system startup, ensuring availability for daily retail operations without additional user intervention.
- **Secure API Communication:** All interactions with the Laravel backend are secured through RESTful APIs using Laravel Sanctum, ensuring authentication and encrypted data exchange across all operations including sales, inventory, and report generation.
- **Multi-OS Compatibility:** The application is compiled and packaged for Windows, Linux, and macOS using Electron Builder, enabling seamless deployment across diverse hardware environments within the hypermarket branches.

Advantages:

- Reduces dependence on browsers and external links.
- Offers a consistent and user-friendly experience across devices.
- Simplifies the installation and update process for non-technical users.
- Enhances performance by bundling required resources locally.

By integrating Electron, the Hypermarket Management System bridges the gap between web functionality and desktop convenience, delivering a polished and dependable user experience critical for high-frequency retail environments.

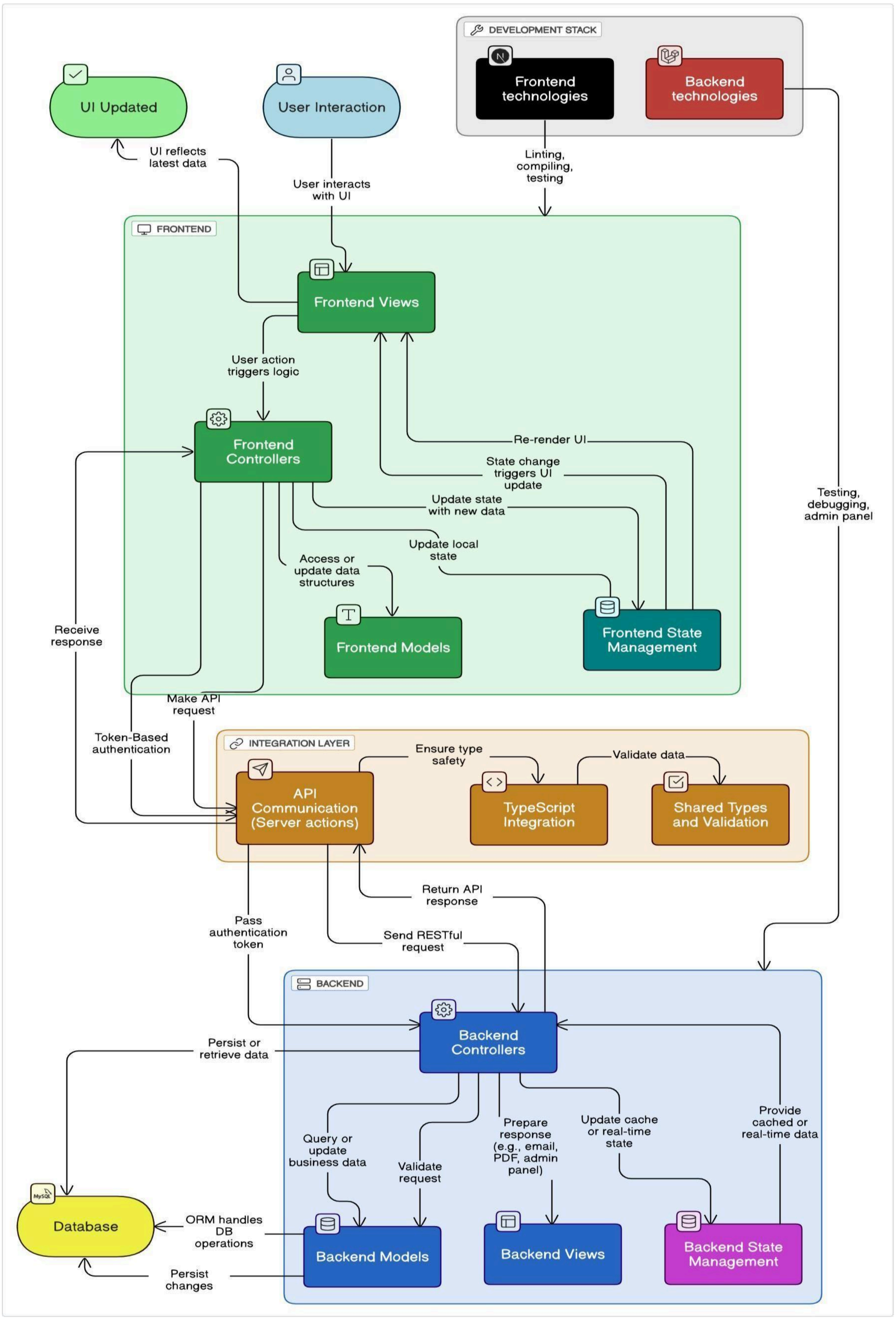


Figure 3.1 - MVC flow

Chapter IV: Implementation Details

1. Sales Processing

The cashier frontend, developed using Next.js, communicates with the Laravel backend through secure API requests authenticated by Laravel Sanctum. During a sales transaction, the frontend sends details such as selected products, their quantities, and the chosen payment method.

On the backend, the `CashierController` handles the transaction. It performs several operations:

- Validates the payment method and checks product availability using the `Stock` model.
- Calculates the total amount due, including any taxes or discounts.
- Updates stock levels by reducing quantities for sold products.
- Creates a record of the transaction using the `Sale` model and logs each item using the `SaleProduct` model.
- Wraps all these operations within a database transaction to ensure data integrity, meaning if any part fails, the entire process is rolled back.

All API endpoints related to sales are protected using Sanctum tokens, ensuring only authenticated users can access them.

2. Inventory Updates

Inventory is automatically adjusted in response to two main events: sales and completed transfers.

When a sale is completed, the system decreases the product quantities in the corresponding supermarket's inventory using Laravel's Eloquent relationships with the `Stock` model.

When a stock transfer is completed (marked as delivered), the system reduces stock from the source supermarket and increases it at the destination.

Additionally, the system includes a scheduled task, implemented using Laravel's scheduler, that compares current stock levels with predefined minimum thresholds. If the quantity drops below the threshold (e.g., 30 units), the system automatically creates a transfer request using the `ShortestPathService`. This service calculates the shortest route between supermarkets using Dijkstra's algorithm and selects the nearest location with available stock.

Transfer requests remain in a "Pending" state until approved. **Only administrators** have permission to approve transfers and change their status to "In Transit".

3. Role-Based Permissions

The system enforces strict access control using Laravel's authorization mechanisms and Filament's role-resource integration.

There are three primary roles:

- **Admin**: Full access to all system functions, including reporting, user management, and transfer approvals.
- **Manager**: Access to inventory control and supermarket-specific operations, but cannot approve transfers.
- **Cashier**: Limited to sales processing and basic reporting.

Roles determine which Filament resources a user can view or edit, and also restrict access to backend API routes. Role assignments are managed through the `UserResource` interface in Filament, and the frontend uses the `getRouteForRole` function to direct users to their appropriate dashboard.

4. **Report Generation**
Sales data is processed and summarized using scheduled Laravel commands. A dedicated Artisan command (`DailyReports`) runs on a daily schedule. It aggregates sales by supermarket, calculates revenue, and generates structured reports using the `SaleReport` model.

   Reports are stored in JSON format and sent via email to relevant stakeholders using queued jobs (`SendDailyReportEmailJob`) for asynchronous delivery. The `SaleReportResource` interface in Filament allows users to view, filter, and export these reports as needed.

5. **Admin Panel Configuration**
The admin interface is built using Filament and managed via the `AdminPanelProvider`. It offers a structured dashboard with branding options, dark mode, and authentication settings.

6. **Tracking and Transfer Optimization**

   The system uses **Dijkstra's algorithm** to calculate the shortest route between supermarkets when generating stock transfer requests. Implemented in the `ShortestPathService`, it:

   - Identifies supermarkets with available stock
   - Calculates the shortest delivery path
   - Selects the nearest source location

   Once approved by an **admin**, the transfer is marked as **In Transit**. After delivery confirmation, inventory levels are updated for both supermarkets. This ensures efficient, automated, and optimized stock distribution.

Chapter V: Development Environment

Our system is built upon a modern and robust stack, carefully selected to ensure high performance, scalability, and a consistent user experience across all platforms.

Backend (Server-side Logic):

- **PHP 8.3 with Laravel Framework (v12):** This serves as the backbone of our application, providing a robust and elegant structure for handling business logic and API services.
- **Filament:** Utilized for the administration panel, enabling efficient UI, resource management, and dashboard functionalities.
- **Laravel Sanctum:** Powers our API authentication, ensuring secure communication.
- **Laravel Queue:** Handles background tasks, such as report generation, improving application responsiveness.

Frontend (User Interface):

- **TypeScript:** Provides strong static typing across the entire codebase, significantly enhancing reliability and maintainability.
- **Next.js Framework (v15.3):** Our core frontend framework, leveraging server-side rendering (SSR) and static site generation (SSG) for optimal performance.
- **React 19:** The foundational library for building interactive, component-based user interfaces.
- **Zustand:** A lightweight solution for efficient state management within React applications.

Styling:

- **TailwindCSS (v4.0):** A utility-first CSS framework applied consistently across both the Filament- based admin panel and the cashier frontend for rapid and responsive design implementation.

Database - SQLite :

Our chosen relational database management system, providing robust data storage and retrieval capabilities (specific version dependent on deployment environment).

Deployment & Version Control:

- **Electron Builder:** Used for packaging and distributing the application as a high-performance, cross- platform **desktop application** across Windows, Linux, and macOS.
- **Git & GitHub:** Employed for version control, facilitating collaborative development and tracking changes.

Conclusion

The hypermarket sales management system developed for “Shopping Center” represents a significant advancement compared to the manual processes and disparate systems previously used. This integrated solution meets the set objectives by offering:

- Complete automation of sales processes, reducing errors and improving efficiency.
- Real-time inventory tracking, allowing better inventory management.
- A smooth stock transfer system between different branches.
- Automatic generation of daily reports, providing managers with valuable information for decision making.
- Role-based access control, ensuring data security and integrity.

Encountered difficulties:

During the development of our Hypermarket Management System, we faced several technical and organizational challenges:

- **API Integration Issues:** Ensuring smooth and secure communication between the Electron frontend and the Laravel backend required careful handling of CORS policies and authentication using Sanctum.
- **State Synchronization:** Maintaining real-time synchronization of inventory data and user sessions across multiple branches posed a challenge, especially when handling concurrent sales and transfers.
- **User Role Management:** Implementing a flexible yet secure role-based access control system demanded fine-grained permission configuration and rigorous testing.
- **Desktop Deployment:** Packaging the Electron app across different platforms (Windows, Linux, macOS) required additional configuration and troubleshooting to support native features like auto- launch and updates.

Future Perspectives

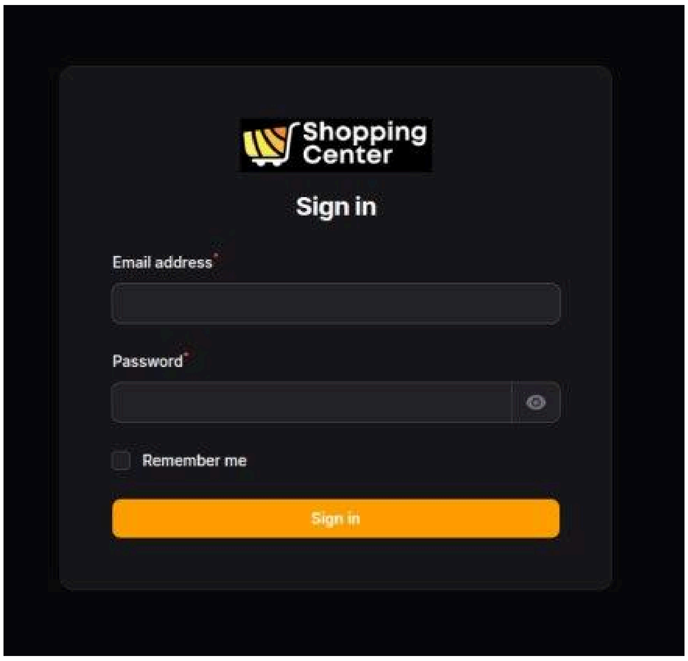
Looking ahead, we envision several improvements and extensions to the system:

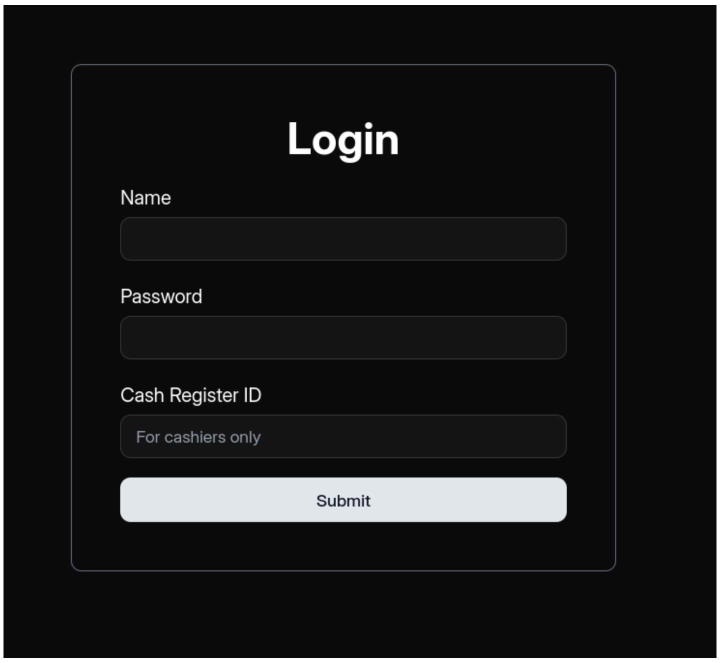
- **Mobile App Integration:** Building a companion mobile app for on-the-go access and remote management by managers or suppliers.
- **Advanced Analytics:** Integrating machine learning models to predict sales trends and optimize stock levels.
- **Multilingual Support:** Expanding the system to support multiple languages, increasing accessibility for a broader range of users.
- **Cloud-Based Deployment:** Migrating to a cloud-based infrastructure to enhance scalability and enable real-time global access to data and reports.

Appendices

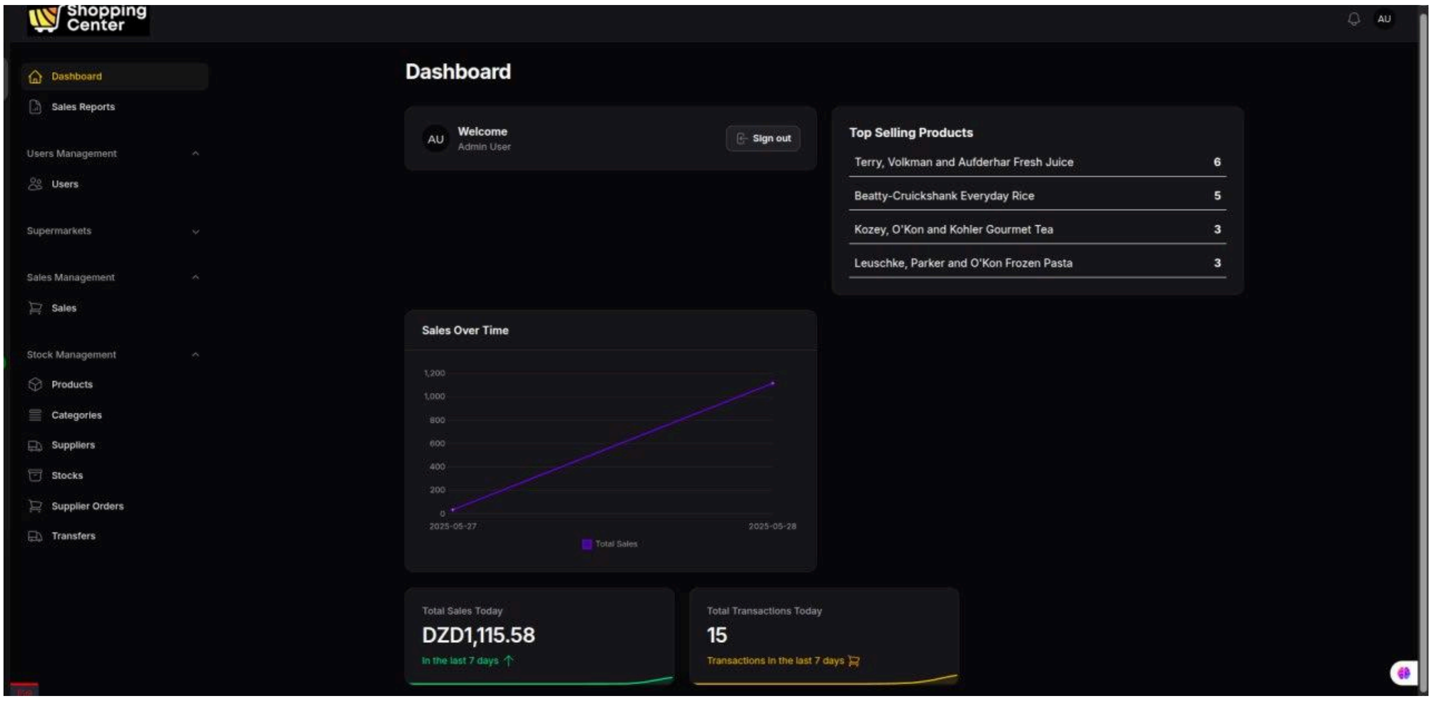
Screenshots

Login pages :





Admin's Dashboard:



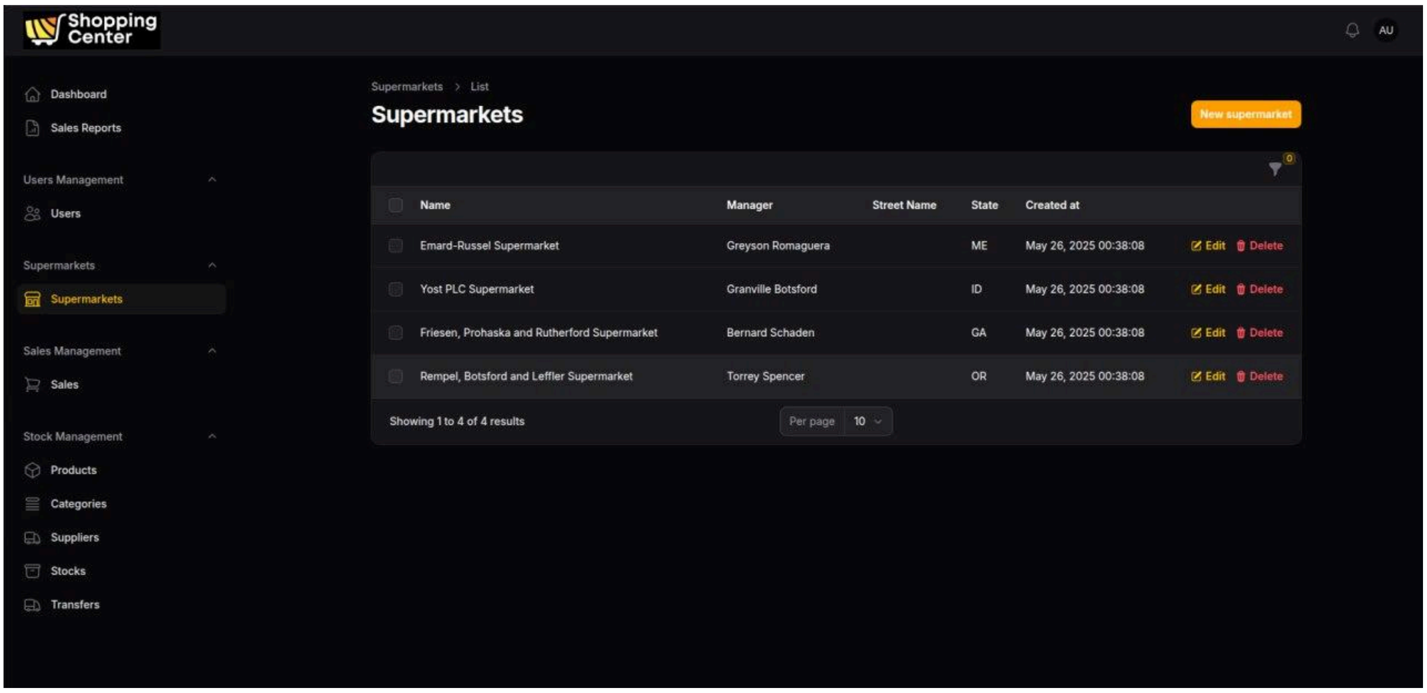
Top Selling Products

Terry, Volkman and Aufderhar Fresh Juice	6
Beatty-Cruickshank Everyday Rice	5
Kozey, O'Kon and Kohler Gourmet Tea	3
Leuschke, Parker and O'Kon Frozen Pasta	3

Total Sales Today
DZD1,115.58
In the last 7 days

Total Transactions Today
15
Transactions in the last 7 days

Admin’s Supermarkets (Branches) page:



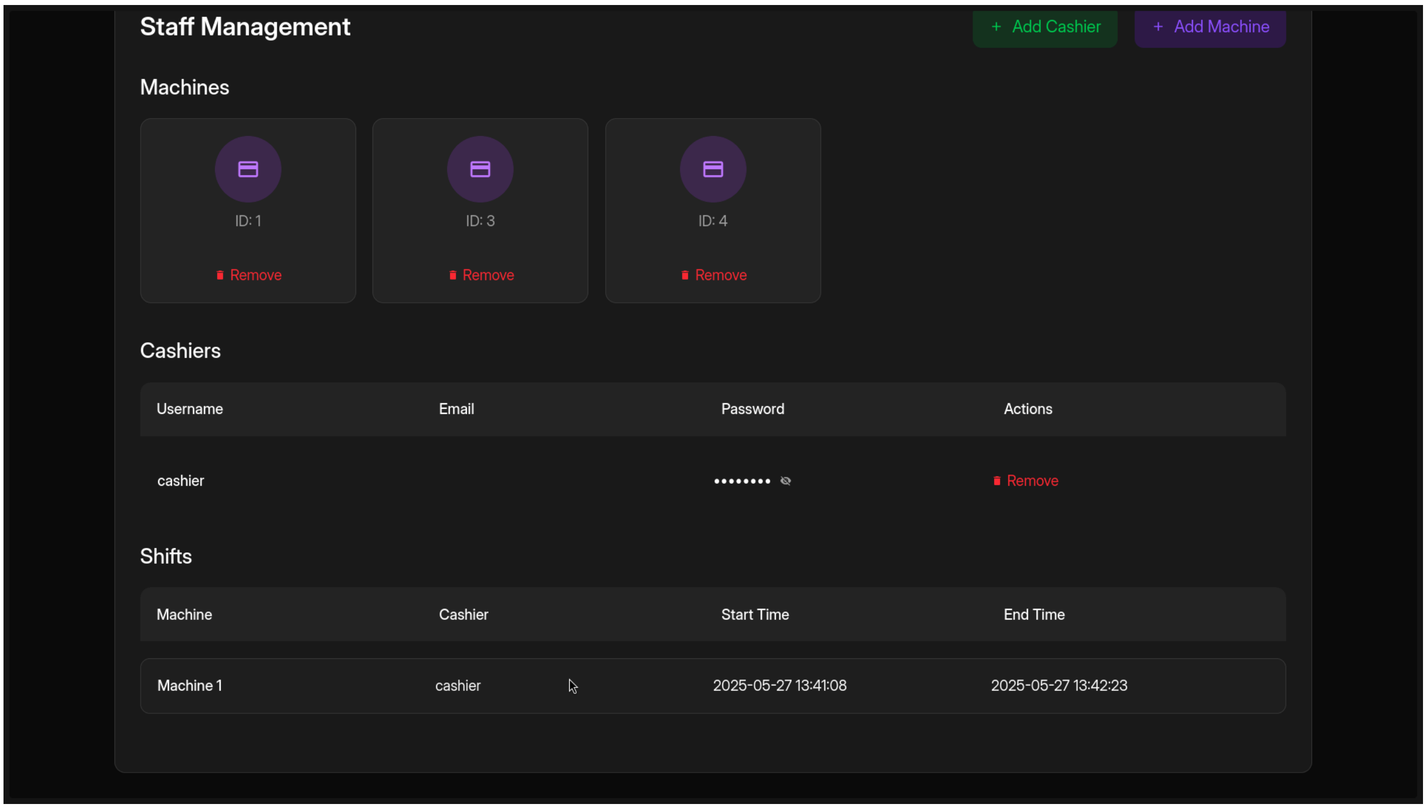
Supermarkets

Name	Manager	Street Name	State	Created at	
Emard-Russel Supermarket	Greyson Romaguera		ME	May 26, 2025 00:38:08	Edit Delete
Yost PLC Supermarket	Granville Botsford		ID	May 26, 2025 00:38:08	Edit Delete
Friesen, Prohaska and Rutherford Supermarket	Bernard Schaden		GA	May 26, 2025 00:38:08	Edit Delete
Rempel, Botsford and Leffler Supermarket	Torrey Spencer		OR	May 26, 2025 00:38:08	Edit Delete

Showing 1 to 4 of 4 results

Per page 10

Manager’s Dashboard:



Staff Management

Machines

ID: 1 Remove

ID: 3 Remove

ID: 4 Remove

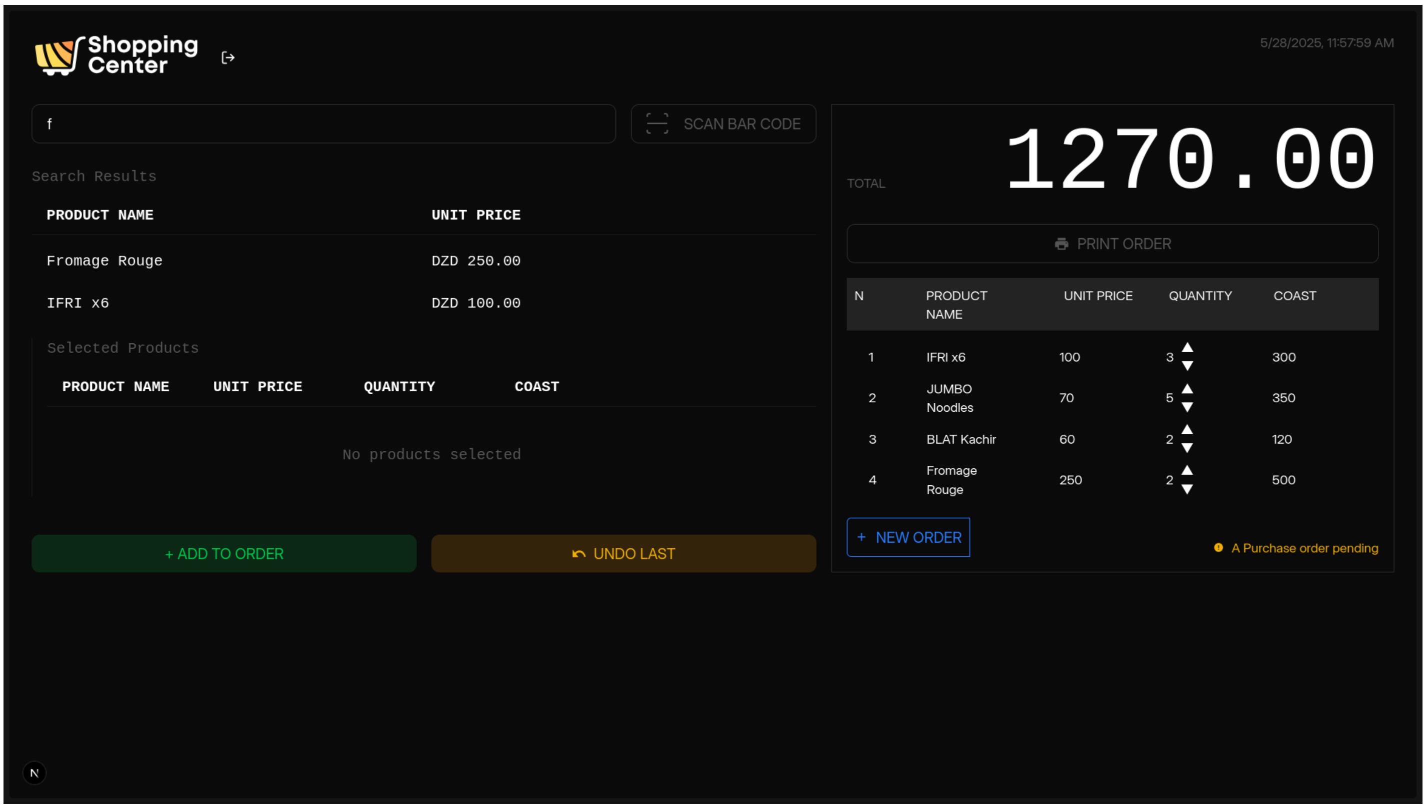
Cashiers

Username	Email	Password	Actions
cashier		••••••••	Remove

Shifts

Machine	Cashier	Start Time	End Time
Machine 1	cashier	2025-05-27 13:41:08	2025-05-27 13:42:23

Cashier's Dashboard:



Search Results

PRODUCT NAME	UNIT PRICE
Fromage Rouge	DZD 250.00
IFRI x6	DZD 100.00

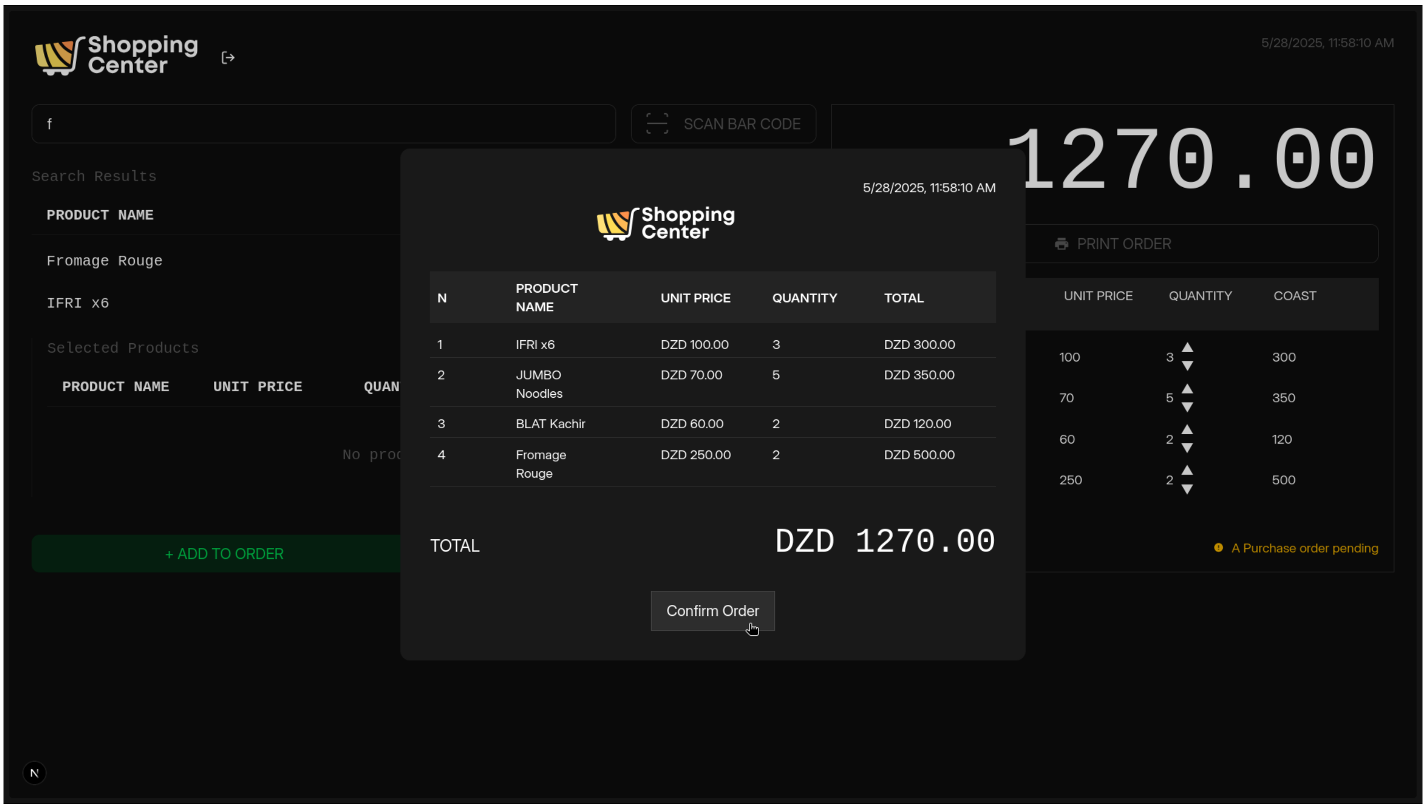
Selected Products

PRODUCT NAME	UNIT PRICE	QUANTITY	COAST

No products selected

N	PRODUCT NAME	UNIT PRICE	QUANTITY	COAST
1	IFRI x6	100	3	300
2	JUMBO Noodles	70	5	350
3	BLAT Kachir	60	2	120
4	Fromage Rouge	250	2	500

Cashier's order confirmation (Receipt Printing):



N	PRODUCT NAME	UNIT PRICE	QUANTITY	TOTAL
1	IFRI x6	DZD 100.00	3	DZD 300.00
2	JUMBO Noodles	DZD 70.00	5	DZD 350.00
3	BLAT Kachir	DZD 60.00	2	DZD 120.00
4	Fromage Rouge	DZD 250.00	2	DZD 500.00

TOTAL DZD 1270.00

API Endpoints List

Authentication Endpoints
These endpoints manage user login sessions and retrieve authenticated user data.

- `POST /api/login`
  Logs a user into the system.
  Controller: AuthController@login

- `POST /api/logout`
  Logs out the currently authenticated user.
  Controller: AuthController@logout

- `GET /api/user`
  Retrieves information about the currently authenticated user.
  Controller: AuthController@get_user

Sales Processing Endpoints
Endpoints used by the cashier to perform sales operations.

- `POST /api/search`
  Searches for products by name or barcode.
  Controller: CashierController@search

- `POST /api/ticket`
  Generates a sales ticket and processes the transaction.
  Controller: CashierController@generate_ticket

Cash Register Management
Endpoints for managing cash registers, typically used by administrators or managers.

- `POST /api/user/addCashRegister`
  Adds a new cash register to the system.
  Controller: ManagerController@AddCashRegister

- `GET /api/user/cashRegisters`
  Lists all existing cash registers.
  Controller: ManagerController@showAllCashRegisters

- `DELETE /api/user/cashRegisters/{id}`
  Deletes a specific cash register by ID.
  Controller: ManagerController@deleteCashRegister

Cashier Management
Endpoints to manage cashier accounts and their details.

- `POST /api/user/addCashier`
  Creates a new cashier user.
  Controller: ManagerController@AddCashier

- `GET /api/user/cashiers`
  Lists all cashiers.
  Controller: ManagerController@showAllCashiers

- `DELETE /api/user/cashiers/{id}`
  Deletes a specific cashier by ID.
  Controller: ManagerController@deleteCashier

- `PUT /api/user/cashiers/{id}`
  Updates the details of a specific cashier.
  Controller: ManagerController@editCashier

Shift Management
Endpoint for viewing available work shifts.

- `GET /api/user/shifts`
  Lists all available or configured shifts.
  Controller: ManagerController@showAllShifts

References

Main websites consulted between March and Mai 2025

- **Laravel Documentation – https://laravel.com/docs**
- **Filament Admin Panel – https://filamentphp.com**
- **React Official Docs – https://react.dev**
- **Next.js Documentation – https://nextjs.org/docs**
- **Tailwind CSS Docs – https://tailwindcss.com/docs**
- **Zustand State Management – https://zustand-demo.pmnd.rs**
- **SQLite Documentation – https://www.sqlite.org/docs.html**
- **Electron Documentation – https://www.electronjs.org/docs**