AN-Najah National University

Faculty of Engineering & Information Technology

Computer Engineering Department

First semester 2026/2027

Course: Graduation project I

MARS

Project made by:

Ahmad Tubaila (ID: 12028317)

Abdulkareem Abuzahra (ID: 12112886)

Under the supervision of: **Dr. Sufian Samara**

"Presented in partial fulfillment of the requirements for Bachelor degree in Computer Engineering."

January 2026-2027

Table of Contents

Disclaimer

This report was prepared by students of the Computer Engineering Department at An-Najah National University. Apart from minor editorial corrections, it has not been formally revised or edited based on assessment feedback and may therefore contain errors in language and content.

The views, findings, and recommendations presented here are solely those of the student authors. An-Najah National University is not responsible for the content of this report or for any consequences arising from its use for purposes other than that for which it was originally intended.

Acknowledgment

We extend our deepest thanks to our project supervisor, **Dr. Sufian Samara**, for his constant guidance, support, and technical expertise. His insight was pivotal in shaping our work and helping us navigate significant engineering and design challenges.

We are also sincerely grateful to the **Computer Engineering Department at An-Najah National University** for providing the essential facilities, tools, and environment needed to complete this project. The access to labs and resources enabled us to effectively prototype and test our ideas.

A special thank you to our families and friends for their unwavering patience and encouragement, which kept us focused and motivated, especially during the most demanding phases of the work.

Finally, we appreciate all our classmates and professors for their valuable suggestions, feedback, and moral support throughout this journey.

This project and its documentation would not have been possible without the generous contributions of everyone mentioned above.

Abstract

Modern electronics retail stores often rely on fragmented tools and manual procedures to manage sales, inventory, payments, and employee activities. This separation increases operational complexity, introduces inconsistencies in records, and limits effective oversight. To address these challenges, this project presents MARS, an integrated software system that unifies customer-facing e-commerce functionality with internal management operations for an electronics store.

MARS is implemented as a full-stack application composed of React-based web and mobile clients, a Node.js and Express backend, and a MySQL relational database. The system provides centralized product management, point-of-sale and order processing, contract and payment recording, employee duty-hour tracking, and internal project and task management. Role-based access control ensures that customers, store workers, and administrative users interact with the system according to their responsibilities. In addition, the system includes prototype services for optical character recognition (OCR) to extract data from receipt images and a lightweight recommendation component to support product suggestions during sales workflows.

The project results in a functional prototype that demonstrates how a unified platform can reduce data duplication, improve record consistency, and streamline routine retail operations. Design choices emphasize modularity, separation of concerns, and maintainability, while acknowledging constraints related to security hardening, scalability, and data-driven model accuracy.

Overall, MARS serves as a practical academic implementation that applies software engineering principles to a real-world problem. The system provides a solid foundation for future enhancements, including improved authentication mechanisms, cloud-based deployment, and more advanced OCR and recommendation models.

Chapter one

Introductory

1.1 Statement of the Problem

Many electronics retail stores still rely on a mix of separate software tools and manual procedures to manage daily operations. Point-of-sale transactions, contract records, payment tracking, and employee duty management are often handled independently or, in some cases, maintained on paper. This fragmented approach leads to operational inefficiencies, increases the likelihood of human error, and makes it difficult to obtain a clear and accurate overview of store activities.

Because data is not centralized, information sharing between employees and management is often delayed or inconsistent. As a result, reporting becomes more time-consuming, and customers may experience issues such as mismatched records or delays during transactions. These challenges highlight the need for a single system that integrates operational and commercial functions into one coherent platform.

1.2 Objectives of the Work

The objective of this project is to design and implement an integrated software system that supports both customer-facing activities and internal management operations for an electronics store. The main objectives of the project are as follows:

- develop a unified platform that combines online product browsing and purchasing with internal store management tools.
- support point-of-sale operations so that in-store sales are recorded within the same system used for online transactions.
- centralize contract and payment records in order to reduce data duplication and improve financial accuracy
- provide basic employee management features, including duty-hour recording and task assignment related to store operations.
- enforce role-based access control so that customers, workers, and administrators interact with the system according to their responsibilities.

Together, these objectives aim to simplify daily operations, reduce reliance on manual coordination, and provide store management with clearer visibility over sales, payments, and staff activities.

1.3 Scope of the Work

The scope of this project is intentionally limited to the essential operational needs of a medium-sized electronics retail store. The system includes web and mobile interfaces designed primarily for staff usage and is intended for local or on-premises deployment. The implementation focuses on core functionalities such as product management, sales and point-of-sale recording, contract and payment handling, employee duty-hour tracking, and basic internal task coordination.

Features such as large-scale enterprise integration, advanced analytics, or extensive third-party service connections are outside the scope of this work. This constrained scope allows the project to concentrate on correctness, clarity of design, and practical usability within an academic context.

1.4 Significance of the Work

This project has both practical and academic significance. From a practical standpoint, integrating store operations into a single system can reduce repetitive data entry, improve record consistency, and simplify financial and employee tracking. Centralized information enables management to make better-informed decisions and reduces the effort required for reconciliation and reporting. Customers also benefit from more consistent and reliable transaction handling.

From an academic perspective, the project demonstrates the application of software engineering principles to a real-world problem. It illustrates system decomposition, role-based access control, and the evaluation of design trade-offs when integrating multiple functional components into a single application. The work therefore serves as a concrete example of how theoretical concepts are applied in practical software development.

Chapter Two

Constraints, Standards/ Codes and Earlier course work

2.1 Constraints and Limitations

The MARS project was developed under constraints typical of an academic graduation project. Limited time required prioritizing essential system functionality, with core retail operations implemented first. Advanced features, large-scale optimizations, and extensive refactoring were postponed to ensure timely completion within the project schedule. In addition, the small team size influenced design choices, favoring clear and maintainable solutions over complex architectural patterns.

Deployment was designed primarily for local or small-scale server environments. The selected technology stack—Node.js with a MySQL database—provides stable operation for such settings but does not inherently support advanced scalability or high-availability features associated with cloud-native systems. Consequently, topics such as distributed deployment, automated backups, and horizontal scaling were considered conceptually but not implemented.

Some system components depend heavily on data availability and quality. In particular, the OCR and recommendation features rely on limited datasets available during development. Their performance is therefore constrained, and the project does not claim production-level accuracy for these components. Instead, they are implemented as functional prototypes that demonstrate feasibility and integration within the overall system.

Security considerations were addressed at a level appropriate for demonstration and academic evaluation. The system includes password hashing and role-based access control, but advanced security measures such as penetration testing, secure credential rotation, and enterprise-grade monitoring were beyond the project's scope.

Testing focused primarily on verifying functional correctness of key workflows. Comprehensive automated testing, continuous integration pipelines, and stress testing were not implemented due to time and resource limitations. These aspects are identified as potential areas for future improvement.

Overall, MARS is presented as a well-structured academic prototype. The project emphasizes clarity, correctness, and maintainability, while explicitly acknowledging the limitations imposed by available resources and development time.

2.2 Standards, Tools, and Development Practices

The project follows commonly accepted software engineering standards and development practices to ensure consistency and reproducibility. Established tools and frameworks were selected to support reliable development and clear system organization.

On the backend, Node.js with the Express framework was used to implement the server-side API. The application follows a RESTful design approach, separating routing, business logic, and data access into distinct modules. This structure improves readability and simplifies maintenance.

The frontend and mobile clients were developed using React with Vite as the build and development tool. A shared approach to service modules and session handling was adopted to maintain consistency between the web and mobile interfaces. Client-side logic focuses on presentation, user interaction, and communication with the backend API.

MySQL was selected as the relational database management system. Database access is organized through model modules that encapsulate SQL queries and handle mapping between relational records and application data structures. This approach centralizes data access logic and reduces duplication.

RESTful API design principles were applied throughout the system. Resources such as products, employees, contracts, payments, projects, and tasks are exposed through clearly defined endpoints using standard HTTP methods. During development, Postman was used to test API endpoints and validate request and response behavior.

Configuration values, including database credentials and server ports, are managed through environment variables. This practice separates configuration from source code and supports safer local testing.

Role-based access control governs system usage. User roles—including customer, worker, admin, and co-admin—determine permitted actions and visible interface elements. Enforcement occurs at both the backend route level and the frontend interface to ensure consistent access restrictions.

The project emphasizes modular code organization, incremental development, and basic input validation. Third-party libraries were selected based on stability and community support, such as bcrypt for password hashing and multer for file uploads. Overall, the focus remains on clarity, maintainability, and explicit system behavior rather than experimental optimizations.

2.3 Earlier Coursework and Knowledge Foundations

The design and implementation of MARS were informed by knowledge gained in earlier coursework throughout the software engineering program.

Concepts from software engineering courses—such as requirements analysis, system architecture, modular design, and documentation—guided the overall project structure and development process. Iterative implementation and clear separation of responsibilities reflect these principles.

Database-related coursework contributed to the design of normalized relational schemas and the use of structured SQL queries. Understanding of transactions, data integrity, and basic security considerations influenced how persistent data is managed within the system.

Web programming courses provided a foundation in HTTP communication, client–server interaction, and component-based frontend development. These skills supported the implementation of responsive user interfaces and API-driven workflows.

Introductory security topics informed decisions regarding authentication, password storage, and role-based authorization. These concepts shaped the selection of hashing techniques and access control mechanisms used in the project.

Basic exposure to artificial intelligence and machine learning concepts enabled the prototyping of OCR and recommendation features. An understanding of supervised learning, training data dependency, and evaluation limitations helped establish realistic expectations for these components.

Together, these academic foundations provided the technical and conceptual basis required to balance system ambition with practical constraints. They also supported a disciplined approach to design, implementation, and documentation throughout the project.

Chapter Three

Literature Review

3.1 E-commerce and Integrated Retail Systems

Contemporary retail software spans a spectrum from closed, monolithic point-of-sale suites provided by commercial vendors to modular, API-driven platforms assembled from interoperable components. Empirical studies and practitioner reports consistently highlight the operational benefits of consolidating product, transaction and inventory data into a single canonical model: reduced reconciliation effort, more accurate reporting, and simpler promotion and pricing strategies across channels. For small and medium-sized retailers, research suggests that the most successful deployments are those that prioritize usability for in-store personnel while maintaining a consistent catalog model for online sales channels.

Key design patterns in this domain include a resource-oriented API (REST) layered above canonical domain models, a synchronization layer or event log for cross-client consistency, and a thin client architecture that delegates business rules to the server. The MARS project follows these patterns by centralizing product and transactional data in a relational store and exposing resource-based endpoints for clients. The academic literature also cautions about migration and versioning challenges when evolving a shared data model; establishing clear contracts and backward-compatible APIs is recommended when multiple clients are in active use.

3.2 Point-of-Sale and Inventory Integration

Point-of-sale (POS) systems occupy a critical junction between customer interaction and back-office inventory control. Scholarly and technical discussions classify inventory integration strategies broadly into event-driven models—where sales and stock movements are captured as immutable events—and transactional models that atomically reserve and update stock counts within database transactions. Event-driven approaches facilitate eventual consistency and support analytic event streams, while transactional approaches simplify correctness guarantees for stock levels during concurrent operations.

For retail contexts with modest concurrency and limited infrastructure complexity, the literature often recommends a pragmatic transactional model with conservative validation: check availability, decrement stock, and record the sale within a single controlled operation. This approach simplifies reasoning about correctness and aligns with audit and reporting requirements. MARS adopts conservative safeguards: it enforces availability checks before finalizing sales, records actor attribution for auditability, and uses server-side logic to preserve inventory integrity. The literature further emphasizes reconciliation mechanisms—periodic audits, alerts on negative stock, and administrative correction workflows—as essential complements to automated updates.

3.3 OCR in Retail Workflows

Optical character recognition (OCR) is an established tool for automating the transcription of printed and scanned text from receipts, invoices and other transactional documents. The academic literature and industrial case studies demonstrate that OCR accuracy is a function of multiple variables: image resolution and noise, variability in document layouts, language and font diversity, and the presence of graphical elements. Consequently, domain adaptation—training or fine-tuning OCR models on representative datasets—substantially improves extraction quality.

In practice, many retail deployments implement hybrid pipelines that combine automated OCR extraction with human verification for low-confidence outputs. Such pipelines increase throughput while mitigating the cost of manual entry errors. Best practices include surface-level confidence scoring, structured output (line-item parsing), and UIs that present OCR results alongside original images for rapid validation. MARS implements a prototype OCR workflow that emphasizes explainability and verification: extracted text is linked to payment or order records and low-confidence items are flagged for review. The literature also recommends logging extraction errors and tracking correction actions to guide iterative improvements in models and preprocessing steps.

3.4 Recommendation Systems for Retail

Recommendation systems in retail range from low-cost heuristics—such as popularity ranking and frequently-bought-together lists—to sophisticated machine-learning models including collaborative filtering, factorization machines and neural recommenders. Empirical research shows that when historical interaction data is sparse, simple item-based similarity or association-rule techniques often outperform complex models that overfit limited data. As datasets grow, however, data-driven models yield stronger personalization and higher long-term engagement when coupled with proper evaluation strategies.

The literature stresses iterative development: deploy simple baselines first, instrument outcomes with offline and online metrics, and progressively introduce more advanced models. Explainability and user control are recurring themes; interpretable recommendations improve trust and staff adoption in retail contexts. For MARS, starting with heuristic-based recommendations is appropriate given the prototype's dataset scale; the project's future roadmap aligns with literature guidance to introduce collaborative or hybrid recommenders once sufficient interaction data and evaluation pipelines are in place.

3.5 Role-based Access and Security Practices

Access control is a foundational security concern in multi-user systems. The literature promotes several enduring principles: enforce authorization at the service boundary rather than relying solely on client-side checks, adopt least-privilege role assignments, and implement comprehensive logging and auditing to detect misuse. Token-based authentication (with signed tokens or session mechanisms) and transport-layer protection (HTTPS/TLS) are standard recommendations to protect credentials and session integrity.

For prototype systems, developers often balance usability and security by combining client-side role gating (for user experience) with server-side enforcement (for correctness). MARS follows this approach: the client adapts its interface to the current user's role for convenience, while the server enforces role checks on all sensitive routes. The literature further advises that role design should be minimal and well-documented, and that privilege escalation vectors be periodically reviewed—practices that reduce the attack surface and ease later audits or compliance efforts.

3.6 Architecture and Tooling Choices in Similar Projects

Projects developed in academic settings and small teams typically prioritize toolchains that maximize developer productivity and reduce integration friction. The combination of a JavaScript runtime for the server (Node.js/Express), a relational database for transactional consistency (MySQL), and a component-based frontend (React with Vite) is a common pragmatic choice. This stack allows developers to maintain a unified language ecosystem and iterate rapidly on full-stack features.

The literature documents a common lifecycle: initial development favors simple patterns (including callback-based database access and direct SQL queries) to achieve functionality quickly; as the system evolves, teams migrate to more robust abstractions—Promise-based APIs, ORMs, or service layers—to manage complexity. The MARS tool choices align with this progression: they provide a low barrier to building a coherent prototype while preserving clear migration paths for later refactoring and performance enhancements.

3.7 Summary and Relevance to MARS

The literature reviewed in this chapter corroborates the principal architectural and design choices made in MARS. Centralizing catalog and transactional data, enforcing server-side authorization, adopting hybrid OCR workflows with verification, and commencing recommendation capabilities with heuristic baselines are all grounded in established practice for small and medium-scale retail systems. Moreover, the surveyed work underscores the importance of iterative improvement: augmenting OCR datasets and model tuning, establishing rigorous evaluation for recommenders, and refactoring asynchronous control flow for maintainability.

These insights validate MARS as a research-grade prototype: its design choices balance rapid delivery and operational correctness while exposing specific directions for maturation. The chapter's findings inform the implementation and future work described in subsequent chapters, providing a literature-backed rationale for prioritizing security improvements, model training, and architectural refactors as the system transitions toward production readiness.

Chapter Four

Methodology

4.2 System Overview

MARS is implemented as a full-stack system composed of three main layers: client applications (web and mobile), a server-side API, and a relational database. The client applications are built using a component-based frontend framework and are responsible for handling user interaction, data presentation, and communication with backend services through RESTful APIs.

The backend is implemented using Node.js and acts as the central coordination layer of the system. It exposes resource-oriented endpoints that enforce business rules, manage access control, and coordinate interactions with the database. The relational database stores all persistent system data, including products, users, contracts, payments, projects, and tasks, ensuring consistency and integrity across the system.

Communication between layers follows a standard request–response model. Clients issue HTTP requests to retrieve or modify data, and the server validates incoming requests, executes the required business logic, performs database operations through model modules, and returns structured JSON responses. File uploads are handled as multipart form data and processed using dedicated middleware. Where applicable, uploaded files are stored or further processed, such as extracting text using OCR, before results are returned to the client.

The overall architecture emphasizes separation of concerns. Presentation logic is confined to the client layer, request handling and validation are managed by the API, and data access is isolated within model modules. This separation improves maintainability and allows individual components to be tested and extended independently.

4.3 Backend Architecture

The server-side application is implemented using Node.js with the Express framework. Backend functionality is organized into modular route files, where each module corresponds to a specific resource domain such as items, employees, projects, tasks, or payments. These route modules define the available HTTP endpoints and delegate request handling to controller or handler functions.

Cross-cutting concerns, including authentication, authorization, file upload handling, and basic input validation, are implemented using middleware. This approach reduces duplication and ensures that shared logic is applied consistently across relevant routes.

Data persistence is handled through a set of model modules that encapsulate SQL queries and interactions with a MySQL database. The project uses a callback-based database access style provided by the mysql/mysql2 driver. This design choice was made to keep implementation straightforward within the project timeline and based on team familiarity. However, it is identified as a candidate for future refactoring, as migrating to a Promise-based or async/await approach would simplify asynchronous flow control and improve error handling in larger systems.

A clear separation is maintained between controllers and models. Controllers translate HTTP requests into application-level actions and prepare input for database operations, while model modules focus exclusively on SQL construction, execution, and result mapping. Database configuration is centralized, allowing connection settings and pooling behavior to be managed consistently.

Authentication and authorization mechanisms are enforced at both the route and controller levels. Users are assigned predefined roles—such as customer, worker, admin, or co-admin—which determine the operations they are allowed to perform. Middleware inspects user roles and restricts access to protected resources accordingly.

4.4 File Upload and OCR Processing Methodology

File uploads in MARS are handled using a middleware library that supports different storage strategies. Two approaches are applied depending on the type of uploaded content and its intended usage.

For product images, memory-based storage is used. Uploaded images are temporarily stored in memory buffers and may be converted to base64 or other serializable formats for immediate use in API responses. This approach simplifies handling small image files and avoids managing a large number of files on disk, but it increases memory usage and is therefore suitable only for relatively small uploads.

For project and task attachments, disk-based storage is employed. Uploaded files are saved to a dedicated directory on the server and served statically when required. Disk storage is more appropriate for larger or persistent files, but it introduces additional considerations such as file cleanup, access control, and backup management.

After upload, certain files are passed to an OCR service for processing. The OCR component uses trained datasets to extract textual information from receipts or scanned documents. Depending on the storage strategy, the OCR service processes either an in-memory buffer or a file path on disk. The extracted text is then made available to other system components, such as attaching recognized content to payment records or supporting downstream recommendation logic.

4.5 Design Trade-offs

Several design trade-offs influenced the project methodology:

- Memory versus disk storage: Memory-based uploads simplify short-term image handling but increase memory consumption and are unsuitable for large files. Disk-based storage supports persistence and larger files but requires additional operational management.

- Callback-based database access: Using callbacks reduced initial complexity and development time but results in more nested control flow. Refactoring to Promises or async/await would improve readability and error handling in future versions.

- Local deployment focus: Emphasizing local deployment simplified development and testing but limits scalability and availability. Transitioning to a cloud-based architecture would require redesigning configuration, storage, and scaling strategies.

- Prototype AI components: The OCR and recommendation features are implemented as prototypes. Their effectiveness depends on the quality and quantity of available training data, and production-grade accuracy would require further dataset expansion and model tuning.

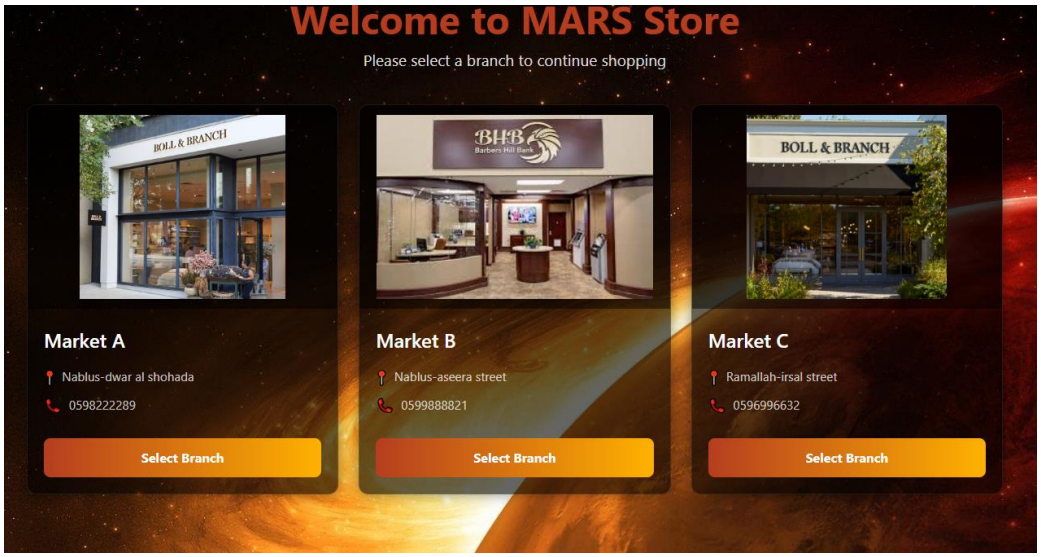
Overall, the methodology prioritizes clear separation of concerns, incremental development, and explicit documentation of limitations and future improvements. These choices align with the academic nature of the project and its resource constraints while leaving a clear path for future enhancement and scaling.

Chapter Five

Implementation and Testing

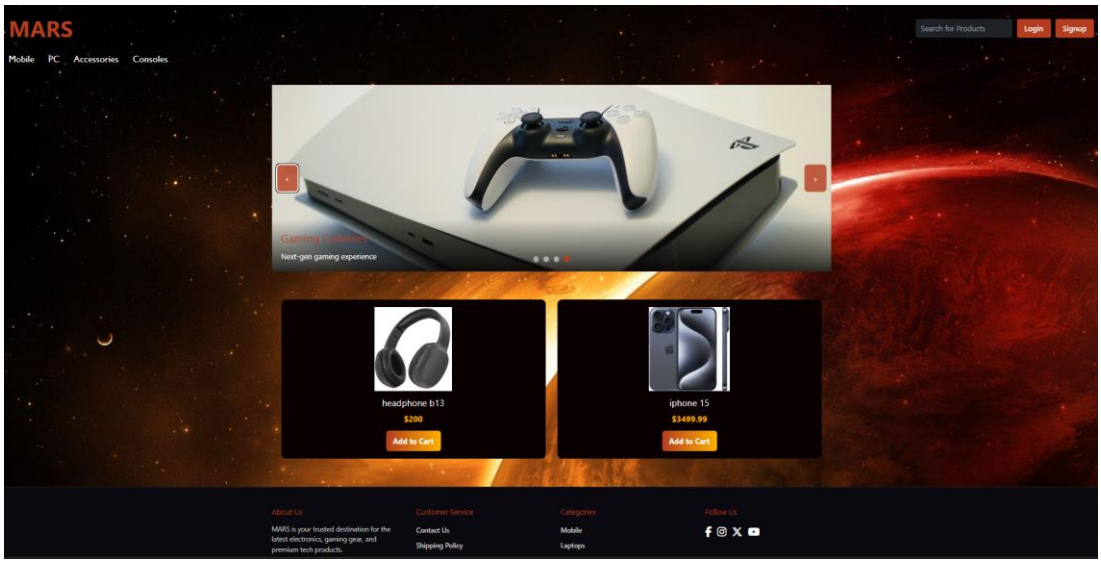
5.1 Common Features

5.1.1 Branch selection



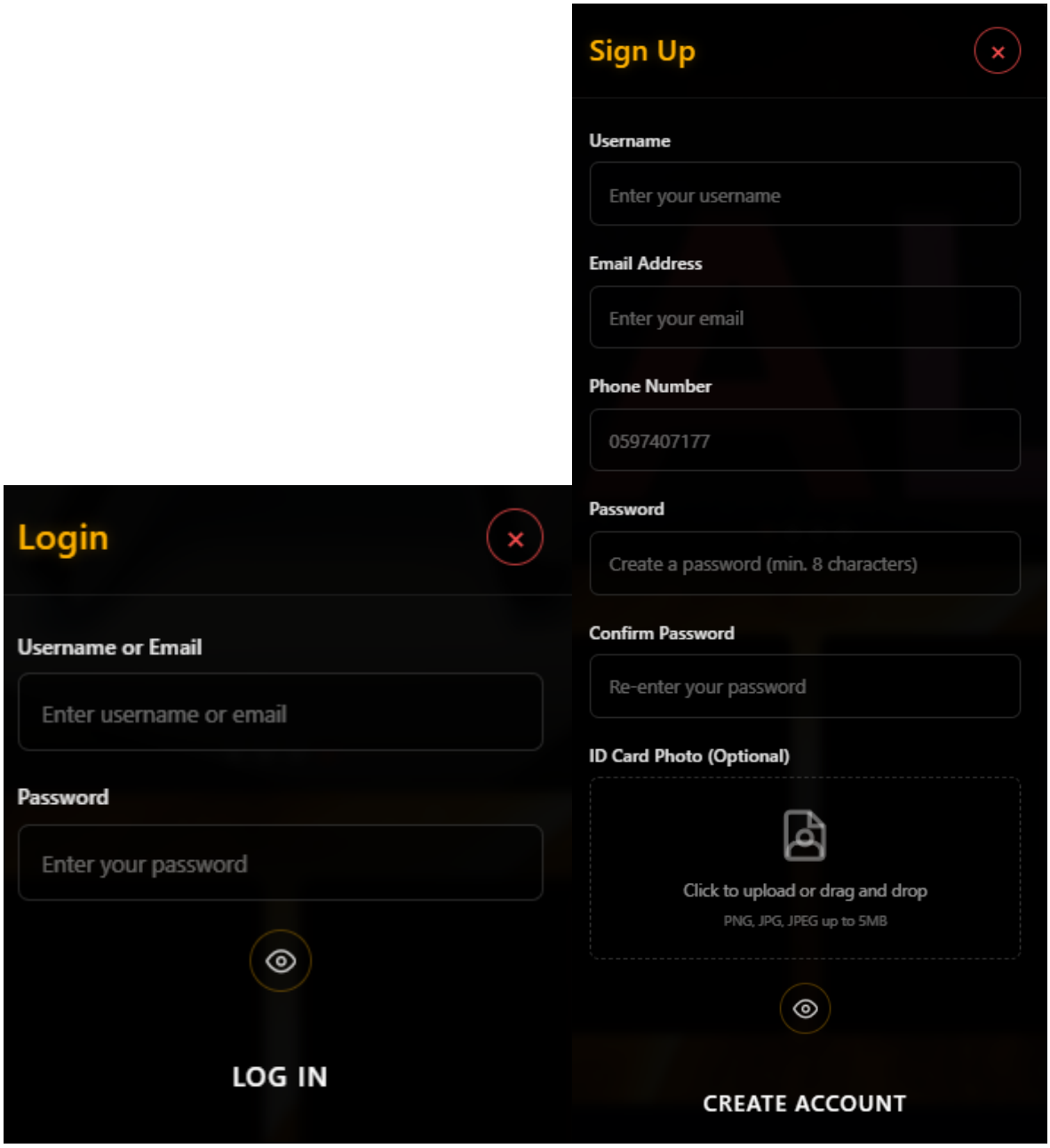
This is the Startup of the pages for all MARS users.

5.1.2 Home Page



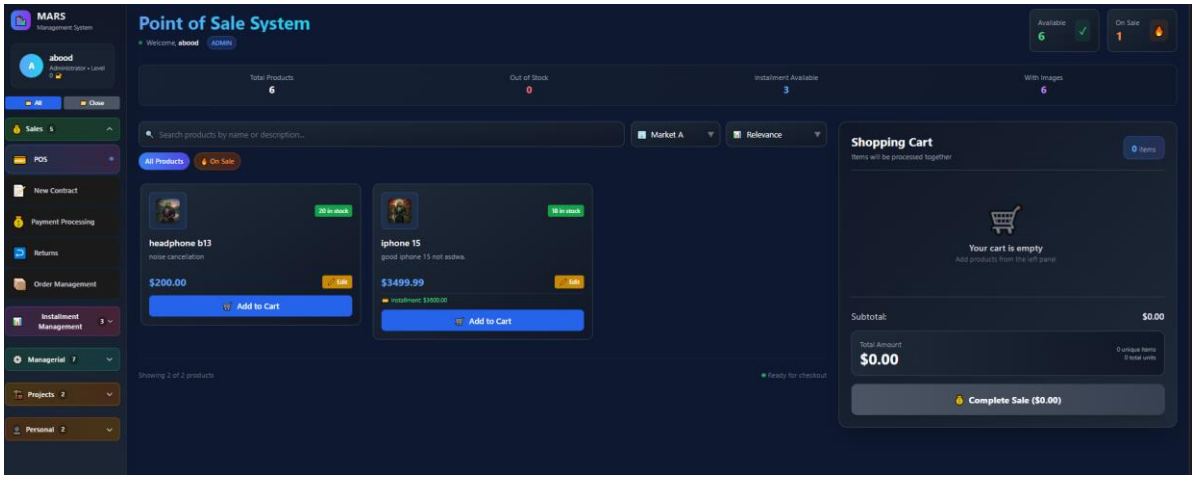
For Showing what each branch has items letting the customer to browse with comfort.

5.1.3 Login / Sign Up



The Sign up here is only for customers, where employees sign up by an admin of the store.

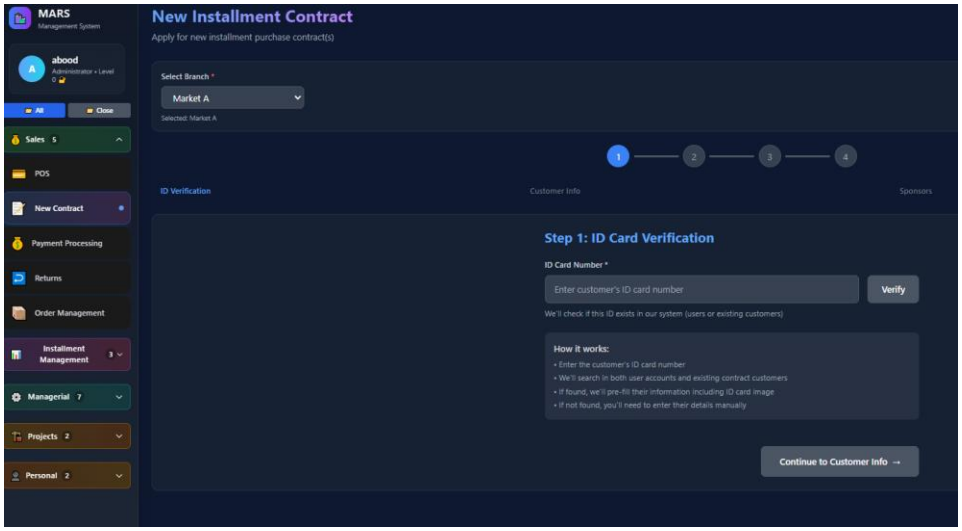
5.1.4 Point of Sale



This page was designed for employees that are selling inside the real Store, recording each transaction of anything sold.

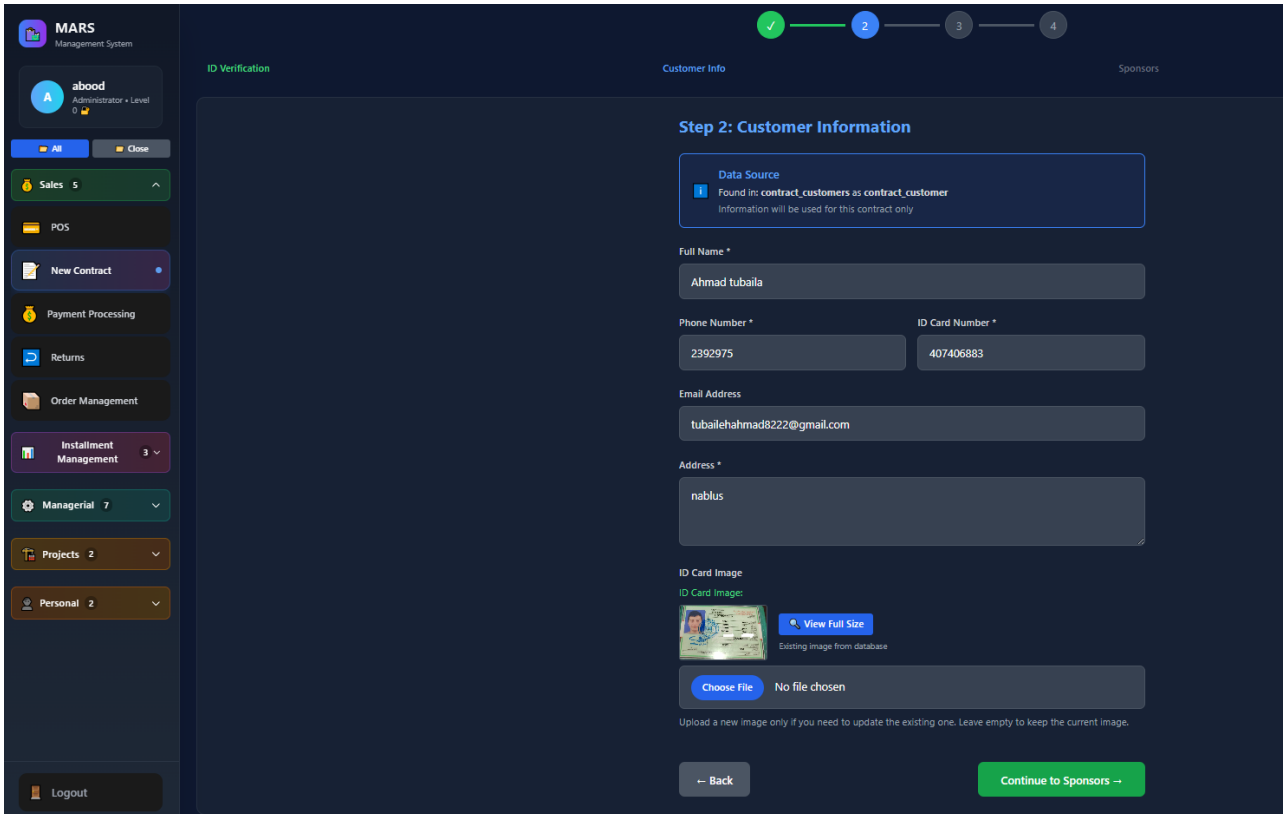
5.1.5 Installment Contracts

5.1.5.1 ID Verification



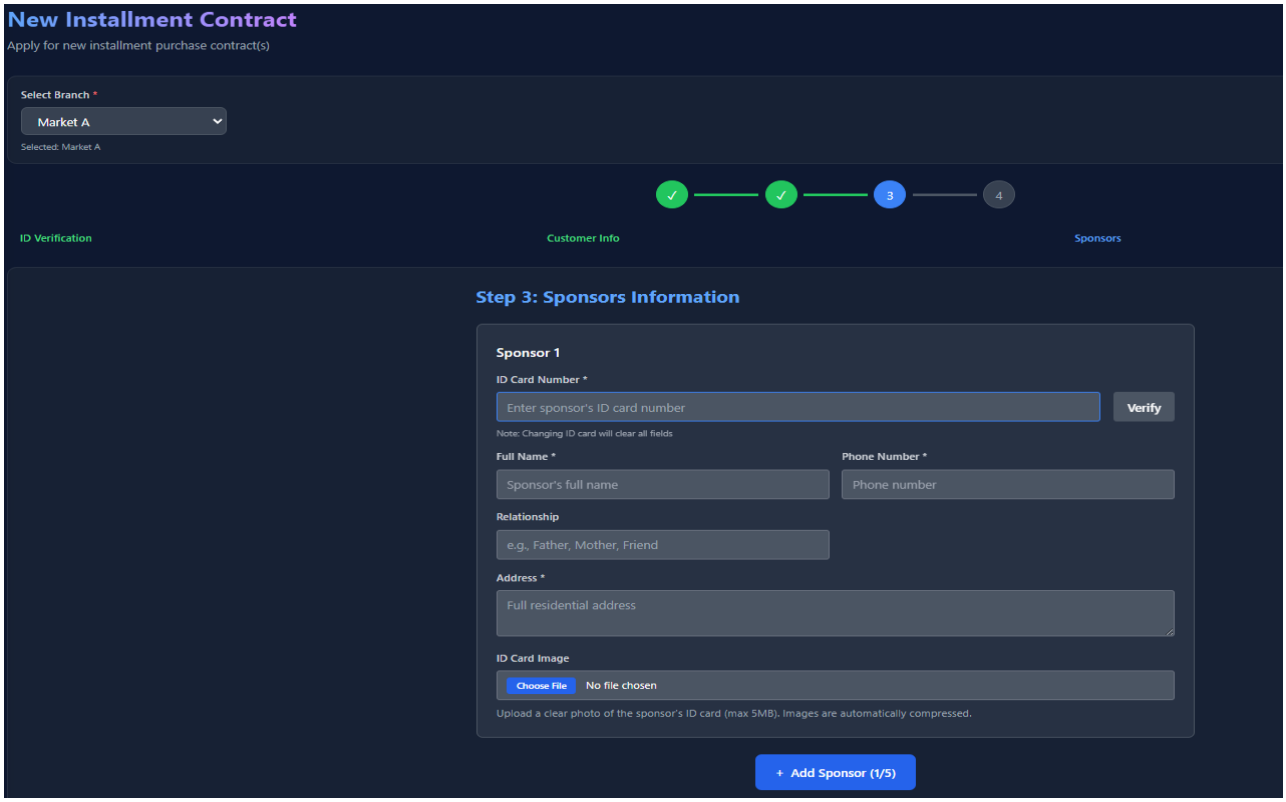
In this page we verify whether the customer was in the database already or not, if not it will be added to database.

5.1.5.2 Customer Information Gathering and Checking



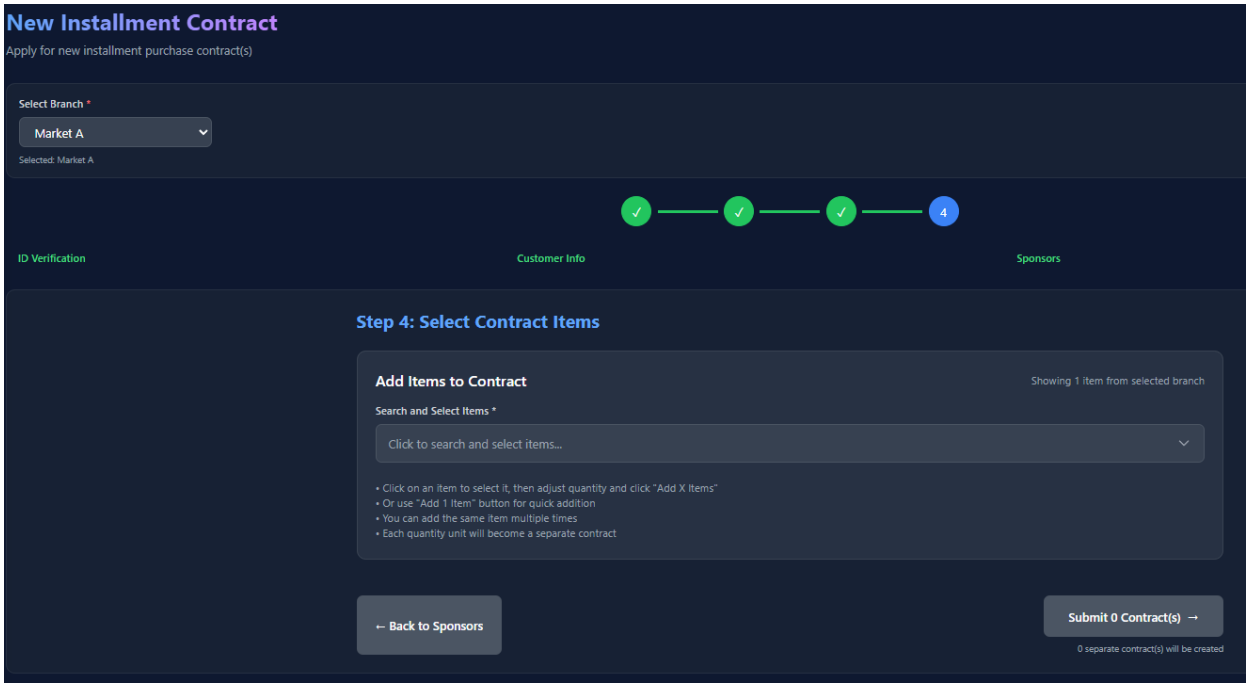
In this page customers information are gathered, if it was already in data base it will be auto filled, if not it will be manually filled.

5.1.5.3 Sponsors Selection



Sponsors info will be gathered where each customer installment must have at least one.

5.1.5.3 Item selection for installment

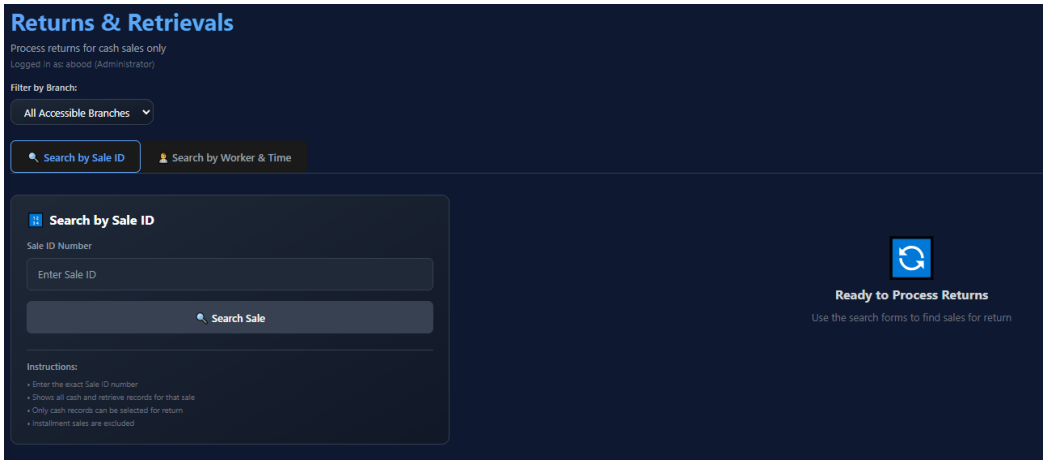


This is the place where the employee can choose what item to make the installment happen.

5.1.6 Payment processing

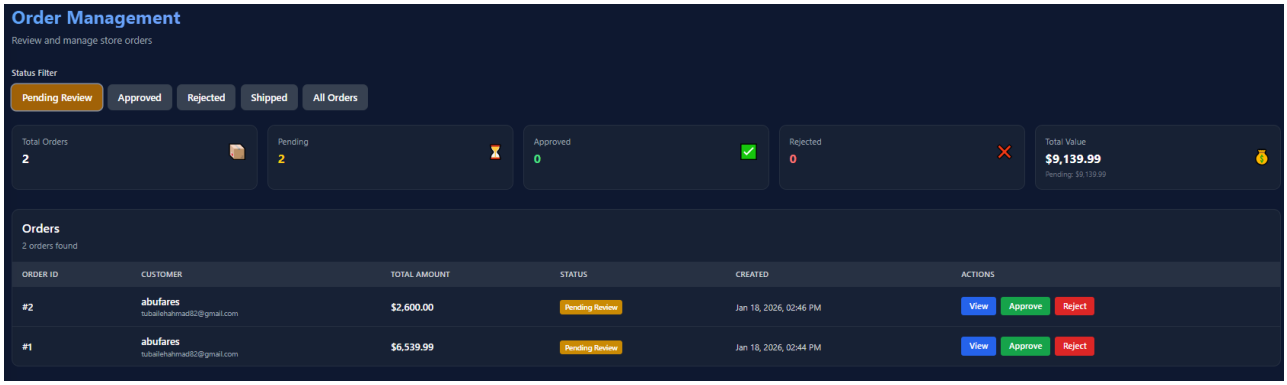
In this page the employee can search for installments were made by searching for the requested user.

5.1.7 Returned items



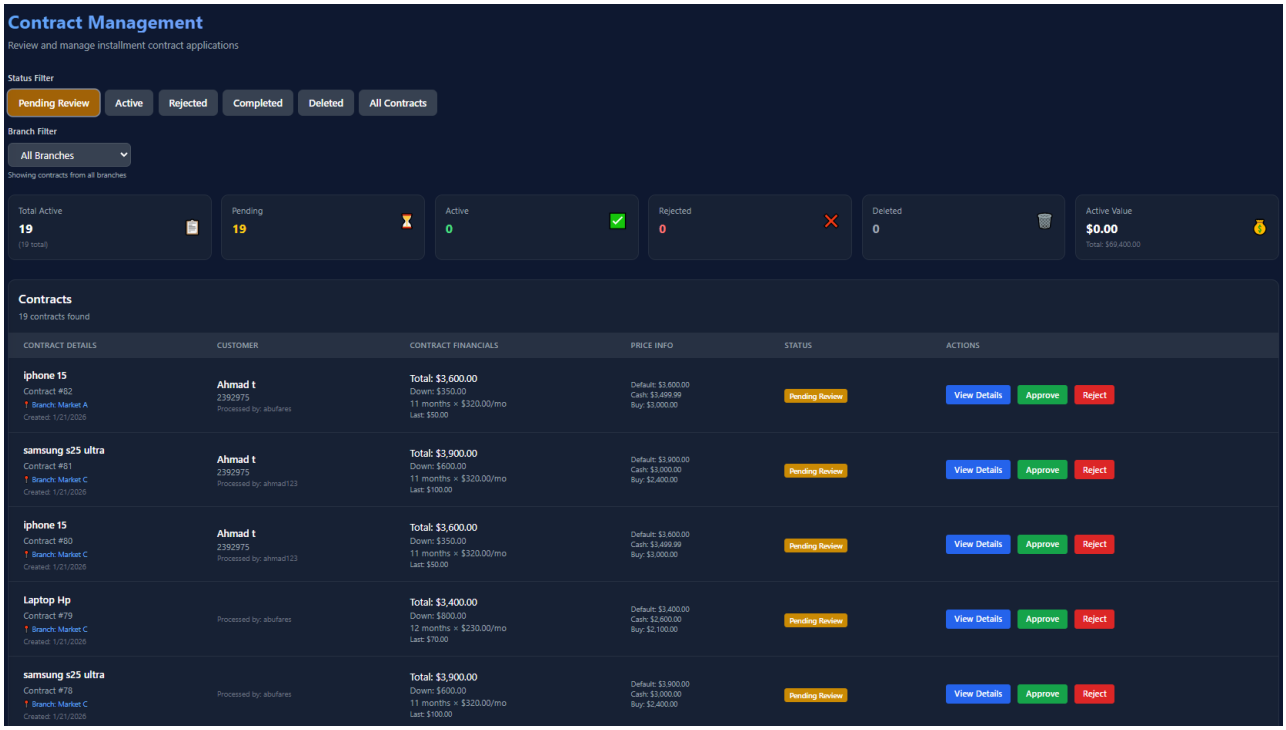
Incase a customer returned an item, then the employee or admin can check for returned items.

5.1.8 Store Customers Order management



In this page the employee/admin can view the orders that comes from the website and they are able to reject or approve on them.

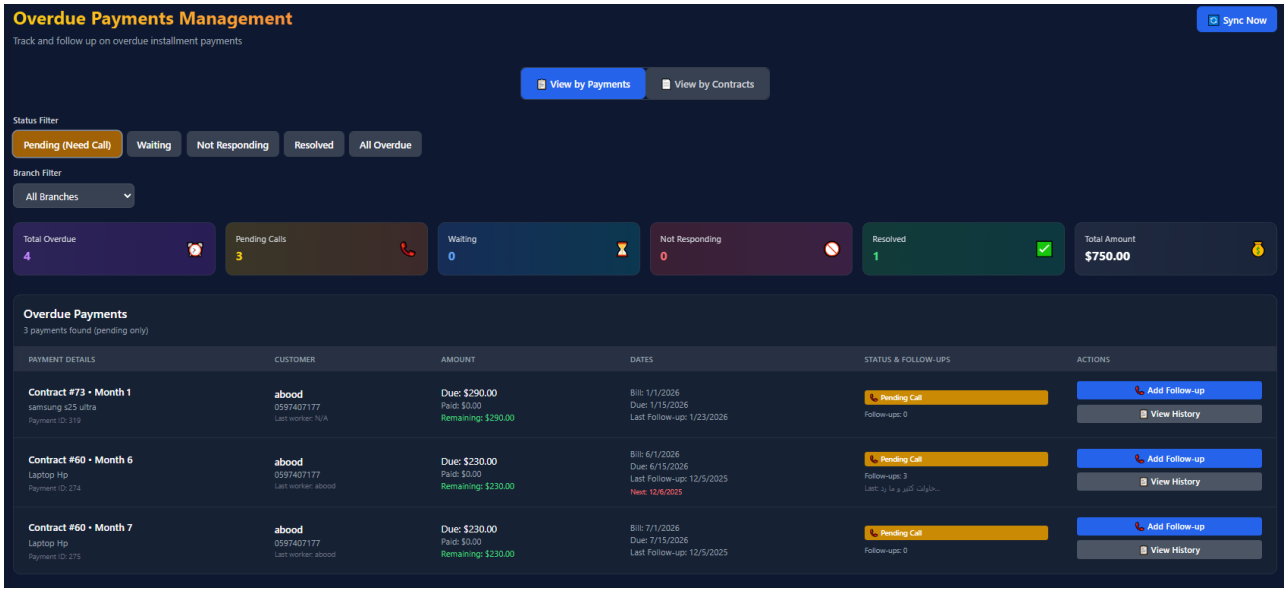
5.1.9 Installment Contracts management



A page to check on all contracts made from customers either from the store website or from the real-life store, with the ability to reject or approve on them.

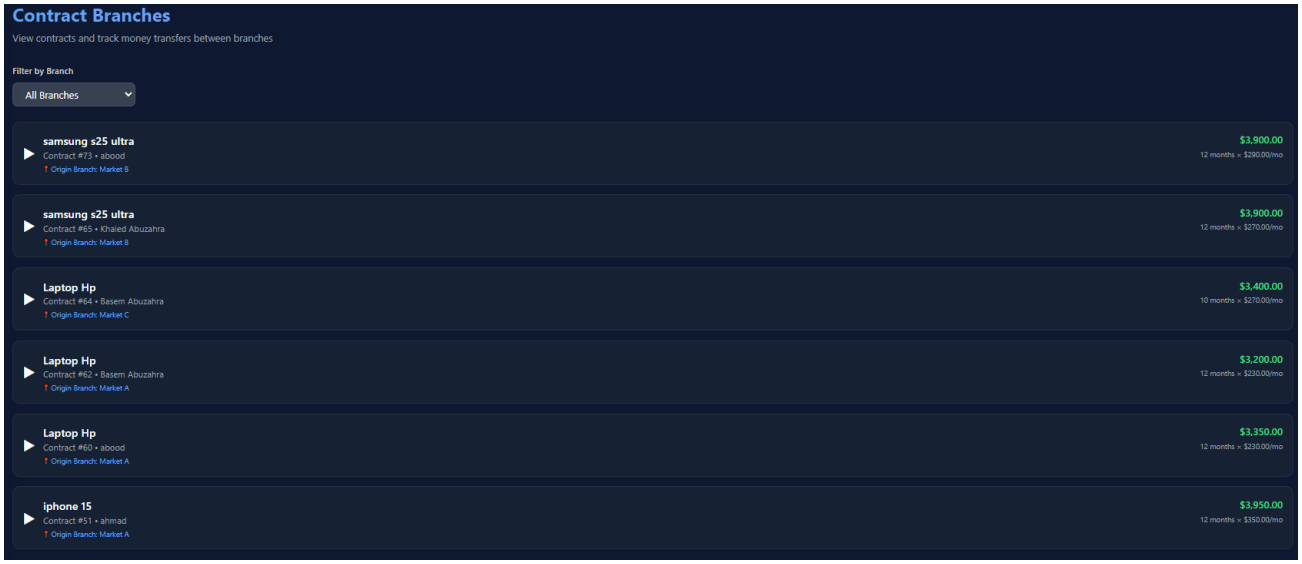
If rejected then a reason must be present and wrote to the customer.

5.1.10 Overdue Payments Management



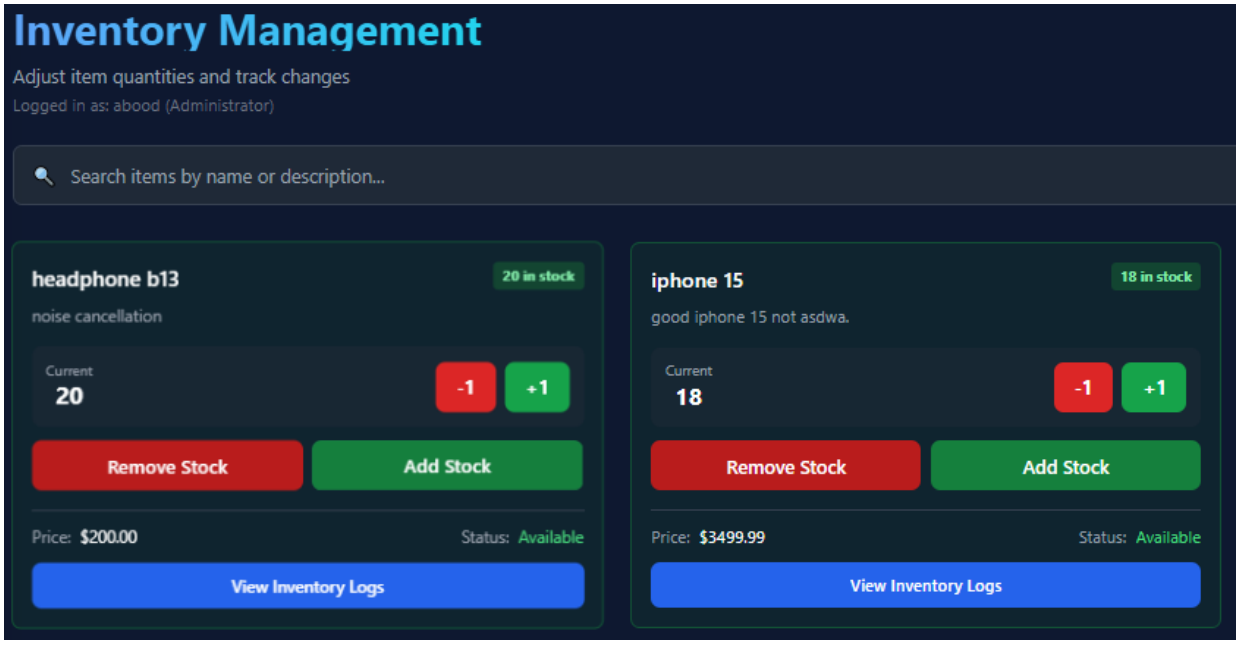
This page present any contract for installment that are being late.

5.1.11 Contracts View of all Branches



This page shows up all the installment contracts for all branches.

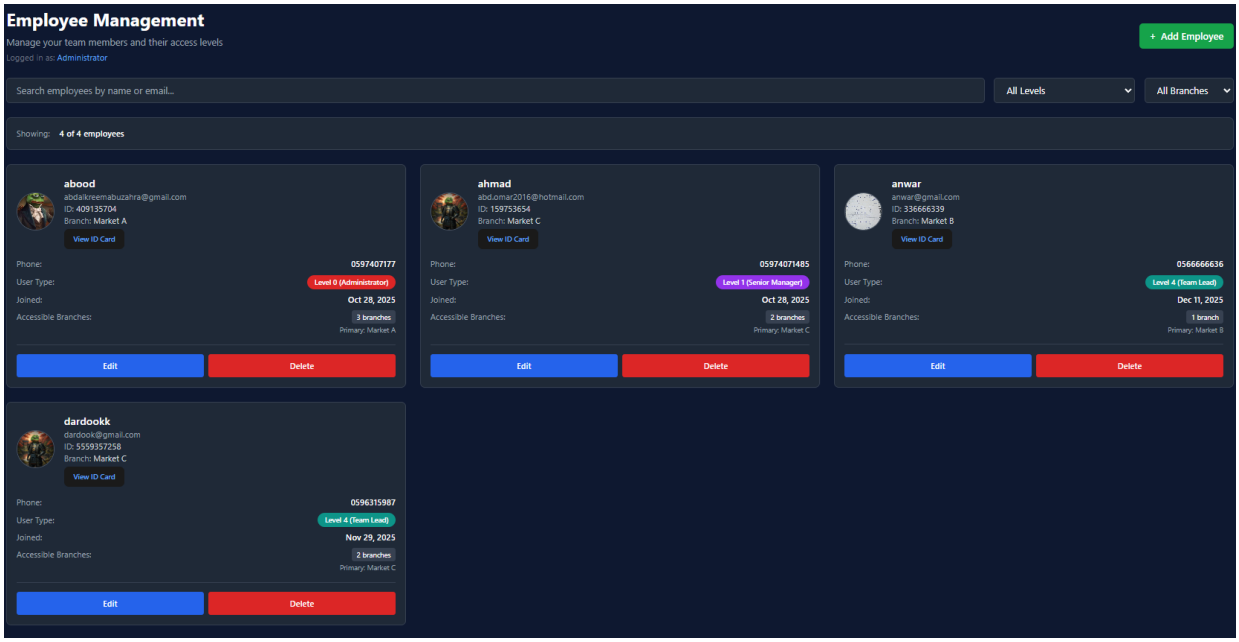
5.1.12 Stock Management



This page the employee can edit the stock and select the number he wants

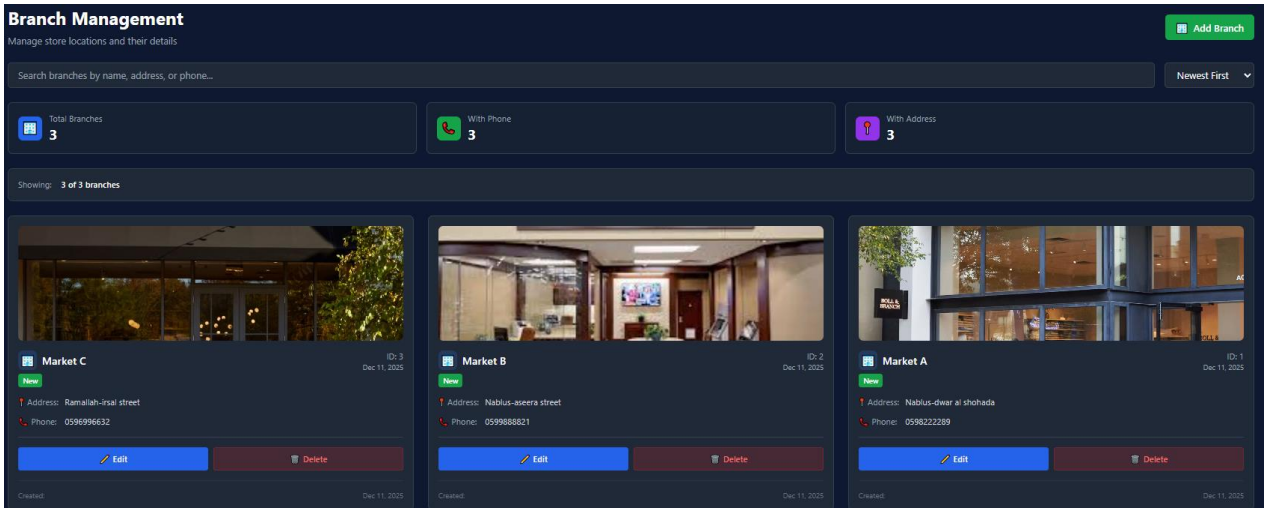
.

5.1.13 Employees Management (For Admin only)



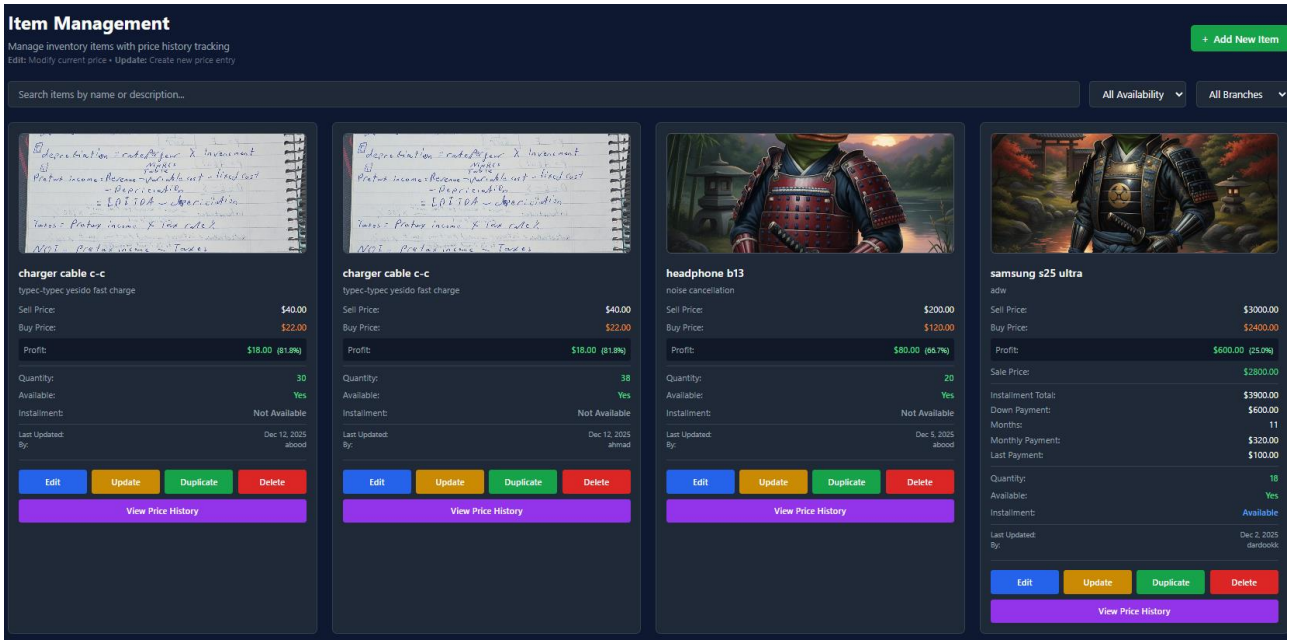
In this page the admin can edit the role of each employee.

5.1.14 Branch Management



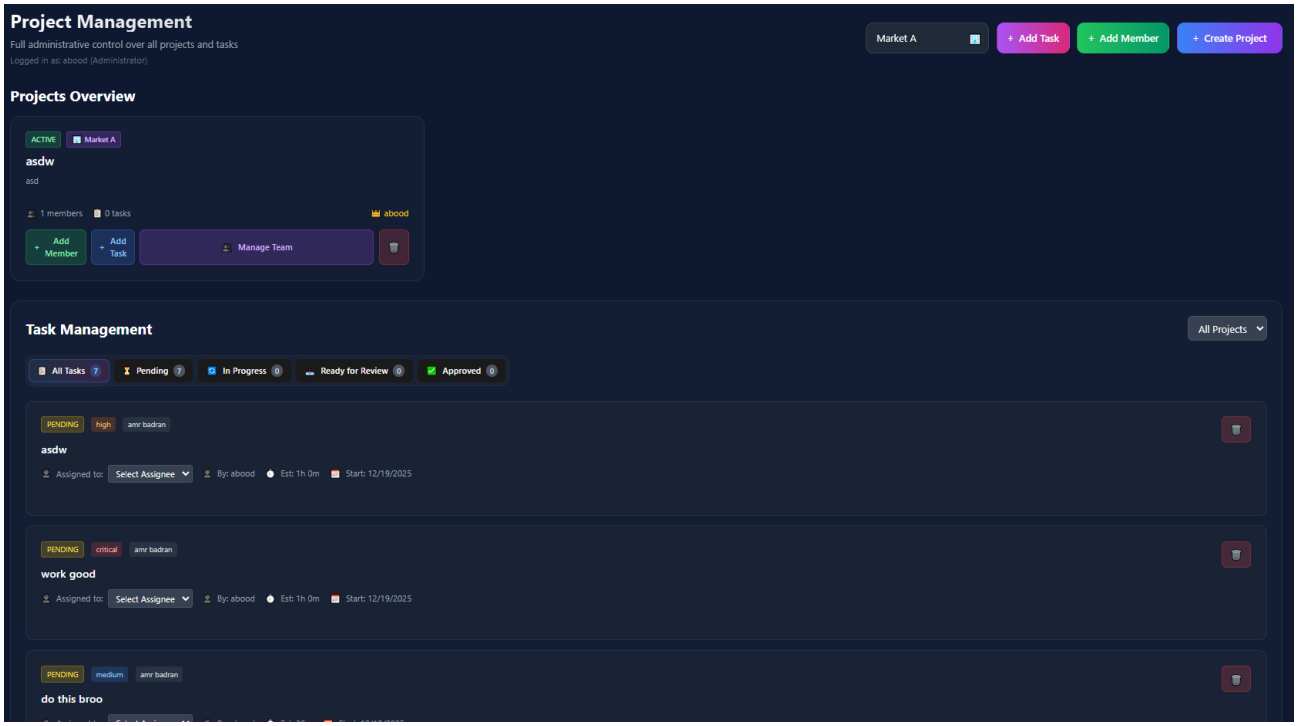
This page for adding/removing any branch.

5.1.15 Items Management



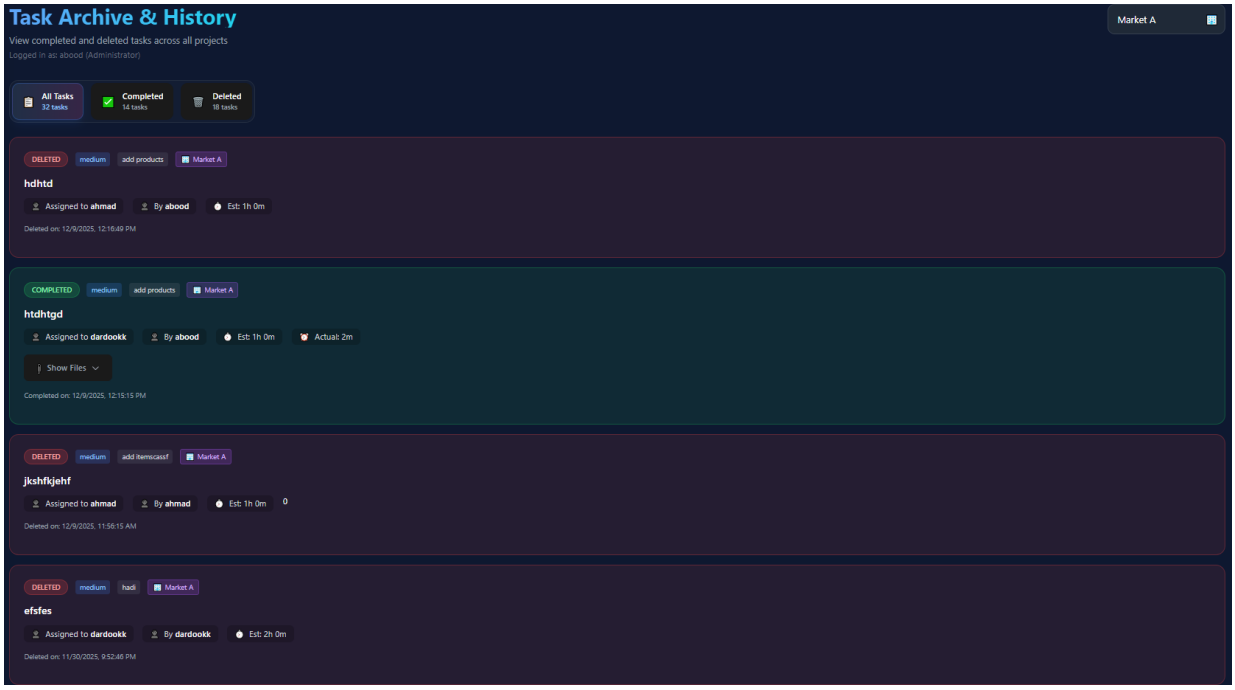
This page is specialized for adding items with the access to manipulate them.

5.1.16 Tasks / Project Management



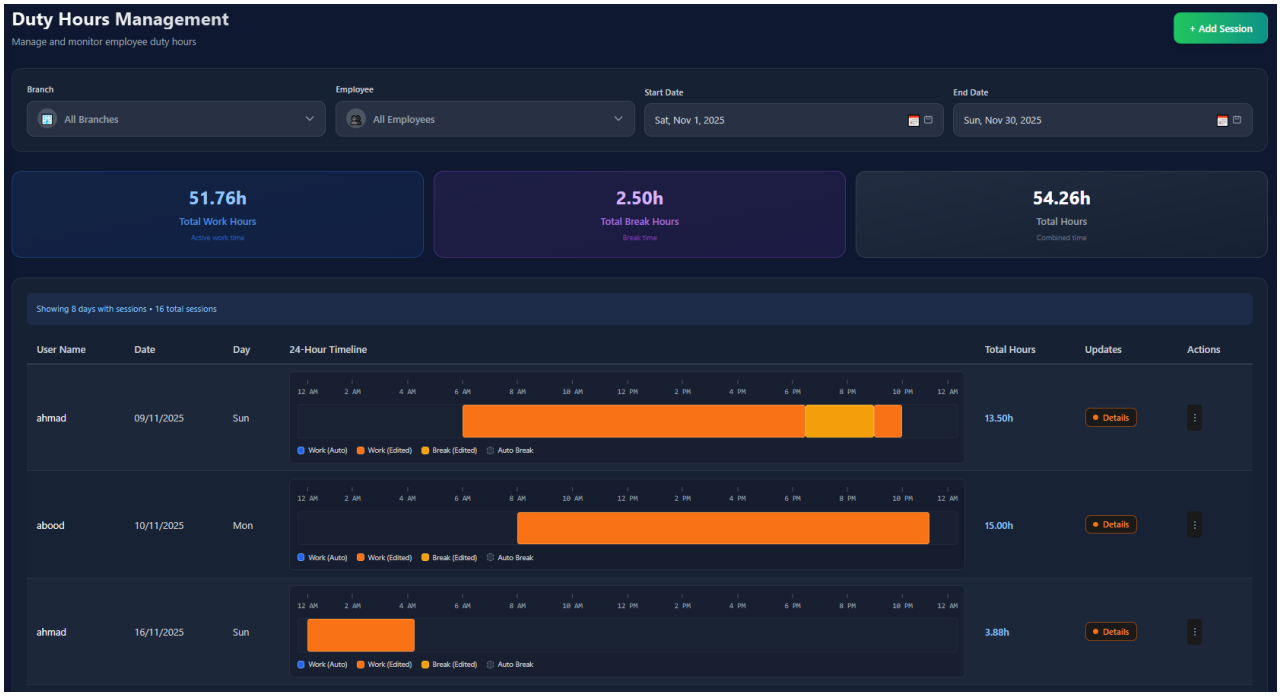
In this page the admin can assign tasks for each employee he has.

5.1.17 Tasks History



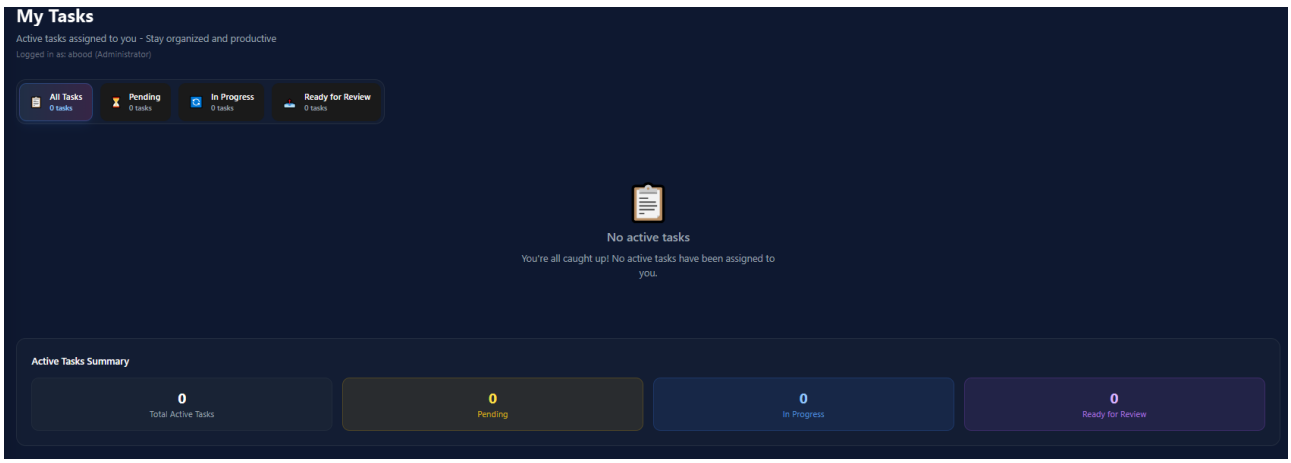
This page shows the history of assigned tasks and whether it was done or not.

5.1.18 Employees Duty Hours



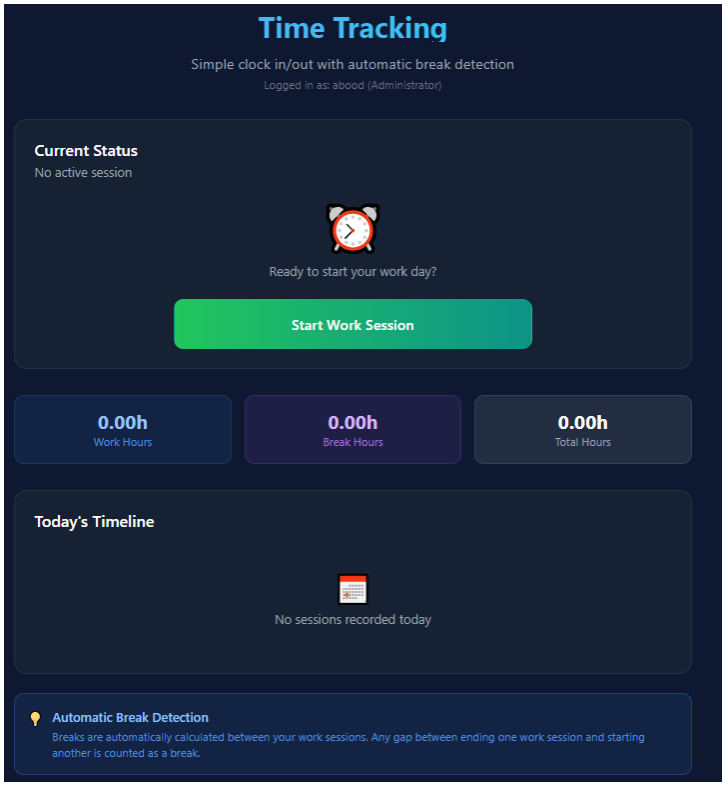
This is the side where the manager can see his employees working times.

5.1.19 Tasks Board



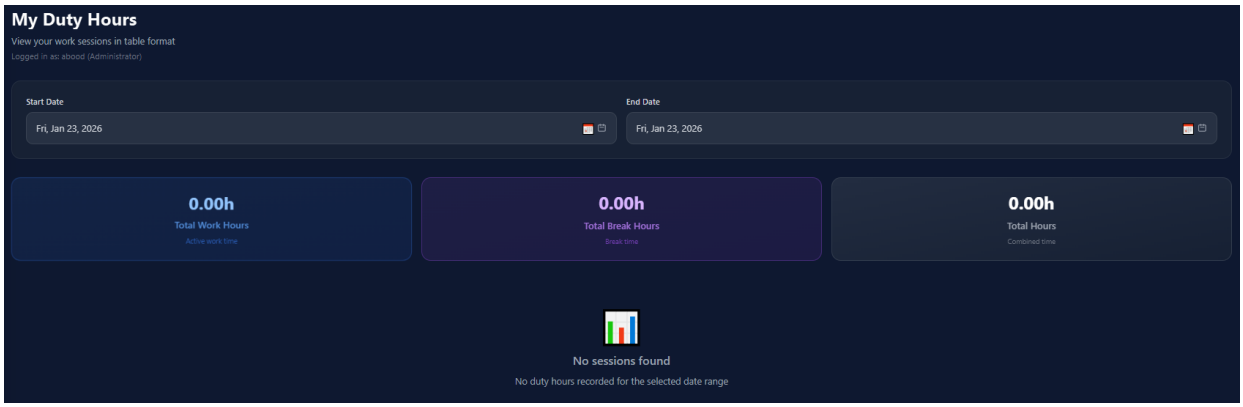
This page is special for employees so they can see their assigned tasks by the higher roles.

5.1.20 Day Time Hours Tracking



An employee page for calculating his working time during the day from the start till the end of his work.

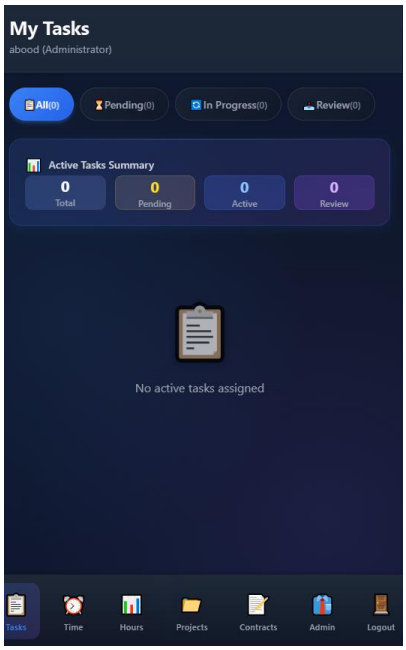
5.1.21 Total Working Hours



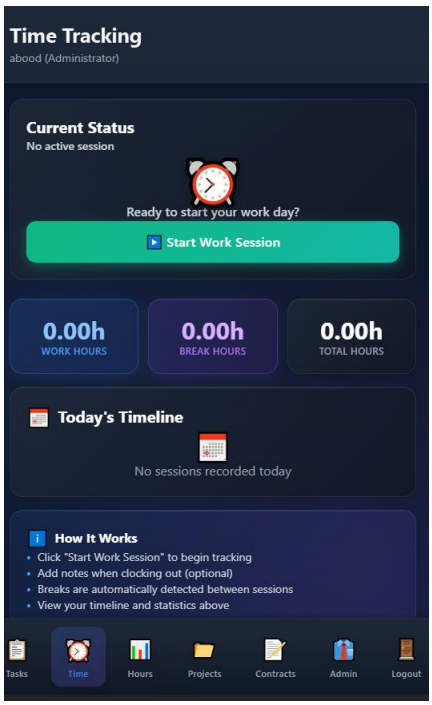
To show each worker his working times in the store.

5.1.22 Mobile Employee/Admin Side

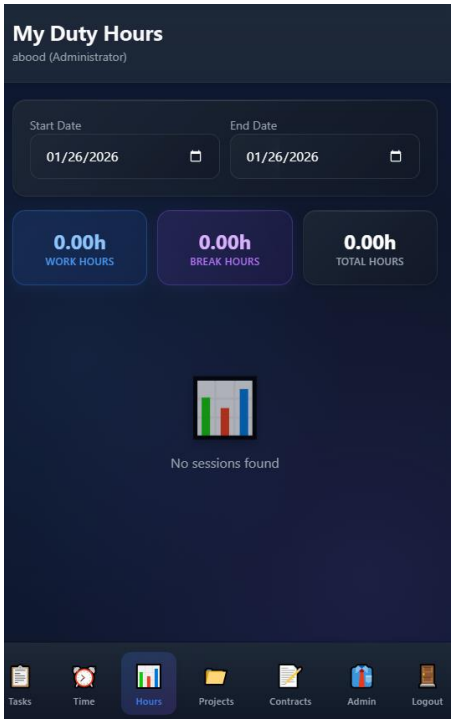
5.1.22.1 Tasks Page



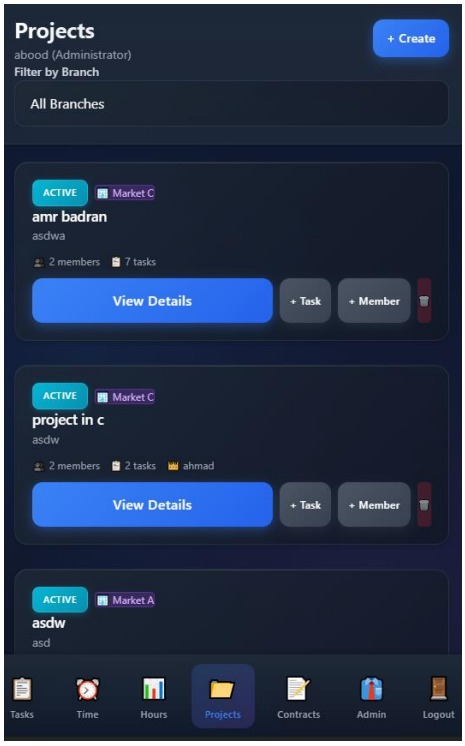
5.1.22.2 Time Tracking



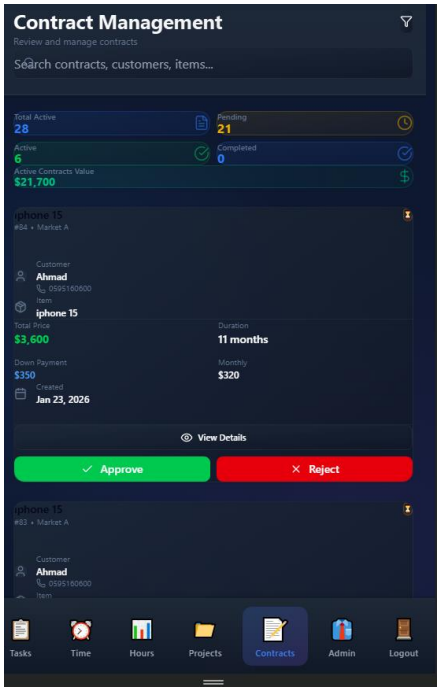
5.1.22.3 Hours Tracking Page



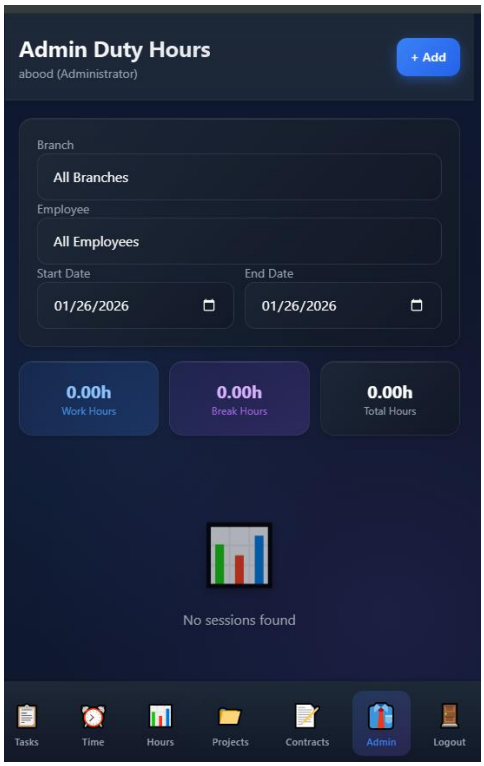
5.1.22.4 Projects Page



5.1.22.5 Contract Management Page

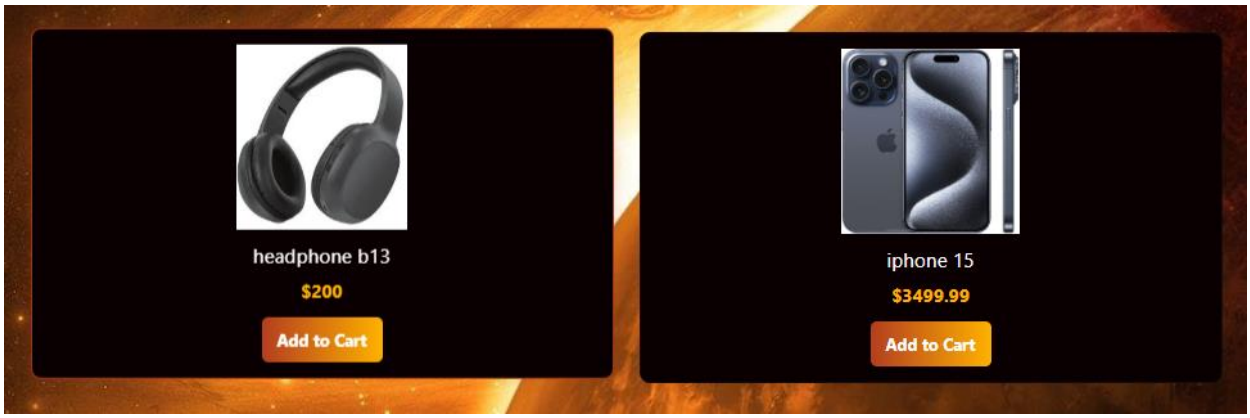


5.1.22.6 Duty Hours Administrative View

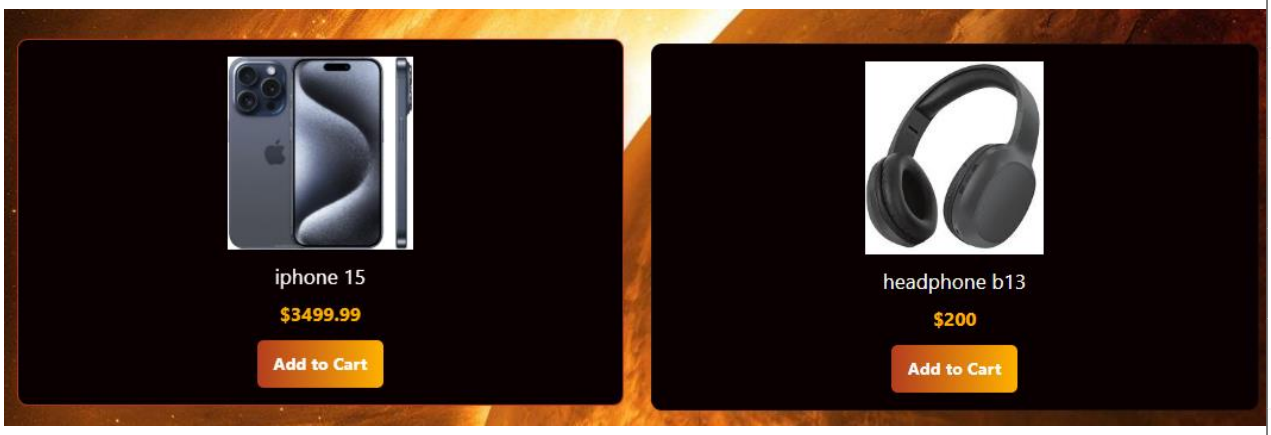


5.1.23 Recommendation System

Before clicking on the iPhone:



After clicking on the iPhone:



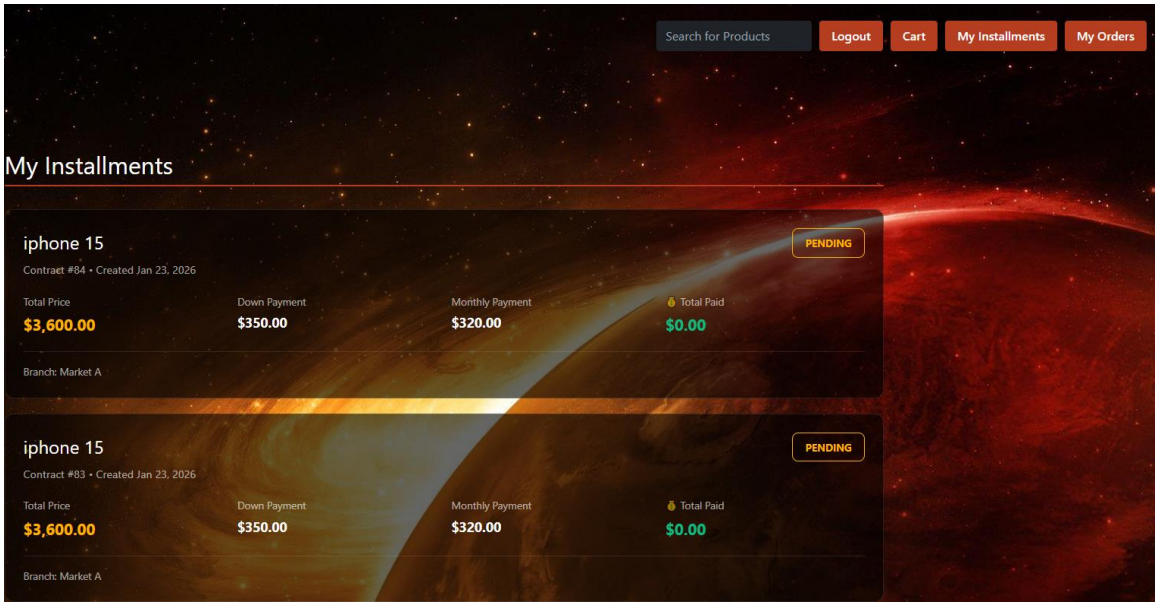
This is a machine learning system designed for cold-startup and for logged in customers, where when clicking on an item it will increase the item's score.

Increasing the score of an item will bring it on top and on left side of the store showing how the popular the item is. However, this is common for both, cold starters and logged in users.

The main difference comes in the purchasing system that is specialized for logged users where purchasing a product increases its own score three times than only view (1 purchase = 3 views).

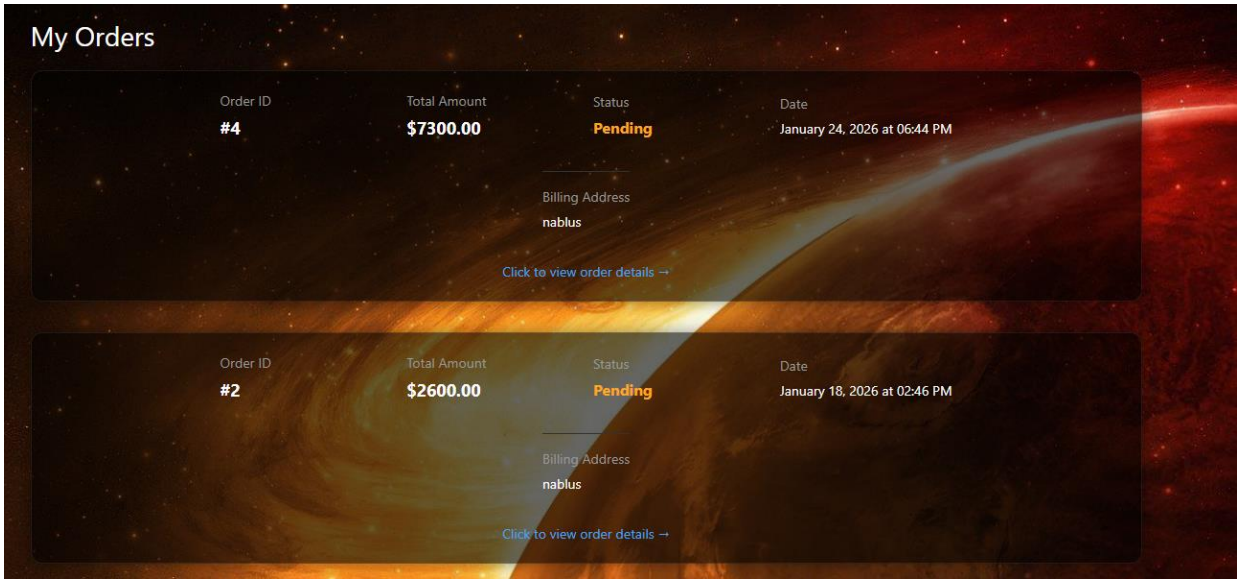
In that way not only the product being held up, but also the category the item comes from will be also presented for some more items from that category.

5.1.24 Customer Installment Contracts History



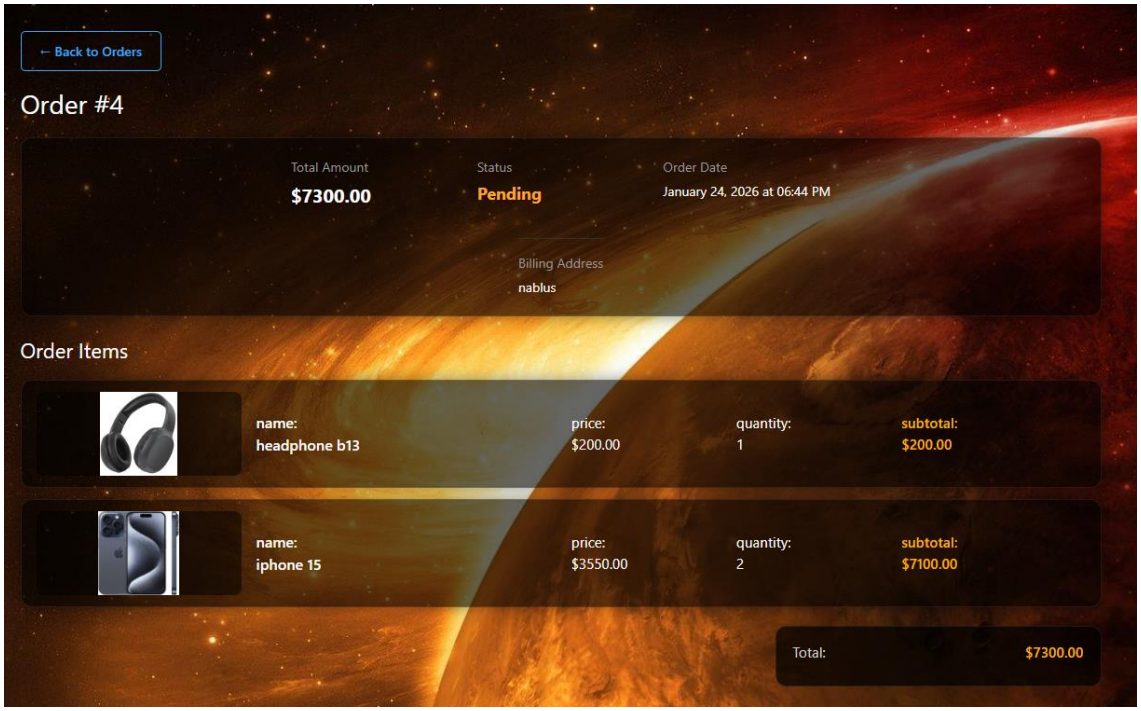
This page we show the user his own made installment contracts and their status.

5.1.25 Customer group Order History



In this page customer can see each order group he has uploaded, showing the status of the order, knowing that each order of them may contain more than one item, so each of them is clickable.

5.1.26 Customer Single Order



This is a sample of one of group orders, when clicking on any group order, a page with more details of a single order showing all the items included for that order.

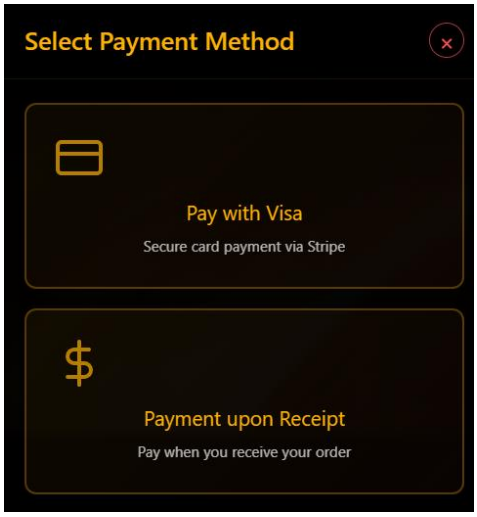
5.1.27 Product Page

This is a product own page where price, photos of the product and choosing how to own the item is shown.

5.1.28 Cart Page

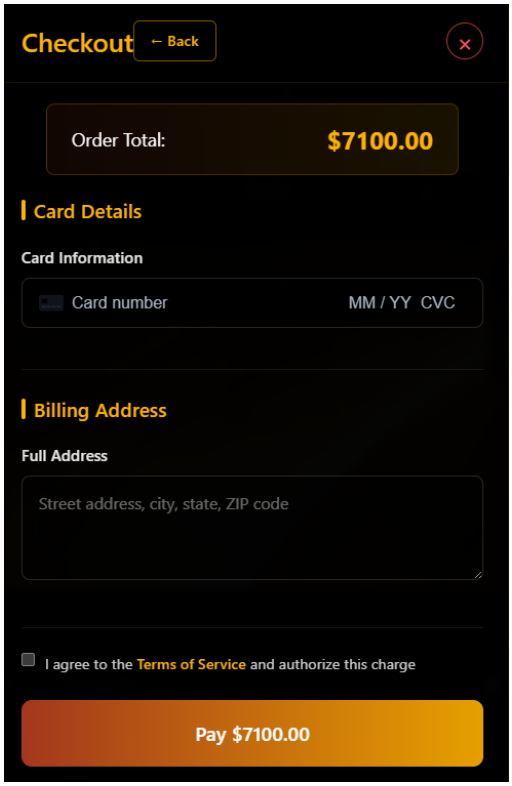
When a customer adds an item to cart it will be brought here showing the details of the items.

5.1.29 Payment Selection



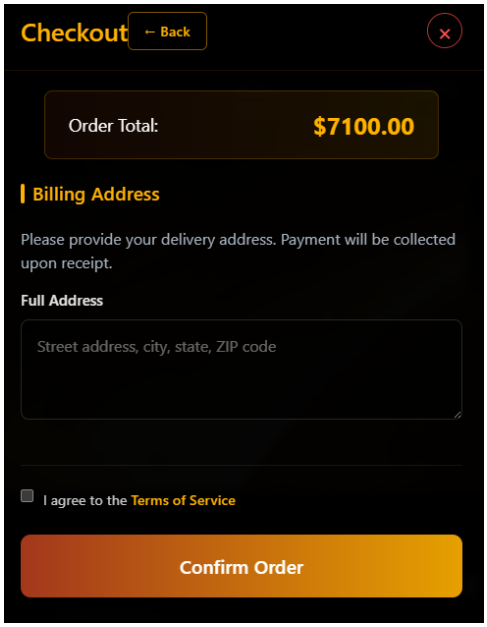
Via visa or upon Receipt payment

5.1.30 Cart Checkout



After a customer wants to proceed to purchase the cart items, a sub-page will appear for credit card info.

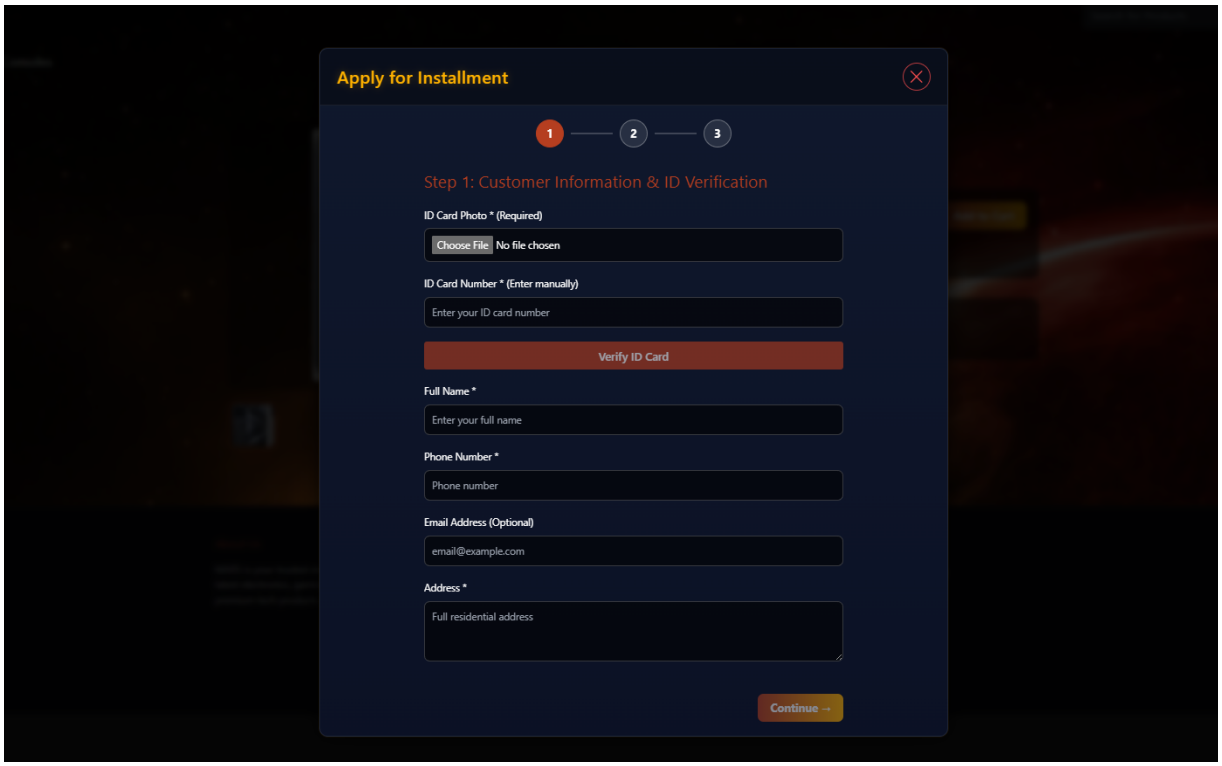
5.1.31 Payment Upon Request



This page is special for payment after delivery is accomplished.

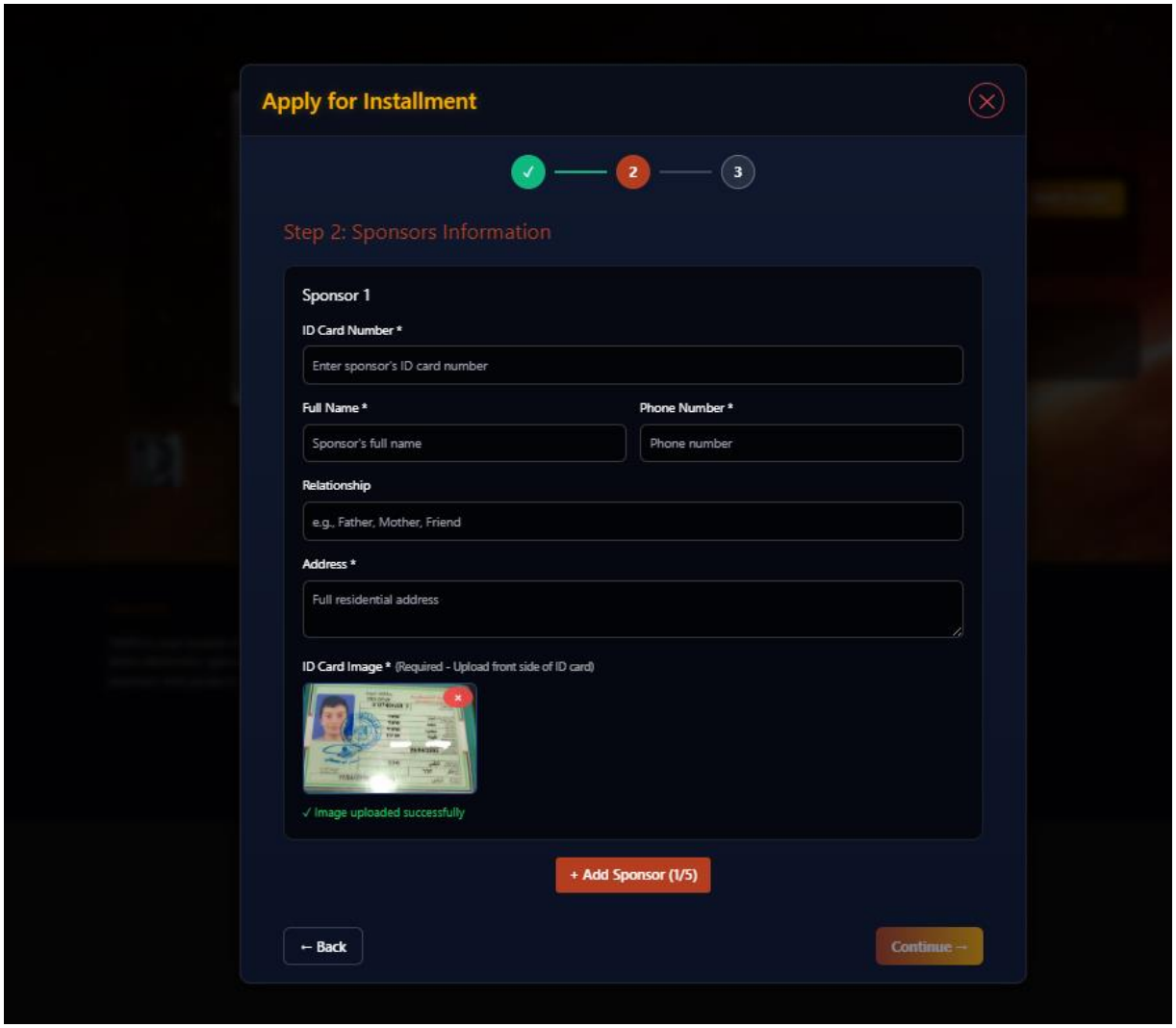
5.1.27 Product Installment

5.1.27.1 Customer Details for Installment



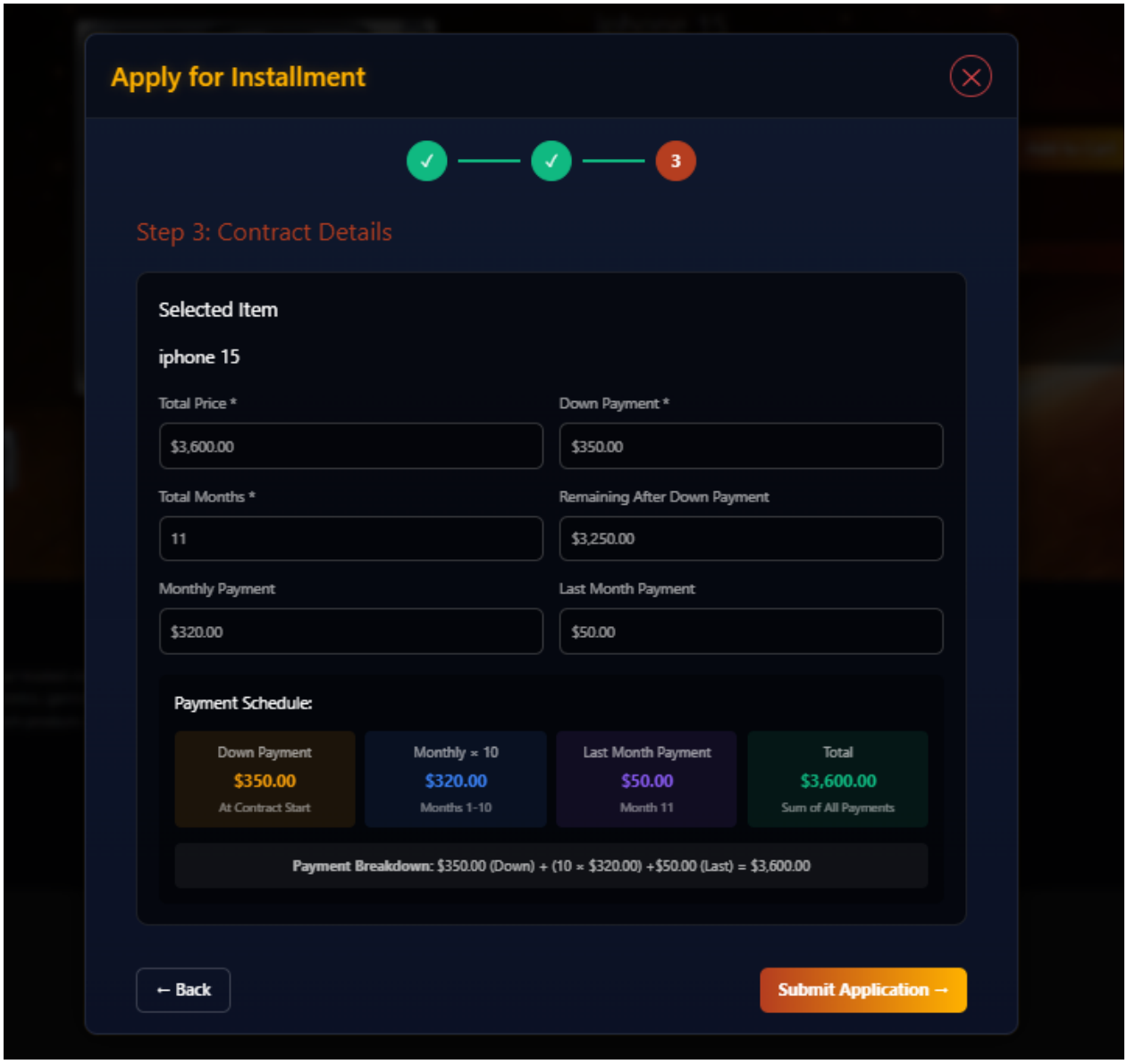
In this page if a customer wants to proceed with installment for an item, it will be the first page to appear asking the customer for personal information such as the name, id card photo, etc.

5.1.27.2 Sponsors Details



The second page of installment, where at least one sponsor is required to proceed.

5.1.28.3 Contract Details

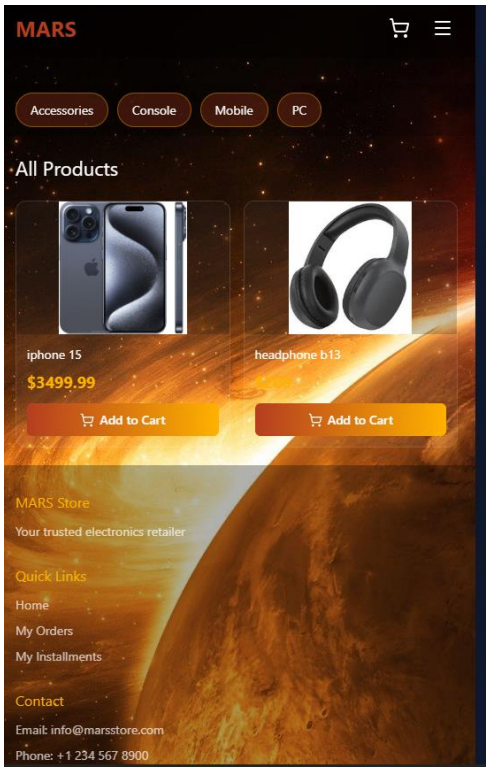


Third and final page of installment, showing the item price in installment and other details like the downpayment and monthly payment.

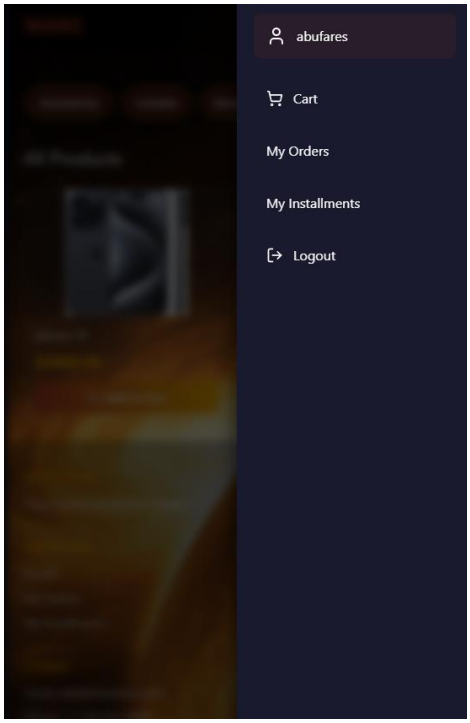
5.1.29 mobile section

5.1.29.1 Branch selection

5.1.29.2 Home Page

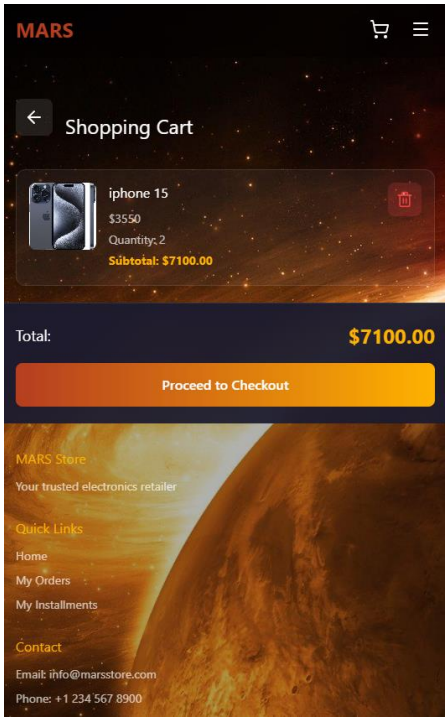


5.1.29.3 Side Bar

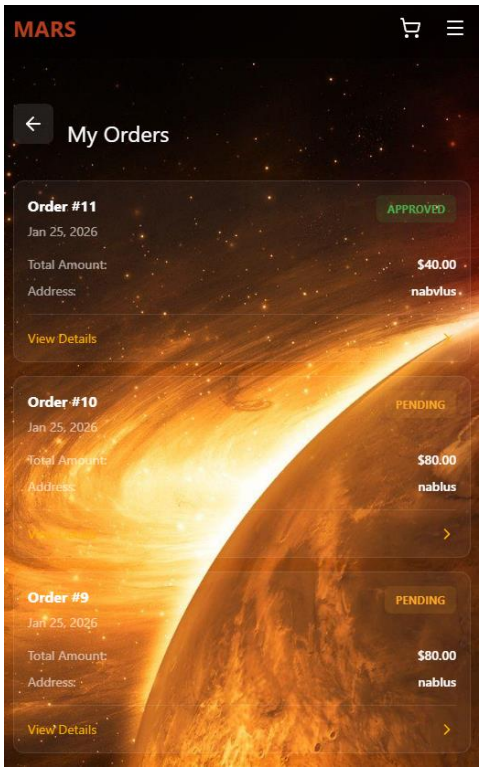


5.1.29.4 Product page

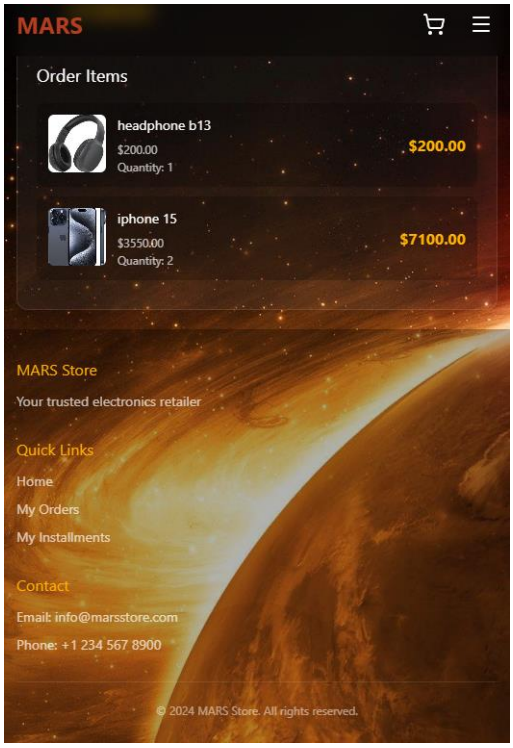
5.1.29.5 Cart page



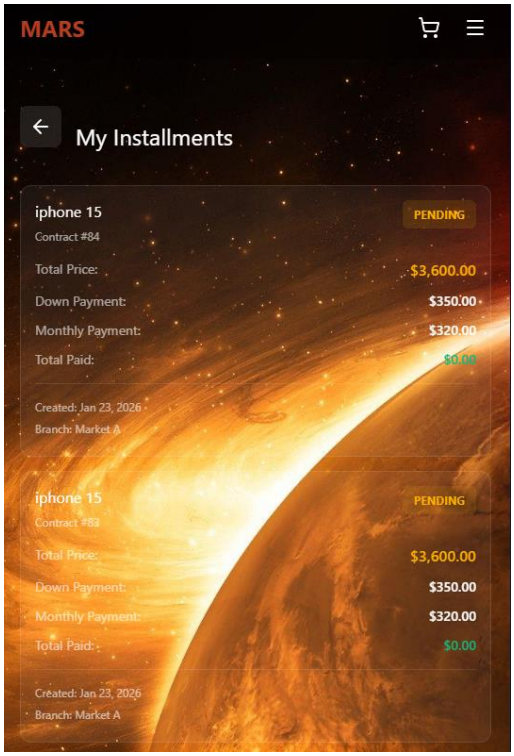
5.1.29.6 Ordered Groups Page



5.1.29.7 Items of an Order



5.1.29.8 Installment Contracts



5.2 Product Management

Purpose:

The Product Management component defines the lifecycle and representation of merchandise within the MARS system. It centralizes product information—including descriptive attributes, pricing, inventory status, and associated media—so that both customer-facing catalog views and internal operations such as point-of-sale processing and inventory reconciliation rely on a single, consistent data source.

CRUD Operations:

Product management supports standard Create, Read, Update, and Delete (CRUD) operations. New products can be added to the system, existing products can be retrieved and listed for display, product details and availability can be updated, and obsolete items can be removed or deactivated. These operations support routine retail activities such as catalog maintenance, price updates, and inventory corrections.

Backend Routes and Models:

On the backend, product-related functionality is exposed through resource-oriented API routes. These routes handle requests for creating, retrieving, updating, and deleting product records. A dedicated model module encapsulates the corresponding SQL queries and manages the mapping between relational database records and application-level objects. The model performs basic validation and translates request data into database operations, returning structured results to the API layer for serialization.

Frontend Interaction:

The frontend communicates with the product services using RESTful API calls. Administrative interfaces provide forms for adding and editing product information, including images, while customer-facing views retrieve paginated product lists and individual item details for browsing. The client handles input validation, user feedback, and state management, and it coordinates image uploads through the backend upload middleware. Current pricing and inventory information are displayed dynamically in the user interface.

Role-Based Access:

Access to product management functionality is controlled through role-based restrictions. Administrative and worker roles are authorized to create and modify product records, while customers are limited to read-only access to catalog data. These constraints are enforced on the server to prevent unauthorized changes and are reflected in the frontend interface to ensure that users are only presented with permitted actions.

5.3 POS and Sales Processing

Purpose and Scope:

The Point-of-Sale (POS) module supports in-store sales workflows and integrates directly with product management and payment handling. It enables store workers to create orders, manage product line items, apply pricing adjustments, and record payments, thereby unifying sales and inventory data within a single system.

Order Creation and Lifecycle:

An order represents a sales transaction composed of one or more product line items, quantities, pricing details, and metadata such as the responsible worker and timestamp. Orders are created through the POS interface or during customer checkout. Once recorded in the database, inventory quantities are adjusted to reflect sold items, and the order is assigned a status indicating whether payment is pending, completed, or refunded.

Payment Handling:

Payment processing in MARS focuses on recording payment events and associating them with the corresponding orders and customers. The system supports multiple payment methods and updates the financial state of orders based on recorded payments. Changes in payment status can trigger transitions in order state, such as marking an order as paid. For local deployment and academic demonstration, payment handling emphasizes accurate record keeping rather than integration with external payment gateways, which are identified as future enhancements.

Inventory Integration:

Inventory levels are updated as part of the order workflow. When an order is finalized, product stock quantities are decremented to maintain an accurate representation of availability. Safeguards are implemented to prevent orders that would result in negative stock levels. Administrative interfaces also allow authorized users to perform manual inventory adjustments when reconciliation is required.

Role-Based Controls and Audit:

POS operations are restricted to authorized worker roles. Each transaction records the acting user and timestamp to support basic auditing and accountability. Administrative users have additional privileges, including correcting or voiding transactions and viewing aggregated sales information.

5.4 OCR-Based Receipt Processing and Recommendation Services

Service Roles:

The OCR and recommendation components extend the core transactional functionality of the system. The OCR service extracts textual data from uploaded receipt images or scanned documents, supporting semi-automated entry of payment and order information. The recommendation service analyzes product and transaction data to suggest items of potential interest to customers or to assist staff with upselling during POS interactions.

Data Flow:

For OCR processing, uploaded images are received by the backend and passed to the OCR service, which applies optical character recognition using available trained datasets. Extracted information—such as merchant names, item lines, totals, and dates—is validated and linked to relevant payment or order records. When extraction confidence is low, the system allows manual review and correction through administrative interfaces.

The recommendation service consumes historical transaction data and product metadata to generate candidate product suggestions. Input factors include purchase co-occurrence, item categories, and simple heuristics derived from sales frequency. Recommendations are displayed in customer-facing views and within the POS interface to support sales assistance.

Limitations:

Both OCR and recommendation features are implemented as prototypes and are constrained by limited datasets. OCR accuracy depends on image quality and the suitability of trained models, and extraction errors may require manual correction. The recommendation service relies on lightweight heuristics and limited historical data, making its outputs indicative rather than definitive. The system does not claim production-level performance for these components.

Practical Use Cases:

- Automated payment data entry: Staff can upload receipt images to populate payment records, reducing manual data entry and reconciliation effort.
- Sales assistance at POS: Recommendations can suggest complementary accessories or frequently purchased items when a product is selected, supporting upselling.
- Basic sales analysis: Aggregated OCR data and recommendation outputs support simple insights into popular products and sales trends.

Evaluation and Future Work:

As prototype implementations, the OCR and recommendation services highlight several directions for future improvement. These include expanding and refining training datasets, adopting more advanced machine learning models, incorporating feedback mechanisms for continuous learning, and integrating external payment gateways for complete transactional workflows. Such enhancements would support transitioning the system from an academic prototype toward production readiness.

Chapter Six

Discussion

6.1 Achievements

The MARS project successfully achieved its primary objectives by delivering an integrated platform that combines customer-facing e-commerce functionality with internal management tools for an electronics store. The system centralizes product information, supports order creation and tracking, records payments and contracts, and provides mechanisms for employee duty-hour tracking as well as basic project and task coordination.

Role-based access control was implemented to ensure that customers, store workers, and administrative users interact with the system according to their responsibilities. In addition, prototype services for OCR-based receipt processing and product recommendation were developed to demonstrate the feasibility of reducing manual data entry and supporting decision-making during sales workflows. Together, these outcomes confirm that the project meets its intended functional and academic goals.

6.2 System Strengths

Several strengths emerge from the final implementation. First, the system enforces a single source of truth for product, order, and payment data, which reduces duplication and simplifies reconciliation across different operational workflows. This consistency improves reliability and reduces the likelihood of conflicting records.

Second, the modular organization of the backend—separating routes, controllers, models, and services—enhances code clarity and maintainability. This structure makes the system easier to understand, test, and extend incrementally. Third, providing both web and mobile clients improves accessibility for staff and allows system usage to adapt to different operational contexts.

Finally, the inclusion of auxiliary services such as OCR processing and lightweight product recommendations demonstrates how automation can be integrated into retail systems to reduce administrative effort and support staff at the point of sale. Even as prototypes, these components add practical value and illustrate potential directions for future enhancement.

6.3 Technical Challenges

The project encountered several technical challenges typical of full-stack systems developed under time and resource constraints. Integrating multiple functional domains—including product management, POS, contracts, payments, employee tracking, and task management—required careful API design and consistent data modeling to avoid redundancy and inconsistencies.

Handling file uploads and coordinating OCR processing introduced additional complexity, particularly in managing data flow and error handling when extracted text required human verification. Ensuring smooth interaction between upload mechanisms, storage strategies, and downstream processing services required iterative testing and adjustment.

Maintaining consistency in inventory and payment records also posed challenges. Ensuring that inventory updates and payment state changes remain correct under concurrent operations is a non-trivial problem. Given the local deployment context, the system adopts conservative validation and sequencing strategies rather than advanced transactional middleware or distributed locking mechanisms.

Finally, the development of OCR and recommendation features highlighted limitations related to dataset size and model maturity. Achieving a balance between improving model accuracy and delivering a complete system within the available timeframe was a key challenge during implementation.

6.4 Design Trade-offs

Several deliberate design trade-offs were made to balance scope, simplicity, and demonstrable functionality:

Callback-based database access:

The backend uses callback-style interactions with the MySQL database. This approach enabled rapid development using familiar patterns and straightforward SQL execution. However, callback nesting can complicate control flow and error handling. The project explicitly identifies this choice as a refactoring candidate, with a future migration to Promises or async/await recommended for improved readability and maintainability.

Client-side session storage:

User session information is stored on the client using local storage to maintain authentication state. This design simplifies client-side implementation and supports rapid development and testing. However, it provides weaker security guarantees than server-managed sessions or HTTP-only cookies and assumes a trusted deployment environment. As such, this approach is appropriate for demonstration and local use, with more robust session management recommended for production systems.

Local deployment focus:

The system was designed primarily for local or small-server deployment to reduce infrastructure complexity during development. While this enabled full end-to-end validation of system functionality, it limits scalability and operational resilience. Cloud-based deployment strategies, managed storage, and centralized configuration management were intentionally deferred as future work.

These trade-offs reflect the project's academic objectives: delivering a coherent and maintainable prototype within limited time and resources while clearly documenting areas for future improvement.

6.5 Conclusion of Discussion

In summary, the MARS project demonstrates how integrating e-commerce capabilities with internal management functions can reduce operational friction in a retail environment. The system validates its architectural choices while also highlighting challenges that must be addressed for production deployment, including improved asynchronous control flow, stronger session security, scalable infrastructure, and expanded datasets for AI-driven components.

These findings position MARS as a substantial graduation project that balances ambition with practical engineering decisions. The system provides both a functional prototype and a clear roadmap for future enhancement, aligning well with the academic goals of the software engineering program.

Chapter Seven

Conclusions and Future Work

7.1 Conclusions

This project presented MARS, an integrated software system that unifies customer-facing e-commerce functionality with internal management operations for an electronics store. The system combines React-based web and mobile clients, a Node.js and Express backend, and a MySQL relational database to deliver core capabilities. These include centralized product management, point-of-sale and order processing, contract and payment recording, employee duty-hour tracking, and prototype services for OCR-based receipt processing and product recommendations. Implemented using pragmatic engineering choices aligned with academic constraints, MARS demonstrates how a unified platform can reduce data duplication, improve record consistency, and streamline routine retail operations.

The project resulted in a functional prototype that satisfies its primary objectives. It validates consistent resource modeling across catalog and sales domains, applies role-based access control to separate responsibilities and restrict privileges, and integrates auxiliary automation to reduce manual effort in selected workflows. The system architecture emphasizes separation of concerns by clearly dividing client presentation, API orchestration, and database persistence, while also identifying areas that would benefit from further technical refinement.

7.2 Future Work

Several realistic enhancements can be pursued to evolve MARS from an academic prototype toward a production-ready system:

- JWT-based authentication:
  Replace client-stored session data with signed JSON Web Tokens and server-side validation to improve security, enable stateless authentication, and support controlled token expiration and refresh policies.

- Cloud deployment and managed services:
  Migrate the system to cloud infrastructure using managed databases, object storage for file uploads, and container-based deployment. This would improve availability, simplify backups, and enable horizontal scalability.

- Async/await refactoring of data access:
  Refactor callback-based MySQL models to use Promise-based APIs with async/await or adopt a lightweight ORM. This would simplify asynchronous control flow, improve error handling, and enhance maintainability and testability.

- Improved OCR accuracy:
  Expand and refine training datasets, integrate more advanced OCR engines or hybrid models, and introduce human-in-the-loop verification for low-confidence outputs to improve reliability in receipt processing.

- Recommendation model enhancements:

  Transition from heuristic-based recommendations to data-driven approaches such as collaborative filtering or lightweight machine learning models. Introducing feedback mechanisms and A/B evaluation would allow continuous measurement and improvement of recommendation relevance.

- Native mobile application:

  Develop a native mobile application for iOS and Android to improve performance, enable offline functionality, and provide deeper integration with device hardware such as cameras and local storage, enhancing in-store and POS workflows.

Taken together, these enhancements form a clear and practical roadmap for future development. They address the technical limitations identified throughout the project—particularly in security, asynchronous control flow, deployment scalability, and AI component maturity—while preserving the core design principles established by MARS. As such, the project serves both as a completed academic contribution and a solid foundation for further professional development.