

Members: Quoc Nhan Tra, Binod Chhantyal, Solomon Rosenfeld
Date: 11/21/2025
Course: CSCE 4350.001

Project Phase 1: Project Proposal

Project Description

The Retail Auto Parts System project is a database centric application designed by undergrad students to computerize the operations of a small auto parts company. The hypothetical premise of this project is that the business is currently operating with paper records using limited computational capabilities. Our system will be designed to process:

- Customer orders and payments. (In Customer Mode)
- Record inventory management, purchases, and supplier records. (In Store Mode)
- Employee management and reports. (In Store Mode)

Purpose of phase I

The purpose of this phase is to identify and document all problems, solutions, hardware choices, and software choices. It will also document the project timeline, the System Requirement Analysis, and all team member's roles and contributions. This proposal will articulate the concept of the Retail Auto Parts System into a clear actionable plan with a timeline. By evaluating and establishing all aspects of the system into a clear plan, the team will have a structured method for tracking progress.

Project Timeline

| Project Phase | Tasks | Due Date | Deliverables |
|---------------|--------------------------------------|------------|--|
| Phase 1 | Analysis and Conceptual Design | 10/03/2025 | Proposal report |
| Phase 2 | Logical and Physical database design | 10/31/2025 | ER diagram, Schema Diagram |
| Phase 3 | System implementation and testing | 11/21/2025 | Final System Product |
| Phase 4 | Final Presentation | 12/05/2025 | Power Point Presentation and Demonstration |

Problem Encountered and Solutions

Problem 1: Technology Stack Selection

Choosing between different DBMS options (MySQL, PostgreSQL, MongoDB) and web frameworks

Solutions: After evaluating requirements, we selected PostgreSQL for its robust ACID compliance, excellent handling of complex queries, and strong support for relational data modeling essential for inventory management. For the web framework, we chose a modern stack that balances performance with developer productivity.

Why choose PostgreSQL: PostgreSQL offers some advanced features like JSONB support for flexible data storage, transaction handling, and scalability. Additionally, it's open source, well-documented, and has strong community support.

Hardware Requirements

Workstation: Laptop or Desktop

Operating System: Windows/macOS/Linux

Network: connectivity between frontend, backend, and database.

Software and Technology Stack

Database Management System

- PostgreSQL
- Enterprise-grade features, ACID compliance, excellent performance with complex queries
- pgAdmin 4 for database administration

Backend Development

- Framework: Python with Django
- RESTful API design

Frontend Development

- HTML/CSS/Javascript
- Django templates for web interface, creates menu system

System Requirement Analysis

The system must have two modes of operations, “Customer mode” and “Store mode”.

Customer Mode:

- The system must allow ordering products online
- The system shall support payment
- The system shall store customer's personal information (name, address, phone, email) for billing.
- The system shall have user and password authentication (for both customers and employees).
- The system shall allow customers to browse existing products.
- The system shall contain a search system.
- The system shall display details when a product is selected (price, quantity, new/used, ect).
- The system shall store customer history activity.
- The system shall have a “shopping cart” function.

Store mode:

- The system shall store employee information (add/change/delete).
- The system shall have employee log in authentication.
- The system must have a section to:
 - Order new products, reorder, cancel orders, and handle returns.
 - Update database for personnel and parts information.
 - Generate reports on products and employee status.
 - Include delivery information such as order date, delivery date, payment method, and cancellations.

The system shall be menu-driven.

Database Design Overview

Initial draft identifying core entities:

- Customer(Customer_ID, Customer_name, Address, Phone, Email, Username, Password)
- Employee(Employee_ID, Employee_name, Role, Login_ID, Password)
- Auto Parts(Part_ID, Part_name, Quantity, Category, Condition)
- Order(Order_ID, Customer_ID, Order_date, Payment_Method, Total_amount)
- Supplier(Delivery, Supplier_name, Supplier_Contact_info)
- Delivery(Delivery_ID, Order_ID, Date, Status.)

Phase I: Planning and Analysis

- Requirements gathering
- Technology stack selection
- Team formation and role assignment
- Initial documentation
- Deliverable: Phase I Report

Phase II: Database Design and Implementation

- Create detailed Entity-Relationship (ER) diagrams
- Design normalized relational schema with all constraints
- Implement database in PostgreSQL
- Populate with sample/test data
- Validate database design through testing

Phase III: System Development, Testing & Deployment

- Develop backend API and business logic
- Create frontend user interface
- Integrate all system components
- Perform comprehensive testing
- Deploy to production environment
- Create user documentation

Phase IV: Final Presentation

- Present the complete project to the class
- Demonstrate functionality and project outcomes

Team Member Contribution

Quoc Nhan Tra – Requirements & Frontend Specialist

- Phase I (Analysis & Conceptual Design):
 - Collect user requirements (customer/store mode features).
 - Draft system requirement analysis section.
 - Help with hardware/software/DBMS selection.
- Phase II (Database Design):
 - Contribute to ER diagrams by focusing on customer- and store-facing entities (e.g., Customers, Orders, Payments).
 - Help normalize relations and refine schema.

- Phase III (Implementation & Testing):
 - Develop frontend (UI/UX) for customer/store menus.
 - Connect frontend forms to backend API.
 - Write test cases for user interaction (login, search, ordering).
- Programming Focus: Frontend development, user input validation, integration with backend.

Binod Chhantyal – Database & Backend Specialist

- Phase I:
 - Research DBMS options and justify choice (MySQL, PostgreSQL, etc.).
 - Contribute to system plan and data flow.
- Phase II:
 - Contribute to ER diagram and schema design (tables, keys, constraints).
 - Insert sample data for testing.
- Phase III:
 - Implement backend API (CRUD, authentication, reports).
 - Optimize SQL queries and indexing.
 - Test queries and backend response.
- Programming Focus: Database schema creation, SQL queries, backend API.

Solomon Rosenfeld – System Integrator & Tester

- Phase I:
 - Document project plan and timeline.
 - Draft Phase I proposal with contribution requirements with other members.
- Phase II:
 - Assist with schema designs and integrity checks (foreign keys, normalization).
 - Prepare ER/schema diagrams for the report.
- Phase III:
 - Develop/Program business logic (order processing, employee management).
 - Develop system integration (connecting frontend, backend, and database).
 - Lead system testing & demonstrations
- Programming Focus: Middleware code, integration testing, error handling.

Project Phase 2: Logical and Physical Database Design

Introduction/Purpose of Phase II

The purpose of this phase is to expand on the problems encountered in Phase I, document revisions to the Phase I report, and team member contribution. It will also document the ER Diagrams, Schema Diagram, and the results of the database schema creation. By expanding on these aspects, the team will be able to further refine the architecture of the software and improve the problematic aspects.

Phase 1 and Revisions

Initially, we agreed on using PostgreSQL as our DBMS. However, we encountered problems while using pgAdmin 4, and in the end decided to switch to DBeaver as our database administration tool, as it is more intuitive and functional. Additionally, after analyzing the project requirements, we identified additional core entities and attributes which were added to the ER diagram and relational schema.

Problems Encountered and Solutions

Problem 1: Relationship complexities, such as some many to many relationships caused anomalies.

Solution: In order to clear this problem, we created junction tables such as the case with Inventory and Orderitem with composite primary keys. For example, the primary key in Inventory is created with (store_id, part_id), which becomes a unique key. This method helps eliminate duplicates and helps maintain integrity.

Problem 2: When we first started defining objects we faced difficulty determining which should become entities and which should be attributes. Some data overlapped, and several attributes were unintentionally excluded. Each time we revised the ER diagram, new dependencies appeared that we forgot which requires updating the schema.

Solution: We had to go back and carefully review the project requirements and clearly identify the entity's purpose. In the end we created a list defining the entities and attributes one by one as required by the project outline, then began to proceed from there.

Documentation Produced in Phase II

Defining entities and attributes

- **Store** — store_id (PK), name, phone, address_line1/2, city, state, postal_code
- **Customer** — customer_id (PK), full_name, customer_email, customer_phone, created_at, username, password
- **Employee** — employee_id (PK), store_id (FK), full_name, role, email, username, password
- **Supplier** — supplier_id (PK), name, contact_email, phone, Address
- **Auto_Part** — part_id (PK), name, category, condition, unit_price, reorder_level
- **Inventory** — (store_id, part_id) (PK, FK), quantity_on_hand
- **Purchase_Order** — po_id (PK), store_id (FK), supplier_id (FK), order_date, expected_date, status
- **Order_Item** — (po_id, part_id) (PK, FKs), quantity, unit_cost
- **Customer_Order** — order_id (PK), customer_id (FK), store_id (FK), order_date, status
- **OrderItem** (assoc.) — (order_id, part_id) (PK, FKs), qty, unit_price
- **Payment** — payment_id (PK), order_id (FK), payment_method, amount, paid_date, card_last_four_digit, authentication_code
- **Delivery** — delivery_id (PK), order_id (FK), ship_date, delivery_date, employee_id(FK), tracking_number, deilvery_status
- **ReturnItem** — return_id (PK), order_id (FK), part_id (FK), quantity, reason, created_date

Entity-Relationship Diagram

E-R Diagram for Auto Parts Retail System

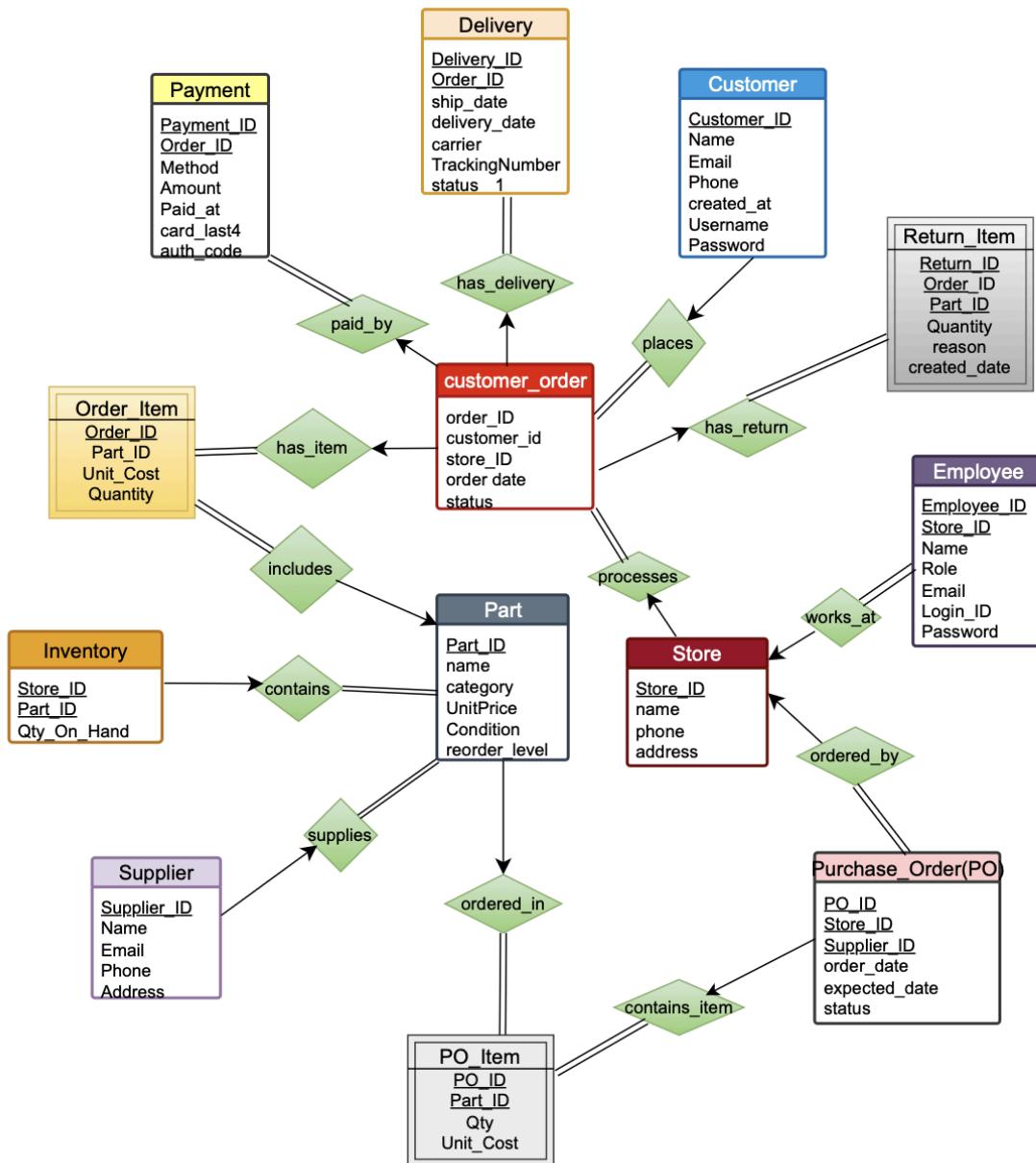


Figure1: E-R diagram

Relational Schema

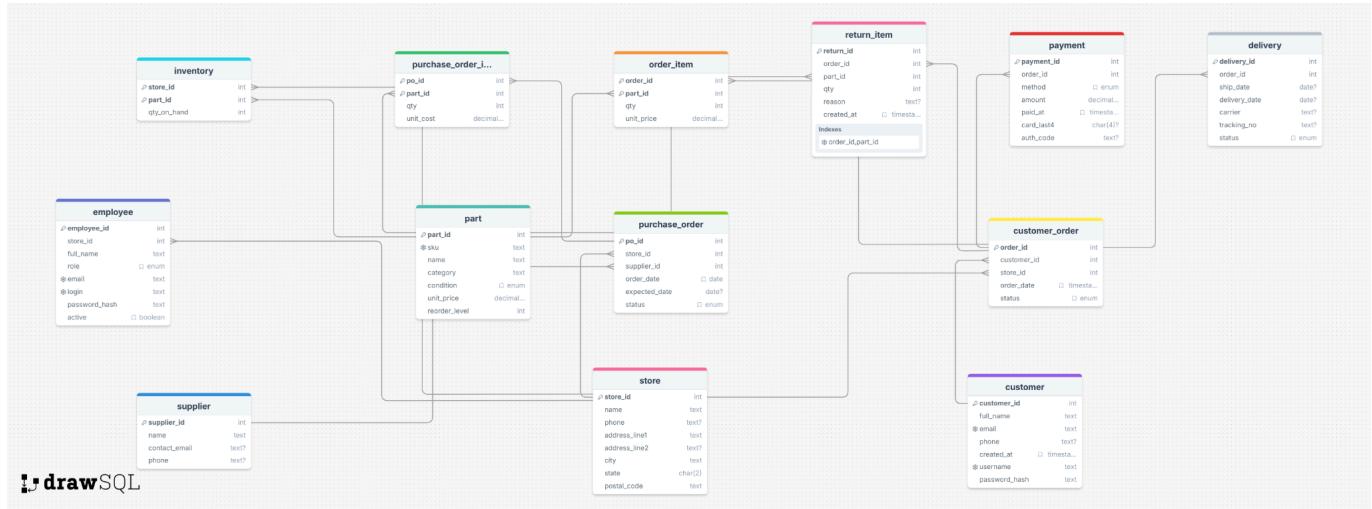


Figure 2: Relational Schema Diagram

Sample Data and Output Results

CREATE TABLE

DBeaver 25.2.3 - <postgres> Retail_parts

```

CREATE TABLE customer (
    customer_id INT NOT NULL,
    full_name TEXT,
    @email TEXT,
    phone TEXT,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    status TEXT NOT NULL CHECK (status IN ('created', 'sent', 'received', 'cancelled'))
);

CREATE TABLE purchase_order_item (
    po_id INT NOT NULL REFERENCES purchase_order(po_id) ON DELETE CASCADE,
    part_id INT NOT NULL REFERENCES part(part_id) ON DELETE RESTRICT,
    qty INT NOT NULL CHECK (qty > 0),
    unit_cost NUMERIC(10,2) NOT NULL CHECK (unit_cost >= 0),
    PRIMARY KEY (po_id, part_id)
);

```

pg_tables 1 × part 1 (2) customer_order 1 (3) payment 1 (5)

```

SELECT tablename FROM pg_tabledef WHERE schemaname = 'public' | Data filter is not supported

```

| Grid | AZ tablename |
|------|---------------------|
| 1 | customer |
| 2 | customer_order |
| 3 | delivery |
| 4 | employee |
| 5 | inventory |
| 6 | order_item |
| 7 | part |
| 8 | payment |
| 9 | purchase_order |
| 10 | purchase_order_item |
| 11 | return_item |
| 12 | store |
| 13 | supplier |

Records: 13 / 13

0 row(s) fetched - 0.0s, on 2025-10-31 at 17:33:34

Parts table output

```
SELECT sku, name, category, condition, unit_price FROM part ORDER BY name;
```

The screenshot shows the pgAdmin interface with a script editor and a results grid.

Script Editor:

```
<practice_postgres> Script *<postgres> Retail_parts <-->
  INSERT INTO part (sku, name, category, condition, unit_price, reorder_level) VALUES
  ('BRK-001', 'Brake Pad Set', 'Brakes', 'new', 39.99, 10),
  ('FLT-010', 'Oil Filter', 'Engine', 'new', 8.49, 25),
  ('SPK-100', 'Spark Plug', 'Ignition', 'new', 5.99, 50);
  |
  -- Stock the store with initial inventory
  INSERT INTO inventory (store_id, part_id, qty_on_hand)
  SELECT 1, part_id,
    CASE sku WHEN 'BRK-001' THEN 50 WHEN 'FLT-010' THEN 200 ELSE 120 END
  FROM part;
```

Results Grid:

| | A-Z sku | A-Z name | A-Z category | A-Z condition | 123 unit_price |
|---|---------|---------------|--------------|---------------|----------------|
| 1 | BRK-001 | Brake Pad Set | Brakes | new | 39.99 |
| 2 | FLT-010 | Oil Filter | Engine | new | 8.49 |
| 3 | SPK-100 | Spark Plug | Ignition | new | 5.99 |

Team Member Contribution

Quoc Nhan Tra

- Implemented tables in PostgreSQL
- Helped define entity, attributes, and datatypes
- Tested queries and provided samples with screenshots
- Aided in compiling Phase II report

Binod Chhantyal

- Drew ER diagram
- Designed relational schema
- Defined entity, attributes, and datatypes

Solomon Rosenfeld

- Validated normalization process
- aided in establishing referential integrity
- Compiled Phase II report

Conclusion

In phase II, we were able to successfully create an executable relational database schema. The database we created enforces strong integrity constraints, and supports major operations required by the Retail Auto Parts System. The ER and Schema diagrams illustrate a conceptual map for the logical model. Additionally, we were able to produce screenshots using sample data for execution to confirm the database is working within the scope of the designs and constraints. Lastly, improvements to planning the software architecture were made through identifying problems surfaced from practical testing and implementation.

Project Phase 3: System Implementation, Testing & Demonstration

1. Introduction

1.1 Project Overview

This document presents the comprehensive Phase III report for the Retail Auto Parts System, a database-driven software solution designed to modernize and streamline the operations of a small auto parts retail company. The system addresses the critical need to transition from a paper-driven environment to a fully computerized operation that supports business expansion and multi-store management.

1.2 Purpose of Phase III

Phase III represents the culmination of our development efforts, focusing on the complete implementation, rigorous testing, and demonstration of the Retail Auto Parts System. This phase builds upon the conceptual design from Phase I and the logical/physical database design from Phase II, delivering a fully functional product that meets all specified requirements and operational needs.

1.3 System Architecture

The implemented system consists of three core components:

Backend: Django 4.2, Django REST Framework, PostgreSQL

Frontend: Django templates (HTML/CSS), simple JS fetch for login

Database: PostgreSQL

The system supports two distinct operational modes:

- **Customer Mode:** Web-based interface for customers to browse parts, place orders, manage accounts, and track deliveries
- **Store Mode:** Employee interface for inventory management, order processing, report generation, and administrative functions

Revisions

Since the Phase II report, there has been no additional revisions to the chosen technology stack for the project. The frameworks that were outlined in the design phase have been satisfactory for the requirements of the system functions. After the design phase, the practical implementations were made by closely following the design of the entity and attribute diagrams that were prepared.

In the implementation point of this phase, there were minor revisions made that arose from encountered problems. These were small scale choices and not representative of the broad scope of the system design. These smaller scale revisions are outlined in the next section below.

Problems Encountered and Solutions

Problem 1: During early stages of implementation, customers and employees login even though their credentials exist and are valid. This is the result of Django storing passwords using secure hashing algorithms by default, and initially the passwords were stored in plaintext, inserted into PostgreSQL tables. As a result, Django's authentication cannot compare hashed passwords with plain text, causing the failed login attempts.

Solution: To resolve this issue, we used Django's built in standardized password handling. `make_password()` was used to hash all existing passwords, and `check_password()` was used during authentication to check user credentials.

Problem 2: When users attempted to filter auto parts by category or conditions, the catalog often crashed. This was caused by missing URL parameters, for example, visiting `/catalog/?category=` sent blank values, which the view processes as a normal filter.

Solution: We updated catalog view logic to handle cases where filters were not provided or are empty. This was done by adding a default/fallback query such as

`AutoPart.object.all()`. Additionally, we had to make sure that request parameters exist before filtering, using conditional filters when parameters are valid. Finally, the frontend links were updated to ensure they send valid string query.

Problem 3: When the navigation bar was being implemented to include more features, the Logout link disappeared. This was due to changes made in `base.html` where conditional blocks determined whether the user was logged in or not. When the sign in logic was being updated, the logout block was unintentionally removed, causing users to be permanently stuck "signed in", without a way to sign out.

Solution: In the end, we had to rebuild the navigation bar to differentiate between being logged in and logged out. Signed in was displayed when no user sessions existed, and Logout was displayed for users who have been authenticated.

Problem 4: While integrating the frontend with the database, several inconsistencies appeared in the data. These inconsistencies include auto parts referencing non-existent store ID, displaying blanks, and PostgreSQL sequences becoming out of sync with manually inserted items.

Solution: In the end we had to reset the database and reset SQL settings. After resetting all the data input and starting fresh, we created sample auto parts, customers, and employees accounts directly through Django. This created a reliable foundation dataset that works across all features.

Documentation Produced in Phase III

System Setup Instructions

1. Clone the repository and navigate to the backend directory:

- **git clone**
- **cd retail-auto-parts-system-main/backend**

2. Create a Python virtual environment:

- **python3 -m venv .venv**

Activate the virtual environment:

- On macOS/Linux:
- **source .venv/bin/activate**
- On Windows (PowerShell):
- **.venv/Scripts/Activate.ps1**

3. Install required Python packages:

- **pip install -r requirements.txt**

4. Apply database migrations and start the system:

Run migrations:

- **python3 manage.py migrate**

Create an admin (superuser) account:

- **python3 manage.py createsuperuser**

Launch the application server:

- **python3 manage.py runserver**

5. Access the system through a web browser:

Home Page: <http://127.0.0.1:8000/>

Customer Login: <http://127.0.0.1:8000/customer/login/>

Employee Login: <http://127.0.0.1:8000/employee/login/>

Admin Panel (superuser only): <http://127.0.0.1:8000/admin/>

Getting Started

1. Access the Django Admin Panel

Open the following URL in a browser:

<http://127.0.0.1:8000/admin/>

Log in using the superuser credentials created during setup.

2. Create a Customer Account

Inside the Django admin dashboard:

- Select “Customers” from the sidebar
- Click “Add Customer”
- Enter all required fields (full name, email, phone, username, password, etc.)
- Save the record

This account can now be used to log into the customer-facing website.

3. Log in as a Customer

After adding some auto parts to the database:

- Log out of the admin panel
- Go to: <http://127.0.0.1:8000/customer/login/>
- Enter the customer username and password created earlier

4. Begin using the customer features:

Once logged in, the customer may:

- Browse and search available auto parts
- Filter by category or condition
- View detailed product information
- Add items to the shopping cart
- Complete purchases
- Review past orders through the order history page

Figure1: Django Administration page login: <http://127.0.0.1:8000/admin/>

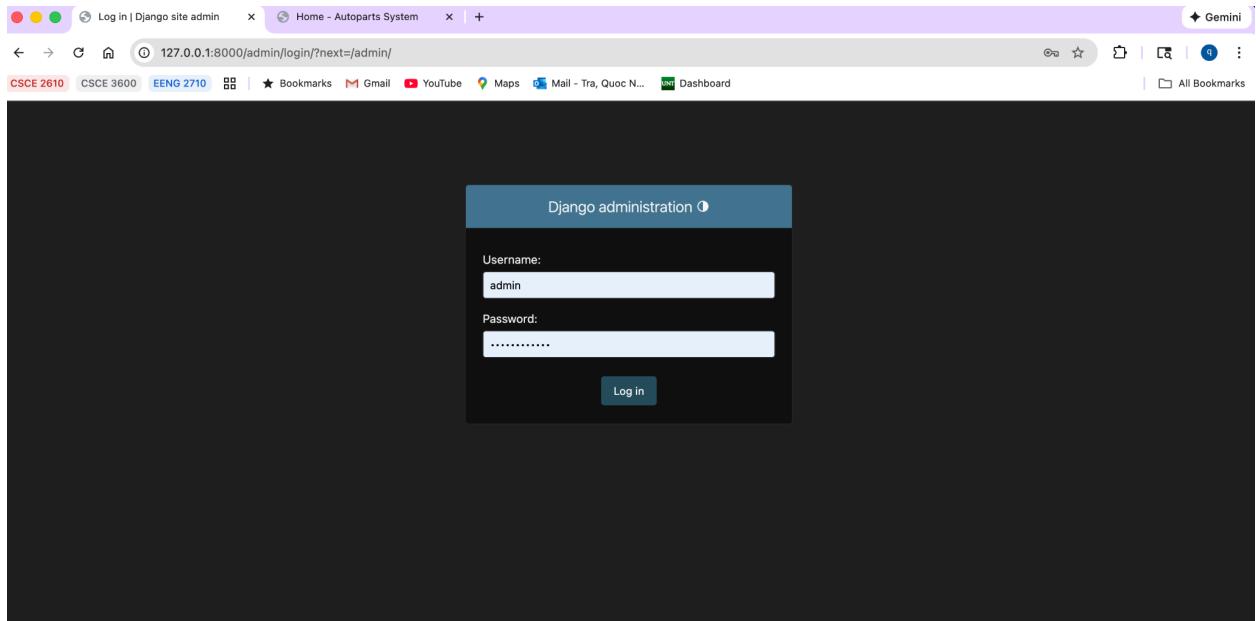


Figure2: Django Main Dashboard

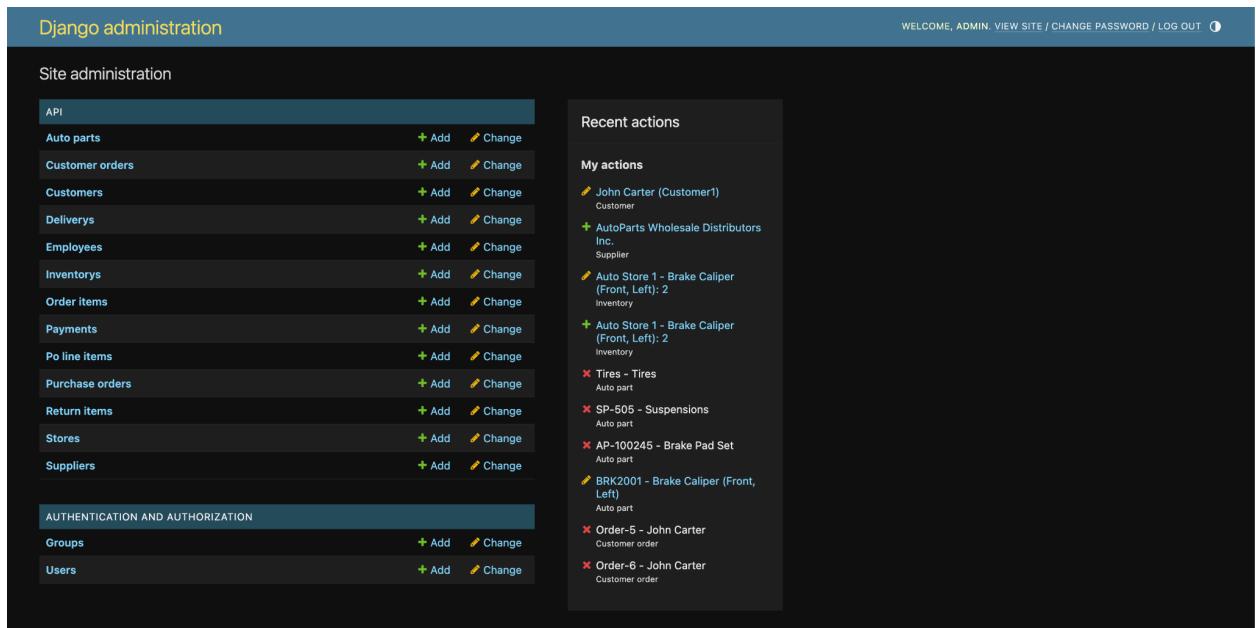


Figure3: Django Customer API: Create Customer Accounts with Authentication Credential

The screenshot shows the Django Admin interface for creating a customer account. The top navigation bar includes 'WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT'. The left sidebar has a 'Start typing to filter...' search bar and lists various models: API, Auto parts, Customer orders, Customers, Deliverys, Employees, Inventorys, Order items, Payments, Po line items, Purchase orders, Return items, Stores, and Suppliers. Under 'AUTHENTICATION AND AUTHORIZATION', there are 'Groups' and 'Users' sections. The main content area is titled 'Change customer' for 'John Doe (customer2)'. It contains fields for 'Full name' (John Doe), 'Customer email' (John@email.com), 'Customer phone' (0987654321), 'Username' (customer2), and 'Password' (pbkdf2_sha256\$600000\$WLy7j36FzX4vfC). At the bottom are buttons for 'SAVE', 'Save and add another', 'Save and continue editing', and a red 'Delete' button.

Figure4: Populate Database with Auto Parts

The screenshot shows the Django Admin interface for managing auto parts. The left sidebar lists the same models as Figure 3, including 'Auto parts' which is currently selected. The main content area is titled 'Select auto part to change'. It shows a list of 18 selected auto parts, each with a checkbox and a description. The descriptions include: BDY3001 - Taillight Assembly, BDY2001 - Front Bumper, BDY1001 - Side Mirror (Left), EXH3001 - Exhaust Pipe, EXH2001 - Catalytic Converter, EXH1001 - Muffler, ELE3001 - Power Window Motor, ELE2001 - Start Motor, ELE1001 - Ignition Coil, SUS3001 - Strut Assembly, and SUS2001 - Control Arm. A 'Go' button and a status message '0 of 18 selected' are at the top of the list. A 'ADD AUTO PART +' button is located in the top right corner.

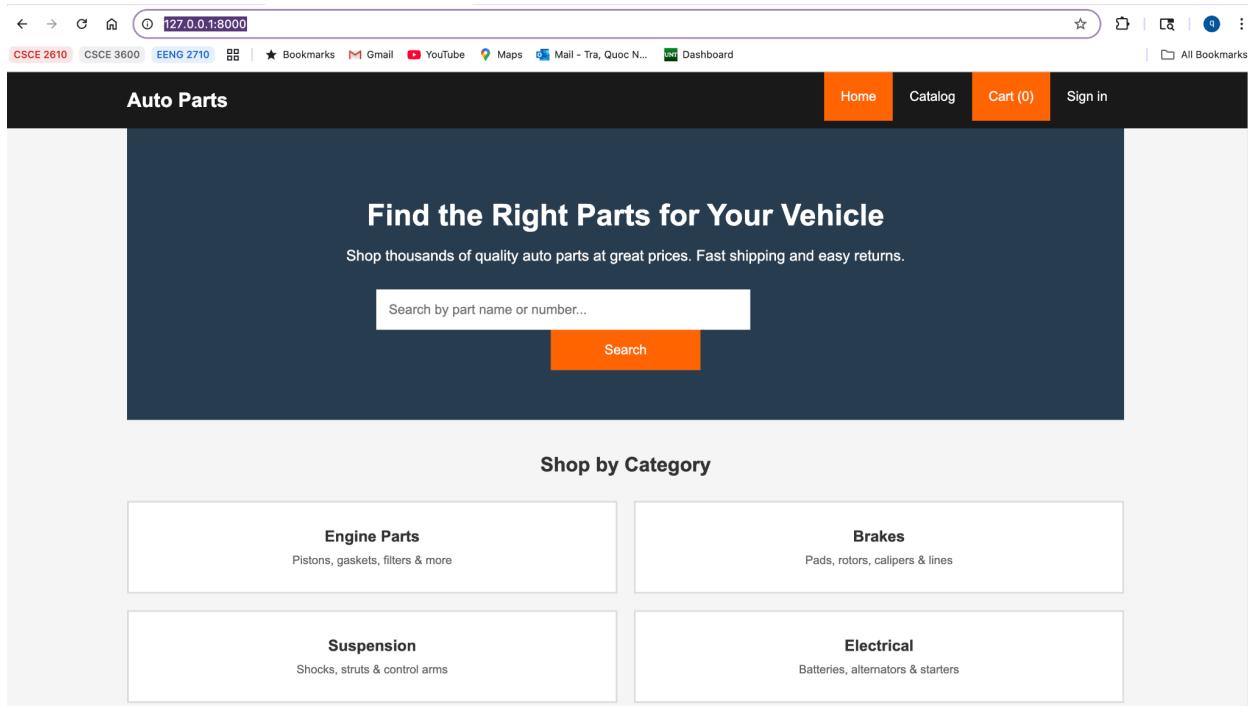
Figure5: Create Users/Managers

The screenshot shows the Django administration interface under the 'Users' section. A new user named 'Employee' is being created. The 'Personal info' section includes fields for first name ('quoc nhan'), last name ('tra'), and email address ('exmaple@mail.com'). In the 'Permissions' section, the 'Active' and 'Staff status' checkboxes are selected. The 'Superuser status' checkbox is unselected. The sidebar on the left lists various models like Auto parts, Customer orders, and Employees.

Figure6: Assign Employees Role

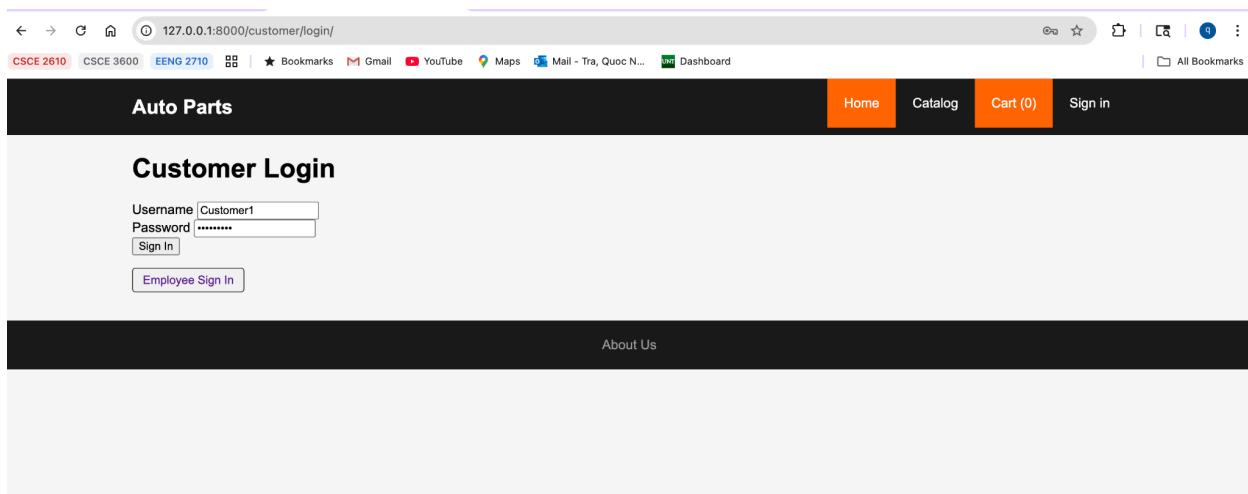
The screenshot shows the Django administration interface under the 'Users' section. A user's permissions are being assigned. Under 'User permissions', several options are listed, including 'api | payment | Can add payment' and 'api | delivery | Can change delivery'. These permissions are being moved from the 'Available user permissions' list to the 'Chosen user permissions' list. The 'Important dates' section shows the user's last login (2025-11-21) and date joined (2025-11-21). The sidebar on the left lists various models like Auto parts, Customer orders, and Employees.

Figure7: Retail-Auto-Parts-System Home: http://127.0.0.1:8000/



The screenshot shows a web browser window for the URL <http://127.0.0.1:8000/>. The page has a dark blue header bar with the text "Auto Parts" on the left and navigation links "Home", "Catalog", "Cart (0)", and "Sign in" on the right. Below the header is a large dark blue banner with the text "Find the Right Parts for Your Vehicle" and a subtext "Shop thousands of quality auto parts at great prices. Fast shipping and easy returns." A search bar with placeholder text "Search by part name or number..." and an orange "Search" button are centered below the banner. Below the search area, there is a section titled "Shop by Category" featuring four categories in boxes: "Engine Parts" (Pistons, gaskets, filters & more), "Brakes" (Pads, rotors, calipers & lines), "Suspension" (Shocks, struts & control arms), and "Electrical" (Batteries, alternators & starters).

Figure8: Public Facing Customer Website: http://127.0.0.1:8000/customer/login/



The screenshot shows a web browser window for the URL <http://127.0.0.1:8000/customer/login/>. The layout is identical to Figure 7, with a dark blue header bar, a "Customer Login" section containing a form with fields for "Username" (Customer1) and "Password" (*****), and a "Sign In" button. Below the form is a link "Employee Sign In". At the bottom of the page, there is a black footer bar with the text "About Us".

Figure9: User Log in and Search: by name or ID

The screenshot shows a web browser window for an Auto Parts website. The URL in the address bar is 127.0.0.1:8000. The top navigation bar includes links for CSCE 2610, CSCE 3600, ENG 2710, Bookmarks, Gmail, YouTube, Maps, Mail - Tra, Quoc N..., Dashboard, Home, Catalog, Cart (0), Hi, John Carter, and Logout.

The main content area features a dark blue header with the text "Find the Right Parts for Your Vehicle" and a subtext "Shop thousands of quality auto parts at great prices. Fast shipping and easy returns." Below this is a search bar containing the text "Ignition oil" and a "Search" button.

A section titled "Shop by Category" displays four categories in a grid:

- Engine Parts**: Pistons, gaskets, filters & more
- Brakes**: Pads, rotors, calipers & lines
- Suspension**: Shocks, struts & control arms
- Electrical**: Batteries, alternators & starters

The "Suspension" category is highlighted with an orange border.

A second section titled "Featured Products" displays three products in a grid:

- Ignition Coil**: New, \$39.99. Add to Cart button.
- Shock Absorber (front)**: New, \$79.99. Add to Cart button.
- Taillight Assembly**: Used, \$29.99. Add to Cart button.

Figure10: Dedicated Catalog Page with Filters

The screenshot shows a web browser window with the URL `127.0.0.1:8000/catalog/?category=brakes`. The page has a black header bar with the text "Auto Parts". On the left side, there is a sidebar with two sections: "Filter by Category" and "Filter by Condition". The "Filter by Category" section contains links for All Parts, Engine Parts, Brakes (which is highlighted in orange), Suspension, Electrical, Exhaust, and Body Parts. The "Filter by Condition" section contains links for All Conditions (highlighted in orange), New, Rebuilt, and Used. The main content area is titled "Brakes" and displays a message "3 product(s) found". Below this, there are three product cards:

- Brake Caliper (Front, Left)**
Rebuilt
\$89.50
[Add to Cart](#)
- Brake Pad Set**
New
\$49.99
[Add to Cart](#)
- Brake Rotor**
Used
\$24.99
[Add to Cart](#)

Figure11: Add-to-Cart Function

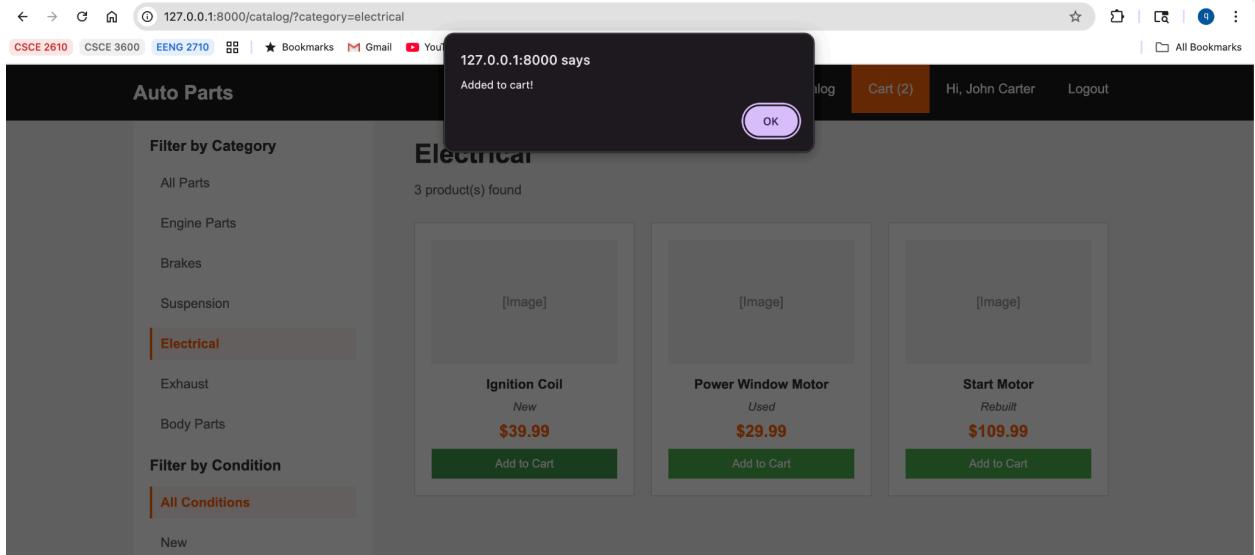


Figure12: Place Order and Payment

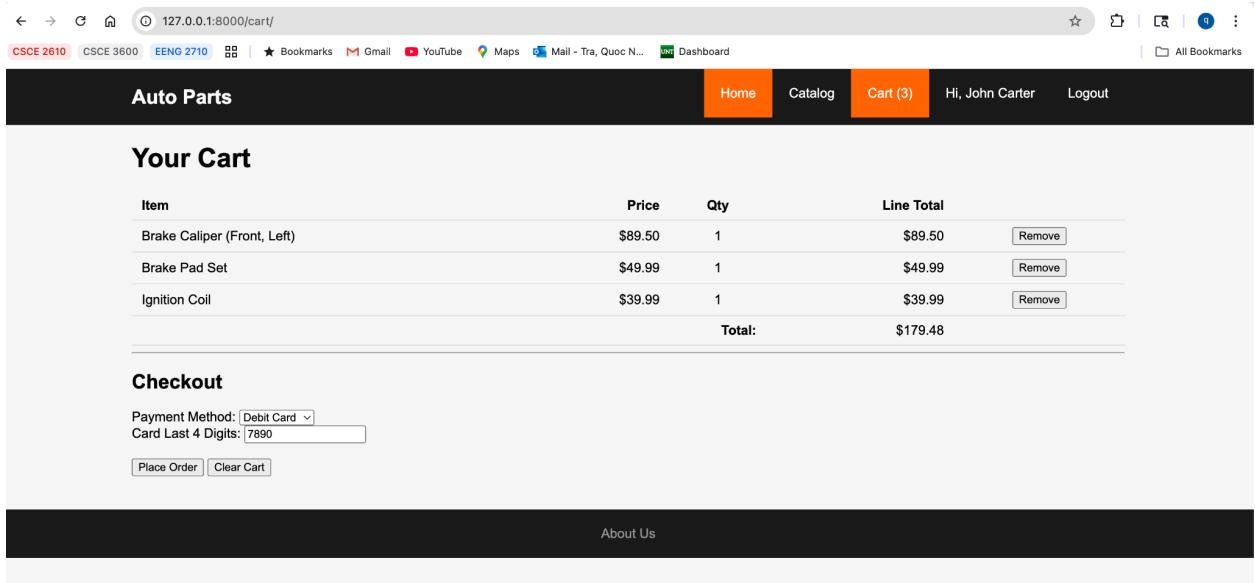


Figure13: View Customer History

The screenshot shows a web browser window with the URL `127.0.0.1:8000/customer/history/`. The page title is "Auto Parts". The top navigation bar includes links for Home, Catalog, Cart (0), Hi, John Carter, and Logout. The main content area is titled "Your Information" and displays the user's name, username, and email. Below this is a section titled "Your Order History" which lists two orders. Each order entry includes the order number, store name, date, status, and total amount. Under each order, there is a detailed table of items with columns for Part, SKU, Condition, Qty, Unit Price, and Line Total.

| Order # | Store | Date | Status | Total | |
|-----------------------------|--------------|------------|------------|------------|------------|
| #14 | Auto Store 1 | 2025-11-22 | PROCESSING | \$179.48 | |
| Items: | | | | | |
| Part | SKU | Condition | Qty | Unit Price | Line Total |
| Brake Caliper (Front, Left) | BRK2001 | REBUILT | 1 | \$89.50 | \$89.50 |
| Brake Pad Set | BRK1001 | NEW | 1 | \$49.99 | \$49.99 |
| Ignition Coil | ELE1001 | NEW | 1 | \$39.99 | \$39.99 |
| #13 | Auto Store 1 | 2025-11-21 | PROCESSING | \$164.48 | |
| Items: | | | | | |
| Part | SKU | Condition | Qty | Unit Price | Line Total |
| Brake Caliper (Front, Left) | BRK2001 | REBUILT | 1 | \$89.50 | \$89.50 |
| Brake Pad Set | BRK1001 | NEW | 1 | \$49.99 | \$49.99 |
| Brake Rotor | BRK3001 | USED | 1 | \$24.99 | \$24.99 |

Figure14: Employee Order Management From Public Site

The screenshot shows a web browser window with the URL `127.0.0.1:8000/employee/login/`. The page title is "Auto Parts". The top navigation bar includes links for Home, Catalog, Cart (0), and Sign in. The main content area is titled "Employee Login" and contains fields for Username (quoc) and Password (****). Below these fields is a welcome message: "Welcome, Quoc Nhan Tra! Role: Delivery". The next section is titled "Order Processing" and shows a table of orders for store #1. The table includes columns for Order #, Customer, Date, Status, Total, and Actions (with options to Mark Shipped or Mark Delivered).

| Order # | Customer | Date | Status | Total | Actions |
|---------|-------------|------------------------|------------|----------|---|
| 14 | John Carter | 11/21/2025, 8:46:31 PM | PROCESSING | \$179.48 | <input type="button" value="Mark Shipped"/> <input type="button" value="Mark Delivered"/> |
| 13 | John Carter | 11/21/2025, 5:41:01 PM | PROCESSING | \$164.48 | <input type="button" value="Mark Shipped"/> <input type="button" value="Mark Delivered"/> |
| 12 | John Carter | 11/21/2025, 5:23:12 PM | PROCESSING | \$214.96 | <input type="button" value="Mark Shipped"/> <input type="button" value="Mark Delivered"/> |

About Us

Figure15: Employee Sign In From Admin Site

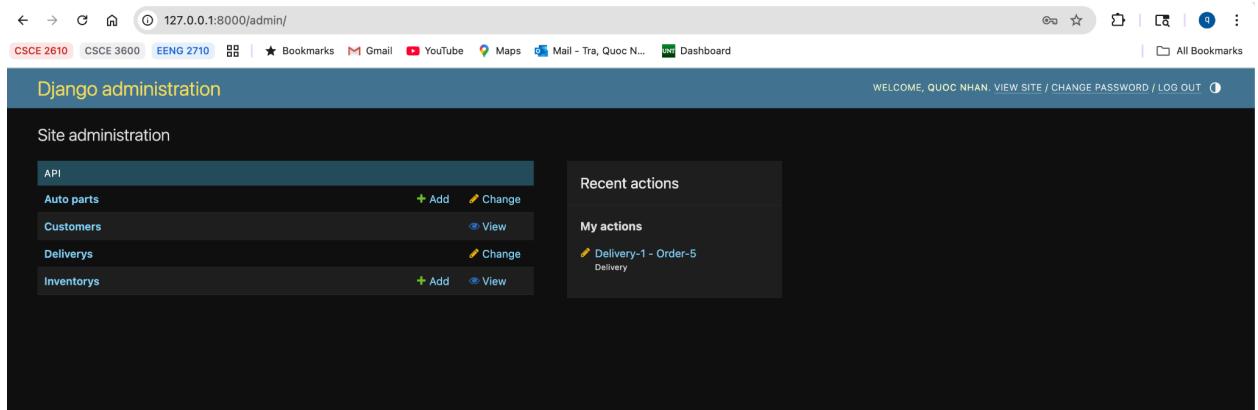


Figure16: Employee Order Management from Admin Site

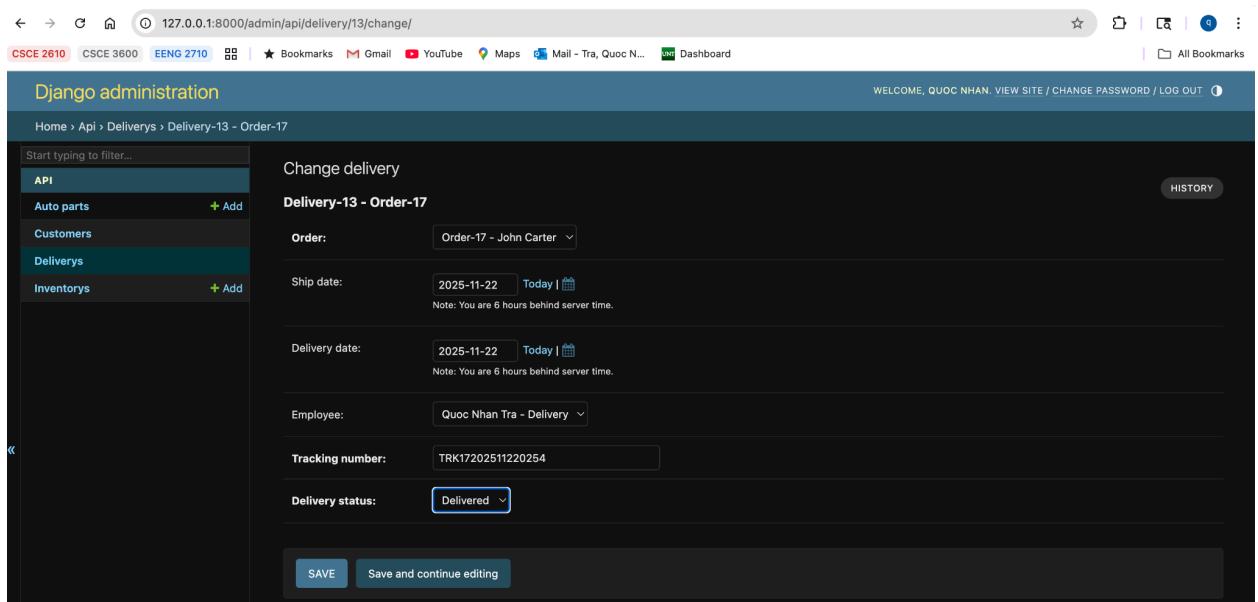


Figure17: Employee Stocking Inventory

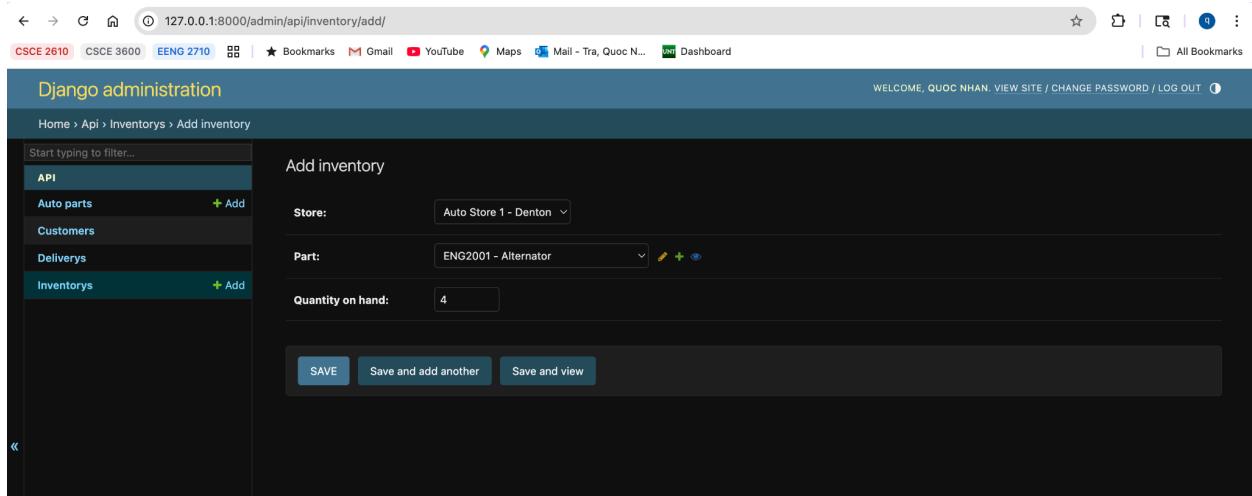


Figure18: frontend/base.html

```

base.html – 4350 project
base.html .../customer > base.html > login.html .../employee &gt; views.py .../api 9+ &gt; views.py .../retail_auto_parts 2 > < > ...
EXPLORER
OPEN EDITORS
4350 PROJECT
  asgi.py
  settings.py
  urls.py 3
  views.py 2
  wsgi.py
  db.sqlite3
  manage.py
  requirements.txt
  frontent
    customer
      cart.html
      history.html
      login.html
    employee
      login.html
    layout
    static/css
      # styles.css
      base.html
      catalog.html
  base.html
  catalog.html

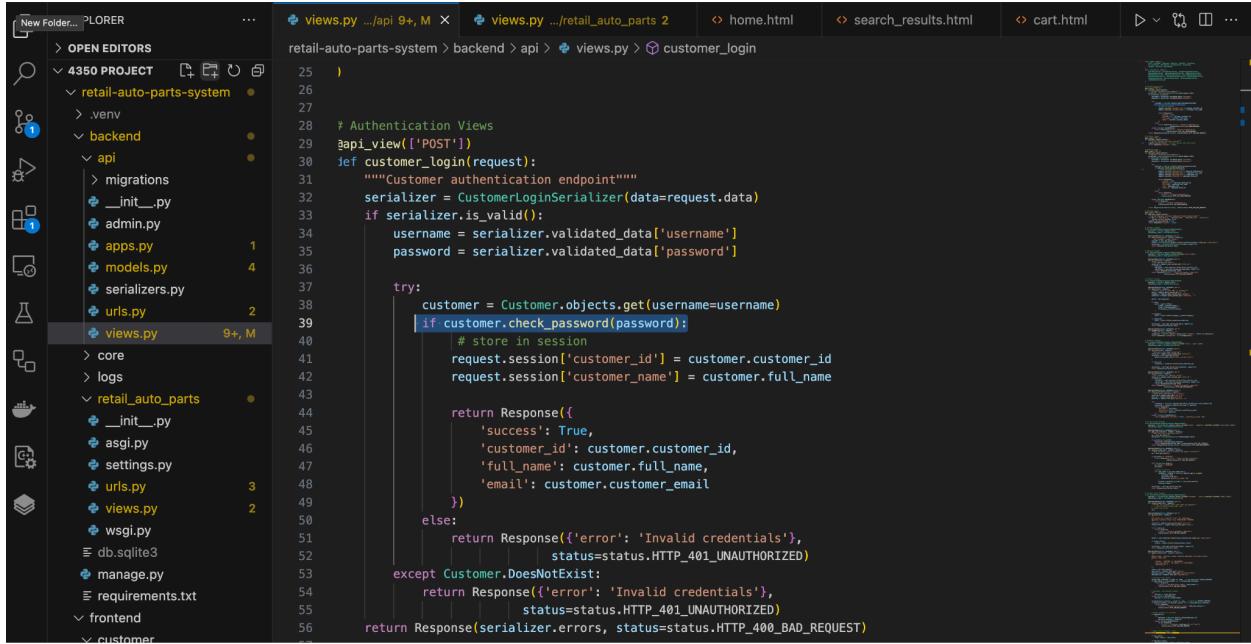
1  <!DOCTYPE html>
2  <html lang="en">
3  {% load static %}
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>{% block title %}Autoparts System{% endblock %}</title>
8    <link rel="stylesheet" href="{% static 'css/styles.css' %}">
9  </head>
10 <body>
11   <nav>
12     <div class="nav-container">
13       <div class="logo">Auto Parts</div>
14       <ul class="nav-links">
15         <li><a href="/" class="active">Home</a></li>
16         <li><a href="/catalog/">Catalog</a></li>
17
18         <!-- Updated cart link -->
19         <li><a id="nav-cart" href="/cart/" style="background-color: #ff6600;">Cart (0)</a></li>
20
21         <!-- Sign in / Logout -->
22         <li><a id="nav-signin" href="{% url 'customer-history-page' %}">Sign in</a></li>
23         <li><a id="nav-logout" href="#" style="display: none;">Logout</a></li>
24
25       </ul>
26     </div>

```

The screenshot shows a code editor with the file `base.html` open. The code defines a navigation bar (`<nav>`) containing a logo (`<div class="logo">Auto Parts</div>`), a list of links (`<ul class="nav-links">`), and a cart link (`Cart (0)`). Below the navigation bar, there are sections for sign in and logout, each with a link (`Sign in` and `Logout`).

Master layout for the website: all other pages, home.html, catalog.html, cart.html extend this file. Defines the main page layout, containing links to home, catalog, customer login and logout.

Figure19: backend/api/views.py



```
25 )
26
27
28 # Authentication Views
29 @api_view(['POST'])
30 def customer_login(request):
31     """Customer authentication endpoint"""
32     serializer = CustomerLoginSerializer(data=request.data)
33     if serializer.is_valid():
34         username = serializer.validated_data['username']
35         password = serializer.validated_data['password']
36
37     try:
38         customer = Customer.objects.get(username=username)
39     except Customer.DoesNotExist:
40         return Response({'error': 'Invalid credentials'},
41                         status=status.HTTP_401_UNAUTHORIZED)
41
42     if customer.check_password(password):
43         # store in session
44         request.session['customer_id'] = customer.customer_id
45         request.session['customer_name'] = customer.full_name
46
47         return Response({
48             'success': True,
49             'customer_id': customer.customer_id,
50             'full_name': customer.full_name,
51             'email': customer.customer_email
52         })
53     else:
54         return Response({'error': 'Invalid credentials'},
55                         status=status.HTTP_401_UNAUTHORIZED)
56
57     return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Contains logic for handling user requests for websites. URLs visited by users are each connected to a view function responsible for reading and writing the data stored in the database, processing, and returning an HTML page or redirect. In the **customer_login** views function is responsible for authenticating customers with usernames and passwords. It receives the information from the frontend, validates it by checking it against the database, and then establishes a session for the user.

Figure20: backend/api/[urls.py](#)

```
# Create router and register viewsets
router = DefaultRouter()
router.register(r'customers', views.CustomerViewSet, basename='customer')
router.register(r'employees', views.EmployeeViewSet, basename='employee')
router.register(r'parts', views.AutoPartViewSet, basename='autopart')
router.register(r'inventory', views.InventoryViewSet, basename='inventory')
router.register(r'purchase-orders', views.PurchaseOrderViewSet, basename='purchaseorder')
router.register(r'customer-orders', views.CustomerOrderViewSet, basename='customerorder')

urlpatterns = [
    # include router URLs
    path('', include(router.urls)),

    # Authentication endpoints
    path('auth/customer/login/', views.customer_login, name='customer-login'),
    path('auth/employee/login/', views.employee_login, name='employee-login'),

    path('auth/customer/login/', views.customer_login, name='customer-login-api'),
    path('auth/customer/logout/', views.customer_logout, name='customer-logout-api'),

    # Report endpoints
    path('reports/daily-sales/', views.daily_sales_report, name='daily-sales-report'),
    path('reports/inventory/', views.inventory_report, name='inventory-report'),
    path('reports/employee-performance/', views.employee_performance_report, name='employee-performance-report'),

    path('cart/add/', views.cart_add, name='cart-add-api'),
    path('cart/summary/', views.cart_summary, name='cart-summary-api'),
    path('cart/clear/', views.cart_clear, name='cart-clear-api'),
    path('cart/remove/', views.cart_remove, name='cart-remove-api'),
    path('cart/checkout/', views.cart_checkout, name='cart-checkout-api'),
]
```

Defines URL routing path for the api, mapping specific paths such as **auth/customer/** and **/cart/remove/** to view functions in **views.py**

Team Member Contributions

Quoc Nhan Tra: Full-Stack (frontend & backend) Development

- Built public customer-facing webpage UI (home page, catalog, cart, order history).
- Developed backend logic supporting catalog filtering, product search, and cart operations.
- Integrated frontend templates with Django backend views and database models.
- Designed global site navigation and fixed incorrect routing issues (e.g., logout button).
- Performed debugging including fixing catalog filter errors and eliminating duplicate alert banners.

- Assisted testing user workflows, and ensuring smooth customer/employee mode transitions.

Binod Chhantyal: Backend & Database Development

- Implemented backend CRUD functionality for products, employees, orders, and suppliers.
- Developed authentication for customers and employees.
- Created the business logic for purchase orders and payments.
- Refined SQL queries and improved indexing for large catalog and order operations.
- Defined Django models and managed database migrations to reflect the schema generated in Phase II.
- Performed backend integration of API and testing data consistency.

Solomon Rosenfeld: Frontend Development & System Testing

- Built Store Mode frontend pages including employee dashboards and management interfaces.
- Prepared system-wide base files for UI.
- Implemented frontend UI for customers operations, search menu, catalog page, and item cards.
- Integrated UI components with backend logic for order management and delivery workflows.
- Conducted comprehensive end-to-end testing of all system modules (customer and store modes).
- Ensured database consistency by validating workflows such as checkout, delivery, and returns.
- Prepared demonstration for final presentation.

Conclusion

In Phase III, the Retail Auto Parts System was completely implemented in its entirety, with a fully functioning product that satisfies all specified requirements and operational needs. The team built a database centric retail and inventory management system web application for the purpose to computerize the operations of a small auto parts company. It is complete with two modes: Customer Mode that supports the functions of browsing products, placing orders, managing accounts, and tracking deliveries; and Store Mode that supports inventory management, order processing, report generation, and administrative functions. We maintained the architecture design choices from

Phase II and, with the guidance of the design charts, we have successfully realized our vision outlined from the initial proposal.

