SYSTEM DOCUMENTATION

WENYURE MENDEZ LAFAURIE

2025-11-20

SYSTEM DOCUMENTATION

1. Project Information

Project Name: AUTOMOTIVE WORKSHOP
Student Name: Wenyure Mendez Lafaurie
Course: Database II
Semester: 6
Date: 19/11/2025
Instructor: Jaider Quintero Mendoza

Short Project Description:

This project is a management system designed for an automotive workshop, allowing the administration of customers, vehicles, mechanics, work orders, services, spare parts, insurance companies, appointments, and payments. The system organizes and tracks all workshop operations, including service scheduling, work order creation, insurance validations, service assignments, and spare part usage. The backend manages all business logic and database entities, while the frontend provides an interactive interface for users to register, update, and monitor all workshop activities efficiently.

---

2. System Architecture Overview

2.1 Architecture Description

The system is based on a modular client–server architecture consisting of a Django REST Framework backend, an Angular frontend, and a relational database responsible for storing workshop information.
The architecture allows efficient management of clients, vehicles, mechanics, work orders, services, spare parts, insurance companies, payment records, and scheduled appointments.

The backend exposes RESTful API endpoints that handle CRUD operations, validations, authentication, and database interactions.
The frontend consumes these endpoints to provide users with dynamic interfaces for registering, editing, and tracking all workshop activities.

This structure ensures scalability, maintainability, and clear separation of responsibilities across components.

2.2 Technologies Used

- **Frontend:**
  Angular v20

  PrimeNG

  Bootstrap

- **Backend:**
  Django REST Framework (DRF)

- **Database Engine:**

  MySQL

  MySQL Server (Microsoft SQL Server)

  Oracle

- **Additional Libraries / Tools:**

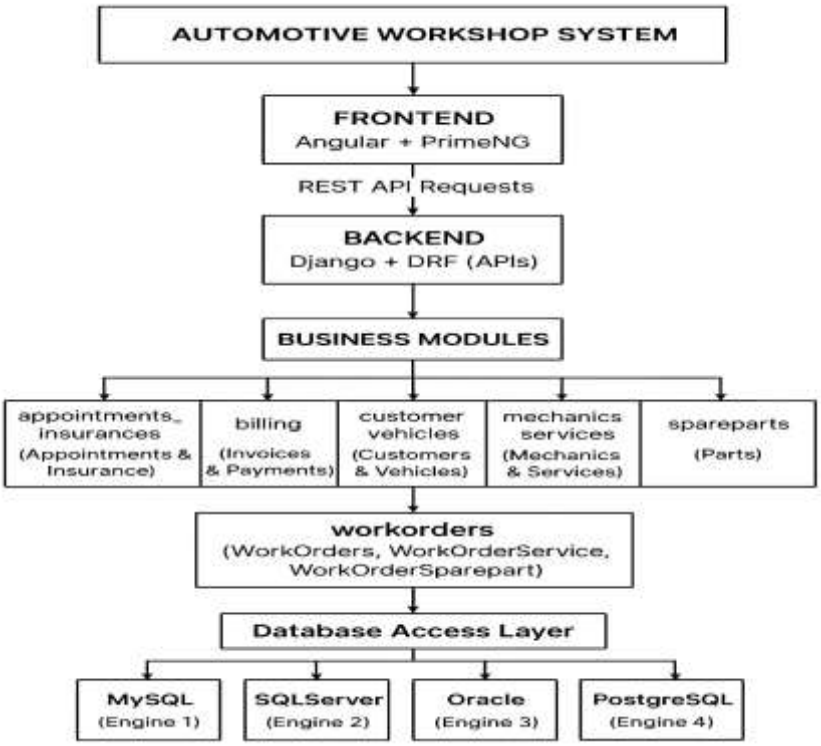
  Node.js & NPM

  RStudio / RMarkdown (.Rmd)

  Postman / Thunder Client — Tools used to test API endpoints.

2.3 Visual explanation of the system's operation

(Insert ASCII diagram or architecture sketch)



---

3. Database Documentation (ENGLISH)

3.1 Database Description

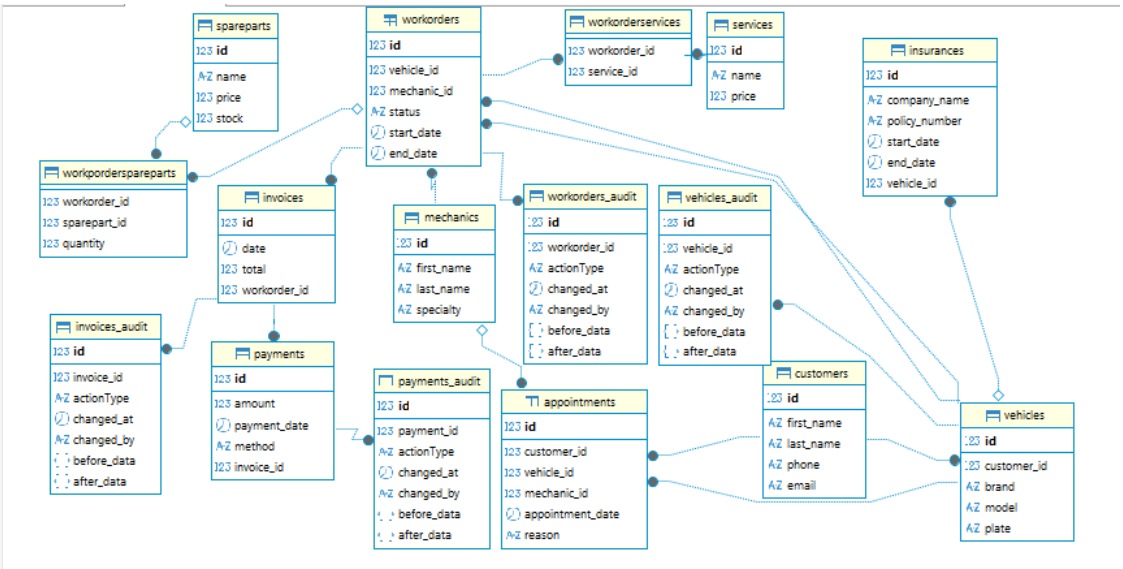
The database was designed to support the operational workflow of an automotive workshop. It centralizes information about customers, vehicles, appointments, mechanics, services, work orders, spare parts, billing, and auditing.

The design follows a modular structure, aligned with the system's business components:

- Customer & Vehicle Management
- Appointments & Insurance
- Work Orders & Operations

- Mechanics & Services
- Spare Parts & Inventory
- Billing (Invoices & Payments)
- Audit Logging for Data Changes

3.2 ERD – Entity Relationship Diagram



3.3 Logical Model

The logical model is built using a normalized relational design to ensure consistency and avoid redundancies.

Main logical entities:

- **Customers**

Stores the personal data of the workshop's customers.

- **Vehicles**

Linked to customers; they contain identifying and descriptive information about the vehicles.

- **Appointments**

Represent scheduled visits for inspection or repair; they connect customers with vehicles.

- **Insurance**

Stores information about the policies linked to each vehicle.

- **Mechanics**

Contains information about the technicians for job assignment.

- **Services**

Represents labor operations (maintenance, diagnostics, repairs).

- **Spare Parts**

Manages the inventory and pricing of spare parts.

- **Work Orders (Main Entity)**

Define the repair task, linking the vehicle, the mechanic, the dates, and the status.

- **Work Order Services**

A junction table that links work orders to the services performed.

- **Work Order Parts**

A junction table that links work orders to the parts used.

- **Invoices**

Contains billing information for completed work orders.

- **Payments**

Stores payment records linked to invoices.

- **Audit Tables**

Tracks changes to the following key tables:

audit_invoices

audit_payments

audit_work_orders

audit_vehicles

3.4 Physical Model (Tables)

Table	Column	Type	PK/FK	Description
customers	id	int	PK	Unique customer identifier
	first_name	Varchar	-	Customer first name
	last_name	Varchar	-	Customer last name
	phone	varchar	-	Contact phone number
	email	varchar	-	Email address
Table	Column	Type	FK/PK	Description
Vehicles	id	int	PK	Unique vehicle indentifier
	customer_id	int	FK–>customer.id	Vehicle owner
	brand	varchar	–	Vehicle brand
	model	varchar	-	Vehcile model
	plate	varchar	–	License plate number
Table	**Column**	**Type**	**PK/FK**	**Description**
appointments	id	int	PK	Appointment indetifier
	customer_id	int	FK–>customer.id	Customer who scheduled
	vehicle_id	int	FK–>vehicles.id	Vehicle being serviced
	appointment_date	datetime	-	Date and time of appoinment
	reason	varchar	-	Visit reason
Table	**Column**	**Type**	**PK/FK**	**Description**
insurances	id	int	PK	Insurance record ID
	company_number	varchar	-	Insurance company
	policy_number	varchar	-	Policy number
	start_date	date	-	policy start date
	end_date	date	-	Policy end date
	vehicle_id	int	FK–>	Insured vehicle
Table	**Column**	**Type**	**PK/FK**	**Description**

Table	Column	Type	PK/FK	Description
mechanics	id	int	PK	Mechanic identifier
	first_name	varchar	-	Mechanic first name
	last_name	varchar	-	Mechanic last name
	specialty	varchar	-	Area of specialization
Table	**Column**	**Type**	**PK/FK**	**Description**
Services	id	int	PK	Service ID
	name	varchar	-	service name
	price	decimal	-	Cost of the service
Table	**Column**	**Type**	**PK/FK**	**Description**
spareparts	id	int	PK	Spare part ID
	name	varchar	-	Name of the part
	price	decimal	-	Price
	stock	int	-	Quantityavailable in inventory
Table	**Column**	**Type**	**PK/FK**	**Description**
workorders	id	int	PK	Work order indentifier
	vehicle_id	int	FK–>vehicles.id	Vehicle associated with the job
	mechanic_id	int	FK → mechanics.id	Assigned mechanic
	status	varchar	-	job status-diagnosis, repair etc
	start_date	datetime	-	When the work started
	end_date	datetime	-	When the work was completed
Table	**Column**	**Type**	**PK/FK**	**Description**
workorderservices	workorder_id	int	FK → workorders.id	Related work order
	service_id	int	FK → spareparts.id	Part used in the work
	quantity	int	-	Units used

Table	Column	Type	PK/FK	Description
Table	**Column**	**Type**	**PK/FK**	**Description**
invoices	id	int	PK	Invoice identifier
	date	datetime	-	Invoice date
	total	decimal	-	Total amount billed
	workorder_id	int	FK–>workorders.id	Related work order
Table	**Column**	**Type**	**PK/FK**	**Description**
payments	id	int	PK	Payment indentifier
	amount	decimal	-	Payment amount
	payment_date	datetime	-	Date of the payment
	method	varchar	-	payment method(cash,card, etc.)
	invoice_id	int	FK–>invoices.id	invoice paid

---

4. Use Cases – CRUD

4.1 Use Case: Create appointments

Actor:Administrador
Description:

The actor registers a new appointment by selecting a Customer, a Vehicle, assigning a Mechanic, choosing the appointment date, and providing the reason for the visit.
Preconditions:

- The Customer exists in the system.

- The Vehicle exists and belongs to that Customer.
- A Mechanic must exist and be available.
- User must have permissions to create appointments
  Postconditions:
- The new appointment is saved in the database.

The appointment becomes available for further processes such as diagnostics or work order creation.
Main Flow:

- The actor navigates to the Appointments module.
- The actor selects Create Appointment.
- The system displays the creation form.
- The actor selects an existing Customer.
- The system filters and displays only vehicles belonging to that customer.
- The actor selects a Vehicle.
- The actor selects a Mechanic.
- The actor selects the Appointment Date.
- The actor enters the Reason for the appointment.
- The actor submits the form.
- The system validates: date, relationships, mechanic availability.
- The system stores the appointment.
- The system confirms the successful creation.

4.2 Use Case: Read [Entity]

Actor:
Administrator

Description:
The actor views the list of scheduled appointments with details like customer, vehicle, mechanic, date, and reason.

Preconditions:

- There must be at least one appointment registered.
- User must have permission to view appointments.

Postconditions:

- The actor sees appointment details or filters appointments as needed.

Main Flow:

1. The actor navigates to the Appointments list.
2. The system retrieves all appointments from the database.
3. The actor may filter by date, customer, vehicle, or mechanic.
4. The actor selects an appointment to view its full details.

The system displays complete information:

ID

Customer

Vehicle

Mechanic

Appointment Date

Reason

4.3 Use Case: Update [Entity]

Actor: Administrator

Description: The actor modifies an existing appointment (date, reason, customer, vehicle, mechanic).

Preconditions:

- The appointment must exist.
- User must have permission to update appointments.

Postconditions:

The updated appointment data is saved correctly in the database.

Main Flow:

- The actor accesses the Appointments list.

- The actor selects an appointment to update.
- The system loads the appointment details in a form.
- The actor updates one or more fields:

Customer

Vehicle

Mechanic

Date

Reason

- The actor submits the form.
- The system validates the data.
- The system updates the appointment.
- A confirmation message is shown.

4.4 Use Case: Delete [Entity]

Actor: Administrator

Description: The actor deletes an appointment that is no longer necessary.

Preconditions:

- The appointment must exist.
- The appointment must NOT be linked to an active work order (if your business rules apply).

Postconditions:

The appointment is permanently removed from the database.

Main Flow:

- The actor opens the Appointments list.
- The actor selects an appointment to delete.
- The system deletes the appointment.

4.1 Use Case: Create Insurance

Actor: Administrator

Description: The actor registers a new insurance policy for a specific vehicle, entering the insurance company, policy number, start date, and end date.

Preconditions:

- The vehicle must exist in the system.
- User must have permissions to create insurance records.
- No active policy should exist for the vehicle if the business rule enforces “one active policy per vehicle.”

Postconditions:

- A new insurance policy is stored in the database.
- The policy becomes associated with the selected vehicle.

Main Flow:

- The actor navigates to the “Insurances” module.
- The actor selects Create Insurance.
- The system displays the insurance creation form.
- The actor selects a Vehicle.
- The actor enters the Company.
- The actor enters the Policy Number.
- The actor selects the Start Date.
- The actor selects the End Date.
- The actor submits the form.
- The system validates the dates, policy number, and vehicle relationship.
- The system stores the new insurance record.
- The system confirms the successful creation.

4.6 Use Case: Read Insurance

Actor: Administrator

Description: The actor views insurance policies and their associated vehicle information.

Preconditions:

- At least one insurance record must exist.
- The user must have permission to read insurance records.

Postconditions:

The system displays the list or specific insurance details.

Main Flow:

1. The actor opens the Insurances list.
2. The system retrieves all insurance records from the database.
3. The actor applies optional filters (company, policy number, vehicle).
4. The actor selects a policy to view detailed information.
5. The system displays:

ID

Company

Policy Number

Start Date

End Date

Associated Vehicle

4.7 Use Case: Update Insurance

Actor: Administrator

Description: The actor updates existing insurance information such as company, dates, or associated vehicle.

Preconditions:

- The insurance record must exist.
- User must have permission to modify insurance data.

Postconditions:

The updated insurance information is saved in the database.

Main Flow:

- The actor opens the list of insurance records.
- The actor selects an insurance policy to update.
- The system loads the form with the current data.
- The actor edits the necessary fields:

Company

Policy Number

Start/End Dates

Vehicle

- The actor submits the updated form.
- The system validates the changes.
- The system saves the updated insurance record.

The system displays a success message.

4.8 Use Case: Delete Insurance

Actor: Administrator

Description: The actor deletes an insurance policy no longer needed or replaced by a new one.

Preconditions:

The insurance record must exist.

The policy must not be required for active legal or billing processes (optional business rule).

Postconditions:

The insurance record is removed from the database.

Main Flow:

- The actor opens the Insurances list.
- The actor selects an insurance policy to delete.
- The system asks for confirmation.
- The actor confirms deletion.

- The system deletes the record.

The system shows a success message.

4.9 Use Case: Create Customer

Actor: Administrator

Description: The actor creates a new customer by entering personal information such as name, email, and phone number.

Preconditions:

User must have permission to create new customer records.

Email format must be valid (business rule).

Postconditions:

- A new customer is stored in the database.
- The customer becomes available for vehicle registration and appointment scheduling.

Main Flow:

- The actor navigates to the Customers module.
- The actor selects Create Customer.
- The system displays the customer creation form.
- The actor enters the Name.
- The actor enters the Email.
- The actor enters the Phone Number.
- The actor submits the form.
- The system validates the input (email format, phone format, required fields).
- The system saves the new customer.
- The system displays a successful creation message.

4.10 Use Case: Read Customer

Actor: Administrator

Description: The actor views customer information, either in list format or detailed view.

Preconditions:

- At least one customer must exist in the system.
- User must have permission to view customers.

Postconditions:

The system displays the requested customer information.

Main Flow:

- The actor navigates to the Customers list.
- The system retrieves all customers.
- The actor may filter by name, email, or phone.
- The actor selects a customer to view details.
- The system displays:

ID

Name

Email

Phone

4.1.1 Use Case: Update Customer

Actor: Administrator

Description: The actor updates an existing customer's information (name, email, phone).

Preconditions:

- The customer must exist.
- User must have permission to edit customer data.
- Email must continue to follow the correct format.

Postconditions:

Updated customer information is stored in the database.

Main Flow:

- The actor opens the Customer list.
- The actor selects a customer to edit.
- The system loads the customer form with current data.
- The actor edits one or more fields (Name, Email, Phone).
- The actor submits the changes.
- The system validates the updated data.
- The system saves the changes.
- The system confirms successful update.

4.1.2Use Case: Delete Customer

Actor: Administrator

Description: The actor deletes a customer record that is no longer needed.

Preconditions:

- The customer must exist.
- The customer must not have vehicles or appointments linked, unless the system allows cascading deletions (business rule).
- User must have delete permissions.

Postconditions:

The customer is removed from the database.

Main Flow:

- The actor accesses the Customer list.
- The actor selects a customer to delete.

The actor confirms the deletion.

4.1.3 Use Case: Create Vehicle

Actor: Administrator

Description: The actor registers a new vehicle for an existing customer, entering vehicle details such as brand, model, license plate, and setting its initial status.

Preconditions:

- The customer must exist in the system.
- The license plate must be unique (business rule).
- User must have permission to create vehicle records.

Postconditions:

- The new vehicle is stored in the database.
- The vehicle becomes available for appointments and work orders.

Main Flow:

- The actor navigates to the Vehicles module.
- The actor selects Create Vehicle.
- The system displays the vehicle creation form.
- The actor selects a Customer.
- The actor enters the Brand.
- The actor enters the Model.
- The actor enters the License Plate.
- The actor sets the Status (Available / In Repair / Inactive).
- The actor submits the form.
- The system validates:
- Unique license plate
- Valid customer
- Valid status
- The system saves the new vehicle.
- The system confirms successful creation.

4.1.4 Use Case: Read Vehicle

Actor: Administrator

Description: The actor views vehicle information, including its status and associated customer.

Preconditions:

- At least one vehicle must exist in the database.
- The user must have permission to view vehicles.

Postconditions:

The system displays the requested vehicle information.

Main Flow:

- The actor opens the Vehicles list.
- The system retrieves all vehicles.
- The actor filters by brand, model, license plate, status, or customer.
- The actor selects a vehicle to view details.

The system displays:

ID

Brand

Model

License Plate

Status

Customer

4.1.5 Use Case: Update Vehicle

Actor: Administrator

Description: The actor updates existing vehicle information such as brand, model, license plate, status, or associated customer.

Preconditions:

The vehicle must exist.

User must have permission to update vehicles.

New license plate (if changed) must still be unique.

Postconditions:

Updated vehicle information is saved correctly in the system.

Main Flow:

- The actor accesses the Vehicles list.
- The actor selects a vehicle to edit.
- The system loads the form with existing data.

The actor edits the necessary fields:

Brand

Model

License Plate

Status

Customer

- The actor submits the form.
- The system validates the data.
- The system saves the updated vehicle.
- The system confirms the update.

4.1.6 Use Case: Delete Vehicle

Actor: Administrator

Description: The actor deletes a vehicle record that is no longer needed.

Preconditions:

- The vehicle must exist.
- The vehicle must not be linked to active appointments or work orders (business rule).
- User must have delete permissions.

Postconditions:

The vehicle is removed from the system.

Main Flow:

1. The actor opens the Vehicles list.

2. The actor selects a vehicle to delete.
3. The system asks for confirmation.
4. The actor confirms deletion.
5. The system removes the vehicle (or denies deletion if linked to active operations).

The system displays a success message.

4.1.8 Use Case: Create Invoice

Actor: Administrador

Description:

Creates a new invoice associated with an existing work order.

Preconditions:

- A work order must exist.
- The user must have permission to create invoices.

Postconditions:

- The invoice is saved in the system.
- The invoice is linked to the selected work order.

Main Flow:

- The user opens the “Create Invoice” form.
- The system displays the required fields.
- The user enters Date, Total, and selects the Work Order.
- The user submits the form.
- The system validates:
- Work order exists.
- Total is a valid amount.
- The system creates the invoice.
- A confirmation message is shown.

4.1.9 Use Case: Read Invoice

Actor: Admininistrador

Description:

Retrieves and displays invoice information.

Preconditions:

At least one invoice must exist.

Postconditions:

The system presents the requested invoice data.

Main Flow

- The user opens the “Invoices” module.
- The system displays a list of invoices.
- The user selects an invoice.
- The system shows: ID, Date, Total, Work Order.

4.1.10 Use Case: Update Invoice

Actor: Admininistrador

Description:

- Updates existing invoice information.
- Preconditions
- The invoice must exist.
- The user must have permission to edit invoices.

Postconditions:

The invoice information is updated in the system.

Main Flow:

- The user selects an invoice to edit.
- The system displays its current data.
- The user edits Date or Total.

- The user submits the changes.
- The system validates the new data.
- The system updates the invoice.
- A success message is shown.

4.2.1 Use Case: Delete Invoice

Actor: Admininistrador

Description:

Deletes an invoice from the system.

Preconditions:

- The invoice must exist.
- It must not be locked by business rules (e.g., payment already processed).

Postconditions:

The invoice is removed from the system.

Main Flow:

- The user selects an invoice to delete.
- The system asks for confirmation.
- The user confirms.
- The system deletes the invoice.

A confirmation message is shown.

4.2.3 Use Case: Create Payment

Actor: Admininistrador

Description:

Registers a new payment for an invoice.

Preconditions

- The related invoice must exist.
- The user must have permission to create payments.

Postconditions

- The payment is saved.
- The payment is linked to the corresponding invoice.
- The invoice's paid status may be updated (if applicable).

Main Flow

- The user opens the "Create Payment" form.
- The system displays the payment fields.
- The user enters Invoice, Amount, Date, and Method.
- The user submits the form.
- The system validates the data.
- The system creates the payment.
- A confirmation message is displayed.

4.2.3 Use Case: Read Payment

Actor

Admininistrador

Description

Displays a list of payments or the details of one payment.

Preconditions

Payments must exist.

Postconditions

The requested information is shown.

Main Flow

- The user enters the "Payments" module.
- The system displays all payments.
- The user selects one payment.
- The system displays: ID, Invoice, Amount, Date, Method.

4.2.4 Use Case: Update Payment

Actor

Admininistrador

Description

Modifies an existing payment.

Preconditions

- The payment must exist.
- The user must have permission to edit payments.

Postconditions

The updated payment data is stored.

Main Flow

- The user selects a payment to edit.
- The system shows the current payment details.
- The user updates fields such as Amount, Date, or Method.
- The user submits the changes.
- The system validates the data.
- The system updates the payment.
- A success message is shown.

4.2.5 Use Case: Delete Payment

Actor Administrador

Description

Deletes a payment from the system.

Preconditions

- The payment must exist.
- It must not violate accounting/business rules.

Postconditions

The payment is removed from the database.

Main Flow

- The user selects a payment to delete.
- The system requests confirmation.
- The user confirms.
- The system deletes the payment.

The system displays a confirmation message.

4.2.6 Use Case: Create Mechanic

Actor

Administrador

Description

Creates a new mechanic record in the system.

Preconditions

User must have permission to manage mechanics.

Postconditions

The mechanic is saved in the system.

Mechanic becomes available for assignment to work orders.

Main Flow

- The user opens the “Create Mechanic” form.
- The system displays required fields.
- The user enters First Name, Last Name, Specialty, and Status.
- The user submits the form.

The system validates:

Fields are not empty.

Status is valid (Active/Inactive).

The system creates the mechanic.

A success confirmation is shown.

4.2 Use Case: Read Mechanic

Actor

Administrador

Description

Displays mechanic information or lists all mechanics.

Preconditions

At least one mechanic must exist.

Postconditions

Mechanic data is successfully displayed.

Main Flow

- The user enters the “Mechanics” module.
- The system lists all mechanics with basic fields.
- The user selects a mechanic.
- The system displays: ID, First Name, Last Name, Specialty, Status.

4.2.7 Use Case: Update Mechanic

Actor

Administrador

Description

Updates the attributes of an existing mechanic.

Preconditions

The mechanic must exist.

User must have permission to update mechanic records.

Postconditions

The mechanic’s information is updated in the system.

Main Flow

- The user selects a mechanic to edit.
- The system displays the mechanic’s current information.

- The user modifies any field: First Name, Last Name, Specialty, or Status.
- The user submits the changes.
- The system validates the updated data.
- The system saves the updated mechanic.
- A confirmation message is shown.

4.2.8 Use Case: Delete Mechanic

Actor

Administrador

Description

Deletes a mechanic from the system.

Preconditions

The mechanic must exist.

The mechanic must not be assigned to ongoing work orders.

Postconditions

The mechanic is removed from the database.

Main Flow

- The user selects a mechanic to delete.
- The system checks if the mechanic is currently assigned to active work orders.
- The system requests confirmation.
- The user confirms.
- The mechanic is deleted.

A confirmation message is displayed.

4.2.9 Use Case: Create WorkOrderService

Actor

Administrador

Description

Adds a specific service to an existing work order, associating a mechanic and price.

Preconditions

- The work order must exist.
- The service must exist.
- The mechanic must exist.
- User must have permission to modify work orders.

Postconditions

The service is added to the work order.

The service cost contributes to the total billing of the work order.

The mechanic becomes responsible for this service entry.

Main Flow

- The user opens the “Add Service to Work Order” form.
- The system displays fields for Work Order, Service, Price, Mechanic, and Vehicle.
- The user selects the Work Order and the Service.
- The user enters or confirms the Price.
- The user selects the Mechanic.
- The user submits the form.
- The system validates:
- Work order exists.
- Service exists.
- Mechanic exists.
- Price is a valid numeric value.
- The system creates the WorkOrderService entry.
- A confirmation message is shown.

4.3.1 Use Case: Read WorkOrderService

Actor

Administrador

Description

Displays the list of services associated with a work order or retrieves a single service entry.

Preconditions

At least one WorkOrderService must exist.

Postconditions

The system displays the requested information.

Main Flow

- The user opens the “Work Order Services” module.
- The system lists all WorkOrderService entries.
- The user selects one.
- The system displays:

ID

Work Order

Service

Price

Mechanic

Vehicle

4.3.2 Use Case: Update WorkOrderService

Actor

Administrador

Description

Modifies the service details assigned to a work order.

Preconditions

The WorkOrderService entry must exist.

The user must have permission to update work order services.

Postconditions

The service entry is updated.

- **Main Flow**
- The user selects a WorkOrderService entry to edit.
- The system displays its current information.
- The user updates:
- Service
- Price
- Mechanic
- The user submits the changes.
- The system validates all fields.
- The system updates the WorkOrderService entry.

A success message is shown.

4..3.3 Use Case: Delete WorkOrderService

Actor

Admininistrador

Description

Removes a service from a work order.

Preconditions

The WorkOrderService entry must exist.

It must not create inconsistencies (e.g., invoice already generated).

Postconditions

The service is removed from the work order.

Main Flow

- The user selects a WorkOrderService to delete.
- The system verifies that it can be safely removed.
- The system requests confirmation.
- The user confirms.
- The WorkOrderService entry is deleted.

The system shows a confirmation message.

4.3..4 Use Case: Create WorkOrderSparepart

Actor

Administrador

Description

Adds a spare part to an existing work order, specifying the quantity required.

Preconditions

The work order must exist.

The spare part must exist.

The user must have permission to modify work orders.

Postconditions

The spare part is linked to the work order.

The quantity affects the total cost of the work order during billing.

Main Flow

The user opens the “Add Spare Part to Work Order” form.

The system shows fields for Work Order, Spare Part, and Quantity.

The user selects the Work Order.

The user selects the Spare Part.

The user enters the Quantity.

The user submits the form.

The system validates:

Work order exists.

Spare part exists.

Quantity is a valid positive integer.

The system creates the WorkOrderSparepart entry.

The system displays a confirmation message.

4.3.5 Use Case: Read WorkOrderSparepart

Actor

Administrador

Description

Displays the list of spare parts assigned to work orders or the details of a specific entry.

Preconditions

At least one WorkOrderSparepart record must exist.

Postconditions

The system displays the requested information.

Main Flow

- The user enters the “Work Order Spare Parts” module.
- The system lists all WorkOrderSparepart records.
- The user selects a record.
- The system displays:
- ID
- Work Order
- Spare Part
- Quantity

4.3.6 Use Case: Update WorkOrderSparepart

Actor

Administrador

Description

Modifies a spare part entry within a work order.

Preconditions

The WorkOrderSparepart record must exist.

User must have permission to update work order components.

Postconditions

The record is updated with the new information.

Main Flow

The user selects a WorkOrderSparepart record to edit.

The system shows its current details.

The user updates the Spare Part or Quantity.

The user submits the changes.

The system validates:

Quantity is a positive integer.

Spare part exists.

The system updates the entry.

A success message is shown.

4.3.7Use Case: Delete WorkOrderSparepart

Actor

Administrador

Description

Removes a spare part record from a work order.

Preconditions

The record must exist.

The work order must not be invoiced yet (to avoid billing inconsistencies).

Postconditions

The spare part is removed from the work order.

Main Flow

The user selects a WorkOrderSparepart record to delete.

The system checks for restrictions (e.g., linked invoice).

The system requests confirmation.

The user confirms.

The record is deleted.

A confirmation message is displayed.

---

5. Backend Documentation

5.1 Backend Architecture

The backend is built with Django and the Django REST Framework (DRF).

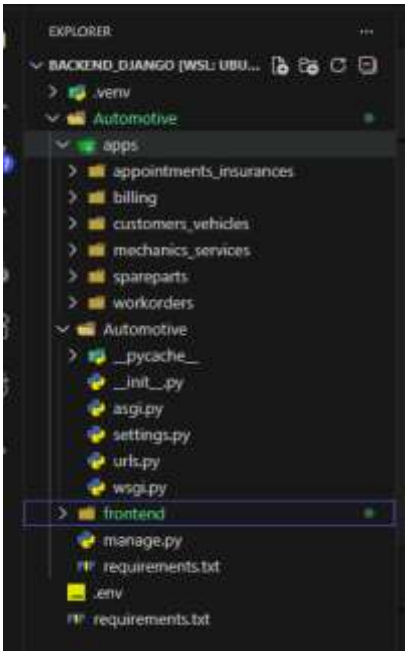
It follows a pattern similar to MVC, where:

- Models represent the database structure.
- Serializers transform the data from the models into JSON.
- ViewSets automatically handle CRUD operations.
- Routers expose endpoints in the /api/ namespace.

The backend communicates with a MySQL database and other engines, and exposes REST endpoints used by the Angular frontend.

5.2 Folder Structure

(Insert backend folder structure)



5.3 API Documentation (REST)

Method Path:

- **POST /api/appointments_insurances/appointments/**
  Create a new appointment.

  GET /api/appointments_insurances/appointments/
  Retrieve all appointments.

  GET /api/appointments_insurances/appointments/{id}/
  Retrieve a single appointment.

  PUT /api/appointments_insurances/appointments/{id}/
  Update an appointment.

  DELETE /api/appointments_insurances/appointments/{id}/
  Delete an appointment.

Purpose:

Creates a new appointment in the system.

Request Body Example:

```
{
"date": "2025-01-20",
  "reason": "Oil change",
  "customer": 1,
  "vehicle": 3,
  "mechanic": 2
}
```

Responses:
POST /api/appointments_insurances/appointments/

Responses:

- **201 Created** – Appointment successfully created
- **400 Bad Request** – Invalid or missing fields

GET /api/appointments_insurances/appointments/

Responses:

- **200 OK** – List of appointments returned successfully

GET /api/appointments_insurances/appointments/{id}/

Responses:

- **200 OK** – Appointment details retrieved

  404 Not Found – Appointment does not exist

PUT /api/appointments_insurances/appointments/{id}/

Responses:

- **200 OK** – Appointment successfully updated

  400 Bad Request – Invalid fields

  404 Not Found – Appointment not found

DELETE /api/appointments_insurances/appointments/{id}/

Responses:

- **204 No Content** – Appointment deleted

  404 Not Found – Appointment not found

5.4 REST Client

This project includes a REST client for testing API endpoints.
The client can be used through:

- **VS Code REST Client extension**,

  Postman, or

  Thunder Client.

Example REST Client Request

Create Appointment

POST http://localhost:8000/api/appointments_insurances/appointments/ Content-Type: application/json

{ “date”: “2025-01-20”,

“reason”: “Oil change”,

“customer”: 1,

“vehicle”: 3,

“mechanic”: 2 }

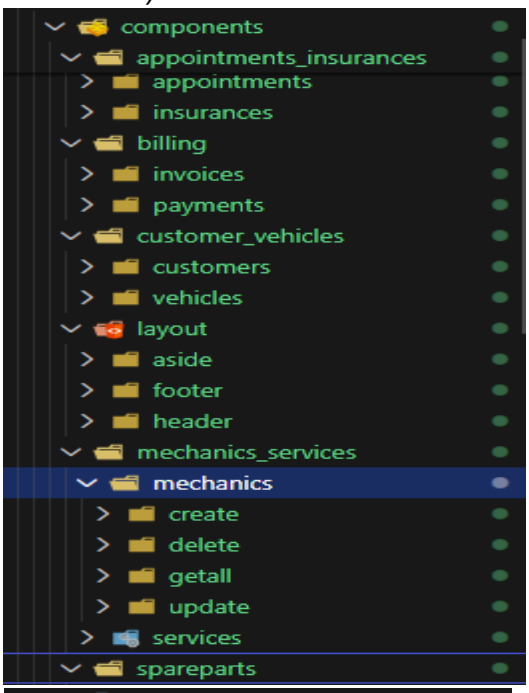
---

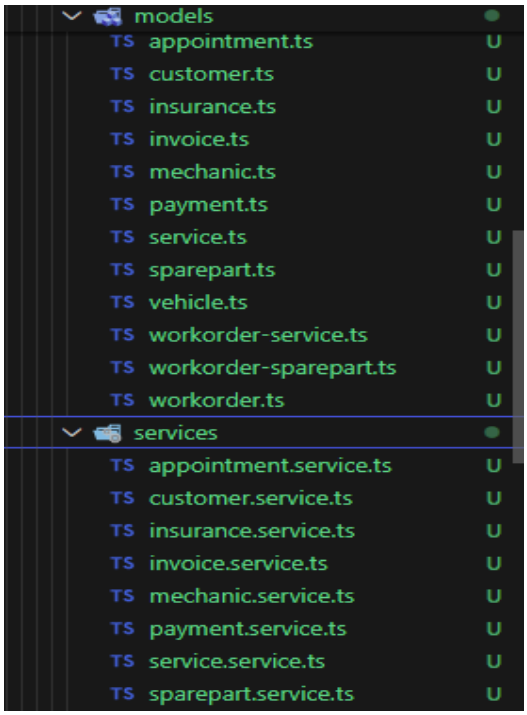
6. Frontend Documentation

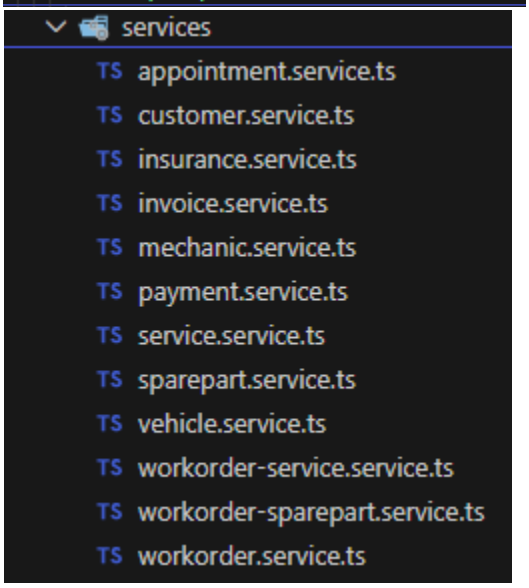
6.1 Technical Frontend Documentation

Framework Used:

Angular 20.3 (Standlone Componentts)
Folder Structure:
(Insert frontend folder structure)





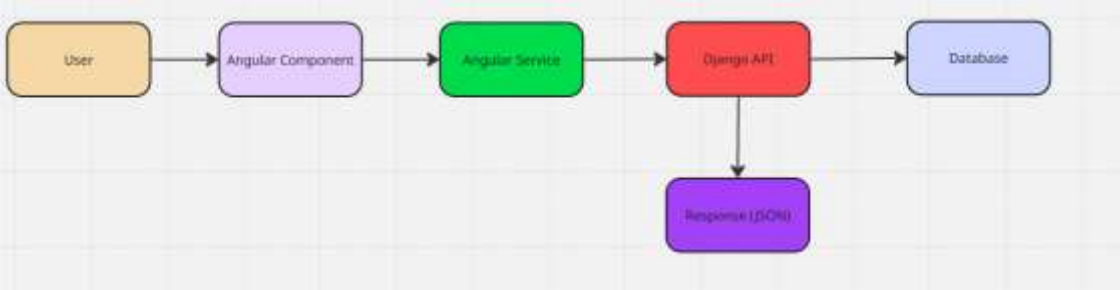


Models, services and Components

6.2 Visual explanation of the system's operation

The frontend communicates with the backend through:

1. **User interacts with UI components**
2. **Angular service sends HTTP request to Django REST API**
3. **Backend processes request and returns JSON**
4. **Frontend updates UI with the response**



---

7. Frontend–Backend Integration

Integration is done through REST API calls using Angular's HttpClient.

Each module has its own service:

- appointments.service.ts
- customers.service.ts
- vehicles.service.ts
- mechanics.service.ts
- services.service.ts
- invoices.service.ts
- payments.service.ts
- workorders.service.ts
- workordersparepart.service.ts
- workorderservice.service.ts

Each service contains methods such as:

- getAll()
- getById(id)
- create(data)
- update(id, data)
- delete(id)

All endpoints follow the pattern:

http://localhost:8000/api/module/enty

---

8. Conclusions & Recommendations

Conclusions

The system successfully integrates a complete automotive workshop workflow.

The backend (Django + DRF) provides structured, secure, and scalable REST APIs.

The frontend (Angular + PrimeNG) delivers an interactive and modular UI.

The database supports MySQL, SQL Server, Oracle, and PostgreSQL, ensuring compatibility with different environments.

Recommendations

Implement authentication (JWT) for production.

Add role-based access control (Admin, Mechanic, Customer).

Improve logging & monitoring for debugging.

Add unit tests for API endpoints and Angular components.

Deploy using Docker for easier environment control.

---

9. Annexes (Optional)