Project Report

Arun Venkatesh - 862247205

Ponmanikandan Velmurugan - 862206551

Professor. Petko Bakalov

TA : Muhammad Shihab Rashid

University of California, Riverside

CS 166 Summer 2021

08/26/2021

Overview & The Design:

The overall objective of the project is to ensure that all DBMS concepts taught in CS166 are practically implemented through hands-on experience.

It is achieved by implementing a management system for a mechanic shop (A-Team's Mechanic Shop). The model follows a client-server architecture as shown in Figure 1 below.

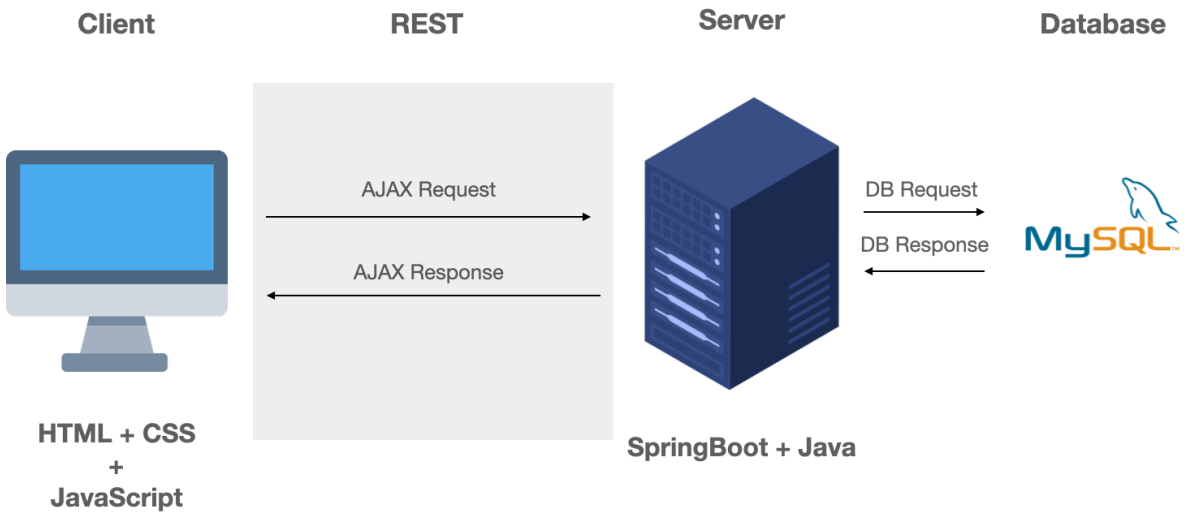


Figure 1: Architecture of A-Team's Mechanic Shop

Client : The client side is written in **HTML+CSS** along with **Javascript** to render the necessary DOM elements on the client.
The client also makes use of **AJAX**(Asynchronous Javascript and XML) to communicate with the server. The client is capsuled and run as a **Python** server that locally to ensure to-and-fro communications between it and the server

REST : In order to enforce standards and constraints, the whole interaction between the client and the server is facilitated through the state of the art architectural style **REST**. Detailed information regarding the same can be found in the upcoming sections.

Server : The server side is written completely in **Java** backed by **SpringBoot** framework. SpringBoot is used for the reason of implementing APIs quite easily, given the time frame to complete this project. The structure of the server code can be found in the upcoming sections

MySQL : All the data that is made use of is permanently stored in the **MySQL** database. This is the core of the project that ensures every concept is strived to be implemented.

ER Diagram:

Representation

The following is the key to understand the ER Diagram:

Color/ Representation	Meaning
	Entity
	Attribute
	Relationship
	Foreign Key Attribute
Underlined	Primary Key Attribute

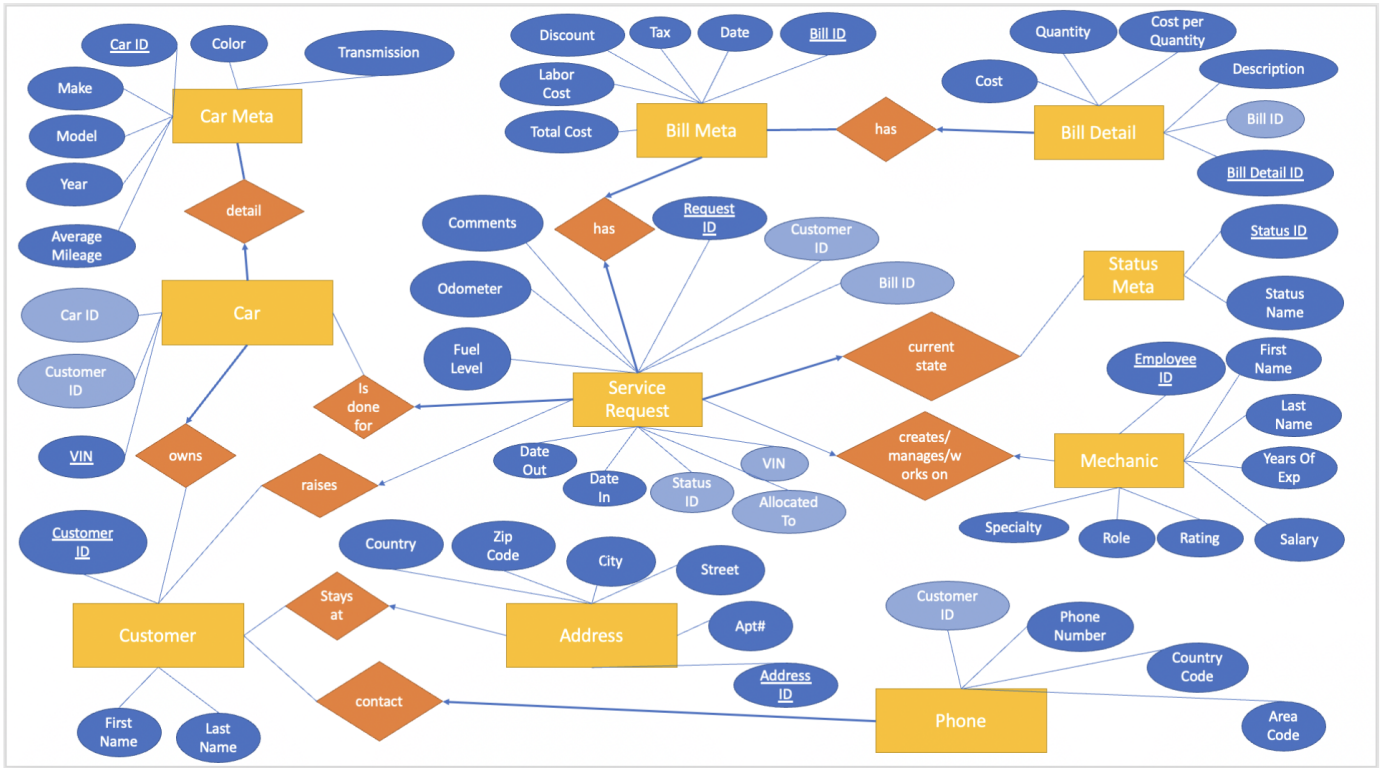


Table Additions/ Extensions:

1. **Car Meta**
This table is introduced in order to group the same cars as a single row in the relational database. Many customers can have a similar kind of car. For example, Honda Civic – Automatic & Grey color can be owned by many customers. A separate CarId is allocated for every car (with varying colors) in order to identify a car uniquely.

2. **Mechanic**
This table is introduced for storing the information of a mechanic. Additional attributes such as Salary, Rating and Role are introduced in order to provide a clean user interface and additional functionalities such as to allow customers to rate the mechanic based on the service's outcome.

3. **Service Request**
This table is primarily used to contain the basic information needed for a service request to be created. Additional parameters like DateOut and Fuel Level are introduced to provide a better bill generation for a customer to understand and to track the time of each service.

4. **Bill Meta**
This table is primarily used to store the basic structure of a bill. This contains basic information such as the bill ID, the date in which the bill is generated, the tax and discounts that are applicable, the labor and the final cost that must be paid by the customer.

5. **Bill Detail (In-progress)**
This table is a subset of the Bill Meta table. This is introduced to track, support and display multiple changes that are made in a particular service (A service can have multiple changes like oil change, coolant replacement, etc...). The costs per quantity and the total cost associated with these are also tracked in this table.

Assumptions Made:

1. A customer will have one address and one phone number
2. A service request will take a day or more than a day to complete
3. The statuses of a request are "Not allocated/Open" and "Completed/ Closed"
4. Comments would be added for a particular service and not for individual changes made in the service.
5. A service request can have only one status at any instant of time.
6. A customer, car and a service request once created cannot be deleted (They have been linked as foreign keys and according to the problem statement, service requests should not be deleted)
7. The bill detail implementation is in development stages and is currently merged along with the bill table implementation

Source Code Design

Server:

Folder	File Name	Purpose
root	Cs166BackendApplication.java	Initialization class that sets up necessary connections
	MechanicShop.java	Interacts with MySQL and returns response
api	CustomerAction.java	API lander class for all customer based actions
	CarMetaAction.java	API lander class for all Car Meta based actions
	CarAction.java	API lander class for all Car based actions
	MechanicAction.java	API lander class for all Mechanic based actions
	ServiceRequestAction.java	API lander class for all Service Request based actions
	BillAction.java	API lander class for all Bill based actions
	BillDetailAction.java	API lander class for all Bill Detail based actions
	FilterAction.java	API lander class for all filter based actions (Hardcoded for the scope of this project)
database	DBUtil.java	Layer between the API landers and the MechanicShop class (Format the input to the mechanic shop and structures the response back to client)
pojo	Customer.java	Object oriented implementation of Customer table
	CarMeta.java	Object oriented implementation of Car Meta table
	Car.java	Object oriented implementation of

		Car table
	Mechanic.java	Object oriented implementation of Mechanic table
	ServiceRequest.java	Object oriented implementation of Service Request table
	Bill.java	Object oriented implementation of Bill table
	BillDetail.java	Object oriented implementation of Bill Detail table
utils	OperationType.java	Enum class that defines the current type of operation (Ex : GET, POST, GET_ALL, PUT, DELETE)

Client:

Folder	**File Name**	**Purpose**
root	Index.html	Shared layout that setups the CSS and Scripts
	Team_Information.html	Page displaying the Project and Team Information
Master	Customer_Details	Displays all the Customer records in the database
	Customer_Profile	Enables the user to add and update existing customer records
	Mechanic_Details	Displays all the Mechanic records in the database
	Mechanic_Profile	Enables the user to add and update existing mechanic records
	Car_Details	Displays all the Car records from the joined tables Car and Car Meta
	Car_Profile	Enables the user to add and update existing car and car meta records
	Pending_Service_Details	Displays all the Pending Service Records from the database

	Pending_Service_Profile	Enables the user to add and update existing open service records
	Closed_Service_Details	Displays all the Closed Service Records from the database
	Closed_Service_Profile	Enables the user to view closed service records
Filters	Filter_Six	Displays the result of Query Six
	Filter_Seven	Displays the result of Query Seven
	Filter_Eight	Displays the result of Query Eight
	Filter_Nine	Displays the result of Query Nine with user input for k
	Filter_Ten	Displays the result of Query Ten

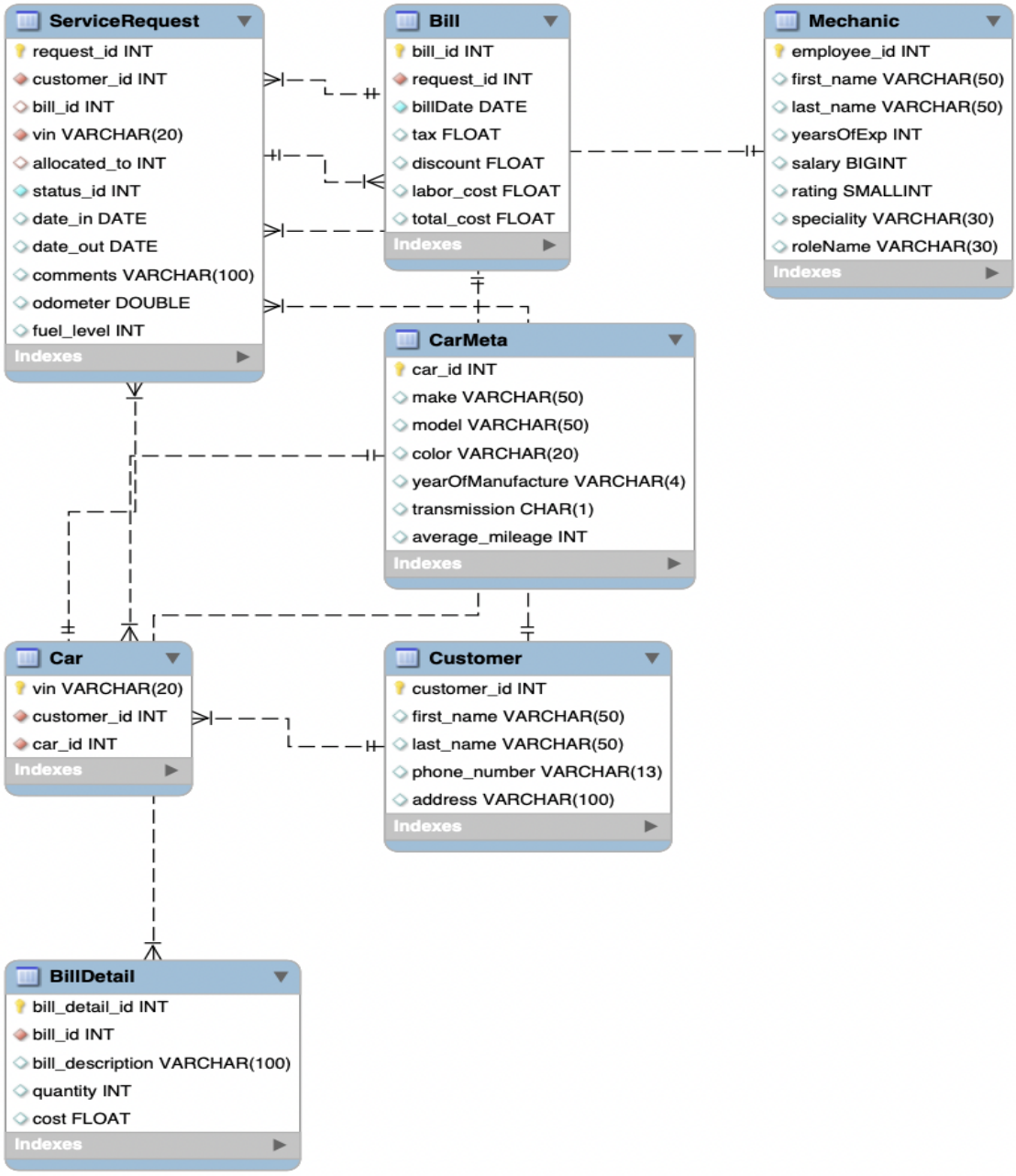
API List: (prefer /api/)

Type	**URL**	**HTTP Method**	**Purpose**
Customer	/customer	GET	Get customer given customer ID
		POST	Create a customer
		PUT	Update a customer
		DELETE	Delete a customer given customer ID
	/customer/all	GET	Get all customers
Car Meta	/car/meta	GET	Get Car ID given car Meta
		POST	Create a Car Metadata
	/car/meta/id	GET	Get Car Meta given Car ID
	/car/meta/all	GET	Get all Car Metadata
Car	/car	GET	Get car details given VIN

		POST	Create a new Car
		PUT	Update a Car
		DELETE	Delete a car given VIN
	/car/customer	GET	Get all cars for a customer given customer ID
	/car/all	GET	Get all cars of all customers
Mechanic	/mechanic	GET	Get mechanic details given employee ID
		POST	Create a new mechanic
		PUT	Update a Mechanic details
		DELETE	Delete mechanic details given employee ID
	/mechanic/all	GET	Get all mechanic details
Service Request	/servicerequest	GET	Get service request details given request ID
		POST	Create a service request
		PUT	Update a service request
	/servicerequest/close	PUT	Close a service request
	/servicerequest/all	GET	Get all service requests
Bill	/bill	GET	Get bill details given bill ID
		POST	Create a bill
		PUT	Update a bill

		DELETE	Delete a bill
Bill Detail	/billdetail	POST	Create a bill detail given a bill ID
		PUT	Update a bill detail
		DELETE	Delete a bill detail
	/billdetail/all/billid	GET	Get all bill details for a bill ID
Filters	/filter/six	GET	Get response for Query 6
	/filter/seven	GET	Get response for Query 7
	/filter/eight	GET	Get response for Query 8
	/filter/nine	GET	Get response for Query 9
	/filter/ten	GET	Get response for Query 10

Database Design:



SQL Queries (As asked in the project description)

Add Customer

INSERT INTO Customer(first_name, last_name, phone_number, address) VALUES ('%s','%s','%s','%s');

Add Mechanic

INSERT INTO Mechanic(first_name, last_name, yearsOfExp, salary, rating, speciality, roleName) VALUES ('%s','%s', %d, %d, %d, '%s', '%s');

Add Car

INSERT INTO Car(vin, customer_id, car_id) VALUES ('%s','%d', %d);

Add Service Request

INSERT INTO ServiceRequest(customer_id, vin, allocated_to, status_id, date_in, comments, odometer, fuel_level) VALUES ('%d', '%s', '%d', '%d', '%s', '%s', '%.2f', '%d') ;

Close Service Request - status_id = 2 and /api/servicerequest (HTTP PUT)

UPDATE ServiceRequest SET allocated_to='%d', status_id='%d', date_in = '%s', comments = '%s', odometer = '%.2f', fuel_level = '%d' WHERE request_id= '%d' ;"

List date, comment, and bill for all closed requests with bill lower than 100

SELECT date_in, date_out, comments, Bill.total_cost FROM ServiceRequest, Bill WHERE ServiceRequest.request_id = Bill.request_id AND total_cost < 100 AND status_id = 0;

List first and last name of customers having more than 20 different cars

SELECT first_name, last_name FROM Customer C1 WHERE 20 < (SELECT count(*) FROM Car CA2 WHERE CA2.customer_id = C1.customer_id);

List Make, Model, and Year of all cars built before 1995 having less than 50000 miles

SELECT CM.make, CM.model, CM.yearOfManufacture FROM CarMeta CM, Car C, ServiceRequest SR WHERE CM.car_id = C.car_id AND C.vin = SR.vin AND SR.odometer < 50000 AND CM.yearOfManufacture < 1995;

List the make, model and number of service requests for the first k cars with the highest number of service orders (k=3)

SELECT CM.make, CM.model, count(*) as total FROM CarMeta CM, Car C1, ServiceRequest SR where CM.car_id=C1.car_id AND C1.vin=SR.vin GROUP BY SR.vin ORDER BY total desc limit 3;

List the first name, last name and total bill of customers in descending order of their total bill for all cars brought to the mechanic

```
SELECT C1.first_name, C1.last_name, SUM(B1.total_cost) AS total_cost FROM Customer C1, Bill B1,
ServiceRequest SR WHERE C1.customer_id = SR.customer_id AND SR.request_id = B1.request_id GROUP BY
C1.customer_id ORDER BY total_cost DESC;
```

Since all these are exposed in an UI backed with REST APIs, we always fetch and display the latest data as a drop down list, thereby avoiding unnecessary data to be accessed, created, updated or deleted.

For other constraints handling, in the client, we make sure of the following:
1. We make field(s) read-only, especially primary keys and other field(s) which should not be updated(ex: Foriegn key constraint)
2. We also have enforced field level restrictions through the input tag of HTML

As far as the server constraints handling is concerned, we make sure of the following:
1. Calls to the database are only allowed when there are no issues in the input parameters that are obtained from the API, thereby avoiding unnecessary write database operations.
2. The database design is structured using a combination of foreign keys along with additional constraints such as cascading to preserve database integrity.

Work Split-up:

The client, server and the database design were implemented in parallel, equally. The effort was also equally split between the team so as to ensure both the team members had the knowledge of both the repositories.

Testing Instructions

Server:
1. Clone the repository from Git
2. Import it in eclipse using Git import or import it as a Maven project (.pom)
3. Run it as a maven configuration
4. Server will be up on the port 8080
5. Accessed through http://localhost:8080/

Client:
1. Clone the repository from Git
2. Open terminal and go to that folder where the repository was cloned
3. Run the following command:
   python -m SimpleHTTPServer 8000 or python3 -m http.server
4. Accessed through http://localhost:8000/

MySQL:

1. Create a database called as ‘cs166db’
2. Import the .sql files present inside SQL Dump folder in MySQLWorkbench (or similar softwares) mapped to the above database
3. Create a user called root mapped to the above database
4. Set the password as ‘Test#135’ for the above root user