BMW Gallery Shop Management System

CPSC 471

Fall 2020

Project Final Report

Suhaib Tariq

Minnie Thai

Abstract

This shop management system allows for users to manage employees, customers, repair orders, vehicles, as well as inventory management for film sizes used to complete orders. The design of the product was completed through an Entity Relationship Diagram, a Relational Model, and an Object-Oriented Model as well. These diagrams have been converted to an SQL database in a MySQL server and connected to a Django REST framework API. This report has documented all the steps we have made to complete this project from design to API implementation.

Introduction

For this CPSC 471 Project, our group has taken a real world problem proposed to us to find a solution for. As mentioned in our proposal, the customer we have created this shop management system for is a local car service shop in Calgary, Alberta called the BMW Gallery. It is a sub department of the service department in a dealership and it requires an updated system to track work orders. Our customer specifically works on 3M wraps for cars of all different models and makes to prevent them from rock chip and dent damage. Our customer had no way of tracking their work orders other than traditional forms organized in filing cabinets. To modernize their shop, they would like digitized work orders, as well as the additional function of being able to track how much markup for the film they are to use.

In this project, we achieved the goal of solving the problem of being unable to track work orders and inventory for the film the customer uses for car wrapping jobs. We have implemented these features:

- An SQL database to store all information used and kept by the application
- Differentiated levels of login (details provided in the guide)
  - Administrators can set permissions for managers and employees and each user group has their own set of permissible actions
- Database connection to an API with connected endpoints
  - Connected endpoints allows user to check existing instances for each entity as well as complex queries for specific entities in the database
  - These complex queries are stored in SQL as stored procedures to be executed when needed

We have solved their main two problems by providing them a system to track work orders, as well as being able to check the amount of markup by film type depending on what they need to get a job done.

Implementation

The Django REST framework API was chosen alongside the MySQL database as we wanted to provide a modern system that can be supported for years to come. Please note that our group lost a member about two-thirds into the semester, just before the API implementation of the project. Due to this, we have adjusted our actual implementation with reduced requirements as we did not have the resources or time to go full-scale with our project.

Project Design

Transaction Endpoints

In Django, these are all implemented in the file "**views.py**" as a view. The function depends on the call from Postman and will connect to the database to retrieve what the user is requesting. Here is the link to our postman documentation for all the endpoints:
https://documenter.getpostman.com/view/13837720/TVmV7Zst#af33aece-0f35-d752-82cf-da237ea45177
This documentation also provides all the examples as well.

- **getAllCustomers   GET**
  - Retrieves full list of customers

```
[
    {
        "CustID": "001",
        "CustName": "Customer1",
        "CustPhone": "22222",
        "CustAddress": "address update example"
    },
    {
        "CustID": "002",
        "CustName": "Customer2",
        "CustPhone": "2222222222",
        "CustAddress": "U of c (also)"
    },
    {
        "CustID": "003",
        "CustName": "APITest",
        "CustPhone": "1234569999",
        "CustAddress": "Help me"
    },
    {
        "CustID": "004",
        "CustName": "Sample Customer",
        "CustPhone": "0001112222",
        "CustAddress": "Calgary"
    },
    {
        "CustID": "006",
        "CustName": "Customer6",
        "CustPhone": "22222",
        "CustAddress": "U of C"
    }
]
```

- **insertNewCustomer   POST**
  - Requires user to input these fields:
    - CustID
    - CustName
    - CustPhone
    - CustAddress
  - In this format:

```
{
    "CustID": "001",
    "CustName": "Customer1",
    "CustPhone": "22222",
    "CustAddress": "address update example"
}
```

- **getCustomerByCustID   GET**
  - Retrieves specific detail of a customer by CustID
  - CustID has to be inputted as a an existing 3 digit CustomerID otherwise it will not work

```json
{
    "CustID": "001",
    "CustName": "Customer1",
    "CustPhone": "22222",
    "CustAddress": "address update example"
}
```

- **deleteCustomerByCustID   DEL**
  - Delete a specific customer by CustID
  - In an instance where CustID is a foreign key, it will be return a NULL value

- **updateCustomerbyCustID   PUT**
  - Updates the details for a specific customer by CustID
  - Input: similar to adding a new customer, all fields must be there but it will update the what is changed except for the primary key that is CustID
  - User is not allowed to change CustID

```json
{
    "CustID": "001",
    "CustName": "Customer1",
    "CustPhone": "22222",
    "CustAddress": "address update example"
}
```

- **getAllCars   GET**
  - Retrieves full list of cars

```json
[
    {
        "VNNo": "00001",
        "CarMaker": "Honda",
        "CarModel": "Civic EX",
        "Year": "2015",
        "CustID": "001"
    },
    {
        "VNNo": "00002",
        "CarMaker": "Toyota",
        "CarModel": "Corolla",
        "Year": "2002",
        "CustID": "002"
    },
    {
        "VNNo": "00003",
        "CarMaker": "Tesla",
        "CarModel": "Model X",
        "Year": "2020",
        "CustID": "001"
    }
]
```

- **insertNewCar**   **POST**
  - Required user to input these fields:
    - VNNO
    - CustID
    - CarMaker
    - CarModel
    - Year

- **getCarByVNNo**   **GET**
  - Retrieves specific detail of a Car by VNNo
  - VNNo has to be inputted as a an existing 5 digit VNNo otherwise it will not work

```json
{
    "VNNo": "00001",
    "CarMaker": "Honda",
    "CarModel": "Civic EX",
    "Year": "2015",
    "CustID": "001"
}
```

- **updateCarByVNNo**   **PUT**
  - Updates the details for a specific car by VNNo
  - Input: similar to adding a new car, all fields must be there but it will update the what is changed except for the primary key that is VNNo
  - User is not allowed to change VNNo

- **deleteCarByVNNo**   **DEL**
  - Delete a specific Car by VNNo
  - In any other instance where VNNo is a foreign key, it will be return a NULL value

- **getAllEmployees**   **GET**
  - Retrieves full list of Employees

```json
[
    {
        "EmployeeID": "0001",
        "Name": "Employee1",
        "Phone": "1234567890",
        "Address": "The shop"
    },
    {
        "EmployeeID": "0002",
        "Name": "Employee2",
        "Phone": "1234567899",
        "Address": "the shop x 2"
    }
]
```

- **insertNewEmployee**   **POST**
  - Requires user to input these fields:

- ■ EmployeeID
- ■ Name
- ■ Phone
- ■ Address
- **getEmployeeByID   GET**
  - ○ Retrieves Specific details of Employee by EmployeeID
  - ○ EmployeeID has to be inputted as a an existing 4 digit employeeID otherwise it will not work

```json
{
    "EmployeeID": "0001",
    "Name": "Employee1",
    "Phone": "1234567890",
    "Address": "The shop"
}
```

- **updateEmployeeByID   PUT**
  - ○ Updates the details for a specific employee by employeeID
  - ○ Input: similar to adding a new employee, all fields must be there but it will update the what is changed except for the primary key that is EmployeeID
  - ○ User is not allowed to change EmployeeID

- **deleteEmployeeByID   DEL**
  - ○ Deletes Employee by EmployeeID
  - ○ In any other instance where EmployeeID is a foreign key, it will be return a NULL value

- **getAllFilms   GET**
  - ○ Retrieves full list of 3M Film Types

```json
[
    {
        "ThreeMFilmType": "Sedan Front Only",
        "Size": 150,
        "Markup": 30,
        "QuantityRemain": 80
    },
    {
        "ThreeMFilmType": "Sedan-Full",
        "Size": 150,
        "Markup": 30,
        "QuantityRemain": 1000
    },
    {
        "ThreeMFilmType": "Truck Front Only",
        "Size": 200,
        "Markup": 50,
        "QuantityRemain": 100
    }
]
```

- **insertNewFilm   POST**
  - ○ Requires user to input these fields:
    - ■ 3MFilmType
    - ■ Size
    - ■ Markup

- ■ QuantityRemain

- **getFilmByDetail**   **GET**
  - ○ Retrieves Specific details of Films by FilmType

```
{
    "ThreeMFilmType": "Sedan Front Only",
    "Size": 150,
    "Markup": 30,
    "QuantityRemain": 80
}
```

- **deleteFilmByType**   **DEL**
  - ○ Delete the film by ThreeMFilmType
  - ○ If this ThreeMFilmType is being used as a foreign key in another entity, it will not allow for deletion, users must modify where it is being used and assign a new value to delete

- **updateFilmByType**   **PUT**
  - ○ Update Film with new Details based on ThreeMFilmType
  - ○ Input: all fields must be there but it will update the what is changed except for the primary key that is ThreeMFilmType
  - ○ User is not allowed to change ThreeMFilmType

- **getAllServices**   **GET**
  - ○ Retrieves Full list of services

```
[
    {
        "Service_Type": "Full Body Wrap",
        "Service_Price": 500,
        "ThreeMFilmType": "Sedan-Full"
    },
    {
        "Service_Type": "Sedan Front",
        "Service_Price": 200,
        "ThreeMFilmType": "Sedan Front Only"
    }
]
```

- **postNewService**   **POST**
  - ○ Requires user to input these fields:
    - ■ Service_Type
    - ■ Service_Price
    - ■ 3MFilm_Type

- **getServiceByName**   **GET**
  - ○ Retrieves list of service by Service_Type

```
{
    "Service_Type": "Sedan Front",
    "Service_Price": 200,
    "ThreeMFilmType": "Sedan Front Only"
}
```

- **deleteServiceByName   DEL**
  - Deletes Specific service by ServiceType
  - If this ServiceType is being used as a foreign key in another entity, it will not allow for deletion, users must modify where it is being used and assign a new value to delete

- **updateServiceByName PUT**
  - Updates service by ServiceType by inputting any fields
    - ServiceType
    - ServicePrice
    - 3MFilmType
  - Input: all fields must be there but it will update the what is changed except for the primary key that is ServiceType
  - User is not allowed to change ServiceType

- **getAllROs   GET**
  - Retreives full list of ROs

```json
[
    {
        "RONumber": "00000001",
        "Date": "2020-12-08",
        "HoursWorked": 10,
        "FilmUsed": 40,
        "VNNo": "00001",
        "CustID": "001",
        "EmployeeID": "0001",
        "ServiceType": "Full Body Wrap"
    },
    {
        "RONumber": "00000002",
        "Date": "2020-12-08",
        "HoursWorked": 10,
        "FilmUsed": 200,
        "VNNo": "00001",
        "CustID": "001",
        "EmployeeID": "0002",
        "ServiceType": "Full Body Wrap"
    },
    {
        "RONumber": "00000003",
        "Date": "2020-12-09",
        "HoursWorked": 15,
        "FilmUsed": 250,
        "VNNo": "00001",
        "CustID": "001",
        "EmployeeID": "0002",
        "ServiceType": "Full Body Wrap"
    },
```

- **insertNewRO   POST**
  - Requires user to input these fields:
    - RONumber
    - VNNo
    - CustID
    - EmployeeID
    - Date
    - ServiceType
    - ServicePrice
    - FimUsed
    - Hours

- **getROByNumber**   **GET**
  - Retrieves specific detail by RONo

```
{
    "RONumber": "00000005",
    "Date": "2020-12-09",
    "HoursWorked": 30,
    "FilmUsed": 200,
    "VNNo": "00003",
    "CustID": "001",
    "EmployeeID": "0001",
    "ServiceType": "Full Body Wrap"
}
```

- **deleteROByNumber**   **DEL**
  - Deleted RO by the RONumber
  - User is not allowed to delete ROs with a linked EmployeeID, CustID, and VNNo

- **updateROByNumber**   **PUT**
  - Updates RO, can be done by inputting any of these fields:
    - RONumber
    - VNNo
    - CustID
    - EmployeeID
    - Date
    - ServiceType
    - ServicePrice
    - FilmUsed
    - Hours

- **getRequests**   **GET**
  - Gets all the Requests

```
[
    {
        "RequestID": "001",
        "Roll_size": 300,
        "Quantity": 15,
        "Date": "2020-12-08",
        "EmpID": "0001"
    },
    {
        "RequestID": "002",
        "Roll_size": 300,
        "Quantity": 50,
        "Date": "2020-01-01",
        "EmpID": "0002"
    }
]
```

- **insertNewRequest**   **POST**
  - Requires user to input these fields:
    - RequestID
    - EmployeeID
    - Amount
    - Quantity

- **■** Date
- **getRequestDetailById** **GET**
  - ○ Retrieves specific Request by RequestID

```
{
    "RequestID": "001",
    "Roll_size": 300,
    "Quantity": 15,
    "Date": "2020-12-08",
    "EmpID": "0001"
}
```

- **deleteRequestDetailById** **DEL**
  - ○ Delete specific Request by RequestID

- **updateRequestDetailById** **PUT**
  - ○ Updates Request, can be done by inputting any if these fields:
    - ■ Requires user to input these fields:
      - ● RequestID
      - ● EmployeeID
      - ● Amount
      - ● Quantity
      - ● Date

- **getAllCustomersCars** **GET**
  - ○ Gets all Cars of a Specific Customer
  - ○ Requires input parameter of customer ID

```
[
    {
        "CustID": "001",
        "CustName": "Customer1",
        "VNNo": "00001",
        "CarMaker": "Honda",
        "CarModel": "Civic EX",
        "Year": "2015"
    },
    {
        "CustID": "002",
        "CustName": "Customer2",
        "VNNo": "00002",
        "CarMaker": "Toyota",
        "CarModel": "Corolla",
        "Year": "2002"
    },
    {
        "CustID": "001",
        "CustName": "Customer1",
        "VNNo": "00003",
        "CarMaker": "Tesla",
        "CarModel": "Model X",
        "Year": "2020"
    }
]
```

- **getOrdersbyCust** **GET**
  - ○ Gets all orders and the customers associated with those orders

```
[
    {
        "RONumber": "00000001",
        "VNNo": "00001",
        "CustID": "001",
        "CustName": "Customer1",
        "CustPhone": "22222"
    },
    {
        "RONumber": "00000002",
        "VNNo": "00001",
        "CustID": "001",
        "CustName": "Customer1",
        "CustPhone": "22222"
    },
    {
        "RONumber": "00000003",
        "VNNo": "00001",
        "CustID": "001",
        "CustName": "Customer1",
        "CustPhone": "22222"
    },
    {
        "RONumber": "00000004",
        "VNNo": "00002",
        "CustID": "001",
        "CustName": "Customer1",
        "CustPhone": "22222"
    },
    {
        "RONumber": "00000005",
        "VNNo": "00003",
        "CustID": "001",
        "CustName": "Customer1",
        "CustPhone": "22222"
    }
]
```

- **getMarkups   GET**
  - Gets all the markups by ThreeMFilmType, allows for users to check for how much mistake space is allowed

```
[
    {
        "Service_Type": "Full Body Wrap",
        "Size": 150,
        "Markup": 30
    },
    {
        "Service_Type": "Sedan Front",
        "Size": 150,
        "Markup": 30
    }
]
```

- **getCarsbyCustomer   GET**
  - Gets Cars by Customer ID
  - Requires input of 3 digit Customer ID
  - Example: all cars CustID = 001 owns

```
[
    {
        "inputID": "001",
        "CustName": "Customer1",
        "VNNo": "00001",
        "CarMaker": "Honda",
        "CarModel": "Civic EX",
        "Year": "2015"
    },
    {
        "inputID": "001",
        "CustName": "Customer1",
        "VNNo": "00003",
        "CarMaker": "Tesla",
        "CarModel": "Model X",
        "Year": "2020"
    }
]
```

- **getOrdersByCar** **GET**
  - Gets order made my Car's VNNo
  - Requires input of 8 digit Car VNNo
  - Example: all the work that has been completed on Car with VNNo 00000001

```
[
    {
        "RONumber": "00000001",
        "VNNo": "00001",
        "Date": "2020-12-08",
        "CustID": "001",
        "CarMaker": "Honda",
        "CarModel": "Civic EX",
        "Year": "2015",
        "ServiceType": "Full Body Wrap"
    },
    {
        "RONumber": "00000002",
        "VNNo": "00001",
        "Date": "2020-12-08",
        "CustID": "001",
        "CarMaker": "Honda",
        "CarModel": "Civic EX",
        "Year": "2015",
        "ServiceType": "Full Body Wrap"
    },
    {
        "RONumber": "00000003",
        "VNNo": "00001",
        "Date": "2020-12-09",
        "CustID": "001",
        "CarMaker": "Honda",
        "CarModel": "Civic EX",
        "Year": "2015",
        "ServiceType": "Full Body Wrap"
    }
]
```

- **getMarkupByService** **GET**
  - Gets amount markup of the ServiceType
  - Requires input of ServiceType name
  - Example: A Full Body Wrap requires a markup of 30 to allow for mistakes

```
[
    {
        "inputServiceType": "Full Body Wrap",
        "ThreeMFilmType": "Sedan-Full",
        "Markup": 30,
        "QuantityRemain": 1000
    }
]
```

- **getROByEmployee** **GET**
  - Gets Ro made by employee by checking EmployeeID
  - Requires input of 4 digit employee ID
  - Example: the repair orders EmployeeID = 0001 has worked on

```json
[
    {
        "inputID": "0001",
        "RONumber": "00000001",
        "VNNo": "00001",
        "Date": "2020-12-08",
        "HoursWorked": 10
    },
    {
        "inputID": "0001",
        "RONumber": "00000005",
        "VNNo": "00003",
        "Date": "2020-12-09",
        "HoursWorked": 30
    }
]
```

- **getROByNew**   **GET**
  - Gets the newest ROs by date of Repair Order

```json
[
    {
        "RONumber": "00000001",
        "VNNo": "00001",
        "CustID": "001",
        "EmployeeID": "0001",
        "Date": "2020-12-08",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 10,
        "FilmUsed": 40
    },
    {
        "RONumber": "00000002",
        "VNNo": "00001",
        "CustID": "001",
        "EmployeeID": "0002",
        "Date": "2020-12-08",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 10,
        "FilmUsed": 200
    },
    {
        "RONumber": "00000003",
        "VNNo": "00001",
        "CustID": "001",
        "EmployeeID": "0002",
        "Date": "2020-12-09",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 15,
        "FilmUsed": 250
    },
    {
        "RONumber": "00000004",
        "VNNo": "00002",
        "CustID": "001",
        "EmployeeID": "0002",
        "Date": "2020-12-09",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 8,
        "FilmUsed": 150
    },
    {
        "RONumber": "00000005",
        "VNNo": "00003",
        "CustID": "001",
        "EmployeeID": "0001",
        "Date": "2020-12-09",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 30,
        "FilmUsed": 200
    }
]
```
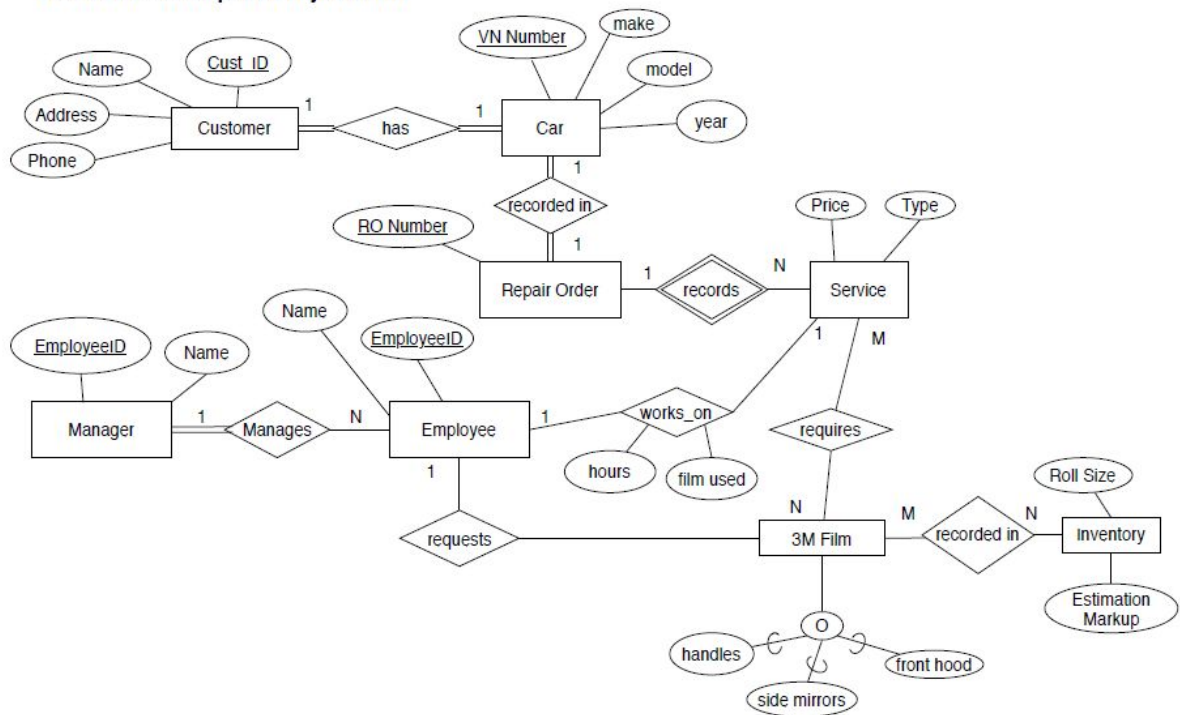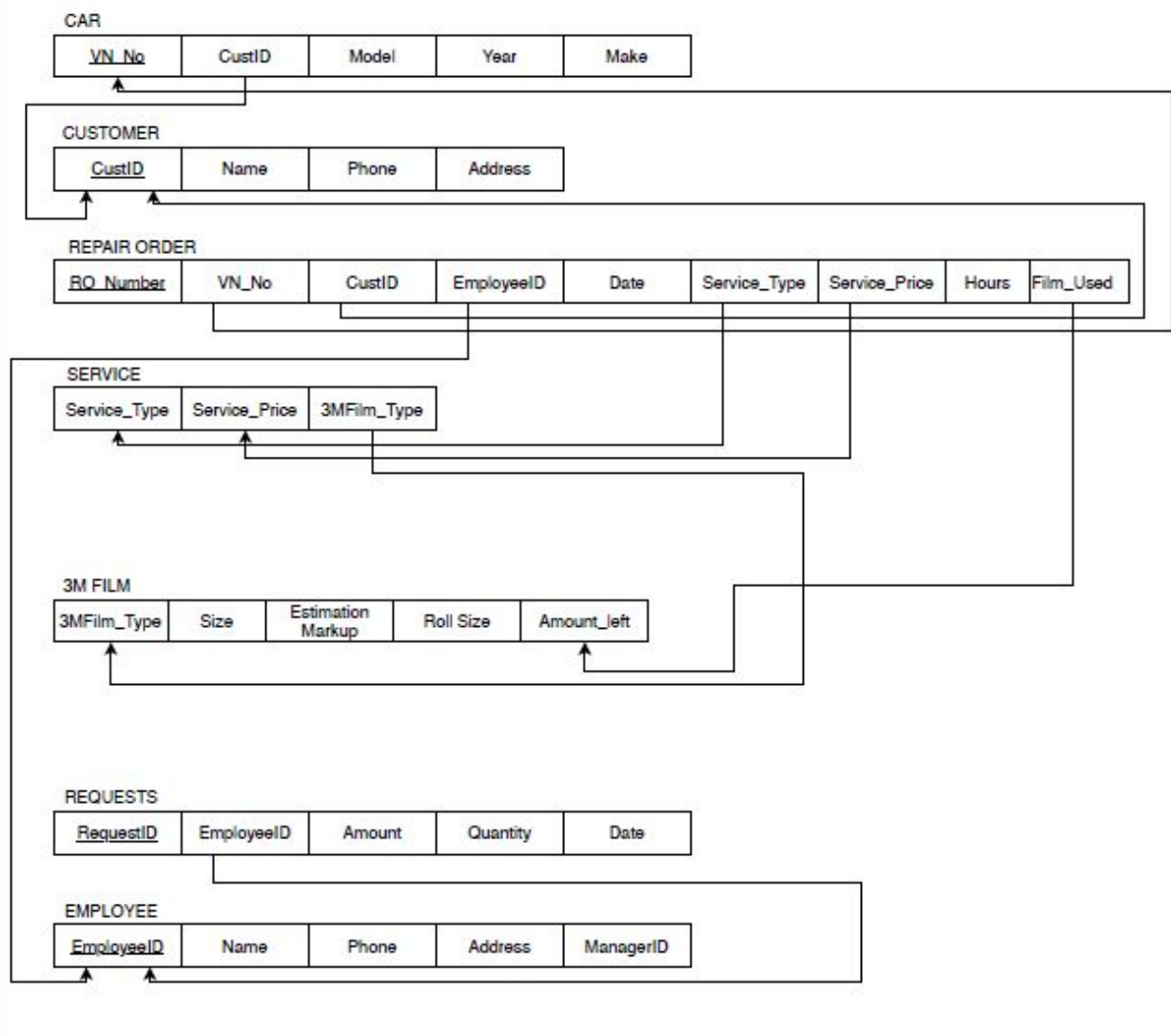
- **getROByYear**   **GET**
  - Gets RO by each year
  - Requires input of 4-digit year
  - Example: all repair orders completed in 2020 sorted by date

```
[
    {
        "RONumber": "00000001",
        "VNNo": "00001",
        "CustID": "001",
        "EmployeeID": "0001",
        "Date": "2020-12-08",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 10,
        "FilmUsed": 40
    },
    {
        "RONumber": "00000002",
        "VNNo": "00001",
        "CustID": "001",
        "EmployeeID": "0002",
        "Date": "2020-12-08",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 10,
        "FilmUsed": 200
    },
    {
        "RONumber": "00000003",
        "VNNo": "00001",
        "CustID": "001",
        "EmployeeID": "0002",
        "Date": "2020-12-09",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 15,
        "FilmUsed": 250
    },

    {
        "RONumber": "00000004",
        "VNNo": "00002",
        "CustID": "001",
        "EmployeeID": "0002",
        "Date": "2020-12-09",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 8,
        "FilmUsed": 150
    },
    {
        "RONumber": "00000005",
        "VNNo": "00003",
        "CustID": "001",
        "EmployeeID": "0001",
        "Date": "2020-12-09",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 30,
        "FilmUsed": 200
    }
}
```

- **getROByDate**   **GET**
  - Gets RO by Specific Date
  - Requires input of date in format: YYYY-MM-DD
  - Example: all repair orders completed on December 9, 2020

```json
[
    {
        "RONumber": "00000003",
        "VNNo": "00001",
        "CustID": "001",
        "EmployeeID": "0002",
        "Date": "2020-12-09",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 15,
        "FilmUsed": 250
    },
    {
        "RONumber": "00000004",
        "VNNo": "00002",
        "CustID": "001",
        "EmployeeID": "0002",
        "Date": "2020-12-09",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 8,
        "FilmUsed": 150
    },
    {
        "RONumber": "00000005",
        "VNNo": "00003",
        "CustID": "001",
        "EmployeeID": "0001",
        "Date": "2020-12-09",
        "ServiceType": "Full Body Wrap",
        "ServicePrice": 0,
        "HoursWorked": 30,
        "FilmUsed": 200
    }
]
```

Entity Relational Diagram

CPSC471 - Group 35 Project ERD

Relational Model

CAR

VN_No	CustID	Model	Year	Make

CUSTOMER

CustID	Name	Phone	Address

REPAIR ORDER

RO_Number	VN_No	CustID	EmployeeID	Date	Service_Type	Service_Price	Hours	Film_Used

SERVICE

Service_Type	Service_Price	3MFilm_Type

3M FILM

3MFilm_Type	Size	Estimation Markup	Roll Size	Amount_left

REQUESTS

RequestID	EmployeeID	Amount	Quantity	Date

EMPLOYEE

EmployeeID	Name	Phone	Address	ManagerID

Adjustments made (in comparison to ERD submitted previously):
- Took away "PARTS" as an entity, it made more sense to have part_type be an attribute that PARTS_DEPARTMENT has
- Added Requests and request ID to keep track of requests sent from Employee to Parts Department for better tracking
- Orders_From has is their own relationship table as there are attributes such as date, quantity, and price associated with these orders
- 3M Film can be sorted by tuple - for type displayed in ERD
- INVENTORY condensed into 3M Film Entity - records estimation markup for 3M Film Type
- There should be a SERVICE table for storing the different types of services and the associated prices for other entities to use

Object Oriented Diagram (Initial version)

CUSTOMER
+ CustID   :Integer
+ Name   :String
+ Phone   :String
+Address   :String

+ Add(a,b,c,d)
+ Update(a, newB, newC, newD)
+ Search(x)
+ Delete(a)
+ ViewAll()

CAR
+ VN_No   :Integer
+ CustID   :String
+ Model   :String
+ Year   :Integer
+ Make   :String

+ ViewAll()
+ Search(x)
+ Add(a,b,c,d,e)
+ Delete(a)
+ Update(a,newB,newC,newD,newE)

EMPLOYEE
+ EmployeeID   :Integer
+ Name   :String
+ Phone   :Integer
+ Address   :String
+ ManagerID   :Integer

+ Add(a,b,c,d,e)
+ Delete(a)
+ Search(x)
+ Update(a, newB, newC, newD, new E)
+ ViewAll()

SERVICE
+ Service_Type   :String
+ Service_Price :Integer
+ 3MFilm_Type   :String

+ ViewAll()
+ Search(x)
+ Delete(a)
+ Add(a, b, c)
+ Update(a, newB, newC)

REPAIR_ORDER
+ RO_Number   :Integer
+ VN_No   :Integer
+ CustID   :Integer
+ EmployeeID   :Integer
+ Date   :Date
+ Service_Type :String
+ Service_Price :Integer

+ Add(a,b,c,d,e,f,g)
+ Update(a, newB, newC, newD, newE, newF, newG)
+ ViewAll()
+ Search(x)
+ Delete(a)

REQUESTS
+ RequestID   :Integer
+ Employee_ID   :Integer
+ Amount   :Integer
+ Quantity   :Integer
+ Date   :Date

+ ViewAll()
+ Search(x)
+ Delete(a)
+ Add(a, b, c, d, e)
+ Update(a, newB, newC, newD, newE)

WORKS_ON
+ EmployeeID   :Integer
+ RO_Number   :Integer
+ Film_Used   :Integer
+ Hours   :Integer

+ ViewAll()
+ Search(x)
+ Delete(a)
+ Add(a, b, c, d)
+ Update(a, newB, newC, newD)

3M FILM
+ 3MFilm_Type   :String
+ Size   :Integer
+ Roll_size   :Integer
+ Estimation_Markup :Integer
+ Amount_left   :Integer

+ ViewAll()
+ Search(x)
+ Delete(a)
+ Update(a, newB, newC, newD, new E)
+ Add(a, b, c, d, e)

Service_Dept
+ SDept_ID   :Integer
+ Roll Size   :Integer
+ Film_Roll_Quantity :Integer

+ ViewAll()
+ Search(x)
+ Delete(a)
+ Update(a, newB, newC)
+ Add(a, b, c)

Object Oriented Diagram (Updated)
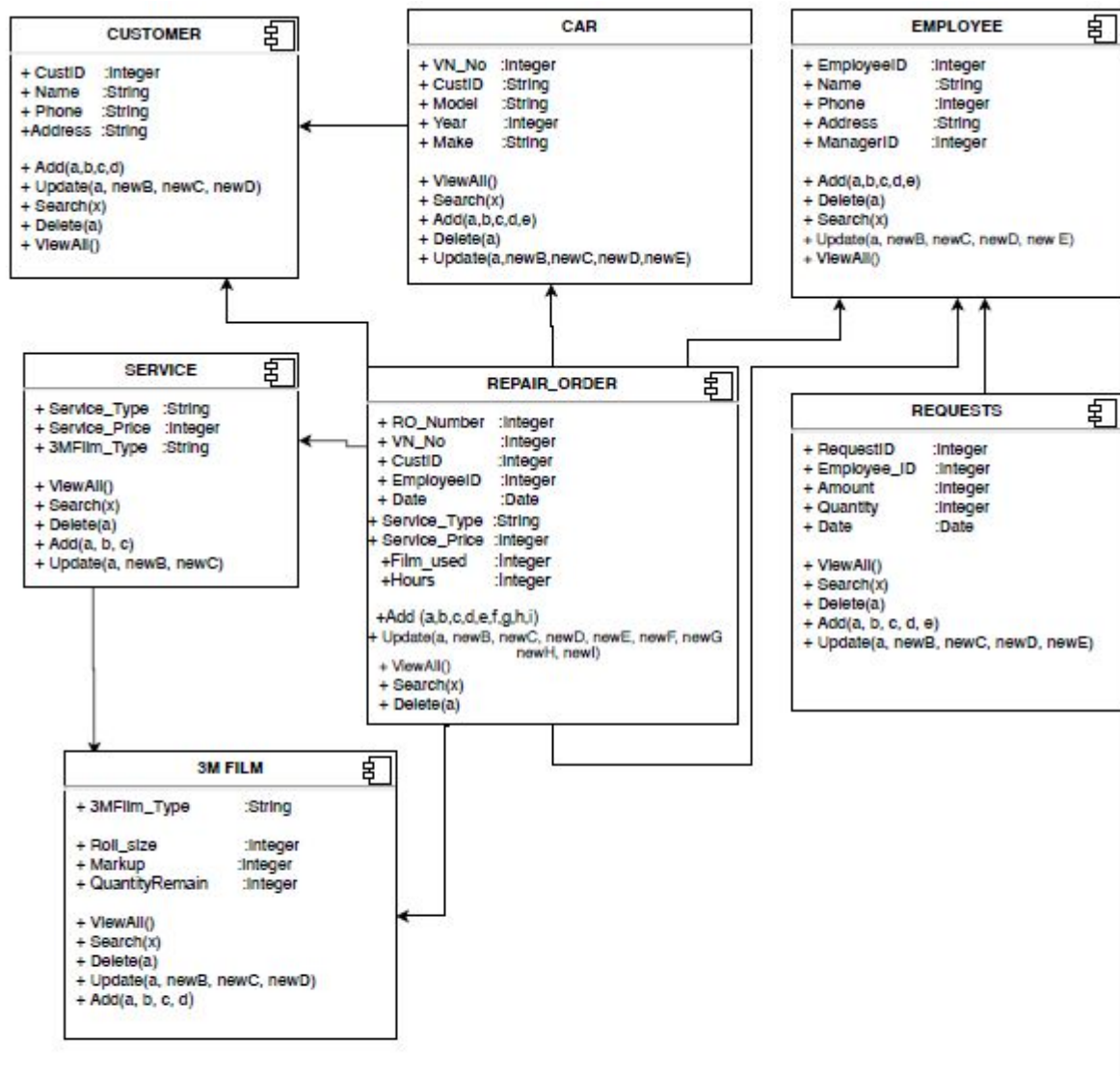
Notes:

- "Works_on" relationship is incorporated into Repair Order now
- Repair Order now handles Film_Used and Hours worked
- Django would not allow a relationship consisting of two foreign keys but no primary key
- This was mentioned in the project presentation as one of the complications we had to overcome working with the Django framework
- "Service_Dept" was also removed as it made no sense implemented in our actual database model
- Service_Dept made more sense when there were multiple departments but as we simplified, it was redundant

Database Specifications

For this project, we have chosen to use the Microsoft SQL Server.

These are the statements we used to create the database. For executable code, please refer to the file named "Project-Create" in our project folder.

Table Creation Functions

```sql
CREATE TABLE customer
        (CustID         CHAR(8)     NOT NULL,
        CustName        VARCHAR(15)     NOT NULL,
        CustPhone       CHAR(10)        NOT NULL,
        CustAddress     VARCHAR(30)     NOT NULL,
        PRIMARY KEY (CustID));

CREATE TABLE car
        (VNNo           CHAR(12)        NOT NULL,
        CustID          CHAR(8)         NOT NULL,
        CarMaker        VARCHAR(15)     NOT NULL,
        CarModel        VARCHAR(15)     NOT NULL,
        Year            CHAR(9)         NOT NULL,
        PRIMARY KEY (VN_No),
        FOREIGN KEY (CustID) REFERENCES customer (CustID) );

CREATE TABLE employee
        (EmployeeID     CHAR(10)            NOT NULL,
        Name            VARCHAR(15)     NOT NULL,
        Phone           CHAR(10)        NOT NULL,
        Address         VARCHAR(30)     NOT NULL,
        PRIMARY KEY (EmployeeID),
        unique (EmployeeID));

CREATE TABLE threemfilm
        (ThreeMFilmType VARCHAR(20)     NOT NULL,
        Size            int             NOT NULL,
        Markup          int             NOT NULL,
        RollSize        int             NOT NULL,
        unique (ThreeMFilmType));
```

```sql
CREATE TABLE service
        (Service_Type    VARCHAR(8)       NOT NULL,
        Service_Price    int              NOT NULL,
        ThreeMFilmType   VARCHAR(20)      NOT NULL,
        unique (Service_Type),
        FOREIGN KEY (ThreeMFilmType) REFERENCES ThreeMFilm(ThreeMFilmType));


CREATE TABLE repair_order
        (RONumber        CHAR(8)          NOT NULL,
        VNNo             CHAR(12)         NOT NULL,
        CustID           CHAR(8)          NOT NULL,
        EmployeeID       CHAR(10)             NOT NULL,
        Date             date             NOT NULL,
        ServiceType      VARCHAR(20)      NOT NULL,
        ServicePrice     int              NOT NULL,
        PRIMARY KEY (RO_Number),
        FOREIGN KEY (CustID) REFERENCES customer(CustID),
        FOREIGN KEY (VN_No) REFERENCES car(VN_No),
        FOREIGN KEY (EmployeeID) REFERENCES employee(EmployeeID));


CREATE TABLE works_on
        (WorksOnID       VARCHAR(10)      NOT NULL,
        EmployeeID       CHAR(10)         NOT NULL,
        RONumber         CHAR(8)          NOT NULL,
        Hours            int              NOT NULL,
        Film_Used        int              NOT NULL,
        FOREIGN KEY (EmployeeID) REFERENCES employee(EmployeeID),
        FOREIGN KEY (RO_Number) REFERENCES repair_order(RO_Number));


CREATE TABLE requests
        (RequestID       VARCHAR(10)          NOT NULL,
        EmpID            CHAR(10)             NOT NULL,
        Roll_size        int              NOT NULL,
        Quantity         int              NOT NULL,
        Date             date             NOT NULL,
        PRIMARY KEY (RequestID),
        FOREIGN KEY (EmployeeID) REFERENCES employee(EmployeeID));
```

Complex Queries

```sql
DELIMITER
CREATE DEFINER=`root`@`localhost` PROCEDURE `CustomerCars`()
BEGIN
    SELECT customer.CustID, customer.CustName, car.CarMaker, car.CarModel, car.Year
    FROM customer
    INNER JOIN car
    ON customer.CustID=car.CustID;
END;
END;
DELIMITER;


DELIMITER
CREATE DEFINER=`root`@`localhost` PROCEDURE `Markups`()
BEGIN
    SELECT service.Service_Type, threemfilm.Size, threemfilm.Markup
    FROM threemfilm, service
    WHERE service.ThreeMFilmType = threemfilm.ThreeMFilmType
    GROUP BY service.Service_Type;
END;
DELIMITER;



CREATE DEFINER=`root`@`localhost` PROCEDURE `OrdersByCust`()
BEGIN
    SELECT repairorder.RONumber, repairorder.VNNo, customer.CustID
    FROM repairorder
    INNER JOIN customer
    ON repairorder.CustID=customer.CustID;
END;


CREATE DEFINER=`root`@`localhost` PROCEDURE `GetCarsByCustomer`(
    IN inputID VARCHAR(10)
)
BEGIN
    SELECT inputID, CustName, VNNo, CarMaker, CarModel, Year
    FROM customer
    CROSS JOIN car
    WHERE inputID = car.CustID and inputID = customer.CustID;
END
```

```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetOrdersByCar`(
    IN inputVN CHAR(12)
)
BEGIN
    SELECT RONumber, car.VNNo, repairorder.CustID, car.CarMaker, car.CarModel, car.Year, repairorder.ServiceType
    FROM car
    CROSS JOIN repairorder
    WHERE inputVN = car.VNNo and inputVN = repairorder.VNNo;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `RepairByEmployee`(
    IN inputID CHAR(8)
)
BEGIN
    SELECT inputID, RONumber, repairorder.VNNo, repairorder.Date, repairorder.HoursWorked
    FROM employee
    CROSS JOIN repairorder
    WHERE inputID = repairorder.EmployeeID and inputID = employee.EmployeeID;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `GetMarkupByService`(
    IN inputServiceType VARCHAR(15)
)
BEGIN
    SELECT inputServiceType, threemfilm.ThreeMFilmType, threemfilm.Markup, threemfilm.QuantityRemain
    FROM service
    CROSS JOIN threemfilm
    WHERE inputServiceType = service.Service_Type and service.ThreeMFilmType = threemfilm.ThreeMFilmType;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `GetRObyDate`(
    IN inputDate datetime
)
BEGIN
    SELECT *
    FROM repairorder
    WHERE Date = inputDate
    ORDER BY inputDate;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `SortROByDate`()
BEGIN
    SELECT *
    FROM repairorder
    Order by date ASC;
END
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetRObyYear`(
    IN inputYear char(4)
)
BEGIN
    SELECT *
    FROM repairorder
    WHERE year(Date) = inputYear
    ORDER BY inputDate;
END
```

API Documentation with Postman

Here is the link to our published documentation for Postman:

https://documenter.getpostman.com/view/13837720/TVmV7Zst#af33aece-0f35-d752-82cf-da237e
a45177

- The left hand side shows all the functions available for the API
- When you look into them individually, it will have a brief explanation of what they do, an example on the right hand side, as well as any input parameters if needed
- It should look like this:

User Guide

To smoothly navigate this project:

1. Download the project folder
2. To access this API in browser mode as an admin, you must register as a superuser. To do that, use the command "**python3 manage.py createsuperuser**" in command prompt
   - It will ask you to create a username, input an email and create a password
   - If you leave username blank, it will take default username of pc user name
   - When it is successful, it will look like this:

   ```
   C:\Users\diana\OneDrive\Desktop\Project\MyProject>python3 manage.py createsuperuser
   Username: example
   Email address: dminniet@gmail.com
   Password:
   Password (again):
   Superuser created successfully.
   ```

3. Now we can start up the server for usage. First, call the command "**python3 manage.py runserver**" into your command prompt where the folder is located

   ```
   python3 manage.py runserver
   ```

   - When server is properly activated, this should show:

   ```
   System check identified no issues (0 silenced).
   December 11, 2020 - 19:51:57
   Django version 3.1.3, using settings 'MyProject.settings'
   Starting development server at http://127.0.0.1:8000/
   Quit the server with CTRL-BREAK.
   ```

4. Open the file "urls.py" with any editor you prefer, this will list all of the available endpoints for the API:

```
urlpatterns = [
    path('customers/', views.CustomerList.as_view()),
    path('customers/<pk>/', views.CustomerDetail.as_view()),

    path('cars/', views.CarList.as_view()),
    path('cars/<pk>/', views.CarDetail.as_view()),

    path('employees/', views.EmployeeList.as_view()),
    path('employees/<pk>/', views.EmployeeDetail.as_view()),

    path('threemfilms/', views.ThreeMFilmList.as_view()),
    path('threemfilms/<pk>/', views.ThreeMFilmDetail.as_view()),

    path('services/', views.ServiceList.as_view()),
    path('services/<pk>/', views.ServiceDetail.as_view()),

    path('orders/', views.ROList.as_view()),
    path('orders/<pk>/', views.RODetail.as_view()),

    # path('works/', views.WorksOnList.as_view()),
    path('requests/', views.ReqList.as_view()),
    path('requests/<pk>/', views.ReqDetail.as_view()),

    #Queries
    # <---- LIST OF CUSTOMERS AND THEIR CARS ---->
    path('custcars/', views.CustCars.as_view()),
    path('ordersbycust/', views.OrdersByCust.as_view()),
    path('markups/', views.Markups.as_view()),
    path('getcarsbycustomers/<value>/', views.GetCarsByCustomer.as_view()),
    path('getordersbycar/<value>/', views.GetOrdersByCar.as_view()),
    path('getmarkupbyservice/<value>/', views.GetMarkupByService.as_view()),
    path('getROByEmployee/<value>/', views.getROByEmployee.as_view()),
    path('sortRONew/', views.sortRONew.as_view()),
    path('getROByYear/<value>/', views.getROByYear.as_view()),
    path('getROByDate/<value>/', views.getROByDate.as_view())
]
```
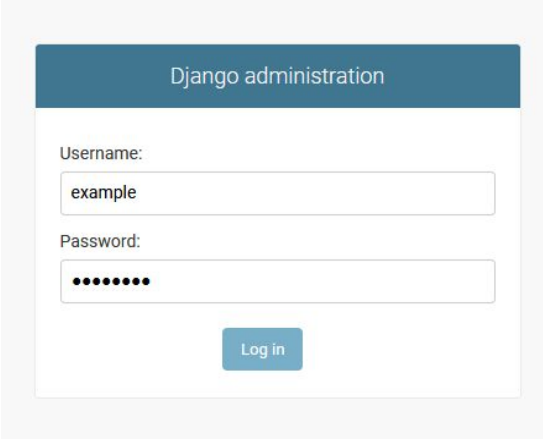
- The ones with a <value> or <pk> located at the end of the path will require an input, we will get to that further into the guide. For now, just have this open.

5. In browser mode, you can access http://127.0.0.1:8000/admin/ to view the API as a superuser
   ○ This prompt will show up:

Django administration
Username:
example
Password:
••••••••
Log in

   ○ Enter the details you used to sign up as a superuser to login
   ○ Now you should be Django Administration Portal of the API, it should look like this:

- In this portal you can see all the models that have been registered to this API
- This portal allows you to view all existing tuples of the entities - the models of the Django rest framework

6. Authentication and Authorization:
   - In this portal you as a superuser can manage the permissions by user group

   - Here is a user group named Employee and they have some but not all permissions such as:
      i. Ability to add, remove, view, and update all entities except for the Employee entity (only managers would be able to remove employee entities)

- ○ Managers have another level of permissions, similar to employees but with the ability to update and delete employee entities
- ○ Administrators have the most permissions, being allowed to do everything on that list such as:
  - i. Adding, updating, deleting, viewing any session logs
  - ii. Adding, updating, deleting, viewing any other permissions and groups of the API

7. In the "Users" Tab, you can add users as well, so other users do not have to go through the command prompt method of creating a user profile to login:

- ○ Here you can manage all users as well:

- ○ In each user instance, you can also manage their permission levels in the user profile as well
- ○ For example, as an administrator, you can manage the group(s) the users are a part of:

- ○ Usernames, personal info, and email addresses can be modified and updated as well

8. Moving on to the actual API, we have the models that have been integrated into the API:
    - ○ Cars, Customers, Employees, Repair Orders, Reqs, Services, Three m films
    - ○ Each represent an entity in our model

Api administration

API		
Cars	+ Add	✏ Change
Customers	+ Add	✏ Change
Employees	+ Add	✏ Change
Repair orders	+ Add	✏ Change
Reqs	+ Add	✏ Change
Services	+ Add	✏ Change
Three m films	+ Add	✏ Change

- ○ When you click on a model, it will show you all existing tuples of that entity with the newest ones first
- ○ For example:

- ○ In the top right corner, you can add a new car tuple into the database, it will require to put in all the necessary info as all of it is set to non-nullable defaults
- ○ When looking into a tuple, you are able to modify or delete it

○ An interesting feature of this API is that it will track the history of any changes and updates to existing records:

Change history: 00003

DATE/TIME	USER	ACTION
Dec. 9, 2020, 2:59 a.m.	diana	Added.
Dec. 9, 2020, 5:26 p.m.	diana	No fields changed.

9. Now moving onto the urls that we will be using. These views can be done in browser mode or Postman, but following project requirements, this guide will be made for the Postman app.
   ○ Please open up the Postman app and have our Postman Documentation ready and available
   ○ Here is the link again if you do not want to scroll back up: https://documenter.getpostman.com/view/13837720/TVmV7Zst#af33aece-0f35-d752-82cf-da237ea45177

10. With Postman, you can put in the url after "http://127.0.0.1:8000/"
    ○ For example, here is the getROByEmployee/ endpoint:
    ○ Let us look at all repair orders completed by employee 0001
    ○ Please input: http://127.0.0.1:8000/getROByEmployee/0001/ into the params field
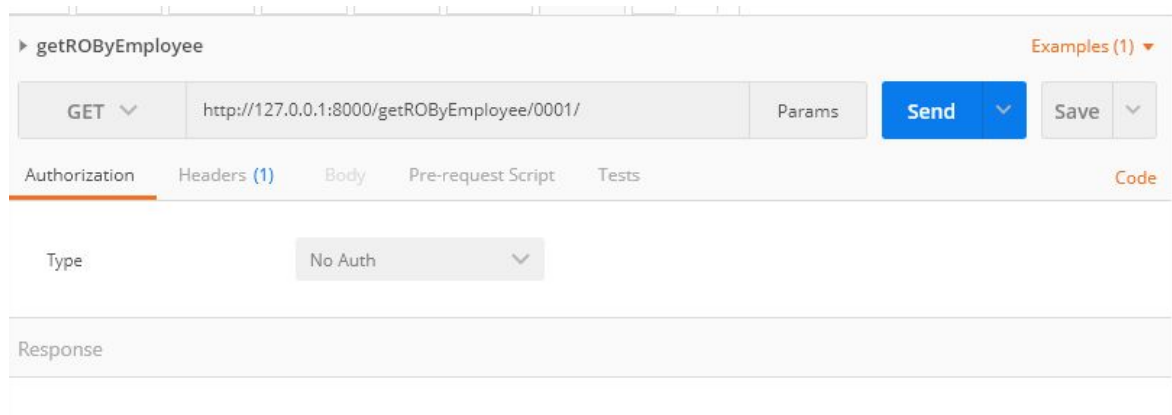
○ This will only work if you input an existing employee ID
○ You can view all the existing employee IDs by viewing all the employees
○ The url endpoints that require an input parameter either end with "<pk>/" or "<value>/"
  i. "<pk>/" endpoints require an existing value that existing as a primary key in the database, otherwise it will return an error
  ii. "<value>/" endpoints require a value that also existing in the database for what it is inputting into the stored procedure in SQL, if the value does not exist, it will return empty
○ Please refer to the Postman Documentation for any examples of the API functions that are available to use as the full list is there with explanation of what task each call does

- You can do these with all the available endpoints in the urls.py file of the project

11. That concludes the guide of working through our API, hopefully this was useful.

References

- Youtube Tutorial :Django REST Framework Full Course For Beginners | Build REST API With Django (https://www.youtube.com/watch?v=B38aDwUpcFc&t=7195s)
- YouTube Tutorial: Django Rest API CRUD - GET, POST, PUT and DELETE (https://www.youtube.com/watch?v=1k0fRG098cU&t=910s
- YouTube Tutorial: How to Connect MySQL Database with Django Project (https://www.youtube.com/watch?v=SNyCV8vOr-g)
- Django Documentation: https://docs.djangoproject.com/en/3.1/
- Sample project provided by TA's (https://d2l.ucalgary.ca/d2l/le/content/327858/viewContent/4343101/View)