Auto Repair Shop

Final Report

CPSC 471 (Fall 2020)

Professor – *Dr. Reda* Elhajj

TA – T*amer Jarada*

Group 43

Jeffrey Layton

Simone Mendonca

Teodor Tebeica

Table of Contents

Abstract

AutoRepair is an auto shop administration system allowing users to easily handle employee, customer, and work order management as well as store new vehicles and keep an inventory on parts. The design of the product was first done through the representation of it in an extended entity relation diagram showcasing the relations between all entities in our system after which we have converted to a relational model to help us with the final implementation in T-SQL for Microsoft SQL Server. This report will contain details about every step we made from our database design to the final implementation.

Introduction

Problem Overview

The automotive industry is growing faster and faster every day. The need for reliable database systems in the industry is greater than ever before. Individuals working in automotive repair shops need quick and dependable access to their work orders, parts inventory, vehicle specifications, owner information and much more. The significantly used software in the market is not designed with ease of use in mind. It can take a long time for employees to get accustomed to the system features and how they are meant to be used. There are many systems out there that are very comprehensive and encompass every aspect required by automobile mechanic shops to complete daily activities, however, many of them suffer from being inconvenient and annoying to use.

Solution Overview

AutoRepair is our solution to this problem. We have designed AutoRepair with ease of use in mind, so right from the backend, our product is not overly complicated with long procedures. Our API can handle some error checking and most of our endpoints are simple to navigate and use. Users can expect a smooth and easy interaction with the system, with any request they make, be it adding a new employee to the database, creating a new work order, adding a new customer or modifying any of the existing entries!

Project Design

User Entities

This project was designed to provide automobile shop employees with easy access to information about the requested work, general and specific vehicle information, parts inventory and information about the vehicle owner and shop employees. There are four user types of this system: admin, manager, mechanic, and clerk. They each have certain roles and relations set by their corresponding flags (AFlag, ManFlag, MecFlag, CFlag). Admin has access to all features and can update and change the vehicle and parts library that is locally stored. Managers can manage daily back office operations, client data, and create employee invoices. Finally, clerks can view the status of the repair, add new customers/work orders to the system and create billing invoices for the workorders.

The following is a list of entities and their relations:

- *EMPLOYEE* – all the entities that can log in to the system.
  - Total participation, overlapping specialization – MECHANIC OR CLERK OR MANAGER OR ADMIN (an employee can be multiple)
  - MANAGED_BY relation from multiple employees to ONE MANAGER

    CLERK – Employees that have status 'clerk'

    - ASSOCIATED_WITH many-to-many relation linked to WORK_ORDER (a work_order does not need to have a corresponding clerk)

    MECHANIC – Employees that have status 'mechanic'

    - ASSIGNED_TO many-to-many relation linked to WORK_ORDER (a work_order must have at least one mechanic assigned to it)

    Modifications brought to this entity:

    - Added attribute EPassword that stores an employee's password, used upon logging in using their employee_id, EPassword combination
    - Had to change "Address" to "EAddress" due to SQL syntax constraints.

- *EMPLOYEE_INVOICE* – a list of invoices for the GETS_PAID relationship between employee and employee_invoice.

  Modifications brought to this entity:

  - Had to change "Hours" to "WHours" due to SQL syntax constraints.

- *WORK_ORDER* – specific information about customer, vehicle info, parts needed to complete order contained within
  - *TIED_TO* relation linked to CUSTOMER (a customer must have at least one work order, a work order must have at least one customer)

- *TO_FIX* relation linked to VEHICLE (a work order must have at least one vehicle and a vehicle must have at least one work order)

  Modifications brought to this entity:

  - Foreign key from the customer table became CustomerID, due to changes to the customer table.

- *CATALOG_PART* – a library of parts and their specific compatibility on vehicles
  - *INSTANCE_OF* identifying relation of PART
  - *COMPATIBLE_WITH* many-to-many relation linked to VEHICLE_MODEL (represents whether a specific part of CATALOG_PART is compatible with a specific Vehicle model)

- *PART* – a specific instance of a Part that is used in a work order – this is our WEAK ENTITY
  - *BILLED_TO* many to one relation linked to WORK_ORDER (multiple of the same part can be used in one work order (e.g., spark plugs))

- *VEHICLE_MODEL* – a generic vehicle model library

  Modifications brought to this entity:

  - Replaced attribute "Name" with "Make" and added attribute "Model" to properly classify certain makes and models.
  - Had to change "Year" to "VYear" due to SQL syntax constraints.
  - The primary key for this table is {Make, Model, VYear}.

  - *INSTANCE_OF* one-to-many relation linked to VEHICLE (a specific vehicle instance of Vehicle_model that it inherits)

- *VEHICLE* – an instance of a vehicle model to be FIXED for the workorder and OWNED BY a customer
  - *BELONGS_TO* many to one relation linked to CUSTOMER (a customer can own multiple vehicles but there must be at least one customer and one vehicle that are linked together)

  Modifications brought to this entity:

  - Now references Vehicle_model.make, vehicle_model.model, vehicle_model.year and customer. CustomerID due to the other changes made to this database model.

- *CUSTOMER* – holds information about a customer that owns a vehicle & is linked to a work order

  Modifications brought to this entity:

  - Removed the key set {Fname, Phone_number} (attributes still used) and added a unique CustomerID to be the primary key.
  - Had to change "Address" to "CAddress" due to SQL syntax constraints.

Authentication: *the employee_id is what is being used for username and EPassword as their password.*

ENDPOINT	MecFlag	CFlag	ManFlag	AFlag
getEmployee	✔ *	✔ *	✔	✔
addEmployee			✔	✔
payEmployee			✔	✔
updateEmployeeInfo			✔	✔
getInvoices	✔ *	✔ *	✔	✔
getInvoice	✔ *	✔ *	✔	✔
getManagersDelegates			✔	✔
assignMechanic		✔	✔	✔
assignClerk		✔	✔	✔
createWorkOrder		✔	✔	✔
updateWorkOrder		✔	✔	✔
getCustomerWorkOrders		✔	✔	✔
getWorkOrder	✔	✔	✔	✔
getEmployeeWorkOrders	✔	✔	✔	✔
removeWorkOrder			✔	✔
addCustomer		✔	✔	✔
updateCustomer		✔	✔	✔
getCustomer		✔	✔	✔
getCustomerVehicles		✔	✔	✔
addVehicle		✔	✔	✔
updateVehicle		✔	✔	✔
getVehicle	✔	✔	✔	✔
checkVehicleModel			✔	✔
addVehicleModel				✔
removeVehicleModel				✔
getWorkOrderParts	✔	✔	✔	✔
getPartsOnStock	✔		✔	✔
updatePart	✔		✔	✔
getPart	✔		✔	✔
addPart	✔		✔	✔
searchCompatiblePart	✔		✔	✔
getCatalogPart	✔		✔	✔
addCatalogPart				✔
removeCatalogPart				✔
addCompatibility				✔
checkPartCompatibility	✔		✔	✔

* - restricted access, can only view items linked to their Employee_id

Transaction Collection

Disclaimer: the data fields input by the user must adhere to the table found at the end of this report - Appendix #2
Note: The implementation of SQL commands and stored procedures protect against SQL injections

The following are example uses of our full transaction collection:

- getEmployee GET

  This endpoint is a utility that helps a user sort through the employee table provided they know that employee's unique ID. It returns a tuple in the employee table that matches the employee_id passed as parameter (if it exists in the table).

  - Employee_id INT

- addEmployee POST

  This endpoint is a utility for managers and admins to use in order to add new users to the system (employee table). It requires the following as parameters {@EPassword, @Lname, @Fname, @EAddress, @Bank_acc_no, @Salary_rate, @Hourly_rate, @Pay_type, @MecFlag, @CFlag, @ManFlag, @AFlag, @Manager_id). The result will be a new tuple added to the employee table.

  - @EPassword VARCHAR(128)
  - @Lname VARCHAR(128)
  - @Fname VARCHAR(128)
  - @EAddress VARCHAR(256)
  - @Bank_acc_no BIGINT
  - @Salary_rate DECIMAL(10,2)
  - @Hourly_rate DECIMAL(10,2)
  - @Pay_type BIT
  - @MecFlag BIT
  - @CFlag BIT
  - @ManFlag BIT
  - @AFlag BIT
  - @Manager_id INT

- payEmployee POST

  This endpoint is a utility for managers to use in order to add a new invoice to the system (employee_invoice table). It requires the following parameters {@Invoice_id, @Employee_id, @Amount, @Interval_start_date, @Interval_end_date, @Payment_date, @WHours}

  - @Employee_id INT
  - @Amount DECIMAL(10,2)
  - @Interval_Start_Date DATE
  - @Interval_End_Date DATE
  - @Payment_Date DATE
  - @WHours DECIMAL(10,2)

- updateEmployeeInfo PUT

  This endpoint is a utility for managers and admins to use in order to modify the stored information of a certain employee. Given an Employee_id and new attributes for that specific employee, the procedure will first check if that Employee_id exists and update the values stored for them.

  - @Employee_id INT
  - @EPassword VARCHAR(128)
  - @Bank_acc_no INT
  - @EAddress VARCHAR(256)
  - @Lname VARCHAR(128)
  - @Fname VARCHAR (128)
  - @Pay_type BIT
  - @Salary_rate DECIMAL(10,2)
  - @Hourly_rate DECIMAL(10,2)
  - @MecFlag BIT
  - @CFlag BIT
  - @ManFlag BIT
  - @AFlag BIT
  - @Manager_id INT

- getInvoices GET

  This endpoint is a utility for users to view their payment invoices given a certain Employee_id. The expected result will be a list of invoices (tuples in employee_invoice table) for a specific employee

  - @Employee_id INT

- getInvoice GET

  This endpoint is a utility for users to view a specific invoice, given the Invoice_id and Employee_id. The expected result is a singular tuple from the employee_invoice table.

  - @Invoice_id INT
  - @Employee_id INT

- getManagersDelegates GET

  This endpoint can be used to query the database given an Employee_id and see which employees are being managed by the given Employee_id. The expected outcome is 0 or more employee_id's from the employee table that a given employee (that is a manager) delegates.

  - @Employee_id INT

- assignMechanic POST

  This endpoint can be used to assign a work order to a mechanic (inserting a new tuple into the assigned_to table). This way, mechanics are able to view the work that is assigned to them and query the work_order table to get more information based on the work_order_id.

  - @Employee_id INT
  - @Work_order_id INT

- assignClerk POST

  This endpoint can be used to assign a work order to a clerk (inserting a new tuple into the associated_with table). This way, clerks are able to view the work that is assigned to them and query the work_order table to get more information based on the work_order_id. In this example, the employee_id is known as a clerk_id in the associated_with table.

  - @Employee_id INT
  - @Work_order_id INT

- createWorkOrder POST

  This endpoint can be used to create a new work order. The work_order table references the customer and vehicle table to add specific customer and vehicle information. This is where a clerk or manager would use their role to correctly create a new work order.

  - @Closed BIT
  - @Amount_due DECIMAL(10,2)
  - @Vehicle_VIN VARCHAR(17)
  - @CustomerID INT

- updateWorkOrder PUT

  This endpoint can be used to modify an existing work order. The procedure first checks the work_order table for the specified work_order_id and if it exists, it will modify its attributes.

  - @Work_order_id INT
  - @Closed BIT
  - @Amount_due DECIMAL(10,2)
  - @Vehicle_VIN VARCHAR(17)
  - @CustomerID INT

- getCustomerWorkOrders GET

  This endpoint can be used to retrieve a specific customer's work orders given their unique CustomerID. This endpoint is useful for finding the right work order for a customer that has had multiple mechanical work done on their vehicles. This endpoint can be used if the user does not know the specific work_order_id that they are looking for before using getWorkOrder.

  - @CustomerID INT

- getWorkOrder GET

  This endpoint retrieves a specific work_order tuple from the database given the Work_order_id that is being passed as parameter exists in that table.

  - @Work_order_id INT

- getEmployeeWorkOrders GET

  This endpoint can be used to retrieve a specific employee's work orders given that it was assigned to them using `assignMechanic` or `assignClerk` (i.e. the tuple {Employee_id, Work_order_id} shows up in the assigned_to or associated_with tables).

  - @Employee_id INT

- removeWorkOrder DEL

  This endpoint can be used to remove a specific work order tuple from the work_order table. Users must specify a Work_order_id to remove. Entries in the work order table should not be deleted, but instead modified using updateWorkOrder. This method is here for testing purposes and to show that the database is capable of performing this action.

  - @Work_order_id INT

- addCustomer POST

  This endpoint is used to add a new customer to the system. The customer table holds the following values about a customer: a customer ID (automatically assigned by the system), first name, last name, customer address and phone number. This endpoint has to be performed before creating a new work order since the work_order table references CustomerID.

  - @Fname VARCHAR(128)
  - @Lname VARCHAR(128)
  - @CAddress VARCHAR(256)
  - @Phone_number VARCHAR(15)

- updateCustomer PUT

  This endpoint is used to update an existing customer's attributes. Given a matching CustomerID, and new information about that customer, the endpoint can modify the database and update it as the user intends.

  - @CustomerID INT
  - @Fname VARCHAR(128)
  - @Phone_number VARCHAR(15)
  - @Lname VARCHAR(128)
  - @Address VARCHAR(256)

- getCustomer GET

  This endpoint is used to retrieve an existing customer's attributes from the database to be viewed. The user must specify a CustomerID and can expect in return a tuple from the customer table (provided that the CustomerID specified exists).

  - @CustomerID INT

- getCustomerVehicles GET

  This endpoint is for retrieving information on all the vehicles that belong to a given customer – based on the CustomerID key. First, it checks if the customerID has any vehicle tied to it.

  - @CustomerID INT

- getVehicle GET

  This endpoint can be used to view a specific vehicle's information. If the vehicle has ever been serviced, a quick search using the vehicle's VIN will return a tuple from the vehicle table containing information about it.

  - @VIN VARCHAR(17)

- addVehicle POST

  This endpoint is used to add a new vehicle to the system. The vehicle table holds the following values about a vehicle: a customer ID (referencing the customer table), unique Vehicle Identification Number, Color, registration number, vehicle make, model and year (referencing vehicle_model table). During the creation of a new work order, this endpoint has to be performed before creating a new customer (which is performed before creating a new work order) since the vehicle table references the CustomerID.

  - @VIN VARCHAR(17)
  - @Vehicle_make VARCHAR(32)
  - @Vehicle_model VARCHAR(32)
  - @Vehicle_year INT
  - @Color VARCHAR(32)
  - @CustomerID INT
  - @Registration_No VARCHAR(20)

- updateVehicle PUT

  This endpoint is used to edit a vehicle that is already in the system. The vehicle table holds the following values about a vehicle: a customer ID (referencing the customer table), unique vehicle identification number, color, registration number, and vehicle make, model and year (referencing vehicle_model table).

  - @VIN VARCHAR(17)
  - @Vehicle_make VARCHAR(32)
  - @Vehicle_model VARCHAR(32)
  - @Vehicle_year INT
  - @Color VARCHAR(32)
  - @CustomerID INT
  - @Registration_No VARCHAR(20)

- checkVehicleModel GET

  This endpoint can be used to check if a specific vehicle model (i.e. 2007 Ford Explorer) exists in the database, specifically in the vehicle_model table. This endpoint should mostly be used by an admin user to double check specific vehicles are part of the system. If not, they can perform addVehicleModel to add that vehicle in.

  - @Vehicle_make VARCHAR(32)
  - @Vehicle_model VARCHAR(32)
  - @Year INT

- addVehicleModel POST

  This endpoint can be used to add a new vehicle model (i.e. 2007 Ford Explorer) to the database, specifically in the vehicle_model table. This endpoint should mostly be used by an admin user to insert new vehicles into the system.

  - @Vehicle_make VARCHAR(32)
  - @Vehicle_model VARCHAR(32)
  - @Year INT

- removeVehicleModel DEL

  This endpoint can be used to remove a specific vehicle model (i.e. 2007 Ford Explorer) from the database, specifically in the vehicle_model table. This endpoint should mostly be used by an admin user to remove specific vehicle models that are part of the system.

  - @Vehicle_make VARCHAR(32)
  - @Vehicle_model VARCHAR(32)
  - @Year INT

- getWorkOrderParts GET

  This endpoint is useful to retrieve the parts that are used for a specific work order. It takes in a unique work_order_id and looks through the parts data table to find parts with the same work_order_id attribute value. Given this information, the prices of the parts for a work order can be used to the final amount due for the work order.

  - @Work_order_id INT

- getPartsOnStock GET

  This endpoint is useful to retrieve the parts that are in inventory (meaning they do not have a work order id linked to them). This procedure takes no input from the user and returns a list (can empty) of parts that have their work_order_id set to null or 0.

- updatePart PUT

This endpoint is used to modify the stored information on a particular part in the system. The user needs to know the Part_id and the unique Part_instance_no in order to access it. Specifically, this method can be used to update the state of a part, the price of a part or to add or update the work_order_id the part is associated with.

- @Part_id INT
- @Part_instance_no INT
- @PState VARCHAR(32)
- @Price DECIMAL(10,2)
- @Work_order_id INT

- getPart GET

This endpoint can be used to retrieve information on a part given its Part_id and Part_instance_no. The user can get the part's name, price, state and associated work order.

- @Part_id INT,
- @Part_instance_no INT

- addPart POST

This endpoint is used to add a new part to the system. The user provides the parameters: {Part_id, PState, Price, Word_order_id}. If the part is not yet assigned to a work order, the value can be passed in as 0 and will be set to null in the data table.

- @Part_id INT,
- @PState VARCHAR(32),
- @Price DECIMAL(10,2),

- searchCompatiblePart GET

This endpoint can be used by users to find part IDs and names of parts given a vehicle_model reference (vehicle_make, vehicle_model, vehicle_year) and a partial or complete name of a part. This is useful when users do not know the exact part id for specific parts they are looking for.

- @Vehicle_make VARCHAR(32)
- @Vehicle_model VARCHAR(32)
- @Vehicle_year INT
- @Part_name_string VARCHAR(256)

- getCatalogPart GET

This endpoint is used to retrieve a catalog part using its Part_id. The Part_name is given by the data table.

- @Part_id INT

- addCatalogPart POST

  This endpoint is used to add an object (part) to the catalog table. It takes in the Part_name and automatically assigns a part_id using an auto-increment feature. This Part_id is the primary key

  - @Part_name VARCHAR(64)

- removeCatalogPart DEL

  This endpoint is used to delete a particular catalog_part tuple from the data table based on its Part_id. Once deleted, the tuples in the part table and compatible_with tables with the matching Part_id will also be deleted

  - @Part_id VARCHAR(32)

- addCompatibility POST

  This endpoint is used to add a relation between a part and a vehicle_model. It takes in the Part_id, and the vehicle make, model and year. These 4 attributes together make up the primary key

  - @Part_id INT,
  - @Vehicle_Make VARCHAR(32),
  - @Vehicle_Model VARCHAR(32),
  - @Vehicle_Year INT

- checkPartCompatibility GET

  This endpoint takes in a Part_id, and Vehicle model details (make, model year) and checks for the existence of a tuple in the compatible_with data table with the matching values. If the tuple exists, it returns a 1 and otherwise returns 0.

  - @Part_id INT,
  - @Vehicle_Make VARCHAR(32),
  - @Vehicle_Model VARCHAR(32),
  - @Vehicle_Year INT

EER Diagram

Figure 1. shows our original design for the system. Once we started converting to the relational model , we quickly realized we were missing some attributes for primary key identification.

For example, we initially had the customer's first name and phone number as primary keys for customer.

In our redesign, we added a CustomerID as a single, unique primary key for the Customer entity.

Figure 1.

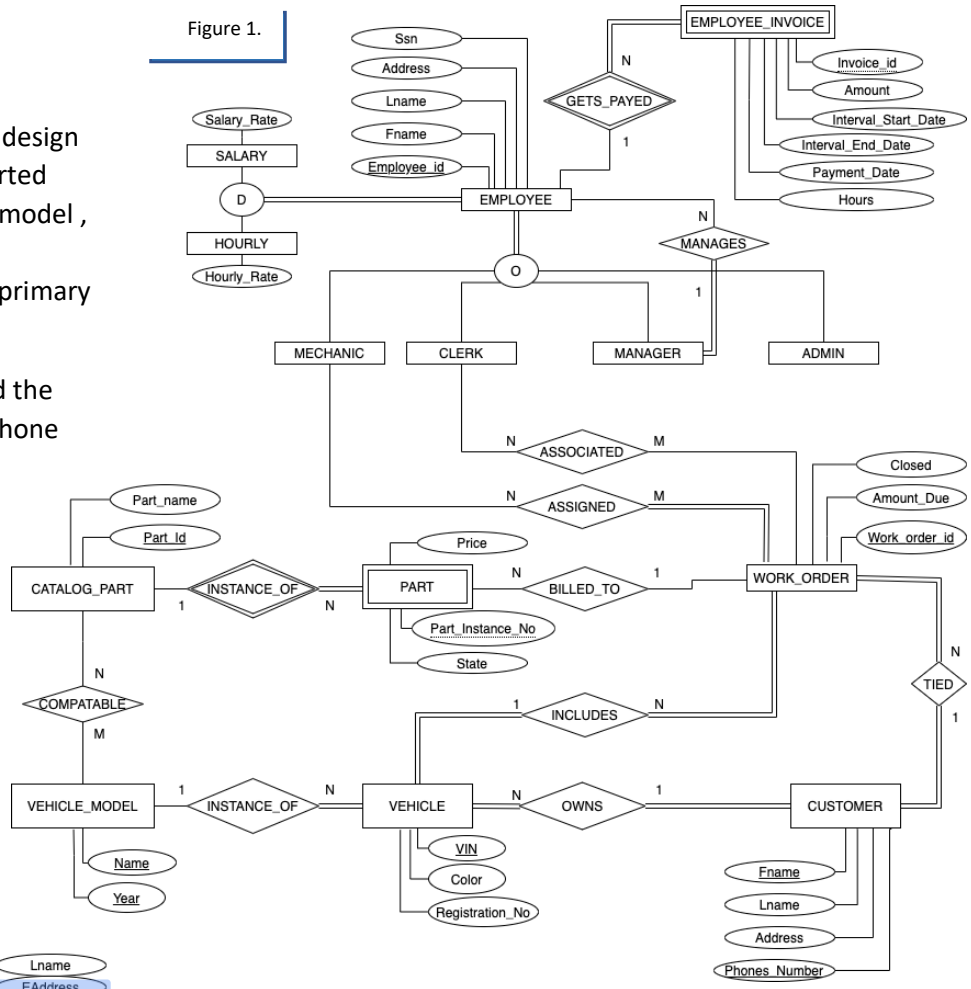


Figure 2.

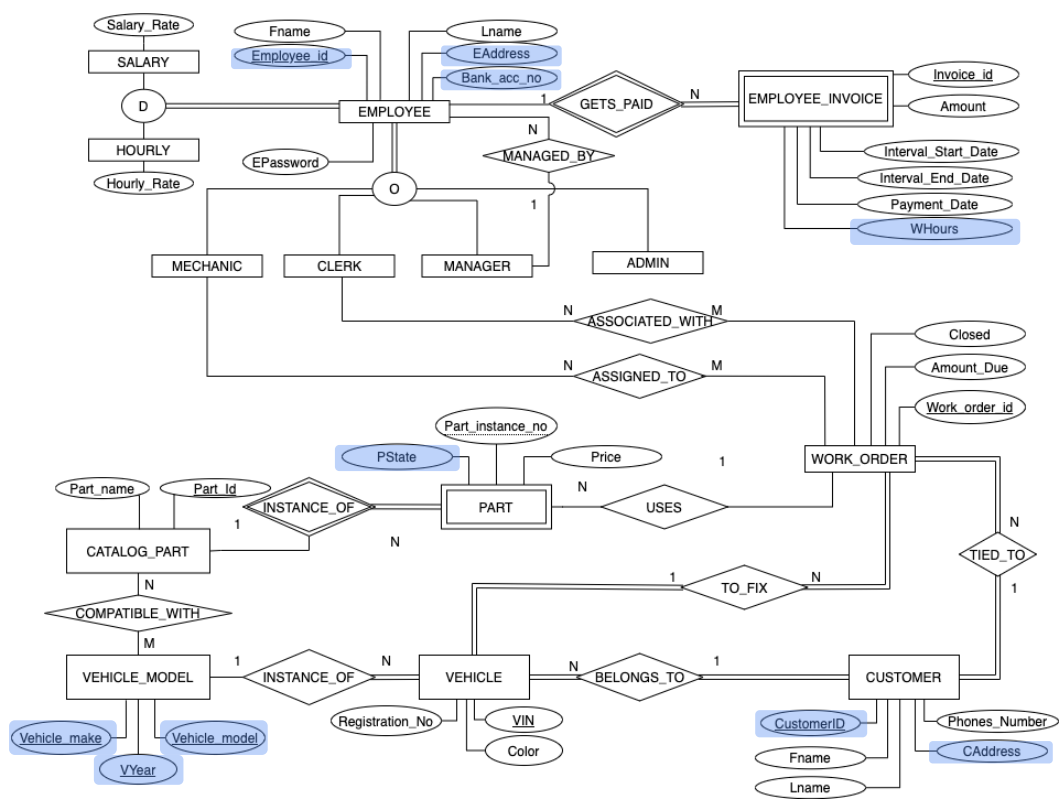
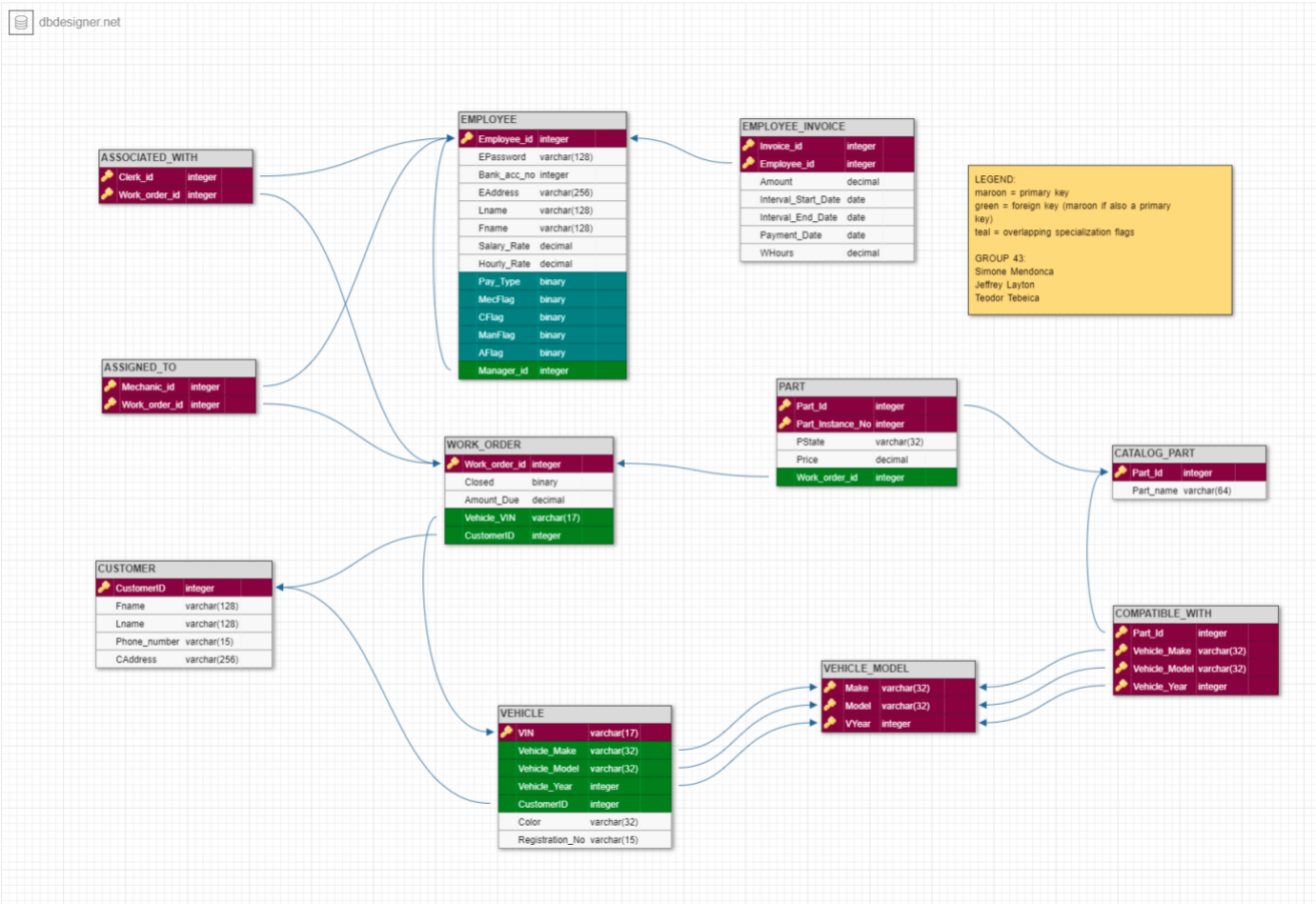


Figure 2. As seen here, we had to adjust the names of some attribute to remove overlap with any SQL keywords.

Another significant adjustment was the keys for the Vehicle_model entity. We separated the 'name' attribute into 'make' and 'model' to allow for searches based on vehicle make.

Implementation (Solution)

Relational Model Diagram



Database Specifications

The DBMS we chose is Microsoft SQL Server. The following are the SQL statements we used to define our stored procedures:

```
CREATE PROCEDURE [dbo].[addCatalogPart]
    @Part_name VARCHAR(64)
AS
    BEGIN
        INSERT INTO catalog_part(Part_name)
        VALUES (@Part_name)
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[addCompatibility]
    @Part_id INT,
    @Vehicle_Make VARCHAR(32),
    @Vehicle_Model VARCHAR(32),
    @Vehicle_Year INT
AS
    BEGIN
        IF NOT EXISTS (SELECT * FROM compatible_with
                        WHERE Part_id = @Part_id AND
                        Vehicle_make = @Vehicle_Make AND
                        Vehicle_model = @Vehicle_Model AND
                        Vehicle_year = @Vehicle_Year)
        BEGIN
            INSERT INTO compatible_with(Part_id, Vehicle_make, Vehicle_model,
Vehicle_year)
            VALUES (@Part_id, @Vehicle_Make, @Vehicle_Model, @Vehicle_Year)
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[addCustomer]
    @Fname VARCHAR(128),
    @Lname VARCHAR(128),
    @CAddress VARCHAR(256),
    @Phone_number VARCHAR(15)
AS
    BEGIN
        IF NOT EXISTS (SELECT * FROM customer
                        WHERE Fname = @Fname AND
                        Phone_number = @Phone_number AND
                        Lname = @Lname AND
                        CAddress = @CAddress)
        BEGIN
            INSERT INTO customer(Fname, Phone_number, Lname, CAddress)
            VALUES(@Fname, @Phone_number, @Lname, @CAddress)
        END
    END


-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[addEmployee]
    @EPassword VARCHAR(128),
    @Lname VARCHAR(128),
```

```
    @Fname VARCHAR (128),
    @EAddress VARCHAR(256),
    @Bank_acc_no BIGINT,
    @Salary_rate DECIMAL(10,2),
    @Hourly_rate DECIMAL(10,2),
    @Pay_type BIT,
    @MecFlag BIT,
    @CFlag BIT,
    @ManFlag BIT,
    @AFlag BIT,
    @Manager_id INT
AS
    BEGIN
        IF NOT EXISTS (SELECT * FROM employee
                        WHERE EPassword = @EPassword AND
                        Bank_acc_no = @Bank_acc_no AND
                        EAddress = @EAddress AND
                        Lname = @Lname AND
                        Fname = @Fname AND
                        Pay_type = @Pay_type AND
                        Salary_rate = @Salary_rate AND
                        Hourly_rate = @Hourly_rate AND
                        MecFlag = @MecFlag AND
                        CFlag = @CFlag AND
                        ManFlag = @ManFlag AND
                        AFlag = @AFlag AND
                        (Manager_id = @Manager_id OR (Manager_id IS NULL AND
@Manager_id < 1)))
        BEGIN
            IF(@Manager_id < 1)
            BEGIN
                INSERT INTO employee(EPassword, Bank_acc_no, EAddress, Lname,
Fname,
                            Pay_type, Salary_rate, Hourly_rate, MecFlag,
CFlag, ManFlag,
                            AFlag, Manager_id)
                VALUES (@EPassword, @Bank_acc_no, @EAddress, @Lname, @Fname,
@Pay_type,
                        @Salary_rate, @Hourly_rate, @MecFlag, @CFlag,
@ManFlag,
                        @AFlag, NULL)
            END
            ELSE
            BEGIN
                INSERT INTO employee(EPassword, Bank_acc_no, EAddress, Lname,
Fname,
                            Pay_type, Salary_rate, Hourly_rate, MecFlag,
CFlag, ManFlag,
                            AFlag, Manager_id)
                VALUES (@EPassword, @Bank_acc_no, @EAddress, @Lname, @Fname,
@Pay_type,
                        @Salary_rate, @Hourly_rate, @MecFlag, @CFlag,
@ManFlag,
                        @AFlag, @Manager_id)
            END
        END
    END
```

```
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[addPart]
    @Part_id INT,
    @PState VARCHAR(32),
    @Price DECIMAL(10,2),
    @Work_order_id INT
AS
    BEGIN
        IF NOT EXISTS (SELECT * FROM part P
                        WHERE P.Part_id = @Part_id AND
                            P.PState = @PState AND
                            P.Price = @Price AND
                            (P.Work_order_id = @Work_order_id OR
(P.Work_order_id IS NULL AND @Work_order_id < 1)))
        BEGIN
            IF(@Work_order_id < 1)
            BEGIN
                INSERT INTO part(Part_id, PState, Price, Work_order_id)
                VALUES (@Part_id, @PState, @Price, NULL)
            END
            ELSE
            BEGIN
                INSERT INTO part(Part_id, PState, Price, Work_order_id)
                VALUES (@Part_id, @PState, @Price, @Work_order_id)
            END
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[addVehicle]
    @VIN VARCHAR(17),
    @Vehicle_make VARCHAR(32),
    @Vehicle_model VARCHAR(32),
    @Vehicle_year INT,
    @Color VARCHAR(32),
    @CustomerID INT,
    @Registration_No VARCHAR(20)
AS
    BEGIN
        IF NOT EXISTS (SELECT * FROM vehicle V
                        WHERE v.VIN = @VIN)
        BEGIN
            INSERT INTO vehicle(VIN, Vehicle_make, Vehicle_model,
Vehicle_year, Color, CustomerID, Registration_No)
            VALUES (@VIN, @Vehicle_make, @Vehicle_model, @Vehicle_year,
@Color, @CustomerID, @Registration_No)
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[addVehicleModel]
    @Vehicle_make   VARCHAR(32),
    @Vehicle_model  VARCHAR(32),
    @Year           INT
AS
    BEGIN
        IF NOT EXISTS (SELECT * FROM vehicle_model
                        WHERE Make = @Vehicle_make AND
                        Model = @Vehicle_model AND
```

```
                        VYear = @Year)
        BEGIN
            INSERT INTO vehicle_model(Make, Model, VYear)
            VALUES (@Vehicle_make, @Vehicle_model, @Year)
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[assignClerk]
    @Work_order_id INT,
    @Employee_id INT
AS
    BEGIN
        IF NOT EXISTS (SELECT * FROM associated_with
                        WHERE Work_order_id = @Work_order_id AND
                        Clerk_id = @Employee_id)
        BEGIN
            INSERT INTO associated_with(Clerk_id, Work_order_id)
            VALUES (@Employee_id, @Work_order_id)
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[assignMechanic]
    @Work_order_id INT,
    @Employee_id INT
AS
    BEGIN
        IF NOT EXISTS (SELECT * FROM assigned_to
                        WHERE Work_order_id = @Work_order_id AND
                        Mechanic_id = @Employee_id)
        BEGIN
            INSERT INTO assigned_to(Mechanic_id, Work_order_id)
            VALUES (@Employee_id, @Work_order_id)
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[checkPartCompatibility]
    @Part_id        INT,
    @Vehicle_Make   VARCHAR(32),
    @Vehicle_Model  VARCHAR(32),
    @Vehicle_Year   INT
AS
    BEGIN
        SELECT CAST (CASE WHEN EXISTS (
            SELECT * FROM compatible_with W
            WHERE ( W.Part_id = @Part_id AND
                    W.Vehicle_make = @Vehicle_Make AND
                    W.Vehicle_model = @Vehicle_Model AND
                    W.Vehicle_year = @Vehicle_Year) )
            THEN 1
            ELSE 0
            END AS BIT)
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[checkVehicleModel]
    @Vehicle_make   VARCHAR(32),
    @Vehicle_model  VARCHAR(32),
    @Year           INT
```

```
AS
    BEGIN
        SELECT CAST (CASE WHEN EXISTS (
            SELECT * FROM vehicle_model V
            WHERE ( V.Make = @Vehicle_Make AND
                    V.Model = @Vehicle_Model AND
                    V.VYear = @Year) )
            THEN 1
            ELSE 0
            END AS BIT)
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[createWorkOrder]
    @Closed BIT,
    @Amount_due DECIMAL(10,2),
    @Vehicle_VIN VARCHAR(17),
    @CustomerID INT
AS
    BEGIN
        IF NOT EXISTS (SELECT * FROM work_order W
                        WHERE W.Closed = @Closed AND
                            W.Amount_Due = @Amount_due AND
                            W.Vehicle_VIN = @Vehicle_VIN AND
                            W.CustomerID = @CustomerID)
        BEGIN
            INSERT INTO work_order(Closed, Amount_Due, Vehicle_VIN,
CustomerID)
            VALUES (@Closed, @Amount_due, @Vehicle_VIN, @CustomerID)
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getCatalogPart]
    @Part_id    INT
AS
    Begin
        IF EXISTS (SELECT * FROM catalog_part C
                    WHERE C.Part_id = @Part_id)
        BEGIN
            SELECT * FROM catalog_part C
                    WHERE C.Part_id = @Part_id
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getCustomer]
    @CustomerID INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM customer C
                    WHERE C.CustomerID = @CustomerID)
        BEGIN
            SELECT * FROM customer C
                    WHERE C.CustomerID = @CustomerID
        END
    END
```

```
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getCustomerVehicles]
    @CustomerID INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM vehicle V
                    WHERE V.CustomerID = @CustomerID)
        BEGIN
            SELECT * FROM vehicle V
            WHERE V.CustomerID = @CustomerID
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getCustomerWorkOrders]
    @CustomerID     INT
AS
    BEGIN
        IF EXISTS (Select * from work_order W
                    WHERE W.CustomerID=@CustomerID)
        BEGIN
            Select * from work_order W
            WHERE W.CustomerID=@CustomerID
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getEmployee]
    @Employee_id INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM employee E
            WHERE E.Employee_id = @Employee_id)
        BEGIN
            SELECT * FROM employee E
            WHERE E.Employee_id = @Employee_id
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getEmployeeWorkOrders]
    @Employee_id INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM employee E
                    WHERE E.Employee_id = @Employee_id)
        BEGIN
            SELECT DISTINCT W.Work_order_id, W.Closed, W.Amount_Due,
W.CustomerID, W.Vehicle_VIN FROM work_order W, assigned_to A, associated_with
B
            WHERE   ((A.Work_order_id = W.Work_order_id) AND (A.Mechanic_id =
@Employee_id)) OR
                    ((B.Work_order_id = W.Work_order_id) AND (B.Clerk_id =
@Employee_id))
        END
    END
```

```
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getInvoice]
    @Invoice_id INT,
    @Employee_id INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM employee_invoice E
                    WHERE E.Employee_id=@Employee_id AND
E.Invoice_id=@Invoice_id)
        BEGIN
            SELECT * FROM employee_invoice
            WHERE Invoice_id = @Invoice_id AND
            Employee_id = @Employee_id
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getInvoices]
    @Employee_id INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM employee_invoice E
                    WHERE E.Employee_id=@Employee_id)
        BEGIN
            SELECT * FROM employee_invoice
            WHERE Employee_id = @Employee_id
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getManagersDelagates]
    @Manager_id INT
AS
    BEGIN
        IF EXISTS (SELECT e.Employee_id FROM employee as e
                    WHERE e.Manager_id = @Manager_id)
        BEGIN
            SELECT e.Employee_id FROM employee as e
            WHERE e.Manager_id = @Manager_id
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getPart]
    @Part_id INT,
    @Part_instance_no INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM part P
                    WHERE P.Part_id = @Part_id AND
                    P.Part_instance_no = @Part_instance_no)
        BEGIN
            SELECT * FROM part P
            WHERE P.Part_id = @Part_id AND
            P.Part_instance_no = @Part_instance_no
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getPartsOnStock]
AS
```

```
    BEGIN
        SELECT * FROM part P
        WHERE (P.Work_order_id IS NULL) OR (P.Work_order_id = 0)
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getVehicle]
    @VIN VARCHAR(17)
AS
    BEGIN
        IF EXISTS (SELECT * FROM vehicle V
                    WHERE V.VIN = @VIN)
        BEGIN
            SELECT * FROM vehicle V
            WHERE V.VIN = @VIN
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getWorkOrder]
    @Work_order_id INT
AS
    BEGIN
        IF EXISTS (SELECT * from work_order W
                    WHERE W.Work_order_id = @Work_order_id)
        BEGIN
            SELECT * FROM work_order W
            WHERE W.Work_order_id = @Work_order_id
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[getWorkOrderParts]
    @Work_order_id INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM part P
                    WHERE P.Work_order_id = @Work_order_id)
        BEGIN
            SELECT * FROM part P
            WHERE P.Work_order_id = @Work_order_id
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[payEmployee]
    @Employee_id INT,
    @Amount DECIMAL(10,2),
    @Interval_Start_Date DATE,
    @Interval_End_Date DATE,
    @Payment_Date DATE,
    @WHours DECIMAL(10,2)
AS
    BEGIN
        IF NOT EXISTS (SELECT * FROM employee_invoice
                        WHERE Employee_id = @Employee_id AND
                            Interval_Start_Date = @Interval_Start_Date AND
                            Interval_End_Date = @Interval_End_Date)
        BEGIN
            INSERT INTO employee_invoice(Employee_id, Amount,
Interval_Start_Date, Interval_End_Date, Payment_Date, WHours)
```

```
                VALUES (@Employee_id, @Amount, @Interval_Start_Date,
@Interval_End_Date, @Payment_Date, @WHours)
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[removeCatalogPart]
    @Part_id VARCHAR(32)

AS
    BEGIN
        IF EXISTS (SELECT * FROM catalog_part
                    WHERE Part_id = @Part_id)
        BEGIN
            DELETE FROM catalog_part
            WHERE (Part_id = @Part_id)
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[removeVehicleModel]
    @Vehicle_make VARCHAR(32),
    @Vehicle_model VARCHAR(32),
    @Vehicle_year INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM vehicle_model V
                    WHERE V.Make = @Vehicle_make AND
                            V.Model = @Vehicle_model AND
                            V.VYear = @Vehicle_year)
        BEGIN
            DELETE FROM vehicle_model
                WHERE (Make = @Vehicle_make AND
                        Model = @Vehicle_model AND
                        VYear = @Vehicle_year)
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[removeWorkOrder]
    @Work_order_id INT
AS
    BEGIN
        IF EXISTS (Select * from work_order w
                    WHERE w.Work_order_id = @Work_order_id)
        BEGIN
            DELETE FROM work_order
                WHERE (Work_order_id = @Work_order_id)
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[searchCompatiblePart]
    @Vehicle_make       VARCHAR(32),
    @Vehicle_model      VARCHAR(32),
    @Vehicle_year       INT,
    @Part_name_string   VARCHAR(256)
AS
    BEGIN
        IF EXISTS ( SELECT cat.Part_id, cat.Part_name
                    FROM catalog_part as cat, compatible_with as com
```

```
                    WHERE (com.Part_id = cat.Part_id AND
com.Vehicle_make=@Vehicle_make AND com.Vehicle_model=@Vehicle_model
                        AND com.Vehicle_year=@Vehicle_year AND cat.Part_name
LIKE '%' + @Part_name_string + '%'))
        BEGIN
            SELECT cat.Part_id, cat.Part_name
            FROM catalog_part as cat, compatible_with as com
            WHERE (com.Part_id = cat.Part_id AND
com.Vehicle_make=@Vehicle_make AND com.Vehicle_model=@Vehicle_model
                        AND com.Vehicle_year=@Vehicle_year AND cat.Part_name
LIKE '%' + @Part_name_string + '%')
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[updateCustomer]
    @CustomerID     INT,
    @Fname          VARCHAR(128),
    @Phone_number   VARCHAR(15),
    @Lname          VARCHAR(128),
    @Address        VARCHAR(256)
AS
    BEGIN
        IF EXISTS (SELECT * FROM customer WHERE CustomerID = @CustomerID)
        BEGIN
            UPDATE customer
            SET     Fname = @Fname,
                    Lname = @Lname,
                    CAddress = @Address,
                    Phone_number = @Phone_number
            WHERE
                CustomerID = @CustomerID
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[updateEmployeeInfo]
    @Employee_id INT,
    @EPassword VARCHAR(128),
    @Bank_acc_no BIGINT,
    @EAddress VARCHAR(256),
    @Lname VARCHAR(128),
    @Fname VARCHAR (128),
    @Pay_type BIT,
    @Salary_rate DECIMAL(10,2),
    @Hourly_rate DECIMAL(10,2),
    @MecFlag BIT,
    @CFlag BIT,
    @ManFlag BIT,
    @AFlag BIT,
    @Manager_id INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM employee
                    WHERE Employee_id = @Employee_id)
        BEGIN
            IF(@Manager_id < 1)
            BEGIN
                UPDATE employee
```

```
                SET Bank_acc_no = @Bank_acc_no,
                    EPassword = @EPassword,
                    EAddress = @EAddress,
                    Lname = @Lname,
                    Fname = @Fname,
                    Pay_type = @Pay_type,
                    Salary_rate = @Salary_rate,
                    Hourly_rate = @Hourly_rate,
                    MecFlag = @MecFlag,
                    CFlag = @CFlag,
                    ManFlag = @ManFlag,
                    AFlag = @AFlag,
                    Manager_id = NULL
                WHERE Employee_id = @Employee_id
            END
            ELSE
            BEGIN
                UPDATE employee
                SET Bank_acc_no = @Bank_acc_no,
                    EPassword = @EPassword,
                    EAddress = @EAddress,
                    Lname = @Lname,
                    Fname = @Fname,
                    Pay_type = @Pay_type,
                    Salary_rate = @Salary_rate,
                    Hourly_rate = @Hourly_rate,
                    MecFlag = @MecFlag,
                    CFlag = @CFlag,
                    ManFlag = @ManFlag,
                    AFlag = @AFlag,
                    Manager_id = @Manager_id
                WHERE Employee_id = @Employee_id
            END
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[updatePart]
    @Part_id INT,
    @Part_instance_no INT,
    @PState VARCHAR(32),
    @Price DECIMAL(10,2),
    @Work_order_id INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM part P
                    WHERE P.Part_id = @Part_id AND P.Part_instance_no =
@Part_instance_no)
        BEGIN
            IF(@Work_order_id < 1)
            BEGIN
                UPDATE part
                SET     PState = @PState,
                        Price = @Price,
                        Work_order_id = NULL
                WHERE Part_id = @Part_id AND Part_instance_no =
@Part_instance_no
            END
```

```
            ELSE
            BEGIN
                UPDATE part
                SET     PState = @PState,
                        Price = @Price,
                        Work_order_id = @Work_order_id
                WHERE Part_id = @Part_id AND Part_instance_no =
@Part_instance_no
            END
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[updateVehicle]
    @VIN                VARCHAR(17),
    @Vehicle_make       VARCHAR(32),
    @Vehicle_model      VARCHAR(32),
    @Vehicle_year       INT,
    @Color              VARCHAR(32),
    @CustomerID         INT,
    @Registration_No    VARCHAR(20)
AS
    BEGIN
        IF EXISTS (SELECT * FROM vehicle V WHERE V.VIN = @VIN)

        BEGIN
            UPDATE vehicle
            SET Vehicle_make = @Vehicle_make,
                Vehicle_model = @Vehicle_model,
                Vehicle_year = @Vehicle_year,
                Color = @Color,
                CustomerID = @CustomerID,
                Registration_No = @Registration_No
            WHERE VIN = @VIN
        END
    END
-----------------------------------------------------------------------------
CREATE PROCEDURE [dbo].[updateWorkOrder]
    @Work_order_id INT,
    @Closed BIT,
    @Amount_due DECIMAL(10,2),
    @Vehicle_VIN VARCHAR(17),
    @CustomerID INT
AS
    BEGIN
        IF EXISTS (SELECT * FROM work_order W
                    WHERE W.Work_order_id = @Work_order_id)
        BEGIN
            UPDATE work_order
            SET Closed = @Closed,
                Amount_Due = @Amount_due,
                Vehicle_VIN = @Vehicle_VIN,
                CustomerID = @CustomerID
            WHERE
                Work_order_id = @Work_order_id
        END
    END
-----------------------------------------------------------------------------
```

API Documentation

For the professional API documentation created with Postman, please follow the following link or refer to Appendix 1:

https://documenter.getpostman.com/view/13510045/TVmTbaAP

User Guide

Prerequisites:

Software: Microsoft SQL Sever Mangement (2019), Microsoft Visual Studio (2019)

An instance of SQL Sever running

Step 1: Download archived file from D2L.

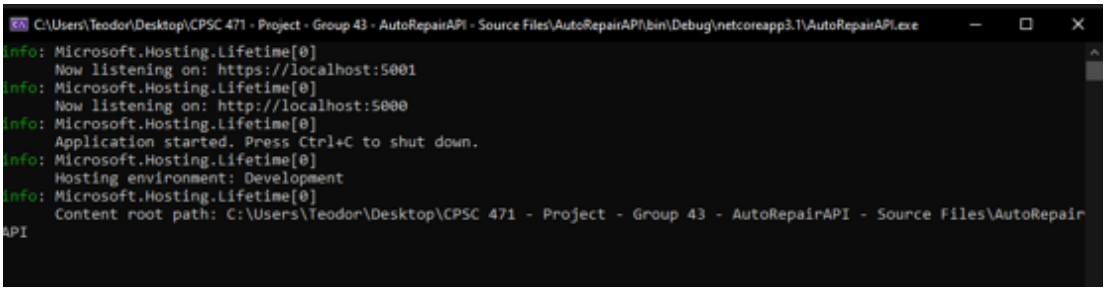
Step 2: Open MS SQL Server Mgmt and connect to local server
Open & Run generation file (*AutoRepairGenerationFile.sql*) in SQL Server Mgmt Studio

Step 3: Open & Run database population file (*AutoRepairPopulateFile.sql*) – ignore any errors)

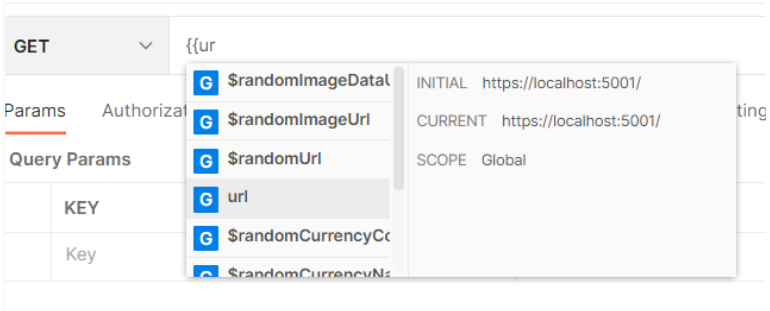
Step 4: Open Visual Studio → "Open a project or solution" → *AutoRepairAPI.sln*

Step 5: Open *AutoRepairAPIController.cs* & ensure all packages in '*using*' statements work
→ If not working, install required dependencies

Step 6: Run *AutoRepairAPI*
→ Window should look something like this:



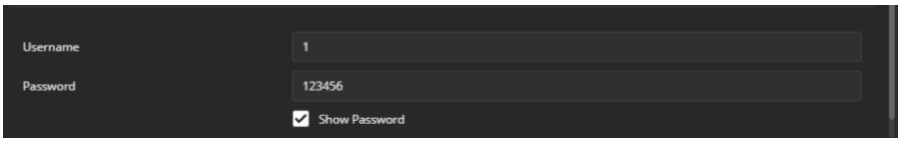
Step 7: Open Postman (follow this initial url path):



Step 8: If required, Disable SSL certification

→ If you get *CORS Error*, try using the Postman application

Step 9: Try using the following credentials:



→ Username options:

- '1' – Manager
- '4' – Admin
- '5' – Mechanic
- '22' – Clerk

→ Password: '123456'

References

https://www.rockauto.com/ - An existing web-application for buying car parts.

https://www.autorepairbill.com/ – A nicely laid out and mostly easy to navigate web-application for buying car parts.

https://www.shopmonkey.io/ - smart and simple shop management software

Appendix 1 – API Documentation Printout

AutoRepairAPI

Introduction

The AutoRepairAPI delivers the user's request to the provider -- a Microsoft SQL Server Database, and then delivers the response back to you.

Overview

The functionalities mostly pertain to the database's stored procedures and include but are not limited to the following: adding and viewing employees, adding and viewing customers and work orders, removing work orders, checking part compatibility based on a specific vehicle model, viewing the employee id's a manager is delegating, searching compatible parts, updating info on customer, employee, work order, vehicle.

Usage

Local testing purposes and showcasing the backend capabilities of our Autorepair Database. Each request requires a basic authorization login with the respective employee in the employee table. The database is configured to use Microsoft SQL Server. Development was used in conjunction with Microsoft SQL Server Management Studio and Microsoft Visual Studio 2019.

Error Codes

Our input error checking is done through the backend, as our procedures first check the database model for the given input and if it does not match the expected format, the database will not perform the procedure. Our API will also check for generic errors before the request is even sent to the database: for example, whether or not the Employee_id entered is greater than 0. If it is not greater than 0, the user can expect a generic error code to be returned, letting them know they need to pick an employee_id from the correct range. All data restrictions and specifications can be found in appendix 2 of the final report.

Example Error Codes

A user can expect to see the following error codes if they are trying to interact with the database wrong:

- Error 100: Tuple does not exist in the database
- Error 200: Attempting to insert an invalid FK
- Error 300: Attempting to assign a null or invalid PK
- Error 4XX: System logical error (eg, assign a mechanic to the associated table)

- Error 500: Invalid data value
- Error 600: Invalid Login
- Error 700: Authentication of login rejected

Documentation Settings

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

GET getEmployee

{{url}}api/AutoRepairAPI/getEmployee/{{Employee_id}}

The type is GET. The expected parameters are {Employee_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or the login id must match the {Employee_id}. A successful response contains a employee JSON object for the respective {Employee_id}.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Example Request getEmployee

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getEmployee/5'
```

Example Response 200 OK

Body Headers (4)

Documentation Settings


```
{
  "Employee_id": 5,
  "Password": "123456",
  "Manager_id": 1,
  "Bank_acc_no": 3552807951599814,
  "Address": "79 Bayside Way",
  "Lname": "Conan",
  "Fname": "Enrichetta",
  "Salary_rate": 0,
  "Hourly_rate": 58.99,
```


View More

POST addEmployee

{{url}}api/AutoRepairAPI/addEmployee

The type is POST. The expected parameters are an employee JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager or admin. A successful response will notify the user with the success string: "Employee Successfully Entered". NOTE: The {Employee_id} used is a dummy variable and will not be assigned to the employee. An autogenerated id will be assigned by the SQL Server database.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
    "Employee_id" : {{Employee_id}},
    "Password": {{Password}},
    "Fname" : {{Fname}},
    "Lname" : {{Lname}},
    "Address" : {{Address}},
    "Bank_acc_no" : {{Bank_acc_no}},
    "Salary Rate" : {{Salary_Rate}},
    "Hourly_rate" : {{Hourly_Rate}},
    "Pay_Type" : {{Pay_Type}},
```

Documentation Settings

View More

Example Request

addEmployee

```
curl --location --request POST '{{url}}api/AutoRepairAPI/addEmployee' \
--data-raw '{
    "Employee_id" : 0,
    "Password": "123456",
    "Fname" : "Jeffrey",
    "Lname" : "Layton",
    "Address" : "153 Upsidedown Street",
    "Bank_acc_no" : 374622246502396,
    "Salary Rate" : null,
    "Hourly_rate" : 0.05,
```

View More

Example Response

200 OK

Body Headers (4)

```
Employee Successfully Entered
```

POST payEmployee

{{url}}api/AutoRepairAPI/payEmployee

The type is POST. The expected parameters are an invoice JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager or admin. A successful response will notify the user with the success string: "Invoice Successfully Entered". NOTE: The {Invoice_id} used is a dummy variable and will not be assigned to the invoice. An autogenerated id will be assigned by the SQL Server.

AUTHORIZATION

Basic Auth

Documentation Settings

Username

<username>

Password

<password>

BODY raw

```
{
    "Invoice_id" : {{Invoice_id}},
    "Employee_id" : {{Employee_id}},
    "Amount" : {{Amount}},
    "Interval_Start_Date" : {{Interval_Start_Date}},
    "Interval_End_Date" : {{Interval_End_Date}},
    "Payment_Date" : {{Payment_Date}},
    "Hours" : {{Hours}}
}
```

Example Request

payEmployee

```
curl --location --request POST '{{url}}api/AutoRepairAPI/payEmployee' \
--data-raw '{
    "Invoice_id" : 0,
    "Employee_id" : 11,
    "Amount" : 2222.22,
    "Interval_Start_Date" : "2014-01-01",
    "Interval_End_Date" : "2014-01-01",
    "Payment_Date" : "2014-01-01",
    "Hours" : 32.9
}'
```

Example Response

200 OK

Body Headers (4)

```
Invoice Successfully Entered
```

PUT updateEmployeeInfo

{{url}}api/AutoRepairAPI/updateEmployeeInfo

Documentation Settings

The type is PUT. The expected parameters are an employee JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager or admin. A successful response will notify the user with the success string: "Employee Info Successfully Updated".

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
    "Employee_id" : {{Employee_id}},
    "Password": {{Password}},
    "Fname" : {{Fname}},
    "Lname" : {{Lname}},
    "Address" : {{Address}},
    "Bank_acc_no" : {{Bank_acc_no}},
    "Salary Rate" : {{Salary_Rate}},
    "Hourly_rate" : {{Hourly_Rate}},
    "Pay_Type" : {{Pay_Type}},
```

View More

Example Request

updateEmployeeInfo

```
curl --location --request PUT '{{url}}api/AutoRepairAPI/updateEmployeeInfo' \
--data-raw '{
    "Employee_id" : 11,
    "Password": "123456",
    "Fname" : "Jeff",
    "Lname" : "Layton",
    "Address" : "153 Upsidedown Street",
    "Bank_acc_no" : 374622246502396,
    "Salary Rate" : null,
    "Hourly_rate" : 0.05,
```

View More

Example Response

200 OK

Body Headers (4)

Documentation Settings

```
Employee Info Successfully Updated
```

GET getInvoices

{{url}}api/AutoRepairAPI/getInvoices/{{Employee_id}}

The type is GET. The expected parameters are {Employee_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or the login id must match the {Employee_id}. A successful response contains an array of invoice JSON objects for the respective {Employee_id}.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Example Request getInvoices

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getInvoices/19'
```

Example Response 200 OK

Body Headers (4)

Documentation Settings


```
[
  {
    "Invoice_id": 1,
    "Employee_id": 19,
    "Amount": 2073.36,
    "Interval_Start_Date": "2020-07-08T00:00:00",
    "Interval_End_Date": "2020-07-22T00:00:00",
    "Payment_Date": "2020-07-29T00:00:00",
    "Hours": 54
  },
```


View More

GET getInvoice

{{url}}api/AutoRepairAPI/getInvoice/{{Employee_id}}/{{Invoice_id}}

The type is GET. The expected parameters are {Employee_id} and {Invoice_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or the login id must match the {Employee_id}. A successful response contains an employee JSON object for the respective {Employee_id} and {Invoice_id}.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Example Request getInvoice

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getInvoice/19/1'
```

Example Response 200 OK

Body Headers (4)

```
{
  "Invoice_id": 1,
  "Employee_id": 19,
  "Amount": 2073.36,
  "Interval_Start_Date": "2020-07-08T00:00:00",
  "Interval_End_Date": "2020-07-22T00:00:00",
  "Payment_Date": "2020-07-29T00:00:00",
  "Hours": 54
}
```


GET getManagersDelegates

{{url}}api/AutoRepairAPI/getManagersDelagates/{{Manager_id}}

The type is GET. The expected parameters are {Manager_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager or admin. A successful response contains an array of employee ids for the respective {Manager_id}.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Example Request getManagersDelegates

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getManagersDelagates/1'
```

Example Response 200 OK

Body Headers (4)

```
[
  5,
  8,
  14,
  19,
  22,
  25,
  28,
  30
]
```


POST assignMechanic

{{url}}api/AutoRepairAPI/assignMechanic

The type is POST. The expected parameters are an assigned_to JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or a clerk. A successful response will notify the user with the success string: "Mechanic Successfully Assigned".

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
    "Mechanic_id" : {{Mechanic_id}},
    "Work_order_id" : {{Work_order_id}}
}
```

Example Request

```
curl --location --request POST '{{url}}api/AutoRepairAPI/assignMechanic' \
--data-raw '{
    "Mechanic_id" : 10,
    "Work_order_id" : 1
}'
```

Documentation Settings

Example Response 200 OK

Body Headers (4)

```
Mechanic Successfully Assigned
```

POST assignClerk

{{url}}api/AutoRepairAPI/assignClerk

The type is POST. The expected parameters are an associated_with JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or a clerk. A successful response will notify the user with the success string: "Successfully Added Clerk Association".

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
    "Clerk_id" : 28,
    "Work_order_id" : 1
}
```

Example Request

assignClerk

Documentation Settings

```
curl --location --request POST '{{url}}api/AutoRepairAPI/assignClerk' \
--data-raw '{
    "Clerk_id" : 28,
    "Work_order_id" : 1
}'
```

Example Response 200 OK

Body Headers (4)

```
Successfully Added Clerk Association
```

POST createWorkOrder

{{url}}api/AutoRepairAPI/createWorkOrder

The type is POST. The expected parameters are a work order JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or a clerk. A successful response will notify the user with the success string: "Successfully Created Work Order". NOTE: The {Work_order_id} used is a dummy variable and will not be assigned to the work order. Instead, an autogenerated id will be assigned by the SQL server.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
    "Work_order_id" : {{Work_order_id}},
    "Closed" : {{Closed}},
    "Amount_Due" : {{Amount_Due}},
    "Vehicle_VIN" : {{Vehicle_VIN}},
    "CustomerID" : {{CustomerID}}
}
```

Documentation Settings

Example Request

createWorkOrder

```
curl --location --request POST '{{url}}api/AutoRepairAPI/createWorkOrder' \
--data-raw '{
    "Work_order_id" : 0,
    "Closed" : false,
    "Amount_Due" : 1111111.22,
    "Vehicle_VIN" : "1G6AY5S39E0782777",
    "CustomerID" : 6
}'
```

Example Response

200 OK

Body Headers (4)

```
Successfully Created Work Order
```

PUT updateWorkOrder

{{url}}api/AutoRepairAPI/updateWorkOrder

The type is PUT. The expected parameters are a work order JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or a clerk. A successful response will notify the user with the success string: "Successfully Updated Work Order".

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Documentation Settings

BODY raw

```
{
    "Work_order_id" : {{Work_order_id}},
    "Closed" : {{Closed}},
    "Amount_Due" : {{Amount_Due}},
    "Vehicle_VIN" : {{Vehicle_VIN}},
    "CustomerID" : {{CustomerID}}
}
```

Example Request updateWorkOrder

```
curl --location --request PUT '{{url}}api/AutoRepairAPI/updateWorkOrder' \
--data-raw '{
    "Work_order_id" : 5,
    "Closed" : false,
    "Amount_Due" : 1111111.22,
    "Vehicle_VIN" : "WAUA2AFD2DN666409",
    "CustomerID" : 5
}'
```

Example Response 200 OK

Body Headers (4)

```
Successfully Updated Work Order
```

GET getCustomerWorkOrders

{{url}}api/AutoRepairAPI/getCustomerWorkOrders/{{CustomerID}}

The type is GET. The expected parameters are {CustomerID} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or clerk. A successful response contains an array of work order JSON objects for the respective {CustomerID}.

AUTHORIZATION

Basic Auth

Documentation Settings

Username

<username>

Password

<password>

Example Request

getCustomerWorkOrders

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getCustomerWorkOrders/6'
```

Example Response

200 OK

Body Headers (4)


```
[
  {
    "Work_order_id": 6,
    "Closed": true,
    "Amount_Due": 3071.95,
    "Vehicle_VIN": "1G6AY5S39E0782777",
    "CustomerID": 6
  },
  {
    "Work_order_id": 15,
```


View More

GET getWorkOrder

{{url}}api/AutoRepairAPI/getWorkOrder/{{Work_order_id}}

The type is GET. The expected parameters are {Work_order_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must be any valid employee. A successful response contains a work order JSON object for the respective {Work_order_id}.

AUTHORIZATION

Basic Auth

Username

<username>

Documentation Settings

Password

<password>

Example Request getWorkOrder

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getWorkOrder/6'
```

Example Response 200 OK

Body Headers (4)

```
{
  "Work_order_id": 6,
  "Closed": true,
  "Amount_Due": 3071.95,
  "Vehicle_VIN": "1G6AY5S39E0782777",
  "CustomerID": 6
}
```

GET getEmployeeWorkOrders

{{url}}api/AutoRepairAPI/getEmployeeWorkOrders/{{Employee_id}}

The type is GET. The expected parameters are {Employee_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must be any valid employee. A successful response contains an array of work order JSON objects for the respective {Employee_id}.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Documentation Settings

Example Request getEmployeeWorkOrders

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getEmployeeWorkOrders/21'
```

Example Response 200 OK

Body Headers (4)


```
[
  {
    "Work_order_id": 1,
    "Closed": true,
    "Amount_Due": 458.61,
    "Vehicle_VIN": "JN8AF5MR0BT743790",
    "CustomerID": 1
  },
  {
    "Work_order_id": 2,
```


View More

DEL removeWorkOrder

{{url}}api/AutoRepairAPI/removeWorkOrder/{{Work_order_id}}

The type is DELETE. The expected parameters are {Work_order_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager or admin. A successful response will notify the user with the success string: "Successfully Removed Work Order".

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Example Request

removeWorkOrder

Documentation Settings

```
curl --location --request DELETE '{{url}}api/AutoRepairAPI/removeWorkOrder/5'
```

Example Response

200 OK

Body Headers (4)

```
Successfully Removed Work Order
```

POST addCustomer

{{url}}api/AutoRepairAPI/addCustomer

The type is POST. The expected parameters are a customer JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or a clerk. A successful response will notify the user with the success string: "Successfully Added Customer". NOTE: The {CustomerID} used is a dummy variable and will not be assigned to the customer. An autogenerated id will be assigned by the SQL server.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
    "CustomerID" : {{CustomerID}},
    "Fname" : {{Fname}},
    "Lname" : {{Lname}},
    "Phone_number" : {{Phone_number}},
    "Address" : {{Address}}
}
```

Example Request

addCustomer

Documentation Settings

```
curl --location --request POST '{{url}}api/AutoRepairAPI/addCustomer' \
--data-raw '{
    "CustomerID" : 0,
    "Fname" : "Paige",
    "Lname" : "Hess",
    "Phone_number" : "430-999-8594",
    "Address" : "Who Knows Honestly Lane"
}'
```

Example Response 200 OK

Body Headers (4)

```
Successfully Added Customer
```

PUT updateCustomer

{{url}}api/AutoRepairAPI/updateCustomer

The type is PUT. The expected parameters are a customer JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or a clerk. A successful response will notify the user with the success string: "Successfully Updated Customer".

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
    "CustomerID" : {{CustomerID}},
    "Fname" : {{Fname}},
    "Lname" : {{Lname}},
    "Phone_number" : {{Phone_number}},
    "Address" : {{Address}}
}
```

Documentation Settings

Example Request

updateCustomer

```
curl --location --request PUT '{{url}}api/AutoRepairAPI/updateCustomer' \
--data-raw '{
    "CustomerID" : 10,
    "Fname" : "Harman",
    "Lname" : "Deacon Jr.",
    "Phone_number" : "430-900-8594",
    "Address" : "NORTH POLE"
}'
```

Example Response

200 OK

Body Headers (4)

```
Successfully Updated Customer
```

GET getCustomer

{{url}}api/AutoRepairAPI/getCustomer/{{CustomerID}}

The type is GET. The expected parameters are {CustomerID} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or clerk. A successful response contains a customer JSON objects for the respective {CustomerID}.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Documentation Settings

Example Request

getCustomer

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getCustomer/11'
```

Example Response

200 OK

Body Headers (4)

```
{
  "CustomerID": 11,
  "Fname": "Alexia",
  "Lname": "Sayse",
  "Phone_number": "593-445-2973",
  "Address": "643 Eastwood Avenue"
}
```

GET getCustomerVehicles

{{url}}api/AutoRepairAPI/getCustomerVehicles/{{CustomerID}}

The type is GET. The expected parameters are {CustomerID} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or clerk. A successful response contains an array of vehicle JSON objects for the respective {CustomerID}.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Documentation Settings

Example Request getCustomerVehicles

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getCustomerVehicles/1'
```

Example Response 200 OK

Body Headers (4)


```
[
  {
    "VIN": "JN8AF5MR0BT743790",
    "Vehicle_model": "Esprit",
    "Vehicle_make": "Lotus",
    "Vehicle_year": 1996,
    "CustomerID": 1,
    "Color": "Yellow",
    "Registration_No": "4060740594"
  },
```


View More

POST addVehicle

{{url}}api/AutoRepairAPI/addVehicle

The type is POST. The expected parameters are a vehicle JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or a clerk. A successful response will notify the user with the success string: "Successfully Added Vehicle".

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
    "VIN": {{VIN}},
    "Vehicle_model": {{Vehicle_model}},
    "Vehicle_make": {{Vehicle_make}},
    "Vehicle_year": {{Vehicle_year}},
    "CustomerID": {{CustomerID}},
    "Color": {{Color}},
    "Registration_No": {{Registration_No}}
}
```

Documentation Settings

Example Request addVehicle

```
curl --location --request POST '{{url}}api/AutoRepairAPI/addVehicle' \
--data-raw '    {
        "VIN": "111111111111211",
        "Vehicle_model": "Z4 M",
        "Vehicle_make": "BMW",
        "Vehicle_year": 2006,
        "CustomerID": 6,
        "Color": "Invisible",
        "Registration_No": "55537268"
    }'
```

Example Response 200 OK

Body Headers (4)

```
Successfully Added Vehicle
```

PUT updateVehicle

{{url}}api/AutoRepairAPI/updateVehicle

The type is PUT. The expected parameters are a vehicle JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or a clerk. A successful response will notify the user with the success string: "Successfully Updated Vehicle".

AUTHORIZATION

Basic Auth

Username

<username>

Documentation Settings

Password

<password>

BODY raw

```
    {
        "VIN": {{VIN}},
        "Vehicle_model": {{Vehicle_model}},
        "Vehicle_make": {{Vehicle_make}},
        "Vehicle_year": {{Vehicle_year}},
        "CustomerID": {{CustomerID}},
        "Color": {{Color}},
        "Registration_No": {{Registration_No}}
    }
```

Example Request

updateVehicle

```
curl --location --request PUT '{{url}}api/AutoRepairAPI/updateVehicle' \
--data-raw '    {
        "VIN": "1G6AY5S39E0782777",
        "Vehicle_model": "MPV",
        "Vehicle_make": "Mazda",
        "Vehicle_year": 1994,
        "CustomerID": 6,
        "Color": "YELLOW",
        "Registration_No": "4491953856"
    }'
```

Example Response

200 OK

Body Headers (4)

```
Successfully Updated Vehicle
```

GET getVehicle

{{url}}api/AutoRepairAPI/getVehicle/{{VIN}}

The type is GET. The expected parameters are {VIN} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a valid employee. A successful response contains a vehicle JSON object for the respective {VIN}.

Documentation Settings

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Example Request getVehicle

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getVehicle/1G6AY5S39E0782777'
```

Example Response 200 OK

Body Headers (4)

```
{
  "VIN": "1G6AY5S39E0782777",
  "Vehicle_model": "MPV",
  "Vehicle_make": "Mazda",
  "Vehicle_year": 1994,
  "CustomerID": 6,
  "Color": "YELLOW",
  "Registration_No": "4491953856"
}
```

GET checkVehicleModel

{{url}}api/AutoRepairAPI/checkVehicleModel

The type is GET. The expected parameters are a vehicle model JSON object in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager or admin. A successful response contains a Boolean (T/F) value.

AUTHORIZATION

Basic Auth

Documentation Settings

Username

<username>

Password

<password>

BODY raw

```
{
        "Vehicle_model": {{Vehicle_model}},
        "Vehicle_make": {{Vehicle_make}},
        "Year": {{Year}}
}
```

Example Request

checkVehicleModel

```
curl --location --request GET '{{url}}api/AutoRepairAPI/checkVehicleModel' \
--data-raw '{
        "Vehicle_model": "Z4 M",
        "Vehicle_make": "BMW",
        "Year": 2006
}'
```

Example Response

200 OK

Body Headers (4)

```
true
```

POST addVehicleModel

{{url}}api/AutoRepairAPI/addVehicleModel

The type is POST. The expected parameters are a vehicle model JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match an admin. A successful response will notify the user with the success string: "Successfully

Added Vehicle Model".

Documentation Settings

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
        "Vehicle_model": {{Vehicle_model}},
        "Vehicle_make": {{Vehicle_make}},
        "Year": {{Year}}
}
```

Example Request addVehicleModel

```
curl --location --request POST '{{url}}api/AutoRepairAPI/addVehicleModel' \
--data-raw '{
        "Vehicle_model": "THE MODEL",
        "Vehicle_make": "THE MAKE",
        "Year": 202020
}'
```

Example Response 200 OK

Body Headers (4)

```
Successfully Added Vehicle Model
```

DEL removeVehicleModel

{{url}}api/AutoRepairAPI/removeVehicleModel

The type is DELETE. The expected parameters are a vehicle model JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match an admin. A successful response will notify the user with the success string: "Successfully Removed Vehicle Model".

Documentation Settings

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
        "Vehicle_model": {{Vehicle_model}},
        "Vehicle_make": {{Vehicle_make}},
        "Year": {{Year}}
}
```

Example Request removeVehicleModel

```
curl --location --request DELETE '{{url}}api/AutoRepairAPI/removeVehicleModel' \
--data-raw '{
        "Vehicle_model": "Jimmy",
        "Vehicle_make": "GMC",
        "Year": 1999
}'
```

Example Response 200 OK

Body Headers (4)

```
Successfully Removed Vehicle Model
```

GET getWorkOrderParts

{{url}}api/AutoRepairAPI/getWorkOrderParts/{{Work_order_id}}

Documentation Settings

The type is GET. The expected parameters are {Work_order_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a valid employee. A successful response contains an array of part JSON objects for the respective {Work_order_id}.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Example Request getWorkOrderParts

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getWorkOrderParts/14'
```

Example Response 200 OK

Body Headers (4)


```
[
  {
    "Part_id": 14,
    "Part_instance_id": 1,
    "State": "1",
    "Price": 666.65,
    "Work_order_id": 14
  },
  {
    "Part_id": 14,
```


View More

GET getPartsOnStock

{{url}}api/AutoRepairAPI/getPartsOnStock

The type is GET. There are no expected parameters. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or mechanic. A successful response contains an array of part JSON objects that are not assigned to a work order,

Documentation Settings

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Example Request getPartsOnStock

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getPartsOnStock'
```

Example Response 200 OK

Body Headers (4)

```
[
  {
    "Part_id": 8,
    "Part_instance_id": 1,
    "State": "1",
    "Price": 1237.77,
    "Work_order_id": 0
  },
  {
    "Part_id": 14,
```

View More

PUT updatePart

{{url}}api/AutoRepairAPI/updatePart

The type is PUT. The expected parameters are a part JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or mechanic. A successful response will notify the user with the success string: "Successfully Updated Part".

Documentation Settings

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
    {
        "Part_id": {{Part_id}},
        "Part_instance_id": {{Part_instance_id}},
        "State": {{State}},
        "Price": {{Price}},
        "Work_order_id": {{Work_order_id}}
    }
```

Example Request updatePart

```
curl --location --request PUT '{{url}}api/AutoRepairAPI/updatePart' \
--data-raw '    {
        "Part_id": 14,
        "Part_instance_id": 5,
        "State": "1",
        "Price": 0.65,
        "Work_order_id": 0
    }'
```

Example Response 200 OK

Body Headers (4)

```
Successfully Updated Part
```

GET getPart

{{url}}api/AutoRepairAPI/getPart/{Part_id}/{Part_instance_id}

Documentation Settings

The type is GET. The expected parameters are {Part_id} and {Part_instance_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or mechanic. A successful response contains a part JSON object for the respective {Part_id} and {Part_instance_id}.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Example Request getPart

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getPart/14/1'
```

Example Response 200 OK

Body Headers (4)

```
{
  "Part_id": 14,
  "Part_instance_id": 1,
  "State": "1",
  "Price": 666.65,
  "Work_order_id": 14
}
```

POST addPart

{{url}}api/AutoRepairAPI/addPart

The type is POST. The expected parameters are a part JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or mechanic. A successful response will notify the user with the success string: "Successfully Added Part". NOTE: The {Part_instance_id} used is a dummy variable and will not be assigned to the part. An autogenerated id will be assigned by the SQL server.

Documentation Editor

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
    {
        "Part_id": {{Part_id}},
        "Part_instance_id": {{Part_instance_id}},
        "State": {{State}},
        "Price": {{Price}},
        "Work_order_id": {{Work_order_id}}
    }
```

Example Request addPart

```
curl --location --request POST '{{url}}api/AutoRepairAPI/addPart' \
--data-raw '    {
        "Part_id": 14,
        "Part_instance_id": 0,
        "State": "1",
        "Price": 63322.65,
        "Work_order_id": 0
    }'
```

Example Response 200 OK

Body Headers (4)

```
Successfully Added Part
```

GET searchCompatiblePart

Documentation Settings

{{url}}api/AutoRepairAPI/searchCompatiblePart/{{searhString}}

The type is GET. The expected parameters are a vehicle model in the body and a {searchString} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or mechanic. A successful response contains an array of catalog part JSON objects for the respective vehicle model and {searchString}.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
        "Vehicle_model": {{Vehicle_model}},
        "Vehicle_make": {{Vehicle_make}},
        "Year": {{Year}}
}
```

Example Request searchCompatiblePart

```
curl --location --request GET '{{url}}api/AutoRepairAPI/searchCompatiblePart/bal' \
--data-raw '{
        "Vehicle_model": "Azera",
        "Vehicle_make": "Hyundai",
        "Year": 2010
}'
```

Example Response 200 OK

Body Headers (4)

```
[
  {
    "Part_id": 1,
    "Part_name": "Balance Shaft Chain"
  },
  {
    "Part_id": 2,
    "Part_name": "Balance Shaft Chain Guide"
  }
]
```


GET getCatalogPart

{{url}}api/AutoRepairAPI/getCatalogPart/{{Part_id}}

The type is GET. The expected parameters are {Part_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or mechanic. A successful response contains a catalog part JSON object for the respective {Part_id}.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Example Request getCatalogPart

```
curl --location --request GET '{{url}}api/AutoRepairAPI/getCatalogPart/3'
```

Example Response 200 OK

Body Headers (4)

```
{
  "Part_id": 3,
  "Part_name": "Balance Shaft Chain Tensioner"
}
```

Documentation Settings

POST addCatalogPart

{{url}}api/AutoRepairAPI/addCatalogPart

The type is POST. The expected parameters are a part JSON object contained in the body. Authorization requires an employee id as the username and a password for the employee. The credentials must match an admin. A successful response will notify the user with the success string: "Successfully Added Catalog Part". NOTE: The {Part_id} used is a dummy variable and will not be assigned to the catalog part. An autogenerated id will be assigned by the SQL server.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
    "Part_id" : {{Part_id}},
    "Part_name" : {{Part_name}}
}
```

Example Request

addCatalogPart

```
curl --location --request POST '{{url}}api/AutoRepairAPI/addCatalogPart' \
--data-raw '{
    "Part_id" : -100,
    "Part_name" : "RARE PART THAT IS HONESTLY GARBAGE"
}'
```

Documentation Settings

Example Response 200 OK

Body Headers (4)

```
Successfully Added Catalog Part
```

DEL removeCatalogPart

{{url}}api/AutoRepairAPI/removeCatalogPart/{{Part_id}}

The type is DELETE. The expected parameters are {Part_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match an admin. A successful response will notify the user with the success string: "Successfully Deleted Catalog Part".

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

Example Request removeCatalogPart

```
curl --location --request DELETE '{{url}}api/AutoRepairAPI/removeCatalogPart/11'
```

Example Response 200 OK

Body Headers (4)

POST addCompatibility

{{url}}api/AutoRepairAPI/addCompatibility/2

The type is POST. The expected parameters are a vehicle model JSON object in the body and a {Part_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match an admin. A successful response will notify the user with the success string: "Successfully Added Compatibility".

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
        "Vehicle_model": {{Vehicle_model}},
        "Vehicle_make": {{Vehicle_make}},
        "Year": {{Year}}
}
```

Example Request

addCompatibility

```
curl --location --request POST '{{url}}api/AutoRepairAPI/addCompatibility/2' \
--data-raw '{
        "Vehicle_model": "Esprit",
        "Vehicle_make": "Lotus",
        "Year": 1996
}'
```

Example Response

200 OK

Documentation Settings

Body Headers (4)

```
Successfully Added Compatibility
```

GET checkPartCompatibility

{{url}}api/AutoRepairAPI/checkPartCompatibility/{{Part_id}}

The type is GET. The expected parameters are a vehicle model JSON object in the body and a {Part_id} in the URL. Authorization requires an employee id as the username and a password for the employee. The credentials must match a manager, admin, or mechanic. A successful response contains a Boolean (T/F) value.

AUTHORIZATION

Basic Auth

Username

<username>

Password

<password>

BODY raw

```
{
        "Vehicle_model": {{Vehicle_model}},
        "Vehicle_make": {{Vehicle_make}},
        "Year": {{Year}}
}
```

Example Request

checkPartCompatibility

```
curl --location --request GET '{{url}}api/AutoRepairAPI/checkPartCompatibility/1' \
--data-raw '{
        "Vehicle_model": "Esprit",
        "Vehicle_make": "Lotus",
        "Year": 1996
}'
```

Documentation Settings

Example Response 200 OK

Body Headers (4)

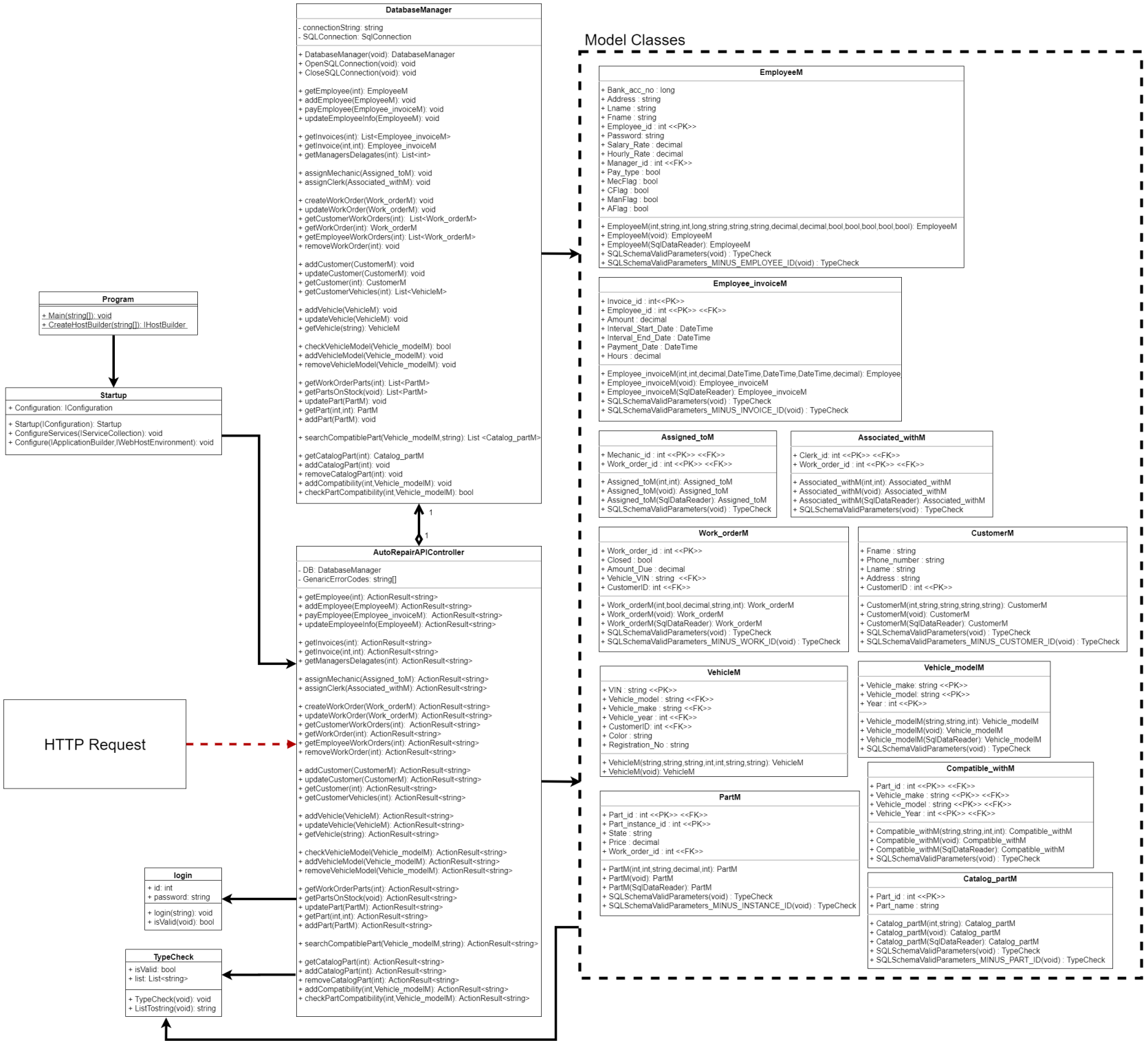
```
true
```

Appendix 2 – Input Table

Attribute	Type	Request Data Restriction
Assigned_toM		
Mechanic_id	int	Greater then 0
Work_order_id	int	Greater then 0
Associated_withM		
Clerk_id	int	Greater then 0
Work_order_id	int	Greater then 0
Catalog_partM		
Part_id	int	Greater then 0
Part_name	string	Null or [0,64] length
Catalog_partM		
Vehicle_model	string	[0,32] length
Vehicle_make	string	[0,32] length
Vehicle_year	int	N/A
Part_id	int	Greater then 0
CustomerM		
CustomerID	int	Greater then 0
Fname	string	Null or [0,128] length
Lname	string	Null or [0,128] length
Phone_number	string	Null or [0,15] length
Address	string	Null or [0,256] length
Employee_invoiceM		
Invoice_id	int	Greater then 0
Employee_id	int	Greater then 0
Amount	decimal	[-99999999.99 , 99999999.99]
Interval_Start_Date	DateTime	N/A
Interval_End_Date	DateTime	N/A
Payment_Date	DateTime	N/A
Hours	decimal	[-99999999.99 , 99999999.99]
EmployeeM		
Employee_id	int	Greater then 0
Password	string	[0,128] length
Manager_id	int	Greater then or equal to 0
Bank_acc_no	long	Greater then or equal to 0
Lname	string	Null or [0,128] length
Fname	string	Null or [0,128] length
Salary_rate	decimal	[-99999999.99 , 99999999.99]
Hourly_rate	decimal	[-99999999.99 , 99999999.99]
Pay_Type	bool	N/A
MecFlag	bool	N/A
CFlag	bool	N/A
ManFlag	bool	N/A
Aflag	bool	N/A
PartM		
Part_id	int	Greater then 0

Part_instance_id	int	Greater then 0
State	string	Null or [0,32] length
Price	decimal	[-99999999.99 , 99999999.99]
Work_order_id	int	Greater then or equal to 0
Vehicle_modelM		
Vehicle_model	string	[0,32] length
Vehicle_make	string	[0,32] length
Year	int	N/A
VehicleM		
VIN	string	[0,17] length
Vehicle_model	string	[0,32] length
Vehicle_make	string	[0,32] length
Year	int	N/A
CustomerID	int	Greater then 0
Color	string	Null or [0,32] length
Registration_No	string	Null or [0,20] length
PartM		
Work_order_id	int	Greater then 0
Closed	int	N/A
Amount_Due	string	[-99999999.99 , 99999999.99]
Vehicle_VIN	decimal	[0,17] length
CustomerID	int	Greater then 0

Appendix 3 – Functional Design Model (OO)


DatabaseManager
- connectionString: string
- SQLConnection: SqlConnection
+ DatabaseManager(void): DatabaseManager
+ OpenSQLConnection(void): void
+ CloseSQLConnection(void): void
+ getEmployee(int): EmployeeM
+ addEmployee(EmployeeM): void
+ payEmployee(Employee_invoiceM): void
+ updateEmployeeInfo(EmployeeM): void
+ getInvoices(int): List<Employee_invoiceM>
+ getInvoice(int,int): Employee_invoiceM
+ getManagersDelagates(int): List<int>
+ assignMechanic(Assigned_toM): void
+ assignClerk(Associated_withM): void
+ createWorkOrder(Work_orderM): void
+ updateWorkOrder(Work_orderM): void
+ getCustomerWorkOrders(int): List<Work_orderM>
+ getWorkOrder(int): Work_orderM
+ getEmployeeWorkOrders(int): List<Work_orderM>
+ removeWorkOrder(int): void
+ addCustomer(CustomerM): void
+ updateCustomer(CustomerM): void
+ getCustomer(int): CustomerM
+ getCustomerVehicles(int): List<VehicleM>
+ addVehicle(VehicleM): void
+ updateVehicle(VehicleM): void
+ getVehicle(string): VehicleM
+ checkVehicleModel(Vehicle_modelM): bool
+ addVehicleModel(Vehicle_modelM): void
+ removeVehicleModel(Vehicle_modelM): void
+ getWorkOrderParts(int): List<PartM>
+ getPartsOnStock(void): List<PartM>
+ updatePart(PartM): void
+ getPart(int,int): PartM
+ addPart(PartM): void
+ searchCompatiblePart(Vehicle_modelM,string): List <Catalog_partM>
+ getCatalogPart(int): Catalog_partM
+ addCatalogPart(int): void
+ removeCatalogPart(int): void
+ addCompatibility(int,Vehicle_modelM): void
+ checkPartCompatibility(int,Vehicle_modelM): bool
Program
+ Main(string[]): void
+ CreateHostBuilder(string[]): IHostBuilder
Startup
+ Configuration: IConfiguration
+ Startup(IConfiguration): Startup
+ ConfigureServices(IServiceCollection): void
+ Configure(IApplicationBuilder,IWebHostEnvironment): void
1
1
AutoRepairAPIController
- DB: DatabaseManager
- GenericErrorCodes: string[]
+ getEmployee(int): ActionResult<string>
+ addEmployee(EmployeeM): ActionResult<string>
+ payEmployee(Employee_invoiceM): ActionResult<string>
+ updateEmployeeInfo(EmployeeM): ActionResult<string>
+ getInvoices(int): ActionResult<string>
+ getInvoice(int,int): ActionResult<string>
+ getManagersDelagates(int): ActionResult<string>
+ assignMechanic(Assigned_toM): ActionResult<string>
+ assignClerk(Associated_withM): ActionResult<string>
+ createWorkOrder(Work_orderM): ActionResult<string>
+ updateWorkOrder(Work_orderM): ActionResult<string>
+ getCustomerWorkOrders(int): ActionResult<string>
+ getWorkOrder(int): ActionResult<string>
+ getEmployeeWorkOrders(int): ActionResult<string>
+ removeWorkOrder(int): ActionResult<string>
+ addCustomer(CustomerM): ActionResult<string>
+ updateCustomer(CustomerM): ActionResult<string>
+ getCustomer(int): ActionResult<string>
+ getCustomerVehicles(int): ActionResult<string>
+ addVehicle(VehicleM): ActionResult<string>
+ updateVehicle(VehicleM): ActionResult<string>
+ getVehicle(string): ActionResult<string>
+ checkVehicleModel(Vehicle_modelM): ActionResult<string>
+ addVehicleModel(Vehicle_modelM): ActionResult<string>
+ removeVehicleModel(Vehicle_modelM): ActionResult<string>
+ getWorkOrderParts(int): ActionResult<string>
+ getPartsOnStock(void): ActionResult<string>
+ updatePart(PartM): ActionResult<string>
+ getPart(int,int): ActionResult<string>
+ addPart(PartM): ActionResult<string>
+ searchCompatiblePart(Vehicle_modelM,string): ActionResult<string>
+ getCatalogPart(int): ActionResult<string>
+ addCatalogPart(int): ActionResult<string>
+ removeCatalogPart(int): ActionResult<string>
+ addCompatibility(int,Vehicle_modelM): ActionResult<string>
+ checkPartCompatibility(int,Vehicle_modelM): ActionResult<string>
HTTP Request
login
+ id: int
+ password: string
+ login(string): void
+ isValid(void): bool
TypeCheck
+ isValid: bool
+ list: List<string>
+ TypeCheck(void): void
+ ListTostring(void): string
Model Classes
EmployeeM
+ Bank_acc_no : long
+ Address : string
+ Lname : string
+ Fname : string
+ Employee_id : int <<PK>>
+ Password: string
+ Salary_Rate : decimal
+ Hourly_Rate : decimal
+ Manager_id : int <<FK>>
+ Pay_type : bool
+ MecFlag : bool
+ CFlag : bool
+ ManFlag : bool
+ AFlag : bool
+ EmployeeM(int,string,int,long,string,string,string,decimal,decimal,bool,bool,bool,bool,bool): EmployeeM
+ EmployeeM(void): EmployeeM
+ EmployeeM(SqlDataReader): EmployeeM
+ SQLSchemaValidParameters(void) : TypeCheck
+ SQLSchemaValidParameters_MINUS_EMPLOYEE_ID(void) : TypeCheck
Employee_invoiceM
+ Invoice_id : int<<PK>>
+ Employee_id : int <<PK>> <<FK>>
+ Amount : decimal
+ Interval_Start_Date : DateTime
+ Interval_End_Date : DateTime
+ Payment_Date : DateTime
+ Hours : decimal
+ Employee_invoiceM(int,int,decimal,DateTime,DateTime,DateTime,decimal): Employee
+ Employee_invoiceM(void): Employee_invoiceM
+ Employee_invoiceM(SqlDateReader): Employee_invoiceM
+ SQLSchemaValidParameters(void) : TypeCheck
+ SQLSchemaValidParameters_MINUS_INVOICE_ID(void) : TypeCheck
Assigned_toM
+ Mechanic_id : int <<PK>> <<FK>>
+ Work_order_id : int <<PK>> <<FK>>
+ Assigned_toM(int,int): Assigned_toM
+ Assigned_toM(void): Assigned_toM
+ Assigned_toM(SqlDataReader): Assigned_toM
+ SQLSchemaValidParameters(void) : TypeCheck
Associated_withM
+ Clerk_id: int <<PK>> <<FK>>
+ Work_order_id : int <<PK>> <<FK>>
+ Associated_withM(int,int): Associated_withM
+ Associated_withM(void): Associated_withM
+ Associated_withM(SqlDataReader): Associated_withM
+ SQLSchemaValidParameters(void) : TypeCheck
Work_orderM
+ Work_order_id : int <<PK>>
+ Closed : bool
+ Amount_Due : decimal
+ Vehicle_VIN : string <<FK>>
+ CustomerID: int <<FK>>
+ Work_orderM(int,bool,decimal,string,int): Work_orderM
+ Work_orderM(void): Work_orderM
+ Work_orderM(SqlDataReader): Work_orderM
+ SQLSchemaValidParameters(void) : TypeCheck
+ SQLSchemaValidParameters_MINUS_WORK_ID(void) : TypeCheck
CustomerM
+ Fname : string
+ Phone_number : string
+ Lname : string
+ Address : string
+ CustomerID : int <<PK>>
+ CustomerM(int,string,string,string,string): CustomerM
+ CustomerM(void): CustomerM
+ CustomerM(SqlDataReader): CustomerM
+ SQLSchemaValidParameters(void) : TypeCheck
+ SQLSchemaValidParameters_MINUS_CUSTOMER_ID(void) : TypeCheck
VehicleM
+ VIN : string <<PK>>
+ Vehicle_model : string <<FK>>
+ Vehicle_make : string <<FK>>
+ Vehicle_year : int <<FK>>
+ CustomerID: int <<FK>>
+ Color : string
+ Registration_No : string
+ VehicleM(string,string,string,int,int,string,string): VehicleM
+ VehicleM(void): VehicleM
Vehicle_modelM
+ Vehicle_make: string <<PK>>
+ Vehicle_model: string <<PK>>
+ Year : int <<PK>>
+ Vehicle_modelM(string,string,int): Vehicle_modelM
+ Vehicle_modelM(void): Vehicle_modelM
+ Vehicle_modelM(SqlDataReader): Vehicle_modelM
+ SQLSchemaValidParameters(void) : TypeCheck
Compatible_withM
+ Part_id : int <<PK>> <<FK>>
+ Vehicle_make : string <<PK>> <<FK>>
+ Vehicle_model : string <<PK>> <<FK>>
+ Vehicle_Year : int <<PK>> <<FK>>
+ Compatible_withM(string,string,int,int): Compatible_withM
+ Compatible_withM(void): Compatible_withM
+ Compatible_withM(SqlDataReader): Compatible_withM
+ SQLSchemaValidParameters(void) : TypeCheck
PartM
+ Part_id : int <<PK>> <<FK>>
+ Part_instance_id : int <<PK>>
+ State : string
+ Price : decimal
+ Work_order_id : int <<FK>>
+ PartM(int,int,string,decimal,int): PartM
+ PartM(void): PartM
+ PartM(SqlDataReader): PartM
+ SQLSchemaValidParameters(void) : TypeCheck
+ SQLSchemaValidParameters_MINUS_INSTANCE_ID(void) : TypeCheck
Catalog_partM
+ Part_id : int <<PK>>
+ Part_name : string
+ Catalog_partM(int,string): Catalog_partM
+ Catalog_partM(void): Catalog_partM
+ Catalog_partM(SqlDataReader): Catalog_partM
+ SQLSchemaValidParameters(void) : TypeCheck
+ SQLSchemaValidParameters_MINUS_PART_ID(void) : TypeCheck