CAR RENTAL MANAGEMENT SYSTEM

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

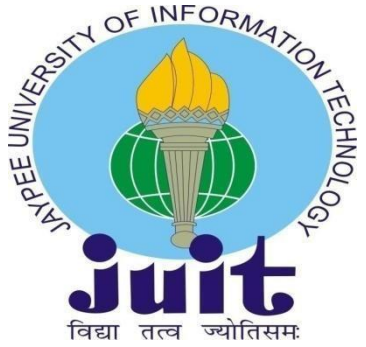
By

Abhiti Labroo (191225)

Under the supervision of

Dr. Deepak Gupta

To



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this report entitled **"Car Rental System Management"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from February 2023 to May 2023 under the supervision of **Dr. Deepak Gupta** (Assistant Professor).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Abhiti Labroo, 191225

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr.Deepak Gupta
Assistant Professor(SG)
Department CSE
Dated: 15-05-2023

PLAGIARISM CERTIFICATE

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: .....

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: _____ **Department:** _____ **Enrolment No** _____

Contact No. _____ **E-mail.** _____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

_____

_____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at .....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

<table>
<tr><th>Copy Received on</th><th>Excluded</th><th>Similarity Index (%)</th><th colspan="2">Generated Plagiarism Report Details (Title, Abstract & Chapters)</th></tr>
<tr><td rowspan="2"></td><td rowspan="4">• All Preliminary Pages
• Bibliography/Images/Quotes
• 14 Words String</td><td rowspan="2"></td><td>Word Counts</td><td></td></tr>
<tr><td rowspan="2">Report Generated on</td><td>Character Counts</td><td></td></tr>
<tr><td rowspan="2"></td><td rowspan="2">Submission ID</td><td>Total Pages Scanned</td><td></td></tr>
<tr><td></td><td>File Size</td><td></td></tr>
</table>

Checked by
Name & Signature **Librarian**

.....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

ACKNOWLEDGEMENT

First and foremost, I want to give God the highest praise and gratitude for His heavenly grace, which made it possible to finish the project work successfully.

My supervisor, **Dr. Deepak Gupta**, an assistant professor in the Computer Science Engineering department at the Jaypee University of Information Technology in Waknaghat, has my deepest gratitude and gratitude. To complete this assignment, my supervisor had extensive knowledge and significant enthusiasm for cloud computing. His unwavering persistence, intellectual leadership, and ability to persevere made this endeavour possible, the constant encouragement, persistent and rigorous oversight and helpful criticism, insightful advice, reading multiple imperfect copies, and any necessary corrections.

I want to extend my sincere appreciation to Dr.Deepak Gupta, Department of CSE, for his gracious assistance in seeing my research through to completion.

A warm welcome would also be extended to everyone who has directly or indirectly assisted me in making this project successful. In this particular circumstance, I would like to thank the personnel —teaching and non-teaching—who have provided me with practical assistance and made my task easier.

Abhiti Labroo
Project Group No. : 44
Rollno.: 191225

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

1. **ASP.NET** - Active Serve Pages.NET
2. **MVC** - Model View Controller
3. **SQL** - Structured Query Language
4. **HTML** - Hypertext Markup Language
5. **CSS** - Cascading Style Sheets
6. **API** - Application Programming Interface
7. **AJAX** - Asynchronous JavaScript and XML
8. **CRUD** - Create, Read, Update, Delete
9. **ORM** - Object-Relational Mapping
10. **JWT** - JSON Web Tokens
11. **DAL** – Data Access Layer
12. **URL** - Uniform Resource Locator
13. **HTTPS** - Hypertext Transfer Protocol Secure
14. **DNS** - Domain Name System
15. **SMTP** - Simple Mail Transfer Protocol
16. **FTP** - File Transfer Protocol
17. **LINQ** - Language Integrated Query
18. **XML** - Extensible Markup Language
19. **JSON** - JavaScript Object Notation

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Automobile companies need effective tools at their disposal in order to administer their complex operational tasks such as managing fleets , customers and reservations efficiently . Our program provides one such solution that offers a range of inventory management services to rental firms to help achieve this. Rental businesses can count on our software to maintain the up to date information of all vehicles in their inventory . By using this program managers can easily add, edit, or delete any car belonging to the company fleet and keep accurate records of its vital attributes like make, model , year and mileage for improved decision making purposes. Ultimately cutting down on time consuming activities such as paperwork thus freeing up more time that can be dedicated to serving customers better.

A significant advantage offered by modern technology in the field of car rentals is its ability to provide companies with valuable customer insights garnered through tracking personal details and rental histories. By assessing this information over time businesses can leverage it for improving customer relations through personalized services tailored specifically towards their needs. The Car Lease Management Systems adept booking management feature further simplifies the process by enabling clients to browse available vehicles online in real time – ensuring hassle free travel planning and access to necessary cars at all times.

Clients can pay securely via the payment interface built into the system, thus enhancing the convenience of the booking procedure. This project has functions including fleet management, client management, reservation management, and payment processing, and it is simple to use. Additionally, the system is dependable and safe, protecting user information and ensuring continuous system available.

CHAPTER -1 INTRODUCTION

1.1 Introduction

The car rental industry has grown significantly over the last few years as a result of the constant increase of interest for vehicles for both business and personal travel. Managing a car rental company is challenging for a variety of reasons, including managing the inventory, handling clients, scheduling management, the processing of payments, and data confidenttiality. These issues can be resolved by developing a comprehensive and efficient system for managing car rentals.

The system will offer ressources for better fleet management, such as tools for analyzing upkeep, locating underutilized cars, and making information-driven fleet acquisition decisions. Additionally, it will offer resources for more effective management of client data, such as personal information, rental historys, and likes and dislikes. By giving clients a quick and accessible way to look for accessible cars, bookings online, and select their choosen model and other brand, the system will simplify reservation administration. Clients can make payments online using the system's safe and dependable payment gateway.

1.2 Problem Statement

The laborious, susceptible to error, and ineffective manual procedures employed by automobile rental firms include manual documentation, telephone-based reservation territory, and hands-on payments. The dearth of modern conveniences like real-time accessibility, e-payments, and management of clients in out-of-date software packages may make it challenging for vehicle rental companies to compete in a market that is changing quickly.

As a result, the problem statement of this project would be to offer a thorough, protected, and effective computerized solution to the obstacles that firms encounter while trying to coordinate their processes, fleet, clients, and schedules. For the system to adapt to the evolving demands of the automobile rental sector, it ought to be simple to use, extensible, and adaptable. In order to safeguard the platform and its clients, it ought to additionally incorporate real-time car availability, payment processing, client administration, and strong security measures.

1.3 Objectives

1.3.1 Simplify reservation management: The system should provide customers with a simple and convenient way to search for available cars, make reservations online, and choose their preferred pick-up and drop-off locations. The system should also provide real-time availability of cars, making it easier for customers to plan their trips and ensuring that they can get the cars they need when they need them.

1.3.2 Improve fleet management: The system should provide car rental companies with tools to manage their fleet more efficiently, such as tracking vehicle maintenance, identifying underutilized or over utilized vehicles, and making data-driven decisions regarding fleet acquisition and retirement.

1.3.3 Streamline customer management: The system ought to give automobile rental businesses the instruments they need to better handle client data, such as personal information, rental history, and preferences. The client experience is able to be enhanced by using the aforementioned data to deliver improved client service.

1.3.4 Provide scalability and flexibility: The system should be created with scalability as well as adaptability in mind, enabling automobile rental businesses to adjust to shifting client demands and competitive circumstances. This can assist automobile rental businesses in long-term profitability and competitiveness.

1.4 Scope

A car rental management system developed is a software application that providess a web-based platform for car rental companies to manage their operations. The scope of a car rental mannagement system developed includes the following features:

Requirement Number	Requirement Name	Requirement Description
Requirement-01	Car Inventory module	This module is a Middleware Microservice that performs following operations: This module will allow customers who wish to rent cars to see from inventory, its pictures, model, price and other important details

Requirement-02	Car Rental module	This module is a Middleware Microservice that performs the following operations: This module works in way that it allows customers to see the total amount of rent that has to be paid including security amount.
Requirement -03	Car Total Payment module	This module is a Middleware Microservice that performs the following operations: This module basically performs functions that will provide what amount has to be paid.
Requirement -04	User Management portal	A Web Portal that allows a user to Login and allows to do following operations: Login Load the Customer Detail Invoke the Process Car Inventory module

Table 1.4.1: The Table shows requirements of the microservices that are being developed in this project.

CHAPTER -2 LITERATURE SURVEY

A comprehensive array of features is offered by car rental management systems to assist organizations in managing their inventory of cars and streamlining their internal operations. Car inventory management, booking and reservation administration, rental administration, transaction management, and maintenance scheduling are a few of the key characteristics and capabilities of these computerized platforms.

The management of leasing operations, car inventory, and network upkeep are only a few of the difficulties faced by car rental organizations. A huge network of cars requires routine service, maintenance, and tidying up, which is a difficult undertaking. Cars must always be in mint condition and ready for lease, according to car rental businesses. It can be difficult to keep track of all these tasks for an extensive inventory of cars. Yet another big difficulty is to make sure that there are certainly constantly sufficient cars on hand to rent. To maximize utilization and lower the chance of accumulating a surplus of idle cars, car rental firms have to manage their fleet of cars.

Through automating numerous of the procedures required in handling an inventory of cars, car rental management systems are able to help in overcoming these obstacles. These kinds of systems frequently have functions like managing stock, automated warnings for maintenance and repairs, and upkeep scheduling. The result is that it is easier for car rental firms to maintain tabs of their fleet, plan repair duties more effectively, and guarantee that cars are accessible for rental whenever they're required. Additionally, by streamlining operations like making the reservation, bookings, and billing purposes, car rental management systems can assist rental companies in managing rental purchases successfully. This lowers the possibility of inaccuracies and raises customer loyalty by enabling rental organizations to handle rental payments fast and properly.

Authors	Published Title	Technology	Description
Shikha Dhiman Pratibha Sharma [1]	Performance Testing: A Comparative Study and Analysis of Web Service Testing Tools, ICMSR, 2021	ASP.NET	In order to pinpoint bottlenecks in performance and assess the effects of various configuration options and tuning techniques, the authors combine load testing, profiling, which is an efficiency metric analysis.

Sandra Sarasan, Ayana Ajith , Archana A.B. [2]	Detection of Security Attacks and their Countermeasures in ASP.NET Web Applications, IJCSIS,2021	ASP.NET	This study is concerned with detecting security vulnerabilities in ASP.NET online applications and suggesting solutions to reduce those vulnerabilities. systems, and encryption methods.
Fanie Reynders [3]	Introduction to ASP.NET Core Springer,2018	ASP.NET	The authors go over the framework and parts of ASP.NET Core, including its backing for dependency injection, middleware pipeline, and flexible layout.
Jean-Rémy Falleri & Xavier Blanc [4]	Automated generation of REST API specification from plain HTML documentation	Web-API	In order to identify and build APIs based on their functionality and semantics, this paper provides a novel method for automated API creation and testing.
Lucas Pelloni, Andrei Zgirvaci, and Thomas Fritz [5]	RESTful API Integration Testing: A Survey, IEEE, 2017	Web-API	The authors list popular API test types as well as the programmes and frameworks that can be used to automate API testing. The limits and unanswered research problems in this field are also covered in the report.
Julie Lerman[6]	Entity Framework 6: Extending Database-First with Code-First, IEEE 2018	Entity Framework-6	This paper explores the hybrid approach of combining Database-First and Code-First approaches in EF6. The author demonstrates how to use Database-First to generate an initial model, and then extend and customize the model using Code-First techniques.
M.Prajapati[7]	Asp.net MVC - generic repository	Entity Framework-6	This study investigates EF6's hybrid strategy, which combines the Database-First and Code-

	pattern and unit of work, IJARW,2019		First techniques. The author shows how to create an initial model using database-first techniques, then how to enhance and customize the model using code-first techniques.
Rowan Miller, Julian Bucknall, and Chris Anderson[8]	Entity Framework 6 in Action, Springer 2020	Entity Framework-6	This book provides an in-depth guide to Entity Framework 6, a popular Object-Relational Mapping (ORM) framework for .NET applications. The authors cover a wide range of topics related to EF6, including data modelling, querying, performance tuning, and database migrations.
Spadini, Davide and Antiche,Mauricio [9]	To Mock or Not To Mock? An Empirical Study on Mocking Practices, IEEE,2017	Mocking	This paper's main contribution was a categorization of the most often mocked and not mocked dependencies, based on a quantitative analysis on three OSS systems and one industrial system and the main challenges faced by developers when making use of mock objects in the test suites, also extracted from the interviews and surveys
Shaikh Mostafa and Xiaoyin Wang[10]	An Empirical Study on the Usage of Mocking Frameworks in Software Testing, QSIC,2014	Mocking	In practice, mock objects have been used in software testing to simulate such missing dependencies, and A number of popular mocking frameworks have been developed for software testers to generate mock objects more conveniently.
Mohammad Mahdi Hassan and Wasif Afzal.[11]	Testability and Software Robustness: A Systematic	Unit Testing	It works on Software robustness and software testability

	Literature Review, IEEE,2015		
Sharma, Rashmi and Saha, Anju[12]	A Systematic Review of Software Testability Measurement Techniques, IEEE,2018	Unit Testing	This paper works upon software quality, testability assessment, testability improvement, controllability, observability.
Mustafa M. Tikir and Jeffrey Kenneth Hollingsworth[13]	Efficient Instrumentation for Code Coverage Testing, ACM, 2002	Unit Testing	The paper presents an approach to dynamically insert and remove instrumentation code to reduce the runtime overhead of code coverage. It also explores the use of dominator tree information to reduce the number of instrumentation points needed

Table 2.1: The Table shows the Author's name, the Proposed Approaches, the Journal it was published in, the Year of publication and their technologies.

CHAPTER -3 SYSTEM ANALYSIS AND DEVELOPMENT

3.1 System Design

3.1.1 Proposed Design

The design would consist of three main microservices that are being developed and secondary technologies that will aid in proper functioning of the project. The project design components are as follows:

1. **Car Management Microservice**: The management of the rental car inventory would fall under the purview of the fleet of vehicles microservice. Then, for each car, it would log the year, make, model, and any other relevant data. The microservice would also keep track of the availability, location, and history of each car's rentals.

2. **Rental Microservice**: In addition to monitoring the number of vehicles, length, kind, clientele for whomever it is being hired, period of rental; additionally, and status of payment, the car rental micro-services could also be in responsibility for renting cars to customers.

3. **Payment Microservice**: The money transfer microservice would have the responsibility in charge of tracking payments, comprising the billing process, the sum paid, its type, and its current status, as well as calculating any potential price breaks.

4. **Database:** All pertinent information, including client identities, car specifics, rental contracts, and financial information, must be kept in a centralized database.

5. **User Interface**: Clients must be able search for, reserve, and keep track of automobiles through a platform that is easy to use. Employees will be able to manage bookings, modify car accessibility, and examine reports thanks to the user interface.

Overall, the system is broken down into smaller, more manageable components that can be developed, deployed, and maintained independently. This approach offers greater scalability, resilience, and flexibility, making it easier to add new features or update existing ones.

3.1.2 System Architecture Design

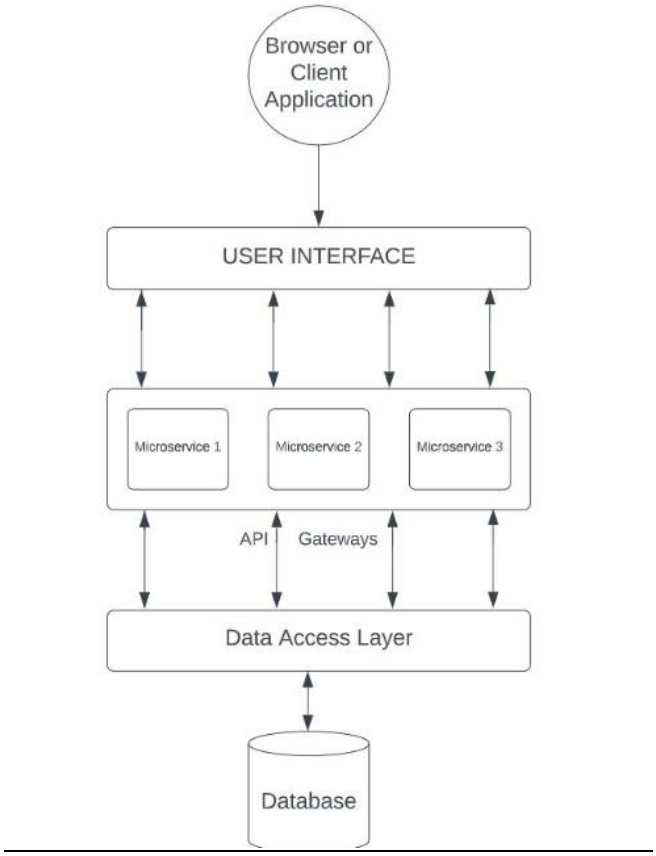


Figure 3.1.2.1 This above figure depicts the software architecture design of the project with three major microservices.

3.1.3 Schematic Diagram

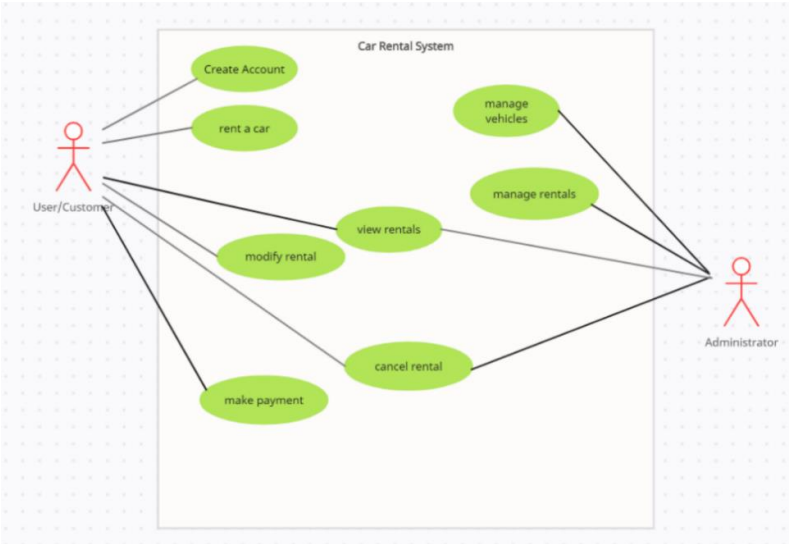


Figure 3.1.3.1 This above figure depicts the Schematic Diagram design of the project with three major microservices with two major use case clients, that are User and Admin.

3.1.4 Use Case Diagram (Contextual Level)

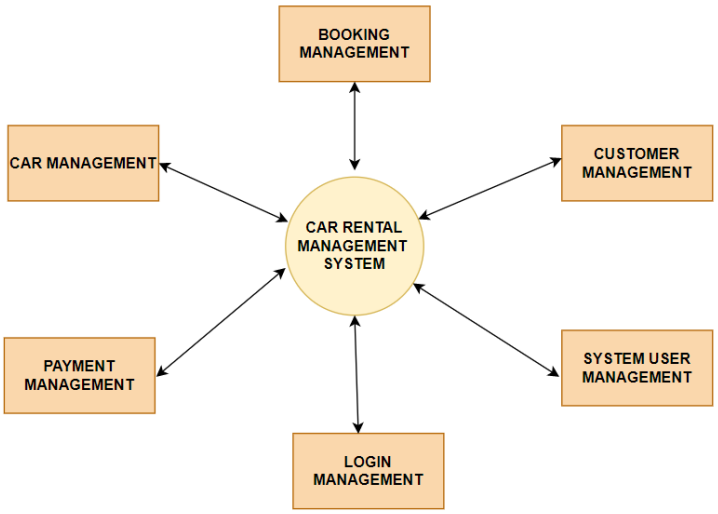


Figure 3.1.4.1 This above figure depicts the use case diagram (Contextual) of the project depicting how the how customer and rental company interacts with each other on the application.

3.1.5 Use Case Diagram (High Level)

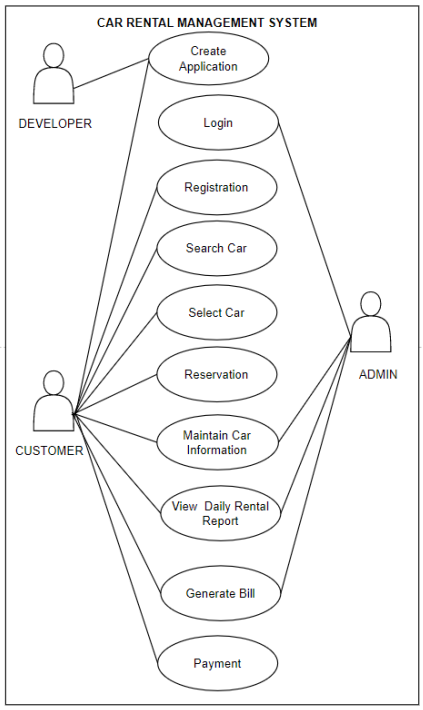


Figure 3.1.5.1 This above figure depicts the use case (high) diagram of the project depicting how the how customer and rental company interacts with each other on the application.

3.2 System Analysis

3.2.1 Assumptions

The following could be presumptions for an automobile rental management system:

1. The programme will store data about customers, vehicles, rental agrements, and various other appropriate data in a single centralised location.

2. The portal will let customers search for and reserve cars based on their prefereneces, such as the kind of caar, how long they want to rented it for, where they live, and other details.

3. The system will control the entire rentaal process, includingg picking up and returning the car, billing, and managing payments.

4. The online platform will have security features to protect confidential client information and prevent fraud.

3.2.2 Dependencies

The dependencies of a car rental management system project could include:

1.Hardware and software infrastructure: In order to execute the programme, the platform would need hardware like server infrastructure, network components, and storage devices. Additionally, it would require software dependencies including programming tools, management systems for databases, and operating systems.

2.Data management: The platform depends on a dependable and safe database to record all pertinent data, including client details, car specifics, rental contracts, and information about payments.

3.**User interface design**: For consumers as well as staff members to interact with the system, it would be necessary to have an intuitive user interface.

4.**Testing and quality assurance**: In-depth analysis as well as quality management procedures would be necessary to make sure the system performs as intended, complies with the demands of the company, and provides free of defects and faults.

3.2.3 Risks

There are several risks that could be associated with a car rental management system project, including:

1. **Data breaches**: Since the system will likely retain private consumer and banking details, it might become an easy target for attacks via the internet and data breaches. By putting in place robust safety precautions like encryption, access controls, and recurring security inspections, the danger of breaches of data could be reduced.

2. **System downtime**: Any system outage could put clients out of their comfort zone and cost the car rental company money. Employing resilience regulations such as servers for backups and rollover methods, could reduce the likelihood of system outages.

3. **User adoption**: The appliance's success would be contingent on how well clients and staff members used it. Any problems with customer acceptance could lead to poor system utilization and reduced revenue. By educating and assisting consumers, doing studies on users, and enhancing interface design, the likelihood of user adoption problems may be reduced.

4. **Regulatory compliance**: The entire system would have to abide by all applicable regulations and legislation, including those governing safeguarding information and the handling of payments. Administrative and economic penalties could be imposed for any non-compliance. By being aware of and abiding by all pertinent laws and regulations, the risk of non-compliance may be reduced.

   These potential risks might have been recognized, assessed, and reduced by carrying out a thorough risk assessment and putting risk management techniques into place.

3.2.4 Hardware and Software Requirements

3.2.4.1 Hardware Requirements

1. Developer Desktop PC with 8GB RAM

3.2.4.2 Software Requirements

1.Frontend/UI:

1.1 HTML/CSS/JavaScript

1.2 A frontend framework or library (e.g., React, Angular, Vue.js)

1.3 UI design tools (e.g., Adobe XD, Sketch)

2.Backend/API:

2.1 C#/.NET framework

2.2 ASP.NET Core for building the RESTful API

2.3 Entity Framework Core for handling database operations

2.4 Swagger for API documentation

3. Database:

3.1 Microsoft SQL Server or another relational database management system (RDBMS)

3.2 Additionally, some other tools and services that may be helpful for development and deployment include:

3.2.1 Git/GitHub for version control

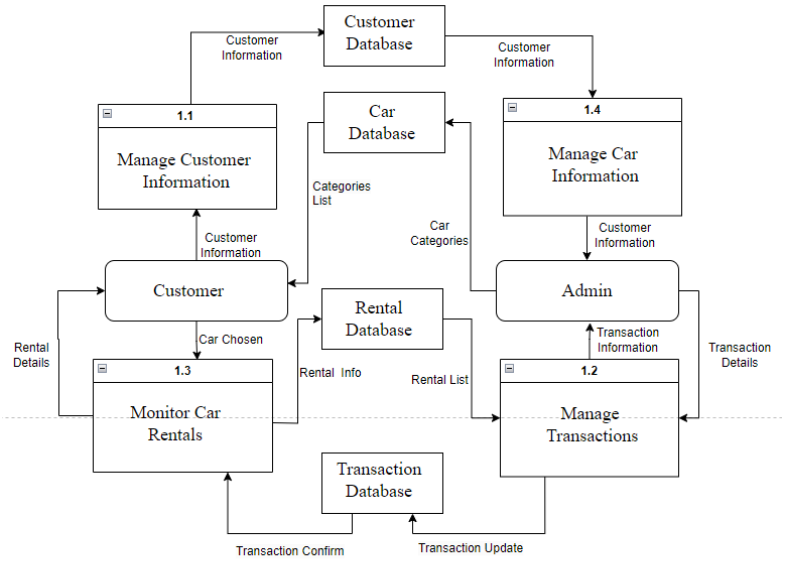
3.2.2 Visual Studio for development

3.3 System Development

System evaluation, development, execution, and validation are usual phases during developing of a car rental management system. Finding the system's minimum specifications, which include the capacity to manage client data, car inventory, bookings, and transactions, is the initial step in the system evaluation.

Using a programming language like ASP.NET, the application is put into effect once the data model and application framework have been developed. The user interface, database access layer, and business logic must all be coded in order to accomplish this.

The following phase is to develop the system's framework and data model after the application's specifications have been established. This entails figuring out the general framework of the system, together with the numerous elements and their interactions, the relational database schema, and the links amongst tables.



CAR RENTAL SYSTEM - DATA FLOW DIAGRAM

Figure 3.3.1 This above figure depicts the data flow diagram of the project depicting how the how customer and rental company interacts with each other on the application with each of the mircroservices.

3.3.1 System Development – Car Microservice

The Car microservice will be defined in the data model, which has the features like and data variables like CarId, CarType, Brand etc. Taking advantage of the ASP.NET Web API framework to create the Car micro-services API. For maintaining the Car entity, the API ought to encompass CRUD actions (Create, Read, Update, and Delete).

Then next step would be to integrate car microservice with the Rental microservices and Payment microservice to give complete car rental system capabilities. To validate the car microservice's usability and dependability, testing it employing a unit testing framework like NUnit would be the next phase.

3.3.2 System Development – Rental Service Microservice

RentalId, CarId, CustomerId, RentalSTime, RentalETime, and RentalType are just a few of the components that make up the Rental microservice in the automobile rental system. All rental payment's special identification number is the RentalId, which is produced automatically. To preserve the connections among the things, CustomerId are foreign keys that respectively point to the car microservices. While the RentalETime column records the completion time of the rental

duration, the RentalSTime column records the beginning of the rental time frame. The RentalType column lists the many types of rentals, including hourly, daily, and weekly rentals. The car rental company may supervise rental operations and keep account of which automobiles have been rented out and for duration thanks to this microservice.

3.3.3 System Development –Payment Microservice

The payment is the name of the microservice in charge of handling transactions for renting a vehicle. The PaymentId column, which has an integer data type, serves as the Payment table's primary key. This update-incrementing field assigns an identity number to each customer who makes a purchase. The RentalId column is an integer data type foreign key that corresponds to the Rental table's primary key. It depicts the renting process that is related to payments.

3.3.4 System Development – User Authorisation and Authentication

Throughout the car rental management system, user profiles, authorization, and authentication are managed by the customer authorization and authentication microservice. This microservice's purpose is to handle and safely preserve user data, such as credentials for login, roles, and permissions. Only those individuals who have been granted access to the application and the associated assets can use it thanks to the secure authentication and authorisation processes provided by these protocols. In its entirety, through guaranteeing that client accounts are administered safely and effectively and that clients are given adequate access and rights to the platform and its facilities.

3.4 Database Design

The database design, which dictates the way data will be retained and accessed across the system, is an essential part of the vehicle rental management system. The database design for the undertaking contains tables for cars, clients, rentals, and payments.

The cars inventory table provides details concerning the automobiles being rented, such as their model and manufacturer, rental rates per hour, day, and week, date as well as cost of purchase, and capacity. The renters' names, contact details, and previous rentals are all kept on file.

The rentals table keeps account of every rental transaction, comprising the person who leased the vehicle, the vehicle itself, how long it was rented for, and how much it charged. The rental ID and the amount paid are two pieces of data that are included in the payments table for every one of the payments provided by the client.

Relationships amongst those tables are also part of the relational database schema. To differentiate between the car that is being leased and the individual who is borrowing it, for instance, the rentals table has foreign keys that link to the cars and payments tables. In order to connect payments to rental operations, the payments table additionally has a foreign key that refers to the rentals table.

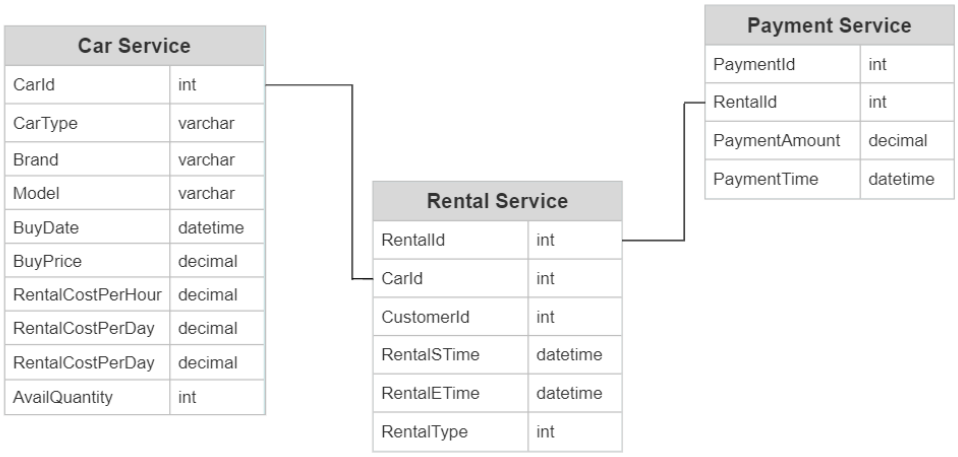


Figure 3.4.1 This above figure depicts the data model diagram of the project depicting how the how all three tables are connected to each other for each of the following microservices.

1.Car Inventory Service

Column Name	Data Type	Length	Nulls
CarId	Int	-	No
CarType	varchar	50	No
Brand	varchar	50	Yes
Model	varchar	50	Yes
BuyDate	datetime	-	Yes
BuyPrice	decimal	-	Yes
RentalCostPerHour	decimal	-	Yes
RentalCostPerDay	decimal	-	Yes
RentalCostPerWeek	decimal	-	Yes
AvailQuantity	int	-	No

Table 3.4.1: The Table shows Car Module table and Column Name and its DataType

2.Rental Car Service:

Column Name	Data Type	Length	Nulls
RentalId	int	-	No
CarId	int	-	No
CustomerId	int	-	No
RentalSTime	datetime	-	No
RentalETime	datetime	-	Yes
RentalType	int	-	No

Table 3.4.2: The Table shows Rental Car Module table and Column Name and its DataType

3.Payment Service:

Column Name	Data Type	Length	Nulls
PaymentId	Int	-	No
RentalId	Int	-	No
PaymentAmount	decimal	-	No
PaymentTime	datetime	-	No

Table 3.4.3: The Table shows Payment Module table and Column Name and its DataType

CHAPTER -4 PERFORMANCE ANALYSIS

4.1 Technology Stack

The advanced technology underpinning the automobile rental management system is adaptable and will take the growth department's demands and requirements into account. However, a few of the developments and those that are widely applied in implementations are as follows:

4.1.1 ASP.NET

The Application Programming Interface is the primmary web application building platform used in the this projeect. The MVC construction, one of its many characteristics, allows developers to separate a platform's display and functionality layers, making code administration and upkeep easier. A range of built-in instruments and features, such as input validation, registration and authentication, and the use of encryption, are also provided by ASP.NET for developing secure applications for the interneet. Finally, ASP..NET is a powerful development framework that provideees developers with the tools and capabiliities they require to build scallable, simple to manage, and dependable web-basedd applications.

4.1.2 Entity Framework Core-6

The multiple platforms EF-Core- 6 framework is an inexpensive and free to download. Programmer may interact regarding database using .NET entities thanks to a method that lets them map .NET instances to table contents in databases and the other way around. Among of the numerous database providers confirmed by EF Core 6 that enables developers to generate inquiries using C# programming language is Microsoft SQL Server.

4.1.2.1 Code First Approach

An ORM framework called EF offers a mechanism for mapping table contennts in databases to CLR instances. In EF Core, the entity classes that represent the database tables are made, and the resulting classes are then used for producing the database schema. The database structure is then developed by the EF Core using the entity classes. The structure of the database can be created and maintained as code using the CF methodology, which makes it simpler to manage and modify. Migrations refers to the technique of building a database schema from entitty types. The migrations functionality is used by EF Core to monitor modificattions to the entity classes and implement those modifications to the database schema. Developers need to initialise the entity classses and relationships betweeen them in order to adopt the CF strategy in EF Core.

4.1.3 Database

ASP.NET can be implemented in conjunction with SQL Server, a database management system, for managing and retrieving data. The outcome is a robust database architecture that provides a number of characteristics and abilities to manage data, including the capacity to process interactions, backing up data and rehabilitation, along with data confidentiality. SQL Server is widely used in ASP.NET applications that are intended for business use due to its reactivity and versatility. It can handle a lot of data and integrate easily with other Microsoft programmes. It also offers cutting-edge security techniques like data compression, position-based protection, and the field of auditing, which can assist protect critical database information.All things considered, Microsoft's SQL Server is a reliable and practical database platform that can be utilised to administer and save data in ASP.NET programmes.

4.1.4 Web-API

The term "Web API," which stands for "Web Application Programming Interface," refers to a specific type of API designed only for web-based applications. Numerous programmes can communicate with other applications using HTTP and HTTPS, two widely used web-based guidelines, through a connection to the web. A RESTful API is an administrative framework for a website that is based on the HTTP protocol. The acronym RST stands for a set of guidelines that describe how interactions among servers and their clients should be handled. GET, POST, PUT, and DELETE are common HTTP methods that the client can use to interact with the server-provided contents in a RESTful architecture.

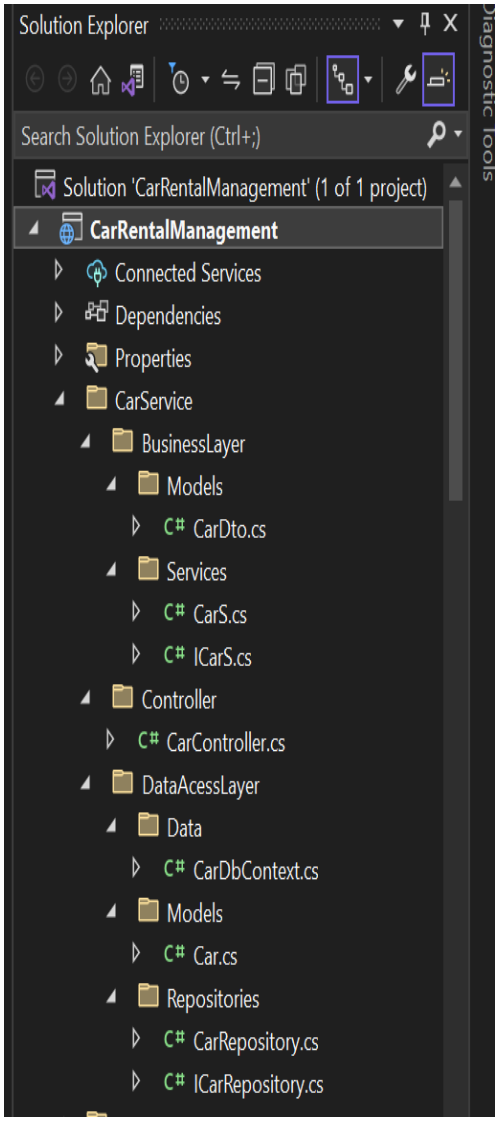
4.1.5 Unit Testing and Mocking

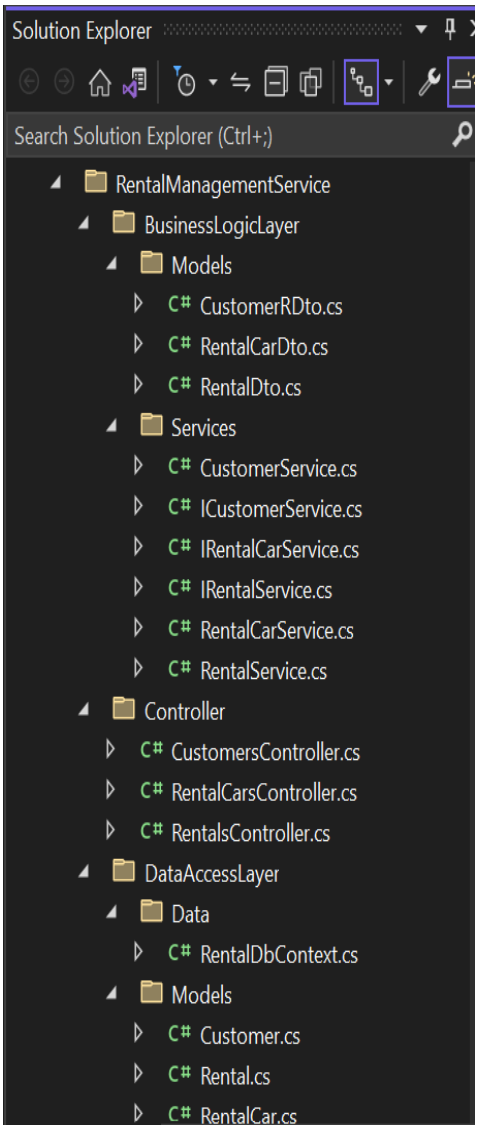
Mocking and Unit Testing are crucial elements in the process of producing software because they assure the precision and standard of the manuals being written. Unit evaluation is the practise of examining solitary individual components or sections to ensure they are functioning properly and as intended in the overall scheme of the rental car operation. To do this, one must write autonomous execution tests that simulate various situations and relevant data in order to verify that the final product of the produced code produces the desired outcomes. It offers an arrangement for creating and executing digital investigations that test the effectiveness of the many steps, techniques, and methodologies used to create software applications.

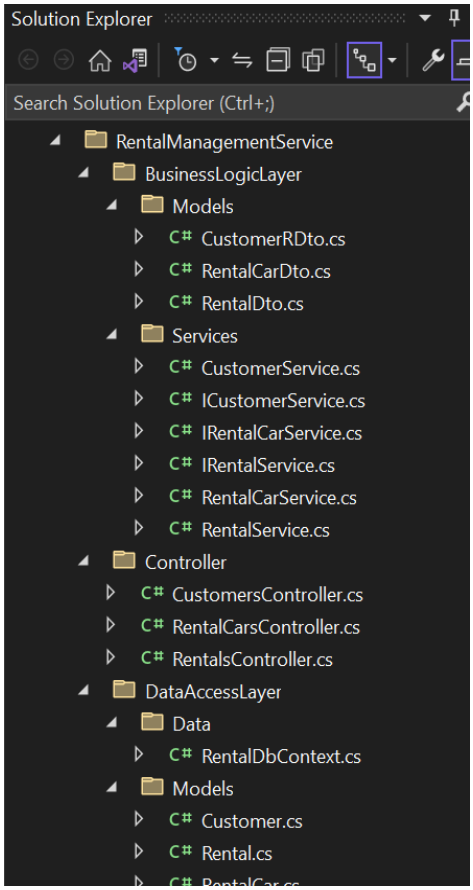
4.2 Source Code

Following figures will show the source code of the CarRentalManagement Project which is written on VS Code. This source code follows the code first approach. For each microservice the flow go as follows, it will start from data access layer folder which has the file in models folder then to go to the file Data folder then it will traverse itself fully in business layer and then it will move to repositories in DAL and then move to controller.

Flow of Whole Project







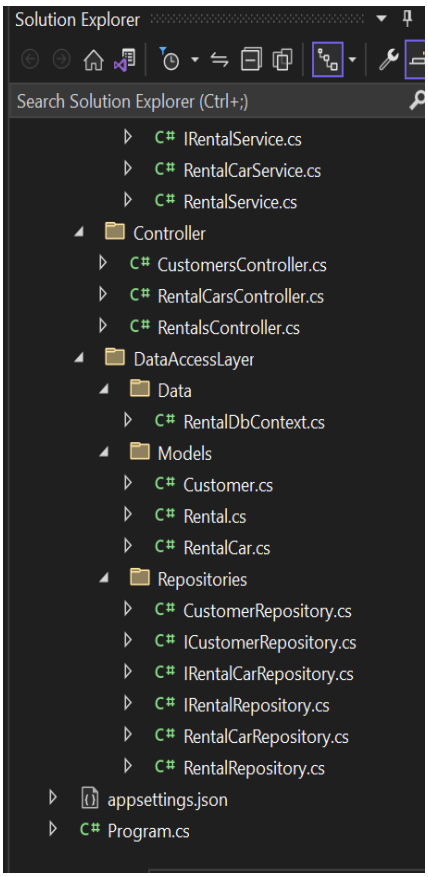


Figure 4.2.1 – 4.2.4 These images show the workflow and all files that are available for this project. These figures basically cover microservices and layers formed in each one of them.

Car Microservice –> Data Access Layer -> Models -> Car.cs

```
namespace CarRentalManagement.CarService.DataAcessLayer.Models
{
    28 references
    public class Car
    {
        9 references
        public int CarId { get; set; }
        10 references
        public string CarType { get; set; } = null!;
        8 references
        public string CarBrand { get; set; } = null!;
        6 references
        public string CarModel { get; set; } = null!;
        8 references
        public DateTime BuyDate { get; set; }
        8 references
        public decimal BuyCost { get; set; }
        8 references
        public decimal RentCostPerHour { get; set; }
        8 references
        public decimal RentCostPerDay { get; set; }
        8 references
        public decimal RentCostPerWeek { get; set; }
        9 references
        public int Inventory { get; set; }
    }
}
```

Figure 4.2.5.1 This figure depicts the attributes that will be present in Car database which is the data that would be available for this microservice hence in the data access layer.

Car Microservice –> Data Access Layer -> Data -> CarDbContext.cs

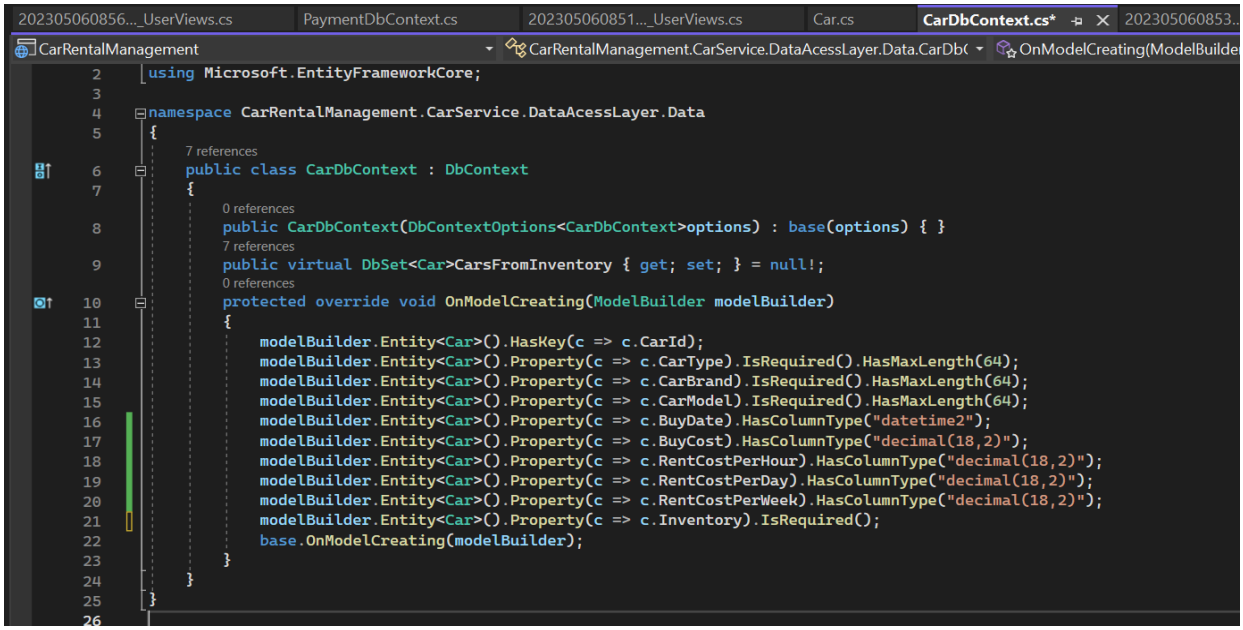
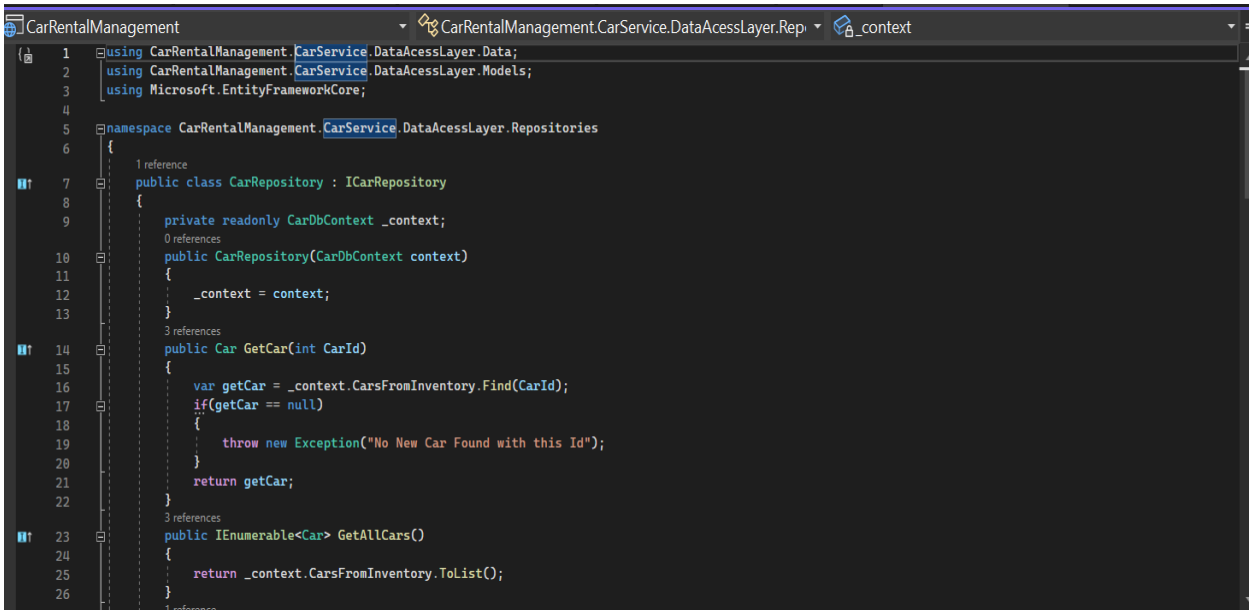
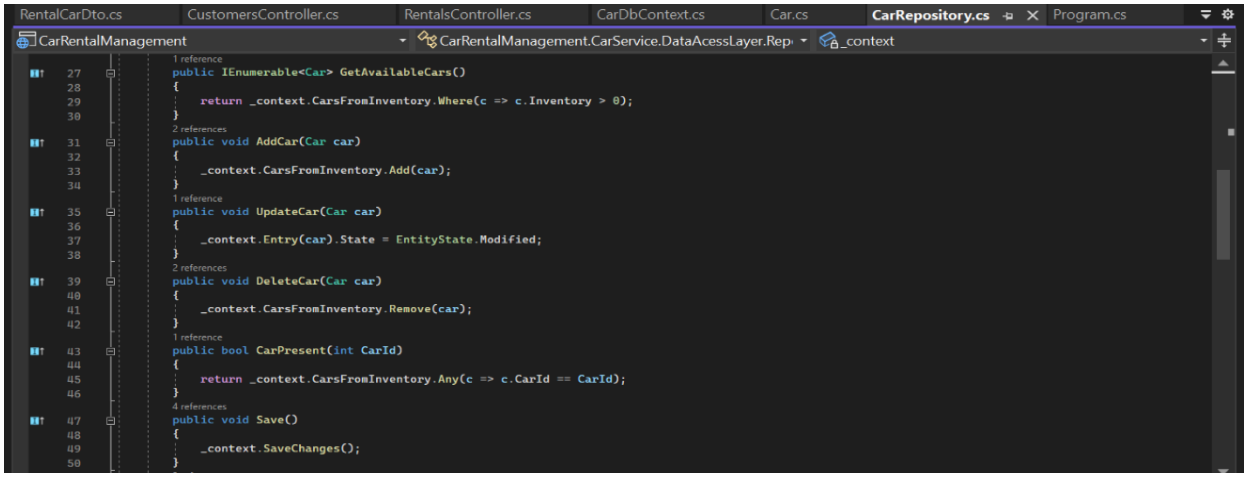
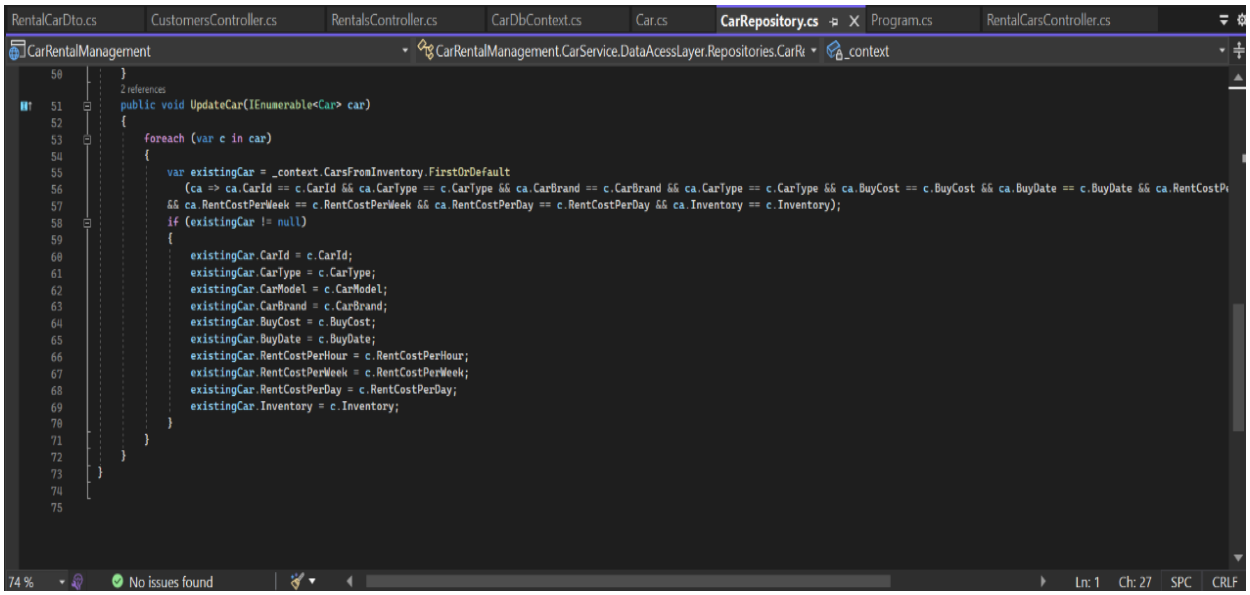


Figure 4.2.5.2 This figure depicts the Database of *CarDbContext* which also makes a table *CarsFromInventory* and then show the results that each of the attributes has.

Car Microservice –> Data Access Layer -> Repositories-> CarRepository.cs







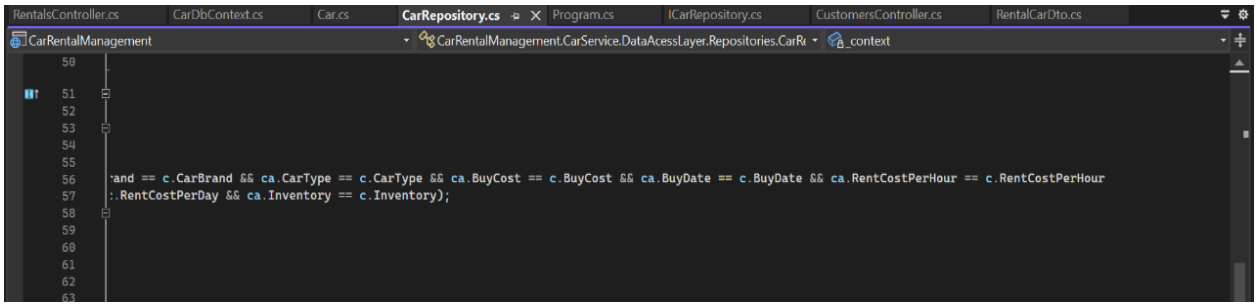


Figure 4.2.6-4.2.9 This figures shows the *CarRepository* file which comprises classes or methods that encapsulates all data logic.

Car Microservice –> Data Access Layer -> Repositories-> ICarRepository.cs

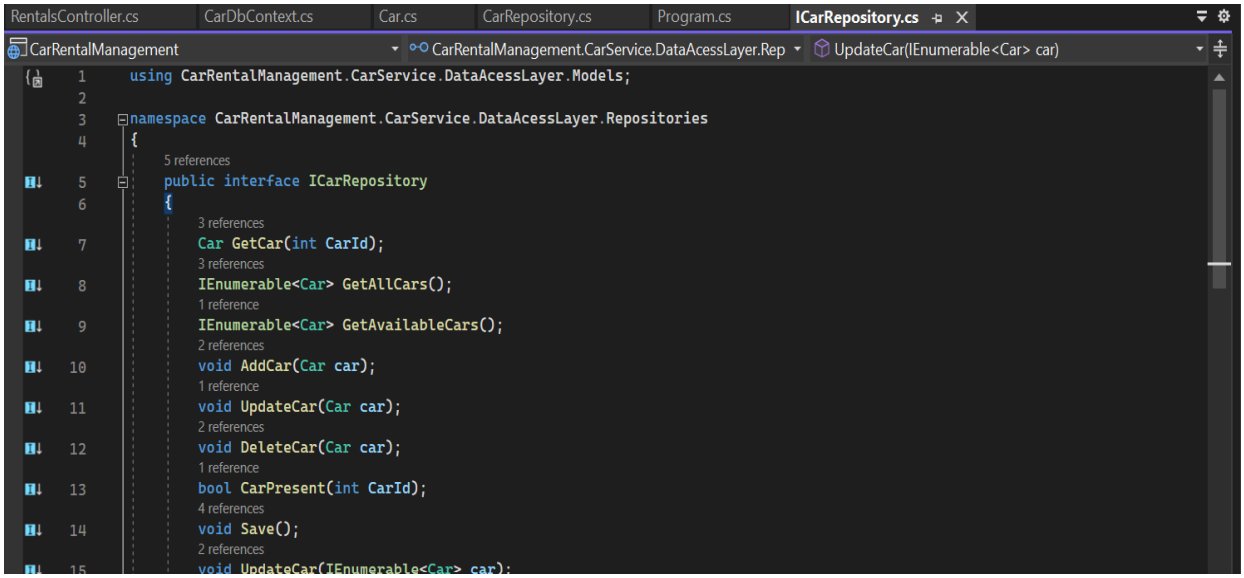


Figure 4.2.10 This figure shows the *ICarRepository* which is an interface for the implementation for CarRepository.

Car Microservice –> Business Layer ->Models-> CarDto.cs

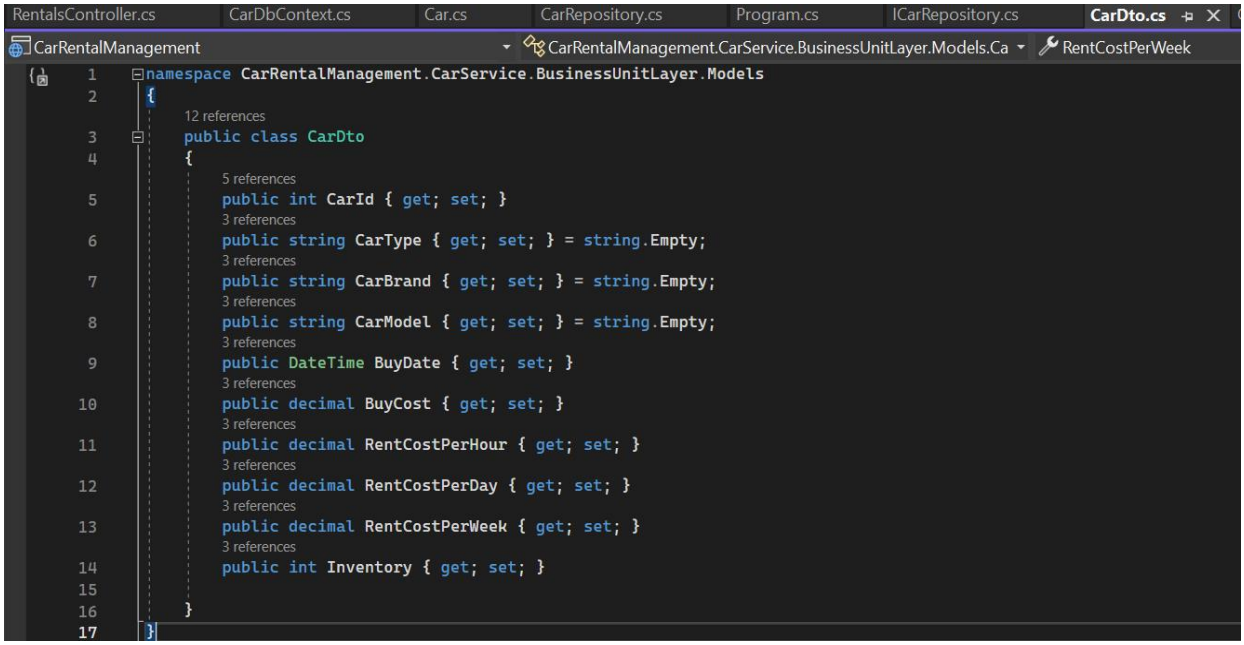
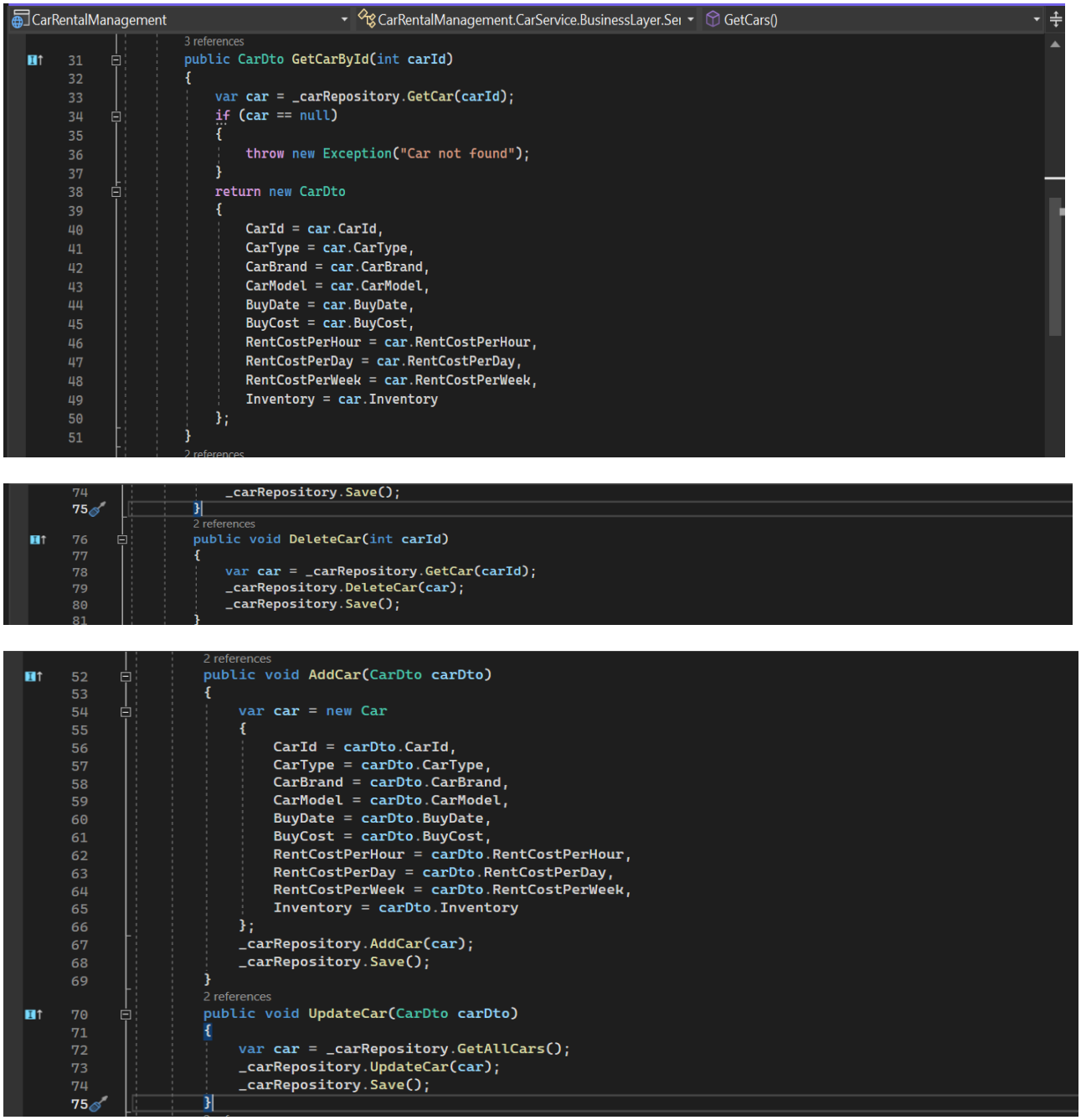


Figure 4.2.11: This figure shows the *CarDto* which is data transfer objects which is used to make application more modular and flexible.

Car Microservice –> Business Layer ->Services -> CarS.cs



```
CarRentalManagement | CarRentalManagement.CarService.BusinessLayer.Ser | GetCars()

3 references
31  public CarDto GetCarById(int carId)
32  {
33      var car = _carRepository.GetCar(carId);
34      if (car == null)
35      {
36          throw new Exception("Car not found");
37      }
38      return new CarDto
39      {
40          CarId = car.CarId,
41          CarType = car.CarType,
42          CarBrand = car.CarBrand,
43          CarModel = car.CarModel,
44          BuyDate = car.BuyDate,
45          BuyCost = car.BuyCost,
46          RentCostPerHour = car.RentCostPerHour,
47          RentCostPerDay = car.RentCostPerDay,
48          RentCostPerWeek = car.RentCostPerWeek,
49          Inventory = car.Inventory
50      };
51  }
2 references
74          _carRepository.Save();
75      }
2 references
76  public void DeleteCar(int carId)
77  {
78      var car = _carRepository.GetCar(carId);
79      _carRepository.DeleteCar(car);
80      _carRepository.Save();
81  }
2 references
52  public void AddCar(CarDto carDto)
53  {
54      var car = new Car
55      {
56          CarId = carDto.CarId,
57          CarType = carDto.CarType,
58          CarBrand = carDto.CarBrand,
59          CarModel = carDto.CarModel,
60          BuyDate = carDto.BuyDate,
61          BuyCost = carDto.BuyCost,
62          RentCostPerHour = carDto.RentCostPerHour,
63          RentCostPerDay = carDto.RentCostPerDay,
64          RentCostPerWeek = carDto.RentCostPerWeek,
65          Inventory = carDto.Inventory
66      };
67      _carRepository.AddCar(car);
68      _carRepository.Save();
69  }
2 references
70  public void UpdateCar(CarDto carDto)
71  {
72      var car = _carRepository.GetAllCars();
73      _carRepository.UpdateCar(car);
74      _carRepository.Save();
75  }
```

Figure 4.2.12 – 4.2.14: These figures depicts the *CarService* include the business services, the services that are in charge of carrying out the program's business logic. The services themselves can interface with the presentation layer and the data access layer in a standardised way by using Data Transfer Objects (DTOs) in the services.

Car Microservice –> Business Layer ->Services -> ICarS.cs

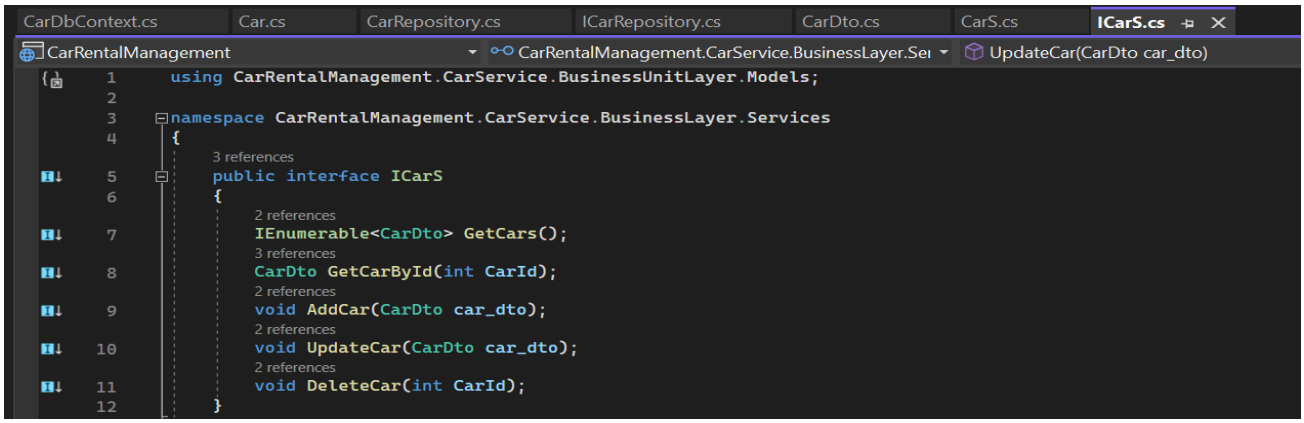
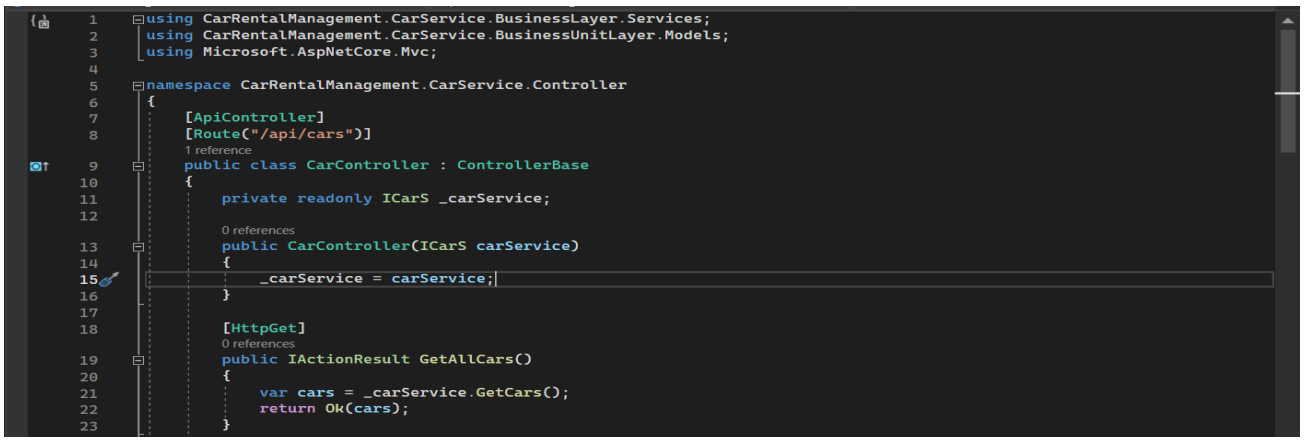
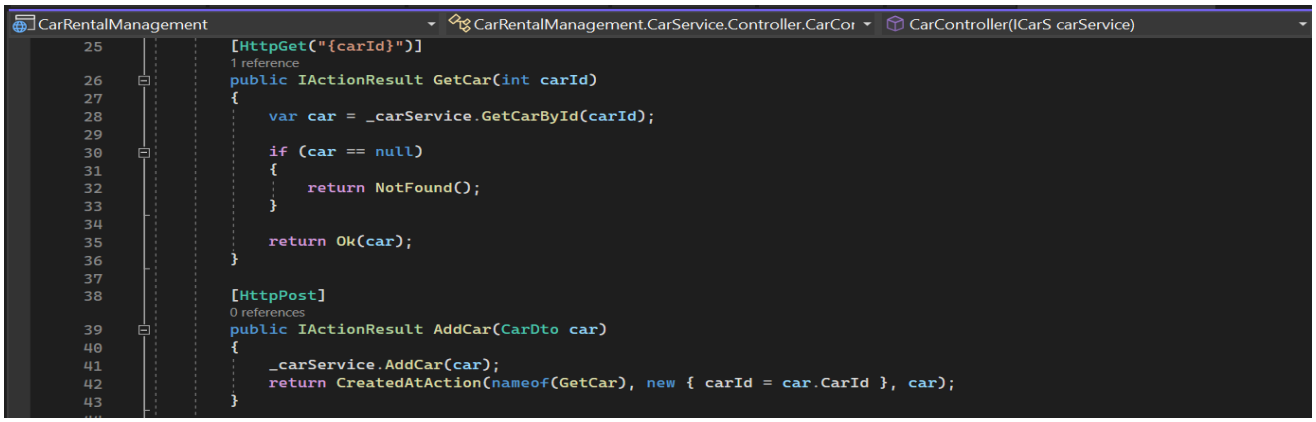


Figure 4.2.15: This figure shows the *ICarService* which is used in order to give the layer that presents data the necessary features, the business logic layer needs to implement a set of functions that are defined by the interface

Car Microservice –>Controller-> CarController.cs





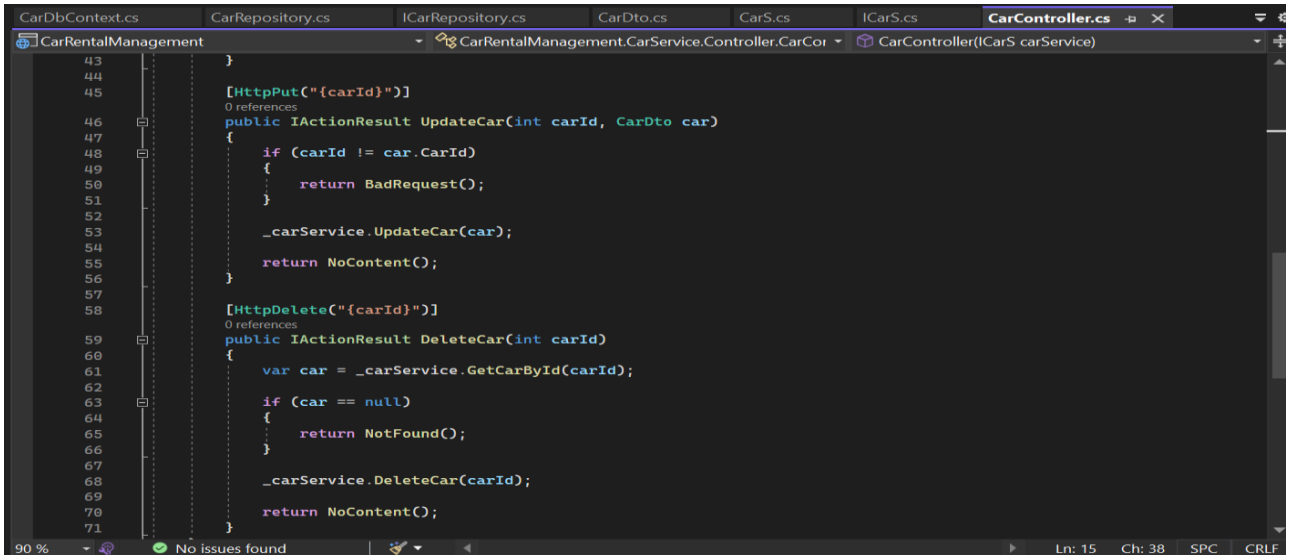


Figure 4.2.16- 4.2.18: This figure shows the *CarController* is essential for handling requests that come in, sending them through the right action method, and producing a response to send back to the user.

RentalManagementService Microservice –>Data Access Layer-> Models

-> Customer.cs

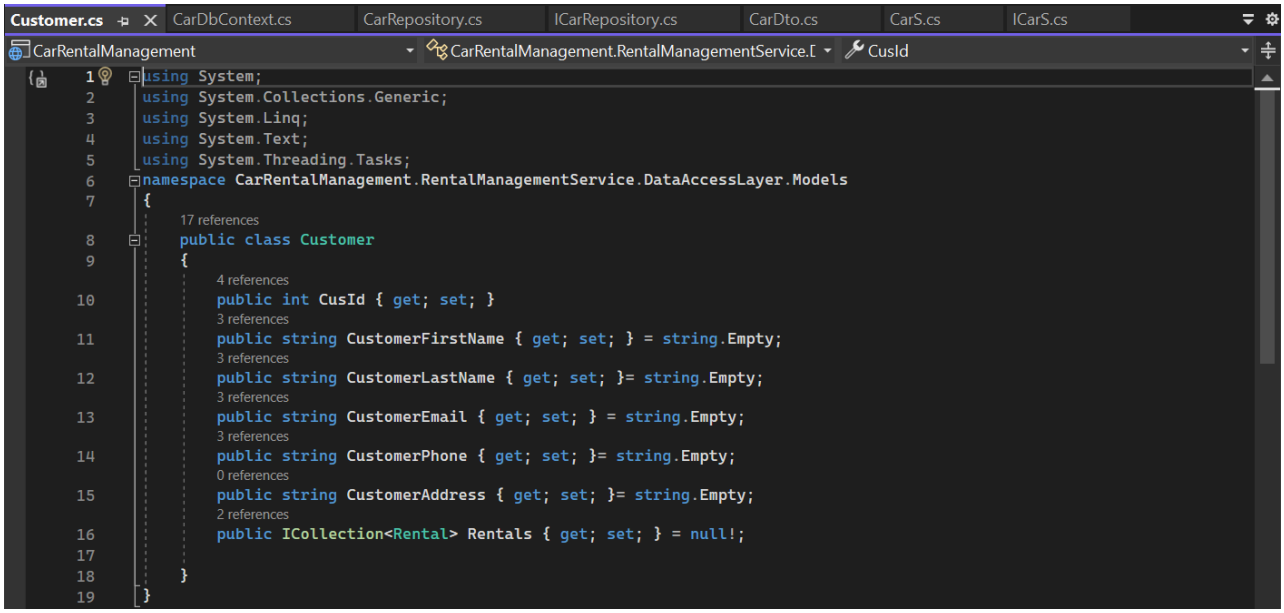


Figure 4.2.19: This figure shows the file *Customer.cs* essential for handling attributes and the columns in the customer table regarding the customers in the microservices.

RentalManagementService Microservice –>Data Access Layer-> Models -> Rental.cs

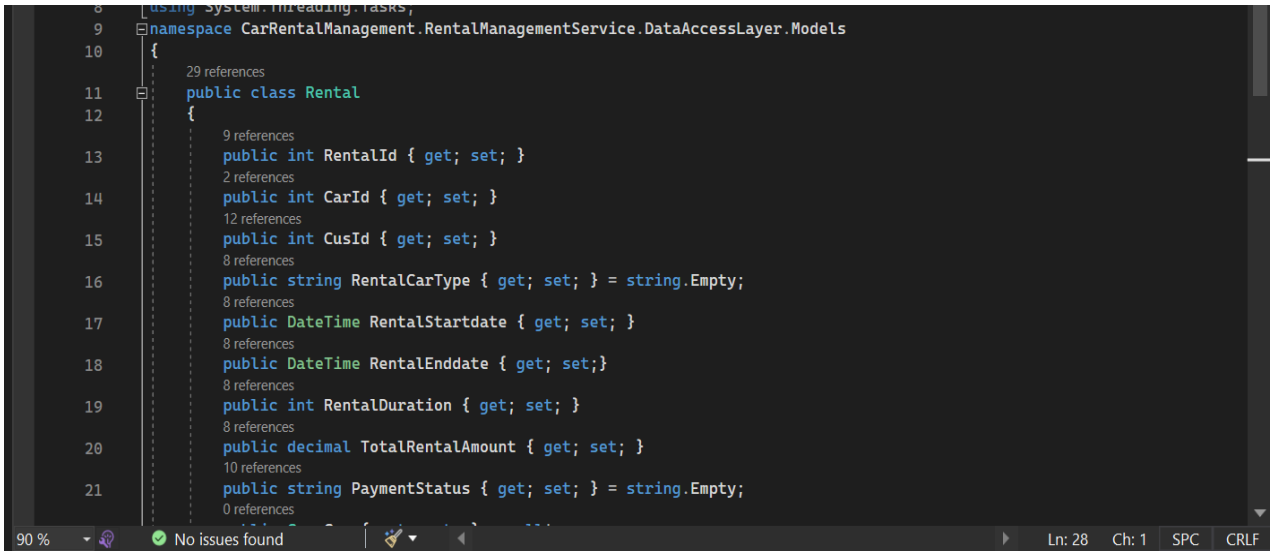


Figure 4.2.20: This figure shows the file *Rental.cs* is essential for handling attributes and the columns in the rental table regarding the rentals in the microservices.

RentalManagementService Microservice –>Data Access Layer-> Models -> RentalCar.cs

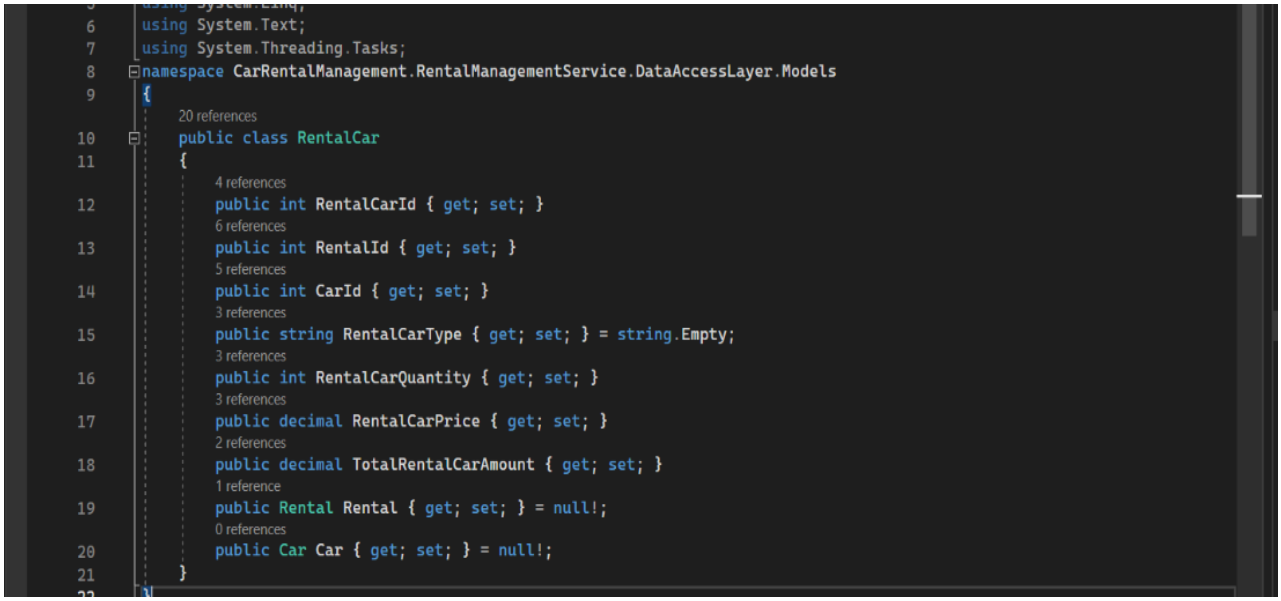
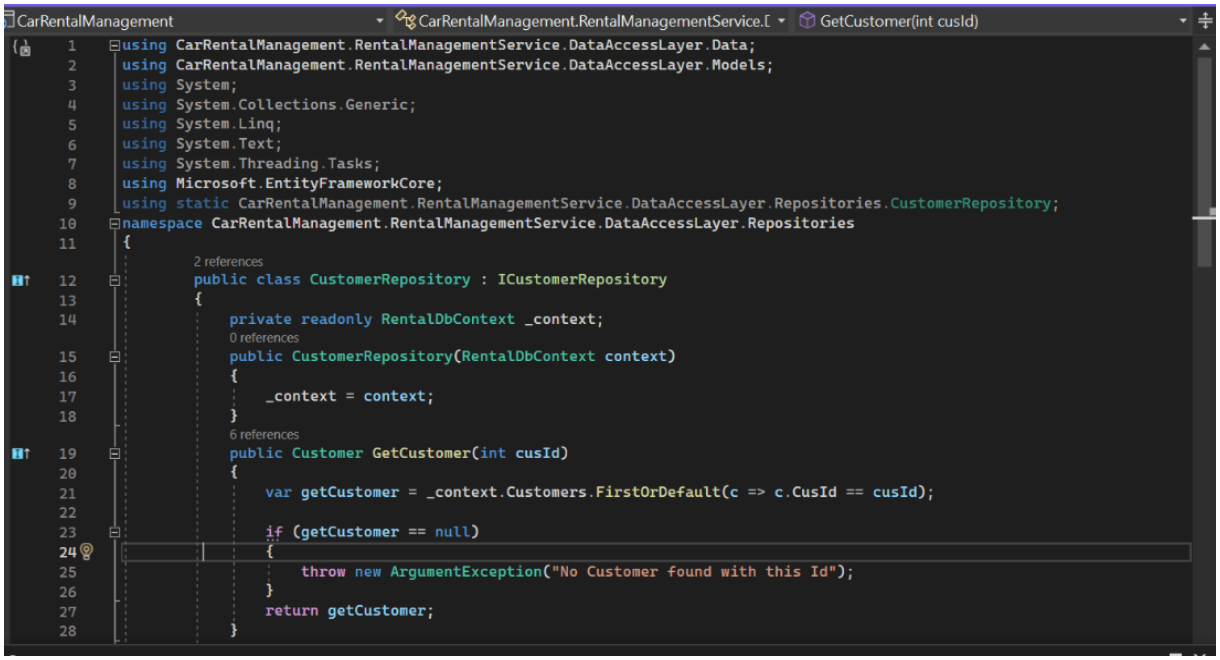


Figure 4.2.21: This figure shows the file *RentalCar.cs* is essential for handling attributes and the columns in the rented cars table regarding the rentals in the microservices.

RentalManagementService Microservice –>Data Access Layer-> Repositories-> CustomerRepository.cs



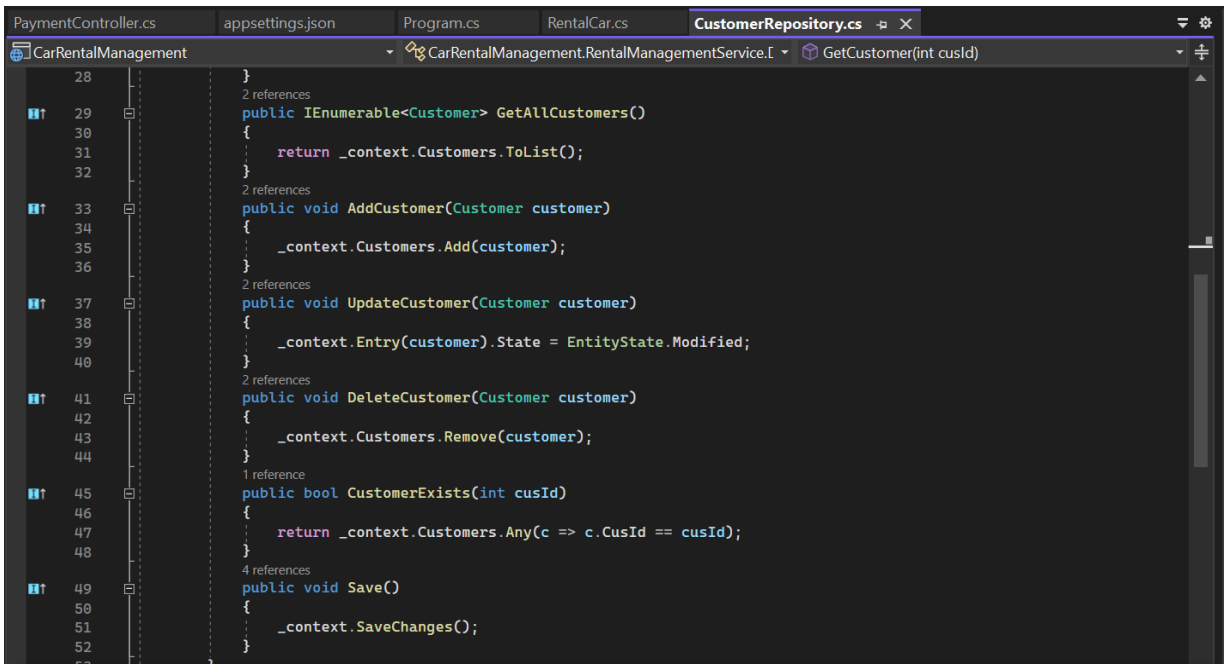


Figure 4.2.22-4.2.23: This figures shows the *CustomerRepository* file which comprises classes or methods that encapsulates all data logic.

RentalManagementService Microservice –>Data Access Layer-> Repositories-> ICustomerRepository.cs

```
using CarRentalManagement.RentalManagementService.DataAccessLayer.Models;
using Microsoft.EntityFrameworkCore;
namespace CarRentalManagement.RentalManagementService.DataAccessLayer.Repositories
{
    5 references
    public interface ICustomerRepository
    {
        6 references
        Customer GetCustomer(int cusId);
        2 references
        IEnumerable<Customer> GetAllCustomers();
        2 references
        void AddCustomer(Customer customer);
        2 references
        void UpdateCustomer(Customer customer);
        2 references
        void DeleteCustomer(Customer customer);
        1 reference
        bool CustomerExists(int customerId);
        4 references
        void Save();
    }
}
```

Figure 4.2.24 - This figure shows the *ICustomerRepository* which is an interface for the implementation for CustomerRepository

RentalManagementService Microservice –>Data Access Layer-> Repositories-> IRentalRepository.cs

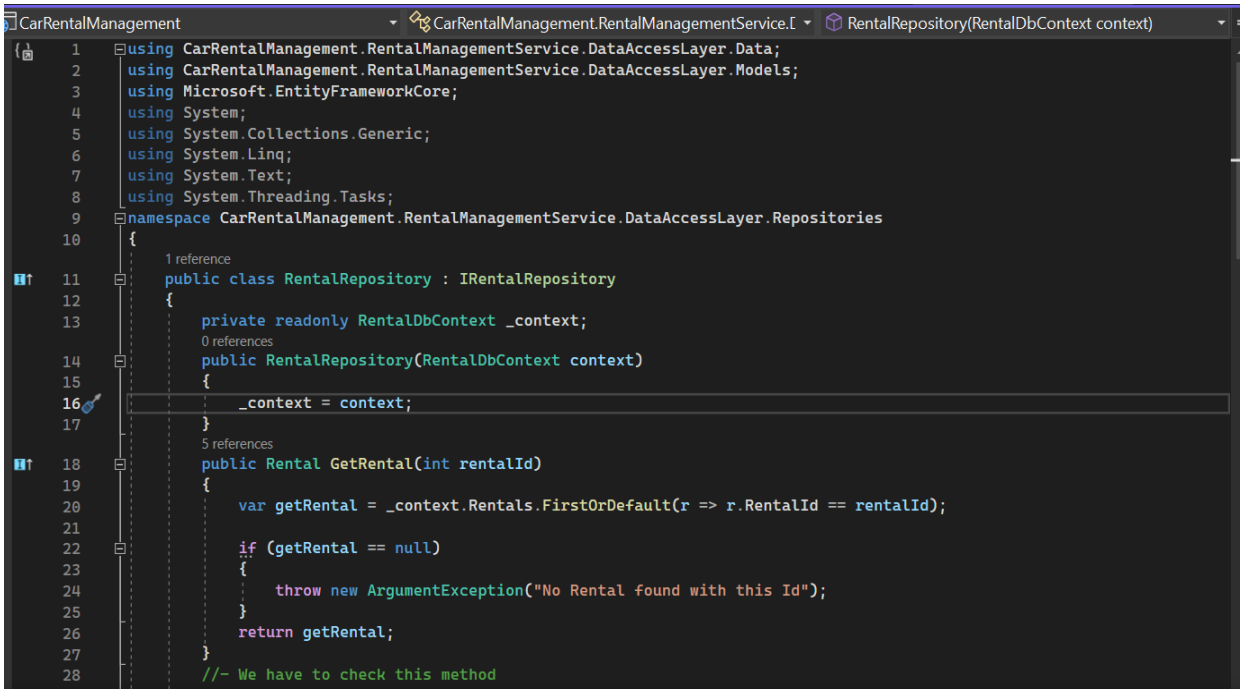
RentalCar.cs | CustomerRepository.cs | ICustomerRepository.cs | IRentalCarRepository.cs | IRentalRepository.cs

CarRentalManagement | CarRentalManagement.RentalManagementService.D | AddRental(Rental rental)

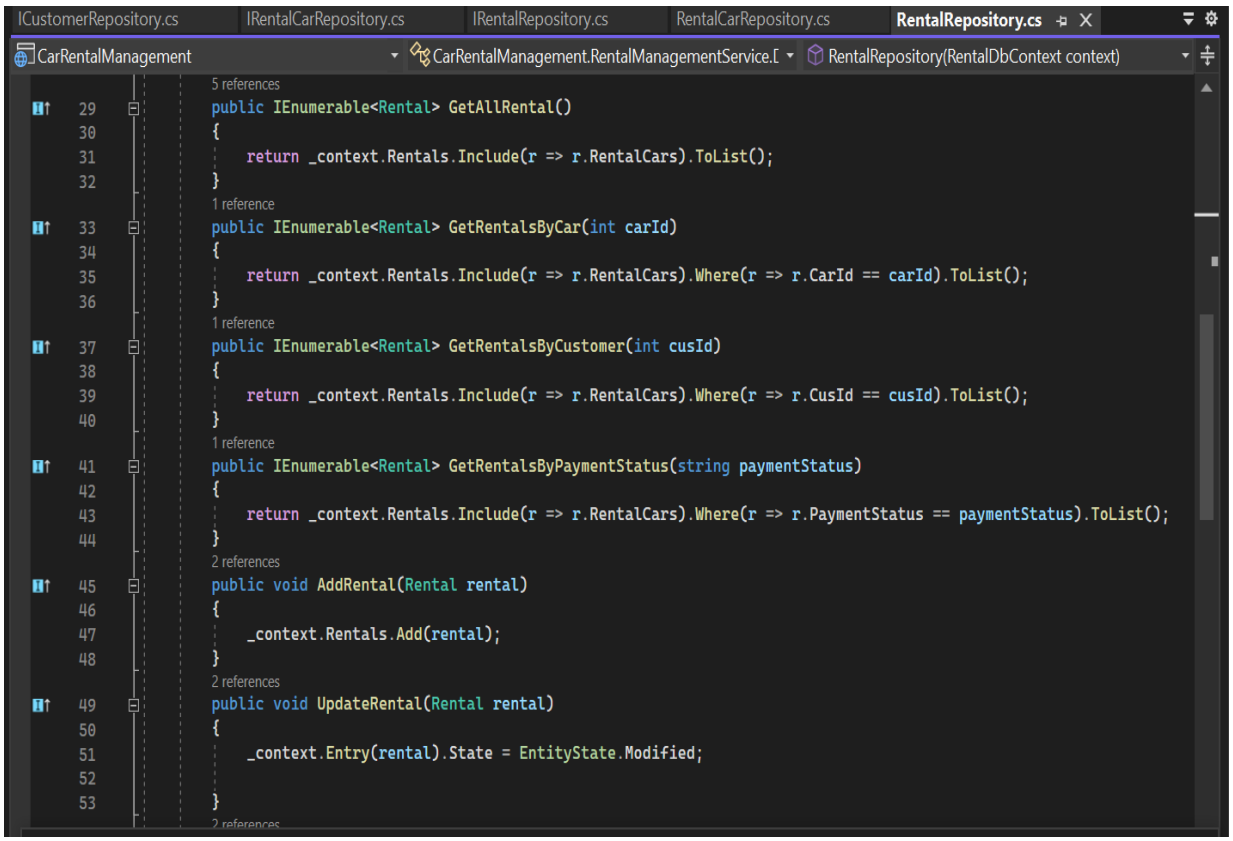
```
using CarRentalManagement.RentalManagementService.DataAccessLayer.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace CarRentalManagement.RentalManagementService.DataAccessLayer.Repositories
{
    3 references
    public interface IRentalRepository
    {
        5 references
        Rental GetRental(int rentalId);
        5 references
        IEnumerable<Rental> GetAllRental();
        1 reference
        IEnumerable<Rental> GetRentalsByCar(int carId);
        1 reference
        IEnumerable<Rental> GetRentalsByCustomer(int cusId);
        1 reference
        IEnumerable<Rental> GetRentalsByPaymentStatus(string paymentStatus);
        2 references
        void AddRental(Rental rental);
        2 references
        void UpdateRental(Rental rental);
        2 references
        void DeleteRental(Rental rental);
        1 reference
        bool RentalExists(int rentalId);
        4 references
        void Save();
    }
```

Figure 4.2.25 - This figure shows the *IRentalRepository* which is an interface for the implementation for RentalRepository.

RentalManagementService Microservice –>Data Access Layer-> Repositories-> RentalRepository.cs



```
CarRentalManagement | CarRentalManagement.RentalManagementService.D | RentalRepository(RentalDbContext context)

using CarRentalManagement.RentalManagementService.DataAccessLayer.Data;
using CarRentalManagement.RentalManagementService.DataAccessLayer.Models;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace CarRentalManagement.RentalManagementService.DataAccessLayer.Repositories
{
    1 reference
    public class RentalRepository : IRentalRepository
    {
        private readonly RentalDbContext _context;
        0 references
        public RentalRepository(RentalDbContext context)
        {
            _context = context;
        }
        5 references
        public Rental GetRental(int rentalId)
        {
            var getRental = _context.Rentals.FirstOrDefault(r => r.RentalId == rentalId);

            if (getRental == null)
            {
                throw new ArgumentException("No Rental found with this Id");
            }
            return getRental;
        }
        //- We have to check this method
```



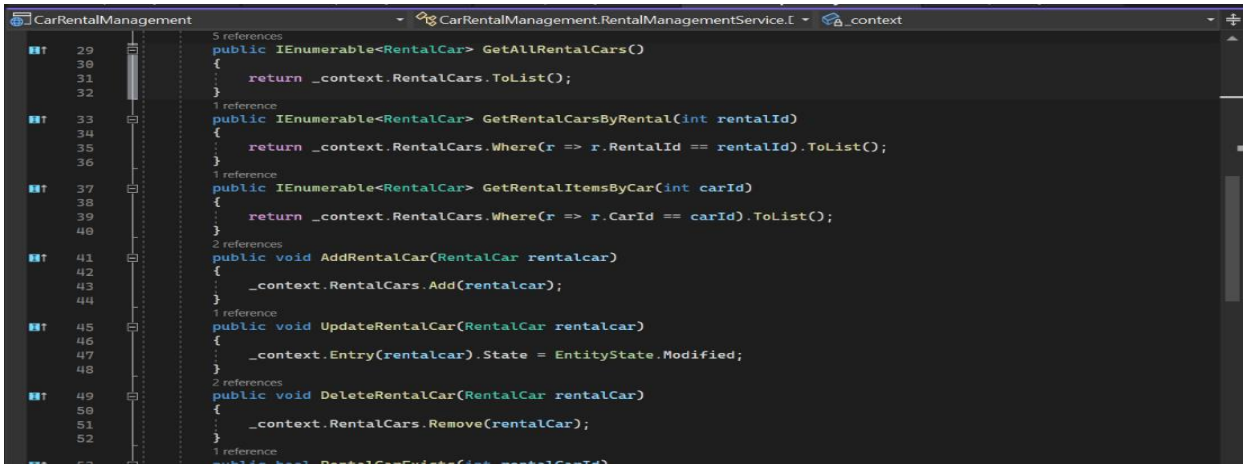
```
ICustomerRepository.cs | IRentalCarRepository.cs | IRentalRepository.cs | RentalCarRepository.cs | RentalRepository.cs
CarRentalManagement | CarRentalManagement.RentalManagementService.D | RentalRepository(RentalDbContext context)

    5 references
    public IEnumerable<Rental> GetAllRental()
    {
        return _context.Rentals.Include(r => r.RentalCars).ToList();
    }
    1 reference
    public IEnumerable<Rental> GetRentalsByCar(int carId)
    {
        return _context.Rentals.Include(r => r.RentalCars).Where(r => r.CarId == carId).ToList();
    }
    1 reference
    public IEnumerable<Rental> GetRentalsByCustomer(int cusId)
    {
        return _context.Rentals.Include(r => r.RentalCars).Where(r => r.CusId == cusId).ToList();
    }
    1 reference
    public IEnumerable<Rental> GetRentalsByPaymentStatus(string paymentStatus)
    {
        return _context.Rentals.Include(r => r.RentalCars).Where(r => r.PaymentStatus == paymentStatus).ToList();
    }
    2 references
    public void AddRental(Rental rental)
    {
        _context.Rentals.Add(rental);
    }
    2 references
    public void UpdateRental(Rental rental)
    {
        _context.Entry(rental).State = EntityState.Modified;

    }
    2 references
```

```
CarRentalManagement | CarRentalManagement.RentalManagementService.[ | RentalRepository(RentalDbContext context)
42          {
43              return _context.Rentals.Include(r => r.RentalCars).Where(r => r.PaymentStatus == paymentStatus).ToList();
44          }
            2 references
45          public void AddRental(Rental rental)
46          {
47              _context.Rentals.Add(rental);
48          }
            2 references
49          public void UpdateRental(Rental rental)
50          {
51              _context.Entry(rental).State = EntityState.Modified;
52
53          }
            2 references
54          public void DeleteRental(Rental rental)
55          {
56              _context.Rentals.Remove(rental);
57          }

58          public bool RentalExists(int rentalId)
59          {
60              return _context.Rentals.Any(r => r.RentalId == rentalId);
61          }
            4 references
62          public void Save()
63          {
64              _context.SaveChanges();
65          }
66      }
```

Figure 4.2.26-4.2.28: This figures shows the *RentalRepository* file which comprises classes or methods that encapsulates all data logic.

RentalManagementService Microservice –>Data Access Layer-> Repositories-> RentalCarRepository.cs

```
CustomerRepository.cs | ICustomerRepository.cs | IRentalCarRepository.cs | IRentalRepository.cs | RentalCarRepository.cs
CarRentalManagement | CarRentalManagement.RentalManagementService.[ | _context
1   using CarRentalManagement.RentalManagementService.DataAccessLayer.Models;
2   using CarRentalManagement.RentalManagementService.DataAccessLayer.Data;
3   using Microsoft.EntityFrameworkCore;
4   using System;
5   using System.Collections.Generic;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9   namespace CarRentalManagement.RentalManagementService.DataAccessLayer.Repositories
10  {
        1 reference
11      public class RentalCarRepository : IRentalCarRepository
12      {
13          private readonly RentalDbContext _context;
            0 references
14          public RentalCarRepository(RentalDbContext context)
15          {
16              _context = context;
17          }
            5 references
18          public RentalCar GetRentalCar(int rentalCarId)
19          {
20
21              var getRental = _context.RentalCars.FirstOrDefault(r => r.RentalCarId == rentalCarId);
22
23              if (getRental == null)
24              {
25                  throw new ArgumentException("No Rental item found with this Id");
26              }
27              return getRental;
28          }
```



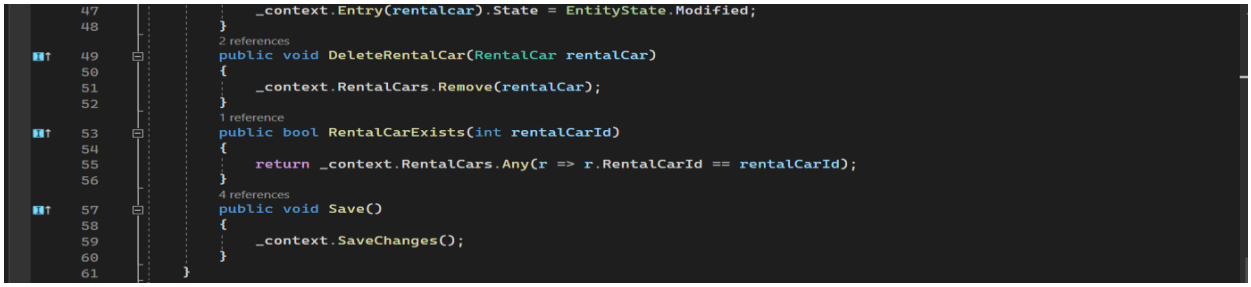


Figure 4.2.29-4.2.31: This figures shows the *RentalCarRepository* file which comprises classes or methods that encapsulates all data logic.

RentalManagementService Microservice –>Data Access Layer-> Repositories-> IRentalCarRepository.cs

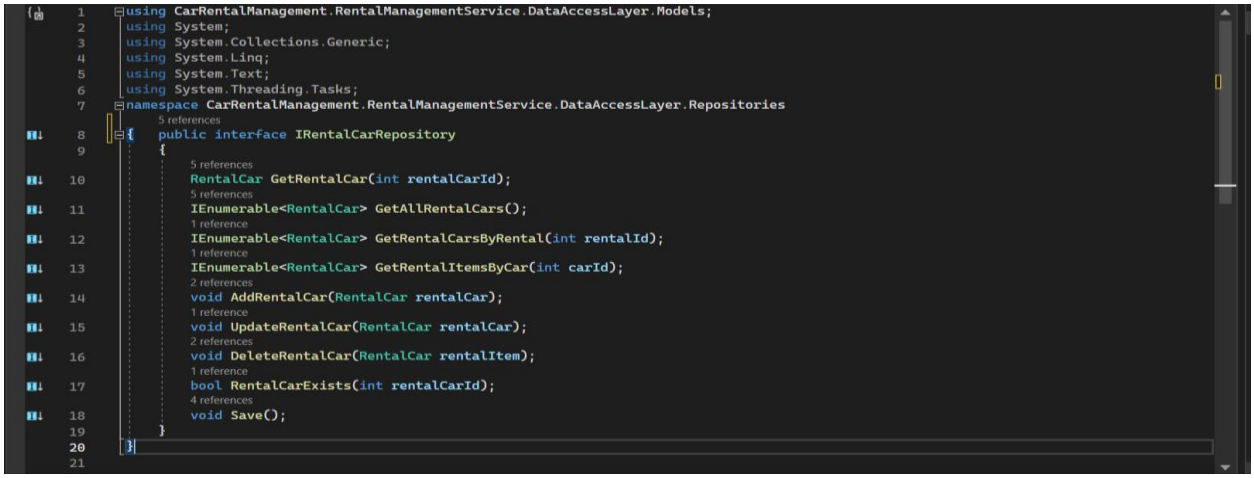


Figure 4.2.32 - This figure shows the *IRentalCarRepository* which is an interface for the implementation for RentalCarRepository.

RentalManagementService Microservice –>Data Access Layer-> Data-> RentalDbContext.cs

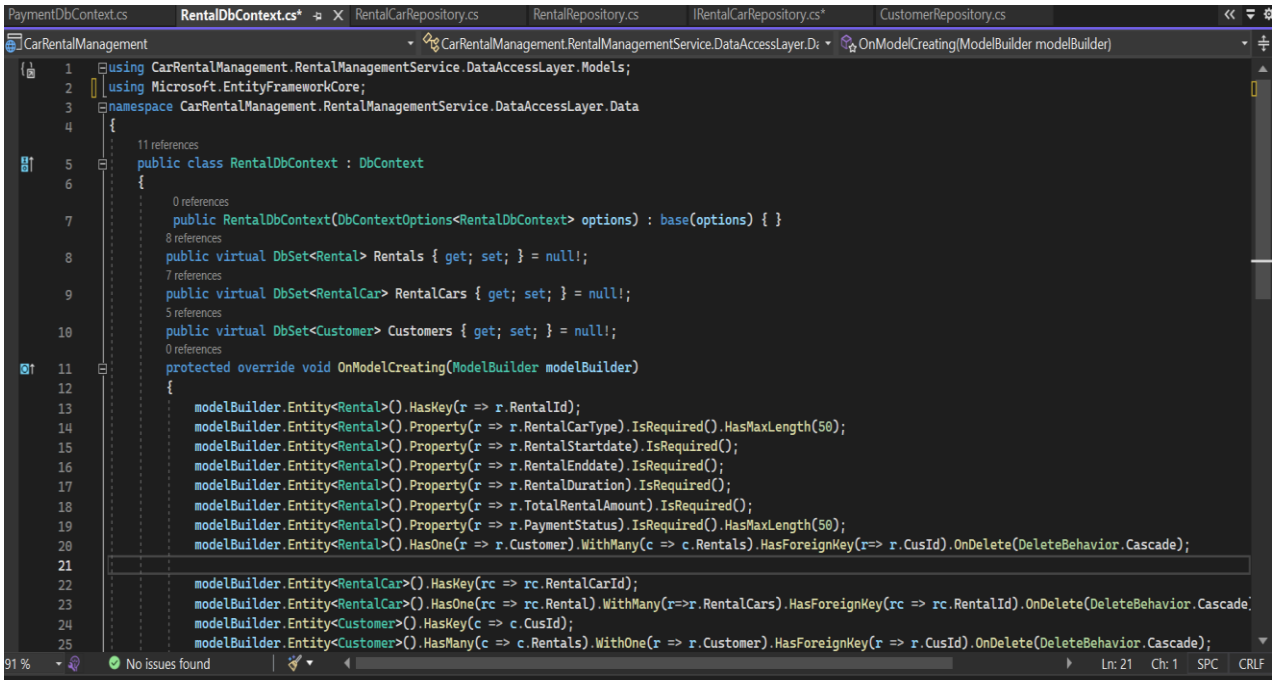
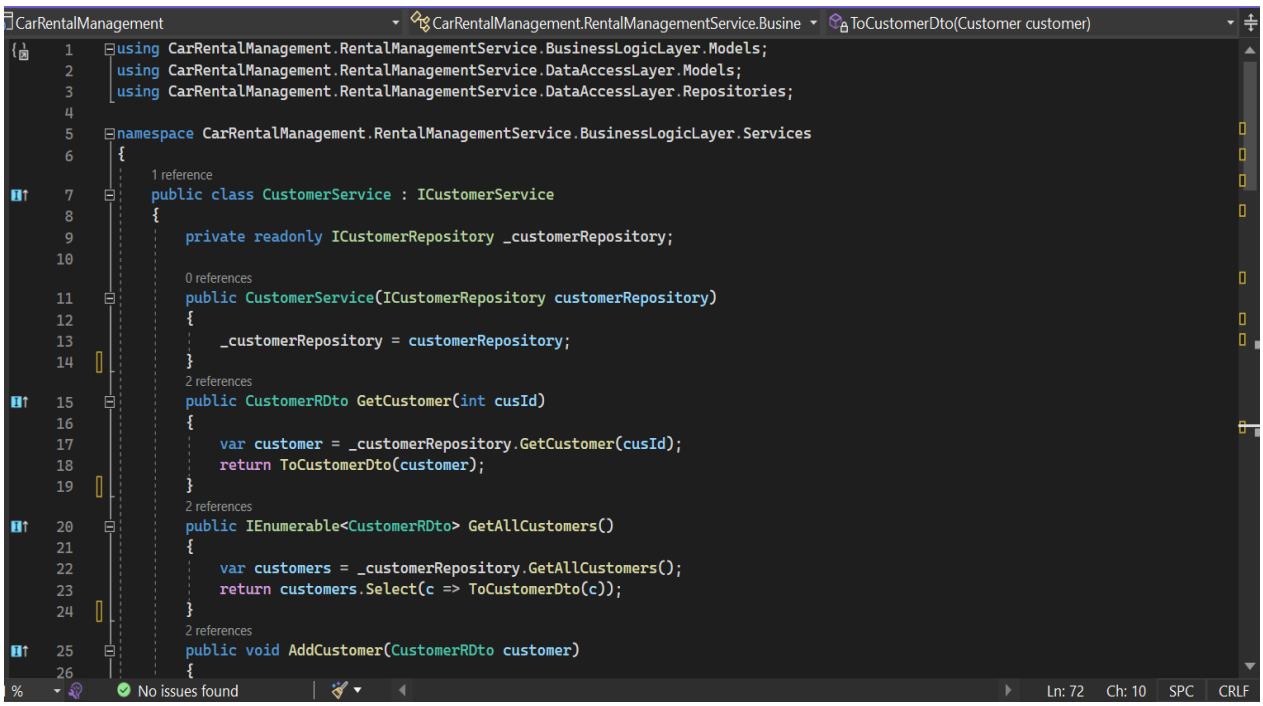
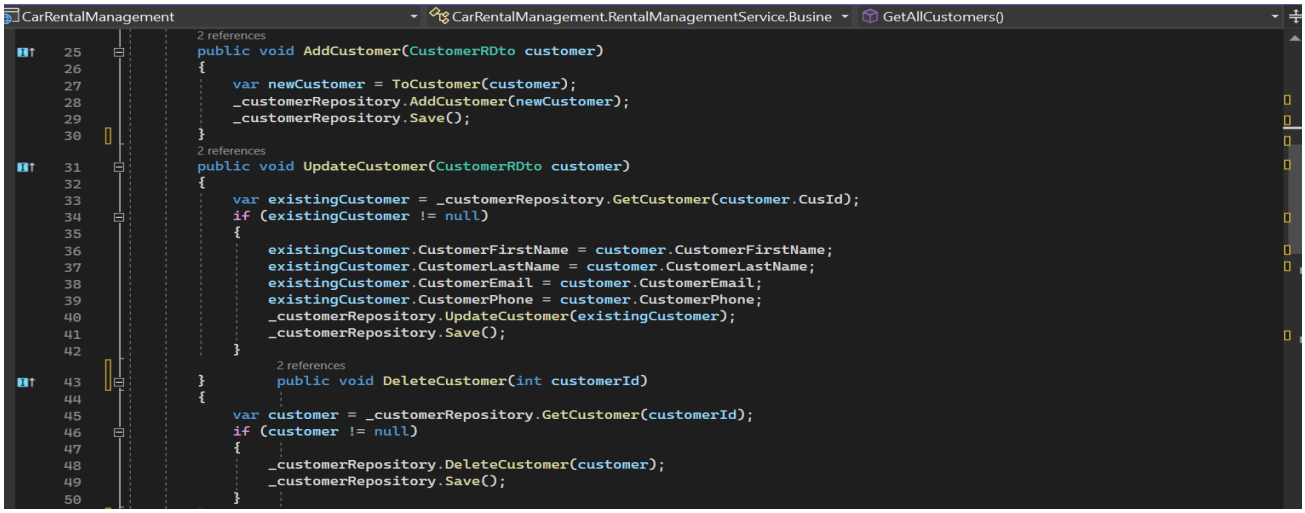
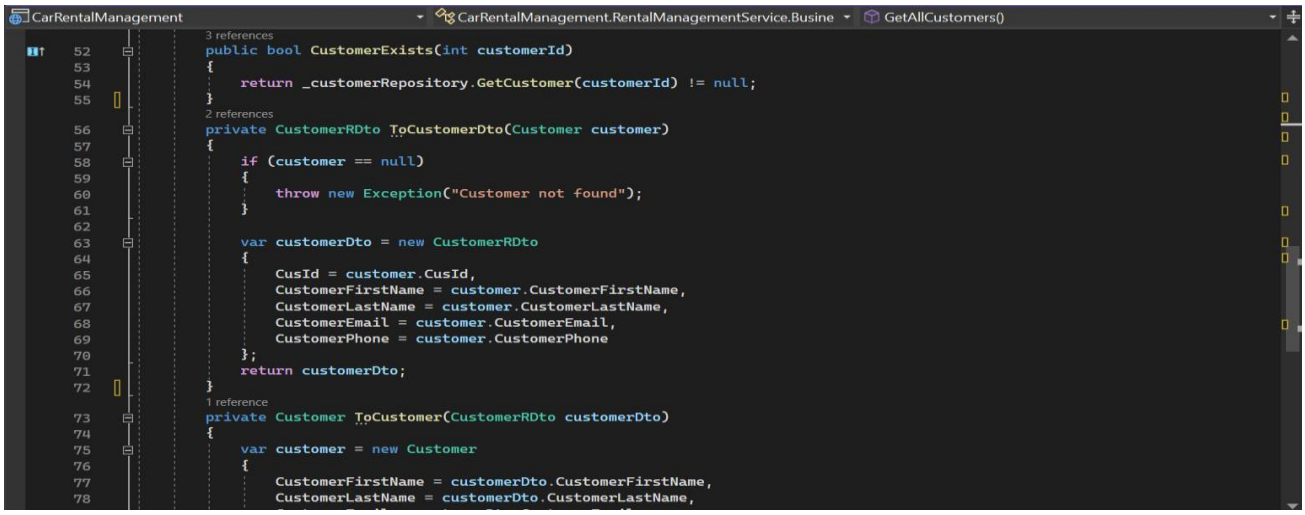


Figure 4.2.33-This figure depicts the Database of *RentalDbContext* which also makes a table *Rentals, RentalCars, Customers table* and then show the results that each of the attributes has.

RentalManagement Microservice –> Business Layer ->Services -> CustomerService.cs







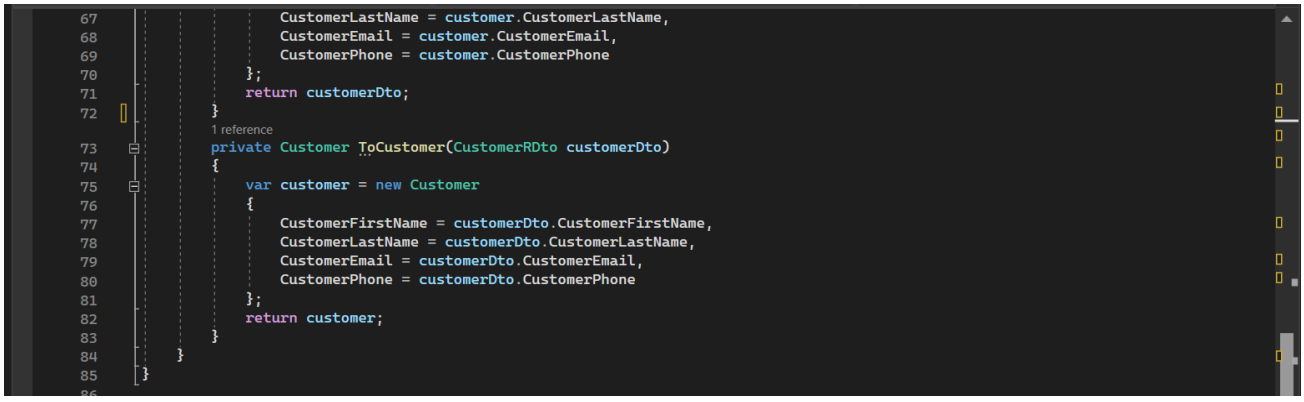
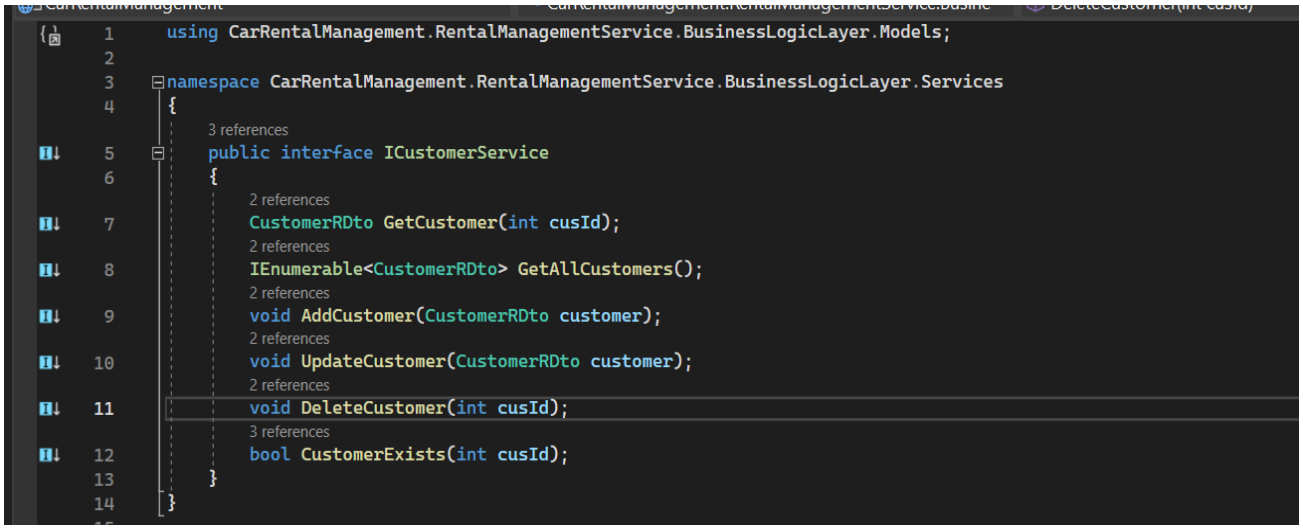


Figure 4.2.34- 4.2.37 These figures depicts the *CustomerService* include the business services, the services that are in charge of carrying out the program's business logic. The services themselves can interface with the presentation layer and the data access layer in a standardised way by using Data Transfer Objects (DTOs) in the services.

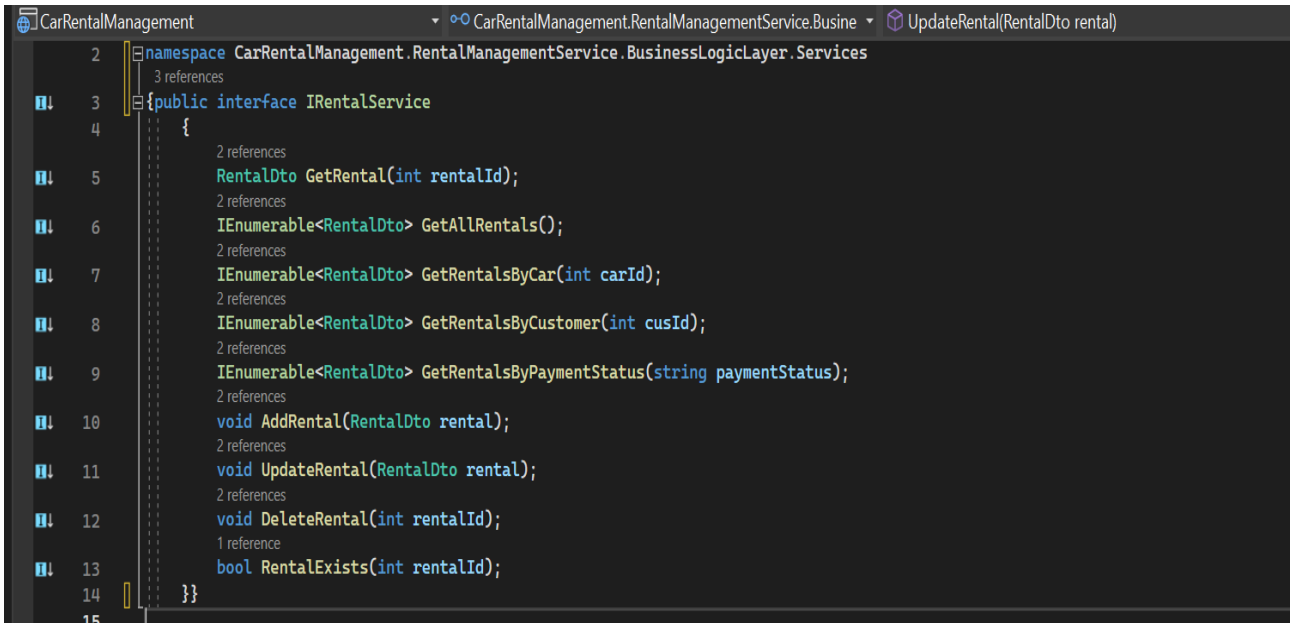
RentalManagement Microservice –> Business Layer ->Services -> ICustomerService.cs



```
using CarRentalManagement.RentalManagementService.BusinessLogicLayer.Models;

namespace CarRentalManagement.RentalManagementService.BusinessLogicLayer.Services
{
    3 references
    public interface ICustomerService
    {
        2 references
        CustomerRDto GetCustomer(int cusId);
        2 references
        IEnumerable<CustomerRDto> GetAllCustomers();
        2 references
        void AddCustomer(CustomerRDto customer);
        2 references
        void UpdateCustomer(CustomerRDto customer);
        2 references
        void DeleteCustomer(int cusId);
        3 references
        bool CustomerExists(int cusId);
    }
}
```

Figure 4.2.38 This figure shows the *ICustomerService* which is used in order to give the layer that presents data the necessary features, the business logic layer needs to implement a set of functions that are defined by the interface.

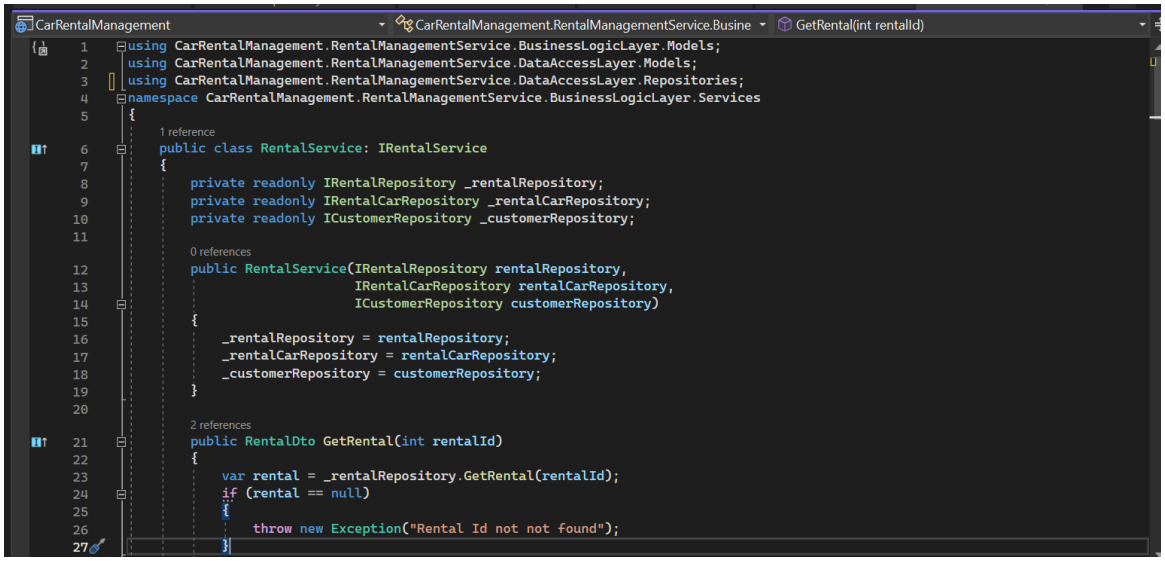
RentalManagement Microservice –> Business Layer ->Services -> IRentalService.cs



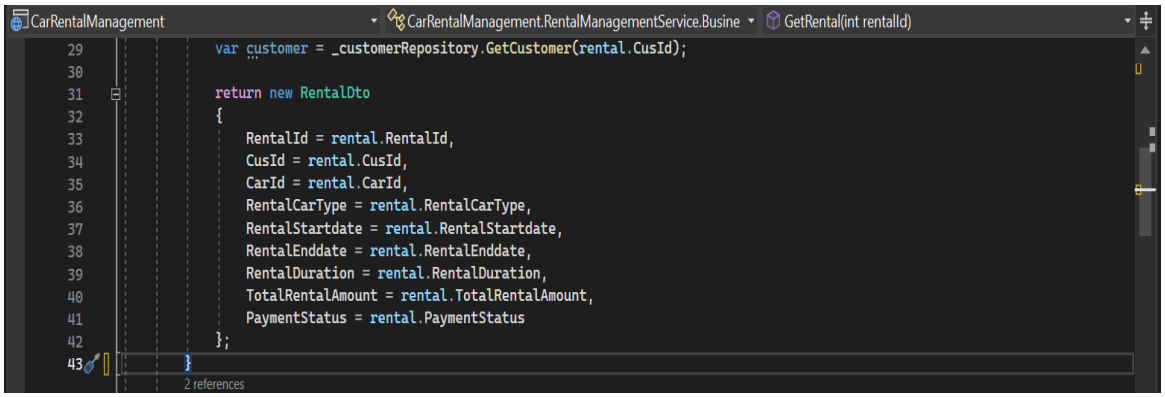
```
CarRentalManagement  CarRentalManagement.RentalManagementService.Busine  UpdateRental(RentalDto rental)

namespace CarRentalManagement.RentalManagementService.BusinessLogicLayer.Services
    3 references
{public interface IRentalService
    {
        2 references
        RentalDto GetRental(int rentalId);
        2 references
        IEnumerable<RentalDto> GetAllRentals();
        2 references
        IEnumerable<RentalDto> GetRentalsByCar(int carId);
        2 references
        IEnumerable<RentalDto> GetRentalsByCustomer(int cusId);
        2 references
        IEnumerable<RentalDto> GetRentalsByPaymentStatus(string paymentStatus);
        2 references
        void AddRental(RentalDto rental);
        2 references
        void UpdateRental(RentalDto rental);
        2 references
        void DeleteRental(int rentalId);
        1 reference
        bool RentalExists(int rentalId);
    }}
```

Figure 4.2.39 This figure shows the *IRentalService* which is used in order to give the layer that presents data the necessary features, the business logic layer needs to implement a set of functions that are defined by the interface

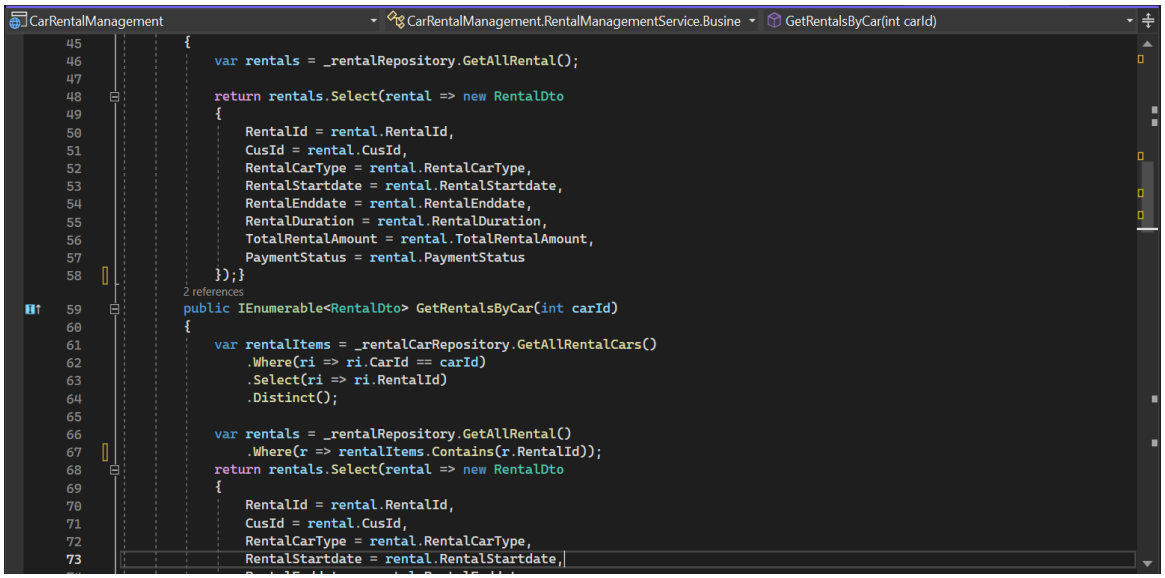
RentalManagement Microservice –> Business Layer ->Services -> RentalService.cs



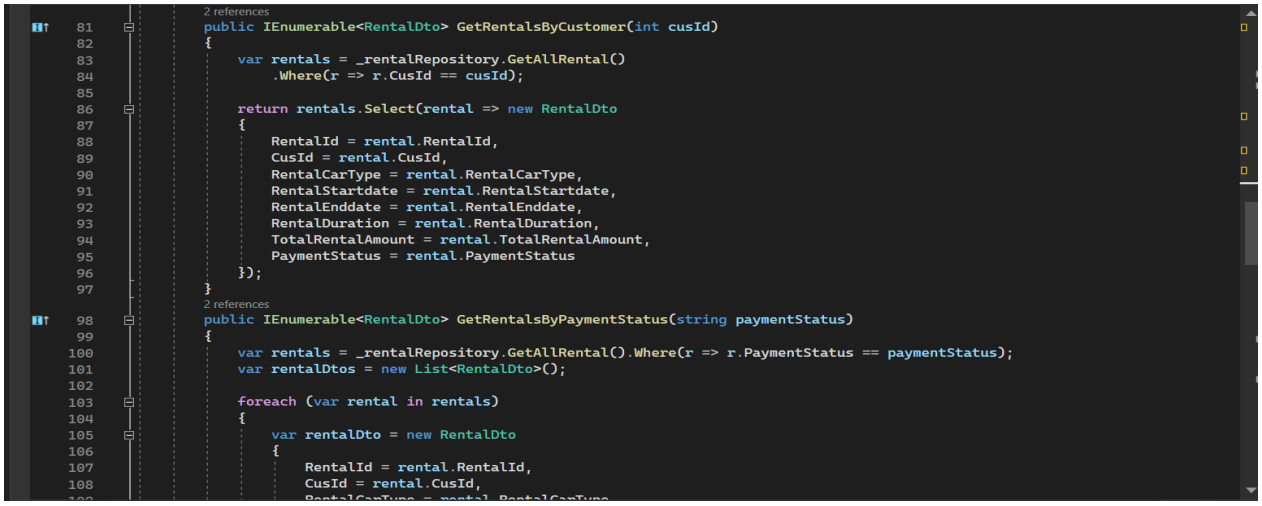

```
using CarRentalManagement.RentalManagementService.BusinessLogicLayer.Models;
using CarRentalManagement.RentalManagementService.DataAccessLayer.Models;
using CarRentalManagement.RentalManagementService.DataAccessLayer.Repositories;
namespace CarRentalManagement.RentalManagementService.BusinessLogicLayer.Services
{
    1 reference
    public class RentalService: IRentalService
    {
        private readonly IRentalRepository _rentalRepository;
        private readonly IRentalCarRepository _rentalCarRepository;
        private readonly ICustomerRepository _customerRepository;

        0 references
        public RentalService(IRentalRepository rentalRepository,
                             IRentalCarRepository rentalCarRepository,
                             ICustomerRepository customerRepository)
        {
            _rentalRepository = rentalRepository;
            _rentalCarRepository = rentalCarRepository;
            _customerRepository = customerRepository;
        }

        2 references
        public RentalDto GetRental(int rentalId)
        {
            var rental = _rentalRepository.GetRental(rentalId);
            if (rental == null)
            {
                throw new Exception("Rental Id not not found");
            }
```



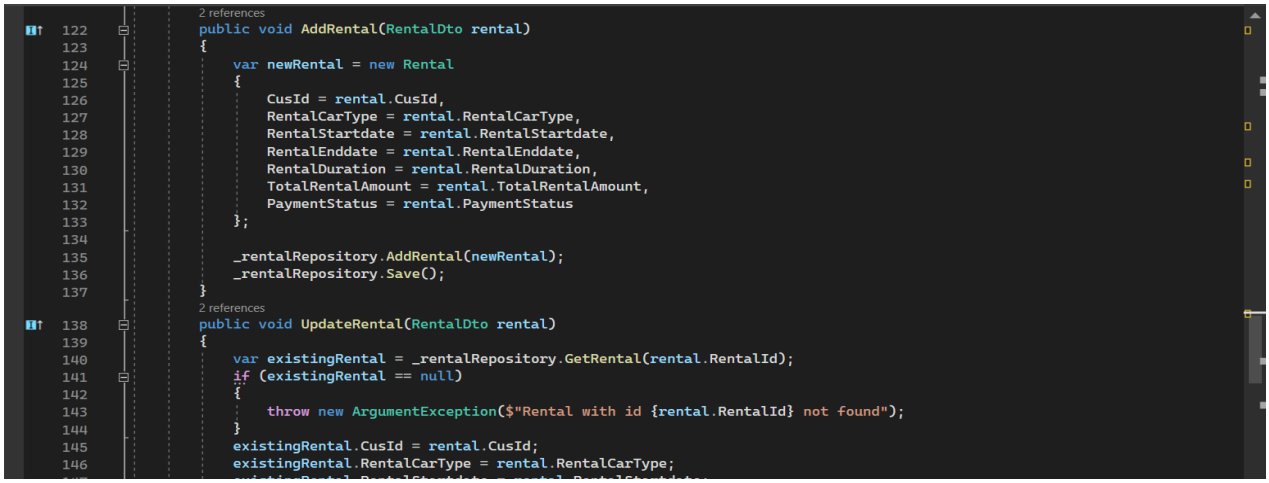

```
            var customer = _customerRepository.GetCustomer(rental.CusId);

            return new RentalDto
            {
                RentalId = rental.RentalId,
                CusId = rental.CusId,
                CarId = rental.CarId,
                RentalCarType = rental.RentalCarType,
                RentalStartdate = rental.RentalStartdate,
                RentalEnddate = rental.RentalEnddate,
                RentalDuration = rental.RentalDuration,
                TotalRentalAmount = rental.TotalRentalAmount,
                PaymentStatus = rental.PaymentStatus
            };
        }
        2 references
```



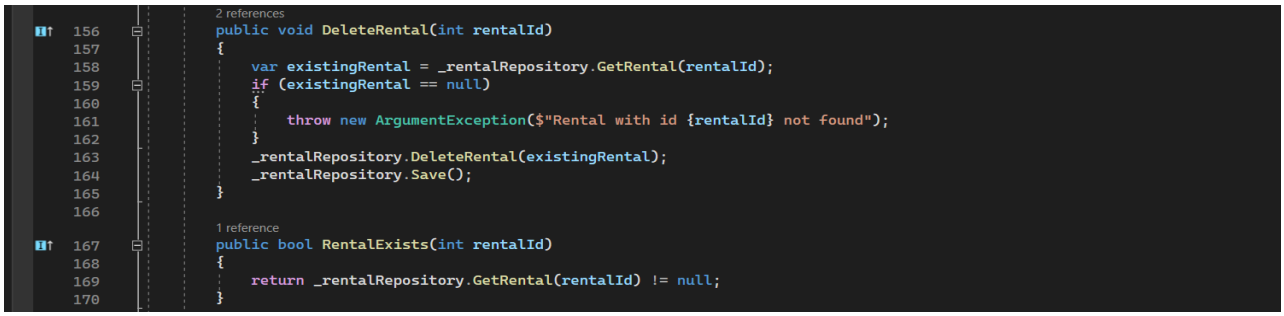

```
        {
            var rentals = _rentalRepository.GetAllRental();

            return rentals.Select(rental => new RentalDto
            {
                RentalId = rental.RentalId,
                CusId = rental.CusId,
                RentalCarType = rental.RentalCarType,
                RentalStartdate = rental.RentalStartdate,
                RentalEnddate = rental.RentalEnddate,
                RentalDuration = rental.RentalDuration,
                TotalRentalAmount = rental.TotalRentalAmount,
                PaymentStatus = rental.PaymentStatus
            });}
        2 references
        public IEnumerable<RentalDto> GetRentalsByCar(int carId)
        {
            var rentalItems = _rentalCarRepository.GetAllRentalCars()
                .Where(ri => ri.CarId == carId)
                .Select(ri => ri.RentalId)
                .Distinct();

            var rentals = _rentalRepository.GetAllRental()
                .Where(r => rentalItems.Contains(r.RentalId));
            return rentals.Select(rental => new RentalDto
            {
                RentalId = rental.RentalId,
                CusId = rental.CusId,
                RentalCarType = rental.RentalCarType,
                RentalStartdate = rental.RentalStartdate,
```



```
2 references
public IEnumerable<RentalDto> GetRentalsByCustomer(int cusId)
{
    var rentals = _rentalRepository.GetAllRental()
        .Where(r => r.CusId == cusId);

    return rentals.Select(rental => new RentalDto
    {
        RentalId = rental.RentalId,
        CusId = rental.CusId,
        RentalCarType = rental.RentalCarType,
        RentalStartdate = rental.RentalStartdate,
        RentalEnddate = rental.RentalEnddate,
        RentalDuration = rental.RentalDuration,
        TotalRentalAmount = rental.TotalRentalAmount,
        PaymentStatus = rental.PaymentStatus
    });
}
2 references
public IEnumerable<RentalDto> GetRentalsByPaymentStatus(string paymentStatus)
{
    var rentals = _rentalRepository.GetAllRental().Where(r => r.PaymentStatus == paymentStatus);
    var rentalDtos = new List<RentalDto>();

    foreach (var rental in rentals)
    {
        var rentalDto = new RentalDto
        {
            RentalId = rental.RentalId,
            CusId = rental.CusId,
```



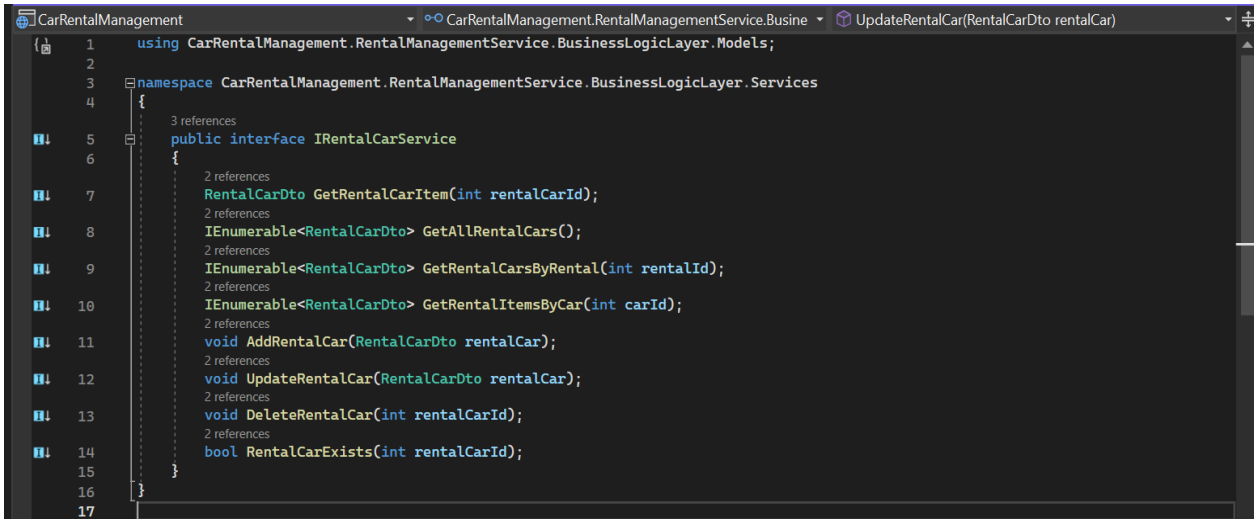
```
2 references
public void AddRental(RentalDto rental)
{
    var newRental = new Rental
    {
        CusId = rental.CusId,
        RentalCarType = rental.RentalCarType,
        RentalStartdate = rental.RentalStartdate,
        RentalEnddate = rental.RentalEnddate,
        RentalDuration = rental.RentalDuration,
        TotalRentalAmount = rental.TotalRentalAmount,
        PaymentStatus = rental.PaymentStatus
    };

    _rentalRepository.AddRental(newRental);
    _rentalRepository.Save();
}
2 references
public void UpdateRental(RentalDto rental)
{
    var existingRental = _rentalRepository.GetRental(rental.RentalId);
    if (existingRental == null)
    {
        throw new ArgumentException($"Rental with id {rental.RentalId} not found");
    }
    existingRental.CusId = rental.CusId;
    existingRental.RentalCarType = rental.RentalCarType;
```



```
2 references
public void DeleteRental(int rentalId)
{
    var existingRental = _rentalRepository.GetRental(rentalId);
    if (existingRental == null)
    {
        throw new ArgumentException($"Rental with id {rentalId} not found");
    }
    _rentalRepository.DeleteRental(existingRental);
    _rentalRepository.Save();
}

1 reference
public bool RentalExists(int rentalId)
{
    return _rentalRepository.GetRental(rentalId) != null;
}
```

Figure 4.2.40- 4.2.45 These figures depicts the *Rental Service* include the business services, the services that are in charge of carrying out the program's business logic. The services themselves can interface with the presentation layer and the data access layer in a standardised way by using Data Transfer Objects (DTOs) in the services.

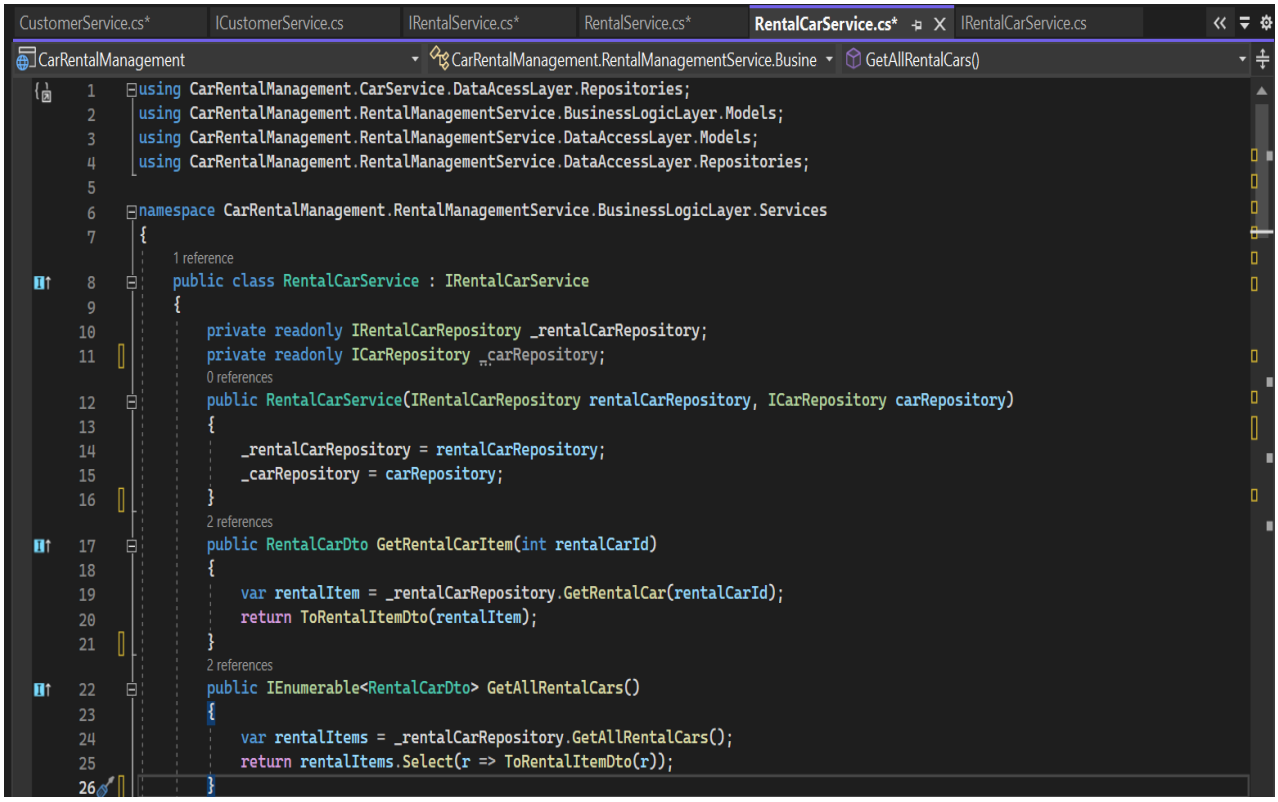
RentalManagement Microservice –> Business Layer ->Services -> IRentalCarService.cs




```
using CarRentalManagement.RentalManagementService.BusinessLogicLayer.Models;

namespace CarRentalManagement.RentalManagementService.BusinessLogicLayer.Services
{
    3 references
    public interface IRentalCarService
    {
        2 references
        RentalCarDto GetRentalCarItem(int rentalCarId);
        2 references
        IEnumerable<RentalCarDto> GetAllRentalCars();
        2 references
        IEnumerable<RentalCarDto> GetRentalCarsByRental(int rentalId);
        2 references
        IEnumerable<RentalCarDto> GetRentalItemsByCar(int carId);
        2 references
        void AddRentalCar(RentalCarDto rentalCar);
        2 references
        void UpdateRentalCar(RentalCarDto rentalCar);
        2 references
        void DeleteRentalCar(int rentalCarId);
        2 references
        bool RentalCarExists(int rentalCarId);
    }
}
```

Figure 4.2.46 This figure shows the *IRentalCarService* which is used in order to give the layer that presents data the necessary features, the business logic layer needs to implement a set of functions that are defined by the interface

RentalManagement Microservice –> Business Layer ->Services -> RentalCarService.cs




```
using CarRentalManagement.CarService.DataAcessLayer.Repositories;
using CarRentalManagement.RentalManagementService.BusinessLogicLayer.Models;
using CarRentalManagement.RentalManagementService.DataAccessLayer.Models;
using CarRentalManagement.RentalManagementService.DataAccessLayer.Repositories;

namespace CarRentalManagement.RentalManagementService.BusinessLogicLayer.Services
{
    1 reference
    public class RentalCarService : IRentalCarService
    {
        private readonly IRentalCarRepository _rentalCarRepository;
        private readonly ICarRepository _carRepository;
        0 references
        public RentalCarService(IRentalCarRepository rentalCarRepository, ICarRepository carRepository)
        {
            _rentalCarRepository = rentalCarRepository;
            _carRepository = carRepository;
        }
        2 references
        public RentalCarDto GetRentalCarItem(int rentalCarId)
        {
            var rentalItem = _rentalCarRepository.GetRentalCar(rentalCarId);
            return ToRentalItemDto(rentalItem);
        }
        2 references
        public IEnumerable<RentalCarDto> GetAllRentalCars()
        {
            var rentalItems = _rentalCarRepository.GetAllRentalCars();
            return rentalItems.Select(r => ToRentalItemDto(r));
        }
```

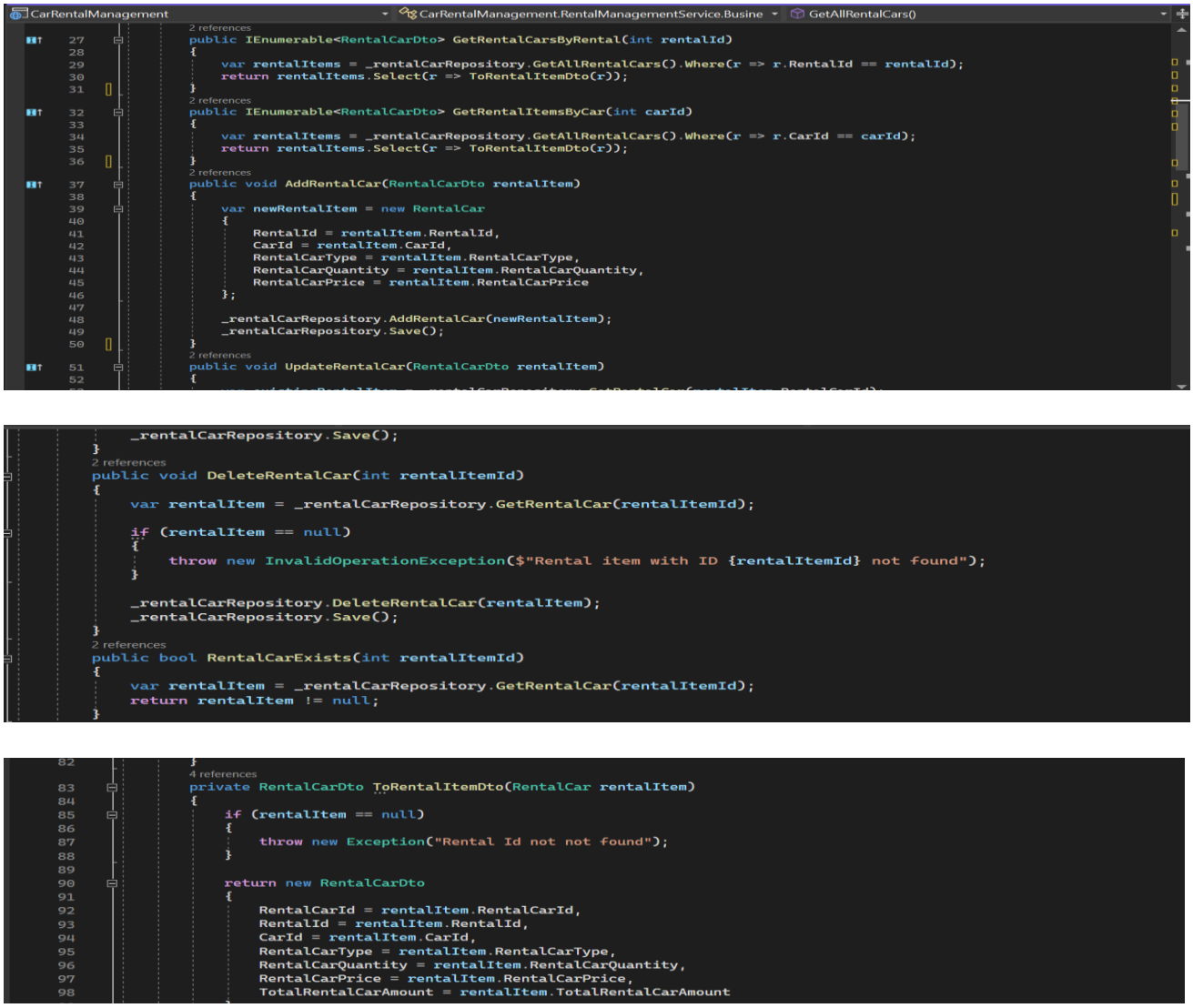


Figure 4.2.47- 4.2.50 These figures depicts the *RentalCarService* include the business services, the services that are in charge of carrying out the program's business logic

RentalManagement Microservice –> Business Layer ->Models ->CustomerRDto.cs

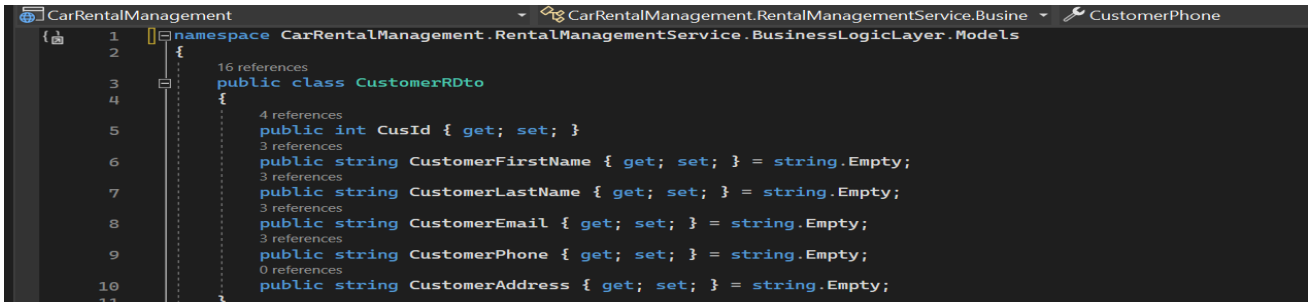


Figure 4.2.51: This figure shows the *CustomerDto* which is data transfer objects which is used to make application more modular and flexible.

RentalManagement Microservice –> Business Layer ->Models ->RentalDto.cs

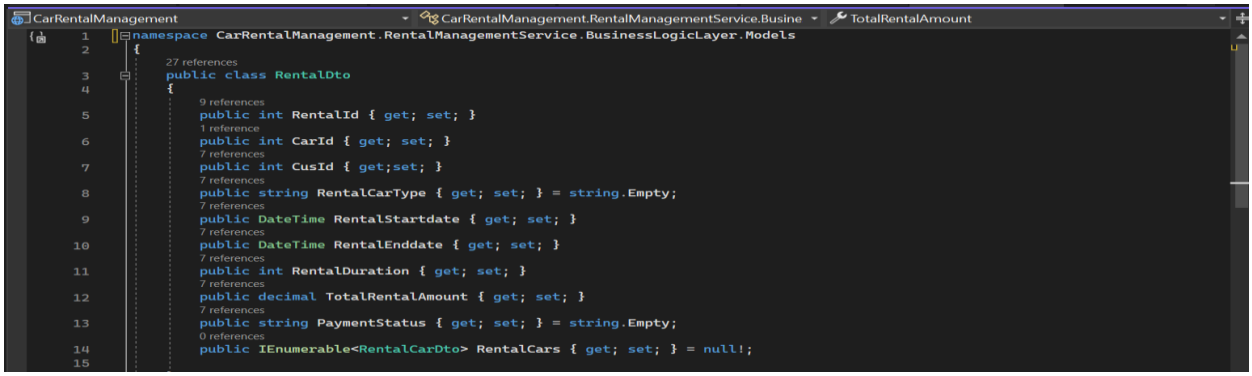


Figure 4.2.52: This figure shows the *RentalDto* which is data transfer objects which is used to make application more modular and flexible.

RentalManagement Microservice –> Business Layer ->Models ->RentalCarDto.cs

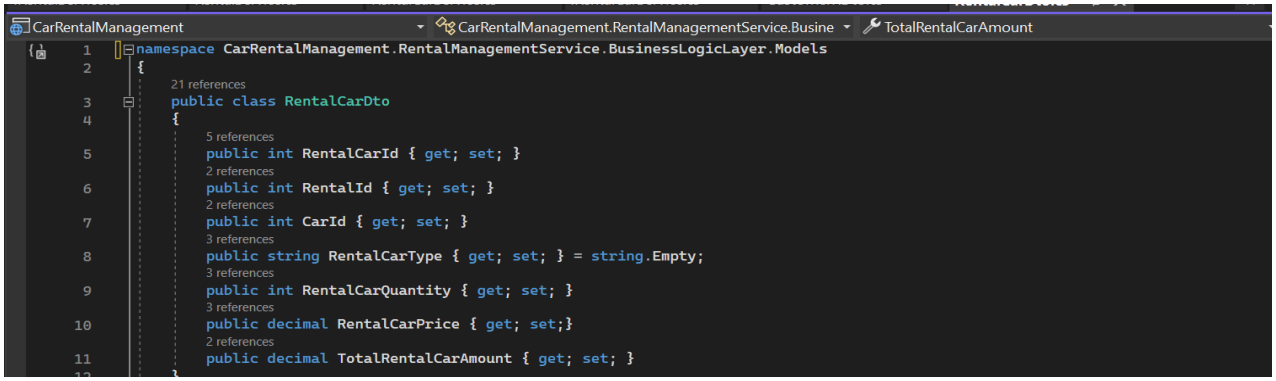
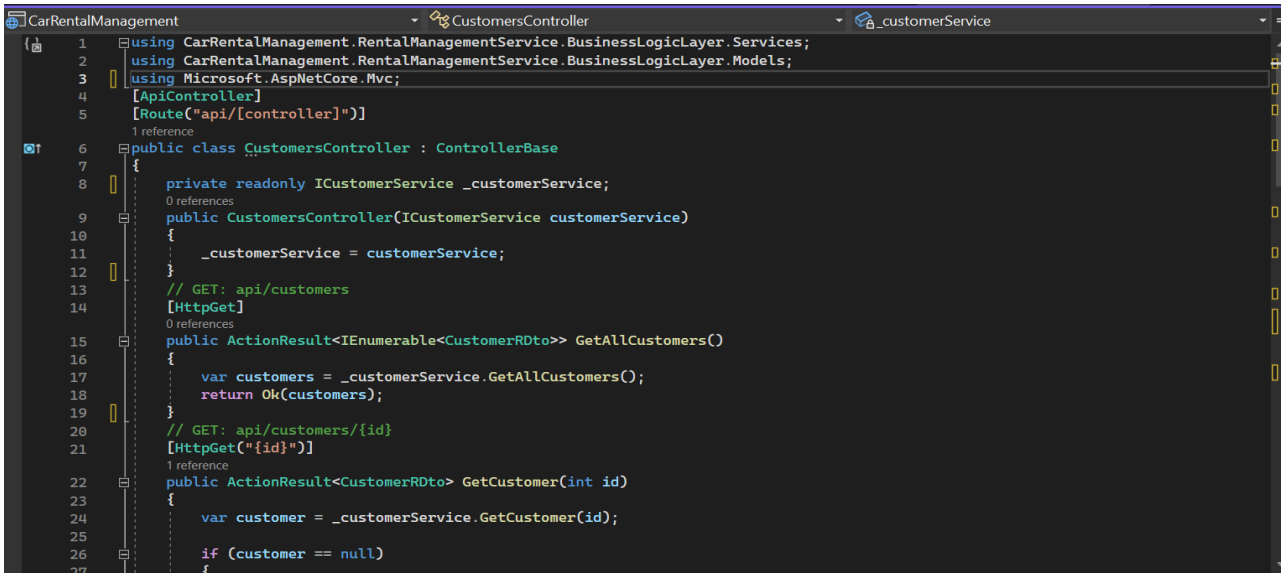
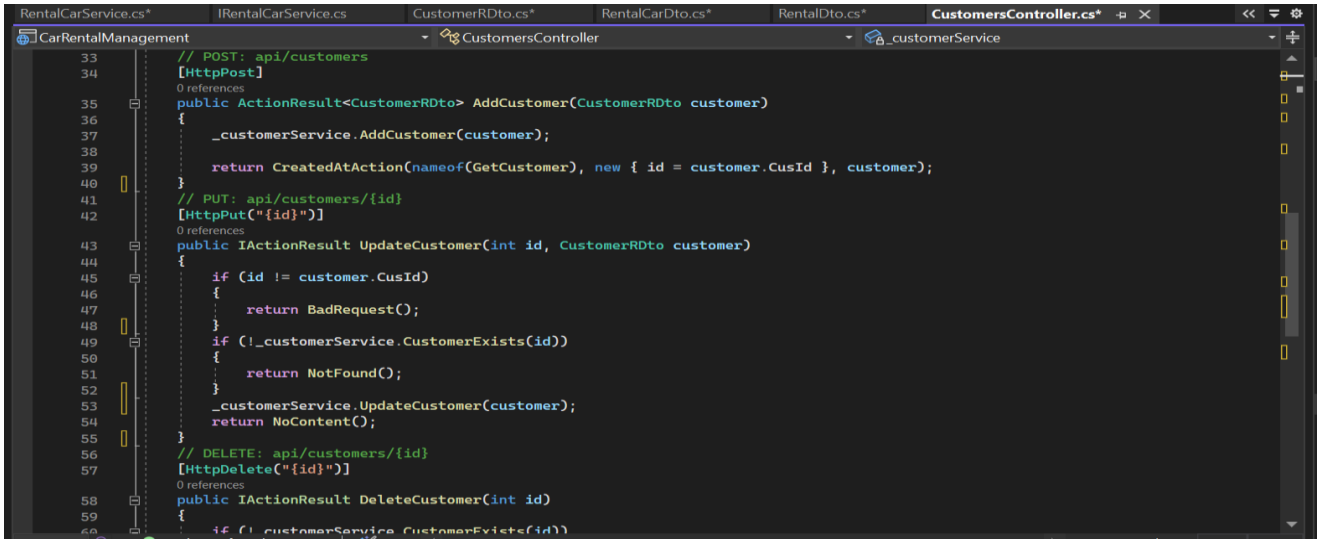


Figure 4.2.53: This figure shows the *RentalCarDto* which is data transfer objects which is used to make application more modular and flexible.

RentalManagement Microservice –>Controller-> CustomerController.cs





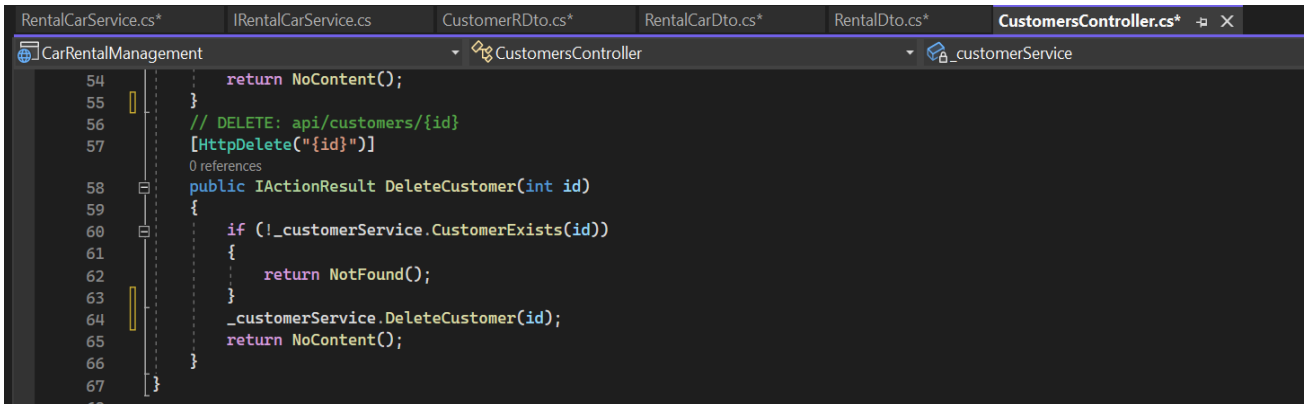


Figure 4.2.54 – 4.2.56: This figure shows the *CustomerController* is essential for handling requests that come in, sending them through the right action method, and producing a response to send back to the user.

RentalManagement Microservice –>Controller-> RentalCarController.cs

```
0 references
public ActionResult<IEnumerable<RentalCarDto>> GetAllRentalItems()
{
    var rentalItems = _rentalCarService.GetAllRentalCars();
    return Ok(rentalItems);
}
[HttpGet("rental/{rentalId}")]
0 references
public ActionResult<IEnumerable<RentalCarDto>> GetRentalItemsByRental(int rentalId)
{
    var rentalItems = _rentalCarService.GetRentalCarsByRental(rentalId);
    return Ok(rentalItems);
}
[HttpGet("car/{carId}")]
0 references
public ActionResult<IEnumerable<RentalCarDto>> GetRentalItemsByBike(int carId)
{
    var rentalItems = _rentalCarService.GetRentalItemsByCar(carId);
    return Ok(rentalItems);
}
[HttpPost]
0 references
public ActionResult AddRentalItem(RentalCarDto rentalItem)
{
    _rentalCarService.AddRentalCar(rentalItem);
    return CreatedAtAction(nameof(GetRentalItem), new { rentalItemId = rentalItem.RentalCarId }, rentalItem);
}
[HttpPut("{rentalCarId}")]
0 references
public ActionResult UpdateRentalItem(int rentalCarId, RentalCarDto rentalItem)
{
```

```
    }
    [HttpPut("{rentalCarId}")]
    0 references
    public ActionResult UpdateRentalItem(int rentalCarId, RentalCarDto rentalItem)
    {
        if (rentalCarId != rentalItem.RentalCarId)
        {
            return BadRequest();
        }
        _rentalCarService.UpdateRentalCar(rentalItem);
        return NoContent();
    }
    [HttpDelete("{rentalCarId}")]
    0 references
    public ActionResult DeleteRentalItem(int rentalCarId)
    {
        if (!_rentalCarService.RentalCarExists(rentalCarId))
        {
            return NotFound();
        }

        _rentalCarService.DeleteRentalCar(rentalCarId);
        return NoContent();
    }
```

Figure 4.2.57 – 4.2.59: This figure shows the *RentalsCarController* is essential for handling requests that come in, sending them through the right action method, and producing a response to send back to the user.

RentalManagement Microservice –>Controller-> RentalsController.cs

```
CustomerRDto.cs*  RentalCarDto.cs*  RentalDto.cs*  CustomersController.cs*  RentalCarsController.cs*  RentalsController.cs
CarRentalManagement  CarRentalManagement.RentalManagementService.Contrc  UpdateRental(int rentalId, RentalDto rentalDto
using CarRentalManagement.RentalManagementService.BusinessLogicLayer.Models;
using CarRentalManagement.RentalManagementService.BusinessLogicLayer.Services;
using Microsoft.AspNetCore.Mvc;

namespace CarRentalManagement.RentalManagementService.Controller
{
    [ApiController]
    [Route("/api/[controller]")]
    1 reference
    public class RentalsController : ControllerBase
    {
        private readonly IRentalService _rentalService;

        0 references
        public RentalsController(IRentalService rentalService)
        {
            _rentalService = rentalService;
        }

        [HttpGet("{rentalId}")]
        1 reference
        public ActionResult<RentalDto> GetRental(int rentalId)
        {
            var rental = _rentalService.GetRental(rentalId);
            if (rental == null)
            {
                return NotFound();
            }
            return Ok(rental);
        }
```

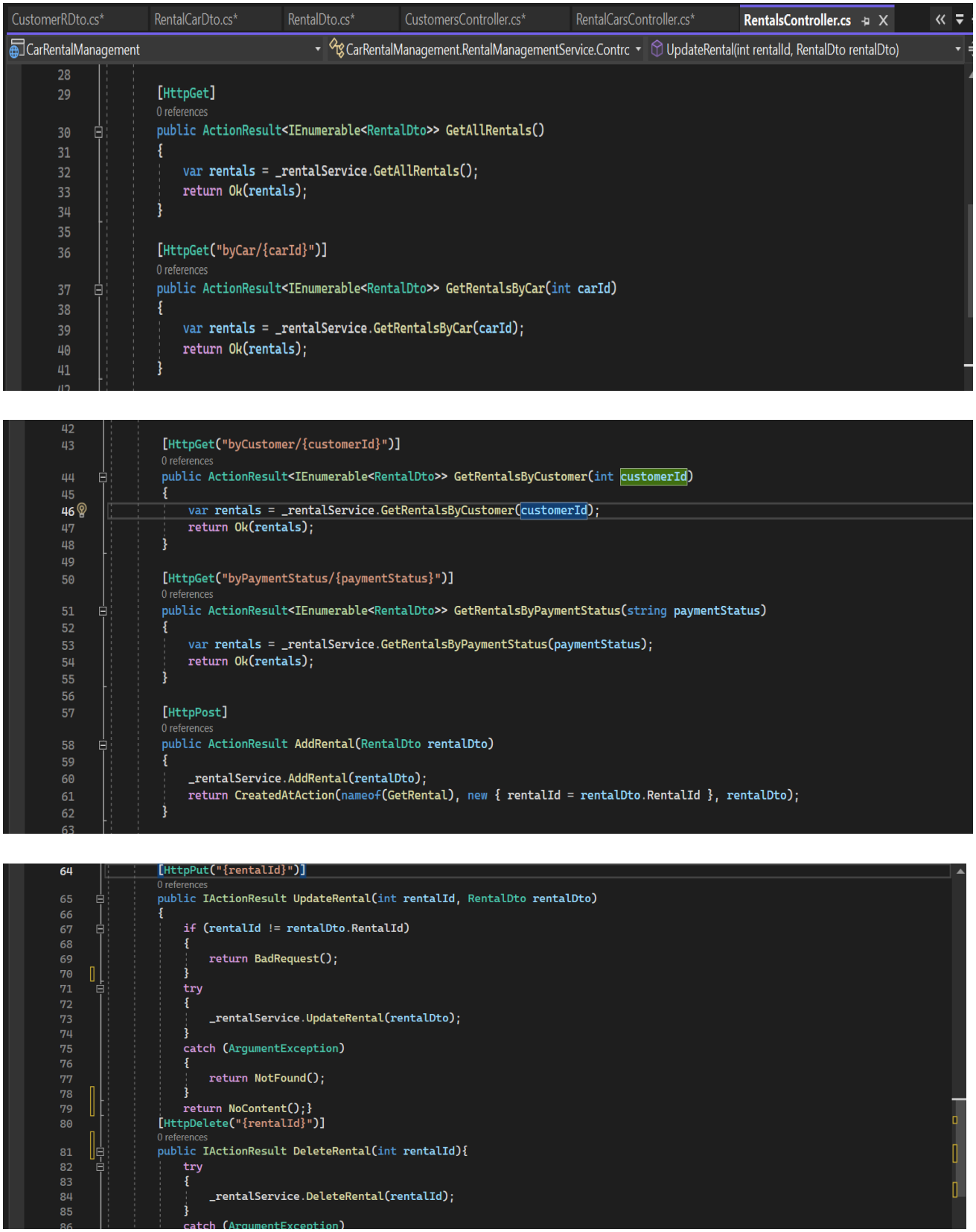
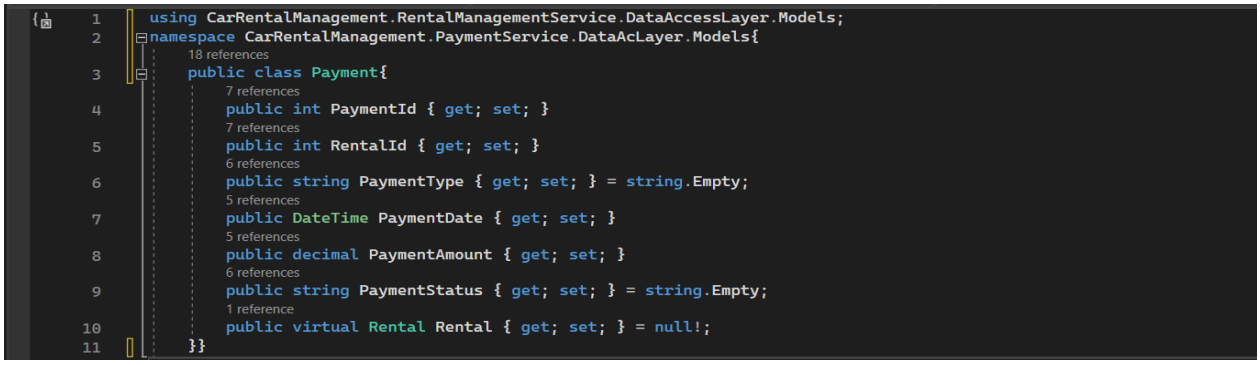


Figure 4.2.60 – 4.2.63: This figure shows the *RentalsController* is essential for handling requests that come in, sending them through the right action method, and producing a response to send back to the user.

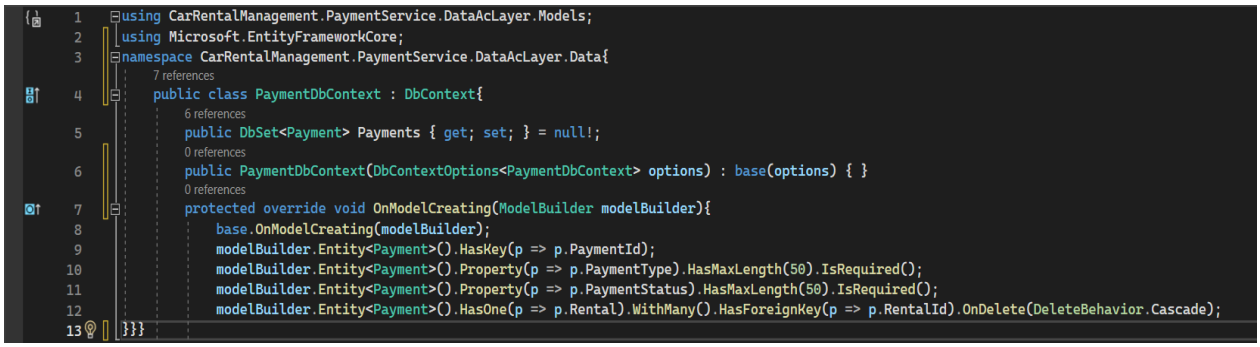
PaymentService -> DataAccessLayer ->Models-> Payment.cs



```
using CarRentalManagement.RentalManagementService.DataAccessLayer.Models;
namespace CarRentalManagement.PaymentService.DataAcLayer.Models{
    18 references
    public class Payment{
        7 references
        public int PaymentId { get; set; }
        7 references
        public int RentalId { get; set; }
        6 references
        public string PaymentType { get; set; } = string.Empty;
        5 references
        public DateTime PaymentDate { get; set; }
        5 references
        public decimal PaymentAmount { get; set; }
        6 references
        public string PaymentStatus { get; set; } = string.Empty;
        1 reference
        public virtual Rental Rental { get; set; } = null!;
    }}
```

Figure 4.2.64: This figure shows the file *Payment.cs* is essential for handling attributes and the columns in the rented cars table regarding the rentals in the microservices.

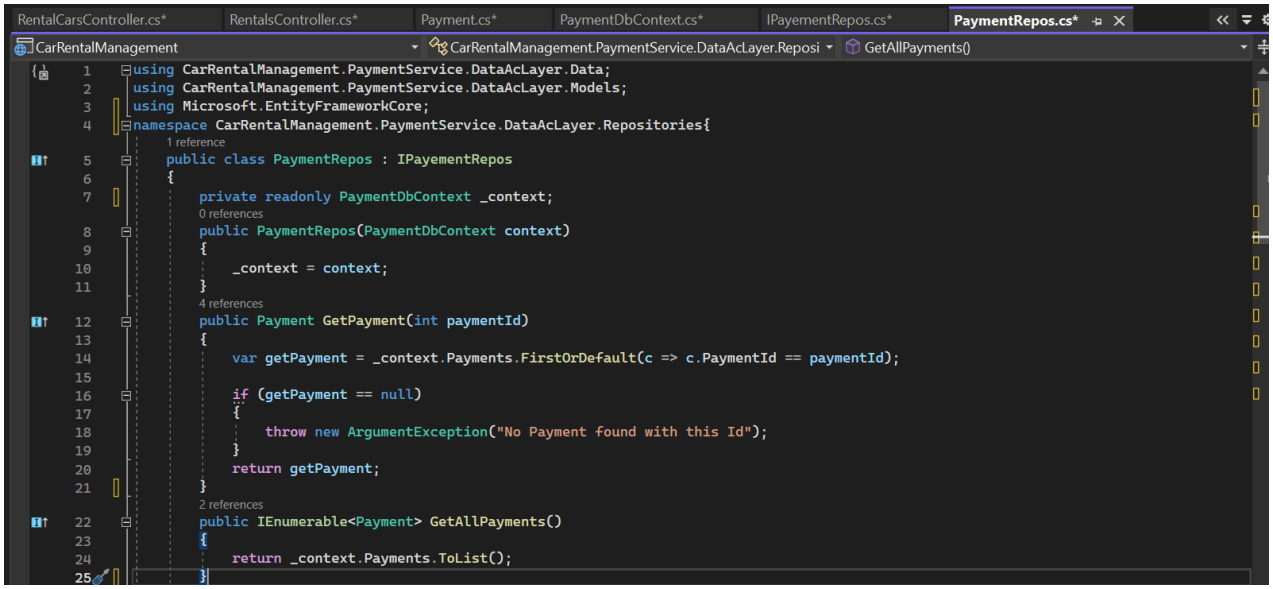
PaymentService -> DataAccessLayer ->Data-> PaymentDbContext.cs



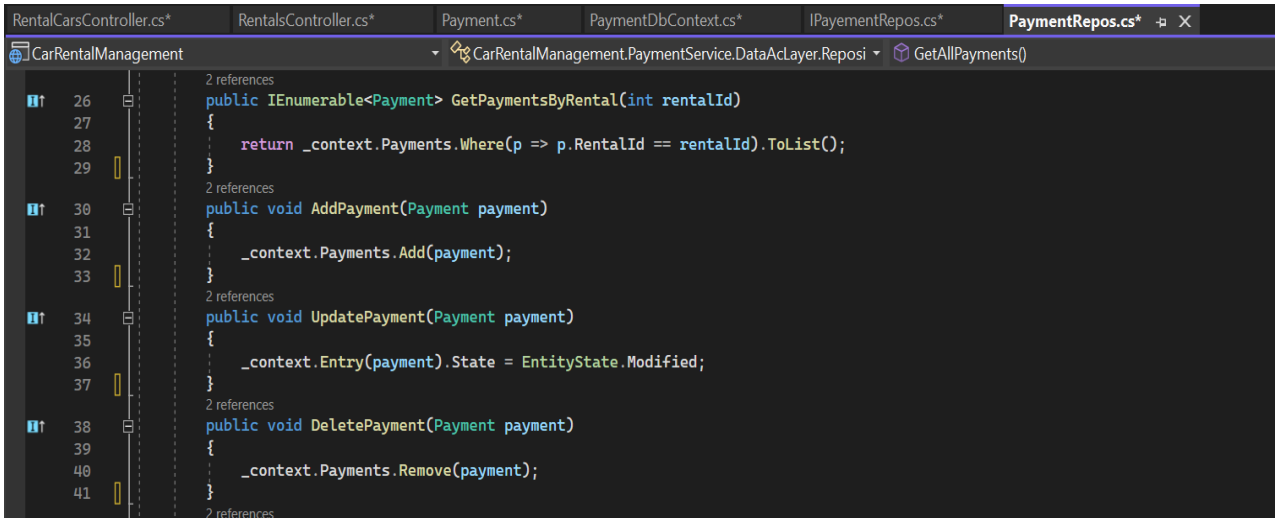
```
using CarRentalManagement.PaymentService.DataAcLayer.Models;
using Microsoft.EntityFrameworkCore;
namespace CarRentalManagement.PaymentService.DataAcLayer.Data{
    7 references
    public class PaymentDbContext : DbContext{
        6 references
        public DbSet<Payment> Payments { get; set; } = null!;
        0 references
        public PaymentDbContext(DbContextOptions<PaymentDbContext> options) : base(options) { }
        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder){
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<Payment>().HasKey(p => p.PaymentId);
            modelBuilder.Entity<Payment>().Property(p => p.PaymentType).HasMaxLength(50).IsRequired();
            modelBuilder.Entity<Payment>().Property(p => p.PaymentStatus).HasMaxLength(50).IsRequired();
            modelBuilder.Entity<Payment>().HasOne(p => p.Rental).WithMany().HasForeignKey(p => p.RentalId).OnDelete(DeleteBehavior.Cascade);
    }}}
```

Figure 4.2.65 This figure depicts the Database of *PaymentDbContext* which also makes a table *Payments* and then show the results that each of the attributes has.

PaymentService -> DataAccessLayer ->Repositories-> PaymentRepos.cs



```
RentalCarsController.cs*  RentalsController.cs*  Payment.cs*  PaymentDbContext.cs*  IPayementRepos.cs*  PaymentRepos.cs*
CarRentalManagement  CarRentalManagement.PaymentService.DataAcLayer.Reposi  GetAllPayments()
using CarRentalManagement.PaymentService.DataAcLayer.Data;
using CarRentalManagement.PaymentService.DataAcLayer.Models;
using Microsoft.EntityFrameworkCore;
namespace CarRentalManagement.PaymentService.DataAcLayer.Repositories{
    1 reference
    public class PaymentRepos : IPayementRepos
    {
        private readonly PaymentDbContext _context;
        0 references
        public PaymentRepos(PaymentDbContext context)
        {
            _context = context;
        }
        4 references
        public Payment GetPayment(int paymentId)
        {
            var getPayment = _context.Payments.FirstOrDefault(c => c.PaymentId == paymentId);

            if (getPayment == null)
            {
                throw new ArgumentException("No Payment found with this Id");
            }
            return getPayment;
        }
        2 references
        public IEnumerable<Payment> GetAllPayments()
        {
            return _context.Payments.ToList();
        }
```



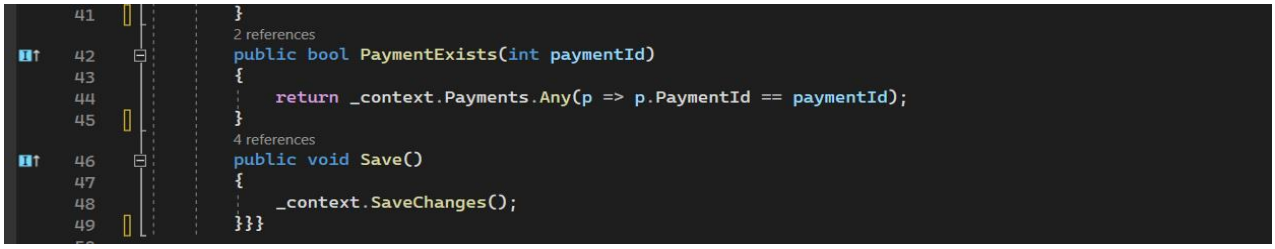


Figure 4.2.66-4.2.68: This figures shows the *PaymentRepos* file which comprises classes or methods that encapsulates all data logic.

PaymentService -> DataAccessLayer ->Repositories-> IPaymentRepos.cs

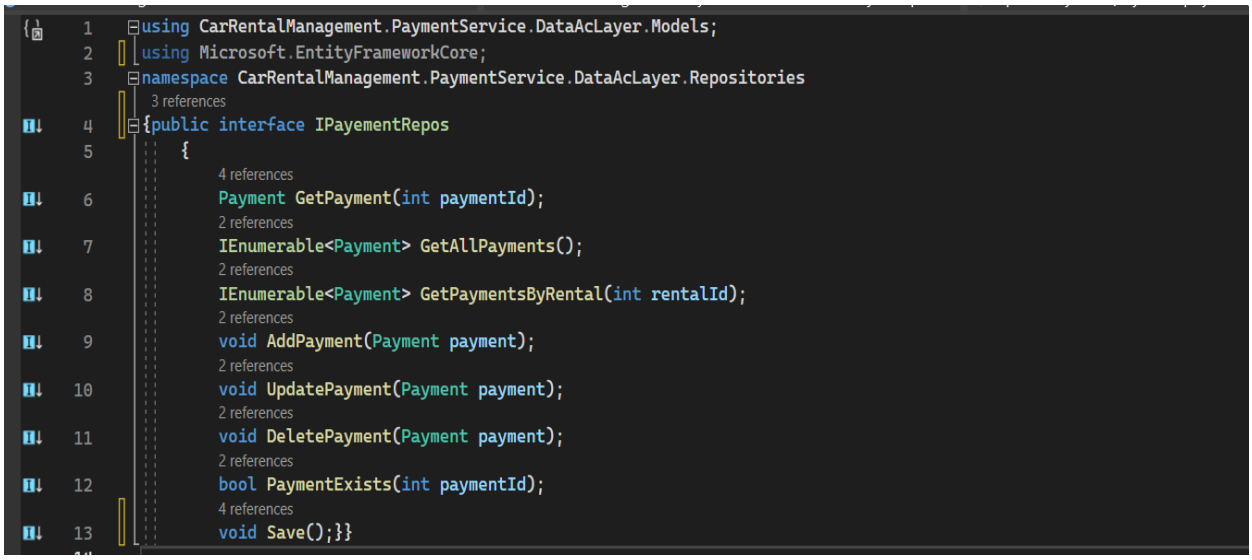


Figure 4.2.69: This figures shows the *IPaymentRepos* file which comprises classes or methods that encapsulates all data logic.

PaymentService -> BusinessLogicLayer ->Models-> PaymentDto.cs

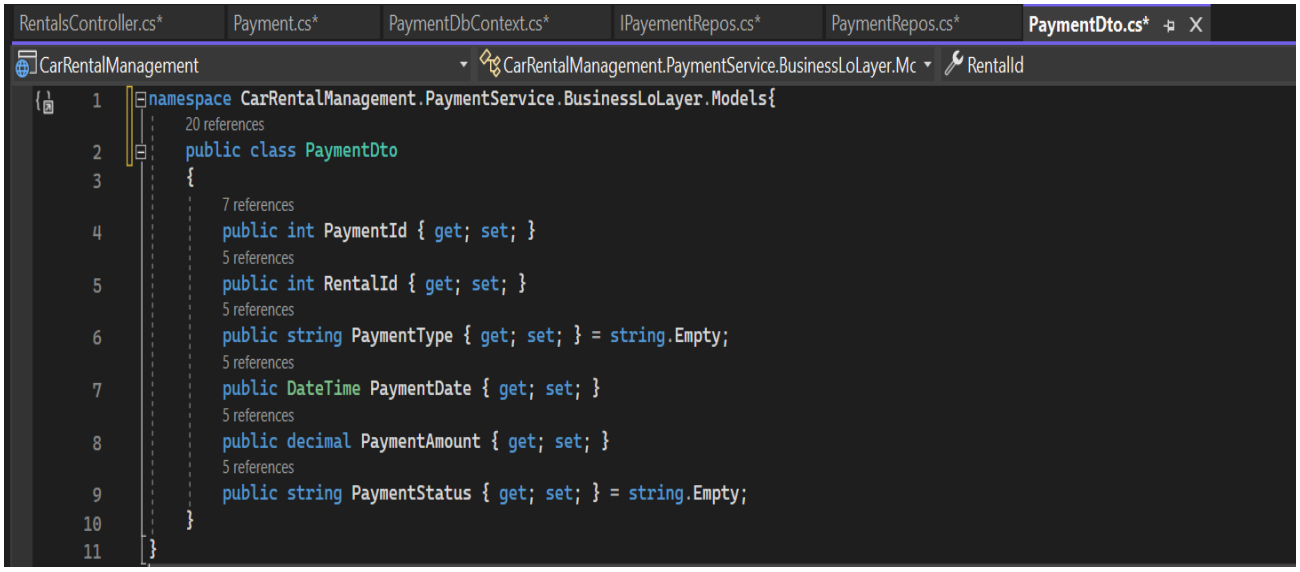


Figure 4.2.70: This figure shows the *PaymentDto* which is data transfer objects which is used to make application more modular and flexible.

PaymentService -> BusinessLogicLayer ->Services -> IPaymentService.cs

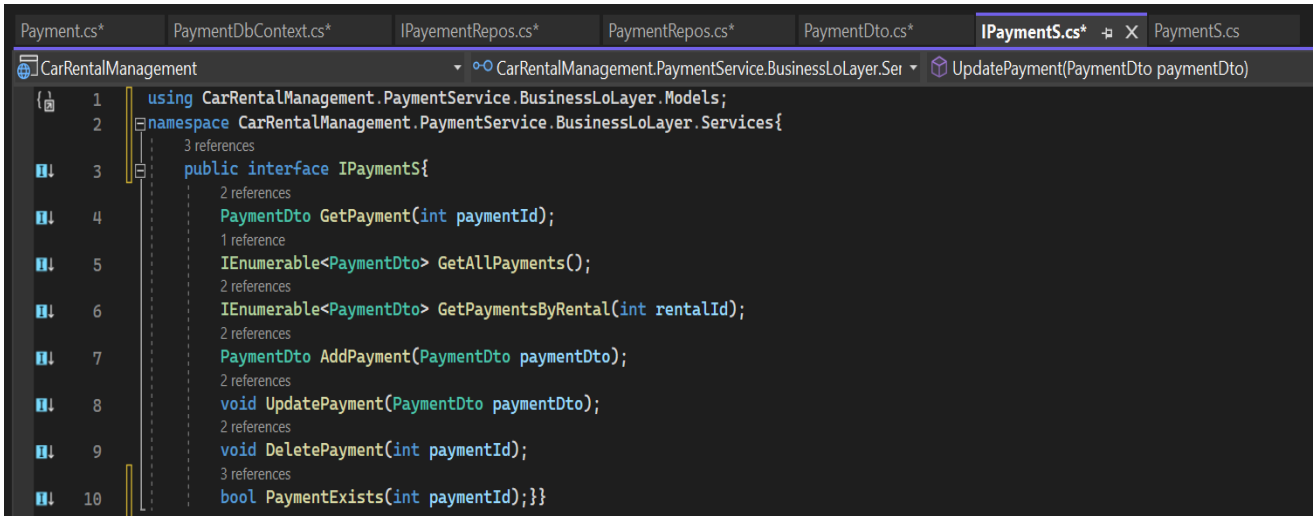
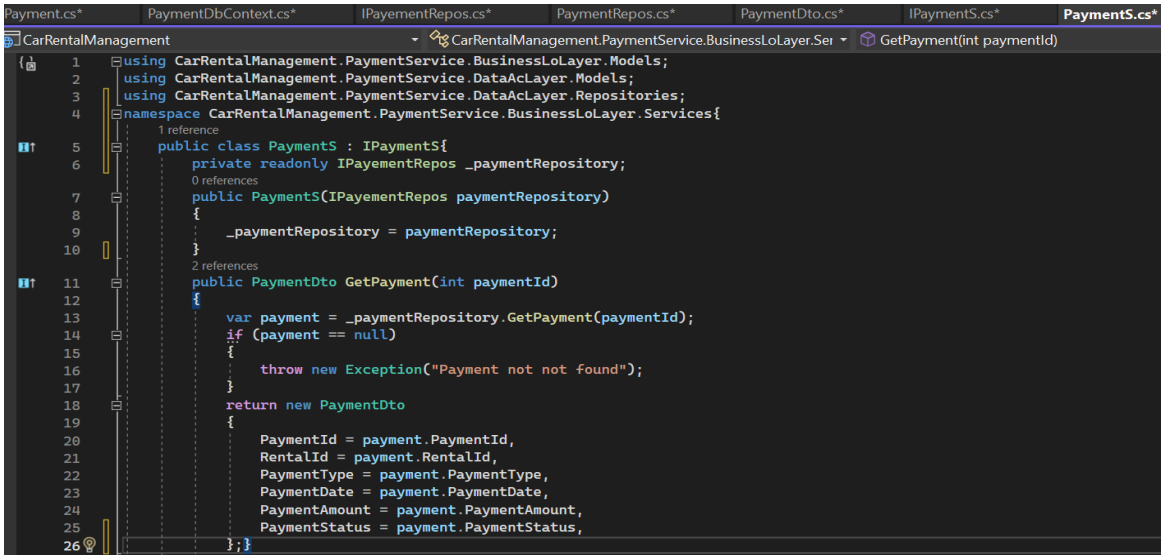


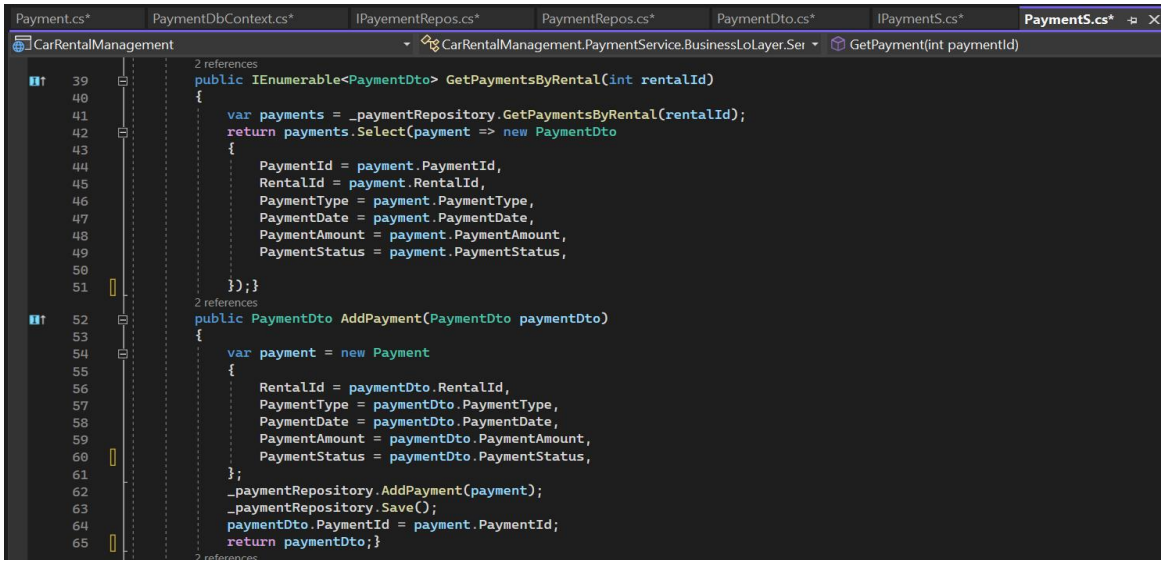
Figure 4.2.71: This figure shows the *IPaymentService* which is used in order to give the layer that presents data the necessary features, the business logic layer needs to implement a set of functions that are defined by the interface

PaymentService -> BusinessLogicLayer ->Services -> PaymentS.cs



```
using CarRentalManagement.PaymentService.BusinessLoLayer.Models;
using CarRentalManagement.PaymentService.DataAcLayer.Models;
using CarRentalManagement.PaymentService.DataAcLayer.Repositories;
namespace CarRentalManagement.PaymentService.BusinessLoLayer.Services{
    1 reference
    public class PaymentS : IPaymentS{
        private readonly IPayementRepos _paymentRepository;
        0 references
        public PaymentS(IPayementRepos paymentRepository)
        {
            _paymentRepository = paymentRepository;
        }
        2 references
        public PaymentDto GetPayment(int paymentId)
        {
            var payment = _paymentRepository.GetPayment(paymentId);
            if (payment == null)
            {
                throw new Exception("Payment not not found");
            }
            return new PaymentDto
            {
                PaymentId = payment.PaymentId,
                RentalId = payment.RentalId,
                PaymentType = payment.PaymentType,
                PaymentDate = payment.PaymentDate,
                PaymentAmount = payment.PaymentAmount,
                PaymentStatus = payment.PaymentStatus,
            };}
```

```
    };}
    1 reference
    public IEnumerable<PaymentDto> GetAllPayments()
    {
        var payments = _paymentRepository.GetAllPayments();
        return payments.Select(payment => new PaymentDto
        {
            PaymentId = payment.PaymentId,
            RentalId = payment.RentalId,
            PaymentType = payment.PaymentType,
            PaymentDate = payment.PaymentDate,
            PaymentAmount = payment.PaymentAmount,
            PaymentStatus = payment.PaymentStatus,
        });}
    2 references
```



```
    2 references
    public IEnumerable<PaymentDto> GetPaymentsByRental(int rentalId)
    {
        var payments = _paymentRepository.GetPaymentsByRental(rentalId);
        return payments.Select(payment => new PaymentDto
        {
            PaymentId = payment.PaymentId,
            RentalId = payment.RentalId,
            PaymentType = payment.PaymentType,
            PaymentDate = payment.PaymentDate,
            PaymentAmount = payment.PaymentAmount,
            PaymentStatus = payment.PaymentStatus,

        });}
    2 references
    public PaymentDto AddPayment(PaymentDto paymentDto)
    {
        var payment = new Payment
        {
            RentalId = paymentDto.RentalId,
            PaymentType = paymentDto.PaymentType,
            PaymentDate = paymentDto.PaymentDate,
            PaymentAmount = paymentDto.PaymentAmount,
            PaymentStatus = paymentDto.PaymentStatus,
        };
        _paymentRepository.AddPayment(payment);
        _paymentRepository.Save();
        paymentDto.PaymentId = payment.PaymentId;
        return paymentDto;}
    2 references
```

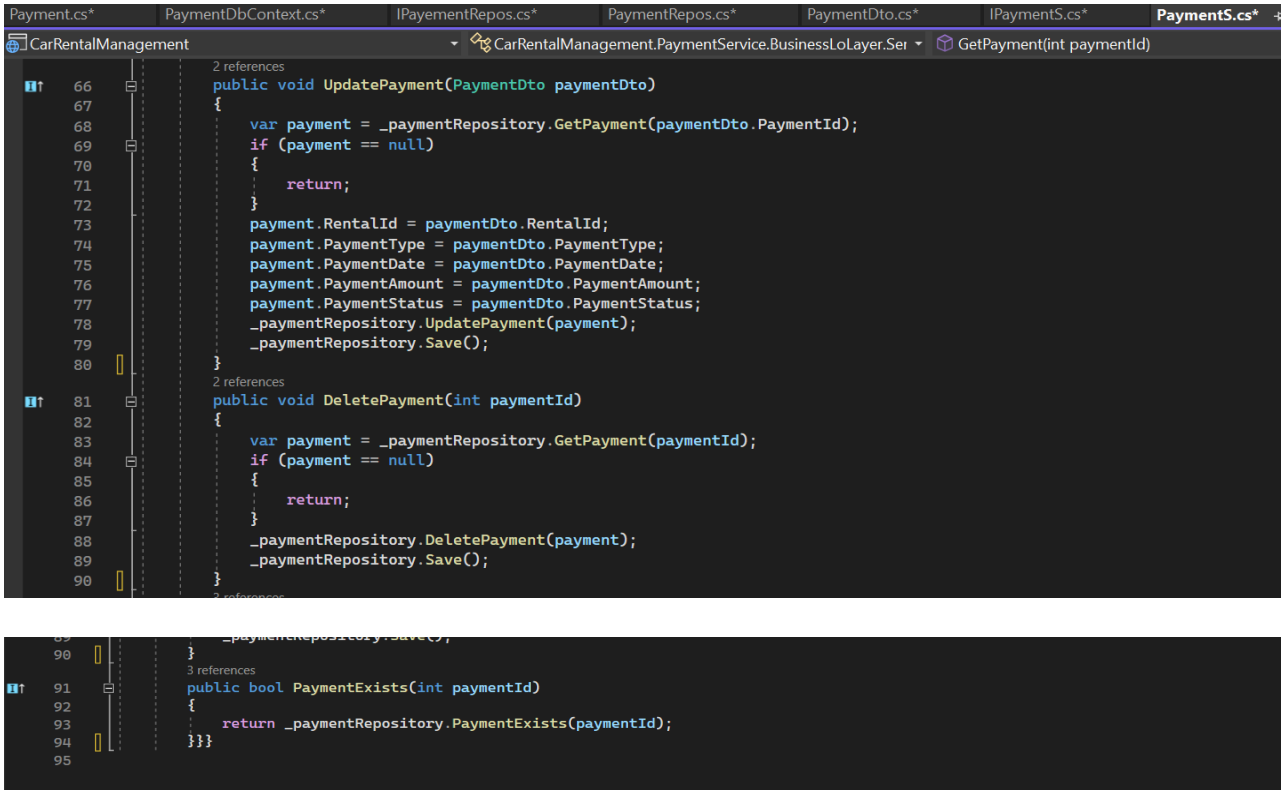
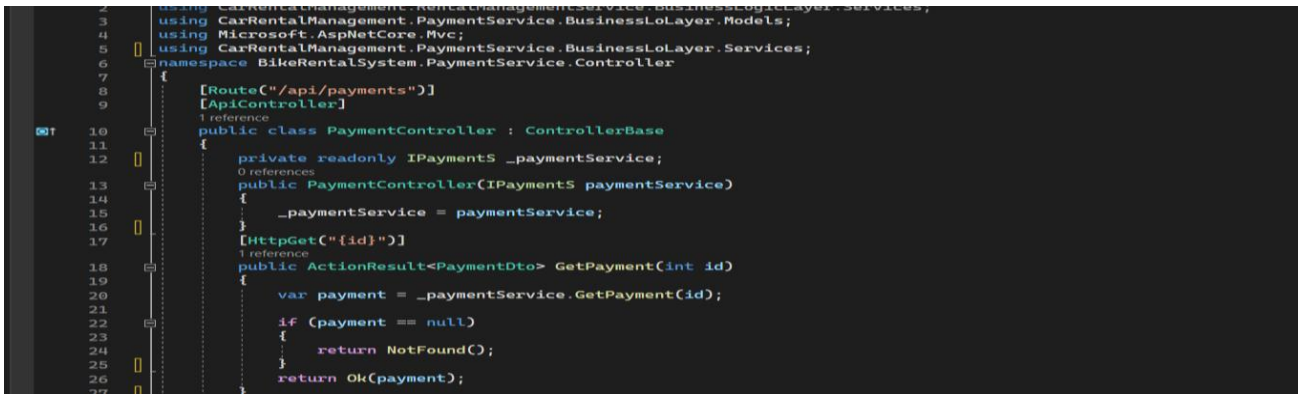


Figure 4.2.72 – 4.2.76: These figures depicts the *PaymentService* include the business services, the services that are in charge of carrying out the program's business logic. The services themselves can interface with the presentation layer and the data access layer in a standardised way by using Data Transfer Objects (DTOs) in the services.

PaymentService -> BusinessLogicLayer ->Controller -> PaymentController.cs



```
0 references
public ActionResult<IEnumerable<PaymentDto>> GetPaymentsByRental(int rentalId)
{
    var payments = _paymentService.GetPaymentsByRental(rentalId);
    if (payments == null || payments.Count() == 0)
    {
        return NotFound();
    }
    return Ok(payments);
}
[HttpPost]
0 references
public ActionResult<PaymentDto> AddPayment(PaymentDto paymentDto)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    var payment = _paymentService.AddPayment(paymentDto);
    return CreatedAtAction(nameof(GetPayment), new { id = payment.PaymentId }, payment);
}
[HttpPut("{id}")]
0 references
public IActionResult UpdatePayment(int id, PaymentDto paymentDto)
{
    if (id != paymentDto.PaymentId)
    {
        return BadRequest();
    }
    if (!_paymentService.PaymentExists(id))
```

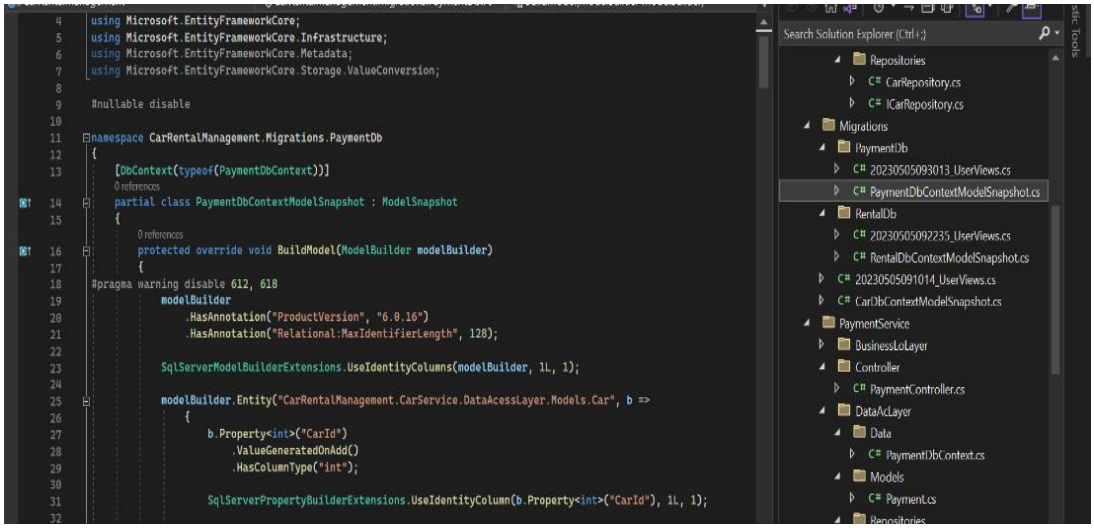
```
    }
    var payment = _paymentService.AddPayment(paymentDto);
    return CreatedAtAction(nameof(GetPayment), new { id = payment.PaymentId }, payment);
}
[HttpPut("{id}")]
0 references
public IActionResult UpdatePayment(int id, PaymentDto paymentDto)
{
    if (id != paymentDto.PaymentId)
    {
        return BadRequest();
    }
    if (!_paymentService.PaymentExists(id))
    {
        return NotFound();
    }
    _paymentService.UpdatePayment(paymentDto);
    return NoContent();
}
[HttpDelete("{id}")]
0 references
public IActionResult DeletePayment(int id)
{
    if (!_paymentService.PaymentExists(id))
    {
        return NotFound();
    }
    _paymentService.DeletePayment(id);
    return NoContent();
}}
```

Figure 4.2.77-4.2.79 This figure shows the *PaymentController* is essential for handling requests that come in, sending them through the right action method, and producing a response to send back to the user.

Migrations

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace CarRentalManagement.Migrations.PaymentDb
{
    1 reference
    public partial class UserViews : Migration
    {
        0 references
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Car",
                columns: table => new
                {
                    CarId = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    CarType = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    CarBrand = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    CarModel = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    BuyDate = table.Column<DateTime>(type: "datetime2", nullable: false),
                    BuyCost = table.Column<decimal>(type: "decimal(18,2)", nullable: false),
                    RentCostPerHour = table.Column<decimal>(type: "decimal(18,2)", nullable: false),
                    RentCostPerDay = table.Column<decimal>(type: "decimal(18,2)", nullable: false),
                    RentCostPerWeek = table.Column<decimal>(type: "decimal(18,2)", nullable: false),
                    Inventory = table.Column<int>(type: "int", nullable: false)
                },
                constraints: table =>
                {
```

```
Search Solution Explorer (Ctrl+;)
    Repositories
        CarRepository.cs
        ICarRepository.cs
    Migrations
        PaymentDb
            20230505093013_UserViews.cs
            PaymentDbContextModelSnapshot.cs
        RentalDb
            20230505092235_UserViews.cs
            RentalDbContextModelSnapshot.cs
        20230505091014_UserViews.cs
        CarDbContextModelSnapshot.cs
    PaymentService
        BusinessLolayer
        Controller
            PaymentController.cs
        DataAcLayer
            Data
                PaymentDbContext.cs
            Models
                Payment.cs
            Repositories
```

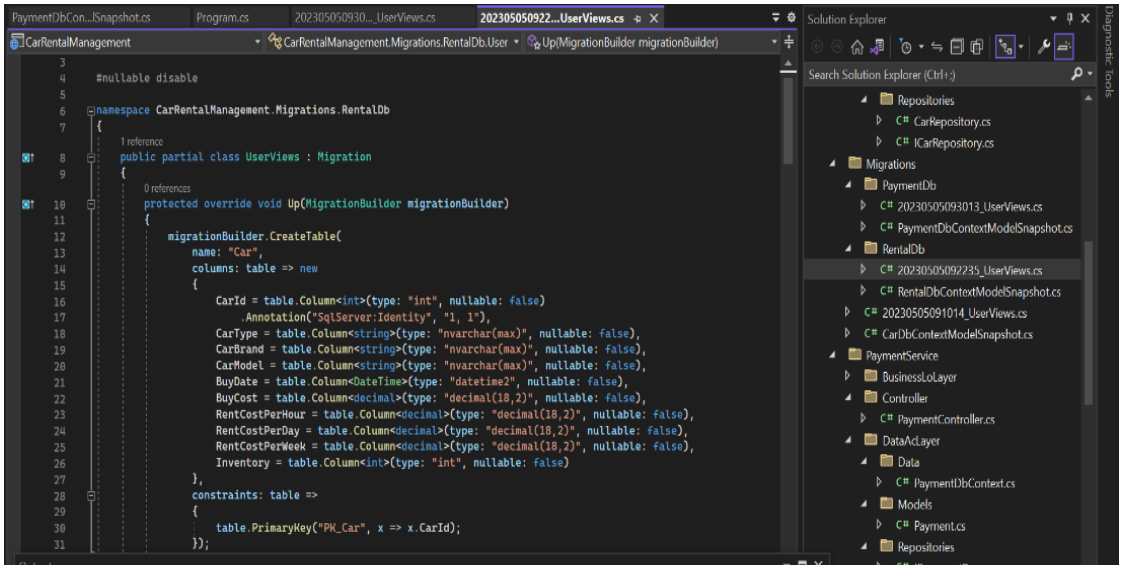


```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

#nullable disable

namespace CarRentalManagement.Migrations.PaymentDb
{
    [DbContext(typeof(PaymentDbContext))]
    0 references
    partial class PaymentDbContextModelSnapshot : ModelSnapshot
    {
        0 references
        protected override void BuildModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("ProductVersion", "6.0.16")
                .HasAnnotation("Relational:MaxIdentifierLength", 128);

            SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder, 1L, 1);

            modelBuilder.Entity("CarRentalManagement.CarService.DataAcessLayer.Models.Car", b =>
                {
                    b.Property<int>("CarId")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("int");

                    SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("CarId"), 1L, 1);
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Repositories
  - CarRepository.cs
  - ICarRepository.cs
- Migrations
  - PaymentDb
    - 20230505093013_UserViews.cs
    - PaymentDbContextModelSnapshot.cs
  - RentalDb
    - 20230505092235_UserViews.cs
    - RentalDbContextModelSnapshot.cs
  - 20230505091014_UserViews.cs
  - CarDbContextModelSnapshot.cs
- PaymentService
  - BusinessLoLayer
  - Controller
    - PaymentController.cs
  - DataAcLayer
    - Data
      - PaymentDbContext.cs
    - Models
      - Payment.cs
    - Repositories



PaymentDbCon...lSnapshot.cs | Program.cs | 20230505093O..._UserViews.cs | 20230505092...UserViews.cs

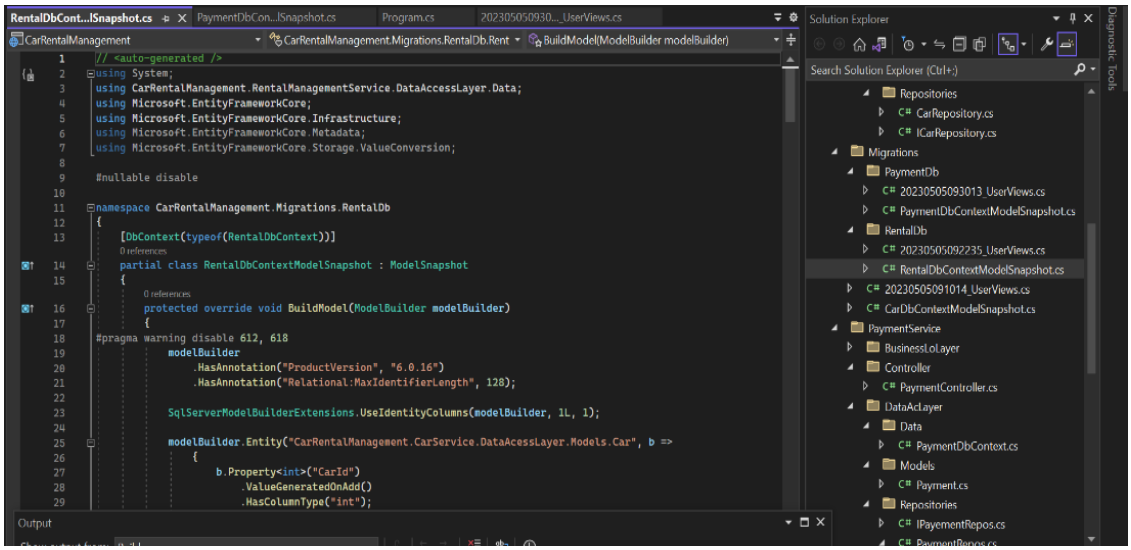
CarRentalManagement | CarRentalManagement.Migrations.RentalDb.User | Up(MigrationBuilder migrationBuilder)

```
#nullable disable

namespace CarRentalManagement.Migrations.RentalDb
{
    1 reference
    public partial class UserViews : Migration
    {
        0 references
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Car",
                columns: table => new
                {
                    CarId = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    CarType = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    CarBrand = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    CarModel = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    BuyDate = table.Column<DateTime>(type: "datetime2", nullable: false),
                    BuyCost = table.Column<decimal>(type: "decimal(18,2)", nullable: false),
                    RentCostPerHour = table.Column<decimal>(type: "decimal(18,2)", nullable: false),
                    RentCostPerDay = table.Column<decimal>(type: "decimal(18,2)", nullable: false),
                    RentCostPerWeek = table.Column<decimal>(type: "decimal(18,2)", nullable: false),
                    Inventory = table.Column<int>(type: "int", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Car", x => x.CarId);
                });
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Repositories
  - CarRepository.cs
  - ICarRepository.cs
- Migrations
  - PaymentDb
    - 20230505093013_UserViews.cs
    - PaymentDbContextModelSnapshot.cs
  - RentalDb
    - 20230505092235_UserViews.cs
    - RentalDbContextModelSnapshot.cs
  - 20230505091014_UserViews.cs
  - CarDbContextModelSnapshot.cs
- PaymentService
  - BusinessLoLayer
  - Controller
    - PaymentController.cs
  - DataAcLayer
    - Data
      - PaymentDbContext.cs
    - Models
      - Payment.cs
    - Repositories



RentalDbCont...lSnapshot.cs | PaymentDbCon...lSnapshot.cs | Program.cs | 20230505093O..._UserViews.cs

CarRentalManagement | CarRentalManagement.Migrations.RentalDb.Rent | BuildModel(ModelBuilder modelBuilder)

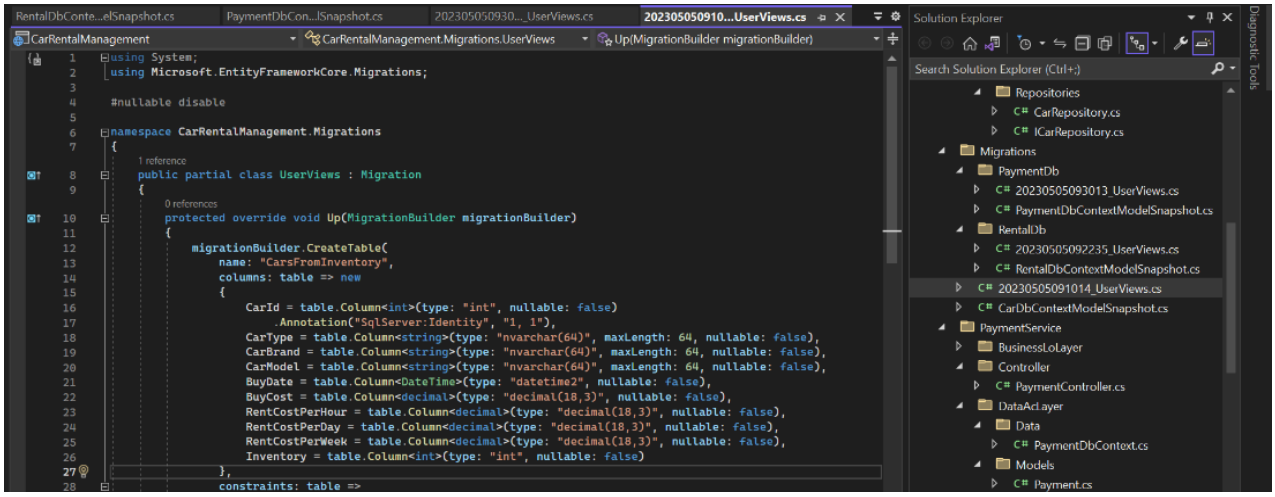
```
// <auto-generated />
using System;
using CarRentalManagement.RentalManagementService.DataAccessLayer.Data;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

#nullable disable

namespace CarRentalManagement.Migrations.RentalDb
{
    [DbContext(typeof(RentalDbContext))]
    0 references
    partial class RentalDbContextModelSnapshot : ModelSnapshot
    {
        0 references
        protected override void BuildModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("ProductVersion", "6.0.16")
                .HasAnnotation("Relational:MaxIdentifierLength", 128);

            SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder, 1L, 1);

            modelBuilder.Entity("CarRentalManagement.CarService.DataAcessLayer.Models.Car", b =>
                {
                    b.Property<int>("CarId")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("int");
```

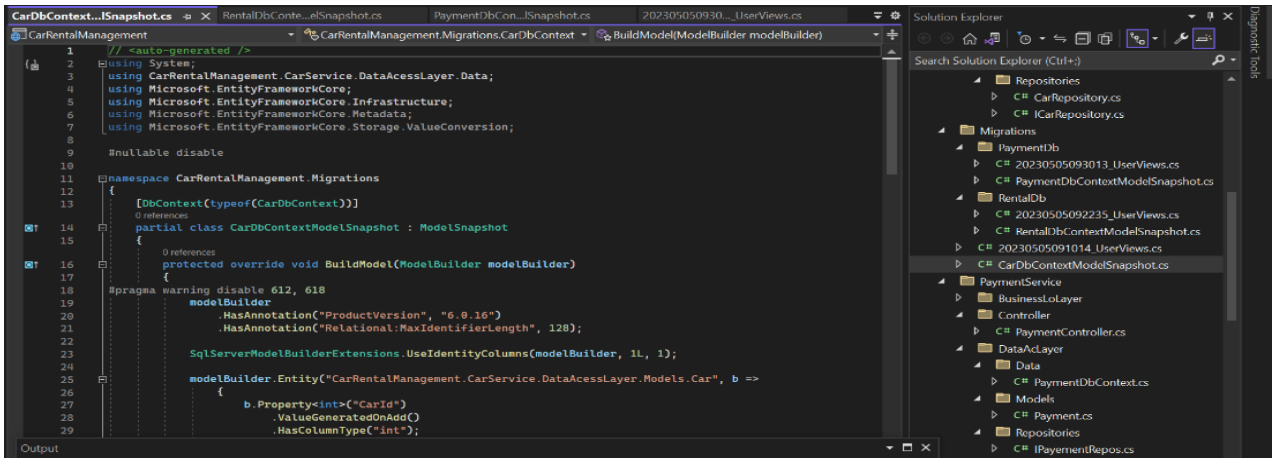
Output

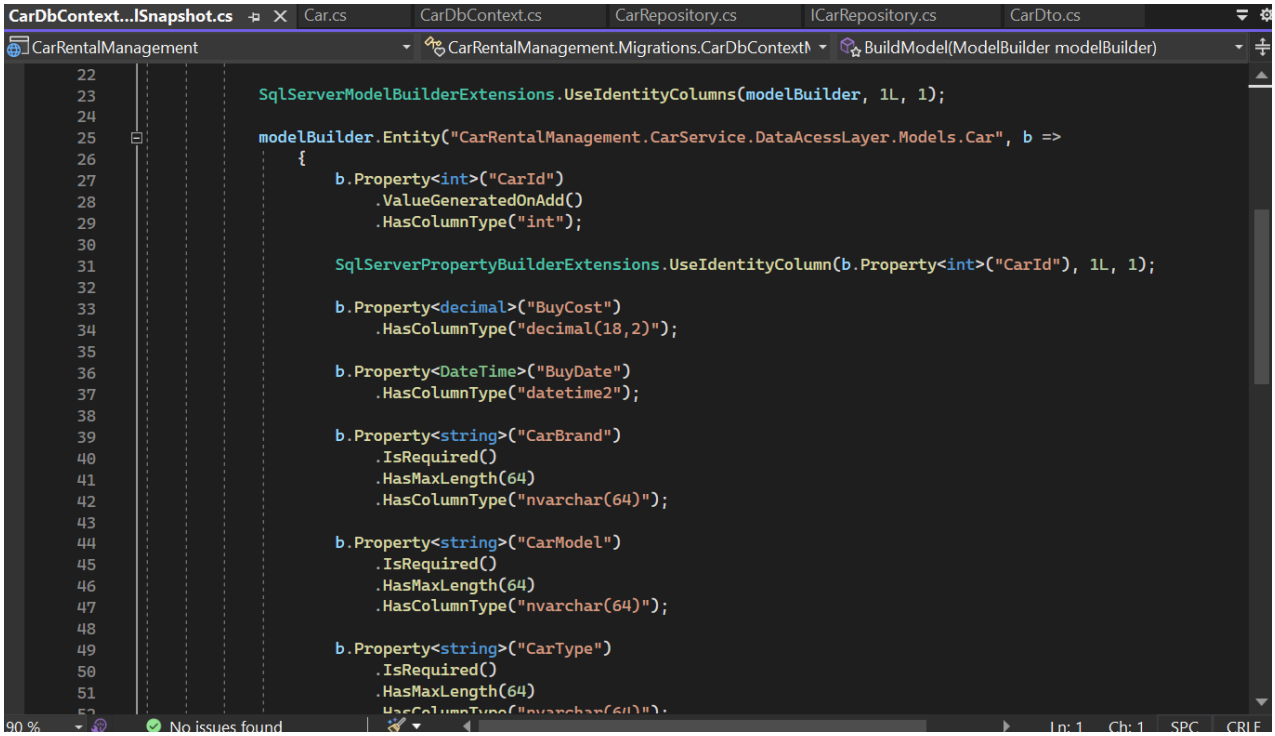
Solution Explorer

Search Solution Explorer (Ctrl+;)

- Repositories
  - CarRepository.cs
  - ICarRepository.cs
- Migrations
  - PaymentDb
    - 20230505093013_UserViews.cs
    - PaymentDbContextModelSnapshot.cs
  - RentalDb
    - 20230505092235_UserViews.cs
    - RentalDbContextModelSnapshot.cs
  - 20230505091014_UserViews.cs
  - CarDbContextModelSnapshot.cs
- PaymentService
  - BusinessLoLayer
  - Controller
    - PaymentController.cs
  - DataAcLayer
    - Data
      - PaymentDbContext.cs
    - Models
      - Payment.cs
    - Repositories
      - IPayementRepos.cs


RentalDbConte...elSnapshot.cs
PaymentDbCon...lSnapshot.cs
202305050930..._UserViews.cs
202305050910...UserViews.cs
CarRentalManagement
CarRentalManagement.Migrations.UserViews
Up(MigrationBuilder migrationBuilder)
using System;
using Microsoft.EntityFrameworkCore.Migrations;
#nullable disable
namespace CarRentalManagement.Migrations
{
1 reference
public partial class UserViews : Migration
{
0 references
protected override void Up(MigrationBuilder migrationBuilder)
{
migrationBuilder.CreateTable(
name: "CarsFromInventory",
columns: table => new
{
CarId = table.Column<int>(type: "int", nullable: false)
.Annotation("SqlServer:Identity", "1, 1"),
CarType = table.Column<string>(type: "nvarchar(64)", maxLength: 64, nullable: false),
CarBrand = table.Column<string>(type: "nvarchar(64)", maxLength: 64, nullable: false),
CarModel = table.Column<string>(type: "nvarchar(64)", maxLength: 64, nullable: false),
BuyDate = table.Column<DateTime>(type: "datetime2", nullable: false),
BuyCost = table.Column<decimal>(type: "decimal(18,3)", nullable: false),
RentCostPerHour = table.Column<decimal>(type: "decimal(18,3)", nullable: false),
RentCostPerDay = table.Column<decimal>(type: "decimal(18,3)", nullable: false),
RentCostPerWeek = table.Column<decimal>(type: "decimal(18,3)", nullable: false),
Inventory = table.Column<int>(type: "int", nullable: false)
},
constraints: table =>
Solution Explorer
Search Solution Explorer (Ctrl+;)
Repositories
CarRepository.cs
ICarRepository.cs
Migrations
PaymentDb
20230505093013_UserViews.cs
PaymentDbContextModelSnapshot.cs
RentalDb
20230505092235_UserViews.cs
RentalDbContextModelSnapshot.cs
20230505091014_UserViews.cs
CarDbContextModelSnapshot.cs
PaymentService
BusinessLoLayer
Controller
PaymentController.cs
DataAcLayer
Data
PaymentDbContext.cs
Models
Payment.cs
Diagnostic Tools


CarDbContext...lSnapshot.cs
RentalDbConte...elSnapshot.cs
PaymentDbCon...lSnapshot.cs
202305050930..._UserViews.cs
CarRentalManagement
CarRentalManagement.Migrations.CarDbContext
BuildModel(ModelBuilder modelBuilder)
// <auto-generated />
using System;
using CarRentalManagement.CarService.DataAcessLayer.Data;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
#nullable disable
namespace CarRentalManagement.Migrations
{
[DbContext(typeof(CarDbContext))]
0 references
partial class CarDbContextModelSnapshot : ModelSnapshot
{
0 references
protected override void BuildModel(ModelBuilder modelBuilder)
{
#pragma warning disable 612, 618
modelBuilder
.HasAnnotation("ProductVersion", "6.0.16")
.HasAnnotation("Relational:MaxIdentifierLength", 128);
SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder, 1L, 1);
modelBuilder.Entity("CarRentalManagement.CarService.DataAcessLayer.Models.Car", b =>
{
b.Property<int>("CarId")
.ValueGeneratedOnAdd()
.HasColumnType("int");
Output
Solution Explorer
Search Solution Explorer (Ctrl+;)
Repositories
CarRepository.cs
ICarRepository.cs
Migrations
PaymentDb
20230505093013_UserViews.cs
PaymentDbContextModelSnapshot.cs
RentalDb
20230505092235_UserViews.cs
RentalDbContextModelSnapshot.cs
20230505091014_UserViews.cs
CarDbContextModelSnapshot.cs
PaymentService
BusinessLoLayer
Controller
PaymentController.cs
DataAcLayer
Data
PaymentDbContext.cs
Models
Payment.cs
Repositories
IPayementRepos.cs
Diagnostic Tools


CarDbContext...lSnapshot.cs
Car.cs
CarDbContext.cs
CarRepository.cs
ICarRepository.cs
CarDto.cs
CarRentalManagement
CarRentalManagement.Migrations.CarDbContext
BuildModel(ModelBuilder modelBuilder)
SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder, 1L, 1);
modelBuilder.Entity("CarRentalManagement.CarService.DataAcessLayer.Models.Car", b =>
{
b.Property<int>("CarId")
.ValueGeneratedOnAdd()
.HasColumnType("int");
SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("CarId"), 1L, 1);
b.Property<decimal>("BuyCost")
.HasColumnType("decimal(18,2)");
b.Property<DateTime>("BuyDate")
.HasColumnType("datetime2");
b.Property<string>("CarBrand")
.IsRequired()
.HasMaxLength(64)
.HasColumnType("nvarchar(64)");
b.Property<string>("CarModel")
.IsRequired()
.HasMaxLength(64)
.HasColumnType("nvarchar(64)");
b.Property<string>("CarType")
.IsRequired()
.HasMaxLength(64)
90 %
No issues found
Ln: 1
Ch: 1
SPC
CRLF

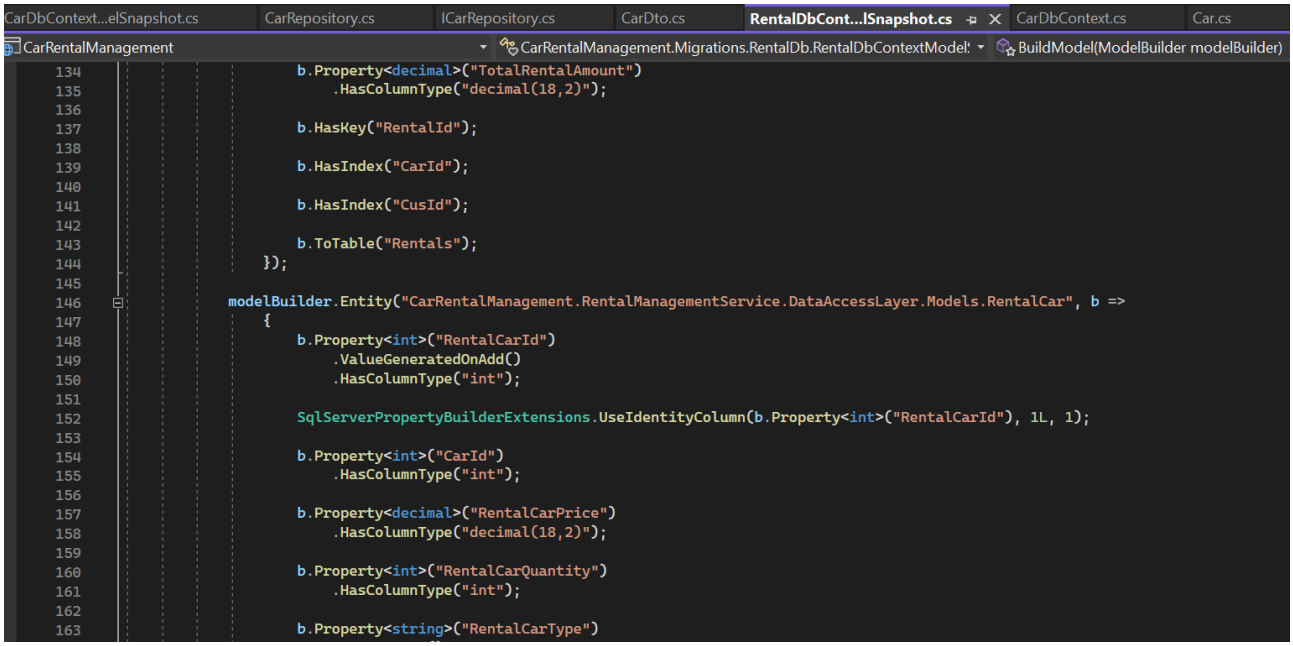


Figure 4.2.80 – 4.2.87 These figure shows the *Migrations* which are used in database management systems to help developers apply changes to the database schema.

AppSettings.json

Figure 4.2.88 This figure shows the *appSettings.json*, which has a connection string

Program.cs

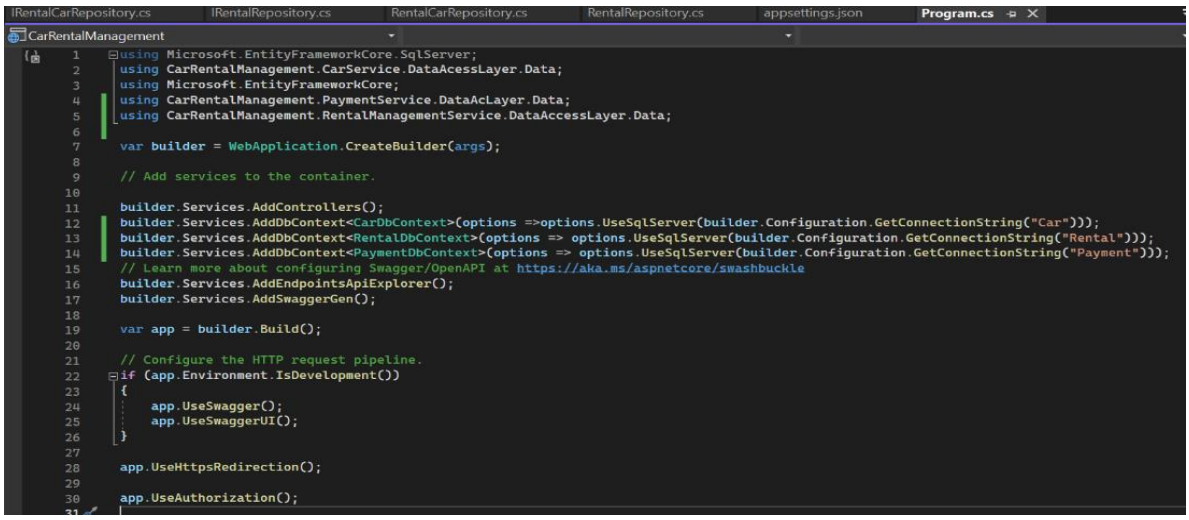
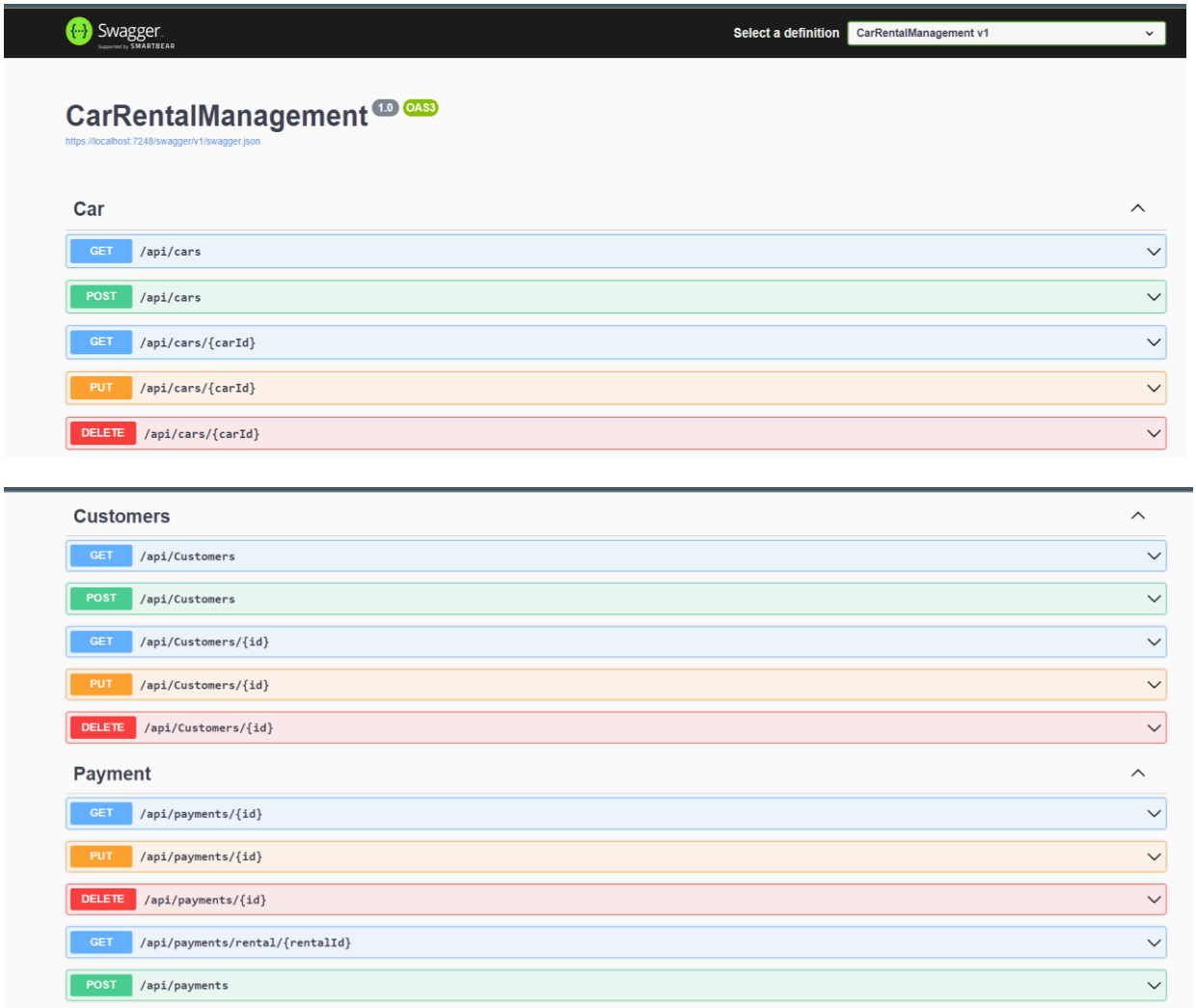


Figure 4.2.89 This figure shows the *Program.cs*

Swagger - OpenAPI



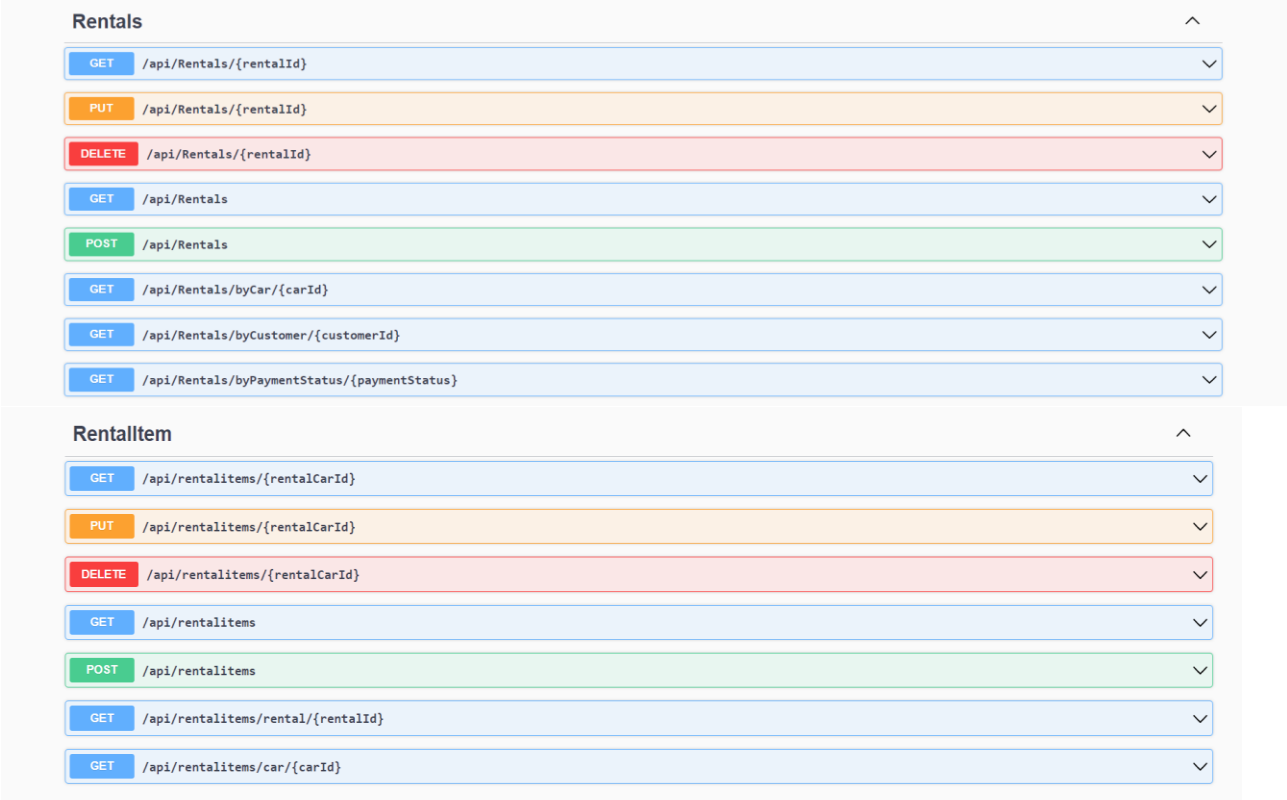


Figure 4.2.90 – 4.2.93 This figure shows the Swagger API is a set of open-source tools built to help programmers develop, design, document, and use REST APIs.

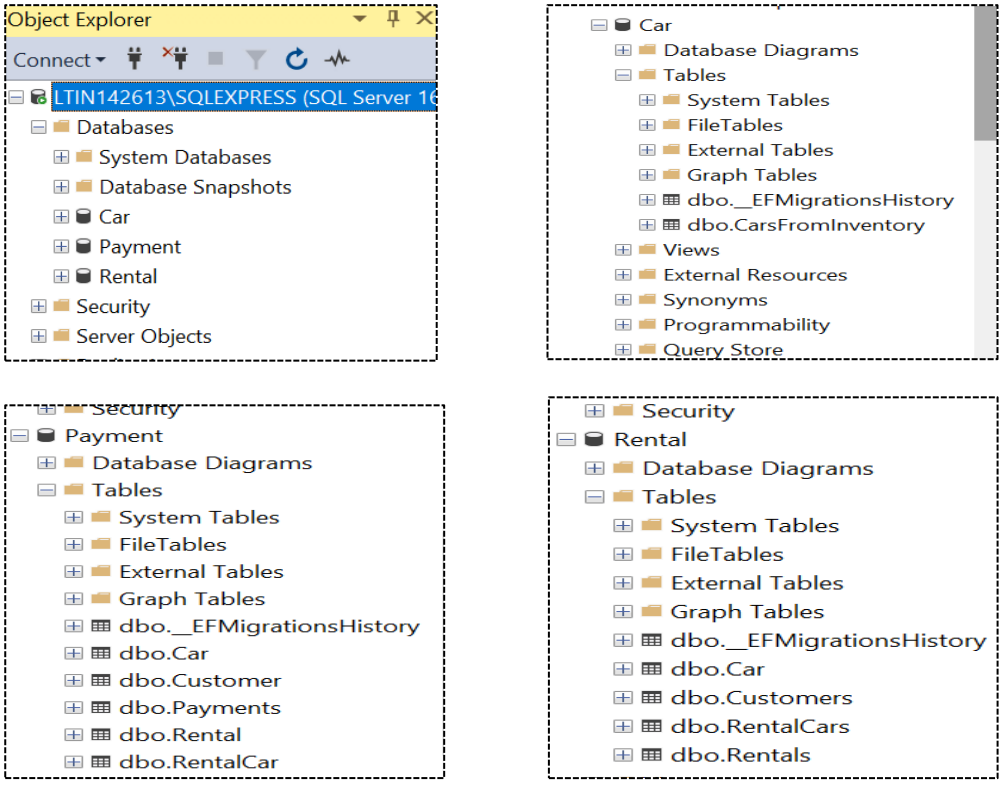


Figure 4.2.94 – 4.2.97 This figure shows the database was created on SSMS with the help of Code First Approach.

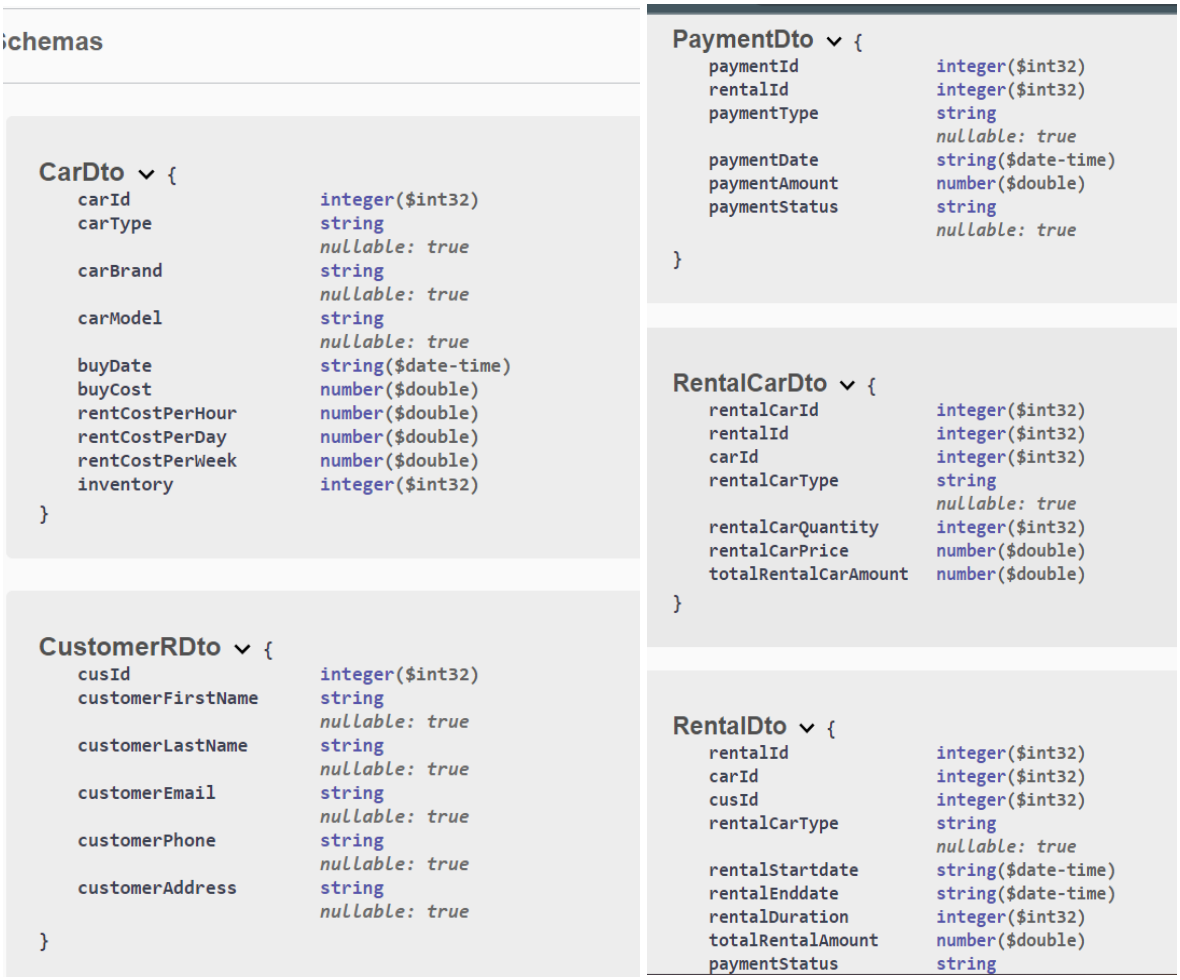


Figure 4.2.98 – 4.2.99 These figure shows the schema of 5 tables that are made with the columns.

4.3 Limitations and restrictions

1. To establish and carry out the capacity to rent a car:

- Clients must sign onto their individual profiles.
- Each visitor is required to set up an account.

2. This application will not enable you do scheduling, booking and renting functionalities until the billing step is finished.

3. This project prohibits renting the exact same vehicle on the exact same day. If such a result occurred, it would throw a fatal error.

CHAPTER -5 RESULT AND CONCLUSION

5.1 Result and Conclusion

The primary goal of the application aims to offer a straightforward, intuitive user interface for managing the rental sector. Customers may rapidly search for available vehicles, reserve them in their final days and review their rental histories. It additionally provides a trustworthy financial system that allows customers to pay for their services in a secure manner.

The project's architectural layout made use of a microservices framework, which allowed for flexible development and adaptability. A variety of services, including those for purchases, permission from users and authentication, rentals, and cars, are included in the project.

This project develops the backend of the project which includes the three microservices and each microservices have all the three-layer architecture is developed. Here, we have followed best practises were used in technologies like Entity Framework Core, SQL Server etc. The use of architectural frameworks, interdepenndence injection, and the separation of responsibilities are all adhered to in this project in accordance with standard behaviours for software development.

The Swagger port shows that WebAPI layer of all the microservices have been developed correctly as all the 5 controllers have been listed and show all the HTTP verbs used are being tested out. The Migrations were also developed so that all the tables that were made in entity models are made can be seen on SSMS which means that migrations have been successfully made and will have the data populated it

The PostMan was used to test and debug. The use of NUnit and Mocking was done to ensure efficient testing and to ensure that all of it is working fine and as expected. Through the course of this project, I understood the concepts of the CLR, CLS, FCL, code coverage, code analysis etc.

The car lease project based ASP..NET has, in general, demonstrated the practicalitty and advantages of employing the mix of technologies to create an effective tool. Comprehension of all facets of computer programme development, from gathering requirements to deployment, as well as the value of applying effective developing software adheres to throughout the whole procedure, have been provided.

5.2 Future Scope

The future scope of this project is to work up upon the limitations of the project that were discussed earlier and also identify and add more functionalities to this project such as:

1. **Language Support**: This functionality would allow the web application to be accessible to not only to the clients that understand English but also native languages of the country basically make it diverse.

2. **Integrating with social medias**: It would help clients to share their locations to their family members on different social media applications such as WhatsApp.

3. **Rating and Feedback Services** : It could develop a microservice where the client can give rating and feedback after every rental experience.

REFERENCES

[1] S. Dhiman and P. Sharma, "Performance testing: A comparative study and analysis of web service testing tools," International Journal of Computer Science and Mobile Computing, vol.5, no. 6, pp. 507-512, Jun. 2016.

[2] S. Sarasan, A. Ajith, and A. B. Archana, "Detection of Security Attacks and their Countermeasures in ASP //.NET Web Applications," in Proceedings of the IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, India, 2016, pp. 906-910. doi: 10.1109/ICACCI.2016.7732226.

[3] F. Reynders, "Introduction to ASP. NET Core," in Modern API Design with ASP. NET Core 2: Building Cross-Platform Back-End Systems, 1st ed., Birmingham, UK: Packt Publishing, 2018, pp. 9-22.

[4] H. Cao, J. R. Falleri, and X. Blanc, "Automated generation of REST API specification from plain HTML documentation," in Service-Oriented Computing: 15th International Conference, ICSOC 2017, Malaga, Spain, November 13–16, 2017, Proceedings, Cham, Switzerland: Springer International Publishing, 2017, pp. 453-461.

[5] A. Arcuri, "RESTful API automated test case generation," in 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2017, pp. 9-20.

[6] J. Lerman and R. Miller , Programming Entity Framework: Code First: Creating and Configuring Data Models from Your Classes, Sebastopol, CA, USA: O'Reilly Media, Inc., Nov. 2011.

[7] M. Prajapati, "ASP. NET MVC-generic repository pattern and unit of work," International Journal of All Research Writings, vol. 1, no. 1, pp. 23-30, Jul. 2019.

[8] C. Cassell and P. Johnson, "Action research: Explaining the diversity," Human Relations, vol. 59, no. 6, pp. 783-814, Jun. 2006.

[9] D. Spadini, M. Aniche, M. Bruntink, and A. Bacchelli, "To mock or not to mock? An empirical study on mocking practices," in 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), 2017, pp. 402-412.

[10] S. Mostafa and X. Wang, "An empirical study on the usage of mocking frameworks in software testing," in 2014 14th International Conference on Quality Software (QSIC), 2014, pp. 127-132.

[11] M. M. Hassan, W. Afzal, M. Blom, B. Lindström, S. F. Andler, and S. Eldh, "Testability and software robustness: A systematic literature review," in 2015 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2015, pp. 341-348.

[12] R. Sharma and A. Saha, "A systematic review of software testability measurement techniques," in 2018 International Conference on Computing, Power and Communication Technologies (GUCON), 2018, pp. 299-303.

[13] M. M. Tikir and J. K. Hollingsworth, "Efficient instrumentation for code coverage testing," in ACM SIGSOFT Software Engineering Notes, vol. 27, no. 4, pp. 86-96, July 2002.

Abhi

ORIGINALITY REPORT

8%	5%	3%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Jaypee University of Information Technology Student Paper	4%
2	repository.tudelft.nl Internet Source	1%
3	Pascal Zaragoza, Abdelhak-Djamel Seriai, Abderrahmane Seriai, Anas Shatnawi, Mustapha Derras. "Leveraging the Layered Architecture for Microservice Recovery", 2022 IEEE 19th International Conference on Software Architecture (ICSA), 2022 Publication	<1%
4	ftp.math.utah.edu Internet Source	<1%
5	A Samhitha, Sharadadevi Kaganurmath, Swetha S Rajulu. "Framework for Near Field Communication Controller Interface Protocol over Virtual IC", 2022 International Conference on Futuristic Technologies (INCOFT), 2022 Publication	<1%