MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS

Computer Engineering Program

CNG 300

Summer Practice Report

Name of Student :	Umut BAYBECE
ID Number :	2385201
Name of Company :	KOD YAZILIM ve PORJE HİZ. TUR. TİC. LTD. ŞTİ.
Project Title :	Car Tracking System
Date of Submission :	17.04.2022

ABSTRACT

This report contains details of the difficulties and solutions I came up with during my internship at KOD YAZILIM ve PORJE HİZ. My design and practices mainly were intended to improve my programming abilities and get experience in professional business life. I have acquired familiarity with utilizing libraries, frameworks, and applications. I also researched new technologies and methods and created software to assist my workplace. I have tested my designs by using additional applications. I also created two front-end designs with Angular and linked them with my web application as a secondary task. Therefore, I have become familiar with two important concepts: "Software Testing" and "Software Design." My first application was a server-client-based registration system for my car tracking application, and the second one was the car tracking application for my company. I have designed the back-end part of these systems using ASP.NET Core and Microsoft SQL Server Management Studio and tested them using an application named Postman and an open API support tool. While learning such concepts, I also practiced C# programming language concepts and SQL queries. Also, as a secondary task, I practiced building an Angular application to create a front-end by using several programming languages such as HTML, CSS, TypeScript, and Boostrap's front-end open-source toolkit. In conclusion, my internship benefited me in many aspects besides practicing software tests, developments, and design tools; I also understood the importance of the working environment and group work and the value of workload sharing between coworkers for the given tasks.

TABLE OF CONTENTS

LIST OF FIGURES

1. KOD YAZILIM ve PORJE HİZ. TUR. TİC. LTD. ŞTİ.

KOD YAZILIM ve PORJE HİZ. is a software producer company, and it is one of the companies with a high reputation in Antalya. KOD YAZILIM ve PORJE HİZ. was established in 2003, and it still develops new software projects and provides training and support services for the needs of accommodation facilities and tourism agencies.

Kod Yazılm's missions are improving customer satisfaction without sacrificing performance, maintaining R&D operations to adapt their software, and renewing and improving the company's products in line with industry requirements.

The company's building is located at 758 Dumlupınar Boulevard, Konyaaltı, Antalya. And it has several departments: Department of Finance, Department of Support and Education, Department of Programming, Department of Accounting, and Department of Management. I have accomplished my internship at the Department of Programming. The total number of employees of the company is more than fifty and fifteen of them working in the programming department.

The technologies developed by the company include building hotel management software, building customer relations management modules, building accounting management software, and designing mobile versions of their projects. Also, they test their software against potential errors.

2. INTRODUCTION

I have chosen KOD YAZILIM ve PORJE HİZ. to complete my internship since I closely followed their projects before starting my internship. Their projects are highly related to my interest areas, such as developing web applications based on database management systems. Also, their building location is in Antalya Teknokent, and they attract many customers and create competition with other companies. Also, because of this competition, they were trained on the most current methods in the industry.

In KOD YAZILIM ve PORJE HİZ., I have gained valuable experiences in many parts of working life, and it was a great starting point for my career. My company has offered a safe and disciplined working environment. And my supervisor Soykan Saydam was already experienced in many software designs and testing concepts and guided me in my projects.

When I started my internship, my team was working on several projects, and the most important one was Veboni. Veboni is an online payment application for hotels and tourism agencies that works on many devices. When I started my internship, they were recently released Veboni, and there was vast traffic of error handling. I observed their teamwork, and it allowed me to understand a professional working style.

After I have completed some tutorials, I have given my first task. It was creating a registration system. After that, I started to develop a car tracking system for my company. For these applications, I make use of ASP.NET, which is a web development framework for designing web apps on the .NET platform, and I also used Microsoft SQL Server Management Studio for database management. I created user interfaces with Angular, a development platform constructed by TypeScript.

In conclusion, during my internship, I have understood the basic concepts of developing web applications and developed my related coding skills. I have designed two software and solved the company's two problems.

3. PROBLEM STATEMENT

During my internship, I was asked to develop two software to solve the company's two different problems.

Firstly, my supervisor asked me to explore application development and database usage before working on the given problems. I was asked to do some research and get familiar with the concepts of web application development and database usage. Also, I have been given simple tutorials to get familiarized with Angular, ASP.NET Core, and Microsoft SQL Server Management Studio software.

Afterward, the company requested that I develop a simple registration page for a car tracking project. Therefore my first assignment was to create a registration application for a website to record users' information in a database. I have also been asked to do the front-end for the same web application.

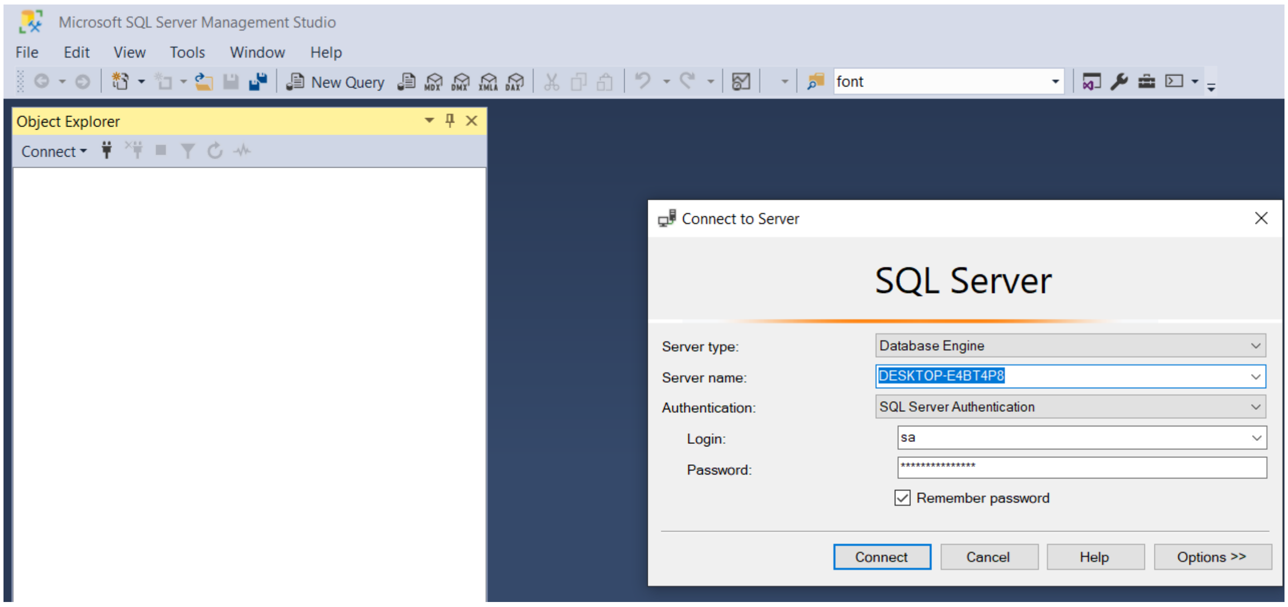
Finally, my company requested that I create a car tracking application to follow their car usage. The company needed this application to track its vehicle records, vehicle availability, and driver details. My supervisor also requested that I create a user interface and back-end to record driver and driving details into a database. I learned to apply basic CRUID operations (update, insert, delete, get) to handle entered data. During this period, my supervisor provided me with feedback and shared many sources to make my job easier.

4. DEVELOPMENT TOOLS AND THEIR ARCHITECTURE

In this part of the report, as a part of the solution, I will explain the usage of programming tools that I used to develop my project and their structural and functional behaviors.

4.1. MICROSOFT SQL SERVER MANAGEMENT STUDIO

I use Microsoft SQL Server Management Studio to store, query, and retrieve the data. It is widely used in industry. Since my company was using it in their professional business life, they helped me understand concepts and provided a product key of the application to reach its all services. As you can see in Figure 1, Microsoft SQL Server Management Studio uses server authentication to allow users to reach and edit stored data.

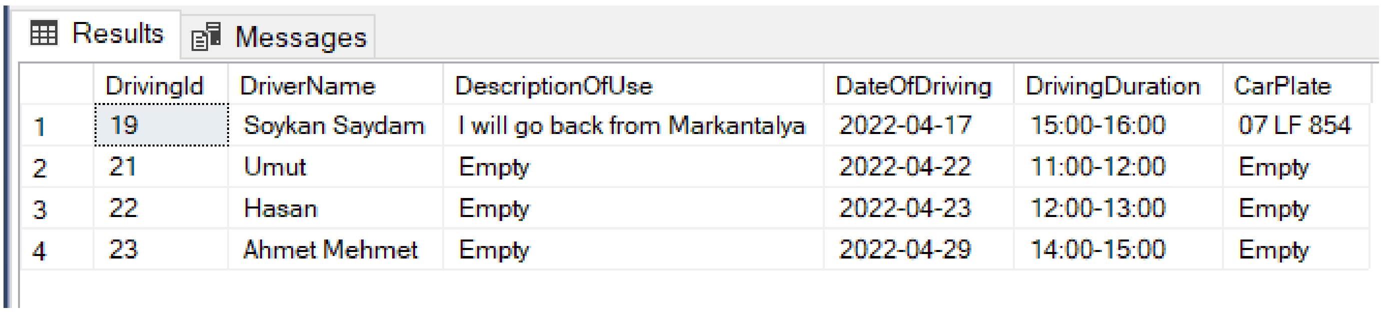


Server type: Database Engine
Server name: DESKTOP-E4BT4P8
Authentication: SQL Server Authentication
Login: sa
Password: ***************
☑ Remember password

Figure 1: Microsoft SQL Server Management Studio "Server Authentication" page

4.1.1. DATABASE AND ITS ARCHITECTURE

“A database is an organized collection of structured information, or data, typically stored electronically in a computer system” (Oracle, n.d.). There are many database management systems in the industry, and people are currently using them. Microsoft SQL Server Management Studio is one of the most widely used ones, and it provides a row-based table structure, as you can see in Figure 2.

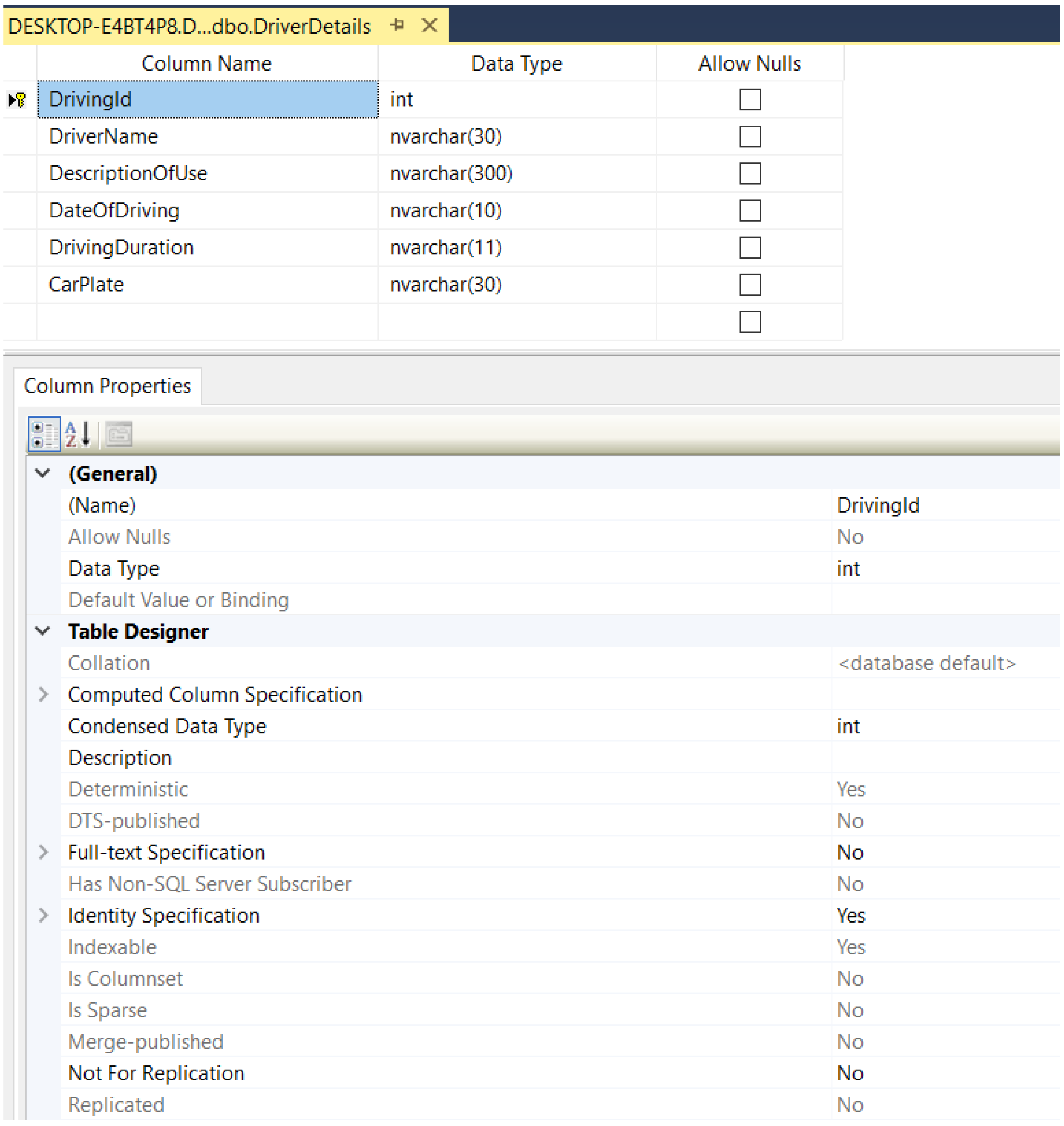


	DrivingId	DriverName	DescriptionOfUse	DateOfDriving	DrivingDuration	CarPlate
1	19	Soykan Saydam	I will go back from Markantalya	2022-04-17	15:00-16:00	07 LF 854
2	21	Umut	Empty	2022-04-22	11:00-12:00	Empty
3	22	Hasan	Empty	2022-04-23	12:00-13:00	Empty
4	23	Ahmet Mehmet	Empty	2022-04-29	14:00-15:00	Empty

Figure 2: Microsoft SQL Server Management Studio “raw-based structure” table

In a database, each table requires at least one primary key, and the primary keys are unique and not repeated for any other raws of the same table. Furthermore, the data is stored in tables in a specific order to speed up queries and reduce their complexity. There are different approaches for optimizing query speed and memory usage, e.g., using statistics, heuristics, etc.

Database tables can be connected to each other to prevent data redundancy. To achieve the connection between tables, we use foreign keys. Additionally, the data type may vary for each column in a table. For example, in my project, I store different data types, such as integers, strings, and so on, as you can see in Figure 3. The collected data, e.g., user entries, may be used for many operations via data queries and transactions. Also, we may decide to allow null values or specific lengths of variables for particular columns.



DESKTOP-E4BT4P8.D...dbo.DriverDetails

Column Name	Data Type	Allow Nulls
DrivingId	int	☐
DriverName	nvarchar(30)	☐
DescriptionOfUse	nvarchar(300)	☐
DateOfDriving	nvarchar(10)	☐
DrivingDuration	nvarchar(11)	☐
CarPlate	nvarchar(30)	☐
		☐

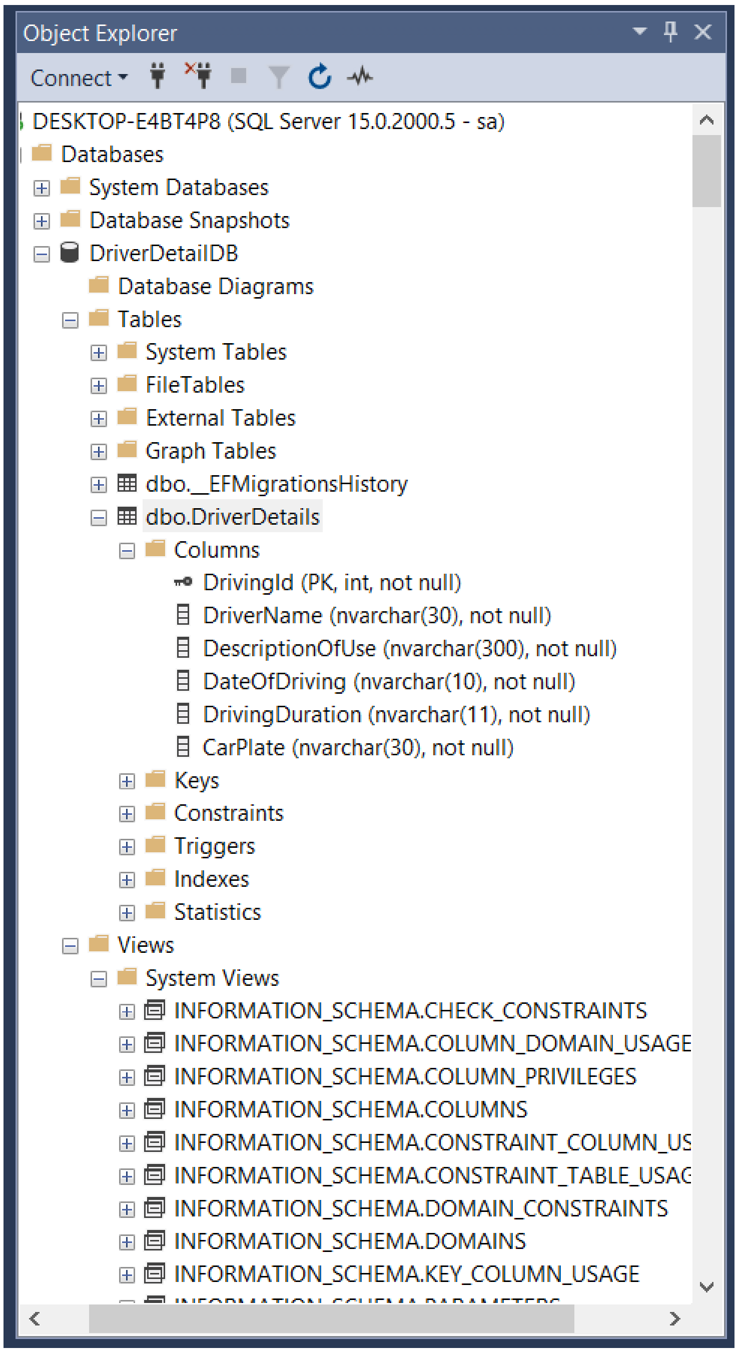
Column Properties

(General)	
(Name)	DrivingId
Allow Nulls	No
Data Type	int
Default Value or Binding	
Table Designer	
Collation	<database default>
Computed Column Specification	
Condensed Data Type	int
Description	
Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
Indexable	Yes
Is Columnset	No
Is Sparse	No
Merge-published	No
Not For Replication	No
Replicated	No

Figure 3: “data types” table in Microsoft SQL Server Management Studio

In a database, we can create security protocols and process the data in different ways. For example, we may use encryption or token creation to secure our data transactions.

Furthermore, the database's engine is responsible for generating tables, views, and all other structures in the database. Therefore, to complete the required actions, we can use the database engine to create additional tables and views for entered criteria, as shown in Figure 4.



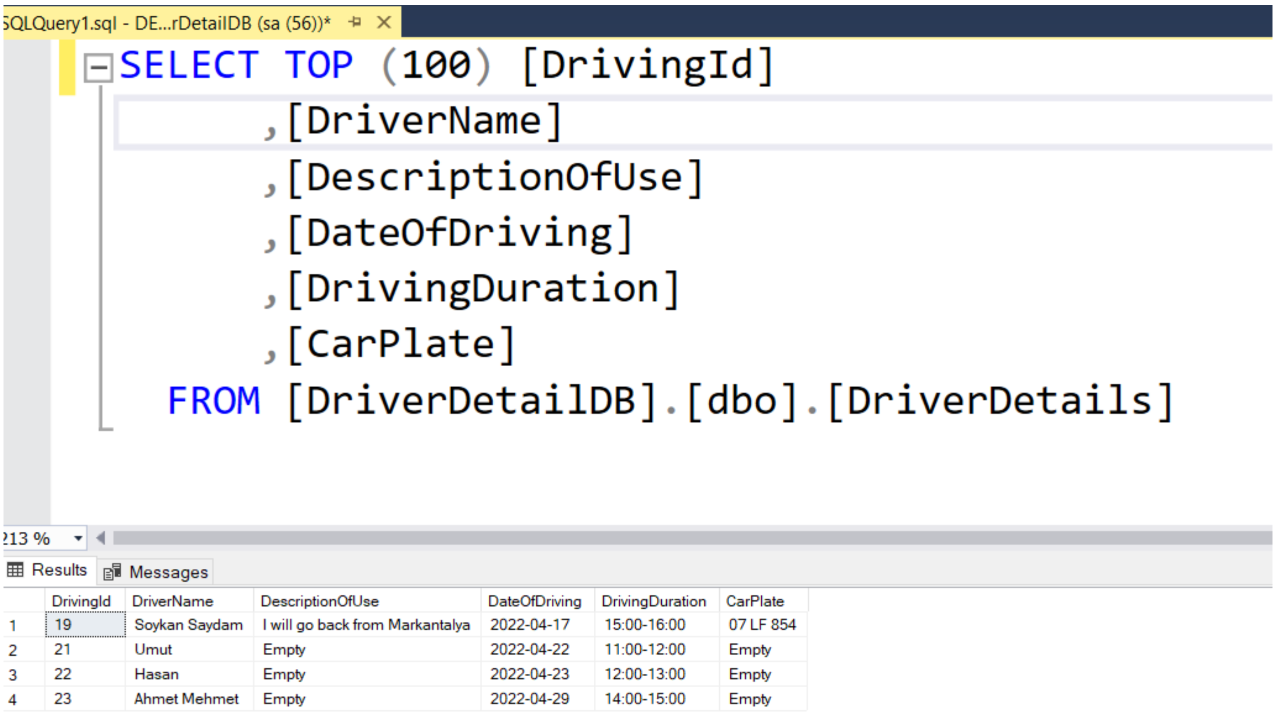
```
DESKTOP-E4BT4P8 (SQL Server 15.0.2000.5 - sa)
  Databases
    System Databases
    Database Snapshots
    DriverDetailDB
      Database Diagrams
      Tables
        System Tables
        FileTables
        External Tables
        Graph Tables
        dbo.__EFMigrationsHistory
        dbo.DriverDetails
          Columns
            DrivingId (PK, int, not null)
            DriverName (nvarchar(30), not null)
            DescriptionOfUse (nvarchar(300), not null)
            DateOfDriving (nvarchar(10), not null)
            DrivingDuration (nvarchar(11), not null)
            CarPlate (nvarchar(30), not null)
          Keys
          Constraints
          Triggers
          Indexes
          Statistics
      Views
        System Views
          INFORMATION_SCHEMA.CHECK_CONSTRAINTS
          INFORMATION_SCHEMA.COLUMN_DOMAIN_USAGE
          INFORMATION_SCHEMA.COLUMN_PRIVILEGES
          INFORMATION_SCHEMA.COLUMNS
          INFORMATION_SCHEMA.CONSTRAINT_COLUMN_US
          INFORMATION_SCHEMA.CONSTRAINT_TABLE_USAG
          INFORMATION_SCHEMA.DOMAIN_CONSTRAINTS
          INFORMATION_SCHEMA.DOMAINS
          INFORMATION_SCHEMA.KEY_COLUMN_USAGE
```

Figure 4: “tables and views” table in Microsoft SQL Server Management Studio

4.1.2. SQL

Structured Query Language(SQL) is a database-centered language for creating queries and achieving other operations to reach the desired data from a database. There are certain operations and commands in SQL such as "select," "insert," and "create." To see the usage of the "select" operation, you may check Figure 5.

In the Figure 5, I reach the first 100 tuples of the DriverDetails table from the DriverDetailDB database by using a select query.



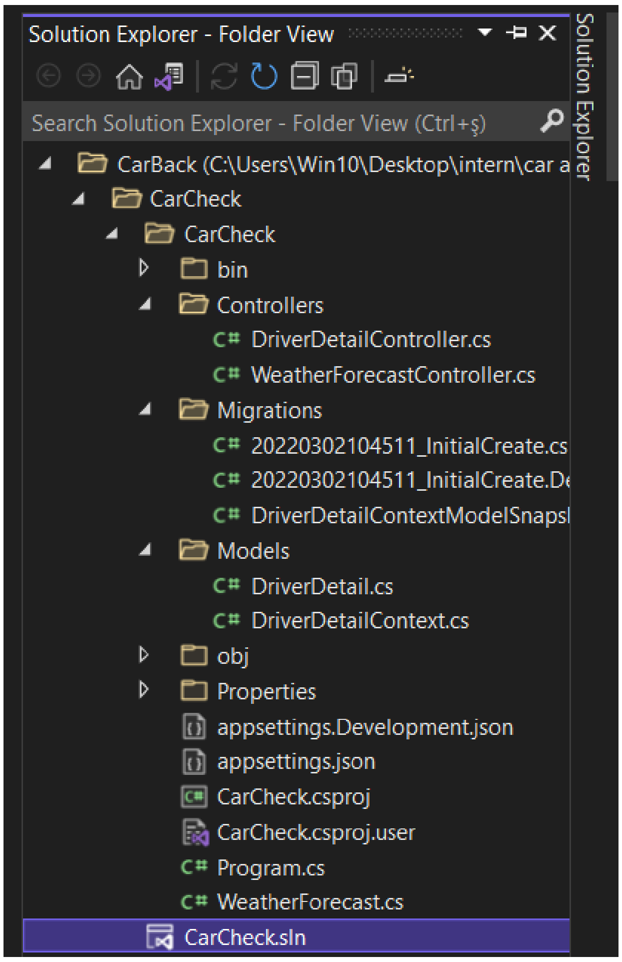
```sql
SELECT TOP (100) [DrivingId]
      ,[DriverName]
      ,[DescriptionOfUse]
      ,[DateOfDriving]
      ,[DrivingDuration]
      ,[CarPlate]
  FROM [DriverDetailDB].[dbo].[DriverDetails]
```

	DrivingId	DriverName	DescriptionOfUse	DateOfDriving	DrivingDuration	CarPlate
1	19	Soykan Saydam	I will go back from Markantalya	2022-04-17	15:00-16:00	07 LF 854
2	21	Umut	Empty	2022-04-22	11:00-12:00	Empty
3	22	Hasan	Empty	2022-04-23	12:00-13:00	Empty
4	23	Ahmet Mehmet	Empty	2022-04-29	14:00-15:00	Empty

Figure 5 : Microsoft SQL Server Management Studio “Select Query” page

4.2. MICROSOFT VISUAL STUDIO

Microsoft Visual Studio is a software development environment. It is a programming software utilized to develop computer programs, web services, and many other application types. In my intern project, I required two web services. To create them, I make use of Microsoft Visual Studio. I downloaded and used the last version of the Visual Studio, the 2022 release. You may observe the project structure of Visual Studio's ASP.NET Core application in Figure 6. Also, the file structure of Visual Studio 2022 is different from previous versions. The "Program.cs" file was two separate files in the earlier version. Therefore, it is reduced to a single file for enhancing the file integrity, as shown in Figure 6.



```
Solution Explorer - Folder View
Search Solution Explorer - Folder View (Ctrl+ş)
◢ CarBack (C:\Users\Win10\Desktop\intern\car a
  ◢ CarCheck
    ◢ CarCheck
      ▷ bin
      ◢ Controllers
          DriverDetailController.cs
          WeatherForecastController.cs
      ◢ Migrations
          20220302104511_InitialCreate.cs
          20220302104511_InitialCreate.De
          DriverDetailContextModelSnapsl
      ◢ Models
          DriverDetail.cs
          DriverDetailContext.cs
      ▷ obj
      ▷ Properties
        appsettings.Development.json
        appsettings.json
        CarCheck.csproj
        CarCheck.csproj.user
        Program.cs
        WeatherForecast.cs
    CarCheck.sln
```

Figure 6: Microsoft Visual Studio "file structure" page

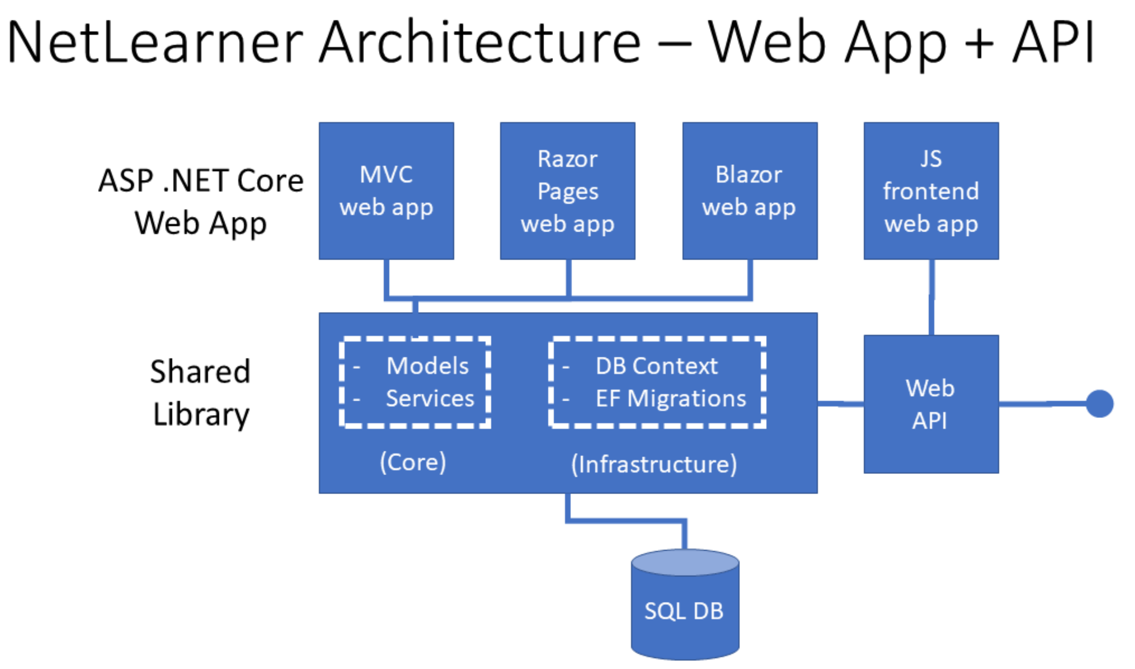
4.2.1. C#

For creating my web service, I utilized the C# programming language. According to TechTerms (2014), C# is a widely used language for creating .NET Framework-based web applications. Furthermore, it is specially designed for Microsoft's .NET Framework. With C#, the programmer may create objects, strings, and other .NET types and use different generic and implicit variable types. Therefore it is beneficial for building web APIs.

4.2.2. ASP.NET CORE FRAMEWORK

ASP.NET Core is one of the open-source frameworks of .NET for creating web APIs. "A framework is a structure that you can build software on" (Codecademy, 2022). This sentence indicates that the user doesn't need to build everything from scratch by using a framework. We might create our program on top of a framework designed for particular purposes. The usage of frameworks might also have additional benefits, such as making debugging and testing the written code easier.

In Figure 6, you can observe the project structure of ASP .NET Core Web App. Also, you can see the integration and connection with other components of a complete web project. Furthermore, connecting the frontend with the database via a Web API is crucial to creating a full design.

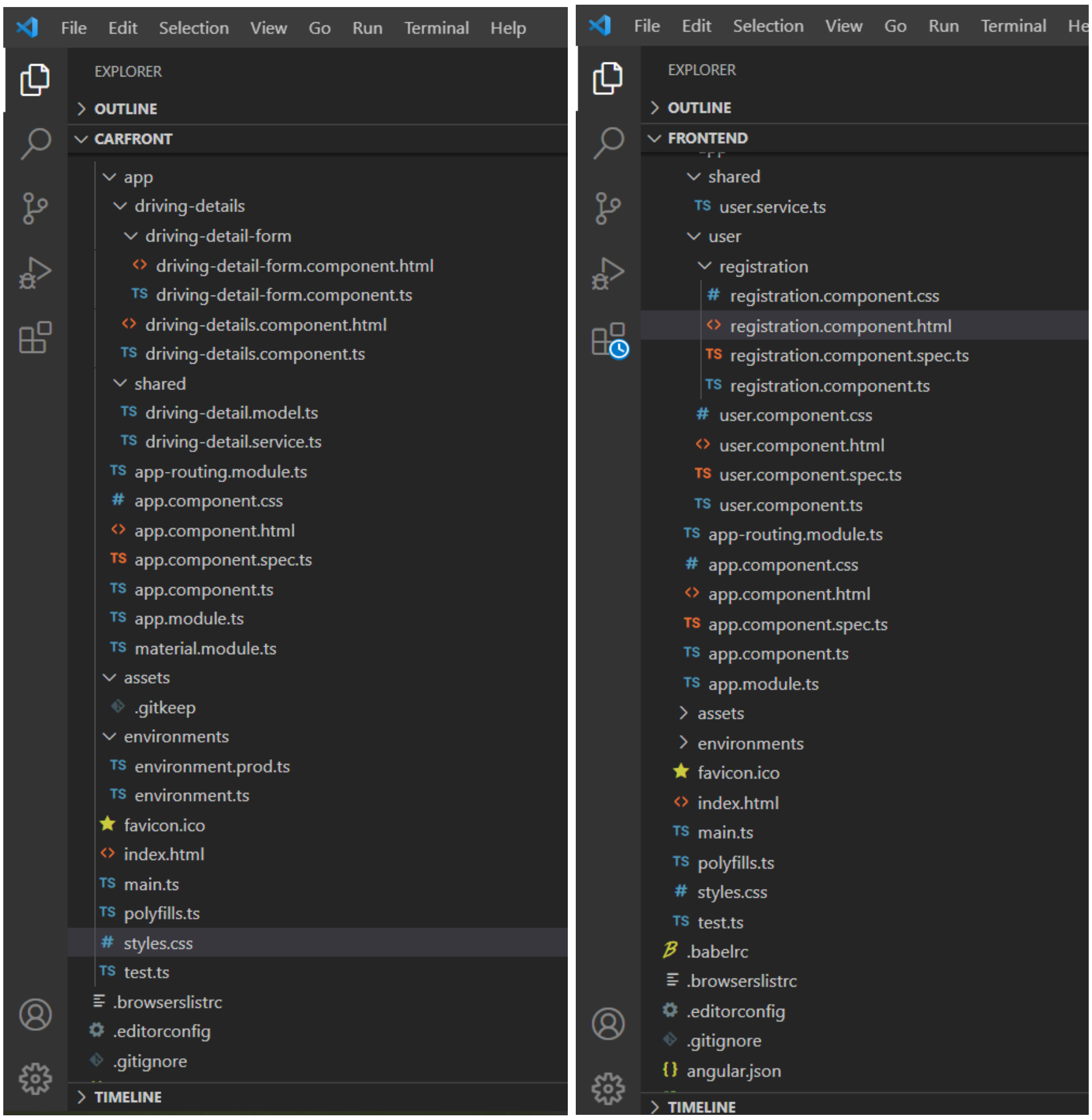


(LaptrinhX, 2019)

Figure 7: Complete design of a Web App with an Web API

4.3. ANGULAR PROJECT STRUCTURE

Angular is a front-end development framework. It allows programmers to create user interfaces using HTML, CSS, and TypeScript programming languages. Additionally, angular supports many functionalities of TypeScript libraries. Several modules and components might be created to serve specific requirements to increase the functionality and handle the complexity of an Angular project. You may observe the project structures of my Angular projects in Figure 8.



```
EXPLORER
> OUTLINE
v CARFRONT
  v app
    v driving-details
      v driving-detail-form
        driving-detail-form.component.html
        driving-detail-form.component.ts
      driving-details.component.html
      driving-details.component.ts
    v shared
      driving-detail.model.ts
      driving-detail.service.ts
    app-routing.module.ts
    app.component.css
    app.component.html
    app.component.spec.ts
    app.component.ts
    app.module.ts
    material.module.ts
  v assets
    .gitkeep
  v environments
    environment.prod.ts
    environment.ts
  favicon.ico
  index.html
  main.ts
  polyfills.ts
  styles.css
  test.ts
  .browserslistrc
  .editorconfig
  .gitignore
> TIMELINE
```

```
EXPLORER
> OUTLINE
v FRONTEND
      v shared
        user.service.ts
      v user
        v registration
          registration.component.css
          registration.component.html
          registration.component.spec.ts
          registration.component.ts
        user.component.css
        user.component.html
        user.component.spec.ts
        user.component.ts
      app-routing.module.ts
      app.component.css
      app.component.html
      app.component.spec.ts
      app.component.ts
      app.module.ts
    > assets
    > environments
    favicon.ico
    index.html
    main.ts
    polyfills.ts
    styles.css
    test.ts
  .babelrc
  .browserslistrc
  .editorconfig
  .gitignore
  angular.json
> TIMELINE
```

Figure 8: Angular project structures of Registiration and Car Checker pages

4.4. POSTMAN

Postman is software for testing web applications to test methods such as sending, getting, updating, and deleting without creating a front-end. I used it before I created my front end, and it was beneficial for testing my methods quickly. It gives you error messages if something is wrong with your methods or if something is wrong with the integrity of your database.

5. DESIGN AND IMPLEMENTATION STAGES

Understanding the company's car usage was my first job. I talked to the company's secretary who is responsible for taking records of cars and drivers. She explained to me that the company has several vehicles, and to follow them, they keep a record of car plate, driver name, and driving date.

They were already using an old application to keep a record of the drivers. The problem with this application was. It was designed for its old cars, which the company doesn't have anymore. And all data entry allowed was according to old car plates. Therefore, I am requested to design a new application to enable them to enter car plates via hand. As an additional feature to the old application, they wanted me to create a place to record the purpose of the entered driving.

5.1. ER DIAGRAM DESIGN

Before starting the registration system project, I did some research on the internet to understand the requirements of my project. I checked other websites' registration pages and their database designs. And I realized that my registration page was for a relatively small company and the system only had one entity, the user. Therefore, I decided to use a single SQL table to record the registered users' data. Later I connected this single-user entity with the whole system, as shown in Figure 9 since this single entity is only for the registration page of the car tracking application. In other words, the registration page is only a small part of my car tracking application. Therefore I kept its design as simple as possible. For creating a user account, they need to enter their name, surname, username, e-mail, and password into the system. After that, the system automatically creates a unique AuthenticationID for each user in the database.

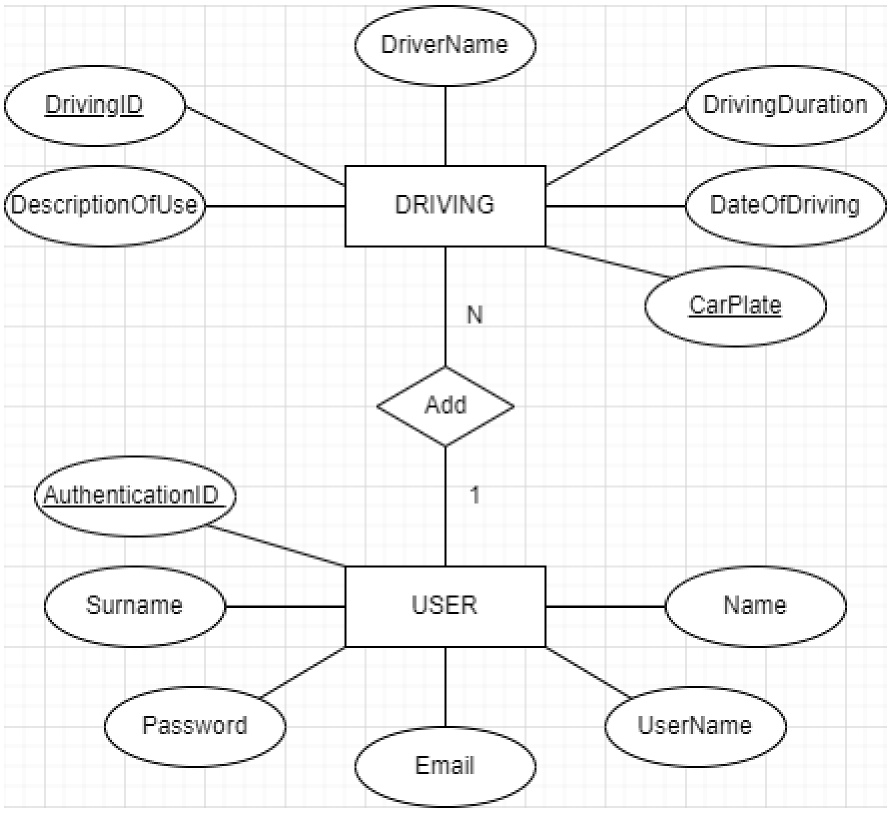
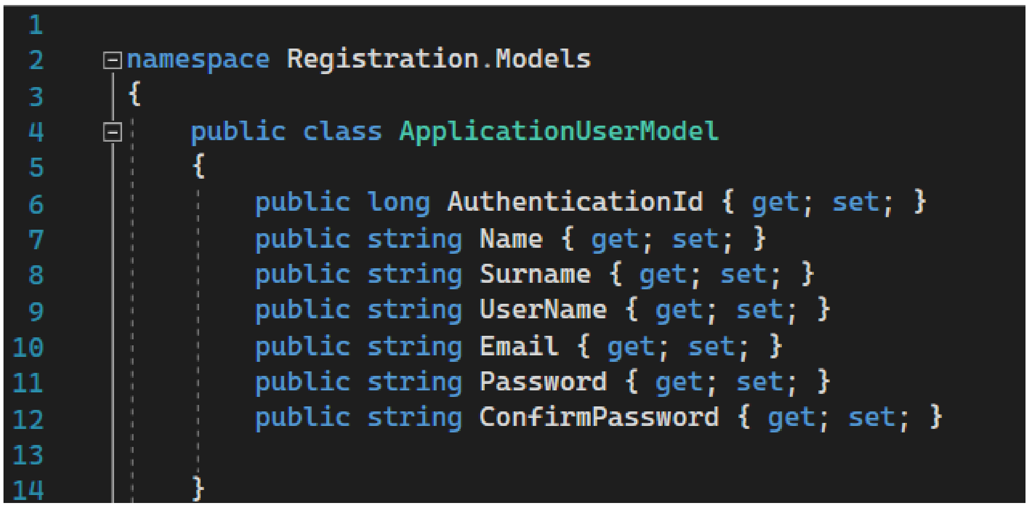


Figure 9: ER Diagram Of The Car Tracking System

In Figure 9, you can see my complete design for the car tracking application and registration page. I draw a complete ER diagram to show the connection between these two parts of my application. According to my design choices, company workers will have authentication to add driving to the database after the registration. Driving includes car plate, date of driving, driving duration, driver name, and use description.

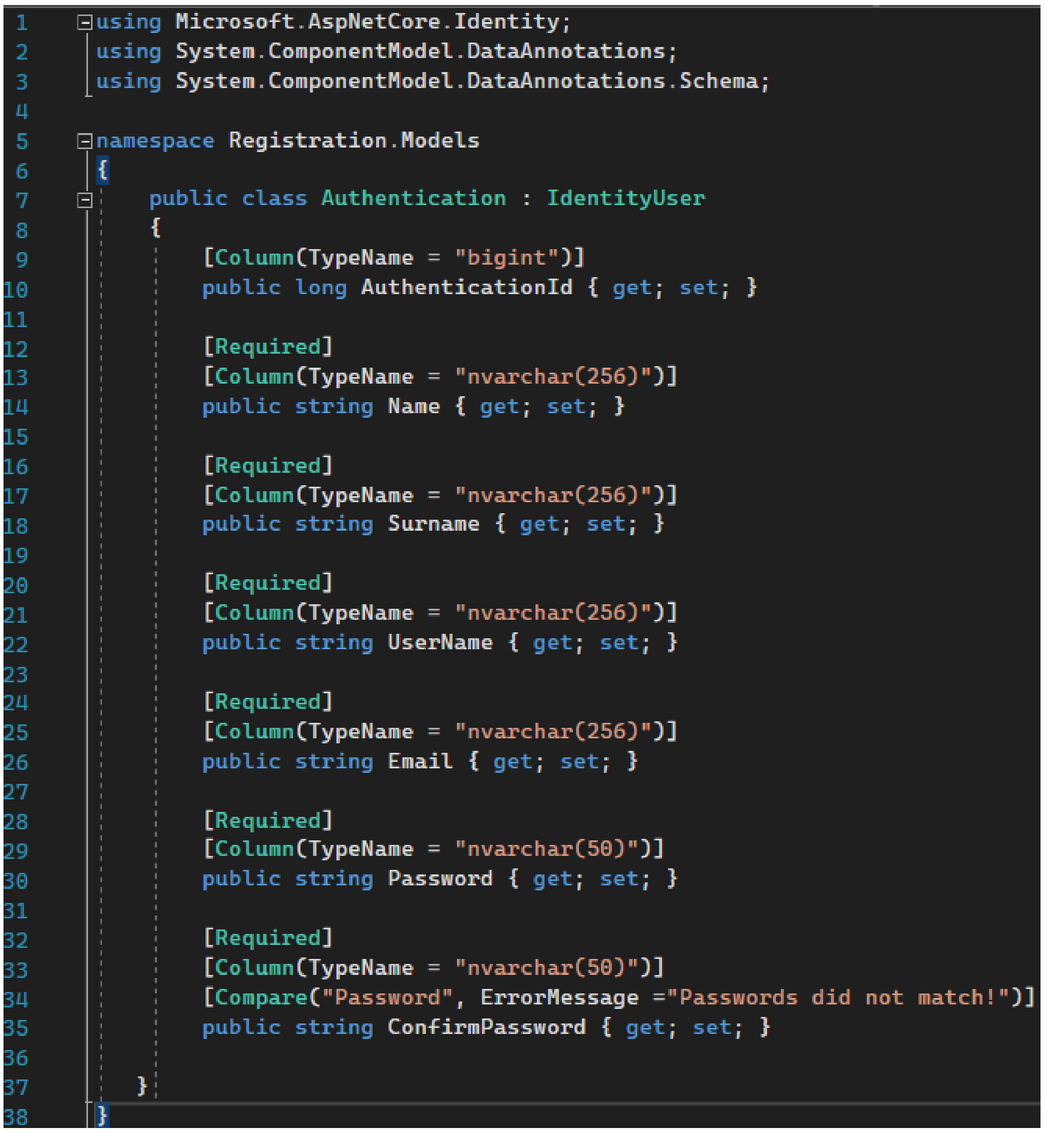
5.2. ASP.NET CORE IMPLEMENTATION

The entities in ER diagram represent classes in the ASP.NET Core project, as you can see in Figure 10. “Authentication” class represents the user entity, and the “Authentication” class has attributes as column names for the SQL table, as shown in Figure 10. Furthermore, I also created a comparison control for ConfirmPassword to match the original Password. If passwords do not match, the entered error message will appear. Since “AuthenticationId” is created automatically by Microsoft SQL Server Management Studio, I didn’t define it as a required attribute.



```
namespace Registration.Models
{
    public class ApplicationUserModel
    {
        public long AuthenticationId { get; set; }
        public string Name { get; set; }
        public string Surname { get; set; }
        public string UserName { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }
        public string ConfirmPassword { get; set; }

    }
```

Figure 10: ASP.NET Core Registration Models

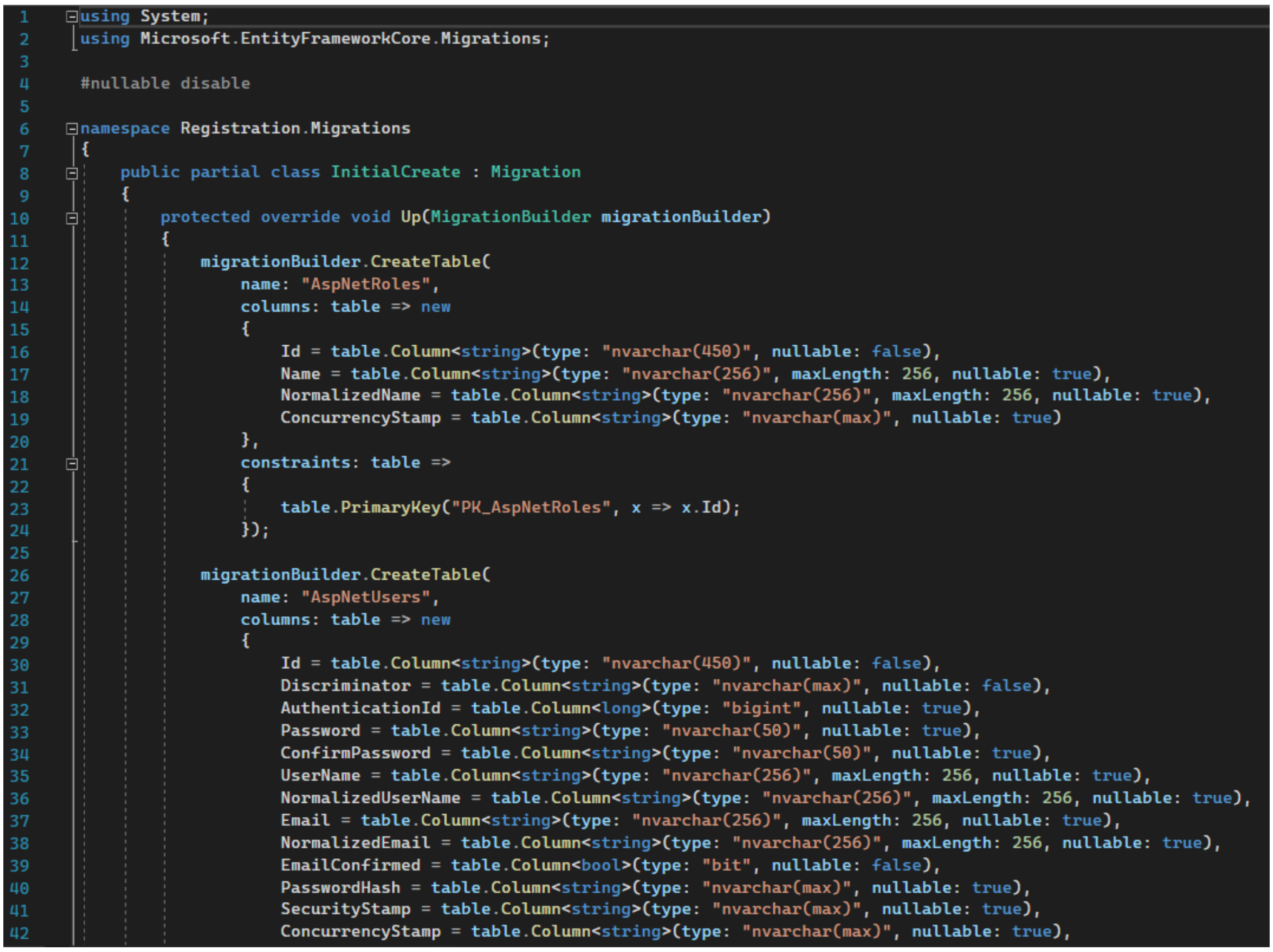


```
using Microsoft.AspNetCore.Identity;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Registration.Models
{
    public class Authentication : IdentityUser
    {
        [Column(TypeName = "bigint")]
        public long AuthenticationId { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(256)")]
        public string Name { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(256)")]
        public string Surname { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(256)")]
        public string UserName { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(256)")]
        public string Email { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(50)")]
        public string Password { get; set; }

        [Required]
        [Column(TypeName = "nvarchar(50)")]
        [Compare("Password", ErrorMessage ="Passwords did not match!")]
        public string ConfirmPassword { get; set; }

    }
}
```

Figure 11: ASP.NET Core “Authentication” Class

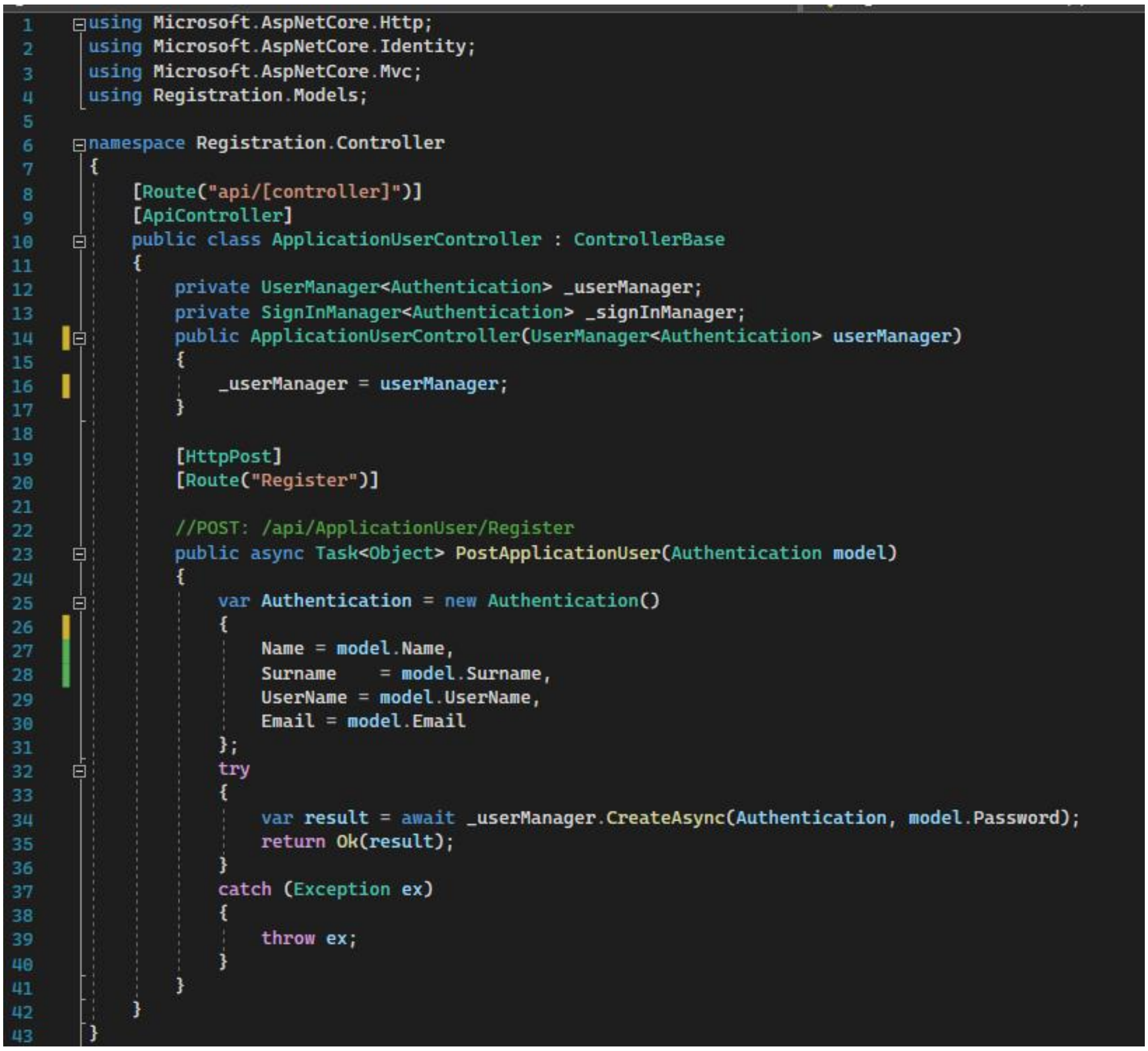
While defining my ‘Authentication’ class, I use ASP.NET Core Identity. “ASP.NET Core Identity is a membership system which allows you to add login functionality to your application.” (Rastogi et al., n.d.) Using ASP.NET Core Identity made my job a lot easier since it efficiently manages user information. As shown in Figure 11, to make us of ASP.NET Core Identity, I build the ‘Authentication’ class with ‘IdentityUser.’

To create and migrate all data into the database, I used migration in ASP.NET Core. Since the migration command creates default tables, as you can see in Figure 12, I changed unnecessary parts and adapted them to my ‘Authentication’ class. After that, I created a ‘Post’ method to allow a user to send their information to the database. You can see my ‘Post’ method in Figure 13.



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

#nullable disable

namespace Registration.Migrations
{
    public partial class InitialCreate : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "AspNetRoles",
                columns: table => new
                {
                    Id = table.Column<string>(type: "nvarchar(450)", nullable: false),
                    Name = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    NormalizedName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetRoles", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUsers",
                columns: table => new
                {
                    Id = table.Column<string>(type: "nvarchar(450)", nullable: false),
                    Discriminator = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    AuthenticationId = table.Column<long>(type: "bigint", nullable: true),
                    Password = table.Column<string>(type: "nvarchar(50)", nullable: true),
                    ConfirmPassword = table.Column<string>(type: "nvarchar(50)", nullable: true),
                    UserName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    NormalizedUserName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    Email = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    NormalizedEmail = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    EmailConfirmed = table.Column<bool>(type: "bit", nullable: false),
                    PasswordHash = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    SecurityStamp = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true),
```

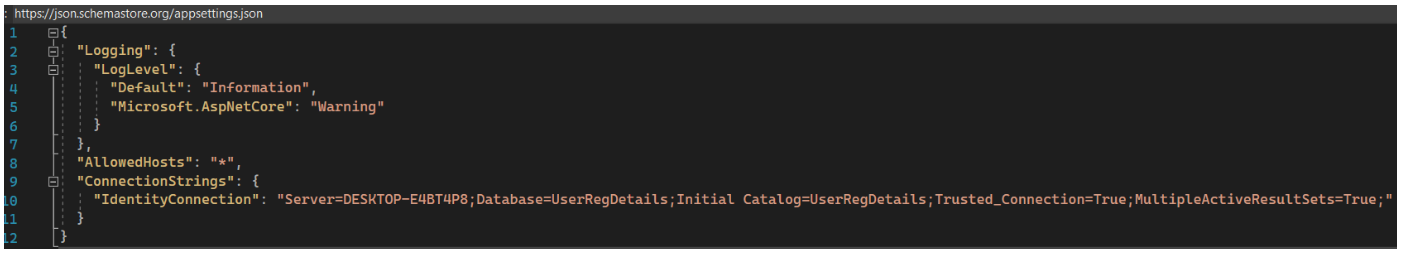
Figure 12: Using migration to create an AspNetUsers table



```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Registration.Models;

namespace Registration.Controller
{
    [Route("api/[controller]")]
    [ApiController]
    public class ApplicationUserController : ControllerBase
    {
        private UserManager<Authentication> _userManager;
        private SignInManager<Authentication> _signInManager;
        public ApplicationUserController(UserManager<Authentication> userManager)
        {
            _userManager = userManager;
        }

        [HttpPost]
        [Route("Register")]

        //POST: /api/ApplicationUser/Register
        public async Task<Object> PostApplicationUser(Authentication model)
        {
            var Authentication = new Authentication()
            {
                Name = model.Name,
                Surname     = model.Surname,
                UserName = model.UserName,
                Email = model.Email
            };
            try
            {
                var result = await _userManager.CreateAsync(Authentication, model.Password);
                return Ok(result);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
    }
}
```

Figure 13: Post method for registering a user in ASP.NET Core

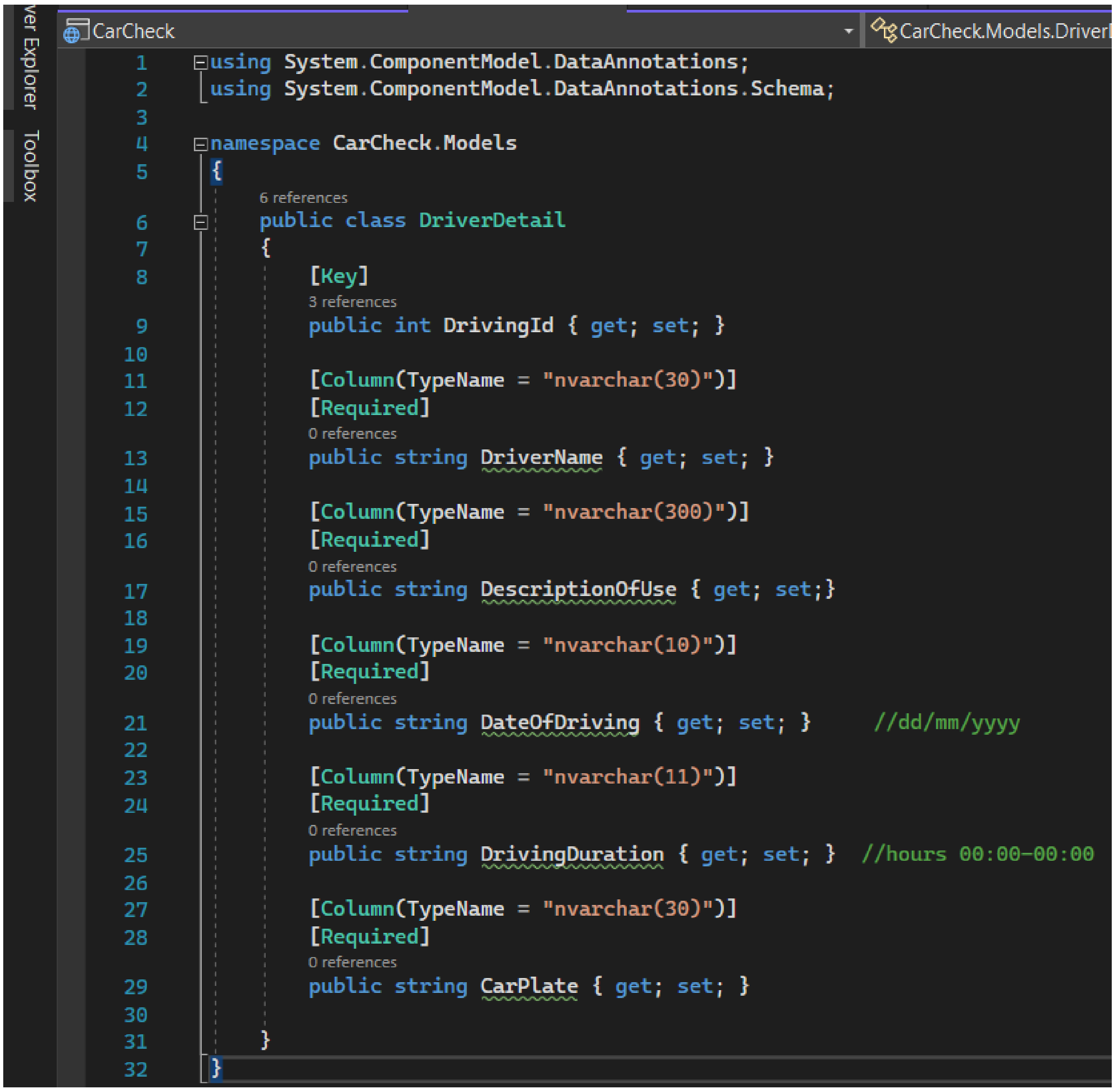
A connection string may include information of destination, connection protocols, and source server. It allows the application to connect to a specified database in ASP.NET Core. As you can see in Figure 14, I use a connection string to connect the UserRegDetails database with my application.



```
https://json.schemastore.org/appsettings.json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "IdentityConnection": "Server=DESKTOP-E4BT4P8;Database=UserRegDetails;Initial Catalog=UserRegDetails;Trusted_Connection=True;MultipleActiveResultSets=True;"
  }
}
```

Figure 14: UserRegDetails database's "Connection String" in ASP.NET Core

After registration models and methods were done, I started designing the car tracking part of the system. You can see the model part of my car tracking system design in Figure 14. The car tracking part of my application includes the “DriverDetail” class, which represents the ‘Driving’ entity. “DriverDetail” class has attributes as column names for SQL table as shown in Figure 15.

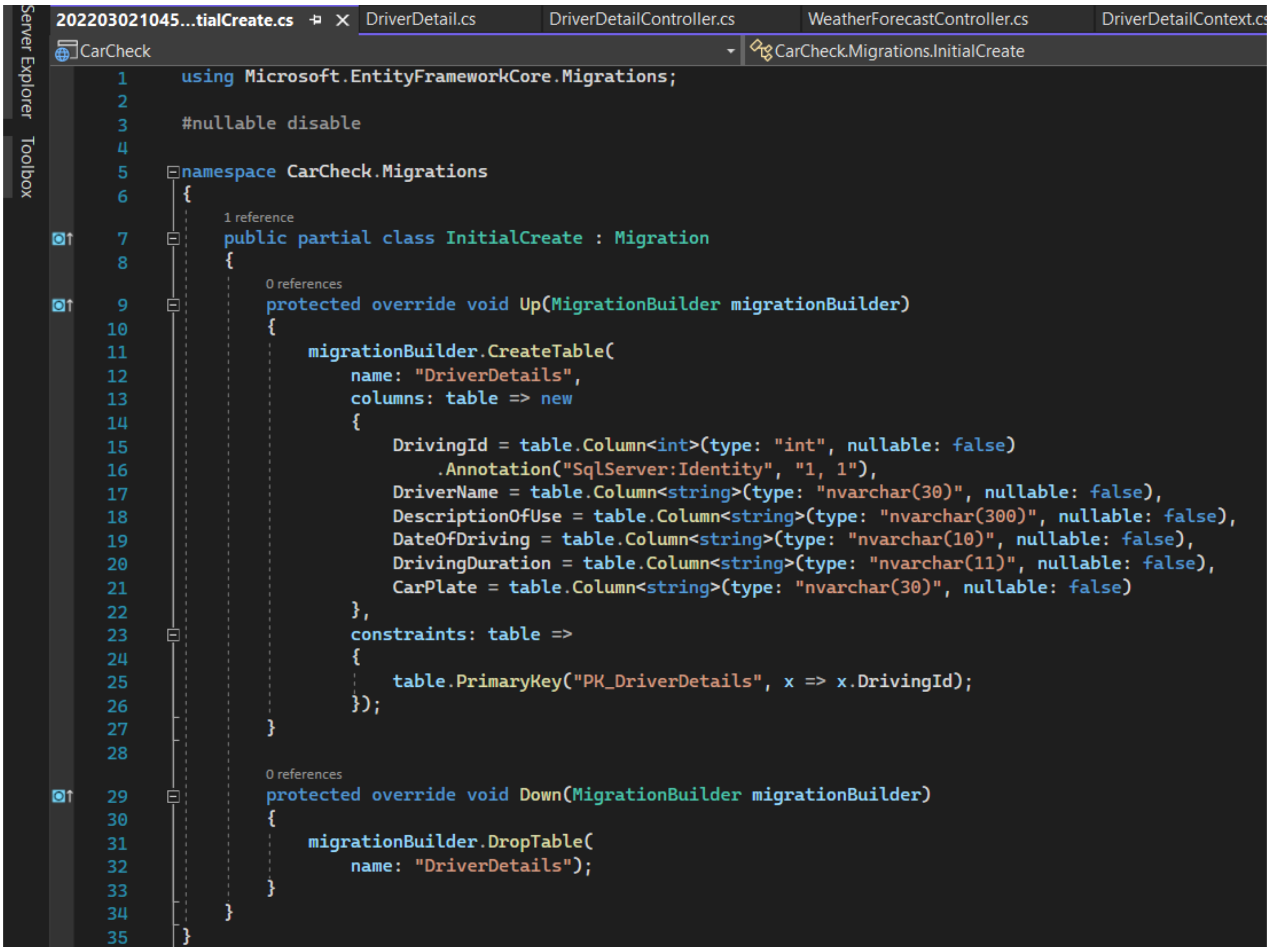


```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace CarCheck.Models
{
    6 references
    public class DriverDetail
    {
        [Key]
        3 references
        public int DrivingId { get; set; }

        [Column(TypeName = "nvarchar(30)")]
        [Required]
        0 references
        public string DriverName { get; set; }

        [Column(TypeName = "nvarchar(300)")]
        [Required]
        0 references
        public string DescriptionOfUse { get; set;}

        [Column(TypeName = "nvarchar(10)")]
        [Required]
        0 references
        public string DateOfDriving { get; set; }      //dd/mm/yyyy

        [Column(TypeName = "nvarchar(11)")]
        [Required]
        0 references
        public string DrivingDuration { get; set; }  //hours 00:00-00:00

        [Column(TypeName = "nvarchar(30)")]
        [Required]
        0 references
        public string CarPlate { get; set; }

    }
}
```

Figure 15: “DriverDetail” class

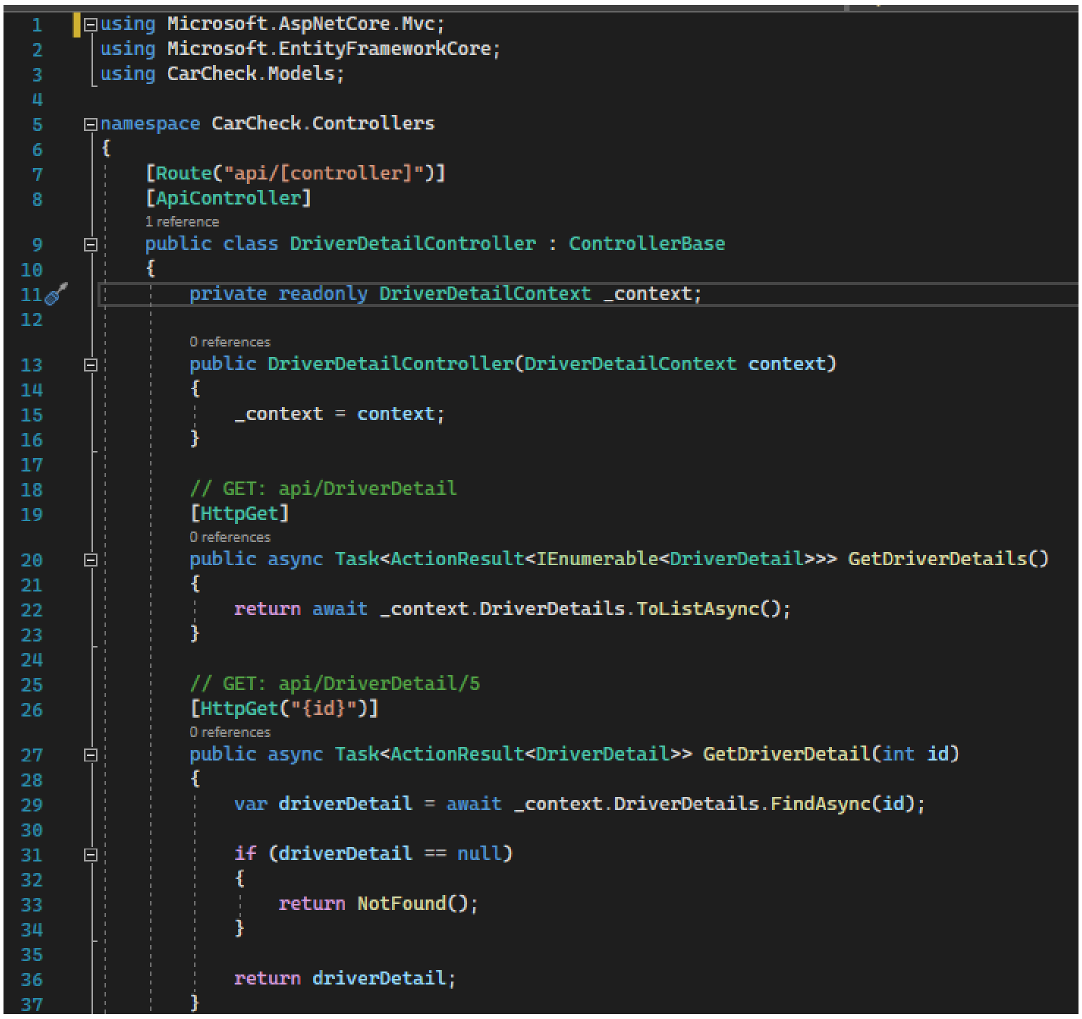
After creating the “DriverDetail” class, I built a controller for introducing get, delete, put and post methods, as shown in Figures 17, 18, and 19. These methods are called ASP.NET Core Web API CRUD(Create, Read, Update, Delete) operations.

As shown in Figure 16, I created the DriverDetails table via migration command to keep a record of all driving details. Furthermore, for each new driving, it automatically makes a unique DrivingId.


Server Explorer
Toolbox
202203021045...tialCreate.cs
DriverDetail.cs
DriverDetailController.cs
WeatherForecastController.cs
DriverDetailContext.cs
CarCheck
CarCheck.Migrations.InitialCreate

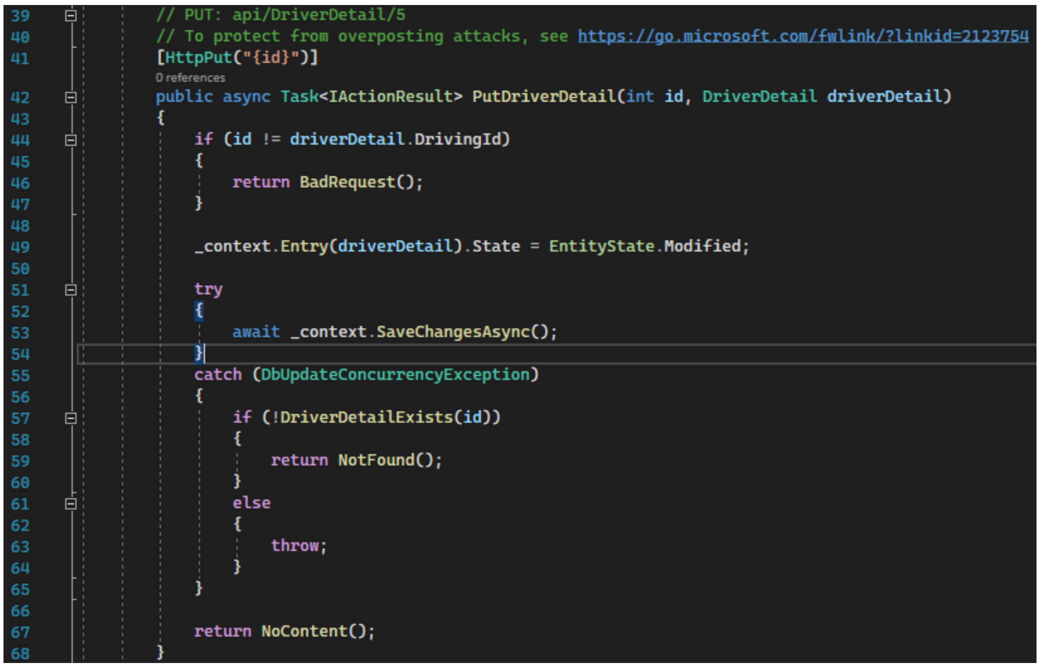
```
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace CarCheck.Migrations
{
    1 reference
    public partial class InitialCreate : Migration
    {
        0 references
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "DriverDetails",
                columns: table => new
                {
                    DrivingId = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    DriverName = table.Column<string>(type: "nvarchar(30)", nullable: false),
                    DescriptionOfUse = table.Column<string>(type: "nvarchar(300)", nullable: false),
                    DateOfDriving = table.Column<string>(type: "nvarchar(10)", nullable: false),
                    DrivingDuration = table.Column<string>(type: "nvarchar(11)", nullable: false),
                    CarPlate = table.Column<string>(type: "nvarchar(30)", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_DriverDetails", x => x.DrivingId);
                });
        }

        0 references
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "DriverDetails");
        }
    }
}
```

Figure 16: Using migration to create "DriverDetails" table



```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using CarCheck.Models;

namespace CarCheck.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class DriverDetailController : ControllerBase
    {
        private readonly DriverDetailContext _context;

        0 references
        public DriverDetailController(DriverDetailContext context)
        {
            _context = context;
        }

        // GET: api/DriverDetail
        [HttpGet]
        0 references
        public async Task<ActionResult<IEnumerable<DriverDetail>>> GetDriverDetails()
        {
            return await _context.DriverDetails.ToListAsync();
        }

        // GET: api/DriverDetail/5
        [HttpGet("{id}")]
        0 references
        public async Task<ActionResult<DriverDetail>> GetDriverDetail(int id)
        {
            var driverDetail = await _context.DriverDetails.FindAsync(id);

            if (driverDetail == null)
            {
                return NotFound();
            }

            return driverDetail;
        }
```

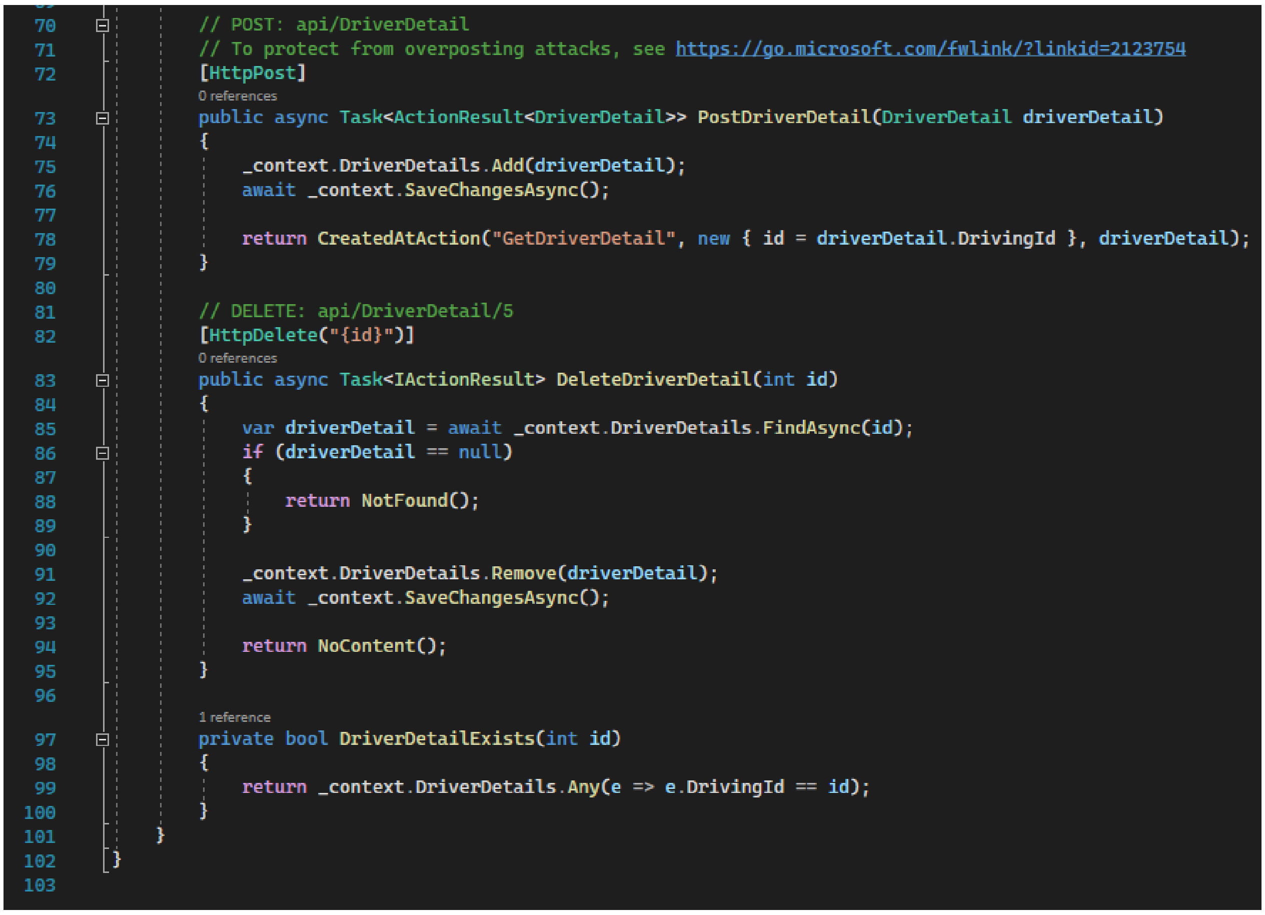
Figure 17: Get method for DriverDetails



```
        // PUT: api/DriverDetail/5
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPut("{id}")]
        0 references
        public async Task<IActionResult> PutDriverDetail(int id, DriverDetail driverDetail)
        {
            if (id != driverDetail.DrivingId)
            {
                return BadRequest();
            }

            _context.Entry(driverDetail).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!DriverDetailExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }
```

Figure 18: Put method for DriverDetails

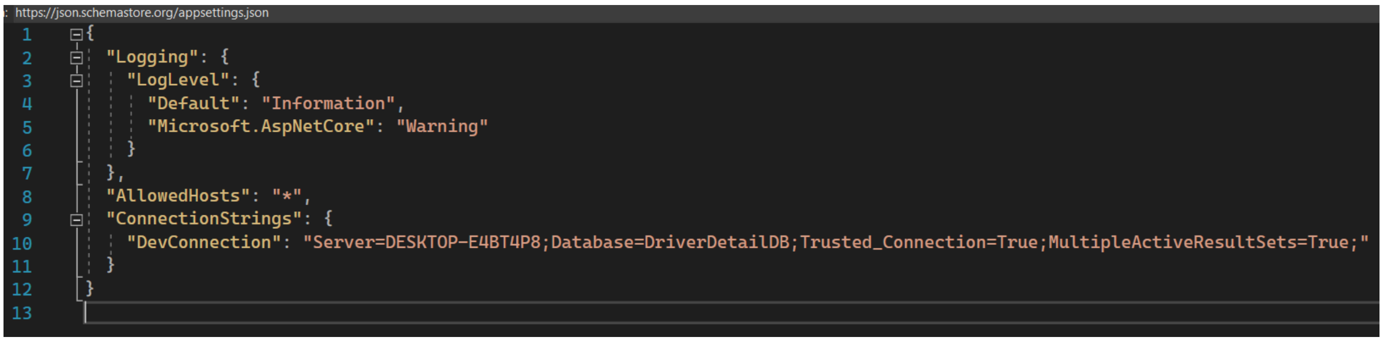
As shown in Figure 17, the created HttpGet method will be used to get data from Web API; however, the HttpPost method, as shown in Figure 19, is developed to send new data to the database. Also, as shown in Figure 18 HttpPut method will be used for updating an existing item via Web API. Lastly, the HttpDelete method will be used for deleting existing data from the database.



```
        // POST: api/DriverDetail
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        0 references
        public async Task<ActionResult<DriverDetail>> PostDriverDetail(DriverDetail driverDetail)
        {
            _context.DriverDetails.Add(driverDetail);
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetDriverDetail", new { id = driverDetail.DrivingId }, driverDetail);
        }

        // DELETE: api/DriverDetail/5
        [HttpDelete("{id}")]
        0 references
        public async Task<IActionResult> DeleteDriverDetail(int id)
        {
            var driverDetail = await _context.DriverDetails.FindAsync(id);
            if (driverDetail == null)
            {
                return NotFound();
            }

            _context.DriverDetails.Remove(driverDetail);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        1 reference
        private bool DriverDetailExists(int id)
        {
            return _context.DriverDetails.Any(e => e.DrivingId == id);
        }
    }
}
```

Figure 19: Post and delete methods for DriverDetails

As you can see in Figure 20, I use a connection string to connect the DriverDetailDB database with my application.

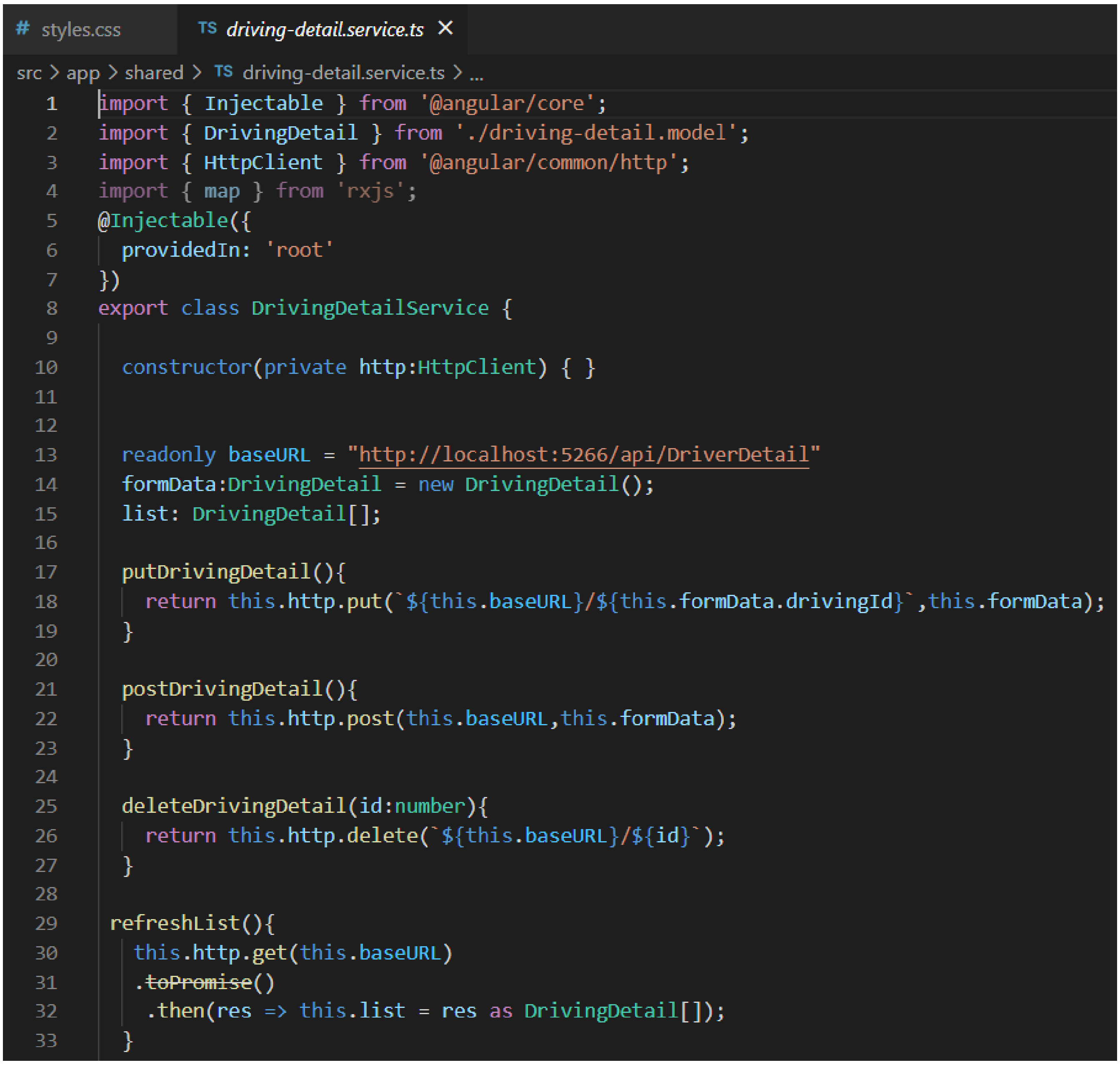


```
https://json.schemastore.org/appsettings.json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DevConnection": "Server=DESKTOP-E4BT4P8;Database=DriverDetailDB;Trusted_Connection=True;MultipleActiveResultSets=True;"
  }
}
```

Figure 20: DriverDetailDB database connection string in ASP.NET Core

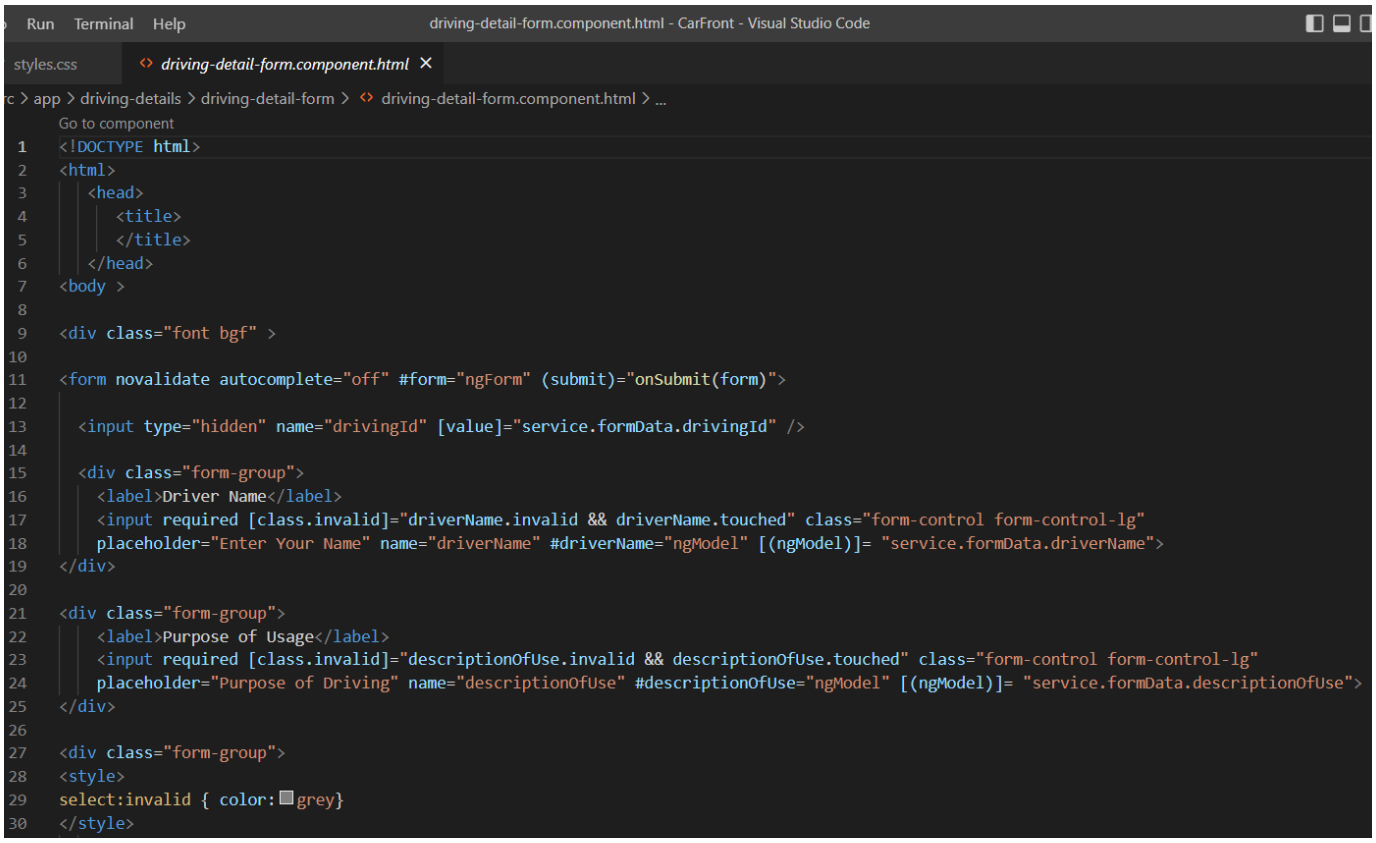
5.3. ANGULAR IMPLEMENTATION

After finishing my back-end development, I created two basic interfaces for my application using Angular. Since I have developed interfaces as secondary tasks, I did not detail their project structure in my report. However, as shown in Figure 21, you can observe how my Angular project reaches the methods created in the back-end. Also, In Figures 22 and 23, you may see how my main HTML page structure is designed for Car Tracking main page.


# styles.css
TS driving-detail.service.ts ×
src > app > shared > TS driving-detail.service.ts > ...

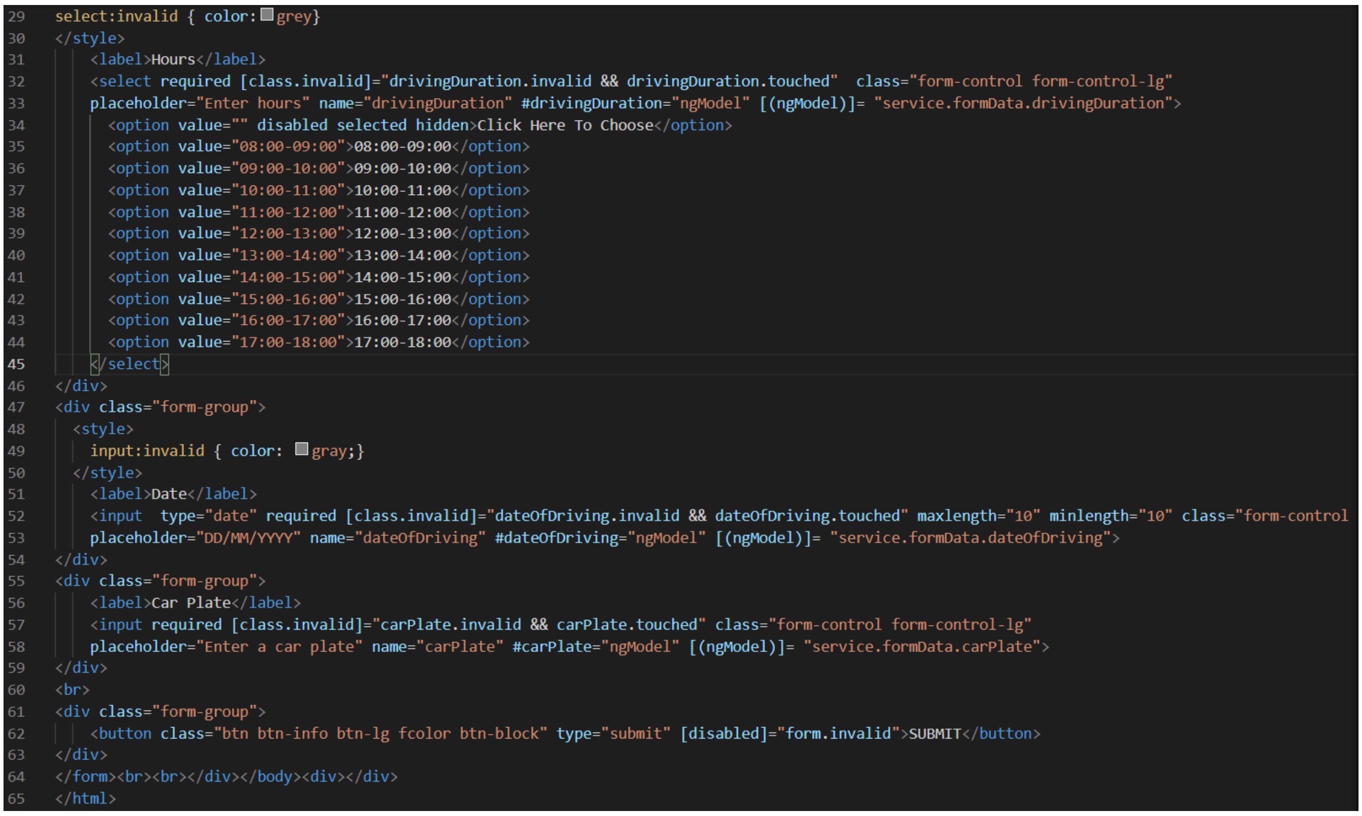
```typescript
import { Injectable } from '@angular/core';
import { DrivingDetail } from './driving-detail.model';
import { HttpClient } from '@angular/common/http';
import { map } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class DrivingDetailService {

  constructor(private http:HttpClient) { }


  readonly baseURL = "http://localhost:5266/api/DriverDetail"
  formData:DrivingDetail = new DrivingDetail();
  list: DrivingDetail[];

  putDrivingDetail(){
    return this.http.put(`${this.baseURL}/${this.formData.drivingId}`,this.formData);
  }

  postDrivingDetail(){
    return this.http.post(this.baseURL,this.formData);
  }

  deleteDrivingDetail(id:number){
    return this.http.delete(`${this.baseURL}/${id}`);
  }

 refreshList(){
  this.http.get(this.baseURL)
  .toPromise()
   .then(res => this.list = res as DrivingDetail[]);
 }
```

Figure 21: Using put, post, delete methods with Angular


Run Terminal Help
driving-detail-form.component.html - CarFront - Visual Studio Code
styles.css
driving-detail-form.component.html
c > app > driving-details > driving-detail-form > driving-detail-form.component.html > ...
Go to component

```html
<!DOCTYPE html>
<html>
    <head>
        <title>
        </title>
    </head>
<body >

<div class="font bgf" >

<form novalidate autocomplete="off" #form="ngForm" (submit)="onSubmit(form)">

  <input type="hidden" name="drivingId" [value]="service.formData.drivingId" />

  <div class="form-group">
    <label>Driver Name</label>
    <input required [class.invalid]="driverName.invalid && driverName.touched" class="form-control form-control-lg"
    placeholder="Enter Your Name" name="driverName" #driverName="ngModel" [(ngModel)]= "service.formData.driverName">
</div>

<div class="form-group">
    <label>Purpose of Usage</label>
    <input required [class.invalid]="descriptionOfUse.invalid && descriptionOfUse.touched" class="form-control form-control-lg"
    placeholder="Purpose of Driving" name="descriptionOfUse" #descriptionOfUse="ngModel" [(ngModel)]= "service.formData.descriptionOfUse">
</div>

<div class="form-group">
<style>
select:invalid { color:■grey}
</style>
```

Figure 22: HTML design of car tracking application



```html
select:invalid { color:■grey}
</style>
    <label>Hours</label>
    <select required [class.invalid]="drivingDuration.invalid && drivingDuration.touched"  class="form-control form-control-lg"
    placeholder="Enter hours" name="drivingDuration" #drivingDuration="ngModel" [(ngModel)]= "service.formData.drivingDuration">
      <option value="" disabled selected hidden>Click Here To Choose</option>
      <option value="08:00-09:00">08:00-09:00</option>
      <option value="09:00-10:00">09:00-10:00</option>
      <option value="10:00-11:00">10:00-11:00</option>
      <option value="11:00-12:00">11:00-12:00</option>
      <option value="12:00-13:00">12:00-13:00</option>
      <option value="13:00-14:00">13:00-14:00</option>
      <option value="14:00-15:00">14:00-15:00</option>
      <option value="15:00-16:00">15:00-16:00</option>
      <option value="16:00-17:00">16:00-17:00</option>
      <option value="17:00-18:00">17:00-18:00</option>
    </select>
</div>
<div class="form-group">
  <style>
    input:invalid { color: ■gray;}
  </style>
    <label>Date</label>
    <input  type="date" required [class.invalid]="dateOfDriving.invalid && dateOfDriving.touched" maxlength="10" minlength="10" class="form-control
    placeholder="DD/MM/YYYY" name="dateOfDriving" #dateOfDriving="ngModel" [(ngModel)]= "service.formData.dateOfDriving">
</div>
<div class="form-group">
    <label>Car Plate</label>
    <input required [class.invalid]="carPlate.invalid && carPlate.touched" class="form-control form-control-lg"
    placeholder="Enter a car plate" name="carPlate" #carPlate="ngModel" [(ngModel)]= "service.formData.carPlate">
</div>
<br>
<div class="form-group">
    <button class="btn btn-info btn-lg fcolor btn-block" type="submit" [disabled]="form.invalid">SUBMIT</button>
</div>
</form><br><br></div></body><div></div>
</html>
```

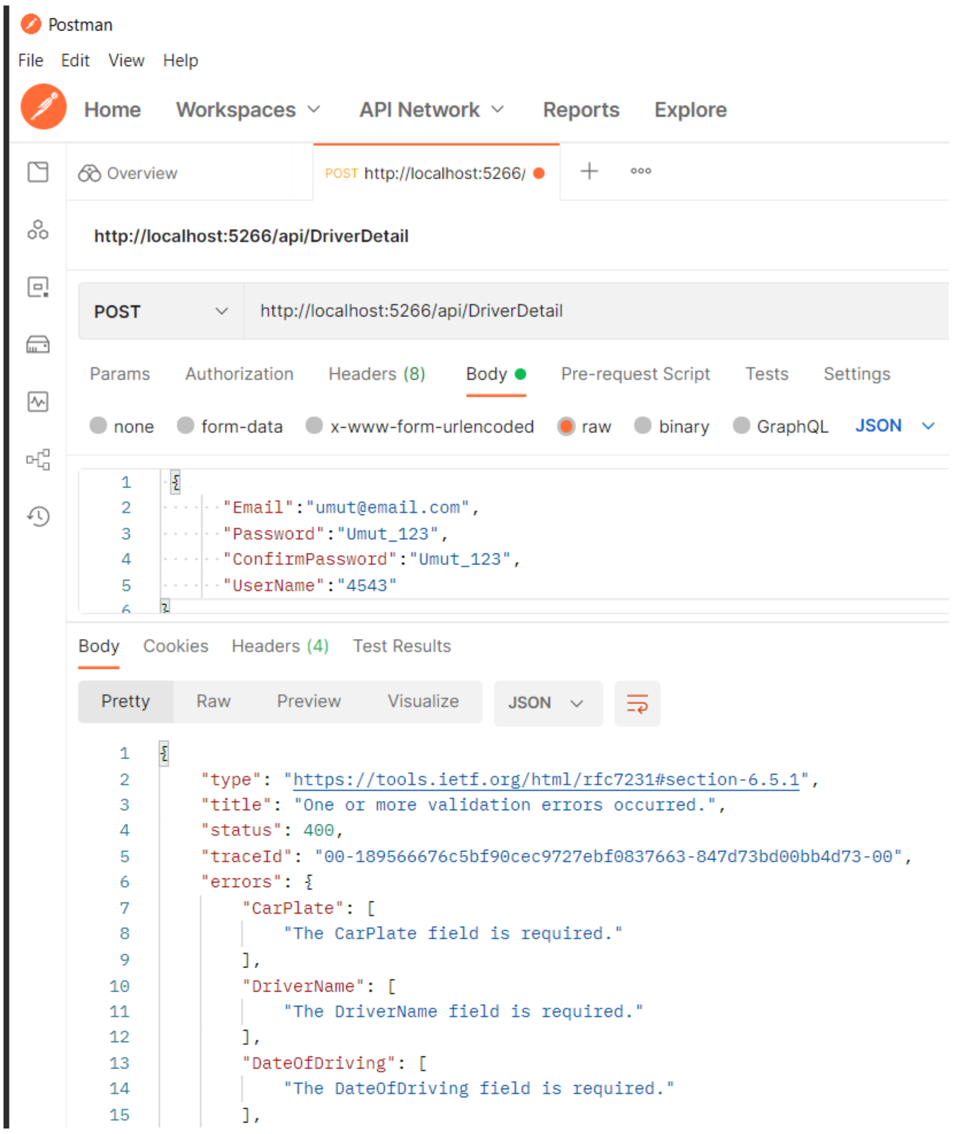
Figure 23:HTML design of car tracking application part two

6. TESTING STAGES

After I fully completed my application, my Supervisor asked me to test it via Postman and open API support interface. It benefited me to understand how my application works with different inputs and what it lacks. I also detected possible errors and safety issues.

6.1. POSTMAN TEST AND OPEN API SUPPORT TEST

As shown in figure 24, by using Postman, I observed that my web API rejects a post request since the structure of the post request is invalid. However, as shown in Figure 25, when a user enters valid inputs in a correct structure, it successfully completes the post operation.


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Overview
POST http://localhost:5266/
Params Authorization Headers (8) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Body Cookies Headers (4) Test Results
Pretty Raw Preview Visualize JSON

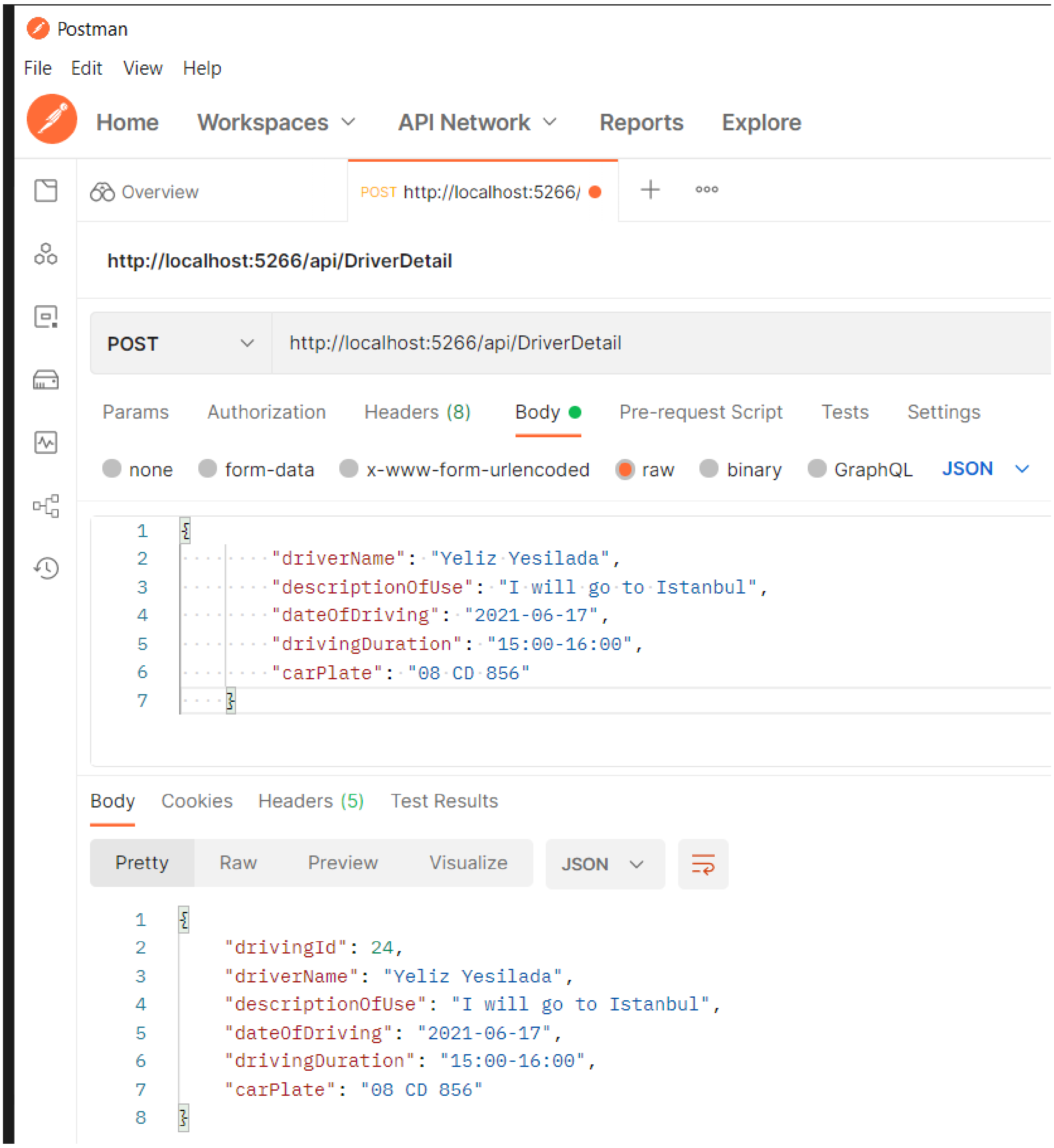
http://localhost:5266/api/DriverDetail

POST http://localhost:5266/api/DriverDetail

```
{
    "Email":"umut@email.com",
    "Password":"Umut_123",
    "ConfirmPassword":"Umut_123",
    "UserName":"4543"
}
```

```
{
    "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
    "title": "One or more validation errors occurred.",
    "status": 400,
    "traceId": "00-189566676c5bf90cec9727ebf0837663-847d73bd00bb4d73-00",
    "errors": {
        "CarPlate": [
            "The CarPlate field is required."
        ],
        "DriverName": [
            "The DriverName field is required."
        ],
        "DateOfDriving": [
            "The DateOfDriving field is required."
        ],
```

Figure 24: Testing wrong input structure by using post method via Postman


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Overview
POST http://localhost:5266/
Params Authorization Headers (8) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Body Cookies Headers (5) Test Results
Pretty Raw Preview Visualize JSON

http://localhost:5266/api/DriverDetail

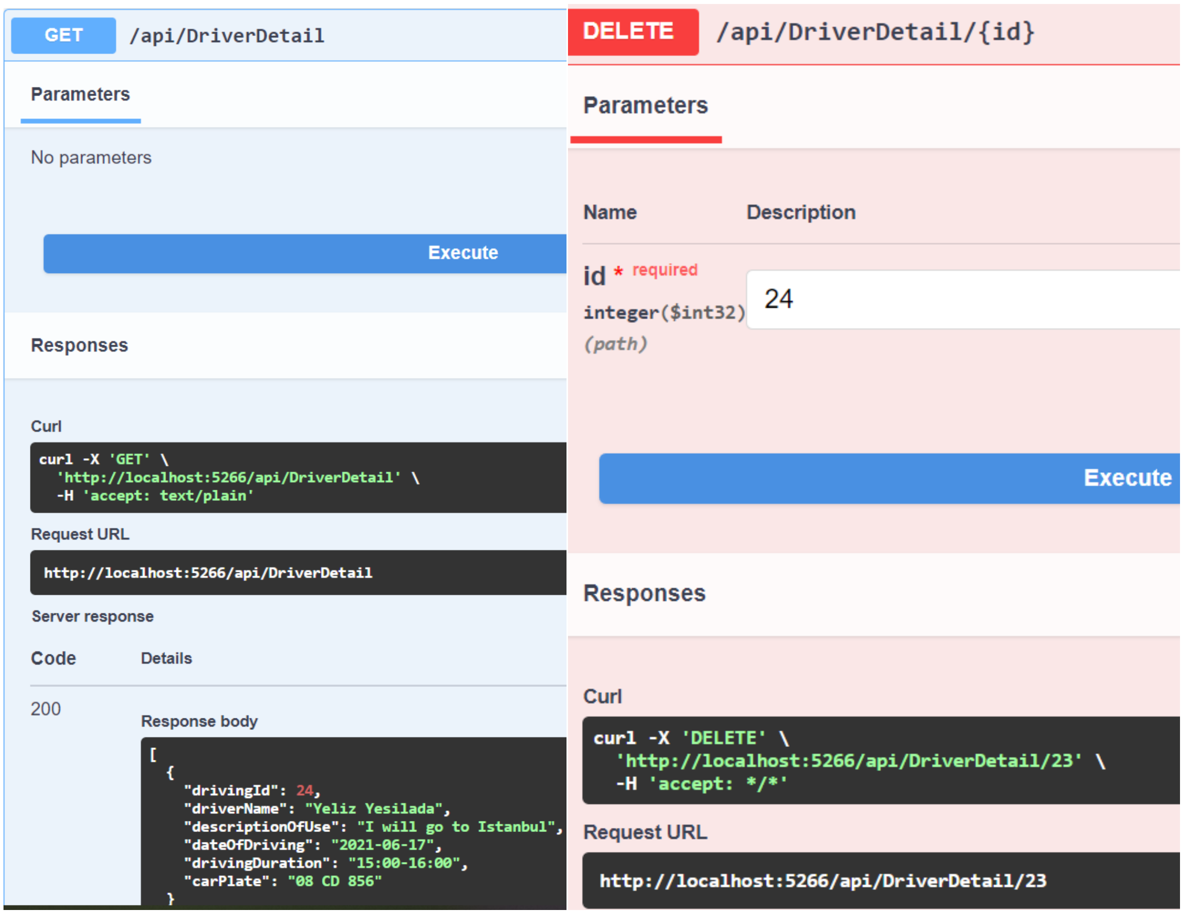
POST http://localhost:5266/api/DriverDetail

```
{
    "driverName": "Yeliz Yesilada",
    "descriptionOfUse": "I will go to Istanbul",
    "dateOfDriving": "2021-06-17",
    "drivingDuration": "15:00-16:00",
    "carPlate": "08 CD 856"
}
```

```
{
    "drivingId": 24,
    "driverName": "Yeliz Yesilada",
    "descriptionOfUse": "I will go to Istanbul",
    "dateOfDriving": "2021-06-17",
    "drivingDuration": "15:00-16:00",
    "carPlate": "08 CD 856"
}
```

Figure 25: Testing correct input structure by using post method via Postman

I also tested my get, delete and update methods via the open API support tool of ASP.NET Core. It is similar to Postman and allows us to test our methods quickly. You can see testing of get and delete methods in Figure 26. The delete method allows the user to enter the ID as a parameter and delete the whole tuple from the database that belongs to that ID. However, the get method shows whole tuples in a table, and columns belong to that tuple.



Curl

```
curl -X 'GET' \
  'http://localhost:5266/api/DriverDetail' \
  -H 'accept: text/plain'
```

Request URL

```
http://localhost:5266/api/DriverDetail
```

Server response

Code	Details
200	Response body

```
[
  {
    "drivingId": 24,
    "driverName": "Yeliz Yesilada",
    "descriptionOfUse": "I will go to Istanbul",
    "dateOfDriving": "2021-06-17",
    "drivingDuration": "15:00-16:00",
    "carPlate": "08 CD 856"
  }
```

Name	Description
id * required integer($int32) (path)	24

Curl

```
curl -X 'DELETE' \
  'http://localhost:5266/api/DriverDetail/23' \
  -H 'accept: */*'
```

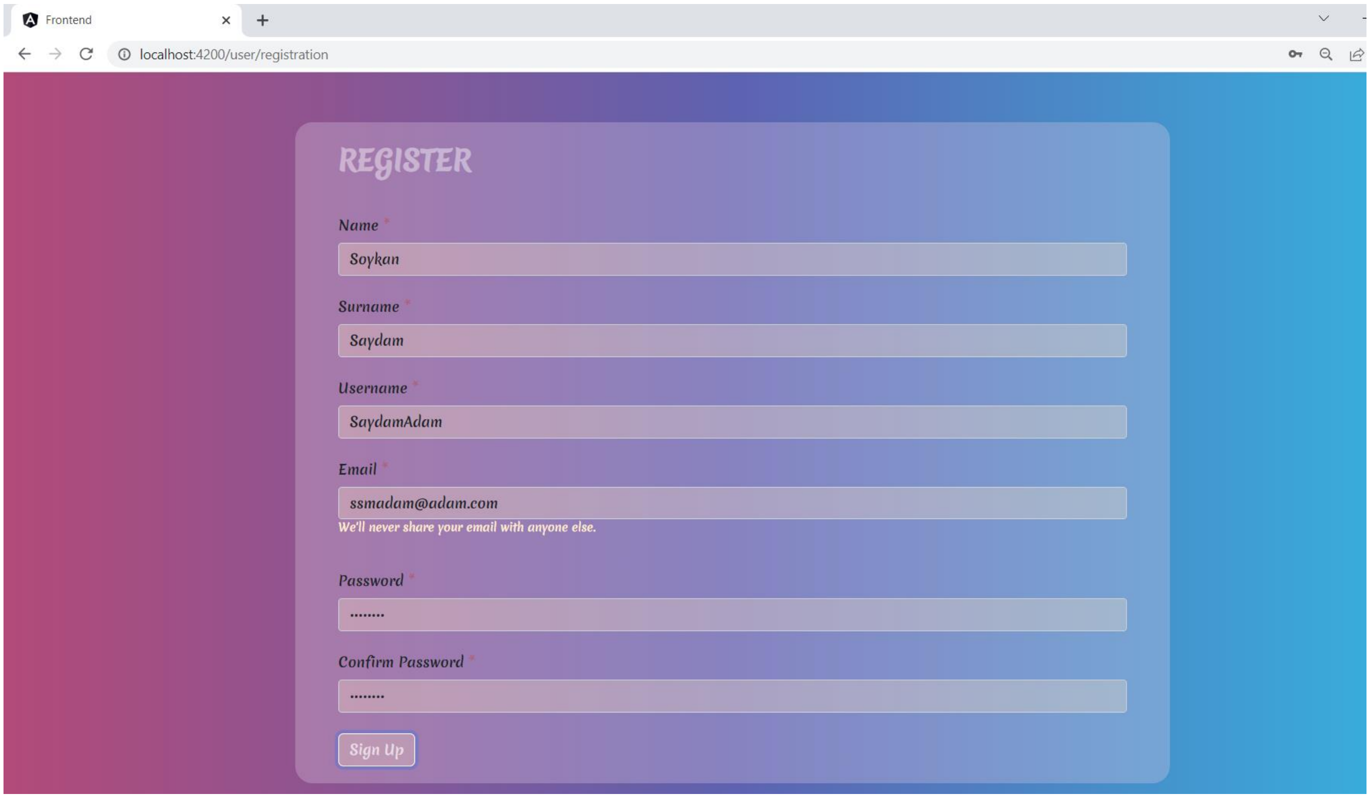
Request URL

```
http://localhost:5266/api/DriverDetail/23
```

Figure 26: Testing get and delete methods via open API Support

6.2. INTERFACE AND INTEGRITY TEST

After observing that all my methods were working without a problem, I decided to test the whole project with its interfaces. As shown in Figure 27, I created a registration page interface and tested it by registering a new user. After that, I tested my Car Checker interface, as shown in Figure 28. I also tested the implementation of delete, post, and update methods to make manipulations over the table, as shown in Figures 29,30, and 31, respectively. By finishing the interface and integrity test step, my project reached its final version.



Frontend

localhost:4200/user/registration

REGISTER

Name *

Soykan

Surname *

Saydam

Username *

SaydamAdam

Email *

ssmadam@adam.com

We'll never share your email with anyone else.

Password *

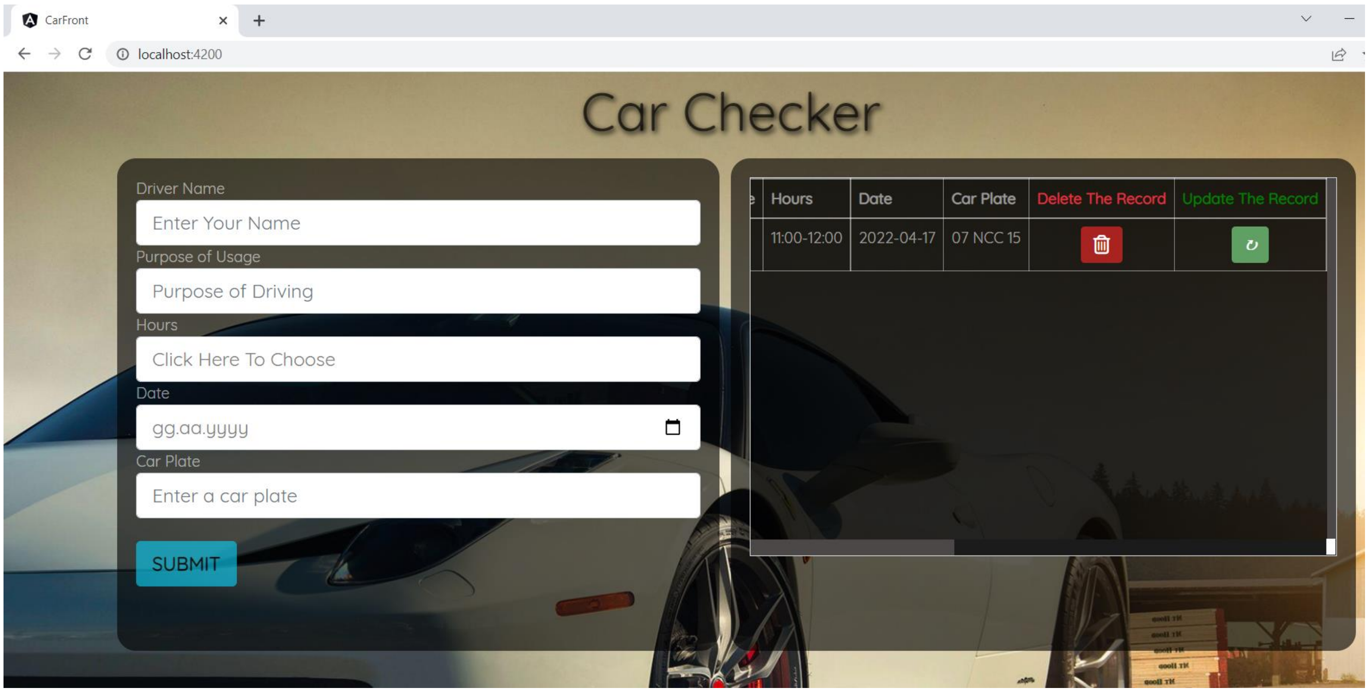
·······

Confirm Password *

·······

Sign Up

Figure 27: "Registiration Interface" page



CarFront

localhost:4200

Car Checker

Driver Name

Enter Your Name

Purpose of Usage

Purpose of Driving

Hours

Click Here To Choose

Date

gg.aa.yyyy

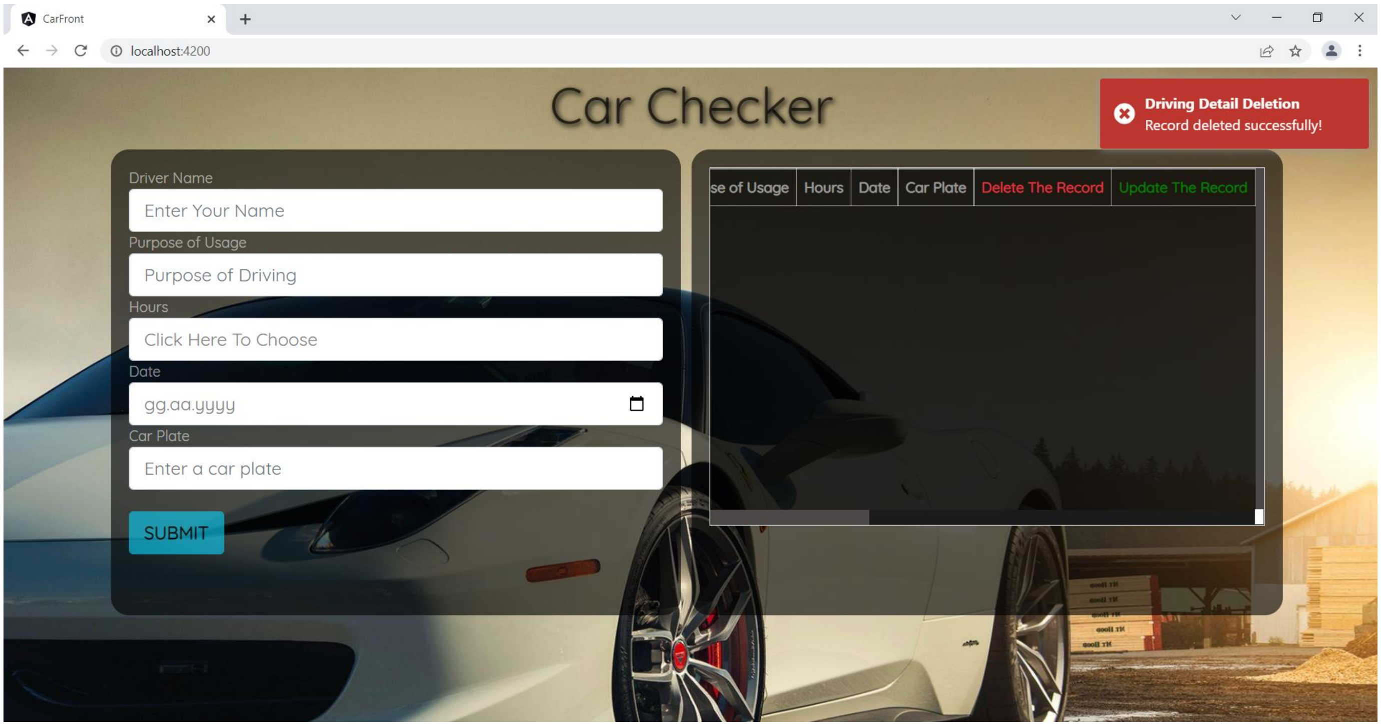
Car Plate

Enter a car plate

SUBMIT

Hours	Date	Car Plate	Delete The Record	Update The Record
11:00-12:00	2022-04-17	07 NCC 15		

Figure 28: "Car Checker Interface" page


CarFront

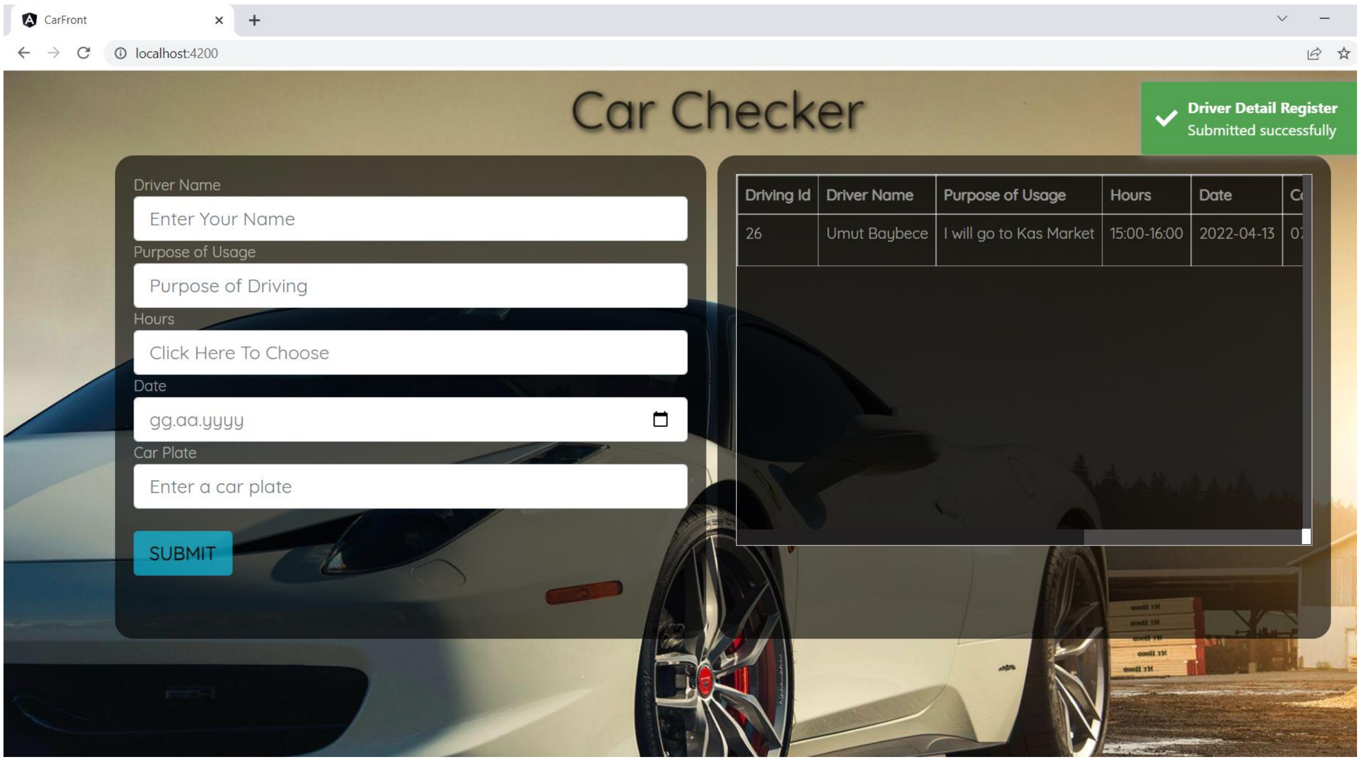
localhost:4200

Car Checker

Driving Detail Deletion
Record deleted successfully!

se of Usage	Hours	Date	Car Plate	Delete The Record	Update The Record

Figure 29: Testing delete method via "Car Checker Interface" page


CarFront

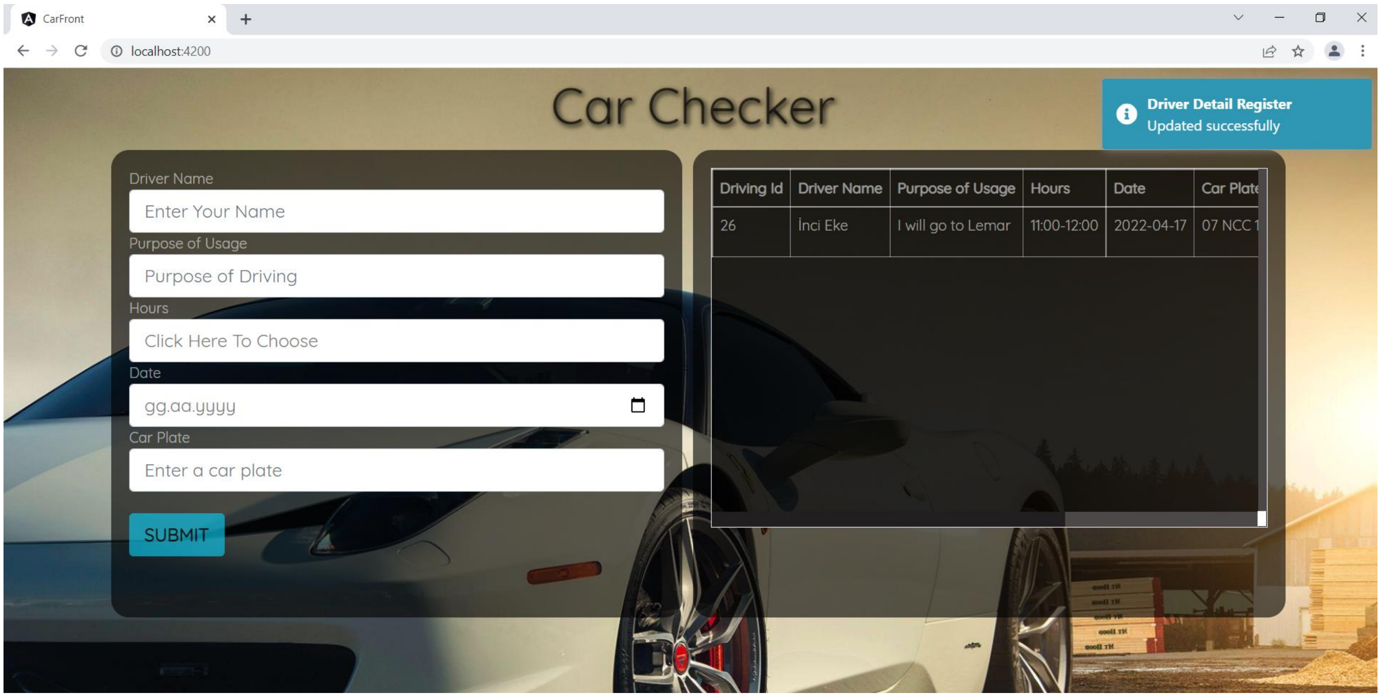
localhost:4200

Car Checker

Driver Detail Register
Submitted successfully

Driving Id	Driver Name	Purpose of Usage	Hours	Date	C
26	Umut Baybece	I will go to Kas Market	15:00-16:00	2022-04-13	0

Figure 30: Testing post method via "Car Checker Interface" page



Driving Id	Driver Name	Purpose of Usage	Hours	Date	Car Plate
26	İnci Eke	I will go to Lemar	11:00-12:00	2022-04-17	07 NCC 1

Figure 31: Testing update method via "Car Checker Interface" page

7. CONCLUSIONS

During my internship, for 20 working days, I worked on software design and software testing problems. I have developed a car tracking application to keep track of the company's vehicle usage since the company's old software was not working correctly. Also, I have created a registration page for the same application. I developed this car tracking system using ASP.NET Core framework, Microsoft SQL Server Management Studio, and Angular framework tools. I have tested my program by using Postman and open Web API tools. My car tracking application can currently store, update, get, and delete selected data. Since it allows users to enter car plates and driver names freely, it is adjustable for adding new drivers or new cars to the database later. The program also checks if input types are in the wanted structure or not to prevent possible mistakes related to the users. As a result, I have developed a fully working, simple car tracking application for my company.

Additionally, in this project, I have followed agile methodologies to keep up with possible updates of requirements during the project, and it benefited me to shorten delivery time. I also developed my analytical thinking and problem-solving skills by learning new software tools. Furthermore, this was the first time for me to be in a professional working environment, and it helped me create new networks.

Finally, I want to thank Soykan Saydam and other staff working in KOD YAZILIM ve PORJE HİZ. for offering me this excellent opportunity to improve myself by doing my internship with them.

REFERENCES

Angular. (n.d.). Angular. https://angular.io/guide/architecture

Codecademy. (2022, March 11). *What is a framework?* Codecademy News. https://www.codecademy.com/resources/blog/what-is-a-framework/

Computer Hope. (2019, June 7). *What is visual studio?* Computer Hope's Free Computer Help. https://www.computerhope.com/jargon/v/visual-studio.htm

LaptrinhX. (2019, December 30). *NetLearner on ASP .NET core 3.1*. https://laptrinhx.com/netlearner-on-asp-net-core-3-1-1572715673/

Oracle. (2019, June 17). *What is Microsoft SQL server? A definition from WhatIs.com*. SearchDataManagement. https://www.techtarget.com/searchdatamanagement/definition/SQL-Server#:~:text=Microsoft%20SQL%20Server%20is%20a,applications%20in%20corporate%20IT%20environments

Oracle. (n.d.). *What is a database?* https://www.oracle.com/database/what-is-database/

Rastogi, Anderson, Dykstra, Galloway, & Reitan. (n.d.). *Introduction to identity — ASP.NET documentation*. ASP.NET Core Documentation — ASP.NET documentation. https://jakeydocs.readthedocs.io/en/latest/security/authentication/identity.html

TechTerms. (2014, June 4). *C# definition*. The Tech Terms Computer Dictionary. https://techterms.com/definition/c_sharp