# LAB 1

OBJECTIVE**:**

To set up various Software which are required for us doing this .NET Lab session.

THEORY**:**

A lab environment is a collection of virtual and physical machines that we can use to develop and test applications.

A lab environment can contain multiple roles needed to test multi-tiered applications, such as workstations, web servers, and database servers.

In this lab, we have setup various Software such are Visual Studio Code: It is a most usable text Editor from where we have easily coding debugging

and running our code, Sql Server : Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run

either on the same computer or on another computer across a network, Postman: Simplify workflows and create better APIs – faster , a collaboration platform for API development. Git Bash: Git Bash is an application for Microsoft Windows environments which provides an emulation layer for a

Git command line experience.

GROUP DISCUSSION / PROCEDURE: We are installed required Software such are Visual Studio Code, Postman,GitBash,Sql Server etc.

CONCLUSION:

From this lab we are concluded that firstly we are installed required editors and sql server as well as required software.

# LAB 2

OBJECTIVE: To Familiar with MVC framework at .Net Framework.

THEORY: The Model-View-Controller (MVC) framework is an architectural pattern that separates an application into three main logical components Model, View, and Controller. MVC separates the business logic and presentation layer from each other. It was traditionally used for desktop graphical user interfaces (GUIs). Nowadays, MVC architecture has become popular for designing web applications as well as mobile apps.

Three important MVC the components are:

• Model: It includes all the data and its related logic

• View: Present data to the user or handles user interaction

• Controller: An interface between Model and View component

# GROUP DISCUSSION / PROCEDURE:

Model

using System;

namespace lab1\_mvc.Models

{

public class ErrorViewModel

{

public string RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

}

public class Person {

public string name {get;set;}

public string address {get;set;}

}

}

Controller:

namespace lab1\_mvc.Controllers

{

public class HomeController : Controller

{

public IActionResult Index()

{

var model = new Person();

model.name = "Joshan Pradhan";

model.address ="Ilam ";

return View(model);

}

public IActionResult Privacy()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

View:

@model lab1\_mvc.Models.Person

@{

ViewData["Title"] = "Home Page";

}

<div class="text-center">

<h1 class="display-4">Welcome</h1>

<h2>

@Model.name

</h2> <br>

<h2>

@Model.address

</h2>

</div>

Program.cs

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

# CONCLUSION:

From this lab I understood MVC framework.

# LAB 3

OBJECTIVE: To understand the basic of  building a web API with ASP.NET Core. To learn the following step by step.

We learned the basics of how to:

* Create a web API project.
* Add a model class and a database context.
* Add a controller.
* Add CRUD methods.
* Configure routing and URL paths.
* Specify return values.
* Call the web API with Postman

DICSUSSION: The Visual Studio Code instructions use the .NET Core CLI for ASP.NET Core development functions such as project creation. **.**

dotnet new webapi -o TodoApi

code -r TodoApi

These commands create a new web API project and open a new instance of Visual Studio Code in the new project folder.

CONCLUSION: In this class, I learned the basics of MVC and how to start a new web API.

# LAB 4

OBJECTIVE: To Setup required Repository for the project ApexRestaurant and to learn SQL Queries for the project ApexRestaurant.

DISCUSSION:

using System.Linq;

using Microsoft.EntityFrameworkCore;

namespace ApexRestaurant.Repository

{

    public abstract class GenericRepository<T> : IGenericRepository<T>

    where T : class, new()

    {

        protected RestaurantContext DbContext { get; set; }

        public T Get(int id)

        {

            return DbContext.Find<T>(id);

        }

        public IQueryable<T> Query()

        {

            return DbContext.Set<T>().AsQueryable();

        }

        public void Insert(T entity)

        {

            DbContext.Set<T>().Add(entity);

            DbContext.SaveChanges();

        }

        public void Update(T entity)

        {

            DbContext.Entry(entity).State = EntityState.Modified;

            DbContext.SaveChanges();

        }

        public void Delete(T entity)

        {

            DbContext.Set<T>().Remove(entity);

            DbContext.SaveChanges();

        }

    }

}

Then We created class RestaurantContext.cs:

using ApexRestaurant.Repository.Domain;

using Microsoft.EntityFrameworkCore;

namespace ApexRestaurant.Repository {

    public class RestaurantContext : DbContext {

        public RestaurantContext (DbContextOptions<RestaurantContext> options):

            base (options) { }

        public DbSet<Customer> Customers { get; set; }

        public DbSet<Meal> Meals { get; set; }

        public DbSet<MealDish> MealDishes { get; set; }

        public DbSet<Menu> Menus { get; set; }

        public DbSet<MenuItem> MenuItems { get; set; }

        public DbSet<Staff> Staffs { get; set; }

        public DbSet<StaffRole> StaffRoles { get; set; }

        protected override void OnModelCreating (ModelBuilder modelBuilder) {

            base.OnModelCreating (modelBuilder);

        }

    }

}

Then we create a folder “Domain” and add a class “Customer.cs in it.

using System;

namespace ApexRestaurant.Repository.Domain

{

    public class Customer

    {

        public int Id { get; set; }

        public string FirstName { get; set; }

        public string LastName { get; set; }

        public string Address { get; set; }

        public string PhoneRes { get; set; }

        public string PhoneMob { get; set; }

        public DateTime EnrollDate { get; set; }

        public bool IsActive { get; set; }

        public string CreatedBy { get; set; }

        public DateTime CreatedOn { get; set; }

        public string UpdatedBy { get; set; }

        public DateTime UpdatedOn { get; set; }

    }

}

Finally, we we added packages to ApexRestaurant.Repository

Dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Newtonsoft.Json

CONCLISION: From this lab I learned to commmit and push in Git. I learned about different packages required for the project.

**LAB 6**

OBJECTIVE:To Setup required Services for the project ApexRestaurant .

THEORY:The built-in container is represented by IServiceProvider implementation that supports constructor injection by default. The types (classes) managed by built-in IoC container is called services.

There are basically two types of services in ASP.NET Core:

• Framework Services: Services which are a part of ASP.NET Core framework such as IApplicationBuilder, IHostingEnvironment, ILoggerFactory etc.

• Application Services: The services (custom types or classes) which you as a programmer create for your application

# GROUP DISCUSSION / PROCEDURE:

Create an interface “IGenericService.cs”

code:

using System.Collections.Generic;

namespace ApexRestaurant.Services

{

public interface IGenericService<T>

{

IList<T> GetAll();

T GetById(int id);

void Insert(T entity);

void Update(T entity);

void Delete(T entity);

}

}

Create a class “GenericService.cs”

using System.Collections.Generic;

using System.Linq;

using ApexRestaurant.Repository;

namespace ApexRestaurant.Services

{

public abstract class GenericService<T> : IGenericService<T>

where T : class, new()

{

protected GenericService(IGenericRepository<T> entityRepository)

{

EntityRepository = entityRepository;

}

protected IGenericRepository<T> EntityRepository { get; }

public void Insert(T entity)

{

EntityRepository.Insert(entity);

}

public void Update(T entity)

{

EntityRepository.Update(entity);

}

public IList<T> GetAll()

{

return EntityRepository.Query().ToList();

}

public T GetById(int id)

{

return EntityRepository.Get(id);

}

public void Delete(T entity)

{

EntityRepository.Delete(entity);

}

}

}

Add a class “ServiceModule.cs”

using Microsoft.Extensions.DependencyInjection;

using ApexRestaurant.Services.SCustomer;

namespace ApexRestaurant.Services

{

public static class ServicesModule

{

public static void Register(IServiceCollection services)

{

services.AddTransient<ICustomerService, CustomerService> ();

services.AddTransient<IMealService, MealService> ();

services.AddTransient<IMealDishService, MealDishService> ();

services.AddTransient<IMenuService, MenuService> ();

services.AddTransient<IMenuItemService, MenuItemService> ();

services.AddTransient<IStaffService, StaffService> ();

services.AddTransient<IStaffRoleService, StaffRoleService> ();

}

}

}

Added similar interface , class for every new services. Next, add a folder “SCustomer” “Sstaff”,”SStaffRole”,”SMenu”,”SMenuItem”,”SMeal”,”SMealDish”where S stands for services. Add an interface “ICustomerService.cs” inside the folder “Scustomer” and other every new interfaces inside necessary folder.

Added a class “CustomerService.cs” and other every classes inside the folder “Scustomer”. And other required folder as per our need for a complete project.

And finally we have success fully build the project.

**CONCLUSION**: From this lab we have conclude that, the Service classes are designed to do two things:

1. Inherit from the Repository classes AND

2. Implement their own functionality, which is only necessary when said functionality deals with more than one business object.

**LAB 7**

**OBJECTIVE:** To setup the connection for API and test the api from postman.

**THEORY:**

REST is acronym for REpresentational State Transfer. It is architectural style for distributed hypermedia systems and was first presented by Roy Fielding in 2000 in his famous [dissertation](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm).

**GROUP DISCUSSION / PROCEDURE:**

**Navigate to “appsettings.json” file and add “ConnectionStrings” section:**

{

"ConnectionStrings": {

"DefaultConnection": "Server=.;Database=ApexRestaurantDb;Trusted\_Connection=True;"

},

"Logging": {

"IncludeScopes": false,

"LogLevel": {

"Default": "Warning"

}

}

}

Navigate to “Startup.cs” and add entries for RepositoryModule and ServiceModule in the

method ConfigureServices as below.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ApexRestaurant.Repository; (added)

using ApexRestaurant.Services; (added)

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.HttpsPolicy;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Options;

namespace ApexRestaurant.Api

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to

the container.

public void ConfigureServices(IServiceCollection services)

{

RepositoryModule.Register(services,

Configuration.GetConnectionString("DefaultConnection"),

GetType().Assembly.FullName);

ServicesModule.Register(services);

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

// This method gets called by the runtime. Use this method to configure the

HTTP request pipeline.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseMvc();

}

}

}

# Under “Controllers” folder, added“CustomerController.cs”

using ApexRestaurant.Repository.Domain;

using ApexRestaurant.Services.SCustomer;

using Microsoft.AspNetCore.Mvc;

namespace ApexRestaurant.Api.Controller

{

[Route("api/customer")]

public class CustomerController : ControllerBase

{

private readonly ICustomerService \_customerService;

public CustomerController(ICustomerService customerService)

{

\_customerService = customerService;

}

[HttpGet]

[Route("{id}")]

public IActionResult Get([FromRoute] int id)

{

var customer = \_customerService.GetById(id);

if (customer == null)

return NotFound();

return Ok(customer);

}

[HttpGet]

[Route("")]

public IActionResult GetAll()

{

var customers = \_customerService.GetAll();

return Ok(customers);

}

[HttpPost]

[Route("")]

public IActionResult Post([FromBody] Customer model)

{

\_customerService.Insert(model);

return Ok();

}

[HttpPut]

[Route("")]

public IActionResult Put([FromBody] Customer model)

{

\_customerService.Update(model);

return Ok();

}

[HttpDelete]

[Route("")]

public IActionResult Delete([FromBody] Customer model)

{

\_customerService.Delete(model);

return Ok();

}

}

}

**Added necessary dependencies to the project.**

cd ApexRestaurant.Api

dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.Abstractions

dotnet add package Microsoft.EntityFrameworkCore.Analyzers

dotnet add package Microsoft.EntityFrameworkCore.Relational

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.Extensions.Caching.Abstractions

dotnet add package Microsoft.Extensions.Caching.Memory

dotnet add package Microsoft.Extensions.Configuration

dotnet add package Microsoft.Extensions.Configuration.Abstractions

dotnet add package Microsoft.Extensions.Configuration.Binder

dotnet add package Microsoft.Extensions.DependencyInjection

dotnet add package Microsoft.Extensions.DependencyInjection.Abstractions

dotnet add package Microsoft.Extensions.Logging

dotnet add package Microsoft.Extensions.Logging.Abstractions

dotnet add package Microsoft.Extensions.Options

# LAB 8 & 9

# OBJECTIVE:

ApexRestaurant.Api Setup for Web Services/Web API & Testing via Postman.

# DISCUSSION:

NavigateD to “appsettings.json” file and added “ConnectionStrings” section:

{

  "ConnectionStrings": {

  "DefaultConnection": "Server=DESKTOP-L0CP8VR\\SQLEXPRESS;Database=ApexRestaurantDb;Trusted\_Connection=True;"

  },

  "Logging": {

  "IncludeScopes": false,

  "LogLevel": {

  "Default": "Warning"

  }

  }

 }

Then added the following dependencies to the project:

cd ApexRestaurant.Api

dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.Abstractions

dotnet add package Microsoft.EntityFrameworkCore.Analyzers

dotnet add package Microsoft.EntityFrameworkCore.Relational

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.Extensions.Caching.Abstractions

dotnet add package Microsoft.Extensions.Caching.Memory

dotnet add package Microsoft.Extensions.Configuration

dotnet add package Microsoft.Extensions.Configuration.Abstractions

dotnet add package Microsoft.Extensions.Configuration.Binder

dotnet add package Microsoft.Extensions.DependencyInjection

dotnet add package Microsoft.Extensions.DependencyInjection.Abstractions

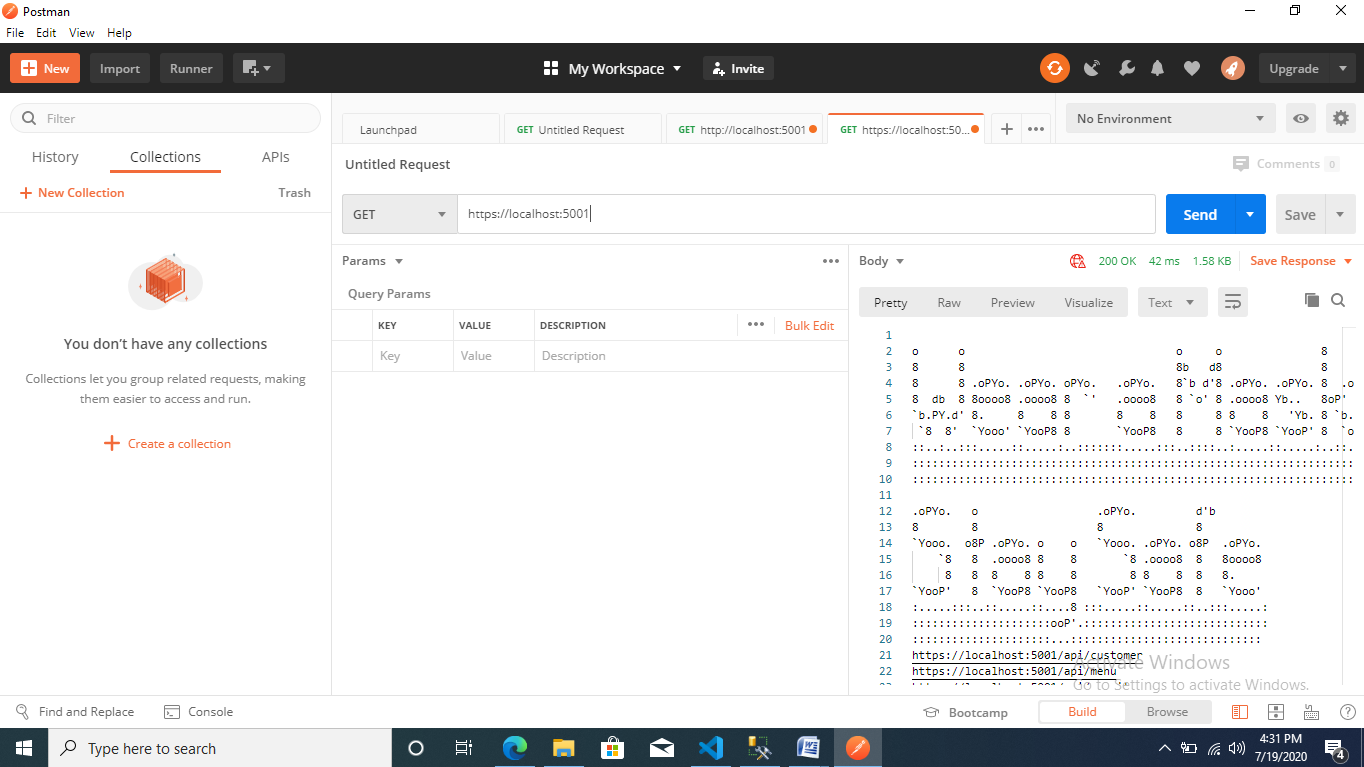
dotnet add package Microsoft.Extensions.Logging

dotnet add package Microsoft.Extensions.Logging.Abstractions

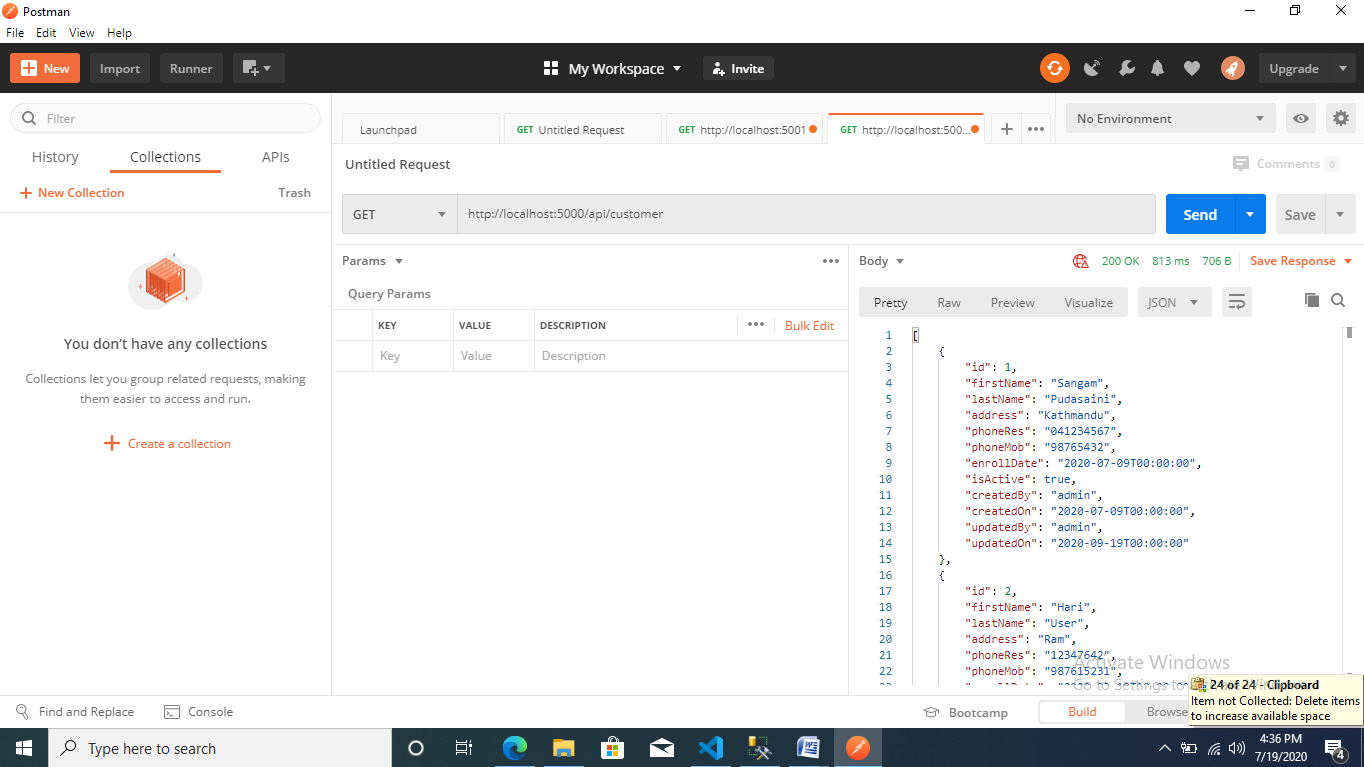
dotnet add package Microsoft.Extensions.Options

Then Navigated to http://localhost:5000/api/customer/ in my browser for verification. Then I perform the following in Postman for detailed verification.

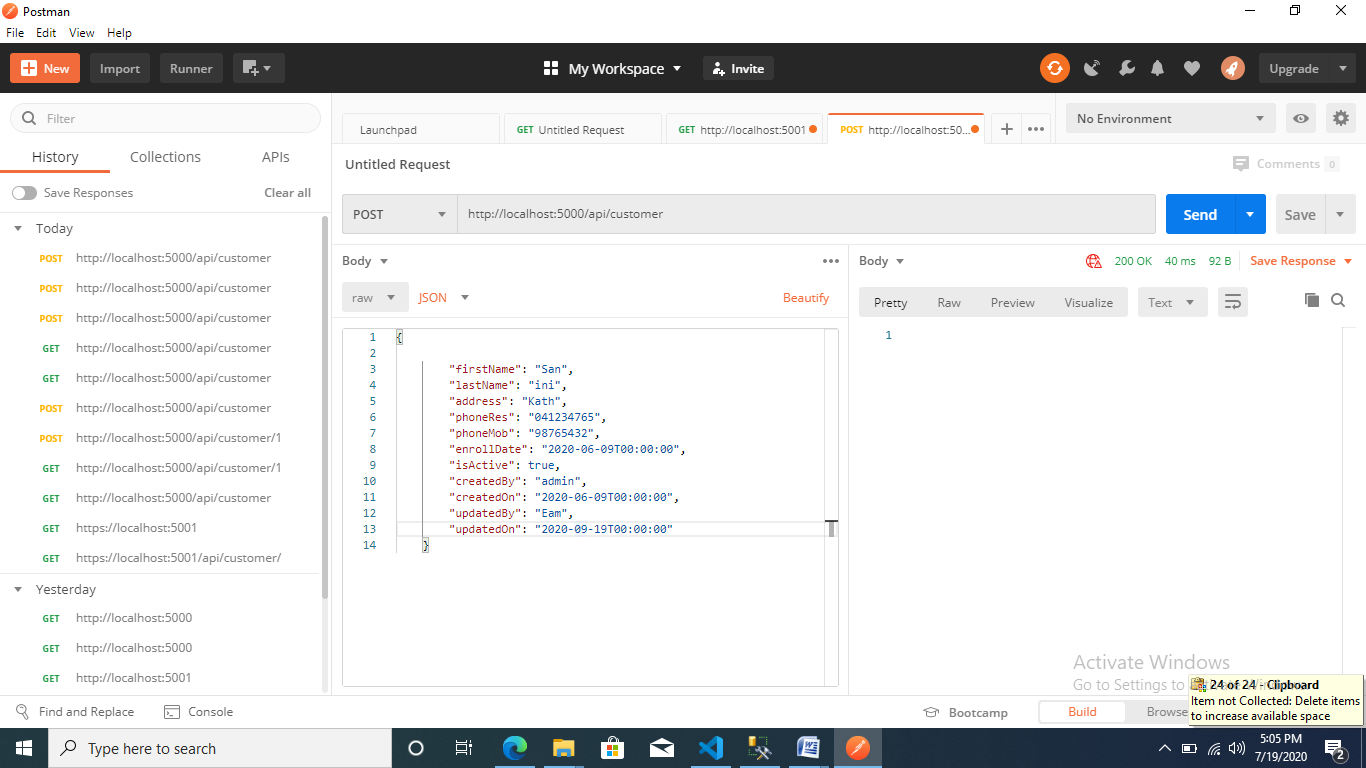
GET: <http://localhost:5000>



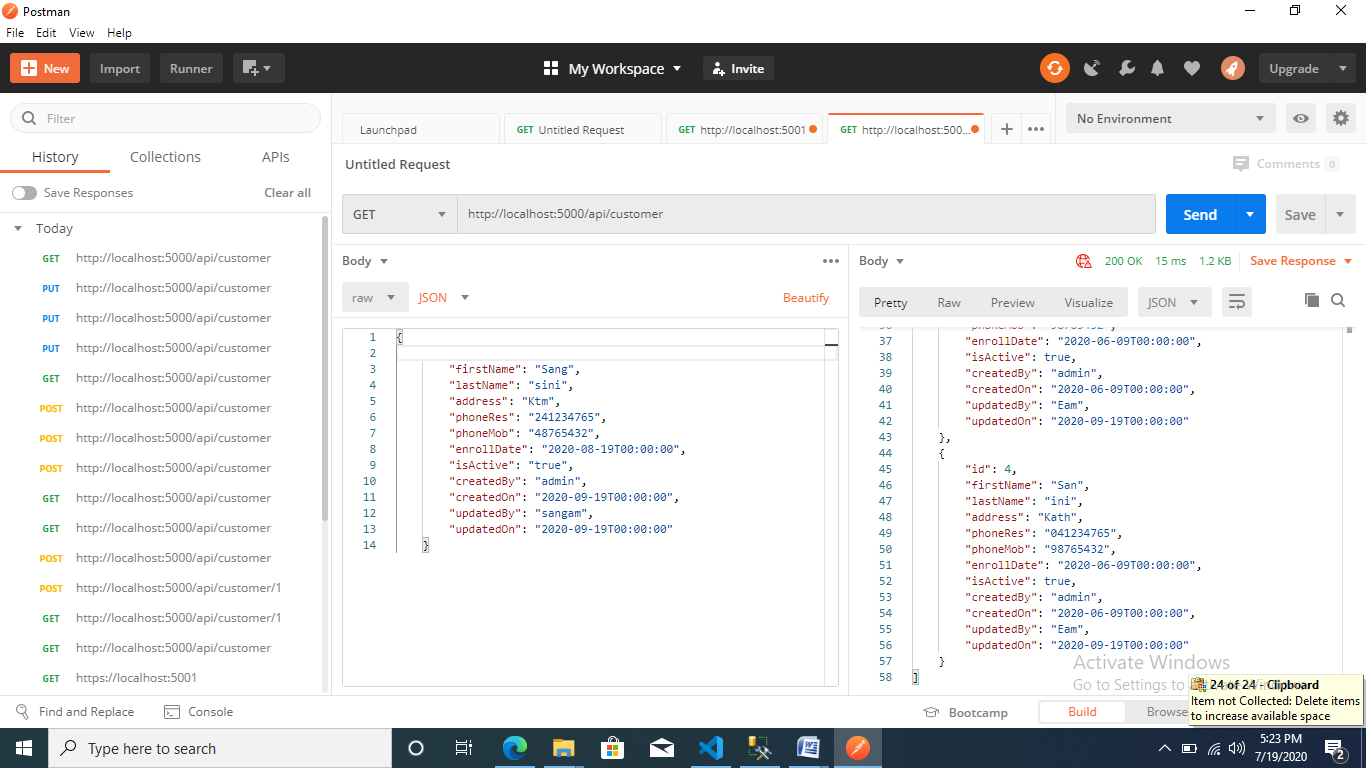
**GET:** [**http://localhost:5000/api/customer**](http://localhost:5000/api/customer)



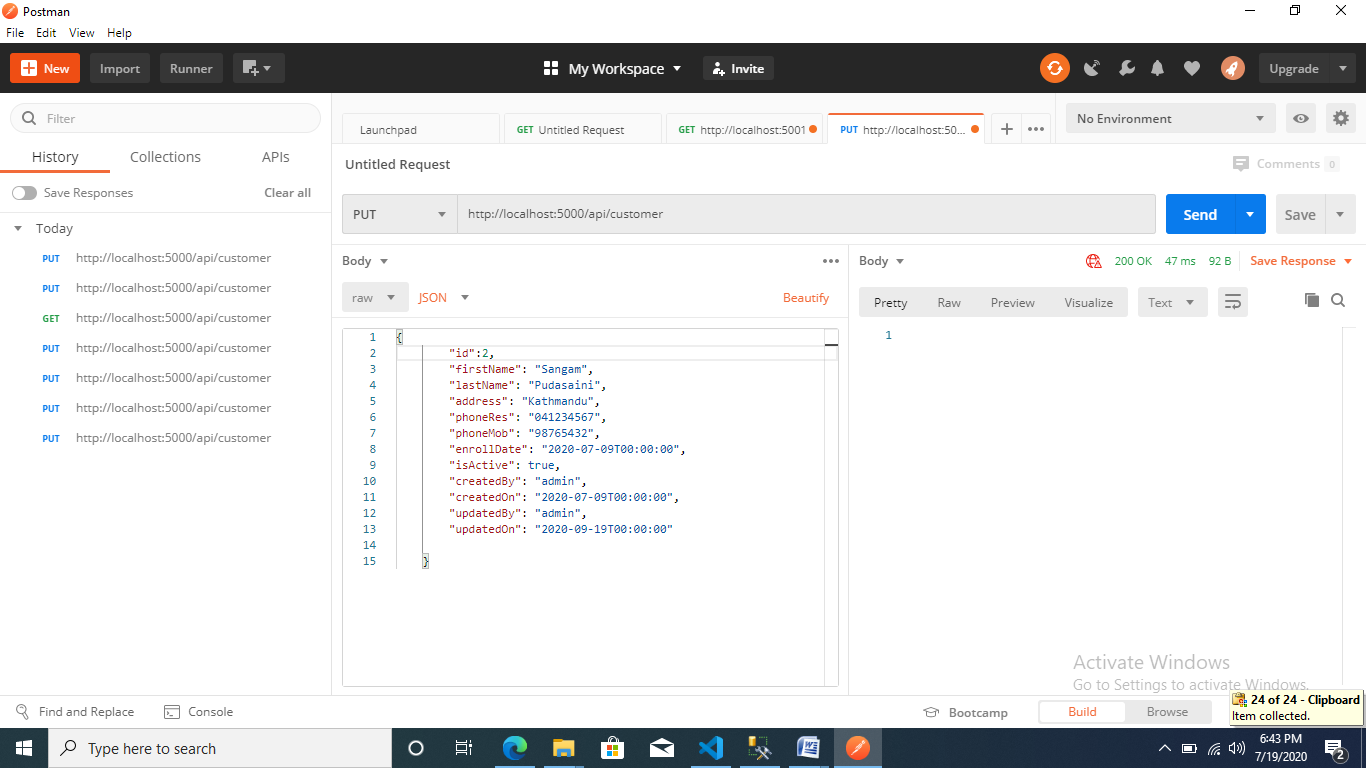
POST: <http://localhost:5000/api/customer/>



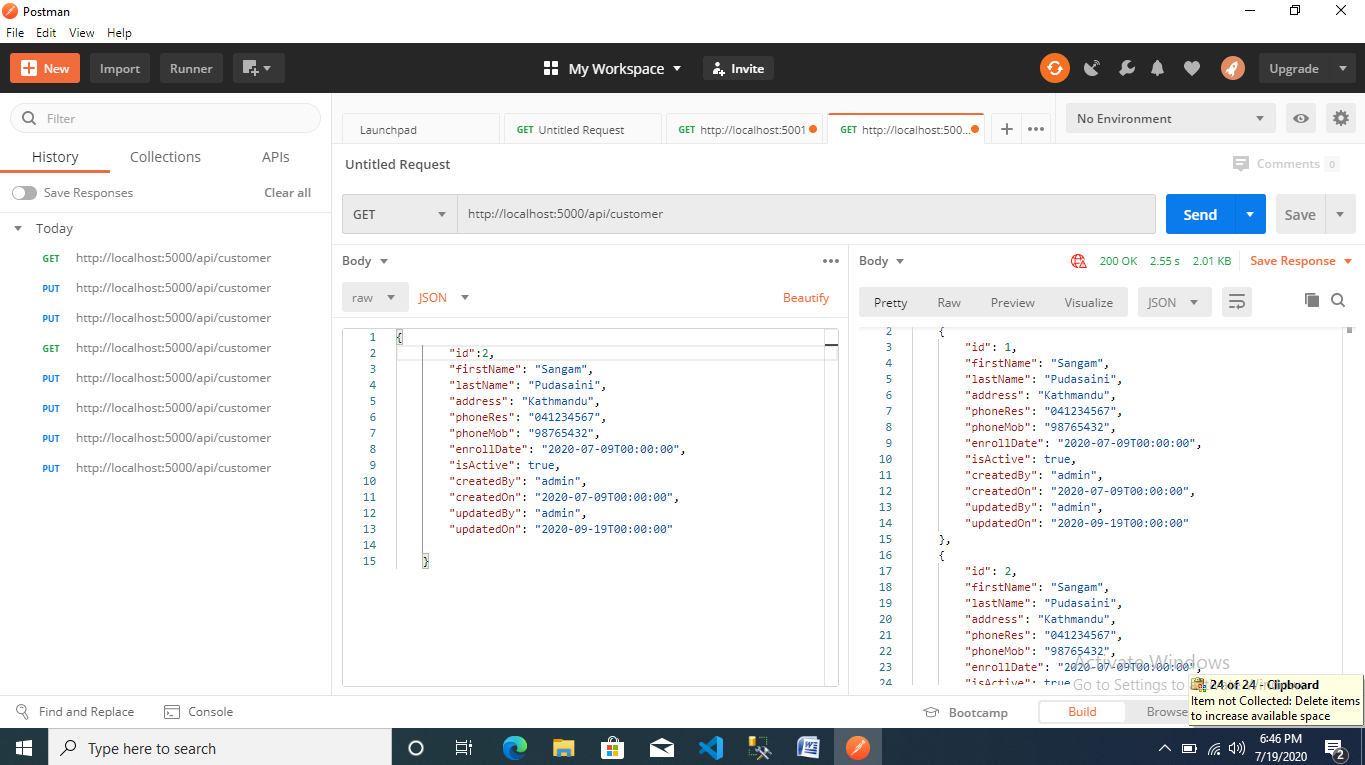
After POST



PUT: <http://localhost:5000/api/customer/>

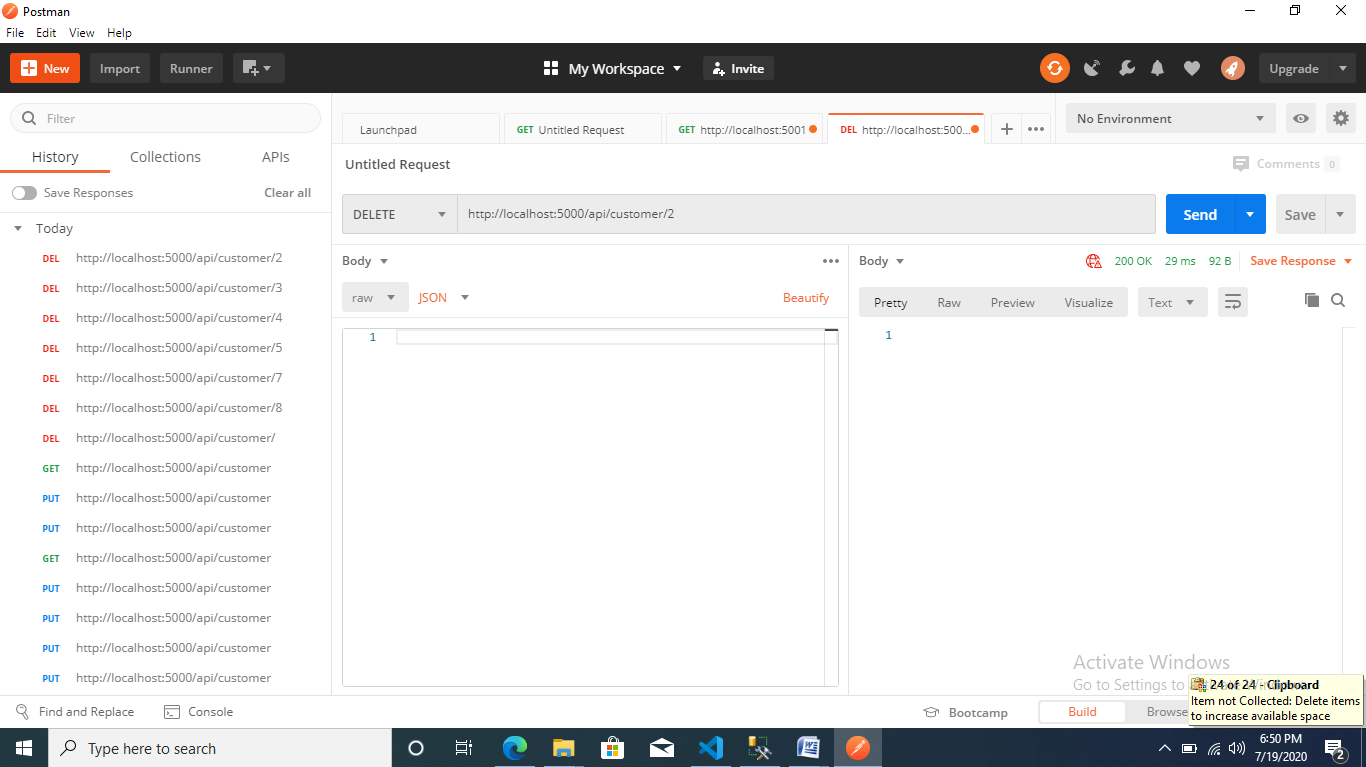


After Updated from PUT method

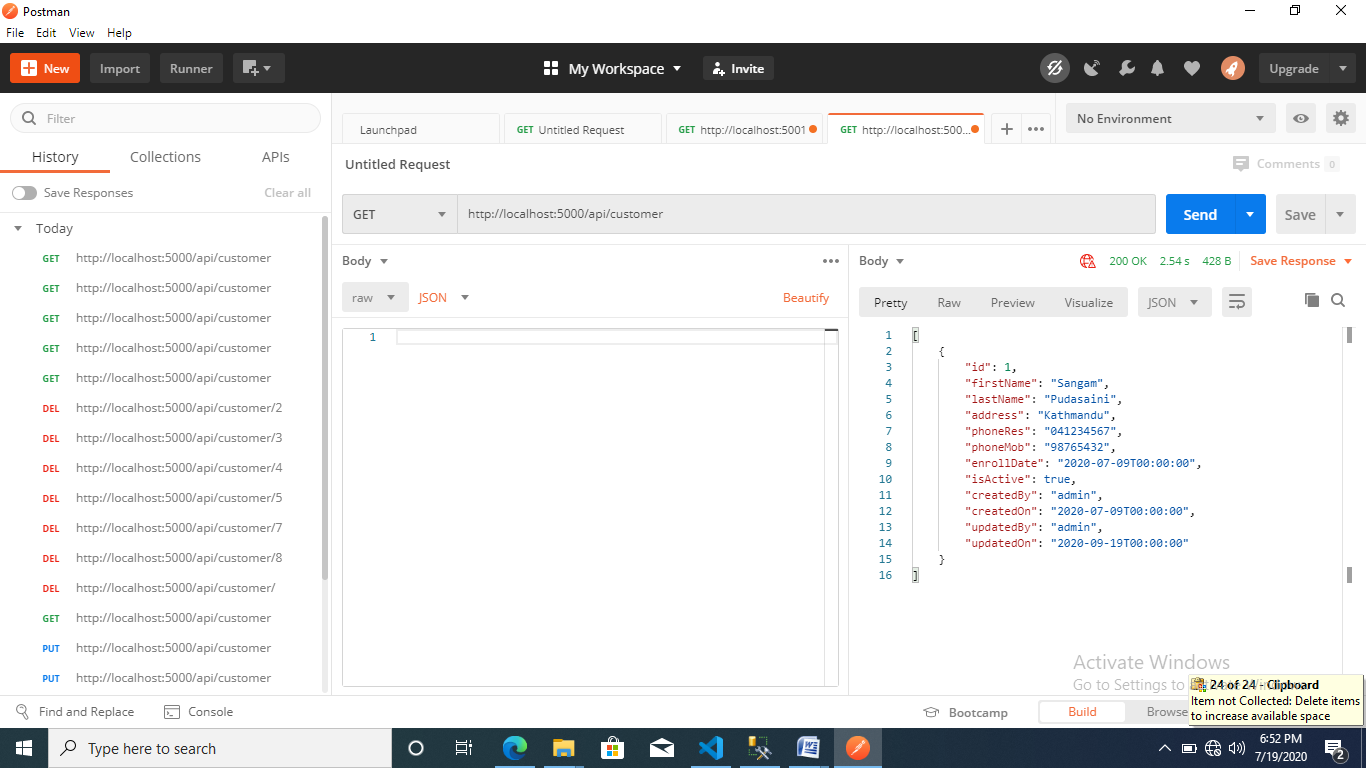


Below Showing

DELETE: <http://localhost:5000/api/customer/2>



Below Showing after DELETE: <http://localhost:5000/api/customer/2>



CONCLUSION: Thus from this Lab I learned to

* Quickly and easily send REST, SOAP, and GraphQL requests directly within Postman.
* Learned to stay up-to-date on the health of my API by checking performance and response times at scheduled intervals
* Learned about a shared context for building and consuming APIs, and collaborate in real-time with built-in version control.