Pokhara University

.NET Technology

LAB MANUAL

Compiled by:

Er. Bishwo Adhikari

Asst. Professor

Apex College

January 2020

LAB PLAN

Guidelines for Lab

Language to be used

Labs will be conducted in C#. The first few labs will be introductory to Visual Studio Code and ASP.NET Core MVC using C#. Microsoft SQL Server will be used as back-end. Please note that .NET core can run on multiple operating system like Windows, Linux, Mac OS.

Lab Procedure:

The lab plan will be provided to the students during the start of semester. Students are required to study the related chapters from the book or research relevant topic from the internet and prepare a pre-report before coming to the lab. The lab instructor will make a presentation explaining the concepts, through the use of one or several typical program. In the lab, students will modify the program as instructed.

Pre Requisite:

a. Basic programming (Conditionals, Loops, HTML, JavaScript, CSS, MVC, DB, etc.)
b. Visual Studio Code
c. C# for Visual Studio Code (latest version)
d. .NET Core 3.1 SDK or later
e. SQL Server 2017 Express Edition,
f. SQL Operations Studio

The Visual Studio Code instructions use the .NET Core CLI for ASP.NET Core development functions such as project creation. You can follow these instructions on any platform (macOS, Linux, or Windows) and with any code editor.

To check if you have already installed .NET Core, type the following in command/ terminal: `dotnet --version`

Buffer Labs:

In addition to accommodate for labs that could not be conducted, buffer labs can be used to repeat past labs, or complete labs that were not completed in the assigned session. However, at the end of the buffer lab, the student should have submitted all lab reports until that point.

Lab 1: Introduction to Visual Studio Code and ASP.NET Core MVC
Lab 2: An Introduction to Web API - Part I
Lab 3: An Introduction to Web API - Part II
Lab 4: ER Diagram, Create Database Tables and N-Layered Architecture
Lab 5: Buffer Lab
Lab 6: ApexRestaurant.Repository Setup
Lab 7: ApexRestaurant.Services Setup
Lab 8: ApexRestaurant.Api Setup for Web Services/Web API
Lab 9: ApexRestaurant.Api Testing via Postman

Lab 10: Presentation / Viva

Bonus Lab I: ApexRestaurant.Mvc Setup

Bonus Lab II: Mobile Application (ApexRestaurant.Mobile) Setup

Setup Environment

- Open the integrated terminal.
- Change directories (`cd`) to the folder that will contain the project folder.
- Run the following commands:

  .NET Core CLI

  ```
  dotnet new webapi -o TodoApi
  cd TodoApi
  dotnet add package Microsoft.EntityFrameworkCore.SqlServer
  dotnet add package Microsoft.EntityFrameworkCore.InMemory
  code -r ../TodoApi
  ```

- When a dialog box asks if you want to add required assets to the project, select **Yes**.

  The preceding commands:

  - Creates a new web API project and opens it in Visual Studio Code.
  - Adds the NuGet packages which are required in the next section.

Lab 2:

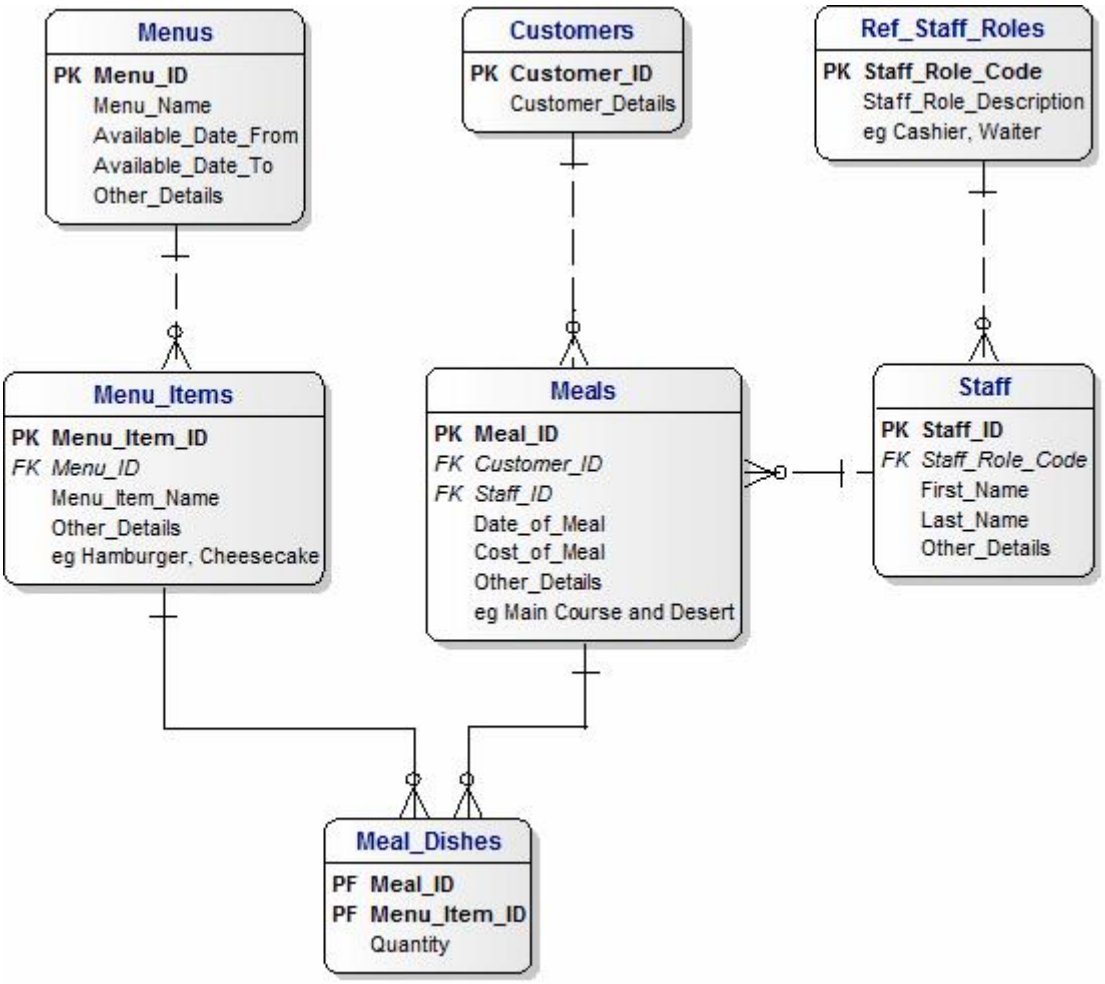
Create a Web API Project

- Follow this document and create REST API for your project.
- https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-2.2&tabs=visual-studio-code

Lab 3:

ER Diagram, Create Database Tables and N-Layered Architecture

Below is the ER Diagram for Restaurant and Customers.



Run the following code in SQL Server:

```
-- Create database

CREATE DATABASE ApexRestaurantDB;

-- Use db
USE ApexRestaurantDB;

-- Create table
CREATE TABLE Customers(
	Id INT PRIMARY KEY IDENTITY(1,1),
	FirstName NVARCHAR(200),
	LastName NVARCHAR(200),
	Address NVARCHAR(500),
	PhoneRes  NVARCHAR(50),
	PhoneMob NVARCHAR(50),
	EnrollDate DATETIME,
	IsActive BIT,
```

```
	CreatedBy NVARCHAR(200),
	CreatedOn DATETIME,
	UpdatedBy NVARCHAR(200),
	UpdatedOn DATETIME
);
```

Then insert a seed data to the table Customers. This is done to ensure that you get at least one record to test your application.

```
INSERT INTO Customers
           (FirstName
           ,LastName
           ,Address
           ,PhoneRes
           ,PhoneMob
           ,EnrollDate
           ,IsActive
           ,CreatedBy
           ,CreatedOn
           ,UpdatedBy
           ,UpdatedOn)
VALUES
(
	'Ram Bahadur',
	'Thapa',
	'Kathmandu',
	'0123456789',
	'9876543210',
	'2019-01-01 00:00:00.000',
	1,
	'sysuser',
	'2019-01-01 10:00:00.000',
	'sysuser',
	'2019-01-01 11:00:00.000'
);
```

Follow the same process for all the tables in your database.

Initializing Project Structure:

Create a folder D:\ApexRestaurant\ and initialize projects:

```
dotnet new mvc -o ApexRestaurant.Mvc --auth Individual¶
```

```
dotnet new webapi -o ApexRestaurant.Api¶
```

```
dotnet new classlib -o ApexRestaurant.Services¶
```

```
dotnet new classlib -o ApexRestaurant.Repository¶
```

Create a solution:

```
dotnet new sln -n ApexRestaurant¶
```

Add the above projects to this newly created solution:

```
dotnet sln ApexRestaurant.sln
add .\ApexRestaurant.Repository\ApexRestaurant.Repository.csproj¶
```

```
dotnet sln ApexRestaurant.sln
add .\ApexRestaurant.Services\ApexRestaurant.Services.csproj¶
```

```
dotnet sln ApexRestaurant.sln
add .\ApexRestaurant.Api\ApexRestaurant.Api.csproj¶
```

```
dotnet sln ApexRestaurant.sln
add .\ApexRestaurant.Mvc\ApexRestaurant.Mvc.csproj¶
```

Add reference of projects:

```
dotnet add .\ApexRestaurant.Services
reference .\ApexRestaurant.Repository\ApexRestaurant.Repository.csproj¶
```

```
dotnet add .\ApexRestaurant.Api
reference .\ApexRestaurant.Services\ApexRestaurant.Services.csproj¶
```

```
dotnet add .\ApexRestaurant.Api
reference .\ApexRestaurant.Repository\ApexRestaurant.Repository.csproj¶
```

Restore the dependencies and build:

```
dotnet restore ApexRestaurant.sln¶
```

```
dotnet build ApexRestaurant.sln¶
```

Verify that the API project is running:

```
dotnet run --project .\ApexRestaurant.Api\ApexRestaurant.Api.csproj¶
```

Ensure that the Controllers folder has ValuesController.cs file, then run the following URL in your preferred web browser:

```
https://localhost:5001/api/values
```

If the above returns output similar to below, your project is running!

Since you're in a working state, initialize and put the project to git.

Go to github.com and create a new **public repository**, e.g. yourname_dotnet. You need to submit a fully working instance of your github repository for project marks. I have created a public repository as apexrestaurant. The public url of my github repository is "https://github.com/bishwo633/apexrestaurant".

Initialize git inside the folder "D:\ApexRestaurant\"

```
git init¶
```

Add all changed files to be tracked by git

```
git add .¶
```

Commit with message "initial commit"

```
git commit -m "initial commit"¶
```

Setup your github url as your remote git location

```
git remote add origin
https://github.com/bishwo633/apexrestaurant.git¶
```

Push to github

```
git push -u origin master¶
```

Go to your github location in browser and see the changes reflected online.

ApexRestaurant.Repository Setup

Create an interface IGenericRepository.cs:

```
using System.Linq;

namespace ApexRestaurant.Repository
{
    public interface IGenericRepository<T>
    {

        T Get(int id);

        IQueryable<T> Query();

        void Insert(T entity);

        void Update(T entity);

        void Delete(T entity);

    }
}
```

Create a class GenericRepository.cs

```
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();
        }
    }
}
```

Create a class RepositoryModule.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using ApexRestaurant.Repository.RCustomer;

namespace ApexRestaurant.Repository
{
    public static class RepositoryModule
    {
        public static void Register(IServiceCollection services, string connection,
string migrationsAssembly)
        {
```

```
            services.AddDbContext<RestaurantContext>(options =>
options.UseSqlServer(connection, builder =>
builder.MigrationsAssembly(migrationsAssembly)));

            services.AddTransient<ICustomerRepository, CustomerRepository>();

        }

    }
}
```

Note:

- Whenever you add a new repository, add a line similar to the highlighted line.

- Transient objects are always different; a new instance is provided to every controller and every service. Scoped objects are the same within a request, but different across different requests. Singleton objects are the same for every object and every request.

Create a class RestaurantContext.cs:

```
using Microsoft.EntityFrameworkCore;

using ApexRestaurant.Repository.Domain;


namespace ApexRestaurant.Repository

{

    public class RestaurantContext : DbContext

    {


        public RestaurantContext(DbContextOptions<RestaurantContext> options) :
base(options)

        {

        }


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


        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            base.OnModelCreating(modelBuilder);

        }


    }

}
```

Note:

- Whenever you add a new table, add a line similar to the highlighted line.

Create a folder “Domain” and add a class “Customer.cs”.

```
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; }


    }
}
```

Create a folder “RCustomer” where R stands for Repository.

Create an interface “ICustomerRepository.cs” inside “RCustomer” folder

```
using ApexRestaurant.Repository.Domain;

```

```
namespace ApexRestaurant.Repository.RCustomer
{
    public interface ICustomerRepository : IGenericRepository<Customer>
    {
    }
}
```

Create a class “CustomerRepository.cs” insider “RCustomer” folder

```
using ApexRestaurant.Repository.Domain;


namespace ApexRestaurant.Repository.RCustomer
{
    public class CustomerRepository : GenericRepository<Customer>, ICustomerRepository
    {
        public CustomerRepository(RestaurantContext dbContext)
        {
            DbContext = dbContext;
        }
    }
}
```

To add related packages:

```
cd ApexRestaurant.Repository¶
dotnet add package Microsoft.EntityFrameworkCore¶
dotnet add package Microsoft.EntityFrameworkCore.SqlServer¶
dotnet add package Newtonsoft.Json¶
```

Good work! Your repository project is complete. Now build your project to ensure there are no errors.

```
dotnet build¶
```

Then add the changes to git.

```
git add .¶
git commit -m "added repository for customer"¶
git push -u origin master¶
```

Follow the similar steps for other database models as per your ER diagram.

Lab 5:

Buffer Lab

Lab 6:

ApexRestaurant.Services Setup

Create an interface “IGenericService.cs”

```
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>();
        }
```

```
    }
}
```

Note: Add similar line to the highlighted line for every new services.

Next, add a folder “SCustomer” where S stands for services.

Add an interface “ICustomerService.cs” inside the folder “SCustomer”.

```
using ApexRestaurant.Repository.Domain;


namespace ApexRestaurant.Services.SCustomer
{
    public interface ICustomerService : IGenericService<Customer>
    {


    }
}
```

Add a class “CustomerService.cs” inside the folder “SCustomer”.

```
using ApexRestaurant.Repository.Domain;
using ApexRestaurant.Repository.RCustomer;


namespace ApexRestaurant.Services.SCustomer
{
    public class CustomerService : GenericService<Customer>, ICustomerService
    {
        public CustomerService(ICustomerRepository customerRepository) :
base(customerRepository)
        {


        }
    }
}
```

Great work! Services project is setup. Build to ensure there are no erros.

```
cd ApexRestaurant.Services¶

dotnet build¶
```

Then add the changes to git.

```
git add .¶

git commit -m "added services for customer"¶
```

```
git push -u origin master¶
```

Follow the similar steps for other models as per your ER diagram.

Lab 7:

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

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;
using ApexRestaurant.Services;
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, add “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();
        }

    }
}
```

Add 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¶
```

```
dotnet add package Microsoft.Extensions.Primitives¶
```

Build the project to ensure there are no errors.

```
dotnet build¶
```

Then add the changes to git.

```
git add .¶

git commit -m "added api controller for customer"¶

git push -u origin master¶
```

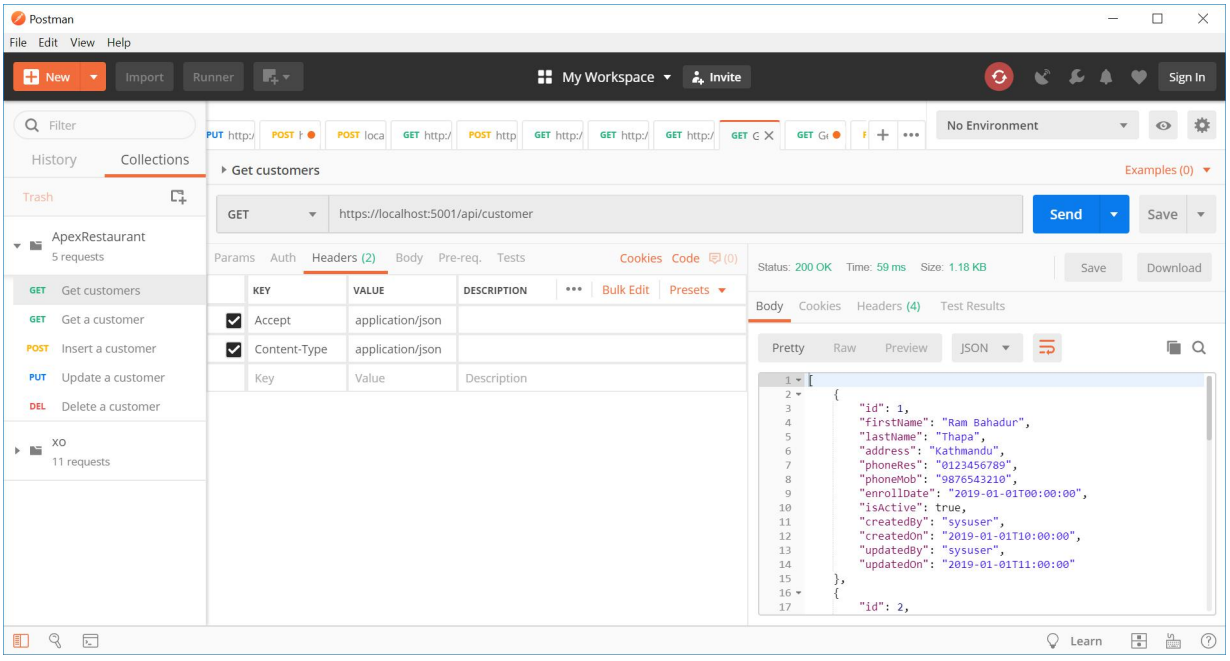
Run and check that API is working correctly.

```
dotnet run
```

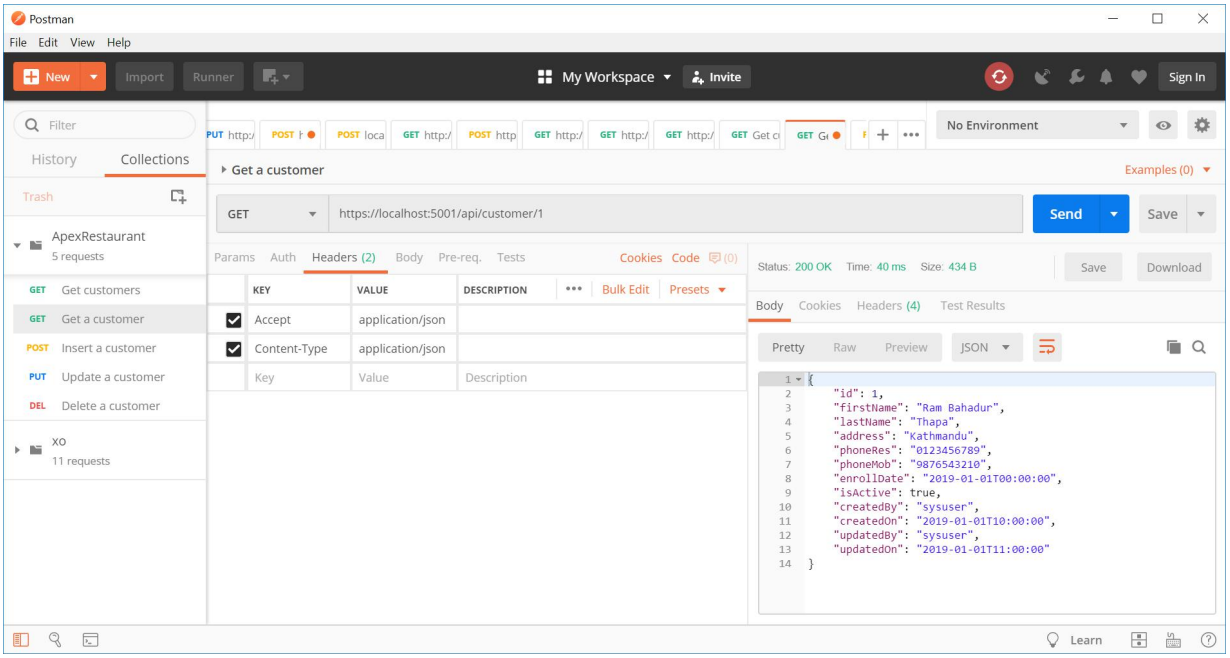
Navigate to http://localhost:5000/api/customer/ in your browser for verification.

Perform the following in Postman for detailed verification.

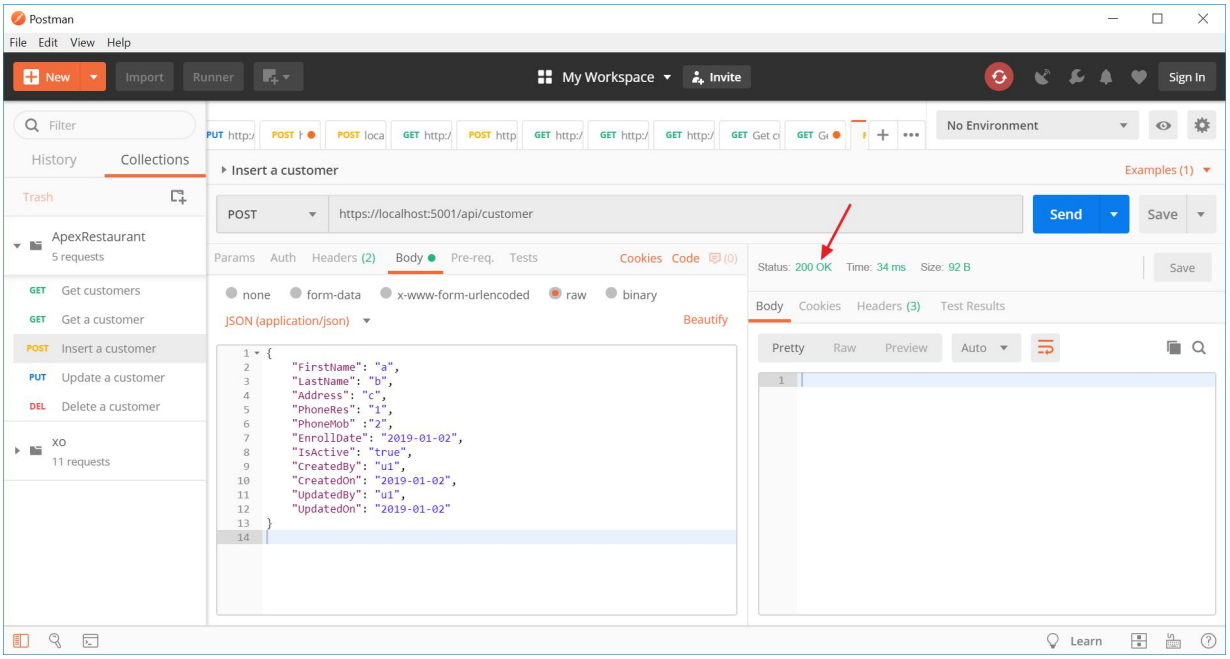
GET: http://localhost:5000/api/customer/



GET: http://localhost:5000/api/customer/1/

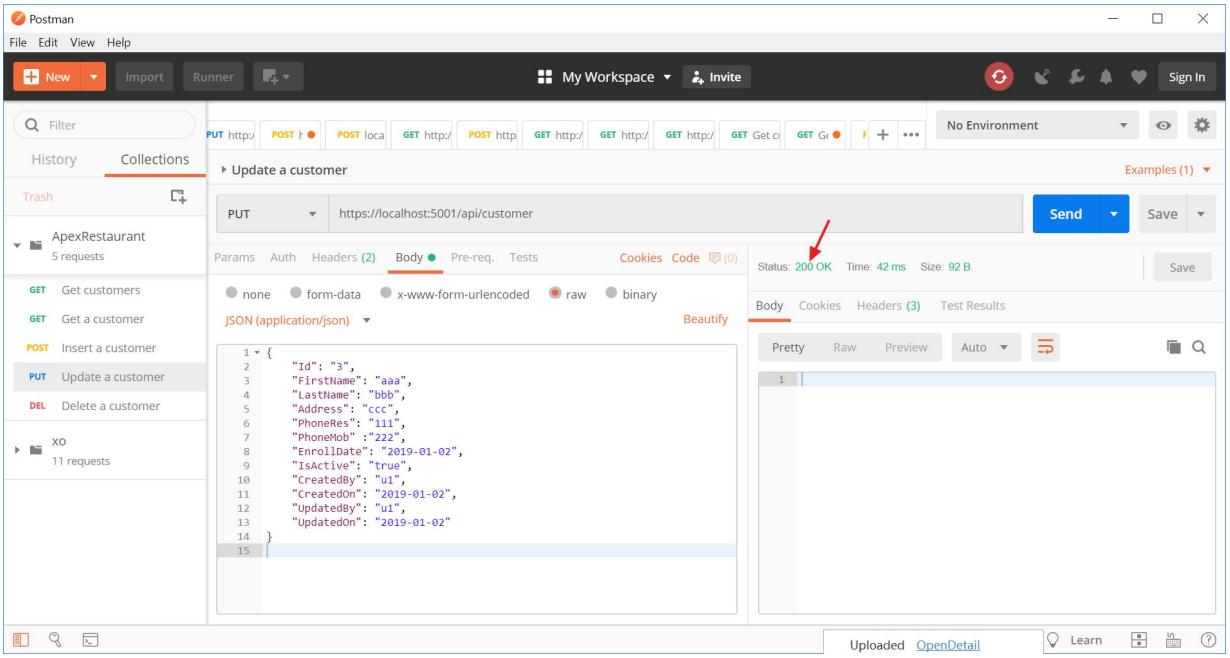


POST: http://localhost:5000/api/customer/

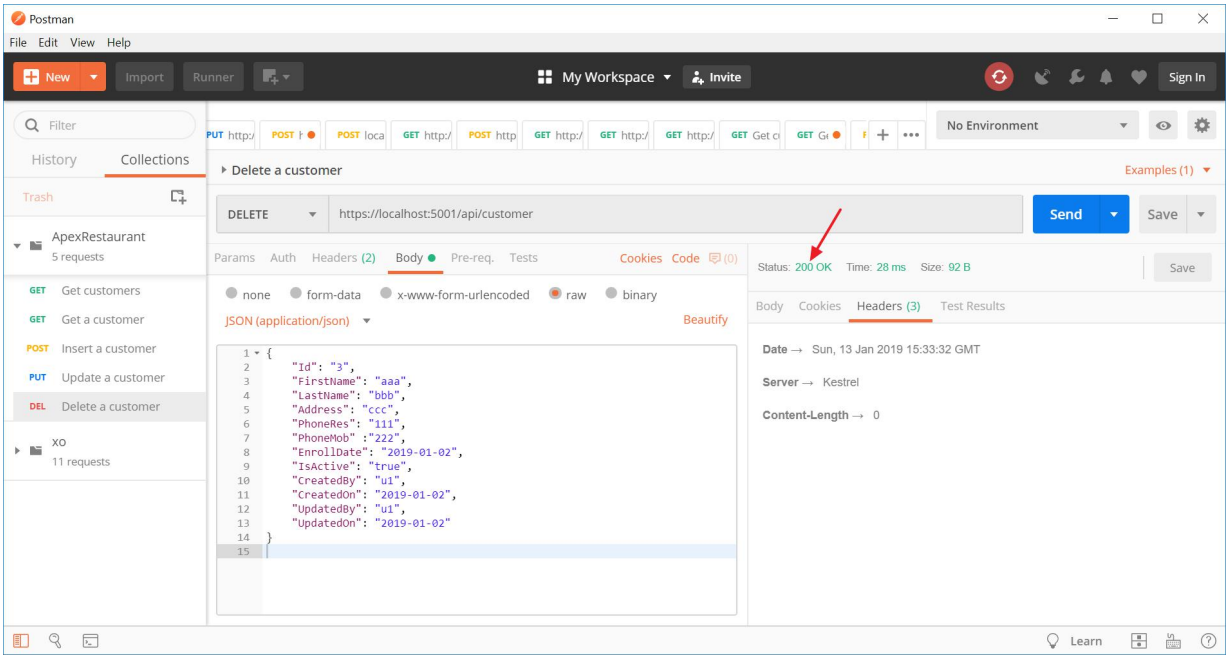


Notice the field “Status: 200 OK” for successful insertion of customer.

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



DELETE: http://localhost:5000/api/customer/



Now the API is ready to be used by applications.

Lab 8&9:

Call Web API with jQuery/Ajax

https://www.codeofaninja.com/2015/06/php-crud-with-ajax-and-oop.html

.NET Technology

Lab 10:

Exam

Send an email with the link to your public github repository to bishwomitra.adhikari@apexcollege.edu.np.

Based on your commit history and correctness of working modules, your evaluation will be done.

Bonus Lab I:

ApexRestaurant.Mvc Setup

Let's use CLI code generator "dotnet-aspnet-codegenerator" to generate controllers and views for us.

First, install dotnet-aspnet-codegenerator.

```
dotnet tool install --global dotnet-aspnet-codegenerator ¶
```

Then create a controller and view for customer.

Go to root folder of MVC project

```
cd ApexRestaurant\ApexRestaurant.Mvc ¶
```

Generate code:

```
dotnet-aspnet-codegenerator -p "ApexRestaurant.Mvc.csproj" controller -name
CustomersController -m Customer -outDir Controllers ¶
```

Similar to ApexRestaurant.Api, call Services inside ActionResult of Controllers and proceed ahead.

Bonus Lab II:

Mobile Application (ApexRestaurant.Mobile) Setup

Use the Web API with Ionic 4, Angular 7 and Cordova.

Follow: https://www.djamware.com/post/5b87894280aca74669894414/angular-6-httpclient-consume-restful-api-example

Note: Make use of your own Web API in this project.

References:

- https://medium.com/@kmignaz/create-your-forecast-application-with-net-core-and-dockerize-it-945329e0fa83
- https://code-maze.com/net-core-web-api-ef-core-code-first/
- https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-3.1&tabs=visual-studio-code
- https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-javascript?view=aspnetcore-3.1
- https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-3.1&tabs=visual-studio-code
- https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-3.1

--- ☺ ---

BEST OF LUCK