**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 components

**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 we understand MVC and how to use MVC .

**LAB 3**

**OBJECTIVE:**To create an MVC project and setup the different models required for restaurant such as Customer, Meal, Meal Dish, Staff etc and WebApi Project using .net Framework.

**THEORY:** ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful applications on the . NET Framework.The Web API is an open source and it can be hosted in the application or on the IIS .The request may be GET, POST, DELETE or PUT.

There are many differences between MVC and Web API, including:

* We can use the MVC for developing the Web application that replies as both data and views but the Web API is used for generating the HTTP services that replies only as data.
* In the Web API the request performs tracing with the actions depending on the HTTP services but the MVC request performs tracing with the action name.
* The Web API returns the data in various formats, such as JSON, XML and other format based on the accept header of the request. But the MVC returns the data in the JSON format by using JSONResult.
* The Web API supports content negotiation, self hosting. All these are not supported by the MVC.

**GROUP DISCUSSION / PROCEDURE :**

Run the sql code

-- 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

CreatedOn

UpdatedBy

UpdatedOn

NVARCHAR(200),

DATETIME,

NVARCHAR(200),

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'

);

Note: Similarly create database tables for others models as well and insert some dummy data to make sure data is inserted in tables.

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

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

dotnet new web api -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

**CONCLUSION:** From this lab we are creating one project with properties like MVC, Services,Repository and Api.

**LAB 4**

**OBJECTIVE**:To Setup required Repository for the project ApexRestaurant .

**THEORY:** The repository layer isolates Business layer from the Data Access Layer. The Repository contains Data Mapper entity. This entity can be used as a model entity for providing schema of the data for performing CRUD operations, by using the CRUD operations defined in the repository Repository pattern C# is a way to implement data access by encapsulating the set of objects persisted in a data store and the operations performed over them, providing a more object-oriented view of the persistence layer.

**GROUP DISCUSSION / PROCEDURE:**

ApexRestaurant\ and initialize projects:

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();

}

}

}

WhenEver we are follows same rules and some line are changed as per our requirement in different classes.

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> ();

services.AddTransient<IMealRepository, MealRepository> ();

services.AddTransient<IMealDishRepository, MealDishRepository> ();

services.AddTransient<IMenuRepository, MenuRepository> ();

services.AddTransient<IMenuItemRepository, MenuItemRepository> ();

services.AddTransient<IStaffRepository, StaffRepository> ();

services.AddTransient<IStaffRoleRepository, StaffRoleRepository> ();

}

}

}

After Create class RepositoryModule we have add different table as per our requirement in our projects.

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; }

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);

}

}

}

Create a folder “Domain” and add a class “Customer.cs”,“Staff.cs”,”StaffRoles”,”Menu.cs”,”MenuItem.cs”,”Meal.cs”and” MealDish.cs” etc

Customers.cs

Code:

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” and other folder which are necessary for our project, where R stands for Repository.

Code :

using ApexRestaurant.Repository.Domain;

namespace ApexRestaurant.Repository.RCustomer

{

public interface ICustomerRepository : IGenericRepository<Customer>

{

}

Create an interface “ICustomerRepository.cs” inside “RCustomer” folder and create other Interface as follow ICustomerRepository .

using ApexRestaurant.Repository.Domain;

namespace ApexRestaurant.Repository.RCustomer

{

public class CustomerRepository : GenericRepository<Customer>, ICustomerRepository

{

public CustomerRepository(RestaurantContext dbContext)

{

DbContext = dbContext;

}

}

}

**At Finally,**

To add related packages:

cd ApexRestaurant.Repository

dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Newtonsoft.Json

And finally we have successfully build the project.

**CONCLUSION:** From this lab we have conclude that,Repository Pattern separates the data access logic and maps it to the entities in the business logic. It works with the domain entities and performs data access logic. In the Repository pattern, the domain entities, the data access logic and the business logic talk to each other using interfaces

**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> ();

}

}

}

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

Add 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.

Add 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;

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 ¶

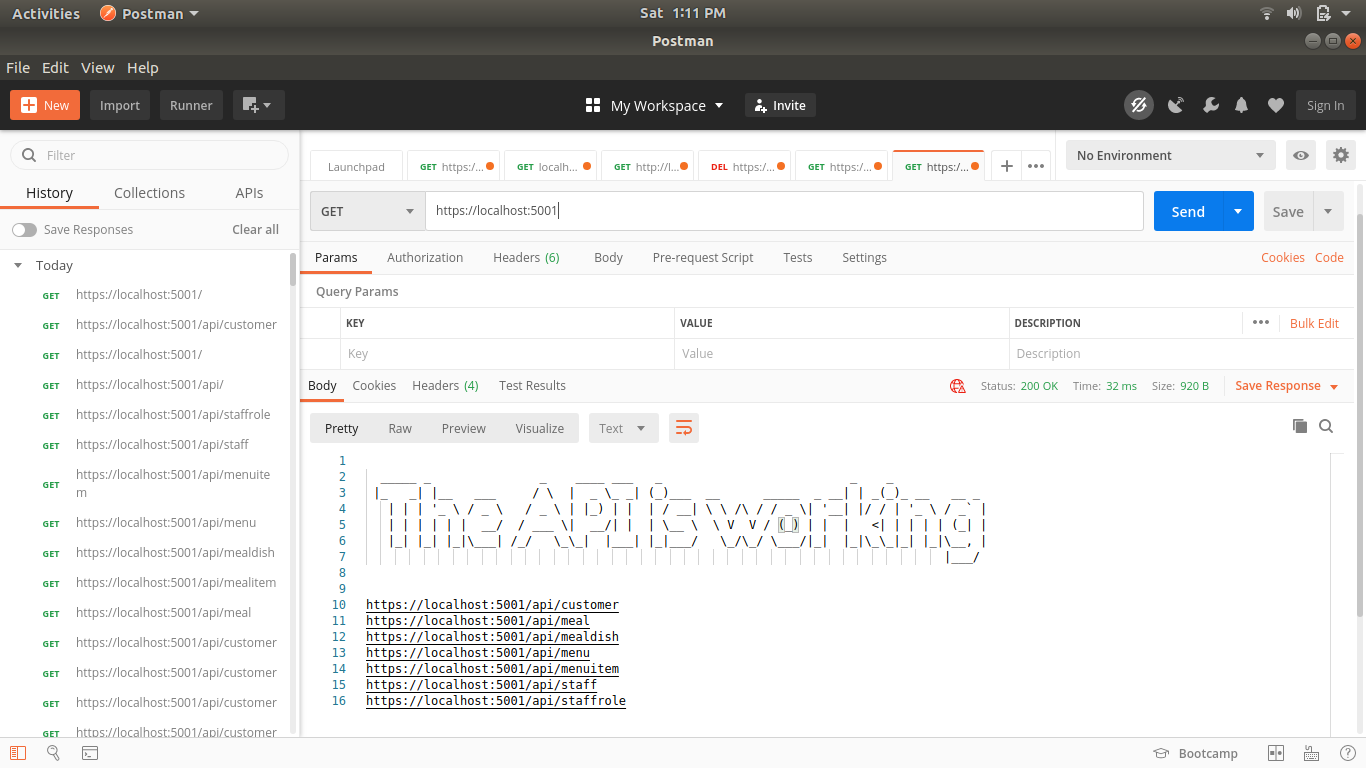
dotnet add package Microsoft.Extensions.Logging ¶

dotnet add package Microsoft.Extensions.Logging.Abstractions ¶

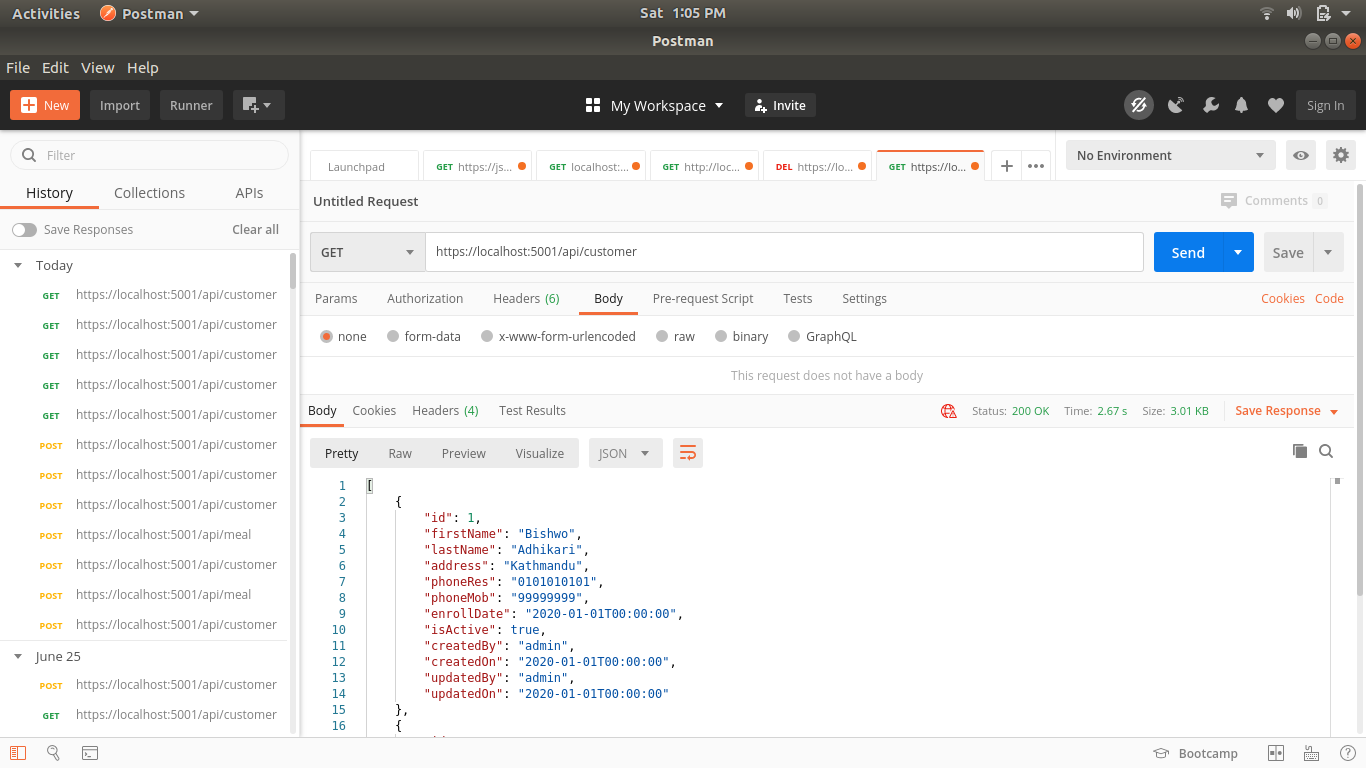
dotnet add package Microsoft.Extensions.Options ¶

**OUTPUT**

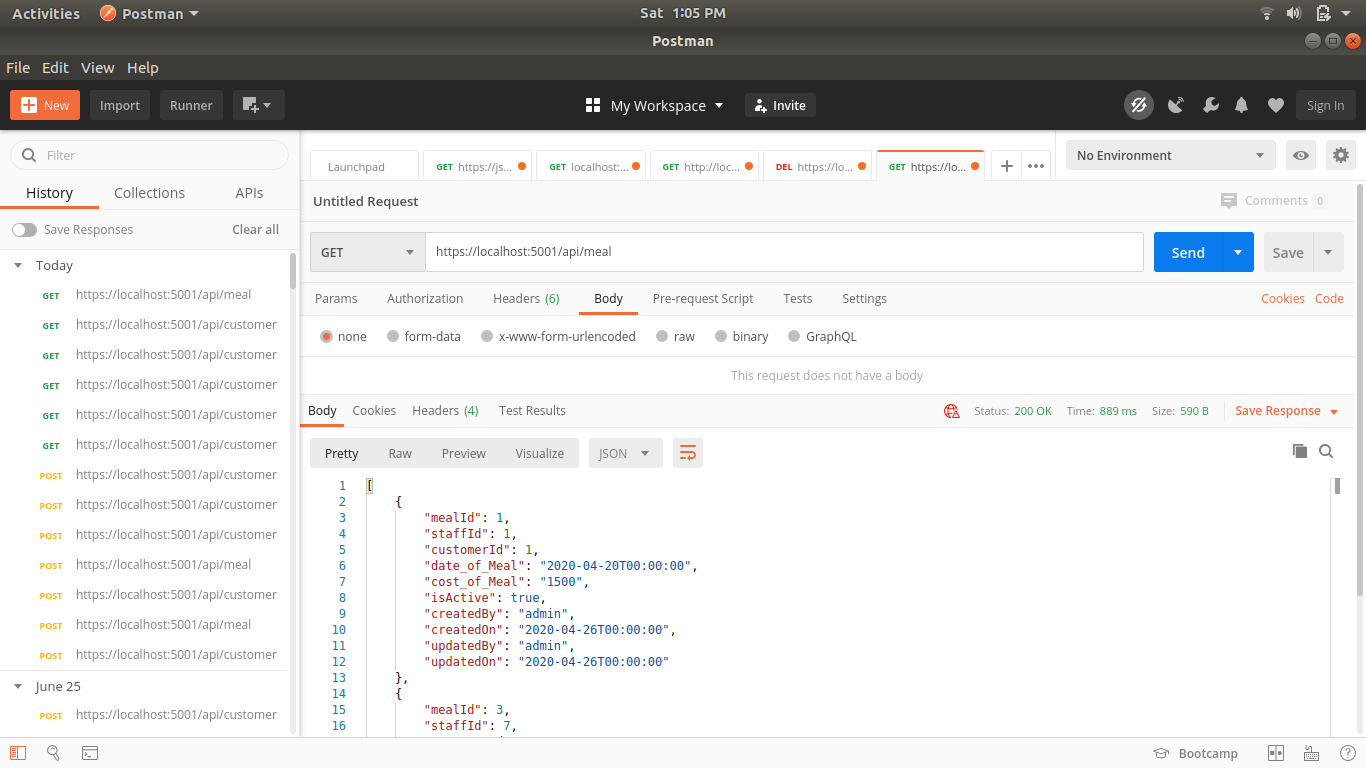
**GET:** [**http://localhost:5000/**](http://localhost:5000/api/customer)

****

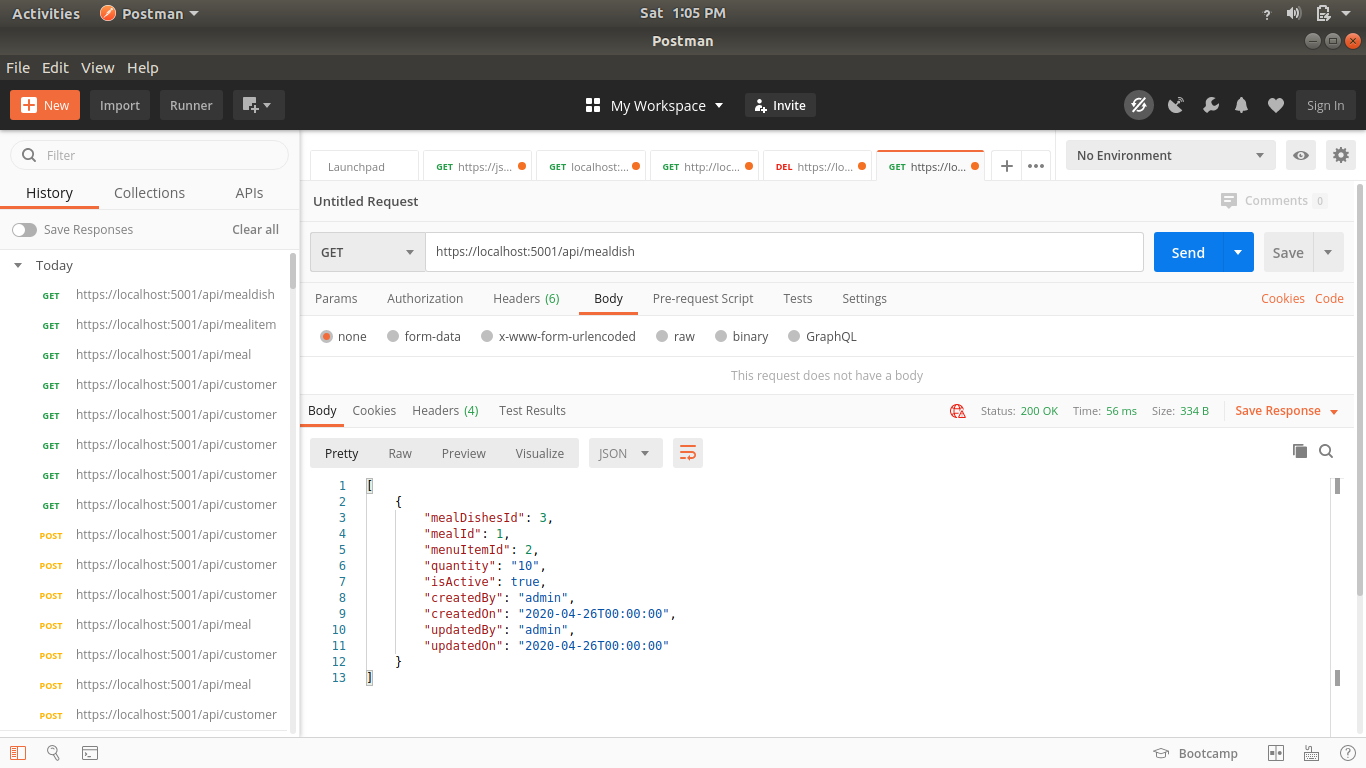
**GET:** [**http://localhost:5000/api/customer**](http://localhost:5000/api/customer)

****

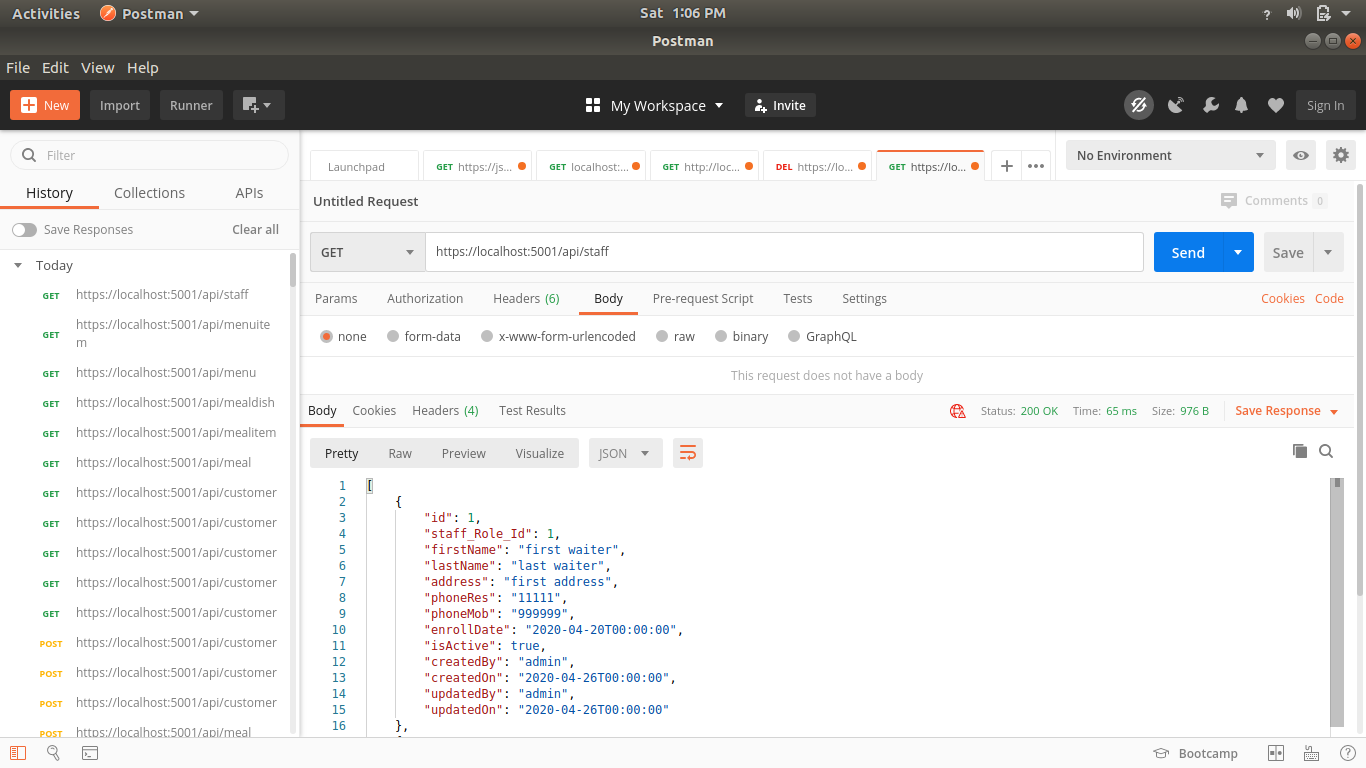
**GET:** [**http://localhost:5000/api/**](http://localhost:5000/api/customer)**meal**

****

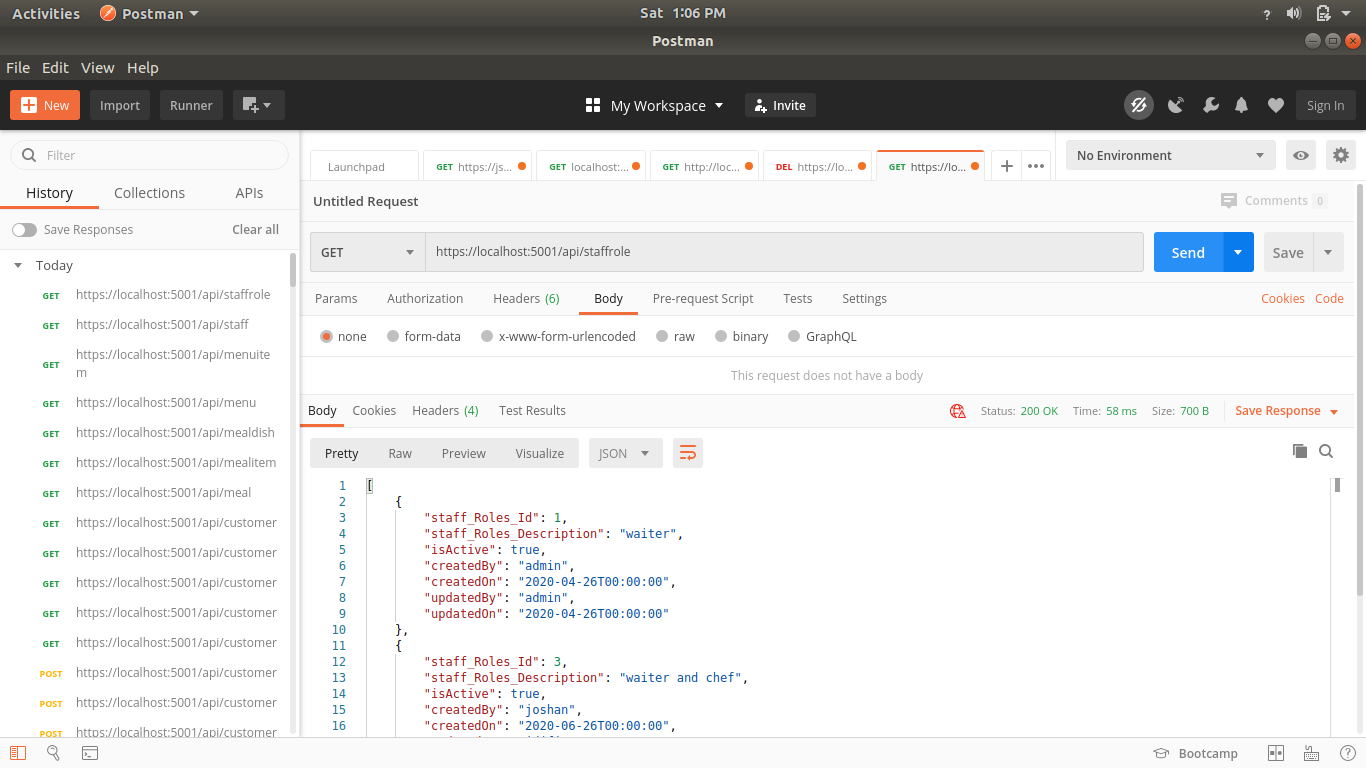
**GET:** [**http://localhost:5000/api/**](http://localhost:5000/api/customer)**mealdish**

****

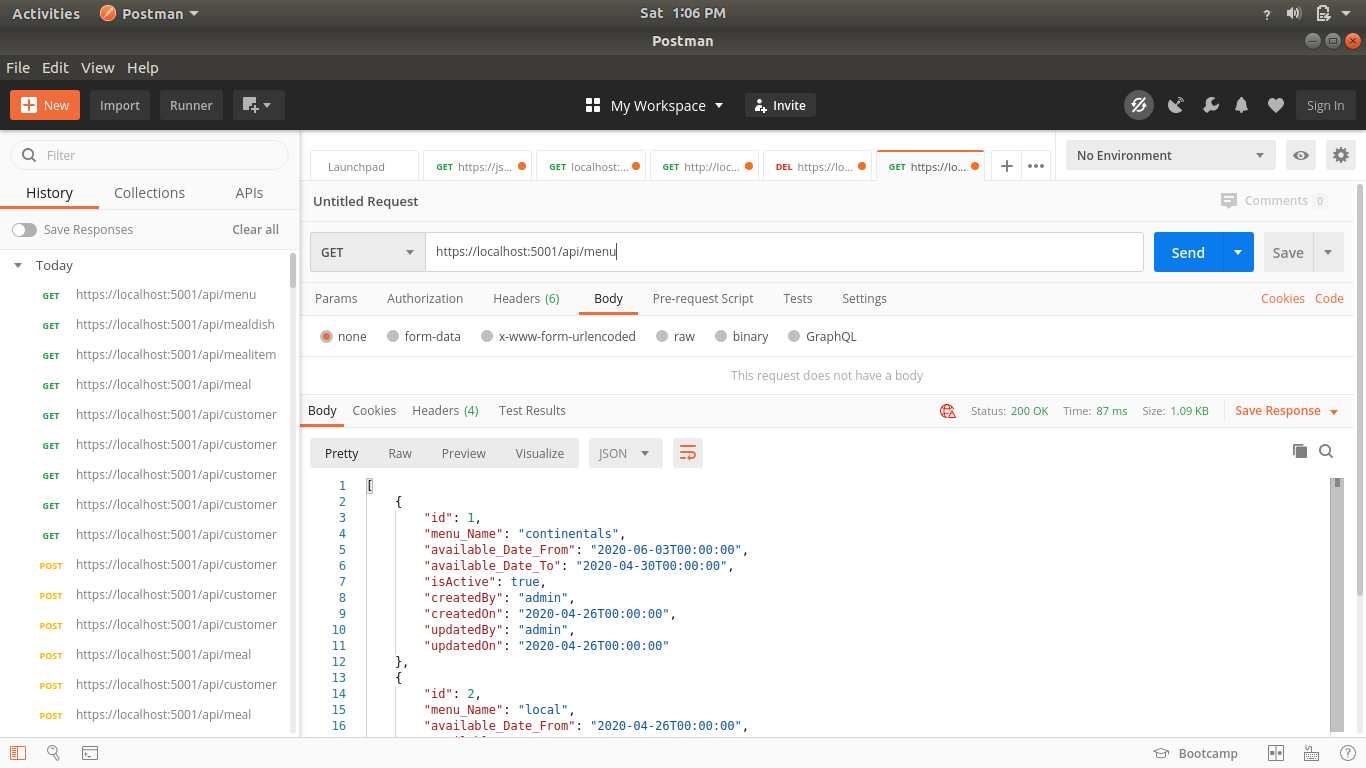
**GET:** [**http://localhost:5000/api/**](http://localhost:5000/api/customer)**staff**

****

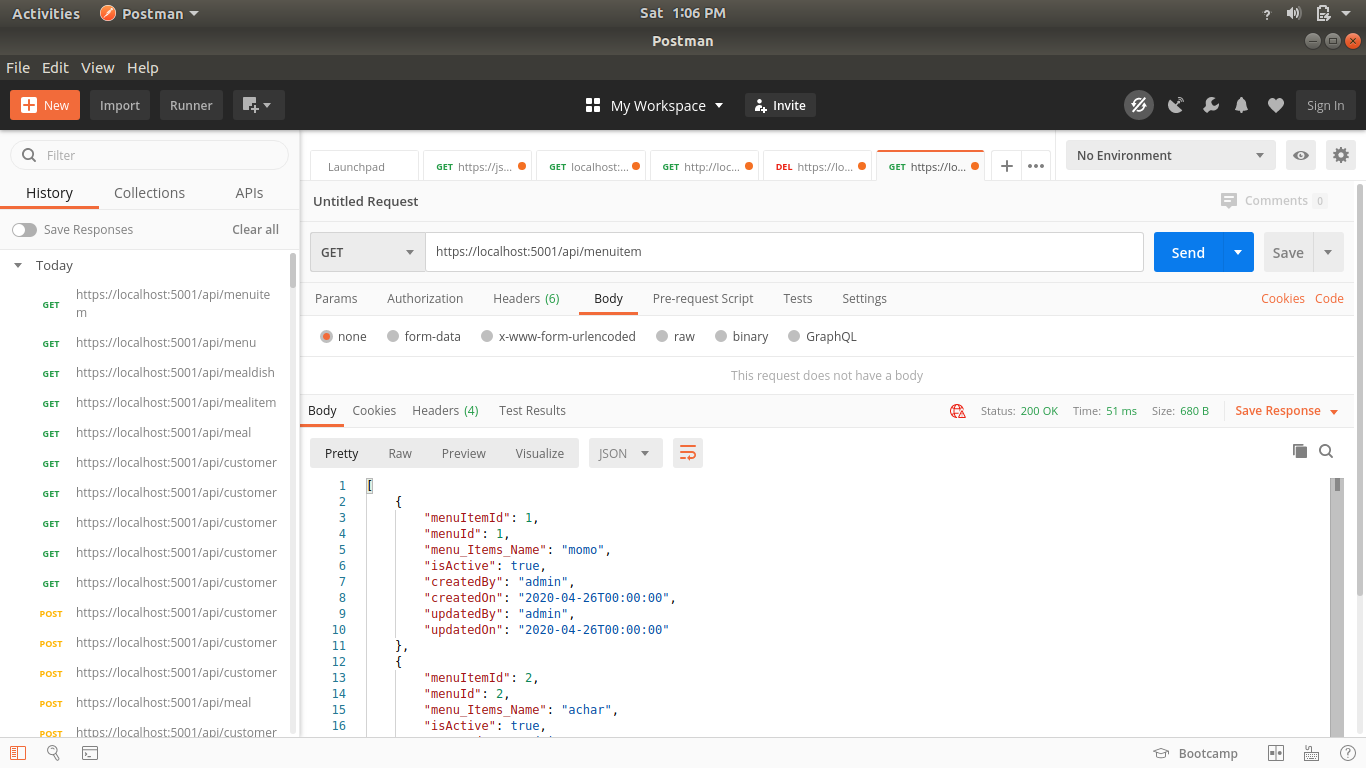
**GET:** [**http://localhost:5000/api/**](http://localhost:5000/api/customer)**staffrole**

****

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

****

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

****

**CONCLUSION:** On this lab we have learned to config the connection and use the postman to test the API. Learning different request methods and status code with specific messages.

**LAB 8**

**OBJECTIVE:** Learn the react native to build the UI(User Interface) for mobile application.

**THEORY**: React Native is a JavaScript framework for writing real, natively rendering mobile applications for iOS and Android. It’s based on React, Facebook’s JavaScript library for building user interfaces, but instead of targeting the browser, it targets mobile platforms. In other words: web developers can now write mobile applications that look and feel truly “native,” all from the comfort of a JavaScript library that we already know and love. Plus, because most of the code you write can be shared between platforms, React Native makes it easy to simultaneously develop for both Android and iOS.

**CONCLUSION**: With the learning to build mobile app I learned to integrate the api on mobile app using react native. For github link of react native user interface <https://github.com/joshanpradhan/ApexRestaurantReactNativeUI>