BIRLA VISHVAKARMA MAHAVIDYALAYA

ENGINEERING COLLEGE

[An Autonomous Institution]

A
Project Report
On

Tiffin Service Web Application

Under the course of

DESIGN ENGINEERING -3CP08

B. E., Semester – VI

(Computer Engineering)

Submitted by:

Sr.	Name of student	Enrolment No.
1	Zarana Solanki	180070107054
2	Kavya Jani	180070107018

Faculty Guide

Prof. Kirti J Sharma

Prof. Mosin I Hasan

Academic year

(2020-2021)

CERTIFICATE

This is to certify that the students namely, **Ms. Zarana Solanki (180070107054), Ms. Kavya Jani (180070107018)** of ***B. E. (Computer Engineering) Semester VI*** have successfully completed the course work and related tasks for the course of **Design Engineering 3CP08** during the academic term ending in the month of May 2021.

Date: __________

Place: __________

Prof. Kirti J Sharma Head of the Department

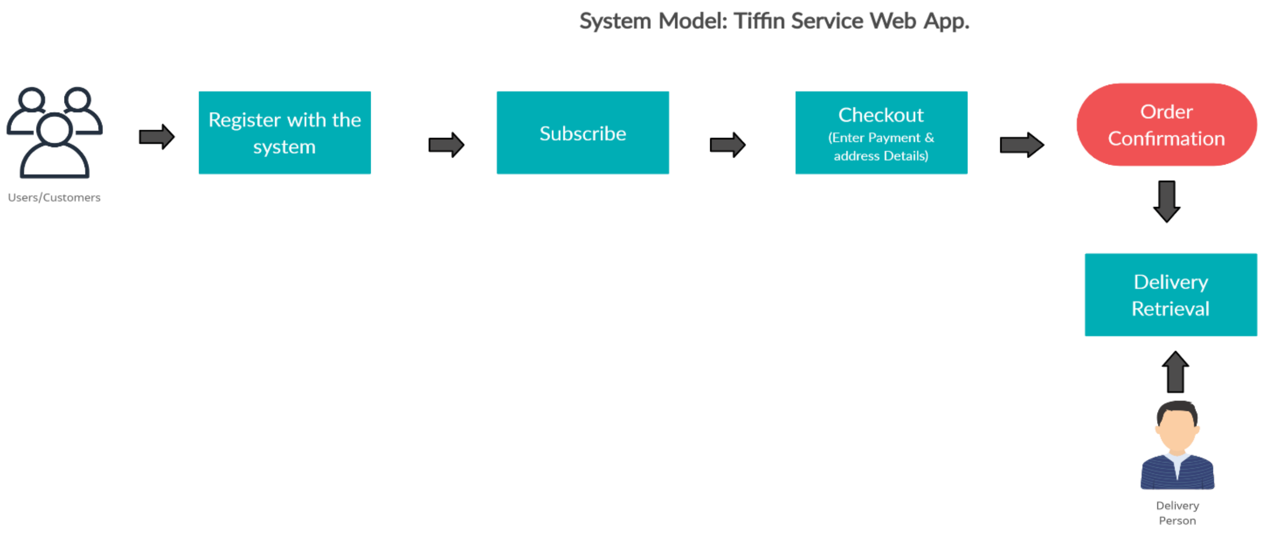
Prof. Mosin I Hasan

Table of Contents

Page no.

Introduction

The Tiffin Service Web Application is a web-based system. Basically, it allows the user to get 'home-made' food delivered to him/her. This product uses easy and manageable interface and hence allowing both the users i.e., customer and delivery person a trouble-free experience.



This project is the need of the hour because of the following reasons:

- In challenging COVID-19 times, people think a lot before ordering food from restaurants or tiffin services and so this app is needed to provide home-made food for people.
- Some people have to take their lunch along-with them when they leave their home for office. Hence, their food becomes cold till lunch time. With this app, you will be able to have fresh and hot food.
- Also, the person would not have to pay for the food but only have to pay for the delivery charges.

Literature Review/ Secondary Research

FOOD DELIVERY APPS CASE STUDY

1. Ghar Ka Dabba

Ghar Ka Dabba is the app to get meal/tiffin that you want at your door. This app delivers the food ‘made at their own kitchen’. It is based in Gandhinagar, Gujarat.

<u>How to use:</u>

1. Login to with your mobile number.

2. Add your address.

3. Check today's menu.

4. Choose Dabba/Items quantity to order.

5. Select your preferable delivery time

6. Place order.

<u>Technical Features:</u>

-Login is provided, can be done using Mobile Number(any other authentication is not provided)

- Stores our address(multiple addresses)

- Displays daily menu

- Multiple items can be ordered at a time

- Also provides flexible delivery timings

- Fast delivery

- Schedule order

- Easy checkout options with Cash on delivery

- Does not provide tracking of orders

2. Tiffi-Lo se Tiffin Lo

You can search and order from ‘Tiffin services nearby’. Tiffin service will be home delivered.

<u>How to Use?</u>

1. Login.

2. Select tiffin service nearby you.
3. Check today’s menu.
4. Choose quantity to order / take subscription.
5. Place order.

Technical Features:

- Login methods used are: Truecaller/Mobile Phone
- Suggests nearby kitchens/vendors
- Provides subscription which can be paused or cancelled
- App interface is easy to use
- Payment options of COD, Card, UPI are available
- Also shows previously ordered items and suggests similar places to order again
- Order tracking is available
- Search and discover top tiffin providers
- Details of the vendors are also provided
- UI is quite simple
- App uses GPS to track our location

3. Zomato

You can get your favorites delivered at your doorstep within minutes from your ‘favorite restaurants’.
Zomato is available across India, UAE and Lebanon.

How to use?

1. Login using your mobile number.
2. Add address.
3. Choose restaurant and food from its menu.
4. Choose quantity to order.
5. Place order.
6. Track your order and order status.

Technologies Used:
Zomato runs AWS Cloud & the backend is powered by Varnish, HAPROXY, APACHE/PHP, MEMCACHE, MYSQL, SOLR, Redis, Nodejs, Hypertable.

4. Swiggy Genie

Pick up or deliver anything in your city with Swiggy Genie. Send documents, packages and food items, pick up something you've forgotten, deliver gifts to your loved ones, get medicines delivered from pharmacies, order supplies from a local Kirana, and more. 'Launching soon in all cities.'

Features:

- Send documents, packages etc.
- Send or pick- up food items.
- Order anything from grocery stores as well.

Technical Features:

These are still not known as the app is not yet launched.

5. Khaoji Food and Tiffin Services

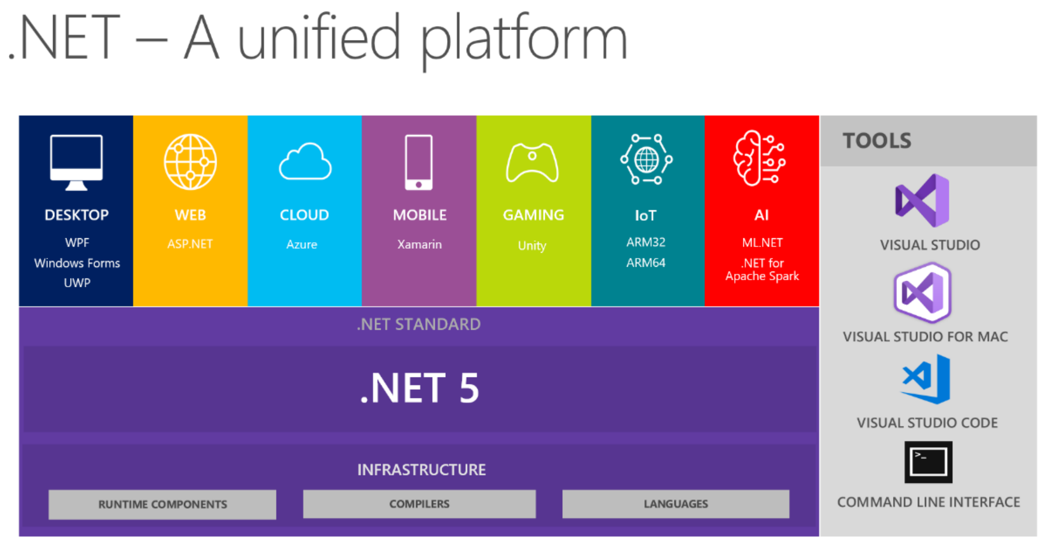
Online Tiffin & Catering Services with free delivery from 'Krupa's kitchen tiffin service'. Delivers 6 days from Monday to Saturday to your office or residence within specified time and area of delivery. Delivery time is between 12.30 pm to 2.00 pm.

How to use?

1. Login using your mobile number.
2. Select Punjabi or Gujarati cuisine.
3. Select subscription and quantity.
4. Place order.

ASP .NET CORE MVC TECHNOLOGY

- .NET core was not a new version of the .NET Framework, but it was an entirely new framework created to build Desktop, Web, Cloud and Mobile Applications.
- It is a Cross-Platform and Open-Source framework developed by Microsoft and released under MIT License.
- Same as .NET Core, it was architected modular with minimum overhead, and then other more advanced features can be added as NuGet packages as per application requirement. This results in high performance, require less memory, less deployment size, and easy to maintain.
- The .NET framework is a modular framework, and so it is possible to run two web applications with different versions of .NET on the same server.



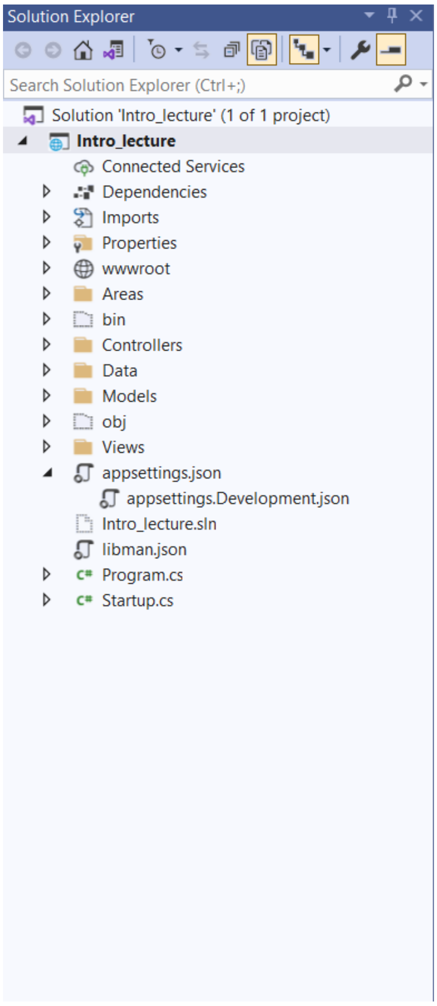
(Reference:https://devblogs.microsoft.com/dotnet/introducing-net-5/)

Some components of .NET

- Entity Framework Core- It is an open-source and cross platform version of Entity framework. It serves as Object Relation model(ORM) to connect to the database using the entity model.
- Identity Core- Identity Core is used to implement the form of authentication and roles and permission. It allows us to implements the login and register feature to the application.
- MVC Core- MVC Core is an open-source MVC project under the .NET Core Framework.

PROJECT STRUCTURE OF TSWA IN MVC:-

- **ASP .NET CORE PROJECT STRUCTURE**



Solution Explorer

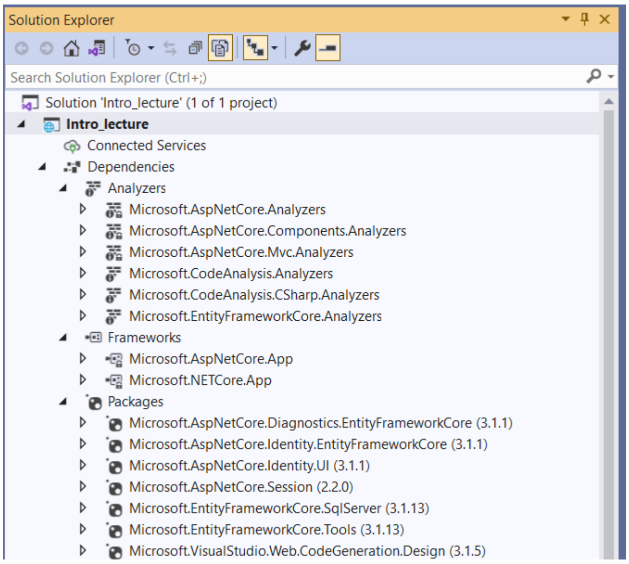
Search Solution Explorer (Ctrl+;)

```
Solution 'Intro_lecture' (1 of 1 project)
◢ Intro_lecture
    Connected Services
  ▷ Dependencies
  ▷ Imports
  ▷ Properties
  ▷ wwwroot
  ▷ Areas
  ▷ bin
  ▷ Controllers
  ▷ Data
  ▷ Models
  ▷ obj
  ▷ Views
  ◢ appsettings.json
      appsettings.Development.json
    Intro_lecture.sln
    libman.json
  ▷ Program.cs
  ▷ Startup.cs
```

fig: The Solution Explorer of TSWA from Visual Studio

Dependencies

The Dependencies in the ASP.NET Core 3.0 project contain all the installed server-side NuGet packages.

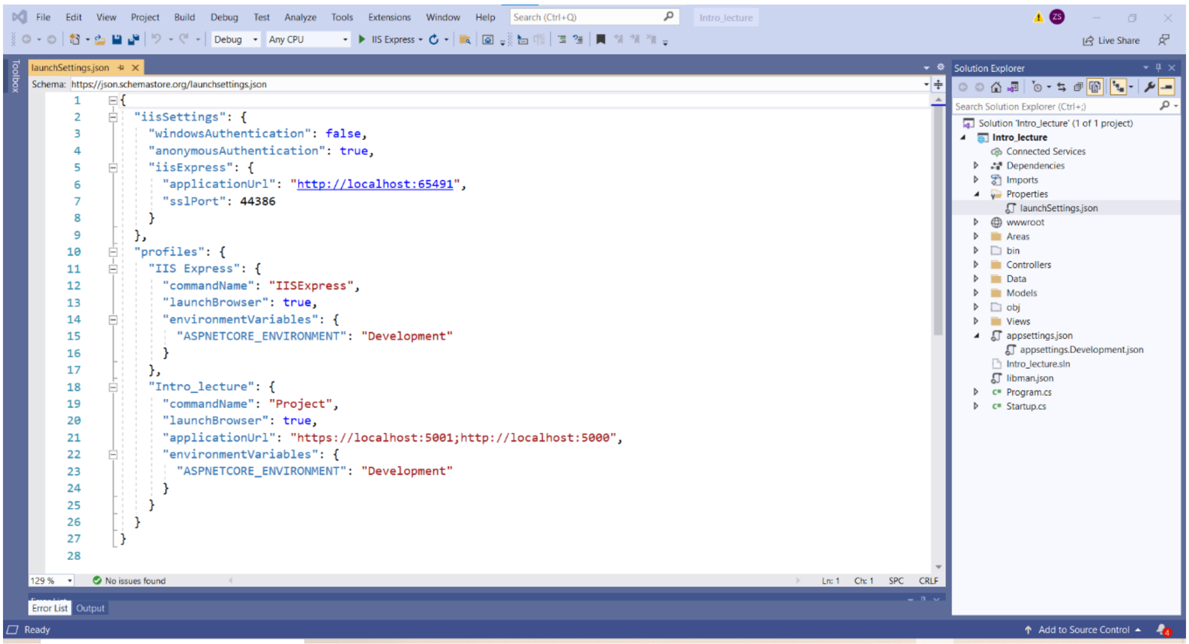

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Intro_lecture' (1 of 1 project)
Intro_lecture
Connected Services
Dependencies
Analyzers
Microsoft.AspNetCore.Analyzers
Microsoft.AspNetCore.Components.Analyzers
Microsoft.AspNetCore.Mvc.Analyzers
Microsoft.CodeAnalysis.Analyzers
Microsoft.CodeAnalysis.CSharp.Analyzers
Microsoft.EntityFrameworkCore.Analyzers
Frameworks
Microsoft.AspNetCore.App
Microsoft.NETCore.App
Packages
Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore (3.1.1)
Microsoft.AspNetCore.Identity.EntityFrameworkCore (3.1.1)
Microsoft.AspNetCore.Identity.UI (3.1.1)
Microsoft.AspNetCore.Session (2.2.0)
Microsoft.EntityFrameworkCore.SqlServer (3.1.13)
Microsoft.EntityFrameworkCore.Tools (3.1.13)
Microsoft.VisualStudio.Web.CodeGeneration.Design (3.1.5)

(fig: The Dependencies folder of TSWA project)

Properties

The Properties node includes launchSettings.json file which includes Visual Studio profiles of debug settings. The following is a default launchSettings.json file.

(fig: the launchSettings.json file in Properties folder)



```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:65491",
      "sslPort": 44386
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "Intro_lecture": {
      "commandName": "Project",
      "launchBrowser": true,
      "applicationUrl": "https://localhost:5001;http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

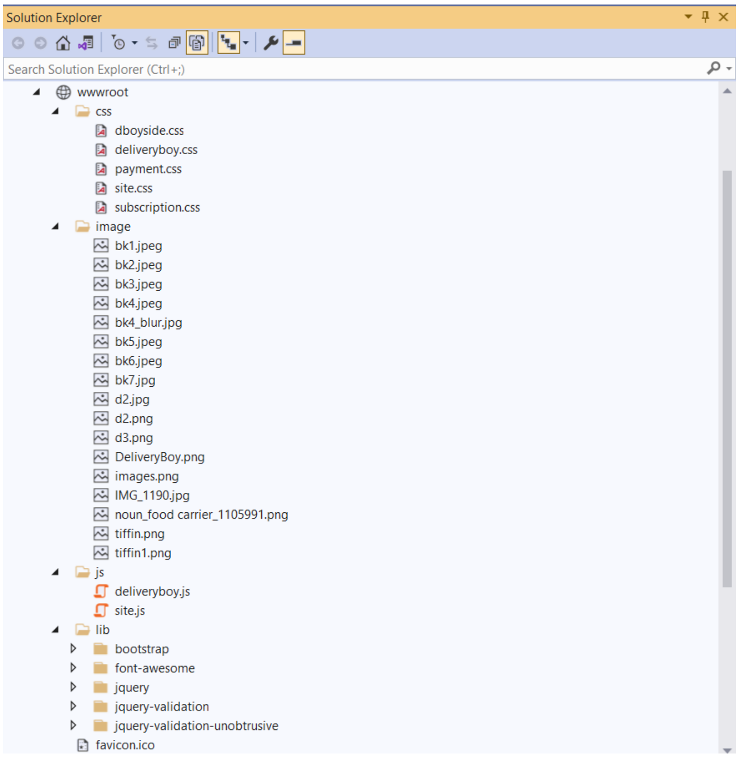
wwwroot Folder

By default, the wwwroot folder in the ASP.NET Core project is treated as a web root folder. Static files can be stored in any folder under the web root and accessed with a relative path to that root.

In the standard ASP.NET application, static files can be served from the root folder of an application or any other folder under it. This has been changed in ASP.NET Core. Now, only those files that are in the web root - wwwroot folder can be served over an http request. All other files are blocked and cannot be served by default.

Generally, there should be separate folders for the different types of static files such as JavaScript, CSS, Images, library scripts etc. in the wwwroot folder as shown below.

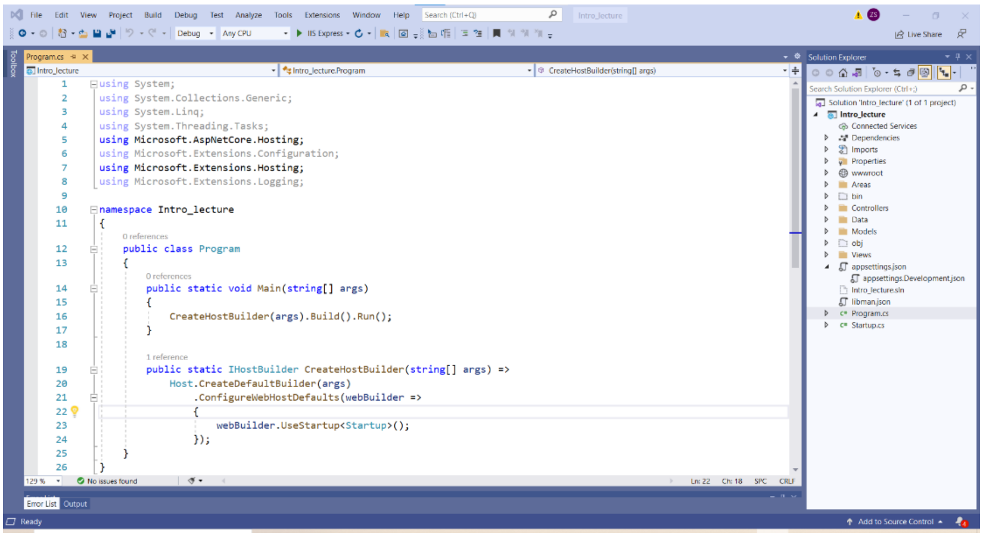
(fig: the wwwroot folder of TSWA project)



```
wwwroot
  css
    dboyside.css
    deliveryboy.css
    payment.css
    site.css
    subscription.css
  image
    bk1.jpeg
    bk2.jpeg
    bk3.jpeg
    bk4.jpeg
    bk4_blur.jpg
    bk5.jpeg
    bk6.jpeg
    bk7.jpg
    d2.jpg
    d2.png
    d3.png
    DeliveryBoy.png
    images.png
    IMG_1190.jpg
    noun_food carrier_1105991.png
    tiffin.png
    tiffin1.png
  js
    deliveryboy.js
    site.js
  lib
    bootstrap
    font-awesome
    jquery
    jquery-validation
    jquery-validation-unobtrusive
  favicon.ico
```

Program.cs

ASP.NET Core web application is actually a console project which starts executing from the entry point public static void Main() in Program class where we can create a host for the web application.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace Intro_lecture
{
    0 references
    public class Program
    {
        0 references
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        1 reference
        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

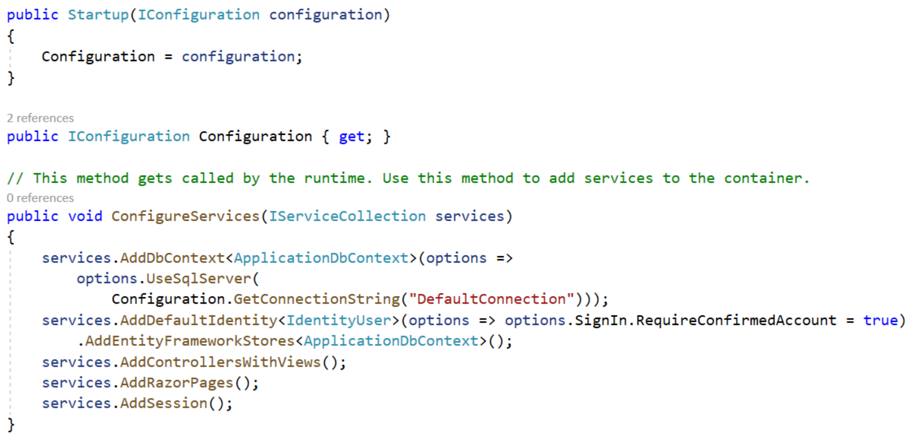
(fig: Program.cs file of the project)

As you can see above, the Main() method calls method expression CreateHostBuilder() to build web host with pre-configured defaults. The CreateHostBuilder expression can also be written as a method that returns IHostBuilder as shown.

The HostBuilder is a static class which can be used for creating an instance of IWebHost and IWebHostBuilder with pre-configured defaults. The CreateDefaultBuilder() method creates a new instance of WebHostBuilder with pre-configured defaults. Internally, it configures Kestrel, IISIntegration and other configurations. The following is CreateDefaultBuilder() method.

Startup.cs

The name "Startup" is by ASP.NET Core convention. However, we can give any name to the Startup class, just specify it as the generic parameter in the UseStartup<T>() method. For example, to name the Startup class as MyStartup, specify it as .UseStartup<MyStartup>().

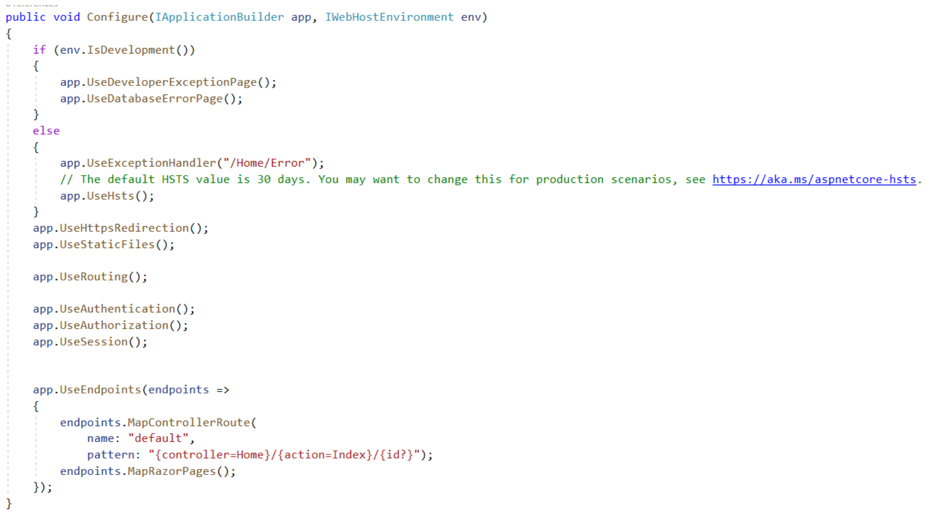


```
public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}

2 references
public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)
        .AddEntityFrameworkStores<ApplicationDbContext>();
    services.AddControllersWithViews();
    services.AddRazorPages();
    services.AddSession();
}
```

(fig: Startup.cs file of the project)



```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthentication();
    app.UseAuthorization();
    app.UseSession();


    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
        endpoints.MapRazorPages();
    });
}
```

(fig: Startup.cs file-continuation)

As you can see, Startup class includes two public methods: ConfigureServices and Configure.

The Startup class must include a Configure method and can optionally include ConfigureService method.

ConfigureServices()
The Dependency Injection pattern is used heavily in ASP.NET Core architecture. It includes built-in IoC container to provide dependent objects using constructors.

The ConfigureServices method is a place where you can register your dependent classes with the built-in IoC container. After registering dependent class, it can be used anywhere in the application. You just need to include it in the parameter of the constructor of a class where you want to use it. The IoC container will inject it automatically.

ASP.NET Core refers dependent class as a Service. So, whenever you read "Service" then understand it as a class which is going to be used in some other class.

ConfigureServices method includes IServiceCollection parameter to register services to the IoC container. Learn more about it in the next chapter.

Configure()
The Configure method is a place where you can configure application request pipeline for your application using IApplicationBuilder instance that is provided by the built-in IoC container.

ASP.NET Core introduced the middleware components to define a request pipeline, which will be executed on every request.

DEPENDENCY INJECTION

Dependency Injection is a technique whereby one object(or static method) supplies the dependencies of another object.

A dependency is an object that can be used (a service).

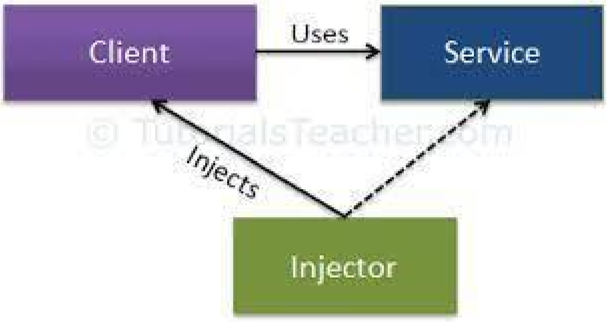
Dependency Injection is an implementation of "Inversion of Control". Inversion of Control (IoC) says that the objects do not create other objects on which they rely to do their work; instead, they get the objects that they need from an outside source (for example, an XML configuration file).

The Dependency Injection pattern involves 3 types of classes.

Client Class: The client class (dependent class) is a class which depends on the service class

Service Class: The service class (dependency) is a class that provides service to the client class.

Injector Class: The injector class injects the service class object into the client class.



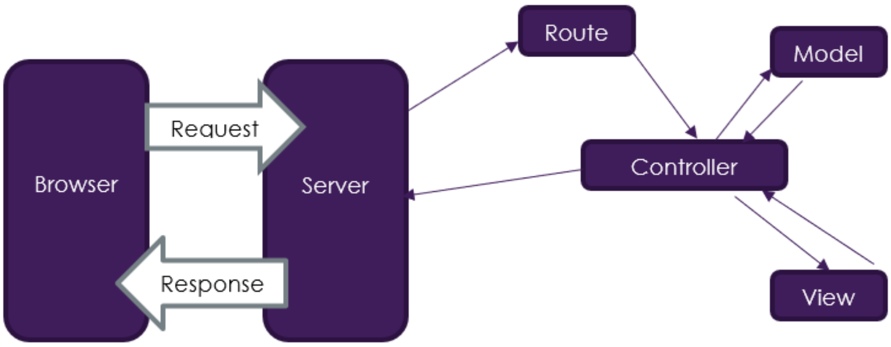
Reference
:https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.tutorialsteacher.com%2FContent%2Fimages%2Fioc%2FDI.png&imgrefurl=https%3A%2F%2Fwww.tutorialsteacher.com%2Fioc%2Fdependency-injection&tbnid=5BOdcGvAN3sk2M&vet=12ahUKEwjj_ouW5IXwAhVHAnIKHcS8D64QMygRegUIARDUAQ..i&docid=Vzuvt0rPCEK7AM&w=369&h=198&q=dependency%20injection%20in%20mvc&ved=2ahUKEwjj_ouW5IXwAhVHAnIKHcS8D64QMygRegUIARDUAQ

WHAT EXACTLY IS MVC?

MODEL: The model is responsible for managing the data of the application. It contains no business logic. In simple term, models are the entity representation of the database tables that consists the properties of table columns.
VIEW: View is the user interface; it contains the Markup(Razor Syntax). It uses the Razor engine to render the view. View renders the model data passed to it via any controller. C# code can also be written in a view.
CONTROLLER: Controller handles the user interaction, and it is responsible for invoking the action bases on the route and fetching data from the model to render the view.

- In MVC architecture, the controller receives the HTTP request and then works with the model to get the data and render it to the view.
- MVC isolates the application logic from the user interface layer and supports the separation of concerns.
- The controller specifies which view to render based on the HTTP request.

Workflow of MVC architecture.

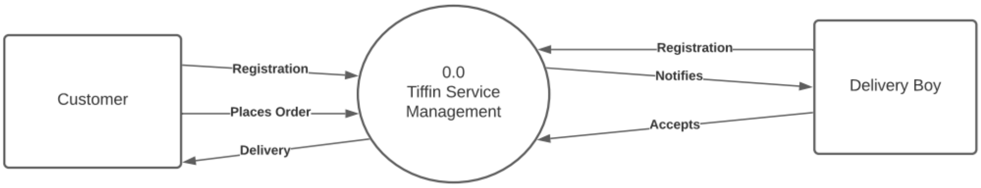


(fig: Workflow of MVC architecture)

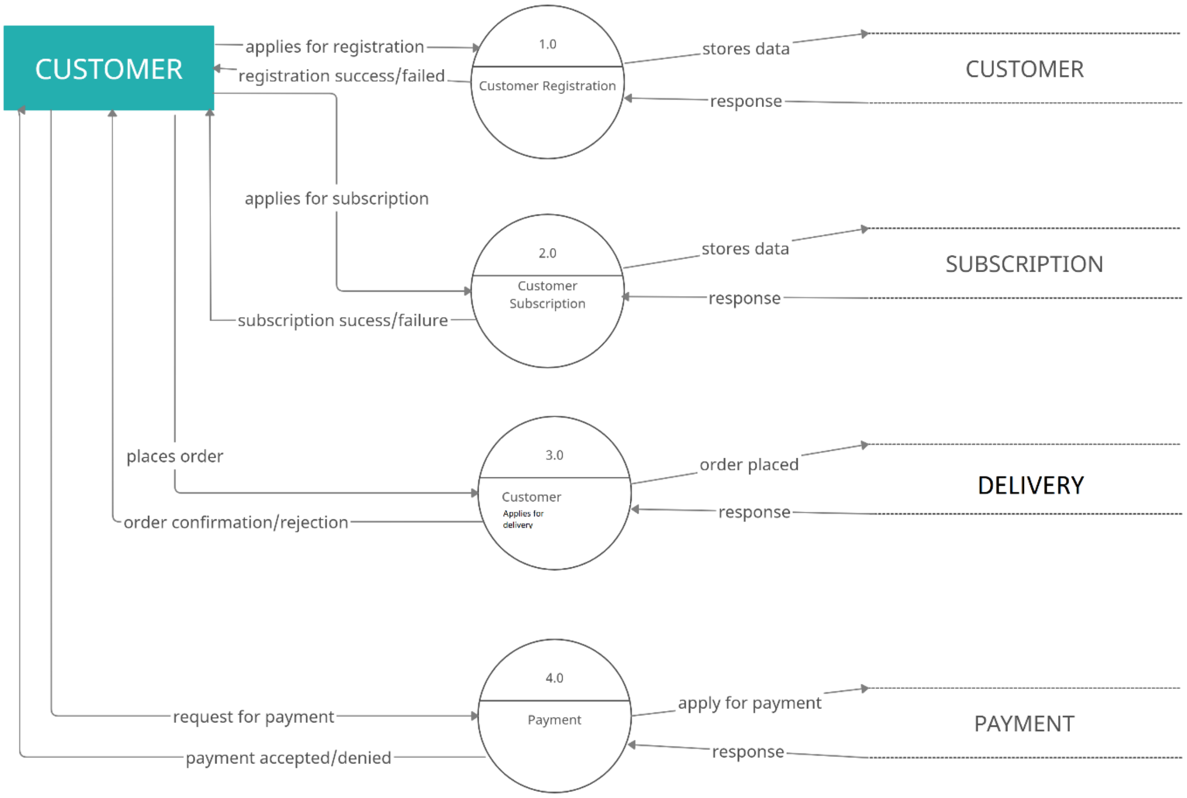
Design Considerations for Detail Design

1) Data Flow Diagram

Level 0:

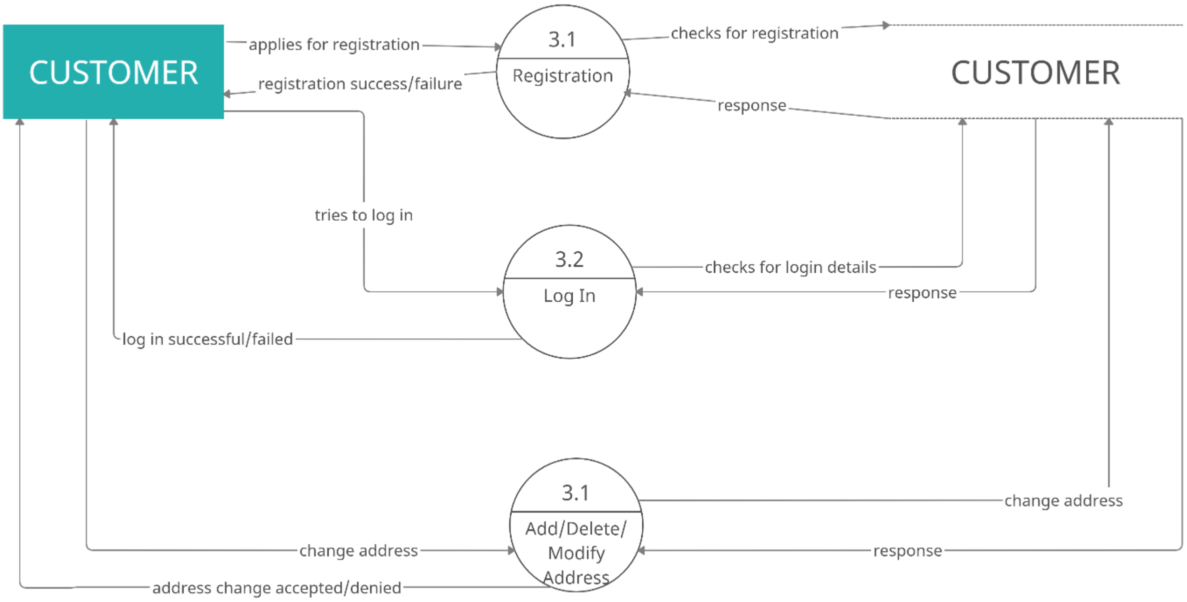


Level 1:

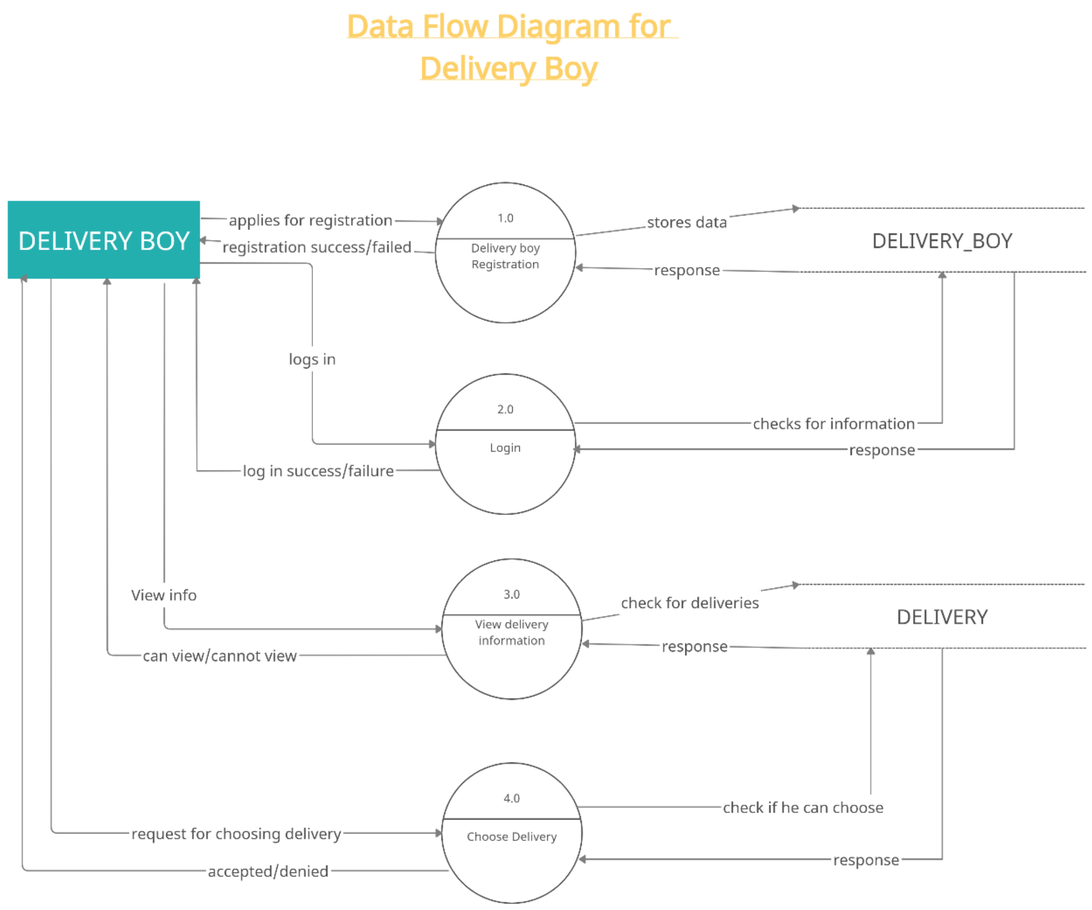

CUSTOMER
applies for registration
registration success/failed
1.0
Customer Registration
stores data
response
CUSTOMER
applies for subscription
2.0
Customer Subscription
subscription sucess/failure
stores data
response
SUBSCRIPTION
places order
order confirmation/rejection
3.0
Customer
Applies for delivery
order placed
response
DELIVERY
request for payment
payment accepted/denied
4.0
Payment
apply for payment
response
PAYMENT

Level 2:

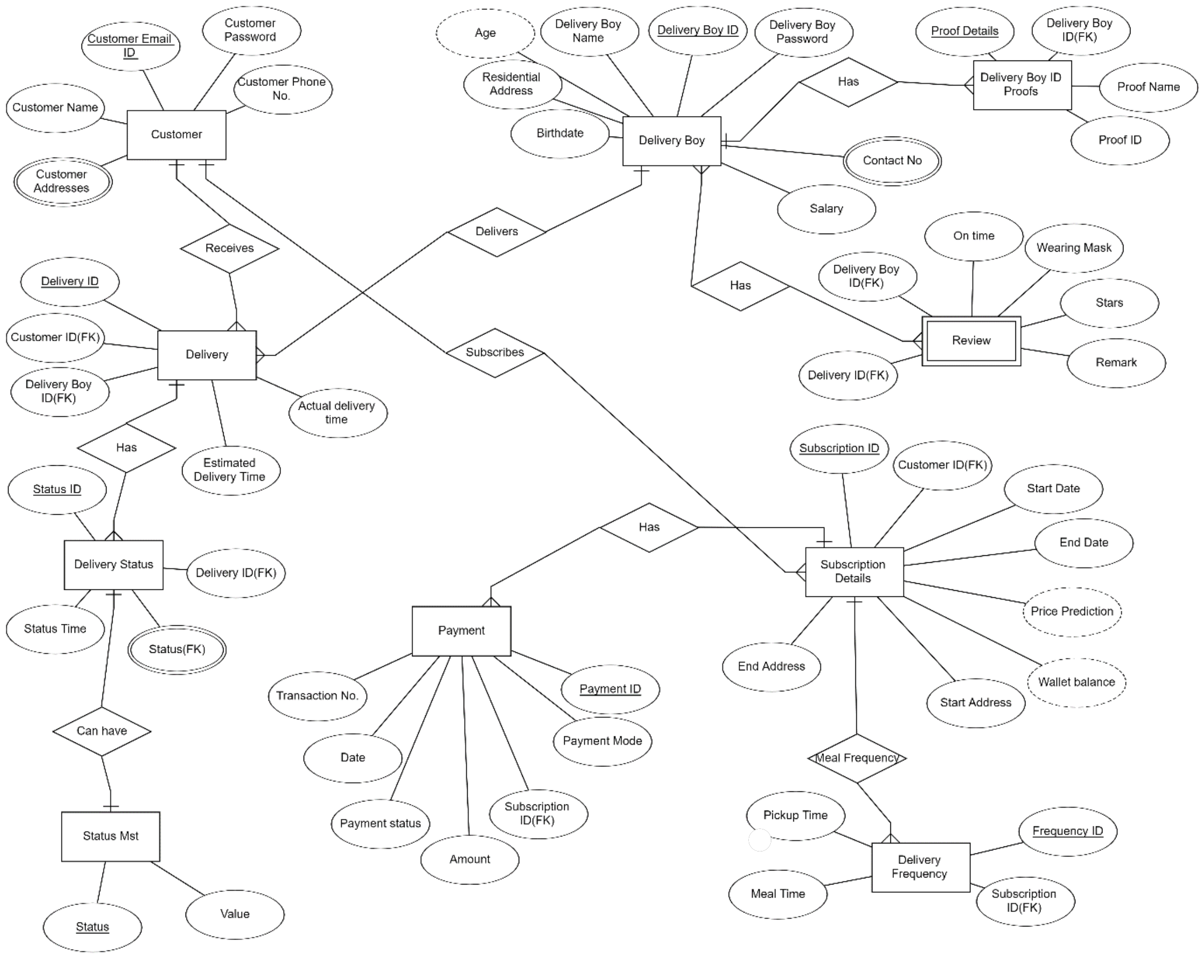
2nd Level DFD for Customer



Level 2: Delivery Boy

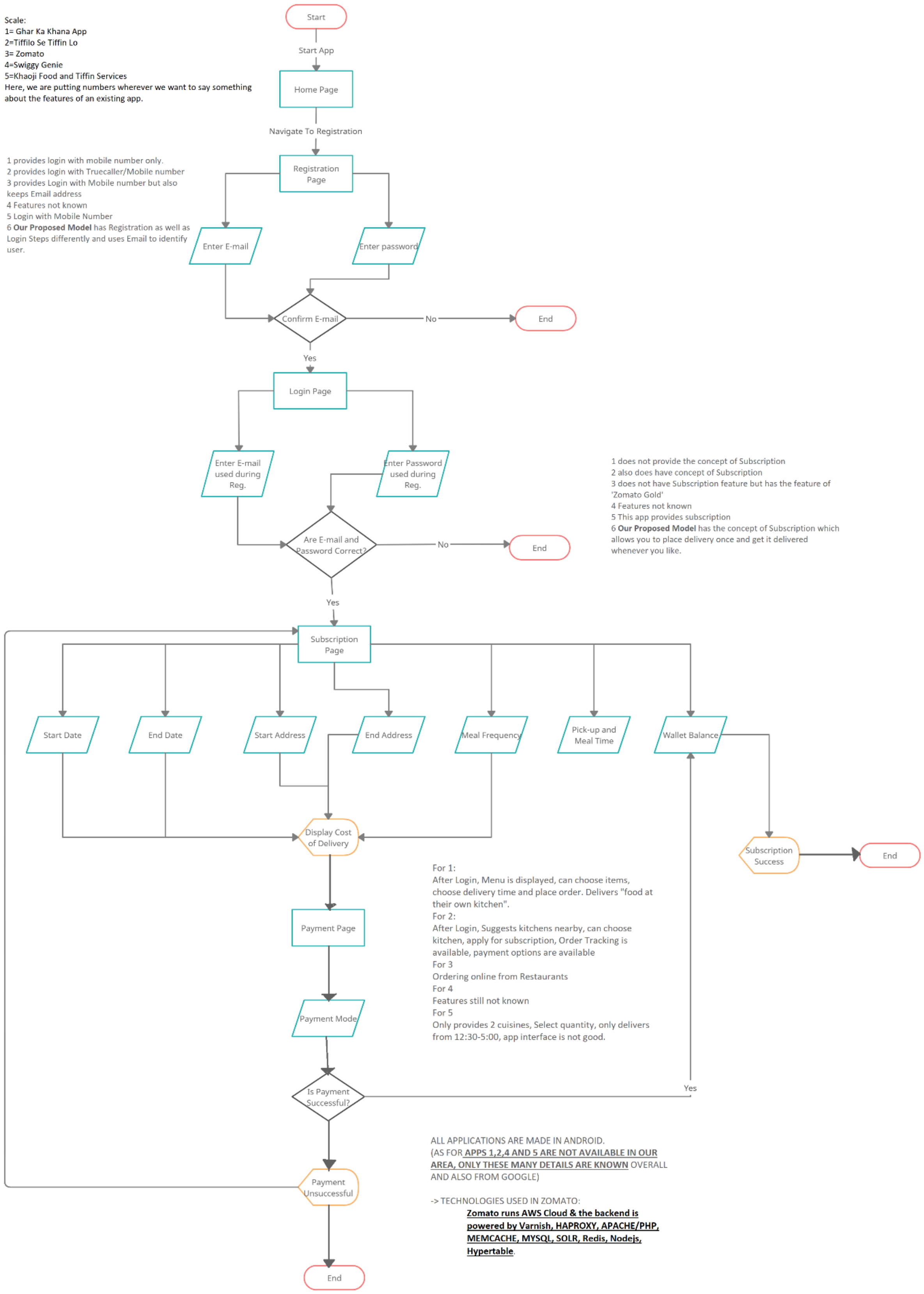


2) ER Diagram


Customer Email ID
Customer Password
Customer Phone No.
Customer Name
Customer
Customer Addresses
Age
Delivery Boy Name
Delivery Boy ID
Delivery Boy Password
Residential Address
Birthdate
Delivery Boy
Has
Contact No
Salary
Proof Details
Delivery Boy ID(FK)
Delivery Boy ID Proofs
Proof Name
Proof ID
Receives
Delivers
Delivery ID
Customer ID(FK)
Delivery
Delivery Boy ID(FK)
Actual delivery time
Estimated Delivery Time
Has
Subscribes
Has
On time
Wearing Mask
Delivery Boy ID(FK)
Stars
Review
Remark
Delivery ID(FK)
Status ID
Delivery Status
Delivery ID(FK)
Status Time
Status(FK)
Can have
Status Mst
Status
Value
Has
Payment
Transaction No.
Date
Payment ID
Payment Mode
Payment status
Amount
Subscription ID(FK)
Subscription ID
Customer ID(FK)
Start Date
End Date
Subscription Details
Price Prediction
End Address
Wallet balance
Start Address
Meal Frequency
Pickup Time
Delivery Frequency
Frequency ID
Meal Time
Subscription ID(FK)

3) Unique Prototyping Diagram for proposed model

User/Customer:



Scale:
1= Ghar Ka Khana App
2=Tiffilo Se Tiffin Lo
3= Zomato
4=Swiggy Genie
5=Khaoji Food and Tiffin Services
Here, we are putting numbers wherever we want to say something about the features of an existing app.

1 provides login with mobile number only.
2 provides login with Truecaller/Mobile number
3 provides Login with Mobile number but also keeps Email address
4 Features not known
5 Login with Mobile Number
6 **Our Proposed Model** has Registration as well as Login Steps differently and uses Email to identify user.

1 does not provide the concept of Subscription
2 also does have concept of Subscription
3 does not have Subscription feature but has the feature of 'Zomato Gold'
4 Features not known
5 This app provides subscription
6 **Our Proposed Model** has the concept of Subscription which allows you to place delivery once and get it delivered whenever you like.

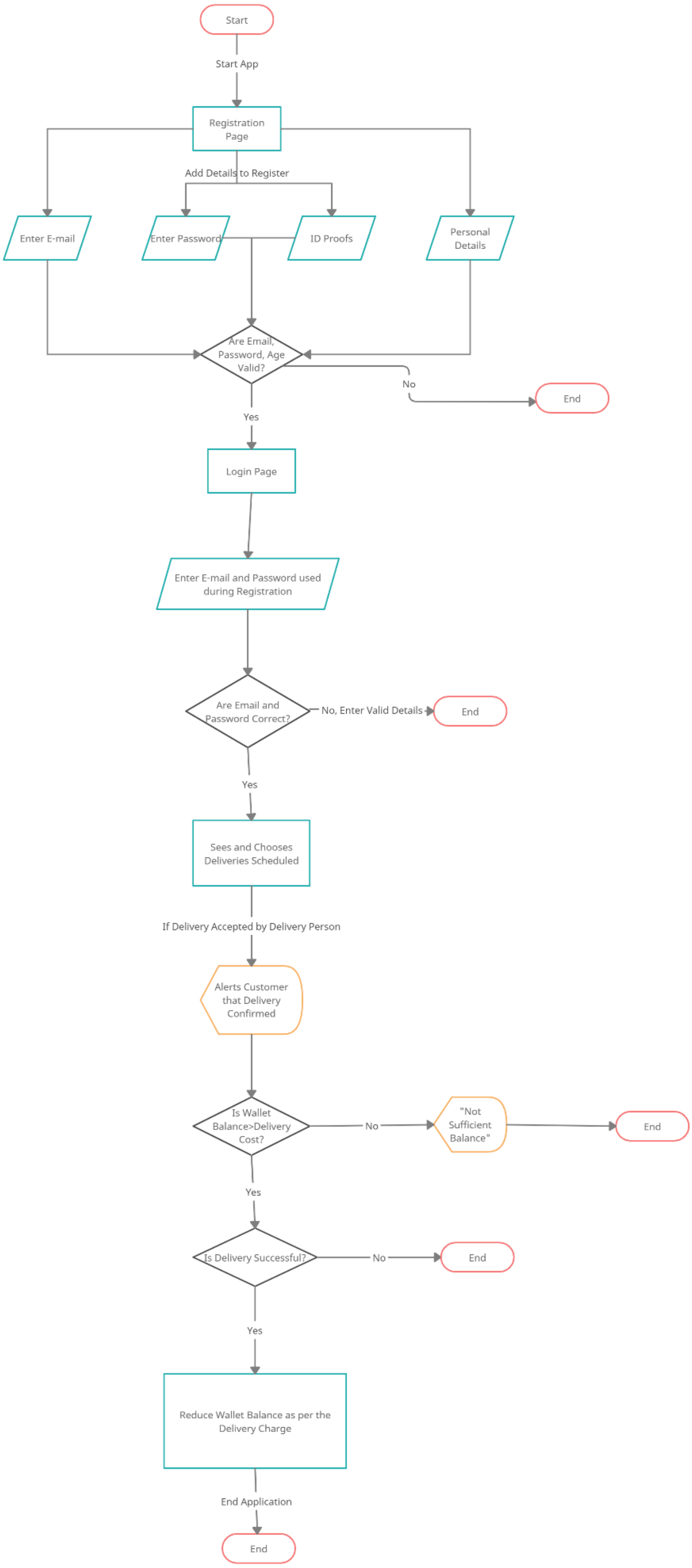
For 1:
After Login, Menu is displayed, can choose items, choose delivery time and place order. Delivers "food at their own kitchen".
For 2:
After Login, Suggests kitchens nearby, can choose kitchen, apply for subscription, Order Tracking is available, payment options are available
For 3
Ordering online from Restaurants
For 4
Features still not known
For 5
Only provides 2 cuisines, Select quantity, only delivers from 12:30-5:00, app interface is not good.

ALL APPLICATIONS ARE MADE IN ANDROID.
(AS FOR **APPS 1,2,4 AND 5 ARE NOT AVAILABLE IN OUR AREA, ONLY THESE MANY DETAILS ARE KNOWN** OVERALL AND ALSO FROM GOOGLE)

-> TECHNOLOGIES USED IN ZOMATO:

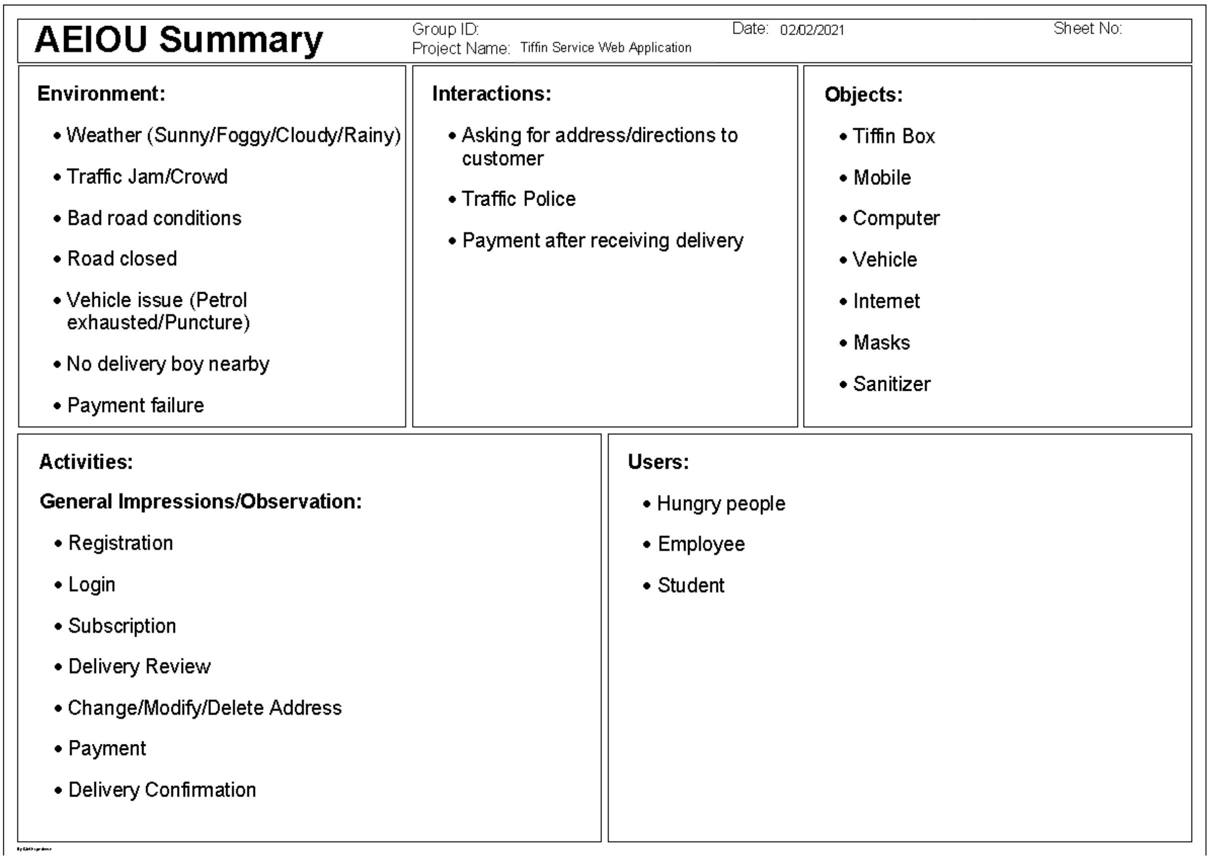
Zomato runs AWS Cloud & the backend is powered by Varnish, HAPROXY, APACHE/PHP, MEMCACHE, MYSQL, SOLR, Redis, Nodejs, Hypertable.

Delivery Boy:


Start
Start App
Registration Page
Add Details to Register
Enter E-mail
Enter Password
ID Proofs
Personal Details
Are Email, Password, Age Valid?
No
End
Yes
Login Page
Enter E-mail and Password used during Registration
Are Email and Password Correct?
No, Enter Valid Details
End
Yes
Sees and Chooses Deliveries Scheduled
If Delivery Accepted by Delivery Person
Alerts Customer that Delivery Confirmed
Is Wallet Balance>Delivery Cost?
No
"Not Sufficient Balance"
End
Yes
Is Delivery Successful?
No
End
Yes
Reduce Wallet Balance as per the Delivery Charge
End Application
End

Designing Canvas

1)AEIOU



AEIOU Summary

Group ID:
Project Name: Tiffin Service Web Application
Date: 02/02/2021
Sheet No:

Environment:

- Weather (Sunny/Foggy/Cloudy/Rainy)
- Traffic Jam/Crowd
- Bad road conditions
- Road closed
- Vehicle issue (Petrol exhausted/Puncture)
- No delivery boy nearby
- Payment failure

Interactions:

- Asking for address/directions to customer
- Traffic Police
- Payment after receiving delivery

Objects:

- Tiffin Box
- Mobile
- Computer
- Vehicle
- Internet
- Masks
- Sanitizer

Activities:

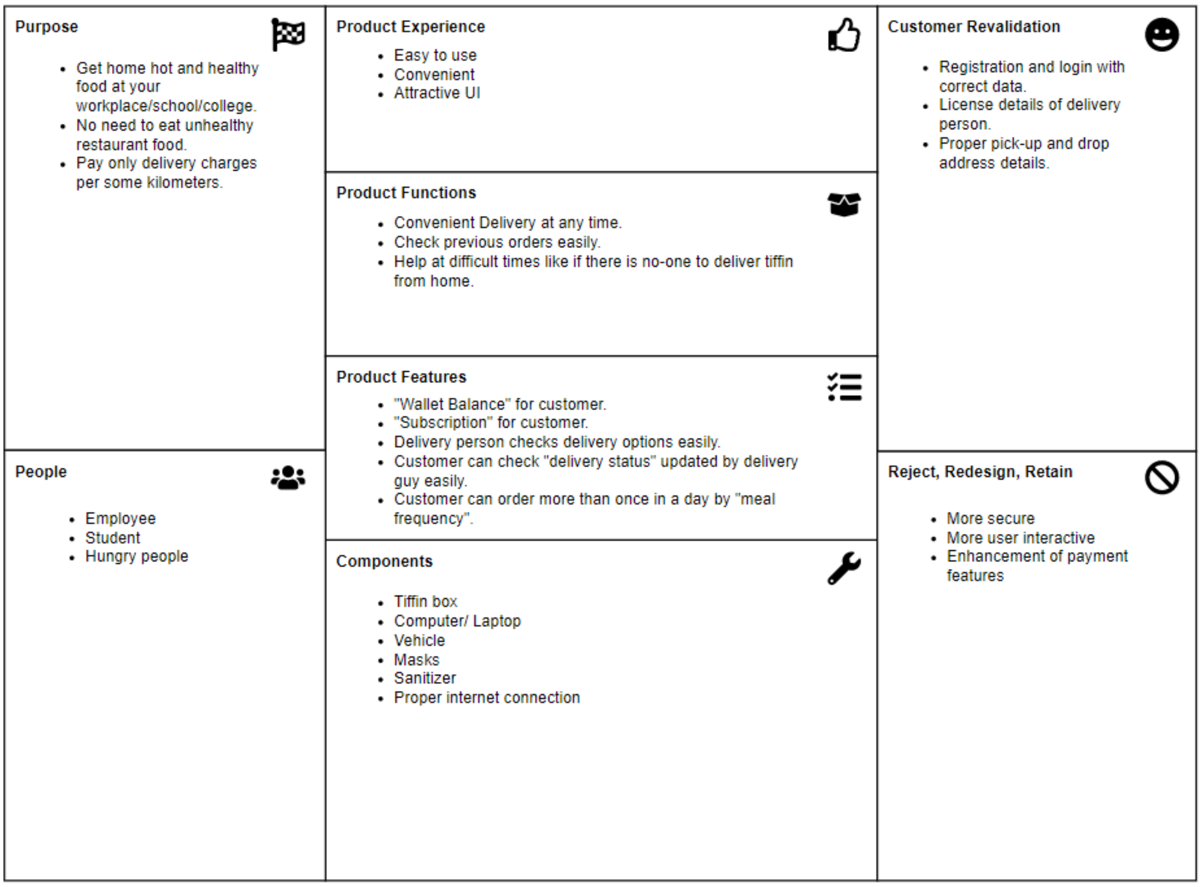
General Impressions/Observation:

- Registration
- Login
- Subscription
- Delivery Review
- Change/Modify/Delete Address
- Payment
- Delivery Confirmation

Users:

- Hungry people
- Employee
- Student

2) Product Development Canvas



Purpose

- Get home hot and healthy food at your workplace/school/college.
- No need to eat unhealthy restaurant food.
- Pay only delivery charges per some kilometers.

People

- Employee
- Student
- Hungry people

Product Experience

- Easy to use
- Convenient
- Attractive UI

Product Functions

- Convenient Delivery at any time.
- Check previous orders easily.
- Help at difficult times like if there is no-one to deliver tiffin from home.

Product Features

- "Wallet Balance" for customer.
- "Subscription" for customer.
- Delivery person checks delivery options easily.
- Customer can check "delivery status" updated by delivery guy easily.
- Customer can order more than once in a day by "meal frequency".

Components

- Tiffin box
- Computer/ Laptop
- Vehicle
- Masks
- Sanitizer
- Proper internet connection

Customer Revalidation

- Registration and login with correct data.
- License details of delivery person.
- Proper pick-up and drop address details.

Reject, Redesign, Retain

- More secure
- More user interactive
- Enhancement of payment features

Implementation/ Simulation and Analysis

Technologies Used: -	HTML, CSS, JavaScript, Bootstrap, Asp .Net Core Mvc, Sql Server, Entity Core Framework
Tools: -	IDE: Visual Studio 2019 Database: MS-SQL Web Browser: Google Chrome Web Server: IIS Express

Module Description: -

The modules of our proposed Tiffin Service Web Application are as follows: -

a. Registration for Customer

This module deals with the basic registration of the customer who wants to avail the delivery service. The Email Id and password entered during the registration process can be used to login into the account.

b. Subscription

The person who wants to book their delivery will need to subscribe with the application and then only they will be allowed to proceed further.

c. Registration for the Delivery Boy

If a person who wants to work as a delivery partner with this app need to register themselves with the application and provide all the details required. After the registration process is completed, he/she can select the deliveries and deliver them.

d. Login

Customer/Delivery boy can login into the system to see their account details/ deliveries for the delivery boy as well as customer.

e. Payment

Payment options like card, UPI transaction are available.

f. Delivery Status

The delivery status is for the customer to know the whereabouts of their food. It can be updated by the delivery boy and can be viewed on the customer side.

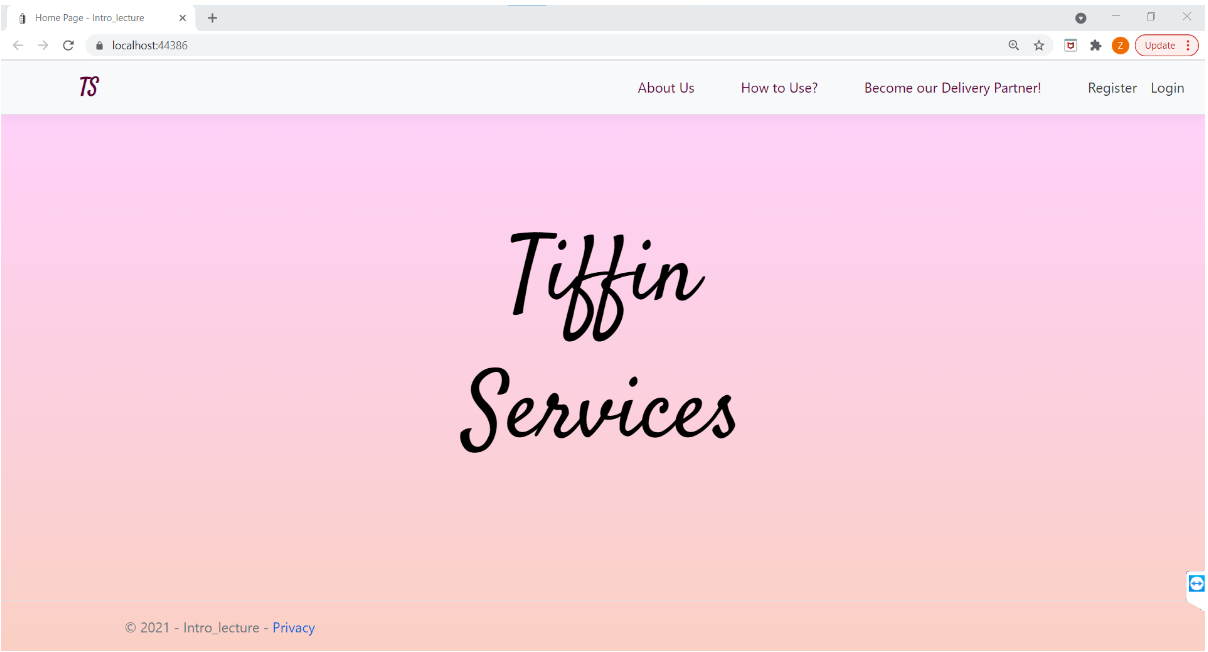
Database Description: -

The database of our proposed system is composed of following data-tables: -

- **AspNetUsers**: containing User/Customer Details
- **Subscription_Details:** containing details of Subscription of the customer
- **DeliveryBoy:** containing data of delivery boy
- **DeliveryFinal**: the data of particular delivery-deliveryboy-customer
- **Payment:** payment data

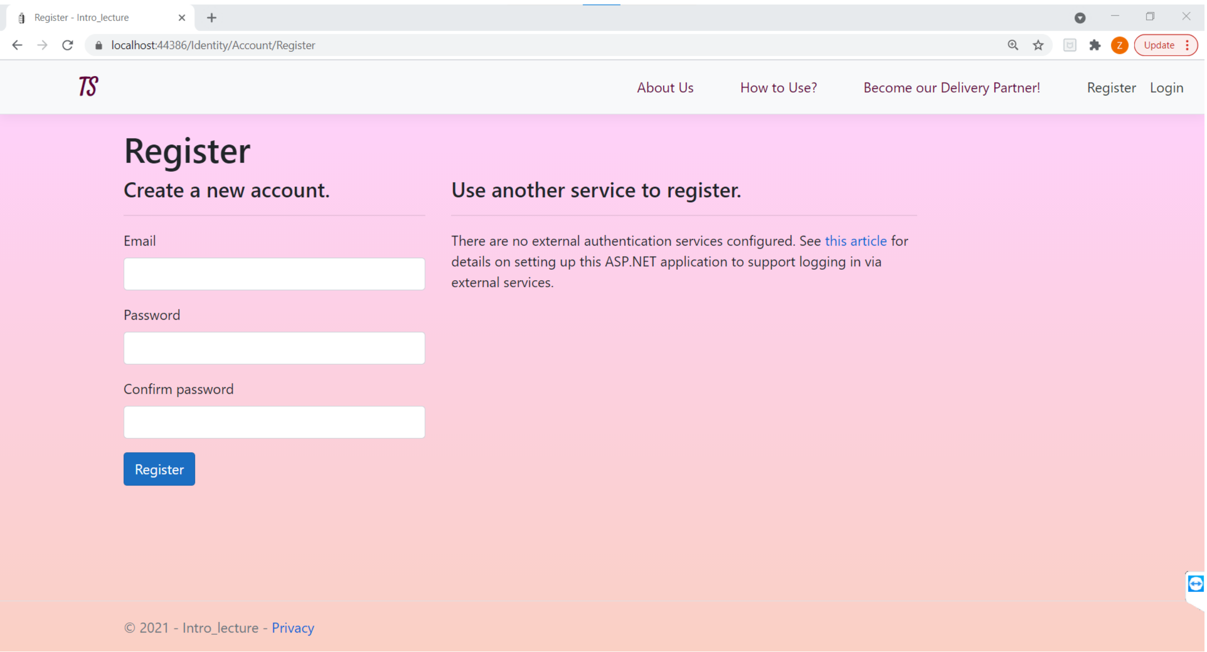
Snapshots of the Prototype: Tiffin Service Web Application

1. Home Page/Welcome Page



© 2021 - Intro_lecture - Privacy

2. Registration Page


Register - Intro_lecture
localhost:44386/Identity/Account/Register
Update
TS
About Us
How to Use?
Become our Delivery Partner!
Register
Login

Register

Create a new account.

Email

Password

Confirm password

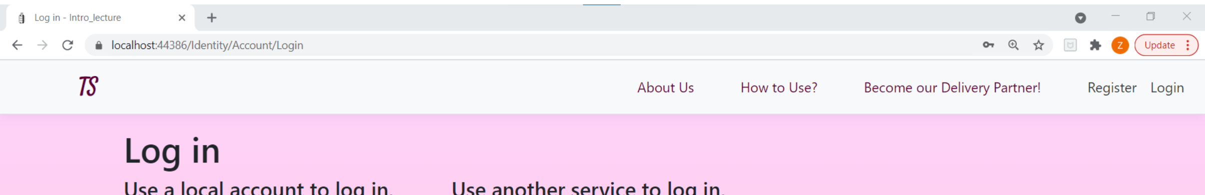
Register

Use another service to register.

There are no external authentication services configured. See this article for details on setting up this ASP.NET application to support logging in via external services.

© 2021 - Intro_lecture - Privacy

3. Customer Login Page

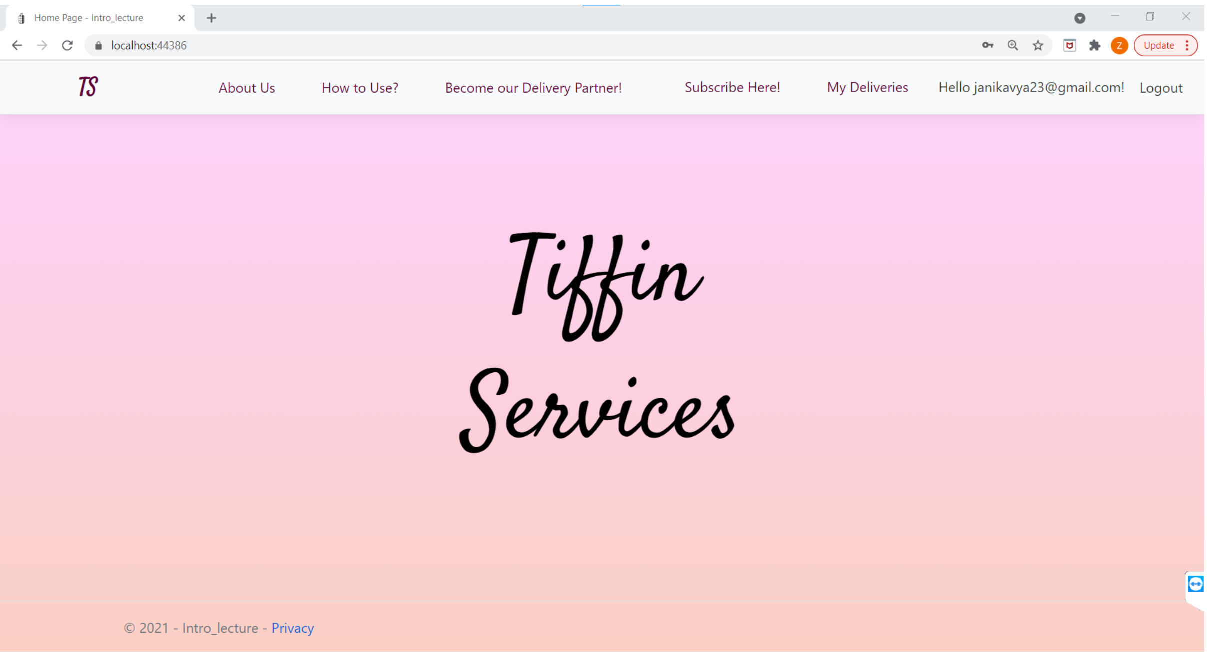

Log in - Intro_lecture
localhost:44386/Identity/Account/Login
Update
TS
About Us
How to Use?
Become our Delivery Partner!
Register
Login

Log in

Use a local account to log in.

Use another service to log in.

4. User Home Page after Login



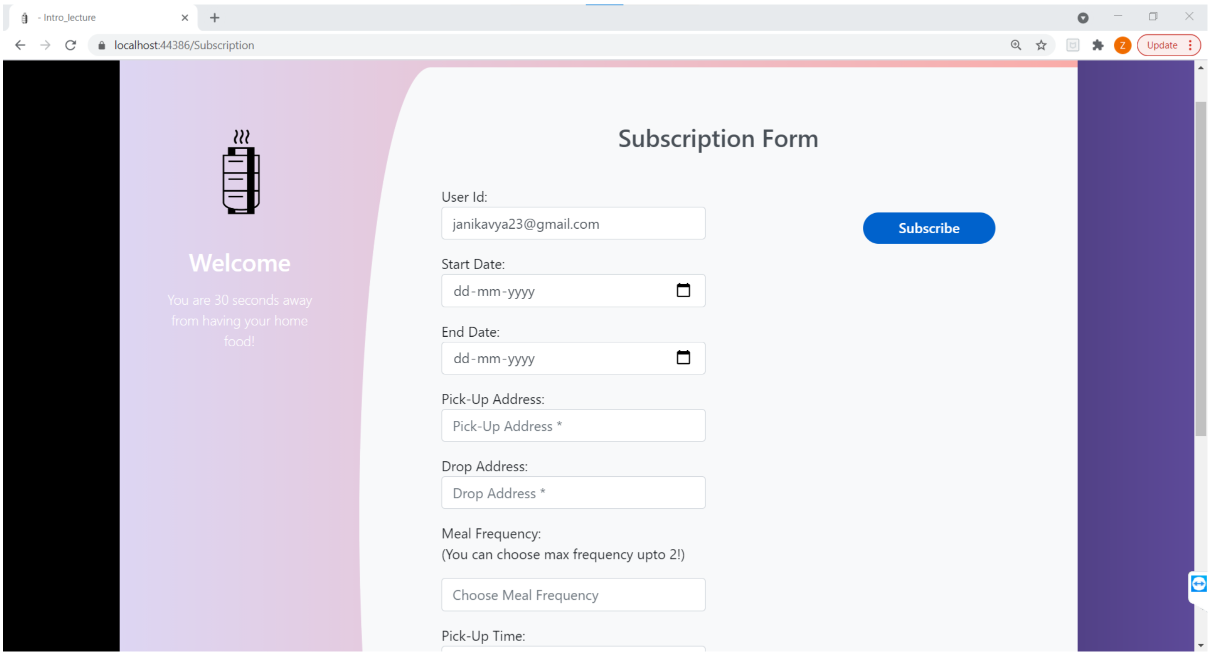
Home Page - Intro_lecture

localhost:44386

Tiffin Services

© 2021 - Intro_lecture - Privacy

5. Subscription form



Welcome

You are 30 seconds away from having your home food!

Subscription Form

User Id:
janikavya23@gmail.com

Start Date:
dd-mm-yyyy

End Date:
dd-mm-yyyy

Pick-Up Address:
Pick-Up Address *

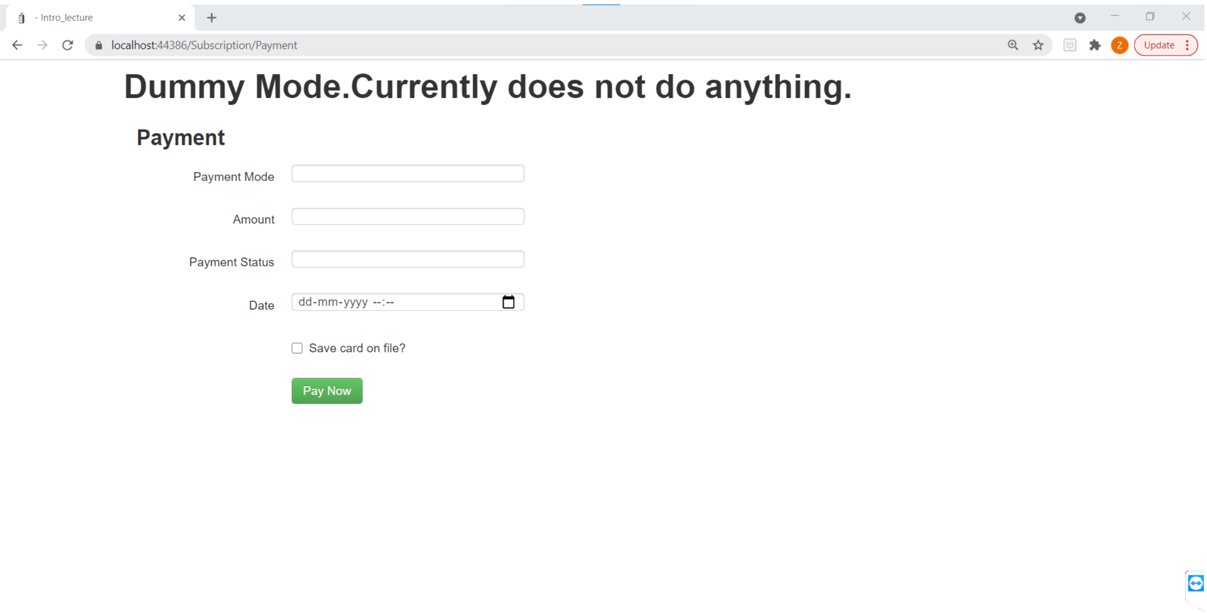
Drop Address:
Drop Address *

Meal Frequency:
(You can choose max frequency upto 2!)

Choose Meal Frequency

Pick-Up Time:

6. Payment Page(dummy)



Dummy Mode.Currently does not do anything.

Payment

Payment Mode

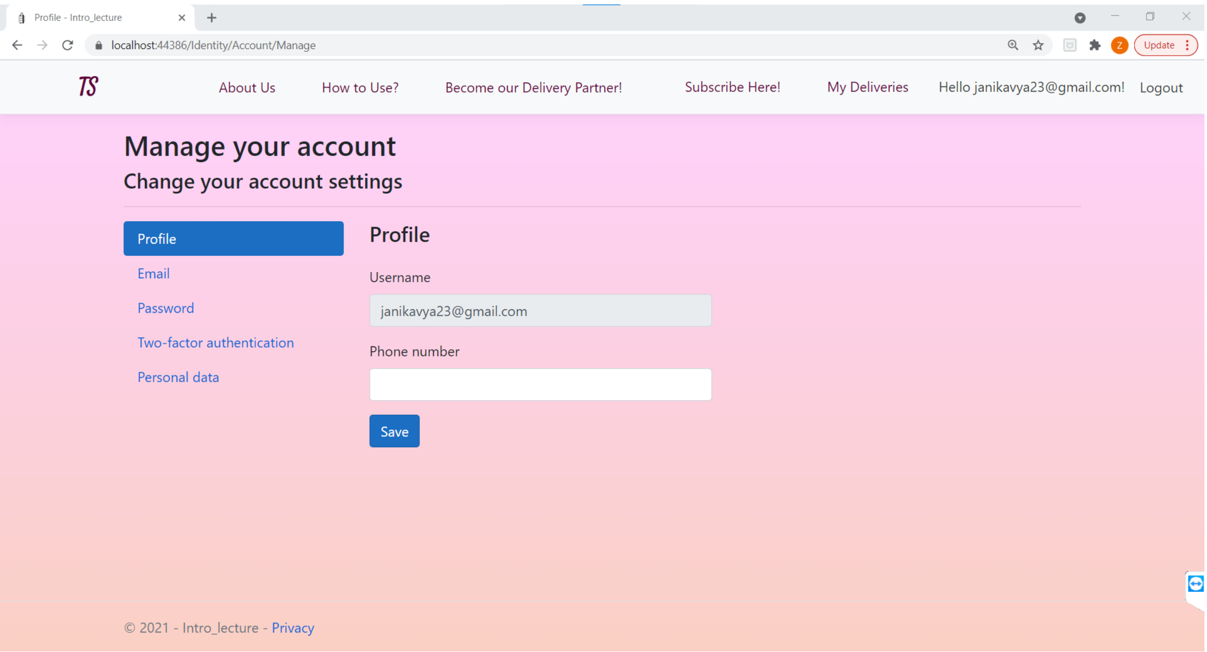
Amount

Payment Status

Date dd-mm-yyyy --:--

□ Save card on file?

7. User Account Manage Page


Profile - Intro_lecture
localhost:44386/Identity/Account/Manage
Update
TS
About Us
How to Use?
Become our Delivery Partner!
Subscribe Here!
My Deliveries
Hello janikavya23@gmail.com!
Logout

Manage your account

Change your account settings

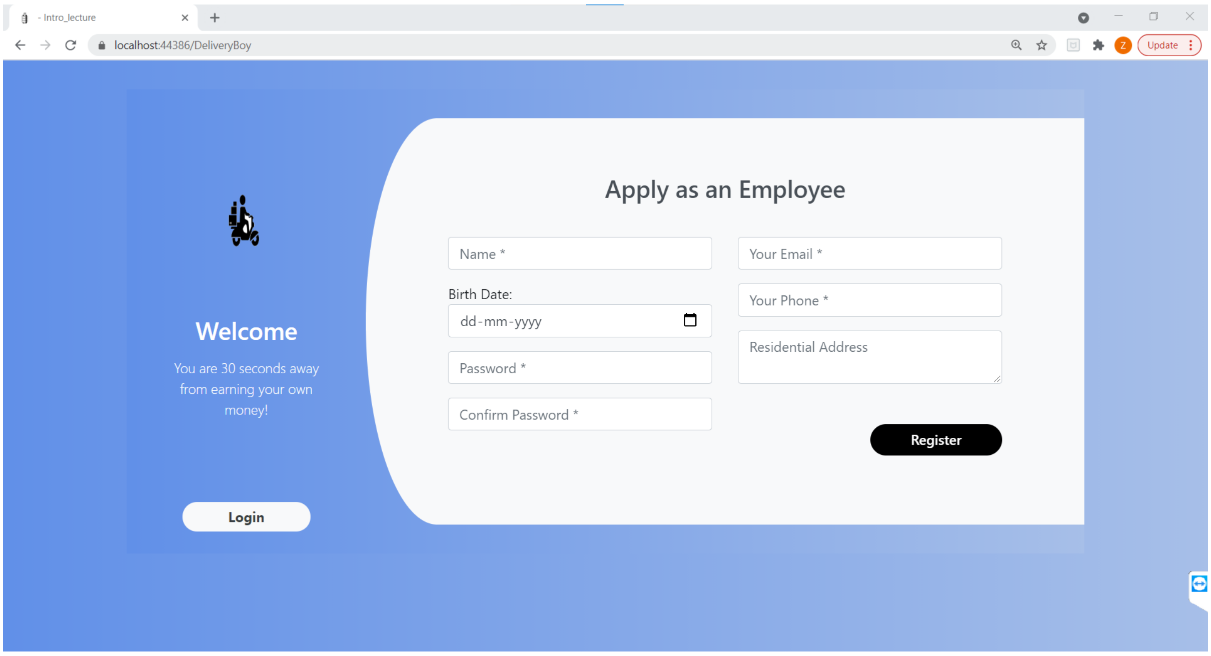
Profile

Username: janikavya23@gmail.com

Phone number:

© 2021 - Intro_lecture - Privacy

8. Delivery Boy Registration Page



Welcome

You are 30 seconds away from earning your own money!

Apply as an Employee

Name *

Your Email *

Birth Date: dd-mm-yyyy

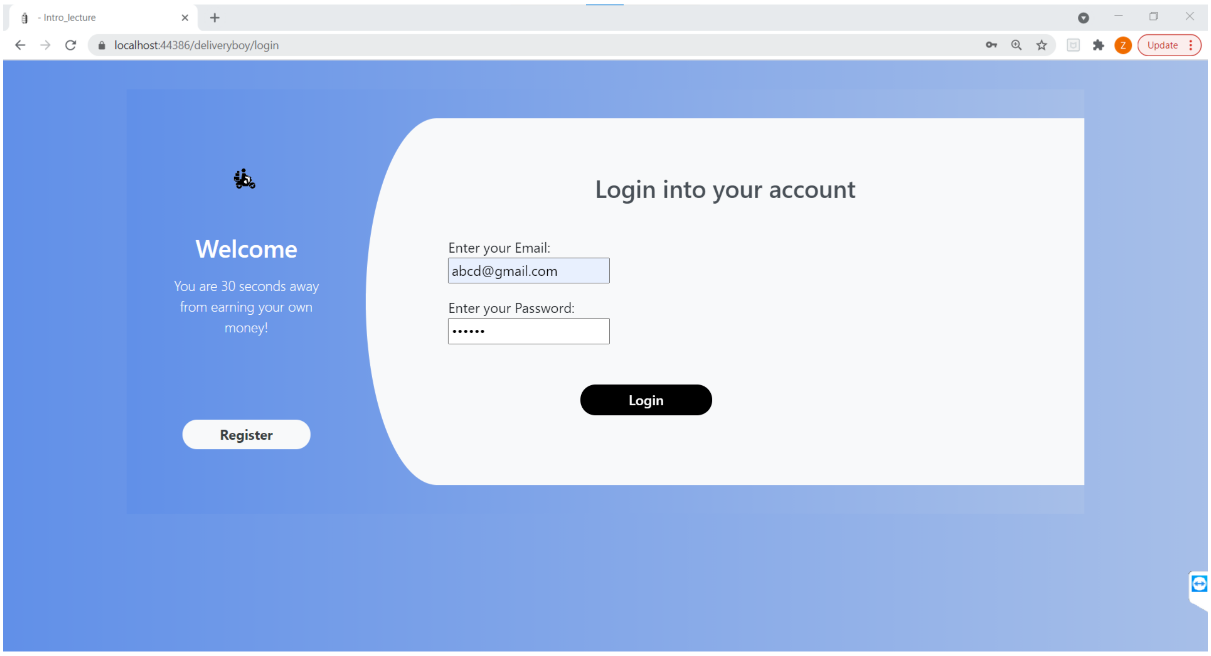
Your Phone *

Residential Address

Password *

Confirm Password *

9. Login Page for Delivery Boy



Welcome

You are 30 seconds away from earning your own money!

Register

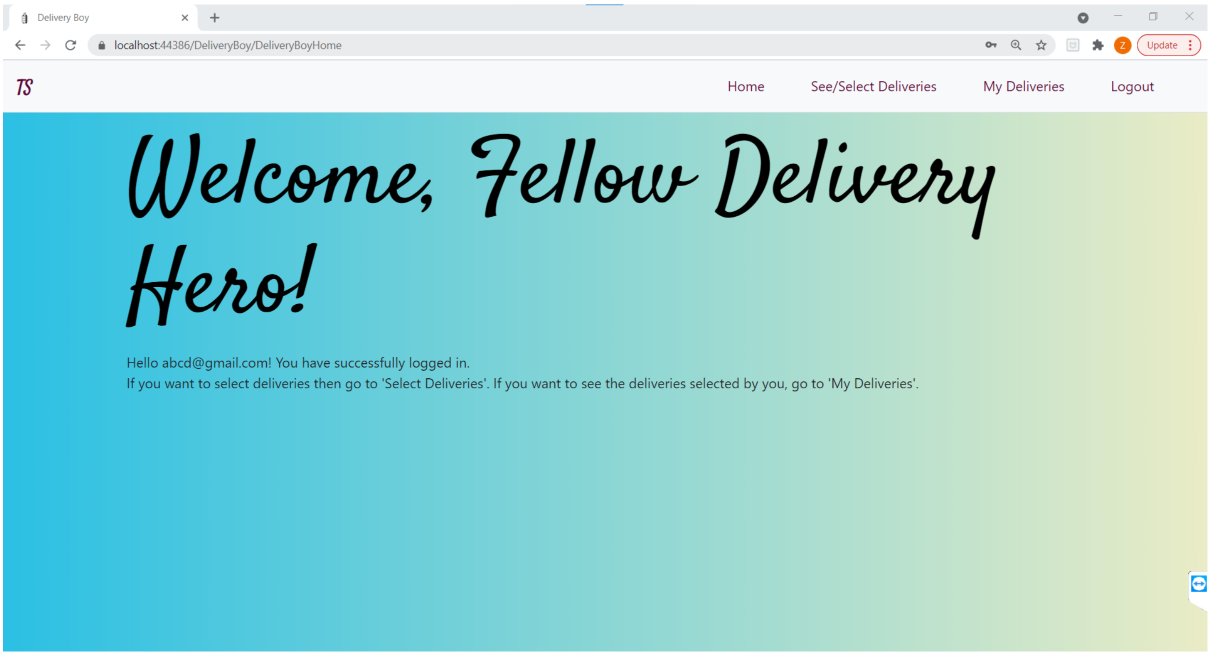
Login into your account

Enter your Email: abcd@gmail.com

Enter your Password: ••••••

Login

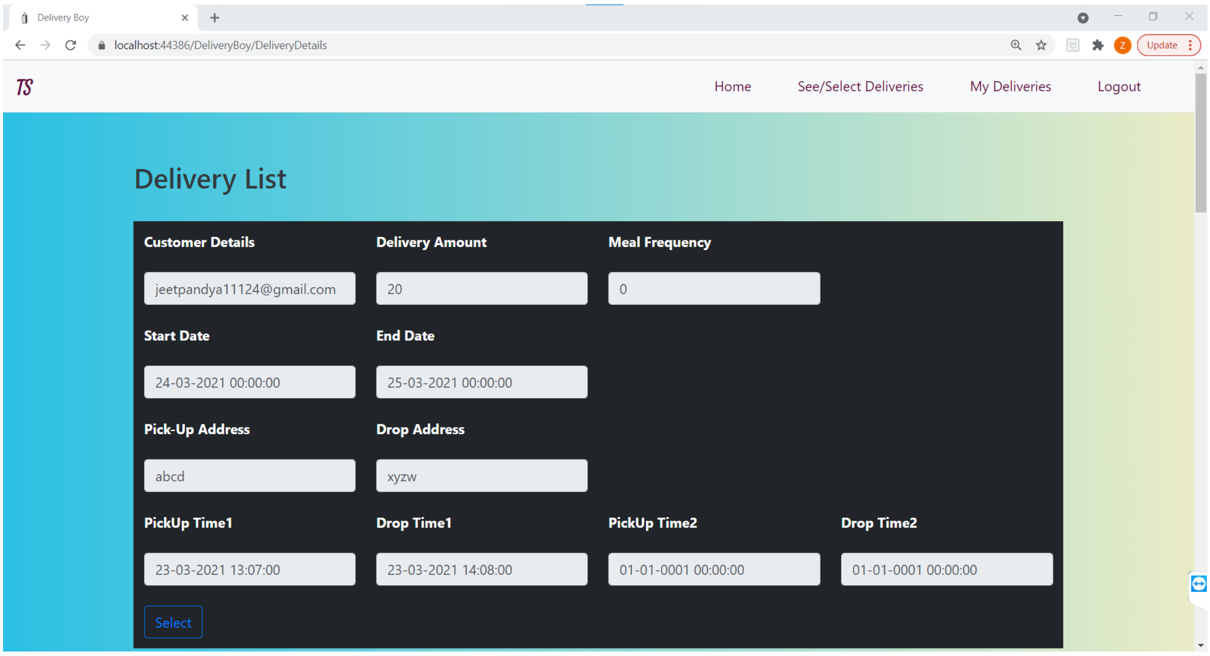
10. Delivery boy home page



Welcome, Fellow Delivery Hero!

Hello abcd@gmail.com! You have successfully logged in.
If you want to select deliveries then go to 'Select Deliveries'. If you want to see the deliveries selected by you, go to 'My Deliveries'.

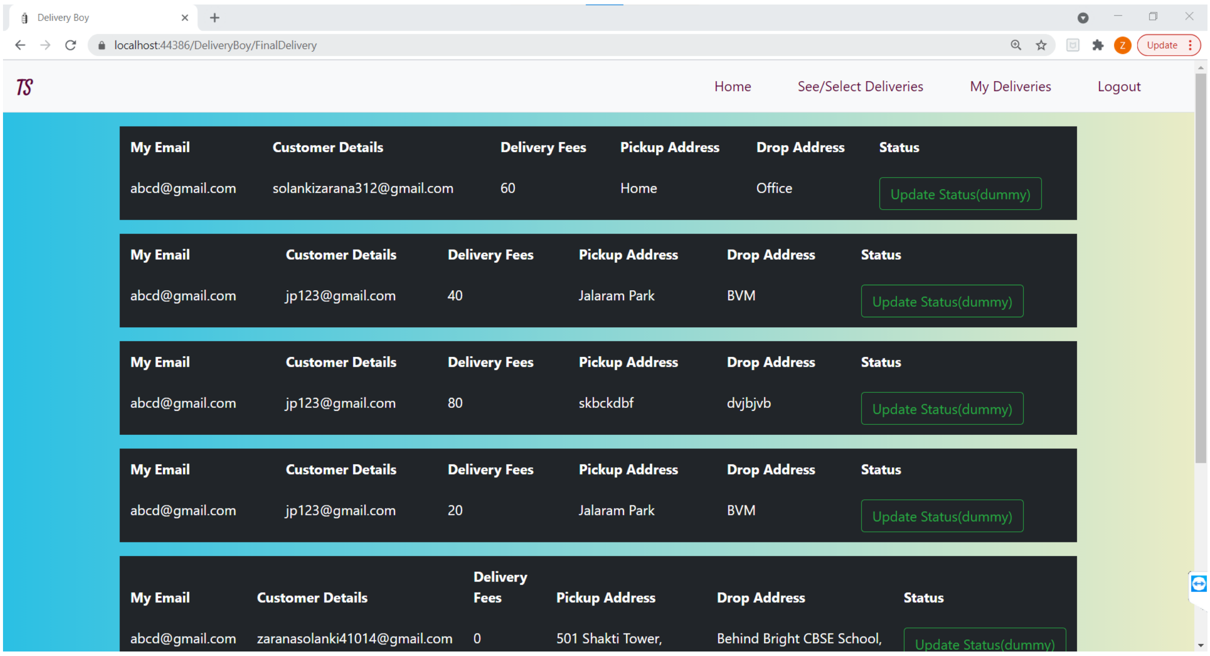
11. Pending Delivery List



Delivery List

Customer Details	Delivery Amount	Meal Frequency	
jeetpandya11124@gmail.com	20	0	
Start Date	End Date		
24-03-2021 00:00:00	25-03-2021 00:00:00		
Pick-Up Address	Drop Address		
abcd	xyzw		
PickUp Time1	Drop Time1	PickUp Time2	Drop Time2
23-03-2021 13:07:00	23-03-2021 14:08:00	01-01-0001 00:00:00	01-01-0001 00:00:00

12. Delivery Boy's Deliveries



My Email	Customer Details	Delivery Fees	Pickup Address	Drop Address	Status
abcd@gmail.com	solankizarana312@gmail.com	60	Home	Office	Update Status(dummy)

My Email	Customer Details	Delivery Fees	Pickup Address	Drop Address	Status
abcd@gmail.com	jp123@gmail.com	40	Jalaram Park	BVM	Update Status(dummy)

My Email	Customer Details	Delivery Fees	Pickup Address	Drop Address	Status
abcd@gmail.com	jp123@gmail.com	80	skbckdbf	dvjbjvb	Update Status(dummy)

My Email	Customer Details	Delivery Fees	Pickup Address	Drop Address	Status
abcd@gmail.com	jp123@gmail.com	20	Jalaram Park	BVM	Update Status(dummy)

My Email	Customer Details	Delivery Fees	Pickup Address	Drop Address	Status
abcd@gmail.com	zaranasolanki41014@gmail.com	0	501 Shakti Tower,	Behind Bright CBSE School,	Update Status(dummy)

Conclusion

TSWA is a website primarily designed for people wanting to eat home-made food in their workplace or schools, colleges. It is a food delivery application which will allow customers to get hygienic, home-made food. The system also allows a quick and easy-to-get subscription in just a few steps. Delivery partners then use these orders through an easy to navigate graphical interface for efficient processing. Hence, this application is really efficient and useful.

Future Scopes

The proposed system can be used by students, faculties as well as office going people to get home food. This project can be enhanced by adding real-time tracking of the food using Google Maps API and GPS. Also, this project could be made into an Android application for much easier use.

References

1. https://devblogs.microsoft.com/dotnet/introducing-net-5/
2. https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.tutorialsteacher.com%2FContent%2Fimages%2Fioc%2FDI.png&imgrefurl=https%3A%2F%2Fwww.tutorialsteacher.com%2Fioc%2Fdependency-injection&tbnid=5BOdcGvAN3sk2M&vet=12ahUKEwjj_ouW5IXwAhVHAnIKHcS8D64QMygRegUIARDUAQ..i&docid=Vzuvt0rPCEK7AM&w=369&h=198&q=dependency%20injection%20in%20mvc&ved=2ahUKEwjj_ouW5IXwAhVHAnIKHcS8D64QMygRegUIARDUAQ
3. https://www.youtube.com/watch?v=DqD-NJf7-OM