AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Computer Science and Enineering

CSE 3224

Information System Design and Software Engineering Lab

Spring 2020

Street Vendors

Lab Section:B1

Group:6

Submitted To

Dr. -Ing. Nusrat Jahan Lisa	Mr. Nibir Chandra Mandal
Assistant Professor	Lecturer
CSE, AUST	CSE, AUST

Submitted by

Mohammad Najrul Islam	170204061
Rahat Kader Khan	170204074
Shweta Bhattacharjee Porna	170204111

22 April, 2021

Contents

In this project, we use code first migration approach. We use entityframe work for migration in our database. So, there is no SQL written in the code. Our IdentityModel handle the connection part with our database. Every change in the database was done by migration.

1 Contribution of Mohammad Najrul Islam (ID-17.02.04.061)

1.1 APIs and Librarys

1.1.1 Maps JavaScript API

Brief: The Maps JavaScript API lets you customize maps with your own content and imagery for display on web pages and mobile devices.

Used for:

- Showing map in the project's "MAP" page.
- Showing real-time active individual Street Vendor's marker in the map.
- Showing the active Vendor's info in a small window on the marker.

Loading Maps JavaScript API

Listing 1: InitMap

```
<script async defer src=
    "https://maps.googleapis.com/maps/api/js?key=MyAPIKeyWasHere&
    callback=initMap"type="text/javascript"></script>
```

Functionality : I used an active API key in place of 'MyAPIKeyWasHere' and loaded the maps Javascript API shown in the upper script .

1.1.2 ASP.NET SignalR Hubs API

Brief: The SignalR Hubs API enables one to make remote procedure calls (RPCs) from a server to connected clients and from clients to the server. SignalR is also known as an open source library from Microsoft that allows ASP.NET developers to take advantage of the "WebSocket API". SignalR also falls back on other communication technologies in case WebSocket isn't available, which makes it even more useful when dealing with the wide variety of browsers in use today.

Used for:

- Real-time communication for real-time location updates of the street vendors.
- two-way interactive communication session between the user's browser and the server.
- persistent connection between a client and server.

Configuring SignalR

Listing 2: Startup.cs

```
public void Configuration(IAppBuilder app)
{
    app.MapSignalR();
}
```

Functionality : Every OWIN Application has a startup class. Here I specified SignalR for the application pipeline. I also had to install related NuGet packages before configuration.

1.1.3 Animations Using AOS Library

Used for: Animations in Welcome Page, MarketPlace Page etc. The 'data-aos' class was used from this library.

Configuring AOS Library

Listing 3:

```
<script src="https://unpkg.com/aos@2.3.1/dist/aos.js"></script>
```

This is the implemented 'Javascript CDN source'.

Listing 4:

```
<script>
    AOS.init();
</script>
```

Here AOS is initialized.

1.2 Server Side

For server side implementation C-sharp language was used. And this was implemented in MapHub and also used instance of the ApplicationDbContext.

1.2.1 Map Hub

Functionality : Fetches all the users info for various backend calculations using "ApplicationDbContext". Mainly used for fetching individual userId for connected users. Entity framework was used for this.

Listing 5:

```
public class MapHub : Hub { }
```

Functionality : Inherits all the Hub properties.

1.2.2 Users Info

Listing 6:

```
_context.Users.ToList();
```

Functionality : Fetches all the users info using "ApplicationDbContext" for various backend calculations. Mainly used for fetching individual userId for connected users. Entity framework was used for this.

1.2.3 On Connected

Listing 7:

```
public override System.Threading.Tasks.Task OnConnected() { }
```

Functionality : This is called when a user connects with the MapHub i.e. opens the map in our website.

1.2.4 Send Info to Server

Listing 8:

```
public void send(string location) { }
```

Functionality : Called by a user internally to send his location and information to the server.

1.2.5 To All Connected User

Listing 9:

```
Clients.All.user(string);
```

Functionality : Called by a user internally to send his information to other connected users through the server. As a result you can see the connected users list in the map page.

1.2.6 To Server for Caller

Listing 10:

```
Clients.Caller.message(string);
```

Functionality : Called by a user internally to send his location and information to the server for himself.

1.2.7 To Server for Others

Listing 11:

```
Clients.Others.message(string);
```

Functionality : Called by a user internally to send his location and information to the server for other connected users on that time.

1.3 Client Side

For client side implementation mainly Javascript was used. Besides, HTML, CSS and Bootstrap were used. And this was implemented in app/app.js and Views/Map/Index.cshtml.

1.3.1 Connecting to Map Hub

Listing 12:

```
$.connection.mapHub;
```

Functionality : Establishes the connection to the Map Hub.

1.3.2 Hub Connection Starts

Listing 13:

```
$.connection.hub.start(function () { })
```

Functionality : Starts communication with the hub.

1.3.3 Send Info to Hub

Listing 14:

```
function sLocation() { }
```

Functionality :

- Sends information of a user to the hub.
- This information is based on his location status i.e. whether user wants to broadcast his/her own location or not.
- Also if user's browser supports navigation or not.

1.3.4 Navigation

Listing 15:

```
navigator.geolocation.getCurrentPosition(success, failure);
```

Functionality : Navigates user's current location. If the navigation becomes successful then it runs a function called "success". Else it runs "failure" function.

1.3.5 Navigation Successful

Listing 16:

```
function success(object) { }
```

Functionality : Sends location co ordinates to the hub.

1.3.6 Navigation Fail

Listing 17:

```
function failure() { }
```

Functionality : Sends error messages to the hub.

1.3.7 Showing Connected Users

Listing 18:

```
hub.client.user = function (string) { }
```

Functionality :

- Called whenever a new user gets connected to the hub.
- Shows his/her and any new connected user/s name on the left of the map in the map page.

1.3.8 Showing Map

Listing 19:

```
function initMap() { });
```

Functionality : Shows the map in the page based on the properties defined in the function.

1.3.9 Hub to User

Listing 20:

```
hub.client.message = function (loc) { }
```

Functionality :

- Called everytime server gets a new information of any user.
- Shows user's own position via a special blue colored geo-location marker in the map.
- Shows others position via red colored geo-location marker in the map.

1.3.10 Showing Geo-location Markers

Listing 21:

```
new google.maps.Marker({ });
```

Functionality : Creates a new marker based on various criteria

- One for each user i.e. for his/her own position.
- There is generated markers for every connected users real-time position.

1.3.11 Reinitializing Center and Zoom of the Map

Listing 22:

```
map.setCenter(Coords);
map.setZoom(12);
```

Functionality :

- Centers the map based on the user's own location.
- Sets the zoom level of the map to 12.

1.3.12 Marker's Information Window

Listing 23:

```
new google.maps.InfoWindow({ });
```

Functionality : Initializes every marker's information window. Which shows info about that marker's user(here street vendors).

1.3.13 Hover on Marker

Listing 24:

```
Marker.addListener('mouseover', function () {
    infowindow.open(map, Marker);
});
```

Functionality : Hovering on the marker will open up it's corresponding information window.

1.3.14 Remove Hover on Marker

Listing 25:

```
Marker.addListener('mouseout', function () {
    infowindow.close();
});
```

Functionality : Removing cursor from the marker will close it's corresponding information window.

1.3.15 Click on Marker

Listing 26:

```
Marker.addListener("click", () => { });
```

Functionality : Clicking on the marker will take you to that corresponding marker's shop page.

1.3.16 Real-time Update for Markers Position

Listing 27:

```
Marker.setPosition(Coords);
```

Functionality : For all connected users' marker, this function updates the position in real-time. This update happens every five seconds for each of the markers.

1.4 Security Features

1.4.1 Permission

Our website asks for permission to access your location information when you enter in the map page. No one can see your location if you do not ALLOW our website to access your location information.

1.4.2 Instant Hideout/Showoff

If you allow and want to hide your location from others, you can do that just by deactivating the toggle button which you will find at the left side of the map. And nothing to worry, you can always show your location to others by activating that same toggle button.

1.4.3 No Database Storage for Location Information

For your ULTIMATE SECURITY, none of your location information is stored in any of the databases.

1.5 Deployment in Azure App Service

1.5.1 Deployment and Issues Faced

Our website is deployed in Azure App Service. Though some of features do not work for some "blob-storage" issues.

1.5.2 Fully Working Deployed Web Pages

Fully working pages in our deployed website are Registration, Login, Profile (except uploading profile picture), Map and Community.

1.5.3 Link and Suggestions

You can go here "https://streetvendors13.azurewebsites.net" and register/login. To watch the map and community parts' work you can ask others to register/login and join at that same time. If the link does not work please contact.

2 Contribution of Rahat Kader Khan(ID-17.02.04.074)

2.1 Model

2.1.1 IdentityModels

Listing 28: ApplicationUser

```
public class ApplicationUser : IdentityUser
{
    public string Type { get; set; }
    public string ImagePath { get; set; }
    public async Task<ClaimsIdentity> GenerateUserIdentityAsync
                              (UserManager<ApplicationUser>
                                 manager)
    {
        CookieAuthenticationOptions.AuthenticationType
        var userIdentity = await manager.CreateIdentityAsync(this,
                           DefaultAuthenticationTypes.ApplicationCookie);
        return userIdentity;
    }
}
```

Functionality : This class for Identity user. There is two additional properties that is added to the user table. This class handles the database connection for user data.

Listing 29: ApplicationDbContext

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext()
        : base("DefaultConnection", throwIfV1Schema: false)
    {
    }
    public static ApplicationDbContext Create()
    {
        return new ApplicationDbContext();
    }
    public System.Data.Entity.DbSet<Street_Vendors.Models.Item>
                                            Items { get; set; }
}
```

Functionality : This class connects all other table in database with the frontend. Every change in the database is done by migration. This class handle the migration in our database.

2.1.2 AccountViewModels

Listing 30: ExternalLoginConfirmationViewModel

```
public class ExternalLoginConfirmationViewModel
{
    [Required]
    [Display(Name = "Email")]
    public string Email { get; set; }
}
```

Listing 31: ExternalLoginListViewModel

```
public class ExternalLoginListViewModel
{
    public string ReturnUrl { get; set; }
}
```

Listing 32: SendCodeViewModel

```
public class SendCodeViewModel
{
    public string SelectedProvider { get; set; }
    public ICollection<System.Web.Mvc.SelectListItem> Providers { get;
        set; }
    public string ReturnUrl { get; set; }
    public bool RememberMe { get; set; }
}
```

Listing 33: VerifyCodeViewModel

```
public class VerifyCodeViewModel
{
    [Required]
    public string Provider { get; set; }

    [Required]
    [Display(Name = "Code")]
    public string Code { get; set; }
    public string ReturnUrl { get; set; }

    [Display(Name = "Remember this browser?")]
    public bool RememberBrowser { get; set; }

    public bool RememberMe { get; set; }
}
```

Listing 34: ForgotViewModel

```
public class ForgotViewModel
{
    [Required]
    [Display(Name = "Email")]
    public string Email { get; set; }
}
```

Listing 35: LoginViewModel

```
public class LoginViewModel
{
    [Required]
    [Display(Name = "Email")]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    public bool RememberMe { get; set; }
}
```

Listing 36: RegisterViewModel

```
public class RegisterViewModel
{
    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    public string Email { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2}
       characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation
       password do not match.")]
    public string ConfirmPassword { get; set; }
}
```

Listing 37: ResetPasswordViewModel

```
public class ResetPasswordViewModel
{
    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    public string Email { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2}
       characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation
       password do not match.")]
    public string ConfirmPassword { get; set; }

    public string Code { get; set; }
}
```

Listing 38: ForgotPasswordViewModel

```
public class ForgotPasswordViewModel
{
    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    public string Email { get; set; }
}
```

Functionality : Above model is for user account information. Each properties characteristics are specified by their above data annotations.

2.1.3 ManageViewModels

Listing 39: IndexViewModel

```
public class IndexViewModel
{
    public bool HasPassword { get; set; }
    public IList<UserLoginInfo> Logins { get; set; }
    public string PhoneNumber { get; set; }
    public bool TwoFactor { get; set; }
    public bool BrowserRemembered { get; set; }
}
```

Listing 40: ManageLoginsViewModel

```
public class ManageLoginsViewModel
{
    public IList<UserLoginInfo> CurrentLogins { get; set; }
    public IList<AuthenticationDescription> OtherLogins { get; set; }
}
```

Listing 41: FactorViewModel

```
public class FactorViewModel
{
    public string Purpose { get; set; }
}
```

Listing 42: SetPasswordViewModel

```
public class SetPasswordViewModel
{
    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2}
       characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "New password")]
    public string NewPassword { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm new password")]
    [Compare("NewPassword", ErrorMessage = "The new password and
       confirmation password do not match.")]
    public string ConfirmPassword { get; set; }
}
```

Listing 43: ChangePasswordViewModel

```
public class ChangePasswordViewModel
{
    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Current password")]
    public string OldPassword { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2}
       characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "New password")]
    public string NewPassword { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm new password")]
    [Compare("NewPassword", ErrorMessage = "The new password and
       confirmation password do not match.")]
    public string ConfirmPassword { get; set; }
}
```

Listing 44: AddPhoneNumberViewModel

```
public class AddPhoneNumberViewModel
{
    [Required]
    [Phone]
    [Display(Name = "Phone Number")]
    public string Number { get; set; }
}
```

Listing 45: VerifyPhoneNumberViewModel

```
public class VerifyPhoneNumberViewModel
{
    [Required]
    [Display(Name = "Code")]
    public string Code { get; set; }

    [Required]
    [Phone]
    [Display(Name = "Phone Number")]
    public string PhoneNumber { get; set; }
}
```

Listing 46: ConfigureTwoFactorViewModel

```
public class ConfigureTwoFactorViewModel
{
    public string SelectedProvider { get; set; }
    public ICollection<System.Web.Mvc.SelectListItem> Providers { get;
        set; }
}
```

Listing 47: ChangeSellerTypeViewModel

```
public class ChangeSellerTypeViewModel
{
    [Required]
    [Display(Name = "Type")]
    public string SellerType { get; set; }
}
```

Functionality : Above model is for managing the user account information. Each properties characteristics are specified by their above data annotations.

2.2 Controller

2.2.1 HomeController

Listing 48: Index

```
public ActionResult Index()
{
    return View();
}
```

Functionality : This function is for returning the default view of our home page.

2.2.2 AccountController

Listing 49: SignInManager

```
public ApplicationSignInManager SignInManager
{ }
```

Functionality : This function is for returning the signin information of the sign in user.

Listing 50: UserManager

```
public ApplicationUserManager UserManager
{ }
```

Functionality : This function is for returning the user information of the sign in user.

Listing 51: Login

```
[AllowAnonymous]
public ActionResult Login(string returnUrl)
{
    return View();
}

[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Login(LoginViewModel model,
                                     string returnUrl)
{
    return View(model);
}
```

textbfFunctionality : This function handles the login part of users.

Listing 52: Register

```
[AllowAnonymous]
public ActionResult Register()
{
    return View();
}

[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
    return View(model);
}
```

textbfFunctionality : This function handles the registration part of users.

Listing 53: ConfirmEmail

```
[AllowAnonymous]
public async Task<ActionResult> ConfirmEmail(string userId,
                                            string code)
{
    return View(result.Succeeded ? "ConfirmEmail" : "Error");
}
```

Functionality : This function handles the email confirmation part of users.

Listing 54: ForgotPassword

```
    [AllowAnonymous]
    public ActionResult ForgotPassword()
    {
        return View();
    }

    [HttpPost]
    [AllowAnonymous]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult> ForgotPassword(ForgotPasswordViewModel
       model)
    {
        return View(model);
    }
```

Functionality : This function handles the forgotPassword part during login.

Listing 55: ForgotPasswordConfirmation

```
    [AllowAnonymous]
    public ActionResult ForgotPasswordConfirmation()
    {
        return View();
    }
```

Functionality : This function handles the forgotPassword confirmation part during login.

Listing 56: ResetPassword

```
    [AllowAnonymous]
    public ActionResult ResetPassword(string code)
    {
        return code == null ? View("Error") : View();
    }

    [HttpPost]
    [AllowAnonymous]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult> ResetPassword(ResetPasswordViewModel
       model)
    {
        return View();
    }
```

Functionality : This function handles the reset password part of users.

Listing 57: ResetPasswordConfirmation

```
[AllowAnonymous]
public ActionResult ResetPasswordConfirmation()
{
    return View();
}
```

Functionality : This function handles the reset password confirmation part of users.

Listing 58: ExternalLogin

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public ActionResult ExternalLogin(string provider, string returnUrl)
{
    return new ChallengeResult(provider, Url.Action(
    "ExternalLoginCallback","Account", new { ReturnUrl = returnUrl }));
}
```

Functionality : This function handles the external login part of users.

Listing 59: ExternalLoginCallback

```
[AllowAnonymous]
public async Task<ActionResult> ExternalLoginCallback(string returnUrl)
{
    return View("ExternalLoginConfirmation", new
        ExternalLoginConfirmationViewModel { Email = loginInfo.Email
           });
}
```

Functionality : This function also handles the external login part of users.

Listing 60: ExternalLoginConfirmation

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> ExternalLoginConfirmation
        (ExternalLoginConfirmationViewModel model, string returnUrl)
{
    return View(model);
}
```

Functionality : This function handles the external login confirmation part of users.

Listing 61: LogOff

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult LogOff()
{
    AuthenticationManager.SignOut(DefaultAuthenticationTypes
                                                    .ApplicationCookie);
    return RedirectToAction("Index", "Home");
}
```

Functionality : This function handles the LogOff part of users.

Listing 62: ExternalLoginFailure

```
[AllowAnonymous]
public ActionResult ExternalLoginFailure()
{
    return View();
}
```

Functionality : This function handles the error part for external login.

Listing 63: Dispose

```
protected override void Dispose(bool disposing)
{ }
```

Functionality : Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources

Listing 64: AuthenticationManager

```
private IAuthenticationManager AuthenticationManager
{
    get
    {
        return HttpContext.GetOwinContext().Authentication;
    }
}
```

Functionality : This function handles the authentication of users.

Listing 65: AddErrors

```
private void AddErrors(IdentityResult result)
{ }
```

Functionality : This function is for showing error massage.

Listing 66: RedirectToLocal

```
private ActionResult RedirectToLocal(string returnUrl)
{
    return RedirectToAction("Index", "MarketPlace");
}
```

Functionality : This function is for redirecting the users to the market place after a successfull login.

2.2.3 ManageController

Listing 67: SignInManager

```
public ApplicationSignInManager SignInManager
{ }
```

Functionality : This function is for returning the signin information of the sign in user.

Listing 68: UserManager

```
public ApplicationUserManager UserManager
{ }
```

Functionality : This function is for returning the user information of the sign in user.

Listing 69: Index

```
public async Task<ActionResult> Index(ManageMessageId? message)
{
    return View(user);
}
```

Functionality : This function is for returning the default view of profile setting.

Listing 70: RemoveLogin

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> RemoveLogin(string loginProvider,
                                            string providerKey)
{
    return RedirectToAction("ManageLogins", new { Message = message });
}
```

Functionality : This function is for remove external login.

Listing 71: AddPhoneNumber

```
public ActionResult AddPhoneNumber()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> AddPhoneNumber(AddPhoneNumberViewModel
                                                                model)
{
    return RedirectToAction("Index");
}
```

Functionality : This function is for adding phone number.

Listing 72: RemovePhoneNumber

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> RemovePhoneNumber()
{
    return RedirectToAction("Index", new { Message =
                                  ManageMessageId.RemovePhoneSuccess });
}
```

Functionality : This function is for removing phone number.

Listing 73: ChangePassword

```
public ActionResult ChangePassword()
{
    return View();
}
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> ChangePassword(ChangePasswordViewModel
                                                                model)
{
    return View(model);
}
```

Functionality : This function is for changing the password.

Listing 74: SetPassword

```
public ActionResult SetPassword()
{
    return View();
}
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> SetPassword(SetPasswordViewModel model)
{
    return View(model);
}
```

Functionality : This function is for seting password for external login.

Listing 75: ManageLogins

```
public async Task<ActionResult> ManageLogins(ManageMessageId? message)
{
    return View(new ManageLoginsViewModel
    {
        CurrentLogins = userLogins,
        OtherLogins = otherLogins
    });
}
```

Functionality : This function is for external login.

Listing 76: LinkLogin

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult LinkLogin(string provider)
{
    current user
    return new AccountController.ChallengeResult(provider, Url.Action
        ("LinkLoginCallback", "Manage"), User.Identity.GetUserId());
}
```

Functionality : This function is for multiple login.

Listing 77: LinkLoginCallback

```
public async Task<ActionResult> LinkLoginCallback()
{
    return result.Succeeded ? RedirectToAction("ManageLogins") :
       RedirectToAction("ManageLogins", new { Message =
       ManageMessageId.Error });
}
```

Functionality : This function is for link login callback.

Listing 78: Dispose

```
protected override void Dispose(bool disposing)
{ }
```

Functionality : Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources

Listing 79: EditPic

```
public ActionResult EditPic()
{
    return View();
}
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditPic(HttpPostedFileBase ImageFile)
{
    return View();
}
```

Functionality : This function is for editing profile picture.

Listing 80: SellerType

```
public ActionResult SellerType()
{
    return View();
}
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult SellerType(ChangeSellerTypeViewModel model)
{
    return RedirectToAction("Index");
}
```

Functionality : This function is for editing seller type.

2.2.4 MarketPlaceController

Listing 81: Index

```
public ActionResult Index()
{
    return View();
}
```

Functionality : This function is for returning the default view of market place.

Listing 82: Details

```
public ActionResult Details(int? id)
{
    return View(item);
}
```

Functionality : This function is for viewing the details of each item.

Listing 83: ProfileDetails

```
public ActionResult ProfileDetails(string id)
{
    return View(user);
}
```

Functionality : This function is viewing the seller profile.

Listing 84: likeItem

```
public ActionResult likeItem(int? id)
{
    return RedirectToAction("Details", new { id });
}
```

Functionality : This function handle the item like part.

Listing 85: dislikeItem

```
public ActionResult dislikeItem(int? id)
{
    return RedirectToAction("Details", new { id });
}
```

Functionality : This function handle the item dislike part.

2.2.5 CommunityController

Listing 86: Index

```
public ActionResult Index()
{
    return View();
}
```

Functionality : This function is for returning the default view of community.

2.3 View

2.3.1 Home

Home has only on default view that is:

1. Index.cshtml

2.3.2 Account

Account has 8 view in total and 1 partial view. Below views is for account setup.

1. ExtternalLoginsListPartial.cshtml
2. ConfirmEmail.cshtml
3. ExternalLoginConfirmation.cshtml
4. ForgotPassword.cshtml
5. ForgotPasswordConfirmation.cshtml
6. Login.cshtml
7. Register.cshtml
8. ResetPassword.cshtml
9. ResetPasswordConfimation.cshtml

2.3.3 Manage

Manage has total 7 view. Those view is for profile settings.

1. AddPhoneNumber.cshtml
2. ChangePassword.cshtml
3. EditPic.cshtml
4. Index.cshtml
5. ManageLogins.cshtml
6. SellerType.cshtml
7. SetPassword.cshtml
8. VerifyPhoneNumber.cshtml

2.3.4 MarketPlace

Market place has total 3 view. Those view is shows all page of market place.

1. Details.cshtml
2. Index.cshtml
3. ProfileDetails.cshtml

2.3.5 Community

Community has only on default view that is:

1. Index.cshtml

2.3.6 Layout

Below view handle the layout for all other view. We use two layout view for our project. They are:

1. Layout.cshtml
2. LayoutHome.cshtml

Layout.cshtml is the layout for home page and LayoutHome.cshtml is layout for all the other pages.
We use three navbar layout for our project. They are:

1. navHome.cshtml
2. navWelcome.cshtml
3. LoginPartial.cshtml

navWelcome.cshtml is the navbar for home page and navHome.cshtml is the layout for all the other pages. LoginPartial.cshtml is a partial view for those two navbar.

2.4 API

Listing 87: ConfigureAuth

```
public void ConfigureAuth(IAppBuilder app)
{
    app.CreatePerOwinContext(ApplicationDbContext.Create);
    app.CreatePerOwinContext<ApplicationUserManager>
                                (ApplicationUserManager.Create);
    app.CreatePerOwinContext<ApplicationSignInManager>
                                (ApplicationSignInManager.Create);

    app.UseCookieAuthentication(new CookieAuthenticationOptions
    {
        AuthenticationType=DefaultAuthenticationTypes.ApplicationCookie,
        LoginPath = new PathString("/Account/Login"),
        Provider = new CookieAuthenticationProvider
        {
            OnValidateIdentity
               =SecurityStampValidator.OnValidateIdentity
                           <ApplicationUserManager, ApplicationUser>(
                validateInterval: TimeSpan.FromMinutes(30),
                    regenerateIdentity: (manager, user) =>
                       user.GenerateUserIdentityAsync(manager))
        }
    });
    app.UseExternalSignInCookie
                           (DefaultAuthenticationTypes.ExternalCookie);
    app.UseTwoFactorSignInCookie(DefaultAuthenticationTypes
                           .TwoFactorCookie, TimeSpan.FromMinutes(5));

    app.UseTwoFactorRememberBrowserCookie(DefaultAuthenticationTypes
                                     .TwoFactorRememberBrowserCookie);

    app.UseFacebookAuthentication(
          appId: "286553600705031l",
          appSecret: "3b42311f510b9830d0c5e638386f2f71");

    app.UseGoogleAuthentication(new GoogleOAuth2AuthenticationOptions()
    {
        ClientId = "1094704316554-hphdrve2t35e4jr936c50e92bl009gql
                                      .apps.googleusercontent.com",
        ClientSecret = "IljNjawPD5VC4WJXY9T3qE57"
    });
}
```

Functionality : In the abobe fuction i configure the db context, user manager and signin manager to use a single instance per request. Then enable the application to use a cookie to store information for the signed in user and to use a cookie to temporarily store information about a user logging in with a third party login provider and also configure the sign in cookie. Lastly i use facebook and google api for external login.

2.5 ChatHub

We use SignalR for our real time chat. Below two function is for showing connected user and sending messages.

Listing 88: OnConnected

```
public override System.Threading.Tasks.Task OnConnected()
{
    Clients.All.user(Context.User.Identity.Name);
    return base.OnConnected();
}
```

Functionality : This function show all the user active in real time.

Listing 89: send

```
public void send(string message)
{
    Clients.Caller.message("You~You: " + message);
    Clients.Others.message("others~" + Context.User.Identity.Name + ":
       " + message);
}
```

Functionality : This function is for sending message to the community.

3 Contribution of Shweta Bhattacharjee(ID-17.02.04.111)

The Classes and the Methods under the classes are documented below.

3.1 Model

Listing 90: Item

```
public class Item
  {
      public int ID { get; set; }
      public string ItemName { get; set; }
      public string ItemDescription { get; set; }
      public int ItemPrice { get; set; }
      public string ItemImage { get; set; }
      public int ItemRating { get; set; }
      public string UserId { get; set; }
      [ForeignKey("UserId")]
      public ApplicationUser ApplicationUser { get; set; }
  }
```

Used For : Used for showing item details in myShop from database,create new item,edit or delete it.

Functionality : Above properties belongs to the items. Each properties characteristics are specified by their above data annotations. Item controller includes:

- Id:This get set method is used for Item Id.
- ItemName:This method is used for item Name in Database
- ItemDescription:This method is used for Item Description
- ItemPrice: This includes Item Price
- ItemImage: Image path will store in Database
- ItemRating : This method is used for Rating
- UserId : UserId is using as Foreignkey
- Application users

There is one primary key and one Foreign Key Item controller includes:

- Primary Key: ID
- Foreign Key: UserId

s

3.2 Controller

Below given methods belong to the **ItemsController** that controls our item properties.

3.2.1 Index

Listing 91: Index

```
public ActionResult Index()
{
   return View(items.ToList());
}
```

Used For : Showing item details in myShop from database like ItemName,description,Price etc.

Functionality : This returns the default view of items and the list of items.

3.2.2 Details

Listing 92: Details

```
public ActionResult Details(int? id)
{
    return View(item);
}
```

Used For : Showing item details from database.

Functionality : It takes ItemId as a parameter and return it's details.

3.2.3 Create

Listing 93: Create

```
 public ActionResult Create()
{
   return View();
}
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult
    Create([Bind(Include="ID,ItemName,ItemDescription,ItemPrice,ItemImage,
                     ItemRating,UserId")] Item item, HttpPostedFileBase
                         ImageFile)

{
   return View(item);
}
```

Used For : Creating new items in database.

Functionality : Here we created overloading methods.The method without parameter returns default Create view.On HttpPost of default Create view the method with parameter send request to the server for new item with the help of ValidateAntiForgeryToken.On successful execution this method return default item view.

3.2.4 Edit

Listing 94: Edit

```
public ActionResult Edit(int? id)
{
    return View(item);
}
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult
   Edit([Bind(Include="ID,ItemName,ItemDescription,ItemPrice,ItemImage,
                        ItemRating,UserId")] Item item, HttpPostedFileBase
                            ImageFile)
```

Used For : Editing items in database.

Functionality : There are two overloading methods.The method without parameter returns default Edit view.On HttpPost of default Edit view the method with parameter send request to the server for editing existing item with the help of ValidateAntiForgeryToken.On successful execution this method return default item view.

3.2.5 Delete

Listing 95: Delete

```
public ActionResult Delete(int? id)
{
    return View(item);
}
```

Used For : Taking ItemId from database and showing details of Items before confirming delete.We can cancel or confirm delete from here.

Functionality : It takes ItemId as a parameter and return it's details with functionality of deleting the item.

3.2.6 DeleteConfirmed

Listing 96: DeleteConfirmed

```
public ActionResult DeleteConfirmed(int id)
{
    return View(item);
}
```

Used For : Deleting items from database.

Functionality : It takes ItemId as a parameter and on HttpPost it sends request to server to delete the item with the help of ValidateAntiForgeryToken.

3.2.7 dispose

Listing 97: dispose

```
protected override void Dispose(bool disposing)
```

Functionality : Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources

3.3 View:

There are five default view of items.They are:

1. Index.cshtml
   - Index.cshtml display all the items from server of each individual account.
2. Create.cshtml
   - Create.cshtml create items with their name,price,details and item image and saves it to database
3. Edit.cshtml
   - edit items information and image and save changes database.
4. Details.cshtml
   - shows the details of each item from database.
5. Delete.cshtml
   - it takes item id from database and shows details to delete it

3.4 Additional Activity:

3.4.1 .bubble-boxes

Functionality : This css class controls the background animation of customize shapes.This shapes of animation is fully customize for our project.

3.4.2 .star-boxes

Functionality : We used some animated star shapes.This shapes of animation is fully customize for our project.

3.4.3 Animations

Animated Create,Delete,Back buttons were used.Used AOS(Animate on scroll) for animating the cards of MyShop,and MarketPlace and Profile page.Animated texts were used also.Used customize css for changing coloured background.Used svg wave generator for details page.Animations with different background and AOS(Animate on scroll) library were used in manage Profile page.