Project Description :

Socioboard is an advanced, easy, versatile, customizable and scalable open-source social media marketing tool to engage your audience with great features such as advanced scheduling and publishing tools, prompt news feeds, interactive social discovery, sophisticated analytics, social CRM, help desk integration, collaboration tools, customizable reports and much more. Unlike other social media management tools, **socioboard** serves your customization needs like specific editorial, workflow, governance etc. and allows you to listen to and engage your audience in so many different ways across numerous social media profiles from one simple platform.

Socioboard offers you an open-source Community version and paid Enterprise and SaaS versions with unmatched world-class technical support round the clock. It makes it easy for you to keep tabs on social conversations and provides everything you need to build your brand further by efficiently engaging your audience on one platform.

Technology :

Socioboard is created using ASP.NET, MVC 4.0, and JQuery with back end as MySQL.

APIs to be used to retrieve or post the data to social networking sites would be:

- Twitter - REST API v1.1 – Wrapper Written in C#.net
- Facebook – Graph API – Facebook C# SDK
- linkedin – REST API - Wrapper Written in C#.net
- Google Plus – HTTP API - Wrapper Written in C#.net
- Instagram's API - Wrapper Written in C#.net

Structure of SocioBoard-

> Api.Socioboard(Individual MVC4(WEB API) Project in the Solution)

> Domain.Socioboard(Console Application)

>Lib-

- GlobusGooglePlusLib(Console Application)
- GlobusInstagramLib(Console Application)
- GlobusLinkedinLib(Console Application)
- GlobusMailLib(Console Application)
- GlobusMultilanguageLib(Console Application)
- GLobussDropboxLib(Console Application)
- GlobusTumblrLib(Console Application)
- GlobusTwiterLib(Console Application)
- GLobusWordpressLib(Console Application)

>Socioboard(Another Individual MVC4(WEB API) project in the solution)

>SocioboardDataServices

1. Api.Socioboard

Api.Services provides the service to connect with database to access data and also make the data access platform independent.
To make this service platform independent we have made this as a separate project.

We have used NHibernate ORM for Data access from Mysql DataBase . Which require-
1.Domain(To Declare all properties)
2.Mapping(To map tables with with domain classes)
3.Helper(For SessionFactory)
4.Hibernate.cfg.xml(for configuration and give the connection string)
5.Models(contains methods to access with database)

1.1 SERVICES

All Services we use is present inside Services folder. These are –
>> Admin.asmx
>> AdminNews.asmx
>> AdminPackageDetails.asmx
>> AdminUserDetails.asmx
>> Ads.asmx
>> Affiliates.asmx
>> ArchiveMessage.asmx
>> Blog_Comments.asmx.cs
>> Blog_Posts.asmx.cs
>> BusinessSetting.asmx.cs
>> CompanyDashboard.asmx.cs
>> Companypage.asmx.cs
>> Coupon.asmx.cs
>> DiscoverySearch.asmx.cs
>> Drafts.asmx.cs
>> Dropbox.asmx.cs
>> DropboxAccount.asmx.cs
>> Facebook.asmx.cs
>> FacebookAccount.asmx.cs
>> FacebookFanPage.asmx.cs
>> FacebookFeed.asmx.cs
>> FacebookInsightPostStats.asmx.cs
>> FacebookInsightStats.asmx.cs
>> FacebookMessage.asmx.cs
>> FacebookStats.asmx

>> FbPageComment.asmx
>> FbPageLiker.asmx
>> FbPagePost.asmx
>> GoogleAnalyticsAccount.asmx
>> GoogleAnalyticsStats.asmx
>> GooglePlusAccount.asmx
>> GooglePlusActivities.asmx
>> GroupProfile.asmx
>> Groups.asmx
>> GroupScheduleMessage.asmx
>> Instagram.asmx
>> InstagramAccount.asmx
>> InstagramComment.asmx
>> InstagramFeed.asmx
>> Invitation.asmx
>> Linkedin.asmx
>> LinkedinAccount.asmx
>> LinkedinCompanyPage.asmx
>> LinkedInFeed.asmx
>> LinkedinMessage.asmx
>> Log.asmx
>> LoginLogs.asmx
>> MailSender.asmx
>> Messages.asmx
>> News.asmx
>> NewsLetter.asmx
>> Package.asmx
>> PaymentTransaction.asmx
>> RssFeeds.asmx
>>RssReader.asmx
>>ScheduledMessage.asmx
>>SentimentalAnalysis.asmx
>>SocialProfile.asmx
>>SocialProfiles.asmx
>>TaskComment.asmx
>>Tasks.asmx
>>Team.asmx
>>TeamMemberProfile.asmx
>>TicketAssigneeStatus.asmx
>>Tumblr.asmx
>>TumblrAccount.asmx
>>TumblrFeed.asmx
>>Twitter.asmx
>>TwitterAccount.asmx

>>TwitterAccountFollowers.asmx
>>TwitterDirectMessages.asmx
>>TwitterFeed.asmx
>>TwitterMessage.asmx
>>TwitterStats.asmx
>>User.asmx
>>UserActivation.asmx
>>UserPackageRelation.asmx
>>UserRefRelation.asmx
>>Wordpress.asmx
>>WordpressAccount.asmx
>>WordpressFeeds.asmx
>>WordpressSites.asmx
>>Youtube.asmx
>>YoutubeAccount.asmx
>>YoutubeChannel.asmx

These services are used in socioboard as these services has references as a web reference in socioboard project. We will discuss about the required service when it will be used in the socioboard project.

1.2 Mappings

This Folder contains all the Hibernate mapping files with the extension **hbm.xml**
An Object/relational mappings are usually defined in an XML document. This mapping file instructs Hibernate how to map the defined class or classes to the database tables.
Some of the Example of these files are:-
Admin.hbm.xml
Facebookfeed.hbm.xml
TwitterFeed.hbm.xml

1.3 Model

A *Model* , which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface. The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

It contains Repository classes that has Functions which contains the actual NHibernate queries and sessions that is invariably used to Insert or fetch data from the tables of the Database.

FOR EXAMPLE

```
public static void Update(Domain.Socioboard.Domain.Admin user)
        {
            //Creates a database connection and opens up a session
            using (NHibernate.ISession session = SessionFactory.GetNewSession())
            {
                //After Session creation, start Transaction.
                using (NHibernate.ITransaction transaction = session.BeginTransaction())
                {
                    try
                    {
                        // Proceed Sction to update Data.
                        // And Set the reuired paremeters to find the specific values.
                        int i = session.CreateQuery("Update Admin set Image =:profileurl, 
UserName =: username , TimeZone=:timezone,FirstName =:first,LastName =:last  where Id = 
:twtuserid")
                                  .SetParameter("twtuserid", user.Id)
                                  .SetParameter("profileurl", user.Image)
                                  .SetParameter("username", user.UserName)
                                  .SetParameter("timezone", user.TimeZone)
                                  .SetParameter("first",user.FirstName)
                                  .SetParameter("last",user.LastName)
                                  .ExecuteUpdate();
                        transaction.Commit();
                    }
                    catch (Exception ex)
                    {
                        Console.WriteLine(ex.StackTrace);

                    }
                }//End Using trasaction
            }//End Using session
                   }
```

1.4 NHIBERNATE FILE(Nhibernate.cfg.xml)

Hibernate requires to know in advance where to find the mapping information that defines how your classes relate to the database tables. Hibernate also requires a set of configuration settings related to database and other related parameters. All such information is usually supplied as a standard properties file called **hibernate.properties**, or as an XML file named **hibernate.cfg.xml.**

1.4.1 NHibernate PROPERTIES

```
<?xml version=”1.0” encoding=”utf-8” ?>
<hibernate-configuration xmlns=”urn:nhibernate-configuration-2.2”>
  <session-factory>
    <property
name=”connection.provider”>Nhibernate.Connection.DriverConnectionProvider</property>
    <property name=”dialect”>Nhibernate.Dialect.MySQLDialect</property>
    <property name=”connection.driver_class”>Nhibernate.Driver.MySqlDataDriver</property>
    <property name=”connection.connection_string”>
      Data Source=XXXX;User ID=XXXX;Password=XXXX;persist security info=False;initial
catalog=XXXX;pooling=false; pooling=false;charset=utf8;

    </property>
    <property name=”show_sql”>true</property>
  </session-factory>


</hibernate-configuration>
```

2. Domain.Socioboard

2.1 Domain

It consist all properties classes required in the socioboard project. All socioboard classes in the Domain folder are as given -

>AddFacebookPage.cs
>Admin.cs
>Ads.cs
>Affiliates.cs
>ArchiveMessage.cs
>Blog_Comments.cs
>Blog_Posts.cs
>BusinessSetting.cs
>CompanyProfiles.cs
>CompanySocialProfiles.cs
>Coupon.cs
>DiscoverySearch.cs
>Drafts.cs
>DropboxAccount.cs
>FacebookAccount.cs
>FacebookFanPage.cs
>FacebookFeed.cs
>FacebookGroup.cs
>FacebookGroupData.cs
>FacebookInsightPostStats.cs
>FacebookInsightStats.cs
>FacebookMessage.cs
>FacebookStats.cs
>FbPageComment.cs
>FbPageLiker.cs
>FbPagePost.cs
>FbPagePostCommentLiker.cs
>FbPageSharer.cs
>FBTwitterFeeds.cs
>FeedSentimentalAnalysis.cs
>GoogleAnalyticsAccount.cs
>GoogleAnalyticsStats.cs
>GooglePlusAccount.cs
>GooglePlusActivities.cs
>GroupProfile.cs

>Groups.cs
>GroupScheduleMessage.cs
>GroupStatDetails.cs
>IAdsRepository.cs
>IArchiveMessageRepository.cs
>IBlog_CommentsRepository.cs
>IBlog_PostsRepository.cs
>IBusinessSettingRepository.cs
>IDiscoverySearchRepository.cs
>IDraftsRepository.cs
>IFacebbookInsightStatsRepository.cs
>IFacebookAccountRepository.cs
>IFacebookFeedRepository.cs
>IFacebookInsightPostStats.cs
>IFacebookMessageRepository.cs
>IFacebookStatsRepository.cs
>IGoogleAnalyticsAccountRepository.cs
>IGoogleAnalyticsStatsRepository.cs
>IGooglePlusAccountRepository.cs
>IGooglePlusActivitiesRepository.cs
>IGroupProfilesRepository.cs
>IGroupRepository.cs
>IInstagramAccountRepository.cs
>IInstagramComment.cs
>IInstagramFeed.cs
>ILinkedInAccountRepository.cs
>ILinkedInFeedRepository.cs
>ILogException.cs
>INewsLetterRepository.cs
>INewsRepository.cs
>InstagramAccount.cs
>InstagramComment.cs
>InstagramFeed.cs
>Invitation.cs
>IPackageRepository.cs
>IRssFeedsRepository.cs
>IRssReader.cs
>IScheduledMessageRepository.cs
>ISocialProfilesRepository.cs
>ISocialSite.cs
>ISocialSiteAccount.cs
>ISocialSiteFeed.cs
>ISocialSiteMessage.cs
>ITaskCommentRepository.cs

>ITaskRepository.cs
>ITeamMemberProfileRepository.cs
>ITeamRepository.cs
>ITwitterAccountRepository.cs
>ITwitterDirectMessagesRepository.cs
>ITwitterFeedRepository.cs
>ITwitterMessageRepository.cs
>ITwitterStatsRepository.cs
>IUserPackageRelation.cs
>IUserRepository.cs
>LinkedIn_Update_Messages.cs
>LinkedinAccount.cs
>LinkedinCompanyPage.cs
>LinkedinCompanyPagePosts.cs
>LinkedInFeed.cs
>LinkedInGroup.cs
>LinkedInMessage.cs
>LinkedinPageComment.cs
>LinkedInUser.cs
>Log.cs
>LoginLogs.cs
>MandrillAccount.cs
>Messages.cs
>News.cs
>NewsLetter.cs
>Package.cs
>PaymentTransaction.cs
>ProjectDetails.cs
>ReplyMessage.cs
>RssFeeds.cs
>RssReader.cs
>ScheduledMessage.cs
>SocialProfile.cs
>TaskComment.cs
>Tasks.cs
>Team.cs
>TeamMemberProfile.cs
>TicketAssigneeStatus.cs
>TumblerData.cs
>TumblrAccount.cs
>TumblrFeed.cs
>TwitterAccount.cs
>TwitterAccountFollowers.cs
>TwitterDirectMessages.cs

>TwitterEngagement.cs
>TwitterFeed.cs
>TwitterMessage.cs
>TwitterStats.cs
>TwitterStatsReport.cs
>User.cs
>UserActivation.cs
>UserPackageRelation.cs
>UserRefRelation.cs
>WordpressAccount.cs
>WordpressFeeds.cs
>WordpressSites.cs
>YoutubeAccount.cs
>YoutubeChannel.cs

These classes are having their respective properties.

3. SocioBoard

The entire solution contains 2 Projects one is **API.SocioBoard** which has already been mentioned , the other one being **SocioBoard** which almost has all the main coding structure of the socioboard , technically having all the **Controllers** and **Views** and Most Importantly **Web Services**.

3.1 Controllers

A **controller** can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document).

SocioBoard has these many Controllers:-

AccountController
BusinessSettingController
CompanydashboardController
DiscoveryController
DropBoxController
FacebookController
FacebookManagerController
FeedsController
ForgotPasswordController
GroupController
Home Controller
IndexController
InstagramManagerController
LinkedinManagerController
MessageController
MultiLanguageController
MyStuffController
PersonalsettingController
PublishingController
ReportController
SocialManagerController
TumblrManagerController
TwitterManagerController
UserGroupController
VersionsController
YoutubeManagerController

3.2 VIEWS

A **Views** requests information from the model that it uses to generate an output representation to the user. The view manages the display of information.

The **Views** folder stores the files (HTML files) related to the display of the application (the user interfaces). These files may have the extensions html, asp, aspx, cshtml, and vbhtml, depending on the language content.

The Home folder is used for storing application pages like the home page and the about page.

The Shared folder is used to store views shared between controllers (master pages and layout pages).

SOCIOBOARD USES ASP.NET Razor C# VIEWS,PARTIAL VIEWS AND MASTER LAYOUT.

SocioBoard Uses 2 Main .cshtml file in the SocioBoard>Themes>SocioBoard>Views>Shared Folder and they are:-

1.) Site.cshtml: The Views File Contains the Hearder,footers,Left Informations and dropdown Boxes HTML Contents. It also has some Javascripts written at the end. This Page is an After Login Master Page.
2.) SocialSuite.cshtml: This Views Files has the Header and Footer Html Contents of the main Homepage of the SocioBoard site.Including the Login and registering Pop-up Box.It also has some Javascripts written at the end.This Page is Socioboard Masterpage Before Login.

The HTML Code for the HOMEPAGE Description is written in SocioBoard>Themes>SocioBoard>Views>Index>Index.cshtml.

Some of the Views also has many Javascripts code written at the end.

3.3 JAVASCRIPTS

Although Most of the Javascripts are written in the Views,A Particular base folder is also present where all the Main JS codes are present

The Address of this main JS Folder is SocioBoard>Themes>Contents>JS.

3.4 Web References

Web Service is very general model for building applications that can be implemented for any operating system that supports communication over the web.

Web Reference is the old-style, deprecated ASP.NET webservices (ASMX) technology (using only the XmlSerializer for your stuff) - if you do this, you get an ASMX client for an ASMX web service.

Adding a service reference allows you to create a WCF client, which can be used to talk to a regular web service provided you use the appropriate binding. Adding a web reference will allow you to create *only* a web service (i.e., SOAP) reference.

Steps to Add Web Reference:-

1.) Right Click on Service Reference and Select 'Add Reference'.

2.) Click on Advance.

3.) Click on 'Add Web Reference'.

4.) **When you Select 'Web Services in this solution',It will fetch and show all the services that is added in the project Api.SocioBoard >Services,as this project also present in the same Solution.**

For example:-

Api.Affiliates
Api.Facebook
Api.TwitterFeed
Api.TumblrFeed
etc.

4. Library Files

The SocioBoard Project contains the Following Library Files that helps and Maintains the Structure of Socioboard Clear and Understandable:-

- GlobusGooglePlusLib
- GlobusInstagramLib
- GlobuslinkedinLib
- GlobusMailLib
- GlobusMultiLangaugeLib
- GlobusDropBoxLib
- GlobusTumblrLib
- GlobusTwitterLib
- GlobusWordpressLib

The basic Functionality of these console Applications is to make the entire program clean and understandable and segregate each and every Modules like twitter,Linkedin,for decreasing clutter of code and memory.

In case of use of any of the classes and functions of any particular Library into any of the project i.e API.SocioBoard or SocioBoard,Just ADD THE DLL File of the respective Library into your Projects' references.

You will find the DLL(Dynamic Link Library) of the particular Library/Console Application in the "bin" folder of that particular Library/Console Application.

5. CONTROL FLOW

The control flow of Socioboard Starts from the UI Model,to Javascript POST function,which takes the Control flow to the Controller Action,Following with API Interaction and Then Fetching/Inserting Data into the database through Repository classes having Jqueries and then revert back to same UI with the same reverse procedure.

Lets take an Example Of Publishing a Message into one of the profiles added by you in you SocioBoard account.

5.1 UI Control Flow

The Main HTML content of the publishing page is present in the PUBLISHING VIEWS Folder and INDEX.CSHTML View.

```
<button id="scheduled" class="btn btn-outline btn-default"
type="button">@GlobusMultiLanguageLib.Resources.Publishing_Index_Schedule</button>
```

- The Above HTML content is Main Button Content that activates the whole Scheduling Process after all the details have been entered Correct and The USER Clicks on **"Schedule"** Button.

- Looking at the above Example,once can see <button id="scheduled" function which **invariably calls the Javascript written with the ID 'scheduled'**
- Co-incidently you will find this script written at the bottom of the INDEX.CSHTML page after all the HTML contents and components are completed.

Take a look at the Javascript Written over:-

```
        $('#scheduled').click(function () {
            debugger;
            var curdate = new Date();
            var dd = curdate.getDate();
            if (dd < 10) {
                dd = '0' + dd
            }
            else {
                dd = curdate.getDate();
            }
            var mm = curdate.getMonth() + 1;
            if (mm < 10) {
                mm = '0' + mm
            }
            else {
                mm = curdate.getMonth() + 1;
            }
            var hh = curdate.getHours();
            if (hh < 10) {
                hh = '0' + hh;
            }
            else {
                hh = curdate.getHours();
            }
            var minm = curdate.getMinutes();
            if (minm < 10) {
                minm = '0' + minm;
            }
            else {
                minm = curdate.getMinutes();
            }
            var sec = curdate.getSeconds();
            if (sec < 10) {
                sec = '0' + sec;
            }
            else {
                sec = curdate.getSeconds();
            }


                     var now = curdate.getFullYear() + "/" + mm + "/" + dd + " "
+ hh + ":" + minm + ":" + sec;
            var fd = new FormData();
            var filesimage =
document.getElementById('scheduleimageupload').files[0];
            if (filesimage != null) {
                fd.append('file', filesimage);
            }
            var scheduledmessage = $('#txtscheduled').val();
            var scheduleddate = $('#scheduleddate').val();
            var scheduledtime1 = $('#scheduledtime').val();
            var scheduledtime2 = scheduledtime1.split(" ");
            var scheduledtime3 = scheduledtime2[0] + ':00';
            var schdd = scheduledtime3.split(":");

            var shdd = schdd[0];
            var shmin = schdd[1];
            var shsec = schdd[2];
            if (shdd < 10) {
                shdd = '0' + shdd;
            }
            else {
                shdd = schdd[0];
            }
            if (shmin < 10) {
                //shmin = '0' + shmin;
                shmin = shmin;
            }
            else {
                shmin = schdd[1];
            }
            if (scheduledtime2[1] == "PM") {
                if (shdd != "12") {
                    shdd = +shdd + +12;
                }
            }
            var scheduledtime = shdd + ":" + shmin + ":" + shsec;
            var allprofiles = '';
            if (scheduledmessage == "") {
                alertify.set({ delay: 1000 });
                alertify.error('Please Enter a message ');
                return false;
            }
            if (scheduleddate == "") {
                alertify.set({ delay: 1000 });
                alertify.error('Please Select a Date');
                return false;
            }
            if (scheduledtime == "") {
                alertify.set({ delay: 1000 });
                alertify.error('Please Select a Time');
                return false;
            }
            if (arrbinduser.length < 1) {
                alertify.set({ delay: 1000 });
                alertify.error('Please Select Profiles to Scheduled a message');
                return false;
            }
            for (var i = 0; i < arrbinduser.length; i++) {
                allprofiles += arrbinduser[i] + "<:>";
            }
            for (var i = 0; i < arrbinduser.length; i++) {
                var account = arrbinduser[i];
                var strarray = account.split('~');
                var ProfileType = strarray[1];
                if (ProfileType == 'facebook') {
                    if (scheduledmessage.length > 5000) {
                        alertify.error("Facebook Not Support more than 5000
characters");
                        return;
                    }
                }
                if (ProfileType == 'twitter') {
                    if (scheduledmessage.length > 140) {
                        alertify.error("Twitter Not Support more than 140 
characters");
                        return;
                    }
                }
                if (ProfileType == 'linkedin') {
                    if (scheduledmessage.length > 600) {
                        alertify.error("Linkedin Not Support more than 600 
characters");
                        return;
                    }
                }
            }
            allprofiles = allprofiles.substr(0, allprofiles.length - 3);
            $.ajax({
                type: 'POST',
                url: '../Publishing/ScheduledMessage?scheduledmessage=' + 
scheduledmessage + '&scheduleddate=' + scheduleddate + '&scheduledtime=' + 
scheduledtime + '&profiles=' + allprofiles + '&clienttime=' + now,
                data: fd,
                processData: false,
                contentType: false,
                success: function (msg) {
                    $('#scheduleddate').datepicker('setDate', null);
                    $('#txtscheduled').val('');
                    $('#scheduleddate').val('');
                    $('#scheduledtime').val('');
                    $('#tumbnail').children().html('');
                    alertify.set({ delay: 1000 });
                    alertify.success('Message Scheduled Successfully');
                    arrbinduser = [];

                    $('.bindprof').hide(1000, function () {
                        $('#bindcomposescheduledprofiles').html('');
                        $('#MessageCount').html('');
                    });
                }
            });

           });
```

- When This JavaCsript with ID ‘scheduled’ gets called,The flow of Execution reaches to script function and gets to execute line by line.
- Inside this Script you will find a POST Method that call the Controller Action of the Publishing Controller,That POST METHOD Looks like this.

```
$.ajax({
                type: 'POST',
                url: '../Publishing/ScheduledMessage?scheduledmessage=' + scheduledmessage
+ '&scheduleddate=' + scheduleddate + '&scheduledtime=' + scheduledtime + '&profiles=' +
allprofiles + '&clienttime=' + now,
                data: fd,
                processData: false,
                contentType: false,
                success: function (msg) {
```

- This AJAX POST Method call Takes the flow of execution to the Controller action Publishing Controller with arguments.
- The Controller name and Action Result name in the controller and jScript **MUST BE SAME.**
- The Arguments Name which are passed from Jscript to the Controller **MUST BE SAME** in both Jscript and Controller.

- Once the Execution Flow reaches the controller from Javascript,the serial execution starts.

The flow of execution reaches to that Service.In this case ,

```
  public ActionResult ScheduledMessage(string scheduledmessage, string scheduleddate, string
scheduledtime, string profiles, string clienttime)
        {
            var fi = Request.Files["file"];
            string file = string.Empty;
            if (Request.Files.Count > 0)
            {
                if (fi != null)
                {
                    var path = Server.MapPath("~/Themes/" +
System.Configuration.ConfigurationManager.AppSettings["domain"] + "/Contents/img/upload");

                    // var path =
System.Configuration.ConfigurationManager.AppSettings["MailSenderDomain"]+"Contents/img/upload";
                    file = path + "\\" + fi.FileName;
                    if (!Directory.Exists(path))
                    {
                        Directory.CreateDirectory(path);
                    }
                    fi.SaveAs(file);
                    path = path + "\\" + fi.FileName;
                }
            }

             User objUser = (User)Session["User"];
            Api.ScheduledMessage.ScheduledMessage ApiobjScheduledMessage = new
Api.ScheduledMessage.ScheduledMessage();
            string retmsg = ApiobjScheduledMessage.AddAllScheduledMessage(profiles,
scheduledmessage, clienttime, scheduleddate, scheduledtime, objUser.Id.ToString(), file);
            return Content("_ComposeMessagePartial");
            }
```

- The API Service that is called here is ScheduledMessage and the method of that service that is called is AddAllScheduledMessage(profiles, scheduledmessage, clienttime, scheduleddate, scheduledtime, objUser.Id.ToString(), file); with Arguments.

- Here,the Flow of Execution Reaches to the API ScheduledMessage and into the method AddAllScheduledMessage.

```
public string AddAllScheduledMessage(string typeandid, string ShareMessage, string
ClientTime, string scheduleddate, string scheduletime, string UserId, string PicUrl)
        {
            string[] datearr = scheduleddate.Split(',');
            foreach (var date in datearr)
            {
                DateTime scheduleddatetime =
Convert.ToDateTime(CompareDateWithclient(ClientTime, (date + " " +
scheduletime).ToString()));
                string[] profileandidarr = Regex.Split(typeandid, "<:>");
                foreach (var item in profileandidarr)
                {
                    string[] profileandid = item.Split('~');
                    string profiletype = profileandid[1];
                    string profileid = profileandid[0];
                    objScheduledMessage = new
Domain.Socioboard.Domain.ScheduledMessage();
                    objScheduledMessage.CreateTime = DateTime.Now;
                    objScheduledMessage.ProfileType = profiletype;
                    objScheduledMessage.ProfileId = profileid;
                    objScheduledMessage.Id = Guid.NewGuid();
                    objScheduledMessage.PicUrl = PicUrl;
                    DateTime client = Convert.ToDateTime(ClientTime);
                    objScheduledMessage.ClientTime = client;
                    objScheduledMessage.ScheduleTime = 
scheduleddatetime.ToLocalTime();
                    objScheduledMessage.ShareMessage = ShareMessage;
                    objScheduledMessage.UserId = Guid.Parse(UserId);
                    objScheduledMessage.Status = false;
                    if 
(!objScheduledMessageRepository.checkMessageExistsAtTime(objScheduledMessage.UserId, 
objScheduledMessage.ShareMessage, objScheduledMessage.ScheduleTime, 
objScheduledMessage.ProfileId))
                    {

objScheduledMessageRepository.addNewMessage(objScheduledMessage);
                    }
                }
            }
            return "";
        }
```

- Again the flow starts linearly and reaches to the object reference of the repository class containing the Queries that connects to the database for adding the details like message,time and date and other necessary details for scheduling the message into the data table.

```
public void addNewMessage(Domain.Socioboard.Domain.ScheduledMessage schmesg)
        {
            //Creates a database connection and opens up a session
            using (NHibernate.ISession session = 
SessionFactory.GetNewSession())
            {
                //After Session creation, start Transaction.
                using (NHibernate.ITransaction transaction = 
session.BeginTransaction())
                {
                    //Proceed action, to save new data.
                    session.Save(schmesg);
                    transaction.Commit();
                }//End Transaction
            }//End Session
        }
```

- After Inserting the Details into the database the flow of execution reverts back with the same flow and reaches to Jscript with the “Success” Message displayed.
- Once the Detials reaches the database,the backend Scheduler will do its job in scheduling the message.