Social Integration Platform

Authors: Vineet Ahirkar,Kunal Dhande,Ramyata Havaldar,Omkar Kulkarni

April 30, 2014

ACKNOWLEDGEMENT

Undertaking a project in the final year of Bachelor's Degree in Engineering is a mountainous task in itself, and it require able assistance along the way to make it successful, interesting, motivating, learning, professional and above all easy! In this we have been blessed with many people who have served as light and mentored us towards a successful realization of this project. It is with utmost humility and gratitude that we take this opportunity to recognize every one for your invaluable inputs. We address you all to express our heartfelt greetings.

We express our sincere gratitude towards the Head of our Information Technology Department, ***Prof. Rupali Chopade*** *for his invaluable support.*

We received zestful guidance from our project guide ***Prof. Jyoti Raghatwan*** *who provided us with new ideas throughout the project. We honor her consistent inputs and able guidance that has extensively served towards the completion of the task.*

We would also like to express our sincere gratitude to ***Mr Vishwas Kulkarni****, Computer-Home Pvt Ltd. for his idea and invaluable guidance throughout the project. This project wouldn't have been possible without our regular meetings held with him and his consistent efforts extended towards for the project.*

We sincerely thank all of the above for making this a very enriching experience.

Vineet Ahirkar
Kunal Dhande
Ramyata Havaldar
Omkar Kulkarni

ABSTRACT

The world is using social networks and other social media-based services to stay in touch, communicate and collaborate. Typical examples of social networks are Facebook, twitter and linked-in. So enterprises cannot grow without their existing applications including their web-sites, CRM etc. are integrated into the above social networks.

Today leading organizations are those organizations who have integrated their most of the applications with social networks. The revenue growth of social businesses is 24% higher than less social firm. However most of medium and small scale organizations do not have capability to integrate their existing applications with social media even if they want to. Each Social Media like Facebook, twitter and linked-in have different and complex mechanism for integration. So there is long learning curve to master these integration capabilities. So the proposed Social Integration Platform will allow all the enterprises to integrate their existing applications including their web-sites for all current and futures social networks.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 NEED

All the users of the social networking sites and other websites do not have the knowledge of all the languages and the APIs provided by these websites. When the user wants to integrate the plug-ins into his blogs or websites, he needs to study the APIs which is a difficult and time consuming task. Moreover it is difficult for him to access all his social networking sites at the same time. So we thought of creating a social platform in which the user can access the three integrated social networking sites using a single user-name and password. With this he will also be able to integrate whichever plug-ins in his sites or blogs.

This website is related with Social CRM and thus it is necessary to provide the analysis of his activities. Thus, with this website, it will provide the user to carryout sentimental analysis of the posts of Facebook and Twitter. For example, how many posts are positive, negative or neutral and also the number of positive, negative and neutral posts.

1.2 BASIC CONCEPTS

Social plug-ins are mainly used by bloggers. Some basic plug-ins are there on the social networking websites. If they want to customize these plug-ins, they need to know the languages in which these sites are implemented. It is very difficult for them to study all these languages. Our website is an intermediate between them to connect social networking and organizations to improve their business.

Our site gives all news feeds from social sites like Facebook, LinkedIn, and Twitter which help the user to understand customer reviews, expectations about their product and it also produces some statistical analysis about customer likes.

1.3 PROPOSED SYSTEM

Our system is divided into three modules -

1. **Plug-ins Generators -**
   Plug-ins from the different social sites are integrated together to give user customized plug-ins.

2. **Integrated Social Wall -**
   News feeds from different social media are collected together to give user a time-line effect on the wall. So he can access his all social accounts at the same time on the same site.

3. **Statistical Analysis -**
   Posts retrieved from social sites are analyzed and statistical and sentimental analysis result that gives number of likes/dislikes, list of positive and negative response messages that may help an organization to decide their future plans is produced.

Chapter 2

LITERACTURE SURVEY

2.1 EARLIER SYSTEMS

Before the boom of the internet technology, marketing and customer support used to be based mainly on the traditional Customer Relationship Model (CRM). This models had the following modules:

1. **Sales :**
   CRM is sales driven. By collecting customer data, companies can target campaigns to specific customers. The idea is to keep getting them back for more.
2. **Marketing :**
   Traditional marketing is all about direct advertising with the aim of boosting sales. Communication is one way, from companies to customers.
3. **Service and Support :**
   Traditional customer service operated within the hours the hours and channels dictated by the company. It is often automated and impersonal, i.e., a call center worker following a script.
4. **Feedback :**
   Companies directly contact customers to receive feedback. Customers may share experiences through word-of-mouth, but within small circles.

2.2 CURRENT AVAILABLE SYSTEMS

Today's social customer wants to interact with the companies in a different way. The response has been the rise of Social CRM. It is estimated that, 20% of the world's population makes use of social networks.
Although, Social CRM has modules similar to CRM, its functionality is radically different. Following are the modules:

1. **Sales :**
   SCRM is not a direct pathway to customers, but to potential customers. Sales via social commerce are expected to reach $30 billion within five years.

2. **Marketing :**
   Customers want conversations and engagement; not direct advertising. 58% businesses saw a drop in market costs by moving to social marketing.

3. **Service and Support :**
   Customers expect brands to respond quickly using their preferred platform. 15% of 1624 years olds prefer to interact with customer service using social media.

4. **Feedback :**
   Customers share their experiences with millions online. 53% of twitter users recommend companies in their tweets.

2.3 SENTIMENT ANALYSIS

Sentiment analysis (also known as opinion mining) refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials.

Generally speaking, sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document. The attitude may be his or her judgment or evaluation, affective state (that is to say, the emotional state of the author when writing), or the intended emotional communication (that is to say, the emotional effect the author wishes to have on the reader).

The rise of social media such as blogs and social networks has fueled an interest in sentiment analysis. With the proliferation of reviews, ratings, recommendations and other forms of online expression, online opinion has turned into a kind of virtual currency for businesses looking to market their products, identify new opportunities and manage their reputations. As businesses look to automate the process of filtering out the noise, understanding the conversations, identifying the relevant content and actioning it appropriately, many are now looking to the field of sentiment analysis. If web 2.0 was all about democratizing publishing, then the next stage of the web may well be based on democratizing data mining of all the content that is getting published.

One step towards this aim is accomplished in research. Several research teams in universities around the world currently focus on understanding the dynamics of sentiment in communities through sentiment analysis. The Cyber Emotions project, for instance, recently identified the role of negative emotions in driving social networks discussions. Sentiment analysis could therefore help understand why certain e-communities die or fade away (e.g., Myspace) while others seem to grow without limits (e.g., Facebook).

The problem is that most sentiment analysis algorithms use simple terms to express sentiment about a product or service. However, cultural factors, linguistic nuances and differing contexts make it extremely difficult to turn a string of written text into a simple pro or con sentiment. The fact that humans often disagree on the sentiment of text illustrates how big a task it is for computers to get this right. The shorter the string of text, the harder it becomes.

A basic task in sentiment analysis is classifying the polarity of a given text at the document, sentence, or feature/aspect level ? whether the expressed opinion in a document, a sentence or an entity feature/aspect is positive, negative, or neutral. Advanced, "beyond polarity" sentiment classification looks, for instance, at emotional states such as "angry," "sad," and "happy."

A different method for determining sentiment is the use of a scaling system whereby words commonly associated with having a negative, neutral or positive sentiment with them are given an associated number on a 0 to 10 scale (most negative up to most positive) and when a piece of unstructured text is analyzed using natural language processing, the subsequent concepts are analyzed for an understanding of these words and how they relate to the concept. Each concept is then given a score based on the way sentiment words relate to the concept, and their associated score. This allows movement to a more sophisticated understanding of sentiment based on an 11 point scale. Alternatively, texts can be given a positive and negative sentiment strength score if the goal is to determine the sentiment in a text rather than the overall polarity and strength of the text.

There are some websites available related our system:

- **Addthis.com :**
  A widely used social bookmarking service that can be integrated into a website with the use of a web widget. Once the widget is added, visitors to the website can bookmark an item using a variety of services, such as Facebook, Myspace, Google Bookmarks, Pinterest, and Twitter. Such plugins are easily customizable as the user wants to, which can give him a way to modify them to fit to the user?s web site. AddThis's audience platform enables brand marketers to deliver interest-based advertising to social influencers. AddThis doesn?t provide a platform to be active for the user to integrate into the user?s own website nor does it do any statistical or sentiment analysis of the user?s posts. Thus, this web site offers only social plugins to be used by people who want to have a social presence for their company or product.

- **Nimble.com :**
  Nimble basically provides an integrated social platform for people who want to be socially active on all the social networking sites. Such a tool such gives the users to operate all their social profiles from a single website rather than going to the websites of the corresponding social networks. This saves tremendous time and effort, which the user can use to do better things. Also this gives the user a quick glance at his/her social notifications and also an easy and fast way to respond to the notifications. Nimble doesn?t provide plugins for the user to integrate into the user?s own website nor does it do any statistical or sentiment analysis of the user?s posts. Thus, it provides a way for the user to be socially active on all the social networks from a single platform.

- **Batchbook Social CRM:**
  Batchbook is also a social integration platform without any analytical tools. It helps a user to integrate his/her emails, mailing lists, spreadsheets, social feed, social messages, etc. By bringing it all together, helps the user to focus on the most important customers and advanced features like custom fields and tasks. Although it doesn?t provide any social plugins for the user to integrate to his website, it provides a unified platform for users to be socially active on the social networks and leverage its benefits. If communicating with your customers is something you need to do in your business, and a megaphone is not the most effective way to do so, then Batchbook might be the solution. It thus helps to build better relationships with your customers which is a part of Customer Relationship Management (CRM).

- **Microsoft Dynamics CRM (Yammer):**
  Yammer combined with Microsoft Analytics is full stack Social CRM package. It provides all the functionalities a SCRM should provide. This includes, a sustained social presence, analysis of customer activity on the user?s product/service, listening to all the positive/negative comments and appropriately analyzing them, generating reports/statistical charts to show the transitions in social influence. Also, it does a detailed sentiment analysis of the user?s posts and comments on that post, to judge the general opinion of the people about a particular post regarding a specific product of the user?s firm as a whole. But, this product is meant for huge organizations which are willing to invest lots into social marketing strategies. And due to this reason, individuals and other small scale businesses can?t use this product for social integration.

Chapter 3

CONCEPTS AND SPECIFICATION

3.1 PEOBLEM STATEMENT

To create a website that will help a user to integrate various social networking websites by connecting to them with just one login on this proposed website. He can thus manage or create a social platform for his/her company or organization and make analysis out of it.

3.2 OBJECTIVES

To study-

- Ruby On Rails documentation,
- CSS and Twitter Bootstrap related documentation on bootstrap.com
- Facebook documentation on developers.facebook.com,
- Open Graph API,
- LinkedIn documentation on linkedin.com,
- Twitter documentation on dev.twitter.com

To implement-

- Rails framework,
- Twitter Bootstrap CSS,
- Plugins Generator,
- Social integration with social networking websites like Facebook, LinkedIn, Twitter,
- And build topic wise vocabulary.

To perform-

- Sentiment Analysis to produce structured analysis.

3.3 TECHNOLOGIES USED

- Front End : HTML, CSS
- Back end : Ruby On Rails, MySQL 5.1
- Scripting Languages : JavaScript, JQuery

3.4 SOFTWARE REQUIREMENTS

- Operating System : Any
- Browsers : All Browsers with HTML 5 support

3.5 HARDWARE REQUIREMENTS

- Processor : Pentium 4
- RAM : 256 MB
- Network Interface Card (NIC)

Chapter 4

DESIGN SPECIFICATIONS

4.1 DATA FLOW DIAGRAM:

A data-flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design).

On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process.

A DFD provides no information about the timing or ordering of processes, or about whether processes will operate in sequence or in parallel. It is therefore quite different from a flowchart, which shows the flow of control through an algorithm, allowing a reader to determine what operations will be performed, in what order, and under what circumstances, but not what kind of data will be input to and output from the system, nor where the data will come from and go to, nor where the data will be stored.

When it comes to conveying how information data flows through systems (and how that data is transformed in the process), data flow diagram (DFD) are the method of choice over technical descriptions for three principal reasons.

1. DFDs are easier to understand by technical and nontechnical audiences.
2. DFDs can provide a high level system overview, complete with boundaries and connections to other systems.
3. DFDs can provide a detailed representation of system components.

DFDs help system designers and others during initial analysis stages visualize a current system or one that may be necessary to meet new requirements. System analysts prefer working with DFDs, particularly when they require a clear understanding of the boundary between existing systems and postulated systems. DFDs represent the following:

1. External devices sending and receiving data
2. Processes that change that data

3. Data flows themselves
4. Data storage location

The hierarchical DFD typically consists of a top-level diagram (Level 0) underlain by cascading lower level diagrams (Level 1, Level 2?) that represent different parts of the system.

4.1.1 Level 0

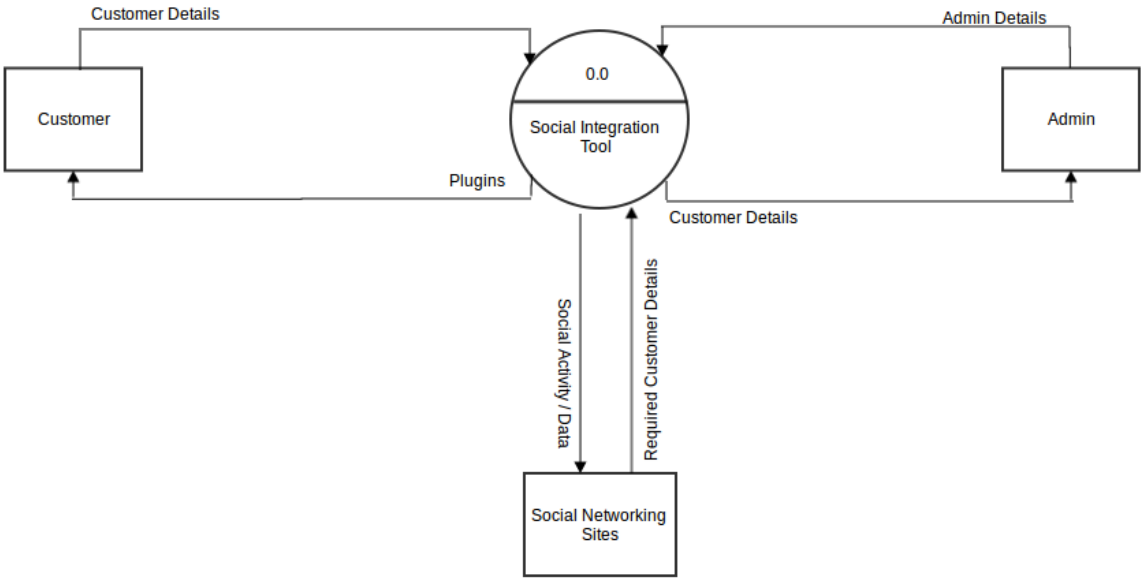

Figure 4.1: DFD Level 0

4.1.2 Level 1

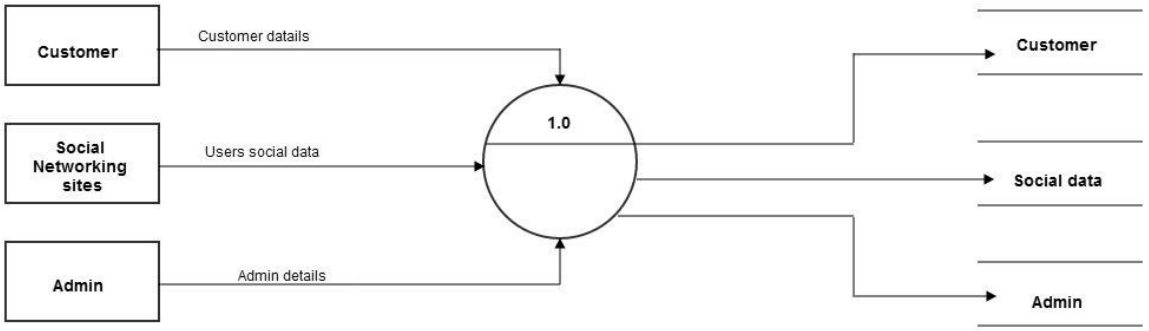

Figure 4.2: DFD Level 1

4.1.3 Level 2

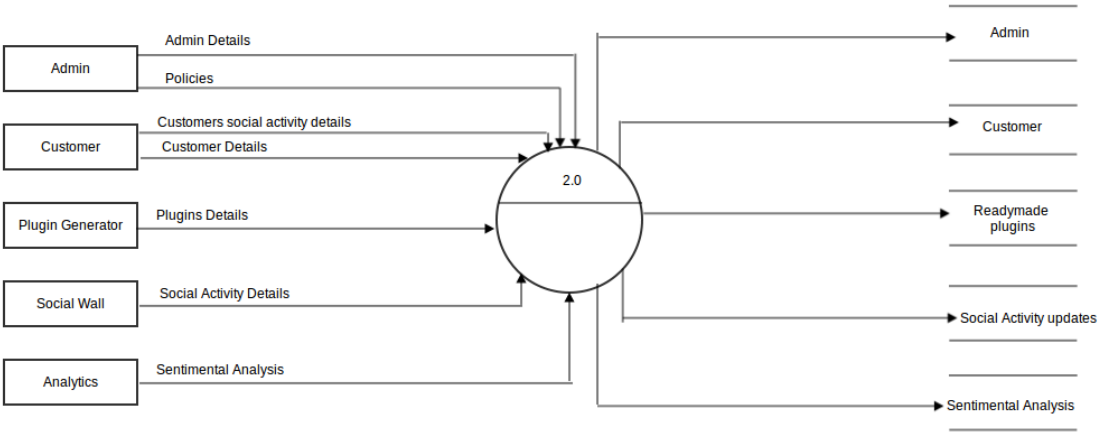

Figure 4.3: DFD Level 2

4.2 UML DIAGRAMS:

The Unified Modeling Language is a standard visual modeling language intended to be used for modeling business and similar processes, analysis, design, and implementation of software based systems.

UML is a common language for business analysts, software architects and developers used to describe, specify, design, and document existing or new business processes, structure and behavior of artifacts of software systems.

UML can be applied to diverse application domains (e.g., banking, finance, internet, aerospace, healthcare, etc.) It can be used with all major object and component software development methods and for various implementation platforms

The UML diagrams shown in the report are:

1. Use-case diagram
2. Class diagram
3. Activity diagram
4. Sequence diagram

4.2.1 Use-case diagram

Use Case Diagram is a type of behavioral diagram defined by UML created from a use case analysis. Its purpose is to present a graphical overview of the functionality provided by a system

in terms of actors, their goals are represented as use cases and any dependencies between those use cases.

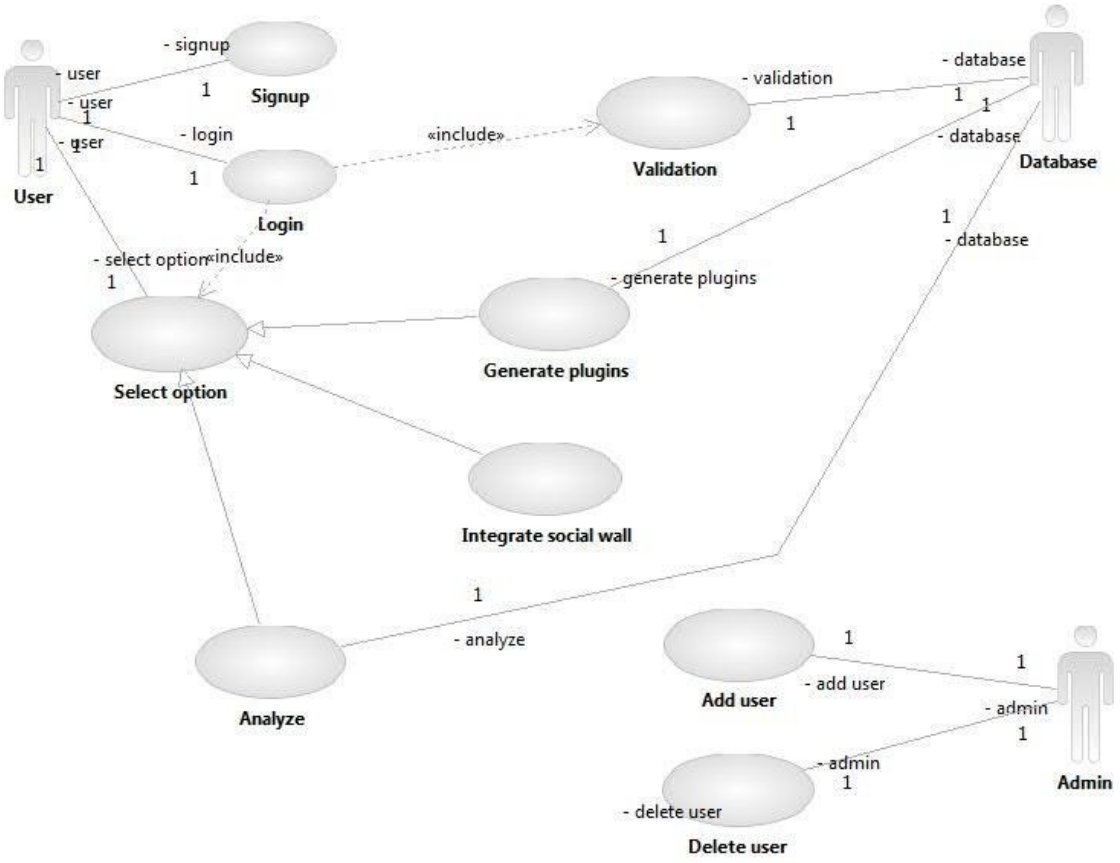

Figure 4.4: Use Case Diagram

The actors for the system "'Social Integration Platform"' are:

- Admin
- User
- Database

The admin is responsible for creating a new user if there is a new request and also for deleting the user if some user wants to delete the user. The user can sign up if he is a new user. If the user has already registered then he can directly login to the website. Then he can select the option he wants. If the user wants to include the social plugins in his post, then he can select the option of generate plugins. If he wants to view the messages that he has received on all the 3 social platforms, then he can select the option of integrated social wall. And if the user wants to analyze the number of comments or likes the post has received then he can select the option of analyze.

Database performs the task of validation and analysis. When the user logs in to our web-site, the database validates whether the user already has the account or is he a new user. And the database also analyzes the comments and likes whatever the user has received.

4.2.2 Class Diagram

The Class diagram stands at the center of the object-modeling process. It is the primary diagram for capturing all the rules that govern the definition and use of objects. As the repository for all the rules it is also the primary source for forward engineering (turning a model into code), and the target for reverse engineering (turning code into a model). This ability to generate code places the Class diagram in a unique position relative to all the other diagrams.

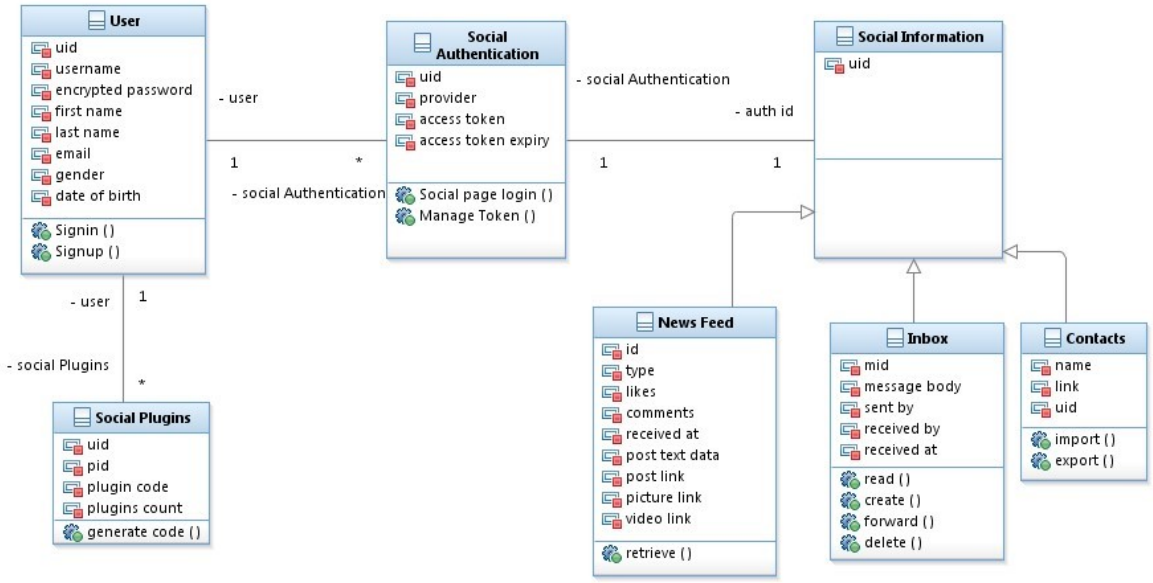

Figure 4.5: Class Diagram

In this diagram the classes are displayed and in the classes all the attributes and the function to be performed are included. The classes used are:

- User
- Social plugins
- Social authentication
- Social information
- News feed
- Inbox
- Contacts

The user performs the function of sign up if he is a new user. He performs the function of sign in if the user has already registered. He generates the social plugins he wants by performing the function of generate code. The user is then provided by a unique authentication ID and then he can access the notifications, his messages and all his contacts.

4.2.3 Activity diagram

Activity Diagram describes logical processes or functions. Each process describes a sequence of task and the decisions that govern when and how they are performed. All the activities

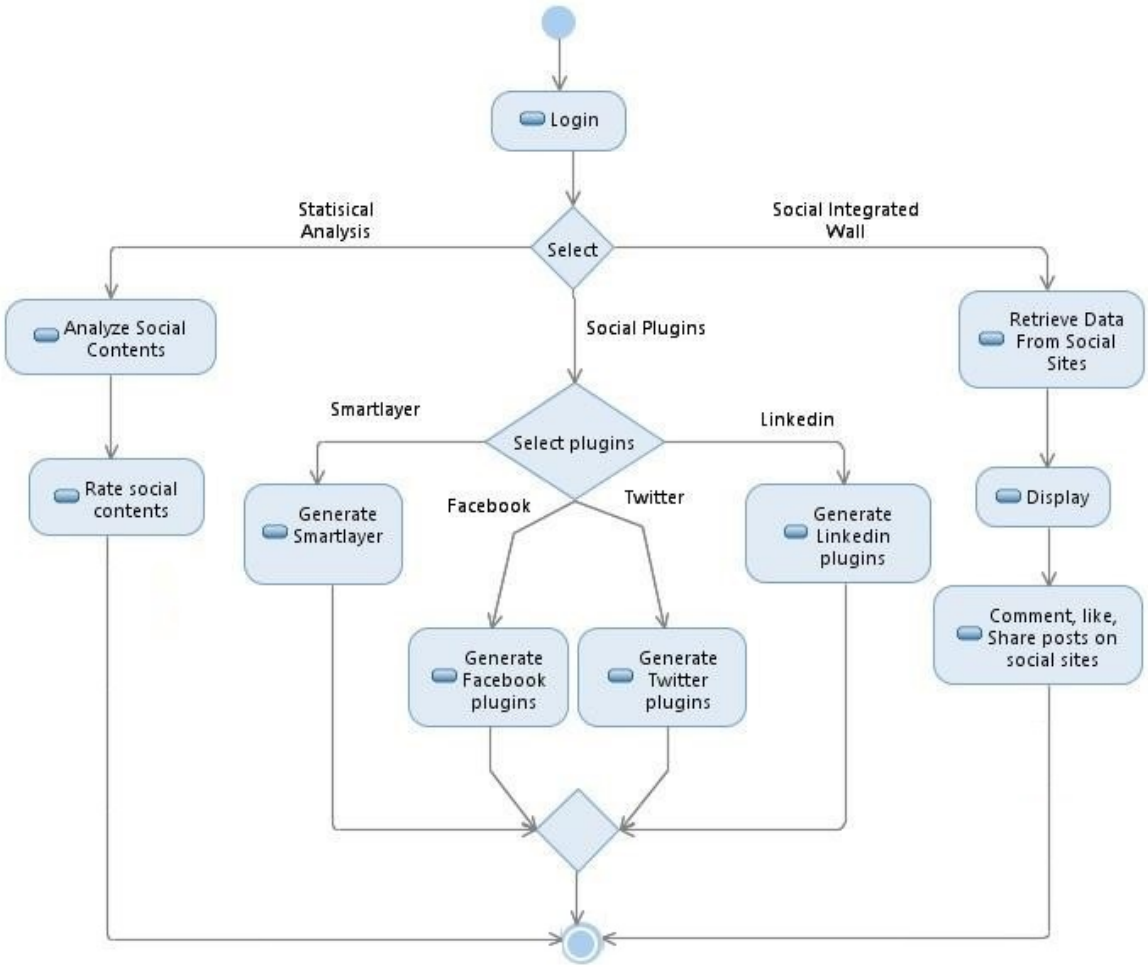

Figure 4.6: Activity Diagram

the user performs are shown in this diagram. After the user logs in he can select one of the 3 option.

- Social wall
- Plugin generation
- Statistical analysis

The messages from all the 3 social sites are retrieved. Then he can view all the comments, likes and notification he has received. If he selects the option of plugin generation then he can generate the code of all the plugins he wants to include in his post. If the statistical analysis option is chosen then the analysis of all his comments and likes is made.

4.2.4 Sequence diagram

Sequence Diagram shows, different processes or objects that live simultaneously as parallel vertical lines, and, the messages exchanged between them in order in which they occur as horizontal arrows. This allows the specification of simple runtime scenarios in a graphical manner.

3 sequence diagrams have been designed for our system.

1. **Sequence diagram for login process-**

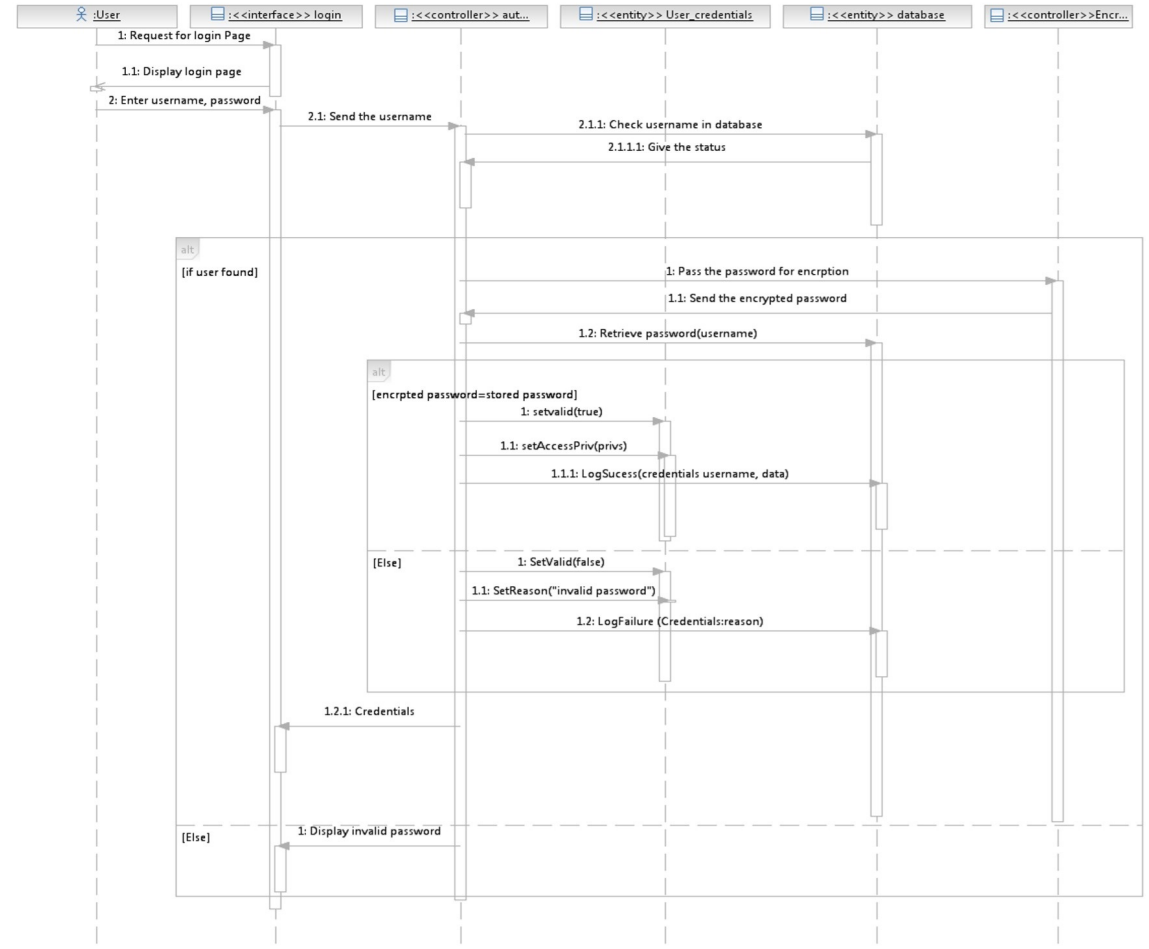

Figure 4.7: Sequence Diagram (Login)

The user requests for the login page to the interface and the interface displays the login page. The user fills his username and password the interface sends the username to the database to verify whether the user exists or not. If the user exists the interface sends the password for encryption. Then the encrypted password is compared with the password store in the database. If the match is found the user is successfully logged in and if the match is not found, the message "Please enter correct password" is displayed. If the user does not exist then the message "invalid user" is displayed.

2. **Sequence diagram for social wall-**

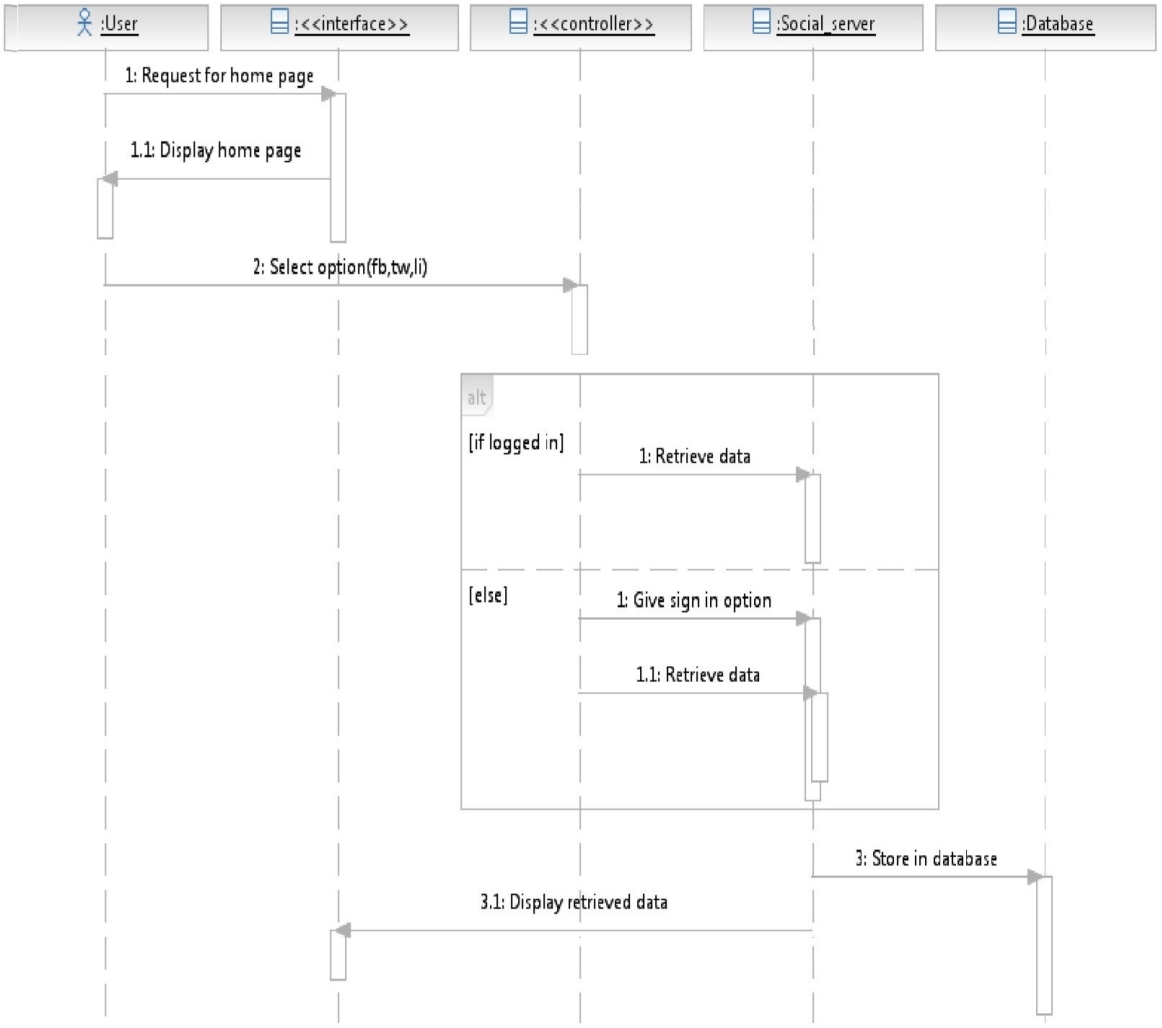

Figure 4.8: Sequence Diagram (Social Wall)

The user request for homepage the interface displays the homepage. User selects the social site which he wants to see (Facebook, Twitter, LinkedIn). The controller checks whether the user is already logged in to the social site he wants and if he is logged in the entire data is retrieved. If the user is not logged in the user is given a sign in option and when the user signs in the data is retrieved. Then the retrieved data is displayed to the user.

3. **Sequence diagram for plugin generator-**

The user requests for plugin page and the plugin page is displayed to the user by the interface. The user selects the plugins he wants to include in his post. The preview is displayed to the user. Then if the user requests for the generation of the code then the code is generated and displayed to the user. If the user cancels the request then the homepage is displayed.

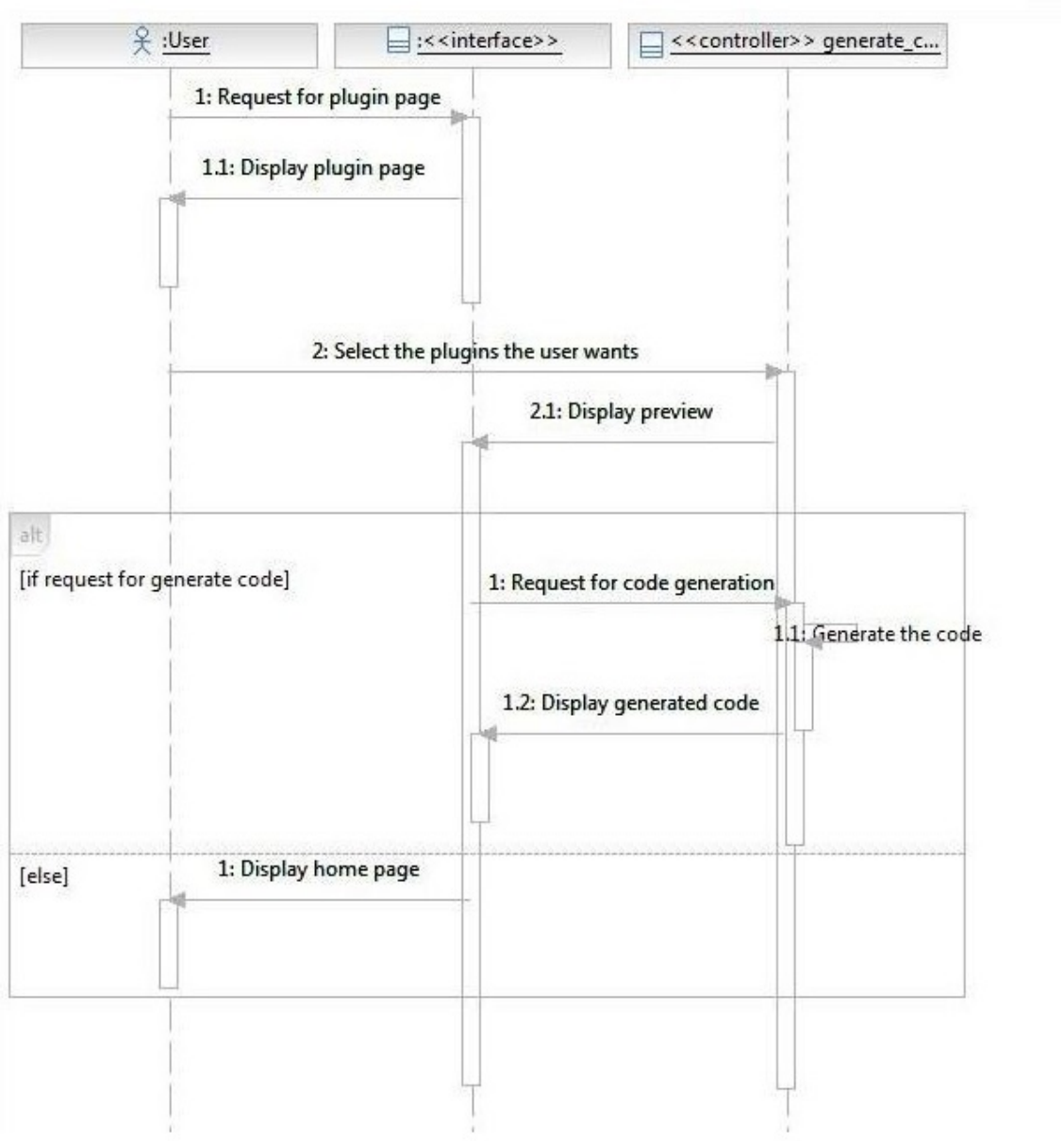

Figure 4.9: Sequence Diagram (Plugin)

Chapter 5

IMPLEMENTATION

5.1 RUBY ON RAILS

Ruby on Rails, often simply Rails or RoR, is an open source web application framework which runs via the Ruby programming language. It is a full-stack framework: it allows creating pages and applications that gather information from the web server, talk to or query the database, and render templates out of the box. As a result, Rails features a routing system that is independent of the web server.

Ruby on Rails emphasizes the use of well-known software engineering patterns and principles, such as active record pattern, convention over configuration (CoC), don't repeat yourself (DRY), and model–view–controller (MVC).

5.1.1 Ruby on Rails MVC framework

The **M**odel **V**iew **C**ontroller principle divides the work of an application into three separate but closely cooperative subsystems.

1. **Model (ActiveRecord):**
   Maintains the relationship between Object and Database and handles validation, association, transactions, and more.

   This subsystem is implemented in ActiveRecord library which provides an interface and binding between the tables in a relational database and the Ruby program code that manipulates database records. Ruby method names are automatically generated from the field names of database tables, and so on.

2. **View (ActionView)**
   A presentation of data in a particular format, triggered by a controller's decision to present the data. They are script based template systems like JSP, ASP, PHP and very easy to integrate with AJAX technology.

This subsystem is implemented in ActionView library which is an Embedded Ruby (ERB) based system for defining presentation templates for data presentation. Every Web connection to a Rails application results in the displaying of a view.

3. **Controller (ActionController):**
   The facility within the application that directs traffic, on the one hand querying the models for specific data, and on the other hand organizing that data (searching, sorting, massaging it) into a form that fits the needs of a given view.

   This subsystem is implemented in ActionController which is a data broker sitting between ActiveRecord (the database interface) and ActionView (the presentation engine).

5.1.2 Pictorial Representation of MVC Framework

A Pictorial Diagram of Ruby on Rails Framework is given here:

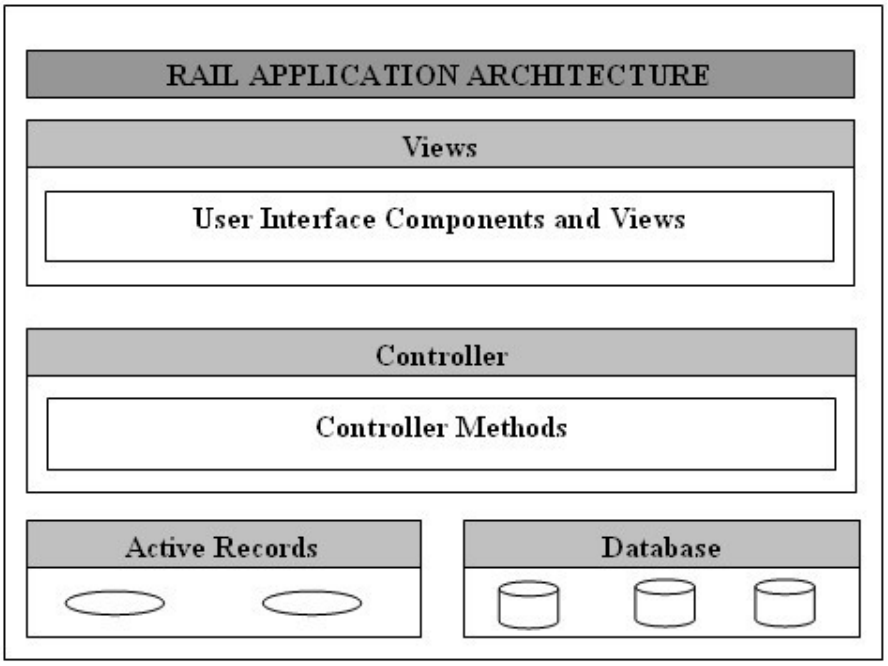

Figure 5.1: Ruby On Rails MVC Architecture

5.1.3 Directory Representation of MVC Framework

Assuming a standard, default installation over Linux, you can find them like this:

```
tp$>$ cd /usr/local/lib/ruby/gems/1.8/gems tp$>$ ls
```

You will see subdirectories including (but not limited to) the following:

- actionpack-x.y.z

- activerecord-x.y.z
- rails-x.y.z

Over a windows installation you can find them link this:

```
C:/ruby/lib/ruby/gems/1.8/gems/> dir
```

ActionView and ActionController are bundled together under ActionPack.

ActiveRecord provides a range of programming techniques and shortcuts for manipulating data from an SQL database. ActionController and ActionView provide facilities for manipulating and displaying that data. Rails ties it all together.

When you use the rails helper script to create your application, it creates the entire directory structure for the application. Rails knows where to find things it needs within this structure, so you don't have to tell it.

Here is a top level view of directory tree created by helper script at the time of application creation. Except for minor changes between releases, every Rails project will have the same structure, with the same naming conventions. This consistency gives you a tremendous advantage; you can quickly move between Rails projects without relearning the project's organization.

To understand this directory structure let's use **demo** application created in installation chapter. This can be created using a simple helper command

```
C:/ruby/> rails demo
```

Now go into demo application root directory as follows:

```
C:/ruby/demo> dir
```

5.1.4 Directory structure

:

- **app:** This organizes your application components. It's got subdirectories that hold the view (views and helpers), controller (controllers), and the backend business logic (models).
- **app/controllers:** The controllers subdirectory is where Rails looks to find controller classes. A controller handles a web request from the user.
- **app/helpers:** The helpers subdirectory holds any helper classes used to assist the model, view, and controller classes. This helps to keep the model, view, and controller code small, focused, and uncluttered.

- **app/models:** The models subdirectory holds the classes that model and wrap the data stored in our application's database. In most frameworks, this part of the application can grow pretty messy, tedious, verbose, and error-prone. Rails makes it dead simple!
- **app/view:** The views subdirectory holds the display templates to fill in with data from our application, convert to HTML, and return to the user's browser.
- **app/view/layouts:** Holds the template files for layouts to be used with views. This models the common header/footer method of wrapping views. In your views, define a layout using the <tt>layout :default</tt> and create a file named default.rhtml. Inside default.rhtml, call <% yield %> to render the view using this layout.
- **config:** This directory contains the small amount of configuration code that your application will need, including your database configuration (in database.yml), your Rails environment structure (environment.rb), and routing of incoming web requests (routes.rb). You can also tailor the behavior of the three Rails environments for test, development, and deployment with files found in the environments directory.
- **db:** Usually, your Rails application will have model objects that access relational database tables. You can manage the relational database with scripts you create and place in this directory.
- **doc:** Ruby has a framework, called RubyDoc, that can automatically generate documentation for code you create. You can assist RubyDoc with comments in your code. This directory holds all theR ubyDoc-generated Rails and application documentation.
- **lib:** You'll put libraries here, unless they explicitly belong elsewhere (such as vendor libraries).
- **log:** Error logs go here. Rails creates scripts that help you manage various error logs. You'll find separate logs for the server (server.log) and each Rails environment (development.log, test.log, and production.log).
- **public:** Like the public directory for a web server, this directory has web files that don't change, such a s JavaScript files (public/javascripts), graphics (public/images), stylesheets (public/stylesheets), and HTML files (public).
- **script:** This directory holds scripts to launch and manage the various tools that you'll use with Rails. For example, there are scripts to generate code (generate) and launch the web server (server).
- **test:** The tests you write and those Rails creates for you all go here. You'll see a subdirectory for mocks (mocks), unit tests (unit), fixtures (fixtures), and functional tests (functional).
- **tmp:** Rails uses this directory to hold temporary files for intermediate processing.
- **vendor:** Libraries provided by third-party vendors (such as security libraries or database utilities beyond the basic Rails distribution) go here.

5.2 BOOTSTRAP

For designing our website we have made use of twitter bootstrap. Bootstrap is basically an html, css, javascript framework that you can be used as basis for creating websites or web applications. It is a front-end framework from Twitter designed to kick start the front-end development of webapps and sites. Among other things, it includes base CSS and HTML for typography, icons, forms, buttons, tables, layout grids, navigation along with custom-built jQuery plug-ins and support for responsive layouts.

The version used for this project is 2.3.2.

Together, the Components and JavaScript plugins sections provide the following interface elements:

- Button groups
- Button dropdowns
- Navigational tabs, pills, and lists
- Navbar
- Labels
- Badges
- Page headers and hero unit
- Thumbnails
- Alerts
- Progress bars
- Modals
- Dropdowns
- Tooltips
- Popovers
- Accordion
- Carousel
- Typeahead

5.3 USER AUTHENTICATION

Gems used- Devise, Bcrypt,Warden

5.3.1 Devise

Devise is a flexible authentication solution for Rails based on Warden. It:

- Is Rack based;
- Is a complete MVC solution based on Rails engines;
- Allows you to have multiple models signed in at the same time;
- Is based on a modularity concept: use just what you really need.

It's composed of 10 modules:

- **Database Authenticatable:** encrypts and stores a password in the database to validate the authenticity of a user while signing in. The authentication can be done both through POST requests or HTTP Basic Authentication.
- **Omniauthable:** adds Omniauth (https://github.com/intridea/omniauth) support.
- **Confirmable:** sends emails with confirmation instructions and verifies whether an account is already confirmed during sign in.
- **Recoverable:** resets the user password and sends reset instructions.
- **Registerable:** handles signing up users through a registration process, also allowing them to edit and destroy their account.
- **Rememberable:** manages generating and clearing a token for remembering the user from a saved cookie.
- **Trackable:** tracks sign in count, timestamps and IP address.
- **Timeoutable:** expires sessions that have no activity in a specified period of time.
- **Validatable:** provides validations of email and password. It's optional and can be customized, so you're able to define your own validations.
- **Lockable:** locks an account after a specified number of failed sign-in attempts. Can unlock via email or after a specified time period.

Devise is guaranteed to be thread-safe on YARV. Thread-safety support on JRuby is on progress.

5.3.2 Bcrypt (ruby-gem)

An easy way to keep your users' passwords secure.

- **Why you should use bcrypt??**
  If you store user passwords in the clear, then an attacker who steals a copy of your database has a giant list of emails and passwords. Some of your users will only have one password — for their email account, for their banking account, for your application. A simple hack could escalate into massive identity theft.

  It's your responsibility as a web developer to make your web application secure — blaming your users for not being security experts is not a professional response to risk.

  bcrypt allows you to easily harden your application against these kinds of attacks.

- **How to install bcrypt??**

```
gem install bcrypt-ruby
```

- **How to use bcrypt in your Rails application??**

  1. The User model :

```
require 'bcrypt'

class User < ActiveRecord::Base
  # users.password_hash in the database is a :string include BCrypt

  def password
    @password ||= Password.new(password_hash)
  end

  def password=(new_password)
    @password = Password.create(new_password)
    self.password_hash = @password
  end
end
```

  2. Creating an account :

```
def create
  @user = User.new(params[:user])
  @user.password = params[:password]
  @user.save!
end
```

3. Authenticating a user :

```
def login
  @user = User.find_by_email(params[:email])
  if @user.password == params[:password]
    give_token
  else
    redirect_to home_url
  end
end
```

- **If a user forgets their password??**

```
# assign them a random one and mail it to them, asking them to change it
def forgot_password
  @user = User.find_by_email(params[:email])
  random_password = Array.new(10).map { (65 + rand(58)).chr }.join
  @user.password = random_password
  @user.save!
  Mailer.create_and_deliver_password_change(@user, random_password)
end
```

- **How to use bcrypt-ruby in general??**

```
require 'bcrypt'

my_password = BCrypt::Password.create("my password") #=>
    "$2a$10$vI8aWBnW3fID.ZQ4/zo1G.q1lRps.9cGLcZEiGDMVr5yUP1KUOYTa"

my_password.version #=> "2a"
my_password.cost #=> 10
my_password == "my password" #=> true
my_password == "not my password" #=> false

my_password = BCrypt::Password.new
    ("$2a$10$vI8aWBnW3fID.ZQ4/zo1G.q1lRps.9cGLcZEiGDMVr5yUP1KUOYTa")

my_password == "my password" #=> true
my_password == "not my password" #=> false
```

- **How bcrypt works??**
  bcrypt is a hashing algorithm designed by Niels Provos and David Mazières of the OpenBSD Project.

- **Background**
  Hash algorithms take a chunk of data (e.g., your user‘s password) and create a "digital fingerprint," or hash, of it. Because this process is not reversible, there‘s no way to go from the hash back to the password.

  In other words:

  ```
  hash(p) => <unique gibberish>
  ```

  You can store the hash and check it against a hash made of a potentially valid password:

  ```
  <unique gibberish> =? hash(just_entered_password)
  ```

- **Rainbow Tables**
  But even this has weaknesses — attackers can just run lists of possible passwords through the same algorithm, store the results in a big database, and then look up the passwords by their hash:

  ```
  PrecomputedPassword.find_by_hash(<unique gibberish>).password #=> "secret1"
  ```

- **Salts**
  The solution to this is to add a small chunk of random data — called a salt — to the password before it‘s hashed:

  ```
  hash(salt + p) #=> <really unique gibberish>
  ```

  The salt is then stored along with the hash in the database, and used to check potentially valid passwords:

  ```
  <really unique gibberish> =? hash(salt + just_entered_password)
  ```

  Bcrypt-ruby automatically handles the storage and generation of these salts for you.

  Adding a salt means that an attacker has to have a gigantic database for each unique salt – for a salt made of 4 letters, that‘s 456,976 different databases. Pretty much no one has that much storage space, so attackers try a different, slower method – throw a list of potential passwords at each individual password:

  ```
  hash(salt + "aadvark") =? <really unique gibberish>
  hash(salt + "abacus") =? <really unique gibberish>
  ```

  This is much slower than the big database approach, but most hash algorithms are pretty quick — and therein lies the problem. Hash algorithms aren‘t usually designed to be

slow, they‘re designed to turn gigabytes of data into secure fingerprints as quickly as possible. bcrypt(), though, is designed to be computationally expensive:

Ten thousand iterations:

Table 5.1: bcrypt Vs md5

	User	System	Total	Real
md5	0.070000	0.000000	0.070000	(0.070415)
bcrypt	22.230000	0.080000	22.310000	(22.493822)

If an attacker was using Ruby to check each password, they could check 140,000 passwords a second with MD5 but only 450 passwords a second with bcrypt().

- **Cost Factors**
  In addition, bcrypt() allows you to increase the amount of work required to hash a password as computers get faster. Old passwords will still work fine, but new passwords can keep up with the times.

  The default cost factor used by bcrypt-ruby is 10, which is fine for session-based authentication. If you are using a stateless authentication architecture (e.g., HTTP Basic Auth), you will want to lower the cost factor to reduce your server load and keep your request times down. This will lower the security provided you, but there are few alternatives.

5.3.3 Warden

- **What is Warden??**
  Warden is a Rack-based middleware, designed to provide a mechanism for authentication in Ruby web applications. It is a common mechanism that fits into the Rack Machinery to offer powerful options for authentication.

  Warden is designed to be lazy. That is, if you don’t use it, it doesn’t do anything, but when you do use it, it will spring into action and provide an underlying mechanism to allow authentication in any Rack-based application.

- **Why it is used??**
  With the push towards using Rack applications many opportunities are opening up. The promise of multiple applications running in the same process, sub-applications and sub-sub-applications can be realized.

  The lure of multiple applications is appealing to many. One of the questions however is how to manage authentication in this situation. Each application could require authentication or a “user”. Overall, authentication will likely be the same “user” for all applications in the Rack graph that is allowed to log in to the system. Authorization aside.

  Warden allows all downstream middlewares and endpoints to share a common authentication mechanism, whilst still allowing the overall application to manage it. Each application can access the authenticated user, or request authentication in the same way,

using the same logic throughout the Rack graph. Each application can layer whatever sugary API on top, and the underlying system will still work.

- **How it is used??**
  Warden sits in the Rack stack, after the session middleware (that stores a session, hash-like object in **env['rack.session']**).

  Warden injects a lazy object into the Rack environment at **env['warden']**. This lazy object allows you to interact with it to ask if it's authenticated or to force authentication to occur in any downstream piece of Rack machinery. If the request is authenticated, warden gets out of the way. If it's not, you can 'fail' it and cause it to react.

```
env['warden'].authenticated?
    # Ask the question if a request has been previously authenticated
env['warden'].authenticated?(:foo)
    # Ask the question if a request is authenticated for the :foo scope
env['warden'].authenticate(:password)
    # Try to authenticate via the :password strategy. If it fails proceed anyway.
env['warden'].authenticate!(:password)
    # Ensure authentication via the password strategy. If it fails, bail.
```

  If you don't want to authenticate a request, just don't ask env['warden'] to authenticate it.

  After authentication is performed, if successful, it will provide access to the "user" object. This can be anything except nil.

```
env['warden'].authenticate(:password)
env['warden'].user  \# the user object
```

  By placing it directly after the session middleware, all downstream middleware and applications will have access to the authentication object. This allows all applications to have a combined, coherent approach to authentication even if they're dropped in from another source. All Rack middlewares and endpoints can use the same underlying authentication system, even if they have layered different sugary APIs on top.

  A "strategy" is the place where the logic of authentication is actually run. See Strategies for more information.

- **Failing Authentication**
  When authentication should be failed, simply throw a :warden symbol at any point downstream of the Warden middleware. You can also add an options hash to the throw to include arbitrary information about the throw.

```
throw(:warden)  \# Bail out for an authentication failure
throw(:warden, :stuff =$>$ "foo") \# Bail out with some options to provide some
    context
```

What this does is bail out to a "failure application" that you must set up. The failure application, a standard Rack application, is there to handle cases of failed authentication. You may use it to render a login form for example.

5.4 OMNIAUTH

- **What is Omniauth?**
  The web application landscape has changed drastically in the past few years. Most users login in to dozens, sometimes hundreds of services each day; sites are no longer silos unto themselves and cannot reasonably expect users to create a unique login and password for each service. This is where OmniAuth comes in.

  OmniAuth is a library that standardizes multi-provider authentication for web applications. It was created to be powerful, secure, and flexible. Any developer can create strategies for OmniAuth that can authenticate users via disparate systems. OmniAuth strategies have been created for everything from Facebook to LDAP.

- **Who's Using OmniAuth?**
  Hundreds of people are leveraging the advantages of super simple, secure authentication with OmniAuth in their applications. From event registration sites to online wholesale distribution stores, to online trading platforms and beyond, developers are relying on OmniAuth to streamline user authentication in their systems.

  Getting Started
  To use OmniAuth in a project with a Gemfile, just add each of the strategies you want to use individually:

  - gem 'omniauth-facebook'
  - gem 'omniauth-twitter'
  - gem 'omniauth-linkedin'

  Now you can use the OmniAuth::Builder Rack middleware to build up your list of OmniAuth strategies for use in your application:

```
Rails.application.config.middleware.use OmniAuth::Builder do
    provider :facebook, ENV['GITHUB_KEY'], ENV['GITHUB_SECRET']
    provider :twitter , ENV['GITHUB_KEY'], ENV['GITHUB_SECRET']
    provider :linkedin, ENV['GITHUB_KEY'], ENV['GITHUB_SECRET']
end
```

  When using OmniAuth in a Rails application you can add it to your middleware:

  By default, OmniAuth will configure the path /auth/:provider. It is created by OmniAuth automatically for you, and you will start the auth process by going to that path.

Also by default, OmniAuth will return auth information to the path /auth/:provider/callback inside the Rack environment. In Sinatra, for example, a callback might look something like this:

```
# Support both GET and POST for callbacks
\%w(get post).each do |method|
  send(method, "/auth/:provider/callback") do
    env['omniauth.auth'] # => OmniAuth::AuthHash
  end
end
```

Also of note, by default, if user authentication fails on the provider side, OmniAuth will catch the response and then redirect the request to the path/auth/failure, passing a corresponding error message in a parameter named message. You may want to add an action to catch these cases. Continuing with the previous Sinatra example, you could add an action like this:

```
get '/auth/failure' do
  flash[:notice] = params[:message] # if using sinatra-flash or rack-flash
  redirect '/'
end
```

5.5 ADMINISTRATOR MANAGEMENT

- **Activeadmin :**
  Active Admin is a framework for creating administration style interfaces. It abstracts common business application patterns to make it simple for developers to implement beautiful and elegant interfaces with very little effort.

- **Getting Started**
  Active Admin is released as a Ruby Gem. The gem is to be installed within a Ruby on Rails 3 application. To install, simply add the following to your Gemfile:

```
# Gemfile
gem 'activeadmin'
```

If you are using Rails >= 3.1, you must also include a beta version of MetaSearch and sass-rails:

```
# Gemfile in Rails >= 3.1
gem 'activeadmin'
gem 'sass-rails'
```

```
gem "meta\_search", '>= 1.1.0.pre'
```

After updating your bundle, run the installer

```
> rails generate active\_admin:install
```

The installer creates an initializer used for configuring defaults used by Active Admin as well as a new folder at app/admin to put all your admin configurations.

Migrate your db and start the server:

```
> rake db:migrate
> rails server
```

Visit http://localhost:3000/admin and log in using:

- **User:** admin@example.com
- **Password:** password

Voila! You're on your brand new Active Admin dashboard.

To register your first model, run:

```
> rails generate active\_admin:resource [MyModelName]
```

This creates a file at app/admin/my_model_names.rb for configuring the resource. Refresh your web browser to see the interface.

5.6 SOCIAL AUTHENTICATION

5.6.1 Koala

- **Introductions**
  Using Koala, you have complete access to all of Facebook's APIs — the Graph API including batch requests and photo uploads, the Rest API, realtime updates, test users, and OAuth validation. Before we delve into using Koala to read and write from the social graph, though, it's worth a moment to go over our four goals for the library.

- **Koala's goals :**

  - **Lightweight**: Koala should be as light and simple as Facebook's own new libraries, providing support for the complete API and returning simple JSON hashes.

- **Fast**: Koala should, out of the box, be quick. We use Facebook's faster read-only servers whenever possible, and support a variety of HTTP libraries so you can make snappy requests however you like. That, of course, that brings us to our next topic:
- **Flexible:** Koala should be useful to everyone, regardless of their configuration. We support Ruby 1.8.7, 1.9.2 and 1.9.3, as well as JRuby, Rubinius, and REE, and use the Faraday library to provide complete flexibility over how HTTP requests are made.
- **Tested:** Koala should have complete test coverage, so you can rely on it. (Our complete test coverage can be run against either mocked responses or the live Facebook servers.)

Also, we want Koala to be great Ruby, and to be responsive to your needs and suggestions. Please, feel free to send us a Github message with any suggestions, feedback, or comments you have.

Getting the Code :

Start by installing the gem:

```
gem "koala"
```

Once that's done, using Koala is simple:

```
require 'koala'
# initialize a API connection, for instance
@graph = Koala::Facebook::GraphAPI.new
```

5.6.2 Twitter

- **Installation**

```
gem install twitter
```

- **Configuration**
  You can configure a Twitter::REST::Client by passing it a block when it's initialized.

```
client = Twitter::REST::Client.new do |config|
    config.consumer_key = "YOUR_CONSUMER_KEY"
    config.consumer_secret = "YOUR_CONSUMER_SECRET"
    config.access_token = "YOUR_ACCESS_TOKEN"
    config.access_token_secret = "YOUR_ACCESS_SECRET"
end
```

Note: oauth_token has been renamed to access_token and oauth_token_secret is now access_token_secret to conform to the terminology used in Twitter's developer documentation.

- **Streaming (Experimental)**
  This library now offers support for the Twitter Streaming API.

  An object may be one of the following:

  - Twitter::DirectMessage
  - Twitter::Streaming::DeletedTweet
  - Twitter::Streaming::Event
  - Twitter::Streaming::FriendList
  - Twitter::Streaming::StallWarning
  - Twitter::Tweet

- **Cursors**
  The Twitter::Cursor class has been completely redesigned with a focus on simplicity and performance.

- **Search Results**
  The Twitter::SearchResults class has also been redesigned to have an Enumerable interface. The #statuses method and its aliases (#collection and #results) have been replaced by #to_a. Additionally, this class no longer inherits from Twitter::Base. As a result, the #[] method has been removed.

- **Trend Results**
  The #trends method now returns an Enumerable Twitter::TrendResults object instead of an array. This object provides methods to determinte the recency of the trend (#as_of), when the trend started (#created_at), and the location of the trend (#location). This data was previously unavailable.

- **Tweets**
  The Twitter::Tweet object has been cleaned up. The following methods have been removed:

  - #from_user
  - #from_user_id
  - #from_user_name
  - #to_user
  - #to_user_id

  - #to_user_name
  - #profile_image_url
  - #profile_image_url_https

  These attributes can be accessed via the Twitter::User object, returned through the #user method.

- **Users**
  The Twitter::User object has also been cleaned up.
  The following aliases have been removed:

  - #favorite_count (use #favorites_count)
  - #favoriters_count (use #favorites_count)
  - #favourite_count (use #favorites_count)
  - #favouriters_count (use #favorites_count)
  - #follower_count (use #followers_count)
  - #friend_count (use #friends_count)
  - #status_count (use #statuses_count)
  - #tweet_count (use #tweets_count)
  - #update_count (use #tweets_count)
  - #updates_count (use #tweets_count)
  - #translator (use #translator?)

5.6.3 LinkedIn

Ruby wrapper for the LinkedIn API. The LinkedIn gem provides an easy-to-use wrapper for LinkedIn's REST APIs.

- **Installation**

```
gem install linkedin
```

Chapter 6

PLANING AND SCHEDULING

A project plan, according to the Project Management Body of Knowledge, is 'a formal, approved document used to guide both project execution and project control'. The primary uses of the project plan are to document planning assumptions and decisions, facilitate communication among stakeholders, and document approved scope, cost, and schedule baselines. A project plan may be summarized or detailed. Project Plan also defined as a statement of how and when a project's objectives are to be achieved, by showing the major products, milestones, activities and resources required on the project.

Project planning is part of project management, which relates to the use of schedules such as Gantt charts to plan and subsequently report progress within the project environment. Initially, the project scope is defined and the appropriate methods for completing the project are determined. Following this step, the durations for the various tasks necessary to complete the work are listed and grouped into a work breakdown structure.

6.1 PLAN OF EXECUTION

1. **Identification**
   Searching for different project ideas. Identifying and finalizing one of them for further implementation.

2. **Literature survey**
   The survey of existing systems and technologies is important for the feasibility study and to find out the drawbacks and strengths of project.

3. **Planning**
   The planning starts with the scope and vision of the project. According to the information prepared the project plan to complete the project in the period. Decide the resources required and technical environment.

4. **Analysis**
   The analysis of the system gets done by finding the major techniques like data forwarding, email verification and authentication scheme. Accordingly finds the major functionality of each module. Then the SRS document gets prepared, which includes the different types of requirements for the system like functional and non-functional requirements.

5. **Design**
   Design is done according to the modules present in the system. The different UML diagrams get drawn in the different perspectives like data, function and behavior that are sequence diagram, activity diagram.

6. **Coding**
   The different modules will get implement and time to time each module will gets review and test. Also the refactoring will get done i.e. nothing but an iterative refinement of the internal module structure.

7. **Testing**
   Test the system quality fix errors if any and improve if needed.

   (a) **Unit Testing :** Initially the backend database will be tested over a number of transactions. Then GUI for each scenario involving all possible cases is tested separately.

   (b) **Integration Testing :** All modules will be integrated and then testing of whole integrated testing will be performed. It also includes evaluation of project.

   (c) **System Testing :** The product was tested in the context of the entire system. Different Windows systems will be used for system testing and the performance will be monitored.

8. **Documentation**
   The Documentation and major deliverables and milestones will get prepare. The major work products like SRS and design documents will be kept as form of documentation.

Requirement Gathering and Plan for the initial part of the project was as follows:

1. Understanding the problem definition
2. Understanding the current scenario in the Enterprise
3. Gathering information about required Software Resources
4. Gathering information about required Hardware Resources
5. Preparing preliminary design of overall workflow of project
6. Deciding the modules required for overall execution
7. The Plan of execution for actual implementation will be as follows:

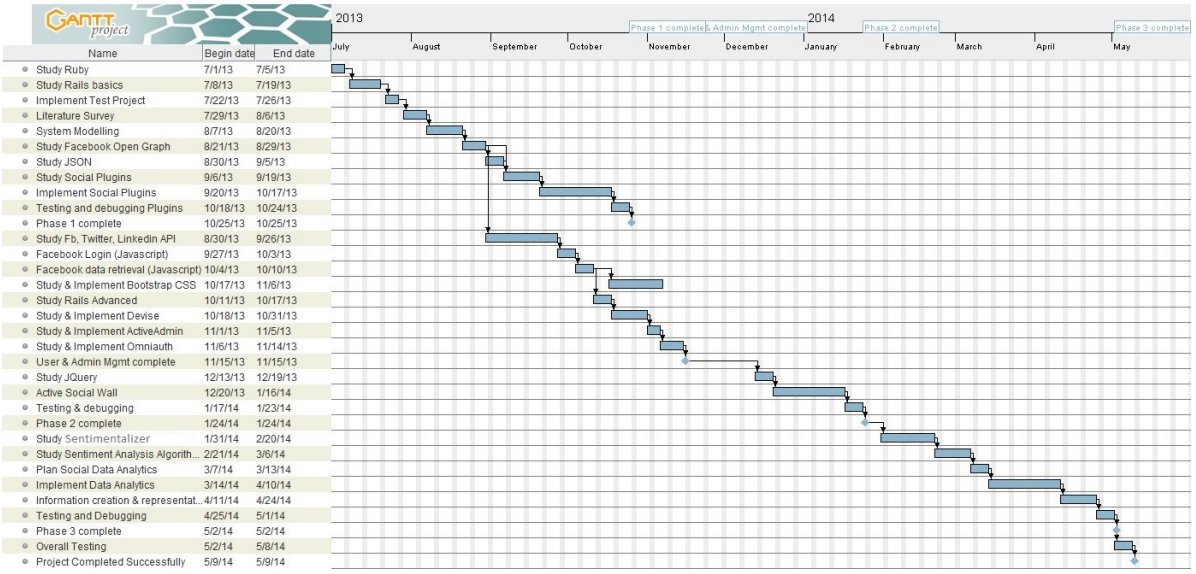

Figure 6.1: Project Schedule

6.2 STAFF ORGANISATION

The manner in which staff is organized and the mechanisms for reporting are noted.

6.2.1 Team structure

The team structure for the project is identified. Roles are defined. Our team consists of 4 individuals namely Vineet Ahirkar, Kunal Dhande, Ramyata Havaldar and Omkar Kulkarni. We have decided to keep the team structure highly flexible throughout the project. Each individual shall contribute equally through all the phases of the project namely Problem Definition, Requirements Gathering and Analysis, Design, Coding, Testing and Documentation.

The overall design issues will be primarily tackled by all team members. For every stage in design each member shall be responsible for bringing relevant knowledge to the discussion table following which there will be an identification and allotment of roles to each individual towards development of a module, all in consultation with our internal and external project guides.

The division of work shall change for every module of the software to be written as per the requirements imposed by that module. However, keeping in mind that one of the key goals of this project is trying to gain certain amount of expertise in the various areas of the project, it shall be ensured that each individual is exposed to different tasks with a view to enhance his knowledge.

In order to ensure that the project work goes in parallel, the team would be split in pairs and work distributed amongst these pairs. The pairing of individuals would also be dynamic so that the aforementioned goal of knowledge enhancement is satisfied. In spite of this work division cross team knowledge sharing would be encouraged so that each pair gets to know what the other pair does.

6.2.2 Management reporting and communication

Mechanisms for progress reporting and inter/intra team communication are identified. The teams shall be created dynamically as per requirements at different stages of the project. The teams shall communicate every 2 days by actual meeting or through remote means of communication like telephone, instant messaging or mail.

The project group will communicate with the internal guide every week or sooner as per project requirements with a view to collectively plan out further development and communicate the current status of the project.

6.3 TRACKING AND CONTROL MECHANISMS

6.3.1 Quality assurance and control

Quality control involves series of inspections, reviews, and tests used throughout the software development process. Our database is scalable. We will start with small amount of data and follow an incremental approach. Quality assurance consists of set of auditing and reporting functions that assess the effectiveness and completeness of quality control.

6.3.2 Change management and control

Change is inevitable when computer software is built.?

Chapter 7

TESTING

7.1 INTRODUCTION

Software testing is the process of testing the functionality and correctness of software by running it. Software testing is usually performed for one of the two reasons:

1. Defect detection
2. Reliability estimation

The problem of applying software testing to defect detection is that software can only suggest the presence of flaws, not their absence (unless the test is exhaustive). The problem of applying software testing to reliability estimation is that the input distribution used for selecting test cases may be flawed. In both of these cases, the mechanisms used to determine whether program output is correct (known as oracle) is often impossible to develop. Obviously the benefit of the entire software testing process is highly dependent on many different pieces. If any of these parts is faulty, the entire process is compromised.

The key to software testing is trying to find the myriad of failure mode-something that requires exhaustive testing of the code on all possible inputs. For most programs, this is computationally infeasible. It is common place to attempt to test as many of the syntactic features of the code as possible.

Techniques that try to exercise as much of the code as possible (within some of the resource constraints) are called white box software testing techniques. Techniques that do not consider the codes structure when test cases are selected are called black box testing.

7.2 OBJECTIVES AND TASKS

Objectives -

- Design Verification or compliance test- to be performed during the development or approval stages of the product, typically on a small sample of units.

- Manufacturing or production test- to be performed during preparation or assembly of the product in an ongoing manner for purpose of performance verification and quality control.
- Acceptance or commissioning test- to be performed at the time of delivery or installation of the product.
- Service and Repair test- to be performed as required over the service life of the product.

The testing objectives are:

1. Testing is a process of executing a program within the intent of finding an error.
2. A good test is one that has high probability of finding an as-yet-undiscovered error.
3. A successful test is one that uncovers an as-yet-undiscovered error.

Integration Testing:
In integration testing, the interactions between various modules and all specified requirements have been tested.

Test cases are based on the requirements specified in the Software Requirement Specification document and are further divided into function testing and integration testing. All test cases beginning with 'I' stand for Integration testing test cases and those with 'F' stand for Function testing.

Tasks :

- Alpha Testing
- System and Integration Testing
- Performance and Stress Testing
- User Acceptance Testing
- Batch Testing
- Automated Regression Testing
- Beta Testing

7.3 TESTING STRATEGY

7.3.1 Unit Testing

Definition:
In computer programming, unit testing is a method of testing that verifies the individual units of source code are working properly. A unit is the smallest testable part of an application. In

procedural programming a unit may be an individual program, function, procedure etc.

Methodology:
Each module of the system will be checked for its individual functionality. Modules such as authentication, registration, administration, user interaction, etc. will be tested to verify their functionality.

7.3.2 System and integration Testing

Definition:
System Integration Testing (SIT), in the context of software systems and software engineering, is a testing process that exercises a software system?s coexistence with others. System integration Testing takes multiple integrated systems that have passed system testing as input and tests their required interactions.

Methodology:
This phase of testing is performed only after performing Unit Testing. It is used to check whole system functionality. Various inputs will be given to check whether the system works for all kind of input or not.

7.3.3 Performance and Stress Testing

Definition:
Stress Testing is a form of testing that is used to determine the stability of a given system or entity. It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results. Performance Testing is used to determine the speed or effectiveness of the computer, network, software program or device. This process can involve quantitative tests done in a lab, such as measuring the response time or the number of MIPS (millions of instructions per second) at which a system functions. Qualitative attributes such as reliability, scalability and interoperability may be evaluated. Performance testing is often done in conjunction with stress testing.

Methodology:
Situations are created to create stress and see how the system performs under stress. Also performance of the system will be tested when the network traffic will be at its peak.

7.3.4 User Acceptance Testing

Definition:
Acceptance Testing is a black box testing performed on a system (e.g. software, lots of manufactured mechanical parts, or batches of chemical products) prior to its delivery. In some engineering sub-disciplines, it is known as Functional Testing, black box testing, release acceptance, QA testing, application testing, confidence testing, final testing, validation testing, usability testing, or factory acceptance testing.

Methodology:

It will be done when the system is complete. Various types of users will test the system for its overall functionality. Only after all the users approve it, the system will be accepted.

7.3.5 Batch Testing

Definition:
The sequential execution of more than one test case is called as Batch testing. Every test Batch consists of multiple dependent test cases. In those batches every end state is base state to next case. Test batch is also known as Test suit or test belt.

Methodology:
Batch testing is keeping the dependency test cases in one place. If we test one test case and if it is successfully executed then all other dependent test cases are passed where as if the test case is failed then all other teat cases are failed.

7.3.6 Automated Regression Testing

Definition:
Regression testing is selected retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still works as specified in the requirements.

Methodology:
The records of previous tests are checked and again those tests are performed to see if previous errors persist.

7.3.7 Beta Testing

Definition:
Beta testing is testing of computer product prior to its release. Beta testing is the last stage of testing, and normally can involve sending the product to beta test sites outside the company for real world exposure or offering the product for free trial download over the internet. Beta testing is often preceded by a round of testing called alpha testing.

Methodology:
After the system is completely built, it will be released for use, to examine the errors or problems that exist after testing.

7.4 HARDWARE REQUIREMENTS

Latest configured Computers with Internet connections.

7.5 CONTROL PROCEDURES

7.5.1 Problem Reporting:

In Beta testing users will be told to note down the errors and report them for keeping records and rectify those errors. While performing testing all the errors will be noted down to keep records.

7.5.2 Change Request:

If changes are to be made to existing system then versioning has to be performed. So that it will be possible to go to previous state.

7.6 DEPENDENCIES

Identify significant constraints on testing, such as test item availability, testing resource availability, and deadlines.

7.7 RISK / ASSUMPTIONS

If a particular test fails to detect problems, it may be costly to rectify it in later stages. So, all the tests should be done rigorously.

Table 7.1: Login Page Testing

Test case id:	1
Test case name:	Check functionality for Authentication
Tester name:	Vineet Ahirkar
Date:	04-03-2014
Test case objectives:	To verify/check the functionality of all buttons in form.
Prerequisites:	Databases should be in valid state

Test case ID	Action/ Objective	Actual Result	Expected Result	Result
ID1	Sign up-enter all the credentials and the suitable password	Login Page Displayed	Display login page	Pass
ID2	Sign in - Enter Username. and Password	Home Page Displayed	Display home page	Pass

Table 7.2: Plugin Page Testing

Test case id:	2
Test case name:	Check functionality for Social Plug-ins
Tester name:	Kunal Dhande
Date:	14-03-2014
Test case objectives:	To verify/check the functionality of all buttons in form.

Test case ID	**Action/ Objective**	**Actual Result**	**Expected Result**	**Result**
ID1	Select The appropriate plug-ins wanted	Get Code button Displayed	Display Get Code button	Pass
ID2	Click Get Code Button	Code for required plugins Displayed	Display Code for required Plugins	Pass

Table 7.3: Social Wall Page Testing

Test case id:	3
Test case name:	Check functionality for Social wall.
Tester name:	Kunal Dhande
Date:	5-04-2014
Test case objectives:	To verify/check the functionality of all buttons in form.

Test case ID	Action/ Objective	Actual Result	Expected Result	Result
ID1	Select Required Social Network	Required Social Network Displayed	Display Required Social Network	Pass
ID2	Select Appropriate Function	Appropriate Page Displayed	Display Appropriate Page	Pass
ID3	Select Update status and enter the text to be updated	"Successfully Updated" message Displayed	Display "Successfully Updated" Message	Pass

Table 7.4: Sentiment Analysis Testing

Test case id:	4
Test case name:	Check functionality for Sentiment Analysis
Tester name:	Vineet Ahirkar
Date:	23-03-2014
Test case objectives:	To verify/check the functionality of all buttons in form.

Test case ID	**Action/ Objective**	**Actual Result**	**Expected Result**	**Result**
ID1	Select Appropriate social page for sentiment Analysis	Required Page Displayed	Display Get Code button	Pass
ID2	Enter the text for sentiment analysis	Appropriate Sentiment Displayed	Display Appropriate Sentiments	Pass

Chapter 8

RESULTS

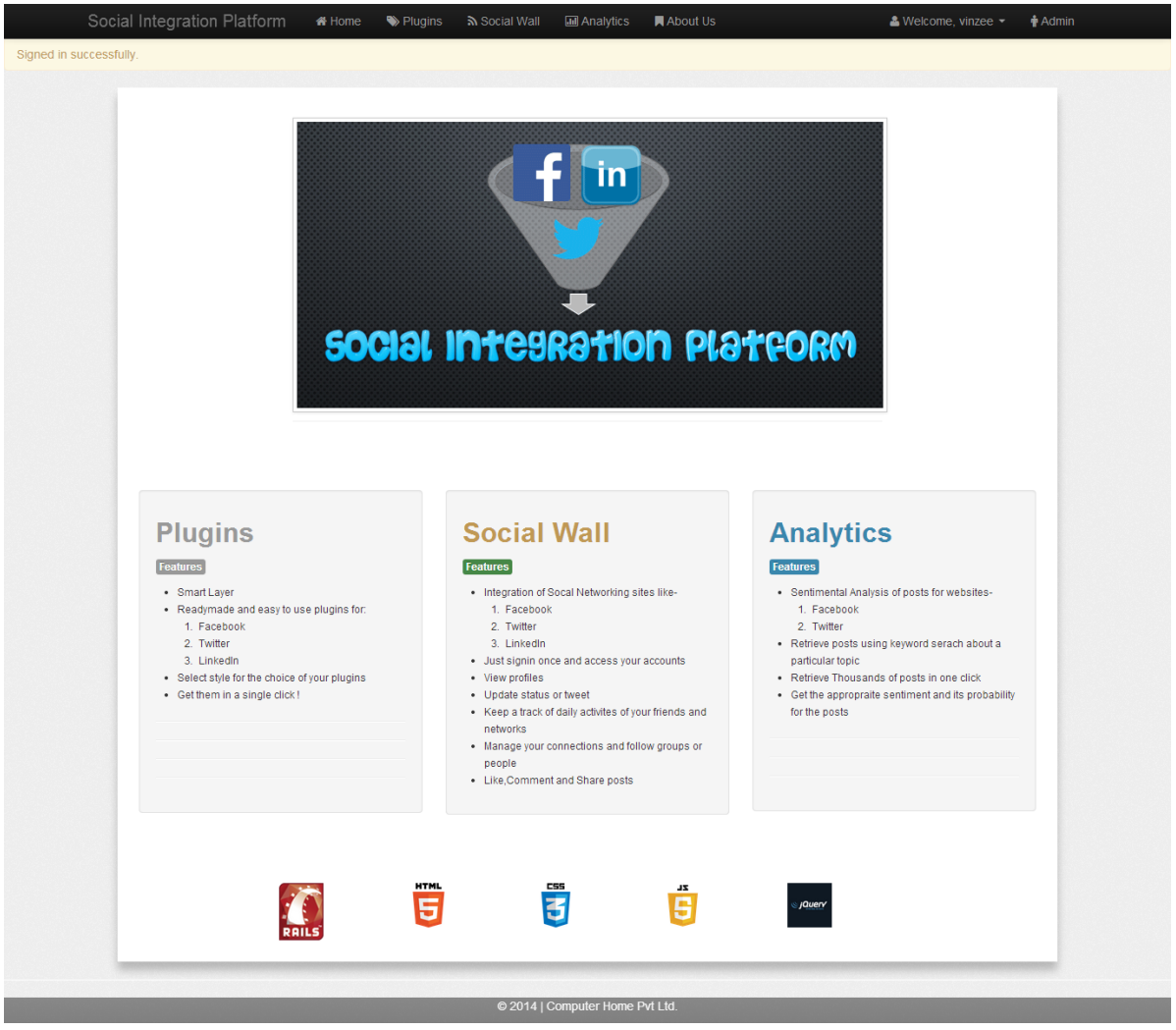

Figure 8.1: Home Page

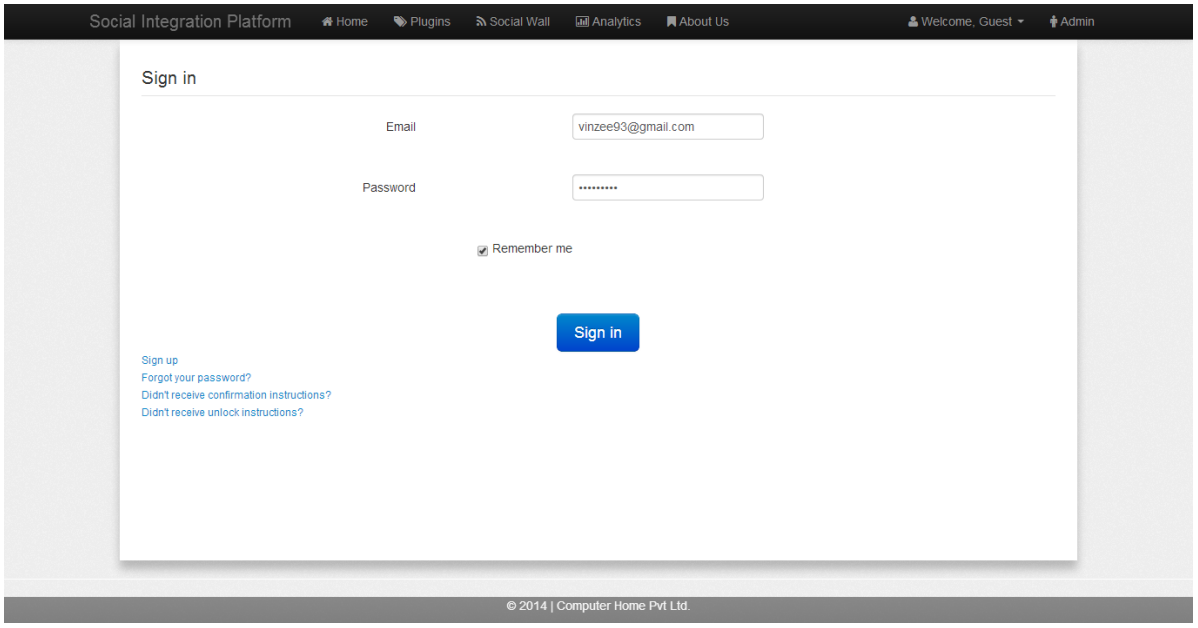

Figure 8.2: Signin Page

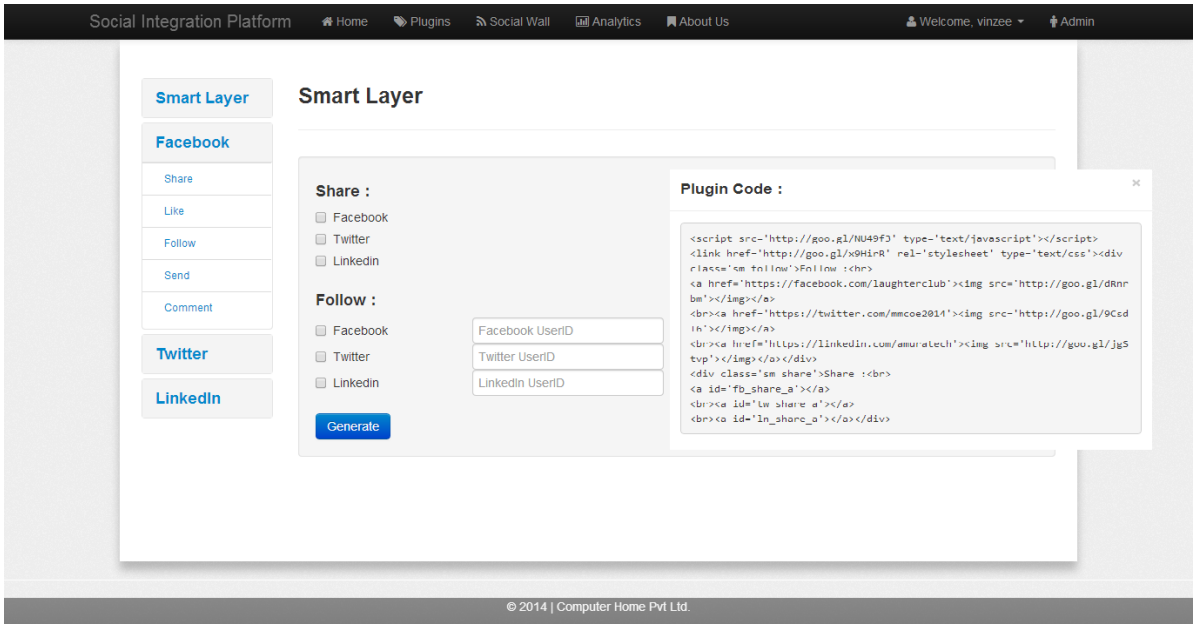

Figure 8.3: Plugins

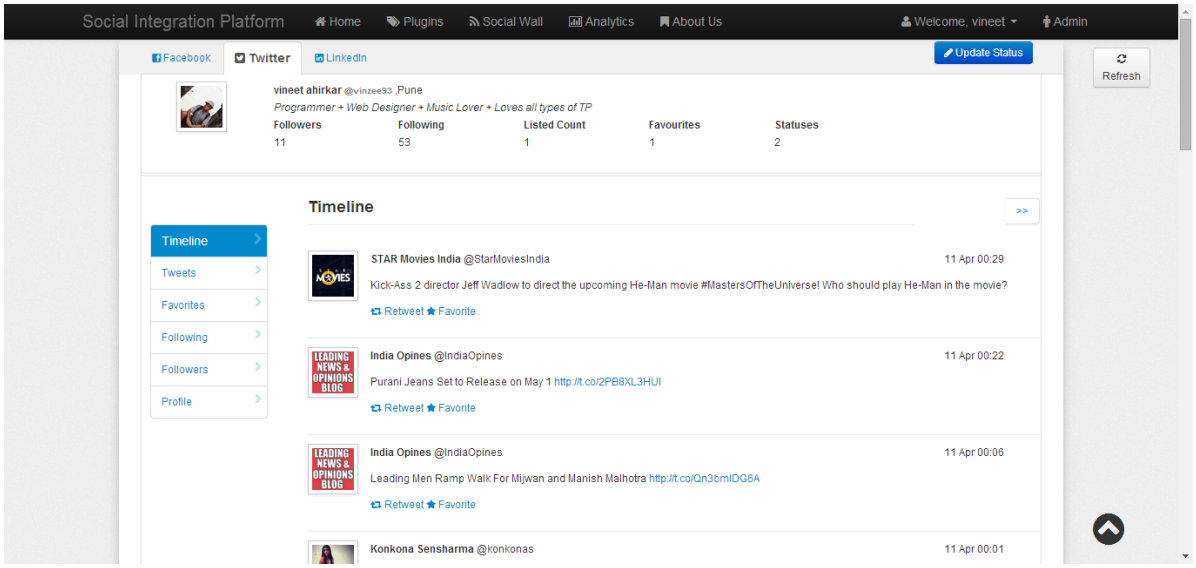

Figure 8.4: Social Wall

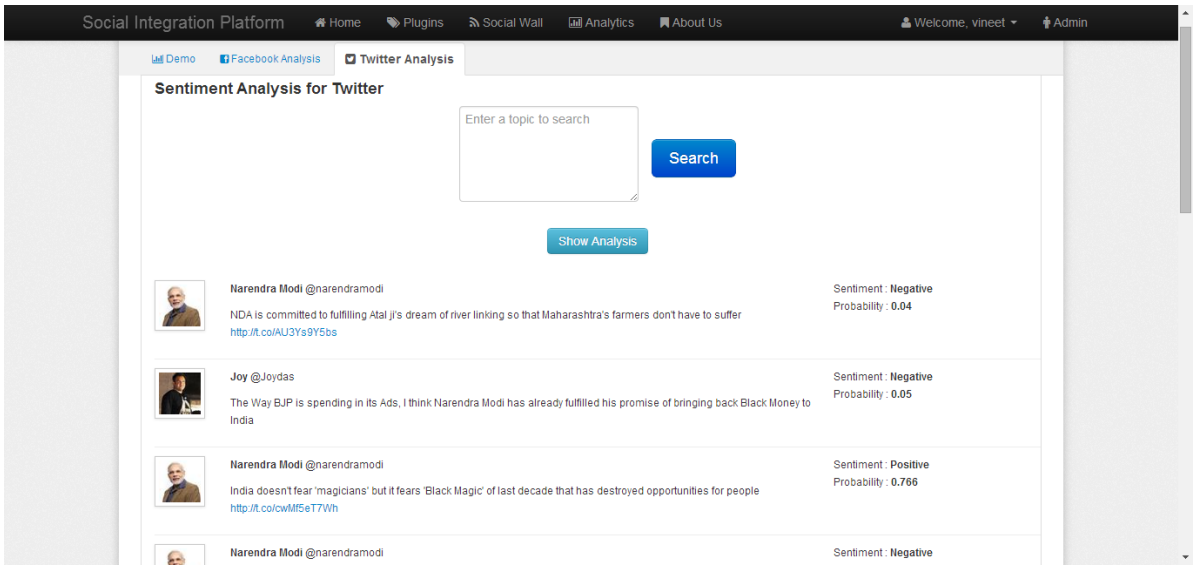

Figure 8.5: Sentiment Analytics

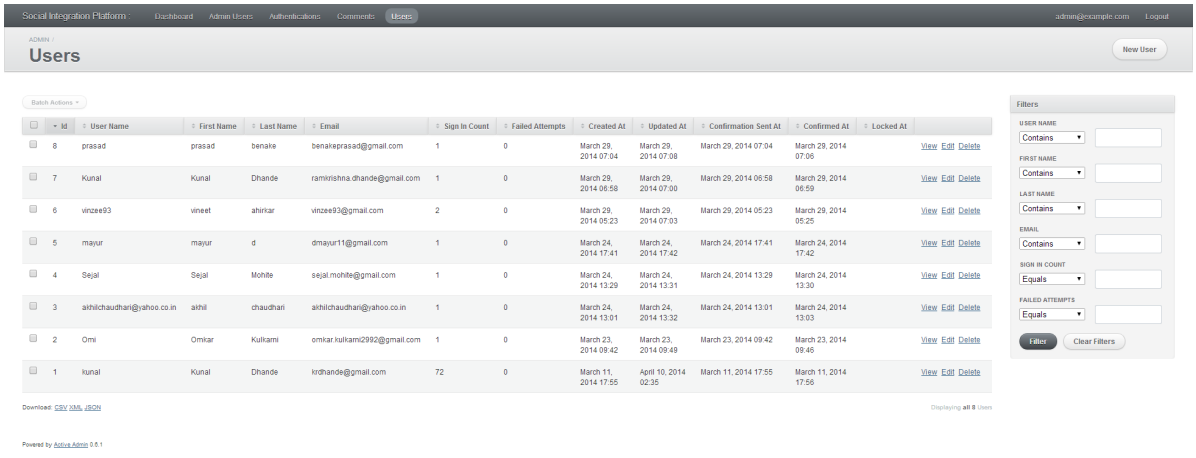

Figure 8.6: Admin

Chapter 9

APPLICATIONS

- Can be used by bloggers or website owners to include plugins of social networking sites like Facebook, Twitter and LinkedIn into their sites and blogs.
- Can be used by companies or organizations to remain socially active on various social networking sites simultaneously.
- Can also be used by organizations to carry out Sentimental Analysis on the topics of their choice on Facebook and Twitter.
- Can be used by an individual to keep a track of his daily social activities.

Chapter 10

CONCLUSION

The world is using social networks and other social media-based services to stay in touch, communicate and collaborate. Typical examples of social networks are Facebook, twitter and linked-in. So enterprises cannot grow without their existing applications including their websites, Customer Relationship Management (CRM) etc. are integrated into the above social networks. Today leading organizations are those organizations who have integrated their most of the applications with social networks. The revenue growth of social businesses is 24% higher than less social firm however most of medium and small scale organizations do not have capability to integrate their existing applications with social media even if they want to. Each Social Media like Facebook, twitter and linked-in have different and complex mechanism for integration. So there is long learning curve to master these integration capabilities.

Thus we have created a Social Integration Platform which is single unified platform or Application Programming Interface (API) which allow all enterprises to integrate their existing applications including their websites using our proposed Social integration platform to all current and future social networks.

Chapter 11

FUTURE SCOPE

The proposed system could be further improved by integrating it with various other social networking sites like Google+, Tumbler, Flicker, Branchout etc.

Also we plan to carry out Sentimental Analysis by implementing it on Apache Hadoop cluster and HDFS which will provide the user with parallel processing capabilities that will decrease the processing time for the analysis.

The next phase of the project could be implementing it as a full stack social CRM.

Bibliography

[1] Khanh Nguyen Duc A. Tran, "An Analysis of Activities in Facebook", the 8th Annual IEEE Consumer Communications and Networking Conference - Emerging and Innovative Consumer Technologies and Applications

[2] Xiaoyue Han, Lianhua Tian, Minjoo Yoon, Minsoo Lee, "A Big Data Model supporting Information Recommendation in Social Networks", 2012 Second International Conference on Cloud and Green Computing

[3] Namrata Godbole, Manjunath Srinivasaiah, Steven Skiena, "Large Scale Sentiment Analysis for News and Blogs", ICWSM?2007 Boulder, Colorado, USA

[4] David Alfred Ostrowski, "Sentiment Mining within Social Media for Topic Identification", International Conference on Semantic Computing 978-0-7695-41549/10 $26.00 2010 IEEE

[5] Sushant S. Khopkar, Rakesh Nagi, and Alexander G. Nikolaev, "An Efficient Map Reduce Algorithm for the Incremental Computation of All-Pairs Shortest Paths in Social Networks", 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining

[6] Nicholas Diakopoulos, Mor Naaman, Funda Kivran-Swaine, "Diamonds in the Rough: Social Media Visual Analytics for Journalistic Inquiry", IEEE Symposium on Visual Analytics Science and Technology October 24 - 29, Salt Lake City, Utah, USA 978-14244-9487-3/10/$26.00 2010 IEEE

[7] Facebook developers, https://developers.facebook.com/

[8] LinkedIn developers, https://developers.linkedin.com/

[9] Twitter developers, https://dev.twitter.com/

[10] Ruby Toolbox, https://ruby-toolbox.com/