

# Final Design Document

***The Average Joes***

***CS205 - Lafayette College***

— Submitted to  
Harvey  
Harvey's Rocket Ranch

## I. User Story

### A. General story:

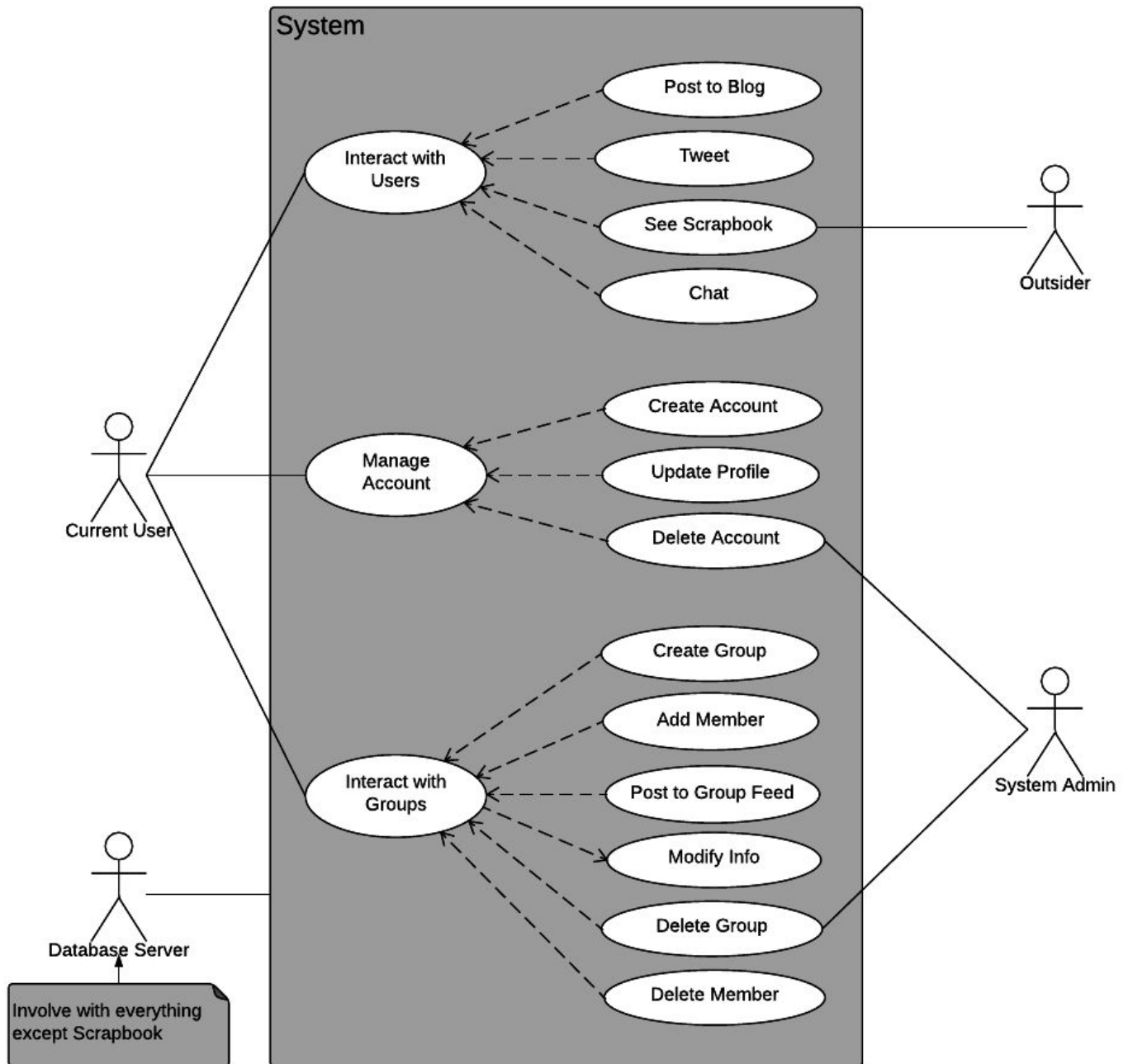
“As the rocket ranch owner, I want to use a communication software that promotes the principles of inclusivity and sharing among the guests. This is because I want the guests to have positive memories of the ranch so that they return in following years with new guests. To do so, I want the users to be able to share information and ideas they are passionate about using an individualized platform. In addition, I want the users to be able to communicate easily across their own project groups, as well as with other project groups.”

### B. User story for individual functions:

Function	User Story (Harvey - The customer)
Create/View Account	“As a rocket ranch owner, I want my customers to have an individualized means of storing the information they want to make available to the ranch so that the ranch operators may more easily separate the users into distinct groups.”
Users write blog posts	“As a rocket ranch owner, I want a means each of my customers can use to display the information they want to make public so that the users can share their interests and passions with one another. The tweeting system can also be used to communicate important announcements.”
Access User Scrapbook	“As a rocket ranch owner, I want each of my customers to be able to store their information in an orderly, easily accessible manner.”
Chat System	“As a rocket ranch owner, I want each of my customers to engage with other users within the groups they are a part of in order to promote inclusivity.”

## II. Use Cases

### A. Use Case Diagram



## B. Use Case Descriptions

<b>User Case Name</b>	Create Account										
<b>Goal In Context</b>	Each user should be able to create a new account upon arrival at the rocket ranch.										
<b>Successful End Condition</b>	The user should create an account that is stored in the system database with a designated username and password. The user should be able to access this account in a new session.										
<b>Failed End Condition</b>	The user will not be able to access this account in a new session. The account will not be stored in the database.										
<b>Primary Actor</b>	Current guest										
<b>Main Flow</b>	<table><thead><tr><th>Step</th><th>Action</th></tr></thead><tbody><tr><td>1</td><td>Current guest creates new account.</td></tr><tr><td>2</td><td>Program provides prompt for user to enter information.</td></tr><tr><td>3</td><td>Program uses entered data and creates new account object.</td></tr><tr><td>4</td><td>Account object is added to aggregate list of accounts within the system.</td></tr></tbody></table>	Step	Action	1	Current guest creates new account.	2	Program provides prompt for user to enter information.	3	Program uses entered data and creates new account object.	4	Account object is added to aggregate list of accounts within the system.
Step	Action										
1	Current guest creates new account.										
2	Program provides prompt for user to enter information.										
3	Program uses entered data and creates new account object.										
4	Account object is added to aggregate list of accounts within the system.										

<b>User Case Name</b>	Update Profile												
<b>Goal In Context</b>	User should be able to change the values within the account's "personal information" fields.												
<b>Successful End Condition</b>	Old values should be replaced with new values after updating personal info.												
<b>Failed End Condition</b>	The user will fail to change the values associated with each field.												
<b>Primary Actor</b>	Current guest												
<b>Main Flow</b>	<table><thead><tr><th>Step</th><th>Action</th></tr></thead><tbody><tr><td>1</td><td>User selects option to update personal information</td></tr><tr><td>2</td><td>User enters new values to each field they want to change</td></tr><tr><td>3</td><td>User chooses whether to save or discard changes</td></tr><tr><td>4</td><td>Discarding will allow the account to retain original values</td></tr><tr><td>5</td><td>Saving will delete previous values, setting each changed field to new values</td></tr></tbody></table>	Step	Action	1	User selects option to update personal information	2	User enters new values to each field they want to change	3	User chooses whether to save or discard changes	4	Discarding will allow the account to retain original values	5	Saving will delete previous values, setting each changed field to new values
Step	Action												
1	User selects option to update personal information												
2	User enters new values to each field they want to change												
3	User chooses whether to save or discard changes												
4	Discarding will allow the account to retain original values												
5	Saving will delete previous values, setting each changed field to new values												

<b>User Case Name</b>	Create Group						
<b>Goal In Context</b>	Goal is to create a new group object that the group manager can then add new users to join.						
<b>Successful End Condition</b>	Use case should create a new group that can be stored and retrieved from the database. The group should also be able to display all programmed functionality.						
<b>Failed End Condition</b>	The use case does not create a group, or it creates a group that cannot be stored or read from the database, or the created group does not display full functionality.						
<b>Primary Actor</b>	Current guest						
<b>Main Flow</b>	<table> <tr> <th>Step</th><th>Action</th></tr> <tr> <td>1</td><td>Create new Group object.</td></tr> <tr> <td>2</td><td>Group object is stored to appropriate location in memory.</td></tr> </table>	Step	Action	1	Create new Group object.	2	Group object is stored to appropriate location in memory.
Step	Action						
1	Create new Group object.						
2	Group object is stored to appropriate location in memory.						

<b>User Case Name</b>	Add Member						
<b>Goal In Context</b>	One user should be able to manually add designated users as members of a given group. Added members are able to view content posted to the feed of that particular group.						
<b>Successful End Condition</b>	The added members should have access to each of the fields belonging to the designated group. The group also recognize the added user as a member.						
<b>Failed End Condition</b>	The group will not recognize the user as a member, or the newly added user will not be able to access the content local to the particular group.						
<b>Primary Actor</b>	Current guest						
<b>Main Flow</b>	<table> <tr> <th>Step</th><th>Action</th></tr> <tr> <td>1</td><td>Current user designates user as a member of a given group.</td></tr> <tr> <td>2</td><td>Group adds user as a member</td></tr> </table>	Step	Action	1	Current user designates user as a member of a given group.	2	Group adds user as a member
Step	Action						
1	Current user designates user as a member of a given group.						
2	Group adds user as a member						

<b>User Case Name</b>	Update Info						
<b>Goal In Context</b>	A designated user should be able to change the fields that are local to the group.						
<b>Successful End Condition</b>	The values associated with each of the fields within the group will be permanently changed by the user.						
<b>Failed End Condition</b>	The values associated with each of the fields within the group will not be changed upon reloading the program.						
<b>Primary Actor</b>	Current guest						
<b>Main Flow</b>	<table> <tr> <th>Step</th><th>Action</th></tr> <tr> <td>1</td><td>Current guest edits fields</td></tr> <tr> <td>2</td><td>Changes are saved, and previous values are replaced with new values.</td></tr> </table>	Step	Action	1	Current guest edits fields	2	Changes are saved, and previous values are replaced with new values.
Step	Action						
1	Current guest edits fields						
2	Changes are saved, and previous values are replaced with new values.						

<b>User Case Name</b>	Post to Blog										
<b>Goal In Context</b>	The user should be able to add text content to the blog.										
<b>Successful End Condition</b>	A post added to a particular medium - the blog. In addition, the added posts should be retrievable in future sessions.										
<b>Failed End Condition</b>	Added posts either will not appear in the list of user's posts, or will not be added to the database.										
<b>Primary Actor</b>	Current guest										
<b>Main Flow</b>	<table> <tr> <th>Step</th><th>Action</th></tr> <tr> <td>1</td><td>User enters information to post interface that they want to post</td></tr> <tr> <td>2</td><td>User decides whether to submit post or discard post</td></tr> <tr> <td>3</td><td>Upon submission, post will be added to designated post repository</td></tr> <tr> <td>4</td><td>Post is added to appropriate database</td></tr> </table>	Step	Action	1	User enters information to post interface that they want to post	2	User decides whether to submit post or discard post	3	Upon submission, post will be added to designated post repository	4	Post is added to appropriate database
Step	Action										
1	User enters information to post interface that they want to post										
2	User decides whether to submit post or discard post										
3	Upon submission, post will be added to designated post repository										
4	Post is added to appropriate database										

<b>User Case Name</b>	Tweet
<b>Goal In Context</b>	The user should be able to add text content to the tweet.
<b>Successful End Condition</b>	A post added to a particular medium - the tweet. In addition, the added posts should be retrievable in future sessions.
<b>Failed End Condition</b>	Added posts either will not appear in the list of user's posts, or will not be added to the database.
<b>Primary Actor</b>	Current guest
<b>Main Flow</b>	<p><b>Step    Action</b></p> <p>1        User enters information to post interface that they want to post</p> <p>2        User decides whether to submit post or discard post</p> <p>3        Upon submission, post will be added to designated post repository</p> <p>4        Post is added to appropriate database</p>

<b>User Case Name</b>	Post to Group Feed
<b>Goal In Context</b>	The user should be able to add text content to the feed, which is a blog.
<b>Successful End Condition</b>	A post added to a particular medium - the feed/ blog. In addition, the added posts should be retrievable in future sessions.
<b>Failed End Condition</b>	Added posts either will not appear in the list of user's posts, or will not be added to the database.
<b>Primary Actor</b>	Current guest
<b>Main Flow</b>	<p><b>Step    Action</b></p> <p>1        User enters information to post interface that they want to post</p> <p>2        User decides whether to submit post or discard post</p> <p>3        Upon submission, post will be added to designated post repository</p> <p>4        Post is added to appropriate database</p>

<b>User Case Name</b>	See Scrapbook						
<b>Goal In Context</b>	The user should be able to see the scrapbook in html format.						
<b>Successful End Condition</b>	Scrapbook is opened in web browser, displaying all relevant content.						
<b>Failed End Condition</b>	Scrapbook is either not opened in web browser or not displaying.						
<b>Primary Actor</b>	Current guest						
<b>Main Flow</b>	<table> <tr> <th>Step</th><th>Action</th></tr> <tr> <td>1</td><td>User clicks open the scrapbook</td></tr> <tr> <td>2</td><td>Scrapbook load content from user's profile and displays</td></tr> </table>	Step	Action	1	User clicks open the scrapbook	2	Scrapbook load content from user's profile and displays
Step	Action						
1	User clicks open the scrapbook						
2	Scrapbook load content from user's profile and displays						

<b>User Case Name</b>	Chat								
<b>Goal In Context</b>	A user communicates with another user or with a group of users using an instant message.								
<b>Successful End Condition</b>	The message is sent and received by the other user.								
<b>Failed End Condition</b>	Either the message is not sent by the first user, or is not received by the second user.								
<b>Primary Actor</b>	Current guest								
<b>Main Flow</b>	<table> <tr> <th>Step</th><th>Action</th></tr> <tr> <td>1</td><td>One user sends a message directed to another user</td></tr> <tr> <td>2</td><td>That message is sent to the database</td></tr> <tr> <td>3</td><td>The second user then accesses the message sent to them from the database</td></tr> </table>	Step	Action	1	One user sends a message directed to another user	2	That message is sent to the database	3	The second user then accesses the message sent to them from the database
Step	Action								
1	One user sends a message directed to another user								
2	That message is sent to the database								
3	The second user then accesses the message sent to them from the database								

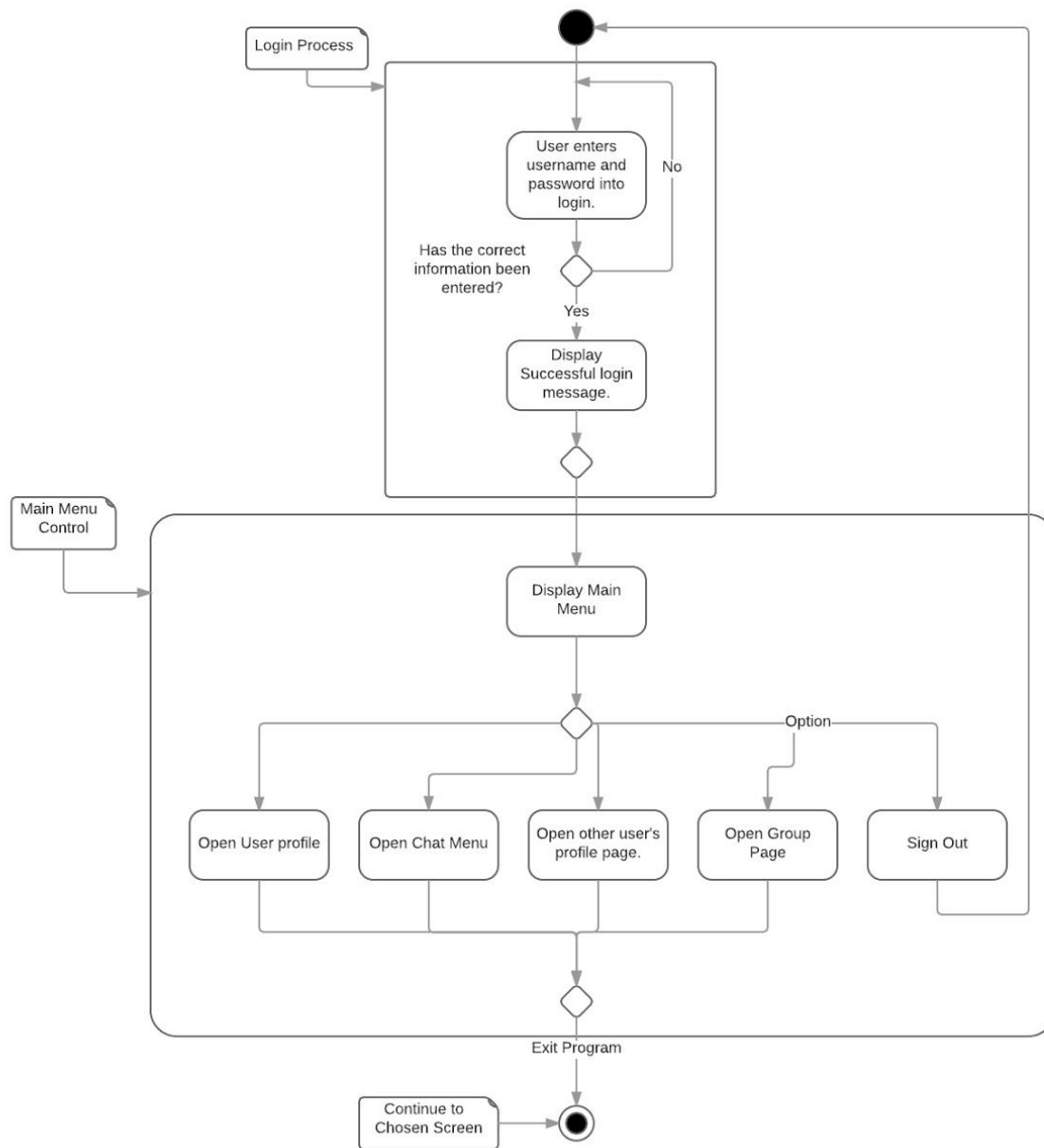


<b>User Case Name</b>	Delete account														
<b>Goal In Context</b>	The user should be able to control whether they want to continue maintaining an account on the rocket ranch system.														
<b>Successful End Condition</b>	The deleted account, and all of its associated information, will be removed from the database.														
<b>Failed End Condition</b>	The deleted account, or at least any portion of its information, will remain in the database.														
<b>Primary Actor</b>	System admin														
<b>Main Flow</b>	<table> <tr> <th>Step</th><th>Action</th></tr> <tr> <td>1</td><td>Account information is deleted</td></tr> <tr> <td>2</td><td>All blog posts belonging to that user are deleted</td></tr> <tr> <td>3</td><td>All messages sent from that user to other users are deleted</td></tr> <tr> <td>4</td><td>The user's scrapbook is deleted</td></tr> <tr> <td>5</td><td>The user is removed from the database</td></tr> <tr> <td>....</td><td></td></tr> </table>	Step	Action	1	Account information is deleted	2	All blog posts belonging to that user are deleted	3	All messages sent from that user to other users are deleted	4	The user's scrapbook is deleted	5	The user is removed from the database	....	
Step	Action														
1	Account information is deleted														
2	All blog posts belonging to that user are deleted														
3	All messages sent from that user to other users are deleted														
4	The user's scrapbook is deleted														
5	The user is removed from the database														
....															

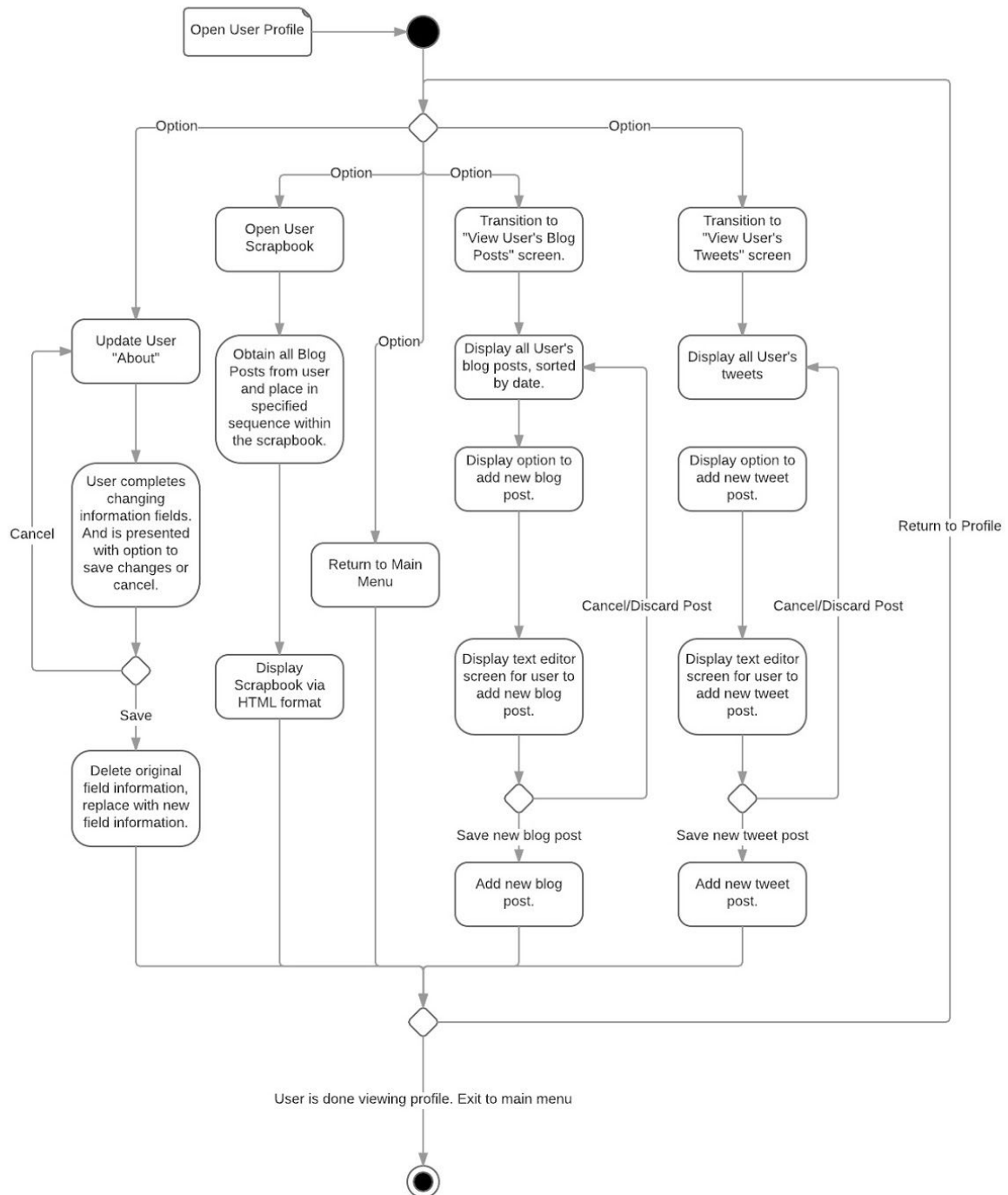
<b>User Case Name</b>	Delete Group								
<b>Goal In Context</b>	The use case is used to remove groups from the Rocket Ranch socialization system.								
<b>Successful End Condition</b>	The group should be removed from memory, and the group should not be able to be accessed in future instances of the program. In other words, the group is removed from the database.								
<b>Failed End Condition</b>	The group is not removed from memory and remains in the database after being deleted.								
<b>Primary Actor</b>	System Admin								
<b>Main Flow</b>	<table> <tr> <th>Step</th><th>Action</th></tr> <tr> <td>1</td><td><b>Members of group are removed from the group</b></td></tr> <tr> <td>2</td><td>Group is deleted from local memory</td></tr> <tr> <td>3</td><td>Group is removed from the database</td></tr> </table>	Step	Action	1	<b>Members of group are removed from the group</b>	2	Group is deleted from local memory	3	Group is removed from the database
Step	Action								
1	<b>Members of group are removed from the group</b>								
2	Group is deleted from local memory								
3	Group is removed from the database								

### III. Process View

#### PROCESS DIAGRAM : LOGIN/MAIN MENU

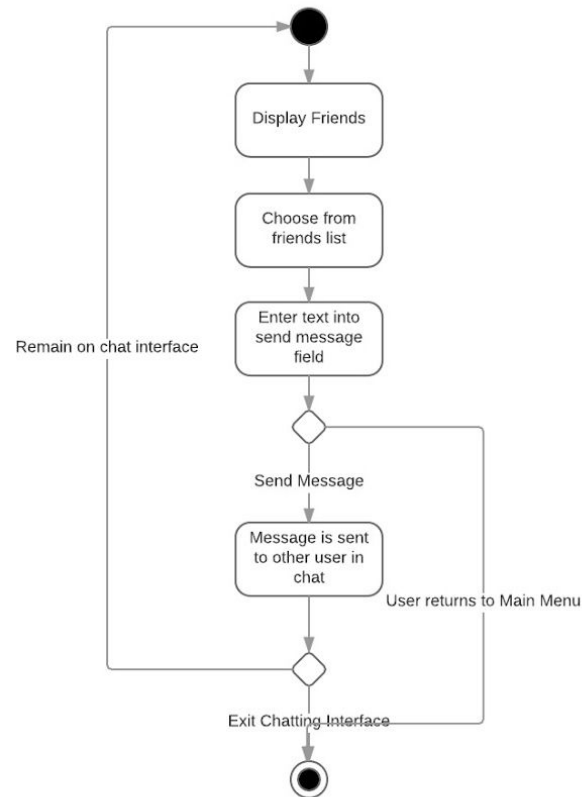


## PROCESS DIAGRAM : USER PROFILE



## PROCESS DIAGRAM : CHAT MENU

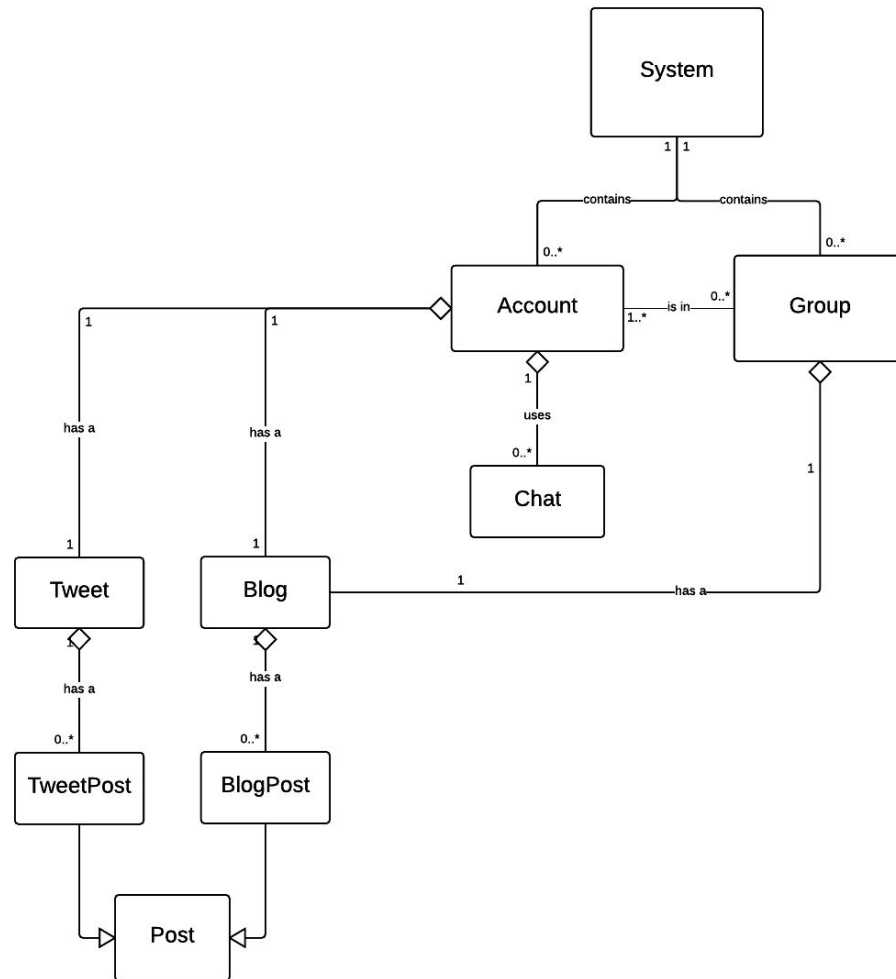
---



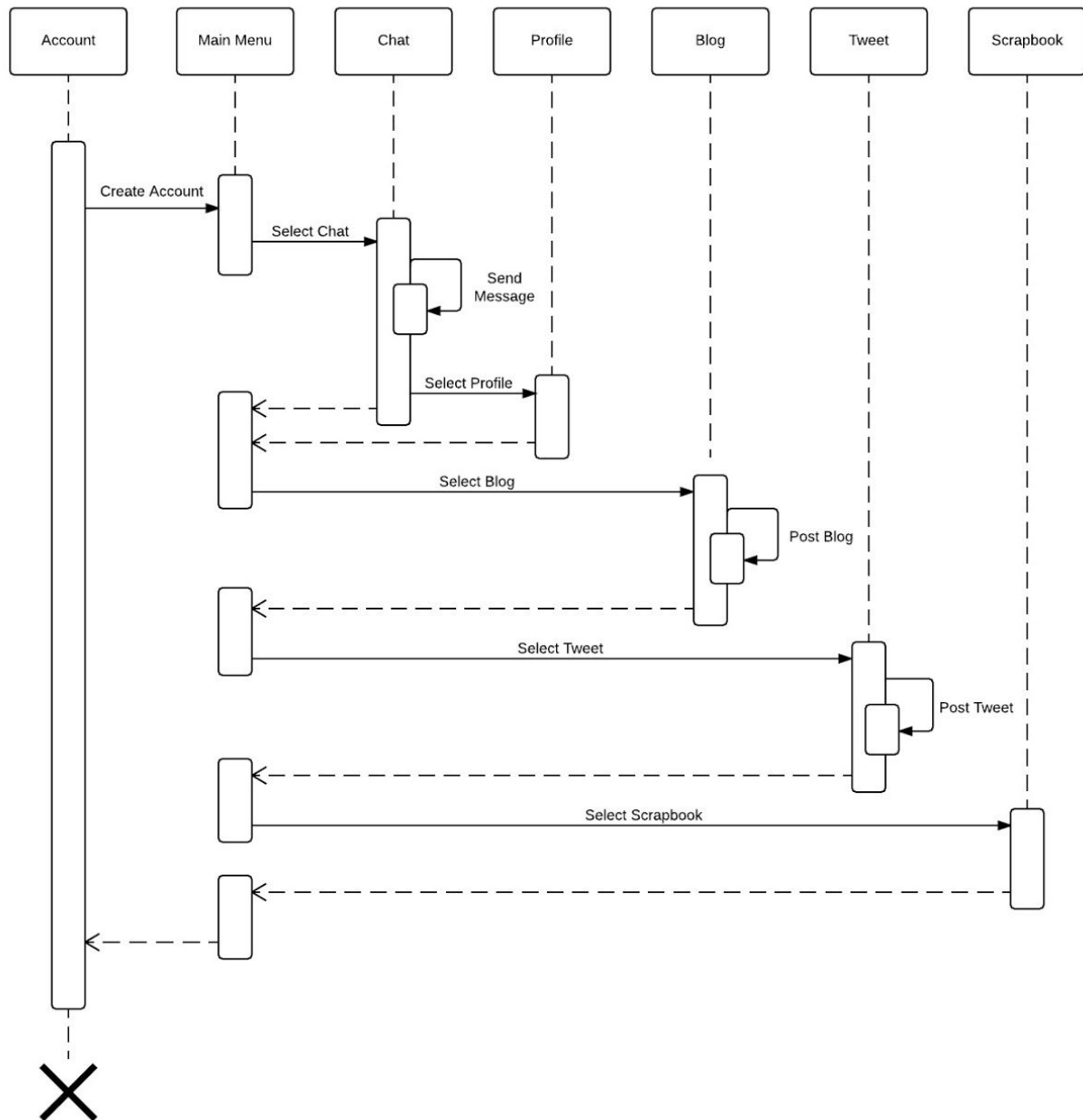
## IV. Logical View

### LOGICAL VIEW: CLASS DIAGRAM, OVERVIEW

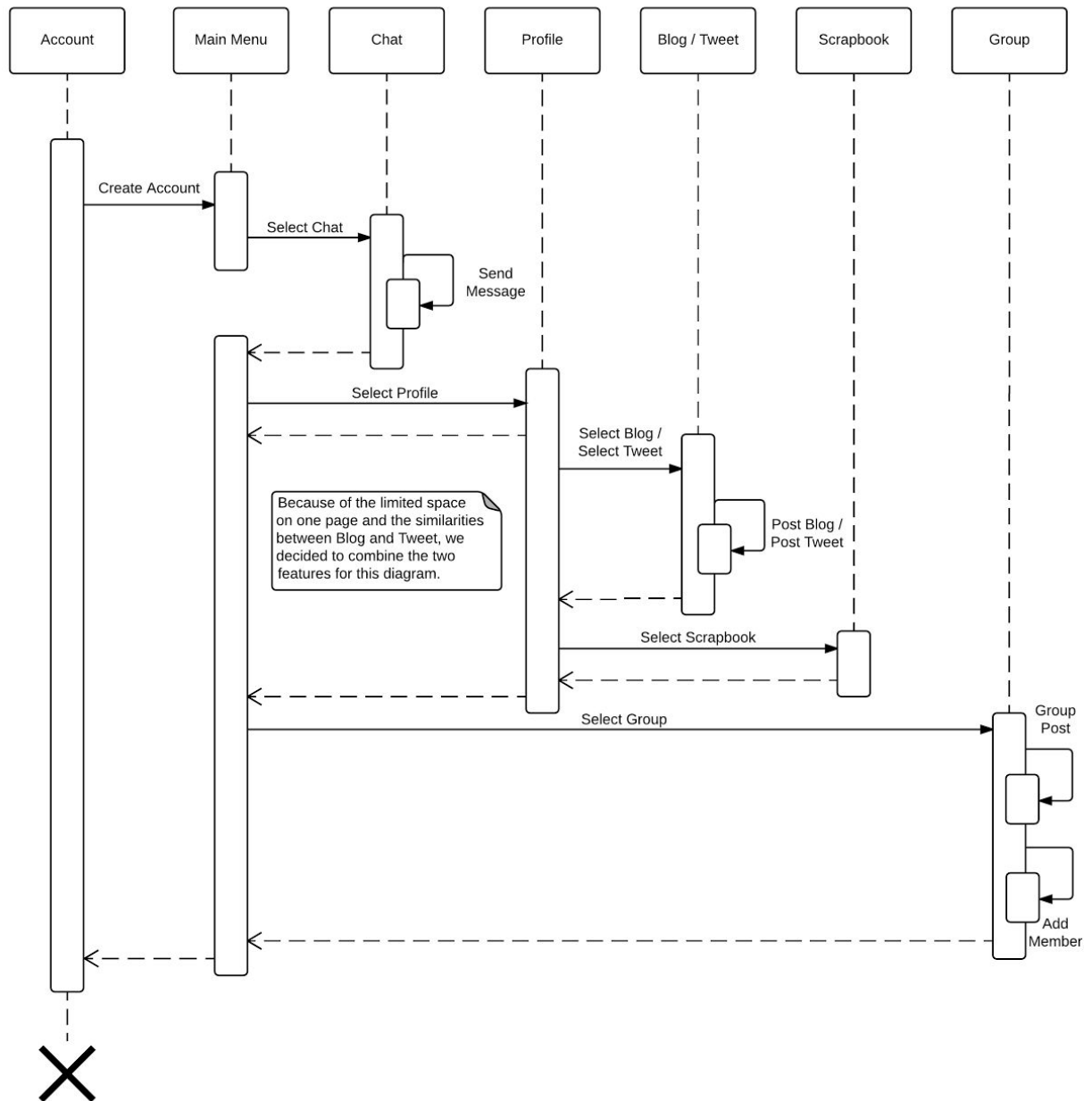
---



## LOGICAL VIEW: TEXT UI SEQUENCE DIAGRAM



## LOGICAL VIEW: GUI SEQUENCE DIAGRAM



## LOGICAL VIEW: SYSTEM CLASS

---

System
<ul style="list-style-type: none"><li>- id: int</li><li>- currentUser : Account *</li><li>- groups: std::vector&lt;Group*&gt;</li><li>- accounts: std::map&lt;QString, Account*&gt;</li><li>- dbm: DbManager*</li><li>- gui : bool</li><li>- textui : bool</li><li>- html_path: std::string</li><li>- openDatabase: void</li></ul>
<ul style="list-style-type: none"><li>+ login: bool</li><li>+ createAccount: bool</li><li>+ addAccount: void</li><li>+ createGroup: Group*</li><li>+ addGroup: void</li><li>+ insertGroup: void</li><li>+ removeGroup: void</li><li>+ refreshSystem: void</li><li>+ linkGroupAccount(): void</li><li>+ retrieveAllAccounts: void</li><li>+ retrieveAllGroups: void</li><li>+ retrieveAllUsersInGroup: void</li><li>+ pairGroupWithUser: bool</li><li>+ addAccountsToGroup: void</li><li>+ deleteAllAccounts: void</li><li>+ deleteAllGroups: void</li></ul>



## LOGICAL VIEW: ACCOUNT CLASS PART 1

---

Account
<ul style="list-style-type: none"><li>- isCurrentGuest : bool</li><li>- isPastGuest : bool</li><li>- isSystemAdmin : bool</li><li>- isGroupAdmin : bool</li><li>- myScrapbook : Scrapbook*</li><li>- myChats : Vector&lt;Chat*&gt;</li><li>- friendList : vector&lt;Account*&gt;</li><li>- groups : vector&lt;Group*&gt;</li><li>- username : QString</li><li>- password : QString</li><li>- firstName : QString</li><li>- lastName : QString</li><li>- gender : QString</li><li>- aboutYourself : QString</li><li>- homeAddress : QString</li><li>- profilePicture : QString</li><li>- mostRecentEmployer : QString</li><li>- age : int</li><li>- phoneNumber : int</li><li>- accountID : int</li><li>- profileUsername : string</li><li>- indexOfProfile : int</li><li>- blogTweetUsername : string</li><li>- myBlog : Blog*</li><li>- myTweet : Tweet*</li><li>- projectsWorkedOn : vector&lt;QString&gt;</li><li>- monthDeparted : int</li><li>- dayDeparted : int</li><li>- yearDeparted : int</li><li>- theSystem : System*</li><li>- dbm : DbManager*</li></ul>

## LOGICAL VIEW: ACCOUNT CLASS PART 2

---

Account
<ul style="list-style-type: none"><li>+ retrieveAllBlogPosts() : void</li><li>+ retrieveAllTweets() : void</li><li>+ retrieveAllChats() : void</li><li>+ retrieveAllMessages() : void</li><li>+ getIsCurrentGuest() : bool</li><li>+ getIsPastGuest() : bool</li><li>+ getIsSystemAdmin() : bool</li><li>+ getIsGroupAdmin() : bool</li><li>+ getUsername() : QString</li><li>+ getPassword() : QString</li><li>+ getProfileUsername() : string</li><li>+ getIndexOfProfile() : int</li><li>+ getChatByPartnerName(talkingTo : QString) : Chat*</li><li>+ getChatVector() : vector&lt;Chat*&gt;</li><li>+ getBlogTweetUsername() : string</li><li>+ getSystem() : System*</li><li>+ setUsername(uName : QString) : void</li><li>+ setPassword(pWord : QString) : void</li><li>+ setSystem(newSystem : System*) : void</li><li>+ setBlogTweetUsername(name : string) : void</li><li>+ setIndexOfProfile(index : int) : void</li><li>+ promoteToCurrentGuest() : void</li><li>+ promoteToPastGuest() : void</li><li>+ promoteToSystemAdmin() : void</li><li>+ promoteToGroupAdmin() : void</li><li>+ getMyBlog() : Blog*</li><li>+ getMyTweet() : Tweet*</li><li>+ setMyBlog(myBlog : Blog*) : void</li><li>+ setMyTweet(myTweet : Tweet*) : void</li><li>+ getMyScrapbook() : Scrapbook</li><li>+ getMyChats() : vector&lt;Chat*&gt;</li><li>+ addChat(newChat : Chat*) : void</li><li>+ insertChat(newChat : Chat*) : void</li><li>+ removeChat(badChat : Chat*) : void</li><li>+ getFriendList() : vector&lt;Account*&gt;</li><li>+ getGroups() : vector&lt;Group*&gt;</li><li>+ joinGroup(newGroup : Group*) : void</li><li>+ leaveGroup(oldGroup : Group*) : void</li><li>+ getFirstName() : QString</li><li>+ setFirstName(fName : QString) : void</li><li>+ getLastName() : QString</li><li>+ setLastName(lName : QString) : void</li><li>+ getGender() : QString</li><li>+ setGender(gend : QString) : void</li><li>+ getAbout() : QString</li><li>+ setAbout(about : QString) : void</li><li>+ getAddress() : QString</li><li>+ setAddress(address : QString) : void</li><li>+ getProfilePicture() : QString</li><li>+ setProfilePicture(picture : QString) : void</li><li>+ getMostRecentEmployer() : QString</li><li>+ setMostRecentEmployer(employer : QString) : void</li><li>+ getAge() : int</li><li>+ setAge(age : int) : void</li><li>+ getPhoneNumber() : int</li><li>+ setPhoneNumber(number : int) : void</li><li>+ getIsCurrentMember() : bool</li><li>+ getIsPastGuest() : bool</li><li>+ getIsSystemAdmin() : bool</li><li>+ getIsGroupAdmin() : bool</li></ul>

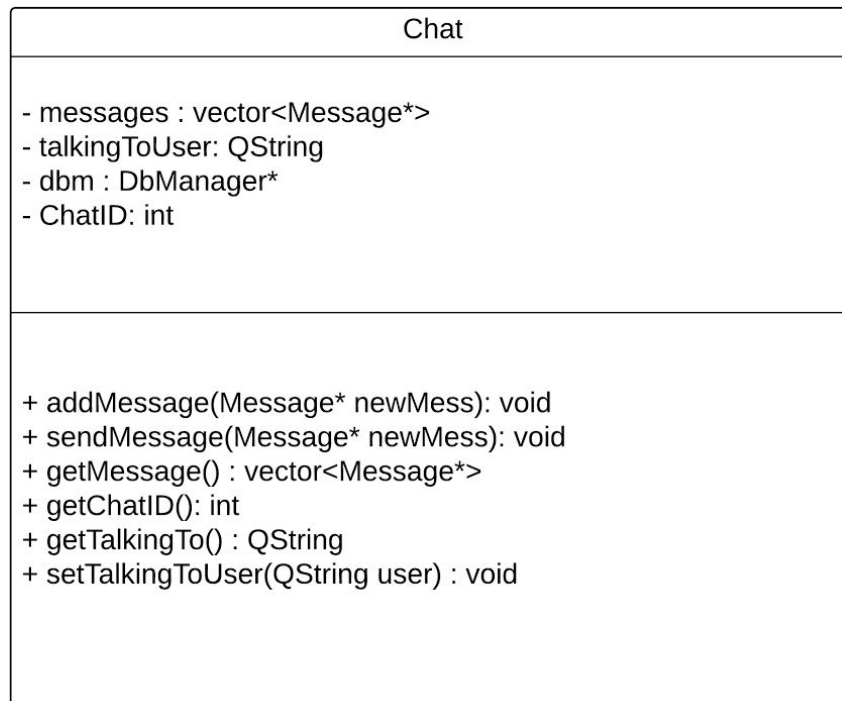
## LOGICAL VIEW: GROUP CLASS

---

Group
<ul style="list-style-type: none"><li>- groupBlog : Feed</li><li>- isActive : bool</li><li>- groupId: int</li><li>- groupMembers : vector&lt;Account&gt;</li><li>- admin : Account*</li><li>- groupName : string</li><li>- currentProject : string</li><li>- aboutGroup: string</li><li>- pastProjects : vector&lt;string&gt;</li><li>- dbm: DatabaseManager*</li></ul>
<ul style="list-style-type: none"><li>+ getBlog() : Blog*</li><li>+ updateFeed(AccountPosted : Account, latestPost : TweetPost) : void</li><li>+ getIsActive() : bool</li><li>+ setStatus(active : bool) : void</li><li>+ getGroupMembers() : vector&lt;Account&gt;</li><li>+ getGroupMemberIDs(): vector&lt;int&gt;</li><li>+ addGroupMember(member : Account) : Account</li><li>+ deleteGroupMember(member : Account) : Account</li><li>+ getAdmin() : GroupAdmin</li><li>+ changeAdmin(newAdmin : GroupAdmin) : void</li><li>+ getGroupName() : string</li><li>+ setGroupName(name : string) : void</li><li>+ getCurrentProject() : string</li><li>+ setCurrentProject(project : string) : void</li><li>+ setAboutGroup(aboutUs: string): bool</li><li>+ newProject(newProject : string) : string</li><li>+ getPastProjects() : vector&lt;string&gt;</li><li>+ retrieveAllPosts(): void</li><li>+ addToPastProjects(project : string) : void</li></ul>

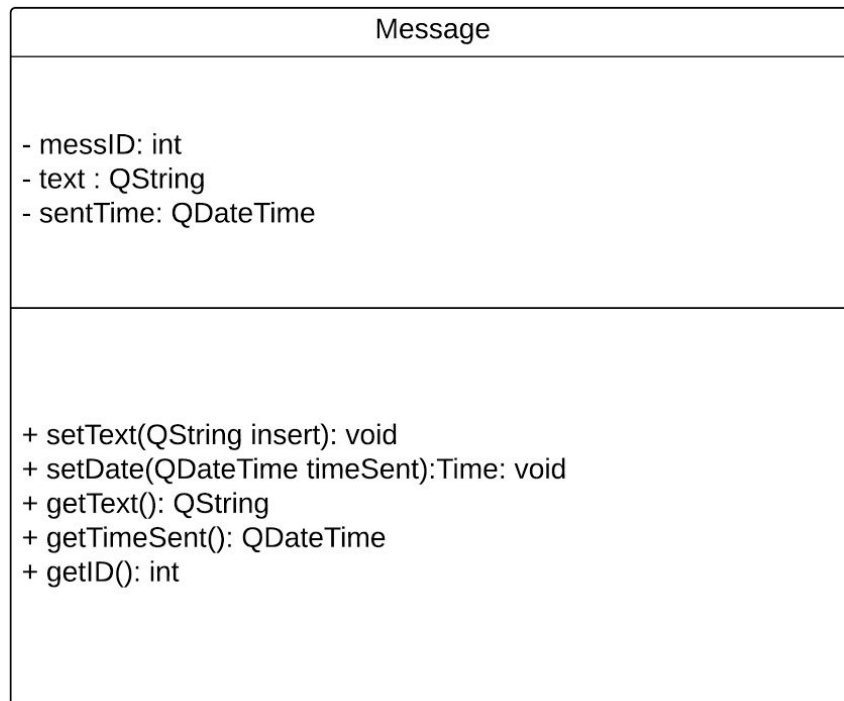
## LOGICAL VIEW: CHAT CLASS

---



## LOGICAL VIEW: MESSAGE CLASS

---



## LOGICAL VIEW: BLOG CLASS

---



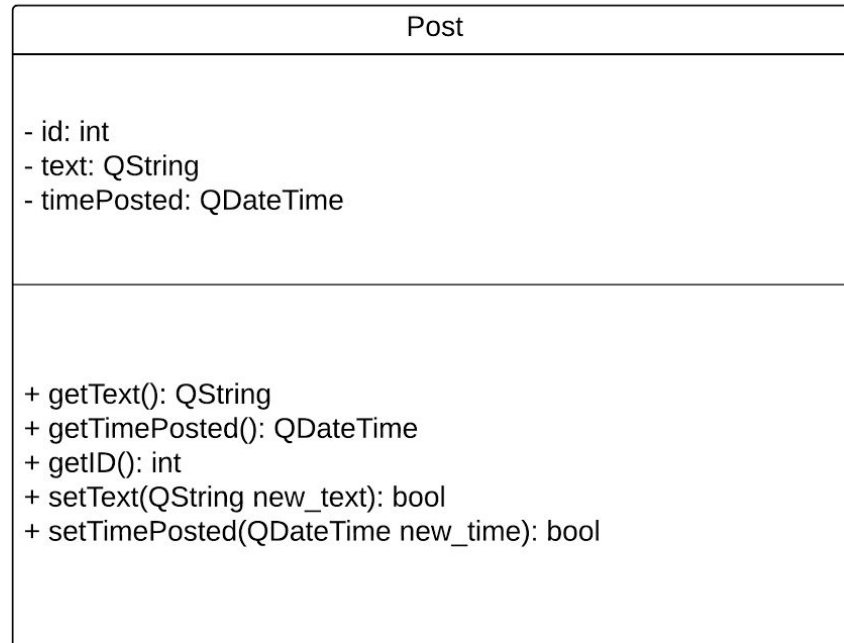
## LOGICAL VIEW: TWEET CLASS

---

Tweet
<ul style="list-style-type: none"><li>- myPosts : vector&lt;TweetPost&gt;</li><li>- tweetID : int</li><li>- dbm : DbManager*</li></ul>
<ul style="list-style-type: none"><li>+ setID(int newID): void</li><li>+ addPost(TweetPost* newPost): void</li><li>+ storePostToDB(TweetPost* newPost): void</li><li>+ deletePost(TweetPost* newPost): void</li><li>+ deleteAllPosts(): void</li><li>+ getMyPosts(): vector&lt;BlogPost*&gt;</li><li>+ getTweetID(): int</li></ul>

## LOGICAL VIEW: POST CLASS

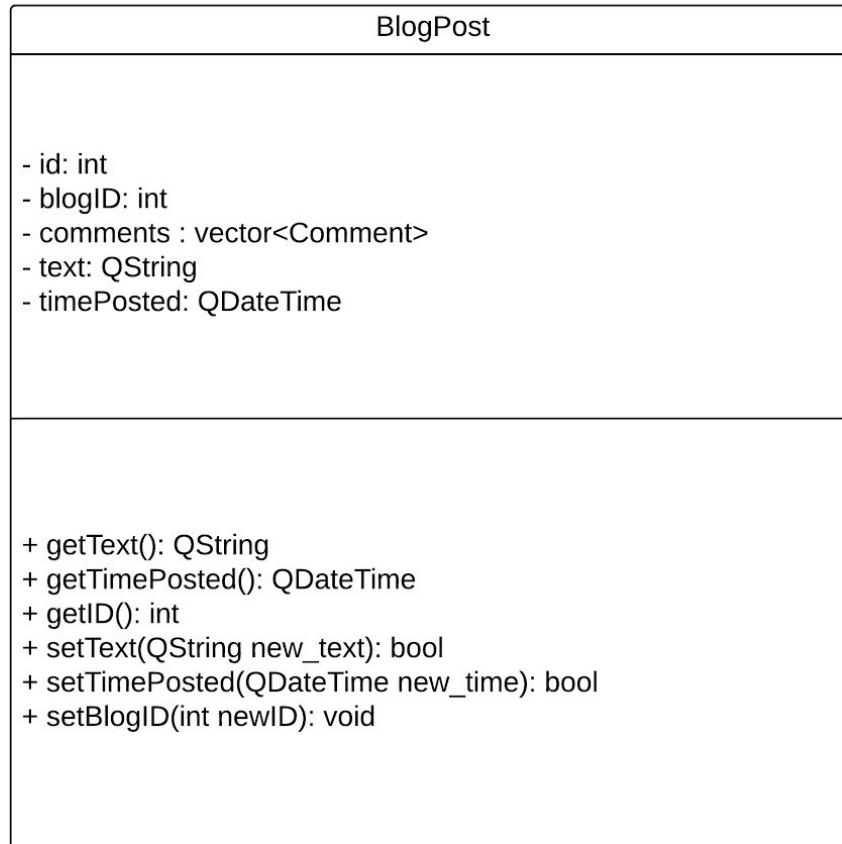
---





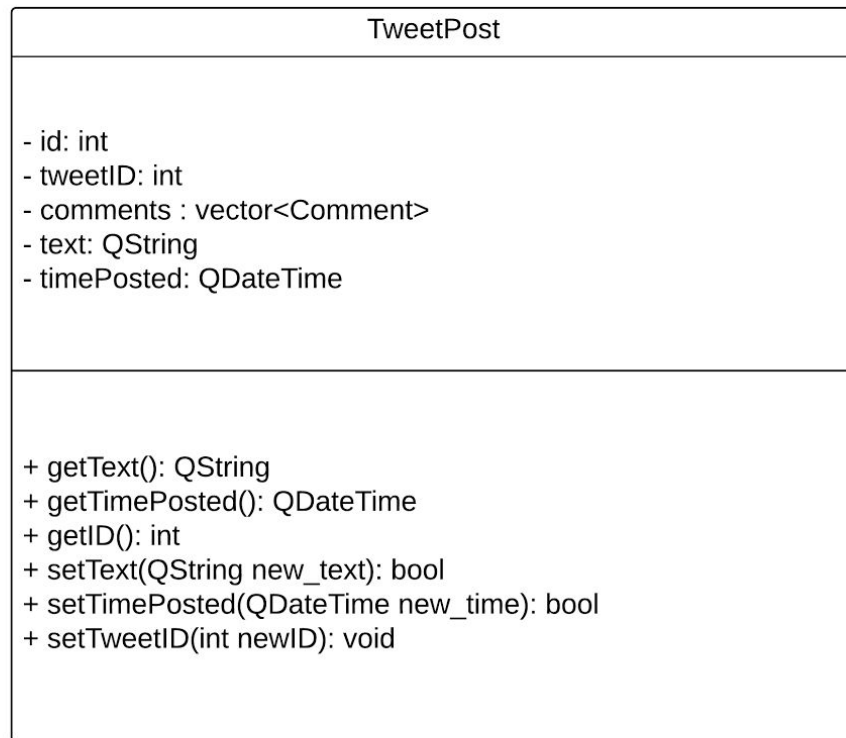
## LOGICAL VIEW: BLOGPOST CLASS

---



## LOGICAL VIEW: TWEETPOST CLASS

---



## LOGICAL VIEW: COMMENT CLASS

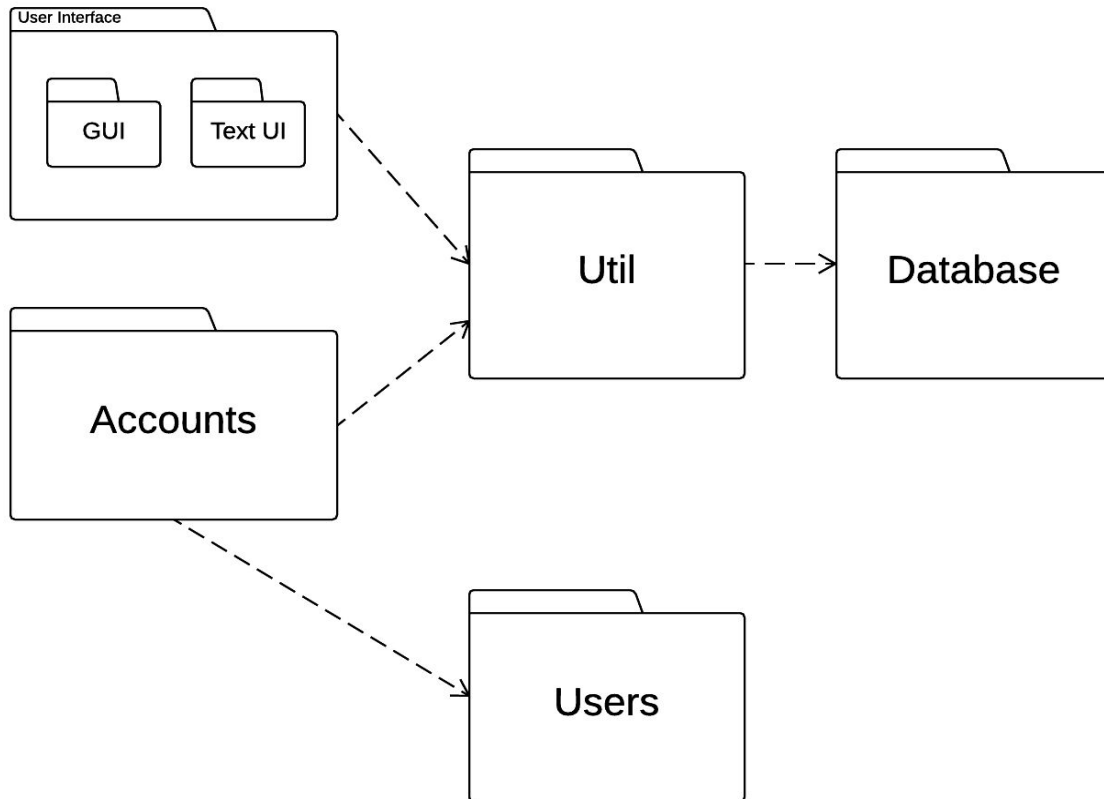
---

Comment
<ul style="list-style-type: none"><li>- text : string</li><li>- date : string</li><li>- time : int</li><li>- fullComment : string</li></ul>
<ul style="list-style-type: none"><li>+ getText() : string</li><li>+ setText(text : string) : void</li><li>+ getDate() : string</li><li>+ setDate(date : string) : void</li><li>+ getTime() : int</li><li>+ setTime(time : int) : void</li><li>+ getFullComment() : string</li><li>+ concatComment() : void</li></ul>

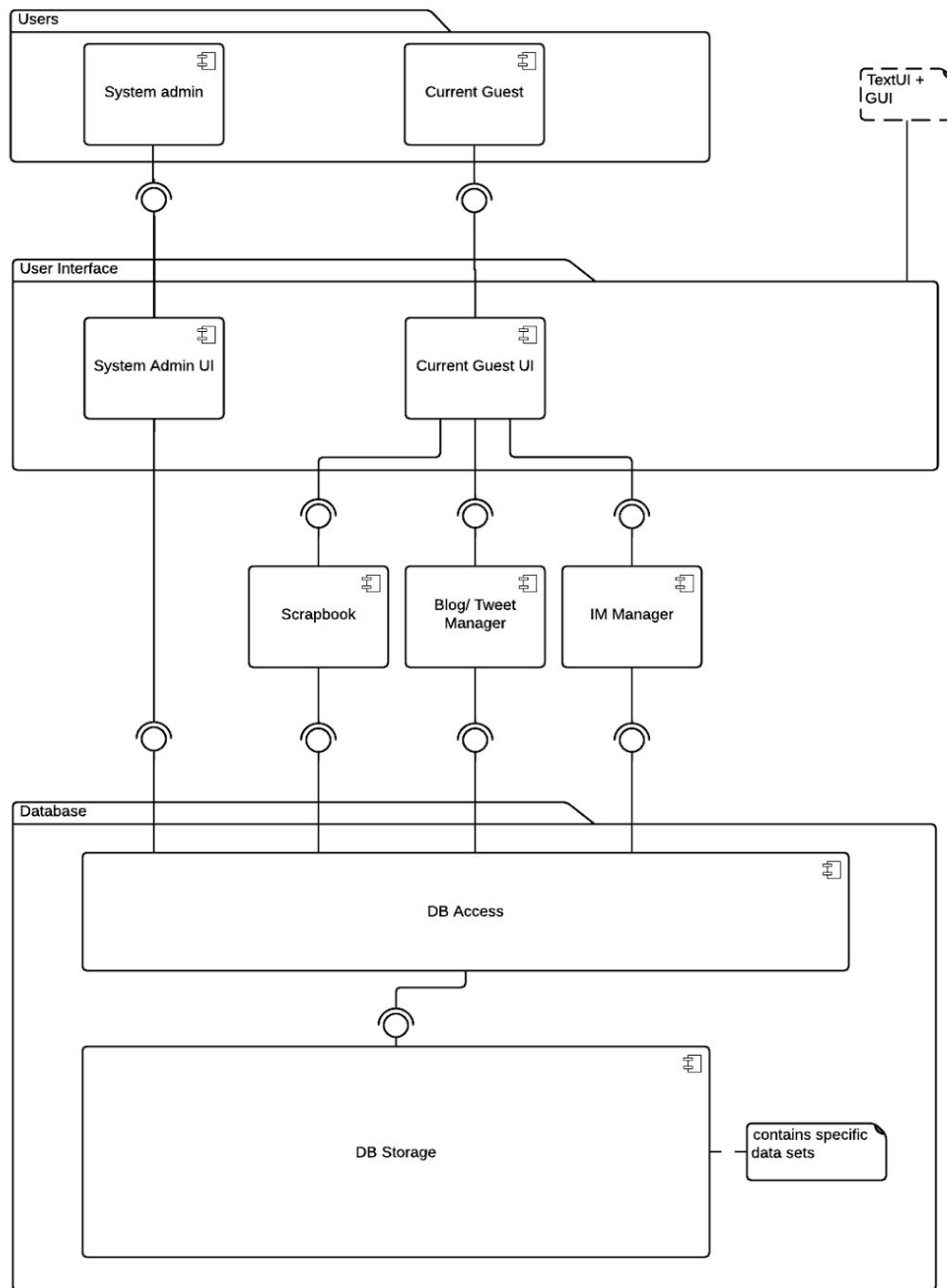
## V. Development View

### DEVELOPMENT VIEW: COMPONENT DIAGRAM OVERVIEW

---

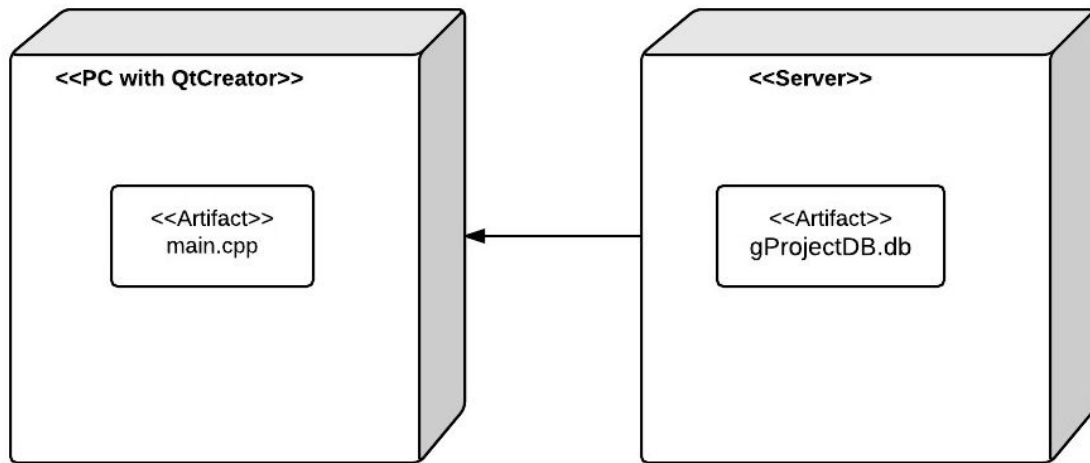


## DEVELOPMENT VIEW: COMPONENT DIAGRAM



## VI. Physical View

### A. Deployment Diagram

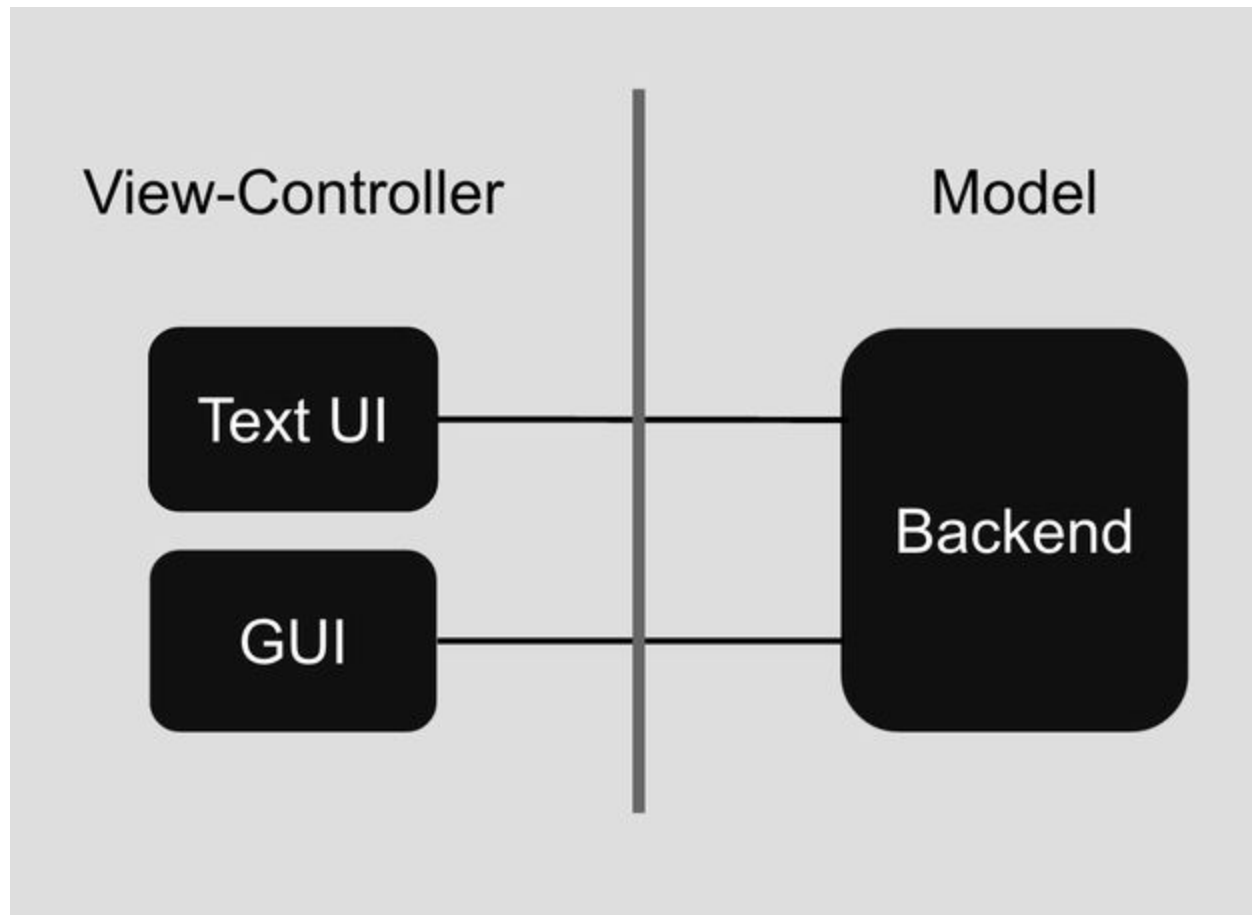


## B. Details

Following is the list of files necessary for operation of program:

project_code.pro	README	textui/
project_code.pro.user	docs/	textui/blogui.cpp
	docs/Doxyfile	textui/blogui.h
model/	docs/Makefile	textui/chatui.cpp
model/account.cpp	docs/docs.pro	textui/chatui.h
model/account.h	docs/main.cpp	textui/choiceui.cpp
model/blog.cpp		textui/choiceui.h
model/blog.h	GUI/	textui/groupui.cpp
model/blogpost.cpp	GUI/GUI.pro	textui/groupui.h
model/blogpost.h	GUI/RR_profile_pic.png	textui/loginui.cpp
model/chat.cpp	GUI/bloggui.cpp	textui/loginui.h
model/chat.h	GUI/bloggui.h	textui/main.cpp
model/comment.cpp	GUI/bloggui.ui	textui/mainmenuui.cpp
model/comment.h	GUI/chatgui.cpp	textui/mainmenuui.h
model/dbmanager.cpp	GUI/chatgui.h	textui/profileui.cpp
model/dbmanager.h	GUI/chatgui.ui	textui/profileui.h
model/feed.cpp	GUI/groupgui.cpp	textui/scrapbookui.cpp
model/feed.h	GUI/groupgui.h	textui/scrapbookui.h
model/feedpost.cpp	GUI/groupgui.ui	textui/screenui.cpp
model/feedpost.h	GUI/images.qrc	textui/screenui.h
model/group.cpp	GUI/login.cpp	textui/textui.pro
model/group.h	GUI/login.h	textui/tweetui.cpp
model/main.cpp	GUI/login.ui	textui/tweetui.h
model/message.cpp	GUI/main.cpp	
model/message.h	GUI/mainmenu.cpp	scrapbook/
model/model.pro	GUI/mainmenu.h	scrapbook/anoceanofs
model/post.cpp	GUI/mainmenu.ui	ky.css
model/post.h	GUI/profilegui.cpp	scrapbook/background.
model/scrapbook.cpp	GUI/profilegui.h	png
model/scrapbook.h	GUI/profilegui.ui	scrapbook/content_bac
model/scrapbookpost.cpp	GUI/rocketicon.png	k.png
model/scrapbookpost.h	GUI/rocketicon_300x166.png	scrapbook/footer.png
model/system.cpp	GUI/scrapbookgui.cpp	scrapbook/index.html
model/system.h	GUI/scrapbookgui.h	scrapbook/rocket.jpg
model/tweet.cpp	GUI/scrapbookgui.ui	scrapbook/tab.png
model/tweet.h	GUI/tweetgui.cpp	
model/tweetpost.cpp	GUI/tweetgui.h	
model/tweetpost.h	GUI/tweetgui.ui	

## VII. Software Architecture



The software architecture used for Rocket Rapport is a modified version of the MVC framework, containing of only two main components: the Model and the View/Controller. The app still has a Model handling the logic of the program. However, we did combine the View and the Controller into one component.

The reason for this is because the user's input from the View and the output from the Model are fairly simple. Thus, there is little need for a medium as the Controller. Based on this judgement, we decided to let the TextUI and GUI talk directly to the backend.

Nonetheless, the fundamental idea of MVC is still applied: Rocket Rapport is broken into major sections with different tasks being weakly linked, in order to increase the robustness and flexibility of each section.



## VIII. Database

The database file the program uses contains 10 different tables. The data fields contained in each and their respective data types are as follows:

### Accounts

AccountID	UserName	Password	AcrcpBkID	BlogID	TweetID	FirstName	LastName
Integer	Varchar	Varchar	Integer	Integer	Integer	Varchar	Varchar

### chatMessages

messageID	chatID	dateTime	text
Integer	Integer	Datetime	Varchar

### feedPosts

feedPostID	FeedID	dateTime	text
Integer	Integer	Datetime	Varchar

### groupUsers

GrpID	UserID
Integer	Integer

### profiles

AccountID	Gender	AbtYsIf	HmAddress	MstRcntEmpl	Age
Integer	Varchar	Varchar	Varchar	Varchar	Integer

**blogPosts**

blogPostID	blogID	dateTime	text
Integer	Integer	Datetime	Varchar

**chats**

ChatID	AccountID	Receiver
Integer	Integer	Varchar

**groupInfo**

GrpID	About
Integer	Varchar

**groups**

GrpID	GrpAdminID	ActiveStat	GrpName	FeedID	GrpInfo
Integer	Integer	Boolean	Varchar	Integer	Varchar

**tweetPosts**

tweetPostID	tweetID	dateTime	text
Integer	Integer	Datetime	varchar

The following classes also store the given objects:

Class	Objects stored within class
System	Group objects Account objects
Account	Blog object Tweet object Chat objects Group objects
Blog	Blog Post objects
Tweet	Tweet post objects
Chat	Message objects

Data retrieval from the database system at the start of the program works in the following way:

1. Program is initiated.
2. New system object is created.
3. All accounts are loaded into the system object.
4. Blog posts, tweet posts, and messages belonging to each account are stored in each account.
5. All groups are loaded into the system.
6. All feed posts belonging to each group are stored in each group.
7. All accounts are paired with groups they belong to.

Here's an example of how the program will obtain data from the database to reconstruct each of the objects in the program. Here, we will load every blog post associated with a particular account to that account.

The program will start by accessing a given accountID. It will then obtain the Blog ID that is associated with the account.

Account ID	UserName	PassWord	Scrapbook ID (int)	Blog ID	Tweet ID	FirstName	LastName
0	mteds	abc	0	0	0	Mike	Teddick
1	vut	xyz	1	1	1	Thanh	Vu

The program will then retrieve every Blog post that contains the given BlogID. The newly created blog will create a new blog post with each of the found data fields.

Blog Post ID	Blog ID	Time + Date	Content
0	0	11:00 May 20th, 2016	I love the rocket ranch!
1	1	11:03 May 20th	I launched three rockets today!

## IX. Test Plan

### A. Objectives

The goal of testing is to make sure that the app's functionalities are correctly implemented, meaning:

- Features are functioning and bug-free.
- Features meets the requirements and expectations of our customer, Harvey.

### B. Scope

Since the project progression consists of three phases, with the first one concerning the design and setup, the main part of the project's testing will lie in the second phase and the third phase, which are the two implementation phases.

The second phase, Core Functionalities, will include all the required features. These "need" features and any sub-features that are used to build them must all be tested carefully and thoroughly. The result of phase 2 should be a functioning app that allows users to:

- Create and manage user accounts
- Create and manage groups
- Interact with each other and with the app environment using Blog, Tweet, and IM features.
- Moderate the app environment, including other user accounts, groups, activities and interactions if the user is a System admin user.

The third phase, Additional Features, extends the app and adds additional "wish" features. Any "wish" feature being added to the app needs to be fully tested for functionality before moving on to the next one. Wish features can include:

- Login feature with password check
- Event object

### C. Testing Strategy

The team will test functionality of a class immediately after finishing writing the class. The team will run both qualitative and quantitative tests on each class. This includes testing for basic functionality, as well as testing for qualitative factors such as ease of use and user friendliness.

In addition to testing individual classes and functions, the team must test large segments of code from different classes that interact with one another. The team will conduct these tests during the process of writing large, interconnected systems. This way, the group will avoid writing large systems, testing them afterwards only to find that the large functions don't work as desired.

#### 1. Team member's responsibilities:

**Test Manager:** Mike Teddick

**Test participants:** All three members of the team will be involved in unit testing. Particularly, each team member will be responsible for the basic unit tests of code that he writes, with the Test Manager overseeing the overall process.

#### 2. Unit Testing

**Methodology:** The team will write the unit tests for each function using Google C++ Testing Framework Gunit.

#### 3. System and Integration Testing

**Methodology:** The team will test each individual connection as they write code connecting classes and functions. This will continue to verify that the interactions between large segments of code are working as desired. The team must verify that the more basic connections work before writing larger connections.

#### 4. Performance and Stress Testing

**Methodology:** The team can measure performance by finding the computational complexity of each of the systems. This can be done by adding  $n$  accounts, each with an increasing number of blog posts, messages, tweets, etc. and measuring the performance of the system as the number of elements included increases. The team can measure time taken to perform different actions under increasing demands from the system memory. This will show how well the software works when placed under stress.

#### 5. User Acceptance Testing

**Methodology:** The team will use the software in accordance to the process diagram created at the beginning of the project. If the team is

able to perform each of the functions in the order described in the process diagram, the software is working in a way that is acceptable for use.

#### **D. Tools and Environment**

**IDE:** Qt Creator

**Framework:** Google C++ Testing Framework Gunit

**Repo Server:** (provided by Prof. Pfaffman)

auto/bunter\_usr11/cs205\_2016\_Grp09/csProjectRepo

**Main Test Environment:** Lafayette CS Department's lab computers

#### **E. Test Schedule**

##### **Milestones:**

- User can create account that is stored in the database.
- User can edit personal information fields within their account.
- Accounts stored in the database can be retrieved from the database with saved personal info fields remaining unchanged since last closed session.
- User can make posts to their designated scrapbook that are stored to the database.
- User must be able to delete content from their scrapbook.
- Changes made to the scrapbook remain in place after scrapbooks are retrieved from the database.
- User should be able to add content to their blog that is stored in the database.
- User should be able to delete designated blog posts from their blog.
- The user's blog posts should be retrievable upon starting a new instance of the program.
- Group Admin type user must be able to create a group.
- Group Admin must be able to add accounts to group. The member of the group must be allowed to view content that is specific to that group.
- Group admin must be able to remove user from a group. Upon removal, the user is unable to view content within the group.
- Group admin must be able to delete group. This means that members must be removed from the group before the group is deleted and removed from the database.
- Accounts must be able to "friend" other accounts. This should grant both accounts the ability to chat with one another.
- Account must be able to send messages to other account that will be viewed by the other account.
- Each account must be able to select an account from either their friends list or from their group to send a chat message to.
- System admin should be allowed to delete any blog post or scrapbook post. In addition, the system admin should be allowed to delete any user from the system.

- Users should be able to write “tweets” that are displayed to the group.
- Users of the type “past guest” should be able to chat with current users and comment on blog posts, but must not be allowed to make blog or scrapbook posts of their own.

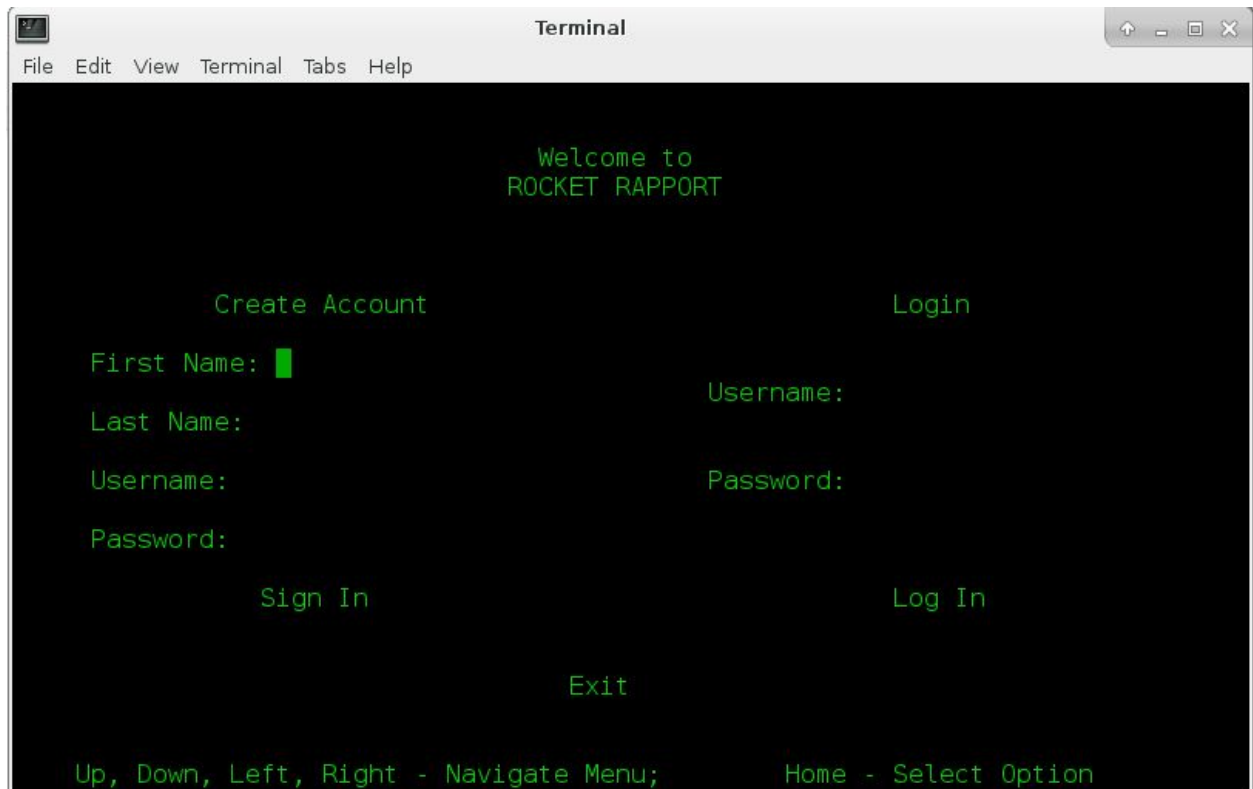
#### **F. Risks/ Assumptions**

Following are the potential risks that have been identified and considered with appropriate counteractions.

	<b>Risk</b>	<b>Contingency plan</b>
<b>1</b>	Any changes to the functionalities may result in negative unit tests.	Constantly update changes between team members. Rewrite test cases if necessary
<b>2</b>	Implementation process goes behind schedule, resulting in delay in testing.	Keep testing closely following implementation, gradually testing functions from small ones, building up to larger ones. Increase “night shift” or reschedule test plan if needed.
<b>3</b>	Having a huge amount of unit tests to do at a time can be daunting and discouraging.	Divide the work of testing between members. Regularly unit test basic, small functions before using them to build bigger ones.

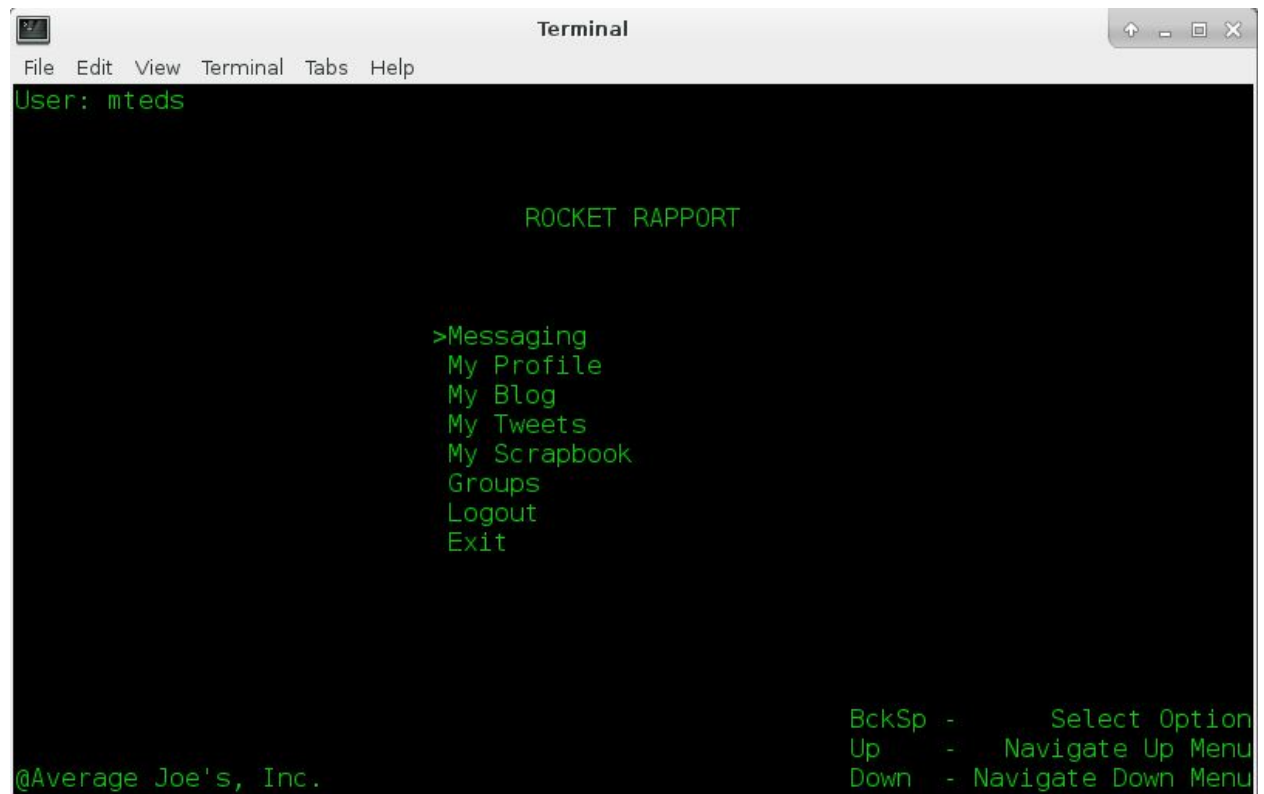
## X. Text UI Preview

### A. Login





## B. Main Menu



```
Terminal
File Edit View Terminal Tabs Help
User: mteds

                                ROCKET RAPPORT

                                >Messaging
                                My Profile
                                My Blog
                                My Tweets
                                My Scrapbook
                                Groups
                                Logout
                                Exit

                                BckSp - Select Option
                                Up - Navigate Up Menu
                                Down - Navigate Down Menu

@Average Joe's, Inc.
```

## C. Messaging

### 1. Typing Message

```
Terminal
File Edit View Terminal Tabs Help

-----
ROCKET RAPPORT MESSAGING                               End - Switch Section
-----
Users                                                     FeelTheBern
Backspace to Chat
Home to View Profile

1-47
Big Donald
>FeelTheBern
Killary
LittleMarco
LowEnergy
LyinTed
SleepyBen
a
jose
mteds

-----
>My Profile      | What's up, Dude? | Press Home
Main Menu        |                  | to Send !
-----
```

### 2. Chat History

```
Terminal
File Edit View Terminal Tabs Help

-----
ROCKET RAPPORT MESSAGING                               End - Switch Section
-----
Users                                                     FeelTheBern
Backspace to Chat
Home to View Profile
2:28 pm May 12 2016
What's up, Dude?

1-47
Big Donald
>FeelTheBern
Killary
LittleMarco
LowEnergy
LyinTed
SleepyBen
a
jose
mteds
2:28 pm May 12 2016
Not much, wbu?

-----
>My Profile      | | Press Home
Main Menu        | | to Send !
-----
```

## D. User Profile

```
Terminal
File Edit View Terminal Tabs Help

-----
                          mteds's Profile
-----

First Name: mike█
Last Name: teddick
Gender:
Home Address:
Most Recent Employer:
Age: 0
Phone Number: 0
About Yourself: I am Mike.

User's Blog      Main Menu      User's Tweets
-----
Up, Down - Navigate Fields;      Home - Select;      Type - Edit
```

## E. Blog

### 1. Typing Blog Post

```
Terminal
File Edit View Terminal Tabs Help

-----
                          mteds's Blog
-----

New Post:
This is post 1█

-----

Former Posts:

-----

                          Main Menu
-----
Up, Down - Navigate Sections; Left, Right - Scroll Posts; Home - Post/Select
```

### 2. Viewing First Blog Post

```
Terminal
File Edit View Terminal Tabs Help

-----
                          mteds's Blog
-----

New Post:

-----

Former Posts:
2:35 pm May 12 2016
This is post 1█

-----

                          Main Menu
-----
Up, Down - Navigate Sections; Left, Right - Scroll Posts; Home - Post/Select
```

### 3. Viewing Second Blog Post

```
Terminal
File Edit View Terminal Tabs Help

-----
mteds's Blog
-----

New Post:

-----

Former Posts:
2:35 pm May 12 2016
This is Post 2█

-----

Main Menu
-----
Up, Down - Navigate Sections; Left, Right - Scroll Posts; Home - Post/Select
```

### 4. Viewing Third Blog Post

```
Terminal
File Edit View Terminal Tabs Help

-----
mteds's Blog
-----

New Post:

-----

Former Posts:
2:35 pm May 12 2016
This is Post 3█

-----

Main Menu
-----
Up, Down - Navigate Sections; Left, Right - Scroll Posts; Home - Post/Select
```

## F. Tweet

### 1. Typing Tweet Post

```
Terminal
File Edit View Terminal Tabs Help

-----
                          mteds's Tweets
-----

New Tweet:
This is a new Tweet Part 2█

-----

Former Posts:
2:39 pm May 12 2016
This is a new Tweet

-----

                          Main Menu
-----
Up, Down - Navigate Sections; Left, Right - Scroll Posts; Home - Post/Select
```

### 2. Viewing First Tweet Post

```
Terminal
File Edit View Terminal Tabs Help

-----
                          mteds's Tweets
-----

New Tweet:
█

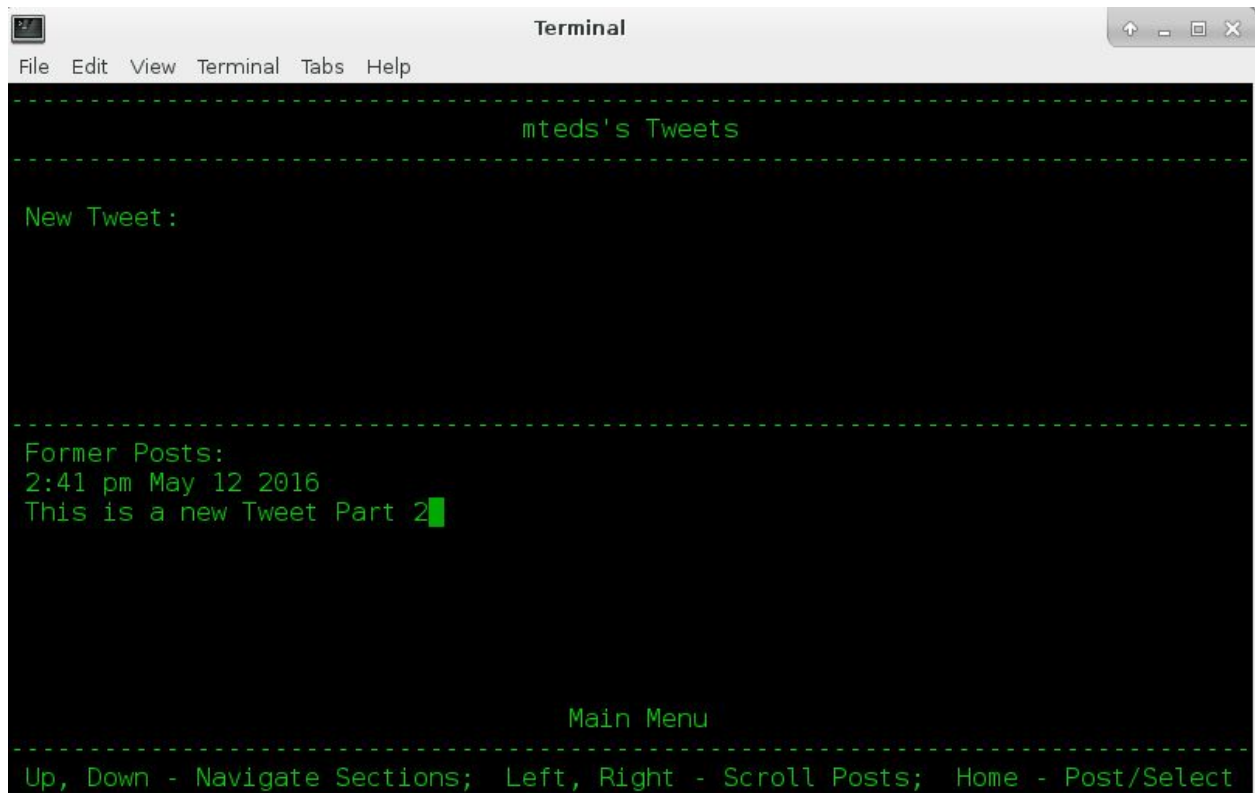
-----

Former Posts:
2:39 pm May 12 2016
This is a new Tweet

-----

                          Main Menu
-----
Up, Down - Navigate Sections; Left, Right - Scroll Posts; Home - Post/Select
```

### 3. Viewing Second Tweet Post



```
Terminal
File Edit View Terminal Tabs Help

-----
mteds's Tweets
-----

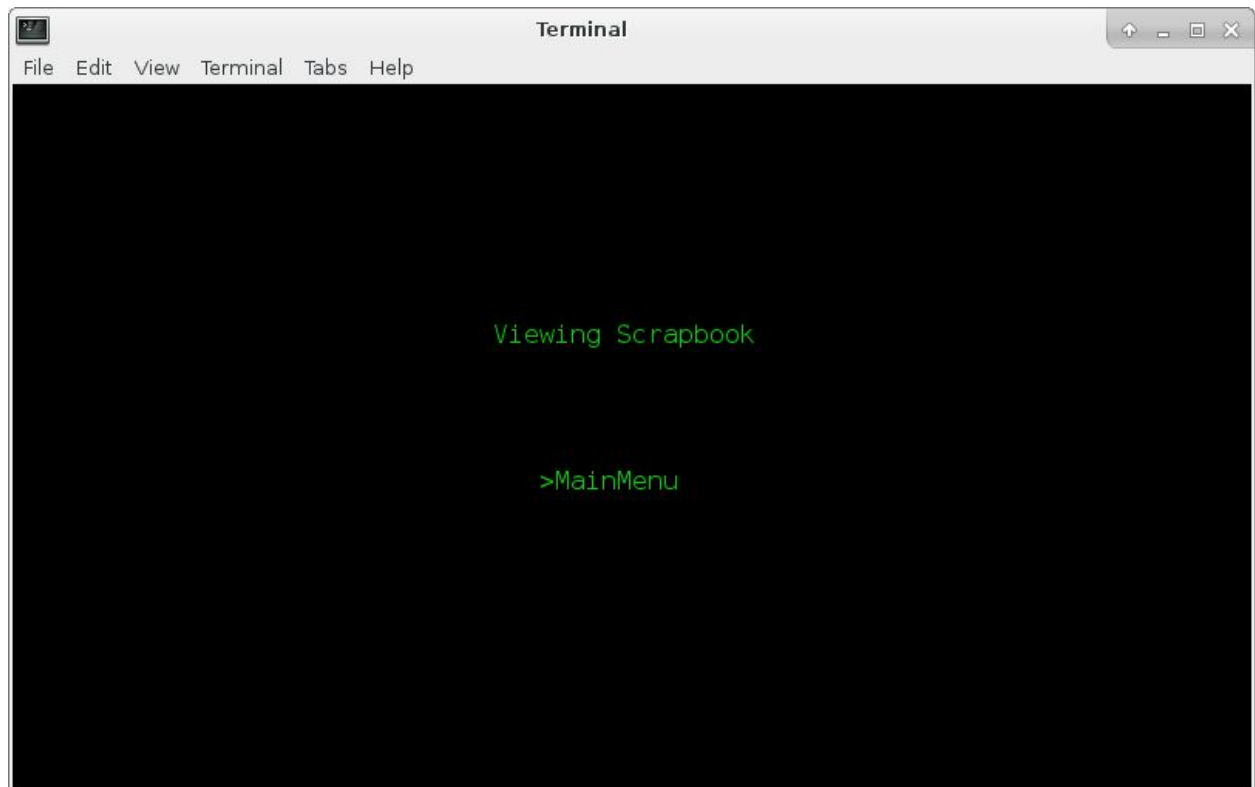
New Tweet:

-----

Former Posts:
2:41 pm May 12 2016
This is a new Tweet Part 2█

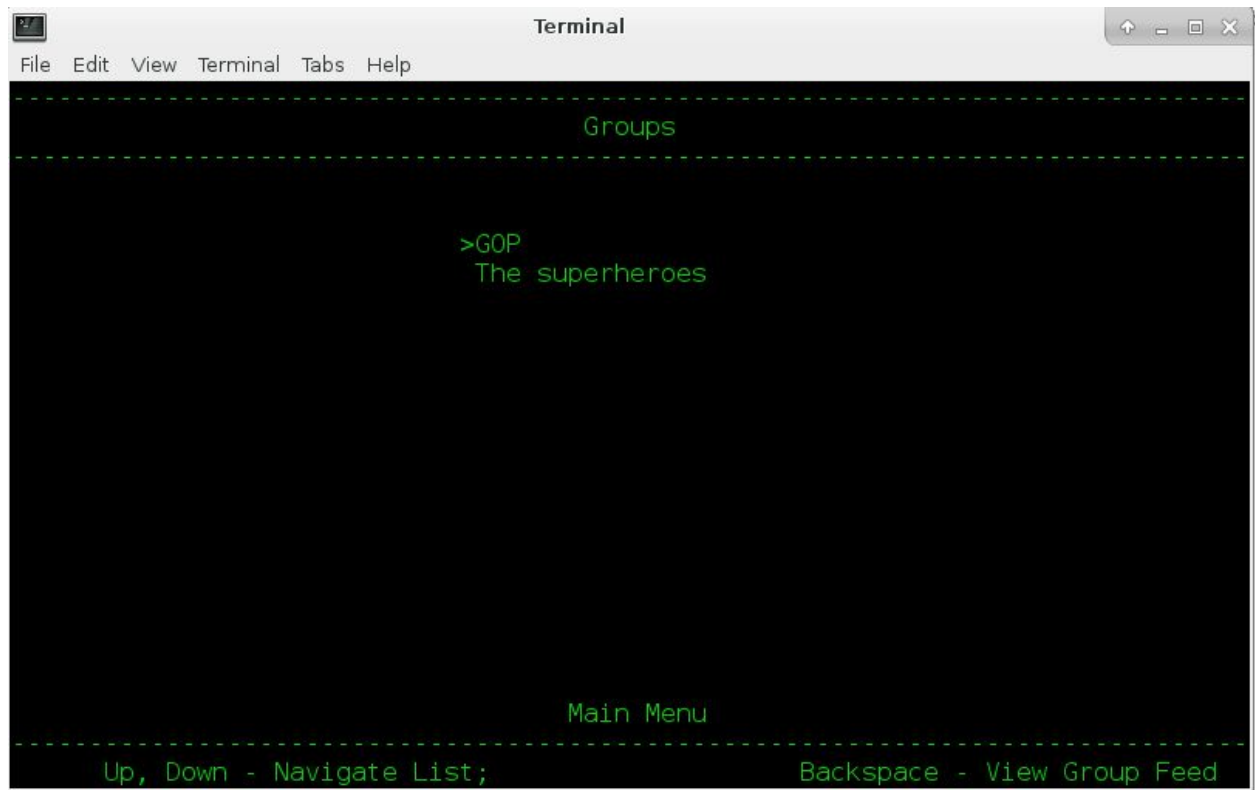
-----
Main Menu
-----
Up, Down - Navigate Sections; Left, Right - Scroll Posts; Home - Post/Select
```

## G. Scrapbook (For static HTML images see Section XII)





## H. Groups



```
Terminal
File Edit View Terminal Tabs Help

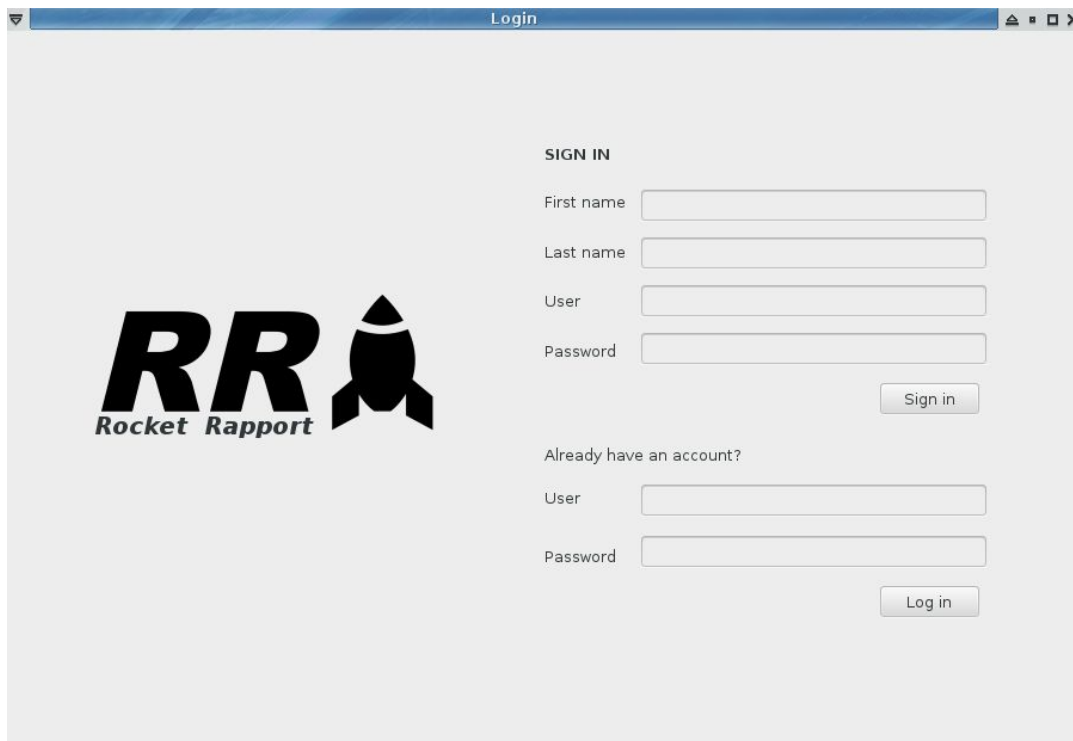
-----
Groups
-----

>GOP
The superheroes

Main Menu
-----
Up, Down - Navigate List;           Backspace - View Group Feed
```

## XI. GUI Preview

### A. Login



The image shows a web browser window titled "Login". On the left side of the page is the "Rocket Rapport" logo, which consists of the letters "RR" in a large, bold, italicized font, followed by a stylized rocket ship icon. Below "RR" are the words "Rocket Rapport" in a smaller, bold, italicized font. On the right side of the page, there are two login sections. The first section is titled "SIGN IN" and contains four input fields: "First name", "Last name", "User", and "Password". Below these fields is a "Sign in" button. The second section is titled "Already have an account?" and contains two input fields: "User" and "Password". Below these fields is a "Log in" button. The entire page has a light gray background.

**RR**  
*Rocket Rapport*

**SIGN IN**

First name

Last name

User

Password

Sign in

Already have an account?

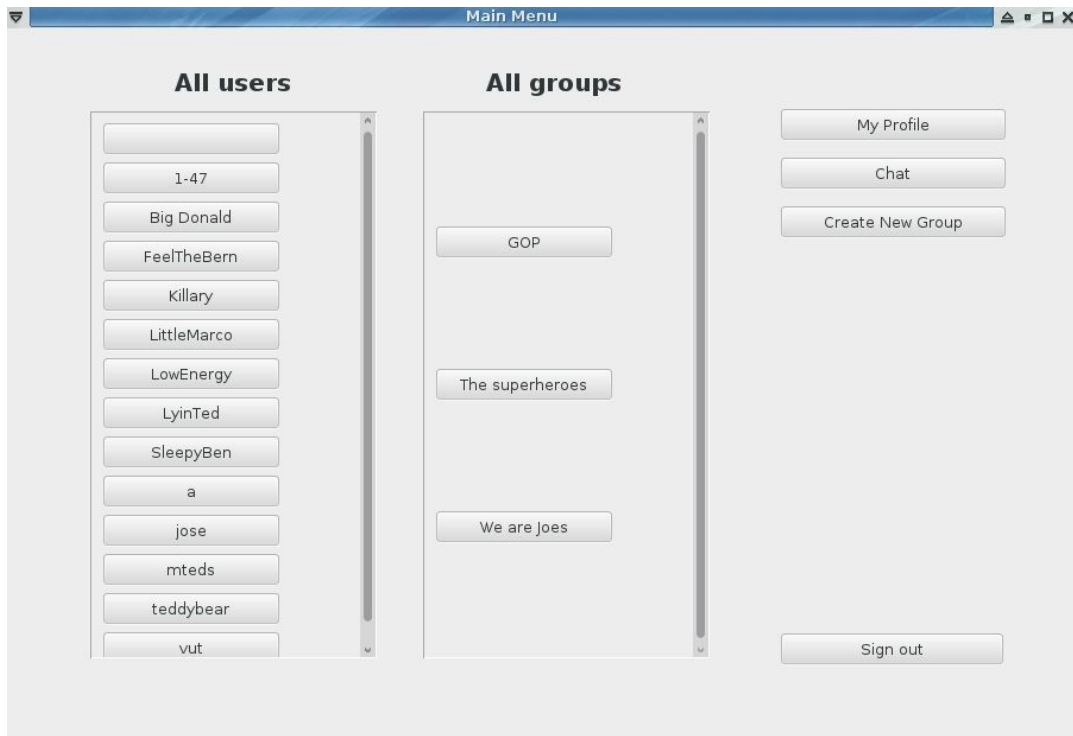
User

Password

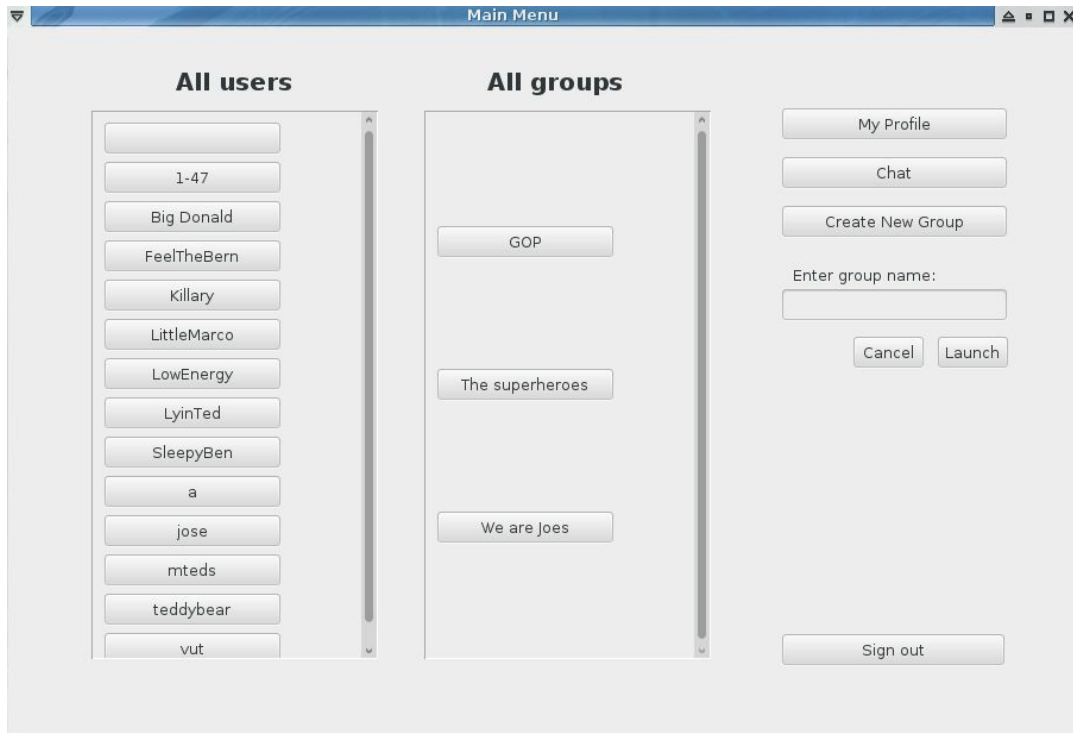
Log in

## B. Main Menu

### 1. Default



### 2. When creating new group

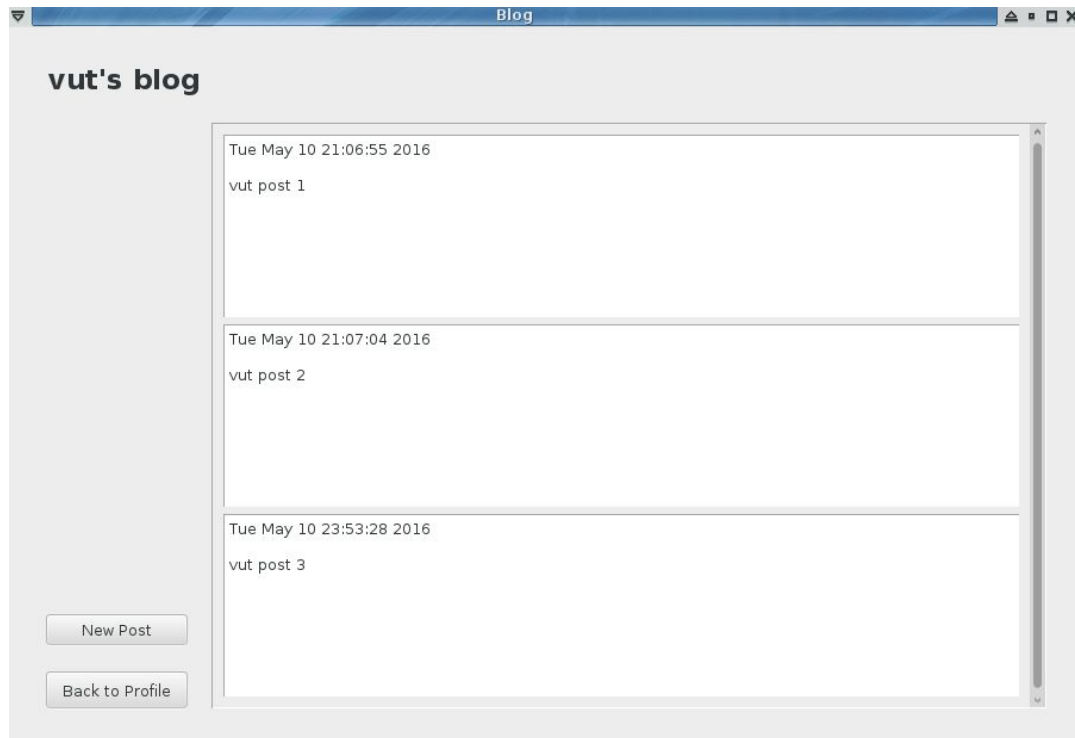


## C. Profile

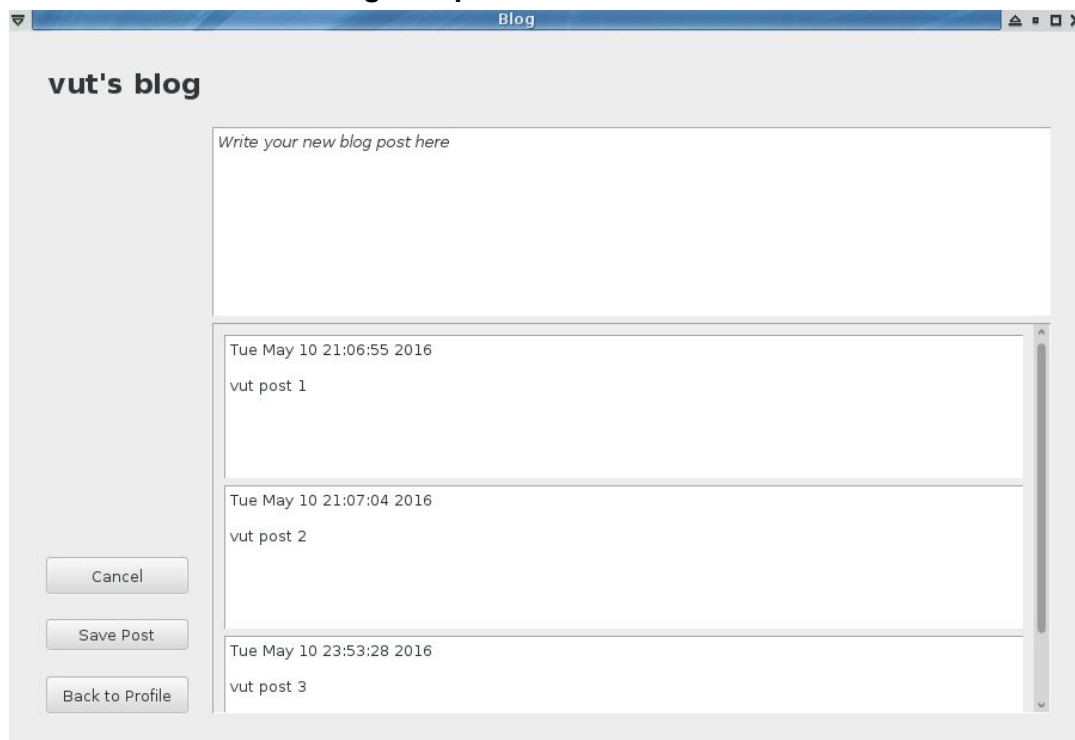


## D. Blog

### 1. Default



### 2. When adding new post



## E. Tweet

Form

**vut's tweet**

vut tweet 3

Fri May 13 00:52:06 2016  
vut tweet 1

Fri May 13 00:52:15 2016  
vut tweet 2

Cancel

Save Post

Back to Profile

## F. Chat

Form

New messages with:

jose

Tue May 10 21:21:40 2016  
it is working

Tue May 10 21:21:44 2016  
i knowwww

Tue May 10 21:21:49 2016  
check the db

Tue May 10 21:21:53 2016  
on it man

Tue May 10 21:22:24 2016  
db looks fine as mars

Tue May 10 22:42:57 2016  
alo

Tue May 10 22:43:09 2016  
hello

Tue May 10 22:43:16 2016  
what

1-47

Big Donald

FeelTheBern

Killary

LittleMarco

LowEnergy

LyinTed

SleepyBen

a

jose

mteds

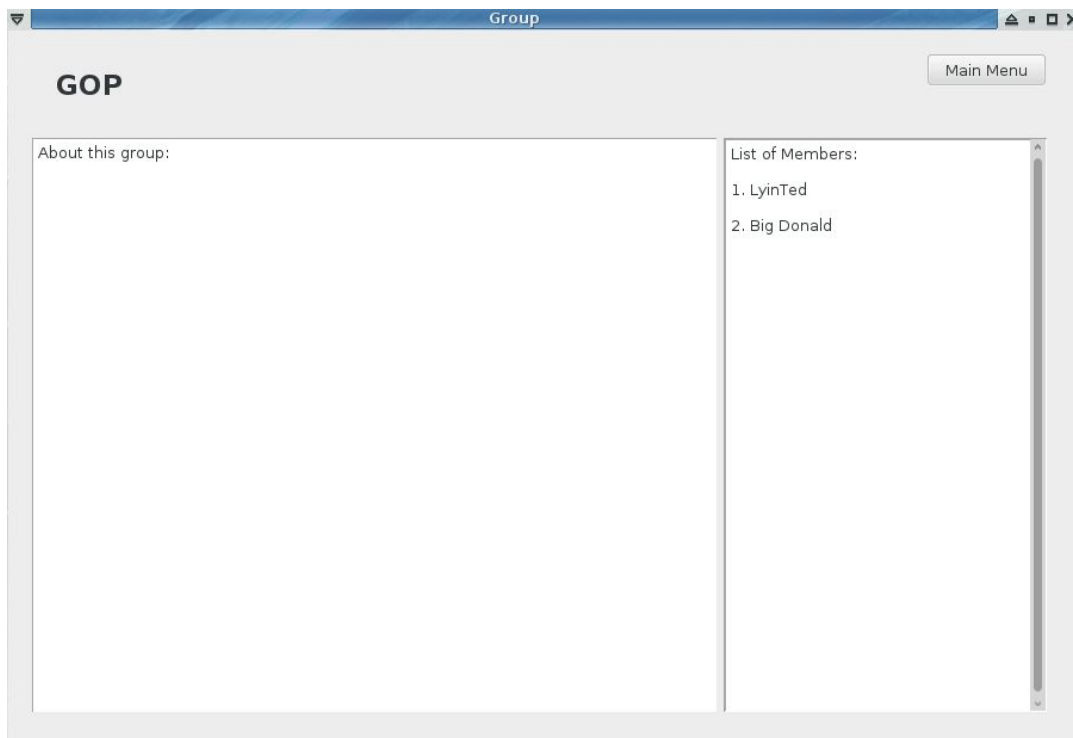
teddybear

Main Menu

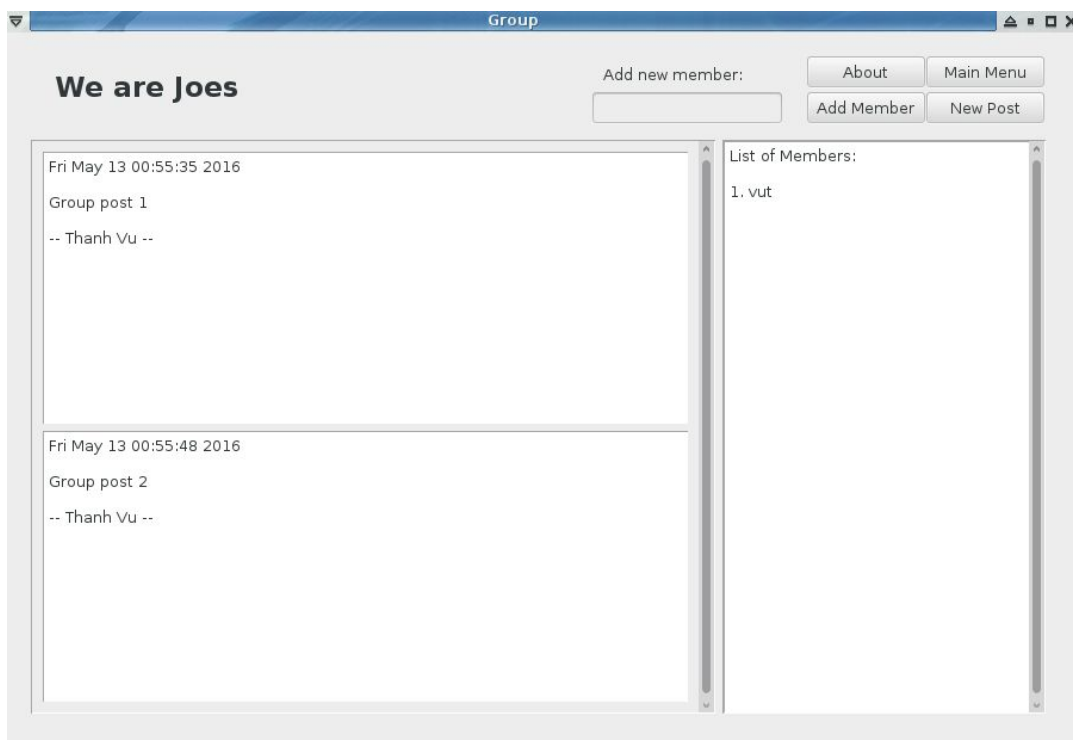
Send

## G. Group

### 1. For non-member




### 2. For member



## XII. Scrapbook Preview

mteds's Scrapbook



First Name

mlr:e

Last Name

teddick

Gender

Home Address

...

Phone Number

0

About Yourself

I am Mlr:e.

Tweets

2:39 pm May 12 2016

This is a new Tweet

2:41 pm May 12 2016

This is a new Tweet Part 2

Blog

2:35 pm May 12 2016

This is post 1

2:35 pm May 12 2016

This is Post 2

2:35 pm May 12 2016

This is Post 3



### **XIII. Team Members and Responsibilities**

The team's breakdown of responsibilities is as followed:

	<b>Team member</b>	<b>Responsibility</b>
1	Thanh Vu	Team leader
2	Andrew Ortiz	Repository/database manager
3	Michael Teddick	Test/documentation manager