PROJECT REPORT

ActiveSpace: Outdoor Enthusiasts Social Network

FIONA SEQUEIRA

ROSHAN SRIDHAR

ActiveSpace is a dynamic web based application driven by a database system to manage application data. It provides a social network experience where users can discuss and share about their outdoor activities with each other in a social setting. A combination of PostgreSQL and PHP is used to design this application.

Contents

GitHub link: https://github.com/roshansridhar/activespace

1. Back-end Design

The back-end for the application consists of a relational schema structured in-order to effectively manage user data. The user data consists of numerous details about the application users and their activities which are explained in detail below. The applications helps to store incoming data, retrieve the stored data and also flexibly share data with other users depending on user requirements.

1.1 Database Design

The design and schema structure is the main pillar and support for the application. This provides the foundation on which the web application is built and executes the required functions. The format and structure is defined in this section.

After careful consideration, the schema consists of the following tables.

Userinfo: This would contain all the user information such as a unique auto-generated id identifying each user, his username, password, name, age, gender, a short bio and his visibility, indicating if he would like his profile contents to be shared only with his friends(visibility=1), friends of friends(FOF) (visibility=2) or everyone (visibility=0)

Location: This contains all the information regarding a location. User can select his location from this table or create a new location, which gets saved in this table.

Multimedia: Users can add multimedia content to their profile such as photos/videos/audio.

Posts: Users can add posts to their profile or on their friend's profiles. They can view posts from their friends, FOFs or everyone on the network subject to their visibility preference. They could look for posts if its contents contain a keyword as well.

Diaryentry: Users can share diary entries on their profile which would be like a small note with a title and location. If the user desires he can add a photo or video from his multimedia collections to be shared with his friends. Again, the user can view diary entries of his friends, FOF or anyone subject to their visibility settings.

Diary_comments: This table contains all the comments made for various diary entries. Diary_id is a foreign key which maps to the diary entry the comment was made on. User_id is the user who placed the comment.

Diary_likes: This table contains all the likes made for various diary entries. Diary_id is a foreign key which maps to the diary entry the like was made on. User_id is the user who placed the like.

Friendrelation: This table maintains the relations between users, including the events o friend request and friend accept. In the event a friend request is declined we would delete the friend relation accordingly.

Events: It is a list of event activities happening in various locations and its description.

event_members: This table consists of a list of users attending an event, the event_id is a foreign key to the events table.

1.2. Entity-relationship diagram

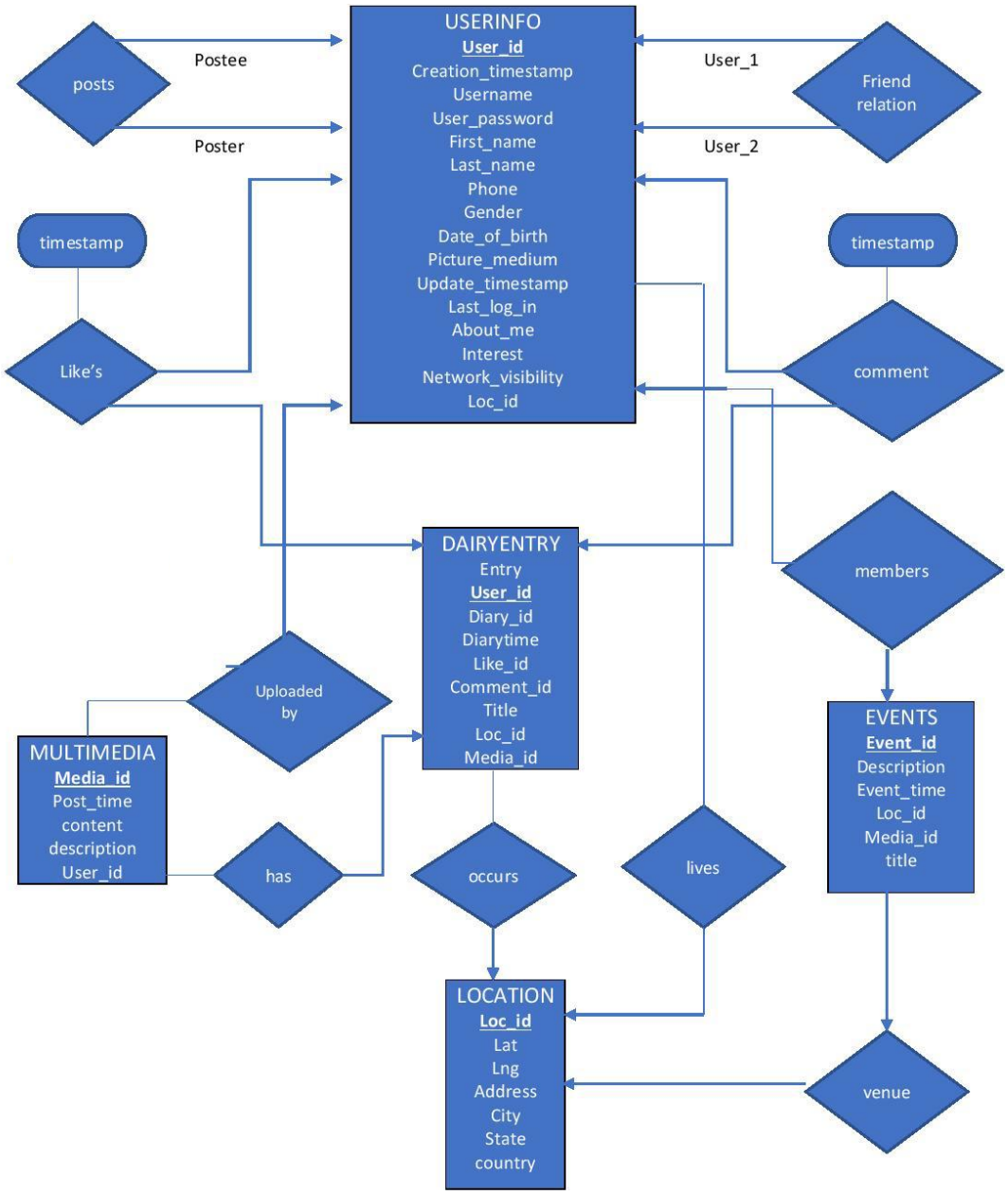


Figure: Entity-relationship diagram of ActiveSpace.

The entity-relationship diagram, or ER diagram, as shown above, gives us an overall understanding of the structure of the schema for our application. It consists of blocks which denote the tables present in the database. As we can see, most tables revolve around the *Userinfo* table. The *Userinfo* table plays an integral part of the application. This is due to the fact that a social network relies mainly on its users and their connections. The *Userinfo* table contains all the information pertaining to a user. The other major tables include *Location*, *Multimedia*, *Diary_entry* and *Events*. The auxiliary tables define connections and additional information for these major tables. They include *likes*, *comments*, *posts*, *friendrelations*, *event_members* etc. They are present to reduce redundancy of information, which in turn provide a multitude of advantages like space conservation, inconsistency during table operations etc.

1.3. Validation and Sample Queries

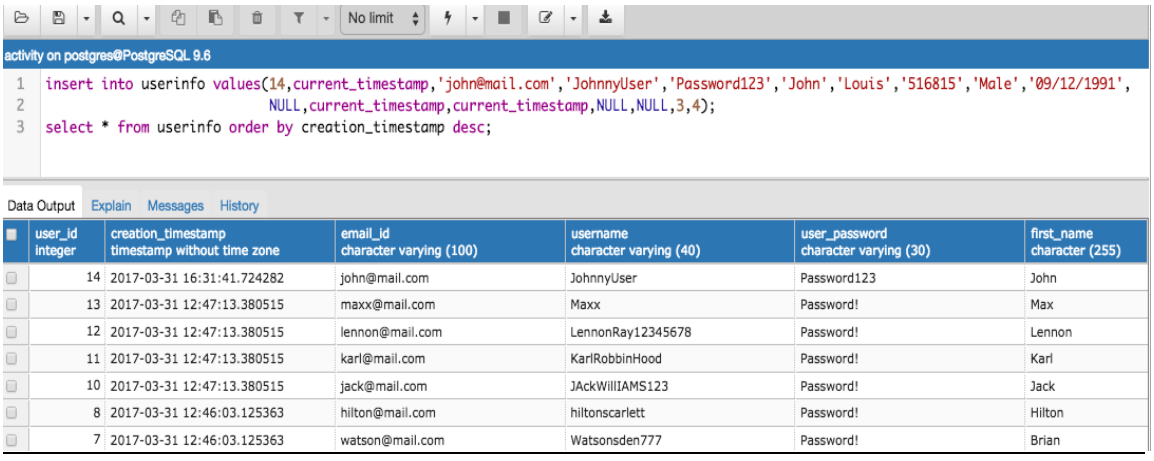
From the ER diagram, it can be noted that there are distinguishable attributes, numerous references and relationships between data, dependences between tables and many other components present in the model. The creation of an efficient data model must be verified and validated by using sample data. The sample test data must be designed carefully to examine and verify all functionality.

We will be validating our sample data if it can be validated against the required functionalities in the following three parts using our user Andrea Jovi, with userid=1.

Part 1:

Content Posting:

<u>New user John needs to sign up, to create his profile</u>



user_id integer	creation_timestamp timestamp without time zone	email_id character varying (100)	username character varying (40)	user_password character varying (30)	first_name character (255)
14	2017-03-31 16:31:41.724282	john@mail.com	JohnnyUser	Password123	John
13	2017-03-31 12:47:13.380515	maxx@mail.com	Maxx	Password!	Max
12	2017-03-31 12:47:13.380515	lennon@mail.com	LennonRay12345678	Password!	Lennon
11	2017-03-31 12:47:13.380515	karl@mail.com	KarlRobbinHood	Password!	Karl
10	2017-03-31 12:47:13.380515	jack@mail.com	JAckWillIAMS123	Password!	Jack
8	2017-03-31 12:46:03.125363	hilton@mail.com	hiltonscarlett	Password!	Hilton
7	2017-03-31 12:46:03.125363	watson@mail.com	Watsonsden777	Password!	Brian

Existing user Andrea would like to edit her profile

Here, we firstly create a trigger for the updtd_timestamp attribute of the userinfo table, so that for every update the last updated timestamp gets refreshed.

activity on postgres@PostgreSQL 9.6

```
1  ALTER TABLE userinfo
2      ADD updtd_timestamp TIMESTAMP;
3
4  ALTER TABLE userinfo
5      ALTER COLUMN updtd_timestamp
6          SET DEFAULT CURRENT_TIMESTAMP;
7
8  UPDATE userinfo
9      SET updtd_timestamp=CURRENT_TIMESTAMP;
10
11 CREATE OR REPLACE FUNCTION updtd_timestamp_column()
12         RETURNS TRIGGER AS '
13   BEGIN
14     NEW.lastmodified = NOW();
15     RETURN NEW;
16   END;'
17   language 'plpgsql';
18
19 CREATE TRIGGER update_timestamp_time BEFORE UPDATE
20   ON userinfo FOR EACH ROW EXECUTE PROCEDURE
21   updtd_timestamp_column();
```

Data Output | Explain | Messages | History

```
CREATE TRIGGER

Query returned successfully in 117 msec.
```

activity on postgres@PostgreSQL 9.6

```
1  update userinfo set about_me='Hi there everyone! Its nice to meet you' where user_id=1;
2  select creation_timestamp,about_me,username,updtd_timestamp from userinfo where user_id=1;
```

Data Output | Explain | Messages | History

creation_timestamp timestamp without time zone	about_me text	username character varying (40)	updtd_timestamp timestamp without time zone
2017-03-31 12:14:27.59014	Hi there everyone! Its nice...	AndreaJovi123	2017-03-31 17:04:36.71594

Andrea wants to post an image

Since to insert images we need an external application such as PHP to obtain binary format of the data and store the following in the required column.

So we replace the column value with NULL for now

```
8
9  insert into multimedia values(30,current_timestamp,NULL,'description text',4);
10 select * from multimedia order by post_time desc;
```

Data Output | Explain | Messages | History

media_id integer	post_time timestamp without time zone	content bytea	description text	user_id integer
30	2017-03-31 22:51:26.98741	[null]	description text	4
2	2017-03-31 12:58:37.324923	[null]	text1	1
3	2017-03-31 12:58:37.324923	[null]	text1	7
4	2017-03-31 12:58:37.324923	[null]	text1	9
5	2017-03-31 12:58:37.324923	[null]	text1	2
6	2017-03-31 12:58:37.324923	[null]	text1	1
7	2017-03-31 12:58:37.324923	[null]	text1	4
8	2017-03-31 12:58:37.324923	[null]	text1	1
9	2017-03-31 12:58:37.324923	[null]	text1	3
10	2017-03-31 12:58:37.324923	[null]	text1	1
11	2017-03-31 12:58:37.324923	[null]	text1	6

Andrea wants to add a new entry to their diaries.

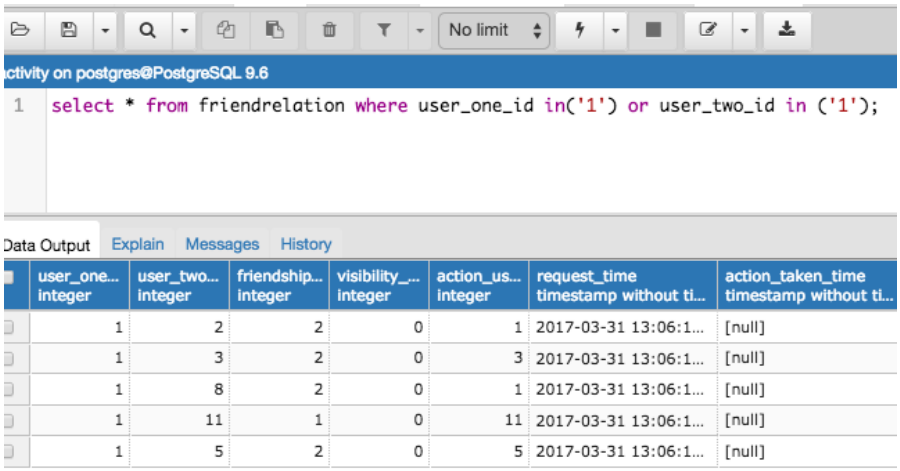
```
activity on postgres@PostgreSQL 9.6
1  insert into diaryentry values('The Sun is shining all so bright. Who wants to hangout?',13,1,
2                                current_timestamp,'Such a beautiful day!',5,NULL);
3  select * from diaryentry where user_id=1 order by diarytime desc;
```

Data Output | Explain | Messages | History

entry text	diary_id integer	user_id integer	diarytime timestamp without time zone	title character varying (100)	loc_id integer	media_id integer
The Sun is shining all so bri...	13	1	2017-03-31 17:29:30.0335...	Such a beautiful day!	5	[null]
description	2	1	2017-03-31 12:59:14.1647...	title	2	15
Hiking plans for Grand Cany...	5	1	2017-03-31 12:59:14.1647...	title	5	10

Part 2:
Friendship:

Our friend relation table has an attribute friendship_status= 1 for friendship requested, 2 for friendship accepted. We currently filter our sample data to user_id=1 to check Andrea's friend status for better understanding, the action_user_id is the user who sent the request:



user_one... integer	user_two... integer	friendship... integer	visibility_... integer	action_us... integer	request_time timestamp without ti...	action_taken_time timestamp without ti...
1	2	2	0	1	2017-03-31 13:06:1...	[null]
1	3	2	0	3	2017-03-31 13:06:1...	[null]
1	8	2	0	1	2017-03-31 13:06:1...	[null]
1	11	1	0	11	2017-03-31 13:06:1...	[null]
1	5	2	0	5	2017-03-31 13:06:1...	[null]

Andrea wants to add someone as a friend.

This would require considerable user interface participation. As first we will select people in the network who aren't friends with Andrea, and she can proceed to select friend/friends which would update the friend relation table accordingly.

```
activity on postgres@PostgreSQL 9.6
1  select first_name,last_name from userinfo where user_id not in (select userinfo.user_id from userinfo, friendrelation where
2                              userinfo.user_id=friendrelation.user_two_id and friendship_status=2 and user_one_id = 1
3  UNION select userinfo.user_id from userinfo,friendrelation where userinfo.user_id=friendrelation.user_one_id and
4                                       friendship_status=2 and user_two_id =1
5  UNION select user_id from userinfo where user_id=1);
```

Data Output | Explain | Messages | History

first_name character (255)	last_name character (255)
Darwin	Johanson
Max	Blunt
Jack	Williams
Karl	Robbin
Lennon	Ray
John	Louis
Fay	Snow
Brian	Watson
Ina	Wood

Inserting a new record as a user has been selected to be a friend.

```
activity on postgres@PostgreSQL 9.6
1  insert into friendrelation values(1,4,1,0,1,current_timestamp,current_timestamp);
2  select * from friendrelation where user_one_id=1 or user_two_id=1;
```

Data Output | Explain | Messages | History

user_one... integer	user_two... integer	friendship... integer	visibility_... integer	action_us... integer	request_time timestamp without tim...	action_taken_time timestamp without tim...
1	2	2	0	1	2017-03-31 13:06:14...	[null]
1	3	2	0	3	2017-03-31 13:06:14...	[null]
1	8	2	0	1	2017-03-31 13:06:14...	[null]
1	11	1	0	11	2017-03-31 13:06:14...	[null]
1	5	2	0	5	2017-03-31 13:06:14...	[null]
1	4	1	0	1	2017-03-31 17:51:00...	2017-03-31 17:51:00...

<u>Andrea wants to accept someone as a friend.</u>

From the above table Andrea has one pending friend request from user_id 11. She has an option to accept this request or decline.

```
10  select * from friendrelation where user_one_id=1 or user_two_id=1;
```

Data Output | Explain | Messages | History

user_one... integer	user_two... integer	friendship... integer	visibility_s... integer	action_us... integer	request_time timestamp without time zone	action_taken_time timestamp without time zone
1	2	2	0	1	2017-03-31 13:06:14.16556	[null]
1	3	2	0	3	2017-03-31 13:06:14.16556	[null]
1	8	2	0	1	2017-03-31 13:06:14.16556	[null]
1	11	1	0	11	2017-03-31 13:06:14.16556	[null]
1	5	2	0	5	2017-03-31 13:06:14.16556	[null]
1	4	1	0	1	2017-03-31 17:51:00.297212	2017-03-31 17:51:00.297212

After updates:

```
update friendrelation set friendship_status=2 where user_one_id=1 or user_two_id=1 and action_user_id=11;
select* from friendrelation where user_one_id=1 or user_two_id=1;
```

Data Output | Explain | Messages | History

user_one... integer	user_two... integer	friendship... integer	visibility_s... integer	action_us... integer	request_time timestamp without time zone	action_taken_time timestamp without time zone
1	2	2	0	1	2017-03-31 13:06:14.16556	[null]
1	3	2	0	3	2017-03-31 13:06:14.16556	[null]
1	8	2	0	1	2017-03-31 13:06:14.16556	[null]
1	11	2	0	11	2017-03-31 13:06:14.16556	[null]
1	5	2	0	5	2017-03-31 13:06:14.16556	[null]
1	4	2	0	1	2017-03-31 17:51:00.297212	2017-03-31 17:51:00.297212

Andrea wants to list all her current friends

activity on postgres@PostgreSQL 9.6

```
select userinfo.first_name,userinfo.last_name from userinfo,friendrelation where
userinfo.user_id=friendrelation.user_two_id and friendship_status=2 and user_one_id = 1
UNION
select userinfo.first_name,userinfo.last_name from userinfo,friendrelation where
userinfo.user_id=user_one_id and friendship_status=2 and user_two_id = 1;
```

Data Output | Explain | Messages | History

first_name character (255)	last_name character (255)
Brenda	Craig
Charles	Daniel
Elle	John
Hilton	Scarlett

Andrea wants to see all their FOFs.

activity on postgres@activespace

```
WITH mutual_user_id as
(select user_two_id from friendrelation where friendship_status=2 and user_one_id in
(select user_two_id from friendrelation where friendship_status=2 and user_one_id=1 UNION
select user_one_id from friendrelation where friendship_status=2 and user_two_id=1) UNION
select user_one_id from friendrelation where friendship_status=2 and user_two_id in(
select user_one_id from friendrelation where friendship_status=2 and user_two_id =1 UNION
select user_two_id from friendrelation where friendship_status=2 and user_one_id =1))

select userinfo.first_name, userinfo.last_name from userinfo,mutual_user_id where mutual_user_id.user_two_id=userinfo.user_id and user_two_id not in(
select user_two_id from friendrelation where friendship_status=2 and user_one_id=1 UNION
select user_two_id from friendrelation where friendship_status=2 and user_two_id=1 UNION
select user_id from userinfo where user_id=1);
```

Data Output | Explain | Messages | History

first_name character (255)	last_name character (255)
Darwin	Johanson
Karl	Robbin
Ina	Wood

Part 3:

Browse/Search Queries:

Andrea want to see all diary entries by her friends and next query show key word 'Grand Canyon'

activity on postgres@activespace

```
1  select * from diaryentry where user_id in
2  (select user_two_id from friendrelation where user_one_id=1
3   UNION
4   select user_one_id from friendrelation where user_two_id=1);
```

Data Output | Explain | Messages | History

entry text	diary_id integer	user_id integer	diarytime timestamp ...	title character va...	loc_id integer	media_id integer
description	1	2	2017-03-31...	title	1	22
Grand Cany...	7	2	2017-03-31...	title	2	22

Friends with diary entries with keyword 'Grand Canyon'

```
9  select diaryentry.entry, diaryentry.user_id from diaryentry where entry like '%Grand Canyon%' and user_id in
10 (select user_two_id from friendrelation where friendship_status=2 and user_one_id=1 UNION
11 select user_one_id from friendrelation where friendship_status=2 and user_two_id=1 UNION
12 select user_id from userinfo where user_id=1);
```

Data Output | Explain | Messages | History

entry text	user_id integer
Grand Canyon! Yes, PLE...	2

Andrea wants to see all diary entries by her FOFs and next with keyword = 'Grand Canyon'

activity on postgres@activespace

```
1  WITH mutual_user_id as
2  (select user_two_id from friendrelation where friendship_status=2 and user_one_id in
3  (select user_two_id from friendrelation where friendship_status=2 and user_one_id=1 UNION
4  select user_one_id from friendrelation where friendship_status=2 and user_two_id=1) UNION
5  select user_one_id from friendrelation where friendship_status=2 and user_two_id in(
6  select user_one_id from friendrelation where friendship_status=2 and user_two_id =1 UNION
7  select user_two_id from friendrelation where friendship_status=2 and user_one_id =1))
8
9  select diaryentry.entry, diaryentry.user_id from diaryentry,mutual_user_id where mutual_user_id.user_two_id=diaryentry.user_id and user_two_id not in(
10 select user_two_id from friendrelation where friendship_status=2 and user_one_id=1 UNION
11 select user_two_id from friendrelation where friendship_status=2 and user_two_id=1 UNION
12 select user_id from userinfo where user_id=1);
```

Data Output | Explain | Messages | History

entry text	user_id integer
Today was super eventful, we barbecue...	9
Hiking plans for Grand Canyon this sum...	4

FOF's with diary entries with Keywords='Grand Canyon'

activity on postgres@activespace

```
1  WITH mutual_user_id as
2  (select user_two_id from friendrelation where friendship_status=2 and user_one_id in
3  (select user_two_id from friendrelation where friendship_status=2 and user_one_id=1 UNION
4  select user_one_id from friendrelation where friendship_status=2 and user_two_id=1) UNION
5  select user_one_id from friendrelation where friendship_status=2 and user_two_id in(
6  select user_one_id from friendrelation where friendship_status=2 and user_two_id =1 UNION
7  select user_two_id from friendrelation where friendship_status=2 and user_one_id =1))
8
9  select diaryentry.entry, diaryentry.user_id from diaryentry,mutual_user_id where diaryentry.user_id=mutual_user_id.user_two_id
10 and entry like '%Grand Canyon%' and user_two_id not in
11 (select user_two_id from friendrelation where friendship_status=2 and user_one_id=1 UNION
12 select user_one_id from friendrelation where friendship_status=2 and user_two_id=1 UNION
13 select user_id from userinfo where user_id=1);
14
```

Data Output | Explain | Messages | History

entry text	user_id integer
Hiking plans for Grand Canyon ...	4

Andrea wants to see diary entries by anyone, that contain certain keywords such as "Grand Canyon".

```
select diaryentry.entry, diaryentry.user_id from diaryentry where entry like '%Grand Canyon%';
```

Data Output | Explain | Messages | History

entry text	user_id integer
Grand Canyon! Yes, PLEASE!	2
Hiking plans for Grand Canyon this ...	4
Looking for hiking company to the ...	13

Andrea wants to list of all diary entries by her friends during the last week.

```
select entry, user_id, diarytime from diaryentry where user_id in (
select user_two_id from friendrelation where friendship_status=2 and user_one_id=1 UNION
select user_one_id from friendrelation where friendship_status=2 and user_two_id=1) and date_part('day',current_timestamp-diarytime)<=7;
```

Data Output | Explain | Messages | History

entry text	user_id integer	diarytime timestamp without time zone
Grand Canyon! Yes, PLEASE!	2	2017-03-29 09:19:22

Andrea may want to see all locations for a conference.

```
select events.description,location.address,location.city from location,events where events.loc_id=location.loc_id and city like '%new york%';
```

Data Output | Explain | Messages | History

description text	address character varying (60)	city character varying (20)
Meditation in Central Park	central park	new york city
Dog Fashion show at Central Park 1/4	central park	new york city

Displaying contents of all the tables

1. Userinfo table

```
select * from userinfo;
```

Data Output | Explain | Messages | History

user_id integer	creation_timestamp timestamp without time zone	email_id character varying (10...	username character varying (40)	user_password character varying (3...	first_name character (25...	last_name character (255)	phone integer	gender character (1...	date_of_birth date	picture_medium bytea
2	2017-03-31 12:14:27.59014	brenda@mail.com	BCraig	Password!	Brenda	Craig	51764	female	1997-03-29	[null]
3	2017-03-31 12:20:59.541...	daniel@mail.com	danielcharles007	Password!	Charles	Daniel	5168	Male	1990-01-05	NULL
4	2017-03-31 12:20:59.541...	darwin@mail.com	darwin_johanson	Password!	Darwin	Johanson	43200	Male	1982-11-02	NULL
5	2017-03-31 12:20:59.541...	elle@mail.com	ellejohn___01	Password!	Elle	John	63199	Female	1991-09-01	NULL
1	2017-03-31 12:14:27.59014	andrea@mail.com	AndreaJovi123	Password!	Andrea	Jovi	516814	female	1987-09-12	[null]
13	2017-03-31 12:47:13.380...	maxx@mail.com	Maxx	Password!	Max	Blunt	9893	Female	1993-08-01	[null]
10	2017-03-31 12:47:13.380...	jack@mail.com	JAckWillIAMS123	Password!	Jack	Williams	4165	Male	1982-02-01	[null]
11	2017-03-31 12:47:13.380...	karl@mail.com	KarlRobbinHood	Password!	Karl	Robbin	2168	Female	1987-09-12	[null]
12	2017-03-31 12:47:13.380...	lennon@mail.com	LennonRay12345678	Password!	Lennon	Ray	92421	Male	1990-10-21	[null]
14	2017-03-31 16:31:41.724...	john@mail.com	JohnnyUser	Password123	John	Louis	516815	Male	1991-09-12	[null]
6	2017-03-31 12:40:16.326...	fay@mail.com	FaySn0w001	Password!	Fay	Snow	9221	Female	1984-04-20	[null]
7	2017-03-31 12:46:03.125...	watson@mail.com	Watsonsden777	Password!	Brian	Watson	4160	Male	1991-01-12	[null]
8	2017-03-31 12:46:03.125...	hilton@mail.com	hiltonscarlett	Password!	Hilton	Scarlett	93611	Male	1995-11-12	[null]
9	2017-03-31 12:46:03.125...	ina@mail.com	InaWoody09	Password!	Ina	Wood	82900	Female	1989-01-09	[null]

2. Location table

```
9  select * from location;
```

Data Output | Explain | Messages | History

loc_id integer	lat real	lng real	address character varying (60)	city character varying (20)	state character varying (20)	country character varying (20)
1	[null]	[null]	central park	new york city	ny	usa
2	[null]	[null]	breakneck ridge	cold springs	ny	usa
3	[null]	[null]	[null]	montauk	ny	usa
4	[null]	[null]	[null]	key west	FL	usa
5	[null]	[null]	[null]	grand canyon	AZ	usa

3. Multimedia table

```
9  select * from multimedia;
```

Data Output | Explain | Messages | History

media_id integer	post_time timestamp without time zone	content bytea	description text	user_id integer
1	2017-03-31 12:58:37.324923	[null]	text1	1
2	2017-03-31 12:58:37.324923	[null]	text1	1
3	2017-03-31 12:58:37.324923	[null]	text1	7
4	2017-03-31 12:58:37.324923	[null]	text1	9
5	2017-03-31 12:58:37.324923	[null]	text1	2
6	2017-03-31 12:58:37.324923	[null]	text1	1
7	2017-03-31 12:58:37.324923	[null]	text1	4
8	2017-03-31 12:58:37.324923	[null]	text1	1
9	2017-03-31 12:58:37.324923	[null]	text1	3
10	2017-03-31 12:58:37.324923	[null]	text1	1
11	2017-03-31 12:58:37.324923	[null]	text1	6
12	2017-03-31 12:58:37.324923	[null]	text1	1
13	2017-03-31 12:58:37.324923	[null]	text1	8
14	2017-03-31 12:58:37.324923	[null]	text1	9
15	2017-03-31 12:58:37.324923	[null]	text1	1
16	2017-03-31 12:58:37.324923	[null]	text1	1
17	2017-03-31 12:58:37.324923	[null]	text1	13
18	2017-03-31 12:58:37.324923	[null]	text1	12

4. Diaryentry table

```
9  select * from diaryentry;
```

Data Output | Explain | Messages | History

entry text	diary_id integer	user_id integer	diarytime timestamp without time zone	title character varying (100)	loc_id integer	media_id integer
description	2	1	2017-03-31 12:59:14.164701	title	2	15
description	9	6	2017-03-31 12:59:14.164701	title	4	20
description	10	6	2017-03-31 12:59:14.164701	title	5	11
Today was super eventful, we barbecue...	6	9	2017-03-31 12:59:14.164701	title	1	14
Walking my dog out in the cold. Does a...	4	1	2017-03-31 12:59:14.164701	title	4	8
Barbecuing at my rooftop and it begins ...	3	1	2017-03-31 12:59:14.164701	title	3	16
The Sun is shining all so bright. Who w...	13	1	2017-03-31 17:29:30.033547	Such a beautiful day!	5	[null]
Hiking plans for Grand Canyon this sum...	5	4	2017-03-31 12:59:14.164701	title	5	10
Looking for hiking company to the Gran...	8	13	2017-03-31 12:59:14.164701	title	3	17
description	1	2	2017-02-20 00:00:00	title	1	22
Grand Canyon! Yes, PLEASE!	7	2	2017-03-29 09:19:22	title	2	22

5. Diary_comments

```
9  select * from diary_comments;
```

Data Output | Explain | Messages | History

comment... integer	comment_entry text	diary_id integer	comment_time timestamp without time zone	user_id integer
1	comment1	1	2017-03-31 13:02:17.926835	1
2	comment2	2	2017-03-31 13:02:17.926835	4
3	comment3	3	2017-03-31 13:02:17.926835	3
4	comment4	4	2017-03-31 13:02:17.926835	6
5	comment5	4	2017-03-31 13:02:17.926835	7
6	comment6	4	2017-03-31 13:02:17.926835	8
7	comment7	4	2017-03-31 13:02:17.926835	9
8	comment8	7	2017-03-31 13:02:17.926835	11
9	comment9	9	2017-03-31 13:02:17.926835	12
10	comment10	10	2017-03-31 13:02:17.926835	12
12	comment12	3	2017-03-31 13:02:17.926835	6

6. Diary_likes

```
9  select * from diary_likes;
```

Data Output | Explain | Messages | History

diary_id integer	user_id integer	like_time timestamp without time zone
2	1	2017-03-31 13:05:29.27906
3	2	2017-03-31 13:05:29.27906
4	1	2017-03-31 13:05:29.27906
5	2	2017-03-31 13:05:29.27906
6	8	2017-03-31 13:05:29.27906
7	4	2017-03-31 13:05:29.27906
8	7	2017-03-31 13:05:29.27906
6	4	2017-03-31 13:05:29.27906
1	3	2017-03-31 13:05:29.27906
1	1	2017-03-31 13:05:29.27906
3	3	2017-03-31 13:05:29.27906
5	11	2017-03-31 13:05:29.27906
3	5	2017-03-31 13:05:29.27906
6	6	2017-03-31 13:05:29.27906
6	3	2017-03-31 13:05:29.27906
9	7	2017-03-31 13:05:29.27906
9	9	2017-03-31 13:05:29.27906
10	10	2017-03-31 13:05:29.27906

7. Events table

```
9  select * from events;
```

Data Output | Explain | Messages | History

event_id integer	description text	event_time timestamp without time zone	loc_id integer	media_id integer
2	Meditation in Central Park	2017-03-30 07:00:00	1	[null]
1	Hiking at Breakneck Ridge	2017-04-09 05:00:00	2	[null]
3	Dog Fashion show at Central Park 1/4	2017-03-20 08:23:53	1	[null]

8. Event_members table

```
9  select * from event_members;
10
```

Data Output | Explain | Messages | History

event_id integer	user_id integer
1	1
1	2
2	3
3	4
3	9
3	10
2	6
1	3
1	7
2	5
3	2
1	3
2	4
2	1
3	1
3	8
1	5
2	1
1	4

9. Posts table

```
9  select * from posts;
```

Data Output | Explain | Messages | History

poster_id integer	postee_id integer	content text	post_time timestamp without time zone	post_id integer
1	2	content	2017-03-31 22:20:15.143144	1
1	5	content	2017-03-31 22:20:15.143144	2
1	3	content	2017-03-31 22:20:15.143144	3
1	6	content	2017-03-31 22:20:15.143144	4
1	3	content	2017-03-31 22:20:15.143144	5
1	7	content	2017-03-31 22:20:15.143144	6
1	8	content	2017-03-31 22:20:15.143144	7
1	9	content	2017-03-31 22:20:15.143144	8
1	10	content	2017-03-31 22:20:15.143144	9
2	12	content	2017-03-31 22:20:15.143144	10

10. Friendrelation table

```
9  select * from friendrelation;
```

Data Output | Explain | Messages | History

user_one... integer	user_two... integer	friendship... integer	visibility_... integer	action_us... integer	request_time timestamp without time zone	action_taken_time timestamp without time zone
3	2	2	0	3	2017-03-31 13:06:14.16556	[null]
8	2	2	0	8	2017-03-31 13:06:14.16556	[null]
4	8	2	0	4	2017-03-31 13:06:14.16556	[null]
4	9	2	0	9	2017-03-31 13:06:14.16556	[null]
3	9	2	0	3	2017-03-31 13:06:14.16556	[null]
7	3	1	0	1	2017-03-31 13:06:14.16556	[null]
11	2	2	0	3	2017-03-31 13:06:14.16556	[null]
6	4	2	0	6	2017-03-31 13:06:14.16556	[null]
1	2	2	0	1	2017-03-31 13:06:14.16556	[null]
1	3	2	0	3	2017-03-31 13:06:14.16556	[null]
1	8	2	0	1	2017-03-31 13:06:14.16556	[null]
1	11	2	0	11	2017-03-31 13:06:14.16556	[null]
1	5	2	0	5	2017-03-31 13:06:14.16556	[null]
1	4	2	0	1	2017-03-31 17:51:00.297212	2017-03-31 17:51:00.297212

1.4. Stored Procedures, Functions and Triggers

The following stored procedures and triggers are created to make our recurring application operations convenient, fast and efficient.

Accepts two user IDs and changes their status to friends if the receiver accepts the friend request.

```
-- For Accepting Friendship
CREATE OR REPLACE FUNCTION accept_friendship(user1 integer, user2 integer)
    RETURNS void AS $$
      BEGIN
     UPDATE  friendrelation SET friendship_status='2' WHERE
action_user_id=user2 and user_one_id in (user1,user2) and user_two_id in
(user1,user2);
    END;
    $$ LANGUAGE plpgsql;
```

Accepts two user IDs and sends a friend request from one user to another.

```
-- For Adding Friendship
CREATE OR REPLACE FUNCTION add_friendship(user1 integer, user2 integer)
    RETURNS void AS $$
      BEGIN
      INSERT INTO friendrelation VALUES (user1,
user2,1,0,user1,current_timestamp,current_timestamp);
    END;
    $$ LANGUAGE plpgsql;
```

Accepts two User IDs and changes the status to rejected if the receiver declines the friend request.

```
-- For Declining Friendship
CREATE OR REPLACE FUNCTION decline_friendship(user1 integer, user2 integer)
    RETURNS void AS $$
      BEGIN
      DELETE from friendrelation  where user_one_id in (user1,user2) and
user_two_id in (user1,user2);
    END;
    $$ LANGUAGE plpgsql;
```

Accepts the image patch, user ID, and a description and adds an entry in the multimedia table for an upload.

```
-- For uploading photos to profile
CREATE OR REPLACE FUNCTION upload_photos(character varchar(100),image
varchar(50),userid integer)
    RETURNS void AS $$
      BEGIN
      INSERT INTO multimedia(post_time,content,description,user_id) VALUES
(current_timestamp,image,text,userid);
    END;
    $$ LANGUAGE plpgsql;
```

Accepts the title, content, media, location and user ID and adds an entry to the diary entry table.

```
-- For uploading diary entries to profile
CREATE OR REPLACE FUNCTION upload_diary(diary_title varchar(100),diary_desc
varchar(50),diary_media varchar(100),location integer, userid integer)
    RETURNS void AS $$
      BEGIN
      INSERT INTO diaryentry(entry,user_id,diarytime,title,loc_id,multimedia)
VALUES
(diary_desc,userid,current_timestamp,diary_title,location,diary_media);
    END;
    $$ LANGUAGE plpgsql;
```

Accepts the poster ID, postee ID and content and adds an entry to the posts table.

```
-- For uploading posts to profile
CREATE OR REPLACE FUNCTION upload_posts(user1 INTEGER,user2 INTEGER,
description varchar(150))
    RETURNS void AS $$
      BEGIN
```

```
      INSERT INTO posts(poster_id, postee_id, content,post_time) VALUES
(user1,user2,description,current_timestamp);
    END;
    $$ LANGUAGE plpgsql;
```

Accepts location details like address, city, state and country and adds a location entry provided by a user.

```
-- Adding location
CREATE OR REPLACE FUNCTION add_location(address1 VARCHAR(100), city1
VARCHAR(100),state1 VARCHAR(100), country1 VARCHAR(100))
    RETURNS void AS $$
      BEGIN
      INSERT INTO location(lat,lng,address,city,state,country) VALUES
(NULL,NULL,address1,city1,state1,country1);
    END;
    $$ LANGUAGE plpgsql;
```

Adds a like from a user defined by the user ID to a specific diary entry defined by its diary ID

```
-- Adding likers
CREATE OR REPLACE FUNCTION add_diarylikers(user1 INTEGER, diary1 INTEGER)
    RETURNS void AS $$
      BEGIN
      INSERT INTO diary_likes(user_id, diary_id,like_time) VALUES
(user1,diary1,current_timestamp);
    END;
    $$ LANGUAGE plpgsql;
```

Accepts content, user ID and diary ID to add an entry to diary_comments table

```
-- Adding commenters
CREATE OR REPLACE FUNCTION add_diarycommenters(user1 INTEGER, diary1 INTEGER,
text varchar(100))
    RETURNS void AS $$
       BEGIN
      INSERT INTO diary_comments(user_id,diary_id,comment_entry,comment_time)
VALUES(userid,diaryid,text,current_timestamp);
    END;
    $$ LANGUAGE plpgsql;
```

Accepts location details and adds a location to a diary entry.

```
-- Adding commenter
CREATE OR REPLACE FUNCTION add_diarylocation(address1 VARCHAR(100), city1
VARCHAR(100),state1 VARCHAR(100), country1 VARCHAR(100))
    RETURNS void AS $$
      BEGIN
      INSERT INTO location(lat,lng,address,city,state,country) VALUES
(NULL,NULL,address1,city1,state1,country1);
    END;
    $$ LANGUAGE plpgsql;
```

Accepts an image and adds it to a user's profile

```
-- UPLOAD DISPLAY PICTURE
    CREATE OR REPLACE FUNCTION upload_dp(image varchar(100),userid integer)
    RETURNS void AS $$
      BEGIN
      UPDATE userinfo SET picture_medium=image WHERE user_id=userid;
    END;
    $$ LANGUAGE plpgsql;
```

A combination of a trigger and procedure to check for updates to a friend request on the friendrelation table to detect changes like accepts, declines etc to a friend request.

```
-- TRIGGER AND STORED FUNCTION FOR FRIENDSHIP STATUS
CREATE OR REPLACE FUNCTION update_modified_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.action_taken_time = now();
    RETURN NEW;
END;
$$ language plpgsql;

CREATE TRIGGER update_customer_modtime BEFORE UPDATE ON friendrelation FOR
EACH ROW EXECUTE PROCEDURE update_modified_column();
```

2. Front-end design

Not all people are skilled in the art of managing and working with databases. To help make our database understandable by a wider audience, a front-end is developed to interact with the underlying database. Hence, the front-end of the application is an important aspect of an application system. It is the reason an application can be made user-friendly and accessible to all. It is a platform where developers can get creative to enhance user-friendliness, attractiveness and ease of access to the application. The following sections will describe the front-end for ActiveSpace.

2.1. Website Structure

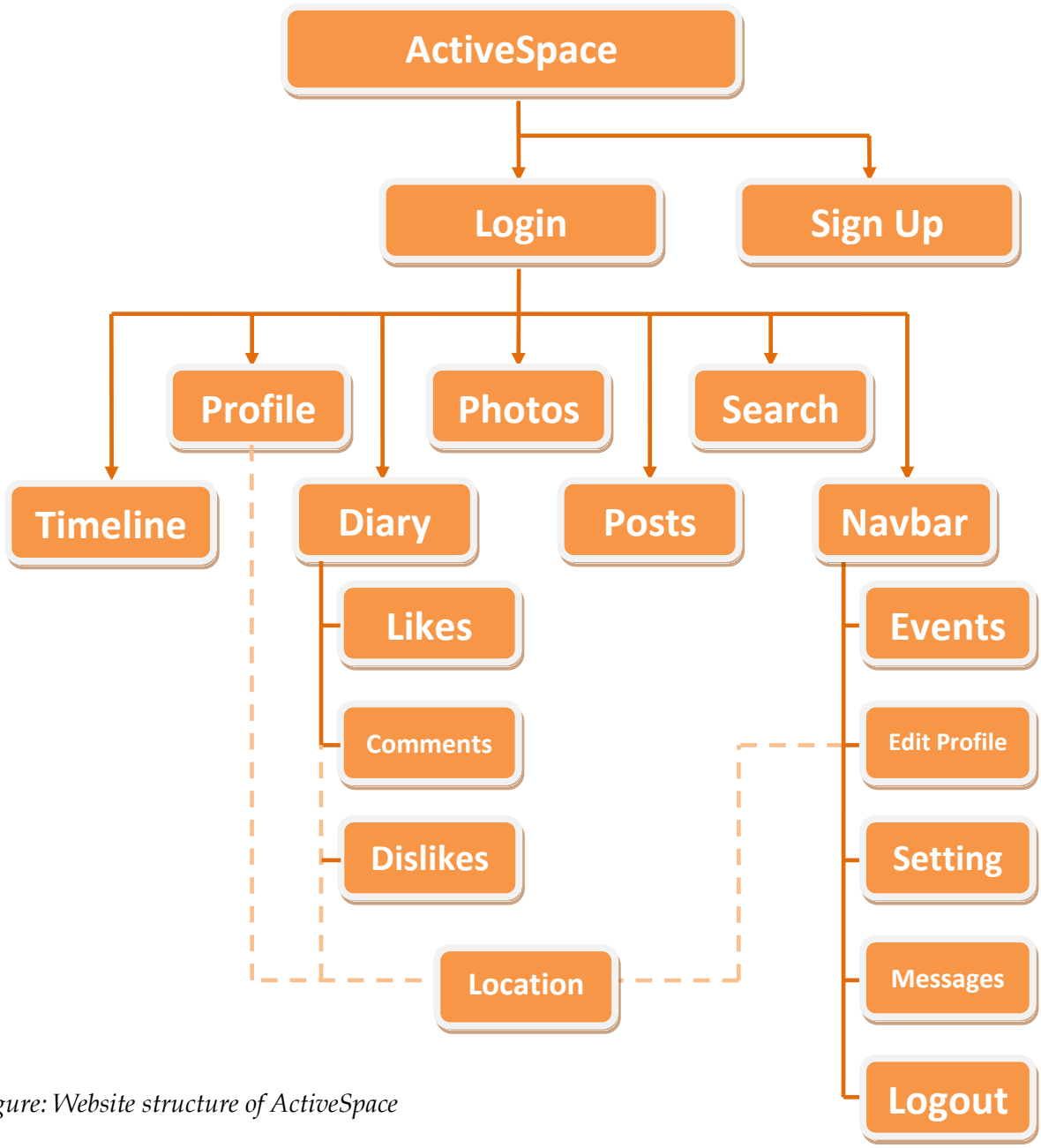


Figure: Website structure of ActiveSpace

2.2. Application Features

ActiveSpace contains numerous features to provide its users with a seamless experience.

Login/Sign Up: The user is first greeted in the login page. To create an account, the user can click on the Sign Up button. In this page, the user can provide his details to be shared on the social network and create credentials to login to the website. The sign up page also checks if an email ID is already registered on the website. There is also a feature to edit the profile information if the user wishes to do so. The timestamp at which users create their account, edits his profile and last logs in are recorded and updated if required. The profile lets other users learn the user's basic information he wishes to share on the network.

Timeline: After being authenticated into the system, the user enters the landing page. This consists of all updates from his network. It contains a mixture of diary entries, posts and photos which other friends have added to ActiveSpace. To navigate to the web pages containing these updates, a row of buttons are present just below the Navbar. Another important feature incorporated here is Search. The user can perform search functions to find specific users or entries throughout ActiveSpace.

Figure: Buttons to navigate the website

Diary, Posts & Photos: The user can upload diary entries, photos and post on users profiles to interact on the social network. User gets to decide if he wants to broadcast messages or pick a specific friend to share a post with. A diary entry can accept a title, description, an image and a location from the user. Diary entries of other users can be liked, disliked and users can provide comments to it too. Images can be added individually too along with a description. Posts can be posted as a broadcast message to everyone in the network or can also be directed to a specific friend. These can be accessed by the three buttons 'Diary It', 'Snap It' and 'Post It'.

Figure: Buttons to add uploads

Locations: Users of ActiveSpace can choose locations that exist in the network to tag their profiles and diary entries. User has the liberty to create a new location while creating his profile or creating a diary entry, which will be stored in the network's database. Once stored in the network, these locations can be used by all users in ActiveSpace.

Search box dedicated for people: Users can search for other users on the network. If no keyword is supplied, the user can search all his friends, all his FOF or everyone and browse to each of their profiles and look for their diary/photos/posts subject to their network visibility settings. The keyword entered is case-sensitive.

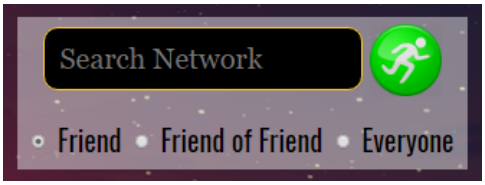


Figure: Search box for people

Search box dedicated for entries: Users can search for Posts, Photos, Events, and Diary entries by providing an input consisting of a certain keyword in each of them. If no keyword is input, all of the posts, photos and diary entries will show up for their friend network. The search also allows Location filtering. A location can be entered in this search bar and all entries existing in that location will show up in the results.

Figure: Search box for entries

User Profile: The user profile contains all details about the user, a profile photo and information that the user wants to share across his network. When viewing another user's profile, one can view their other entries subject to their visibility status.

Logout button: Ends the user sessions and logs the user out of ActiveSpace. This prevents other users from gaining unauthorized access to other user's data.

Add, accept and decline users: This feature provides the user to create, modify and expand his network. Apart from notification inbox, browsing to random users or users in you circle will show you a notification if you are connected/friends/if you have sent request previously/if you have a pending request from their end/ or if you aren't currently connected. You have an option to ADD them/ ACCEPT them or DECLINE their invitation.

2.3. User Interface

"Good design is good business"

— Thomas J Watson Jr., President of IBM

A good user interface provides a seamless experience for users to navigate contents of the application. HTML provides a great set of features and functionality in achieving good design. This section provides a guide to navigating ActiveSpace.

The following screenshots are taken from the working application. They effectively potray the look and feel of ActiveSpace. A brief description is provided to demonstrate the working of its numerous features.

Login page

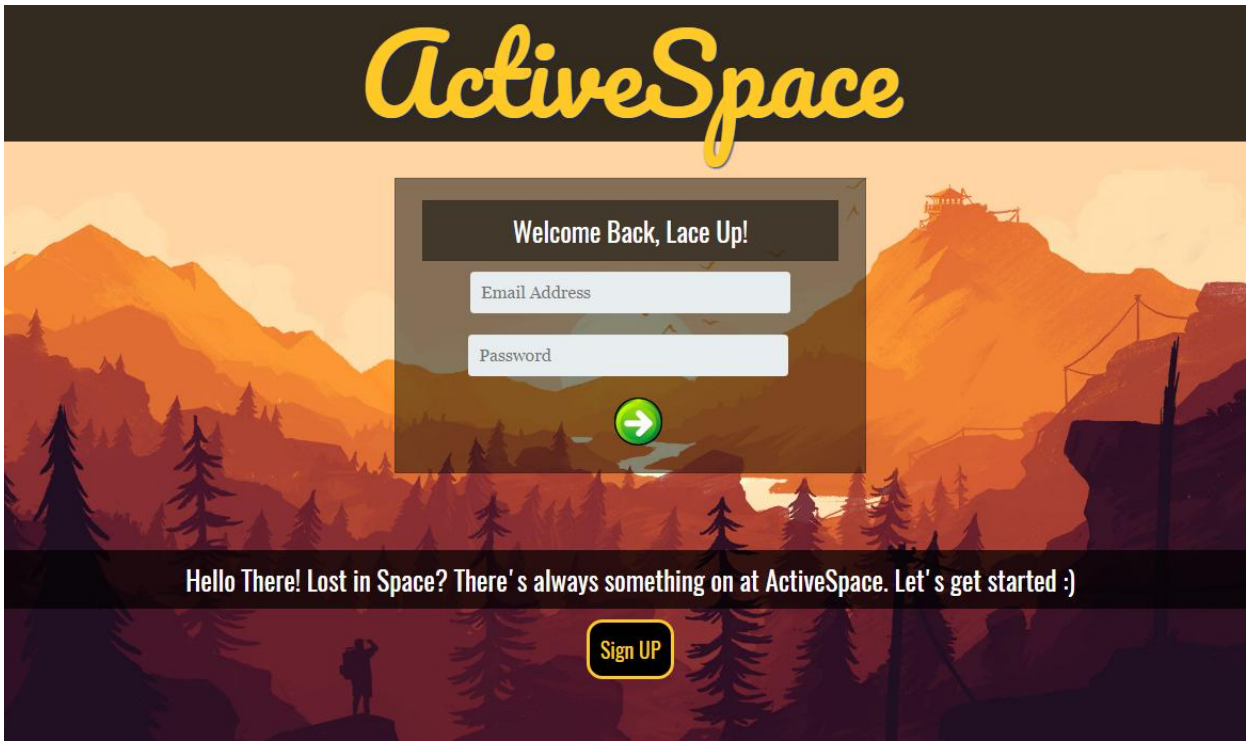


Figure: Login page of ActiveSpace

Figure: Creating new user in ActiveSpace

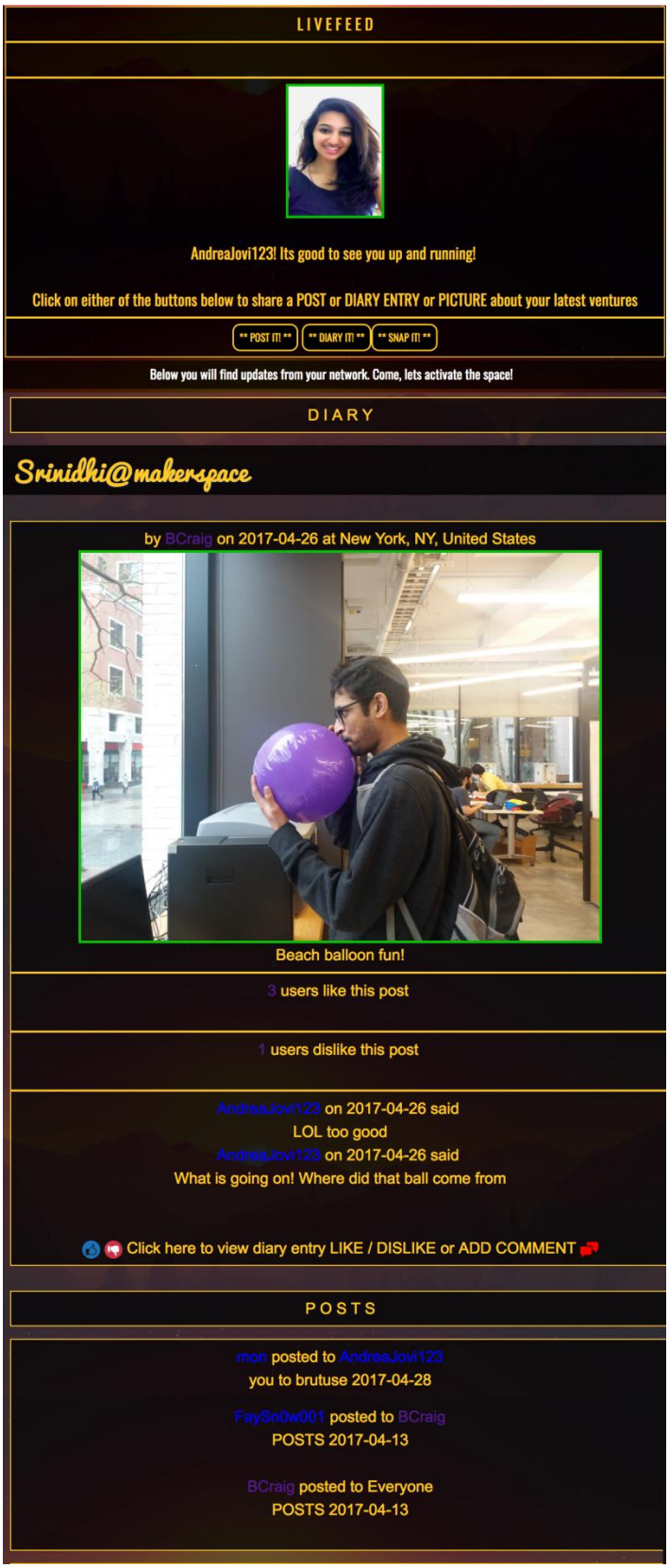


Figure: Homepage of ActiveSpace

Figure: View Profile

Edit Profile

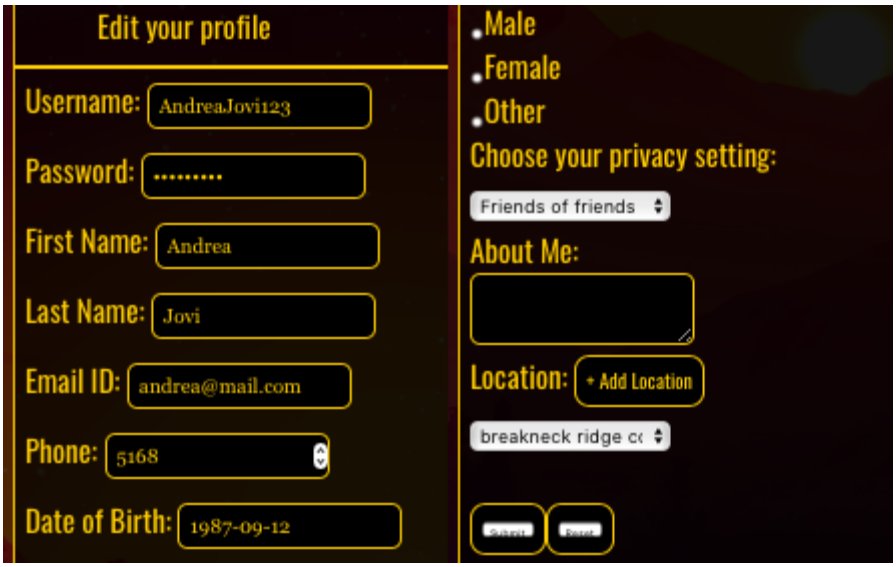


Figure: Edit profile gets all the previous info for user convenience.

Friend Requests: The user can view who has sent a friend request and when they sent it.

Figure: On clicking the requests button, the user can accept or decline invites received.

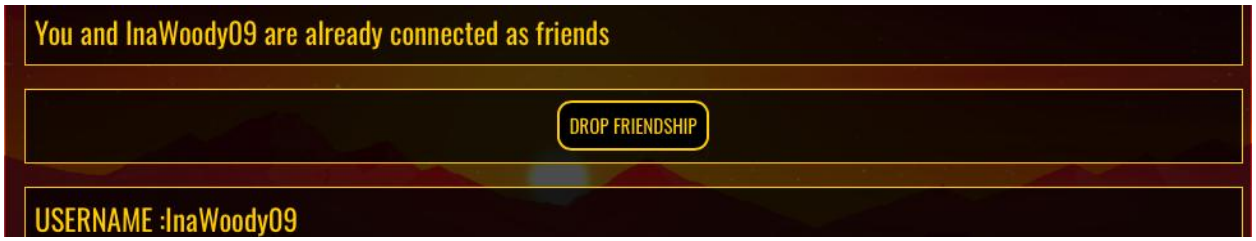


Figure: On clicking the Drop Friendship button, the user can delete existing connections.

Navigating to different users on ActiveSpace can happen through clicking their username hyperlinks on any of the posts, diary entries, messages, events or their profile link during requests to your network of friends etc.

Figure: A profile with an incoming request will be displayed in the above manner.

Search

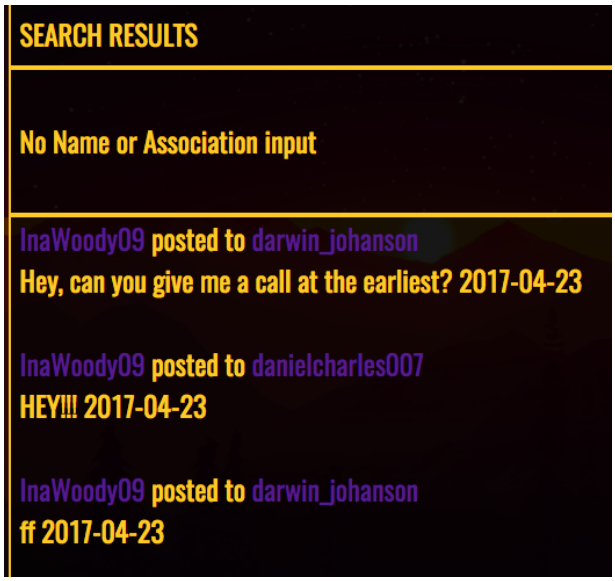


Figure: 'posts' search without any keyword input returns all posts in the user's network

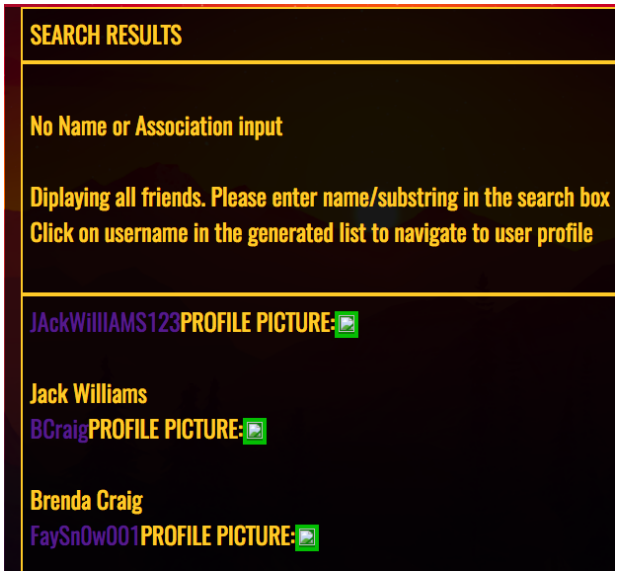


Figure: 'friend' search without any keyword input returns all friends in the user's network

Navigating to different users on ActiveSpace can also happen through searching them on the network with the "Search Network' functionality. Users can be found through searching for Friends, Freiends of Friends, or Everyone.

But the users can only be viewed subject to their visibility constraints. This is a feature which would be explained in the later section.

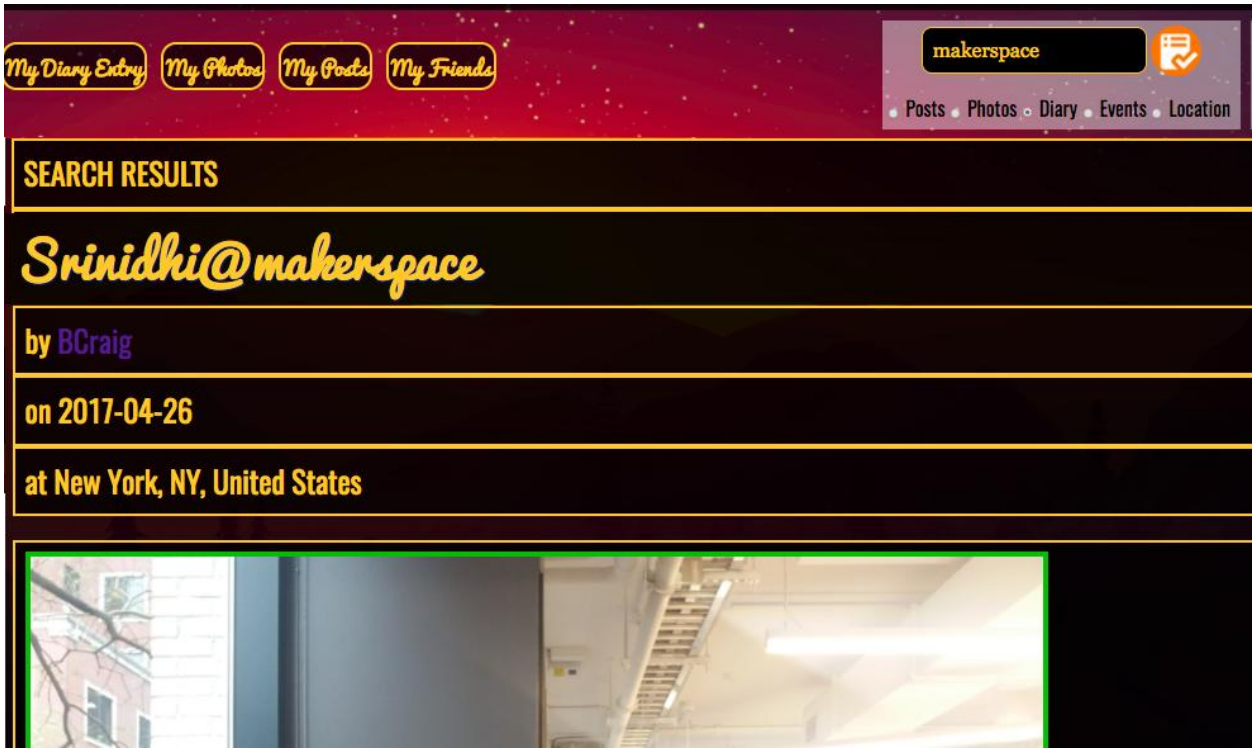


Figure: Keyword 'makerspace' in diary search returns a specific diary entry present in the network

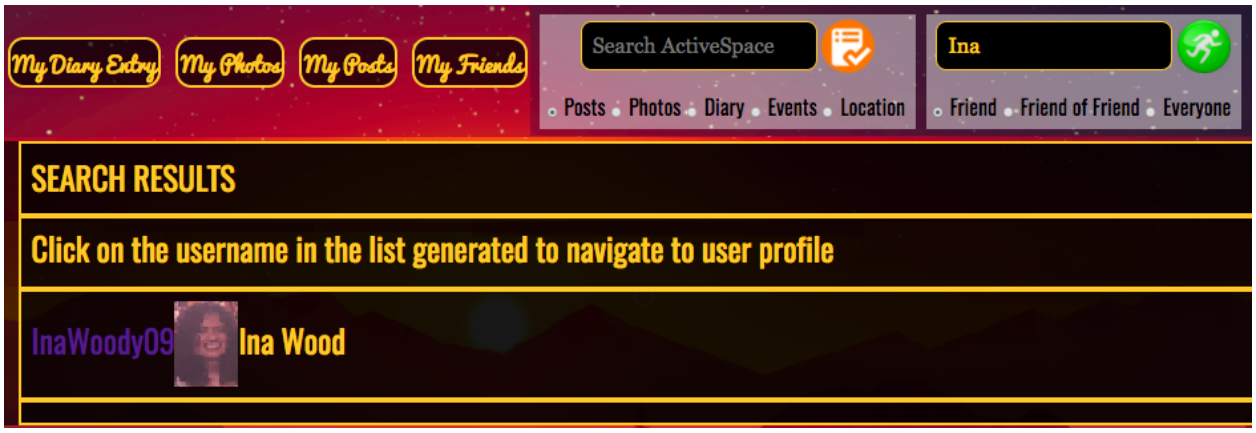


Figure: Keyword 'Ina' in friends search returns users with the keyword present in the name.

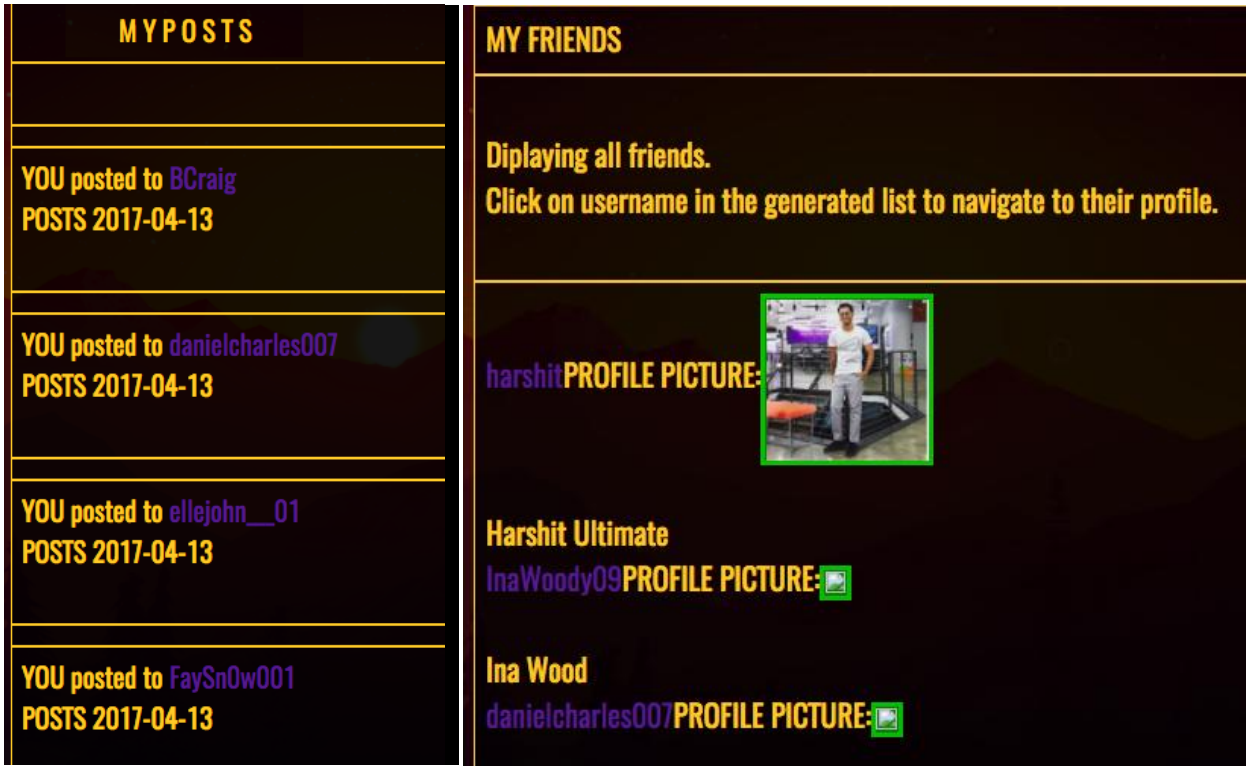


Figure: Page showing My Posts and My Friends

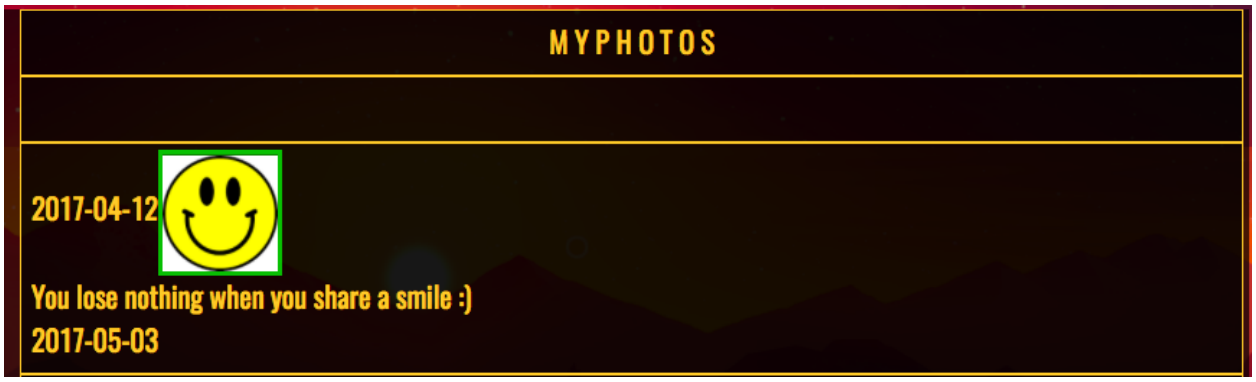


Figure: My Photos page in ActiveSpace

Figure: My Diary Entry page in ActiveSpace

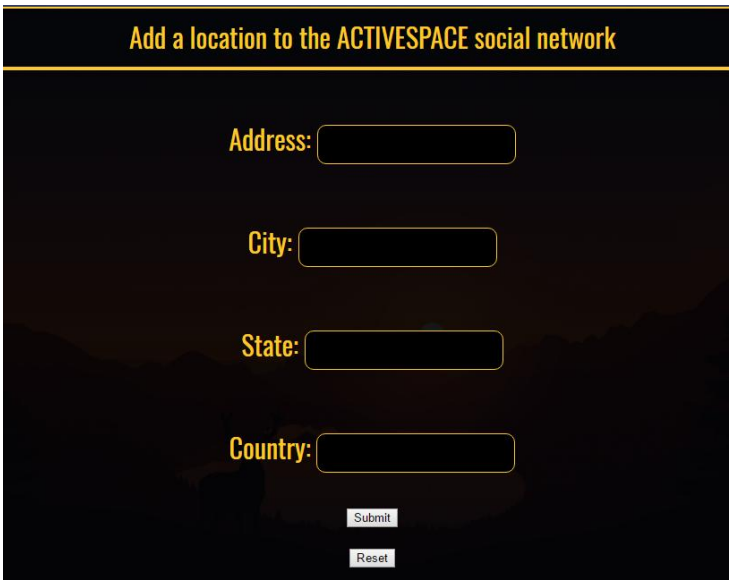


Figure: Add a location to the ActiveSpace network

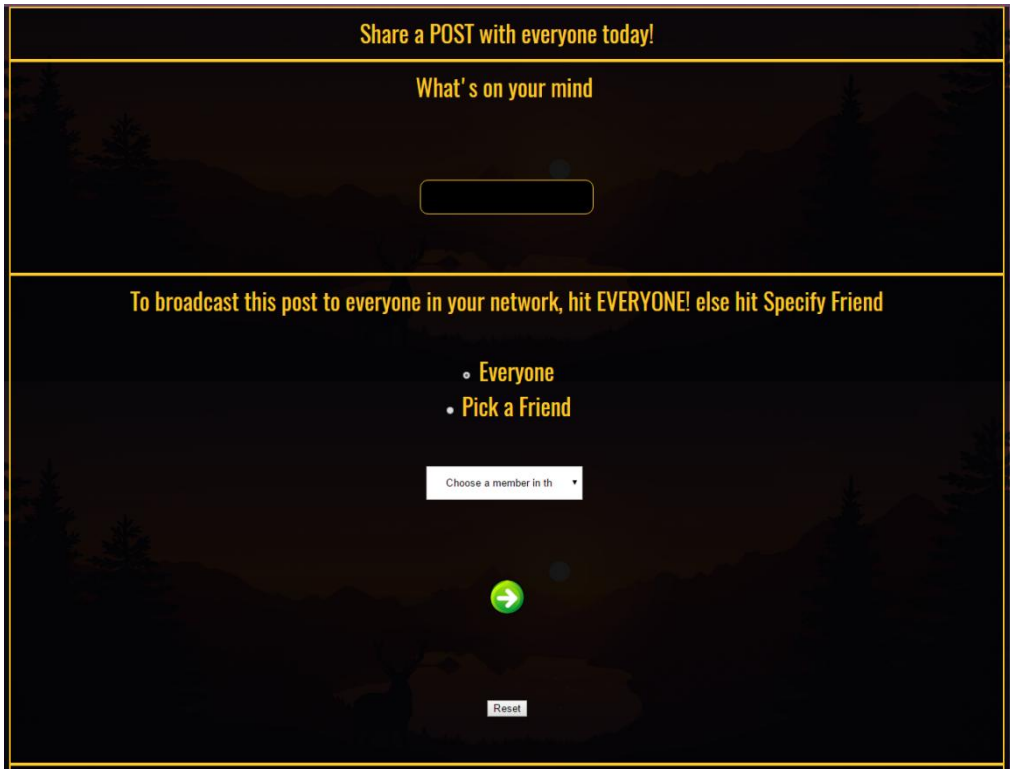


Figure: Add a post

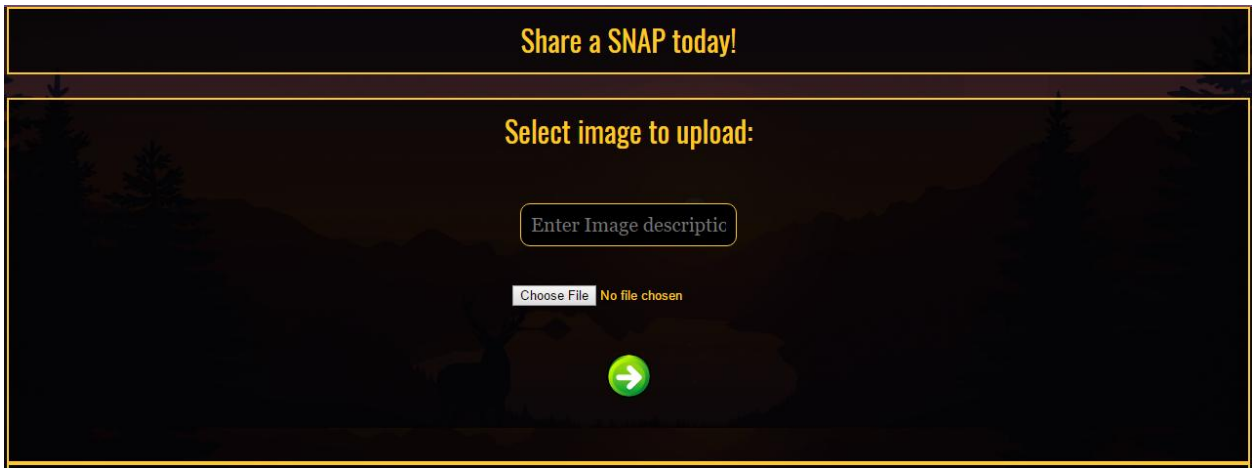


Figure: Add a photo

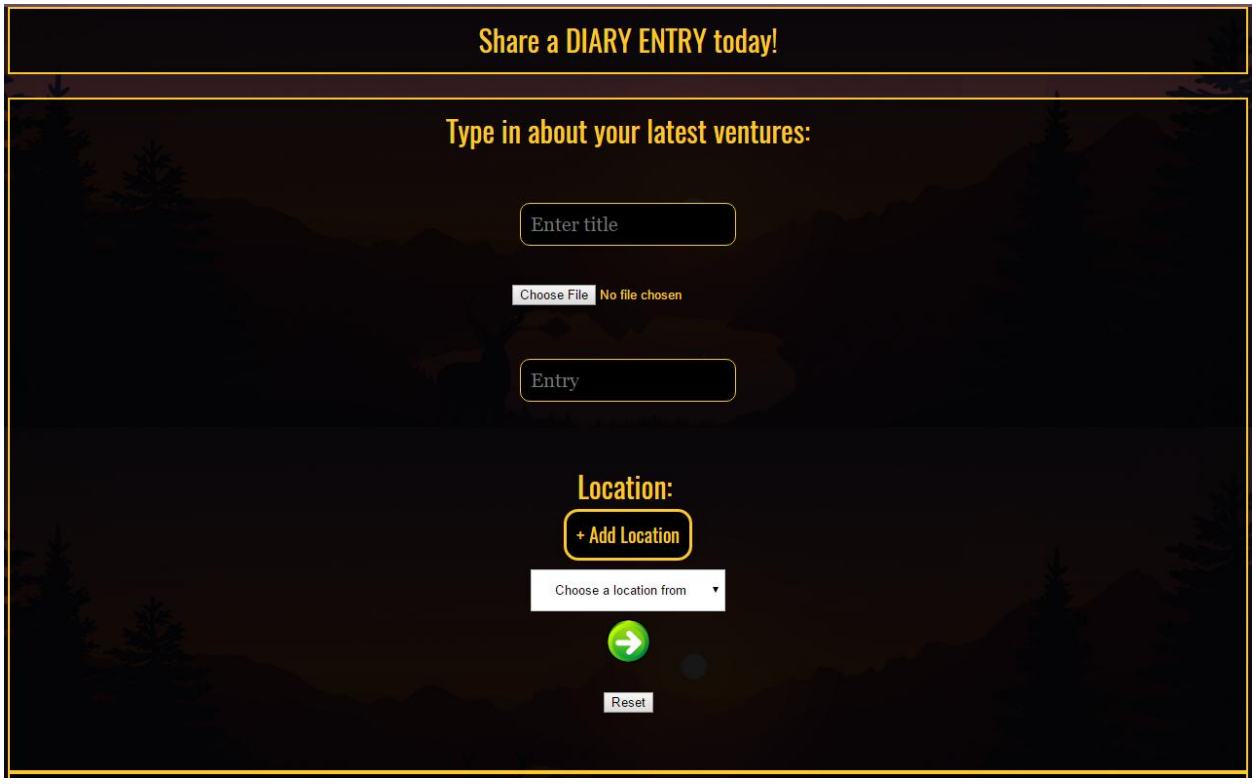


Figure: Add a diary entry

2.4. Extra Features

Notifications: The user can find an overview of friend requests and posts received in ActiveSpace.

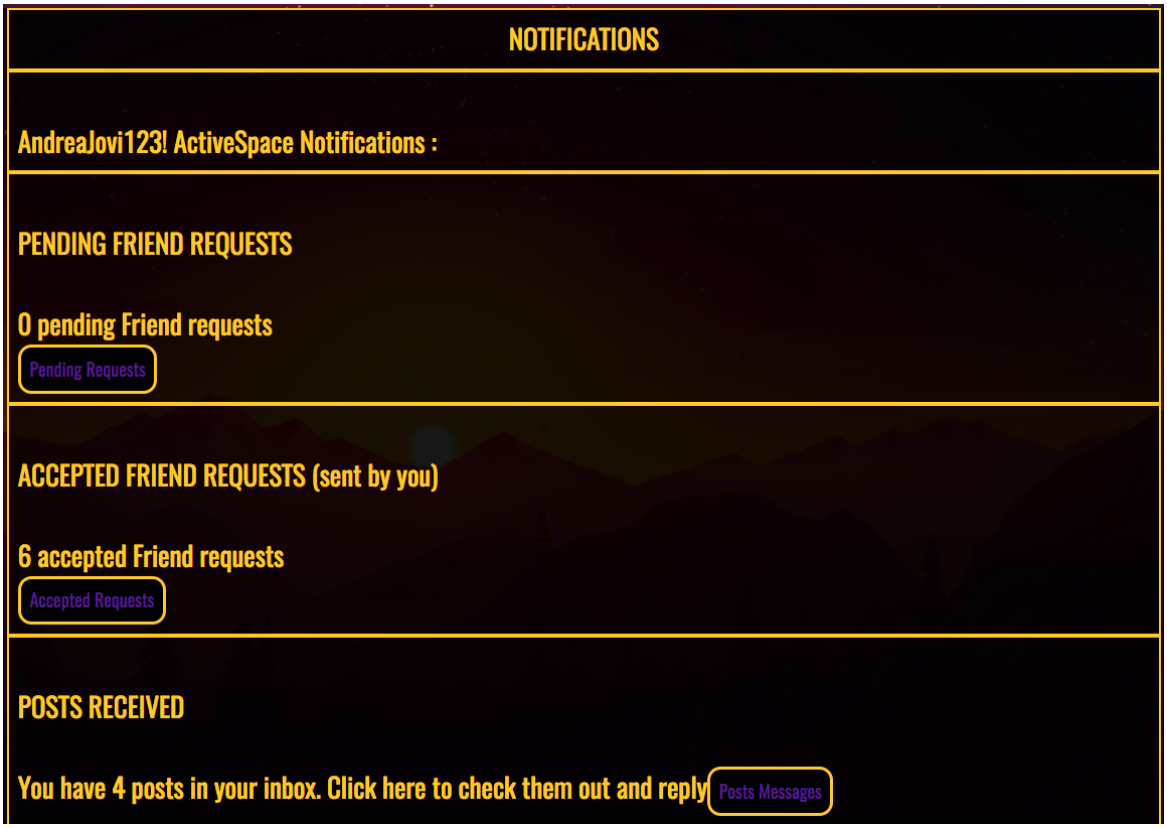


Figure: Notifications in ActiveSpace

Messages: On clicking the Posts Messages button, the user has a message inbox, which when clicked directs him to people who have sent posts directed to the user. The user can reply to these posts by clicking on the reply button.

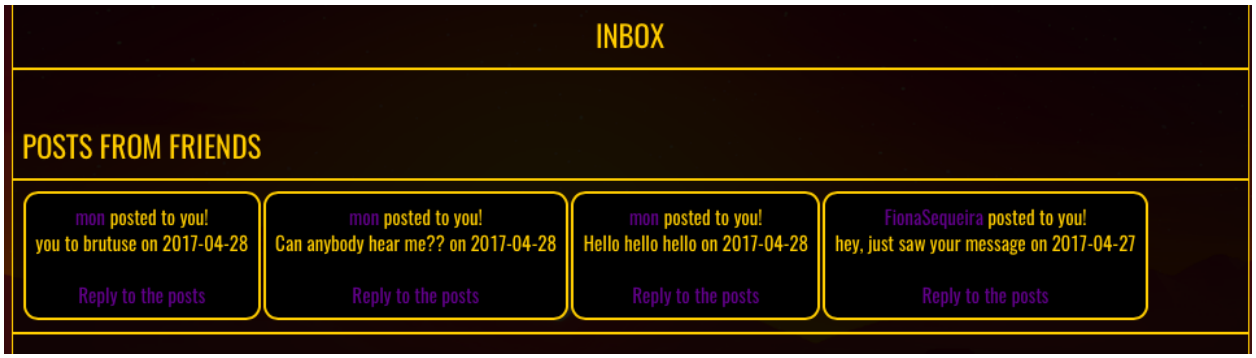


Figure: Messages in ActiveSpace

Navbar: The navigation bar, or navbar in short, is a common element accessible from anywhere in the web application. It is an extremely user-friendly element which provides ease of accessing the main elements of the application namely, timeline, profile, photos, diary entry and posts belonging to the user, in descending order of the date. It also hosts various buttons to change profile details, visibility settings, view events and messages. The mandatory logout button is placed here to increase its accessibility.

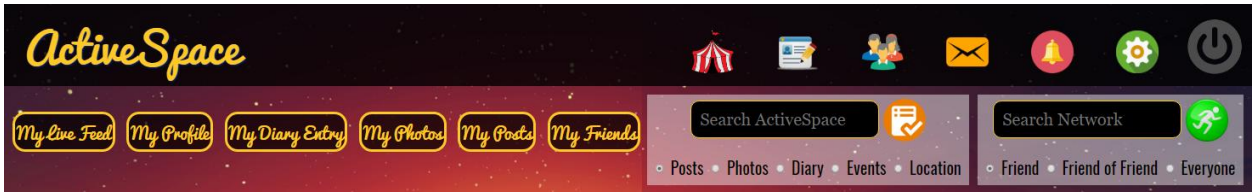


Figure: Navigation bar in ActiveSpace

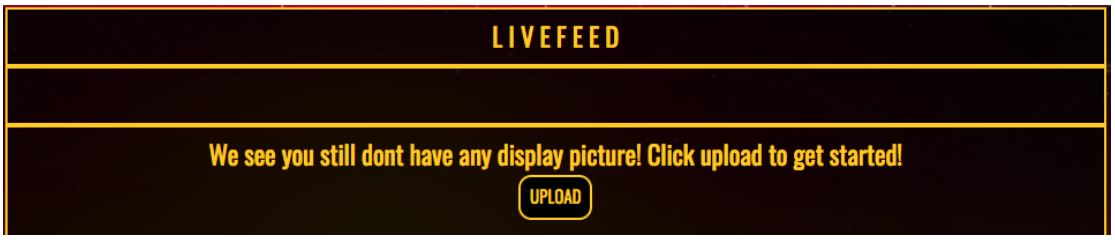
Figure: Icons in the top dock

ICON	FUNCTIONALITY
ActiveSpace	Navigates to user the user's homepage / live feed.
	Navigates to the Events present on ActiveSpace
	Navigates to the Edit Profile page populated with the user's data
	Navigates to the Pending Friend Requests/ Accepted Friend Requests along with timestamps of acceptances.
	Navigates to posts sent to the user from those in his network
	Navigates to visibility settings of the user where he controls who can see his profile + posts etc. Only him/Friends/FOF/Everyone
	User can view all his notification, which include a count of pending friend requests/ accepted requests/ posts from friends.
	User can log out from ActiveSpace and it destroys the current active session.

Icons found elsewhere in the page and their functionalities

ICON	FUNCTIONALITY
	User can like a diary entry only once, user can view liked the diary entry
	User can dislike a diary entry only once, user can view who disliked an entry
	User can comment on the diary entry via this button, page refreshes and comment is visible
	Search network bar button. Searches for users in a network with choices: Friends/Friend Of Friend /Everyone
	Search ActiveSpace entries bar button. Searches for keyword with choices: Posts / Diary Entry / Photos / Events / Location

Profile Photo: User can upload his profile photo, where all his friends, friends of friends or everyone in the network can view when they look the user up.



When the UPLOAD button is clicked, the user is greeted with the following page.

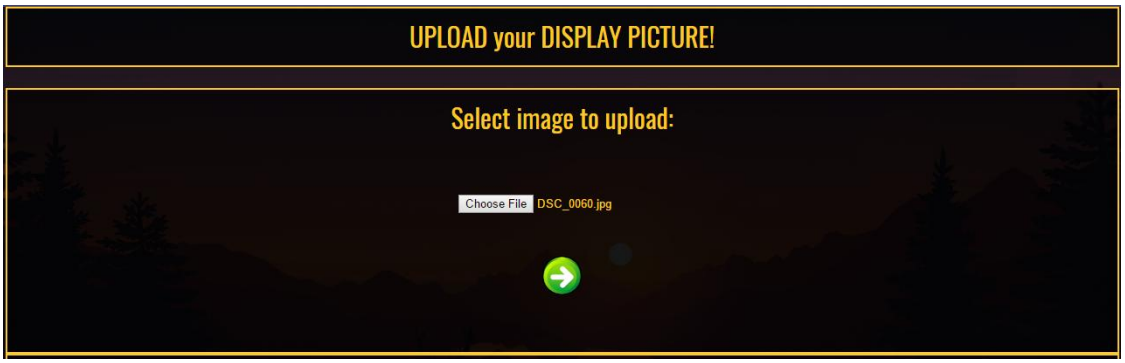


Figure: Adding a profile photo in ActiveSpace

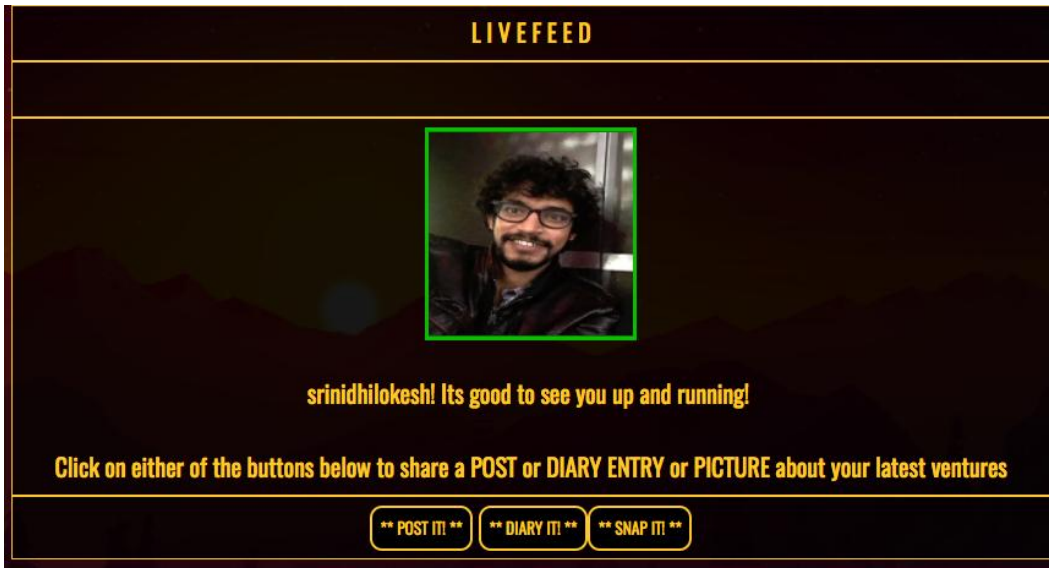


Figure: Livefeed after the profile picture is uploaded

Visibility: Visibility settings are accessible at any time and user can change it to Only me/Friends/Friends of Friends/Everyone. Every user can be looked up by a complete stranger however user can choose whom he can share his photos/diary/posts with by defining his preference here.

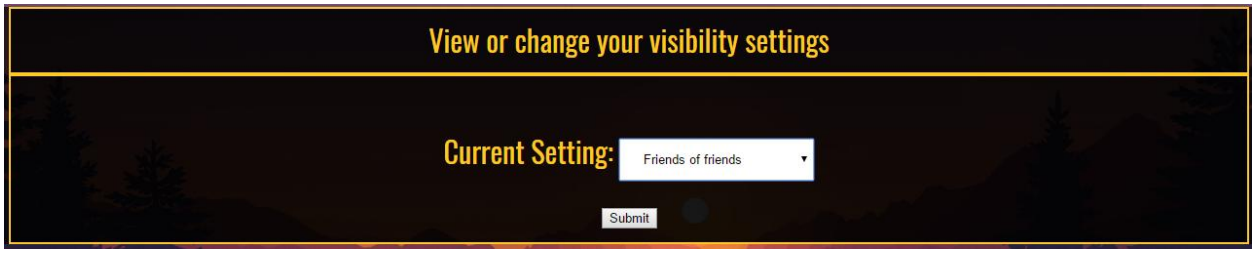


Figure: Setting to change network visibility in ActiveSpace

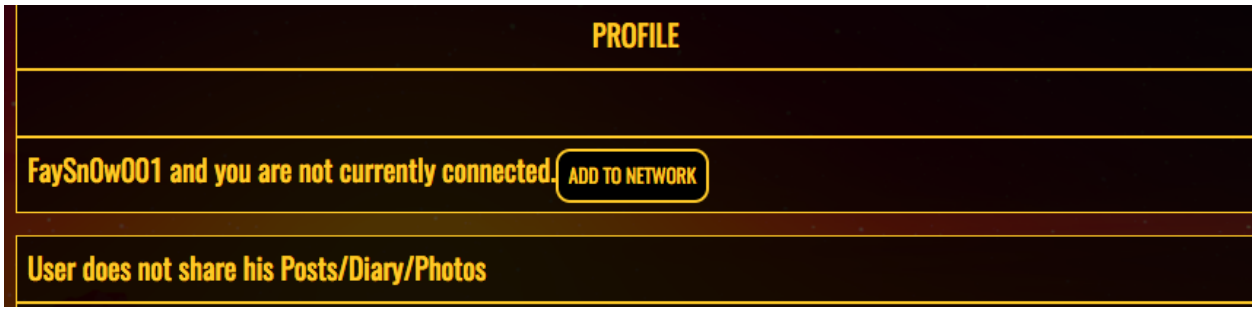


Figure: User with 'Only Me' setting is displayed above.

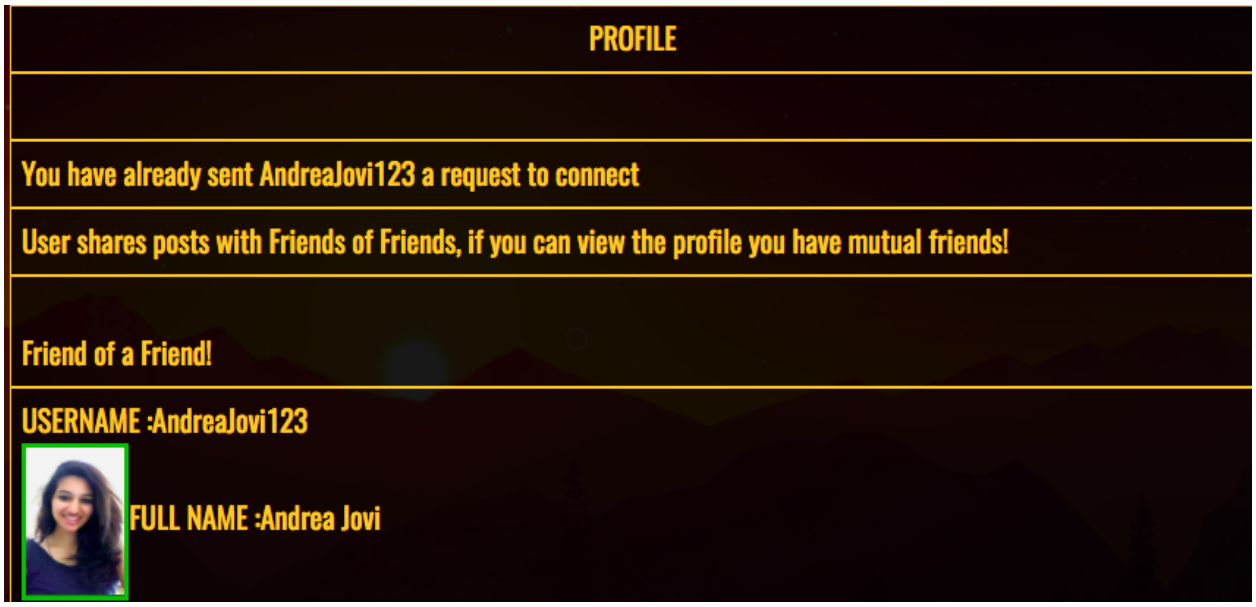


Figure: User with 'Friends of Friends' setting is displayed above.

User Sessions: Sessions are created during user login. The sessions contain user information required to be used across multiple web pages. This feature saves user state to distinguish between different users.

Events: Events in the area are listed in this page and users can join them. Users can also view other members attending these events.

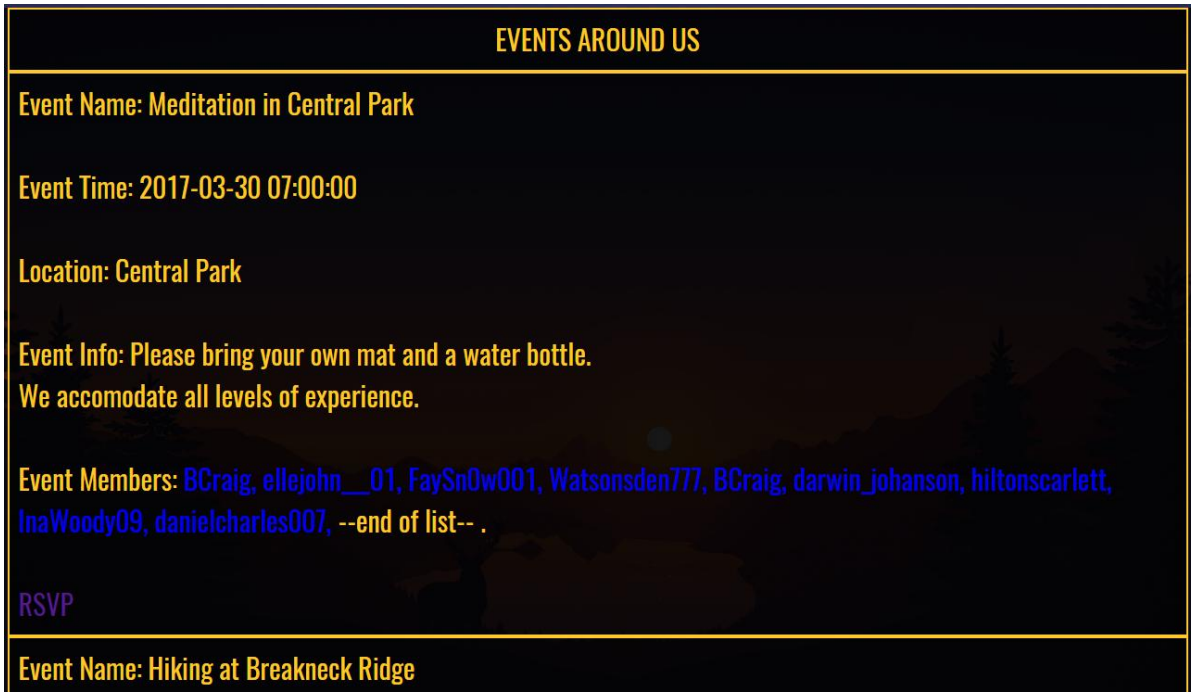


Figure: Happening events in ActiveSpace

3. Bibliography

- Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. *Database system concepts*. Vol. 4. New York: McGraw-Hill, 1997.
- PostgreSQL Documentation - postgresql.org/docs/
- W3Schools - w3schools.com/sql/
- Tutorialspoint - tutorialspoint.com/sql/
- Stack Overflow - stackoverflow.com