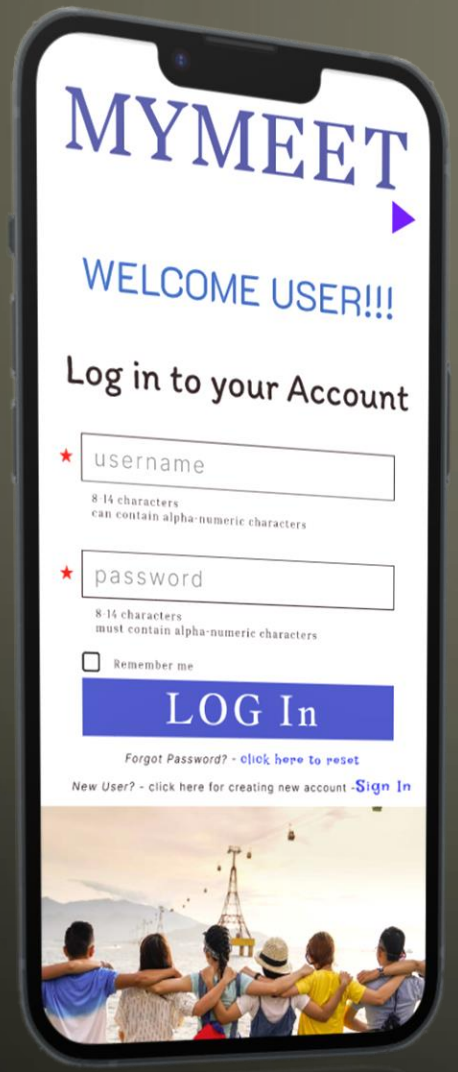
'MYMEET' PROJECT



Planning to take a break with your busy schedule and spend time with friends?

Wondering to connect with whom to meet? You have come to the perfect place.

MyMeet will help you to keep connected with friends easily in just 3 steps.

- Open Account & Create Friends
- Create a Note, Add Event details
- Connect with people and enjoy

SUMMARY

MyMeet application makes users' life happy catching up with their nearer ones. Users can publish events in the form of short notes with location and time, and then apply filter to these notes by radius and visibility. Other users can then receive these notes based on their own location and based on the visibility restriction. User can also comment on the notes visible to them. Users can also delete notes as well as comments.

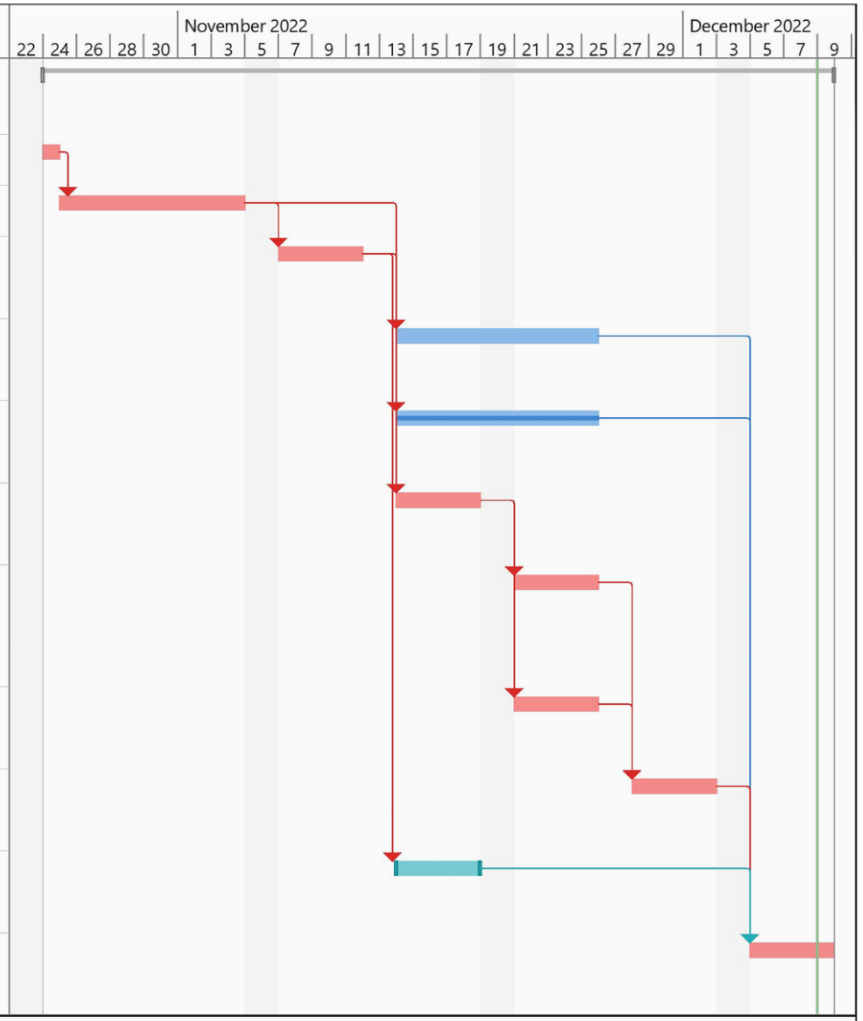
MYMEET

Our Mission-Vision

MYMEET is here to bring your life again in real zone. Get notified & meet your family, friends, relatives who are coming at a place near your house, workplace, or any other location, at a notified destination with no barriers. We are here for you to automatically set up your fun time with those who are near to you. No need to call - No sharing live location with anyone – Just type a text note and select a location and time of your future presence, your friends/public will get notified in a sec based on your visibility-restriction. Comment your friend's note and catch them at their event.

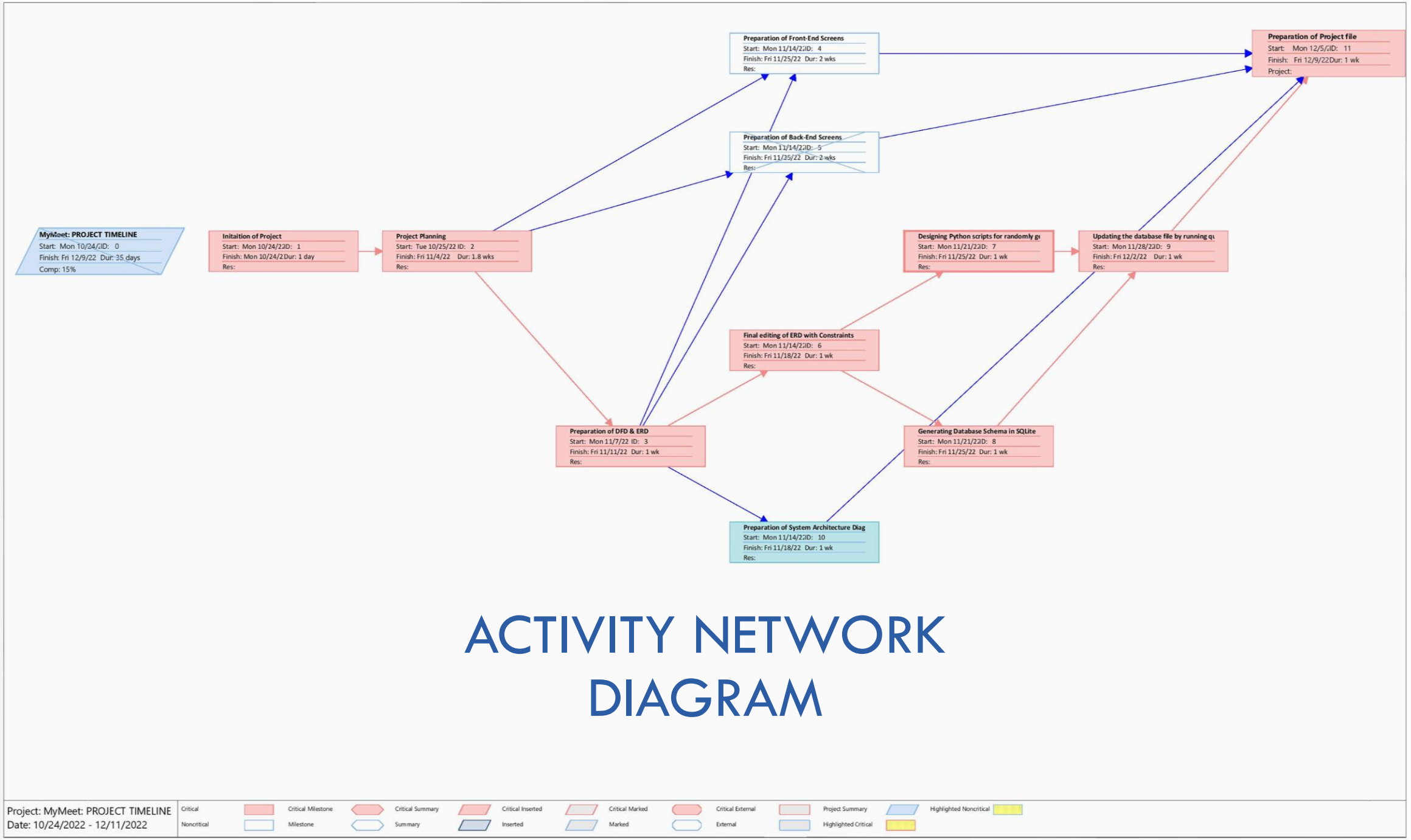
GANTT CHART

ID	Task Name	Duration	Start	Finish	Predecessors
0	**MyMeet: PROJECT TIMELINE**	**35 days**	**Mon 10/24/22**	**Fri 12/9/22**	
1	Initaition of Project	1 day	Mon 10/24/22	Mon 10/24/22	
2	Project Planning	1.8 wks	Tue 10/25/22	Fri 11/4/22	1
3	Preparation of DFD & ERD	1 wk	Mon 11/7/22	Fri 11/11/22	2
4	Preparation of Front-End Screens	2 wks	Mon 11/14/22	Fri 11/25/22	3,2
5	Preparation of Back-End Screens	2 wks	Mon 11/14/22	Fri 11/25/22	3,2
6	Final editing of ERD with Constraints	1 wk	Mon 11/14/22	Fri 11/18/22	3
7	Designing Python scripts for randomly generated data	1 wk	Mon 11/21/22	Fri 11/25/22	6
8	Generating Database Schema in SQLite	1 wk	Mon 11/21/22	Fri 11/25/22	6
9	Updating the database file by running queries	1 wk	Mon 11/28/22	Fri 12/2/22	8,7
10	Preparation of System Architecture	1 wk	Mon 11/14/22	Fri 11/18/22	3
11	Preparation of Project file	1 wk	Mon 12/5/22	Fri 12/9/22	10,9,4,5



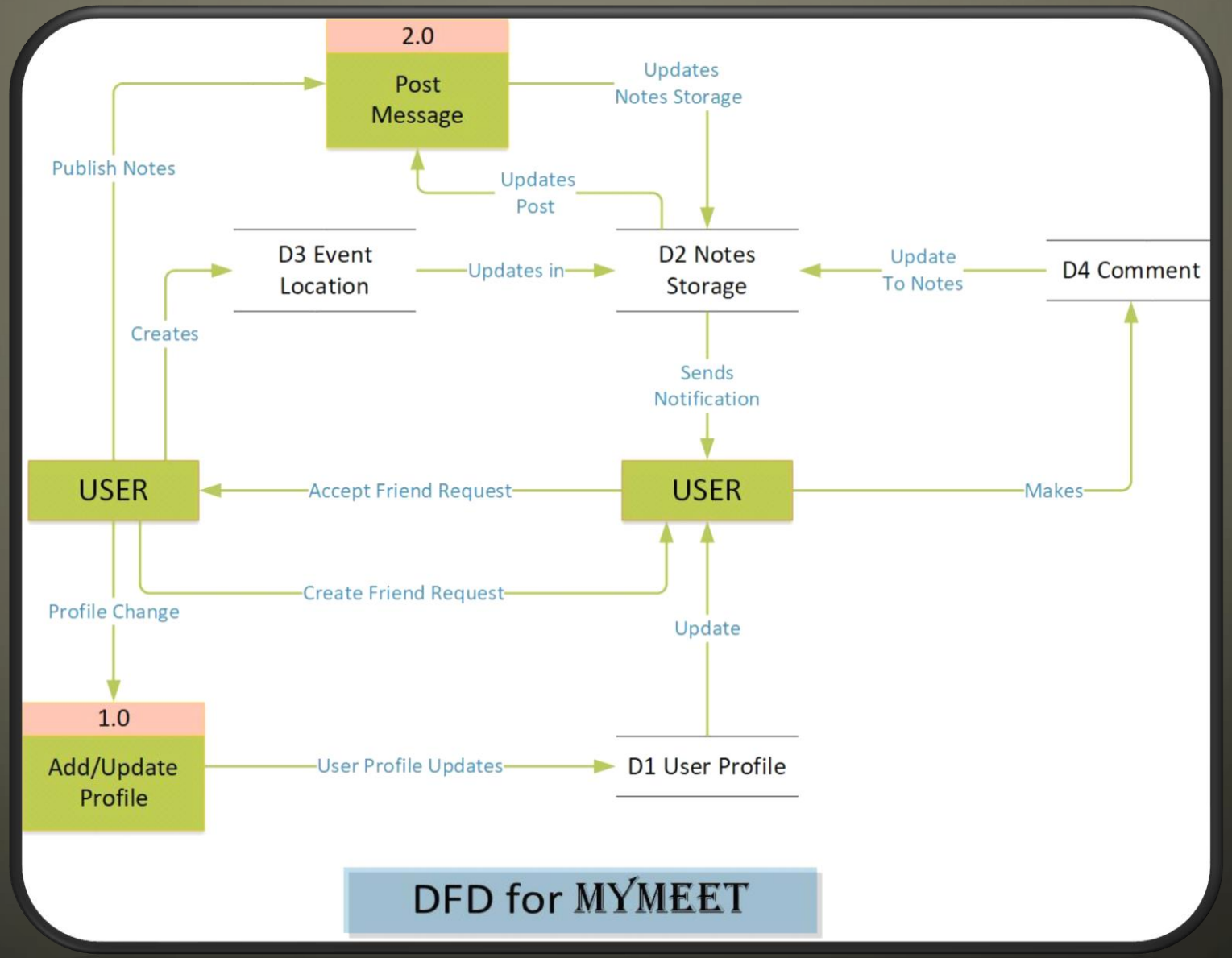
Project: MyMeet: PROJECT TIMELINE
Date: 10/24/2022 - 12/11/2022

Task
Split
Milestone
Summary
Project Summary
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
External Tasks
External Milestone
Deadline
Critical
Critical Split
Progress
Manual Progress



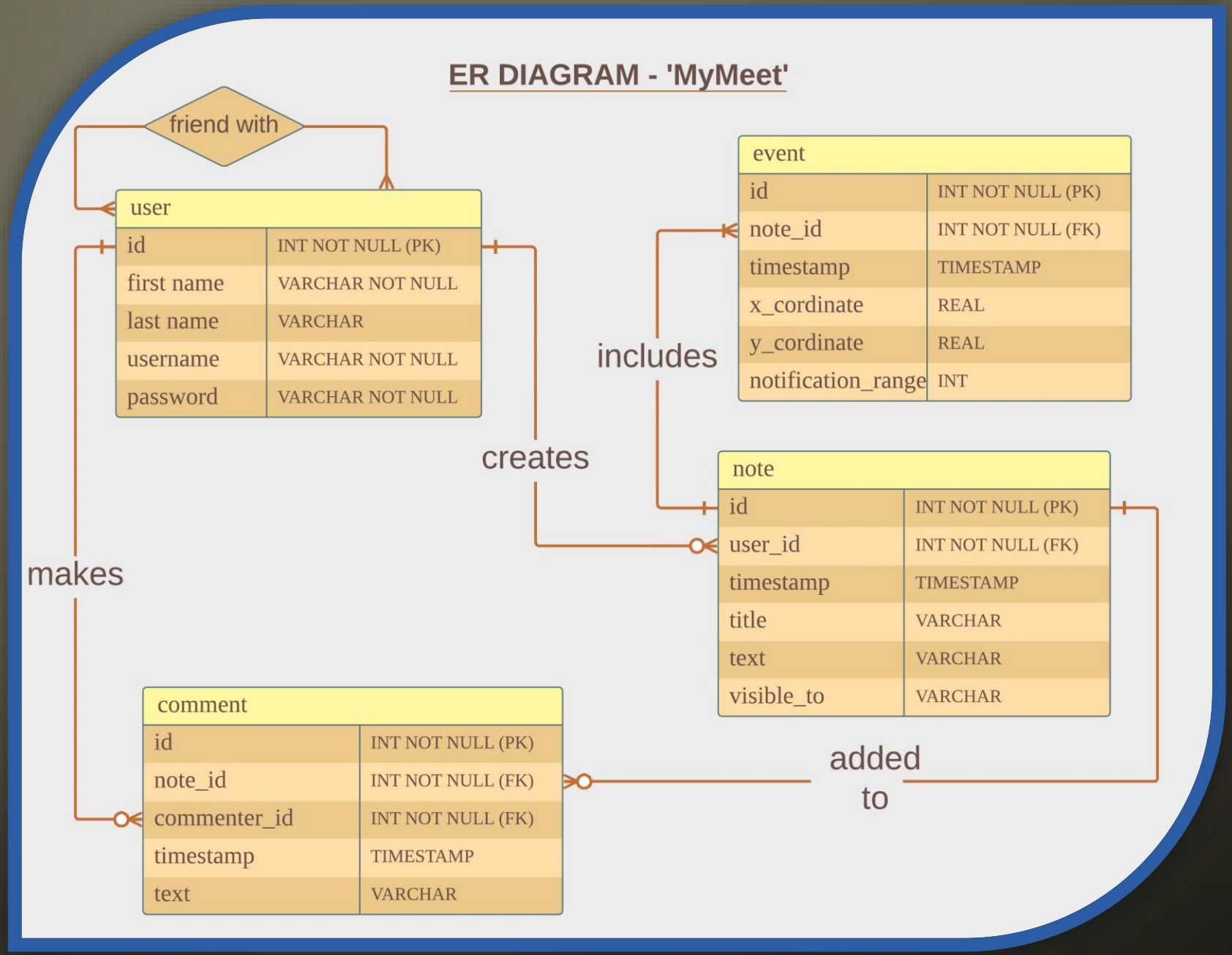
ACTIVITY NETWORK DIAGRAM

DATA FLOW DIAGRAM



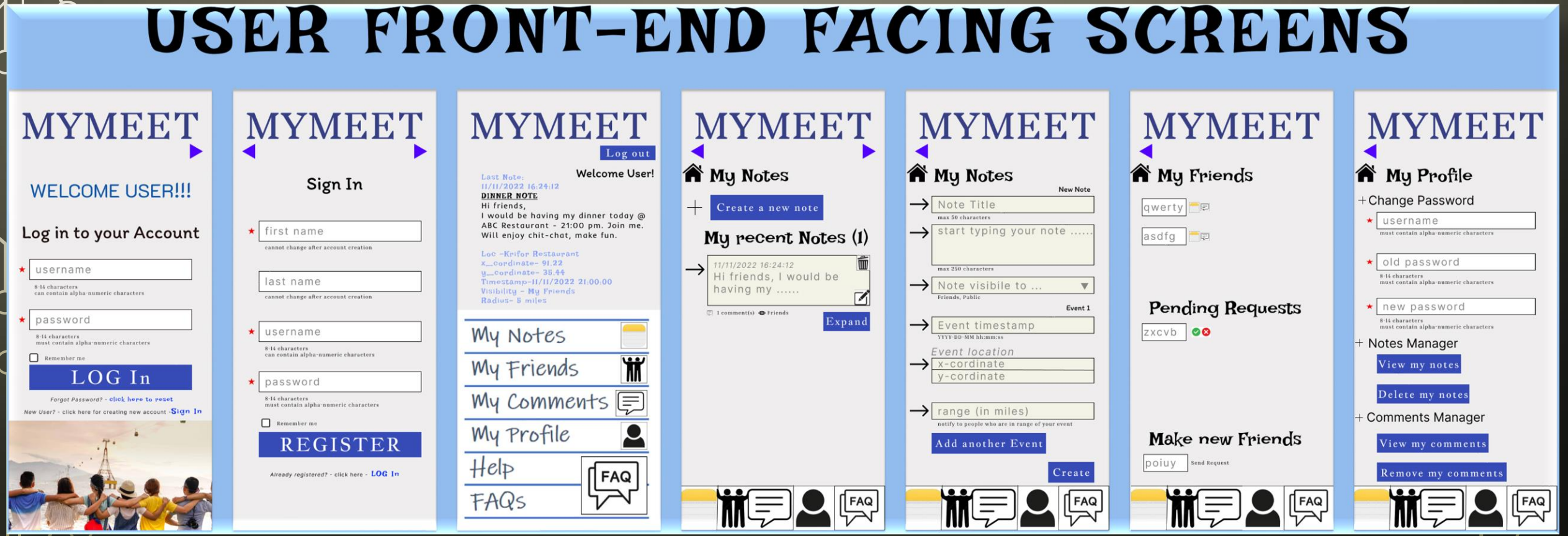
https://lucid.app/lucidchart/057a3d85-e29b-4c28-bfda-a27d443d84b9/edit?viewport_loc=-401%2C110%2C2519%2C1149%2C0DjlVPRn9anX7&invitationId=inv_47b1c330-b5ee-4395-9b45-6dbafa783b42

ENTITY-RELATIONSHIP DIAGRAM



https://lucid.app/lucidchart/215b9bfc-0766-43c5-8608-348fa2107b91/edit?viewport_loc=24%2C-37%2C2645%2C1403%2C0_0&invitationId=inv_d9d2572e-e0ff-4595-9fa1-a69c34f5ab45

USER-FLOW SCREENS



BACK-END SCREENS

REPORT

USER PROFILE TABLE

Generated By:	System Analyst
DATE	11/12/2022

user_id	first_name	last_name	username	password
1	yash	jivani	yashjivani	**********
2	john	dsilva	johndsilva	**********
3	rocky	string	rockystring	**********
4	mol	mang	molmang	**********
5	costan	jury	costanjury	**********

TOTAL USERS	5

REPORT

USER NOTES & COMMENTS

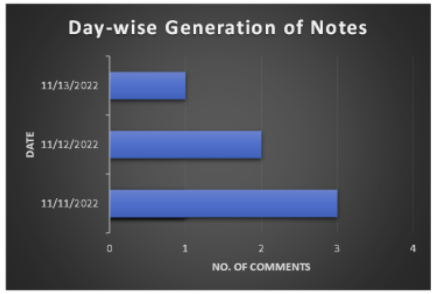
Generated By:	System Analyst
DATE	11/13/2022

created by username	Note Count	Note_timestamp	Event_timestamp	visibilty	comments
yashjivani					
	1	11/11/2022 16:24	11/11/2022 21:00	friends	1
johndsilva					
	1	11/12/2022 14:24	11/12/2022 18:00	public	2
	2	11/13/2022 13:24	11/13/2022 19:00	friends	1
rockystring					
	1	11/11/2022 16:55	11/11/2022 22:00	public	3
	2	11/13/2022 11:24	11/13/2022 8:00	public	0
molmang					
	1	11/11/2022 19:24	11/11/2022 22:30	public	1
costanjury					
TOTAL Notes	6		TOTAL Comments		8

REPORT

USER NOTES Vs COMMENTS

Generated By:	System Analyst
DATE	11/13/2022

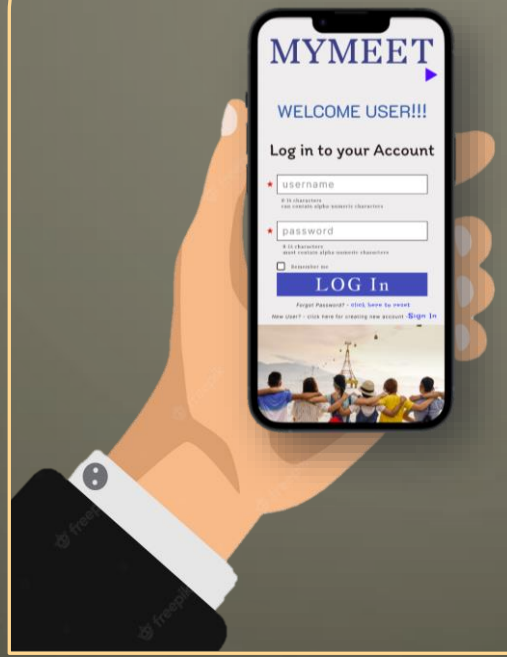


TOTAL Notes	6
TOTAL Comments	8

Create Notes

Display Content

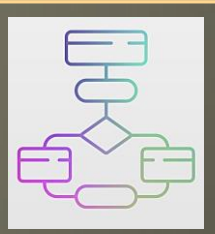
User interaction



FRONT-END

Request

Response



App Logic

SERVER

System

Database(SQL)

BACK-END

SYSTEM ARCHITECTURE DIAGRAM

Source: https://litslink.com/blog/web-application-architecture

Compliance & Regulations

- The “Least Privilege” philosophy calls for users and applications to have only the minimal privileges they actually require to function properly
- User Credentials are not used at any point expect for login purposes

Data Retention Policy

- Users’ log data is kept in system for 1 day in back-end data base system for analysts
- Comments, notes & events data is deleted from the database when users prompt to delete

PROJECT RECOMMENDATIONS

1. Create Map feature to easily navigate users for selection of event location
2. Create Comment feature for each event posted by the user
3. Create Chat feature for users to chat with others personally
4. Add new functionality of storing user's location and timestamp in the database and deleting using triggers
5. Add new filter tags based on location, comments, schedule, event start and end time .
6. Add comment tags like – will visit/ not able to visit, etc in comment table
7. Make data input in the system to flow in natural way.
8. Increase data set to analyze the trends of users in Data visualization software

APPENDIX - Generation of Random Data in 'SQLTableValues' object

data_generate.py > SQLTableValues

```python
import random as r

class SQLTableValues:
    '''creates data for user table from raw txt file'''
    def __init__(self,raw_file = 'raw_file.txt'):
        self.raw_file = raw_file
        self.note_count = r.randint(15,100)
        self.comments_no = r.randint(15,200)
        self.users_count = 0
        self.fileName_list = []
        self.raw_file_opener()

        self.user_dict = {}
        self.friend_dict = {}
        self.note_dict = {}
        self.event_dict = {}
        self.comment_dict = {}

    def raw_file_opener(self):
        '''opens file consisting of names'''
        file_handle = open(self.raw_file,'r')
        for line in file_handle:
            self.fileName_list.append(line.strip().split())
            self.users_count += 1

    def create_user_dict(self):
        '''k: user_id, v:list of [first_name, last_name, username, password]'''
        name = self.fileName_list
        username = self.gen_username()
        password = self.gen_password()
        for user_id in range(self.users_count):
            self.user_dict[user_id] = [name[user_id][0], name[user_id][1], username[user_id], password[user_id]]

    def gen_username(self):
        username_list = []
        for i in range(self.users_count):
            final = ''.join(self.fileName_list[i])
            if len(final) < 8:
                final += str(r.randint(0,9)) * (r.randint(8,14) - len(final))
            elif len(final) > 14:
                final = final[:r.randint(9,15)]
            username_list.append(final)
        return username_list

    def gen_password(self):
        password_list = []
        for _ in range(self.users_count):
            passsword = ''
            for _ in range(r.randint(8,14)):
                digits = str(r.randint(0,9))
                upper_case_alpha = chr(r.randint(65,90))
                lower_case_alpha = chr(r.randint(97,122))

                passsword += r.choice([digits,upper_case_alpha,lower_case_alpha])

            password_list.append(passsword)
        return password_list
```

data_generate.py > SQLTableValues

```python
class SQLTableValues:

    def create_friend_dict(self):
        '''k: user_id, v: [friend_id]'''
        records_count = self.users_count
        user_id_list = r.sample(range(1,records_count),15) #creation of randomly generated sample from total
        for user_id in user_id_list:
            for _ in range(1,r.choice(range(1,10))): # creation of friends per user
                friend_id = r.randint(1,50)
                if user_id != friend_id:

                    if user_id not in self.friend_dict:
                        self.friend_dict[user_id] = [friend_id]
                    else:
                        self.friend_dict[user_id].append(friend_id)

                    if friend_id not in self.friend_dict:
                        self.friend_dict[friend_id] = [user_id]
                    else:
                        self.friend_dict[friend_id].append(user_id)

    def create_note_dict(self):
        ''' creates dict k: note_id, v: list- [user_id, title, text, visible_to]'''
        notes_list = self.gen_notes()

        for i in range(self.note_count):
            for _ in range(r.randint(1,10)): #no. of notes per user
                visibility = r.choice(['public','friends'])
                self.note_dict[i+1] = [r.randint(1,self.users_count), notes_list[i][0], notes_list[i][1], visibility]

    def gen_notes(self):
        notes_list = []
        for _ in range(self.note_count):
            title = 'wefn'
            text = 'ewe'
            notes_list.append([title,text])
        return notes_list

    def create_event_dict(self):
        '''dict- k:note_id, v: list of [x_cordinate, y_cordinate, notification_range]'''
        for note_id in self.note_dict:
            for _ in range(1,r.randint(1,5)): # #no. of event per note
                x_cordinate = r.uniform(-180,180)
                y_cordinate = r.uniform(-90,90)
                ntf_range = r.randint(1,20)
                self.event_dict[note_id] = [x_cordinate, y_cordinate,ntf_range]

    def create_comment_dict(self):
        '''dict k: comment_id , v: list of [note_id, commenter_id, text]'''
        #creation of randomly generated sample of notes to be commented from total
        sample_note_list = r.sample(range(1,self.note_count+1),self.note_count)
        comment_id = 1
        for note_id in sample_note_list:
            for comment_count in range(1,r.randint(1,10)):   #no. of comments per
                if self.note_dict[note_id][-1] == 'public':
                    commenter_id = r.randint(1,self.users_count)
                else:
                    user_id = self.note_dict[note_id][0]
                    if user_id in self.friend_dict:
                        commenter_id = r.choice(self.friend_dict[user_id])
                text = 'wfev'
                self.comment_dict[comment_id] = [note_id, commenter_id, text]
                comment_id += 1
```

APPENDIX - 'SQL' object inheritence with 'SQLTableValues' object

admin.py > SQL

```python
import data_generate as data
import sqlite3

class SQL(data.SQLTableValues):
    def __init__(self,db_file = 'mymeet.db', ):
        super().__init__()
        self.db_file = db_file

    def create_connection(self):
        ''' create a database connection to the SQLite database specified by db_file '''
        try:
            self.conn = sqlite3.connect(self.db_file)
            self.cur = self.conn.cursor()
        except sqlite3.Error as e:
            print(e)

    def create_user_table(self):
        drop_table_sql = '''DROP TABLE IF EXISTS user;'''
        create_table_sql = '''CREATE TABLE user(
            id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
            first_name VARCHAR NOT NULL,
            last_name VARCHAR,
            username VARCHAR CHECK ((LENGTH(password) >= 8) AND (LENGTH(password) <= 14)) UNIQUE NOT NULL,
            password VARCHAR CHECK ((LENGTH(password) >= 8) AND (LENGTH(password) <= 14) AND (username != password)) UNIQUE NOT NULL
        );'''
        self.create_connection()
        try:
            self.cur.execute(drop_table_sql)
            self.cur.execute(create_table_sql)
        except sqlite3.Error as e:
            print(e)
        else:
            print('user_table created')

    def insert_user_record(self):
        self.create_connection()

        insert_user_sql = '''INSERT INTO user (first_name,last_name,username,password)
            VALUES (?,?,?,?)'''
        self.create_user_dict()
        try:
            for user_id in self.user_dict:
                record = self.user_dict[user_id]
                self.cur.execute(insert_user_sql, record)
                self.conn.commit()
        except sqlite3.Error as e:
            print(e)
        else:
            print('user_table: values inserted')

```

admin.py > SQL

```python
class SQL(data.SQLTableValues):

    def create_friend_table(self):
        drop_table_sql = '''DROP TABLE IF EXISTS friend;'''
        create_table_sql = '''CREATE TABLE friend (
            user_id INTEGER,
            friend_id INTEGER CHECK (friend_id != user_id),
            FOREIGN KEY (user_id)
            REFERENCES user (user_id),
            FOREIGN KEY (friend_id)
            REFERENCES user (user_id)
        );'''
        self.create_connection()
        try:
            self.cur.execute(drop_table_sql)
            self.cur.execute(create_table_sql)
        except sqlite3.Error as e:
            print(e)
        else:
            print('friend_table created')

    def insert_friend_record(self):
        self.create_connection()

        insert_friend_sql = '''INSERT INTO friend (user_id, friend_id)
            VALUES (?,?)'''
        self.create_friend_dict()
        try:
            for user_id in sorted(self.friend_dict):
                for friend_id in sorted(set(self.friend_dict[user_id])):
                    record = (user_id, friend_id)
                    self.cur.execute(insert_friend_sql, record)
                    self.conn.commit()
        except sqlite3.Error as e:
            print(e)
        else:
            print('friend_table: values inserted')
```

APPENDIX - 'SQL' object inheritence with 'SQLTableValues' object

admin.py > SQL

```
4   class SQL(data.SQLTableValues):
86
87      def create_note_table(self):
88          drop_table_sql = '''DROP TABLE IF EXISTS note;'''
89          create_table_sql = '''CREATE TABLE note(
90              id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
91              user_id INTEGER NOT NULL,
92              timestamp TIMESTAMP DEFAULT (DATETIME('now','localtime')),
93              title VARCHAR CHECK (LENGTH(title <= 50)),
94              text VARCHAR CHECK (LENGTH(text <= 250)),
95              visible_to VARCHAR CHECK (visible_to = 'public' or visible_to = 'friends'),
96              FOREIGN KEY (user_id)
97              REFERENCES user (user_id)
98                  ON UPDATE CASCADE
99                  ON DELETE CASCADE
100         );'''
101         self.create_connection()
102         try:
103             self.cur.execute(drop_table_sql)
104             self.cur.execute(create_table_sql)
105         except sqlite3.Error as e:
106             print(e)
107         else:
108             print('note_table created')
109
110     def insert_note_record(self):
111         self.create_connection()
112
113         insert_note_sql = '''INSERT INTO note (user_id, title, text, visible_to)
114             VALUES (?,?,?,?)'''
115         self.create_note_dict()
116         try:
117             for note_id  in self.note_dict:
118                 record = self.note_dict[note_id]
119                 self.cur.execute(insert_note_sql, record)
120                 self.conn.commit()
121         except sqlite3.Error as e:
122             print(e)
123         else:
124             print('note_table: values inserted')
125
```

admin.py > SQL > create_event_table

```
4   class SQL(data.SQLTableValues):
126     def create_event_table(self):
127         drop_table_sql = '''DROP TABLE IF EXISTS event;'''
128         create_table_sql = '''CREATE TABLE event (
129             id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
130             note_id INTEGER NOT NULL,
131             timestamp TIMESTAMP
132                 DEFAULT (DATETIME(DATETIME('now','localtime'),'+'
133                 || CAST((ABS(RANDOM()) % (300-1) + 1) AS TEXT) || ' minutes')),
134             x_cordinate REAL,
135             y_cordinate REAL,
136             notification_range INT,
137             FOREIGN KEY (note_id)
138             REFERENCES note (note_id)
139                 ON UPDATE CASCADE
140                 ON DELETE CASCADE
141         );'''
142         self.create_connection()
143         try:
144             self.cur.execute(drop_table_sql)
145             self.cur.execute(create_table_sql)
146         except sqlite3.Error as e:
147             print(e)
148         else:
149             print('event_table created')
150
151     def insert_event_record(self):
152         self.create_connection()
153
154         insert_event_sql = '''INSERT INTO event (note_id, x_cordinate, y_cordinate, notification_range)
155             VALUES (?,?,?,?)'''
156         self.create_event_dict()
157         try:
158             for note_id in self.event_dict:
159                 record = [note_id]
160                 record.extend(self.event_dict[note_id])
161                 self.cur.execute(insert_event_sql, record)
162                 self.conn.commit()
163         except sqlite3.Error as e:
164             print(e)
165         else:
166             print('event_table: values inserted')
167
```

APPENDIX - 'SQL' object inheritence with 'SQLTableValues' object

admin.py > SQL > integrity_check

```
4   class SQL(data.SQLTableValues):
168
169     def create_comment_table(self):
170         drop_table_sql = '''DROP TABLE IF EXISTS comment;'''
171         create_table_sql = '''CREATE TABLE comment (
172             id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
173             note_id INTEGER NOT NULL,
174             commenter_id INTEGER NOT NULL,
175             timestamp TIMESTAMP DEFAULT (DATETIME('now','localtime')),
176             text VARCHAR,
177             FOREIGN KEY (note_id)
178             REFERENCES note (note_id)
179                 ON UPDATE CASCADE
180                 ON DELETE CASCADE,
181             FOREIGN KEY (commenter_id)
182             REFERENCES user (user_id)
183                 ON UPDATE CASCADE
184                 ON DELETE CASCADE
185         );'''
186         self.create_connection()
187         try:
188             self.cur.execute(drop_table_sql)
189             self.cur.execute(create_table_sql)
190         except sqlite3.Error as e:
191             print(e)
192         else:
193             print('comment_table created')
194
195     def insert_comment_record(self):
196         self.create_connection()
197
198         insert_comment_sql = '''INSERT INTO comment (note_id, commenter_id, text)
199             VALUES (?,?,?)'''
200         self.create_comment_dict()
201         try:
202             for comment_id  in self.comment_dict:
203                 record = self.comment_dict[comment_id]
204                 self.cur.execute(insert_comment_sql, record)
205                 self.conn.commit()
206         except sqlite3.Error as e:
207             print(e)
208         else:
209             print('comment_table: values inserted')
210
211     def integrity_check(self):
212         try:
213             check = self.cur.execute('PRAGMA integrity_check;')
214             self.conn.commit()
215         except sqlite3.Error as e:
216             print(e)
217         else:
218             print('Integrity Check:', check.fetchall())
219
```

```
DB = SQL()
DB.create_user_table()
DB.create_friend_table()
DB.create_note_table()
DB.create_event_table()
DB.create_comment_table()
print()
DB.integrity_check()
print()
DB.insert_user_record()
DB.insert_friend_record()
DB.insert_note_record()
DB.insert_event_record()
DB.insert_comment_record()
```

```
user_table created
friend_table created
note_table created
event_table created
comment_table created

Integrity Check: [('ok',)]

user_table: values inserted
friend_table: values inserted
note_table: values inserted
event_table: values inserted
comment_table: values inserted
```

Generation of Database File using SQL object

APPENDIX - SQLite queries

```sql
1) users who has friend_id -NULL (users who haven't made any friends):
SELECT
    id as users_with_no_friends
FROM user
WHERE id NOT IN
    (SELECT
        DISTINCT user_id
    FROM friend);

2) users who haven't posted any notes yet
SELECT
    id AS users_with_no_notes
FROM user
WHERE id NOT IN
    (SELECT
        DISTINCT user_id
    FROM note);

3) user who have not commented in any of the notes
SELECT
    id
FROM user
WHERE id NOT IN
    (SELECT
        DISTINCT commenter_id
    FROM comment);

4) user who are most inactive (intersection of 1 & 2 & 3)
SELECT
    id,
    COUNT(id)
FROM
    (SELECT
        id
    FROM user
    WHERE id NOT IN
        (SELECT
            DISTINCT user_id
        FROM friend)
    UNION ALL
    SELECT
        id
    FROM user
    WHERE id NOT IN
        (SELECT
            DISTINCT user_id
        FROM note)
    UNION ALL
    SELECT
        id
    FROM user
    WHERE id NOT IN
        (SELECT
            DISTINCT commenter_id
        FROM comment))
GROUP BY id;
```

	users_with_no_friends
1	35
2	22
3	8
4	16
5	24
6	44
7	32

	users_with_no_notes
1	42
2	31
3	39
4	16
5	4
6	3
7	41

	id
1	24
2	44
3	18
4	29
5	20

	id	COUNT(id)
1	2	1
2	3	1
3	4	1
4	8	1
5	11	1
6	16	2
7	18	2
8	20	2
9	22	1
10	24	2
11	29	1
12	31	1
13	32	1
14	35	1
15	39	1
16	40	1
17	41	1
18	42	1
19	44	2
20	46	1
21	47	2
22	50	1

```sql
5) no. of friends for each of the users
SELECT
    id,
    COUNT(DISTINCT friend.friend_id)
FROM user
LEFT JOIN friend ON
    user.id = friend.user_id
GROUP BY id;

6) compare the count of notes published - friends vs public
SELECT visible_to, COUNT(visible_to) from note GROUP BY visible_to;

7) no.of comments per notes for all users
SELECT
    user.id,
    note.id as note_id,
    COUNT(comment.commenter_id)
FROM user
INNER JOIN note ON
    user.id = note.user_id
LEFT JOIN comment ON
    note.id = comment.note_id
GROUP BY user.id
ORDER BY COUNT(comment.commenter_id);

8) no. of notes per user
SELECT
    user.id as user_id,
    COUNT(note.id) as total_notes
FROM user
LEFT JOIN note ON
    user.id = note.user_id
GROUP BY user.id;

9)no.of events per note
SELECT
    note.id,
    COUNT(event.id)
FROM note
LEFT JOIN event ON
    note.id = event.note_id
GROUP BY note.id;

10) Total no. of connections btwn users
SELECT COUNT(*)/2 FROM friend;
```

	id	COUNT(DISTINCT friend.friend_id)
1	1	2
2	2	0
3	3	2
4	4	7
5	5	3
6	6	2
7	7	1

	visible_to	COUNT(visible_to)
1	friends	37
2	public	26

	id	note_id	COUNT(comment.commenter_id)
1	26	24	0
2	43	36	0
3	22	14	1
4	24	17	1
5	7	56	2
6	20	22	2

	user_id	total_notes
1	1	2
2	2	1
3	3	0
4	4	0
5	5	1
6	6	2
7	7	1

	id	COUNT(event.id)
1	1	4
2	2	4
3	3	1
4	4	4

	COUNT(*) / 2
1	58

REFERENCES

- https://www.iconfinder.com/
- https://litslink.com/blog/web-application-architecture