Project Design Report
Team B – Improve Campus Morale

2 August 2019

Christopher Morcom
William Javier Purificacion

CS 487 – Software Engineering

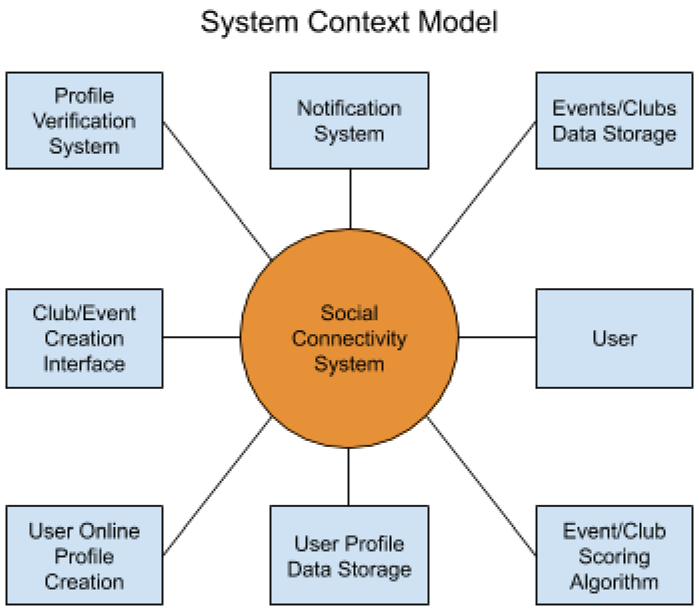
[1] Summary

Goal: The goal of this application is for the user to have an easier time to be engaged in university events, clubs, activities, or more. All of the user's inputs regarding their university/major/preferences will affect how the application will make notifications/reminders for the user.

Assumption: All the information shown in the application (which includes contact information of people/clubs/etc.) are already available online. The application makes it simpler for the user to see all of this information, rather than in some other inefficient way (Ex. Via email. We all know that our inbox gets messy, and we lose interest on checking every single email if there's a lot of clutter.)

Design Summary: The user will choose all the information that they want to see from their university based on their preferences. The user will get notifications based on their preferences. Every time there are new events that correspond to the user's interests (or anything in particular), the user will be notified. All of the user's information will be stored locally within the application. There is no need to have the user's data stored online because limited information is needed in order for the application to function.

[2] System Context Modeling

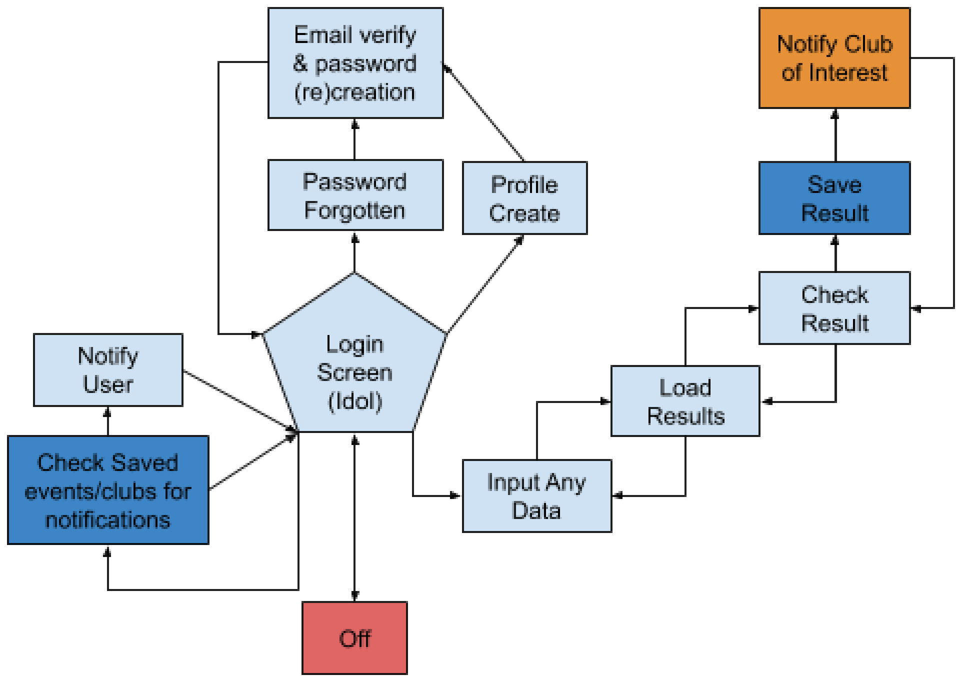

System Context Model
Profile Verification System
Notification System
Events/Clubs Data Storage
Club/Event Creation Interface
Social Connectivity System
User
User Online Profile Creation
User Profile Data Storage
Event/Club Scoring Algorithm

Assumption: The application is universal. It will work on all students as long as the university is registered in the list. It is still up to debate whether to make this application separately for each universities that want to use it. (This detail is very important for the *Profile Verification System*)

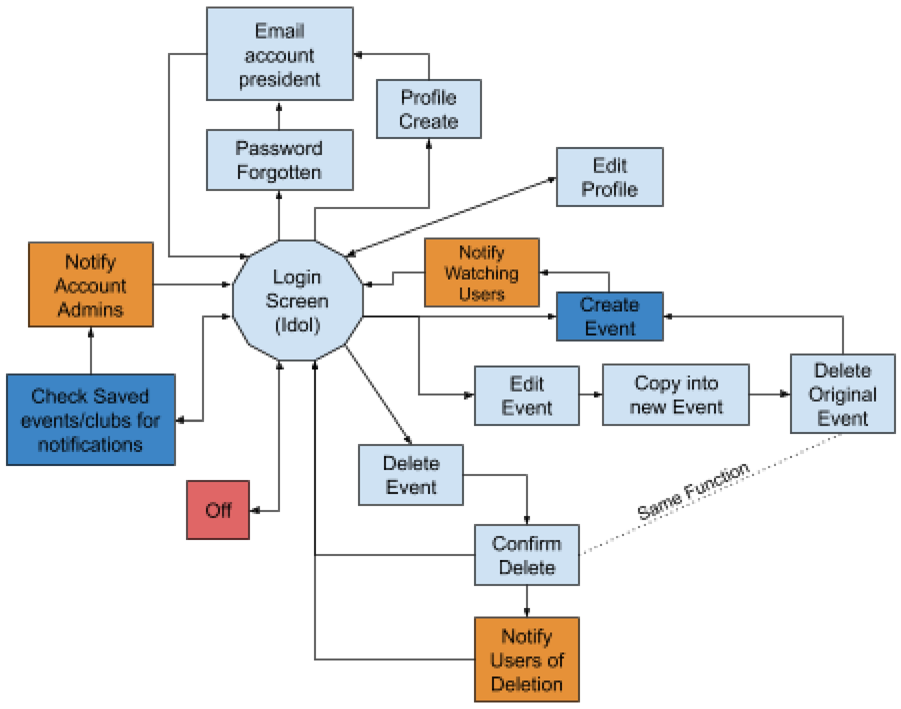
Description: The user creates an account using their university account. Once the account is verified, the user can see information from the university which includes classes, major description, ongoing events, club activities, etc. The user can create events. The user will choose their preferences which will affect how the user will be notified. All of the information received will be stored into the university's data storage.

[3] User Perspectives & State Transition Diagrams

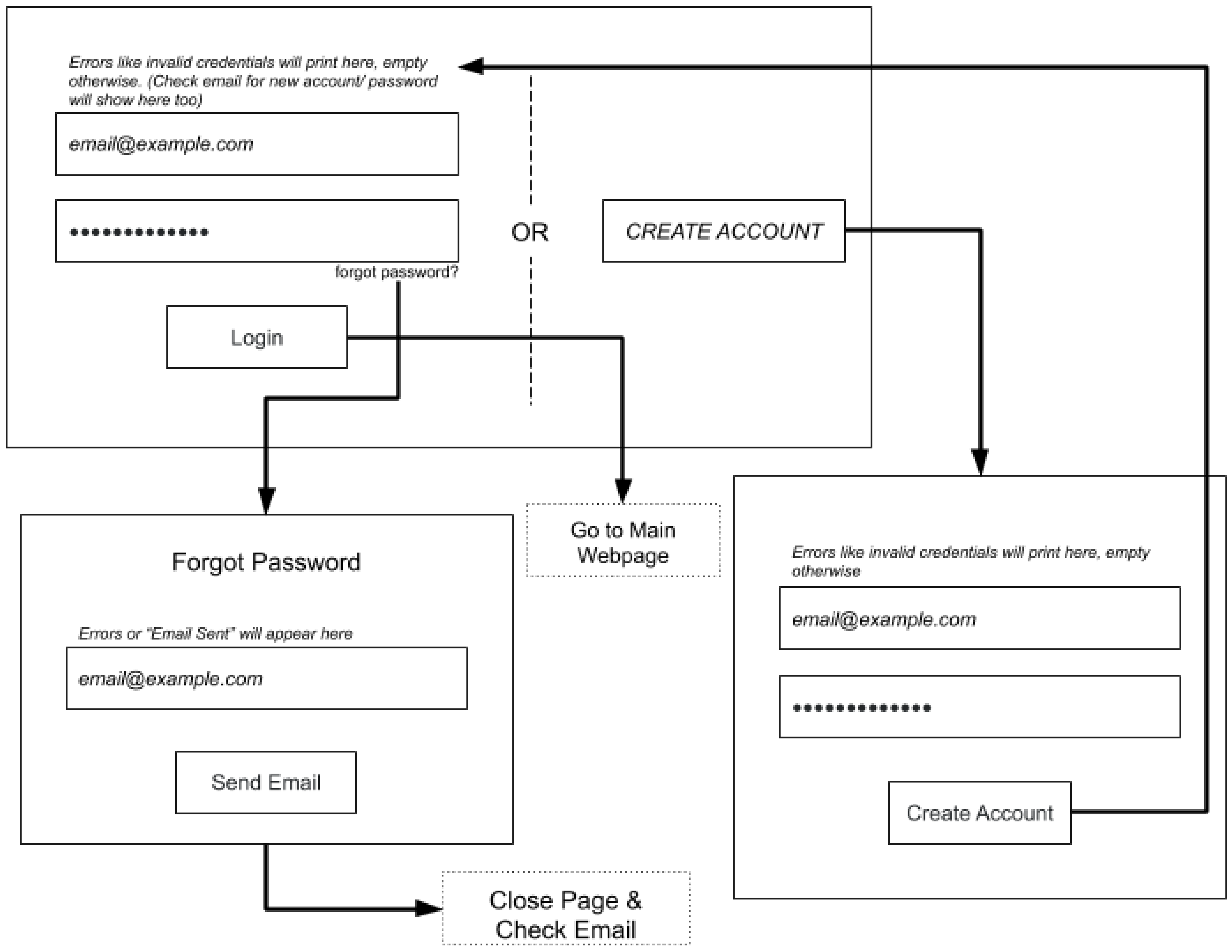
User Account State Transition Diagram



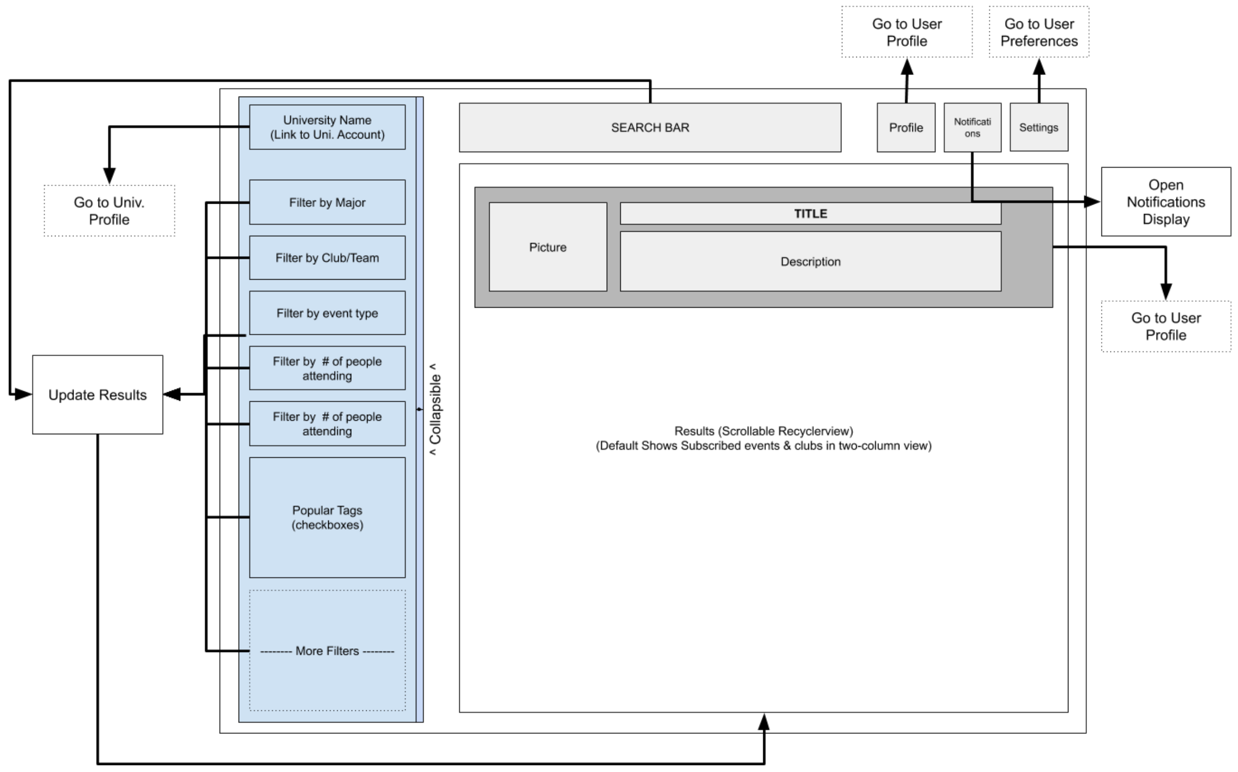
Club Account State Transition Diagram



Login Screen (Flowchart):


Errors like invalid credentials will print here, empty otherwise. (Check email for new account/ password will show here too)
email@example.com
••••••••••••••
forgot password?
Login
OR
CREATE ACCOUNT
Go to Main Webpage
Forgot Password
Errors or "Email Sent" will appear here
email@example.com
Send Email
Close Page & Check Email
Errors like invalid credentials will print here, empty otherwise
email@example.com
••••••••••••••
Create Account

Main Webpage:


Go to User Profile
Go to User Preferences
SEARCH BAR
Profile
Notifications
Settings
University Name (Link to Uni. Account)
Go to Univ. Profile
Filter by Major
Filter by Club/Team
Filter by event type
Filter by # of people attending
Filter by # of people attending
Popular Tags (checkboxes)
-------- More Filters --------
^ Collapsible ^
Update Results
Picture
TITLE
Description
Open Notifications Display
Go to User Profile
Results (Scrollable Recyclerview)
(Default Shows Subscribed events & clubs in two-column view)

[4] Dummy Functions and Algorithmic Analysis

Runtime Analysis:

reset_password():

- This takes exactly 30 minutes at the most. This is the most vulnerable part of the system where if K users forget their passwords at the exact same time, O(k) space is taken for 30 minutes

notify():

- O(1) for notifying a single user since it is bound by indexing a defaultdict

score_relevance():

- TF-IDF index builds in O(mn) time complexity where m is the length of the longest document and n is the number of documents. Space complexity is relatively similar at O(k+mn) where k is the number of unique terms in the collection. However, our data structures permit us an indexing time close to θ(1) since we use a python defaultdict.

search():

- Searching takes θ(nlog(n)) and O(n2) time since it is bound by our sorting mechanism.

edit/create/delete_event():

- Takes O(1) since it is bound by indexing a defaultdict

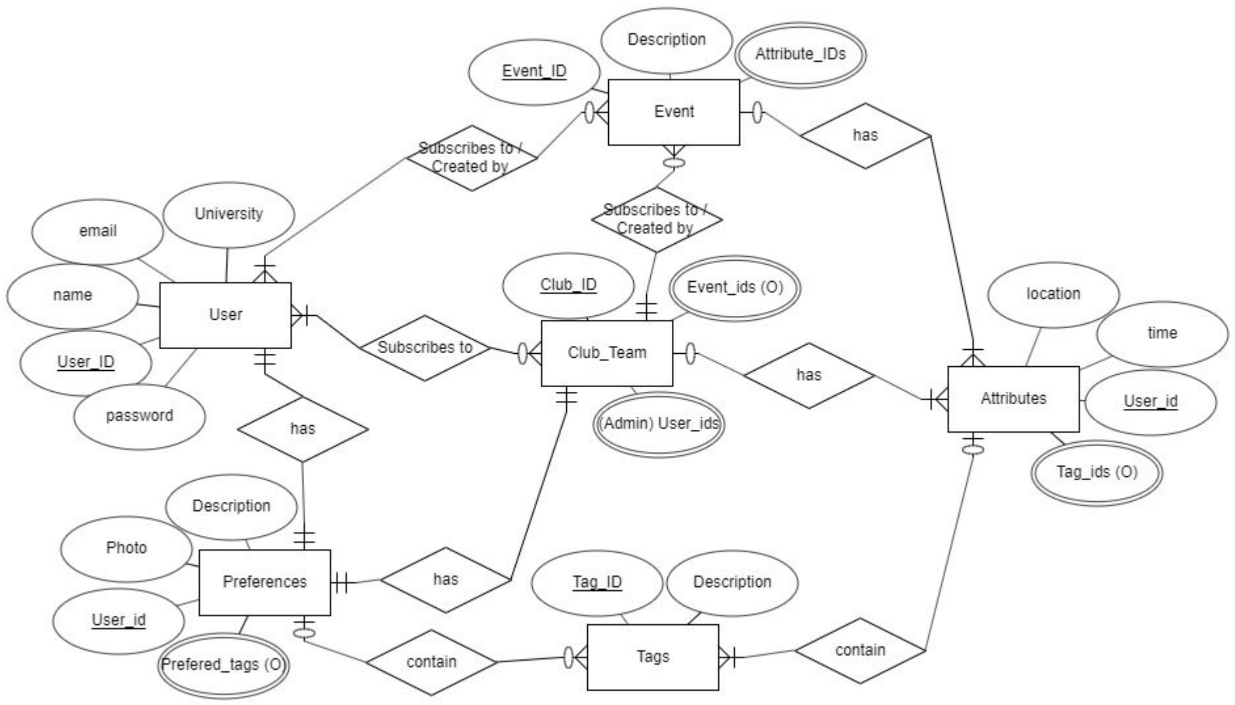
Pseudocode:

```
def reset_password(email):
    for user in system.users:
        if email == user.email:
            reset_link =gen_ssl_link(hash())
            start=time.now()
            password_reset(email, reset_link)
            #30 min before link expires
            while time.now()-start < 1800
                if reset_link.clicked:
                    user.update(password=get_password(reset_link))
                    return
                continue
            expire_link(reset_link)
def notify(user, target, title, message):
    if user.type == "CLUB" or user.type == "TEAM":
        for person in target.user_ids:
            #clubs can only send notifications to users
            send(person.user_id, title, message)
    elif user.type == "USER":
        #users can only send notifications to clubs (who host events)
        send(target.club_ID, title, message)
    elif  user.type == "ADMIN":
        #admins only send notifications to users (who may be a club admin)
        priority_send(target.user_id, title, message)
    else:
        raise Invalid_User_Error

def score_relevance(user, query, target):
    '''
  A. take the cosine similarity between the target's attributes and the user's
     preferences
  B. take the cosine similarity between the target's attributes and each of
     the user's subscribed clubs/events and sum them
  C. take the cosine similarity between the target's title+description and
     the query
    '''
  return sum(A,B,C)
```

```
def search(user, query, filters):
      #truncate all non-relevant results
      results = filter(system.all_events_and_clubs())
      for item in results:
            user.score_relevance(query, item)

def create_event(club_ID, title, description, photolink, tags, params=None):
      """
      creates an event with initial parameters and stores it in the database
      """

def delete_event(event_ID):
      '''
      scan database of users for subscribers and notify event is canceled
      scan database for event and delete the event_ID
      '''

def edit_event(event_ID, params):
      #create new event with same params as the old event
      #params argument is a hash retaining event params and unique key
      create_event(event_ID, params=params)
```

[5] Data Modeling will describe the data perspective using data and/or object models to explain the app's information architecture.

ERD:



[6] Non-Functional Requirements

Security:

- The application doesn't need any highly personal information. Only minimal information is required from the user, such as their preferences or created events.
    - The application is verified using a university email which prevents phishing and a DoS-approach in making fake clubs/events
    - User Types are separated into Admins, Users, and Clubs which have limited functionality
        - For instance, a User cannot modify club attributes unless they are a Club
    - *Moderator Admins maintain that the application will not be misused,* such as creating disorderly or mischievous events that disturbs the student or the school. However, it is also possible to only have the Event Admins allowed to create the events to prevent such cases.
- The login-screen functionality is nothing new. If the user forgets their password, a message will be sent to the corresponding email of the user's account.

        - Password reset links expire after 30 minutes to prevent phishing passwords
- A security system that is already available will be reused for this application (due to lack of experience on how to secure people's data).

Ease of Use:

- The user should have an easier time finding information with an easy-to-use user interface.
    - The buttons and their uses should be straightforward.
- The user-app experience will be highly personalized.
    - The application's strongest quirk is to give the user notifications for any wanted information, and also the power to create an event if they want to organize one.
        - The user can easily choose their preferences, which can be easily done using preference filters shown on the left side of the application.

Performance Optimization:

- There shouldn't be any problems with performance because all the application's functionalities are very basic.
- Unfortunately, the application is highly dependent on being connected online.
- Using python's DefaultDict data structure, we can decrease index accessing times to O(1)

Availability:

- While the application should be very accessible (and free) as long as you are a student, the problem arises with how to maintain this.
    - But doing it free would be a problem… (see **Maintainability**)
        - Having the app run without the ads would be the best course.
    - The application is only as big as its user base. A planned transition period would allow it to grow to a usable size

Maintainability:

- Maintenance
    - Should the app run with ads?
        - Yes! Having clubs sponsor the software to make their clubs appear at the head of a relevant query based on its parameters would fund maintenance.
    - If not, then how?
        - A possible solution would be to copy a previous approach to this problem.
            - Ex. There is IIT Blackboard, UIC Blackboard, CCC Blackboard, etc.
                - With this, each of the universities will do maintenance on their own.