Design Document

Team 5

Dhruvang Makadia (2013CS50289)

Sachin Meena (2013CS50296)

Sachin Kumar (2013CS50297)

Guided By

Prof S. C. Gupta

Contents

1. Introduction

1.1 Purpose

The College Connect is an internet based mobile application which will provide easy communication among college community. The purpose of this document is to as follows:

- Provide system overview using architectural design, defining the modules used in app and describing the tools used in design of application.
- Provide detail design with help of module APIs, explicit design of the database, screen layouts and use cases.
- Providing method for deployment of the web application using testing methods and configuration.

1.2 Scope

The College Connect application will allow individuals to communicate with others without sharing personal information like mobile numbers, to maintain privacy. To add to this, no messages will be stored by server permanently. Application will also feature a news feed where upcoming campus events will be displayed. If the app is not currently open then it will send out a notification for event/message.

This document describes the detailed design of the project. The mobile application architecture and implementation, database schema and all interfaces are explained in this document.

1.3 Overview

The document is divided into following sections:

Section 2 (**System Overview**): Describes the architecture of the application, the modules that need to be implemented and tools and technology used to build the app.

Section 3 (**Detailed Design**): Gives detailed description of APIs of the modules, the design of the database by describing the collections used to store data, the layouts of the various screens in the app and the use cases related to the app.

Section 4 (**Deployment Design**): This section provides details regarding the various test cases as per the use cases defined, and also the detailed method to configure the application.

2. System Overview

2.1 Architectural Design

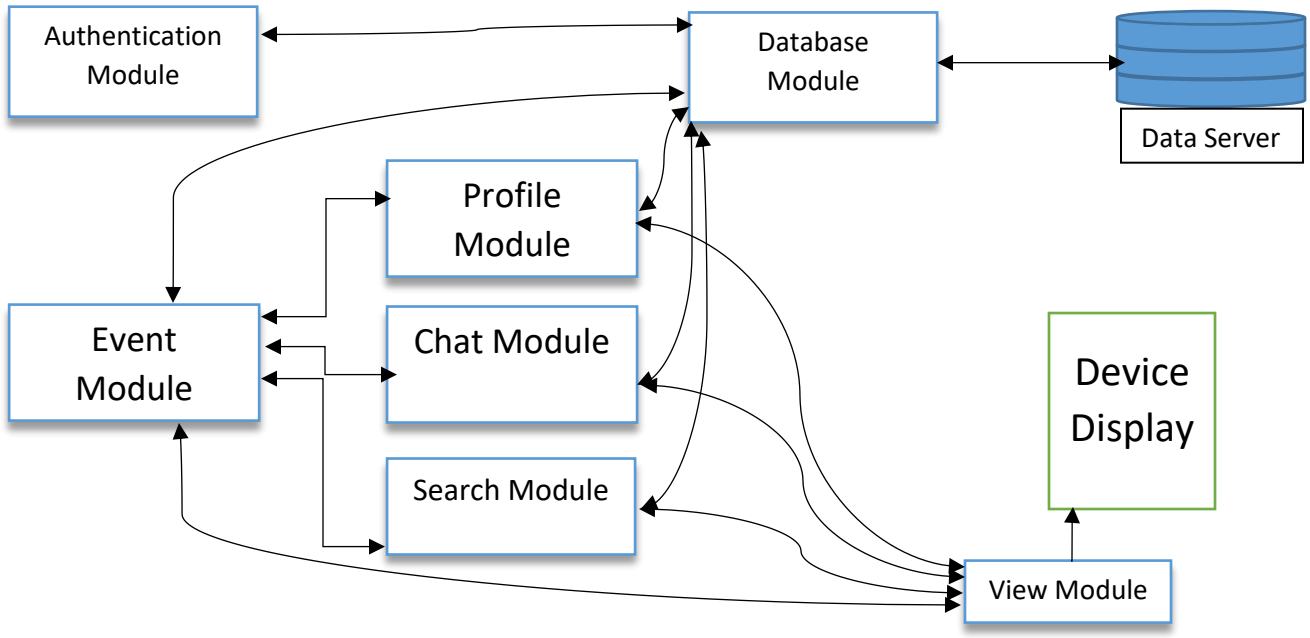

Figure 1: High level Architectural Design

2.2 Module Definition

Various modules in the application are:

1. **Authentication Module**: This module provides the following functionality:
   - While signing up, it checks the validity of form. e.g. it checks if username is not registered before, required entries are filled, etc.
   - While logging in, it checks if the correct username password combination is entered.
2. **Database Module**: This module provides create, insert, update and delete functionality for the manipulation of database.
3. **Event Module**: This module provides functions related to operations on events and news feed such as:
   - Adding a new event for public display
   - Notifying users about the new events
4. **Chat Module**: This module provides functions related to chatting feature of the application such as:

- Fetching the conversation data from storage
- Adding a user to block/mute list
- Sending a new message to any user

5. **Profile Module**: This module performs functions related to profile of the user such as:
    - Getting the profile data from database
    - Updating the profile of a particular user
6. **View Module**: This module performs all the viewing operation after gathering data from other modules.
    - Views the search results of events/profile/conversation
    - Views the profile data of any user
    - Views the news feed to all the user
    - Views various error toasts or general messages to user
7. **Search Module**: This module gathers results of all the search operations for the events, profile and conversations based on the given query.

2.3 Tools Used

We will be using Android Studio for development of our application. Coding will be done in JAVA language. We use node.js and express.js for server side scripting. For database we will use the MongoDB tool, and our smartphones for testing purposes.

3. Detailed Design

3.1 Module APIs

Here we present the module APIs that our application will have, divided up in various categories.

1. **Authentication Module**
    - register_user(data) – Validates the user data and checks the availability of the username. If successfully verified then adds the user to the database.
    - validate_login(username,password) – Checks if the password is correct for the username. If successfully verified then signs in the user.
    - logout() – Logouts the user from the application

2. **Database Module**
    - create_collection(name,structure) – Creates a new collection in the database
    - insert_data(data) – Insert data into the database
    - update_data(data) – Updates data into the database
    - delete_data(data) – Deletes data from the database

3. **Event Module**
    - get_feed() – Fetches the news feed from the database at server
    - check_new_feed() – Checks if there is a new event uploaded at the server
    - notify_feed(notification_msg) – Notifies the user about a new event
    - validate_event(data) – Validates the structure of the event. Checks if all the fields are properly filled up.
    - add_event(username,data) – Creates a new event in the database.

4. **Chat Module**
    - get_conversations() – Fetches the list of conversation for a user from local storage.
    - get_chat(username) – Fetches the chat messages in a conversation from local storage.
    - block_user(username,blockee) – Adds a user to the blocklist in the database
    - blocked_by(user1,user2) – Checks if user1 is blocked by user2.
    - mute_conversation(username,mutee) – Adds mutee to the mutelist of the username and blocks all the notification from conversation with mutee.
    - check_new_msg () – Checks if user received a new message.
    - notify_message(notification_msg) – Notifies the user about a new message.
    - send_msg(sender,receiver,msg) – Sends message from sender to receiver

5. **Profile Module**
    - get_profile(username) – Fetches the profile data for a user from the server.

- edit_profile(username,data) – Allows the user to modify their existing profile.

6. **View Module**
   - show_signup() – Shows the signup screen
   - show_login() – Shows the login screen
   - show_error(msg) – Shows an error message in the form of a toast
   - show_feed(feedlist) – Displays the feedlist on the screen
   - show_msg(msg) – Displays a message on the screen
   - show_events(event_list) – Displays the event list on the screen
   - show_conversations(conversations) – Shows the list of conversations on the conversation screen
   - show_chat(chat_messages) – Shows the chat messages for the specific conversation
   - show_profile(data) – Shows the profile data for a specific user
   - show_profile_list(profile_list) – Shows a list of profiles of some users
   - show_settings() – Displays the setting screen
   - show_about() – Displays the about screen

7. **Search Module**
   - search_event(query) – Searches for some event
   - search_conversation(query) - Searches for some conversation
   - search_profiles(query) – Searches for some users registered in the app

3.2 Database Design

We will be using MongoDB at the backend to maintain the database

Collection:

1. Login – this collection contains the data for validating userlogin. It contains list of username password pair.

```
Login {
    username: 'text',
    password: 'text'
}
```

2. Profiles – this collection contains the data specific to a particular user like his personal information, block lists.

```
Profiles {
    username: 'text',
    entryNo: 'text',
    degree: 'text',
    year: num,
    hostel: 'text',
    contact: num,
    photo: 'text',
    events: ['eventId1', 'eventId2'...],
    blockList: ['username1', 'username'...],
    muteList: ['username1', 'username'...]
}
```

3. Events – this collections contains the list of all the events. An event is composed of a unique id, uploader name and event information.

```
Events {
    eventId: 'text'
    uploader: 'text'
    date: 'date',
    time: 'time',
    venue: 'text',
    desc: 'text',
}
```

3.3 Screen Layouts

1. **Chat Screen**: This screen will display the list of previous conversations. If the user clicks on any conservation, it will open up to display the chat interface where messages can be sent and receive. The chat interface will have a drop down menu providing features like block, mute and view profile.
2. **Profile Screen**: This screen by default will show user his own profile. It will contain an edit button allowing user to edit their profile. There will be a search bar which will provide the search by name feature. There will be an advanced search button which will let the user search on the basis of department, hostel and year, redirecting them to their profile of the desired individual. The profile page will also have a message button to that user.

3. **Newsfeed Screen**: Upcoming events added by users are displayed on this screen.
4. **Recent Chats**: This screen shows friends of users that (s)he has recently had a chat with.

Figure 2: Sample chat screen

Figure 3: Sample Profile Screen

Figure 4: Sample Newsfeed

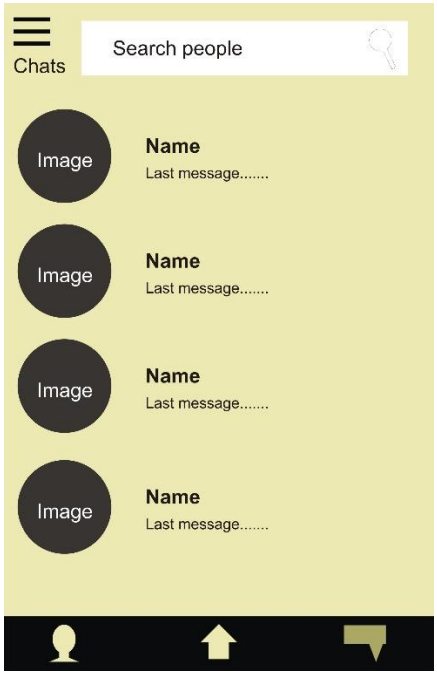

Figure 5: Sample Recent Chats

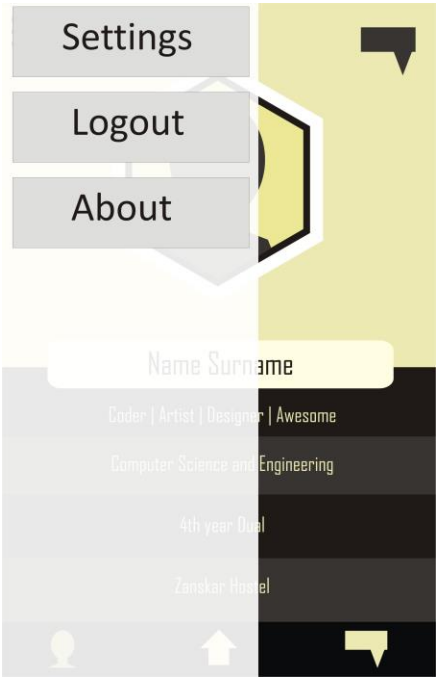

Figure 6: Sample Side Pane

3.4 Use Cases

1. **Signup**

```
register(data){
msg = register_user(data)
if (msg == “Success”)
        show_login()
else
        show_error(msg)
}
```

In the Signup use case, the entered data is validated (username not taken, valid email address, etc). If the entered data is valid then user is redirected to login page (with a successful registration message) and user data is entered in the database. If the entered data is invalid, user stays on the signup page and error is displayed.

2. **Login**

```
signIn(username,password){
        msg = validate_login(username,password)
        if (msg == “Success”){
                feedlist = get_feed()
                show_feed(feedlist)
        }
        else
                show_error(Invalid credentials msg)
}
```

In the login use case, the entered combination of username & password is sent to the database for verification. If the credentials are correct then user is redirected to homescreen (news feed screen). In case of discrepancy, an error for invalid credentials is displayed and user stays on the login screen.

3. **View Feed**

```
viewFeed(){
feedlist = get_feed()
if (feedlist == empty)
        show_msg(no feed message)
else
        show_feed(feedlist)
}
```

View feed involves fetching the feed data from the database and displaying it to the user. If there is no feed activity then a “no feed” message is shown.

4. **News Feed Notification**

```
notifyFeed(){
        if (check_new_feed ())
                notify_feed(notification_msg)
}
```

The news feed notification will run as a service and constantly check if there is any new feed to notify about.

5. Event Search

```
searchEvent(query){
        event_list = search_event(query)
        if (event_list == empty)
                show_msg(no result message)
        else
                show_events(event_list)
}
```

To provide the event search, we fetch the event data matching the query from the database. In case of no match, a message “no events found” will be displayed.

6. Add Event

```
addEvent(username,data){
        msg = validate_event(data)
        if (msg == “Success”)
                add_event(username,data)
        else
                show_error(msg)
}
```

To add an event for public display, a user provides the event information which is then validated. If there is no discrepancy in the filled data (no required field empty, etc), then event is added to the database. Else an error message is displayed.

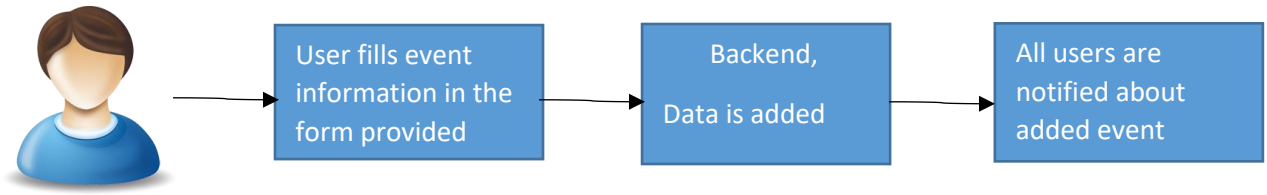

Figure 7: Flow diagram for new event upload

7. **View Conversations**

```
viewConversations(){
        conversations = get_conversations()
        if (conversations == empty)
                show_msg(no conversation message)
        else
                show_conversations(conversations)
}
```

For view conversations use case, the data is fetched from the local storage and all the conversation headings are displayed.

8. **Conversation search**

```
searchConversation(query){
        conversations = search_conversation(query)
        if(conversations = empty)
                show_msg(no conversation msg)
        else
                show_conversations(conversations)
}
```

For search conversation use case, the user enters the search query. Data is fetched and sorted from the local storage and then displayed on the screen.

9. **View Chat**

```
viewChat(username){
        chat_messages = get_chat(username)
        show_chat(chat_messages)
}
```

For viewing a particular chat, the chat with the concerned individual is fetched from local storage and displayed on the screen.

10. Send Message

```
sendMessage(receiver,sender,msg){
        if (blocked_by(sender,receiver) or blocked_by(receiver,sender))
                show_error(block msg)
        else
                send_message(receiver,sender,msg)
}
```

In the Send Message use case, the user mentions the receiver and the message to be sent. If the sender hasn't blocked the receiver or vice versa the message is sent to the server which is then relayed to the receiver.

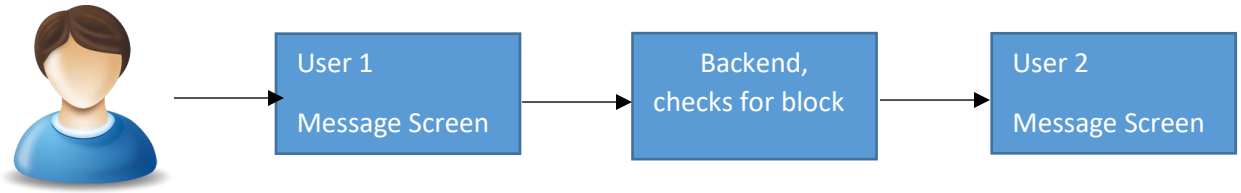

Figure 8: Flow diagram for messaging

11. Block User

```
blockUser(username,blockee){
        block_user(username,blockee)
}
```

For Block User use case a direct API call is provided which will add the information to the database.

12. Mute Conversation

```
muteConversation(username,   ute){
        mute_conversation(username,   ute)
}
```

For Mute Conversation use case a direct API call is provided which will add the information to the database.

13. Message Notification

```
notifyMessage(){
        if (check_new_msg ())
```

```
            notify_message(notification_msg)
    }
```

The message notification will run as a service and constantly check if there is any new message to notify about.

14. View Profile

```
    viewProfile(username){
        data  = get_profile(username)
        show_profile(data)
    }
```

In the view profile use case, the user enters the username of the individual whose profile needs to be displayed. The profile data is then fetched from the database and displayed to the user.

15. Profile Search

```
    searchProfile(query){
        profile_list = search_profiles(query)
        if(profile_list = empty)
            show_msg(no profile msg)
        else
            show_profile_list(profile_list)
    }
```

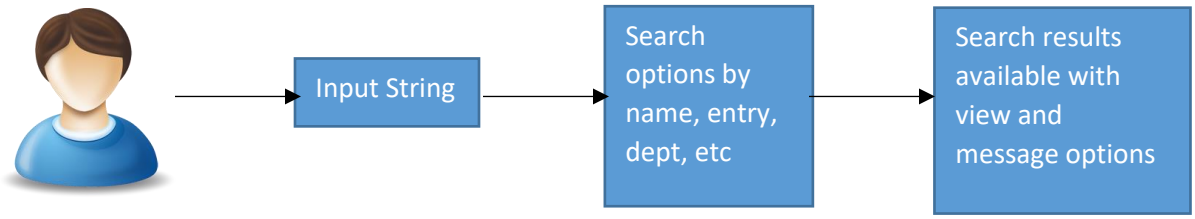

Figure 9: Flow diagram for profile search

For profile search use case, the list of all profiles who match the query is fetched from the database. If the list is non empty, then it is displayed to the user else the “no profile match” message is displayed.

16. Navigation

```
navigation(choice){
        if (choice == “Settings”){
                show_settings()
        }else if (choice == “Logout”){
                logout()
                show_login()
        }else if (choice == “About”){
                show_about()
        }else {
                show_error(“unknown input”)
        }
}
```

The navigation panel offers 3 choices: Settings, Logout and About. Different action is taken corresponding to each of the choices.

4 Deployment Design

The application will be available for download from a server that is publicly accessible.

4.1 Testing

We plan on testing the systems/modules on the lines of testing done in Extreme Programming (XP) approach. That is, we will first draw up some test cases for the module to be built, then we will build the model, integrate it into the system, and then run test cases to see if the module works correctly by itself and that it has not introduced any new bug in the overall application.

Test cases to be passed by each module are given below:

A. Testing authentication module:

- Registration: Information filled by user should reach database module. If all input constraints are satisfied, then user is logged in. Else failure alert is passed to view module and user is asked to try registration again.

- Login: User credentials should be passed on to database module after checking for empty of invalid entry in the provided fields.

B. Testing Event Module:

- Event Creation: Virtually simulate multiple users to fill up the event creation form. Information filled by user should reach database module. If all input constraints are satisfied, then an event is created in the data server corresponding to each user entry. In case of successful uploads, database module indirectly updates the view module, leading to each user's newsfeed getting populated with the newly added events. Else failure alert is passed to view module and user is asked to try again.

C. Testing Profile Module:

- Edit Profile: Information modified by user should reach database module. If all input constraints are satisfied, then profile is updated in the data server. Else failure alert is passed to view module and user is asked to try again.

D. Testing Search Module:

- Search by name: Search keywords are checked with names of students available in the database and matching students are returned as result of the query.
- Search by entry number: Search keywords are checked with entry number of students available in the database and matching students are returned as result of the query.
- Search by department: Search keywords are checked with department names of students available in the database and matching students are returned as result of the query.
- Search by hostel: Search keywords are checked with hostel names of students available in the database and matching students are returned as result of the query.

E. Testing Chat Module:

- Sending messages: Virtually simulate multiple users and make them communicate pairwise.

- Blocking a user: Make a user block another user. Then try sending messages from the blocked user to the one blocking. Try a search for the blocking user from blocked user's account.

F. Testing Database Module:

- Updating server: Try inserts, updates, deletes into the data server.
- Fetching data: Try fetching instances of all types of data stored on server for various users.

G. Testing View Module:

- Checking that all the screens such as chat screen, newsfeed screen, profile screen and other screens are displayed on the device screen correctly.

----------End of Design Document----------