MeetUp!

MeetUp! will be a social networking app designed specifically for college students. It will help students meet other students based on their class schedules and on their typical walking paths. It will allow students to communicate with other students using the app, enriching the college experience by connecting students with similar interests. The app will be customized for each college campus.

To use MeetUp!, a student will download the app to their smart phone and complete the registration process to create an account; only currently registered students will be allowed to have accounts. The student will either enter his/her schedule manually or will allow that schedule to be downloaded from a university server. The map interface will display a campus map. The student will be able to enter points of interest on the map, including dorm location, parking location, bus stop location (depending on how the student gets to campus) – one of these must be entered – plus other campus destinations, such as the cafeteria, student union, stadium, or gym. The app will then generate walking directions and, depending on the campus, bus route directions for the student to get to the classes on his/her schedule.

Based on the student's schedule and on the walking directions, the app will make suggestions for friend requests. Suggestions will be prioritized based on the amount of overlap between schedules and between walking paths and bus directions. These suggestions will be sent to the student, who can then decide whether to try to make the connection, delete the suggestion, or defer a decision. Both students must agree to a connection before they become friends.

Once a connection is made, a student will be able to send text messages to his/her contacts. Other types of message types will also be allowed: emojis, gifs, jpegs, other multimedia content, and live web links. Given the possibility of abuse, a student may request that a contact be blocked, removing that person from the student's list of contacts; the reason for the request is optional. A student may also contact a system administrator at any time, for example to report problems with the app or with suggestions for improvements.

There will be a system administrator role with the following capabilities. A variety of system statistics will be available, such as the number of students using the system, the number of connections suggested, the number of connections made, the number of connections deleted, the number of connections deferred, the number of messages sent; both numeric and graphical displays of each will be show. A dashboard will be available to show live statistics of the current system usage. All block requests will be shown. Any serious reasons for block requests will be flagged for immediate action, such as threats of harm or sexual content. The admin will be able to block student accounts for these and other reasons.

The system should be able to scale up to handle 10 million users. The system will be used in many countries around the world. Performance is important; each user request should be handled quickly, but actual times will be specified later. The user interface should be easy to use: features should be easy for users to find and use, and users should easily comprehend what to do next. User data should be secure: no outsiders should be able to access user data, and users should not be able to access other users' data without permission. The system should be available 99.99% of the time. User actions should be logged. New features should be easy to add to the system.

Tasks:

1. Create a set of user stories from the specifications. Clear up any fuzzy requirements.

2. Create a set of nonfunctional requirements (also known as Quality Attributes) from the specifications and classify them by type. Make any generic requirements more specific.

3. Design a software architecture using one of the following architecture patterns. Draw the diagram, describe each of the modules, and tell how the modules interact.

Your group will use one of these patterns: microservice, monolithic, service-oriented, model-view-controller.

4. Trace two or three user stories through the modules.

5. Create the code for one or two modules.

6. Create unit test cases and integration tests for the modules in #5.

7. Create a set of acceptance test cases for the overall system.