List of members:
Evan Dong
Nirujan Velvarathan
Bridget Green
Xuan Zheng

Overview of Project
The project we are working on will be a student helper app. The app aims to help students with their learning by giving them access to helpful features that are not on the school's official app or otherwise easily found elsewhere. Useful features that exist elsewhere will also be included. These features include a schedule maker, SAS helper, friends list, and a campus map. Details on these features will be explained further in the "List of Features" section.

As a starting point for the project, the team will focus on features that will specifically help students of Ontario Tech University, and will branch out to cover different Universities and Colleges if possible.

List of Group Members and Responsibilities

Responsibilities	Names
Login Page (Front-end) ● Use school sign in? ● Can not sign in / use guest mode (limited features)	Nirujan Velvarathan Bridget Green
Login (Back-end) ● If using school login, need to access school login and have option to save login so student doesn't have to login each time (i.e. "remember me" button) ● If not, make our own account storage and stuff	Bridget Green
Main Menu page + appbar, sidebars, etc. ● Coding the pages and what goes where ● Making the buttons work	Nirujan Velvarathan Bridget Green
Settings page ● Change notifications ● Change preferences ○ light/dark theme ● Logout/login	Nirujan Velvarathan Bridget Green
Miscellaneous / Small Helpful Features (mostly buttons on the main menu) ● Button that directs to email - check email and email to professors and other students ● Button that links to "Canvas Student" app ● Button that links to "OntarioTechMobile" app ● Button that directs to university website (homepage) ○ Either in app or opens up the default browser for the phone	Nirujan Velvarathan Bridget Green Evan Dong
Schedule Maker pages (Front-end) ● "Make schedule" / "add courses" page ○ Have a display/search for available courses page ● Display possible schedules page ○ Zoom in on schedule (could be part of same page instead) ● View weekly schedule (can click on a saved schedule to view a more 'zoomed' in view of the classes for each week)	Evan Dong Xuan Zheng
View program map for required courses page (Front-end + backend)	Evan Dong

• Maybe just have a link to the program maps website	
Schedule Maker code (Back-end) • Making all the possible combinations of schedules • Fetch data (class times, name, prof?) about the courses from school api. This way, they don't have to go check the add/drop course website and switch between that and our app. • Display saved schedules	Evan Dong Xuan Zheng

SAS helper pages (Front-end)
• "Main page" with buttons to different uses
• Login / switch account page (this would just be borrowed from login)
• Update accommodations page
  ○ Appears in the form of an Add button
• Edit accommodations
  ○ User can edit information on their accommodations
  ○ User can remove an accommodation if needed
  ○ This button would be disabled if the user has not entered any accommodations
• Accommodations list page
  ○ What accommodations they have
  ○ What do the accommodations give them for specific tests/assignments
• "Upcoming Assessment" Button
  ○ This would lead to either a pop up or another page
  ○ In a search bar, user would write "Test", "Presentation", etc.
  ○ This would bring up a List of what accommodations the user would be entitled to (double time, extra space, a scribe, etc.)
• "Renew Accommodations" Button/"Accommodation Letters" Button
  ○ These buttons would act as an easy link for the user to access whatever their university allows them to renew their accommodations for the semester, and where to send off the accommodation letters for their professors
  ○ This could be accomplished by set up by the user themselves (they enter in the links themselves)
  ○ If there is no links provided, these buttons would be disabled
	Bridget Green

SAS helper page (Back-end) ● Fetching data on accommodations, reorganizing it and displaying the info in a neat way ● Link to school website on SAS ● Email function to professor(s) ○ Only if required. For instance, UOIT will send these letters to the professors instead of the student themselves, provided that the student requests this to be done	Bridget Green Evan Dong
Friends list (Front-end) ● View friends page ● Add friends page ● View friend profile (when clicking into a friend) ○ Displays some of their info ● Friend chat page (click on friend to open up their chat)	Nirujan Velvarathan Xuan Zheng
Friends list (Back-end) ● Allow people to add each other using names/student id ● Allow friends to view other friends info ● Send messages through chat	Xuan Zheng Evan Dong
Campus map ● Getting map info ● Display important things / relevant things for students ○ Maybe display capacity of places like the library or rooms that students can book	Bridget Green Evan Dong

List of Features and how Functional Requirements will be used

Schedule Maker: this feature will aid students in picking their courses and finding their preferred schedule among the class times given for the courses.
To start, the user will click a button (i.e. “Make Schedule”) to bring them to a new page. On this page, they will be prompted for the number of courses they want in the schedule and to enter their first-course name and the class meeting time associated with that course. There will also be the option to click a button to allow them to add any dependency class times (i.e. labs and tutorial) and to add multiple class meeting times for that course (if the course has more than one class). After filling out the first course, the user can then click a “+” button, which will bring out the fields for them to enter the details for their second course. This is repeated until they enter all the courses they wish to register in, upon which they can click the “Finish” button (e.g. “Generate Schedules”).
The app will take the inputs and then show the user a list of all the possible *n* course schedules that can be made with the courses given (*n* being the number of courses the user wants in their schedule specified at the start). Each schedule in this list is shown using a table - a Sun-Sat (horizontal) and time scale based on a min/max (vertical) table with the classes highlighted with their respective times. Ideally, an option will allow the user to filter the schedules for specific courses they want (i.e., only showing schedules with “course 1”). Users will have the option to save the schedules they like so that they can reference them later; the saved schedules will appear on their own page that can be accessed from the main schedule maker page.
Users can set a saved schedule as their primary schedule, which will be displayed on the main schedule maker page. This would allow users access to the weekly schedule page which just shows the classes that the student has for each day of the week (essentially just a more ‘zoomed’ schedule).

SAS Helper: this feature will help students with accommodations by giving them a more detailed and specific list of their accommodations and how they are applied for an upcoming assignment. This would ideally also make it easier for the student to understand their accommodations and how to ensure that they are met during the semester.
The user will click a button (i.e. “What are my Accommodations”). If this is their first time (or if they haven’t used the feature), they will have the option to enter their student details (e.g. login through school or something similar and secure), after which the app will fetch the accommodations available to them. If they’ve already entered their student details, the app should save the accommodations available to them and display that information right away. Alternatively, the user could manually input their accommodations themselves. This would be done via an Add button, which would

provide the user with TextFields to input any information. The user will also have the option to “sign out” and re-enter their student details for accommodations again.
In addition, there will be a button that will link the user to the university’s website for student accommodations so students can easily check their accommodations officially.
The user can make edits to the information in their accommodations or even remove one manually via an Edit button. This would give the user a chance to select what to edit or remove. This button will be disabled if no accommodations are available.
Having the user manage their accommodations manually may seem like extra work. That being said, there could potentially be issues ethically of an app gathering that much personal data on someone by itself. While just getting the information from the user’s login info would be easier, the alternative option may be preferable for the sake of user privacy. Also, there could be issues surrounding accessing Accessibility’s database, but at the time of writing, this can not be confirmed or denied.
When the accommodation details are displayed, the user can click a button to see what their accommodations give them for upcoming assignments. This is done via either a pop-up (preferably) or a redirect to another page. This would contain a search bar function, potentially with predictive text. The user would be able to type in what type of assessment they have coming up, and a list of accommodations they’re entitled to for this assessment would be displayed (such as double time, extra space, a scribe, needing a quiet and isolated environment, etc.). This would be done preferably in a ListView or at least a Column.
Finally, there will be a button provided for renewing accommodations and another for sending off accommodation letters to professors. If the user has entered what university they attend, the app will find links to the relevant information. Alternatively, the user could input this information themselves. Using Ontario Tech as an example, an Ontario Tech student would use these buttons as a quick and easy way to renew their accommodations for the upcoming semester and send off the accommodations letters. This would allow the student to do this without having to go through the entire process of navigating the student portal and then navigating the accessibility services homepage for what they need (which, from the experience of one of the group members, is a frustrating hassle. Especially as Ontario Tech’s site doesn’t make it easy for new students to figure out how to do any of this)
Of course, the idea that a student could add in their accommodations and whatnot manually does assume that they are aware of what accommodations they even have.
If there is time before the due date, an information option will be added in the form of a miniature (?) icon. When clicked, a pop-up will give the user information about what accommodations are, where to find yours, how to register for them, etc.

Friends List: After students log in, this feature allows students to search for their friends and add them using their friends' student numbers or friend codes. Added

friends will have their names and status on the "Friend List" Page. The user can click on a friend to enter their profile, which will display more information about their friend - entered by the user or their friend. The friends list will also link to a chat interface, allowing the app's friends to communicate. They will be allowed to send the usual - text, images, videos, etc. Users can also manage friendships by removing, muting, or blocking friends.

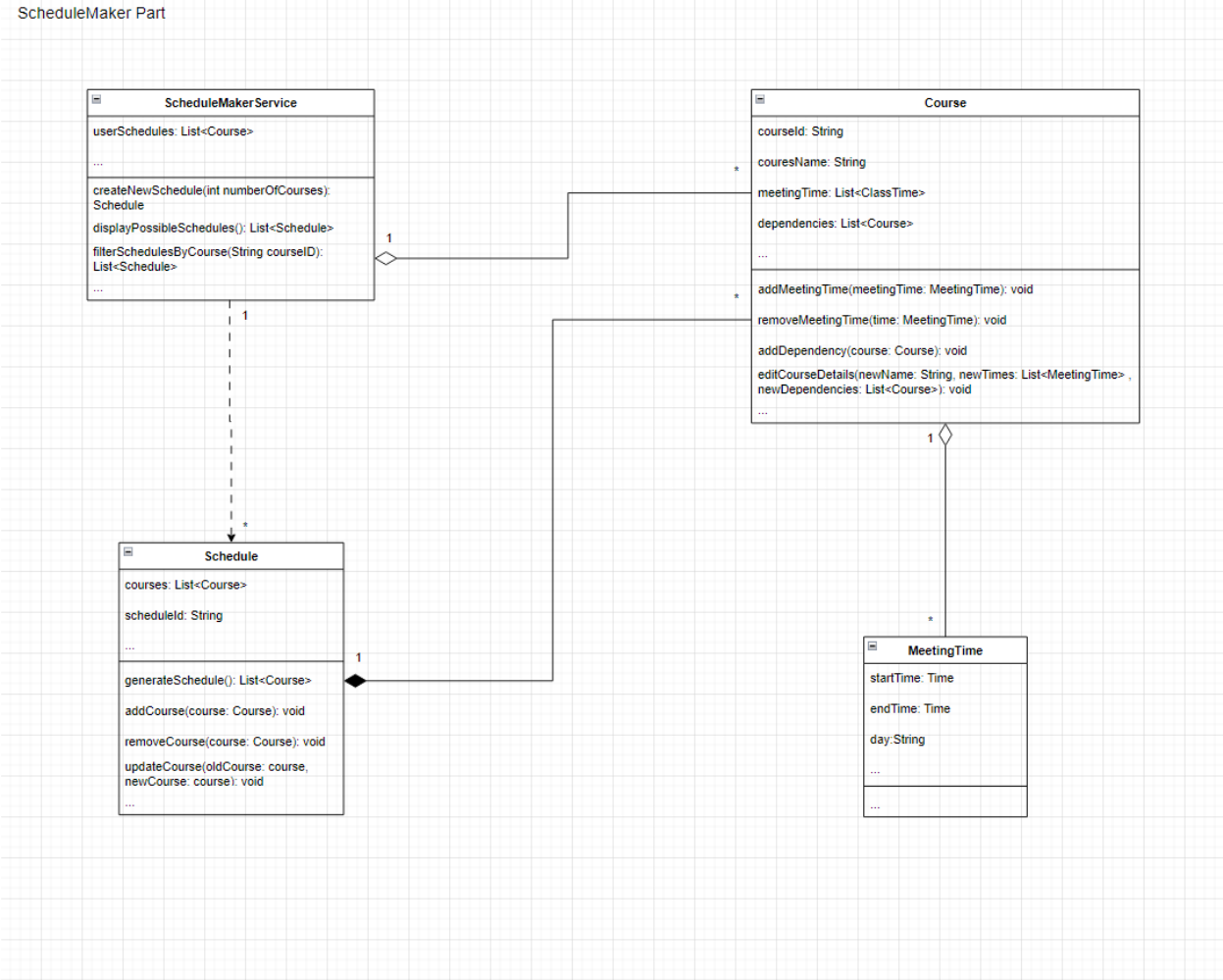
Campus Map: This feature allows users to select a location on the shown campus map or enter the building's name into the search box to get a route and navigation. When entering this page first, it will ask for the user's location permissions. If permissions are allowed, the app will display the user's location on the map and use it as the starting point for navigation. If permissions are denied, the user can still use the other features of the map, and they would have to type in the starting point for navigation use manually. This feature will most likely use outside apps/services (like Google Maps) for its implementation.

Functional Requirements (more exact list of where each requirement is)**:**

- Dialogs and pickers
    - Dialogs will be used for warnings (such as not filling in all the required fields for the schedule maker) and other potential information that needs the users' immediate attention
    - Pickers will be used to select the number of courses wanted and time of classes for schedule maker feature
- Multiple screens and navigation
    - The different pages for each of the features and the buttons that bring the user through them
- Snack bars
    - Appears after the user has completed a request
        - User has added class
        - User has added friend (sent friend request)
        - Accommodation has been successfully requested
- Notifications
    - Notifications will be sent for chat messages from friends and when receiving emails
- Local storage (SQLite)
    - Store login token (if allowed by user) so that they don't have to log in each time the app is opened
    - Save app settings
        - light/dark theme, notifications, etc

- Cloud storage (Firestore or other)
    - Save information that is tied to the user's account
        - Friends list
        - If user has logged in, save their schedules and accommodation information
- HTTP requests
    - Fetch data from school website for course information
        - Used as a part of the schedule maker feature so that students can easily view what courses are available for a given term and its information (class times, professor, etc.)
    - Fetch data on accommodations
    - Get map data (from google maps?)
    - Fetch data from websites to display in app
        - This may be scraped and we instead just redirect the user by opening up the website in their default browser

Code Design



Explanation:

1. ScheduleMakerService:
This service layer class is an intermediary between the user interface and the underlying data or logic.

Attributes:

userSchedules: A list of courses the user has or plans to add. Each course is an instance of the "Course" class.

Methods:

createNewSchedule(int numberOfCourses): Creates a new schedule based on the number of courses the user wants to take.
displayPossibleSchedules(): Returns a list of possible schedules for the user.
filterSchedulesByCourse(String courseID): Returns a filtered list of schedules that include a specific course.

2. Schedule:

This represents an individual schedule containing a set of courses.

Attributes:

courses: A list of courses that are a part of this schedule.

scheduleId: A unique identifier for the schedule.

Methods:

generateSchedule(): Generates or refreshes the list of courses for the schedule.

addCourse(Course course): Adds a course to the schedule.

removeCourse(Course course): Removes a course from the schedule.

updateCourse(Course oldCourse, Course newCourse): Updates a course's details within the schedule.

3. Course:

This represents an individual course offering.

Attributes:

courseId: A unique identifier for the course.

courseName: The name of the course.

meetingTime: A list of class meeting times.

dependencies: A list of related courses, like labs or tutorials, that depend on this course.

Methods:

addMeetingTime(MeetingTime meetingTime): Adds a meeting time to the course.

removeMeetingTime(MeetingTime time): Removes a specified meeting time from the course.

addDependency(Course course): Adds a dependent course (like a lab or tutorial) to this course.

editCourseDetails(...): Allows editing of course details like its name, meeting times, and dependencies.

4. MeetingTime:

This represents the times at which the classes for a course are held.

Attributes:

startTime: The start time of the class.

endTime: The end time of the class.

dayString: The day of the week when the class is held.

Relationships:

The 1-to-many relationship between ScheduleMakerService and Schedule indicates that one ScheduleMakerService can manage multiple Schedules.

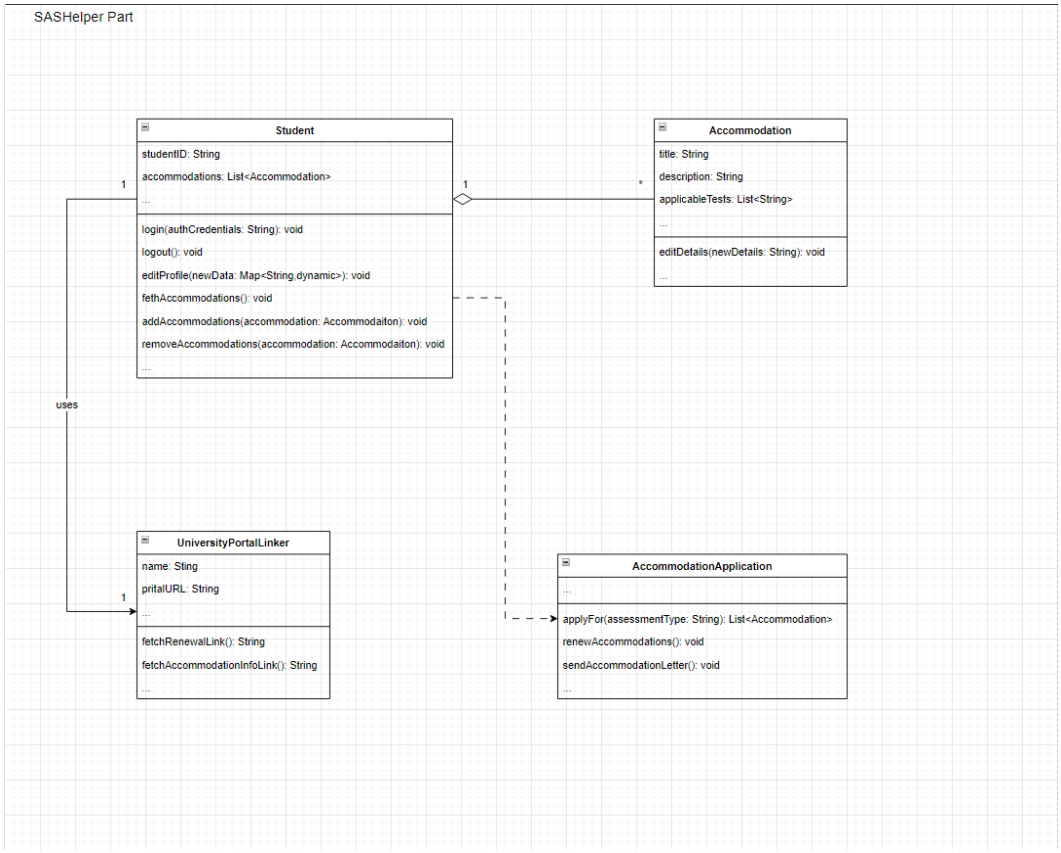
The 1-to-many relationship between Schedule and Course indicates that one Schedule can have multiple Courses.

The 1-to-many relationship between the Course and MeetingTime shows that a Course can have multiple MeetingTimes (e.g., a course might meet numerous times a week).

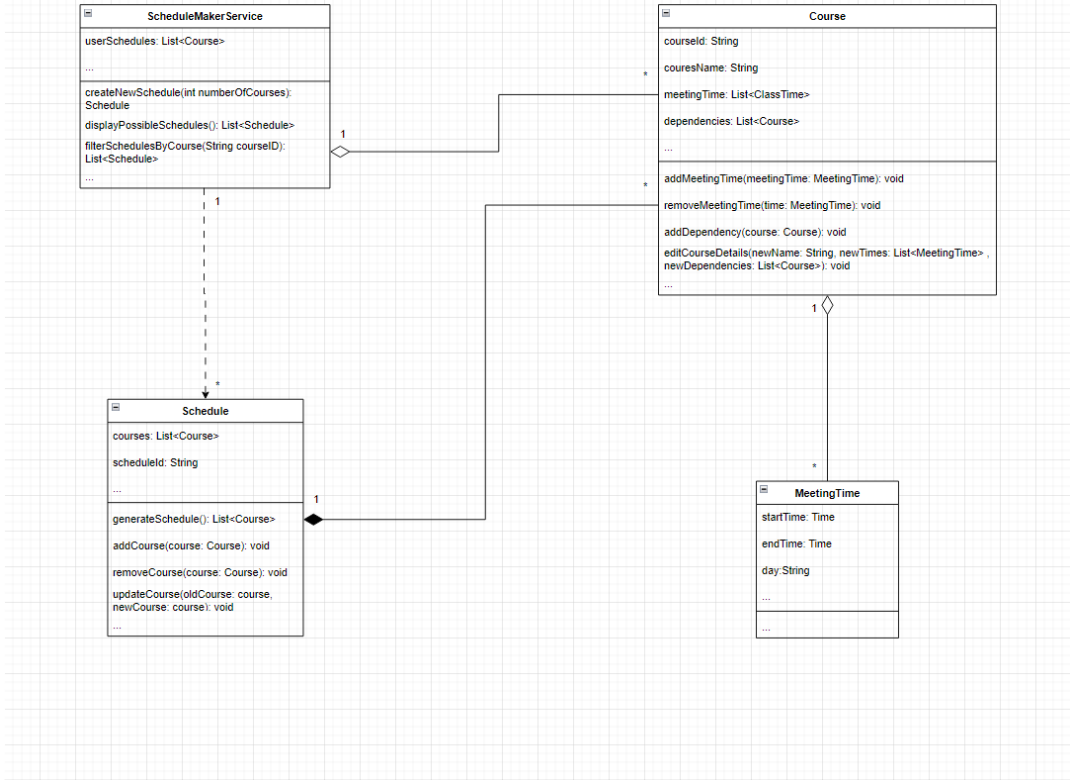
Similarly, under the dependencies attribute, the 1-to-many relationship between the Course and itself implies that a Course can have multiple dependent courses (like labs or tutorials).

In summary, this UML class diagram illustrates the structure and interactions of the primary classes within a "ScheduleMaker" system. It lays out how courses, meeting times, and schedules relate to one another and how the system can manage them.

SASHelper Part



ScheduleMaker Part



Explanation:

1. Student:
This class represents individual students in the system.
    Attributes:
        studentID: A unique identifier for the student.
        accommodations: A list of accommodations that have been granted or are associated with the student.
    Methods:
        login(authCredentials: String): The student can log into the system using authentication credentials.
        logout(): Allows the student to log out of the system.
        editProfile(newData: Map<String, dynamic>): Enables students to update their profile details with the provided data.
        fetchAccommodations(): Fetches the list of accommodations associated with the student.
        addAccommodations(accommodation: Accommodation): Adds a specific accommodation to the student's list.
        removeAccommodations(accommodation: Accommodation): Removes a specified accommodation from the student's list.
2. Accommodation:
This class represents specific accommodations provided to students.
    Attributes:
        title: The name or title of the accommodation.
        description: A brief description of the accommodation.
        applicableTests: A list of tests or exams for which the accommodation is applicable.
    Methods:
        editDetails(newDetails: String): Allows updating the accommodation details.
3. UniversityPortalLinker:
This class interfaces the SASHelper system and an external university portal.
    Attributes:
        name: The name of the university or institution.
        portalURL: The URL or link to the university's portal.
    Methods:
        fetchRenewalLink(): Fetches a link from the university portal for renewing accommodations.
        fetchAccommodationInfoLink(): Retrieves a link from the university portal containing information about accommodations.

4. AccommodationApplication:

This class deals with applying for accommodations.

Methods:

applyFor(assessmentType: String): Allows a student to apply for accommodations for a specific type of assessment. It returns a list of accommodations that apply to the given assessment type.
renewAccommodations(): Allows a student to renew their existing accommodations.
sendAccommodationLetter(): This method sends a formal letter or notification regarding accommodations.

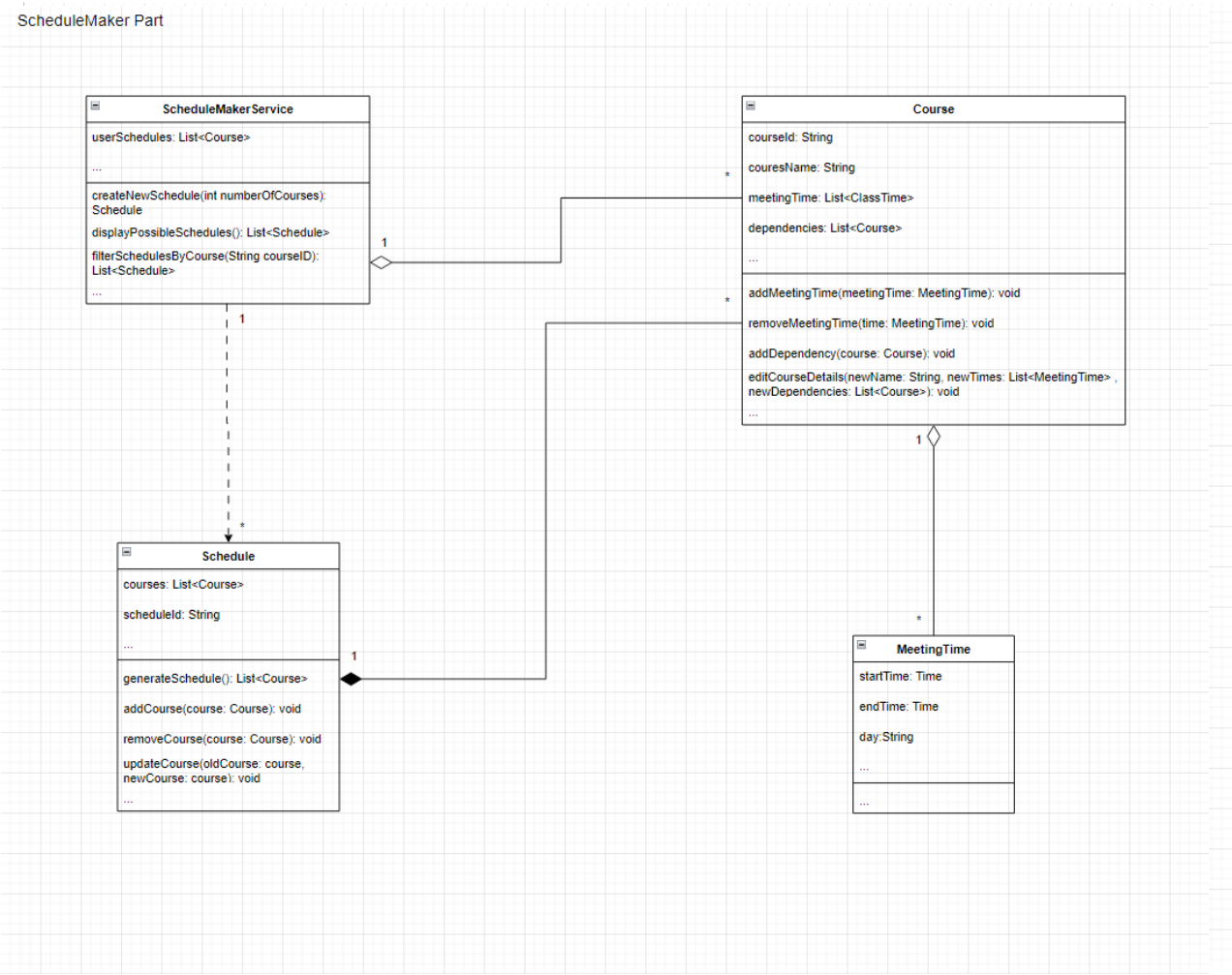
Relationships:

The 1-to-many relationship between Student and Accommodation implies that one student can have multiple accommodations.

The "uses" relationship between Student and UniversityPortalLinker indicates that the Student class utilizes the services or methods provided by the UniversityPortalLinker class.

The dashed arrow from AccommodationApplication to Accommodation with the label applyFor suggests that the applyFor method in AccommodationApplication returns instances of the Accommodation class.

In essence, this UML class diagram illustrates how students interact with accommodations, how those accommodations are described, and how the system interfaces with an external university portal. It provides a high-level view of the key components of the "SASHelper" system and their relationships.



Explanation:

1. Friend:

This class represents a friend or contact in the system.

Attributes:

studentNumber: A unique identifier for the friend, likely their student ID.

name: The friend's name.

chatHistory: A reference to chat interactions/history with this friend.

Methods:

sendMessage(message: Message): A user can send a message to this friend.

2. UserProfile:

This class represents the profile of a user in the system.

Attributes:

studentNumber: A unique identifier for the user, likely their student ID.

name: The user's name.

status: The current status of the user (e.g., online, offline, busy).
friendList: A list containing references to the user's friends.

Methods:

updateStatus(newStatus: String): Allows users to update their current status.
searchFriend(studentNumber: String): The user can search for a friend using their student number.
addFriend(student: Student): Allows the user to add another student as a friend.
removeFriend(friend: Friend): The user can remove a friend from their friend list.

3. ChatSession:

This class represents an active or historical chat session between users.

Attributes:

message: A list containing the messages exchanged in this chat session.

Methods:

sendMessage(message: Message): Sends a message within the chat session.
retrieveHistory(): Returns a list of messages in this chat session.

4. Message:

This class represents individual chat messages.

Attributes:

messageId: A unique identifier for the message.
content: The text content of the message.
timestamp: The date and time when the message was sent.
sender: Reference to the student/user who sent the message.

Methods:

delete(): Allows the deletion of this message.

5. ChatService:

This class encapsulates various chat functionalities.

Attributes:

currentChat: A reference to the currently active chat session.

Methods:

openChatWith(friend: Friend): Opens a chat session with the specified friend.
sendChatMessage(content: String): Sends a message within the current chat session.
displayChatHistory(chat: Chat): Displays the history of messages in the specified chat session.

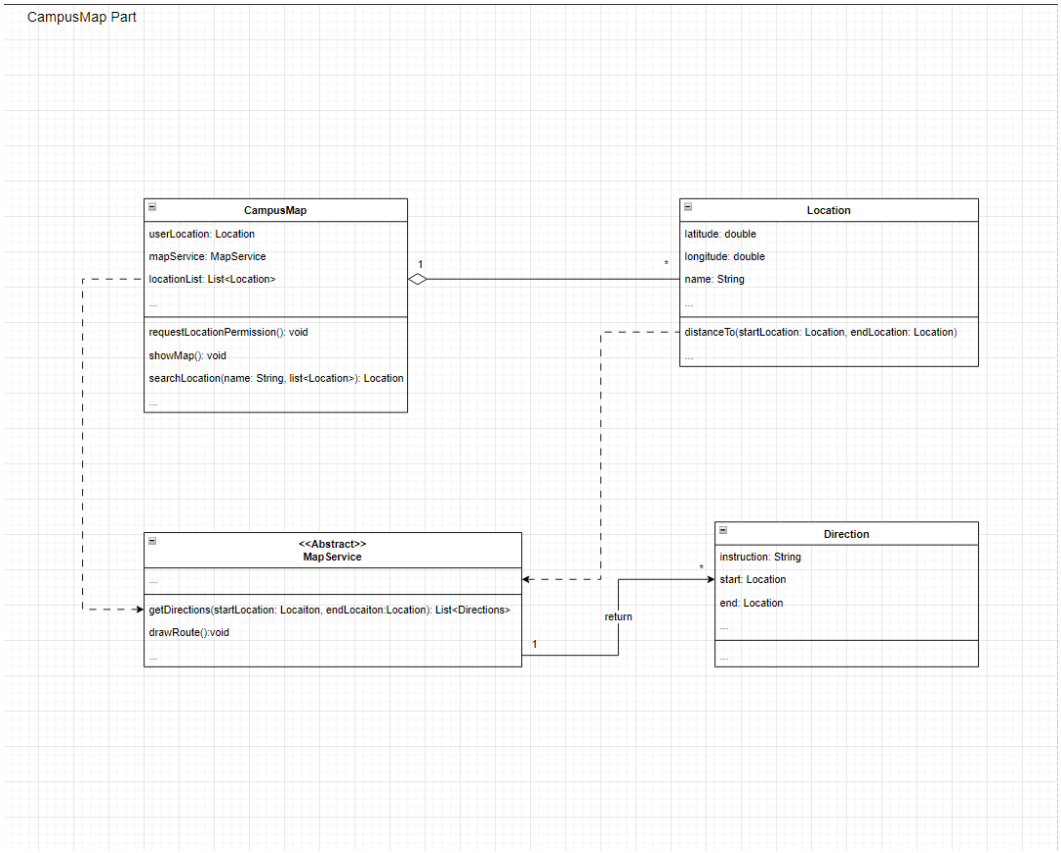
Relationships:

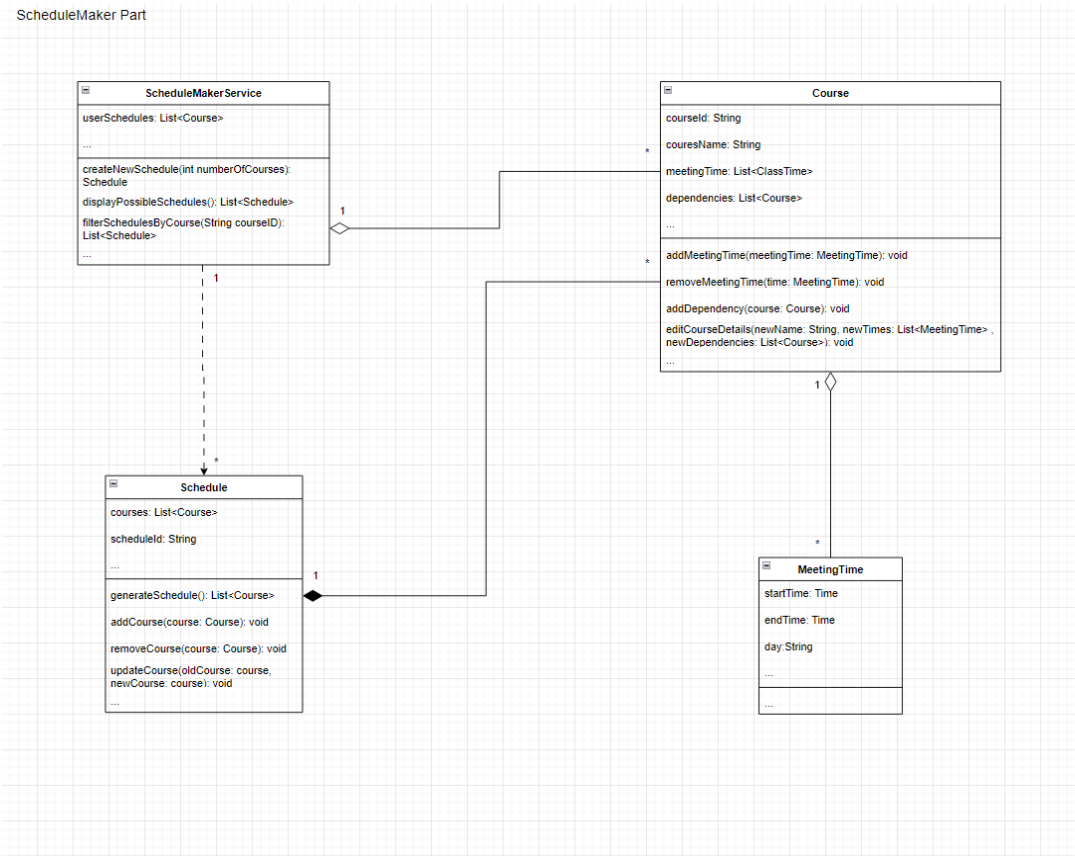
The 1-to-many relationship between UserProfile and Friend implies that one user profile can have multiple friends.

The 1-to-many relationship between ChatSession and Message indicates that a chat session can contain multiple messages.

The 1-to-many relationship between ChatService and ChatSession suggests that the chat service can manage multiple chat sessions, but only one session is active at any time.

In summary, this UML class diagram illustrates the architecture of a friend list and chat system, showing how users can manage their profiles, interact with friends, and exchange messages in chat sessions. It provides a high-level view of the primary components and their relationships in the "FriendList" system.





Explanation:

1. CampusMap:
This class represents the main interface or component that users will interact with for map-related functionalities on the campus.

Attributes:

userLocation: Represents the current location of the user.
mapService: A reference to the map service that provides various functionalities.
locationList: A list containing references to various locations on the campus.

Methods:

requestLocationPermission(): Asks the user for permission to access their current location.
showMap(): Displays the map to the user.
searchLocation(name: String, list<Location>): Allows the user to search for a specific location on the campus using its name.

2. Location:
This class represents individual locations on the campus.

Attributes:

latitude: The latitude coordinate of the location.
longitude: The longitude coordinate of the location.
name: The name of the location.

Methods:

distanceTo(startLocation: Location, endLocation: Location): Calculates the distance between two given locations.

3. MapService (Abstract):

This is an abstract class, meaning it provides a blueprint for other classes and cannot be instantiated directly. It is designed to provide map-related services.

Methods:

getDirections(startLocation: Location, endLocation: Location): Provides directions between two given locations. Returns a list of Direction objects.
drawRoute(): Displays or draws the route on the map.

4. Direction:

This class represents directions for navigating from one location to another.

Attributes:

instruction: Textual direction or instruction for navigation.
start: The starting location of this direction.
end: The ending location of this direction.

Relationships:

The 1-to-1 relationship between CampusMap and MapService implies that each CampusMap is associated with one MapService.

The 1-to-many relationship between CampusMap and Location indicates that a campus map can reference multiple locations.

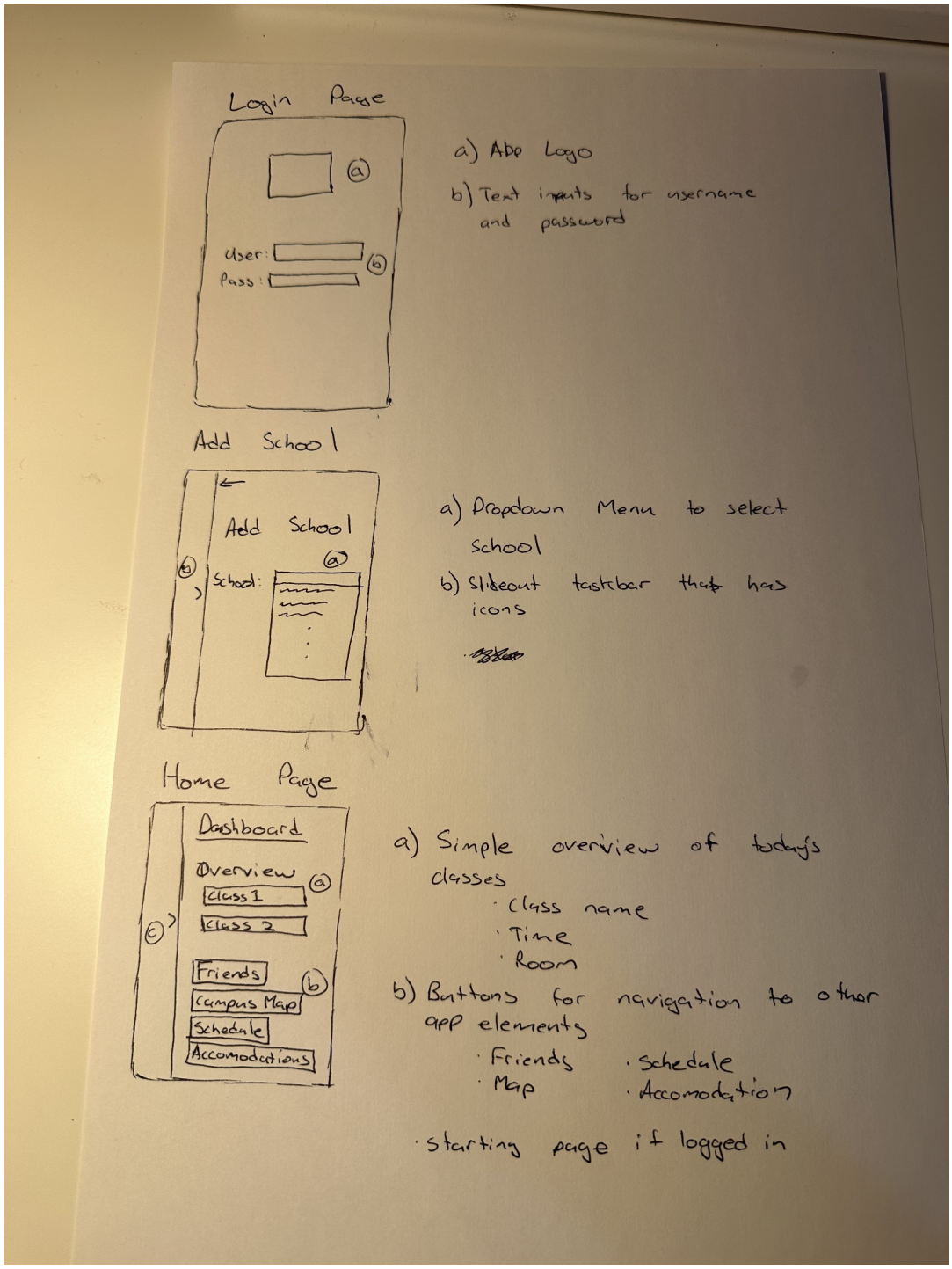
The association between MapService and Direction suggests that the map service uses direction objects to provide navigation instructions.

The return arrow from MapService to Direction in the getDirections method indicates that this method returns a list of Direction objects.

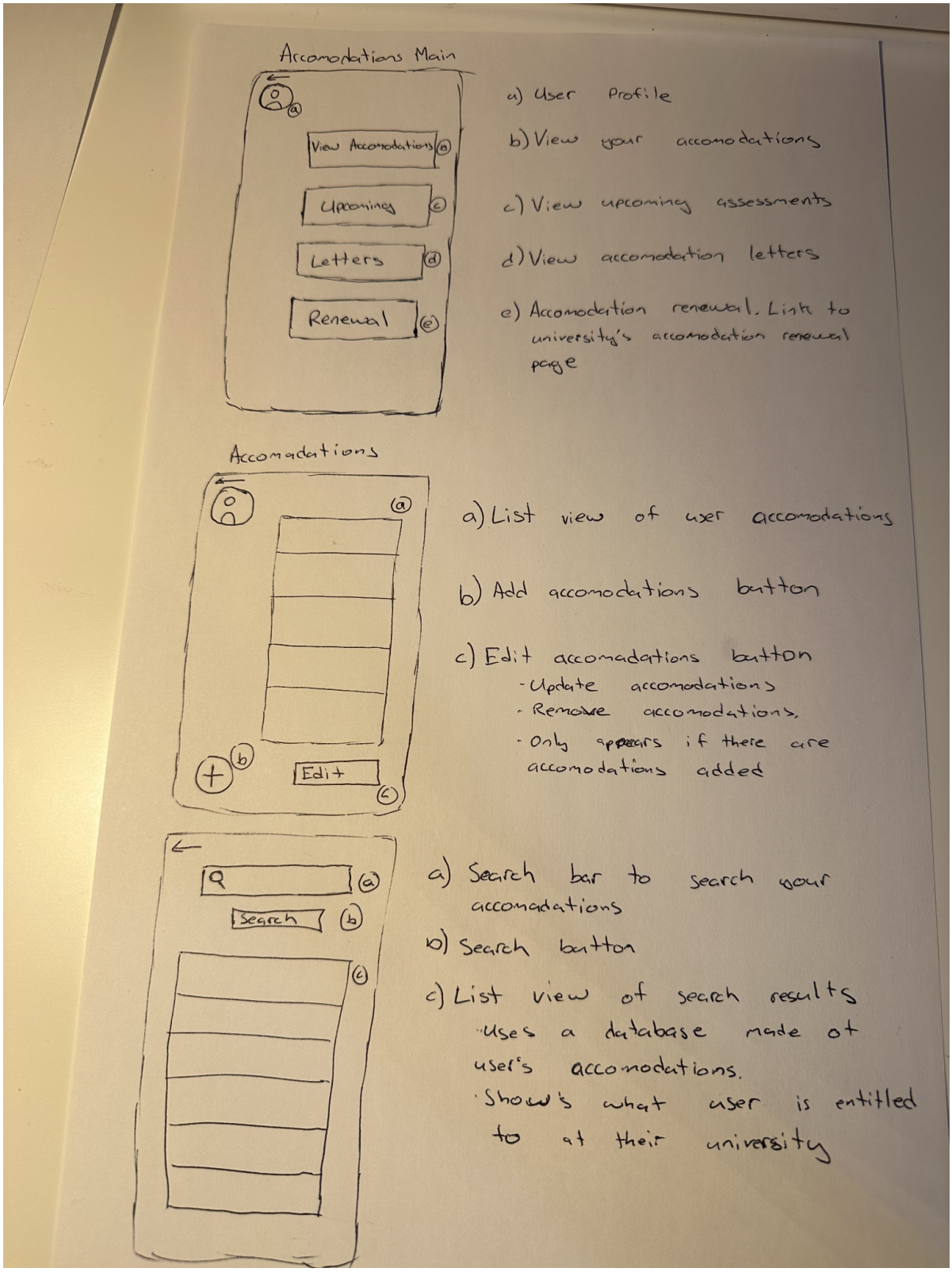
In summary, this UML class diagram illustrates the architecture of a campus map and navigation system. It provides an overview of the main components related to mapping and direction functionalities and their relationships in the "CampusMap" system.

Mockup of User Interface

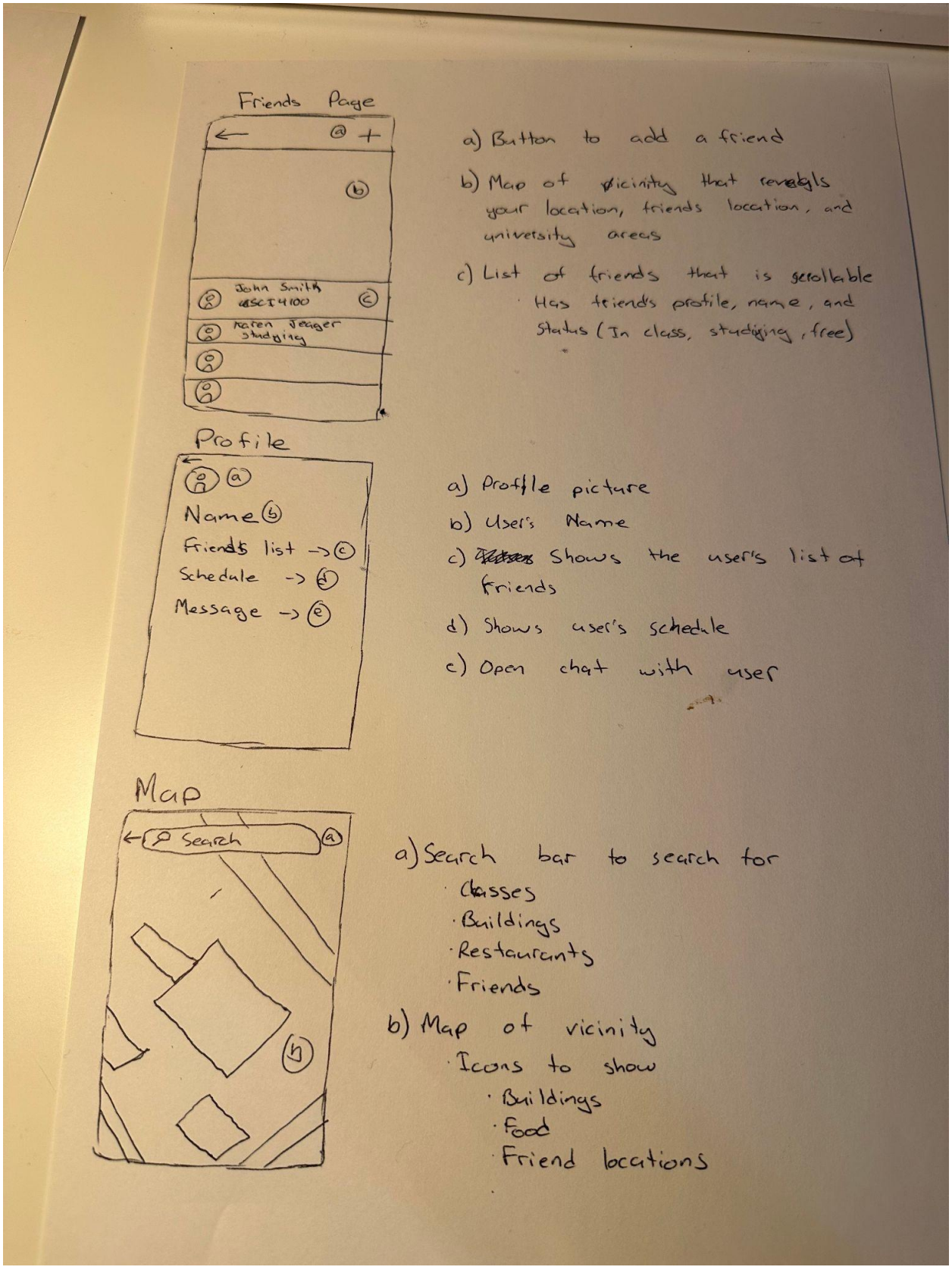
Login, Add School, Home/Main Menu pages



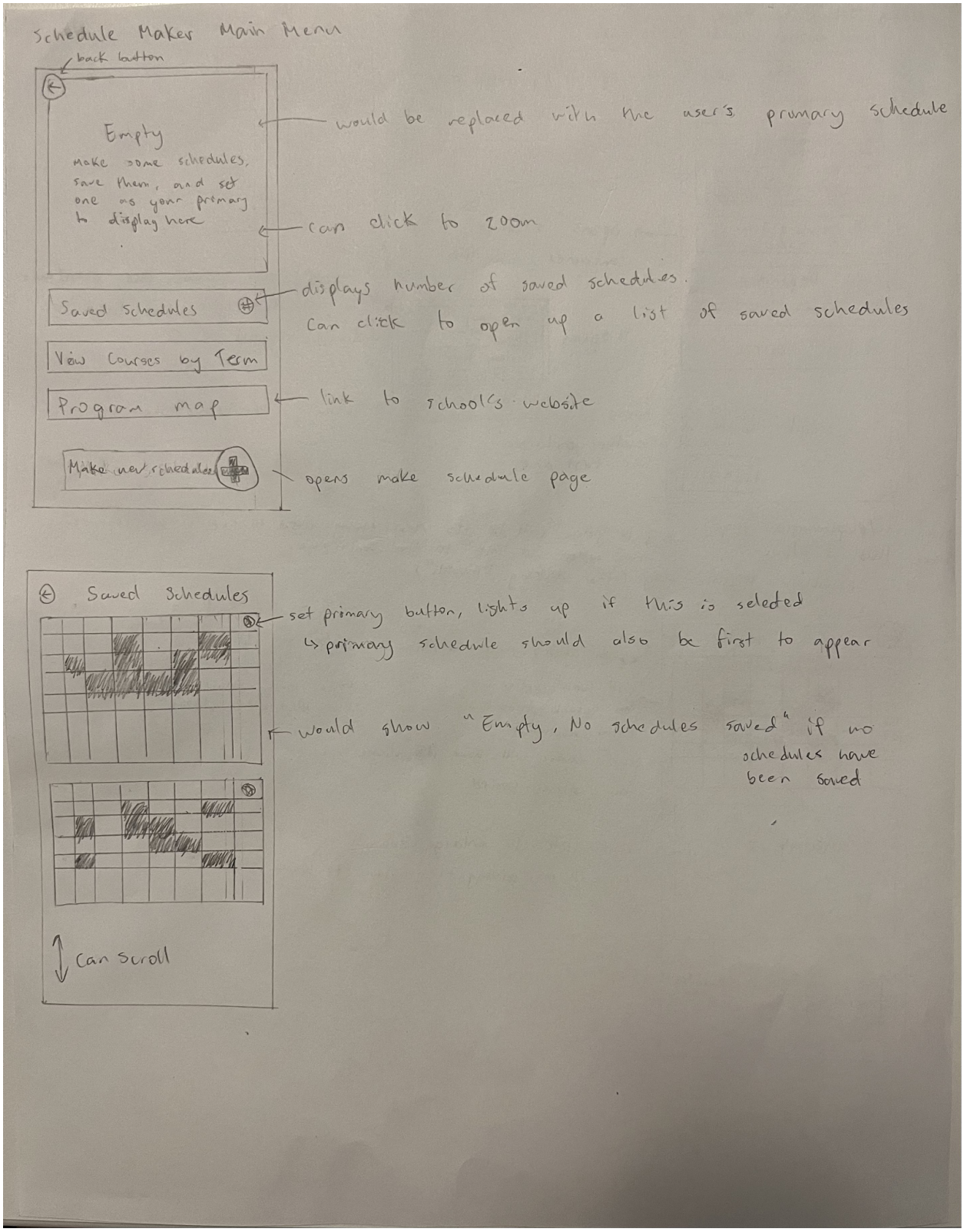
Accommodations Main Menu, Accommodations list (add/edit), Accommodations Search pages



Friends list, Friend/user profile, Map pages



Schedule Maker Main, View Saved Schedules pages



Make Schedule (add courses), Display Schedule Combinations pages