Prof. Olga Baysal **Deliverable 3: Comp 3004** **2016-11-28**

Nav U

Comp 9001

Ted Kachulis - 100970278
Parth Patel - 100963711
Alexander Skipper - 100978454
Mike Stupich - 100973305

Architectural Description
The intent of this part (1) is to identify, describe, and justify the architecture of our project.

- **Functional Attributes**
  The major functional attributes of this program come together to provide the functionality of our end project goal - offer the user a quick, reliable way to:

  a) Find his or her classes in a confusing, or new building.
  b) Get reminded of upcoming classes.

  The following are the four major functional attributes that we have identified:

  1. **Maps**
     a. Provide to-scale, top-down view maps for navigation.
     b. Floor plans show rooms, hallways, staircases, etc.
     c. These will be loaded in as .png files.

  2. **Directions**
     a. Provide a pathfinding algorithm to map user from location entered to destination entered.
     b. Draws path for a user between endpoints (Red line on top of our drawn .png maps)

  3. **Search Functionality**
     a. Allow the user to search for rooms immediately upon opening the app, and press a single button to display his/her path.

  4. **Schedule**
     a. Allow user to input, edit, and save their class schedule.
     b. Allow user to turn on/off notifications, and notification times for their classes.

- **Non-Functional Attributes**
  The non-functional attributes all benefit the major non-functional goals of this project, which are:

  a) Reliable functionality
  b) Offline use
  c) Device functionality (Low local storage, fast performance, low battery consumption)

  The following are the four major non-functional attributes that we have identified which support the goals of our project:

  1. **Performance**
     a. Load .png maps quickly, without a significant buffer or load time.
     b. Fast calculation of pathing algorithm, and searching for nodes.
     c. Well optimized for the platform - make sure that the application functions with current and recent backdated Android systems.

  2. **Storage**
     a. Small storage size; many users are concerned about having enough memory space on their devices - we would like to ensure that Nav U does not burden the user in this fashion.

  3. **Ease of Use**
     a. Clean and intuitive UI - the screen is laid out in such a manner that most people familiar with Android or iPhone UI can navigate Nav U with ease.

  4. **Battery Consumption**
     a. We want to avoid using unnecessary power. Being an offline app, and having no background processes open while not in use, we have accomplished this.

  5. **Accessibility**
     a. As previously stated, our application is a completely offline navigation system. Therefore no need for network connection, greatly improving the portability and ease of access for our project.
     b. In the future, if we were to incorporate the tunnels, this would be a vital attribute.

- **What Architectural Styles and Patterns We Implement**
  During our project, while encountering certain issues or remodeling our goals, our group tactics as well as architectural strategies have varied.

  From the beginning of our project we had a very clear idea of the functionality we wanted our app to bring into the user's life - this allowed us to start out with a bottom up design strategy. During a few major design decisions, we took a parallel design approach to decide on the better route. The major architectural style we have followed to this point though, is adaptive

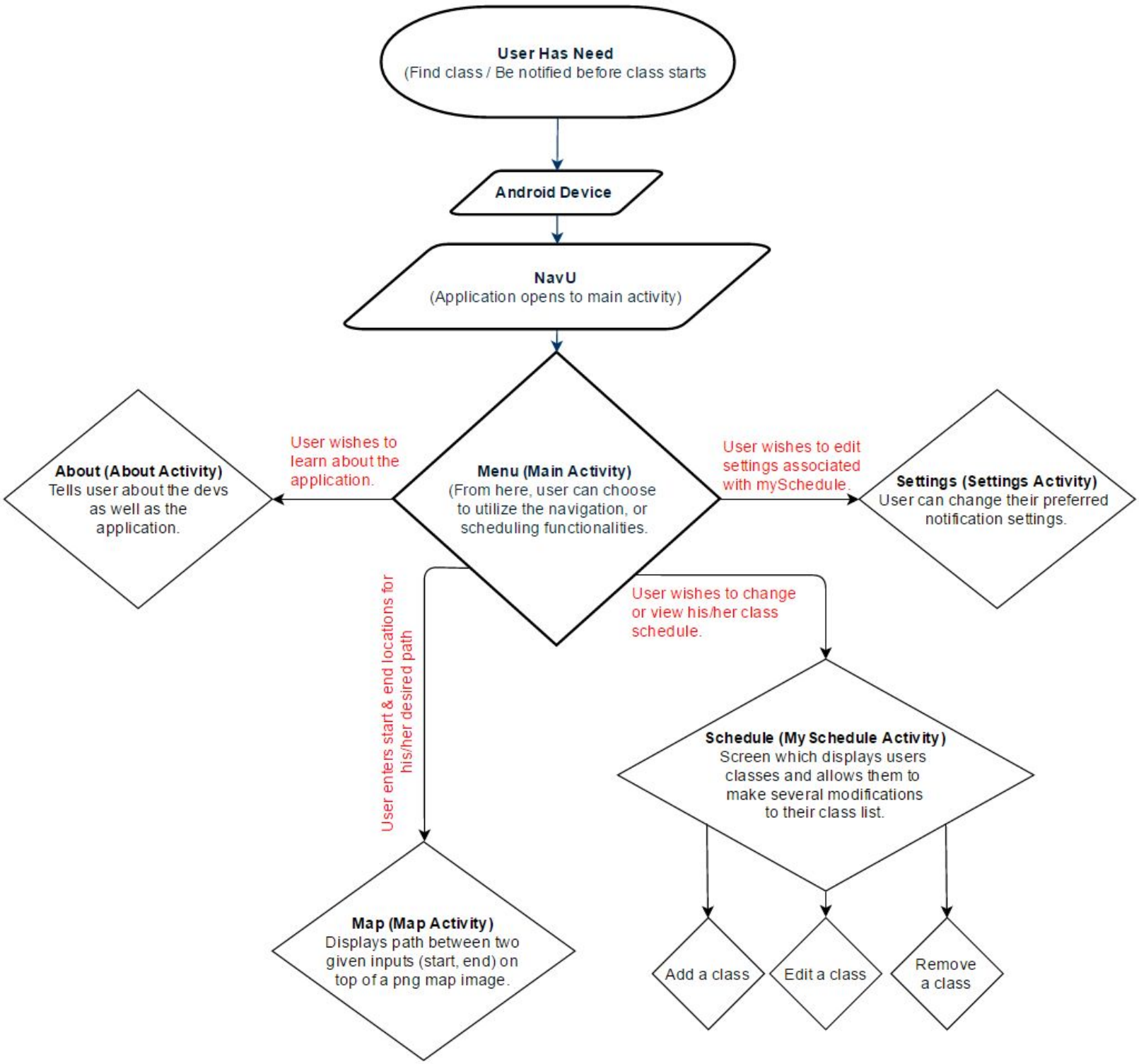
- **How the Structure is Suited to Give Functionality to Our Project Goals**
  Each architectural strategy above benefit us in one way or another during the development process. Below, it will be explained how each strategy was the best to use at that time in our development - whether it was to make a design decision, fix a problem, etc.

    - **Bottom Up:** We started out with some very specific ideas about map-related features (pathfinding, navigation) and built up from there, adding in UI and user functionality to complement our core implementation.

    - **Adaptive:** We followed an adaptive style because many the final implementation of certain features was contrary to our initial idea, forcing us to complete the majority of each feature before adding the next and connecting them together.

    - **Parallel:** Parallel design style was used for the map feature of our application. Initially we wanted to display the map via a vector file, because the maps were created using Adobe Illustrator (which creates vector files) and it would allow for resizing of images without compromising resolution. However, since we had already used bitmap images (for logos and artwork), we also considered rendering the maps as PNG files. In the end, this technique worked out for us, as we encountered no obstacles for the PNG files, compared to many issues we had with rendering the SVG vectors (such as unnecessarily large files, which used more resources to render). Another reason we decided PNG files are a better option, is because the maps take up the entire phone screen, making it easy to read without the need to resize.

Architecture Functionality Overview Diagram (Figure 1)



Design

Design Decisions for Non-Functional Attributes

To allow our app to remain reliable, and operate properly in any situation, we decided to not make use of any online API or resources. We use no server storage, and all classes are located physically on the user's device.

In the future, we may expand our mapping coverage to the tunnels, or campuses without WI-FI. This would require an offline method of loading maps and pathfinding - another reason why we have gone for an offline infrastructure in our design.

Significant Data Structures

The only significant data structure is the hash map used to store our MapNode objects. We used a hash map for O(1) node lookup, where the keys are the name of the nodes (nodes are unique), and the values are the nodes holding those names. The downside of this is that we use an extra string worth of space for each node, but the upside is a decrease in complexity of the nodes themselves. Instead of storing references to other nodes as node objects, we can store them as strings. This requires us to look up nodes by name, however, which for efficiency's sake meant switching from a simple ArrayList to a HashMap.

Accommodating future evolution

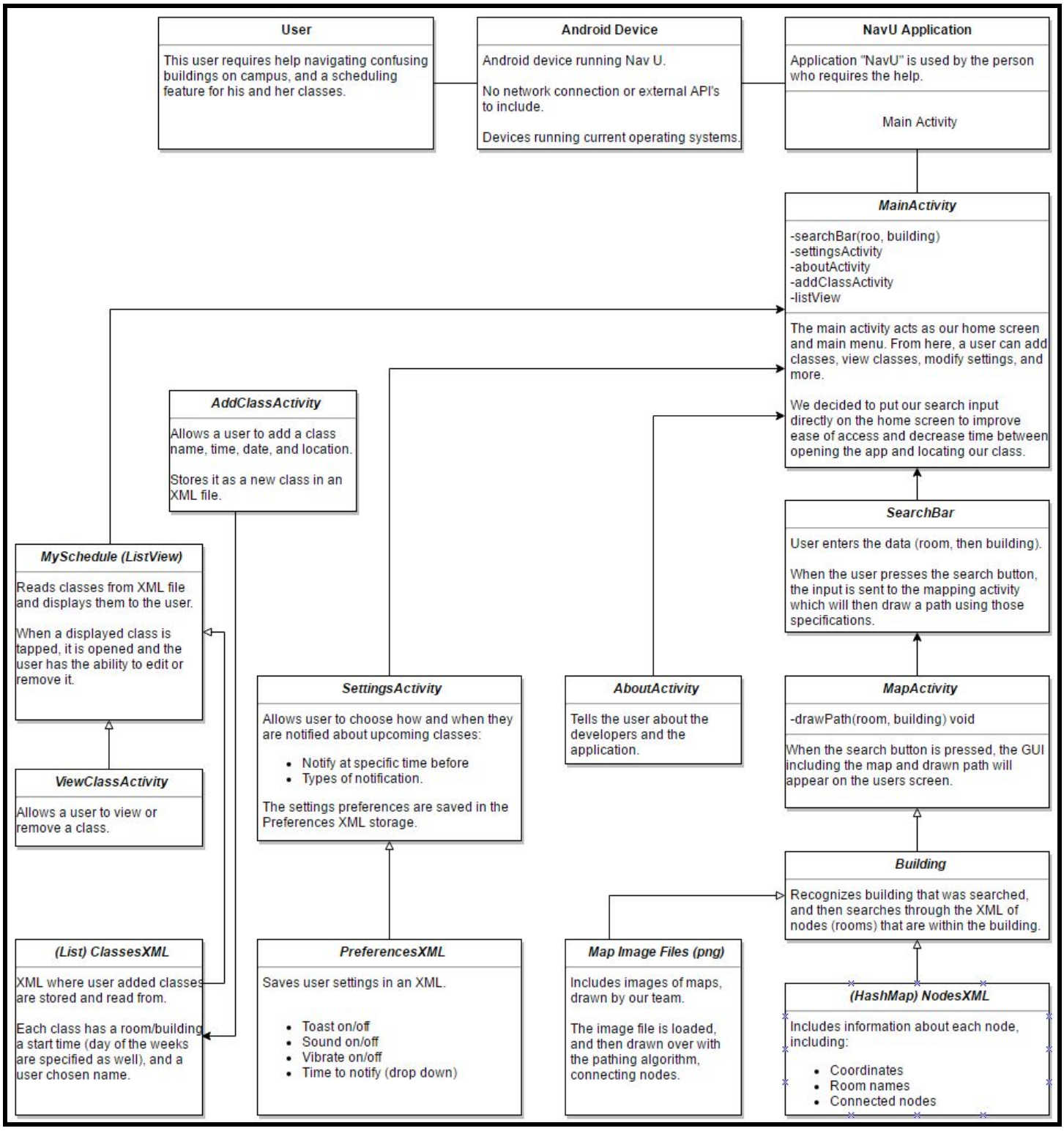
The most obvious evolution would be to include additional schools in our app. To avoid huge storage requirements on the phone (client), we might want to set up a database (server) that the app could access to download only the map data it requires.

Since all the data is stored as static files (XMLs/.png), and is processed as required, our design would permit easy swapping of those files, and assuming common formatting among all the files, the rest of the functionality could remain completely unchanged.

The only change would be perhaps a new activity accessible from the main activity allowing the user to choose what University data to download.

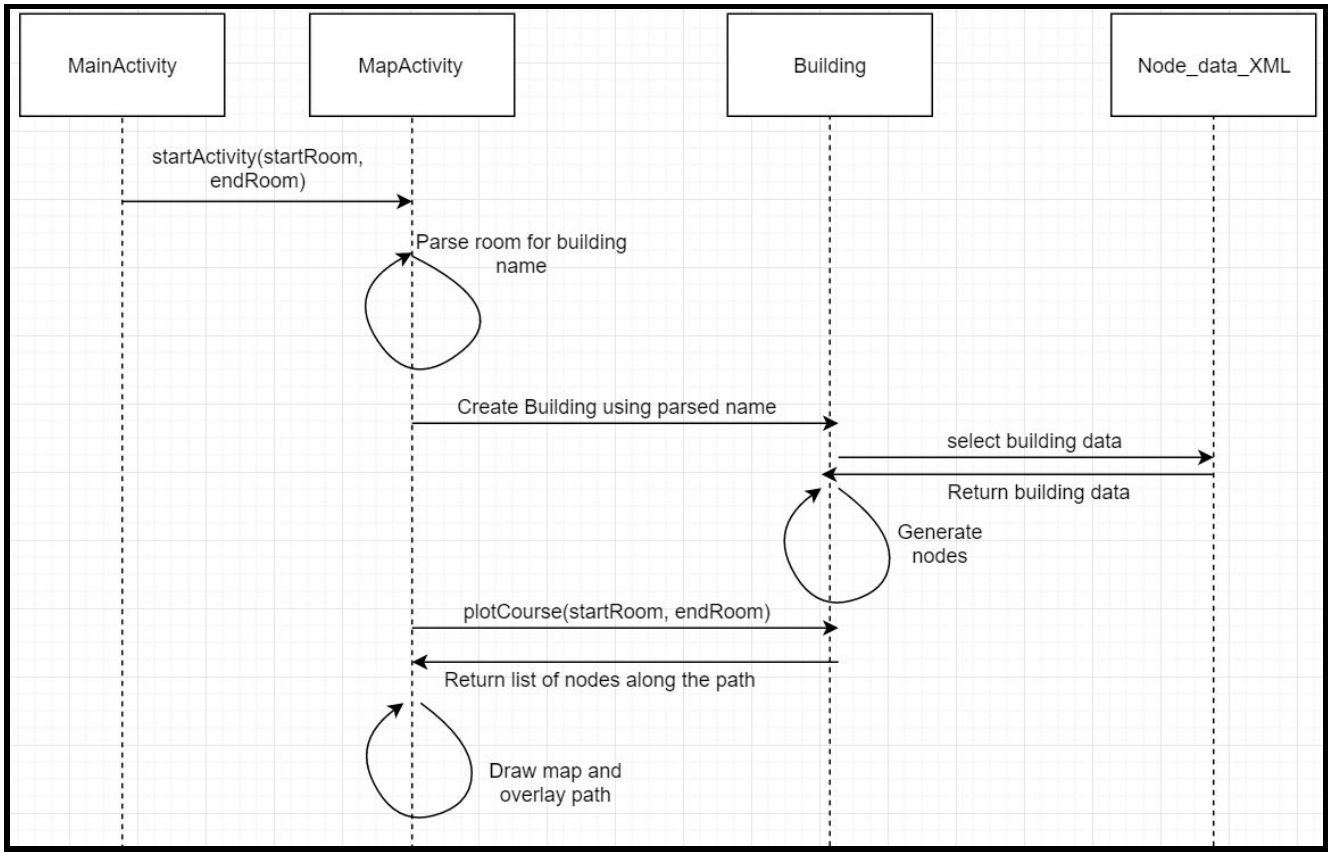
Another potential change is to have a "close up" 3D representation rather than a top down 2D view. In this case, we may need to add some additional art assets and would need to overhaul the rendering, but the pathfinding the node data wouldn't need to be changed at all.

Architecture Functionality Overview Diagram (Figure 1)



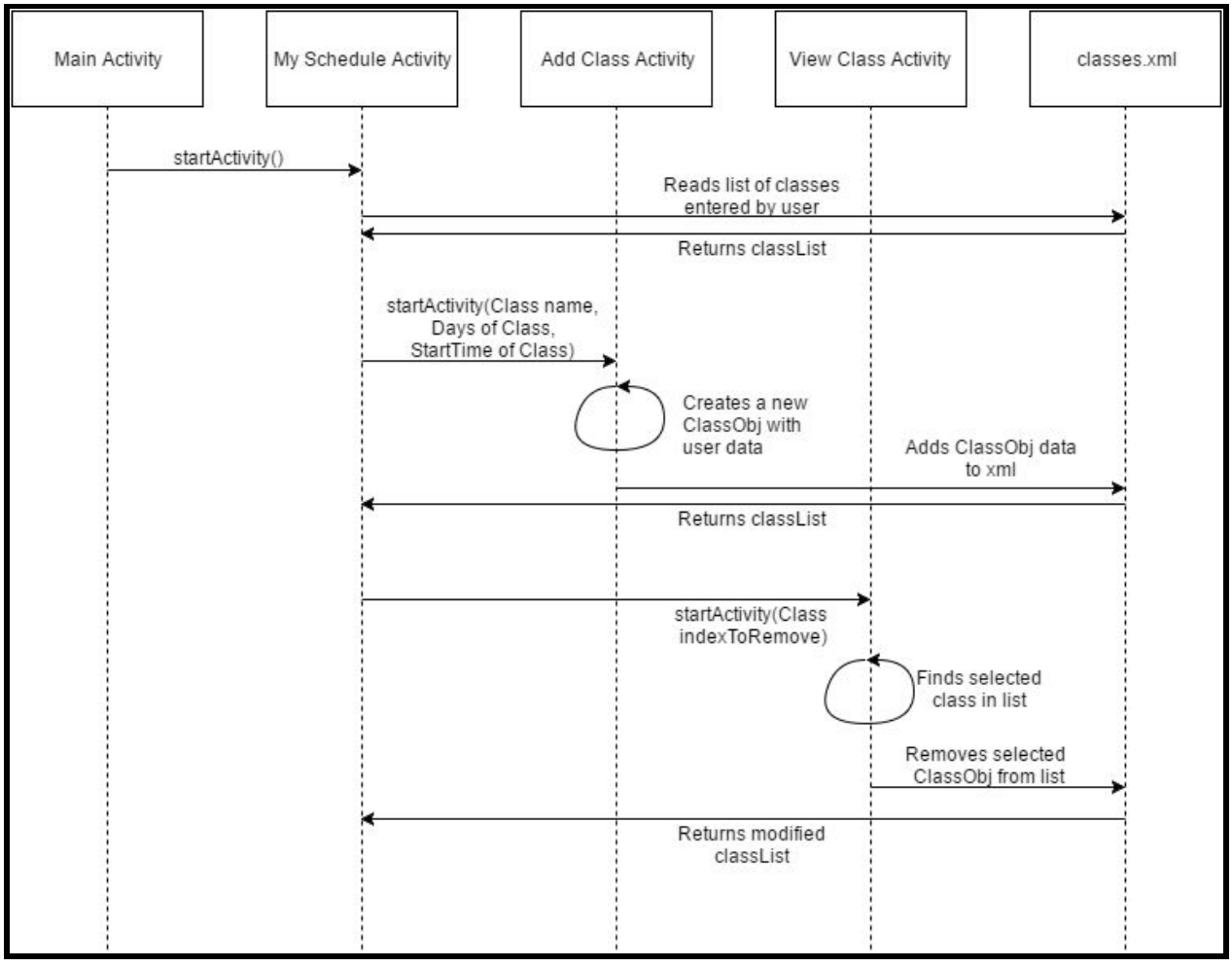
User Scenario 1

A student is in Herzberg laboratories and just finished a tutorial in HP4155, but really needs to see her Prof about some course material. Her Prof is in room HP5312 and the student has no idea how to get there, so she opens NavU and inputs HP4115 and HP5312 into the appropriate boxes. NavU then displays a map with a path drawn to her destination. The following sequence diagram shows how NavU would handle such an event:



User Scenario 2

A student has begun their year of school at Carleton University. They want to save their schedule somewhere so that they have quick access to it, and can check when their classes are, so they don't miss anything. The student opens NavU and enters their class data into the app, which stores it and displays it for the student to see. The user later withdraws from one of their classes, and would like to update their NavU class list accordingly. The following sequence diagram is how the program would handle such an event.



Team Tasks

Generally, we split the team into two sub groups - mapping/navigation/data management team (Parth/Alex) and scheduling/notification/design team (Ted/Mike). That being said, there was much overlap between the two, and often we all worked together to get tasks done.

The team would work in groups of two on specific components, and then also hold full group meetings to help each other solve issues we ran into individually, share status reports, and assign further tasks.

Below, each member has written what they feel themselves and others have contributed to most:

Alexander Skipper - 100978454 ● Design system for storing and reading map data. ● Write pathfinding algorithm. ● Display map and overlay paths ● Write search functionality for rooms. ● Link the main activity and notification system with the mapping system. ● Responsible for all GUI elements inside the map system.	**Mike Stupich - 100973305** ● Implemented functionality to save or remove local data (classes). ● Implemented settings page to allow user to specify notification settings. ● Implemented notification system to alert user of upcoming classes. ● Linked class storage with notification system to allow the above functionality. ● Worked with Alex to link the two major team projects together.
Ted Kachulis - 100970278 ● Designed screen/activity plans for presentation and for user interaction purposes. ● Architectural analysis, diagram construction, etc. ● Lead on presentation content and delivery. ● Lead on documentation for written deliverables. ● Write functionality/design GUI for the user to input a class and add it to their schedule.	**Parth Patel - 100963711** ● Obtained and edited photos of Herzberg Laboratories floor plans. ● Designed custom maps based on photos using graphic design tools. ● Helped map node coordinates on floorplans, which were used to create XML file for use by Alex's pathfinding algorithm. ● Responsible for passing user input to Map Activity. ● Helped with main menu design