COMP2100/6442 Group Project | **The Next Viral Social Network App**

v.01 - Project Description released (30.08.2021)

Assessment weight: This project is **worth 30% of your overall mark**.
Due Date: October 22nd, 2021 at 23h59 (Friday, Week 11) (**no late submission allowed**)
Presentations: Group Presentation (October 25th, 2021 at 4pm) and Individual Presentation (Week 12/TBD)

The objective of this project is to gain some experience in the process of software construction (the design, specification, documentation, implementation, and testing of substantial software). This project will also give you some practice in the design and implementation of a graphical user interface (GUI) application along with the use of several important development tools (particularly Android Studio and Git). **It is also an opportunity to put into practise and reason about some of the concepts presented during this course such as Data Structures, Tokenizer, Parser, Data Persistence, Design Patterns, Software Testing, etc.**

As you complete this project you should reflect on the overall design along with the software engineering process that you used in bringing this project to completion. This is a project so part of the objective is to gain experience in working in a team.

This semester the project involves the development of a social network app using Android/Java. Social networks are part of our everyday lives. We use them for sharing information, keeping in touch with family and friends, career development, keeping up-to-date, teaching and learning, and **even for social activism and other causes.** Besides that, the development of a social network app demands understanding several concepts covered in this course and it is an opportunity to put all of them in practice and together in a real and fun application. Although the theme for your social network is open (be creative!), there are some basic requirements your app must implement. Note that some simplifications have been made as developing a fully operational social network app can be very complex and unfeasible given the time we have available.

Part 1: Basic App (30 marks)
In the first part of the assignment, you must create an app which allows users to login, view and search for posts. Marks will be awarded for the following listed requirements. It is advised, although not necessary, that you complete this basic application before you can proceed to the additional features.

1. Users must be able to login (not necessarily sign up). **(up to 3 marks)**
2. Users must be able to load (from file(s) or Firebase) and view posts (e.g. on a timeline activity). **(up to 10 marks)**
3. Users must be able to search for posts by tags (e.g. #COMP2100isTheBest). The search functionality must make use of a tokenizer and parser with a grammar of your own creation. **(up to 3 marks without a tokenizer and parser. Up to 12 marks with a tokenizer and parser)**

4. There must be a data file with at least 1,000 valid data instances. There must be a data file that is used to feed the social network app simulating a data stream. For example, every x seconds, a new item is read from a file. An item can be a post or an action (e.g. like, follow, etc). **(up to 5 marks)**

The underlying implementation must contain:

1. At least one fully implemented data structure taught in this course (e.g., Binary Search Tree, Red-Black trees, AVL, etc) for organizing, processing, retrieving and storing data. We will also evaluate your choice and use of data structures on your project (not only trees, but also other data structures such as arrays, lists, maps).
2. Your app shall implement at least two design patterns covered in the course.
3. Your app shall retrieve data from a local file (JSON, XML or Bespoken).

Part 2: Features of the App (up to 70 marks)

The following is a list of features your application may employ. They are separated into categories whose features are listed in increasing difficulty. Please note that '[stored in-memory]' refers to storing data in the application's temporary memory and not storing the data persistently on the device itself.

You do not need to accomplish all features to achieve a good grade in this section, quality over quantity. Greater marks will be attributed to the excellent implementation of a few features as opposed to poor attempts at several features. For example, achieving the entirety of the 'peer to peer messaging' category to an excellent level of quality (great code documentation, excellent use of data structures, etc.) will net you greater marks than poor attempts at several features from various different categories. That being said, if a great number of features can be accomplished at a great level of quality, that would be preferred above all. We expect you to implement *at least* 7 features in total, with at least 4 features classified as 'easy', 2 features classified as 'medium', and 1 feature classified as 'hard'.

Improved Search

1. Search functionality can handle partially valid and invalid search queries. (medium)

UI Design and Testing

1. UI must have portrait and landscape layout variants as well as support for different screen sizes. Simply using Android studio's automated support for orientation and screen sizes and or creating support without effort to make them look reasonable will net you zero marks. (easy)
2. UI tests using espresso or similar. Please note that your tests must be of reasonable quality. (For UI testing, you may use something such as espresso)
    a. Espresso is not covered in lectures/labs, but is a simple framework to write Android UI tests. (hard)

Greater Data Usage, Handling and Sophistication

1. Read data instances from multiple local files in different formats (JSON, XML or Bespoken). (easy)
2. User profile activity containing a media file (image, animation (e.g. gif), video). (easy)

3. Use GPS information (see the demo presented by our tutors. For example, your app may use the latitude/longitude to show posts). (easy)
4. User statistics. Provide users with the ability to see a report of total views, total followers, total posts, total likes, in a graphical manner. (medium)
5. Deletion method of either a Red-Black Tree and or AVL tree data structure. The deletion of nodes must serve a purpose within your application (e.g. deleting posts). (hard)
    a. Note that this advanced feature will only be considered if the Red-Black tree or AVL tree is the most suitable data structure to the App you are developing. Note that the deletion is not covered in lectures, this is optional (see deletion algorithm in the references of the data structure lecture).

User Interactivity

1. The ability to micro-interact with 'posts' (e.g. like, report, etc.) [stored in-memory]. (easy)
2. The ability to repost a message from another user (similar to 'retweet' on Twitter) [stored in-memory]. (easy)
3. The ability for users to ‘follow’ other users. There must be an adjustment to either the user’s timeline in relation to their following users or a section specifically dedicated to posts by followed users. [stored in-memory] (medium)
4. The ability to send notifications based on different types of interactions (posts, likes, follows, etc). A notification must be sent only after a predetermined number of interactions are set (>= 2 interactions [e.g., one post and one follow or two posts or two follows]). Note that it is not mandatory to use the Android Notification classes. (medium)
5. Scheduled actions. At least two different types of actions must be schedulable. For example, a user can schedule a post, a like, a follow, a comment, etc. (medium)

User Privacy

1. Friendship. Users may send friend requests which are then accepted or denied. (easy)
2. Privacy I: A user must approve a friend's request based on privacy settings. (easy)
3. Privacy II: A user can only see a profile that is Public (consider that there are at least two types of profiles: public and private). (easy)
4. Privacy III: A user can only follow someone who shares at least one mutual friend based on privacy settings. (medium)

Peer to Peer Messaging

1. Provide users with the ability to message each other directly. (hard)
2. Privacy I: provide users with the ability to ‘block’ users. Preventing them from directly messaging them. (medium)
3. Privacy II: provide users with the ability to restrict who can message them by some association (e.g. a setting for: can only message me if we are friends). (hard)
4. Template messages or Macros (for peer to peer messaging or template posts (e.g. a quick one-tap post)). For example, "Hi %USERNAME%, I am not available now. Call

to %PHONE_NUMBER% if it is urgent. Cheers, %MY_USERNAME%". The use of tokenizer and parser is mandatory. (hard)

Firebase Integration

1. Use Firebase to implement user Authentication/Authorisation. (easy)
2. Use Firebase to persist all data used in your app (this item replace the requirement to retrieve data from a local file) (medium)
3. Using Firebase or another remote database to store user posts and having a user's timeline update as the remote database is updated without restarting the application. E.g. User A makes a post, user B on a separate instance of the application sees user A's post appear on their timeline without restarting their application. (very hard)

Suggest a new feature

Is there a feature you would like to implement but it is not listed here? No worries, you can post your feature idea on our "*voice your feature*" public discussion forum with the information below. We will assess the proposed features and if it is approved, will be given a difficulty classification. Please note:

1. Any features which are approved in the forum can be pursued by any group.
2. Any feature that is refused can still be pursued but will NOT net your team any marks. Although it might count towards the creativity criterion.
3. We will only accept new features until the end of Week 8.

A feature suggestion MUST contain:

- Post subject: feature name
- A description of the feature.
- Details as to what the feature would entail. E.g. An additional tokenizer, custom B-Tree, etc.
- Why is this feature relevant to the group project and course? (short explanation, please, link to the course content)
- Suggested difficulty level: (easy, medium, hard)

Please try and keep any features mentioned relevant to the course and the assignment context. Any features that stray too far from either the course content or the assignment context will be refused.

Feature Request Example:

Subject: Partially valid and invalid search query handling.

Description: The application's search bar will be able to handle both valid and invalid search queries without crashing the application and still providing the user at least some search responses inline with what was valid.

What the feature entails: modifying the tokenizer/parser that the application uses to handle valid and invalid search queries without crashing and still providing a response.

Feature relevance: Tokenization and parsing.

Difficulty: medium.

A kind reminder: A feature that does not work is not a feature, it is a bug.

Surprise! Building software can be an exciting activity. At some point in **Week 10**, features may change (or added) and you may need to adapt your project to meet them. This is how it works in REAL LIFE, and we will simulate it here.

Be prepared for changes! Build your software in a way that it is easy to extend, change, or add new features.

You may be asked to make small changes. This is to practice the software development/construction process. If you successfully develop it, **your final mark may be increased by up to 2 marks** (your client will "pay" more to have better software).

If a group does not participate in the checkpoint, they will not be eligible for this "surprise" (missed opportunity).

Checkpoint

In week 10, we will have a checkpoint. You must participate in your lab session and show your tutor what stage of development you are in (at least ONE member of the group has to present). We expect that you have at least a schedule of your development activities, a good part of the code developed and that you have divided the roles between the members of your group by Week 10. Your tutor will review your schedule and will provide quick feedback. **The checkpoint is mandatory.**

*Note that every customer wants to know how the software is progressing. Here's the same, your tutor is your customer, you need to show that the app is being developed and that you will meet the deadline**. If you do not meet this checkpoint, your final mark for the group project will be reduced by 5 marks (you missed an important meeting!). Only one member of your group is required to present it** (make sure someone in your group did it).

Question and Answers

What kind of social network app should we develop? Twitter, Facebook, Instagram, TikTok are just a few examples. However, we expect you to think outside the box and develop your own idea! Creativity and Novelty are part of the marking criteria. It does not need to be complex, simple is also good.

Why and How to simulate a data stream? To simplify the development of the social network instead of using a client-server model, we allow you to create a file to simulate users interacting with your app. You may want to create a method or module to read from a file every *x* seconds and feed the new information into your application.

How should I develop the tag search? Your search mechanism is responsible for tokenizing the query, parsing and evaluating it. You must define your own grammar and document it. You can use the CFG Stanford tool to help you create your grammar.

Example: #topic (# used for finding posts) and @users (@ used for finding users)

Search: #comp2100 @john
In this case the user is looking for posts tagged with "comp2100" authored by the user "john".

Which data structure would be most appropriate (or efficient) for this case? Your group must decide how to implement it.

What about the data? The data must be in a file and be structured in a way that is easy to be retrieved/processed.

Let's say the data for your app is stored in a local XML file structure as follows. Here is an example for posts (not necessarily the best format, you must decide the best format for your app, we will just evaluate it). This format does not consider, for example, *who* liked a given post.

```
…
<posts>
        <post id="143" author="1213" media="...">
                <tag>COMP2100</tag>
                <content>Yay, my app is running well!!</content>
                <like count="10"/>
                …
        </post>
        …
</posts>
...
```

*Note that you need to generate the data instances for your application as well the data format/structure. You are free to choose any file format (JSON, XML, Bespoken). You can create a script to generate your data, a script to scrape from the Web or manually create it. You can also get data provided by some API (e.g. Twitter API). This and only this script can be written in any programming language (Java, Python, etc). Please, make sure you are allowed to download the data from external sources and that the script is included in your repository.

**As an option, you can use Firebase (and JSON) if you feel comfortable with it and this is the best choice for your app.

What about the data structures? You must know where, which, how and when to use it. It depends on how you design your app. Think about what data structure you would use if you needed to look for posts containing a specific tag. Think about what data structure you would use if you need to look for a user. For example, if you go for a tree, what would be the key of your tree? Is this the most appropriate data structure to your app? Discuss with your group.

Which design pattern should I use? Again, you must know where, which, how and when to use it. It depends on how you design your app and the features you implemented. Several features were proposed thinking about the most appropriate design pattern to be used. You can use design patterns that were not covered in the course, but do not forget to explain/justify in your report.

How complicated should my grammar be? It does not need to be complicated, but it must be unambiguous and easily extendable. Most importantly, you have to demonstrate that you know how to implement it and be consistent with your app theme.

What should my app look like? You are free to design your app. Be creative and check the rubrics.

Do you have any advice? ***Read this document and do not leave it to the last minute. Define tasks to each member of the group. Regularly check if your group is advancing with the tasks given to them. Read and discuss each item in the list (parts 1 and 2) with your group before implementing it. Be prepared for a plan B if something unexpected happens. Don't leave it to the last minute (yes, I intentionally repeated it).***

General Information about the Project (assessment, documentation, submission, presentations, etc)

Working physically apart, but remotely together!

Some of you will need to work remotely to complete this project. No worries! It is very common to have remote collaboration on software development and there are many tools to help you accomplish this work. Use this opportunity to practice collaborative tools, like Git, Zoom, Microsoft Teams, Google Doc, Slack and others to plan, design and implement your project. This is exactly what you will find in the industry, therefore, use this project as an opportunity to get some experience in these tools.

Use the forum on Wattle to help you find teammates: Introduce yourself, explain how you can contribute to the project, give some details of the theme of the app you want to develop, etc.

What is the minimum/maximum number of members per group?

Each group should **consist of 4 students** (minimum 3). Groups can contain a mix of undergraduate and masters students.

****Your group must be created in Week 6.**
Do not start late, it may impact your project outcomes.**

How will projects be assessed?

1. Everyone in the group must implement and commit some code (**we will check it!**).
2. You will be assessed individually and as part of the group. You must make sure that the work is divided up so everyone has the opportunity to undertake some coding and contribute to the overall of the project. Marks will be reduced for those group members who contributed much less than others, and possibly for the group for poor group management (remember, this is a group project and you must work as a team and collaborate effectively). Note that **each group** is expected to contribute approx. 80 hours for this project. By default, the marks **WILL NOT** be the same for each member of the group. Marks will be based on several criteria (check rubrics). Group members who contributed much less than others may receive a 25% penalty. Members who did not contribute to the group assignment will receive zero marks in this assignment.
3. Based on the quality and depth of your report and code documentation.
4. Based on your App, implemented features and quality of decisions. See the Rubrics section for more information about the marking criteria.

Report (important)

The report should give us evidence and details of the functioning of your group, project decisions and implementation. The technical outcome is a key aspect of the evaluation of this project and the evaluation of your teamwork is just as important.

Therefore, along with the actual implementation, you will be required to produce a report relating to the design and team management of your project. This must include (but not limited to):

- GitLab markdown report document titled ‘report.md’. We advise you to use the provided template which will cover at least the required topics.
- Team structure and roles
- An app summary with screenshots
- A design summary page (include justification for decisions made, diagrams, etc)
  - Details about the parser (describe the grammar and language used)
  - Decisions made (e.g., explain why you chose one or another data structure, why you used a specific data model, etc.)

    - Details about the design patterns used (where in the code, why did you choose this design pattern, etc)
    - If you implement the surprise item, explain how your solution addresses the surprise task.
  - A summary of known errors/bugs (list of bugs).
  - A list of examples/use cases of your app.
  - A UML diagram (e.g. class diagram).
  - A testing summary section (e.g., number of test cases, coverage, etc.)
  - A summary of implemented features
    - List all features implemented in your project
    - Separate features into their categories and provide the difficulty classification.
  - Conflict resolution protocol (if your group has problems, what is the procedure for reaching consensus or solving a problem, etc.). Suggestion: Define this protocol in your first meeting.
  - Team meeting minutes (at least 3)
    - Your first meeting should develop a clear plan of how the work will be divided, documented in meeting minutes.
    - Right after every meeting, upload the minute to your project repo. Note that we will check the dates of submission in your repo.
  - Git commit history (starting from the release of this group project)
    - Use your own account to commit. Failing to do this may result in zero marks.
  - A statement of originality (template available in our GitLab Repo).
  - *Individual Reflection* (Please include in your individual presentation):
    - *Individual reflections: You must write 100-120 words related to your experience during the group project (be concise and direct). For example: how was your experience working in a team? Reflections on what your team could have done better, what worked and what did not work? How was the work divided and was that fair? Individual reflections must be submitted to Wattle in Week 11, at the same time as the rest of the work for this submission.
- A template for the report is available in our GitLab Repo. The design summary is an important item of your report and should include all decisions made and information required to understand your decisions. Be concise (just in case, according to the Oxford dictionary, concise means "giving a lot of information clearly and in a few words; brief but comprehensive").

All attempted features MUST be documented

Please list all features you have attempted. Preferably, we would like you to separate them by category and classify them for difficulty (this helps with marking). You may copy the format straight from the feature's list itself (as seen below). Remember that we expect you to implement *at least* 4 features classified as 'easy', 2 features classified as 'medium', and 1 feature classified as 'hard'.
For example:
....

User Privacy

1. Friendship. Users may send friend requests which are then accepted or denied. (easy)
2. Privacy I: A user must approve a friend's request based on privacy settings. (easy)
3. Privacy II: A user can only see a profile that is Public (consider that there are at least two types of profiles: public and private). (easy)
4. Privacy III: A user can only follow someone who shares at least one mutual friend based on privacy settings. (medium)

Firebase Integration

1. Use Firebase to implement user Authentication/Authorisation. (easy)
2. Use Firebase to persist all data used in your app (this item replace the requirement to retrieve data from a local file) (medium)

…

Check assessment rubrics for more details.
*If you are not familiar with YAML, you can learn at: https://commonmark.org/help/.

Project Submission

We will use the school's GitLab server for submitting the project.

To get you started, use the markdown files available in our Repo and create your own repo in your GitLab. Check the project's markdown files and modify them as required. This also means people will have basically the same layout which makes it a little easier for marking.

Files available: Report.md (use it as a template for your report); Checklist.md (project minimum requirements); statement-of-originality.yml (all members must be listed there); MeetingTemplate.md (a template for your meeting minutes).

You do not need to submit the project files to Wattle but it must be in your Project GitLab Repo. We will check your repository during the individual demonstration and after for marking.

The project **cannot** be updated after the deadline. You may get zero marks.

Group Project Demonstration

In week 12 there will be an online group demonstration (Zoom, Microsoft Teams or a similar tool). This will be an opportunity to showcase your project and also to get feedback and marks on what you have done. This Project Demonstration will follow the Minute Madness format where each group will present a set of timed slides within a predetermined time (slides will change automatically every x seconds). The total number of groups will determine the duration of each presentation and the number of slides.

Your presentation must clearly present your topic and convince the audience that your app is innovative and could be used in the real world. You must briefly talk about the structure of

your project, decisions made, and solutions for the problems faced during the project (use of a particular data structure, design pattern, etc).

Your presentation must be clear, convey your ideas and give an overview of the software construction process you and your team experienced.

I would recommend using some screenshots to show how your app looks (images can be better than words sometimes).

More details related to the project demonstration will be released in Week 10. Slides must be uploaded to the link to Microsoft PowerPoint in Week 11.

Individual Project Demonstration

In Week 12, and possibly Week 13, there will be individual presentations during the lab slots (we may need extra slots to accommodate all of you).

We will ask specific questions related to any part of your app and the development process. We expect you to demonstrate knowledge of any part of the code, test cases, documentation, report, etc. As a team member, **you should be able to answer questions of any part of the code (regardless of whether you developed that part of the code or not).**

Your final Group Project mark will be based on this presentation and according to the demonstrated knowledge criterion (see assessment rubrics and overall mark calculation).

OVERALL MARK CALCULATION

The overall mark will be calculated for each student as:

$$\textbf{Final Group Project (FGP) = GP * (DK}_s\textbf{/100)},$$

where GP corresponds to the marks obtained by implementing all features and DK_s corresponds to the knowledge demonstrated by each student in the individual presentation (see "demonstrated knowledge" criterion). **Note that a student who has actively contributed to the development of the project and has collaborated with other group members should have no difficulty in obtaining individual full marks (DK = 100).**

For example: Let's say the group "Android Masters" obtained 80 out of 100 marks (considering all assessment criteria/features [except the demonstrated knowledge criterion]). In this case, the GP mark for the Android Masters group will be 80. The Android Masters group consists of 3 students (student A, student B and student C). Each of them presented the group's project individually and were evaluated based on the criterion "demonstrated knowledge". Student A, who did not collaborate well and was not aware of other parts of the code, obtained 90 for the individual presentation whereas Student B and C obtained the same mark 100. Thus, Student's A final mark will be calculated as:

$$FGP = 80 * (90/100) = 80 * 0.9 = 72 \text{ marks}$$

And Student's B and C will be calculated as:

$$FGP = 80 * (100/100) = 80 * 1.0 = 80 \text{ marks}$$

So, Student A overall mark for the group project is 72, Student B overall mark is 80 and Student C overall mark is also 80. You will be required to show your Uni ID before starting the individual presentation. **The maximum mark for this project is 100.**

Due date and late submission policy

The project is due on the Friday of **Week 11 at 11:59 pm**. As the project will be done iteratively over the second half of the semester, every group should have something that will gain a pass mark well before the due date. No late submissions shall happen. Whatever your group has done up until the due date will be assessed. You are not allowed to submit/update any files after the deadline.

Note that there will be no Lab activities in Week 10 and Week 11, there will be checkpoints. It is recommended that you plan the development of the project considering these two weeks as a Sprint (as in Agile Methodologies) and have a release of the App. Ensure your App meets all requirements.

Originality

The project must be your own original work. If you make use of any code that is not your own it must be clearly referenced. This can be done by adding a simple comment next to the code stating where you obtained the code from. You must also add this to the statement of originality document. This is **very** important, as any breach of this needs to be investigated and reported. You are much better off not doing this project then copying a small part of code and risking academic misconduct. Remember, we are assessing your code, not someone else' code.

Every person in the group is responsible for the originality of every part of the project (regardless of who actually wrote or contributed to it). Any significant break of the academic honesty and plagiarism policy will result in the entire group receiving the mark of zero for the group project.

Assessment Rubrics

Remember this project **is worth 30% of the overall mark**. Teamwork is a key learning outcome for this project, so I would encourage people to prioritise working well as a team over extending your project.

There are two components in total marks: **basic app** and implemented **features.** The full total mark of the project is **100 marks**. The basic app and its features will be assessed based on the criteria listed below.

A full mark will only be granted if there are no bugs or design issues for each implemented feature. Partial marks may be given depending on the quality and progress of implementation.

The **basic app and its features** will be marked based on diverse aspects of software design. The following rubric lists the aspects of consideration based on the judgement of the markers. The full mark requires to be outstanding in all aspects.

Criteria	Excellent	Very Good	Satisfactory	Unsatisfactory
Demonstrated Knowledge (only for individual presentations, check how it is calculated)	Student can accurately present and answer all questions with sufficient justification and proper references to the existing work. Presentation is very clear and covers all topics, within the time limit.	Student can accurately present and answer most questions with sufficient justification and proper references to the existing work. Presentation is clear, good topic coverage, within the time limit.	Student can accurately present and answer half of the questions with sufficient justification and proper references to the existing work. Presentation is mostly clear, limited topic coverage, and exceeds the time limit by 1 min.	Student appears to have insufficient knowledge to present and answer most of the questions and/or has no or little understanding of the existing work. Presentation is unclear, limited topic coverage, and exceeds the time limit by more than 1 min.
Data Structures	Chosen data structures well suited to subsequent usage, leading to efficient code in terms of both programmer and computer time. The chosen data structure can grow as the input grows without concern for memory (scalability).	Data structures well suited for subsequent usage with minimal conversion or transformation required.	Satisfactory choice of data structures leading to inefficiency or repeated need to convert data structures or values to solve tasks.	Unsuitable data structures chosen leading to significant inefficiency or inability to solve tasks. Data structures loaded in each task or different data structures used for each task.
Code Quality and Organization	Excellent documentation, naming and style, following conventions and making the code easy to read. Code is well commented and code reviewers can clearly understand the code. Code is very well organized, using generics and inheritance as appropriate. Construction for reuse. Easy to extend. The overall program structure makes the code easy to follow.	Good Documentation, naming and style are complete, consistent and appropriate. Code is well commented and code reviewers can partially understand the code. Good code organization with appropriate use of inheritance. (mostly) Construction for reuse. Easy to extend. The overall program structure makes the code easy to follow.	Reasonable attempt at documentation, naming and consistent style, but could be improved in places. Code is partially commented and code reviewers can partially understand the code. Alternatively, more significant shortcomings in one aspect. Inconsistent organization. Repeated code. Construction with Reuse. Code can be partially extendable.	Missing or poor documentation, poor naming or inconsistent or poor coding style. Significant unnecessary code. Lack of comments in code / Code reviewers cannot understand the code. Poor code organization. Code is very difficult to follow. (mostly) Construction with Reuse. Difficult to extend.
Report	Report is well organized, well presented, clear and concise. Project decisions are well detailed with examples and discussion. It also presents code and	Report is well organized and presented. Project decisions are detailed with suitable examples and	Report is organized and presented. Project decisions are not clear but present some examples and	Limited discussion or understanding of issues. Poor decisions made and choice of examples. Poor organization, or lack of clarity makes the

	analysis. UML diagrams are presented. A clear and detailed testing summary is provided. References are provided.	discussion. It partly presents code and analysis. Some UML diagrams are presented. A clear testing summary is provided. References are provided.	discussion. It does not present code and analysis. References are partially provided.	report hard to follow. References are not provided.
Testing	JUnit coverage test achieves at least 70% of code (without UI). Clear evidence of testing to verify correctness and robustness. Exceptions and error cases are checked. Clear evidence of Unit and Integration tests.	JUnit coverage test achieves at least 60% of code (without UI). Some evidence of testing to verify correctness and robustness. Clear evidence of boundary and normal functioning tests. Some evidence of Unit and Integration tests.	Repeatable unit testing is performed on the majority of the project. Appropriate use of Test Suites and Parameterized tests.	Tests are not well designed (random tests). Minimal or no test cases can be found.
User Interface	User interface is intuitive and can be easily used without guidance. Use of consistent theme and style. Feedback is provided to users based on interactions. UI is responsive (adapt to different screen sizes and orientations). Stylish and friendly look and feel.	User interface is mostly intuitive and can be used with little guidance. Use of consistent theme and style. Some feedback is provided to users based on interactions. UI is mostly responsive (reasonably adapt to different screen sizes and orientations) Friendly look and feel.	Standard user interface and not very intuitive (needs some guidance). Use of theme and style are not consistent. Navigation is not clear. Lack of feedback to users based on interactions. UI is responsive and provides only screen orientation mode.	Standard user interface and unintuitive (needs guidance). Inconsistent use of theme and styles. Navigation is not clear. Lack of useful feedback to users based on interactions. No guidance to users. UI is not responsive and provides only one screen orientation mode.
Creativity / Uniqueness / Specific Theme	App is innovative and can be applied in a real world scenario. Problem statement is clearly defined and has the potential to impact multiple beneficiaries. Unsolved problem with high significance.	App is innovative and can be applied in a real world scenario to a limited number of beneficiaries. Problem statement is clear and has the potential to impact a limited number of beneficiaries.	The developed app can be applied in a real world scenario to a limited number of beneficiaries. Problem statement is clear but with low potential for impact.	Standard App created only for demonstration purposes. Lack of innovation, creativity and special features. Low or no potential for impact.

	High degree of creativity in the design and features.	Problem with high significance. Good combination of unique design and features.	Some special features are incorporated.	
Teamwork	A clear decision-making procedure is formally established by the group, a document formalizes the roles and contributions/ideas given by each member of the group. Goals are well established, priorities are well documented and organized. All members respect each other, conflicts are resolved with open dialogue and compromise (well documented). A conflict resolution approach is documented and well defined. Disputes are described along with their outcomes.	Communication has worked well within the group and you have been able to adapt to a situation that has arisen. A procedure for making decisions is informally established by the group. Goals are clear and achievable. Priorities are clearly documented. All members respect each other, some conflicts are resolved with open dialogue and documented. Interactions between group members are documented. A conflict resolution approach is documented. Disputes are described along with their outcomes.	Tasks have been well divided with each member completing a significant part of the project (Git history and minutes). A procedure for making decisions exists but it is not clear. Goals are unclear, too general or unachievable. Some members did not feel free to contribute, ask questions, or share ideas. Interaction between group members is limited and not documented. A superficial conflict resolution approach is documented. Disputes are described along with their outcomes.	Cannot find (at least 3) meeting minutes. One person team (by Git history). Decisions are made by individuals. The group atmosphere is competitive and individualistic. Conflicts cannot be resolved between group members. Low interaction between group members. No conflict resolution approach is documented.

*Any images, or other assets that you copy from the Web you must attribute where you obtained them from. This must be added to your statement of originality. Ideally, you should only use assets that you have the right to copy, such as ones you create yourself, are in the public domain, or under a creative commons licence. With the statement of originality, you are safe and will be assessed only based on work done by you and your group.

**If you want to have your assignment remarked (appeal), you need to explain with reference to the rubrics why your original mark was wrong.

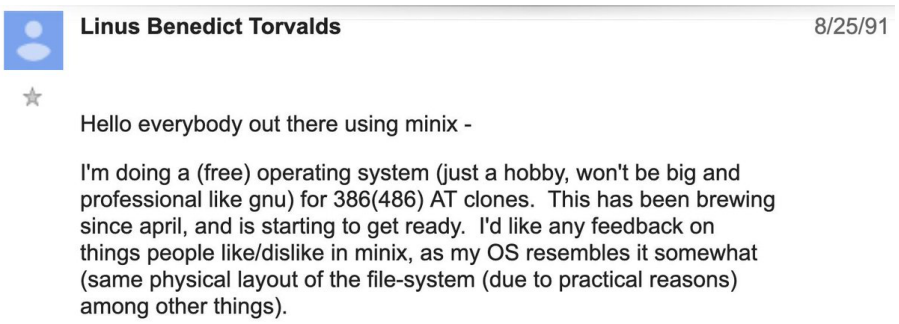
Other relevant information

You may use any version of Android Studio and Android SDK. But make sure your app code can be compiled by the tutors. It is recommended that you develop your app on Android Studio emulator, and then you can demo your app via a sharing computer screen. The app must be developed using Java.

There is flexibility for the server part *(if you want to implement it)* using different options, for example, PHP, Google Firebase, or third-party services. Only for the project, you can use any version of JUnit.

There will be no restriction on external libraries. Make sure that the external libraries are clearly referenced in your documentation and report. Only remember that we will evaluate what your group did, this is very important to understand!

Last but not least, here is what Linus Torvalds said about his project in 1991:



Linus Benedict Torvalds 8/25/91

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

It is time to code! Have fun!