

Group assignment document but with a contents page

that's it

Contents

Introduction	1
The app	2
Part 1: basic app	2
Part 2; additional features	3
Improved search	3
UI design and testing	3
Greater data usage, handling and sophistication	3
User interactivity	4
Privacy	4
Creating processes	5
Resource management	5
Peer to peer messaging	5
Firebase integration	6
Suggest a new feature	6
Surprise feature info	7
Checkpoints	7
FAQ	7
General further info	10
Working remotely	10
Number of group members	10
Assessment process	11
Report	11
Documentation of features	13
Project submission	13
Demonstration	14
Mark calculation	14
Due data and late submissions	14
Originality	15
Assessment rubric	15
Other relevant information	18

COMP2100/6442 Group Project | App 4 Social Good

v.01 - Project Description released (28.03.2022)

Assessment weight: This project is worth 30% of your overall mark.

Code due date: Thursday, Week 11 at 11h59 pm (last commit for the code)

Report due date: Friday, Week 11 at 11h59 pm (last commit for the report)

Presentation due date: Friday, Week 11 at 11h59 pm (slides uploaded to Office365)

Group Presentation: Monday, Week 12, at 4 pm

No late submission allowed

Introduction

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 practice 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. Part of the objective is to gain experience in working in a team.

This semester the project involves the development of an App for Social Good using Android/Java. Mobile phones are part of our everyday lives and we use them for sharing information, keeping in touch with family and friends, keeping up-to-date and even for social activism, creating awareness, funding for important social causes, education and charity. Besides that, the development of this kind of 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 impactful application. Although the theme for your 'App 4 Social Good' is open (B-creative!), there are some basic requirements your app must implement.

*** Note that we do not expect a fully operational app, simplifications can be made given the limited time to implement your app. ***

The App

Part 1: Basic App (30 marks)

In the first part of the assignment, you must create an app (with generated APK) which allows users to: (i) login, (ii) visualise and (iii) search for information within the app.

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. Use the report to explicitly link the features your group implemented to the requirements listed.

1. Users must be able to login (not necessarily sign up). (up to 3 marks)
2. Users must be able to load data/information (from file(s) or Firebase) and visualise it (e.g. on a list or a timeline activity). (up to 10 marks)
3. Users must be able to search for information on your app. Depending on your app theme, you may want to allow users to search for other users, data (e.g. text), tags (e.g. #buildingABetterWorldWithCOMP2100). The search functionality must make use of a tokenizer and parser with a formal 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 2,500 valid data instances. The data file must be used to feed your app simulating a data stream. For example, every x seconds, a new item is read from a file. An item can be an action (e.g. like, follow, support, post, new activities, 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 tree, AVL tree, B-Tree) for organising, 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, etc).
2. Your app shall implement at least three design patterns covered in the course.
3. Your app shall retrieve data from a local file (JSON, XML or Bespoken).

Can I use pre-made solutions (e.g. databases, APIs, etc)?

You can and should, software reuse is part of software construction. However, there are some compulsory items (e.g. tree data structure), and we expect you to implement them. So, as long as the pre-made solutions do not impact the core features, there is no problem with using them.

We remind you that we will assess what you have implemented, not what others have implemented. All used third-party code/library/solution must be referenced (see the Originality Section).

Part 2: Additional 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 classified by level of difficulty (easy to hard). 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 8 features in total, with at least 4 features classified as 'easy', 3 features classified as 'medium', and 1 feature classified as 'hard'.

Can implementing a higher level difficulty feature replace the requirement for a lower level difficulty feature?

Yes! Note that this is optional and we will not award you extra marks for that.

Note: Implementing a set of lower level difficulty features does not fulfil the requirement for a higher level difficulty feature.

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 Bspoken). (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 some information relevant to your app). (easy)
4. Report viewer. Provide users with the ability to see a report of interactions with your app (e.g., total views, total likes, total raised for a campaign, etc, in a graphical manner. (medium)
5. Deletion method of either a Red-Black Tree, AVL tree or B-Tree data structure. The deletion of nodes must serve a purpose within your application. (hard)
 - a. Note that this advanced feature will only be considered if the chosen tree is the most suitable data structure (out of the data structures covered in this course) for the App you are developing. Note that the RB-Tree deletion is not covered in lectures, this is optional (see deletion algorithm in the references of the data structure lecture).

Can I use the lab code?

Yes, you can, ***but reference and show us what you've improved in the labs code.*** There are several ways to improve the labs code. Remember that we will only assess your code, not ours. Only significant improvements can count towards completion of a feature.

User Interactivity

1. The ability to micro-interact with items in your app (e.g. like/dislike/support/report a post/message/event) [stored in-memory]. (easy)
2. The ability to reshare a message/event from another user/organisation (similar to 'retweet' on Twitter) [stored in-memory]. (easy)
3. The ability to 'follow' users, events, movements, hashtags, topics. There must be a section specifically dedicated to 'things' followed. [stored in-memory] (medium)
4. The ability to send notifications based on different types of interactions (posts, likes, follows, abuse reports, etc). A notification must be sent only after a predetermined number of interactions are set (≥ 2 interactions [e.g., one like and one follow or two likes 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 an item (e.g., launch a social cause, like an item, post a comment, etc). (medium)

Privacy

1. Users may send requests which are then accepted or denied by another user or organisation (e.g., a request to follow an event, a person or organisation). (easy)
2. A user can only see a profile/event that is Public (consider that there are at least two types of profiles: public and private). (easy)
3. A user can only follow/support someone/something who shares at least one mutual connection based on privacy settings. (medium)

4. A user can only visualise actions from someone who has set their actions visibility to public. (easy)
5. Provide users with the ability to 'block' things. Things (e.g., events, users, messages containing abusive language, etc) shall not be visible to the user who blocked that activity. (medium)

Creating Processes

1. Your app may implement a chain of action/steps (at least 3 steps) to follow up on a process required by your app's theme (e.g., an event to be active requires n supporters (step 1), after reaching the target number of supporters, the process moves on to the next step (step 2) which can be the execution of the event; and finally, after the execution, a report of the event is presented to all supporters (step 3)). (hard)
2. Process visualisation. Your app may implement a graphical element to visualise the progress of a process/event. (easy)
3. Your app may allow users to micro-interact and/or add messages to each step of the process (easy).
4. Only users with permission (e.g. admin users, users who created the process, etc) can view messages in the processes. (easy)

Resource Management

1. Transfer resources from one account to another. Each user/organisation may have the ability to transfer and receive resources from others. (easy)
2. Log transfers. All transfers may be logged and reported (textually or graphically) for transparency. (easy)
3. Transfer approval. Transfers of specific resources may require approval from an organisation or a user. Your app may implement an approval mechanism where after a transfer is made, an organisation or user receives a notification to approve the transfer. If the transfer is not accepted or takes long (e.g., 2 days to be approved), then the resource must be transferred back to the original user/organisation. (medium)
 - a. The transfer may require approval from multiple stakeholders (e.g., two or more users or organizations). If implemented this way, the difficulty level of this feature becomes 'hard'.

Peer to Peer Messaging

1. Provide users with the ability to message each other or an institution directly (e.g., a user can message an event/movement that is managed by another user). (hard)
2. Provide users with the ability to 'block' users. Preventing them from directly messaging them. (medium)
3. 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, if we support the same social cause/movement/event). (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 replaces the requirement to retrieve data from a local file) (medium)
3. Using Firebase or another remote database to store user information and having the app updated as the remote database is updated without restarting the application. E.g. User A makes a transfer, user B on a separate instance of the application sees user A's transfer appear on their app instance without restarting their application. (very hard)

Suggest a new feature | Voice your 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 (only features following this template will be considered):

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.

Suggested Difficulty Level: 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 7 and week 10, we will have checkpoints. You must participate in your lab session and show your tutor what stage of development you are in (only ONE member of the group has to present). For the first checkpoint, we expect you to have at least a schedule of your development activities, the roles of each member described, and at least one meeting minute besides an initial code structure. For the second checkpoint, we expect your app to be almost fully implemented and tested. You also need to show your report and the completed sections. Your tutor will review your schedule and will provide quick feedback. The checkpoints are 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. Every time you do not meet a 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).

Frequently Asked Questions

What kind of app should we develop?

It can go from a simple app that helps our community avoid [Magpie Swoops](#) to apps that volunteers lend their 'eyes' to solve tasks such as the [Be My Eyes](#) app. Other examples are the United Nations World Food Program's ShareTheMeal app; Charity Miles and Orange Harp. Think of a social cause you'd like to support and create your app. It can be in any area: Education, Environment, Energy, Infrastructure, etc. The impact can be local, think about our ANU Community, or Global, think about recent global issues. 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 ideas are also good.

Why and how to simulate a data stream?

To simplify the development of the app 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. Note that there is a Wiki article written by a former student that may be helpful for this task.

How should I develop the search mechanism?

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:

Search: #betterEducation and supporters > 1000 and @bernardo

In this case the user is looking for social causes annotated with the tag "betterEducation", have more than 1000 supporters and is authored by the user "bernardo".

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 representing social causes (not necessarily the best format, you must decide the

best format for your app, we will just evaluate it). This format considers only the total number of supporters and not who the supporters are.

```
...
<causes>
    <cause id="143" author="United Nations" media="../logo.jpg">
        <tag>Refugees</tag>
        <description>Support Refugees...</description>
        <Supporters count="100"/>
        <Title>The Refugee Cause</Title>
        ...
    </cause>
    ...
</causes>
...
```

*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 search for a user or a social cause. 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. Choose your teammates carefully. Be upfront about your expectations and agree on that with your teammates. 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 number of members per group?

Each group should consist of **4 students** (in exceptional cases, we will accept groups of 3 or 5). Groups can contain a mix of undergraduate and master students and students from different labs. 3-4 person groups will be assessed in the same way. **5-person groups must implement additional features: +2 easy, +1 medium, and +1 hard.**

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

How will projects be assessed?

1. Everyone in the group must implement and commit a non-trivial amount of 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 an extra [10,50]% 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 (via your provided APK), 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.

Note that if it is not in your report, we may not consider the feature, data structure, design pattern, etc, as completed. It is your responsibility to report everything that was implemented in your project. We are not playing treasure hunt and appeals will be rejected if what was implemented is not explicitly described in your report.

Therefore, along with the actual implementation, you will be required to produce a detailed 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
- Summary of individual contributions
 - This section must contain each individual contribution to the project including classes, methods and lines of code implemented by each team member.
 - Each team member is responsible for writing their own subsection.

- A generic summary will not be acceptable and may result in team members losing a significant amount of marks.
- An app summary with screenshots
- A design summary page (include justification for decisions made, diagrams, etc)
 - Details about the parser (describe the formal 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 4)
 - Your first meeting should develop a clear plan of how the work will be divided, documented in meeting minutes.
 - Right after every meeting (max 2 days after), 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:*
 - *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).

For example:

....

Feature Category: Privacy

Implemented features:

1. Privacy X. Description of the feature (easy)
 - a. Class X, methods Z, Y, Lines of code: 10-100
 - b. Class Y, methods K, L, M, Lines of code: 35-150
2. ...

Feature Category: Firebase Integration

Implemented features:

1. Use Firebase to implement user Authentication/Authorisation. (easy)
 - a. Class A: methods A, B, C, lines of code: whole file
2. ...

...

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.

You must bundle your app into a standalone APK (included in the GitLab repo) that can be loaded and execute correctly on an AVD.

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.

Use your own GitLab user (UID) to commit your contributions. Add this information to your report.

Group Project Demonstration

In week 12 there will be a hybrid group demonstration during our lecture time. 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.

OVERALL MARK CALCULATION

The overall mark will be calculated for each student as:

$$\text{Final Group Project (FGP)} = \text{IF} * \frac{2}{3} + \text{IC} * \frac{1}{3}$$

where IF corresponds to the marks obtained by implementing all features and IC corresponds to the contributions reported by each student in the report. IF is worth $\frac{2}{3}$ of the overall marks and IC is worth the remaining $\frac{1}{3}$ of the overall marks. Both IF and IC are calculated out of 100. Read the "*How will projects be assessed?*" section for more details on penalties for low or now contribution.

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.

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's 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 for this course. 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
Data Structures	<p>Chosen data structures well suited to subsequent usage, leading to efficient code in terms of both programmer and computer time.</p> <p>The chosen data structure can grow as the input grows without concern for</p>	<p>Data structures well suited for subsequent usage with minimal conversion or transformation required.</p>	<p>Satisfactory choice of data structures leading to inefficiency or repeated need to convert data structures or values to solve tasks.</p>	<p>Unsuitable data structures chosen leading to significant inefficiency or inability to solve tasks.</p> <p>Data structures loaded in each task or different data structures used for each task.</p>

	memory (scalability).			
Code Quality and Organization	<p>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.</p> <p>Code is very well organized, using generics and inheritance as appropriate.</p> <p>Construction for reuse. Easy to extend.</p> <p>The overall program structure makes the code easy to follow.</p>	<p>Good Documentation, naming and style are complete, consistent and appropriate. Code is well commented and code reviewers can partially understand the code.</p> <p>Good code organization with appropriate use of inheritance.</p> <p>(mostly) Construction for reuse. Easy to extend.</p> <p>The overall program structure makes the code easy to follow.</p>	<p>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.</p> <p>Alternatively, more significant shortcomings in one aspect.</p> <p>Inconsistent organization.</p> <p>Repeated code.</p> <p>Construction with Reuse. Code can be partially extendable.</p>	<p>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.</p> <p>Poor code organization.</p> <p>Code is very difficult to follow.</p> <p>(mostly) Construction with Reuse. Difficult to extend.</p> <p>Missing APK.</p>
Report	<p>Report is well organized, well presented, clear and concise. Project decisions are well detailed with examples and discussion.</p> <p>It also presents code and analysis. UML diagrams are presented.</p> <p>A clear and detailed testing summary is provided.</p> <p>References are provided.</p>	<p>Report is well organized and presented.</p> <p>Project decisions are detailed with suitable examples and discussion.</p> <p>It partly presents code and analysis. Some UML diagrams are presented.</p> <p>A clear testing summary is provided.</p> <p>References are provided.</p>	<p>Report is organized and presented.</p> <p>Project decisions are not clear but present some examples and discussion.</p> <p>It does not present code and analysis.</p> <p>References are partially provided.</p>	<p>Limited discussion or understanding of issues. Poor decisions made and choice of examples. Poor organization, or lack of clarity makes the report hard to follow.</p> <p>References are not provided.</p>
Testing	<p>JUnit coverage test achieves at least 70% of code (without UI).</p> <p>Clear evidence of testing to verify correctness and robustness. Exceptions and error cases are checked.</p> <p>Clear evidence of Unit and Integration tests.</p>	<p>JUnit coverage test achieves at least 60% of code (without UI).</p> <p>Some evidence of testing to verify correctness and robustness.</p> <p>Clear evidence of boundary and normal functioning tests.</p> <p>Some evidence of Unit and Integration tests.</p>	<p>Repeatable unit testing is performed on the majority of the project.</p> <p>Appropriate use of Test Suites and Parameterized tests.</p>	<p>Tests are not well designed (random tests).</p> <p>Minimal or no test cases can be found.</p>

User Interface	<p>User interface is intuitive and can be easily used without guidance.</p> <p>Use of consistent theme and style.</p> <p>Feedback is provided to users based on interactions.</p> <p>UI is responsive (adapt to different screen sizes and orientations).</p> <p>Stylish and friendly look and feel.</p>	<p>User interface is mostly intuitive and can be used with little guidance.</p> <p>Use of consistent theme and style.</p> <p>Some feedback is provided to users based on interactions.</p> <p>UI is mostly responsive (reasonably adapt to different screen sizes and orientations)</p> <p>Friendly look and feel.</p>	<p>Standard user interface and not very intuitive (needs some guidance).</p> <p>Use of theme and style are not consistent. Navigation is not clear.</p> <p>Lack of feedback to users based on interactions.</p> <p>UI is responsive and provides only screen orientation mode.</p>	<p>Standard user interface and unintuitive (needs guidance).</p> <p>Inconsistent use of theme and styles. Navigation is not clear.</p> <p>Lack of useful feedback to users based on interactions. No guidance to users.</p> <p>UI is not responsive and provides only one screen orientation mode.</p>
Creativity / Uniqueness / Specific Theme	<p>App is innovative and can be applied in a real world scenario.</p> <p>Problem statement is clearly defined and has the potential to impact multiple beneficiaries.</p> <p>Unsolved problem with high significance.</p> <p>High degree of creativity in the design and features.</p>	<p>App is innovative and can be applied in a real world scenario to a limited number of beneficiaries.</p> <p>Problem statement is clear and has the potential to impact a limited number of beneficiaries.</p> <p>Problem with high significance.</p> <p>Good combination of unique design and features.</p>	<p>The developed app can be applied in a real world scenario to a limited number of beneficiaries.</p> <p>Problem statement is clear but with low potential for impact.</p> <p>Some special features are incorporated.</p>	<p>Standard App created only for demonstration purposes.</p> <p>Lack of innovation, creativity and special features.</p> <p>Low or no potential for impact.</p>
Teamwork	<p>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.</p> <p>All members respect each other, conflicts are resolved with open dialogue and</p>	<p>Communication has worked well within the group and you have been able to adapt to a situation that has arisen.</p> <p>A procedure for making decisions is informally established by the group.</p> <p>Goals are clear and achievable. Priorities are clearly documented.</p> <p>All members respect each other, some conflicts are resolved with open</p>	<p>Tasks have been well divided with each member completing a significant part of the project (Git history and minutes).</p> <p>A procedure for making decisions exists but it is not clear.</p> <p>Goals are unclear, too general or unachievable.</p> <p>Some members did not feel free to contribute, ask questions, or share ideas. Interaction</p>	<p>Cannot find (at least 3) meeting minutes. One person team (by Git history). Decisions are made by individuals.</p> <p>The group atmosphere is competitive and individualistic. Conflicts cannot be resolved between group members. Low interaction between group members.</p> <p>No conflict resolution approach is documented.</p>

	<p>compromise (well documented).</p> <p>A conflict resolution approach is documented and well defined. Disputes are described along with their outcomes.</p>	<p>dialogue and documented.</p> <p>Interactions between group members are documented.</p> <p>A conflict resolution approach is documented. Disputes are described along with their outcomes.</p>	<p>between group members is limited and not documented.</p> <p>A superficial conflict resolution approach is documented. Disputes are described along with their outcomes.</p>	
--	--	--	--	--

*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!