GroceryListManager:
An App for Managing Grocery Lists
(Deliverable 4–Transition)

Background

Brad and Janet are tired of doing grocery lists with pencil and paper and are unhappy with each and every existing app for managing grocery lists. Because they are rich and bored, they decide to hire a team of programmers to develop *GroceryListManager*, an Android app for managing grocery lists that works exactly the way they want it. Your team has been contacted by Brad and Janet, who provided the set of requirements listed below and want to see a possible design for the system before moving forward and formally hiring your team. Because Brad and Janet studied computer science in college and are familiar with UML, they want the design to be represented using a UML class diagram. Luckily, the members of your team already have experience designing systems of this kind, so the team should be able to hit the ground running and produce a good design quickly.

Goals

- Develop an Android app that implements the grocery list manager that Brad and Janet envisioned (and that your team already designed).
- Get experience with (a simplified version of) the Unified Software Process.

Requirements

See the requirements in Assignment 5.

Deliverables

- D1 – *Software design* [done]
- D2 – *Inception and Elaboration* [done]
- D3 – *Construction* (see corresponding lesson): [done]
- D4 – *Transition* (see corresponding lesson): **[due in one week]**
    - Possibly revised versions of earlier documents (based on your better understanding of the system).
    - Test plan with final results.
    - Final version of the app.

Notes (important–make sure to read carefully)

1. For Section 3.1 of the design document (Class Diagram), you should simply use the design that your team created for Deliverable 1.
2. Although you will have to submit code for this deliverable, please keep in mind that what we said for the previous deliverables still holds: Documents (use case model, design document, …) are valued just as much as the code, and sometimes more, so having a great app with a set of poorly prepared documents will likely result in a low grade.
3. If in a later deliverable you need to update any of the documents in an earlier deliverable based on your better understanding of the system, you can do so, as explicitly stated in the description of the deliverables. (We will also do our best to give you feedback if you have specific questions, but we cannot guarantee that we will be able to do it in a timely fashion.) If you do so, please make sure to add a version number at the beginning of the document and provide a concise description, as indicated in the document templates.
4. To clarify the previous point, please note that this is not an invitation to procrastinate. It is true that we will grade the various artifacts submitted based on their latest version. However, **teams will be penalized if they simply submit "placeholder documents"**. In other words, we are perfectly fine with the documents evolving throughout the project, and will grade their latest versions, but the documents have to be reasonable at every deliverable.
5. If you identify incompleteness or inconsistency issues in the requirements, feel free to either ask for clarifications or make **explicit** assumptions and develop the system accordingly.
6. Once you satisfy all of the requirements, feel free to be creative and extend the functionality of the app if you feel like it and believe you can afford it. This is not required, of course.
7. Automated test cases are always appreciated, and ultimately useful for you, but it is fine to have a list of test cases that can be run manually (as long as for each test case you describe how to run it, what inputs to provide, and its expected output). Whether manual or automated, make sure that your test cases adequately cover the functionality of the app. If you are keen to generate automated system test cases, you can try Barista, a tool developed at Georgia Tech for recording test cases and generating automated Espresso test cases based on these recordings.
8. Go to the directory "`GroupProject`" that you created for Deliverable 1 in the **team repo** that we assigned to you. Hereafter, we will refer to this directory as `<dir>` .
    a. The minimum API level for the app should be Level 21.
    b. Having a local DB for the grocery items is perfectly fine (and for simplicity that's what we recommend to do). You are free to implement the database any way you want, but we recommend the use of SQLite with the Room Persistence Library.
    c. The code of the app must be uploaded to the repository as a regular Android Studio project called `GLM`, whose root is directory `<dir>/GLM`. All classes should

be in package `edu.qc.seclass.glm`, and all the files needed to run your app must be uploaded to the repo. (To check this, we recommend that you clone your repo in a separate directory/workspace and run your project from there. Cloning your repo on a different machine would be even better, if you can do that.)

   d. In addition, to help with cases in which we may have problems compiling your app, **add an APK file for your app in directory** `<dir>/APK`.
   e. All other deliverables, including the user manual, must also be pushed to your repository, in directory `<dir>/Docs`.

9. As usual, after submitting a deliverable by pushing your changes to GitHub, the project manager (and only the project manager) must post on Blackboard the corresponding commit ID. **The other team members should submit the full name of the project manager on Blackboard, to avoid confusion and possible inconsistencies.**