Mobile Application Report

<u>Front-End Considerations</u>

- Option 1: React
  - Pros
    - Popular mobile-development framework
      - Many tutorials and community discussion online to make learning easier
      - Very common amongst developers (2nd according to StackOverflow 2019 survey)
    - Since we are creating a simple checkout price calculator with no complicated backend logic, React's library allows for easy creation of a responsive, clean UI.
    - Applications are very fast an responsive
    - Can also be used for web applications
  - Cons
    - Never worked with React so will have to learn from scratch
    - Very large library which can be daunting
- Option 2: XML
  - Pros
    - Integrated in the Android Studio IDE framework
      - Extensive tutorials and community discussion forums online for making mobile layouts with XML using Android studio
      - Android Studio has a UI provided for XML layout creation, making it quite simple to create front end layouts
    - Slight familiarity working with in the past
  - Cons
    - XML is considered as obsolete
    - XML syntax is redundant and verbose

- Option 3: Angular
    - Pros
        - Extremely popular among software engineers
            - Lots of online learning resources and tutorials available
        - Angular's Model-View-Controller architecture allows the front end to be easily synchronized with the backend
        - Can also be used for web applications
        - Easy to test
            - Angular makes use of components, which encourages dependency injection, making testing simple and easy
    - Cons
        - Hard to pick up
            - Angular syntax is verbose and relatively complicated
            - Angular requires an understanding of many topics to before being able to use effectively using. For example, one has to be familiar with dependency injection, components etc.

We chose to use XML for the front end due to its integration in the Android Studio IDE. Android studio makes use of model-view-adapter architecture. Using XML format is desirable in the android studio MVA context, since it allows for easy reference and manipulation from backend controllers.

<u>Back-End Considerations</u>

- Option 1: React (NodeJS/Javascript)
    - Pros
        - Ease of use, very simple to learn especially with no complicated backend architecture.
        - Due to popularity, many helpful guides and solutions can be found online for ease of learning.
        - Easy testing with Jest.
    - Cons

        - Not fully secure, versatile backend
- Option 2: Java (with Android Studio IDE)
    - Pros
        - Previous experience working with Java from CSC207
        - Java allows for Object Oriented Programming (OOP). Allows for easy manipulation between objects in our calculator app
        - Very popular. Lots of tutorials and online forums which allows easy learning
    - Cons
        - Poor performance due to slow compilation and build time
        - Java syntax is verbose and complicated
        - Mainly use for android development but there are ways to make it work for ios development as well
- Option 3: Swift
    - Pros
        - Swift has great performance. The language is created with a focus on speed and performance.
        - Regarded as the 6th most loved programming languages in the StackOverflow 2019 survey.
        - Easily scalable
    - Cons
        - Relatively small number of native libraries and online learning resources
        - Mainly use for ios development but there are ways to make it work for android development as well

We choose to use Java as our backend development language with the IDE Android studio. My experience in programming in Java as well as the popularity of using Java for android app development allowed me to effectively learn to create a visually pleasing application within the two-week timeframe of the assignment.

Database Consideration

Our goal is to create a simple checkout calculator that allows users to calculate the prices of provided items by adding them to the cart. Due to the motivation of creating our application, we found it unnecessary to implement a database. As a result, no database was used in our project and grocery item objects are created in the MainActivity module of our application. However, our mobile application is built with a solid model-view-adapter framework grounded in the principles of object oriented programming. As a result, it is reasonably simple to integrate a database into our application if someone were to create an ecommerce website using our application as the starting framework.

Testing

Android studio allows for easy testing for Unit Tests (using Junit and Mockito), Instrumentation tests (also using Junit and Mockito), as well as UI tests (using Expresso). Unfortunately, I could not get the testing component of our project to work as I could not solve the issue of Android studio being unable to recognize and make use of testing dependencies. However, I included a sample test for MainActivity.java to show that, if the problem with gradle recognizing and making use of testing dependencies were to be solved, testing is easily implementable with our chosen mobile tech stack.

Final Application

The final application has 4 main features. Adding and removing items to a cart system, price calculation of products in the cart, and the ability to apply discounts. Each of these features very easy to use and intuitive for the user (as outlined in the README.md file). Additionally, our mobile application is responsive and aesthetically pleasing. We discussed extensively whether to implement an ordering system, due to the nature of this assignment, to create a shopping-cart calculator, determined it to be unnecessary. However, the mobile application is designed in a way that allows for easy extension of any of these potential future development paths.