**Project Part 4**

**Team:** Andrew Gordon, Tommy Hoffman, Connor McGuinness, Daniel Zurawksi

**Title:** What to Wear Weather

**Implemented Features/Requirements and Final Class Diagram**

Features:

* Fetches user’s location using GPS.
* Retrieves weather forecast based on user’s location.
* Makes clothing suggestions based on user’s gender and weather forecast.

Requirements:

* US-001: As a user, I want to see daily clothes recommendations when opening the app.
* FN-001: On app startup, current GPS coordinates will be retrieved
* FN-003: App will use weather API to get weather at current GPS
* FN-004: Use a relational database (MYSQLlite) to hold user data (age, gender)
* NF-001: Coordinates will use the Android system’s fastest “best guess”
* NF-003: App will respond quickly and precisely to user touch/interaction

(updated class diagram goes here)

discuss benefits of designing before coding

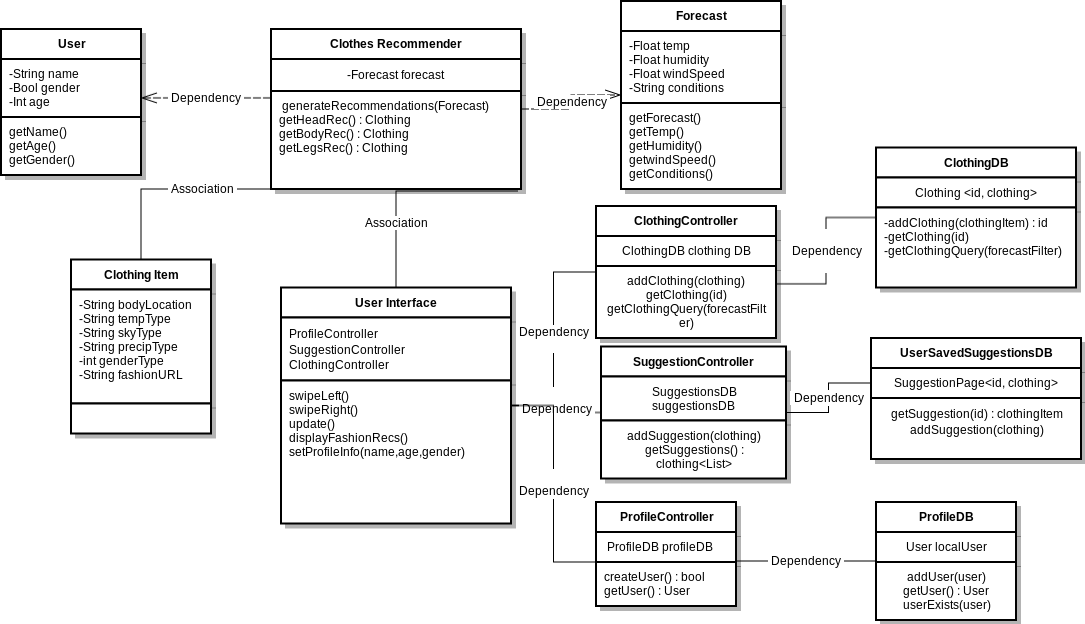
**Design Patterns**

A design pattern implemented in the final prototype of What to Wear Weather is **State**. The purpose of State is to allow an object to behave in different ways based on the object’s internal state. By fetching a forecast summary from a weather API, the object that contains all of the clothing suggestions will change its behavior. So based on the state of the forecast summary, clothing suggestions will change based on that state. State helped keep code length to a minimum and with readability.

One design pattern that was not utilized but could have been is **Observer**. Most methods and classes in our system talk only directly to each other instead of together. Observer would have been used to update information stored in the database, change the suggested clothing options, and change the forecast day and summary, all by the click of one button. Objects and methods would notify each other when one was changed, thus reducing repeated code and methods. In our system, methods only update each other when specifically called upon. Therefore, the database could be out of sync with a changing forecast. Fortunately, we avoided synchronous problems by making sure that classes and objects are explicitly updated when they need to be.

Instead of **Observer**, the group decided to implement **Singleton** in our system. Since our application is only running on one operating system, there is only instances of system objects. Therefore, all of our methods access the system objects during runtime and do not need to account for third party systems. This allowed us to focus on implementation exclusively for Android OS. Without having to worry about multiple platforms, the group was able to write more robust code for Android. There wasn’t a need for base functionality across platforms, so the system’s code remains organized and clear.

**Comparison of Part 2 Class Diagram and Final Class Diagram**

include part 2 class diagram and point out the necessary changes. You can answer why you didn't realize you needed things during the design and/or how next time you will be able to make a better design after this learning experience.

**Lessons Learned About Analysis and Design**

The process of analysis and design is not something to be rushed through. Although we diligently prepared for implementation, we did not account for timing well. During the design phase of our system, we felt that the aspects we thought would be easy to implement, proved to be difficult.

One such aspect is requesting/sending information to and from the database. The system relied heavily on this functionality yet it was not very clearly defined in our design phase. We tied it into multiple features like saving gender preferences and storing clothing suggestions. As a group we simply thought that database manipulation would be easier than expected to implement. Since we had to spend more time on this functionality, other features had to be delayed. Therefore, one lesson learned is to plan the timing of implementing features and to expect errors. Without accounting for errors, the implementing phase will take longer to finish. By expecting errors during the design phase, implementation will be easier to handle because the group has already figured out what error is likely causing problems.

Another important lesson the group learned about analysis and design is project consistency. By defining key components like SDK’s, API’s, platforms, coding languages, and databases early in the design phase, it helps to stick to these choices through implementation. A small example can be seen in our UI Mockups, that were created using Xcode. Although, these mockups did provide a good reference for how our application should look, it was not consistent with how Android applications look. Since we had defined it would be an Android application, creating UI Mockups with Android Studio would have been a more consistent choice. By keeping project consistency, implementation is more straightforward and organized. Fortunately, the group remained consistent on most of the design choices during implementation. Therefore, it was easier to stay focused on specific errors and remain organized even when working on different parts of the system at the same time.

Overall, planning ahead for errors and maintaining project consistency throughout all phases of creating the system are important lessons learned by the group. By not accounting for the amount of time debugging a specific error would take, the group had to “brute force” the error until the application would work. Also, by not keeping certain aspects of the design phase consistent with the implementation phase, the group had to create new parts of the project different from the original plan. However, these fallbacks were pushed through in order to create a functional system. As a group, we are going to remember these design pitfalls when working on a new project. Analysis and design is a learned skill that is best developed from experience.