***Pocket Closet App’s Implementation Report***

***Prepared By***

***Logan Pensa,***

***Adrian Gallant,***

***Aarron Banson,***

***Hemant Maheshwari,***

***Daniel Wilson***

**Table of Contents**

**Introduction**……………………………………………………………………………………………………………………………...3

**Overview**…………………………………………………………………………………………………………………………………..4

**Code Layout**……………………………………………………………………………………………………………………………...5

**Design Patterns**………………………………………………………………………………………………………………………….5

**Diagrams**…………………………………………………………………………………………………………………………………...9

***Introduction***

With the younger generations in high school and college, there is a lot of pressure to maintain a sense of fashion by their peers and more so, social media. Social media pressures the younger generation by trying to look good and to some level, it is not wrong. Going to a job interview, a picnic or dressing for a concert require a different sense of fashion in order to dress properly. Though this app can certainly help out with such situations, there are also people who are looking to go beyond and enhance their sense of fashion. Bearing both types of audiences in mind, this app will help both sets of people to enhance their sense of fashion no matter how good or bad they are with the concept. Pocket Closet is an app that allows its users to mix and max outfits from the user’s current wardrobe and with products they find from the stores that market their products to them. This app is meant to be sold to stores first and the stores will make necessary modifications and add information about their current and upcoming products which they can use to market their products to users. Once the app has been tweaked to the store’s liking, the store will be able to put their products on posts and the store or will market their posts (with their products) to their customers. Should their customers take the option to use the app, the customer will create an account and their direct feed/home page will display posts about products advertised by the store. Over time, as the user follows more people and gains more followers and follows more stores, the direct feed/home page will display different posts and that’s how the user will develop their sense of fashion (or stay ahead of the fashion curve).

***Overview***

The Pocket Closet app is a mobile application for use by stores and regular users. Regular users will have access to most of the functions of Pocket Closet while stores will have access to all of the functionalities of the mobile application. Regular users will have access to the Home/Direct Feed page, the Search page, the Create Outfit page, the Profile page, and the Settings page. The only functionality that regular users will not have access to is the post function. As for this current version of the app, the post function will allow stores to post products catered uniquely to the regular user. The Home/Direct Feed page is an ongoing feed of posts from followers and stores that the user is following. The Search page is where the user can search up other users and stores to follow. The Create Outfit Page will allow the user to assemble their virtual wardrobe based on pictures taken from their physical closet, outfits from other users and other clothing products from the store. The Profile page is where key credentials are listed and are able to change their profile picture as needed. The Settings page allows the user to modify key credentials as they need to and these changes will be updated on the database end as well. The rest of the document describes the software and how the resulting designs were created for the purpose of the Pocket Closet app.

***Code Package Layout***

The Pocket Closet app consists of three main parts. The three parts are the Mobile portion of the app, the Web API and the Database. The collection of these coded sections are set up in a manner where the software can be maintained by others and is easy to understand. It is also possible to update and patch the software for any issues that crop up in the process.

***Design Patterns***

**Singleton Design Pattern**

With the singleton design pattern, implementing such a pattern ensures that a class has only one instance and provides a global point of access to it. Typically a singleton design pattern can be recognized when static methods are present and return the same object. While there are no static methods present, the same object is being returned in a manner of speaking. Bearing the purpose of the Settings page which is to update (change) key user credentials, the object has the same field types to return but the values of the fields are changed. For example, consider the object is User user and the method public void updateUserAccount() is called on (bear in mind that User user is being accessed globally). Also consider the fields of first name and last name (there are many fields but these two are sufficient to explain the point). These fields have an initial value set upon the creation of the account, and that will change when the User user object runs this method to change the value of the initial first name and last name values to new values for the first name and last name fields.

**Facade Design Pattern**

With the facade design pattern, implementing such a pattern hides the complexities and provides an interface for the client to use to access the system. Classes that need to access any type of module for certain functions, these classes would be able to use a high-level interface that would be defined by a facade that would manage how these classes are permitted access or denied access into the other parts of the program. The entirety of the Web API portion of the app functions using a Facade Design Pattern, and the RestAPICRUDService of the mobile portion of the app uses this design pattern. For example, using Program.cs from the Controllers code package, is where the main method is. The main method calls for CreateHostBuilder(args).Build().Run() which eventually calls for CreateHostBuilder to start webBuilder.UserStartup<Startup>() which accesses the Startup.cs. Startup eventually calls for other methods which eventually will call endpoints.MapControllers() and eventually calls for the sub-controller classes within the Controller code package. Essentially with this process, the classes do not repeatedly need to keep accessing other classes but rather allow the facade to forward the appropriate requests from the client, to the appropriate subsystems within the program. Facade objects simply allow for a simplified interface for the user and ultimately this cuts down on the complexity of the program significantly.

**Private Data Class Design Pattern**

With the Private Data Class design pattern, the focus of this design pattern is to restrict accessor or mutator access from other classes or objects. Essentially this will prevent other unnecessary pieces of data (that include fields or other objects from other classes) from tampering with data within the private class. For example, in the SettingsPage class, this class has two private objects. The objects are named, private UserController userController and private User user. With the “private” keyword, this prevents unauthorized access from other classes and the class that can perform method calls on these two objects is the SettingsPage class itself.

**Null Object Design Pattern**

With the Null Object design pattern, this pattern is designed in a way that sets default values to certain objects. For example, in the RestAPIService class, there is a method called checkuserAsync(User user). There is a line that refers to a default value in the line of code, return default(User). This method returns a default value for the User object and that default value is null. This is present in the else clause because this method is checking to see if the user exists, and if something went wrong on the web API’s end, then User with the default value of null is sent back.

**CLRUD Design Pattern**

With the CLRUD design pattern, this pattern was crucial for implementation of the CRUD Stored procedure on the Database end of the program. This pattern is based on the concept that each object in a database (tables, usually) has four basic operations: Creating data, Reading data, Updating data and Deleting data. The CLRUD design pattern encapsulates the CRUD design pattern into a single partial class per SQL object. For example, the create\_user stored procedure is a stored procedure that creates a new user based on a specific set of attributes. In the SQL object created in this procedure, the object’s attributes are username, password, first\_name, last\_name, email, user\_type, gender, and dob. Everytime this stored procedure is called on, the procedure will create an SQL object with these attributes (that have different values saved to them which depends on the user). Bearing in mind that the other stored procedures have different purposes, this design pattern is what allows the systematic creation of the CRUD stored procedure.

***Diagrams***

