**2b. Software Architecture Representation**

The following package diagram illustrates the decomposition of the software architecture:

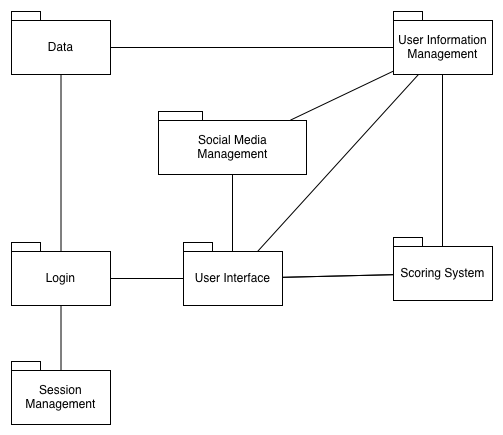


Figure 1: Package diagram for Healthy Eating Application

**Session Management Package**: a collection of classes associated with session management.

**Login Package:** a collection of classes related to gaining access to features available only to registered users.

**User Interface Package:** a collection of classes that enable visual interaction with the user.

**Social Media Management Package:** a collection of classes related to the interaction of the application with Facebook.

**User Information Management Package:** a collection of classes that handle user’s personal account data such as their food entries, status and friend list.

**Data:** a collection of classes that represent data objects used by the application; this package also includes any operations related to database interaction.

**Scoring System:** a collection of classes associated with evaluating user food logs and assigning appropriate scores. This package also contains operations linked to determining if the user needs prompts for healthier eating choices.

The architectural style used is the repository style. It is essential to describe each of the major components and connectors, as well as their interactions. Figure 2 illustrates the components and connectors present in the architecture.

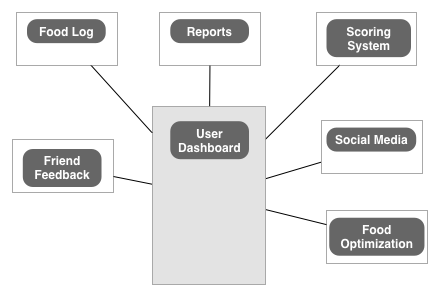


Figure 2: Components and connectors diagram for Healthy Eating Application

Description of Major Components

In accordance with the repository style there is one central data structure that represents the current state; for the Healthy Eating Application this is the User Dashboard. The user’s dashboard will contain a list of all the user’s food log entries for the current month (earlier entries can be seen by selecting a specific month and year). This component also has the option to display reports with graphical representation of user’s food related data for archiving. Furthermore, the user’s current score, and food optimization suggestions, friend’s feedback for logged entries and social media elements are to be displayed and accessed from the dashboard.

Moreover, there are several independent components that operate on this central data structure. The Food Log component process all food entry related data. It displays the first line of each entry on the user dashboard, with the option to view the entire entry if required.

In addition, there is the Reports component. This component is responsible for processing all the calorie information in the entries present in the dashboard for the current month. The data is converted into different graphs that offer the user insight into their eating habits and patterns.

Further, the Scoring System component analyzes the data entries and calculates the user’s current score accordingly.

The Social Media component is responsible for taking information reachable via the dashboard and displaying it on a relevant Facebook page as well as allowing the user to import certain application related data from Facebook.

Lastly, the food optimization component analyzes the user’s food entries and statistics and suggests better food alternatives, if necessary.

Description of Major Connectors

The connectors for the repository style architecture proposed are mainly procedure calls. For instance the Food Log component might issue a AddNewEntrySynopsis() procedure call to the User Dashboard. Similarly the User Dashboard can issue a GetFullEntry() procedure call to the Food Log component.

DO WE NEED TO GIVE EXAMPLES OF POSSIBLE PROCEDURE CALL FOR ALL COMPONENTS? OR IS THIS EXAMPLE ENOUGH?

**3a. Architectural Goals and Constraints**

The repository architectural style is a justified selection for the Healthy Eating Application. This is because the user’s dashboard is the central data structure. This is the case seeing as almost all major actions and activities affecting the application are displayed and accessed from there. What’s more, any updates or modifications are displayed in some form on the dashboard. Additionally, the application lent itself to having a collection of independent components that operate on the central data structure. This was so, since most computations that occur in the application result in some form of change or update to the user’s dashboard.

It is important to mention the architectural rational for creating the different components. Each component encapsulates processing and data associated with a particular functionality within the application. For instance, the Food Log component encapsulates data related to a log entry and any processing associated with it. The Report component encapsulates data associated with generated reports and the accompanying computations. The same can be said for the Scoring System, Social Media, Food Optimization and Friend Feedback components.

Moreover, the architectural rational for creating different connectors was... NOT SURE WHAT TO PUT HERE, CAN’T HAVE ANYTHING ELSE BUT PROCEDURE CALLS AS CONNECTORS... IF NOT THEN THERE IS NO RATIONAL TO MENTION.

**3b.**

Since there is a central data structure in the application, this gives the opportunity to have centralized management. In consequence, one can easily perform backups of essential data and focus mainly on the security of one central component. All of the aforementioned points increase the ease of maintaining the software. However, a downside to the chosen architecture is that data evolution might prove to be expensive. This is because the dashboard has limited space to show different type of data and displays. Therefore, the amount and type of data stored in the central data structure (in this case, the user’s dashboard) should be decided in the beginning, modifying this substantially in later stages would be difficult.

HOW DOES REPOSITORY STYLE AFFECT PERFORMANCE?

MENTIONED MAINTAINABILITY, DATA EVOLUTION..... WHAT OTHER QUALITY ASPECTS?

**4. Use Case View**

The relevant use cases for the application are shown in Figure 3 and 4. The use cases have been divided according to if the user is logged in or not. Each figure is followed by the corresponding use case descriptions.

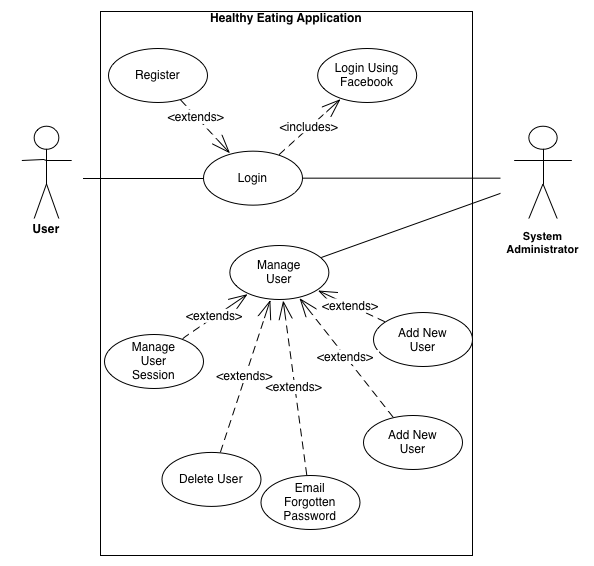
****

Figure 3: Use Cases concerned with logging into the application

Use Case Description

|  |  |
| --- | --- |
| Login | |
| **Participating Actor** |  |
| **Entry Condition** |  |
| **Exit Condition** |  |
| **Flow of Events** |  |
| **Exceptional Cases** |  |

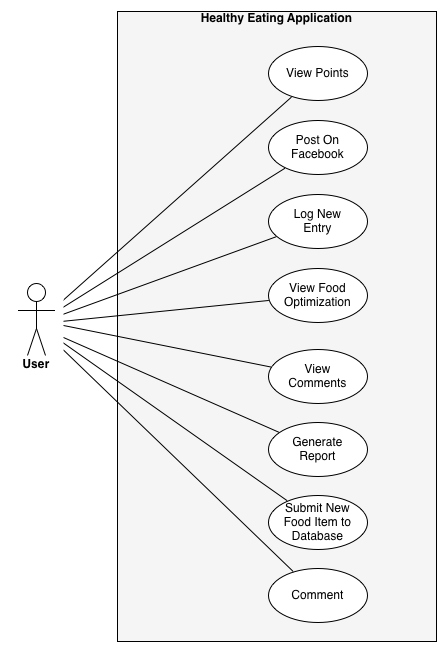
****

Figure 4: Use Cases concerned with when user is logged in to the application

Use Case Description

|  |  |
| --- | --- |
| View Points | |
| **Participating Actor** |  |
| **Entry Condition** |  |
| **Exit Condition** |  |
| **Flow of Events** |  |
| **Exceptional Cases** |  |