**2b. Software Architecture Representation**

This section illustrates the software architecture decomposition in terms of a package diagram.

High Level Decomposition

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

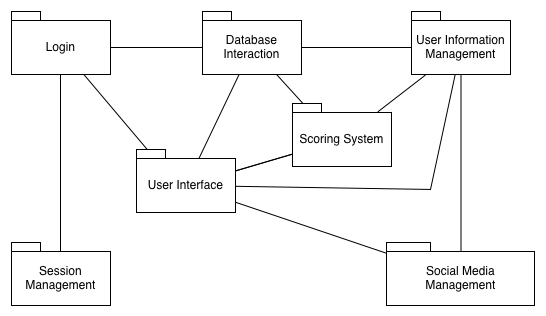


Figure 1: Package diagram for Healthy Eating Application

**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.

**Database Interaction**

This package is related to database interaction operations. The Healthy Eating Application uses a considerable amount of different types of queries to the database thus a separate package for this activity is warranted.

**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.

Second Level of Decomposition

A second level of decomposition, identifying the main components within the package, can be derived from Figure 1. This detailed decomposition highlights the major components present within the package. Note that components are units of computations and may be comprised of more than one class

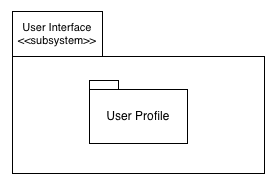


Figure 2: Decomposition of Data package

TODO: ADD OTHERS, showing the components mentioned below within the respective packages

Description of Major Components

**User Profile component:**

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 profile page. The user’s profile 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 profile.

Moreover, there are several independent components that operate on this central data structure.

**Food Log Management component**

The Food Log component process all food entry related data. It displays the first line of each entry on the user profile, with the option to view the entire entry if required. It is important to note that the user profile does not perform any action’s of its own, it simply acts as a central body of information.

**Report Management component:**

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

**Scoring System component**

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

**The Social Media Management component:**

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

**Food optimization component:**

This 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 Profile. Similarly the User Profile can issue a GetFullEntry() procedure call to the Food Log component.

TODO: NEED TO GIVE EXAMPLES OF POSSIBLE PROCEDURE CALL FOR ALL COMPONENTS

**3a. Architectural Goals and Constraints**

The architectural style used is the repository style. Figure 3 illustrates how some of the major components and connectors present fit in to the architectural style chosen.

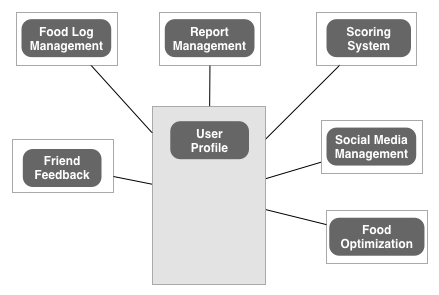


Figure 3: Components and connectors diagram for Healthy Eating Application

Justification of Style Selection

The repository architectural style is a justified selection for the Healthy Eating Application. This is because the user’s profile 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 profile. 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 profile.

Architectural Rationale for Components and Connectors

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.

In addition, the repository architecture is very efficient for sharing large amount of data. This is because all components have access to a common known data structure, which represents the central body of information.

However, a downside to the chosen architecture is that data evolution (defined in this report as integration of new type of information) might prove to be expensive. This is because the profile 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 profile) should be decided in the beginning, modifying this substantially in later stages would be difficult.

**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.

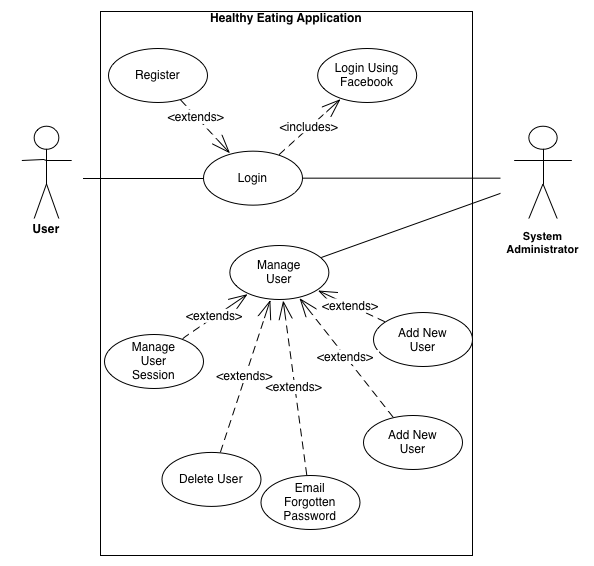
****

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** |  |

TODO: Add the rest, fill in tables

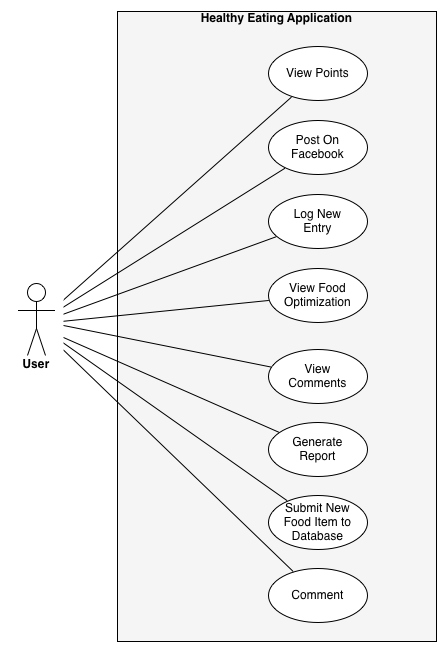
****

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** |  |

TODO: Add the rest, fill in tables