GROUP NAME \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Diet Manager V1 – Rubric

SWEN.383 SW Design Principles and Patterns

Final Java Solution for Diet Manager V1

|  |  |  |  |
| --- | --- | --- | --- |
| **Level** | **Tasks** | **Points** | **Score** |
| ☐ 1 | Load a foods.csv file with only basic foods and an empty log.csv file. User can see an empty log for today with default calories and weight. User can view the basic foods loaded.  TODO: Explain what you did here and where to find this feature in your code   * **What was done**: The application loads data from a **foods.csv** file containing basic food items and an empty **log.csv** file at startup. * **Where to find it**:   + The loading mechanism is implemented in the **CsvHandler** class within the **loadFoodData** and **loadLogData** methods​​.   + Initialization and loading occur in the **DietManagerModel** constructor, which calls **loadCsvData** to load foods and logs from CSV files​​.   . | (60) |  |
| ☐ 2 | Level 1 PLUS Select a basic food(s) for the daily intake. The selection is stored in the database (log.csv) the log view is updated with the dietary information about the nutrients consumed.   * TODO: Explain what you did here and where to find this feature in your code.  **What was done**: Users can select basic foods from a loaded list and add them to the daily log, which updates the nutrient information  displayed. * **Where to find it**:   + The **DietManagerView** class creates a user interface allowing selection of foods and addition to the log through a form and button​​.   + The **DietManagerController** handles the addition of food items to the daily log and updates the log CSV file via **addLogEntry** method​​. | +10 (70) |  |
| ☐ 3 | Level 2 PLUS Add new basic food(s) to the food database. This implies the ability to then add such basic food(s) to today's log as in level 2   * TODO: Explain what you did here and where to find this feature in your code.  **What was done**: Users can add new basic food items to the system, which then can be selected for daily intake. * **Where to find it**:   + In the **DietManagerView**, there's a form for entering new basic food details​​.   + The **addBasicFood** method in **DietManagerController** adds the new food to the model and updates the CSV files​​. | +5 (75) |  |
| ☐ 4 | Level 3 PLUS Loading and viewing a foods.csv file with recipes.  TODO: Explain what you did here and where to find this feature in your code.   * **What was done**: The application can load and display both basic foods and recipes from the **foods.csv** file. * **Where to find it**:   + The **CsvHandler** class’s **loadFoodData** method parses the CSV file and distinguishes between basic foods and recipes based on a prefix ('B' for basic foods, 'R' for recipes)​​. | +5 (80) |  |
| ☐ 5 | Level 4 PLUS Select recipe(s) as well as basic food(s) for the daily intake. The selection is stored in the database (log.csv) and the log view is updated with the dietary information about the nutrients consumed.  TODO: Explain what you did here and where to find this feature in your code.   * **What was done**: Users can select not just basic foods but also recipes for inclusion in the daily log. * **Where to find it**:   + The logic to select foods and recipes and add them to the daily log is managed in the **DietManagerController**​​.   + The **DietManagerView** provides a UI for selecting these items​​. | +5 (85) |  |
| ☐ 6 | Level 5 PLUS add new recipe(s) to the food database. This implies the ability to  then add such recipe(s) to today's log as in level 5.   * TODO: Explain what you did here and where to find this feature in your code.  **What was done**: Users can define and add new recipes to the application, which can then be added to the daily log. * **Where to find it**:   + In the **DietManagerView**, there's a section for entering new recipe details, including ingredients and servings​​.   + The **createAndAddRecipe** method in **DietManagerController** adds this new recipe to the model and updates the CSV files​​. | +5 (90) |  |
| ☐ 7 | Level 6 PLUS the ability to read a non-empty log.csv file, to navigate to  different days in the log, and to select foods for the intake for the days other than today.   * TODO: Explain what you did here and where to find this feature in your code.  **What was done**: The application supports reading existing log entries from **log.csv**, navigating through different dates in the log, and adding food items to logs for any date. * **Where to find it**:   + The **CsvHandler** class handles loading of log data, allowing for logs from different dates to be read into the system​​.   + The **DietManagerView** includes a date picker component for selecting a specific date for the log entries​​. | +5 (95) |  |
| ☐ 8 | Level 7 PLUS the ability to save the log and food database back to the log.csv and foods.csv files.   * TODO: Explain what you did here and where to find this feature in your code.   **What was done**: Any changes made within the application to the food items or daily logs are saved back to the respective CSV files. * **Where to find it**:   + The **saveCsvData** method in the **DietManagerModel** triggers the CSV save operations, calling **saveFoodData** and **saveLogData** methods in the **CsvHandler**​​​ | +5 (100) |  |
| SUBTOTAL: | | **100** |  |

To receive any credit for level N, the preceding levels must be sufficiently functional to test level N. In general, this means previous levels must work without failure when the user enters normal (non-error) data.

Remember, the solution should apply design patterns as required. The final grade will be adjusted based on how good you have applied the pattern. You may lose up to 25% of the final project grade in this regard.