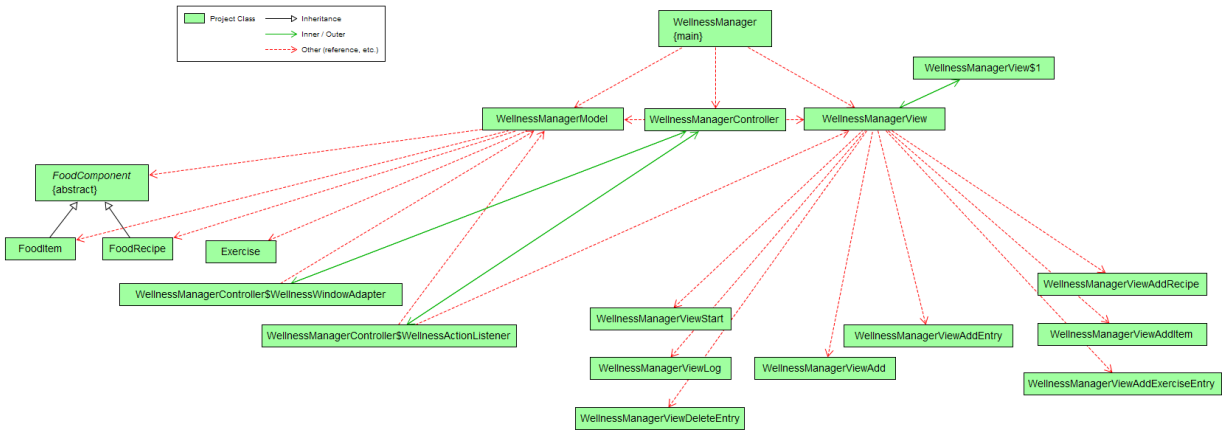
Team: Group 2
Members: Ivo Masar and Josip Peter Lozančić

Date: 04.26.2021

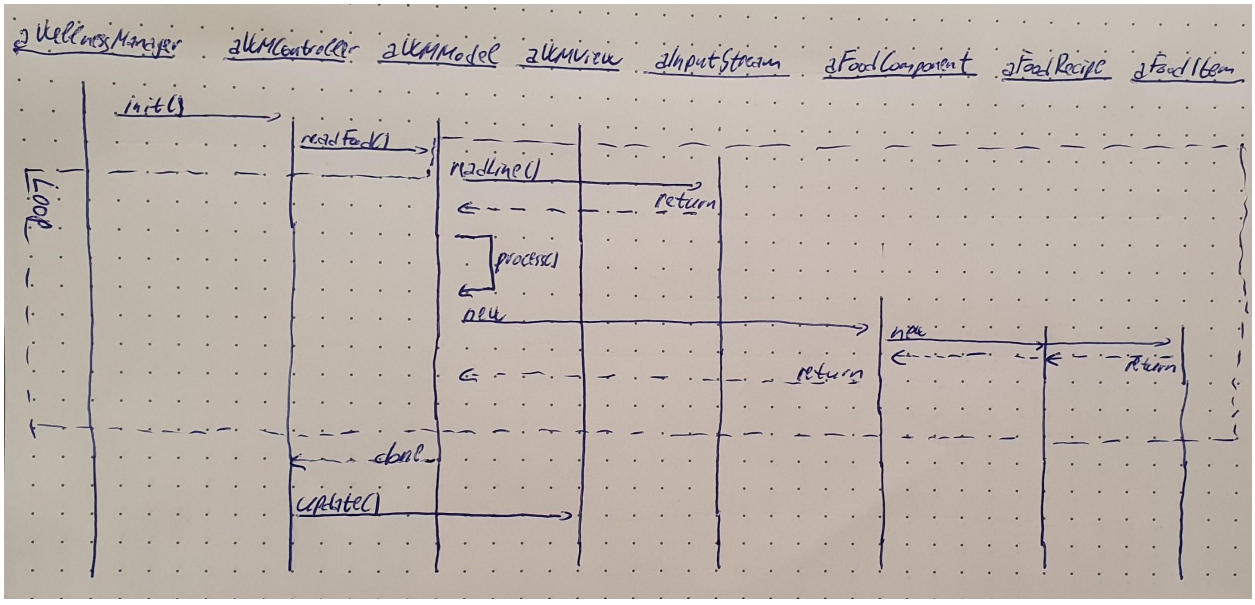
Design Sketch

Class diagram:

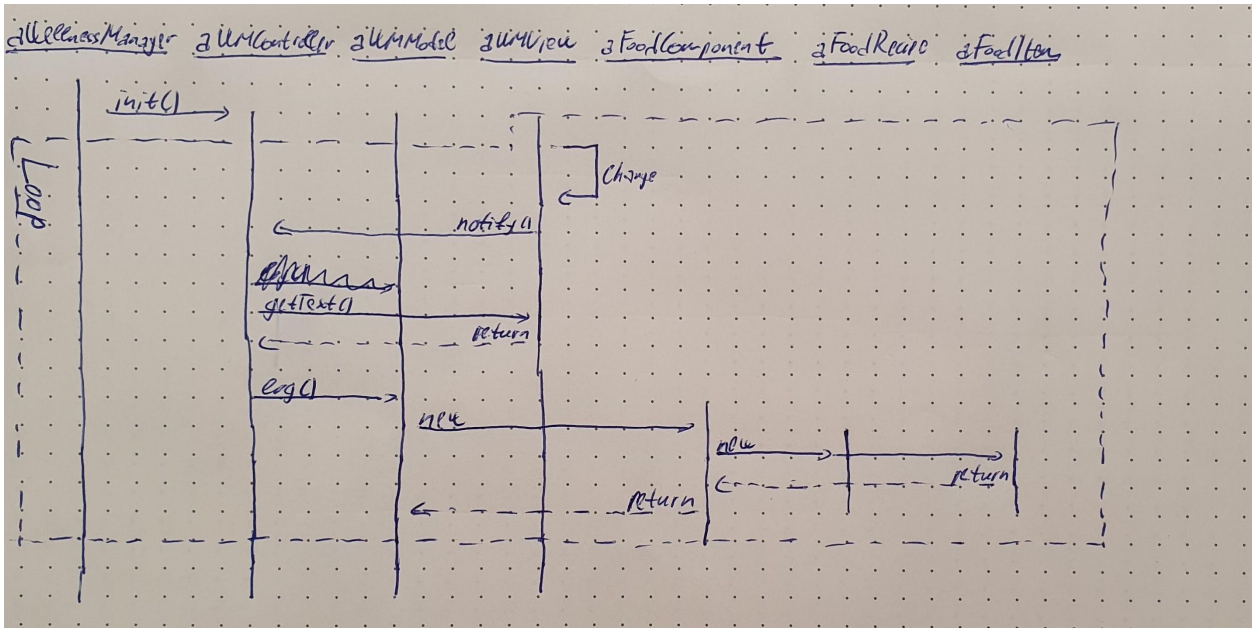


Sequence diagrams:

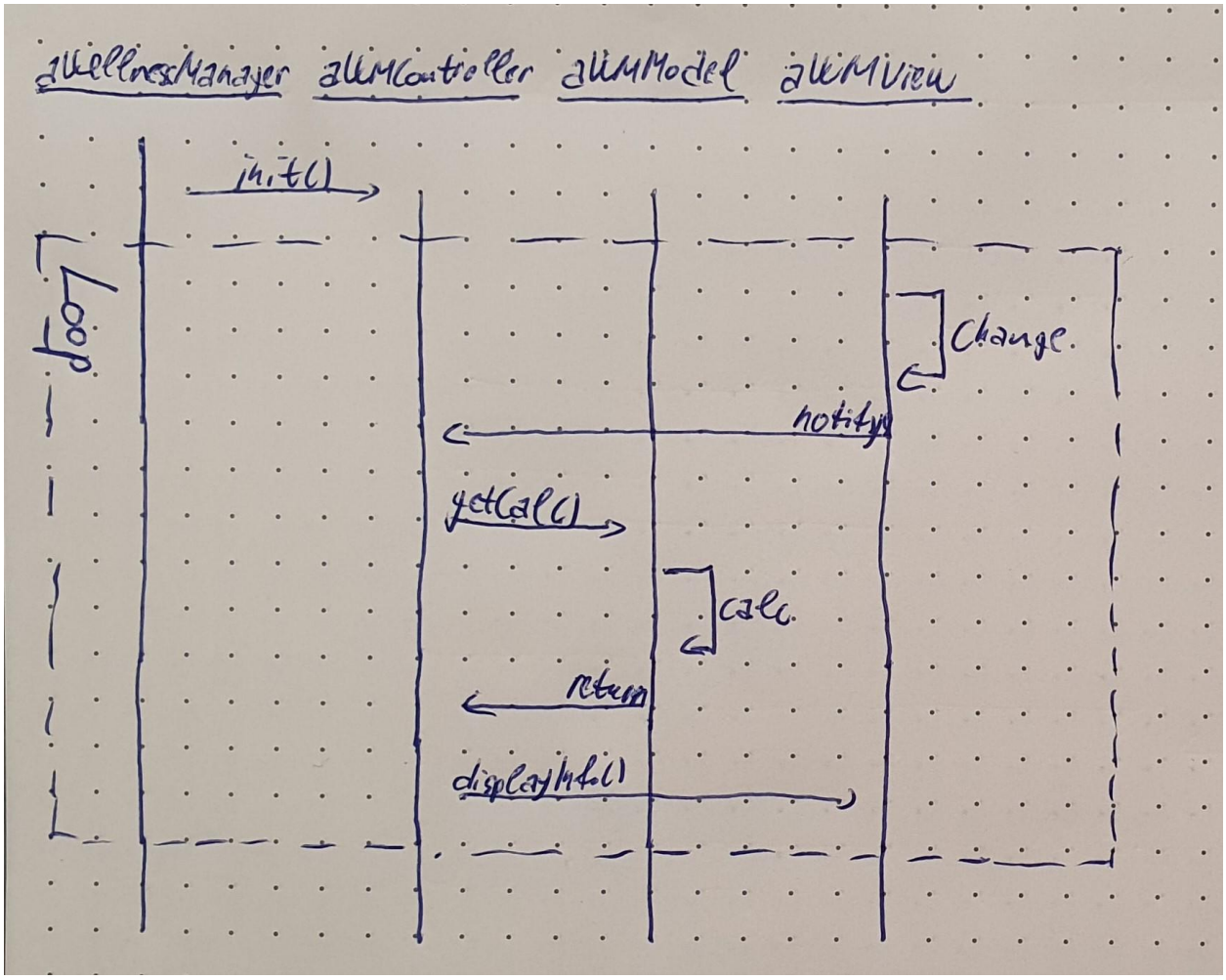
1. Reading food database:



2. Add new intake log:



3. Compute total number of calories:



Explanation:

Classes:

WellnessManager - Client class which starts the application.

WellnessManagerController - Controller class which keeps tabs on both the model and the view.

WellnessManagerModel - Model class in charge of holding data and manipulating said data.

WellnessManagerView - View class in charge of everything GUI related. Visual facade which utilizes different classes to change window content.

WellnessManagerViewStart - The starting page of the GUI.

WellnessManagerViewAdd - The page of the GUI in charge of food items and recipes.

WellnessManagerViewAddItem - The page of the GUI in charge of adding new basic food items.

WellnessManagerViewAddRecipe - The page of the GUI in charge of adding new food recipes.

WellnessManagerViewLog - The page of the GUI in charge of handling the logging of daily food intake.

WellnessManagerViewAddEntry - The page of the GUI in charge of adding new food intake logs for a certain date.

WellnessManagerViewAddExerciseEntry - The page of the GUI in charge of adding new exercise logs for a certain date.

WellnessManagerViewDeleteEntry - The page of the GUI in charge of deleting logs for a certain date.

Exercise - Class which defines the structure of objects which keep information about exercises from the exercise.csv file.

FoodComponent - Abstract component class which defines the basic behavior of FoodItems and FoodRecipes.

FoodItem - Leaf class which defines a single basic food item.

FoodRecipe - Composite class which hold food recipes and by that other recipes and/or basic food items

Rationale:

The entire application is based on the MVC pattern in order to separate concerns. The model is tasked with holding data and specifying in which way the data is handled/processed. The view is tasked with everything GUI related. The controller is the bridge between the model and the view. This increases cohesion and decreases coupling which makes the

application easier to maintain. It also makes the application more suitable to be worked on by multiple people. The composite pattern is implemented around basic food items and recipes where each recipe is a composite of either other recipes and/or basic foods. It makes doing calculations on the entire collection easier to be done (especially in the case of recursive traversal). The facade pattern is implemented on top of the view. Each different GUI page is described and handled in a different class which simplifies the view and increases cohesion (however, gives an increase in coupling as well). Dependency injection used through the MVC pattern.

The shortcomings of the application are:

1. The implementation of the WellnessManagerView class which is connected to all the other GUI classes. I feel like it could be done better to decouple the entire GUI a little more, but every try significantly decreased the cohesion of the entire application.
2. The WellnessManagerModel class is a little bloated, however, as there's only two people on the team it was easier working with one bloated class than many little classes.

Skeleton implementation:

Available in github directory **swen-best-group**.