# DietManager Test plan – Exam

**Identifier**: DietManager Test Plan

**Version**: 1.0

**Authors:** Harry Liam, Daniel Lysak, Habil Lai, Isak Standal and Elsa Lossius

**Change log**

| Version | Modification/description | Author | Date |
|---|---|---|---|
| 0.1 | Initial draft | all | 12.04.2021 |
| 0.2 | Added overall system design | Everyone | 14.04.2021 |
| 0.3 | Modified overall system design | Everyone | 20.04.2021 |
| 0.4 | Acceptance test | all | 20.04.2021 |
| 0.5 | Modified schedule | Habil | 23.04.2021 |
| 1.0 | Quality assured | Everyone | 26.04.2021 |

# Introduction

This is a master test plan for the DietManager for the exam in Introduction to Software Testing (ITP2200). References: The system requirement and overall design is defined in the exam document Exam-April-2021.pdf

**Information about the SUT**
The System under test is a system for managing diets for people with different dietary restrictions and preferences, such as allergies, veganism and favorite foods.

**Purpose of the test plan**
The test plan defines the overall test strategy and the tactical approach of testing. It describes how and why the tests are made and how they are carried out as well as the scheduling of these activities. It also describes risk assessments and contingency planning. The test plan will:

- Help us systemize the test process
- Make sure we fulfill the requirements
- Ensures quality at every step in the development process
- Help Identify bugs early on
- Provide a roadmap
- Keep track of the progress
- Clarifies roles and responsibilities
- Ensures that requirements are met

**Change control process**

We will make ongoing changes as we gain more insight throughout the project period. Before making any changes, the group will discuss and reflect upon these changes. In case of disagreement, the decision is made by a majority vote. If there is a deadlock, the project manager decides. Changes in the document will be made by the person who proposed the change, and must be quality assured by Harry. This will be recorded in the changelog.

Communication and coordination take place mainly through daily stand-up meetings and at the sprint review at the end of each week. We will have unplanned meetings if necessary. We will be using the following software:

- Github: For version control and collaboration.
- Buddy: For continuous integration testing.
- Slack: Formal communication and planned meetings.
- Discord: For digital meetings as well as casual communication.

## Test items

Our System under test is the Diet Manager. The Diet Manager shall be tested so that the specified requirements are met.

## Features to be tested

**Requirement 1 (Habil/Daniel)**
Diets have several restrictions:

a. If a diet contains any non-vegan food, it is considered not vegan (i.e., isVegan = false).
b. If a diet contains only vegan food, it is considered vegan, even if it is not a VeganDiet (e.g., it could be a LowCarbDiet).
c. A VeganDiet cannot contain non-vegan food.
d. The preferred meat in a FlexitarianDiet MUST be non-vegan food of protein type.
e. The maximum carb-type foods that can be included in a LowCarbDiet is two.

**Requirement 2 (Elsa/Harry)**
A person can be following any given diet, except in the following cases:

a. If their favorite food is non-vegan, they cannot follow a VeganDiet.
b. They cannot follow a diet if they are allergic to 50% or more of the food allowed by the diet.
c. If they weigh less than the limit set by the VeganDiet or the LowCarbDiet, they cannot be following these diets (for health reasons).
d. If they weigh more than the limit set by the HypercaloricDiet, they cannot be following this diet (for health reasons).

**Requirement 3 (Daniel)**
The Diet class should implement the following methods:

a. Write the duration of a diet in terms of years, months and days, e.g., “This VeganDiet lasts for 2 years, 3 months and 5 days”.
b. Write the allowed food, e.g., “The following food is allowed in this FlexitarianDiet: Salad, Soup, Apple, Strawberry, Salmon”.

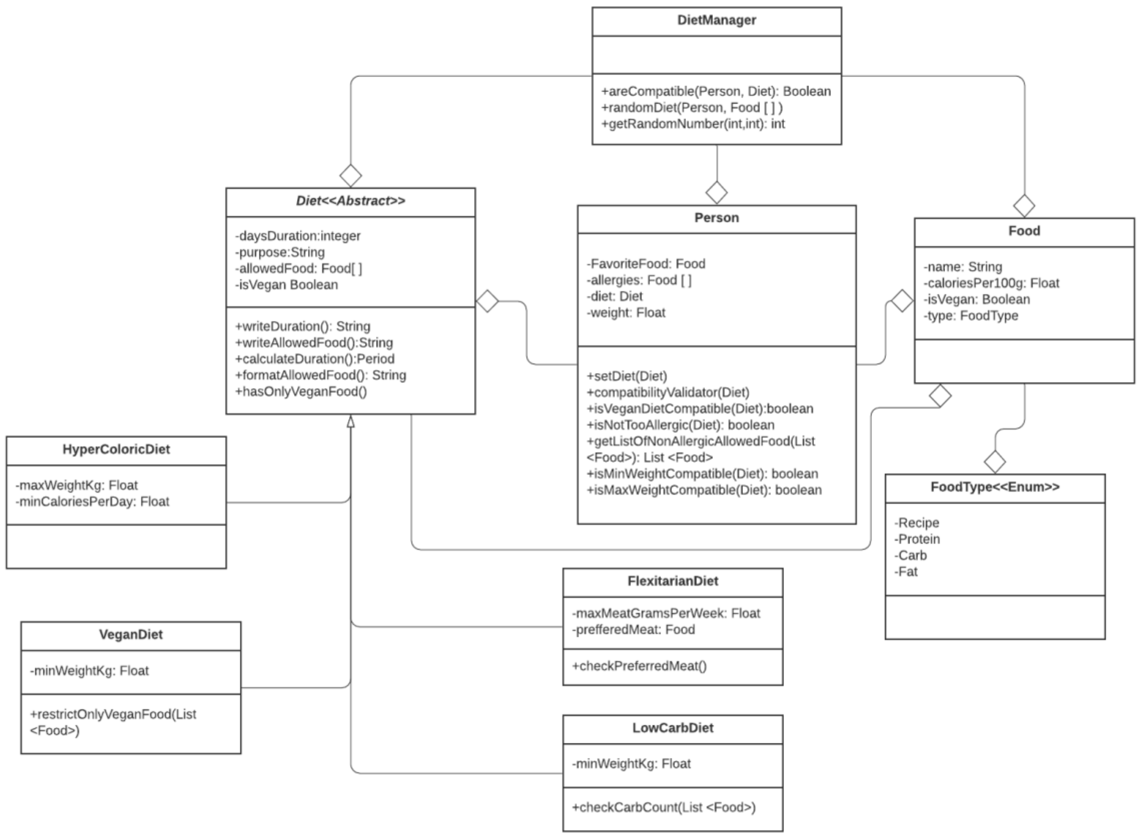
**Requirement 4 (Isak)**
The DietManager class should implement methods for the following purposes:

a. Given a Person and a Diet, return true if they are compatible, false otherwise.

b. Given a Person and a list of Food, create a random HypercaloricDiet with the following attributes:

i. daysDuration: random number between 1 and 100.
ii. purpose: “Random diet”.
iii. allowedFood: all the Food from the list that the person is not allergic to.
iv. isVegan: false if there is some non-vegan Food, true otherwise.
v. maxWeightKg: random number between Person.weight and Person.weight + 20.
vi. minCaloriesPerDay: random number between 2000 and 4000.

## Overall System design


DietManager
+areCompatible(Person, Diet): Boolean
+randomDiet(Person, Food [ ] )
+getRandomNumber(int,int): int
Diet<<Abstract>>
-daysDuration:integer
-purpose:String
-allowedFood: Food[ ]
-isVegan Boolean
+writeDuration(): String
+writeAllowedFood():String
+calculateDuration():Period
+formatAllowedFood(): String
+hasOnlyVeganFood()
Person
-FavoriteFood: Food
-allergies: Food [ ]
-diet: Diet
-weight: Float
+setDiet(Diet)
+compatibilityValidator(Diet)
+isVeganDietCompatible(Diet):boolean
+isNotTooAllergic(Diet): boolean
+getListOfNonAllergicAllowedFood(List <Food>): List <Food>
+isMinWeightCompatible(Diet): boolean
+isMaxWeightCompatible(Diet): boolean
Food
-name: String
-caloriesPer100g: Float
-isVegan: Boolean
-type: FoodType
FoodType<<Enum>>
-Recipe
-Protein
-Carb
-Fat
HyperColoricDiet
-maxWeightKg: Float
-minCaloriesPerDay: Float
VeganDiet
-minWeightKg: Float
+restrictOnlyVeganFood(List <Food>)
FlexitarianDiet
-maxMeatGramsPerWeek: Float
-prefferedMeat: Food
+checkPreferredMeat()
LowCarbDiet
-minWeightKg: Float
+checkCarbCount(List <Food>)

## FEATURES TO NOT BE TESTED

There’s no features which will not be tested. We will come back to this point to specify which features we won’t test.

- We won’t be testing Food since it’s only a constructor.

- Hypercaloric won’t be tested as well. Only getters and setters/constructor. MaxWeightKg() will be tested in Person class.

## Approach

**Test oracle:**
We will be using “specified” test oracle as we have the full specification and know the exact behavior of the system under test.

**Tools**
We will create automated tests using the JUnit framework. The advantage is we will save a lot of time, as it will streamline the testing process, provide a high degree of reusability and make it easier to measure the quality.

**Metric**
As metrics for measuring the quality of testing we will use coverage and number of test cases.

**Unit testing**
We will test each method in each class as a single unit in isolation. This will make it easier to identify and correct bugs at the unit level.

**Test driven development (TDD)**
We will create the test before coding. The test will first fail, and will pass when it meets test requirements. This is a good practice and will make debugging easier.

**Continuous integration(CI)**
For automated tests that give feedback on the impact to the previous code base when introducing new code/merging.

**Integration testing**
After unit testing, we will test det units as a group and how they behave together.

**Top-down approach**
We are going to follow a top-down approach as we first are focusing on testing the overall requirement in the system, and then gradually integrating the lower-level components.

**Regression testing**
As we are integrating the different components, we will re-run all unit tests to make sure that they still maintain the correct behavior as previously tested.

## Pass/fail criteria

**Unit test**
We will test each method in each class and achieve at least 50% coverage as specified in the Exam-April-2021.pdf. We will use the Assert methods in Junit to determine if a test has passed or failed. When the assertion returns true it is considered passed.

**Acceptance test**

| **Requirement** | **Criterium** | **Test Cases** |
|---|---|---|
| 1. Diets have several restrictions: | a. If a diet contains any non-vegan food, it is considered not vegan (i.e., isVegan = false). | **DietTest**.shouldPassDietIsVegan()<br><br>**DietTest**.shouldPassDueToDietNotVegan() |
| | b. If a diet contains only vegan food, it is considered vegan, even if it is not a VeganDiet (e.g., it could be a LowCarbDiet). | **DietTest**.shouldPassDietIsVegan()<br><br>**DietTest**.shouldPassDueToDietNotVegan() |
| | c. A VeganDiet cannot contain non-vegan food. | **VeganDietTest**.shouldThrowErrorWhenVeganDietHasNonVeganFood() |
| | d. The preferred meat in a FlexitarianDiet MUST be non-vegan food of protein type. | **FlexitarianDietTest**.shouldThrowErrorIfFlexitarianPreferredMeatIsVeganOrNotOfProteinType()<br><br>**FlexitarianDietTest**.shouldPassIfFlexitarianPreferredMeatIsNotVeganAndOfProteinType() |
| | e. The maximum carb-type foods that can be included in a LowCarbDiet is two. | **LowCarbDietTest**.shouldThrowErrorIfTooManyCarbFoodTypes()<br>**LowCarbDietTest.**shouldNotThrowErrorIsInRangeForFoodTypeCarb() |

| | | |
|---|---|---|
| 2. A person can be following any given diet, except in the following cases: | a. If their favorite food is non-vegan, they cannot follow a VeganDiet. | **PersonTest.**testCompatibilityValidator_1a()<br><br>**PersonTest**.testCompatibilityValidator_1e() |
| | b. They cannot follow a diet if they are allergic to 50% or more of the food allowed by the diet. | **PersonTest.**testCompatibilityValidator_1b()<br><br>**PersonTest**.testCompatibilityValidator_1e() |
| | c. If they weigh less than the limit set by the VeganDiet or the LowCarbDiet, they cannot be following these diets (for health reasons). | **PersonTest**.testCompatibilityValidator_1c()<br><br>**PersonTest**.testCompatibilityValidator_1e() |
| | d. If they weigh more than the limit set by the HypercaloricDiet, they cannot be following this diet (for health reasons). | **PersonTest.**testCompatibilityValidator_1d()<br><br>**PersonTest**.testCompatibilityValidator_1e() |
| 3. The Diet class should implement the following methods: | a. Write the duration of a diet in terms of years, months and days, e.g., “This VeganDiet lasts for 2 years, 3 months and 5 days”. | **DietTest.**shouldWriteDurationCorrectlyToString() |
| | Write the allowed food, e.g., “The following food is allowed in this FlexitarianDiet: Salad, Soup, Apple, Strawberry, Salmon”. | **DietTest.**shouldWriteAllowedFoodCorrectly() |
| 4. The DietManager class should implement methods for the following purposes: | a. Given a Person and a Diet, return true if they are compatible, false otherwise. | **DietManagerTest.**testAreCompatible_1a()<br><br>**DietManagerTest.**testAreCompatible_1b() |
| | b. Given a Person and a list of Food, create a random HypercaloricDiet with the following attributes:<br>i. daysDuration: random number between 1 and 100.<br>ii. purpose: “Random diet”.<br>iii. allowedFood: all the Food from the list that the person is not allergic to.<br>iv. isVegan: false if there is some non-vegan | **DietManagerTest.**testRandomDiet() |

| | | |
|---|---|---|
| | Food, true otherwise.<br>v. maxWeightKg: random number between Person.weight and Person.weight + 20.<br>vi. minCaloriesPerDay: random number between 2000 and 4000. | |

## Criteria for beginning testing

Before we can begin testing we must first create the test cases as well the skeleton of the system. This includes all the classes and its fields, constructors, getters and setters.

In cases where a higher level component is dependent on functionality from lower level components that we haven't created yet, we will create stubs to simulate these functionalities.

We need to create test data as input for our test cases. In this case it will be instances of Diet, Person and Food.

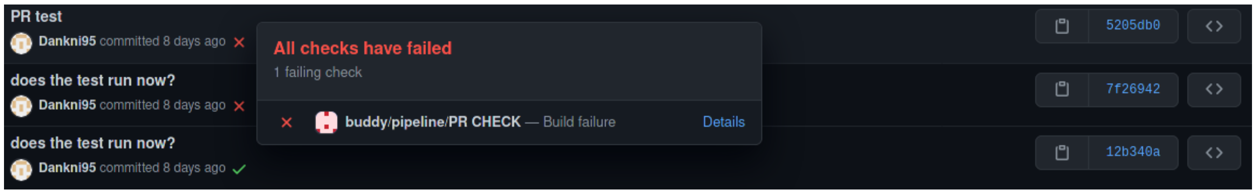
## Criteria for suspending test and requirements for restarting

When any defects are found in SUT which impacts the tests, the team or individual per his/hers responsibility may suspend the tests. Examples for defects that warrant testing suspension:

- Functionality doesn't work as specified in SRS (Software Requirement Specification).
- The output is not the same as expected.
- Conflicting code with previous build.

As we all work on different parts of the code with different responsibilities, we must be aware of merge conflicts/code incompatibility when expanding the code base with additional features.

The use of continuous integration that does functional acceptance tests for the team is a good indicator if the tests should be suspended either individually or for the team as a whole.



PR test
Dankni95 committed 8 days ago ✗
does the test run now?
Dankni95 committed 8 days ago ✗
does the test run now?
Dankni95 committed 8 days ago ✓

All checks have failed
1 failing check
✗ buddy/pipeline/PR CHECK — Build failure Details

5205db0
7f26942
12b340a

If the SUTs tests do not throw an error/fail, but the functionality doesn't work as specified in SRS (Software Requirement Specification), testing should be immediately suspended and the rest of the team notified.

**Resumption/restarting:**

When the defects/problems that caused the suspension have been resolved, the individual or the team as a whole can resume the testing.

## Test deliverables/status communications documents

- Test plan
- Zip file including the following:
    - The System under test
    - Test code
    - README.md file explaining how to run the SUT
- Summary report
- Graph and analyze of a picked method
- Group reflection note
- Individual reflection note from each member

## Risk / limitations:

There will always be a time and due issue linked to a task and especially an exam where the time and date usually is impossible to delay. I think that will be the main risk for us as well. Even with a structured and good schedule we can end up using too much time and effort on unforeseen challenges. Another risk is overtime, spending too much time working overtime by yourself or with other members of the group can cause a bad vibe in the group and leave the team morale, productivity and the quality in bad shape.

I do not think we will meet many limitations when it comes to this exam. We use all what we have learnt from this course and if we meet challenges underway, it is possible to use google **and** other resources to try learn the things we are stuck on or need to learn to move on.

There will always be issues that we don't expect to handle and don’t have a plan for. We will solve all the problems, risks and limitations in the best possible way and deliver a great product.

| Risks | Probability | Contingency Plan |
|---|---|---|
| Bad code and bad practices | Low | Peer review then refactor in collaboration. |
| Interpreting SUTs requirements Seeing as we get a QA session, we find this point to be at low probability. | Low | We ensure to make assumptions and if possible inquire the teacher. |
| Merge conflicts Merge conflicts may occur. Often leading to delays. | Med | We ensure to work in different classes or on different tasks(good design), if merge conflicts occur, that break code CI(buddy.works) will prompt everyone to failing tests. Fix in collaboration. |
| Progress delayed | Med | Physical meetings are prefered. May be delayed due to the current pandemic. Increased meeting duration may be necessary to compensate. |
| Loss of data | Low | Every member is responsible to push changes to GitHub. Person in charge will make sure that every member has sufficient training beforehand. |
| Illness May impact productivity. | High | Digital meetings instead of physical. Responsible will delegate tasks to other members to compensate. |

## HARDWARE AND SOFTWARE REQUIREMENTS

**Hardware requirements**
For this program a standard computer is required.

**Software requirements**
The diet manager is a pretty basic and resource-friendly program. To run the program you need JDK and an **integrated development environment** (**IDE**) of your choice. Code is written using JDK 15, but the SUT should be backwards compatible!
See the pom.xml. (To teacher, we have not learned about maven yet, but for learning purposes we would like to use this in our project, if everything is correctly set up this should make our collaboration easier and for you to run the program.)

## STAFFING AND TRAINING NEEDS

Due to the nature of the project, any further explanation of staffing and training needs won’t be included in the test-plan. We expect that every member has a basic understanding of this course and also has enough to spare time to work in this project according to our schedule as described earlier.

Any member will be able to ask questions during our daily brief to make sure that everyone is up to date in terms of the status of our project, but training may also be offered by the team for the member who may lack knowledge during the meeting.

## Responsibilities

We will make sure that even though a team member is responsible for something, that doesn’t necessarily mean that they are working individually on the task. Each member’s responsibilities are decided based on their past experiences and knowledge.

Habil is responsible for scheduling joint meetings for the whole group. We will try to meet everyone’s wishes, but he/she will have the final word on meeting time and date if team members have different opinions.The individual will also make sure that every team member is adequately informed to avoid misunderstandings.

Elsa is responsible for overseeing the development of the skeleton of the system. This includes all the classes and its fields, constructors, getters and setters.

Elsa is also responsible for writing and drawing the graphs for the method picked for analysis with help and inputs from other team members.

Isak will be responsible to either write down a short report or verbally inform non-attended participants if other errands stop them from attending.

Harry/Habil will mainly be responsible for writing the Test Plan based on inputs from other members. He will not take decisions alone but will have the last word on what to be included in the text.

Daniel has sole responsibility to approve or disapprove new feature/code implementation to the main production “Master” branch. No one is able to merge conflicting code(Having CI failed test cases), this is the rule set up on Github.

Harry will be responsible for which tests to be included or not as well as reviewing the classes to make sure that code makes sense and is readable for others.

Harry will be responsible for deciding if any new modules will be introduced to the Diet Manager.

Tasks were distributed based on the Diet Managers requirements (See page 3).

## SCHEDULE

Disclaimer: The dates for the schedule were changed during the project weeks as long as we were ahead of the schedule. Our first draft assumed that we would deliver 28.04.21. We worked fast and efficiently and managed to complete a few days before.

For example. If we were done with the tasks for day 1, we could jumpstart on one of the tasks for the next few days. This way we completed earlier.

If we ever were behind schedule we made sure to work overtime to catch up.

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | **01.04 - 09.04** |
|---|---|---|---|
| Teambuilding | Multiple casual Discord meetings.<br>A physical meeting at campus where we will code and try to write a fictional test-plan.<br>Different online games together. | Everyone | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | **09.04 - Friday** |
|---|---|---|---|
| Project start | Zoom meeting with Alberto. Making sure to ask all necessary questions regarding | Everyone | |

| | | | |
|---|---|---|---|
| | the project to clarify any misunderstandings. | | |
| Daily brief. | Discord meeting. Go through thoughts and questions about the exam. Start to fill out the test-plan. | Everyone | |
| Starting to form the test-plan | Discord meeting. Everyone will be working on the same Google document. | Everyone | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | **10.04 - 11.04** |
|---|---|---|---|
| Weekend. | Every weekend will be dedicated for other errands. Any member may work on the project if they wish to do so, but will be responsible to update everyone on changes during the next daily brief or through a written report published to Slack. | Everyone | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | **12.04 - Monday** |
|---|---|---|---|
| Daily brief | Status if any changes were done during the weekend. Discuss goals for the day | Everyone | |
| Check that Git is working for every member | Daniel, who is responsible for Git, will check if every member's Git works as expected in Intellij. | Daniel | |
| Go through | Responsible will go | Daniel | |

| | | | |
|---|---|---|---|
| requirements | through the requirements for Diet Manager. Discuss the design and architecture of the software. | | |
| Create a skeleton for the program | Start to code. | Everyone | |
| BREAK | Dinner. | Everyone | |
| Code | Keep working on the skeleton. | Everyone | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | **13.04 - Tuesday** |
|---|---|---|---|
| Daily brief | Go through the basic skeleton of our program. Responsible will double check and verify that others work is up to standards. Inform others if changes are needed. | Everyone | |
| Work on the skeleton. | Will be working on it until responsible is satisfied. | Everyone | |
| | | | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | **14.04 - Wednesday** |
|---|---|---|---|
| Daily brief | Status. | Everyone | |
| Final touches on test-plan | First draft of the test plan should be close to done. Any final touches will be added now.<br><br>We will add more to | Everyone | |

| | | | |
|---|---|---|---|
| | the test plan as we go. | | |
| Verify that the test plan is OK. | Double check that there’s no plagiarism. Spell check and make sure that the test plan is understandable for others. | Harry, Habil | |
| MILESTONE | First milestone. First draft should be done. | Everyone | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | **15.04 Thursday** |
|---|---|---|---|
| Daily brief | Discord meeting. | Everyone | |
| Code | Start to fill out the skeleton. More detailed description of tasks will be specified closer to the date. | Harry | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | 15.04 Thursday |
|---|---|---|---|
| Daily brief | | | |
| Skeleton verification | Responsible double check if the skeleton is designed as intended. | Elsa | |
| Assign classes to each member. | Responsible picks out which classes each member will be working on. | Daniel | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | 16.04 Friday |
|---|---|---|---|
| Daily Brief | Meeting | | |

| | | | |
|---|---|---|---|
| Requirement 1 | Working together online using Discord and screen sharing. | Habil and Daniel | |
| Requirement 2 | Working together face to face. | Elsa and Harry | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | |
|---|---|---|---|
| Daily Brief | | | |
| Requirement 3 | Methods and tests in spare time. | Daniel | |
| Requirement 4 | Done together online on Discord. Whole group attending. | Isak, Everyone. | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | 19.04 Monday |
|---|---|---|---|
| Daily Brief | Physical meeting | Everyone | |
| Refactor | Renaming to understandable names. Too many temporary test and variable names. | Daniel, Harry | |
| | | | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | 19.04 Monday |
|---|---|---|---|
| Daily Brief | Discord meeting. | Everyone. | |
| Unit test. | Every team member is responsible for their own unit tests. Harry, Elsa and Isak will go through the integration test at a later stager. | Everyone | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | 19.04.21 |
|---|---|---|---|
| Daily Brief | Physical meeting. | Everyone. | |
| Integration test. | Making sure that we don’t break the code when integrated. Writing down potential technical obstacles. | Harry, Elsa, Isak. | |
| Discuss to address bugs and faults. | Remove and add more as needed. | Daniel, Harry | |
| Debug. | Debug. | Everyone | |
| Review complete code. | Going through the code on a projector. Removing redundant and unnecessary code. Adding more tests if necessary. | | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | 20.04 Tuesday |
|---|---|---|---|
| Daily Brief | Physical meeting | Everyone | |
| Achieve desired coverage. | Remove getters and setters to achieve desired coverage. | Daniel, Everyone | |
| Acceptance test. | Double check the requirements and add all the code directly related to the requirements to the test-plan. Elsa going through the code and adding necessary tests to the acceptance test. | Elsa | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | 22.04 Thursday |
|---|---|---|---|
| Write reflection note | Written by Habil with input from everyone. | Habil | |
| Individual reflection note | Everyone writes their own individual reflection note. Done separately later this evening. | Everyone | |
| Collect and review all of the reflection notes. | Habil takes the responsibility of collecting all the notes and reviewing them. Making sure everything is ready for delivery. | Habil | |

| **TO DO** | **IMPLEMENTATION** | **RESPONSIBLE** | 23.04 Friday |
|---|---|---|---|
| Daily Brief | Double check everything. Program, Test-plan, reflection note, individual reflection note. | Everyone | |
| MILESTONE | Project done. | | 26.04 Monday |

## Approval

The project is finished when all members have the same collective opinion that the software is usable as intended.