Project Report

Airmed

Group 25

TU Delft

Project Report

Airmed

by

Sara Boby (4645588),
Jack Cook (4543092),
Phillipe Lothaller (4460820),
Paul Simon Schön (4531515),
Florina Sirghi (4648579)

TU Delft

Contents

1 Introduction **1**

2 Software Development Process **2**
2.1 Sprints . . . 2
2.1.1 First Sprint . . . 2
2.1.2 Second Sprint . . . 3
2.1.3 Third Sprint . . . 4

3 Requirements Engineering **6**
3.1 Requirements . . . 6
3.2 Showcase Material . . . 7
3.2.1 User Stories . . . 7
3.2.2 State Diagram . . . 8
3.2.3 Activity Diagram . . . 9

4 Software Architecture **10**
4.1 General Implementation . . . 10
4.2 Implementation of Airmed's Functionalities . . . 10
4.3 Django . . . 10

5 Software Testing **12**
5.1 Coverage . . . 13
5.2 Overview of Tests . . . 14

6 Reflection and Adaptation **16**
6.1 Paul Simon Schön . . . 16
6.2 Sara Boby . . . 16
6.3 Florina Sirghi . . . 16
6.4 Jack Cook . . . 17
6.5 Phillipe Lothaller . . . 17

7 Conclusion **18**

Appendices **19**

A Supporting Material **20**

Bibliography **24**

1 Introduction

Link to access the Airmed app: `https://airmedfitness.herokuapp.com`
GitLab link: `https://gitlab.ewi.tudelft.nl/ti3115tu-2019/group-25/tree/master/mysite`
Screencast link: `https://youtu.be/ia58KpgtS0E`

Airmed is a state of the art, user friendly, fitness and health tracker app for beginner and veteran gym goers. It is one of the first apps to incorporate tracker for both the users' nutritional and physical activity over an extended period of time. With an integrated calorie and fitness tracker, it shows users their progress over time. The app will also have a blogging capability for all its members. It allows for the whole Airmed fitness community to interact with one another, so that more experienced members can share their tips and tricks with newer members. For the beginner to the fitness journey, the app will offer a variety of basic exercises for different muscle groups. The incorporated fitness tracking feature allows users to keep track of their repetitions, consistency and one repetition maximum statistics over time, finally providing graphical representation of their fitness journey. More advanced users can make use of the superior tracking capabilities of the Airmed app, so that they can follow their progression and see if they are on track to achieving their personal goals.

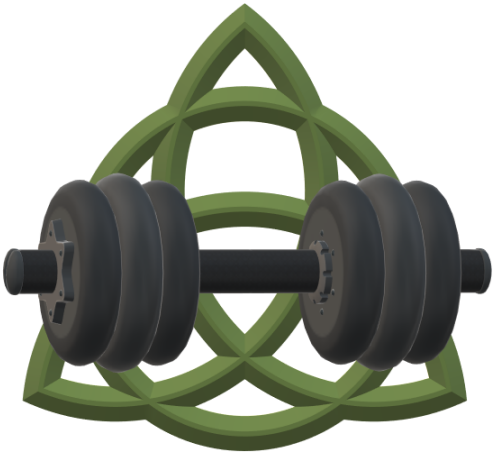

Figure 1.1: The logo of Airmed.

2

Software Development Process

In the software development process of our product, the team made use of Scrum, a highly popular agile process framework, that was introduced to us during the lectures of the *Software Engineering and Methods* course.

Considering that the team consisted of only five members and the product that we developed is not of high complexity, the traditional roles utilised in the Scrum framework- Product Owner, Scrum Master and Development Team- have been covered by all of us at some point during the development process. However, the main role that each of the members within the team focused on was that of being part of the Development Team.

Due to the fact that the time frame of our present project was quite short, the sprints used by the team had to be short as well. Because the amount of tasks that could be completed/code that could be produced in only one week was deemed as not substantial enough by our team for a single sprint, we decided to choose an average length of two weeks for each sprint. In the following sections in this chapter, each of the three sprints performed by our team will be discussed and analysed. Furthermore, the version of the kanban presented during each sprint review will be shown.

2.1. Sprints

2.1.1. First Sprint

The first sprint was planned during the end of week 1.2. Previous work regarding the requirements engineering aspect of the project has been done up to this point. Thus, the backlog for the current sprint included finishing the requirements gathering and prioritising process, as well as creating the UML Use Case, State and Activity diagrams, which can be found in Appendix A. Furthermore, user stories had to be formulated. At the same time, it was planned that a web application framework for our product should be chosen and researched in this sprint.

2.1.1.1. Review and Retrospective

The review of the first sprint took place in week 1.3. This sprint was the shortest out of the three performed. During the sprint review, the kanban shown in Figure 2.1 was displayed. Due to the fact that the actual product development did not start yet, the team decided that at the end of this sprint review there was nothing yet to be shipped to the user.

The main points discussed during the retrospective of this sprint are presented next.

Positive points:

- Efficient daily communication, enabled by our Whatsapp group
- Good productivity during the lab hours

Negative points:

- Even though we met the goal of choosing Django as a framework for our project, the amount of research done in order to understand the way it works was not sufficient by the end of this sprint.

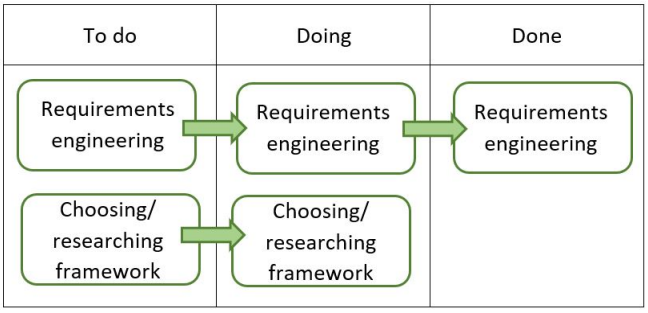

Figure 2.1: Kanban displayed at the end of the first sprint.

2.1.2. Second Sprint

The second sprint was planned at the end of week 1.3. The research into Django had to be finalised during this sprint. Furthermore, the features that were present on this sprint's backlog included:

- Setting up a functional website for our product
- Creating a way for the user to register and log in on the website
- Displaying a list of the main muscle groups, as well as a list of recommended exercises for each muscle group
- Creating a tracker for the calories consumed by the user and the weight of the user
- Creating a page where users could make posts and other users (such as the personal trainers) can interact with them

As it may be observed from the previous list, in this sprint the team tried to tackle all the major features of our product. We decided to work in an iterative manner: we wanted to provide all of the major features of our product at a low (but usable) fidelity by the end of this sprint. In the following iteration, the fidelity level could be raised. Thus, our goal was to get as far as possible with the development process of these features by the end of the current sprint, meaning that there were no harsh conditions imposed on when we would consider a feature "done" within this sprint. We focused on delivering a functional, but not very complex product at the end of this sprint, which included the main discussed features. This decision matched our agile mindset. However, the team decided to leave out the water tracker and the BMI calculator feature due to the lack of time and to simplify the fitness advice feature (it would not allow the user to choose the muscle group from a body diagram, but simply from a list of muscle groups), thus deviating from the original plan and modifying the requirements.

2.1.2.1. Review and Retrospective

The review of this sprint took place in week 1.5. During the sprint review, it was observed that the team managed to finalise the research into Django and to make significant progress in all of the features that have been included in the backlog. Theoretically, the current version could be shipped, but the team decided to wait until the features will be in a more advanced state and more extensive testing would be performed on them, as we valued software quality. Furthermore, the kanban displayed during the second sprint review is shown in Figure 2.2.

The main points discussed during the retrospective of this sprint are presented next.

Positive points:

- The group managed to make progress with all of the features included in the backlog
- By the end of the sprint, our product could provide some functional features for the user
- The group continues to communicate well

Negative points:

- We realised that it was difficult for the team to create automated tests for our software, for reasons presented in Chapter 5, so we could not really experiment with those.
- We had to decide to leave out or simplify some features and that we initially planned on implementing.

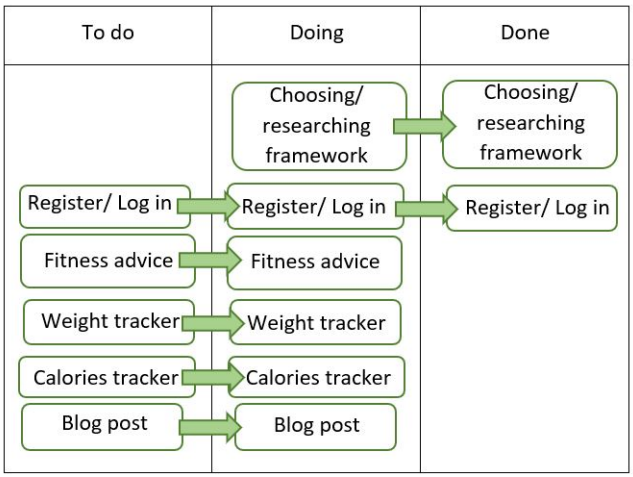

Figure 2.2: Kanban displayed at the end of the second sprint.

2.1.3. Third Sprint

The third sprint was planned at the end of week 1.5. The team decided that during this sprint the features that have been implemented during the last sprint should be made more complex and the overall software quality should be improved by means of refactoring the code. Furthermore, we focused on the user interface aspect of our product. Thus, the backlog for the final sprint of this project included the following:

- Creating a more extensive user profile, which stores and displays the following user data: profile image, username, email, gender, weight, height, birth date, location; furthermore, it offers the user the possibility to update their profile at any time
- Creating separate pages for the recommended exercises (instead of just displaying them as a list), which allow the user to input the number of repetitions, sets and the one repetition maximum that they achieved; the data is stored and the progress for the one repetition maximum over time is shown in the form of a graph
- Implementing graphs that will allow the user to observe their progress over time, both for the calorie tracker and the weight tracker feature
- Improving user interface
- Refactoring the code

2.1.3.1. Review and Retrospective

The review of the final sprint took place in week 1.7. During this sprint review, the team concluded that the features as well as the user interface that have been developed thus far were ready to be shipped to the user. This decision was made due to the fact that the features of our software were deemed of sufficient quality and they could be considered "done" from a requirements engineering/user story point of view. The kanban presented during the final sprint review is shown in Figure.

The main points discussed during the retrospective of the final sprint are presented next.

Positive points:

- The group managed to complete all the tasks/features included in the backlog.
- The final product was considered to be of sufficient quality and could be shipped to the user.
- The decision to refactor the code improved the overall architecture and user interface of our software.

Negative points:

- We did not create our own automated tests. Thus, we had to manually test each feature of our website in order to ensure that they function properly, which was more time consuming.

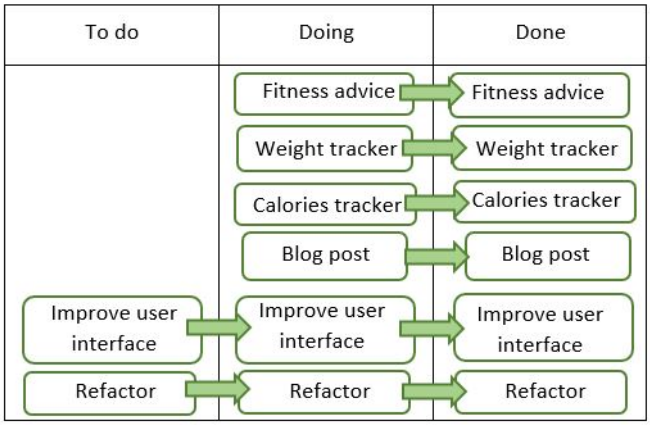

Figure 2.3: Kanban displayed at the end of the final sprint.

3

Requirements Engineering

In order to gather the requirements that act as the guidelines when developing the software for Airmed, the team had to sequentially work through steps beginning at a rough brainstorm of possibly applicable requirements, and then cyclically refining them. This was performed until a list of clear, unambiguous, consistent and prioritized list of requirements were generated. The steps followed are as follows:

1. **Discovering requirements:** At this stage the team is essentially in a brainstorm phase, where possible requirements based on the preliminary breakdown and features of the app is proposed.
2. **Classification and organization:** In this stage, the requirements generated from the prior stage are organized and classified based on their relevance to different functions of the software.
3. **Prioritization and negotiation:** Present requirements are ordered according to their priority. Contradicting requirements and those with negligible priority are recognized and removed.
4. **Recording requirements:** Through converting our list of requirements into user stories, cases and UML diagrams , the requirements were converted into a more accessible format allowing developers to more easily refer back to them. However we did not solely depend on this as UML only proves to be more efficient when all members of the development team are well versed in the language.

3.1. Requirements

Stages 1 through 4 of the process described above were conducted utilizing the Moscow method [2]. The primary difficulty faced while generating and refining the requirements was to be able to prioritize the possible functions and features; distinguishing the fundamental aspects of the app to ensure its success from the features that would be simply enhancing the experience but were not indispensable to the apps functioning. This is important in order to make the development process more efficient and allowed the more effective planning of the sprints, mentioned in Chapter 2.

The refined requirements are as follows:

Must:

- Include a navigation feature that allows users to easily transit between the 3 main features
- Include 3 main features: Calorie Counter, Weight tracker, Fitness assist
- Include a function allowing user to redirect to the main menu without losing any inputted data
- Store all data inputted by the user
- Save progress; save data and load for relevant users
- Let the user choose which feature to use

- Include a platform allowing members to interact with each other

Should:

- Include features that are capable of working independently
- Let user specify which muscle group they would like to focus on to train
- Provide detailed descriptions of suggested exercises
- Provide easily accessible references to videos showing the exercises
- Aggregate data over time to provide graphs displaying user's progress over time

Could:

- Include a body profile displaying all the different muscle groups that the user can target when exercising
- Use user's BMI to determine appropriate daily calorific intake
- Provide recommended water intake
- Keep track of repetitions and sets count for user's workouts
- Aggregate data over time to provide graphs displaying user's progress over time
- Let the user specify what their primary goals are

Won't

- Include a bar code scanner for users to easily log calories of food consumed or include a basic database of calories for basic foods
- Suggest a diet plan
- Provide a workout schedule for the user to follow
- Interacts with the user through reminders or notifications even when the app is not in use

3.2. Showcase Material

3.2.1. User Stories

User stories are short scenarios composed by the team that are vague enough to encompass all the main functions of the app. The first three user stories below cover the three main features of Airmed. The remaining simply point to additional functions that the application is capable of.

3.2.1.1. User Story 1

User Statement: *As a user, I want to be able to find different training exercises for different muscle groups, so that I can train effectively*

Acceptance Criteria

- A menu that displays all major muscle groups, allowing user to pick which muscle group they want to train
- Comprehensive list of exercises targeting specific muscle groups
- Contain input option to track sets and reps during training
- Graphical representation of progression over certain period of time

3.2.1.2. User Story 2

User Statement: *As a user, I would like to input my daily nutritional intake so that I can set goals (personal goals) regarding my body composition*

Acceptance Criteria:

- Calorie tracker, daily calorie intake can be added to app record as an input
- Graphical representation of daily calorie intake over time, allowing user to easily note when there are issues or strange fluctuations in their diet.

3.2.1.3. User Story 3

User Statement: *As a user, I would like to track my weight over a certain period of time so that I can see if I am reaching my personal goals*

Acceptance Criteria:

- Contain an input option to track fluctuations in weight over extended periods of time
- Visual representation of weight over time

When comparing the above three user stories to the Use Case diagram generated, shown in Figure A.1, it is visible that these user stories are in line with the diagram and in fact correspond to the app's three main features: fitness, nutrition and exercise.

3.2.1.4. User Story 4

User Statement: *As a user, I would like to register to the Airmed application and proceed to use it multiple times consistently*

Acceptance Criteria:

- Have a registration page that allows users to make a personal account connected to their email
- User must be logging into the account they registered every time they use the app henceforth
- The account contains any data input during prior use

3.2.1.5. User Story 5:

User Statement: *As a user, I would like to see posts from other users and make some posts about my out fitness journey*

Acceptance Criteria:

- Have a page that displays posts made by other users
- A function allowing user to post information themselves
- The possibility to delete or edit any posts made easily
- All posts are dated and associated with an account holder or the Airmed admin

3.2.2. State Diagram

State diagrams are a method to display the behavior of the software through finite state transitions. Such a diagram was generated with respect to Airmed (shown in Figure A.2), which makes interpreting states, transitions and outcomes more comprehensive.

3.2.3. Activity Diagram

Activity diagrams allow developers to represent the flow of execution and put emphasis on clarifying the sequence and conditions of flow through the software. Such a diagram was constructed in the development process of Airmed (shown in Figure A.3), allowing the decisions and different stages involved in the application to be visible. In addition to this, it can be used accompanying the previously generated user stories to confirm their respective acceptance criteria.

4

Software Architecture

4.1. General Implementation

The implementation of the tools Airmed offers, will happen with use of the python programming language. The intent is to launch a website that grants access to potential users. Django was chosen as the web application frame work to create the website. On the website the user will create an account. This account uses a username and password, as well as an email address that can be used to reset the password in the future sprints. Furthermore the user can choose to input specific personal data regarding their age, height and weight, that is used in determining the main functionalities of the program. This data will be stored in a Structured Query Language based database, that via Django, readily communicates with the website.

4.2. Implementation of Airmed's Functionalities

All of the tools Airmed offers will be programmed in python. The calorie counter, weight tracker and the basic exercises for specific muscle groups will be implemented independently, as these functions currently don't depend on each other. This minimizes coupling. The introduction of additional functionalities is postponed, until a later date after the launch, to ensure that a working implementation is out as quickly as possible. Emphasis remains on producing uncoupled software that can be added, to the already existing application, with ease.

To access a feature, the user needs to share specific personal data. This data will be stored in a database for all functions to use. The website will request the data when the user first initializes his account. The Django framework conveniently ties the python back-end, the website and the database together. The functions provided follow a general format: the user can input data, which updates the already existing data (either the data passed during initialization or zero). The current data point added will be appended to the existing log. This log is used to create a graph that shows progress over time. This set up allows to easily add additional features in the future, as long as they follow the same structure.

4.3. Django

Django is an open-source, python based web-framework intended to make development as easy and quick as possible, ensuring low coupling and offering many functionalities, like a built in web-server, JSON and HTML support, as well as the python unit tests module. Therefore it was a very suitable framework to service Airmed.

Django's design pattern easily abstracts the interaction with the website, using models and managers to handle specific data instances. Models correspond to specific data fields in the database. These models are

organized by so called managers that interact with the database to store or retrieve data. Both models and managers are implemented as Python classes in Django and only require to work with class methods to handle object attributes to manipulate user data. Relational databases use tables and predefined schemes to organize data. The database system that Django uses is based on a structured query language, where each model corresponds to a table in the data base and each attribute to a data field.

Views manage the website response. A view is a simple object that takes the request as input and returns a HTTPResponse instance containing the data and HTML formatting. Django makes use of two types of views, class-based views and function-based views. Initially the project will predominately use function-based views due to time constraints. Forms were used as a structured way to indicate to views what information is required to be entered by the user.

Finally Django does automatic testing of many of its features. It ensures that the communication with the database and the handling of requests are functioning without making additional testing necessary. For the implementation of individual features it also provides a unit test module, to be used in cases where specific implementations need to be tested. However a lot of areas that require unit testing are already covered by Django, as it automatically checks user inputs for validity for example.

A typical Django use case would be: a user accesses the website, enters his account name and password. Django retrieves the data associated with his profile, in the form of class instances (models), when the user accesses specific features and decides to change the data (e.g. inputting new data), the models append or extend the data and a manager object updates the database with the altered attribute of the model. The data that needs to be returned, gets fed into a view that initializes the website response. Since Airmed is a simple application, the implementation of its features with Django is fast and convenient. Additionally the simplicity allows the team to focus on and prioritize other parts of the software, like introducing new features and adding more functionalities, instead of managing tedious website and database requests separately. This simplicity is ultimately what makes Django the most convenient framework to handle this kind of application. More and detailed information about Django's architecture can be found in the Django documentation [1].

In order to provide a better understanding of how the team incorporated the main classes offered by Django in the architecture of our software, a UML Class diagram is shown in Figure 4.1.

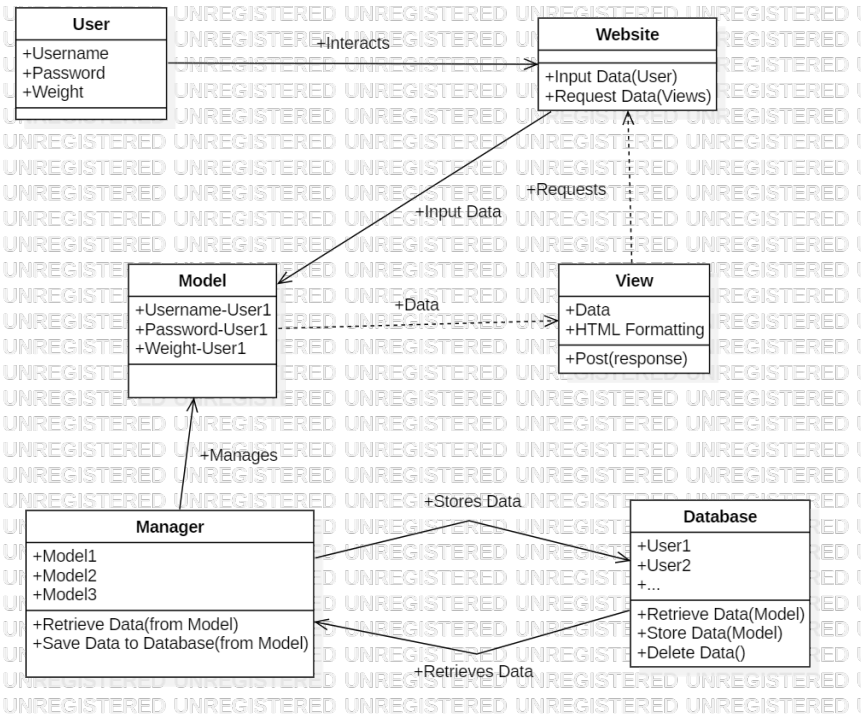

Figure 4.1: UML Class Diagram of the generalized architecture

5

Software Testing

Testing plays a highly important role in the development process of every software. In order to deliver software of good quality, it is necessary to perform as many tests as possible. Preferably, these tests are automated and are coded as soon as a certain function/class/feature of the product is implemented.

Due to the fact that we used Django as the framework for our website, some automated tests have been performed by default. As an example for such a test, we might consider the automated way of verifying whether the password that is entered by the user is deemed acceptable when they want to register on the website. As it can be observed from Figure 5.1, a number of conditions are imposed beforehand regarding the content of the password. If any of these conditions is not satisfied, the website will display the unsatisfied conditions in red and will expect the user to input a correct password, as shown in Figure 5.1.

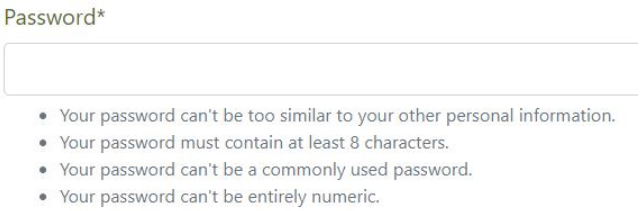

Figure 5.1: Conditions imposed by the framework when choosing a password.

If any of these conditions is not satisfied, the website will display the unsatisfied conditions in red and will expect the user to input a correct password, as shown in Figure 5.2.

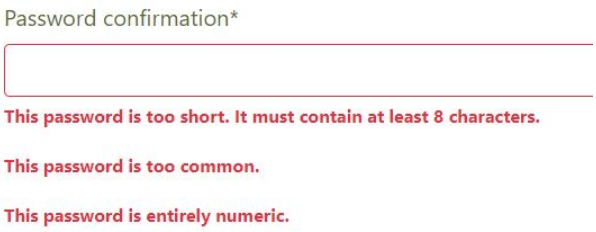

Figure 5.2: What the user sees when the password chosen does not satisfy the conditions imposed.

Another example is the fact that Django automatically does not allow the user to use the letters on the keyboard when typing in an integer numbers field. In this way, it avoids errors.

Taking into account that Django already performed some automated tests on our software and the fact that the product that we developed was not very complex, we decided that our testing strategy would be to use

mostly manual tests. During our testing procedure, we followed the path of *Gray-box testing*, as we knew some but not all of the internals of the methods we were testing (e.g. we knew the methods that we coded, but not the predefined methods that came with the Django package). Furthermore, the main testing method that we focused on was *acceptance testing*: through our tests, we made sure that every user story/requirement is satisfied by our software. A traceability table that proves the previous statement is shown in Figure 5.6.

5.1. Coverage

A coverage report for the main modules in our software was created, as shown in Figures 5.3 and 5.4.

```
C:\Users\Flori\group-25\mysite>coverage run --source='.' manage.py test
nosetests --with-coverage --cover-erase --cover-package=mysite, airmed, users --cover-html --verbosity=1
Creating test database for alias 'default'...

Name                                                    Stmts   Miss  Cover
--------------------------------------------------------------------------
airmed\__init__.py                                          0      0   100%
airmed\admin.py                                             3      3     0%
airmed\apps.py                                              3      3     0%
airmed\migrations\0001_initial.py                           8      0   100%
airmed\migrations\__init__.py                               0      0   100%
airmed\models.py                                           11     11     0%
mysite\__init__.py                                          0      0   100%
mysite\settings.py                                         25     25     0%
users\__init__.py                                           0      0   100%
users\admin.py                                              6      6     0%
users\apps.py                                               5      5     0%
users\migrations\0001_initial.py                            7      0   100%
users\migrations\0002_auto_20191012_1426.py                 5      0   100%
users\migrations\0003_auto_20191012_2300.py                 4      0   100%
users\migrations\0004_auto_20191013_1945.py                 4      0   100%
users\migrations\0005_auto_20191013_2006.py                 7      0   100%
users\migrations\0006_auto_20191013_2007.py                 4      0   100%
users\migrations\0007_auto_20191014_1209.py                 4      0   100%
users\migrations\0008_auto_20191015_1332.py                 7      0   100%
users\migrations\0009_exercise.py                           7      0   100%
users\migrations\0010_auto_20191015_1716.py                 4      0   100%
users\migrations\0011_cable_fly.py                          7      0   100%
users\migrations\0012_inclined_db.py                        7      0   100%
users\migrations\0013_bent_over_row.py                      7      0   100%
```

Figure 5.3: First part of the coverage report.

From Figure 5.4 it can be seen that our software has a coverage of 38%. This might seem to be a low coverage percentage, however, if we look at all the files that were tested, we observe that most of them had a coverage percentage of 100%. This is due to the automated tests performed by Django. It can be seen that the file that lowers the total coverage percentage in a drastic way is the *users\models.py*. This is due to the fact that no automated tests are performed by Django on this file, as the content of the file was coded by the team and not predefined by the framework. In order to improve the overall coverage of our software, the team decided to perform manual tests that accounted for the *users\models.py* file as well as other files with very low coverage.

```
users\migrations\0013_bent_over_row.py                      7      0   100%
users\migrations\0014_bicep_curl.py                         7      0   100%
users\migrations\0014_deadlift.py                           7      0   100%
users\migrations\0015_pullup.py                             7      0   100%
users\migrations\0015_skullcrushers_tricep_extension.py     7      0   100%
users\migrations\0016_back_squat.py                         7      0   100%
users\migrations\0017_lunge.py                              7      0   100%
users\migrations\0018_hip_thrust.py                         7      0   100%
users\migrations\0019_merge_20191017_1843.py                4      0   100%
users\migrations\__init__.py                                0      0   100%
users\models.py                                           157    157     0%
users\signals.py                                            9      9     0%
--------------------------------------------------------------------------
TOTAL                                                     354    219    38%
----------------------------------------------------------------------
```

Figure 5.4: Second part of the coverage report.

In order to create the coverage report, the following lines of code displayed in Figure 5.5 had to be implemented in the *settings.py* file. This was done after installing the *django-nose* package and adding it to the list of installed apps.

```
# Use nose to run all tests
TEST_RUNNER = 'django_nose.NoseTestSuiteRunner'


NOSE_ARGS = [
    '--with-coverage', #states the fact that we want a coverage report
    '--cover-erase', #removes the gathered data about the coverage report
    '--cover-package=mysite, airmed, users', #specifies that we wish to obtain a coverge
                                             # report for the main modules in our software
    '--cover-html', #deliveres a more readable report
]
```

Figure 5.5: Setting up the coverage report.

5.2. Overview of Tests

Table 5.1 displays an overview of the tests executed by our team. Furthermore, from Figure 5.6 it may be observed that the tests executed by our team ensure that all the user stories are covered.

User story	Tests executed to ensure user story is satisfied
1	1), 9), 10), 11)
2	1), 12), 13)
3	1), 14), 15)
4	2), 3), 4), 5), 16)
5	6), 7)

Figure 5.6: User stories and the tests that provide coverage for each user story.

Table 5.1: Overview of all tests, automated or manual, written and/or executed by the team

Short Test Description	Type	Team Member
1) **First/Home page**: tested that the main features are correctly displayed on the navigation bar and that the user can access the desired feature by clicking on its name	manual	Sara Boby
2) **Registration**: tested that the website shows an error for the fields where the input inserted by a new user is non-existent or inconsistent with the rules imposed and displayed on the website	manual	Florina Sirghi
3) **Profile**: tested that the user is able to access their profile after registration and that the profile page can be updated with the information and the profile photo shared by the user	manual	Jack Cook
4) **Login**: tested that a user is able to log in if they provide the correct combination of username and password	manual	Phillipe Lothaller
5) **Login**: tested that if an incorrect combination of username and password are provided, the page will show a warning	manual	Phillipe Lothaller
6) **First/Home page**: tested that the welcome message together with the messages of other users are present on the main page when the user accesses the website	manual	Paul Simon Schön
7) **Blog post feature** tested that the user is able to see the posts of other users and to make a new post after they log into their account	manual	Florina Sirghi
8) **The three main features**: tested that, if the user is not logged in, they will be instructed t do so first before having access to the content of the fitness, calorie-counter ad weight-tracker features	manual	Jack Cook
9) **Fitness feature**: tested that the main muscle groups are displayed correctly on a list and the user may click on one to select the desired one and it will take them to a separate page where a list of exercises is displayed and you may select one	manual	Sara Boby
10) **Fitness feature**: tested that on the individual page of each recommended exercise the user may input the number of reps, sets and one rep max achieved that day	manual	Paul Simon Schön
11) **Fitness feature**: tested that the progress of the one rep max achieved by the user overtime for each exercise is displayed in an informative graph	manual	Sara Boby
12) **Calorie-counter**: tested that the user is able to input the amount of calories consumed multiple times throughout the day and that the total daily amount of calories is calculated correctly	manual	Florina Sirghi
13) **Calorie-counter**: tested that the users's progress with regards to the daily calorific intake is displayed correctly, in the form of an informative graph	manual	Jack Cook
14) **Weight-tracker**: tested that the user is able to input their weight	manual	Paul Simon Schön
15) **Weight-tracker**:tested that the progress of the user's weight is shown in the form of an informative graph and that if the weight is provided multiple times throughout the day, the value displayed on the graph would be the last one provided on that day	manual	Phillipe Lothaller
16) **Log-out**: tested that when the user logs out, they see a message informing them that they have successfully logged out and an option to log in again	manual	Jack Cook

6

Reflection and Adaptation

6.1. Paul Simon Schön

This project was the first time I had to code in collaboration. Usually short scripts, that make up most of my programming experience, were used to plot or manipulate data and thrown out after. This was the first time I shared code with other people. I really appreciated being introduced to Git early on. Coming from a rigid engineering discipline, the agile manifesto was definitely new to me. I do believe that I'm going to come back to it, not just in programming, because it seems very useful for prototyping or whenever the design needs flexibility to meet the constraints. This novel way of approaching a group software programming exercise taught me a lot. It clearly showed how Scrum methodology can be very useful for structuring small groups, ensuring that everyone knows what needs to be done, continuously reflecting on the work that has been done, whilst mapping out the course to take for the next weeks, when necessary and not sooner. The model strongly contrasts with having a fixed plan that becomes less and less viable the further the project progresses. Coming from aerospace engineering where the waterfall model is more or less the industry standard, this was definitely a valuable new perspective on how to approach design.

6.2. Sara Boby

As most people, I joined this minor with the slightly clouded judgement that I would spend 6 months coding and become an expert at software development. Quite obviously, I was very wrong. Software engineering methods was one of the courses that made it most evident that software development had a significant mount of supporting theory, regarding the development process one chooses to follow; more importantly, it made evident that there was extensive planning and decision making involved in the preliminary stages, before developers even start coding. The importance of performing scrum planning and incorporating agile methodology in the ultimate success of the software. However this is not to negate the importance of an airtight source code. The interactive lecture conducted on code testing was one of the most interesting lessons to me and the professor showed us the various ways Pycharm has built in functions that pick apart you code. Furthermore, the concepts we needed to apply in the actual implementation of our software fell in parallel with what we'd been learning in Algorithms and Data Structures, regarding object oriented programming and handling data; this coherent nature of the courses in the minor made it enjoyable to finally have been given a project to apply everything we've been learning, with freedom and independence. Overall, I thoroughly enjoyed the project and for the future hope to be a bit more confident in my programming skills and have a change to use Django myself.

6.3. Florina Sirghi

Before I started this minor, I thought Software Engineering was only about building large amounts of code. After I followed the Software Engineering and Methods course, I found out that it was about so much more than simply coding. Of course, you can not have a valuable software without a high quality source code.

However, this course helped me understand that there are other important steps that need to be taken in the software development process. The first one is to focus on requirements engineering. This part really helped me understand how to help organise the software for our product that we were building during the project/lab hours and to have a clear overview of what was left to do at all times. Furthermore, I learned that testing the software is a crucial step towards developing a product of high quality and high reliability. In addition to this, I liked the fact that this course also included a project. This allowed me to apply the theory learned during the lectures and to gain more experience in the Software Engineering field. It also helped me get started with Django and I definitely see myself using this framework in future projects. In order to pave my way for becoming a successful software engineering, I believe that it is important for me to remember to always keep an agile mindset- to be open to change. Furthermore, I will have to focus more on writing automated tests and I aim to test my code as thoroughly and as efficiently as I can to ensure that I create high quality software that will satisfy the user.

6.4. Jack Cook

We started our project at the beginning of the semester with some experience in coding but very little in the software engineering process. This gave me a belief that software engineering was simply thinking of an idea and executing the code. This was soon found to be a slow and ineffective way of working. As new ideas of structuring the project were taught, for example SCRUM, were introduced the project moved along much more smoothly. The use of aids such as state diagrams meant that the functionality of the app was always clear and could easily be referred to when functions were being written. All through the project we kept to an agile way of working, for me this was one of the most valuable techniques to learn. When I was working on any piece of code it was always good to know that if big problems were to arise I could consult with the team and we could adjust our approach. For my future projects I would always use agile techniques. I also experienced pair programming at different points in the development, as both the pilot and co-pilot, I found that this worked very well for me and would definitely use it again. One disadvantage of pair programming was the speed of working, it could be quicker for the two programmers to work independently on different parts of the app. I think that personally in the future I would also always keep in mind that simplicity is better than complexity as it was easy for me to get carried away in this project in adding new features.

6.5. Phillipe Lothaller

During the course of this project I have learnt that software engineering is so much more than just sitting in front of a computer screen and typing and programming away at a software you are trying to create. I found that software engineering is a delicate process that, if applied correctly, will leave the engineering team with a successfully produced software that is completed on time and will leave the customer satisfied. It was clear that during the beginning phases of the project when we were not so accustomed to applying the software engineering methods, we weren't able to put up a good amount and good quality code. As the process continued and we became more familiar with the process and applied them more diligently, our software also improved (i.e more and better code was produced.) During the process I also noticed how the flexibility of the software engineering process can be both advantageous as well as disadvantage the team. If applied correctly and in time, it was easy to implement new ideas that arose over the course of time, however it also often happened that this added feature interfered with existing features, which meant we needed to spend extra time on fixing the software. I believe to become a good software engineer in the future, I just need to continue applying the different software methods (for example applying sprints and creating UML's) to future projects, no matter how small the project. The more the process is applied, the more comfortable one becomes with it, and the better one will understand what works well for you and what doesn't work so well.

7 Conclusion

During the past quarter, working on the project allowed the team to gain more insight, as well as hands-on experience in Software Engineering. After just seven weeks, with little previous experience in the field, we managed to deliver a functional website that provides the user with features that would help them get started and advance in their health journey.

In the future, we hope that Airmed will be used by a large amount of people and that they will be satisfied with the services provided by our product. Naturally, we would like to improve and expand our software. With the help of Django, the team wants to improve the architecture of the software by creating a class-based view for the recommended exercises forms.

Furthermore, some of the existent features would benefit from changes. For example, for the fitness advice feature, instead of the user having to choose a muscle group from a list, they will be provided with a diagram of the human body (similar to the one in Figure A.4). When the user hovers over a muscle group on the diagram, the chosen muscle group will change colour. By clicking on it, the user will be redirected to the page that includes the recommended exercises to train the specific muscle group. In addition to this, more muscle groups and recommended exercises will be added to the website.

Other features that would be implemented in the software if the team could spend an additional quarter on this project would be a water intake tracker, as well as a sleep tracker. The water intake tracker will allow the user to input the amount of water they had in that day. A number of glasses equal to the recommended daily intake of water will be displayed on the screen. The glasses will be empty at first, but they will become full when the user tracks their water intake. The glasses will all be full once the user reaches the recommended daily intake. Regarding the sleep tracker, this will be a feature that allows the user to input the amount of hours slept every night and information about the quality of their sleep, as well as provide the user with helpful tips on how to achieve better sleep. It will also include an option that allows the user to choose if they want to receive a notification that would inform them when they should go to sleep. This feature would be useful for people with busy lifestyles, sleep disorders, as well as people who travel often and suffer from jet lag.

Appendices

A

Supporting Material

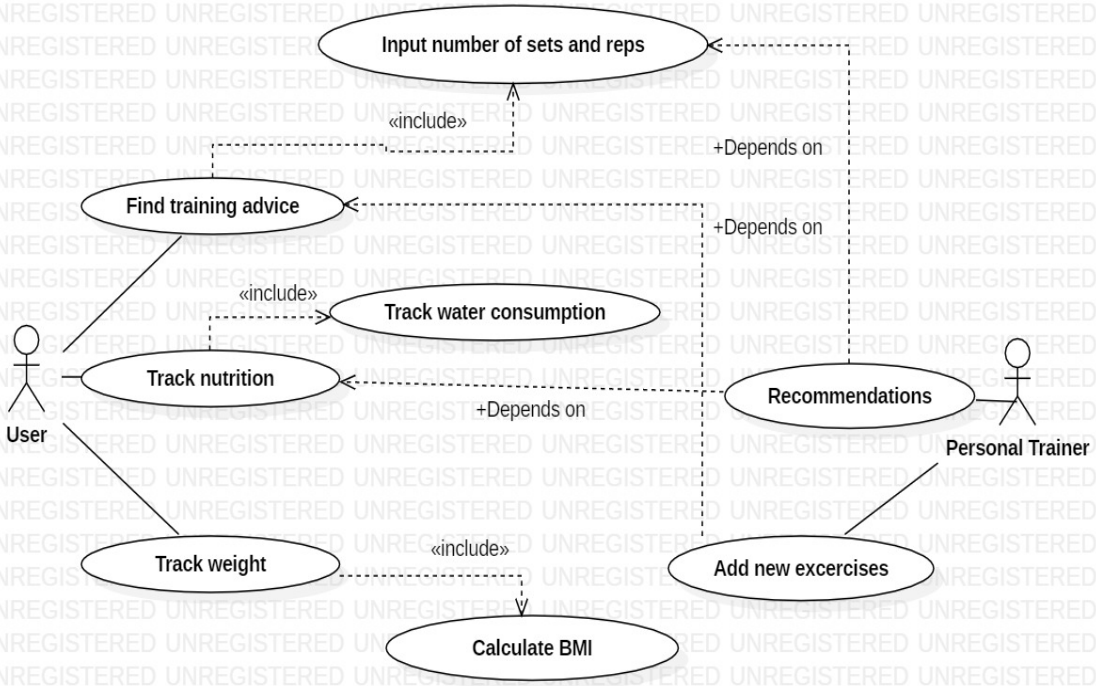

Figure A.1: Use-case diagram

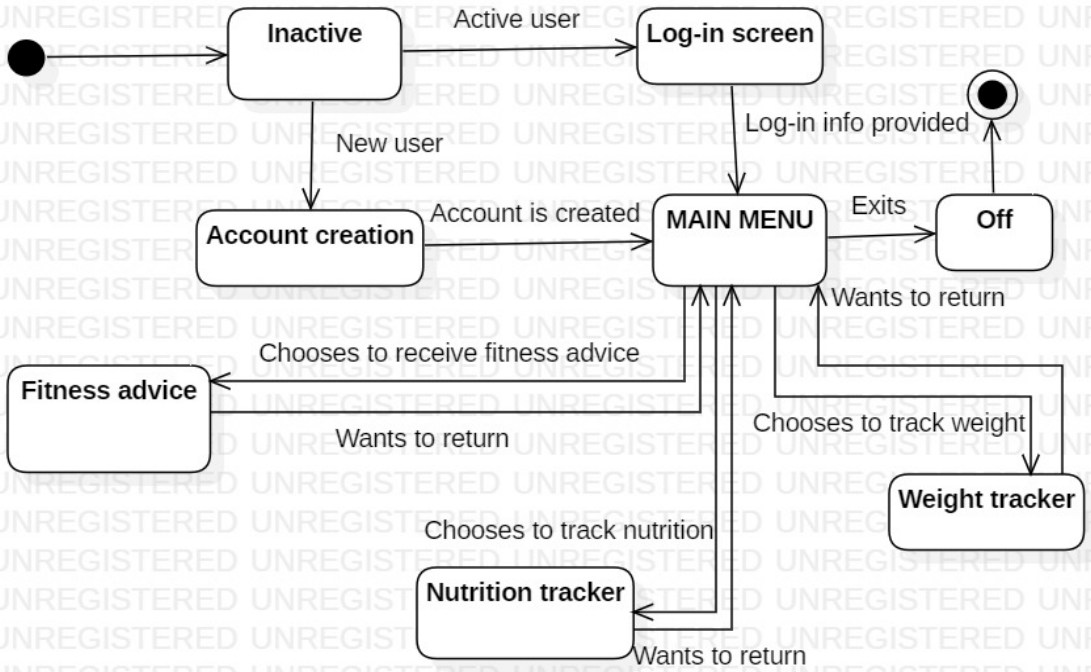

Figure A.2: State chart diagram

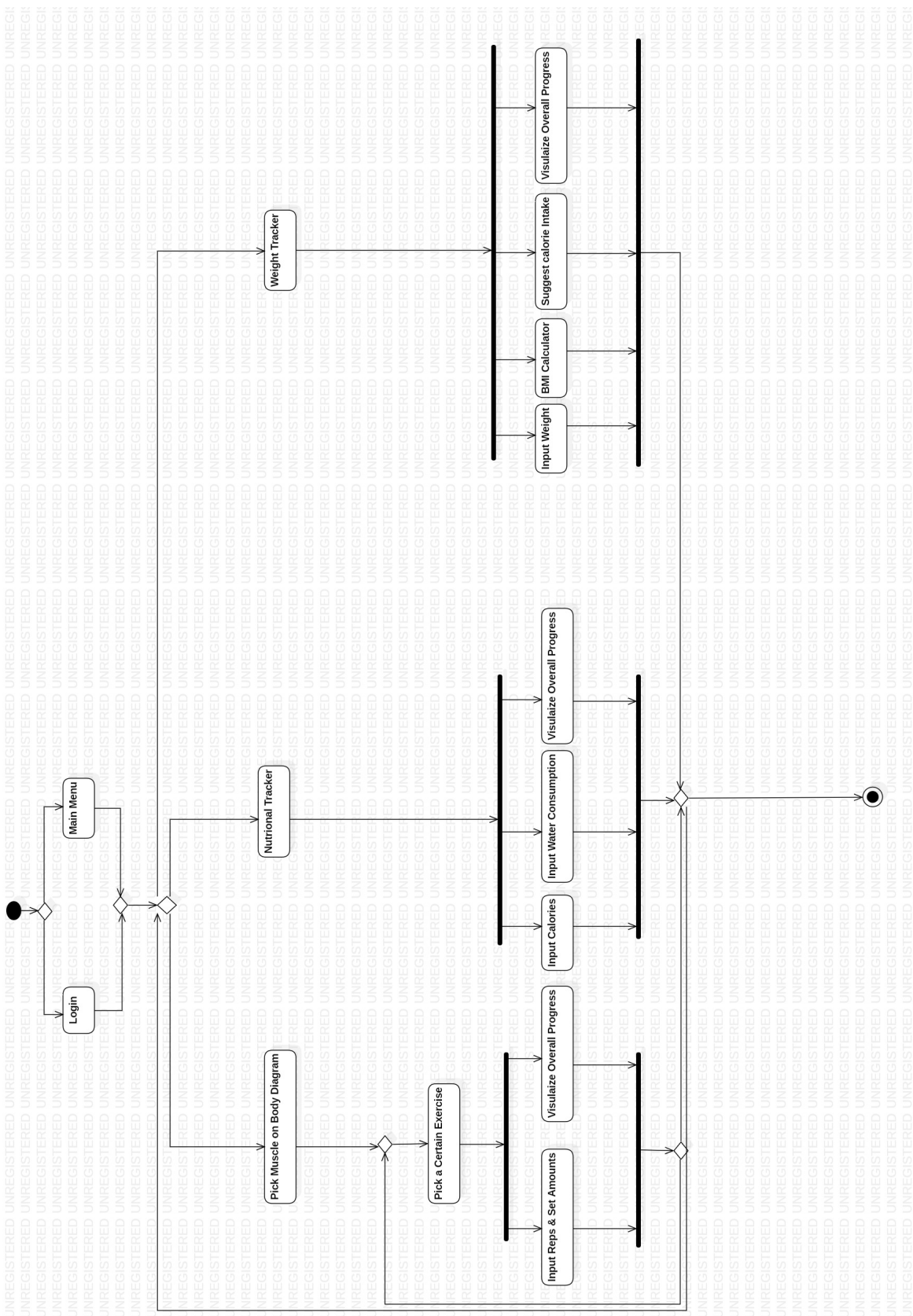

Figure A.3: Activity diagram

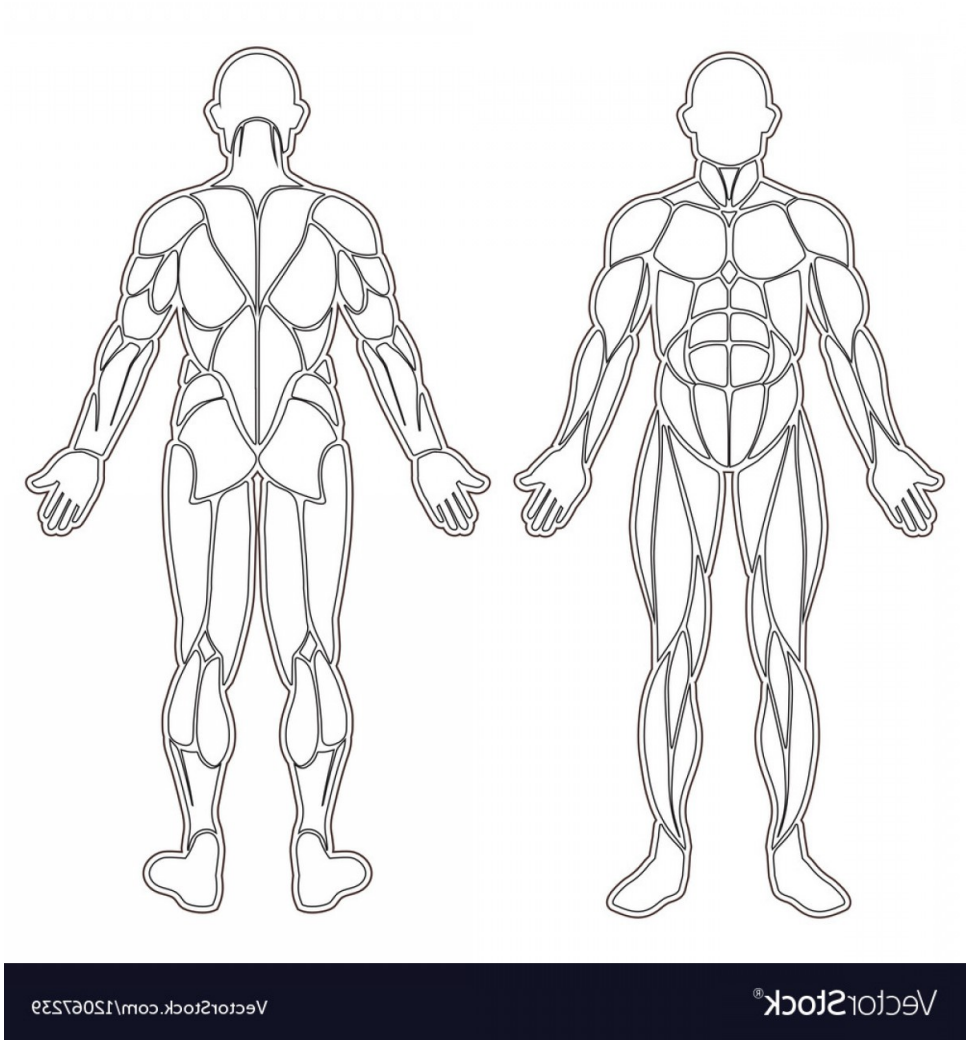

Figure A.4: Diagram of the human body intended to be used in the fitness advice feature.

Bibliography

[1] Django Documentation. https://docs.djangoproject.com/en/2.2/.

[2] Rod Stephens. *Beginning Software Engineering*. John Wiley & Sons, 2015. ISBN 9781118969144.