INTRODUCTION

The aim of this project was to make fitness and healthy eating more enjoyable and easier for consumers. We have created a website called Fitagotchi which is inspired by the virtual pet game, Tamagotchi. It is a fitness website with personalised user profiles, to save and keep track of your progress and goals. The idea of the project combines a BMI calculator and healthy meal plans along randomly generated motivational quotes, to keep the user on track. Fitagotchi provides a solution to more than one problem, it is designed to allow the user to unwind and have some fun, have easy access to healthy meal recipes, be able to enter and keep track of fitness goal(s). This report will consist of a background of the thought process of the project, specifications and design, implementation, execution, as well as testing and evaluation of the application.

BACKGROUND

Worldwide obesity has tripled in the past 50 years[1] – this has caused increased demand for effective weight loss methods given the associated health concerns associated with obesity. Technology has proven to be a beneficial platform in supporting weight management and the growing obesity epidemic. A meta-analysis carried out by Mateo et al. found that weight management aided by technological means such as phones showed substantial changes in weight and BMI when compared to control groups.[2] This was also found to the case in a study conducted by Ross et al., which demonstrated that activity-monitoring smartwatches in combination with smartphones resulted in 44% of participants achieved significant weight loss, in comparison to the 15% using conventional methods.[3] Another study analysing the eating habits in children has concluded that phones with built-in motion sensors showed substantial improvement in fitness and mental health.[4]

The links to the main aim of our web application, which is to make fitness more enjoyable, and have all your fitness needs in one place. The web application is called Fitagotchi, and is inspired from the Tamogotchi theme. The colour scheme of the website includes yellows, blues and blacks, to stand out and improve the users mood. There are multiple gifs and cute images of our Fitagochi mascot. The main homepage is also inspired from Tamogotchi, as shown through the three hyperlinks at the bottom, representing the three buttons on a Tamogotchi. We've chosen a yellow background with white buttons as the images used on the page also utilise those colours which makes for a more cohesive looking website. The 'Fitagotchi' logo is displayed in the 'DynoPuff' google font – this font was chosen as it provides the page with a game-like effect, similar to the Tamagotchi game.

The user can create an account on the website, where it will store their details within the database. They can input their height and weight, and it will calculate their individual BMI. It then advises the user on the status of their BMI, clearly indicating whether they may be deemed underweight, or overweight. The user then has a drop down menu which enables them to enter the goal they would like to achieve based on their BMI results. The three options are to gain, lose or maintain weight. All these results are saved within the SQL database, and are easily accessible for the user whenever they please. The database is connected to the Python program through the mysql database connector.

There is a meal planning page which asks the user what they would like to eat. After inputting a certain ingredient, they are given two randomly generated meals based on their chosen ingredient, and when they click the link they are directed to the website of that recipe. This is a fun way to choose what meal they would like to eat depending on what they are craving or which ingredients they have readily available. They are given two randomly generated meals, which enables decision making for the user a lot easier, and provides them with variety without being overwhelming. These recipes are accessed using the Edemam API.

There is also a fitness page, which pulls various exercises from a fitness API. The user has an option to choose which muscle group they would like to target and are returned with various exercises they could do, either from home or with limited equipment. This enables the user to keep active, and provides instructions on how to safely and effectively complete the workouts.

To ensure Fitagotchi is fun and also allows the user to unwind and provide motivation, the user is prompted with random motivational quotes on every page. The randomiser function greets the user with a motivation quote each time the user signs in.

SPECIFICATIONS AND DESIGN

Requirements technical and non-technical

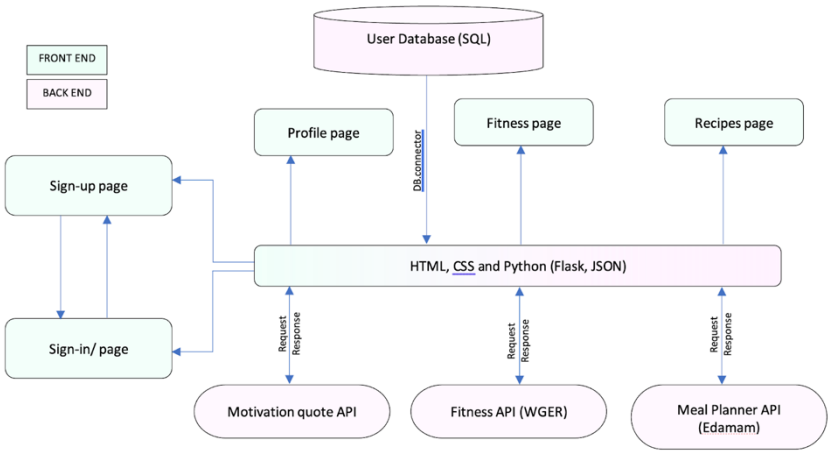
The program is required to have user accounts which have the ability of storing, displaying and updating the user's details. The user's details will be stored in a table in a MySQL database, with the database connecting to the program's code in Python through the MySQL connector tool. The user's name and login information will initially be inserted into the database when the user creates an account on the signup page, with the user's height and weight being added to the database later through their user profile. Through various Python functions centred around user management the program will be able to add new users, update their information, check for unique email addresses - to ensure no duplicate accounts can be created - as well as pushing a users details to their own personal profile page. The web application must also have the ability to redirect between web pages depending on whether the user is signed in or not.

The web application will be able to provide the user with a randomised recipe. The program will allow the user to enter the main ingredient that they would like in their meal. A recipe API will be utilised by filtering recipes to reduce the options to low-fat, in order to fit in with the health aspect of the program. Through a Python function the user input will be combined with the recipe API, with the program returning two randomised recipes to the user containing the ingredient that they had entered.

The program will provide the user with a workout. Employing a fitness API, the program will be able to output a variety of exercises the user can perform in order to have a complete workout. For this, there will be no user input, but rather a function in Python will be combined with the API to pick a number of exercises and automatically output these to the user.

Design and architecture

System Architecture diagram:

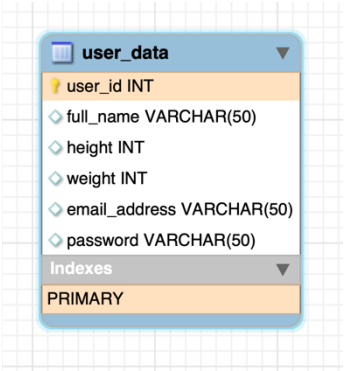


The first page that appears on our website is the sign-in page. The user can enter their email and password, which will allow them access the profile page. There is also a link on the sign-up page that allows you to sign-up if the user doesn't have an account. This page is written in HTML as it allows for a user-friendly interface. The sign-up and profile page both include a motivational quote in the header of the page, to keep users motivated throughout their fitness journey. This randomised quote is accessed through a motivational quote API using JSON requests in Python, which is then linked in the HTML profile and sign-up page.

Once the profile page has been accessed, options for a fitness page and recipes page can be opened through buttons at the base of the screen. The profile page includes user input for name, email, height(cm), weight(kg), and fitness goal (based on the user's BMI calculated in python). The height and weight is saved automatically to prevent having to repeatedly input this data. The Python code used for BMI was weight(kg)/height^2(cm^2). Both the height and weight are saved to the users profile so it won't require repeat inputs. This user info is stored in the 'Fitagotchi' database in the user_info table that stores each user's full name, email, height(cm), weight(kg), and password. This was written in SQL and the data was called using db.connector in Python.

The fitness page allows for access to daily randomised workouts specifying the area of the body that is trained; this data is retrieved from the WGER API. This is to encourage users to complete varying daily workouts which prevents them from losing interest. The recipes page retrieves data from the Edamam API which allows the user to input an ingredient of their choice, and outputs two randomised recipes to choose from that include the user-specified ingredient.

Database design:



The Database Schema shown is kept relatively simple as it is only required to store user data. We wanted to avoid overcomplicating this aspect as we only needed limited information stored, and this was easiest to achieve through a single table that unified the user profile information. The auto-increment function was used for creating the user_id primary key column, to avoid having to manually insert this information into the user_info table; this also makes it simpler to insert new users into the database and prevent duplicate entries. This is also why the user_id column was chosen as the primary key for the table - this column is unique and can be used to link to other tables that may be added to the database in the future. Height and weight data needed to be input as an integer (INT), given that this information is needed to calculate the BMI in Python. All other fields were input as variable characters (VARCHAR) with a limit of 50 as the data being input is not anticipated to go beyond this limit.

Classes and Try/Except

Classes were also used in our code, as can be seen in the fitness and meal planner implementation aspect in Python. The 'Fitness' class utilises functions to get the random exercises, exercise categories and exercise data. The 'Recipes' class uses classes to get recipes and recipe data for specific ingredients, as well try/except in the recipe functions -where the exception is triggered if the user enters an ingredient into the search function that cannot be found in the API.

IMPLEMENTATION AND EXECUTION

Development approach and team member roles

Ayesha had written code to calculate the BMI of the user using their height and weight. This was done using a python file and connecting it with the web page. She had used Flask to create our web pages, and decorators were used to tell our application which URL is associated with each function. She had also used HTML files to create the basic web pages relating to each page of the application. She had researched multiple API's for our project, along with other team members to find which ones would be more suitable to achieve our goals with this project. Ayesha had made sure to participate and be present in group meetings and sprints, as well as contribute to multiple sections throughout this group report.

Nazo's contribution to this project includes creating the user database and finding and implementing the fitness API into the website. She was also part of graphic implementation, organising most of our meetings, and was responsible for the project architecture diagrams, system limitations, classes, and research for the project background. She also wrote helped write the testing aspect of the project.

Olivia contributed to the project by connecting the user database to the program in Python, the management of the user profiles, creating decorators which are used in the web application to restrict user movement throughout the site, and implementing the website's memory in the form of cookies which are set, stored and retrieved as needed through various parts of the website. he also wrote helped write the testing aspect of the project.

Akarsha's contribution to this project involved researching suitable APIs that were compatible with our software target of a nutrition and fitness application, hence the motivational quote API which generates a random motivational quote in the project to inspire users. Akarsha also contributed to the project documentation.

Tools and libraries
The web application was mainly created using Python. HTML and CSS files were used to create the web page, along with the Flask module. Flask is an API in python which easily enables you to make web applications. In our Python code, after installing Flask, a decorator was used to tell our application which URL is associated with each function. This enabled us to create multiple pages within one web application. Instead of using Flask, Django could have also been used. The database is connected to the Python program through the mysql database connector.

Trello was used to organise meeting goals and assign individual tasks and address outstanding backlog items :https://trello.com/b/2PWnzEUL/cfg-project .

Implementation process
The implementation process generally went well for the project, even allowing us the opportunity to add additional functionality to the application. Through the use of an API, we were able to add randomised motivational quotes to different pages of the website, providing the user with words of encouragement as they navigate through the site. We were also easily able to implement a user's BMI into the program, having it displayed on the user's profile page. This was done through a function that passed a user's stored height and weight, calculating their BMI based on the data. A goals feature was also included in this function, with certain BMI levels returning different goals, allowing the user to decide how best to use the app for their own personal goals.

The front-end aspect of our project went especially well. Using the Flask tool, we successfully integrated all aspects of our program into a fully functioning website. We were also able to have the aesthetics of the web application match the theme of a Tamogotchi, utilising bright colours and interesting fonts to bring a unique take on the standard fitness app. The appearance of the Fitagotchi web application will hopefully allow the user to consider health and fitness in a more relaxed and light-hearted way. Generally, the overall main aspects of the program were able to be implemented, although not necessarily exactly how we had originally imagined it.

There were a few changes that we made to our original idea for the Fitagotchi web application. Initially, as the name of our website suggests, we intended to have a Tamagotchi inspired game which was linked to the health and fitness aspect of our program. The Fitagotchi's mood and health would increase or decrease depending on the information that the user submitted - such as drinking water, eating a healthy meal or completing a workout. However, we found implementing this part of the program quite difficult, and eventually ran out of time to complete the coding for this to a satisfactory level, so we decided to remove this element from the program.

Additionally, for the randomised meal ideas, we originally had the idea to output recipes based on a user's inputted calorie limit. However, we struggled to find an API that would easily allow us to do this, so we changed it to the user being able to input an ingredient that they would like included in the meal suggestion instead. The final change we made was regarding the workout aspect of the application. The initial idea involved having the user input the number of calories they wished to burn, in addition to selecting a specific type of exercise - the program would then output to the user the amount of time to do the specified exercise for. Similarly, to the changed recipe idea, we struggled with finding a suitable fitness API for our concept, leading to us modifying it slightly so that the application creates a randomised workout for the user, consisting of a number of exercises they should complete.

Agile development
The project timeframe was from 15th July to 14th August and fitted well with the average duration of an average scrum sprint (4 weeks) and our team of 5 fitted well within general scrum team size between 3-9 members. For this project with its limitations, we took on the roles of both product owner, developer and business stakeholders. We selected the agile iterative approach so that we can focus on delivering value as fast as possible in increments. This means that we can split our software and product development process in multiple iterations and adjust, refine and review our product during the software development process and constantly adapt to changes in our scrum sprint.

We applied the PDCA (plan, design, check, adjust) cycle and started working on the project vision at our kick off meeting. In the sprint planning, we started the plan phase and our discussion on which product to develop, looked at tools, project requirements and resources. As our team all worked

remotely, we decided to use the Trello board as the project management tool and to have our daily scrum check-ins via WhatsApp group as this offered the most agile approach for this project. After deciding on the project goal, we took on the different hats during our meeting, analysed user stories and gathered details for our project vision and what stage tasks are defined as done. Before we started the coding work, we worked on the project architecture and researched available resources to complete the project delivery.

During the design phase, an analysis was conducted and we reviewed front and back end requirements and discussed business requirements, functional specifications and looked at tools needed for our project. Due to implementation challenges, we had to revisit our project vision and were quickly able to adapt and adjust our project vision and goals accordingly. A rough plan of the architecture of the software application was made, including the use of programming languages (i.e. HTML for frontend and python for backend linked to an API for SQL data), overall design, templates, boilerplates, user interface. We discussed methods of problem solving and testing, security requirements and evaluating tools for the project.

Tasks were prioritised and assigned to individual team members. We tracked the progress via the Trello board and kept each other up to date via our check-ins via our WhatsApp group. After we started working on our coding, we applied iteration testing to ensure the deliverable met the project requirements and was able to move backwards to other phases for further improvement or postponed areas for the next sprint, for example to incorporate digital pets onto our website.

We used Github both as version control and to share the completed code for each of our tasks. We planned to use it as a tool to help our group with refactoring and reviewing completed code sections before merging into the main branch. We also worked on and refined our backlog, comprehensively evaluating the work during the cycle and adjusting accordingly, like adapting the project to fit in for homework and the theory assignment. While the code was written using PyCharm.
This agile approach allowed us to be more flexible, quicker to adapt to changes during the project and overall become more innovative and modifiable.

Implementation challenges
We had few implementation challenges for this project. Our initial project vision was for a skincare and beauty product wikipedia website, however during the research phase we didn't have the relevant resources to complete the project within our project timeframe and quickly adapted our approach and changed the project output to creating a motivating fitness and healthy meal planner website.

During the coding implementation we experienced a challenge with external workload which reduced the available resources for our project (available time) and had to reprioritise our tasks. We reviewed the user stories and clarified what needed to be ready for this sprint and decided to postpone the digital pet element for the website for the next sprint.

The website was made by linking the API's and using JSON in Python. We created HTML files in Python, this also created additional challenges and we had to learn more about HTML as it was used as frontend of the application.

Another challenge was us working remotely, having other commitments outside of work and adjusting for holiday and work schedule. Agile methodology was not strictly followed due to short duration of project and unfamiliarity with project. However this enabled a more flexible approach to be undertaken with the varying roles.

Another implementation challenge we faced was finding relevant API's. Our initial idea for the meal API was https://api.calorieninjas.com/v1/nutrition?query= however we did not end up implementing this as this did not provide users with recipe information or meal ideas. Finding an appropriate fitness API was also lengthy, as many API's required subscriptions, or did not provide relevant or enough information.

TESTING AND EVALUATION

Testing Strategy

In a software development lifecycle, testing is a crucial aspect in order to determine a softwares functionality, intuitiveness and its reliability. We've written unit tests for the Fitness, Profile and Recipes page.

For the fitness page testing, we conducted two unit tests 'test_get_exercise_data', and 'test_get_exercise_category_data'. The first test 'test_get_exercise_data' tests that there is at least one exercise that is output when executed. We've chosen a random exercise (barbell lunges) to ensure the API is returning what we expect, and this random exercise exists within the data. We've then filtered the data and filtered it so that we're only looking at the ones with that specific exercise in, stepping through it until the matches are found. However if there were no matches for barbell lunges, then that would mean barbell lunges wasn't there, and the test would fail.

The 'test_get_exercise_category_data' receives data from fitness API, and calls category 14, asserting that the output is 'calves'. This test also necessary as it ensures that the correct data corresponding to their categories are output when exercises are displayed.

For the recipes API, only one test was required - test_get_recipe_data_for_ingredient - in which we tested the recipe data for a specific ingredient (in this case it was 'mushrooms'). Similar to the testing on the fitness API, we also used the filter to test whether the API outputs only the recipes with mushrooms in it. The API was working as expected and this test passed. The test also asserted that at least one recipe was output, with the specified recipe ingredient, which was achieved using the try/except statement.

For the profile page, we conducted two tests- one for testing the BMI code, using basic data to ensure it outputs the expected BMI, and the get_fitness_goal was for testing the three specified fitness goals, which should only display three specified options based on the height and weight information input. The self.assertEqual tests whether the height and weight data for calculating the BMI will result in one of the three specified options (Gain Weight, Maintain Weight, and Lose Weight), which it does.

For user management testing, the tests were more complex. We had carried out five different tests to test different user scenarios. As you can see both get_mock_db_connection and get_mock_db_cursor have an 'enter' and an 'exit', and the purpose of those is that if you go into user_management.py, the 'enter' is run when the 'with' block is started, and the 'exit' is for when you end the 'with' block. In other words, 'enter' is like a set-up, and the 'exit' is like a tear-down. The purpose of 'enter' is that when you open a 'with' block with the get_database_connection, it should set up a connection with the method called 'cursor', which allows you to generate the 'cursor'. This means that the 'enter' and 'exit' are used for the 'with' block, and the cursor is what actually does something when you try to do connection.cursor. It's not a real connection, so we have to ensure when we type .cursor it doesn't error out. This is similar to get_mock_db_cursor, because here again we are setting it up using a 'with' block, which means that we need some way of setting it up ('enter') and some way of tearing it down ('exit'). The cursor should be able to execute and fetch data. This executes through a query and some parameters we don't have a fake database for this kind of testing. 'Fetchone' and 'fetchall', can feed into the get_mock_db_cursor what we want it to give us. Depending on which one is being used, you can feed in some data that you want returned. It will pretend to be every object until the specified point, and then if cursor.execute doesn't cause an error, cursor.fetch will return us exactly the data we had requested.

When testing the available and unavailable emails, we've patched it with mock_db_connection. In checking the unavailable email, when we do 'user_management.check_email_not_in_use' it should be that the email is in use, because we're asserting that the user already has that email and therefore it should be self.assertFalse. This means that we are testing for the available email, it should be self.assertFalse. Essentially, the fetch-one comes out none, we're asserting this will return true, and vice-verse if the fetch-one comes out not 'none', it should return false.

The tests that check for valid and invalid logings have also been run. Assuming someone types in the incorrect password, mock_db_cursor should be returning none. This means that user.fetchone is

going to be 'none', and hopefully, the result we get from our function should also be none. Both tests return the expected output for valid and invalid login.

System limitations

Our current system does not have password encryption which means that anyone accessing our user database, including hackers, can easily access stored passwords. For the future, we would like to implement password encryption/ password hashing, as our system doesn't current implement this. Encryption will allow us to securely store in the database which offers more protection and makes passwords unreadable. We could use simple password hashing to encrypt our passwords.

One of the main limitations our current program faces is that it is restricted to a website, which means we are limited in the automation tools we can use to monitor the user's fitness levels. This could be improved through converting the website into an android or iOS app to be utilised on a smartphone or smart watch. In doing so, we could use tools such as pedometers, heart rate monitors, accelerometers etc. to track activity levels each day, which can be implemented into an algorithm to track the number of calories burned and as a result, tailor the exercises and meals recommended to the user.

Another limitation our system faces is users having to manually input details such as height and weight. This could potentially lead to discrepancies caused by self-measuring. Given that the website is unable to track calories burned or other activity levels, it can become difficult to keep track of progress unless the user updates their details manually when progress is made. This could prevent long term use of the website as users lose patience and might lose motivation as its not automatically recording data and providing relevant reminders.

The website is unable to currently send user reminders regarding meals or exercises – this can mean the most users could forget to log on every day to get this information. One way to overcome this is by sending the user email reminders to let them know to check their progress. If this were to be implemented into an app, sending reminders could be simplified even further through notifications.

A final potential limitation seen in the 'recipes' and 'fitness' page, is that a limited number of options are provided. We had chosen a limited number of meals and exercises so as not to overwhelm the users with too much choice, but some users may want more meal or exercise selections available; however, this could easily be remedied through changing the program to output more options.

CONCLUSION

The aim of our project was to create a fun fitness and healthy eating website. We believe our website meets this aim through our simplistic design format and the use of various API's and user databases, in conjunction with HMTL and CSS for front end development, and Python for backend development. We've met our project requirements of storing, displaying and updating the user's details, providing randomised and user-specified meal options, and providing daily randomised workouts. In the future we hope to develop the website into an app to implement more fitness features that can improve automated data recording.

REFERENCES

1. *Obesity and overweight.* World Health Organisation. (2018). https://www.who.int/en/news-room/fact-sheets/detail/obesity-and-overweight
2. Flores Mateo, G., Granado-Font, E., Ferré-Grau, C. and Montaña-Carreras, X., 2015. *Mobile Phone Apps to Promote Weight Loss and Increase Physical Activity: A Systematic Review and Meta-Analysis.* Journal of Medical Internet Research, 17(11), p.e253.
3. Ross, K. and Wing, R., 2016. I*mpact of newer self-monitoring technology and brief phone-based intervention on weight loss: A randomised pilot study.* Obesity, 24(8), pp.1653-1659.
4. Schiel, R., Kaps, A. and Bieber, G., 2012. *Electronic health technology for the assessment of physical activity and eating habits in children and adolescents with overweight and obesity IDA.* Appetite, 58(2), pp.432-437.