

Exercise Tracker Website

Computer Science A-Level Project

Marcus Ciobanu

Contents

1 Citations	3
2 Analysis	4
2.1 Define the Problem	4
2.2 Why the Problem is suited to a Computational Solution	5
2.2.1 Computational methods that are appropriate	6
2.3 Stakeholders	7
2.4 Research	8
2.4.1 Existing Systems	8
2.4.2 Features of proposed solution	15
2.4.3 Limitations	16
2.4.4 Measurable criteria	16
2.4.5 Solution requirements	17
3 Design	18
3.1 Problem Decomposition	18
3.1.1 Features that need to be included	18
3.2 System Design (Problem Modelling)	19
3.2.1 High Level Overview of System Design	19
3.2.2 System Architecture	20
3.3 Permanent Data Storage	21
3.4 Algorithms	23
3.4.1 Accounts	23
3.4.2 Dashboard	24
3.5 Identification of Key Variables, Data Structures and Classes	24
3.5.1 Accounts	24
3.5.2 Dashboard	25
3.6 Test Plan	25
3.6.1 Test Data	25
3.7 Usability Features	27
4 Development	29
4.1 Iteration 1	29
4.2 Iteration 2	31
4.3 Iteration 3	42
4.3.1 Index, login, master, register	42
4.3.2 Static	43
4.3.3 Templates	43
4.3.4 .gitignore	43
4.3.5 db.sqlite3	43
4.3.6 manage.py	44
4.3.7 Backend functionality	44
4.3.8 Iterations on the frontend	45

1 Citations

Listed in order of their appearance:

- Statista (2022). UK fitness/health club market size 2021 — Statista. [online] Statista. Available at: <https://www.statista.com/statistics/1194831/fitness-health-club-market-size-uk/>.
- MDN Web Docs. (n.d.). Django introduction. [online] Available at: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>.

2 Analysis

2.1 Define the Problem

In recent years, the fitness industry has seen a boom in revenue and participation, with more people than ever being conscious of their health and diet. The figure below shows the market size of the gym, health & fitness clubs industry in the United Kingdom (UK) from 2012 to 2022. As can be seen, there has been a steady increase up until 2020.

Following this point, the COVID-19 pandemic caused a major blow to the gym industry. However, this is not representative of fitness as a whole. What the pandemic actually caused was a huge influx of beginners to fitness who, as a result of their abundance of free time and boredom, took up exercise. Furthermore, the figure also indicates a huge gain in market size from 2021 to 2022 as gyms opened back up. It can be expected that this trend will continue and the fitness industry's market share will soar to new heights as those who began exercising during the pandemic purchase gym memberships.

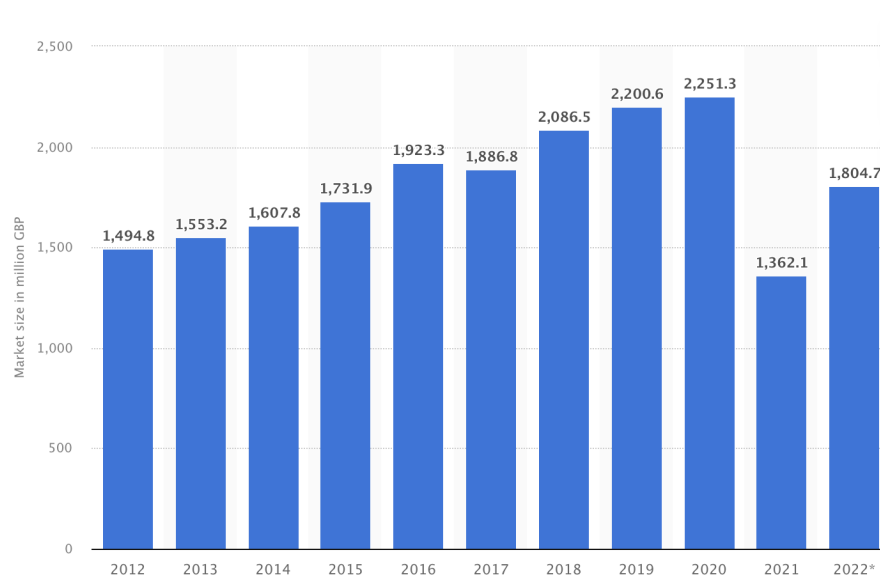


Figure 1: Industry market size

As a result of the growth of the market, there is now room for competition in the digital fitness space (a previously monopolised industry where major players held the majority of the user base). New developments have the opportunity to seize user attention by solving their problems and capitalising on the unique needs of beginners.

From my own previous experience as a beginner in fitness, I can attest to the difficulty and friction associated with trying to record down everything manu-

ally. Having to carry a notebook and pen to every session is cumbersome and the formatting is often inconsistent. For the modern gym-goer, it is important to have a solution to this problem. The reason for tracking progress in the first place is that it is a critical element in optimising results to build muscle and/or lose weight. Rep ranges must be tracked and weight on the bar must be carefully monitored to maximise results for the minimum amount of effort. Evidently, such an endeavour is complex by nature, hence the need to make this as smooth and easy as possible for the beginner (lest they become demotivated and give up altogether). After all, if the experience is frictionless then you are more likely to record down all your workouts, leading to higher consistency and better results due to the compounding nature of incremental progress.

In summary, there are more beginners than ever in fitness. Beginners suffer from a lack of knowledge regarding what to do. A simple solution that is easy and frictionless is required to help them track their activities and maximise progress. Therefore, my programming project involves developing an exercise tracking website that makes this process seamless and intuitive.

2.2 Why the Problem is suited to a Computational Solution

Much of the suitability for this problem being suited to a computational solution stems from why this is a problem in the first place. Simply put, the analogue equivalent is massively inferior to any variation of a digital solution and makes it difficult for people to stick to it which harms progress. There are multiple reasons for this:

1. As you advance, you will reach the point of having done hundreds of workouts. This results in thousands of exercises, tens of thousands of sets and hundreds of thousands of reps. Therefore, this large volume of data is best suited to being kept digitally, as it would result in many pages if done in analogue form.
2. Pages are liable to physical damage such as being lost or set on fire, this is a risk as it could mean that you are unable to see what you have done previously or track your progress over an extended period of time. In contrast, data can be backed up on the cloud when done digitally, meaning it is kept safe.
3. Handwriting can often be untidy and unclear and takes longer than typing or entering values into a database interface digitally.
4. Pages cannot be easily sorted through to gather meaningful insights (producing graphs for example) and could be sped up and made far more efficient through use of algorithms and computing power.
5. You do not need to bring a pen and paper to the gym with you if it is done digitally, as it can all be done on your phone which most people bring everywhere with them regardless.

2.2.1 Computational methods that are appropriate

As well as the reasons listed above that focus more on the comparison between the digital and analogue ways of solving the problem, there are also computational methods and constructs that lend themselves to the efficient and streamlined solving of the problem.

2.2.1.1 Thinking abstractly

For in depth analysis and generation of insights from the data the user records such as producing graphs that display progress in a certain lift over time, abstraction can be used. The mechanisms behind the production of the graph are abstracted from the user for maximum ease of use and pure functionality.

They would simply record their results into a GUI and select the option to produce a graph, much like how a programmer can input parameters into a function imported from a library. They know what it does and what is required to make it produce the results but do not and do not need to understand how it works. If we were to contrast this to the analogue equivalent, the user would need to understand how to plot graphs and translate data into the necessary formats. In this case, the use of abstraction not only lends itself to the computational solution but enhances it.

2.2.1.2 Thinking ahead

Through the use of a computational solution, we can identify recurring scenarios and therefore develop functions that can be reused every time the user wants to carry out that task. For example, we know that if the user is using an exercise tracker they are going to want to record many workouts. Therefore, we can produce functions that take the expected inputs for these tasks (sets, reps, etc.) and produce outputs in a consistent, comparable format.

2.2.1.3 Thinking procedurally



Figure 2: Top-down design

When a user performs a workout, we understand that there will be a set ‘formula’ which they follow each time in a procedural manner. They would typically select an exercise, create a set then perform the reps that make up that set. Therefore, we can model this within the code through the use of a top-down design.

As can be seen in the figure, we can break down the idea of a workout into nested modules of code in an order that the user would typically think of it in. These modules can then communicate with each other along the chain to produce the desired output.

This method of top-design and decomposition is not only natural to the way of thinking of the user, but is also computationally efficient and speeds up the development process. It will also make the program far easier to maintain and test due to its modular nature.

The modular nature also relates to thinking ahead, as it promotes code reusability and applying the same code to multiple iterations of the same scenario, further decreasing development time needed whilst ensuring consistent formatting and familiarity to the user.

2.3 Stakeholders

Naturally, the primary group of stakeholders will be hobbyist gym goers, mostly due to the fact that they make up the majority of consumers in the market. Therefore, the website will be tailored to suit the needs of beginners to intermediates to maximise the usefulness to the average user. For example, all the exercises in the exercise database will include explanations on how to perform the exercise and what muscle groups it trains. Another example is the goal setting feature that will be included in the signup process which generates a recommended goal for them based on the information they enter during signup. Regardless, there will still be many features applicable to more advanced users as well, as the crossover is substantial and the core features of the website have utility throughout the spectrum of users.

The software is not intended for use by people under the age of 16, as this is the typical age requirement for most gyms and so it would be a poor use of resources and development time to make considerations for this user group. Despite this, the program is still usable by this group and there is no hard restriction in place to prevent them from using it. Simply put, it will not be designed in a way that places any priority on their specific needs and use cases, but they may be able to still use it for home workouts involving callisthenics and other bodyweight movements that do not require a gym to perform.

In terms of people with disabilities, both mental and physical, the program retains its utility. Furthermore, no special features need be designed to cater for them. For anyone with physical disabilities, they can simply select the exercises that they can perform safely and to a good standard with their own

disability in mind. Many exercises can be performed without the use of legs, for example dumbbell hammer curls. Therefore, in this case, the user will make their own adjustments so that the program can suit their needs. For anyone with mental disabilities, this should not have much impact on their ability to perform exercises, so the program should work as intended for them. However, the program will not cater for anyone with impaired vision, due to the difficulty of development of a comprehensive audio navigation system. This is a limitation, and restricts the user base for the project.

Here is an example stakeholder to illustrate a typical use case:

Name	Level of ability	Any disability ?	How they will use the project	Why the solution fits their needs	What features would you like to see
Jake Wilson	Professional Athlete	No	"I will definitely be using the workout tracker to stay on top of my regime and uniformly target muscle activation."	"It's a very easy to use and intuitive technology to allow me to achieve my goals"	"The app should suggest workout routines to people, if people are new to it and have no clue what they are doing. The app could also be able to create workout routines on its own"

Figure 3: Example stakeholder

Utilising feedback and requests from potential stakeholders such as Jake will be vital in ensuring that the finished project meets the needs and requirements of those who will be most likely to use it.

This iterative process of feedback and adjustment is important to increase customer satisfaction and widen the user base however it is also important to place limitations on what is realistic and achievable in the timeframe as what is just as important to stakeholders as a feature rich program is that the program is delivered on time and within budget. The balance between these two elements is something that will remain in mind throughout the whole project.

2.4 Research

2.4.1 Existing Systems

There are two major existing systems that offer a workout tracking system similar to the proposed solution. These are Hevy and Strong. Alongside the research that follows, I have been a user of both programs at separate times for over a year so I believe this first-hand experience with both gives me the necessary insight to properly evaluate their features and identify the correct way to take on board this information for my own project.

2.4.1.1 Hevy

Hevy is the premier workout tracking app on the Android Play Store. The platform is different to the proposed solution (mobile vs. browser) but the core features remain the same and so it is valid to analyse. It incorporates workout tracking with a social media system where you can follow others and share your workouts and progress with a following. It works sort of like Instagram in that regard, being able to like and comment on others workouts. However, this element of the program is completely optional as you can make your account private, so for the purpose of this research, I will not be looking at this element of the program. It is also the cheaper of the 2 existing solutions priced at £2.99/month compared to Strong which is £4.69/month.

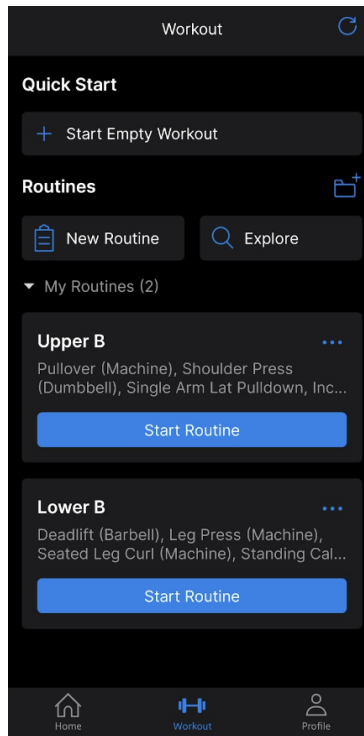


Figure 4: Quick start page

The general consensus among users is that Hevy offers more features for analysis and tracking for a cheaper price whilst Strong offers a slick, clean UI with the actual workout tracking part of the app being very intuitive and easy to use. However, this is all subjective and in terms of user ratings, they both sit at a high 4 star rating with minimal criticism of either application.

In the surrounding pages are a few screenshots from my mobile phone from the app which illustrate a few key features that I will go through and evaluate.

They cover a range of the app's features from its workout tracking capabilities to its extensive section for analysis of progress and workouts.

See Figure 4, the main section of the application which allows you to record your workouts. You can either create your own routines, choose from a range of templates in the explore section or select the quick start option. This will allow you to go straight into a workout and add your exercises on the fly.

There is also an option to create routine folders. This allows increased organisation and to periodise your training into blocks sorted by folder (a useful feature for more advanced lifters who may want to alternate between load and deload periods). This could definitely be a feature to adopt in order to ensure that the program has adequate tailoring for all ability ranges.

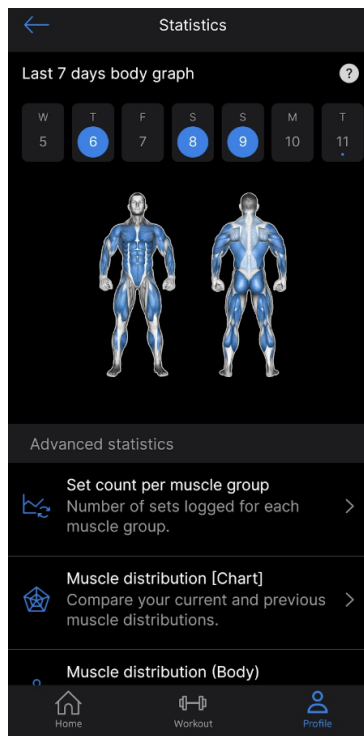


Figure 5: Statistics page

Overall, Hevy does a great job of offering options for all use cases and ability levels, with customizability available if needed but not required. Furthermore, the UI is slick and intuitive, which is necessary for a small mobile screen.

The statistics section of the application is rather unique to Hevy, where you can analyse your progress in as much detail as you want to aid in organisation and planning. In particular, the heatmap is very useful, allowing you to see at a glance which muscle groups you have hit during that week. You can therefore quickly identify lacking muscles and pick exercises for your next workout that

target them specifically.

The advanced statistics section allows for a more indepth look at what the heatmap gives you at a glance. It logs sets per muscle group and shows a spider web chart of the distribution. All of these features can be leveraged for increased control and data on your progress. However, all of this is optional and doesn't need to be used which is a good feature of Hevy. It is as in depth and detailed as you want it to be.

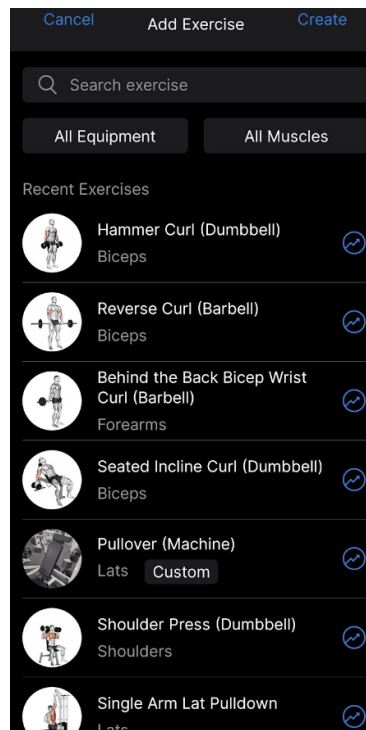


Figure 6: Add exercise page

Figure 6 shows the exercise addition menu once you either start a routine or start empty exercise.

There is a large bank of premade exercises that cover nearly everything you will need. They also include an image and sometimes a video with instructions explaining how to perform every exercise. This is undoubtedly useful for beginners but more advanced lifters may want to do further research to optimise form and technique.

All exercises when selected are automatically logged into a database, as well as the sets and reps you will go on to perform.

As you can see further down the list, there is an exercise called 'Pullover (Machine)'. This is a custom exercise. You can create your own exercises if your gym has some niche machinery or you wish to perform uncommon variations of

certain exercises that cannot be found in the premade exercises.

See Figure 7, this is the exercise creation menu. It can be accessed from the exercise addition menu by clicking on the button in the top right.

You can create any exercise possible due to the features of the creation menu. A custom name and picture can be given. More importantly, the muscles it targets (both primary and secondary) are included, as well as the exercise type and equipment. This allows it to be included in the statistics section (heatmap, spiderweb) and also allows you to adjust weight and reps for progressive overload.

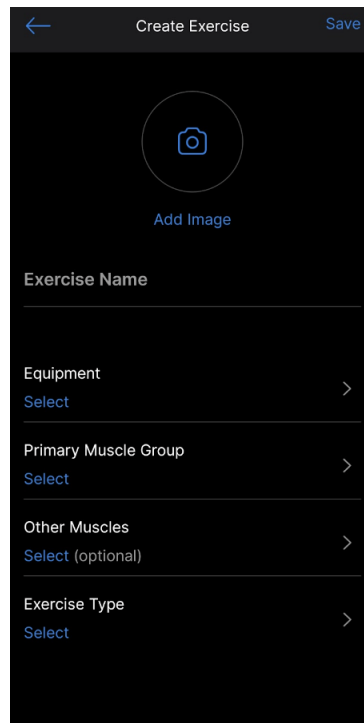


Figure 7: Create exercise page

This is absolutely an essential feature to add in my application as due to development time constraints, it will not be possible for me to create a comprehensive database of premade exercises.

2.4.1.2 Strong

There is a lot of carryover with the features from each app as they are very similar in a lot of ways. One difference I noticed was the far inferior statistics and analysis features that Strong offers in comparison to Hevy. There are only a few options on the profile tab to add charts that track workouts and exercise weight progressions over time.

This information is rather surface level and doesn't really provide much useful insight that would actually help you. Therefore, in my app, I would improve on this and make it more like Hevy.

The profile system is also a lot less developed than Hevy. This makes sense, since Hevy also doubles as a fitness social media app in many ways, so Strong would have little reason to invest in such features. In my app, I would probably take a similar approach to Strong, as it will operate as a standalone package and users will not be able to interact with each other.

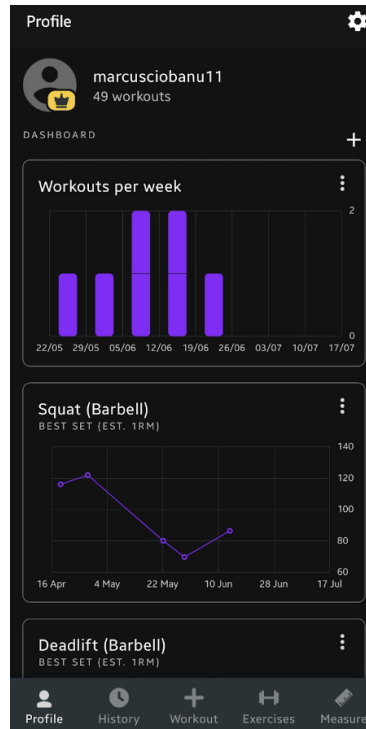


Figure 8: Home page

The workout menu also functions in a very similar way to Hevy. However,

the template system is far less developed, which is definitely something I would like to improve in my app. I found that the folder and stock template features were very useful so I would like to develop my app in that way.

Despite this, all the other features are practically identical to Hevy. They function as intended and have utility. The base template of a workout tracker seems to be consistent no matter the developers which is a testament to its effectiveness. A quick start and template system will be integral to the workout tracking system in my application.

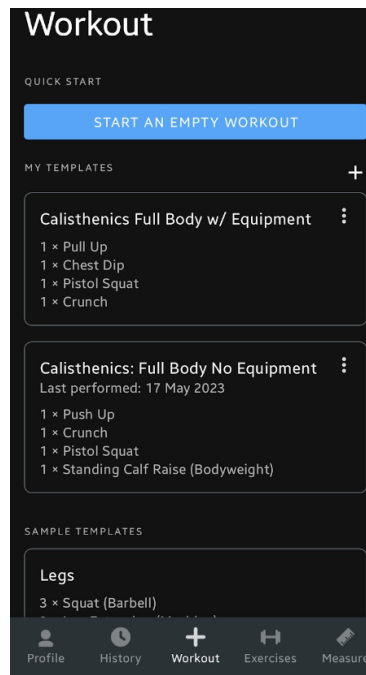


Figure 9: Workout menu

The history menu is something that Hevy also has, but the layout in Strong is a lot more minimalist and focuses on showing the most important information. It is also categorised by month so it allows you to block and periodise your training which can certainly have utility for some people.

Overall, where Strong excels is its functional, intuitive and easy to use GUI. Especially for the mobile format, having little to no clutter is something that consumers would definitely appreciate and Strong has done that in a textbook manner.

It is clear that despite the lack of updates and new features in Strong as compared to Hevy (the Strong developers have given up on the app in many ways), there is still a consistent user base and the reviews remain high. This is a testament to the strong foundation of the app within its core features.

2.4.1.3 Summary of features to add

- Quick Start Workout Feature
 - This will take the form of an interactive UI element that can be pressed
 - This is necessary to further tailor my program to the beginner demographic who may not have a rigid routine and would appreciate the ability to start a workout and choose their exercises as they go along
- Folder Organisation System
 - This will be a niche feature with high utility to ensure that the program also retains good use for more advanced users who may have periodisation blocking and alternate their training regimes.
- Exercise Creation Function
 - Essential feature to reduce development and make it so that users can record their workouts no matter the equipment they use or the gym they go to
 - This will be implemented as its own page that is linked through the workout tracking section.
- Statistics Section
 - Great feature for both beginners and advanced users alike as it is as detailed and informative as you want it to be and gives useful information to see if you are on the right track
 - This will have its own dedicated section on the website with multiple pages

2.4.2 Features of proposed solution

Key features:

- A workout tracking section with a quick start option and a template creation and selection module with a folder organisation section for training blocking and periodisation
- A statistics/analysis section where users can see their muscle group distribution by sets, reps or weight. There should also be graphs to track progressive overload over time.
- A workout history section with a scroll down menu to view all past workouts and the key information from each workout

The listed features above are the most important because they provide the maximum amount of utility to the user. There are many more features I will likely add, but these remain a priority as they are essential to ensuring the program meets the needs of the stakeholders. That being, it is easy to use and intuitive whilst providing useful insight to maximise progress with minimal effort.

2.4.3 Limitations

1. There will be no user to user connection like Hevy has. In other words, there will be no follow, posting or liking systems and you cannot see the progress of friends etc. This is due to development time constraints.
2. I will not be able to port this website to mobile formats, so when a mobile browser loads the web app, they will be faced with the desktop layout. This may prove cumbersome for users, but since I am developing on desktop it would make development far easier. Therefore, I will not provide a mobile reformatting again due to time constraints.
3. Testing will be limited to desktop. This means that there may be undiscovered bugs that arise when the website is opened on mobile. This, combined with the poor mobile formatting, will make the user experience for this demographic poor, but it is necessary to ensure that the solution is completed on time with the necessary features.

2.4.4 Measurable criteria

Objective	Proof
Website has comprehensive login system	Screenshot of login page with all the fields required to be filled
Website has quick start workout function	Screenshot of quick start button and screenshot of workout started after the button is pressed
Website has workout template creation system	Screenshot of template created
Website has statistics section	Screenshot of webpage and a sample graph or metric
Website has workout history section	Screenshot of history webpage
Website has folder organisation	Screenshot of folder navigation
Website has user instruction and guidance	Screenshot of FAQ page or similar

Figure 10: Measurable criteria

2.4.5 Solution requirements

- Users must create or login to an account upon opening website
 - Account creation details:
 - * Name (full)
 - * Username
 - * Email
 - * Password
- Users should be able to track and record workouts
 - Start a workout by template or quick start
 - Add exercises (either from the premade database or custom)
 - Add sets
 - Add reps
 - Specify type of set (drop set, failure set, warm up set, working set)
 - Be able to view all past workouts

These are the core features that will make the web app work, but there are many quality of life features that I will also plan to add. This outlines the bare minimum for it to function as intended with some essential features. Everything else only improves on the user experience in essence.

In terms of hardware requirements, the device the user is currently on must be able to run a web browser and connect to the internet. They must also have around 1GB of local storage available to leave plenty room for database downloads and caching websites locally.

3 Design

3.1 Problem Decomposition

As is to be expected with a program of this complexity, there are many moving parts, which would be best tackled by breaking them down into their constituent parts.

3.1.1 Features that need to be included

Following on from the analysis, we need to decide which features will be implemented as a base to provide the essential functionality of the program. Essentially, we are abstracting away any of the extra things which do not form the core of the program.

3.1.1.1 Landing Page

This should be as minimal and bare-bones as possible. The only purpose it serves is letting the user know that they are at the website when they enter the root URL into their browser (now decided to be named europaFitness), and to give them a starting point to enter the actual functionality of the website.

3.1.1.2 Registration System

This should allow the user to create an account. This will include a username (unique, as it is part of the composite key), email, forename, surname and password. There are a few key requirements for this. Each of the inputs must be validated to ensure they are in the correct format and are unique when needed. Additionally, the password must be hashed before being sent to the database to ensure security.

3.1.1.3 Login System

This naturally follows from the registration system. The login will be made up of the username and password, as that is all that is necessary to uniquely identify a user and make sure it is them who is logging in.

3.1.1.4 Dashboard

This is where the user will be immediately taken to after logging in or registering an account. It will be made up of a sidebar, with the various applications listed on it. At the top of the sidebar will be a website logo, which the user can press to return to the landing page without ending their session. At the bottom of the sidebar will be a traditional logout button. This sidebar will be on the left of the screen, taking up about 15-20% of the screen width. This sidebar can also be collapsed even further if the user wishes to focus on the application.

The rest of the screen will be dedicated to the specific application the user is on.

3.1.1.5 Home

The default app selected when entering the dashboard. Simply welcomes the user to the app.

3.1.1.6 Workout

This is the core of the web app. It allows a user to quick start a workout on the fly and add in whatever exercises they want from the exercise bank. Then they can add sets to each exercise, specifying weight, number of reps and type of set. The workout will also record how long it took and the date. Once the user is finished, they can submit the entire workout and save it to the database.

3.1.1.7 History

Simply allows the user to view a history of workouts done, retrieved from the database, displaying all of the information associated with that workout. It will also allow the user to delete the workout.

3.1.1.8 Templates

Allows the user to create templates of workouts consisting of the exercises they want to do. Saves the user time if they follow a regime which involves repeated workouts. On the workout app, the user will then be able to choose whether they want to quick-start a workout or select from their created templates.

3.1.1.9 Create Exercise

Allows the user to create their own exercises and add them to the database and also view all of their added exercises and delete them if they want.

3.1.1.10 Error pages

A set of pages which restrict user access through URL inspection. For example, if a user tries to access the dashboard when they are not logged in, an error screen will be returned informing the user of the error and redirecting them to the login page.

3.2 System Design (Problem Modelling)

3.2.1 High Level Overview of System Design

Now that the features that are needed to be implemented are clearly defined, we need to decide how these features are going to be implemented in a broad overview.

3.2.1.1 Technologies

The proposed solution will be a website. Therefore, we need to consider the technologies used for both the client side and server side. I have decided on using HTML, CSS and JS to build the front-end (standard in web-development), and Python for the back-end.

3.2.1.2 Frameworks

Various frameworks could be used to help abstract some of the technicalities of many common processes in web-development, both in the front-end and back-end. For the front-end, I will keep it simple and use vanilla HTML, CSS and JavaScript as this is more than sufficient for my needs and gives me extensive control over the design and styling of the website.

For the back-end, it is essential to use a framework to abstract the tedious tasks of routing HTTP requests and the like. For this I will use Django. Django is a batteries-included, opinionated framework enabling the creation of websites using Python. Since it is batteries-included, it comes with all the packages and dependencies you would likely ever need, all with extensive and detailed documentation. This will make development hassle-free.

3.2.2 System Architecture

Handling the client-server dynamic will be a key consideration when developing the solution. There must be a clear structure to separate the various functionalities of the code. I will be using the MVT (Model, View, Template) architecture to structure my code. See figure 11 for an abstraction of how the MVT architecture functions.

By nature, the MVT architecture (and by extension, Django itself as a framework) is modular and delegates all tasks to separate modules. As a HTTP request is received by the server, it uses the URL to decide which 'view' function to delegate the task of handling it to. It does this through a 'urls' file, which acts as a traffic warden, directing the flow of the program. The 'urls.py' file in a Django project would map the values of the URL to an appropriate view function. It can also read up to a certain point of a URL and then cut off the part it has already read, handing the rest to a URL file in a certain app that it would be related to, enhancing modularity and maintainability.

Once it decides which view function to hand it to, the HTTP request is sent to the view. Views control the business logic of a given app. It dictates what you will see when the HTTP response is received, but not necessarily how you will see it. That is, it has no influence over the styling or aesthetics of the rendered page, but does have control over the contents. The view will read the HTTP request and decide what to do next. It may need to gather data from the models.

Models are an abstraction over traditional databases. In Django, OOP is used to define a model, where each model corresponds to a table in a relational database. The attributes represent fields and the methods define things like

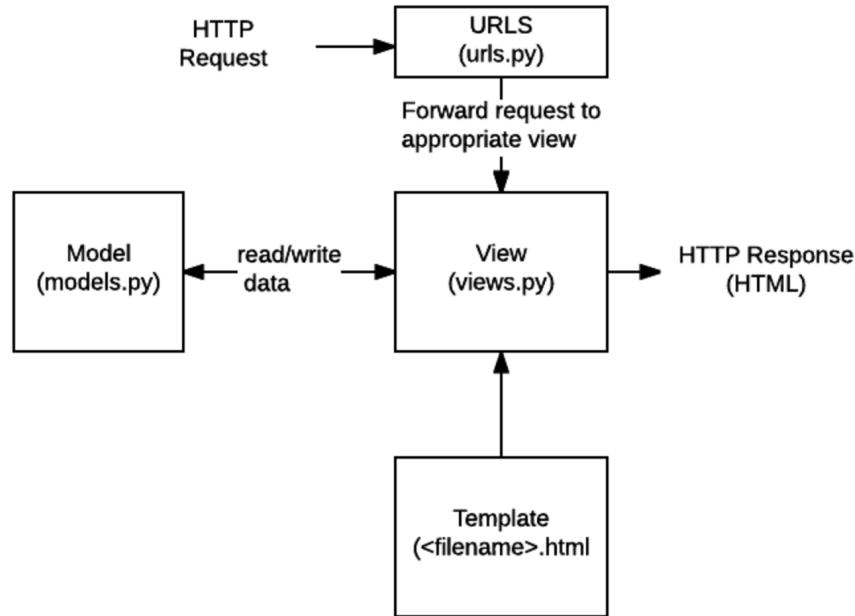


Figure 11: MVT architecture

initialization of an object, deletion, foreign key relationships and more. This layer of abstraction also provides an extra layer of security, as at no point in the program can a user's inputs ever be mapped directly into an SQL statement. This prevents SQL injection.

Back to the view function, once the data has been gathered from the models and all associated logic has been calculated, it then delegates the formatting of the HTTP response to the templates. Templates are an extension of traditional HTML. It would be grossly inefficient to manually format each possible HTML file that could be created from a user's inputs. So, we use templates to introduce placeholders (variables) directly into the HTML markup which will then be read and compiled by the template interpreter built into the framework to render the final HTML to be sent in the HTTP response. A important thing to note, the template does not have to be HTML, but for our purposes, it will nearly always be HTML.

3.3 Permanent Data Storage

Underneath the model abstraction, there will still need to be a traditional database for permanent data storage. For this project, I will be using SQLite. It is a lightweight, minimalist SQL database that is suitable for projects of small size. It works best for sites with low concurrent usage which is appropriate for the use case.

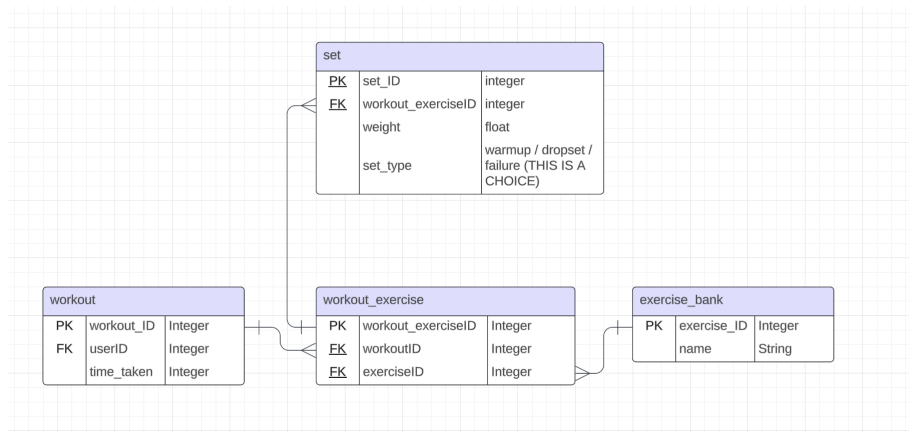


Figure 12: ERD of database

There are a couple of tables that need to be designed. In terms of users, there is a built in module: 'django.contrib.auth' which provides functionality for user creation out of the box. Since there is no use in re-inventing the wheel, I will use this module as it provides hashing of passwords and handling of user registration and logging in forms. As for the workout app, I will design my own relational database. See the ERD in figure 12 for specification of the various fields and relationships of the proposed database.

It follows a top down architecture where the workout acts as the parent of the workout_exercise, which in turn acts as the parent of the set. This is defined through the foreign key relationships. A workout belongs to a user. A workout_exercise belongs to a workout (the exercise being taken from the exercise_bank which defines all the created exercises by the user). A set belongs to a workout_exercise. Many to many relationships have been removed, and the database is normalized.

3.4 Algorithms

The proposed solution, when broken down, broadly separates into two main categories: Accounts and Dashboard. These represent Django apps that separate the functionality of the website into two main modules. Accounts handles the face of the website. That is, the landing page and the account creation process/login process. Dashboard is a broad term used to represent all of the different apps identified in the Problem Decomposition: Workout, Exercise Bank, History etc. Below are pseudocode functions for the most important parts of the program.

3.4.1 Accounts

```
1 FUNCTION loginpage(request):
2   IF the request is a POST:
3     Make a form with the POST data.
4     IF the form is okay:
5       Get username and password from form.
6       Try to log the user in with username and password.
7       IF the user is found:
8         Log the user in.
9         Go to the dashboard home page.
10    ENDIF
11  ELSE:
12    Just show the empty login form.
13  ENDIF
14  Show the login page with the form.
15 ENDFUNCTION
```

Listing 1: Login Page

```
1 FUNCTION registerpage(request)
2   IF form submitted THEN
3     Create form with submitted data
4     IF form data is correct THEN
5       Save new user
6       Log in new user
7       Go to dashboard
8     ENDIF
9   ELSE
10    Show empty form
11  ENDIF
12  Show registration page with form
13 ENDFUNCTION
```

Listing 2: Register Page

```
1 CLASS CustomUserCreationForm INHERITS UserCreationForm
2   SETUP form to use User model and add fields: first name,
   last name, email
```

```
3 ENDCLASS
```

Listing 3: CustomUserCreationForm

3.4.2 Dashboard

```
1 FUNCTION create(request)
2 IF user logged in THEN
3   Get user exercises
4   IF form submitted THEN
5     Create and check form with user data
6     IF form correct THEN
7       Save exercise
8       Reset form after save or if incorrect
9   ELSE
10    Show new form
11    Show page with form and exercises
12  ELSE
13    Show login error page
14  ENDIF
15 ENDFUNCTION
```

Listing 4: Exercise Creation view

```
1 FUNCTION delete_exercise(request, exercise_id)
2 IF user logged in AND form submitted THEN
3   Find exercise by id for the logged-in user
4   Delete exercise
5   Redirect to create page regardless
6 ENDFUNCTION
```

Listing 5: Exercise deletion view

These algorithms do not form the entire solution, as I haven't planned in detail every single function that will make up every app. Arguably, the most complex function will be the workout app main function, as it will involve other modules and classes from throughout the whole function. Therefore, there is an element to iteration to this design, and I will come back to it in the future once I reach the parts that require more design.

3.5 Identification of Key Variables, Data Structures and Classes

3.5.1 Accounts

Each account is represented by a model class derived from the framework module 'django.contrib.auth'. Each instance of a user class has an ID (integer), username (string), forename, surname and email (all strings).

Since the default user creation form in Django only includes the username and password, I will need to create a custom user creation form class that inherits from the superclass user creation form. It will then add in the extra fields so that the user can enter their names and email in the form.

For the registration form, I can use the default registration form as it works perfectly fine for my purposes. Validation is handled by the `.is_valid()` method that every form inherits from `'django.contrib.auth.models'` via the `ModelForm` superclass.

3.5.2 Dashboard

The main data structure to consider for this is the relational database. Models will need to be created for each table of the database. It will inherit from the superclass `'models.Model'`, which will handle many of the methods involved with SQL manipulation. For the `'set'` model, a 2D array will need to be used to create the inputs for set type, as they can take multiple pre-defined values.

Various views may need to take in multiple inputs from the HTTP request. Therefore, a dictionary would be the most appropriate data structure to store these. The key value pairs will consist of the variable name and the variable value. Values can then be popped off the dictionary when they need to be called.

3.6 Test Plan

During the iterative development, I will test the functions as I create them to ensure they function correctly, documenting the testing as I go along. Then, post development, I will test the program as a whole to ensure it all works together cohesively. The test data here may not be complete as I haven't planned out all of the functions I will include in full detail yet, so as I develop iteratively I will come back to this and include more functions.

3.6.1 Test Data

3.6.1.1 Registration Function

- Valid Data:
 - Username: User123
 - Forename: John
 - Surname: Doe
 - Email: john.doe@example.com
 - Password: Password123!
 - Confirm Password: Password123!
- Boundary Data (testing the limits of valid data, such as minimum/maximum lengths):
 - Username: Us

- Forename: J
- Surname: D
- Email: j.d@example.co
- Password: Passw0rd!
- Confirm Password: Passw0rd!
- Erroneous Data (data that should be rejected by the form):
 - Username: User@123
 - Forename: John123
 - Surname: Doe!
 - Email: john.doe.com
 - Password: pass
 - Confirm Password: pass
- Erroneous Data with SQL Injection Attempt (to test for security vulnerabilities):
 - Username: 'User123'; DROP TABLE users; –
 - Forename: John
 - Surname: Doe
 - Email: john.doe@example.com
 - Password: Password123!
 - Confirm Password: Password123!

3.6.1.2 Login Function

- Valid Data:
 - Username: validUser
 - Password: CorrectP@ssword1
- Boundary Data:
 - Username: u (minimum length allowed)
 - Password: P@ssw0rd (minimum complexity)
- Erroneous Data:
 - Username: invalidUser
 - Password: WrongPass
- Complex Erroneous Data (SQL Injection Test):
 - Username: admin' –

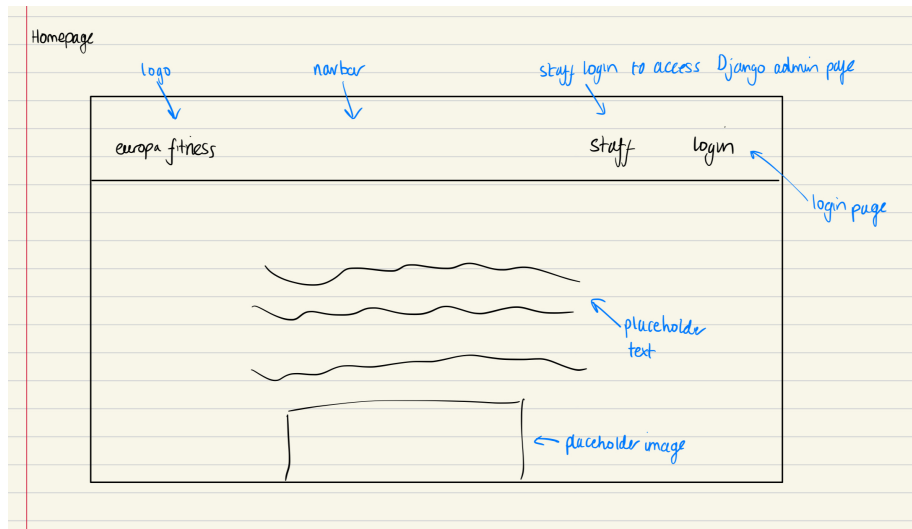


Figure 13: Mockup of the landing page

– Password: anything

In each case, the valid data should work and be accepted, creating an account or logging in the user. The boundary data should do the same. The SQL injection attempts should be ineffective due to the abstraction layer provided by the object relational mapper that prevents user inputs from being mapped directly to SQL statements. The erroneous data should be rejected and errors should be returned specifying what needs to be corrected.

3.7 Usability Features

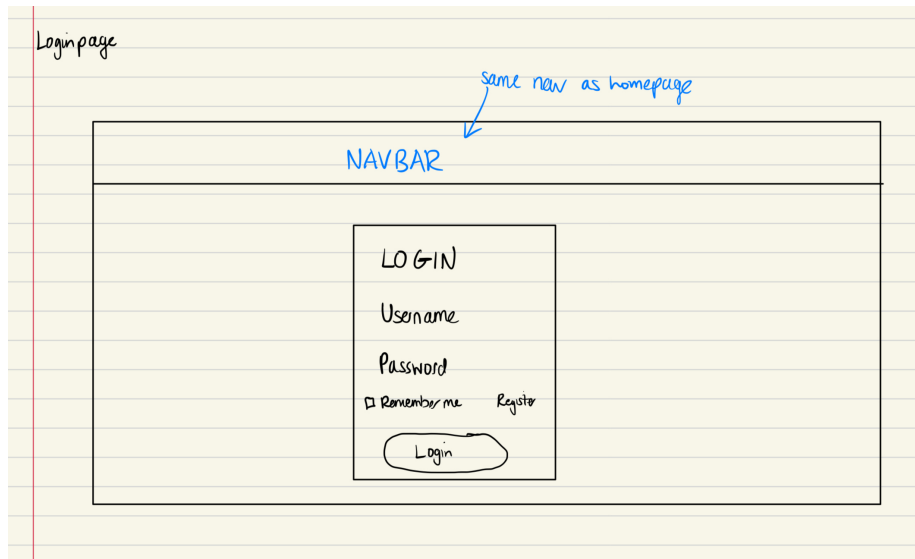


Figure 14: Mockup of the login page

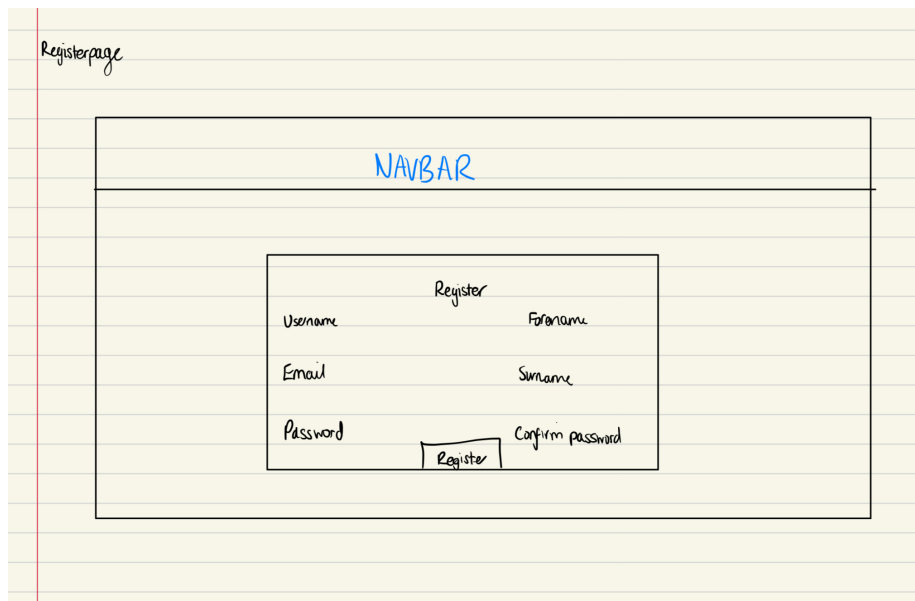


Figure 15: Mockup of the register page

4 Development

4.1 Iteration 1

For the first iteration, I created a basic markup of the landing page with a couple of styles. It was very rudimentary and only served to create an idea of what the website could begin to look like. There was no logic to the page and it was purely made up of static files with no backend code.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width,
6          initial-scale=1.0">
7      <title>trackR</title>
8      <link rel="stylesheet" href="/css/style.css">
9  </head>
10 <body>
11     <header class="name">
12         <a href="landingpage.html">
13             trackR
14         </a>
15     </header>
16
17     <nav class="menubar">
18         <div>
19             <ul>
20                 <li>
21                     <a href="#">
22                         Login
23                     </a>
24                 </li>
25                 <li>
26                     <a href="#">
27                         Create Account
28                     </a>
29                 </li>
30                 <li>
31                     <a href="#">
32                         
34                     </a>
35                 </li>
36             </ul>
37         </div>
38     </nav>
39
40     <div class="placeholder">
```

```

39     <p>Lorem ipsum dolor sit amet consectetur adipisicing
        elit. Obcaecati, voluptatibus laborum iure quia
        neque sint alias repellat eos quod quidem
        molestiae? Ullam, quibusdam ipsam consequuntur
        omnis doloribus ea quis beatae
40     </p>
41 </div>
42
43 </body>
44 </html>

```

Listing 6: index.html

The page consisted of a simple nav bar and placeholder for the main portion of the page. It was more or less a less abstracted version of the landing page detailed in the usability features sections, translated into HTML and CSS.

```

1  body {
2    font-family: Helvetica;
3    font-weight: 100;
4    background-color: white;
5    margin: 0;
6  }
7
8  .name {
9    background-color: #19747E;
10   padding-left: 10px;
11   font-size: 40px;
12   font-weight: bolder;
13   color: black;
14   text-decoration: none;
15 }
16
17 .name a{
18   font-size: 40px;
19   font-weight: bolder;
20   color: black;
21   text-decoration: none;
22 }
23
24 .menubar {
25   background-color: #A9D6E5;
26   overflow: hidden;
27   text-align: right;
28 }
29
30 .menubar ul{
31   height: 50%;
32   margin: auto;
33   list-style: none;
34 }

```

```

35
36 .menubar li{
37     display: inline;
38     padding: 25px;
39 }
40
41 .menubar a{
42     color: black;
43     font-weight: bold;
44     text-decoration: none;
45 }
46
47 .menubar li:hover{
48     background-color: #19747E;
49 }
50
51 .placeholder {
52     padding: 5px 15px;
53     line-height: 1.4em;
54 }

```

Listing 7: style.css

This rudimentary CSS just made the page bearable to look at. The design was inefficient and inelegant but for a first attempt it would suffice.

4.2 Iteration 2

For this iteration, focus was concentrated on the front-end, working to build up a clean and interactive UI from which I could build the program logic around when it came to the backend. It is also important to note that at this point, I had little to no knowledge of backend development, and was essentially learning as I go along, building up from the basics.

This iteration specifically involved a complete overhaul of the landing page design and a first iteration at the login form and register form. Still, there was no actual functionality and it was purely about the GUI.

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width,
        initial-scale=1.0">
6      <title>Homepage</title>
7      <link rel="stylesheet"
        href="/landingPage/css/landingPage.css">
8  </head>
9
10 <body>
11

```

```

12 <!-- Content for menubar -->
13
14 <header class="menubar">
15   <h2 class="logo">europa fitness</h2>
16   <nav class="navigation">
17     <a href="landingpage.html">Home</a>
18     <button class="loginBtn">Login</button>
19   </nav>
20 </header>
21
22 <!-- Content for login form -->
23
24 <div class="wrapper">
25   <span class="closeButton">
26     <ion-icon name="close-sharp"></ion-icon>
27   </span>
28   <div class="loginForm">
29     <h2>Login</h2>
30     <form action="#">
31       <div class="inputBox">
32         <span class="icon"><ion-icon
33           name="mail-outline"></ion-icon></span>
34         <input type="email" required>
35         <label>Email</label>
36       </div>
37       <div class="inputBox">
38         <span class="icon"><ion-icon
39           name="key-outline"></ion-icon></span>
40         <input type="password" required>
41         <label>Password</label>
42       </div>
43       <div class="rememberForgot">
44         <label><input type="checkbox"> Remember
45         me!</label>
46         <a href="#">Forgot Password?</a>
47       </div>
48       <button type="submit" class="btn">Login</button>
49       <div class="loginRegister">
50         Don't have an account? <a href="#"
51         class="registerLink">Register!</a>
52       </div>
53     </form>
54   </div>
55
56 <!-- Content for register form -->
57
58 <div class="registerForm">
59   <h2>Register</h2>
60   <form action="#">
61     <div class="inputBox">

```



```

58         <span class="icon"><ion-icon
           name="person-outline"></ion-icon></span>
59         <input type="text" required>
60         <label>Username</label>
61     </div>
62     <div class="inputBox">
63         <span class="icon"><ion-icon
           name="mail-outline"></ion-icon></span>
64         <input type="email" required>
65         <label>Email</label>
66     </div>
67     <div class="inputBox">
68         <span class="icon"><ion-icon
           name="key-outline"></ion-icon></span>
69         <input type="password" required>
70         <label>Password</label>
71     </div>
72     <div class="rememberForgot">
73         <label><input type="checkbox"> Remember
           me!</label>
74     </div>
75     <button type="submit" class="btn">Create
       Account</button>
76     <div class="loginRegister">
77         <p>Have an account?</p><a href="#"
           class="loginLink">Login!</a>
78     </div>
79 </form>
80 </div>
81 </div>
82
83 <!-- Main bulk content for the landing page -->
84
85 <div class="landingPagePlaceholder">
86     
87     <h1 id="text1">exercise.</h1>
88     <h1 id="text2">diet.</h1>
89     <h1 id="text3">all for</h1>
90     <h1 id="text4">FREE.</h1>
91     
92 </div>
93
94 <!-- Main script for the landing page -->
95
96 <script src="/landingPage/js/landingPage.js"></script>
97
98 <!-- Script to access icons used in login and register
   forms -->

```

```

99
100 <script type="module" src="https://unpkg.com/ionicons@7.1.0
101 /dist/ionicons/ionicons.esm.js"></script>
102 <script nomodule src="https://unpkg.com/ionicons@7.1.0
103 /dist/ionicons/ionicons.js"></script>
104
105 </body>
106
107 </html>

```

Listing 8: landingpage.html

The landing page is now essentially completed. It has a fully fledged and stylized navbar, with a login page that pops up when you press the login button. You can also select the register page from the login page. I also used the Ionicons API to utilize their free open source icons within the markup directly.

```

1  /* Import the font used for the website. */
2
3  @import url('https://fonts.googleapis.com
4    /css2?family=Poppins:wght@500&display=swap');
5
6  /* Defining the base attributes for any element on the
7    page. */
8
9  * {
10    margin: 0;
11    padding: 0;
12    box-sizing: border-box;
13    font-family: 'Poppins', sans-serif;
14  }
15  body{
16    display: flex;
17    justify-content: center;
18    align-items: center;
19    background: black;
20    color: white;
21  }
22
23  /* Animations and attributes for the top navigation bar */
24
25  .menubar {
26    background-color: black;
27    position: fixed;
28    top: 0;
29    left: 0;
30    width: 100%;
31    padding: 20px 100px;
32    box-shadow: 0 10px 8px black;
33    display: flex;
34    justify-content: space-between;

```

```

34     align-items: center;
35     z-index: 100;
36 }
37 .logo {
38     font-size: 2em;
39     user-select: none;
40 }
41 .logo:hover{
42     cursor: default;
43     color: #afeeee;
44     transition-duration: 1s;
45 }
46 .logo:not(:hover){
47     cursor: default;
48     color: white;
49     transition-duration: 1s;
50 }
51 .navigation a{
52     color: white;
53     text-decoration: none;
54     position: relative;
55     font-size: 1.2em;
56     margin-left: 2.5em;
57 }
58 .navigation a::after{
59     content: '';
60     position: absolute;
61     left: 0;
62     bottom: -0.3em;
63     width: 0;
64     height: 0.15em;
65     border-radius: 0.4em;
66     background-color: white;
67     transition: 0.3s ease-in-out;
68     pointer-events: none;
69 }
70 .navigation a:hover::after, .navigation a:focus::after {
71     width: 100%;
72 }
73 .navigation .loginBtn {
74     font-size: 1.2em;
75     width: 5.5em;
76     height: 2.5em;
77     background: transparent;
78     border: 0.15em solid;
79     outline: none;
80     border-radius: 0.4em;
81     cursor: pointer;
82     margin-left: 2.5em;
83 }

```

```

84 .navigation .loginBtn:hover{
85     background-color: white;
86     border-color: white;
87     color: black;
88     transition: 0.5s;
89 }
90 .navigation .loginBtn:not(:hover){
91     background-color: black;
92     color: white;
93     transition-duration: 0.5s;
94 }
95
96
97 /* Designing the login form and the register form */
98
99 .wrapper{
100     position: fixed;
101     top: 10em;
102     width: 25em;
103     height: 35em;
104     background: transparent;
105     backdrop-filter: blur(20px);
106     box-shadow: 0 0 30px rgba(0, 0, 0, 0.5);
107     border: 0.15em white solid;
108     border-radius: 2em;
109     display: flex;
110     justify-content: center;
111     align-items: center;
112     z-index: 10;
113     overflow: hidden;
114     transform: scale(0);
115     transition: transform .5s ease, height .2s ease;
116 }
117 .wrapper.popup{
118     transform: scale(1);
119 }
120 .closeButton{
121     position: absolute;
122     top: 0;
123     right: 0;
124     background-color: transparent;
125     width: 1.5em;
126     height: 1.5em;
127     color: white;
128     font-size: 2em;
129     display: flex;
130     align-items: center;
131     justify-content: center;
132     border-bottom-left-radius: 20px;
133     transition-duration: 0.5s;

```

```

134 }
135 .closeButton:hover {
136     cursor: pointer;
137     background-color: white;
138     color: black;
139 }
140 .loginForm,
141 .registerForm{
142     width: 70%;
143     padding: 1em;
144     background: transparent;
145     display: flex;
146     flex-direction: column;
147     z-index: 11;
148 }
149 .wrapper .loginForm{
150     transition: .18s ease;
151     transform: translateX(0);
152 }
153 .wrapper.active .loginForm{
154     transition: .18s ease;
155     transform: translateX(-30em);
156 }
157 .registerForm{
158     position: absolute;
159     transition: .18s ease;
160     transform: translateX(30em);
161 }
162 .wrapper.active .registerForm{
163     transition: .18s ease;
164     transform: translateX(0);
165 }
166 .loginForm *,
167 .registerForm *{
168     background: transparent;
169 }
170 .loginForm h2,
171 .registerForm h2{
172     font-size: 2em;
173     text-align: center;
174 }
175 .inputBox{
176     position: relative;
177     width: 100%;
178     height: 3em;
179     border-bottom: 0.1em solid white;
180     margin: 30px 0;
181 }
182 .inputBox label{
183     position: absolute;

```

```

184     top: 50%;
185     left: 0.1em;
186     transform: translateY(-50%);
187     pointer-events: none;
188     transition-duration: 0.5s;
189 }
190 .inputBox input:focus~label,
191 .inputBox input:valid~label{
192     top: 0.001em;
193 }
194 .inputBox input{
195     color: white;
196     width: 100%;
197     height: 100%;
198     outline: none;
199     border: none;
200     font-size: 1em;
201     padding: 0.1em 1.5em 0 0;
202 }
203 .inputBox .icon{
204     position: absolute;
205     right: 0.1em;
206     top: 0.6em;
207     font-size: 1.2em;
208 }
209 .rememberForgot{
210     display: flex;
211     flex-direction: row;
212     justify-content: space-between;
213     font-size: 0.8em;
214     margin: -1.2em 0 2em 0;
215 }
216 .rememberForgot label input{
217     accent-color: white;
218     margin-right: 0.3em;
219 }
220 .rememberForgot a{
221     color: white;
222     text-decoration: none;
223 }
224 .rememberForgot a:hover{
225     text-decoration: underline;
226 }
227 .btn{
228     color: white;
229     width: 100%;
230     margin-bottom: 1.5em;
231     padding: 1em;
232     border: white solid 0.175em;
233     border-radius: 1em;

```

```

234     cursor: pointer;
235     transition-duration: 0.5s;
236 }
237 .btn:hover{
238     background: white;
239     color: black;
240 }
241 .loginRegister,
242 .loginLink{
243     width: 100%;
244     font-size: 0.9em;
245     display: flex;
246     flex-direction: row;
247     justify-content: space-between;
248 }
249 .loginRegister a{
250     color: white;
251     font-weight: 600;
252     text-decoration: none;
253 }
254 .loginRegister a:hover{
255     text-decoration: underline;
256 }
257 .registerForm .loginRegister{
258     margin: 0;
259     padding: 0;
260     width: 100%;
261     display: flex;
262     flex-direction: row;
263     justify-content: space-between;
264 }
265 .registerForm .loginRegister p{
266     width: 25em;
267 }
268
269 /* Designing the contents of the landing page*/
270
271 .landingPagePlaceholder {
272     background-color: black;
273     display: flex;
274     flex-direction: column;
275     justify-content: space-around;
276     align-items: center;
277     user-select: none;
278 }
279 .landingPagePlaceholder #text1{
280     font-size: 10em;
281     background-color: transparent;
282     position: absolute;
283     z-index: 5;

```

```

284     bottom: 1.3em;
285 }
286 .landingPagePlaceholder #text2{
287     font-size: 10em;
288     background-color: transparent;
289     position: absolute;
290     z-index: 5;
291 }
292 .landingPagePlaceholder #text3{
293     font-size: 10em;
294     background-color: transparent;
295     position: absolute;
296     top: 14em;
297     z-index: 5;
298 }
299 .landingPagePlaceholder #text4{
300     font-size: 10em;
301     background-color: transparent;
302     position: absolute;
303     top: 15em;
304     z-index: 5;
305 }
306
307 /* Animating the 'FREE' text to change color when hovering
308 and vice versa. */
309
310 .landingPagePlaceholder #text4:hover{
311     cursor: default;
312     color: #afeeee;
313     transition-duration: 1s;
314 }
315 .landingPagePlaceholder #text4:not(:hover){
316     cursor: default;
317     color: white;
318     transition-duration: 1s;
319 }
320
321 /* Styling the images on the landing page */
322
323 .landingPagePlaceholder img{
324     width: 80%;
325     filter: grayscale(100%) brightness(50%);
326     margin-left: auto;
327     margin-right: auto;
328     margin: 5em;
329     position: relative;
330     top: 5em;
331     border-radius: 1em;
332 }

```



```

333 /* In terms of efficiency and conciseness of code, this is
      very bloated. I could have accomplished the same thing
      in a much sleeker way, but for a first attempt it will
      suffice. */

```

Listing 9: landingpage.css

Within the CSS i created animations on hover to make the buttons dynamic. The login form activates via a wrapper that makes it visible, applying appropriate styles.

```

1  // Defining constants to make writing easier and more
      concise
2
3  const wrapper = document.querySelector('.wrapper');
4  const loginLink = document.querySelector('.loginLink');
5  const registerLink =
      document.querySelector('.registerLink');
6  const loginBtn = document.querySelector('.loginBtn');
7  const closeButton = document.querySelector('.closeButton');
8
9  // Detects when user clicks register button and adds class
      to .wrapper
10
11 registerLink.addEventListener('click', ()=> {
12   wrapper.classList.add('active');
13 });
14
15 // Detects when user clicks login button and removes class
      from .wrapper
16
17 loginLink.addEventListener('click', ()=> {
18   wrapper.classList.remove('active');
19 });
20
21 // Detects when user clicks on menubar login button and
      adds popup class to .wrapper
22
23 loginBtn.addEventListener('click', ()=> {
24   wrapper.classList.add('popup');
25   wrapper.classList.remove('active')
26 });
27
28 // Detects when user clicks on cross icon and removes popup
      class from .wrapper
29
30 closeButton.addEventListener('click', ()=> {
31   wrapper.classList.remove('popup');
32 });

```

Listing 10: app.js

Here is the simple JavaScript that adds event listeners to the various buttons in the navbar. It adds active and pop up bars to toggle various CSS classes that will make the login and register pages visible.

4.3 Iteration 3

At this point, work began on the backend of the website. I had initialized the Django project, configured the various settings files to suit my needs and instantiated the database (SQLite3), applying the default migrations which set up the administration page (this included the default User and Groups models). As with any framework, there are a lot of boilerplate files created automatically, which, although contain important information, would be unnecessary to document here. For each iteration from now on, only the essential files will be included.

In terms of the additional functionality I had added at this point, I had routed the URLs to the appropriate views which simply rendered the static pages. I had essentially hooked up the frontend components I had created in the previous iterations to the backend which at this point only serves the correct static files from the HTTP request received when the user enters the URL.

Here is an overview of the file structure of the project at this point in time.

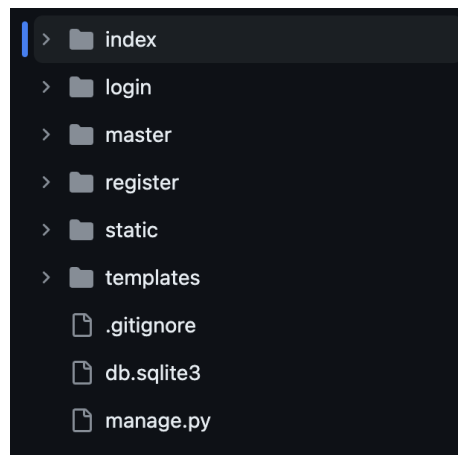


Figure 16: Top level file structure

There is quite a lot to unpack here so I will go one by one and explain the functions of each directory.

4.3.1 Index, login, master, register

Each of these directories represent apps. It is a design methodology that places emphasis on modularity. Each app should be fully independent and contain all the necessary components and static files to function independently. The idea

is that you can 'plug and play' each app, inserting it into a project and reusing it where needed. Each app consists of a couple of key files. A `urls.py`, `views.py`, `models.py`, `forms.py` and more depending on the range of functionality required by the app. Even within an app, modularity is encouraged to separate the code into different modules that serve different purposes.

Within this iteration of the project, there are 4 apps: `index`, `login`, `master`, `register`. `Master` is the entry point for the application. When the `'runserver'` command is ran from the `manage.py` script, the project is configured to enter the application from the `master` application which contains the `urls.py` for the root URL. It also contains the `settings.py` file that contains all the configuration for the project.

4.3.2 Static

Within this iteration, since there weren't too many components, I collected all of the static files underneath one directory within the root directory of the project. For production, you would typically create static directories within each of your apps. The static directory is divided into sub-directories for each app. Each contains the necessary static files (that is, CSS, images, JS, audio etc.) needed to be accessed by the various templates. Configured in the `settings.py` in the `master` app, the framework knows where to look for these static files as defined when I created the Django project.

4.3.3 Templates

These are the templates for the project. Templates are modified HTML files using the Django templating language which allows context variables to be manipulated within the markup directly. That is, you can dynamically pass in values and perform operations on them such as looping over them. When the view renders the template upon a request for that file, the templating engine built into Django dynamically creates the HTML file to be served to the user depending on the data passed through the view function. It is far more efficient and allows dynamic web pages to be created based on data retrieved from the models.

4.3.4 .gitignore

A file that specifies files to ignore when committing to the git repository. For example, you wouldn't want to commit sensitive information, encryption keys and the like when creating a repository in production.

4.3.5 db.sqlite3

This is the database for the project. Currently it only has the default user and group models.

4.3.6 manage.py

A python script that allows you to manage the Django project. Also provides functionality for a simple development server that runs on the local host 127.0.0.1 at port 8000. Not suitable for production but entirely sufficient for development purposes.

4.3.7 Backend functionality

```
1 from django.contrib import admin
2 from django.urls import include, path
3
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6     path('', include('index.urls')),
7     path('login/', include('login.urls')),
8     path('register/', include('register.urls')),
9 ]
```

Listing 11: master/urls.py

This is the entry point for the application. When the user enters a URL that corresponds to the domain name of the website, Django will enter from this file to parse the URL and decide which file to serve. When the root URL is followed by 'login/', the login.urls is included. That means that the urls.py from the login app is called and the rest of the URL is cut off and sent to that module for further processing.

As an example, lets have a look at what happens when the root URL is entered and the URL is sent to index.urls to resolve.

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.homepage, name='homepage')
6 ]
```

Listing 12: index/urls.py

As can be seen, the urls.py assigns the root URL to the views.homepage, which is an imported module from the views.py file within the same app. This means that it is now up to the homepage function within the views file to decide what to serve to the user.

```
1 from django.shortcuts import render
2
3
4 def homepage(request):
5     return render(request, 'index.html')
```

Listing 13: index/views.py

Here is the very simple views.py file which contains only the homepage function. This function takes in the HTTP request as a parameter (this is conventional, even if the response is static by nature and doesn't require any information from the HTTP request) and returns the rendered HTML template that is index.html.

4.3.8 Iterations on the frontend

Within this iteration, I also made some modifications to the HTML templates. Namely, I modularized them further to separate the landing page, login page and register page into separate templates rendered in different URLs.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 {% load static %}
4     <head>
5         <meta charset="UTF-8" />
6         <meta name="viewport" content="width=device-width,
7             initial-scale=1.0" />
8         <title>welcome to europa!</title>
9         <link rel="stylesheet" href="{% static
10             'index/styles/index.css' %}" />
11     </head>
12     <body>
13         <!-- Content for menubar -->
14         <header class="menubar">
15             <h2 class="logo">europa fitness</h2>
16             <nav class="navigation">
17                 <button class="loginBtn">Login</button>
18             </nav>
19         </header>
20
21         <!-- Main bulk content for the landing page -->
22
23         <div class="landingPagePlaceholder">
24             <img src='{% static
25                 'index/img/gymstockimage.jpg' %}' alt="Man
26                 lifting weights">
27             <h1 id="text1">exercise.</h1>
28             <h1 id="text2">diet.</h1>
29             <h1 id="text3">all for</h1>
30             <h1 id="text4">FREE.</h1>
31             <img src='{% static
32                 'index/img/gymstockimage2.jpg' %}' alt="Rack
33                 of weights">
34         </div>
35         <script src="{% static 'index/js/index.js'
36             %}"></script>
37     </body>
```

31 </html>

Listing 14: templates/index.html

Here you can begin to see some of the templating language in action. At the beginning of the file, all of the static files needed are loaded from their respective directories and template expressions are used to reference the directories of the static files relative to their parent directories. The template engine will convert these to absolute urls upon rendering. This approach is far more maintainable as you can move static files around the project and simply change the `STATIC_FILES_DIR` setting in the `settings.py` file and the template engine will recognise this and resolve the URLs based on the new settings rather than you having to go through every template and manually change the URLs.

```
1 <!doctype html>
2 <html lang="en">
3 {% load static %}
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport"
7         content="width=device-width, user-scalable=no,
8         initial-scale=1.0, maximum-scale=1.0,
9         minimum-scale=1.0">
10    <meta http-equiv="X-UA-Compatible"
11        content="ie=edge">
12    <title>Login to account</title>
13    <link rel="stylesheet" href="{% static
14        'login/styles/login.css' %}">
15    <link href='https://unpkg.com/boxicons@2.1.4
16        /css/boxicons.min.css' rel='stylesheet'>
17 </head>
18 <body>
19     <div class="wrapper">
20         <form action="">
21             <h1>Login</h1>
22             <div class="inputbox">
23                 <input type="text"
24                     placeholder="Username" required>
25                 <i class='bx bxs-user'></i>
26             </div>
27             <div class="inputbox">
28                 <input type="password"
29                     placeholder="Password" required>
30                 <i class='bx bxs-key'></i>
31             </div>
32             <div class="rememberForgot">
33                 <label><input type="checkbox">
34                     Remember me?</label>
35                 <a href="#">Forgot Password?</a>
36             </div>
```

```

29         <button type="submit"
30             class="button">Login</button>
31         <div class="registerLink">
32             <p>Don't have an account? <a
33                 href=" ../register">Register</a></p>
34         </div>
35     </form>
36 </div>
</body>
</html>

```

Listing 15: templates/login.html

```

1  <!doctype html>
2  <html lang="en">
3  {% load static %}
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width,
7          user-scalable=no, initial-scale=1.0,
8          maximum-scale=1.0, minimum-scale=1.0">
9      <meta http-equiv="X-UA-Compatible" content="ie=edge">
10     <title>Register for an account</title>
11     <link rel="stylesheet" href="{% static
12         'register/styles/register.css' %}">
13     <link
14         href='https://unpkg.com/boxicons@2.1.4/css/boxicons.min.css'
15         rel='stylesheet'>
16
17 </head>
18 <body>
19     <div class="wrapper">
20         <form action="">
21             <h1>Registration</h1>
22             <div class="inputbox">
23                 <div class="inputfield">
24                     <input type="text" placeholder="Full
25                         Name" required>
26                     <i class='bx bxs-user'></i>
27                 </div>
28                 <div class="inputfield">
29                     <input type="text"
30                         placeholder="Username" required>
31                     <i class='bx bxs-user'></i>
32                 </div>
33             </div>
34             <div class="inputbox">
35                 <div class="inputfield">
36                     <input type="email"
37                         placeholder="Email" required>

```

```

30         <i class='bx bxs-envelope'></i>
31     </div>
32     <div class="inputfield">
33         <input type="number"
34             placeholder="Phone Number"
35             required>
36         <i class='bx bxs-phone'></i>
37     </div>
38 </div>
39 <div class="inputbox">
40     <div class="inputfield">
41         <input type="password"
42             placeholder="Password" required>
43         <i class='bx bxs-key'></i>
44     </div>
45     <div class="inputfield">
46         <input type="password"
47             placeholder="Confirm Password"
48             required>
49         <i class='bx bxs-key'></i>
50     </div>
51     <label>Already have an account? <a
52         href=" ../login">Click here!</a></label>
53     <button type="submit"
54         class="button">Register</button>
55 </form>
56 </div>
57 </body>
58 </html>

```

Listing 16: templates/register.html

It's a similar story with these other 2 templates. Static files are collected and served and the template is rendered and sent to the user when the appropriate URL is accessed. That about sums up the backend functionality added within this iteration.