CFG Nanodegree Fitness Project Report

Fitly: *Fitness Workout Generator*

By Alexandra Cook, Delia Neamtu-Cucu, Raghad Zuraiki,
Regina Osei-Bonsu, Yasmine Leghnider

December 2021

Table of Contents

1 - Introduction

We are proud to introduce Fitly, our group coding project. Our aim for this project was to use the knowledge gained from the CFG Software Nanodegree to create a fitness workout generator. Our project objectives included:

- Designing a fitness workout web application with tailored branding
- Implementing Python logic to allow users to interact with the web application and receive a response/output
- Use of an SQL database
- Use of public APIs (YouTube Data API and Exercise DB API)
- Running the project from a main() file
- Creating a well-organised folder structure

This report explores our project successes, outcomes and challenges over the 4 weeks where Fitly was created. Appendix 1.0 displays our project roadmap in more detail.

2 - Background

What kind of problem does Fitly solve?

Fitly offers fitness-conscious users the ability to pinpoint their exact workout specifications for an innovative and completely personalisable workout experience. Whether it be as a video or as a written guide, Fitly solves the "how do I get started?" conundrum that people often encounter when opting to improve their health and wellness through exercise. Fitly ensures that anyone, regardless of fitness level, experience or confidence, can achieve their fitness goals.

What does Fitly do?

Fitly is a web-based application that recommends a workout for a user based on the user's target muscle group and whether they want to perform a workout at home or in the gym. Fitly integrates with an API called GoogleClientAPI to offer a user a repository of over 200 of the top-recommended workout videos hosted on YouTube. Fitly also allows the user to request a written workout guide which uses the Exercise DB API to build a new custom workout, every time based on the user's chosen body part they want to target in their workout.

How does Fitly work?

As Appendix A 2.0 displays, the user has two dropdowns where they can select the muscle group they want to workout and where they would like to perform their workout (at home or in the gym). Once these dropdown options have been selected, a user is then able to choose from three option buttons to generate a personalised workout of their choice:

1. **Option 1:** *Get a personalised YouTube workout video:* When a user clicks this option, Fitly displays the user three YouTube videos based on the user's dropdown selections
2. **Option 2:** *Get a personalised written workout guide:* This option displays a written workout guide to the user. The guide appears as a table, and includes the exercise name, equipment needed and a GIF demonstration of the exercise. These workouts come from the Exercise DB API and the app selects random exercises based on the user's chosen body part they want to target.
3. **Option 3:** *Still unsure what to workout? Get a random YouTube workout video:* This option caters for the users who want a random recommendation from any of the workout categories. When a user selects this option, Fitly recommends one random workout video from the YouTube API.

Users will never get the same recommendations due to use of the random module, even if their preferences remain the same. Fitly also has additional functionality, where for options 1 and 3 where a YouTube video is generated, the user is able to play the video from the web app as the video is embedded in the application. An example user flow can be found at Appendix A 3.0.

3 - Specifications and Design

3.1 - Requirements

Our system has both key technical and non-technical system requirements which are sorted below in priority order.

3.1.1 - Technical Requirements

- The system must provide the user with a dropdown to select their preferred muscle group to workout (abs, arms, back, legs or shoulders)
- The system must provide the user with a dropdown to select their preferred workout location (at home or in the gym)
- The system must generate a recommended workout based on the two user selections (muscle group and workout location)
- The system must present the user an option to generate a YouTube video or a written workout guide based on their selections and content preferences
- For video workouts, the system must provide an interactive preview of a recommended workout video that is embedded in the web application and can be played by the user
- The system must allow a user to select a completely random workout video from any of the categories if they are unsure which workout to choose
- The system must provide the user with an option to sign up for a Fitly account, login and logout
- The system must present the user with an easy, intuitive and modern user interface

3.1.2 - Non-Technical Requirements

- **Accessibility**
    - The system will use large, clear text for visually-impaired users

    - The system will use best practice, contrasting colours for users with visual impairments or colour vision deficiencies using these guidelines
- **Documentation**
    - The code will be well-documented to ensure that any future development of the system can be achieved efficiently
- **Testability**
    - The system should be unit-tested to ensure output code is high-quality
    - The system should be able to handle unexpected errors
- **Portability**
    - The system design will be responsive based on the user's device

3.2 - Design and architecture

We decided as a group that the most appropriate architecture would be the three-tier client-server model (Figure 1). We chose this model as it is a well-established software application architecture that organises applications into three computing tiers. For Fitly, this meant that the client side on the Fitly web page would interact with the Flask application via the internet (tier 1), the Flask application would display the front end using HTML templates (tier 2) and also send the API requests to YouTube and manage calls to the MySQL database (tier 2) and the MySQL database would also communicate with the database manager (tier 3). We also used imported modules and libraries to enhance Fitly's functionality.

Some examples include:

- googleapiclient: a client library for Google's discovery based APIs (which includes the YouTube API) to allow building the search query for the YouTube video data
- random: to allow for randomisation of the generated YouTube videos
- requests: a simple, yet elegant HTTP library to allow sending of HTTP GET and POST requests
- os: a module that provides functions for interacting with the operating system

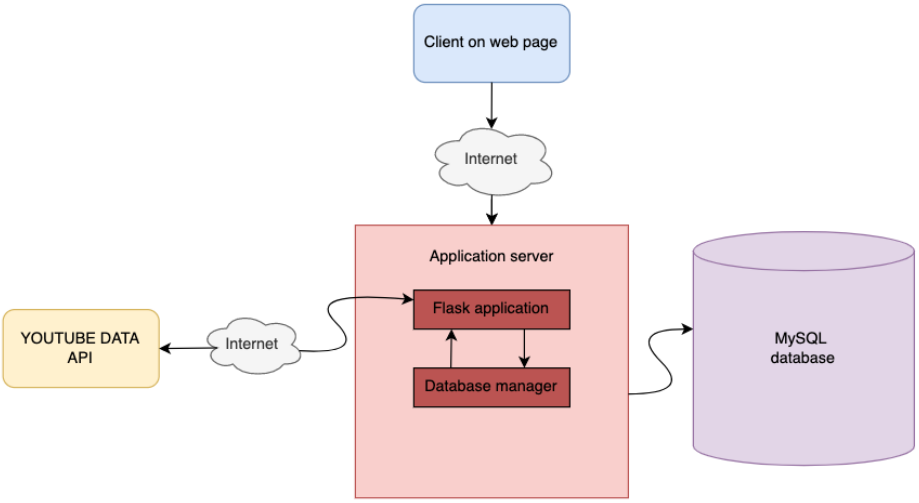


Figure 1: *A diagram displaying our approach to the three-tier client-server model and the interactions between the client, Flask application, SQL database and API*

4 - Implementation and Execution

4.1 - Development approach and team member roles

The Fitly project team followed Agile methodology principles and an iterative approach to our project development using the Scrum framework. We first held a sprint planning meeting in which we discussed the plan going forward, assigned initial tasks and drew up our main aims and objectives. We also created the main user stories and assigned Alexandra Cook as Scrum Master, who initiated a mini SWOT-style analysis, as displayed here, which each member completed; as a result, tasks were assigned based on our strengths/preferences. Due to the short project timescales, we decided to hold a stand-up every three days. This would include a mix of both video meetings via Google Meets and written updates on Slack to ensure the entire team had progress visibility. Following project completion, the team met during a sprint retrospective to reflect on our progress and discuss future improvements (explored in section 5).

To manage our code, we used a Trello board to track our tasks (see here for Trello snapshots), a shared Google Drive to manage the project documentation and a shared GitHub repository where we ensured we regularly reviewed each other's code to prevent logic errors and ensure all requirement cases were fully implemented.

Team roles and responsibilities

As a team, we maintained strong communication throughout the project, sometimes working independently but in tandem with other team members, or collectively as sub-team groups. Each individual's main team roles and achievements are outlined below:

Alexandra

- Led the project as Scrum Master
- Designed, styled and created the entire Fitly frontend system and navigation bar
- Implemented the user dropdown system and quiz logic
- Connected the API to the frontend and backend to ensure system flow
- Embedded YouTube videos so they could be played in the application
- Designed the Fitly branding and colour scheme

Delia

- Researched the Youtube API and wrote the function to generate a random workout video (using the keyword "workout" to query the API)
- Wrote functionality to do the hashing of users' passwords
- Made the login and sign up pages visible only when the user is not logged in, and the home, profile and logout pages if the user is logged in
- Added frontend styling details
- Did unit testing for the Flask app

Raghad

- Contributed towards setting up and starting the report file
- Contributed towards unit testing

Regina

- Created the database that would store the video URLs should the YouTube API not work due its unit limitation. However, as we were able to successfully use the YouTube API, the videos URL database was no longer needed
- Created the initial quiz functions that would ascertain the users' fitness needs. For user ease, we instead opted for a frontend quiz system and also relied on the Youtube API to feed the relevant videos to the user
- Contributed to the login authentication of the web application code
- Did the unit testing for the YouTube Workout and User to Database connection

Yasmine

- Researched the YouTube API and wrote the class for the personalised workout
- Researched the Exercise DB API and wrote the class for the written workout
- Ensured our code base adhered to OOP
- Worked on the Python MySQL DB connection (login authentication, session creation) and storing the YouTube results in the database
- Created the class responsible for checking whether a user exists in the database
- and creating the user if not
- Worked on profile page and embedded URLs so users can watch them back at any time
- Worked on saving the written workout results as a CSV as a workaround the MySQL database not accepting the data type returned by the Exercise DB API

4. 2 - Challenges during the implementation process

Although as a team we enjoyed creating the project, the short timescales and limited collective coding experience made it a challenging project. Our main challenges were that we sometimes lacked the time outside of Nanodegree commitments, 3 team members with full time jobs and other personal commitments, which meant we often could not meet all together at the same time. We also ran the entire project one team member down, as a member was unable to contribute until several days before the deadline which impacted the output we thought we could achieve at the outset. Despite these challenges, we adapted as a team and found solutions. For example, we introduced daily Slack updates which meant that if a team member could not meet at a certain time, we would have their update in message form to discuss as a team. We regularly reviewed what was realistically achievable and adapted our project as required.

5 - Testing and Evaluation

5.1 - Testing strategy

We unit tested the Flask app to check for a 200 response on the web pages when the user is logged in and a 404 response when the user is logged out (meaning they must login to access the page). We also tested that the quiz form loads correctly on the home page. Furthermore, we tested that choosing a written workout redirects the user to the correct results page and that the results page displays 3 personalised videos when the user completes the quiz form. We also unit tested the YouTube API workouts to check that our web app was connected to the API and therefore providing relevant videos to fit the user requirement. We also tested that the results received by the user were completely random. Moreover, we tested to see if the YouTube API would create URLs that would be stored in the database, and finally tested that users could successfully get workouts from YouTube. While testing this class, the mock framework was implemented via calling on patch() which was done to isolate and focus solely on the code being tested.

Finally, we unit tested the user to database connection. We mocked the connection call so that when the connection is called it will return true. We tested the connection to the database to check that the MySQL function is called successfully and that no error messages would occur when the function for commit to database was called. Finally we tested the create user table to check that the execution function is called successfully, the response will run true, meaning a table is created.

5.2 - System limitations

One of the main system limitations is that we weren't able to get around the use of copyright music in our YouTube videos. As videos are retrieved from the YouTube API they are subject to copyright music laws. Consequently, videos subject to copyright infringement will not be displayed to the client. Another limitation of our system arises from the way the YouTube API is structured: each call to the API uses up a certain number of units, ranging from 1 (getting a list of comments using the API, for example) to 1600 units per call. The API has a daily quota of only 10,000 units and our search query is worth 100 units, which means users can only make 100 calls to the API per day. Therefore, potential use of the application on a larger scale would only be possible if we were to be granted a quota extension.

We also faced a database limitation when it came to the Exercise DB API response. The JSON object that it returns is an array of lists, which isn't easy to store in a MySQL table. There are some possible workarounds, such as splitting the data up in python and storing the individual values in their own columns. Another option would have been to use MongoDB or another database that accepts more data structures. However, due to time limitations, we focused on storing the YouTube results data in the database. We also explored the option of converting the results page into a PDF using the pdfkit package but faced difficulties getting it to work. We

also explored the option of saving the written workout as a PDF, which worked so this is somewhat a workaround the database limitation.

5.3 - Evaluating future development opportunities

Given more time, the team discussed future improvements we could make.

These included:

- Styling the results page to fit with the rest of the project theme
- Improve the number of dropdowns a user has to personalise their workout from
- Allow logged in users to keep a history of the videos they watched
- Allow logged in users to rate the workouts recommended to them and see if AI could be included to make Fitly smarter and generate more tailored workouts
- Add in a workout diary where users could add/delete workout notes
- Store the written workout results in the database.

6 - Conclusion

In conclusion, our group project Fitly has achieved our project aims and objectives, and delivered the successful development of a modern, interactive and innovative Flask application that provides users with the opportunity to generate personalised fitness workouts based on their preferences. An ideal companion for any fitness-conscious user, Fitly not only promotes inclusivity by allowing users with any experience, confidence level or ability to achieve their fitness goals, but also solves a real-life conundrum faced by many people wishing to exercise.

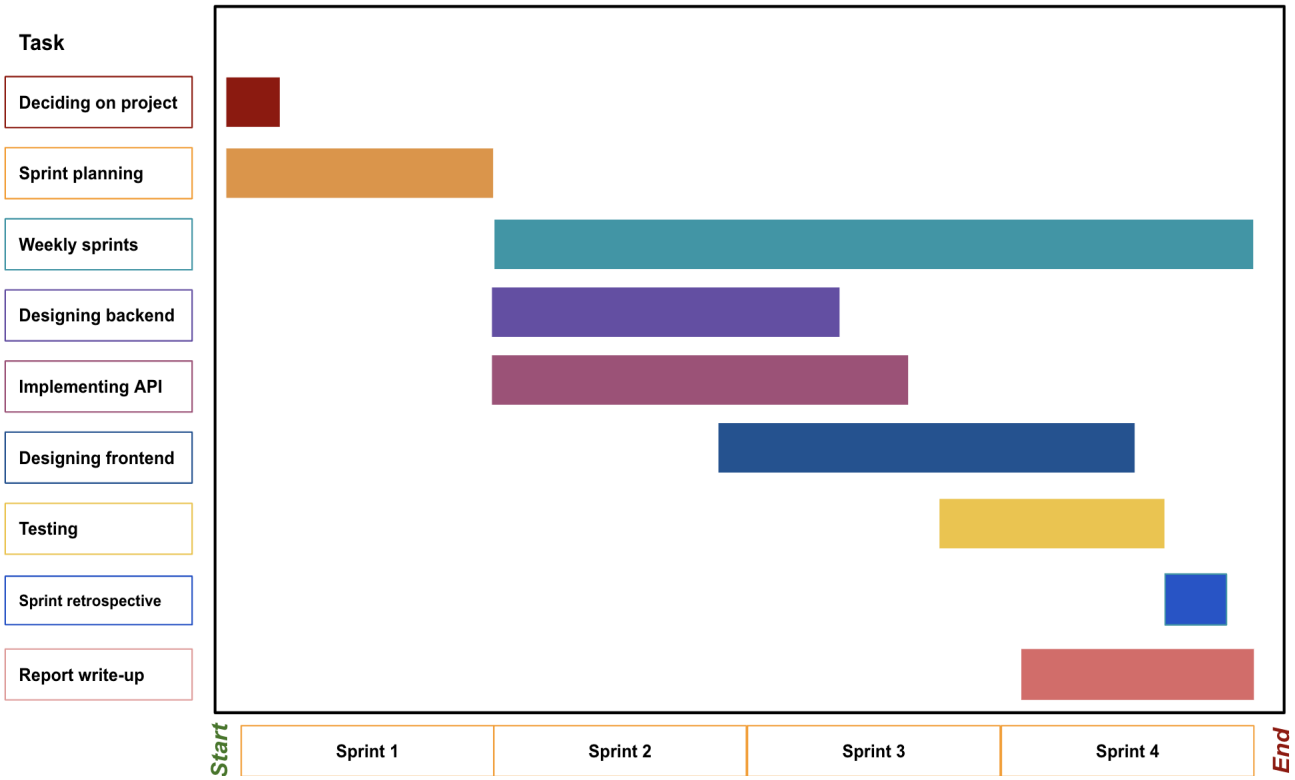
Although as a team we shared both successes and unforeseen challenges, we are exceptionally proud of the final outcome, particularly with just ~12 weeks of Python/SQL coding experience each. We thoroughly enjoyed pulling together the knowledge gained from the CFG Software Nanogree and working as a group to achieve a successful project outcome.

——————— End of Report ———————

Appendix

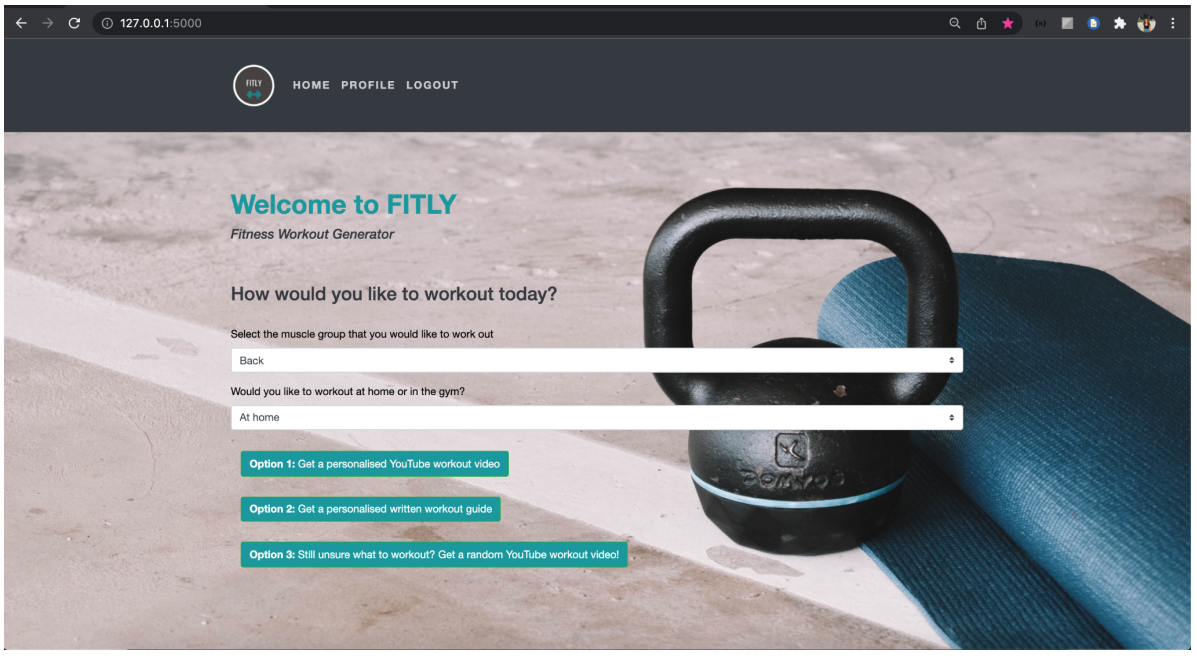
A 1.0

Fitly project roadmap from start of project (November 18) to end of project (December 26)



A 2.0

A screenshot of the Fitly home screen, user dropdown and selection button options.

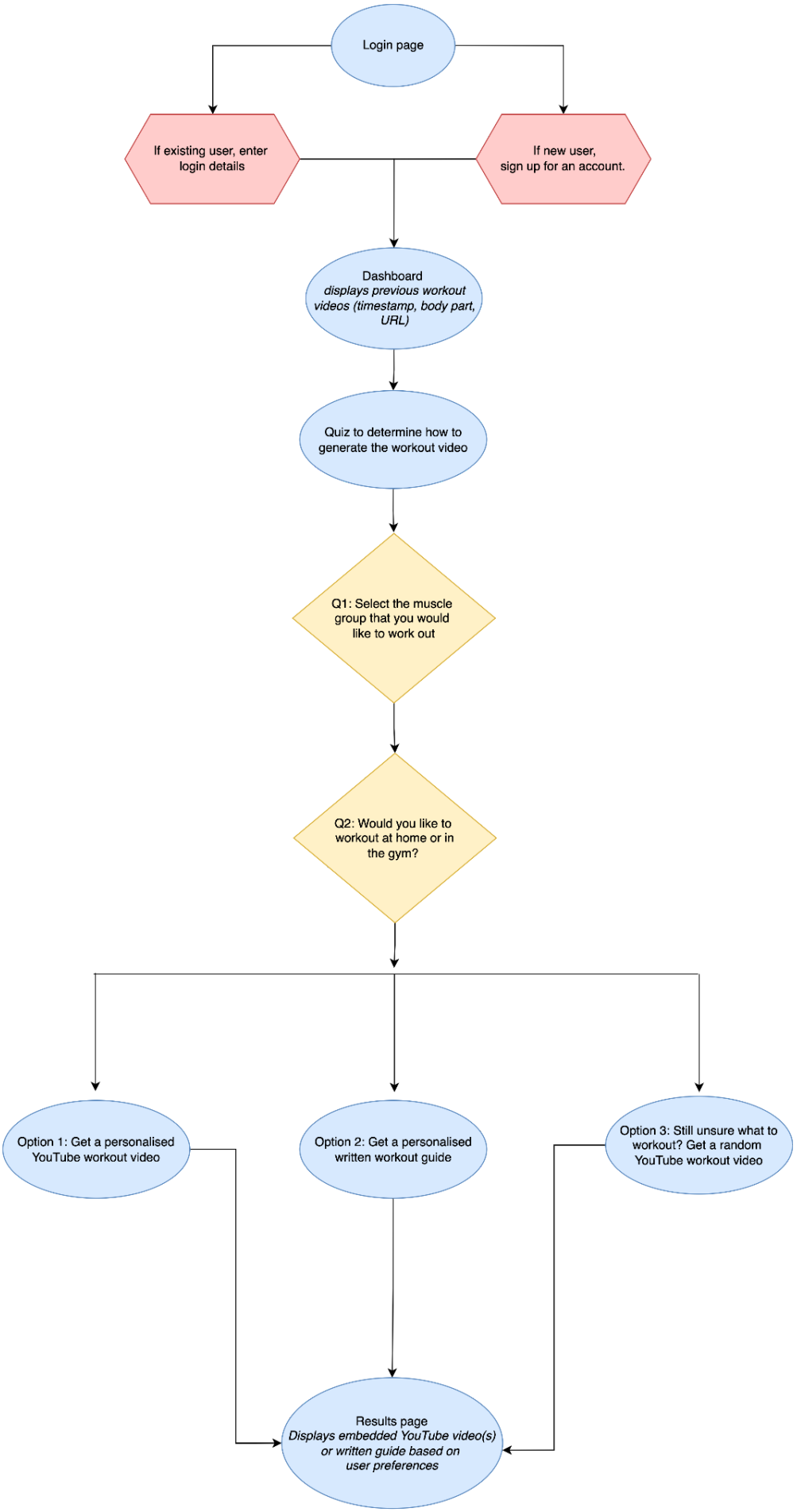


A 3.0

An example of a user flow diagram, from logging in and signing up, to workout generation for the Fitly app (next page).

Key:

- Blue shape: Functionality on a webpage on the Fitly web application
- Red shape: Authentication logic
- Yellow shape: Frontend user dropdown selections


Login page
If existing user, enter login details
If new user, sign up for an account.
Dashboard
displays previous workout videos (timestamp, body part, URL)
Quiz to determine how to generate the workout video
Q1: Select the muscle group that you would like to work out
Q2: Would you like to workout at home or in the gym?
Option 1: Get a personalised YouTube workout video
Option 2: Get a personalised written workout guide
Option 3: Still unsure what to workout? Get a random YouTube workout video
Results page
Displays embedded YouTube video(s) or written guide based on user preferences

A 4.0

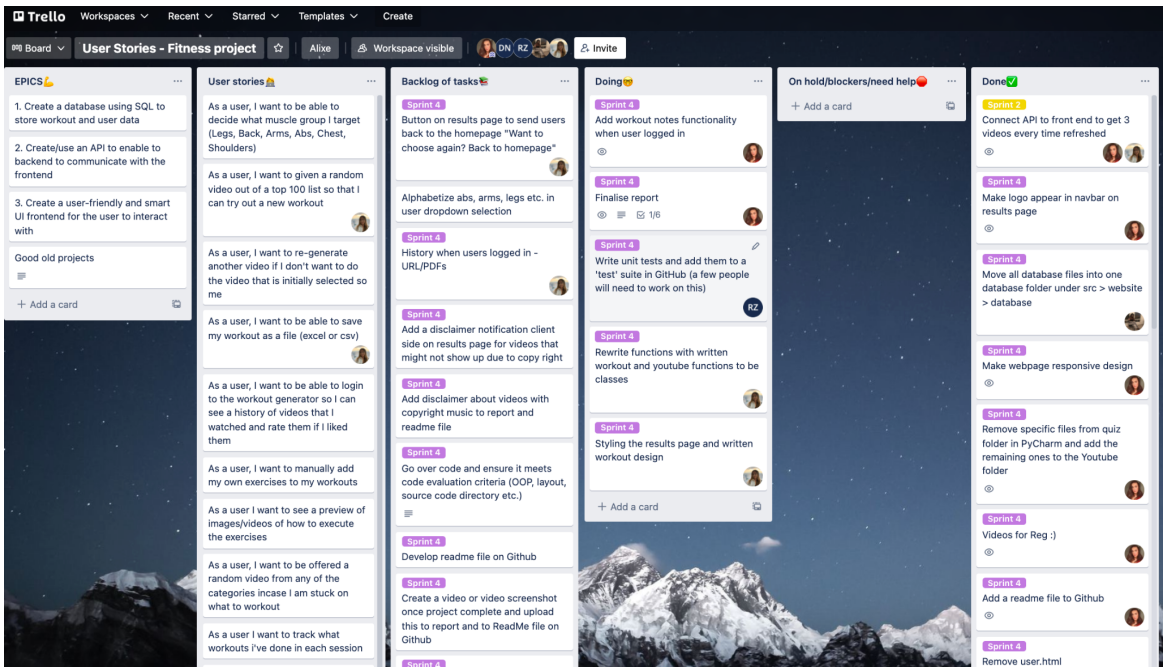
Team SWOT analysis completed by all team members prior to project kick-off.

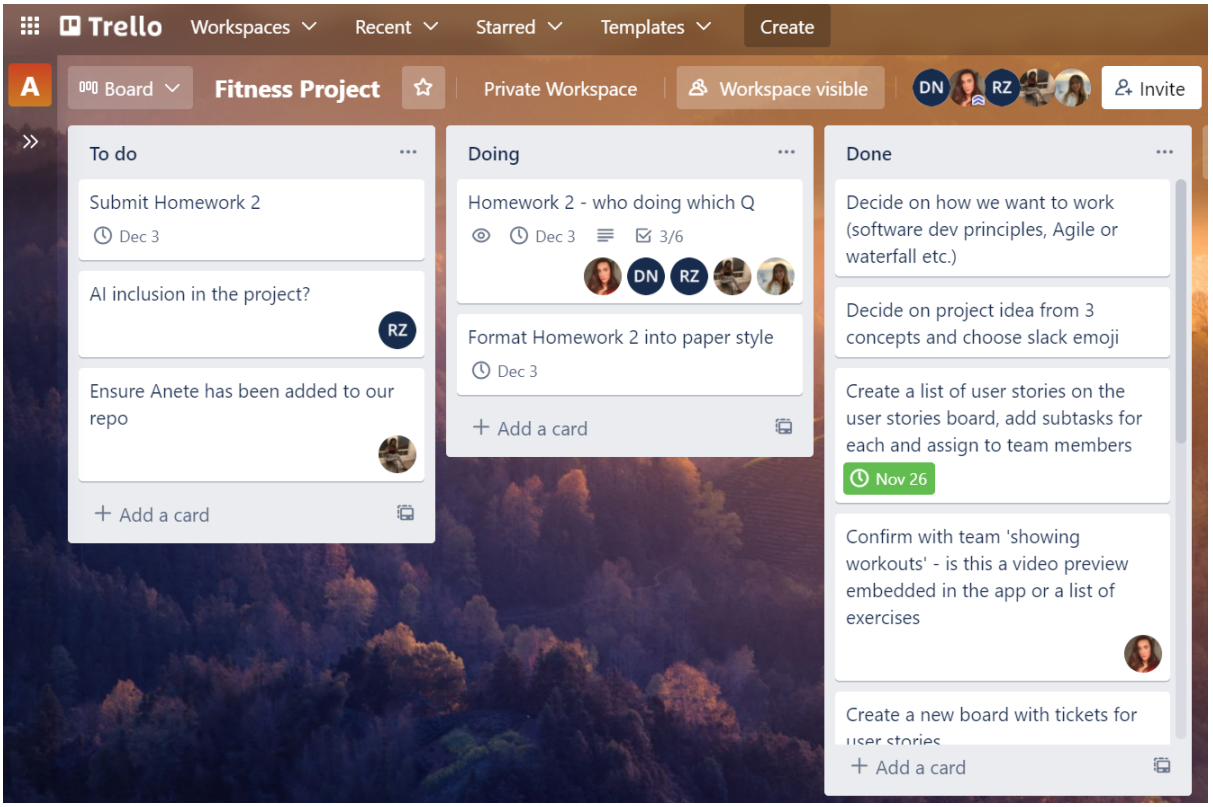
Team SWOT

Name	Company	Background	Availability	Strengths	Weaknesses	Preferred project role(s)
Alixe	Deloitte (London)	Pharmaceuticals Biomedical Science (degree)	Friday evenings Weekends preferred	Databases Front-end UI design	Unit testing	SQL database bits Project management Creativity bits Front-end development
Delia	Deloitte (London)	Accounting and Management (degree)	Flexible	Back-end	Front-end	Back-end (any)
Raghad	Zopa (London)	BSc Mathematics + MSc AI	Flexible (less communicative until 28th Nov)	AI	Unit testing Design	Happy to do anything really! (except writing the report)
Reg	Deloitte (London)	French and International Relations (degree)	Friday evenings Weekends preferred	Back-end	APIs	SQL database Creativity Workout list generation back-end(still a bit shaky on API but willing to get stuck in) Report writing
Yasmine	Deloitte (London)	French and History (degree)	Friday evenings Weekends preferred	Back-end	Front-end	Backend (any but I find server-side APIs and testing difficult!) Use of external APIs System design Project management Presentation slides

A 5.0

Screenshots from our group Trello boards. The top (blue) board focused on our epics, user stories, backlog of tasks and our task progress. The bottom board was for general project maintenance, such as repository checks and group homeworks.





——————— End of Appendix ———————