Software Engineering Group Project

COMP2002 / G52GRP: Final Report

Project information:

Project title: SnapMatch

Project sponsor: UNiDAYS

Academic supervisor: Marjahan Begum

Industry Sponsor: UNiDAYS

Industry Supervisor(s): Daniel Campos, Imogen Wilkinson, Alex Johnson

Team information:

Team number: 17

Jungfeng Zhu, 20411885, scyjz14

Eleanor McCartney, 20485393, psyem3

Adwoa Kumi-Addo, 20456736, psyak20

Desange Nkumu, 20421510, psydn5

Shuli Wang, 20411954, scysw3

Yongkang Wang, 20412957, scyyw17

Kian Surani, 20448650, psyks14

Mari Thorpe, 20465696, psymt9

Table of Contents

Resource links

- **Our git repository:** COMP2002 / 2023-2024 / team17_project · GitLab (nott.ac.uk)
  - **Branches:** Branches · COMP2002 / 2023-2024 / team17_project · GitLab (nott.ac.uk)
- **Minutes:** COMP2002 - GRP - Team 17 - 2023-2024 Notebook
- **Microsoft Planner:** SnapMatch - Planner (office.com)
- **Jest Testing Documentation:** Jest Documentation
- **React Testing Library:** React Testing Library Documentation
- **Attendance:** Attendance.xlsx
- **Timetable:** timetable
- **Retrospective form:** retrospective
- **Updated ui prototypes:** prototypes
- **Ofcom online safety guidelines:** New rules for online services: what you need to know - Ofcom
- **RAM reference:** GitHub - xinyu1205/recognize-anything: Open-source and strong foundation image recognition models.

1 Background & Understanding

1.1 Project Background

The project brief “SnapMatch” outlines the task of creating a software project for our stakeholders, UNiDAYS. The company focuses on aiding students in their time at university, and so students are the desired project demographic. The primary goal is to create a platform that encourages students to explore the outside world and create “memorable moments” whilst “fostering a sense of community.” The platform should make the world seem like a “scavenger hunt” where users can upload images of unique places or objects, challenging others to find and capture matching photos, inciting a sense of fun and freedom.

1.2 Project Interpretations

As a team we concluded that the projects' primary objective is to inspire users, particularly students, to take a break from their studying, by encouraging them to explore and discover the (potentially) new town or city they’ve moved to.

Moreover, this would have the additional benefit of students making new local friends, by engaging in collaborative challenges. To achieve this, users capture images of unique places or objects and create challenges for other users to find matching photos, like an online scavenger hunt. SnapMatch motivates users to seek out new places, landmarks, and experiences they may not have otherwise discovered.

The functional requirements included allowing users to capture images, submit challenges and match challenges. Furthermore, there needs to be a robust mechanism in place to prove that two photos count as a match.

1.3 Defining a match

We interpreted a match to be any two images that contain at least one of the same objects or landmarks. For example, an image containing a balloon and a bottle would be a match with another image that only contained a balloon, or a balloon and a hairbrush. We decided to leave it to the user to identify that it is the balloon, not the bottle, that they want users to match with.

1.4 Potential Requirements from Brief

These were the main specifications for the project:

1. Capture and issue ‘challenges’.
    a. Users can capture images of places or objects to issue as ‘challenges’ to the community.
2. Respond to challenges.
    a. Users capture images to respond to challenges from other users.
3. Mechanism to verify that two images ‘match’ i.e. contain the same location or object.
4. Stand-alone platform
    a. The product should operate independently from the main UNiDAYS platform.

1.5 Key Considerations

The project brief also highlighted key considerations, which we have explored in our report. These included:

1.5.1 Non-technical

- User Engagement - How will we ensure users stay using the app?

- Accessibility - How will we adapt our app to meet varying user needs? How will we ensure challenges are accessible to everyone?
- Safety vs Fun - How will we establish safeguards to ensure content is not harmful? How will we define clear user guidelines? How will we ensure this doesn't prevent enjoyment?

1.5.2 Technical

- Usability - How will we prioritise an intuitive and user-friendly interface for seamless navigation across devices? How will we create a visually appealing user experience?
- Image Recognition - How will we implement an image matching system to discern whether two submitted images contain the same object or location?
- Reliability - How will we handle performance under a high user load?
- Maintainability - How will we design the platform to be maintainable?
- Scalability - How will we handle a growing user base and increasing content?

1.6 Market Analysis

To answer some of these questions, we conducted a thorough market analysis to understand how similar products engage users and integrate technological solutions. We aimed to identify elements that would enhance user interaction and system efficiency in SnapMatch, as well as recognising potential pitfalls to avoid.

1.6.1 Pokémon Go

Pokémon Go is well-known for its ability to get users to engage with the real world, with core gameplay elements involving going to designated areas to find specific Pokémon, enhancing real-world engagement through geolocation-based technology.

This approach heavily relies on GPS technology, which we could adapt for SnapMatch by allowing users to **geo-tag challenges**, giving other users indication of where they need to explore in order to respond. However, there would need to be a cap on geolocation distance, ensuring challenges aren't impossible to complete due to inaccessibility.

1.6.2 Instagram

Both Instagram and SnapMatch involve image sharing as a central activity. Instagram places a strong emphasis on visual appeal and creativity, encouraging users to highlight their artistic side through photos and filters. The proposed product similarly relies on the visual aspect of challenges, requiring users to capture interesting and unique images.

Instagram increases user engagement through the ability to comment on posts and to direct message and follow other users. It could be a good idea to turn SnapMatch into a **social media platform** like Instagram to maintain user engagement, meeting the goal of making friends via the platform, with users being able to send friend requests to other users who they liked the challenges of.

1.6.3 Snapchat

Both the proposed product and Snapchat involve users creating and sharing content, but Snapchat is primarily a social media platform for already-made friends, with fast communication through sharing real-time pictures.

In comparison, SnapMatch focuses on uploading real-time challenges, exploring unique places or capturing specific objects, to the entire user base of the app. Analysing Snapchat, it could be a good

idea to include a **direct messaging** feature where users could send challenges directly to their friends, encouraging users who do not want to post publicly to still use the app.

1.6.4 BeReal

Whilst not similar in functionality, this newer social media application is what we envision when thinking of how challenges are presented. The app opens immediately to the camera, since uploading photos is the primary focus of the app. User engagement is maintained through the sharing of photos daily when a timer goes off, which can then be viewed in a friends' feed.

However, from team personal experience, the expectation of daily uploads proves to be overwhelming, with many users having lost interest and moved on from BeReal.

This gamification could be reflected in SnapMatch, e.g. through **notifications** of new challenges to respond to, but daily notifications should be avoided. Further, sharing a similar focus on photo uploading means that opening straight to the camera would be another useful feature.

1.6.5 Evaluation

Our research highlighted efficient methods for user engagement, including the use of GPS technology, social media features such as 'friending', direct messaging and commenting, and notifications encouraging users back to the app.

For SnapMatch, we intend to adapt some of these features as product requirements, specifically geo-tagging challenges, direct messaging between friends and notifications. Further, we recognise the importance of producing a user-friendly photo-upload-oriented interface, with a backend system capable of handling dynamic content updates without performance lags.

We have opted not to include daily notifications due to their overwhelming nature, and deciding to introduce a GPS distance cap, so that challenges remain accessible.

1.7 Non-technical Considerations Analysis

1.7.1 User Engagement

Challenges are not just to attract users initially, but to set-up the product for long term involvement. We explored features for maintaining user engagement in the market analysis above.

1.7.2 Accessibility

Since SnapMatch is a web app, anyone can access it. This means that people could upload photos of places that are only accessible by able-bodied people, for example. Unfortunately this would render some challenges impossible to complete for some users. We will ensure that the range of uploadable challenges is broad enough so users don't feel left out.

1.7.3 Safety vs Fun

It is vital that our future users feel safe whilst using our software, to create an enjoyable community experience, so we've looked into official safety guidelines.

Online Safety Act

As defined by the Ofcom Online Safety Act, our app is a user-to-user service (a place where people can share and create content), so we must adhere to the criteria outlined by Ofcom:

Assess the risk of harm from illegal content:

Our app definitely has the potential to be used harmfully, due to user-generated challenges.

Assess particular risk to children of harmful content:

There is the opportunity that children could access the platform since it is just a web app, but through requiring user account signup and targeting towards a student demographic, we hope to avoid this.

Make it easy to report illegal content, and content harmful to children:

Challenges should feature a report button, so that we can be swiftly alerted to users not using the site as attended. Since our demographic is for students anyway, we will set an age limit of 18+ in order to use the site, to avoid exposure to harmful content for children.

Steps taken to mitigate risks found:

As users submit challenges and images, it is imperative to implement a content moderation system to review and filter user-generated content, preventing the submission of inappropriate, offensive, or unsafe content.

To support this, anti-bullying measures and a robust reporting mechanism are critical, ensuring the platform remains enjoyable and safe for all users. Our chosen approach to content moderation relies on community reporting; an image flagged by five or more users as inappropriate should be automatically reviewed and, if found violating, removed from the platform.

Further, the site should be 18+ to protect children from any harmful content.

Consider protecting freedom of expression:

With goals of fostering community, it is important that our app recognises people's rights to express themselves, within reason. The core mission is to encourage students to explore the world in a fun and interactive way, so too strict moderation could have a negative impact.

Secure and encrypted data

Prioritising user privacy involves implementing encryption and secure authentication when storing our data, especially with location-based and image-sharing features.

1.8 Technical Considerations Analysis

1.8.1 Usability

Since SnapMatch is a web app, it is vital that it caters for people with diverse backgrounds, because anyone can access it. Users interact with the app through the UI, so it is important that it is designed well. Some key things to address:

Gestalt Principles

An important part of GUI design is the use of spacing and sectioning to ensure users understand what is being presented to them.

Easy-to-use navigation system

This system should help users easily access features like challenges, submissions, and matches. The design principle of guessability is important, since we want users to be able to get started straight away; it should be intuitive what a user is meant to do.

Feedback mechanisms

Clear and prompt notifications of the success or failure of user actions, coupled with instructive guidance for initial app navigation, contribute to a user-friendly and rewarding experience.

1.8.2 Image Recognition

Image recognition is the main deciding factor in whether two photos are a match. There are two approaches we could take: human verification, or using an AI model.

Human verification would involve users themselves confirming that the response image is of the same image taken. This could result in inconsistent verification, with potential malicious dis-allowing of matches.

Using an AI model would be better, since the rate of matches would be consistent. However, the downside of an AI model is that it is only as good as the dataset on which it was trained, and so it could be hard to find a model that is open-source and has been trained to the level needed for unique objects to be identified. Potential further training may be required. In addition, the AI will have to be adjusted to only match the specific object a user intended to be matched, rather than a random object in the background.

1.8.3 Scalability

Another critical consideration is scalability; the project's backend architecture and database system should accommodate future increases in user interactions and data without requiring a complete restructuring—simply adding more servers should suffice.

2 Requirements & Critical Analysis

2.1 Ideas for requirements

In addition to the previous research, we needed a better understanding of the stakeholders' perspectives, so we attended a meeting held at UNiDAYS headquarters. During this session, stakeholders shared their insights on the project brief and how they deliberately wanted to leave requirements open for the team to decide. This meeting was crucial for aligning our understanding with the stakeholders' expectations, who expressed a desire for the team to explore and define the application's direction creatively.

To facilitate this process, we utilised a Directed Acyclic Graph styled "vision board", found in appendix A, centrally featuring the project's name with multiple bubbles radiating outward. Each bubble represented potential functionalities of the application, connected directly to the centre, symbolising their core relationship to the overall project. From these primary nodes, further subdivisions branched out, detailing specific features for each functionality. This graphical representation allowed for an expansive brainstorming session, where every conceivable idea was mapped out.

2.1.1 Impact of the Vision Board

The vision board proved instrumental in several ways:

Guidance: It served as a visual and conceptual guide throughout the project, helping maintain focus on the core functionalities while exploring potential enhancements.

Idea Generation: The format encouraged unrestrained creative input, leading to a rich array of ideas that might not have surfaced through more conventional discussion formats.

Stakeholder Interaction: It facilitated a deeper engagement with stakeholders, making the session more interactive and collaborative, which was highly valued by all participants.

2.1.2 Evaluation of the Vision Board

The vision board was a powerful tool for generating ideas, enhancing stakeholder participation and satisfaction by visually mapping out the thought process, and encouraging comprehensive exploration of possible features.

However, this process was not used again throughout our project duration, since the broad scope captured on the board sometimes diverted attention from prioritising feasible features within the project's constraints of time and resources. Further, it required subsequent sessions to refine the features down to actionable items, but we cannot deny it's benefits as a starting point.

2.2 Requirements from Key Considerations Research, Market Analysis & Vision Board

2.2.1 User Engagement

Challenges are not just to attract users initially, but to set-up the product for long term involvement.

Geo-tagging challenges

Users could tag their challenges with the location it was taken, encouraging users to engage with the app through outdoor exploration.

Gamification

A leaderboard could highlight the top users of the application, awarding points to those who upload and respond to the most challenges, creating competition and keeping users coming back.

Themed challenges

Having themes for the challenges would keep the app exciting. These would be tied to specific events, seasons, or university milestones. Furthermore, they could be time sensitive, making achieving the challenge rewarding.

Notification system

A notification system would be useful for informing users if other users have successfully matched their challenge, or if they themselves have made a successful response. Moreover, these notifications would serve as reminders to interact with the software.

'Friends'

SnapMatch could aim to be a social media platform. Users could befriend other users, letting them upload and respond to private challenges from their friends.

2.2.2 Accessibility

It is important the software ensures inclusivity through a user-friendly experience, catering to the diverse backgrounds of students. The following features should bring this about:

Editing challenges

Users should be able to edit challenges to change the caption or delete them – this should wipe the image from the database completely, including responses, to comply with user ownership over their digital footprint.

Real time preview

We hope to implement a real time preview of the image a user wishes to capture before they take it, so they can confirm the quality of the photo before uploading.

Thumbnail for submitted challenges

Thumbnails will be shown in users' feeds as visual representations of challenges, letting users easily preview challenges they want to respond to without having to see the full challenge detail.

2.2.3 Safety vs Fun

Content moderation

A content moderation system is important for filtering inappropriate content and since our software involves user uploaded content, the ability to report challenges is a priority. This would result in a

five-strike system, banning users that have uploaded too much reported content. Restricting accounts to 18+ is also important.

Secure and encrypted data

Prioritising user privacy involves implementing encryption and secure authentication when storing our data, especially with location-based and image-sharing features.

2.2.4 Usability

We hope to implement an easy-to-use navigation system, offering quick access to challenges and responses, maintaining a visually cohesive and responsive design for a great user experience.

User feedback

Users will need feedback on how to handle errors, and a verification of successful uploads, to keep them informed.

2.2.5 Image Recognition

The image recognition system should be able to identify different objects relatively accurately and return with logical variables true or false. We will detail this requirement further, once a method has been picked out of human verification vs an AI model.

2.2.6 Reliability

When experiencing high user traffic, the system must automatically balance the load across servers to maintain stable operation. This involves assessing the performance and efficiency of the system and selecting a suitable architectural framework to support these requirements.

Error Handling System

Handling errors could be a potential issue, so the platform needs a system in place to do this, in the case that there are problems when submitting/displaying/processing images.

2.2.7 Maintainability

The project must maintain a logical and maintainable structure, enabling future developers to easily comprehend the organization of the project and the relationships among its various components.

2.2.8 Scalability

The number of users on the app may grow, meaning that the product must be able to handle an increase in data and user interactions. There must be backend architecture that can handle growth as well as an efficient database system that can handle storing images. The image processing system must be able to handle requests, simulating a contiguous streamlined process. As the user base grows ensuring security may be harder, so allowing access controls, authentication and other defence mechanisms will limit the amount of manual human security handling.

2.3 Prioritisation of requirements and scope reduction

In the initial phase of the SnapMatch project, our team conducted a comprehensive skills assessment to ensure that the project's requirements aligned with our capabilities and constraints.

From this, we prioritised the development of a Minimum Viable Product, choosing requirements that were ambitious yet achievable. For example, no team members had experience with image processing models, so the MVP accepted human-verified matches. Previously mentioned requirements that didn’t make it into the MVP became ‘could have’ requirements, added to the product backlog to be considered once the MVP was achieved.

Outlined next is the process used for choosing MVP requirements.

2.3.1 Systematic Requirement Identification

To systematically identify the project requirements, we utilised a combination of tools and methods:

Vision Board: This tool was pivotal during our brainstorming sessions at UNiDAYS headquarters, allowing us to visualize and explore a broad range of potential functionalities creatively.

Skills and Constraints Analysis: Each feature on the vision board was evaluated against our team's skill levels, estimated time requirements, and the difficulty of implementation. This analysis helped us to focus on feasible features while setting realistic timelines.

Agile Methodology: Changes to requirements were accommodated for by adopting an Agile workflow. This can be seen in effect throughout the Implementation section of this report, and is described in more detail in Project Management.

2.3.2 Incorporating Feedback and Iterative Development

Throughout the project, we engaged in continuous feedback loops with stakeholders and mentors. This iterative process not only refined our understanding of what was technically and practically feasible but also ensured that the product evolved in line with user needs and team growth in skills.

2.3.3 Conclusion

The systematic approach to requirement identification, grounded in a realistic assessment of our capabilities and a flexible project management style, was critical in creating our MVP, outlined below. It also set a strong foundation for future enhancements, demonstrating the effectiveness of combining creative brainstorming with rigorous analytical methods.

2.4 Minimum Viable Product

1. Create challenge
    1.1. Take a picture in the software
    1.2. Save picture in database
    1.3. Display captured picture
    1.4. Add caption
    1.5. Upload as challenge
2. Respond to challenge
    2.1. Follow same ‘create challenge' procedure 1.1 – 1.4
    2.2. Upload as response
3. Match images
    3.1. User that uploads challenge approves response as match
4. View challenges
    4.1. Display challenges in public feed
    4.2. Individual challenge page shows caption and user responses
    4.3. Respond to challenge
    4.4. Report challenge
5. Create user account
    5.1. Unique username
6. Feed of user’s uploaded challenges
    6.1. View challenge
    6.2. Approve responses as matches
    6.3. Edit or delete challenge
7. Feed of user’s responses to other user's challenges
    7.1. View challenge and response

7.2. Edit or delete response
7.3. View status of response: 'submitted', 'it's a match', 'it's not a match'

8. Safety
   8.1. Manual report button
   8.2. Take image down after 5 reports automatically

2.5 More detail into MVP requirements

2.5.1 User profiles

We need the ability to uniquely identify users, so it is clear who has uploaded what challenges, and so responses were recognised as responses to **other** users. To do this, we need user profiles with unique usernames, and a login system.

2.5.2 Challenges Feed

A user should be able to see a feed of uploaded challenges from other users, so they can find challenges to respond to. A user needs a personal challenge feed too, so they can view their challenges and corresponding responses.

2.5.3 Capturing images

Developing an application that users can use to capture images within the application requires integration with the user's camera hardware, meaning the user must first allow access to the camera.
Image quality is also important, with the application allowing users to capture photos in high resolution. This seems like a simple task but previous applications such as Snapchat have struggled to use the cameras API on the android platform, instead screenshotting the photo that the user captures, resulting in a worse quality photo, so it may be harder than it seems to integrate this across platforms.

2.5.4 Processing image – Challenge submission

Since users are supplying challenges, there must be a step-by-step process they can follow to capture an image, create a challenge and submit it.
Given the potential volume of users submitting challenges, storing the images correctly is important, ensuring the application remains responsive in displaying the image from the stored photo. In addition, when storing the image, the metadata must also be stored, such as geolocation, timestamp and user information, necessary for challenges based on location.

2.5.5 Issuing challenges

Since the brief had an emphasis on keeping people engaged in the application, a user interface for users to create images is important to fulfil this. Designing a fun but also easy to use interface allowing users to add a description to the challenge image and any other criteria that may be potentially linked to the challenge, allowing the user to customize the challenge to their liking. On top of the user adding the user adding their own description of the challenge, if challenges involve location, then integration geolocation services to set location accurately.

2.5.6 Accepting a challenge

Must create a straightforward way for users to be able to find a challenge; with a simple user interface and complete a challenge that has been issued which will streamline simple steps to capture their own image and submit the image for the backend to verify a match. Submission verification is also needed for the user who has accepted the challenge to see if their submission met the criteria of the issued challenge. Handling errors is also relevant.

2.5.7 Processing image – Challenge matching

Images submitted as challenge responses needs to be the same as the users' photo who issued the challenge. This can either be done with the user who has created the challenge being sent the image verifying the image is a match or an algorithm. If it is the first suggestion, the application must have a straightforward way for the images to be sent to the person who has issued the challenged, a way for them to say if it matches and the result is sent back to the other user who has accepted the challenge. If it is the latter, an algorithm must be found or created, able to compare the features extracted from submitted images, determining if the item or location is similar. It must be able to identify key patterns, shapes, and characteristics in an image, along with handling errors.

2.6 Research for actioning requirements

2.6.1 Research into Image Recognition

The requirements specified image matching to accept a response to a challenge. This could be done visually by users, but this would result in dependence on factors out of the app's control, such as how a user defines a match (and whether that stays consistent), the frequency at which a user is verifying matches, and the ability for users to verify large volumes of responses.

A better solution would have an AI doing this work instead. An algorithm would be able to process large number of datasets of images with efficiency, necessary for the potential large audience of the project. The inconsistency of defining a match by human judgement can be avoided through an AI providing impartiality.

2.6.2 Potential Image Matching AIs

Google Cloud Vision API

UNiDAYS recommended the Google Cloud Vision API, a service that integrates vision detection features into applications. This API facilitates image content analysis through pre-trained models, leveraging a combination of advanced machine learning techniques, including deep learning neural networks trained on vast datasets, enabling the recognition and classification of objects, face detection and text extraction.

Ease of Integration: The API is designed to be easily integrated into existing systems with minimal setup, which is crucial for developers looking to enhance applications without extensive modifications.

Documentation Quality: Comprehensive documentation is provided that supports developers through detailed guides, examples, and API references, for a smooth implementation process.

RAM (Recognise Anything Model)

RAM is an open-source package that can be directly modified and integrated seamlessly into any Python code. It has high accuracy and detailed documentation, so is a great option for developers who aren't so used to image matching models.

2.6.3 Image Matching Model Choice & Justification

Both models promised high accuracy, detailed documentation and ease of integration, so the deciding factor came down to cost. RAM was opted for initially, since it was free and could be modified directly, allowing for a tailored integration into SnapMatch. Further, it had high accuracy, so the trade-off compared to not using Google was not so bad.

2.7 High-level Design- Implementation to MVP

2.7.1 Analysis

The high-level architecture of SnapMatch, a Service-Oriented Architecture, shown in this diagram in appendix A was analysed with the goal of creating an engaging, secure, and scalable application.

1. **Frontend Layer:** Built with React, chosen for its robust ecosystem and efficient handling of dynamic user interfaces. React's virtual DOM and reactive data flow ensure optimal rendering performance, crucial for the interactive elements of SnapMatch.
2. **Backend Layer:** Powered by Flask, this layer is designed for easy scalability and flexibility. Flask's lightweight and modular design makes it ideal for applications like SnapMatch, where rapid feature development and iteration are required.
3. **Database Layer:** Utilizes AWS RDS, providing a scalable and reliable data storage solution that supports rapid scaling needs and complex query processing. RDS automates tedious tasks such as hardware provisioning, database setup, patching, and backups.

2.7.2 Discussion

The choice of a three-tier architecture ensures separation of concerns, simplifying **maintenance** and, providing **agility** in development and deployment, through the ability for individual layers to be updated or scaled independently. Moreover, the VPC and Internet Gateway create a **secure network channel**, essential for protecting sensitive user data and ensuring application integrity.

2.7.3 Justification

Compared to **monolithic architectures**, which house all application components in a single, indivisible unit, the chosen three-tier architecture offers greater flexibility and easier scalability. While monolithic architectures are simpler to deploy initially, they become increasingly complex and rigid as an application grows. In contrast, our three-tier approach allows for scaling individual components as needed without impacting the entire system.

Moreover, alternative architectures such as **microservices**, dividing an application into smaller, independently deployable services, were considered; however, given our project's scope and the team's familiarity with the chosen technologies, the three-tier model was deemed most appropriate. Microservices would require more overhead in terms of service coordination and could lead to increased complexity in managing multiple services, which our team structure is not optimised for.

Each technology layer was selected not only for its individual benefits but also for how it integrates into the overall system architecture, ensuring a seamless, secure user experience. This architecture is strategically designed to handle increasing user activity and data volumes efficiently, ensuring sustained performance as SnapMatch grows.

2.8 Technology Stack

2.8.1 Front-end

Front-end Programming Language Analysis

JavaScript was chosen for its client-side execution, which facilitates real-time interactivity essential for our application's image processing requirements. This allows users to see changes instantly without server delays, crucial for a responsive UI during image uploads and processing. Compared to other languages that require server-side processing, JavaScript provides a more seamless user experience by minimizing latency.

Front-end Framework Analysis

React was selected due to its extensive ecosystem and the team's prior experience, which expedited development. Its component-based architecture makes it ideal for building dynamic web applications that need to respond instantly to user interactions. Unlike frameworks like **Angular**, which offers a more comprehensive suite of solutions for application development but can be more complex to use, or **Vue.js**, known for its simplicity and gradual learning curve yet less mature ecosystem, React's simplicity and focused design make it particularly well-suited for our project's needs for rapid development and future scalability.

Front-end Test Framework Analysis

Cypress was chosen for front-end testing due to its direct execution within the browser and real-time reloading, which significantly boosts testing efficiency. In contrast, **Selenium** requires a more complex setup and tends to operate slower as it manages a separate browser instance, complicating rapid test iterations. Additionally, while **Jest** is a robust framework for unit testing, it lacked compatibility with some team members' systems, which led to challenges in implementing it effectively for our project needs. Cypress's ability to handle both unit and end-to-end testing for dynamic applications made it the ideal choice for SnapMatch, allowing for comprehensive testing coverage and easier debugging.

2.8.2 Backend

Backend Programming Language Analysis

Python's extensive library support, especially OpenCV for image processing, made it the optimal choice for our backend development. Its simplicity and readability streamline the development process, unlike **Java**, which, while featuring extensive enterprise support, tends to require more verbose code and has less specialized libraries for quick prototyping in image processing. **C++**, offering high-performance capabilities and fine control over system resources, can be too complex for rapid development cycles due to its intricate syntax and manual memory management.

Backend Framework Analysis

Flask provides the flexibility needed for our web application, supporting rapid development and easy integration of extensions such as database integration and authentication. Its lightweight nature and ability to scale with minimal setup make it preferable over more heavyweight frameworks like **Django**, which, while robust, typically requires more setup and configuration.

Backend Test Framework Analysis

Pytest was chosen over other frameworks like **Unittest** or **Nose** due to its simplicity and powerful fixtures that simplify complex test scenarios. Unittest, Python’s built-in testing framework, offers a more traditional style of testing which can be less flexible and more verbose compared to pytest. Nose, on the other hand, extends unittest to make testing easier, but it lacks the more modern features and plugin ecosystem that pytest offers, making pytest more suitable for both small unit tests and complex functional testing.

2.8.3 Database

Database Server Analysis

Amazon RDS was selected for its managed service capabilities, which reduce the overhead of routine database tasks. This allows our team to focus more on application development rather than database management. RDS provides scalability and reliability more effectively compared to managing our **own database servers**, which would require extensive setup and maintenance.

2.8.4 Hardware

Deployment Server Analysis

Amazon Web Services (AWS) was the platform of choice due to its comprehensive service offerings that integrate seamlessly with each other, such as EC2 and RDS, providing a cohesive environment for both front-end and back-end hosting. Its scalability and reliability are superior to alternatives like **Microsoft Azure** or **Google Cloud** for our specific needs due to better documentation, more mature community support, and cost-effectiveness.

2.9 Analysis of Established Software Engineering Methods

The application of established Software Engineering methods played a crucial role in shaping the development process and architecture of SnapMatch, detailed further in the Software Manual.

2.9.1 Entity-Relationship Diagram (ERD)

ERD graphically illustrates the structure of a database, which was critical for visualizing the database structure, promoting a unified understanding among team members. This visualization not only accelerated the database design but also prevented potential misalignments between data relationships and application logic.

2.9.2 Service-Oriented Architecture (SOA)

Adopting SOA allowed our components to be developed and deployed independently, which significantly streamlined the integration process. This architecture facilitated scaling specific services without the need to overhaul the entire system, thus enhancing maintainability and future-proofing the application against evolving requirements.

2.9.3 Model-View-Controller (MVC) Pattern

The MVC pattern is an architectural design that separates application data, the user interface, and control logic into three distinct components. The separation of concerns inherent in the MVC pattern enabled concurrent development streams across the user interface, data management, and control logic. This not only improved the efficiency of development cycles but also simplified troubleshooting and updates, as changes in one module minimally impacted others.

2.9.4 Singleton Pattern

The Singleton pattern ensures that a class has only one instance and provides a global point of access to it. Utilizing the Singleton pattern for managing database connections optimized resource utilization and ensured consistency across data transactions. This approach prevented the common pitfalls of redundant connections and data conflicts, which are crucial in maintaining the integrity of user sessions and interactions.

2.9.5 GitLab and GitFlow

We adopted the GitFlow workflow for version control, a structured branching model for Git, clearly distinguishing development, release, and maintenance activities. For new features, we would make a new branch for testing, then create a merge request where another member reviewed our changes, and addressed merge conflicts when merging it into the dev branch. This structured approach enhanced collaboration, allowing for efficient feature development.

3 Initial software implementation

3.1 Sprint 1

First, we set up the development environment with our chosen software libraries and tools, and configured our GitLab repository. We conducted tests simulating user behaviour to ensure our databases efficacy.

3.2 Sprint 2

In our next sprint, we focused on allowing a user to upload a challenge. We found a plugin for camera access and set up photo-capturing functionality. The photo would then be inserted into the database and a caption could be added before being uploaded, seen here in Appendix B. We implemented Dropbox API functionality, where the photo is uploaded to the cloud and an image URL is generated. After fetching the URL, this and the caption is added to the database.

We encountered challenges when attempting deployment to mobile devices, with attempts made at port tunnelling failing, so we put this off to a later date.

3.3 Sprint 3

Our last sprint of the semester focused on UI design and AI integration, as detailed next.

3.3.1 AI Integration

We deleted useless functions in the RAM package to improve its efficiency, and implemented into our project with ease, due to it being an open-source Python package. Now, when uploading a challenge, the images were processed by the AI model, with an output array of identified objects being displayed in the frontend, as seen in Appendix B here. We added the ability to choose the object users wanted people to match with.

3.3.2 AI Model Size

Despite its **high accuracy**, RAM posed significant challenges for practical deployment. The model required approximately 5GB of storage for its AI components, making it **impractical for users to download** and use, especially on mobile devices. Furthermore, since RAM ran locally, its **performance** heavily depended on the user's hardware, which varied significantly and often did not meet the necessary standards to run such an intensive AI model efficiently. Described later on in the report, to combat this, we decided to research deploying the AI on the cloud, using cloud computing to process the images.

3.3.3 UI Design

We designed UI prototypes for both desktop and mobile layouts (in Appendix A), but due to development on laptop devices, we focused on implementing the desktop design first. Adhering to our requirement of cross-platform responsive design, we evolved the UI to include a taskbar for transition between screens, as seen here in appendix B, with colour choice being a work in progress.

4 Software Implementation & Changes to Requirements

The second semester involved some requirements changes, detailed within each sprint.

4.1 Sprint 4

The team were focused on solving previous issues and making progress with app development.

4.1.1 Cloud AI & Change to ClarifAI

As previously mentioned, our model RAM was too large. We researched more suitable models, finding ClarifAI's API. It provided a range of pre-built models for object detection and image classification, trained on large datasets of labelled images to learn patterns and features. ClarifAI emerged as a suitable replacement due to its **lightweight nature** and the ability to operate entirely via cloud-based API calls, which eliminated the need for local deployment and extensive storage.

Although ClarifAI did not match RAM in terms of recognition accuracy, its ease of integration, minimal deployment costs, and **rapid response** times made it a more viable option. This decision allowed us to maintain user-friendly application performance and manage resource utilization more **effectively**. The trade-off in **accuracy** was considered acceptable in light of the significant benefits in application scalability and user accessibility.

4.1.2 User Accounts

We recognised that to respond to a challenge, a user would need their own account. Work started on a dedicated front-end login page, along with database additions to store session IDs.

4.2 Sprint 5

This sprint had a renewed focus on developing for mobile devices, pushing for web deployment.

4.2.1 Deployment

For the deployment of the front-end, we utilised an AWS EC2 instance. The project was cloned from our Git repository onto the server, followed by the acquisition and binding of the domain name snapmatch.top to the server's elastic IP. This binding ensured that the domain remained linked even after server restarts. We then built our React project and set up NGINX to listen for specific user requests, which were directed to the compiled static resources of the project. Additionally, we configured free SSL certificates using CertBot, enabling the project to support HTTPS protocols.

The backend is deployed on a separate AWS EC2 instance, using NGINX to listen for requests from the front end. Concurrently, we utilised Gunicorn to handle requests received from NGINX by passing them to the running backend Python code. This setup enhanced the robustness of our server responses and ensured seamless communication between the front and backend environments in production.

4.2.2 UI Prototypes

Accommodating the focus on mobile design principles, some changes needed to be made, such as moving the taskbar to the bottom, due to mobile users using their thumbs for app navigation.

Updated UI designs were produced using Figma, a useful online wireframing tool for quick lo-fi prototyping.

4.2.3 Google Login

To simplify the login process by utilising existing accounts, we included the ability to login with a Google account, enhancing security compared to traditional email and password setups which require additional layers of security to match Google's standards. However, recognising not everyone has a Google account, we kept the standard login too.

4.3 Sprint 6

This sprint began with a tidy-up of our Git repository, deleting stale branches providing confusion.

4.3.1 Front-end Testing

By this point, one member had been working tirelessly trying to use Jest for front-end testing, but to no avail. Support was given at our second UNiDAYS workshop, but it proved fruitless, so the decision was made to swap to Cypress, a similar framework that could be easily integrated into our project.

4.3.2 Leaderboard & Point Allocation

To meet our user engagement criteria, we implemented a leaderboard, adjusting the user database to store points for responding to challenges. Designating points was complicated, since we realised the leaderboard would need to be accessible to both frequent users and newcomers. We suggested a daily and all-time leaderboard to address this, but due to time constraints, only implemented all-time.

We settled on time-based points allocation, with each responder gradually getting less points than the first. Testing this, users would run out of challenges to respond to, so points for uploading challenges was also implemented, incentivising users to fully utilise the app.

4.3.3 Verifying a Response

After deciding that for two pictures to be a match, the response object array had to include the intended object from the challenge. This led to the eradication of the 'Other' category, used previously to provide a user-inputted object description. There would have been no way to guarantee that the AI would identify an object as the same as what the user inputted. Unfortunately, this meant more reliance on the AI's set categories for object identification but was a price worth paying for a fully automated verification system.

4.4 Sprint 7

4.4.1 Implementing Responses

The response endpoint was implemented, giving the ability to take a photo response. True was returned for a match, and false for a non-match, prompting the user to retry.

4.4.2 Login Features

Password confirmation was added, along with a logout button, ending the session for the user.

4.4.3 Changing UI

The changes to the UI were finalised, resulting in a design that was responsive to both desktop and mobile layouts. We used browser developer tools to test for mobile, which was not a true test of how the app would look, so we redeployed the front-end to allow for proper mobile testing.

4.5 Final Additions

Over Easter, we completed our product, adding a page for viewing user responses, with ease, since it repeated the component of the challenges list.

Web deployment meant testing on actual mobile devices could take place, leading to slight UI tweaks and bug fixes, ensuring the app stuck to the goal of a responsive design.

5 Project Management

5.1 Software Management Approach

To tackle this project, we used the software management method Scrum, a widely used agile development framework. This involved two week-long 'sprints', each focussed on delivering a distinct feature, ending in a demonstration to UNiDAYS of work achieved in the timeframe.

We had the pleasure of attending a UNiDAYS-run workshop about Agile, which was useful since we got to see the approach in action, performed by a real software development team. They showed us how they used the project management tool Jiro to create sprint tickets, and guided us through our first retrospective meeting, helping us gain an outside perspective on how our team was functioning.

5.2 Sprint Contents & Planning

Eleanor, as team administrator, set up the project management tool MS Planner to keep track of the workload each sprint. User stories produced during requirements analysis were split into 'epics' and then smaller 'tickets'. Epics describe a broader topic i.e. 'upload a challenge', and tickets describe smaller tasks within that, i.e. allow camera access.

5.2.1 Tickets

As influenced by UNiDAYS, we came up with a consistent ticket template, as seen in our ticket in Appendix A. A short summary was written for the title, along with a detailed description of what was required, so anyone could understand what the ticket meant.

An essential feature of MS Planner was the ability to add ticket labels, useful for quick understanding. These were used to indicate what epic a ticket was part of, whether it was front-end/back-end or a would/could/should have item, for example.

5.2.2 Kanban Board

Using MS Planner, we designed our Kanban board with six columns. The 'Product Backlog' column stored tickets that were yet to be part of the sprint, and the two 'To Do' columns were used as sprint backlogs, storing tickets (split into front end and back end) that we wanted to be achieved during the sprint, but were yet to be assigned. 'In Progress' was for assigned tickets and 'Review' was for completed tickets that were moved into 'Done' once verified they were finished.

This was useful for visualising the amount of work assigned, viewing progress made throughout the sprint and seeing what was left to complete. An example of the board at the start of a sprint can be found in Appendix A.

5.3 Sprint Execution

Initially, our meetings were scheduled according to the module's prescribed times. However, as we transitioned to the Agile sprint methodology, we increased meeting frequency to align with development needs. We created a timetable that accommodated all team members' schedules.

5.3.1 Meeting Notes

We used OneNote to take meeting minutes, differentiating note-taking approach per meeting type. After interim feedback, these notes were amended to show a clear agenda and assigned actions for members to take after the meeting. Also, we were encouraged to include a ground rules section by our academic supervisor, documenting team expectations for punctuality and missed meetings etc.

5.3.2 Meeting Timetabling

Given COMP2002 is not a full-time module, we deviated from the SCRUM standard of daily meetings, instead spreading them throughout the week, as outlined below. We still maintained a consistent communication flow, whilst accommodating the part-time nature of the module.

Sprint planning meetings

At the start of a sprint, we held an hour-long planning session, where the sprint focus would be picked based off the product backlog. Tickets related to the focus were moved from the backlog into the 'To Do' columns.

To choose tickets, we assigned ‘story points’ to each one by asking everyone how difficult they thought it would be from the numbers 1, 3, 5, 8, 13. A difference in numbers resulted in a discussion on why they were chosen, with the process being repeated until everyone said the same number. This was a verbal process, so that answers were quick and based off intuition. Tickets with high story points were recognised as too difficult and broken into smaller parts. Story points were added to the ticket description as seen here.

Tickets were then equally distributed among team members, with equal numbers of story points, and moved to the 'In Progress' column for the sprint.

Stand-ups

In the first semester, we had three ‘stand-ups’: 15-minute-long online meetings where we would discuss our current progress, what was holding us back and what we were going to do next. This was useful to facilitate collaboration and keep team members up to date. An hour long in-person meeting was sandwiched between these standups, useful for assigning extra tickets if work was completed quicker than expected.

Meeting frequency was reduced to two standups and one in-person mid-sprint meeting in second semester, after the team discussed they were overwhelmed with the original number of meetings.

Demonstrations

At the closing sprint meeting, everything achieved was demonstrated to our stakeholders at UNiDAYS, providing an opportunity for critical feedback, technical guidance and changing requirements. Any technical queries we had encountered were also addressed, offering us a rich learning experience.

Retrospective Meetings

The sprint would end with a retrospective session, involving discussions of what went well and what could have gone better. Shoutouts were given for commendable work, fostering a culture of recognition and continuous improvement. An example can be found here.

During the first semester, this was held at the end of our demonstration meeting with UNiDAYS, where they would also comment on our strengths and weaknesses. Due to timetabling clashes in the second semester, we could no longer do this, so instead moved these sessions to the start of our next sprint planning session. Luckily, this proved beneficial, since any suggested improvements could be actioned straight away when planning the next sprint.

At the end of the retrospective, MS Planner was refreshed, ready for the next sprint.

Supervisor Meetings

Throughout the project, meetings were scheduled with our academic supervisor, to keep her updated with progress made.

5.4 Team Organisation

5.4.1 Communication

Effective communication was pivotal in our project management strategy. For internal team communications, we primarily utilised Microsoft Teams and WhatsApp. Teams was particularly valuable for file sharing; we initially tried using WhatsApp for this but found it inefficient due to difficulties in navigating historical files.

For communication with UNiDAYS, we opted for Discord, providing quick and effective sponsor interactions. This setup ensured that all communication channels were optimised for different aspects of our project collaboration.

5.4.2 Use of Team Skills

Junfeng, as team leader, initially divided the group into frontend, backend, and AI image processing teams, as seen in graph A6 in Appendix A.

However, this was found to be not so effective. To foster comprehensive project understanding, we implemented a **cross-training** approach, allowing team members to rotate roles across different sprints. This dynamic allocation of responsibilities ensured that no one was confined to a single aspect of the project, boosting engagement, and preventing burnout.

Team members were enabled to develop a broad skill set, as they transitioned from one technical area to another—such as moving from backend development in one sprint to frontend tasks in the next. This flexibility was instrumental in addressing evolving requirements, and distributing workload efficiently, whilst also deepening members' understanding of the project architecture.

5.4.3 Work Ethic & Attendance

Our team demonstrated a strong work ethic, characterised by strict adherence to our meeting schedule and punctuality, set out in the ground rules. We kept track of meeting attendance, with members alerting of absences in the WhatsApp chat, and checking the minutes to catch up. Unannounced absences were rare, reflecting high commitment levels.

Deadlines were strict and members consistently completed tasks on time. To manage heavy workloads, tasks perceived as too demanding were assigned to multiple team members, helping maintain steady progress but also fostering a supportive team environment.

Autonomy was encouraged, giving members significant leeway in managing their assigned tasks. For instance, those responsible for project deployment chose their own deployment tools.

Through these practices, our team not only adhered to software engineering standards, but also cultivated a collaborative, and proactive work culture, critical for achieving our program goals.

5.5 Approach Analysis

This structured approach brought numerous benefits.

Concentrating on a single feature per sprint ensured that attention and resources were dedicated, leading to higher-quality outcomes.

Weekly check-ins allowed for continuous progress checks, ensuring that adjustments needed could be caught early, so development remained on track.

Collaborative planning empowered team members to collectively decide the difficulty of a task, to make sure no-one ended up with too heavy of a workload.

Regular stakeholder (UNiDAYS) engagement made sure that requirement changes could be recognized quickly and actioned.

6 Reflection

As a team, we made it through the project successfully, meeting the MVP we created based off the brief given by our stakeholders. In this section, we will outline our main achievements, but also address weaknesses of the project, with ways to improve for next time.

6.1 Technical Reflection

6.1.1 Methods and Implementation

SnapMatch employs a layered architectural approach, integrating modern technologies to ensure a robust and responsive application. The three-tier architecture comprises:

Frontend

Utilizing React to deliver dynamic user interfaces that respond to user interactions seamlessly.

Backend

Powered by Flask, this layer handles the business logic and data handling, providing a lightweight yet powerful framework for rapid API development.

Database

Leveraging AWS RDS for reliable and scalable data storage, which supports the application's need for high availability and complex data management.

6.1.2 Technology Choices and Justification

React was chosen for the frontend due to its efficiency in updating and rendering components, which is crucial for the real-time interactivity required in SnapMatch.

Flask was selected for the backend for its simplicity and flexibility, allowing for quick development and easy scaling.

AWS RDS was used for data management to take advantage of its scalability and robust management features, which simplify operations such as patching, backups, and scalability.

6.1.3 Difficulties Encountered

Integration Complexity

In the initial stages, our frontend and backend were configured to communicate via localhost ports, using a local SQLite3 database for development ease. However, this setup was not suitable for production. To address this, we transitioned to a Service-Oriented Architecture (SOA), where frontend and backend operations were deployed on separate AWS EC2 instances within a secured Virtual Private Cloud (VPC). Communication between these services was upgraded to HTTPS from HTTP, enhancing security with domain validation to ensure backend interactions were only permissible from authorized domains. The database was migrated to AWS RDS, utilizing MySQL with password authentication for secure and scalable interactions.

Performance Optimization

Early development faced challenges with managing the load between client-side and server-side processing, crucial for maintaining responsiveness as user activity increased. By implementing AWS Elastic Load Balancing within our VPC, we prepared our infrastructure to scale efficiently, accommodating user growth without compromising performance.

Security Measures

Initially, user passwords were stored in plaintext, posing a significant security risk. We enhanced our security protocols by adopting MD5 encryption for passwords, incorporating a salt to each password before hashing. This method, widely recognized for its security effectiveness, ensures that stored passwords are safeguarded against unauthorised access.

6.1.4 Future Directions

Enhanced Interactivity:

Further improvements in real-time data handling are planned, using WebSockets for a more synchronous interaction between the client and server.

Advanced Image Processing:

Expanding the capabilities of the image processing features using more sophisticated algorithms and machine learning models to improve match accuracy.

Scalability Improvements:

Continuous evaluation of application performance under load to refine and enhance scalability strategies, ensuring the infrastructure can handle increased traffic and data load.

6.2 Realisation reflection

6.2.1 Achievements

Praised by UNiDAYS, the completion of the Minimum Viable Product (MVP) for SnapMatch represented a significant milestone in our project. The platform has been successfully deployed and is live at https://snapmatch.top, where users are actively engaging with core functionalities. Users can create challenges, respond to them and view responses. In addition, we achieved a 'could have' requirement where users can participate in a leaderboard system that tracks and displays scores based on successful matches, a crucial gamification feature for sustained user engagement.

6.2.2 Strengths

The project's strengths lies in its scalable architecture and the use of modern web technologies which facilitate real-time interactions and a seamless user experience. The implementation of a Service-Oriented Architecture has ensured that each component can be independently scaled and maintained, which is crucial for handling increasing user loads.

6.2.3 Issues Encountered and Limitations

During the development process, we faced multiple challenges. One significant hurdle was the initial difficulty in ensuring efficient communication between the frontend and backend components. This was crucially addressed by adopting a secure and scalable cloud architecture, specifically through AWS, which offered a cost-effective solution with one year of free EC2 server access for verified students. This choice, while economically viable, presents a potential limitation: the sustainability of our project depends on continuous funding, as server costs will apply after the free year ends.

6.2.4 Weaknesses & future directions

Listed below are some weaknesses, detailing requirements that we would have liked to achieve with more time.

Privacy and Content Moderation

Currently, there is no functionality for users to report accounts using the platform inappropriately. We also lack a content filtering process to screen and block explicit or inappropriate uploads.

Any user can create an account, and all accounts are public, meaning children could be exposed to harmful content, violating Ofcom safety guidelines described earlier in the report. Although the likelihood of children finding and signing up for the app is slim, this is still a major danger, and would have been the next thing addressed with more time.

Social Connectivity

Our platform does not currently support the ability to connect with friends through private challenges. This limitation may deter users who do not wish to participate in public challenges.

Engagement Features

We envisioned introducing monthly themes to inspire engagement and creativity. However, this feature was not implemented, potentially affecting the platform's ability to retain users.

Leaderboard Functionality

The leaderboard only allows scores to increase, which disadvantages less frequent users and new-comers. Introducing a daily leaderboard that resets each morning could provide all users an equal opportunity to achieve top rankings, fostering fairer competition.

6.2.5 Lessons Learned

The development of SnapMatch taught us the importance of adaptability in software architecture. Initially planned structures needed continuous revision to align with evolving requirements and user feedback. This experience highlighted the necessity for an iterative approach to architectural changes, ensuring our system remained scalable and maintainable, adapting quickly to feedback.

6.3 Project management reflection

Having an adaptive project management approach was a main driver in completing our project to a good standard. We have outlined some strengths and weaknesses below.

6.3.1 Approach and Efficiency

After our first semester, we realised meetings had been too frequent, often lacking substance, leading to some members harbouring a feeling of inadequacy for not producing anything new. Reducing the number of meetings in our second semester was advantageous since it led to more meaningful updates on progress made, boosting team morale through respecting personal downtime, particularly over weekends.

Retrospectives held with UNiDAYS were useful for gaining an outsider’s perspective on the team’s functioning, allowing us to notice things we might have missed, such as at the first workshop, where it was discovered that non-native English team members were finding it hard to keep up with conversation.

Regular interaction with our stakeholders ensured the project stayed on track and allowed for suggested changes to be actioned quickly.

However, moving our retrospectives to the start of our sprint planning sessions was also efficient, allowing insights gained to directly influence the approach to the following sprint. Further, retrospectives were great for boosting morale, with recognition of people’s efforts and achievements.

6.3.2 Posterior Evaluation

Reflecting on our management tools, we realized the initial setup of our Kanban board with six columns was cumbersome, causing some tasks to be overlooked. Simplifying this by merging columns could have streamlined our workflow and prevented tasks from becoming stagnant. We learned that a less-is-more approach in tool setup can significantly enhance project fluidity.

6.3.3 Contingency Planning

Fortunately, the project did not experience significant changes in requirements from the stakeholders (UNiDAYS). However, adjustments were necessary due to time and resource constraints. One major shift was moving from RAM to ClarifAI for image recognition, driven by the need for a more resource-efficient solution. Additionally, we initially designed the UI for desktop environments but later had to modify it to ensure compatibility across all devices, including mobile. This transition involved redesigning the interface and implementing responsive components to accommodate various screen sizes. We addressed these changes proactively in our meetings, allowing for immediate action and integration into our development workflow.

6.3.4 Lessons Learned

From above reflection, we've learnt that adhering to an Agile methodology is invaluable, since it allows for flexibility needed to change meeting schedules and switch between development teams as required, maintaining team morale and productivity during busier periods. Further, retrospectives are a great team introspection tool for recognising members' concerns and achievements.

6.4 Actioning of Previously Identified Forward Measures

Our interim report identified several challenges and planned measures to address them. Here's a review of the actions taken and their outcomes:

6.4.1 AI Model Deployment

The chosen AI model of RAM for handling image processing was found to be too big and would've been problematic for deployment on users' devices. Instead, a cloud-based API solution was opted for, avoiding the need to deploy the model directly on devices. This adjustment significantly reduced the app's local resource requirements, enabling smoother operation without sacrificing functionality.

6.4.2 AI Model Limitations

The previous AI model sometimes failed to recognise objects or landmarks not present in its database, potentially reducing user engagement. Switching to ClarifAI did not solve this. While the initial intention was to fine-tune the open-source model, this was not addressed due to resource constraints. With additional resources, to solve this, a more sophisticated AI solution could be found, or a proprietary model could be tailored to the app's needs, focussing on unique objects.

6.4.3 Mobile Testing

In the interim stage of the project, development took place on desktop devices, but due to the nature of SnapMatch, ensuring the app functioned seamlessly on mobile devices was essential. To do this, we utilised browser developer tools to simulate mobile environments and conducted thorough testing with different screen dimensions. This approach allowed us to adjust the UI and UX effectively, ensuring compatibility and a responsive design across various devices.

6.4.4 Team Communication

Non-native English speakers initially struggled with the pace of discussions, so we enhanced our meeting minutes and ensured all members understood what had been spoken about, improving communication over time, with non-native speakers becoming more confident and active in talks.

6.4.5 Meeting Efficiency

Frequent meetings, especially near weekends, were found to be unproductive. This was fixed by a schedule restructure, resulting in meaningful sessions that respected personal downtime.

6.5 Summary and Conclusion

We are happy with our final product, proud to have achieved the requirements we set out in our MVP, along with a few additions along the way.

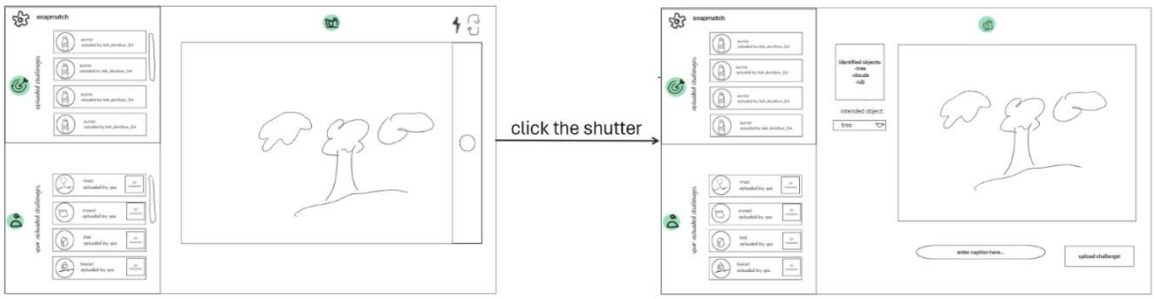
We have outlined key lessons learnt throughout this reflection section. Overall, the main lesson learnt is that Agile is great for project management, since it allows for flexibility, whilst maintaining a tangible goal for the end of each sprint. Team members being able to adapt as the project developed was crucial to achieving our goals.

Appendices

Appendix A – Graphs and Diagrams

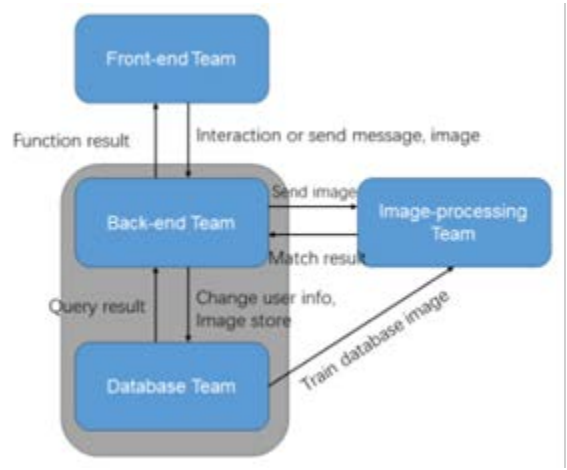
A1 – Directed Acyclic Graph Vision Board

A2 – UI Design (desktop)

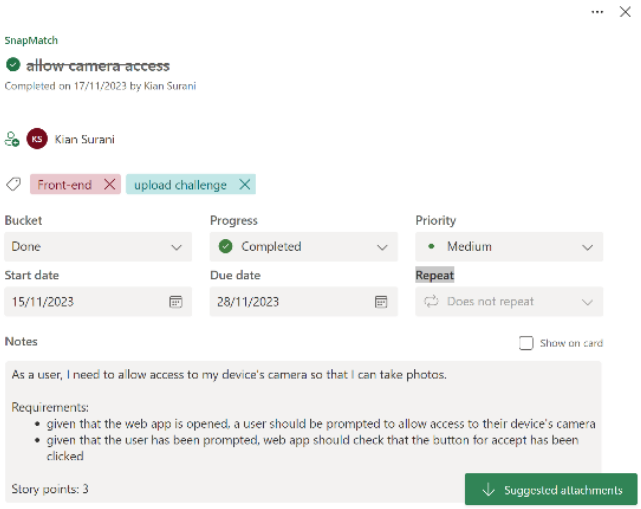


A3 – UI Design (mobile)

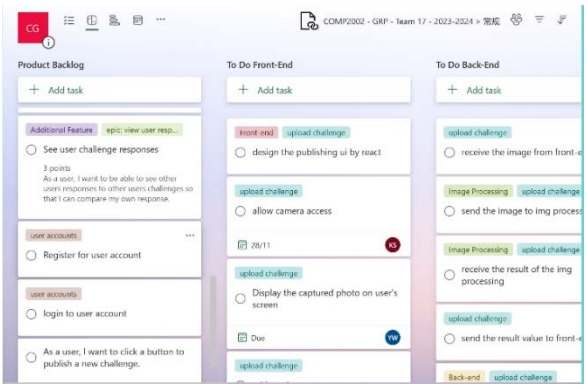
A4 – Team Structure



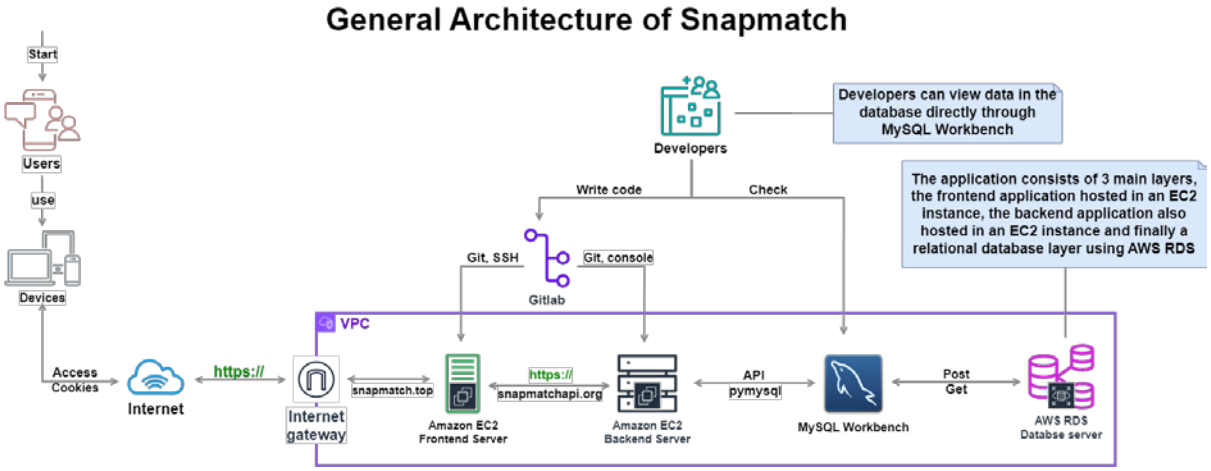
A5 – Example Ticket



A6 – Kanban board

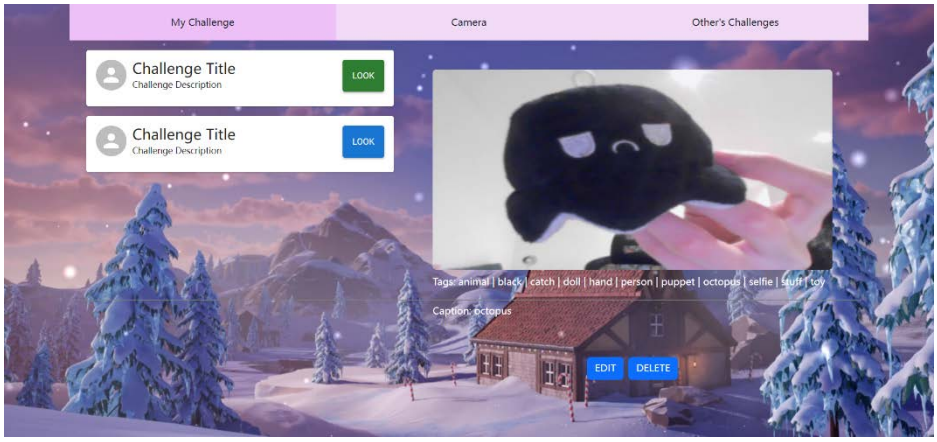


A7 – High level architecture

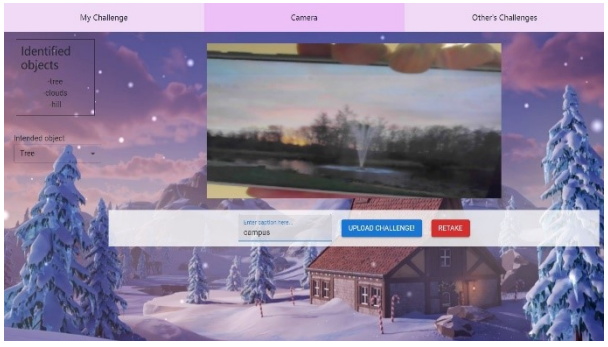


Appendix B – Software screenshots

B1 – View challenge screen



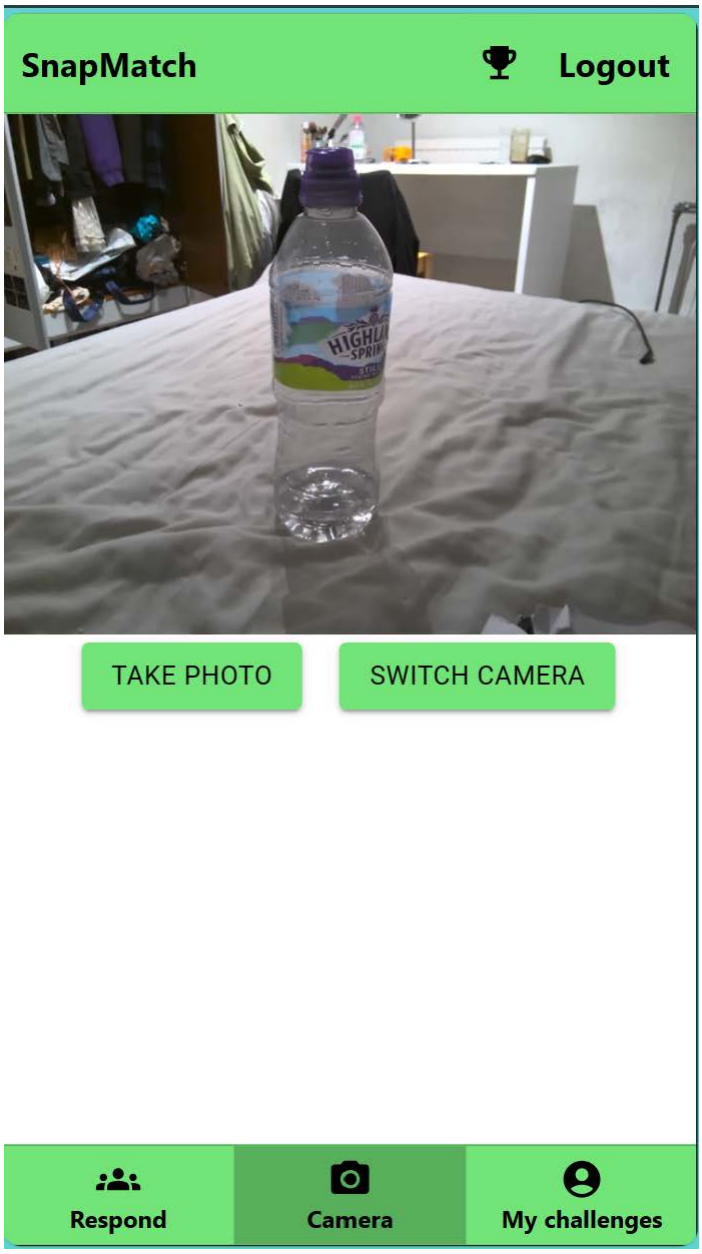
B2 – Upload challenge screen



B3 – Inline documentation

```
        # Generate a unique filename, for example, using a timestamp
        import time
        timestamp = str(int(time.time()))
        filename = f'captured_photo_{timestamp}.jpg'

        # Upload the photo to Dropbox
        dropbox_path = os.path.join(DROPBOX_FOLDER_PATH, filename)
        dropbox_client.files_upload(photo_binary, dropbox_path, mode=WriteMode('add'))

        # Get the shared link of the uploaded photo
        challenge_image_url = get_direct_image_url(dropbox_client, dropbox_path)

        # Insert challenge into the database
        user_id = data.get('userID', random.randint(1, 100))  # If userID is not provided, default to 1

        # turn the objects list into a string
        objects_str = ' | '.join(objects)
        insert_challenge_into_db(user_id, challenge_image_url, objects_str, caption)

        # TODO: refactor this part
        # 1. Send the objects list to the frontend by return jsonify(objects)
        # 2. Share the recource(image_url) with the route /api/saveSelectedObjects
        return jsonify({'message': 'Photo received and uploaded to Dropbox successfully'})
```

B4 - updated UI design



Snap Match

Software Manual

Team17

Table of Contents

Introduction

Problem Statement

In today's digital age, university students face immense pressure from academic commitments, often leading to a lack of social interaction and exploration of their surroundings.

"SnapMatch," a novel initiative by UNiDAYS, aims to mitigate this issue by encouraging students to engage with the world outside their textbooks through a digital platform. The challenge lies in crafting a secure, accessible, and compelling online space where students can share and partake in photo-based "challenges" related to specific places or objects.

A significant aspect of SnapMatch is devising an accurate method to verify that the images submitted in response to these challenges genuinely "match," thereby preserving the platform's integrity and user engagement. Additionally, integrating a scoring system to track and reward user participation adds complexity, requiring careful consideration to ensure fairness and sustained interest.

Goals and Scope

Goals:

The primary objective of the SnapMatch project is to create a dynamic software platform that acts as a digital scavenger hunt, specifically tailored for university students. The platform aims to:

- **Encourage Exploration and Social Interaction:** Enable students to issue and respond to photographic "challenges," promoting a deeper connection with their surroundings and fellow students.
- **Foster a Sense of Community:** Utilize a points-based leaderboard to highlight active participants, creating a competitive yet collaborative atmosphere.
- **Ensure Authenticity and Engagement:** Employ advanced image recognition technology to accurately determine matches, maintaining user interest and platform credibility.
- **Promote Safety and Fun:** Balance engaging activities with user safety and privacy through thoughtful feature design and implementation.

Scope:

The scope of the SnapMatch project includes:

- **User-Friendly Design:** Crafting an intuitive interface that resonates with the student demographic, ensuring ease of use and broad accessibility across different devices.
- **Standalone Operation:** Creating the platform to function independently from the main UNiDAYS platform, with potential for future integration.
- **Innovative Scoring System:** Developing a transparent and motivating scoring mechanism that rewards users for successful matches, highlighting the top participants on a leaderboard to stimulate continuous engagement.
- **Cross-Platform Accessibility:** Guaranteeing seamless access and functionality across various platforms, enabling widespread participation.
- **Scalability and Expansion:** Designing the system to support a growing number of users and potentially expanding the range of challenges and features over time.

SnapMatch is designed to enrich the university experience by motivating students to explore new locations, foster connections, and engage in friendly competition, all within a digitally secure and stimulating environment.

Terminology

General Terms

- **Challenge**: Refers to an image of a place or object uploaded by a user that serves as a prompt for others to find and capture a matching image.
- **Response**: An image submitted by another user that attempts to match the challenge based on predefined criteria.
- **Match**: A successful validation where two images are recognized as containing the same or similar objects or landmarks.

Technical Terms

- **Image Recognition**: Technology used to analyze and recognize content within images, crucial for verifying matches between challenges and responses.
- **Scalability**: The ability of the software architecture to handle increased loads, whether in data volume, number of users, or both, effectively and efficiently.
- **AI (Artificial Intelligence)**: Refers to the machine learning models used to automate image matching processes, reducing dependency on manual validation.

Project Specific Terms

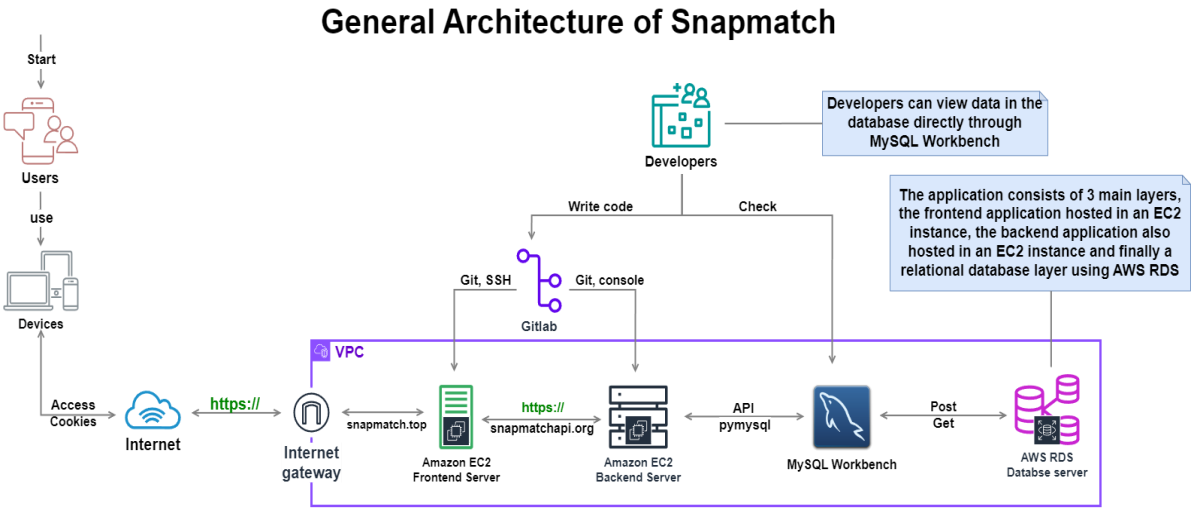
- **Leaderboard**: A feature within the app that ranks users based on the points earned from successful matches to incentivize engagement.
- **Gamification**: The application of game-design elements in non-game contexts, utilized in SnapMatch to enhance user interaction and retention.
- **Accessibility**: Ensuring the app is usable for a wide range of students, including those with disabilities, emphasizing user-friendly design and compliance with accessibility standards.

User Interface (UI) Terms

- **Thumbnail**: A small image representation used in user feeds to give a preview of challenges.
- **Real-time Preview**: A feature allowing users to view a live display of potential images to capture, ensuring the desired quality and composition before submission.

High-Level Structure and Concepts

> This section provides a comprehensive overview of SnapMatch, detailing the system's organization, functionality, and its alignment with the project's goals.



Overview of the Application

SnapMatch is meticulously architected to foster a dynamic and engaging user experience, encouraging students to explore and interact with their surroundings. The system is partitioned into three primary layers, which collectively work to seamlessly deliver the application's core functionality:

- **Frontend Application**: Serves as the first point of interaction for users, offering a responsive and intuitive interface to capture and issue challenges.
- **Backend Application**: Acts as the operational core, handling business logic, user requests, and responses.
- **Database Layer**: Utilizes AWS RDS to ensure robust, scalable, and reliable data storage and retrieval mechanisms.

Main Components

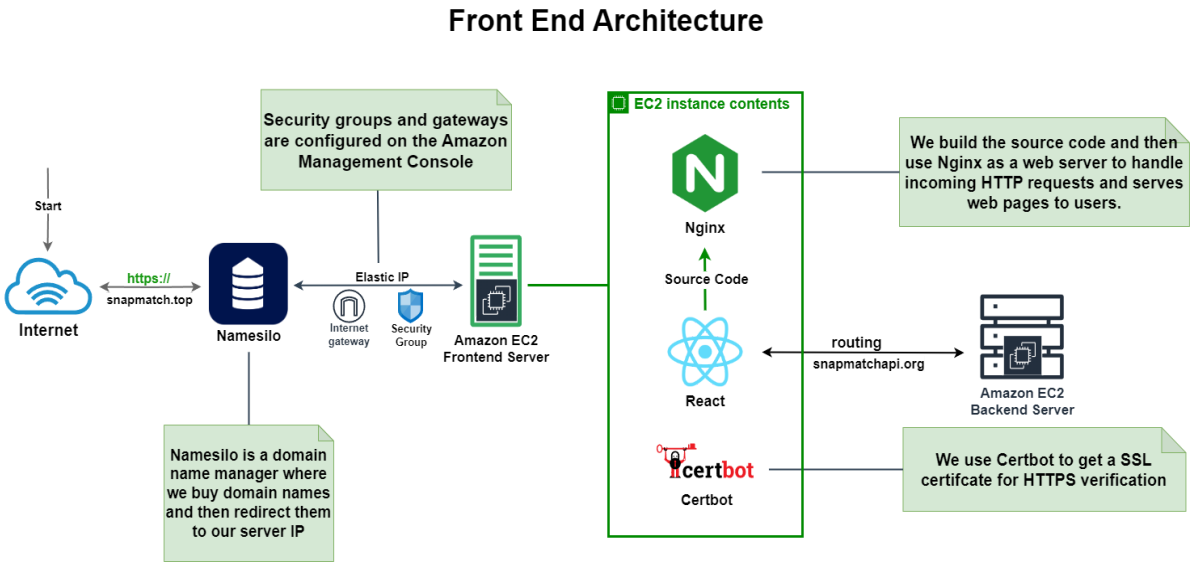
- **Users and Devices**: Users access SnapMatch through a multitude of devices, each interaction being a testament to the platform's cross-compatibility and user-centric design.
- **Overall Flow**: Initiating with a challenge creation or response by a user, the flow progresses through a secure and swift backend process before culminating in a rewarding user experience.
- **VPC and Internet Gateway**: A Virtual Private Cloud (VPC) safeguards the environment, with an Internet Gateway serving as a nexus to the external digital world, assuring secure and fluid connectivity.
- **Layers of the System**: At its essence, SnapMatch is a confluence of a user-facing frontend, a robust backend, and a comprehensive database layer. This trinity harmonizes to deliver an experience that is both technically sound and user-friendly.

Architectural Design

> This part delves deeper into the technical specifics of how the system is designed

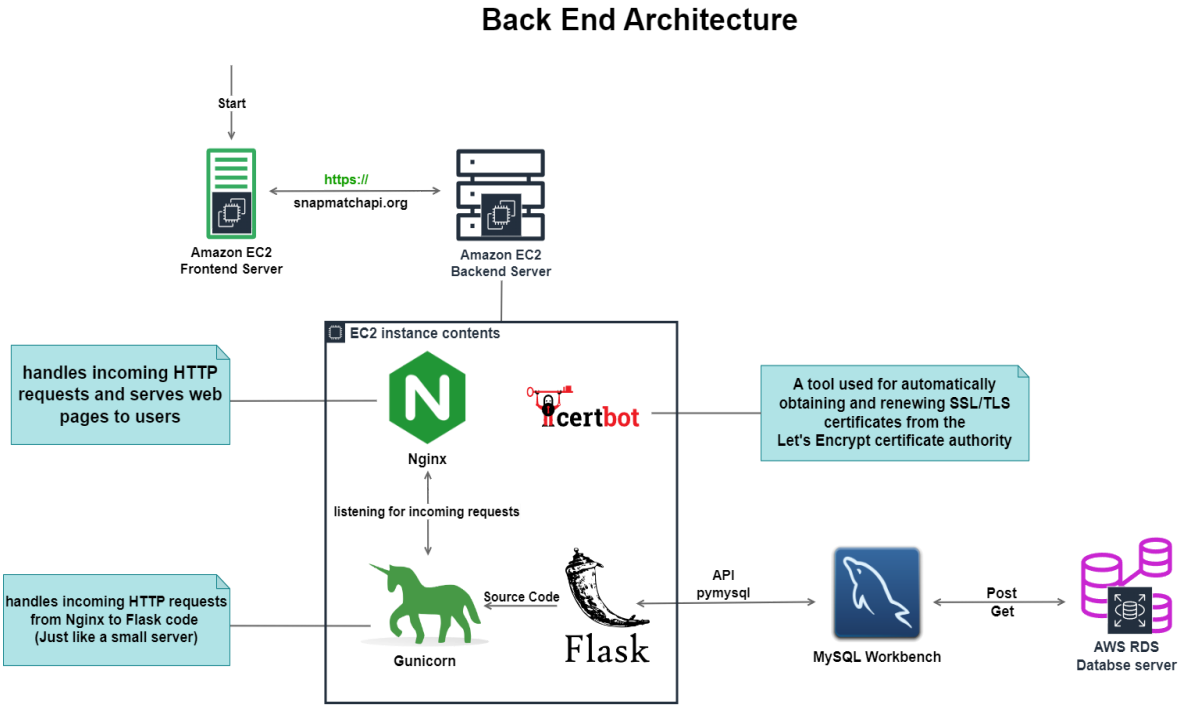
Logical Architecture

Components and Layers



Frontend Application:

- Users interact with the application through various devices, accessing it via the `snapmatch.top` domain.
- The frontend is hosted on an Amazon EC2 instance, providing the interface for user interactions.
- React is used for building the user interface, offering a dynamic and responsive experience.



Backend Application:

- Backend services are accessed through the `snapmatchapi.org` domain.
- The backend is also hosted on an Amazon EC2 instance, handling business logic, authentication, and serving API endpoints.
- Flask, a Python web framework, is employed to facilitate the creation of web services and ensure communication with the frontend.

Database:

- A relational database managed by AWS RDS stores all application data, including user-generated challenges and responses.
- MySQL Workbench is utilized for database administration and data visualization, allowing developers direct access to the database structure and data.

Online documentation for the tools is here

Data Flow

- The data flow begins with users submitting requests from their devices, which are directed to the frontend application.
- The frontend communicates with the backend application via HTTPS requests, secured with SSL/TLS certificates managed by Let's Encrypt and Certbot.
- Backend processes the requests, interacts with the database for data retrieval or storage, and sends responses back to the frontend.
- In the case of developers, GitLab is used for version control and code repository management.

Service Architecture

- Nginx acts as a reverse proxy, routing incoming requests to the appropriate Flask application routes.
- Security groups and Internet Gateways in the AWS environment ensure secure and controlled access to the application services.

Physical Architecture

Infrastructure

> SnapMatch is built on a robust and scalable cloud infrastructure provided by Amazon Web Services (AWS).

The physical backbone consists of:

- **Amazon EC2 Instances**: The front-end application is hosted on a dedicated EC2 instance for handling user interfaces and client interactions. Similarly, the backend logic resides on a separate EC2 instance, ensuring a clear separation of concerns and enhanced manageability.
- **AWS RDS**: For data persistence, we utilize AWS Relational Database Service (RDS), offering high availability and scalability to handle the dynamic nature of user-generated challenges and responses.

Online documentation for the AWS is here

Deployment

> SnapMatch's deployment utilizes a service-oriented architecture (SOA) to ensure that each service is developed, deployed, and managed independently. This allows for greater flexibility and agility in the development process.

The deployment strategy is as follows:

- **Isolation**: Each core feature of SnapMatch is deployed as a separate service on a different AWS EC2 instance, enhancing maintainability
- **Version Control Integration**: With GitLab, we ensure that our codebase remains up-to-date and facilitate collaborative development. This integration aids in maintaining a clear history of changes and simplifies the process of rolling back if necessary.

Scalability and Performance Considerations

> Our infrastructure is optimized to handle varying user activities on SnapMatch without compromising performance:

- **Service Scalability**: If user volume surges, individual services can scale horizontally by adding more instances as needed, and use AWS Elastic Load Balancing (ELB) for load balancing
- **Database Management**: AWS RDS is employed to manage our database needs, providing a managed relational database environment that scales in response to our application's demands.
- **Status monitoring:** The back-end console continuously monitors system logs, which can provide insights into the application's operating status and user behavior to guide our expansion decisions.

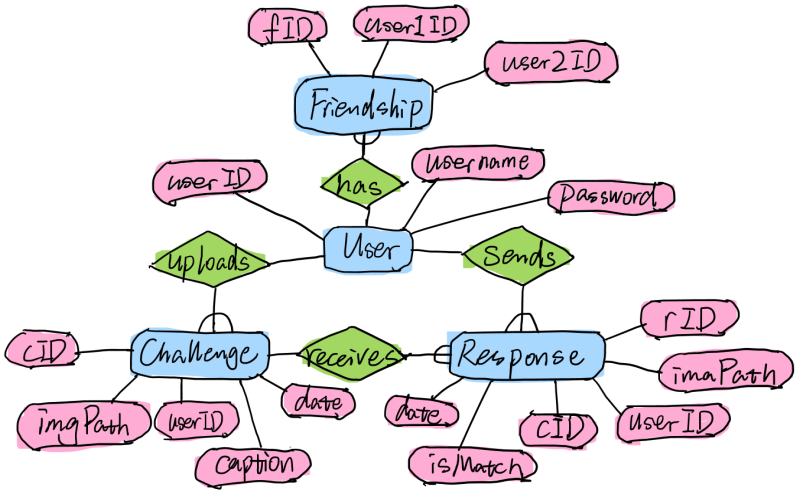
Data Storage

Entity Relationship Diagram

Purpose and Structure

> The Entity Relationship Diagram (ERD) serves as a visual representation of the data model for the SnapMatch application. It outlines the database's entities, their attributes, and the relationships between these entities.

Diagram Explanation



The ERD for SnapMatch demonstrates the relational structure between the following primary entities: `Users`, `Challenges`, `Responses`, and `Friends` (optional). The `Users` entity forms the core of the user management system, detailing user information and login credentials. `Challenges` and `Responses` are central to the application's interactive features, allowing users to post and reply to challenges. The optional `Friends` entity can be implemented for adding a social networking dimension to the application.

Entity Descriptions

- **Users**: This entity represents the registered users of the SnapMatch platform. Key attributes include `UserID`, a unique identifier, as well as `UserName`, `UserAccount`, `PasswordHash`, `Email`, and `Phone`. Users are associated with the challenges they create and the responses they provide.
- **Challenges**: Identified by `ChallengeID`, this entity represents the challenges created by users. Attributes include `UserID`, linking to the creator, `ImgPath` for storing image URLs, and `Caption` for a brief challenge description. Challenges are linked to responses to determine matches.
- **Responses**: This entity captures users' submissions in response to challenges. It includes the `ResponseID`, `ChallengeID`, and `UserID`, along with `ImagePath`.
- **Friends** (Optional): Should a friendship feature be implemented, this entity would track the connections between users, identified by their `UserID`.

Online documentation for our specific database is here

Technologies

Frameworks

Our application is built upon two robust and versatile frameworks: **React** for the frontend and **Flask** for the backend.

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It enables us to create reusable UI components which makes the development process more efficient and the codebase more maintainable.

On the server side, **Flask** is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. We chose Flask for its simplicity and flexibility, allowing us to create a RESTful API that our React frontend could interact with.

External Dependencies and Libraries

SnapMatch utilizes a series of external dependencies and libraries to enhance functionality and streamline development.

Frontend Dependencies:

- **Axios (v1.6.2)**: A robust HTTP client for making asynchronous requests, allowing for seamless communication between the frontend and backend services of SnapMatch.
- **React Router (v6.20.1)**: Empowers the single-page application with dynamic routing, enabling smooth user navigation without page reloads.
- **Bootstrap (v5.3.3)**: Offers a wide range of responsive design elements, which help in achieving an accessible and mobile-friendly user interface.

Backend Dependencies:

- **Flask-CORS (v4.0.0)**: Handles cross-origin requests from the frontend, enabling secure and restricted resource sharing.
- **PyMySQL (v1.1.0)**: Serves as a database connector, allowing the Flask backend to communicate with the AWS RDS MySQL instance efficiently.
- **Flask-Session**: Integrates server-side sessions into Flask, providing a robust mechanism for user state management across requests.
- **Google OAuth2 Client**: Facilitates user authentication via Google accounts, offering a convenient sign-in option for users.

Cloud Storage and AI Integration:

- **Dropbox SDK (v11.36.2)**: Utilized as an image hosting solution, the Dropbox SDK allows SnapMatch to upload and store challenge images securely in the cloud, making them accessible across the platform.
- **Clarifai API(v10.0.1)**: Incorporating Clarifai's advanced image recognition models, SnapMatch can analyze and compare images for challenge responses, ensuring that submissions match the criteria of the issued challenges

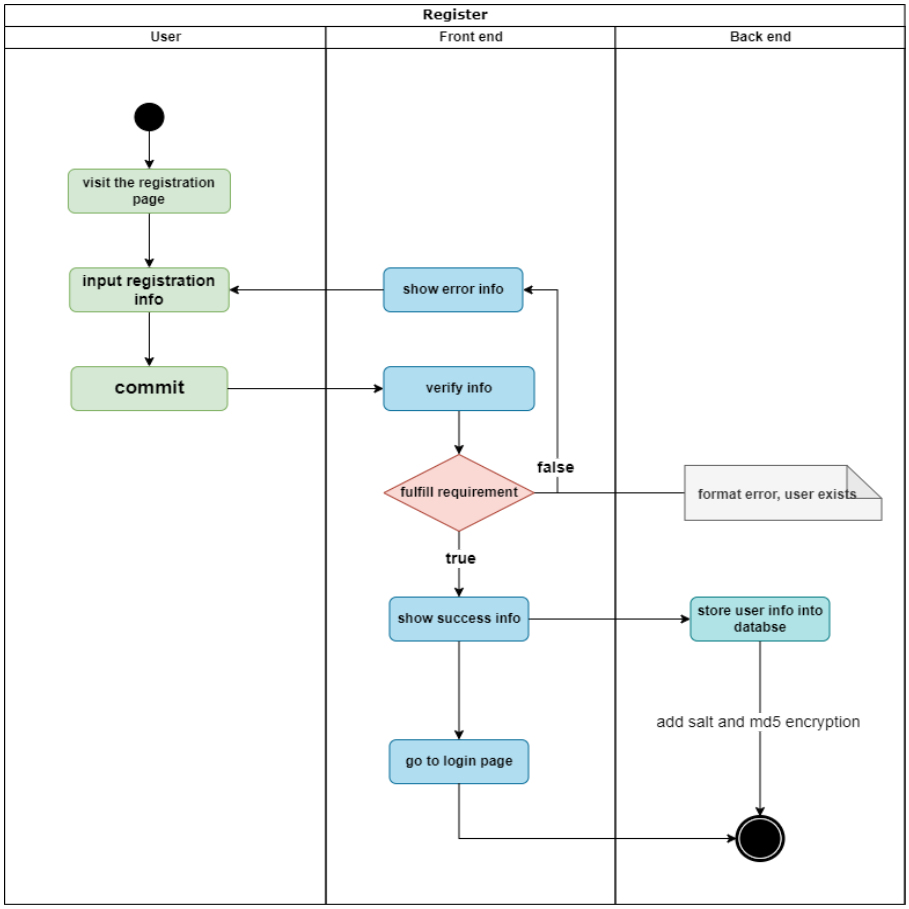
Online documentation for the libraries is here

System Operations

> This section will detail the actions taken by the user, the logic behind how these actions are handled in the system, and what happens on the front and back ends.

Register

Activity Diagram:



User Action: Input new account username and password to register

Front-End Process:

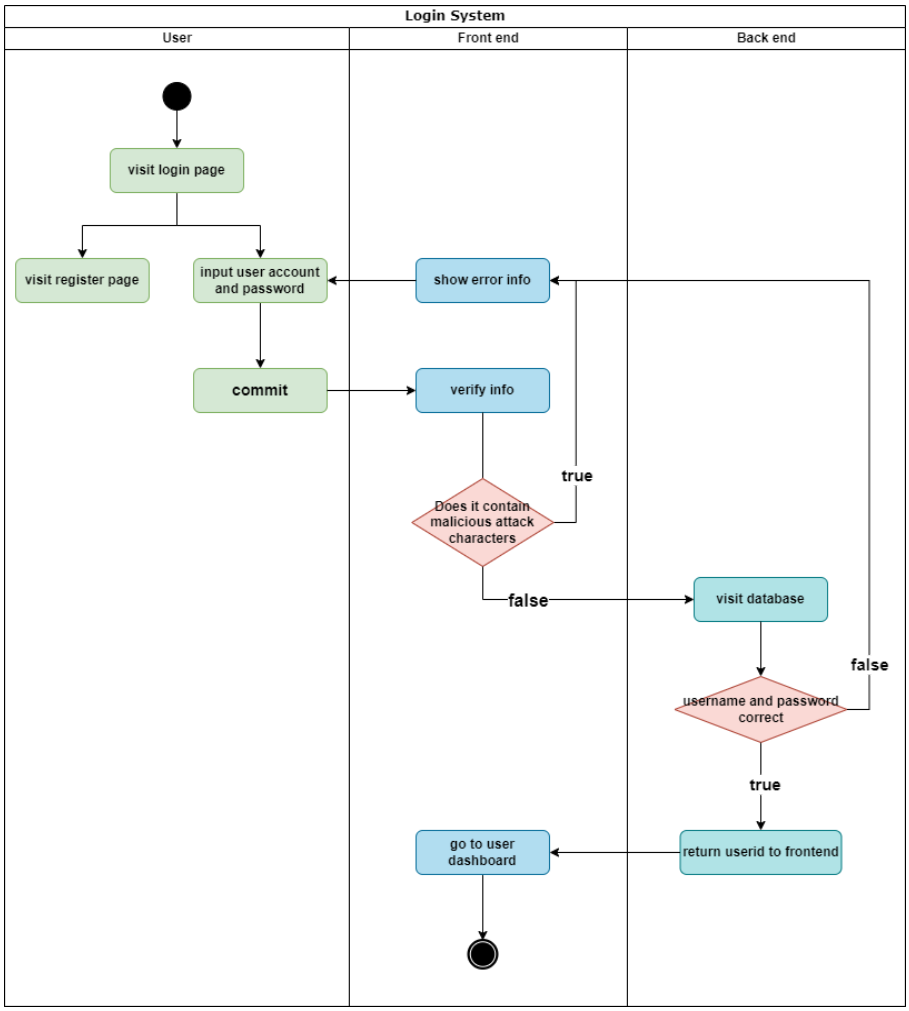
- Displays registration interface.
- Collects user registration details.
- Validates input for format and existing user conflicts.
- Submits registration details to back end.

Back-End Process:

- Receives registration details.
- Validates and checks if user already exists in the database.
- Encrypts password using salt and hash before storing.
- Confirms registration success and prompts front end to redirect user to login page.

Login

Activity Diagram:



User Action: Enter username and password to login

Front-End Process:

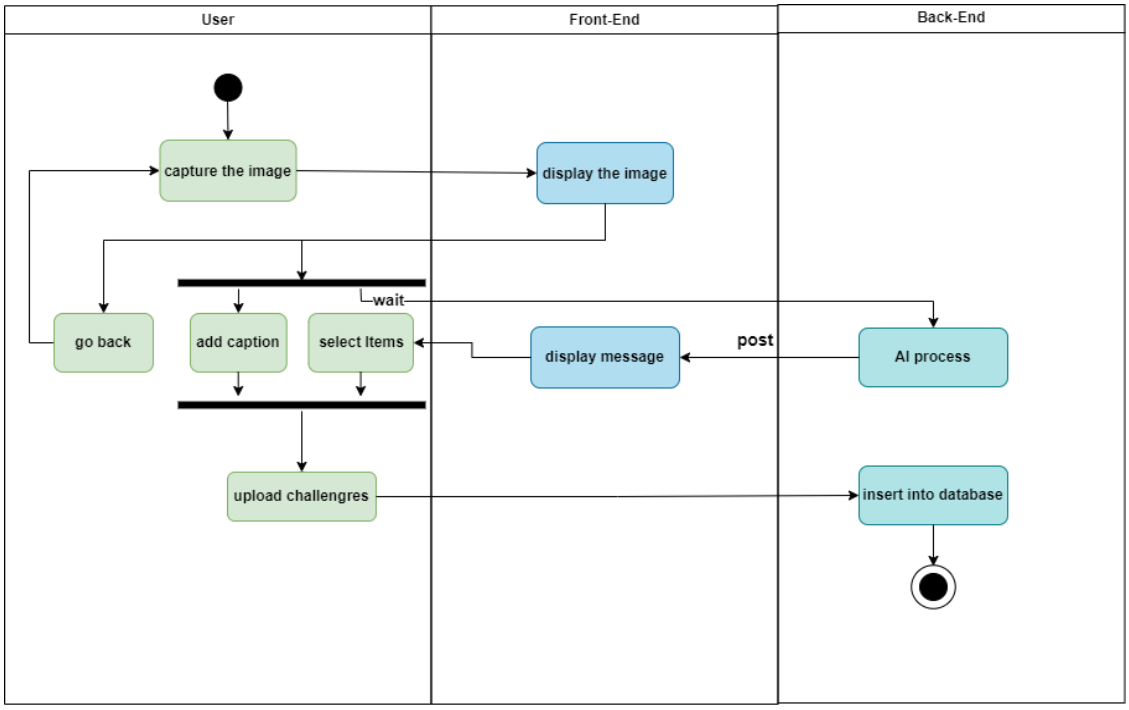
- Displays login interface.
- Collects user input for account and password.
- Validates input format and checks for security breaches, like SQL injection.
- Sends login credentials to back end.

Back-End Process:

- Receives login request.
- Validates against database records.
- If credentials are incorrect, sends error message to front end.
- If credentials are correct, returns user ID to front end, which directs the user to their dashboard.

Upload Challenge

Activity Diagram:



User Action: Captures an image to issue a challenge.

Front-End Process:

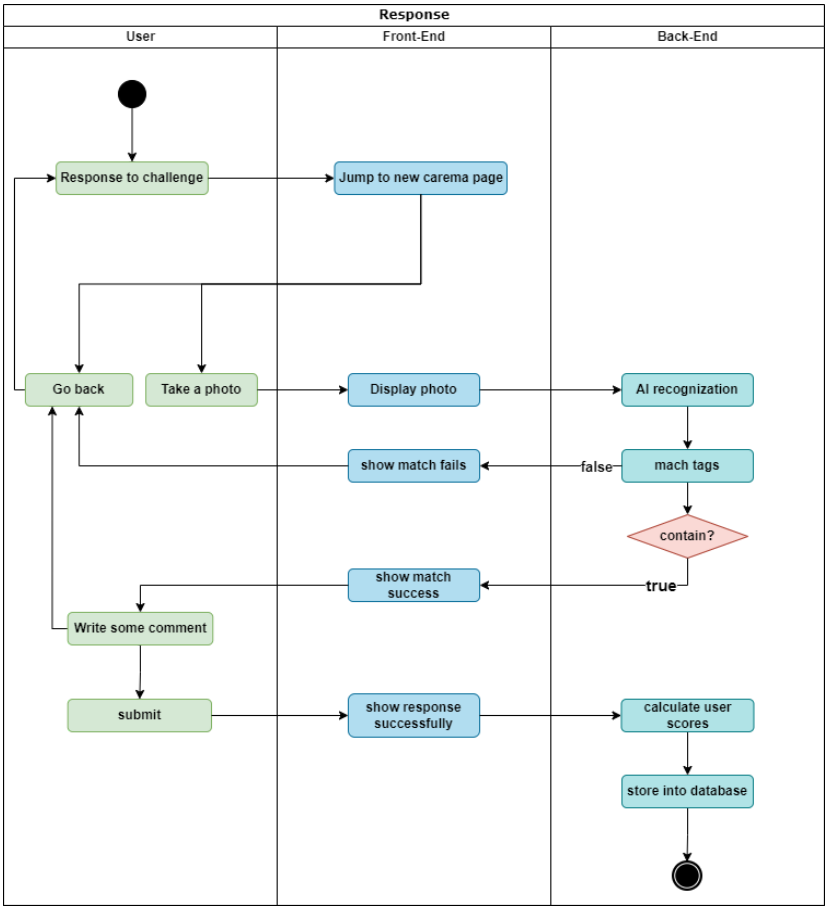
- Provides interface to upload the image and add a caption.
- Users can review their image and add related details.
- Submits the challenge data to back end.

Back-End Process:

- Receives challenge upload request.
- Processes image data, possibly utilizing AI for content verification.
- Inserts challenge data into the database for community access.

Respond to Challenge

Activity Diagram:



User Action: Takes a photo to respond to a challenge.

Front-End Process:

- Displays interface for photo capture and submission.
- Collects user response and submits to back end.

Back-End Process:

- Receives response data.
- Invokes AI recognition to verify if the response matches the challenge criteria.
- Updates the database with the response status and, if a match, calculates and updates user scores.

Coding and Maintainability Conventions

Coding Standards

To ensure readability and maintainability, SnapMatch adheres to widely accepted coding standards.

Python

In the Flask backend, we conform to PEP 8, the official style guide for Python code, which include conventions for code layout, indentation, comments, naming conventions, etc. This guide helps us maintain a consistent style and catch potential errors early in the development cycle.

JavaScript (React)

For our React-based frontend, we adhere to best practices as recommended by the React community. Components and JavaScript files are written following ES6+ syntax to make use of the latest language features, such as arrow functions, classes, let, and const.

Online documentation for the standards is here

Naming Conventions

Consistent and meaningful naming conventions are a cornerstone of our project's code clarity and maintainability.

Python

We employ the snake_case naming convention for our Python code as per PEP 8 guidelines. Variables, functions, and methods are named using all lowercase letters with underscores between words, e.g., `user_profile` or `save_image`. This approach extends to file names as well.

JavaScript (React)

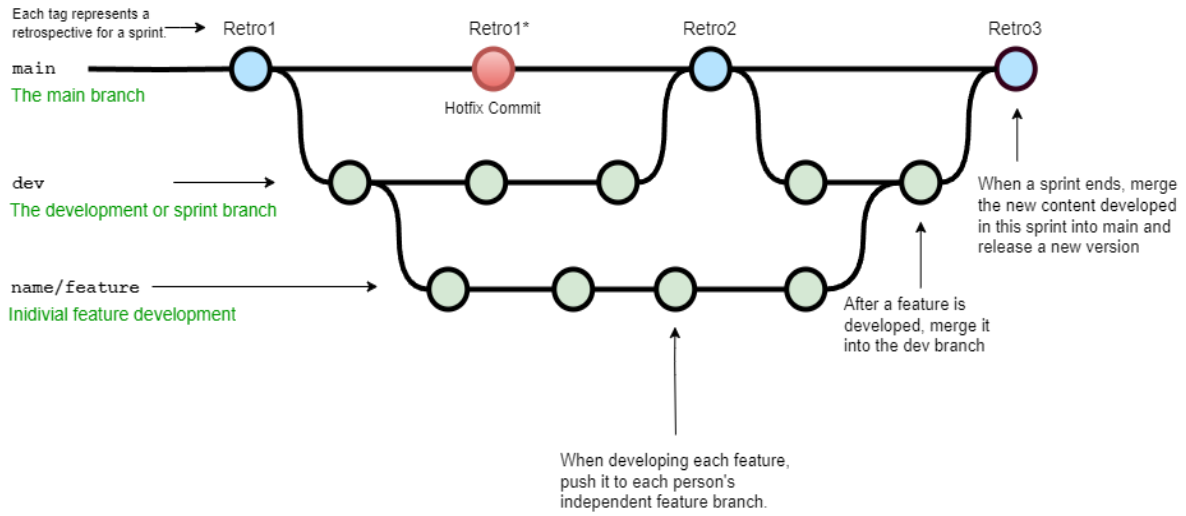
In the React components, we adopt the camelCase naming convention for variables and functions, while class names and component names begin with a capital letter. e.g. `UserProfile` and `ImageGallery`. Additionally, braces `{}` align with common JavaScript practices, predominantly positioned to the right to open blocks of code, enhancing readability and adhering to the community standards.

GitFlow Workflow

Our team adheres to the GitFlow workflow, which is a branching model designed to provide a robust framework for managing larger projects.

Diagram for Team 17 's "Git-flow" workflow

See: https://projects.cs.nott.ac.uk/comp2002/2023-2024/team17_project



Main Branch:

- The `main` branch holds the official release history and is the source of truth for production-ready state.

Development Branch:

- Parallel to the `main` branch, the `dev` branch serves as an integration branch for features. It represents the next planned release and contains the state of the upcoming version.

Feature Branches:

- Individual feature branches (`name/feature`) branch off from `develop` and are used to develop new features in isolation. After completion, these are reviewed and merged back into the `develop` branch.

Hotfix Branches:

- If an issue is identified post-release, a `hotfix` branch is created to quickly patch the production release. Hotfixes are the only branches that can fork directly off `main` .

Online documentation for the Gitflow is here

Design Patterns Used

In the SnapMatch project, we have strategically utilized several design patterns to streamline our development process and enhance our codebase's maintainability and scalability.

Model-View-Controller (MVC):

- The **Model** component is our database that stores all data-related logic.
- The **View** is represented by our front-end React application, which renders the UI.
- The **Controller** is our Flask backend application, handling incoming requests and business logic, effectively bridging the Model and View.

Singleton Pattern:

- Our database connection is managed by a Singleton pattern, encapsulated within a function `get_db_connection` . This ensures that there is a single, globally available connection to the database, reducing overhead and preventing multiple connections from spawning.

Observer Pattern:

- The Observer pattern is employed on the front end, particularly for handling uploads. Asynchronous functions like `uploadResponse` observe and react to the state of file uploads, allowing the application to handle user interactions and data updates in real-time, providing a responsive user experience.

Testing Approaches

The testing strategy for SnapMatch has been a combination of manual and automated tests, aimed at verifying the application's functionality and user experience without incurring excessive maintenance overhead.

Manual Testing

In the early stages, we embraced manual testing due to the application's rapid development cycle. This approach provided the flexibility to adapt to changes quickly and offered immediate feedback on new features and integrations.

Integration and End-to-End Testing with Cypress

Cypress was used for comprehensive front-end end-to-end testing, running tests directly in the browser to simulate real user interactions. Cypress's built-in test runner helped us quickly identify and resolve issues, ensuring all application features functioned as expected.

Our testing began with basic assertions for each page, such as verifying the presence of essential elements like cameras and buttons. This initial check ensured completeness before we proceeded to interactive tests, which involved navigating through the app and engaging with its features.

For more complex tasks like uploading responses, we implemented stubs to emulate backend interactions, providing a thorough validation of the app's operational capabilities. This structured testing approach streamlined debugging and maintained app consistency across various components.

Clarity of test scripts is also emphasized, with detailed comments and descriptions provided to enhance understanding and maintainability in future code reviews.

Pytest for Backend Testing

For backend code, **pytest** was chosen for its simplicity and powerful features. It enabled us to focus on validating our backend functionality through a series of integration tests that mimic the actual use of the system rather than isolated units.

API Unit Testing

We conducted unit testing for key API endpoints like "Upload Challenge" using a unit test framework. This process involved sending POST requests with base64 encoded sample images and verifying successful responses, ensuring the reliability of the API as development progressed.

Alpha and Beta Testing

Our alpha testing was integrated into the sprint review process, with demonstrations and tests conducted during retrospective meetings. For beta testing, we deployed the website and invited friends to use the application, gathering user feedback to identify and fix any emerging bugs.

Online documentation for the Test Record is here and more details please see the repository

Build Manual

Backend Setup

1. **Installation**:
    - Clone the repository from GitLab using the command `git clone https://projects.cs.nott.ac.uk/comp2002/2023-2024/team17_project.git`.
    - Install Python(v3.10.6), ensuring `pip` is also installed.
2. **Dependencies**:
    - Navigate to the `Enviroment\snapmatchproject\backendFolder\APIFiles` directory.
    - Execute `pip install -r requirements.txt` to install the necessary Python packages.
3. **Database Configuration**:
    - Set up an AWS RDS instance for the MySQL database.
    - Configure the database connection in `database_utils.py`.
4. **Server Initialization**:
    - Inside the backend directory, run `python app.py` to start the Flask server.
    - Or Execute `npm run start-flask-api` in `package.json`
    - Ensure the server is running by accessing `localhost:5000` or the configured port.

Frontend Setup

1. **Environment Setup**:
    - Ensure Node.js(v16.14.2) and npm are installed.
    - Use `npm install` to install all the dependencies listed in `package.json`.
2. **Running the Development Server**:
    - Execute `npm start` to launch the React application in development mode.
    - The application should now be accessible via `localhost:3000` or the next available port.
3. **Building for Production**:
    - Run `npm run build` to create an optimized production build.
    - Deploy the build on a suitable web server.

Project Online Document

- Test Record
- Bug list
- Frontend Manual
- Backend Manual
- Database Design
- Issue Workflow
- Backend Requirement
- React App and API

Further Online Documentation

Frameworks:

- Flask (v2.0.2)
- React(v18.2.0)

Extensions:

- Flask-cors (v3.0.10)
- Dropbox Python SDK Documentation
- PyMySQL(v1.1.0)
- Clarifai API(v10.0.1)
- Google OAuth2 Client

Testing:

- Cypress
- Pytest

Production Environment:

- Nginx
- Gunicorn
- Certbot

Coding Standards:

- PEP 8
- ES6+

Workflow:

- Gitflow

Amazon Server:

- AWS EC2
- AWS RDS

Snap Match

User Manual

Team17

Table of Contents

Introduction

The SnapMatch app is created to motivate people from various backgrounds to make more memorable experiences and explore their surroundings. This app aim to turn the world into a scavenger hunt. A user should be able to upload images of unique places or objects, challenging the community to find and capture matching photos.

Online Resources

Here is a **YouTube link** showing the product and teaching how to use it.

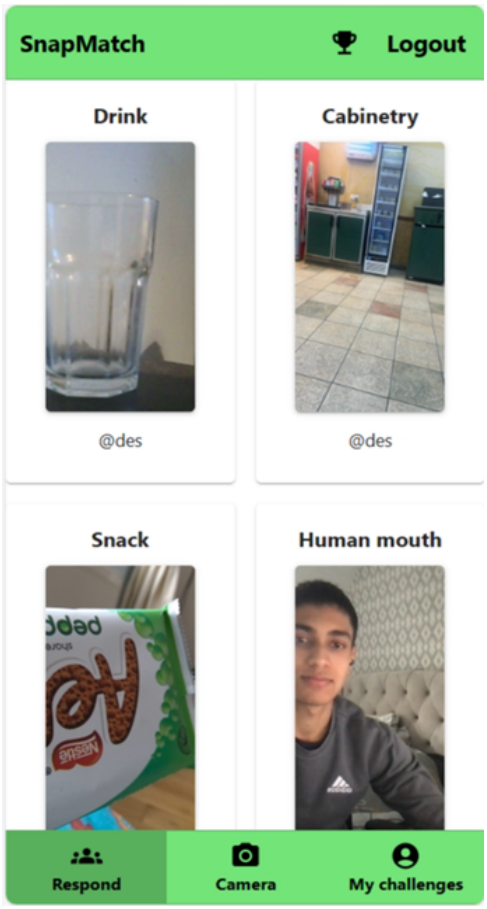
Requirements

A device (desktops, laptops, tablets, smartphones etc.) with a camera and has access to the internet.

How to Access

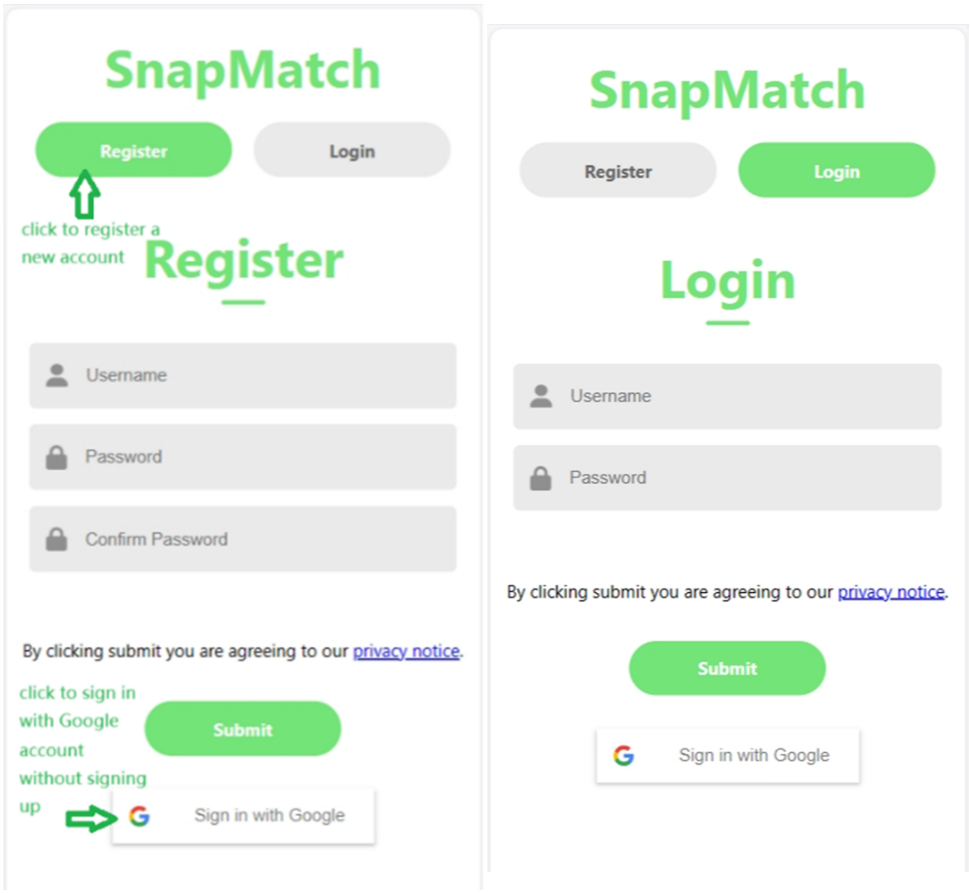
Our app is currently an online website, you can go to **https://snapmatch.top/** to access our app.

User Interface Overview



Getting Started

1. Open our website(**https://snapmatch.top/**) on your device.
2. Sign Up:
    - If you have a Google account, you can skip this step and directly login using your Google account by clicking “Sign in with Google” button on the bottom of the login page.
    - If you would like an exclusive new account for our app, click “register” button in the upper left and type in the username and password you like to register a new account.

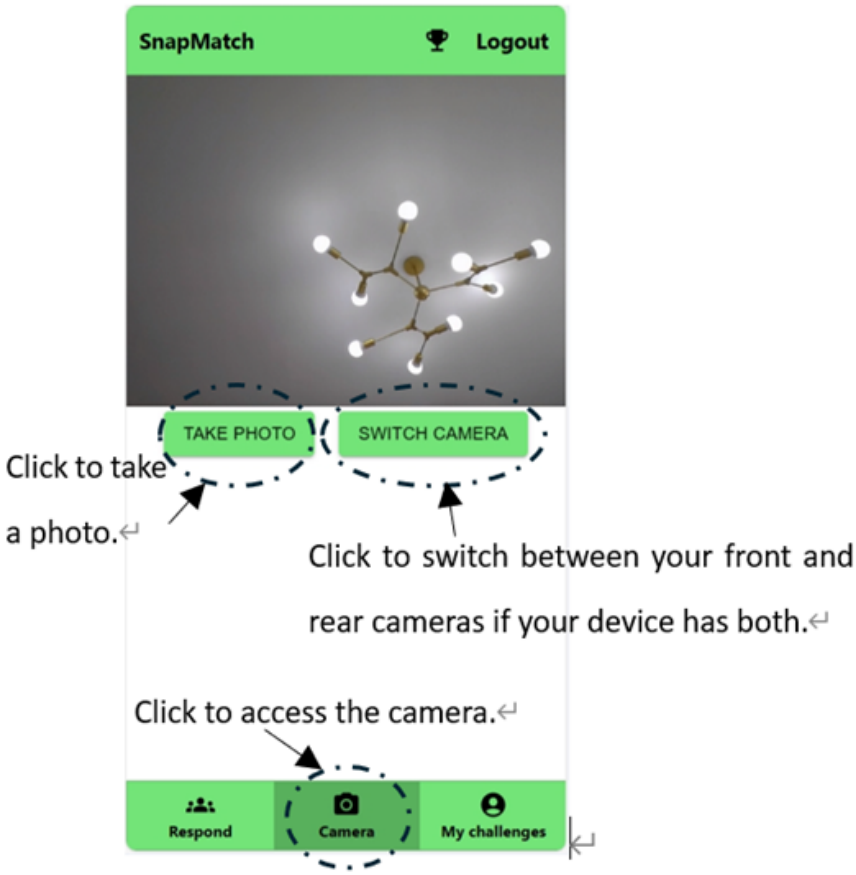


3. Login: After signing up, log in with your username and password.

Main Features

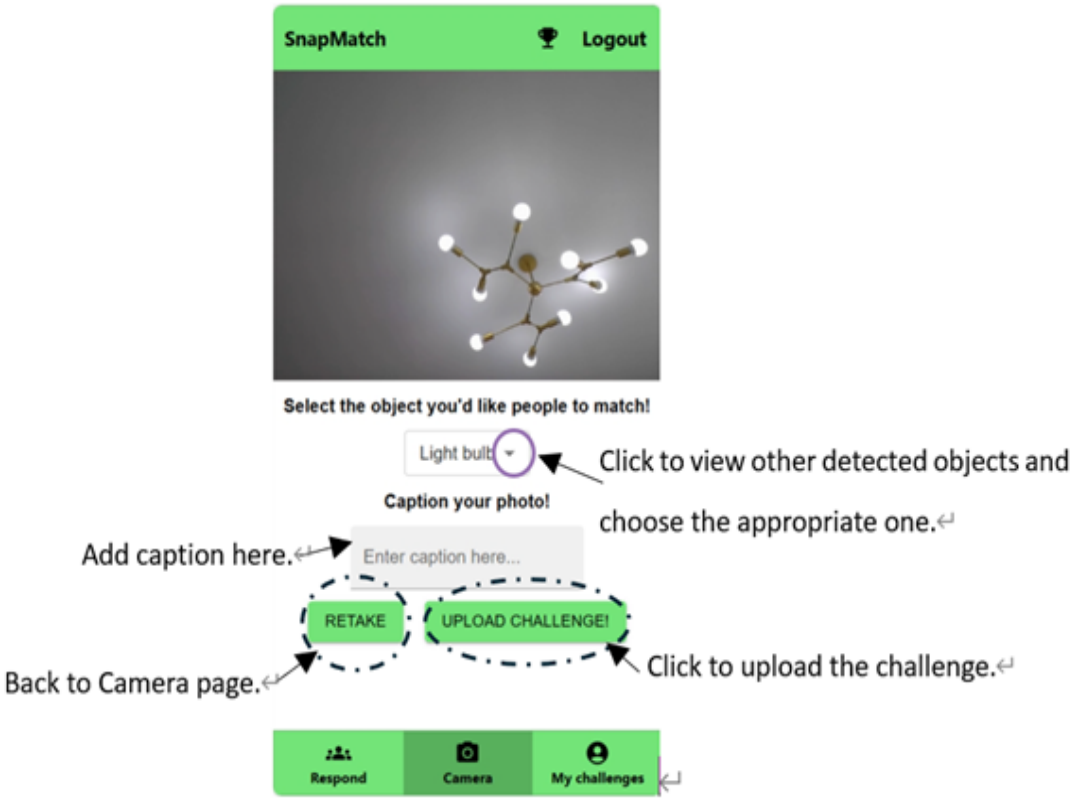
1. Take a photo and upload as a challenge for others to respond to:

1. Click “Camera” in middle of the bottom bar to access the camera on your device. If it’s your first time to access the camera, your device will ask you whether allow the website to access the camera. After choosing yes, your camera will work in this page and your can see the dynamic scene captured by the camera.
2. If you would like to switch between your front and back camera, click “SWITCH CAMERA”. After capturing the object you like, click “TAKE PHOTO” to freeze it for further AI detecting and editing.

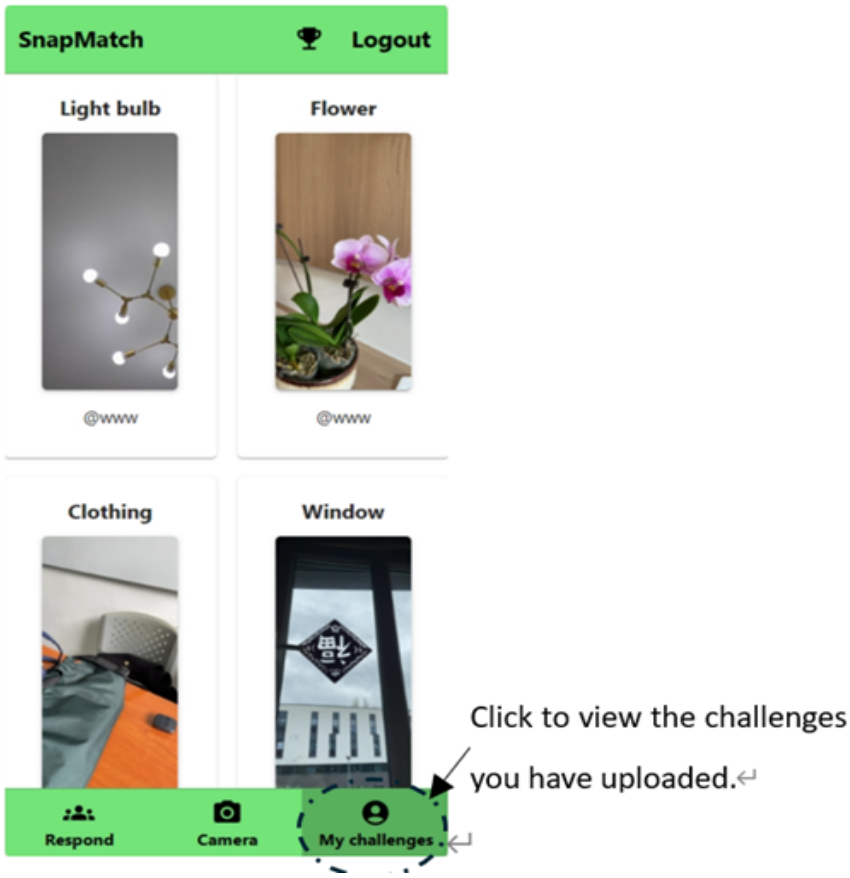


3. After clicking “TAKE PHOTO”, it may take a few seconds for the AI tool to detect the objects in the captured photo, please be patient 😊. Then it will take you to a page including the detected objects for you to select and a text box for you to add caption.

   If you’re not satisfied with the photo, you can click “RETAKE” and it will bring you back to the camera page. Or if the photo is ok and you can find the appropriate object in the expandable box (circled by a purple circle in the picture below), after entering the caption, click “UPLOAD CHALLENGE!” to upload it!

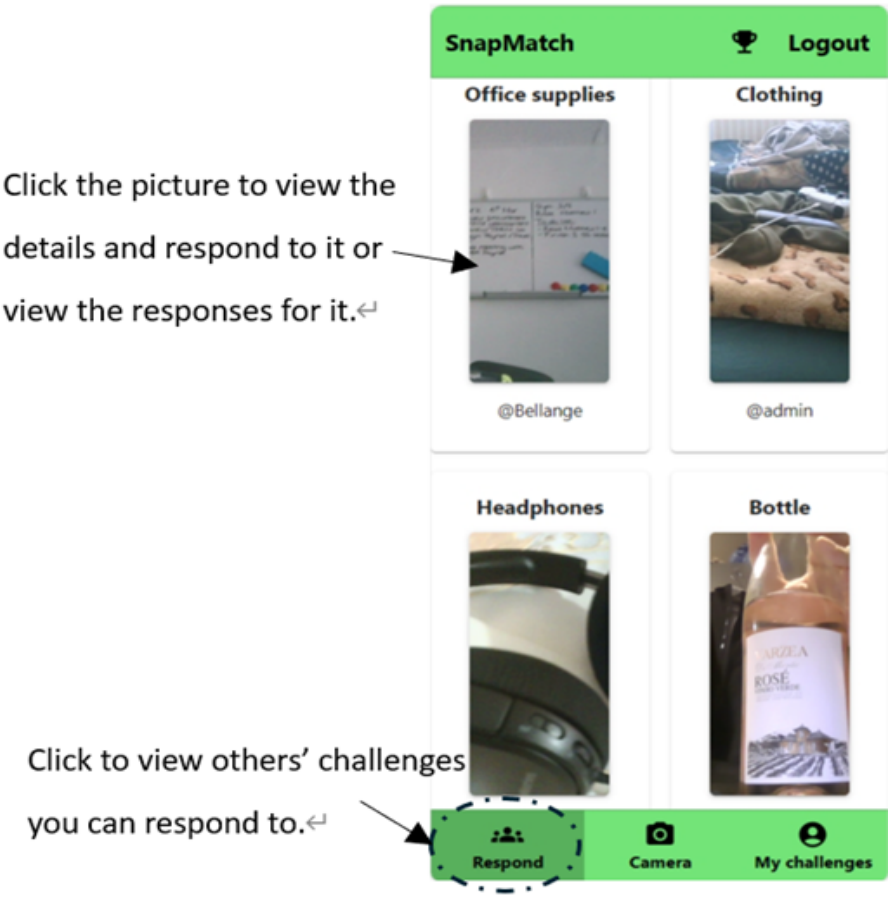


4. After the challenge is uploaded successfully, it will appear in “My challenges” and you can review it and see others’ responses here. However, you can’t response to your own challenges.



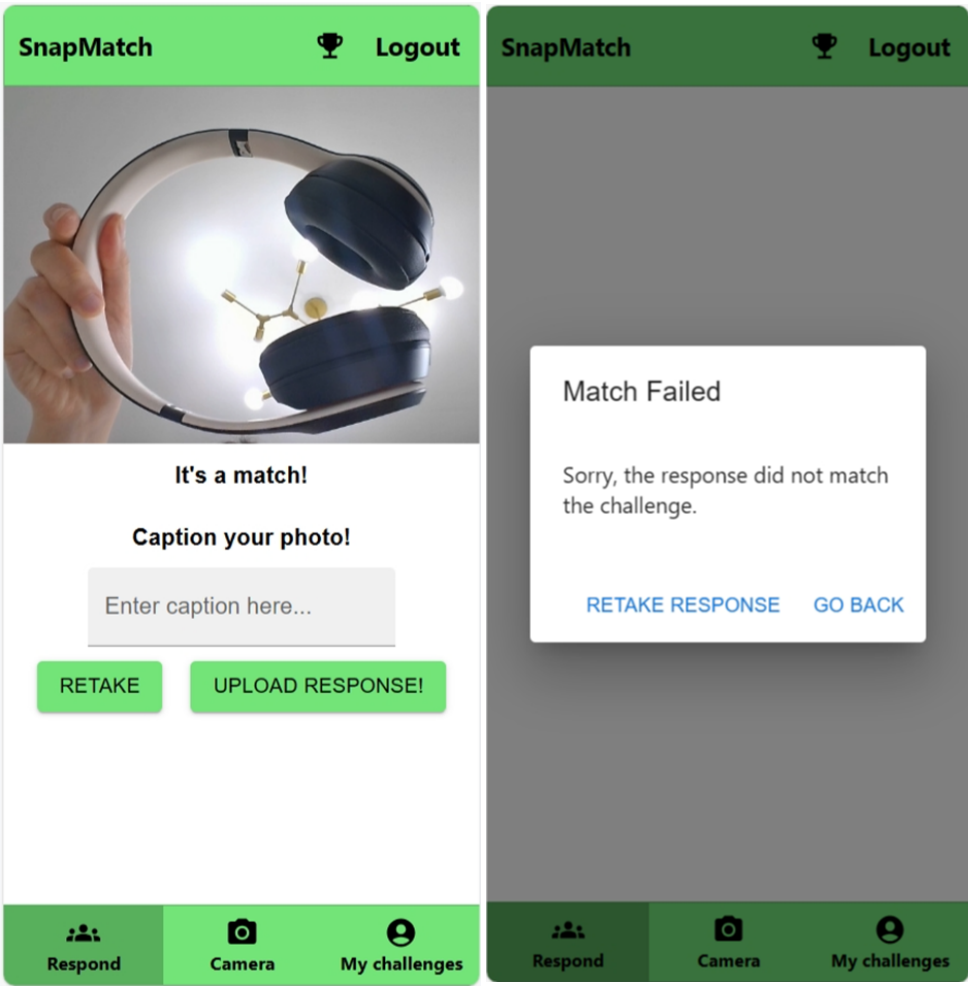
2. Respond to others' challenges:

1. Go to the "Respond" page and choose a challenge you would like to respond to.

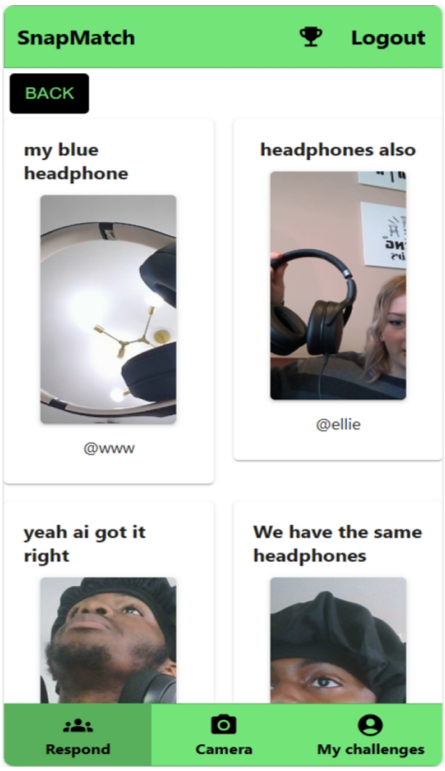


2. Click the thumbnail of the challenge you like to view its details. In this page, you can click "RESPOND!" to respond to the challenge by taking a new photo including the same object as the challenge's. You can also click "VIEW RESPONSES" to view the previously uploaded responses to this challenge by other users.

3. After clicking “RESPOND!” and taking a response photo, the AI tool will detect all the objects in the photo and check if they include the same object as the challenge’s (this process may take a few seconds). If match successfully, it’ll take you to the page as the first screenshot below. Otherwise, you will be informed that “Match Failed” as the second screenshot below.

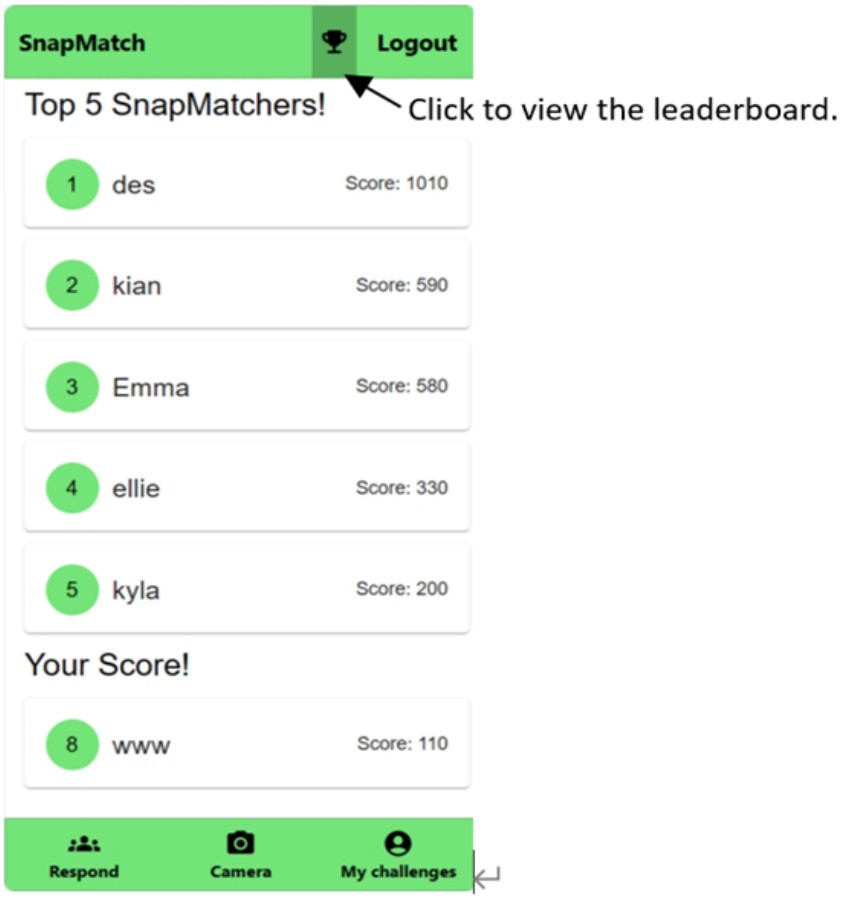


4. If your response matches successfully, after adding some caption, you can click “UPLOAD RESPONSE!” to upload it. Then your response will appear in the challenges’ responses page. In the meantime, every time you successfully uploaded a response, your score will increase by 10. If it’s your first time to make a response, your score will increase by 50!



3. View Your Score and the Leaderboard:

Click the small trophy icon in the upper right corner to view the leaderboard which will show users with the top 5 highest scores. Your score will appear in the bottom of this page. To leave this page, click whatever button in the bottom bar.



Caveats

- To make our application environment harmonious and suitable for all groups of people, please do not upload any content related to pornography, blood, discrimination, or any other illegal or inappropriate content. Once you're found to upload such contents, the inappropriate contents will be fully removed, and your account will be rooted out.
- Don't use any uploaded photo or information in our app without the admission from the user who uploaded it.

Troubleshooting

- **My response exactly has the same object as the challenge but always fail to match:**

  It's because the AI tool we are using now is still limited in ability, it's possible that it can't detect objects in a photo correctly. Please try to take the photo of the object you want to be detected as clearly as possible😊.

- **I forget my username or password:**

  Sorry but we currently don't have the "Retrieve password" functionality, so please remember your password firmly.

- **Not able to see the pictures for each challenge:**

  Try VPN to access our website from another country.

FAQs

What if someone uploads an illegal photo as a challenge or response?

Our team will check the newly uploaded photos regularly. If there is any inappropriate photo uploaded, we'll delete them from the database compulsorily.

Privacy Notice

In **https://snapmatch.top/privacy-notice**, you can find how we collect, use and safeguard your information when you use SnapMatch. Please check it carefully before using our app. By using SnapMatch, you agree to the terms of this privacy notice.

Feedback and Support

We are delighted to receive any feedback and give you support!

Please email us at **scysw3@nottingham.ac.uk** if you have any suggestions or have some trouble when using our app.