Snap Challenges

Photography Gamified App

JACOB ALLEN

19003931

GitHub Repositories:

Code: https://github.com/JacobA2000/Snap-Challenges

Related Content: https://github.com/JacobA2000/UWE-DSP

Main Body Word Count: 10982

UXCFXK-30-3

Digital Systems Project

Table of Contents

Contents

Abstract

Photography is a very popular and enjoyable hobby, however the current existing solutions for sharing images are not very engaging. Existing platforms such as Instagram do not do a good enough job and are not specific enough for Photographers and have recently started shifting focus from the art, “Recently, the head of Instagram, Adam Mosseri, publicly stated that the focus for the popular photo-sharing app, owned by Facebook, is shifting toward video going forward.” (Murphy, 2021). For these reasons photographers need a new innovate and engaging platform to shar there art. This report proposes a new react native app, Snap-Challenges which utilises gamification methods such as challenges and badges to engage users, whilst providing educational benefits for the user base thanks to meta-data about each photo being shared.

Acknowledgements

I would like to thank my supervisor, Frazer Barnes, for his continued support and frequent availability for meetings during the process of writing this report.

I would also like to thank Dr. Martin Serpell, for running Getting Back on Track sessions which frequently provided some insightful support for this module.

Finally, I would like to thank the many other individuals who helped shape this project, my family friends, and fellow students, all of whom helped me in unique ways.

Table of Figures

Table of Tables

1 – Introduction

1.1 – Project Timeline (Gantt Chart)

2021 → **2022**

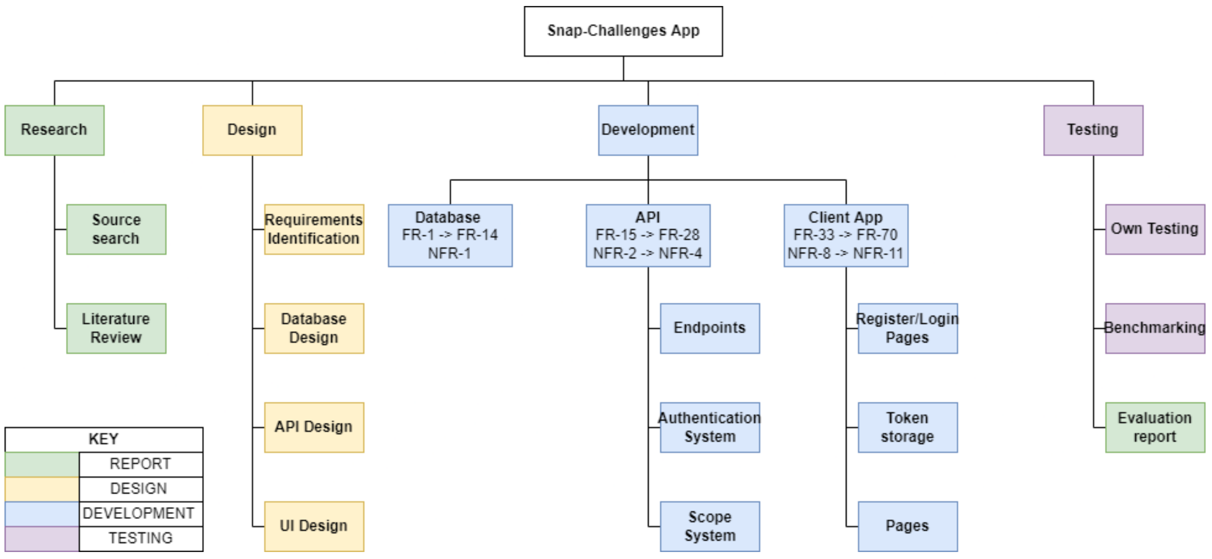
Table 1- Gantt Chart

	OCT	NOV	DEC	JAN	FEB	MAR	APR
Research	🟩	🟩					
Requirements		🟩					
Database Design		🟨					
API Design		🟨					
UI Design			🟨				
API Implementation		🟦	🟦				
Database Implementation		🟦	🟦				
Front End Implementation			🟦	🟦	🟦		
Internal Testing (Own)						🟪	
Finalise Report							🟩

Legend

Report Tasks	🟩
Design Tasks	🟨
Development Tasks	🟦
Testing Tasks	🟪

1.2 – Hierarchical breakdown



1.3 – The problem

There are many existing popular photography sharing platforms, Instagram, 500px and Flicker to name a few. However, none of these sites offer an engaging way to get more images shared and often do not include detailed information about the photographs on the platform. Snap-Challenges will attempt to resolve the gaps left by these sites by offering an engaging and informative photo sharing experience.

Photographers often experience burn out from their art, this can be for a variety of reasons, such as:

- Not being able to reach new locations.
- Not experimenting with styles outside of their comfort zone.
- High cost of new gear.
- A lack of engagement with the content they share.

Whilst Snap-Challenges will not claim to be able to eliminate these burn out factors, it will attempt to mitigate this via gamification methods by providing users with challenges created by each-other. Challenges will be achievable yet stimulating. This approach should result in more engagement with the platform than typical photography sharing platforms.

1.4 – Why it's important

One of the main ways to improve at photography is to get out, and practice. Snap-Challenges gives the user a reason to-do this. Practice will lead to an improvement in their photography which benefits the whole community as we get to see better photos.

The app will show camera settings such as:

- Make
- Model
- Aperture

- Shutter Speed
- ISO
- Focal Length

Providing users with this meta-data about a photograph provides an insight in to how the photo was taken. This will mean that the community can collaboratively improve their photography by observing others work.

1.5 – Scope

Snap-Challenges will be a social-media like experience focused on gamifying photography via challenges and badges. The app will feature a database, API, CDN, and a cross-platform front-end. The app will include several pages, a login system, a challenges page, a profile page, a badges page, and a post page. The app should be intuitive but remain engaging.

1.6 – Aims and Objectives

AIMS:

1. To develop a platform independent app to gamify photography via the use of challenges.
2. To minimise burn out amongst photographers using these challenges to engage them with the app and their hobby.
3. To educate the users of the app by providing information about the camera settings used to achieve the resulting photo.

OBJECTIVES:

1. Research gamification benefits and drawbacks, and other times it has been used in the photography field.
2. Research React Native in comparison to Progressive Web Apps (PWAs).
3. Research data storage solutions MYSQL vs NOSQL.
4. Design and develop the database-schema.
5. Design and develop the Flask API.
6. Design and develop the front-end client as either a React Native app or PWA.
7. Test the system.
8. Produce a report to summarise findings, research, implementation, and testing stages of development.

1.7 – Potential ethical and legal issues

The system handles user data. Passwords will be stored in the database, this could be a security concern, to minimise risk passwords will be hashed using the SHA256 algorithm and salted. This will be done via werkzeug.security a python module which is already a requirement of flask so using it doesn't increase the number of dependencies. The plaintext password will only be known to the user.

2 – Literature Review

2.1 Gamification

(Hamari and Huotari, 2012) Define gamification as "a process of enhancing a service with affordances for gameful experiences in order to support user's overall value creation." This implies that utilising gamification offers higher engagement from users of a service and therefore making each user more valuable to that service. This definition is backed up by a survey completed by (Wang, et al, 2017) who state, "With regard to users' perceptions, participants found that the gamified HCS (Kpoprally) provided a better engagement experience compared to the non-gamified one."

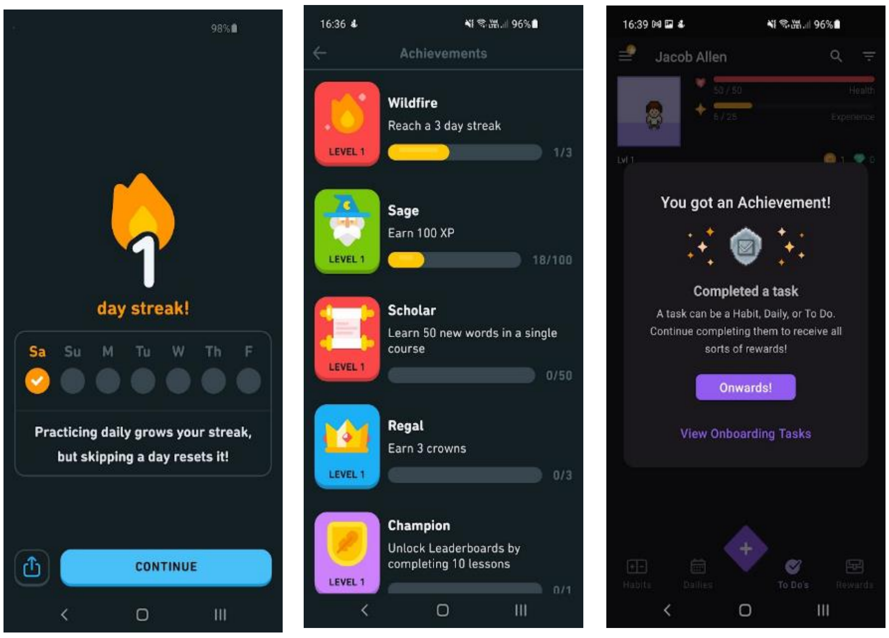
What roles does gamification have to play in an app?

Gamification is applied via a variety of unique approaches, some of which are likely, yet to be conceptualized. Examples of gamification practices are:

- Rewards
- Challenges/Quests
- Achievements
- Leaderboards
- Progress-bars

(Hence, et al, 2017) states "badges, leaderboards, and performance graphs also seemed to contribute to an increase in perceived task mean-ingfulness." This shows that tried-and-true methods of gamification work to ensure the user has a more engaging and positive outlook on the app in general.

Examples of gamification can be seen throughout a wide range of applications both web and mobile, in both massively popular and small niche apps, some of which can be seen in figure 1.


1 day streak!

Sa Su M Tu W Th F

Practicing daily grows your streak, but skipping a day resets it!

CONTINUE

Achievements

Wildfire — Reach a 3 day streak — LEVEL 1 — 1/3

Sage — Earn 100 XP — LEVEL 1 — 18/100

Scholar — Learn 50 new words in a single course — LEVEL 1 — 0/50

Regal — Earn 3 crowns — LEVEL 1 — 0/3

Champion — Unlock Leaderboards by completing 10 lessons — LEVEL 1 — 0/1

Jacob Allen

You got an Achievement!

Completed a task

A task can be a Habit, Daily, or To Do. Continue completing them to receive all sorts of rewards!

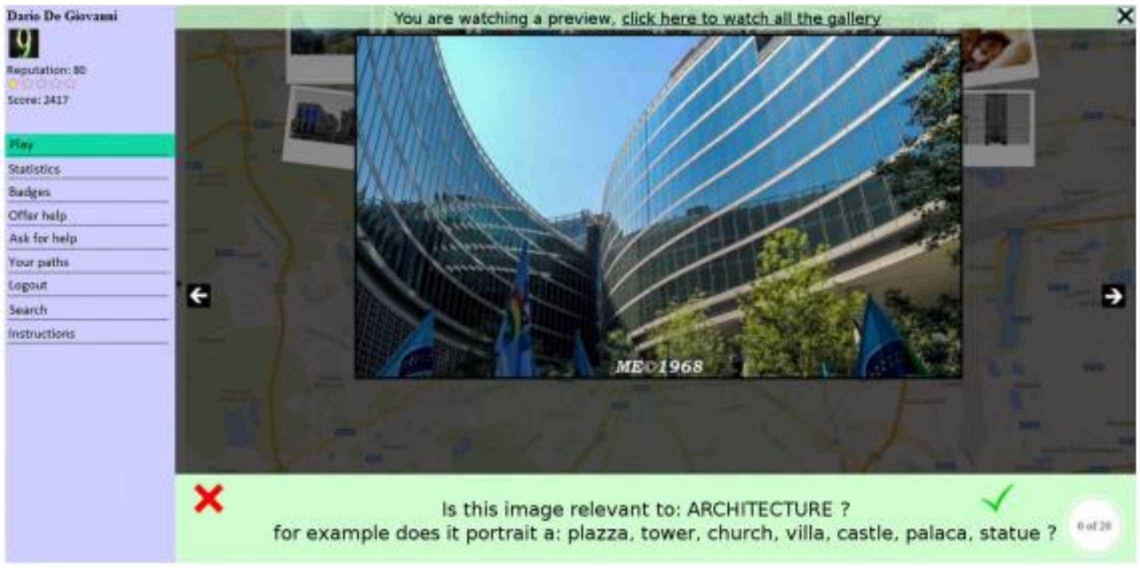
Onwards!

View Onboarding Tasks

Figure 1 - Examples of Gamification in Duolingo and Habitica

Gamification's previous use in photography:

There are a few previous use cases of gamification in the photography field. One of which is PhotoTrip a website for helping tourists find less popular cultural/interesting sites, whose creators state "In the gamified version of PhotoTrip, we adopt three most common gamification strategies—points, badges, and leaderboards (PBL). Points are provided as scores, corresponding to the measure of each quest the user is involved in and badges are awarded whenever certain thresholds are met."



You are watching a preview, click here to watch all the gallery

Is this image relevant to: ARCHITECTURE ?
for example does it portrait a: plazza, tower, church, villa, castle, palaca, statue ?

Figure 2 - Gamification being used via use of a voting system in PhotoTrip

(Bujari, et al, 2016).

The implementation of gamification in this website is a starting point however it is quite basic only utilising three gamification techniques. And whilst gamification might aid its user's engagement, it isn't the only factor, and unfortunately this website lacks other engaging factors. The app feels like it was designed with minimal thought towards aesthetic and a focus on functionality. The website appears to be designed purely for desktop, which reduces its usability by the very demographic it tries to serve, as tourists would likely prefer spontaneously checking the site during their trip rather than for preplanning.

Another use of Gamification in relation to photography was the Narva application which was used to find the location and vantage point from which historical photographs were taken as shown in figure 3.

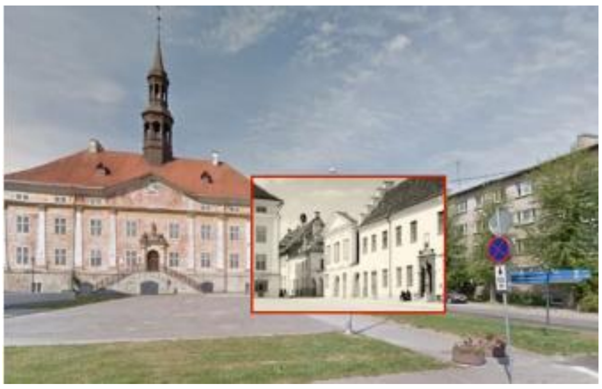


Figure 3 - An example of image positioning with Narva (Liestøl, 2018)

The Narva app's approach to gamification was not to follow traditional implementations such as badges, quests, e.t.c. But instead, was to use traditional games such as Hot & Cold and Jigsaw and to modify their rules to fit the context of the app as is shown by the following quotes:

- "The project or quest in the Hot & Cold game is comparable to that of the rephotographer who seeks to discover the original photograph's vantage point by comparing the current view with the old photograph."
- "the relationship between individual photos, and the space they record and depict, as well as their positions, quickly pointed to the jigsaw puzzle"

(Liestøl, 2018)

However, it is made clear that this approach wasn't as beneficial as they initially thought. They state, "Instead of searching among traditional analogue games for defining rules comparable to and shared by the activity in question (photography), we could have adapted more superficial strategies for the implementation of game elements, using extrinsic rewards like achievements and badges". This is likely due to the superficial human-nature, we see value in gaining something, in this case badges or achievements. We see little motivation to do something we are not being rewarded for.

Final Findings:

There are many varied and unique approaches to gamification, and it can be applied in a wide range of different forms, however, some forms are tried and tested and are almost guaranteed to be more engaging. These are badges/achievements, challenges, and leaderboards. This is likely due to a desire to be rewarded and compete, people are more likely to stick with something if there is a well-defined goal. Completing a challenge or earning an achievement provides a boost of dopamine, the effect of which is "well known for their strong responses to rewards and their critical role in positive motivation." (Bromberg-Martin, 2010)

2.2 HTTP Protocol

The system will partially consist of an API, this will be developed using Python and Flask. To successfully develop this an understanding of the HTTP Protocol will be required. In particular, understanding the HTTP verbs will be required.

These HTTP Verbs are defined by the HTTP Protocol (Request for comments, 1999) and the definitions are as follows:

GET:

> "The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process."

POST:

> "The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line."

PUT:

> "The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI. If a new resource is created, the origin server MUST inform the user agent via the 201 (Created) response. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request. If the resource could not be created or modified with the Request-URI, an appropriate error response SHOULD be given that reflects the nature of the problem. The recipient of the entity MUST NOT ignore any Content-* (e.g. Content-Range) headers that it does not understand or implement and MUST return a 501 (Not Implemented) response in such cases."

DELETE:

> "The DELETE method requests that the origin server delete the resource identified by the Request-URI. This method MAY be overridden by human intervention (or other means) on the origin server. The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully. However, the server SHOULD NOT indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location."

The API the system will use will be a CRUD style REST API. This means that each of the CRUD operations can be mapped to different HTTP verbs as follows.

Table 2 - CRUD to HTTP Mappings

CRUD	HTTP
CREATE	POST
READ	GET
UPDATE	PUT
DELETE	DELETE

These mappings fit the definitions of the HTTP verbs provided above.

2.3 Database Choice

There are many different options when it comes to databases. The main comparison to investigate will be between SQL vs NOSQL based databases.

SQL:

SQL stands for Structured Query Language and acts as the main interface between the database and the client. It was initially conceived in 1974 by IBM Researchers and was first standardized in 1986 Most SQL based database software are relational-database-management-systems (RDBMS). These databases consist of records, fields, relations and derived relvars. These can also be described using SQL terms:

Table 3 - RDBMS vs SQL Terms

Relation DB Term	SQL
Record	Row
Field	Column
Relation	Table
Derived Relvar	View

Groff (2002) States "SQL has become the standard database management language across a broad range of computer systems and application areas, including mainframes, workstations, personal computers, OLTP systems, client/server systems, data warehousing, and the Internet." This shows the importance of SQLs role in relational databases.

SQL is successful due to its early support from IBM, throughout the early 80s IBM pushed for the mainstream adoption of SQL, via the use of a commercialized product, SQL/Data System which was announced in 1981. In 1983 IBM announced a version of this system to run on their mainframe operating-system VM/CMS. Later in same year they announced DB2, which began shipping in 1985, DB2 ran on IBM's MVS operating system, which was used by large datacentres. Then in 1986 the first SQL standard ANSI SQL1 is ratified. In 1987 ISO follows and ratifies ISO SQL1. However, despite the existence of these standards many versions of SQL have small variations between each other.

NOSQL:

NOSQL is a "Next Generation Database Management Systems mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontally scalable.**

The original intention has been **modern web-scale database management systems**. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: **schema-free, easy replication support, simple API, eventually consistent / BASE** (not ACID), a **huge amount of data** and more. So the misleading term "nosql" (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above." (NOSQL, 2009).

NOSQL DBs address some of the limitations of relational databases, mainly scalability, complexity, complex querying, and feature bloat.

Scalability:

Relational databases can be hard to scale due to their design. The only ways to scale relational databases are to upgrade the hardware it is running on or distributing the database. However, "relational databases aren't designed to function with data partitioning, so distributing their functionality is a chore" (Leavitt, 2010)

Complexity:

Traditional DBs require all data to be put into tables however, not all data can be fit into tables. Because this data doesn't fit well with the relational DB structure, it can be hard to force unsuitable data to work with this format.

Complex Querying:

Relational DBs utilise SQL which although very strong for querying structured data, it struggles with other types as it is not designed to handle unsupported data. Utilising SQL also requires a significant amount of code making it more complex. "SQL can entail large amounts of complex code and doesn't work well with modern, agile development" (Leavitt, 2010).

Feature bloat:

"Relational data-bases offer a big feature set and data integrity. But NoSQL proponents say database users often don't need all the features, as well as the cost and complexity they add." (Leavitt, 2010).

Final Verdict:

Data Snap-Challenges will be handling is very structured so lends itself well to an SQL based server. For this reason, snap-challenges will use MYSQL. However, if demand requires it a hybrid-database architecture could be used by integrating an in-memory key value store such as Redis as a cache. "Redis is an in-memory remote database that offers high performance, replication, and a unique data model to produce a platform for solving problems." (Carlson, 2013).

2.4 React Native vs Progressive Web App

What is React Native?

"React Native is a framework for building native mobile apps in JavaScript using the React JavaScript library; React Native code compiles to real native components." (Dabit, 2019)

React native supports many platforms, some officially (Android and iOS), others via community-built tools which can be seen on the React Native web page covering Out-of-Tree Platforms (React Native (a), 2022) some of which include Windows Desktop, macOS, Android TV. On top of that react native also supports the web using react-native-web, and Desktop applications using Electron.

React native is open source however it is owned and maintained by Meta (previously known as Facebook). This raises concerns given Meta's history and less than trustworthy reputation. On the react native website they state "Facebook released React Native in 2015 and has been maintaining it ever since. In 2018, React Native had the 2nd highest number of contributors for any repository in GitHub." (React Native (b), 2022).

Benefits of React Native

React Native is faster than most other cross platform solutions because "React Native renders using its host platform's standard rendering APIs" (Eisenman, 2017) which differs from other applications that typically render HTML, JavaScript, and CSS instead of utilising the built-in components for the OS. This ensures a native feel when using an app developed with React Native.

This native approach has also allowed for lots of community development to provide solutions for easily communicating with existing features of the host device. An example of this is the expo-camera package (Expo, 2022) which provides easy access to the devices camera and is supported on Android, iOS, and the Web. Being able to access device feature like this is not possible with most other cross-platform tools, due to their HTML rendering style approach.

Another advantage React Native has is its familiarity, it is based on React which is one of the most popular web development tools, this means developers are already aware of most of the concepts used in React Native and most can easily adjust to using react native.

Drawbacks of React Native

The main disadvantage is that it adds a layer of complexity to a project, which inevitably makes it harder to debug. The use of JavaScript can also be problematic as it is a dynamically typed language and strong variable declarations are not required, this also makes debugging harder, however this can also be mitigated by using TypeScript which is also officially supported by react native (React Native (c), 2022).

Another potential drawback is identified in the Learning React Native Book (Eisenman, 2017) which states, "when updates are released for the host platform—say, a new suite of APIs in a new version of Android—there will be a lag before they are fully supported in React Native". This could lead to delays in support for new updates which could be displeasing for users.

What is a Progressive Web App?

"Progressive Web App (PWA) is a new generation of Web application designed to provide native app-like browsing experiences even when a browser is offline." (Kim, et al, 2018) PWAs can be installed by the click of a button from the web browser as shown in figure 4.

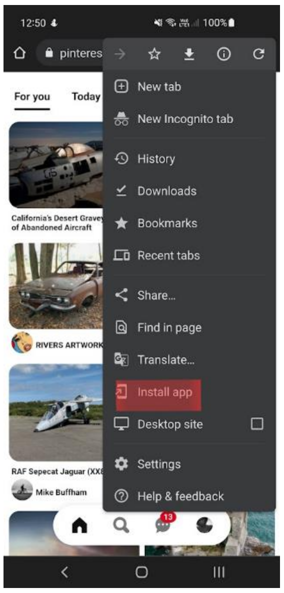


Figure 4 – PWA Install

A PWA can be created from any HTML code served via HTTPS and they can be implemented by including a web-manifest, icon, and service-worker with your app.

Benefits of PWAs

PWAs can be implemented from an existing website with extreme ease, shortening development time.

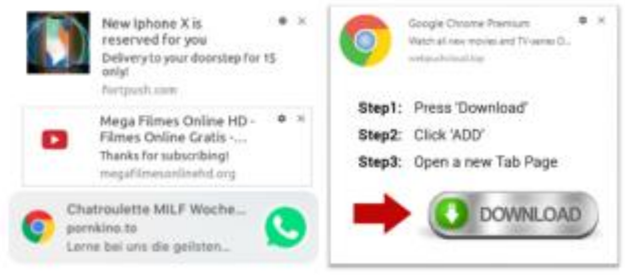
PWAs are also supported quite heavily by Google, and they can be listed on the play store directly meaning the user doesn't have to visit a website to download the app. Whilst also providing additional benefits such as those mentioned in Google's official article on listing a PWA on their store "Google Play also offers app ratings and reviews, giving users insight into your PWA before installing it." (Google, 2021).

Drawbacks of PWAs

Whilst PWAs may be widely supported by Google, Apple have a much more restrictive policy and do not allow them on the app store stating "Your app should include features, content, and UI that elevate it beyond a repackaged website. If your app is not particularly useful, unique, or "app-like," it doesn't belong on the App Store." (Apple, 2021).

PWAs don't have native access to device features. You cannot access hardware such as the camera from a PWA, limiting functionality of the app and constraining the developer.

PWAs do however let you send HTML5 notifications, unfortunately this also has a downside, these notifications have been associated with scams and phishing attempts some of which are shown in figure 5.



New Iphone X is reserved for you
Delivery to your doorstep for 1$ only!
fortpush.com

Mega Filmes Online HD - Filmes Online Gratis -...
Thanks for subscribing!
megafilmesonlinehd.org

Chatroulette MILF Woche...
pornkino.to
Lerne bei uns die geilsten...

Google Chrome Premium
Watch all new movies and TV-series O...

Step1: Press 'Download'
Step2: Click 'ADD'
Step3: Open a new Tab Page

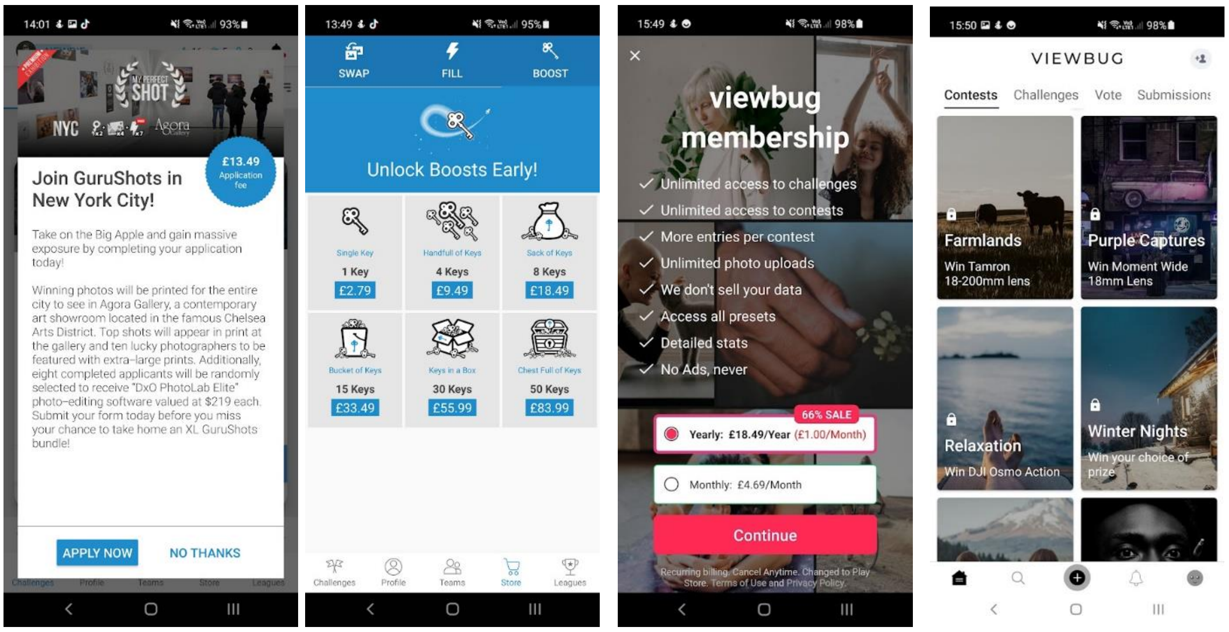
Figure 5 - Scam and Phishing Attempts (Kim, et al, 2018)

Final Verdict

Both React Native and Progressive Web Apps are suitable candidates for developing Snap-Challenges, however the extra versatility offered by React Native is clearly heavily beneficial. Being able to access the device hardware is crucial for a photography-based app, after all it is going to need access the device's camera.

2.5 Existing Solutions

There are a few different existing apps the main ones being GuruShots and ViewBug. However, these apps both seem to suffer from similar flaws. The main flaw being and extreme focus on a pay-to-win style of gamification and at times the apps almost feel like they solely exist as a quick cash and data grab scheme. Figure 4 shows some examples of features where these apps ask for payment:


Join GuruShots in New York City!

£13.49 Application fee

Take on the Big Apple and gain massive exposure by completing your application today!

Winning photos will be printed for the entire city to see in Agora Gallery, a contemporary art showroom located in the famous Chelsea Arts District. Top shots will appear in print at the gallery and ten lucky photographers to be featured with extra-large prints. Additionally, eight completed applicants will be randomly selected to receive "DxO PhotoLab Elite" photo-editing software valued at $219 each. Submit your form today before you miss your chance to take home an XL GuruShots bundle!

APPLY NOW NO THANKS

Unlock Boosts Early!

Single Key 1 Key £2.79	Handfull of Keys 4 Keys £9.49	Sack of Keys 8 Keys £18.49
Bucket of Keys 15 Keys £33.49	Keys in a Box 30 Keys £55.99	Chest Full of Keys 50 Keys £83.99

viewbug membership

- Unlimited access to challenges
- Unlimited access to contests
- More entries per contest
- Unlimited photo uploads
- We don't sell your data
- Access all presets
- Detailed stats
- No Ads, never

66% SALE Yearly: £18.49/Year (£1.00/Month)

Monthly: £4.69/Month

Continue

Recurring billing. Cancel Anytime. Changed to Play Store. Terms of Use and Privacy Policy.

VIEWBUG

Contests Challenges Vote Submissions

Farmlands Win Tamron 18-200mm lens

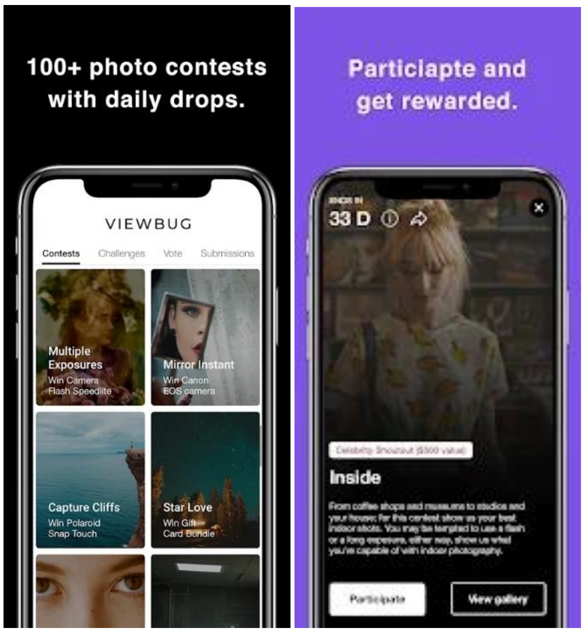
Purple Captures Win Moment Wide 18mm Lens

Relaxation Win DJI Osmo Action

Winter Nights Win your choice of prize

Figure 6 - Examples of Gurushots (left 2 images) and ViewBug (right 2 images) asking for payment.

Overall, Gurushots is less obtrusive with its microtransactions and membership, however ViewBug is almost unusable without a membership which costs 18.49 GBP per year. The rightmost image in figure 4 shows different contests, the small padlock above the contest name indicates that it requires this membership. No contents in the screenshot can be freely accessed, with a total of 21 free contests out of a total of 78 available contests at the time of writing. This means that roughly 73% of contests at the time of writing required membership. This does somewhat make sense as these contests do offer expensive prizes, however this practice is misleading as ads for ViewBug such as the one shown in figure 7 do not show any of these locked challenges and do not mention the membership cost.



100+ photo contests with daily drops.
VIEWBUG
Contests Challenges Vote Submissions
Multiple Exposures Win Camera Flash Speedlite
Mirror Instant Win Canon EOS camera
Capture Cliffs Win Polaroid Snap Touch
Star Love Win Gift Card Bundle
Particiapte and get rewarded.
33 D
Inside
Participate
View gallery

Figure 7 - ViewBug ads shown on Google Play (Google Play, 2013)

Both Gurushots and ViewBug do list camera metadata alongside images. However, this is hidden, deep in menus, of which most users wouldn't think to look through. This means it is ignored by all but the most curious of users and therefore is almost redundant.

Public opinion on these apps seems to mostly be mixed. One user on the r/photography subreddit posted the question "Your Take on Apps Like GuruShots and Other Photo Challenge Apps" (Anon, 2018) to which another user submitted this response

"

I like the 'idea' of gurushots and I like that you can see your ranking after the challenge is over.

*But it encourages people to just vote on every image they see to vote more/faster. (Or to vote on sh***y images because, maybe, they're less likely to get other votes)*

*I do think that truly 'great' images rise to the top, but I know when I decide to 'grind it out' and vote, I'm just voting yes as fast as possible - because NOT voting doesn't do s**t for me. Scrolling past an image is a waste of effort.*

If it was a tinder-like yes/no vote, and you got 'rewarded' for each one, I'd be more thoughtful with my votes.

" (Anon, 2018)

From this quote it is clear to see that photographers do enjoy the concept of a challenge-based photography app; however, it is also clear that they feel Gurushot's implementation requires too much dedication in order to succeed.

3 – Requirements

3.1 MoSCoW:

MoSCoW is a technique used to prioritise the requirements needed to reach a launch-ready product. This is done by splitting the requirements into the four following categories:

- **Must have (M)** – must be included at launch, without these components the product will not function as intended. If these are not met the launch should be seen as a failure.
- **Should have (S)** – requirements critical to the success of the system but not crucial for a minimum viable product.
- **Could have (C)** – Less critical requirements which are seen as a nice to have.
- **Won't have (W)** – the least critical requirements, which are not seen as worthwhile at the current time. However, may be worth returning to in the future.

These definitions have been adapted from "The project manager's guide to mastering agile" by (Cobb, 2015).

3.2 Functional Requirements:

Functional Requirements are the requirements used to clarify the functionality of the product. These requirements have been split into sub-sections for each component of the app.

DATABASE:

Table 4 - DB Functional Requirements

ID	DESCRIPTION	MoSCoW
FR-1	A Badges Table	S
FR-2	A Challenges Table	M
FR-3	A Challenges Has Posts Table	M
FR-4	A Countries Table	M
FR-5	A Photos Table	M
FR-6	A Posts Table	M
FR-7	A Users Table	M
FR-8	A User Has Badges Table	S
FR-9	A User Has Challenges Table	M
FR-10	A User Has Posts Table	M

API:

*Table 5 - AP*I Functional Requirements

ID	DESCRIPTION	MoSCoW
FR-15	**Register Endpoint** – Handles a new user POST Request sent to the API. Passwords sent must be hashed and salted with sha256 encryption.	M

	Returns: • 201 and JSON containing user data if completed successfully. • 400 and relevant message if the user already exists or no data was provided to the endpoint.	
FR-16	**Login Endpoint** – Handles a get request sent to it and returns a JSON Web Token (JWT) if the login is valid. **Returns:** • 200 and the normal JWT as well as a refresh token JWT if completed successfully. • 401 and relevant message if the user can't be verified.	M
FR-17	**Refresh Token** – Will be called whenever the JWT is due to expire, will return a new token. A refresh token will need to be given. **Returns:** • 200 and a new JWT if completed successfully. • 401 if the refresh token given is invalid.	S
FR-18	**Users Endpoint** – Used for data about the user. Accepts GET/PUT and DELETE requests. **GET returns:** • 200 and user data if successful. • 404 if the user data can't be found. **PUT returns:** • 204 and user data if successful • 404 if user not found. **DELETE returns:** • 204 if successful • 404 if not found.	M

CDN:

Table 6 - CDN Functional Requirements

ID	DESCRIPTION	MoSCoW
FR-29	Static assets stored on the server.	M
FR-30	Static assets available via a URL.	M
FR-32	An upload system that allows images to be uploaded to the cdn.	M

React Native App:

Table 7 - React Native App Functional Requirements

ID	DESCRIPTION	MoSCoW
FR-33	A login page.	M
FR-34	A sign-up page.	M
FR-35	A challenges page.	M

FR-36	A profile page.	M
FR-37	A badges page.	S
FR-38	A post page.	M

Login Page:

Table 8 - Login Page Functional Requirements

ID	DESCRIPTION	MoSCoW
FR-40	A login button which when clicked will call the API to validate the login.	M
FR-41	A sign-up button which will redirect the user to the signup page.	M

Sign-up page:

Table 9 - Sign-up Page Functional Requirements

ID	DESCRIPTION	MoSCoW
FR-44	A sign-up button which when clicked calls the register api endpoint with data provided in textboxes above by the user.	M
FR-45	A date picker which utilises the android default date picker – on web it will just show a text input.	M
FR-46	A dropdown list for selecting countries.	M
FR-47	After successful sign-up the user will be redirected to the login page.	M

Challenges Page:

Table 10 - Challenges Page Functional Requirements

ID	DESCRIPTION	MoSCoW
FR-49	A list of active challenges (those whose end date is in the future) will be displayed to the user pulled from the API.	M
FR-50	The user will be able to click on a challenge and see all the posts for that given challenge.	M
FR-51	The user should be able to click on a join button next to a challenge and be able to upload a post to said challenge.	M

Profile Page:

Table 11 - Profile Page Functional Requirements

ID	DESCRIPTION	MoSCoW
FR-55	User data will be displayed, including avatar, name, country flag, and bio.	M
FR-58	Number of posts shown.	M
FR-59	Total kudos received shown.	M
FR-60	All of the users' posts should be shown.	M
FR-62	When a post is clicked it should take you to the correct post page.	M

Badges Page:

Table 12 - Badges Page Functional Requirements

ID	DESCRIPTION	MoSCoW
FR-64	A list of all badges available should be shown.	S

Post Page:

Table 13 - Post Page Functional Requirements

ID	DESCRIPTION	MoSCoW
FR-67	The image must be shown in its original aspect ratio. Apart from strange non-standard aspect ratios, such as incredibly tall images.	M
FR-68	Camera settings should be loaded from the post data.	M
FR-69	Kudos and Downvotes should be shown.	M

3.3 Non-Functional Requirements

Database:

Table 14 - Database NFRs

ID	DESCRIPTION	MoSCoW
NFR-1	An SQL query to the database should be executed within a second.	S

API:

Table 15 - API NFRs

ID	DESCRIPTION	MoSCoW
NFR-2	Any API call should respond within 3 seconds.	M
NFR-3	The API will be accessible from an external network.	M

CDN:

Table 16 - CDN NFRs

ID	DESCRIPTION	MoSCoW
NFR-7	The CDN will be accessible from an external network.	M

React Native App:

Table 17 - React Native App NFRs

ID	DESCRIPTION	MoSCoW
NFR-8	Pages should load on average within 3 seconds on a 4g throttled connection.	M

Post Page:

Table 18 - Post Page NFRs

ID	DESCRIPTION	MoSCoW
NFR-11	Camera info such as focal length must be clearly visible on the post page if it is available.	M

THIS IS A BRIEF SELECTION OF THE REQUIREMENTS MORE CAN BE FOUND IN APENDIX A.

4 – Methodology

This project will use an agile style methodology, agile focusses on the following values defined in the Manifesto for Agile Software Development (Beck, et al, 2001):

"

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

"

These values are upheld to ensure projects are completed on-time and in a functional capacity. This is done by following the 12 agile principles defined by the Agile Alliance (Agile Alliance, 2015)

"

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

"

Agile itself has many different methodologies which follow the values and principles stated above. A few examples of these would be:

- Extreme
- SCRUM
- Feature Driven Development

The chosen methodology for this project is Feature Driven Development (FDD)

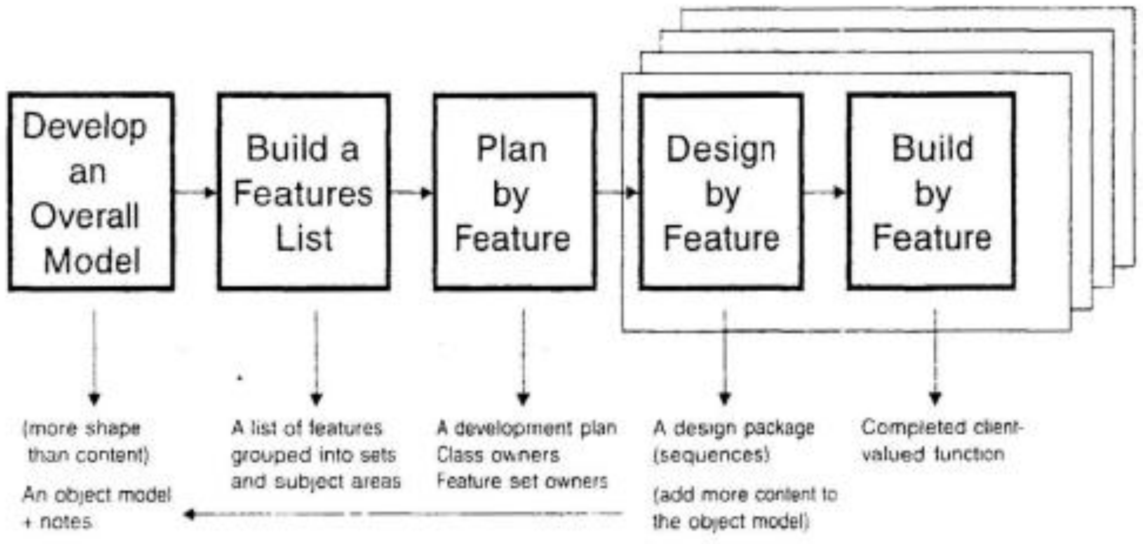

Develop an Overall Model
Build a Features List
Plan by Feature
Design by Feature
Build by Feature
(more shape than content)
An object model + notes
A list of features grouped into sets and subject areas
A development plan
Class owners
Feature set owners
A design package (sequences)
(add more content to the object model)
Completed client-valued function

Figure 8 - Feature Driven Development (Felsing, Palmer, 2002)

Using A FDD approach ensures functionality of the app is completed by focussing on the features. More important features are prioritized as can be seen in the requirements.

5 – Design

5.1 – System Architecture:

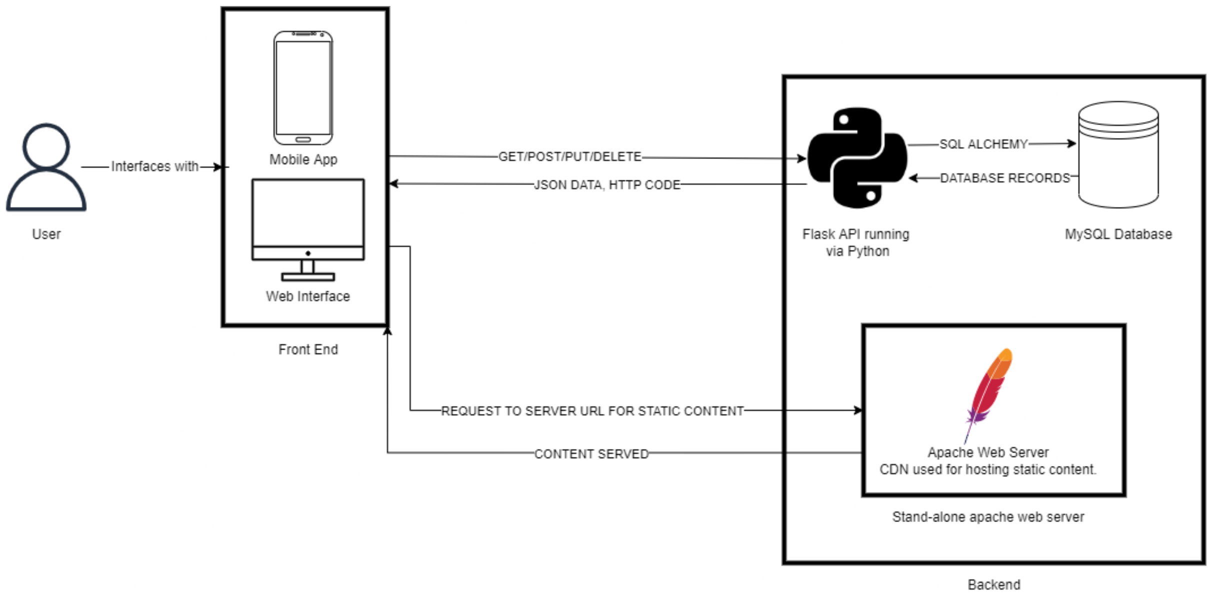


Figure 9 - System Architecture Diagram

Figure 9 shows the architecture of the Snap-Challenges system. The system can be broken down into two main components, a front-end and a back-end, these components can then be subdivided into parts. The front-end consists of client apps built utilising react native and expo to allow for one single codebase for multiple platforms. The back-end can be broken into 3 core components, A database for storage, an API to access the data and a web server to host the static content.

5.2 – Database Design:

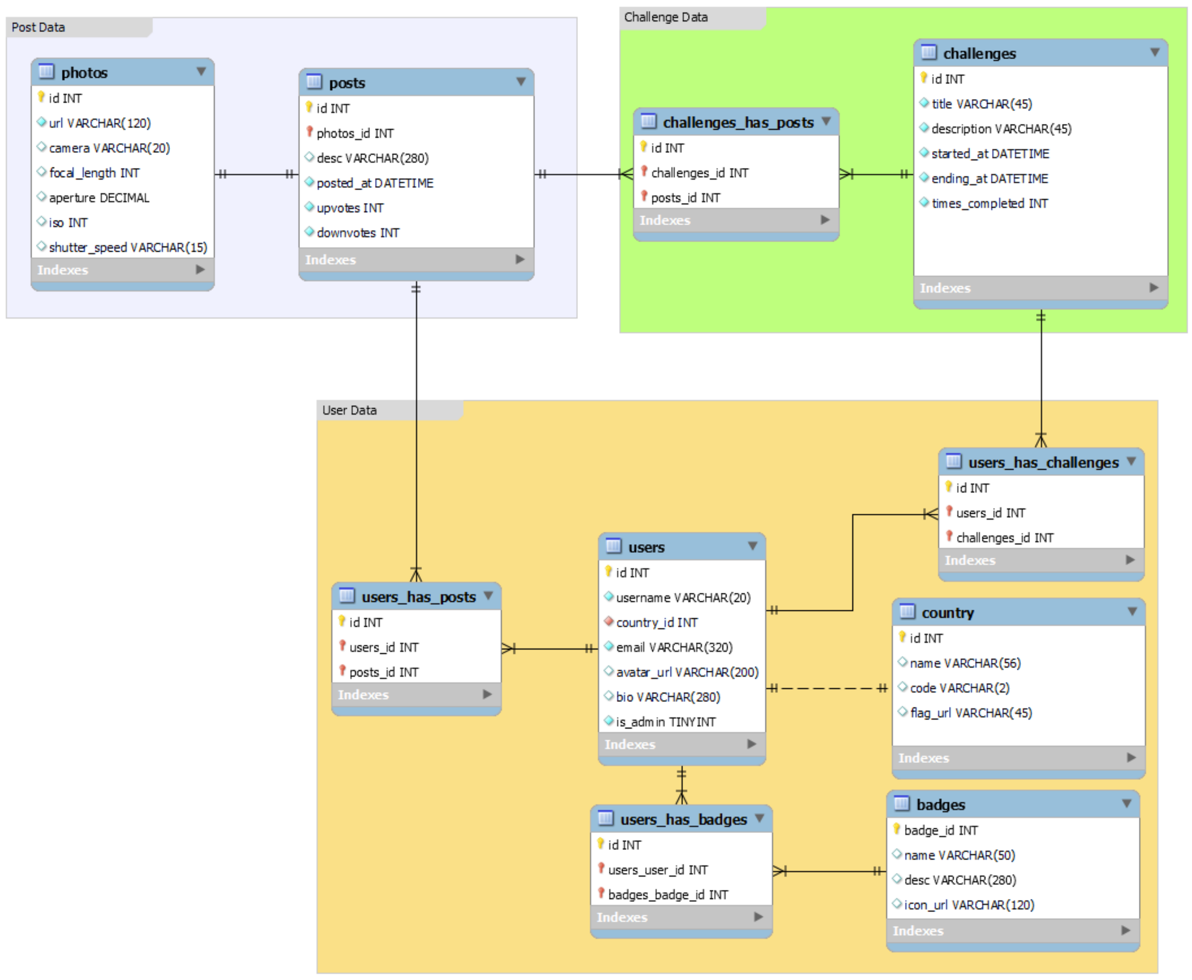


Figure 10 - Entity Relationship Diagram

Figure 10 shows the database tables and how they are linked. The tables are grouped into three separate sections. These are post-data, challenge-data and user-data.

- Post-data contains all the information relevant to posts as well as information about the photo for the post.
- Challenge-data stores all the information about challenges such as challenge info and what posts have been submitted for each challenge.
- User-data holds all info about the users of the system, what posts they've made, what badges they've earnt, what challenges they've joined and information about their country.

5.3 – API Design

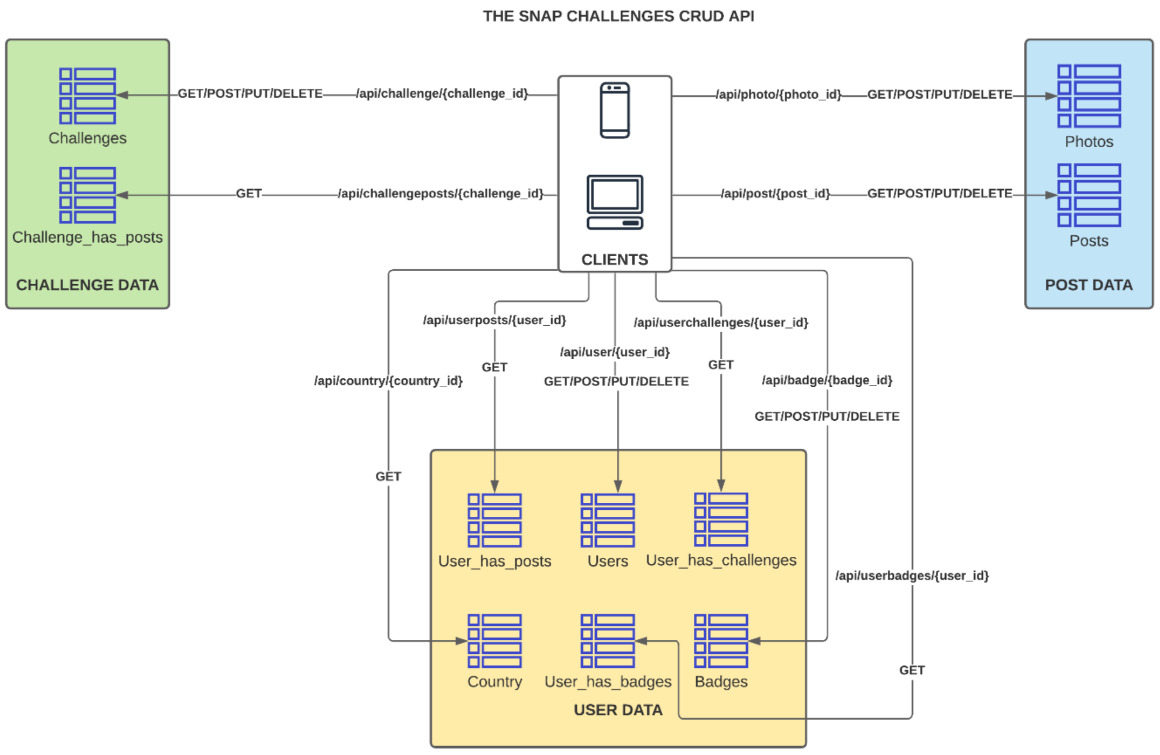


Figure 11 - API Design Diagram

Figure 11 shows the design for the API layer. It consists of different endpoints to be developed in Flask and follows a CRUD style approach based off research conducted in the Literature Review. The clients will interact with the API in a way that is obfuscated to them simply interacting with it via button presses and interactions on the app.

5.4 – User Interface Design:

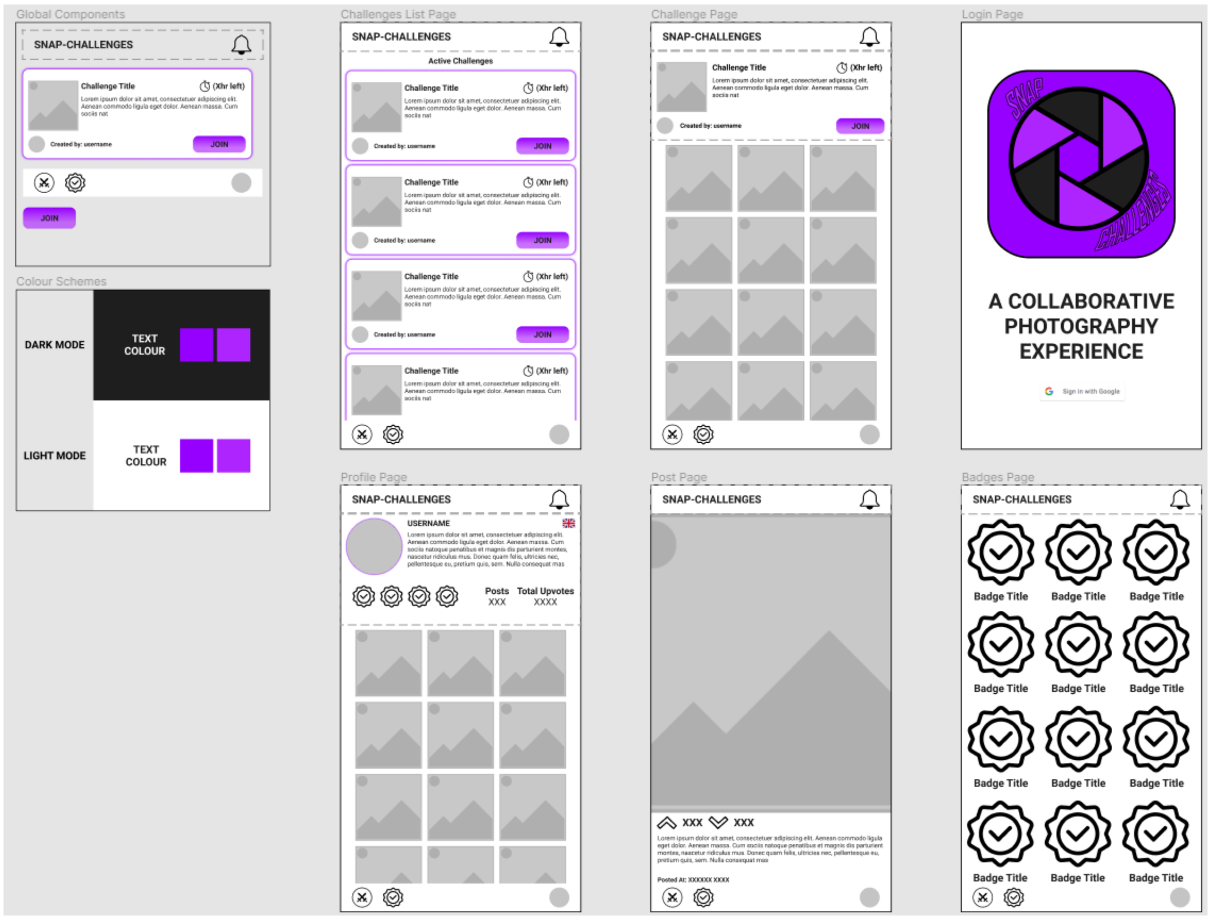


Figure 12 - User Interface Design Composites

Figure 12 shows a general overview of the composites for each page of the app. In total there 6 pages.

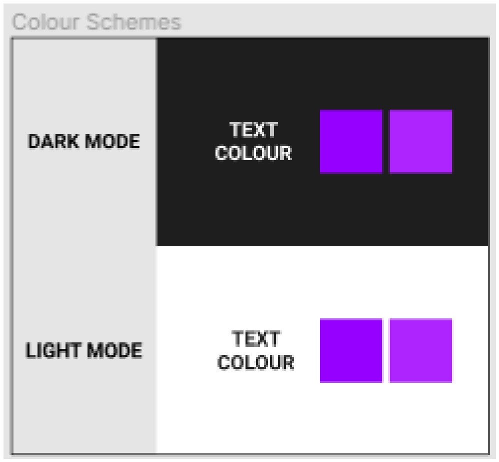


Figure 13 - Colour Schemes

Snap-Challenges should support both light and dark modes as stated in the requirements (FR-39), for this reason separate colour-schemes had to be considered for each setting. The app will use 4 separate colours:

- A backdrop-colour.
- A text-colour.
- A contrasting colour.
- An alternate contrasting colour.

After deliberation It was decided that the colour schemes shown in Figure 13 would be used.

These are as follows:

1. Dark Mode
    a. Backdrop: #1E1E1E
    b. Text: #FFFFFF
    c. Contrast 1: #9500FF
    d. Contrast 2: #AE24FF
2. Light Mode
    a. Backdrop: #FFFFFF
    b. Text: #1E1E1E
    c. Contrast 1: #9500FF
    d. Contrast 2: #AE24FF

These colour schemes allow for an interchangeable design as they maintain the same contrast colours, and simply invert the backdrop and text colours.

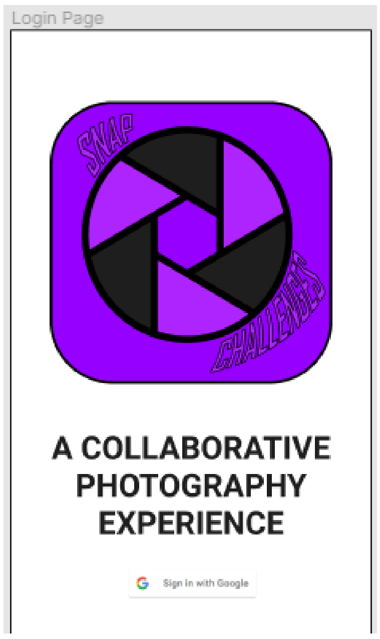


Figure 14 is the composite for the login page, it is the first page and therefore acts as an introduction to the app. For this reason, it will display the Logo of the app as well as a brief text description, both large and visible.

Figure 14 - Login Page

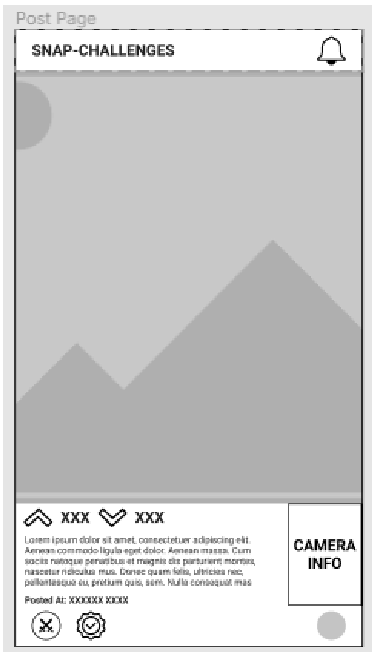


Figure 15 - Post Page

Figure 15 is the initial design for the post page. The post page will be able to show any image of any aspect ratio (within reason) inside the image box to meet FR-67. This design clearly shows the user info about the post and camera settings, kudos and downvotes are clearly visible aswell as a description.

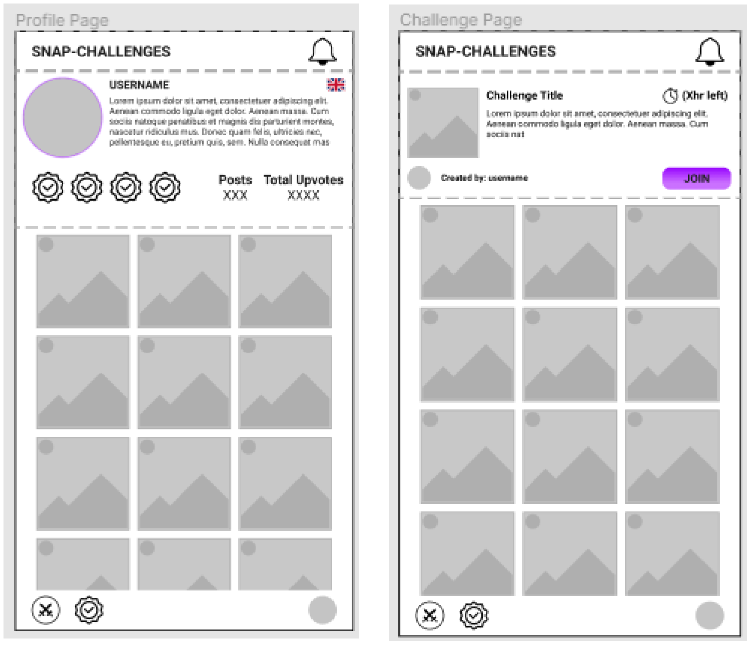


Figure 16 – Profile and Challenge Pages

Figure 16 shows both the profile and challenge pages, they have been grouped as they are very similar. The top of the pages shows info about the user/challenge and the bottom shows a global component image grid. In react native a global component is one which in reused multiple times.

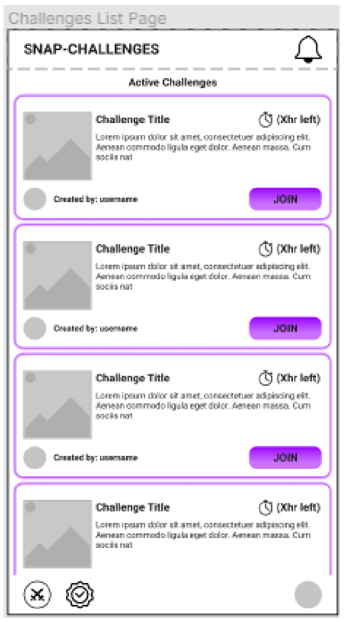

Challenges List Page
SNAP-CHALLENGES
Active Challenges
Challenge Title
(Xhr left)
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis nat
Created by: username
JOIN

Figure 17 - Challenge List Page

Figure 17 shows the challenge list page, this page will get and display all currently active challenges in a list, displaying info such as time left on a challenge card. When clicked a card will go to the challenges page as shown in figure 16. There will also be a join button on the card which will allow users to participate in the challenge.

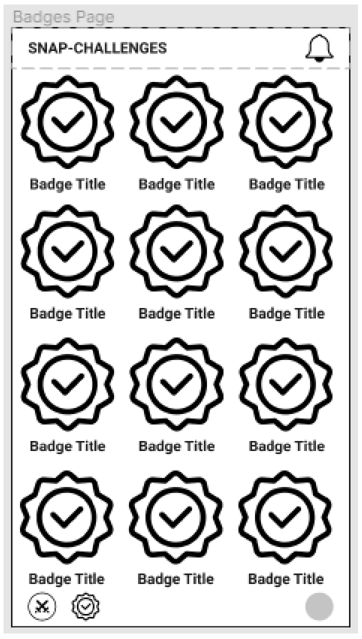


Figure 18 - Badge Page

Figure 18 is the badge page it will display all possible badges distinguishing between those a user has unlocked and those that they are yet to unlock.

5.5 – Test Design

Table 19 - Test Design

Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed?
X	N/A	System is in Y state.	System outputs Z	System outputted Z	Pass

Table 19 shows how Snap-Challenges will be tested. Each test case will test a specific capability of the app and often relate to some of the requirements, if this is the case requirement ID numbers will be provided. Tests will specify preconditions at the point of the test. For example, if a user is logged in. We then specify the result expected and the actual result. If these match the test is marked as passed, if they don't the case is marked as failed.

6 – Implementation

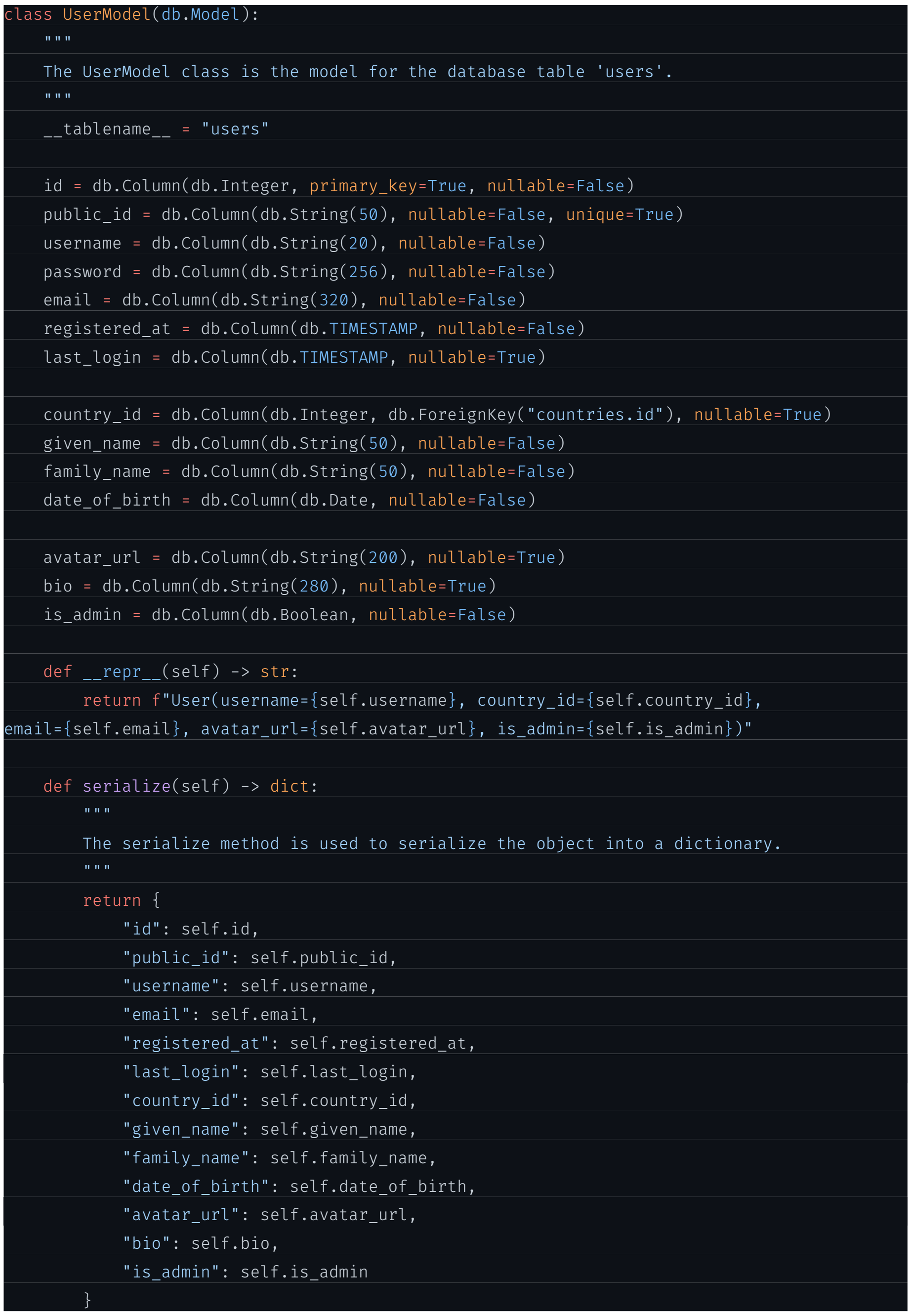
6.1 – Database

6.1.1 – SQLite consideration

During the early stages of development, SQLite was also considered however, was not picked as "SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process." (SQLite, 2022). This would've been problematic for Snap-Challenges as data needs to be shared via multiple clients therefore having a local datastore would not be an option.

6.1.2 – SQLAlchemy & DB Models

The database communicates with the API using the SQL alchemy library for Flask which "aims to simplify using SQLAlchemy with Flask by providing useful defaults and extra helpers that make it easier to accomplish common tasks." (Flask-SQLAlchemy, 2021). SQL Alchemy allows for SQL commands to be obfuscated and run via python functions. Database tables are interpreted as Model Classes an example of which can be seen bellow:



```
class UserModel(db.Model):
    """
    The UserModel class is the model for the database table 'users'.
    """
    __tablename__ = "users"

    id = db.Column(db.Integer, primary_key=True, nullable=False)
    public_id = db.Column(db.String(50), nullable=False, unique=True)
    username = db.Column(db.String(20), nullable=False)
    password = db.Column(db.String(256), nullable=False)
    email = db.Column(db.String(320), nullable=False)
    registered_at = db.Column(db.TIMESTAMP, nullable=False)
    last_login = db.Column(db.TIMESTAMP, nullable=True)

    country_id = db.Column(db.Integer, db.ForeignKey("countries.id"), nullable=True)
    given_name = db.Column(db.String(50), nullable=False)
    family_name = db.Column(db.String(50), nullable=False)
    date_of_birth = db.Column(db.Date, nullable=False)

    avatar_url = db.Column(db.String(200), nullable=True)
    bio = db.Column(db.String(280), nullable=True)
    is_admin = db.Column(db.Boolean, nullable=False)

    def __repr__(self) -> str:
        return f"User(username={self.username}, country_id={self.country_id},
email={self.email}, avatar_url={self.avatar_url}, is_admin={self.is_admin})"

    def serialize(self) -> dict:
        """
        The serialize method is used to serialize the object into a dictionary.
        """
        return {
            "id": self.id,
            "public_id": self.public_id,
            "username": self.username,
            "email": self.email,
            "registered_at": self.registered_at,
            "last_login": self.last_login,
            "country_id": self.country_id,
            "given_name": self.given_name,
            "family_name": self.family_name,
            "date_of_birth": self.date_of_birth,
            "avatar_url": self.avatar_url,
            "bio": self.bio,
            "is_admin": self.is_admin
        }
```

Each model class represents a database table, each model class also includes a __repr__ method which prints out a text representation of the object. There is also a serialize function which returns the object as a dictionary (JSON Format). There are model classes for each table of data stored on the db.

6.1.3 – Changes to the database structure

During development of these model classes, it was found that the system could benefit from more information in comparison to that of the original database design, seen in figure 10.

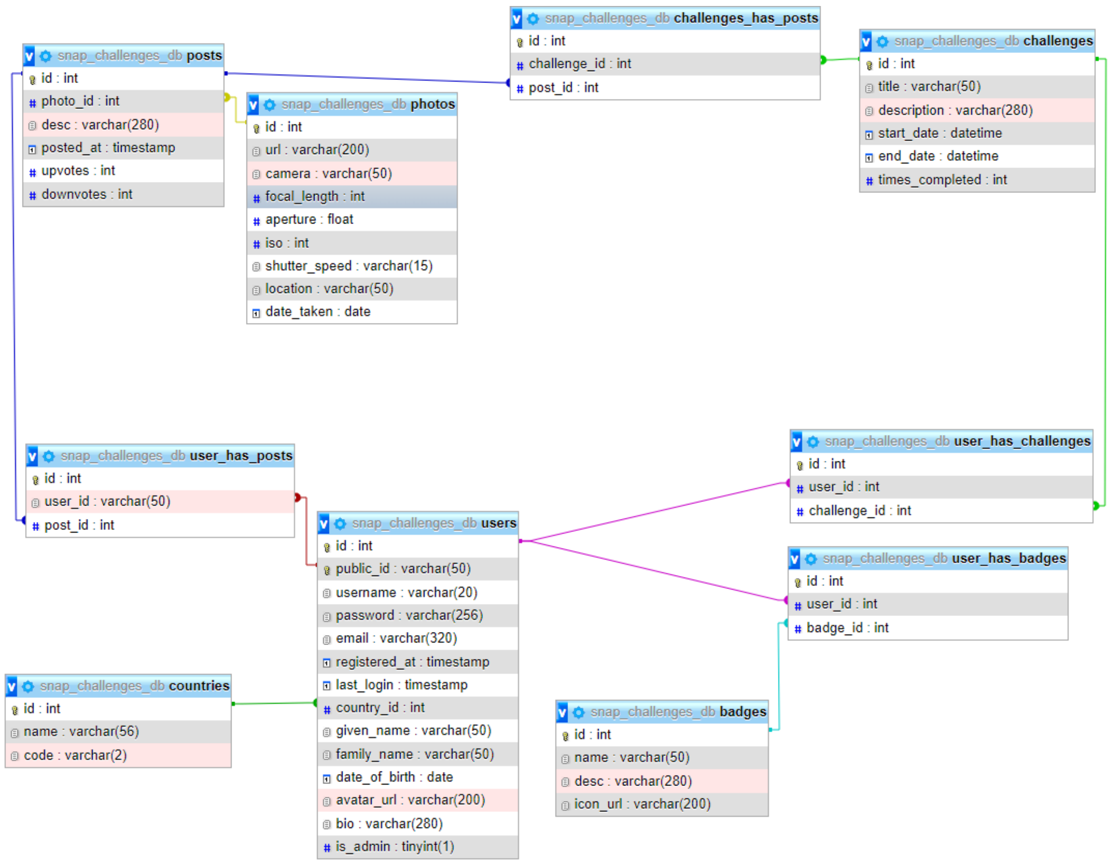


Figure 19 - Updated DB ER Diagram

Figure 19 shows the updated ER which includes extra info. There are no new tables however, the photos, users, and countries tables have been modified.

The photos table now has two new columns the location and date a shot was taken. This data is optional and can be NULL.

The users table now stores a wide array of new information used to handle the login system. The reason behind this drastic change was due to the decision to switch from using Google Authentication, to a custom-built authentication system, this was done to allow for JSON web token security. Usernames and Passwords are now stored in the database and are hashed and salted using the SHA256 algorithm. This is done for security reasons so even if the database was breached user passwords wouldn't be exposed. Info about the date registered and last login is also stored to allow better tracking of user's engagement with the app.

Finally, flag_url was removed from the countries table as it was redundant, due to flags being stored locally on the client using iso codes.

6.2 – API

6.2.1 – API V1

The initial implementation of the API was developed utilising flask restful which is "an extension for Flask that adds support for quickly building REST APIs." (Flask-RESTful, 2020). This allowed for user resource classes which can then be marshalled by resource fields which can be used to validate data sent to the API and return relevant error messages. This simplicity was initially beneficial and allowed for a first implementation of the API to be completed within a very short period. However, this simplicity also became an issue that prevented the API from being secured utilising JSON web token (JWTs). To implement JWTs with flask restful another extension would need to be implemented, Flask-JWT-Extended which “not only adds support for using JSON Web Tokens (JWT) to Flask for protecting routes, but also many helpful (and optional) features built in to make working with JSON Web Tokens easier.” (vimalloc, 2022). Whilst this was certainly an option, the decision was made to not utilise this extension as it added a further layer of complexity to the system. However, this did mean that flask restful also couldn’t be used. As a result, a second implementation of the API was required. This implementation uses flasks default @app.route decorator to handle the HTTP verbs for each URL.

6.2.2 – Tokenization

The API is secured with JSON Web Token (JWT) authentication, this caused many issues with the implementation of the API, some of which have already been mentioned.

One of the biggest issues was that the original design didn’t consider the use of these JWTs, and didn’t include any API endpoints for handling tokens. Handling tokens requires one for registering the user, one for when a user logs in and is given a token, and a final one for refreshing a token when it expires.

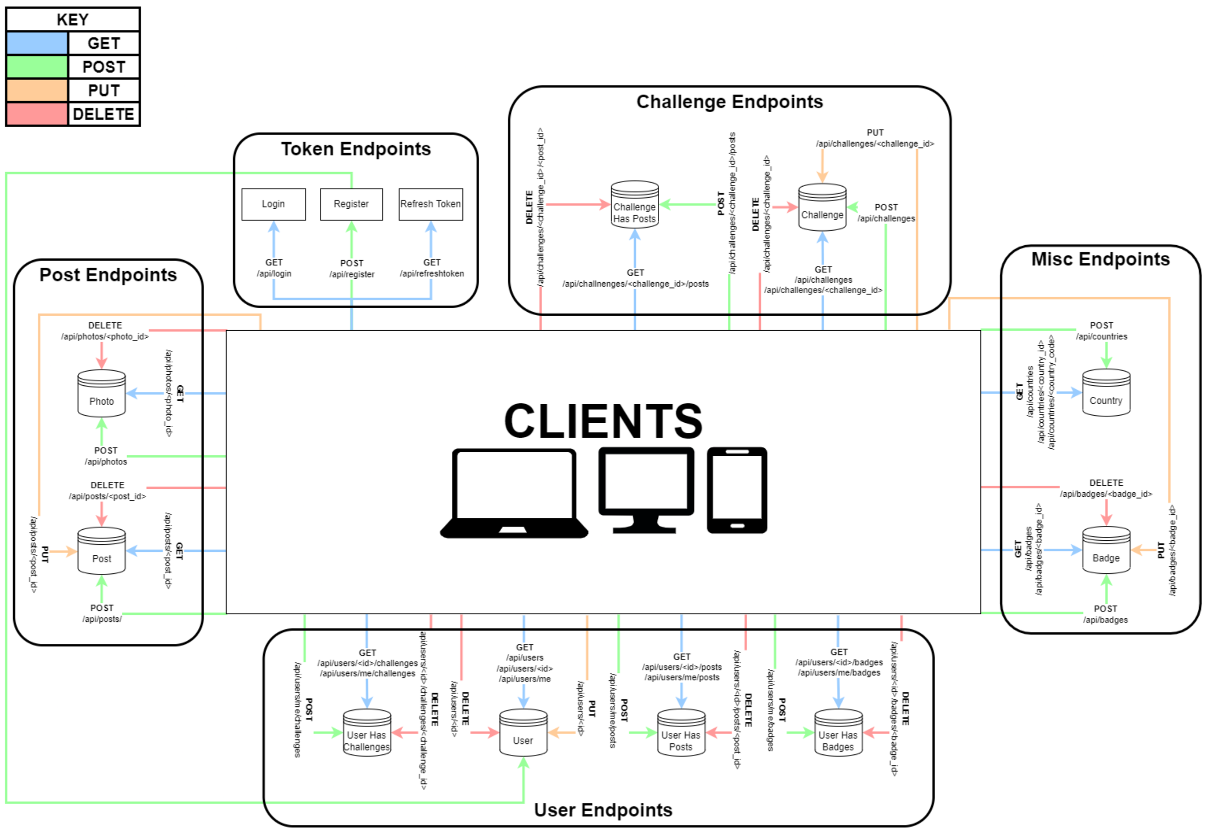

KEY
🟦 GET
🟩 POST
🟧 PUT
🟥 DELETE
Token Endpoints
Login
Register
Refresh Token
GET /api/login
POST /api/register
GET /api/refreshtoken
Challenge Endpoints
DELETE /api/challenges/<challenge_id>/<post_id>
Challenge Has Posts
POST /api/challenges/<challenge_id>/posts
GET /api/challnenges/<challenge_id>/posts
DELETE /api/challenges/<challenge_id>
PUT /api/challenges/<challenge_id>
Challenge
POST /api/challenges
GET /api/challenges /api/challenges/<challenge_id>
Post Endpoints
DELETE /api/photos/<photo_id>
Photo
GET /api/photos/<photo_id>
POST /api/photos
DELETE /api/posts/<post_id>
PUT /api/posts/<post_id>
Post
GET /api/posts/<post_id>
POST /api/posts/
CLIENTS
Misc Endpoints
POST /api/countries
GET /api/countries /api/countries/<country_id> /api/countries/<country_code>
Country
DELETE /api/badges/<badge_id>
GET /api/badges /api/badges/<badge_id>
Badge
PUT /api/badges/<badge_id>
POST /api/badges
User Endpoints
POST /api/users/me/challenges
GET /api/users/<id>/challenges /api/users/me/challenges
User Has Challenges
DELETE api/users/<id>/challenges/<challenge_id>
DELETE /api/users/<id>
GET /api/users /api/users/<id> /api/users/me
User
PUT /api/users/<id>
POST /api/users/me/posts
GET /api/users/<id>/posts /api/users/me/posts
User Has Posts
DELETE /api/users/<id>/posts/<post_id>
POST /api/users/me/badges
GET /api/users/<id>/badges /api/users/me/badges
User Has Badges
DELETE /api/users/<id>/badges/<badge_id>

Figure 20 - Updated API Design Diagram

Figure 20 shows the updated API design which includes these new endpoints.

Tokens are generated on account login with the following route:

```
@app.route("/api/login")
def login():
    auth = request.authorization

    if not auth or not auth.username or not auth.password:
        return make_response('Could not verify', 401, {'WWW-Authenticate' : 'Basic 
realm="Login required!"'})

    user = UserModel.query.filter_by(username=auth.username).first()

    if not user:
        return make_response('Could not verify', 401, {'WWW-Authenticate' : 'Basic 
realm="Login required!"'})

    if check_password_hash(user.password, auth.password):
        token = jwt.encode(
            {
                'public_id' : user.public_id,
                'scope' : 'user',
                'exp' : datetime.datetime.utcnow() + datetime.timedelta(days=12)
            },
            app.config['SECRET_KEY']
        )

        refresh_token = jwt.encode(
            {
                'public_id' : user.public_id,
                'scope' :  'refresh',
                'exp' : datetime.datetime.utcnow() + datetime.timedelta(days=60)
            },
            app.config['SECRET_KEY']
        )

        ts = time.time()
        timestamp = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
        user.last_login = timestamp

        db.session.commit()

        return jsonify({'token' : token.decode('utf-8'), 'refreshToken': 
refresh_token.decode('utf-8')}), 200

    return make_response('Could not verify', 401, {'WWW-Authenticate' : 'Basic realm="Login 
required!"'})
```

The JWTs are created using the PyJWT library installed via the python pip package manager. This library is used for encoding and decoding these JWTs.

When a login request is sent to the API via the api/login endpoint the login is first validated, if it is a valid login two JWTs are generated, one for use for validation and one for refreshing and getting a new token. The token used for validation is given the user scope which has access to the whole API, due to this it is given a shorter expiration time, this is done for security reasons as if the token is exposed it can essentially be used to falsify being the account holder and retrieve information from the API. The refresh token has a much longer expiration time as it is restricted to only being able to access one API endpoint, the refresh endpoint, this token's whole purpose is to simply be used to generate a new validation token once the current token expires. On top of this basic scope information the tokens also contain the user's public id which is used to validate which account is related to the token.

To validate tokens the following python decorator was created:

```python
def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None

        if 'x-access-token' in request.headers:
            token = request.headers['x-access-token']

        if not token:
            return jsonify({'message': 'Token is missing'}), 401

        try:
            data = jwt.decode(token, app.config['SECRET_KEY'])
            #Get the scope from the token data
            scope = data['scope']

            #Check if the scope is valid
            if scope == 'user':
                current_user =
UserModel.query.filter_by(public_id=data['public_id']).first()
            else:
                return jsonify({'message': 'Invalid scope'}), 401

        except:
            return jsonify({'message': 'Token is invalid'}), 401

        return f(current_user, *args, **kwargs)

    return decorated
```

This decorator will check if a request to the decorated function contains the x-access-token header. If a token is present it is decoded by the PyJWT package to verify it is a token that was created by Snap-Challenges, this is done by using the apps secret key to decrypt the token. After the token is decoded, we can access the data stored, such as the user's public id and scope. The scope is used to validate the token given is a user token and not a refresh token. The public id is used to obtain the current user's data.

6.2.3 – CORS Prefilght

When developing the client app, a problem was discovered with the API, whenever a HTTP request was sent to API endpoints the JavaScript would fail and return the error, "Access to x has been blocked by cors policy". This was initially a surprise as when testing the API in python and via Postman there were no reported errors. However, after some research, it was discovered that this error was being caused by the JavaScript fetch method which first sends an OPTIONS request to the endpoint. This OPTIONS request is essentially a verification handshake process to ensure the security of the API and is sent out by default by most devices. A simple diagram of this request is shown below in figure 21.

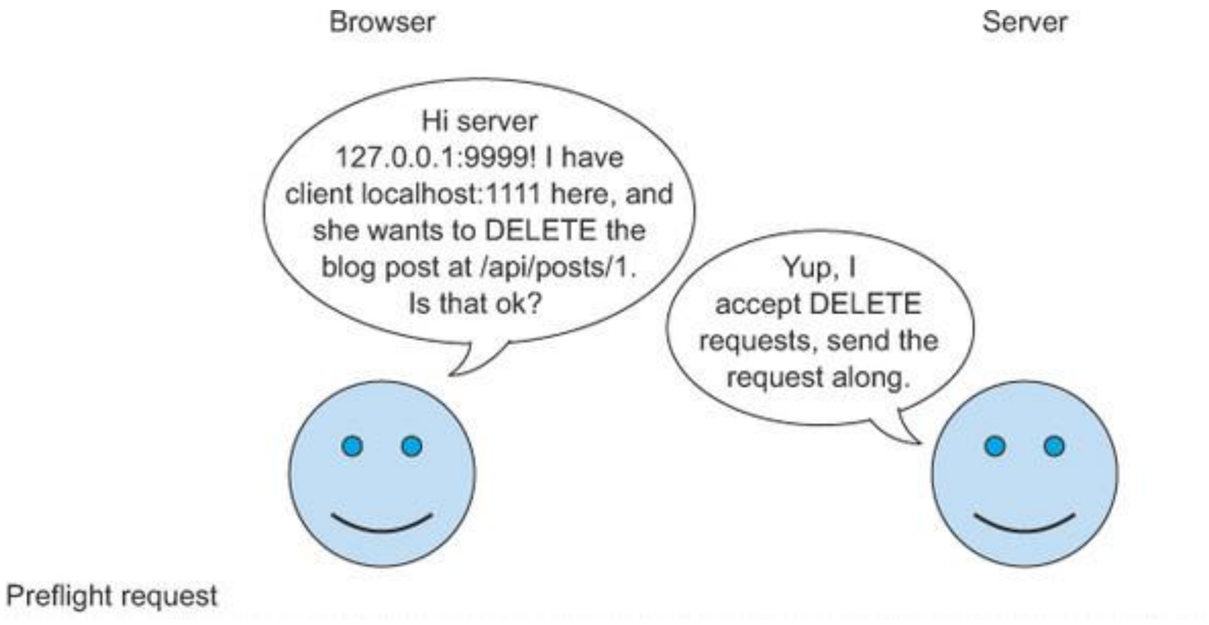

Browser
Server
Hi server 127.0.0.1:9999! I have client localhost:1111 here, and she wants to DELETE the blog post at /api/posts/1. Is that ok?
Yup, I accept DELETE requests, send the request along.
Preflight request

Figure 21 – An OPTIONS pre-flight request (Hossain, 2014)

After more research it was discovered that there are multiple ways of resolving this conundrum, disabling CORS, handling these requests on the API side, or using the Flask-CORS python package to handle this.

Disabling CORS would've been the easiest method to resolve the issue, it would've been as simple as specifying the mode as no-cors in the request, whilst this would've resolved the issue it was a non-solution as doing so avoids the larger problem, CORS requests are sent for security reasons to verify origins and headers sent to the API. So "fixing" it by disabling the CORS doesn't resolve the larger issue and was not the solution implemented.

Flask-CORS describes itself as "A Flask extension for handling Cross Origin Resource Sharing (CORS), making cross-origin AJAX possible.

This package has a simple philosophy: when you want to enable CORS, you wish to enable it for all use cases on a domain. This means no mucking around with different allowed headers, methods, etc." (Flask-CORS, 2022). This certainly would've been a valid and potentially easier solution, but this was also not implemented as it was a bit over complex for the use case, we only needed to allow all origins and some extras headers such as x-access-token.

That left one option implementing our own solution to handle these OPTIONS requests. This was quite simple and only required 6 lines of code extra.

```
@app.after_request
def after_request(response):
    header = response.headers
    header['Access-Control-Allow-Origin'] = '*'
    header['Access-Control-Allow-Headers'] = 'Content-Type,Authorization,X-Access-Token'
    return response
```

This function uses flasks after_request decorator to run this function after each request, the code inside the function handles the CORS pre-flight by generating a response with the headers and origins allowed to communicate with the API.

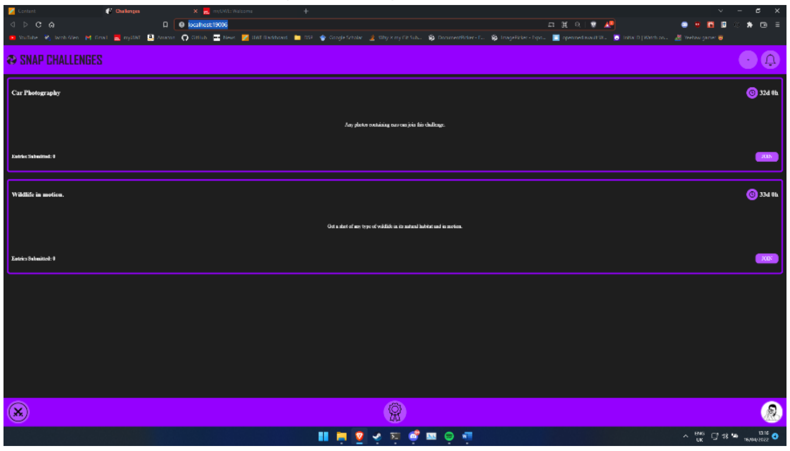
6.3 – Apache2 CDN Web Server

In the architecture of Snap-Challenges the web server plays crucial role, to host static content such as the images uploaded on the app, they are stored on the webserver and accessible via a URL from any network location, not only localhost, this is crucial in order for users to view each other's images. The server is also capable of handling image uploads directly to it, which will be utilised whenever a user uploads an image on the client, this is done via a PHP script which accepts a base64 encoded image, validates it's in JPEG or PNG format and if so, stores the image on the server.

Development of this backend architecture went smoothly with only one minor problem arising during development. Thankfully it was a problem that had already been identified during the API development stage, The CORS check. To solve the issue was very similar in PHP as to JavaScript and just ensued ensuring the OPTIONS request gets the expected values returned.

6.4 – Client App

6.4.1: Cross Platform Operability:



Car Photography 32d 0h

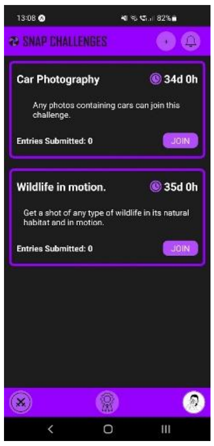
Any photos containing cars can join this challenge.

Entries Submitted: 0 JOIN

Wildlife in motion. 33d 0h

Get a shot of any type of wildlife in its natural habitat and in motion.

Entries Submitted: 0 JOIN



Car Photography 34d 0h

Any photos containing cars can join this challenge.

Entries Submitted: 0 JOIN

Wildlife in motion. 35d 0h

Get a shot of any type of wildlife in its natural habitat and in motion.

Entries Submitted: 0 JOIN

Figure 22 – Snap-Challenges on Desktop (left) and Mobile (right)

The main benefit identified during the literature review of using React Native was its cross-platform operability, meaning a single codebase could power both a mobile and web experience. This functionality of the technology has been utilised by Snap-Challenges which has both a working mobile and web experience which is illustrated above in figure 22. However, this cross operability does make this system more complex than just targeting one final platform. This concern was identified early into development when a date picker wouldn't perform as expected on desktop but worked on mobile. Due to this concern, a decision was made to build for mobile first, but ensure functionality on the web.

6.4.2: User Interface

Designs vs Final Implementation:

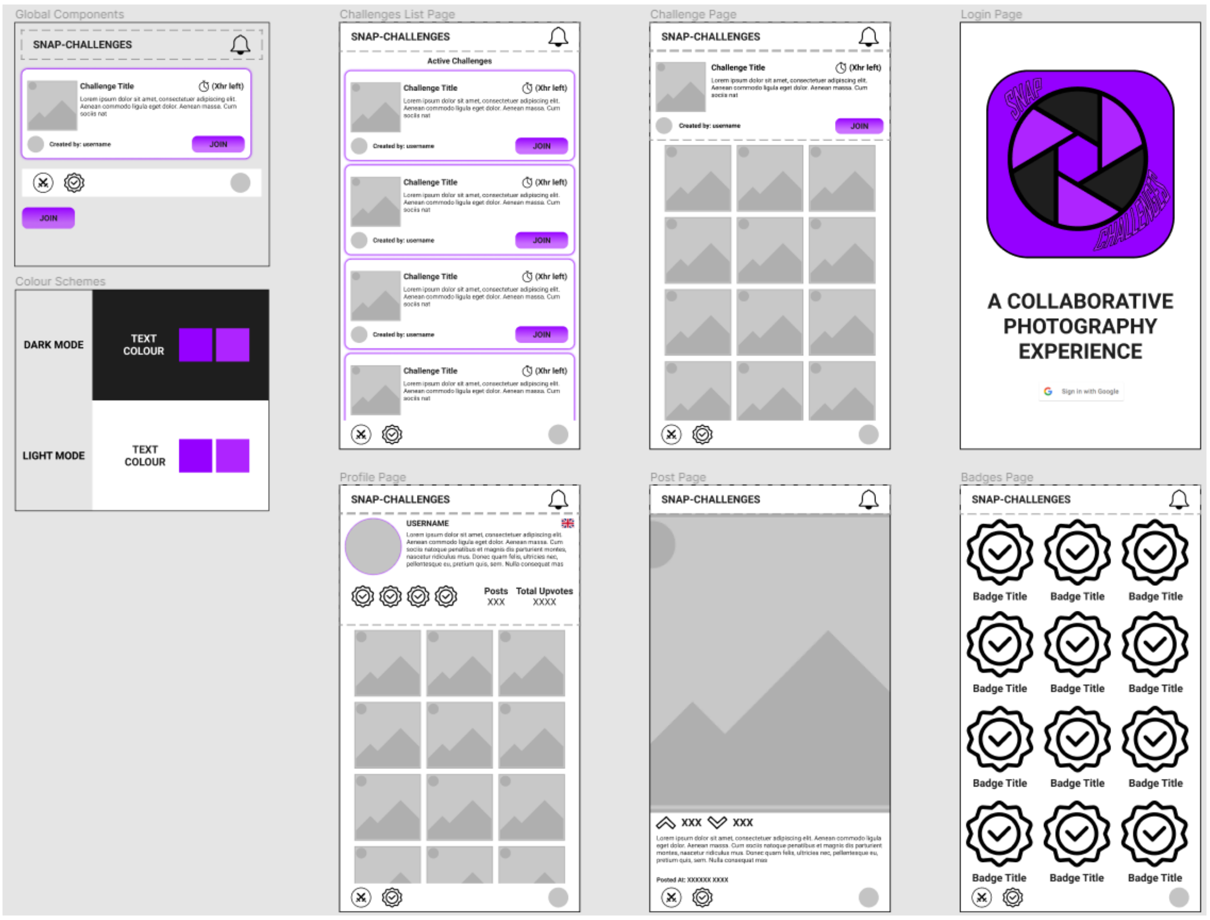


Figure 23 - Initial UI designs.

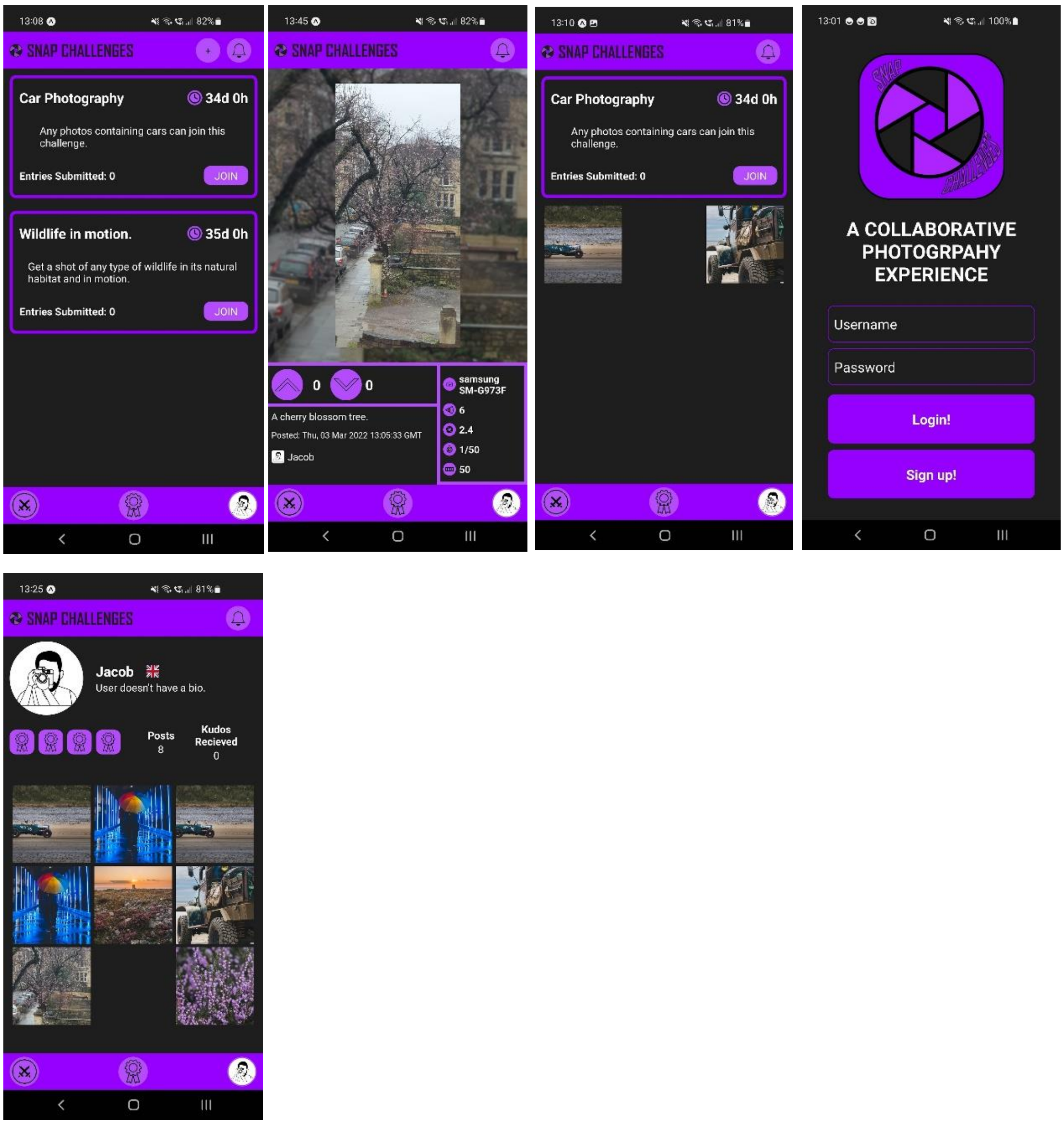


Figure 24 - Final Implementation UI

Overall, the final user interface shown in figure 24, closely resembles the initial planned designs shown in figure 23. Pages are very similar with minimal alterations.

The post page shows the most drastic alteration, the initial design failed to meet the requirement NFR-11, which requires camera metadata to be shown in a predominant fashion that is seen by the user.

The design of the login page has also changed as initially it was planed to handle logins via google oauth2. This plan changed in favour of using accounts stored on the database. This meant a login form and a signup page were required. However, a design for the signup page wasn't planned.

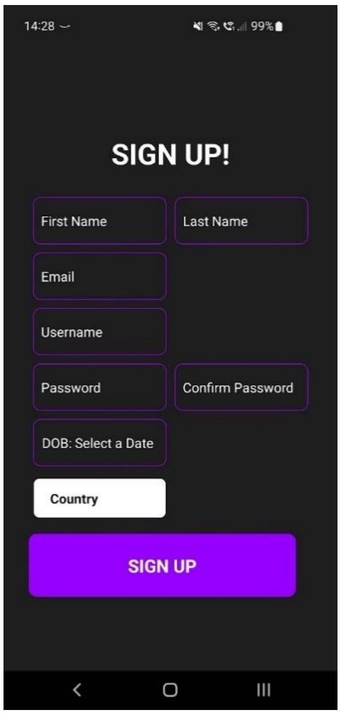


Figure 25 - Signup Page

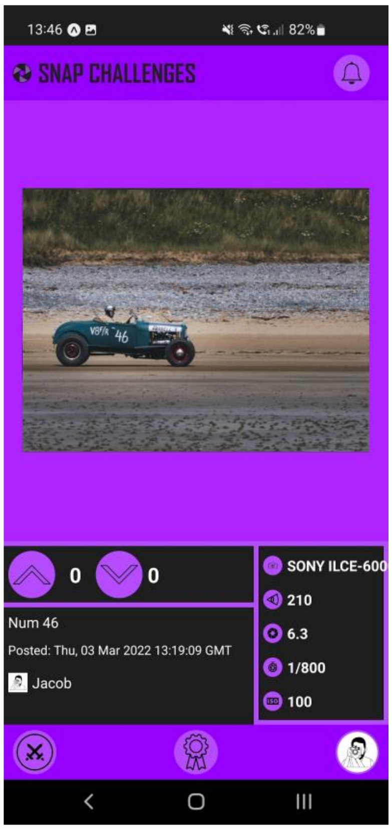
The final UI for the signup page is shown in figure 25, given the simplicity of the page it was decided not to revise the initial designs to accommodate this change.

Cross Platform Issues:

To have a clean user interface, each platform requires separate styling. This was not done for the initial build of Snap-Challenges due to the extra-time it would take to implement. Despite lacking its own styling, the interface on Desktop is still very functional and the user can still interact with the application in the exact same ways as on mobile.

Other Issues:

One of the more complex issues encountered during the development of Snap-Challenges was finding a way to display many different image aspect ratios on the post page. The initial approach was to just display the images as is however, for strange aspect ratios this really didn't work as the UI was pushed off screen, therefore meaning the page didn't conform to certain requirements predominantly NFR-11 which requires the camera meta data to be easily visible to the user.



Num 46

Posted: Thu, 03 Mar 2022 13:19:09 GMT

Figure 26 - Initial approach to handle aspect ratios.

A better method of handling this problem was discovered. This is shown in figure 26, the image is constrained within a bounding box and resizes to fit within it. This can handle a wide range of aspect ratios, and most will display well using this technique. However, there were still improvements to be made.

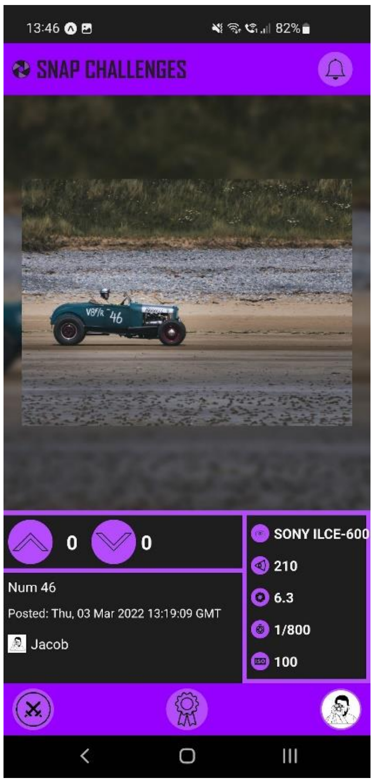


Figure 27 - Improved post UI

Figure 27, shows the final UI to solve this problem. This UI is more aesthetically pleasing than that shown in figure 26, the method of handling the aspect ratios remains the same, however, the image is now displayed ontop of a blurred, scaled, and darkened version of itself.

6.4.3: Badges

Initially it was planned that the user would be able to obtain badges by completing challenges. However, after deliberation it was decided this would be something that was not included in the initial product. This was done due to the inherent complexity involved with such a system. The backend for this system is in place on the database and endpoints are ready, however, implementing it into the app is something that would require a huge amount of planning and is unfeasible during the allocated time.

7 – Testing

7.1 – Introduction:

In this section the project will be tested by both the developer and users. The developer will complete rigorous internal testing of the different requirements implemented.

7.2 – Internal Testing:

7.2.1 – Functional Requirement Testing:

Database:

Table 20 - Database FR Testing

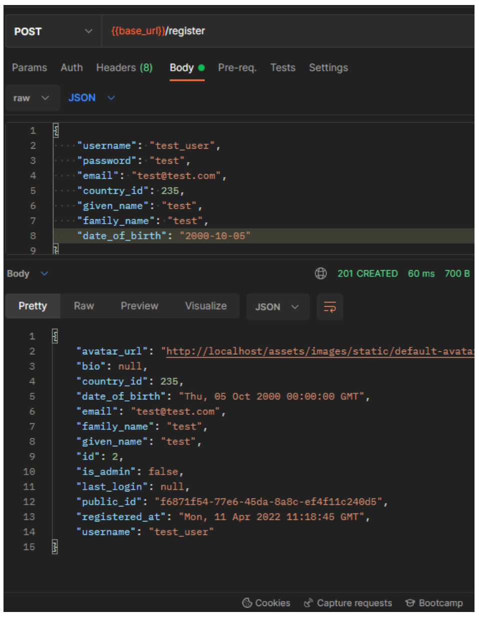
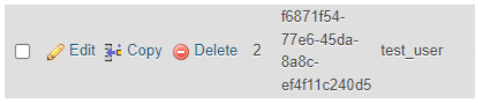
Test case	**Requirements Tested**	**Preconditions**	**Expected Result**	**Actual Result**	**Passed ?**
1 – Validating existence of DB tables after initialisation.	FR-1 -> FR-10	The database is setup.	All tables will exist on the DB	As expected.	Pass
2 – Validating that the countries table is prefilled with country data.	FR-11	Test 1 has passed, and the countries table exists.	All country data given in the given countries.json is stored as a record on the countries table. countries.json	As expected.	Pass

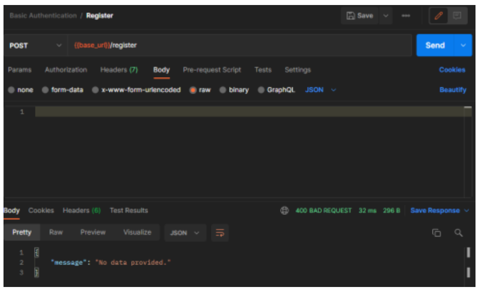
API:

The API has been tested using a free tool named postman.

Table 21 - API FR Testing

Test case	**Requirements Tested**	**Preconditions**	**Expected Result**	**Actual Result**	**Passed ?**

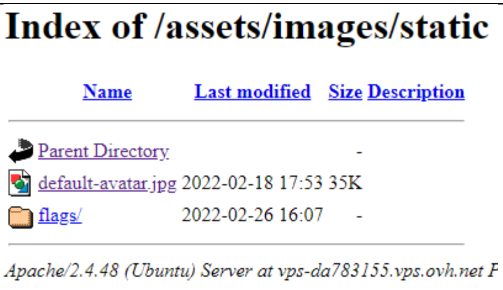
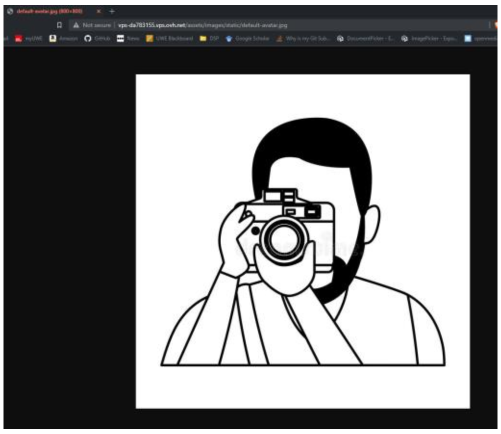
3 – Valid user data is passed to the register API endpoint.	FR-15	The database is setup.
		The API is running.
			Request is received and processed by the API server; HTTP Status code 201 returned alongside user data in JSON format. A record on the DB is also created.	As expected.
				
				POST `{{base_url}}/register`
				Params Auth Headers (8) Body Pre-req. Tests Settings
				raw JSON
				`{`
				`"username": "test_user",`
				`"password": "test",`
				`"email": "test@test.com",`
				`"country_id": 235,`
				`"given_name": "test",`
				`"family_name": "test",`
				`"date_of_birth": "2000-10-05"`
				`}`
				201 CREATED 60 ms 700 B
				`{`
				`"avatar_url": "http://localhost/assets/images/static/default-avata`
				`"bio": null,`
				`"country_id": 235,`
				`"date_of_birth": "Thu, 05 Oct 2000 00:00:00 GMT",`
				`"email": "test@test.com",`
				`"family_name": "test",`
				`"given_name": "test",`
				`"id": 2,`
				`"is_admin": false,`
				`"last_login": null,`
				`"public_id": "f6871f54-77e6-45da-8a8c-ef4f11c240d5",`
				`"registered_at": "Mon, 11 Apr 2022 11:18:45 GMT",`
				`"username": "test_user"`
				`}`
				
				Edit Copy Delete 2 f6871f54-77e6-45da-8a8c-ef4f11c240d5 test_user
					Pass

4 – Empty data is sent to register API endpoint.	FR-15	The database is setup.
		The API is running.
			Request is received and processed by the API server; HTTP Status code 400 returned alongside a "No data provided" message.	As expected.
				
				Basic Authentication / Register
				POST `{{base_url}}/register`
				Params Authorization Headers (7) Body Pre-request Script Tests Settings
				none form-data x-www-form-urlencoded raw binary GraphQL JSON
				400 BAD REQUEST 32 ms 296 B
				`{`
				`"message": "No data provided."`
				`}`
					Pass

CDN:

Table 22 - CDN FR Testing

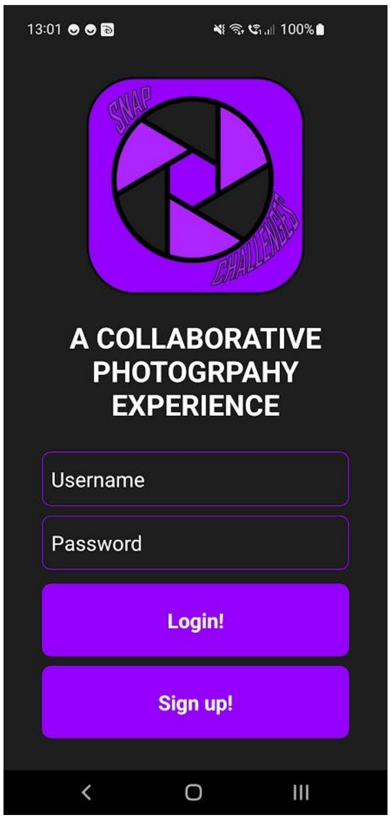
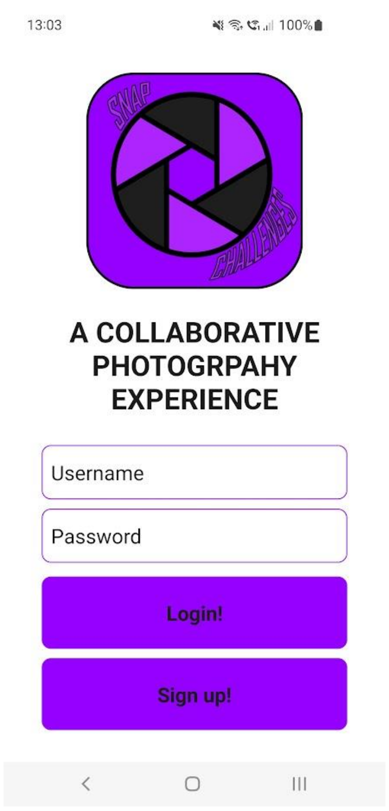
Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed?
55 – The CDN server	FR-29, FR-30	The CDN is running.	Files will be	As expected.	Pass

should store static files used by the clients, accessible via a URL.		The static files are stored on the CDN.	accessible via URL.	 **Index of /assets/images/static** Name Last modified Size Description Parent Directory - default-avatar.jpg 2022-02-18 17:53 35K flags/ 2022-02-26 16:07 - *Apache/2.4.48 (Ubuntu) Server at vps-da783155.vps.ovh.net F* 	

React Native App:

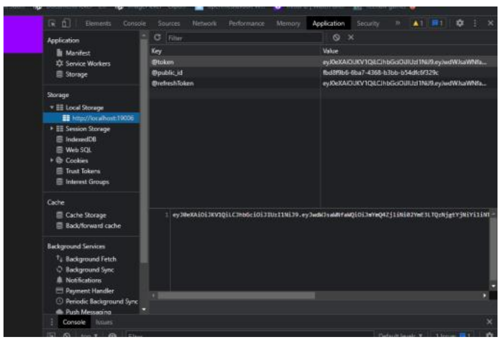
Table 23 - RN App FR Testing

Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed?
56 – All required pages are present.	FR-33, FR-34, FR-35, FR-36, FR-38	N/A	A login, signup, challenges, profile, and post page exists.	All pages present.	Pass

57 – Dark mode is enabled.	FR-39	N/A	The app loads in a dark theme.	As expected.  13:01 100% A COLLABORATIVE PHOTOGRPAHY EXPERIENCE Username Password Login! Sign up!	Pass
58 – Light mode is enabled.	FR-39	N/A	The app loads in a light theme.	As expected.  13:03 100% A COLLABORATIVE PHOTOGRPAHY EXPERIENCE Username Password Login! Sign up!	Pass

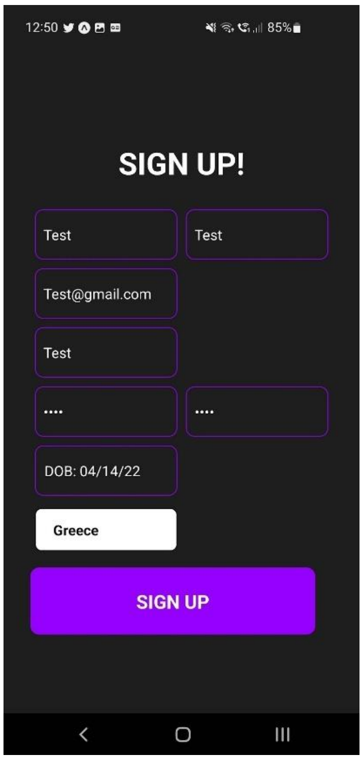
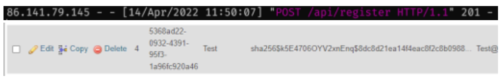
Login Page:

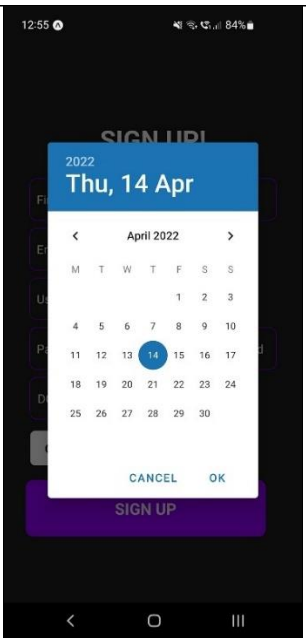
Table 24 - Login Page FR Testing

Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed ?
59 – A login button is present which when clicked logs the user in.	FR-40	The username field is filled. The password field is filled. The API is running. An account exists on the database with the given details.	The user is logged in and navigated to the challenges page.	As expected.	Pass
60 – When clicked the signup button on the welcome page redirects users to the signup page.	FR-41	None	User is navigated to signup page.	As expected.	Pass
61 – After login the token is stored in local storage.	FR-43	The valid login has taken place.	User's token is stored in the devices local storage.	As expected.  Key / Value @token eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJwdWJsaWNfa... @public_id fbd8f9b6-6ba7-4368-b3bb-b54dfc6f329c @refreshToken eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJwdWJsaWNfa...	Pass

SIGNUP PAGE:

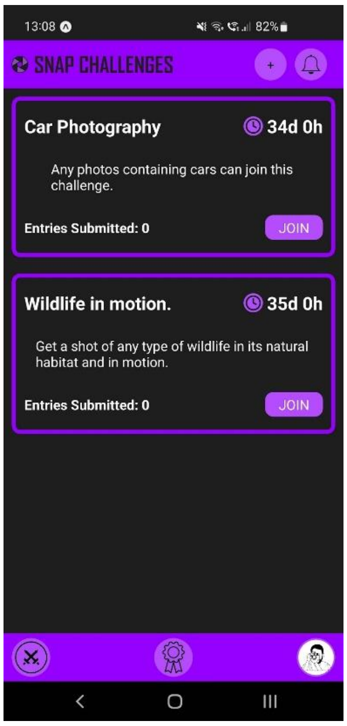
Table 25 - Signup Page FR Testing

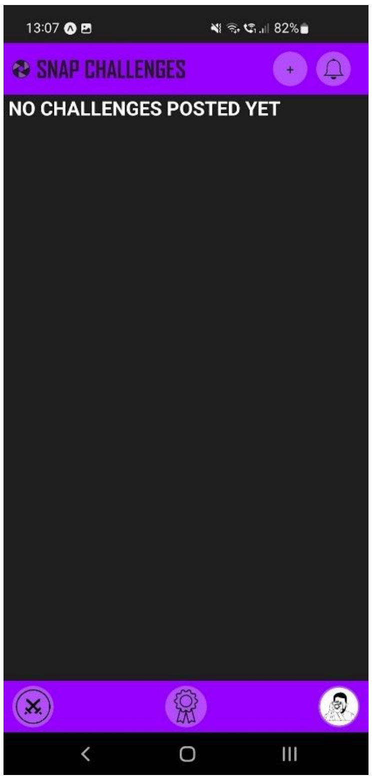
Test case	**Requirements Tested**	**Preconditions**	**Expected Result**	**Actual Result**	**Passed ?**
62 – When the signup button on the signup page is clicked a call to the API register endpoint is sent and the user account is created.	FR-44	All signup fields have been populated with valid data.	API call made, user account created and record stored on DB.	As expected.  12:50 85% SIGN UP! Test Test Test@gmail.com Test .... DOB: 04/14/22 Greece SIGN UP  `86.141.79.145 - - [14/Apr/2022 11:50:07] "POST /api/register HTTP/1.1" 201 -` Edit Copy Delete 4 5368ad22-0932-4391-95f3-1a96fc920a46 Test sha256$k5E4706OYV2xnEnq$8dc8d21ea14f4eac8f2c8b0988... Test@	Pass
63 – A functional date picker on the signup page.	FR-45	None	Onclick of the DOB field a date picker will be shown allowing a user to select a date.	As expected.	Pass

				 12:55 84% SIGN UP! 2022 Thu, 14 Apr April 2022 M T W T F S S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 CANCEL OK SIGN UP	

Challenges Page:

Table 26 - Challenges Page FR Testing

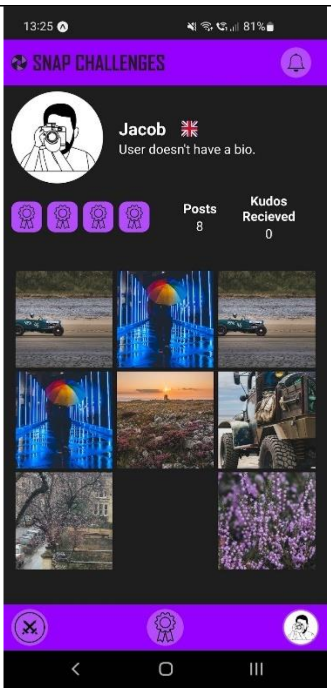
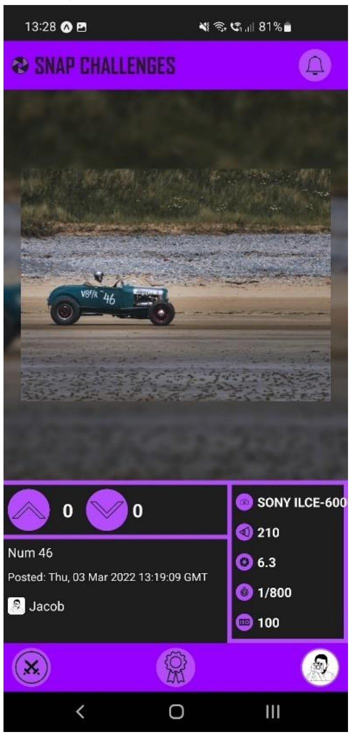
Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed ?
66 – On load a list of active challenges (ones with an end date in the future) is shown.	FR-49	User is logged in. Challenges exist with an end date in the future.	A list of challenges cards is shown.	As expected.  Car Photography 34d 0h Any photos containing cars can join this challenge. Entries Submitted: 0 JOIN Wildlife in motion. 35d 0h Get a shot of any type of wildlife in its natural habitat and in motion. Entries Submitted: 0 JOIN	Pass

67 – On load if there are no active challenges a message indicating this is shown.	FR-49	User is logged in. No Challenges exist with an end date in the future.	A message stating No challenges posted yet is shown.	As expected.  NO CHALLENGES POSTED YET	Pass
71 – A counter to show how many entries a challenge has had is shown.	FR-54	User is logged in and challenges exist.	The counter is shown.	Entries Submitted: 0 Partially working as the information is being pulled down from the database, however, the times_competed column on the challenges table of database is not updated every time a post is uploaded.	Fail

Profile Page:

Table 27 - Profile Page FR Testing

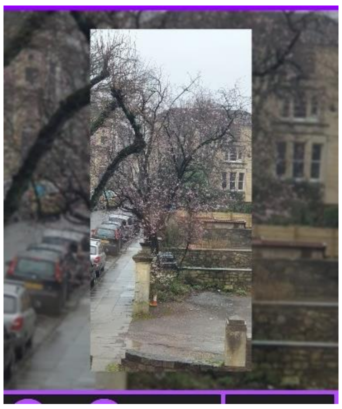
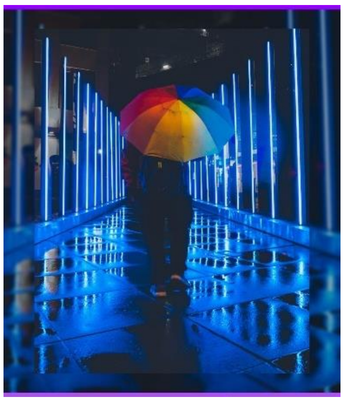
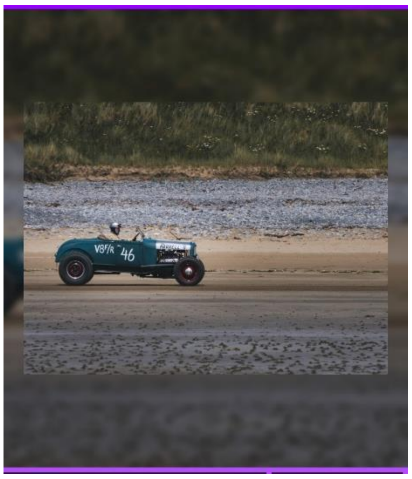
Test case	**Requirements Tested**	**Preconditions**	**Expected Result**	**Actual Result**	**Passed?**
72 – On load user data should be visible alongside a grid view of their posts.	FR-55, FR-58, FR-59, FR-60	User is logged in.	User avatar, username, country flag, bio, total posts,	As expected.	Pass

			and kudos received are all shown.	 13:25 81% SNAP CHALLENGES Jacob User doesn't have a bio. Posts 8 Kudos Recieved 0	
73 – Onclick a post image in the grid view should take you to the post page which displays more detail.	FR-62	User is logged in and posts exist.	User is redirected to the relevant post page.	As expected.  13:28 81% SNAP CHALLENGES 0 0 SONY ILCE-600 210 6.3 1/800 100 Num 46 Posted: Thu, 03 Mar 2022 13:19:09 GMT Jacob	Pass

Post Page

Table 28 - Post Page FR Testing

Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed?

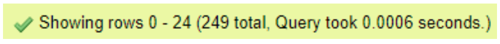
74 – Images shown on the post page must maintain their original aspect ratio. Apparent from really strange ratios such as 1:99.	FR-67	User is logged in post exists.	Most aspect ratios should be displayed in full.	As expected.   	Pass

7.2.2: Non-Functional Requirement Testing:

Database:

Table 29 - Database NFR Testing

Test case	**Requirements Tested**	**Preconditions**	**Expected Result**	**Actual Result**	**Passed ?**

78 - SQL query's should complete within a second.	NFR-1	None	SQL querys should execute in a matter of milliseconds.	Even queries to the largest tables on the DB take fractions of seconds.  Showing rows 0 - 24 (249 total, Query took 0.0006 seconds.)	Pass

API:

Table 30 - API NFR Testing

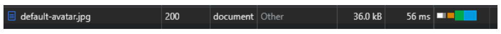
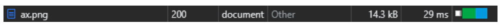
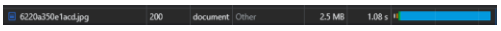
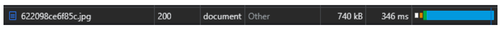
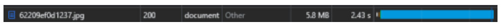
Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed ?
79 – Any API call should respond within 3 seconds.	NFR-2	None.	API calls should mostly complete within a matter of milliseconds.	This was tested by making 22 API calls a measuring the response time with postman. The following results were achieved:	Pass

Call	Response Time (MS)
1	60
2	46
3	60
4	44
5	58
6	40
7	26
8	168
9	51
10	45
11	40
12	46
13	43

				14 | 40 15 | 51 16 | 45 17 | 47 18 | 43 19 | 43 20 | 44 21 | 80 22 | 55 Total = 1175 Mean Response Time = 1175 / 22 = 53.4MS	

CDN:

Table 31 - CDN NFR Testing

Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed?
82 – The CDN should return content within an average of 3 seconds.	NFR-5	Content is stored on the CDN.	Very dependant on file size smaller files should easily be returned within the 3 seconds, however, some larger files may	5 requests for content from the CDN were ran and the response time logged using chrome dev tools to see response time.  default-avatar.jpg 200 document Other 36.0 kB 56 ms  ax.png 200 document Other 14.3 kB 29 ms  6220a350e1acd.jpg 200 document Other 2.5 MB 1.08 s  622098ce6f85c.jpg 200 document Other 740 kB 346 ms  62209ef0d1237.jpg 200 document Other 5.8 MB 2.43 s 56 + 29 + 1080 + 346 + 2430 = 3941 3941/5 = 788.2 ms 788.2/1000 = 0.7 seconds	Pass

			take longer.		

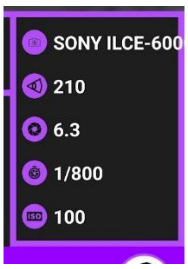
React Native App:

Table 32 - RN App NFR Testing

Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed ?
85 – Any page should load within 3 seconds on a 4G connection.	NFR-8	None	Most pages should load within the expected time frame some with a lot of image content may take longer.	Each pages load time will be measured on a 4G throttled connection (20mbps) using chrome devtools. Login Page: Load: 692 ms Sign Up Page: Load: 695 ms Challenges List Page: Load: 345 ms Challenge Page: Load: 652 ms Profile Page: Load: 822 ms Post Page: Load: 415 ms Total Time : 3621 ms Avg time: 603.5 ms / 0.6 seconds.	Pass

Post Page:

Table 33 - Post Page NFR Testing

Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed ?
86 – Camera meta data should be very clear and easy to spot.	NFR-11	None	Camera info is clear and well positioned.	The info is clearly identifiable, however there are a few slight overlapping issues which need to be addressed.  SONY ILCE-600 210 6.3 1/800 100	Pass

This was a curated selection of all the tests conducted the rest can be seen in Appendix B.

7.3 – Conclusion:

In conclusion, the app manages to meet all the MUST requirements outlined in section 3 of this report. The extent to which it does this varies but all at least have the minimum functionally to complete, most also stand up to testing passed the scope of the requirement. However, one common issue identified through the testing process, specifically the testing focussed on the API requirements. There is a lack of validation on certain API endpoints. There are also a few improvements to the user interface that have been identified by this testing process. For example, the overlapping issue with longer text on the post page.

8 – Evaluation

8.1 – Introduction:

This section of the report is to evaluate the product and development process. Focussing on aims, research, requirements, methodology, design, and the tools used. There will also be an analysis of how Project in Progress Day further shaped my project.

8.2 – Aims

“To develop a platform independent app to gamify photography via the use of challenges.”

Snap-Challenges works on both the web and mobile devices thanks to the decision to use React Native. Therefore, it is safe to conclude that Snap-Challenges delivers on this aim, however, that’s not to say its without improvement, specifically the Desktop version, although functional it lacks aesthetics and is something that needs to be improved in the future.

“To minimise burn out amongst photographers using these challenges to engage them with the app and their hobby.”

This aim is a lot harder to quantify, however, previous research conducted in the literature review leads to suggest that a gamified approach to the application should engage users more than a traditional approach. However, before a conclusion can be drawn for this aim, it is the developer’s opinion that it would need to be released for users to try.

“To educate the users of the app by providing information about the camera settings used to achieve the resulting photo.”

Camera metadata is well shown on the post page, giving the users a prominent and detailed insight into the exact settings used by a photographer. This information is obtained automatically when an image is uploaded, the exif data of the image is read and stored on the database, this means the data is as accurate as is possible as the data is being read directly from the information that the camera stored when the image was taken. Therefore, I think it is fair to say that this aim is well met.

8.3 – Requirements:

Overall, the project had initially set out 82 requirements (71 functional, and 11 non-functional). These were broken down into the following priority:

- 47 MUST
- 19 SHOULD
- 13 COULD
- 3 WON’T

The completed requirement breakdown is as follows:

- 47 MUST
- 9 SHOULD

- 3 COULD
- 0 WON’T

From this break down it is clear to see that Snap-Challenges has not only met all the must requirements but also extended upon what was needed to be completed for a functional minimal viable product. Further development could be used to complete the remaining requirements.

8.4 – Research:

I feel the literature review for this project served its purpose well and some of the knowledge gathered was invaluable to the development of Snap-Challenges. However, I do feel it could’ve benefitted from slightly more academic source material as it appeared there was a lack of such material in the photography gamification field.

8.5 – Methodology:

The chosen methodology of feature-driven-development worked well for the project as it allowed me to focus on specific parts of the final product. This break down of tasks really help divide the workload into a naive but attainable time schedule, which for app development was mostly met with different features being developed in the appropriate time scale. However, this was not the case for the write up of this report which quite frequently fell behind the initial planned schedule. This was due to various reasons, the main being spare time being prioritised to development rather than writeup.

8.6 – Design:

Overall, I am ambivalent about the design, on mobile, it works great, matches initial design diagrams and is aesthetically pleasing. However, the same cannot be said about the web version of the app intended for use on desktop. This is due to the app using the same stylesheet for both platforms which was developed with a focus on mobile as it was decided that mobile would be the primary platform. Whilst I do feel that this decision was the right choice it has without a doubt negatively impacted the web version of the app.

8.7 – Tools & Services Used:

8.7.1 - React Native:

The choice to use React Native of PWAs was made because of my research during the literature review, I feel this choice was crucial to the project as without proper research a wrong decision made here could have been seriously detrimental to the project. React native worked perfectly for my use case due to the bare metal access it gives to device level APIs such as the ability to access device cameras. I feel I made the right choice as achieving similar functionality to this level of device access, with a PWA would’ve been difficult and very time consuming.

8.7.2 – Flask:

Flask is the web-framework used by the API, throughout the project it has proven to be a robust library, it was very beginner friendly, this project being my first time utilising it and being built for Python was very beneficial as Python is the language, I have the most experience programming in.

8.7.3 – OVH Hosting:

The only external service I used for the app was server hosting by OVH, from whom I rented a VPS, this turned out to be a problematic endeavour, thanks to OVH not auto renewing my payment for the VPS, despite it being set to do so and failing to contact me about it. This resulted in me losing access to the VPS and it being wiped. This was a pain as I had to waste about half a day resetting up a new VPS instead of working on the report. In the future I will make a docker image of the server, so it is easy to reinstall.

8.8 – Project in Progress & Supervisor Feedback:

8.8.1 - PiP:

Project in progress day was a great motivator for me as both my supervisor and second marker seemed to be content with the progress, I had made which was good to hear as at that point I had personally felt that I was falling behind. This motivated to put the work in and continue the report.

8.8.2 - Supervisor Feedback:

Overall, I have improved on my project in many ways with the feedback given and questions answered by my supervisor in our meetings. This has been done by implementing any changes suggested that I felt were beneficial to the project.

8.9 – Future Work

There are many different improvements that could be made to the app in the future. Some of which are listed below:

- DESKTOP UI IMPROVEMENTS
- BADGES
- SCALABILITY
- MORE MODES
- PRIZES
- NOTIFICATIONS

9 – Conclusion

9.1 – Aims:

I feel the project has delivered upon its main aims and completed its objectives, some more testing will however be needed to verify if the app is released to be released and have a user base.

9.2 – Limitations:

I do feel that there are some limitations which need fixing before the app can be deemed as complete. The API needs some improvement specifically in terms of input validation, this is something that must be improved. The client app is lacking any of the functionality regarding badges which I had initially planned to be completed, this is rather detrimental as the badges were to be treated as the incentive for completing challenges. However, in its current state the only incentive to completing challenges is the kudos received for each post. I fear that this may not be enough on its own and without the badges engagement with the app will be lower. For this reason, implementing badges should be the main priority of the next update to the app.

9.2 - Closing Statement:

Overall, I feel the project has been successful, all the aims and must requirements have been met and the app provides exactly what it promises, a fun way for photographers to compete in a friendly manner whilst also being educated about how images were created.

10 – References

- Agile Alliance (2015) 12 Principles Behind the Agile Manifesto | Agile Alliance. Available from: https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/ [Accessed 9 March 2022].

- Anon (2018) r/photography - Your Take on Apps Like GuruShots and Other Photo Challenge Apps. Available from: https://www.reddit.com/r/photography/comments/9vnjvm/your_take_on_apps_like_gurushots_and_other_photo/ [Accessed 16 January 2022].

- Apple (2021) App Store Review Guidelines - Apple Developer. Available from: https://developer.apple.com/app-store/review/guidelines/ [Accessed 23 January 2022].

- Beck, K., Beedle, M., Bennekum, A. V., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martrin, R. C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001) Manifesto for Agile Software Development. Available from: https://agilemanifesto.org/ [Accessed 9 March 2022].

- Bromberg-Martin, E., S., et al (2010) Dopamine in Motivational Control: Rewarding, Aversive, and Alerting. *Neuron* [online]. 68(5), pp.815-834. [Accessed 16 January 2022]

- Bujari, A., et al (2016) Using gamification to discover cultural heritage locations from geo-tagged photos. *Personal and Ubiquitous Computing* [online]. 21 (no part/issue), pp.235-252. [Accessed 16 January 2022]

- Carlson, J. (2013) *Redis in Action* [online] New York: Manning Publications Co. [Accessed 28 December 2021]

- Cobb, C. G. (2015) *The project manager's guide to mastering agile: principles and practices for an adaptive approach* [online] Hoboken, New Jersey: Wiley. [Accessed 09 Feb 2022]

- Dabit, N. (2019) *React Native in Action* [online] New York: Manning Publications [Accessed 23 January 2022]

- Eisenman, B. (2017) *Learning React Native* [online] 2nd Edition. California: O'reilly Media, Inc. [Accessed 21 January 2022]

- Expo (2022) Camera - Expo Documentation. Available from: https://docs.expo.dev/versions/latest/sdk/camera/ [Accessed 23 January 2022].

- Felsing, J. M., Palmer, S. R. (2002) *A Practical Guide to Feature-Driven Development.* 1st Edition. Upper Sadle River, NJ, USA: Prentice Hall.

- Flask-CORS (2022) Flask-CORS — Flask-Cors 3.0.10 documentation. Available from: https://flask-cors.readthedocs.io/en/latest/ [Accessed 30 March 2022].

- Flask-RESTful (2020) Flask-RESTful — Flask-RESTful 0.3.8 documentation. Available from: https://flask-restful.readthedocs.io/en/latest/ [Accessed 24 March 2022].

- Flask-SQLAlchemy (2021) Flask-SQLAlchemy — Flask-SQLAlchemy Documentation (2.x). Available from: https://flask-sqlalchemy.palletsprojects.com/en/2.x/# [Accessed 23 March 2022].

- Google (2021) List your Progressive Web App in Google Play. Available from: https://chromeos.dev/en/publish/pwa-in-play [Accessed 23 January 2022].
- Google Play (2013) ViewBug - Photography. Available from: https://play.google.com/store/apps/details?id=com.viewbug.viewbug&hl=en_GB&gl=US [Accessed 16 January 2022].
- Groff, J. R. and Weinberg, P. N. (2002) *SQL : The Complete Reference.* New York: McGraw-Hill Professional. Available at: https://search-ebscohost-com.ezproxy.uwe.ac.uk/login.aspx?direct=true&db=nlebk&AN=80523&site=ehost-live [Accessed: 21 December 2021].
- Hamari, J., Huotari, K. (2012) 16th International Academic Mindtrek Conference [online], Tampere, Finland, 3-5 October 2012. Association for Computing Machinery. Available From: https://dl-acm-org.ezproxy.uwe.ac.uk/doi/10.1145/2393132.2393137 [Accessed 05 January 2022]
- Hence, J., et al (2017) How gamification motivates: An experimental study of the effects of specific game design elements on psychological need satisfaction. *Computers in Human Behavior* [online] 69 (no part/issue), pp.271-380. [Accessed 12 January 2022]
- Hossain, M. (2014) *CORS in Action: Creating and consuming cross-origin APIs* [online]. New York: Manning Publications. [Accessed 24 March 2022]
- Kim, H., et al (2018) *CCS: Computer and Communications Security 18* [online], Toronto, ON, Canada, 15-19 October 2018. Association of Computing Machinery. Available from: https://dl-acm-org.ezproxy.uwe.ac.uk/doi/10.1145/3243734.3243867 [Accessed 23 January 2022]
- Leavitt, N. (2010) Will NoSQL Databases Live Up to Their Promise?. *Computer* [Online]. 43 (2), pp. 12-14. [Accessed 27 December 2021].
- Liestøl, G. (2018). The Photo Positioning Puzzle : Creating Engaging Applications for Historical Photographs by Combining Mobile Augmented Reality and Gamification. In: 2018 3rd Digital Heritage International Congress (DigitalHERITAGE) held jointly with 2018 24th International Conference on Virtual Systems & Multimedia (VSMM 2018). [online] Digital Heritage International Congress (DigitalHeritage). Available from: https://ieeexplore-ieee-org.ezproxy.uwe.ac.uk/document/8810038/ [Accessed 16 January 2022].
- Murphy, K. (2021) Instagram alienates photography community after CEO's recent statement DPReview.2 July 2021 [online]. DPReview. Available from: https://www.dpreview.com/news/1468763598/instagram-alienates-photography-community-after-ceo-s-recent-statement [Accessed 20 April 2022].
- NOSQL (2009) *NOSQL Database.* Available From: https://hostingdata.co.uk/nosql-database/ [Accessed: 22 December 2021].
- React Native (a) (2022) Out-of-Tree Platforms – React Native. Available From: https://reactnative.dev/docs/out-of-tree-platforms [Accessed 23 January 2022]
- React Native (b) (2022) React Native · Learn once, write anywhere. Available from: https://reactnative.dev/ [Accessed 23 January 2022].

- React Native (c) (2022) Using Typescript – React Native. Available From: https://reactnative.dev/docs/typescript [Accessed 23 January 2022].
- Request for Comments (1999) *2616 Hypertext Transfer Protocol -- HTTP/1.1* [online]. IETF Datatracker. (no place): RFC Editor. Available from: https://datatracker.ietf.org/doc/html/rfc2616
- SQLite (2022) About SQLite. Available from: https://www.sqlite.org/about.html [Accessed 23 March 2022].
- vimalloc (2022) vimalloc/flask-jwt-extended: An open source Flask extension that provides JWT support (with batteries included)! Available from: https://github.com/vimalloc/flask-jwt-extended [Accessed 24 March 2022].
- Wang, X., et al (2017) Examining the Effectiveness of Gamification in Human Computation. *International Journal of Human–Computer Interaction* [online] 33 (10), pp.813-821. [Accessed 11 January 2022]

11 – Appendices

Appendix A: Requirements:

Functional:

Database:

Table 34 - Appendix Database FRs

ID	DESCRIPTION	MoSCoW
FR-11	The Countries table should be auto filled with data on database initialisation.	S
FR-12	Data stored in the database could be regularly backed up.	C
FR-13	SQL Views to allow administrators to view data all on one table.	W
FR-14	Redis in memory cache	W

API:

Table 35 - Appendix API FRs

ID	DESCRIPTION	MoSCoW
FR-19	**Users Has Challenges Endpoint** – Used for handling data involving the challenges users have joined. Could be used for showing a list of challenges the user has joined. Accepts GET/POST/DELETE requests. **GET returns:** • 200 and user challenge data if successful. • 404 if the user challenge data can't be found. **POST returns:** • 201 and user challenge data if successful. • 400 if the user challenge data isn't provided. **DELETE returns:** • 204 if successful • 404 if not found.	C
FR-20	**Users Has Posts** – Used for handling data involving the posts users have made. Accepts GET/POST/DELETE requests. **GET returns:** • 200 and user post data if successful. • 404 if the user post data can't be found. **POST returns:**	M

	• 201 and user post data if successful. • 400 if the user post data isn't provided. **DELETE returns:** • 204 if successful • 404 if not found.	
FR-21	**Users Has Badges** – Used for handling data involving the badges users have been granted. Accepts GET/POST/DELETE requests. **GET returns:** • 200 and user badge data if successful. • 404 if the user badge data can't be found. **POST returns:** • 201 and user badge data if successful. • 400 if the user badge data isn't given. **DELETE returns:** • 204 if successful • 404 if not found.	S
FR-22	**Photos Endpoint** – Used for handling data about photos specified posts. Accepts GET/POST/DELETE requests. **GET returns:** • 200 and photo data if successful. • 404 if photo data can't be found. **POST returns:** • 201 and photo data if successful. • 400 if the no photo data provided. **DELETE returns:** • 204 if successful • 404 if not found.	M
FR-23	**Posts Endpoint** – Used for handling post data. Accepts GET/POST/PUT/DELETE requests. **GET returns:** • 200 and post data if successful. • 404 if post data can't be found. **POST returns:** • 201 and post data if successful. • 400 if the no post data provided. **PUT returns:** • 204 and updated post data if successful.	M

	• 404 if can’t find a post to update. • 400 if required data not provided. **DELETE returns:** • 204 if successful • 404 if not found.	
FR-24	**Challenge endpoint** – Used for handling challenge data. Accepts GET/POST/PUT/DELETE requests. **GET returns:** • 200 and challenge data if successful. • 404 if challenge data can’t be found. **POST returns:** • 201 and challenge data if successful. • 400 if the no challenge data provided. **PUT returns:** • 204 and updated challenge data if successful. • 404 if can’t find a challenge to update. • 400 if required data not provided. **DELETE returns:** • 204 if successful • 404 if not found.	M
FR-25	**Challenge Has Posts endpoint** – Used for handling data related to posts which belong to challenges. Accepts GET/POST/DELETE requests. **GET returns:** • 200 and challenge post data if successful. • 404 if challenge post data can’t be found. **POST returns:** • 201 and challenge post data if successful. • 400 if the no challenge post data provided. **DELETE returns:** • 204 if successful • 404 if not found.	M
FR-26	**Countries Endpoint –** Handles data about countries. Accepts GET/POST requests. **GET returns:** • 200 and country data if successful. • 404 if country data can’t be found.	M

	POST returns: • 201 and country data if successful. • 400 if the no country data provided.	
FR-27	**Badges Endpoint –** Handles data about badges. Accepts GET/POST/PUT/DELETE requests. **GET returns:** • 200 and badge data if successful. • 404 if badge data can't be found. **POST returns:** • 201 and badge data if successful. • 400 if the no badge data provided. **PUT returns:** • 204 and updated badge data if successful. • 404 if can't find a badge to update. • 400 if required data not provided. **DELETE returns:** • 204 if successful • 404 if not found.	S
FR-28	**A scope system -** certain users have unrestricted access, others are limited by scopes, for example admins, mods may have access to endpoints that users wouldn't.	C

CDN:

Table 36 - Appendix CDN FRs

ID	DESCRIPTION	MoSCoW
FR-31	CDN should load assets from a server local to the user's geo location in order to increase loading speeds.	W

REACT NATIVE APP:

Table 37 - Appendix RN App FRs

ID	DESCRIPTION	MoSCoW
FR-39	The app should support both light and dark themes and should determine which to used based on the users pre-set system preference.	S

LOGIN PAGE:

Table 38 - Appendix Login Page FRs

ID	DESCRIPTION	MoSCoW
FR-42	A message should be displayed if login has failed.	S
FR-43	If login is successful the token returned must be stored in the local storage, this will allow logins to be skipped if the token hasn't expired.	S

SIGN UP PAGE:

Table 39 - Appendix Signup Page FRs

ID	DESCRIPTION	MoSCoW
FR-48	Input validation to warn users if their input is incorrect. This is done on the API side anyway, but a warning message on the client side would be a nice to have.	C

CHALLENGES PAGE:

Table 40 - Appendix Challenges Page FRs

ID	DESCRIPTION	MoSCoW
FR-52	A timer should be displayed next to each challenge to indicate how much time is left on the challenge.	S
FR-53	The user could bookmark a challenge in order to indicate that they are interested in it but don't immediately have an image to post for it.	C
FR-54	The number of posts for each challenge could be shown on the card for each challenge.	C

PROFILE PAGE:

Table 41 - Appendix Profile Page FRs

ID	DESCRIPTION	MoSCoW
FR-56	A selection of recent badges will be displayed on the profile.	S
FR-57	Users could be able to pick which badges are displayed.	C
FR-61	User join date could be shown.	C
FR-63	When a badge is clicked details about it should be shown.	S

BADGE PAGE:

Table 42 - Appendix Badge Page FRs

ID	DESCRIPTION	MoSCoW
FR-65	Badges the user has achieved should be shown fully whilst locked ones should be slightly less opaque.	S
FR-66	Progress towards each badge could be shown.	C

POST PAGE:

Table 43 - Appendix Post Page FRs

ID	DESCRIPTION	MoSCoW
FR-70	Time posted should be shown.	S
FR-71	A menu that will allow the user to edit or delete the post if it is their own.	S

Non-Functional:

API:

Table 44 - Appendix API NFRs

ID	DESCRIPTION	MoSCoW
NFR-4	Hosted on a Virtual Private Server (VPS) so the API will be accessible almost all the time.	C

CDN:

Table 45 - Appendix CDN NFRs

ID	DESCRIPTION	MoSCoW
NFR-5	The CDN should return content within 3 seconds on average (larger images may take longer due to their larger file sizes).	S
NFR-6	Hosted on a Virtual Private Server (VPS) so the API will be accessible almost all the time.	C

REACT NATIVE APP:

Table 46 - Appendix RN APP NFRs

ID	DESCRIPTION	MoSCoW
NFR-9	A text scale slider could be made available to allow people who struggle to read smaller font sizes to have a good experience on the app.	C
NFR-10	A colour theme selector could be made to support people who struggle to read with certain colours as backdrops.	C

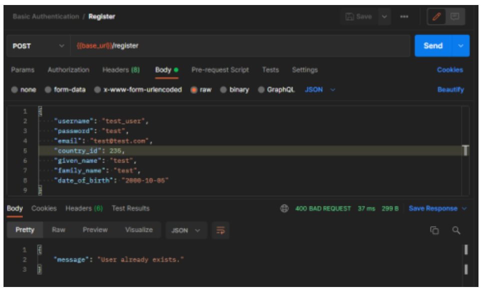
Appendix B: Testing:

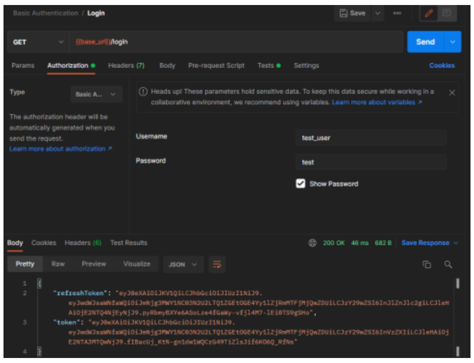
Functional Requirement Testing:

API:

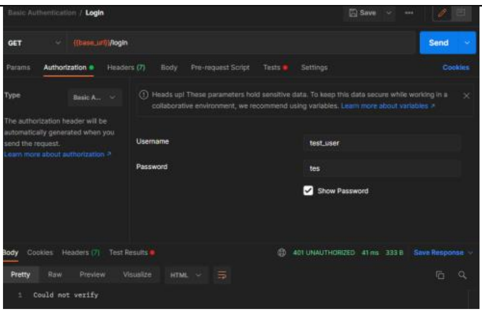
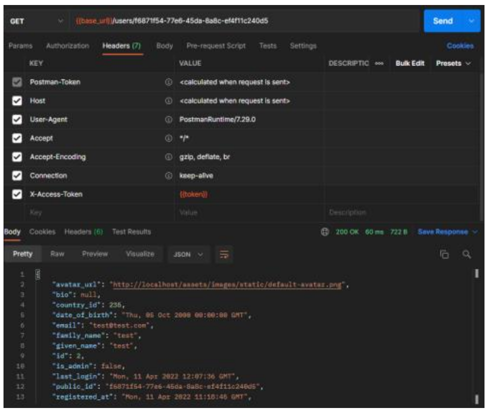
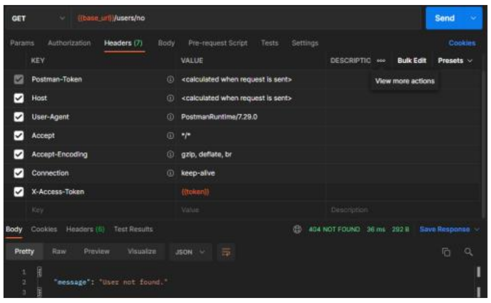
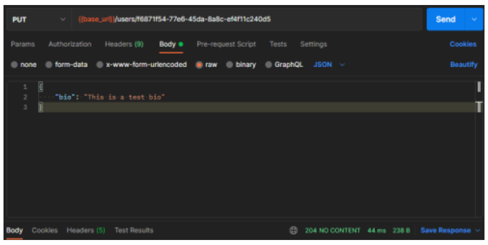
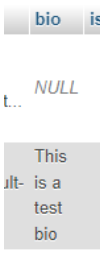
Table 47 - Appendix API FR Testing

Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed ?

5 – An existing username is sent to the register API endpoint.	FR-15	The database is setup.
		The API is running.
		A test user exists.
			Request is received and processed by the API server; HTTP Status code 400 returned alongside a “User already exists.” message.	As expected.
				
				Basic Authentication / Register
				POST {{base_url}}/register
				Body
				raw
				JSON
				`{`
				`"username": "test_user",`
				`"password": "test",`
				`"email": "test@test.com",`
				`"country_id": 235,`
				`"given_name": "test",`
				`"family_name": "test",`
				`"date_of_birth": "2000-10-05"`
				`}`
				400 BAD REQUEST 37 ms 299 B
				`{`
				`"message": "User already exists."`
				`}`
					Pass

6 – Valid data sent to the login API endpoint.	FR-16	The database is setup.
		The API is running.
		A test exists.
			Request is received and processed by the API server; HTTP Status code 200 returned alongside tokens in JSON format.	As expected
				
				Basic Authentication / Login
				GET {{base_url}}/login
				Type Basic A...
				The authorization header will be automatically generated when you send the request.
				Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.
				Username test_user
				Password test
				✓ Show Password
				200 OK 46 ms 682 B
				`{`
				`"refreshToken": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJwdWJsaWNfaWQiOiJmNjg3MWY1NC03N2U2LTQ1ZGEtOGE4Yy1lZjRmMTFjMjQwZDUiLCJzY29wZSI6InJlZnJlc2giLCJleHAiOjE2NTQ4NjEyNjJ9.pyRbmyBXYe6ASoLze4fGaWy-vfjl4M7-lEi0TS9gSHo",`
				`"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJwdWJsaWNfaWQiOiJmNjg3MWY1NC03N2U2LTQ1ZGEtOGE4Yy1lZjRmMTFjMjQwZDUiLCJzY29wZSI6InVzZXIiLCJleHAiOjE2NTA3MTQwNjJ9.fIBacUj_KtN-gn1dw1WQCrG49TiZlsJif6KO6Q_RfNs"`
				`}`
					Pass

7 – Invalid login informatio	FR-16	The database is setup.	HTTP STATUS code 401	As expected.	Pass

n is sent to the login endpoint e.g. an incorrect password.		The API is running.	returned alongside a “could not verify” message.		
8 – Valid GET request for an existing user is sent to the API.	FR-18	The database is setup. The API is running. The test user exists.	HTTP Code 200 and user data returned.	As expected. 	Pass
9 – A GET request for a non-existent user is sent.	FR-18	The database is setup. The API is running.	HTTP Code 404 and cannot be found message returned.	As expected. 	Pass
10 – A valid PUT request is sent to the users endpoint.	FR-18	The database is setup. The API is running. A test user exists.	HTTP code 204 returned and db record updated.	As expected.  	Pass

Basic Authentication / Login

GET {{base_url}}/login

Type: Basic A...

The authorization header will be automatically generated when you send the request.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.

Username	test_user
Password	tes

✓ Show Password

401 UNAUTHORIZED 41 ms 333 B

```
Could not verify
```

GET {{base_url}}/users/f6871f54-77e6-45da-8a8c-ef4f11c240d5

KEY	VALUE
Postman-Token	<calculated when request is sent>
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.29.0
Accept	*/*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
X-Access-Token	{{token}}

200 OK 60 ms 722 B

```
{
    "avatar_url": "http://localhost/assets/images/static/default-avatar.png",
    "bio": null,
    "country_id": 236,
    "date_of_birth": "Thu, 05 Oct 2000 00:00:00 GMT",
    "email": "test@test.com",
    "family_name": "test",
    "given_name": "test",
    "id": 2,
    "is_admin": false,
    "last_login": "Mon, 11 Apr 2022 12:07:36 GMT",
    "public_id": "f6871f54-77e6-45da-8a8c-ef4f11c240d5",
    "registered_at": "Mon, 11 Apr 2022 11:18:46 GMT",
```

GET {{base_url}}/users/no

KEY	VALUE
Postman-Token	<calculated when request is sent>
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.29.0
Accept	*/*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
X-Access-Token	{{token}}

404 NOT FOUND 36 ms 292 B

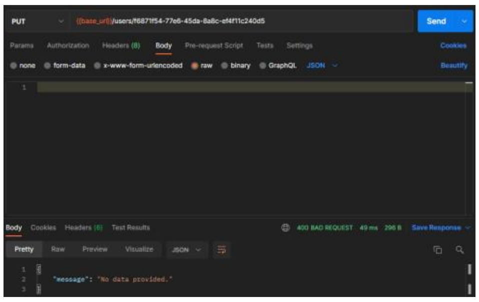
```
{
    "message": "User not found."
}
```

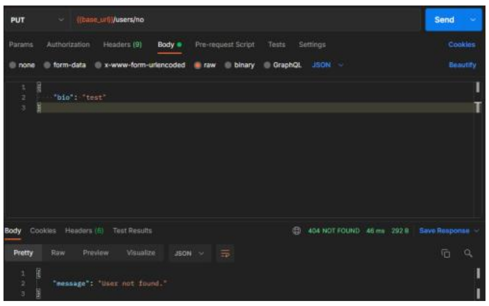
PUT {{base_url}}/users/f6871f54-77e6-45da-8a8c-ef4f11c240d5

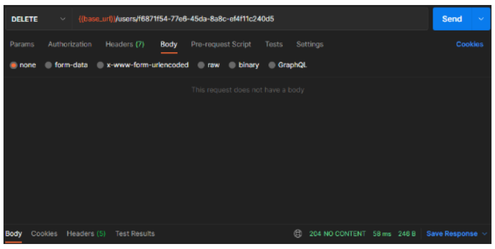
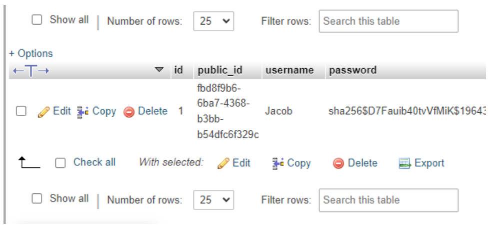
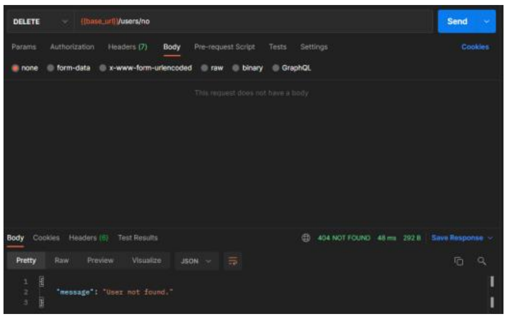
```
{
    "bio": "This is a test bio"
}
```

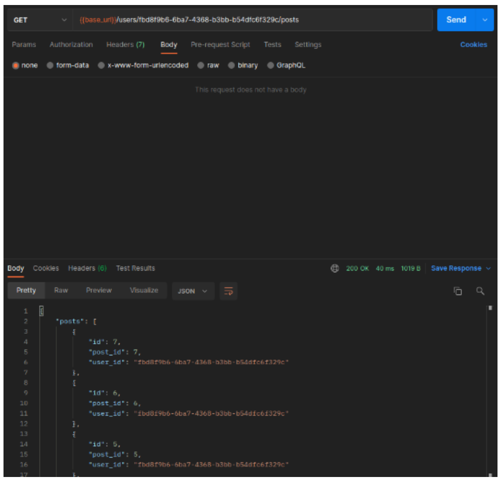
204 NO CONTENT 44 ms 238 B

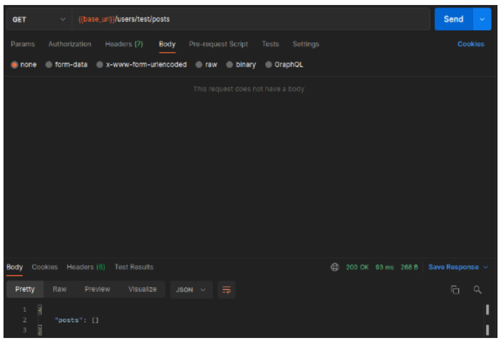
	bio	is
t...	NULL	
ult-	This is a test bio	

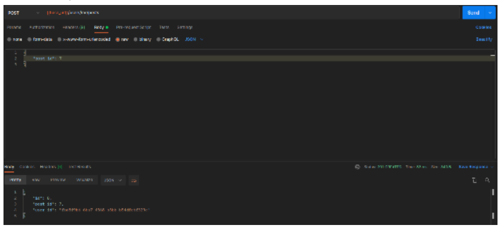
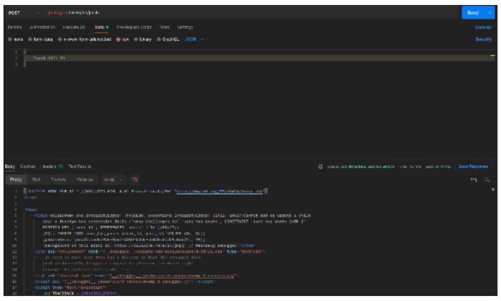
11 – A PUT request with no data is sent to users endpoint.	FR-18	The database is setup. The API is running.	HTTP code 400 and no data provided message returned.	As expected.  PUT {{base_url}}/users/f6871f54-77e6-45da-8a8c-ef4f11c240d5 Body · raw · JSON 400 BAD REQUEST 49 ms 296 B `"message": "No data provided."`	Pass

12 - A PUT request for a non existent user is sent to the API	FR-18	The database is setup.
		
		The API is running.
			HTTP code 404 and user not found message returned.	As expected.
				
				PUT {{base_url}}/users/no
				Body · raw · JSON
				`"bio": "test"`
				404 NOT FOUND 46 ms 292 B
				`"message": "User not found."`
					Pass

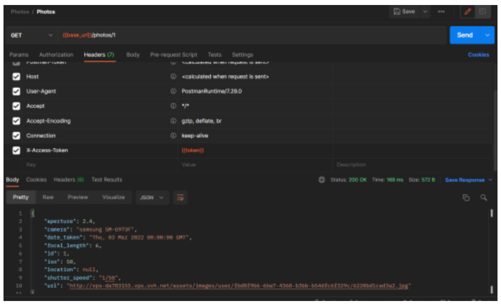
13 – A valid DELETE request is sent to the users endpoint.	FR-18	The database is setup. The API is running. A test user exists.	HTTP code 204 returned and the record is removed from the db.	As expected.  DELETE {{base_url}}/users/f6871f54-77e6-45da-8a8c-ef4f11c240d5 Body · none This request does not have a body 204 NO CONTENT 58 ms 248 B  id · public_id · username · password 1 · fbd8f9b6-6ba7-4368-b3bb-b54dfc6f329c · Jacob · sha256$D7Fauib40tvVfMiK$19643	Pass
14 – A DELETE request is sent to users endpoint for a non-existent user.	FR-18	The database is setup. The API is running.	HTTP code 404 and user not found message returned.	As expected.  DELETE {{base_url}}/users/no Body · none This request does not have a body 404 NOT FOUND 48 ms 292 B `"message": "User not found."`	Pass

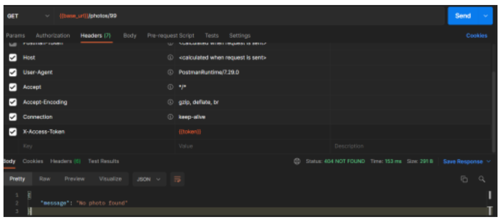
15 – A valid GET request is sent to the Users has Posts endpoint	FR-20	The database is setup. The API is running. A test user exists. A test post exists.	HTTP code 200 and user post data returned.	As expected  GET {{base_url}}/users/fbd8f9b6-6ba7-4368-b3bb-b54dfc6f329c/posts This request does not have a body 200 OK 40 ms 1019 B { "posts": [{ "id": 7, "post_id": 7, "user_id": "fbd8f9b6-6ba7-4368-b3bb-b54dfc6f329c" }, { "id": 6, "post_id": 6, "user_id": "fbd8f9b6-6ba7-4368-b3bb-b54dfc6f329c" }, { "id": 5, "post_id": 5, "user_id": "fbd8f9b6-6ba7-4368-b3bb-b54dfc6f329c"	Pass

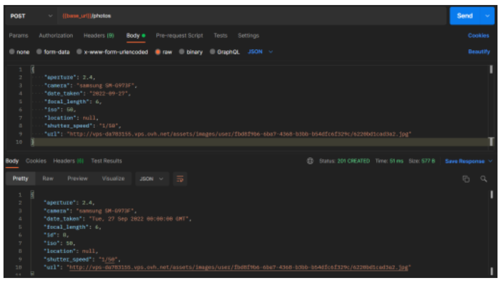
16 – An Invalid GET request is sent to the endpoint Users has Posts endpoint for a user that doesn't exist.	FR-20	The database is setup.
		
		The API is running.
			HTTP Code 404 and an error message stating that the data can't be found.	An empty posts array is returned. Likely caused by a forgotten check to verify the user given exists.  GET {{base_url}}/users/test/posts
				This request does not have a body
				200 OK 85 ms 268 B
				{
				"posts": []
				}
					Fail

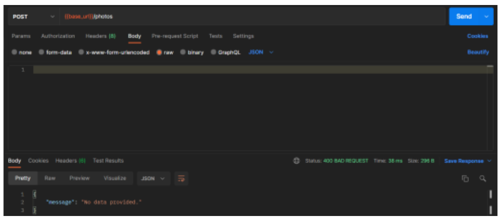
17 – A valid POST request sent to the User Has Posts Endpoint.	FR-20	The database is setup. The API is running. A test user exists. A test post exists.	HTTP Code 201 and user post data returned.	As expected. 	Pass
18 – An Invalid POST request containing no post_id is sent.	FR-20	The database is setup. The API is running. A test user exists.	HTTP Code 400 is returned.	 Code 500 (Internal server error) is	Fail

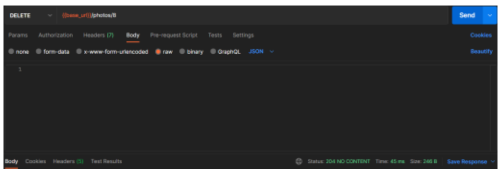
				returned, a check is missing to validate that the post_id exists.	

19 – A valid GET request is sent to the Photos endpoint.	FR-22	The database is setup.
		
		The API is running.
		
		A test photo exists.
			HTTP code 200 and photo data is returned.	As expected.
				
				Photos / Photos
				GET `{{base_url}}/photos/1`
				Host: <calculated when request is sent>
				User-Agent: PostmanRuntime/7.29.0
				Accept: */*
				Accept-Encoding: gzip, deflate, br
				Connection: keep-alive
				X-Access-Token: {{token}}
				Status: 200 OK Time: 168 ms Size: 572 B
				`{`
				`"aperture": 2.4,`
				`"camera": "samsung SM-G973F",`
				`"date_taken": "Thu, 03 Mar 2022 00:00:00 GMT",`
				`"focal_length": 6,`
				`"id": 1,`
				`"iso": 50,`
				`"location": null,`
				`"shutter_speed": "1/50",`
				`"url": "http://vps-da783155.vps.ovh.net/assets/images/user/fbd8f9b6-6ba7-4368-b3bb-b54dfc6f329c/6220bd1cad3a2.jpg"`
					Pass

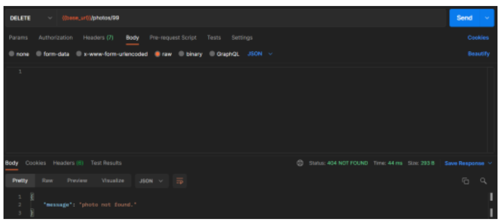
20 – An invalid get request for a photo that doesn't exist is sent.	FR-22	The database is setup.
		
		The API is running.
			HTTP code 404 returned alongside an error message.	As expected.
				
				GET `{{base_url}}/photos/99`
				Host: <calculated when request is sent>
				User-Agent: PostmanRuntime/7.29.0
				Accept: */*
				Accept-Encoding: gzip, deflate, br
				Connection: keep-alive
				X-Access-Token: {{token}}
				Status: 404 NOT FOUND Time: 153 ms Size: 291 B
				`{`
				`"message": "No photo found"`
				`}`
					Pass

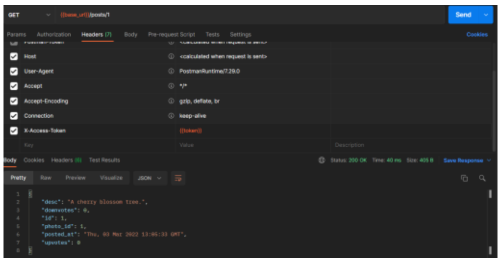
21 – A valid POST request is sent to the Photos endpoint.	FR-22	The database is setup.
		
		The API is running.
			HTTP code 201 and photo data returned.	As expected.
				
				POST `{{base_url}}/photos`
				Body raw JSON
				`{`
				`"aperture": 2.4,`
				`"camera": "samsung SM-G973F",`
				`"date_taken": "2022-09-27",`
				`"focal_length": 6,`
				`"iso": 50,`
				`"location": null,`
				`"shutter_speed": "1/50",`
				`"url": "http://vps-da783155.vps.ovh.net/assets/images/user/fbd8f9b6-6ba7-4368-b3bb-b54dfc6f329c/6220bd1cad3a2.jpg"`
				`}`
				Status: 201 CREATED Time: 51 ms Size: 577 B
				`{`
				`"aperture": 2.4,`
				`"camera": "samsung SM-G973F",`
				`"date_taken": "Tue, 27 Sep 2022 00:00:00 GMT",`
				`"focal_length": 6,`
				`"id": 8,`
				`"iso": 50,`
				`"location": null,`
				`"shutter_speed": "1/50",`
				`"url": "http://vps-da783155.vps.ovh.net/assets/images/user/fbd8f9b6-6ba7-4368-b3bb-b54dfc6f329c/6220bd1cad3a2.jpg"`
					Pass

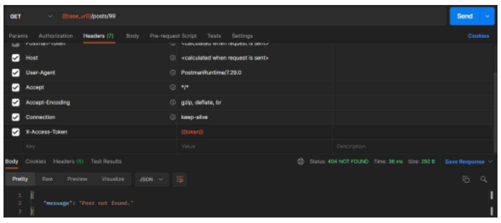
22 – Invalid POST request sent to photos endpoint containing no data.	FR-22	The database is setup.
		
		The API is running.
			HTTP Code 400 returned and error message.	As expected.
				
				POST `{{base_url}}/photos`
				Body raw JSON
				Status: 400 BAD REQUEST Time: 38 ms Size: 298 B
				`{`
				`"message": "No data provided."`
				`}`
					Pass

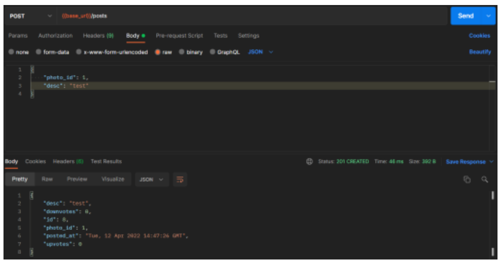
23 – Valid DELETE request sent to photos endpoint.	FR-22	The database is setup. The API is running.	HTTP code 204 returned.	As expected.  DELETE `{{base_url}}/photos/8` Body raw JSON Status: 204 NO CONTENT Time: 45 ms Size: 246 B	Pass

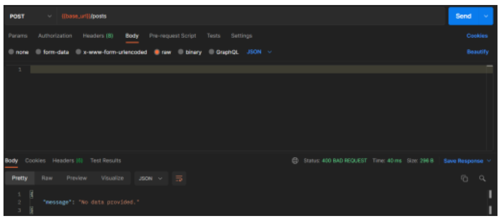
		Test photo is in the system.			

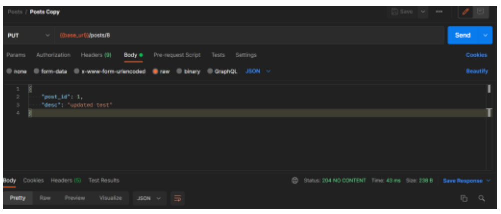
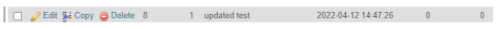
24 – Invalid DELETE request sent for a photo id which doesn't exist.	FR-22	The database is setup.
		
		The API is running.
			HTTP code 404 returned.	As expected.
				
				DELETE {{base_url}}/photos/99
				Params Authorization Headers (7) Body Pre-request Script Tests Settings
				none form-data x-www-form-urlencoded raw binary GraphQL JSON
				Body Cookies Headers (8) Test Results Status: 404 NOT FOUND Time: 44 ms Size: 293 B
				{
				"message": "photo not found."
				}
					Pass

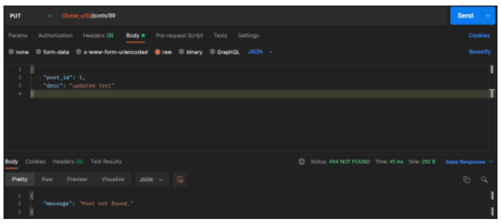
25 – Valid GET request sent to the Posts endpoint.	FR-23	The database is setup.
		
		The API is running.
		
		A post record exists.
			HTTP code 200 and post data returned.	As expected.
				
				GET {{base_url}}/posts/1
				Params Authorization Headers (7) Body Pre-request Script Tests Settings
				Host <calculated when request is sent>
				User-Agent PostmanRuntime/7.29.0
				Accept */*
				Accept-Encoding gzip, deflate, br
				Connection keep-alive
				X-Access-Token {{token}}
				Key Value Description
				Body Cookies Headers (6) Test Results Status: 200 OK Time: 40 ms Size: 405 B
				{
				"desc": "A cherry blossom tree.",
				"downvotes": 0,
				"id": 1,
				"photo_id": 1,
				"posted_at": "Thu, 03 Mar 2022 13:05:33 GMT",
				"upvotes": 0
				}
					Pass

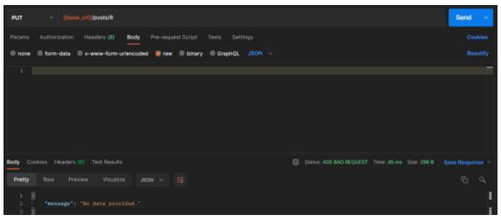
26 – Invalid GET request sent to posts endpoint for record that doesn't exist.	FR-23	The database is setup.
		
		The API is running.
			HTTP code 404 and error message.	As expected.
				
				GET {{base_url}}/posts/99
				Params Authorization Headers (7) Body Pre-request Script Tests Settings
				Host <calculated when request is sent>
				User-Agent PostmanRuntime/7.29.0
				Accept */*
				Accept-Encoding gzip, deflate, br
				Connection keep-alive
				X-Access-Token {{token}}
				Key Value Description
				Body Cookies Headers (6) Test Results Status: 404 NOT FOUND Time: 36 ms Size: 292 B
				{
				"message": "Post not found."
				}
					Pass

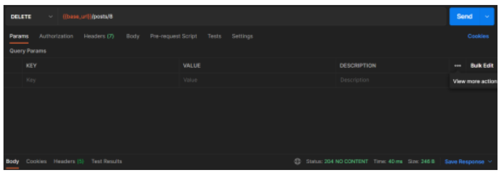
27 – Valid POST request sent to the posts endpoint.	FR-23	The database is setup.
		
		The API is running.
		
		A post exists.
			HTTP code 201 and post data returned.	As expected.
				
				POST {{base_url}}/posts
				Params Authorization Headers (9) Body Pre-request Script Tests Settings
				none form-data x-www-form-urlencoded raw binary GraphQL JSON
				{
				"photo_id": 1,
				"desc": "test"
				}
				Body Cookies Headers (6) Test Results Status: 201 CREATED Time: 46 ms Size: 392 B
				{
				"desc": "test",
				"downvotes": 0,
				"id": 6,
				"photo_id": 1,
				"posted_at": "Tue, 12 Apr 2022 14:47:26 GMT",
				"upvotes": 0
				}
					Pass

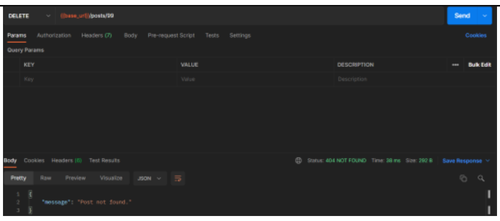
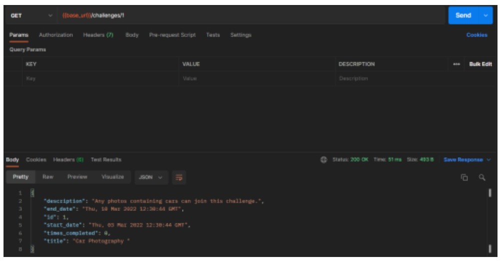
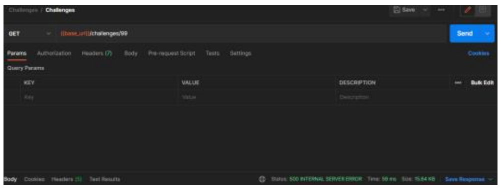
28 – Invalid POST request sent to posts endpoint with no post data.	FR-23	The database is setup.
		
		The API is running.
			HTTP code 400 and error message returned.	As expected.
				
				POST {{base_url}}/posts
				Body raw JSON
				Status: 400 BAD REQUEST Time: 40 ms Size: 296 B
				{
				"message": "No data provided."
				}
					Pass

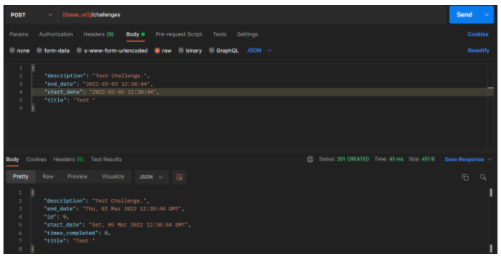
29 – Valid PUT request sent to posts endpoint.	FR-23	The database is setup.
		
		The API is running.
		
		A post record exists.
			HTTP code 204, record on db is updated.	As expected.
				
				Posts / Posts Copy
				PUT {{base_url}}/posts/8
				Body raw JSON
				{
				"post_id": 1,
				"desc": "updated test"
				}
				Status: 204 NO CONTENT Time: 43 ms Size: 238 B
				
				Edit Copy Delete 8 1 updated test 2022-04-12 14:47:26 0 0
					Pass

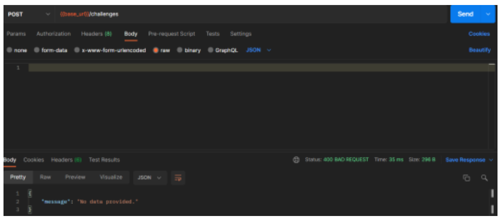
30 – Invalid PUT request sent to the posts endpoint for a post id that doesn't exist.	FR-23	The database is setup.
		
		The API is running.
			HTTP code 404 and error message returned.	As expected.
				
				PUT {{base_url}}/posts/99
				Body raw JSON
				{
				"post_id": 1,
				"desc": "updated test"
				}
				Status: 404 NOT FOUND Time: 41 ms Size: 292 B
				{
				"message": "Post not found."
				}
					Pass

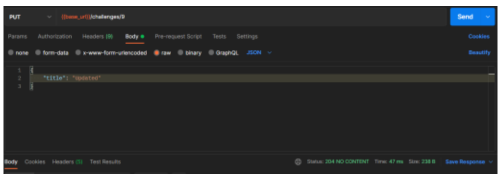
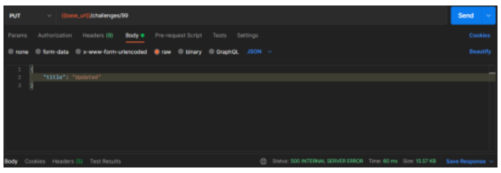
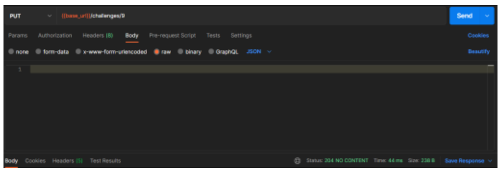
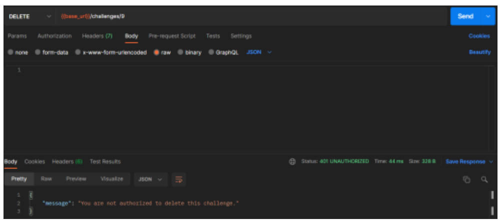
31 – Invalid PUT request containing no data is sent to the posts endpoint.	FR-23	The database is setup.
		
		The API is running.
		
		A post record exists.
			HTTP code 400, and error message returned.	As expected.
				
				PUT {{base_url}}/posts/8
				Body raw JSON
				Status: 400 BAD REQUEST Time: 45 ms Size: 296 B
				{
				"message": "No data provided."
				}
					Pass

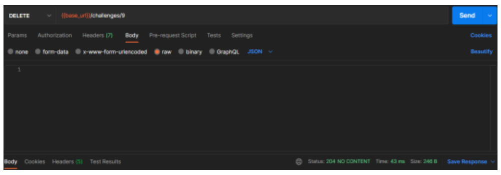
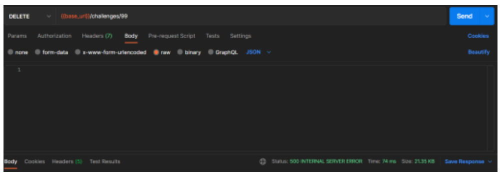
32 – Valid DELETE request sent to the posts endpoint.	FR-23	The database is setup. The API is running. A post record exists.	HTTP code 204 returned.	As expected.  DELETE {{base_url}}/posts/8 Query Params KEY VALUE DESCRIPTION Key Value Description Status: 204 NO CONTENT Time: 40 ms Size: 246 B	Pass
33 – Invalid DELETE	FR-23	The database is setup.	HTTP code 404	As expected.	Pass

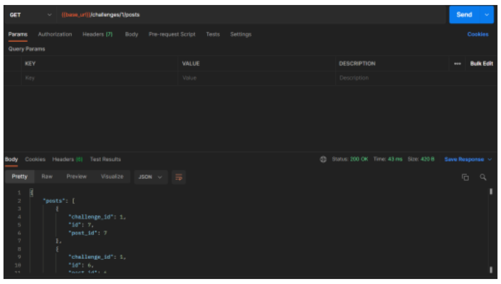
request sent to posts endpoint for a post id that doesn't exist.		The API is running.	and an error message returned.	 DELETE `{{base_url}}/posts/99` Status: 404 NOT FOUND Time: 38 ms Size: 292 B `{ "message": "Post not found." }`	
34 – Valid GET request sent to the challenges endpoint.	FR-24	The database is setup. The API is running. A challenge record exists.	HTTP code 200 and challenge data returned.	As expected.  GET `{{base_url}}/challenges/1` Status: 200 OK Time: 51 ms Size: 493 B `{ "description": "Any photos containing cars can join this challenge.", "end_date": "Thu, 10 Mar 2022 12:38:44 GMT", "id": 1, "start_date": "Thu, 03 Mar 2022 12:38:44 GMT", "times_completed": 0, "title": "Car Photography " }`	Pass
35 – Invalid GET request sent to the challenges endpoint for a challenge id that doesn't exist.	FR-24	The database is setup. The API is running.	HTTP code 404 and error message returned.	Code 500, a validation check to verify the id given exists is missing.  Challenges / Challenges GET `{{base_url}}/challenges/99` Status: 500 INTERNAL SERVER ERROR Time: 58 ms Size: 15.64 KB	Fail

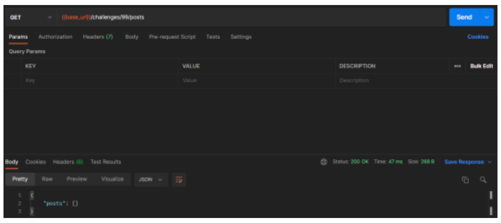
36 – Valid POST request sent to the Challenges endpoint.	Fr-24	The database is setup.
		The API is running.
			HTTP code 201 and challenge data returned.	As expected.
				
				POST `{{base_url}}/challenges`
				`{ "description": "Test Challenge.", "end_date": "2022-03-03 12:30:44", "start_date": "2022-03-05 12:30:44", "title": "Test " }`
				Status: 201 CREATED Time: 45 ms Size: 451 B
				`{ "description": "Test Challenge.", "end_date": "Thu, 03 Mar 2022 12:30:44 GMT", "id": 9, "start_date": "Sat, 05 Mar 2022 12:30:44 GMT", "times_completed": 0, "title": "Test " }`
					Pass

37 – Invalid empty POST request sent to the challenges endpoint.	FR-24	The database is setup. The API is running.	HTTP code 400 and error message returned.	As expected.  POST `{{base_url}}/challenges` Status: 400 BAD REQUEST Time: 35 ms Size: 296 B `{ "message": "No data provided." }`	Pass

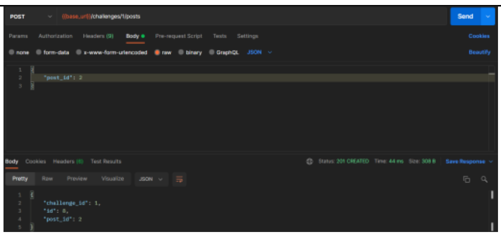
38 – Valid PUT request sent to the challenges endpoint.	FR-24	The database is setup. The API is running. A challenge record exists.	HTTP code 204 returned.	As expected.  PUT {{base_url}}/challenges/9 `{ "title": "Updated" }` Status: 204 NO CONTENT Time: 47 ms Size: 238 B	Pass
39 – Invalid PUT request sent to challenges endpoint for challenge id that doesn't exist.	FR-24	The database is setup. The API is running.	HTTP code 404 and error message returned.	Code 500, a validation check to ensure the id given is valid is missing.  PUT {{base_url}}/challenges/99 `{ "title": "Updated" }` Status: 500 INTERNAL SERVER ERROR Time: 60 ms Size: 15.57 KB	Fail
40 – Invalid PUT request sent to challenge endpoint containing no data.	FR-24	The database is setup. The API is running. A challenge record exists.	HTTP code 400 and error message returned.	Code 204, returned, this is a result of a change to the API since the requirements were defined, PUT requests now populate required empty fields with the previous data. So there was nbo change in the data stored.  PUT {{base_url}}/challenges/9 Status: 204 NO CONTENT Time: 44 ms Size: 238 B	Fail
41 – Valid DELETE request sent to the challenge endpoint. From a non admin user account.	FR-24	The database is setup. The API is running. A challenge record exists.	HTTP code 401 and unauthorized message.	As expected.  DELETE {{base_url}}/challenges/9 Status: 401 UNAUTHORIZED Time: 44 ms Size: 328 B `{ "message": "You are not authorized to delete this challenge." }`	Pass

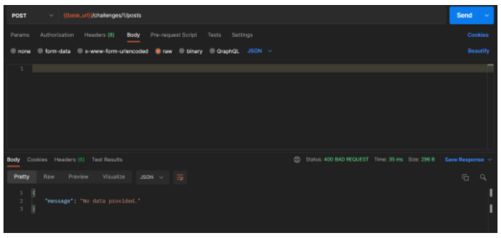
42 – Valid DELETE request sent to the challenge endpoint. From an admin user account.	FR-24	The database is setup. The API is running. A challenge record exists.	HTTP code 204.	As expected.  DELETE {{base_url}}/challenges/9 none form-data x-www-form-urlencoded raw binary GraphQL JSON `1` Status: 204 NO CONTENT Time: 43 ms Size: 246 B	Pass
43 – Invalid delete request sent to the challenges endpoint for a challenge id that doesn't exist.	FR-24	The database is setup. The API is running.	HTTP code 404 and error message.	Code 500, validation of id missing.  DELETE {{base_url}}/challenges/99 none form-data x-www-form-urlencoded raw binary GraphQL JSON `1` Status: 500 INTERNAL SERVER ERROR Time: 74 ms Size: 21.35 KB	Fail

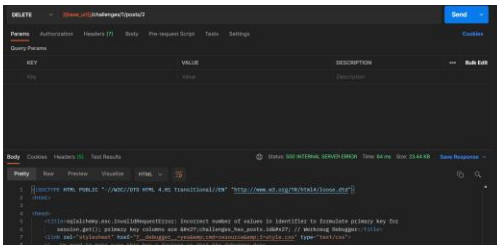
44 – Valid GET request sent to the challenge has posts endpoint.	FR-25	The database is setup.
		
		The API is running.
		
		A challenge post exists.
			Code 200 and challenge post data.	As expected.
				
				GET {{base_url}}/challenges/1/posts
				Status: 200 OK Time: 43 ms Size: 420 B
				`{`
				`  "posts": [`
				`    {`
				`      "challenge_id": 1,`
				`      "id": 7,`
				`      "post_id": 7`
				`    },`
				`    {`
				`      "challenge_id": 1,`
				`      "id": 6,`
					Pass

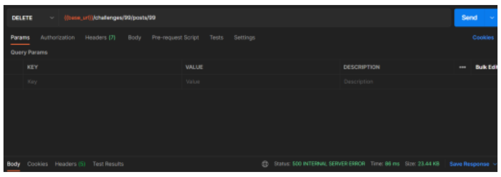
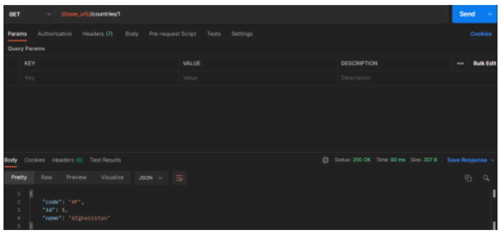
45 – Invalid GET request sent to challenge has posts endpoint for a challenge id that doesn't exist.	FR-25	The database is setup.
		
		The API is running.
			Code 404 and error message.	Code 200 and empty post data returned. Missing validation check on the challenge id.
				
				GET {{base_url}}/challenges/99/posts
				Status: 200 OK Time: 47 ms Size: 268 B
				`{`
				`  "posts": []`
				`}`
					Fail

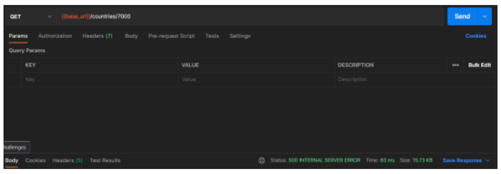
46 – Valid POST request	FR-25	The database is setup.	Code 201 and challenge	As expected.	Pass

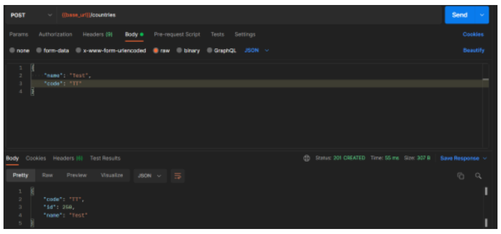
sent to the challenge has posts endpoint.		The API is running.
		
		A challenge exists.
		
		A post exists.
			post data returned.	
				POST {{base_url}}/challenges/1/posts
				`{ "post_id": 2 }`
				Status: 201 CREATED Time: 44 ms Size: 308 B
				`{ "challenge_id": 1, "id": 8, "post_id": 2 }`	

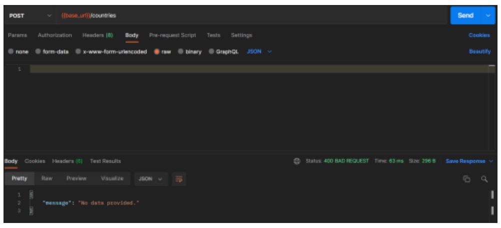
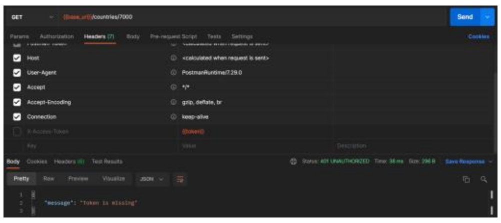
47 – Invalid POST request containing no data sent to the challenge has posts endpoint.	FR-25	The database is setup. The API is running.	Code 400 and error message returned.	As expected.  POST {{base_url}}/challenges/1/posts Status: 400 BAD REQUEST Time: 35 ms Size: 296 B `{ "message": "No data provided." }`	Pass

48 – Valid DELETE request sent to the challenge has posts endpoint.	FR-25	The database is setup.
		
		The API is running.
		
		A challenge post exists.
			Code 204 returned.	Code 500, The values used to get the record to delete are incorrect, this should be changed to use the challenge post id instead of two separate ids.
				
				DELETE {{base_url}}/challenges/1/posts/2
				Status: 500 INTERNAL SERVER ERROR Time: 64 ms Size: 23.44 KB
				`<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">`
				`<html>`
				`<head>`
				`<title>sqlalchemy.exc.InvalidRequestError: Incorrect number of values in identifier to formulate primary key for session.get(); primary key columns are &#x27;challenges_has_posts.id&#x27; // Werkzeug Debugger</title>`
				`<link rel="stylesheet" href="?__debugger__=yes&amp;cmd=resource&amp;f=style.css" type="text/css">`
					Fail

49 – Invalid DELETE request sent to the challenge has posts endpoint.	FR-25	The database is setup. The API is running.	Code 404 returned.	Code 500, missing validation check on id.  DELETE {{base_url}}/challenges/99/posts/99 Status: 500 INTERNAL SERVER ERROR Time: 86 ms Size: 23.44 KB	Fail
50 – Valid GET request sent to the countries endpoint.	FR-26	The database is setup. The API is running. The country table is populated.	Code 200 and country data returned.	As expected.  GET {{base_url}}/countries/1 Status: 200 OK Time: 80 ms Size: 307 B `{ "code": "AF", "id": 1, "name": "Afghanistan" }`	Pass

51 – Invalid GET request sent to countries endpoint for country id that doesn’t exist.	FR-26	The database is setup. The API is running.	Code 404 and error message returned.	Code 500, missing validation check on the id.  GET {{base_url}}/countries/7000 Query Params: KEY / VALUE / DESCRIPTION Status: 500 INTERNAL SERVER ERROR Time: 63 ms Size: 15.73 KB	Fail

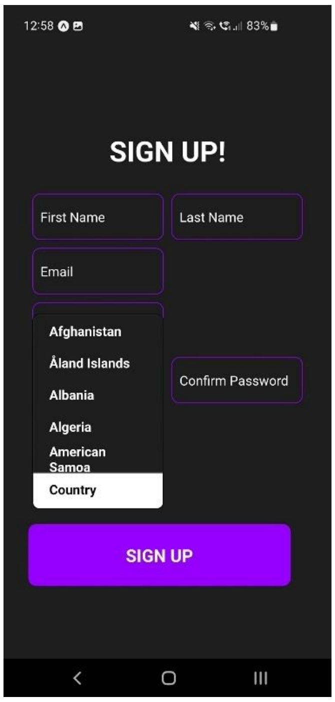
52 – Valid POST request sent to countries endpoint.	FR-26	The database is setup.
		
		The API is running.
			Code 201 and country data returned.	As expected.
				
				POST {{base_url}}/countries
				Body
				`{ "name": "Test", "code": "TT" }`
				Status: 201 CREATED Time: 55 ms Size: 307 B
				`{ "code": "TT", "id": 260, "name": "Test" }`
					Pass

53 – Invalid POST request containing to data sent to countries endpoint.	FR-26	The database is setup. The API is running.	Code 400 and error message returned.	As expected.  POST {{base_url}}/countries Body Status: 400 BAD REQUEST Time: 63 ms Size: 296 B `{ "message": "No data provided." }`	Pass
54 – Most API endpoints require the user to be authorized via a JWT sent alongside their request in the X-Access-Token header, if this is not present the API should return 401	N/A	The database is setup. The API is running.	Code 401 and unauthorized message returned.	As expected.  GET {{base_url}}/countries/7000 Headers (7): ✓ Host: <calculated when request is sent> ✓ User-Agent: PostmanRuntime/7.29.0 ✓ Accept: */* ✓ Accept-Encoding: gzip, deflate, br ✓ Connection: keep-alive ☐ X-Access-Token: {{token}} Status: 401 UNAUTHORIZED Time: 38 ms Size: 296 B `{ "message": "Token is missing" }`	Pass

unauthorized.					

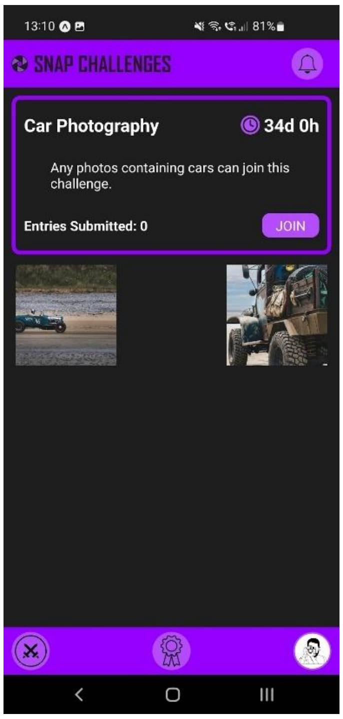
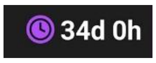
SIGNUP PAGE:

Table 48 - Appendix Signup Page FR Testing

Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed ?
64 – A dropdown list prefilled with all countries on the signup page.	FR-46	None	On click of the countries field a dropdown box is shown prepopulated with all the countries in countries.json. countries.json	As expected.  12:58 83% SIGN UP! First Name Last Name Email Afghanistan Åland Islands Albania Algeria American Samoa Country Confirm Password SIGN UP	Pass
65 – User is redirected to login page after signup.	FR-47	User has successfully signed up.	User is navigated to login page.	As expected.	Pass

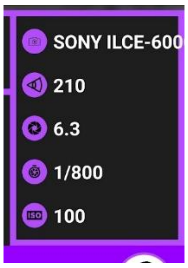
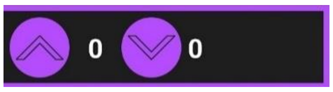
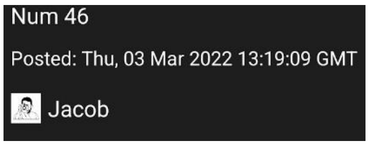
CHALLENGES PAGE:

Table 49 - Appendix Challenges Page FR Testing

Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed ?
68 – Onclick of a challenge more details and posts are shown for that challenge.	FR-50	User is logged in and challenges exist.	A challenge details page is shown showing information about the challenge and the current posts for that challenge.	As expected.  13:10 81% SNAP CHALLENGES Car Photography 34d 0h Any photos containing cars can join this challenge. Entries Submitted: 0 JOIN	Pass
69 – Onclick of the join button next to the challenge the user should be redirected to the post upload page.	FR-51	User is logged in and challenges exist.	User is redirected to post upload page.	As expected.	Pass
70 – A timer should be shown to indicate the time left on a challenge.	FR-52	User is logged in and challenges exist.	A timer is shown.	As expected.  34d 0h	Pass

POST PAGE:

Table 50 - Appendix Post Page FR Testing

Test case	Requirements Tested	Preconditions	Expected Result	Actual Result	Passed?
75 – Camera exif metadata should be shown alongside images.	FR-68	User is logged in post exists.	The exif data should be visible in the bottom left of the page.	Data is indeed shown in the correct place, however, there is some overlap on longer details such as camera name. May require a font size change.  SONY ILCE-600 210 6.3 1/800 100	Pass
76 – Kudos and Downvotes are shown.	FR-69	User is logged in post exists.	Kudos and downvotes should be visible just under the photo.	As expected.  0 0	Pass
77 – Time posted is shown.	FR-70	User is logged in post exists.	Time posted should be shown underneath image description.	As expected.  Num 46 Posted: Thu, 03 Mar 2022 13:19:09 GMT Jacob	Pass

NFR TESTING:

API:

Table 51 - Appendix API NFR Testing

Test case	**Requirements Tested**	**Preconditions**	**Expected Result**	**Actual Result**	**Passed?**
80 – The API is accessible outside of localhost.	NFR-3	The network connection is port forwarded to allow access to port 80 and 443, the HTTP and HTTPS ports which the API runs on.	API calls can be made from external networks.	As expected.	Pass
81 – The API is hosted on a Virtual Private Server (VPS) to allow for better uptime.	NFR-4	None	Accessibility to the VPS will be reliable.	The VPS is hosted with OVH and as of yet has experienced no unexpected downtime in the past 3 months.	Pass

CDN:

Table 52 - Appendix CDN NFR Testing

Test case	**Requirements Tested**	**Preconditions**	**Expected Result**	**Actual Result**	**Passed?**
83 – The CDN is accessible outside of localhost.	NFR-6	The network connection is port forwarded to allow access to port 80 and 443, the HTTP and HTTPS ports which the CDN runs on.	CDN Calls can be made from external networks.	As expected.	Pass
84 – The CDN is hosted on a Virtual Private Server (VPS) to allow for better uptime.	NFR-7	None	Accessibility to the VPS will be reliable.	The VPS is hosted with OVH and as of yet has experienced no unexpected downtime in the past 3 months.	Pass

Appendix C - GitHub Repositories:

Code: **https://github.com/JacobA2000/Snap-Challenges**

Related Content: **https://github.com/JacobA2000/UWE-DSP**