Software Engineering Group Project

COMP2002 / G52GRP: Final Report

Project information:

Project title: FindAR

Academic supervisor: Damla Kilic

Industry Sponsor: John McNamara

Team information:

Team number: 29

(Vishal, Pittala, 20366311, psyvp2)

(Alfie, Rushby, 20337059, psyar13)

(Alan, Stephen, 20358814, psyas15)

(Oliver, Davy-Bowker, 20292673, efyod1)

(Aidan, Cowtan, 20369917, psyac10)

(Rodion, Rasulov, 20374053, psyrr4)

(Alexander, Lockey, 20355597, psyal6)

Documentation (links):

Trello board: Trello

Code repository: Repo

Document repository: Documents

Further Online Documentation (if applicable): Sprint Documentation, Brief

Contents

Part 1: Final Report

All google docs referenced will have a copy in the **documentation** branch in our repository under the **Main Report Documents & Documentation** folders.

Project Background & Understanding

A common problem within society, especially within recent years, is that people are becoming more and more forgetful. Nowadays, there are so many items, tasks, ideas etc that we try to remember, and it is difficult to keep track of everything.

The aim for this project is to provide a solution for those who are forgetful either naturally or due to a medical condition to locate items they use regularly. The goal, is to place tags for items you forget, and find them in AR.

I.e., we need to make some app that can let users easily record where their items are in real life, via tags, and then find them in AR in case they forget these tags.

After meeting with the supervisor from our industry sponsor IBM to discuss our approach for this project, we created several themes as a base for our development:

"*As a user I want to manage my tags, so I know what items I have stored*" relates to the user being able to place tags in augmented reality.
"*As a user I want a visual AR indication of where my tag is, so I know where my items are.*" so that the app knows to activate that tag again relates to the user being able to place the tag back down once they have returned the item.
"*As a user I want to be able to manage my account, to log in and use the service*" relates to the user being able to search through their tags

We came up with these themes through envisioning ourselves as the user carrying out a task according to the given brief. See Personas & Scenarios. After this, we identified the key features we felt would be most useful to us and would improve our experience with the application.

The general method of deriving these themes was the use of designing story maps in several meetings with the whole team, outlining the direction of our project. It is important to demarcate the project into parallel themes, and from this we identified the obvious three: Tag Management in the user interface, AR for finding tags, and User Management for standard app functionality.

IBM wanted the ability to use AR to find things you typically lose. Our industry sponsor stated in a meeting that the ability to 'pick up' and 'place' tags was an important key, with a reminder system to tell you if you have a 'picked up' tag to place it back down.

I.e.

- A system to manage the ‘data of tags’.
- A system to ‘find the tags in AR’.
- And an authentication system to uniquely identify every user’s tags.

We placed ourselves in the shoes of the user and thought of the logical steps the user would take while also noting the things not present in the brief but we thought would improve the app’s usability.

After brainstorming similar products that we knew of, as well as research into AR location applications we found two examples that we thought best encompassed the key attributes we hope to implement into FindAR.

- One such example is Apple's AirTag which uses a physical tag to send location data to an app which then guides the user to the tag using a virtual compass arrow. However, this requires the user to purchase a physical tag and place it in the world whereas our product will be fully virtual and won't require a payment to add a new tag.
- Another app more like ours is Wikitude Drive which is a car GPS app that accesses the phone's camera to implement an AR path to guide the user to their location. Despite the similarity Wikitude Drive has a different goal in mind than FindAR does. Its primary focus is mapping a user’s journey to a certain location whereas our primary focus is to help guide our users to objects that they’ve placed and tagged. It is designed to be used on the roads whereas FindAR is designed to be used in more domestic settings.

We decided on Apple’s AirTag as a reference point as we were familiar with the product and its applications in the real world.

We also chose Wikitude Drive as we felt that it’s implementation of AR was close to what we envisioned for our application

Going forward these examples will be important reference points to our project.

Personas & Scenarios

We also decided to create some personas and scenarios to help us align our goals with the app's core audience:

	Name: Susanna **Age:** 66 **Occupation:** Retired	**Motivation:** Due to old age Susanna's memory has worsened and she is unable to effectively remember where she last placed her personal items. Especially her medication for her liver. She needs a method to help her find her medication when she needs to take it. **Tech Experience:** Not very literate, knows how to access a few websites & apps on her iPad. Usually asks her children for help.
	Name: Dan **Age:** 45 **Occupation:** Unemployed	**Motivation:** Dan was in an accident a couple of years ago in which he sustained a head injury. Since then he has trouble keeping track of items such as his wallet and keys. He would like assistance in locating these items after he has put them down. **Tech Experience:** Literate, Has extensive experience with smartphones thought due to his accident his efficiency using such devices has decreased.
	Name: Julia **Age:** 53 **Occupation:** School Teacher	**Motivation:** Julia has never had the best memory and frequently misplaces important documents such as exams. In order to get better at swiftly giving feedback on recent assignments. She would like a method which would help her better locate papers she has marked in order to return them to her students. **Tech Experience:** Moderately literate, frequently uses technology in daily life but sometimes requires a little help with new software & features.

These 3 personas represent an expected range of ability. And from this:

Scenario 1:

Susanna has just finished making dinner for her and her husband. Usually she takes her liver medication with food but its not where she last remembers leaving it.

It is critical that she takes the medication at the same time every day for it to be effective. Usually she leaves on the chest of drawers in the dining room but it is not there.

Due to the medication being in simple plastic pill organiser it is not easy to locate just by looking around the house. Especially with her decreasing eyesight.

Scenario 2:

Dan has to go to the hospital for his bi-monthly check up after his accident but he has misplaced his keys.

He needs his keys in order to get back into his house after his appointment so it is very important he has them before he leaves.

His keys only consist of a few keys and a novelty key ring which doesn't make them stand out among the clutter in his house. Also they are not hung up by the door where he usually leaves them due to hearing stories where thieves enter homes using keys left next to doors.

Scenario 3:

Julia's class has a big assignment coming up and to prepare for it she instructed them to complete a mock essay which she will mark and give feedback.

After collecting them in she put them down somewhere before her next class but she cannot remember where she left them. They could be anywhere in her classroom, the staff room or even at home.

The class was promised feedback by the end of the week so it is imperative she marks and returns them in a swift manner.

We can derive scenarios where our app may prove useful.

Web-Application vs Mobile-Application

Whilst setting out the requirements for the project, as a team we explored and discussed what the best methods to approach it would be. For example, in terms of the platform, we were torn between producing the solution on either a mobile application or a web application.

After reading an article on the advantages and disadvantages of a web app vs a mobile app, the main points we found were that with a web app we have the advantage of the solution being universally available on any device, whereas if we were to create a mobile app we would have to aim to produce the solution for either Apple or Android. However, one key advantage of a mobile app is that it can be used offline, although this approach would be more expensive to build and maintain. However, we decided that the advantages of the mobile app are not worth the costs. They hold a much greater audience and are not tied to a single platform like an app must be.

A mobile app would be impractical for us, as a team, to work on since we do not have any experience within that field. We explored the possibility of learning mobile app development, however within our given time frame for the project, it would not be a viable solution. In the end, we decided that to fulfil the criteria of the project to the best of our abilities and within the given time frame, the most logical approach would be to work on a web app.

Speech-to-Text

Another area where we had to introduce some limitations with the scope of our project was deciding whether to have a speech-to-text solution that links with IBM's Watson AI to allow the user to "say" what item they are looking for and the program would understand it.

This comes with a lot of limitations. Voice to action is something we have no experience in, and therefore cannot do a proper risk assessment in its feasibilities without attempting it. This, along with the large unknown at the accuracy and efficacy of the AR solution led us to push this feature as an option if development on more important areas were finished.

Some questions to note on using this system is:

- What is the accuracy of the voice recognition (Accuracy of other voice inputs)?
- Will this feature be used when a reliable input of touch is available?
- With a voice method of input, what reliable way would it be integrated into a website?

An important idea in risk management is to know the risk. To implement a voice method of input, we'd first need a touch method as a baseline. It would not be in the interest of the user to only have voice to interact in the app, where if the recognition software does not work with their accent, it would prove frustrating to use.

Thus, this option comes as a 'next level input' after touch. We have not dismissed the idea of a speech-to-text aspect as we plan to assess the idea of including it after our first prototype has been complete, where we will assess what changes and improvements need to be made.

Hardware Requirements

For this project no specific hardware tools are needed for it to be functional or to test it, due to our decision to implement this project via a web app. Any device with access to the internet, an internet browser, and camera can access and use the application.

Selecting our server

As we're creating a web app, choice of server was something we needed to consider, our initial thoughts were to find an external party to host the server and apply for funding, but we soon realised that IBM provided their own server hosting software that we could utilise, we quickly settled on using that due it's low cost for us and the possibilities of superior integration with IBM Watson in the future.

Plans for the front-end

Our initial plans for the front-end of our program were to use pure html/css but after further research into web development we found out it was a common practice to be using a web framework. Due to the vast ecosystem of web frameworks, we shortlisted 3:

- Angular
- React
- Svelte

We decided that angular and react were both too bloated and excessive for what we needed. We also found out that both are extremely difficult to learn, this does not forebode well for a group inexperienced in web development. This led to a quick decision to use svelte, a lightweight and easy to use web framework. It has a high popularity within developers (Stack Overflow Developer Survey 2021), so we found that it may be wise to follow those who have more experience in the field.

Selecting our database

Another decision we had to consider was the choice of database to store user usernames, passwords hash codes and tag information. Fortunately, this decision was easy to resolve as IBM provided their own database system called DB2, we picked it in the hopes that it would be simple to integrate with our IBM server.

AR Libraries

Our decision to choose to create a web app led to us having access to a developing ecosystem of libraries. We considered the use of Three.js, a library which allowed the rendering of 3D graphics on a webpage using WebGL, this would allow us to render our own AR tokens on top of the incoming camera feed. We also considered the option to use AR.js or Argon.js, Argon.js was much more specialised than something like three.js with having many tools already in place to develop an AR application, AR.js had a similar methodology except more extreme with having many typical solutions for AR problems already built in, like location based tokens.

Requirements & Critical Analysis

Identifier	**Functional** Requirements	**Non-Functional Requirements**
R1	The application must show a token in AR representing the location of a token the user marked.	The AR Token must be accurate, the token must be no further than 10 meters from the actual location of the tagged item
R2	The application must allow the user to record the location of a tag.	The recorded location of the tag must be accurate to 5 meters.
R2 cont.		Recording the location of the tag must take no longer than 5 seconds.
R3	The user must be able to record a tag as being on their person	The act of doing so must take no longer than 5 seconds, in terms of API calls.
R4	The user must be able to view the tags he has picked up.	
R5	The user must be able to view the tags he has put down	
R6	The user should be able to create a new tag.	The act of doing so should take no longer than 5 seconds
R7	The user should be able to write a description for the tag on creation	
R8	The user should be able to specify an icon for the tag on creation.	
R9	The user must be able to create an account	Account creation process must limit insecure passwords and already usernames.
R9 cont.		Act of submitting account creation form should take no longer than 5 seconds to create account
R10	The user must be able to login in to his account	Users must be redirected to the homepage of the website after 5 seconds upon logging in.
R11	The user must have his tags paired to his account.	
R12	The user must be able to login from any browser with an internet connection	
R13	The user must be able to view his tags from any browser on any mobile system.	
R14	The user must be able to edit the location of a created tag.	Recording of new location must comply with non-functional requirements of R2
R15	The user must be able to edit of a created tag	Must comply with non-functional requirements of R7

R16	The user must be able to delete a created tag.	Should stop appearing with the other tags no longer than 5 seconds after deletion
R16 cont.		Information about the tag should be deleted from any database storing it.
R17	The user should be able to set a tag as being not on their person	The act of doing so should take no longer than 5 seconds.
R18	The AR token should get larger as the user gets closer to the actual tagged item.	Must comply with the accuracy requirements of R2
R19	The user should be able to change their password.	Must comply with the password requirements of R9.
R20	The user should be able to change their email.	Must comply with non-functional requirements of R9.
R21	The user should be able to easily navigate the website	

Products' Visual Design

The user interface is critical to the use of FindAR, so it is important that it was done correctly.

Before the initial design was conceptualised, we carried out research into what makes a good UI in general and specifically what makes a good UI for elderly people. During the research we discovered 7 key aspects to UI design:

- Simplicity
- Consistency
- Purposeful layout
- Strategic use of colour & texture
- Use of typography
- User communication
- Defaults

These were used throughout the entire process when constructing our interface and will be used as a set of heuristics during a heuristic evaluation of our product.

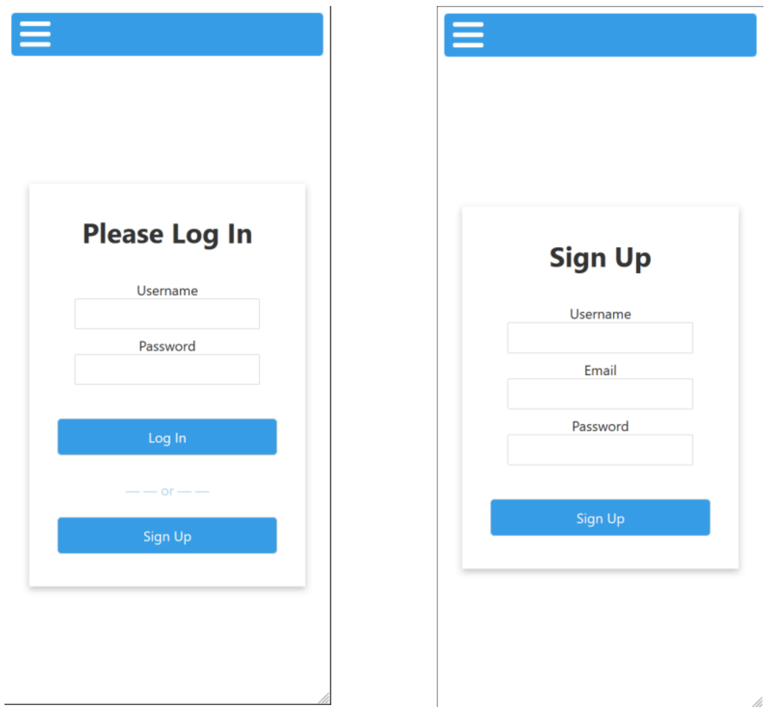
Key UI design aspects: User Interface Design Basics | Usability.gov
Our findings from this research (*see repo documentation branch for copy*): What Makes A Good UI

Simplicity

The end goal was to produce a minimalistic product, and this is clearly visible throughout the design. The interface clearly and succinctly indicates what it is meant to do by only including elements that aid the completion of the user's task.

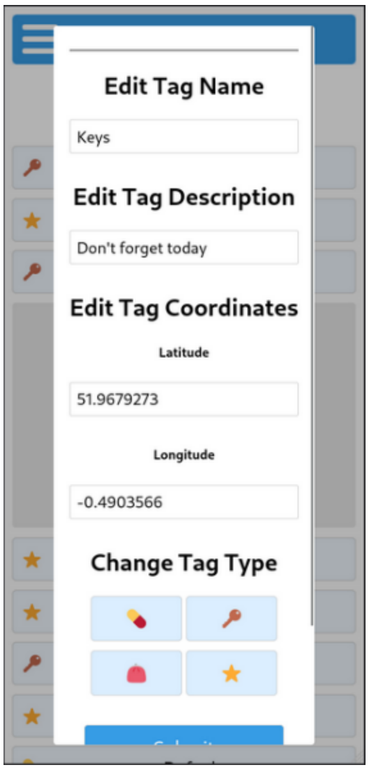
One example of this is on the "Log In" and "Sign Up" screens.

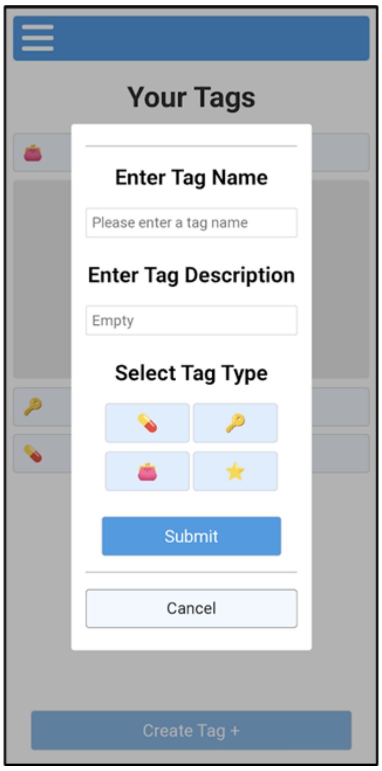
These screens perfectly display the number one goal in our design, guidance through simplicity. Here the required actions are as clear as possible to aid the user with their desired goal without confusing them or overloading them with information.



The Log In and Sign up screens

The simplicity of the design was slightly endangered in the “Edit Tag” screen with the introduction of editing the tag's coordinates. This feature did not harm the streamlined nature of the interface but was argued that it did not conform to the minimalistic structure as it could be deemed unnecessary to most users. Therefore, we removed it from the ‘add tags’ feature as few users would need to specify their coordinates on addition.



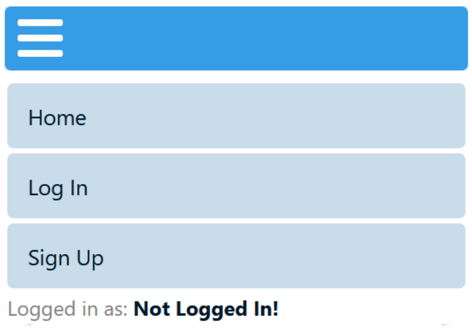


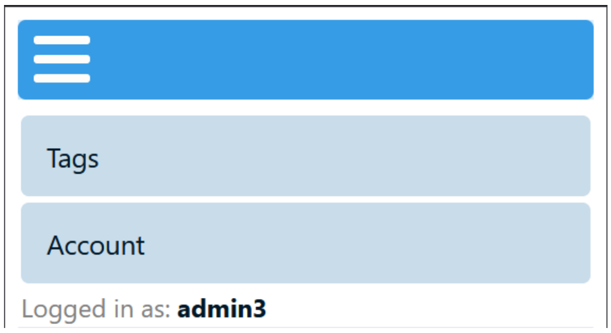
Edit Tags and Add Tags screens

Consistency

The design values consistency above all else and this is reflected in the screenshots above. Every piece of text, button and input boxes have the same design paired with consistent centre alignment throughout the entire design. This was achieved using a singular CSS file which defined the design of all the elements. By linking this into every page we ensured that the design in each page would be consistent. By keeping everything consistent it enables the user to feel more confident using the app while simultaneously increasing the rate at which they learn the app.

Another application of this is the method of navigation.





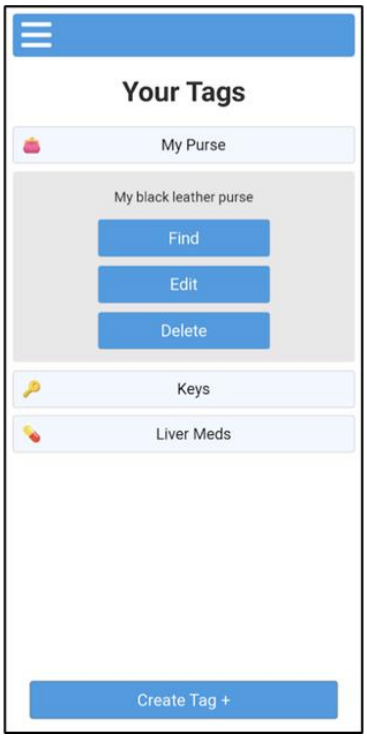
Menu Navigation, left logged out, right logged in

No matter what area the user is in, this menu will always exist on top of the app, and dynamically changes depending on whether they are logged in or not. It displays their username as a helpful reminder in case multiple people use the device.

Purposeful Layout

This relates to the intentional placing of elements to help draw attention to the most important elements. The main way our design achieves this is using central alignment and spacing.

By grouping all elements in the centre of the screen the user's eye is naturally drawn to them. Furthermore, elements with similar importance have consistent spacing between them so that the user interprets them as a group and understands that they are somewhat related. A key example of this is the tags vs their action buttons. The tags have consistent spacing between one another yet when expanded their action buttons are intentionally spaced further apart from the other tags. This is to subconsciously indicate their difference in purpose.

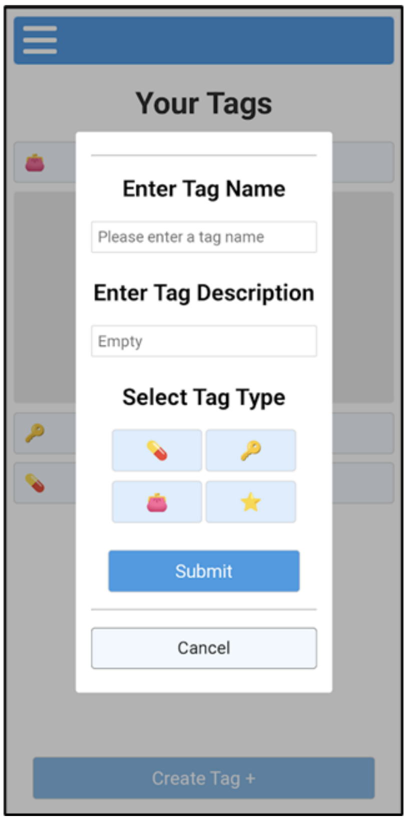


Example spacing to subconsciously group buttons

Strategic Use of Colour & Texture

Colour was key throughout the design process, especially when it comes to colour reception among the older population and colour relating to memory retention. When a person ages their ability to differentiate between low contrast colours greatly decreases as well as the increasing confusion between shades of red and green. We conducted further research (*See repo documentation branch for copy*) into this and created a colour scheme document to aid us throughout the process. We determined that blue was the best colour to use throughout due to its calming nature and lack of degradation as a person's vision ages. Blue also aids cognitive retention which is closely tied in with the goal of our product.
The background colour was chosen to be white as it gives the best contrast to the blue, ensuring the buttons will always stand out no matter their age or vision level.

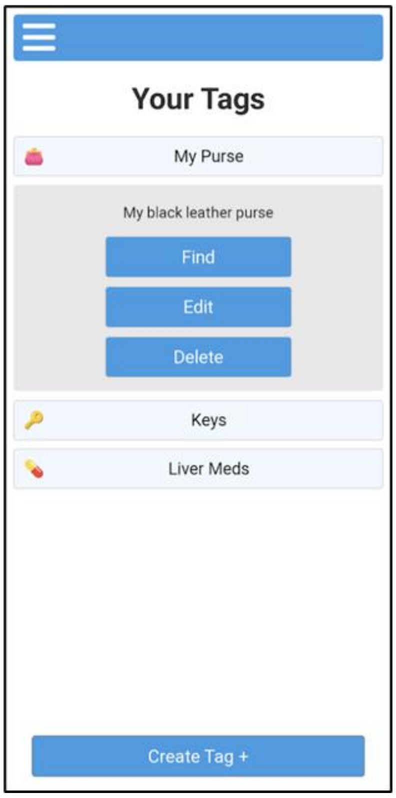
A lighter shade of blue is used for certain features such as the tag buttons and the icon selection buttons, this could be an issue due to the reduced contrast between the button and the background. Meaning that someone less able to distinguish the difference may be slightly unclear about the button boundaries. However, this is accounted for with the use of text to clearly indicate to the user where they should press for a certain feature. As most users tend to directly click on text rather than button boundaries.



Example of the lighter blue shade for the tag icon buttons

Use Of Typography

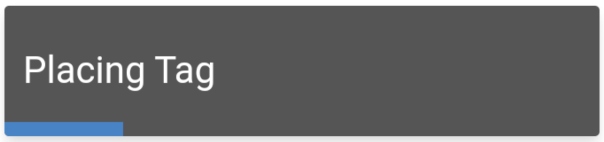
This relates to the selection of font type and size. The font chosen for the interface is a part of the San's Serif font family. This family was intentionally chosen for its readability and simplicity which helps reduce the amount of cognitive effort the user must use when navigating our interface. This font also contributes towards our overall idea of minimalism. We further use variation in font size to help indicate the importance of certain pieces of text. One clear example is that header text is enlarged to quickly communicate what the related page/section is for while the subsections employ a smaller font size. This supports the idea of cognitive chunking which aids in information digestion and retention.



Example of employing different font sizes to indicate importance

User Communication

User feedback is integral to increase user confidence when using an interface. We achieve this through “toast” pop ups. These are small rectangular boxes that contain text. They generally appear when the user has done something and the server responds, such as logging in, the user will receive feedback on whether their log in attempt was successful or not.
These pop ups are further employed when the product needs a bit of time to set key data such as location. The pop up will confirm to the user that they did the right thing, and that the system just needs a bit more time. This positive reinforcement is integral especially when the product is designed for those who might not be as technically literate and could be easily confused when the system does not do as they expect.


Placing Tag

Example of the “toast” pop ups

Defaults

This refers to anticipating the goals the user brings to the app and designing around them. The idea is to “default” around these goals to help speed up the process. The main goal with this app is to find placed items. Upon further communication with the sponsor, it was discussed that the main use case for this was medication as the app would typically be marketed to an older demographic.

To account for this we made a few defaults to streamline this process. Such as when creating a tag, it automatically places itself to reduce the steps required to set up the tag. Furthermore, the tag icon is by default set to medication as we anticipated this is what the app will mainly be used for. Finally, we specifically placed the “Find” button as the first button in the tag manipulation list so that they will not have to scan down the list to find their tags. We could have taken this further by introducing default “sets” that have the tag information pre-filled for certain generic items such as medication types, keys, wallets etc.

System Flow

Generally, our system follows a natural linear flow from logging in to finding tags. It follows the logical route of logging in to view your personal tags, then selecting a tag to perform operations on, to eventually locating said tag. However, there are a few caveats that may impede the logical flow of operations.
For instance, after signing up the user must then log in again to use the app. The flow of that specific section could be improved so that when the user signs up it automatically logs them in.

Also, on the tags page there is a list of tags with just their names initially showing. This could be an extra cognitive hurdle for some as they would have to figure out that to perform operations with the tags they must first click and expand the tag. To combat this, we made it so that on log in the tags page will automatically open the first tag on the screen.

After this, it will not open a tag automatically, as we found that this impedes the usage of the tag page where it opens the first tag when another tag was just selected. This method will clearly show the usage of the tags page to any new time user and won't impede more experience users at the same time.

AR

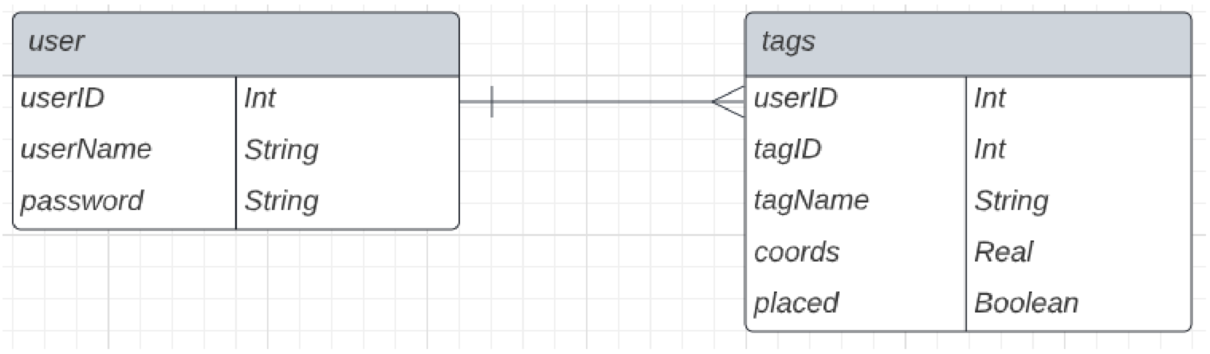
Our implementation of augmented reality employs the ar.js library. This is an AR library designed specifically for use in web development. It also utilises geolocation which is ideal for our application. However, this geolocation is not entirely accurate as it has about a 10m allowance from a specific location.

Due to this, we are not able to guide the user to the exact location of their item but more in the general area. This however is still useful because guiding the user to this general location should stimulate their memory and help them remember the specifics of where the item is placed. Accuracy seems to be a general issue with location based augmented reality as we have concluded from our research. There was another option which did help provide slightly better accuracy however it required the user to open the app in the same location every time which does not fit with how we intend for FindAR to be used.

Database

The database is required to store our user information. We started with the goal to use IBM's own services, a MySQL method, but that was infeasible due to the method to connect to it required a multi-gigabyte suite, and the server that serve such a database were not renewed by IBM, so we had no method to host it.

But we did still formulate a MySQL schema:



As shown above we plan to have 2 tables in our database, one to store users of the application and another to store information about their tags. Using the userID as a link between the two tables.

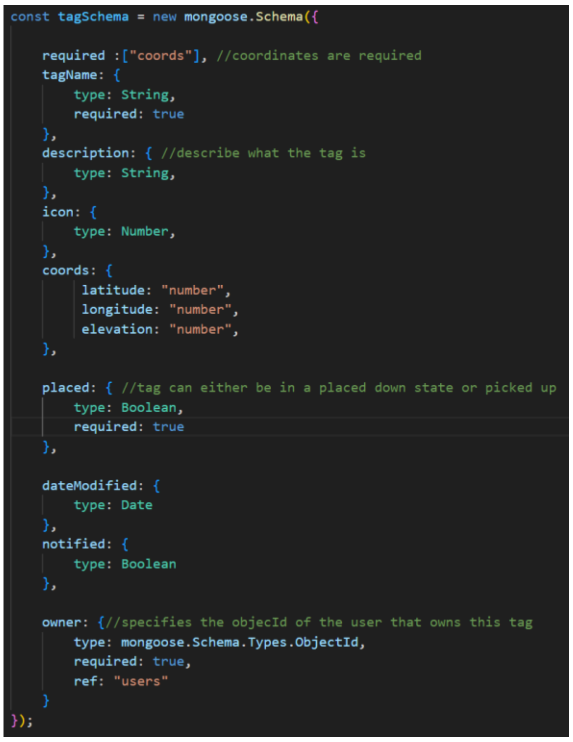
To test this database, we created an instance in IBMs db2. We started by using SQL statements to generate these tables.

```
CREATE TABLE user ( userID INT GENERATED ALWAYS AS IDENTITY (START WITH 1
INCREMENT BY 1) PRIMARY KEY, userName varchar(255) NOT NULL, password
varchar(255) NOT NULL )
```

```
CREATE TABLE tags ( userID INT, tagID INT GENERATED ALWAYS AS IDENTITY
(START WITH 1 INCREMENT BY 1), tagName VARCHAR(255) NOT NULL, coords REAL
NOT NULL, placed BOOLEAN NOT NULL, FOREIGN KEY (userID) REFERENCES
```

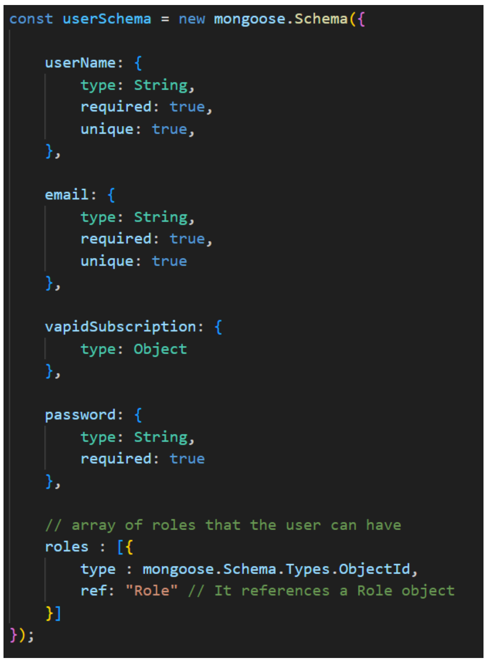
But, as we could no longer use IBMs systems, we had to find another service to host our database. From this, we came to MongoDB. This is a **non-relational** database and doesn't use tables.

However, transplanting to it was a very easy thing, as it had a similar idea of a 'schema', but just represented every instance in a table as an 'object'. From this, and after development and additions, we defined a tag object as:



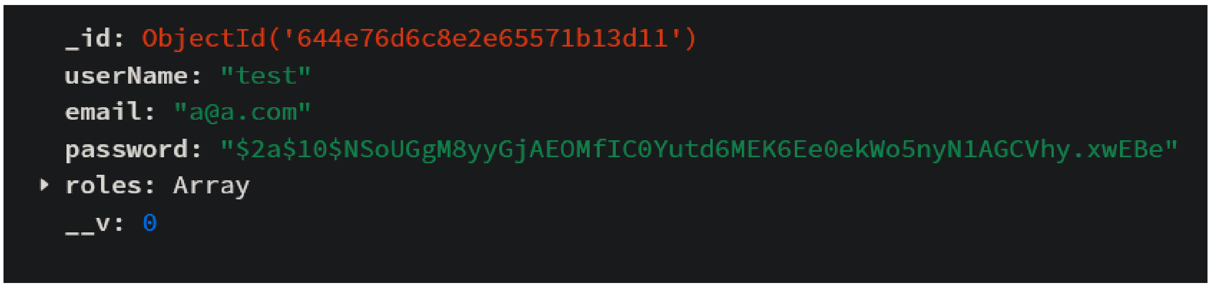
```
const tagSchema = new mongoose.Schema({

    required :["coords"], //coordinates are required
    tagName: {
        type: String,
        required: true
    },
    description: { //describe what the tag is
        type: String,
    },
    icon: {
        type: Number,
    },
    coords: {
         latitude: "number",
         longitude: "number",
         elevation: "number",
    },

    placed: { //tag can either be in a placed down state or picked up
        type: Boolean,
        required: true
    },

    dateModified: {
        type: Date
    },
    notified: {
        type: Boolean
    },

    owner: {//specifies the objecId of the user that owns this tag
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: "users"
    }
});
```

And a user object as:



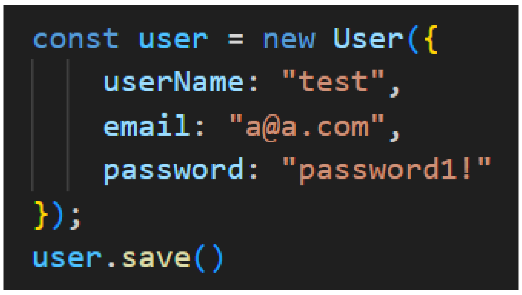
```
const userSchema = new mongoose.Schema({

    userName: {
        type: String,
        required: true,
        unique: true,
    },

    email: {
        type: String,
        required: true,
        unique: true
    },

    vapidSubscription: {
        type: Object
    },

    password: {
        type: String,
        required: true
    },

    // array of roles that the user can have
    roles : [{
        type : mongoose.Schema.Types.ObjectId,
        ref: "Role" // It references a Role object
    }]
});
```

MongoDB hosts a database for us for free, and we cannot host this database locally as our team needed a central store to test. Modifying such a database uses simple commands in OOP style. For example:



```
  _id: ObjectId('644e76d6c8e2e65571b13d11')
  userName: "test"
  email: "a@a.com"
  password: "$2a$10$NSoUGgM8yyGjAEOMfIC0Yutd6MEK6Ee0ekWo5nyN1AGCVhy.xwEBe"
▸ roles: Array
  __v: 0
```

Is created via:



```
const user = new User({
        userName: "test",
        email: "a@a.com",
        password: "password1!"
});
user.save()
```

It is important to understand that MongoDB is primarily a JavaScript system. It is heavily optimised in express.js, and includes middleware for encryption, etc, that we can use to handle systems we would otherwise not be confident to handle.

This is superior to MySQL for that reason, as it is something easily deployable, and handles a lot of the systems in MySQL for us. Its integration in JavaScript makes code readable, contrary to strings of SQL commands, via the MongoDB Node.js driver.

Deployment

To deploy our application, we chose the local hosting option ngrok. This method requires a computer to be running as a pseudo-server then other devices can connect via link and use the app as intended. Ideally, we would have a remote server host the app so that we don't need someone to keep the connection open, but we have found this method meets our needs in this specific situation.

Project Management & Progress

We have managed our team in an Agile approach, with weekly sprints, where we split it into three sub-teams working on a specific set of user stories assigned to them for the sprint. The team administrator would assign the sprints in the meetings we had where our team went over what the sub-teams completed in the last sprint, reflection on the sprint, and then we would assign new user stories for the next sprint.

We used a Trello board to manage our work and assign individual people user stories and other tasks related to the project for the current sprint. We also documented each of our sprints and went over what was achieved, what went wrong, how the team coped with the problems and adapted to overcome them as well as plans for future sprints. The team was punctual in trying to attend these meetings as often as possible and if someone missed a meeting, we would tell them the summary of what was done in the meeting and the tasks assigned to that team member as soon as possible.

Our user stories were organised in a Trello board that we would use to plan our work for the next sprints. The project was split into 3 themes which in turn were split into Epics and each user story would be associated with one of these epics. The 3 themes were:

- "As a user, I want to manage my tags, so I know what items I have stored"
- “As a user I want a visual AR indication of where my tag is, so I know where my items are"
- "As a user, I want to be able to manage my account, to log in and use the service".

These themes would be broken down into manageable epics which would then consist of a few user stories and tasks to be completed to achieve the epic's requirements. After each sprint, we would mark the user stories that the team has completed as "finished" and assign new user stories based on the epics we still haven't completed.

When we decided on how we should split the sub-teams we considered the previous experience of team members to make our choice. The 3 sub-teams were to: work on the backend, frontend, and Augmented Reality part of the project.

We have decided that team members who had some web development experience, such as Alfie, should work on the backend of our project as it was deemed the most challenging part of the project and had to start being developed earlier.

We didn't have anyone with experience in Augmented Reality so we assigned people who had an interest to research this area such as Alan. The frontend team mainly consisted of people interested in the design of the project and people who would be taking optional modules in Human-Computer Interaction such as Vishal. The sub-teams were not set in stone and a lot of transfers happened throughout to accommodate current challenges. For example, once the backend was mostly developed some people, such as Rodion, were assigned to work on the frontend as that was the only part of the project that had a lot of unfinished work.

Apart from frequent meetings, we would maintain good communication with our team members through messaging apps such as Discord to discuss challenges during the development of our project. Frequent communication allowed us to overcome these challenges and by the end of the sprint, if the challenges were not solved by the sub-teams, the rest of the team would offer help to solve them. Apart from frequent meetings with the team, we would also aim to meet our supervisor frequently to discuss our progress and get feedback on what we can improve in our project.

Our team would aim to document our progress throughout the development of the project. Each sprint would be documented by our team administrator and documents related to our project, such as sequence diagrams of the backend, would also be documented. As well as this, we would document attendance to the meetings and a summary of what the meetings discussed to see what team members may need additional help if they e.g., don't turn up regularly.

Everyone in the team was punctual and all members attended most of the meetings we had, however sometimes some team members couldn't attend some of our meetings and we would accommodate this by later communicating with them about what the tasks assigned to them are, as well as try to get them to join our meetings online through e.g., Microsoft Teams if they can do so.

During our development, we used a git repository to manage our work in sub-teams and have separate branches for the development of frontend, backend, and AR. We would then merge these branches into the main branch after the sprint was finished to conform with continuous integration principles.

In summary, our team was managed well throughout the entire development of this project. We approached this project in an Agile manner with frequent meetings and with the usage of a Trello board to manage our work. We have documented every sprint as well as the meetings and provided further documents related to the project such as activity diagrams and designs of our front page. We would have frequent communication during the sprint to discuss our work and overcome challenges, and then do a reflection at the end of a sprint, as well as set new targets for the next sprint.

To provide a better understanding of our sprints, here are a couple of examples of how our sprints were documented. To see (git repo copy) all 14 sprints documented, go here.

Sprint 1 (09/11/2022-16/11/2022)

In this sprint, we decided to first create the foundation of our project. Thus, it was made to be shorter than normal.

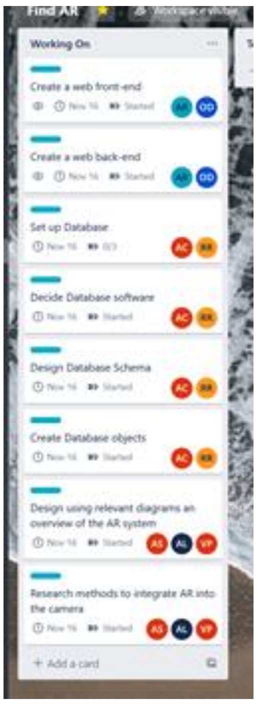
We split the team into 3 sub-teams:

- AR Team -> Designing/researching how to use AR in websites.
- Website Team -> Setting up a website that everyone can develop on in the future
- Database Team -> Designing some extensible database that reflects our current goal of a viewable list of tags and a user login system.

We have also set up the sub-teams to reflect the skills of everyone by allowing people to be in the teams which they have the most experience with and most comfortable with. However, for the AR Team, no one in the team has much practical experience, this is why we made it the largest team so that we can focus more on researching and understanding the AR aspect.

Our plan is to have members in each team switching over throughout the course of the project to allow the whole team to collaborate with each other and develop a firm understanding of every aspect within the project.

The tasks assigned were:



Find AR

Working On

- Create a web front-end — Nov 16 — Started — AR, OD
- Create a web back-end — Nov 16 — Started — AR, OD
- Set up Database — Nov 16 — 0/3 — AC, RR
- Decide Database software — Nov 16 — Started — AC, RR
- Design Database Schema — Nov 16 — Started — AC, RR
- Create Database objects — Nov 16 — Started — AC, RR
- Design using relevant diagrams an overview of the AR system — Nov 16 — Started — AS, AL, VP
- Research methods to integrate AR into the camera — Nov 16 — Started — AS, AL, VP

+ Add a card

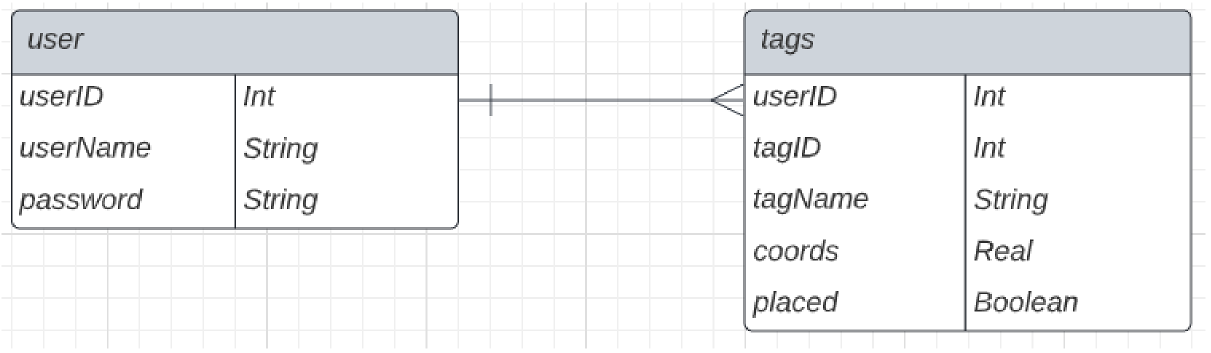
Sprint 1 Retrospective

Sub-Team Website Creation:

- Unable to progress much due to being unable to access IBM services.
- To solve this we, set up a meeting with our sponsor and got everyone to sign up to IBM services as the prerequisite to access the services.
- Thus, these tasks have been moved to the next sprint to do.

Sub-Team Database Creation:

- Decided on a database creation software.
- Selected IBM's db2 as this software is stored on IBMs cloud which will allow us easier integration with our IBM cloud server when it is up and running.
- This software operates using SQL documents which we can load into the program to perform actions on the database. We designed a preliminary database schema which includes record that we have deemed key to this project



user	
userID	*Int*
userName	*String*
password	*String*

tags	
userID	*Int*
tagID	*Int*
tagName	*String*
coords	*Real*
placed	*Boolean*

Using the above schema, we drafted SQL code to create the relevant tables and the connection between them. This code also ensures that the ID values are auto-incrementing, so we don't have to manually keep track of IDs.

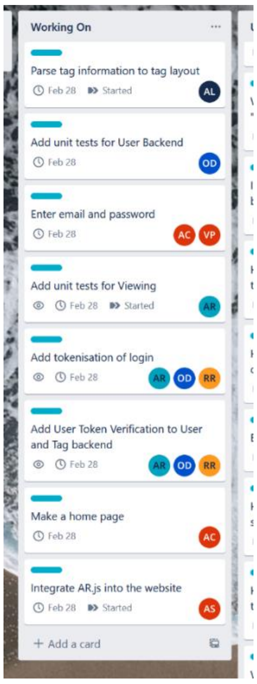
Sub-Team AR:

- Our primary goal during this sprint was to research methods to develop a prototype of a location-based AR token system.
- The main tool we found that will help is the AR.js studio code generator which will generate code for us to use.
- Something we realised after this sprint was that we severely overestimated how much time the AR aspect of this program will take us, we don't think it will be necessary for us in the future to have in depth knowledge of AR methods we just need to use the AR.js pre-written code and embed it into our website.
- The specific AR Problem we are solving also seems to be well documented.

The goal for the next sprint is to get actual prototype up and running so that we can demonstrate location-based AR tokens.

Sprint 7 (21/02/2023-28/02/2023)

For this sprint, the assigned stories are:



Working On

Parse tag information to tag layout
Feb 28 Started AL

Add unit tests for User Backend
Feb 28 OD

Enter email and password
Feb 28 AC VP

Add unit tests for Viewing
Feb 28 Started AR

Add tokenisation of login
Feb 28 AR OD RR

Add User Token Verification to User and Tag backend
Feb 28 AR OD RR

Make a home page
Feb 28 AC

Integrate AR.js into the website
Feb 28 Started AS

+ Add a card

Sprint 7 Retrospective

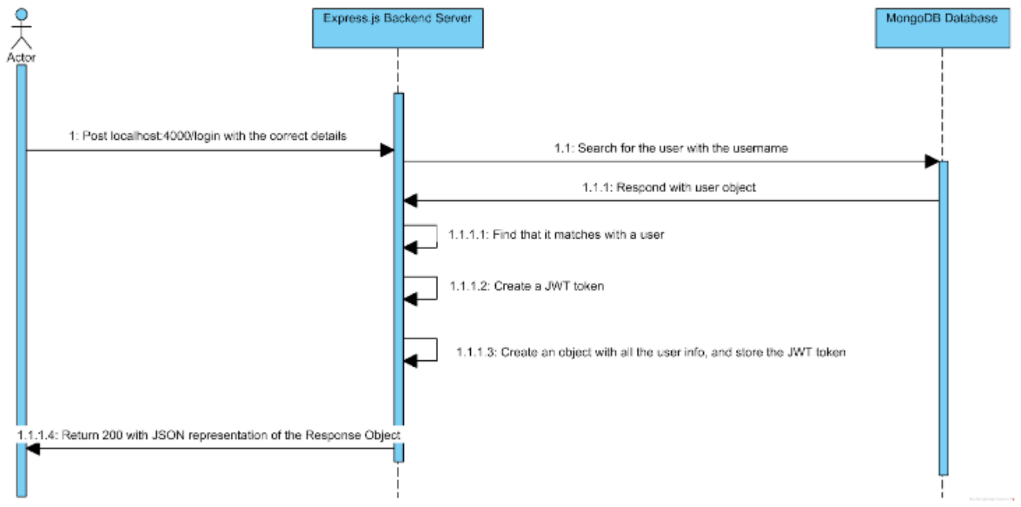
Front End Team

- Built a fetch API post request including the inputted name and password to query the database.
- Upon the query the database returns either true or false depending on whether that user existed, and the password matched.
- Implemented a rudimentary home page with buttons linking to the tags, login, signup and AR page. Currently only the links to tags and login work.

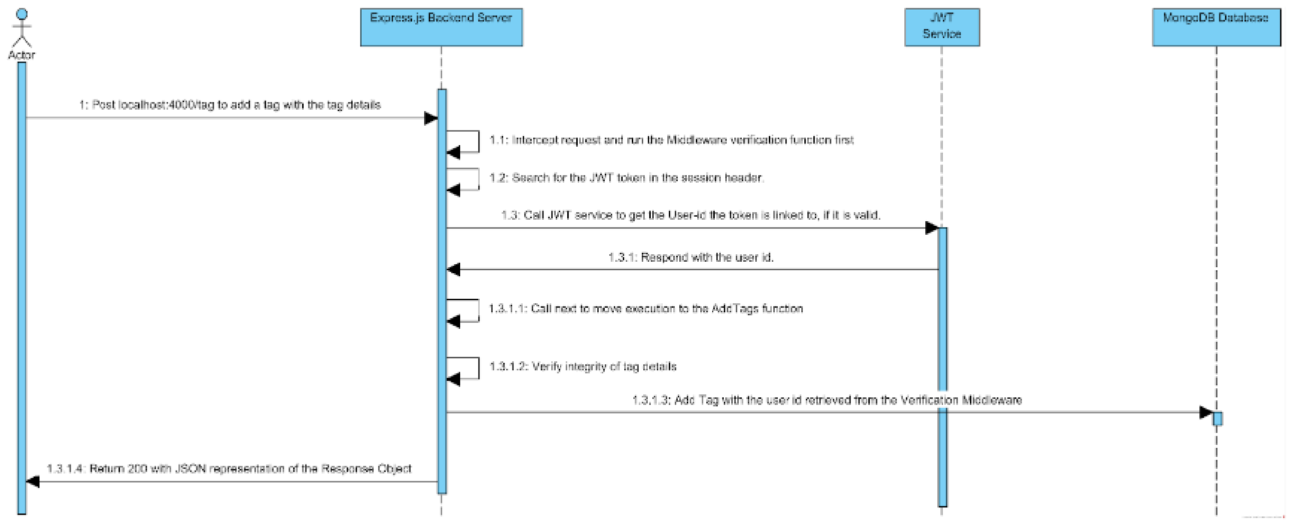
- The AR link will be included when the AR and Front-End branches are merged to main, and the AR svelte page becomes accessible to the front-end team.

Backend Team

- Created a method for the server to create a 'token' that verifies someone is logged in as someone.
- The general framework:

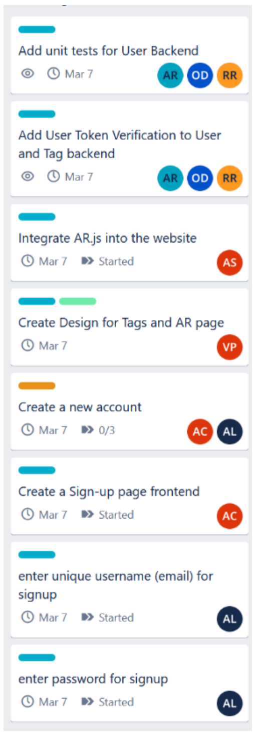

Actor
Express.js Backend Server
MongoDB Database
1: Post localhost:4000/login with the correct details
1.1: Search for the user with the username
1.1.1: Respond with user object
1.1.1.1: Find that it matches with a user
1.1.1.2: Create a JWT token
1.1.1.3: Create an object with all the user info, and store the JWT token
1.1.1.4: Return 200 with JSON representation of the Response Object

- Created middleware to verify the token is in the request and is valid.
- This is shown here:


Actor
Express.js Backend Server
JWT Service
MongoDB Database
1: Post localhost:4000/tag to add a tag with the tag details
1.1: Intercept request and run the Middleware verification function first
1.2: Search for the JWT token in the session header.
1.3: Call JWT service to get the User-id the token is linked to, if it is valid.
1.3.1: Respond with the user id.
1.3.1.1: Call next to move execution to the AddTags function
1.3.1.2: Verify integrity of tag details
1.3.1.3: Add Tag with the user id retrieved from the Verification Middleware
1.3.1.4: Return 200 with JSON representation of the Response Object

Sprint 8 (28/02/2023-07/02/2023)

For this sprint, the assigned stories are:



Add unit tests for User Backend
Mar 7 — AR OD RR

Add User Token Verification to User and Tag backend
Mar 7 — AR OD RR

Integrate AR.js into the website
Mar 7 — Started — AS

Create Design for Tags and AR page
Mar 7 — VP

Create a new account
Mar 7 — 0/3 — AC AL

Create a Sign-up page frontend
Mar 7 — Started — AC

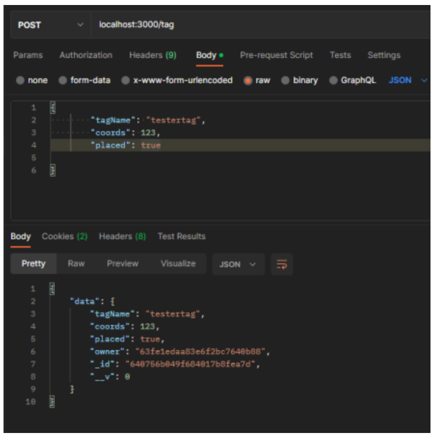
enter unique username (email) for signup
Mar 7 — Started — AL

enter password for signup
Mar 7 — Started — AL

Sprint 8 Retrospective

Backend Team

- Completed the backend token verification user stories.
- This means that every route in the backend now requires the user to be logged in, as they must send a JWT token via a cookie they get after they login.
- This token can be verified, and if real will return the UserID the token was created from.



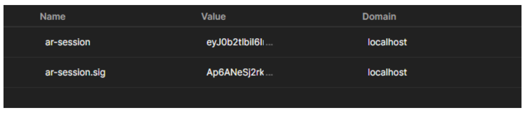
POST localhost:3000/tag

none · form-data · x-www-form-urlencoded · ● raw · binary · GraphQL · JSON

```
{
    "tagName": "testertag",
    "coords": 123,
    "placed": true

}
```

```
{
    "data": {
        "tagName": "testertag",
        "coords": 123,
        "placed": true,
        "owner": "63fe1edaa83e6f2bc7640b88",
        "_id": "640756b049f684017b8fea7d",
        "__v": 0
    }
}
```

Although no owner ID is specified, the id is retrieved from the 2 cookies you get after logging in:



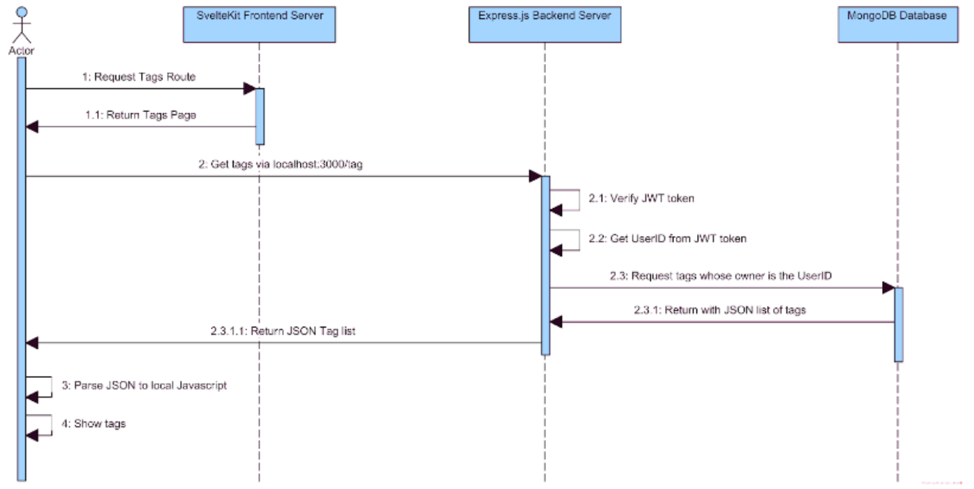
Name	Value	Domain
ar-session	eyJ0b2tlbiI6Ii…	localhost
ar-session.sig	Ap6ANeSj2rk…	localhost

This essentially guarantees users can only create tags if they are logged in, and such a tag will always be linked to them.

This idea applies to all routes, such as deleting tags, or updating tags. You can only update tags you own, and this is verified by using the session token to get the UserID that the user is logged in as. The token acts as a password to verify who they are.

We found that updating the getTags route to only show the tags the user owns was propagated to the tags page with little effort to get working, so we had completed that user story as well.

An example can thus be shown. This diagram assumes the user has already logged in and has a JWT token cookie:

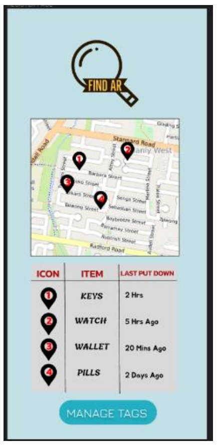

Actor
SvelteKit Frontend Server
Express.js Backend Server
MongoDB Database
1: Request Tags Route
1.1: Return Tags Page
2: Get tags via localhost:3000/tag
2.1: Verify JWT token
2.2: Get UserID from JWT token
2.3: Request tags whose owner is the UserID
2.3.1: Return with JSON list of tags
2.3.1.1: Return JSON Tag list
3: Parse JSON to local Javascript
4: Show tags

AR Team

- Got the AR.js page to work with reliability. This means that when going to the AR page, it will display a red cube in a specified location.
- Next steps are to customize the cube into something more visually appealing for the user.

Front-end Team

- Designs of the sign-up page was completed
- Next steps are to integrate the backend call to add a user from the form’s inputs.
- Below is an initial design for the tags page that was prototyped.

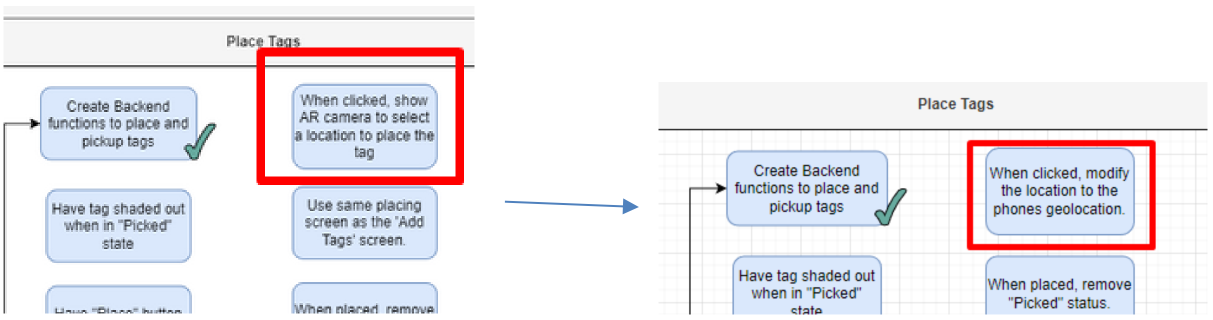


ICON	ITEM	LAST PUT DOWN
1	KEYS	2 Hrs
2	WATCH	5 Hrs Ago
3	WALLET	20 Mins Ago
4	PILLS	2 Days Ago

After further discussion we decided some elements like the map would not be possible to complete for this release within the scope of the project. We felt the map would probably be unneeded, as the accuracy is low to a house level. Others, such as the tags list, were chosen to implement into the final design (except the last put down column).

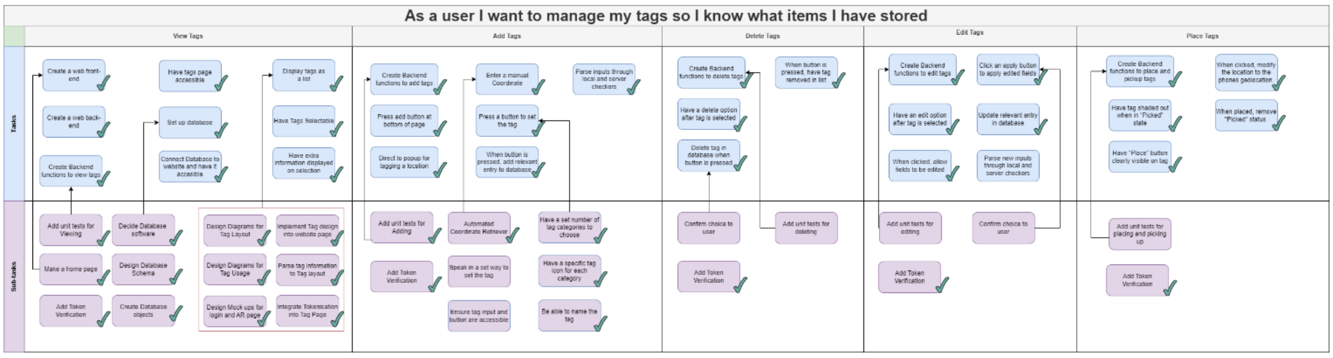
We also found a discrepancy in the user stories, one where placing a tag entailed entering AR to do so.

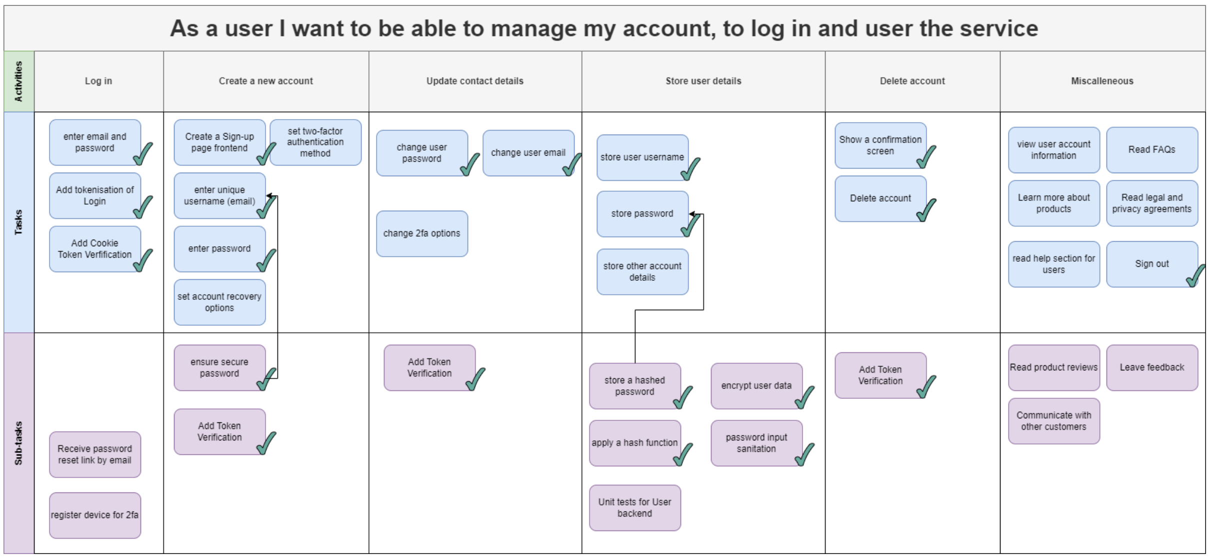
We discussed that there wouldn’t be much point doing that, as we could just get the phone's GPS and place it there. AR would provide no benefit because there would be nothing to find when placing a picked-up tag. So, we changed it:

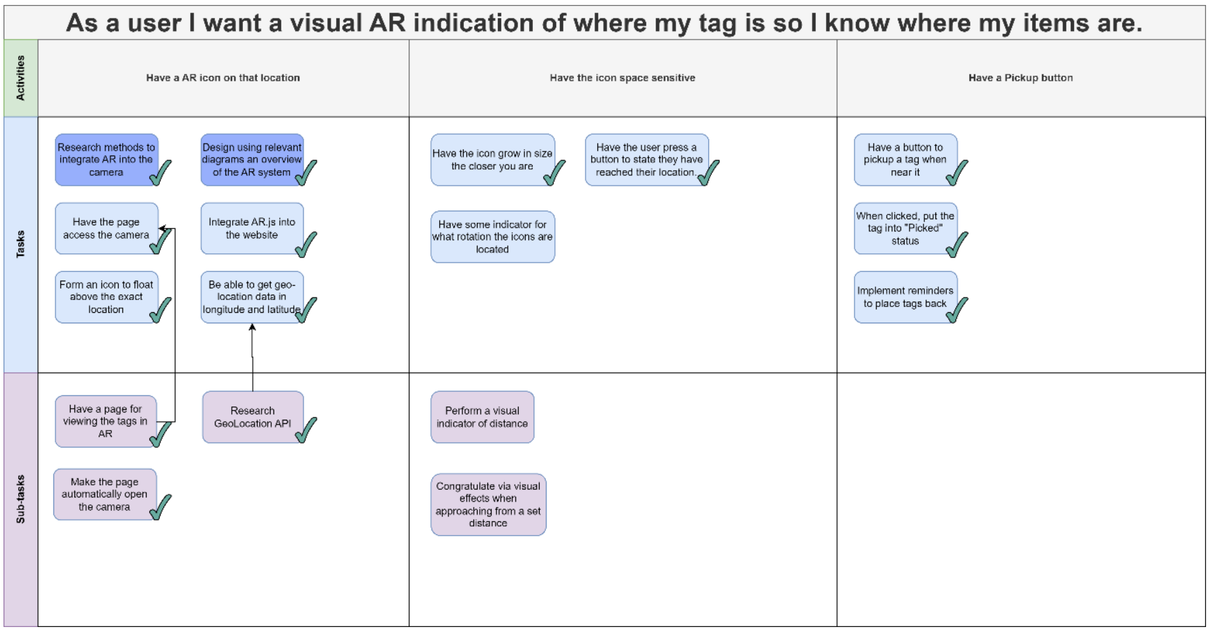

Place Tags
Create Backend functions to place and pickup tags
When clicked, show AR camera to select a location to place the tag
Have tag shaded out when in "Picked" state
Use same placing screen as the 'Add Tags' screen.
Place Tags
Create Backend functions to place and pickup tags
When clicked, modify the location to the phones geolocation.
Have tag shaded out when in "Picked" state
When placed, remove "Picked" status.

Story Maps

During the sprints the team administrator managed 3 ‘story maps’ that map out the general workflow for completing the 3 themes of the project. After the project was complete, these story maps can be shown as:


As a user I want to manage my tags so I know what items I have stored
View Tags
Add Tags
Delete Tags
Edit Tags
Place Tags
Tasks
Sub-tasks
Create a web front-end
Have tags page accessible
Display tags as a list
Create a web back-end
Set up database
Have Tags Selectable
Create Backend functions to view tags
Connect Database to website and have it accesible
Have extra information displayed on selection
Add unit tests for Viewing
Decide Database software
Design Diagrams for Tag Layout
Implement Tag design into website page
Make a home page
Design Database Schema
Design Diagrams for Tag Usage
Parse tag information to Tag layout
Add Token Verification
Create Database objects
Design Mock ups for login and AR page
Integrate Tokenisation into Tag Page
Create Backend functions to add tags
Enter a manual Coordinate
Parse inputs through local and server checkers
Press add button at bottom of page
Press a button to set the tag
Direct to popup for tagging a location
When button is pressed, add relevant entry to database
Add unit tests for Adding
Automated Coordinate Retriever
Have a set number of tag categories to choose
Add Token Verification
Speak in a set way to set the tag
Have a specific tag icon for each category
Ensure tag input and button are accessible
Be able to name the tag
Create Backend functions to delete tags
When button is pressed, have tag removed in list
Have a delete option after tag is selected
Delete tag in database when button is pressed
Confirm choice to user
Add unit tests for deleting
Add Token Verification
Create Backend functions to edit tags
Click an apply button to apply edited fields
Have an edit option after tag is selected
Update relevant entry in database
When clicked, allow fields to be edited
Parse new inputs through local and server checkers
Add unit tests for editing
Confirm choice to user
Add Token Verification
Create Backend functions to place and pickup tags
When clicked, modify the location to the phones geolocation.
Have tag shaded out when in "Picked" state
When placed, remove "Picked" status.
Have "Place" button clearly visible on tag
Add unit tests for placing and picking up
Add Token Verification


As a user I want to be able to manage my account, to log in and user the service
Activities
Tasks
Sub-tasks
Log in
Create a new account
Update contact details
Store user details
Delete account
Miscalleneous
enter email and password
Add tokenisation of Login
Add Cookie Token Verification
Create a Sign-up page frontend
set two-factor authentication method
enter unique username (email)
enter password
set account recovery options
change user password
change user email
change 2fa options
store user username
store password
store other account details
Show a confirmation screen
Delete account
view user account information
Read FAQs
Learn more about products
Read legal and privacy agreements
read help section for users
Sign out
Receive password reset link by email
register device for 2fa
ensure secure password
Add Token Verification
Add Token Verification
store a hashed password
encrypt user data
apply a hash function
password input sanitation
Unit tests for User backend
Add Token Verification
Read product reviews
Leave feedback
Communicate with other customers


As a user I want a visual AR indication of where my tag is so I know where my items are.
Activities
Tasks
Sub-tasks
Have a AR icon on that location
Have the icon space sensitive
Have a Pickup button
Research methods to integrate AR into the camera
Design using relevant diagrams an overview of the AR system
Have the page access the camera
Integrate AR.js into the website
Form an icon to float above the exact location
Be able to get geo-location data in longitude and latitude
Have a page for viewing the tags in AR
Research GeoLocation API
Make the page automatically open the camera
Have the icon grow in size the closer you are
Have the user press a button to state they have reached their location.
Have some indicator for what rotation the icons are located
Perform a visual indicator of distance
Congratulate via visual effects when approaching from a set distance
Have a button to pickup a tag when near it
When clicked, put the tag into "Picked" status
Implement reminders to place tags back

These represent the progress we have made by the end of the project. Not all the user stories here were required, thus some were not completed.

Reflection

Luckily, most initial choices in frameworks happened to be a good choice, svelte provided us with sufficient features to implement whatever was needed whilst also being easy to learn, an important property as web development was not our forte. We choose express.js for our backend, a node.js framework with useful middleware, such as cookie handling, etc. See the software manual for more details. We decided AR.js was to be used for our initial implementation of AR but we kept with it until release.

We faced many difficulties throughout the development of the website, the four main ones being:

- Dealing with IBMCloud
- Our general inexperience with web development
- Implementing The AR system
- Development without a server

Our initial development ideology was to get a working system up as quick as possible, this was delayed significantly due to problems with IBMCloud, Alfie found it very hard to get this system to work and it didn't work properly until mid-December, and by this time we created an entire workflow around IBMs server and CI. However, we were on a trial, on the word that it was something we could renew from IBM. This trial, however, was found to not be renewable, and thus we lost our web server we worked on to create.

This involved switching from IB2 to no-SQLI mongoDB and having to use ngrok to host the website temporarily during the testing and the demo. This method of development was partly retrieved from IBMs serverless system we got, but we also had to derive a working development method from scratch again. It cannot be understated that this caused weeks of delays waiting on IBM to fix our web server expiration, before moving away from their services entirely.

Most of our group was inexperienced when it came to handling web development. This caused many problems with the velocity of our development; many members had to learn the basics of web development first before being able to contribute. Our lack of experience in web development also created problems predicting the estimated time to develop and the feasibility of a feature that only got amplified with having to deal with IBM's complex CI tools and server, which we eventually had to scrap due to them expiring.

Our biggest problem turned out to be the AR system, initial research suggested it would be the easier part of the development because we could just copy the template code provided by AR.js studio and modify where needed but we encountered 4 main problems:

- The template code provided by AR.js studio was deprecated and no longer working, after lots of time trying to make this template code work, we discovered more recent template code which was used instead.

- AR.js did not work inside the context of a node.js/svelte website, this turned out to be a problem with library loading, the fact this was not explicitly made clear in AR.js docs made it difficult to debug.
- AR.js worked inconsistently, for some devices it worked perfectly fine, for others it didn't work at all. This reoccurred many times but updating a dependency of AR.js, A-frame, seemed to fix the issue, this was also difficult to debug as we had no indication of what was causing this issue.
- AR.js tag was very inaccurate; it would be about 20 meters away from the intended location and the distance would only update every 5 meters.

We concluded that our problem with inaccuracy is one we couldn't solve; this is mainly because GPS positioning is inaccurate indoors as the GPS signals get bounced off the roofs of buildings.

We briefly considered an implementation not using GPS. Instead of having the coordinate system be global, we would instead use the coordinate system localised, anchored to a specific point e.g., a door, every time you open the website, you'd walk to the anchor point so it could recalibrate. This would theoretically much better precision, but it came with a myriad of problems:

- AR.js does not support tracking of motion without GPS, so we must use a different library
- The other main library is AR-Core which has a lot of compatibility issues.
- Additionally having a reference space tied to a position means that the user will have to position their phone in the exact same orientation every single time when using AR.

Due to these reasons, we found it appropriate to just stick with AR.js. We did not expect an older user to go through such motions when using the app, especially when it requires precision to orient.

It is important to note that these libraries are not perfect. Three.js removed GPS orientation entirely from their repo, which is used to orient and correct the camera of the device to the gps.

This issue is divisive. It came from IOS and Android using different systems for location and orientation privileges. This system is from non-standard APIs in AR and is a consequence of the infancy of the technology. AR.js includes this system, but it is not officially supported in three.js, and is thus prone to glitches and bugs, prevalent on IOS devices due to their non-committal API that has entirely different permissions for orientation and location.

AR.js provides a solution to make this work, however their npm package which allows us to use three.js code does not provide such a solution, and doesn't work on IOS. Thus, we had to stick with the html ar.js method. We found no difference in AR accuracy between them, and as they use the same underlying GPS method this makes sense.

There are a lot of areas to improve given more time:

- The flexibility of our system
- AR component accuracy
- More assistance features

When AR-Core's compatibility issues are finally fixed it may be feasible to redo the AR component with GPS, by using a plane relative to a fixed position. More assistance would mean to have e.g., tutorials to guide users into using the website effectively and not forgetting their items.

What we set out to make initially was a website where you could login, manage your tags e.g., add, remove, edit, or view them in an AR environment so you can tell where your tags are in the house / the general direction of them. We believe that most of the requirements we set out have been achieved:

- A homepage where you can login / sign-up
- A page to manage your tags, deleting, editing, as mentioned before
- A page to modify your account e.g., delete your account, change password etc.
- Have an AR section where you can see where your tag is / pick it up.
- Put down a tag which will mark the coordinates of it for later use.
- Notifications to remind the user to put down their tag.
- An intuitive user-friendly interface

But as mentioned before, Arguably the most important aspect of the website, The AR content, is extremely lacking in accuracy.

Our approach to managing the group and divvying out jobs was to create a kanban board of user stories to be fulfilled this sprint and allowing our group members to pick which ones to take, this would lead to group members more proficient in specific areas to do a user stories they would be better suited for without the overhead of manually assigning tasks, generally this may promote group members doing a limited number of user stories, we made sure that everyone was taking an equal amount of work.

The approach to development was similarly liberal, we focused more on implementation then planning, we think this worked out great. Our environment for development is much more prone to unforeseen circumstances than most (again due to being inexperienced with website development), having a rigorous design is helpful but when one unpredicted constraint is introduced the entire design will have to be re-evaluated. By rapidly implementing and testing new features we can discover unforeseen circumstances much quicker and therefore handle them quicker.

The only decision we would change would be to make meetings a lot more frequent, often members would be bottleneck by a problem or system that another member was working on, a simple problem can be alleviated through the messaging but face to face was much more effective at dealing with non-trivial problems, which we faced a myriad of. We believe that having daily stand-up meetings would have significantly improved the velocity of development.

Another area we would improve is contingency planning, due to the fast pace of development we had to take on, we didn't have time to plan for contingencies, which caused a lot of problems in the AR system, as we discovered our system was lacking in accuracy, we found ourselves having to research alternatives instead of having a plan to fall back on.

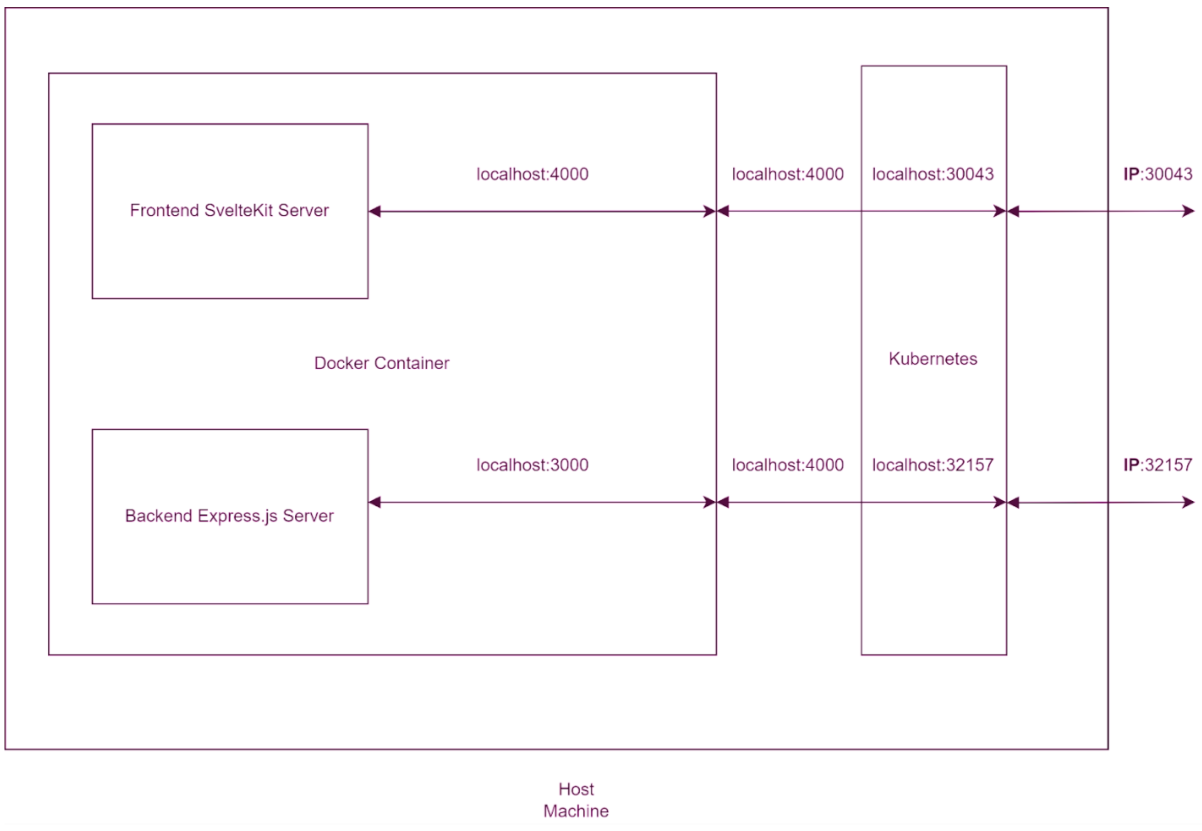
One thing we discovered was that having members switch to different areas of the system was particularly expensive, there is significant cost involved with learning another area of the system well enough to start contributing to it, we noticed that the first sprint following a member's assignment to a different area often resulted in a spike of bugs and decrease in progress.

Part 2: Software Manual

Project Overview

The aim of this project was to manage a set of tags, storing information including their location, to (via AR) locate said tags. The project has the aim to be simple, easy to access, and relies on GPS to track locations.

Technology Basis



Kubernetes

Before using IBM's services became untenable, the usage of Kubernetes was an important aspect of this project and will still work if set up again with a similar service. Following the above diagram's generalization, Kubernetes has the task of translating outside requests from the internet into routed requests that get sent to the docker container.

Other features of Kubernetes have not been actively utilized in the project, and the routing has been configured using a **Helm chart** configuration.

Docker Container

It is important for us to be able to develop this project without relying on a service such as IBM and remove any variability of differing local environments. Therefore, we applied the **Serverless** method of development.

This included the creation of a docker container, mandated by a **Dockerfile**, using the **node:18-alpine** image to create a 'bare-bones' Linux environment that is standardized. Two servers were made to run in this docker container. The docker container acts like any Linux host machine but is lightweight and easily deployed for development on any machine.

Frontend SvelteKit Server

This is the first server made to be set up in the docker container. It uses the **Svelte** framework as its basis, with **Sveltekit** being the method of **routing** the various web pages. Sveltekit is not required to run a multi-page website with many routes.

This server is accessed either via the localhost port 4000, or port 30043 if the Kubernetes server has been setup. This server contains the entire frontend and is what every user is expected to use to interface with the backend.

Backend Express.js Server

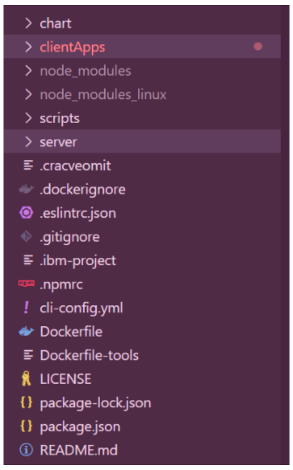
This is the API of the website, and is what the frontend calls to perform various actions, described in the API Routes section. It uses Express.js as its basis, which is a framework for Node.js, including extra **middleware,** such as **cors** management, **JWT tokens**, **cookie sessions,** etc. A full list is here.

The backend server follows a clear MVC system, where the models are the MongoDB database items, the controllers are the functions executed when calling a specific route, and the view is the routes which defines what routes you must call to execute certain functions.

Middleware is included as functions that are executed before the main function when calling a specific route and are important in verification and validation tasks.

The backend and frontend work on separate ports, and thus require CORS to function. Express.js handles this cors issue. Cookie session handles storing tokens received when logging in, and cron is used to handle a cron schedule on picked up tag reminders.

For explicit usages, see the **server.js** file in the root of the backend server.



```
> chart
> clientApps
> node_modules
> node_modules_linux
> scripts
> server
.cracveomit
.dockerignore
.eslintrc.json
.gitignore
.ibm-project
.npmrc
cli-config.yml
Dockerfile
Dockerfile-tools
LICENSE
package-lock.json
package.json
README.md
```

Project Structure

All other folders and files are for docker & kubernetes purposes.

The frontend server is in the **clientApps/AR-app** directory.

The backend server is in the /**server** directory.

Database

The database is hosted using a free MongoDB Atlas instance. MongoDB is a NoSQL document database that permits a more dynamic approach to design. However, Mongoose, an Object Data Modelling library, has been used to enforce qualities such as uniqueness, types, and requirements. It also provides an opportunity to intervene when saving data to the database, in the form of middleware.

This is useful for providing validation before the data has reached the backend controllers. The backend controllers provide a set of functions to allow the frontend to interact with the database. These functions are then mapped to routes within their respective category.

For example, **server/routes/user-route.js** (line 9) specifies that a HTTP GET request on **user/** will call the verification middleware function and then call **userController.getUser()**.

Models have been created for each of the ‘entities’ in the database. Data relating to the users of the app and the tags they place is structured using three models. The user and tag models establish the fields, such as **userName**, and describe the type of data that is accepted, **String**(s), along with qualities, such as **required** and **unique**.

Three such models have been defined to create ‘rigid’ collections of data. One for a user, tag, and a role model. The role model is used to assign categories to users according to the privileges they have access to. It currently contains two items: representing standard and admin users respectively.

Mongodb is a malleable database, and thus these models are technically guidelines only, but it has been set to force the structure to ensure no bad actors can set pointless data.

JWT Tokens

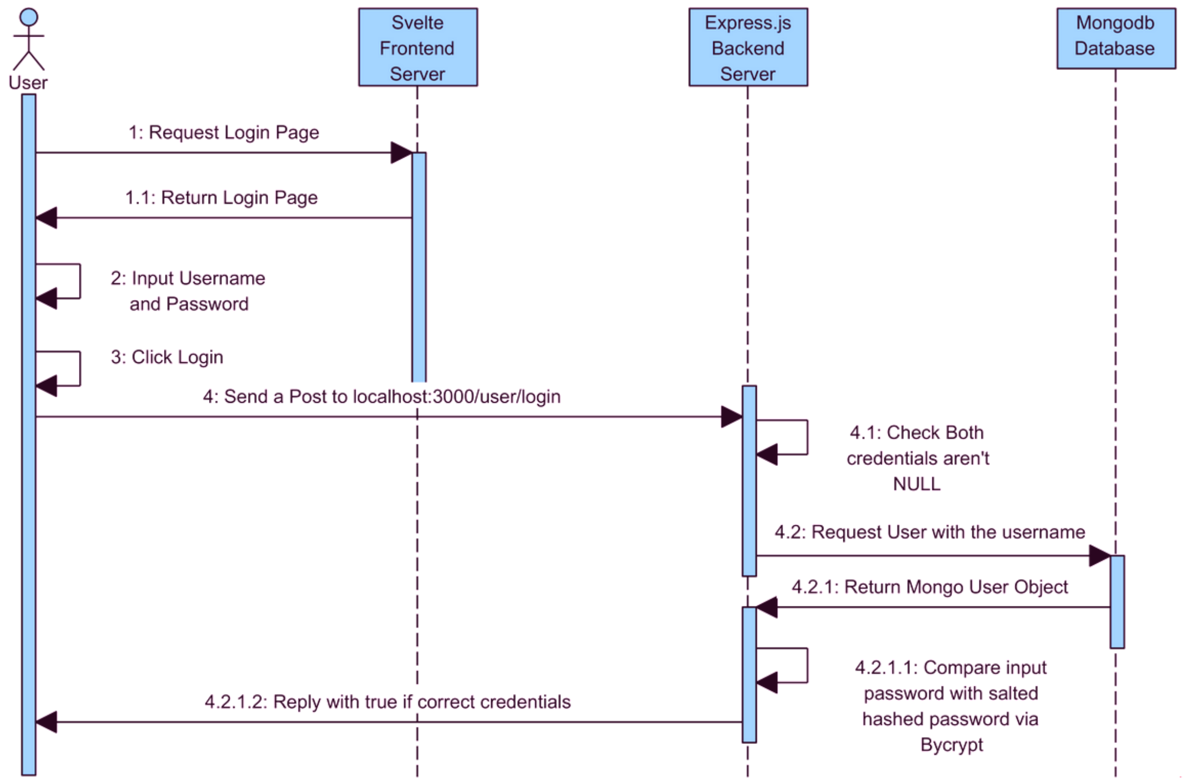
The backend uses JWT tokens to authenticate the user and restrict certain operations if they are not logged in such as creating a tag. It also makes sure that the user doesn’t have to re-login each time they refresh a page. The token is stored inside a cookie, so when the user logins in that cookie essentially becomes invalid.

Functionality Examples

To help in understanding, there will be some explained areas of the backend that are of importance. These will illustrate a general methodology of handling requests, and thus should help with attaining a standard in understanding other features of the codebase.

Logging In

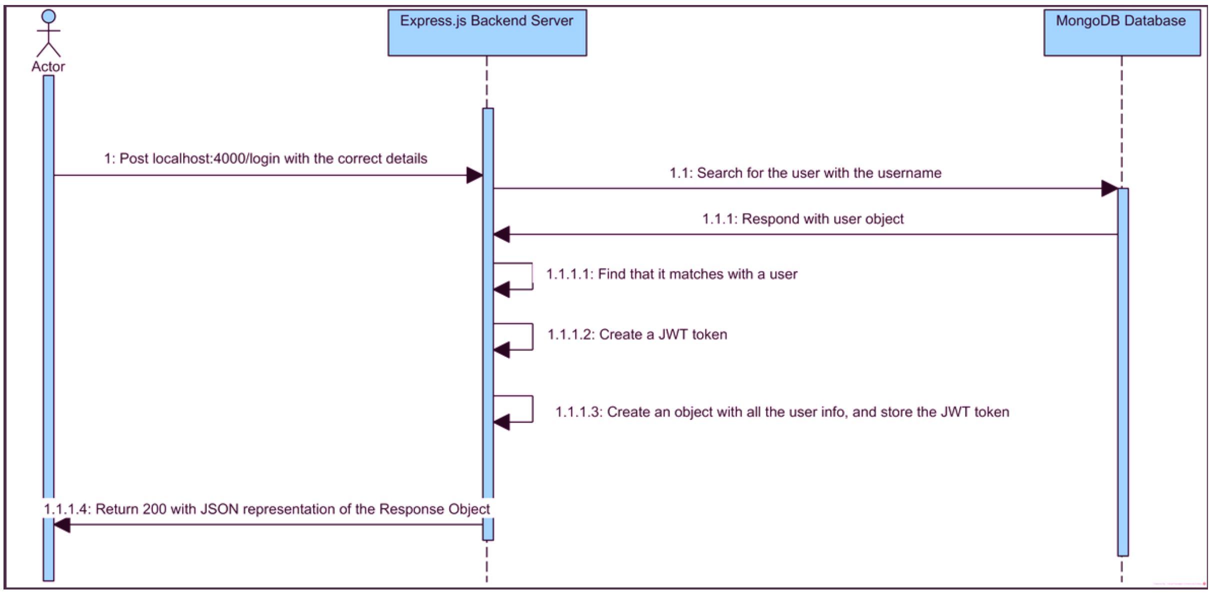
Logging in is an important part of using the website, and it is important that credentials are stored right. Therefore, I will illustrate first a basic example:


User
Svelte Frontend Server
Express.js Backend Server
Mongodb Database
1: Request Login Page
1.1: Return Login Page
2: Input Username and Password
3: Click Login
4: Send a Post to localhost:3000/user/login
4.1: Check Both credentials aren't NULL
4.2: Request User with the username
4.2.1: Return Mongo User Object
4.2.1.1: Compare input password with salted hashed password via Bycrypt
4.2.1.2: Reply with true if correct credentials

Generally, all credentials are hashed with bycrypt. This library ensures they are salted and follows a secure standard. Therefore, to check an input credential is correct, all we need to do is compare the hashes.

It is important to understand that a 'login' is just a method of storing a temporary token that can be used as a 'hello, I am X' message to the server whenever you make a privileged request. Thus, a login does not just return true.

Showing this:


Actor
Express.js Backend Server
MongoDB Database
1: Post localhost:4000/login with the correct details
1.1: Search for the user with the username
1.1.1: Respond with user object
1.1.1.1: Find that it matches with a user
1.1.1.2: Create a JWT token
1.1.1.3: Create an object with all the user info, and store the JWT token
1.1.1.4: Return 200 with JSON representation of the Response Object

JWT Token

A JWT token is a token that stores a ‘certain bit of information’, and in this case it will store the user id of the user if they have entered the correct credentials. When returned, it is set to a specific header in the server such that the Express.js middleware, **CookieSession,** will intercept it and set it to **set-cookie (info)** headers in the response.

The backend uses JWT tokens to authenticate the user and restrict certain operations if they are not logged in, such as creating a tag. It also makes sure that the user doesn’t have to re-login each time they refresh a page. The token is stored inside a cookie, so when the user logs-in, it is included in every server request when specified.

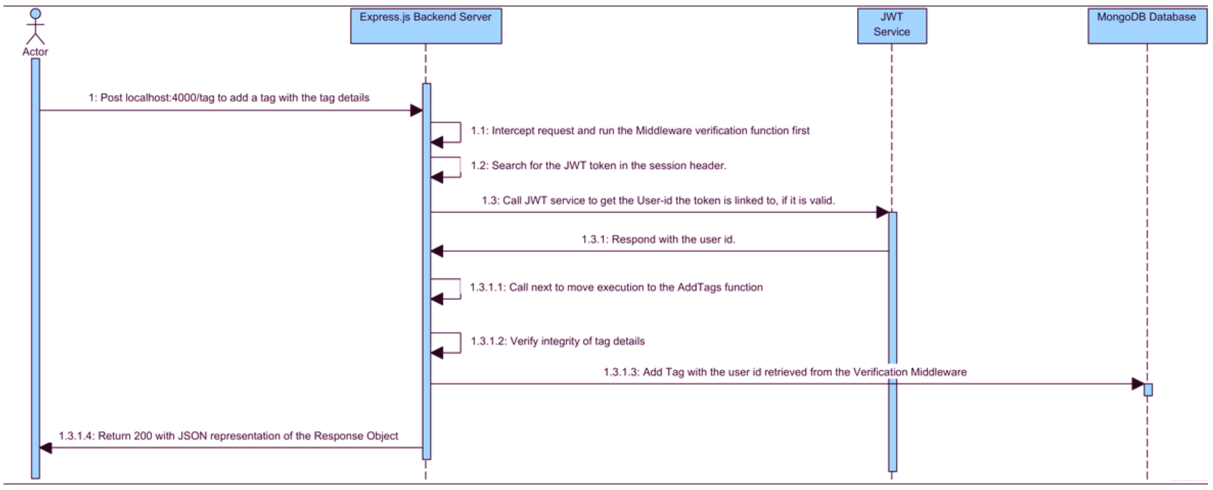
This entails that the local machine’s browser will store a ‘session token’ cookie that will, when specified in a request header, be sent up with the request for the server to check ‘if you’re logged in’.

Naturally, this token is a temporary password that shouldn’t ever be shared.

To be clear, this exchange **must** occur on the same domain, and thus it is important that when proxying, to use the /api route in the :4000 port, the frontend, as proxying typically only exposes one port. All frontend code uses the /api route for this reason. See API Routes for more details.

Adding a Tag

Adding a tag is a common action done by the user. Showing this:


Actor
Express.js Backend Server
JWT Service
MongoDB Database
1: Post localhost:4000/tag to add a tag with the tag details
1.1: Intercept request and run the Middleware verification function first
1.2: Search for the JWT token in the session header.
1.3: Call JWT service to get the User-id the token is linked to, if it is valid.
1.3.1: Respond with the user id.
1.3.1.1: Call next to move execution to the AddTags function
1.3.1.2: Verify integrity of tag details
1.3.1.3: Add Tag with the user id retrieved from the Verification Middleware
1.3.1.4: Return 200 with JSON representation of the Response Object

This illustrates an important area of functionality.

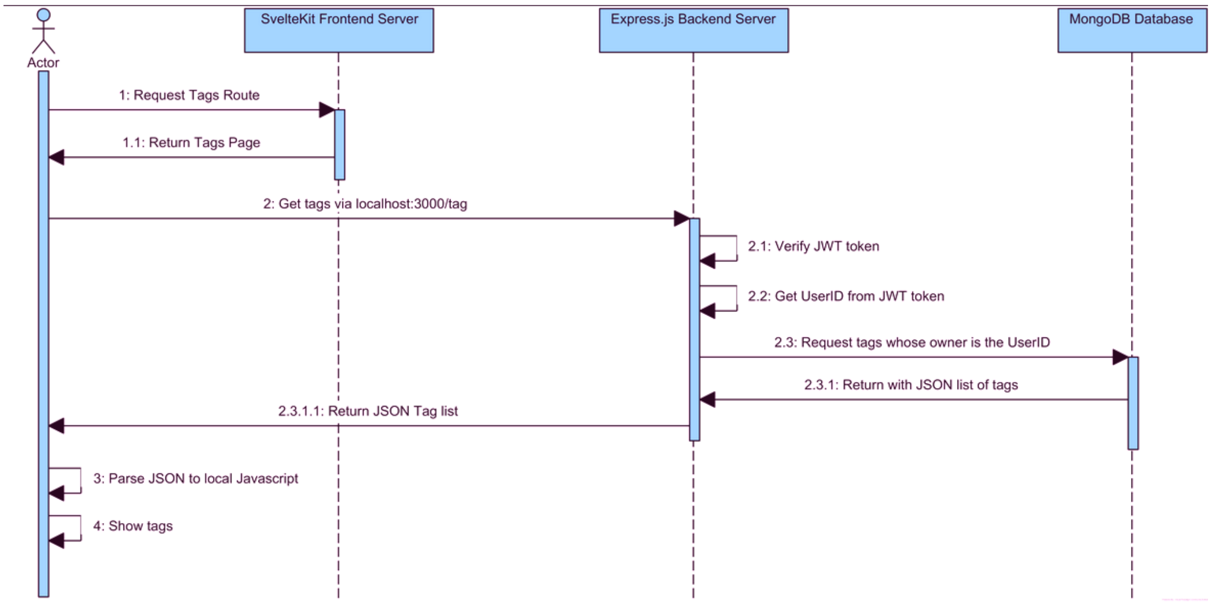
Token Verification Middleware

A lot of routes in the backend require a user to be logged in. Adding a tag, for example, wouldn't make sense when you are not logged in, as there would be no user to add the tag to (a required field in a tag object).

Therefore, the verifyToken middleware function was made to be a check before any route that is 'protected'. Its job is to check for a token in the request, and if it can find it, decode it and set the 'userId' field in the request object to the user id the jwt token stores. If it fails in any place, it will return an error response.

Getting Tags

As the last example, this will show a typical route implementing all these ideas.


Actor
SvelteKit Frontend Server
Express.js Backend Server
MongoDB Database
1: Request Tags Route
1.1: Return Tags Page
2: Get tags via localhost:3000/tag
2.1: Verify JWT token
2.2: Get UserID from JWT token
2.3: Request tags whose owner is the UserID
2.3.1: Return with JSON list of tags
2.3.1.1: Return JSON Tag list
3: Parse JSON to local Javascript
4: Show tags

Getting all the tags of a user, requires a JWT verification, and then will get all the tags from the MongoDB database.

API Routes

This section will detail the available routes for the API backend.

The API backend, on a local server, functions either on the 3000 port or the frontend 4000 port in the /api route. This corresponds to:

- localhost:3000
- localhost:4000/api

The 4000 port will proxy to the 3000 port and is used in all frontend code to allow easy proxying with ngrok and similar services, otherwise a specific configuration to allow two ports would be required when connecting the website to the internet.

When detailing the routes, this prefix will be ignored, and we will only detail the relative routes.

User Routes:

These routes handle management of the account.

Route POST /user

Call Method: POST
Request Body:
This handles adding a user, and when called the BODY of the request is expected to follow:

```
{
        userName: STRING,
    email:    STRING,
    password: STRING
}
```

The data goes through 3 middleware functions, checking the validity of the password and making sure the email and username don't already exist in the database.
Middleware: **checkDuplicateEmail**,**checkDuplicateUsername**,**validatePassword**
Response:
It will respond with the created entry in the database in the 'data' entry. Ie:

```
{
        data:  { "userName", "email", "password", "roles", "_id" ,"__v"}
}
```

The data entry will contain the entries specified in the request, with the password entry encrypted. It will also contain the _id entry that uniquely identifies the entry in the database. Roles contains a list of the IDs of the roles of the user. This is only useful for the server to identify what they can and cannot do, and if you want this translated use the GET request.
Special Comments:
Do note that in non-productions forms, the body of the request also accepts a 'roles' entry in the body that can specify if you want the user to be admin. To do this, add:

```
roles : ["ADMIN"]
```

To the body.

Route GET /user

Call Method: GET
Request Body:
Middleware: verifyToken
Response:
It will respond with an array of the form:
```
{ “userName", "email", "password", "roles" : [..], "_id" ,"__v”}
```
Where roles is an array of arrays of form:
```
    "roles": [
        {
            "_id": STRING
            "name":  STRING
            "__v": NUMBER
        }
    ],
```
The name will specify the role, and is either ‘admin’ or ‘user’. Admin specifies this user has admin permissions.

Route GET /user/all

Call Method: GET
This gets all the users in the database. Requires you to be logged in as an admin user for it to work.
Middleware: **verifyToken, isAdmin**
Response:
It will respond with an array of users in the same form as usual.
```
[
        { “userName", "email", "password", "roles" : [..], "_id" ,"__v”},
        …
]
```
The roles entry will be an array containing a ‘name’ entry that states what role the user is in. It will be an array, and everything else is a single readable value, acting as a string.
For example:
```
    "roles": [
        {
            "_id": "63fbca31f6b3ff65d28fea07",
            "name": "user",
            "__v": 0
        }
    ],
```
Is what a user correlates to.

Route PUT /user/email

Call Method: PUT
Request Body:
Updates user email in the database,cannot edit username.
```
{
    email:    STRING
}
```
Middleware: **verifyToken**
Response: The saved user with the edited email.
```
{ “userName", "email", "password", "roles" : [..], "_id" ,"__v”}
```
Special Comments:
Note that the roles array is not populated. Do not use it.

Route PUT /user/pass

Call Method: PUT
Request Body:
Updates user password in the database of the user with supplied ID.

```
{
      password:    STRING
}
```

Middleware: **verifyToken, validatePassword**
Response: The saved user with the edited password..

```
{ “userName", "email", "password", "roles" : [..], "_id" ,"__v”}
```

Route DELETE /user/:userId

Call Method: DELETE
Request Body:
Deletes the user by specifying their user ID. No body is required**.** The userID is specified in the parameter url.
Middleware: **verifyToken, IsAdmin**
Response:

```
{
  "acknowledged": true,
  "deletedCount": 1
}
```

This is the expected response if a correct userID is specified. If not, it will state false and 0.
Special Comments:
This route can only be executed by a user with the ‘admin’ role.

Route DELETE /user/

Call Method: DELETE
Request Body:
No Request body is required. Deletes the **logged in** user.
Middleware: **verifyToken**
Response:

```
{
  "acknowledged": true,
  "deletedCount": 1
}
```

Special Comments:
This functions as a self delete route that any logged in user can call to delete their account, if they so wish. No user ID is specified as it is retrieved from the token they get when they login.

Route POST /user/login

Call Method: POST
Request Body:
Logs you in by taking in a password and username. Returns a cookie used for tokenization.

```
{
  "userName": STRING
  "Password": STRING
}
```

Middleware: None
Response:
Responds with the user details and a token cookie that will be set in the browser automatically, defined to expire in 24 hours. This token will be used in any privileged routes.
Special Comments:
This route is required to be used before doing any routes that have the 'verifyToken' middleware function.

Route POST /user/logout

Call Method: POST
Request Body:
This is a request for removing the cookies on the user and thus logging them out. It requires no request body.
Middleware: **verifyToken**
Response:

```
{ message: "You've been signed out!" }
```

Special Comments:
If the user was logged in, it will send a response back to the browser to delete the cookies responsible for holding the tokens. **It does not blacklist these tokens.**

Route POST /user/loggedIn

Call Method: POST
Request Body:
No request body is required. Is used to check the user is logged in. Not used in the frontend for anything important
Middleware:
Response:

```
{ loggedIn: BOOL }
```

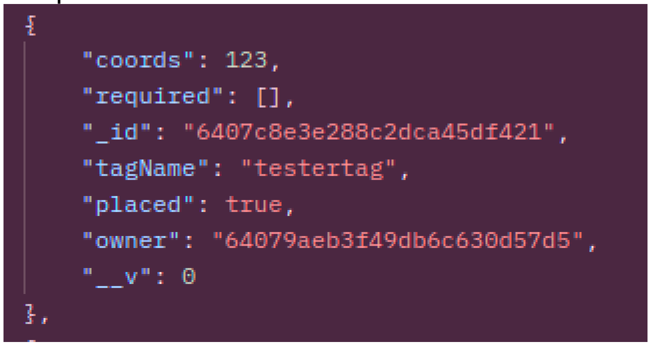
If the user is logged in, this entry is true, otherwise false.
Special Comments:
This route is mostly defunct, but has been kept in case it is required in the future.

Tag Routes:

These routes handle the management of the Tags that are associated with each user.

Route GET /tag/admin

Call Method: GET
Request Body:
No request body is required. Is used to get all the tags that are in the database for the admin user
Middleware: isAdmin,verifyToken
Response:
Responds with JSON data associated with each tag like this:

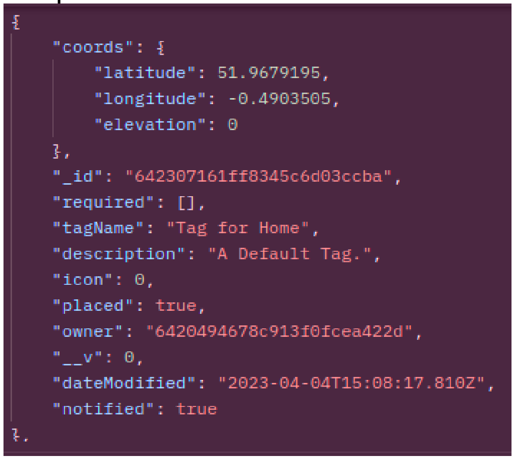


```
{
    "coords": 123,
    "required": [],
    "_id": "6407c8e3e288c2dca45df421",
    "tagName": "testertag",
    "placed": true,
    "owner": "64079aeb3f49db6c630d57d5",
    "__v": 0
},
```

Special Comments:
Route is mainly used for testing and isn't supposed to be used in release unless it is for admin work

Route GET /tag

Call Method: GET
Request Body:
No request body is required. Is used to get all the tags that are in the database associated with this specific user
Middleware: verifyToken
Response:
Responds with JSON data associated with the user's tags like this:

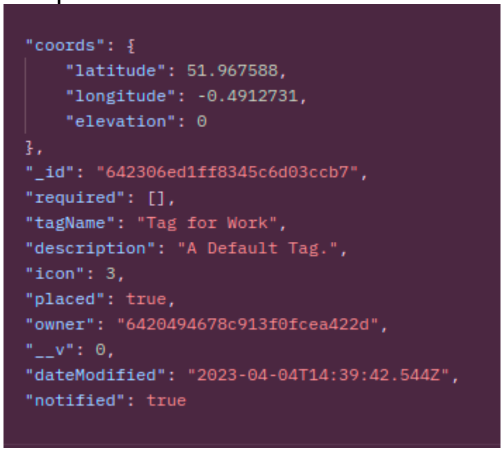


```
{
    "coords": {
        "latitude": 51.9679195,
        "longitude": -0.4903505,
        "elevation": 0
    },
    "_id": "642307161ff8345c6d03ccba",
    "required": [],
    "tagName": "Tag for Home",
    "description": "A Default Tag.",
    "icon": 0,
    "placed": true,
    "owner": "6420494678c913f0fcea422d",
    "__v": 0,
    "dateModified": "2023-04-04T15:08:17.810Z",
    "notified": true
},
```

Special Comments:

Route GET /tag/:tagId

Call Method: GET
Request Body: Gets the tag by specifying the tagID. No body is required**.** The tagID parameter is specified in the route itself
Middleware: verifyToken
Response:
Responds with JSON data associated with the tag like this:

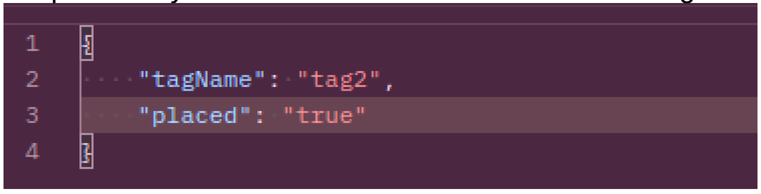


```
"coords": {
    "latitude": 51.967588,
    "longitude": -0.4912731,
    "elevation": 0
},
"_id": "642306ed1ff8345c6d03ccb7",
"required": [],
"tagName": "Tag for Work",
"description": "A Default Tag.",
"icon": 3,
"placed": true,
"owner": "6420494678c913f0fcea422d",
"__v": 0,
"dateModified": "2023-04-04T14:39:42.544Z",
"notified": true
```

Special Comments:

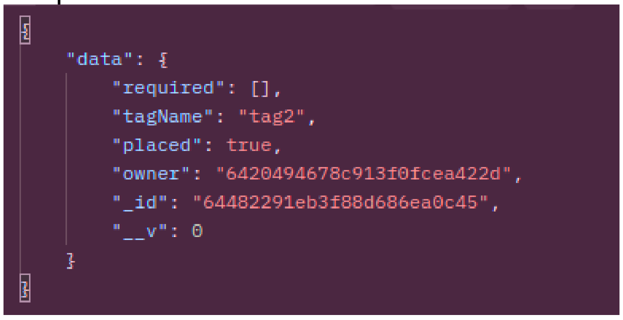
Route POST /tag

Call Method: POST
Request Body:
Request body contains the JSON data of the new tag to be added to the database like this:



```
{
    "tagName": "tag2",
    "placed": "true"
}
```

Middleware: verifyToken
Response:
Responds with JSON data associated with the new tag in the database:

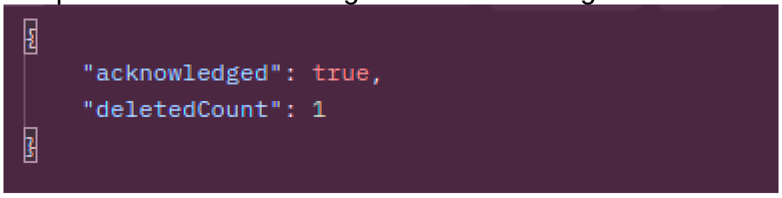


```
{
    "data": {
        "required": [],
        "tagName": "tag2",
        "placed": true,
        "owner": "6420494678c913f0fcea422d",
        "_id": "64482291eb3f88d686ea0c45",
        "__v": 0
    }
}
```

Special Comments:

Route DELETE /tag/:tagId

Call Method: DELETE
Request Body:
The parameter of the route contains the tagID of the tag to be deleted. No request body required
Middleware: verifyToken
Response:
Responds with acknowledgement that the tag has been deleted in the database



```
{
    "acknowledged": true,
    "deletedCount": 1
}
```

Special Comments:

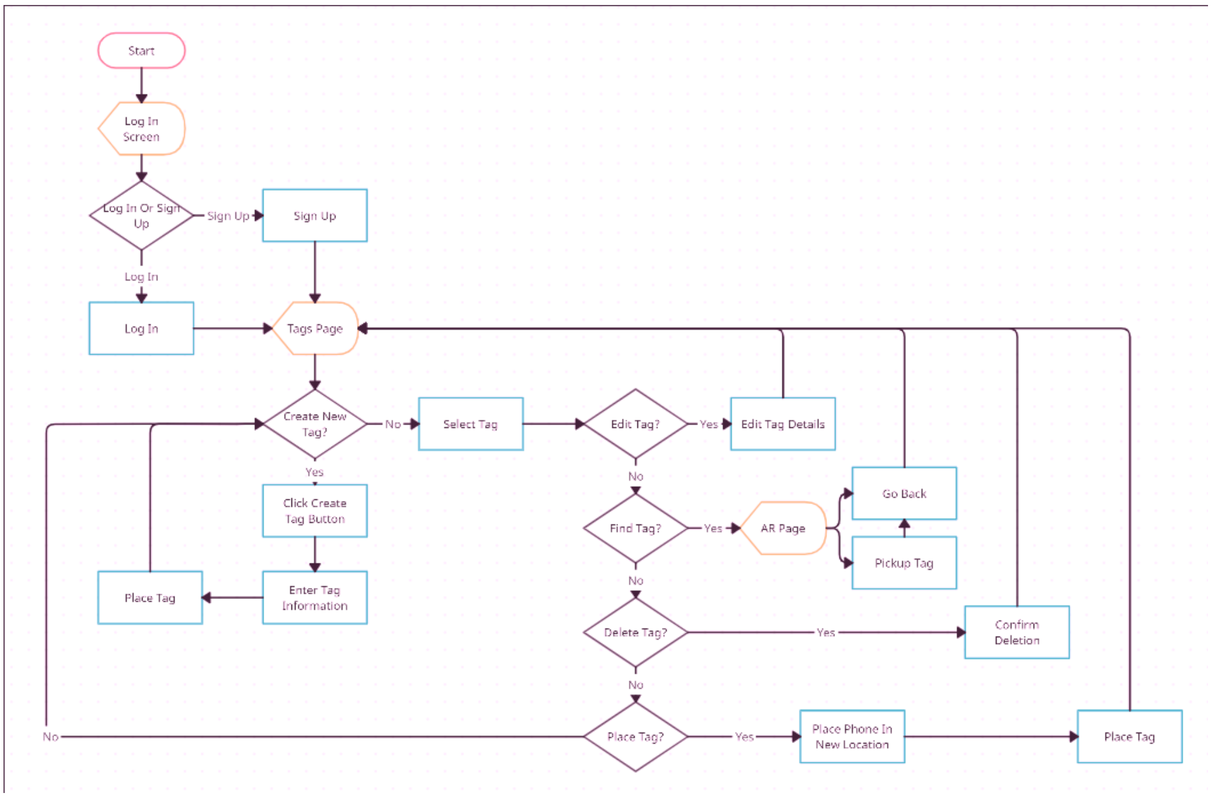
Route PUT /tag/:tagId

Call Method: PUT
Request Body:
Request body contains the JSON data of the tag, with the specified tagID in the route itself, to be appended to the database like this:
Middleware: verifyToken
Response:
Responds with JSON date of the tag that has been updated in the database
Special Comments:

Front End diagrams:

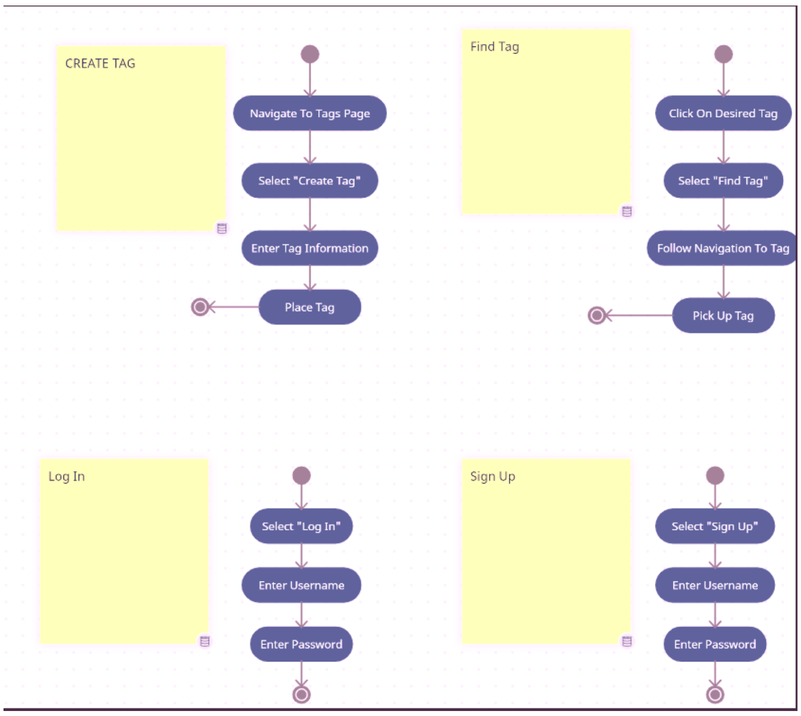
This flow diagram shows the general overview of our project and what the user can do in it. The user can sign up or log in and after that they are directed to the Tags page. In the Tags page the user can do these certain actions:

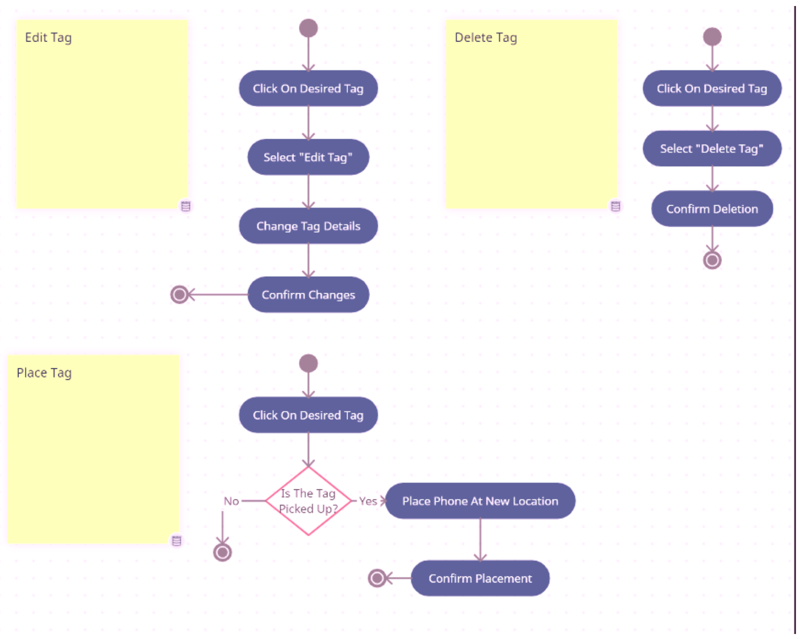
- Create a new tag with specified data that the user provides such as the tag's name and description
- Edit the tag's data
- Find a specific tag so that the user can then place it or pick it up
- Delete a tag
- Place a tag by moving their phone into a new location and placing the tag down(tag's Placed boolean field becomes true after this action)



These 2 diagrams show the steps that are needed to perform the operations described above. For example, if the user wants to create a new tag they need to:

- Navigate to the Tags page
- Select “Create Tag”
- Then enter the tag data
- Place the tag



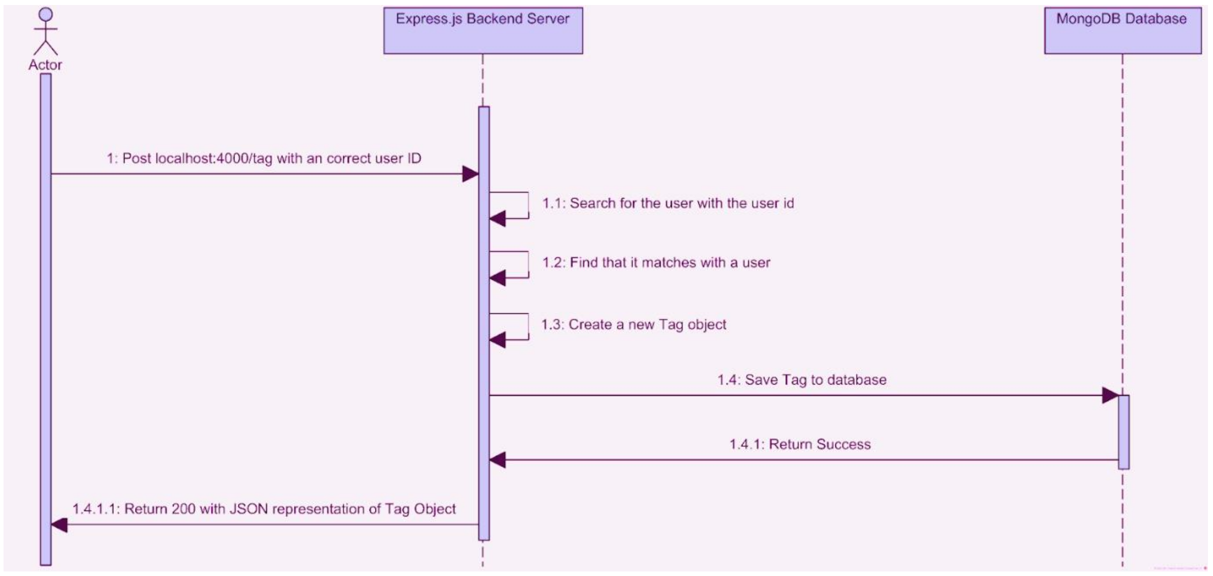


Unit Tests

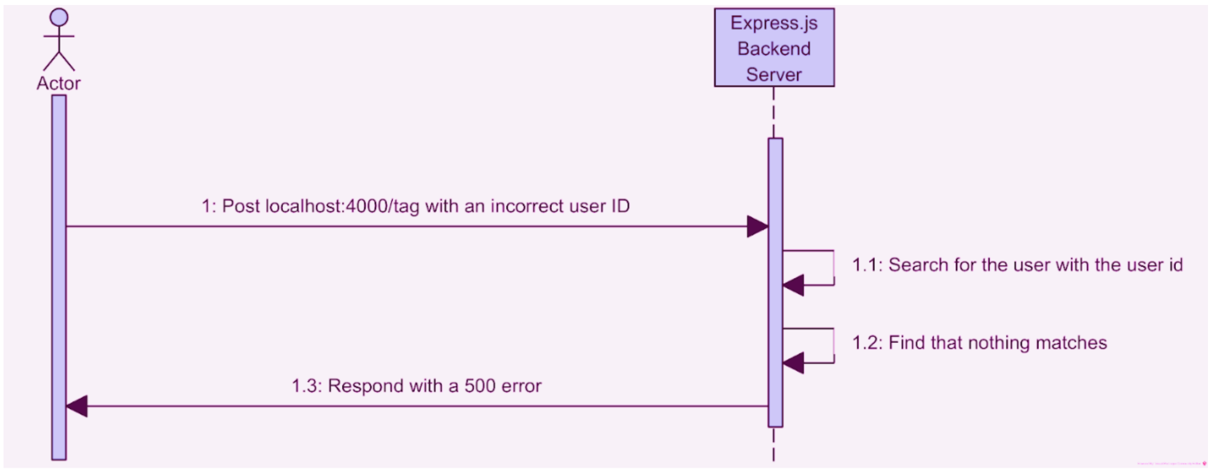
Unit tests are used to ensure that the API behaves correctly when presented with various scenarios. A Node.js testing framework, called Mocha, has been used to spin up potential situations that the API may encounter. The response is then assessed using the Chai HTTP assertion library to confirm that the desired behaviour is observed.

A pair of sequence diagrams are included below to illustrate the general principle of unit testing in this context.

Adding a tag correctly


Actor
Express.js Backend Server
MongoDB Database
1: Post localhost:4000/tag with an correct user ID
1.1: Search for the user with the user id
1.2: Find that it matches with a user
1.3: Create a new Tag object
1.4: Save Tag to database
1.4.1: Return Success
1.4.1.1: Return 200 with JSON representation of Tag Object

Adding a tag incorrectly


Actor
Express.js Backend Server
1: Post localhost:4000/tag with an incorrect user ID
1.1: Search for the user with the user id
1.2: Find that nothing matches
1.3: Respond with a 500 error

Test Plan

The test plan comprises a list of tests that can be run at any time to ascertain the behaviour of the API under a variety of circumstances. As the API is essentially a pair of controllers each with a set of functions for accessing and modifying the database, the tests have been grouped by their respective controllers and are presented in **server/test/test-tag-controller.js** and **server/test/test-user-controller.js**.

To run the tests, the app must be built in debug mode and initiated with **ibmcloud dev test**.

Tags

	Test Description
1	Adds a user with valid credentials
2	Adds an admin user with valid credentials
3	Login valid user
4	Adds a tag
5	Adds another tag
6	Adds a third tag
7	Gets user's tags
8	Gets a specific tag
9	Rejects non-admin usage of getAllTags()
10	Updates a tag
11	Deletes a tag
12	User can signout by deleting their session cookie
13	Login admin user
14	Admin user can get all tags

15	User's tags are deleted when they are deleted
16	User's tags are deleted when they delete themselves

Users

	Test Description
1	Adds a user with valid credentials
2	Adds an admin user with valid credentials
3	Rejects addition of duplicate user
4	Login valid user
5	Grabs valid user's details
6	Rejects non-admin usage of getUsers()
7	Rejects non-admin usage of deleteAnyUser()
8	Updates valid user's email
9	Updates valid user's password
10	User can signout by deleting their session cookie
11	Login admin user
12	Admin user can get all users
13	Admin user can delete any other user
14	User can delete itself

Building and running the code

Git repository link: https://projects.cs.nott.ac.uk/comp2002/2022-2023/team29_project

In order to build this project the user needs to install the IBMCloud CLI, and Docker.

Dependencies:

- Docker
- IBMCloud CLI Version v2.15.0

Do note that the IBMCloud CLI is used to setup and run the server in an easy single command. Fundamentally it is just a docker container defined by a dockerfile, and thus can be set up if you have sufficient docker knowledge.

Do **NOT** use the latest IBMCloud cli version, as the specific commands used are **deprecated**.

Command to build the project:

- sudo ibmcloud dev build [--debug]

Command to run the project:

- sudo ibmcloud dev run

Command to test the project:

- sudo ibmcloud dev test

It is sufficient to first build, and then run the project. You do not need to build more than once, and once running you can use tools such as Dev Containers to enter the container and perform edits.

Part 3: User Manual

What Is FindAR?

FindAR is a web app that uses AR (Augmented Reality) to help you tag and locate items you usually might misplace! It's as simple as tagging where you place an item and then using your phone's camera to locate said item when you need it.

We created this app in order to help those who suffer from memory recall issues due to their health, age or general bad memory.

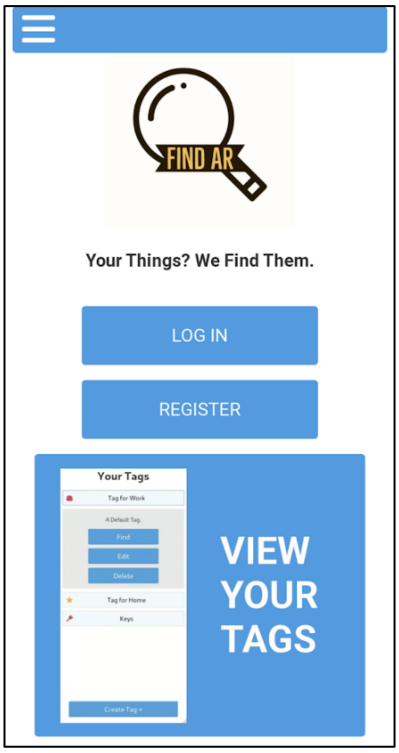
How Do I Access FindAR?

Accessing FindAR is simple and easy, it's a web page just like Facebook, BBC News or Amazon.
Simply click on the link provided and it will take you to our welcome page. You can then bookmark it for easier access later.

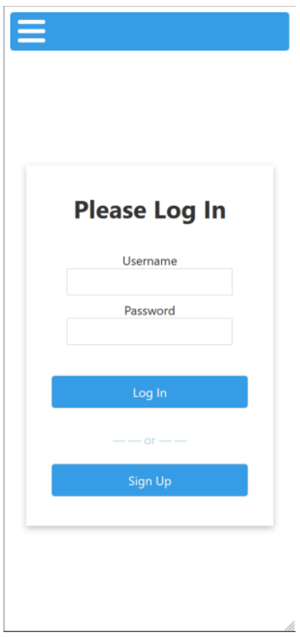
How Do I Use FindAR?

Logging In & Signing Up

On our welcome page you will be prompted to login or sign up for our service.

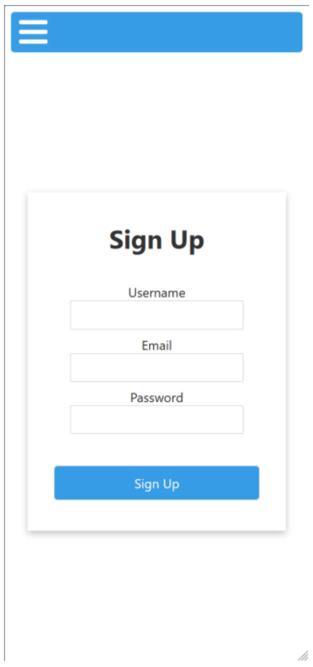


If you already have an account with us, simply select the login button. Then enter your username and password, then click the login button.



If you don't have an account with us, you can create one by clicking the signup button, this will direct you to a new page where you will enter your desired username and password.

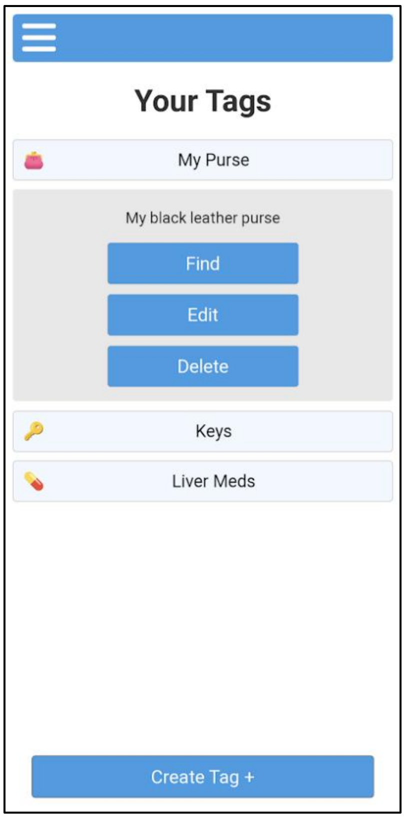
Once you have entered your details click the sign-up button and you will be registered.



Now just enter your details on the login page to log in.

Managing Your Tags

After logging in or signing up you will find yourself on the tags page.

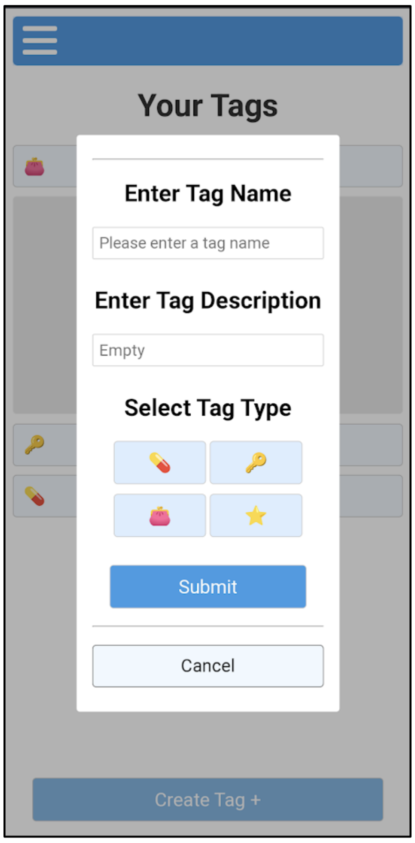


This page displays all your personal tags, using this page you can:

- Add new tags
- Find tags
- Place tags
- Edit tag information
- Delete tags

To add a new tag simply click the “Create Tag” button, from here you will be prompted to enter the tag’s name, a description of the tag and then select an icon to represent the tag.

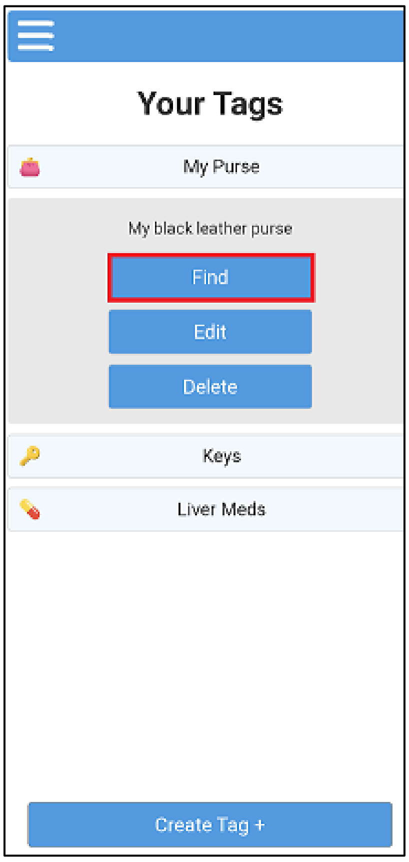
Press the “Submit” button to create the tag.



Note: creating a tag will automatically set your current location as the tag’s location.

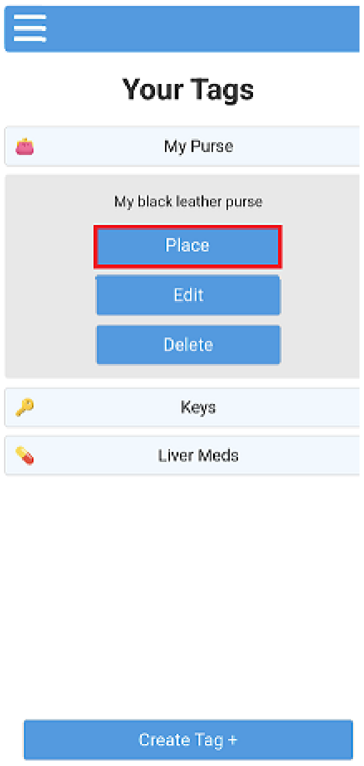
Each tag will have its own box labelled with the tag name.
You can press on this box to expand it. Once expanded you will be able to view your tag's description, place/find your tag, edit its details, or delete the tag.

To find your tag, select the "Find" button on the desired tag. This will take you to a separate page to help you locate your tag.

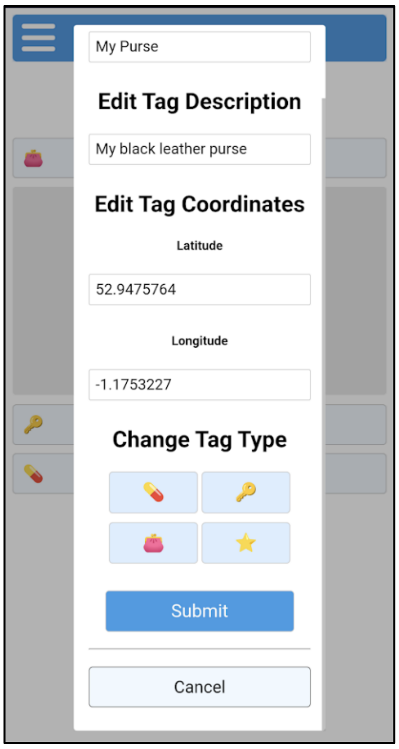


If you've picked up your tag, you'll need to place it again.

Place your phone where you want to place the tag and select the “Place” button, then confirm the location.



If you need to change your tag’s information, select the “Edit” button. This will allow you to change your tag’s name, description, and icon.


My Purse

Edit Tag Description

My black leather purse

Edit Tag Coordinates

Latitude

52.9475764

Longitude

-1.1753227

Change Tag Type

Submit

Cancel

Select the “Submit” button to save the changes.

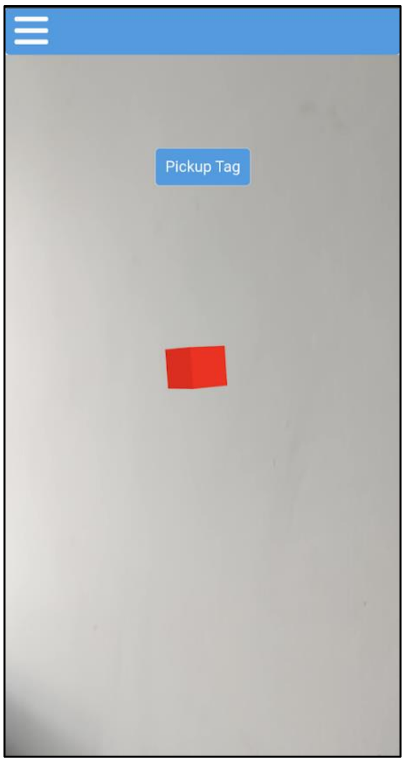
If you no longer need a tag, you can select the “Delete” button. Simply confirm your choice and the tag will be deleted and removed from your tags list.

Finding Your Tags

Finding your tags is simple.
Select the tag’s “Find” button as mentioned earlier.
You may be asked for permission to use your camera, allow this to find your tag.

You will then be able to use your camera to locate your tag.

Each tag is represented by a floating 3D object at the tag’s location.
Simply locate this object using your camera and move towards it. The object should grow and as you get closer.

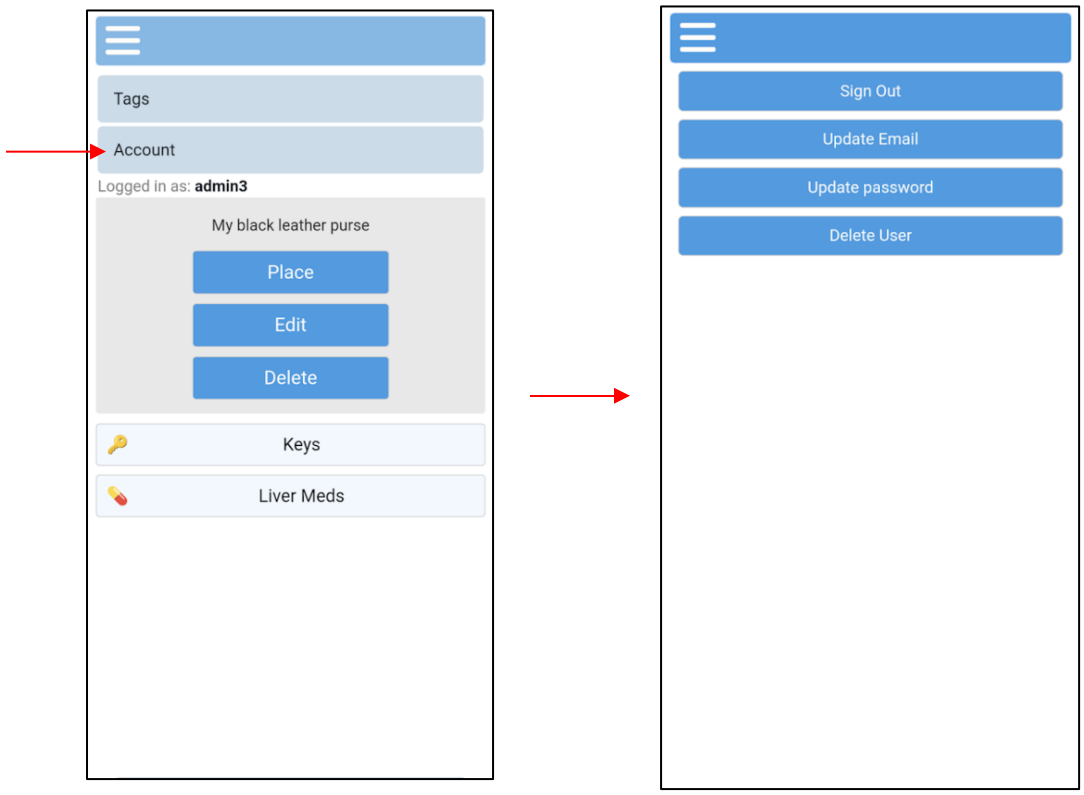


When at the item’s location, simply press the “Pickup Tag” button to indicate you have picked your item up.

Don't forget to place the tag when you put your item back down.
You will receive reminders until you place or delete the tag so that you don't forget.

Your Account Page

Your account page is accessible by selecting the three white lines in the top left corner of your screen.



On this page you can

- Sign Out
- Update Your Email
- Update Your Password
- Delete Your Account

To sign out, select the sign out button and this will sign you out of our service. You will have to log in again to view your tags.

If you want to change your email or password. Select the corresponding option and enter your new email or password. Then submit the changes.

If you have decided you no longer want to use our service, you can delete your account. This will permanently delete all your tags and data stored on our server.

Commonly Asked Questions

Q - How many tags can I have at once?
A - As many as you like!

Q - I can't see the 3D object while trying to find my item, how do I fix this?
A - Sometimes the object fails to load in, just refresh the page and it should appear.

Q - Am I able to use the app without an internet connection?
A - Unfortunately we require an internet connection to access your tags.

Q - Why do you need access to my location and camera?
A - Without these the AR wouldn't work and we would be unable to help you locate your items.

Q - How do you store my data and is it secure?
A - All data is stored in a private database and sensitive information such as your password is encrypted. We understand that your data and privacy are very important to you, that's why we only ask for minimal information.

Q - How accurate is FindAR?
A - While not pinpoint accurate we aim to be within 5-10 metres of your tagged location. This way we can at least help you identify the area your item is in.

Software & Hardware Requirements

FindAR doesn’t require a lot to function, we just need a few simple things.

- A mobile device (phone or tablet) with a back camera and capable of location services
- A stable internet connection

You can access the app on any other internet capable device. You will be able to view & manage your tags however, you will not be able to place or find them.