Maintenance Plan Summary

Sustainable Work through Women-in-tech Application for Older Women in Malaysia and Thailand: Integrating Action Research and Design Science Approach

1. Authentication

The login system utilizes Firebase Phone Authentication. The users are stored in the realtime firebase ‘users’ object.

The web application’s log in and sign up was initially created using firebase phone authentication system. However, due to client’s requirements to remove the firebase’s mandatory SMS verification, the login has been performed manually. Due to this, at any point in time whenever the user is logged in to the system, firebase authentication is not used. To keep this consistent, although the system utilizes firebase phone authentication on user sign up, we do not use it across the application. Therefore, there is a lack of firebase security rules against unauthorized read or write access to the data. This is something that may have to be improved upon in the application in order to ensure the security of user data.

2. General improvements for the application

2.1 Multi-region deployment

The application is currently deployed on a single server located in the US, this method of deployment causes a multitude of issues regarding network quality For starters, since users can connect to the application from various locations throughout the globe, they would often experience high latency due to the fact that not everyone has close proximity to US servers. Secondly, various ISPs would often route users to the application differently, which creates a large inconsistency in application responsiveness. This effect is also seen in our tests where we observed that routing with Unifi’s DNS servers significantly lowered our application’s performance as opposed to using Google’s DNS servers.

Hence, by deploying our application in more than one region, the chances of users getting high latency or low responsiveness would in turn be reduced. However, due to the lack of support for such a feature by our current hosting provider, Firebase Hosting, this feature has not been implemented. Not only that, without a hosting provider providing the feature in-house, developing mechanisms to route users to the best server based on network conditions on our own also incurs significant development costs which are not present in this project.

2.2 Language Cookies

The language selection functionality stores cookies. At times, cookies may persist from one session to another. This can cause issues with the translation as the wrong button may be disabled and/or the translation may translate to the wrong language.

This issue can greatly confuse the user and cause them to want to leave the application. Therefore, it is of a high priority to make sure to delete any previous session's translation cookies related to the women in tech web application. The cookie's deletion is best to be done on the landing page, as this is the first page the user sees. This will ensure that all translation cookies are only related to the current session.

2.2 Access restriction for unauthenticated users

Currently, pages of our application can be accessible without being authenticated. This poses a problem as user data can be exposed to any individual with knowledge of the URLs of the application's different pages. For example, a malicious user will be able to access the admin dashboard without knowledge of anyone in charge of the application and conduct malicious acts such as reselling users' information retrieved from the admin dashboard.

One of the solutions to this problem is to implement visibility restrictions client-side when the page loads. It is a trivial solution as the browser would simply hide the webpage, preventing the user from interacting with it. However, this solution presents a vulnerability as it still does not prevent a skilled attacker by injecting scripts to unhide the page UI.

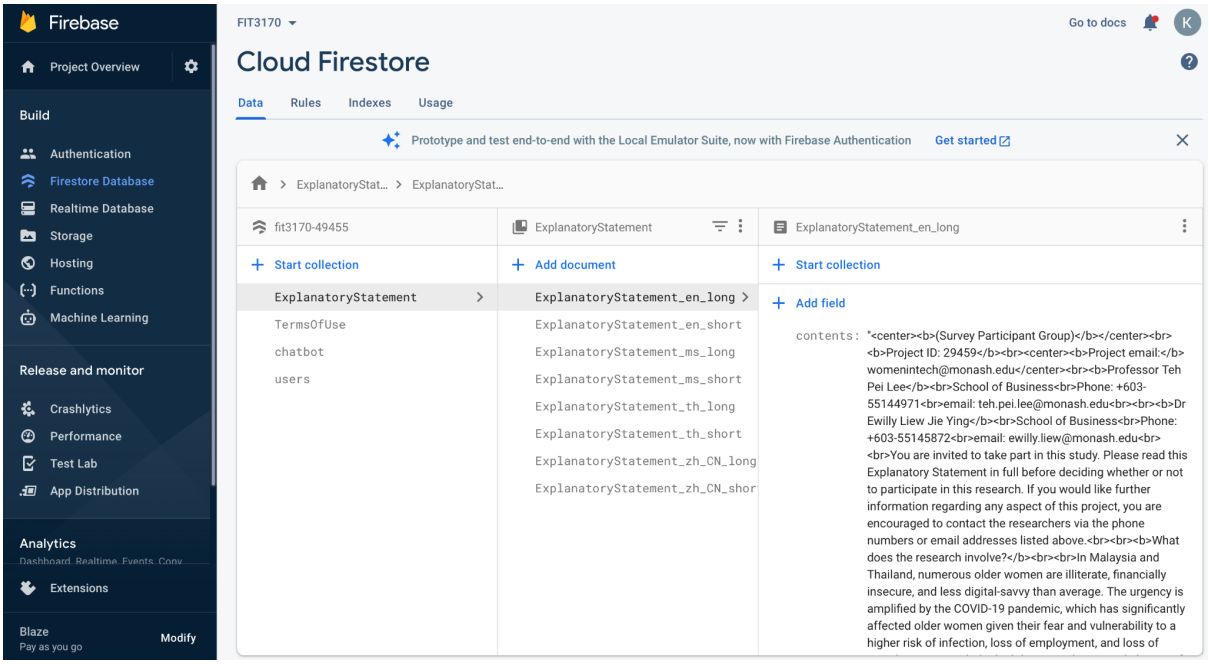
A non-trivial but robust solution to this problem is to implement visibility restrictions server-side before the page loads. It is a non-trivial solution for a multitude of reasons. Firstly, our current application renders pages client-side, hence implementing such a solution would require large refactors to take place in order to modify pages to be rendered server-side. This solution prevents skilled attackers from avoiding visibility restrictions by injecting scripts to unhide the page as the server doesn't serve the page until an authentication token is supplied. However, this approach incurs significant development costs due to reasons mentioned above.

This feature is not implemented in our application due to time constraints.

3. Chatbot

3. 1 Maintenance

The chatbot section of the application mainly uses the project firebase's firestore database to store the survey questions and responses, the Explanatory Statements, Terms of Use and user's information on their survey. Upon entering the Cloud Firestore on firebase, we have 4 main collections being "ExplanatoryStatement", "TermsOfUse", "chatbot" and "users". The chatbot uses all these collections except "TermsOfUse" which is being used by the Sign Up page.



3.1.1 Keywords

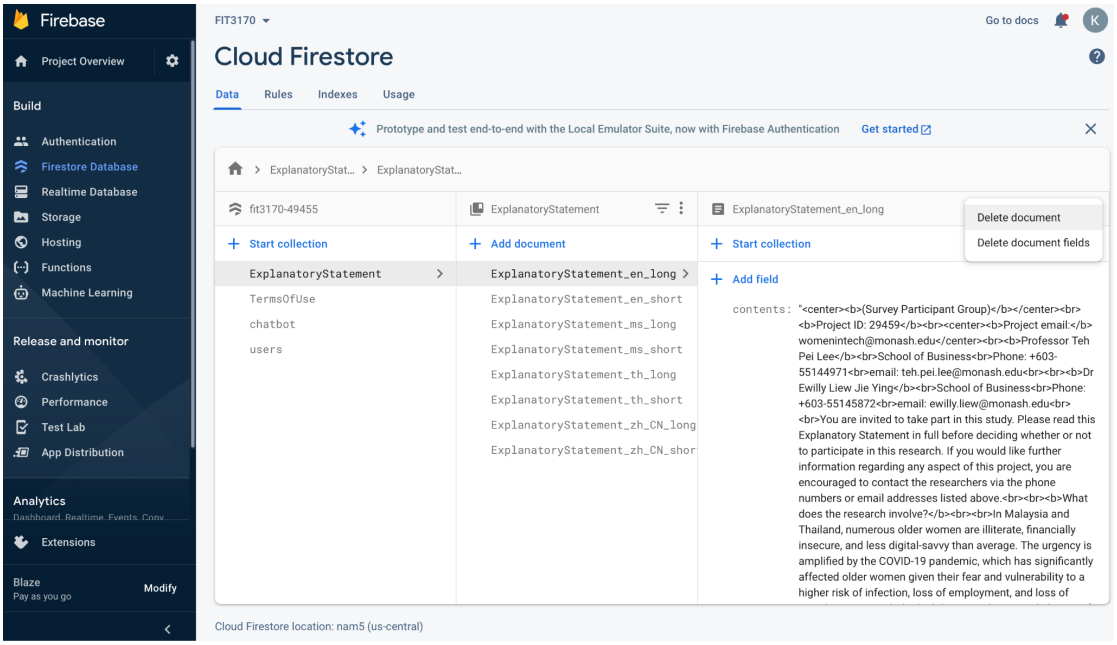
Seeing any of these words in a file, collection or document name on firebase or in the project's public folder means the following:
"en" - English
"ms" - Malay
"th" - Thai
"zh_CN" - Chinese (Simplified)
"long" - Long version
"short" - Short version
"Uploader" - this file has a function to upload something to firestore

3.1.2 Explanatory Statements Firestore Collection

This collection is used by any files that have the prefix “explanatoryStatement” in the script folder that is within the public folder of the project. It holds the 8 documents within it, 2 versions of the explanatory statement for the 4 languages the application supports. In each document, there is a “content” field that holds the formatted html of the explanatory statement to display on the explanatoryStatementPage.html or the explanatoryStatementLongPage.html. Onload of the explanatory statement pages, it will check if the required documents for the currently selected language of the user is on firestore. If it is, it will get the document’s “content” to display onto the page via innerHTML. If the required document is not on firestore, the page will use the function from the relevant explanatory statement uploader file to upload the missing required explanatory statement document then the page will get the newly uploaded document’s “content” from firestore to display onto the page.

If you want to change the explanatory statement document that is being displayed:

1. make the changes to the relevant explanatory statement uploader file’s function’s content variable.
2. delete the explanatory statement document that was changed from firestore manually as shown below.



3. Go onto the page where the changed document is used in the language of the changed document. For example, to upload a changed short version of the chinese translation of the explanatory statement, go to the explanatoryStatementPage.html of the app with chinese as the selected

language then wait as the page will automatically upload the recently deleted document back onto firebase with the changes made in step 1.

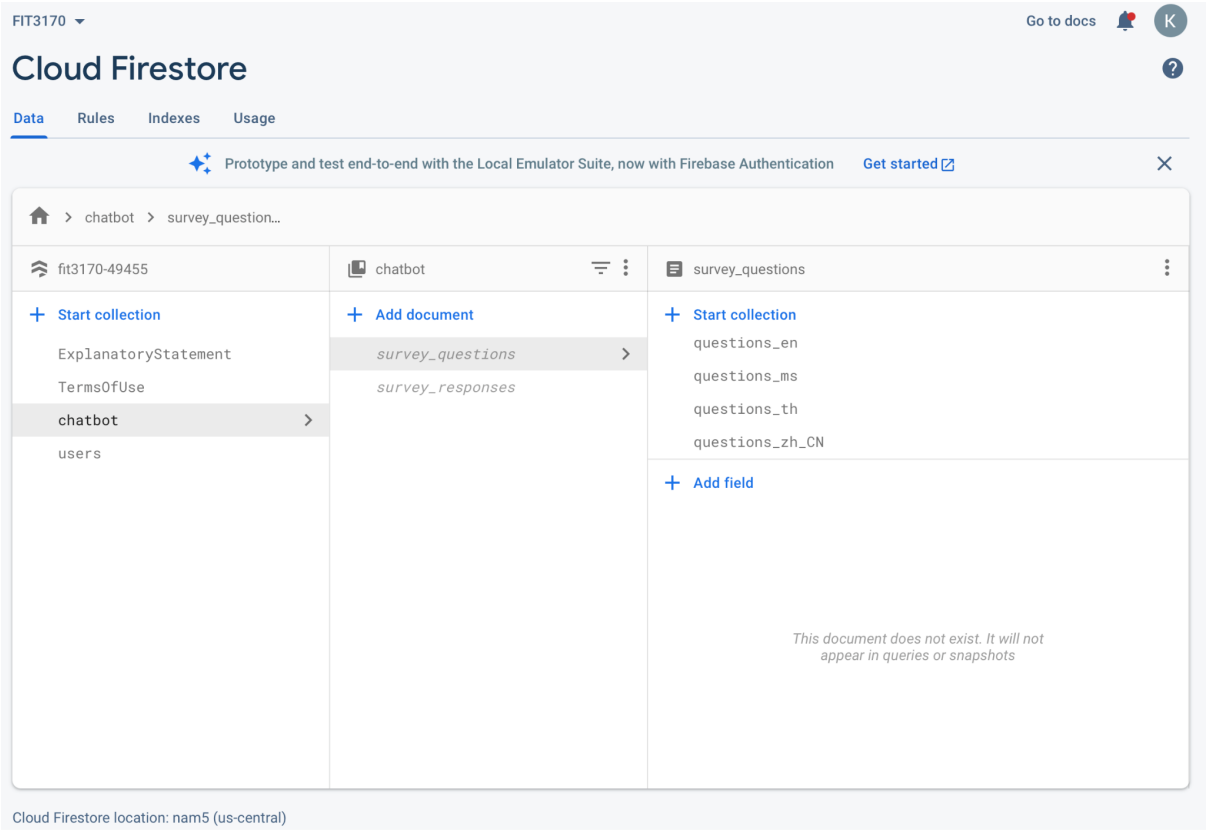
This whole process also applies to updating and uploading the terms and conditions for the sign up page.

3.1.3 Chatbot Firestore Collections

In the chatbot collection, there are 2 documents that contain collections within them which are the “survey_questions” and “survey_responses” documents.

Survey Questions

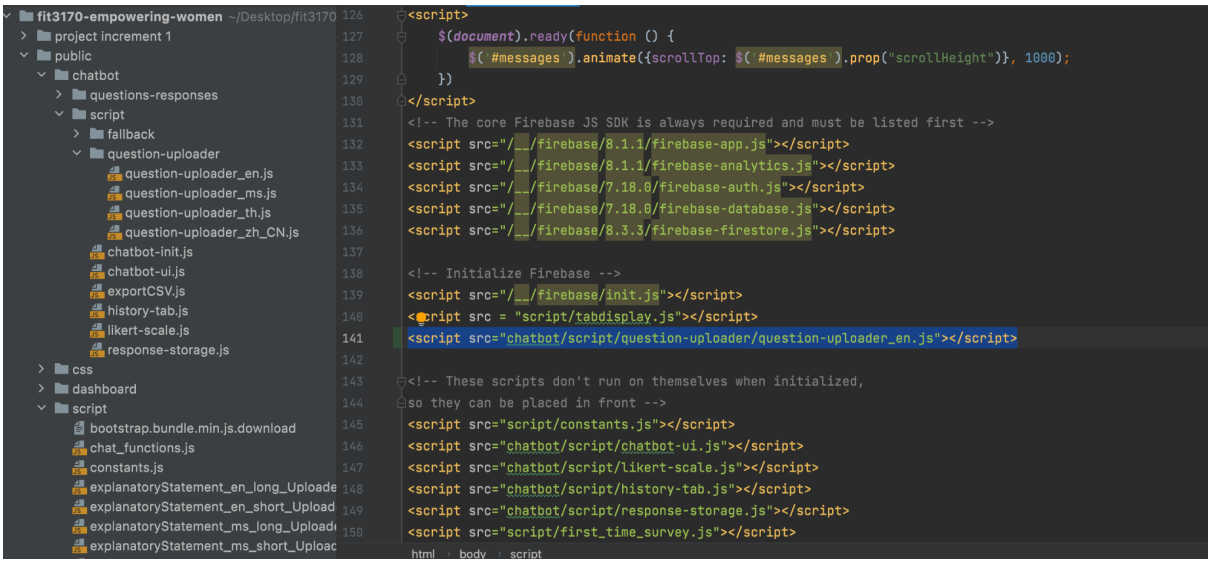
This document contains 4 collections as shown below. Each collection holds the documents detailing each question in the survey respective to the languages the collection is named after.



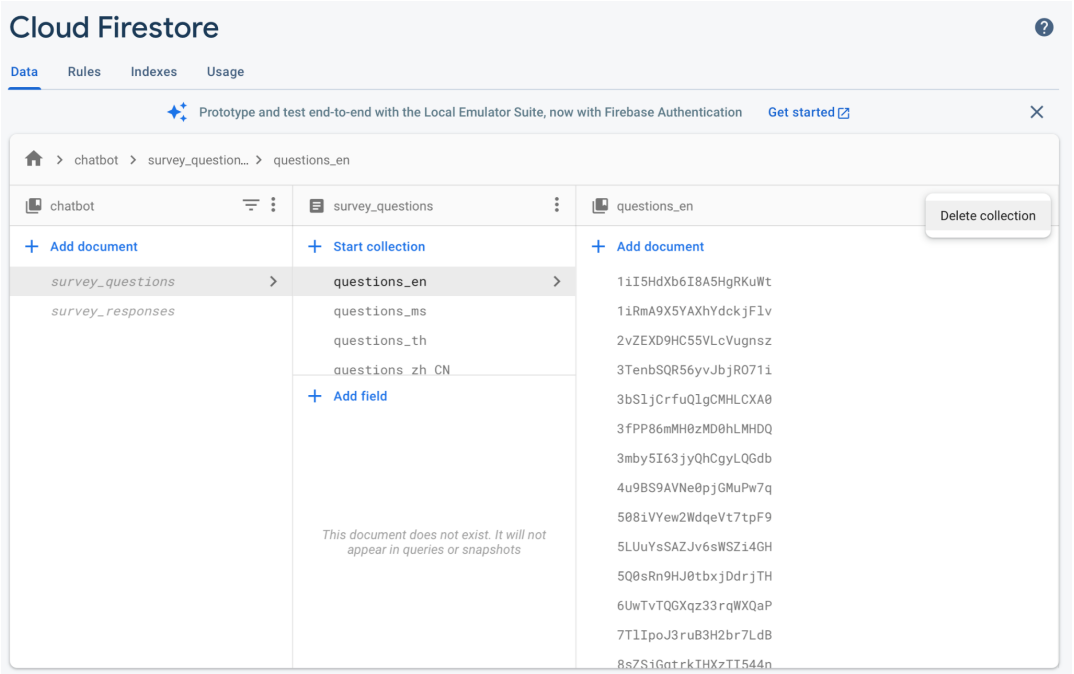
If you want to update or reupload the questions for the survey for a particular language, Please do the following:

1. Make the changes in the relevant question uploader files (found in chatbot > script > question-uploader).

2. Add the changed question uploader js file as in a script tag in a html page of the app (Preferably the chatbot.html file) after the firebase scripts tags. An example is shown below adding the uploader file in the chatbot.html.

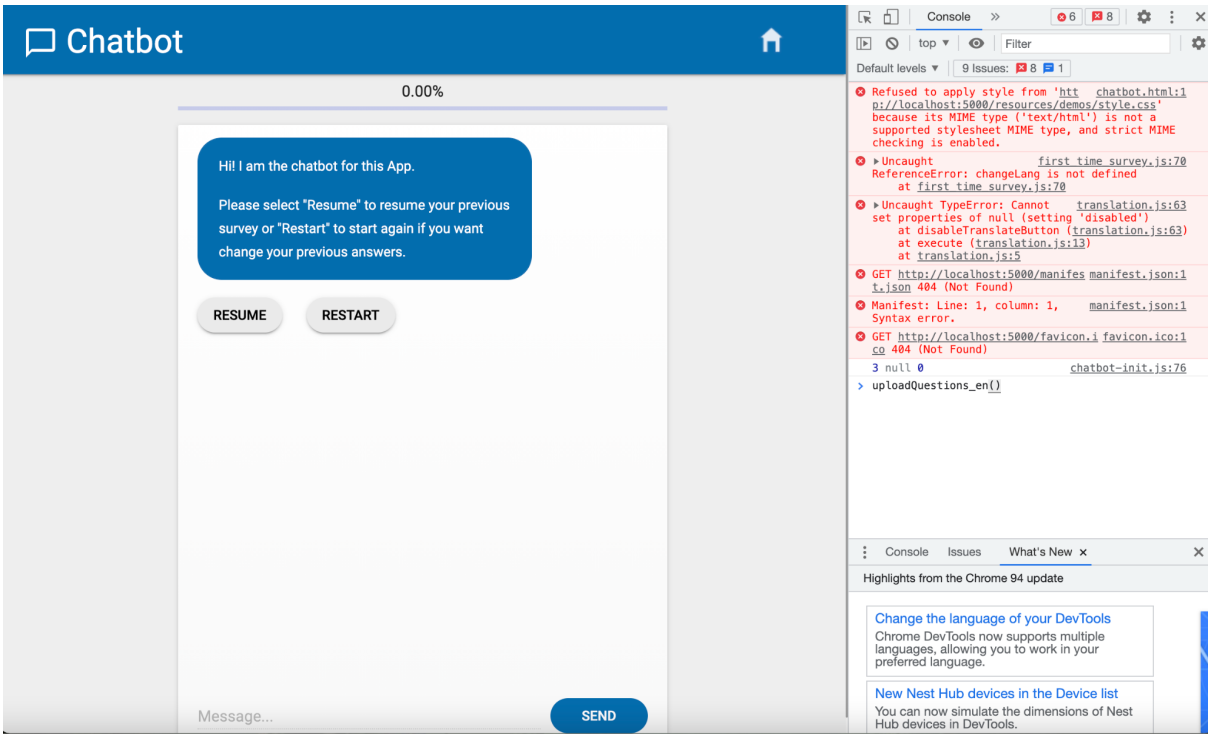


3. Delete the old question collection from the firestore.

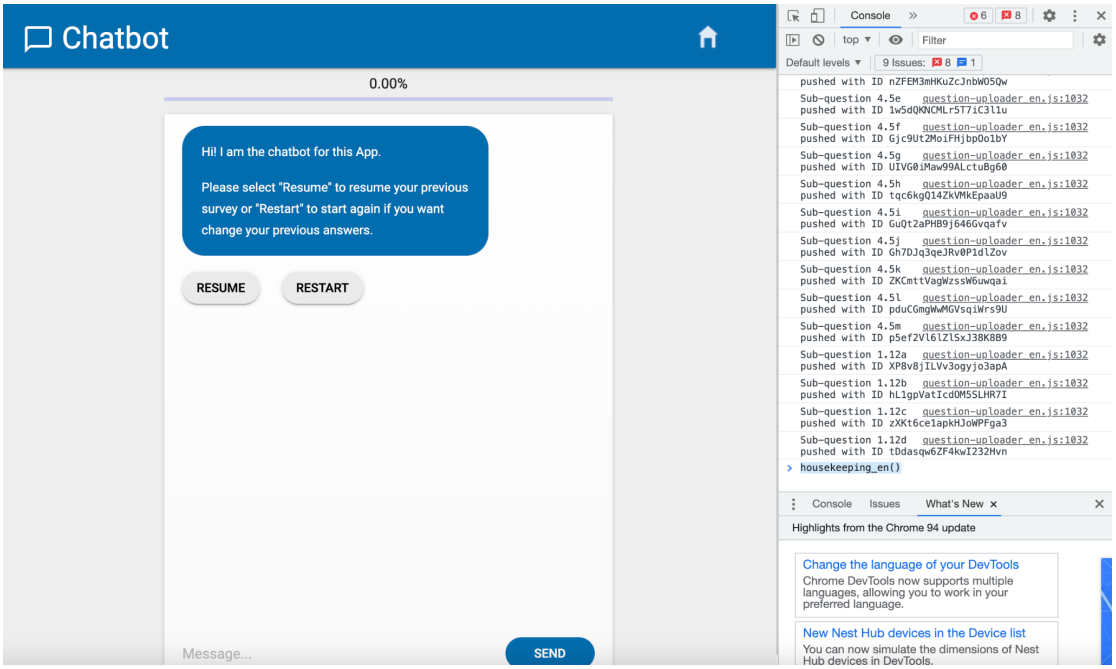


4. Host the application locally and open the local host link in google chrome. Then go to the page where the uploader file's script tag was placed and open google chrome's "Developer Tools". Then run the uploadQuestion function of

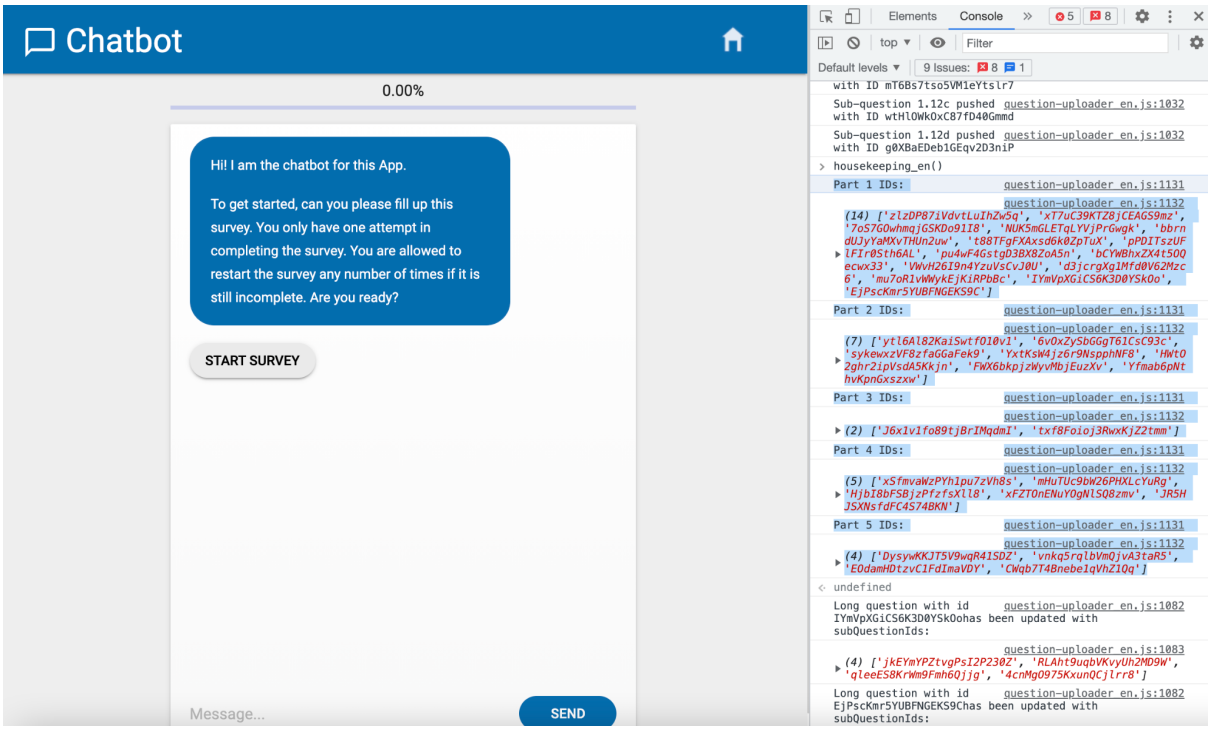
the uploader file in the console section of “Developer Tools”. (There will be a suffix to that function related to the language of the question uploader file)



5. Upon activation of the function, you will see words being printed out onto the console. They will say the question and what ID it was assigned in the firestore database. After the function is complete, **DO NOT CLOSE CONSOLE**. Next, we will **run the housekeeping function** of the uploader function to add the sub question IDs to their main question objects so that the chatbot knows where to navigate to when a question with sub questions is being answered.



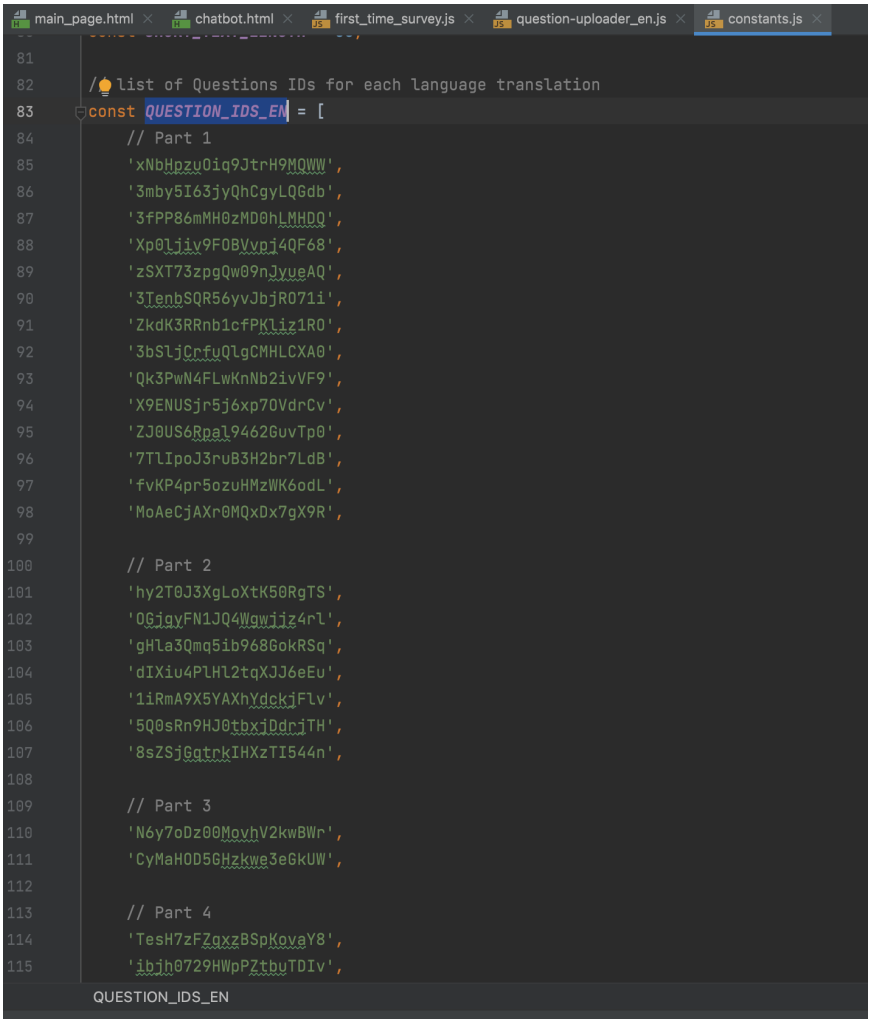
6. Upon the activation of the housekeeping function, more words will be printed out onto the console. Scroll up the console log to the part after the function was called and there are the Questions IDs for each part of the survey in order. (As shown in the picture below)



7. Save these IDs somewhere or save the console log by right clicking on it and select the “save as” option to save the console as a .log file.

8. Next, find the “QUESTION_ID” arrays for the updated survey question collection in the “constants.js” file which is in the public > script folder of the project. (The question ID arrays will have a suffix for the language of the survey question branch; see “Keywords” section)

9. Finally, copy and paste the saved question IDs of the newly uploaded questions into their respective sections in the array.



3.2 Plans for Production Systems

3.2.1 Enhancements that can be made to the application

1. Improved compilation time for survey responses

Our current implementation of the compilation algorithm is very inefficient and requires more than 10 seconds in order to compile the survey response data of all users. This is due to the fact that the compilation algorithm is not efficient.

Devising a more efficient compilation algorithm can then speed up the process of compiling survey responses into a CSV file. However this was not done in our current implementation of the chatbot dashboard. This is due to the fact that devising an efficient algorithm requires development resources that we lack due to our overloaded development schedule as a result of large amounts of change items requested by clients to be prioritized. Not only that, usage of a more efficient compilation algorithm will likely entail refactors to be done to the database structure, which incurs significant development costs.

On the other hand, the decision to not implement a more efficient algorithm lies within the fact that compiling responses is an action that is not performed as frequently as other more critical and essential features of the application, which require much more attention than this issue. With that being said, this decision remains the best possible choice we could've made given that it avoids hindering progress on more critical tasks.

2. Simplified process for modifying survey question content and logic

Development of the chatbot suffered from many delays due to the complicated process of entering questions and their associated logic to the cloud. This is due to the fact that clients had to supply us with a docx file containing survey questions as well as their associated logic, which we would then derive logic and subsequently enter the questions to the cloud in a multilingual manner. This process creates a significant time overhead when the clients wish to make changes to either the questions or their logic as the process involves a middleman, the developers.

With that in mind, it is possible to eliminate the significant time overhead by allowing clients to modify the survey questions and their associated logic themselves, such as by having a user-friendly graphical web interface for modifying question data and associated logic. Due to resource constraints, this was not possible in our project. This can be explained by the fact that such a feature, although simple at first glance, actually encompasses a large set of complex requirements, such as:

I. Allowing for the modification of question arrangement and content in realtime and for it to be synced with the cloud
II. Having modularized, executable survey logic that can be assigned to questions by the non-technical clients.
III. An implementation of promise-based logic that can determine the right order to upload related questions, such as subquestions.
IV. Multi-lingual support for string value assignment

These complex requirements create an implementation difficulty that is comparable with reimplementing the Firebase Console from scratch. It can be seen that implementing such a feature would require significant higher development resources than what is available to us currently. This is evidenced by the fact that industrial implementations of the Firebase Console exist and are often expensive:

I. https://firefoo.app/buy
II. https://retool.com/pricing

3.2.2 Missing requirements

3. Voice inputs

Due to a multitude of factors, this feature was not implemented in our application due to time constraints. To start off, other change items such as design changes and critical issues concerning the chatbot were prioritized by the clients over the implementation of voice inputs. Not only that, the clients also wanted features other than voice inputs to be implemented before the actual launch date (pre-launch). In addition, critical issues and changes raised during the development process of our application also had to be prioritized over the implementation of voice inputs. In short, voice inputs was not implemented as the team was occupied with other tasks that carry more priority than voice inputs.

With that being said, any development team should still be able to implement voice inputs using Google's Cloud Speech-to-Text API. However, the API cannot be run client-side and must be run server-side. Hence, the transcription logic has to be implemented server-side even though our application is rendered client-side. In addition, the implementation of voice inputs would also entail usages of Firebase's Cloud Storage API if the clients wish to store users' voice data.

Once the feature is implemented, costs would be incurred due to usages of Firebase's Cloud Function and Cloud Storage API (https://firebase.google.com/pricing), as well as Google's Cloud Speech-to-text API (https://cloud.google.com/speech-to-text/pricing) as a result.

4. Forum

4.1 Maintenance:

4.1.1 Post Delete function in the post detail page

In the post page of the forum, there is delete functionality, which is represented by the trash icon. The delete functionality is only allowed for an owner of the post. Hence, other users cannot delete any posts that do not belong to them. Once an owner of a specific post clicks the icon, it deletes the post data, replies, comments, and likes on comments of the post.
Issue with the function:

If a developer decides to add a firebase JSON Object for the forum, linked with post data then they may have to add additional code that deletes the data from the JSON Object added.

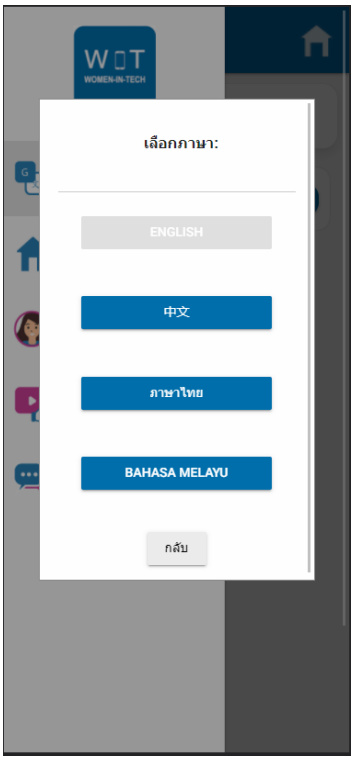
4.1.2 Authentication and Authorization

For some of the forum functionality like post deletion, the forum submodule makes sure to check whether the user is signed in, and makes sure the signed in user is the creator of the post before the system will allow them to delete it. The authentication is performed manually. Upon logging in, the user's phone number and username are stored in the local storage under the “USER” key. Not to mention, the forum.html, and post.html check the logged in user’s credentials before they display any data to the user.

Upon making changes to the authentication system as stated in section 1, it may be required to change the way the forum authenticates the users in order to allow them to view or interact with data.

4.1.3 Language Selection

For translation purposes of the application, google translate element was used. The related code is located in the translation.js file. Upon switching a language in the language selection modal:

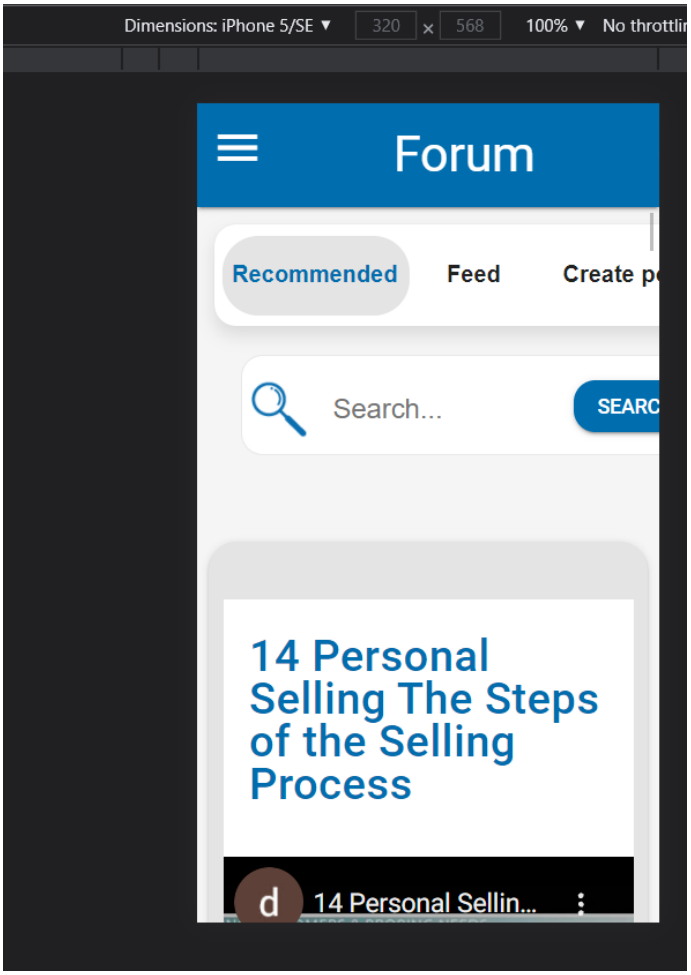


It is important to note that the language selected is to be stored in the local storage as the value of the button. This is used in order to disable the current language's button in the translation modal. Forgetting to do so may result in the user clicking the same translation button multiple times, which, in turn, may cause unexpected behaviour. Therefore, upon adding new languages, the button value has to be stored under "LANGUAGE" key in the local storage and appended to the disabletranslateButton function in the translation.js file.

4.2 Plans for Production Systems

4.2.1 UI Issues

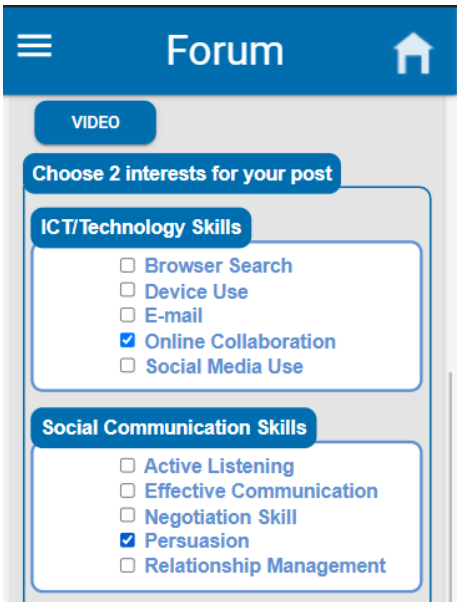
1. Not-responsiveness for some screen size



The forum submodule was designed to be used both on mobile and desktop devices, however there are some screens that were not taken into consideration and thus the UI does not fit them properly. For example, the Iphone 5 screen was considered outdated, and it's screen size is not supported by the application.

Considering the fact that the main target users of the application are older women, they may not be up to date with technology and use outdated devices. Thus, this issue needs to be resolved by ensuring that the application is responsive for as many screen sizes as possible in order to expand the potential users range.

2. Small elements



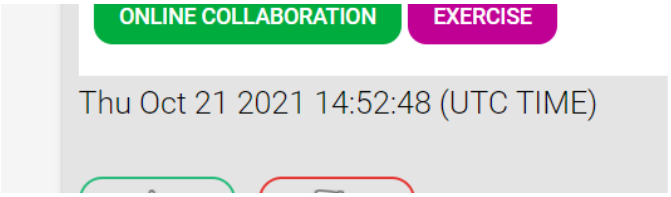
Considering the target audience of elderly women and knowing that the application is to be used on the mobile phone, the check boxes at the create post section must be significantly increased in size. The checkboxes are hard to click from the phone, which degrades the usability of an application.

3. Lack of alerts

Forum does not give any feedback to important user actions, such as deleting or uploading a post, which may leave the user confused on whether their action was performed successfully. In order to improve that these alerts are to be added into the application:

- Confirmation box for deletion of the post
- The post is posted successfully alert
- The user cannot upload empty comments/replies
- The post is successfully added to favourites alert

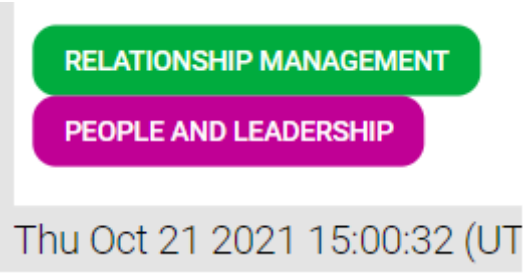
4. Time representation on posts and comments



Time on the posts and comments is represented in the UTC time format, which may be really inconvenient for most of the users, since in order to convert it to their own time zone, they have to be aware of the difference between UTC time and time zone in their country. In order to improve this, the application should request the

user for their location access and if the user accepts it, the time should be converted into the proper time according to the user's time zone.

5. Interest representation

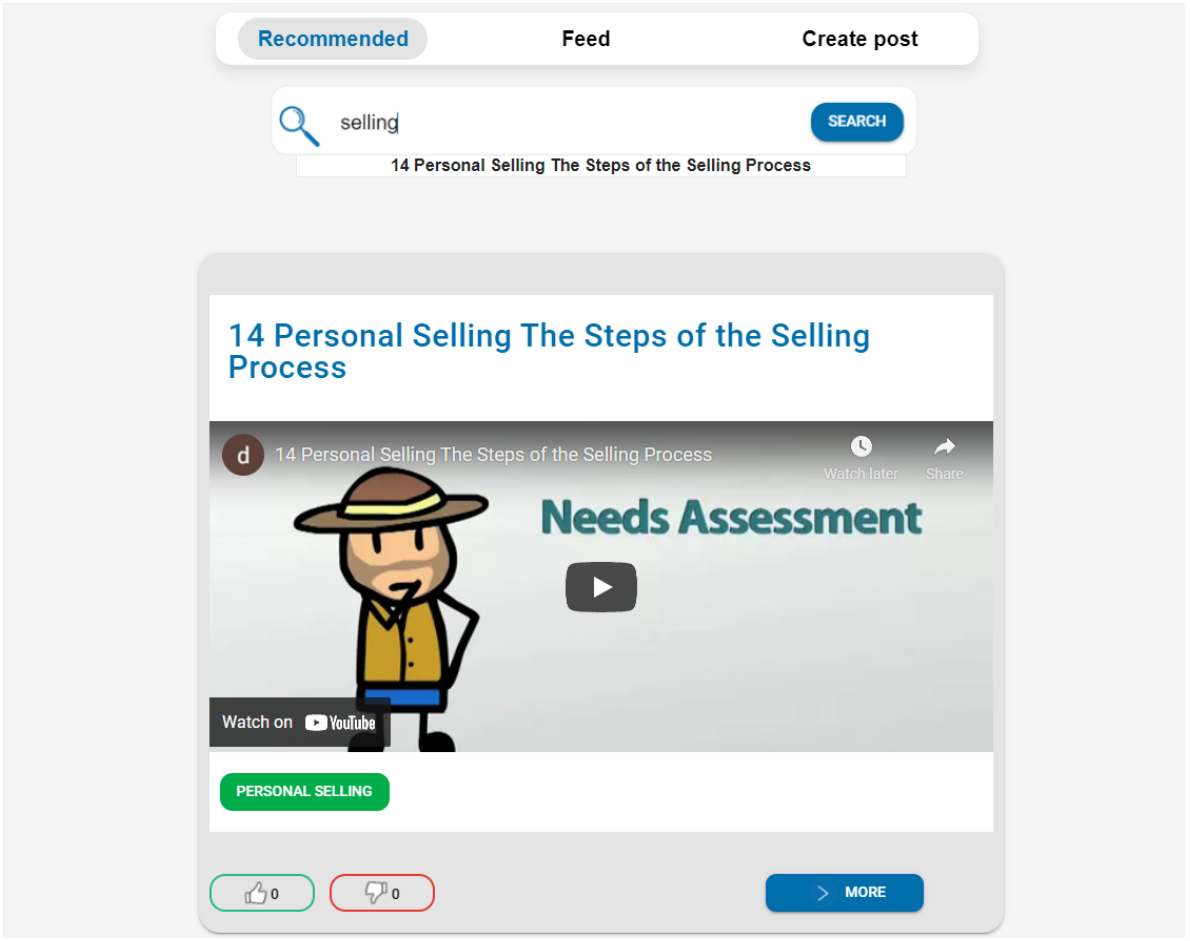


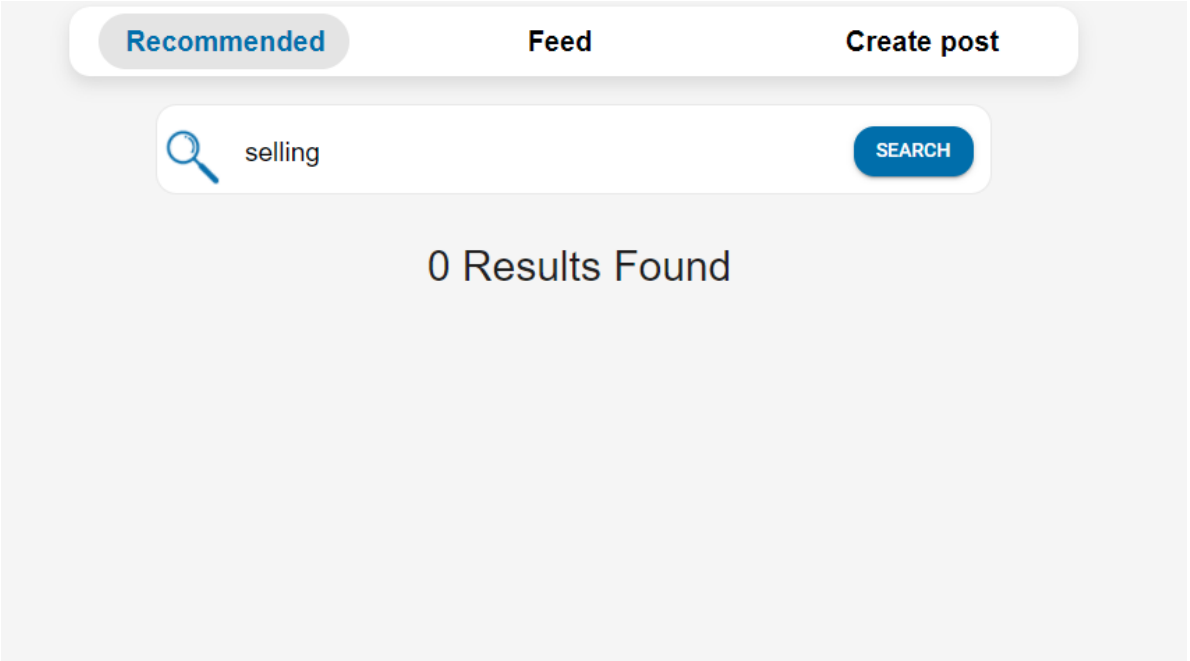
The representation of the interests from the mobile view could be improved by changing the placement of the interest components further apart in order to enhance the visibility.

4.2.2 Search Function

The search function is able to find posts based on their title and their interests. However, when making a search the title or the interest interred has to be case-sensitive and start from the first character of the post title / interest that you are looking for.

For instance, if the post title is "How to manage", the valid search for this title will start with the "H" character. Searching for "to" or "manage" or "ow", will result in no results found. To counteract this, the team has created an autocomplete on the search that allows us to suggest to the user the post titles based on the entered input in a non-case-sensitive manner and doesn't have to start from the beginning of the post title:

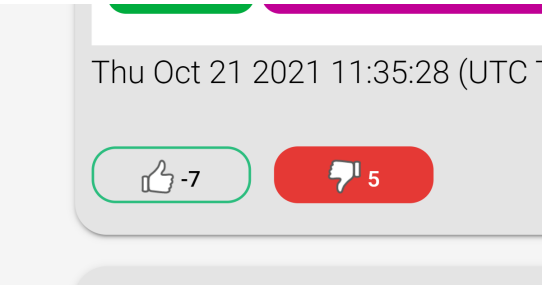




This autocomplete function only shows autocomplete for post titles and not interests.

To improve the submodule it is preferable to adapt the search function to allow users to search for post title and interest using substrings as well as the entire title, in a non-case-sensitive manner. Not to mention that the autocomplete should also suggest distinct post interests.

4.2.3 Likes/dislikes on Posts and Likes on Comments



Likes and dislike buttons exhibit erroneous behaviour if the buttons are being clicked too fast. The number of likes and dislikes presented on the button is stored in the database as the likes and dislikes attributes, which is getting updated every time the user clicks on any of the buttons. Since the time taken to access and update the database is highly dependent on the user's internet speed and is significantly slower than the speed of I/O interactions, the user may perform several I/O actions while the database is updating values for only one of the performed actions. This leads to the number of likes and dislikes being decreased/increased wrongly (e.g. user liked and immediately disliked the post led to decreasing number of likes by 2).

In order to prevent this issue, the alert that doesn't allow the user to double click the like/dislike button was created. This way the fault is reduced but not completely eliminated, since it still persists when the user clicks the button with a certain speed.

To completely resolve this issue, the application should make sure that the database has finished updating and only after that the user may perform the next action. Moreover, the number of likes/dislikes on the buttons may be identified by counting the number of likes/dislikes for that post in the database rather than storing the number of likes/dislikes as an attribute for the post. That way it would be easier for the developers to work with the database as well, since currently once like or dislike is deleted from the database, the likes and dislikes number has to be manually updated as well.

4.2.4 Reply Function In Post Detail Page:

The reply function allows the user to reply to another user's comment/reply by clicking on the add reply button. Once the add reply button is clicked, an input text box will appear for the user to write their reply in. They then click on the send button

to post it. In the input text box that appears after clicking the add reply button, you will notice the presence of “@” followed by the username of the reply that the user is replying to. This functionality resembles the same way that instagram formats their comment section.

Despite the user’s ability to interact with other users by commenting and replying. The comment section reloads every single time a user adds a comment/reply under the post. This can be inefficient and causes confusion especially since when the comment section is reloaded the user is taken back to the top of the page rather than the comment/reply that they just made. The feature was implemented this way in order to allow the input text box to reload and empty from the previous comment as well as print the comment and replies in the correct order.

Code implementation of the functionality is not clean and will be extremely difficult to maintain. Due to the way the comments and reply section is implemented (using indexes to specify divs that the html will print to/ take input from), the code has multiple functions that execute printing replies (i.e. printReplies, printRepliesToReplies), multiple functions to show the input text box (i.e. ShowReplyInput, ShowReplyToReplyInput, and showReplyToReplyToReplyInput) and multiple functions to get the reply from the front-end and save it in database (i.e addReply, addReplyToReply, and addReplyToReplyToReply). All these functions have similar codes and are very repetitive resulting in the complexity of the code and making it less readable.

With all the issues stated above, the functionality needs to be maintained by finding a way to implement the comment section without the need to reload it after each reply/comment made by the user. It is important to also find a more readable and efficient way to implement the functionality by reducing repetition in the code.

4.2.5 CSV Export function In Admin Dashboard page under forum tab

The csv export functionality found in the admin dashboard main page under the forum tab is used to export all the data related to the forum from the firebase realtime database.

Once the button is clicked, it generates 5 csv files, which are forum_Comment_data that contains comment data of the forum, forum_LikesComments_data that contains who liked which comment, forum_Post_data that contains post data of the forum, and forum_User_data that contains all the users present in the application.

Since the firebase realtime database is not relational, if one wishes to find related data from a specific index in the file content, such as post id, they have to manually lock up the related data. For example, if they want to look up comment data with a specific post id, then they have to check each post Id of the comment data to see if it is identical with the specific post id they are looking for.

As mentioned above, the files are not formatted for the relational database and hence it cannot be converted into a specific format that the relational database may require. Hence, it can be improved to be able to convert all the NoSQL data into a specific format that the relational database may require so that data analysts can analyze the data, using relational databases.

The second improvement point is to include data that contains all the replies, comments, likes and dislikes, and likes on comments of the posts that users interact with the most. This improvement point was suggested by the clients, however, since the request was optional, we decided not to implement it due to the time constraints.

4.2.6 Favourite Functionality In Post Details Page

The provided favourite functionality allows users to add the post into their favourites. As of now, the functionality is provided when the user goes through the details of a certain post and all the posts are available under the 'Feed' tab in the main Forum Page. How it works is when they click on the 'ADD FAVOURITE' button, users can see that the UI component of the same button has been changed which reflects on saying the user has successfully added the current post to their favourites. Furthermore, when they click on the same button, which is now called 'REMOVE FAVOURITE', it will run a function which removes the selected post from the user's favourites.

Despite the overall functionality working fine, it is quite inconsistent with other functionalities such as the likes/dislike functionality. The reason is because the functionality should also be provided when the user scrolls through the list of posts in the main Forum page and not only in the Post Details page.

Aside from that, as of now, all the posts that the user has added to their favourites previously in the Post Details Page and posts created by them can be found under the Feed tab in the main Forum page. Because of that, users may have a hard time differentiating between their own personal posts and their favourites. Another reason why is because users could favourite their own post as well which now makes it hard to differentiate between each post. This may affect the discoverability of the feature as it is not clearly shown to the user.

With all the issues stated above, one improvement to overcome the issue is to figure out an effective way on how to categorize the user's favourites from their personal post and output them so that it is easier for users to navigate themselves to find their favorited posts. Aside from this, future developers may also add in the functionality in the main Forum page which allows users to favourite any post as they are scrolling through the list of posts in the main page.

5. Recommender

5.1 Maintenance:

5.1.1 Viewing a video from favourites or history

Each learning interest/skill has one video each stored in the Realtime database under 'posts' as part of the trial run being conducted at the time of writing. Currently, when a user wants to watch a video that is in their favourite or history page, it will redirect them back to the main video player. The user will lose their previous progress from an earlier session if the category of the video is different. Thus, they will be unable to continue watching when selecting the playback buttons They would need to go to edit skills selection and select another skill to be able to continue watching different videos. As more videos will most likely be added in the future, this would need to be taken into consideration.

5.1.2 Adding videos of different languages to the database

Currently, videos that are in the database are all in English. Since users have the choice to select a different language anytime, there may be a requirement in the future to add videos of different languages. Thus, the system may have to account for the current language being used and display the corresponding videos. This would depend on the client's specifications, however.

5.1.3 Using Google Tag Manager for tracking videos and other site analytics data

Getting started

Video analytics are tracked and captured using Google Tag Manager (GTM) as it can natively support YouTube videos. The script to install GTM is shown below in the recommender_Ui.html page:

recommender_Ui.html > html > head > meta

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Google Tag Manager (must be placed in head tag)-->
    <script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
    new Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName(s)[0],
    j=d.createElement(s),dl=l!='dataLayer'?'&l='+l:'';j.async=true;j.src=
    'https://www.googletagmanager.com/gtm.js?id='+i+dl;f.parentNode.insertBefore(j,f);
    })(window,document,'script','dataLayer','GTM-NF4CK22'); //GTM-NF4CK22 is the container ID
    //console.log(dataLayer)
    </script>
    <!-- End Google Tag Manager -->
```

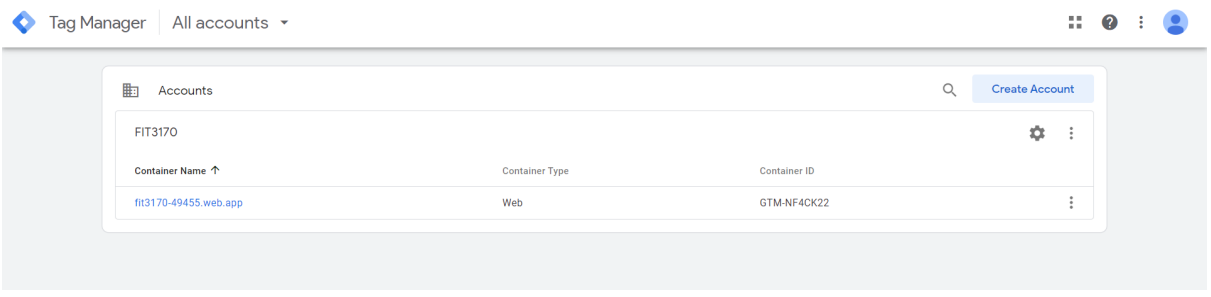
The following script is placed right after the opening <body> tag to tell the browser if the user does not have JavaScript enabled, render an iframe version of the Google Tag Manager Container to the page.

recommender_Ui.html > html > head > meta

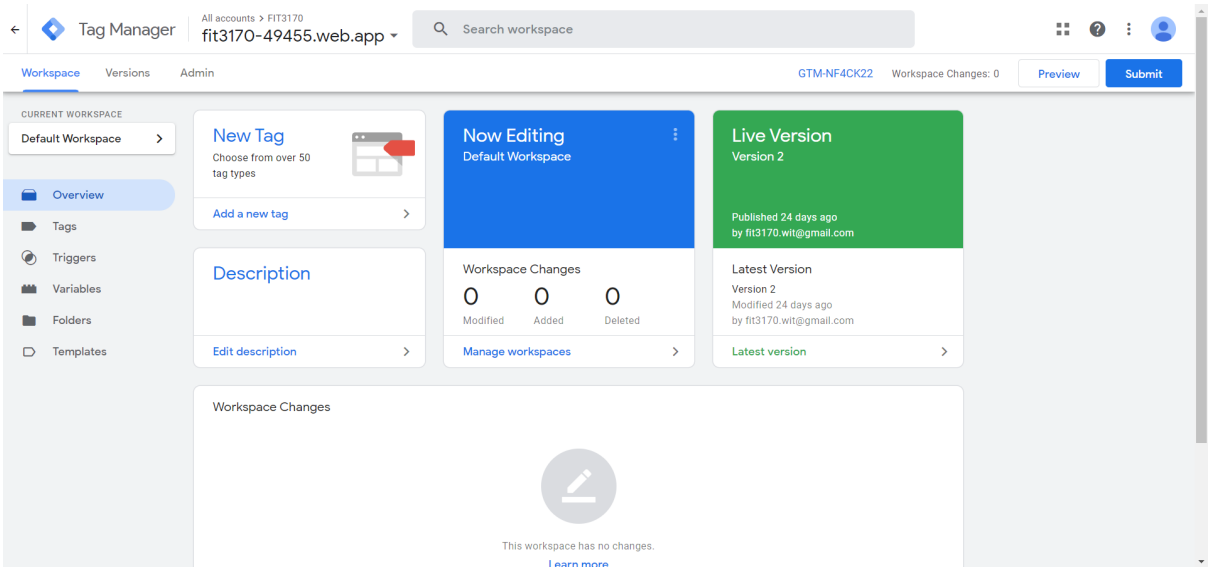
```
<body>
  <!-- Google Tag Manager (noscript) -->
  <noscript><iframe src="https://www.googletagmanager.com/ns.html?id=GTM-NF4CK22"
  height="0" width="0" style="display:none;visibility:hidden"></iframe></noscript>
  <!-- End Google Tag Manager (noscript) -->
```

GTM-NF4CK22 is the current container ID used by a non-university Google Account. **Please contact the following email (ylau0009@student.monash.edu) for the account details**. Once you have obtained the credentials, access the GTM workspace following the steps below:

1. Go to https://tagmanager.google.com/ and sign in using the given Google Account.
2. Once logged in, you should see the following page.



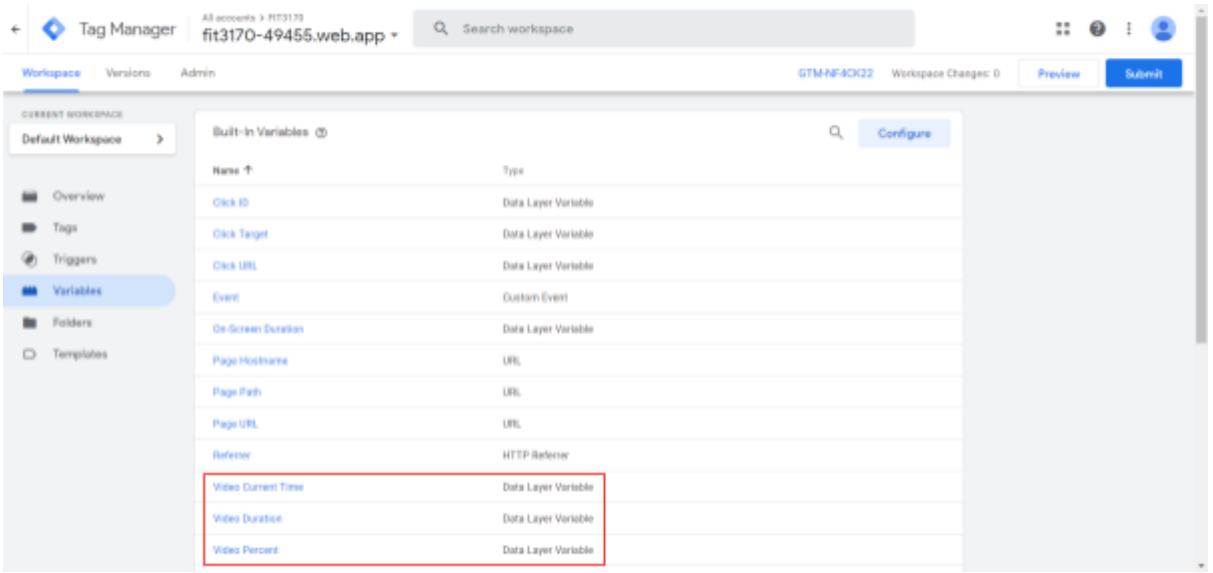
3. Click on the container name fit3170-49455-web-app to access the workspace.

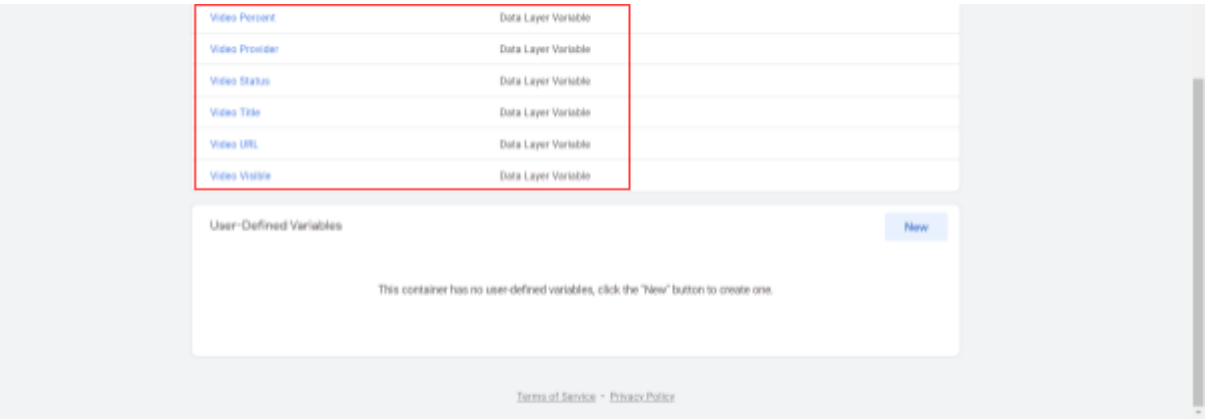


Brief overview

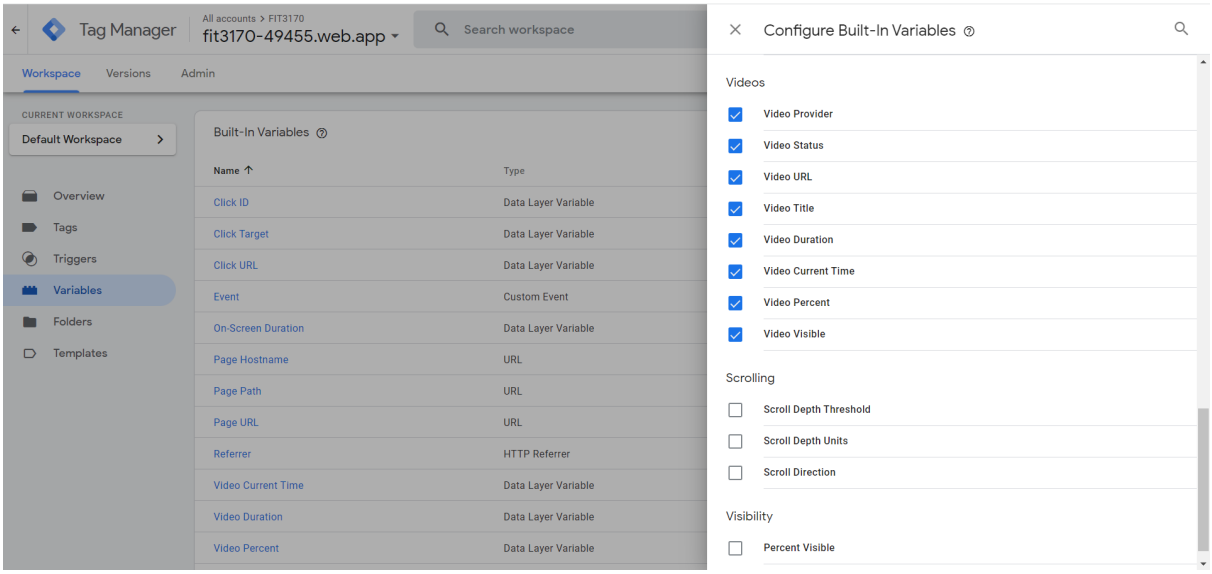
Various analytic activities such as page clicks and scrolling actions can be captured by GTM. In this case, for how tracking was setup for the YouTube player:

1. Click on the **Variables** tab to see the current list of built-in and user-defined variables tracked by GTM.

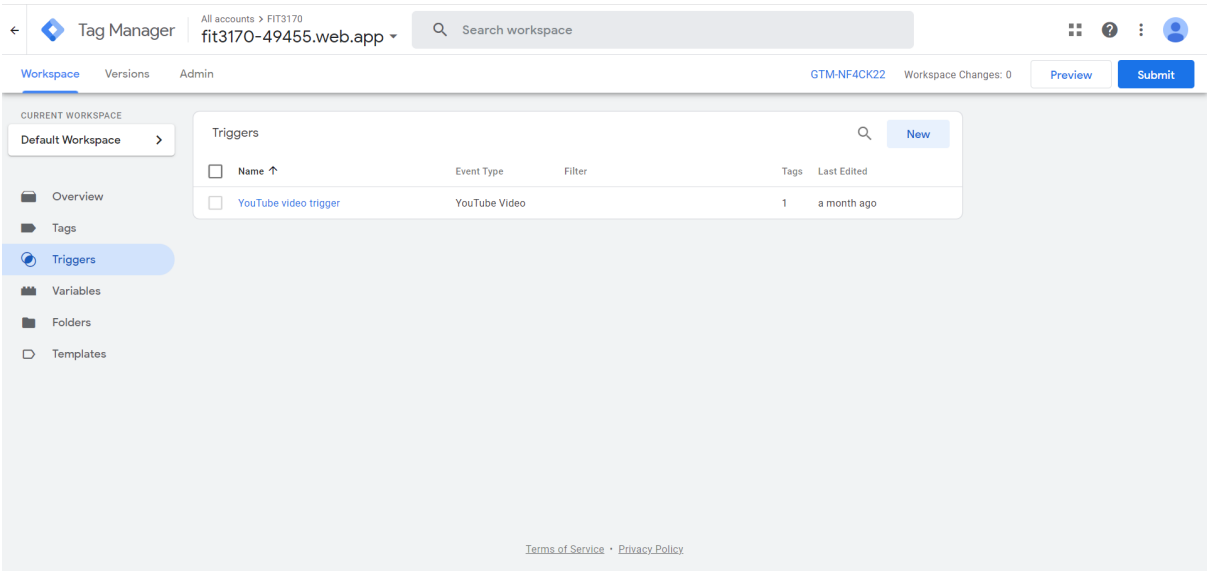




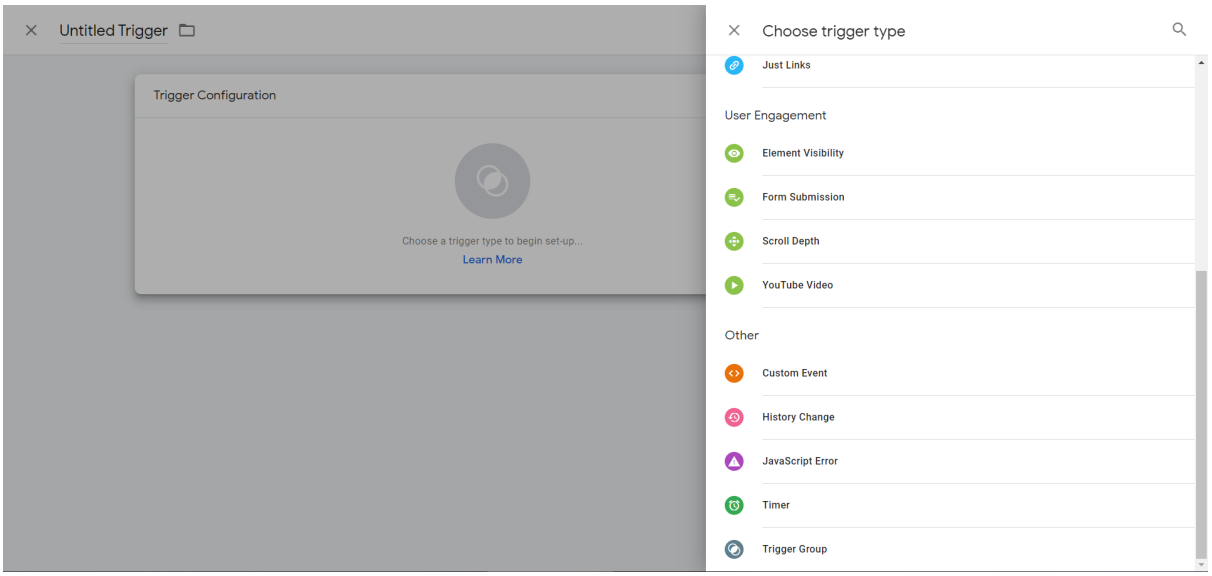
2. Since YouTube videos can be natively tracked, the variables can be configured by clicking on the Configure button above. If future requirements involve more analytics to track, GTM has some built-in options provided.



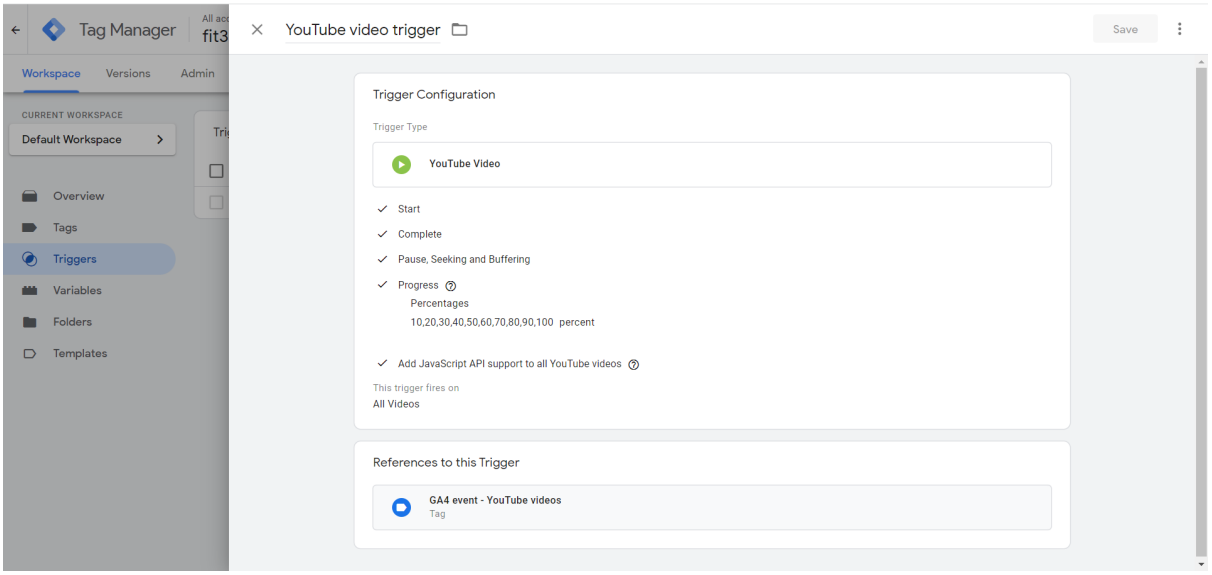
3. Once variables are configured, navigate to the **Triggers** tab



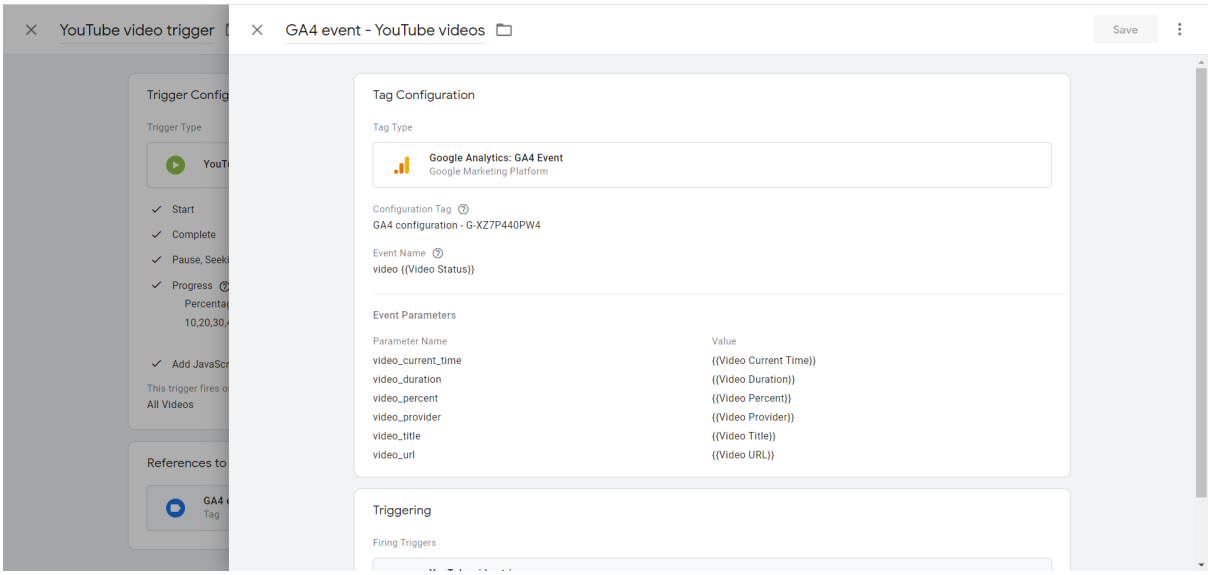
4. Here, a YouTube video trigger has been added. To create new triggers, click on the New button and then click on the area below the ‘Trigger Configuration’ header.



5. Go back to the Triggers page and click on the YouTube video trigger in the table



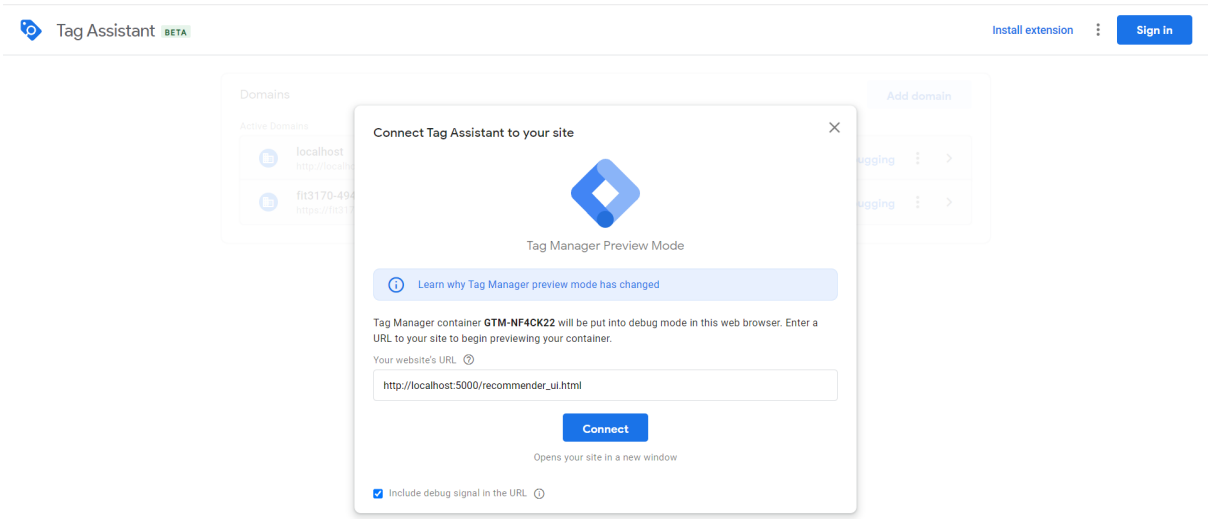
6. All triggers require a reference tag for the tracking to take place. Here, a Google Analytics GA4 event tag is used. To go into edit mode, click on any area below the ‘Tag Configuration’

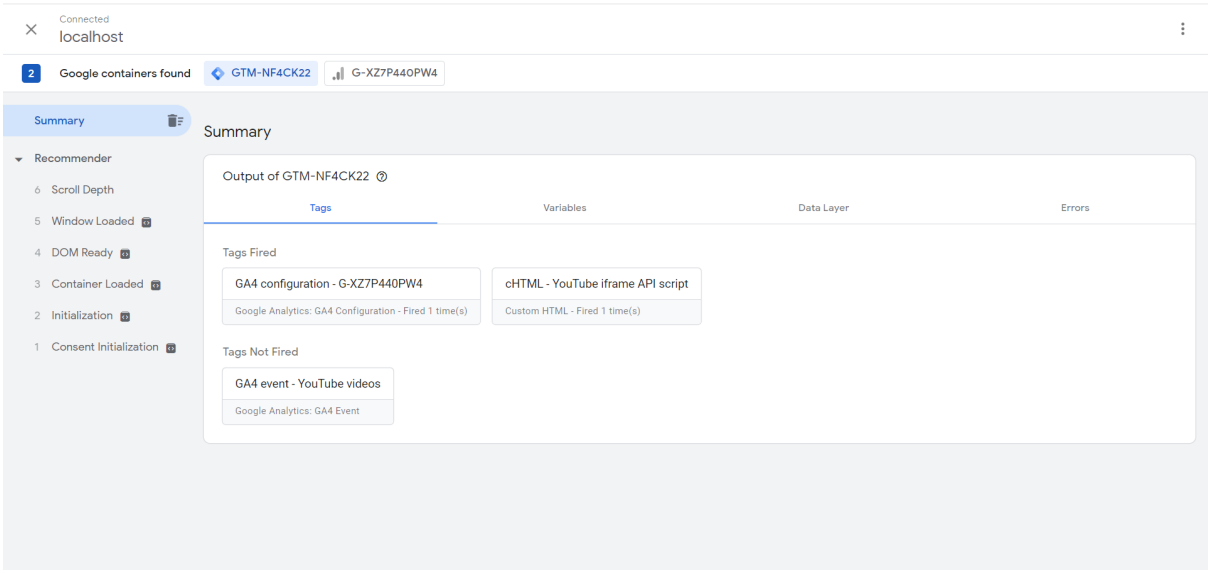


7. To access information related to Google Analytics, go to https://analytics.google.com/

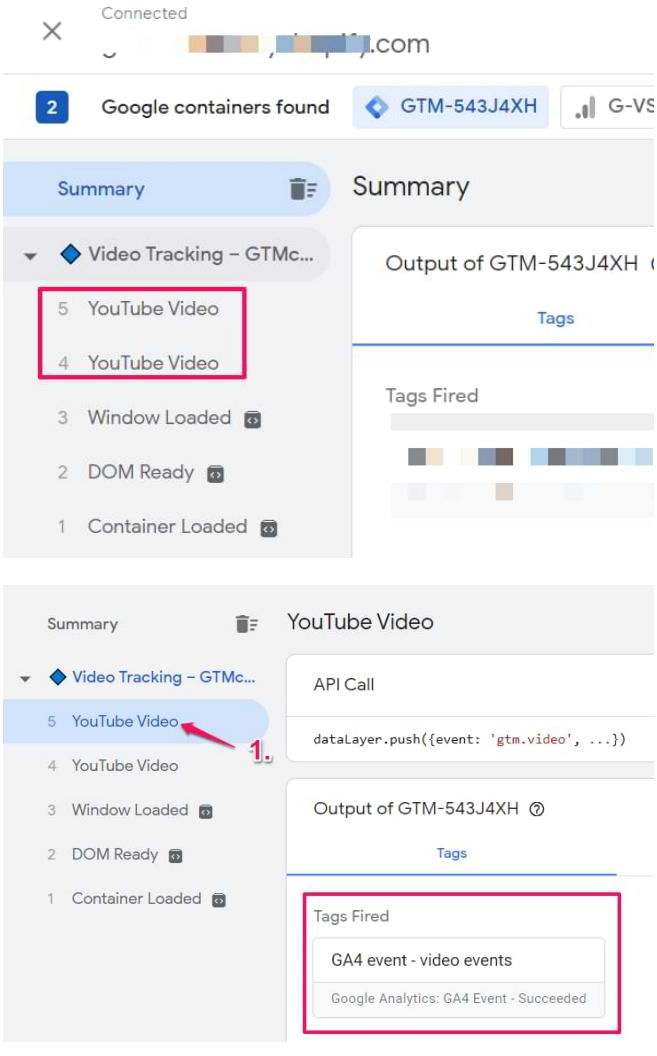
Previewing and publishing the container

1. On the main workspace page, click on the ‘Preview’ button at the top of the page to test the setup.
2. Enter the url (either localhost:5000 or the deployed web app https://fit3170-49455.web.app/) and the debug view will open.





3. Navigate to the app and interact with the YouTube player. On the left hand of the debug view page you should see that the tag has fired. (At the time of writing, we were unable to get the preview mode working)



Images taken from
https://www.analyticsmania.com/post/track-videos-with-google-analytics-4-and-google-tag-manager/

4. For publishing the container, return to the workspace page and click ‘Submit’ to deploy the setup.

For more information, visit the following links:

- https://www.analyticsmania.com/post/how-to-install-google-tag-manager/
- https://www.analyticsmania.com/post/track-videos-with-google-analytics-4-and-google-tag-manager/
- https://developers.google.com/tag-manager/quickstart
- https://www.analyticsmania.com/post/google-tag-manager-preview-mode-not-working/

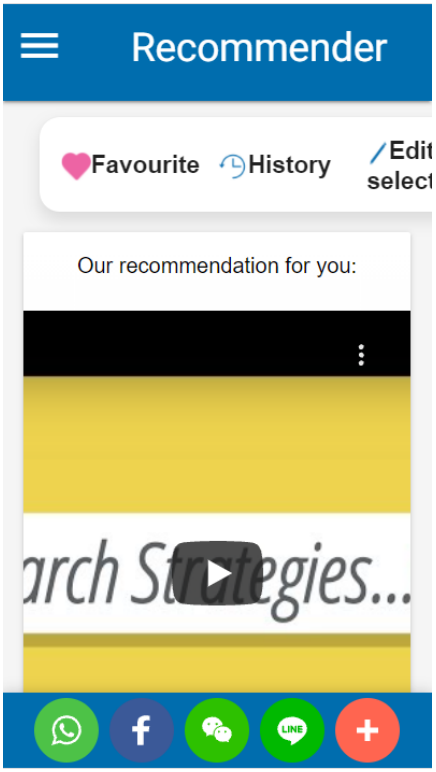
Seeing the tracking in action

1. Visit the recommender module, select a learning interest skill and play a video. Perform interactions such as seeking or pausing with the player too.
2. Go to the project’s Firebase Realtime Database and select users > {phone number} > videoHistory > videoAnalytics
3. If the video is still playing, you may be able to see the data being updated in real-time.

5.2 Plans for Production Systems

5.2.1 UI Issues

1. Non-responsiveness of page on some mobile screens



Similar to the Forum submodule (4.2.1), the recommender pages are not able to completely fit into some mobile screen sizes. They were not taken into consideration and thus the UI does not fit them properly. This issue needs to be resolved by ensuring that the application is responsive for as many screen sizes as possible in order to expand the potential users range.

2. Small icons/elements

Depending on the screen size, some icons might be too small for the elderly women to see properly. Adjust the sizes accordingly when needed.

5.2.2 Video player playback

The video player page has its own custom playback buttons to allow users to explore the videos however they like.

There is a bug where if the user has liked/disliked and/or added a video to favourites, when the user goes to the next or previous video the button icons will still be coloured. Also, the next or previous video will autoplay. The video should not autoplay because when this happens the system will not record the user's watch history and analytics for that video.

The quick fix we have implemented is to reload the whole page itself every time the user clicks/taps on the playback buttons so that the video would not autoplay and the icons would not be filled in. Our current code involves checking the database if the user has already liked/disliked/favourited the video and filling in the icons if the conditions are met. There may be a better way to handle this bug - one way would be to find a way to perform these tasks asynchronously by adjusting the YouTube player's settings, or improving on the existing checking code.

5.2.3 Time interval for saving video analytics data to Firebase

Currently, the video player page retrieves data from the GTM dataLayer variable and saves the video metrics every 10s to the database. This was done so that the database would not be filled up with a lot of data in a short amount of time. Even so, it is an issue when the user leaves the page before their last interaction with the video can be saved.

5.2.4 Admin dashboard

1. Users, Favourites, Skill pages

   Each page currently needs to be refreshed for the data to be updated. The pages should be updated in real-time if possible.

2. CSV export function in the admin dashboard page under the Recommender tab

   The function is used to export all the data related to the recommender from the Firebase realtime database. How it is implemented is similar to the forum submodule (4.2.5).

   Once the button is clicked, it generates 5 csv files, which are general_user_data (contains the id, username and phone number), recommender_user_watch_history (contains data about the videos users have watched, liked/disliked), recommender_user_video_analytics (contains the

latest watch data for each video the user watched), recommender_user_favourited_videos (contains each user's favourited videos) and recommender_skill_selected_frequency (contains data about how many times a user has selected a skill or favourited a video related to that skill).

It has similar issues in terms of the database structure as noted in 4.2.5. There is also some nested data present under the 'users' node, namely the 'videoHistory' node. Each user will have a video history with a default total watch count of 0 - this indicates that the user landed on the video but did not click on the video to play it. If the video is played, video analytics for the current date and time is recorded. Thus, the structure is as follows: /users/phone/videoHistory/index/videoAnalytics/date/timestamp. If by any chance the structure of the csv needs to be changed, do take note of how the data is stored and make changes when needed.