Handover Documents

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

1. Introduction

This document outlines the third party dependencies that are required to start using the application, along with steps to set it up. These will be discussed in the sections below. Furthermore, the data structure that we have used in Firebase will be clarified to allow new developers familiarise with the data structure being applied. Additionally, a versioning strategy to be used for future releases will be discussed appropriately.

It is worth noting that this document is targeted towards developers and not users. For a general overview of the application, refer to the user guide.

2. Quick Startup Guide

The following instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

2.1 Prerequisites

Prior to the project, you will need to install npm locally and the latest version of Node.js on your computer.

- *npm install*

For any clarification, please view the provided link given below:

https://github.com/npm/cli

2.2 Clone and Installation

Clone the following repository

git clone https://git.infotech.monash.edu/fit3077-empowering-women/fit3170-empowering-women

- If you have trouble logging in after successful cloning of repo, you will need to create your own personal access token which can be created in Access Tokens under User Settings of your Monash Git Profile.
    - Do ensure the following options are selected
        - Enter a sensible name for easy tracking of the project's access token
        - It is preferable to add the expiry date of the token so that it is automatically deletes once you are done working with the project
        - Select "api" under Scopes, which enables you to read and write access to the API

Install Firebase CLI and login into your Google account

In order to be able to host your project locally, you are required to install Firebase CLI on your device. By using Firebase CLI, you can run the local development server of the project, and deploy content. The installation steps are as follows:

- npm install -g firebase-tools
- firebase login

For any clarification, please view the provided link given below:
https://github.com/firebase/quickstart-js/blob/master/database/README.md

Initialise Project in your Computer and Test Locally

The project link in Firebase Realtime Database is provided as below:
https://console.firebase.google.com/u/2/project/fit3170-49455/settings/general/web:MjhkNmFiZWEtODRhZi00MmYxLTg1MWQtZjY3MGM3MDQwM2Jh

To initialize the project from your command line

- firebase use --add (*project directory*)

To test out the project and start the application

- firebase serve
    - Once your server has started, go to this url http://localhost:5000/ and you will now see the application running on the Development Server.

Please ensure that all developers in the team have privileges to access this project and select the correct project name in the command line.

For any clarification, you can find the following video link to be helpful for you to initialise and test out the project:

- https://www.youtube.com/watch?v=qbxj_36UkWs

Deploy Project

Alternatively, to deploy the application, run the following command:

- firebase deploy

3. Key Features

Below are the main features for each submodules:

3.1 Chatbot

3.1.1 Questions and Responses

Currently, questions and responses are stored in the FIrebase. Their details will be outlined in the Firebase section below. All question ids are auto generated, they are not based on question numbers. Such IDs will then be pasted to the file constants.js found in the path public/script/constants.js. They are pasted when questions are uploaded to the cloud.

3.1.2 Translation

In the current implementation, questions and terms of use instructions, along with their translated variants, use hardcoded strings stored in Firebase. However, Google Translate is used for the button descriptions and is not ideal as they occupy screen real-estate. It is suggested that future developers use hard coded strings for these as well.

3.1.3 Response storage and question uploader

All handling of response storage logic is located in the response-storage.js file located under path public/chatbot/script.

Question uploader logic is located in the various js files under public/chatbot/script/question-uploader, which consists of a separate uploader for different languages.

To re-upload questions with a new set of auto-generated IDs, The following steps shall be performed:

1. Link the question-uploader javascript file of the desired language in chatbot.html under filepath public/. For example, for uploading of English questions, use
<script src="chatbot/script/question-uploader/question-uploader_en.js"></script>,
For uploading of Chinese questions, use
<script src="chatbot/script/question-uploader/question-uploader_zh_CN.js"></script>

2. Call uploadQuestions() in the terminal, this would upload the questions to Firebase. It will also output a lot of logs in the console.

3. Make sure uploadQuestions() has finished running. No new logs being printed in the console indicates this.

4. Call housekeeping() in the terminal.

5. Copy the IDs being displayed in the console log. And paste them inside constants.js, under the appropriate variable. Eg: for english questions, paste them to the variable QUESTION_IDS_EN, for malay, QUESTION_IDS_MS, etc. - This would link the correct question IDs to the chatbot.

6. Your new set of questions will then be correctly referenced by the application.

3.1.4 User Interaction Logic

User interaction logic is found in the first_time_survey.js file located under filepath public/script/

3.2 Web-based Recommender System

3.2.1 Learning Interests / Skills selection functionality

The logic for this functionality can be found under the filepath:

- *public/script/recommender_select_skills.js*

Selection of a skill is recorded for the user and the total number of times it has been cumulatively selected by users is updated to the Firebase Realtime Database. The data details will be covered in the later sections.

3.2.2 Video player functionality

The logic for this functionality can be found under the filepath:

- *public/script/recommender_main.js*

This functionality controls the video player and the content being shown on the recommender_Ui.html page. It loads the YouTube player asynchronously and fetches videos stored in the Firebase Realtime Database that are categorised based on the current skill selected by the user in the learning interests page. Each video stored also has a unique ID used to identify them. More details are in the later sections below.

3.2.4 Tracking video analytics functionality

The script and logic for this functionality can be found under the filepath:

- *public/script/recommender_main.js*
- *public/recommender.Ui.html*

This functionality implements Google Tag Manager (GTM). The script to enable GTM can be found at the opening of the <head> tag. In recommender_main.js, it makes use of the dataLayer variable from GTM to retrieve the video analytics that have been tracked. The relevant data is then stored in the Firebase Realtime Database.

For a tutorial on Google Tag Manager (alongside Google Analytics), a suggested resource we found useful:

- 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/

3.2.5 Add video to favourite functionality

The logic for this functionality can be found under the filepath:

- *public/script/recommender_main.js*

A video is stored into a user's favourites and the database will be updated accordingly with the relevant information about the video. It will also increment the

number of times the associated skill has been added to all users' favourites in the database.

3.2.6 View/delete favourite videos functionality

The logic for this functionality can be found under the filepath:

- *public/script/favpage.js*

Videos that have been added to the users' favourites are retrieved from the database to be rendered and displayed. Users can filter videos according to skill to show favourited videos under that skill. Viewing a video will redirect them back to the video player from 3.2.1. Videos can be deleted all at once or one at a time.

3.2.7 View/delete watch history functionality

The logic for this functionality can be found under the filepath:

- *public/script/recommender_history.js*

Videos that have been watched are retrieved from the database to be rendered and displayed. Viewing a video will redirect them back to the video player from 3.2.1. Videos can be deleted all at once or one at a time.

3.2.8 Likes and dislikes functionality

The logic for this functionality can be found under the filepath:

- *public/script/recommender_main.js*

When a user likes or dislikes a video, it will be saved as a true or false value under the video history. More details will be provided in the upcoming sections.

3.3 Web-based Forum

3.3.1 Post Creation Functionality

All the logic required for the post creation functionality can be found under:

- *public/script/forum.js*

For every new post, it would be assigned a unique ID which is auto-generated. Each post ID holds all the information a user has inputted and will be stored in Firebase Realtime Database. Such IDs will constantly be used in all the order functionalities that will be discussed more below. The data details for a post will be discussed more in the upcoming sections.

3.3.2 Likes and Dislikes Functionality

All the logic related to the likes and dislikes functionality are located under the following files:

- *public/script/post_detail.js*
- *public/script/forum.js*
- *public/script/forum_likes_dislikes.js*

Whenever a user likes or dislikes a post or comment or reply, their unique ID is used and saved under the related post indicating they have liked or disliked the post/comment/reply. It also comes with an alert function which mainly alerts users if the system has found that they have clicked on the button way too fast. How the data is stored will be discussed further below.

3.3.3 Comment and Replies Functionality

The codes related to the comments and replies functionality can be found under the path:

- *public/script/post_detail.js*.

Similar to the creation of a post, each comment or reply is assigned a unique ID which is also auto-generated. The details for every comment and reply will be discussed more in the upcoming sections.

As of the current implementation, the page will refresh every time a new comment or reply is being added to a post. It is suggested that the overall functionality will be improved further on to improve the effectiveness and efficiency of the application.

3.3.4 Favourite Functionality

This functionality allows users to add a post into their favorites. All the logic behind this functionality can easily be found in:

- *public/script/post_detail.js*

similarly with the comment and reply logic.

In terms of the backend work, the unique user ID will be saved under fit3170-49455-default-rtdb/posts/user_favourite where posts would represent the unique post ID that they have favourited.

As of now, the favourite functionality is available when a user views the details of a post and we wish that future developers would make improvements in terms of adding the functionality when users are casually scrolling through the list of posts.

3.3.5 Delete Post Functionality

This functionality is provided when a user views their own personal post that was previously created by them. It would not be provided when a user views another user's post. The logic behind the functionality can be found in:

- *public/script/post_detail.js*

Do note that when a post is being deleted, all comments, replies, likes and dislikes associated with the post will also be removed as of the current implementation.

3.3.6 Search Functionality

The search functionality is only provided in the main page of the forum and all logic behind it can be found under:

- *public/script/forum.js*

The functionality also comes with an autocomplete feature which would display the top 10 options similar to the input made by the user. As of now, the functionality is case-sensitive and will successfully find the wanted post by writing the title in the right order so we wish future developers could find ways to improve on the following functionality.

3.4 Back-end Admin Dashboard

As of now, it is only accessible when the directory is specified in the browser, no href links are used to point to them. All files related to the admin dashboard are available in “dashboard”, under the path public/dashboard/

3.4.1 Chatbot

In the chatbot dashboard, administrators can view responses to a particular question, along with its translated variants. A CSV file can also be downloaded if admins wish to do further data analysis.

3.4.2 Forum

In the forum admin dashboard, as of now the dashboard displays 3 main submodules statistics for: Post,Interest, User. Each will show a summary of all the different types of interactions that are allowed in the forum.

In the posts dashboard, the admin can view the number of posts made by all the users, the number of posts made by the recommender system, the details of a specific post (id, title, content, etc.) and the interactions with that specific post (i.e. number of likes, number of comments, etc.).

In the users dashboard, the admin can view the total number of users that are currently using the application, the interaction of a specific user with the forum (i.e. summary of posts they created, summary of posts they liked and disliked, summary of posts that they added to their favorites, etc.), statistics that display the interests they interacted with the most, the number of posts they created, the number of comments and replies they made, the number of likes and dislikes they made, etc.

In the interest dashboard, the admin can view a summary of the total number of interactions made of every single interest as a bar chart as well as a table summary of all the posts that were made under specific interest.

A CSV file can also be downloaded if admins wish to do further data analysis.

3.4.3 Recommender

In the recommender admin dashboard, there are currently 3 main submodules that show the statistics for User (video analytics), Favourites and Skill. Each page shows an overview of interactions and activity that have occurred in the recommender.

The User (video analytics) dashboard initially only shows a list of users currently using the application in a table. For each row of users there is a ‘view’ button that when clicked will show a bar chart for video history interest distribution (the number of videos in all users’ watch history categorised by interest) and another bar chart for favourited videos interest distribution (the number of videos in all users’ favourites categorised by interest). At the bottom of the page is a table displaying a list of videos the user has watched. If there are any, each video entry will have a ‘view’ button that when clicked will show a table of the latest video analytics captured. The video analytics table will show if the video has at least a watch count of 1.

The Favourites dashboard shows the number of users that have added a video of a particular skill in a bar chart. The admin can use the dropdown menu to select the category (learning interest) they want to view the chart for.

The Skill dashboard shows the total number of times users have selected a skill from a particular learning interest. The admin can use the dropdown menu to select the category (learning interest) they want to view the chart for. At the bottom of the page shows the total number of video likes and dislikes for each skill under that learning interest.

A CSV file can also be downloaded if admins wish to do further data analysis.

4. Summary of Information

This section touches on a brief summary of all the major frameworks/libraries used throughout the project.

4.1 JavaScript

Javascript is a high-level language that is used for mobile and web application development. The reasoning behind the choice of JS as our main language is due to the experience of developers. Moreover, as we are using Firebase, we found that JS is compatible with it and provides virtually endless capabilities for web programming.

4.2 JQuery

JQuery is a JavaScript library that was used in order to simplify HTML element manipulation, as well as event handling and CSS animation. Besides making the website noticeably faster, JQuery provided numerous plugins that were used in the development of the application

4.3 Bootstrap

Bootstrap is an open-source CSS framework that has assisted us in improving the UI of the application since it contains various CSS- and JS-based design templates for typography, forms, buttons, etc. For instance, the font used was imported from Bootstrap.

4.4 MDL

MDL is a library, which follows Material Design concepts. Most of the components in the Forum, including cards, buttons, text fields, and other interface components tightly rely on MDL, which makes the website more portable and usable. MDL was chosen over other libraries, because of its familiar appearance, since it is used by Google.

5. Required Third-Party Software

5.1 Firebase

5.1.1 Chatbot

The chatbot section of the application uses the Firestore database for storage of questions, responses and user data. There are four branches that are being used by the Chatbot. Namely

1. TermsOfUse

This branch consists of the terms of use for the chatbot. It is stored in a field called “contents” and contains the html text, along with formatting identifiers. There are multiple variants of sub-branches that exist within this branch, there are all translated variants of the default terms of use English text. Currently there is: TermsOfUse_en, TermsOfUse_ms, TermsOfUse_th, TermsOfUse_zn_CN, where en is english, ms malay, th thai, zn_CN, mandarin.

{ contents: }	
Attributes	**Meaning**
contents	String: Html text + formatting identifiers for this specific version of the terms of use

2. chatbot/survey_questions

This branch consists of all of the chatbot’s questions, including translated variants. The questions are stored in their language’s sub-branch. Please note that

the IDs that are used for each question are auto-generated. This is to allow for ease of uploading questions, without having the need to delete question ids frequently. The IDs are referenced inside the constants.js file, under directory public/script/constants.js

{
category:
hint:
longQuestionId:
question:
question_number:
restrictions:
type:
}	

Attributes	**Meaning**
Category	String: Category of questions,with reference to the original word document from clients
hint	String: Hint for users to answer question
longQuestionId	String: This attribute only applies to sub-questions; Question ID of the parent question
question	String: Text of the question, along with HTML formatting identifiers
question_number	String: Question Number
restrictions	Map: Contains a list of attributes that corresponds to the question's restrictions; these include MCQ options, numerical upper limit and lower limit, etc.
type	String: Question type; can be multiple-choice, numeric, multiple-choice-sub-question, long-text

3. chatbot/survey_responses

This branch consists of all the chatbot's responses. In this branch, all translated variants are stored under chatbot/survey_responses and not further categorised into sub-branches.

Within this branch, the ids used correspond to the question_id, where it stores all responses to a specific question_id. Please take note that each user can only record one set of responses in a question. Additional responses when users restart the

survey will delete all their previous responses

{
Answer:
Phone:
timestamp:
}	

Attributes	**Meaning**
answer	Number: Answer as responded by users
phone	String: Phone number of user
timestamp	Timestamp: Time in which response was submitted

4. users

This branch stores the user's progress of the survey. For example, the question number that they have reached. It also stores some additional information, clarified below.

{
currentSubQuestionIds:
questionIndex:
subQuestionIndex:
closedSurvey:
skippedToEnd:
joinFutureResearch:
}	

Attributes	**Meaning**
currentSubQuestionIds	Array: a set of Ids that corresponds to a long question. For example, a user is in the process of answering a long question and exited the survey, this attribute is to conveniently retrieve the question_ids of the long question. This is necessary as subquestion IDs are not stored in constants.js - Its question_ids are stored in Firebase, under its long_question ID's "arrangement" field.
questionIndex	Number: Index of question that the user has most recently answered

subQuestionIndex	Number: Index of subquestion that user has most recently answered; only applies to long questions.
closedSurvey	Boolean: Whether a user has closed their survey
skippedToEnd	Boolean: Whether a user is not our target-audience, Survey is skipped to the closing message if this attribute is set to True.
joinFutureResearch	Boolean: Whether a user is willing to participate in plans for future research

5.1.2 Forum and forum dashboard

The forum section of the application uses the Realtime database in order to store information regarding the users and the interactions they have with the forum. The Realtime database stores data as one large JSON tree that consists of smaller JSON objects. In our case, each child of the root of the tree represents objects that store smaller JSON objects of the same type.

For the forum, there are multiple smaller JSON objects that are used to represent the data we were required to store. These objects are:

1. fit3170-49455-default-rtdb/users

Child objects under users store data about the users that are currently using the application. The unique key used to differentiate each user is the phone number of the user. Each JSON object stored under this section stored the mobile number of a user and their username. It was represented like:

{
phone:
username:
}	

Attribute	**Meaning**
phone	String: User's phone number
username	String: User's username

2. fit3170-49455-default-rtdb/posts

Child objects under posts store data that contains all the posts available in the forum regardless of whether they were made by the users themselves or whether

it was added from the recommender system. A random unique key is used to differentiate between each post. Each JSON object stored under this section has the following format:

{
anonymous:
created:
description:
dislikes:
id:
interest:
likes:
recommender:
title:
username:
users_faviourte:
videoURL:
}	

Attributes	**Meaning**
anonymous	Number: the number of anonymous comments and replies present under the post
created	String: the date and time the post was created
description	String: description of the post
dislikes	Number: the number of dislikes on the post
id	String: the unique id that identifies the post
interest	Array: maximum length of 2. The elements represents interest that the post belong to
likes	Number: the number of likes on the post
recommender	Boolean: True - if the post was made by the recommender system False- if the post was made by the user
title	String: the title of the post
username	String: the username of the user who made the post
users_faviourte	Array: The elements represent the number of the user who has added the post to their favourite posts.
videoURL	0 - if the user have not inputted a video url to include in their post String - if the user included a video url to include in their

	post. The string is the video url

3. fit3170-49455-default-rtdb/comments

Child objects under comments store data that contains all the comments available in the forum. A random unique key is used to differentiate between each comment. Each JSON object stored under this section has the following format:

{
anonymous:
commenterID:
content:
created:
id:
likes:
postID:
username:
}	

Attributes	**Meaning**
anonymous	0 - if the comment is not anonymous > 1: If the comment is anonymous. Where the number represents the display of the username. For instance, if the number is 1 then the comment will be presented as made by “Anonymous 1”.
commenterID	String: the unique Id used to identify the user who made the post (i.e. their phone number)
content	String: contains the content of the comment
created	String: the date and time the comment was made
id	String: the unique id that identifies the comment
likes	Number: the number of likes on the comment
postID	String: the unique Id used to identify the post this comment belongs under.
username	String: the username of the user who made the comment.

4. fit3170-49455-default-rtdb/replies

Child objects under replies store data that contains all the comments available in the forum. A random unique key is used to differentiate between each reply. Each JSON object stored under this section has the following format:

{
anonymous:
replierId:
content:
created:
id:
like:
dislike:
reply_comment_parent:
username:
}	

Attributes	**Meaning**
anonymous	0 - if the reply is not anonymous > 1: If the comment is anonymous. Where the number represents the display of the username. For instance, if the number is 1 then the reply will be presented as made by “Anonymous 1”.
replierId	String: the unique Id used to identify the user who made the post (i.e. their phone number)
content	String: contains the content of the reply
created	String: the date and time the reply was made
id	String: the unique id that identifies the reply
like	Number: the number of likes on the reply
dislike	Number: the number of dislikes on the reply
reply_comment_parent	String: the unique Id used to identify the parent reply/comment this reply belongs under.
username	String: the username of the user who made the reply.

5. fit3170-49455-default-rtdb/likesDislikes

Child objects under likesDislikes store data that contains all the likes and dislikes made by the users on all the posts available in the forum. The key used is the post ID of the post which the like/dislike belong to. Under each post ID a child JSON object exists with the keys being the username of the users that liked/disliked the post. Each JSON object stored under the username has the following format:

{
action:
}	

Attributes	Meaning
action	1- if the user liked the post -1 - if the user disliked the post

6. fit3170-49455-default-rtdb/likesComments

Child objects under likesComments store data that contains all the likes made by the users on all the comments available in the forum. The key used is the comment ID of the comment which the like belongs to. Under each comment ID a child JSON object exists with the keys being the username of the users that liked the comment. Each JSON object stored under the username has the following format:

{
action:
}	

Attributes	**Meaning**
action	1- if the user liked the comment

5.1.3 Recommender

Same as with 5.1.2, the recommender uses the Realtime Database to store information regarding the users and the interactions they have made with the recommender. The recommender also uses the same objects fit3170-49455-default-rtdb/users and fit3170-49455-default-rtdb/posts.

1. fit3170-49455-default-rtdb/users

Child objects under users store data about the users that are currently using the application. The unique key used to differentiate each user is the phone number of the user. Each JSON object stored under this section stored the mobile number of a user and their username.

In the case of the recommender, the child objects used are preferences, videoFavourite and videoHistory.

{ phone: preferences: username: videoFavourite:	

videoHistory: }	
Attributes	**Meaning**
phone	String: User's phone number
preferences	Array: Maximum length of 1 - stores the current skill selected from the learning interests page
username	String: User's username
videoFavourite	Array: Stores information about the video that the user has added to favourites. Each video is accessed using their index in the array. More details are covered under fit3170-49455-default-rtdb/users/videoFavourite/index
videoHistory	Array: Stores information and analytics about the video that the user has watched. Each video is accessed using their index in the array. More details are covered under fit3170-49455-default-rtdb/users/videoHistory/index

2. fit3170-49455-default-rtdb/users/videoFavourite/{index}
   Below are the child objects under each index in the videoFavourite attribute from 1.

{
interest:
postId:
videoThumbnail:
videoTitle:
videoUrl:
}	

Attributes	**Meaning**
interest	String: The learning interest skill associated with the video
postId	String: The id used to identify the video under fit3170-49455-default-rtdb/posts
videoThumbnail:	String: A URL link that contains the thumbnail used for

	the video
videoTitle	String: The name of the video
videoUrl	String: The YouTube link to the video in embed format

3. fit3170-49455-default-rtdb/users/videoHistory/{index}

Below are the child objects under each index in the videoHistory attribute from

{
dislike:
interest:
like:
postId:
totalWatchCount:
videoAnalytics:
videoThumbnail:
videoTitle:
videoUrl:
}	

Attributes	**Meaning**
dislike	Boolean: Default value is false - if the user clicks on the dislike button for a video value is set to true. It is changed to false if like is true
interest	String: The learning interest skill associated with the video
like	Boolean: Default value is false - if the user clicks on the like button for a video value is set to true. It is changed to false if dislike is true
postId	String: The id used to identify the video under fit3170-49455-default-rtdb/posts
totalWatchCount	Number: The number of times the user has played the video and watched for at least 10 seconds. Default value is 0 which indicates that the user has landed on the video but did not play it
videoAnalytics	Array: Stores the analytics for a video on a particular date. Entries are accessed using the date the video is watched on using the format ‘yyyy-mm-dd’. Once accessed, each video data is accessed by timestamp in

	the format ‘hh:mm:ss’. More details are covered in fit3170-49455-default-rtdb/users/videoHistory/videoAnalytics/date/timestamp
videoThumbnail:	String: A URL link that contains the thumbnail used for the video
videoTitle	String: The name of the video
videoUrl	String: The YouTube link to the video in embed format

4. fit3170-49455-default-rtdb/users/videoHistory/videoAnalytics/{date}/{timestamp}

Child objects stored here measure the time and status of a video being watched by the user.

{
videoCurrentTime:
videoDuration:
videoElapsedTime:
videoPercent:
videoStatus:
videoVisible:
}	

Attributes	**Meaning**
videoCurrentTime	Number: The current timestamp of the video where the viewer is at (in seconds)
videoDuration	Number: The duration of the video in seconds
videoElapsedTime	Number: Actual watch time of the user in seconds
videoPercent	Number: The threshold of the video (without the % sign)
videoStatus	String: The current state of the video - can be one of the following start - when the user plays the video the first time progress - the video is continuously being played paused - the user has stopped the video seek - part of the video has been skipped forward or

	back by the user complete - the video has finished playing
videoVisible	Boolean: Value is true if the player is visible on the screen while the video engagement was tracked and false otherwise

5. fit3170-49455-default-rtdb/recommenderData

Below are the child objects stored for recommenderData. This is mainly used for the admin dashboard.

{
favourites:
skills:
}	

Attributes	**Meaning**
favourites	Array: Stores a list containing the number of times a video associated with a skill has been added to a user's favourites. Each entry is accessed using a unique ID. More details are covered in fit3170-49455-default-rtdb/recommenderData/favourites
skills	Array: Stores a list of learning interest skills. Each entry is accessed by using the name of the skill. More details are covered in fit3170-49455-default-rtdb/recommenderData/skills

6. fit3170-49455-default-rtdb/recommenderData/favourites/{id}

{
favouriteAmount:
favouritedUsers:
preferenceType:
}	

Attributes	Meaning
favouriteAmount	The number of times a video associated with a skill has been added to a user's favourites
favouritedUsers	Array: List of users who have at least one video associated with the skill in their favourites
preferenceType	String: The name of the learning interest skill

7. fit3170-49455-default-rtdb/recommenderData/skills/{skill name}

{
favouriteAmount:
selectedAmount:
skill:
}	

Attributes	**Meaning**
favouriteAmount	Number: The number of times a video associated with a skill has been added to a user's favourites
selectedSkill	Number: The number of times a user has chose this skill on the learning interests page
skill	String: The name of the learning interest skill

5.2 Google Translate

Google Translate is embedded into our whole application, which translates the default English text to the current user's selected language. If developers wish to prevent a certain element from being translated, they can include a "notranslate" tag. The scripts for google translate are already included in the html files.

5.3 Authentication

The system utilizes Firebase's phone authentication system when a user is performing a sign up. The firebase authentication system comes with firebase tools installation. Due to the client's requirements, we have removed the firebase's phone

authentication from the login as they did not wish to include a pin code whenever a user attempts to login.

When a user attempts to login, a user has to input a phone number that is already registered in the realtime database. Then, recaptcha is enabled and the user has to follow the procedure. The login information including username and phone number is stored in the local storage with the key name “USER”. Then, it will be redirected to the terms of use page after successful login.

If a user enters a phone number that does not exist in the realtime database, then we redirect to the signup page after the user agrees with the terms and conditions in the terms of use page. In the signup page, a user has to enter an username and phone number. It checks if the username and phone number are valid or not. For example, if the phone number includes any character, then it shows that the user has to enter a valid phone number. After the user succeeds in entering the phone number and username without facing any error message then the user is asked to click the “Send Pin” button that sends a 6-digut pin number to the entered phone number. Once the user successfully enters the pin code received from the phone number, then the recaptcha is prompted to select a number of pictures asked. Finally, users’ information including username and phone number is stored in the local storage and realtime database.

If developers wish to strengthen the security for login, then they could include a password section to include password made upon registration. Also, if developers wish to include logout functionality, they could include a logout functionality that deletes the information stored in the local storage. Moreover, if developers wish to restrict pages that an unauthorized user can go to, then they could add code that checks if user information exists in the local storage because currently an unauthorized user can go to any pages they wish to go to.

6. Pull Request Strategy

When developing the application, it is likely that multiple branches would be stemmed out from the master branch, for ease of development. As such, a pull request strategy is needed when features of a sub-branch have been completed, and is ready to be merged back into the master branch. There are a few strategies that are viable for this. Animations of the strategies discussed below can be found in: https://techmunching.com/git-merge-vs-git-rebase/

6.1 Explicit Merges (Non fast forward)

This is a straightforward method and is the default merge type. This method creates a merge commit that records the event of two branches merging together. The merge commit would then have two parents - The two individual branches that were merged together.

6.1.1 Advantages

It preserves history of commits

6.1.2 Disadvantages

Some teams would prefer to avoid using explicit merges as it creates clutter in the linear history of commits (the additional merge commit), or even referred to as “noise”.

6.2 Implicit Merges (rebase or fast forward)

6.2.1 Rebase

This strategy will precisely replay the sub-branches’ commit one-by-one on top of the master branch. This way, an additional merge commit would not be needed.

6.2.2 Fast Forward

Similar to the rebase strategy, a fast-forward merge replay the sub-branch’s commit on top of the master branch one-by-one, but by moving the master’s HEAD towards the latest commit of the sub-branch. It has one caveat though,

which is that the master must not have additional recent commits than the commits of the sub-branch.

6.2.3 Advantages

No extra merge commit is made so commit history is clean

6.2.4 Disadvantages

Some context of the commits made before the merge would be lost

6.3 Squash Merge

This strategy involves squashing all of the commits of the sub-branch into a single commit before performing an implicit merge through rebase or fast-forward, as discussed earlier.

6.3.1 Advantages

It keeps the branch history linear and clean, as it isolates a feature into a single commit.

6.3.2 Disadvantages

By squashing features into a single commit, sub-branch development history would be lost; Insights and details on how a feature was developed will be vanished.

6.4 Conclusion

After evaluating the various strategies listed above, it is suggested that future developers use the explicit merge strategy, as outlined above, this is because it provides good traceability of commits and if something were to go wrong, it can be easily reverted.

7. Versioning Strategy

V1.0.0 - 2021.10

Added Features

Chatbot - Admin Dashboard Chatbot Home Page - Chatbot page - User input validation - Error message display - Hint display - Text-to-speech feature - Data export to csv file feature
Recommender System - Admin Dashboard Recommender Home Page - Admin Dashboard Recommender User Page - Admin Dashboard Recommender Favourite Page - Admin Dashboard Recommender Skill Page - Learning Interests Page - Video Player Page - Favourite Videos Page - Watched History Page - Video Analytics Feature - CSV Export Feature
Forum - Admin Dashboard Forum Home Page - Admin Dashboard Login Page - Admin Dashboard Forum User Page - Admin Dashboard Forum Interest Page - Admin Dashboard Forum Post Page - Forum Page - Main Page - Post Detail Page - Forum Login Page - Admin Dashboard Login Page - Translation Feature - CSV Export Feature

V1.1.0 - 2022.2

Proposed Changed Features

Chatbot - Improve responsiveness of chatbot and load up speed for other languages - Polish text-to-speech to have an accent matching language used - Hidden bug fixes
Recommender System - Improve placement of buttons on video player page so that every element can fit on the mobile screen without scrolling - Improve responsiveness for different mobile screen sizes
Forum - Polish on the layout for ‘New Forum Post’ section and time representation for a post - Improve on making screen more responsive for different screen size - Improve the functionality of Search function - Include alerts on any important user actions - Improve on the structure of the database for replies data

V1.2.0 - 2022.5

Proposed Changed Features

Chatbot - Implement speech-to-text
Recommender System - Improve on video analytics functionality - Rename attributes and improve structure/consistency under fit3170-49455-default-rtdb/recommenderData   - Remove either favourites or skill
Forum - Polish the layout under Feed tab to be able to differentiate between user favourite post and personal post - Improve on the likes and dislikes functionality - Improve on the reply functionality