Design and Implementation of LBW – A Mental Health Application for Children

By

Sexi Wu

Thesis
Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

in

Computer Science

December 18, 2021

Nashville, Tennessee

Approved:

Douglas C. Schmidt, Ph. D.

Christopher J. White, Ph. D.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

As an infectious disease spreads quickly to every corner of the world in 2020, some countries and territories around the world have enforced lockdowns of varying levels to prevent and slow down the transmission of this disease. It is caused by a newly discovered coronavirus (COVID-19). Due to the threat of COVID-19, only essential businesses are allowed to remain open, and most workers must work from home. Schools, universities, and colleges have closed in 63 countries, which affects approximately 47 per cent of the world's student population.[1] As a result of mandatory quarantine, people need to go online and use social applications to maintain connections between each other. Despite there being dozens of social applications in Google or Apple market, few of them are suitable for children to use. Therefore, it is important to identify children's needs for special UI designs, functions, and contents, and design a kid-centric application aiming to help them deal with pandemic challenges.

This thesis will explore the whole process of developing Love In A Big World app, a content-driven mental health mentor application for children. The practical output of this thesis will be a functional mobile application that applies machine-learning and deep learning-based natural language processing (NLP) methods.

This thesis is divided into eight chapters. In Chapter 2, the motivation is present. Chapter 3 describes the analysis of functional and non-functional requirements. Chapter 4 provide the background information of technologies and libraries applied in the implementation. Chapter 5 gives an overview of the system structure to help understand how different frameworks

combined in this project. In Chapter 6, the implementation of the whole application is described, where the core NLP models are discussed. In Chapter 7 and Chapter 8, future improvement plans and conclusions are listed separately.

Chapter 2

Motivation

The motivation for this thesis comes from Love In A Big World (LBW) Team who provides educational content for kids in Nashville. The goal of LBW is to connect with kids, to let them know that they are not alone in this big world, to teach them how to make wise choices, and to give them hope. Because of COVID-19, LBW team decided to make an online platform for kids to obtain personalized mental health support. This platform aims to provide affordable, scalable, and suitable mental health services for middle childhood (ages 8 - 14), their families, and their educators during and post the pandemic.

The ultimate goal of this project is to build a functional web application based on LBW education, professional development, student assemblies, and family resources. A key differentiator of LBW web application over the competition is that it is personalized by applying machine learning models. By using our model to analyze users' search inputs and match it with our stored video contents, it will be possible to recommend related videos and show tips on it. Besides, kids can talk with our polite and interesting chatbot to cheer them up or get tips on their questions.

There are a number of challenges involved in this project. Firstly, we need to start from scratch. Since web applications are rarely designed to be used mainly by kids and few of them are for mental health care, we need special UI designs to ingratiate middle childhood users. Secondly, it is hard to match kid's queries with our videos, because most of them are not able to conclude their issues in a few words. They usually either ask some simple questions like "How can I fall

asleep?" or input some issues like "I'm always tired." Therefore, it is essential to calculate sentence similarities of client-side queries and video contents. Thirdly, compared with regular chatbots, ours that are designed for kids need some modifications like preventing toxic contents from kids and behaving smart to help solve problems.

In conclusion, the aim is to implement a mental health web application called LBW App. LBW App will not only provide a platform for kids to enjoy recommended videos that help solve personal trouble during and post the pandemic, but also a virtual mental health mentor to find answers to life worries. What's more, it applies strategies to strengthen the connection between kids, their families, their friends, and their teachers.

Chapter 3

Requirement Analysis

In this section, requirement analysis is described, which involves analysis of both the functional and nonfunctional requirements.

3.1 Functional Requirements

Functional requirements define the system behavior and usually include calculations, data inputs, and business processes. There are three authorized levels of users in our system: unregistered users, normal registered users, and administrators. Since a normal registered user can do every task that an unregistered user is able to do, we start from unregistered users and climb up level by level without repeating the same task that can be done by a lower level of users. Compared with these two levels, administrators have special requirements that can only be done by this authorized level.

3.1.1 Unregistered Users' Use Case Analysis

3.1.1.1 Registration

- Requirements: Allow a new user to Register in the system.
- Preconditions: User does not exist in the system.
- Postconditions: User has a record in the database.
- Scenarios:

1. User requests his/her registration in the system by providing email, first name, last name, and password. One email is unique in the database.

3.1.1.2 Video Search

- Requirements: Allow users to search videos by words or sentences. The search results are obtained by calculating the similarity of inputs and comparing it with those of video transcripts in our database through our machine learning backend.
- Preconditions: None.
- Postconditions: None.
- Scenarios:

1. Retrieve the video list from the database according to the search inputs.

3.1.1.3 Select Videos to watch

- Requirements: Allow users to get a list of recommended videos and users are able to watch it on the list or in its detailed page. The video can be displayed in full screen.
- Preconditions: None.
- Postconditions: None.
- Scenarios:

1. User chooses a video to watch and presses the play button.

2. User clicks on the comment bar of a video on the screen and enters a detailed screen of the video. User clicks the play button to play or clicks the full screen button at the bottom to play it full screen.

3.1.1.4 Filter Video List by Tag

- Requirements: Allow users to filter videos based on tags: Me, Friend, Family and Community.
- Preconditions: None.
- Postconditions: None.
- Scenarios:

    1. User selects one or more tags, and filtered results will be displayed in the list.

3.1.1.5 Chatbot

- Requirements: Allow users to ask questions regarding the video contents and find answers or tips to them. The chatbot is conversational, smart, and talkative.
- Preconditions: None.
- Postconditions: None.
- Scenarios:

    1. Chatbot greets the user when he/she firstly enters the chatbot screen. User asks question based on the video he/she watched. Chatbot answers the question by providing tips or materials to refer.

2. Chatbot is designed to automatically prevent harmful messages from children.

3.1.2 Registered Users' Use Case Analysis

3.1.2.1 Write Journals

- Requirements: Allow users to ask questions regarding the video contents and find answers or tips to them. The chatbot is conversational, smart, and talkative.
- Preconditions: User has logged in.
- Postconditions: User's journal updates in the database.
- Scenarios:
    1. User chooses a date and writes a journal of it.
    2. User is able to see his/her own journal history.

3.1.2.2 Comment on videos

- Requirements: Allow users to comment on certain videos and see their crew member's comments.
- Preconditions: User has logged in.
- Postconditions: User's comments displayed below the videos.
- Scenarios:
    1. User chooses a video and writes down some comments.
    2. User on the My Home and My Life Screen sees the comments from their crew members.

3.1.2.3 Manage Profile

- Requirements: Allow users to edit their profile, first name, last name, country, zip code, state, school district, school name, grade, password, and avatar.
- Preconditions: User has logged in.
- Postconditions: Information has changed according to the user's modification.
- Scenarios:
    1. User submits an editing request.
    2. User’s information is updated in the database.

3.1.3 Administrators’ Use Case Analysis

3.1.3.1 Check Chat Records

- Requirements: Allow admin users to check the chat records between the chatbot and the normal users.
- Preconditions: Admin user has logged in.
- Postconditions: None.
- Scenarios:
    1. User chats with the chatbot.
    2. Admin User searches the chat records by username.

3.2 Non-functional Requirements

Non-functional requirements define system attributes such as security, performance, usability, maintainability, and scalability.

3.2.1 Security

The security of the application is very critical as its target users are kids and their information is sensitive. The application uses a client-side database to handle most of the user data on client side so that sensitive data cannot get exposed to other people. Other data that is stored in cloud database is secured by cloud service and only admin users have access to them.

3.2.2 Performance

LBW should be the application providing educational contents to primary and middle school children. The main reason for creating this platform is to make kids keep in touch with each other and help them solve mental issues during the pandemic. The NLP model working on the backend is aimed to respond to users' requests at the average speed of 5 seconds per search.

3.2.3 Usability

Appropriate error messages will be shown, and errors are carefully handled.

3.2.4 Maintainability

The system will keep updating after getting the first version tested by groups of users. The code can be reused and integrated into other projects if/when necessary. The main modules of this project are maintained on GitHub, which is carefully handled by version controls and project management. The frameworks we apply on the server and client sides make it easy for developers to maintain both.

3.2.5 Scalability

The system is designed to be highly scalable for thousands of users' registration and search requests.

Chapter 4

Technology, Tools & Programming Languages Used

4.1 Client Side

4.1.1 Client-Side Scripting

Client-side scripting [2] refers to running scripts on the users' device, usually within a browser. The most common scripts that are used on the client-side are JavaScript. This type of scripting is universally supported and used to display static or dynamic content depending on a user's input. The implementation of display is achieved because JavaScript can interact with the web page via Document Object Model (DOM), to query page state and modify it.

In this project, JavaScript is used, which is a popular language supported by all common web browsers and different mobile platforms. Another reason why JavaScript is chosen to use is that over 97% of websites use it client-side for web page behavior [3], and various third-party libraries that keep up to date are contained in JavaScript. As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the DOM [4].

4.1.2 Client-Side Framework

A software framework is an abstraction in which software can be selectively changed by additional user written codes. [5] JavaScript frameworks are those written in JavaScript where

the programmers can manipulate the functions and adopt them conveniently. The most popular JavaScript frameworks in 2021 are Angular, React, Vue.JS, Ember.js and Meteor. [6]

This project is developed by using React Native framework. React Native is a custom renderer for React. In terms of React, it was firstly introduced by Jordan Walke at Facebook in 2011. [7] It is a declarative framework which makes it painless to create interactive user interfaces (UIs). Programmers can design a simple view for each state in an application and React can efficiently update and render the components according to different states. Furthermore, React is component based, thus in order to make complex UIs, it is essential to build encapsulated components that manage states and then compose them all. In this process, rich data can be passed through the application and the state stays out of the DOM. As for React Native, it's a framework based on React but for mobile applications. By contrast, it uses native components instead of web components. In addition to converting React code to work on iOS and Android, it can also access the features of these platforms.

In order to better demonstrate functions of our application to stockholders, Expo framework is also applied to warp our React Native application so that to help develop, build, deploy and quickly iterate on Android and iOS. It is built around React Native and native platforms and shares the same JavaScript/TypeScript codebase with these platforms, but it enables display of the front-end screens by scanning the QR code with the Expo app.

4.1.3 Client-Side Database

Client-side database is a kind of database that stores data on the clients' machine and retrieves it when needed. The features of client-side databases are that it can allow offline use of the

application, cost less time of download for the first use, persist previous activity, and personalize application preferences. We value the privacy of our clients, so a database is set up on the client-side to store private and sensitive user data. This database only provides access to the local user of the device, which is ideal for protecting children's data.

There are several local databases supporting react native app development. After balancing their strengths and weaknesses, we decided to use SQLite which is ideal for mobile apps. SQLite is a lightweight database that needs minimal setup efforts. The benefits of applying it are various. Firstly, it is possible to integrate it within the application, so users don't need to wait until all the data get downloaded to enjoy the mobile app. Secondly, for developers, it's an ACID-compliant database, implementing almost all SQL standards, which means it's painless to interact with this database using SQL commands. In addition, SQLite has file-based library architecture, and it enables capability of handling all data types efficiently.[8] Finally and most importantly, our frontend framework – Expo has an expo-sqlite plugin to manage the data within the mobile application for offline persistence.

4.2 Server Side

4.2.1 Server-Side Scripting

Server-side scripting is a technique that involves employing scripts on a web server. [9] These scripts can produce a response customized for each client's request in the application. In this project, we used Python as our sever-side scripting language. Python is a high-level general-purpose programming language. It's a structured, object-oriented, and functional programming language, which aims to help programmers write straight-forward, logical, and easy-to-read code for both small and large-scale projects.

4.2.2 Server-Side Framework

Flask is applied at the server side to support smart search engines. First created by Armin Ronacher in 2004, Flask is a lightweight WSGI web application framework written in Python. [10] As it does not require tools or libraries when using, it is classified as a micro web framework. Also, it has no database abstraction layer or any other pre-existing third-party library components. However, Flask supports extensions so that programmers can add application features and those extensions are for object-relational mappers, upload handling and other common framework related tools.

The reason for using Flask is that it is easy to use in our case as we need a lightweight Python framework to better connect our machine learning model to our backend calculation and database. In addition, this framework is easy to develop and easy to maintain applications but still has high scalability. In conclusion, it is an ideal framework for us to save time and money in the developing and testing stage.

4.2.3 Server-Side Database

Since Flask is a lightweight framework at the back end, to match it, we use SQLite database which is also a lightweight disk-based database on the server-side. SQLite database does not require a separate server process and it can be accessed by nonstandard variants of the SQL query language. The process of SQLite database connection is simple and easy, just use connect() method to connect a certain database after importing the sqlite3 module.

Chapter 5

System Architecture and Design

System architecture refers to the structural design of systems and it consists of system components development. In this chapter, system architecture and its designs are described. These covered the designs of system architecture, system behavior, database, and components.

5.1 System Architecture

5.1.1 React Native Architecture

React Native architecture is composed of three main modules: the core, the mobile app, and the web application (in figure 1). The core contains the business logic and state of the application. These are maintained by the Redux framework. Redux is a library that deploys pure Flux variants. Redux and Flux are both used for a unidirectional flow of data within the app. The mobile app is implemented as a standard React Native application with the use of containers and components. In terms of the web application, it's a standard React application.

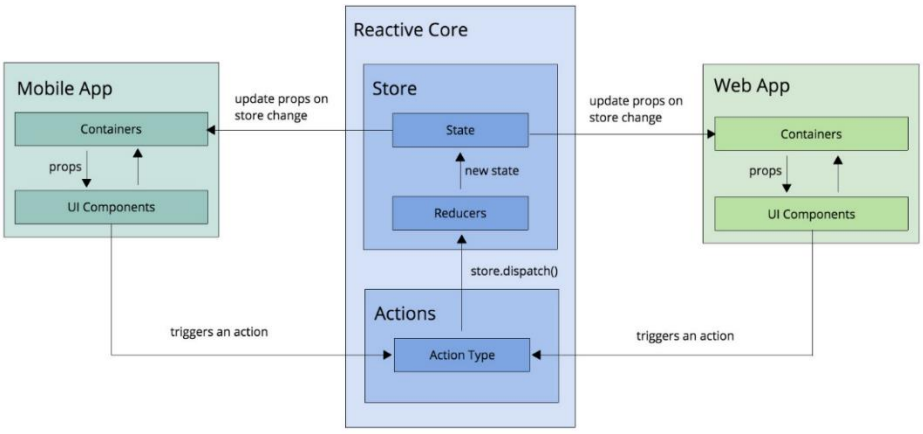

Figure 1 React Native Architecture Diagram.

In a nutshell, this architecture allows the developer to structure a multi-platform mobile application and keep the business logic in a reusable and maintainable sub-module. In the native realm, the developers can use native languages to develop for either iOS using Object Swift or Android using Java, but also the main UI thread is available for changing basic UIs. [11]

5.2 Database Design

For this application to handle users' data access, management, modification, update, control and organizing, SQLite plugin has been installed on client side and configured to store private data.

Figure 2 describes the structure of table Videos. The data in this table gives all the unique information that is related to a video, and segment indicates our segment training results. The segments are close to the main topic in each video. Another table vector_list is associated with this table where ids are used as a foreign key. In table vector_list, the pre-processed video transcripts are stored.

Table: videos		
Column	Data type	Constraints
id	INTEGER	
segment	TEXT	
url	TEXT	
transcript	TEXT	
tag	TEXT	
tip	TEXT	
segment 2	TEXT	
Global table constraints		
•PRIMARY KEY("id")		

Figure 2 Table Videos Structure Diagram.

5.3 File Structure

Table 1 describes the file and folder structure. These files are on the front end and based on Expo framework to use and test. The babel.config.js is the project-wide configuration file where located configurations that need to be applied broadly. The index.js registers the whole app and ensures that whether loading the app in the Expo client or a native build, while the App.js controls what the app looks like. App.js contains the header, bottom tab, and all screen components so that screens are in the same stack containers and can be able to switch from one to another with the change of the header content.

Table 1 File Structure and Functions

File	Function
Home.js	Home.js contains the home component that controls what My Home screen looks like. Home Component is on the top of the navigator stack. The special font used in the app is loaded starting from this file.
Gives.js	The display of My Gives screen is controlled by this file. A new Stack container is also created in this file to be able to enter the Create a New Give screen.
Mission.js	Mission.js controls the display of My Mission screen where Add a New Mission screen is associated.
Life.js	Life.js does not only consist of the styling, structure, and layout of My Life screen, but to activate smart search, asynchronized database loading is also contained in this file.
CustomHeader.js	This file is a global file that displays the custom layout and styles of the header.
FloatingButton.js	FloatingButton.js contains the global view and animation of the floating toolbox on each screen.
PlayVideo.js	PlayVideo.js includes the container of video view and its styles.
GPTChatBot.js	This file includes the chatbot view and it uses CallGPT.js to send and listen to API calls. The returned messages will be sent to the front view.

5.4 APIs and Libraries

5.4.1 Hugging Face Models

Hugging Face models are applied on our backend to support our machine learning features. In our My Life Screen search engine, we want our young age users to get optimal video results no matter what questions or concerns they input in the search bar. Therefore, we use a sentence transformers model to map sentences and paragraphs to dimensional dense vector space, contributing to sentence similarity comparison. msmarco-MiniLM-L-6-v3 model is chosen as it has a good performance and high speed but with low size cost. It is an ideal model for us to use for search engines. Its response to each search is quick and highly accurate.

Msmarco-MiniLM-L-6-v3 model applied BERT network and made a modification on it using Siamese and triplet networks. [12] The modified version is called Sentence-BERT (SBERT). The basic BERT network is composed of 12 successive transformer layers and each of them has 12 attention heads. The improvement done by SBERT is reducing the effort for finding the most similar pair from 65 hours to about 5 seconds while maintaining the accuracy from BERT. Derived semantically sentence embeddings can be compared by cosine similarity.

5.4.2 GPT-3

Since Generative Pre-trained Transformer 3 (GPT-3) is a powerful conversational language model created by OpenAI, we decided to use it in our chatbot feature to improve user experience. GPT-3 is a successor to GPT-2, but its capacity is much larger with 175 billion machine learning parameters, possessing a high quality and speed of text generation ability. [13] It can be difficult

to determine whether the text generated by GPT-3 is written by a real human or not, so the OpenAI researchers warned that GPT-3 had potential dangers and should be under monitoring to avoid risk. Therefore, when using the chatbot in LBW app, it will say at the beginning it is only a chatbot but a real person.

Chapter 6

System Implementation

In this chapter, the main parts of the LBW mobile application implementation are discussed.

6.1 App Structure Implementation

Almost every screen in the app consists of a header, a bottom navigation bar, a side bar, and main contents.

6.1.1 The Header

Figure 3 describes the header of the main screen as well as the main content. The header on the main screen is dynamic. It will change according to the name of the logged in user. On other screens, the header is static, which shows the name of the present screen. To achieve the header, we used React Stack Navigator to implement the header structure we want. Stack Navigator allows an app to transition between screens where a new screen is placed on top of a stack. Hence, by applying it we are able to comfortably navigate from parent screens to child screens. Besides, for building a more attractive UIs for children, we custom our header style. This is achieved by constructing a custom header component and passing properties to it if needed.

Figure 3 Home Screen & Hovering Toolbox

6.1.2 Navigation Bar

Figure 3 also describes the bottom navigation bar. The navigation bar on each screen is static. It is implemented by applying Bottom Tab Navigator. A tab bar on the bottom can let users switch between routes. Routes are lazily loaded in React Navigation. the screen components are not mounted until we first focus on them.

In most cases, apps only use one navigator to display more contents. However, in our case, we consist of many functions and both two navigators can let new users know which screen they are on and what they can do here. In order to create a custom bottom tab navigator and a custom header, we set up an individual stack and tab navigator for each screen and include the main content and header of each screen into its tab screen, making the header match the bottom tab.

6.1.3 Side Bar

On each screen a hovering toolbox button is displayed. The screen when opening the side bar is described in Figure 3. If clicking this button, it will give a hovering bar that contains a sidebar navigating to Journal, Question and Score screens. This is implemented by creating a custom floating button. An animation is added, and it shows whenever opening or closing the sidebar. This animation makes the opening action look smoother and users can know if they open the toolbox, the displayed side bar is focused. Users can close the sidebar by clicking outside of it or clicking the close icon.

We add a sidebar rather than an app bar in the header because the header design is already rich in color and contents. Using the app bar can make the header look too complicated. Compared with an app bar, a side bar is excellent to place commonly used tools and the visual design is also great to attract users. The stack navigator of this side bar is created as a part of each screen component, which means the sidebar on each screen is independent.

6.1.4 Main Contents

The main contents are warped in the navigators. There are mainly four categories of them: My Home, My Life, My Mission, and My Gives. The implements of each screen and functions will be discussed in the next few sections.

6.2 Main App Implementation

In this section, the implementation of each main content is explained. Expo and React Native have many useful and powerful libraries and widgets, which simplifies our front-end implementation.

6.2.1 My Home Screen

Figure 3 describes the main screen. The bottom tab is selected My Home Screen by default when first enter. The main screen contains a video, crew members (friend list) and a comment list. The video is a welcome video that introduces our LBW project and how we help kids within our app. The friend list is contained in a horizontal FlatList whose length is dynamic. The comment list is put in a vertical FlatList displaying the comments on the video from crew members.

6.2.2 My Mission & My Gives

My Mission and My Gives screen are displayed in Figure 4. On My Mission screen, there is a scrollable tab view that is implemented by ScrollableTabView fragment. As its name goes, it’s scrollable and has a similar character as a tab. User can scroll to change from level1 to another and can also tab the bottom tab bar to select different levels. Each flat list in each level is standing alone. Their other features are still under development.

Figure 4 My Missions & My Gives Screens

6.2.3 My Life

My Life is a vital fragment in our app, showing in Figure 5. This screen mainly aims to display LBW videos from YouTube. This screen consists of three parts. Top custom tag bar, smart search bar and a video list. As the tag bar indicates, videos are divided by four different categories, me, family, community, and environment. They can be filtered by clicking on the single or multiple tags at the top of the screen. Below the tag bar is the smart search bar, working distinctly from the common search function. The implementation of those features will be discussed in detail in the next section. When clicking on the comment bar of a video, it will navigate to video detail screen. In this screen, the video, tips for this video and comment input field are displayed.

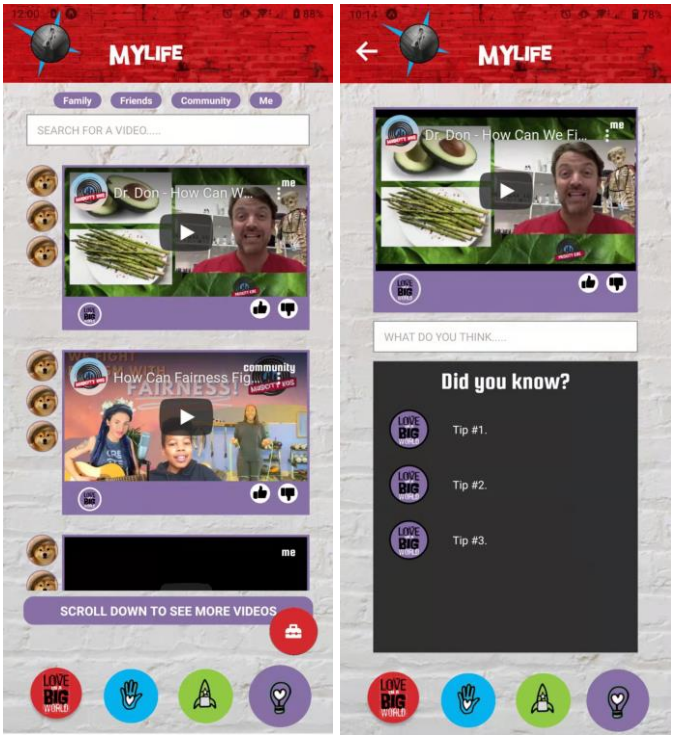

Figure 5 My Life & Video Detail Screens

6.3 Implementation of Features

6.3.1 Video List Display

Videos are listed in FlatList on My Life Screen in Figure 5. This list is scrollable, and users can see more videos after scrolling down. A video window is composed of a decoration border, a tag, a comment field, like & dislike buttons, and a crew member list.

The Play Video feature is implemented by applying React Native WebView. It is a cross-platform web view for iOS, Android, macOS and Windows react native apps. Also, it has Expo SDK support. WebView is a component that can load web pages within React Native app. It can stand alone without the box in React Native.

6.3.2 Smart Search

Smart search is activated by simply entering questions or words in the search bar below the tag view. After the python backend API receives the input through POST request, it will handle the sentence by following code in figure 6.

```
@app.route("/post", methods = ['POST'])
def query_api():
    payload = request.json['input']
    if payload is None:
        return "Loading False"
    model = SentenceTransformer('sentence-transformers/msmarco-MiniLM-L-6-
v3')
    embeddings = model.encode(payload)
    transcript_list = getVectors()
    lists = embeddings.tolist()
    similarity_list = []
    for i in range(60) :
        similarity_list.append(cosine_similarity(lists, transcript_list[i]))
    json_str = json.dumps(lists)
    #get the two largest similarities in the list
    largest_list = heapq.nlargest(2, range(len(similarity_list)), key = similarity_list._
__getitem__)
    #res_list = getUrls(largest_list)
    json_str = json.dumps(largest_list)
    if payload is None:
        return "loading false"
    return json_str
```

Figure 6 Smart Search Code

This code firstly handles “no data received” issue. If the backend receives nothing in POST request, it will return a “Loading false” message that can see in front end console. Otherwise, the sentence transformer model msmarco-MiniLM-L-6-v3 will be called and its encoder will translate the sentences to vector list. Then, pre-progressed vectors of video transcripts will get loaded from database. The aim of the pre-progress is to avoid time and space consuming during

API calls. The transcripts are dealt with the same model for compiling through the same word cluster as the input sentence so that the similarity is computable using vector lists of each transcript and input. For similarity result calculations, the measurement of distance between two non-zero vectors of input and transcripts is essential. Cosine similarity is applied here to measure an inner product space of two vectors. The calculating equation is:

$$cosine\ similarity = Sc(A,B) \coloneqq \cos\theta = \frac{A \cdot B}{||A|| \cdot ||B||} = \sum_{i=1}^{n} \frac{A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \quad (1.1)$$

In equation 1.1, A_i and B_i are components of vector A and B respectively. [14] After returning the results from cosine similarity, they are ranked, and top two videos are finally returned to the front end.

Since the font end side receive the URL of videos, they need some processing to turn to fit the Play Video field. This is achieved by changing them to embedded YouTube video links with necessary setting extensions, such as invalid auto-play & info shows and valid video controls. The processed links enable users to alter video playing preference just like in YouTube webpages. In figure 7, one search result is displayed.

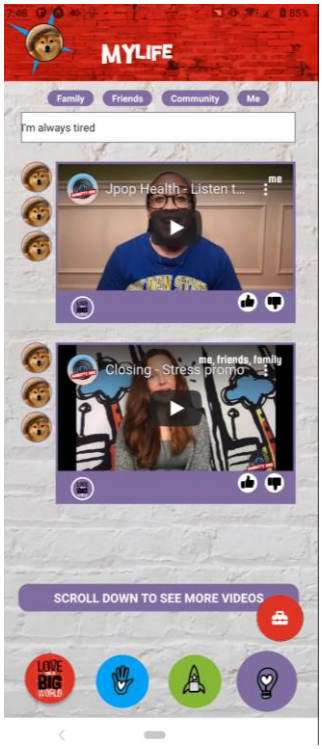

Figure 7 One-try Results of Smart Search

6.3.3 Tag Filter

The tag view is also customed. The tag will change its color to opposite one if it is clicked, giving a visual effect that certain tags are selected correctly. After one or multiple tags are selected, the video list will automatically refresh and filter the videos by selected tags. This is handled by a call back method to acknowledge which tags are selected or not and then reset the state of video list data. The call back method is called each time when users modify the tag selection. The example of using tag filter is in Figure 8.

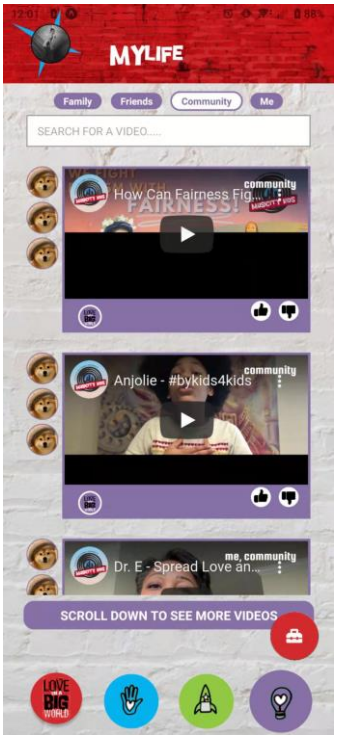

Figure 8 Results of Using Tag Filter

6.3.4 Chatbot

When the user selects the icon in the side bar, the chatbot screen is entered. The screen is shown in Figure 9. The chatbot will automatically greet the user, give a brief introduction that she is not human, and guide users to ask question. Since she is a conversational chatbot, she can answer the questions based on some sentence logic but not as good as a real mental health mentor. The aim of the chatbot is to provide some suggestions and tips on how to deal with problems users have and cheer them up if they meet something depressed. The chatbot can be divided into two main functional fragments – language filter and chatbot model.

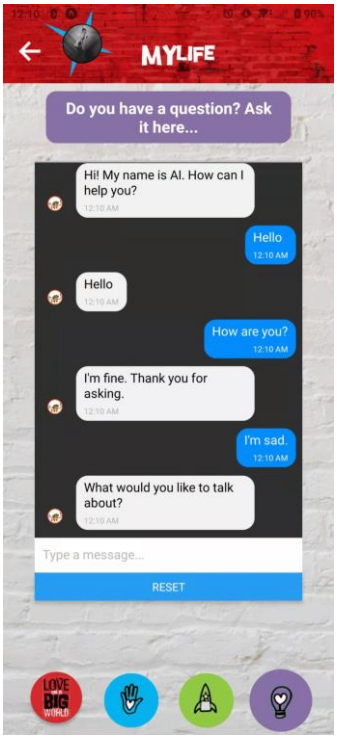

Figure 9 Chatbot Screen

6.3.4.1 Language Filter

Since the target users of the LBW app are middle children and the contents generated by machine learning models may be threatening to them, it is important to add a language filter to our chatbot to prevent potential harmful words. The first step of adding a language filter for the team is to compare which language filter works better on filtering toxic sentences. We mainly compared two language filters that can work properly in our app. The first one is GPT-3 content filter coming with GPT-3 model that can detect sensitive or unsafe text generated by the API. The other one is Toxic-BERT that aims to stop harmful content in multiple languages. To compare the two language filters, we tested each of them with the Jigsaw Unintended Bias in Toxicity Classification dataset and the compared results are in Table 2.

Table 2 Compared Scores of GPT-3 Content Filter and Toxic-BERT

	GPT-3 Content Filter	Toxic-BERT
Precision Score	34.024	68.059
Recall Score	23.541	49.187
F1 Score	27.828	57.104
Total Score	89.019	93.639

For evaluation, we calculated the score by regarding labeled 0.5-1 as toxic and the rest of them as nontoxic, and data for score counting is randomly taken 10% from the dataset. After comparing this score, we found Toxic-BERT has higher accuracy. Despite that, taking a deeper look into the results, we can conclude GPT-3 content filter is very sensitive to racial, political, nationality and religious topics. Different from giving a score of toxicity, GPT-3 content filter labels text into 3 levels – “0” for safe text, “1” for sensitive text and “2” for toxic text. Most of our test data are labeled “1”, and when we manually checked the contents and labels, the finding is that even a word in a sentence is related to the sensitive or protected topics mentioned above, it will mark “1” for that sentence.

After balancing their advantages and disadvantages and thinking of our young target audiences, we decided to apply both of language filters together. The output from the chatbot will be sent to the two language filters one by one and then return to the front end.

6.3.4.2 Chatbot

After comparing most existing chatbot models are available for low cost, firstly chose Rasa for training and testing. However, the version of Rasa at our research time did not satisfy our user requirements that the chatbot has memory during a chat and it knows what it has said. In other

words, it should be a conversational AI. Researching on the conversational AIs in the current market, we find GPT-2 and GPT-3 are ideal for the application. Among these two, GPT-3 is an incredible and powerful conversational AI introduced in 2020 but it did not offer fine-tuning options. In contrast, GPT-2 whose performance is lower at most of time but can get fine-tuned to grow stronger. Thus, exploring the performance of GPT-3 without fine-tuning and GPT-2 with specific fine-tuning is important.

For GPT-2 fine tuning, we used the dataset collected during ConvAI2. The data were collected from dialogues of PersonaChat dataset. The purpose of training GPT-2 with these data is to improve dialogue coherence of the chatbot as well as give it a dedicated personality. In Figure 10, it is the loss curve in every ten epochs from 0 to 200. The best network epoch number is 175.

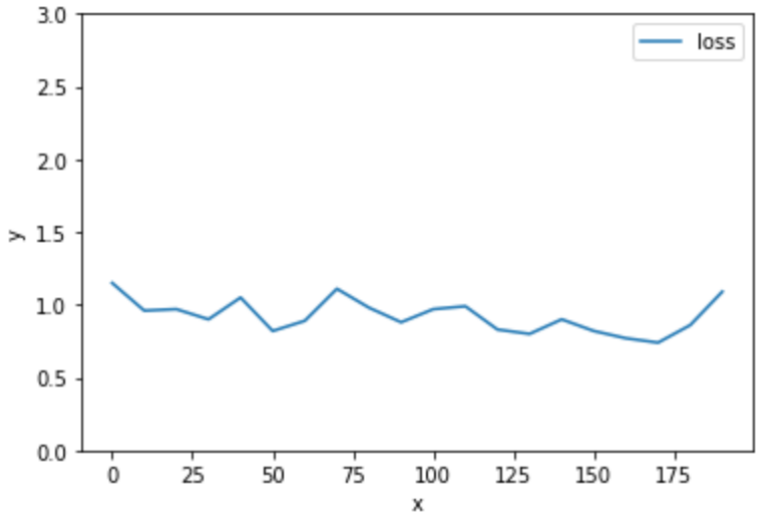

Figure 10 GPT-2 Training Loss Curve

From Figure 11, we can see that fine-tuned GPT-2 is able to generate answers to certain questions and even ask questions in some situations. Also, it is happy to share its persona in our chat. However, when we ask the same question in one conversation, it may say different answers to it. We can conclude that fine-tuned GPT-2 still lack logic when using it as a chatbot.

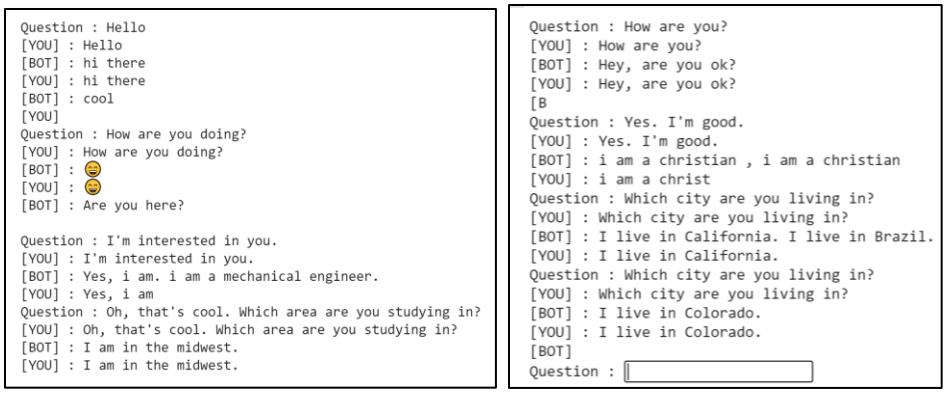

Figure 11 Sample Conversations with Fine-tuned GPT-2

There are several models contained in GPT-3, such as Davinci, Curie, Babbage, and Ada. The model we chose is Davinci who is the most capable engine among the four. For applications requiring a lot of understanding of the content, Davinci can produce the best results. Furthermore, Davinci is good at understanding the intent of text. It can solve many kinds of logic problems and explain the motives of characters. [15] In Figure 12, it is pointed out that GPT-3 has stronger logic in conversations. When set prompts to give it a persona, it will keep this persona and answer/ask questions in the shoes of the persona. Therefore, GPT-3 is embedded in the system to provide the chat AI feature.

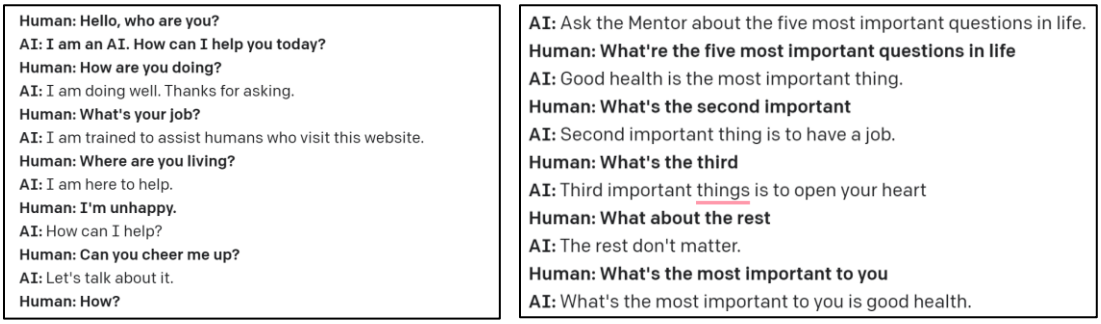

Figure 12 Sample Conversations with Fine-tuned GPT-2

The implementation of chat display is by applying React-Native-GiftedChat. The messages of conversations between bot and users are controlled by state messages. The different characters

can be distinguished by different ids, names, and avatars. The onSend property controls the action after “send message” is pressed. In our case, it sends the message to GPT-3 API and then the results will be appended to the message states which will finally display in the GiftChat widget.

Chapter 7

Drawbacks and Future Enhancements

There were some errors occurring during development, mostly related to project planning and approach selection. Also, the development process did not adopt agile development at the first beginning. Due to the lack of development members, some planned features and screens are still unfinished. A better timetable would have been necessary with proper time estimations and clear destinations. Moreover, the breadth of knowledge required for this project is beyond expectation, consuming much time in learning, training and trying.

There is also big shortcoming that the application lacks pop-ups and notifications to instruct users. Although the UI designs are straight forward, for new users, more notifications can effectively improve user experience and help avoid misunderstanding.

In terms of future enhancements, our application still has a long way to go. Below are some of areas that should get further developed or are still in-progress:

- Login, setting and other user-related screens are in demand. In the future development, the experience of different levels of users should be carefully considered.
- Automated tests should be highlighted from the developer's perspective. Such tests are helpful to ensure user requirements have been satisfied when sharing it with stockholders.
- The chatbot can get upgraded with the release of fine-tuning GPT-3 options.

Chapter 8

Conclusion

The LBW mental health mentor app is implemented on the Android platform. The app applies various frontend and backend technologies, achieving the demands from middle-aged users. These include open-source Expo and React Native framework, which is built from JavaScript, Expo & third-party libraries and its own syntax, and also with technologies such as RESTful APIs. At the server side, Flask framework is chosen to carry Hugging Face machine learning models.

In this project, many development issues occurred at both the client-side and server-side during the creation of the web application. These issues were resolved eventually by conducting research of solutions and making changes to more compatible structures.

To enable smart search and chatbot features in this web application, a research of NLP models was done, and strategies were applied to compare cost and efficiency of a model. Msmarco-MiniLM-L-6-v3 model is used to process vectors of video transcripts and user search inputs so that we can calculate the similarity of them and recommend the closest two videos for the user. Both GPT-3 content filter and Toxic-BERT were decided as the language filter for chatbot. GPT-3 beat our fine-tuned GPT-2 model and is chosen for processing chat output.

This project gets off to a good start of LBW app. It is constructed by most recent framework and supported by up-to-date APIs, libraries, and machine learning models. The cross-platform compatibility of the app will be achieved soon when adding iOS structure files at the frontend.

Bibliography

[1] "COVID-19 Educational Disruption and Response". UNESCO. 2020-03-04. Retrieved 2021-08-23.

[2] Wikipedia. Client-Side Scripting, 2021. https://en.wikipedia.org/wiki/Dynamic_web_page#Client-side_scripting, accessed Aug 2021.

[3]"Usage statistics of JavaScript as client-side programming language on websites". w3techs.com. Archived from the original on 2021-08-13. Retrieved 2021-04-09.

[4] Wikipedia. JavaScript, 2021. https://en.wikipedia.org/wiki/JavaScript#cite_note-deployedstats-12, accessed Aug 2021.

[5] Wikipedia. Software Framework, 2021. https://en.wikipedia.org/wiki/Software_framework, accessed Sep 2021.

[6] Hackr. Best JavaScript Framework, 2021. https://hackr.io/blog/best-javascript-frameworks, accessed Sep 2021.

[7] React (JS Library): How was the idea to develop React conceived and how many people worked on developing it and implementing it at Facebook. Quora. Archived from the original on February 11, 2015. Retrieved 2021-08-23.

[8] QA InfoTech. Top Local Databases For React Native Mobile App Development In 2021, 2021. https://qainfotech.com/local-databases-for-react-native-mobile-app-development/, accessed Sep 2021.

[9] Wikipedia. Sever-side Scripting, 2021. https://en.wikipedia.org/wiki/Server-side_scripting, accessed Sep 2021.

[10] Wikipedia. Flask,2021. https://en.wikipedia.org/wiki/Flask_(web_framework), accessed Sep 2021.

[11] Rahul Dubey. React Native Architecture, 2020. https://www.educba.com/react-native-architecture/, accessed 2021.

[12] Nils Reimers, Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Archived from the original on Aug 27, 2019. Retrieved on Sep 23, 2021.

[13] Bussler, Frederik (July 21, 2020). "Will GPT-3 Kill Coding?". Towards Data Science. Retrieved August 1, 2020.

[14] "COSINE DISTANCE, COSINE SIMILARITY, ANGULAR COSINE DISTANCE, ANGULAR COSINE SIMILARITY". www.itl.nist.gov. Retrieved 2020-07-11.

[15] OpenAI. GPT3 Engines Docs. https://beta.openai.com/docs/engines, accessed Sep 2021.