FYP22033

Recommendation-Based Community Helpers Mobile Application

Interim Report

Group Members:
Lau Chun Leung 3035757066
Wong Ho Yin 3035762580
Wong Sze Lung 3035762190

Supervisor:
Dr. Chim T W

Date of Submission: 22 January 2023
Abstract

In the 21st century, although rapid advances in technology means that it is now easier than ever to connect with others, however, the connection between people in a community can still be poor. This project aims to combine the concept of share economy to create a platform that links up the community. We will develop a multi-platform application, and allow users to seek help from the community, or become a helper to help others in the community. Additionally, this project will explore and apply the recommender system to provide the best matching with the helpers and supplicants. The project adopts an approach of building a mobile apps prototype by React Native and Nest.js. The business model of the prototype will refer to similar apps on the market such as Gaifong, Toby, Airbnb and Uber. Then, collecting the user’s implicit and explicit data to build a well-trained recommender system. Currently, this project has already finished phase one and two deliverables, including the system architecture design, starting development cycle of front-end and back-end sides. Next, our team will initiate the most important part of the platform, which is the recommender system design and training. It is estimated that the application will start the testing phase in early 2023.
Acknowledgements

We would like to thank our supervisor, Dr. TW Chim, accept our invitation to supervise this final year project. Without his valuable advice, inspiring discussion, and continuous support, this project would not be completed.

Moreover, we wish to praise our teammates, who are Lau Chun Leung, Wong Ho Yin, and Wong Sze Lung. Their hardworking and cooperation bring energy to this project to push it forward.
# Contents

Abstract I  
Acknowledgements II  
List of Figures V  
List of Tables V
Abbreviations VI

1 Introduction 1  
  1.1 Background and Motivation ................................. 1  
    1.1.1 Talent in the neighbourhood ............................ 1  
    1.1.2 Related Work ........................................ 1  
  1.2 Project Objective .......................................... 3  
  1.3 Project contribution ........................................ 3  
  1.4 Outline of the report ....................................... 4  

2 Methodology 5  
  2.1 Introduction .................................................. 5  
  2.2 System Overview ............................................. 5  
  2.3 Main Features (FE) .......................................... 6  
    2.3.1 FE-1: Request Helper .................................... 6  
    2.3.2 FE-2: Browse Requests ................................... 6  
    2.3.3 FE-3: Helper Matching .................................... 7  
    2.3.4 FE-4: Helper Search ...................................... 7  
    2.3.5 FE-5: Chat Room .......................................... 7  
    2.3.6 FE-6: Transaction ........................................ 7  
    2.3.7 FE-7: Content Management System (CMS) .................. 8  
  2.4 Frontend ...................................................... 8  
  2.5 Backend ...................................................... 9  
  2.6 Recommender System ......................................... 10  
    2.6.1 Collaborative Filtering .................................. 11  
    2.6.2 Hybrid Model ............................................ 11  
    2.6.3 Graph Neural Network ................................... 11  
  2.7 Summary ..................................................... 12

3 Project status 12  
  3.1 Introduction .................................................. 12  
  3.2 System Architecture ......................................... 12  
    3.2.1 PaaS - Azure Cloud Computing Service ................. 13  
    3.2.2 CI/CD .................................................. 14  
  3.3 Recommender System design ................................ 14  
    3.3.1 Batch vs Real-time Recommender System ............ 14
3.3.2 Tentative workflow of the Recommender System . . . . . . . . . . . 15
3.3.3 ML model in the Recommender System . . . . . . . . . . . . . . . . . 15
3.4 Frontend Progress . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
  3.4.1 Community Helper Apps Overview . . . . . . . . . . . . . . . . . . 16
  3.4.2 More about React Native and Expo . . . . . . . . . . . . . . . . . . 20
3.5 Backend Progress . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
  3.5.1 Authentication . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
  3.5.2 Security . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
  3.5.3 Matching . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
  3.5.4 Preparation for future development . . . . . . . . . . . . . . . . . . 24
3.6 Challenge and Limitation . . . . . . . . . . . . . . . . . . . . . . . . . . 24
3.7 Project Schedule and Future plan . . . . . . . . . . . . . . . . . . . . . 24

4 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
List of Figures

1.1 Picture of GaiFong application ............................................. 2
1.2 Picture of Toby application .................................................. 2
2.1 System Overview Flow Diagram ............................................. 5
3.1 System Architecture .......................................................... 13
3.2 Workflow of the Recommender System ................................. 15
3.3 Model structure of the Wide & Deep model ............................. 16
3.4 An animated splash screen ................................................... 17
3.5 The default screen of apps ................................................. 17
3.6 The details screen for a help request ..................................... 17
3.7 The main screen if user login .............................................. 18
3.8 The screen to new a help request .......................................... 18
3.9 The details screen of a request with accept button .................. 19
3.10 The job status for a helper ............................................... 19
3.11 ”Fix my TV” request removed on the home screen ................. 20
3.12 Expo go helps testing the apps on my own device ................. 21
3.13 Posting login mutation method to backend server ................. 22
3.14 The response of login ...................................................... 22
3.15 Decoded JWT token .......................................................... 23
3.16 Flow of recommender modules .......................................... 24

List of Tables

3.1 Project Schedule .............................................................. 25
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CF</td>
<td>Collaborative Filtering</td>
</tr>
<tr>
<td>CI/CD</td>
<td>Continuous Integration/Continuous Deployment</td>
</tr>
<tr>
<td>CMS</td>
<td>Content Management System</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update, Delete</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>FE</td>
<td>Main Feature</td>
</tr>
<tr>
<td>FPS</td>
<td>Faster Payment System</td>
</tr>
<tr>
<td>GNN</td>
<td>Graph Neural Network</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JSX</td>
<td>JavaScript XML</td>
</tr>
<tr>
<td>JWT</td>
<td>JSON Web Token</td>
</tr>
<tr>
<td>KGAT</td>
<td>Knowledge Graph Attention Network</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>NCF</td>
<td>Neural Collaborative Filtering</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>RS</td>
<td>Recommender System</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UX</td>
<td>User Experience</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background and Motivation

1.1.1 Talent in the neighbourhood

We notice that people in a neighborhood have many different special skills. They also have a lot of problems that need to be solved every day or every minute. There will always be some problems that people cannot solve on their own or solving them alone is not efficient. For example, electrical maintenance, equipment rental, etc. We believe people would appreciate that there is someone who has the knowledge and is willing to help others out to solve a particular problem in their neighborhood. Conversely, if someone from the community who has a lot of spare time and he/she has specific knowledge or professional skills and is willing to share with their neighborhood, then, as long as he/she becomes a helper, he/she not only can help people but also could be able to earn money. It should be a win-win situation for the community. Therefore, we want to build a platform to make people strongly connect with their neighbors. Let us share neighbors a helping hand.

This project will create a multi-platform application that could link up the community. For example, the service provider can post their profile on the platform and illustrate what kind of personal skills or materials they can provide to the community as a service. They might be good at cooking, provide tools that others need, or even help someone repair furniture and appliances. The helper can receive a number of earnings in return.

1.1.2 Related Work

There are several applications with similar ideas in the market. All of them have a common objective which is better utilize some idle resources in our society and use them to serve others. However, those existing platforms have room for improvement. The following would highlight the similarity and improvements of the platforms in the market.

1.1.2.1 Gaifong

Gaifong is an application that allows users to rent stuff from their neighborhood [1]. Users can rent books or even repair equipment from their neighborhood. It provides a location service to find things that can be borrowed nearby. Users could receive rent for lending their own stuff to others. The Figure 1.1 is a picture of GaiFong application.
The idea of this application is very similar to our idea. But instead of lending stuff to others, we want to provide service to our users. For example, instead of lending electric drills, we want the neighborhood to provide the repair service. With this alternative perspective of sharing resources, we can provide some services that users are not able to complete or want someone to do for them.

1.1.2.2 Toby - Hire Local Service

The business operation of Toby is very similar to the application that we envision. They provide a lot of services on the platform. Users can browse different categories and choose the service they want [2]. The Figure 1.2 is a picture of Toby application.
Those services came from small and medium-sized enterprises on their platform which
gives a good chance for those businesses to grow up. The services they provide might
be more professional. For example, the service provider must have a business register
for a moving company to start providing service on the platform. What we want to
achieve is that users do not need to be a professional to help others. As long as users
are able to complete the task and the help targets are near the user. Then, a user can
start providing service.

The problem with those applications is that they do not provide personalized experience
to users. It would be much better if the platform can be able to suggest the related
service to users.

1.2 Project Objective

In this project, we aim to develop a multi-platform (iOS & Android) application that
includes the following main features (FE),

- FE-1: Request Helper
- FE-2: Browse Requests
- FE-3: Helper Matching
- FE-4: Helper Search
- FE-5: Chatroom
- FE-6: Transaction
- FE-7: Content Management System (CMS)

Further detail would be discussed in chapter 2.

1.3 Project contribution

Share economy has gained prominence in recent years and this economic model indeed
has achieved success in some industries. For example, Uber in the ridesharing industry,
Airbnb in the holiday accommodation industry, etc. We notice that there is a lot of
potential in the share economy model. On the other hand, currently, there is no com-
munity helper matching application in the market. As mentioned in section 1.2, there
are some similar idea applications in the market, which are Gaifong and Toby. However,
there is a key difference between our application and those existing applications, which
are general services and have low barriers to entry in our application.

In terms of general services, Gaifong limited users only can rent some things from other
users. The user can not provide other services apart from rental services. In our design, there are no limits to the range of services provided by our users. Users can provide cooking service, maintenance service, etc., in order to provide a more general service to our users.

In terms of low barriers to entry, some applications in the market, such as Toby, do provide various services to the users. However, in Toby, there is a barrier to becoming a service provider, such as a business registration certificate. The high barriers to entry rejected ordinary people in the community to become service providers or helpers. In our design, we would like to let everyone in the community become a helper to provide service to others.

Therefore, our application is different from the existing solution and can provide a unique service to the market.

1.4 Outline of the report

There are four chapters in this report. The first chapter is an introduction, which gives an overview of this project, including the project motivation, and existing similar projects in the market. Then, it set up the project objective as well as the main features of this application and the significance of this work.

In chapter two, it offers the methodology used in this project. First, it provides an overview of the system architecture to illustrate how the whole system works. Then, it will mention the frontend and backend technology used in this project in detail. Lastly, it analyzes the recommender system that will be used in this system.

Chapter three will report the current status of this project. It states the project progress that we completed now, including the system architecture, progress of the recommender system, frontend, and backend. Also, it gives the project schedule and the future plan of the project.

Lastly, Chapter four will conclude this report. It will recap the project background and solution. Then, it summarizes the completed work, challenges and future steps.
2 Methodology

2.1 Introduction

This chapter will mainly introduce the system design and technology that we would apply to this project. First, we would present the system overview in a flow diagram. Then, we would go through each main feature of this application in detail. Finally, it would also discuss the technology that we would use for the frontend, backend, as well as recommender system.

2.2 System Overview

Figure 2.1 presents the overall flow of this system which briefly show what user can do in our application. When the user opens our application, they can become a helper, or seek help from the helper.

![System Overview Flow Diagram](image)

Figure 2.1: System Overview Flow Diagram

If the user decides to seek help, he/she can just create a help request in the system, then our system intelligence would automatically match the help request to the most suitable helper. Apart from that, users can just search for the helper and then create a direct help request for that helper. If both sides agree with the help request, then a help event would be established.
If the user decides to become a helper, he/she can actively browse the helper requests and decide whether to accept or not. Or, users can passively wait for a helper request that matches the user. Again, once both sides agree with the help request, a help event would be established.

Therefore, there are two parties for our user to be in this application, helper and the person who seeking help. There is no limitation for users to become which party. A user can be a helper this time, and seek help from a helper next time. Therefore, this system provides flexibility to the user.

2.3 Main Features (FE)

2.3.1 FE-1: Request Helper

Seeking help from the community is one of the main features of this application. In this app, users can create a request in the system to seek help from the helpers. When a user creates a new request, it should be provided some description/detail of the request, such as

- Category
- Short description of the request
- Date & time
- Location
- Illustration image/photo
- etc.

Note that, the user can specify the time period to schedule the requests when creating a request. After the request is created, the system would start to find the most suitable helper for this request. For more detail on helper findings refer to FE-3: Helper Match section. Once the system finds a suitable helper, that helper would receive a notification to invite him/her to that helper request. That notification could be sent via system notification (iOS/Android) or email, etc. From the helper’s perspective, he/she could review the request detail to decide whether to accept this request or not. If the helper accepts that request, then a help event is established.

2.3.2 FE-2: Browse Requests

Apart from passively waiting for a helper request, the helper can actively browse over the pending helper request. When the helper finds a helper request that he/she is interested in, the helper can directly accept the request to establish a help event for
the requester without waiting for the request notification. The list of pending helper requests should be personalized by each helper. This means each helper would see a different list of pending helper requests based on their feature, such as help history, age, strengths, etc. The personalized request list is expected to be generated by the recommender system in this system.

2.3.3 FE-3: Helper Matching

Helper matching is the key component in this system as most of the features, such as FE-1: Request Helper and FE-2: Browse Requests, would be based on the helper matching result. The matching process would be driven by the recommender system. In this feature, we would develop a system that can find a list of suitable helpers by giving a helper request (refer to FE-1) and find a list of suitable helper requests by giving a helper (refer to FE-2). The criteria for suitable helpers is that the helper is interested in that request and also knowledgeable in it. We expect that the system would consider a basket of factors, including the request nature, helper’s background, etc., to perform the matching.

2.3.4 FE-4: Helper Search

Besides matching a helper with a user, our app also allows users to browse services that their neighbors provide. A list of services will be shown when users search the keywords by the search bar. Instead of requesting help and matching a helper, users can discover a different kind of service themselves. The results might prioritize in terms of helper popularity, displacement, credit level, promotion, or the search history and request type of user in the past.

2.3.5 FE-5: Chat Room

Chat Room is a feature that allows users to interact with each other. Users and helpers will negotiate the details of the service. After the system matches a user to a helper, chat room features will be enabled and they can talk more about the details and the deals. They can exchange their information such as the exact meet-up location or the fees. It would be a traditional chat room that allows users to send and receive messages.

2.3.6 FE-6: Transaction

Helping people is one of the main goals of our project. However, we also want our users to increase their income with their special ability in their spare time. We also charge 5% of each deal as our service fee.

If a user raises a new post and pays HKD$100 as a remuneration, our platform will cost the user HKD$105 (HKD$100 + 5% of HKD$100).
The platform allows users to pay for the service with credit cards alongside with apple pay and google pay or Faster Payment System (FPS). We would temporarily keep the charge. If the service is done, we would release the remuneration as the post mentioned, to the service provider in a certain period of time (e.g. a weekly automatic direct deposit). This is the workflow of payment and we charge an extra fee to ensure the platform delivers consistently great service.

2.3.7 FE-7: Content Management System (CMS)

In order to make the administration of the app easier, we plan to develop a simple webpage to manage our content inside the application. It provides several features:

- A dashboard for some analytics use:
  - Which service(s) is popular
  - Our income
- Admin page for modifying database data
  - browsing customer data (profile/service/request)
  - UI for checking anything from the database if there are any complaints for customer or helper
- difficult for admin to technical stuff (SQL/GraphQL/API) to modify data
- make it easier for us to demo the application with better UI/UX (without coding to modify the app data)
- Allow admin to handle dispute

2.4 Frontend

This project would run on both iOS and Android devices. We also want the layout to be rendered in a native environment. The most convenient way is using React Native with Expo for making the JavaScript XML (JSX) code to be compiled into native UI elements for the iOS and Android platforms. Briefly introduced to JSX, JSX allows us to write HTML elements in JavaScript and place them in the DOM. Most people find it helpful and as a visual aid when working UI inside JavaScript code and they always use JSX with React to describe what the UI should look like [3].

React.js is a JavaScript library that is independent of React Native and it is used for building user interfaces. Therefore, React.js is typically used for web development. But, if working with React.js, there is actually another library called ReactDOM that adds actual web support. Since React.js library itself without ReactDOM is platform-agnostic. In order to build a web application, React.js need to be in conjunction with
ReactDOM. It means React library does not care about the underlying platform and just provides tools for managing state or building virtual component trees. Then, an extra library like ReactDOM for translating the result that React.js produced to an actual platform like the browser.

Obviously, what React Native basically does is an alternative to ReactDOM for building mobile applications. React Native gives a collection of special React components, which are built-in components, that can be used in JSX code. Also, those components are compiled to native UI elements for iOS and Android platforms. Therefore, React Native will handle the compilation step.

In addition, React Native also exposes native platform APIs, developers can call it and use the feature such as using the device camera in JavaScript code. Overall, React Native is like ReactDOM, it does not connect React to the web platform but iOS and Android platforms instead.

It is clear that the JSX elements will be compiled to native code of mobile applications. But the JavaScript code outside of JSX like the statement, function code, or state management. These logic codes will not be compiled. Basically, it is running on a JavaScript thread, hosted by React Native in the native app that was built. So, React Native spins up a JavaScript process as part of the native app and manages the process for us.

Expo is a collection of tools and services developed around React Native [4]. It has numerous features, the most important of which is that it can get developers to write a React Native app in minutes. It has a managed app development workflow. It means creating a project or tapping into native device functionalities is easy. Expo is a crucial tool for running the React Native app on testing devices and simulators as well as building the app to upload to the app store on iOS or Google play on Android.

2.5 Backend

Comparing different languages and frameworks, python (FastAPI), PHP (Laravel), and JavaScript (Node.js), we choose our tools based on the degree of convenience and the performance of building scale applications. Node.js fulfills both of our criteria and it would be the tool for building our backend.

Node.js is a JavaScript runtime environment that allows us to run JavaScript on server-side [5]. The biggest advantage of using JavaScript on our backend side is that we could align the choice of programming language. We are not required to spend extra time learning another language such as PHP/python in order to start our project, so that all team members could work on different parts of the application without learning
multiple languages. It is easier and more efficient for us to debug the program if we are all on the same page. Node.js also provides asynchronous features which allow servers to process multiple requests at the same time [5]. It could provide better performance compared to PHP which uses blocking I/O. When a non-blocking API is called with an asynchronous function in Node.js, the server is still free to process other API. These features largely increase our server performance.

There are also several frameworks to choose for building web applications. We have examined nest.js and express.js. After comparing what features each framework can provide, we are considering using Nest.js in our project.

Nest.js is a framework that helps to build server-side applications. In our project, Nest.js is used to build API for frontend and communicate with databases. It is built on top of express.js which simplifies the idea of using express.js but also allows us to use other third-party modules made for express [6]. Nest.js is an opinionated framework that has a set of rules for users to follow for avoiding errors. For example, Nest.js uses a model-view-controller design pattern (while express.js is not) which provides proper structure to design our application [7]. It also supports typescript which is JavaScript that supports static type. Using typescript in our application would be more reliable because there would be fewer errors in type and make our development more smooth and more efficient.

Nest.js also comes with a built-in command-line tool that would largely increase our productivity [7]. For example, it can use commands to generate file templates for controllers and providers. We are not required to remember lengthy code. Once we use a certain command, we could start work on the design and the logic of the application. It is much better than spending time writing template code multiple times. It also supports various databases such as mongoDB, postgreSQL. In our project, we plan to use GraphQL for querying APIs since it gets back all the data we need in a single request. The support from Nest.js is one of the deciding factors that we choose as our web framework.

2.6 Recommender System

As a recommendation-based application, users are expected to rely on the system’s recommended offer heavily. Therefore, the performance of the recommender system is critical to this project. Recommender system is a system that provides item suggestions to a user who probably would be interested in that item [8]. In this project, we aim to develop a machine learning based recommender system model to drive the matching system in our application. To achieve that, we would explore different recommender system approaches, such as collaborative filtering, content-based filtering, Graph Neural Network (GNN) based recommendation model, etc.
2.6.1 Collaborative Filtering

Collaborative Filtering (CF) is a very popular approach in recommender systems. The idea behind CF is that similar users should share a similar interest in items [9]. In our setting, “items” in CF would likely be the helper request, and “users” would be our application user. For example, suppose user A has completed a cooking request and user B is similar to user A, the system would consider user B also likely interested in cooking though user B has not completed any cooking requests. One of the popular methods to implement CF is matrix factorization, such as Funk MF, SVD++, etc. Matrix factorization is trying to decompose the user-item interaction matrix to predict the item rating by a user. Nowadays, with deep learning development, He, Xiangnan et. al. proposed Neural Collaborative Filtering (NCF) which is using deep learning techniques, and claims that it can perform better recommendations over the traditional method in 2017 [10]. In this project, NCF could be a potential option model to perform helper recommendations. However, in general, most Collaborative Filtering approaches would suffer from the cold-start problem. The cold-start problem refers to a problem that a new user or a new item does not have any interaction with other users/items [11], and the recommendation quality for those new users/items would become really bad. As for a solution, a hybrid model could be considered.

2.6.2 Hybrid Model

Hybrid Model is a recommender system that combines Collaborative Filtering and Content-based filtering. Content-based filtering is another recommendation approach that would only consider a user’s features/preference to recommend items that share similar features then that user without considering other user data [12]. When a new user does not have enough interaction with the system, using Content-based filtering can alleviate the cold-start problem. Since this project is a new project, we do not have any prior user data. Therefore, the cold-start problem is foreseeable. Wide & Deep model is a hybrid model that was invented by Google in 2016 and it was applied to the Google Play app recommendation. The hybrid design of Wide & Deep model shows that it is able to handle the cold-start problem [13]. Therefore, Wide & Deep model would also be an option model for this project.

2.6.3 Graph Neural Network

Another direction of the recommender system is using graph topology to model user-item interaction and then applying Graph Neural Network technique to perform recommendations. Knowledge Graph is a graph-structured that represents the relationship between different entities [14]. Knowledge Graph is a good way to represent the user-item relationship in this application, as the “item” is a helper request in this project and each helper request could relate to different entities, such as the user, request’s cat-
egory, location, etc. Knowledge Graph Attention Network (KGAT) is a model that is based on Knowledge Graph and attention mechanism to provide item recommendations [15]. So, in this project, KGAT would also be a possible model for the implementation of the recommender system.

2.7 Summary

In summary, this chapter offers the main features of this project and the overall system design, such as system flow, to introduce how users can interact with the system. In order to offer a whole picture of this application. Then, it presents which technologies to be chosen to apply to the system components and justifies the reason for the choice. Finally, it also explores various approaches in the recommender system. In the next chapter, it will present our current project status.

3 Project status

3.1 Introduction

This chapter will report the current project status. First, section 3.2 illustrate the current system architecture in this project aiming to introduce different components of the system, including the frontend side and the backend side. Then, sections 3.3, 3.4, and 3.5 would report the current progress in terms of the recommender system, frontend, and backend. After that, section 3.6 would describe the challenges and limitations of this project. Finally, section 3.7 would address the project schedule and the future plan of this project.

3.2 System Architecture

The following Figure 3.1 illustrates the current system architecture in this project. It presents the interaction between each component in this system.
3.1 System Architecture

From the backend side, there are three main components, including the NestJS backend web server, the Recommender System (RS), and the PostgreSQL database (DB). The interaction between the RS and DB would pass through the NestJS backend web server. All these three components would be developed in the Azure cloud environment.

From the frontend side, users would download and install the community helper application on their local device. Then, when users use this application, the application would only communicate with the NestJS backend web server to perform different operations via the GraphQL endpoint. The user application would not directly communicate with the RS and DB. Another component from the frontend side is Content Management System (CMS). Similar to the user application, CMS also would directly connect to the RS and DB. It also needs to pass through the NestJS backend web server to connect to the backend via the GraphQL endpoint. Therefore, the Node.js backend server is kinda like a gateway that enters the backend system. The NestJS server would take care of all the user requests and then further fetch or call the required data or services, and respond to the user. All the communication between the frontend side and the backend side is traversed via the internet.

3.2.1 PaaS - Azure Cloud Computing Service

Azure is a Cloud Computing service provided by Microsoft, which is one of the Platform as a Service (PaaS) in the market. As shown in Figure 3.1, the whole backend system in this project would deploy in the Azure cloud environment. This project, currently, used the following Azure services,
• Azure app service
• Azure PostgreSQL
• Azure Machine Learning

Specifically, the Azure app service is used as a web server which refers to the NestJS web server in the System Architecture. Azure PostgreSQL is used as the database in this project. Lastly, Azure Machine Learning is used as the recommender system in this system.

3.2.2 CI/CD

Continuous Integration/Continuous Deployment (CI/CD) is a development practice that is applied to this project. In short, CI/CD aims to automate the integration, testing, building, delivery, and deployment phases in software development [16]. This project is trying to apply CI/CD in the development cycle, and, for now, by using GitHub action and a YAML file, we completed a basic CI/CD pipeline for the NestJS server that is able to automatically build and deploys the server to the Azure environment whenever there is a new git commit to the main branch in GitHub.

3.3 Recommender System design

As mentioned in the previous section, this project would develop a recommender system (RS) to provide the matching feature in the system. According to the project schedule, the RS hasn’t been implemented yet. Therefore, this section would mainly discuss the design of the recommender system.

3.3.1 Batch vs Real-time Recommender System

Typically, there are two types of recommender systems (RS), one is batch and another one is real-time. In general, a batch recommender system is easier to implement than a real-time recommender system, since a batch RS only needs to generate new recommended items every day or hour by fetching the new data in the database. That is just a scheduled event for the system to execute which does not need to consider any online changes of the users. However, this characteristic also is a shortcoming of a batch RS, since the 2-phases design of batch RS (offline model-building phase and on-demand recommendation phase) restricted the system to be unable to quickly respond to new/real-time user activity [17]. Therefore, the ultimate goal of the recommender system in this project is to implement a real-time recommender system. However, due to the complexity of a real-time recommender system, this project would first implement a batch recommender system as a baseline system, then migrate to a real-time recommender system. This would depend on the progress of the development.
3.3.2 Tentative workflow of the Recommender System

Figure 3.2 describes the tentative workflow of the recommender system in this project. As discussed in the previous section, this project would first implement a batch recommender system so the following workflow of the RS is a batch RS.

![Workflow of the Recommender System](image)

First, the NestJS server would fetch the users and help-requests incremental data from the database. Note that here it only fetches the data that is newly generated by the user after the last batch. Then the NestJS server would send it to the recommender system. Inside the RS, it would use those new data to update the machine learning model. After that, it would generate a new set of recommended items and then send it back to the NestJS server. When the NestJS server received it, it would send it to the database and store it. The new batch of recommended items is expected to replace the previous batch of recommended items. When the user requests the recommended items, the system is expected to always send the latest batch of recommended items back to the user.

The above workflow is a scheduled loop event, the recommender system would generate a new batch of recommended items after a specific time interval, such as every day or hour. Therefore, the current workflow of the RS is a batch RS.

3.3.3 ML model in the Recommender System

As mentioned in the previous section, the recommender system would be driven by machine learning technology. Under the current status, we plan to use the Wide &
Deep model as the ML model in the recommender system’s first implementation. As introduced in section 2, Wide & Deep is a deep learning model proposed by Google in 2016.

![Figure 3.3: Model structure of the Wide & Deep model](image)

Figure 3.3 illustrated the model structure of the Wide & Deep model. As the name of this model, there are two parts in this model, including wide models and deep models. The wide part is a generalized linear model and the deep part is a Deep neural network. It claims that the model is able to do memorization and generalization [18]. That means, for example, in terms of memorization, it can memorize that older users interested in cooking help-request more. And, in terms of generalization, it can generalize that older users are also interested in education help-request by converting and learning the sparse features of user and item (help-request) to lower-dimensional embedding vectors for generalization.

However, the choice of the ML model might be changed during the development, since there might be some unpredictable technical challenges or we simply find a better model.

### 3.4 Frontend Progress

#### 3.4.1 Community Helper Apps Overview

The layout of community helpers apps are almost halfway done. The apps launch an animated splash screen to users as shown below.
The platform shows all the help requests in the main screen if the user has not been logged in yet. Also, the greeting message is “Hello Anonymous”. Anyone can click the button “Details” in each help request and navigate to the detail screen for a particular request.
Anyone can browse the details of a help request. Since the user has not been logged in, the user is not allowed to accept the help request as a job. In order to accept a help request, users need to login.

Figure 3.7: The main screen if user login  Figure 3.8: The screen to new a help request

Figure 3.7 captures the home screen after a user login. The greeting message includes the display name of the user and the user icon on the top right corner has been changed. More importantly, the help requests shown in the home screen are the requests, which are recommended by the recommender system. For example, the recommender system only delivers two help requests for user1.

If the user wants to create a new help request as a supplicant, the user should click the “+” button and navigate to the screen shown on Figure 3.8. The “+” button only appears if a user login.
This accept button in the detail screen also appears only if the user login (see Figure 3.9). As a helper, they can read the details of a help request and chat with the supplicant, which is the feature implemented later, helper also can accept the help request recommended by the recommender system.

Figure 3.10 is the screen collecting all the help requests that are accepted by the user. Since the user accepts the help request titled “Fix my TV”, this job will be collected on the other screen. This screen mainly records the status of a help request, such as ongoing or done.
As the user accepts the request named “Fix my TV”, the apps navigate to the home screen and the corresponding help request is removed.

### 3.4.2 More about React Native and Expo

All the elements in the apps are written in JavaScript XML (JSX) code with React Native. JSX allows us to write HyperText Markup Language (HTML) elements in JavaScript and place them in the Document Object Model (DOM). React Native framework contains a compiler called Babel and it can transpile JSX to normal JavaScript code. React Native is responsible for making the JavaScript code to be compiled to native UI elements for the iOS and Android platforms.

The UI elements have been compiled as a component in React. React manages the state of the component and the DOM as a virtual DOM, it enables incremental rendering of the virtual DOM. If the state of the components changed, React will re-render it.

So, the state of the help requests list in the home screen has been changed if the user accepts the help request. React re-render the help requests list immediately.
Expo Go is a free, open-source client for testing React Native apps on Android and iOS without needing to build anything locally. After starting the project on the computer by typing “npx expo start”, A QR code will be shown. Using Expo Go to scan the QR code provided, the react native app can run on iOS or Android devices.

### 3.5 Backend Progress

#### 3.5.1 Authentication

Authentication was the first part that was completed in the project. Since the team has no prior experience on using Nest.Js building GraphQL, this part has been considered as a material to learn the tools. There are a lot of resources from the internet that are related to Nest.Js and GraphQL Authentication. Our team had consumed some of it and built the current backend system. During the development, the team spent time to learn the definition and behavior of GraphQL and how Nest.js utilize module, resolver, and service to build GraphQL API.

The authentication system is now able to use GraphQL mutation method to sign up and login a user. In figure 3.13, when the login mutation method is sent to the backend server with the variable of username and password, the server would return the user object in JSON format which looks like figure 3.14. The sign up mutation method also works similarly to the login mutation.
3.5.2 Security

Our team projects that our platform would have a lot of user engagement. Therefore, security is one of the aspects that we want to complete in the early phase of development. The backend system currently has two major security features. They are the access token and hash password.

For the access token, the team has considered whether to use JWT or Sessions for the session management for the user requesting the API and the authentication status on the frontend application. JWT was chosen in the final implementation as it is easier for frontend managing the user session. The benefit of JWT over sessions is that JWT generates a token that can decode into a JSON that stores user information and the expiration constraint in Epoch time. The frontend application is only required to store the token which does not need to rely on the server to determine whether the session expired. It also needs a secret string on encoding the data which provides extra safety on the user information.
For the hash password part, our team has implemented the sign up mechanism with the “bcrypt” npm package. The package provides the functionality to encrypt the password to garbled text. This ensures that the user password would not be exposed to the developer or admin of our application.

### 3.5.3 Matching

One of the main features of the application is the matching system. The matching system is currently a prototype version. The team is still designing the matching algorithm with the technique of machine learning. Hence, we are using a random matching approach in the current system. There are several modules inside the backend system that involve the matching mechanism. They are Help Request, Recommender, and Matching modules. The team has implemented CRUD operation for help requests and matching respectively. The Recommender module is an internal module for the matching mechanism only. Thus, there is no CRUD functionality for the module.

The above modules are used to build the pipeline to give suggestions to the user. Figure 3.16 shows how the job is suggested to the user that the team has implemented. When a user requests for help, the help request will pass to the recommender modules. The recommender will randomly send the job to five users excluding the user himself which the matching results will save to the matching table.
3.5.4 Preparation for future development

The team has implemented some modules that are currently undisplayable as the frontend side has not implemented the related features. It is the service category. In our business model, each user would have several skills to provide. The module has been implemented and is able to perform CRUD action on the database. It is planned to add as criteria inside the matching mechanism. However, this step would push to the next phase as the time constraint on the first presentation.

3.6 Challenge and Limitation

One of the potential challenges in this project is the lack of system interaction data to develop the recommender system. Since this project is a new project which means there are no prior data for us to train or build the recommender system. This is as known as the cold-start problem mentioned in the previous section. In short, the number of users and help-requests interaction records should have a certain amount, otherwise, the recommendation quality for those new users or items would become really bad. Regarding this potential problem, our team would try to explore more different approaches, such as collecting some similar datasets or generating artificial data, in order to minimize the negative effect of that problem.

3.7 Project Schedule and Future plan

In the current period, we have implemented a prototype on our frontend, backend application, and recommender system. The team is currently working on the recommender system design. It is expected to start the actual implementation in January. In the coming few months, the team would continue to implement the rest of the features and integrate the machine learning algorithm into the backend system. It is expected
our implementation would complete before April. During April, the team would focus on debugging the application and doing UAT testing. For example, testing how the matching performance works and user feedback on the application. The application would be presentable in May. Table 3.1 shows the detailed project plan for the coming few months.

Table 3.1: Project Schedule

<table>
<thead>
<tr>
<th>Time Periods</th>
<th>Tasks</th>
<th>Status</th>
</tr>
</thead>
</table>
| 2022 Sep           | • Detailed project plan  
                       | • Project web page  
                       | • Project background research | Completed |
| 2022 Oct           | • UI/UX design  
                       | • Database design  
                       | • Recommender system design | Completed |
| 2022 Nov - Dec     | • Basic implementation of frontend  
                       | • Basic implementation of backend  
                       | • Basic implementation of Recommender system | Completed |
| 2023 Jan           | • First presentation  
                       | • Detailed interim report  
                       | • Recommender system design | Completed |
| 2023 Feb - Mar     | • Implement Recommendation System  
                       | • Implement search function of helpers  
                       | • Implement transaction system  
                       | • Implement content management system  
                       | • Implement chat room | In-Progress |
| 2023 Apr           | • UAT Testing and debugging  
                       | • Final presentation  
                       | • Finalized tested implementation  
                       | • Final report | Pending |
| 2023 May           | • Project exhibition | Pending |

The immediate next step of the project is to finalize the recommender system design.
After that, we would start to implement a workable recommender system and then integrate it into the current existing backend system. We aim to complete that before Mid February 2023.

4 Conclusions

In conclusion, this project aims to create a multi-platform application that links up the community. Users can become a helper to help other people in the community or seek help from other helpers in the community. In the end, we would use React Native to develop the frontend application, NestJS and PostgreSQL to construct the backend component as well as a recommender system to provide the matching feature. Our application has a different system design or philosophy compared to other existing solutions. Therefore, we believe this project would provide a unique service to the market.

Up until now, our team finish the first presentation and complete the deliverables of phase 2, such as the preliminary implementation and detailed interim report. A workable system is implemented, including the mobile app frontend, and backend system, which the basic function completed, such as login, creating help-request, etc. The system is currently deployed on the Azure cloud platform. Finally, our next step in this project is to implement the recommender system and the rest of the system’s main features.
References


