Project Report

**HKUST**

**CAMPUS-TRANSACTION**

**APPLICATION**

Proposed by

CHEN, Siyu 20678097 schendf

ZHENG, Yueyan 20677469 yzhengbl

Supervised by

Professor

MUPPALA, Kumaraswamy R Jogesh

Submitted in fulfillment of the requirements for COMP 4521

in the

School of Engineering

The Hong Kong University of Science and Technology

2022-2023

Date of submission: October 31, 2022

Catalog

[*1.* *Introduction* 3](#_Toc121436850)

[*2.* *Design and Implementation* 4](#_Toc121436851)

[2.1. Requirements Analysis 4](#_Toc121436852)

[2.1.1. Server and database 5](#_Toc121436853)

[2.1.2. Dependencies 7](#_Toc121436854)

[2.2. Design 8](#_Toc121436855)

[2.2.1. Server data specification (MongoDB) 8](#_Toc121436856)

[2.2.2. Server API specification 10](#_Toc121436857)

[2.2.3. Internal data specification 22](#_Toc121436858)

[2.2.4. Application Activities and Fragments 24](#_Toc121436859)

[2.3. Implementation 28](#_Toc121436860)

[2.3.1. Server API in python 28](#_Toc121436861)

[2.3.2. Server API in Kotlin 29](#_Toc121436862)

[2.3.3. API function call in SDK 30](#_Toc121436863)

[2.3.4. Permission and File Provider 31](#_Toc121436864)

[*3.* *Testing and Evaluation* 32](#_Toc121436865)

[3.1. Postman 32](#_Toc121436866)

[3.2. Studio 3T 32](#_Toc121436867)

[3.3. FileZilla and Xshell. 33](#_Toc121436868)

[3.4. Testing 34](#_Toc121436869)

[3.5. Evaluation 34](#_Toc121436870)

[*4.* *Conclusions* 36](#_Toc121436871)

[*5.* *References* 37](#_Toc121436872)

# *Introduction*

A brief introduction to your application idea. State the goals of the project.

So far, there are 2 typical ways for HKUST students (especially mainland Chinese students) to sell and purchase second-hand goods. One is through WeChat groups, and another is through a WeChat mini-application called Stardust. However, both require users to read each post and search by themselves. And it is also not convenient to add WeChat friends first to chat privately. Besides, sellers or buyers always lack some important information such as intentional price or commodities parameters which makes the transaction hard to continue smoothly. No notification is also another problem that disables sellers or buyers to reply to messages rapidly.

So inspired by that, we plan to develop a campus second-hand transaction application.

# *Design and Implementation*

Detailed system description and design and implementation details.

In particular, this section should contain:

Important components in your application, including a description of all the activities, services, content providers, and broadcast receivers that you used in your application and how they are interrelated.

Source of the data used in the application, or a description of how you will store the data for the application.

Specific features like multimedia, graphics, location, maps, and sensors that you made use of in your project and how they were used.

## Requirements Analysis

The basic functionality, as well as objectives of our project, mainly contains:

* Create a new account.
* Log in through password. (Forget password-> reset password)
* Show posts by favorite field or keyword.
* Create a new post (image by shoot or from the album).
* Add comments to a post.
* Private messages with other users. (Message list)
* See detailed information on other users.
* See your own or another user’s posts.
* Set personal information

## Server and database

We use MongoDB to store all data, including user information, post content, and private messages. Detailed data specifications can be found in [2.2.1](#_Server_data_specification). The application can get access to the server data only through server API. A detailed API specification can be found in [2.2.2](#_Server_API_specification). The application may need to store some important data. Detailed internal data specifications can be found in [2.2.3](#_Internal_data_specification). And the application fragments and their navigation can be found in [2.2.4](#_Application_Fragments).

图形用户界面, 文本, 应用程序

描述已自动生成

Figure 1: Amazon Web Services (AWS)

Elastic Compute Cloud (EC2)

Linux (Ubuntu) system instance.

We use **Nginx + Gunicorn** as the proxy.

We built the **MongoDB** database in an Amazon Web Services (AWS) Elastic Compute Cloud (EC2) Linux (Ubuntu) system instance as shown in Fig.1. The application SDK is written in Kotlin while the server API will be written in python with the library Flask. Flask is a lightweight web-based application framework written in Python. Data transferred between the server and application will be in JSON format. Images will be encoded and decoded through the base64 method. The Linux system runs the API python code listening to any API URL call. The design is a one-way connection which means the server cannot send information to the application and the application needs to call the corresponding API to get or upload the necessary data from the server.

图形用户界面, 应用程序, 表格, Excel

描述已自动生成

Figure 2: A sample Studio 3T connected to the MongoDB database.

Besides, we can use the Studio 3T application to check the data inside the MongoDB database, as shown in figure 2.

We have tested the send validation code function by creating a free email address [schendf@163.com](mailto:schendf@163.com) to send validation to the target email address through the SMTP server: smtp.163.com.

## Dependencies

Note: Can be found in build.gradle (Module).

**What we learned (brief):**

lifecycle, retrofit2, material, navigation-fragment, data binding, view binding, recyclerview, constraint-Layout, live-data, GridLayout.

**What’s new (function):**

Gson: serialize JSON file.

Glide: load Uri or Url image into view.

Bubble-picker: open-source project for bubble-like click picker.

Bubble-view: for communication bubble-like view.

NestedScrollView: for a scrollable view with multiple recyclerview.

SwipeRefreshLayout: for sliding down to refresh

CircleImageView: for a circle head portrait.

## Design

**This part mainly illustrates the final detailed design. It is different from the proposal version since we found a lot of bugs in the earlier version.**

## Server data specification (MongoDB)

**This server table stores the basic information of application users.**

|  |  |  |  |
| --- | --- | --- | --- |
| **UserInfos** | | | |
| **Field** | **Format** | **Detail** | **Source** |
| CreateTime | date | Last create time | Server auto-generated |
| UpdateTime | date | Last update time |
| EmailAddress | string | E.g. schendf@connect.ust.hk | Upload by the application while new users log in or modified. |
| ValidCode | Int | E.g. 206780 |
| CodeTime | string | Last time to send ValidCode. |
| Password | string | E.g. 12345678 |
| StudentID | string | E.g. 20678097 |
| FirstName | string | E.g. SIYU |
| LastName | string | E.g. CHEN |
| NickName | string | E.g. Alex |
| Birthday | string | E.g. 2001-04-27 |
| Gender | boolean | 0 for male; 1 for female; empty for unknown |
| Profile | string | Personality Profile |
| Region | string | E.g. Hong Kong (+852) |
| School | string | Choose from a list of labels: CSE, CPEG, ECE, etc. |
| PhoneNumber | string | 51020772 |
| HeadPortrait | base64 string | E.g. A standard (width, height) size image |
| FavoriteFields | array of strings | Choose a list of labels: e.g. Digital, Fashion, Makeup, Game, and so on. |

**This server table stores the content of posters published by users, including descriptions, images, and possible comments.**

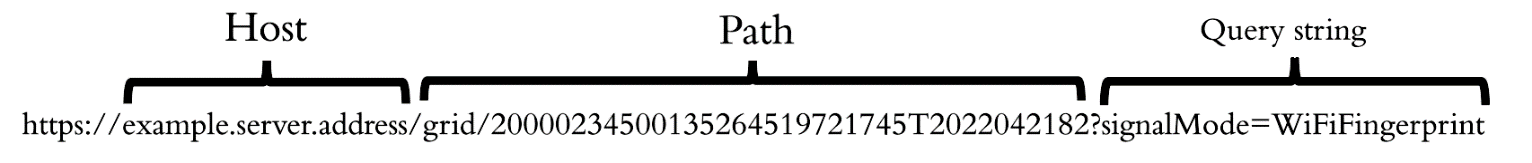
|  |  |  |  |
| --- | --- | --- | --- |
| **Posts** | | | |
| **Field** | **Format** | **Detail** | **Source** |
| CreateTime | date | Last create time. |  |
| UpdateTime | date | Last update time. |
| PostOwner | EmailAddress | Refer to table User\_Info. | Provided by the application while the application call server API. |
| Title | string | Title. |
| Text | string | Detail descriptions. |
| Images | list of base64 strings | The images for goods. |
| Comments | list of { Commenter, NickName, CommentTime, Text} | E.g. { schendf@connect.ust.hk, Alex, 2022-10-31, “it is so beautiful😀”} |
| Fields | list of strings | Choose a list of labels: e.g. Digital, Fashion, Makeup, Game, and so on. |
| Auction | boolean | Whether it is an auction or not. |
| LostFound | boolean | Whether it is a Lost and Found or not. |
| Deleted | boolean | Whether it is deleted by its owner or not. |

**This server table stores the content of unread messages from both chatroom users.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Messages** | | | |
| **Field** | **Format** | **Detail** | **Source** |
| CreateTime | date | Last create time |  |
| UpdateTime | date | Last update time |
| Sender | EmailAddress | Sender | Provided by the application while the application call server API. |
| Receiver | EmailAddress | Receiver |
| Contents | list of {Text, Image, Time} | Several messages in time order. Possible that only exist one of them. |

## Server API specification

This is the Server API structure. Each API Uniform Resource Locator (URL) consists of a scheme, a host, a path, and an optional query string.



* Scheme:

The scheme is HTTPS by default.

* Host:

The host is the address of the server.

* Path:

The path is an identifier of the service. The path of each service is defined in this standard.

* Query string:

The query string is one of the methods to send input parameters of the API to the server, which is GET mothed.

There are two main ways for a user to upload and get data from the server: GET and POST. We decided to use POST more because POST enables higher data security than GET. But for some common data that does not include user information, we can use get.

**Email Validation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /email-validation?EmailAddress={EmailAddress}&InputCode={InputCode} | | | |
| If only EmailAddress is provided, by clicking the send validation code button, the app will ask the server to send a validation code to the target email address. Return acknowledge message.  If both EmailAddress and ValidCode are provided, the server will check if the email address and code match. It is matched, returns acknowledge message, and the app will jump to create account page. Else, return an error. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| EmailAddress | | string | yes | Email address. First checks if is in email format, then sends the validation code. |
| InputCode | | int | no | Integer in length 6 |
| Response | | | | |
| {  “Success”: True/False  “Error”: Error  } | | | | |
| Implementation | | | | |
| from flask import Flask, request  app = Flask(\_\_name\_\_)  @app.route('/email-validation', methods=[GET])  def EmailValidation(EmailAddress, InputCode):  EmailAddress = request.args.get(“EmailAddress”) or “N/A”  InputCode = request.args.get(“InputCode”) or “N/A”  …  return return\_data | | | | |

**Set Reset Password**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /set-reset-password | | | |
| Input | {  “EmailAddress”: EmailAddress  “Password”: Password  “NewPassword”: NewPassword  } | | | |
| Login in through an existing email address. Return an error if the email does not exist or the password is wrong. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| EmailAddress | | string | yes | Email Address |
| Password | | string | yes | Password |
| NewPassword | | string | yes | New Password |
| Response | | | | |
| {  “Success”: True/False,  “Error”: Error,  } | | | | |
| Implementation | | | | |
| @app.route('/reset-password’, methods=['POST'])  def SetResetPassword():  EmailAddress = request.form.get(‘EmailAddress’)  …  return return\_data | | | | |

**Create New User (Account)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /create-account | | | |
| Input | {  “EmailAddress”: EmailAddress,  “StudentID”: StudentID,  “FirstName”: FirstName,  “LastName”: LastName,  “NickName”: NickName,  “Birthday”: Birthday,  “Gender”: Gender,  “Profile”: Profile,  “Region”: Region,  “School”: School,  “PhoneNumber”: PhoneNumber,  “HeadPortrait”: HeadPortrait,  “FavoriteFields”: FavoriteFields,  } | | | |
| After Email Validation, the user needs to set an account password. If the password has already been set, it means resetting it with a new password. The same for other information. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| EmailAddress | | string | yes | Email Address |
| StudentID | | string | no | StudentID |
| FirstName | | string | no | FirstName |
| LastName | | string | no | LastName |
| NickName | | string | no/default | NickName |
| Birthday | | string | no | Birthday |
| Gender | | boolean | no | Gender |
| Profile | | string | no/default | Profile |
| Region | | string | no/default | Region |
| School | | string | no | School |
| PhoneNumber | | string | no | PhoneNumber |
| HeadPortrait | | base64 string | no | HeadPortrait |
| FavoriteFields | | array of strings | no | FavoriteFields |
| Response | | | | |
| {  “Success”: True/False,  “Error”: Error,  } | | | | |
| Implementation | | | | |
| @app.route('/create-account', methods=['POST'])  def CreateAccount():  EmailAddress = request.form.get(‘EmailAddress’)  …  return return\_data | | | | |

**Log in (Account)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /login | | | |
| Input | {  “EmailAddress”: EmailAddress  “Password”: Password  } | | | |
| Login in through an existing email address. Return an error if the email does not exist or the password is wrong. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| EmailAddress | | string | yes | Email Address |
| Password | | string | yes | Password |
| Response | | | | |
| {  “Success”: True/False,  “Error”: Error,  } | | | | |
| Implementation | | | | |
| @app.route('/login', methods=['POST'])  def Login():  EmailAddress = request.form.get(‘EmailAddress’)  …  return return\_data | | | | |

**Request Detailed User Information**

|  |  |  |  |
| --- | --- | --- | --- |
| GET | /get-user-info?EmailAddress={EmailAddress} | | |
| Request detailed information about certain users. | | | |
| Parameters | | | |
| Name | Data type | Mandatory | Description |
| EmailAddress | string | yes | Email address |
| Response | | | |
| {  “EmailAddress”: EmailAddress,  “FirstName”: FirstName,  “LastName”: LastName,  “NickName”: NickName,  “StudentID”: StudentID,  “Birthday”: Birthday,  “Gender”: Gender,  “Profile”: Profile,  “Region”: Region,  “School”: School,  “PhoneNumber”: PhoneNumber,  “HeadPortrait”: HeadPortrait,  “FavoriteFields”: FavoriteFields,  “Success”: True/False,  “Error”: Error,  } | | | |
| Implementation | | | |
| @app.route('/get-user-info', methods=[GET])  def GetUserInfo():  EmailAddress = request.args.get(“EmailAddress”) or “N/A”  …  return return\_data | | | |

**Send a Private Message**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /send-message | | | |
| Input | {  “Sender”: EmailAddress,  “Receiver”: EmailAddress,  “Content”:  {  “Text”: Text,  “Image”: Image,  },  } | | | |
| Send one single private message to one target user. The server will add a timestamp to the message. Return acknowledges or error | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| Sender | | EmailAddress | yes | Send messages. |
| Receiver | | EmailAddress | yes | Receive messages. |
| Content | | {“Text”,” Image”} | yes | The message could be Text or Images. |
| Response | | | | |
| {  “Success”: True/False,  “Error”: Error,  } | | | | |
| Implementation | | | | |
| @app.route('/send-message', methods=[POST])  def SendMessage():  EmailAddress = request.form.get(‘EmailAddress’)  …  return return\_data | | | | |

**Get Private Messages**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /get-messages | | | |
| Input | {  “EmailAddress”: EmailAddress,  “Password”: Password,  “Sender”: Sender  } | | | |
| Request all messages sent to this user. Possibly for many different senders or a single user. Need a password to ensure data security. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| EmailAddress | | string | yes | Email Address |
| Password | | string | yes | Password |
| Sender | | string | no | Another user’s email address |
| Response | | | | |
| {  “Data”:  [{  “Sender”: EmailAddress,  “NickName”: NickName,  “HeadPortrait”: HeadPortrait,  “Contents”:  [{  “Text”: Text,  “Image”: Image,  “Direction”: True/False  “CreateTime”: CreateTime,  }, …],  }, …]  “Success”: True/False,  “Error”: Error,  } | | | | |
| Implementation | | | | |
| @app.route('/get-message', methods=[POST])  def GetMessages():  EmailAddress = request.form.get(‘EmailAddress’)  …  return return\_data | | | | |

**Create New Post**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /new-post | | | |
| Input | {  “PostOwner”: EmailAddress,  “Title”: Title,  “Text”: Text,  “Auction”: True/False  “LostFound”: True/False  “Price”: Price  “Fields”:  [{  Field  }, …]  “Images”:  [{  Images  }, …]  } | | | |
| Which user sends a new post with a title, text, and a list of images. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| PostOwner | | EmailAddress | yes | Refer to table Post in 2.1 |
| Title | | string | yes |
| Text | | string | no |
| Images | | Base64 string | no |
| Price | | Int | no |
| Fields | | List of strings | no |
| Auction | | boolean | No |  |
| LostFound | | boolean | no |  |
| Response | | | | |
| {  “Success”: True/False,  “Error”: Error,  } | | | | |
| Implementation | | | | |
| @app.route('/new-post', methods=[POST])  def NewPost():  EmailAddress = request.form.get(‘EmailAddress’)  …  return return\_data | | | | |

**Delete Post**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /delete-post | | | |
| Input | {  “PostOwner”: EmailAddress,  “Password” Password,  “PID”: PID,  } | | | |
| Only the post owner can delete the PID. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| PostOwner | | EmailAddress | yes | Email address |
| Password | | string | yes | Password |
| PID | | string | yes | PID |
| Response | | | | |
| {  “Success”: True/False,  “Error”: Error,  } | | | | |
| Implementation | | | | |
| @app.route('/delete-post', methods=[POST])  def DeletePost():  EmailAddress = request.form.get(‘PostOwner’)  …  return return\_data | | | | |

**Get Own Posts**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /user-posts?EmailAddress={EmailAddress} | | | |
| Get all the posts for a certain user. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| EmailAddress | | EmailAddress | yes | Email address |
| Response | | | | |
| {  “Posts”:  [{  “PID”: PID,  “PostOwner”: PostOwner,  “CreateTime”: CreateTime,  “Title”: Title,  “Text”: Text,  “Auction”: True/False,  “LostFound”: True/False,  “Images”:  [{  Image  }, …],  “Comments”:  [{  “Commenter”: Commenter,  “NickName”: NickName,  “CreateTime”: CreateTime,  “Text”: Text  }, …]  },…]  “Success”: True/False,  “Error”: Error  } | | | | |
| Implementation | | | | |
| @app.route('/user-posts', methods=[GET])  def UserPosts():  EmailAddress = request.agrs.get(‘EmailAddress)  …  return return\_data | | | | |

**Add Comment to Post**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| POST | /post-comment | | | |
| Input | {  “EmailAddress”: EmailAddress,  “PID”: PID,  “Text”: Text  } | | | |
| Add comment to the target post. The timestamp will be added | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| Commenter | | string | yes | Email address |
| PID | | string | yes | PID |
| Text | | Text | yes | text |
| Response | | | | |
| {  “Success”: True/False,  “Error”: Error  } | | | | |
| Implementation | | | | |
| @app.route('/post-comment', methods=[POST])  def PostComment():  EmailAddress = request.form.get(‘EmailAddress’)  …  return return\_data | | | | |

**Request Posts (Recommendation System)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GET | /get-posts?EmailAddress={EmailAddress}&Keyword ={Keyword} | | | |
| Find some posts with possible keywords.  If the keyword is not provided, the random post will be returned. | | | | |
| Parameters | | | | |
| Name | | Data type | Mandatory | Description |
| EmailAddress | | string | yes | Email Address |
| Keyword | | string | no | keyword |
| Response | | | | |
| {  “Posts”:  [{  “PID”: PID,  “PostOwner”: PostOwner,  “CreateTime”: CreateTime,  “Title”: Title,  “Text”: Text,  “Auction”: True/False,  “LostFound”: True/False,  “Images”:  [{  Image  }, …],  “Comments”:  [{  “Commenter”: Commenter,  “NickName”: NickName,  “CreateTime”: CreateTime,  “Text”: Text  }, …]  },…]  “Success”: True/False,  “Error”: Error  } | | | | |
| Implementation | | | | |
| @app.route('/get-posts', methods=[GET])  def GetPosts():  Keyword = request.args.get(‘Keyword) or ‘N/A’  …  return return\_data | | | | |

## Internal data specification

**This part is to show some possible data classes we will build for android applications.**

**Data class UserInfo**

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Default value** |
| EmailAddress | string | required |
| Password | string | null |
| StudentID | Int | null |
| FirstName | string | null |
| LastName | string | null |
| NickName | string | null |
| Birthday | string | null |
| Gender | boolean | null |
| Profile | string | null |
| Region | string | “Hong Kong” |
| School | string | null |
| PhoneNumber | Int | null |
| HeadPortrait | base64 string | null |
| FavoriteFields | List of strings | Only required for the request, null for respond |

**Data class Message**

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Default value** |
| Sender | EmailAddress | required |
| Receiver | EmailAddress | Only required for the request, null for respond |
| Content | list of Content | required |

**Data class Content**

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Default value** |
| Text | string | null |
| Image | Base64 string | null |
| CreateTime | Date string | Only required for respond, null for request |

**Data class Post**

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Default value** |
| PID | string | Only required for respond, null for request |
| PostOwner | EmailAddress | required |
| Title | string | required |
| Text | string | required |
| Fields | List of strings | null |
| Images | List of base64 strings | null |
| Comments | list of Comment | Only required for respond, null for request |
| Auction | boolean | false |
| LostFound | boolean | false |

**Data class Comment**

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Default value** |
| EmailAddress | string | required |
| NickName | string | Only required for respond, null for request |
| Text | string | required |
| CreateTime | date string | Only required for respond, null for request |

## Application Activities and Fragments

**This part is to name all activities and fragments that will be used in the application.**

Navigation between fragments may have popUpTo and popUpInclusive properties. And bundle data and navigation-safe-args are used to transfer data from loginFragment to activity\_home.

**Activity\_main:**

Mainly for functions such as Login, Create Account, Forget Password.

* startFragment.xml
* createAccountFragment.xml
* loginFragment.xml
* resetPasswordFragment.xml

图形用户界面, 应用程序, Teams

描述已自动生成

Figure 1: activity\_main

**Activity\_home:**

Use BottomNavigationView with four main navigations (fragments). The all share the same ViewModel which is UIViewModel.

* mainNavigation.xml
  + bannerDetailFragment.xml
* postsNavigaiton.xml
* messageNavigation.xml
  + messageDetailFragment.xml
* userInfoNavigation.xml
  + personalCenterFragment.xml
  + userPostFragment.xml
    - postDetailFragment.xml
    - otherPersonalCenterFragment.xml
  + aboutFragment.xml
  + resetPasswordFragment.xml
    - photofragment.xml

图形用户界面, 应用程序, Teams

描述已自动生成

Figure 2: activity\_home (left) and activity\_favorite(right)

**Activity\_favorite:**

A bubble picker that can only be implemented in activity (fragment error)

图表, 气泡图

描述已自动生成

Figure 3: Activity\_favorite

**Other fragments:**

* Item\_post.xml
* Item\_chat.xml
* Item\_comment.xml
* Item\_message.xml
* Item\_picture.xml

Are defined for the corresponding adapter and recycleView. All adapters are more or less the same. While the onClickListener() is different to achieve.

Note: Detail activity and fragment information can be found under the res -> navigation folder of the project code.

文本

描述已自动生成

## Implementation

**This part demonstrates the implementation details.**

## Server API in python

Mainly by importing flask and pymongo.

**电脑屏幕截图

描述已自动生成**

## Server API in Kotlin

Mainly by retrofit2 and Gson. Similar to what we learned in Week 10.

文本

描述已自动生成

Figure 4: Dependency for network communication.

文本

描述已自动生成

Figure : Server API in Kotlin.

**Note:** Details can be found under api folder. The data class definition for request and response can also be found there.

文本

描述已自动生成

## API function call in SDK

Since all API calls are **Asynchronous,** we use Livedata to observe when the application receives the response feedback.

**文本

描述已自动生成**

Figure : API function call in UIViewModel.

**Note:** Details can be found in UIViewModel. Sorry to write the annotations in Chinese.

## Permission and File Provider

Since we also need functions, such as taking a picture or choosing a picture from an album, we need to ask for permission and have a file provider.

**文本

描述已自动生成**

Figure : Permissions added by us.

**文本

描述已自动生成**

Figure : provider added by us.

**Note:** Details can be found in AndroidManifest.xml

## Other Implementations

For the rest of effort we paid, like adapters, fragments code, we can find many sample codes online. All we need to do is to understand why they do so and transfer the code from Java into Kotlin. And then test and debug over and over again. Most of the work focuses on onCreate () and onCreateView (), onClickListener () and so on.

One big challenge we faced is to use the camera. Since the function startActivityForResult () is deprecated in our version. Find an alternative way to call the camera cost us a lot of time and efforts. startActivity () might be an alternative, but it does not have a return value, so the application doesn’t know when and where the phoneFragment stores the picture Uri.

# *Testing and Evaluation*

Describe the strategies and techniques you used to test your application.

## Postman

We use the software postman to test the server API. This software is quite useful and helps a lot.

**图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成**

Figure : All server API call in Postman

## Studio 3T

We use this software to check if the application uploads new messages or posts successfully. In addition, I use this software to set the Text-Index for the table Posts. We can also delete tables or certain instances through this software.

**表格

描述已自动生成**

Figure : Table Posts are shown in Studio 3T

## FileZilla and Xshell.

We use these two software to upload a new version of the server API to the server and let the server start listening.

**图形用户界面, 文本

描述已自动生成**

Figure : API.py file in server shown by FileZilla



Figure : Restart the server API by Xshell

## Testing

We have uploaded many posts, and messages by sample users to the database to test whether our application works well. Besides, logcat and toast.show() are also useful when doing debugging.

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成图形用户界面, 文本

中度可信度描述已自动生成 手机屏幕的截图

中度可信度描述已自动生成

## Evaluation

We compared our application to some of the mainstream second-hand transaction applications and believed that we have designed and implemented most of the main functions for such kind of application. We also learned a lot of ideas from fellow students at other universities.

The server also crashed several times. Fixing requires a lot of searches and reading.

The drawback of our application is that we still lack a nicer and more user-friendly UI interface. This is due to the time limitation. Besides, no local database is probably another problem.

# *Conclusions*

Briefly state what results you expect from your project.

**Application MongoDB Server**

图示

描述已自动生成 图示

描述已自动生成

It would be quite challenging for us to design a web-based application and run it on a physical phone. Data synchronization and security is a big stuff for us to consider. We both need to design the frontend SDK and backend server API to have better stability for our application and database.

Since we are using the latest Kotlin and compileSdk versions, some of the functions that used to be written in Java are no longer available to us. Finding a new way to achieve our goals is a time-consuming but valuable experience.

All in all, we believe that we have successful fulfill our target application idea. And most importantly, learn a lot of knowledge about the android application in this course.

# *References*

Give references to any material/websites/books etc. relevant to your proposal.

Similar App (open-source, exclude market survey above):

<https://github.com/search?p=2&q=Campus-transaction&type=Repositories>.

MongoDB (the database we plan to use):

<https://segmentfault.com/a/1190000022435473>.

Recommendation system:

<https://github.com/MrQuJL/product-recommendation-system>.

Email validation:

<https://github.com/search?l=Python&q=email+validation&type=Repositories>.

Flask routing:

<https://flask.palletsprojects.com/en/2.2.x/quickstart/#routing>.

Base64 encoding and decoding:

<https://stackabuse.com/encoding-and-decoding-base64-strings-in-python/>.

Studio 3T:

<https://studio3t.com/>.

Keyword search:

<https://www.mongodb.com/docs/manual/core/index-text/>

banner

<https://github.com/youth5201314/banner>

bubble picker

<https://github.com/hantrungkien/Bubble-Picker>

MongoDB text index

<https://www.mongodb.com/docs/manual/core/index-text/>

Retrofit Get and Post

<https://cloud.tencent.com/developer/article/1130060>

TimeUnit Count Down

<https://blog.51cto.com/u_15360378/3808309>