

**HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**School of Information and communications technology**

# **GRADUATION THESIS**

## **PhotoHub - A system for pose suggestion and photography services**

**HÀ QUÝ DŨNG**

dung.hq150661@sis.hust.edu.vn

**Program: Global ICT**

**Supervisor:**

Ph.D Trịnh Tuấn Đạt

**Department:**

Software Engineering

**Institute:**

School of Information and communications  
technology

**Hanoi, 06/2020**

# Declaration

Student name: Hà Quý Dũng

Phone number: 0971872297

Class: ICT – K60

Email : dung.hq150661@sis.hust.edu.vn

Program : Global ICT

I – *Hà Quý Dũng* – hereby declare that this graduation thesis is my original work under the supervision of Ph.D Trịnh Tuấn Đạt. The results mentioned in this project is my own achievement, no copy under any other works. All references in this project including figures, tables, statistics, and quotes are fully and exactly documented in References section. I take full responsibility for even one copy that violates that university's rules.

*Hanoi, 20<sup>th</sup> June, 2020*

Author

*Hà Quý Dũng*

# Acknowledgment

I would like want to thank sencerely to my lecturer – Ph.D. Trịnh Tuấn Đạt – lecturer of Software Engineering Department – School of Information Communication Technology – Hanoi University of Science and Technology, who guided and assisted me during process of performing graduation thesis. His teaching has motivated and inspired me and lots of other students.

I am so thankful for all of the lecturers at Hanoi University of Science and Technology, specially the lectures in Information and Communications Technology institute for teaching me priceless knowledge and skills for the last five years of study. The knowledge and skills will be so useful for me in my life and my future career.

Lastly, I would like to express my sincere appreciation to my parents friends for encouraging and supporting me throughout the study.

# **Abstract**

In recent years, thanks to advanced technology, a large amount of smart devices have been introduced every year. The camera functions of those phones become better and better. However, those phones can not replace professional cameras. A DSLR or mirror-less cameras along with professional photography skills are indispensable capture our worthy moments. Professional photographers are always ready to stay everywhere to help us do that. For photography purpose before, we just need to look for the studio, professional photographers and exchange wishes, aspirations so that they can help us to get beautiful photographs. However, if the demand to take photos happens suddenly, perhaps we can only use the smartphone to save those moments, whether the photos created are beautiful or not.

The system approaches users who want to take the best photos and amateur photographers. For normal users, they can search for nearby photographers within a certain radius from their current location or by region address and connect with them. It is very helpful, especially suitable for accidental needs without any preparation in advance, or when customers want to have the most worthy photographs from the photographer instead of using their smartphone. For photographers, they can register locations with the needed so that customers can find out them. In addition, the system also allows users to search for different photographs poses fitting some circumstances, creating and managing photograph collections, sharing collections to support the photography process to become more convenient.

# Contents

<b>Declaration .....</b>	<b>ii</b>
<b>Acknowledgment .....</b>	<b>iii</b>
<b>Abstract .....</b>	<b>iv</b>
<b>Contents.....</b>	<b>v</b>
<b>List of figures .....</b>	<b>ix</b>
<b>List of tables .....</b>	<b>xii</b>
<b>List of acronyms .....</b>	<b>xiii</b>
<b>Chapter 1 Introduction.....</b>	<b>1</b>
1.1 Problem Description .....	1
1.2 Objective and Scope .....	2
1.3 Solution Approach .....	2
1.4 Thesis outline .....	3
<b>Chapter 2 Survey and requirements analysis.....</b>	<b>5</b>
2.1 Survey .....	5
2.2 Overall Description.....	6
2.2.1 Overall Use case Diagram .....	6
2.2.2 Decomposition of use case “Search photographer” .....	8
2.2.3 Decomposition of use case “Connect to photographer” .....	9
2.2.4 Decomposition of use case “Manage collection” .....	10
2.2.5 Decomposition of use case “Manage individual collection” .....	11
2.2.6 Decomposition of use case “Manage message” .....	12

2.2.7 Decomposition of use case “View photo” .....	13
2.2.8 Decomposition of use case “Manage user” .....	14
2.2.9 Decomposition of use case “Manage photo” .....	15
2.2.10 Decomposition of use case “Manage tag” .....	16
2.2.11 Decomposition of use case “Scan QR code for collection” .....	17
2.2.12 Business processes .....	17
2.3 Specific requirements. ....	21
2.3.1 Specification of use case “View photo” .....	21
2.3.2 Specification of use case “Search photographer” .....	22
2.3.3 Specification of use case “Filter photos” .....	24
2.3.4 Specification of use case “Scan QR code for collection” .....	25
2.4 Non-functional requirements .....	26
<b>Chapter 3 Technology .....</b>	<b>28</b>
3.1 ReactJS.....	28
3.2 React Native.....	29
3.3 Firebase .....	30
3.4 Cloud Firestore .....	31
3.5 Cloud Storage .....	32
3.6 Cloud Functions and NodeJs .....	33
3.7 Geohash .....	33
3.8 QR code .....	35
<b>Chapter 4 Software development and deployment.....</b>	<b>37</b>
4.1 Architecture Design .....	37
4.1.1 Architecture Design.....	37
4.1.2 General Design .....	40
4.1.3 Detail Package Design.....	42
4.2 Detail Design .....	47

4.2.1 User interface .....	47
4.2.2 Class Design .....	51
4.2.3 Conceptual data model .....	56
4.2.4 Logic data model .....	56
4.3 Application Construction .....	60
4.3.1 Libraries and tools .....	60
4.3.2 Achievements .....	61
4.3.3 Illustrations of main features .....	61
4.4 Test.....	64
4.5 Implementation .....	67
<b>Chapter 5 Solutions and main contributions.....</b>	<b>68</b>
5.1 Search nearby photographer by Geohash .....	68
5.1.1 Problem Description.....	68
5.1.2 Solution .....	68
5.2 Create friendly UI/UX – Animation on Map Screen for searching nearby photographer. .....	71
5.2.1 Problem description.....	71
5.2.2 Solution .....	71
5.3 Share collection by QR Code.....	73
5.3.1 Problem description.....	73
5.3.2 Solution .....	73
5.4 Filter photos by tags.....	75
5.4.1 Problem description.....	75
5.4.2 Solution .....	76
5.5 Firestore and surrounding issues.....	77
5.5.1 Problem description.....	77
5.5.2 Solution .....	78

<b>Chapter 6 Conclusions and Future Work.....</b>	<b>84</b>
6.1 Conclusions.....	84
6.2 Future Work.....	84
<b>Reference.....</b>	<b>85</b>



# List of figures

<b>Figure 1</b> Overall use case diagram.....	7
<b>Figure 2</b> Decomposition of use case "Search photographer".....	8
<b>Figure 3</b> Decomposition of use case "Connect to photographer".....	9
<b>Figure 4</b> Decomposition of use case "Manage collection".....	10
<b>Figure 5</b> Decomposition of use case "Manage individual collection".....	11
<b>Figure 6</b> Decomposition of use case "Manage message".....	12
<b>Figure 7</b> Decomposition of use case "View photo".....	13
<b>Figure 8</b> Decomposition of use case "Manage user".....	14
<b>Figure 9</b> Decomposition of use case "Manage photo".....	15
<b>Figure 10</b> Decomposition of use case "Manage tag".....	16
<b>Figure 11</b> Decomposition of use case "Scan QR code for collection".....	17
<b>Figure 12</b> Searching and connecting to photographer process.....	18
<b>Figure 13</b> Filtering photo by tags.....	19
<b>Figure 14</b> Creating and Scanning QR code for collection.....	20
<b>Figure 15</b> Store data in Cloud Firestore.....	32
<b>Figure 16</b> Geometrical representation.....	34
<b>Figure 17</b> Illustrate geohash on map.....	35
<b>Figure 18</b> Example of QR code in PhotoHub application.....	36
<b>Figure 19</b> System architecture design.....	37
<b>Figure 20</b> MVC pattern.....	38
<b>Figure 21</b> Flux pattern.....	39

<b>Figure 22</b> General Design of server-side .....	40
<b>Figure 23</b> General design of PhotoHub and PhotoHub PG front-end client application....	41
<b>Figure 24</b> General design of PhotoHub Admin front-end client application.....	42
<b>Figure 25</b> Detail design of package “navigation” .....	42
<b>Figure 26</b> Detail design of package “redux” .....	43
<b>Figure 27</b> Detail design of package “component” .....	44
<b>Figure 28</b> Detail design of package “configs” .....	45
<b>Figure 29</b> Detail design of package “utils” .....	46
<b>Figure 30</b> Photohub application screens .....	49
<b>Figure 31</b> Photohub Admin web application screens .....	50
<b>Figure 32</b> Photohub PG application screens .....	50
<b>Figure 33</b> Class diagram of use case “Search nearby photographer by suggested location” .....	51
<b>Figure 34</b> Sequence diagram of use case “Search nearby photographer by suggested location” .....	53
<b>Figure 35</b> Class diagram of use case “Search nearby photographer by suggested location” .....	54
<b>Figure 36</b> Sequence diagram of use case “Search nearby photographer by suggested location” .....	55
<b>Figure 37</b> Entity Relationship Diagram .....	56
<b>Figure 38</b> user entity .....	56
<b>Figure 39</b> image entity .....	57
<b>Figure 40</b> tag entity .....	57
<b>Figure 41</b> collection entity .....	58
<b>Figure 42</b> chatroom entity .....	58
<b>Figure 43</b> location entity .....	58

<b>Figure 44</b> message entity .....	59
<b>Figure 45</b> Searching and connecting nearby photographer screens.....	62
<b>Figure 46</b> Photograph filter screens .....	63
<b>Figure 47</b> QR code scan screen.....	64
<b>Figure 48</b> UI/UX – Animation on Map Screen.....	72
<b>Figure 49</b> Collection sharing screen .....	74
<b>Figure 50</b> Scan QR code screen .....	74
<b>Figure 51</b> Picture description by tags.....	76
<b>Figure 52</b> Photographs filter screen .....	77
<b>Figure 53</b> Illustration of relationship between photograph, user, and comment.....	79
<b>Figure 54</b> Illustration of relationship between user and album.....	80
<b>Figure 55</b> Illustration of storing tags in a map .....	82

# List of tables

<b>Table 1</b>	Survey of photography demand with smartphones and cameras .....	5
<b>Table 2</b>	Specification of use case “View photo” .....	21
<b>Table 3</b>	Specification of use case “Search photographer” .....	22
<b>Table 4</b>	Specification of use case “Filter photos” .....	24
<b>Table 5</b>	Specification of use case “Scan QR code for collection” .....	25
<b>Table 6</b>	User interface information .....	47
<b>Table 7</b>	Specification of class MapScreen .....	51
<b>Table 8</b>	Specification of class BookingController .....	54
<b>Table 9</b>	Libraries and tools.....	60
<b>Table 10</b>	The difference between Top-level collection and Sub-collection .....	78
<b>Table 11</b>	The differences between array, map and collection.....	81

# List of acronyms

<b>API</b>	Application Programming Interface
<b>UI</b>	User Interface
<b>UX</b>	User Experience
<b>REST</b>	Representational State Transfer
<b>QR code</b>	Quick Response code

# Chapter 1 Introduction

## 1.1 Problem Description

In the past, professional photography services were expensive and were used for special events such as graduation and weddings only. In recent years, with modern smartphones, photography has become popular; people photograph everywhere to save memorable moments. However, a quality photograph depends on various factors such as: equipment, photography skills, weather conditions, etc. It is difficult for us, who are not professional photographer, to take quality photographs with smartphones and limited photography skills.

Due to development of technology, the price of professional photographic equipment is going down. Because of this, the number of amateur and semi-professional photographers is also increasing rapidly. They may be photography enthusiasts, amateur photographers who want to train their own skills, or professional photographer to make money. However, those photographers may not know how to find their customers. On the other hand, getting in touch with those photographers is an issue as their customers do not have a mean to find nearby photographers. This problem becomes more serious as the need for photographer is accidental, during their trips or outdoor activities.

Another problem is posing during photography. Regardless of how photos are taken, it is very important to get the best poses that appropriate the best scene. However, normal people don't know how to pose and express to fit each situation, such as a single, group, family, different scenes. Therefore, sometimes people take up to a few hundred photos but only choose a few satisfying images.

Thus, there are two main issues: (1) there is no effective connection between people who want to take photos and photographers who do not know how to reach clients; (2) there are many limitations in supporting users to pose in accordance with different circumstances. It is necessary to design an appropriate solution to solve the above two problems.

## 1.2 Objective and Scope

PhotographMe<sup>1</sup> is one of the very few applications that support searching photographers. The application allows users to search for photographers by address.

Some mobile applications such as Ulike<sup>2</sup> and SOVS<sup>3</sup> aim to support poses for users. The poses in the application are diverse. Especially, those are drawn directly on the camera screen for users to apply, followed by a series of filters, image editing.

This project objective is building a photography system PhotoHub that connects between users and photographers, and suggests photograph poses for them.

To connect photographers and their clients, PhotoHub application follows the popular ride-hailing models such as Grab<sup>4</sup>, Uber<sup>5</sup>, GoViet<sup>6</sup>, Bee<sup>7</sup>. It provides users with an intuitive map interface with different searching ways such as using their location, choosing the location on the map or address of the region. Users can search the route and connect with the photographer by sending a message to them.

To help users to take their photographs, PhotoHub system focuses on providing users with various poses that have been photographed in real life, users can also search for photographs that suit their outfit, accessories, age, gender, and landscape, etc. PhotoHub system also allows users to create their own collections to manage the photos more easily and can share these collections with friends or photographers.

## 1.3 Solution Approach

The PhotoHub system has three different types of users.

Firstly, for normal users who want to connect with photographers or search poses, we need to build an application that can help users find photography anytime, anywhere. To this end,

---

<sup>1</sup> <https://play.google.com/store/apps/details?id=com.photographmeapp.photographme2>

<sup>2</sup> <https://play.google.com/store/apps/details?id=com.gorgeous.liteinternational>

<sup>3</sup> <https://play.google.com/store/apps/details?id=me.sovs.sovs2>

<sup>4</sup> <https://play.google.com/store/apps/details?id=com.grabtaxi.passenger>

<sup>5</sup> <https://play.google.com/store/apps/details?id=com.ubercab>

<sup>6</sup> <https://play.google.com/store/apps/details?id=com.goviet.app>

<sup>7</sup> <https://play.google.com/store/apps/details?id=xyz.be.customer>

we decide to develop mobile application. Thus, users can easily connect with photographers, and the search for posing, storage also helps in maximum during the photography process.

Secondly, for photographers who can update location and information for users to search, building an application on mobile platforms is also the most reasonable solution.

Thirdly, for administrator who are data managers, they can create data sources on posing, as well as the functions related to administration, building applications on the website platform will help administrators can cover data and information, manipulation is also easier.

The system is built following the client-server architecture. More detail:

- i) Data of system is stored on Firestore which is a NoSQL database of Firebase.
- ii) The server is built on a stateless framework called “Cloud Functions for Firebase” running on Google Cloud Platform. This framework is developed on Node.JS environment. It provides connections to Firestore database, Firebase Storage and Client-side application.
- iii) The client consists of three applications geared towards three different types of user as mentioned above. PhotoHub application is built on React Native (mobile application) platform approached to normal users who have photography demand. PhotoHub PG application is built on React Native (mobile application) platform approached photographers. PhotoHub Admin application is built on React Js (web application) platform approached to system administrators.

## 1.4 Thesis outline

The rest of this graduation thesis report is organized as follows:

**Chapter 2** presents the following four contents: (i) Surveying photography using smartphone and camera, surveying mobile and web applications and examining applications that support photographic poses, (ii) Describe functions of the application through the overall use case diagram, (iii) Specification of some important use cases of the system such as searching for photographer, connecting with photographer, searching photographs by tags , sharing collections, etc, (iv) Non-functional requirements of the system.

Chapter 3 presents the technologies used for application development. NodeJs with the "Cloud Function for Firebase" framework to build back-end systems. NoSQL database Firestore to store data of the system. The user interface uses ReactJs, React Native with number of accompanying libraries to support the development process. Geohash is an approach is used to search nearby photographers. For each technology, you will present the



advantages and disadvantages of the reason for choosing that technology for application development.

**Chapter 4** presents the content of application development and deployment including architecture design, package and class design, interface design, application construction, application testing and deployment.

**Chapter 5** presents the content of the application's contributions and solutions in this system.

**Chapter 6** presents a summary of the achieved results of the system and the orientation of future development solutions and makes experiences after completing building the system.

# Chapter 2 Survey and requirements analysis

## 2.1 Survey

Smartphone can not replace the camera in some cases. We tend to capture special moments with a camera rather than a smartphone. This is inferred from qandme<sup>8</sup> survey as **Table 1**

Number of respondents: 600

From 20 - 39 year olds in Hanoi and Ho Chi Minh

Conducted on Nov, 2018

**Table 1** Survey of photography demand with smartphones and cameras

Photo Objects	Smartphone	Camera
Photo Objects	Selfie (65%) Scenery (59%) Family / Kid (55%) Foods (53%)	Scenery (77%) Night shot (57%) Family / Kids (56%) Friend (43%)
Satisfaction	Easiness to use (60%) Easiness to upload/share (44%) Easiness to retouch (38%)	Quality in dark place (62%) Zoom feature (65%) Easiness to use (50%)
Dissatisfaction	Zoom feature (33%) Quality in dark place (32%) Hand shaking (29%)	Weight (35%) Shutter speed (28%) Easiness to upload/share (28%)

---

<sup>8</sup> <https://qandme.net/en/report/camera-usage-situation-vietnam.html>

Satisfactions of smartphone cameras are "Easiness to use camera function"(60% of 600 respondents), "Easiness to upload/share"(44%) and "Easiness to use retouch apps"(38%). While dissatisfactions of smartphone cameras are "Zoom feature"(33%), "Quality in dark place"(32%) and "Hand shaking"(29%). Dissatisfactions of smartphones are satisfactions of digital camera. Therefore, people who are middle or high income tend to have digital camera and they use both smartphone and digital camera depending on occasions.

Thereby, we can see that the smartphone can not replace the camera in some cases. Normal users will tend to capture special moments with a camera rather than a smartphone.

Almost applications that offer services to connect photographers and their clients are web-based; PhotographyMe is a rare mobile-based application. Those applications typically operate as a social network, asking for the exact address, it is appropriate with the intended photography. In case that the need for photographer is accidental, we may not know the exact address of the location we are in, these applications become more difficult to connect photographers and their clients.

There are some applications such as ULike, SOVS provide users with various poses, but these poses are not specific in different circumstance such as backgrounds or accessories. The poses are just strokes drawn on the camera background, that makes it much easier to pose. However, because these poses are not real photographs that have been taken outside, it makes people don't imagine the results after creating these poses, which makes it more difficult to choose which posture to shoot.

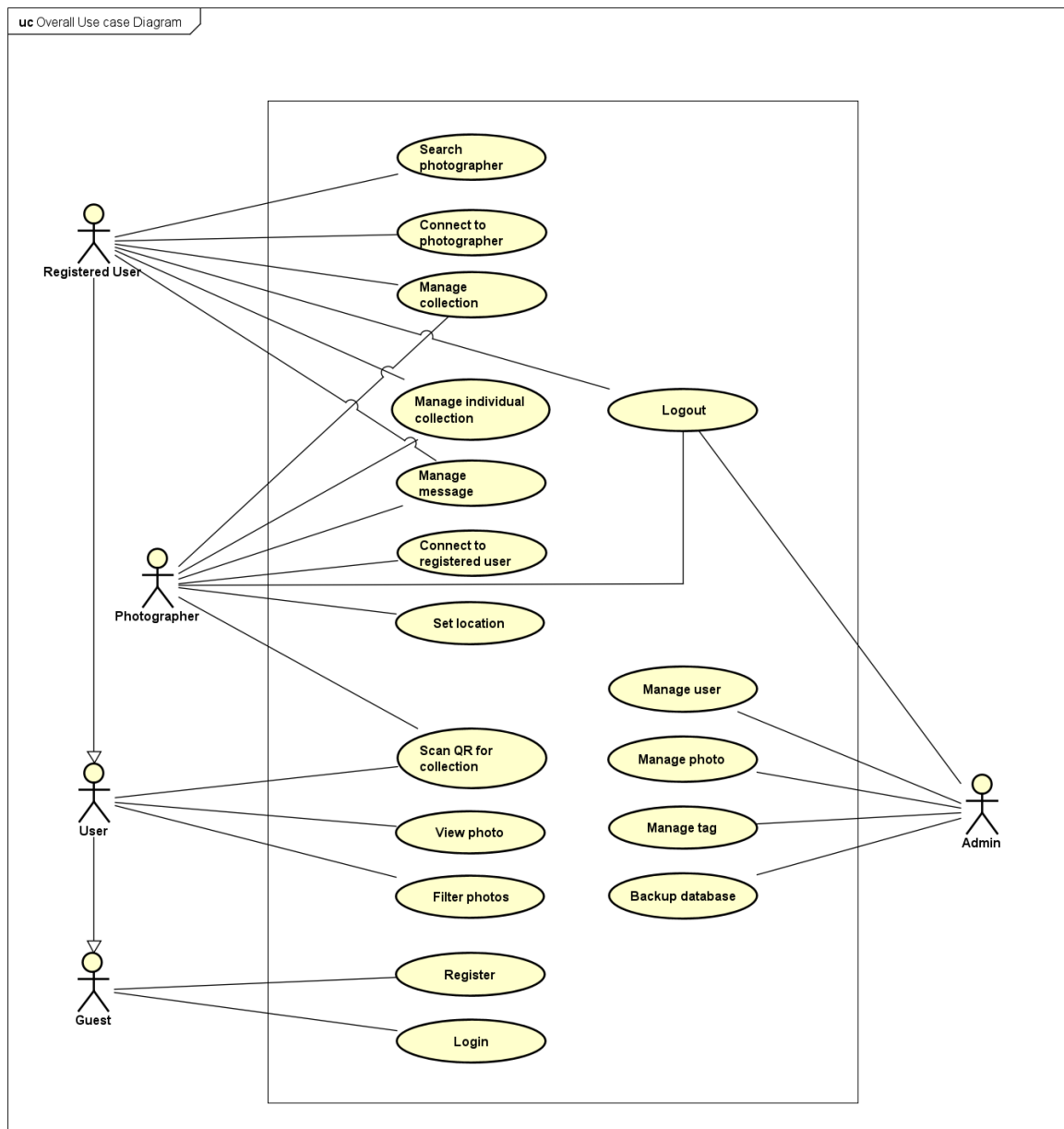
From the aforementioned analysis, the PhotoHub system was created to connect photographers and their clients, and provide users various poses supporting for photography. PhotoHub application supports the search of photographers by address, by your location or by optional position on the map through the map interface (similar to the models of Grab, GoViet, Bee). Users and photographers can also find their direction through the application interface. The application also provides a photograph gallery to assist users in the photography process, users can search for postures through categories such as background, accessories, age, gender, etc.

## **2.2 Overall Description**

### **2.2.1 Overall Use case Diagram**

Follow the result of survey, all use cases of system describe by overall use case diagram as **Figure 1**.

System includes 5 main actors: Guest, User, Registered User, Photographer, Admin.



**Figure 1** Overall use case diagram

Guest is actor that doesn't have account or doesn't login. The first time when going into application, people using the app will play a role as guest. Guest only can register a new account or login to become another role in application.

For PhotoHub application, Guest also is User. User is actor using application without login in PhotoHub application. A user can view list of photographs, search photographs with tags, and see detail of each photograph. In addition, user also can scan QR code to retrieve collection from registered user.

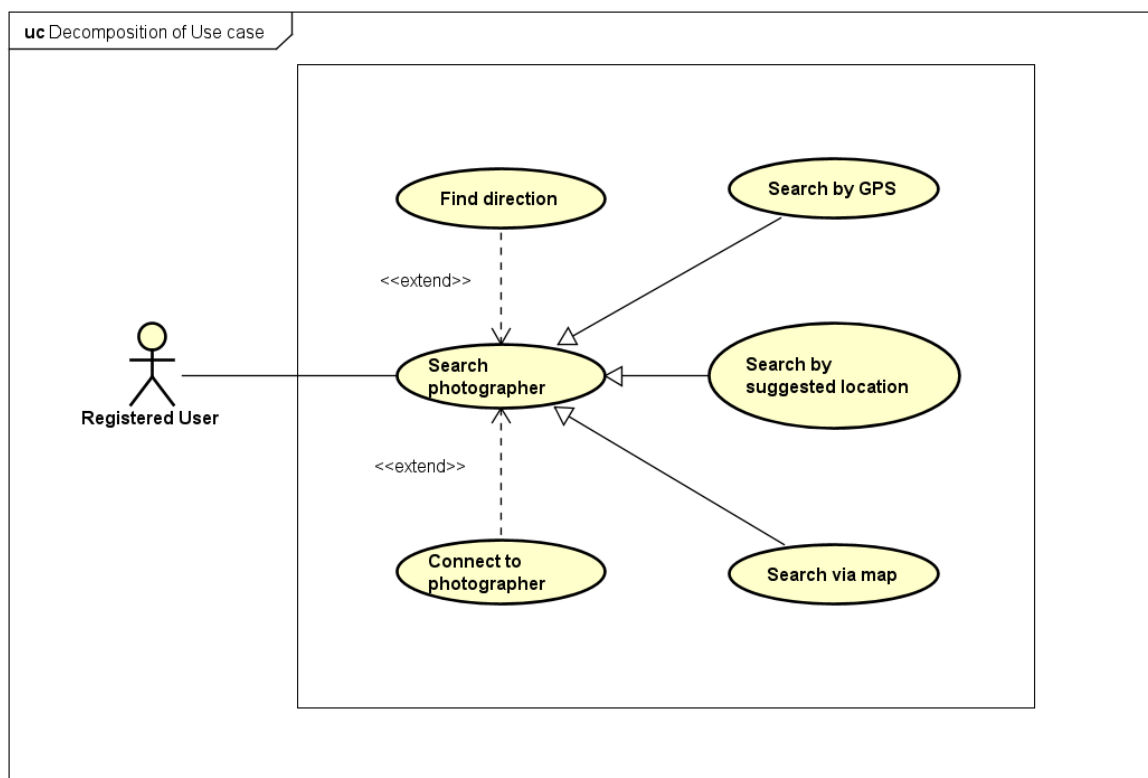
In order to more features of PhotoHub application, user need to become registered user. Registered user is actor who owns their account and uses that account log into the system.

Registered user can use not only all functions that user has, but also can do some actions on viewing photograph, managing collection, searching and connecting to photographer. Registered user can like and add image to any collection their have, create their own collection and manage them, search any photographer who is in given radius surrounding determined location.

Another main actor is photographer. Photographer is actor who has already had account and use that account log into PhotoHub PG application. This actor can set location, manage collection, scan QR code and response to registered user.

The last main actor is admin. Admin is actor who has already had account and use that account log into PhotoHub Admin application. This actor can manage database, configure system, etc.

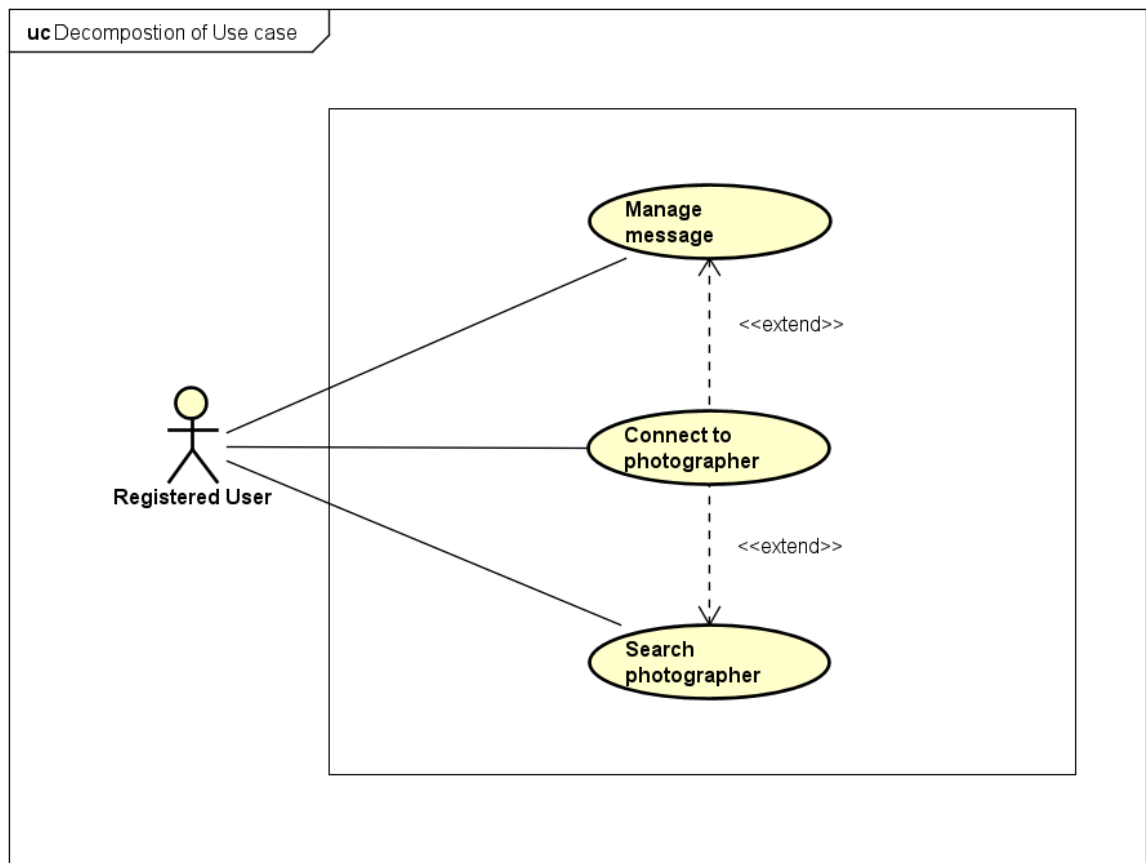
### 2.2.2 Decomposition of use case “Search photographer”



**Figure 2** Decomposition of use case "Search photographer"

**Figure 2** Decomposition of use case “Search photographer” describes all functions that registered user can use for searching photographer. There are three types of searching photographer: search by your location, search via map and search by address. After searching completed, registered user can find the way to go to photographer’s location or connect to them by sending messages.

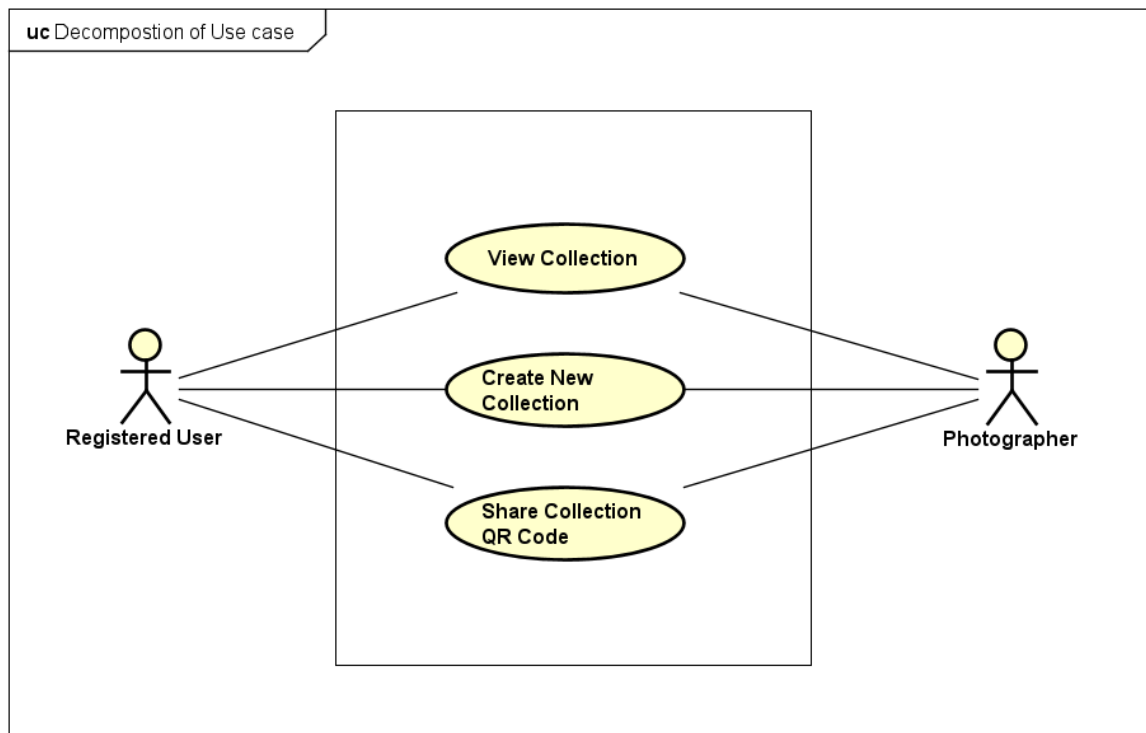
### 2.2.3 Decomposition of use case “Connect to photographer”



**Figure 3** Decomposition of use case “Connect to photographer”

**Figure 3** Decomposition of use case “Connect to photographer” describes the functions that registered user can use when connecting to photographer. Registered user can connect to photographer from either managing message or searching photographer.

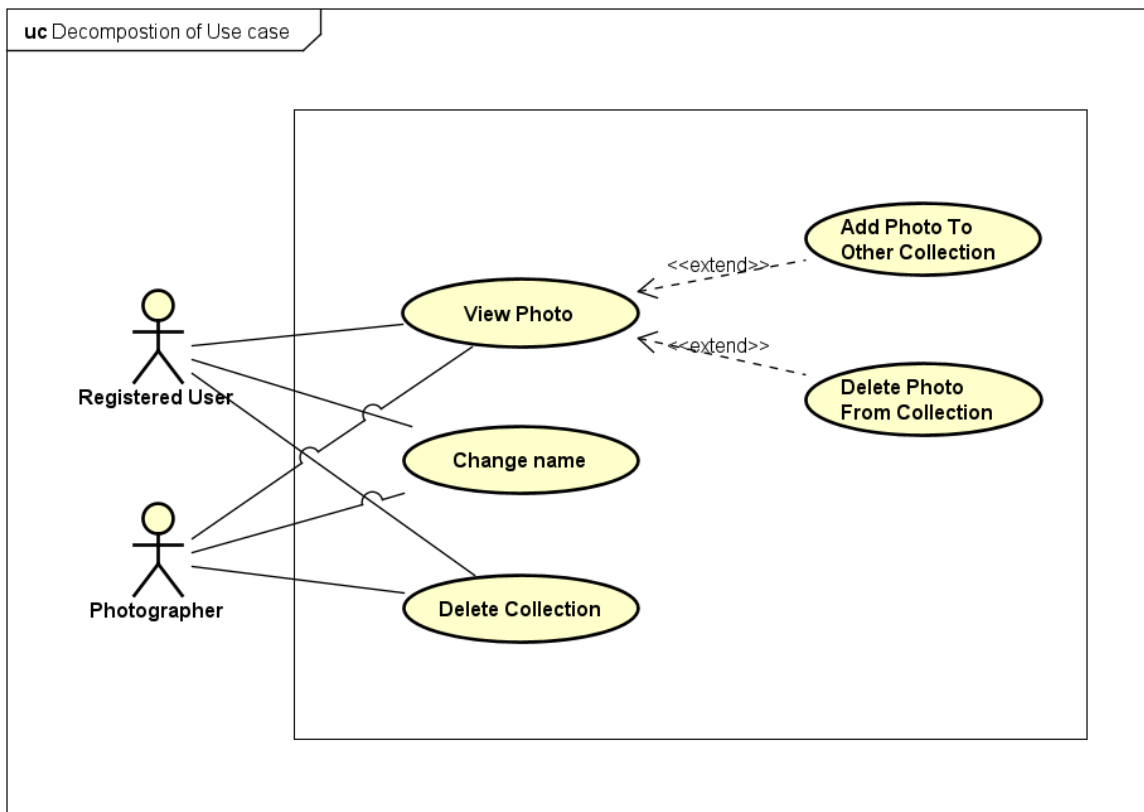
## 2.2.4 Decomposition of use case “Manage collection”



**Figure 4** Decomposition of use case “Manage collection”

**Figure 4** Decomposition of use case “Manage collection” describes the functions that used by two actors who are registered user and photographer. Accessible actors can create new collection, view all collections they have, and share any collection to another users.

### 2.2.5 Decomposition of use case “Manage individual collection”

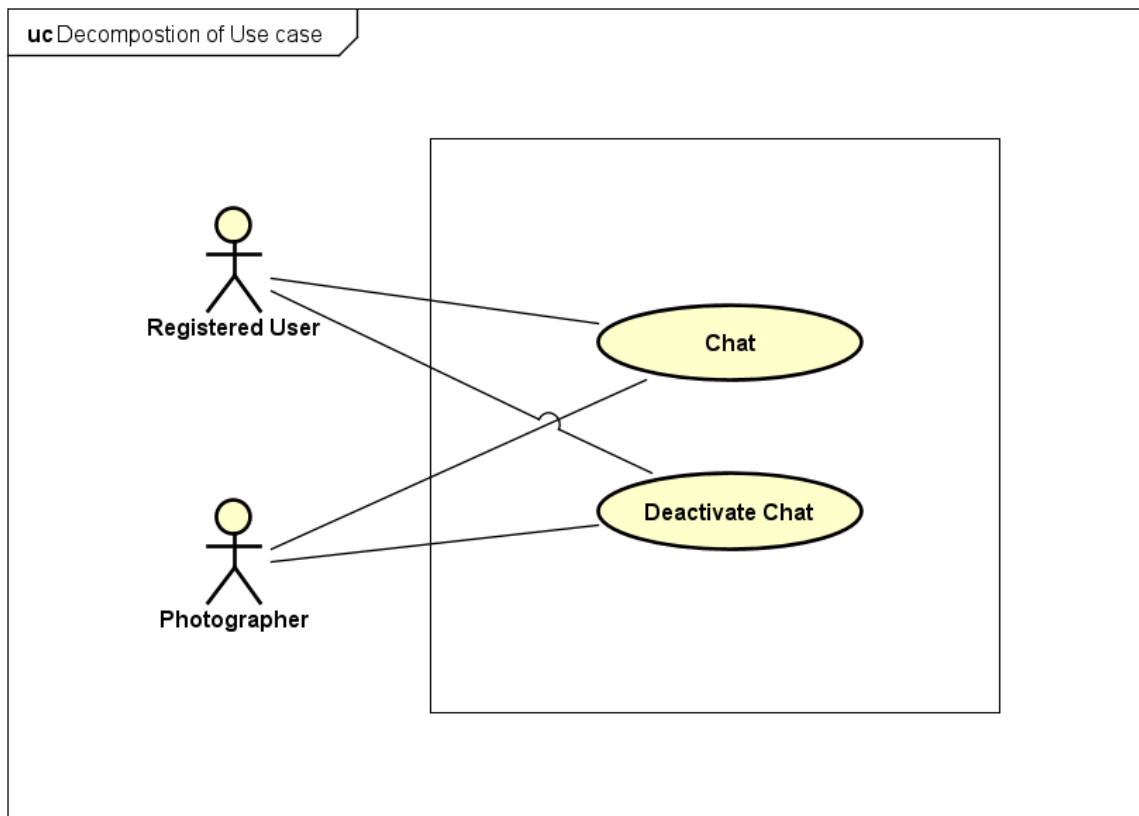


**Figure 5** Decomposition of use case “Manage individual collection”

**Figure 5** Decomposition of use case “Manage individual collection” describes the functions that used by two actors who are registered user and photographer. Accessible actors can do some actions inside each collection like viewing photograph, changing name of collection, and deleting collection. Moreover, adding photograph to other collection and deleting photograph from collection is available for managing individual collection.



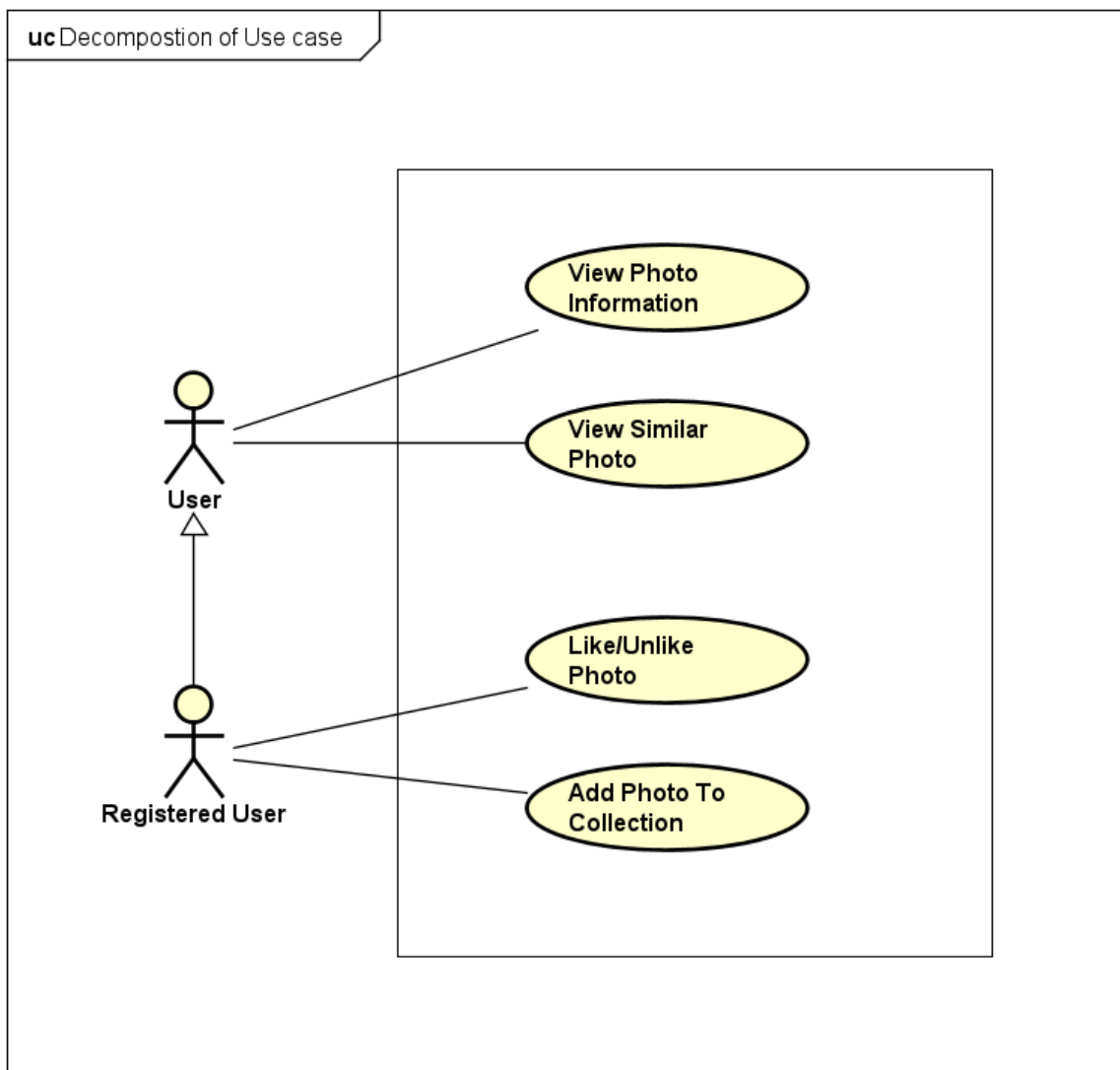
## 2.2.6 Decomposition of use case “Manage message”



**Figure 6** Decomposition of use case “Manage message”

**Figure 6** Decomposition of use case “Manage message” describes the functions that used by two actors who are registered user and photographer. Registered user can chat with photographer and deactivate chatroom.

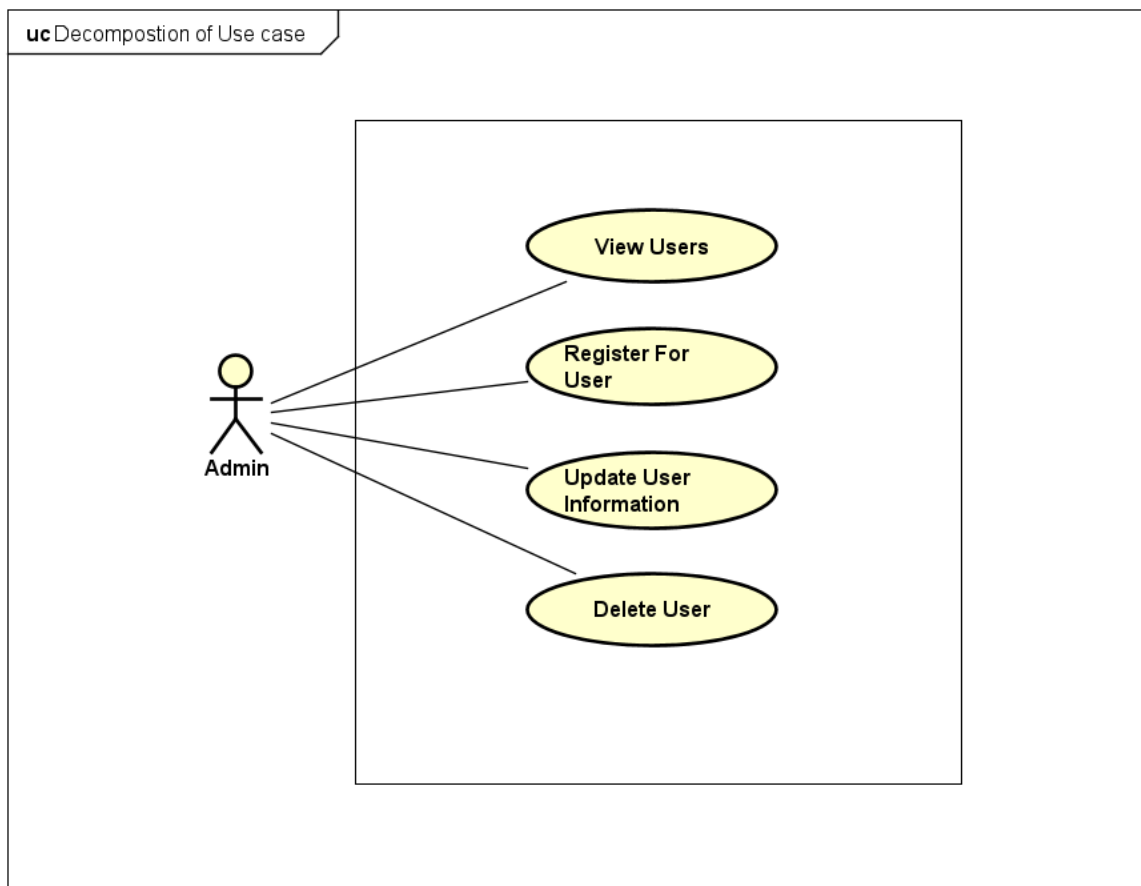
### 2.2.7 Decomposition of use case “View photo”



**Figure 7** Decomposition of use case “View photo”

**Figure 7** Decomposition of use case “View photo” describes the functions that used by two actors who are user and registered user. User can view photograph information like name, tags, etc, also can see similar photographs when filtering photograph by tags. Registered user can do more actions than user like like or unlike photograph and add photograph to any collection.

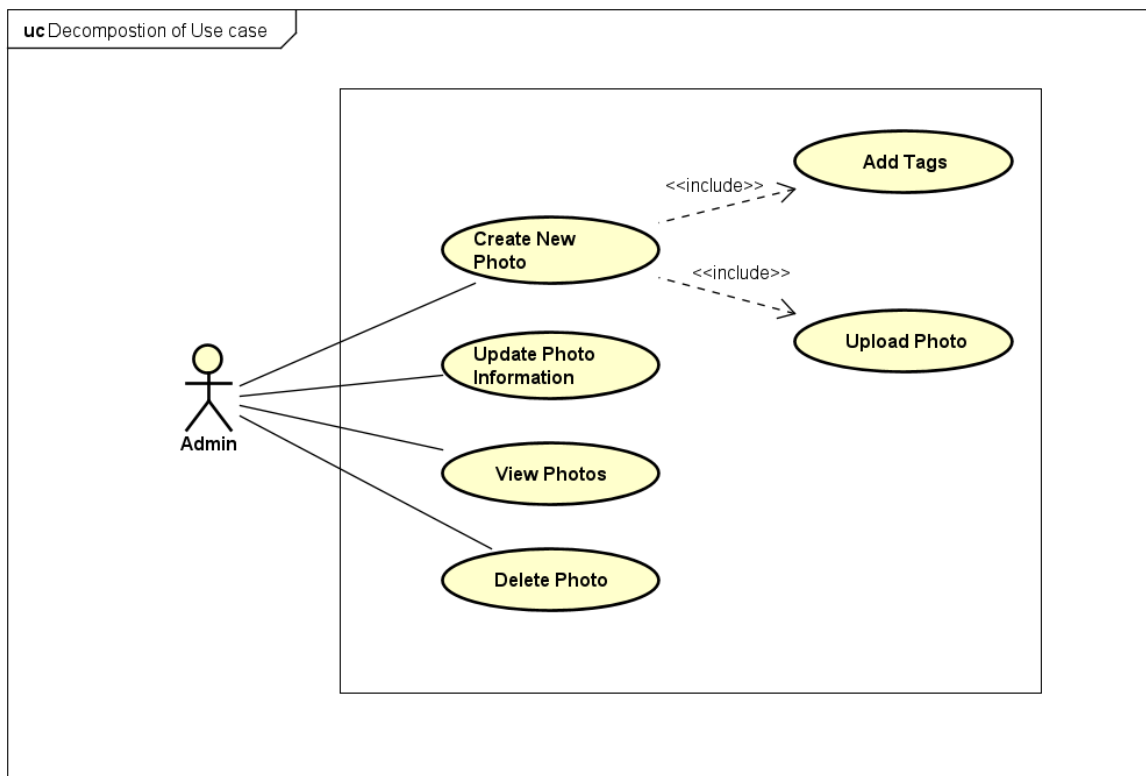
### 2.2.8 Decomposition of use case “Manage user”



**Figure 8** Decomposition of use case “Manage user”

**Figure 8** Decomposition of use case “Manage user” describes the functions that used by admin. Admin can view list of all users in database, register a new account for registered user or photographer, update user information and delete a user from database.

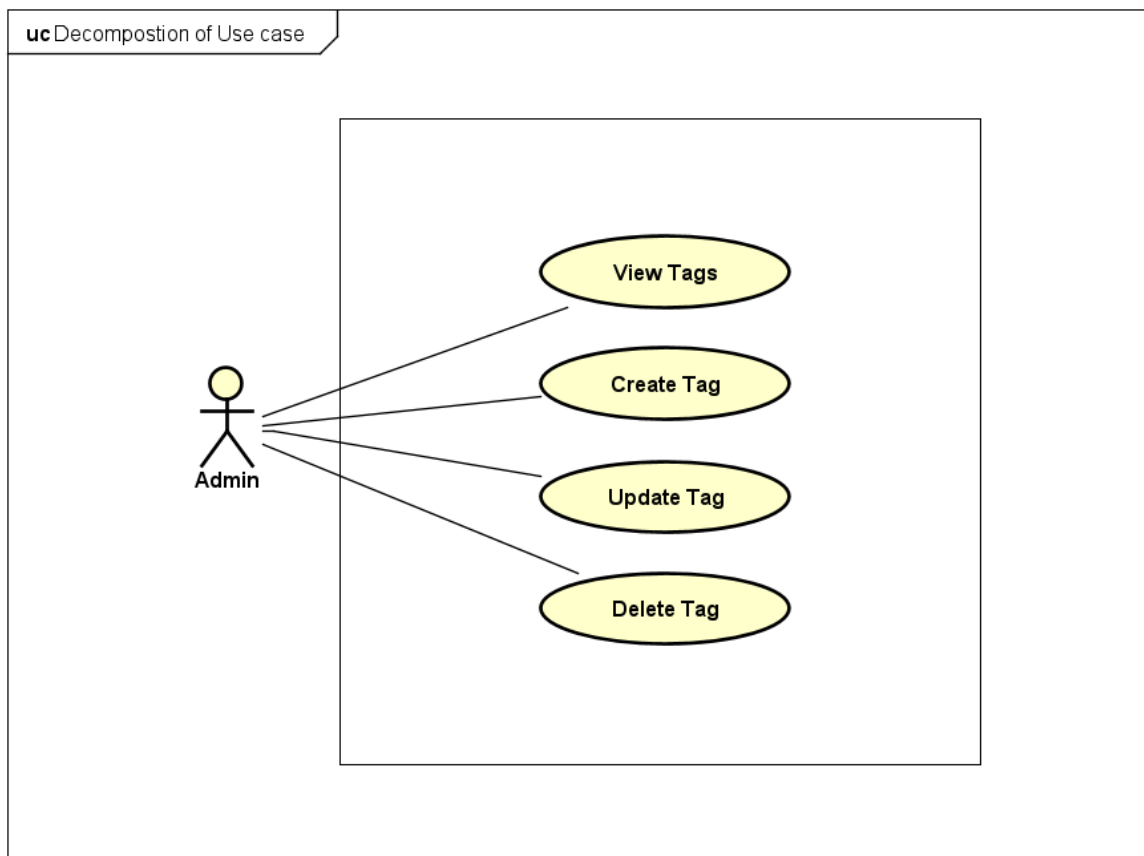
### 2.2.9 Decomposition of use case “Manage photo”



**Figure 9** Decomposition of use case “Manage photo”

**Figure 9** Decomposition of use case “Manage photo” describes the functions that used by admin. Admin can view list of all photographs in database, create new photo including some added tags and uploaded photograph, update photograph information, and delete a photograph from database.

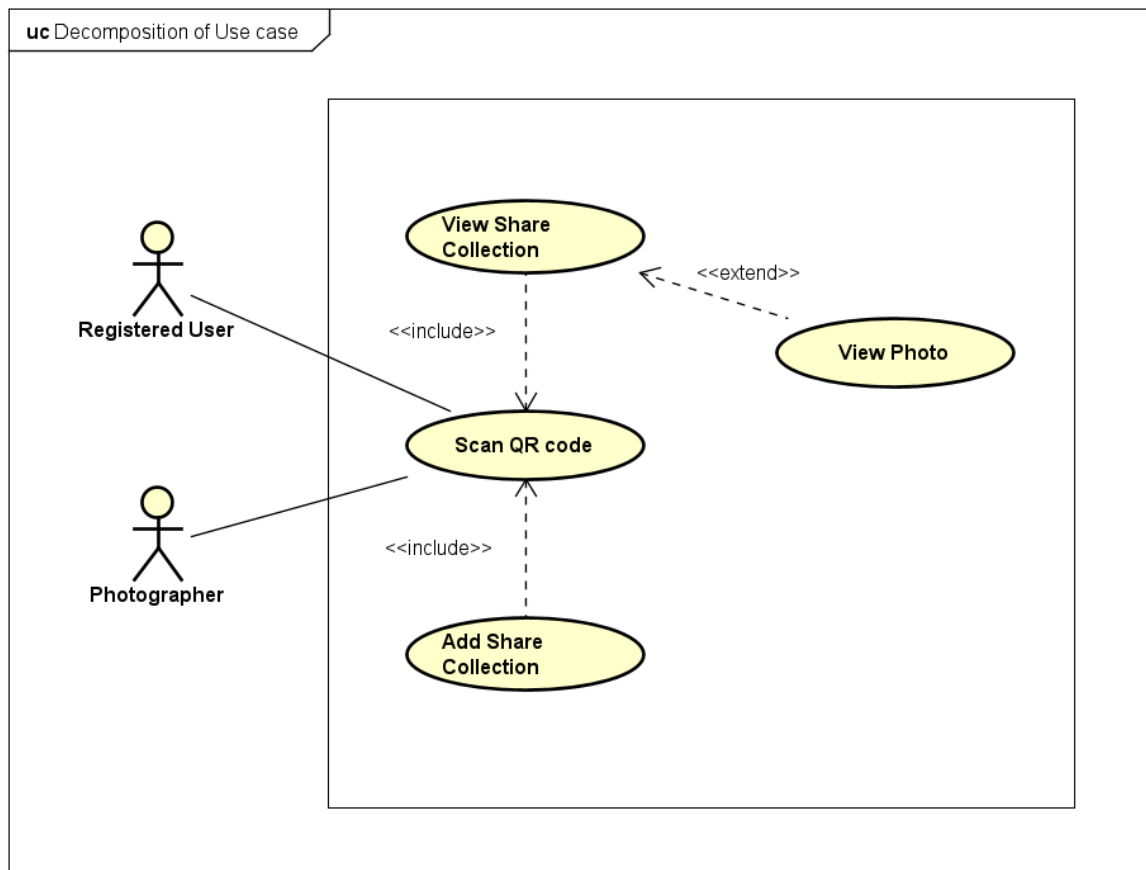
### 2.2.10 Decomposition of use case “Manage tag”



**Figure 10** Decomposition of use case “Manage tag”

**Figure 10** Decomposition of use case “Manage tag” describes the functions that used by admin. Admin can view list of all tags in database, create new tag with determined category, update information of any tag, and delete a tag from database.

### 2.2.11 Decomposition of use case “Scan QR code for collection”



**Figure 11** Decomposition of use case “Scan QR code for collection”

**Figure 11** Decomposition of use case “Scan QR code for collection” describes the functions that used by two factors who are registered user and photographer. Accessible actors can scan QR code by capturing QR code using camera of mobile phone, then retrieve collection from QR code to view and add to collection.

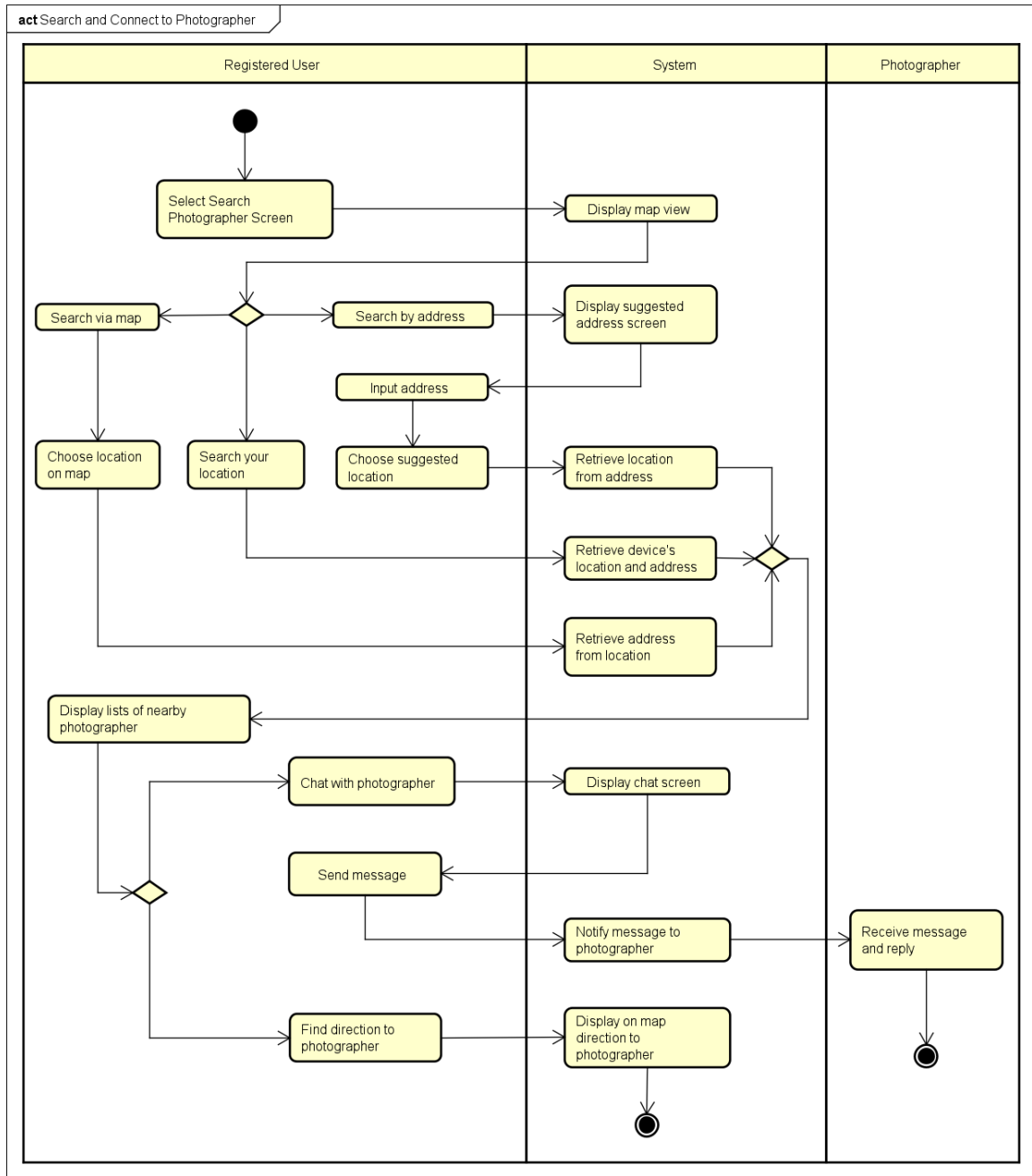
### 2.2.12 Business processes

Developing system includes some important business processes needed to detail.

Firstly, searching and connecting to photographer process is described as **Figure 12**. From the beginning, registered user needs to go into “Search photographer” screen to see some available ways to search nearby photographer. There are three ways to do that.

- i) Search your location. That means the system will automatically retrieve device’s location and try to figure out address corresponding to location.

- ii) Search by address. Registered user can move to suggested address screen, type address and choose suggested location. System will retrieve location from address.
- iii) Search via map. Registered user can move center of map view to choose exact location, therefore system will retrieve location and address from selected point on map.



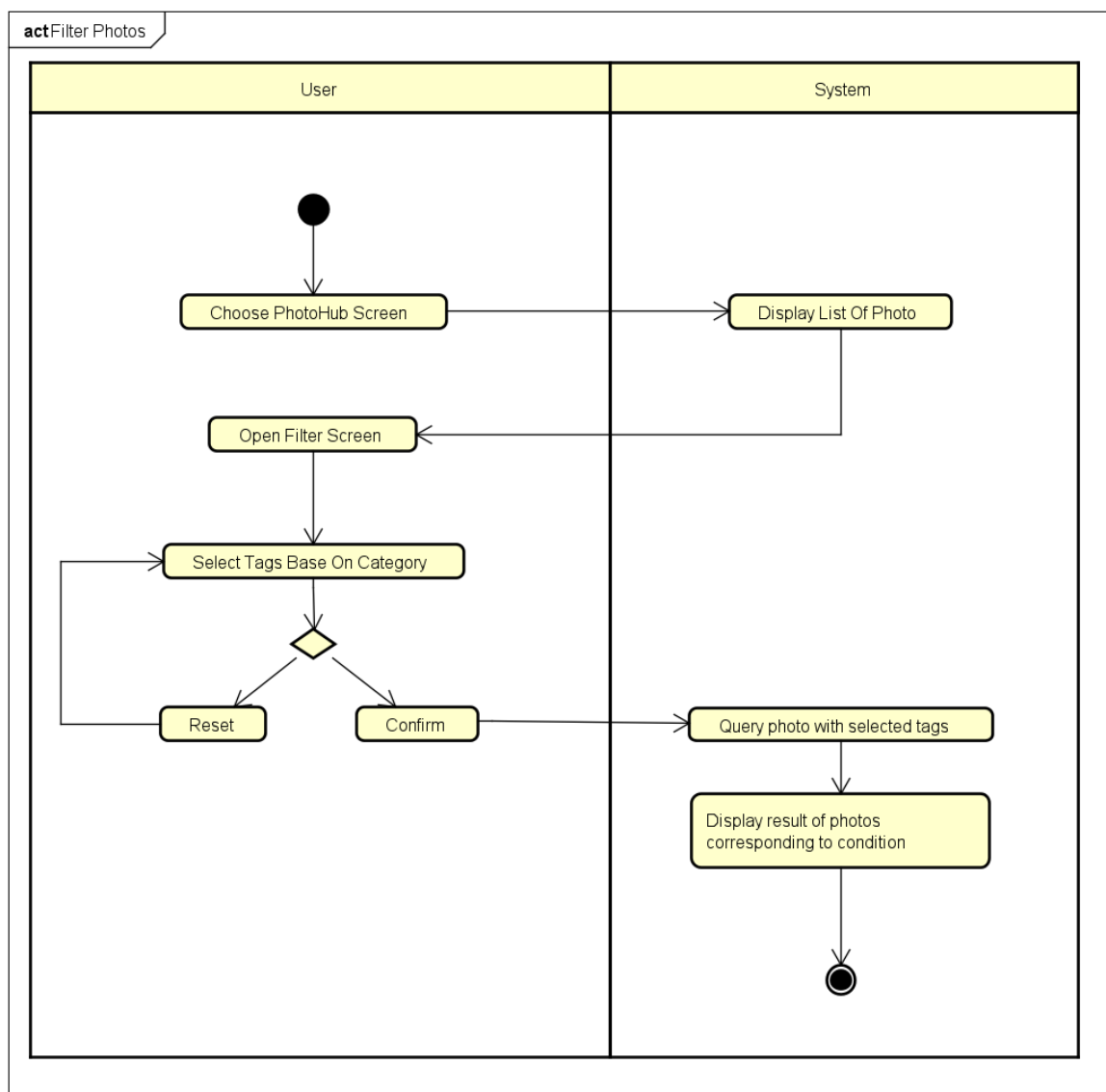
**Figure 12** Searching and connecting to photographer process

After having location and address, system will search around by certain radius from location. Subsequently, registered user can see all available nearby photographer on map. They can

either retrieve direction to photographer or connect to photographer by sending message to photographer. For getting direction, registered user can see a path displayed on map. For connecting to photographer, registered user sending messages, system will notify message to photographer, photographer will receive a push notification and do some actions like reply in conservation.

Secondly, filtering photographs by tags is described as

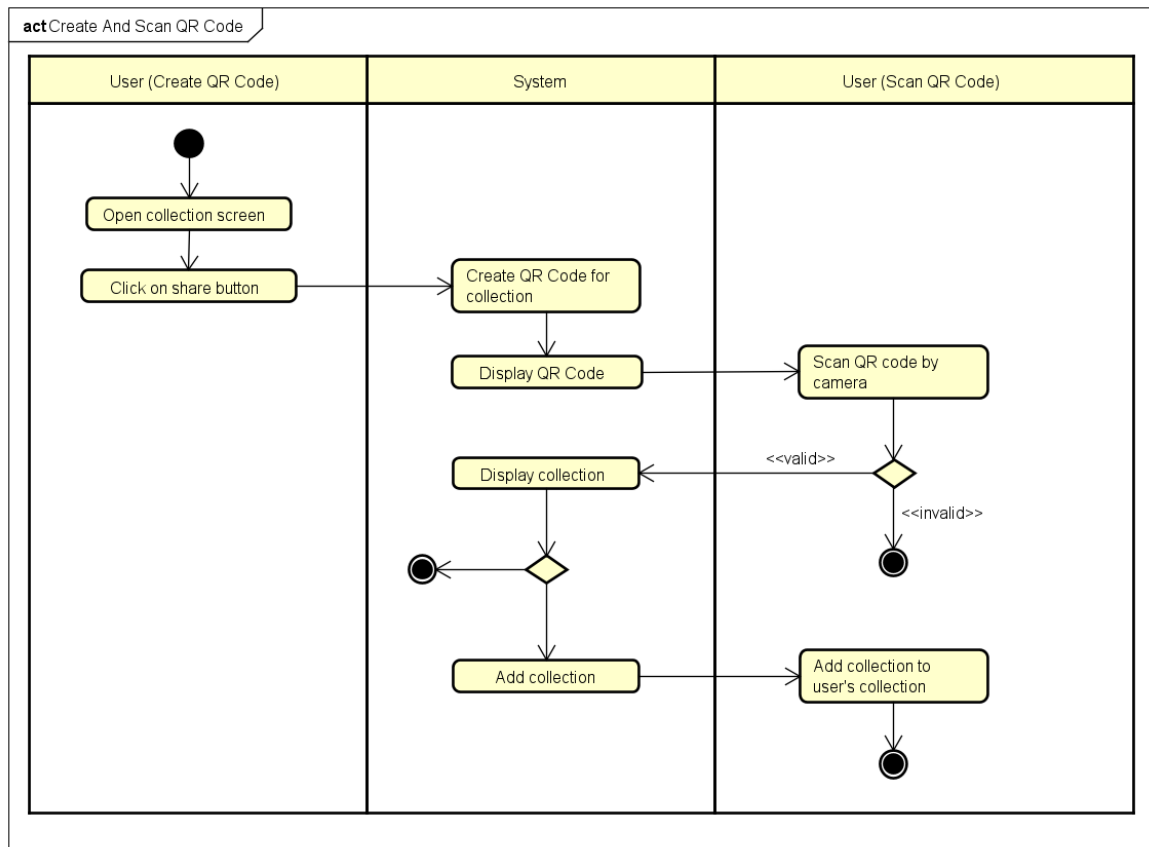
**Figure 13.** From the beginning, actors including user and registered user need to go into PhotoHub screen to see list of available photographs. After that, user open filter screen and select some tags. User can reset to remove all selected tags or confirm to filter photographs with selected tags. System will search all photographs satisfying selected tags and display result corresponding to conditions.



**Figure 13** Filtering photo by tags



Thirdly, creating and scanning QR code for collection is described as **Figure 14**. From the beginning, User including registered user or photographer who intend to create QR code from collection need to open collection screen and click share button to retrieve QR code. QR code will displayed on screen. User who want to scan QR code will use camera to capture code. If QR code is invalid, the process will end, user need to try again or out function. In constrast, if code is valid, a collection corresponding to QR code will display on screen, user can choose add collection or not.



**Figure 14** Creating and Scanning QR code for collection

## 2.3 Specific requirements.

### 2.3.1 Specification of use case “View photo”

**Table 2** Specified details of given use case “View photo”.

**Table 2** Specification of use case “View photo”

<b>Use case code</b>	UC001	<b>Use case name</b>	View photo
<b>Actor</b>	User, Registered user		
<b>Pre-condition</b>	Install successfully Photohub application.		
<b>Main flow of events (Success)</b>	<b>No</b>	<b>Actor</b>	<b>Action</b>
	1.	User/ Registered user	Select photograph to view
	2.	System	Display photograph with detail information: name, tags, similar photos.
	3.	User/ Registered user	Like photograph, add photograph to collection or view similar photographs.
	3.1	Registered user	Like photograph
	3.1.1	Registered user	Select button “Like” to like photograph
	3.1.2	System	Increase number of likes of photograph and automatically add photograph to collection “Favourite”
	3.2	Registered user	Add photograph to collection
	3.2.1	Registered user	Select button “Add to collection” to add photograph to any collection.
	3.2.2	System	Display modal which lists all user’s available collections
	3.2.3	Registered user	Click to select/unselect collection
	3.2.4	System	Add/remove photograph to/from collection

	3.3	Registered user	Click on a similar photo to view
<b>Alternative flow of events</b>	<b>No</b>	<b>Actor</b>	<b>Action</b>
	3.1.2a	System	Do nothing if like/unlike photograph unsuccessfully
	3.2.4a	System	Do nothing if add/remove photograph to/from collection unsuccessfully
<b>Post-condition</b>	None		

### 2.3.2 Specification of use case “Search photographer”

**Table 3** Specified details of given use case “Search photographer”.

**Table 3** Specification of use case “Search photographer”

<b>Use case code</b>	UC001	<b>Use case name</b>	Search photographer
<b>Actor</b>	Registered user		
<b>Pre-condition</b>	Log in successfully as registered user and go into page booking		
<b>Main flow of events (Success)</b>	<b>No</b>	<b>Actor</b>	<b>Action</b>
	1.	Registered user	Select option “Search”
	2.	System	Display a map view on screen with some button displaying the ways to search photographer
	3.	Registered user	Search nearby photographer
	3.1	Registered user	Search by device’s location
	3.1.1	Registered user	Click on button “Your location”
	3.1.2	System	Retrieve location by GPS, find all nearby photographers in certain given radius and display results on map
	3.2	Registered user	Search by suggested address
	3.2.1	Registered user	Click on input button “Search for a

			location”
	3.2.2	System	Move to address suggestion screen
	3.2.3	Registered user	Type address
	3.2.4	System	List all suggested addresses match to typed address
	3.2.5	Registered user	Select address
	3.2.6	System	Go back map view screen. Retrieve location corresponding to address and find all nearby photographers in certain given radius, display results on map
	3.3	Registered user	Search via map
	3.3.1	Registered user	Click on input button “Search for a location”
	3.3.2	System	Move to address suggestion screen
	3.3.3	Registered user	Click on button “Search via map”
	3.3.4	System	Go back map view screen
	3.3.5	Registered user	Move map around to determine which location wanted to search
	3.3.6	Registered user	Click on button “Pick up location”
	3.3.7	System	Retrieve all nearby photographers in certain given radius, display results on map
	4.	Registered user	Manipulate results of nearby photographers
	4.1	Registered user	Find direction to photographer
	4.1.1	Registered user	Click on button “Direction”
	4.1.2	System	Find direction from user to photographer, draw a path on map
	4.2	Registered user	Chat with photographer

	4.2.1	Registered user	Click on button “Connect”
	4.2.2	System	Go into chatroom
	4.2.3	Registered user	Send messages to photographer and wait for responds
<b>Alternative flow of events</b>	<b>No</b>	<b>Actor</b>	<b>Action</b>
	3.1.2a	System	Display alert message that user has no permission to access device’s location
	3.1.2b	System	Display output message “Sorry! No photographer works in this area!” if system can not find any photographer in certain given radius. Use case ends
	3.2.6a	System	Display output message “Sorry! No photographer works in this area!” if system can not find any photographer in certain given radius. Use case ends
	3.2.4a	System	Return nothing if no suggested address match typed address
	3.3.7a	System	Display output message “Sorry! No photographer works in this area!” if system can not find any photographer in certain given radius. Use case ends
	4.1.2a	System	Don’t draw path on map if no direction found
<b>Post-condition</b>	Registered user finds out and connect to photographers		

### 2.3.3 Specification of use case “Filter photos”

**Table 4** Specified details of given use case “Filter photos”.

**Table 4** Specification of use case “Filter photos”

<b>Use case code</b>	UC001	<b>Use case name</b>	Filter photos
<b>Actor</b>	User, Registered user		

<b>Pre-condition</b>	Install successfully Photohub application		
<b>Main flow of events (Success)</b>	<b>No</b>	<b>Actor</b>	<b>Action</b>
	1.	User/ Registered user	Click on button “Filter”
	2.	System	Open filter screen
	3.	User/ Registered user	Select some tags by click on those
	4.1	User/ Registered user	Click on button “Confirm”
	4.1.1	System	Find all photographs that match to tag conditions and display results on screen
	4.1.2	User/ Registered user	Click on photograph to see detail of photograph
	4.2	User/ Registered user	Click on button “Reset”
	4.2.1	System	Remove all selected tags
<b>Alternative flow of events</b>	<b>No</b>	<b>Actor</b>	<b>Action</b>
	4.1.1a	System	Display message if there are no photographs found
<b>Post-condition</b>	List of photographs that match to tag conditions		

### 2.3.4 Specification of use case “Scan QR code for collection”

**Table 5** Specified details of given use case “Scan QR code for collection”.

**Table 5** Specification of use case “Scan QR code for collection”

<b>Use case code</b>	UC001	<b>Use case name</b>	Scan QR code for collection
<b>Actor</b>	User, Registered user, Photographer		

<b>Pre-condition</b>	Install successfully Photohub application		
<b>Main flow of events (Success)</b>	<b>No</b>	<b>Actor</b>	<b>Action</b>
	1.	User/ Photographer	Select Scan QR code function
	2.	System	Open scanner screen
	3.	User/ Photographer	Capture QR code
	4.	System	Check valid QR code
	5.	System	Retrieve a copy of collection corresponding to QR code and display on screen
	6.	Photographer/ Registered user	Click on button “Add collection”
<b>Alternative flow of events</b>	7.	System	Create a copy of collection and add to user’s collection
	<b>No</b>	<b>Actor</b>	<b>Action</b>
	4a	System	Display alert message that QR code is invalid if system can not find any collection that matches to QR code
<b>Post-condition</b>	7a	System	Display error message if add collection unsuccessfully
	List of photographs that match to tag conditions		

## 2.4 Non-functional requirements

In order to ensure that the application can run stably and meet the needs of users as well as overcome special cases, the project needs non-functional requirements such as performance, ease of use, feasibility, etc.

Mobile application PhotoHub and PhotoHub PG can run smoothly on both iOS devices and Android devices, with easy manipulation without troubles. Web application PhotoHub Admin can run well on web with high responsive.

User interface and User experience need to be basic for all of users can easy to see and do any action without troubles. By using consistent color scheme, gradient colors, interface become more friendly and impressive to user. By adding animations, application reduce lag and delay, and make the process that waiting data response is not become uncomfortable to user. All processes have effective output, any error need handled and alert to user.

Security is very important. All sensitive datas need to be encoded before transferring to anywhere on internet. Actor must log into application before using any feature that need authorization. All actions that manipulated by user need to verify the role to ensure that user have permission to do that actions. Database can easily backup and restore to prevent losing data.



# Chapter 3 Technology

## 3.1 ReactJS<sup>9</sup>

PhotoHub Admin is developed base on ReactJs.

Created by Facebook, ReactJS become popular library for front-end developer. ReactJS is a declarative, efficient, and flexible JavaScript library which uses for building user interfaces. It lets developers compose complex UIs from small and isolated pieces of code called “components”.

ReactJs has some unique core concepts. It works base on virtual DOM, JSX components, input properties, and props. Also, because of provided lifecycle in each React component, ReactJS is more powerful to developers.

The virtual DOM is node tree, fimilar to real DOM working within a web browser. It’s similar, but it’s absolutely virtual. There is a list of elements, attributes, content that exists as Javascript objects with properties. When we call a render function, it actually creates a node tree from that React component whether it's just one single component, or whether rendering child components as well. It lists out the whole tree. It also updates that same tree whenever data models are changed - whenever we update state or change anything within the component. React updates the real DOM by three steps. Whenever something has changed, the virtual DOM will re-render, then the difference between the old virtual DOM and new virtual DOM will be calculated. After that, from there the real DOM will be updated based on these changes. Instead of constantly having to work with the real DOM which is very expensive, everything is handled virtually.

React lets us split the UI into independent reusable pieces which call “Components”. Components are like JavaScript functions, but it is like a container that contains group of code representing for a front-end component. We have an amount of input, set the props, and then we return the React elements. We always return a render function that has the elements we want it to display. Every component will have a render function. We’ll see more clearly when comparing to pure javascript syntax

---

<sup>9</sup> <https://reactjs.org/>

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

In ReactJS, we can wrap Welcome with powerful control in class component or function component

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

There are so many open-source platform for making the front-end web application development easier as ReactJS, Angular, VueJS. ReactJS is very competitive technology because of some benefits:

- i) Simplicity: JSX is syntax mix up between HTML with javascript, therefore we can easily learn.
- ii) Easy to learn: Anyone with a basic previous knowledge in programming can easily understand React while Angular are referred to as “Domain-specific Language”, implying that it is difficult to learn them. With React, you just need a basic knowledge of CSS and HTML also basic of javascript.
- iii) Native approach: React is not only for building web application, but also creating mobile application (React Native). At a same time, we can make IOS, Android, and Web applications.
- iv) Data binding: ReactJS uses one-way data binding and an application architecture called Flux to control the flow of data to components. It's easier to debug self-contained components.
- v) Testability: ReactJS applications are very easy to test. We can manipulate with the state or props we pass to the ReactJS component and take a look at the output and triggered actions, events, functions, etc.

## 3.2 React Native<sup>10</sup>

PhotoHub and PhotoHub PG are developed base on React Native

React Native was also developed by Facebook to address its need for a dynamic and high performing User Interface. React Native is very similar to React JS with the same syntax

---

<sup>10</sup> <https://reactnative.dev/>

and same workflow. React Native compiles to native app components which makes it possible to build native mobile applications. While ReactJS is the base abstraction of React DOM for the web platform, React Native is still the base abstraction but of React Native. Therefore, the syntax and workflow remain similar, but the components are different.

There are some business advantages of using React Native that make React Native is very competitive compare to other language:

- i) React Native comes with Native Modules and Native component that improve performance.
- ii) React Native build the same application for iOS and Android with same common logic layer.
- iii) React Native is very appropriate for web developers because of syntax mixed with HTML and javascript. All you need to know is Javascript, platform API, some native UI elements.
- iv) React Native bring high speed, responsiveness, and ability of web app development along with effectual processing and best user experience to the hybrid space, to provide your users with a native app experience.

### 3.3 Firebase<sup>11</sup>

Firebase is used to develop backend server, store data and database in PhotoHub system.

Owned by Google, Firebase is a complete package of products that allows us build web and mobile apps, improve the app quality and help your clients grow their business. Firebase by Google can be used for the following:

- i) Firebase manages all data real-time in the database which can be Realtime Database or Cloud Firestore. So, the exchange of data to and from the database is easy and quick. Hence, Firebase is very appropriate to develop mobile apps such as live streaming, chat messaging, etc.
- ii) Firebase allows syncing the real-time data across all the devices Android, iOS, and the web without refreshing the screen by easy ways.
- iii) Firebase offers integration to Google Ads, AdMob, DoubleClick, Play Store, Data Studio, BigQuery, etc to make app development with efficient and accurate management and maintenance.

---

<sup>11</sup> <https://firebase.google.com/>

- iv) Everything from databases, analytics to crashing reports are included in Firebase. Therefore, the app development teams can stay focused on improving the user experience.

Firebase have many benefits. We can create application without backend server, sync real time data in the application, use NoSQL database that is faster, push notification, cloud storage, etc.

Firebase provide a powerful system with large of major features: Firebase Machine Learning, Firebase Authentication, Firebase Realtime Database, Cloud Firestore, Firebase Push Notification, Firebase Storage, Firebase Cloud Functions, Firebase Hosting, Firebase Cloud Storage, Firebase Analytics, Firebase Crash Reports, Firebase Performance, etc.

PhotoHub system are using:

- i) Cloud Firestore as NoSQL database to store data.
- ii) Cloud Functions for Firebase as a backend server using Nodejs with express library to create RESTful API and access to Firestore and Firebase Storage.
- iii) Firebase Storage to storage file as images.
- iv) Firebase Hosting to deploy PhotoHub Admin web application and cloud functions as backend server of application.

### 3.4 Cloud Firestore<sup>12</sup>

Google introduces Cloud Firestore as a NoSQL database which is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud Platform. It keeps your data in sync across client apps through realtime listeners and offers offline support for mobile and web, therefore you can build responsive apps that work regardless of network latency or Internet connectivity. Firestore supports many types of data as String, Integer, Array, etc.

There are some benefits when using Cloud Firestore:

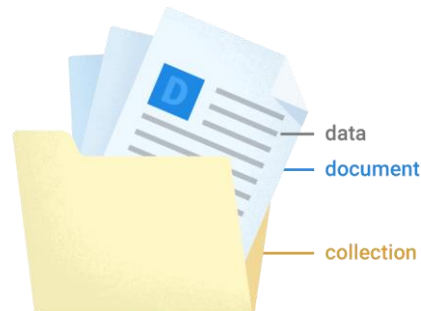
- i) Works not just online, when the users are offline, it still stores the changes and sync it whenever the users come online.
- ii) Cloud Firestore integrates well with the Firebase Authentication that helps build simple and intuitive security model for the developers.
- iii) We don't need to use servers to build the apps because the Cloud Firestore allows it with mobile and web SDKs

---

<sup>12</sup> <https://firebase.google.com/docs/firestore>

- iv) Real-time data syncing is possible across any device with Cloud Firestore

Cloud Firestore is a NoSQL, organize data in a document-oriented database. There are no tables or rows like SQL database. As an alternative, you store data in documents, which are organized into collections.



**Figure 15** Store data in Cloud Firestore

Each document includes a set of key-value pairs. All documents must be stored in collections like a list of documents. Documents can contain subcollections and nested objects, both of which can contain primitive fields like strings or complex objects like lists. Collections and documents are created implicitly. We can simply assign data to a document within a collection, if either the collection or document does not exist, Cloud Firestore creates it.

### 3.5 Cloud Storage<sup>13</sup>

Google introduces Cloud Storage as a powerful, simple, and cost-effective object storage service build for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for Firebase apps, regardless of network quality. We can store images, audio, video, or other user-generated content. On the server, we can use Google Cloud Storage to access the same files.

There are some benefits when using Cloud Storage:

- i) Automatically pause and resume functionalities for uploading and downloading the online or offline files.
- ii) Saves users time and improves the bandwidth.
- iii) Integrates with Firebase Authentication to offer a secured environment for the end users.

---

<sup>13</sup> <https://firebase.google.com/docs/storage>

### 3.6 Cloud Functions and NodeJs<sup>14</sup>

Google introduces Cloud Functions as a serverless framework that lets us automatically run backend code in response to events triggered by Firebase features and HTTPS requests. JavaScript or TypeScript code is stored in Google's cloud and runs in a managed environment. There's no need to manage and scale servers.

After writing and deploying functions, Google's servers begin to manage the function immediately. The function can be fired directly with an HTTP request, or in the case of background functions, Google's servers will listen for events and run the function when it is triggered.

When the code is running in the cloud instead of the traditional approach, you can come across various advantages:

- i) There is no necessity for you to run your own server and maintain it.
- ii) In case of the back-end code, an isolated code base can be used.
- iii) The billing is done only for the code's real executing time.
- iv) High scalability is possible in case of the cloud infrastructure.

In order to use Cloud Functions, NodeJs environment need to be installed. And because of Cloud Function run on NodeJs environment, we can use all libraries that managed by npm. PhotoHub backend server uses Cloud Functions with Express library to create restful API to communicate to client application, and uses Firebase-admin for accessing to Cloud Firestore to manage datas of system, also accessing to Cloud Storage to manage files.

### 3.7 Geohash

PhotoHub system using geohash to query all nearby photographer in certain given radius.

Geohash<sup>15</sup> is a public domain geocode which encodes a geographic location into a short string of letters and digits. It is a hierarchical spatial data structure which subdivides space into buckets of grid shape, which is one of the many applications of what is known as a Z-order curve, and generally space-filling curves. Geohashes offer properties like arbitrary precision and the possibility of gradually removing characters from the end of the code to reduce its size (and gradually lose precision). Geohashing guarantees that the longer a shared

---

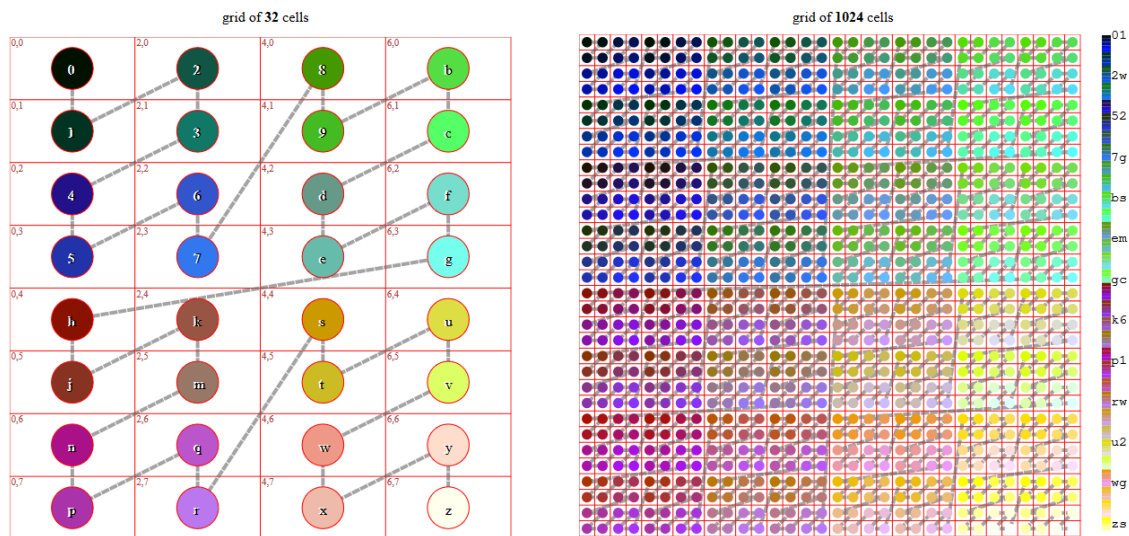
<sup>14</sup> <https://firebase.google.com/docs/functions>

<sup>15</sup> <https://en.wikipedia.org/wiki/Geohash>

prefix between two geohashes is, the spatially closer they are together. The reverse of this is not guaranteed, as two points can be very close but have a short or no shared prefix.

The geometry of the Geohash have a mixed spatial representation:

- i) Geohashes with 2, 4, 6, ... e digits (even digits) are represented by Z-order curve in a "regular grid" where decoded pair (latitude, longitude) has uniform uncertainty
- ii) Geohashes with 1, 3, 5, ... d digits (odd digits) are represented by "H-order curve". Latitude and longitude of the decoded pair has different uncertainty (longitude is truncated)



**Figure 16** Geometrical representation

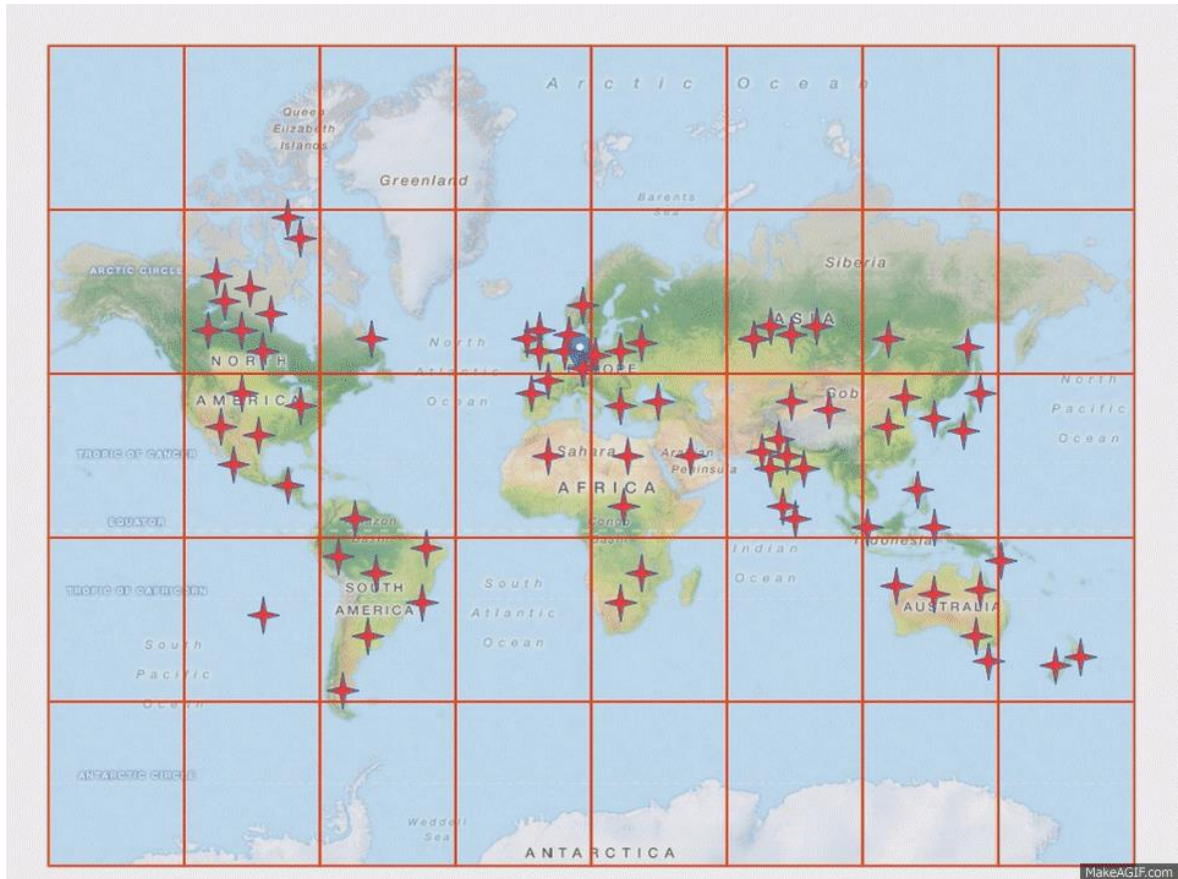
Using geohash, we will divide whole map to small boxes which is index from 0 to z. From each small box, we continue to divide each box to smaller boxes also indexed from 0 to z. We have an example of calculation the geohash: For a point in NYC, say (40.75798, -73.991516), distance: 800 Meters and Geohash length: 12.

- i) NorthWest: dr5ru j4477kd
- ii) SouthWest: dr5ru 46ne2ux
- iii) SouthEast: dr5ru 6ryw0cp
- iv) NorthEast: dr5ru mpfq534

From these Geohashes, calculate the Query Bounding Box (MBR) Prefix: dr5ru. This would give you the coarser Geohash which completely contains our MBR and hence the query region. In other words, all points indexed by dr5ru, yielding with 32 GeoHashes from dr5ru0 - dr5ruz.



To find the exact grids (or) GeoHashes that correspond to our Query Circle (Square (MBR) to be precise), we should pick from these 32 GeoHashes by representing a recurring (4X8) Matrix using 2D Array. In our example: we get dr5ru + J, M, H, K, 5, 7, 4, 6. All these GeoHashes represent the points that are within 800 meters from the Central Query Point, Except very few GeoHashes, which could not be avoided, because of considering MBR instead of a perfect circle.



**Figure 17** Illustrate geohash on map

### 3.8 QR code

PhotoHub application uses QR code<sup>16</sup> to share collection to other users.

A QR code (short for "quick response" code) is a type of barcode that contains a matrix of dots. It can be scanned using a QR scanner or a smartphone with built-in camera. Once scanned, software on the device converts the dots within the code into numbers or a string of characters. For example, scanning a QR code with your phone might open a URL in your phone's web browser.

<sup>16</sup> [https://en.wikipedia.org/wiki/QR\\_code](https://en.wikipedia.org/wiki/QR_code)



All QR codes have a square shape and include three square outlines in the bottom-left, top-left, and top-right corners. These square outlines define the orientation of the code. The dots within the QR code contain format and version information as well as the content itself. QR codes also include a certain level of error correction, defined as L, M, Q, or H. A low amount of error correction (L) allows the QR code to contain more content, while higher error correction (H) makes the code easier to scan.



**Figure 18** Example of QR code in PhotoHub application

QR codes have two significant benefits over traditional UPCs – the barcodes commonly used in retail packaging. First, since QR codes are two-dimensional, they can contain significantly more data than a one-dimensional UPC. While a UPC may include up to 25 different characters, a 33x33 (version 4) QR code, can contain 640 bits or 114 alphanumeric characters. A 177x177 (version 40) QR code can store up to 23,648 bits or 4,296 characters.

Another advantage of QR codes is that they can be scanned from a screen. Standard UPC scanners use a laser to scan barcodes, which means they typically cannot scan a UPC from a screen (like a smartphone). QR scanners, however, are designed to capture 2D images printed on paper or displayed on a screen. This makes it possible to use a QR code on your smartphone as a boarding pass at the airport or as a ticket for an event.

# Chapter 4 Software development and deployment

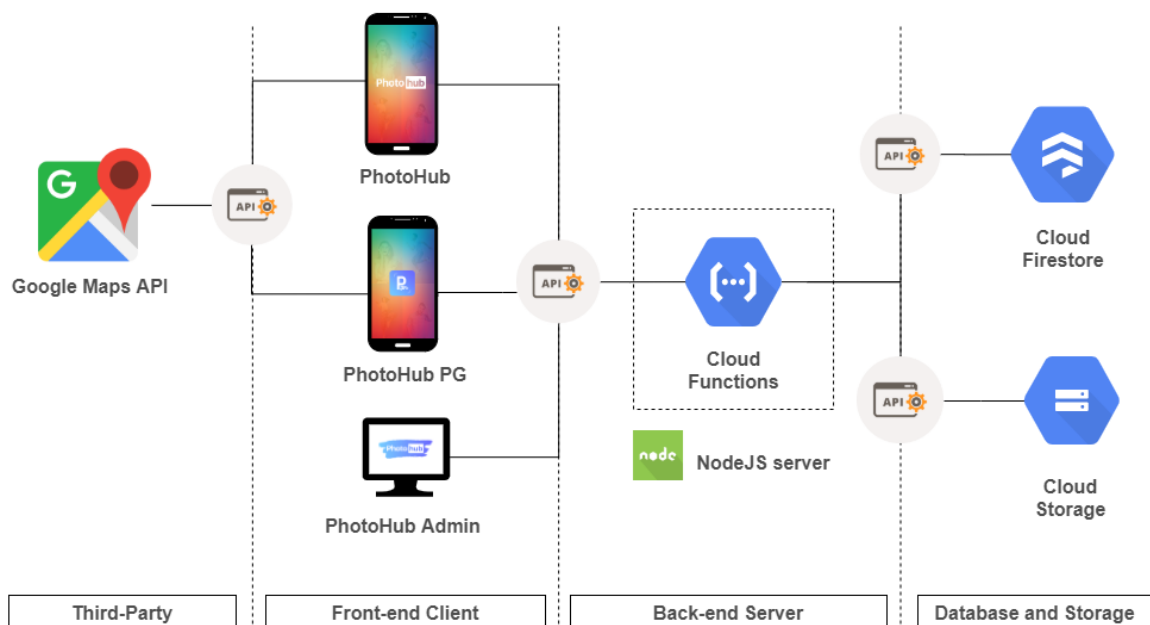
## 4.1 Architecture Design

### 4.1.1 Architecture Design

PhotoHub system contains three applications: PhotoHub, PhotoHub PG, PhotoHub Admin. These three applications are built on different platform with different purposes, but they also have some similar behaviors, functions. Therefore, we need to build a system that can develop flexible on different platforms without any conflict.

#### 4.1.1.1 System Architecture Design

PhotoHub system uses Client-Server architecture to separate development of system in different parts.



**Figure 19** System architecture design

A client-server application is a piece of software that runs on a client interface and makes requests to a remote server-side. Many such applications are written in high-level visual

programming languages where UI, forms, and most business logic reside in the client-side application. Often such applications are database applications that make database queries to a remote central database server.

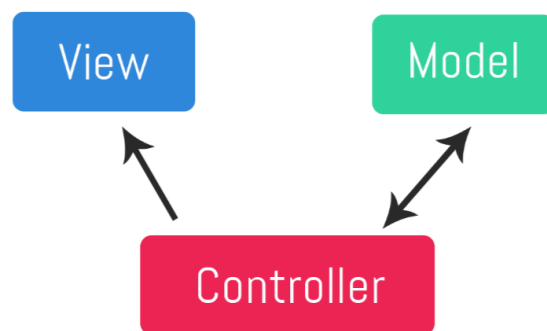
A client-server application can be cross platform if it is written in a cross platform language, or it can be platform specific because client-side and server-side work absolutely independently. In the case of a cross platform language there is an advantage that the application can potentially provide a different user interface that is native in appearance to the different OS or platform environment it is running under.

PhotoHub system uses Client-Server architecture, communicating between components through Restful API make the systems become more easier to develop, expand and maintain. Moreover, because the server is built to create Restful API, client-side application can be developed on multiple platforms in easy way. **Figure 19** presents how system architecture looks, then we can understand the flow of connections between each part of system.

More detail, we develop server-side and client-side independently. Server-side uses MVC architecture, and Client-side uses Flux architecture. We will talk about that in next section.

#### 4.1.1.2 MVC pattern

We built back-end server following MVC pattern.



**Figure 20** MVC pattern

MVC stands for Model, View and Controller. MVC separates application into three components - Model, View and Controller.

- i) **Model:** Model represents shape of the data and business logic. It maintains the data of the application. Model objects retrieve and store model state in a database. Model is a data and business logic.
- ii) **View:** View is a user interface. View display data using model to the user and also enables them to modify the data. View is a User Interface.

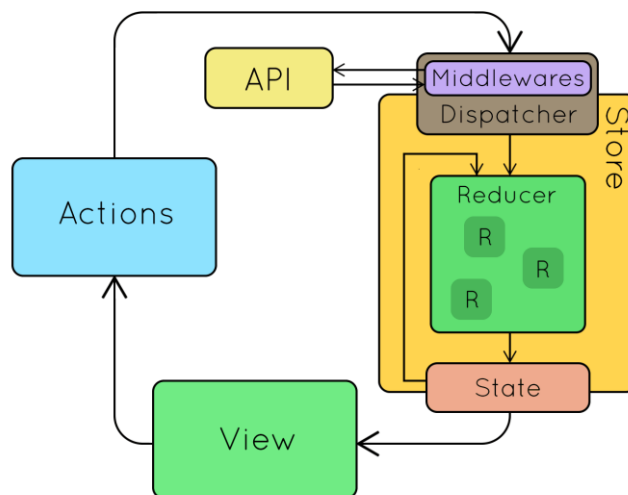
- iii) **Controller:** Controller handles the user request. Typically, user interact with View, which in-turn raises appropriate URL request, this request will be handled by a controller. The controller renders the appropriate view with the model data as a response. Controller is a request handler.

#### 4.1.1.3 Flux pattern

We built front-end client applications following flux pattern.

Flux is the application architecture that Facebook uses for building client-side web applications. React is technically considered to be the "View" in the MVC structure but due to complications with MVC, Facebook also released their own architecture called Flux. Instead of having two-way data binding from the controller to the view, Flux uses one-way data flow. Two-way data binding was difficult to debug as it could have many side effects on your application. Using one-way data flow allowed for easier debugging and scaling with larger applications. Flux is made up of four different parts:

- i) **Actions:** An object with data about the action that just occurred on the application.
- ii) **Store:** Contains the application state (data) and logic.
- iii) **Dispatcher:** Processes action and their callbacks.
- iv) **View:** Renders out to data to the screen and listens to changes in the store to re-render.



**Figure 21** Flux pattern

Notice how the arrow is never going two directions - it's always one-way.

### 4.1.2 General Design

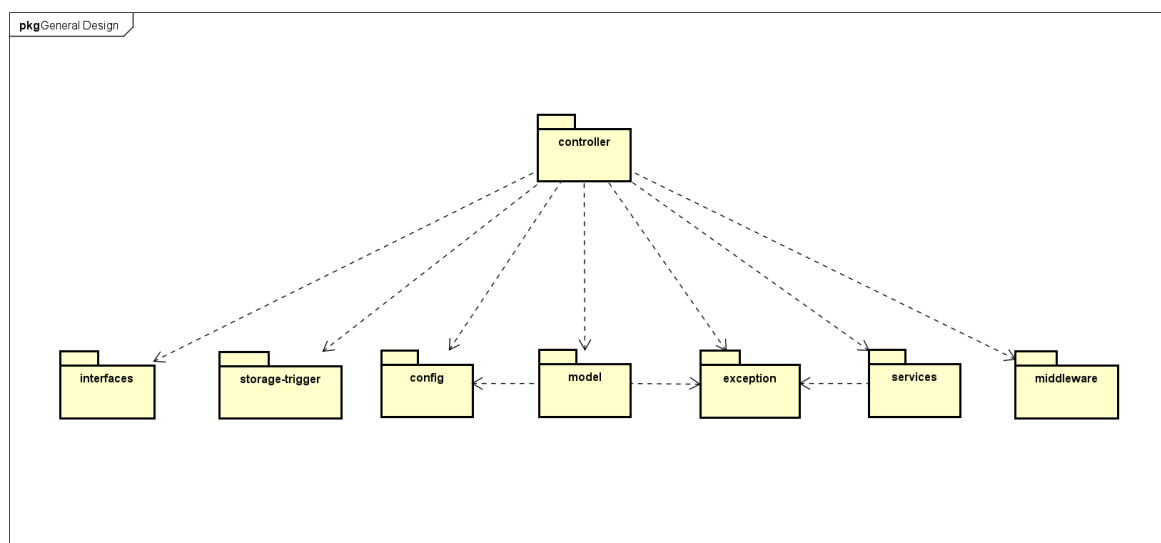
Designing on back-end server, project structures package folders as **Figure 22** below.

Controller is package containing controller class that navigate the application in the appropriate flow. All requests from client will handle by controller which will excute list of actions corresponding to what client request want, then render a JSON format and send to client. Moreover, controller can use model to access to database to read or write on database, also access to storage for file processing purpose.

Model is package that communicate with firestore database, it has permission to read, write, update or delete data, also run query and return value corresponding the query.

Storage-trigger is package that communicate with firebase storage, it has permission to access and do some actions on file inside storage.

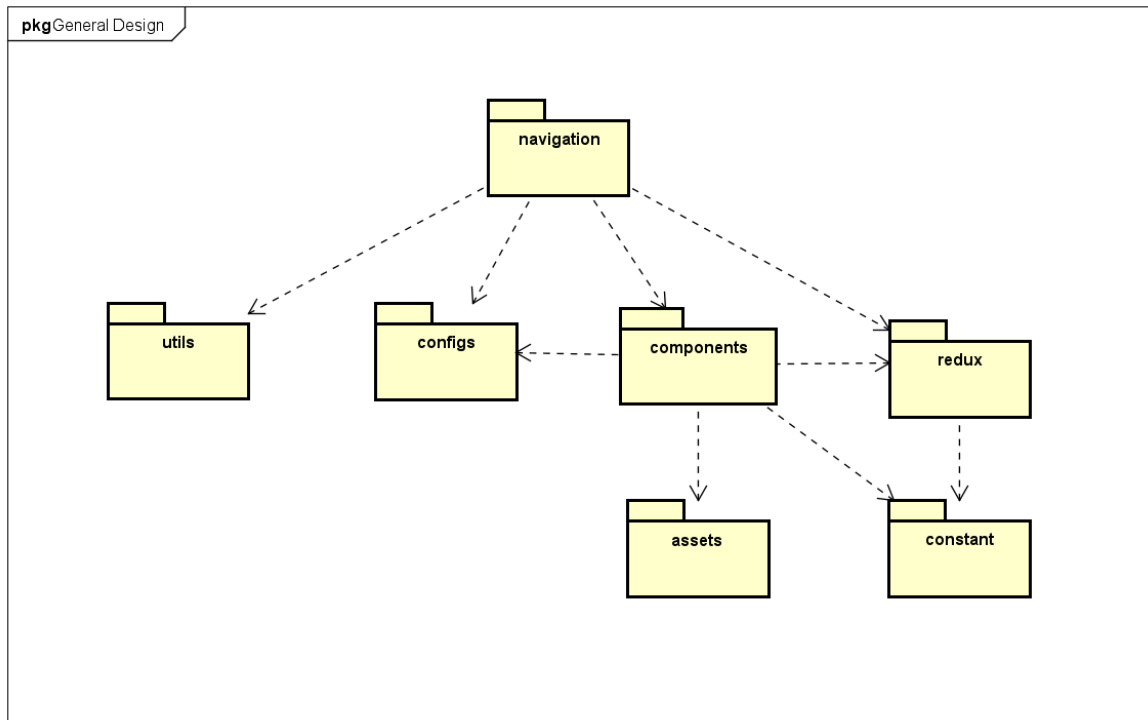
Interface, config, exception, services, and middleware are package supporting developing project. They provide methods that can be very helpful for the coding processing. For example, exception is package that provide how error is handled when some actions get error. There will many bugs happen due to different reasons, exception package contains exception class that handle each unwanted logic in application.



**Figure 22** General Design of server-side

Designing on PhotoHub and PhotoHub PG – front-end client applications, project structures package folders as **Figure 23** below.

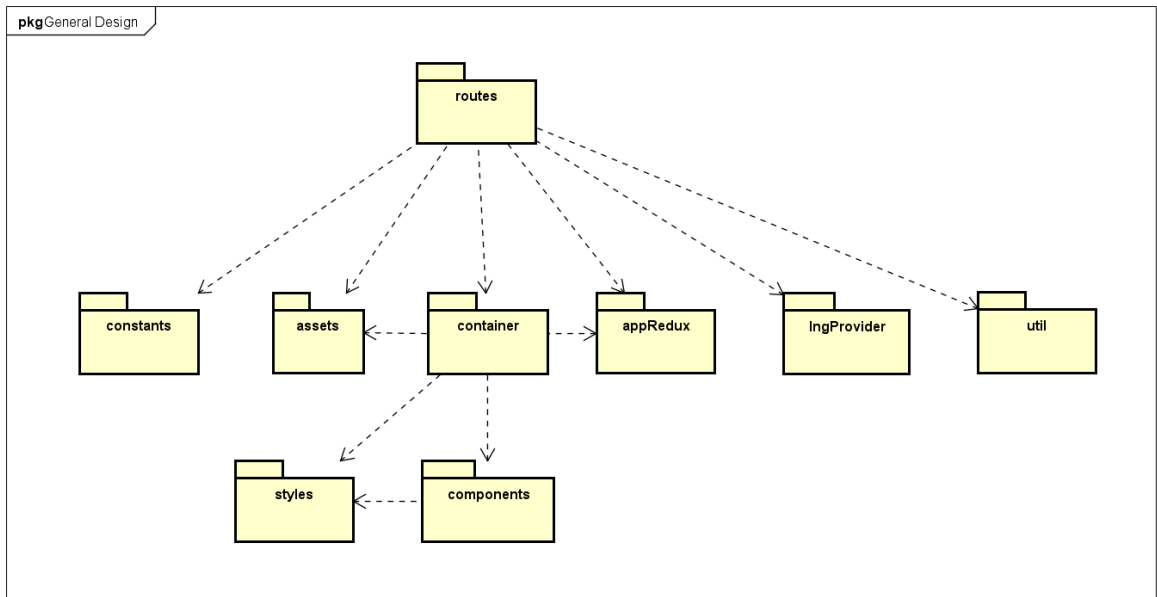
Navigation is package that navigate the application pages corresponding to functions user want to use. Component is package that contains all interfaces that help users interact with application. Utils package includes all functions helping developer during coding processing. Redux package includes subpackage such as: “Action”, “Reducer”, “Store” for managing state purpose of application. Assets package contains all file such as image used in application.



**Figure 23** General design of PhotoHub and PhotoHub PG front-end client application

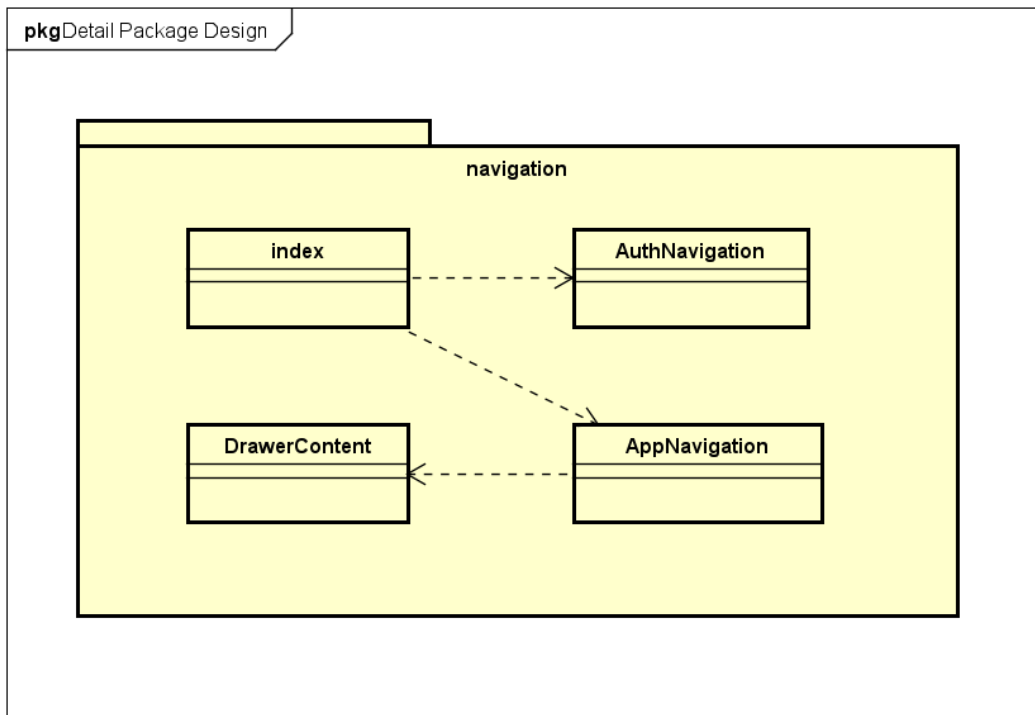
Designing on PhotoHub and PhotoHub PG – front-end client applications, project structures package folders as **Figure 24** below.

Routes is package that navigate the application pages corresponding to functions user want to use. Container is package that contains all interfaces that help users interact with application. Utils package includes all functions helping developer during coding processing. AppRedux package includes subpackage such as: “Action”, “Reducer”, “Store” for managing state purpose of application. Assets package contains all file such as image used in application. Styles package contains global CSS file can be used anywhere in application.



**Figure 24** General design of PhotoHub Admin front-end client application

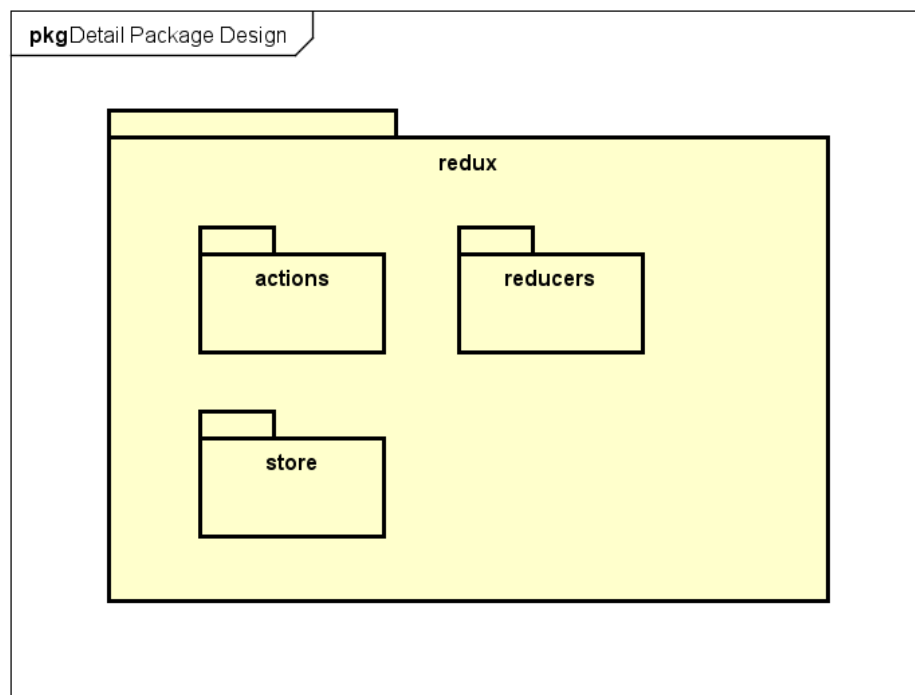
#### 4.1.3 Detail Package Design



**Figure 25** Detail design of package “navigation”

**Figure 25** describes structure of “navigation” package. It contains components: Index.js, AuthNavigation.js, DrawerContent.js, AppNavigation.js. Each component has its own function:

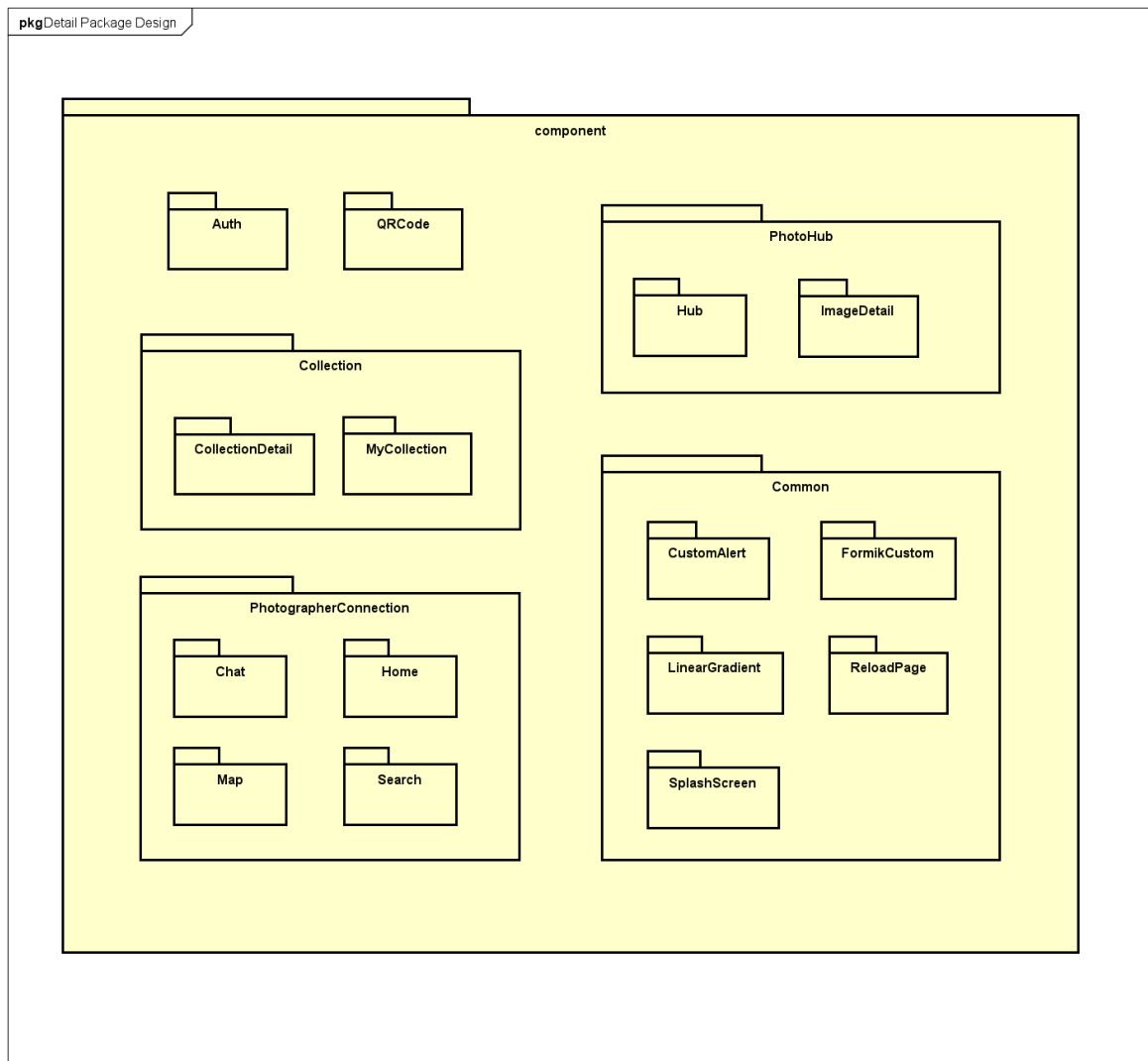
- i) Index is the root of navigation described how flow of navigation start.
- ii) AuthNavigation presents interfaces that helps user can login, register a new account.
- iii) AppNavigation presents all main functions of application.
- iv) DrawerContent presents a navigator to navigate each function of application.



**Figure 26** Detail design of package “redux”

**Figure 26** describes structure of “redux” package. It contains subpackages: actions, reducers, store. Redux package using Flux architecture plays a role as monitor holding all data and state when application is running.



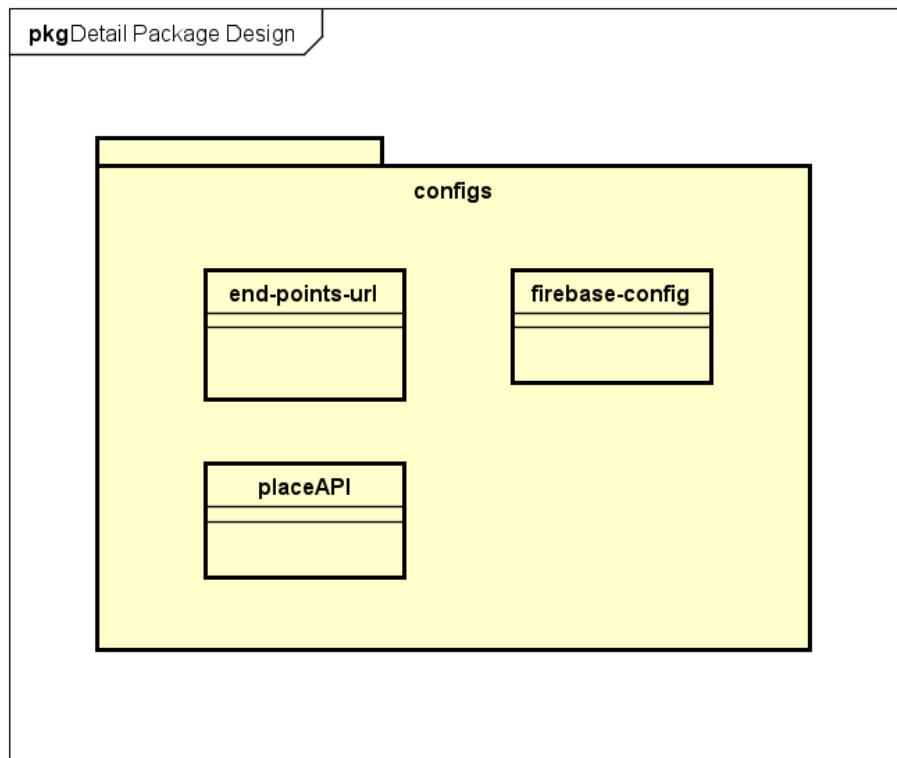


**Figure 27** Detail design of package “component”

**Figure 27** describes structure of “component” package. It contains subpackages: Auth, QRCode, PhotoHub, Collection, PhotographerConnection, Common. Each subpackages has its own function:

- i) Auth presents an interface that help user can interact with application to log in or create new account.
- ii) PhotoHub contains some subpackages that help user can interact with application to manipulate with photograph and filter photographs by tags.
- iii) Collection contains some subpackages that help user can interact with application to manipulate with collection: create, update, delete collection and image inside each collection.
- iv) PhotographerConnection contains some subpackages that help user can interact with application to search nearby photographer and connect to photographer.

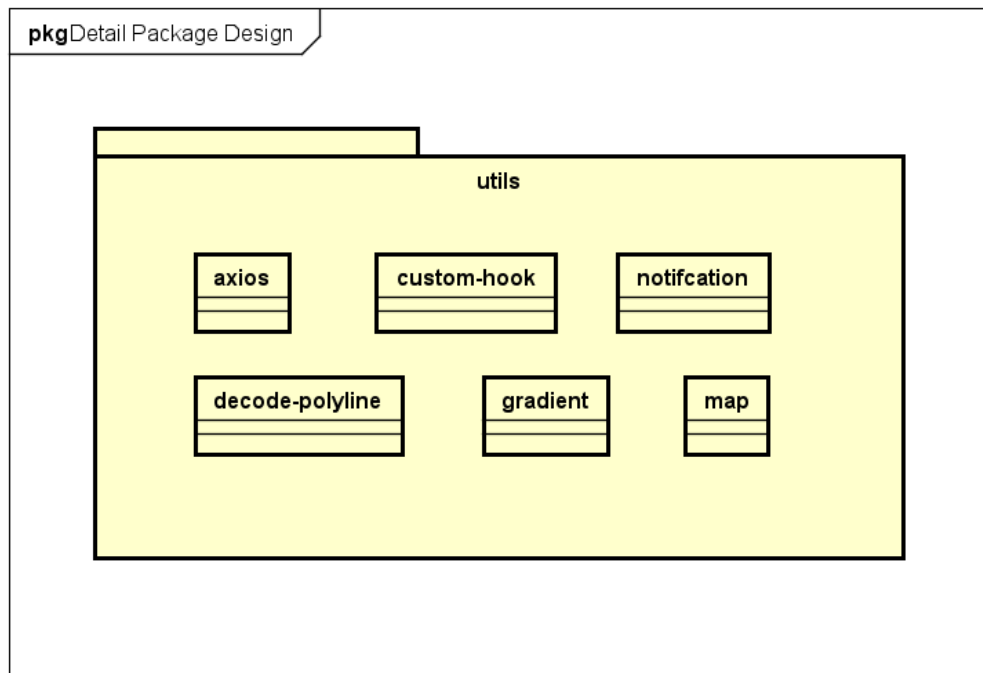
- v) QRCode presents an interface that help user can interact with application to retrieve collection corresponding to QR Code.
- vi) Common contains all common component UI supported for other components in application.



**Figure 28** Detail design of package “configs”

**Figure 28** describes structure of “configs” package. It contains components: end-points-url.js, firebase-config.js and placeAPI.js. Each component has its own function:

- i) End-points-url contains all APIs that client can use to request to server or communicate with third-party.
- ii) Firebase-config contains configuration of installing firebase in application.
- iii) placeAPI contains all keys used for security purpose in search photographer functions.



**Figure 29** Detail design of package “utils”

**Figure 29** describes structure of “configs” package. It contains components: axios.js, custom-hook.js, notification.js, decode-polyline.js, gradient.js, map.js. Each component has its own function:


- i) Axios contains configuration of initial setting on each request from client to server.
- ii) Notification help user can create and receive push notifications in chat functions.
- iii) Decode-polyline contains functions that transform encoded path to real path which display on map view, especially in finding direction function.
- iv) Gradient contains functions that help developer can play around with gradient color, make application become more impressive.
- v) Map contains functions supported when manipulating with map view.

## 4.2 Detail Design

### 4.2.1 User interface

PhotoHub system contains three application with different user interfaces. Detail informations of user's interface described as **Table 6**.

**Table 6** User interface information

	PhotoHub	PhotoHub GP	PhotoHub Admin
<b>Screen resolution</b>	Appropriate for screen with a resolution of HD 720p or higher		Appropriate for screen with resolution of 1360px or higher
<b>Screen size</b>	5.1 inches or higher		14 inches or higher
<b>Color</b>	Fully support RGB color scheme, or RGBA color scheme, and gradient color.		Fully support RGB color scheme, or RGBA color scheme.
<b>Element design</b>	Make use of borders, shadows with appropriate color.		
<b>Font</b>	Segoe UI	Segoe UI	Roboto
<b>Display error</b>	Display alert modal		Display alert
<b>Language</b>	English		English
<b>Logo, brand identity</b>			

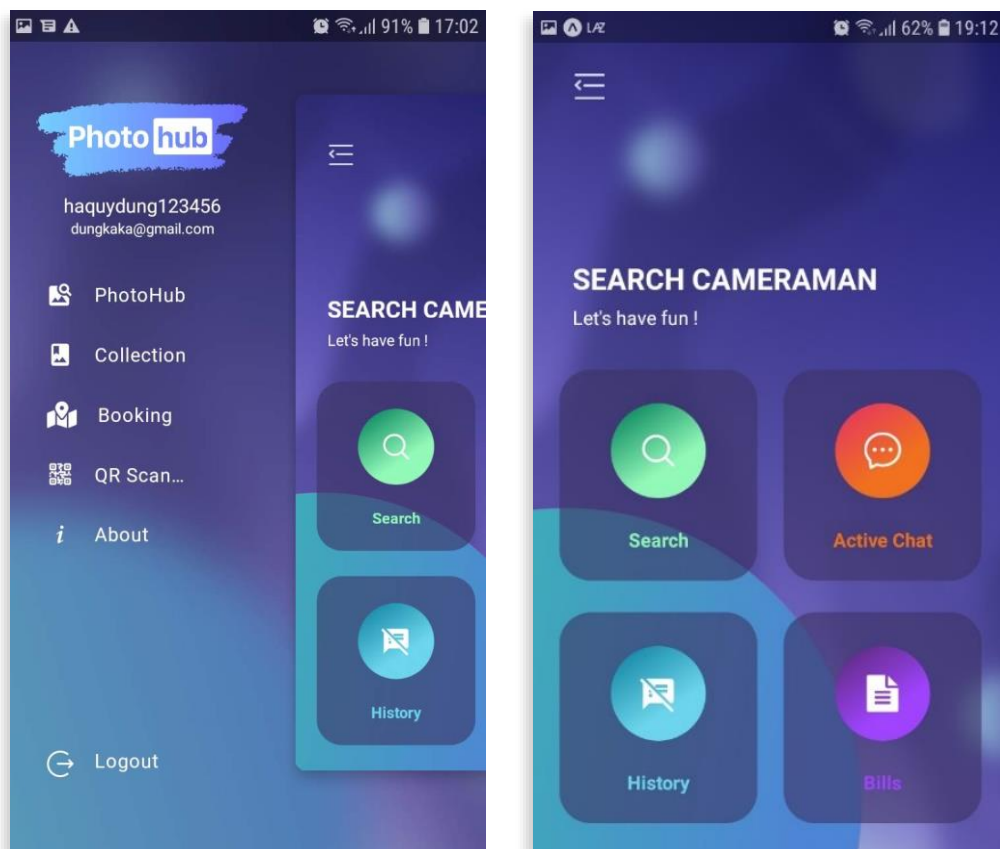
Color pattern used in application:



Gradient color pattern used in application:



**Figure 30, Figure 31 and Figure 32** show some interfaces in applications:



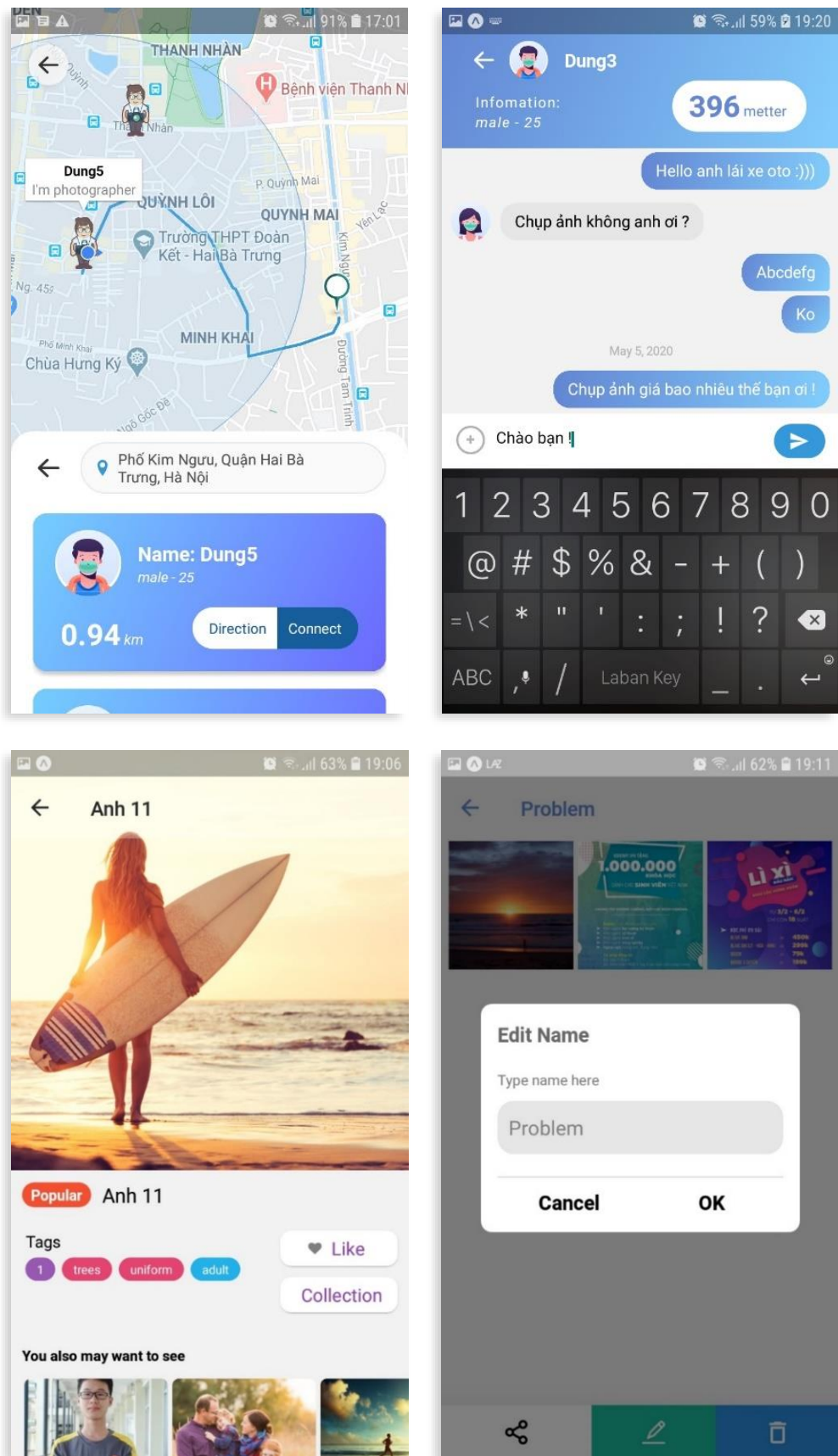
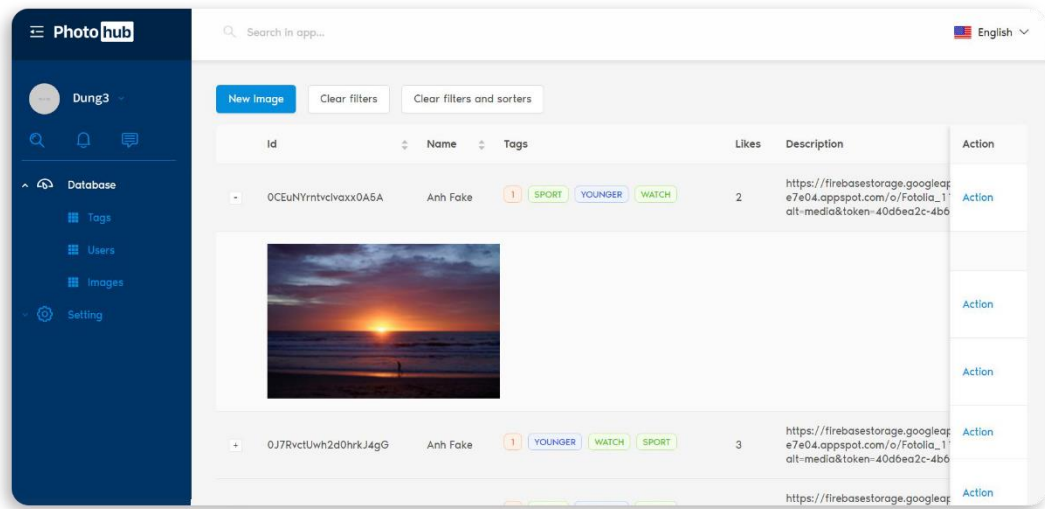


Figure 30 Photohub application screens



**Figure 31** Photohub Admin web application screens



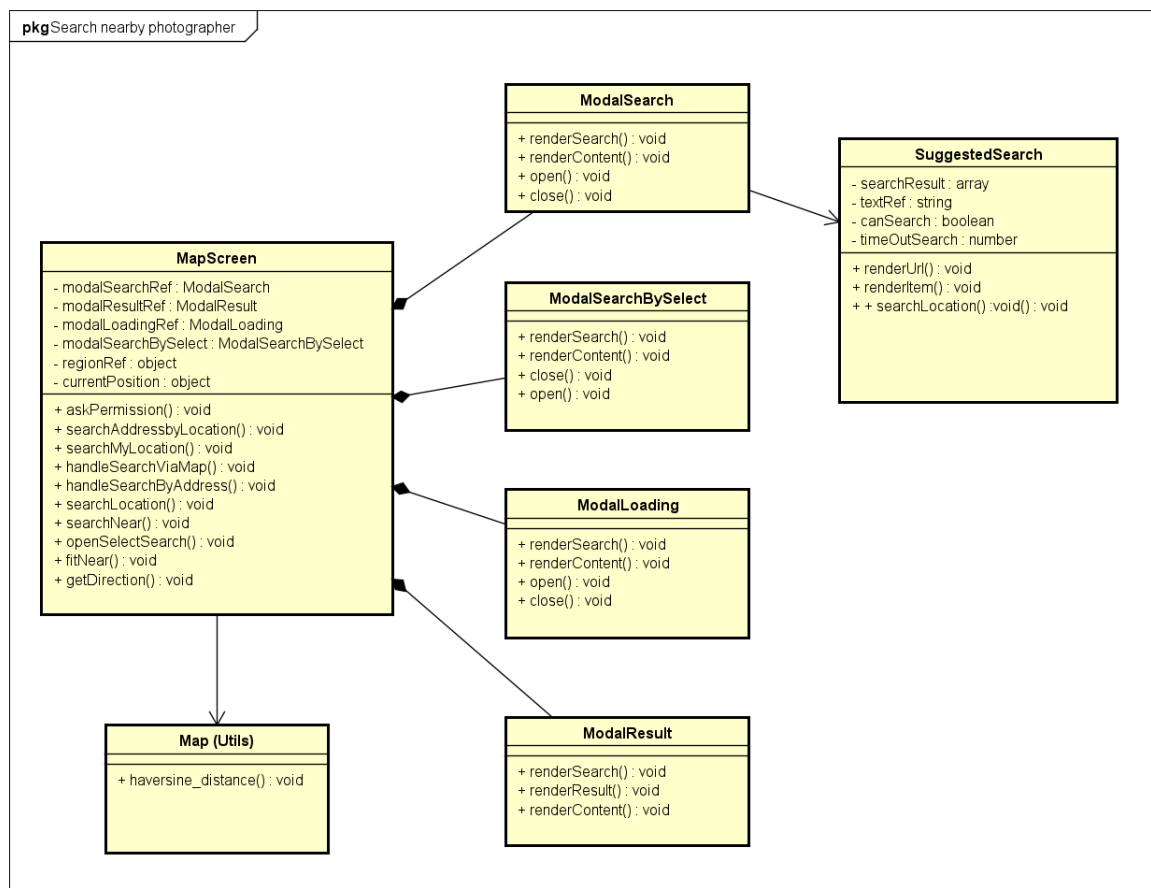
**Figure 32** Photohub PG application screens

## 4.2.2 Class Design

PhotoHub system follows client-server architecture. Therefore, each use case will be separated into two different flows of work. One is developed on backend-side and one is developed on client-side.

For example, class diagram and sequence diagram for use case “Search nearby photographer by suggested location” is presented by two parts.

Firstly, **Figure 33** describes class diagram of use case “Search nearby photographer by suggested location” which developed on client-side.



**Figure 33** Class diagram of use case “Search nearby photographer by suggested location”

**Table 7** describes in detail function of each method in the class MapScreen.

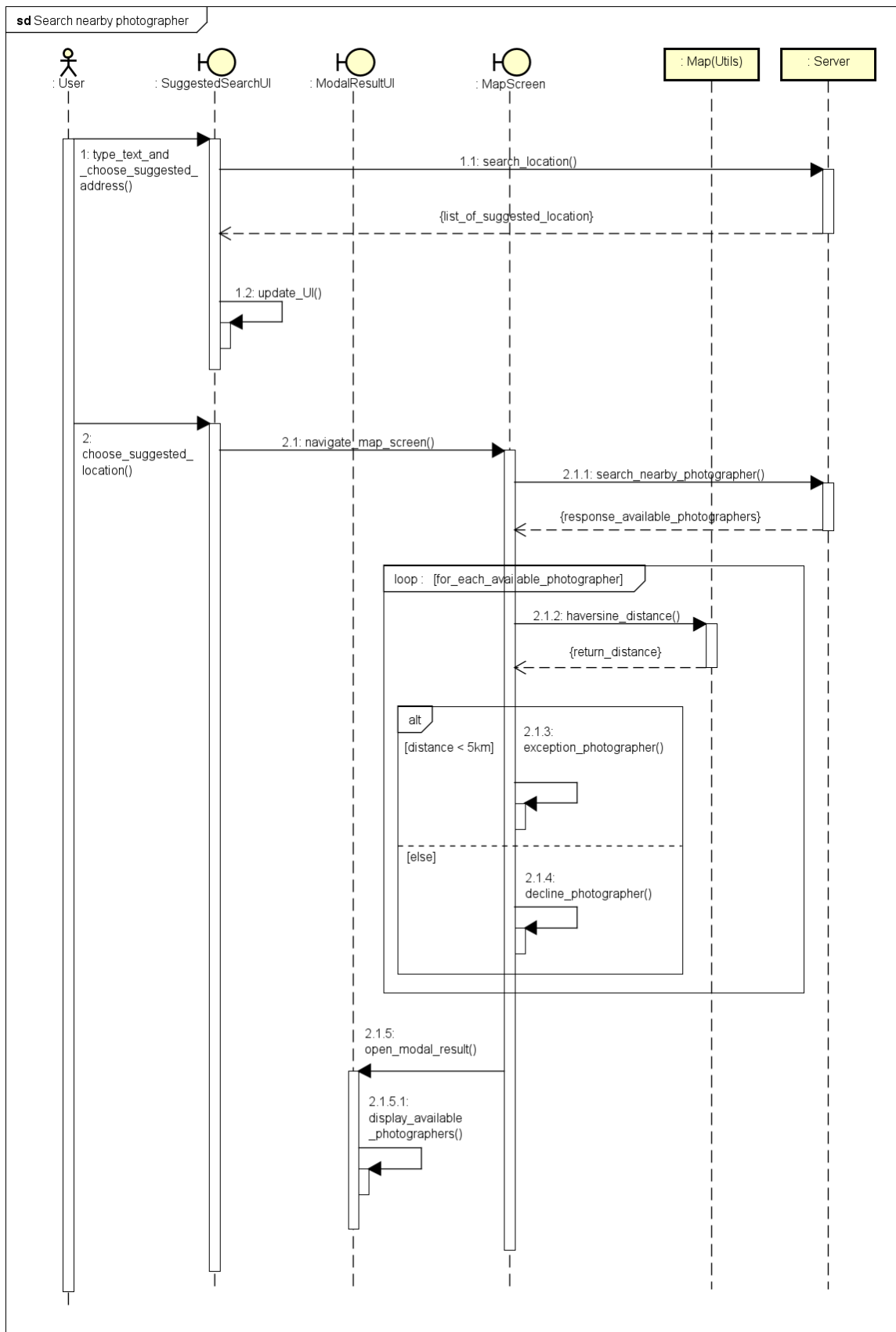
**Table 7** Specification of class MapScreen

Method	Action
askPermission()	Ask permission to access location of mobile phone



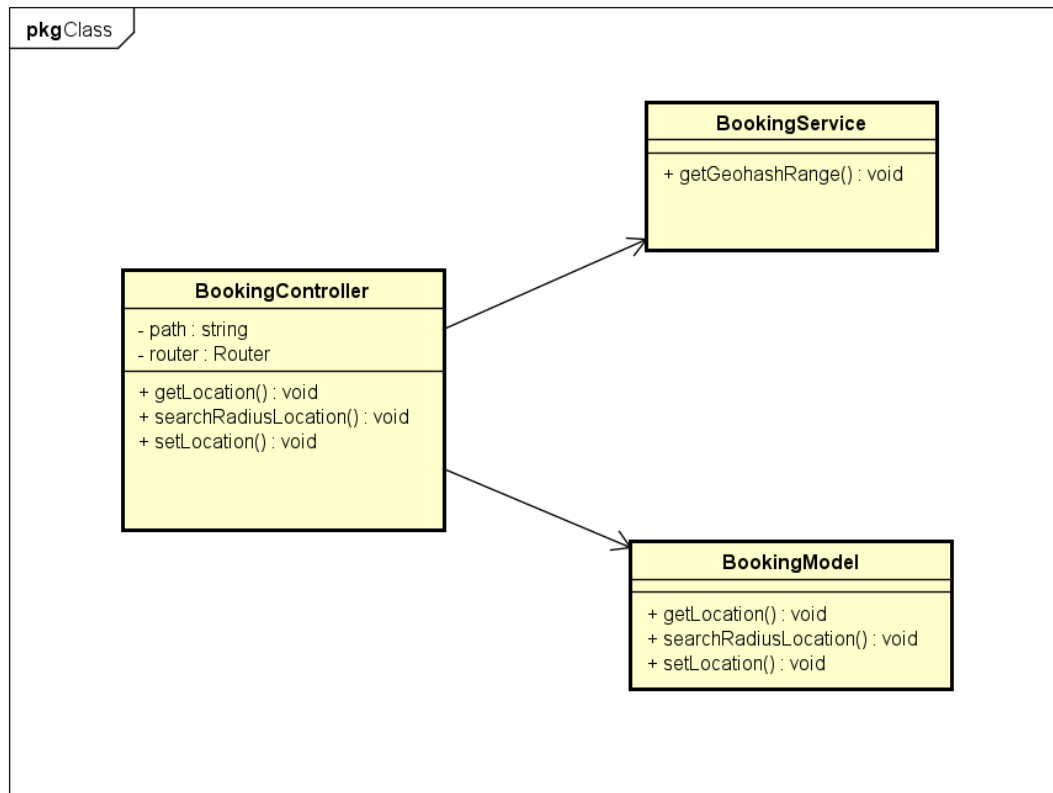
searchAddressByLocation()	Retrieve address name corresponding to coordinate of a location
searchMyLocation()	Handle searching nearby photographers through device's location
handleSearchViaMap()	Handle searching nearby photographers via selecting certain location on map
handleSearchByAddress()	Handle searching nearby photographers through suggested location
searchLocation()	Retrieve coordinate of a location corresponding to address name
searchNears()	Handle searching all photographers in certain radius and display results to user.
openSelectSearch()	Open modal to choose certain location and confirm that
fitNear()	Scale screen to fit result of photographers
getDirection()	Find direction from user

Sequence diagram of use case “Seach nearby photographer by suggested location” is described as **Figure 34**. In the beginning of flows, user need to enter string and choose suggested location which displayed on screen. Then, method `searchLocation()` in `SuggestedSearch` will be triggered in order to retrieve list of suggested locations from through `placeAPI`. Results of suggested locations will display on screen. After that, user will choose one of suggested locations and automatically move on `MapScreen`. `MapScreen` will call method `searchNears()` to send request to server and receive list of available photographers within certain radius. Subsequently, each available photographer will be checked again if it is in certain radius by using `haversin_distance` from `Map(Utills)`. Finally, all photographers that satisfy distance condition will be displayed on `ModalResult`.



**Figure 34** Sequence diagram of use case “Search nearby photographer by suggested location”

Second, **Figure 35** scribes class diagram of use case “Search nearby photographer by suggested location” which developed on server-side.



**Figure 35** Class diagram of use case “Search nearby photographer by suggested location”

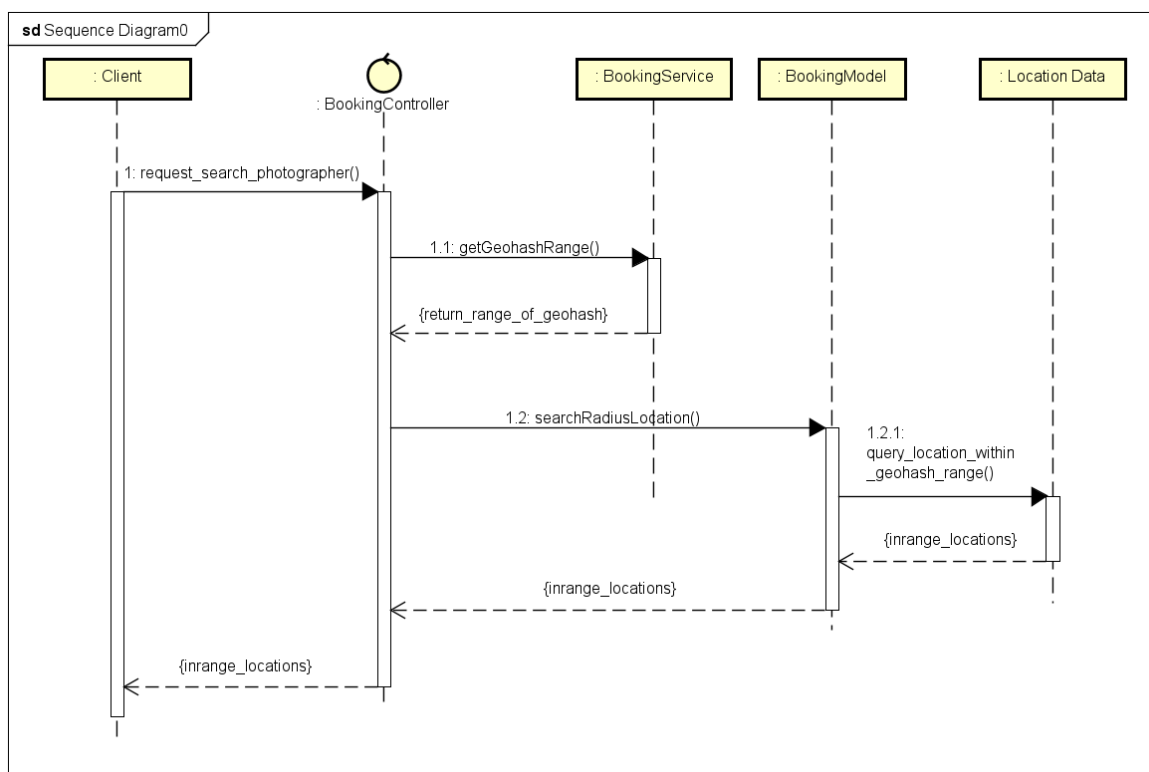
**Table 8** describes in detail function of each method in the class BookingController.

**Table 8** Specification of class BookingController

Method	Action
getLocation()	Receive request from clien, retrieve location of photographer and send it back to client
searchRadiusLocation()	Receive request from client, retrieve all photographers that are in certain radius from user’s location

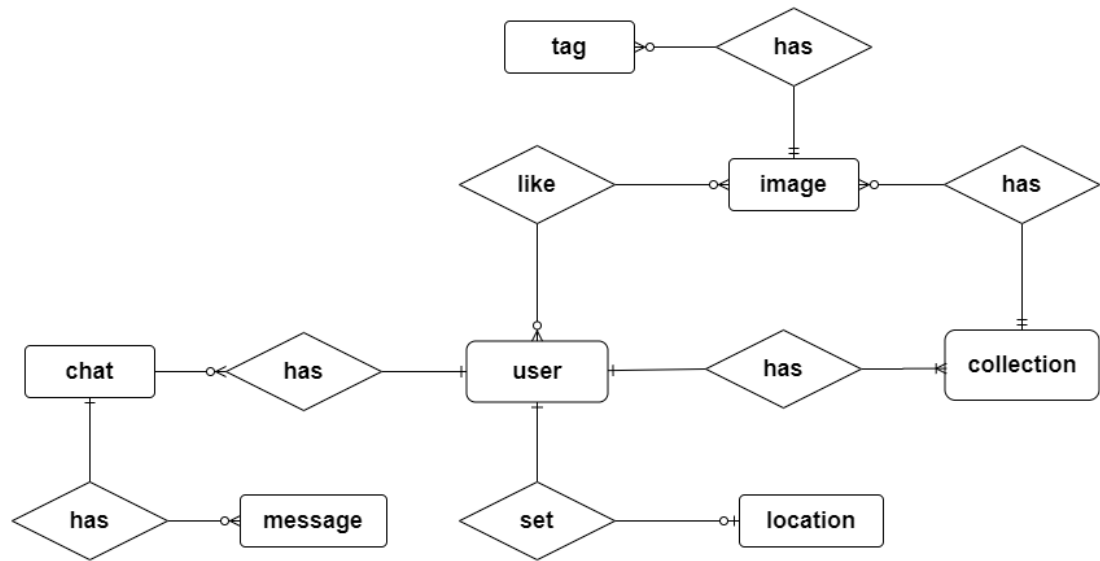
setLocation()	Receive request from photographer, that get location of photographer and create or edit on database
---------------	---

Sequence diagram of use case “Search nearby photographer by suggested location” is described as **Figure 36**. In the beginning of flows, BookingController will receive request from client to search nearby photographers. Next, from coordinate of user’s location that receive from client, BookingCotroller will get it and calculate the boundary for certain radius from user’s location, then represent that boundary in Geohash by using getGaohashRange() method from BookingService. After that, BookingService will call searchRadiusLocation() method which imported from BookingModel to retrieve all photographers that are in Geohash boundary. Finally, BookingController sends all available photographers to client.



**Figure 36** Sequence diagram of use case “Search nearby photographer by suggested location”

### 4.2.3 Conceptual data model



**Figure 37** Entity Relationship Diagram

**Figure 37** describes entity relationship. Database is structured base on Cloud Firestore Database structure which is NoSQL database. Each entity will be represented by JSON objects, which have relationship that one object belongs to one or more other objects. From **Figure 37** Entity Relationship Diagram, A user has at least one collection, and can like other images or not. User also has list of chatrooms that each chatroom contains list of messages. Each collection contains list of images that each image has different tags. User play a role as photographer can set location.

### 4.2.4 Logic data model

**Figure 38** to **Figure 44** specify the detailed document in Firestore database design.

```
age: 25
email: "dungkaka@gmail.com"
gender: "male"
name: "Dung4"
password: "$2b$10$K0qZnG83KWLwMaBFKGqdf.as2sV3tu2bD/sf4B1Q5rJy"
role: "photographer"
username: "haquydung4"
```

**Figure 38** user entity

```

description: ""
▶ info: {hUnits: "px", height: 200...}
▶ like_by: ["user_1", "bfUcDCZZrxYuJ5..."]
likes: 2
name: "Anh Fake"
▶ tags: {1: true, sport: true, wat...}
thumbnail_url: "https://firebasestorage.googleapis.com/v0/b/photohub-
e7e04.appspot.com/o/thumb_Fotolia_115575760_Subscrip
alt=media&token=10312b6c-627c-481d-aa00-289e57f25d6
url: "https://firebasestorage.googleapis.com/v0/b/photohub-
e7e04.appspot.com/o/Fotolia_115575760_Subscription_Monthly_L.jpg?
alt=media&token=40d6ea2c-4b6b-43ea-98db-37da104d85d9"

```

**Figure 39** image entity

```

category: "costume"
▼ categoryName
  en: "Costume"
  vi: "Trang phục"
▼ tags
  ▶ 0 {id: "long-dress", name: {...}}
  ▶ 1 {id: "uniform", name: {en:...}}
  ▶ 2 {id: "sport", name: {en: "...}}
  ▶ 3 {id: "office", name: {en: ...}}
  ▶ 4 {id: "skirt", name: {en: "...}}

```

**Figure 40** tag entity

```
date_create: "2018-1-1"
▼ images_snippet
  ▶ 0 {image_id: "65rk6wzQWObltD...}
  ▶ 1 {image_id: "0RZvWUZGCmmEj5...}
  ▶ 2 {image_id: "0CEuNYrntvclva...}
name: "Hello X"
```

**Figure 41** collection entity

```
active: true
createdAt: May 21, 2020 at 4:40:35 PM UTC+7
▼ photographer
  id: "bfUcDCZZrxYuJ5MLwgWY"
  name: "haquydung123456"
▼ user
  id: "bfUcDCZZrxYuJ5MLwgWY"
  name: "haquydung123456"
```

**Figure 42** chatroom entity

```
geoHash: "s7kd4n0mp"
latitude: 18.66666
longitude: 17.66666
photographerId: "CSdYAxv3aOtkqgLDIqdn"
▼ photographerInfor
  age: 25
  email: "dungkaka@gmail.com"
  gender: "male"
  name: "Ha Quy Dung 1"
```

**Figure 43** location entity

```
_id: "627a0253-8740-42c7-90b3-9a0536b3267b"
createdAt: May 21, 2020 at 4:41:21 PM UTC+7
text: "Hehe"
▼ user
  _id: "bfUcDCZZrxYuJ5MLwgWY"
  avatar: "https://firebasestorage.googleapis.com/v0/b/photohub-
    e7e04.appspot.com/o/avatar%2Favatar_2.jpg?
    alt=media&token=f35731bb-90d0-4a6c-83e6-2192a2742f43"
  name: "haquydung123456"
```

**Figure 44** message entity



## 4.3 Application Construction

### 4.3.1 Libraries and tools

Table 9 presents tools and libraries used for development.

**Table 9** Libraries and tools

Objective	Tool/Lib	URL
Code editor	VSCode	<a href="http://www.eclipse.org/">http://www.eclipse.org/</a>
Language	Javascript	<a href="https://www.javascript.com/">https://www.javascript.com/</a>
Environment development	NodeJs	<a href="https://nodejs.org/en/">https://nodejs.org/en/</a>
Serverless framework	Cloud Functions	<a href="https://firebase.google.com/docs/functions">https://firebase.google.com/docs/functions</a>
Create RESTFul API	Express	<a href="https://expressjs.com/">https://expressjs.com/</a>
Storage	Firebase Storage	<a href="https://firebase.google.com/docs/storage">https://firebase.google.com/docs/storage</a>
UI	ReactJS	<a href="https://reactjs.org/">https://reactjs.org/</a>
UI	React Native	<a href="https://reactnative.dev/">https://reactnative.dev/</a>
Tools and services built around React Native	Expo	<a href="https://docs.expo.io/">https://docs.expo.io/</a>
Animation UI	Reanimation	<a href="https://docs.expo.io/versions/latest/sdk/reanimated/">https://docs.expo.io/versions/latest/sdk/reanimated/</a>
Geohash encryption and decryption	Geohash	<a href="https://github.com/sunng87/node-geohash">https://github.com/sunng87/node-geohash</a>
Database	Firestore	<a href="https://firebase.google.com/docs/firestore">https://firebase.google.com/docs/firestore</a>
Test web application on browser while developing	Chrome	<a href="https://www.google.com/intl/vi_vn/chrome/">https://www.google.com/intl/vi_vn/chrome/</a>
Test mobile application on device while developing	Samsung galaxy S7	

### 4.3.2 Achievements

Project developed successfully with three applications: PhotoHub, PhotoHub PG and PhotoHub Admin. While PhotoHub and PhotoHub PG is built as mobile applications for user purposes such as photograph filtering, collection management, nearby photographers search, user and photographer connection, PhotoHub Admin is built as web application for administration purposes such as database management.

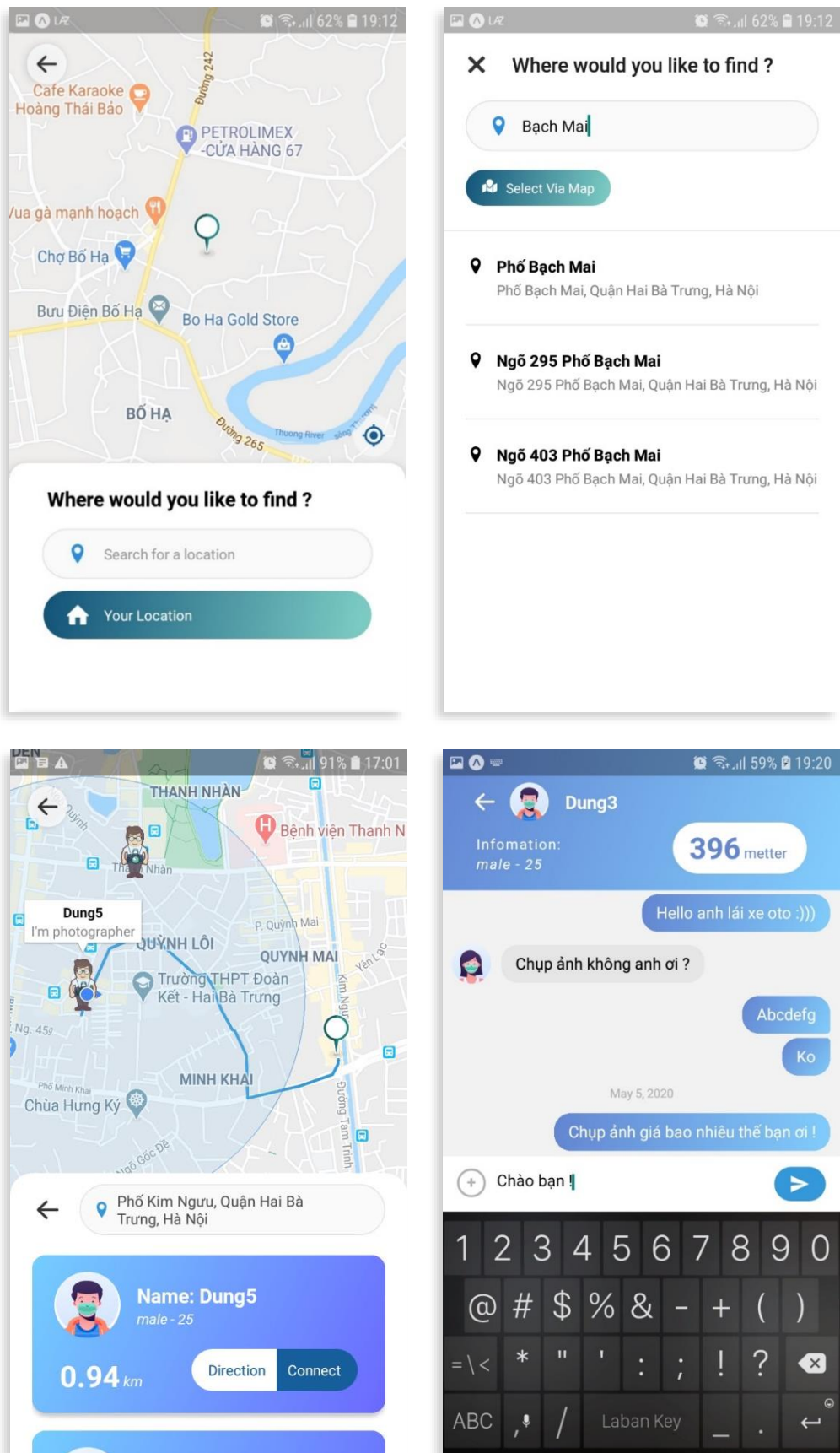
Statistics of system information are shown below.

- i) Number of code line: 29000
- ii) Capacity of the entire source code: 11.11 MB
- iii) Programming environment: window
- iv) Debug tool: Samsung galaxy S7 and Chrome browser

### 4.3.3 Illustrations of main features

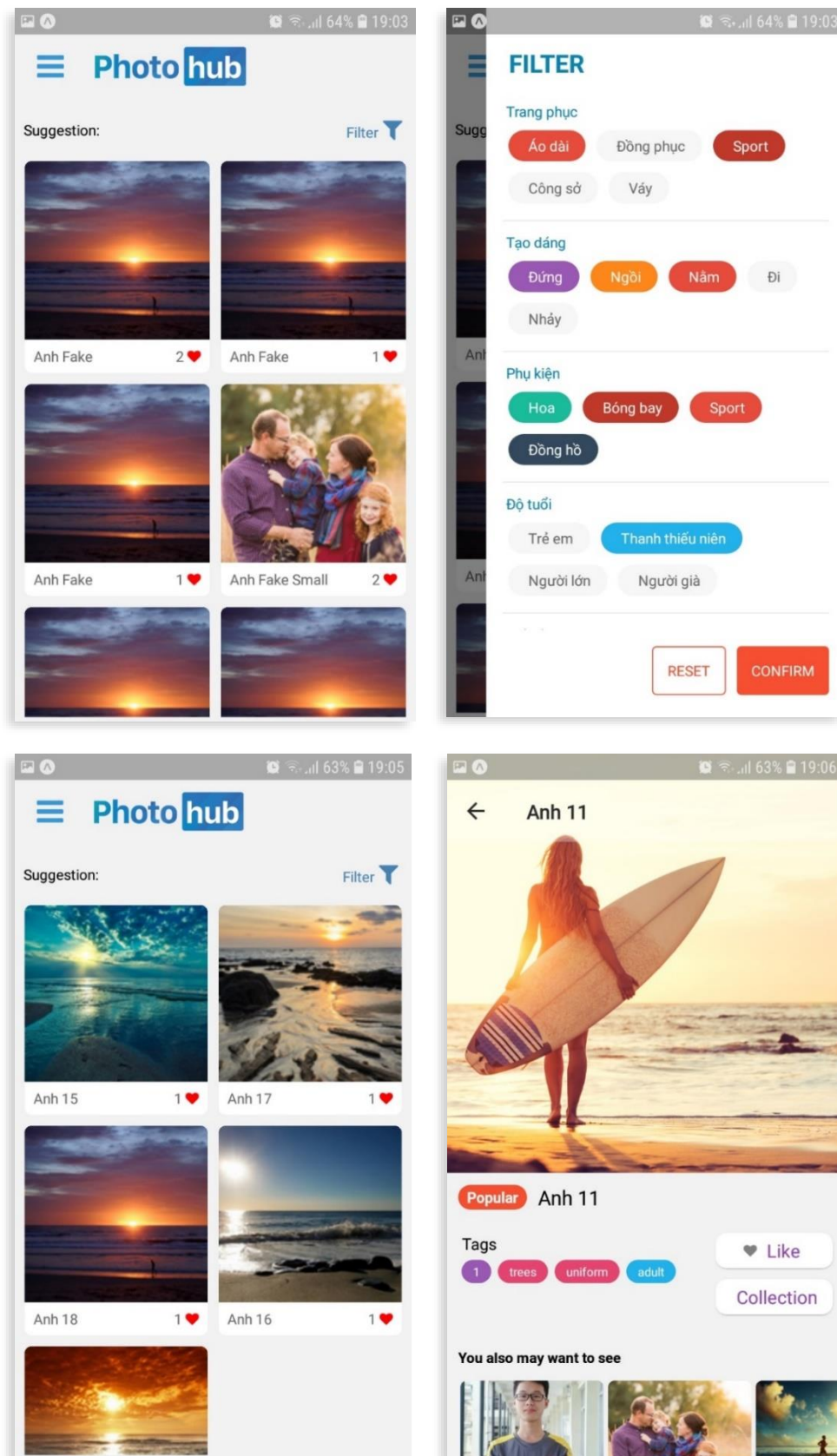
This section will focus on the main features of PhotoHub mobile application.

The most important features which PhotoHub application provided is searching nearby photographers which is described as **Figure 45**. On map screen, user can click on “Search for a location” to move suggested location screen, click on “Your location” for searching nearby photographer by device’s location. List of available will be displayed on screen, user may click on “Direction” button to find the way reach to photographer or click on “Connect” button to chat with photographer.



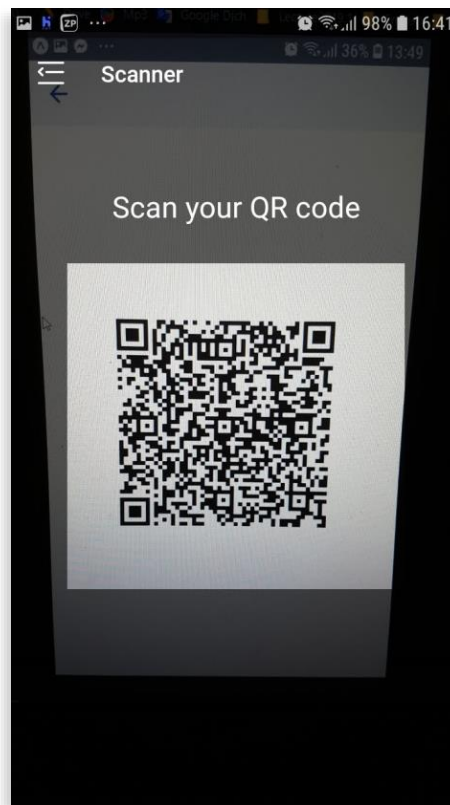
**Figure 45** Searching and connecting nearby photographer screens

Seconds, filter photographs by tags is described as **Figure 46**.



**Figure 46** Photograph filter screens

Third, QR Code scan interface is described as **Figure 47**.



**Figure 47** QR code scan screen

## 4.4 Test

STT	Description	Step by step		Output	Conclusion
		Step by step	Expectation		
Register new account					
1	Skip one of the fields: username, password, confirm password, email	Step 1: Go into register screen.  Step 2: Skip password field.	Invalid field message	Password is required field	PASS

2	Length of password is not in range of 6 to 10 characters	<p>Step 1: Go into register screen</p> <p>Step 2: Enter password whose length is shorter than 6 or longer than 10.</p>	Display message that password must be invalid	<p>Password must be at least 6 characters.</p> <p>Password must be at most 10 characters</p>	PASS
3	Make password and confirm password is different	<p>Step 1: Go into register screen</p> <p>Step 2: Enter password</p> <p>Step 3: Enter confirm password that different to password</p>	Display message that confirm password and password must be the same	Confirm password is not match to password	PASS
4	Enter wrong type of email	<p>Step 1: Go into register screen</p> <p>Step 2: Enter email without notation @</p>	Display message that email is invalid	Email must be a valid email	PASS
Search nearby photographer					
1	Search photographer by your location without enable LOCATION permission	<p>Step 1: Go into search photographer screen</p> <p>Step 2: Turn on location on mobile phone</p>	Display alert that inform location not found	<p>Location provider is unavailable.</p> <p>Make sure that location services are enabled</p>	PASS

		Step 3: Click on button “Your location”			
2	Search photographer	Step 1: Go into search photographer screen  Step 2: Choose type of search and search.	Display list of available photographers working in area	Display list of available photographers working in area	PASS
Scan QR Code for collection					
1	Scan invalid QR code	Step 1: Go into Scan QR Code screen  Step 2: Scan invalid QR Code	Display alert that inform QR Code is invalid	Display alert that inform QR Code is invalid	PASS
2	Scan valid QR code	Step 1: Go into Scan QR Code screen  Step 2: Scan valid QR Code	Display collection responding to QR code	Display collection responding to QR Code	PASS
3	Add collection	Step 1: Scan valid QR code  Step 2: Click on button “Add collection”	Display message that collection is added successfully	Display message that collection is added successfully	PASS
Create new collection					
1	Create new collection with	Step 1: Go into collection screen	Display alert that inform	Display alert that inform	PASS

	same name to exist collection	<p>Step 2: Click on button “Create New Collection”</p> <p>Step 3: Enter collection name with same name to exist collection</p> <p>Step 4: Click button “OK”</p>	collection has already been exist	collection has already been exist	
--	-------------------------------	---	-----------------------------------	-----------------------------------	--

## 4.5 Implementation

PhotoHub system contains three applications:

- i) PhotoHub application is developed on both iOS and Android platform by using React Native.
- ii) PhotoHub PG application is developed on both iOS and Android platform by using React Native. Application is built for photographer.
- iii) PhotoHub Admin web application is developed on web by using ReactJS. Application is built for administrator.



# Chapter 5 Solutions and main contributions

## 5.1 Search nearby photographer by Geohash

### 5.1.1 Problem Description

Based on the simplest logic, finding all photographers within a 5km radius is done through the following steps:

- i) Locate user location in longitude, latitude.
- ii) Query the entire photographer's location database, then calculate the distance from the provided location with the location of each photographer. If this distance is within a radius of 5km, photographer is available to user.

The calculation of the distance from the position of user to the position of each photographer is done easily through the Haversine<sup>17</sup> formula. However, with a system containing the location of millions of photographers, we need to retrieve all the photographer's locations from the database then calculates millions of calculations to identify all photographers within a radius of 5km from a specified location. This causes a huge search cost, which affects the performance of the system. Moreover, follow Cloud Firestore Billing, if i get all location of millions of photographers, i need to pay so much for this operation. The actual cost is huge. Therefore, I thought of finding an alternative which is to use Geohash, that makes searching easy with the least cost, both in performance and price

### 5.1.2 Solution

Geohash definition is described in Chapter 3 – 3.7.

The use of geohash makes searching limited to a certain range. We don't need to calculate the distance for all records in the database. Instead, we save location of each photographer in a boundary area as a string called Geohash. Therefore, it is possible to index this field and the query becomes as optimal as possible.

---

<sup>17</sup> [https://en.wikipedia.org/wiki/Haversine\\_formula](https://en.wikipedia.org/wiki/Haversine_formula)

For example, geohash which corresponds to the position of Hanoi University of Science and Technology is xxxxx, we call this point as O. The radius of 5km from O will correspond to geohash from xxxxx (X) to xxxxxx (Y).

We will calculate upper and lower boundary to find out X and Y

```
const geohash = require('ngeohash');

class BookingService {

  static getGeohashRange = (
    latitude: number,
    longitude: number,
    distance: number, // miles
  ) => {

    // degrees latitude per mile
    const lat = 0.0144927536231884;
    // degrees longitude per mile
    const lon = 0.0181818181818182;

    const lowerLat = latitude - lat * distance;
    const lowerLon = longitude - lon * distance;

    const upperLat = latitude + lat * distance;
    const upperLon = longitude + lon * distance;

    const lower = geohash.encode(lowerLat, lowerLon);
    const upper = geohash.encode(upperLat, upperLon);

    return {
      lower,
      upper
    };
  };
}

export default BookingService;
```

However, geohash can still cause errors, which is the case where the O point is in the boundary line between its two parent regions. Therefore, after retrieving all locations in range from X to Y in database, we need to recalculate the distance to make sure the results are correct.

geohash length	geohash length	lng bits	lat error	lng error	km error
1	2	3	± 23	±23	±2500
2	5	5	±2.8	±5.6	±630

3	7	8	±0.70	±0.70	±78
4	10	10	±0.087	±0.18	±20
5	12	13	±0.022	±0.022	±2.4
6	15	15	±0.0027	±0.0055	±0.61
7	17	18	±0.00068	±0.00068	±0.076
8	20	20	±0.000085	±0.00017	±0.019

Use the haversine distance formula to calculate the distance between two points on the map.

```
export function haversine_distance(mk1, mk2) {
  //Radius in km
  var R = 6371.071;
  // Convert degrees to radians
  var rlat1 = mk1.latitude * (Math.PI / 180);
  // Convert degrees to radians
  var rlat2 = mk2.latitude * (Math.PI / 180);

  // Radian difference (latitudes)
  var difflat = rlat2 - rlat1;
  // Radian difference (longitudes)
  var diff lon = (mk2.longitude - mk1.longitude) * (Math.PI / 180);
  var d =
    2 *
    R *
    Math.asin(
      Math.sqrt(
        Math.sin(difflat / 2) * Math.sin(difflat / 2) +
        Math.cos(rlat1) *
        Math.cos(rlat2) *
        Math.sin(diff lon / 2) *
        Math.sin(diff lon / 2)
      )
    );
  return d;
}
```

## **5.2 Create friendly UI/UX – Animation on Map Screen for searching nearby photographer.**

### **5.2.1 Problem description**

When developing the PhotoHub application, creating maps and manipulating them in the most friendly user interface is not an easy problem. The application includes 3 different ways to search:

- i) Search by your location.
- ii) Search by suggested location.
- iii) Search via map.

The search process must not cause users discomfort. Response time must be not too long, map size must fit screen with the available results of photographers, the color of the direction from user to photographer must be clear, suggested location list must be displayed properly, making it possible for any user to use the application.

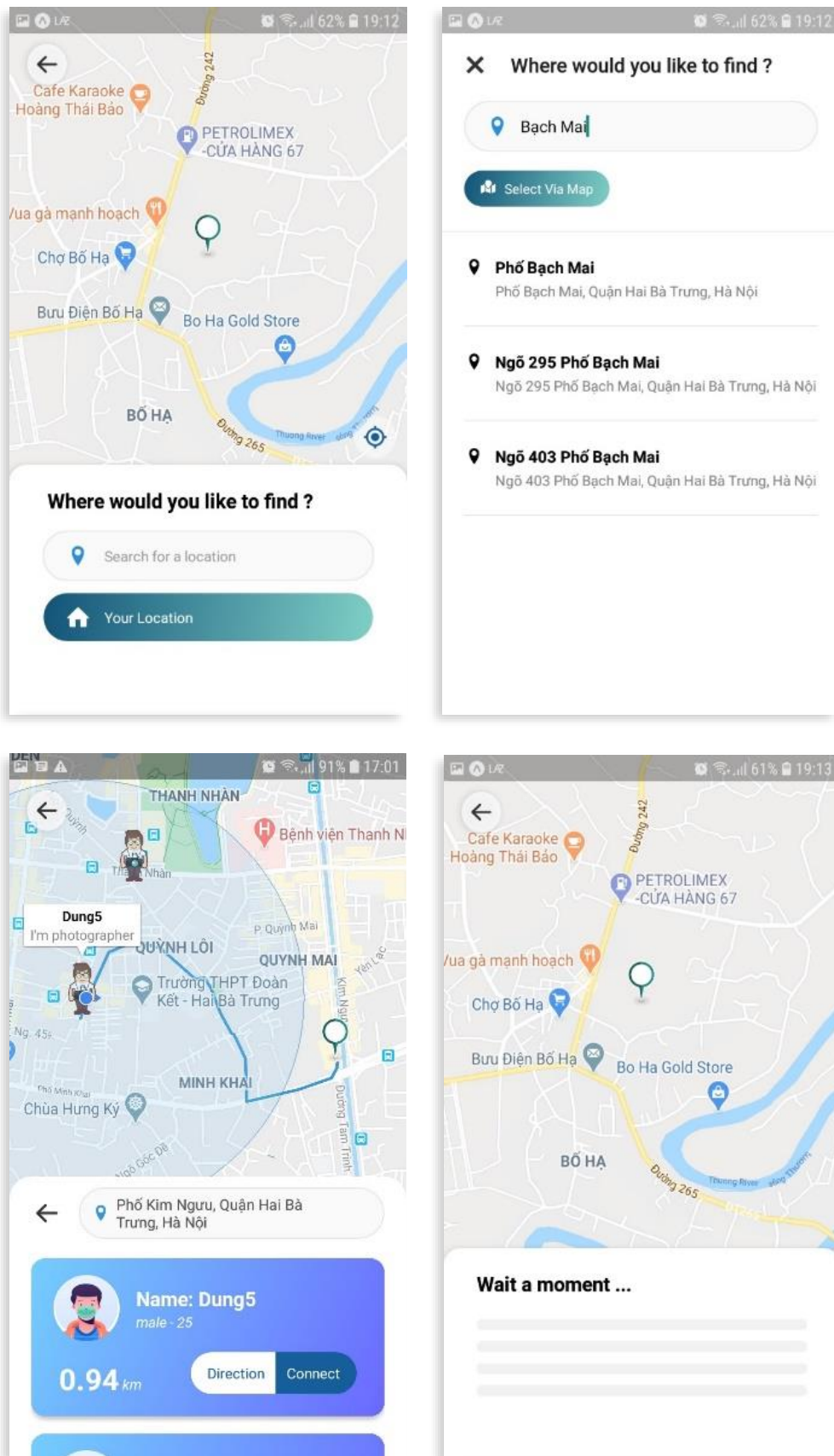
### **5.2.2 Solution**

PhotoHub application has overcome some of the following problems:

- i) Create a maker to determine which is the middle point on the map, which is also the current location that user is using
- ii) Use the bottom sheet where provide buttons and inputs for users to manipulate while the map is still available on the screen
- iii) Suggested location using placeAPI provided from HereMap (during using, because of some billing issues with Google placeAPI, I switched to using the HereMap placeAPI to get suggested location). The suggested location is listed right after 500 milliseconds when you stop typing.
- iv) The waiting process is presented via a bottom sheet helps the user know the system is running correctly.

Animation is used for all the above to enhance the user experience.

**Figure 48** illustrates some interfaces of Map Screen on PhotoHub application.



**Figure 48** UI/UX – Animation on Map Screen

## 5.3 Share collection by QR Code

### 5.3.1 Problem description

Share collection is a fairly necessary function in the system. It is helpful for users to collect poses before taking a photograph and share it with friends or photographers to make the photography process easier.

There are many ways to share this such as: sharing via social media network (Facebook, Google), share by link, share by QR Code, etc. This share must ensure convenience for users: perform less manipulate as much as possible and can perform based entirely on the application without going through a third party.

QR Code is the most reasonable solution.

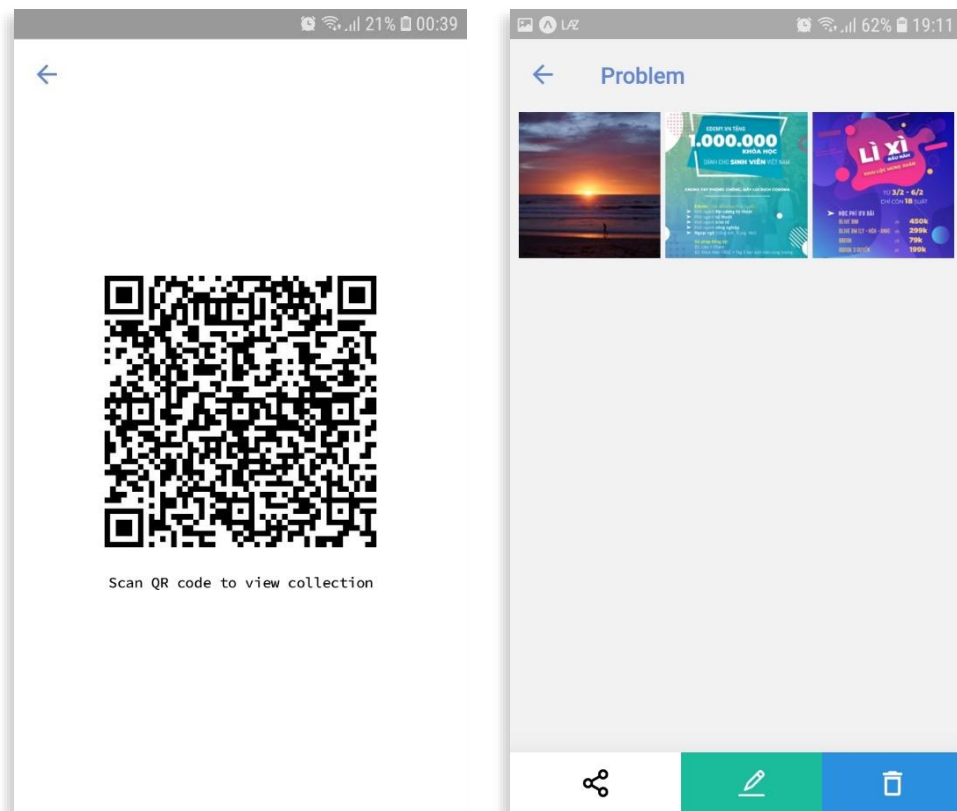
### 5.3.2 Solution

QR Code definition is described in Chapter 3 - 3.8.

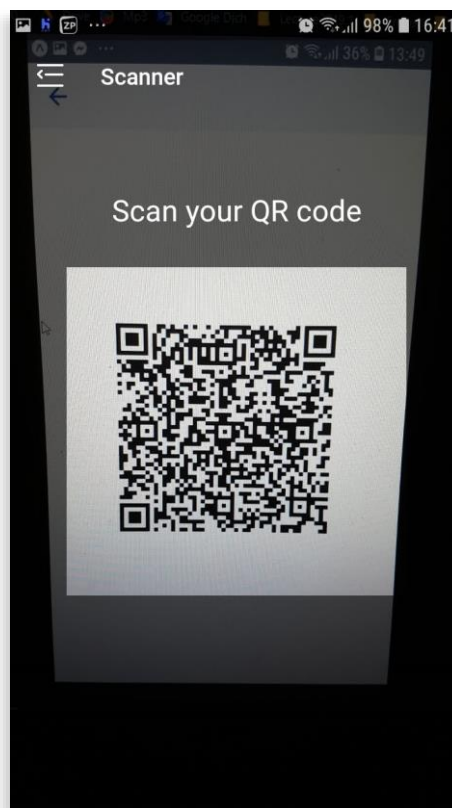
When making a share each collection, application will automatically generate a QR code corresponding to the link to get the collection. Users on the other side just need to select the Scan QR Code function and scan, each valid QR Code will automatically go to that show collection screen.

Sharers: The number of operations performed is 1: (1) Click on the Share button, QR Code automatically show on screen. **Figure 49** describes how simple UX is.

Recipients: The number of operations performed is 2: (1) Click on scan QR Code, (2) capture QR Code from sharer. Collection will automatically show on screen. **Figure 50** describes how to scan QR code



**Figure 49** Collection sharing screen



**Figure 50** Scan QR code screen

## 5.4 Filter photos by tags.

### 5.4.1 Problem description

An important function of PhotoHub application is looking for poses.

A normal user often doesn't know how to pose professionally in accordance with different circumstances. Some applications have overcome this problem. They provide users with a large number of poses organized in categories. However, in case that users want to search for a specific circumstance, they need to scan all photographs in each category. It takes them a long time for search operation. Therefore, it is necessary to find a way of searching that are friendly and easy to use in every circumstance.

After studying, I offer four solutions for filtering poses. This is the goal of user-behavior: When the user wants to search all the yearbook photographs for a girl, the system will find all the photographs with this topic, then display on the screen for the user. I will explain four solutions to solve user-behavior:

- i) Search by entering an arbitrary string. For this search method, the user only needs to enter an unique arbitrary string, then press the search button to search for all images matching or related to that string. Users cannot search multiple selections at the same time. For example, to search the photograph of yearbook with long dress for women, users can only search 1 of 3 options: Proceedings, long dress or female.
- ii) Search by entering a string. The system will give suggested options. Users can rely on it to select the options. The system may limit the selection of one or more options. For example, to search for photographs taken with a long dress for women, users will turn to search for three keywords: Proceedings, dresses, more, and add them to the list of tags, then press the search button to retrieve the link.  
fruit
- iii) Search by category. The system will group images by category: Costumes, etc. Users will open each category to view each image item. However, if searching by category, users cannot search details in each specific case. For example, there's no way to search for yearbook photographs for girls, because these three tags belong to three different categories.
- iv) Search by tags. Each photograph will contain a number of tags that describe the photograph. For example, a photograph of a girl wearing a long dress taking pictures of a yearbook may contain three tags: yearbook, long dress, and schoolgirls. Thus, the system will provide a list of specific tags, users can select



one or more tags at the same time, then the system will search all images containing at least 1 tag in the list of tags.

Some of the outstanding ways to search are outlined above.

PhotoHub application goal is provide users various poses which cover many circumstances: gender, background, age, accessory, etc. Therefore, the most appropriate solution is Search by tags.

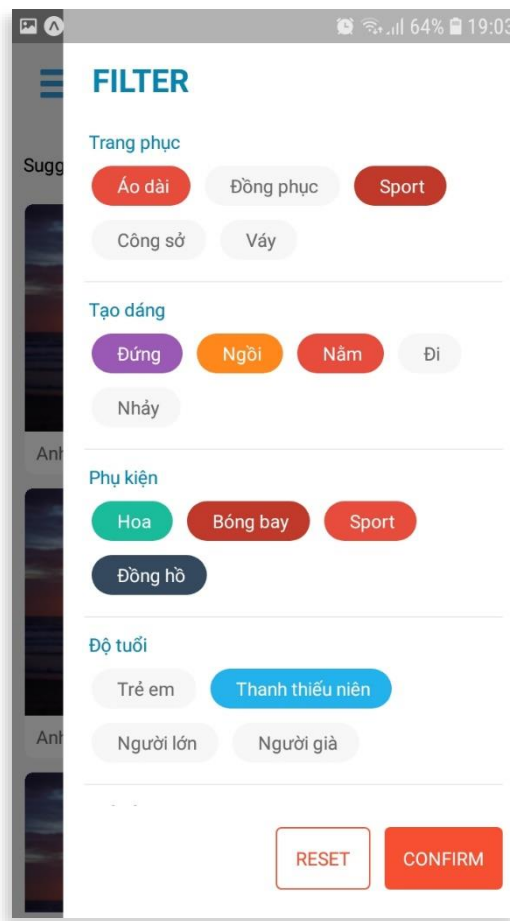
### 5.4.2 Solution

Firstly, we make an example about searching by tags. Picture bellow describe a girl smiling in graduation event. I will descibre that picture by some tags: girl, smile, graduation, one person as **Figure 51**.



**Figure 51** Picture description by tags

The search operation must be easy, user-friendly, and the least manipulated so that new users can easily get the best results. The application provides a built-in interface that lists all the most specific cases that can occur when taking a photograph, a case described as a tag as **Figure 52**. Users can select multiple tags at once, the application will search all images containing at least one tag in the list of tags. As such, users will not need to think about search keywords, as well as the search for images with complex descriptions. All details are displayed on the screen and users just click to select and confirm.



**Figure 52** Photographs filter screen

## 5.5 Firestore and surrounding issues

### 5.5.1 Problem description

Definition and benefits of Firestore is already described on Chapter 3 – 3.4.

Firestore is NoSQL databases and have its own structure. In PhotoHub system, we need an appropriate structure for developing and expanding purpose.

Firestore store data as collections of documents. It is quite difficult to choose an appropriate structure for powerful query, and scale problem. I have faced with some problems when working with Firestore:

- i) Relationship problem: Represent relationship between documents that is quite similar with relationship using foreign\_key in SQL database. Nevertheless, in Firestore, there is no option to handle this. In my case, using top-level collection and sub-collection which are two ways to handle relationship in database.

- ii) List problem: List is an organization of records such as users, images. Firestore provide 3 ways to store a list: array, map, sub-collection. In some case, choosing an appropriate practice is difficult for optimize performance, scale and price.
- iii) Index problem: Optimize query by using index. In some case which you do not index on some field, a query may not run. We need to figure out why need to index and how to index for optimizing single query and compound query in Firestore.

## 5.5.2 Solution

### 5.5.2.1 Relationship problem

First of all, we discuss two options for solving relationship problems:

- i) Use top-level collection
- ii) Use sub-collection

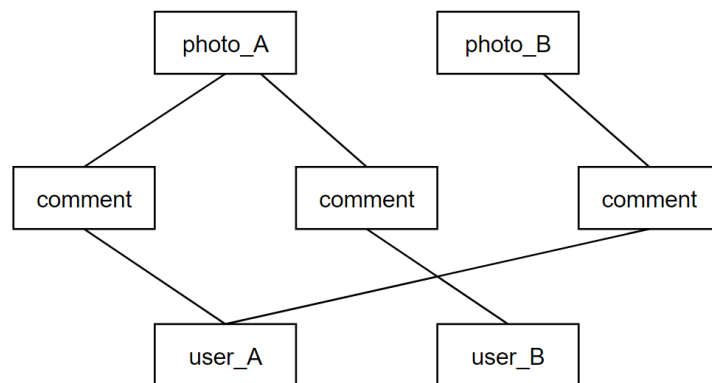
Both of them are collection. While top-level collection is collection that does not belong to any document, sub-collection is collection that is inside other document, when document is removed, sub-collection will also be removed.

**Table 10** shows the different use between top-level collection and sub-collection.

**Table 10** The difference between Top-level collection and Sub-collection

Top-level collection	Sub-collection
Using to solve many-to-many relationship problems. In other words, when a document has two primary keys, we use top-level collection.	Using to solve one-to-many relationship problems.

**Example 1:** We have list of photographs, the user can comment on each photograph.



**Figure 53** Illustration of relationship between photograph, user, and comment

In order to store list of comments, we have two options:

- i) Store list of comments in a collection which is outside photograph document. That call top-level collection. In comment collection, each document contains some needed informations of photograph and user.
- ii) Store list of comments in a collection which is inside photograph document. That call sub-collection. In comment collection, each document contains some needed information of user.

**Question:** Which one is better for example 1?

**Table 10** show us that, top-level collection is the best choice. One of the most important reason related to query with Firestore.

**Explain:**

First, we want to retrieve all comment of user\_A on photo\_A. If we use top-level collection for comment list, we need to run query:

```
Firestoredb.collection(comments)
.where(user, "==", user_A)
.where(photo, "==", photo_A)
```

If we use sub-collection, we need to run query

```
Firestoredb.collection(photos).doc(photo_A)
.where(user, "==", user_A)
```

After that, we want to retrieve all comment of user\_A on all photos of system. If we use top-level collection for comment list, we need to run query.

```
Firestoredb.collection(comments)
.where(user, "==", user_A)
```

However, if we use sub-collection, there are no way to solve that. We realize that, each comment contains many-to-many relationship. Each photograph can have many comments and each user can comment many photographs. However, if we use sub-collection for comment list, now list of comment belong to each photograph. When we want to retrieve all comment of user\_A on all one million photographs of system, we need to run one million queries. That is impossible.

```
Firestoredb.collection(photos).doc(photo_A)
.where(user, "==", user_A)

Firestoredb.collection(photos).doc(photo_A)
.where(user, "==", user_B)

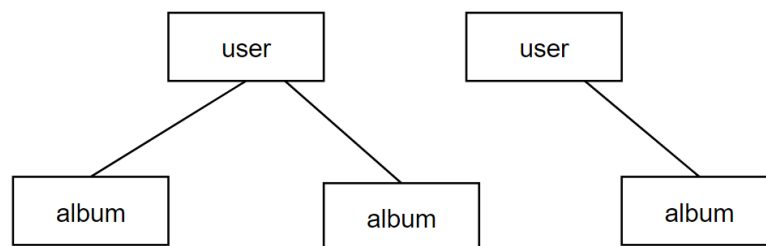
Firestoredb.collection(photos).doc(photo_A)
.where(user, "==", user_C)

...
```

Thus, for this example, top-level collection is the best solution.

**Example 2:** We have list of users, each user owns their albums.

In this example, sub-collection is better to store list of albums because each album just belongs to only one user. The relationship is one-to-many. A user can have many collections.



**Figure 54** Illustration of relationship between user and album

### 5.5.2.2 List problem

**Table 11** distinguish the different between of some concepts.

**Table 11** The differences between array, map and collection

	Array	Map	Collection
Example	[1,2,3,4]	{vi: “nhay”, en: “jump”}	List of documents that represents for an entity
Usage	Put data in the same document if you’re always going to display it together. Don’t make it to big or too short		Put data into collection if you want to search indivisual pieces of data or if you want your data to have room to grow
	Leave data in array if you don’t care about exact position of them	Leave data as a map field if you’re going to want to search your “parent” object based on that data	

### 5.5.2.3 Index problem

Indexes are an important factor in the performance of a database. Much like the index of a book which maps topics in a book to page numbers, a database index maps the items in a database to their locations in the database. When you send a database a query, the database can use an index to quickly look up the locations of the items you requested.

There are two types of indexes that Cloud Firestore uses, single-field indexes and composite indexes. Single-field indexes are used when you run a query with single field condition. Composite indexes are used when you run a query with more than one field condition, on other words, it is compound query.

There are some rules for index in Firestore:

- i) Besides ascending and descending mode, Firestore have array-contains index mode that supports array-contains query clauses on the field.
- ii) By default, Cloud Firestore automatically maintains single-field indexes for each field in a document and each subfield in a map

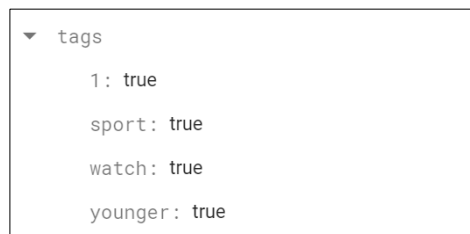
- iii) Cloud Firestore does not automatically create composite indexes like it does for single-field indexes because of the large number of possible field combinations. Instead, Cloud Firestore helps you identify and create required composite indexes as you build your app.

#### 5.5.2.4 Particular problems in Photohub system database

**Situation 1:** Store tags in image collection and query to retrieve images by tags.

A photograph is described by a list of tags. Therefore, tags must be stored as a list. There are some options for choosing type of tags: array, map, sub-collection.

Unfortunately, Firestore does not support multiple array-contain condition in single query. Therefore, we can not store tags in an array to retrieve an image that have some tag conditions. We also don't need to search for more detail information of tags. Thus, map is the best solution.



**Figure 55** Illustration of storing tags in a map

For retrieving all images that contain the tags: “sport”, “watch”, “younger”, we run following query:

```
Firestoredb.collection(images)
.where(tags.sport, "==", true)
.where(tags.watch, "==", true)
.where(tags.younger, "==", true)
```

**Situation 2:** Paginate images.

Firestore does not provide offset() method to retrieve list of documents from certain offset. However, there are alternative ways to solve pagination problems.

Use the startAt() or startAfter() methods to define the start point for a query. The startAt() method includes the start point, while the startAfter() method excludes it. The startAt(image\_A) means that query will retrieve all images after image\_A in image collection as following query

```
Firestoredb.collection(images).startAt(image_A).limit(40)
```

**Situation 3:** Store album as user's sub-collection.

User have their own album. This is one-to-many relationship. Therefore, storing album as user's sub-collection is better solution.

**Situation 4:** Store list of images in each album.

Each album has list of images. We always going to display it together and don't want to search search for any indivisual pieces of list of images. Therefore, we will store list of images in each album as an array instead of sub-collection.

**Situation 5:** Store chat as top-level collection.

Each chat room has list of chat conversation. Each chat room has information of user and photographer information as two primary keys. This is many-to-many relationship. Therefore, store chat collection as top-level collection is the best solution.



# Chapter 6 Conclusions and Future Work

## 6.1 Conclusions

The system is built to help normal users can connect with the photographer. The process of searching for a photographer is easy and convenient. It is simple for photographer in reaching out to users. The system also provides a filter of poses for photographs according to different circumstances to better serve the demands of the user. Moreover, the system also helps users to store photographs, share collections of poses to help the photography process more convenient.

The application has the drawback that doesn't solve the commercial problem in connecting users with photographers.

## 6.2 Future Work

Within the scope of the graduation project, the system has initially developed the necessary functions to serve the needs of looking for photographers as well as poses. However, in order to make the application even more perfect, I need to solve some more problems in the future:

- i) Currently, PhotoHub application has built a function to find the most optimal filtering that is searching by tags. In the near future, I will research and build more approaches of searching to help users have more options
- ii) The system was able to help connect users and photographers. However, the system has not solved commercial problem yet, how to charge between connecting users and photographers. In the future, I will research and build this feature
- iii) Performance and user experience. PhotoHub and PhotoHub PG application have not had an offline photograph view yet, in the future you will develop this feature

# Reference

- [1] <https://firebase.google.com/docs>
- [2] <https://reactjs.org/docs/getting-started.html>
- [3] <https://reactnative.dev/docs/getting-started>