**Chapter 4 System Design**

In this chapter, we will first discuss our system design based on our experiment results. And then we will describe how to implement the system in detail.

1. User Case

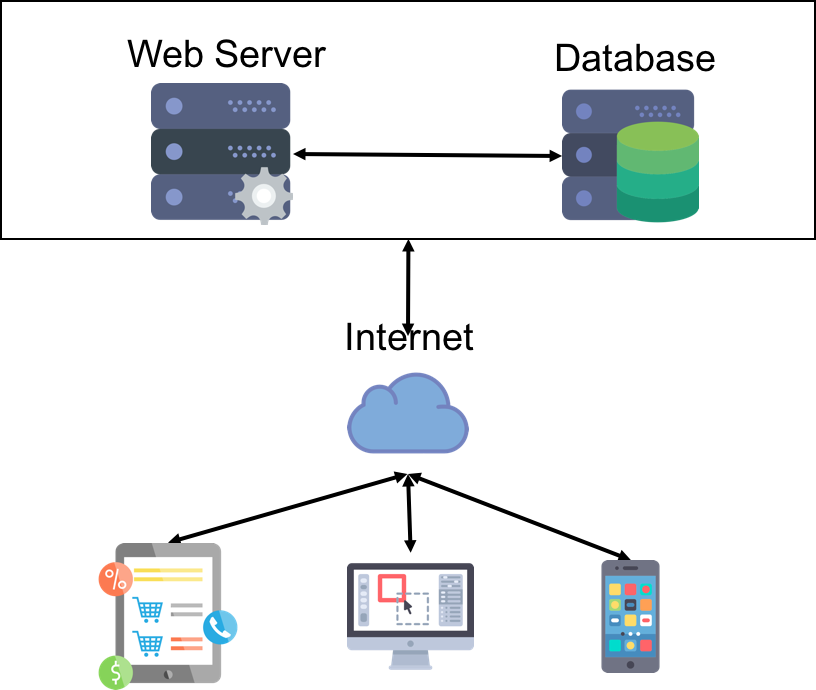
User case is the definition and description of a whole unit function of a system or subsystem without the presentation of the internal structure of a system or subsystem, so user cases can help user to understand the system better. This part will introduce the user cases of the system.

1. Import Images which recorded by user

After the day’s record was finished, user can click the choose-image button to choose the whole day’s images for importing into our system. And then click the import-button. The system will automatically process these metadata, such as faces, objects, location, creation time and so on. Upon the process of importing image is finished, the result will be show on the page. The user can view it easily.

1. View images via single cue
2. The user met somebody, and he/she cannot remember the other cues. He/She can double click the person image to view the related images.
3. The user remembers an object in his memory, and he/she cannot remember the other cues. He/She can double click the object image to view the related images.
4. The user remembers there is a thing happened somewhere, and he/she cannot remember the other cues. He/She can double click the location image to view the related images.
5. Query images via cues combination
6. The user met someone at about 2pm but he didn’t remember the specific time. He can click to choose this man as one cue and choose 14:00 as another cue, and the system will search the images between 13:30 to 14:30 which contain this person and show result.
7. The user met someone somewhere, he wants to review these pictures. He can choose this guy and this location to search images and recall what happened.
8. The user used some object or saw some object somewhere. He can choose this object and this location as cues combination and search the result.
9. The user remembered some object around 3pm, he can choose this object and 15:00 as cues. The system will find all images during 14:30 and 15:30 contain the object.
10. System Architecture

The system is designed using Browser/Server pattern. The advantage of this pattern is that we can use our system in any platforms which contains mobiles, pad and personal computer upon we deploy our system into the Internet. The system architecture can be designed as picture 4-1.



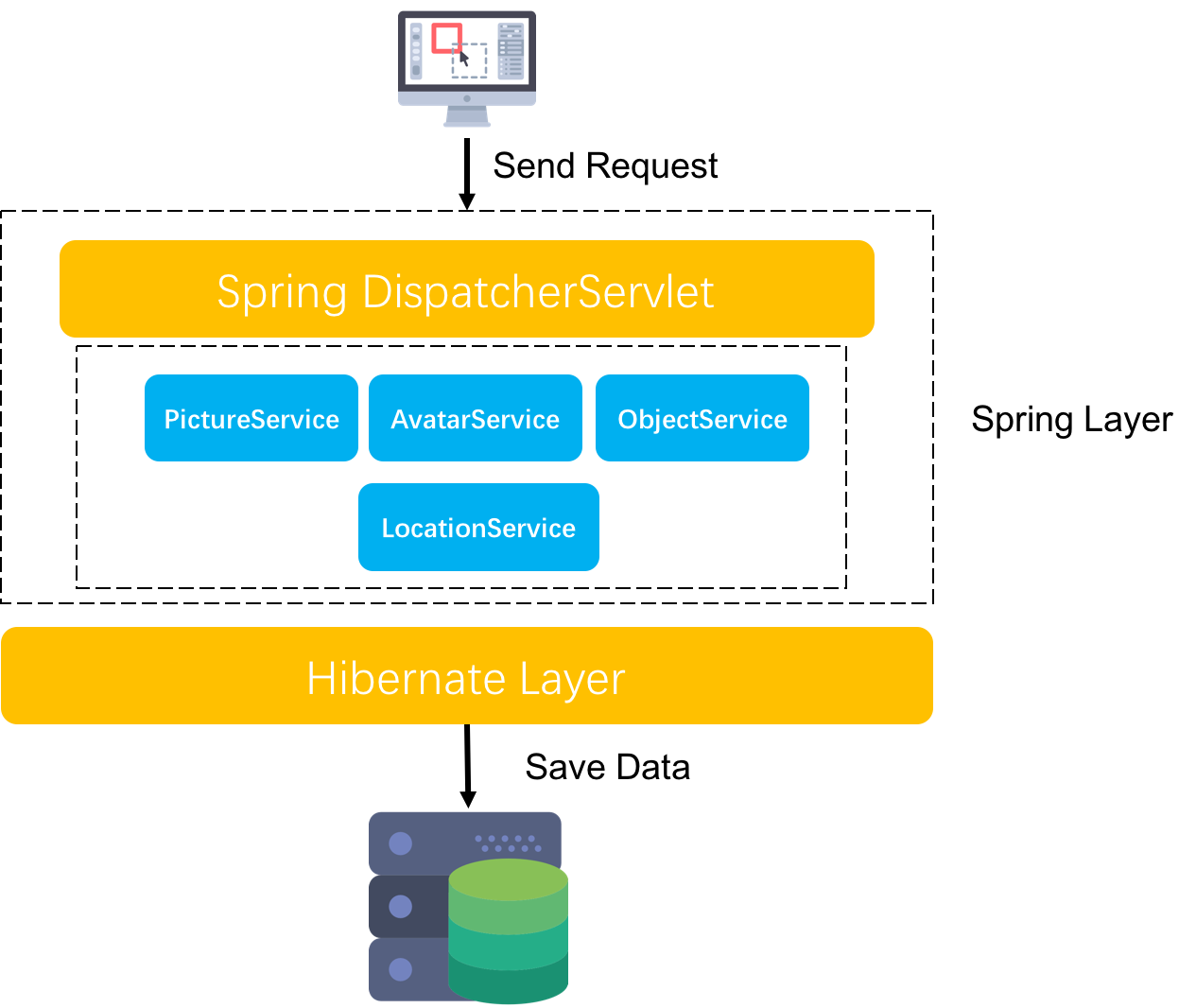
Picture 4-1

From the picture, we can see that after we finished to develop our web server and database, and the deploy it on the Internet. We can reach our system in any platforms. It’s very convenience for user to use. The Browser/Server pattern also has other advantages are listed aa below:

1. Operate anywhere without installing any specialized software
2. The expansion of the system is very strong
3. The maintenance and the logic of upgrade is very simple

For the first point, the reason is that we using our system via a browser, so if there is a browser in a device and our system is deployed on the Internet, the user can reach the website easily. As a result, the user could use our system without installing any specialized software. For the second point, we said the expansion is very strong because if we want a new feature in our system, we only need to update our web server. For the third point, the reason is similar with the second point. We have no client to maintenance. So we could spend less time on maintenance and upgrade. We just need to focus these on our web server.

We need to notice our web server in our system architecture. This part not only includes ourselves server, but also it incudes the services from Microsoft and Google. We integrate the services from Microsoft and Google into our system. And the services include Computer Vision, Face Recognize, Bing Image Search and Google Maps Server. We use these services to handle the face, object and location in user’s images.

1. System Design
2. System Framework Design
   1. The design of the framework

The implementation of the project mainly uses the combination of the Spring Boot and the Hibernate. The main purpose of using Spring Boot is that to realize the separation of the front-end and the back-end, and to reduce the coupling of the project. What’s more, Spring Boot makes it easy to create stand-alone production-grade Spring based Applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. We also use the Hibernate to realize the separation of the business code and the data. The advantages of the Hibernate includes Object/Relational mapping, JPA provider, idiomatic persistence, high performance, scalability, reliable and extensibility. Hibernate is an object relational mapping framework of open source code, it is a very lightweight object encapsulation of JDBC, it will POJO and database mapping, is a fully automatic ORM framework, hibernate can automatically generate SQL statements automatically, using object-oriented programming thinking makes Java programmers may want to manipulate the database.

From the image we can see that when user send a request to our server via browser, the request will achieve to the Spring Dispatcher Servlet and the Spring Boot will dispatch this request to ourselves service. What’s more, our service will get data and save data via Hibernate Framework. Then the server will return the data which we treated to the browser.

* 1. The design of the business code

Code-architecture

In our system, the design of the code follows the Spring MVC principles. MVC is a software design model, a business logic and data display interface, tissue isolation method code, business logic will be gathered in a component inside, and improvement in customization interface and user interaction at the same time, do not need to write business logic. MVC is uniquely developed for mapping traditional input, processing, and output functions in a logical graphical user interface structure.

From the image, we can see that our system is made of model, view, controller. The model includes all the business logic of our system, such as Data Model, Services Injected, Bing Image Search, Face Recognize, Computer Vision and so on. The function of the Controller is to handle the requests from user’s browser. Then we can treat the request using our service. The function of the View is that to show the page to the user. Using Spring MVC pattern can realize the separation of the back-end and front-end.

1. System Database Design

// 1. The design of the tables

// 2. The relationships between the tables

1. System Front-End Design
2. System Implementation
   1. Development Environment
3. The environment of hardware

The personal computer from labs: 电脑的型号

1. The environment of software
2. Platform

Windows 10 Operation System, Eclipse Neon, MySQL V5.7.18, Apache Tomcat V7.0

1. Running

Chrome Browser, Firefox Browser

* 1. API Introduction

1. Face Recognition API from Microsoft
2. Create a face list

***Request URL:***

http://westus.api.cognitive.microsoft.com/face/v1.0/facelists/{faceListId}

***Request Parameters:***

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| faceListId | String | Id of the created face list. |
| name | String | Name of the created face list. |

The function of this API is that creating a face list which stores the faces we recognized from user’s images. When we use this API, we need to provide a list id like ‘Memory\_Face\_List\_Id’and a list name like ‘Memory\_Face\_List\_Name’. And the API will return a response to show the result of creating the face list. If the response status is 200, that means we create the face list successfully.

1. Get a face list

***Request URL:***

[http://westus.api.cognitive.microsoft.com/face/v1.0/facelists/{faceListId}](http://westus.api.cognitive.microsoft.com/face/v1.0/facelists/%7BfaceListId%7D)

***Request Parameters:***

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| faceListId | String | Id of the created face list. |

***Response Body:***

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| faceListId | String | Id of the created face list. |
| name | String | Name of the created face list. |
| persistedFaces | Array | Faces in the face list. |

The function of this API is that we can get the face list we created using face list id. If we used the above API to create a face list which id is “Memory\_Face\_List\_Id”, we will use this face list id to get the face list. If we send this request, we will get a face list which contains all the faces we recognize from the user’s images.

1. Add a face into face list

Request URL:

[http://westus.api.cognitive.microsoft.com/face/v1.0/facelists/{faceListId}/persistedFaces[?targetFace](http://westus.api.cognitive.microsoft.com/face/v1.0/facelists/%7BfaceListId%7D/persistedFaces%5B?targetFace)]

Request Parameters:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| faceListId | String | The id of the face list we created. |
| targetFace | String | The rectangle area of the face on the image. |

Request Body:

The bytes of the image

Response Body:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| persistedFaceId | String | The unique id of the face in the face list. |

The function of this API is that adding a face we detected from user’s image into the face list which we created using above API. When we use this API to add a face into the face list, we need to provide the id of the face list which we created before and the area of the face location on the image. Then we will get a result of the action, the result contains the face id which was stored in the face list which we created and we can get the face in the next time.

1. Detect faces from image

***Request URL:***

http://westus.api.cognitive.microsoft.com/face/v1.0/detect[?returnFaceId]

***Request Parameters:***

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| returnFaceId | boolean | Return face ids of the detected faces or not. |

***Request Body:***

The bytes of the image.

***Response Body:***

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| faceId | String | Unique face id of the detected face. |
| faceRectangle | Object | A rectangle of the face location on image. |

The function of this API is that detecting the faces on the image which user imported via our system. When we use this API, we need to convert user’s image to bytes and set the ‘returnFaceId’field to true. Then we will get the face unique id and the location of the face on the image.

1. Find the similar faces with the face

***Request URL:***

<http://westus.api.cognitive.microsoft.com/face/v1.0/findsimilars>

***Request Parameters:***

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| faceId | String | Unique face id of the detected face. |
| faceListId | String | The id of the face list we created before. |
| maxNumber | Number | The number of the top similar faces returned. |

***Response Body:***

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| persistedFaceId | String | The persisted id of the face in the face list. |
| confidence | Number | Similarity confidence of the candidate faces. |

The function of this API is that finding the similar faces from the face list which we created to store our user’s images. When we use this API, we need to provide a face id which we detect from the image, the id of the face list which we created before and the maximum number the similar faces returned. Then we will get the persisted id and the similar confidence of the faces.

Above all, we can use these API to finish the work of recognizing face from the image and query the images via a person’s avatar.

1. Computer Vision API from Microsoft

In our system, we just use a part of the API. It is the API of analyzing image which can recognize the objects in the image. So we will only introduce the API which we use in our system.

1. Analyze Image

***Request URL:***

[http://westus.api.cognitive.microsoft.com/vision/v1.0/analyze[?language](http://westus.api.cognitive.microsoft.com/vision/v1.0/analyze%5B?language)]

***Request Parameters:***

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| language | String | The name of the object which language return. |

***Request Body:***

The bytes of the image

***Response Body:***

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| tags | Array | The name and confidence of the objects in the image. |

The function of this API is that recognizing the objects from the image which user imported into our system. When we use this API, we need to provide the language which we want to get and the bytes of the image. Then we will get the objects’ name and the confidence from the server.

1. Bing Image Search API from Microsoft

The aim of using this API is just for enhancing to show the object image. Because when we record our daily life using the device, we will find there is too much objects in the image. If we just show the origin image on the website, user cannot ensure the image stands for which object. So we use this API to show the image which just contains the object.

***Request URL***:

[https://api.cognitive.microsoft.com/bing/v5.0/images/search[?q](https://api.cognitive.microsoft.com/bing/v5.0/images/search%5B?q)]

***Request Parameters:***

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| q | String | The object name which you want to search. |

***Response Body:***

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| value | Array | Information of the images which contains the object. |

The function of this API is that searching the images which only contains the object which you want. When we use this API, we need to provide an object name. Then we will get a lot of images which only contains the object and we can choose one to store into our database for the image of the object.

1. Google Maps API

This API is a library of maven module. So we just depend on it in our project. Our project is managed by maven, as a result, we just add the following code in our pom file:

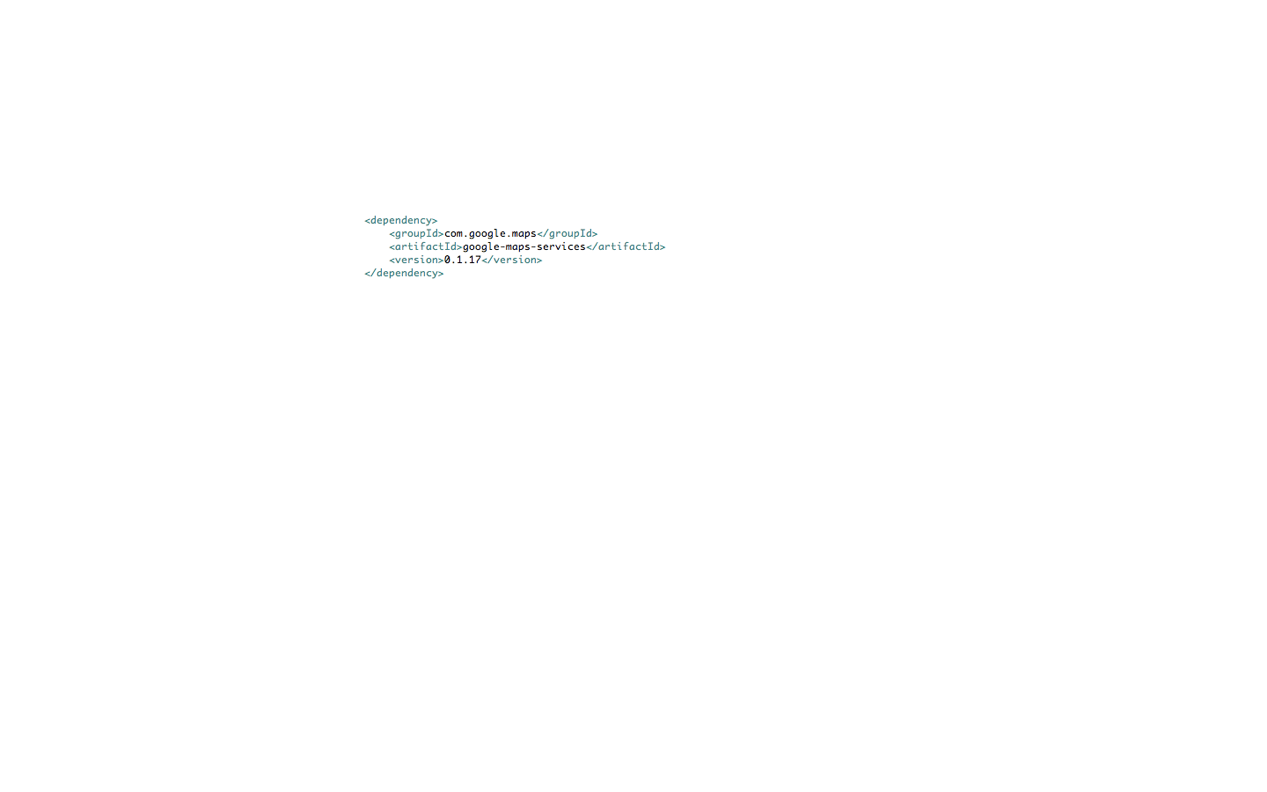
When we import this library into our project, we will find there is a class file to get address via latitude and longitude. In our system, we follow the below steps to get the address from an image:

1. Convert an image into bytes
2. Load latitude and longitude from the image bytes
3. Use the function to get address:

GeocodingApi.reverseGeocode()

1. Done. We already get the address from the image

Google-maps-dependency

1. Internal API
2. Import images into server

The function of this API is that helping user to import their images into our system.

***Request URI:***

/Images/upload

***Request Method:***

POST

***Request Parameters:***

MultipartFile[] images

The realization of this API:

1. Get the bytes of the image from the MultipartFile type
2. Save the bytes of the image into server’s location
3. Handle the creation time of the image
4. Handle the location of the image
5. Recognize the faces in the image
6. Recognize the objects in the image
7. Save all the cues and the image into our database
8. Load avatar image from server to browser

The function of this API is that to show the avatar images for the users in our system. After user imported the images, we will show the avatar images which we deal with the images.

Request URL:

/avatars/download/{id}

Request Method:

GET

Request Parameters:

Long id

The realization of this API:

* + - 1. Get the avatar via the avatar id from database
      2. Load the bytes of the image via avatar picture name
      3. Return the bytes to the response body

1. Load object image from server to browser

The function of this API is that to show the object images for the users in our system. After user imported the images, we will show the object images which we recognized from the images.

Request URL:

/objects/download/{id}

Request Method:

GET

Request Parameters:

Long id

The realization of this API:

1. Get the object via the object id from database
2. Load the bytes of the image via object picture name
3. Return the bytes to the response body
4. Load location image from server to browser

The function of this API is that to show the location images for the users in our system. After user imported the images, we will show the location images which we get from the images via Google Maps API.

Request URL:

/locations/download/{id}

Request Method:

GET

Request Parameters:

Long id

The realization of this API:

1. Get the location via the location id from database
2. Load all the images which are in this location
3. Choose the first image in the image list
4. Load the bytes of the image using image name
5. Return the bytes to the response body
6. Load images via avatar id

The function of this API is to get the images which are contain the avatar in our system. When user view the images in our system, user can double click the avatar image, then we will use this API to get the related images and show to user.

Request URL:

/avatars/{id}/images

Request Method:

GET

Request Parameters:

Long id

The realization of this API:

1. Get the avatar via the avatar id from database
2. Detect the face in this avatar via Face API
3. Find all the similar faces whose confidence is larger than 0.5 in the face list we created via Face API. This process will return the face UUIDs.
4. Load all the images via face UUIDs.
5. Convert the image to bytes, and return to the response body
6. Load images via object id

The function of this API is that to show the images which contain the object. When user view images in our system, user can double click the object image, and then we will call this API to show the images which contain the object.

Request URL:

/objects/{id}/images

Request Method:

GET

Request Parameters:

Long id

The realization of this API:

1. Get the object via the object id from database
2. Load the image via object name
3. Return the bytes to the response body
4. Load images via location id

The function of this API is that to show the images which are in the location. When user view images in our system, user can double click the location image, and then we will call this API to show the images which are in the location.

Request URL:

/locations/{id}/images

Request Method:

GET

Request Parameters:

Long id

The realization of this API:

1. Get the location via the location id from database
2. Load the bytes of the image via location description
3. Return the bytes to the response body
4. Query images via cues combination

The function of this API is that to query the images in our system when user choose a cues combination. After user imported images in our system, if he/she want to view images using cues combination, the system will call this API to show the images for user.

Request URL:

/images/query

Request Method:

GET

Request Parameters:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| faceIds | Array | The id collection of faces |
| objectIds | Array | The id collection of objects |
| locationIds | Array | The id collection of locations |
| times | Array | The time collection |

The realization of this API:

1. Use the time to filter the images
2. Use the objects to filter the images
3. Use the lcoations to filter the images
4. Use the avatars to filter the images
5. Return the image list
6. To be decided
   1. To be decided