Course Number: PE076DS

Class: 8

Group Project: Photo Share Share

Group Members:

Ma Wai Kin

Tse Sui Man

Zeng Shuyue

Tsang Yuen Yu

Lai Wing Wa

1. **Introduction**

**1.1 Overview**

The summary provided describes a full-stack responsive photosharing application built

using MongoDB, Express.js, and Node.js. The application follows a three-layered architecture and Model View Controller(MVC).

The application offers user registration with comprehensive validation. Users can register, sign in by password, create their own display name, upload their profile picture and access to a clean and user-friendly Album list. Each user can have one photo album and users can upload, edit, and delete photos in their own photo album. Users can also update their uploaded photos’ specification including ISO, aperture, shutter, EV and etc. Users can view all users’ photo album, visit to every photo album and present their favor over the photos by clicking the “like” button (also “unlike” if they want to do reversal). The number of Like count is collected and saved in the database MongoDB. The number of Like count for each photo will be shown in the page of photo list. Further, the users can see the photo details in the page of photo details.

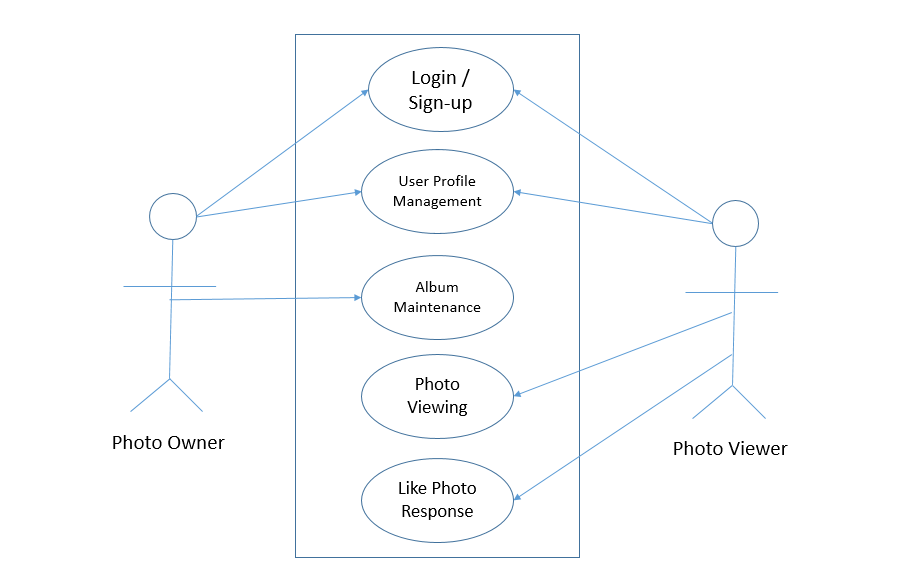
On the backend, Node.js serves as the running environment, while extended module Express.js acts as the backend framework. MongoDB used as the database is employed for managing database operations. Authentication is processed by using JSON Web Tokens (JWT) and password hashing function Bcrypt. Multer is used for file uploading. Middleware Node.js API is implemented to authenticate HTTP requests before sending them to the server.

On the frontend, HTML and JavaScript are implemented to build the framework with extensive use of Bootstrap templates.

The goal of the project is to implement the fundamental elements of the CRUD (Create, Read, Update, and Delete) by using operations including MongoDB, Express.js, and Node.js platform. Using the fundamental features of a photosharing web application, such as sign-up and sign-in, upload, update, and delete the album photos, like and unlike function.

**1.2 Use-Case Diagram**

From the perspective of photo owner, they can have three use functions including login/sign-up, user profile management, and album maintenance. From the perspective of photo viewer, they can have four functions including login/sign-up, user profile management, photo viewing, and like photo response.



* 1. **Technologies Used**

a. Node.js

b. Express.js

c. MongoDB

1. **Setup and Installation**
   1. **Prerequisites**

Node.js is a cross-platform server system application and runs on the V8 JavaScript engine. Node.js can be used on Windows, Linux, Unix, macOS, and executes JavaScript code outside a web browser. By using node.js, developers can use JavaScript to write command line tools and for server-side scripting. Node.js allows user to run JavaScript code on the server in order to generate dynamic web page content before the page is sent to the user's web browser. As a result, node.js represents a "JavaScript everywhere" paradigm, unifying web-application development within a single programming language, as opposed to using different languages for the server- versus client-side programming.

Express.js is an extended module from node.js. It offers a wide range of cutting-edge features for web and mobile development. Express.js provides more tools and functions to the developers with better programming environment.

Outsides SQL, MongoDB is one of the most popular source of database with high availability and flexibility. MongoDB is a cross-platform database that uses a collection and document based strategy to create sharp output, high availability, and simple scalability. It helps increasing the efficiency and functionality.

Software Configuration:

|  |  |
| --- | --- |
| **Software** | **Version** |
| Node.js | Node.js v22.11.0 |
| Express.js | Express.js 4.21.1 |
| MongoDB | MongoDB 6.10.0 |
| Bcrypt | Bcrypt 5.1.1 |
| Body-parser | Body-parser 1.20.3 |
| Cookie-parser | Cookie-parser 1.4.7 |
| Debug | Debug 2.6.9 |
| Dotenv | Dotenv 16.4.5 |
| Ejs | Ejs 3.1.10 |
| Express - session | Express – session 1.18.1 |
| morgan | Morgan 1.10.0 |
| Multer | Multer 1.4.5-lts.1 |
| Nodemon | Nodemon3.1.7 |
| Serve-favicon | Serve-favicon2.5.0 |

* 1. **Installation Guide & Running the Application**

Prerequisites

Before installing this project, make sure you have the following prerequisites:

Node.js (version 14 or higher)

MongoDB (version 4.4 or higher)

install git for window/mac/linux

Installation Steps and Running Application

Clone the repository:

git

clone

https://github.com/erbbep8/erbbep8.git

Navigate to the project directory:

cd

final-proj

Install the dependencies:

Set up the environment variables:

Create a .env file in the root directory of the project and add the following content:

dbConnectString=mongodb://localhost:27017

Replace mongodb://localhost:27017 with your MongoDB connection string.

Set up the MongoDB database and start the MongoDB server:

npm install

mongod --dbpath data/db

Start the server:

npm start

Access the application in your web browser:

http://localhost:8888

Configuration

The project uses the config.js file for configuration. You can customize the configuration by modifying the values in the config object.

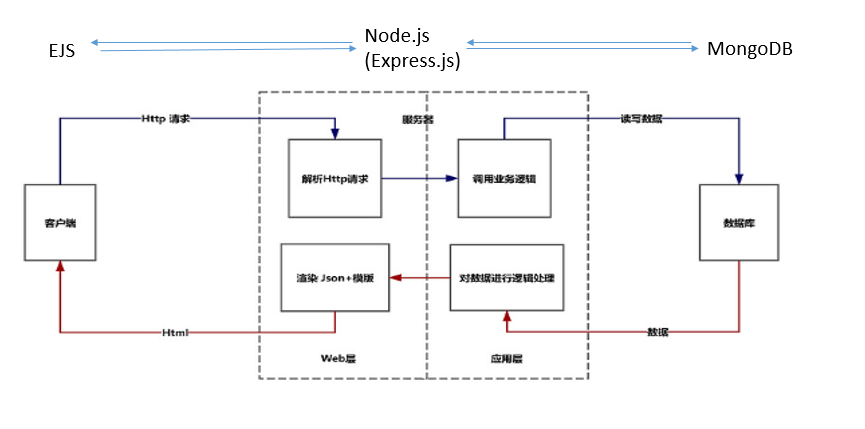
Notes

The data/db directory is used to store the MongoDB data. Make sure to create it before starting the MongoDB server.

That's it! You should now have a working installation of the project.

1. **Architecture and Design**
   1. **System Architecture**

Structure Diagram



This system architecture for a web application using EJS (Embedded JavaScript), Node.js, Express.js, and MongoDB.

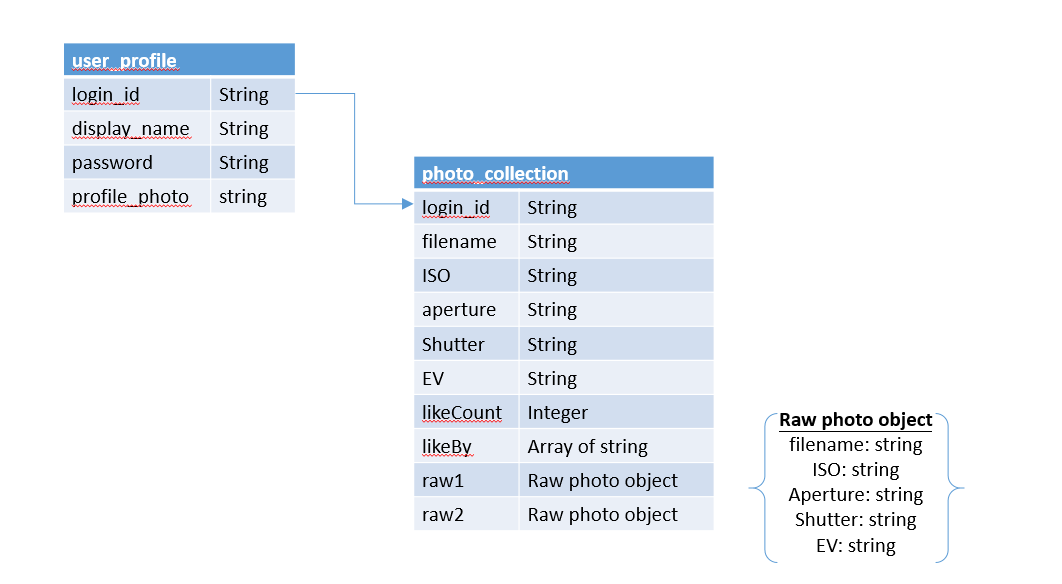
Client (Front End), These are used HTML/CSS/JavaScript to build the structure, style, and behavior of the web pages. Used Embedded JavaScript (EJS) templates for server-side rendering of HTML. Used Fetch API to make HTTP requests to the server for data.

Server, Used Node.js a lightweight framework to build web applications and APIs. Used Router Layer Defines the application's endpoint routes and handles HTTP requests. Used Controller Layer Contains the business logic and interacts with the Model Layer to process requests. Used Model Layer defines the data schema and interacts with MongoDB to perform CRUD operations.

Database, Use a NoSQL database MongoDB to store application datas . There are include two collections that stores user information and stores information related to photos.

**3.2 Database Schema**

ER Diagram



**3.3 Data Dictionary**

Collection: user\_profile

|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Description** |
| login\_id | String | Unique identity used to enter into the system |
| display\_name | String | Name of the user used to show in system |
| password | String | Hash value of secret key used by the user to log-in to system |
| profile\_photo | String | Filename of an image file used by the user as his/her profile picture |

Collection: photo\_collection

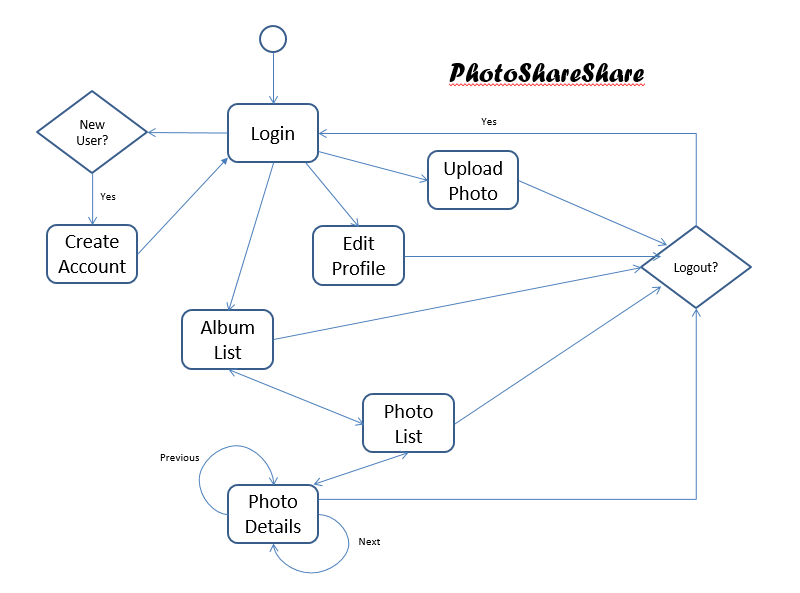
|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Description** |
| login\_id | String | Used as identity of album folder owned by the user |
| filename | String | Filename of a photo uploaded by the user |
| ISO | String | Technical aspects of a photo |
| aperture | String |
| shutter | String |
| EV | String |
| likeCount | Integer | Total number of like response on a photo |
| likeBy | Array of string | Login ID of user(s) who response like a photo |
| raw1 | Raw photo object | Raw photos used to compose the photo in this record |
| raw2 | Raw photo object |

Object: raw photo

|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Description** |
| filename | String | Filename of a raw photo uploaded by the user |
| ISO | String | Technical aspects of a photo |
| aperture | String |
| shutter | String |
| EV | String |

**3.4 System Overview**

State Diagram



Assumption and Limitation

As a pilot project, few assumptions have been made to simplify system implementation process. Those assumptions are listed as below: -

Each user is assumed to have one and only one album

Each multi-exposure photo has maximum of 2 raw photos

This project is assumed as low usage volume application

Due to above assumptions, system inherits with below limitation: -

Each photo belongs to one and only one album. It cannot be shared among different album.

System will create an album for each new user, and cannot be changed. User do not have rights to create any additional album.

System only provides two slots for each photo to upload raw photos in photo upload module

No performance issue is considered when design and implement this application.

User Account Management

System allows user to create/modify/remove its own user record. Besides of user ID, system will also record his/her chosen name showed in system and profile picture (optional).

Album Management

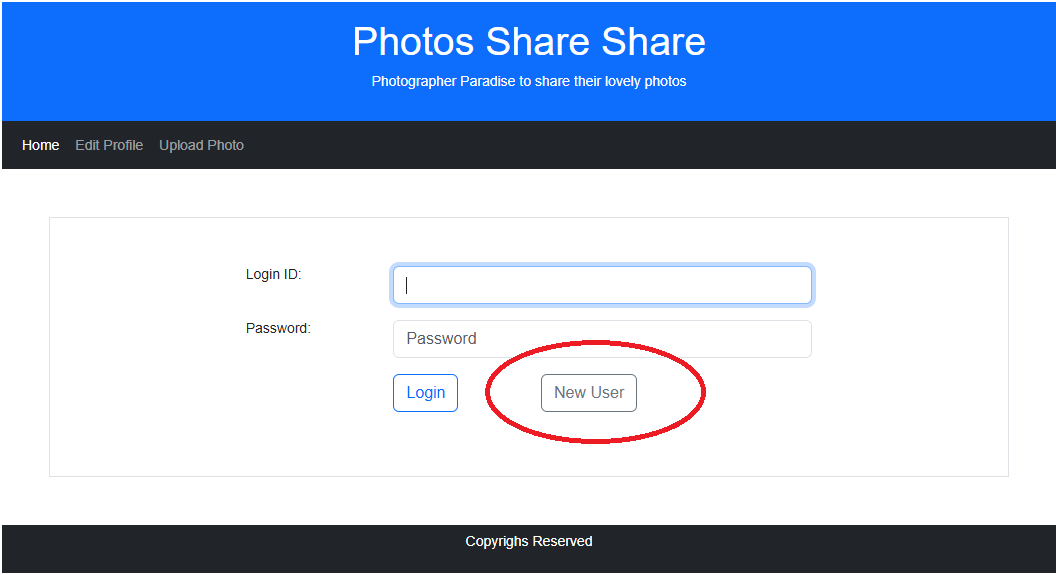
Each user has its own album. User can upload its personal photo into its own album. Photo in the album can also be removed.

Photo Browsing

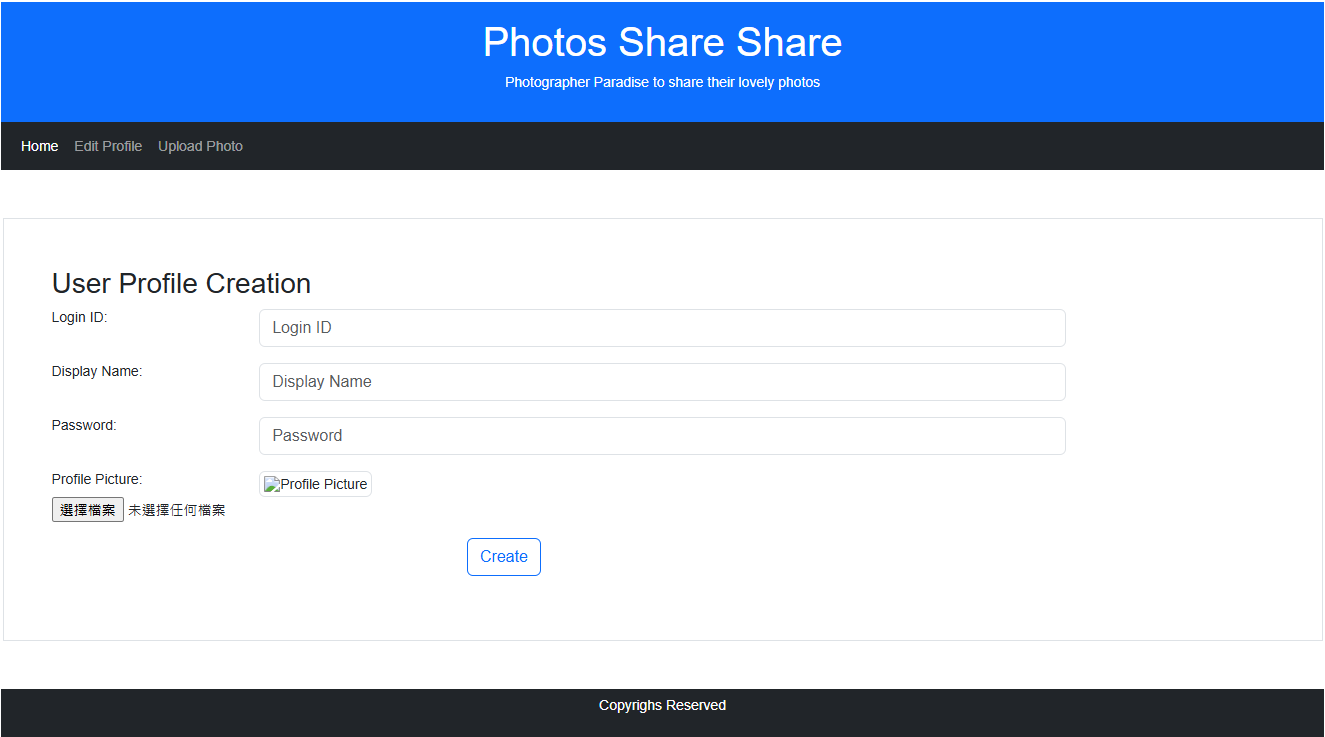
User can browse all albums recorded in the system. At photo detail level, user can also view raw photos of those multi-exposure photo. Like icon is provided for user to show appreciation on any photo.

1. **User Management**

**4.1 Sign Up**



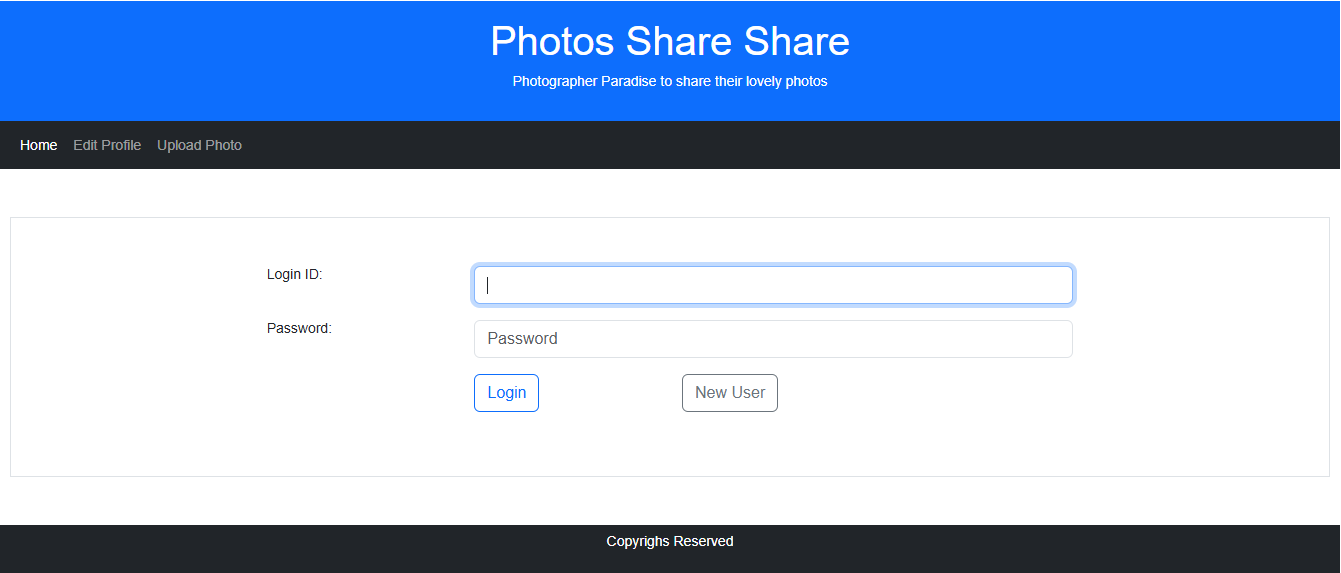
For new user who do not have account in the system, click <New User> button in log-in screen. It will lead to below new user creation screen.



Except for profile picture, all fields are mandatory. After user input all required information, press <Create> button to generate a user record. If success, system will direct to login screen then.

**4.2 Login and Logout**

To log-in into system, enter log-in and password in log-in screen. Then press <Log-in> to proceed

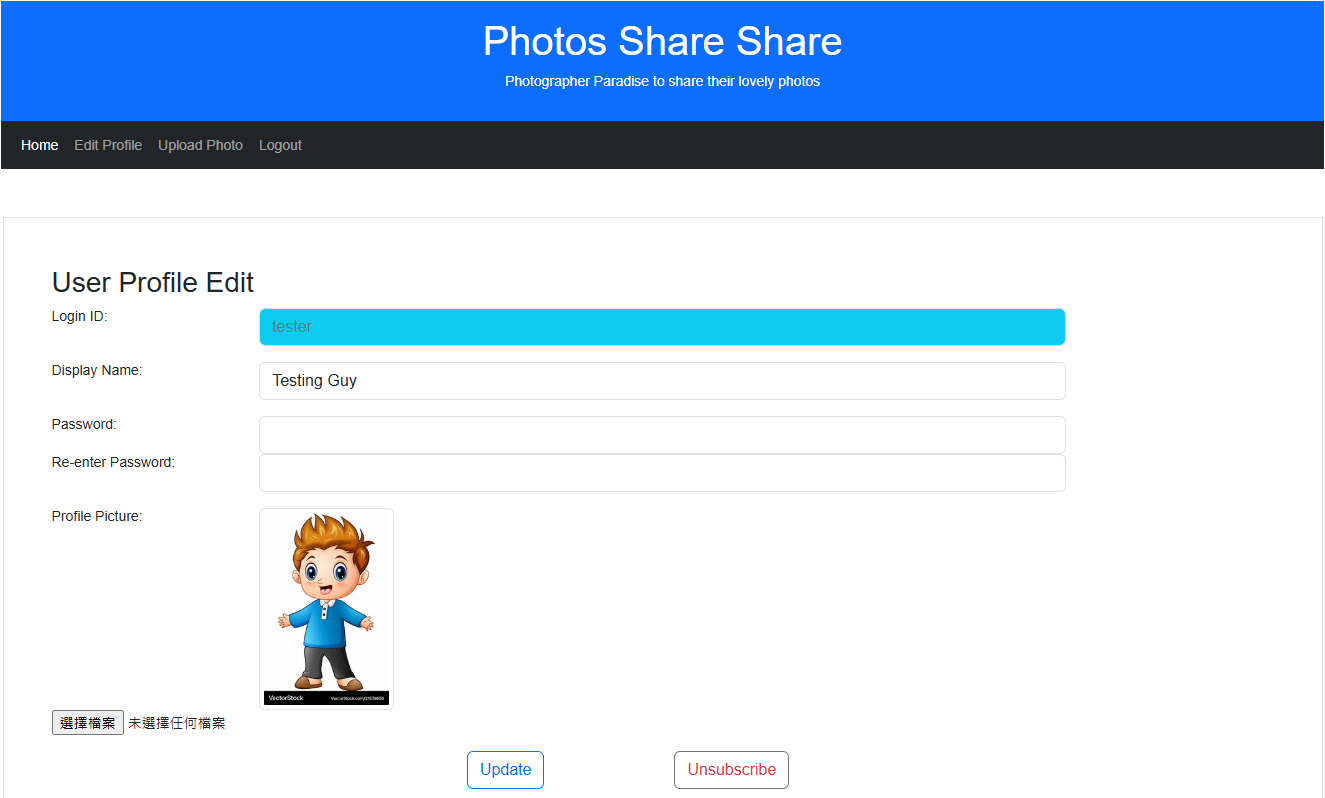


In main menu, there is a <Logout> menu item. Press it to log-out from system. System will then direct to log-in screen.

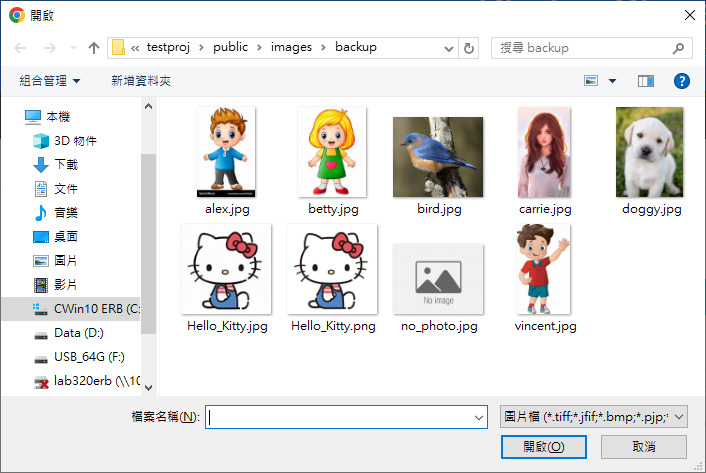


**4.3 User Profile Edit**

Created user profile can be edited by its owner. After log-in to system, press <Edit Profile> menu option will lead to user profile edit screen, similar as below: -



To change profile picture, press <Choose File> button. A file selection dialog box will pop-up.

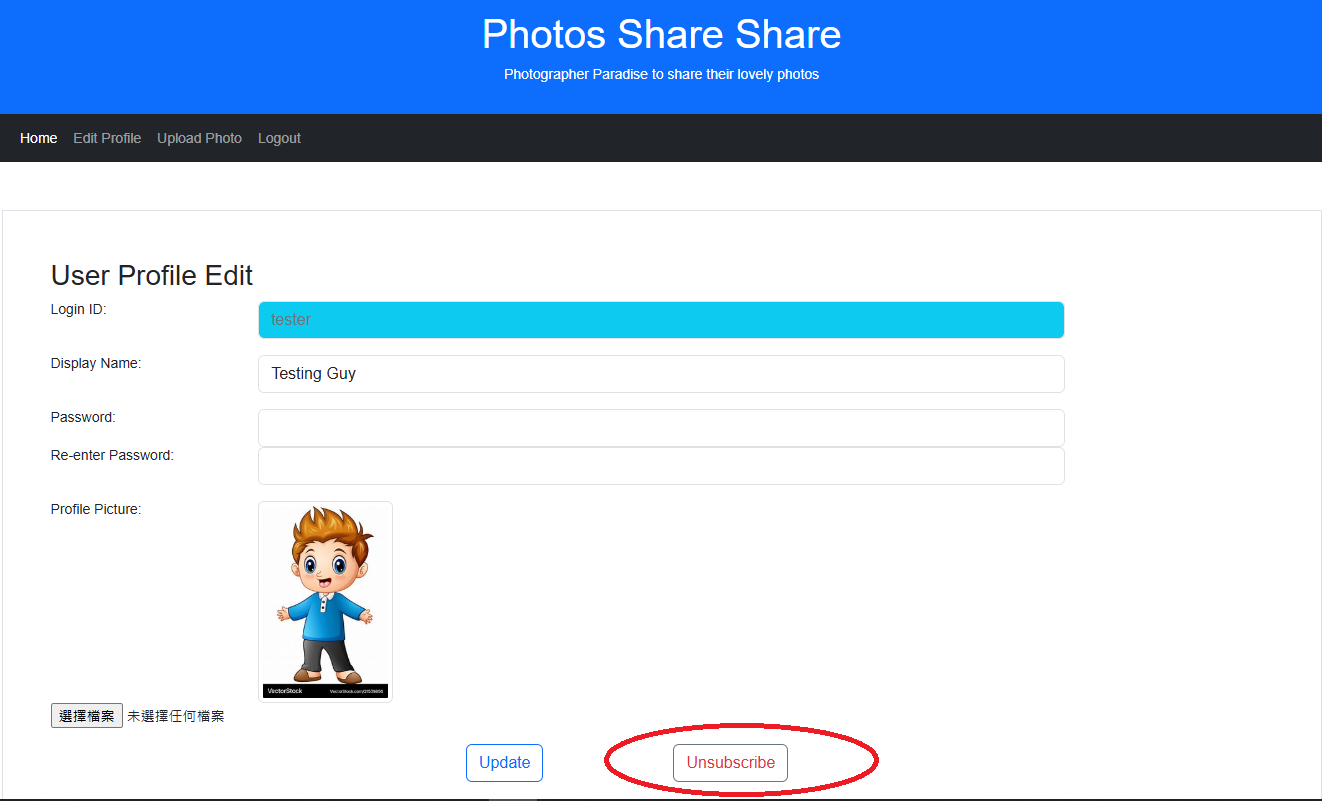


choose an image file, then press <Open> button

After necessary changes has been made, press <Update> to save changes.

**4.4 Un-subscription**

In user profile edit screen, there is a <Unsubscribe> button. Click to remove user’s own user record from system.



Please note that all uploaded photos will be removed also. This process is IRREVERSIBLE!

After process done, system will direct to log-in page.

1. **Photo Management**
   1. **Photo Upload**

Photo File Upload Guide

Introduction

This guide will walk you through the process of uploading photo files to the application. The application allows users to upload main photos along with their raw files. The uploaded files are stored in the data dictionary under project's root path, also stored a related record in the MongoDB database under the photo\_collection collection and can be accessed through the application.

Uploading Photo Files

Open your web browser and navigate to the application's URL:

http://localhost:8888

Log in to the application using your credentials.

Once logged in, you will be redirected to the dashboard. Click on the "Upload" button to navigate to the file upload page.

On the file upload page, you will see a form with the following fields:

Main Photo: Click on the "Choose Main Photo" button to select the main photo file from your computer.

Raw Photo 1: Click on the "Choose Raw Photo 1" button to select the first raw photo file from your computer.

Raw Photo 2: Click on the "Choose Raw Photo 2" button to select the second raw photo file from your computer.

After selecting the files, you can enter the following optional information for each photo:

ISO

Aperture

Shutter

EV +/- (Exposure Value)

Once you have filled in the required fields, click on the "Upload" button to upload the files.

The application will display a success message along with the uploaded files' names.

Notes

The application supports the following image file types: APNG, BMP, GIF, JPEG, PNG, SVG, TIFF, and WEBP.

The maximum file size for each image is 100MB.

The uploaded files are stored in the data dictionary under project's root path and also stored a related record in the MongoDB database under the photo\_collection collection.

Troubleshooting

If you encounter any issues while uploading photo files, please check the following:

Make sure the selected files are in the supported image formats.

Ensure that the file size of each image does not exceed the maximum limit.

Verify that the MongoDB server is running and the application is properly connected to the database.

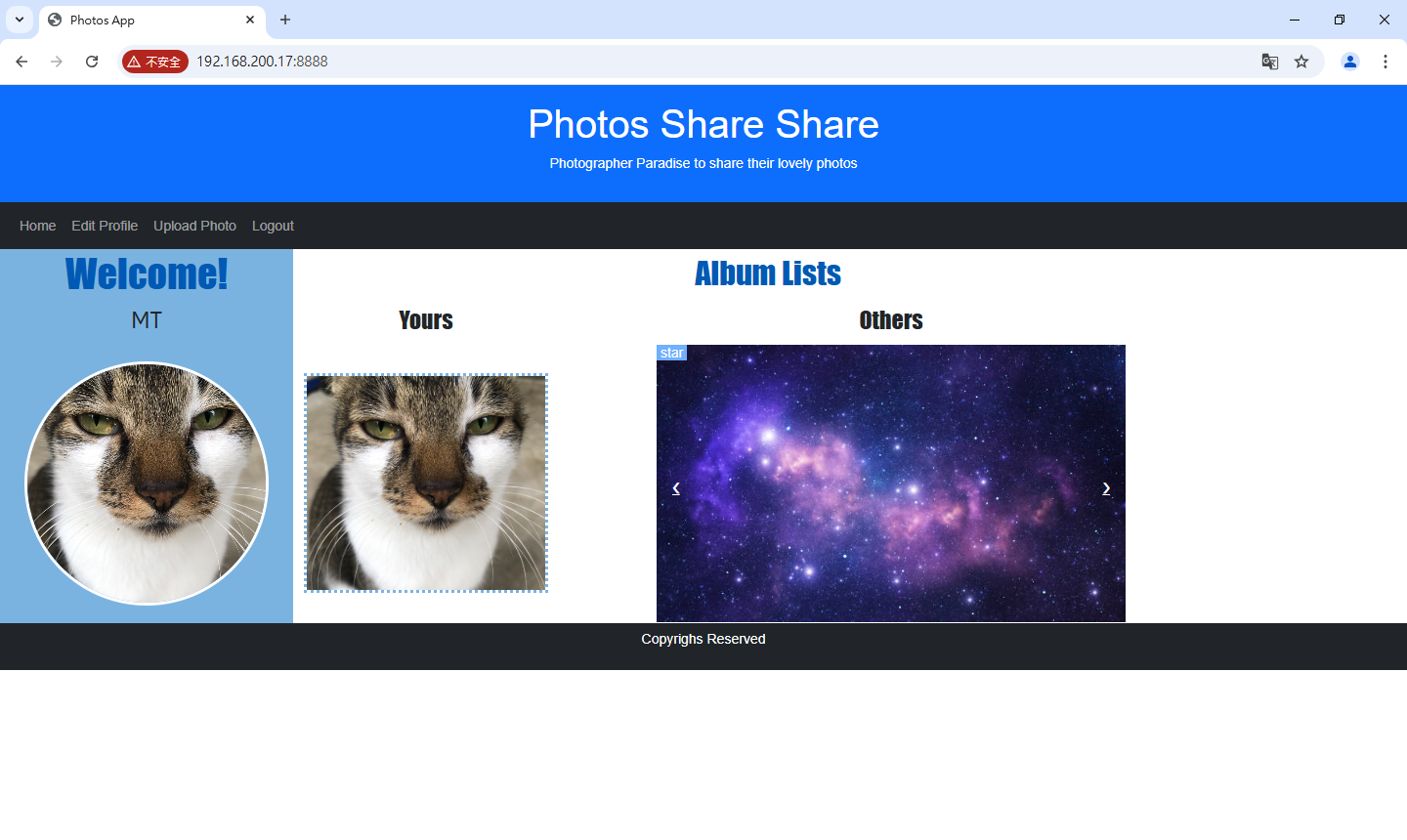
If the issue persists, please contact the application support team for further assistance.

* 1. **Photo Retrieval and Display**
     1. **Album List**

This is the first page after logged-in which lists out the main contents of our website including user’s profile, user’s own album and the albums of other users. The presentation of this page looks clear and simple, which is definitely in accordance with our aims. The purpose is that to create a website for the photography lovers who can easily upload their own photos as an album and check out the others’ albums if they want to. Now, let me invite you to go through our tour of this page, you might be having a wonderful site-seeing experience.

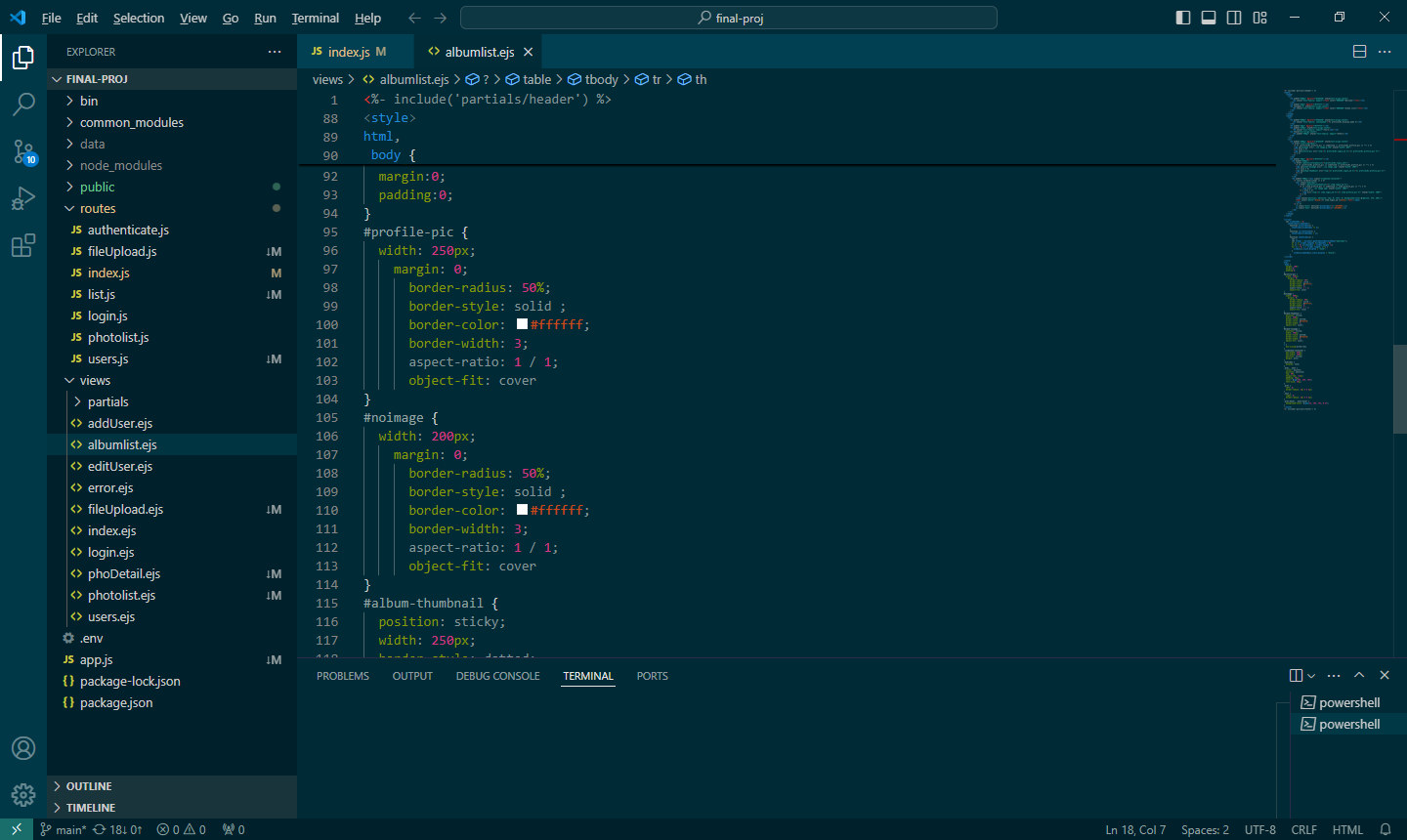
First step of all, I would like to introduce an easy-to-read layout style, which is divided the page into two main parts horizontally by different background colours. As you see, the main colour tone of our page is blue, so we have used different lightness of blue colour on the website header, one-side background and the heading of the page to enhance the layering.

About the layers, there is light blue background colour on the left-hand side with an eye-catching word “Welcome!” showing on the top (showing on Pic1). It is also a classic but important message on communication with users because we would like to tell them, “this is your own page, just feel free to do your job”. This is about the first impression of the user on using a page. We surely cannot miss it. After that, there is showing the logged-in user’s profile included display name and profile picture underneath the greetings in order to emphasize the message of HOME PAGE.



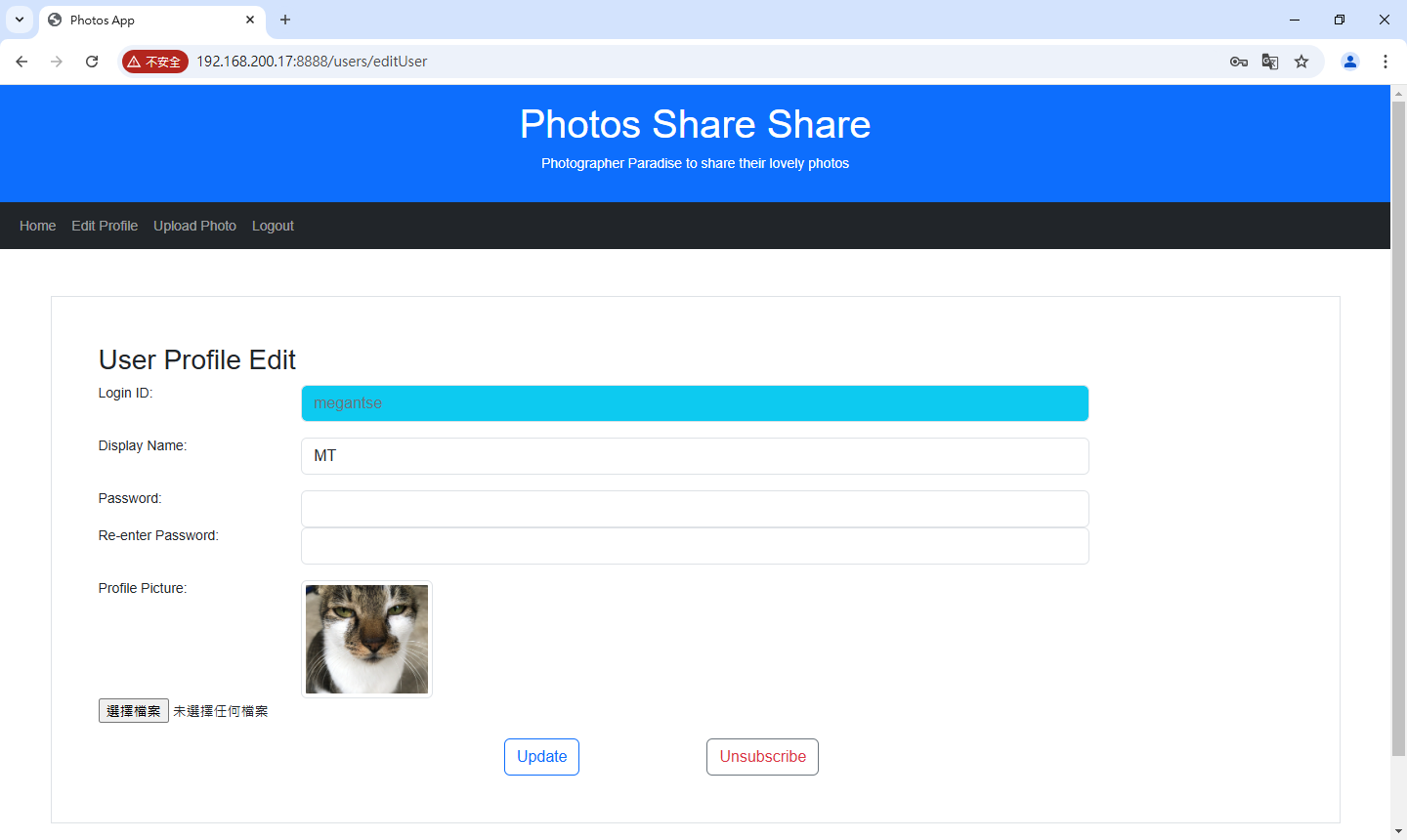
Pic1

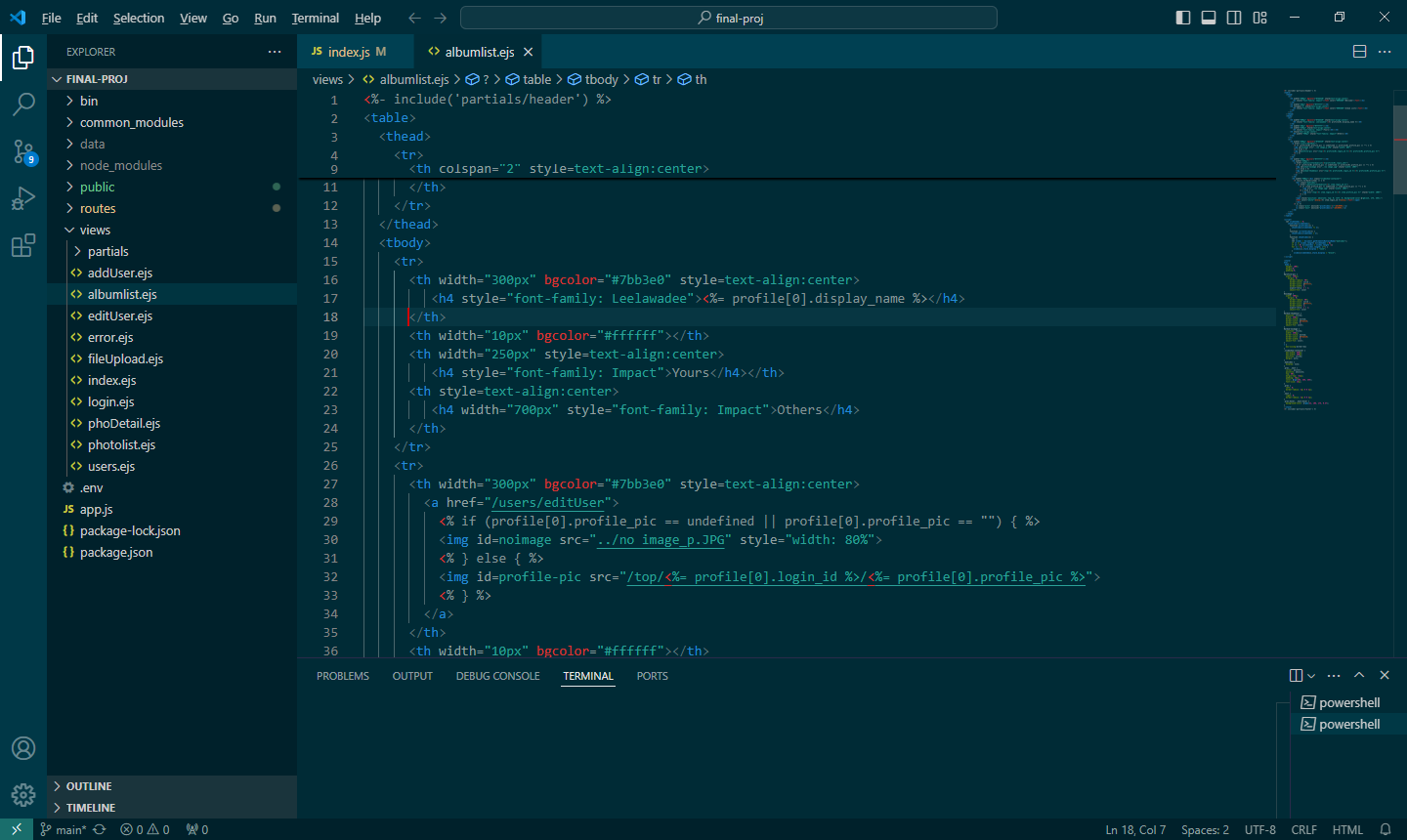
Let’s talk about the spot here about the profile picture. We have used CSS to design its display style (check out the coding on Pic2). In CSS, 50% on border-radius is for showing the circle shape. And the result of border-color and border-width is calling to the edge surrounding the profile picture.



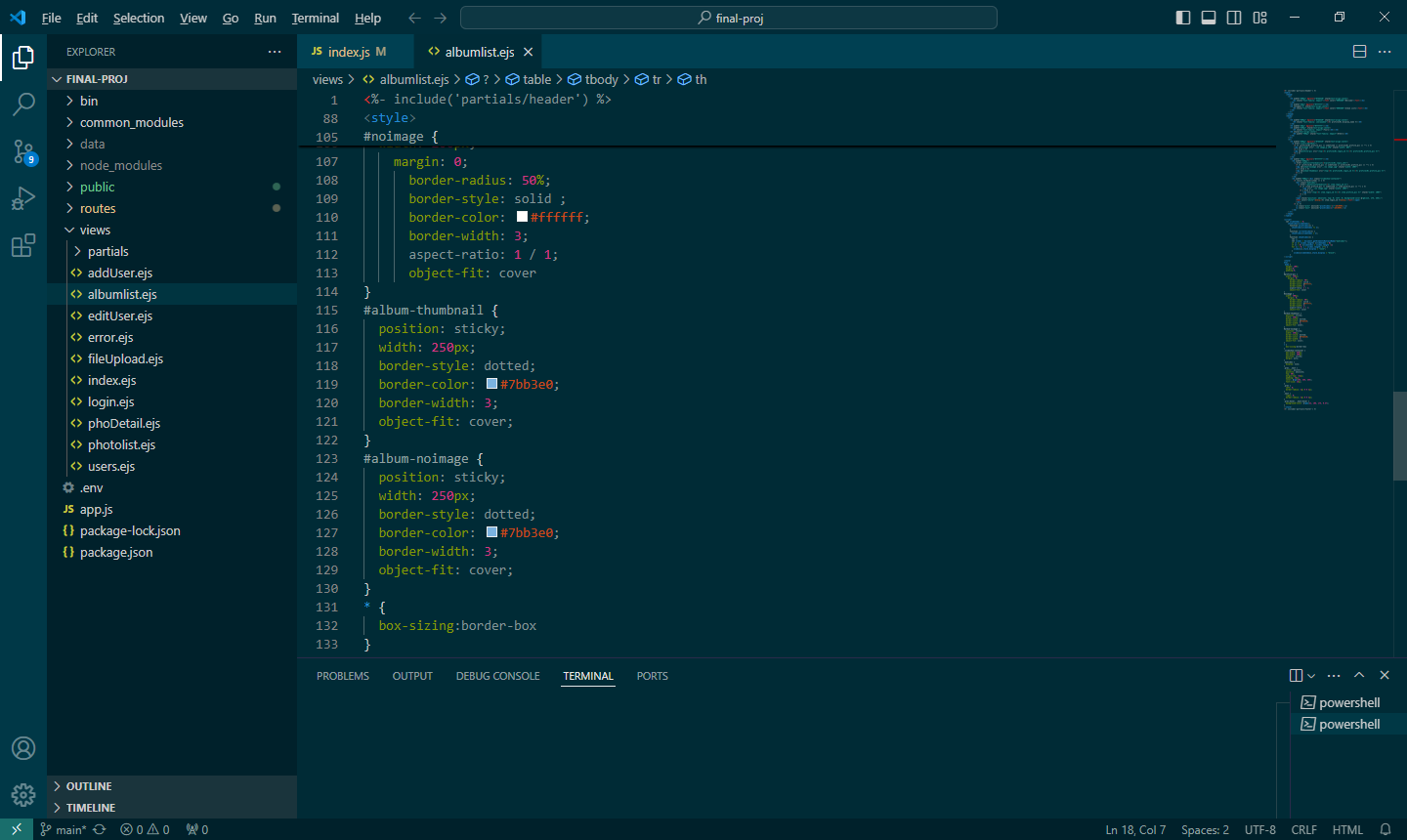
Pic2

Also, when you move your mouse over on the profile picture, there is a surprise waiting for you. YES! A hyperlink which links to “User profile edit” (check it out on 4.3). You are able to change your own details listing out there whenever you want except Login ID (Showing on Pic3).

Pic3

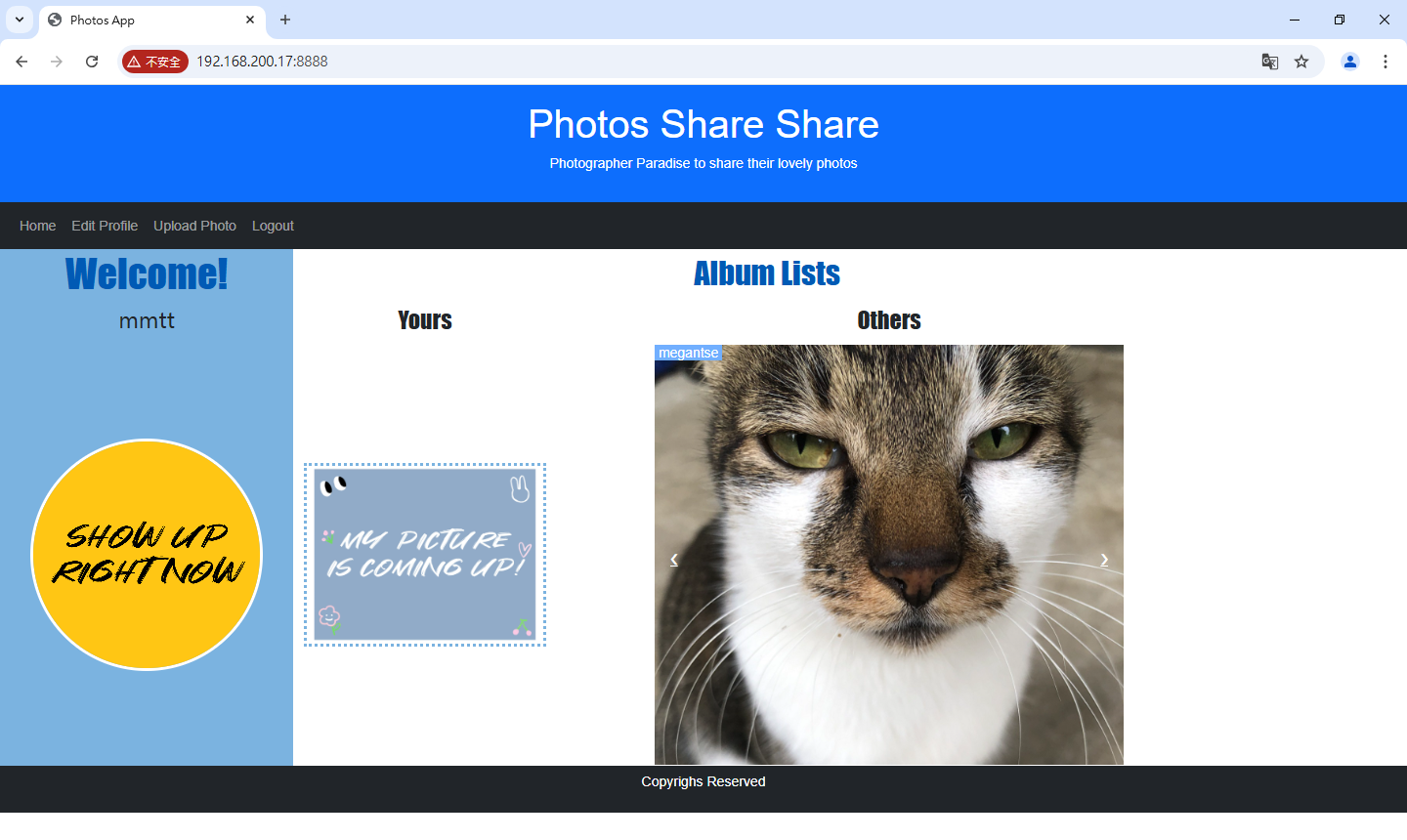
Next, let’s move to “Album Lists”! You can see that there are “Yours” and “Others” under “Album Lists” on the right column. Obviously, the photo cover showing under “Yours” which is the entrance of your album. Its display style has been written by CSS and html (check out the coding on Pic4 & Pic5).

Pic4

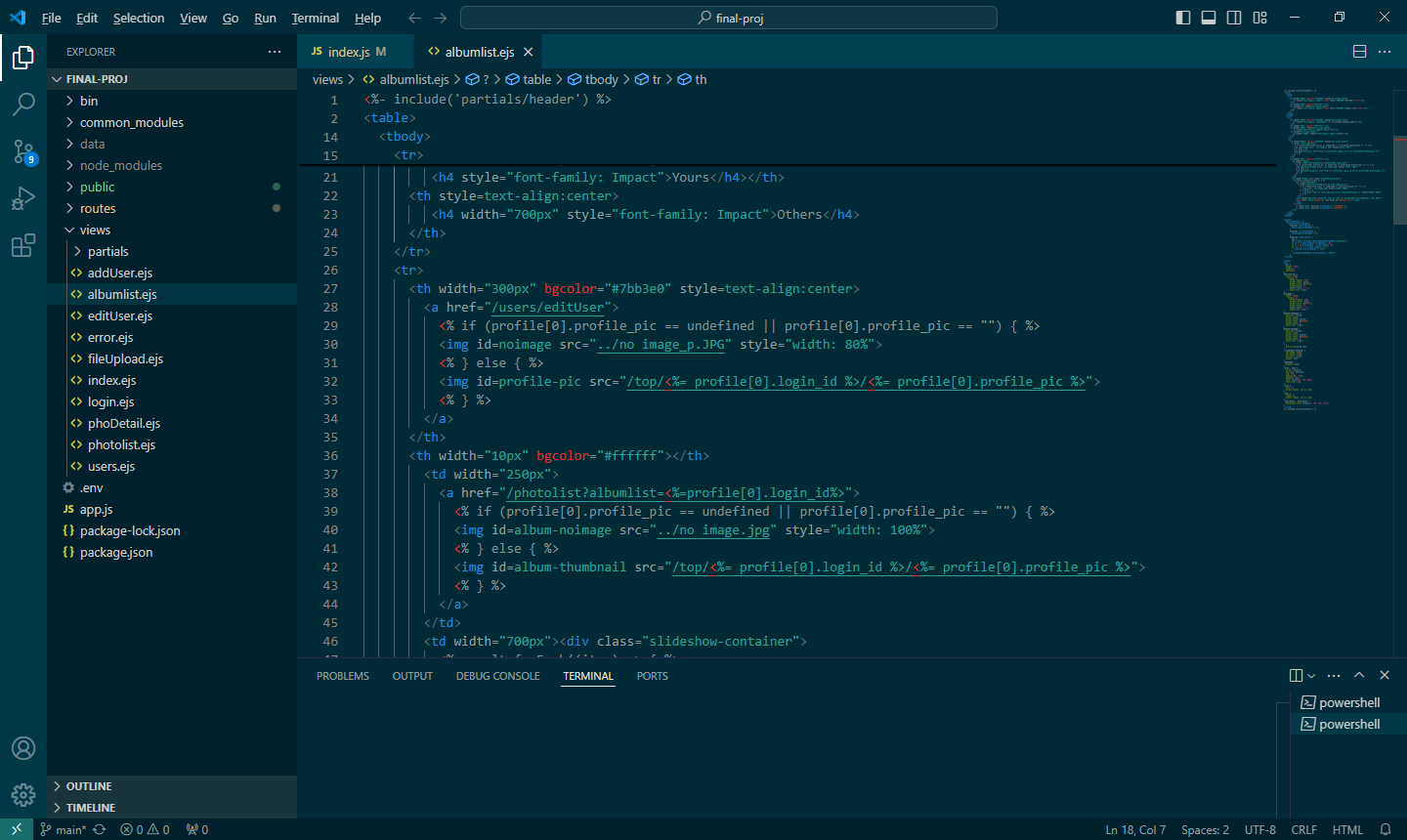


Pic5

When you click on your album cover, it contains a hyperlink which will bring you to the next page named “Photo List” (check it out on 5.2.2). There is your own album, listing out all the photos you have uploaded. But if you haven’t uploaded your profile or no photo on your album? No worries! We have already prepared a default photo for you (Showing on Pic6 & check out the coding on Pic7).

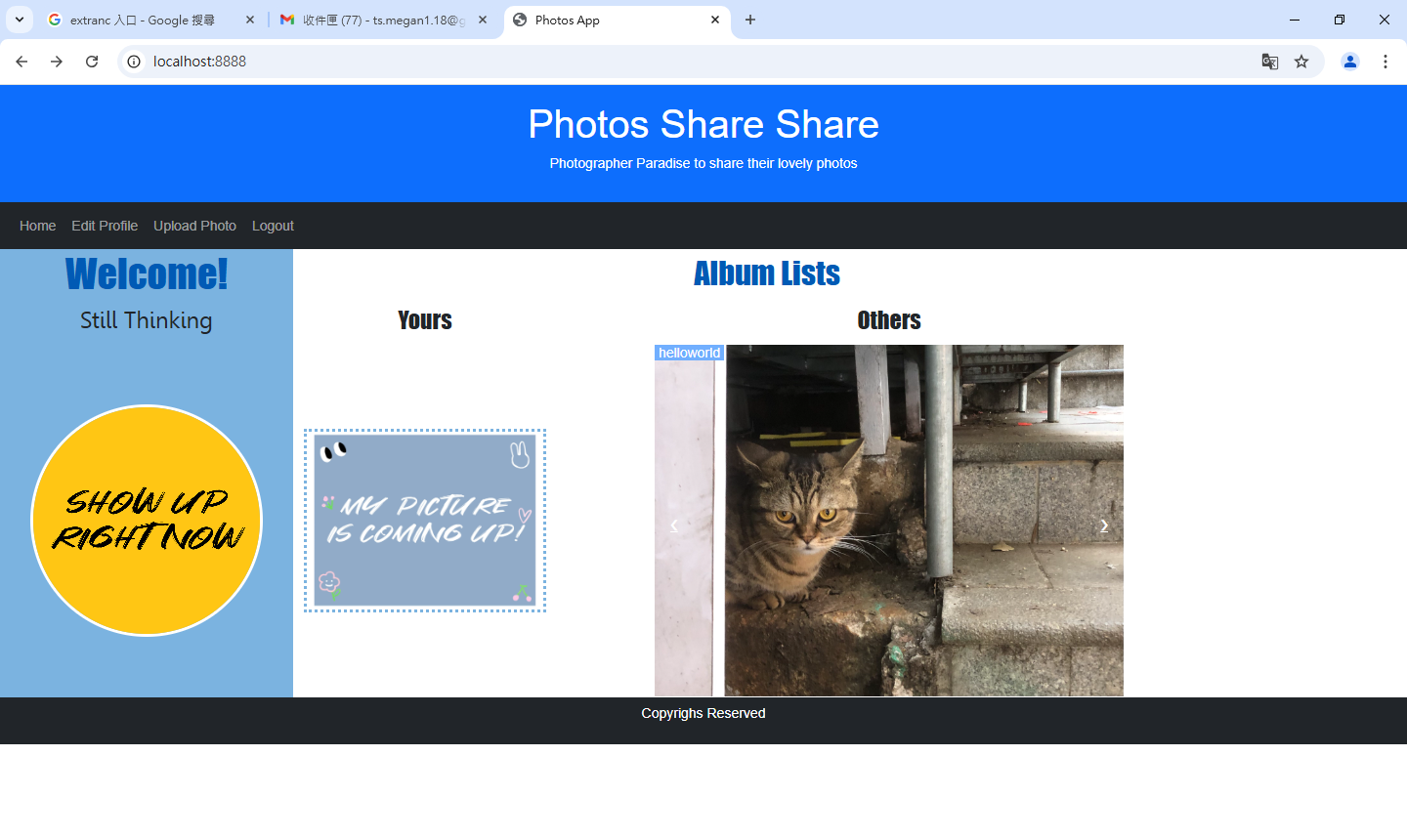


Pic6

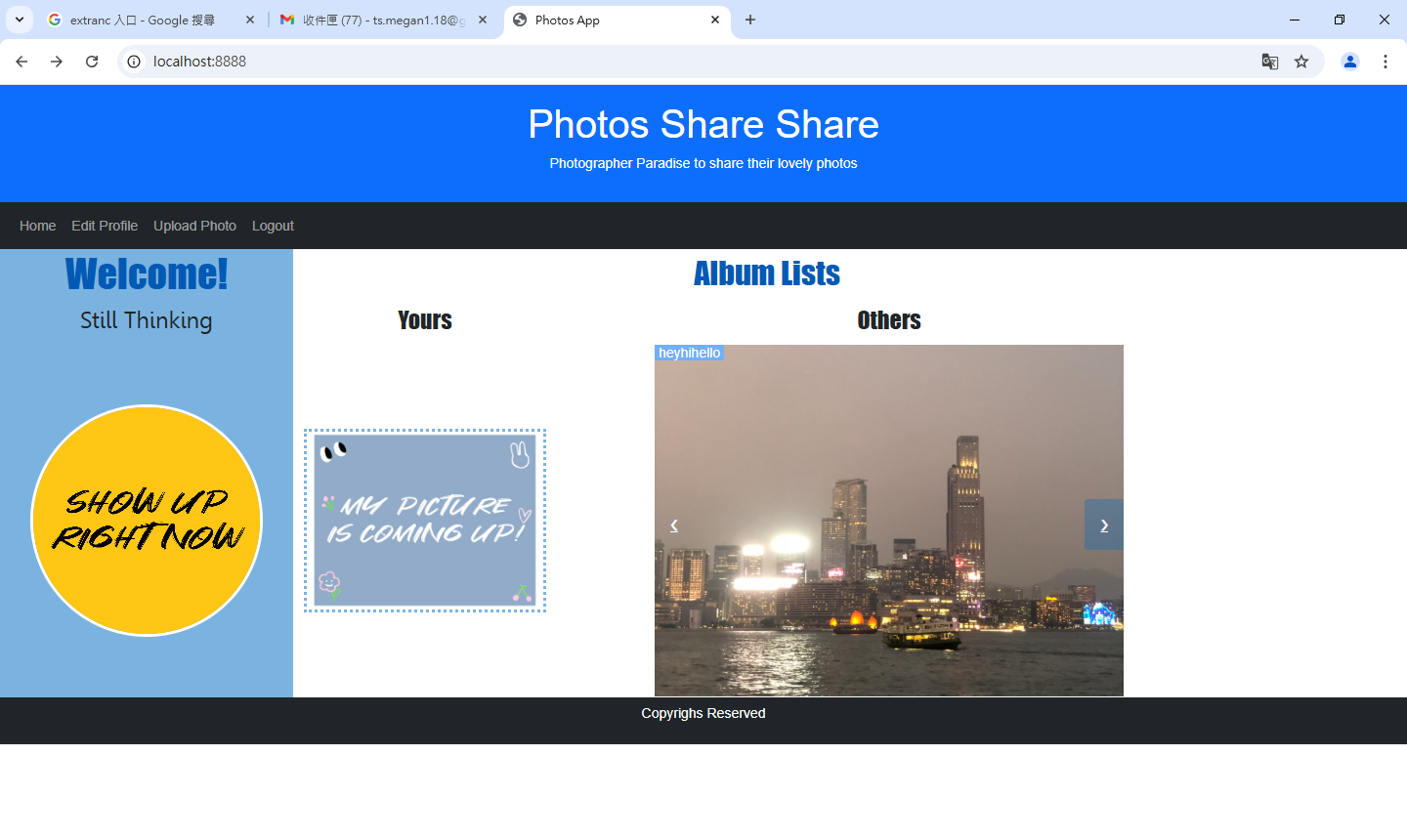


Pic7

So, how about “Others”? You can check out other users’ albums which are showing under the sub-title named “Others” (Showing on Pic8). All of them are signed-in users of “Photos Share Share”. You have right to choose and check out their albums by clicking on their photo cover with a display name. When you move your mouse over on the little white arrow, there is a colour-change to remind you there are another options for you (Showing on Pic9).

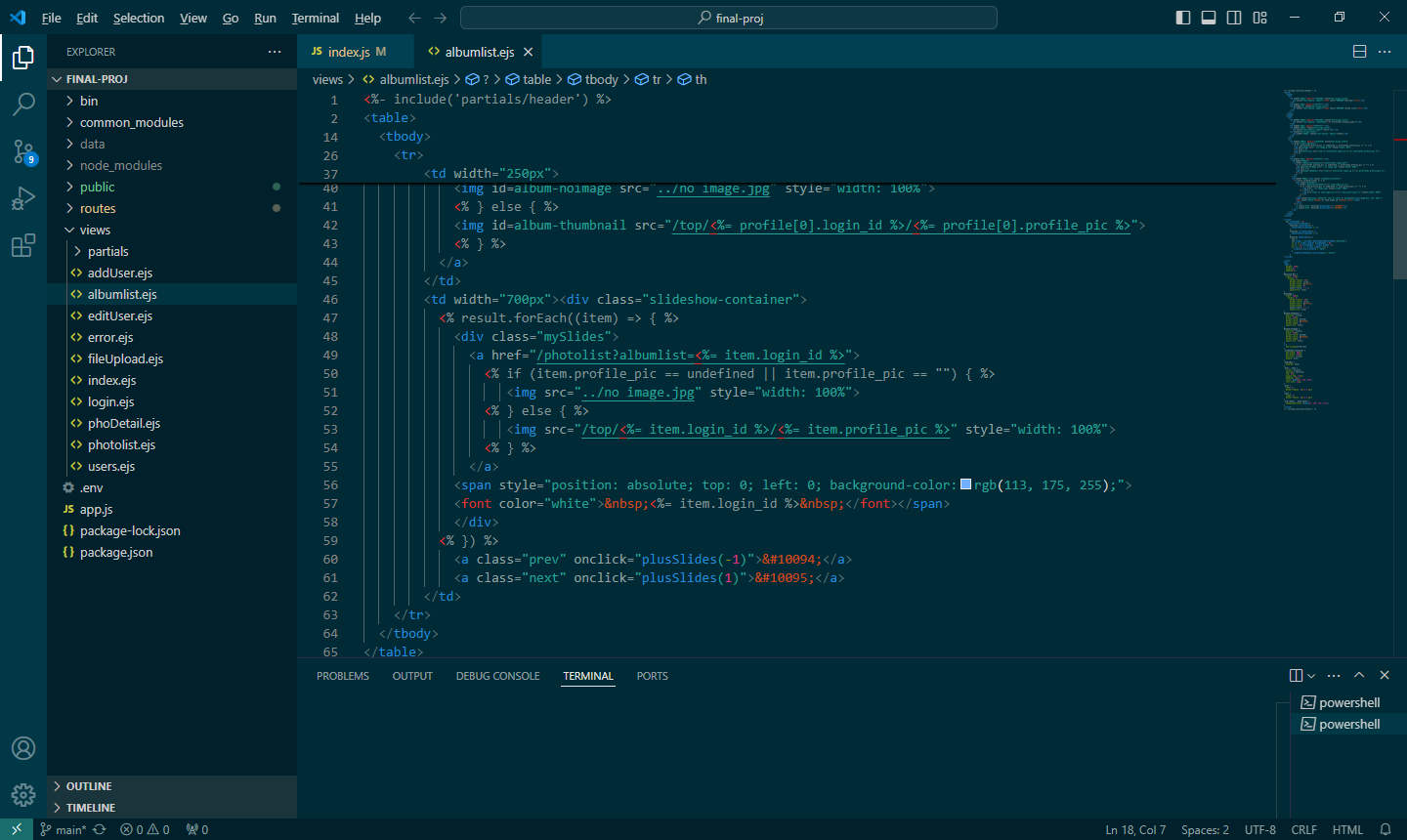


Pic8

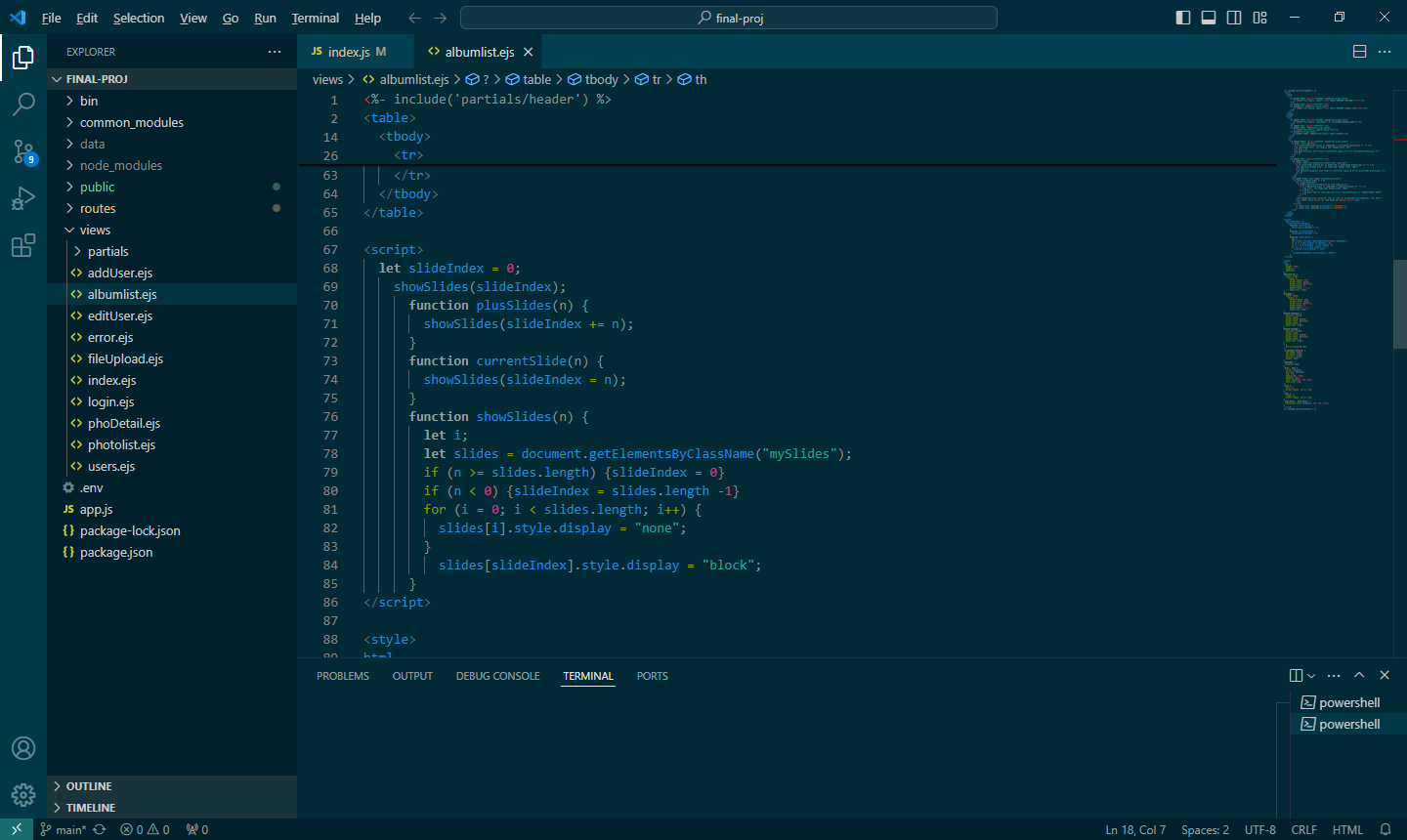


Pic9

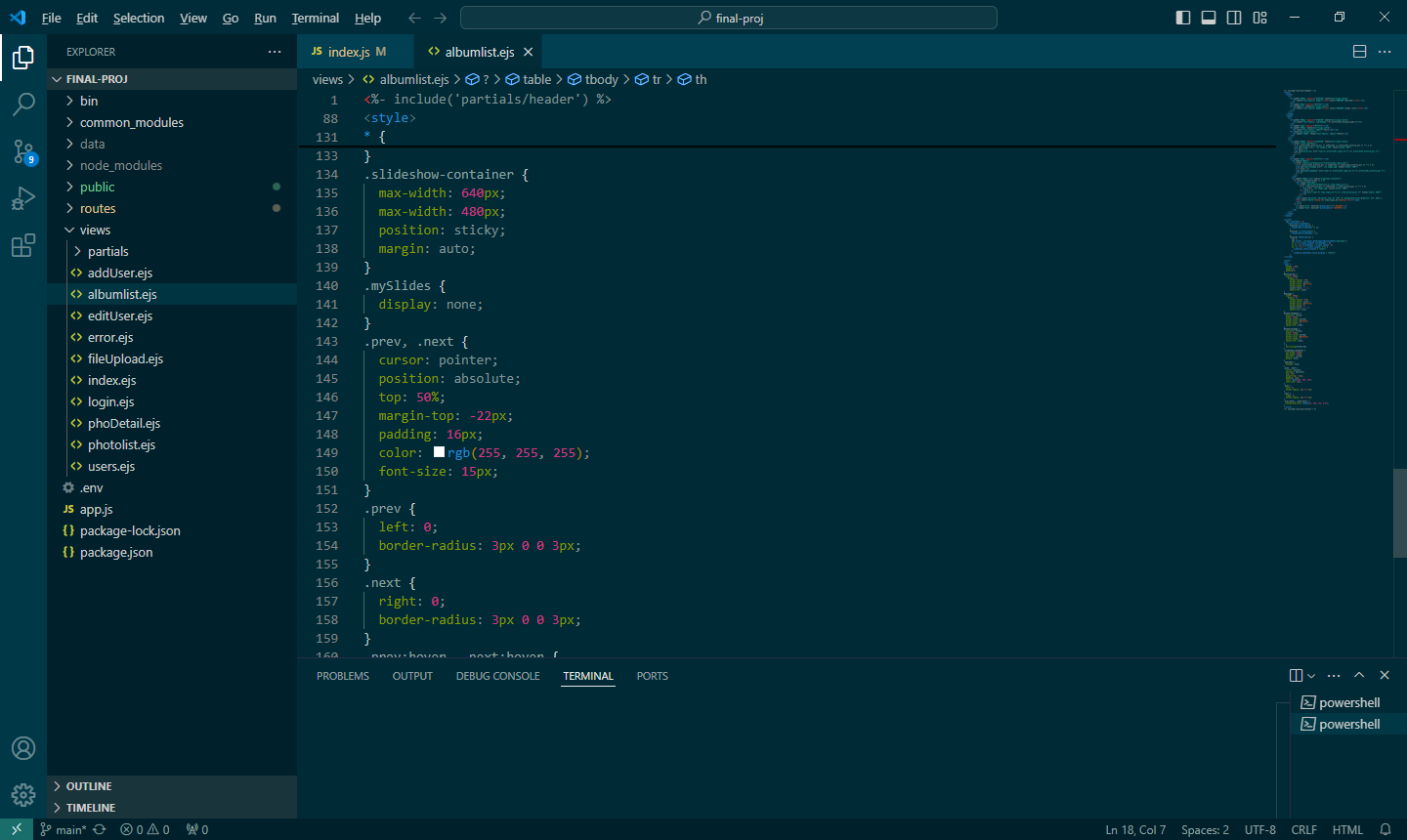
If you want to know how does it work. Please check out our coding on Node.js. We have written with CSS, html and JAVASCRIPT (Showing on Pic10, Pic11 & Pic12).



Pic10

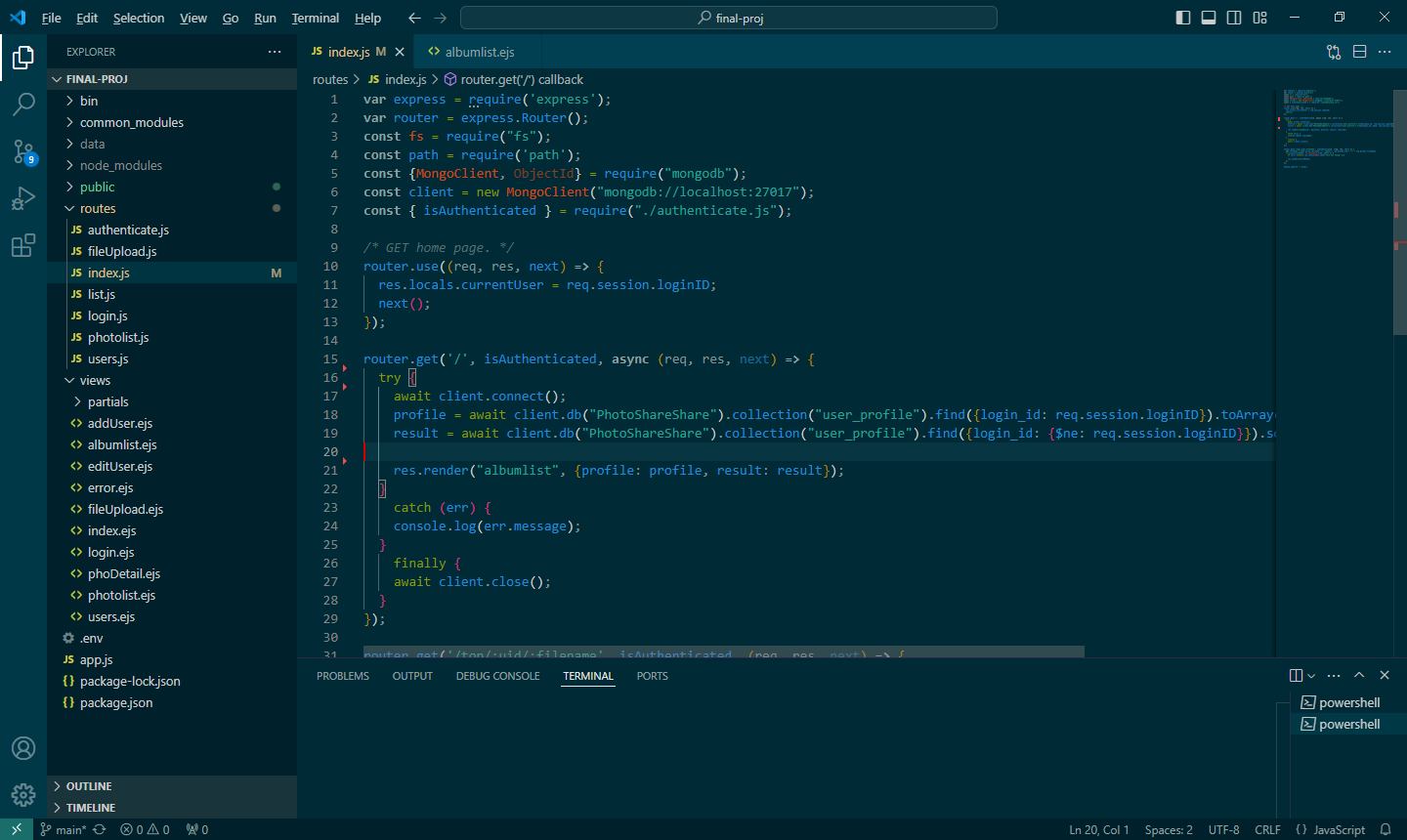


Pic11

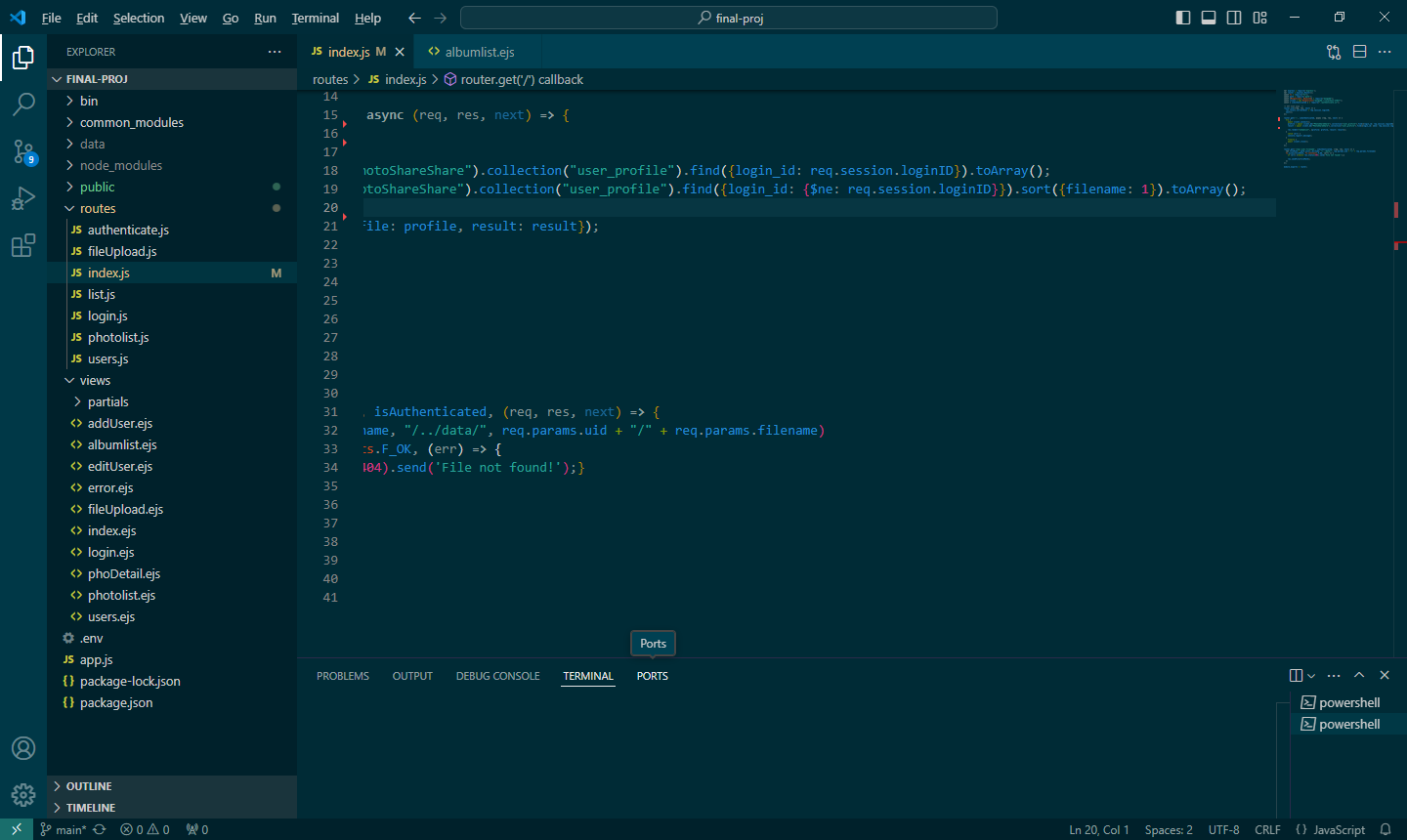


Pic12

* GET /
  + If users logged in, call out his/her display name, profile picture and photo list
  + (Pic13.1 & Pic13.2)
* GET /top/
  + Call out the users’ data from our data
  + Send out “file not found!” if can’t find any data
  + (Pic14)



Pic13.1



Pic13.2

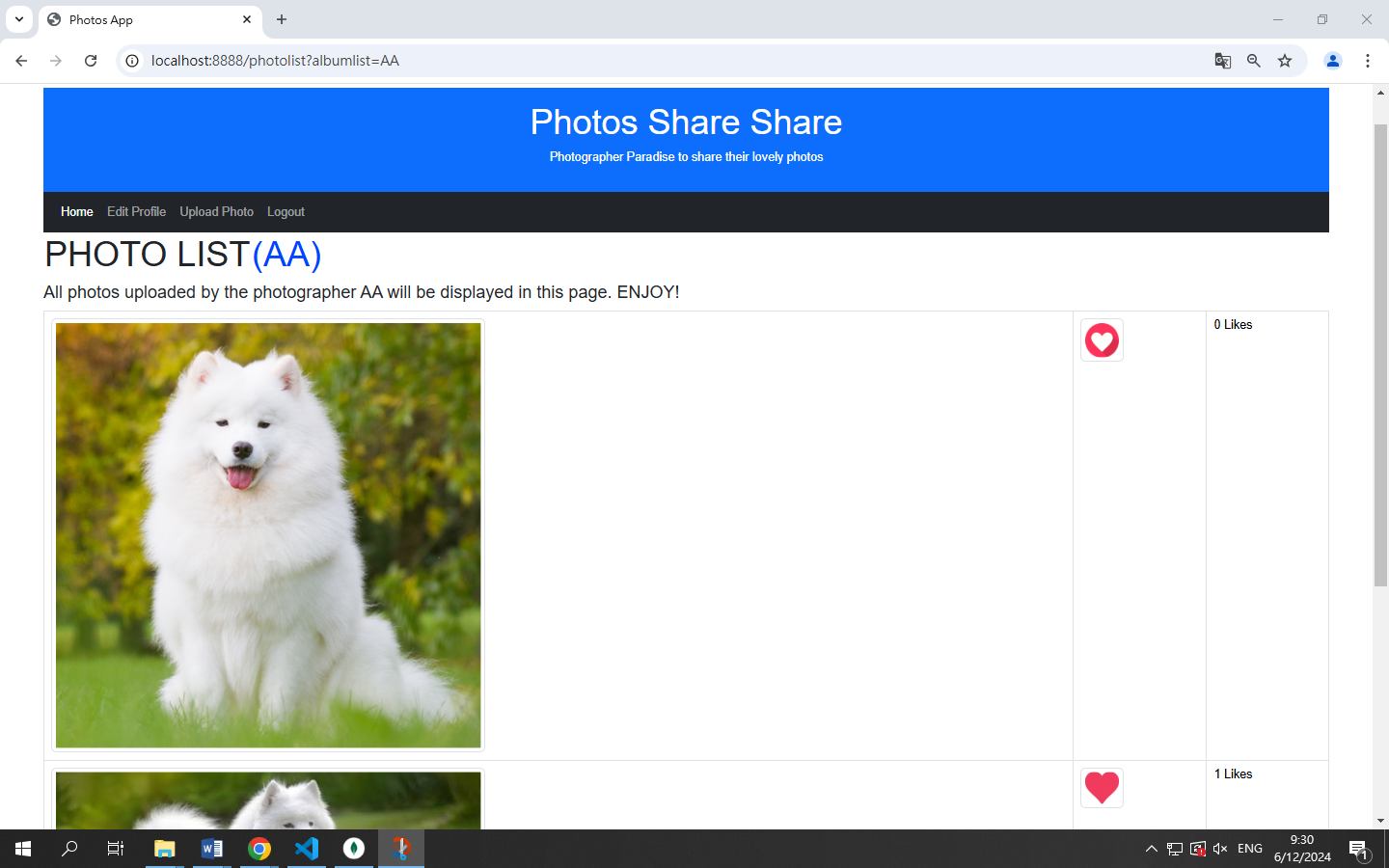


Pic14

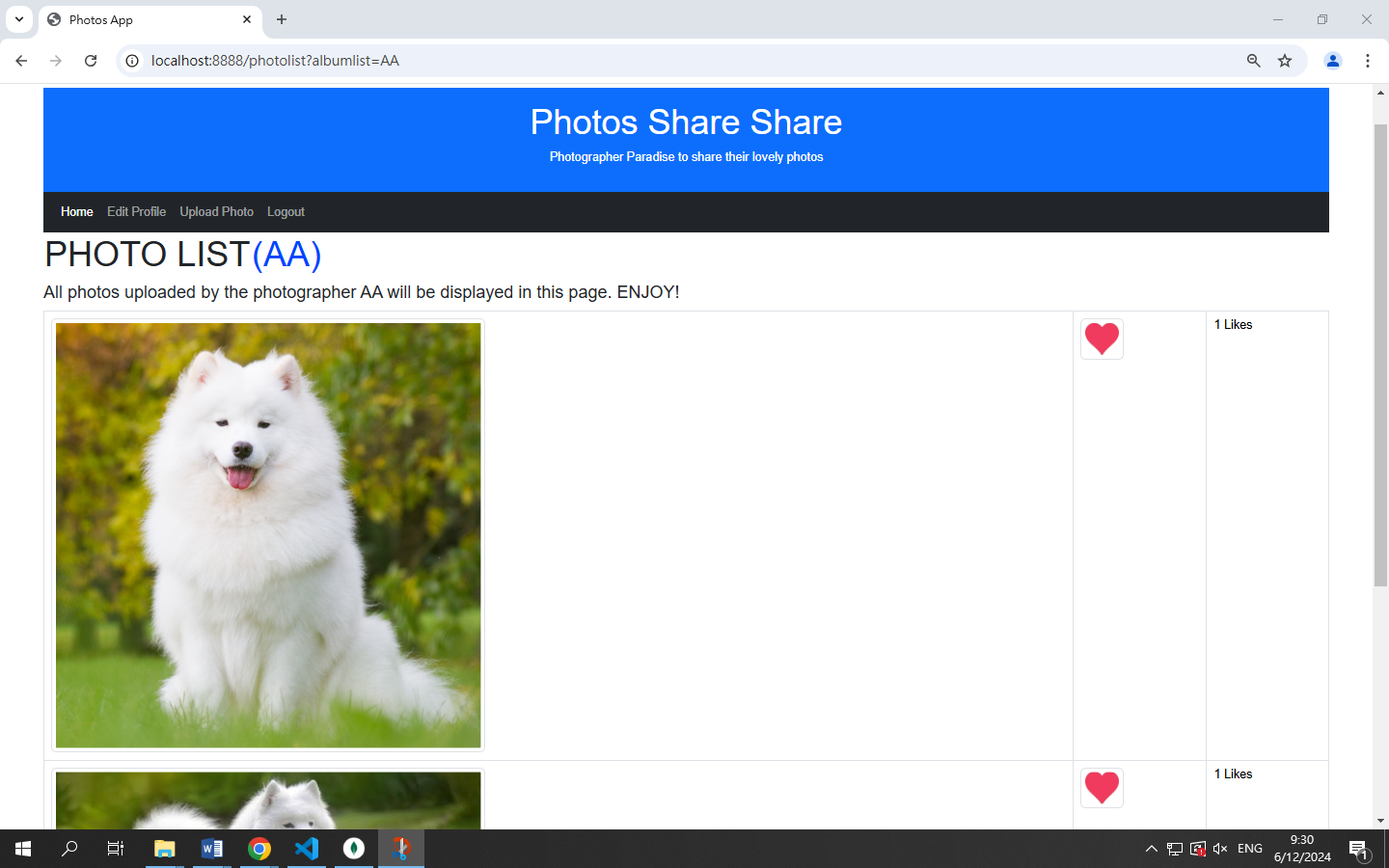
* + 1. **Photolist Page**

The photos uploaded by the photographer will be presented under their album in the page. All registered users can present their favors over the photos by clicking the button of empty heart (like function) and the button of filled heart (the unlike function(reversal)). The number of like count is displayed next to the heart button. It increases the interaction between the users.

Before clicking the button of “empty heart”, the number of like count is zero likes. Each user can present their like once on each photo by clicking the button.



After clicking the button of “empty heart”, the button will turn into “pink heart” and the like count is collected. The number of like count shown asides will be updated accordingly. The reversal of like will be the same.



Photolist API

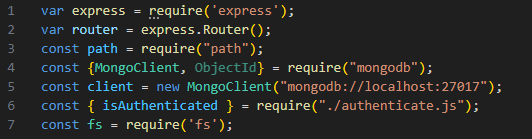
In the page of “PHOTOLIST”, the photos uploaded by the photographer (registered user) will be presented. All registered users can present their favors over the photos by clicking the “like” function (clicking the heart button, which will turn into pink heart shape after liked) and the “unlike” function (clicking pink heart shape, which will turn back to the original heart button after unliked. In MongoDB system, the numbers of “likes” will be counted and collected. Back to “PHOTOLIST”, the summation of likes will be shown next to the photo.

For understanding the operation logic behind “PHOTOLIST”, “photolist.js” and “photolist.ejs” will be referred and explained.

Explanation of the Code

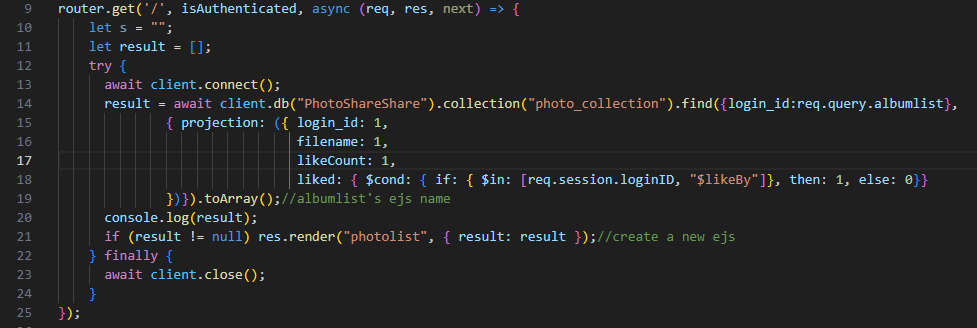
1. PHOTOLIST.JS

The given JavaScript code creates an Express router for handling a specific GET request. It involves connecting to a MongoDB database to retrieve photo information and rendering a result view based on the query results.



a. router.get (isAuthenicated)

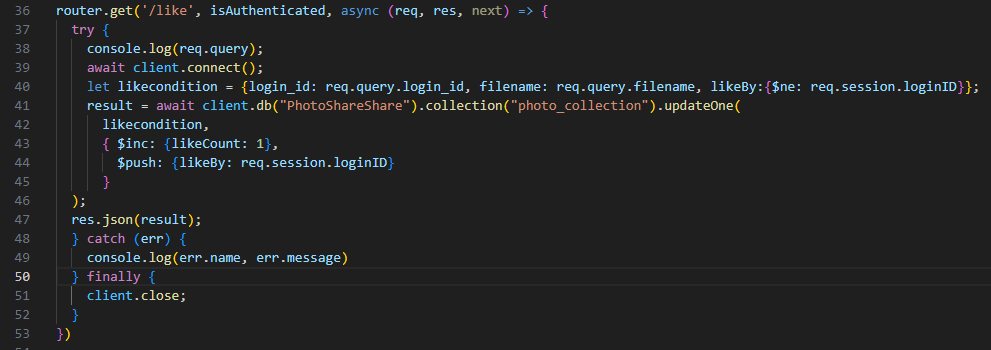
The code sets up an Express router that handles a GET request to the root path. When an authenticated user accesses this route, the server connects to a MongoDB database, retrieves photo information for the specified album, and renders a view with the retrieved data. The data includes details about the photos, such as their filenames, like counts, and whether the current user has liked them. Once the user is authenticated, the system connects to a MongoDB database. This connection enables efficient data management and retrieval.



b. router.ger(‘/like, isAuthenicated)

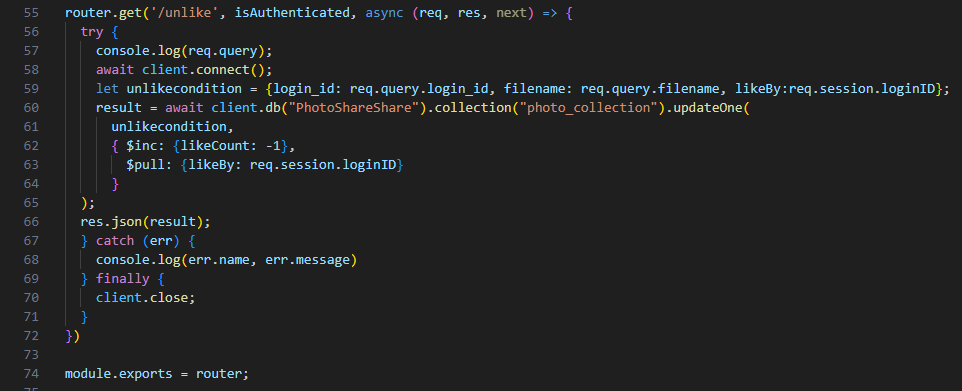
Within this database, a specific condition is defined to check if the photo has already been liked by the current user. This check prevents users from liking the same photo multiple times. Upon verifying that the user has not previously liked the photo, the system updates the "like" count for the specified photo. It adds the current user to the list of users who have liked the photo. This update operation is crucial for maintaining accurate engagement metrics for each photo.

After updating the "like" count, the system sends the result of the update operation as a JSON response. This comprehensive approach ensures that the process of liking a photo is handled securely, efficiently, and accurately, providing a seamless experience for the user.



c. router.get(‘/unlike, isAuthenicated)

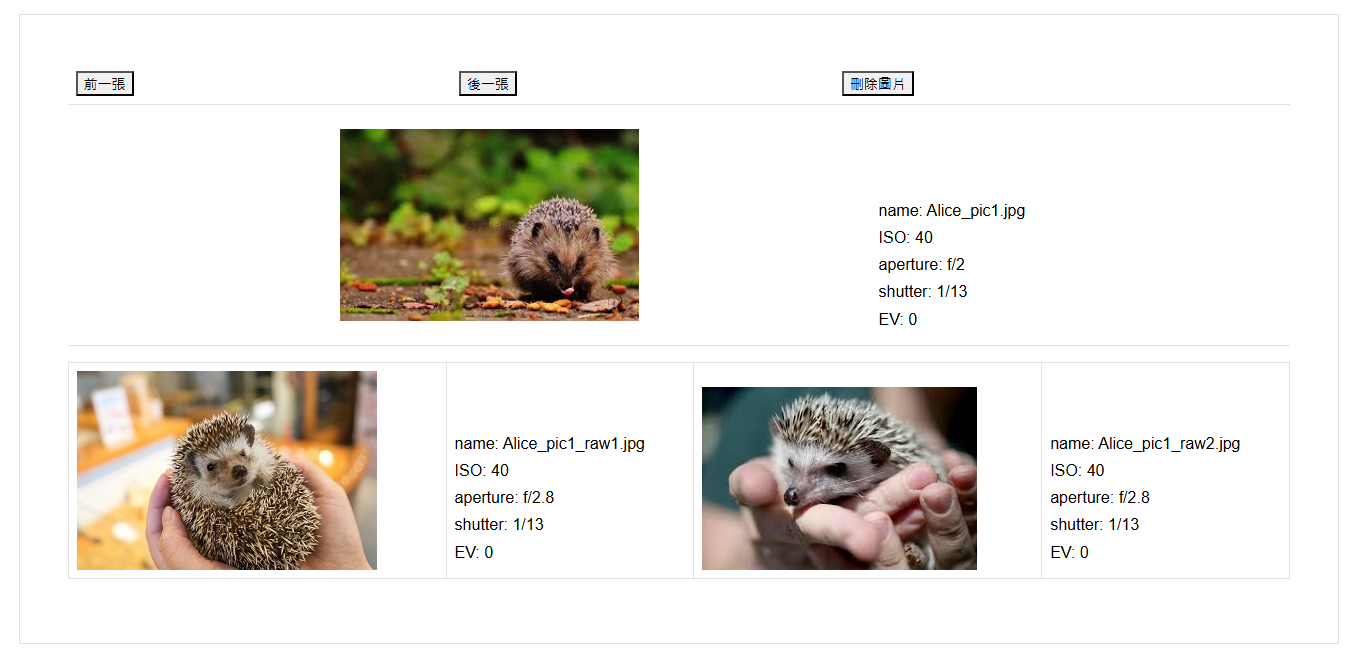
The code involves defining a condition to check if the photo has already been liked by the current user. This condition is crucial to ensure that users can only unlike photos they have previously liked. If the photo has been liked by the current user, the system updates the "like" count by decrementing it. Additionally, it removes the current user from the list of users who have liked the photo. This update operation is essential to maintain accurate engagement metrics for each photo. Afterward, the system sends the result of the operation as a JSON response.



2. PHOTOLIST.EJS

The code provides a dynamic photo listing page with functionalities for like and unlike photos. It ensures that the user's actions are immediately reflected on the page by updating the like/unlike buttons and the like count. This setup enhances user interaction and provides real-time feedback on their actions. The use of JavaScript functions ensures a seamless user experience by handling the like and unlike interactions asynchronously, without requiring a page reload.

* + 1. **Photo Details**



This is a photo details page.

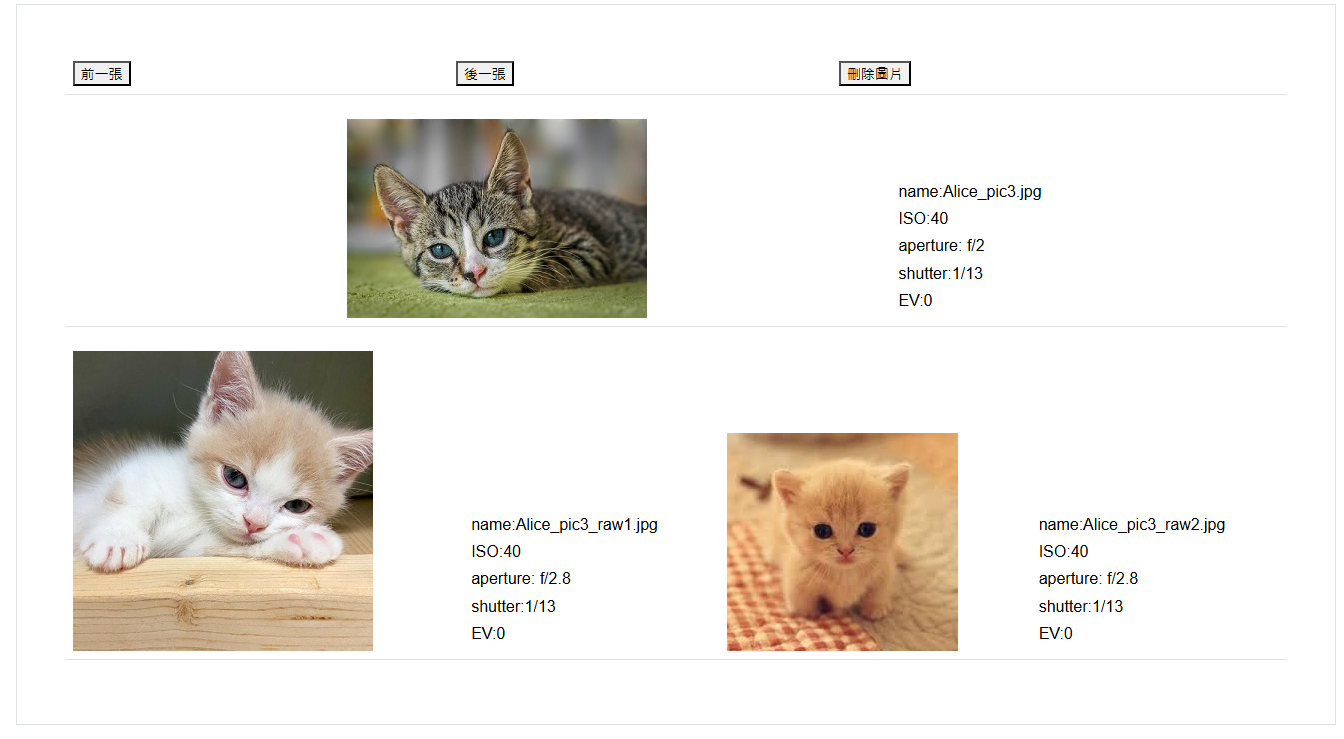
This page included three buttons(" previous picture"," next picture"," delete picture") , one main picture ,two raw pictures and some parameters for pictures.

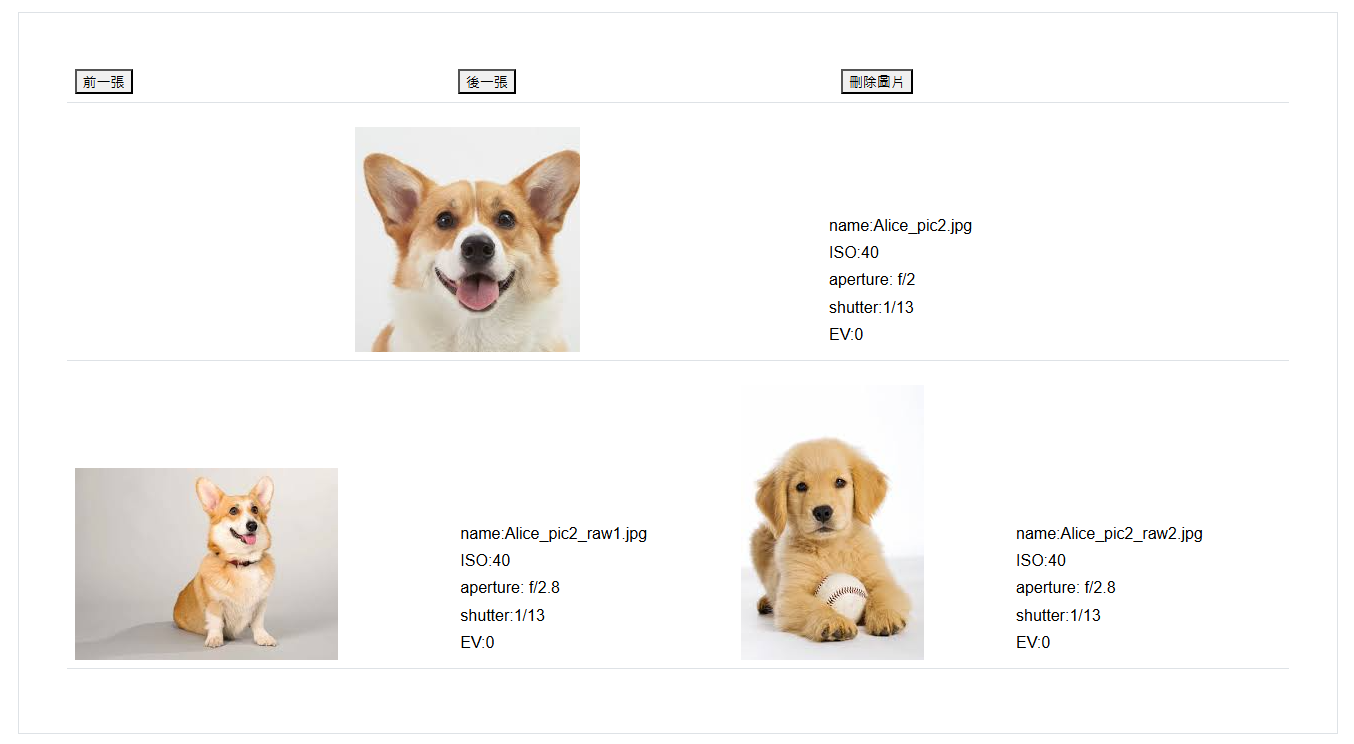
Three buttons are displayed at the top of the page and the main picture and some parameters of the main picture, and the two raw pictures and some parameters of the raw picture below the main picture.

Click " previous picture", It will display the previous picture. If this picture is the first picture, "Already the first picture" will pop up at the page top.

Click " next picture", It will display the next picture. If this picture is the last picture, "Already the last picture" will pop up at the page top.

Click "Delete Picture" , It will delete this picture and the two raw pictures and parameters under this picture. Only pictures uploaded by yourself can see this button . In this way, users who log in to the system can delete their own pictures, but not other people's pictures.





Get list/photodetail: Retrieve a single photo details with its raw photos from photo list by ID.

Get list/delephoto: Retrieve a single photo details with its raw photos from photo list by ID.

1. **APIs and Routes**

RESTful API Endpoints and Route Handlers

Introduction

This document outlines the RESTful API endpoints and their corresponding route handlers about photo upload module. The file contains routes for uploading and retrieving files, and it utilizes MongoDB for storing file-related data.

API Endpoints

**GET /fileUpload/**: This endpoint renders the file upload page. It requires the user to be authenticated.

**GET /fileUpload/:filename**: This endpoint retrieves a specific file based on the filename provided in the URL. The file is sent back to the client if it exists; otherwise, a 404 error is returned.

**POST /fileUpload/different**: This endpoint handles the upload of a main file along with two raw files. Each file is renamed and saved in a directory named after the user's login ID. The file-related data is then inserted into the MongoDB collection photo\_collection. The original filename is converted to UTF-8 and a unique filename is generated to avoid conflicts.

Route Handlers

Each API endpoint is associated with a route handler that processes incoming requests and sends responses accordingly. The route handlers perform various tasks such as authentication, handling multipart/form-data requests...

The isAuthenticated middleware is used to ensure that certain routes can only be accessed by authenticated users.

The multer middleware is used for handling multipart/form-data, which is primarily used for uploading files.

Example Usage

To use the API endpoints, you can make HTTP requests to the corresponding URLs using a tool like Postman or cURL. For example, to retrieve a specific file, you can make a GET request to /fileUpload/filename.jpg.

Additional Notes

The application's configuration, including the database connection string and file upload settings, is defined in the config.js file.

The helper object contains utility functions that are used in the route handlers, such as generating unique filenames and normalizing file paths.

The fileUpload.ejs file is the view template that is rendered when the file upload page is requested.

* GET /login
  + Present log-in page to front-end.
  + If URL carries unsuccessful log-in status of last log-in, corresponding error message will be embedded in the page.
* POST /login
  + Retrieve corresponding record from database with login-ID as parameter
  + If found, check if input password matches with database record
  + If password match, direct to system main page. Also, generate a session variable to carry login-ID
  + Otherwise, show log-in page again. URL is formulated to carry unsuccessful log-in reason.
* GET /login/logout
  + Clean-up login-ID session variable
  + Direct to log-in page
* GET /user/addUser
  + Present add new user page to front-end
* POST /user/addUser
  + Generate hash value for input password
  + Create a user record in database, per user inputs
  + Create an album folder under /data
  + If profile picture is provided, store into the user’s album folder
* GET /user/editUser
  + Retrieve user record from database
  + Present edit user page to front-end, with retrieved data as default value of form fields (except password field)
* POST /user/editUser
  + Generate hash value for input password
  + Update user record per user inputs
  + If new profile picture provided, remove old image file (if any) and store new image file into the user’s album folder.
  + Direct to main page
* GET /user/delUser
  + Remove all photo collection record(s) from database
  + Remove user profile record from database
  + Remove user’s album folder

**7 Security Control**

To prevent application from unauthorized or unauthenticated use, certain security measurements are enforced.

* User password is hashed before stored into database. It prevents anyone to spy on someone password by directing retrieving data from database.
* As /public folder is opened for anyone to access. If image files are kept under this folder, anyone can by-pass the application to access those files directly. To prevent this, A /data folder separated from /public folder is used for image files storage. Outsiders has no access rights to this folder. When application accesses those files, API is used to re-route access path to /data folder.