MACAU UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Information Technology

Thesis for Degree of Bachelor of Science

Title: Full-stack development of a web application for blogs

Student Name : ZHU ZHE HAO

Student No. : 1709853G-I011-0069

Supervisor : LIANG ZHI YAO

May, 2021

澳門科技大學

資訊科技學院

理學學士學位畢業論文

論文題目: 一個博客網頁的全棧開發

姓　　名: 朱哲昊

學　　號: 1709853G-I011-0069

指導老師: 梁之遙

2021 年 5 月

Abstract

Due to the fact that there are many blogs application in the web. But some powerful function blogs web app, for example Github and Stack overflow, they are too difficult for program beginner to use them publish and ask question. Some easy blogs but I don't like their colorful ui interface. So that I want to develop an easy use and simple style blog web application

The paper provides the all logic of my program and most of the technology I used in this development. Including signin signup part, blog part.

Main technology: Vue, html, css, javascript, python, flask, axios

Main tool: pycharm, vscode, postman, Typora

Keywords: blog, vue, flask

摘要

現在網絡上有非常多的代碼類型博客網頁應用，但是擁有強大功能的代碼網站，例如 github，stackoverflow 這類網站應用，對於代碼初學者來說進行發表自己的代碼和尋求幫助並不是非常的友好。而一些簡單的網站的 ui 交互頁面太過於絢麗。所以我想開發一款可以擁有基礎功能，並且 ui 界面相對簡約的博客網站

論文將會將我程序所有的設計邏輯，理念還有大部分所使用的到的技術列入其中，並且解析。程序主要包含兩塊：登錄模塊，博客模塊

主要的技術：vue，html，css，javascript，python，flask，axios

主要的工具：pycharm，vscode，postman，Typora

關鍵詞：博客，vue，flask

Table of Contents

目錄

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Background and motivation

Due to the fact that there are many blogs application in the web. But some powerful function blogs web app, for example Github and Stackoverflow, they are too difficult for program beginner to use them publish and ask question. Some easy blogs but I don't like them colorful ui interface. So that I want to develop an easy use and simple style blog web application

1.2 Thesis organization

Chapter 2 design concept

Chapter 3 technology

Chapter 4 test and question

Chapter 5 conclusion and future works

Chapter 2 Design concept

This is a part of my design and thinking. It shows what I need.

2.1 UML picture

2.1.1 User UML

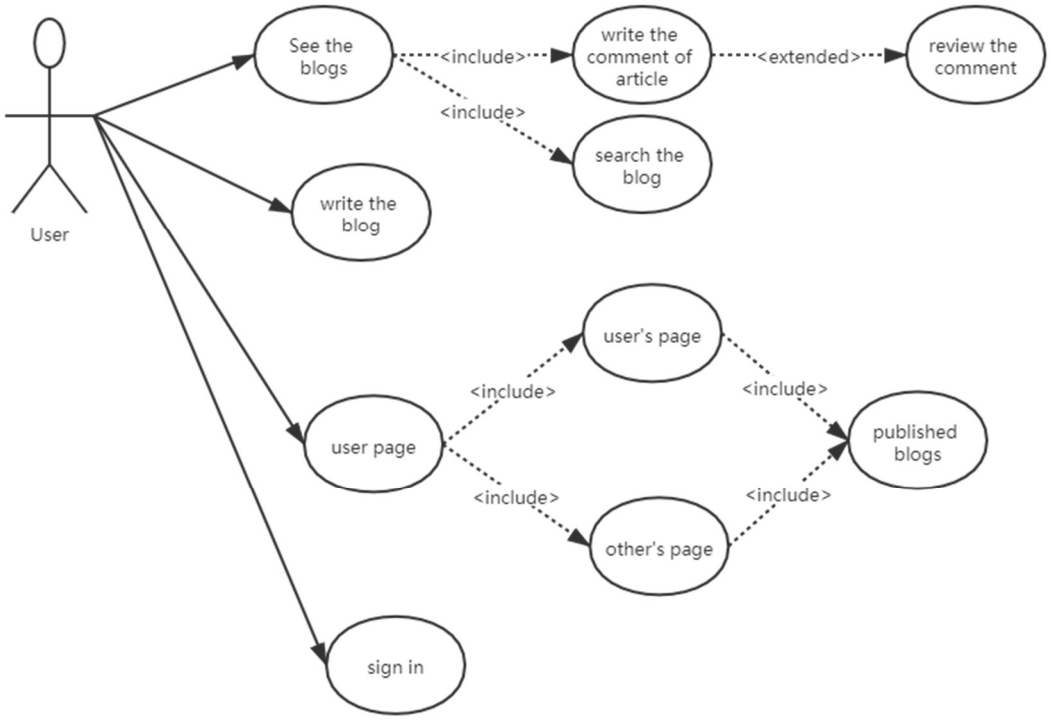


Figure 1 Uml for user

This picture show that the main function of users. User can sign in. After it sign in, it can use write blog function, comment the blog and review the comment.

User can find himself page, and find he published blog

2.1.2 Visitor UML

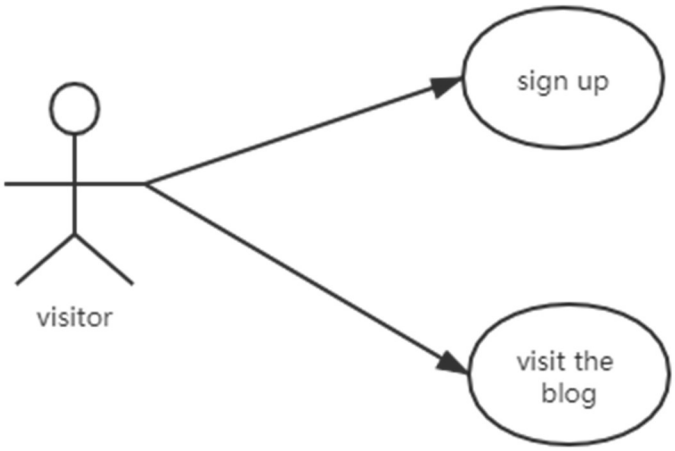


Figure 2 Uml for visitor

Visitor only can sign up and visit the blog, cannot review, comment and publish article.

2.2 UI concept

This blog web application obeys the Material Design.

2.2.1 Material Design content

Material is a design system created by Google to help teams build high-quality digital experiences for Android, IOS, Flutter, and the web. It like use the card, paper, color compared to make up the web page.

Material-Design official link: https://material.io/

Material is symbol: visual cues must be based on reality.

Eye catching, image, planning: the basic design principles (font, grid, space, scale, color and image use principles) must guide the visual effect.

Meaningful movement: moving objects or actions should not interfere with the user experience, but ensure the continuity of the user experience.

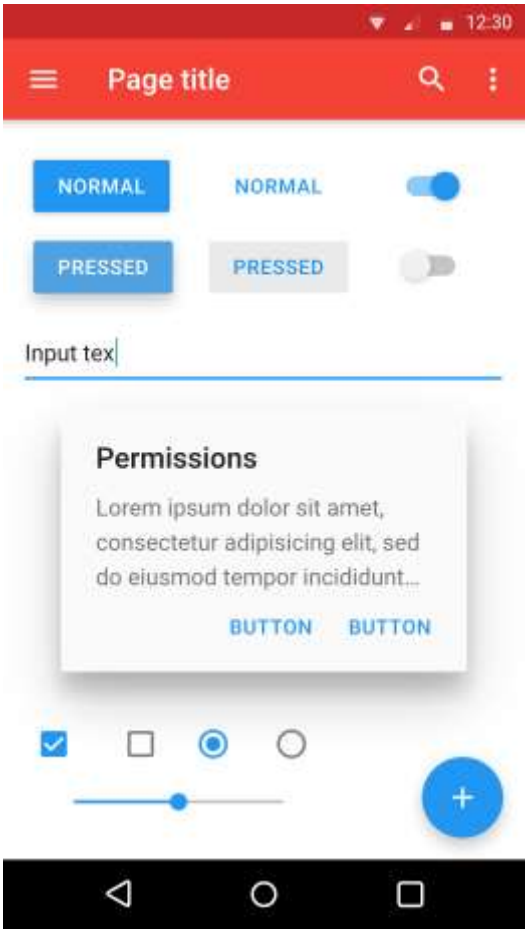


Figure 3 Material Design example

2.2.2 UI framework choose

I choose the Vuetify UI framework consists of my web application, this UI framework is a Material Design Framework. It is a vue ui library for vue 2.0,

Vuetify UI framework link: https://vuetifyjs.com/en/

Chapter3 Technology

This part will introduce the technology that I used in this web application. There are three parts of Technology: Front-end, Back-end, Front-Back-connection. All development environment is Windows 10

3.1 Front-End

This part will show that I used technology and logic in front-end. It contain 6 parts:

1. Main technology tools introduction
2. How to install the environment
3. Page design
4. Markdown
5. Login logic

3.1.1 Main technology tools

I choose the vue framework to make up my front-end.

Vue is a javascript web framework for single page web application. Its official web is: https://vuejs.org/. Due to the fact that vue 3.0 is not stable, I choose the vue 2.0 version. Vue is a progressive framework for building user interface. It unlike other framework, vue only focus on the view layer. By using it and other modern tooling, I can make a single page easier.

Vue-cli is a vue standard develop tool for vue.js. I choose the vue-cli 4.5.11 version. Its official web is: https://cli.vuejs.org/ . It can help user easier to create vue project. It has a graphic UI to manage the plugin and pack.

Vuex is a vue data management tool. I choose the vuex version 3.6.2. Its official web is: https://vuex.vuejs.org/ . Vuex is a data control in vue project, I can control the global variable in this tool

Vue-router is a router management tool for vue.js. I choose the vue-router version 3.5.1. Its official web is: https://router.vuejs.org/ . It will give user more convenient to manage the router in user design for app.

Node.js and npm are packs management tools. I used the Node.js 14.15.4 version

and npm 6.14.11 version. They official web are https://nodejs.org/en/ and https://www.npmjs.com/

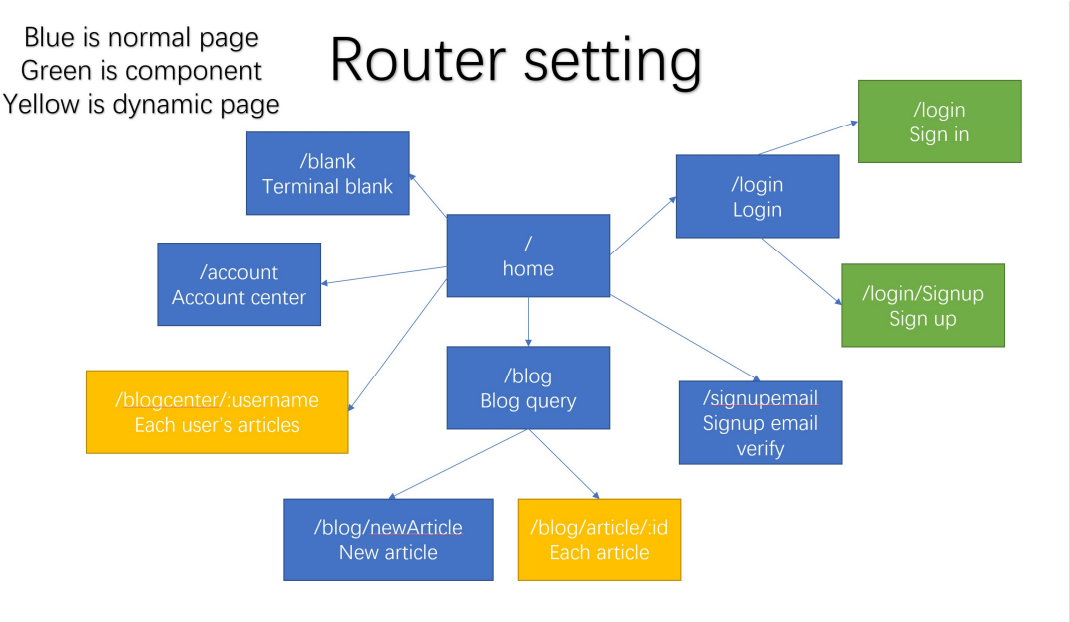
3.1.2 Install and develop environment

After installing all the things that I listed above. I choose the Windows10 as development system, Visual studio code as development software.

3.1.3 Page design

Because the blog web application have two main function: Login and Blog, so need design some page to show them.

Router design: home is the root page, it can go to 4 part, blog part, login part, author part, account centre part.



3.1.3.1: Login page

Main page: It is made up of two components for changing. Sign up component and Sign in component. Sign up component need input four data to upload to Back-End to operate, Username, Email, Password, repeated password. Sign in component need input 2 data to upload to Back-End to operate, Email, Password.

Source code show:

```
<template>
  <div id="Login" style="width: 500px; margin: 50px auto">
    <v-card color="basil">
        <v-tabs grow>
            <v-tab to="/login">SignIn</v-tab>
            <v-tab to="/login/SignUp">SignUp</v-tab>
        </v-tabs>
        <router-view></router-view>
    </v-card>
  </div>
</template>
```

This is the page show, need import two component in router-view setting

```
    path: '/login',
    name: 'Login',
    component: () => import('../views/Login.vue'),
    children:[
      {
        path:'',
        component: () => import('../components/SignIn.vue')
      },
      {
        path:'SignUp',
        component: () => import('../components/SignUp.vue')
      },
    ]
```

Email verify rule: We must input correct email format for verify. I used the regular expression to realize it

This regular expression means that: a correct email form. It must has xxxx@xx.xxx from

/^([A-Za-z0-9_\-\.\u4e00-\u9fa5])+\@([A-Za-z0-9_\-\.])+\.([A-Za-z]{2,8})$/g

Due to the fact that these few years some Chinese email is valid, so I add Chinese verify in regular expression \u4e00-\u9fa5

Password verify rule: For the account safe, we must demand the password complexing:

This regular expression means that the password must have 8-16 bits of an upper case letter, a lower case letter, a number, and a special symbol

/^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[^\w\s]).{8,16}$/g

Email verifying code page:

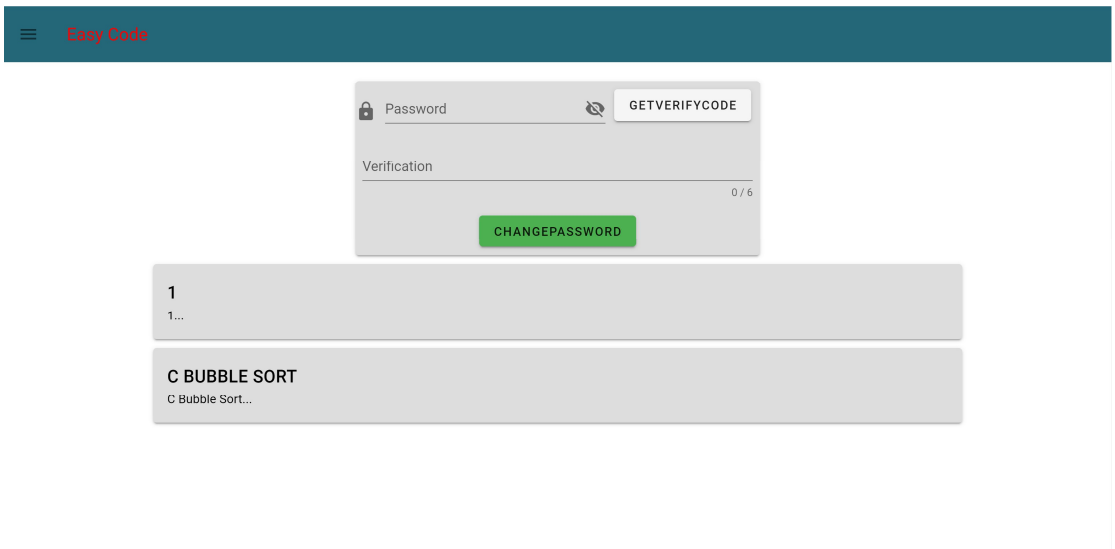
This page will show a upload verifying code input, user should input 6 characters.

3.3.1.2 User center page

This page have two parts, one part is user's, one part is authors'. User can find this author all blogs.

User center only can after user login to visit.

```
if (this.$store.state.quanjulogin == 'true') {
  this.$router.push(item.to);
}
else{
  alert("please login")
  this.$router.push('/login')
}
```



3.3.1.2 Blog page

Blog page can divide to 3 parts: newblog, blogquery and blogdetail.

Blogquery page: this page will show blogs from Back-End. User can choose type and search the blogs' detail key of title or author

```
    filteredBlogs: function () {
      return this.items.filter((item) => {
        console.log(this.listtype);
        if (this.listtype === "all") {
          return (
            item.author.match(this.search) || item.title.match(this.search)
          );
        } else if (item.type === this.listtype) {
          return (
            item.author.match(this.search) || item.title.match(this.search)
          );
        } else {
          return null;
        }
      });
    },
```

This part is a style render. It gives the blog query title and introduction detailed style

```
Vue.filter("to-uppercase",function(value){
  return value.toUpperCase()
})

Vue.filter("snippet",function(value){
  return value.slice(0,100) + "..."
})
```

```
      <v-list-item-group
        v-model="selectedItem"
        color="primary"
      >
        <v-list-item
        dense
          v-for="(item, i) in item1s"
          :key="i"
```

```
        @click="listtype=item.type"
      >
        <v-list-item-icon>
          <v-icon v-text="item.icon"></v-icon>
        </v-list-item-icon>
        <v-list-item-content>
          <v-list-item-title v-text="item.text"></v-list-item-title>
        </v-list-item-content>
      </v-list-item>
    </v-list-item-group>
```

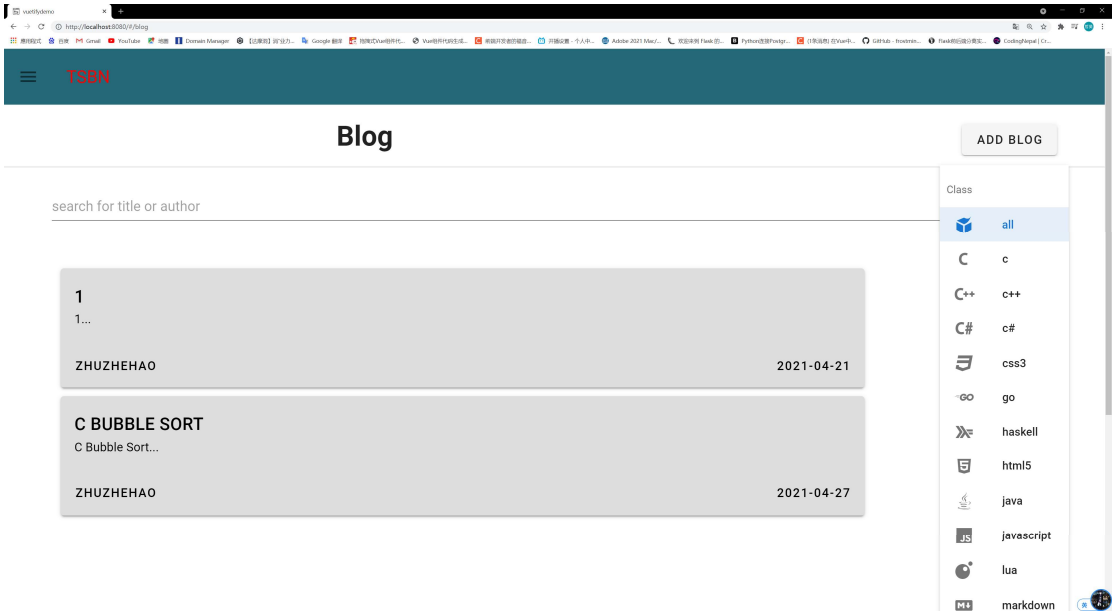


Figure 4 blog query

First we should create a list to show the class of code. Because the material icon only has parts of codes icon, so I create an all class and some detailed class.

```
      { text: "all", icon: "mdi-semantic-web", type:"all" },
      { text: "c", icon: "mdi-language-c", type:"c" },
      { text: "c++", icon: "mdi-language-cpp", type:"cpp" },
      { text: "c#", icon: "mdi-language-csharp", type:"csharp"  },
      { text: "css3", icon: "mdi-language-css3", type:"css3" },
      { text: "go", icon: "mdi-language-go", type:"go" },
      { text: "haskell", icon: "mdi-language-haskell", type:"haskell" },
      { text: "html5", icon: "mdi-language-html5", type:"html5" },
      { text: "java", icon: "mdi-language-java", type:"java" },
```

```
        { text: "javascript", icon: "mdi-language-javascript", type:"javascript" },
        { text: "lua", icon: "mdi-language-lua", type:"lua" },
        { text: "markdown", icon: "mdi-language-markdown", type:"markdown" },
        { text: "php", icon: "mdi-language-php", type:"php" },
        { text: "python", icon: "mdi-language-python", type:"python" },
        { text: "ruby", icon: "mdi-language-ruby", type:"ruby" },
        { text: "rust", icon: "mdi-language-rust", type:"rust" },
        { text: "swift", icon: "mdi-language-typescript", type:"swift" },
        { text: "vue", icon: "mdi-vuejs", type:"vue" },
        { text: "jquery", icon: "mdi-jquery", type:"jquery" },
        { text: "nodejs", icon: "mdi-nodejs", type:"nodejs" },
```

Then I will make a cycle for blog to show the query from the Back-End

Then give them link to the author center or blogdetail, it should use the Dynamic Route Matching

```
    path:'/blog/article/:id',
    component: blogarticle
```

/:id means that should transform the variable data.

```
    tiaozhuan: function (i) {
      this.$router.push({ path: "/blog/article/" + i, query: { id: i } });
    },
```

In the javascript, we can use the str + variable to a new str. We need to transform some data to next page to post it get some data from Back-end

Blogdetail page:
This page will show the blog main things, title, introduction, content, comment, review

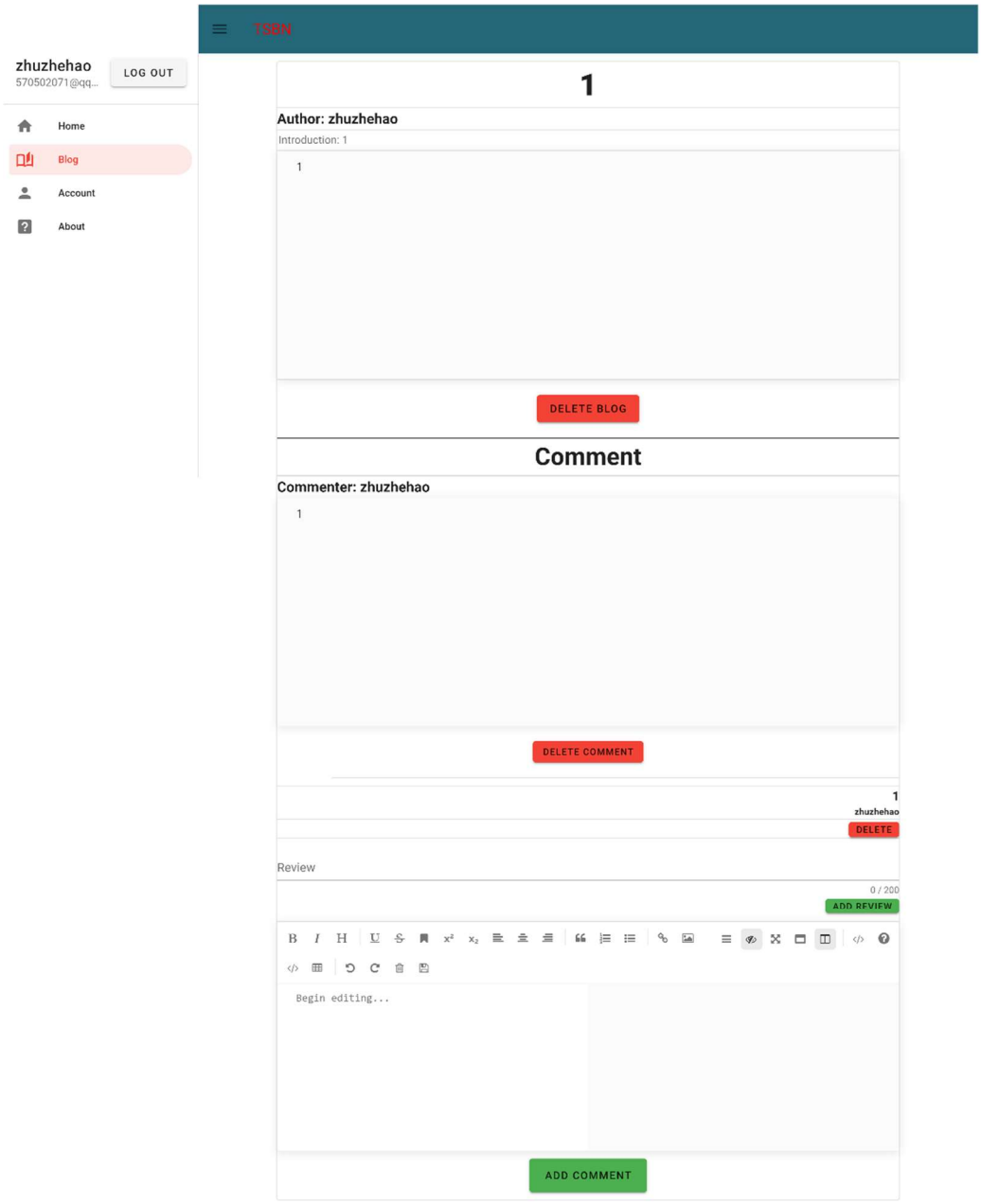


Figure 5 long screen shot for blogdetail

This page need get the query data from blogquery

```
this.$route.query.id
```

when click button, it will use corresponding function:
Add comment(), Add review(): need user login can operate

Delete comment(), Delete review(), Delete blog(): need user login and username equal to author will show

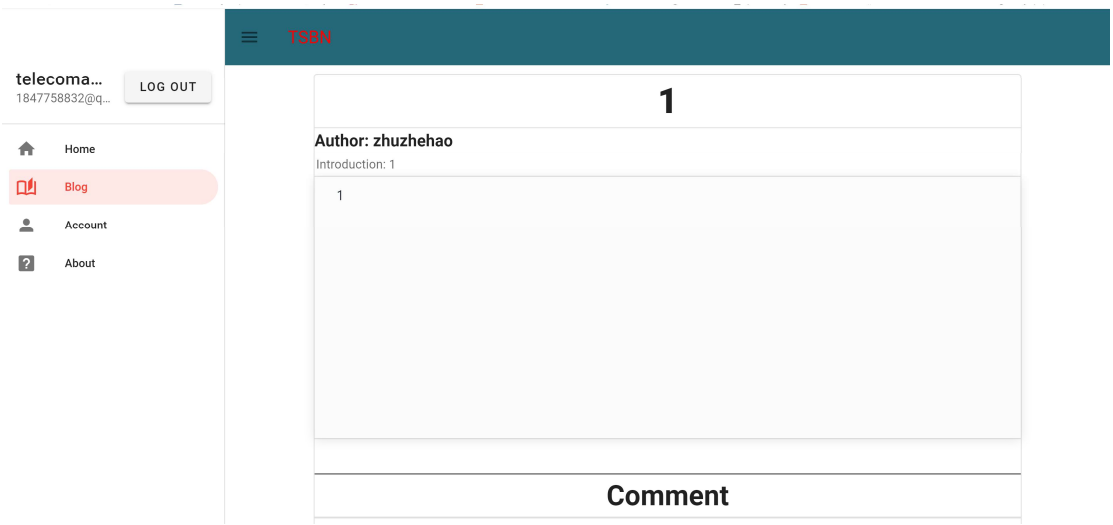


Figure 6 username not equal to author

Addblog page:

This page will give four part for user to input: title, introduction, type and content. When user click the upload button, it will upload to Back-End save.

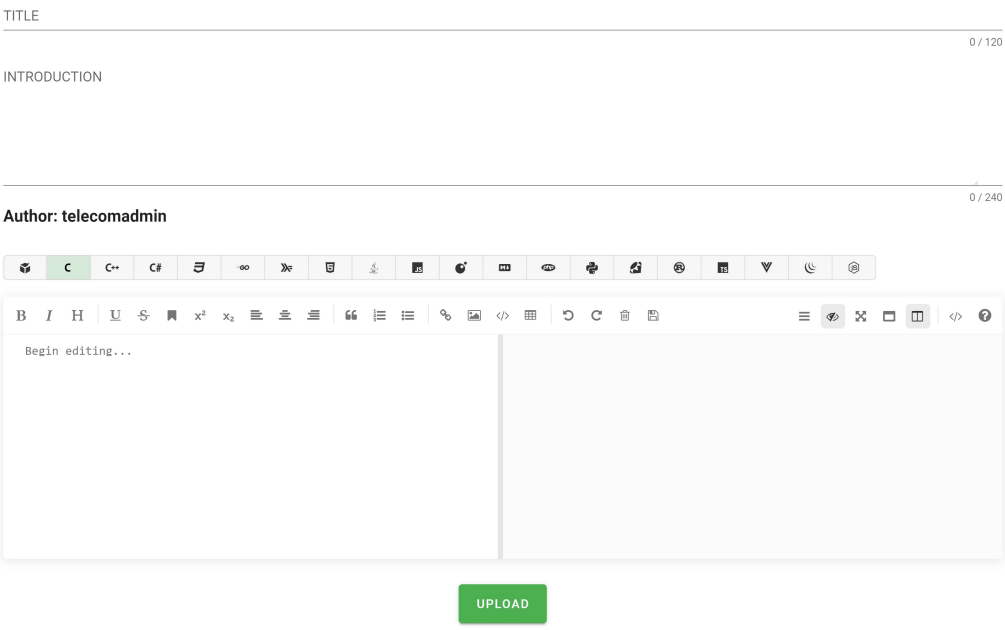


Figure 7 Add new blog

3.1.4: Markdown

This is main part of all function: how to write blog and show blog in web application.
Markdown is a light markup language. It can write beautiful blog in some easy code. I choose a editor pack in vue to use, Mavon-editor. Its official github is:
https://github.com/hinesboy/mavonEditor/blob/master/README-EN.md

Install mavon:
npm install mavon-editor –save

Import mavon-editor:
In main.js

```
import mavonEditor from 'mavon-editor'
import 'mavon-editor/dist/css/index.css'
Vue.use(mavonEditor)
```

Use the mavon-editor in write

```
<mavon-editor
  ref="md"
  style="z-index: 0"
  :value="value1"
  language="en"
  v-model="value"
  :ishljs="true"
  @save="save"
  @imgAdd="$imgAdd"
/>
```

Use the mavon-edtor in read: should set default-open=”preview” and toolbarsFlag=”false”

```
<mavon-editor
  :subfield="false"
  :toolbarsFlag="false"
  language="en"
  default-open="preview"
  v-model="comment.content"
/>
```

3.1.5: Login logic:

Nowadays, more and more application can save the login state. In order to realize it, I use the vuex and cookie.

Vue-cookie version 1.1.4: https://github.com/alfhen/vue-cookie#readme

```
import Vue from 'vue'
import Vuex from 'vuex'
import cookie from 'vue-cookie'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    quanjuemail: cookie.get("email"),
    quanjupassword: cookie.get("password"),
    quanjulogin: cookie.get("islogin"),
    quanjuusername: cookie.get("username")
  },
  mutations: {
    quanjulogin(state,loginarray){
      state.quanjuemail = loginarray[0],
      state.quanjupassword = loginarray[1],
      state.quanjulogin = loginarray[2],
      state.quanjuusername = loginarray[3],
      cookie.set("email",state.quanjuemail,3),
      cookie.set("password",state.quanjupassword,3),
      cookie.set("islogin",state.quanjulogin,3),
      cookie.set("username",state.quanjuusername,3)
    }
  },
  actions: {
  },
  modules: {
  }
})
```

State is a space to save the global variable data. It equal to cookie.get() means that User can auto login if he has logined 3 days.
The four global variables are username, password, email and login state.

Mutation is a submit method, user should add method in this part, then register in needed component.

Cookie.set("email", state.quanjuemail,3)
Set a email str into cookie, state.quanjuemail is data, 3 is 3day.
So I set a email str cookie is state.quanjuemail 3day will delete

Mutation is a change data function: when we login or log out, should use this function to change the cookie and vuex

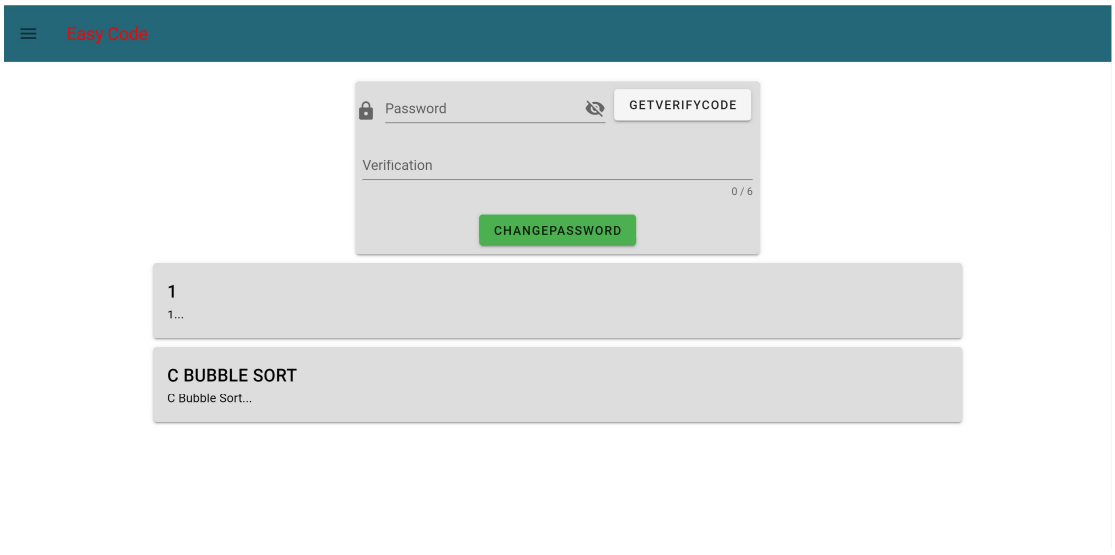
```
import {mapMutations} from 'vuex'
              this.quanjulogin(loginarray)
    ...mapMutations(['quanjulogin']),
```

3.1.6 User Center

This page have two parts, one part is user's, one part is authors'. User can find this author all blogs.

User center only can after user login to visit.

```
        if (this.$store.state.quanjulogin == 'true') {
          this.$router.push(item.to);
        }
        else{
          alert("please login")
          this.$router.push('/login')
        }
```



3.2 Back-End

This part will show that I used technology and logic in Back-End. It consist of 4 part:

1. Main technology introduction
2. Flask structure design
3. Database design
4. Connect between front-end and back-end

3.2.1 Main technology

I choose the flask to build my app Back-End. It is a python framework. Its official web site is: https://flask.palletsprojects.com/en/1.1.x/

I will list all python pack as follows:

alembic==1.1.0
aniso8601==7.0.0
astroid==2.4.2
Babel==2.3.4
backcall==0.1.0
bleach==1.5.0
blinker==1.4
certifi==2020.12.5
cffi==1.14.5
chardet==4.0.0
Click==7.0
colorama==0.4.4
DateTime==4.3
decorator==4.4.0
dominate==2.2.1
Flask==1.0.3
Flask-Bootstrap==3.3.7.1
Flask-Cors==3.0.8
Flask-HTTPAuth==3.3.0
Flask-Login==0.3.2
Flask-Mail==0.9.1
Flask-Migrate==2.5.2
Flask-Moment==0.5.1
flask-nav==0.5
Flask-PageDown==0.2.2
flask-redis==0.4.0
Flask-RESTful==0.3.7
Flask-Script==2.0.5
Flask-SQLAlchemy==2.4.0
Flask-Uploads==0.2.1
Flask-WTF==0.13.1
ForgeryPy==0.1
gevent==1.4.0
greenlet==0.4.15
gunicorn==19.9.0
html5lib==0.9999999
idna==2.10
importlib-metadata==1.7.0
ipython==7.6.1
ipython-genutils==0.2.0
isort==5.7.0
itsdangerous==1.1.0
jedi==0.14.1
Jinja2==2.10.1
jsonschema2md==0.1.1
lazy-object-proxy==1.4.3
livereload==2.4.1
Mako==1.1.0
Markdown==2.6.7
MarkupSafe==1.1.1
mccabe==0.6.1
mysql-connector-python==8.0.23
numpy==1.19.4
pandas==1.1.4
parso==0.5.1
passlib==1.7.1
pexpect==4.7.0
pickleshare==0.7.5
prompt-toolkit==2.0.9
psycopg2==2.8.6
ptyprocess==0.6.0
pycparser==2.20
Pygments==2.4.2
pylint==2.6.0
PyMySQL==0.9.3
python-dateutil==2.8.0
python-editor==1.0.4
pytz==2019.1
redis==3.3.8
requests==2.25.1
six==1.12.0
SQLAlchemy==1.3.4
toml==0.10.2
tornado==4.4.2
traitlets==4.3.2
typed-ast==1.4.2
urllib3==1.26.3
visitor==0.1.3
wcwidth==0.1.7
Werkzeug==0.16.0
wrapt==1.12.1
WTForms==2.1
zipp==3.4.1
zope.interface==5.4.0

3.2.2 Flask structure

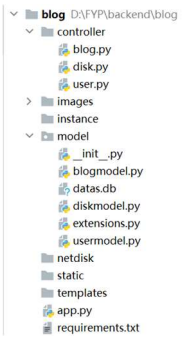


Figure 8 flask structure

In the main file app.py, I register the app, and give some default data.

```
import os
from flask import Flask
from controller.user import app_user
from controller.blog import app_blog
from controller.disk import app_disk
from flask_cors import *
from model.extensions import mail
def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY='dev',
    )
    mail.init_app(app)
    app.register_blueprint(app_user)
    app.register_blueprint(app_blog)
    CORS(app, supports_credentials=True)
    if test_config is None:
        app.config.from_pyfile('config.py', silent=True)
    else:
        app.config.from_mapping(test_config)
    try:
        os.makedirs(app.instance_path)
    except OSError:
        pass
    return app
```

```
    app.register_blueprint(app_user)
    app.register_blueprint(app_blog)
```

These 2 line codes are register blueprint for 2 parts to import them.

Blog.py and user.py are controller, they control all function to support Back-End run.

3.2.2.1 User.py

Register: register part need get user upload 3 data: username, email, password. The request is a post request.

```
@app_user.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    username = data.get('username')
    email = data.get('email')
    password = data.get('password')
```

then need search this username and email whether exist in database.

```
    searchusername = exist_user_username(username)
    searchemail = exist_user_email(email)
```

```
def exist_user_email(email):
    return len(db.execute(f"SELECT email FROM userinfo WHERE email='{email}'").fetchall(
)) > 0
def exist_user_username(username):
    return len(db.execute(f"SELECT username FROM userinfo WHERE username='{username}'").
fetchall()) > 0
```

In order to avoid some people use other email to register the account. I add a verify code in Back-End. When user upload email username and password. System will send an email to user. User need input the verify code that email sending to register success.

```
    if not searchemail and not searchusername:
        code_list = []
        for i in range(2):
            random_number = random.randint(0, 9)
            a = random.randint(65, 90)
            b = random.randint(97, 122)
            random_uppercase_letter = chr(a)
            random_lowercase_letter = chr(b)
```

```
            code_list.append(str(random_number))
            code_list.append(random_uppercase_letter)
            code_list.append(random_lowercase_letter)
        verification_code = ''.join(code_list)
        print(verification_code)
        msg = Message(subject="helloworld",
                      sender='3588034273@qq.com',
                      recipients=[f'{email}'])
        msg.body = f'{verification_code}'
        mail.send(msg)
        insert_verifycode(email, verification_code)
        print("success")
        return jsonify({"status": '1', "message": "please go to find verifycode from ema
il"})
    if searchusername:
        return jsonify({"status": '0', "message": "username has been used"})
    if searchemail:
        return jsonify({"status": '-1', "message": "email has been used"})
```

The sending email is a pack of flask is flask_mail, its official website is https://pythonhosted.org/Flask-Mail/

Now give the email how to set, for example qqemail:

Go to the setting to open the IMAP/SMTP, copy the code, then input it in mail_password

```
    app.config['MAIL_SERVER'] = 'smtp.qq.com'
    app.config['MAIL_PORT'] = 465
    app.config['MAIL_USE_SSL'] = True
    app.config['MAIL_USE_TLS'] = False
    app.config['MAIL_USERNAME'] = '3588034273@qq.com'
    app.config['MAIL_PASSWORD'] = 'wmxbvgdcdqiqchhj'
    app.config['MAIL_DEFAULT_SENDER'] = '3588034273@qq.com'
    mail.init_app(app)
```

Emailverify: This part will deal with the user upload verifycode and verify it. If correct, insert the user account into database

```
        searchverifycode = search_verifycode(email)
    if searchverifycode[0][0] == verifycode:
        if time.time() - float(searchverifycode[0][1]) <= 60 * 30:
            insert_user(username, email, password)
            delete_verifycode(email)
            return jsonify({"status": '1', "message": "register success"})
        else:
            return jsonify({"status": '-1', "message": "verifycode expired"})
    return jsonify({"status": '0', "message": "register error"})
```

```
def insert_verifycode(email, verifycode):
    if len(db.execute(f"SELECT email FROM emailverify WHERE email = '{email}'").fetchall
()) > 0:
        db.execute(f"DELETE FROM emailverify WHERE email = '{email}'")
    ticks = time.time()
    db.execute(f"INSERT INTO emailverify (email, verifycode, addtime) VALUES('{email}',
'{verifycode}', '{ticks}')")
    return 1
```

These function add a time check,
Ticks=time.time() create a python timestamp, when user upload the verify, will compute they difference, more than 60*30 seconds will error

3.2.2.2 Blog.py

This part main deal with blog things, including addblog, delete blog, add comment, delete comment, add review, delete review.

I will choose two important part detailed introduce in page.

Add parts: add blog function need Front-End upload six data: title, author, introduction, content, email, type

```
def insert_blog(title, author, introduction, content, email, type):
    print(content)
    tz = pytz.timezone('Asia/Shanghai')  # 东八区
    t = datetime.fromtimestamp(int(time.time()),
                               pytz.timezone('Asia/Shanghai')).strftime('%Y-%m-%d')
    print(t)
    title=title.replace("'","''")
    author = author.replace("'", "''")
    introduction = introduction.replace("'", "''")
    content = content.replace("'", "''")
    print(content)
    db.execute(f"INSERT INTO bloginfo(title, author, introduction, content, email, type,
 date) "
               f"VALUES('{title}', '{author}', '{introduction}', '{content}', '{email}',
 '{type}', '{t}')")
    return 1
```

Due to the fact that the Sqlite3 has a complex question, it cannot insert ‘ . Because most of codes have ‘ , I add a function replace, replace ‘ => ‘’ . When the data add into database, it will reduce 1 ‘ , so the data in database is ‘.

Blogdetail function:

```
def query_detail(id):
    datas = {}
    datas['blog'] = db.execute(f"SELECT id, title, author, introduction, content, email
"
                               f"FROM bloginfo "
                               f"WHERE id={id}").fetchone()
    blogcomments = db.execute(f"SELECT id, author, content "
                              f"FROM blogcommentinfo "
                              f"WHERE blog_id={datas['blog'][0]}").fetchall()
    datas['comment'] = []
    for blogcomment in blogcomments:
```

```
        datas['comment'].append(
            {"comment_id": blogcomment[0], "author": blogcomment[1], "content": blogcomm
ent[2], "review": []})
        commentcomments = db.execute(f"SELECT id, author, content "
                                     f"FROM commentcommentinfo "
                                     f"WHERE comment_id={blogcomment[0]}").fetchall()
        for commentcomment in commentcomments:
            datas['comment'][-1]['review'].append(
                {"review_id": commentcomment[0], "author": commentcomment[1], "content":
 commentcomment[2]})
    return datas
```

This is a difficult function, because its data structure is complex. It have blog content, blog comment, comment review.

First select user upload comparing id blog content. Then use blog_id select comment. Then use comment_id select review

So the data structure is

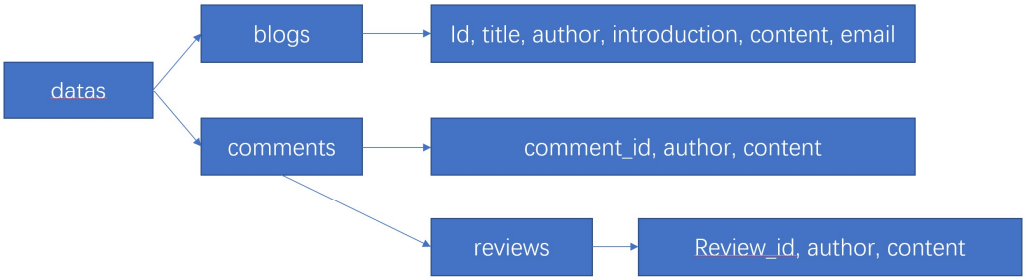


Figure 9 Data structure

3.2.3 Database Design

Database use the Sqlite3, its official website is: https://www.sqlite.org/index.html
Database searching tool is SQLiteStudio, its official website is: https://sqlitestudio.pl/

Create database function I write in python

```
sql = '''CREATE TABLE IF NOT EXISTS bloginfo(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            title TEXT,
            author TEXT,
            introduction TEXT,
            content TEXT,
            email TEXT,
            type TEXT,
            date TEXT)'''
db.execute(sql)
sql = '''CREATE TABLE IF NOT EXISTS blogcommentinfo(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            blog_id INTEGER,
            author TEXT,
            content TEXT)'''
db.execute(sql)
sql = '''CREATE TABLE IF NOT EXISTS commentcommentinfo(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            comment_id INTEGER,
            author TEXT,
            content TEXT)'''
db.execute(sql)
```

```
sql = '''CREATE TABLE IF NOT EXISTS userinfo(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT,
            email TEXT,
            password TEXT)'''
db.execute(sql)

sql = '''CREATE TABLE IF NOT EXISTS emailverify(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            email TEXT,
            verifycode TEXT,
            addtime TEXT)'''
db.execute(sql)
```

3.3.4 Front-Back-connection

After complete Front-End and Back-End code. I use the axios to connect them. Its official website is: https://github.com/axios/axios.

The test tool is postman, its official website is: https://www.postman.com/

In the Front-End file. Use the npm code to install it:

Npm install axios

Then set it in main.js

```
import axios from 'axios'
Vue.prototype.$axios = axios
```

This is a using way.

```
this.$axios.get("http://127.0.0.1:5000/blog/query").then((response) => {
      console.log(response.data);
      for (var i = 0; i < response.data.datas.length; i++) {
        var datas = {
          id: response.data.datas[i][0],
          title: response.data.datas[i][1],
          author: response.data.datas[i][2],
          introduction: response.data.datas[i][3],
          mainbody: response.data.datas[i][4],
          type: response.data.datas[i][5],
          date: response.data.datas[i][6]
        };
        this.items.push(datas);
      }
      console.log(this.items);
      console.log(this.items[0].id);
    });
```

Chapter 4 Test and Question

This part will introduce I complete this web application having test and encountering difficulties

4.1 Difficult

First is the ui framework. In my initial opinion, the web application is a adaptive web, when user open it in different equipment, it will show different ui. But it is a large project and I am not a professional ui designer, so I choose only make a computer web application ui.

Second is the connection between the Front-End and Back-End. Cross-Origin Resource Sharing(CORS) is a based http-header mechanism. Because my Front-End and Back-End is not a server run, so I add a cors in flask

```
from flask_cors import *
    CORS(app, supports_credentials=True)
```

Third is the picture in blog upload question. Because normal md cannot share the picture, it need a website to save the picture and import in md file. So I create a http server to receive the picture. I create a folder named images in my Back-End file. Then write the code

```
@app_blog.route('/upload_image', methods=['POST'])
def upload_image():
    file = request.files.get('image')
    filename = str(time.time()) + "." + file.filename.split('.')[-1]
    file.save("images" + os.path.sep + filename)
    return f"http://localhost:8900/{filename}"
```

Then when I want to run the blog, I will create a http server in images folder

Forth is the written of code, I read many books of html5, javascript, css3 and flask.
For example is flex and grid container in web:
Flex is an one dimension design, all things will set in one line. Grid is a two dimension design
, I need to control columns and rows at the same time. It is more free but more difficult.

4.2 Test and Run

I will upload a compress packets

In the beginning. You should install vue 2.0, vue-cli 4.5, npm and python 3.7. All official website I list:
https://vuejs.org/
https://cli.vuejs.org/
https://nodejs.org/en/

First use the vscode open the frontend file and open the terminal :input npm install to install dependence
Second use the pycharm open the backend file and open the terminal: input pip install -r requirements.txt to install dependence
Third in vscode terminal input npm run serve to run the frontend
Four in pycharm terminal input three code to run the backend:
set flask_app=app.py
set flask_env=development
flask run
Five in backend folder go into images folder open the cmd and input: python -m http.server 8900 to create the file server to receive the picture.

Now you can use the blog.

Chapter 5 Conclusion and future works

5.1 conclusion

I learn many things from this final project. In the half year, I learn many technology of web application, html5, javascript, css3, vue, python, flask. Learn the all web framework system, including the data structure and http server. I list some things as follows.

1: flex, using the flex to control the web interface. By a small game, I command it expertly.
https://flexboxfroggy.com/

2: flask, I read two books and its documents to learn it. In this project, I master the base api design and function design.

Book name:

Flask Web 開發 基於 python 的 web 應用開發實戰

Flask Web 開發實戰 入門、進階與原理分析

3: vue, Vue is a new tool of web developments, it integrate the html5 javascript css3 together. It is the most popular tool in web application now.

5.2 Future work

In the future, I plan add some function in my web blog.

First is mobile equipment ui: in order to let user read blog in anywhere, I want to design some easy ui for the blog, when user use the equipment in upright, they can use dragging left and right to read the md file.

Second is netdisk function. In the initial plan, I want to create a private netdisk for user upload and import the link for md file using. But it encounter a difficult is http server speed and safe. So I remove it now, but I want to I have able to console it in future.

Third is the recommend system. After I learn the machine learning, I will add this function for user to find what they want easier.

Appendix

Official documents
https://vuejs.org/
https://router.vuejs.org/
https://vuex.vuejs.org/
https://cli.vuejs.org/
https://flask.palletsprojects.com/en/1.1.x/
https://github.com/hinesboy/mavonEditor/blob/master/README-EN.md
https://vuetifyjs.com/en/
https://material.io/

reference web
https://github.com/
https://developer.mozilla.org/zh-CN/
https://stackoverflow.com/

easy learning web

https://www.runoob.com/

Resume

Acknowledgement

From the topic selection and data collection to the completion of the thesis, I have received enthusiastic help from many teachers and classmates.

I owe my greatest debt of gratitude to my supervisor Zhi Yao Liang who gives me many useful and benefit suggestions. Teach me the project direction and give the evaluation for my upload reports. I also thanks all come to my (thesis) oral defense teachers, thanks you come up with the question and your suggestions