COMP9900 Project Report

Group NEWCODER

18 November , 2022

[Scrum Master]

Zhe Tong	Front-end	z5298319	z5298319@ad.unsw.edu.au

[Group Member]

Yunfei Wang	Front-end	z5295602	z5295602@ad.unsw.edu.au
Yuhao Wu	Front-end	z5286612	z5286612@ad.unsw.edu.au
Yuechuan Li	Back-end	z5289723	z5289723@ad.unsw.edu.au
Jinbang Xu	Back-end	z5319620	z5319620@ad.unsw.edu.au

Contents

1 Overview 3

1.1 Background for the Project 3

1.2 Aim / Project Objectives 3

1.3 System architecture 3

1.4 Page structure 4

1.5 Page Flow Diagram 5

1.6 Database Design 6

2 Functionality Details 6

2.1 Base Page 6

2.2 Home Page 7

2.3 Login Page 9

2.4 Registration Page & reset password page 10

2.5 Profile page 12

2.6 Recipe page 14

2.7 Create recipe page and Change recipe page 17

2.8 Comment page 18

2.9 Search page & recommendation page 19

2.10 Subscription Page & Saved Page 20

3 Technology Description 21

3.1 Front-end 21

3.2 Back-end 22

3.3 Code Structure 22

4 Installation and Prerequisites 23

5 User document 24

5.1 For all users: 24

5.2 As a creator: 26

5.3 As a visitor: 27

6 Reference 31

1 Overview

1.1 Background for the Project

With the increasing popularity of social platforms, people who love to cook are looking for interesting recipes and want to share their recipes with others, so more and more people who love to cook choose to share their exquisite dishes and recipes on social media. However, because the recipes and dishes on social media are scattered and trivial, the explorers cannot easily and conveniently find the recipes they want, so the explorers are more eager to have a practical and easy-to-use recipe learning and communication platform for them. This project is dedicated to building a functional catering recommendation system platform.

1.2 Aim / Project Objectives

Simplify the web threshold

When implementing functions such as user registration and login, we need to simplify the user's login process to avoid letting users spend a lot of time at this stage.

Improve user experience

Design functions including but not limited to various users and user interactions in user stories, including like, comment on recipes, follow recipe creators you like, recommend recipes and search recipes to motivate more creators.

User-friendly operation

We also hope to make our website pages cleaner and easier to operate, so that users can know the recipes they want without spending a lot of time familiarizing themselves with the functions of the page. Therefore, we have integrated commonly used functions in one operating module, which is convenient for all users.

Solve user problems

To provide users with an excellent search and recommendation engine, we hope to associate the tags of all recipes to recommend the recipe information they most want to see and may be interested in.

1.3 System architecture

Client Side

The front-end (client side) which is a graphical user interface for users to interact. Front-end development is the process of creating front-end interfaces such as web pages or apps (this project refers specifically to the Web) and presenting them to users. The user interface of Internet products is realized through html, CSS, JavaScript and various derived technologies, frameworks and solutions.

Application side

WSGI acts as a web server gateway interface. Plays a vital role when deploying Flask applications. We use WSGI as a bridge between Web server and Python web application.

Back-end side

The web backend is responsible for storing and organizing data, and making sure everything on the web frontend works, the back-end system which is not accessible to the user but supports our designed functions.

Database Side

MySQL is mainly used to store data in the form of tables. The data information of the front-end update page is stored in the database, and the information displayed on the page is also obtained from the database.

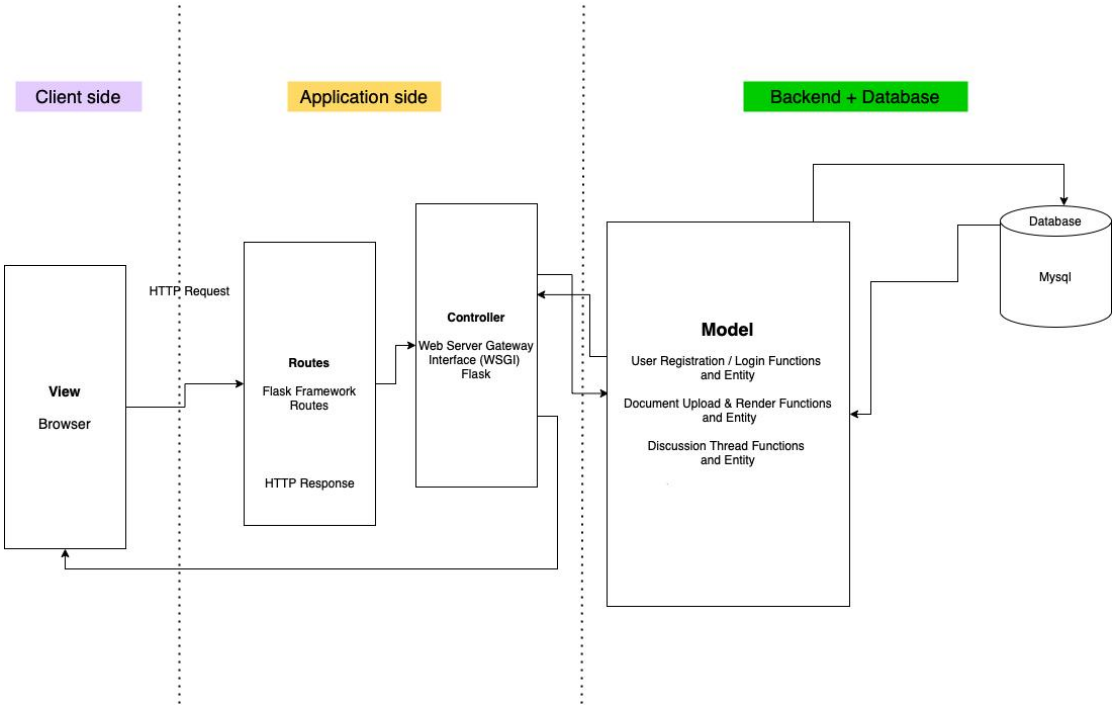


Figure 1.1: Web System architecture

1.4 Page structure

Home page

The home page consists of six modules:
'Log-in', 'Sign-up', 'Profile', 'Recommendation', 'Category' and 'Search'.

Sign-up page

The Sign page provides the explorer with the function of registering as this Web user. The registration content consists of four parts: 'username', 'set password', 'confirm password' and 'email'.

Login page

After successfully registering the user, you can click the Log-in button, the user will enter this page, and the user only needs to fill in the email address and password to complete the login. At the bottom of the login box, users are also provided with the option to retrieve their password through email.

Profile page

Profile page contains seven sub-modules, namely 'Reset Password', 'Login Out', 'My Subscription', 'My Saved', 'My Comments', 'Create Recipe' and 'My Recipes', the functions of each module are Literally.

Recommendation page:

After users have favorited some recipes that they are interested in, this page will recommend similar recipes for users based on the category of the user's favorite recipes and the ingredients used.

Category page:

There are three sub-modules under ‘Category’ module, namely 'Breakfast', 'Lunch' and 'Dinner', each sub-module corresponds to the index page of recipes of this category.

Search page:

The search box can recommend recipes that users may be interested in based on the keywords they search for.

1.5 Page Flow Diagram

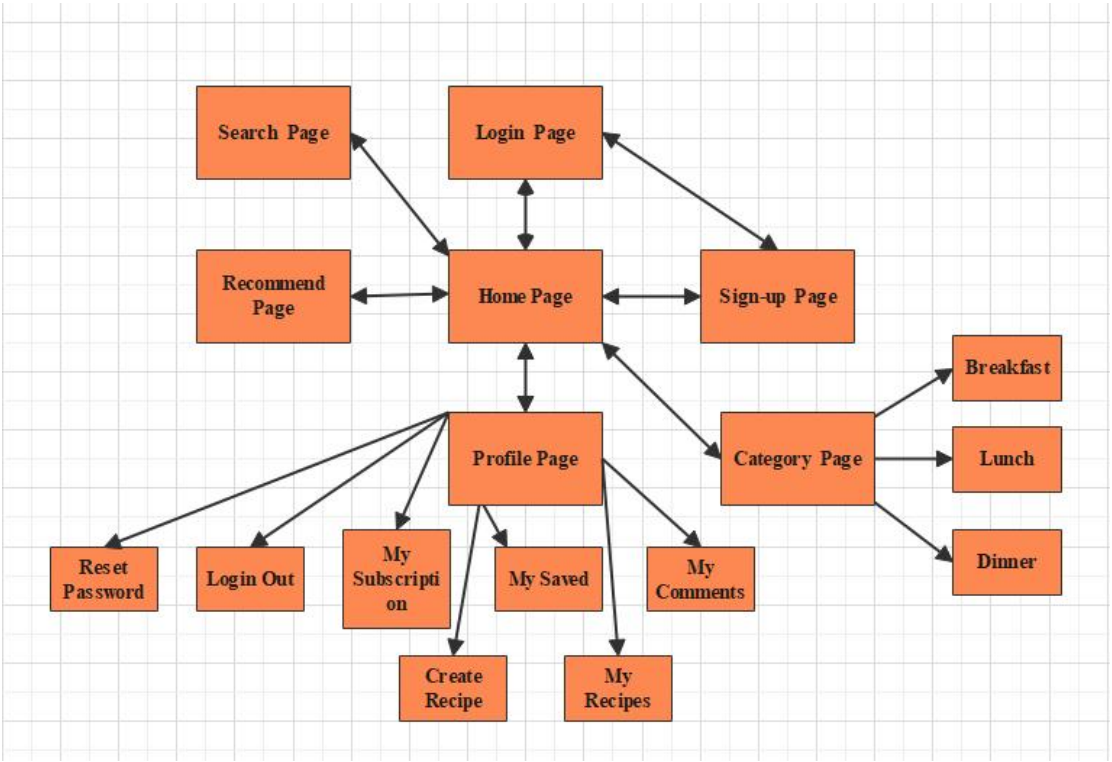


Figure 1.2: Page Flow Diagram

1.6 Database Design

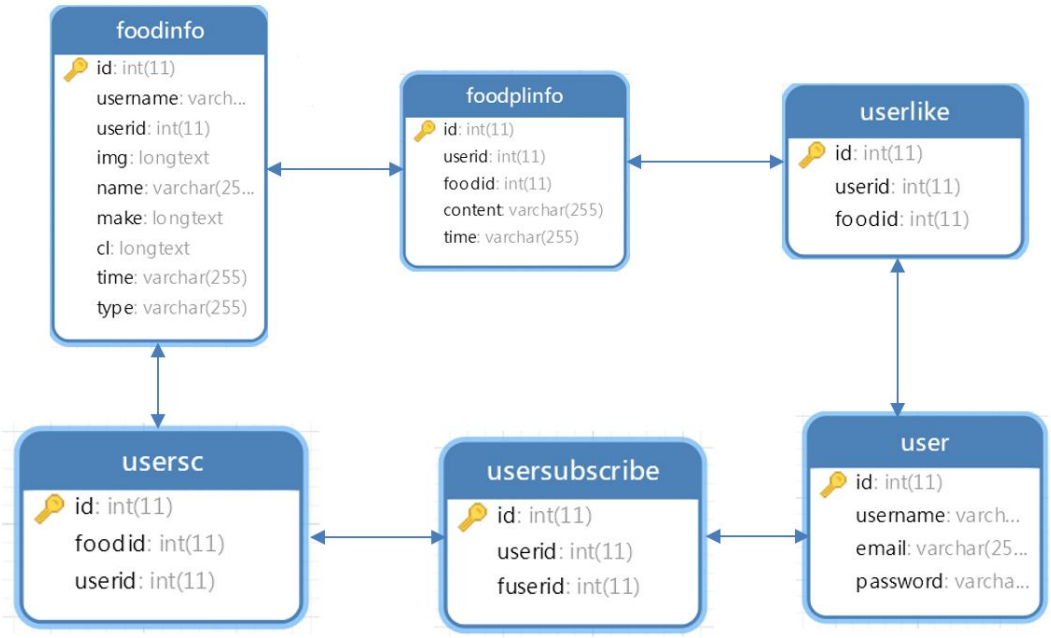


Figure 1.3: Database Design

1. 'foodinfo' is a table for storing created recipe data
2. 'foodplinfo' is a data table for storing recipe comments
3. 'user' is a data table for storing platform user account passwords
4. 'userlike' is a data table for storing users who like recipes
5. 'usersc' is a data table for storing user subscriptions to recipe creators Data table
6. 'usersubscribe' is a data table that stores user favorite recipes.

2 Functionality Details

The fronted file shows below:

Address
../static/js/jquery.form.min.js

2.1 Base Page

At the beginning of website design, we found every page of our websit has similar points which are the header and initial setting. Therefore, we wrote a base page used for reference. Then a lot of duplication of effort is reduced. More details in the following:

Condition	Result
default	Buttons 'Home', 'Recommendation', 'Profile', 'Log in', 'Sign up' show on the header.

The code of HTML framework shows below：

```
<nav class="navbar">
    <div class="nav-center">
        <div class="nav-header">
            <a href="/" class="nav-logo">
                <img src="../static/assets/logo.png" alt="simply recipes"/>
            </a>
            <button class="nav-btn btn">
                <i class="fas fa-align-justify"></i>
            </button>
        </div>
        <div class="nav-links">
            <a href="/" class="nav-link"> home </a>
            <a href="javascript:void(0)" onclick="navrecommendation()" class="nav-link">Recommendation</a>
            <a href="javascript:void(0)"  onclick="navprofile()" id="geren" class="nav-link" >Profile</a>
            <div class="nav-link contact-link">
                <a href="login" class="btn" id="signupbtn"> Log in </a>
                <a href="signin" class="btn" id="signinbtn"> Sign up </a>
            </div>
        </div>
    </div>
</nav>
```

2.2 Home Page

Front-end

the home page is the first page showing for visitors. So we choose light green and white as the global setting color to make an impression on all visitors. The home page consists of the following 5 parts which are shown below:

Header:

According to the base page, the header part contains all of buttons from the base page. More details in the following:

Condition	Result
Not login	Buttons ‘Home’, ’Recommendation’, ’Profile’, ’Log in’, ’Sign up’ show on the header. But ‘Recommendatioin’ and ‘Profile’ would remind visitor ‘Please log in first’ when the visitor click them but have not login.
Login Success	Buttons ‘Home’, ’Recommendation’, ’Profile’, ’Log in’, ’Sign up’ show on the header. All button can show the information for the user.

Introduction:

This part is another part to improve the interface visualization of the website. In this part, we showed the name and slogan of website and put the background image as a delicious food that is attractive.

The code of HTML framework shows below：

```
<header class="hero">
    <div class="hero-container">
        <div class="hero-text">
            <h1>Kitchening</h1>
            <h4>Love Life, Keep Health</h4>
        </div>
    </div>
</header>
```

Search:

Users can search for recipes by inputting keywords like potato, dinner, etc. Then they would go to the search result page. More details in the following:

Condition	Result
default	It will turn to the search result page when visitor input the key words.

The code of HTML framework shows below：

```
<label>
    <input type="search" id="search" name="search" placeholder="search by recipe name or material name" class="form-input">
    <button class="btn search" onclick="sub()">Search</button>
</label>
```

Sort:

There shows the recipe's category. Users sort the recipe into three kinds that are breakfast, lunch and dinner when they create a recipe. Therefore visitors can browse recipes that they want to find by clicking different items. The code of HTML framework shows below：

```
<div class="tags-container">
    <h4>categories</h4>
    <div class="tags-list">
        <a href="/?type=Breakfast"> Breakfast</a>
        <a href="/?type=Lunch"> Lunch</a>
        <a href="/?type=Dinner"> Dinner</a>
    </div>
</div>
```

Recipe list:

There shows some example recipes for visitors to know details of the website especially visitors who have not yet registered. Every recipe has an image and brief information such as name, contributor, the number of likes and saved. The code of HTML framework shows below：

```
<div class="recipes-list">
    {% for datum in data %}
        <a href="recipe?foodid={{ datum.id }}" class="recipe">
            <img
                    src="{{ datum.img }}"
                    class="img recipe-img"
                    alt=""
            />
            <h5>{{ datum.name }}</h5>
            <p>Author : {{ datum.username }} | Save : {{ datum.scnumber }}| Like : {{ datum.likenumber }}</p>
        </a>
    {% endfor %}
```

Back-end

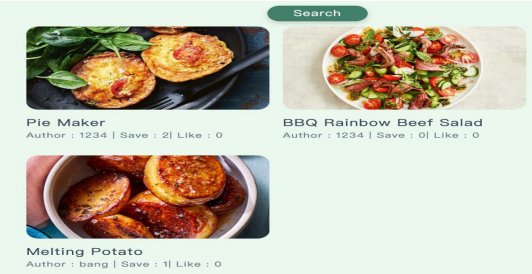
As the main page is mainly used for displaying information and requires less back-end, we only need to go through the recipe information table (foodinfo) to get the recipe id, picture information, recipe name and username of the recipe creator for each recipe for display, as for

the number of subscriptions and likes, the recipe ids obtained above are counted in the corresponding tables, which are the subscription table (tb_ usersc), and the likes table (tb_userlike).

Finally, the recipe id, image information, recipe name, recipe creation user name, number of subscriptions and likes are integrated into a dictionary (see Figure 1) and returned to the front-end for display (see Figure 2).

```
sql = 'select id,img,name,username from foodinfo'
res = until.serch(sql)
data = []
for i in res:
    scnumber = until.serch('select count(id) from tb_usersc where foodid = {0}'.format(i[0]))[0][0]
    likenumber = until.serch('select count(id) from tb_userlike where foodid = {0}'.format(i[0]))[0][0]
    s = {
        "id": i[0],  # recipe id
        "img": i[1],
        "name": i[2],  # recipe name
        "username": i[3],
        "scnumber": scnumber,  # number of subscriptions
        "likenumber": likenumber,  # number of likes
    }
    data.append(s)
return render_template('index.html', data=data)
```

(Figure 1)



(Figure 2)

2.3 Login Page

Front-end

As we know, the login page is for users to log in their account. We designed the account should be loged in by email with correct password.

'Login', 'Sign up', 'Forget password' show on this page. The password set into invisible for security reasons. More details in the following:

Condition	Result
Login Success	Turn to the home page and all of buttons can be used.
Password incorrect	The website would notice that the password is incorrect.
Invalid Email	The website will notice 'the account have not sign up'
Email or password is empty	There will have a warning on which is blank.

The code of HTML framework of Login shows below:

```
<h4 class="page-title">User Login</h4>
<hr>
<section class="contact-container">
    <article>
        <form class="form contact-form" action="login" method="post" id="ajax" enctype="multipart/form-data">
            <div class="form-row">
                <label html="email" class="form-label">Mailbox</label>
                <input type="email" name="email" id="email" class="form-input"
                       placeholder="Please enter your mailbox" required/>
            </div>
            <div class="form-row">
                <label html="password" class="form-label">Password</label>
                <input type="password" name="password" id="password" class="form-input"
                       placeholder="Please enter your password" required/>
            </div>
            <button type="submit" class="btn btn-block">
                Login
            </button>
            <div class="contact-line">
                <label class="contact-right">Don't have an account? <a href="signin">Sign up
                </a></label><br>
                   <label class="contact-right">  <a href="resetpassword">Forget password
                </a></label>
            </div>
        </form>
    </article>
</section>
```

Back-end

Receive the email,password from the front-end user, then go to the user table (tb_user) to check if there is this record according to the email, if there is, then confirm if the password is the same, if it is, then login successfully and return the successful login information to the front-end to show to the user, if it is not the same, then login failed and return the wrong password information to the front-end.

If the user table (tb_user) is empty, the login will fail and the user will be returned to the front-end with the unregistered information.

2.4 Registration Page & reset password page

Front-end

Under our website design, an email address only can register one account. We have ‘Sign up’ and ‘Login’ button on this page. After the user logs in, the password can be changed on the profile page. At the beginning of the website design, from the perspective of user information security, we hope to perform email verification when changing the password. However, in the follow-up work, it was found that networking is required to realize the above function, so it was simplified. And the page includes all these parts:

Name of parts	Description
Username	Name of the user
Password	The input should be invisible
Confirm password	This should be the same as the password
Email	Must in email fomat
Log in	If the user is already registered, you can log in directly

All possible conditions are listed in the following:

Condition	Result
Register success	User can see their profile page
The format of email error	The website would notice user
Email has already registered	Pop up the notice 'The email is already exists'
Confirm password is different with before	The website would show that passwords are different
Any item is empty	The website will have a warning.

The code of HTML framework shows below:

```
<form class="form contact-form" action="signin" method="post" enctype="multipart/form-data" id="ajax"
    <div class="form-row">
        <label html="name" class="form-label">Username</label>
        <input type="text" name="name" id="name" class="form-input"
               placeholder="please enter your username" required>
    </div>
    <div class="form-row">
        <label html="password" class="form-label">Password</label>
        <input type="password" name="password" id="password" class="form-input"
               placeholder="please enter your password" required/>
    </div>
    <div class="form-row">
        <label html="password" class="form-label">Confirm Password</label>
        <input type="password" name="rpassword" id="rpassword" class="form-input"
               placeholder="please confirm your password" required/>
    </div>
    <div class="form-row">
        <label html="email" class="form-label">Mailbox</label>
        <input type="email" name="email" id="email" class="form-input"
               placeholder="please enter your email" required/>
    </div>
    <button type="submit" class="btn btn-block">
        Sign up
    </button>
    <div class="contact-line">
        <label class="contact-right">Already have an acount? <a href="login">Login</a></label>
```

Back-end

Receive the email,password from the front-end user, then go to the user table (tb_user) to check if there is this record according to the email, if there is, then confirm if the password is the same, if it is, then login successfully and return the successful login information to the front-end to show to the user, if it is not the same, then login failed and return the wrong password information to the front-end.

If the user table (tb_user) is empty, the login will fail and the user will be returned to the front-end with the unregistered information. (see Figure 3)

```python
def login(email, passwrod):
    sql = 'select * from tb_user where email = "{0}"'.format(email)
    res = serch(sql)
    if res != ():
        if res[0][3] == passwrod:
            data = {
                'code': 200,
                'msg': "login successful",
                'userid': res[0][0]
            }
            return data
        else:
            data = {
                "code": 202,
                "msg": "wrong account or password"
            }
            return data
    else:
        data = {
            "code": 201,
            "msg": "unsign username"
        }
        return data
```

(Figure 3)

2.5 Profile page

Front-end

After logging in as a user, you can enter your profile page to view and manage information, which includes resetting passwords, logging out, my subscriptions, my saves, my comments and my recipes. Due to time constraints, some details have not been perfected, such as not being able to view people who follow the user. More details in the following:

Name of parts	Description
Resetting password	Jump to reset password page
Log out	Log out, return to the home page
My subscriptions	View other users subscribed by the user and their updates
My saved	View user favorite recipes
My comments	View and manage comments made by users
Create recipes	Create a new recipe
My recipes	Manage and view user-published recipes

The code of HTML framework shows below:

```
<header>
    <div class="user-left">
        <img src="../static/assets/logo.png" style="height: 115px" class="userlogo">
        <br>
        <a href="resetpassword" class="user-btn btn">Reset password</a>
        <br>
        <a href="javascript:void(0)" onclick="loginout()" class="user-btn btn">Log out </a>
    </div>
    <div class="user-right">

        <p>Username: {{ data.userinfo.username }}</p>
        <p>Email:{{ data.userinfo.email }}</p>
        <a href="Mysubscription" class="user-btn visitor-follow">My subscription</a>
        <a href="Mysaved" class="user-btn visitor-follow">My saved</a>
        <a href="Mycomments" class="user-btn visitor-follow">My comments</a>
        <br>
        <a href="userupdatesp" class="user-btn visitor-save">Creat Recipe</a>
        <br>
    </div>
</header>
```

Back-end

The user id is received from the front end and we first use the user id to access the recipe information table (foodinfo) to find all the recipe ids created by the user and then access the tb_foodplinfo table based on the recipe id to get all the information about all the recipes used for display.

The user's id is then used to query the user's table (tb_user) for other information, such as the favourites list, the followers list and the comments list. (see Figure 4)

Finally, all the information about the user (except the password) is returned to the front-end for display, along with all the information used to create the recipe.

```
def getuserspinfo(userid):
    sql = 'select * from foodinfo where userid = {0}'.format(userid)
    res = serch(sql)
    data = []
    for i in res:
        plcount = serch('select count(id) from tb_foodplinfo where foodid = {0}'.format(i[0]))[0][0]
        s = {
            'id': i[0],
            'img': i[3],
            'foodname': i[4],
            'plcount': plcount
        }
        data.append(s)
    return data


def getuserscinfo(userid):...


def getuserplinfo(userid):...
```

(Figure 4)

2.6 Recipe page

Front-end

On this page, we can see the recipe published by users. The left side is the display picture, and the right side is the name of the recipe, contributors, ingredients, steps, tags and comment area. Below the whole, we placed a recipe recommendation area designed according to the ingredients. In addition, we also placed three buttons on this page, like, save and follow. Relevant data is linked to recipe information on different pages and details on personal pages.

Name of parts	Description
Recipe information	All the details about the recipe
Recommendation area	Recipe recommendations based on ingredients

Condition	Result
Click “like”	The first click increases the number of likes for the recipe, and the second click cancels the like operation
Click “save”	The first click will save the recipe to my favorites, and the second time will cancel the favorite operation
Click “follow”	Click to follow the menu contributor for the first time, and you can check the other party's update information at any time. Second click to unfollow action
Input and sent comment	Comment on the recipe. The comment message is visible to all users.

The code of HTML framework shows below:

```
<div class="recipe-page">
    <section class="recipe-hero">
        <img
            src="{{ data.img }}"
            class="img recipe-hero-img"
        />
        <article class="recipe-info">
            <h2>
                {{ data.name }}
                <div class="visitor-upper"> author:<i>{{ data.username }} <span id="date"></span></i></div>
                <div class="visitor-action">
                    <button class="visitor-func visitor-like" onclick="userlike({{ data.id }})">
                        Like({{ data.likecount }})
                    </button>
                    <button class="visitor-func visitor-save" onclick="usersc({{ data.id }})">Save</button>
                    <button class="visitor-func visitor-follow" onclick="usersubscribe({{ data.userid }})">
                        Follow
                    </button>
                </div>
            </h2>
            <article class="second-column">
                <div>
                    <h4>ingredients</h4>
                    <p class="single-ingredient">
```

```
                        {% for foo in data.cl %}
                            {{ foo }}
                        {% endfor %}
                    </p>
                </div>
            </article>
            <h4>Process</h4>
            <p>
                {% for foo in data.make %}
                {{ foo }}
                {% endfor %}
            </p>
            <p class="recipe-tags">
                Tags :
                {% for foo in data.cl %}
                    <a href="tag-template.html" style="margin:5px ">{{ foo }}</a>
                {% endfor %}

            </p>
            <input class="form-input visitor-comment" id="message" placeholder="Comment here"></input>
            <button class="btn" onclick="sub({{ data.id }})">send</button>
```

Back-end

Get the recipe id sent from the front-end request, then use the recipe id as a condition to get the corresponding recipe information from three different tables: the recipe info table (foodinfo), the subscription table (tb_usersc) and the like table (tb_userlike).

The recipe details obtained (picture information, recipe name, user name created, step information, ingredient information) are displayed to the front-end for presentation to the user. (see Figure 5)

```
def foodfind(foodid):
    sql = 'select cl from foodinfo where id = {0}'.format(foodid)
    res = serch(sql)[0][0].split(',')
    data = []
    infolist = []
    cl_r = 0
    for x in res:
        list1 = serch('select id from foodinfo where cl like  "%{0}%"'.format(x))
        for i in list1:
            if i[0] not in data and foodid != str(i[0]):
                data.append(i[0])
    for x in data:
        list2 = serch('select id,img,name,username,cl from foodinfo where id =  "{0}"'.format(x))
        for i in list2:
            scnumber = serch('select count(id) from tb_usersc where foodid = {0}'.format(i[0]))[0][0]
            cl_p = i[4].split(',')
            random.shuffle(res)
            for y in res:
                if y in cl_p:
                    cl_r = y
                    break
            s = {
                "id": i[0],
                "img": i[1],
                "title": i[2],
                "username": i[3],
                "scnumber": scnumber,
                'cl':cl_r
            }
            infolist.append(s)
    return infolist
```

(Figure 5)

If the user has filled in the comment information for the recipe page, the back-end receives the comment information from the user again. If the comment is empty, the comment fails and a message is returned to the front end that the content cannot be empty. If it is not empty, then the user id, recipe id, comment, time, which created the recipe, is recorded as a row in the tb_foodplinfo table, and the comment succeeds and a successful comment message is returned to the front-end user for prompting. (see Figure 6)

```
def userpl(userid, foodid, content):
    import time
    if content == " ":
        data = {
            'code': 201,
            'msg': "input shouldn't empty"
        }
        return data
    time = time.strftime('%Y-%m-%d', time.localtime())
    sql = 'insert into tb_foodplinfo(userid,foodid,content,time) values("%s","%s","%s","%s")' % (
        userid, foodid, content, time
    )
    insert(sql)
    data = {
        'code': 200,
        'msg': "comment successful"
    }
    return data
```

(Figure 6)

2.7 Create recipe page and Change recipe page

Front-end

We have simplified the steps of creating a new recipe. On our website, you only need to fill in the recipe name, category, ingredients, steps and upload food pictures. When users want to change the published recipes, they can go back to the personal homepage, and there are delete and edit buttons under each recipe. More details in the following:

Name of parts	Description
Recipe name	Name of recipe
Category	Choose between breakfast, lunch and dinner
Ingredients	Only fill in the main ingredients for tags extraction and recommendation systems
Process	Food production steps
Upload image	Upload food image locally

Condition	Result
Submit success	Successfully published the recipe
Reset	All content will be initialized
Any item is empty	The website will show a warning.

The code of HTML framework shows below:

```
<form action="userspcahnge" id="ajax" enctype="multipart/form-data"
    method="post"        >
    <input class="form-input" type="text" name="fooid" value="{{ data.id }}"
        placeholder="please enter recipe name" hidden>
    <div class="create-recipe">
        Recipe Name:
        <input class="form-input" type="text" name="cpname" value="{{ data.name }}"
            placeholder="please enter recipe name" required>
    </div>
    <div class="create-recipe">
        <label for="dropdown">type<br>
            <select id="dropdown" class="form-input" required name="sel">
                <option value="" selected>{{ data.type }}</option>
                <option value="1">Breakfast</option>
                <option value="2">Lunch</option>
                <option value="3">Dinner</option>
            </select>
        </label>
    </div>
    <div class="create-recipe">
        <p>Materials:</p>
        <textarea rows="10" class="form-input"
            placeholder="Each step is directly separated by | such as "thawing| slicing""
            required name="cl">{{ data.cl }}</textarea>
```

```
    <div class="create-recipe">
        <p>Process:</p>
        <textarea rows="10" class="form-input"
            placeholder="Each material is directly separated by | such as "beef|tomato""
            required name="make">{{ data.make }}</textarea>
    </div>
    <div class="create-recipe">
        <tr>
            <td><label>Recipe Image:</label></td>
            <td><input type="file" name="img"/></td>
        </tr>
    </div>
    <div class="create-recipe">
        <input type="submit" value="Create" class="btn">
        <input type="reset" value="back" class="btn">
    </div>
```

Back-end

Create:

The back-end receives the recipe name, type, materials, process, recipe image from the user input and also records the time, then all the above data is inserted into a new line of foodinfo and we return the successful recipe creation message to the front-end for prompting the user. (see Figure 7)

```
def userupdatesp(userid, cpname, make, cl, pic_base64, time, type):
    username = serch('select username from tb_user where id = {0}'.format(userid))[0][0]
    sql = 'insert into foodinfo(`username`, `userid`, `img`, `name`, `make`, `cl`, `time`,`type`) values ("%s","%s","%s","%s","%s","%s","%s","%s")' % (
        username, userid, pic_base64, cpname, make.replace('|', ','), cl.replace('|', ','),
        time, type
    )
    insert(sql)
```

(Figure7)

Change:

The back-end receives the recipe name, type, materials, process, recipe image from the user input, and also records the time, then updates the data in the foodinfo table corresponding to that row based on the above data, and then we return the successfully updated recipe information to the front-end for prompting the user. (see Figure 8)

```python
def changeinfo(foodid):
    sql = 'select * from foodinfo where id = {0}'.format(foodid)
    res = serch(sql)[0]
    data = {
        'id': res[0],
        'username': res[1],
        'userid': res[2],
        'img': res[3],
        'name': res[4],
        'make': res[5].replace(',', '|'),
        'cl': res[6].replace(',', '|'),
        'type': res[8]
    }
    return data
```

(Figure 8)

2.8 Comment page

Front-end

On this page, the comments posted by users and the corresponding recipes and recipe information will be displayed. Users can delete their own comments here.

Condition	Result
Delete comment	Delete the corresponding comment.

The code of HTML framework shows below:

```
{% for i in data %}
    <div>
        <div><span style="color: gray;font-size: 10px">foodname  </span><br>{{ i.foodname }}</div>


    </div>
    <span style="color: gray;font-size: 10px">content</span><br>
     <span>{{ i.cotent }}</span><br>
    <span style="color: gray;font-size: 10px">{{ i.time }}</span>
    <div>
        <button type="button" style="background-color: aquamarine;Border-radius:15px"
                onclick="deluserpl({{ i.id }})">
            Delete comment
        </button>
    </div>
    <hr>
```

Back-end

The back-end receives the id of the user from the front-end, then we use the user's id to check the recipe display table (tb_foodplinfo) to get the user id, recipe id, comment, time of these information, which represents what the user has commented on the recipe at that time.

The user id is then used to get the name of the recipe from the recipe information table

(foodinfo), and this information is then combined and returned to the front-end for display. (see Figure 4)

2.9 Search page & recommendation page

Front-end

Both the search page and the recommendation page are result display pages. Display optional recipes and brief information based on the user's searched content and recommended content.

The code of HTML framework shows below:

```
<h4>Search Result</h4>
<!-- recipes list -->
<div class="recipes-list">
    {% for datum in data %}
        <a href="recipe?foodid={{ datum.id }}" class="recipe">
            <img
                    src="{{ datum.img }}"
                    class="img recipe-img"
                    alt=""
            />
            <h5>{{ datum.name }}</h5>
            <p>Author : {{ datum.username }} | Save : {{ datum.scnumber }}</p>
        </a>
    {% endfor %}
```

Back-end

Reads the keywords entered by the user in the search box on the home page and uses the keywords read as criteria to go to the recipe information table (foodinfo) for a fuzzy search, (see Figure 9). name stands for the name of the recipe and cl stands for the ingredients required for the recipe.

```
'select * from foodinfo where name like "%{0}%"  or cl like "%{1}%"'
```

(Figure 9)

The recipe ids, recipe user ids and names, image information and recipe names are then packed into a dictionary and inserted into a list, which is then returned to the front-end(see Figure 10)

```
    res = serch(sql)
    data = []
    for i in res:
        s = {
            'id': i[0],
            'username': i[1],
            'userid': i[2],
            'img': i[3],
            'name': i[4],
        }
        data.append(s)
    return data
```

(Figure 10)

2.10 Subscription Page & Saved Page

Front-end

The subscription page will display the information of the people followed by the user and the recipes recently uploaded by them. The saved page will display saved recipes.

The code of HTML framework shows below:

```
{% for datum in data %}
    <span style="color: gray">The author name of the subscription</span>
    <h2>{{ datum[0].username }}</h2>
    <span>His recipes</span>
    <br><br>
    <div class="recipes-list">
        {% for foo in datum %}
            <a href="recipe?foodid={{ foo.id }}" class="recipe">
                <img
                    src="{{ foo.img }}"
                    class="img recipe-img"
                    alt=""
                />
                <h5>{{ foo.name }}</h5>
            </a>
        {% endfor %}
    </div>
    <hr>
{% endfor %}
```

Back-end

Subscribe:

The back-end receives the id of the user sent by the front-end, then we use the user's id to check the user subscription table (tb_usersubscribe), then take the first step to check as a condition to the recipe information table (foodinfo) to get the corresponding recipe details (recipe id, created user id, type, picture, etc.), and then integrate this information and return it to the front-end for display.(see Figure 11)

```
def Mysubscription(userid):
    sql = 'select fuserid from tb_usersubscribe where userid = {0}'.format(userid)
    res = serch(sql)
    result = []
    for i in res:
        data = []
        sql = 'select * from foodinfo where userid = {0} order by time DESC'.format(i[0])
        res = serch(sql)
        for x in res:
            s = {
                'id': x[0],
                'username': x[1],
                'userid': x[2],
                'img': x[3],
                'name': x[4],
                'make': x[5].replace(',', '|'),
                'cl': x[6].replace(',', '|'),
                'time': x[7],
                'type': x[8]
            }
            data.append(s)
        result.append(data)
    return result
```

(Figure 11)

Save:

The back-end receives the id of the user sent by the front-end, then we use the user's id to check the tb_usersc table to get the id of the recipe to which the user is subscribed, and then go to the foodinfo table to find the corresponding picture and recipe name based on the recipe id obtained, and then package this information back to the front-end for display. (see Figure 4)

3 Technology Description

3.1 Front-end

HTML

The Hyper Text Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

JavaScript

JavaScript (JS) is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. JavaScript has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

Bootstrap

Bootstrap is a powerful toolkit when designing web user interfaces --it is a collection of HTML, CSS, and JavaScript tools for creating and building web pages and web applications. Bootstrap offers consistent design by using highly reusable components. As we do not have enough frontend development knowledge, this framework helps our group to design clean web page appearances, add smooth animations, and provide useful functionalities.

Icon

Well-designed icons, in combination with text, can help web developers to create a meaningful link between the icons and ideas expressed in the content, which can enhance memorability and readability of our website.

3.2 Back-end

Python 3

A general-purpose programming language which consistently ranked as one of the most popular programming languages. Developers announced it as an object-oriented, interpreted, and interactive programming language.

Flask

Flask is a micro web framework written in Python. It does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

MySQL

MySQL is an open-source relational database management system. It is free and open-source software under the terms of the GNU General Public License, and is also available under a variety of proprietary licenses. It has stand-alone clients that allow users to interact directly with a MySQL database using SQL, but more often, MySQL is used with other programs to implement applications that need relational database capability.

3.3 Code Structure

- static---------- Static Files of the website
  - css---------- styling
  - assets---------- default resources used in website
  - images---------- stores images used in website
  - js---------- bootstrap and jquery scripts

- templates---------- HTML pages
  - base.html---------- base elements of the head of pages
  - index.html---------- landing page of website
  - login.html---------- login page
  - signup.html--------- signup page
  - resetpassword.html-- resetting user's password page
  - profile.html-------- user's profile page
  - userupdatesp.html--- recipe creating and uploading page
  - change.html--------- recipe editing page
  - recipe.html--------- recipe detail page
  - searchjg.html------- searching page
  - recommendation.html- recommendation page
  - Mysaved.html-------- user's saved recipes
  - Mysubscription.html- user's subscription

- Mycommonts.html----- user's conmments

- app.py---------- Main Flask Application Code
- until.py---------- mysql configuration and back-end functions
- food.sql---------- dumped sql file for importing into database when required

4 Installation and Prerequisites

We recommend you run this app on Python 3.9 with the dependencies listed below, make sure you have pip package manager to install all of the requirement packages for application (comes by default with python).

You will also need MySQL installed in system to run the application and optionally Apache server to interact with database from phpMyAdmin. Install XAMPP server from 8.1.6 from https://www.apachefriends.org/download.html.

Installation

Step 1:

Open the path of this folder and run the following command in the terminal. This will install all the required packages.

$ pip install flask
$ pip install json
$ pip install os
$ pip install base64
$ pip install pymysql
$ pip install numpy
$ pip install random

Step 2:

Start MySQL Server & Apache from Xampp.

Adjust the username, password and port number in until.py according to your MySQL configuration.

go to localhost/phpMyAdmin.

Create a database called 'food'.

Click 'Import' from top navigation file.

Under the 'File to Import', browse and select the 'food.sql' file present in this repo. This will import all tables into the database.

Step 3:

Open terminal inside this repository in your local system and execute the following code (Note: if your python 3+ is configured to python3, replace python below with python3).

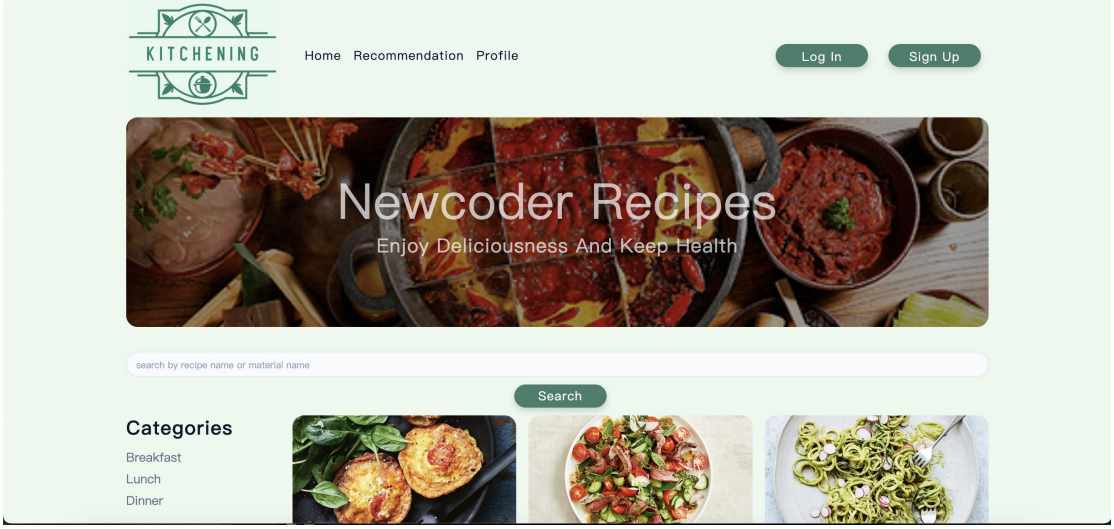
$ python app.py

This will start flask and give you the localhost url for the webapp, use the same to load the webapp.

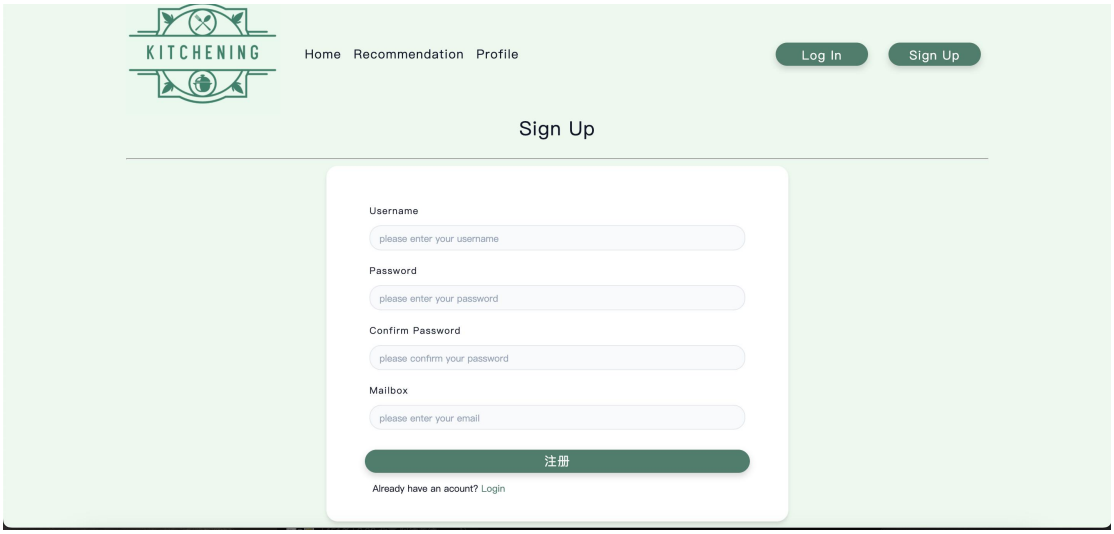
5 User document

5.1 For all users:

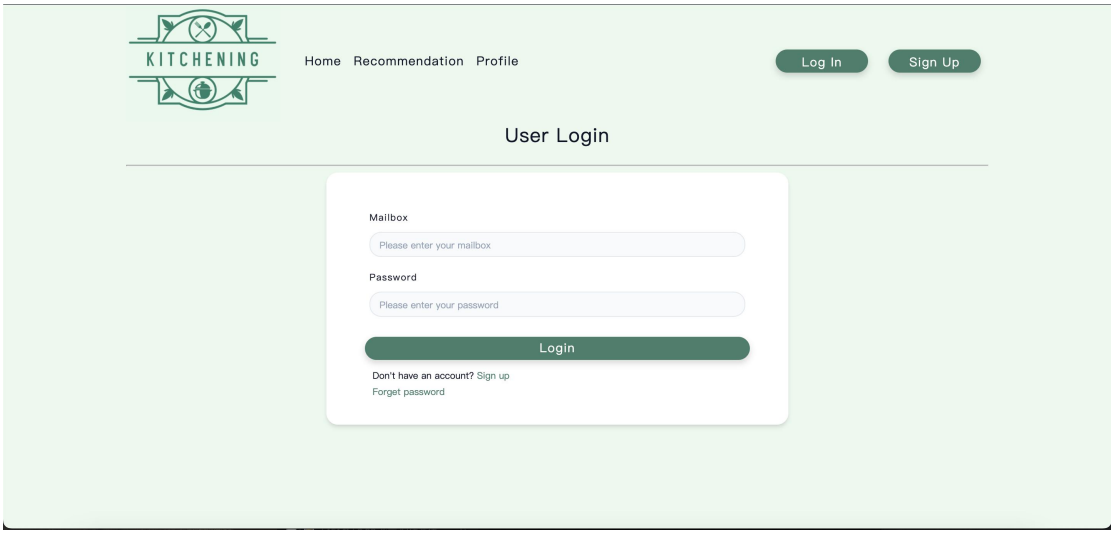
1. First of all, the landing page for all visitors will be the home page, which has the URL address" http://127.0.0.1:5000/"
   Visitors can take a view of the popular recipes on this page which site recommended.



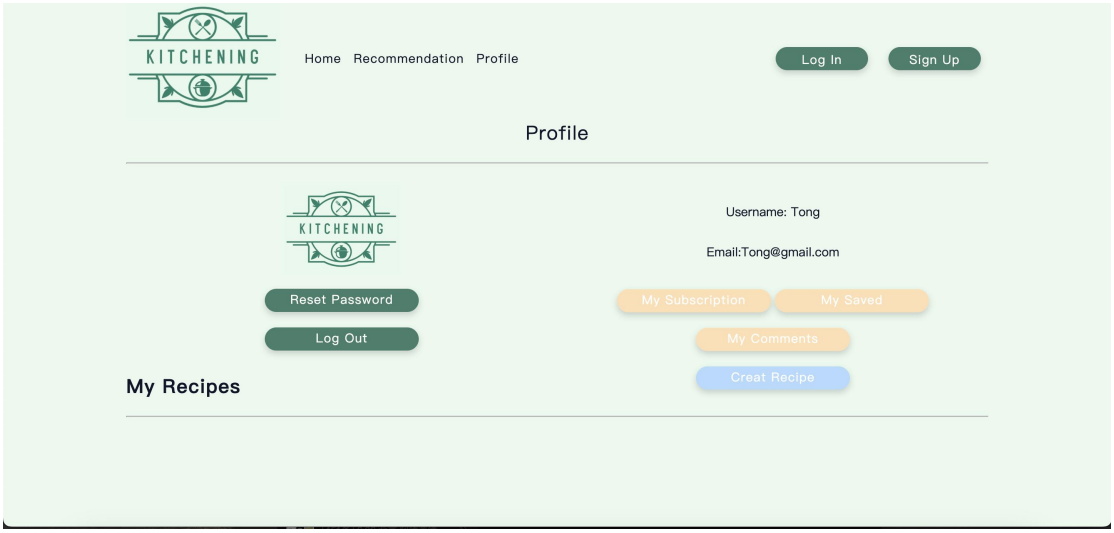
2. If visitor want to create the recipes or try the like/save/comment/follow functions, they can log in to an account. They can sign up first, if they already have an account, just jump to step 3.



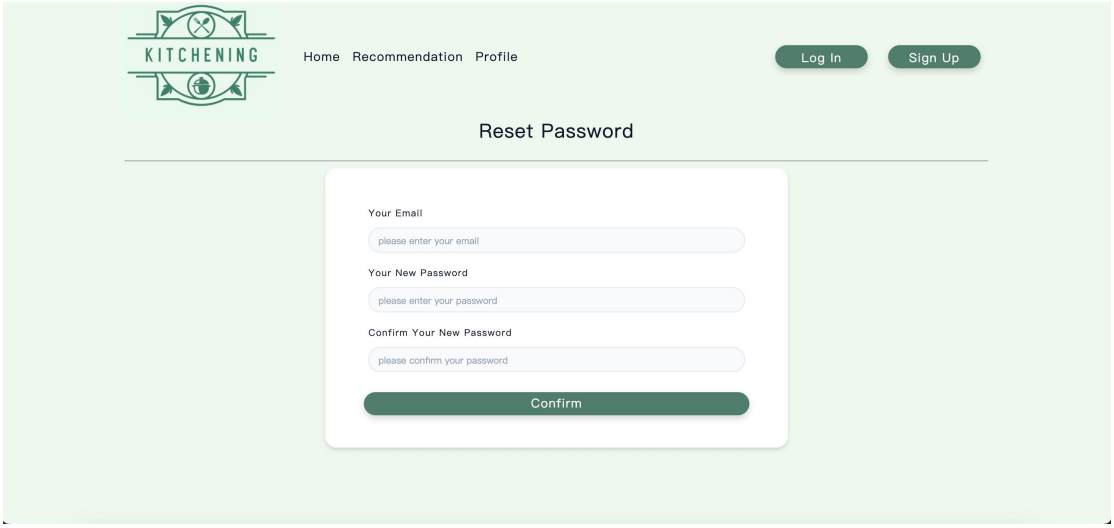
3. User log in.



4. User can check and manage their profile including password, user name, subscription, saved recipes, comments and created recipes.

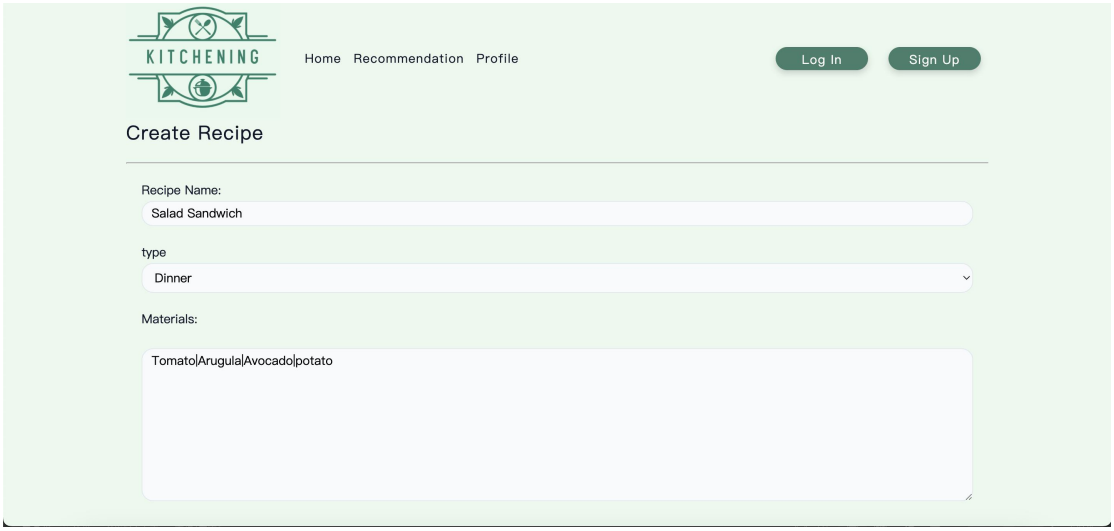


5. Use correct email, users can reset their password.

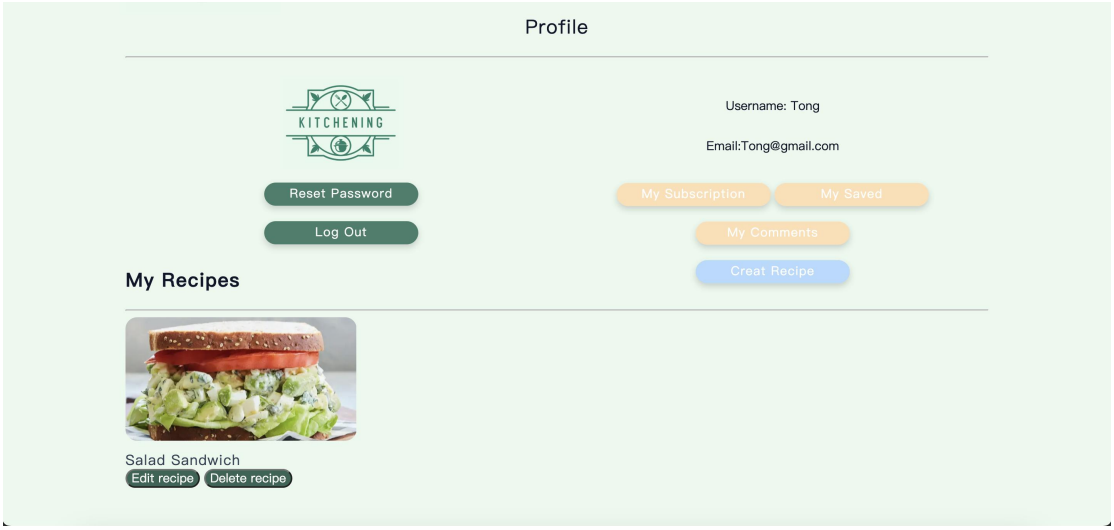


5.2 As a creator:

1. Click to create recipe button which in profile page, user can create their own recipe and can also change it in after.

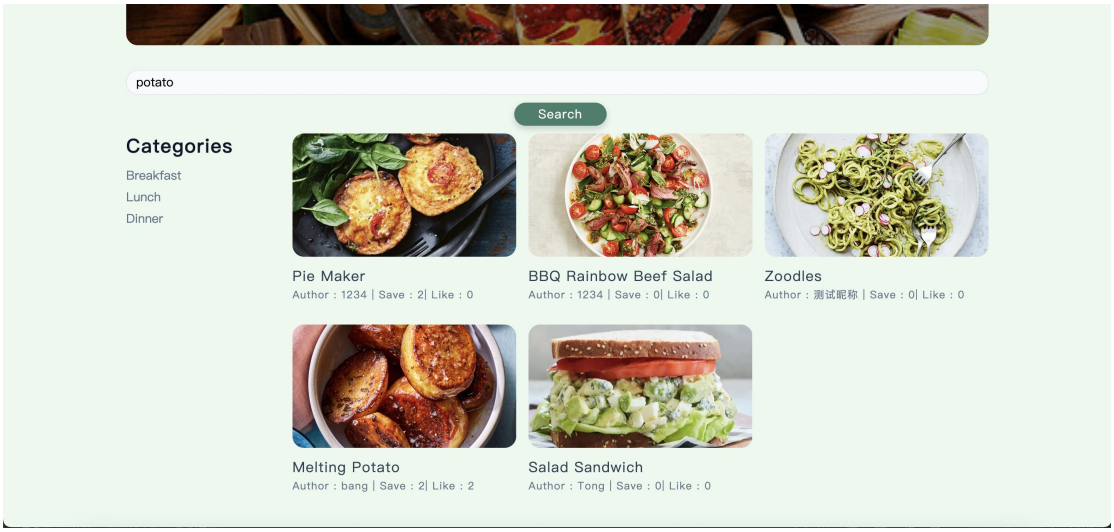


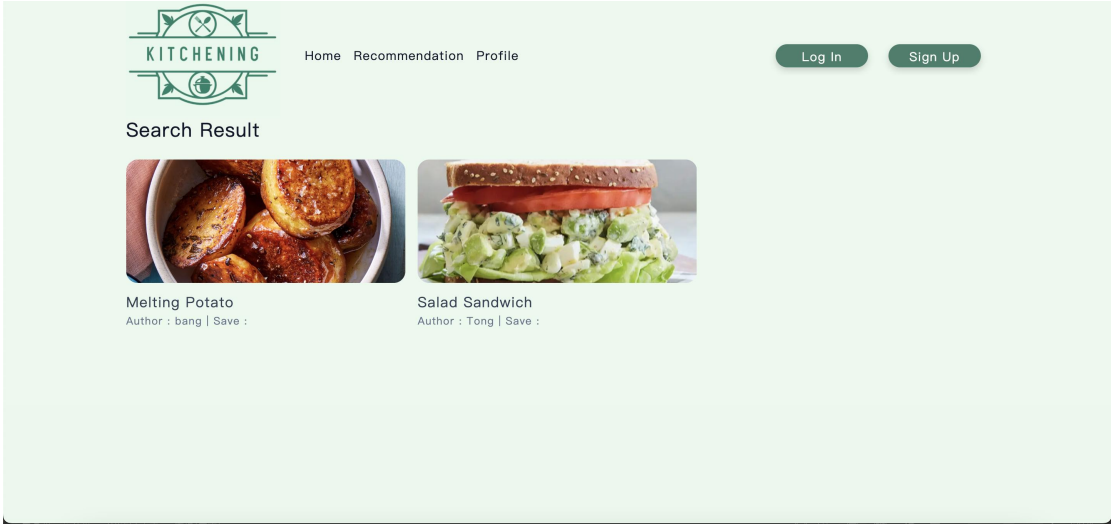
2. User can check the recipe which just created in profile page. And can also edit it or delete it.



5.3 As a visitor:

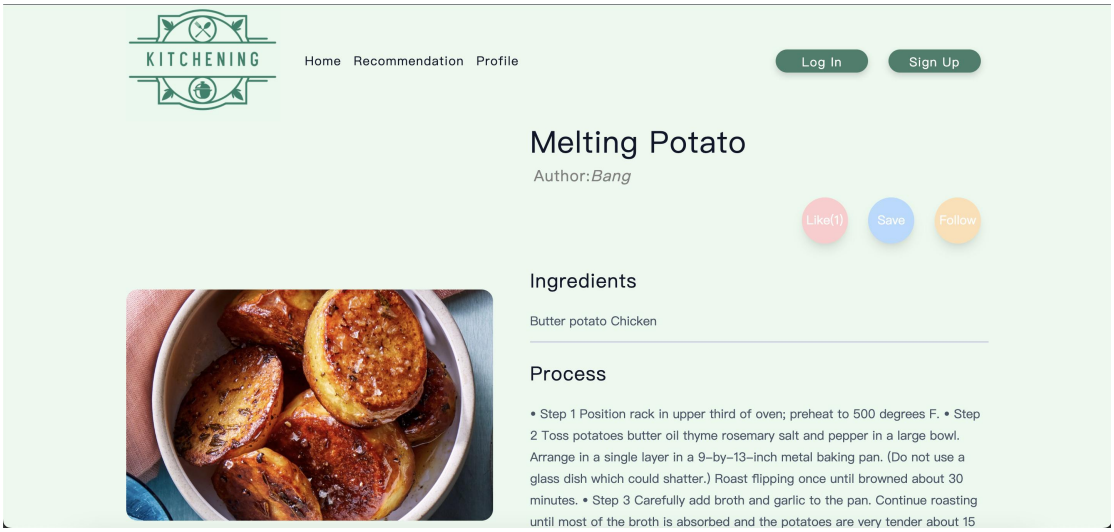
1. User can input one or more key words in search bar, site will give the results according to the key words.



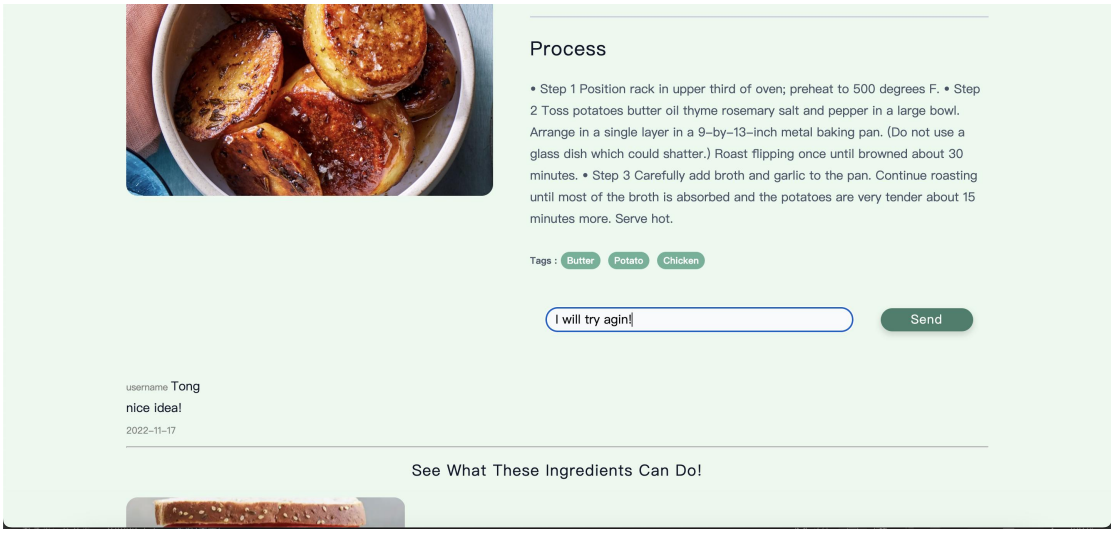


2. When user in recipe page, they can click the like, save and follow button to like/cancel

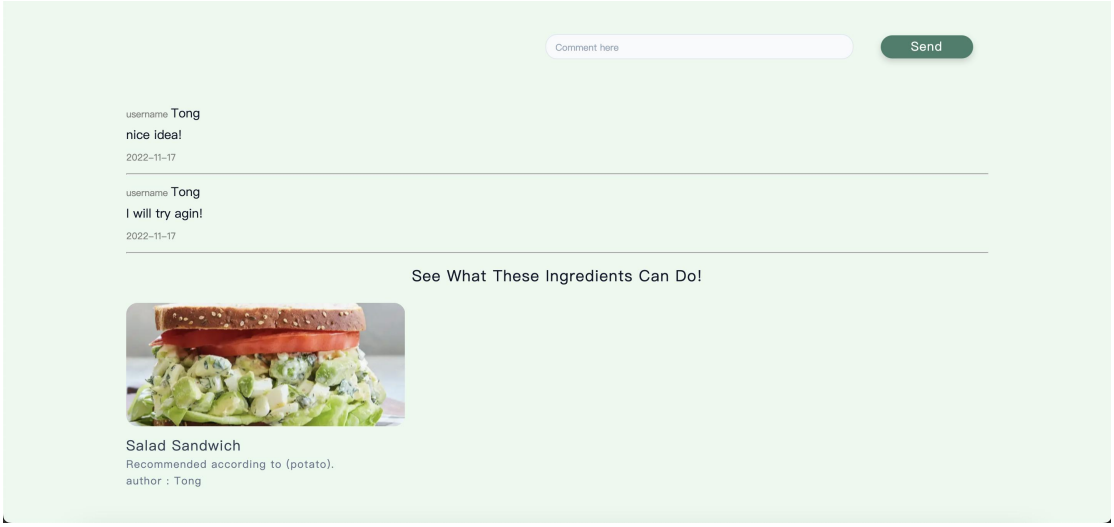
like, adding this recipe to save list, follow/unfollow the creator.



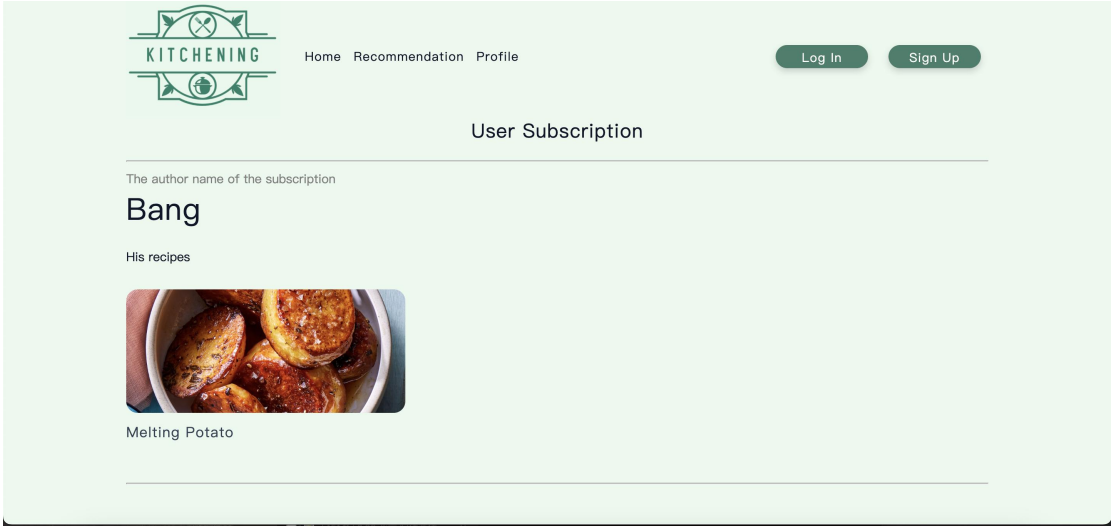
3. User can input the comment message in comment bar to comment. And the comment would be show at below which is viewable for all visitors.



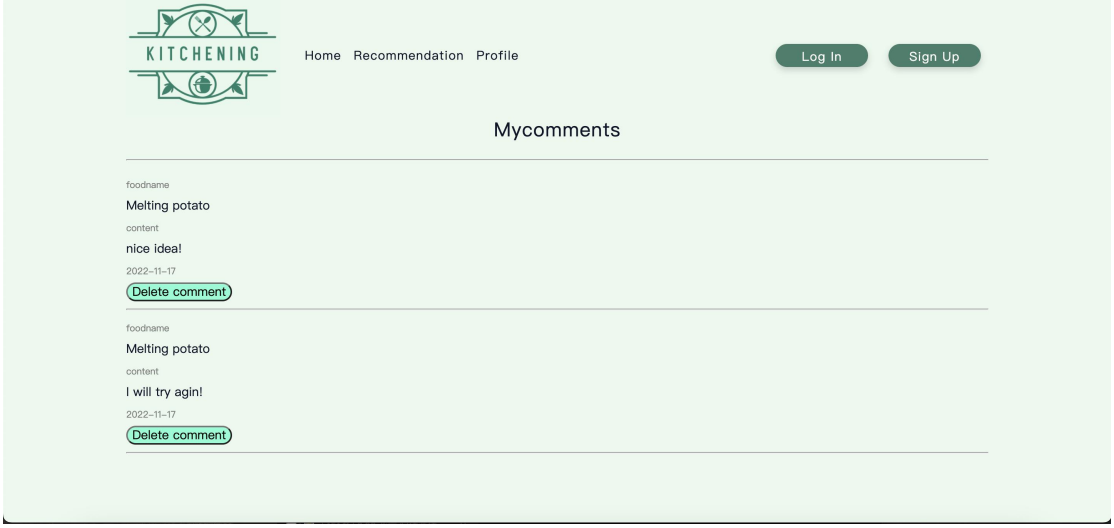
4. Meanwhile, user can find more recipes below the recipe page, which are be recommended by the relation of this recipe.



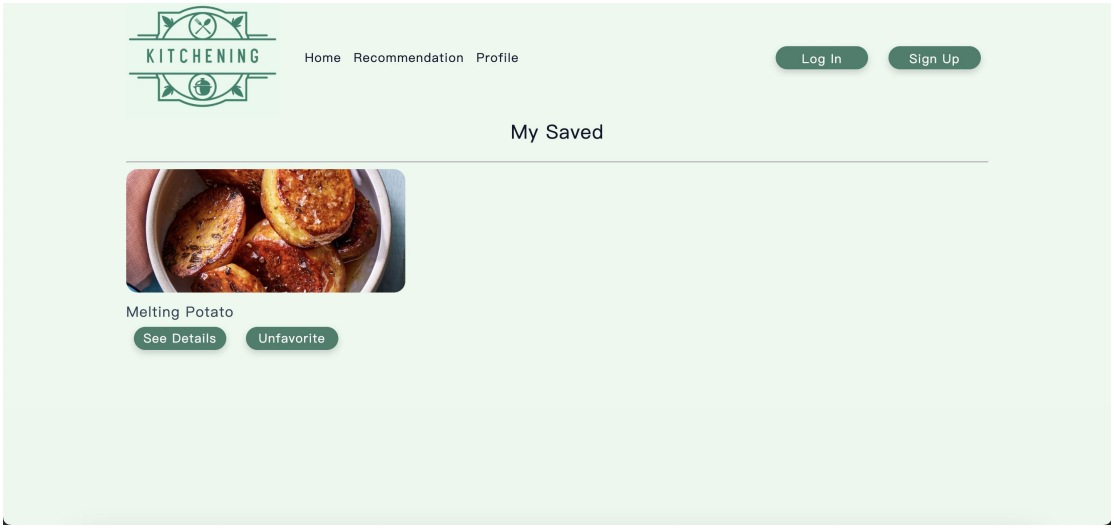
5. In profile page, user can get into subscribe page, which they can check the subscribe list and get the new recipes from the creators who be followed.



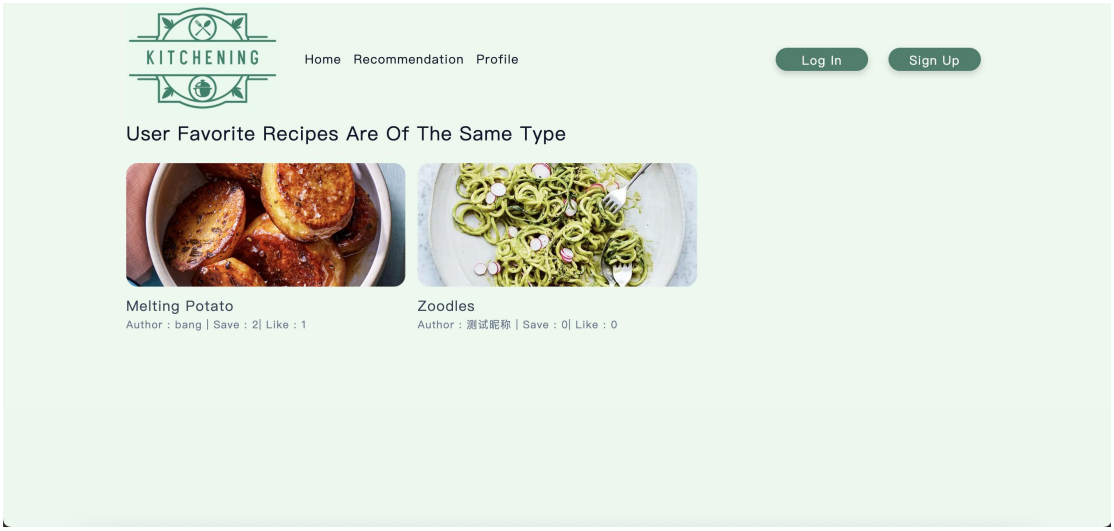
6. In profile page, user can also get into comment page to check and mange what they commented.



7. User can also check and manage the save recipes in save page, which also get into from profile page.



8. In home page, user can also get into a recommendation page, which is based on the like and save.



6 Reference

JavaScript official website (https://www.javascript.com/)
Bootstrap official website (https://getbootstrap.com/)
Python official website (https://www.python.org/)
Python Json module official website (https://docs.python.org/3/library/json.html)
Flask official website (https://flask.palletsprojects.com/en/2.2.x/)
XAMPP official website (https://www.apachefriends.org/)
MySQL official website (https://www.mysql.com/)