**205 CDE: Developing the Modern Web**

**Reflective Report**

**Name of the website: Mud Ye Blog**

**GitHub URL: \_** [**https://github.com/OscarLeong19/205-CDE.git**](https://github.com/OscarLeong19/205-CDE.git) **\_**

**Backup Link: \_\_\_\_\_\_\_\_\_**

**YouTube Video Link: \_\_\_\_\_\_\_\_\_**

**Student ID: 56235350**

**Student Name: Leong Ho Yin**

**Content**

Introduction1

What languages used in the web2

Structure3

Feature and function of Mud Ye Blog

Code

Reference

**Introduction**

What is Mud Ye Shop?

Mud Ye Shop is a real online store created by a teenage couple, the shop provided various products with brands, such as Dickies, BenDavis, etc. You can likewise discover some Disney gadgets in our shop. As the slogan of our shop is “What are we selling for? We sell anything you want!” Thus, we hope that we can sell more different types of products like clothes, shoes, accessories, foods in the future, to coordinate with every customer needs.

Why do we create Mud Ye Blog?

Mud Ye Blog is extended from Mud Ye Shop. And at the time of running Mud Ye Shop, we want to provide a platform for customers to share their shopping experience in Mud Ye Shop and express their opinions on the quality of products and the service of our shop. More importantly, our purpose is to make improvement and progression in the future through customers’ opinions and be understandable to customer's feelings after they have shopping in our shop.

**What languages used in the website?**

**HTML**

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

</head>

(layout.html)

**CSS**

body {

    background: #ffffff;

    color: #000000;

    font-family: 'Merriweather Sans', sans-serif;

  }

  header {

    display: flex;

    justify-content: space-between;

    align-items: center;

    padding: 10px 10%;

    background: #24252A;

  }

(main.css)

**Python Flask**

from flask import Flask

(\_\_int\_\_ py)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

(hwhw.py)

**JavaScript**

    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>

**Bootstrap**

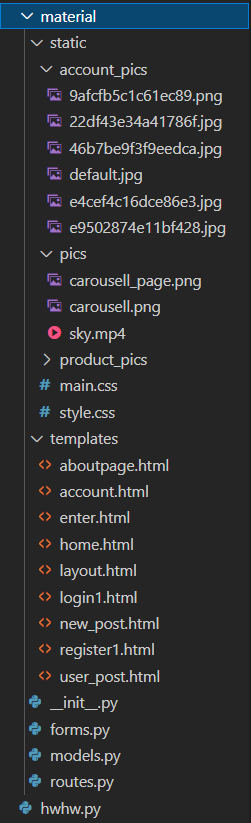
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

(layout.html)

MySQL

INSET INTO `product` (`product\_id`, `product\_img`) VALUES (`10`, `threeeyes\_umbrella.jpg`)

(In phpMyAdmin database)

**Structure**

Here is the structure of the whole website. I have created multiple folders to put same kind of files together which can make the overall structure more perfect and allow me to find some of the files more easily.

*(Figure 1: The files of Mud Ye Blog)*

**Code**

1 from material import app

2

3 if \_\_name\_\_ == '\_\_main\_\_':

4    app.run(debug=True)

*(code in* ***hwhw.py****)*

This is the code in **hwhw.py**. As we separated the files. So, the first line is used to import the ‘app’ from the **\_\_int\_\_.py** file in **material** folder. For line 4, app.run is used to run the server. And the debug=True is used to find out what and where is the error and restart if the app is changed.

1 from flask import Flask

2 from flask\_sqlalchemy import SQLAlchemy

3 from flask\_bcrypt import Bcrypt

4 from flask\_login import LoginManager

5

6 app = Flask(\_\_name\_\_)

7 app.config['SECRET\_KEY'] = '451002d5999c0e9cd687a4c0a1f92e5c'

8 app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql+pymysql://oscar:205CDE@ localhost/dbflask'

9

10

11 db = SQLAlchemy(app)

*(code in* ***\_\_int\_\_.py****)*

This is the code in **\_\_int\_\_.py**. As we are using Flask in our website, we import Flask in the first line. In line 6, we are creating an instance of class and name as \_\_name\_\_, it allows the server know the position of the application. For the secret key in line 7 is used to sign session cookies for protection against cookie data getting change. For SQLAlchemy, this is a function that can change the python code to the SQL command and put it in the database. In line 8, is how I connect to my database is Linux.

 **Enter page**

Carousell icon

*(Figure 2: Enter page of Mud Ye Blog)*

This is the **enter page**(enter.html) of Mud Ye Blog, on the right-hand corner, we can see a icon which linked to the Carousell page of Mud Ye Shop.

<a href="{{ url\_for('enter') }}" class="logo">MudYeBlog</a>

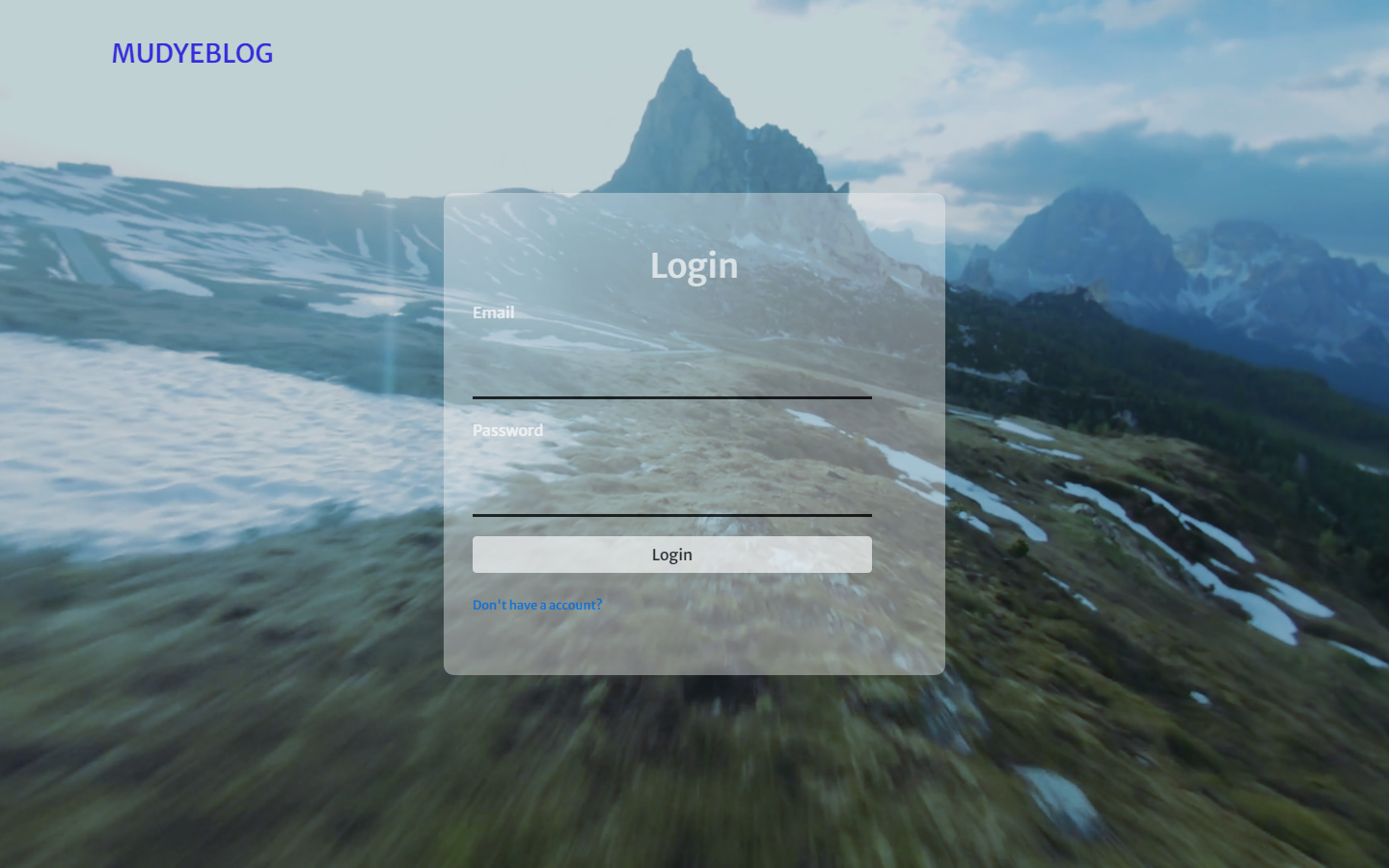
On the left-hand corner, I used a hyperlink, which linked to the enter.html. That’s means users clicked in, they can refresh the page.

<div><a href="https://www.carousell.com.hk/mud\_ye\_shop/">

<img src="{{ url\_for('static', filename='pics/carousell.png') }}">

</a></div>

And this is the right-hand corner image that are linked to the MudYeShop Carousell page.

**Login system**

Don’t have an account

Email & Password

*(Figure 3: Login page of Mud Ye Blog)*

Here is the **login page**(login1.html), after user clicked the ‘Explore’ button, system will lead user to login before they enter the blog page. They have to type in email and password to login. If users have no account, they can click the ‘[Don't have an account?](http://127.0.0.1:5000/register1)’ button to create.

31     {% with messages = get\_flashed\_messages(with\_categories=true) %}

32      {% if messages %}

33          {% for category, message in messages %}

34          <div class="alert alert-{{ category }}">

35              {{ message }}

36          </div>

37          {% endfor %}

38      {% endif %}

39      {% endwith %}

(*code in* ***login1.html***)

This part is to show the alert of message, telling the users that they are entering wrong password or email.

44 {{ form.email.label() }}

45              {% if form.email.errors %}

46                  {{ form.email() }}

47                  <div class="invalid-feedback">

48                      {% for error in form.email.errors %}

49                          <span>{{ error }}</span>

50                      {% endfor %}

51                  </div>

52              {% else %}

53                  {{ form.email() }}

54              {% endif %}

(*code in* ***login1.html***)

This code is the email entering bar and error handling, if the email has errors on it, it will print out the from field and the error message. And if there have no errors, it will just print out what there have before. Of course, the form of password is using the same way to do.

@app.route("/login1", methods=['GET', 'POST'])

def login1():

    if current\_user.is\_authenticated:

        return redirect(url\_for('home'))

    form = LoginForm()

    if form.validate\_on\_submit():

        user = User.query.filter\_by(email=form.email.data).first()

        if user and bcrypt.check\_password\_hash(user.password, form.password.data):

            login\_user(user)

            next\_page = request.args.get('next')

            flash('Welcome to Mud Ye Blog!', 'warning')

            return redirect(next\_page) if next\_page else redirect(url\_for('home'))

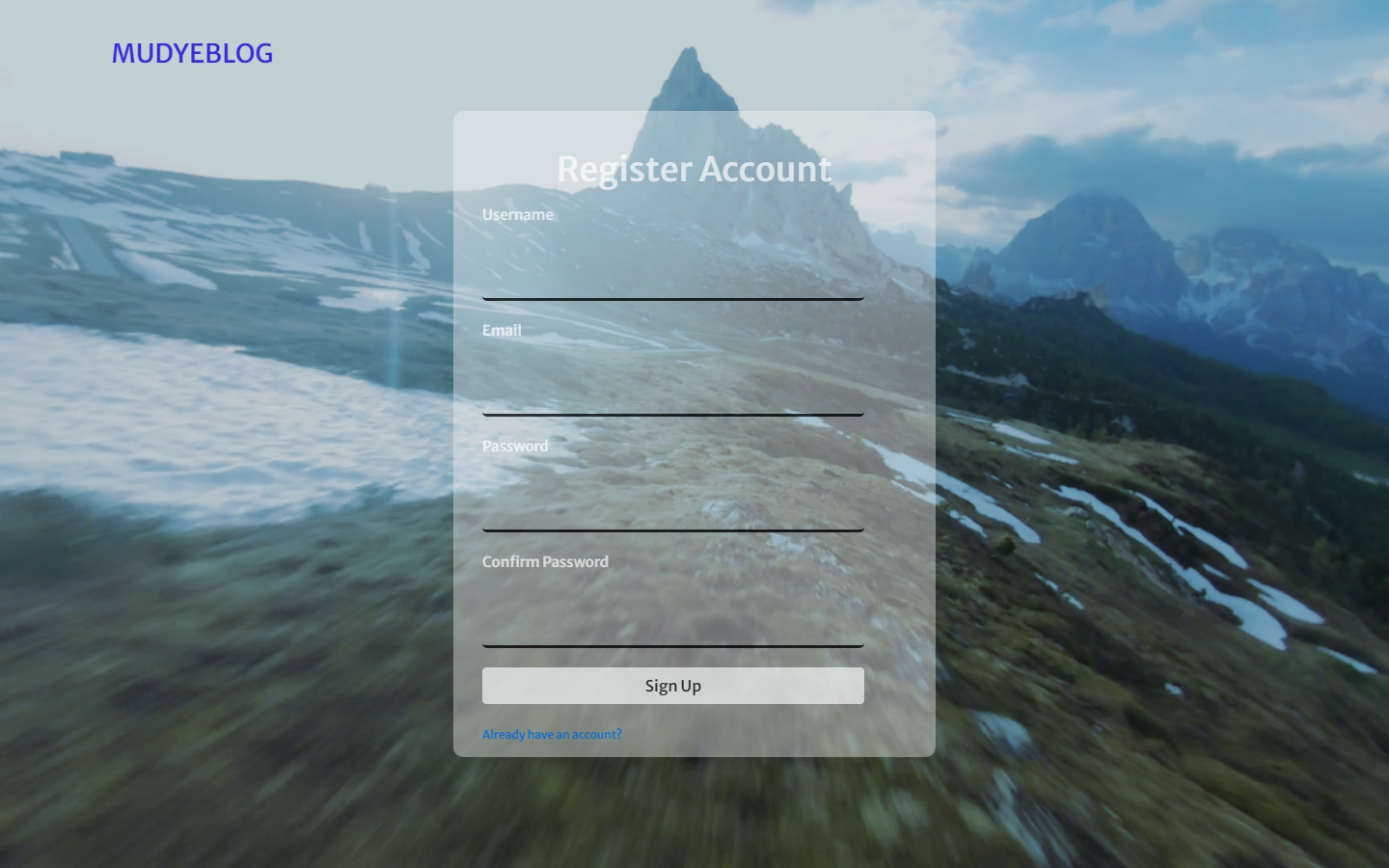
        else:

            flash('Wrong email or password! Please type again!', 'danger')

    return render\_template('login1.html', title='Login', form=form)

(*login part in* ***routes.py***)

To the login function, if user is already logged in the system will bring user to the **home page**. If not, it will render out the login page for user to login. And if user enter correct email and password and submit, system will redirect to the **home page**. Otherwise, it will show a alert message that telling user that they are entering wrong email or password and tell them type again.

**Register system**

Password & Confirm Password

Username & Email

*(Figure 3: Register page of Mud Ye Blog)*

Here is the **register page**(register1.html), if user have no account in the system, they can register a new account here. Users must type in their Username, email, password and confirm password to create an account.

45 @app.route("/register1", methods=['GET', 'POST'])

46 def register1():

47     if current\_user.is\_authenticated:

48         return redirect(url\_for('home'))

49     form = RegistrationForm()

50     if form.validate\_on\_submit():

51         hashed\_password = bcrypt.generate\_password\_hash(form.password.data).decode('utf-8')

52         user = User(username=form.username.data, email=form.email.data, password=hashed\_password)

53         db.session.add(user)

54         db.session.commit()

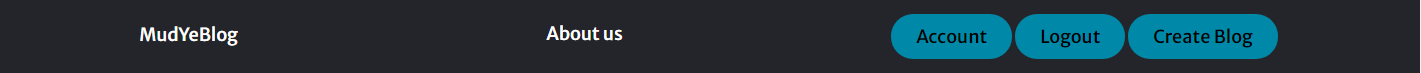
55        flash('Lets login!', 'primary')

56         return redirect(url\_for('login1'))

57     return render\_template('register1.html', title='Register Account', form=form)

(*register part in* ***routes.py****)*

To the register function, also, if user is already logged in the system will bring user to the **home page**. If not, user can create an account here in the form. For line 51, after user entered the password, the system will hash the password and we are using decode('utf-8') as we want a string. When user submit, system will add this data (username, email and password) into the database. Then, system will bring user to the login page to login using their new created account. The reason why I hash the password, it is because if someone want to steal password from the password, they would not be able to use hash table to crack the password, so it could be more secure.

**Layout**

The nav bar and the photo slider (Carousel[[1]](#footnote-1)) (made by Bootstrap) is the layout template (after logged in). This is the function in Flask called ‘Template Inheritance’[[2]](#footnote-2). It gets all template that contains comment elements and then defines the block content that can override. If we want to extend this template(layout.html) to another template, we have to enter **{% extends "layout.html" %}** to the first tag in other template.

Like the example below:

1 {% extends "layout.html" %}

2 {% block content %}

{% endblock content %}

(*code in* ***home.html*)**

I the nav bar, it contains the refresh button (MudYeBlog on left hand side), about us button linked to the **about page** and the user account button (**Account**), Logout button **(Logout**) and create blog button (**Create Blog**). When you clicked the logout button, system will bring you back to the **enter page**.

@app.route("/logout")

def logout():

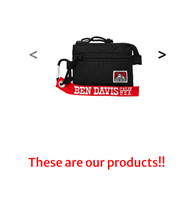
    logout\_user()

    return redirect(url\_for('enter'))

(*logout part in* ***routes.py***)

This is the function of Logout button. It allows user to logout from their account and the system will redirect them to the **enter page**.

On the right-hand side, there is a photo slider (Carousel), users can click the left or right arrow to see different products that are selling on Mud Ye Shop. (The slider will also slide automatically)



Carousel slideshow

56 <div id="carouselExampleIndicators" class="carousel slide col-md-4" style="width: 500px; height: 400px; data-ride="carousel">

57        <div class="carousel-inner">

58          <div class="carousel-item active">

59            <img class="d-block w-100" src="{{ url\_for('static', filename='product\_pics/BENDAVIS\_WALLET\_RED.jpg') }}" alt="First slide">

60          </div>

.

.

.

89 <a class="carousel-control-prev" style="color: black;" href="#carouselExampleIndicators" role="button" data-slide="prev">

90          <i class="fas fa-arrow-left" aria-hidden="true"></i>

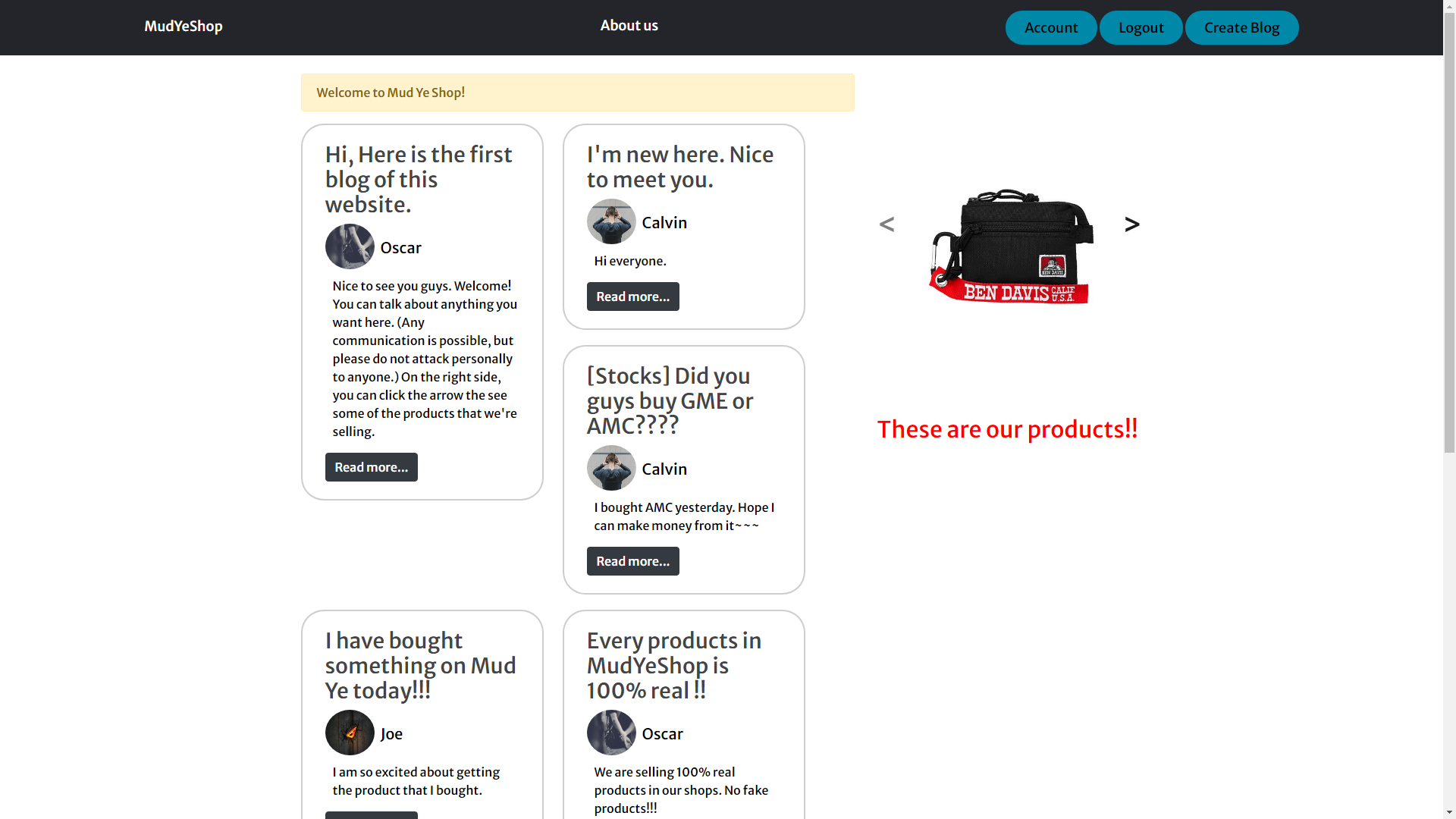
91          <span class="sr-only">Previous</span></a>

92        <a class="carousel-control-next" href="#carouselExampleIndicators" role="button" data-slide="next">

93          <i class="fas fa-arrow-right" aria-hidden="true"></i>

94          <span class="sr-only">Next</span></a>

This is one of the photos in the Carousel photo slider, line 89-94, I creating the left and right arrow which can slide to another photo when user clicked

**Home Page**

Blogs

*(Figure 4: Home page of Mud Ye Blog)*

**Home Page**(home.html) of Mud Ye Blog. In this page, users can browse many blogs by different users.

<div class="card-body">

       <h3><a class="card-title">{{ post.title }}</a></h3>

       <img class="rounded-circle article-img" style="width: 65px; height: 60px; margin-bottom: 10px;" src="{{ url\_for('static', filename='account\_pics/' + post.author.image\_file) }}">

       <a class="card-author">{{ post.author.username }}</a>

       <p class="card-content">{{ post.content }}</p>

       <a href="{{ url\_for('post', post\_id=post.id) }}" class="btn btn-dark">Read more...</a>

</div>

(*code in* ***home.py***)

This is the code of each blog. And the blog exterior is made by ‘Bootstrap-card[[3]](#footnote-3)’.

17 @app.route("/home")

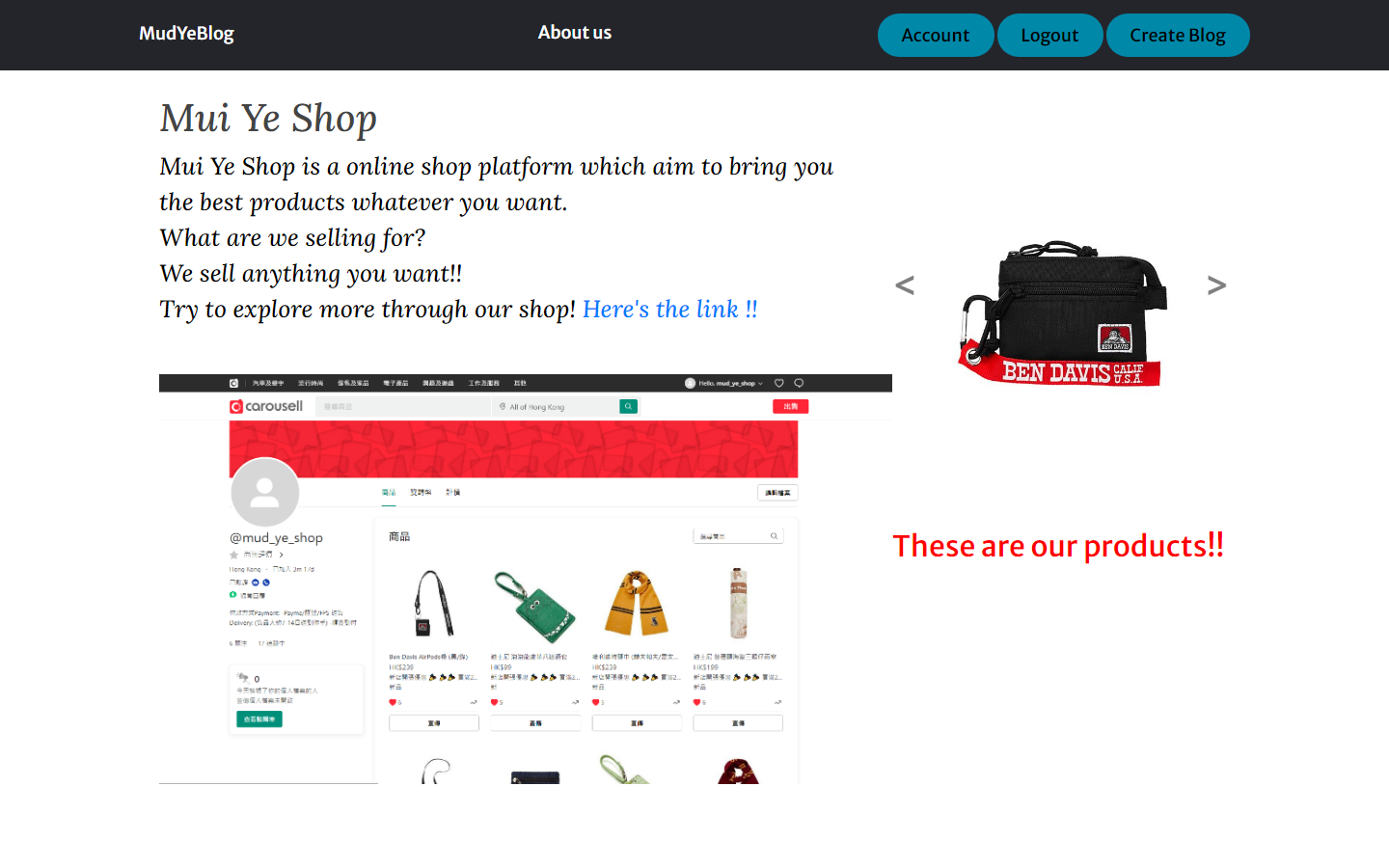
18 def home():

19     posts = Post.query.all()

20     return render\_template('home.html', posts=posts)

(*home part in* ***routes.py***)

This is the function of home page. The function named ‘home’. In line 19, the posts in home page are already in the database. So we need to use Post.query.all() to find it out in the database.

**About page**

*(Figure 5: About page of Mud Ye Blog)*

**About page**(aboutpage.html) of Mud Ye Blog. In this page, here introduced about Mud Ye Shop. Under the introduction, there is a picture of the Mud Ye Shop in Carousell.com.

**Update your account**

Emial &

Username &

User Image

*(Figure 6: Account page of a user)*

**Account page(**account.html**)** of the user ‘Oscar’. (Logged in with the account Oscar) In this page, user can update their information, such as email address, username and their profile image.

66 @app.route("/account", methods=['GET', 'POST'])

67 @login\_required

68 def account():

69     form = UpdateAccountForm()

70     if form.validate\_on\_submit():

71         if form.picture.data:

72             picture\_file = save\_picture(form.picture.data)

73             current\_user.image\_file = picture\_file

74         current\_user.username = form.username.data

75         current\_user.email = form.email.data

76         db.session.commit()

77        flash('Account updated!', 'primary')

78         return redirect(url\_for('account'))

79    elif request.method == 'GET':

80         form.username.data = current\_user.username

81         form.email.data = current\_user.email

82     image\_file = url\_for('static', filename='account\_pics/' + current\_user.image\_file)

83     return render\_template('account.html', title='Your Account', image\_file=image\_file, form=form)

84

85 def save\_picture(form\_picture):

86     random\_hex = secrets.token\_hex(8)

87     \_, f\_ext = os.path.splitext(form\_picture.filename)

88     picture\_fn = random\_hex + f\_ext

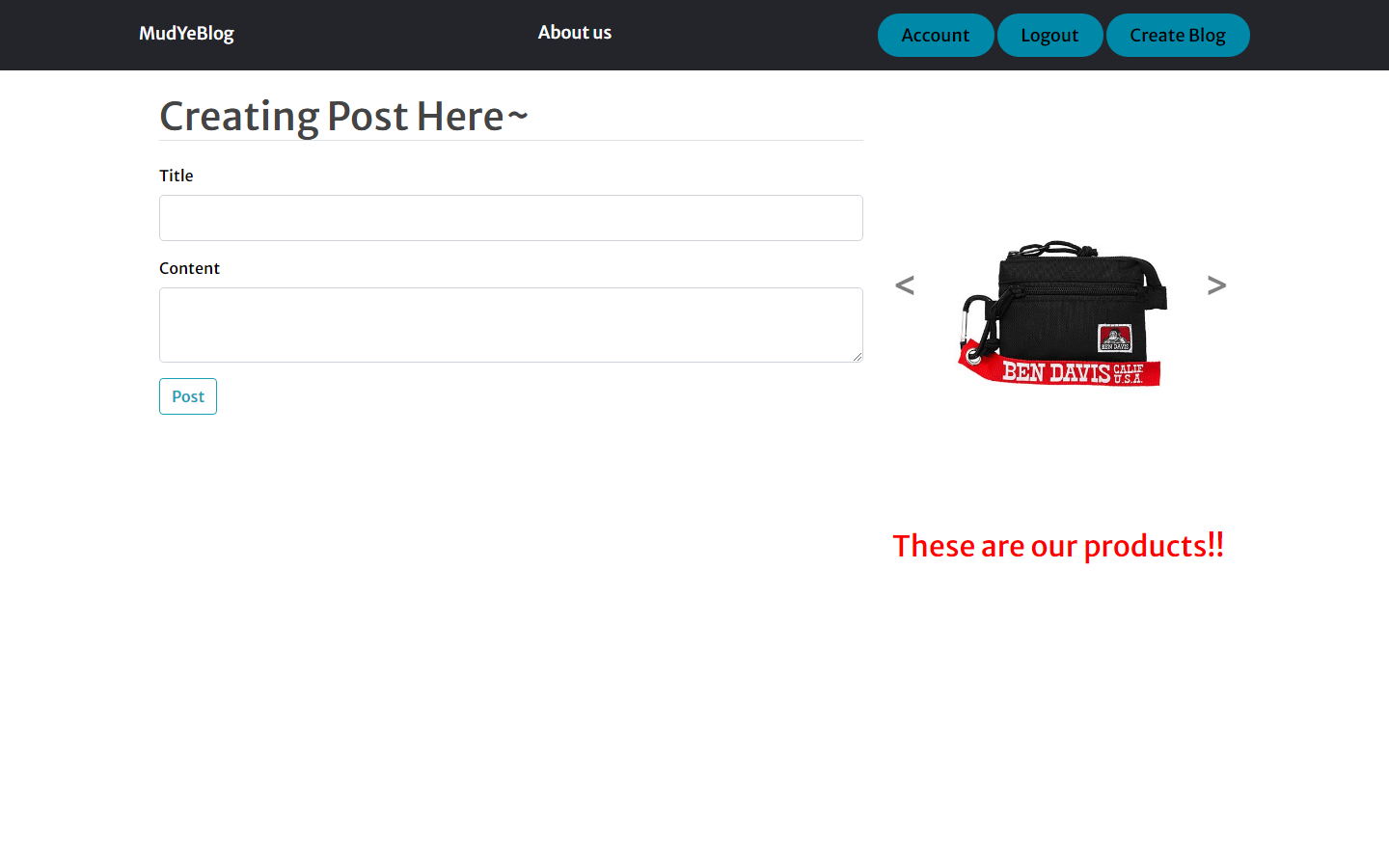
89     picture\_path = os.path.join(app.root\_path, 'static/account\_pics', picture\_fn)

80     form\_picture.save(picture\_path)

90     return picture\_fn

(*account part in* ***routes.py***)

This is the update account function, at first, system will get the data from database of current user and paste in the form (include username, email and profile image). And if the user changes some data in the form and submit, system will send it back to database and cover the old data to finish the updating.

**Create your new post / blog**

*(Figure 7: Create post page)*

This is the create blog page(**new\_post.html**), user can create a new post with typing the title and content. When they press the Post button, their will see the new post in home page(**home.html**).

@app.route("/post/newpost", methods=['GET', 'POST'])

@login\_required

def create\_post():

    form = PostForm()

    if form.validate\_on\_submit():

        post = Post(title=form.title.data, content=form.content.data, author=current\_user)

        db.session.add(post)

        db.session.commit()

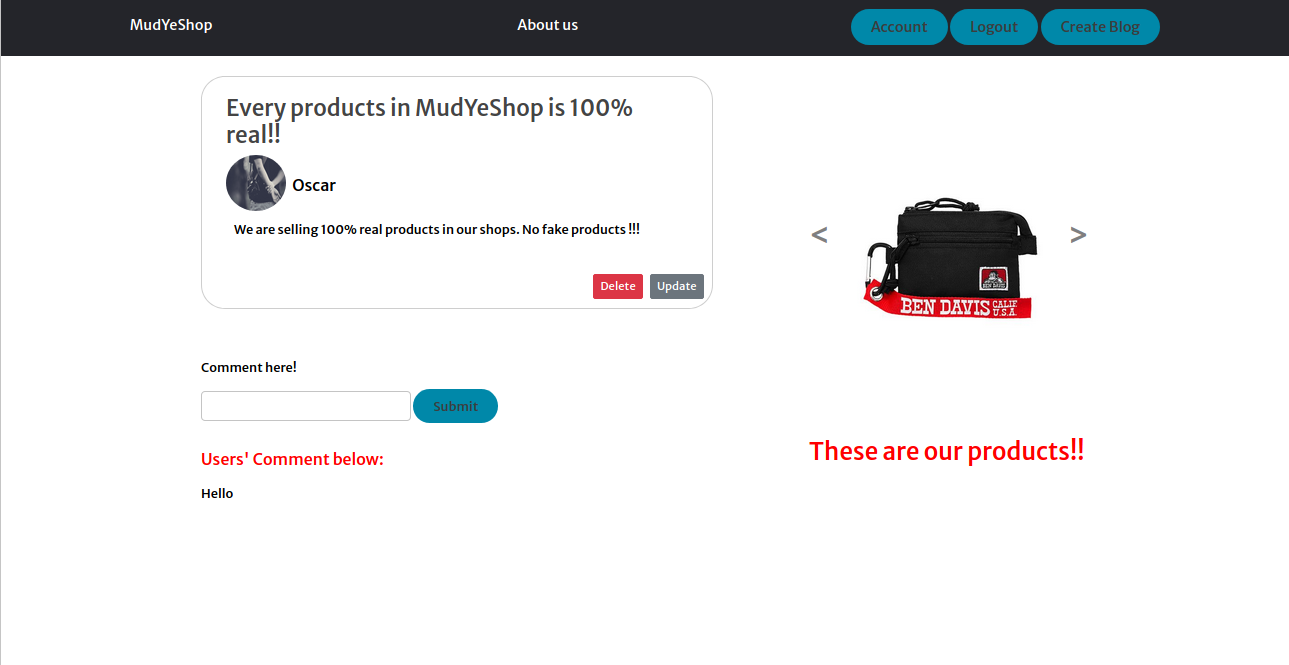
        flash('Done! Post has been created.', 'primary')

        return redirect(url\_for('home'))

    return render\_template('new\_post.html', title='Create Post', form=form, legend="Creating Post Here~")

(*create post part in* ***routes.py***)

This is the function of create post page, user must log in before they create post/blog. User is asked to enter the title and content of the post/blog. When they submit it, system will add these data to the post table in database and it will redirect users to the **home page** and show a alert that the post has been created.

**Update or Delete your blog**

*(Figure 6: Update or delete post page)*

If users what to read more about that post/blog, they can click the ‘Read me’ button and go to the post/blog page. In this page, users can see full information and the comments of the post/blog. If this is the user own post/blog, they have the right to delete it and update it.

4   <div class="card" style="width: 40rem; float: left; ">

5      <div class="card-body">

6           <h3><a class="card-title">{{ post.title }}</a></h3>

7           <img class="rounded-circle article-img" style="width: 75px; height: 70px; margin-bottom: 10px;" src="{{ url\_for('static', filename='account\_pics/' + post.author.image\_file) }}">

8           <a class="card-author">{{ post.author.username }}</a>

9           <p class="card-content">{{ post.content }}</p>

10      </div>

11      {% if post.author == current\_user %}

12      <div>

13        <a class="btn btn-secondary btn-sm mt-2 mb-2" style="float: right;" href="{{ url\_for('update\_post', post\_id=post.id) }}">Update</a>

14        <button type="button" class="btn btn-danger btn-sm m-2" style="float: right;"  data-toggle="modal" data-target="#deleteModal">Delete</button>

15      </div>

16      {% endif %}

    </div>

(*code in* ***user\_post.html***)

This is the code of this post/blog page, line 11 mentioned that, if the user now logged in is the post author, it will show the ‘Update’ and ‘Delete’ button under the content of the post/blog.

<div class="modal fade" id="deleteModal" tabindex="-1" role="dialog" aria-labelledby="deleteModalLabel" aria-hidden="true">

    <div class="modal-dialog modal-dialog-centered" role="document">

      <div class="modal-content">

        <div class="modal-header">

          <h5 class="modal-title" id="deleteModalLabel">Are you sure you want to delete this Post?</h5>

          <button type="button" class="close" data-dismiss="modal" aria-label="Close">

            <span aria-hidden="true">&times;</span>

          </button>

        </div>

        <div class="modal-footer">

          <button type="button" class="btn btn-secondary" data-dismiss="modal">No</button>

          <form action="{{ url\_for('delete\_post', post\_id=post.id) }}" method="POST">

            <input class="btn btn-danger" type="submit" value="Delete">

          </form>

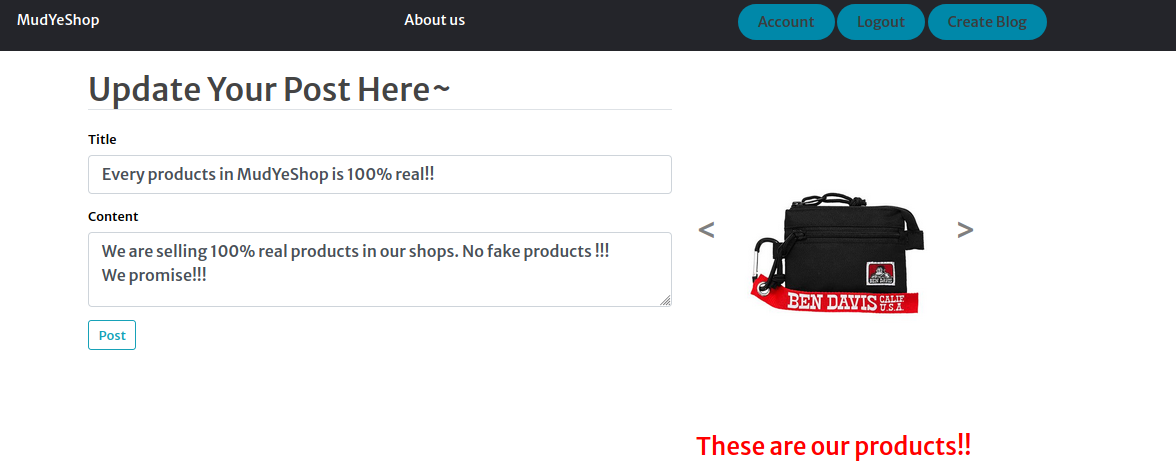
        </div>

      </div>

    </div>

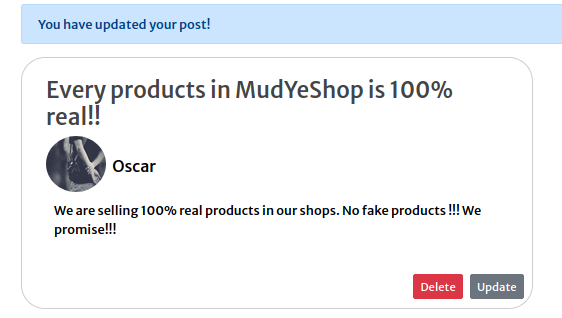
  </div>

For the ‘Delete’ button, I used Bootstrap – Modal[[4]](#footnote-4) to make it look nicer. After user clicked to ‘Delete’ button, there are a window appeared and ask if they are sure to delete the post/blog. Allow user to make a double confirm.

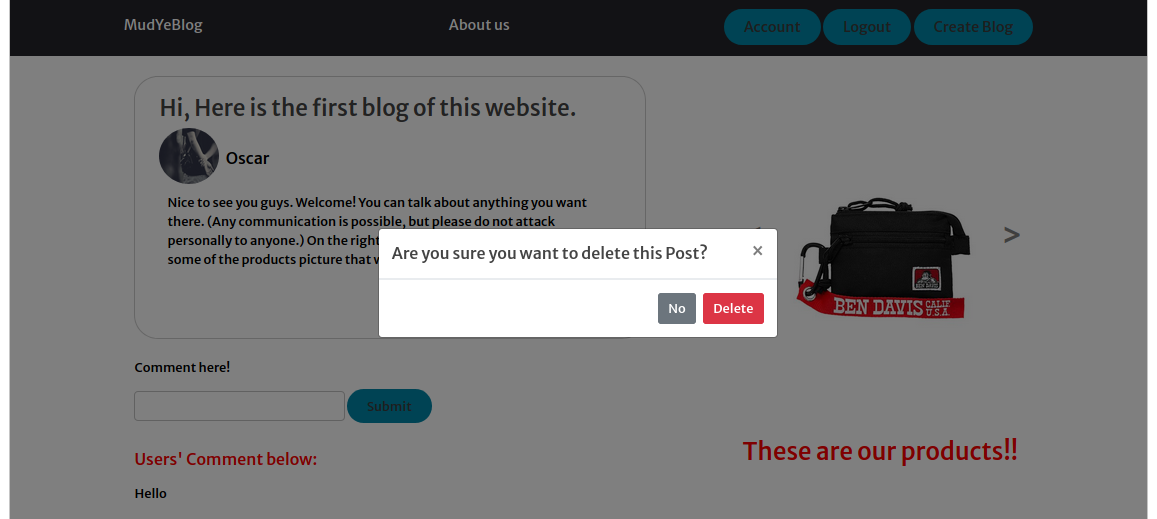


Information that wants to update

If users want to update the post/blog, they press ‘Update’, then system will show the form which is similar to the form that they used to create the post/blog. After they finished editing, press ‘Post’ again. And the post/blog will be updated and also show an alert saying that they have updated their own post/blog.



Alert message



Double confirm

122 @app.route("/post/<int:post\_id>/updatepost", methods=['GET', 'POST'])

123 @login\_required

124 def update\_post(post\_id):

125     post = Post.query.get\_or\_404(post\_id)

126     if post.author != current\_user:

127         abort(403)

128     form = PostForm()

129     if form.validate\_on\_submit():

130         post.title = form.title.data

131         post.content = form.content.data

132         db.session.commit()

133        flash('Post updated!', 'primary')

134         return redirect(url\_for('post', post\_id=post.id))

135     elif request.method == 'GET':

136         form.title.data = post.title

137         form.content.data = post.content

138     return render\_template('new\_post.html', title='Update', form=form, legend="Update Your Post Here~")

(*update and delete post part in* ***routes.py***)

For the update function, we need to get every data of a post by a post\_id, if there is no that post\_id, it will go to the 404Not found page. Then, system will check the current user if it is a post author. If not, it will show up a 403Forbidden error. If yes, the user can change the title and content of the post/blog in the form. After submit, system will pass these data to database and change it. If it is only a ‘GET’ method, system will only take the data of that post/blog from the database.

140 @app.route("/post/<int:post\_id>/deletepost", methods=['POST'])

141 @login\_required

142 def delete\_post(post\_id):

143     post = Post.query.get\_or\_404(post\_id)

144    if post.author != current\_user:

145         abort(403)

146     db.session.delete(post)

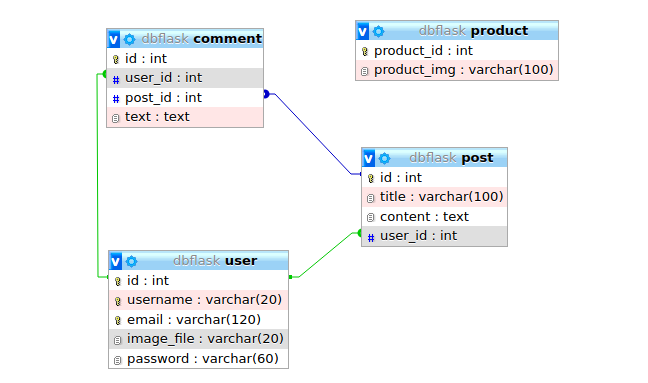
147     db.session.commit()

148     flash('Post deleted!', 'primary')

149     return redirect(url\_for('home'))

(*update and delete post part in* ***routes.py***)

For the delete function, we also need to get every data of a post by a post\_id and the line 146,147 means that system will delete the data of the post with that post\_id. Then, system will redirect user back to the home page and also give a alert message saying that the post is deleted.

**Database**

This is the database of the website. It contains post, user, product, and comment. We can see that there is a line connecting user, post, and comment as they have a relationship. The comment will be saved with a user ID and the post ID as this is the comment of one post and it is commented by one user, the post will be saved with the user ID and the comment will be saved with user ID too. But the product is an independent table, so there are no lines are linked to another table.

Reference

Flask QuickStart

[https://flask.palletsprojects.com/en/1.1.x/quickstart/#](https://flask.palletsprojects.com/en/1.1.x/quickstart/)

Python Flask Tutorial

<https://www.youtube.com/watch?v=MwZwr5Tvyxo>

W3schools – HTML

<https://www.w3schools.com/html/default.asp>

Flask - Model

<https://flask-appbuilder.readthedocs.io/en/latest/quickhowto.html>

404 403 ERROR

<https://flask.palletsprojects.com/en/1.1.x/patterns/errorpages/>

1. Bootstrap - Carousel

   <https://getbootstrap.com/docs/4.0/components/carousel/> [↑](#footnote-ref-1)
2. Flask – Template Inheritance

   <https://flask.palletsprojects.com/en/1.1.x/patterns/templateinheritance/#template-inheritance> [↑](#footnote-ref-2)
3. Bootstrap - Card

   <https://getbootstrap.com/docs/4.0/components/card/> [↑](#footnote-ref-3)
4. Bootstrap – Modal

   <https://getbootstrap.com/docs/4.0/components/modal/> [↑](#footnote-ref-4)