Flask
레포트

김호권

목 차

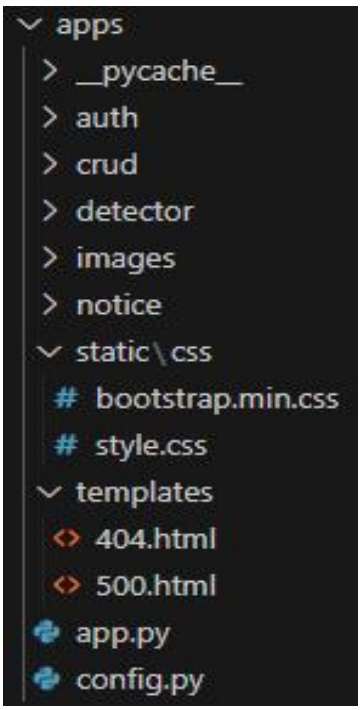
1. Flask란

플라스크는 파이썬으로 만든 마이크로 웹 프레임워크이다. 기본적으로 데이터베이스 기능이 포함되어 있지 않는 등 최소한의 기능만 제공하여 앱 구성을 자유롭게 결정할 수 있으며 확장 기능을 많이 지원하는 장점이 있다.

이번에는 플라스크를 이용하여 로그인, 공지사항, 사용자명단, 메인화면, 사진 감지 기능 게시판을 만들어서 하나의 웹페이지를 만드려고 한다.(이용하는 파이썬은 3.11.5이다.)

2. 파일 구성

2-1. 총괄 파일



1. static/style.css: 웹 애플리케이션의 스타일을 정의하는 CSS 파일.

2. static/bootstrap.min.css: 부트스트랩 프레임워크에서 제공하는 CSS 파일로, 웹 애플리케이션의 디자인을 향상시키는 데 사용.

3. templates/404.html: 404 오류가 나왔을 경우 보여주는 페이지의 HTML 템플릿.

4. templates/500.html: 500 오류가 나왔을 경우 보여주는 페이지의 HTML 템플릿.

5. app.py: Flask 애플리케이션의 핵심 파일로, 웹 애플리케이션의 라우팅 및 뷰 로직을 정의.

6. config.py: Flask 애플리케이션의 설정을 정의하는 파일로, 데이터베이스 연결 정보, 시크릿 키 및 기타 구성 옵션을 설정.

2-1-1. 코드 상세 정보

```html
<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="UTF-8" />
    <title>404 Not Found</title>
</head>

<body>
    <h1>404 Not Found</h1>
    <p><a href="/">애플리케이션 톱으로</a></p>
</body>

</html>
```

```html
<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="UTF-8" />
    <title>500 Internal Server Error</title>
</head>

<body>
    <h1>500 Internal Server Error</h1>
    <p><a href="/">애플리케이션 톱으로</a></p>
</body>

</html>
```

404.html와 505.html의 코드로 404에러, 505에러가 나오면 각 오류의 이름과 함께 톱 페이지로 넘어갈수 있는 하이퍼링크가 걸려있는 '애플리케이션 톱으로'라는 글씨가 나오게 된다.

```python
from pathlib import Path
from flask import Flask,render_template
from flask_migrate import Migrate
from flask_sqlalchemy import SQLAlchemy
from flask_wtf.csrf import CSRFProtect
from flask_login import LoginManager

from apps.config import config

db = SQLAlchemy()
csrf = CSRFProtect()
login_manager = LoginManager()
login_manager.login_view = "auth.login"
login_manager.login_message = ""

def create_app(config_key):

    app = Flask(__name__)
    app.config.from_object(config[config_key])

    csrf.init_app(app)

    db.init_app(app)
    Migrate(app, db)
    login_manager.init_app(app)

    from apps.crud import views as crud_views
    app.register_blueprint(crud_views.crud, url_prefix="/crud")
    from apps.auth import views as auth_views
    app.register_blueprint(auth_views.auth, url_prefix="/auth")
    from apps.detector import views as dt_views
    app.register_blueprint(dt_views.dt)
    from apps.notice import views as notice_views
    app.register_blueprint(notice_views.notice, url_prefix="/notice")
    app.register_error_handler(404, page_not_found)
    return app


def page_not_found(e):
    return render_template("404.html"),404
def internal_server_error(e):
    return render_template("500.html"),500
```

app.py

필수 모듈 임포트: pathlib, Flask, render_template, SQLAlchemy, CSRFProtect, LoginManager, Migrate 등

create_app 함수를 통해 애플리케이션을 생성하고, 주어진 구성키를 기반으로 애플리케이션을 설정.

보안 기능 설정: CSRF 보호 및 로그인 관리자 등의 보안 기능을 설정하고 초기화.
Blueprint 등록: 각각의 기능을 담당하는 Blueprint를 등록하고 URL에 매핑.

오류 핸들러 등록: 404 오류와 500 오류를 처리하는 커스텀 오류 핸들러를 등록.

create_app 함수는 주어진 구성키를 기반으로 Flask 애플리케이션을 설정하여 반환하고, SQLAlchemy, CSRF 보호, 로그인 관리자 등의 확장을 초기화하고 설정.

Blueprint를 사용하여 각 기능을 담당하는 라우팅을 설정하고 /crud, /auth, /dt, /notice 등의 URL에 대한 라우팅을 설정한다.

커스텀 오류 핸들러를 등록하여 404 오류와 500 오류를 처리한다.

```python
from pathlib import Path

basedir = Path(__file__).parent.parent

class BaseConfig:
    SECRET_KEY = "2AZSMss3p5QPbcY2hBsJ"
    WTF_CSRF_SECRET_KEY = "AuqcyzslUSsqQN7YZs6f"
    UPLOAD_FOLDER = str(Path(basedir, "apps", "images"))
    LABELS = ["unlabeled","person","bicycle","car","motorcycle","airplane","bus","train","truck","boat","traffic light","fire hydrant",
        "street sign","stop sign","parking meter","bench","bird","cat","dog","horse","sheep","cow","elephant","bear","zebra","giraffe",
        "hat","backpack","umbrella","shoe"]
class LocalConfig(BaseConfig):
    SQLALCHEMY_DATABASE_URI = f"sqlite:////{basedir / 'local.sqlite'}"
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    SQLALCHEMY_ECHO = True
class TestingConfig(BaseConfig):
    SQLALCHEMY_DATABASE_URI = f"sqlite:////{basedir / 'testing.sqlite'}"
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    WTF_CSRF_ENABLED = False
config = {
    "testing": TestingConfig,
    "local": LocalConfig,
}
```

config.py

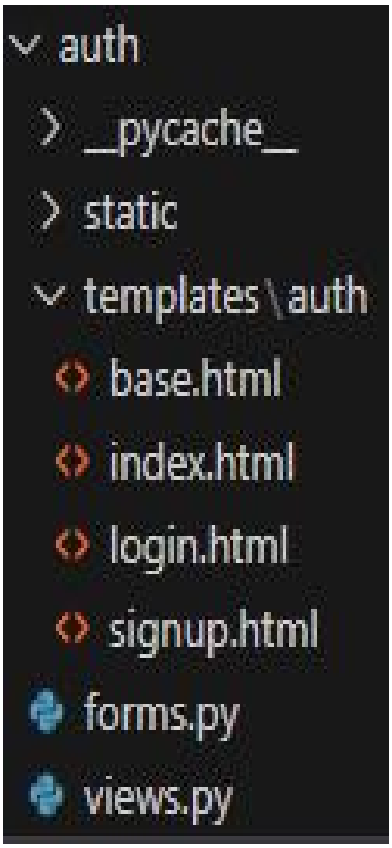
애플리케이션과 CSRF토큰 생성에 사용되는 시크릿 키를 설정.

이미지 업로드를 위한 폴더 경로를 설정하고 인식 가능한 라벨들을 리스트로 정의.

로컬 데이터베이스의 URI를 설정하고 SQLAlchemy의 모델 변경 내역 추적을 비활성화하고 쿼리를 콘솔에 출력.

테스트 데이터베이스의 URI를 설정하고 CSRF 보호를 비활성화.

2-2. 로그인



1. base.html: 기본 레이아웃을 정의하는 파일로 모든 페이지의 공통적인 요소들을 포함.

2. index.html: 홈페이지를 정의하는 파일로 접속할 때 보여지는 첫 번째 페이지.

3. login.html: 사용자가 로그인하는데 사용되는 파일이며 로그인 폼과 사용자 이름과 비밀번호를 입력하는 필드를 제공.

4. signup.html: 사용자가 사용자가입하는데 사용되는 파일이며 사용자가입 폼과 가입할 때 필요한 정보를 입력하는 필드를 제공.

5. forms.py: 사용자가 로그인할 때 필요한 정보를 입력하는 폼 파일이며 사용자 입력의 유효성을 검사하는 파일.

6. views.py: 로그인을 담당하는 컨트롤러 파일이며 사용자의 요청을 처리하고 제공한 로그인 정보를 확인하여 인증을 수행하는 파일로 로그인 정보가 올바른지 검증하고 성공하면 로그인 세션을, 실패하면 오류 메시지를 반환하여 사용자에게 알리는 역할.

2-2-1. 코드 상세 정보

```html
<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="UTF-8" />
    <title>{% block title %}{% endblock %}</title>
</head>

<body>
    {% block content %}{% endblock %}
</body>

</html>
```

auth/base.html

```
{% extends "auth/base.html" %}
{% block title %}인증 페이지{% endblock %}
{%block content %} 인증 페이지 표시 확인 {% endblock %}
```

auth/index.html

base.html은 기본 틀을, index.html은 사용자에게 표시되는 페이지를 정의하는 코드.
index에서 extends를 통해 base.html에서 정의한 기본 틀을 그대로 가져오면서 내용을 추가 할 수 있다.

```
{% extends "detector/base.html" %}

{% block title %}로그인{% endblock %}

{% block content %}
<div class="mx-auto dt-auth-main">
    <div class="card dt-auth-login">
        <header>로그인</header>
        <section>
            <form method="post" action="{{ url_for('auth.login') }}" onsubmit="return submit();">
                {% for message in get_flashed_messages() %}
                <span class="dt-auth-flash">{{ message }}</span>
                {% endfor %}
                {{ form.csrf_token }}
                <input type="hidden" name="next" value="{{next}}" />
                {{ form.email(class="form-control dt-auth-input",placeholder="메일 주소") }}
                {{ form.password(class="form-control dt-auth-input",placeholder="비밀번호") }}
                {{ form.submit(class="btn btn-md btn-primary btn-block dt-auth-btn") }}
            </form>
        </section>
    </div>
</div>

{% endblock %}
```

login.html

detector/base.html을 기반으로 로그인 페이지를 정의함.

페이지의 제목은 로그인 이며 실제 내용은 로그인 폼이며 제목과 내용을 동적으로 정의함.

로그인 폼은 CSRF 토큰, 이메일 입력 필드, 비밀번호 입력 필드, 제출 버튼으로 구성됨.

url_for("auth.login") 함수를 사용하여 로그인 폼이 제출될 때 사용자를 로그인 처리하는데 필요한 URL로 설정.

get_flashed_messages를 사용하여 로그인 실패, 성공의 상태 메시지를 표시.

```
{% extends "detector/base.html" %}

{% block title %}사용자 신규 등록{% endblock %}

{% block content %}
<div class="mx-auto dt-auth-main">
    <div class="card dt-auth-signup">
        <header>사용자 신규 등록</header>
        <section>
            <form method="post" action="{{ url_for('auth.signup', next=request.args.get('next')) }}"
                class="form-signin">
                {{ form.csrf_token }}
                {% for message in get_flashed_messages() %}
                <div class="dt-auth-flash">{{ message }}</div>
                {% endfor %}

                {{ form.username(size=30, class="form-control dt-auth-input", placeholder="사용자명") }}
                {{ form.email(class="form-control dt-auth-input",placeholder="메일 주소") }}
                {{ form.password(class="form-control dt-auth-input",placeholder="비밀번호") }}
                {{ form.submit(class="btn btn-md btn-primary btn-block dt-auth-btn") }}
            </form>
        </section>
    </div>
</div>
{% endblock %}
```

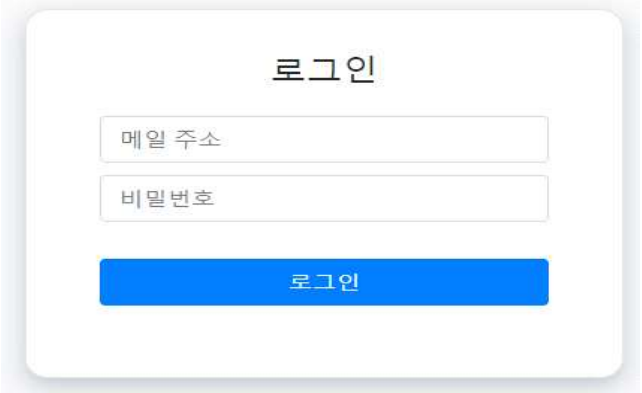
signup.html

detector/base.html을 기반으로 사용자가입 페이지를 정의함.

페이지의 제목은 사용자 신규 등록으로 설정하고 구성은 사용자명 입력 필드, 이메일 입력 필드, 비밀번호 입력 필드, 제출 버튼으로 구성.

사용자가 제출한 데이터를 post방식으로 전송되도록 폼을 구성.

get_flashed_messages를 사용하여 사용자가입 성공, 실패 등 메시지를 표시.



-로그인 화면, 사용자가입 화면-

```python
from flask_wtf import FlaskForm
from wtforms import PasswordField, StringField, SubmitField
from wtforms.validators import DataRequired, Email, Length
class SignUpForm(FlaskForm):
    username = StringField(
        "사용자명",
        validators=[
            DataRequired("사용자명은 필수입니다."),
            Length(1, 30, "30문자 이내로 입력해 주세요."),
        ],
    )
    email = StringField(
        "메일 주소",
        validators=[
            DataRequired("메일 주소는 필수입니다."),
            Email("메일 주소의 형식으로 입력해 주세요."),
        ],
    )
    password = PasswordField("비밀번호", validators=[DataRequired("비밀번호는 필수입니다.")])
    submit = SubmitField("신규 등록")
class LoginForm(FlaskForm):
    email = StringField(
        "메일 주소",
        validators=[
            DataRequired("메일 주소는 필수입니다."),
            Email("메일 주소의 형식으로 입력하세요."),
        ],
    )
    password = PasswordField("비밀번호", validators=[DataRequired("비밀번호는 필수입니다.")])
    submit = SubmitField("로그인")
```

form.py

사용되는 사용자 가입 폼과 로그인 폼을 정의하는 코드.

사용자명, 이메일, 비밀번호 필드 및 등록버튼이 포함되어있는 SignUpForm 클래스를 생성.
각 필드에는 유효성 검사 기능이 설정되어 있음.
사용자명 필드에는 필수 입력 및 길이 제한 적용.

이메일과 비밀번호 필드, 로그인 버튼이 포함되어있는 loginForm 클래스를 생성

각 필드에는 필수 입력이 적용되어 있으며 이메일에는 이메일 형식 유효성 검사가 적용.

```python
from flask import (Blueprint, flash, redirect, render_template, request, url_for)
from flask_login import login_user,logout_user
from apps.app import db
from apps.crud.models import User
from apps.auth.forms import LoginForm, SignUpForm
auth = Blueprint("auth",
                 __name__,
                 template_folder="templates",
                 static_folder="static")
@auth.route("/")
def index():
    return render_template("auth/index.html")
@auth.route("/signup", methods=["GET", "POST"])
def signup():
    form = SignUpForm()
    if form.validate_on_submit():
        user = User(
            username=form.username.data,
            email=form.email.data,
            password=form.password.data,
        )
        if user.is_duplicate_email():
            flash("지정한 이메일 주소는 이미 등록되어 있습니다.")
            return redirect(url_for("auth.signup"))
        db.session.add(user)
        db.session.commit()
        return redirect(url_for("crud.users"))
    return render_template("auth/signup.html",form=form)
@auth.route("/login", methods=["GET", "POST"])
def login():
    next_ = request.args.get("next")
    form = LoginForm()
    if request.method=="POST":
        user = User.query.filter_by(email=form.email.data).first()
        next_ = request.form["next"]
        if user is not None and user.verify_password(form.password.data):
            login_user(user)
            if next_ is None or not next_.startswith("/"):
                next_ = url_for("detector.index")
            return redirect(next_)
        else:
            flash("메일 주소 또는 비밀번호가 일치하지 않습니다")
            return redirect(url_for("auth.login",next=next_))
    return render_template("auth/login.html",form=form,next = next_)
@auth.route("/logout")
def logout():
    logout_user()
    return redirect(url_for("auth.login"))
```

views.py

인증과 관련된 라우트와 로직을 그룹화하는 Blueprint 클래스를 사용하여 생성.

signup 라우트는 GET 방식으로 사용자 가입 폼을 렌더링하고, POST 방식으로 사용자가 가입 폼을 제출할 때의 로직을 처리.

SignUpForm을 사용하여 사용자 입력을 검증하고, 이메일 중복 검사 수행.

유효한 입력이면 새로운 사용자를 데이터베이스에 추가하고, 그렇지 않으면 다시 가입 페이지를 표시.

login 라우트는 GET 방식으로 로그인 폼을 렌더링하고, POST 방식으로 사용자가 로그인 폼을 제출할 때의 로직을 처리.

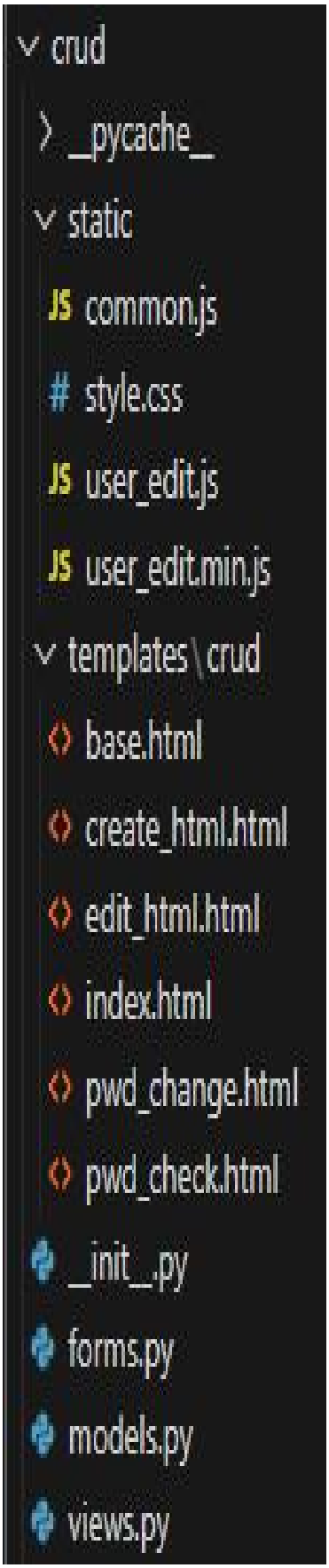
LoginForm을 사용하여 사용자 입력을 검증하고, 데이터베이스에서 이메일에 해당하는 사용자를 탐색.

비밀번호가 일치하는 경우 사용자를 로그인하고, 그렇지 않으면 다시 로그인 페이지를 표시.

로그인 후에는 사용자가 요청한 다음 페이지로 리다이렉트.

logout 라우트는 사용자를 로그아웃하고 로그인 페이지로 리다이렉트.

2-3. CRUD



1. static/common.js: 웹 애플리케이션에서 공통적으로 사용되는 JavaScript 코드를 포함하는 파일.

2. static/style.css: 웹 애플리케이션의 스타일을 정의하는 CSS 파일.

3. static/user_edit.js: 사용자 정보 편집 페이지에 사용되는 JavaScript 코드 파일로 사용자가 프로필 정보를 수정할 때 발생하는 이벤트나 서버와의 통신을 처리하는 기능을 구현.

4. static/user_edit.min.js: 위의 `user_edit.js` 파일을 압축한 버전.

5. templates/base.html: 기본 레이아웃을 정의하는 파일로 모든 페이지의 공통적인 요소들을 포함.

6. templates/create_html.html: 새로운 데이터를 생성하는 페이지의 HTML 템플릿으로 새로운 글을 작성하는 페이지나 새로운 사용자를 등록하는 페이지에 사용.

7. templates/edit_html.html: 기존 데이터를 수정하는 페이지의 HTML 템플릿으로 기존의 정보를 수정하거나 업데이트할 때 사용.

8. templates/index.html: 홈페이지를 정의하는 파일로 접속할 때 보여지는 첫 번째 페이지.

9. templates/pwd_change.html: 사용자의 비밀번호를 변경하는 페이지의 HTML 템플릿.

10. templates/pwd_check.html: 사용자의 비밀번호를 확인하는 페이지의 HTML 템플릿으로 비밀번호 변경을 위해 현재 비밀번호를 확인하는 폼을 제공.

12. forms.py: 사용자 입력을 처리하고 데이터를 유효성 검사하는 폼을 정의하는 Python 파일.

13. models.py: 데이터베이스 테이블의 구조와 상호작용을 정의하는 Python 파일.

14. views.py: 클라이언트 요청을 처리하고 데이터를 조회하여 HTML 페이지를 렌더링하여 응답을 생성하는 Python 파일.

base.html

이 html 템플릿은 crud, detector, notice에 공통적으로 들어가는 템플릿으로 어느 메뉴를 들어가도 코드는 같기 때문에 여기에서만 설명.

네비게이션 바는 로그인된 사용자와 로그인되지 않은 사용자를 위한 다른 메뉴 항목을 포함하며 사용자가 인증되면 공지사항, 사용자일람, 감지와 같은 추가 항목이 표시되며, 사용자 이름과 로그아웃 링크를 표시.

웹 페이지에 CSS 스타일을 적용하기 위해 link를 이용하여 bootstrap와 style를 연결

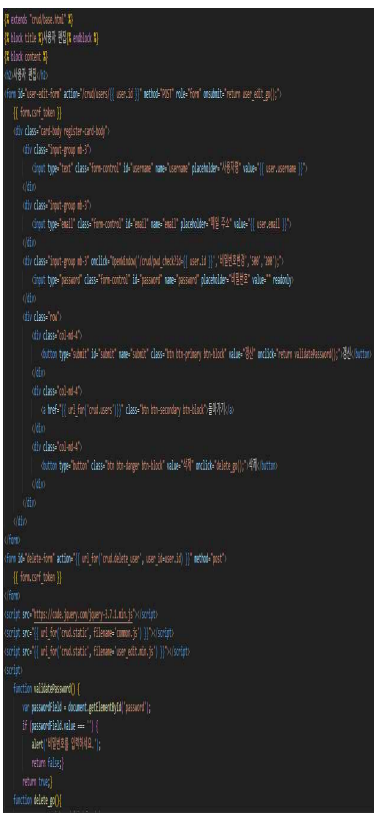
create_html.html

사용자 신규 등록 페이지를 정의하는 html 템플릿으로 사용자는 이 페이지를 통해 새로운 계정 생성 가능.

사용자는 각 필드(사용자명, 이메일, 비밀번호)를 입력하여 새로운 계정을 등록할 수 있음.

만약에 입력을 하지 않았다면 유효성 검사를 통해 입력되지 않은 것을 확인, '필수 입력 사항입니다.'라는 메시지를 출력.

전부 입력하고 등록버튼을 누르면 서버로 데이터가 제출하면 서버 측에서 유효성 검사 및 데이터 처리가 이루어지고 다 이루어지면 신규 사용자가 등록됨.



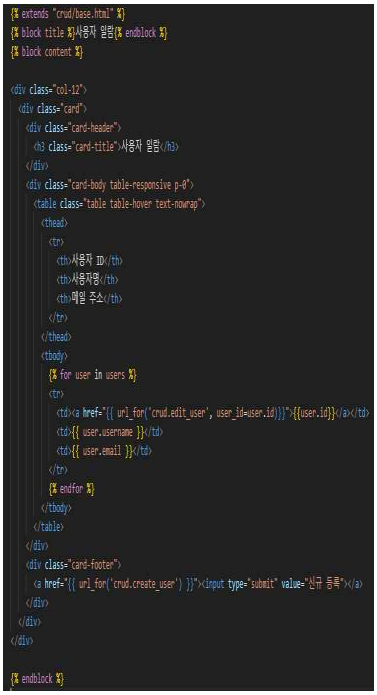
edit_html.html

사용자 정보를 수정하기 위한 html 템플릿으로 사용자는 이 페이지를 통해 자신 계정의 정보를 수정 가능.

사용자는 각 필드(사용자명, 이메일, 비밀번호)를 입력하고 갱신 버튼을 누르면 데이터가 엔드포인트로 데이터를 전송.

비밀번호 입력 필드는 읽기 전용으로 되어 있으며 비밀번호 필드를 클릭하면 비밀번호 변경을 위한 창이 열리고 거기서 사용자의 현재 비밀번호를 누르면 변경 가능.

갱신, 돌아가기, 삭제 총 3개의 버튼이 있는데
갱신은 정보 수정할 때 입력후 데이터를 전송시키기 위한 버튼.
돌아가기는 수정을 하지 않고 다시 유저 목록으로 돌아가는 버튼.
삭제는 사용자 정보를 삭제하는 버튼으로 확인을 받은 후에 삭제 함수를 호출하여 사용자 정보를 삭제.



index.html

사용자 명단을 보여주는 html 템플릿으로 사용자는 이 페이지를 통해 현재 가입 되어 있는 사용자의 명단을 볼 수 있고 클릭하여 상세정보를 볼 수 있으며 등록할 수 있음.

사용자 정보를 표시하기 위해 사용자ID, 사용자명, 메일 주소의 데이터를 가진 테이블을 구성.
users 변수에 저장된 사용자 리스트를 순회하면서 각 사용자의 정보를 테이블의 행으로 표시.
사용자 ID는 링크로 구성되어 있어 사용자 편집 페이지로 이동 가능.

신규 사용자 등록을 위한 버튼이 있어 이 버튼을 누르면 사용자 등록 페이지로 이동.

pwd_check.html

현재 사용자의 비밀번호를 확인하는 템플릿.

사용자는 현재 비밀번호를 입력하고 전송 버튼을 클릭하면 데이터가 서버로 전송되고 서버에서 입력된 비밀번호와 실제 저장된 비밀번호를 비교.

만약 일치하지 않는 경우에는 사용자에게 알림 메시지를 표시.

pwd_change.html

현재 사용자의 비밀번호를 새로운 비밀번호로 변경하고 확인하는 템플릿.

사용자는 새로운 비밀번호와 비밀번호 확인 필드를 입력 후 확인 버튼을 누르면 입력된 비밀번호와 확인된 비밀번호를 비교하여 일치하면 부모 창에게 비밀번호를 전달하고 창을 닫음.

비밀번호가 일치하지 않으면 알림 메시지를 표시.

form.py

```python
from flask_wtf import FlaskForm
from wtforms import PasswordField, StringField, SubmitField
from wtforms.validators import DataRequired, Email, length, InputRequired
from wtforms import HiddenField

class UserForm(FlaskForm):
    username = StringField(
        '사용자명',
        validators=[
            DataRequired(message='사용자명은 필수입니다.'),
            length(max=30, message='30문자 이내로 입력해 주세요.'),
        ],
    )
    email = StringField(
        '메일 주소',
        validators=[
            DataRequired(message='메일 주소는 필수입니다.'),
            Email(message='메일 주소의 형식으로 입력해 주세요.'),
        ],
    )
    password = PasswordField('비밀번호', validators=[DataRequired(message='비밀번호는 필수입니다.')])

    submit = SubmitField('신규 등록')

class PasswordCheckForm(FlaskForm):
    user_id = HiddenField('user_id', validators=[InputRequired()])
    password = PasswordField('password', validators=[InputRequired()])
```

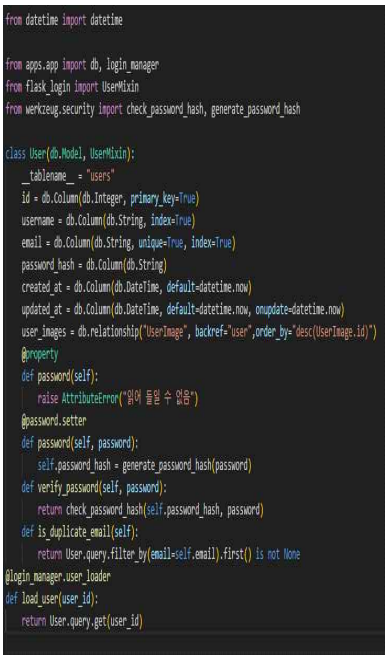
사용자의 정보를 입력하는 폼과 비밀번호를 확인하는 폼이 포함되어있는 .py 파일.

사용자의 이름, 이메일, 비밀번호를 입력하는 폼이 있으며 각 입력 필드에는 데이터가 필수적으로 요구되며 입력하지 않았을 경우 이에 대한 메시지 출력.

사용자명은 최대 30자까지 입력 가능하도록 설정.
이메일은 이메일 형식으로 입력하지 않으면 메시지 출력.
'신규 등록' 버튼을 통해 새로운 사용자로 등록 가능.

비밀번호 체크 폼을 통해 입력된 데이터를 서버로 전달하고 유효성을 검사하는 데 사용.

model.py



```python
from datetime import datetime

from apps.app import db, login_manager
from flask_login import UserMixin
from werkzeug.security import check_password_hash, generate_password_hash

class User(db.Model, UserMixin):
    __tablename__ = "users"
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String, index=True)
    email = db.Column(db.String, unique=True, index=True)
    password_hash = db.Column(db.String)
    created_at = db.Column(db.DateTime, default=datetime.now)
    updated_at = db.Column(db.DateTime, default=datetime.now, onupdate=datetime.now)
    user_images = db.relationship("UserImage", backref="user",order_by="desc(UserImage.id)")
    @property
    def password(self):
        raise AttributeError("읽어 들일 수 없음")
    @password.setter
    def password(self, password):
        self.password_hash = generate_password_hash(password)
    def verify_password(self, password):
        return check_password_hash(self.password_hash, password)
    def is_duplicate_email(self):
        return User.query.filter_by(email=self.email).first() is not None
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(user_id)
```

사용자의 ID, 이름, 이메일, 비밀번호 등의 필드가 포함된 사용자 모델을 정의하는 코드.

users 테이블에 id, username, email, password_hash, created_at, updated_at, user_images를 저장.

패스워드는 비밀번호를 설정하고 해시화하여 저장하는데 입력된 비밀번호가 올바른지 확인할 수 있도록 설정.

이메일은 중복된 이메일 주소가 있는지 확인.

load_user 함수를 이용하여 데이터베이스에서 사용자 ID를 기반으로 검색하여 반환.

```python
from apps.app import db
from apps.crud.models import User
from apps.crud.forms import UserForm
from werkzeug.security import check_password_hash
from flask import Blueprint,render_template, redirect, url_for, request, flash
from flask_login import login_required

crud = Blueprint(
    "crud",
    __name__,
    template_folder="templates",
    static_folder="static",
)

@crud.route("/")
@login_required
def index():
    return render_template("crud/index.html")

@crud.route("/sql")
@login_required
def sql():
    user = db.session.query(User).filter_by(id=1).all()
    print(user)
    return "콘솔 로그를 확인해 주세요"

@crud.route("/users/new", methods=["GET", "POST"])
@login_required
def create_user():
    form = UserForm()
    if form.validate_on_submit():
        user = User(
            username=form.username.data,
            email=form.email.data,
            password=form.password.data,
        )
        db.session.add(user)
        db.session.commit()
        return redirect(url_for("crud.users"))
    return render_template("crud/create_html.html", form=form)
```

views.py

crud 기능을 구현하는 코드.

crud 블루프린트를 생성하여 해당 모듈에서 사용할 URL과 템플릿 및 정적 파일의 경로를 설정.

users 함수: 사용자 리스트를 조회.

create_user 함수: 사용자를 생성. 사용자 생성 페이지는 UserForm을 사용하여 입력 폼을 생성하고, 폼이 유효하면 데이터베이스에 사용자를 추가.

edit_user 함수: 사용자를 편집하는 페이지를 처리. 사용자의 정보를 수정하고 데이터베이스에 반영.

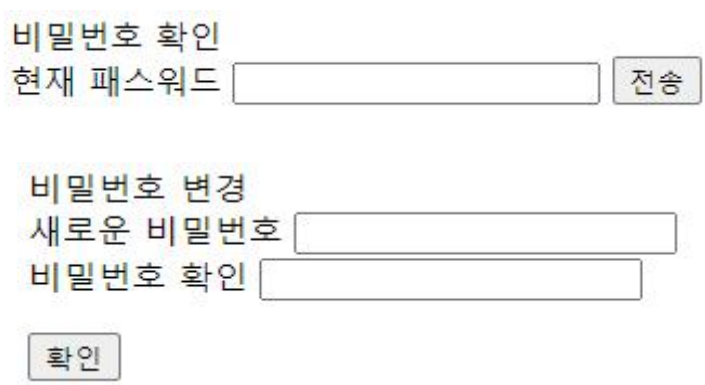
pwd_check 함수: 사용자가 비밀번호를 확인하고 변경하는 페이지를 처리. 비밀번호 확인 후 올바르면 비밀번호 변경 페이지로 이동.

delete_user 함수: 사용자 정보를 데이터베이스에서 삭제하고 변경사항을 커밋.

- 사용자 신규 등록 폼 -

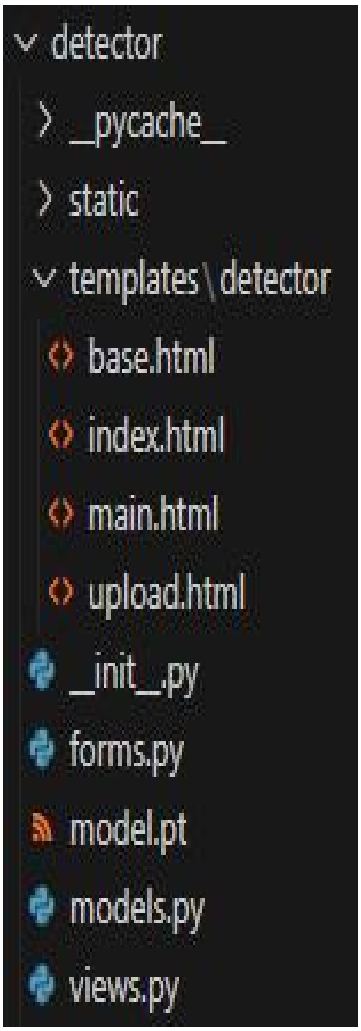
- 사용자 명단 -

- 사용자 편집-



- 비밀번호 확인 및 변경 화면 -

2-4. 메인화면 및 감지화면



1. templates/base.html: 기본 레이아웃을 정의하는 파일로 모든 페이지의 공통적인 요소들을 포함.

2. templates/index.html: 홈페이지를 정의하는 파일로 접속할 때 보여지는 첫 번째 페이지.

3. templates/main.html: 메인페이지를 정의하는 파일로 로그인여부와 관계없이 보여지는 화면 페이지.

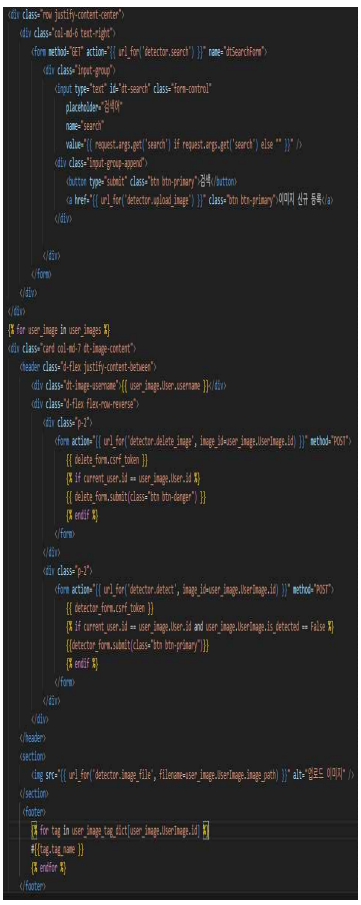
4. templates/upload.html: 파일 업로드를 위한 템플릿 파일로 사진감지를 위한 사진파일을 서버에 업로드 하기 위해 필요한 파일.

5. forms.py: 사용자 입력을 처리하고 데이터를 유효성 검사하는 폼을 정의하는 Python 파일.

6. model.pt: 파이토치 라이브러리에서 훈련된 딥 러닝 모델을 저장하는 데 사용되는 파일.

7. models.py: 데이터베이스 테이블의 구조와 상호작용을 정의하는 Python 파일.

8. views.py: 클라이언트 요청을 처리하고 데이터를 조회하여 HTML 페이지를 렌더링하여 응답을 생성하는 Python 파일.



index.html

이미지 감지기 화면을 렌더링 하는데 사용되는 코드.

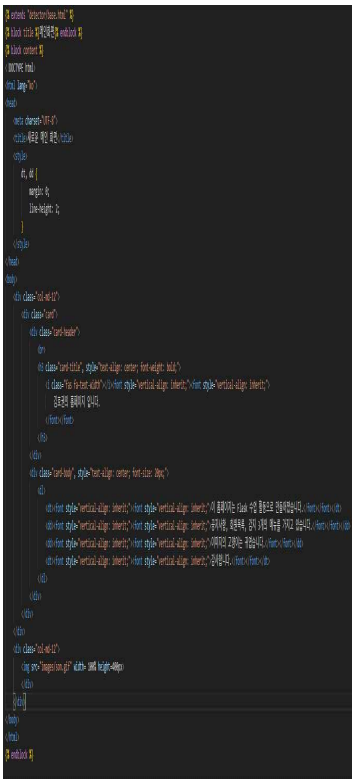
사용자에게 플래시 메시지를 표시하며 메시지는 작업 결과 또는 오류 메시지 등을 보여준다.

이미지 검색을 위한 입력 필드와 검색 버튼을 포함한 검색 폼을 제공하며 검색 버튼을 클릭하면 검색어에 따라 이미지가 필터링.

이미지 업로드를 위한 링크를 제공하며 링크를 클릭하면 새 이미지를 업로드하는 페이지로 이동.

사용자가 업로드한 각 이미지를 카드 형식으로 표시하며 각 이미지에는 다음 요소가 포함.
- 이미지를 업로드한 사용자의 이름
- 이미지 삭제를 위한 버튼 (사용자가 이미지를 업로드한 경우에만 표시)
- 이미지 감지를 위한 버튼 (이미지가 감지되지 않은 경우에만 표시)
- 이미지 자체와 이미지에 연결된 태그들의 목록

main.html

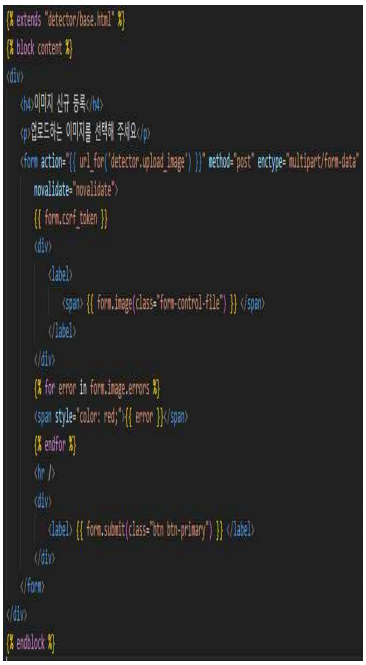


이 홈페이지의 전체 메인화면 html 템플릿으로 로그인을 한 사용자도 하지 않은 사용자도 처음 들어오면 보게 되는 메인 화면.

스타일은 style 태그 안에 포함.

전체적인 내용
1. 김호권의 홈페이지임을 나타내는 텍스트
2. <dl>, <dt>, <dd>태그를 사용하여 홈페이지의 목적이나 제작 배경에 대한 설명을 포함.
3. 주석과 감사인사.
4. 이미지
로 구성되어 있으며 홈페이지 방문자에게 정보를 제공하고 홈페이지의 콘텐츠와 기능을 소개.

upload.html

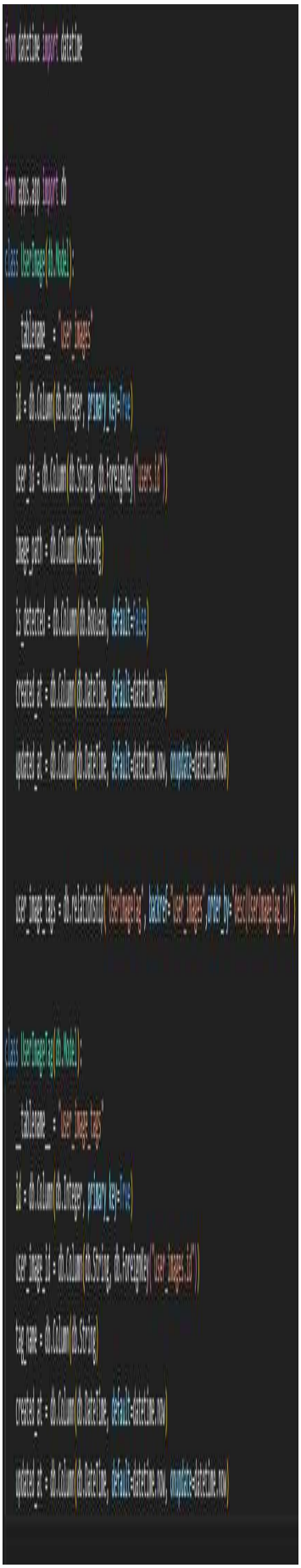


신규 이미지를 업로드 하는 화면 html 템플릿.

form 태그를 사용하고, action 속성에 이미지를 업로드 하는 URL을 지정.
method 속성은 POST.

이미지를 업로드하기 위한 입력 필드를 표시.
필드에는 이미지를 업로드 하는 버튼과 사용자가 이미지 파일을 로컬에서 선택해 가져올 수 있는 기능 구현.
업로드 시 만약 오류가 발생하였다면 오류 메시지를 가져와서 출력.

models.html



데이터베이스에 이미지와 관련된 정보를 저장하기 위한 모델을 정의.

UserImage모델은 테이블 이름을 user_images로 명하고 다음 요소를 포함하여 구성

- id: 이미지의 고유 식별자로 사용되는 정수형.

- user_id: 해당 이미지를 업로드한 사용자의 ID를 외래 키로 참조.

- image_path: 이미지 파일의 경로를 저장하는 문자열 컬럼.

- is_detected: 이미지에서 감지된 여부를 나타내는 부울 값으로 기본값은 False로 지정.

- created_at: 레코드가 생성된 날짜와 시간을 나타내는 DateTime 컬럼으로 기본값은 현재 시간으로 지정.

- updated_at: 레코드가 마지막으로 업데이트된 날짜와 시간을 나타내는 DateTime 컬럼으로 기본값은 현재 시간으로 지정.

UserImageTag 모델은 테이블 이름을 user_image_tags로 명하고 다음 요소를 포함하여 구성.

- id: 태그의 고유 식별자로 사용되는 정수형.

- user_image_id: 해당 태그가 속한 이미지의 ID를 외래 키(foreign key)로 참조.

- tag_name: 태그의 이름을 저장하는 문자열 컬럼.

- created_at: 레코드가 생성된 날짜와 시간을 나타내는 DateTime 컬럼으로 기본값은 현재 시간으로 지정.

- updated_at: 레코드가 마지막으로 업데이트된 날짜와 시간을 나타내는 DateTime 컬럼으로 기본값은 현재 시간으로 지정.

```python
from apps.detector.forms import UploadImageForm,DetectorForm,DeleteForm

dt = Blueprint("detector", __name__, template_folder="templates")


@dt.route('/')
def index():
    return render_template('detector/main.html')


@dt.route("/dindex")
def dindex():

    user_images,user_image_tag_dict = get_user_images()

    detector_form = DetectorForm()

    delete_form = DeleteForm()

    return render_template("detector/index.html",
                           user_images=user_images,
                           detector_form=detector_form,
                           delete_form=delete_form,
                           user_image_tag_dict=user_image_tag_dict,
                           )

@dt.route("/images/<path:filename>")
def image_file(filename):
    return send_from_directory(current_app.config["UPLOAD_FOLDER"], filename)


@dt.route("/upload", methods=["GET", "POST"])
@login_required
def upload_image():
    form = UploadImageForm()
    if form.validate_on_submit():
        file = form.image.data
        ext = Path(file.filename).suffix
        image_uuid_file_name = str(uuid.uuid4()) + ext
        image_path = Path(current_app.config["UPLOAD_FOLDER"], image_uuid_file_name)
        file.save(image_path)

        user_image = UserImage(user_id=current_user.id, image_path=image_uuid_file_name)
        db.session.add(user_image)
        db.session.commit()

        return redirect(url_for("detector.dindex"))
```

views.html

이미지를 업로드하고 이미지에 대해 물체를 감지하는 기능을 구현하는 html 템플릿.

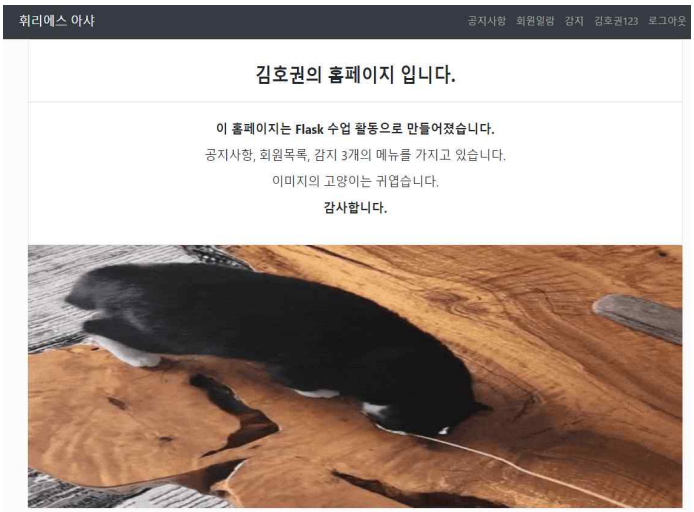
dindex 라우트는 업로드된 이미지를 화면에 표시하고, 물체 검지를 실행할 수 있는 페이지를 렌더링. 또한, upload_image 라우트는 이미지를 업로드할 수 있는 페이지를 렌더링하고, 사용자가 이미지를 선택하여 업로드 가능.

exec_detect 함수는 이미지에 대한 물체 검지를 실행하고, 검지된 이미지 파일을 저장. 이후, save_detected_image_tags 함수를 사용하여 검지된 물체에 대한 태그 정보를 저장.

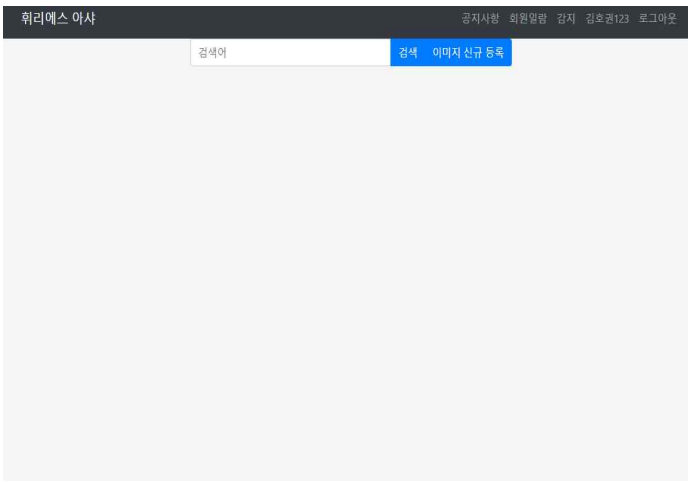
delete_image 라우트는 업로드된 이미지를 삭제. 사용자가 이미지를 삭제할 때는 해당 이미지와 관련된 태그 정보도 함께 삭제.

search 라우트는 사용자가 입력한 태그로 이미지를 검색하고 결과를 화면에 표시.

page_not_found 함수는 404 에러가 발생했을 때 사용자에게 적절한 오류 페이지를 표시.

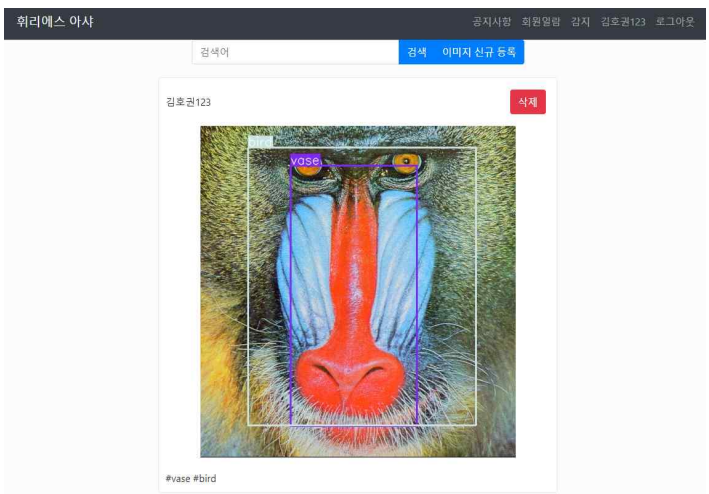


- 메인 화면 -



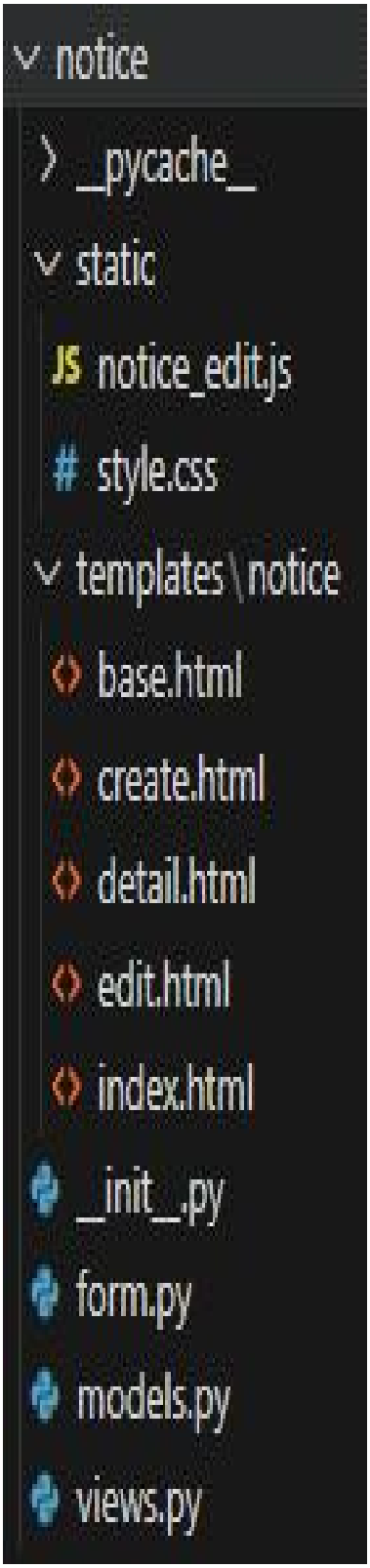
- 이미지 감지 게시판 -

- 이미지 신규 등록 -



- 이미지 등록 후 감지 버튼 -

2-5. 공지사항



1. static/style.css: 웹 애플리케이션의 스타일을 정의하는 CSS 파일.

2. static/notice_edit.js: 공지사항 편집 페이지에 사용되는 JavaScript 코드 파일로 사용자가 프로필 정보를 수정할 때 발생하는 이벤트나 서버와의 통신을 처리하는 기능을 구현.

3. templates/base.html: 기본 레이아웃을 정의하는 파일로 모든 페이지의 공통적인 요소들을 포함.

4. templates/create.html: 새로운 데이터를 생성하는 페이지의 HTML 템플릿으로 새로운 글을 작성하는 페이지나 새로운 사용자를 등록하는 페이지에 사용.

5. templates/edit.html: 기존 데이터를 수정하는 페이지의 HTML 템플릿으로 기존의 정보를 수정하거나 업데이트할 때 사용.

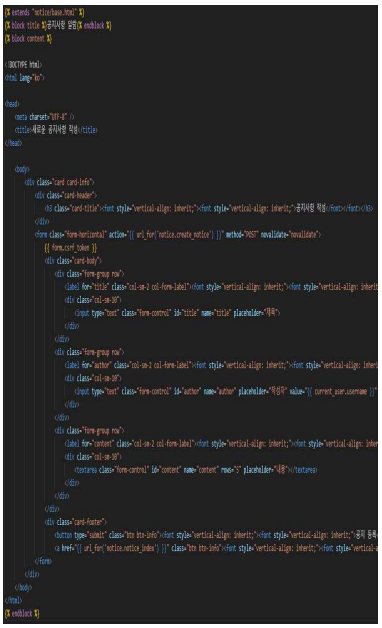
6. templates/index.html: 홈페이지를 정의하는 파일로 접속할 때 보여지는 첫 번째 페이지.

7. templates/detail.html: 공지사항의 상세정보 페이지의 HTML 템플릿.

8. forms.py: 사용자 입력을 처리하고 데이터를 유효성 검사하는 폼을 정의하는 Python 파일.

9. models.py: 데이터베이스 테이블의 구조와 상호작용을 정의하는 Python 파일.

10. views.py: 클라이언트 요청을 처리하고 데이터를 조회하여 HTML 페이지를 렌더링하여 응답을 생성하는 Python 파일.



create.html

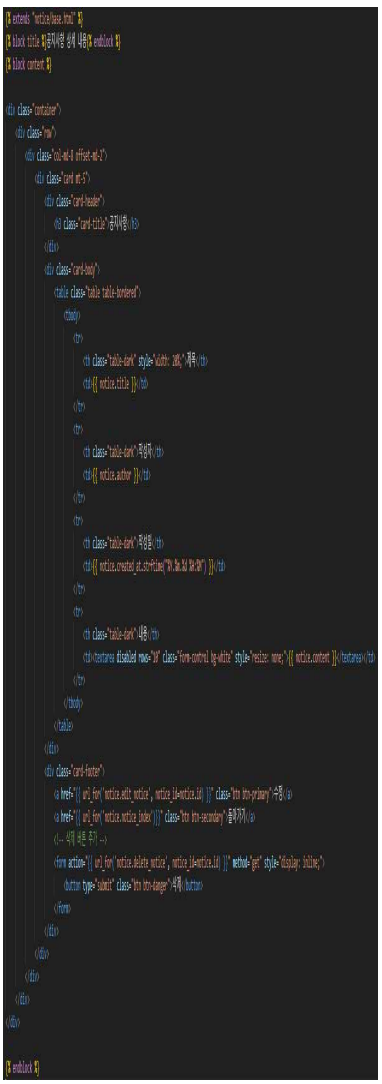
공지사항 작성하는 페이지 html 템플릿으로 구성 요소는 제목, 작성자, 내용을 입력하여 새로운 공지 등록 가능.

제목은 반드시 입력해야 한다.

작성자는 현재 로그인한 사용자의 이름이 자동으로 입력.

내용까지 다 입력한 후 공지 등록 버튼을 클릭하면 입력된 정보가 서버로 전송되어 공지사항이 등록.

등록하지 않고 공지 등록 페이지를 떠나고 싶을 경우 취소 버튼을 누르면 이동.



detail.html

공지사항의 상세 내용을 보여주는 페이지 html 템플릿으로 페이지에는 공지사항의 제목, 작성자, 작성일, 내용이 표시되며 수정 및 삭제 버튼을 제공되어 수정하거나 삭제 가능.

notice.title를 통해 공지사항의 제목을 가져와서 표시.

notice.author를 통해 공지사항을 작성한 사용자의 이름을 표시.

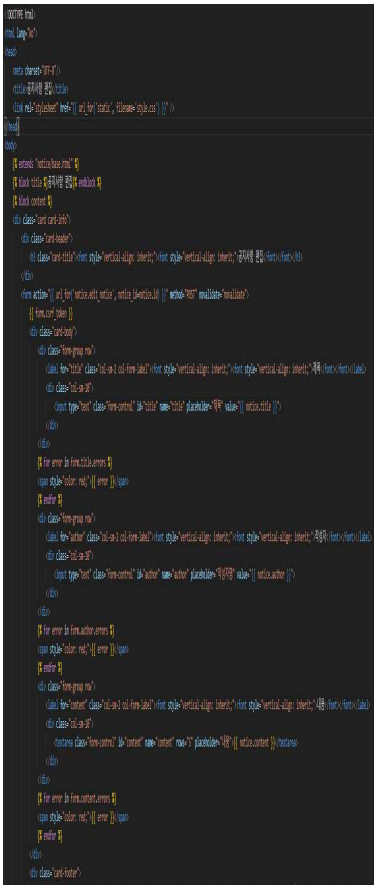
notice.created_at을 통해 공지사항이 작성된 날짜와 시간이 표시되며 strftime 함수를 사용하여 날짜 및 시간 형식을 지정.

notice.content를 통해 공지사항의 내용을 가져와서 표기되며 사용자가 편집할 수 없도록 설정.

수정 버튼을 통해 해당 공지사항을 수정할 수 있는 링크가 제공되며 클릭하면 수정 페이지로 이동.

돌아가기 버튼을 통해 공지사항 목록 페이지로 돌아갈 수 있는 링크가 제공되며 클릭하면 공지사항 목록 페이지로 이동.

삭제 버튼을 통해 클릭하면 해당 공지사항이 삭제.



edit.html

공지사항을 편집하는 페이지 html 템플릿으로 공지사항의 제목, 작성자, 내용을 편집할 수 있는 폼이 제공되며 수정된 내용을 저장하기 위한 "수정" 버튼과 편집을 취소하고 상세 내용 페이지로 돌아가기 위한 "돌아가기" 버튼이 제공.

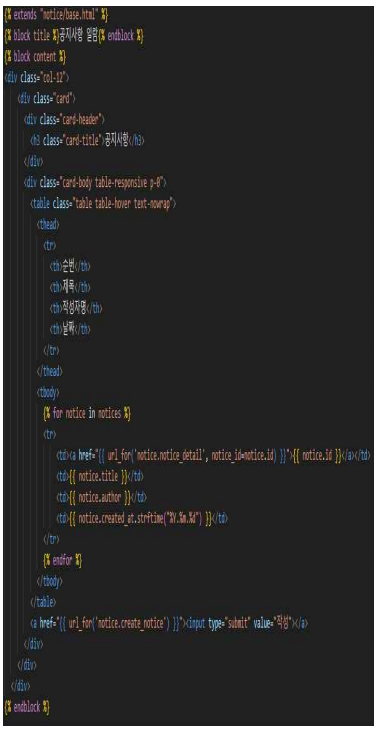
제목은 초기 값으로 현재 공지사항의 제목을 표시.

작성자는 초기 값으로 현재 공지사항의 작성자를 표시.

내용은 초기 값으로 현재 공지사항의 내용이 표시

수정 버튼을 클릭하면 수정된 공지사항이 저장되고 해당 공지사항의 상세 내용 페이지로 이동.

돌아가기 버튼을 누르면 편집을 취소하고 해당 공지사항의 상세 내용 페이지로 이동.



index.html

공지사항 목록을 보여주는 페이지 html 템플릿으로 다음과 같은 요소로 구성.

제목은 공지사항으로 설정.

공지사항 항목으로 "순번", "제목", "작성자명", "날짜"로 구성.

"순번"은 각 공지사항의 고유 식별자를 나타내며, 링크를 통해 해당 공지사항의 상세 내용 페이지로 이동.

"제목", "작성자명", "날짜"는 각 공지사항의 제목, 작성자명, 작성일을 표시.

작성 버튼을 클릭하면 공지사항을 작성하는 페이지로 이동.

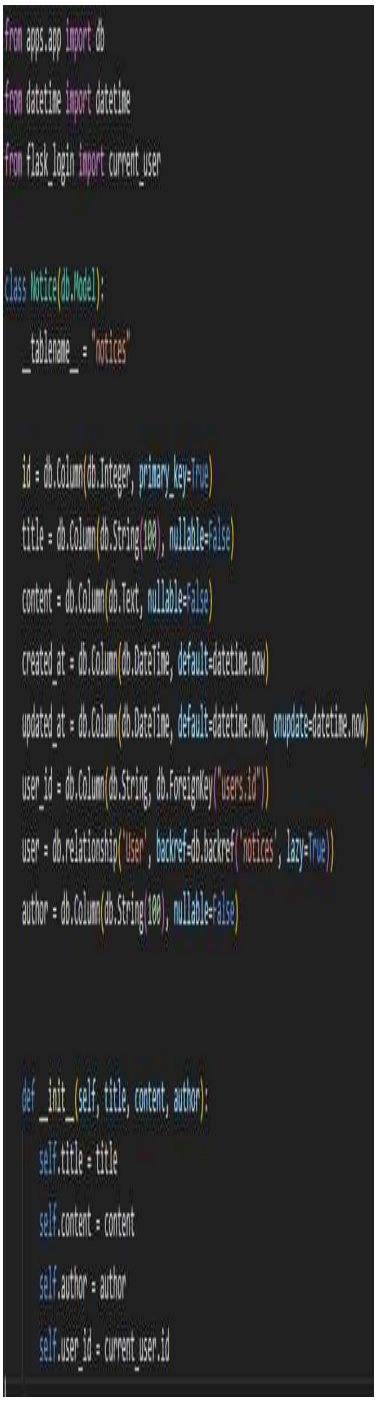
```python
from flask_wtf import FlaskForm
from wtforms import StringField, TextAreaField, SubmitField
from wtforms.validators import DataRequired
from wtforms import DateField

class NoticeForm(FlaskForm):
    title = StringField('제목', validators=[DataRequired()])
    author = StringField('작성자명',validators=[DataRequired()])
    content = TextAreaField('내용', validators=[DataRequired()])
    submit = SubmitField('작성')
```

form.py

공지사항 작성 폼을 정의하는 코드로 다음과 같은 필드가 포함.

title: 공지사항의 제목을 입력하는 문자열 필드로 데이터를 필수로 입력해야 함.
author: 공지사항을 작성한 사용자의 이름을 입력하는 문자열 필드로 데이터를 필수로 입력해야 함.
content: 공지사항의 내용을 입력하는 여러 줄의 문자열 필드로 데이터를 필수로 입력해야 함.
submit: 공지사항을 제출하는 버튼입니다.



```python
from apps.app import db
from datetime import datetime
from flask_login import current_user

class Notice(db.Model):
    __tablename__ = 'notices'

    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    content = db.Column(db.Text, nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.now)
    updated_at = db.Column(db.DateTime, default=datetime.now, onupdate=datetime.now)
    user_id = db.Column(db.String, db.ForeignKey("users.id"))
    user = db.relationship('User', backref=db.backref('notices', lazy='true'))
    author = db.Column(db.String(100), nullable=False)


    def __init__(self, title, content, author):
        self.title = title
        self.content = content
        self.author = author
        self.user_id = current_user.id
```

models.py

공지사항을 데이터베이스에 저장하기 위한 Notice 모델을 정의하는 코드로 다음과 같은 필드와 메서드를 포함.

id: 공지사항의 고유한 식별자를 나타내는 정수형 필드로 이 필드는 기본 키로 사용.

title: 공지사항의 제목을 나타내는 문자열 필드로 최대 길이는 100자로 제한되어 있고, 빈 값이 허용되지 않음.

content: 공지사항의 내용을 나타내는 텍스트 필드로 빈 값이 허용되지 않음.

created_at: 공지사항이 생성된 일시를 나타내는 날짜 및 시간 필드로 기본값으로 현재 날짜와 시간으로 설정.

updated_at: 공지사항이 마지막으로 업데이트된 일시를 나타내는 날짜 및 시간 필드로 공지사항이 생성될 때는 created_at과 동일한 값을 갖고 이후 업데이트될 때마다 해당 필드가 업데이트.

user_id: 공지사항을 작성한 사용자의 고유한 식별자를 나타내는 문자열 필드로 이 필드는 User 모델의 외래 키로 사용.

author: 공지사항을 작성한 사용자의 이름을 나타내는 문자열 필드로 최대 길이는 100자로 제한되어 있고, 빈 값이 허용되지 않음.

user: User 모델과의 관계를 나타내는 백릴레이션으로 User 모델의 인스턴스를 참조.

__init__: Notice 클래스의 생성자 메서드로 공지사항의 제목, 내용, 작성자를 인수로 받아 인스턴스를 초기화. 또한 현재 사용자의 고유한 식별자를 user_id 필드에 설정.

```python
notice = Blueprint('notice', __name__, template_folder='templates')

@notice.route('/create', methods=['GET', 'POST'])
def create_notice():
    form = NoticeForm()

    if form.validate_on_submit():
        author = current_user.username if current_user.is_authenticated else "Anonymous"
        notice = Notice(
            title=form.title.data,
            author=author,
            content=form.content.data
        )
        db.session.add(notice)
        db.session.commit()
        return redirect(url_for('notice.notice_index'))


    return render_template('notice/create.html', title='새로운 공지사항 작성', form=form)

@notice.route('/index')
def notice_index():
    notices = Notice.query.all()
    user = current_user
    notices.reverse()
    return render_template('notice/index.html', title='공지사항 알림', notices=notices, user=user)

@notice.route('/detail/<int:notice_id>')
def notice_detail(notice_id):
    notice = Notice.query.filter_by(id=notice_id).first()

    return render_template('notice/detail.html', notice=notice)



@notice.route('/edit/<int:notice_id>', methods=['GET', 'POST'])
def edit_notice(notice_id):
    notice = Notice.query.get(notice_id)
    form = NoticeForm()
    if request.method == 'POST':
        if form.validate_on_submit():
            notice.title = form.title.data
            notice.content = form.content.data
            db.session.commit()
            return redirect(url_for('notice.notice_detail', notice_id=notice.id))
```

views.py

공지사항을 생성, 조회, 수정, 삭제하는 데 사용되는 코드로 다음과 같은 라우트를 지정.

/create: GET 요청으로 들어오면 새로운 공지사항을 생성하는 폼을 제공하고, POST 요청으로 들어오면 공지사항을 생성.

/index: 모든 공지사항을 조회하고 공지사항 목록을 보여주는 페이지를 렌더링.

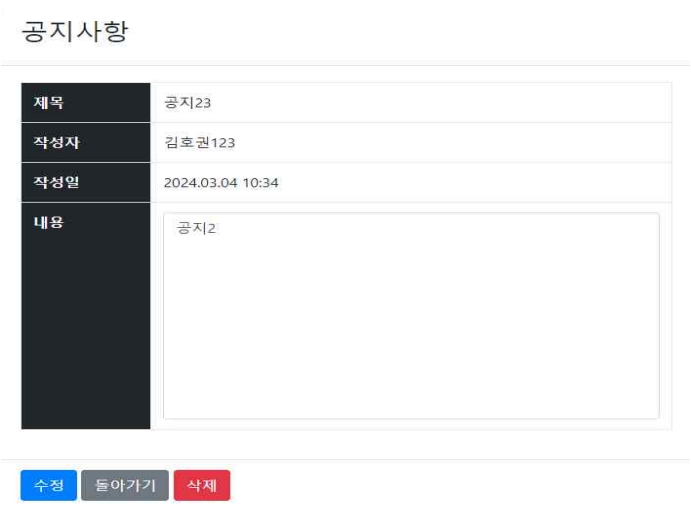
/detail/<int:notice_id>: 특정 공지사항의 상세 내용을 보여주는 페이지로 이동.

/edit/<int:notice_id>: 특정 공지사항을 수정. GET 요청으로 들어오면 수정 폼을 제공하고, POST 요청으로 들어오면 공지사항을 수정.

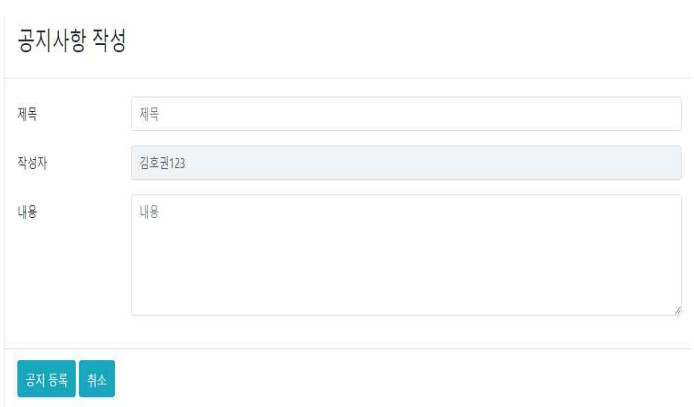
/delete/<int:notice_id>: 특정 공지사항을 삭제.

Flask-Login을 사용하여 현재 사용자를 추적하고, 사용자가 로그인한 경우 공지사항의 작성자로 설정.

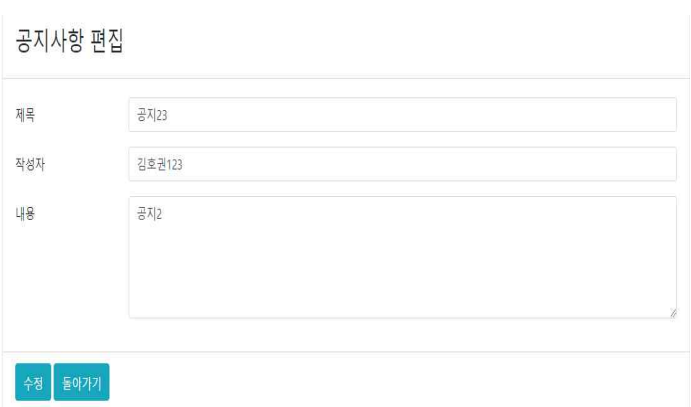
- 공지사항 게시판 -



- 공지사항 상세 내용 -



- 공지사항 작성 화면 -



- 공지사항 수정 화면 -

3. 견해

수업의 일환으로 플라스크에 대해 이해하고 그 일환으로 플라스크를 이용하여 이미지 감지 기능이 있는 홈페이지를 제작했다.

플라스크는 파이썬으로 웹 애플리케이션을 개발할 때 매우 유용한 마이크로 웹 프레임워크로 가볍고 유연한 설계로 개발자들에게 편의성을 제공한다. 간단하고 직관적인 구조로 인해 쉽게 학습하고 사용할 수 있다는 것으로 확장성이 뛰어나며 다양한 확장 기능을 사용할 수 있어서 많은 기업이 플라스크를 이용하여 제작하고 있다.

하지만 대규모 애플리케이션을 개발할 때는 다른 웹 프레임워크보다 일부 성능적인 제약이 있을 수 있으며 보안의 문제도 가지고 있다.

그러나 단점을 극복할 수 있으면 개발자들에게 많은 편의성을 제공해주므로 그래서 많은 개발자들이 쓰는 툴이라고 생각한다.