

Abstract geometric lines in the top left corner, consisting of several overlapping, irregular polygons and lines in a light beige color.

Introduction to flask

개요

- 웹 프레임워크

- » 개발자가 대부분의 웹 애플리케이션에 필요한 공통적인 저수준 구현을 직접 작성하지 않고 핵심적인 업무 로직 구현에 집중할 수 있도록 하는 라이브러리와 모듈의 집합

- 플라스크 (Flask)

- » 아민 로나허(Armin Ronacher)가 개발한 파이썬 기반 마이크로 웹 프레임워크
- » Werkzeug WSGI 툴킷과 Jinja2 템플릿 엔진 기반 웹 프레임워크
- » 애플리케이션을 간결하게 만들면서 확장 가능하도록 하는 것이 목표
 - › 데이터베이스 연동, 폼 유효성 검사와 같은 주요 기능을 자동으로 포함하지 않음
 - › 필요한 경우 추가할 수 있도록 확장 모듈 지원

개요

- Web Server Gateway Interface (WSGI)
 - » 파이썬 웹 애플리케이션 개발 표준
 - » 웹 서버와 웹 애플리케이션 사이의 범용 인터페이스에 대한 규칙(Specifications)
- Werkzeug
 - » 요청, 응답 객체 및 다양한 기능을 구현한 WSGI 툴킷
 - » 웹 프레임워크 구현의 기반
 - » 플라스크는 Werkzeug를 주요 라이브러리 중 하나로 포함
- 웹 템플릿 시스템
 - » 템플릿과 데이터 소스를 결합해서 동적인 웹 페이지를 생산하는 시스템
- Jinja2
 - » 파이썬을 위한 인기있는 웹 템플릿 엔진

개발 환경 구축

- Python 설치
 - » Miniconda 배포판 사용
 - » 별도 Python 설치 문서 참고
- Visual Studio Code
 - » 별도 Visual Studio Code 설치 문서 참고
- MySQL Database Management System
 - » 별도 MySQL DBMS 설치 문서 참고
- 가상 환경 만들기
 - » humanda5 가상 파이썬 환경 사용 (python version = 3.11.11, colab python version)
 - » 별도 Python 설치 문서 참고

개발 환경 구축

- Flask

- » pip install flask

```
D:\>pip install flask
Requirement already satisfied: flask in d:\apps\mini
Requirement already satisfied: Werkzeug>=3.1 in d:\ap
Requirement already satisfied: Jinja2>=3.1.2 in d:\ap
Requirement already satisfied: itsdangerous>=2.2 in d
Requirement already satisfied: click>=8.1.3 in d:\app
Requirement already satisfied: blinker>=1.9 in d:\app
Requirement already satisfied: colorama in d:\apps\mi
Requirement already satisfied: MarkupSafe>=2.0 in d:\
```

Hello, World 1

■ 코드 작성

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == "__main__":
    app.run()
```

■ 실행

» python hello_world.py

```
* Serving Flask app 'hello_world'
* Debug mode: off
WARNING: This is a development server. Do not use
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
□
```



Hello, World!

Hello, World 2

- 코드 작성 (디버그 모드)

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == "__main__":
    # app.debug = True
    # app.run()
    app.run(debug=True)
```

- 프로그램 실행 중 변경 사항 발생하면 즉시 적용 (hot reload)
- 오류 추적 기능이 있는 debugger 제공

- 실행

» python hello_world.py

```
* Serving Flask app 'hello_world'
* Debug mode: on
WARNING: This is a development server. Do not use
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 467-275-367
_
```

Hello, World 3

- 디렉터리 구조

```
└─ flask-examples
   └─ hello_world
      ├── __init__.py
      └── hello_world.py
```

- 코드 작성 (__init__.py)

```
from flask import Flask

def create_app():

    app = Flask(__name__)

    @app.route('/')
    def hello_world():
        return 'Hello, World!'

    return app
```

- 실행

» Flask --app **hello_world** --debug run



- Directory name
- Hello_world/__init__.py 실행

```
* Serving Flask app 'hello_world'
* Debug mode: on
WARNING: This is a development server. Do not use
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 467-275-367
```


라우팅

- 특정 웹 페이지 경로(URL)과 해당 요청을 실행하는 함수를 연결하는 설정
- route() decorator 사용

```
@app.route('/')  
def hello_world():  
    return 'Hello, World!'
```

- /hello 경로 요청이 발생하면
hello_world() 함수가 호출됨

라우팅

- url_for() 함수
 - » URL 대신 이름으로 URL을 대체하는 기능 제공

```
def hello_admin():  
    return 'Hello Admin'  
  
@app.route('/guest/<guest>')  
def hello_guest(guest):  
    return 'Hello %s as Guest' % guest  
  
@app.route('/user/<name>')  
def hello_user(name):  
    if name == 'admin':  
        return redirect(url_for('hello_admin'))  
    else:  
        return  
    redirect(url_for('hello_guest', guest = name))  
  
if __name__ == '__main__':  
    app.run(debug = True)
```

라우팅

- 동일한 URL에 요청 method별로 다른 메서드 매핑 가능
 - » 요청 메서드 → GET, POST, HEAD, PUT, DELETE

- 사례

```
@app.route('/success/<name>')
def success(name):
    return 'welcome %s' % name

@app.route('/login', methods = ['POST', 'GET'])
def login():
    if request.method == 'POST':
        user = request.form['nm']
        return redirect(url_for('success', name = user))
    else:
        user = request.args.get('nm')
        return redirect(url_for('success', name = user))
```

라우팅

- 변수를 사용하면 URL을 데이터와 결합하여 동적으로 매핑할 수 있음
- URL에서 변수는 <variable-name> 형식으로 표현 → 함수의 전달인자로 전달됨

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

@app.route('/hello/<name>')
def hello_name(name):
    return f'Hello, {name}!'

if __name__ == "__main__":
    app.run(debug=True)
```

← → ↻ ⓘ localhost:5000 hello/홍길동

Hello, 홍길동!

라우팅

- URL 변수에 자료형을 지정할 수 있음
 - » Int, float, path 등의 자료형 사용 가능

```
from flask import Flask

app = Flask(__name__)

@app.route('/blog/<int:postID>')
def show_blog(postID):
    return 'Blog Number %d' % postID

@app.route('/rev/<float:revNo>')
def revision(revNo):
    return 'Revision Number %f' % revNo

if __name__ == "__main__":
    app.run(debug=True)
```

블루프린트 (Blueprint)

- 웹 애플리케이션이 제공하는 기능이 많아지면 URL과 함수 매핑이 늘어나서 한 개의 파일에 관리하기 어려운 문제 발생
- 플라스크는 블루프린트(Blueprint)를 통해 기능을 종류별로 나누어서 별도의 파일에 관리할 수 있는 기능 제공

```
auth_bp = Blueprint("auth", __name__, url_prefix="/auth")

@auth_bp.route("/register/")
def register():
    pass

@auth_bp.route("/login/")
def login():
    pass

@auth_bp.route('/logout')
def logout():
    pass
```

/auth/register 요청과 매핑됨

/auth/login 요청과 매핑됨

/auth/logout 요청과 매핑됨

Blueprint 모듈 구현

```
app = Flask(__name__)

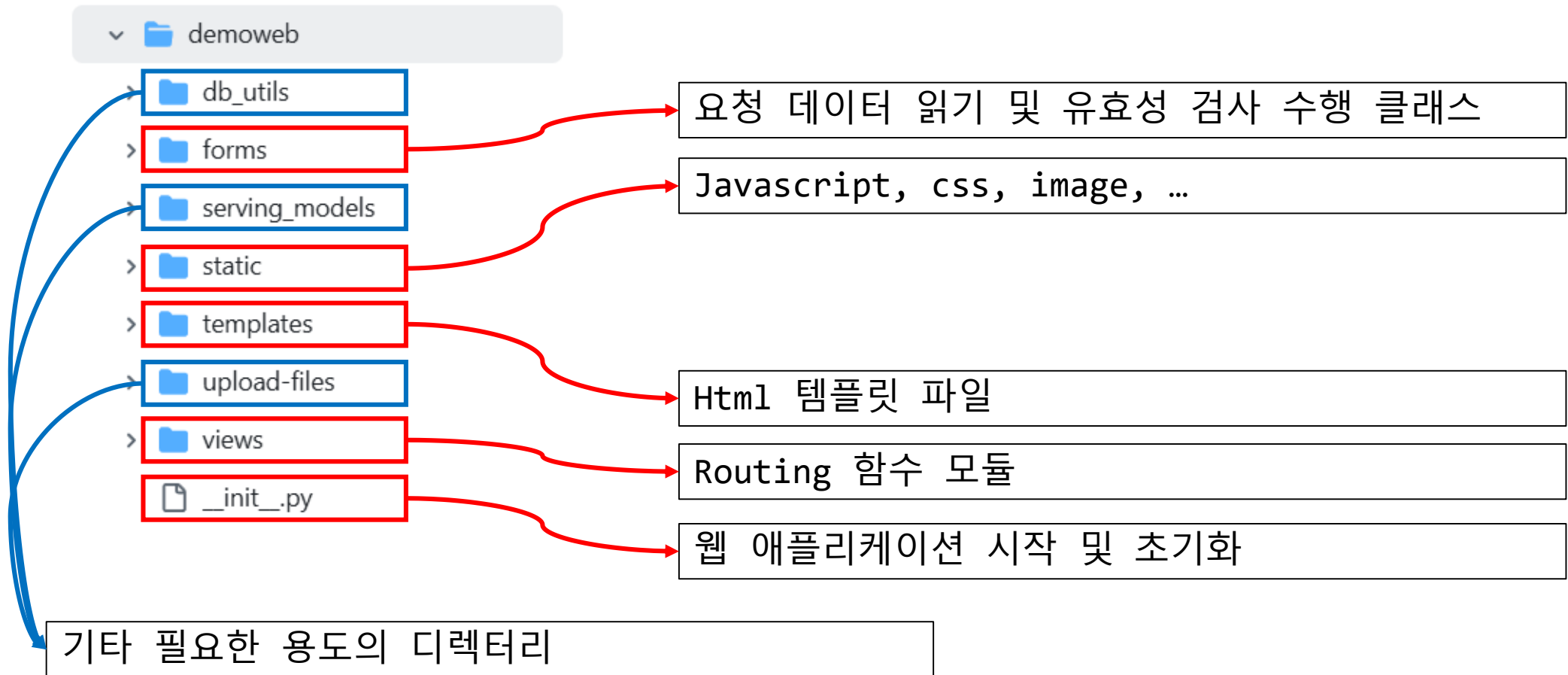
app.register_blueprint(auth_view.auth_bp)
```

Blueprint 등록

플라스크 애플리케이션 구조

- 웹 애플리케이션을 구성하는 각 용도별 파일을 별도의 디렉터리에 분리해서 관리

- 애플리케이션 디렉터리 구조



템플릿

- 요청 처리 메서드가 HTML 콘텐츠를 응답하도록 구현 1

```
@app.route('/hello/')  
def hello():  
    return '<html><body><h1>Hello World</h1></body></html>'
```

- 위와 같은 구현의 문제
 - » 복잡한 HTML 콘텐츠에 대해서는 적용하기 어려움
 - » 특해 HTML과 데이터를 결합해서 동적 콘텐츠를 생산하는 경우 적용하기 어려움
- 이런 문제를 해결하기 위해 플라스크는 Jinja2 템플릿 엔진 기반의 템플릿 시스템 제공

템플릿

- 메서드가 HTML 콘텐츠를 응답하도록 구현 2
 - » 독립된 파일에 작성된 HTML을 사용해서 HTML 응답 콘텐츠 사용
 - » 템플릿 구문 규칙을 통해 HTML에 동적으로 데이터 적용

```
@app.route('/hello/<user>')  
def hello_name(user):  
    return render_template('hello.html', name = user)
```

```
<!doctype html>  
<html>  
  <body>  
    <h1>Hello {{ name }}!</h1>  
  </body>  
</html>
```

- `render_template()` 함수 사용
- 첫 번째 전달인자는 HTML 파일의 경로
- 두 번째 전달인자는 HTML에 결합될 data

템플릿

- 파이썬 코드 삽입을 위한 template 구문

구문 형식	기능
{% ... %}	실행 구문 영역 (반복문, 선택문 등)
{{ ... }}	출력을 위해 데이터 또는 데이터를 만드는 연산 구문 영역
{# ... #}	주석 영역

```
<!doctype html>
<html>
  <body>
    {% if marks>50 %}
      <h1> Your result is pass!</h1>
    {% else %}
      <h1>Your result is fail</h1>
    {% endif %}
  </body>
</html>
```

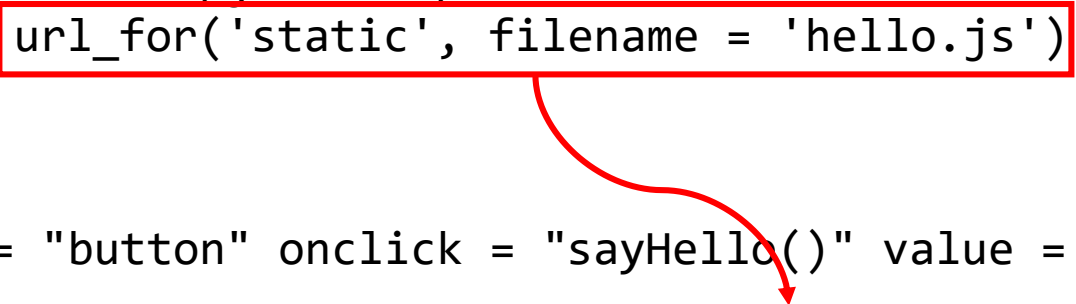
```
<!doctype html>
<html>
<body>
  <table border = 1>
    {% for key, value in result.items() %}
      <tr>
        <th> {{ key }} </th>
        <td> {{ value }} </td>
      </tr>
    {% endfor %}
  </table>
</body>
</html>
```

Static Files

- Javascript, css, image 등 동적 처리 기능이 없는 정적 파일
- 서비스 환경에서는 별도의 웹 서버에서 관리하지만 개발 환경에서는 웹 애플리케이션의 static 폴더에 저장 관리
- static 디렉터리에 저장된 파일은 /static/file-name 경로를 통해 접근

```
<html>
  <head>
    <script type = "text/javascript"
      src = "{{ url_for('static', filename = 'hello.js') }}" ></script>
    </head>

    <body>
      <input type = "button" onclick = "sayHello()" value = "Say Hello" />
    </body>
</html>
```



- <http://domain:port/static/hello.js>로 변환됨

- Static 디렉터리의 hello.js 파일로 연결됨

요청 데이터 읽기

- 플라스크는 웹 클라이언트(브라우저)에서 전송된 요청 데이터를 처리하여 전역 request 객체를 통해 view에서 읽을 수 있도록 제공
- 데이터는 종류별로 request 객체의 여러 속성에 딕셔너리 형식으로 저장됨

속성명	설명
form	Post 방식으로 전송된 데이터 저장
args	Get 방식으로 전송된 데이터 저장
cookies	Cookie 데이터 저장
files	enctype="multipart/form-data" 로 지정된 요청의 파일 저장

요청 데이터 읽기

■ 사례

```
<form action="{{ url_for('auth.login') }}" method="post">
  <div>
    <label>아이디</label>
    <input type="text" name="memberid">
  </div>
  <div>
    <label>패스워드</label>
    <input type="password" name="passwd">
  </div>
  <button type="submit">로그인</button>
</form>
```

```
@auth_bp.route("/login/")
def login():
    if request.method.lower() == 'get':
        pass
    else:
        member_id = request.form.get('memberid', '')
        passwd = request.form.get('passwd', '')
```

상태 관리 (State Management)

- HTTP 프로토콜은 모든 요청을 독립적인 개별 요청으로 처리
- 이로 인해 이전 요청과 관련된 정보를 이후의 요청 처리에서 사용할 수 없음
- 다수의 상태 유지 객체를 통해 상태 정보 유지 기능 제공



- 상태 유지 객체 종류

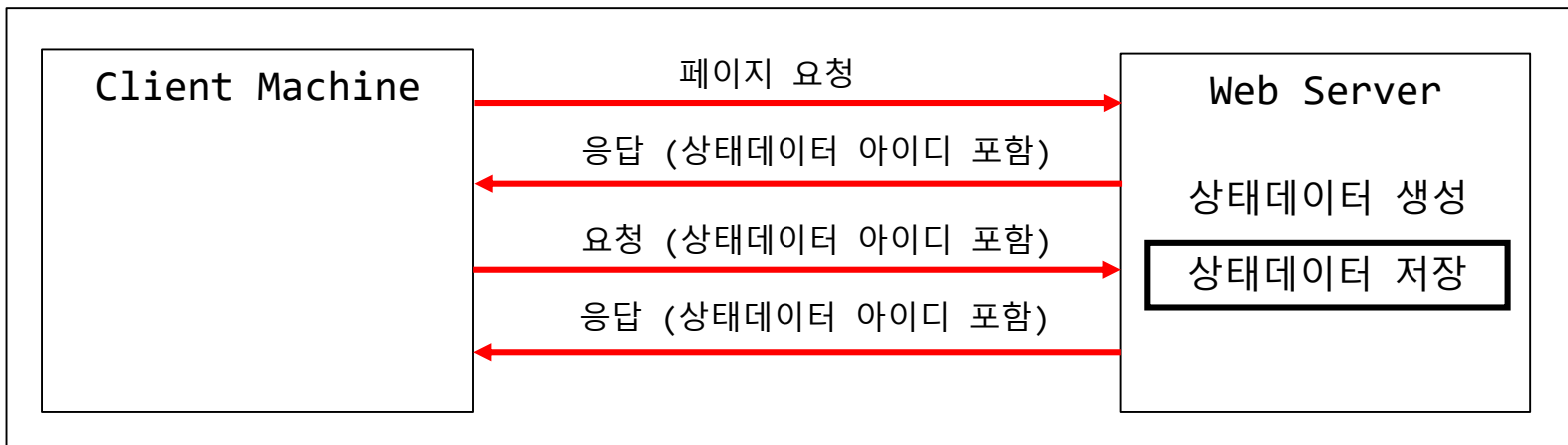
객체명	저장위치	유지범위
cookie	Browser / Client Machine	동일 브라우저/컴퓨터
session	Server	동일 브라우저

상태 유지 방법

▪ Client Side State Management



▪ Server Side State Management



Cookie 사용

■ 쿠키 생성

```
@app.route('/setcookie', methods = ['POST', 'GET'])
def setcookie():
    if request.method == 'POST':
        user = request.form['nm']

        resp = make_response(render_template('readcookie.html'))
        resp.set_cookie('userID', user)

    return resp
```

■ 쿠키 읽기

```
@app.route('/getcookie')
def getcookie():
    name = request.cookies.get('userID')
    return '<h1>welcome ' + name + '</h1>'
```


Session 사용

■ 세션 생성

```
@app.route('/')
def index():
    if 'username' in session:
        username = session['username']
        return f'Logged in as {username}'

    return "You are not logged in"
```

■ 세션 읽기

```
@app.route('/login', methods = ['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))

    return render_template('login.html')
```

■ 세션 제거

```
@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('index'))
```

■ Secret key 등록

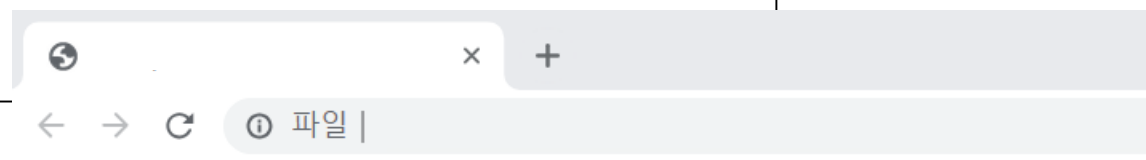
- » 세션 사용을 위한 필수 구현
- » app 객체에 등록 (__init__.py)

```
app = Flask(__name__)
app.secret_key = 'any random string'
```

File Upload

- 웹 요청을 통해 파일을 전송하기 위해서는 별도의 설정과 처리 과정 필요
- 파일 전송을 위한 브라우저 설정
 - » method는 POST 방식으로 지정
 - » enctype은 multipart/form-data로 설정
 - » `<input type='file'>` 입력 요소(파일 선택기) 포함

```
<form action="..." method="post" enctype="multipart/form-data">
  ...
  <input type="file" name="file1" />
</form>
```



File Selector Example

MESSAGE

ATTACHMENT 1 선택된 파일 없음

ATTACHMENT 2 선택된 파일 없음

File Upload

- 웹 애플리케이션에서 파일 읽기
 - » request 객체의 files 속성에 전송된 파일이 저장됨

```
@board_bp.route("/write/", methods=['POST'])
def write():

    attachment = request.files.get("file1")
    if attachment:
        ext = attachment.filename.split('.')[-1] # a/b/c.txt -> ['a/b/c/', 'txt'] -> 'txt'
        unique_filename = f'{uuid.uuid4().hex}.{ext}'
        bp_path = board_bp.root_path # Blueprint 경로 : 여기서는 views
        root_path = Path(bp_path).parent # 부모 경로
        upload_dir = os.path.join(root_path, "upload-files", unique_filename)
        attachment.save(upload_dir)

    return "succeeded to file upload"
```

File Download

- 브라우저는 수신된 콘텐츠를 처리할 수 없을 경우 파일 다운로드 방식으로 처리
 - » 수신된 콘텐츠의 내용 또는 응답 헤더의 mime-type을 통해 처리 방식 결정
- 다운로드 구현

```
@board_bp.route("/download/", methods=['GET'])
def download():
    savedfilename = request.args.get('savedfilename')

    bp_path = board_bp.root_path # Blueprint 경로 : 여기서는 views
    root_path = Path(bp_path).parent # 부모 경로
    upload_dir = os.path.join(root_path, "upload-files")

    return send_from_directory(upload_dir, savedfilename, as_attachment=True)
```