

7주차

리눅스 시스템

2024.동계계절학기

CONTENTS

1. 파이썬 플라스크
2. REST
3. 기타: `vscode - remote ssh`

- **플라스크란?**
 - 플라스크는 Python 기반의 웹 프레임워크
- **Flask의 특징 및 활용사례**
 - 빠르고 가벼운 웹 앱 개발이 가능하여 프로토타입 개발에 용이
 - RESTful API를 쉽게 개발할 수 있음
 - 간단한 웹사이트나 개인 프로젝트용(상업용 서비스는 Django같은 다른 프레임워크를 추천)
 - 교육 목적으로 기본 웹 개념 학습에 사용
- **웹 개발 기본 개념**
 - http: 서버와 웹 클라이언트(브라우저) 간 통신 프로토콜
 - 서버-클라이언트 모델
 - 클라이언트의 요청(Request)에 서버는 응답(Response)을 보냄
 - Flask는 서버 역할을 수행
 - RESTful 설계
 - REST 원칙에 따라 API를 설계
 - URL은 리소스를 나타내고 HTTP 메서드(GET, POST, PUT, DELETE 등)를 사용

플라스크 기본 사용

- **프로젝트 디렉터리 생성**
 - `$ mkdir flaskProject`
- **파이썬 가상환경 설정**
 - 가상환경을 사용하여 프로젝트간 종속성 충돌을 방지
 - 가상환경 생성
 - `$ python3 -m venv venv`
 - 가상환경 활성화
 - `$ source venv/bin/activate`
 - 가상환경 비활성화
 - `$ deactivate`
- **가상환경에 플라스크 설치**
 - 가상환경 활성화 상태에서
 - `(venv) $ pip install flask`

- **플라스크 프로젝트 기본구조**

```
flaskProject/  
├── static/           # CSS, JavaScript, 이미지 파일  
├── templates/        # HTML 템플릿  
├── app.py            # Flask 애플리케이션 코드  
├── venv/              # 가상 환경 디렉터리  
└── requirements.txt  # 필요한 패키지 목록
```

- **Requirements.txt 파일은**

- `$ pip freeze > requirements.txt` 로 생성
- `git`으로 프로젝트 공유 시 라이브러리나 패키지 전체를 공유하지 않고, 목록만 공유함 (`.gitignore` 활용)
- 다른 머신에서 프로젝트를 공유 받은 후 `$ pip install -r requirements.txt` 명령으로 패키지 다시 설치

Hello, Flask! -1

- 간단한 플라스크 앱

```
# app.py
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello, Flask!"

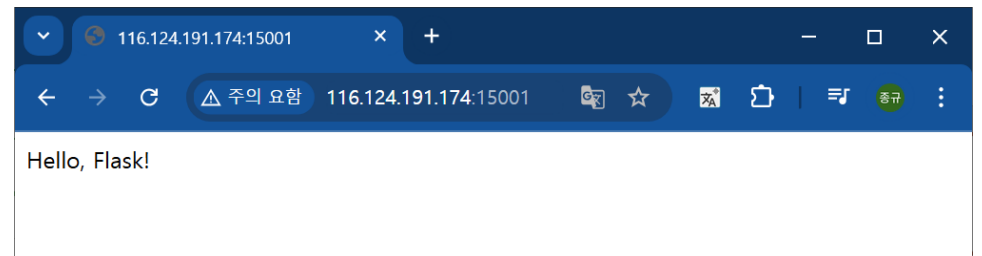
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=x, debug=True) // x는 본인에게 할당된 포트
```

- 실행 방법

- `$ python3 app.py`

- 접속 방법

- 브라우저 주소창에 아래 URL 입력, x는 본인에게 할당된 포트번호
 - `116.124.191.174:x`



Hello, Flask! -2

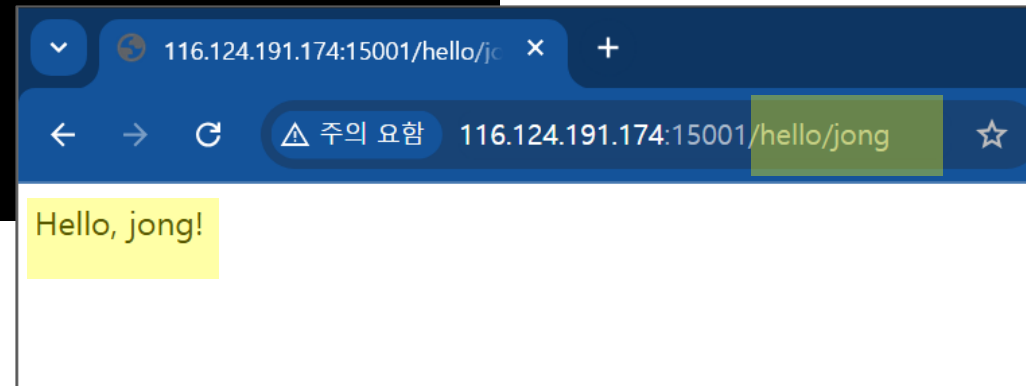
- 라우팅

- 라우팅은 URL 경로와 기능을 연결하는 과정

```
@app.route("/")
def home():
    return "Home"

@app.route("/hello")
def hello():
    return "Hello, Flask!"

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



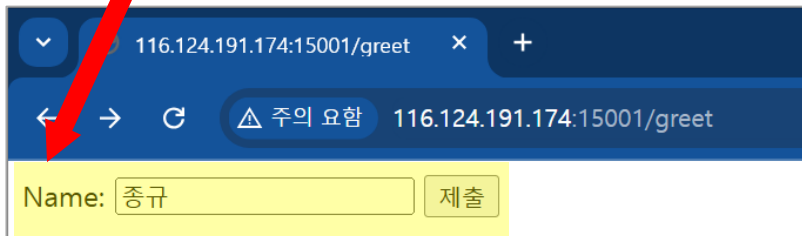
Hello, Flask! -3

- 사용자 요청과 응답 처리
 - GET(페이지 접근)과 POST(사용자 요청 전송) 활용

```
from flask import request

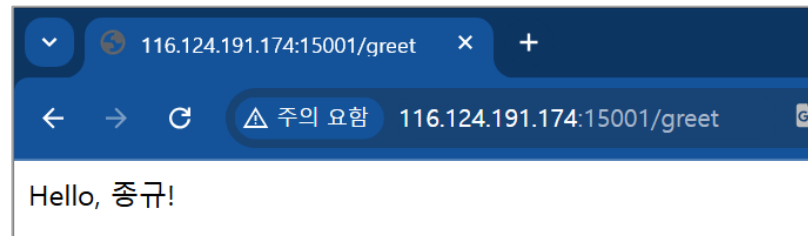
@app.route("/greet", methods=["GET", "POST"])
def greet():
    if request.method == "POST":
        name = request.form.get("name", "Guest")
        return f"Hello, {name}!"
    return '''
    <form method="post">
        Name: <input type="text" name="name">
        <input type="submit">
    </form>
    '''
```

1) 사용자가 116.124.191.174:15001/greet 접속 시 return에 있는 <form ...> </form> 부분을 브라우저에 전달 받음



116.124.191.174:15001/greet

Name: 제출



116.124.191.174:15001/greet

Hello, 종규!

Hello, Flask! -3

- 사용자 요청과 응답 처리
 - GET(페이지 접근)과 POST(사용자 요청 전송) 활용

```
from flask import request

@app.route("/greet", methods=["GET", "POST"])
def greet():
    if request.method == "POST":
        name = request.form.get("name", "Guest")
        return f"Hello, {name}!"
    return '''
    <form method="post">
        Name: <input type="text" name="name">
        <input type="submit">
    </form>
    '''
```

1) 사용자가 116.124.191.174:15001/greet 접속 시 return에 있는 <form ...> </form> 부분을 브라우저에 전달 받음

2) 브라우저에서 "제출" 버튼을 누르면 POST 형식으로 서버에 전송되고 서버의 if request.method=="POST" 부분이 동작함

116.124.191.174:15001/greet

Name:

116.124.191.174:15001/greet

Hello, 종규!

Hello, Flask! -4

- **HTML 파일 분리**
 - **HTML 템플릿 파일 (templates/index.html)**

```
<!DOCTYPE html>
<html>
<head>
  <title>Welcome</title>
</head>
<body>
  <h1>Hello, {{ name }}!</h1>
</body>
</html>
```

- **Flask 코드 (app.py)**

```
from flask import render_template

@app.route("/welcome/<name>")
def welcome(name):
    return render_template("index.html", name=name)
```

Hello, Flask! -5

- **html 파일에서 정적 파일(static file) 참조 방법**

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">
</head>
<body>
    <h1>Hello, Flask!</h1>
    <script src="{{ url_for('static', filename='script.js') }}"></script>
</body>
</html>
```

```
flaskProject/
|
|—— static/
|   |
|   |—— style.css
|   |—— script.js
|—— templates/
|   |—— index.html
```

Hello, Flask! -6

- 페이지 이동 실습
- `templates/home.html`

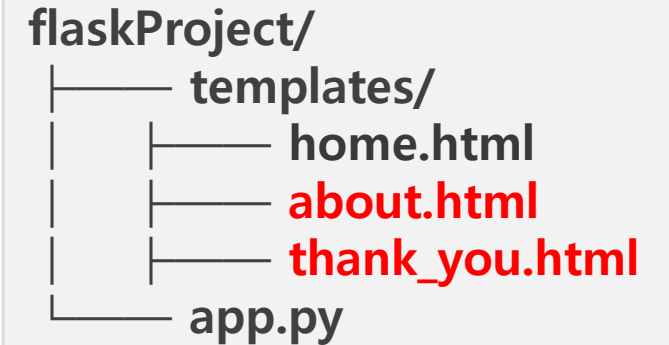
```
<!DOCTYPE html>
<html>
<head>
  <title>Home</title>
</head>
<body>
  <h1>Home Page</h1>
  <form action="/submit" method="post">
    <button type="submit">Submit</button>
  </form>
  <a href="{{ url_for('about') }}">Go to About Page</a>
</body>
</html>
```

```
flaskProject/
├── templates/
│   ├── home.html
│   ├── about.html
│   └── thank_you.html
└── app.py
```

Hello, Flask! -6

- **templates/about.html**

```
<!DOCTYPE html>
<html>
<head>
  <title>About</title>
</head>
<body>
  <h1>About Page</h1>
  <p>This is the about page.</p>
  <a href="{{ url_for('home') }}">Back to Home</a>
</body>
</html>
```



- **templates/thank_you.html**

```
<!DOCTYPE html>
<html>
<head>
  <title>Thank You</title>
</head>
<body>
  <h1>Thank You!</h1>
  <p>Your submission has been received.</p>
  <a href="{{ url_for('home') }}">Back to Home</a>
</body>
</html>
```

Hello, Flask! -6

- **app.py**

```
from flask import Flask, render_template, redirect, url_for

app = Flask(__name__)

@app.route("/")
def home():
    return render_template("home.html")

@app.route("/about")
def about():
    return render_template("about.html")

@app.route("/submit", methods=["POST"])
def submit():
    # 작업 처리 후 'Thank You' 페이지로 리다이렉트
    return redirect(url_for("thank_you"))

@app.route("/thank_you")
def thank_you():
    return render_template("thank_you.html")

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=15001)
```

```
flaskProject/
├── templates/
│   ├── home.html
│   ├── about.html
│   └── thank_you.html
└── app.py
```

- **REST(Representational State Transfer)**
 - **REST**는 웹을 위한 네트워크 아키텍처 스타일로 아래와 같은 특성을 가짐
 - **RESTful** 하다는 것은 REST 아키텍처 원칙을 준수한다는 것

특성	내용
Client-Server Architecture	클라이언트와 서버는 서로 독립적으로 작동해야 하며, 이를 통해 각각의 부분을 독립적으로 발전시킬 수 있음
Stateless	각 요청은 독립적이어야 하며, 서버는 클라이언트의 상태 정보를 저장하지 않아야 함. 모든 필요한 정보는 요청과 함께 전송되어야 함
Cacheable	클라이언트는 응답을 캐시할 수 있어야 하며, 서버 응답은 캐싱 가능 여부를 명시해야 함
Uniform Interface	인터페이스의 통일성을 통해 시스템의 각 부분을 독립적으로 개선할 수 있음. 이를 위해 리소스 식별, 자기 서술적 메시지, 하이퍼미디어 등의 원칙을 따름
Layered System	클라이언트는 서버가 직접적으로 연결되어 있는지, 중간 서버를 통해 연결되어 있는지 알 수 없어야 함
Code on Demand (Optional)	서버가 클라이언트에 실행 가능한 코드를 전송할 수 있어야 함(옵션)

REST (Stateless)

REST

- Stateless

- 클라이언트와 서버 간의 각 통신은 완전히 독립적이어야 하며, 서버는 클라이언트의 상태나 세션 정보를 저장하면 안 됨
- 대신, 각 요청은 필요한 모든 정보를 포함해야 하고, 서버는 그 요청만을 처리하여 응답해야 함
- 요청이 서로 의존적이지 않기 때문에, 각 요청은 이전의 다른 요청들로부터 독립적

능 마시던 걸로...

네! 능 마시던
봄베이로!



능 마시던 걸로...

그게 뭔데...?

- **State를 저장 안 함**

- **독립성**

- 각 요청이 필요한 모든 정보를 포함하므로, 서버는 이전 요청의 상태를 기억할 필요가 없어서 서버 설계를 단순화하고, 다른 서버로 요청을 자유롭게 전달할 수 있게 함

- **확장성**

- 서버가 상태 정보를 유지하지 않기 때문에, 어떤 서버에 요청이 배치되더라도 모든 서버가 동일하게 요청을 처리할 수 있음
 - 이는 서버 클러스터의 확장성을 증가시키며, 서비스의 가용성과 장애 허용성을 향상시킴

- **성능**

- 각 요청이 독립적으로 처리되므로, 서버는 상태를 관리하기 위한 추가적인 리소스를 사용할 필요가 없어 서버의 처리 성능을 최적화함

- **State를 저장함**

- **복잡성 증가**

- 서버가 클라이언트의 상태를 추적하려면 추가적인 로직과 저장 공간이 필요하여, 서버의 복잡성을 증가시키고, 오류 발생 가능성을 높임

- **확장성 제한**

- 서버가 상태 정보를 저장하면, 특정 클라이언트의 요청을 처리할 수 있는 서버가 제한될 수 있어 로드 밸런싱을 어렵게 만들고, 서버 확장을 복잡하게 할 수 있음

- **리소스 사용 증가**

- 상태 정보를 저장하고 관리하기 위해 추가 서버 리소스가 필요하므로 전체 시스템의 성능에 부담을 줌

세션과 쿠키

- 세션(Session)

- 세션은 서버 측에서 사용자 데이터를 저장
- 세션 ID는 일반적으로 쿠키를 통해 클라이언트에 저장되며, 이 ID를 사용하여 서버는 세션 데이터에 접근하고 사용자를 식별
- 세션은 주로 로그인 상태 유지에 사용

- 쿠키(Cookies)

- 클라이언트 측에서 작은 데이터 조각을 저장
- 이 데이터는 웹 브라우저에 저장되며, 사용자가 웹사이트를 다시 방문할 때마다 웹 서버로 전송
- 쿠키는 사용자의 선호, 세션 트래킹, 사용자 식별 정보 등을 저장

- 그럼 세션과 쿠키를 쓰면 RESTful 하지 않은가...?

- 그렇다
- 원칙적으로는 세션과 쿠키는 REST 아키텍처를 위반하는 것이지만, 사용자 편의를 위해 실제 서비스에서는 세션과 쿠키를 사용하고 있음
- 따라서 세션과 쿠키를 쓰더라도 RESTful 하다고 표현하는 경우가 많음

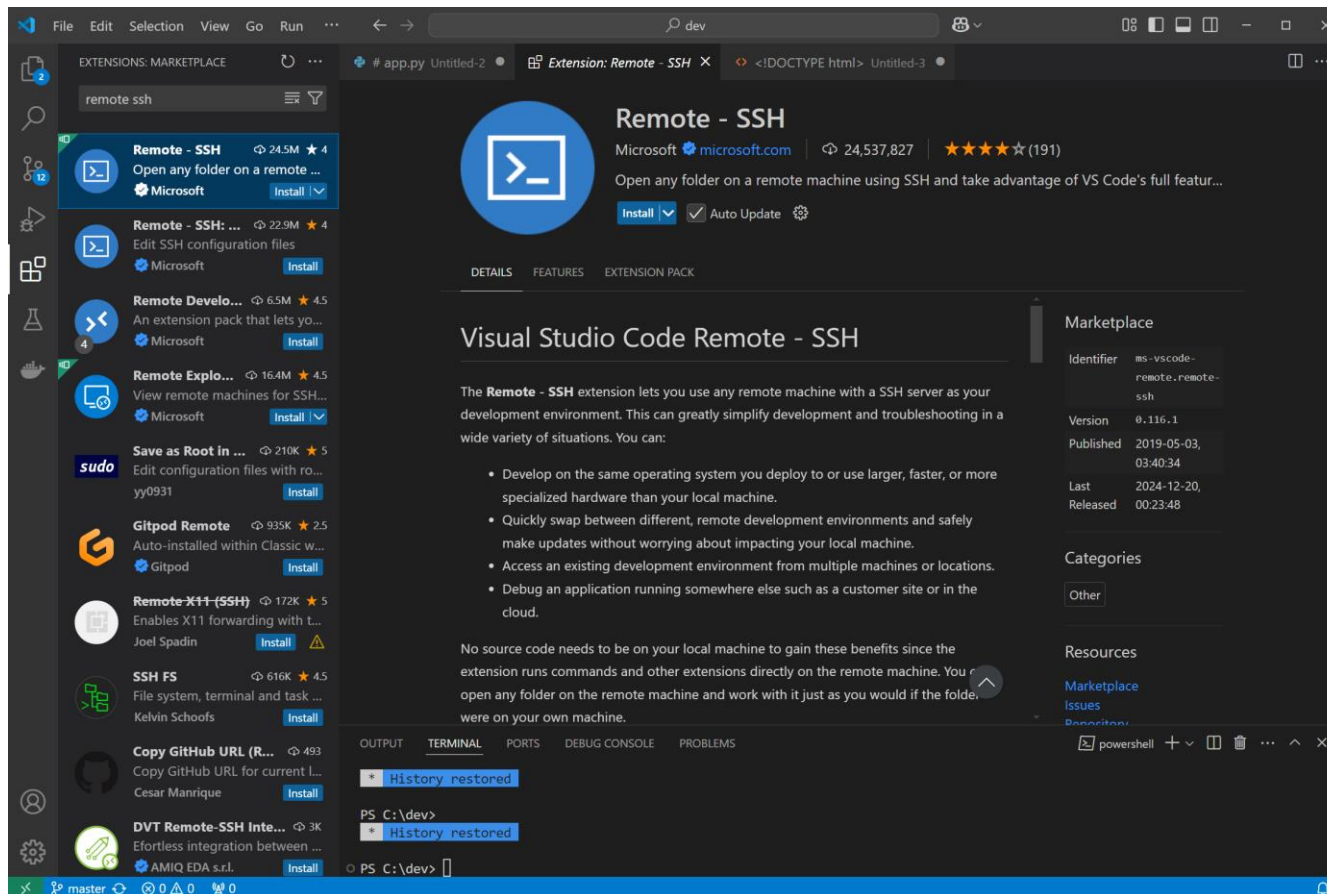


REST API

- REST API는 REST 아키텍처를 기반으로 한 API(Application Programming Interface)
- REST API의 구성요소
 - **GET** : 리소스를 조회
 - **POST** : 새 리소스를 생성
 - **PUT** : 리소스를 갱신
 - **DELETE** : 리소스를 삭제
- REST API 사용의 장점
 - **간단함과 범용성**: HTTP 프로토콜을 사용하므로, REST API는 이해하기 쉽고 사용하기 간편함
 - **언어와 플랫폼 독립적**: 어떤 프로그래밍 언어나 플랫폼에서도 사용할 수 있음
 - **확장성과 유연성**: 새로운 리소스나 메서드를 기존 시스템에 쉽게 추가할 수 있음

VSCode Remote SSH

- VSCode에서 SSH 연결을 통해 서버에 접속하여 바로 개발



VSCode

Remote SSH

1. VSCode 좌측 하단의 녹색 버튼을 눌러 "Connect to Host" 실행
2. + Add New SSH Host 실행
3. 그 이후 뜨는 명령어 창에 아래와 같이 저장
 - `ssh -p 10022 your_account@116.124.191.174`
4. 설정을 다 했으면 다시 "Connect to Host"를 실행하여 방금 추가한 서버에 접속
5. 암호까지 입력하고 나면 서버에 접속하여 사용 가능

