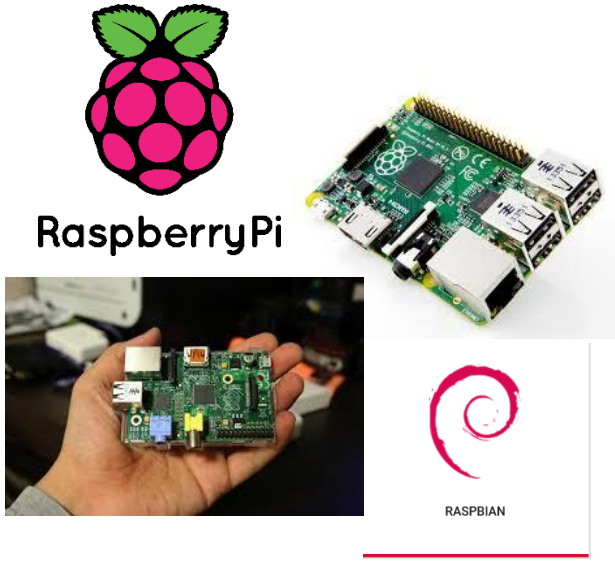
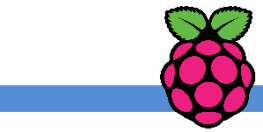
Sensor Programming
센서 프로그래밍

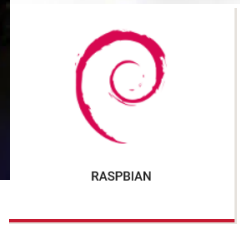


웹 기반 하드웨어 제어

Sensor Programming
센서 프로그래밍

웹 기반 하드웨어 제어
– 웹 소개





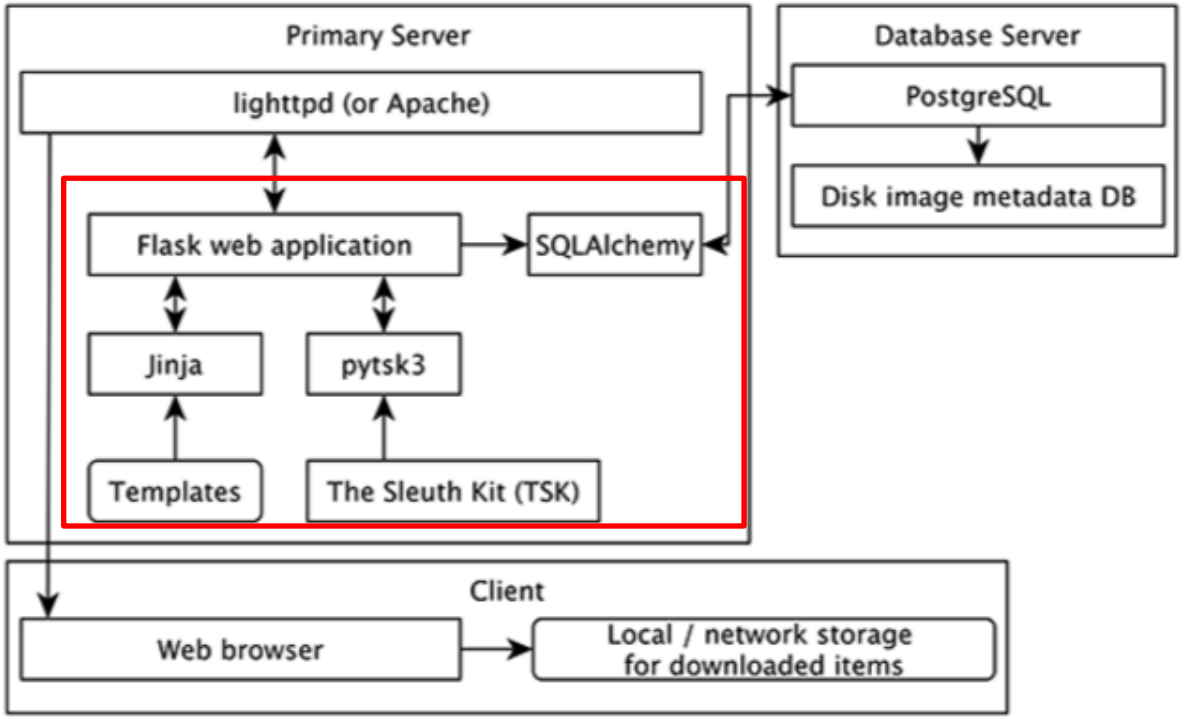
각 절에서 다루는 내용

1. 웹 서버 개요
2. Flask 웹 어플리케이션 설치
3. Flask 웹 어플리케이션 기본 골격
4. 라우팅 설정하기
5. Flask 에 HTML 연동
6. Flask 에 CSS 연동
7. Flask에 GPIO 연동

웹서버 개요

- 웹서버
  - 클라이언트가 브라우저를 통해서 요청하는 데이터를 HTTP 프로토콜을 사용해 제공하는 인터넷 서비스 프로그램 (apache, lighttpd, nginx)
- 웹 클리아언트
  - 대표적인 웹 브라우저로 크롬, firefox, explor등이 있음.
- HTML (Hyper Text Markup Language)
  - 클라이언트가 웹서버에게 데이터를 요청을 할 때 제공하는 정적인 문서를 기술하는 언어
- CSS (cascading style sheets)
  - HTML에서 디자인을 분리하고 여러 HTML로 기술한 문서에 일관된 스타일을 적용
  - 글자의 크기, 글자체, 줄 간격, 배경 색상, 배열 위치 등
- 웹 응용 프로그램
  - 동적인 웹 컨텐츠를 생성하는 프로그램
- 웹 응용 프레임워크
  - 웹 응용 프로그램의 개발에 필요한 서비스와 자원, API를 제공
  - Sping, node.js, Zend framework, bottle, flask 등등

FLASK 웹 응용 프레임워크



- 쉬운 웹 개발을 위해 서비스를 제공하는 마이크로 프레임워크 VS Django(full stack)
- Jinja를 통한 지원
  - WSGI : 파이썬 웹 서버 인터페이스 (웹 프로그램과 웹브라우저간 통신)
  - URL 라우팅을 지원
  - app.run() 함수로 웹 서버를 쉽게 구축하고 실행할 수 있음
- SQLAlchemy를 통한 데이터베이스의 접근을 제공
- TSK를 통한 파일시스템의 접근을 제공

Flask 설치 및 Flask 웹 어플리케이션 기본 골격

```
sudo apt-get install python3-flask
```

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def index():
    return 'Hello world'
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Flask 웹 어플리케이션 기본 골격

- from flask import Flask
  - flask 클래스를 가져옴
- app = Flask(__name__)
  - flask 객체 생성
  - _name__ : 현재 실행 중인 모듈의 이름을 전달, 임의이 문자열로 변경 가능
- Flask 객체
  - 웹 어플리케이션의 전반에 대해 영향을 끼치는 메인 객체
- @app.route('/')
  - URL /로 접속 시 실행할 함수를 지정, 현재는 index():로 지정됨
  - index()에서 return 한 hello world 문자열이 클라이언트(브라우저) 로 반환됨
- app.run(debug=True, host='0.0.0.0')
  - Flask 프레임 워크에 포함된 내장 웹 서버를 실행하는 코드
  - Host= ' 0.0.0.0' # 네트워크의 어떤 디바이스도 접근할 수 있다는 의미

라우팅 설정하기

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello flask!'

@app.route('/cakes')
def cakes():
    return 'Yummy cakes!'

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

- @app.route('/')
  - 웹 브라우저에서 http://127.0.0.1:5000/ 로 접속 시에 실행될 index 함수 지정
- @app.route('/cakes')
  - 웹 브라우저에서 http://127.0.0.1:5000/cakes 로 접속 시에 실행될 index 함수 지정

Flask에 html 연동

Templates

- Flask는 template 엔진(jinja2)을 사용하여 파이썬 소스파일과 html 문서를 분리하여 관리

```
webapp_dir
├──── app.py
└──── templates
    └──── index.html
```

```
$ mkdir webapp
$ cd webapp
$ mkdir templates
```

- webapp_dir 밑에 app.py 작성
- Templates디렉토리 밑에 index.html를 작성

Flask에 html 연동

index.html

```
<html>
<body>
<h1>Hello from a template!</h1>
{% if name %}
  <h1>Hello {{ name }}!</h1>
{% else %}
  <h1>Hello World!</h1>
{% endif %}
</body>
</html>
```

app.py

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html', name='jeon')

if __name__ == "__main__":
   app.run(host='0.0.0.0', port=80, debug=True)
```

Flask에 html 연동

- render_template
  - 웹 브라우져로 템플릿을 보내주기 위한 함수
  - 첫번째 인자 : 템플릿 파일(html)의 이름
  - 두번째 인자 : 템플릿에서 사용되는 변수명과 변수값을 전달

- {{ name }}
  - 파이썬 app에서 전달한 name 변수 값을 참조하여 출력

- 웹 어플리케이션 실행 (flask에 내장된 웹 서버도 함께 실행됨)

```
$ sudo python app.py
```

Flask 에 CSS 연동

- CSS 를 위한 static 경로와 style.css 파일 생성

```
webapp_dir
        ├──── app.py
        └──── templates
            └──── index.html
        └──── static
            └──── style.css
```

- style.css 편집

```
body {
    background: red;
    color: yellow;
}
```

Flask 에 CSS 연동

- style.css 편집

```
<html>
<head>
<link rel="stylesheet" href='/static/style.css' />
</head>
<body>
<h1>Hello from a template!</h1>
</body>
</html>
```

- 실행

```
$ sudo python3 app2.py
```

Flask에 gpio 연동 : app2.py

```
from flask import Flask, render_template
import datetime
import RPi.GPIO as GPIO
app = Flask(__name__)

GPIO.setmode(GPIO.BCM)

@app.route("/")
def hello():
   now = datetime.datetime.now()
   timeString = now.strftime("%Y-%m-%d %H:%M")
   templateData = {
      'title' : 'HELLO!',
      'time': timeString
   }
   return render_template('main.html', **templateData)
```

Flask에 gpio 연동 : app.py

```
@app.route("/readPin/<pin>")
def readPin(pin):
    try:
        GPIO.setup(int(pin), GPIO.IN)
        if GPIO.input(int(pin)) == True:
            response = "Pin number " + pin + " is high!"
        else:
            response = "Pin number " + pin + " is low!"
    except:
        response = "There was an error reading pin " + pin + "."

    templateData = {
        'title' : 'Status of Pin' + pin,
        'response' : response
        }

    return render_template('pin.html', **templateData)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

Flask에 gpio 연동 : main.html, pin.html

main.html

```
<!DOCTYPE html>
  <head>
    <title>{{ title }}</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <h2>The date and time on the server is: {{ time }}</h2>
  </body>
</html>
```

pin.html

```
<!DOCTYPE html>
  <head>
    <title>{{ title }}</title>
  </head>
  <body>
    <h1>Pin Status</h1>
    <h2>{{ response }}</h2>
  </body>
</html>
```

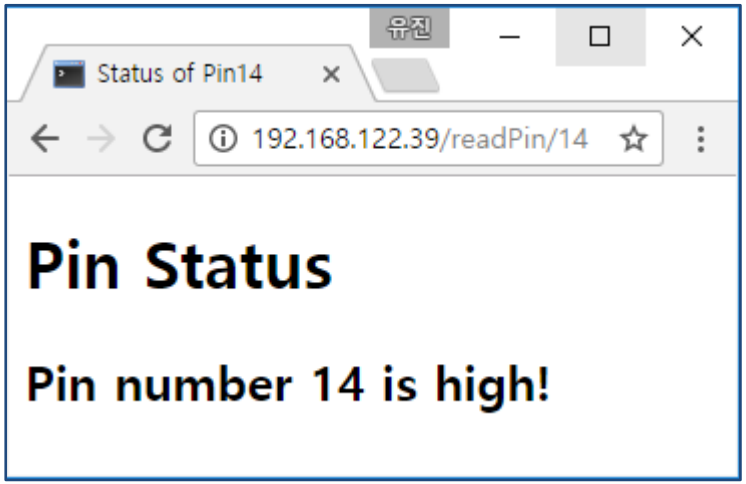
Flask에 gpio 연동

- @app.route("/readPin/<pin>")
  - 동적 라우팅을 추가
  - http://localhost/readPin/5를 호출시 gpio 5 값을 확인가능
  - 5 이외의 gpio number를 입력 가능함

```
templateData = {
        'title' : 'Status of Pin' + pin,
        'response' : response
}
return render_template('pin.html', **templateData)
```

```
<head>
    <title>{{ title }}</title>
</head>

<body>
    <h1>Pin Status</h1>
    <h2>{{ response }}</h2>
</body>
```



Sensor Programming
센서 프로그래밍

Thank you and Question?

RaspberryPi

RASPBIAN