1. **웹 프로그래밍의 이해**
   1. **웹 프로그래밍이란**

: HTTP(S) 프로토콜로 통신하는, 클라이언트와 서버를 개발하는 것.

장고 🡪 웹 서버 개발

웹 클라이언트 = 브라우저

* 1. **다양한 웹 클라이언트**

: 웹 브라우저 이외에도 웹 서버에 요청을 보내는 웹 클라이언트는 다양하게 만들 수 있다.

* 1. **HTTP 프로토콜**

: HTTP는 웹 서버와 웹 클라이언트 사이에 데이터를 주고받기 위해 사용하는 통신 규약으로 TCP/IP 프로토콜 위에서 동작한다. 🡪 웹 이용에는 TCP/IP 동작에 필수적인 **IP 주소**를 가져야 한다.

* + - **HTTP 메시지의 구조**

: 메시지는 요청(client 🡪 server) 메시지와 응답(server 🡪 client) 메시지가 있다.

|  |  |  |
| --- | --- | --- |
| 스타트라인  (Start Line) | 🡪 | 요청라인 or 상태라인  요청 응답 |
| 헤더  (Header) | 🡪 | 생략 가능 |
| 빈 줄  (Blank Line) | 🡪 | 헤더의 끝을 빈 줄로 인식 |
| 바디  (Body) | 🡪 | 생략 가능 |

헤더는 각 행의 끝에 줄 바꿈 문자인 CRLF가 있으며, 헤더와 바디는 빈 줄로 구분한다.

|  |  |  |
| --- | --- | --- |
| 바디가 없는 요청 메시지의 예 | | |
| GET /book/Shakespeare HTTP/1.1 | 🡪 | 요청 라인 : **요청방식, 요청 URL, 프로토콜 버전**으로 구성 |
| Host: www.example.com:8080 | 🡪 | 헤더 : “이름 : 값“ 형식으로 표현 |

Host는 필수로 표시해야 한다. 요청라인에 Host를 표시함으로써 헤더를 생략할 수도 있다.

|  |  |  |
| --- | --- | --- |
| 응답 메시지의 예 | | |
| HTTP/1.1 200 OK | 🡪 | 상태 라인 : **프로토콜 버전, 상태코드, 상태 텍스트**로 구성 |
| Content-Type: application/xhtml+xml;  charset=utf-8  <html>  …  </html> | 🡪 | 헤더 |

* + - **HTTP 처리 방식**

: HTTP는 HTTP 메소드를 통해서 클라이언트가 원하는 처리 방식을 서버에 알려준다.

|  |  |  |
| --- | --- | --- |
| **GET** | 리소스 **취득** | Read |
| **POST** | 리소스 **생성** 및 데이터 **추가** | Create |
| **PUT** | 리소스 **변경** | Update |
| **DELETE** | 리소스 **삭제** | Delete |
| HEAD | 리소스의 헤더(메타데이터) 취득 |  |
| OPTIONS | 리소스가 서포트하는 메소드 취득 |
| TRACE | 루프백 시험에 사용 |
| CONNET | 프록시 동작의 터널 접속으로 변경 |

* + - **GET과 POST 메소드**

: 가장 빈번하게 사용되는 메소드. 폼에서 사용자가 입력한 데이터들을 서버로 보낼 때 방식에 차이가 나타난다.

|  |  |
| --- | --- |
| **GET** 방식 | |
| GET http:// … .com/**?q=forms$release=1** HTTP/1.1 | URL 부분에 ? 뒤에 **이름=값** 쌍으로 이어 붙여 보낸다. |

|  |  |
| --- | --- |
| **POST**방식 | |
| POST http:// … .com/ HTTP/1.1  Content-Type: application/x-www-form-urlencoded  q=forms$release=1 | GET에서 URL에 포함시켰던 파라미터들을 요청 메시지의 바디에 넣는다. |

이러한 파라미터를 보내는 방식의 차이로 인하여 GET 방식은 많은 양의 데이터를 보내기 어렵다.(URL 길이의 제한, 보안상)

따라서 폼을 사용하거나 추가적인 데이터를 서버로 보내는 경우 POST방식을 많이 사용한다. (GET을 사용하는 경우도 존재)

* + - **상태코드**
  1. **URL 설계**

: 웹 애플리케이션을 개발할 때, 프로그램 로직 측면에서는 URL을 설계하게 된다.

즉 URL의 설계는 웹서버 로직 설계의 첫걸음이고, 사용자 또는 웹 클라이언트에게 웹 서버가 가지고 있는 기능을 명시해주는 중요한 단계이다.

로직 변경으로 인한 URL의 변경은 최소화될 수 있도록 유연한 설계가 중요하다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| URL 구성의 예 | | | | | | | | | |
| https | :// | example.com: | 80 | / | services | ? | category=2&kind=patents | # | n10 |
| URL 스킴 | | 호스트명 | PN |  | 경로 |  | 쿼리스트링 |  | 프라그먼트 |

* **URL 스킴** : URL에 사용된 프로토콜
* **호스트명** : 웹서버의 호스트명, 도메인명 또는 IP주소로 표현
* **포트번호** : 웹 서버 내의 서비스 포트번호. 생략 시 http:80, https:443을 디폴트로 갖는다.
* **경로** : 파일이나 애플리케이션 경로
* **쿼리스트링** : 질의 문자열, &로 구분된 이름=값 형식으로 표현
* **프라그먼트** : 문서 내의 앵커 등 조각을 지정
  + - **URL을 바라보는 측면**
* **URL은** 웹 클라이언트에서 **호출**된다는 시점에서 보면, 웹 서버에 존재하는 애플리케이션에 대한 **API**라고 할 수 있다. 이러한 API의 명명 규칙은 **URL을 보는 방식**에 따라 2가지로 구분할 수 있다.

1. **RPC** (**R**emote **P**rocedure **C**all)

**RPC**란 클라이언트가 네트워크상에서 **원격에 있는 서버가 제공하는 API 함수를 호출**하는 방식

REST 방식이 나오면서 빈도는 줄어들었지만, 여전히 많이 사용된다.

1. [**REST**](#REST)[1] (**R**epresentational **S**tate **T**ransfer)

**REST**이란 **리소스(자원)**를 **이름으로 구분**하여 해당 자원의 **상태를 주고받아** URL을 통해 **웹서버의 특정 리소스를 표현**한다는 개념.

**리소스 :** 웹 서버에 존재하는 **모든** **요소**

**리소스 상태에 대한 데이터**를 교환하는 이유 : 리소스는 시간이 지남에 따라 상태가 변할 수 있기 때문

**리소스에 대한 조작**을 **HTTP Method**(POST, GET, PUT, DELETE)를 통해 구분한다.

* + - **간편 URL**
* REST 방식의 URL개념을 기반으로 URL을 표현하면서 기존에 쿼리스프링을 위해 사용되던 특수 문자들(?=&#) 등을 제거하고 간결하게 만드는 방식이다. 🡪 쿼리스프링 없이 경로만 가진 간단한 구조의 URL
  1. **웹 애플리케이션 서버(WAS)\_**SW 측면의 서버

|  |  |  |
| --- | --- | --- |
| 구분 | **웹 서버(WS)** | **웹 애플리케이션 서버(WAS)** |
| 역할 | 웹 클라이언트의 요청을 받아서 요청을 처리하고, 그 결과를 웹 클라이언트에게 응답.  주로 **정적 페이지**(HTML, CSS 등)를 제공할 때 사용.  동적 페이지 처리가 필요하면 WAS에 넘긴다. | 웹 서버로부터 **동적 페이지** 요청을 받아서 요청을 처리하고, 그 결과를 웹서버로 반환한다.  주로 동적 페이지 생성을 위한 프로그램 실행과 DB 연동 기능을 처리. |
| 프로그램 명 | Apache httd, Nginx, IIS 등 | Apache Tomcat, uWSGI, JBoss 등 |

* + - **정적 페이지 VS 동적 페이지**

정적 페이지 : 항상 같은 내용을 보여주는 웹 페이지

동적 페이지 : 동일한 리소스의 요청이라도 조건에 따라 다른 내용을 반환하는 웹 페이지

초창기 웹은 정적이었지만 동적 페이지의 필요로 인해 별도의 프로그램이 필요하게 되고, 이 프로그램과 웹 서버 사이에 정보를 주고받는 규칙을 정의한 것이 **CGI**(Common Gateway Interface) 규격이다.

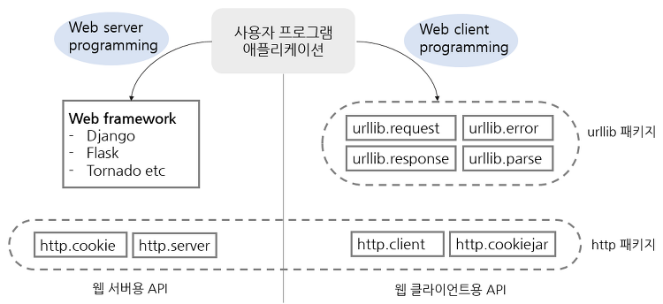
* + - **CGI 방식의 단점**
* CGI 자체는 웹 서버와 독립적인 프로그램 사이에 정보를 주고받는 규격을 의미하기 때문에, 이것을 준수하면 어떤 언어를 사용해도 CGI 프로그램을 개발할 수 있다. CGI 방식의 근본적인 문제는 각각의 클라이언트 요청에 대하여 독립적인 별도의 프로세스가 생성되고, 그로 인해 시스템에 많은 부하를 주는 요인이 된다는 것이다.
* 현재는 따라서 잘 쓰지 않는다.
  + - **CGI 방식의 대안 기술**

**1.** 별도의 애플리케이션을 Perl, PHP 등의 스크립트 언어로 작성하고, 스트립트를 처리하는 스트립트 엔진(인터프린터)을 웹서버에 내장시키는 것.

**2.** 애플리케이션을 처리하는 프로세스를 미리 데몬으로 기동시켜 놓은 후, 웹서버의 요청을 데몬에서 처리. 이 기술이 발전함에 따라 스레드 처리가 보강되고, 객체 지향 기술이 반영되면서 **애플리케이션 서버 방식으로 발전**했다.

* + - **애플리케이션 서버 방식**
* 애플리케이션 서버 방식은 WAS를 통해서 간접적으로 웹 애플리케이션 프로그램을 실행한다.
* 웹 서버는 정적 페이지 처리에 특화되어있고, 외에도 추가적인 기능을 제공한다.
* WAS는 더 큰 메모리를 사용하는 동적 페이지를 처리하며, 웹 서버보다 기능이 추가되고 종류도 다양해지고있다. (자바 – Tomcat, 루비 – Unicon, 파이썬 - uWSGI )
* 대다수의 WAS는 웹 서버의 기능을 제공하지만 상대적으로 성능과 안정성이 떨어져서 개발용으로만 제한적으로 사용된다.
  + - **웹 서버와의 역할 구분**

웹 서버와 WAS를 동일한 HW에서 구현하여 기동시키는 것도 하나의 방법이다. 하지만 최적의 성능을 위해선 따로 HW를 분리하는 것이 좋다.

1. **파이썬 웹 표준 라이브러리**
   1. **웹 라이브러리 구성**

* **urllib 패키지** : **웹 클라이언트**를 작성하는 데 사용되는 모듈들이 있으며, 가장 빈번하게 사용하는 모듈이다.
* **http 패키지** : 크게 Server용과 Client라이브러리로 구분하여 모듈을 담고 있다. 쿠키관련 라이브러리도 http 패키지 내에서 Server용과 Client용으로 구분되어 있다.

1. **urllib 패키지의 모듈**들은 HTTP Server뿐 아니라 FTP Server 및 Local File 등을 처리하는데 **Client에서 공통적으로 필요한 함수와 클래스 등을 제공**한다.

URL 처리와 서버 액세스 관련 API 제공 및 http.client 모듈의API를 더 추상화해서 고수준의 API 제공

1. **웹 프레임워크**는 사용자 프로그램과 저수준의 http.server 라이브러리 중간에 위치하여 **Web Server**의 **App 개발**을 좀 더 편리하게 해주면서, **표준 라이브러리의 기능을 확장해주는 역할**을 하고 있다.
   1. **Web Client 라이브러리**

* **웹 클라이언트** : **웹 서버에 요청을 보내는 APP** ex) 웹 브라우저
  + - **urllib.parse 모듈**
* URL의 분해, 조립, 변경 및 URL 문자 인코딩, 디코딩 등을 처리하는 함수를 제공
  + - **urllib.request 모듈**
* 주어진 URL에서 **데이터**를 가져오는 기본 기능을 제공
  + - **http.client 모듈**
* urllib.request 모듈도 http.client 모듈에서 제공하는 API를 사용해서 만들 모듈이다. 따라서 urllib.request 모듈보다 더 광범위하고 세밀한 기능을 처리할 수 있다.
* 순서 : 연결 객체 생성 🡪 요청을 보냄 🡪 응답 객체 생성 🡪 응답 데이터 읽음 🡪 연결을 닫음
  + - **urllib.request 모듈과 http.client 모듈 활용**

위 두 모듈을 활용하면 실제 서비스 가능한 웹 클라이언트를 만들 수 있다.

* 1. **Web Server 라이브러리**
* 프레임워크는 개발자가 웹 서버 프로그램을 개발하기 쉽도록 저수준의 기능을 이미 만들어 놓은 기반 프로그램으로 웹 서버 프로그램 개발자는 프레임워크를 활용하여 응용 로직만 개발하면 되기 때문에 훨씬 효율적이다.
* 하지만 웹 서버 라이브러리가 웹 프레임워크에 어떻게 사용되는지에 대한 기술을 파악할 필요도 있다.
  + - **간단한 Web Server**
* Web Server의 역할은 http 통신에서 클라이언트의 **요청을 받고** 이를 **처리**하여 **그 결과를 반환하는 것** .

|  |  |  |
| --- | --- | --- |
| *from http.server import HTTPServer, BaseHTTPRequestHandler* | | 1. http.server 모듈 import |
| *class MyHandler(BaseHTTPRequestHandler):*  *def do\_GET(self):*  *self.send\_response\_only(200,'OK')*  *self.send\_header('Content-Type', 'text/plain')*  *self.end\_headers()*  *self.wfile.write(b"Hello World")* | | 2. BaseHTTPRequestHandler를 상속받아 원하는 로직으로 핸들러 클래스 정의 |
| *if \_\_name\_\_ == '\_\_main\_\_':*  *server = HTTPServer(('', 8888), MyHandler)*  *print("Started WebServer on Port 8888...")*  *print("Press ^C to quit WebServer.")* | 3. 서버의 IP, PORT 및 핸들러 클래스를 인자로 하여 HTTPServer 객체 생성 | |
| *server.server\_forever()* | 4.HTTPServer 객체의 serve\_forever() 메소드를 호출 | |

* 웹서버는 일정한 규칙에 의해 설계된다.

|  |  |
| --- | --- |
| **HTTPServer** | 웹 서버를 만들기 위한 클래스, 서버 IP와 PORT를 [바인딩](#바인딩)[2]한다. |
| 핸들러 클래스가 반드시 필요. |
| **BaseHTTPRequestHandler** | 핸들러를 만들기 위한 기반 클래스, HTTP프로토콜 처리 로직이 들어있다. |
| 이 클래스를 상속받아 핸들러 클래스(자신의 로직 처리) 생성. |
| **SimpleHTTPRequestHandler** | BaseHTTPRequestHandler를 상속받아 만든 클래스 |
| GET과 HEAD 메소드 처리 |
| **CGIHTTPRequestHandler** | SimpleHTTPRequestHandler를 상속받아 만든 클래스 |
| 추가로 POST메소드와 CGI 처리 |

* + - **HTTPServer 및 BaseHTTPRequestHandler 클래스**
* 웹 서버를 만들기 위한 기반 클래스로서 HTTP 프로토콜을 처리해주는 기능이 있다. 따라서 관련 로직을 코딩하지 않아도 된다.
  + - **SimpleHTTPRequestHandler 클래스**
* 해당 클래스의 핸들러에는 GET과 HEAD 처리 메소드가 정의되어 있어서 별도의 코딩 없이 웹서버를 실행시킬 수 있다.
* POST 등 그 외의 HTTP메소드는 처리할 수 없다.
  + - **CGIHTTPRequestHandler 클래스**
* POST 방식을 처리할 수 있다. 하지만 CGI 처리 기능만 구현되어 있어서 모든 POST방식이 처리가능한 것은 아니다.
* SimpleHTTPRequestHandler 클래스를 상속받아 GET과 HEAD도 처리 가능
  1. **CGI / WSGI 라이브러리**
* 파이썬에는 WSGI 규격이 정의되어 있다. 이는 웹 서버와 웹 어플리케이션을 연결해주는 규격이고, 웹 프레임워크를 개발하거나 프레임워크를 아파치와 같은 웹 서버와 연동할 때 사용한다.
* WSGI 규격은 기존의 웹 CGI 기술의 단점을 개선하고 파이썬에 맞게 재구성한 것이다.
* **CGI**란 **Dynamic request**를 처리할 때 **웹 서버**와 **애플리케이션** 간에 데이터를 주고 받기위한 **규격**
  + - **CGI 관련 모듈**
* 파이썬 표준 라이브러리에는 cgi모듈과 cgitb 모듈을 제공하지만 WSGI 기술을 사용해 CGI를 처리하므로 cgi모듈을 사용할 일은 많지 않다.
  + - **WSGI 개요 [W**eb **S**erver **G**ateway **I**nterface**]**
* CGI 방식은 요청이 들어올 때마다 처리를 위한 프로세스가 생성된다. 따라서 서버에 과부화가 걸리면 서버에 부담이 커질 수밖에 없다.
* 이러한 CGI의 단점을 개선하기 위해 등장한 것이 WSGI 규격이다. 따라서 WSGI 규격만 맞추면 어떤 웹서버에서도 파이썬 App를 실행할 수 있다. 하지만 이러한 **일반 범용 웹 서버(Apache, Nginx)에는 WSGI 처리 기능이** 없다.

텍스트이(가) 표시된 사진

자동 생성된 설명따라서 **Web Server**와 **파이썬 Web App** 사이에는 WSGI 규격 처리를 위한 **WSGI 서버**(uWSGI 등)가 존재한다.

* + - **WSGI 서버의 APP 처리 과정**
* 웹서버에서 클라이언트의 요청을 받아 WSGI서버로 처리를 위임 🡪 WSGI 서버는 애플리케이션을 실행하여 그 결과를 웹 서버에게 반환 🡪 웹 서버는 클라이언트에게 응답
* WSGI 규격에 따라 애플리케이션을 개발할 때 중요 사항

1. 개발이 필요한 애플리케이션을 함수(클래스의 메소드)로 정의 및 규칙

*def App\_name(environ, start\_response):*

*environ* : 웹 프레임워크에 이미 정의돼 있다 HTTP 환경 변수를 포함한다.

*start\_response* : 응답을 시작하기 위해 반드시 호출해야 하는 함수

2. *start\_response* 함수 인자의 규칙

*start\_response(status, header)*

*status* : 응답 코드 및 응답 메시지를 지정

*header* : 응답 헤더를 지정

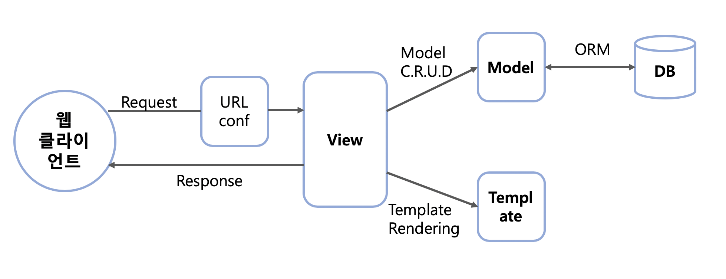
3. App 함수의 **return** 값은 응답 바디에 해당하는 내용이므로, **Iterable 타입**(리스트, 제네레이터 등 )이야한다.

* + - **wsgiref.simple\_server 모듈**
* 파이썬 표준 라이브러리에서 제공하는 wsgiref의 하위 모듈로, 웹 프레임워크 개발자가 웹 서버와의 연동 기능을 개발할 수 있도록 도와준다.
* 다만 모든 웹 프레임워크가 wsgiref 패키지를 사용하는 것은 아니다. 그저 WSGI 스펙에만 준하면 된다.

1. **Django**
   1. **장고에서의 애플리케이션 개발 방식**

* 웹 사이트를 설계할 때 가장 먼저 해야 할 일은 프로그램이 해야 **할 일을 적당한 크기로 나눠서 모듈화** 하는 것이다.
* 프로젝트 : 웹 사이트에 대한 전체 프로그램
* 애플케이션 : 모듈화된 단위 프로그램
* 장고는 기본적으로 MVT 패턴에 따라 애플리케이션을 개발하도록 유도한다.
* **MVT 패턴**
* MVC 패턴 : 데이터(model), UI(view), 로직(controller)을 구분하여 한 요소가 다른 요소들에 영향을 주지 않도록 설계하는 것.
* **MVT**는 MVC에서 **V**iew 🡪 **T**emplate, **C**ontroller 🡪 **V**iew 로 표현이 바뀐 것이다.

|  |  |
| --- | --- |
| **Model** | 데이터베이스에 저장되는 데이터 |
| **View** | 실직적으로 프로그램 로직이 동작하는 과정 수행 |
| **Template** | 사용자에게 보여지는 UI |



* **Model**
* 모델 : 상용될 데이터에 대한 정의를 담고 있는 클래스
* [ORM](#ORM) [3]기법으로 DB를 클래스로 매핑해서 사용한다.
* 장고는 테이블을 모델 클래스로 정의하고 이를 실제 데이터베이스에 반영한 후에도 테이블에 데이터를 입력하고 입력한 데이터를 확인 및 변경할 수 있는 여러가지 기능을 제공한다.
* **URLconf** 
  + URLconf : URL과 처리 함수(뷰)를 매핑하는 것
  + 클라이언트로부터 요청을 받으면 장고는 요청에 들어있는 URL이 정의된 URL 패턴과 매칭되는지를 분석한다.
  + URL 패턴을 정의할 때 <int : id> 같은 부분은 URL 패턴의 일부 문자열을 추출하기 위한 것이며 <type : name >형식으로 사용된다.

[ /detail/5 요청 🡪 해당 URL에 매칭되는 뷰 함수 view(request, id)의 Argument id에 5가 할당되어 호출. ]

* + URL을 좀 더 세밀하게 표현하고자 할 때는 [정규 표현식](#정규표현식)[4]을 사용한다.
* **View – 로직 정의**
  + 장고에서 뷰는 함수 또는 클래스의 메소드로 작성되며, 웹 요청을 받고 응답을 반환한다.
  1. **애플리케이션 설계하기**
  + 장고에서는 전체 프로그램을 프로젝트, 프로젝트 하위의 서브 프로그램을 애플리케이션이라고 한다.
  + 장고의 이런 특징은 하나의 애플리케이션을 여러 곳의 프로젝트에 재사용 함으로서 생산성을 높일 수 있게 한다.

1. **프로젝트 생성** 🡪 *Django-admin startproject* project\_name
2. **애플리케이션 생성** 🡪 *python manage.py startapp* app\_name
3. **프로젝트 설정 파일 변경**

3.1 ALLOWED\_HOSTS 🡪 default = [‘localhost’, ‘127.0.0.1’]

3.2 애플리케이션 등록 🡪 *app\_name* , *app\_name.apps.App\_nameConfig* [정확한 방법]

3.3 데이터베이스 엔진 설정

3.4 타임존 지정 🡪 ‘Asia/Seoul’

**4. 기본 테이블 생성** 🡪 *python manage.py migrate*

* 1. **애플리케이션 개발하기**
* **Model**

|  |  |
| --- | --- |
| *python manage.py makemigrations* | DB에 변경이 필요한 사항을 추출 |
| *python manage.py migrate* | DB에 변경사항을 반영 |

* + - 장고에서는 테이블을 하나의 클래스로 정의하고 테이블의 컬럼은 클래스의 변수로 매핑한다.

|  |  |
| --- | --- |
| *PK(****P****rimary* ***K****ey)* | 클래스에 지정하지 않아도 id라는 이름으로 자동으로 만든다. |
| *FK(****F****oreign* ***K****ey)* | 항상 다른 테이블의 PK에 연결된다. 실제 테이블에서 FK로 지정된 칼럼은 \_id 접미사가 붙는다. |
| *\_\_str\_\_()* | 객체를 문자열로 표현할 사용하는 함수 |

* + - 테이블 클래스는 *djnago.db.models.Model* 클래스를 상속받아 정의하고, 각 클래스 변수의 타입도 미리 정의된 필드 클래스를 사용한다.
    - Admin 사이트에 테이블 반영

|  |  |
| --- | --- |
| *admin.site.register(table\_name.register()* | admin.py 파일에 추가 |

* **View 및 Template**
  + - **URLconf 코딩**
      * URL 패턴 매칭은 위에서 아래로 진행하므로 순서에 유의한다.
      * path() 함수는 route, view 2개의 필수 인자와 kwargs, name 2개의 선택 인자를 받는다.

|  |  |
| --- | --- |
| ***route*** | URL 패턴을 표현하는 문자열 |
| ***view*** | URL 스트링이 매칭되면 호출되는 뷰 함수, HttpRequest 객체와 URL 스트링에서 추출된 항복이 뷰의 인자로 들어간다. |
| *kwargs* | URL 스트링에서 추출된 항목 외에 추가적인 인자(딕셔너리 타입)를 뷰 함수에 전달할 때 사용 |
| *name* | 각 URL 패턴별로 이름을 붙여준다. 이 이름은 템플릿 파일에서 주로 |

* **asd**

**Django**

1. 프로젝트 만들기
2. 장고 설치
3. 장고 프로젝트 만들기
4. 설정하기(데이터베이스,S3)
5. 데이터베이스 초기화
6. 관리자 계정 만들기
7. 앱 만들기
8. 모델 설계 (데이터베이스)
9. 뷰 만들기 (계산,처리)
10. 템플릿 만들기 (프론트)
11. URL 만들기

**model**

: 데이터 서비스를 제공하는 Layer. Django의 Model은 각 Django App안에 기본적으로 생성되는 **models.py 모듈 안에 정의하게** 된다. models.py 모듈 안에 하나 이상의 **모델 클래스를 정의할 수 있으며**, 하나의 모델 클래스는 **데이타베이스에서 하나의 테이블에 해당된다**.

1. **필드 타입 :** 모델의 필드에는 다양한 타입들이 있는데, 모든 필드 타입 클래스들은 **추상 클래스**인 "**Field**" 클래스의 **파생 클래스**들이다.

|  |  |
| --- | --- |
| **Char**Field | **제한된 문자열** 필드 타입. 최대 길이를 max\_length 옵션에 지정해야 한다.  이메일 주소를 체크(EmailField) IP 주소를 체크를 하는 (GenericIPAddressField) 콤마로 정수를 분리한 (CommaSeparatedIntegerField), 특정 폴더의 파일 패스를 표현하는 (FilePathField) URL을 표현하는 (URLField) |
| **Text**Field | **대용량 문자열**을 갖는 필드 |
| **Integer**Field | 32비트 **정수형 필드**. 정수 사이즈에 따라 BigIntegerField, SmallIntegerField |
| **Boolean**Field | **true/false 필드**. Null을 허용하기 위해서는 NullBooleanField를 사용 |
| **DateTime**Field | **날짜와 시간을** 갖는 필드. 날짜(DateField), 시간(TimeField) |
| **Decimal**Field | **소숫점을** 갖는 **decimal** 필드 |
| **Binary**Field | 바이너리 데이터를 저장하는 필드 |
| **File**Field | **파일 업로**드 필드 |
| **Image**Field | **이미지 파일**인지 체크 (FileField의 파생클래스) |
| **UUID**Field | **GUID (UUID)**를 저장하는 필드 |

**Session** : 클라이언트별 정보를 브라우저가 아닌 **웹서버에 저장하는 것**

* 클라이언트의 정보를 웹브라우저에 저장하는 것을 쿠키(cookie)라고 한다. 여기서 Session은 쿠키에 Session.id 만 저장하여 클라이언트와 웹서버간의 연결성을 확보하고, 이를 통해 커뮤니케이션한다.

**-용어 설명-**

**1.** **[REST](#REST_) 구성 요소**

1. **자원(Resource): URI**

* 모든 자원에 **고유한 ID가 존재**하고, 이 자원은 **Server**에 존재한다.
* **자원을 구별하는 ID**는 ‘/groups/:group\_id’와 같은 **HTTP URI** 다.
* Client는 URI를 이용해서 자원을 지정하고 해당 자원의 상태(정보)에 대한 조작을 Server에 요청한다.

1. **행위(Verb): HTTP Method**

* HTTP 프로토콜의 **Method**를 사용한다. [**GET**, **POST**, **PUT**, **DELETE**]

1. **표현(Representation of Resource)**

* Client가 자원의 상태(정보)에 대한 조작을 요청하면 **Server**는 이에 **적절한 응답을 보낸다**.
* REST에서 하나의 자원은 **JSON**, **XML**, TEXT, RSS 등 여러 형태의 **Representation**으로 나타내어질 수 있다.

**2.** **[바인딩](#바인딩_)(Binding) :** 바인딩이란 프로그램의 기본 단위가 가질 수 있는 속성 중에서 일부 필요한 속성만을 선택하여 연결해 주는 것

- 예) **변수를** 구성하는 식별자(이름), 자료형 속성, 하나 이상의 주소(참조), 자료값에 구체적인 값으로 **확정하는 것**.

**3.**[**ORM**](#ORM_)**[O**bject-**R**elational **M**apping**] :** ORM은 객체와 RDB를 연결해주는 역할을 한다.

- ORM에서는 데이터베이스 대신에 객체를 사용해 데이터를 처리할 수 있다.

**4.** **[정규 표현식](#정규표현식_)** : path()대신 re\_path() 함수를 사용

|  |  |
| --- | --- |
| **. (Dot)** | 모든 문자 하나(any single character) |
| **^ (Caret)** | 문자열의 시작 |
| **$** | 문자열의 끝 |
| **[ ]** | [ ]괄호에 있는 문자하나. 예를 들어 [akz]면 a or k or z. |
| **[^ ]** | [ ]괄호에 있는 문자 이외의 문자 하나. 예를 들어 [^ab]이면 a와 b 를 제외한 문자 하나. |
| **\*** | 0번 이상 반복. {0,}과 동일 |
| **+** | 1번 이상 반복. {1,}과 도일 |
| **?** | 0번 또는 1번 반복. {0,1}과 동일 |
| **{n}** | n번 반복 |
| **{m,n}** | 최소 m번에서 최대 n번까지 반복 |
| **|** | OR. 예를 들어 A|B 면 A 또는 B |
| **[a-z]** | a에서 z까지 임의의 문자. 즉 영문 소문자 한 개 |
| **\w** | 영문, 숫자 또는 밑줄(\_) 한개. [0-9a-zA-Z\_]과 동일 |
| **\d** | 숫자 한 개. [0-9]와 동일 |

- **메소드**

1. **쿼리셋**

데이터베이스로부터 데이터를 읽고, 필터를 걸거나 정렬할 수 있다.

get메소드나 filter를 사용해 원하는 데이터를 찾을 때는 "각 필드명\_\_옵션" 의 형태로 질의를 만들 수 있다.

해당 코드에 경우

1. iexact : 대소문자를 구분하지 않고 정확히 일치하는 데이터 찾기

2. lte : 같거나 작다.

3. gte : 같거나 크다.

**objects.get()**

:검색결과에 해당하는 **하나의 객체**만을 **반환**한다. 그래서 **unique한 값을 가지고 검색하는 것이 좋다**.

만일 결과로 여러 개의 객체가 조회되면 MultipleObjectsReturned 에러가 발생한다.

**objects.filter()**

:검색결과에 해당하는 **여러 개의 객체를 포함하는 QuerySet을 반환**한다. 키워드 검색이라 특정 조건을 만족하는 객체를 검색할 때 이용할 수 있다.

**XML [**E**X**tensible **M**arkup **L**anguage**]**

: XML은 HTML과 매우 비슷한 문자 기반의 마크업 언어이며 사람과 기계가 동시에 읽기 편한 구조로 되어 있다.

그러나 XML은 HTML처럼 데이터를 보여주는 목적이 아닌, **데이터**를 **저장**하고 **전달할 목적**으로만 만들어졌다.

또한, XML 태그는 HTML 태그처럼 미리 정의되어 있지 않고, 사용자가 직접 정의할 수 있다.

**JSON [J**ava**S**cript **O**bje ct **N**otation**]**

: 일반적으로 **서버에서 클라이언트로** **데이터를 보낼 때 사용하는 양식**. 과거 웹 초기 시절부터 사용되어 온 XML은 헤더와 태그 등의 여러 요소로 가독성이 떨어지고, 쓸데없이 용량을 잡아먹는다는 단점에 대응해 간결하고 통일된 양식으로 각광을 받고 있는 것이 **JSON**이다.

**AJAX [A**synchronous **J**avaScript **a**nd **X**ML**]**

: 자바스크립트를 이용해서 비동기식으로 XML을 이용하여 서버와 통신하는 방식

**HttpResponse** 객체로 응답을 만들 때 **Content-Disposition 값을 attachment 형식으로** 설정하면 브라우저는 이 응답을 **파일로 다운** 받는다.

**Django Shop 오류 해결  
1. Page Not Found (Fuck 404)**

**1.** config/urls.py에서 urlpatterns의 순서 문제. 🡪 admin는 무관하고, 기본경로(shop)가 app중 제일 아래에 위치

**2.** cart/views.detail에서 중복되는 이름의 html 템플릿에 연결 시 잘 못 찾아감 🡪 cart/templates의 detail.html의 이름을 중복되지 않게 cart\_detail.html 로 변경

**3.** cart\_detail.html에서 템플릿 문법인 {% with A = B %}에서 등호 양옆의 빈 공간으로 에러 🡪 {% with A=B %} 로 변경

**2. Order Process**

1. 결제 버튼 누름
2. *order\_create* 뷰로 이동
3. request.method가 POST 인 경우 created.html 로 이동
4. 하지만 GET이기 때문에 *create.html*로 이동
5. 데이터 입력 후, Place Order 버튼을 눌러 Submit을 보내면 자바스크립트로 인해 checkout.js 가 실행
6. *js.AjaxCreateOrder()*가 실행
7. 문제 발생 : *OrderCheckoutAjaxView* 뷰에서 발생 🡪 merchant\_order\_id = OrderTransaction.objects.create\_new(order=order, amount=amount) 오류

**해결 🡪**

1. order.models.OrderTransactionManager.create\_new() 🡪 해쉬 함수 코드에서 **오타 수정**(sha**l**🡪sha**1**)
2. order.views.OrderCheckoutAjaxView() 뷰에서 **amount**의 타입을 **str에서 Int로 변경**
3. checkout.js에서 IMP.request\_pay에 **[pg : 'html5\_inicis']** 추가 및 **samesite** 오류 해결을 위해 상단에 **[document.cookie = "safeCookie1=foo; SameSite=Lax";]** 코드 추가