**<애플리케이션 개발하기 - View 및 Template 코딩 (Ubuntu) >**

view는 애플리케이션의 로직을 만드는 곳으로 모델에서 필요한 정보를 받아와 로직을 구현하고 Template로 전달하는 역할을 한다.

URLconf 설계 - URL과 뷰 매핑

|  |  |  |
| --- | --- | --- |
| **URL 패턴** | **뷰 이름** | **뷰가 처리하는 내용** |
| /polls/ | index() | index.html 템플릿을 보여준다. |
| /polls/5/ | detail() | detail.html 템플릿을 보여준다. |
| /polls/5/vote/ | vote() | detail.html에 있는 폼을 POST 방식으로 처리한다. |
| /polls/5/results/ | results() | results.html 템플릿을 보여준다. |
| /admin/ | (장고 기능) | Admin 사이트를 보여준다(장고에서 기본으로 제공함.) |

**※ 명령어 실행 순서**

urls.py 작성 // URLconf 내용을 코딩

views.index() 함수 작성 // index.html 템플릿도 같이 작성

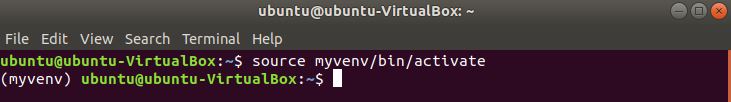
views.detail() 함수 작성 // detail.html 템플릿도 같이 작성

views.vote() 함수 작성 // 리다이렉션 처리 들어있음

views.results() 함수 작성 // results.html 템플릿도 같이 작성

**※ 모든 작업은 가상환경(virtualenv)안에서 진행됌**

리눅스 --> source myvenv/bin/activate

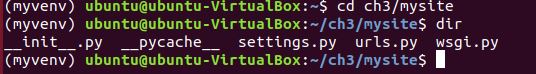


윈도우 --> myvenv\Scripts\activate

**1. URLconf 코딩**

**1-1. ch/mysite 안에서 하나의 urls.py 파일 작성**

$ cd ch3/mysite



$ vim urls.py



- 소스코드 설명

#1 : 필요한 모듈과 함수를 임포트. admin 모듈과 path() 함수는 장고에서 제공한다.

#2 : URL/뷰 매핑을 정의하는 방식은 동일. URL 패턴 매칭은 위에서 아래로 진행하므로, 정의하는 순서에 유의해야 한다.

#3 : Admin 사이트를 사용하기 위해서는 항상 이렇게 정의해야한다.

#4 : path() 함수는 route, view 2개의 필수 인자와 kwargs, name 2개의 선택 인자를 받는다.

▶ route : URL 패턴을 표현하는 문자열 --> URL 스트링

▶ view : URL 스트링이 매칭되면 호출되는 뷰 함수. HttpRequest 객체와 URL 스트링에서 추 출된 항목이 뷰 함수의 인자로 전달됌

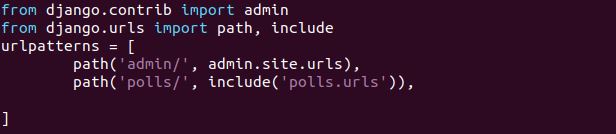
▶ kwargs : URL 스트링에서 추출된 항목 외에 추가적인 인자를 뷰 함수에 전달할 때, 파이썬 사전 타입으로 인자를 정의

▶ name : 각 URL 패턴별로 이름을 붙여줌

**1-2. mysite/urls.py와 polls/urls.py 2개의 파일 작성**

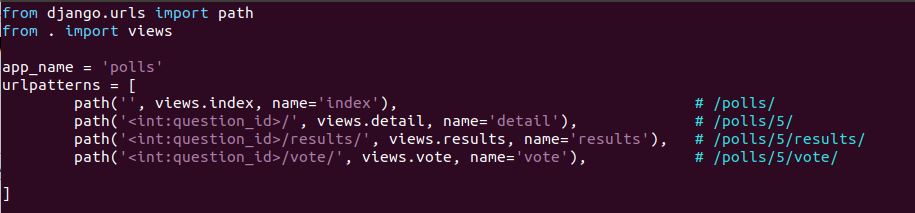
ch3/mysite/urls.py

$ vim urls.py



ch3/polls/urls.py

$ vim urls.py



URLconf 모듈을 계층적으로 구성하는 것이 쉬워지는 것과 확장이 용이해진다는 점에서 1-2가 더 좋은 방법이다.

**2. 뷰 함수 index() 및 템플릿 작성**

화면에 질문을 보여주는 로직을 구현하기 위해 뷰 함수와 템플릿 파일을 각각 코딩. 뷰 함수와 템플릿은 서로에게 영향을 미치기 때문에 같이 작업한다.

**2-1. 템플릿 파일 입력**

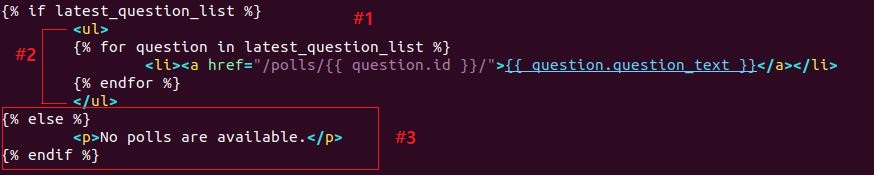
(새로운 디렉토리 생성)

$ mkdir templates

$ mkdir templates/polls

ch3/polls/templates/polls에서 index.html 작성

$ vim index.html



- 소스코드 설명

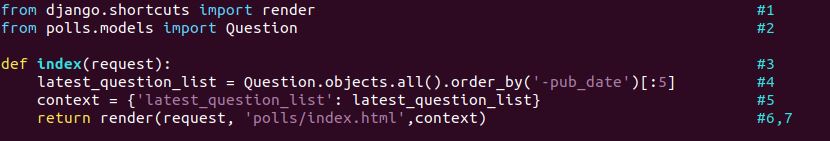
#1 : latest\_question\_list 객체는 index() 뷰 함수에서 넘겨주는 파라미터다.

#2 : latest\_question\_list 객체의 내용을 순회하면서 question\_text를 순서 없는 리스트로 화면에 보여준다.

#3 : 만약 latest\_question\_list 객체에 내용이 없다면, "No polls~" 문장을 화면에 보여준다.

**2-2. polls/view.py - index() 함수 작성**

$ vim views.py



- 소스코드 설명

#1 : 장고의 단축함수[[1]](#footnote-1)인 render() 함수를 임포트한다.

#2 : Question 테이블에 액세스하기 위해 polls.models.Question 클래스를 임포트한다.

#3 : 뷰 함수를 정의. request 객체는 뷰 함수의 필수 인자다.

#4 : 템플릿에게 넘겨줄 객체의 이름은 latest\_question\_list다. latest\_question\_list 객체는 Question 테이블 객체에서 pub\_date 컬럼의 역순으로 정렬하여 5개의 최근 Question 객체를 가져와 만든다.

#5 : 템플릿에서 넘겨주는 방식은 파이썬 사전 타입으로, 템플릿에 사용될 변수명과 그 변수명에 해당하는 객체를 매핑하는 사전으로 context 변수를 만들어 이를 render() 함수에 보내준다.

#6 : render() 함수는 템플릿 파일인 polls/index.html에 context 변수를 적용하여 사용자에게 보여줄 최종 HTML 텍스트를 만들고, 이를 담아서 HttpResponse 객체를 반환한다.

#7 : index() 뷰 함수는 최종적으로 클라이언트에게 응답할 데이터인 HttpResponse 객체를 반환한다.

**3. 뷰 함수 detail() 및 폼 템플릿 작성**

3개의 질문 중 하나를 선택했을 때, 질문에 대한 답변 항목을 보여주고 투표하도록 하는 화면을 만드는 템플릿 파일 코딩.

**3-1. <form> 태그 추가[[2]](#footnote-2)**

ch3/polls/templates/polls에서 작업

$ vim detail.html



- 소스코드 설명

#1 : 질문을 제목으로 사용하기 위해 폰트 크기를 <h1>로 지정한다.

#2 : 에러가 있으면 에러 메시지를 굵은 글자로 화면에 보여준다. 에러를 체크하는 로직은 #3에 vote() 뷰 함수에 있다. vote() 함수에서 익셉션이 발생하면 error\_message를 담아서 detail.html 템플릿을 렌더링하고, 그에 따라 지금 보고 있는 detail.html 템플릿에서 에러 메시지를 굵은 글자로 보여주게 된다.

#3 : 폼에 입력된 데이터는 POST 방식으로 보낸다. 서버 측의 데이터를 변경하는 경우, 일반적으로 GET이 아닌 POST 방식을 사용한다.

#4 : 폼을 처리하는 경우, 보안 측면에서 CSRF 공격을 주의해야 한다. 장고에서는 {% csrf\_token %} 템플릿 태그를 사용하여 공격을 방지한다. <form>태그 다음에 추가한다.

#5 : {% for %} 태그로 뷰 함수에서 넘어온 객체를 순회하고 있다. question.choice\_set.all은 Queion 객체의 choice\_set 속성에 들어있는 항목 모두를 뜻한다.

#6 : 해당 라디오 버튼을 선택하면 POST 데이터가 'choice'='3'(choice.id) 형태로 구성되도록 name과 value 속성을 정의한다.

#7 : forloop.counter 변수는 for 루프를 실행한 횟수를 담고 있는 템플릿 변수다. <lavel for> 속성과 <input id> 속성 값이 같아야 서로 바인딩된다.

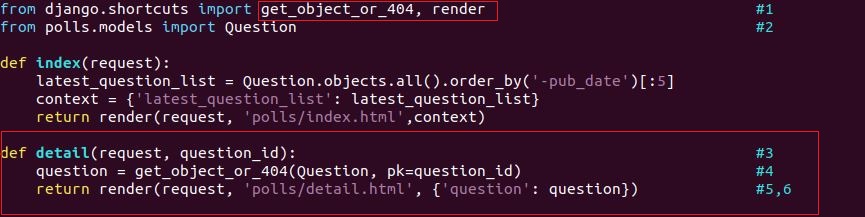
#8 : Vote 버튼을 클릭하면 사용자가 선택한 폼 데이터가 POST 방식으로 polls:vote URL로 전송된다.

**3-2. detail() 함수 작성**

기존의 index() 함수에서 추가

ch3/polls에서 작업

$ vim views.py



- 소스코드 설명

#1 : 기존의 있던 라인에 장고의 단축함수인 get\_object\_or\_404() 함수를 임포트하도록 함수명만 추가한다.

#2 : 기존 라인은 그대로 둔다.

#3 : 뷰 함수 정의. request 객체는 필수 인자이고, 추가적으로 question\_id 인자를 더 받는다.

#4 : get\_object\_or\_404() 단추함수 사용. Question 모델 클래스로부터 pk=question\_id 검색 조건에 맞는 객체를 조회한다. 조건에 맞는 객체가 없으면 Http404 익셉션을 발생시킨다.

#5 : 앞에서 사용했던 render() 함수를 다시 사용함. 템플릿 파일 polls/detail.html에 컨텍스트 변수를 적용하여 사용자에게 보여줄 최종 HTML 텍스트를 만들고, 이를 담아서 HttpResponse 객체를 반환한다.

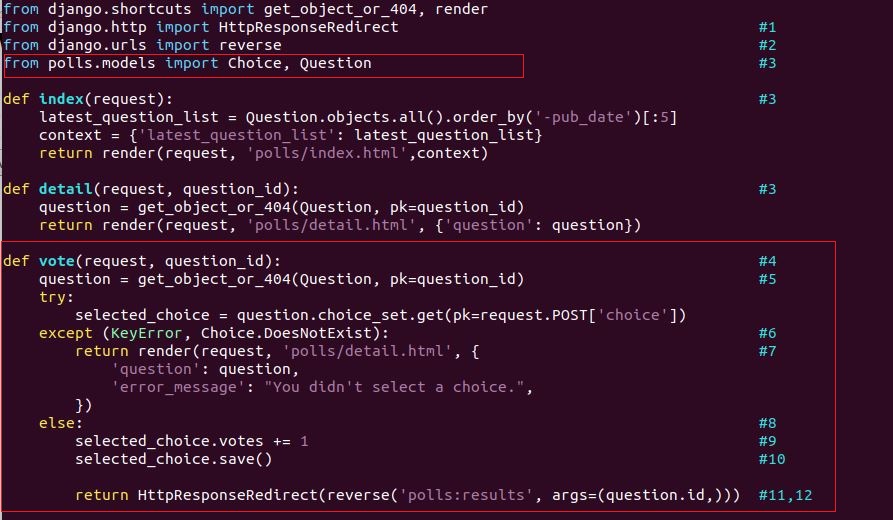
#6 : detail() 뷰 함수는 최종적으로 detail.html의 텍스트 데이터를 담은 HttpResponse 객체를 반환한다.

**4. 뷰 함수 vote() 및 리다이렉션 작성**

vote() 뷰 함수의 호출과 연계된 URL은 detail.html 템플릿 파일에서 받는다.

ch3/polls에서 작업

$ vim views.py



- 소스코드 설명 (오타주의)

#1 : 리다이렉트 기능이 필요함으로 HttpResponseRedirect 클래스를 임포트한다.

#2 : url 처리를 위해 reverse() 함수를 임포트한다.

#3 : 기존 라인은 그대로 둔다.

#4 : 뷰 함수 정의. request 객체는 필수 인자이고, detail() 뷰처럼 question\_id 인자를 더 받는다.

#5 : get\_object\_or\_404() 단축함수 사용. choice 테이블을 검색하고 있다.

검색 조건은 pk=request.POST['choice']다. request.POST['choice']는 폼 데이터에서 키가 'choice에 해당하는 값인 choice.id를 스트링으로 리턴한다.

#6 : 폼의 POST 데이터에서 'choice'라는 키가 없으면 KeyError 익셉션을 발생시킨다. 또는 검색 조건에 맞는 객체가 없으면 Choice.DoesNotExist 익셉션이 발생한다.

#7 : 익셉션이 발생하면 render() 함수에 의해 question과 error\_message 컨텍스트 변수를 detail.html 템플릿으로 전달한다. 사용자에게는 에러 메시지와 함께 질문 항목 폼을 다시 보여줘서 데이터를 재입력할 수 있도록 한다.

#8 : 다음은 익셉션이 발생하지 않고 정상 처리하는 경우다.

#9 : Choice 객체.votes 속성, 즉 선택 카운트를 +1 증가시킨다.

#10 : 변경사항을 해당 Choice 테이블에 저장한다.

#11 : HttpResponseRedirect 클래스의 생성자는 리다이렉트할 타겟 URL을 인자로 받는다. 타겟 URL은 reverse() 함수로 만든다.

#12 : 최종적으로 vote() 뷰 함수는 리다이렉트할 타겟 URL을 담은 HttpResponseRedirect 객체를 반환한다.

웹 프로그램에서 POST 방식의 폼 데이터를 처리하는 경우, 그 결과를 보여줄 수 있는 페이지로 이동시키기 위해 HttpResponseRedirect 객체를 리턴하는 것이 일반적이다.

**5. 뷰 함수 results() 및 템플릿 작성**

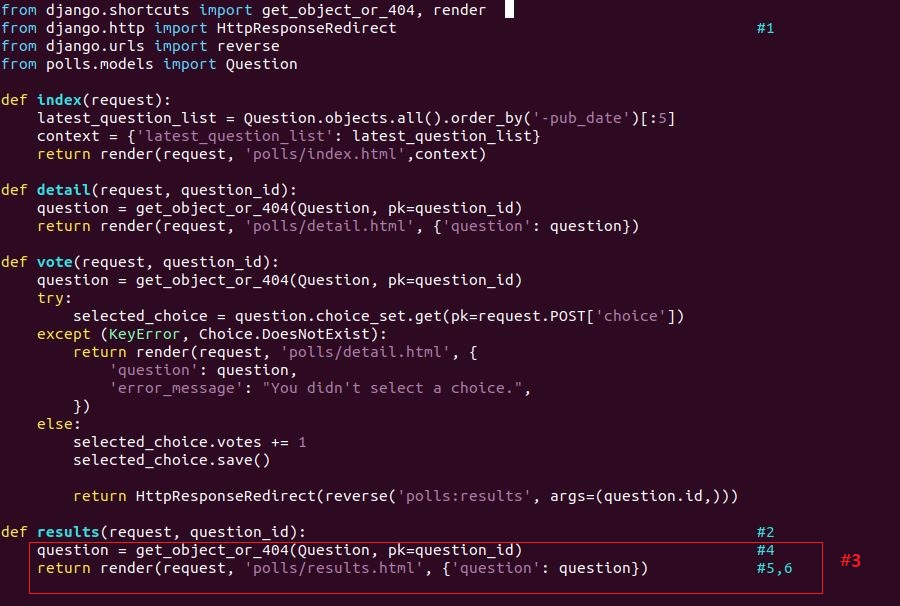
results() 뷰 함수의 호출과 연계된 URL은 votes() 뷰 함수의 리다이렉트 결과로 받는다.

**5-1. results() 함수 작성**

view.py 파일을 다시 열고, 기존의 내용에 results() 뷰 함수의 내용을 추가한다.

ch3/polls에서 작업

$ vim views.py



- 소스코드 설명

#1 : 기존 라인들은 그대로 둔다.

#2 : 뷰 함수 정의. 첫 번째 request 객체는 필수 인자이고, 두 번째 question\_id 인자는 다음 라인의 실행 결과로 넘어온다.

path('polls/<int:question\_id>/results/', views.results, name='results'),

#3 : 두 라인은 템플릿 파일명만 제외하고 detail() 함수와 동일하다.

#4 : get\_object\_or\_404() 단축함수 사용. Question 모델 클래스로부터 pk=question\_id 검색 조건에 맞는 객체를 조회한다. 조건에 맞는 객체가 없으면 Http404 익셉션을 발생시킨다.

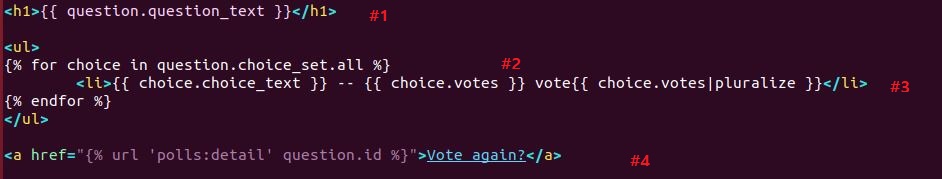
#5 : render() 함수 재사용. 템플릿으로 question 변수를 넘겨주는 것은 동일하지만 템플릿 파일이 다르므로, 사용자에게 보여주는 화면은 달라진다.

#6 : results() 뷰 함수는 최종적으로 results.html 템플릿 코드를 렌더링한 결과인 HTML 텍스트 데이터를 담은 HttpResponse 객체를 반환한다.

**5-2. results.html 작성**

ch3/polls/templates/polls 에서 작업

$ vim results.html



- 소스코드 설명

#1 : question 템플릿 변수는 results() 뷰 함수에서 넘겨준 컨텍스트 변수로, Quetion 클래스 타입의 객체를 담고 있다.

#2 : 뷰 함수에서 넘어온 question 변수를 이용하여 {% for %} 태그로 Choice 객체를 순회하고 있다.

quesion.choice\_set.all의 의미는 Question 객체의 choice\_set 속성에 들어있는 항목, 즉 Choice 객체 모두를 뜻한다.

#3 : Choice 객체의 choice\_text를 순서 없는 리스트로 화면에 보여준다. 또한, 각 텍스트 옆에 투표 카운트(choice.votes)를 숫자로 보여준다.

#4 : Vote again?이라는 문장을 보여주고, 그 문장에 URL 링크를 연결한다.

**6. 지금까지 작업 확인하기**

runserver 실행

cd ~/ch3에서 확인

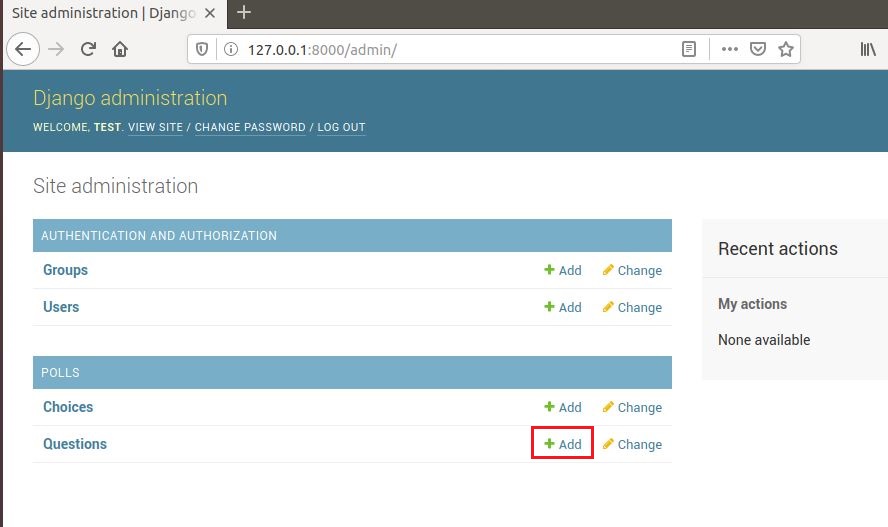
$ python manage.py runserver 0.0.0.0:8000

**6-1. polls 애플리케이션 첫 화면 --> 데이터 입력 전**

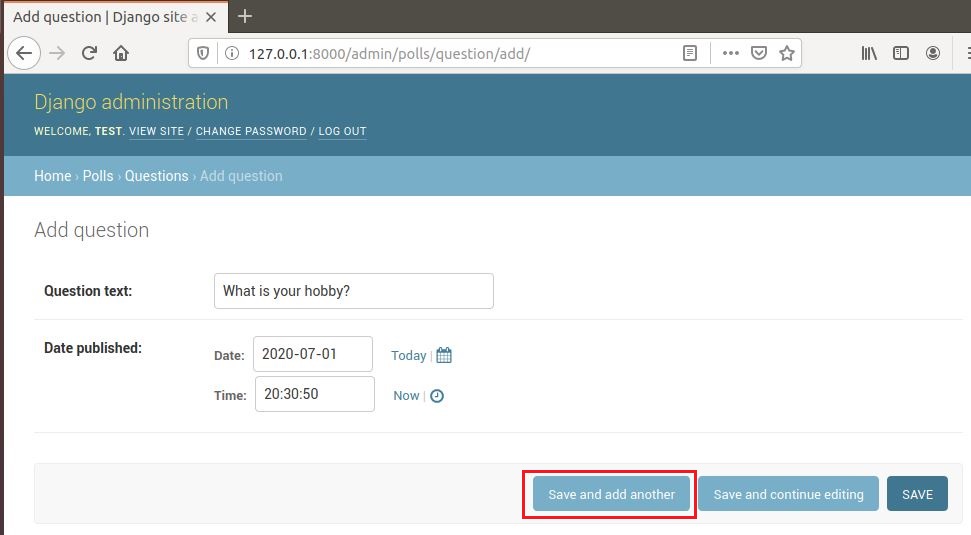


**6-2. Admin 사이트를 이용하여 데이터 입력**

**- Admin 사이트 첫화면. [Questions] 항목의 [Add] 버튼을 클릭하면 질문을 입력하는 화면이 나타난다.**



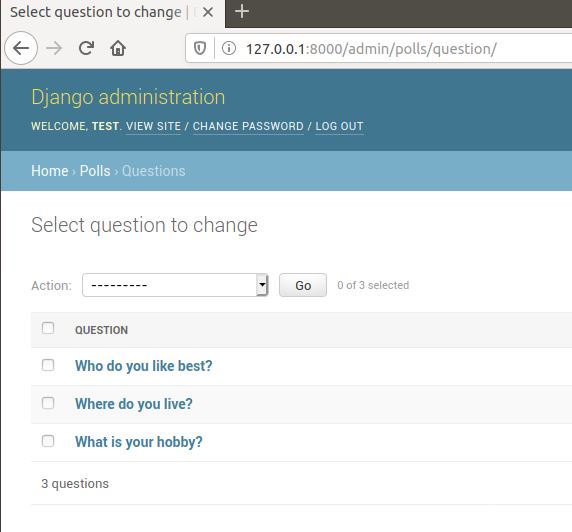
- Quesions 데이터 입력 화면



- 질문으로 입력할 데이터

|  |  |  |
| --- | --- | --- |
| Question text | Date | Time |
| What is your hobby? | 2020-07-01 | 20:30:50 |
| Where do you live? | 2020-06-30 | Noon |
| Who do you like best? | Today | Now |

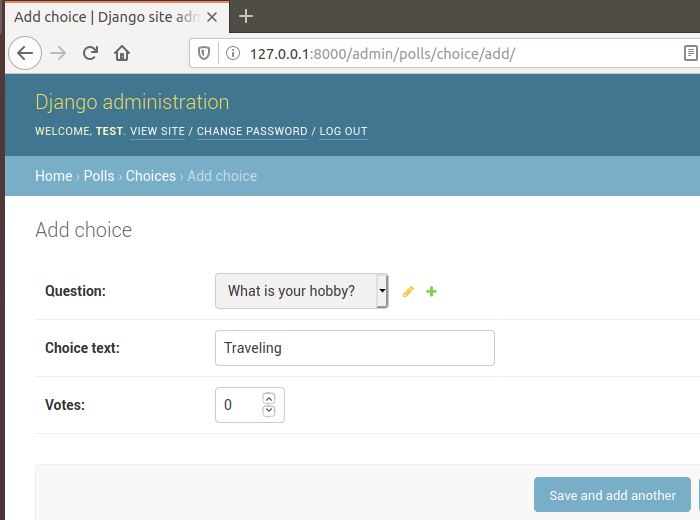
- Questions 데이터 입력 결과



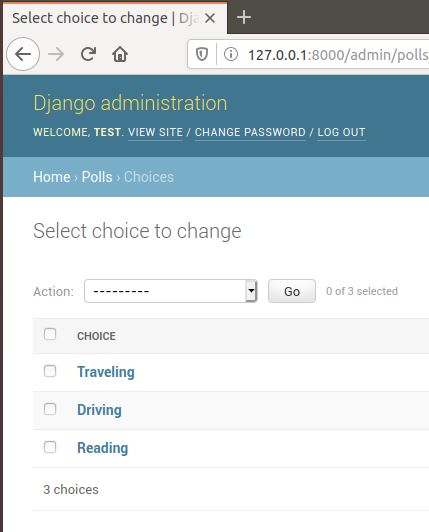
- 질문 답변 항목으로 입력할 데이터

|  |  |  |
| --- | --- | --- |
| Question | Choice text | Votes |
| What is your hobby? | Reading | 0 |
| What is your hobby? | Driving | 0 |
| What is your hobby? | Traveling | 0 |

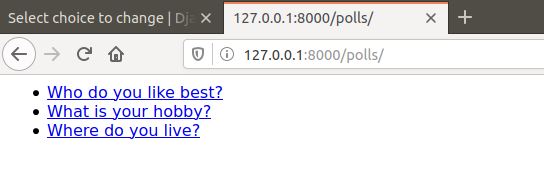
- Choices 데이터 입력 화면



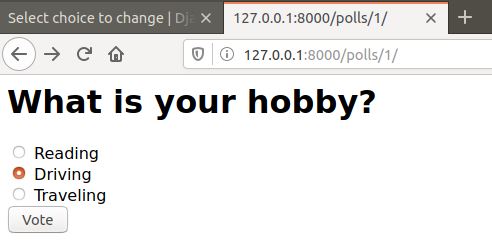
- Choices 데이터 입력 결과



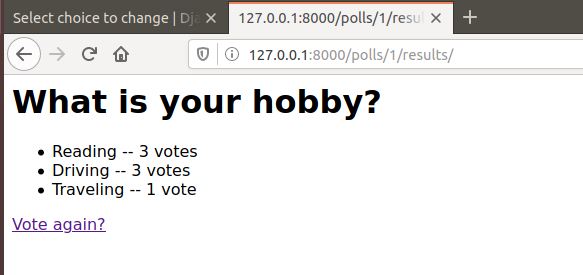
**6-3. polls 애플리케이션 첫 화면 --> 데이터 입력 후**



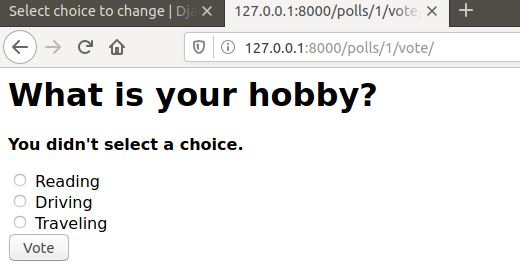
- polls/templates/polls/detail.html



- polls/templates/polls/results.html



- 답변 항목을 선택하지 않은 경우, 에러 메시지 화면



1. 웹 프로그램 개발 시 자주 사용되는 기능들 [↑](#footnote-ref-1)
2. <form>태그 : 라디오 버튼을 보여주고 사용자가 선택할 수 있도록 함. [↑](#footnote-ref-2)