[Tridge] Coding Assignment

Shin Dong Jae

Starting Guide.

Python manage.py runserver

User 계정

<http://127.0.0.1:8000/> - Login(Tridge, dh32616132!) – Testing

Admin 계정

<http://127.0.0.1:8000/admin> - Login(dongjae, tlsehdwo527!) – Testing

Please refer to the below for more information.

**2021/11/24**

* **Project Structure understand, TODO Check , Github (8:00PM ~ 11:00PM)**
* Understand Django App , Model, View , Document
* Set Priority TODO List

“로그인 -> question -> vote -> result” 로 이어지는 프로그램의 흐름과 난이도를 기준으로 우선순위 선정하여 개발

1. Implement Login System – **User**, Security 관점에서 개발
2. Raise 404 if no matching question – **Database – Server** 관점 개발
3. Enable to comment on question , comment, suggest new choice – **User 관점** 개발
4. system question creator approve – Admin 관점
5. Limit the number of choices that can be suggested on one question – Admin 관점 개발
6. Show only questions that are published and not yet closed - **User – Admin 관점** 개발
7. Handle Race condition on handling vote action – Database – Server 관점 개발
8. Implement global search for question and choice – User **관점**
9. In /polls/ fetch only 5 questions through REST API – **Database – Server 관점** 개발

* Create Github repository (<https://github.com/sdj3261/tridge_assignment2>)
* **Login System Implement (11:30AM – 3:00AM)**
* Main Page Redirecting polls index page

def index(request) :  
 return render(request, "polls/index.html")

* If not Login, don’t use poll system

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

<iframe> 태그를 이용한 CSRF(Cross Site Request Forgery) 공격 방지

강제로 Create , Update , Delete 되는 작업을 막기 위해 csrf\_token 사용

<form action = ""method = "post">  
 {% csrf\_token %}  
 <table>  
 {{ form }}  
 </table>  
 <input type = "submit" value = "Login" />  
</form>

**텍스트이(가) 표시된 사진

자동 생성된 설명**

* **Raise 404 if no matching Question Implement (3:00AM~4:00AM)**

(어떤 클래스에서 객체를 가져올지, 클래스의 몇 번째 객체를 가져올 것인지)

Question 객체의 특정 id에서 매칭이 된다면 Redirect

매칭이 되지 않는다면 Raise 404 Error

텍스트이(가) 표시된 사진

자동 생성된 설명

question = get\_object\_or\_404(Question, pk=question\_id)

* **Handle Race condition on handling vote action (4:30AM~5:30AM)**
* Vote와 같이 여러 사용자가 같은 시간에 동시에 버튼을 눌러 프로세스나 쓰레드가 공통 자원을 동시에 read/write 한다면 DB 업데이트 중 race condition이 발생해 한 사용자의 입력이 무시될 수 있다.
* Solve 방법? **Mutex\_lock()** => **Django F()**
* #Use mutex\_lock, Solve Race Condition  
  selected\_choice.votes = F('votes') + 1  
  selected\_choice.save()
* 동시간대에 데이터베이스 수정이 일어나는 부분에는 F를 사용하면 레이스 컨디션 문제를 해결하고, 성능면에서도 이득이 된다.

(일부 작업에 대한 필요 Query 수 감소, 파이썬을 통하지 않고DB 엑세스 가능)

주의 할 점 : 다른 객체를 수정하고 save() 해도 F() 객체가 업데이트 된다.

**11/24 Development Diary**

텍스트이(가) 표시된 사진

자동 생성된 설명

기존의 Flask의 기본 동작 원리만 알고 있어 Django의 대해 알아가는 계기가 되었고 컴퓨터 공학 이론을 적용해보면서 개발을 고민해보는 좋은 시간이었습니다. Handle race condition은 후순위로 정했지만 이론으로만 배웠던 용어가 흥미로워 보여 미리 개발을 시도 했습니다.

Raise 404 ☑, Handle Race condition ☑, Implement Login ☑

**2021/11/25**

* **Enable to comment on question (3:00 PM ~ 6:00 PM)**

1 : N Structure (Question : comment)

on\_delete=cascade를 통해 질문 삭제하는 경우 관련 모든 댓글 모두 삭제

* class Comment(models.Model):  
   # 1:N 구조 (Question 하나에 여러개의 댓글 매칭)  
   question = models.ForeignKey(Question, on\_delete=models.CASCADE, related\_name = "commented\_post")  
   content = models.TextField(max\_length=200)  
   create\_date = models.DateTimeField(default=timezone.now)
* Polls model migration
* def comment\_create(request, question\_id):  
   question = get\_object\_or\_404(Question, pk=question\_id)  
   question.commented\_post.create(content=request.POST.get('content'))  
   return HttpResponseRedirect(reverse("polls:detail", args=(question.id,)))
* HTTPResponseRedirect URL이 변경되어도, Reverse로 변경된URL을추적한다. (누락위험감소 및 유지보수용이)

텍스트이(가) 표시된 사진

자동 생성된 설명

* 댓글에 최근 시간을 표시하는 과정에서 시간 오류 발생 USE\_TZ = Fasle로 바꾸어서 해결

USE\_TZ = False , Asia/Seoul

**+** 최근에 쓴 comment가 위로 오게 개발한다면 사용자 입장에서 더 편리할 것 같다.

* **Enable to comment on comment (PM 8:00 ~ 2:00)**

Comment - Comment => Recursive 방법으로 해결 생각

parent = models.ForeignKey('self', null= True, blank = True, on\_delete=models.CASCADE)

**Question(1) : Comment(N) - Comment(1) : Reply(N)**

def reply\_create(request, comment\_pk):  
 comment = get\_object\_or\_404(Comment, pk=comment\_pk)  
 question = comment.question  
 form = CommentForm(request.POST or None)  
 if request.method == 'POST':  
 reply = form.save(commit=False)  
 reply.parent = comment  
 reply.question = question  
 reply.save()  
 return redirect('polls:detail', pk=question.id)

모델과 뷰는 만들어봤지만 테스트를 하기 위해 HTML에서 어떻게 정보를 가공해서 표시 해야 할지 확인을 할 수가 없었다. Form을 이용한 방법도 있어 다른 방법을 찾아 시도해봤지만 잘 되지 않았다. 세부적인 프레임워크 이해와 재귀에 대한 공부가 더 필요하다고 생각이 들었다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

* **Implement global search for question and choice (AM 3:00 ~ 4:30)**
* 구조만 미리 설계를 시도 했다.
* (global search가 user입장에서 필요한 기능인가? admin / user 입장 모두 생각해서 만들어야 하나?
* 결론적으로는 admin은 각 모델에 대해 search\_filter를 이용하고 유저 입장에서는 global search를 통해 어떤 질문에 해당하는 선택지가 어떤 것들이 있는지 검색하는 것이 옳다고 생각했습니다.

def globalsearch(request):  
 qs = Question.objects.all()  
 qs2 = Choice.objects.all()  
 result = qs.union(qs2)  
 q = request.GET.get('q','')  
 if q:  
 qs = qs.filter(message\_\_icontains=q)  
 return render(request,'polls/search/globalsearch.html', {  
 'globalsearch': qs,  
 })

* Question와 Choice의 테이블을 조인하여 관련 내용 검색 시 Question에 해당하는 - Choice들이 나오게 제작하기 Choice = models.Choices.objects.get(question=question\_id) 활용
* 아직 조금 더 수정이 필요하다고 생각하여 방안만 생각해보고 보류

**11/25 Development Diary**

텍스트이(가) 표시된 사진

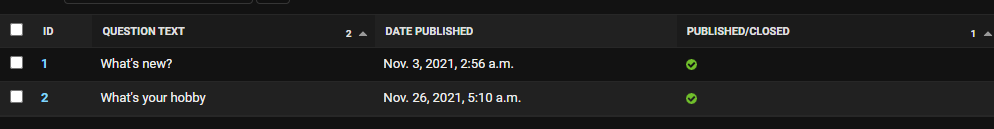
자동 생성된 설명

어렵다고 생각했던 작업들의 기본 틀을 다질 수 있었고 다른 항목들도 프레임워크의 어떤 기능을 이용하여 만들어야 편리할지 찾아보고 생각해보는 시간을 많이 가졌습니다.

Enable to comment = ☑ Comment on comment = △, Implement Global Search = △

**2021/11/26**

* **Show only questions that are published and not yet closed (PM 1:00~ 2:30 PM)**
* Question Model에 is\_public Boolean\_Field 추가
* 관리를 용이하게 하기 위하여 다음과 같이 Admin 페이지 수정
* @admin.register(Question) #Wrapping  
  class QuestionAdmin(admin.ModelAdmin) :  
   list\_display = ['id','question\_text','pub\_date','is\_public']  
   search\_fields = ['question\_text']

****

class DetailView(generic.DetailView):  
 model = Question  
 queryset = Question.objects.filter(is\_public=True)  
 template\_name = "polls/detail.html"

* queryset 파라미터를 이용해서 is\_public값이 true인 게시글만 보이도록 필터링
* **Enable to suggest new choice for question (PM 3:30~ 6:30)**
* Model.objects.create 함수를 통해 suggestion 하여 추가하는 기능 삽입
* /polls/suggestion/create/<int:question\_id> 와 연결

def suggestion(request, question\_id):  
 question = get\_object\_or\_404(Question, pk=question\_id)  
 if request.method == 'POST':  
 Choice.objects.create(  
 question=question,  
 choice\_text = request.POST['text'],  
 )  
 return redirect('polls:detail', pk=question.id)

return render(request, 'polls/detail.html')

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**+ 추후에 연결 할 작업 문제들도 어떻게 처리 할지 생각하면서 제작**

1.Implement system that a question creator can approve suggested choices

- (모델에 boolean is\_approved 항목 추가)

2.Limit the number of Choices

- (한 Question당 suggest 개수 제한)

User Suggestion -> validation\_check -> DB save (commit=false) -> question creator Approve(commit=True)

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

* **Limit the number of Choices (11:30PM ~ 1:00AM)**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

* Set Constant CHICE\_LIMIT

CHOICE\_LIMIT = 5

* Object.filters.count() < CHICE\_LIMIT 으로 비교

def suggestion(request, question\_id):  
 question = get\_object\_or\_404(Question, pk=question\_id)  
 choice\_count = Choice.objects.filter(question=question).count()  
 if request.method == 'POST' and choice\_count < CHOICE\_LIMIT:  
 Choice.objects.create(  
 question=question,  
 choice\_text=request.POST['text'],  
 )  
 return redirect('polls:detail', pk=question.id)

* Extends Question.closed\_at by one day when new choice is suggested **(1:00AM ~ 3:30AM)**

위와 같은 문제를 해결하려면 Django Signal/Receiver System을 이해해야 했다.

시그널 기능은 sender가 어떤 특정한 작업을 수행할 때마다 자동으로 서버에게 알려주고 그 때, 지정한 Receiver가 동작을 수행 할 수 있게 신호를 발생 시키는 기능이다.

동작을 위해 polls/apps.py와 \_\_init\_\_py를 수정한다

class PollsConfig(AppConfig):  
 default\_auto\_field = 'django.db.models.BigAutoField'  
 name = 'polls'  
 def ready(self):  
 import polls.signals

default\_app\_config = 'polls.apps.PollsConfig'

Signals.py 를 다음과 같이 작성한다

@receiver(post\_save, sender=Choice)  
@receiver(post\_save, sender=Choice)  
def question\_post\_save(sender, \*\*kwargs):  
 question = kwargs['instance'].question  
 question.is\_public = False  
 question.pub\_date = timezone.now() + datetime.timedelta(days=1)  
 question.save()  
  
  
post\_save.connect(question\_post\_save, sender=Choice)

Choice는 Sender가 되어 특정 Choice가 저장될 때마다 receiver Function이 작동된다.

Post\_save의 instance 키워드를 통해 연결되어 있는 Question 객체에 접근하여 closed하고 pub\_date를 현재 날짜에 1일 더한다.

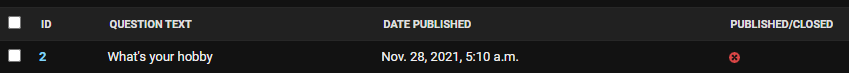
텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

정상적으로 Signal/Receiver System이 돌아가는 것을 확인할 수 있었다.



* In /polls/, fetch only 5 questions through REST API **(4:30AM ~ 6:00AM)**
* Representational State Transfer 의 약자로 자원의 이름으로 구분하여 해당 자원의 상태를 주고 받는 모든 것을 의미한다.
* HTTP URI를 통해 자원을 명시하고 Http Method를 통해 자원에 CRUD 적용
* 자원 : URI(/polls/question\_id) – 행위 : HTTP Method(GET,POST) – 표현(JSON,XML,TEXT)
* 즉, 클라이언트와 서버에서의 개발 내용이 명확해지고 서로간 의존성 감소, 데이터베이스 자료를 직접 전송하는 것이 아니라 HTML, JSON ,TEXT등을 통해 제공
* 장점 : 코드의 재활용성 증가(생산성 향상) , 프론트엔드 개발자와의 협업

Views.py 에 serializer를 통해 정의

* class QuestionViewSet(viewsets.ModelViewSet):  
   queryset = Question.objects.filter(is\_public=True)  
   serializer\_class = QuestionSerializer

urls.py에 라우터 클래스에 등록하여 URL\_Conf를 자동으로 생성

router.register(r"questions", polls.views.QuestionViewSet)

Serializer를 통해 DB 데이터를 JSON으로 변환

class QuestionSerializer(serializers.HyperlinkedModelSerializer):  
 class Meta:  
 model = Question  
 fields = ["id", "question\_text", "is\_public", "pub\_date"]

class IndexView(generic.ListView):  
 template\_name = "polls/index.html"  
 context\_object\_name = "latest\_question\_list"  
  
 def get\_queryset(self):  
 *"""Return the last five published questions."""* response = requests.get(restify("/questions/"))  
 questions = response.json()  
 return questions[:5]

라우터에 등록되어 있던 questions주소 로부터 closed되지 않은 question 리스트를 0번부터 4번까지 가져와서 반환한다.

**11/26 Development Diary**

텍스트이(가) 표시된 사진

자동 생성된 설명

+ 연결된 작업을 진행하다 보니 예상치 못한 오류가 발생하는 경우가 있었습니다. 이러한 점은 내일 수정하고 나머지 기능을 완벽하게 구현하는 것을 목표로 과제를 마무리할 예정입니다.

+ 수정해야 할 내용

* Signal/Receiver구현 중 열려 있다가 닫힌 페이지는 HTML에 그대로 열린 상태로 남아있어 사용자가 여전히 닫힌 페이지를 클릭할 수 있는 오류 발견
* 다음의 작업을 통해 해결 REST API 에서 가져오는 query에 필터링 적용

class QuestionViewSet(viewsets.ModelViewSet):  
 queryset = Question.objects.filter(is\_public=True)  
 serializer\_class = QuestionSerializer

* Suggestion을 Voting한 이후 result.html에서 받는 것이 기능상 자연스럽다고 생각이 들음
* HTML 위치 변경을 통해 해결
* 댓글 시스템은 Create 뿐만 아니라 Update와 Delete기능까지 추가하는 것이 맞다고 생각이 들음

Enable to suggest new choice = ☑, Limit the number of choices that can be on question = ☑

Extends question.closed\_at by one day use signal = ☑, In /polls/ , fetch only 5 question = ☑

Show only questions that are published not closed = ☑

**2021/11/27**

* Implement global search for questions and choices **(3:30AM ~ 6:30AM)**

Q() 에 대한 이해가 필요했다.

def globalsearch(request):  
 choice\_list = Choice.objects.all()  
 search\_key = request.POST.get('search\_key')  
 global\_filter = Q(question\_\_question\_text\_\_icontains=search\_key) | Q(choice\_text\_\_icontains=search\_key) | Q(votes\_\_icontains=search\_key)  
 if search\_key:  
 question\_choice\_list = choice\_list.filter(global\_filter).order\_by('question')  
 return render(request, 'polls/search/globalsearch.html', {  
 'globalsearch': question\_choice\_list,})  
 else :  
 return HttpResponseRedirect(reverse("polls:index"))

* 폼의 input search\_key 를 이용하여 icontains를 통해 question\_text, choice\_text, vote 의 포함되어 있는 문자를 통해 해당 하는 값을 조회 할 수 있게 만들었다.
* 장고에서는 테이블에 대한 복잡한 쿼리를 처리하기 위해 Q 객체를 제공하며 검색 결과가 있을 때만 polls/search를 request 한다.
* Question 모델은 Choice의 Model의 Foreign Key를 통해 Q(question\_\_question\_text) 를 통해 접근할 수 있었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* **Implement a question creator can approve suggested choice** **(8:00PM ~ 5:00AM)**
* **Choice 모델에 Approve 값을 따로 만들어 True 일 때만 추가해주어야 한다.**
* **Approve system 제작 중 기존 설계가 잘못되었다는 것을 깨닫고 버그를 수정하면서 진행했다.**
* **Testing & Bug Fix (12:00 AM ~ 4:30AM)**

1. Render 문제

127.0.0.1:8000 으로 이동했을 때 REST API의 latest\_question\_list를 받아오지 못하는 현상이 있었다. 로그 아웃 / 로그인을 다시해야 정상적으로 값을 받아올 수 있었는데

단순히 index.html을 렌더링하면 값을 가져오지 못해 IndexView를 활용하여 다음과 같이 수정

따라서 hamji 폴더의 urls path 설정

path('', polls.views.IndexView.as\_view())

IndexView를 실행시켜 해결하였다. 아래는 원래 render 했던 방법이다.

# def index(request):  
# return render(request, "polls/index.html")

path('', polls.views.index, name = 'index'),

1. Global Search 확장

유저 입장에서 각 설문조사가 언제 끝나는지 , approved 된 항목과 되지 않은 항목, 정렬을 통해 직관적으로 확인할 수 있게 만들었다.

테이블이(가) 표시된 사진

자동 생성된 설명

1. Suggestion 후 뒤로 가기를 막는 방법?

Sugeestion 버튼을 누르면 Close 되는데 뒤로 가기를 하고 다시 Suggestion이 가능한 오류를 발견했다.

해결하기 위해 검색을 통해 POST 데이터를 처리 후에는 항상 HttpResponseRedirect를 반환해야 해야 이런 오류를 방지할 수 있다는 것을 알았다. Render , HTTPSResponseRedirect , redirect 3가지의 용도를 제대로 이해해야 된다고 생각이 들었다.

1. Signal/Receiver 에서 suggestion이 아니라 voting 할때도 receiver 가 작동되는 현상

@receiver(post\_save, sender=Choice)  
def question\_post\_save(sender, \*\*kwargs):  
 question = kwargs['instance'].question  
 question.is\_public = False  
 question.pub\_date = timezone.now() + datetime.timedelta(days=1)  
 question.save()  
  
post\_save.connect(question\_post\_save, sender=Choice)

기존 코드의 문제점은 다음과 같다.

1. Question의 Closed\_at 항목으로 open/close 했어야 했다.
2. Signal/receiver의 역할은 close를 하는 것이 아니라 close time을 하루 연장하는 것이다.
3. Choise 객체가 Create될때가 아닌 Vote 값이 업데이트 될때도 receiver가 작동한다.

@receiver(post\_save, sender=Choice)  
def question\_post\_save(sender, created, \*\*kwargs):  
 if created :  
 question = kwargs['instance'].question  
 question.closed\_at += datetime.timedelta(days=1)  
 question.save()  
  
  
post\_save.connect(question\_post\_save, sender=Choice)

* create될때만 closed 1day extend

class Choice(models.Model):  
 question = models.ForeignKey(Question, on\_delete=models.CASCADE)  
 approved = models.BooleanField(default=True)  
 choice\_text = models.CharField(max\_length=200)  
 votes = models.IntegerField(default=0)  
  
 def \_\_str\_\_(self):  
 return self.choice\_text

choice모델의 approved 항목 추가

class Question(models.Model):  
 question\_text = models.CharField(max\_length=200)  
 is\_public = models.BooleanField(default=True, verbose\_name='Published')  
 pub\_date = models.DateTimeField("date published")  
 closed\_at = models.DateTimeField(default=timezone.now() + datetime.timedelta(days=3))

question 모델에 closed\_at 추가

class QuestionViewSet(viewsets.ModelViewSet):  
 queryset = Question.objects.filter(closed\_at\_\_gte=datetime.now())  
 serializer\_class = QuestionSerializer

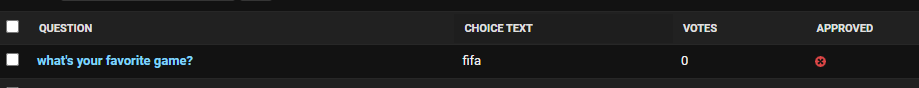
* **날짜 즉 datetime.now()와 question.closed\_at을 비교 하기위해서는 gte , lt와 같은 기능을 이용해 필터링 해야한다.**
* **이를 이용해 closed 예정 시각을 지나지 않은 question만 가져온다.**

queryset = Question.objects.filter(closed\_at\_\_gte=datetime.now())

* **Implement a question creator can approve suggested choice** **(8:00PM ~ 5:00AM)**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

****

**텍스트, 스크린샷, 모니터이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명.**

HTML에는 화면에 approved 된 choice만 뿌려주었다.

{% if choice.approved is True %}

버그들을 고치면서 Suggestion 과 approve를 시스템이 내가 생각했던 대로 정상적으로 작동됨을 확인할 수 있었다.

**11/27 Development Diary**

텍스트이(가) 표시된 사진

자동 생성된 설명

오류를 수정하는데 많은 시간이 걸렸다. 그래도 전체적으로 설문조사 시스템의 의미를 생각하면서 내가 원했던 위치와 방법 대로 기능들을 구현하고 테스트할 수 있었다.

제출 후에는 일주일 이내로 Comment on Comment 기능에 다시 도전 해야겠다. Django의 폼 기능과 Class View Function View에 대해 공부하고 Template 문법만 익히면 충분히 구현이 가능할 것 같다.

Django의 프레임워크에 아직 미숙했던 부분이 있었고 더 효율적으로 코드를 짤 수 있지 않았을까 아쉬운 부분도 있었다. 하지만 과제를 하면서 그동안 이론으로만 배웠던 컴퓨터 공학 지식들을 적용해볼 수 있었고 추후 연결할 기능을 생각하면서 오류가 생기지 않게 시스템을 만들기 위해 고민을 하면서 재밌게 만들 수 있었다.