

HICC 개발 입문 세미나

6주차. REST API(1)

Django의 MTV 구조

Model(모델)

- 데이터베이스에 저장되는 데이터와 View 사이의 연결고리 역할을 합니다. 모델은 클래스로 정의되며 **하나의 클래스가 하나의 DB Table**입니다.
원래 DB를 조작하기 위해선 SQL이라는 DB 전용 언어를 다룰 줄 알아야 하지만 **Django는 ORM(Object Relational Mapping)기능을 지원하기 때문에 파이썬 코드로 DB를 조작할 수 있습니다.**

View(뷰)

- 요청에 따라 적절한 로직을 수행하여 결과를 템플릿으로 렌더링하며 응답합니다. (다만 항상 템플릿을 렌더링 하는 것은 아니고 서버에서 데이터만 보내주는 경우가 대부분입니다. 해당 내용은 6~7주차때 깊게 다룹니다.)

Template(템플릿)

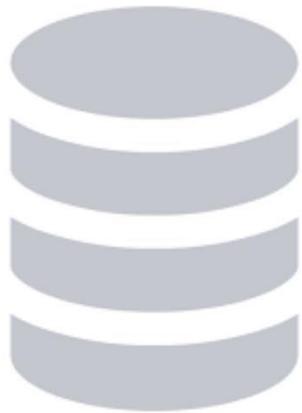
- 유저에게 보여지는 화면을 의미합니다. 장고는 뷰에서 로직을 처리한 후 html 파일을 context와 함께 렌더링하는데 이 때의 html 파일을 템플릿이라 칭합니다.

MTV 구조

• Model Template View 구조

데이터베이스와 연결

Model 영역



Models.py
DB 데이터

3. DB 정보 제공

2. DB에 요청

Controller 영역



Views.py
가공

4. HTML에
정보 삽입

5. 렌더링

1. Http 요청

6. Http 응답
Html 데이터



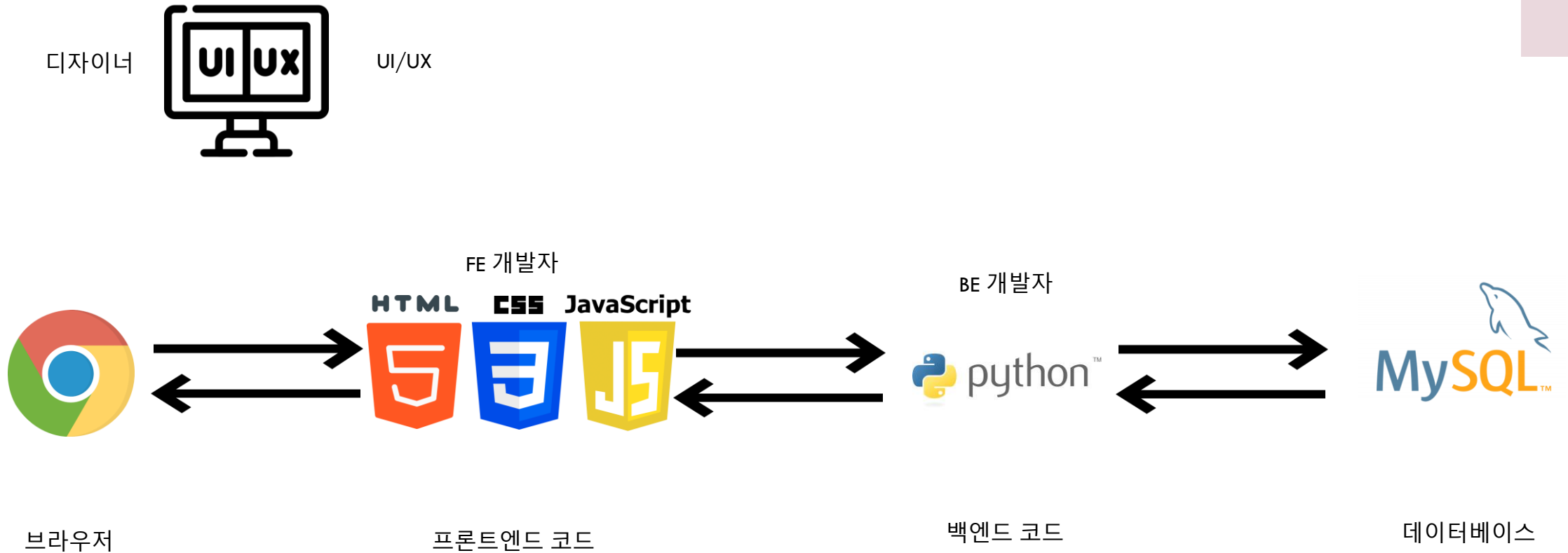
프론트엔드와 연결

View 영역



Template
보여지는 화면

일반적 프로그램 개발 프로세스



MVC 패턴

Django의 MTV 구조처럼, 주요 웹 프레임워크는 MVC(Model – View – Controller) 패턴을 지닙니다.

Model(Django의 Model) 영역에서 데이터베이스와 백엔드 코드를 연결하고,

View(Django의 Template) 영역에서 백엔드 코드와 프론트엔드 코드를 연결하고,

Controller(Django의 View) 영역에서 Model, View, 클라이언트(웹 브라우저) 사이를 연결합니다.

관계형 데이터베이스

관계형 데이터베이스의 기본 용어

- 테이블 – 하나의 개체에 관한 데이터를 2차원 구조로 정의한 것
- 튜플 – 테이블의 행, 쉽게 말해 실제 데이터
- 속성 – 테이블의 열, 쉽게 말해 실제 데이터들의 카테고리
- 도메인 – 실제 데이터 값의 종류, 쉽게 말해 **자료형**

열(속성, 애트리뷰트)

고객아이디	고객이름	나이	등급	직업	적립금
CHAR(20)	CHAR(20)	INT	CHAR(10)	CHAR(10)	INT
apple	김현준	20	gold	학생	1000
banana	정소화	25	vip	간호사	2500
carrot	원유선	28	gold	교사	4500
orange	정지영	22	silver	학생	0

행(튜플)

도메인

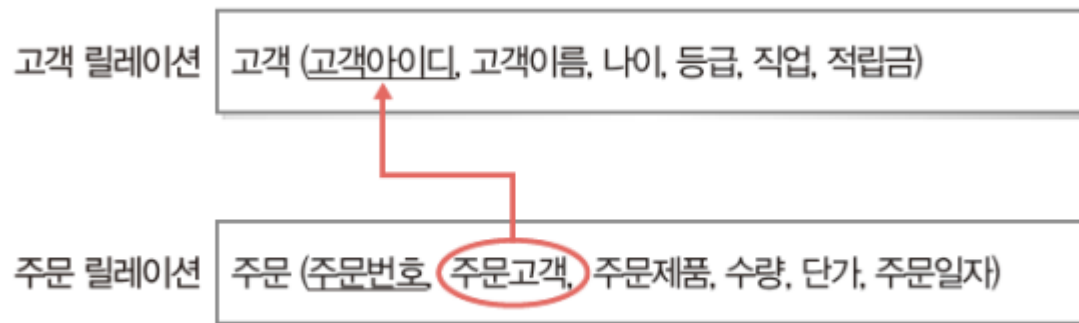
관계형 데이터베이스

기본키(primary key)

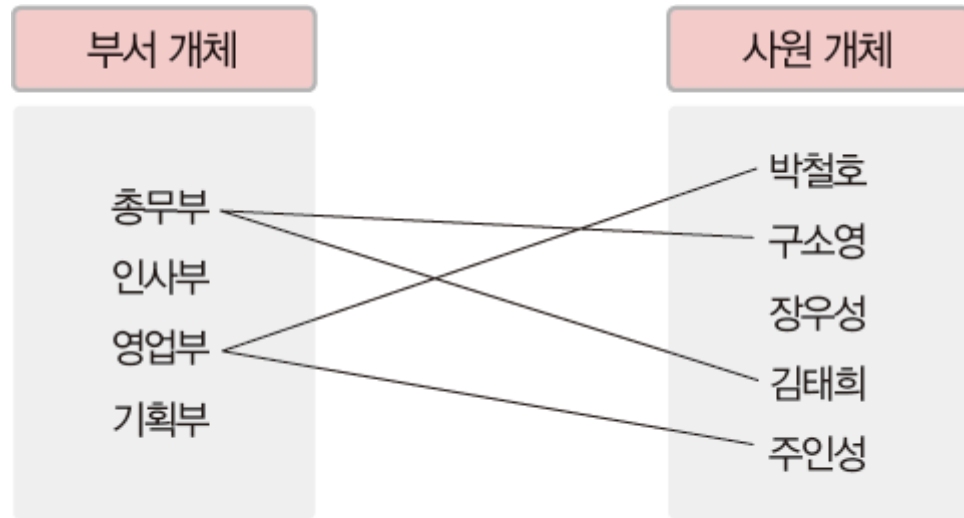
- 테이블에서 튜플 하나가 기본적으로 사용하는 고유의 아이디
- Ex) 고객 테이블의 기본키 : 고객아이디

외래키(foreign key)

- 다른 테이블의 기본키를 참조하는 속성
 - 테이블들 간의 관계를 표현
- ex) 주문 테이블의 외래키 : 주문고객 (고객아이디를 참조하기 때문에 고객 테이블을 참조함)



관계형 데이터베이스



이와같은 1 : N 관계에서 N쪽이 1쪽을 참조함
(즉 N 테이블이 1 테이블의 기본키를 외래키로 지님)

Model 코드 다시보기

```
from django.db import models
```

```
class Question(models.Model): # Question은 테이블 subject, content, create_date는 속성
    subject = models.CharField(max_length=200) # CharField는 데이터 타입, 글자 타입을 의미하고 최대 길이는 200
    content = models.TextField() # TextField는 데이터 타입, 글자 타입이지만 최대 길이가 정해져 있지 않으면 TextField
    create_date = models.DateTimeField() # DateTimeField는 데이터 타입, 날짜 데이터를 저장하는 속성
```

```
class Answer(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    #ForeignKey는 외래키를 지정하는 함수, Question 클래스를 참조함, on_delete=models.CASCADE는 Question 클래스의 데이터가
    삭제됐을 때, Answer 데이터도 같이 삭제됨을 의미하는 옵션
    content = models.TextField()
    create_date = models.DateTimeField()
```

해당 코드와 같이 클래스에서 primary key를 따로 지정하지 않으면, **id**라는 **primary key** 속성이 자동으로 생성됨.

더 많은 필드와 옵션은

<https://github.com/dkyou7/TIL/blob/master/%ED%8C%8C%EC%9D%B4%EC%8D%AC/Django/5.%20%5BDjango%5D%20Model%20%E D%95%84%EB%93%9C%ED%83%80%EC%9E%85%20%EC%A0%95%EB%A6%AC.md> 참고

urls.py 코드 수정

우리가 구현하려는 건, 특정 질문에 대한 디테일한 것들을 확인할 수 있는 페이지입니다.

url 경로를 /question/(question의 ID)

이런식으로 설정하면 **특정 질문에 대한 페이지**라는 것을 충분히 나타낼 수 있습니다.

hicc_app의 urls.py 코드를 아래와 같이 수정해줍시다.

```
from django.urls import path
```

```
from . import views
```

```
urlpatterns = [
```

```
    path('question/<int:question_id>/', views.question_detail)
```

```
]
```

여기서 <int:question_id>는 해당 위치에 들어오는 값을 int 자료형(정수)의 question_id 라는 변수로 처리하겠다는 의미입니다.

templates 폴더에 해당 코드 추가

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>질문 게시글</title>
</head>
<body>
  <h1 id="subject">{{ question.subject }}</h1>
  <div id="content">{{ question.content }}</div>
  <p></p>
  <h5>{{ answers|length }}개의 답변이 있습니다.</h5>
  <div>
    <ul>
      {% for answer in answers %}
      <li class="answer">{{ answer.content }}</li>
      {% endfor %}
    </ul>
  </div>
</body>
</html>
```

해당 코드를 'question_detail_ssr' 이라는 이름으로 templates 폴더에 html 파일로 저장

views.py 코드 수정

앞에서 언급했듯이, `views.py(Controller)`는 해당 프로그램에서 모델, 뷰, 클라이언트 사이에 일어나는 모든 로직을 컨트롤합니다.

현재 `views.py` 코드에서 하는 일은

1. Model 영역에 DB의 데이터를 요청하고
2. 해당 데이터를 받은 후
3. View 영역으로부터 HTML 코드를 받아서
4. HTML 코드에 아까 받은 DB의 데이터들을 추가해주고
5. 클라이언트의 웹 브라우저에 렌더링

이 다섯가지 행위를 포함해야 합니다.

views.py 코드 수정

```
from django.shortcuts import render
from .models import Question, Answer # Model 영역에서 Question과 Answer 클래스를 가져옴 (테이블로서 역할)

def question_detail(request, question_id):
    question = Question.objects.get(id=question_id)
    # Question 클래스에서 question_id에 저장된 값과 일치하는 id의 Question 데이터를 가져와서 question이라는 변수에 저장

    answers = Answer.objects.filter(question=question)
    # Answer 클래스에서 아까 가져온 question 변수에 저장된 질문과 일치하는 질문들을 외래키 속성으로 지니는 (즉 해당 질문에 대한 답변들만 걸러서) Answer 데이터들을 가져와서 answers라는 변수에 저장

    return render(request, 'question_detail_ssr.html', {
        'question': question,
        'answers': answers,
    })
    # 아까 저장한 question_detail_ssr이라는 html 파일을 templates 폴더로부터 가져오는데, 현재 question 변수에 저장된 값, answers 변수에 저장된 데이터들을 'question', 'answers' 라는 이름으로 해당 html에 추가해서 렌더링해줌
```

views.py 코드 수정

1. Model 영역에 DB의 데이터를 요청하고
2. 해당 데이터를 받은 후
3. View 영역으로부터 HTML 코드를 받아서
4. HTML 코드에 아까 받은 DB의 데이터들을 추가해주고
5. 클라이언트의 웹 브라우저에 렌더링

```
from django.shortcuts import render, get_object_or_404
from .models import Question, Answer # Model 영역에서 Question과 Answer 클래스를 가져옴 (테이블로서 역할)
```

```
def question_detail(request, question_id):
    question = Question.objects.get(id=question_id)
    answers = Answer.objects.filter(question=question)
    # 1번 Model 영역에 DB의 데이터를 요청, 2번 해당 데이터를 받음
```

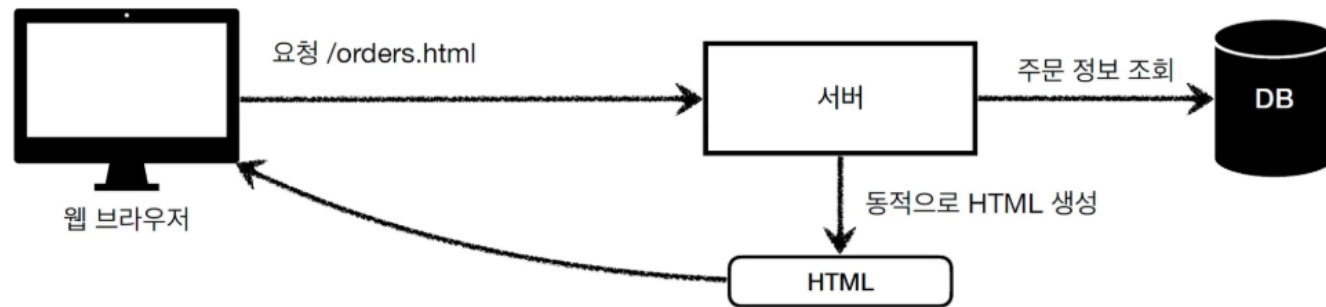
```
    return render(request, 'question_detail_ssr.html', {
        'question': question,
        'answers': answers,
    })
```

```
    # 3번 View 영역으로부터 HTML 코드를 받음, 4번 HTML 코드에 아까 받은 데이터들을 추가해줌, 5번 클라이언트의 웹 브라우저에 렌더링
```

Server Side Rendering

SSR - 서버 사이드 렌더링

서버에서 최종 HTML을 생성해서 클라이언트에 전달



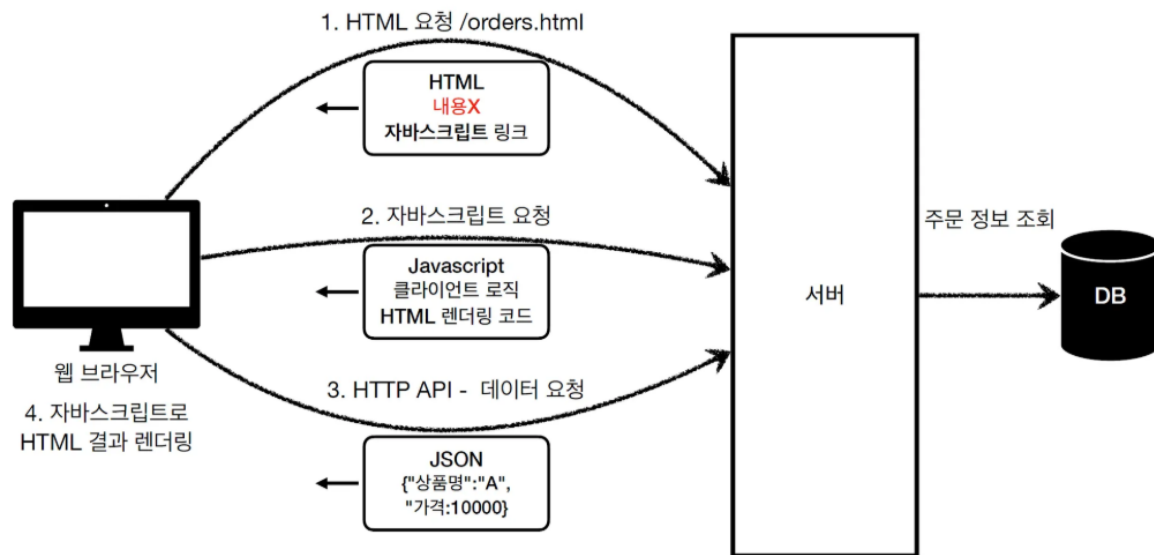
자료 출처 - <https://www.infllearn.com/courses/lecture?courseId=326674&type=LECTURE&unitId=71804&tab=curriculum>

지금까지 설명한 방식을 서버쪽에서 데이터가 추가된 HTML 파일을 전부 완성 시킨 후 그대로 브라우저로 넘겨준다고 해서 Server Side Rendering(SSR)이라고 부릅니다.

프론트엔드와 백엔드의 역할 분리가 명확하지 않았던 과거에 주로 사용된 개발 방식으로, 최근에는 개발 방식의 주류에서 약간 떨어진 방식입니다.

Client Side Rendering

CSR - 클라이언트 사이드 렌더링

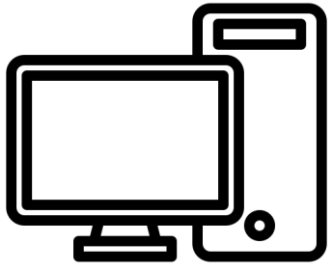


자료 출처 - <https://www.infllearn.com/courses/lecture?courseId=326674&type=LECTURE&unitId=71804&tab=curriculum>

이와 같이 서버쪽에서 HTML을 직접 완성시키는 것이 아닌, 서버는 데이터만 보내주고 클라이언트가 해당 데이터를 받아서 직접 HTML을 완성시킨 후 화면에 띄워주는 방식을 Client Side Rendering(CSR)이라고 부릅니다.

프론트엔드와 백엔드의 역할 분리가 명확해진 최근 주류로 사용되는 개발 방식으로, 프론트엔드는 화면을 꾸미는 일에 집중하고 백엔드는 데이터를 처리하는데 집중합니다.

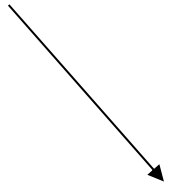
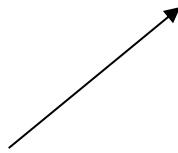
Client Side Rendering



웹 브라우저

3. 브라우저에서 코드 실행

7. 실행된 결과 띄우기



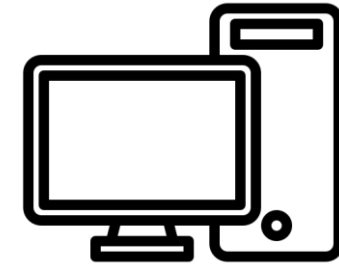
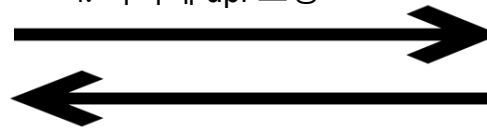
질문 게시판		
번호	제목	작성일시

질문 등록하기

4. 서버에 api 요청

5. 서버에서 코드 실행

6. api 응답 데이터 전송



서버

질문 게시판		
번호	제목	작성일시
3	HICC는 무엇의 약자인가요?	2025. 3. 11. 오전 3:53:20
4	HICC 동방은 몇호인가요?	2025. 3. 11. 오전 4:25:32

질문 등록하기

API

이와 같은 개발 과정에서 클라이언트와 서버는 "API"라는 개념을 통해 서로 데이터를 주고 받으며 통신합니다.

API란?

API (Application Programming Interface)

→ 응용 프로그램이 서로 데이터나 기능을 주고받기 위한 약속된 인터페이스

레스토랑의 메뉴판처럼 세트 A를 시키면 스테이크와 파스타가 오고, 이러한 것들이 정의된 목록이라고 보시면 됩니다.

그리고 우리는 여기서 세트 A와 같은 것을 "엔드포인트"라고 칭합니다.

API



```
{
  "questions": [
    {
      "id": 1,
      "subject": "hicc 세미나는 언제인가요?",
      "content": "모르겠네요.",
      "create_date": "2025-05-07T00:00:00Z"
    },
    {
      "id": 4,
      "subject": "동아리 방은 어디에 있나요?",
      "content": "자세히 알려주세요.",
      "create_date": "2025-05-12T00:00:00Z"
    }
  ]
}
```

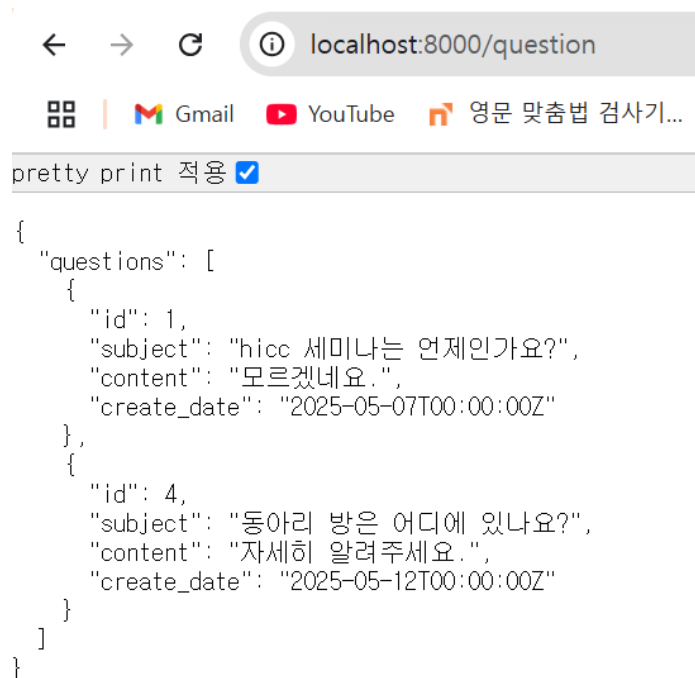
현재 클라이언트와 서버는

"클라이언트가 `/question` 해당 url로 요청하면 서버는 DB에 저장된 질문들의 리스트를 `questions`라는 이름으로 보내줄게" 라는 약속을 합니다.

해당 예시에서 `/question` 이 세트 A 같은 존재,
반환되는 많은 데이터들이 스테이크 + 파스타 같은 존재라고 볼 수 있습니다.

여기서 `"/question"`을 엔드포인트, `"/question`을 입력하면 이러한 데이터를 보내준다는 약속"을 API라고 보시면 됩니다.

JSON



```
{
  "questions": [
    {
      "id": 1,
      "subject": "hicc 세미나는 언제인가요?",
      "content": "모르겠네요.",
      "create_date": "2025-05-07T00:00:00Z"
    },
    {
      "id": 4,
      "subject": "동아리 방은 어디에 있나요?",
      "content": "자세히 알려주세요.",
      "create_date": "2025-05-12T00:00:00Z"
    }
  ]
}
```

그리고 이처럼 클라이언트와 서버가 데이터를 주고받을 때, 위와 같은 획일화된 형식으로 데이터를 교환합니다.
([] 으로 이루어진 부분은 리스트 영역을 의미하는 것이고 여러 개의 JSON 값을 담을 수 있습니다.)

{ "데이터 이름": "실제 데이터 값" } (문자의 경우 값은 " " 사이에 넣고 숫자의 경우 그냥 넣음)

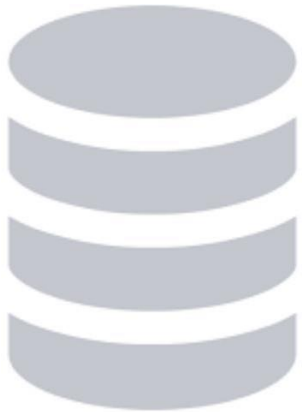
파이썬의 딕셔너리 자료형 { 'A' : 123 } 과 유사한 형식이라는 점에 주목합니다.

MTV 구조 (CSR)

• Model Template View 구조

데이터베이스와 연결

Model 영역



Models.py
DB 데이터

3. DB 정보 제공

2. DB에 요청

Controller 영역



Views.py

가공

1. Http 요청

4. Http 응답
Json 데이터



urls.py 코드 수정

우리가 구현하려는 것은

"클라이언트가 /question 해당 url로 요청하면 서버는 DB에 저장된 질문들의 리스트를 questions라는 이름으로 보내줄게" 이 부분입니다.

먼저 /question 해당 url을 접속했을 때 무언가 실행되도록 urls.py 코드를 수정해줍시다.

```
from django.urls import path
from . import views
urlpatterns = [
    path('question/<int:question_id>/', views.question_detail),
    path('question', views.questions)
]
```

views.py 코드 수정

views.py에서는 "DB에 저장된 질문들의 리스트를 questions라는 이름으로 보내주는 것"만 구현하면 됩니다.
views.py 코드에 해당 함수를 추가해줍니다.

```
from django.http import JsonResponse
```

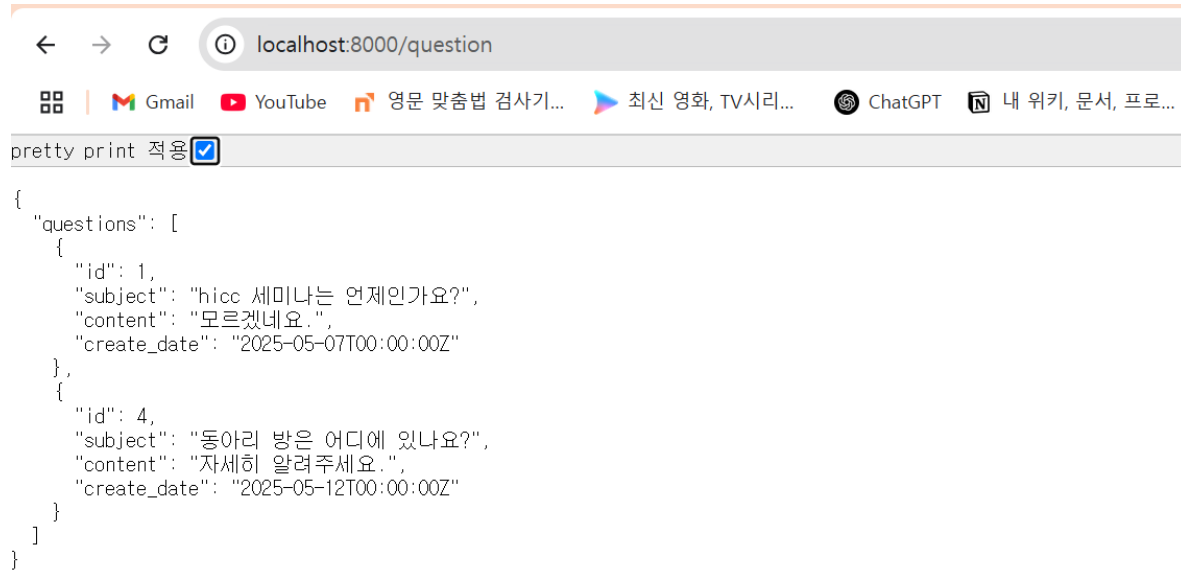
```
def questions(request):
```

```
    if request.method == 'GET':
```

```
        questions = Question.objects.all().values('id', 'subject', 'content', 'create_date')
```

```
        return JsonResponse({'questions': list(questions)})
```

views.py 코드 수정



```
{
  "questions": [
    {
      "id": 1,
      "subject": "hicc 세미나는 언제인가요?",
      "content": "모르겠네요.",
      "create_date": "2025-05-07T00:00:00Z"
    },
    {
      "id": 4,
      "subject": "동아리 방은 어디에 있나요?",
      "content": "자세히 알려주세요.",
      "create_date": "2025-05-12T00:00:00Z"
    }
  ]
}
```

터미널에서 python manage.py runserver 실행 후 잘 나오는지 확인

views.py 코드 자세히 보기

```
def questions(request):
```

```
    if request.method == 'GET':
```

```
        # 모든 http 요청에는 메서드 라는 개념이 존재하는데, url을 직접 입력해서 들어가는건 GET 요청, 자세한건 다음주에
```

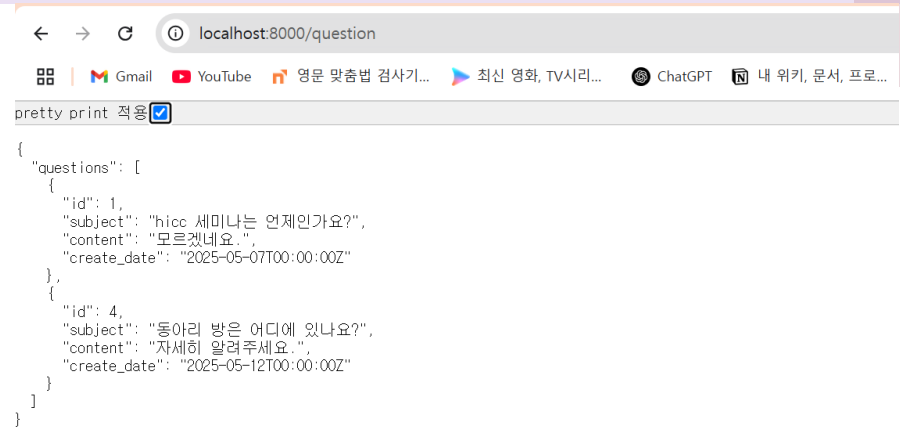
```
        questions = list(Question.objects.all().values('id', 'subject', 'content', 'create_date'))
```

```
        # Question 클래스에서 모든 데이터를 가져올건데, 각각 'id', 'subject', 'content', 'create_date' 라는 이름으로  
        가져옴, 데이터들이 여러 개이므로 list 형식으로 변환해서 저장
```

```
    return JsonResponse({'questions': questions})
```

```
    # questions라는 데이터들을 JSON 형식에 맞게 반환할 수 있도록 딕셔너리 형식으로 바꿔줍니다. 이러한 과정을  
    직렬화(Serialization)이라고 합니다.
```

```
    # 딕셔너리 형식으로 변환된 데이터를 JsonResponse라는 메서드를 통해 JSON 형식으로 그대로 반환해줍니다.
```



직렬화(Serialization)

```
localhost:8000/question

pretty print 적용 ☒

{
  "questions": [
    {
      "id": 1,
      "subject": "hicc 세미나는 언제인가요?",
      "content": "모르겠네요.",
      "create_date": "2025-05-07T00:00:00Z"
    },
    {
      "id": 4,
      "subject": "동아리 방은 어디에 있나요?",
      "content": "자세히 알려주세요.",
      "create_date": "2025-05-12T00:00:00Z"
    }
  ]
}
```

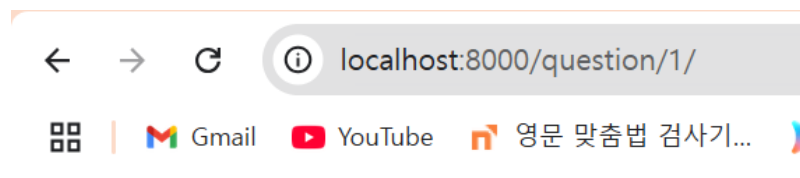
앞서 언급했듯이, 클라이언트와 서버는 JSON이라는 형식으로 데이터들을 주고받습니다.

서버가 클라이언트에게 데이터를 보내주기 위해서는 자신들이 다루는 데이터를 JSON 형태로 변환하는 과정이 필요합니다.

이와 같이 특정 데이터를 JSON으로 변환하는 것을 직렬화(Serialization)이라고 합니다.

Python의 경우에는 딕셔너리 자료형이 JSON과 같은 형식을 띄고 있기 때문에 직렬화를 위해서 데이터를 먼저 딕셔너리 자료형으로 변환해줍니다.

기존 코드 변환

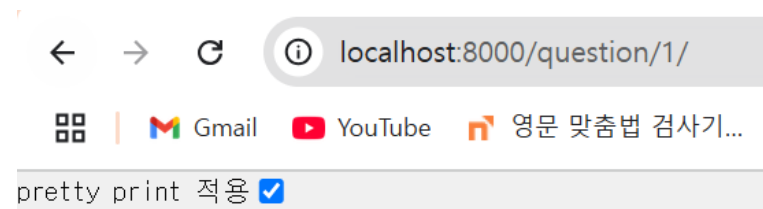


hicc 세미나는 언제인가요?

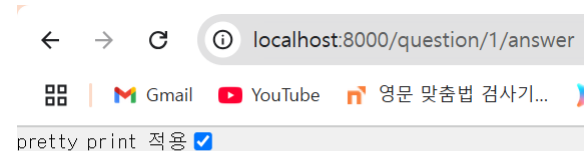
모르겠네요.

1개의 답변이 있습니다.

- 매주 수요일 6시입니다.



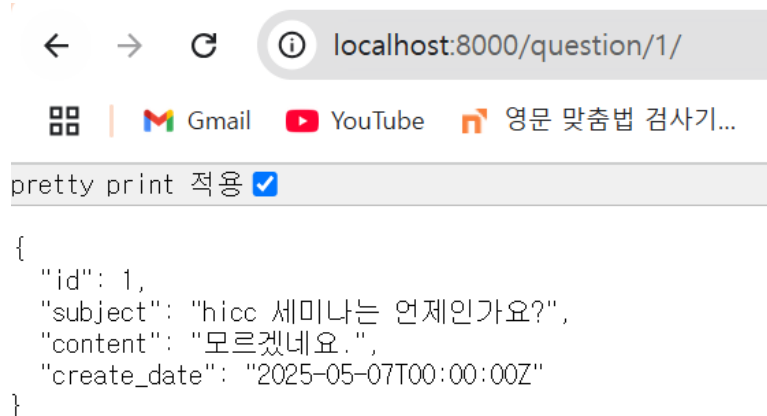
```
{
  "id": 1,
  "subject": "hicc 세미나는 언제인가요?",
  "content": "모르겠네요.",
  "create_date": "2025-05-07T00:00:00Z"
}
```



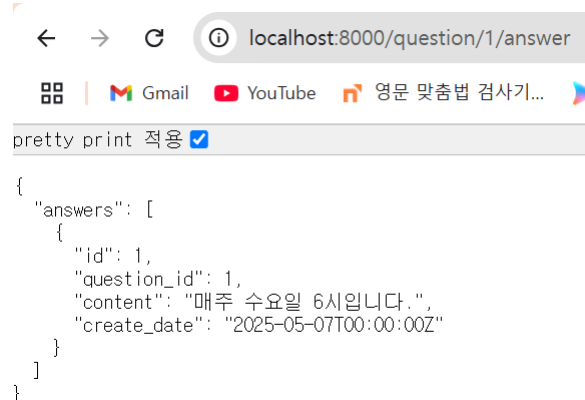
```
{
  "answers": [
    {
      "id": 1,
      "question_id": 1,
      "content": "매주 수요일 6시입니다.",
      "create_date": "2025-05-07T00:00:00Z"
    }
  ]
}
```

SSR 방식으로 개발된 좌측 페이지를 이제 우측의 CSR 방식으로 변환해봅시다.

기존 코드 변환



```
{
  "id": 1,
  "subject": "hicc 세미나는 언제인가요?",
  "content": "모르겠네요.",
  "create_date": "2025-05-07T00:00:00Z"
}
```



```
{
  "answers": [
    {
      "id": 1,
      "question_id": 1,
      "content": "매주 수요일 6시입니다.",
      "create_date": "2025-05-07T00:00:00Z"
    }
  ]
}
```

기존 방식처럼 views.py의 메서드를 한 페이지 단위로 두지않고,

질문에 대한 데이터 반환, 해당 질문에 대한 답변에 대한 데이터 반환
이런 식으로 역할에 따라 분리해서 두는게 바람직합니다.

urls.py 수정

```
from django.urls import path
from . import views

urlpatterns = [
    path('question/<int:question_id>/', views.question_detail), #특정 질문의 내용
    path('question', views.questions),
    path('question/<int:question_id>/answer', views.answers) #특정 질문에 대한 답변
]
```

urls.py에서 answer 데이터를 얻을 수 있는 엔드포인트만 추가해줍니다.

views.py 수정(1)

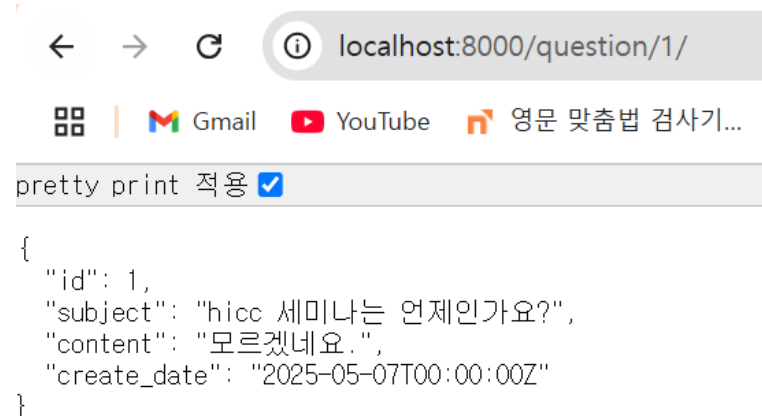
```
def question_detail(request, question_id):  
    if request.method == 'GET':  
        question = Question.objects.get(id=question_id)
```

```
        question_data = {  
            'id': question.id,  
            'subject': question.subject,  
            'content': question.content,  
            'create_date': question.create_date  
        }
```

question이라는 객체의 변수들을 딕셔너리 형식으로 변환하는 직렬화 과정

```
        return JsonResponse(question_data) # 이미 위에서 직렬화가 끝난게 question_data이므로 그대로 반환
```

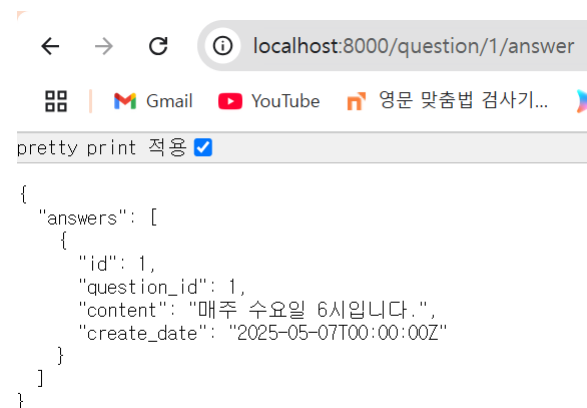
특정 질문에 대한 자세한 데이터를 조회할 수 있는 기능을 구현하기 위해서 해당 메서드를 추가해줍니다.



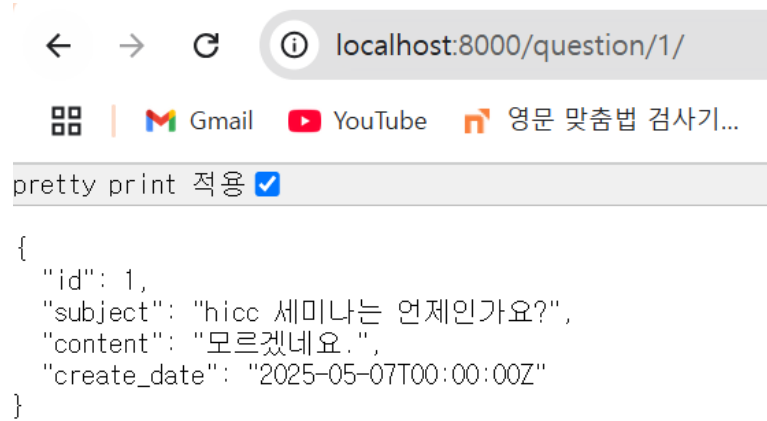
views.py 수정(2)

```
def answers(request, question_id) :  
    if request.method == 'GET':  
        answers = list(Answer.objects.filter(question=question_id).values('id', 'question_id', 'content', 'create_date'))  
        #Answer 클래스에서 url에 있는 question_id 값과 일치하는 질문들만 전부 가져오는데, 각각 'id', 'question_id', ... 등의  
        이름으로 가져옴, 그 후 list 형식으로 변환  
        return JsonResponse({'answers': answers}) # answers는 list 값이기 때문에 직렬화 필요. 'answers' : 로 직렬화 해줌
```

특정 질문에 대한 답변을 조회할 수 있는 기능을 구현하기 위해서
해당 메서드를 추가해줍니다.



views.py 수정(2)

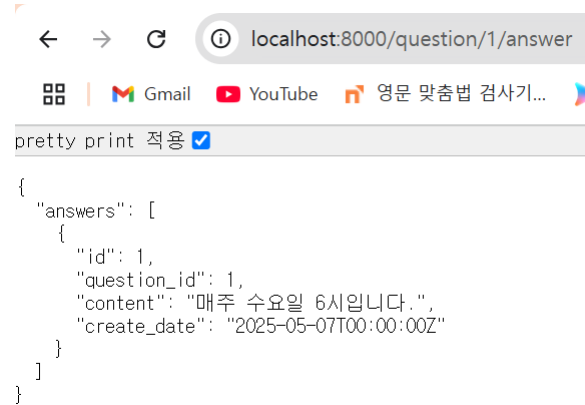


← → ↻ ⓘ localhost:8000/question/1/

☰ | Gmail YouTube 영문 맞춤법 검사기...

pretty print 적용 ☒

```
{
  "id": 1,
  "subject": "hicc 세미나는 언제인가요?",
  "content": "모르겠네요.",
  "create_date": "2025-05-07T00:00:00Z"
}
```



← → ↻ ⓘ localhost:8000/question/1/answer

☰ | Gmail YouTube 영문 맞춤법 검사기...

pretty print 적용 ☒

```
{
  "answers": [
    {
      "id": 1,
      "question_id": 1,
      "content": "매주 수요일 6시입니다.",
      "create_date": "2025-05-07T00:00:00Z"
    }
  ]
}
```

서버를 키고 해당 url에 접속하면 잘 되는걸 확인 가능

다음주 예고

```
from django.http import JsonResponse
from django.shortcuts import render, get_object_or_404
from .models import Question, Answer

def question_detail(request, question_id): 1개의 사용 위치 장현학Wgusgk *
    if request.method == 'GET':
        question = Question.objects.get(id=question_id)

        question_data = {
            'id': question.id,
            'subject': question.subject,
            'content': question.content,
            'create_date': question.create_date
        }

        # question이라는 객체의 변수들을 딕셔너리 형식으로 변환하는 과정

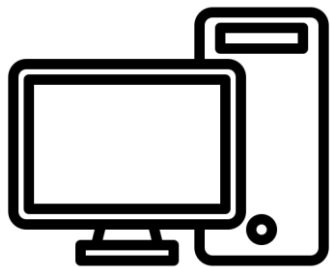
        return JsonResponse(question_data)

def answers(request, question_id) : 1개의 사용 위치 신규 *
    if request.method == 'GET':
        answers = list(Answer.objects.filter(question=question_id).values('id', 'question_id', 'content', 'create_date'))
        # 모든 답변 데이터를 가져올거데, objects.all()이 아닌 objects.filter(question = question_id)는 인자로 받는 question_id와

        return JsonResponse({'answers': answers})

def questions(request): 1개의 사용 위치 신규 *
    if request.method == 'GET':
        questions = list(Question.objects.all().values('id', 'subject', 'content', 'create_date'))
        return JsonResponse({'questions': questions}) # json 형식으로 설정 후 response
```

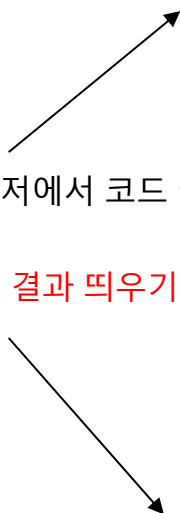
다음주 예고



웹 브라우저

3. 브라우저에서 코드 실행

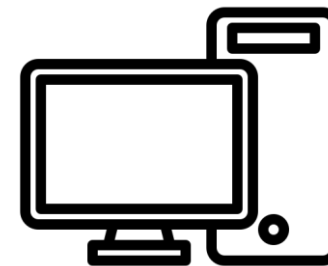
7. 실행된 결과 띄우기



4. 서버에 api 요청

5. 서버에서 코드 실행

6. api 응답 데이터 전송



서버

질문 게시판

번호	제목	작성일시
----	----	------

질문 등록하기

질문 게시판

번호	제목	작성일시
3	HICC는 무엇의 약자인가요?	2025. 3. 11. 오전 3:53:20
4	HICC 동방은 몇호인가요?	2025. 3. 11. 오전 4:25:32

질문 등록하기

다음주 예고

1주차. 웹 통신과 개발의 이해 (OT)

2주차. HTML, CSS, JS 개념 기초

3주차. 웹 프레임워크 이해 (Django를 활용한 페이지 랜더링) -- 정적 웹 서비스 구현

4주차. GIT 사용 실습 -- 필수 개발자 도구 익히기

5주차. MVC 패턴 (MTV) + 데이터베이스 기초

6주차. REST API (1) (Django ORM 하드코딩) -- 동적 웹 서비스 구현

7주차. REST API (2) (템플릿 프론트엔드 연결)