

캡스톤디자인 최종 결과보고서

신입생을 위한 길라잡이 프로그램



담당 교수	심 종 익 교수님	
과제명	캡스톤디자인 최종 결과보고서	
학과	항공정비전공	항공신소재전공
학번	201700796	201700404
이름	김훈섭	김경준
제출 일자	2022.12.01	
수강 요일	데이터베이스 수요일 오후반	
팀명	한서베이스	

목차

1. 프로젝트 주제	1
2. 프로젝트 목적 및 목표	1
2.1 프로젝트 목적	
2.2 프로젝트 목표	
3. 프로젝트 개요	1
3.1 프로젝트 배경	
3.2 프로젝트 구성도	
3.3 프로젝트 메인 페이지 구성도	
3.4 기능 설명	
4. 팀 구성 및 역할	7
5. 개발 도구	7
6. 개발 일정	8
7. 프로젝트 기대효과	8
8. 프로젝트 개발 과정	8
8.1 프로젝트 UI 개발	
8.2 장고와 데이터베이스(MySQL) 연동하기	
8.3 테이블 설계	
8.4 회원 기능 구현 (accounts)	
8.5 게시글 구현 및 정보(장학금, 다전공) 구현 (db_app)	
8.6 게시글 및 정보 검색 기능 구현 (search)	

1. 프로젝트 주제

대학을 입학한 신입생을 위한 길라잡이 프로젝트

2. 프로젝트 목적 및 목표

2.1 프로젝트 목적

대학교에 관련된 정보가 부족한 신입생에게 앞으로의 대학교 생활에 필요한 정보 제공을 목적으로 합니다.

2.2 프로젝트 목표

신입생이 찾고자 하는 정보를 쉽게 찾고, 이용할 수 있도록 하는 것이 이번 프로젝트의 목표입니다.

3. 프로젝트 개요

3.1 프로젝트 배경

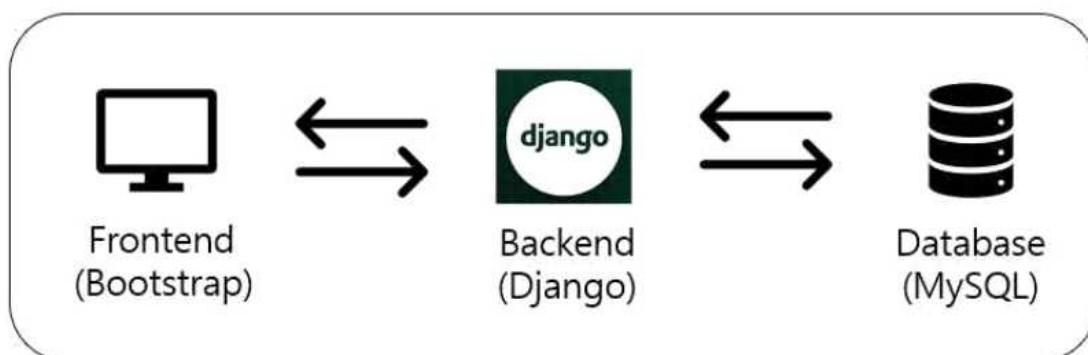
[첫 번째 배경]

대학생 사이에서 정보를 존재하는 대학 익명 커뮤니티가 이미 존재합니다. 하지만 이곳에서 공유되는 정보 중에서 올바르지 않은 정보들이 난무합니다. 그 이유를 생각해봤을 때 누구나 글을 쉽게 등록할 수 있기 때문이라 생각했습니다. 그래서 객관적인 정보와 사용자의 의견이 들어간 주관적인 정보를 구분하기 위해서 프로젝트를 계획했습니다.

[두 번째 배경]

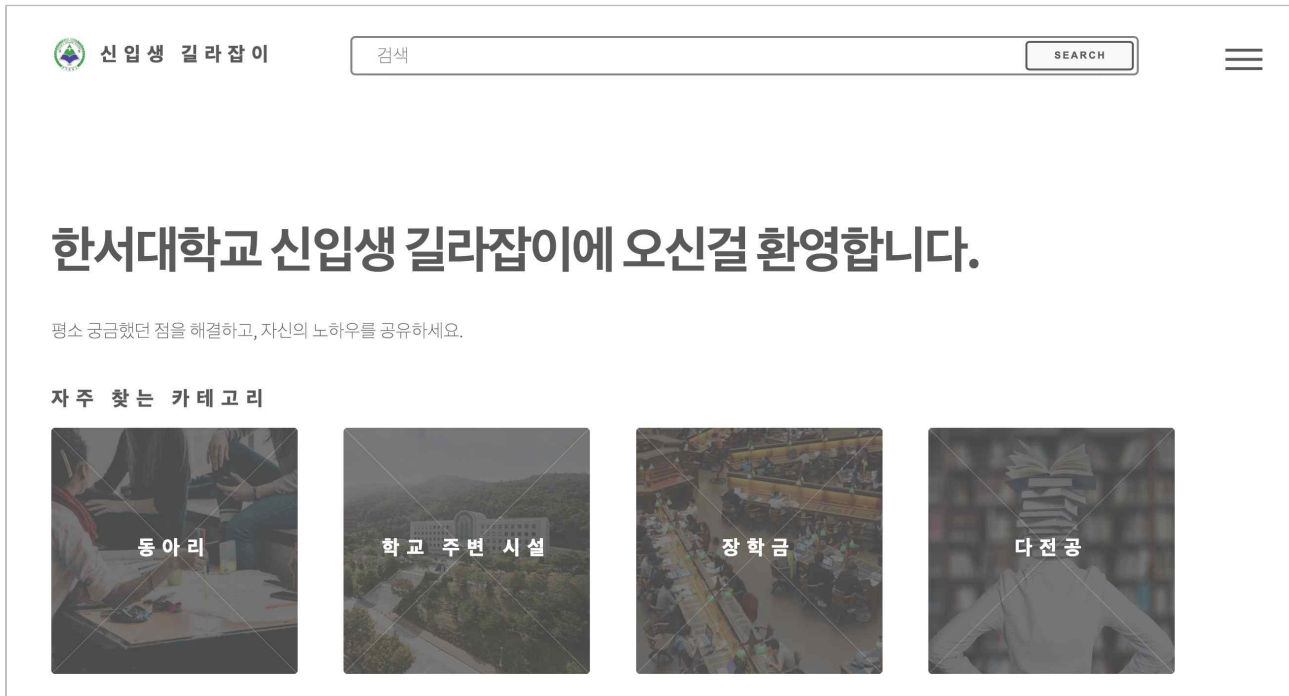
저희는 모두 비전공자 4학년입니다. 소프트웨어에 관련된 정보를 미리 얻고 흥미를 일찍 가졌다면, 부전공이나 복수전공도 도전해볼 수 있었다고 생각합니다. 하지만 관련된 정보를 3학년 2학기가 되고 나서 알게 되었고, 시간이 부족하여 도전할 수 없었습니다. 만약 저학년 때 여러 학과에 대한 정보를 얻을 수 있었다면 늦지 않게 준비할 수 있었을 거 같습니다. 그래서 저희 팀은 신입생들을 위한 프로젝트를 진행하기로 생각했습니다.

3.2 프로젝트 구성도



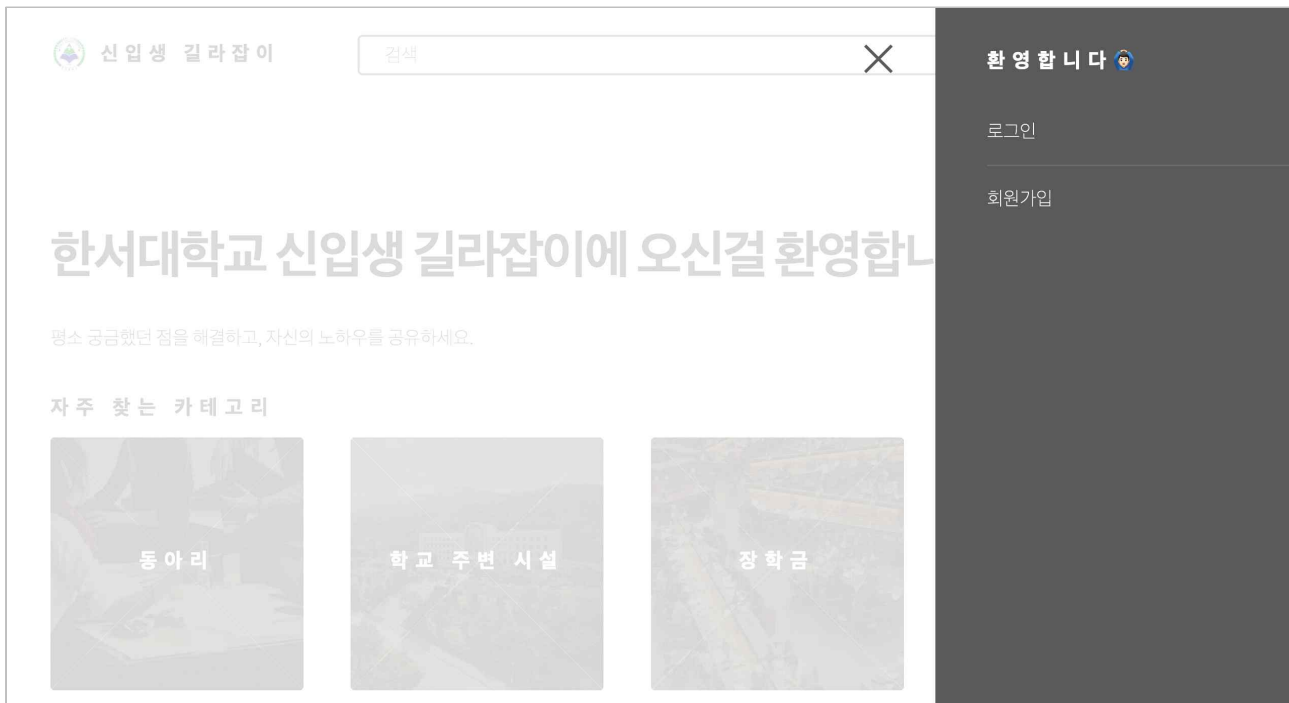
3.3 프로젝트 메인 페이지 구성도

① 메인 페이지



- 주요 카테고리 4개가 존재합니다.
- 제일 상단에 검색창이 있어서 게시글 또는 정보를 검색할 수 있습니다.

② 메뉴 클릭 -> 로그인 / 회원가입 가능



3.4 기능 설명


① 회원과 비회원 기능

구분	회원	비회원
정보	<ul style="list-style-type: none"> ▶ 원하는 정보를 검색하고 읽기 가능 ▶ 장학금, 다전공에 대한 정보 등록 불가능 	
회원 기능	로그인, 로그아웃 기능 사용 가능	회원 가입 기능 사용 가능
게시글	게시글 등록, 수정, 삭제가 가능	게시글 조회만 가능

② 관리자 기능

관리자 기능 (Admin)	<ul style="list-style-type: none"> ▶ 장고에서 제공하는 Admin 페이지에 접근이 가능 -> 모든 데이터에 대한 접근이 가능 ▶ 정보(장학금, 다전공)를 등록할 수 있는 권한을 부여 -> 올바르게 않은 정보를 관리하기 위한 목적
-------------------	---

③ 회원가입 기능


신입생 길라잡이

아래 양식을 확인해서 모두 입력해주세요!

Student ID

Name

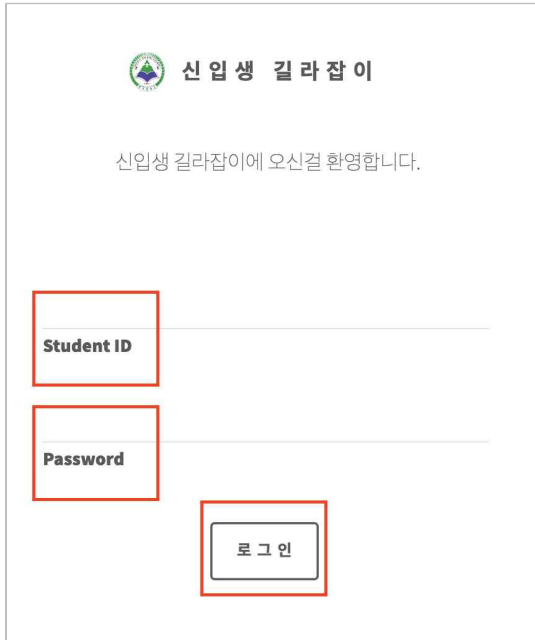
Password

Password Check

회원 가입

- 학번, 이름, 비밀번호, 비밀번호 재입력 과정을 거친 후 회원가입 버튼을 눌러줍니다.

④ 로그인 / 로그아웃 기능



The image shows a web page titled "신입생 길라잡이" (New Student Guide). Below the title is a message: "신입생 길라잡이에 오신걸 환영합니다." (Welcome to the New Student Guide). There are two input fields: "Student ID" and "Password". Below these fields is a button labeled "로그인" (Login).

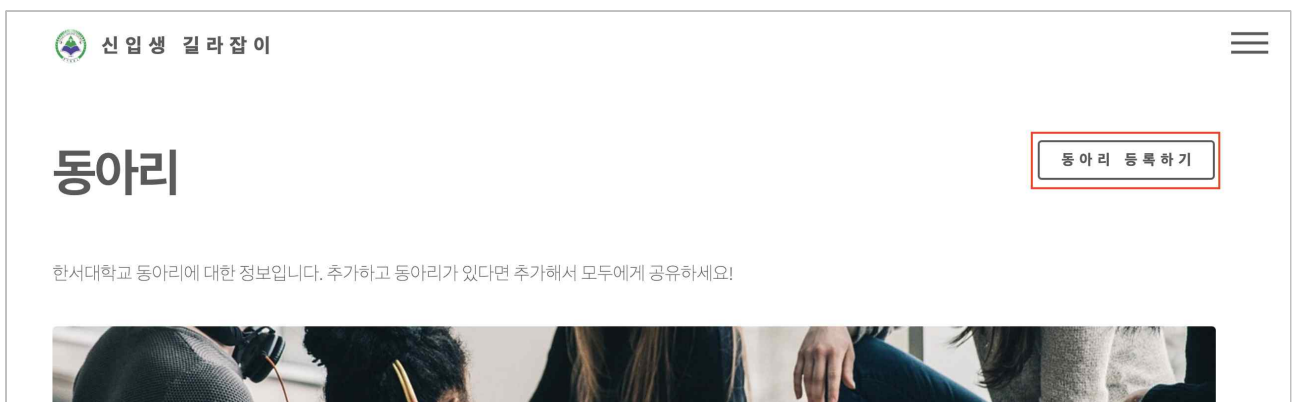
- 위에서 만든 회원 정보 중에서 학번과 비밀번호를 입력한 후 로그인 버튼을 누르면 로그인이 됩니다.



The image shows a user dashboard with a dark background. At the top, it says "안녕하세요. 김경준님" (Hello, Mr. Kim Kyung-jun). Below this is a button labeled "김경준 로그아웃" (Kim Kyung-jun Logout).

- 로그인을 하게되면 “안녕하세요. {본인이름}님”이라는 문구를 확인할 수 있고, 그 아래 ”{본인이름} 로그아웃“이라는 버튼이 존재합니다. 해당 버튼을 누르면 로그아웃 됩니다.

⑤ 게시글 등록하기



The image shows a web page titled "신입생 길라잡이" (New Student Guide) with a hamburger menu icon in the top right corner. The main heading is "동아리" (Club). Below the heading is a button labeled "동아리 등록하기" (Register Club). Underneath the button is a message: "한서대학교 동아리에 대한 정보입니다. 추가하고 동아리가 있다면 추가해서 모두에게 공유하세요!" (This is information about Hanyang University clubs. Add it and if you have a club, add it to share with everyone!). At the bottom is a banner image showing a group of people.

- 동아리 조회 페이지로 이동합니다.
- 로그인이 되어있는 상태라면 “동아리 등록하기” 버튼이 오른쪽 상단에 나타납니다. 클릭해주면 동아리를 등록할 수 있는 Form이 나타납니다.

동 아 리 등 록 하 기

동아리 이름

한서 등산클럽

카테고리

운동

전화번호

010-0000-0000

내용

저희 동아리에 들어와서 재밌는 등산을 함께 해봐요!

관련 URL

https://www.instagram.com/hoons_b/

이미지

파일 선택 등산

작 성 하 기

- 동아리 이름, 카테고리, 전화번호, 내용, 관련 URL을 입력하고, 등록해줄 이미지를 첨부해줍니다.
- 모두 작성했다면 작성하기 버튼을 눌러줍니다.

한서 등산클럽

카테고리: 운동

연락처 : 010-0000-0000

동아리 URL : 동아리 보러가기 🔍

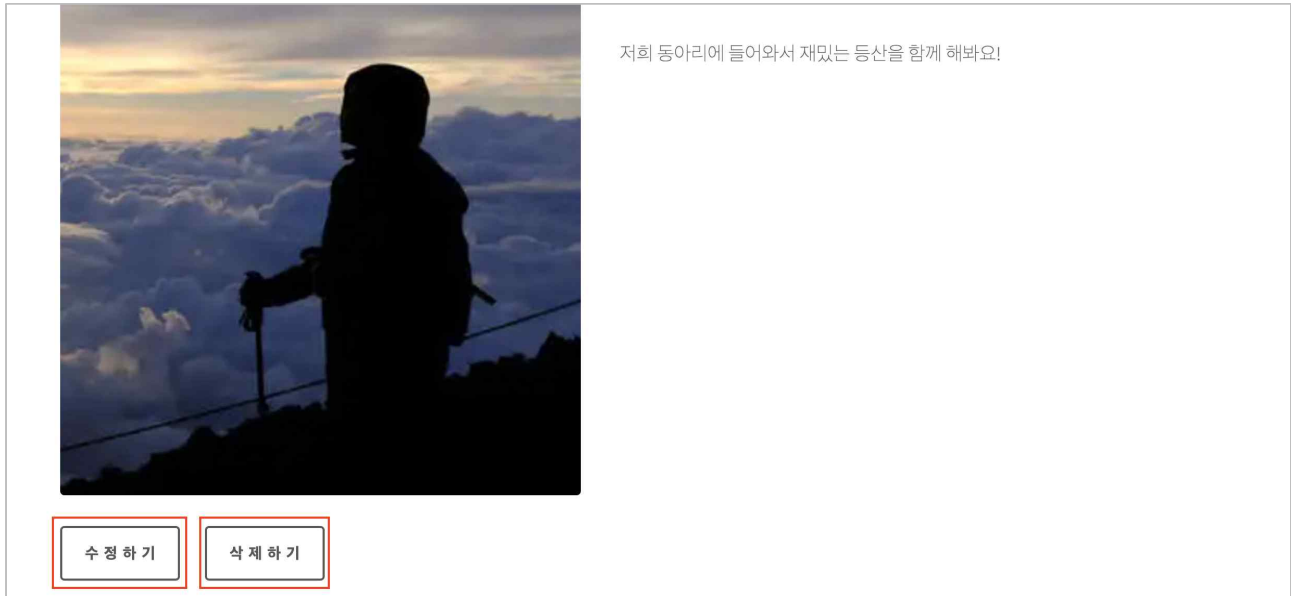


내 용

저희 동아리에 들어와서 재밌는 등산을 함께 해봐요!

- 작성한 동아리가 등록된 것을 확인할 수 있습니다.

⑥ 게시물 수정, 삭제하기

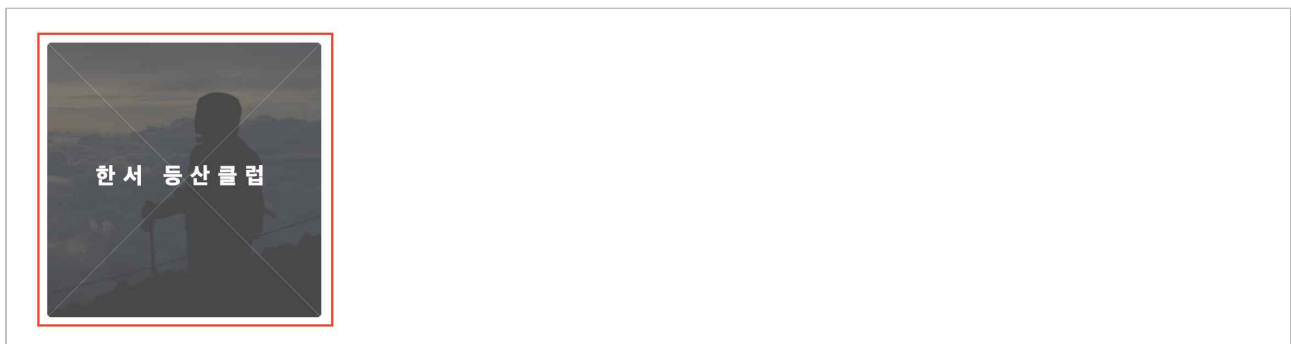


- 게시물 상세 페이지에 들어가보면 “수정하기”, “삭제하기” 버튼을 볼 수 있습니다. 해당 게시글을 작성한 사용자만 해당 버튼을 사용할 수 있습니다.
- “수정하기”를 누르면 앞서 게시물 등록하기에서 봤던 Form이 나오게 되고, 수정하고 싶은 부분만 수정해서 다시 등록하면 됩니다.
- “삭제하기”를 누르면 해당 게시글은 삭제됩니다.

⑦ 게시물 및 정보 검색 기능



- 위에서 등록했던 “한서 등산클럽” 동아리를 검색해봅니다. 검색 키워드로는 제목에 포함되어 있는 “등산”을 입력하고 “SEARCH” 버튼을 눌러줍니다.



- 한서 등산클럽이 검색되어 나오는 것을 확인할 수 있습니다.

4. 팀 구성 및 역할

팀원 이름	역할
김훈섭	<ul style="list-style-type: none"> ▶ 게시글 기능 (등록 / 수정 / 삭제) 구현 ▶ Bootstrap 사용하여 사용자 UI 구성하기 ▶ 데이터 정보 찾고 등록하기 ▶ 데이터 테이블 설계하기
김경준	<ul style="list-style-type: none"> ▶ 회원 기능 (로그인 / 로그아웃 / 회원가입) 구현 ▶ Django와 MySQL 연동하기 ▶ 데이터 정보 찾고 등록하기 ▶ 데이터 테이블 설계하기

5. 개발 도구

개발 도구	설명
Bootstrap (Html, Css, Javascript)	사용자에게 보이는 UI는 Html과 Css를 이용하고, 동적인 요소들은 Javascript를 이용할 계획입니다. 추가적인 요소는 Bootstrap을 이용합니다.
Django Framework	Python 언어를 사용해서 웹 개발을 할 수 있는 Framework인 Django를 이용할 계획입니다.
MySQL	Django와 연동하여 사용할 수 있는 Database인 MySQL을 사용할 계획입니다.
Visual Studio Code	Python을 이용하여 Django 개발에 사용할 수 있는 IDE입니다.
Git / Github	원활한 협업을 위한 개발 도구입니다.

6. 개발 일정

내용	1주차	2주차	3주차	4주차	5주차	6주차	7주차	8주차
프로젝트 계획								
장고 학습 기간								
UI 제작 및 DB 연동								
DB 테이블 설계 및 개발								
데이터 등록 및 테스트								
결과 보고서 작성								

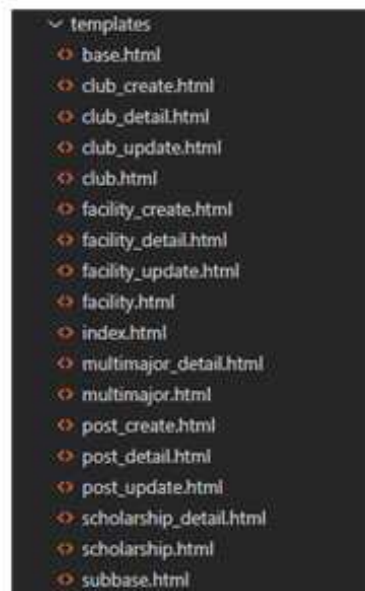
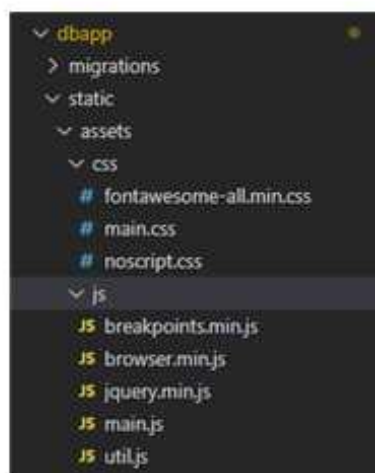
7. 프로젝트 기대 효과

대학교 신입생들이 다양한 정보를 쉽게 찾아서 이용할 수 있고, 대학교 생활의 빠른 적응과 앞으로의 계획을 미리 준비할 수 있을 것입니다. 그리고 신입생을 제외한 기존 대학생도 정확한 정보를 얻을 수 있을 것입니다.

8. 프로젝트 개발 과정

8.1 사용자 UI 개발

Bootstrap에 있는 css와 javascript를 이용하여 html를 제작했습니다.

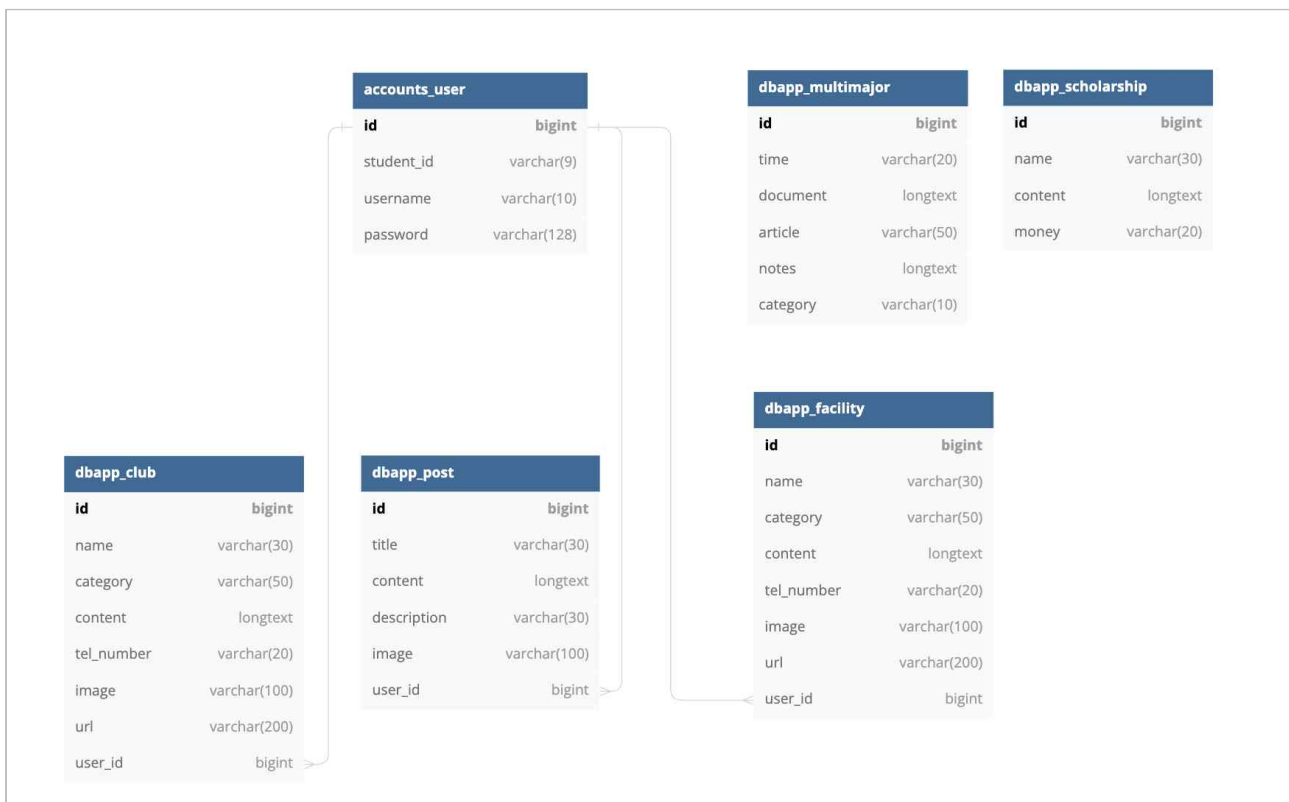


8.2 장고와 데이터베이스(MySQL) 연동하기

MySQL과 Django를 연동하는 setting 부분

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql', # 사용할 DATABASE Engine을 설정  
        'NAME': 'hanseobase', # 로컬에 생성해둔 데이터베이스 스키마 이름  
        'USER': 'root', # 사용자 계정 (root 계정을 사용)  
        'PASSWORD': '1234', # root 계정의 비밀번호를 입력  
        'HOST': 'localhost', # local에서 사용하기 때문에 HOST는 로컬로 설정  
        'PORT': '3306', # MySQL에서 기본적으로 사용하는 port 번호  
    }  
}
```

8.3 테이블 설계



8.4 회원 기능 구현 (accounts)

(1) 모델 (Model)

```
class User(AbstractUser):
    # 학생의 학번 (최대 길이는 9, null은 허용하지 않고, 유일한 값이어야 합니다.)
    student_id = models.CharField(max_length=9, null=False, unique=True)

    # 학생의 이름 (최대 길이는 10, null은 허용하지 않습니다.)
    username = models.CharField(max_length=10, null=False)

    # 참고에서 제공하는 User 기능을 커스텀하기 위해 필요한 코드
    USERNAME_FIELD = 'student_id'

    # 회원가입 시 반드시 입력받아야 하는 속성을 설정하는 코드
    REQUIRED_FIELDS = ['username']
```

(2) 회원 가입, 로그인, 로그아웃에 대한 로직 (View)

① 회원 가입

```
def signup(request):
    # POST 요청일시
    if request.method == 'POST':
        # password1 == password2 값이 일치할 경우
        if request.POST['password1'] == request.POST['password2']:

            # 생성된 user를 DB에 저장합니다
            user = User.objects.create_user(

            # 입력된 값을 각 필드에 저장합니다
                username=request.POST['username'],
                password=request.POST['password1'],
                student_id=request.POST['student_id'],)

            auth.login(request, user)

            # 회원가입 완료후 도메인 페이지로 이동합니다
            return redirect('/')

    # 사용자가 로그인에 실패할 경우 로그인 페이지를 보여줍니다
    return render(request, 'login.html')

# GET 요청일 경우 회원가입 페이지를 보여줍니다
return render(request, 'signup.html')
```

② 로그인

```
def login(request):
    # POST 요청일시
    if request.method == 'POST':

        # student_id 와 password 값을 저장합니다
        student_id = request.POST.get('student_id')
        password = request.POST.get('password')

        # 입력한 student_id , password 에 해당되는계정정보를 있는지 확인을 해줍니다.
        user = authenticate(request, student_id=student_id, password=password)

        if user is not None:
            # db에 user정보에 동일한 student_id와 password를 가진 user가 있으면 로그인을 합니다
            auth.login(request, user)

            # 로그인 완료후 도메인 페이지로 이동합니다
            return redirect('/')
        else:
            # 로그인에 실패할 경우 login 페이지를 보여줍니다
            return render(request, 'login.html')

    # GET요청일 경우 login 페이지를 보여줍니다
    else:
        return render(request, 'login.html')
```

③ 로그아웃

```
def logout(request):
    # 로그아웃 요청을 할 경우 장고의 내장기능을 통해 로그아웃이 됩니다
    auth.logout(request)

    # 도메인 페이지로 이동합니다
    return redirect('/')
```

(3) 회원가입, 로그인, 로그아웃에 대한 URL Mapping

```
urlpatterns = [
    # account/login url 연동시 login 함수로 이동합니다
    path('login/', views.login, name = 'login'),

    # account/signup url 연동시 signup 함수로 이동합니다
    path('signup/', views.signup, name = 'signup'),

    # account/logout url 연동시 logout 함수로 이동합니다
    path('logout/', views.logout, name='logout'),
]
```

8.5 게시글 구현 및 정보(장학금, 다전공) 구현 (db_app)

8.5.1 동아리 게시글 (Club)

(1) 모델 (Model)

```
# 동아리에 대한 모델
class Club(models.Model):
    # 동아리 카테고리들을 Option으로 설정
    opt1 = "음악"
    opt2 = "여행"
    opt3 = "공부"
    opt4 = "종교"
    opt5 = "운동"
    opt6 = "사진"
    opt7 = "문화생활"
    opt8 = "소프트웨어"
    CHOICES = ((opt1, "음악"), (opt2, "여행"), (opt3, "공부"), (opt4, "종교"), (opt5, "운동"), (opt6, "사진"), (opt7, "문화생활"), (opt8, "소프트웨어"))

    # 동아리 이름에 대한 속성 (최대 길이는 30, null을 허용하지 않습니다.)
    name = models.CharField(max_length=30, null=False)

    # 동아리 카테고리에 대한 속성 (위에서 설정한 Option에 대한 값을 고를 수 있고, null을 허용합니다.)
    category = models.CharField(choices=CHOICES, null=True, blank=True)

    # 동아리 설명에 대한 속성 (최대 길이는 MySQL의 기본 varchar(255)로 설정되고, null을 허용합니다.)
    content = models.TextField(null=True)

    # 동아리 연락처에 대한 속성 (최대 길이는 20, null을 허용합니다.)
    tel_number = models.CharField(max_length=20, null=True)

    # 동아리 이미지에 대한 속성 (장고에서 제공하는 image 필드를 사용합니다.)
    # 사진을 등록하지 않으면 기본으로 제공되는 'hanseo_logo.png' 파일이 등록됩니다.
    image = models.ImageField(upload_to='club/', default='hanseo_logo.png', null=True, blank=True)

    # 동아리 SNS나 관련 URL을 설정할 수 있는 속성(null을 허용합니다.)
    url = models.URLField(null=True)

    # 동아리글을 작성하는 유저와 종속관계를 만들고 외래키로서 user의 id(primary key) 값을 가져옵니다.
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True)
```


(2) 동아리 게시물 등록에 필요한 Form

장고에서 Model에 대한 입력을 받을 때, 사용할 수 있는 Form 데이터를 제공합니다.

```
# 동아리 게시글을 작성할 수 있는 Form입니다.
# 장고에서 Model에 대한 Form을 사용할 수 있도록 제공합니다.
class ClubForm(forms.ModelForm):
    class Meta:
        # Model은 앞서 작성한 Club를 사용합니다.
        model = Club

        # Form으로 받을 속성값들을 작성합니다.
        fields = ['name', 'category', 'content', 'tel_number', 'image', 'url']

        # 각 속성 값들을 어떤 Input 타입으로 받을지 설정합니다.
        widgets = {
            'name': forms.TextInput(
                attrs={
                    'class': 'form-control',
                    'placeholder': '동아리 이름',
                },
            ),
            'content': forms.Textarea(
                attrs={
                    'class': 'form-control',
                    'placeholder': '본문',
                },
            ),
            'tel_number': forms.TextInput(
                attrs={
                    'class': 'form-control',
                    'placeholder': '전화번호',
                },
            ),
            'url': forms.TextInput(
                attrs={
                    'class': 'form-control',
                    'placeholder': '관련 URL',
                },
            ),
        }
    }
```

(3) 동아리 게시물 등록, 조회, 수정, 삭제 로직 (View)

① 동아리 게시물 등록과 수정

```
# 동아리 게시물 작성 (Create)에 대한 로직입니다.
def clubCreate(request):
    if request.method == "POST":
        club_form = ClubForm(request.POST, request.FILES)
        context = {"club_form": club_form}
        if club_form.is_valid():
            club = club_form.save(commit=False)
            club.user_id = request.user.id
            club.save()
            return redirect('club')
        else:
            return render(request, 'club_create.html', context)
    else:
        club_form = ClubForm()
        context = {"club_form": club_form}
        return render(request, 'club_create.html', context)

# 동아리 게시물 수정 (Update)에 대한 로직입니다.
def clubUpdate(request, id):
    club = Club.objects.get(pk=id)
    if request.method == "POST":
        club_form = ClubForm(request.POST, instance=club)
        context = {"club_form": club_form}
        if club_form.is_valid():
            club = club_form.save()
            return redirect('club_detail', club.id)
        else:
            club_form = ClubForm(instance=club)
            context = {"club_form": club_form}
            return render(request, 'club_update.html', context)

# 회원의 요청이 POST일때
# 앞서 만든 ClubForm에 요청받은 데이터와 파일을 인자로 제공
# html 파일에 Rendering할 수 있도록 해주는 코드입니다.
# 요청한 데이터 Form이 유효하다면
# 요청한 데이터를 저장합니다. 하지만 아직 db에 commit하지 않습니다.
# 요청한 user에 대한 데이터를 요청한 데이터 정보에 추가합니다.
# db에 요청한 데이터를 commit 합니다.
# 작성이 완료되면 /club/ URL로 되돌아 갑니다.
# 요청한 데이터 Form이 유효하지 않으면
# 동아리 생성 페이지를 다시 Rendering 합니다.
# 회원의 요청이 POST가 아닐때
# 앞서 만든 ClubForm에 대한 정보를 가져옵니다.
# html 파일에 Rendering할 수 있도록 해주는 코드입니다.
# 동아리 생성 페이지를 Rendering 합니다.

# 인자로 받은 id 값을 통해서 해당 id를 primary key로 갖는 동아리를 찾습니다.
# 회원의 요청이 POST일때
# ClubForm에 요청받은 데이터 정보를 인자로 받습니다. 기존 값을 위한 인스턴스 사용
# html 파일에 Rendering할 수 있도록 해주는 코드입니다.
# 요청한 데이터 Form이 유효하다면
# 요청한 데이터를 저장합니다.
# 수정을 완료하면, 해당 동아리 게시글로 다시 되돌아갑니다.
# 회원 요청이 POST가 아닐때
# ClubForm에 기존 값을 위한 인스턴스 사용하여 Rendering 합니다.
# html 파일에 Rendering할 수 있도록 해주는 코드입니다.
# 동아리 수정 페이지를 Rendering 합니다.
```

② 동아리 게시물 조회 (전체 조회, 상세 조회)

```
# 모든 동아리 게시물 조회
def clubView(request):
    try:
        cursor = connection.cursor() # SQL문 사용 시작을 알리는 cursor를 선언합니다.

        # Django에서 제공하는 기본 메서드도 있지만 SQL문을 사용해서 직접 조회해봤습니다.
        sql = "SELECT id, name, category, content, tel_number, image, url FROM hanseobase.dbapp_club;"
        cursor.execute(sql) # 앞서 작성한 SQL문을 연결된 DB에서 실행
        datas = cursor.fetchall() # 실행 결과를 모두 가져옵니다.

        connection.commit() # SQL 실행이 끝났으니 commit을 진행합니다.
        connection.close() # SQL문 사용 끝을 알리기 위해 connection을 닫습니다.

        clubs = [] # SQL을 통해서 가져온 데이터를 html에 사용하기 위한 데이터로 변환
        for data in datas:
            row = {
                'id' : data[0], # 동아리 게시물의 primary key에 해당하는 id 입니다.
                'name' : data[1], # 동아리 게시물의 제목
                'category' : data[2], # 동아리 게시물의 카테고리
                'content' : data[3], # 동아리 게시물의 내용
                'tel_number' : data[4], # 동아리 게시물의 연락처
                'image' : data[5], # 동아리 게시물의 이미지
                'url' : data[6] # 동아리 게시물의 URL
            }
            clubs.append(row) # 반복문을 돌면서 데이터를 하나씩 clubs 리스트에 추가합니다.

    except:
        connection.rollback() # 찾고자 하는 데이터를 찾지 못하거나 예외가 발생하면 rollback 시킵니다.
        print("찾고자 하는 정보가 없습니다.")

    return render(request, 'club.html', { 'clubs' : clubs }) # html에서 데이터를 사용하기 위한 코드입니다.
```

```
# 동아리 게시물 상세 조회
def clubDetailView(request, id):
    try:
        cursor = connection.cursor()

        sql = "SELECT id, name, category, content, tel_number, image, url, user_id FROM hanseobase.dbapp_club WHERE id=(%s);"
        cursor.execute(sql, (id,)) # 전체 조회와는 다르게 찾고자 하는 게시물의 id값을 WHERE문에 넣어서 SQL
        data = cursor.fetchall()

        connection.commit()
        connection.close()

        club = {
            'id' : data[0][0],
            'name' : data[0][1],
            'category' : data[0][2],
            'content' : data[0][3],
            'tel_number' : data[0][4],
            'image' : data[0][5],
            'url' : data[0][6],
            'user_id' : data[0][7]
        }

    except:
        connection.rollback()
        print("찾고자 하는 정보가 없습니다.")

    return render(request, 'club_detail.html', { 'club' : club })
```

③ 동아리 게시물 삭제

```
def clubDelete(request, id):
    club = get_object_or_404(Club, pk=id) # 삭제하려는 동아리 게시물의 id 값을 받아서 club 모델을 통해 해당 게시물 객체를 얻습니다.
    club.delete() # Django에서 제공하는 메서드 delete()를 사용해서 해당 게시물을 삭제합니다.
    return redirect('club/') # 삭제가 완료되면 /club/ URL로 이동합니다.
```

8.5.2 학교 주변 시설 게시물 (Facility)

(1) 모델 (Model)

```
# 주변 시설에 대한 모델
class Facility(models.Model):
    # 주변 시설 카테고리를 Option으로 설정
    opt1= "카페"
    opt2= "음식점"
    opt3= "술집"
    CHOICES = ((opt1, "카페"), (opt2, "음식점"), (opt3, "술집"))

    # 주변 시설 이름에 대한 속성 (최대 길이는 30, null은 허용하지 않습니다.)
    name = models.CharField(max_length=30, null=False)

    # 주변 시설 카테고리에 대한 속성 (위에서 설정한 Option을 사용하고, 최대 길이는 50 null을 허용합니다.)
    category = models.CharField(choices=CHOICES, max_length=50, null=True, blank=True)

    # 주변 시설 내용에 대한 속성 (기본 세팅 varchar(255)가 적용되고, null을 허용합니다.)
    content = models.TextField(null=True)

    # 주변 시설 전화번호에 대한 속성 (최대 길이는 20, null을 허용합니다.)
    tel_number = models.CharField(max_length=20, null=True)

    # 주변 시설 이미지에 대한 속성 (장고에서 제공하는 Image 필드를 사용합니다.)
    # 아무 값도 입력되지 않으면 기본 값으로 설정된 hanseo_logo.png 파일이 들어갑니다.
    image = models.ImageField(upload_to='facility/', default='hanseo_logo.png', null=True, blank=True)

    # 주변 시설의 지도 URL을 등록합니다.
    url = models.URLField(null=True)

    # 주변 시설을 등록한 사용자에게 대한 id 값이 저장되는 속성 (외래키로 user_id를 사용합니다.)
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True)
```

(2) 학교 주변 시설 게시물 등록에 필요한 Form

```
# 학교 주변 게시글을 작성할 수 있는 Form입니다.
# 장고에서 Model에 대한 Form을 사용할 수 있도록 제공합니다.
class FacilityForm(forms.ModelForm):
    class Meta:
        model = Facility
        fields = ['name', 'category', 'content', 'tel_number', 'image', 'url']
        widgets = {
            'name' : forms.TextInput(
                attrs={
                    'class' : 'form-control',
                    'placeholder' : '시설 이름',
                },
            ),
            'content' : forms.Textarea(
                attrs={
                    'class' : 'form-control',
                    'placeholder' : '본문',
                },
            ),
            'tel_number' : forms.TextInput(
                attrs={
                    'class' : 'form-control',
                    'placeholder' : '전화번호',
                },
            ),
            'url' : forms.TextInput(
                attrs={
                    'class' : 'form-control',
                    'placeholder' : '지도 URL',
                },
            ),
        }
    )
```


(3) 학교 주변 시설 게시물 등록, 조회, 수정, 삭제 로직 (View)

① 학교 주변 시설 등록

```
# 학교 주변 시설 등록에 대한 로직입니다.
def facilityCreate(request):
    if request.method == "POST":
        facility_form = FacilityForm(request.POST, request.FILES)
        context = {"facility_form" : facility_form}
        if facility_form.is_valid():
            facility = facility_form.save(commit=False)
            facility.user_id = request.user.id
            facility.save()
            return redirect('facility')
        else:
            return render(request, 'facility_create.html', context)
    else:
        facility_form = FacilityForm(request.POST, request.FILES)
        context = {"facility_form" : facility_form}
        return render(request, 'facility_create.html', context)
```

② 학교 주변 시설 수정

```
# 학교 주변 시설 수정에 대한 로직입니다.
def facilityUpdate(request, id):
    facility = Facility.objects.get(pk=id)
    if request.method == "POST":
        if(facility.user == request.user):
            facility_form = FacilityForm(request.POST, instance=facility)
            context = {"facility_form" : facility_form}
            if facility_form.is_valid():
                facility = facility_form.save()
                return redirect('facility_detail', facility.id)
        else:
            facility_form = FacilityForm(instance=facility)
            context = {"facility_form" : facility_form}
            return render(request, 'facility_update.html', context)
```

③ 학교 주변 시설 조회 (전체 조회, 상세 조회)

```
# 학교 주변 시설 전체 조회를 위한 로직입니다.
def facilityView(request):
    try:
        cursor = connection.cursor()

        # SQL의 SELECT문을 사용하여 학교 주변 시설 테이블에 있는 모든 속성들을 가져옵니다.
        sql = "SELECT id, name, category, content, tel_number, image, url FROM hanseobase.dbapp_facility;"
        result = cursor.execute(sql)
        datas = cursor.fetchall()

        connection.commit()
        connection.close()

        facilities = []
        for data in datas:
            row = {
                'id' : data[0],
                'name' : data[1],
                'category' : data[2],
                'content' : data[3],
                'tel_number' : data[4],
                'image' : data[5],
                'url' : data[6]
            }
            facilities.append(row)

        except:
            connection.rollback()
            print("찾고자 하는 정보가 없습니다.")

        return render(request, 'facility.html', { 'facilities' : facilities })
```

```
# 학교 주변 시설 상세 조회에 대한 로직입니다.
def facilityDetailView(request, id):
    try:
        cursor = connection.cursor()
        sql = "SELECT id, name, category, content, tel_number, image, url, user_id FROM hanseobase.dbapp_facility WHERE id=(%s)"
        cursor.execute(sql, (id,)) # 찾고자 하는 게시글의 id를 WHERE문의 조건으로 제공합니다.
        data = cursor.fetchall()

        connection.commit() # SQL 작업이 끝나면 commit을 합니다.
        connection.close() # connection을 close하여 돌아옵니다.

        facility = {
            'id' : data[0][0],
            'name' : data[0][1],
            'category' : data[0][2],
            'content' : data[0][3],
            'tel_number' : data[0][4],
            'image' : data[0][5],
            'url' : data[0][6],
            'user_id' : data[0][7],
        }
    except:
        connection.rollback()
        print("찾고자 하는 정보가 없습니다.")

    return render(request, 'facility_detail.html', { 'facility' : facility })
```

④ 학교 주변 시설 게시글 삭제

```
# 학교 주변 시설 게시글 삭제에 대한 로직
def facilityDelete(request, id):
    facility = get_object_or_404(Facility, pk=id) # 삭제하려는 게시글의 id 값으로 해당 객체를 얻습니다.
    facility.delete() # 해당 게시글 객체를 삭제합니다.
    return redirect('facility') # 삭제가 완료되면 '/facility/' URL로 돌아옵니다.
```

8.5.3 장학금 정보 (Scholarship)

(1) 모델 (Model)

```
class Scholarship(models.Model):

    # 장학금 이름에 대한 속성 (최대길이는 30, null은 허용하지 않습니다.)
    name = models.CharField(max_length=30, null=False)

    # 장학금 관련 내용에 대한 속성(null 을 허용합니다.)
    content = models.TextField(null=True)

    # 장학금 금액에 대한 속성 (최대길이는 20, null은 허용합니다.)
    money = models.CharField(max_length=20, null=True)
```

(2) 장학금 조회 (전체 조회, 상세 조회) (View)

```
# 장학금 게시물 조회
def scholarshipView(request):
    try:
        cursor = connection.cursor() # SQL문 사용 시작을 알리는 cursor를 선언합니다

        # SQL문을 사용해서 직접 조회해왔습니다
        sql = "SELECT id, name, content, money FROM hanseobase.dbapp_scholarship;"
        result = cursor.execute(sql) # 앞서 작성한 SQL문을 연결된 DB에서 실행
        datas = cursor.fetchall() # 실행 결과를 모두 가져옵니다.

        connection.commit() # SQL 실행이 끝났으니 commit을 진행합니다.
        connection.close() # SQL문 사용 끝을 알리기 위해 connection을 닫습니다.

        scholarship = [] # SQL을 통해서 가져온 데이터를 html에 사용하기 위한 데이터로 변환
        for data in datas:
            row = {
                'id' : data[0], # 장학금 게시글의 primary key에 해당하는 id 입니다.
                'name' : data[1], # 장학금 게시글의 제목
                'content' : data[2], # 장학금 내용
                'money' : data[3] # 장학금 금액
            }
            scholarship.append(row) # 반복문을 돌면서 데이터를 하나씩 scholarship 리스트에 추가합니다

    except:
        connection.rollback() # 찾고자 하는 데이터를 찾지 못하거나 예외가 발생하면 rollback 시킵니다
        print("찾고자 하는 정보가 없습니다.")

    return render(request, 'scholarship.html', { 'scholarship' : scholarship }) # html에서 데이터를 사용하기 위한 코드입니다
```

```
# 해당 id 값을 가진 장학금 게시물 조회
def scholarshipDetailView(request, id):
    try:
        cursor = connection.cursor() # SQL문 사용 시작을 알리는 cursor를 선언합니다.

        sql = "SELECT id, name, content, money FROM hanseobase.dbapp_scholarship WHERE id=(%s);"
        result = cursor.execute(sql, (id,)) # 해당 id를 가진 글을 SQL문을 연결 후 DB에서 실행
        data = cursor.fetchall() # 실행 결과를 모두 가져옵니다.

        connection.commit() # SQL 실행이 끝났으니 commit을 진행합니다.
        connection.close() # SQL문 사용 끝을 알리기 위해 connection을 닫습니다.

        scholarship = {
            'id' : data[0][0], # 해당 id 값을 가진 장학금 글의 primary key
            'name' : data[0][1], # 해당 id 값을 가진 장학금 글의 제목
            'content' : data[0][2], # 해당 id 값을 가진 장학금 내용
            'money' : data[0][3] # 해당 id 값을 가진 장학금 금액
        }

    except:
        connection.rollback() # 찾고자 하는 데이터를 찾지 못하거나 예외가 발생하면 rollback 시킵니다.
        print("찾고자 하는 정보가 없습니다.")

    return render(request, 'scholarship_detail.html', { 'scholarship' : scholarship }) # html에서 데이터를 사용하기 위한 코드입니다
```

8.5.4 다전공 정보 (MultiMajor)

(1) 모델 (Model)

```
# 다전공에 대한 모델
class Multimajor(models.Model):
    # 다전공 카테고리를 Option으로 설정
    opt1 = "부전공"
    opt2 = "복수전공"
    CHOICES = ((opt1, "부전공"), (opt2, "복수전공"))

    # 신청시기에 대한 속성 (최대길이는 20, null은 허용합니다.)
    time = models.CharField(max_length=20, null=True)

    # 구비서류에 대한 속성 (null은 허용합니다.)
    document = models.TextField(null=True)

    # 관련조항에 대한 속성 (최대길이는 50, null은 허용합니다.)
    article = models.CharField(max_length=50, null=True)

    # 참고사항에 대한 속성 (null은 허용합니다.)
    notes = models.TextField(null=True)
```

(2) 다전공 조회 로직 (전체 조회, 상세 조회) (View)

```
def multimajorView(request):
    try:
        cursor = connection.cursor()

        # Django에서 제공하는 기본 메서드 있지만 SQL문을 사용해서 직접 조회해봤습니다.
        sql = "SELECT id, time, document, article, notes, category FROM hanseobase.dbapp_multimajor;"
        result = cursor.execute(sql) # 앞서 작성한 SQL문을 연결된 DB에서 실행
        datas = cursor.fetchall() # 실행 결과를 모두 가져옵니다.

        connection.commit() # SQL 실행이 끝났으니 commit을 진행합니다.
        connection.close() # SQL문 사용 끝을 알리기 위해 connection을 닫습니다.

        multimajor = [] # SQL 을 통해서 가져온 데이터를 html에 사용하기 위한 데이터로 변환
        for data in datas:
            row = {
                'id' : data[0], # 다전공 글의 primary key에 해당하는 id 입니다.
                'time' : data[1], # 다전공 글의 신청시기
                'document' : data[2], # 다전공 글의 구비서류
                'article' : data[3], # 다전공 글의 관련조항
                'notes' : data[4], # 다전공 글의 참고사항
                'category' : data[5] # 다전공 글의 카테고리
            }
            multimajor.append(row) # 반복문을 돌면서 데이터를 하나씩 multimajor 리스트에 추가합니다.

    except:
        connection.rollback() # 찾고자 하는 데이터를 찾지 못하거나 예외가 발생하면 rollback 시킵니다.
        print("찾고자 하는 정보가 없습니다.")
```



```
# 해당 id값에 해당하는 다전공 글을 보여줍니다
def multimajorDetailView(request, id):
    try:
        cursor = connection.cursor()
        # SQL문을 사용해서 직접 조회해봤습니다.
        sql = "SELECT id, time, document, article, notes, category FROM hanseobase.dbapp_multimajor WHERE id=(%s);"
        result = cursor.execute(sql, (id,)) # 해당 id를 가진 글을 SQL문을 연결 후 DB에서 실행
        data = cursor.fetchall()           # 실행 결과를 모두 가져옵니다.

        connection.commit()                # SQL 실행이 끝났으니 commit을 진행합니다.
        connection.close()                 # SQL문 사용 끝을 알리기 위해 connection을 닫습니다.

        multimajor = {
            'id' : data[0][0],              # 해당 id 값의 가진 다전공 글의 primary key
            'time' : data[0][1],            # 해당 id 값을 가진 다전공 글의 신청시기
            'document' : data[0][2],        # 해당 id 값을 가진 다전공 글의 구비서류
            'article' : data[0][3],         # 해당 id 값을 가진 다전공 글의 관련조항
            'notes' : data[0][4],           # 해당 id 값을 가진 다전공 글의 참고사항
            'category' : data[0][5]         # 해당 id 값을 가진 다전공 카테고리
        }

    except:
        connection.rollback()              # 찾고자 하는 데이터를 찾지 못하거나 예외가 발생하면 rollback 시킵니다
        print("찾고자 하는 정보가 없습니다.")

    return render(request, 'multimajor_detail.html', { 'multimajor' : multimajor }) # html에서 데이터를 사용하기 위한 코드입니다
```

8.5.5 선배들이 전하는 이야기 게시글 (Post)

(1) 모델 (Model)

```
class Post(models.Model):
    # 게시글을 작성한 user 정보에 대한 속성입니다.
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True)

    # 게시글 제목에 대한 속성 (최대길이는 30, null은 허용합니다.)
    title = models.CharField(max_length=30, null=True)

    # 게시글 내용에 대한 속성 (null은 허용합니다.)
    content = models.TextField(null=True)

    # 게시글 짧은설명을 나타내는 속성 (최대길이는 30, null은 허용합니다.)
    description = models.CharField(max_length=30, null=True)

    # 아무 값도 입력되지 않으면 기본 값으로 설정된 hanseo_logo.png 파일이 드러납니다
    image = models.ImageField(upload_to='post/', default='hanseo_logo.png', null=True, blank=True)
```

(2) 선배들이 전하는 이야기 등록을 위한 Form

```
class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'content', 'description', 'image']
        widgets = {
            'title' : forms.TextInput(
                attrs={
                    'class' : 'form-control',
                    'placeholder' : '제목',
                },
            ),
            'content' : forms.TextInput(
                attrs={
                    'class' : 'form-control',
                    'placeholder' : '내용',
                },
            ),
            'description' : forms.TextInput(
                attrs={
                    'class' : 'form-control',
                    'placeholder' : '설명',
                },
            ),
        }
    }
```

(3) 선배들이 전하는 이야기 등록, 조회, 수정, 삭제 로직 (View)

① 게시물 등록

```
# 게시물 등록에 대한 로직
def postCreate(request):
    if request.method == "POST":
        post_form = PostForm(request.POST, request.FILES)
        context = {"post_form" : post_form}
        if post_form.is_valid():
            post = post_form.save(commit=False)
            post.user_id = request.user.id
            post.save()
            return redirect('/')
        else:
            return render(request, 'post_create.html', context)
    else:
        post_form = PostForm(request.POST, request.FILES)
        context = {"post_form" : post_form}
        return render(request, 'post_create.html', context)
```

② 게시물 수정

```
# 게시물 수정에 대한 로직
def postUpdate(request, id):
    post = Post.objects.get(pk=id)
    if request.method == "POST":
        post_form = PostForm(request.POST, instance=post)
        context = {"post_form" : post_form}
        if post_form.is_valid():
            post = post_form.save()
            return redirect('post_detail', post.id)
        else:
            post_form = PostForm(instance=post)
            context = {"post_form" : post_form}
            return render(request, 'post_update.html', context)
```

③ 게시물 조회 (전체 조회, 상세 조회)

```
def postView(request):
    try:
        cursor = connection.cursor()
        # SQL 문을 시작하기 위한 cursor를 열어줍니다.

        # 게시물 테이블에서 필요한 정보를 조회할 수 있는 SELECT 문을 작성합니다.
        sql = "SELECT id, title, image, description FROM hanseobase.dbapp_post;"
        result = cursor.execute(sql)
        # 위에서 작성한 SQL 문을 실행합니다.
        datas = cursor.fetchall()
        # 실행 결과를 얻어옵니다.

        connection.commit()
        # 모든 조회작업이 끝난 후 commit을 진행합니다.
        connection.close()
        # 작업이 끝났기 때문에 connection을 닫아줍니다.

        post = []
        # html에서 사용할 수 있도록 데이터를 담아줄 리스트를 선언합니다.
        for data in datas:
            row = {
                'id' : data[0],
                'title' : data[1],
                'image' : data[2],
                'description' : data[3],
            }
            # 게시물 id (primary key 입니다.)
            # 게시물의 제목입니다.
            # 게시물의 이미지 입니다.
            # 게시물의 짧은 설명입니다.

            post.append(row)
            # 리스트에 데이터를 추가해줍니다.

    except:
        connection.rollback()
        # 조회 작업 중 예외가 발생하면 rollback을 진행합니다.
        print("찾고자 하는 정보가 없습니다.")

    return render(request, 'index.html', { 'post' : post })
```

```

def postDetailView(request, id):
    try:
        cursor = connection.cursor()          # SQL 문을 시작하기 위한 cursor를 열어줍니다.
        # 특정 id 값을 primary key로 갖는 게시글에 대한 SELECT 문을 작성해줍니다.
        sql = "SELECT title, content, image ,user_id, id FROM hanseobase.dbapp_post WHERE id=(%s)"
        cursor.execute(sql, (id,))           # 위에서 작성한 SQL 문을 실행시킵니다.
        data = cursor.fetchall()              # 실행 결과를 얻어옵니다.

        post = {
            'title' : data[0][0],              # 특정 게시글의 제목
            'content' : data[0][1],            # 특정 게시글의 본문 내용
            'image' : data[0][2],              # 특정 게시글의 이미지
            'user_id' : data[0][3],            # 특정 게시글을 작성한 유저의 id 값
            'id' : data[0][4],                 # 특정 게시글의 id 값
        }

        user_id = data[0][3]                  # html 화면에 유저에 대한 정보를 Rendering하기 위한 user의 id 값
        # 게시글의 작성한 유저의 정보를 얻어오기 위해서 SELECT 문을 작성해줍니다.
        sql = "SELECT student_id, username FROM hanseobase.accounts_user WHERE id=(%s)"
        cursor.execute(sql, (user_id,))       # 위에서 작성한 SQL문에서 WHERE 조건에 유저의 id를 넣어서 실행합니다.
        data = cursor.fetchall()              # 실행 결과를 얻어옵니다.

        user = {
            'student_id' : data[0][0],         # 게시글을 작성한 유저의 학번
            'username' : data[0][1],          # 게시글을 작성한 유저의 이름
        }

        connection.commit()                  # 조회 작업이 끝난 후 commit을 합니다.
        connection.close()                   # 모든 작업이 끝난 후 connection을 닫아줍니다.

    except:
        connection.rollback()                # 만약 조회 작업 중 예외가 발생 시 rollback 해줍니다.
        print("찾고자 하는 정보가 없습니다.")

    return render(request, 'post_detail.html', { "post" : post , "user" : user})

```

④ 게시글 삭제

```

# 게시글 삭제에 대한 로직
def postDelete(request, id):
    post = get_object_or_404(Post, pk=id)    # 삭제하고자 하는 id 값의 Post 객체를 가져옵니다.
    post.delete()                            # 해당 Post 객체를 삭제합니다.
    return redirect('/')                     # 삭제가 완료되면 메인 페이지로 이동합니다.

```


8.5.6 게시물 및 정보(다전공, 장학금)에 대한 URL Mapping

```
urlpatterns = [
    # 메인 화면 URL 연동시 home 함수로 이동합니다.
    path('', views.home, name = 'home'),

    # club URL 연동시 clubView 함수로 이동합니다.
    path('club', views.clubView, name = 'club'),

    # clubcreate URL 연동시 clubCreate 함수로 이동합니다.
    path('clubcreate', views.clubCreate, name = "club_create"),

    # club/id URL 연동시 id 값을 가지고 clubDetailView 함수로 이동합니다.
    path('club/<int:id>', views.clubDetailView, name = 'club_detail'),

    # club/id/delete URL 연동시 id 값을 가지고 clubDelete 함수로 이동합니다.
    path('club/<int:id>/delete', views.clubDelete, name = 'club_delete'),

    # club/id/update URL 연동시 id 값을 가지고 clubUpdate 함수로 이동합니다.
    path('club/<int:id>/update', views.clubUpdate, name = 'club_update'),

    # facility URL 연동시 facilityView 함수로 이동합니다.
    path('facility', views.facilityView, name = 'facility'),

    # facilitycreate URL 연동시 facilityCreate 함수로 이동합니다.
    path('facilitycreate', views.facilityCreate, name = "facility_create"),

    # facility/id URL 연동시 id 값을 가지고 facilityDetailView 함수로 이동합니다.
    path('facility/<int:id>', views.facilityDetailView, name = 'facility_detail'),

    # facility/id/delete URL 연동시 id 값을 가지고 facilityDelete 함수로 이동합니다.
    path('facility/<int:id>/delete', views.facilityDelete, name = 'facility_delete'),

    # facility/id/update URL 연동시 id 값을 가지고 facilityUpdate 함수로 이동합니다.
    path('facility/<int:id>/update', views.facilityUpdate, name = 'facility_update'),

    # scholarship URL 연동시 scholarshipView 함수로 이동합니다.
    path('scholarship', views.scholarshipView, name = 'scholarship'),

    # scholarship/id URL 연동시 id 값을 가지고 scholarshipDetailView 함수로 이동합니다.
    path('scholarship/<int:id>', views.scholarshipDetailView, name = 'scholarship_detail'),

    # multimajor URL 연동시 multimajorView 함수로 이동합니다.
    path('multimajor', views.multimajorView, name = 'multimajor'),

    # multimajor/id URL 연동시 id 값을 가지고 multimajorView 함수로 이동합니다.
    path('multimajor/<int:id>', views.multimajorDetailView, name = 'multimajor_detail'),

    # postcreate 연동시 postCreate 함수로 이동합니다.
    path('postcreate/', views.postCreate, name = "post_create"),

    # post/id 연동시 해당 id 을 가지고 postDetailView 함수로 이동합니다.
    path('post/<int:id>', views.postDetailView, name = "post_detail"),

    # post/id/delete URL 연동시 해당 id 을 가지고 postDelete 함수로 이동합니다.
    path('post/<int:id>/delete', views.postDelete, name = "post_delete"),

    # post/id/update URL 연동시 해당 id 을 가지고 postUpdate 함수로 이동합니다.
    path('post/<int:id>/update', views.postUpdate, name = "post_update"),
]

# media 파일 경로를 설정해줍니다.
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```


8.6 게시글 및 정보 검색 기능 구현 (search)

(1) 게시글 및 정보 검색에 대한 로직 (View)

```
def searchResult(request):
    # 도메인에서 입력된 파라미터를 kw 값이 있을 경우
    if 'kw' in request.GET:
        query = request.GET.get('kw')

    # 동아리에서 name 필드와 kw값과 동일한 값을 검색합니다
    club = Club.objects.all().filter(
        Q(name__icontains=query)
    )

    # 시설 name 필드와 kw값과 동일한 값을 검색합니다
    facility = Facility.objects.all().filter(
        Q(name__icontains=query)
    )

    # 게시글 title 필드와 kw값과 동일한 값을 검색합니다
    post = Post.objects.all().filter(
        Q(title__icontains=query)
    )

    # 장학금 name 필드와 kw값과 동일한 값을 검색합니다
    scholarship = Scholarship.objects.all().filter(
        Q(name__icontains=query)
    )

    # 동아리, 주변시설을 합병하여 kw와 동일한 필드 이름을 저장합니다
    result = club.union(facility)

    # search 홈페이지에 값을 딕셔너리 형태로 전달합니다
    return render(request, 'search.html', {'query':query, 'result':result, 'post':post, 'scholarship':scholarship})
```

(2) 게시글 및 정보 검색에 대한 URL Mapping

```
urlpatterns = [
    #search url 연동시 searchResult 함수로 이동합니다.
    path('', views.searchResult, name='searchResult'),
]
```