딥러닝 기술을 활용한 기자재 관리 시스템

CSE Capstone project

LKK-99

INDEX

01 진척도 보고

앱 진행 상황

앱 개발 계획

(~10.30)
메인 페이지 구현 : Flutter를 사용하여 Dart로 직접 코딩 (~10.30)
FlutterFlow를 활용하여 재개발 (~11.10)

(~11.20)
카메라 및 알림 기능 : FlutterFlow는 카메라 기능이 없어서 Dart로 직접 코딩해야 할 것으로 예상.
알림기능을 활용하려면 데이터베이스에 접근해야 하기에 이번 주차부터 진행할 예정

(~12.4)
대여 및 반납 페이지 UI 개발 : 대여 페이지 UI 개발 완료
데이터베이스 연동은 추후에 진행할 예정

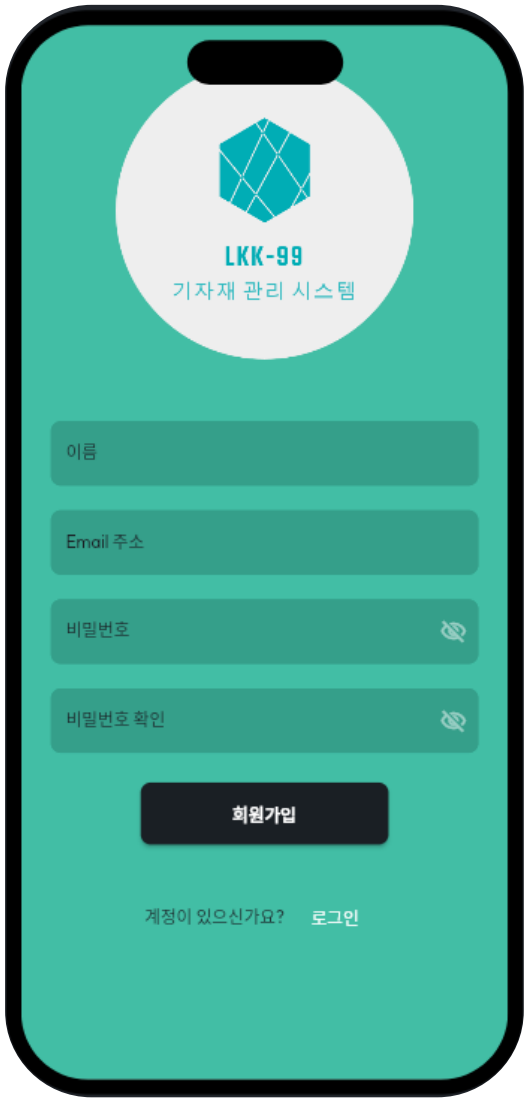
앱 진행 상황

시작 페이지

회원가입과 로그인 버튼을 선택 가능

카메라 버튼을 눌러 비로그인 상태에서도
기자재 인식 및 정보 확인 가능

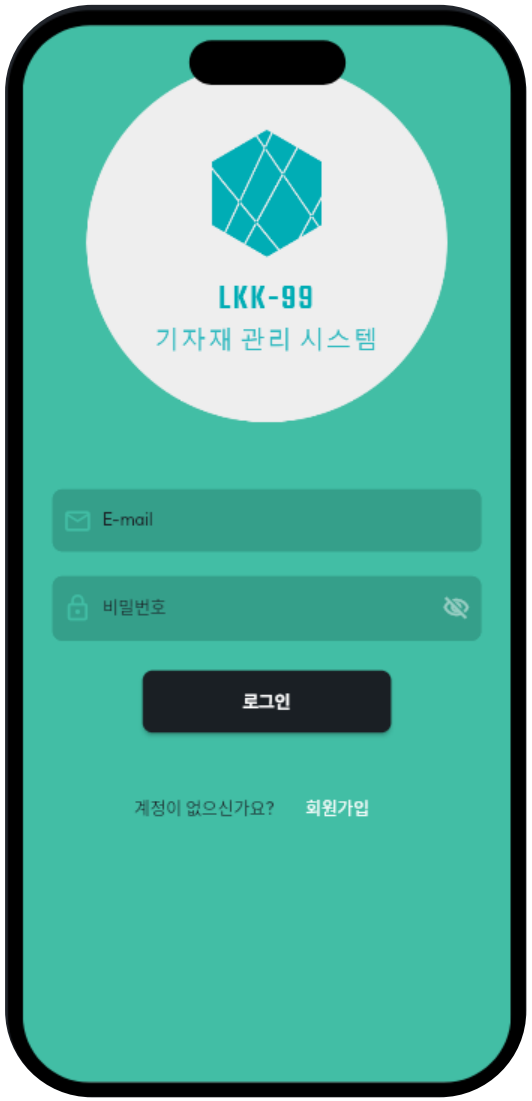
앱 진행 상황



회원가입 페이지

개인정보 (이름, 이메일, 비밀번호)를 입력하여 계정을 생성할 수 있음

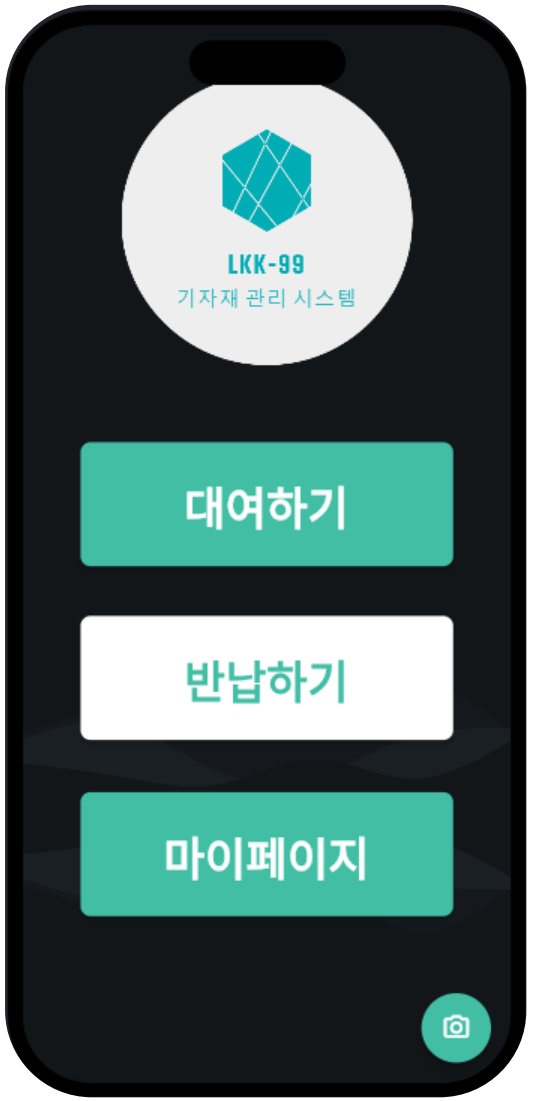
앱 진행 상황



로그인 페이지

가입 되어있는 계정의
이메일과 비밀번호를 입력하여
로그인을 할 수 있음

앱 진행 상황

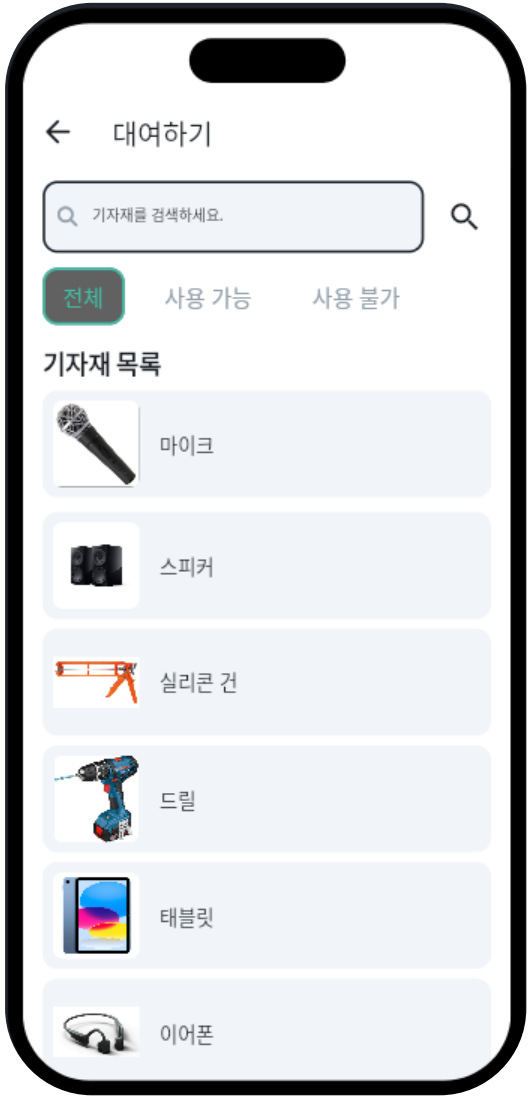


메인 페이지

로그인을 하면 바로 노출되는 페이지
(자동로그인 상태에서는 시작페이지가 됨)

대여, 반납, 마이페이지 의 3개의 탭으로 구성

앱 진행 상황



대여하기 페이지

메인페이지에서 대여하기 탭을 클릭하면 이동

모든 기자재의 리스트를 확인할 수 있음

기자재를 클릭하여 상세 대여 정보 화면으로 넘어갈 수 있음

테이블 형식으로 바꾸고, 상세 정보들도 노출시킬 예정. 분류 기능 추가 예정

앱 진행 상황

상세 대여 정보

기자재 리스트에서 한 기자재 클릭 시

세부 항목들은 추가 예정

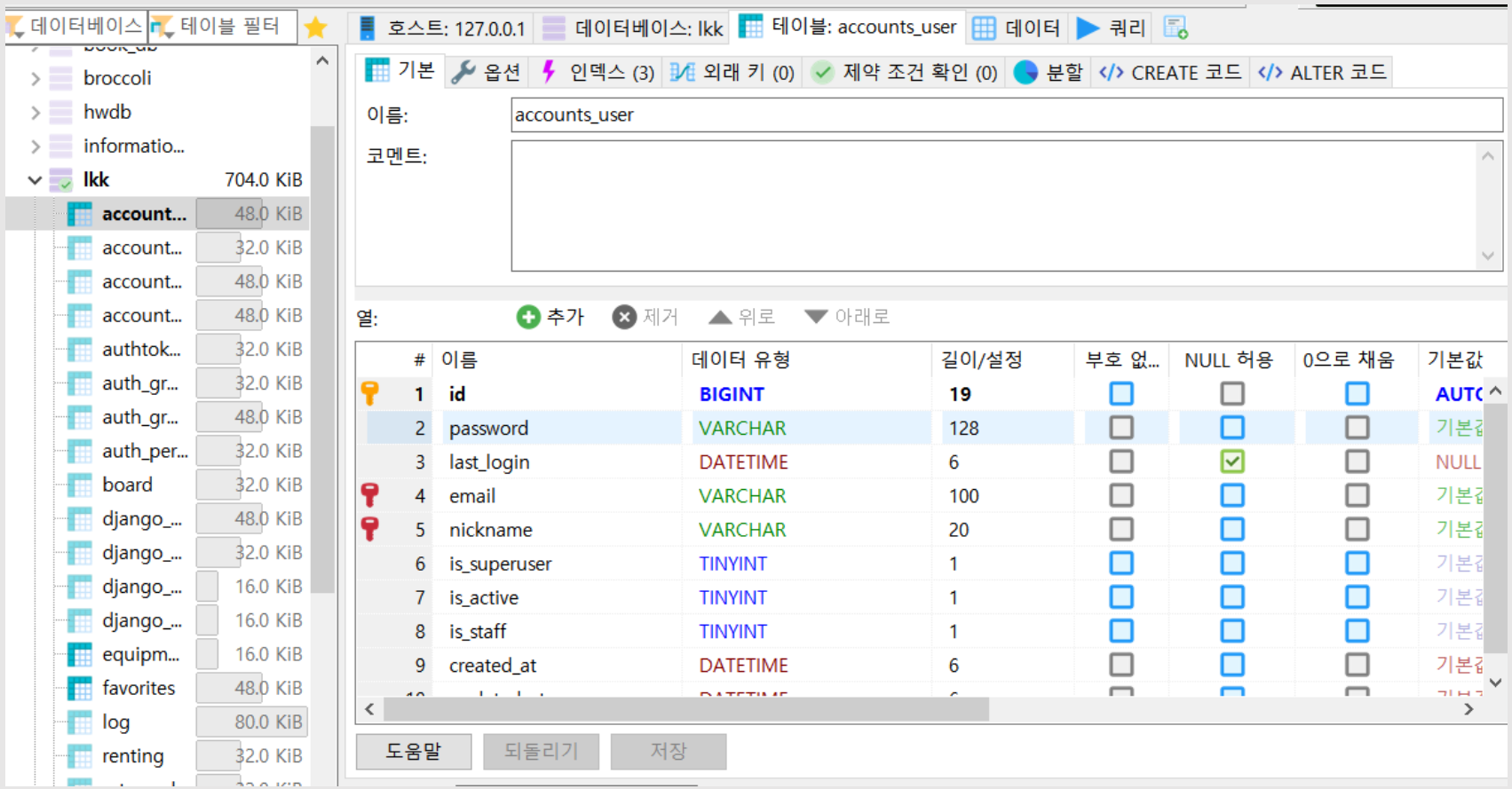
백엔드 변경사항

1. 알림 받는 기능만을 포함하고 있던 예약 테이블을 삭제하고 즐겨찾기 기능에 통합
   -〉 즐겨찾기 한 기자재의 변동 사항에 대해 알림 받기 가능

2. 애매했던 대시보드 기능 확정
   -> 대시보드에서 즐겨찾기에 추가한 기자재의 입출력 내역 확인 가능
   -> 기자재 최신 입출력 현황 확인 가능
   -> 대여 및 반납 승인, 게시판 답변 개인별 알림 확인 가능
   -> 총 대여 횟수와 추천 수에 따른 인기 기자재 순위 확인 가능

3. 데이터베이스 소소한 변경
   -> Django에서 제공하는 커스텀 User 테이블을 사용함에 따라, account_User(커스텀)와 UserInfo로 테이블을 나눔
   -> 반납 완료 테이블 추가(기간(1년)에 따른 나의 반납 완료 리스트 구현 위해)
   -> 기자재 테이블에서 "검색 수" field 삭제(유저별 추천수로 대체-> "좋아요"처럼 구현)

백엔드 변경사항



Django 제공
Account_User 테이블

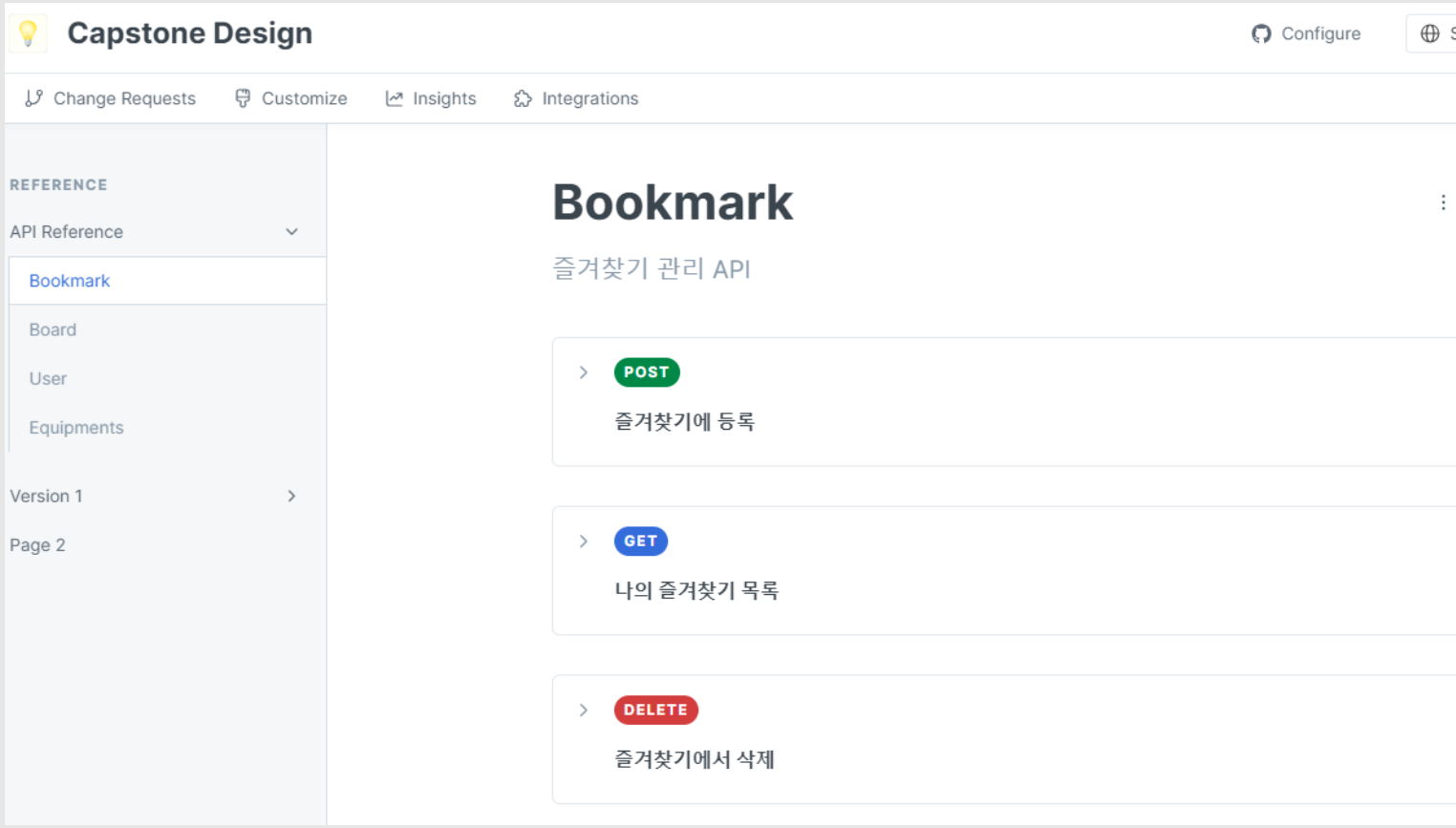
id	password	last_login	email	nickname	is_superuser	is_active	is_s
3	pbkdf2_sha256$320000$4huMwpoy8Ej85PqC4...	(NULL)	1234@naver.com	Juho	0	1	
4	pbkdf2_sha256$320000$raHzAU623wLA7VZhA...	(NULL)	5678@naver.com	Sejin	0	1	
5	pbkdf2_sha256$320000$YIYYbvnXkls9L50PjbP...	(NULL)	jjjj@naver.com	jonghak	0	1	

백엔드 진행 상황(명세서 목록)

계획대로 API 목록 확정 후 구현 시작

지난번 발표 때는 큰 기능만 나눴지만
파트별로 자세히 분류 및 시나리오 작성

즐겨찾기 기능 구현을 위한 API



백엔드 진행 상황(API 명세서)

Board

게시판 관리 API

- POST 게시판에 글 등록
- GET 게시판 글 목록(최신순)
- GET 상세 게시글 정보
- POST 답변 등록
- PUT 질문 글 수정
- PUT 답변 수정
- DELETE 게시글 삭제

게시판 관리를 위한 API 목록

백엔드 진행 상황(API 명세서)

User

계정 관리 API

로그인/로그아웃 관련

- POST /accounts/signin/ — 이메일 회원가입
- POST /accounts/auth/ — 이메일 로그인
- DELETE /accounts/auth/ — 이메일 로그아웃
- GET /accounts/auth/ — 로그인 상태 확인
- GET — 로그인 유저 정보 조회(Nick, email)

회원정보

- POST — 회원정보 입력
- GET /accounts/info/ — 나의 회원정보 조회
- PUT — 회원정보 수정
- DELETE — 회원 탈퇴

User 정보 관리, 로그인/로그아웃/ 소셜 로그인 등을 관리하기 위한 API

백엔드 진행 상황(API 명세서)

Equipments

기자재 관리 API

DashBoard

GET 입출고 현황 조회(최신순)

GET 나의 즐겨찾기 기자재의 로그 내역

알림-> 대여/반납 승인 시 이메일로 알림

기자재 대여/반납 과정

POST 기자재 대여 신청

GET 승인 대기 중인 리스트 조회

GET 나의 반납 완료한 리스트 조회

GET 나의 대여 중인 리스트 조회

GET 나의 연체된 로그 조회

User 정보 관리, 로그인/로그아웃/ 소셜 로그인 등을 관리하기 위한 API

"Django.core.mail" 를 통해 이메일 알림 보내기 가능

백엔드 진행 상황(API 명세서)

기자재 자체 정보

- GET /equip/inventory/
  기자재 상세정보 표시
- GET
  전체 기자재 리스트 표시(이름순)
- GET
  전체 기자재 리스트 표시(가격순)
- GET
  전체 기자재 리스트 표시(총 대여 횟수)
- POST
  기자재 정보 추가
- PUT
  기자재 정보 수정
- DELETE
  기자재 정보 삭제
- POST
  기자재 검색
- POST
  반납 기한 연장
- POST
  기자재 대여 승인
- POST
  반납 신청
- POST
  반납 승인
- DELETE
  대여 승인 거절

백엔드 진행 상황(API 명세서)

POST

기자재 대여 신청

대여 승인 일자가 null인 상태로 로그 테이블에 올리고, 해당 로그 id를 대여 중 테이블에 올림

신청 수량이 재고보다 많다면 자동으로 신청 거부, DB에 로그 올라가지 않음

해당 대여기록 정보를 return

POST

반납 기한 연장

대여 중인 리스트에서 버튼을 누르면 해당 log_id에 대한 반납 기한 연장 가능

연장할 기한(1~7일)을 함께 입력받고 해당 정보를 DB에 반영

API 개발 시작하기 이전에 각각 기능과 프론트에서 사용할 시나리오를 작성해보고 구현 시작

파라미터와 Response도 꼼꼼히 점검

POST

기자재 대여 승인

목록을 보고 해당 log_id에 대해 승인 일자를 변경시킴

기자재 table에 대여 횟수 갱신 필요

이때 log와 대여 중 테이블 모두 바꿔 주어야 함

모델과 수량에 대해 재고 종류 업데이트

승인된 내용 이메일로 알림

POST

반납 승인

Log 테이블의 해당 log_id에 대해 반납 승인 일자를 변경시킴

해당 log_id에 대한 객체를 반납 중 테이블에서 삭제

반납 완료 테이블에 해당 log_id를 추가

모델과 수량에 대해 재고 종류 업데이트

반납 완료 테이블에서 반납 승인 일자가 1년 넘은 건 삭제하기

승인된 내용 이메일로 알림

해당 기자재를 즐겨찾기 하고 있는 유저들에게 알림을 보냄

백엔드 진행 상황(API 구현-User)

```python
Juho
def generate_random_nickname():
    rstring = ''.join(random.choices(string.ascii_letters + string.digits, k=10))
    return rstring
Juho
class User(AbstractBaseUser, PermissionsMixin):
    objects = UserManager()

    email = models.EmailField(max_length=100, blank=False, null=False, unique=True)
    nickname = models.CharField(max_length=20, blank=False, null=False, unique=True,

    is_superuser = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['nickname']

Juho
class UserInfo(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="user")
    name = models.CharField(max_length=20, blank=True, null=True)
    phone_number = models.CharField(max_length=20, blank=True, null=True)
    address = models.CharField(max_length=80, blank=True, null=True)
```

User 모델 -> 커스텀 모델 이용. 필요한 column 선언
Nickname이 없을 경우 Random으로 정의

```python
#django에서 제공하는 커스텀 유저 모델(auth_user)
Juho
class UserManager(BaseUserManager):
    Juho
    def create_user(self, email, nickname, password, **kwargs):
        if not email:
            raise ValueError('Users must have an email address')

        user = self.model(
            email=self.normalize_email(email),
            nickname=nickname
        )
        user.set_password(password)
        user.save(using=self._db)
        return user

    Juho
    def create_superuser(self, email=None, nickname=None, password=None):
        superuser = self.create_user(
            email=email,
            nickname=nickname,
            password=password,
        )
        superuser.is_staff = True
        superuser.is_superuser = True
        superuser.is_active = True
        superuser.save(using=self._db)
        return superuser
```

None
The sole
is freque
of a valu
not pass
None are

백엔드 진행 상황(API 구현-User)

```python
from django.urls import path

from Accounts.views import UserInfoAPIView, AuthAPIView, SigninAPIView

urlpatterns = [
    path('signin/', SigninAPIView.as_view()),
    path('auth/', AuthAPIView.as_view()),
    path('info/', UserInfoAPIView.as_view()),
]
```

회원가입, 로그인/로그아웃, 회원정보 조회 API

각각 URL과 serializer 파일

Serializer는 선언한 모델의 직렬화를 통해
원하는 대로 Response를 가능하게 함

```python
from rest_framework import serializers

from Accounts.models import User, UserInfo

class UserSerializer(serializers.ModelSerializer):
    def create(self, validated_data):
        user = User.objects.create_user(
            email=validated_data['email'],
            nickname=validated_data['nickname'],
            password=validated_data['password']
        )
        return user

    class Meta:
        model = User
        fields = ['id', 'password', 'last_login', 'email', 'nickname', 'is_superuser',


class UserInfoSerializer(serializers.ModelSerializer):
    class Meta:
        model = UserInfo
        fields = '__all__'
```

백엔드 진행 상황(API 구현-User)

```python
class SigninAPIView(APIView):
    # email과 password, nickname을 통한 회원가입 API
    Juho
    def post(self, request):
        serializer = UserSerializer(data=request.data)
        if serializer.is_valid():
            user = serializer.save()

            token = TokenObtainPairSerializer.get_token(user)
            refresh_token = str(token)
            access_token = str(token.access_token)
            res = Response(
                {
                    "user": serializer.data,
                    "message": "Signin Success",
                    "token": {
                        "access": access_token,
                        "refresh": refresh_token,
                    },
                },
                status=status.HTTP_200_OK
            )

            res.set_cookie("access", access_token, httponly=True)
            res.set_cookie("refresh", refresh_token, httponly=True)

            return res
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

회원가입

```
{
    "nickname": "Juho",
    "password": "1234",
    "email": "1234@naver.com"
}
```

request를 UserSerializer를 통해 직렬화 후 토큰큰을 발행하고 문자열로 변환

쿠키를 설정하고 토큰을 안전하게 저장

토큰 만료에 따른 오류가 발생하는 것을 해결하는 것이 현재 목표(30분으로 설정했는데도 오류)

백엔드 진행 상황(API 구현-User)

REST Framework를 통해 테스트

Django REST framework

Signin Api

Signin Api

POST /accounts/signin/

HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept


```
{
    "user": {
        "id": 7,
        "password": "pbkdf2_sha256$320000$c6whyE7JSNxfw9mZlWsoS6$27Z6pAmvy7aXTmbBBwDtWKb9Zv26S/uf17vjGU620w0=",
        "last_login": null,
        "email": "juho@naver.com",
        "nickname": "Hoju",
        "is_superuser": false,
        "is_active": true,
        "is_staff": false,
        "created_at": "2023-11-13T04:46:27.976490+09:00",
        "updated_at": "2023-11-13T04:46:27.976490+09:00"
    },
    "message": "Signin Success",
    "token": {
        "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjk5ODIwMTg3LCJpYXQiOjE2OTk4MTgzODcsImp0aSI6IjU1OGM4MWNhNjg0YjQ5YmNiNWI5ZDY5
        "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTcwMDA3NzU4NywiaWF0IjoxNjk5ODE4Mzg3LCJqdGkiOiJhNGZiOTljZWMxN2Q0OTBjODJjZW
    }
}
```


Django REST framework

Signin Api

Signin Api

POST /accounts/signin/

HTTP 400 Bad Request
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept


```
{
    "email": [
        "user의 email은/는 이미 존재합니다."
    ],
    "nickname": [
        "user의 nickname은/는 이미 존재합니다."
    ]
}
```

백엔드 진행 상황(API 구현-User)

```python
def post(self, request):
    # user 인증
    user = authenticate(
        username=request.data.get("email"), password=request.data.get("password")
    )

    if user is not None:
        serializer = UserSerializer(user)

        token = TokenObtainPairSerializer.get_token(user)
        refresh_token = str(token)
        access_token = str(token.access_token)
        res = Response(
            {
                "user": serializer.data,
                "message": "Login Success",
                "token": {
                    "access": access_token,
                    "refresh": refresh_token
                },
            },
            status=status.HTTP_200_OK
        )

        res.set_cookie("access", access_token, httponly=True)
        res.set_cookie("refresh", refresh_token, httponly=True)
        return res
    else:
        failMessage = {
```

```python
    }
    return Response(failMessage, status=status.HTTP_400_BAD_REQUEST)
```

로그인

```
{
    "password": "1234",
    "email": "1234@naver.com"
}
```

Authenticate를 통해 user를 인증하고 토큰을 발급

쿠키를 설정하고 토큰을 안전하게 저장
로그인 실패 시 에러 반환

백엔드 진행 상황(API 구현-User)

REST Framework를 통해 테스트

Auth Api

`GET /accounts/auth/`

```
HTTP 404 Not Found
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "message": "로그인 되어 있지 않습니다. 로그인 해 주세요."
}
```

Auth Api

`POST /accounts/auth/`

```
HTTP 400 Bad Request
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "message": "아이디 또는 비밀번호가 일치하지 않습니다."
}
```

Auth Api

DELETE OPTIONS GET

`POST /accounts/auth/`

```
HTTP 200 OK
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "user": {
        "id": 3,
        "password": "pbkdf2_sha256$320000$4huMwpoy8Ej85PqC4RPDEs$hewz9UIbdWbHz0gRZu4guv9ST6mUn+oHMdHs3U5iOMQ=",
        "last_login": null,
        "email": "1234@naver.com",
        "nickname": "Juho",
        "is_superuser": false,
        "is_active": true,
        "is_staff": false,
        "created_at": "2023-11-12T23:57:52.973486+09:00",
        "updated_at": "2023-11-12T23:57:52.973486+09:00"
    },
    "message": "Login Success",
    "token": {
        "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjk5ODIwNTE1LCJpYXQiOjE2OTk4MTg3MTUsImp0aSI6IjlhZmQyYWVlN2IwZjQwOTY4YTUyNDc
        "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTcwMDA3NzkxNSwiaWF0IjoxNjk5ODE4NzE1LCJqdGkiOiIwY2EwYWRkYTVkYzE0MjRkYmNlZT
    }
}
```

로그인 되어 있을 때는 현재 유저의 로그인 정보 반환.
패스워드는 암호화 후 반환

백엔드 진행 상황(API 구현-User)

```
#로그아웃 API
Juho
def delete(self, request):
    response= Response({
        "message": "로그아웃 성공"
    })
    response.delete_cookie("access")
    response.delete_cookie("refresh")
    return response
```

로그아웃

쿠키에서 토큰 삭제

백엔드 진행 상황(API 구현-User)

REST Framework를 통해 테스트

Auth Api

```
DELETE /accounts/auth/
```

```
HTTP 200 OK
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "message": "로그아웃 성공"
}
```

백엔드 진행 상황(API 구현-User)

```python
import jwt
from django.shortcuts import redirect

from rest_framework import status
from rest_framework.response import Response

from Accounts.models import User
from config.settings import SECRET_KEY


Juho
def login_check(func):
    Juho
    def wrapper(self, request, *args, **kwargs):
        try:
            access = request.COOKIES.get('access')
            payload = jwt.decode(access, SECRET_KEY, algorithms='HS256')
            user = User.objects.get(id=payload['user_id'])
            request.user = user
        except jwt.exceptions.DecodeError:
            return Response({'message': 'INVALID TOKEN'}, status=status.HTTP_400_BAD_REQUEST)
        except User.DoesNotExist:
            return Response({'message': 'INVALID USER'}, status=status.HTTP_400_BAD_REQUEST)
        return func(self, request, *args, **kwargs)
    return wrapper
```

로그인 체크(유저 확인)

Utils.py에 Login check 기능 구현

현재 쿠키에서 access 토큰을 불러와 User_id를 추출하고 이를 통해 현재 접속한 유저의 정보를 받아올 수 있다.

백엔드 진행 상황(API 구현-User)

```
Juho
class UserInfoAPIView(APIView):
    Juho
    @login_check
    def get(self, request):
        try:
            userInfo = UserInfo.objects.get(user_id = request.user.id)
            serializer = UserInfoSerializer(userInfo)
            return Response(serializer.data)
        except UserInfo.DoesNotExist:
            failMessage= {
                "message": "해당 유저의 정보가 존재하지 않습니다."
            }
            return Response(failMessage, status=status.HTTP_400_BAD_REQUEST)
```

이를 통한 나의 회원정보 조회 구현

Login check 후 request.user.id를 통해 현재 로그인한 유저의 정보만 데이터베이스에서 불러온다.

실패시 failMessage 반환

백엔드 진행 상황(API 구현-User)

REST Framework를 통해 테스트

Auth Api

`GET /accounts/auth/`

```
HTTP 200 OK
Allow: GET, POST, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "user": {
        "id": 3,
        "password": "pbkdf2_sha256$320000$4huMwpoy8Ej85PqC
        "last_login": null,
        "email": "1234@naver.com",
        "nickname": "Juho",
        "is_superuser": false,
        "is_active": true,
        "is_staff": false,
        "created_at": "2023-11-12T23:57:52.973486+09:00",
        "updated_at": "2023-11-12T23:57:52.973486+09:00"
    },
    "message": "로그인 중입니다."
}
```

Django REST framework

User Info Api

User Info Api

`GET /accounts/info/`

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 1,
    "name": "이주호",
    "phone_number": "01023456789",
    "address": "인하아리스타",
    "user": 3
}
```

Django REST framework

User Info Api

User Info Api

`GET /accounts/info/`

```
HTTP 400 Bad Request
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "message": "해당 유저의 정보가 존재하지 않습니다."
}
```

3번 유저로 로그인 된 상황에서 Userinfo return

백엔드 진행 상황(API 구현-Equipment)

```python
from django.db import models

import Accounts.models
Juho
class Equipment(models.Model):
    model_name=models.CharField(max_length=50, primary_key=True)
    name=models.CharField(max_length=30, default=None)
    type=models.CharField(max_length=30, default=None)
    price=models.IntegerField(blank=True, null=True, default=0)
    repository=models.CharField(max_length=30, default=None)
    total_rent=models.IntegerField(blank=True, null=True, default=0)
    total_stock=models.IntegerField(blank=True, null=True, default=0)
    current_stock=models.IntegerField(blank=True, null=True, default=0)
    manufacturer= models.CharField(max_length=30, blank=True, null=True, default=None)
    recommend_count=models.IntegerField(blank=True, null=True, default=0)

    #recommend_user = models.ManyToManyField(Accounts.models.User, related_name='recommend_user')

    Juho
    class Meta:
        managed = False
        db_table= 'equipment'
```

```python
from rest_framework import serializers

from Equipments.models import Equipment
Juho
class EquipmentSerializer(serializers.ModelSerializer):
    Juho
    class Meta:
        model =Equipment
        fields = '__all__'
```

```python
from django.urls import path
from Equipments.views import InventoryAPIView

from . import views

urlpatterns= [
    path('inventory/<str:pk>', InventoryAPIView.as_view()),
]
```

Equipment 모델 -> DB에 맞게 field 선언

백엔드 진행 상황(API 구현-Equipment)

```python
from Equipments.serializers import EquipmentSerializer
from Equipments.models import Equipment

from rest_framework import status
from rest_framework.response import Response
from rest_framework.views import APIView

Juho
class InventoryAPIView(APIView):
    # 기자재 상세정보 조회 API
    Juho
    def get(self, request, pk):
        try:
            equip = Equipment.objects.get(pk=pk)
            serializer = EquipmentSerializer(equip)

            return Response(serializer.data)
        except Equipment.DoesNotExist:
            failMessage = {
                "message": "해당 기자재의 정보가 존재하지 않습니다."
            }
            return Response(failMessage, status=status.HTTP_404_NOT_FOUND)
```

URL에 기자재 모델명(PK) 를 전달하면
그에 따른 상세정보를 return하는 API

해당 기자재나 상세정보가 존재하지 않을 시
404 에러 리턴

백엔드 진행 상황(API 구현-Equipment)

REST Framework를 통해 테스트

Inventory Api

GET /equip/inventory/15UD50P-GX3WK

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept


```
{
    "model_name": "15UD50P-GX3WK",
    "name": "오디세이 ",
    "type": "laptop computer",
    "price": 2000000,
    "repository": "Inha University",
    "total_rent": 3,
    "total_stock": 2,
    "current_stock": 1,
    "manufacturer": "Samsung",
    "recommend_count": 0
}
```


127.0.0.1:8000/equip/inventory/15UD50P-GX3WK

Inventory Api

Inventory Api

GET /equip/inventory/15

HTTP 404 Not Found
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept


```
{
    "message": "해당 기자재의 정보가 존재하지 않습니다."
}
```

백엔드– 앞으로의 계획

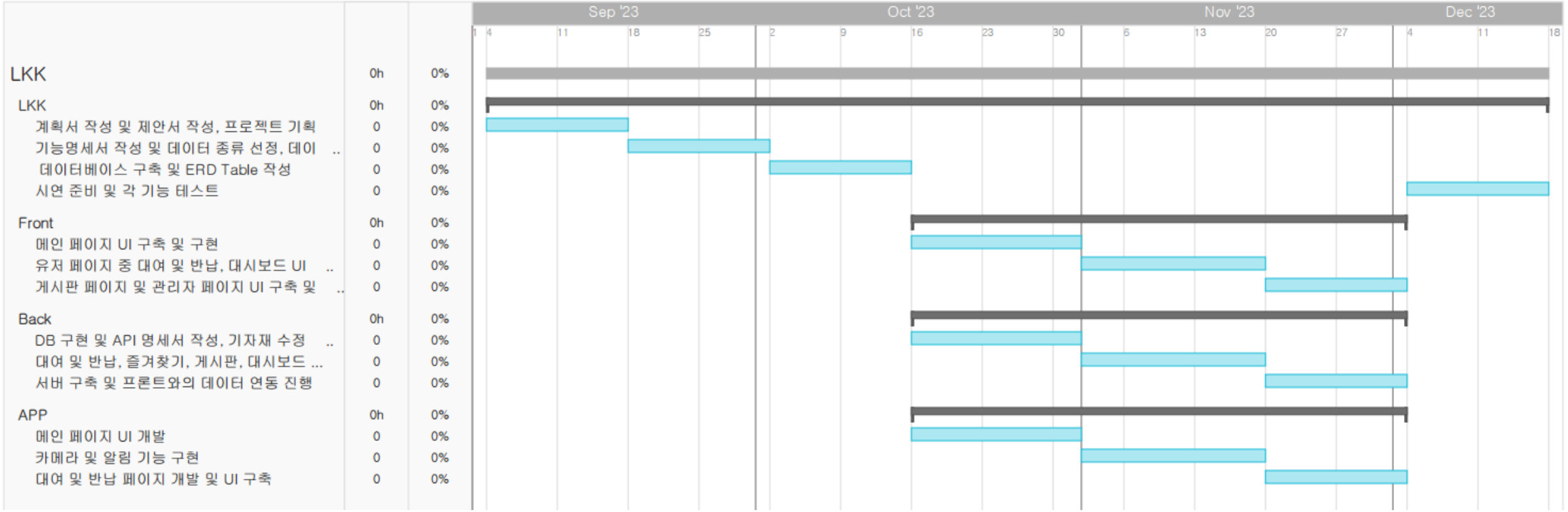
계획대로 이번 주 내에 REST Framework를 통해 테스트하며 남은 API들을 구현 예정.

이후 구름 IDE나 EC2를 통해 테스트 서버를 열고 앱 및 웹 프론트와 데이터를 전달하고

연동을 진행해볼 예정

02 향후 계획 및 변경 사항

향후 계획



현재 딥러닝보다 페이지 구현과 API 개발에 우선순위를 두고 매진 중.

11월 3주차에 API 개발을 마무리하고 프론트 연동 부분과 함께, 딥 러닝 모델을 YOLO로 확정하고 AI Hub에서 얻은 축산 기자재 데이터셋를 토대로 데이터 라벨링 후 학습하는 과정을 다음 발표에 삽입할 예정.

THANK YOU!