

2차년도 주요 결과물

(과제명) 대규모 분산 에너지 저장장치 인프라의 안전한 자율운영
및 성능 평가를 위한 지능형 SW 프레임워크 개발
(과제번호) 2021-0-00077

- 결과물명 : 에너지 저장 장치 전주기별 데이터셋 배포 처리 모듈(SW)
- 작성일자 : 2022년 12월 1일

과학기술정보통신부 SW컴퓨팅산업원천기술개발사업
“2차년도 주요 결과물” 로 제출합니다.

수행기관	성명/직위	확인
한국전자기술연구원	최효섭/책임연구원	chs

정보통신기획평가원장 귀하

사 용 권 한

본 문서에 대한 서명은 한국전자기술연구원 내부에서 본 문서에 대하여
수행 및 유지관리의 책임이 있음을 인정하는 것임.

본 문서는 작성, 검토, 승인하여 승인된 원본을 보관한다.

작성자 :	임홍휘	일자 :	2022. 12. 01
-------	-----	------	--------------

검토자 :	김창우	일자 :	2022. 12. 02
-------	-----	------	--------------


승인자 :	최효섭	일자 :	2022. 12. 03
-------	-----	------	--------------

제 · 개정 이력

버전	변경일자	제·개정 내용	작성자
1.0	2022-12-01	최초 등록	임홍휘

목 차

1. 개요	-----	1
2. 에너지 저장 장치 전주기별 데이터셋 배포 처리 SW 아키텍처 설계	-	3
3. 에너지 저장 장치 전주기별 데이터셋 API 설계	-----	4
4. 에너지 저장 장치 전주기별 데이터셋 배포 처리 SW 개발	-----	7

	에너지 저장 장치 전주기별 데이터셋 배포 처리 모듈(SW)	
	프로젝트	대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 SW 프레임워크 개발

1. 개요

□ 목적

- 본 명세서의 목적은 대규모 분산에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 안전SW 프레임워크를 개발하기 위해 에너지 저장 장치 전주기별 데이터셋 배포 처리 SW 구축에 필요한 설계 및 개발 방법을 기록한다.

□ 범위

- 본 설계서는 에너지 저장 장치 전주기별 데이터셋의 배포 처리에 관한 SW 아키텍처와 API의 설계 및 SW 개발 방법을 중심으로 설명한다.

□ 시스템 개요

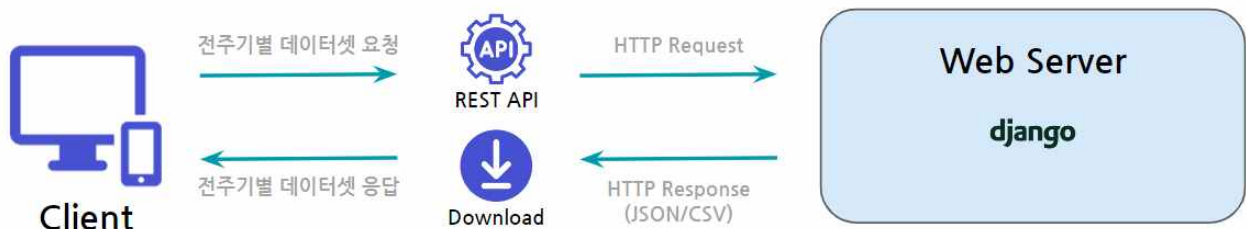



그림 1

- 에너지 저장 장치 전주기별 데이터셋의 배포 처리를 담당하는 시스템은 웹 서비스다. 클라이언트는 API Call 또는 웹 UI(다운로드)를 통해 데이터셋을 요청한다. 클라이언트와 웹 서버의 통신은 내부적으로 HTTP이고, 클라이언트의 요청 방식에 따라 웹 서버는 ‘JSON’ 데이터 포맷 또는 ‘CSV’ 파일로 응답한다.

	에너지 저장 장치 전주기별 데이터셋 배포 처리 모듈(SW)	
	프로젝트	대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 SW 프레임워크 개발

□ 관련 계획 및 표준

o 본 설계서는 아래 계획 및 표준을 참고한다.

구분	식별자	세부 내용	설명
계획서	-	공개 SW 서비스 요구사항정의서	대규모 분산 에너지 저장 장치 인프라의 안전한 자율운영 및 성능 평가를 위한 데이터를 수집하여 전주기별 에너지 데이터셋, 에너지 분석 및 학습 인터페이스를 일반 사용자에게 공개하는 웹 기반의 공개 SW 서비스 개발 요구 사항을 정의한 문서
가이드	-	행정·공공기관 웹사이트 구축·운영 가이드(2021-03, 행정안전부)	행정기관 및 공공기관의 웹사이트 구축(개발)을 추진하는 발주자와 웹사이트를 운영하는 관리자가 지켜야 할 다양한 기준과 관련 사항(규정)들을 정리한 웹사이트 운영관리 활용 안내서
기술	-	RESTAPI	웹 시스템을 위한 소프트웨어 아키텍처의 한 형식으로 HTTP를 활용하는 많은 API가 REST 원리를 따라 제공됨.
설계서	ISO/IEC 9126	9126-1 (품질 모델) 9126-2 (외부 품질) 9126-3 (내부 품질) 9126-4 (사용 품질)	품질 특성 및 측정기준을 제시 소프트웨어의 기능성, 신뢰성, 사용성, 효율성, 유지보수 용이성, 이식성
	ISO/IEC 14598	14598-1 (개요) 14598-2 (계획과 관리) 14598-3 (개발자용 프로세스) 14598-4 (구매자용 프로세스) 14598-5 (평가자용 프로세스) 14598-6 (평가 모듈)	ISO 9126에 따른 제품 평가 표준: 반복성, 공정성, 객관성, 재생산성
	ISO/IEC 12119	소프트웨어 패키지 -제품설명서 -사용자문서 -프로그램과 데이터	패키지 SW 품질 요구사항 및 테스트

2. 에너지 저장 장치 전주기별 데이터셋 배포 처리 SW 아키텍처 설계

□ 개요

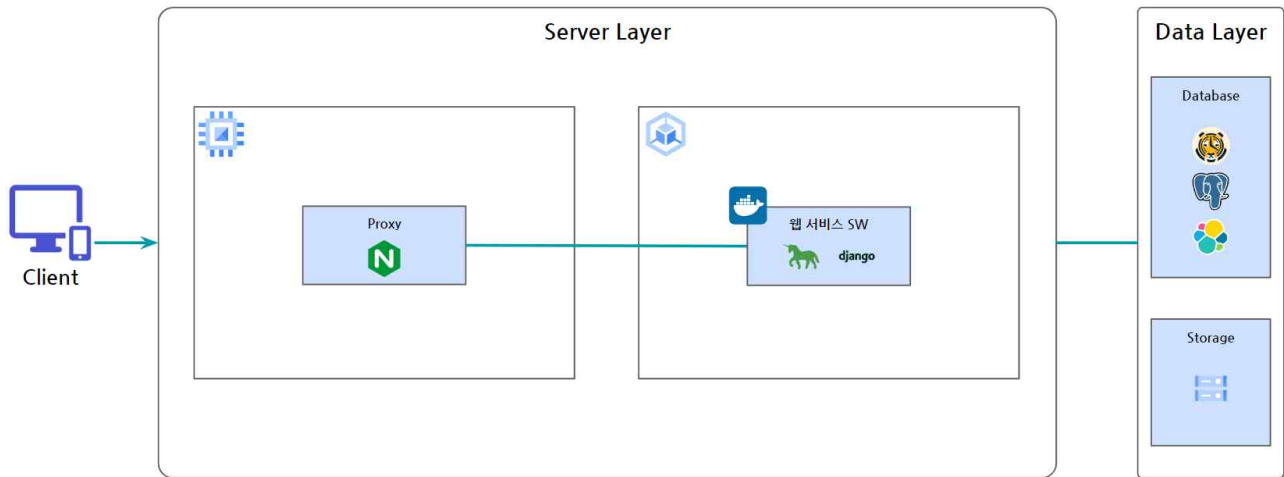


그림 2 에너지 저장 장치 전주기별 데이터셋 배포 처리 SW 아키텍처

- 에너지 저장 장치 전주기별 데이터셋은 웹 서비스로 배포된다. 웹 서비스는 크게 클라이언트의 요청에 응답을 제공하는 Server Layer와 데이터셋을 저장 및 관리하는 Data Layer로 구성된다. Server Layer에서 안정적인 서버 환경을 제공하는 Cloud 가상 서버를 활용해 Proxy와 웹 서비스 SW를 구축하고, Data Layer에서 데이터셋 저장 및 관리를 위한 데이터베이스 관리 시스템과 웹 서비스에 정적 리소스를 제공하는 Cloud Storage를 구축해 웹 서비스와 연결한다. 이 SW 아키텍처는 클라이언트가 HTTP API 요청 또는 웹 UI를 통해 관련 전주기별 데이터셋 요청에 응답할 수 있는 구성이다.

3. 에너지 저장 장치 전주기별 데이터셋 API 설계

□ 데이터셋 종류 및 특징

○ 데이터셋 종류

- 에너지 저장 장치 운영 데이터(구성 요소:Bank, Rack, PCS(전력변환장치), ETC(기타))

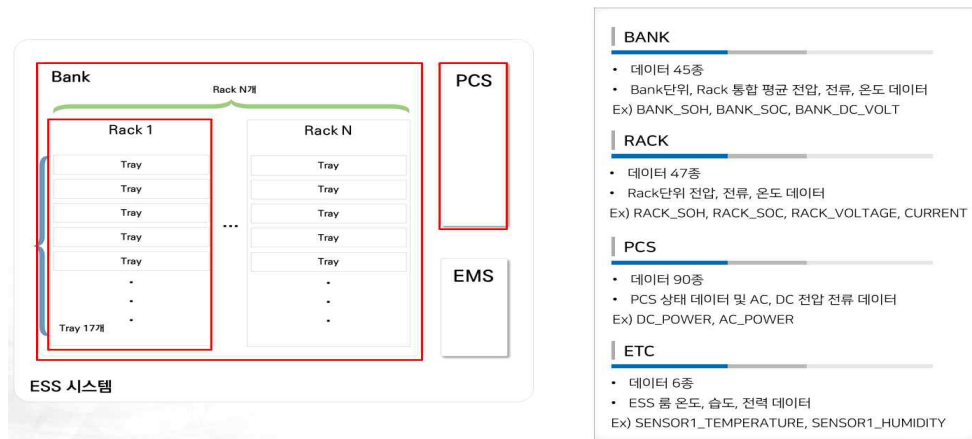


그림 3 에너지 저장 장치 구성 요소 및 데이터 종류
(예. 시온유 태양광 운영 사이트)

○ 데이터셋 특징

- 에너지 저장 장치 운영 데이터는 RDBMS(관계형 데이터베이스 관리 시스템)에서 관리한다. 운영 사이트별로 데이터베이스를 구분하고, 구성 요소별로 독립적인 테이블에 데이터를 저장 및 관리한다. 각 데이터는 시계열 데이터의 특성이 있다. 공통적으로 ‘시간’, ‘Bank 번호’, 또는 ‘Rack 번호’의 데이터 컬럼을 가진다.(단, PCS, ETC 데이터는 물리적인 특성으로 ‘Bank 번호’ 또는 ‘Rack 번호’ 데이터 컬럼이 없을 수도 있다.)

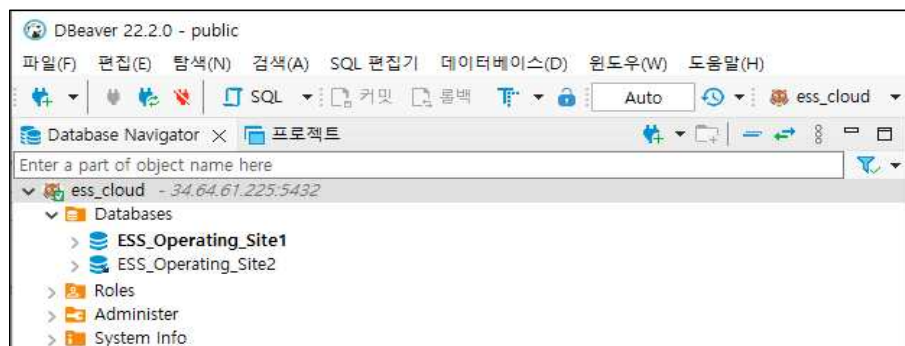


그림 4 에너지 저장 장치 운영사이트별 데이터베이스

bank	rack	pcs	etc
123 TIMESTAMP timestamptz 123 BANK_ID int4 123 BANK_SOC float8 123 BANK_SOH float8 123 BANK_DC_VOLT float8 123 BANK_DC_CURRENT float8 123 BANK_POWER float8 123 MAX_CELL_VOLTAGE_OF_BANK float8 123 MIN_CELL_VOLTAGE_OF_BANK float8 123 MAX_CELL_TEMPERATURE_OF_BANK float8 123 MIN_CELL_TEMPERATURE_OF_BANK float8 123 MAX_MODULE_TEMPERATURE float8 123 MIN_MODULE_TEMPERATURE float8 123 MAX_MODULE_HUMIDITY float8 123 MIN_MODULE_HUMIDITY float8 123 RACK_TEMPERATURE_IMBALANCE_WARNING int4 123 RACK_UNDER_TEMPERATURE_WARNING int4 123 RACK_OVER_TEMPERATURE_WARNING int4 123 RACK_VOLTAGE_IMBALANCE_WARNING int4	123 TIMESTAMP timestamptz 123 BANK_ID int4 123 RACK_ID int4 123 RACK_SOC float8 123 RACK_SOH float8 123 RACK_CURRENT float8 123 RACK_VOLTAGE float8 123 RACK_MAX_CELL_VOLTAGE float8 123 RACK_MAX_CELL_VOLTAGE_POSITION float8 123 RACK_MIN_CELL_VOLTAGE float8 123 RACK_MIN_CELL_VOLTAGE_POSITION float8 123 RACK_CELL_VOLTAGE_GAP float8 123 RACK_CELL_VOLTAGE_AVERAGE float8 123 RACK_MAX_CELL_TEMPERATURE float8 123 RACK_MAX_CELL_TEMPERATURE_POSITION float8 123 RACK_MIN_CELL_TEMPERATURE float8 123 RACK_MIN_CELL_TEMPERATURE_POSITION float8 123 RACK_CELL_TEMPERATURE_GAP float8 123 RACK_MAX_MODULE_TEMPERATURE float8	123 TIMESTAMP timestamptz 123 AI_VDC float8 123 AI_DC float8 123 AI_PDC float8 123 AI_FREQ float8 123 AI_VAB_RMS float8 123 AI_VBC_RMS float8 123 AI_VCA_RMS float8 123 AI_VAS_RMS float8 123 AI_IBS_RMS float8 123 AI_IC_S_RMS float8 123 AI_SAC float8 123 AI_PAC float8 123 AI_QAC float8 123 AI_FF float8 123 AI_C_KWH_ACH float8 123 AI_C_KWH_ACL float8 123 AI_D_KWH_ACH float8 123 AI_D_KWH_ACL float8	123 TIMESTAMP timestamptz 123 SENSOR1_TEMPERATURE float8 123 SENSOR1_HUMIDITY float8 123 SENSOR2_TEMPERATURE float8 123 SENSOR2_HUMIDITY float8 123 ACTIVE_POWER_TOTAL float8 123 ACTIVE_ENERGY_TOTAL_HIGH float8

그림 5 운영 데이터 ER 다이어그램(데이터 컬럼 일부 생략)

□ API 설계


○ 개요

- 클라이언트가 주로 API Call 또는 웹UI 통한 다운로드 방식으로 전주기별 데이터셋을 요청한다고 가정하여 여기서는 내용을 구분한다.(실제 2가지 방식 모두 REST API 설계를 기반으로 함.)

○ API Call

- 위의 데이터셋 종류 및 특징을 기반으로 운영 사이트와 구성 요소는 계층적 관계로 API를 설계할 수 있다. 서버는 운영 사이트별, 구성 요소별로 데이터를 제공한다. 시간 설정을 통해 전주기별 데이터 탐색도 할 수 있도록 한다.(Filtering) 데이터 컬럼 종류 및 데이터 양이 많기 때문에 Field Selection, Slicing 또는 Paging 기법을 통한 효율성도 필요하다. 적절한 데이터 형식의 요청을 처리할 수 있도록 검증도 반드시 필요하다. 시간 설정의 경우 ‘ISO 8601’ 표준으로 API를 설계한다. API의 응답 형식은 키-값 쌍으로 표현 가능한 개방형 표준인 ‘JSON’ 포맷을 기본으로 사용한다. 다음은 API 예시다.

API	이름	운영 데이터 조회	
	설명	운영 데이터를 조회한다.	
	HTTP URL	/api/ess/operating-sites/{operating_site_id}/{data_type}/?fields={fields}&start-time={start_time}&end-time={end_time}&{page}	
	HTTP URL(예시)	/api/ess/operating-sites/1/banks/1/racks/1/?fields=timestamp,rack_soc&start-time=2022-09-01T10:00:00&end-time=2022-09-01T12:00:00&page=1	
	HTTP Method	GET	
	HTTP Parameter	Path	- operating_site_id: 운영 사이트 번호 - data_type: 데이터 형식 • Bank - banks/{bank_id} • Rack - banks/{bank_id}/racks/{rack_id}

	에너지 저장 장치 전주기별 데이터셋 배포 처리 모듈(SW)	
	프로젝트	대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 SW 프레임워크 개발

			<ul style="list-style-type: none"> • PCS - pcs • ETC - etc
		Query	<ul style="list-style-type: none"> - fields: 데이터 필드 선택 - start-time, end-time: 데이터 탐색 시간 - page: 페이지

o 다운로드

- ‘API Call’ 방식과 유사하지만, 응답 형식이 ‘CSV’ 포맷이고, 대용량 파일 스트리밍 방식으로 서버가 응답한다. 클라이언트가 브라우저에서 요청하면 파일을 실시간으로 다운로드하며, API Client 프로그램에서 요청하면 API Call과 비슷하게 응답한다. 파일을 스트리밍하면 서버가 응답을 생성하는 동안 웹 서버가 시간 초과로 인한 연결을 삭제하는 것을 방지한다. 다음은 API 예시다.

API	이름	운영 데이터 다운로드	
	설명	운영 데이터를 다운로드한다.	
	HTTP URL	/api/ess/download/operating-sites/{operating_site_id}/{data_type}?start-time={start_time}&end-time={end_time}	
	HTTP URL(예시)	/api/ess/download/operating-sites/1/rack/?start-time=2022-09-01T10:00:00&end-time=2022-09-01T12:00:00	
	HTTP Method	GET	
	HTTP Parameter	Path	<ul style="list-style-type: none"> - operating_site_id: 운영 사이트 번호 - data_type: 데이터 형식 <ul style="list-style-type: none"> • Bank - bank • Rack - rack • PCS - pcs • ETC - etc
		Query	<ul style="list-style-type: none"> - start-time, end-time: 데이터 탐색 시간

4. 에너지 저장 장치 전주기별 데이터셋 배포 처리 SW 개발

□ 개요

- SW 아키텍처 중 웹 서비스 SW 구축과 배포 처리 모듈의 개발에 대한 내용을 정의한다.

□ 웹 서비스 SW 구축

- 웹 프레임워크를 활용한 웹 서비스 개발

- 최소 요구 환경

- Ubuntu Linux 20.04LTS
- python3 (>=3.8)
- python3-venv
- python3-dev
- libpq-dev
- Docker

- 위의 최소 요구 환경을 만족한 후 다음 코드를 차례대로 실행하면 웹 서비스를 실행할 수 있다. 코드의 내용은 Python 가상 환경을 생성하고, 가상 환경 안에서 ‘requirements.txt’에 정의한 패키지를 설치한다. 이후 Django 프로젝트를 생성한다.

```
# ← 주석으로 정의
# 터미널 명령창이 빈 폴더에 위치해 있다고 가정
# 의존성 패키지 설치
$ python3 -m venv .venv
$ source .venv/bin/activate
(.venv) $ pip install -r requirements.txt
(.venv) $ django-admin startproject config .
```

requirements.txt - Django 패키지 의존성 관리 파일

```
Django==3.2.8
django-filter==21.1
django-rest-framework==3.12.4
drf-flex-fields==0.9.6
gunicorn==20.1.0
psycpg2-binary==2.9.3
python-dotenv==0.19.1
```

o 배포 처리 모듈 개발

- 데이터베이스 연결 및 Django Model 개발

- Django 설정 파일에 내용을 정의하고, 다음 코드를 실행한다. 코드의 내용은 ‘ess’ Django App을 생성하고, 연결된 데이터베이스에 이미 존재하는 테이블이 있는 경우 테이블 정의 내용을 Django Model 객체로 자동 변경하여 터미널에 출력한다. 이 출력된 내용을 ‘ess’ Django App의 ‘models.py’ 파일에 추가한 후 변경 사항을 웹 서비스에 적용한다.

```
(.venv) $ python manage.py startapp ess
(.venv) $ python manage.py inspectdb --database=ess1
(.venv) $ python manage.py inspectdb --database=ess2
# ‘models.py’ 내용 적용 이후 아래 코드 실행
(.venv) $ python manage.py makemigrations ess
```

settings.py - Django 설정 파일(일부 내용 생략)

```
import os
from pathlib import Path
from dotenv import load_dotenv

load_dotenv()

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.getenv("SECRET_KEY")

# Application definition

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "django_filters",
```

```

“rest_framework“,
“ess“,
]

MIDDLEWARE = [
    “django.middleware.security.SecurityMiddleware“,
    “django.contrib.sessions.middleware.SessionMiddleware“,
    “django.middleware.common.CommonMiddleware“,
    “django.middleware.csrf.CsrfViewMiddleware“,
    “django.contrib.auth.middleware.AuthenticationMiddleware“,
    “django.contrib.messages.middleware.MessageMiddleware“,
    “django.middleware.clickjacking.XFrameOptionsMiddleware“,
]

ROOT_URLCONF = “config.urls“

TEMPLATES = [
    {
        “BACKEND“: “django.template.backends.django.DjangoTemplates“,
        “DIRS“: [
            BASE_DIR / “templates“,
        ],
        “APP_DIRS“: True,
        “OPTIONS“: {
            “context_processors“: [
                “django.template.context_processors.debug“,
                “django.template.context_processors.request“,
                “django.contrib.auth.context_processors.auth“,
                “django.contrib.messages.context_processors.messages“,
            ],
        },
    },
]

WSGI_APPLICATION = “config.wsgi.application“

# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    “default“: {
        “ENGINE“: os.getenv(“DEFAULT_DB_ENGINE“),
    }
}

```

```

“HOST”: os.getenv(“DEFAULT_DB_HOST”),
“PORT”: os.getenv(“DEFAULT_DB_PORT”),
“NAME”: os.getenv(“DEFAULT_DB_NAME”),
“USER”: os.getenv(“DEFAULT_DB_USER”),
“PASSWORD”: os.getenv(“DEFAULT_DB_PASSWORD”),
“TEST”: {
    “DEPENDENCIES”: [“ess“],
    “NAME”: os.getenv(“DEFAULT_TEST_DB_NAME”),
},
},
“ess1”: {
    “ENGINE”: os.getenv(“ESS_DB_ENGINE”),
    “HOST”: os.getenv(“ESS_DB_HOST”),
    “PORT”: os.getenv(“ESS_DB_PORT”),
    “NAME”: os.getenv(“ESS_DB_NAME”),
    “USER”: os.getenv(“ESS_DB_USER”),
    “PASSWORD”: os.getenv(“ESS_DB_PASSWORD”),
    “TEST”: {
        “DEPENDENCIES”: [],
        “NAME”: os.getenv(“ESS_TEST_DB_NAME”),
    },
},
},
“ess2”: {
    “ENGINE”: os.getenv(“ESS2_DB_ENGINE”),
    “HOST”: os.getenv(“ESS2_DB_HOST”),
    “PORT”: os.getenv(“ESS2_DB_PORT”),
    “NAME”: os.getenv(“ESS2_DB_NAME”),
    “USER”: os.getenv(“ESS2_DB_USER”),
    “PASSWORD”: os.getenv(“ESS2_DB_PASSWORD”),
    “TEST”: {
        “DEPENDENCIES”: [],
        “NAME”: os.getenv(“ESS2_TEST_DB_NAME”),
    },
},
},
}

DATABASE_ROUTERS = [
    “ess.routers.ESSRouter“,
]

# Password validation
# https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-validators

```

```

AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME": "django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.CommonPasswordValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.NumericPasswordValidator",
    },
]

# Internationalization
# https://docs.djangoproject.com/en/3.2/topics/i18n/

LANGUAGE_CODE = "en-us"

TIME_ZONE = "Asia/Seoul"

USE_I18N = True

USE_L10N = True

USE_TZ = False

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = "/static/"

# Default primary key field type
# https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

REST_FRAMEWORK = {
    "DEFAULT_PERMISSION_CLASSES": ("rest_framework.permissions.IsAuthenticated",),

```

```

“ D E F A U L T _ A U T H E N T I C A T I O N _ C L A S S E S “ :
(“config.authenticate.CustomJWTTokenUserAuthentication“,),
    “DEFAULT_PAGINATION_CLASS“: “rest_framework.pagination.PageNumberPagination“,
    “PAGE_SIZE“: 100,
}

REST_SESSION_LOGIN = False

SIMPLE_JWT = {
    “SIGNING_KEY“: os.getenv(“JWT_SECRET_KEY“),
    “VERIFYING_KEY“: os.getenv(“JWT_SECRET_KEY“),
}

```

ess/models.py - Django Model 파일(일부 내용 생략)

```

from django.db import models

# Common ESS Object Info(But, excluding the Pcs because the Pcs columns are very different)

class CommonBankInfo(models.Model):
    timestamp = models.DateTimeField(db_column="TIMESTAMP", primary_key=True)
    bank_id = models.IntegerField(db_column="BANK_ID")
    bank_soc = models.FloatField(db_column="BANK_SOC", blank=True, null=True)
    bank_soh = models.FloatField(db_column="BANK_SOH", blank=True, null=True)
    bank_dc_volt = models.FloatField(db_column="BANK_DC_VOLT", blank=True, null=True)
    bank_dc_current = models.FloatField(db_column="BANK_DC_CURRENT", blank=True, null=True)
    max_cell_voltage_of_bank = models.FloatField(db_column="MAX_CELL_VOLTAGE_OF_BANK",
blank=True, null=True)
    min_cell_voltage_of_bank = models.FloatField(db_column="MIN_CELL_VOLTAGE_OF_BANK",
blank=True, null=True)
    max_cell_temperature_of_bank =
models.FloatField(db_column="MAX_CELL_TEMPERATURE_OF_BANK", blank=True, null=True)
    min_cell_temperature_of_bank =
models.FloatField(db_column="MIN_CELL_TEMPERATURE_OF_BANK", blank=True, null=True)
    ...

    class Meta:
        abstract = True
        managed = False

class CommonRackInfo(models.Model):

```

```

timestamp = models.DateTimeField(db_column="TIMESTAMP", primary_key=True)
bank_id = models.IntegerField(db_column="BANK_ID")
rack_id = models.IntegerField(db_column="RACK_ID")
rack_soc = models.FloatField(db_column="RACK_SOC", blank=True, null=True)
rack_soh = models.FloatField(db_column="RACK_SOH", blank=True, null=True)
rack_voltage = models.FloatField(db_column="RACK_VOLTAGE", blank=True, null=True)
rack_current = models.FloatField(db_column="RACK_CURRENT", blank=True, null=True)
rack_max_cell_voltage = models.FloatField(db_column="RACK_MAX_CELL_VOLTAGE",
blank=True, null=True)
rack_max_cell_voltage_position = models.FloatField(
    db_column="RACK_MAX_CELL_VOLTAGE_POSITION", blank=True, null=True
)
rack_min_cell_voltage = models.FloatField(db_column="RACK_MIN_CELL_VOLTAGE", blank=True,
null=True)
rack_min_cell_voltage_position = models.FloatField(
    db_column="RACK_MIN_CELL_VOLTAGE_POSITION", blank=True, null=True
)
...

class Meta:
    abstract = True
    managed = False

class CommonEtcInfo(models.Model):
    timestamp = models.DateTimeField(db_column="TIMESTAMP", primary_key=True)
    sensor1_temperature = models.FloatField(db_column="SENSOR1_TEMPERATURE", blank=True,
null=True)
    sensor1_humidity = models.FloatField(db_column="SENSOR1_HUMIDITY", blank=True, null=True)

    class Meta:
        abstract = True
        managed = False

# ESS Object Info

class Bank(models.Model):
    timestamp = models.DateTimeField(db_column="TIMESTAMP", primary_key=True)
    bank_id = models.IntegerField(db_column="BANK_ID")
    bank_soc = models.FloatField(db_column="BANK_SOC", blank=True, null=True)
    bank_soh = models.FloatField(db_column="BANK_SOH", blank=True, null=True)

```



```

bank_dc_volt = models.FloatField(db_column="BANK_DC_VOLT", blank=True, null=True)
bank_dc_current = models.FloatField(db_column="BANK_DC_CURRENT", blank=True, null=True)
max_cell_voltage_of_bank = models.FloatField(db_column="MAX_CELL_VOLTAGE_OF_BANK",
blank=True, null=True)
min_cell_voltage_of_bank = models.FloatField(db_column="MIN_CELL_VOLTAGE_OF_BANK",
blank=True, null=True)
max_cell_temperature_of_bank =
models.FloatField(db_column="MAX_CELL_TEMPERATURE_OF_BANK", blank=True, null=True)
min_cell_temperature_of_bank =
models.FloatField(db_column="MIN_CELL_TEMPERATURE_OF_BANK", blank=True, null=True)
...

class Meta:
    managed = False
    db_table = "bank"

class ESS2Bank(CommonBankInfo):
    max_module_temperature = models.FloatField(db_column="MAX_MODULE_TEMPERATURE",
blank=True, null=True)
    min_module_temperature = models.FloatField(db_column="MIN_MODULE_TEMPERATURE",
blank=True, null=True)
    max_module_humidity = models.FloatField(db_column="MAX_MODULE_HUMIDITY", blank=True,
null=True)
    min_module_humidity = models.FloatField(db_column="MIN_MODULE_HUMIDITY", blank=True,
null=True)
    rack_tray_voltage_imbalance_warning = models.IntegerField(
        db_column="RACK_TRAY_VOLTAGE_IMBALANCE_WARNING", blank=True, null=True
    )
    master_rack_communication_fault = models.TextField(
        db_column="MASTER_RACK_COMMUNICATION_FAULT", blank=True, default=""
    )
    battery_status_for_standby = models.IntegerField(db_column="BATTERY_STATUS_FOR_STANDBY",
blank=True, null=True)
    battery_status_for_discharge =
models.IntegerField(db_column="BATTERY_STATUS_FOR_DISCHARGE", blank=True, null=True)
    rack_to_master_bms_communication_status = models.IntegerField(
        db_column="RACK_TO_MASTER_BMS_COMMUNICATION_STATUS", blank=True, null=True
    )
    charging_stop_of_status = models.IntegerField(db_column="CHARGING_STOP_OF_STATUS",
blank=True, null=True)
    ...

```

```

class Meta(CommonBankInfo.Meta):
    db_table = "bank"

class Rack(models.Model):
    timestamp = models.DateTimeField(db_column="TIMESTAMP", primary_key=True)
    bank_id = models.IntegerField(db_column="BANK_ID")
    rack_id = models.IntegerField(db_column="RACK_ID")
    rack_soc = models.FloatField(db_column="RACK_SOC", blank=True, null=True)
    rack_soh = models.FloatField(db_column="RACK_SOH", blank=True, null=True)
    rack_voltage = models.FloatField(db_column="RACK_VOLTAGE", blank=True, null=True)
    rack_current = models.FloatField(db_column="RACK_CURRENT", blank=True, null=True)
    rack_max_cell_voltage = models.FloatField(db_column="RACK_MAX_CELL_VOLTAGE",
blank=True, null=True)
    rack_max_cell_voltage_position = models.FloatField(
        db_column="RACK_MAX_CELL_VOLTAGE_POSITION", blank=True, null=True
    )
    rack_min_cell_voltage = models.FloatField(db_column="RACK_MIN_CELL_VOLTAGE", blank=True,
null=True)
    ...

    class Meta:
        managed = False
        db_table = "rack"

class ESS2Rack(CommonRackInfo):
    rack_max_module_temperature =
models.FloatField(db_column="RACK_MAX_MODULE_TEMPERATURE", blank=True, null=True)
    rack_max_module_temperature_position = models.FloatField(
        db_column="RACK_MAX_MODULE_TEMPERATURE_POSITION", blank=True, null=True
    )
    rack_min_module_temperature =
models.FloatField(db_column="RACK_MIN_MODULE_TEMPERATURE", blank=True, null=True)
    rack_min_module_temperature_position = models.FloatField(
        db_column="RACK_MIN_MODULE_TEMPERATURE_POSITION", blank=True, null=True
    )
    rack_max_module_humidity = models.FloatField(db_column="RACK_MAX_MODULE_HUMIDITY",
blank=True, null=True)
    rack_max_module_humidity_position = models.FloatField(
        db_column="RACK_MAX_MODULE_HUMIDITY_POSITION", blank=True, null=True
    )
    rack_min_module_humidity = models.FloatField(db_column="RACK_MIN_MODULE_HUMIDITY",

```

```

blank=True, null=True)
    rack_min_module_humidity_position = models.FloatField(
        db_column="RACK_MIN_MODULE_HUMIDITY_POSITION", blank=True, null=True
    )
    rack_status_for_online = models.IntegerField(db_column="RACK_STATUS_FOR_Online",
blank=True, null=True)
    rack_tray_volatge_imbalance_warning = models.IntegerField(
        db_column="RACK_TRAY_VOLATGE_IMBALANCE_WARNING", blank=True, null=True
    )
    ...

class Meta(CommonRackInfo.Meta):
    db_table = "rack"

class Pcs(models.Model):
    timestamp = models.DateTimeField(db_column="TIMESTAMP", primary_key=True)
    bank_id = models.IntegerField(db_column="BANK_ID")
    pcs_ac_l1_thd = models.FloatField(db_column="PCS_AC_L1_THD", blank=True, null=True)
    pcs_ac_l2_thd = models.FloatField(db_column="PCS_AC_L2_THD", blank=True, null=True)
    pcs_ac_l3_thd = models.FloatField(db_column="PCS_AC_L3_THD", blank=True, null=True)
    system_temperature_max = models.FloatField(db_column="SYSTEM_TEMPERATURE_MAX",
blank=True, null=True)
    dc_voltage_1 = models.FloatField(db_column="DC_VOLTAGE_1", blank=True, null=True)
    dc_voltage_2 = models.FloatField(db_column="DC_VOLTAGE_2", blank=True, null=True)
    dc_current = models.FloatField(db_column="DC_CURRENT", blank=True, null=True)
    dc_power = models.FloatField(db_column="DC_POWER", blank=True, null=True)
    ac_frequency = models.FloatField(db_column="AC_FREQUENCY", blank=True, null=True)
    ...

class Meta:
    managed = False
    db_table = "pcs"

class ESS2Pcs(models.Model):
    timestamp = models.DateTimeField(db_column="TIMESTAMP", primary_key=True)
    ai_vdc = models.FloatField(db_column="AI_VDC", blank=True, null=True)
    ai_idc = models.FloatField(db_column="AI_IDC", blank=True, null=True)
    ai_pdc = models.FloatField(db_column="AI_PDC", blank=True, null=True)
    ai_freq = models.FloatField(db_column="AI_FREQ", blank=True, null=True)
    ai_vab_rms = models.FloatField(db_column="AI_VAB_RMS", blank=True, null=True)
    ai_vbc_rms = models.FloatField(db_column="AI_VBC_RMS", blank=True, null=True)

```

```
ai_vca_rms = models.FloatField(db_column="AI_VCA_RMS", blank=True, null=True)
ai_ias_rms = models.FloatField(db_column="AI_IAS_RMS", blank=True, null=True)
ai_ibs_rms = models.FloatField(db_column="AI_IBS_RMS", blank=True, null=True)
...

class Meta:
    managed = False
    db_table = "etc"

class ESS2Etc(CommonEtcInfo):
    sensor2_temperature = models.FloatField(db_column="SENSOR2_TEMPERATURE", blank=True,
null=True)
    sensor2_humidity = models.FloatField(db_column="SENSOR2_HUMIDITY", blank=True, null=True)
    active_power_total = models.FloatField(db_column="ACTIVE_POWER_TOTAL", blank=True,
null=True)
    active_energy_total_high = models.FloatField(db_column="ACTIVE_ENERGY_TOTAL_HIGH",
blank=True, null=True)

    class Meta(CommonEtcInfo.Meta):
        db_table = "etc"
```

- Serializer(직렬 변환기) 개발

- Django Model 객체 내용을 ‘JSON’, ‘CSV’ 와 같은 다른 콘텐츠 유형으로 렌더링하기 위해 Serializer 개발이 필요하다. Django Rest Framework Serializer 파일에 다음 내용을 정의한다.

ess/serializers.py - Django Rest Framework Serializer 파일(일부 내용 생략)

```
from rest_flex_fields import FlexFieldsModelSerializer
from rest_framework import serializers
from .models import Bank, Rack, Pcs, Etc, ESS2Bank, ESS2Rack, ESS2Pcs, ESS2Etc

class BankSerializer(FlexFieldsModelSerializer):
    class Meta:
        model = Bank
        fields = "__all__"

class RackSerializer(FlexFieldsModelSerializer):
    class Meta:
        model = Rack
```

```
fields = "__all__"
```

```
class PcsSerializer(FlexFieldsModelSerializer):
```

```
    class Meta:
```

```
        model = Pcs
```

```
        fields = "__all__"
```

```
class EtcSerializer(FlexFieldsModelSerializer):
```

```
    class Meta:
```

```
        model = Etc
```

```
        fields = "__all__"
```

```
class ESS2BankSerializer(FlexFieldsModelSerializer):
```

```
    # If model type is JSONField, not working - TypeError
```

```
    # so model type is TextField -> serializer JSONField working!!
```

```
    master_rack_communication_fault = serializers.JSONField()
```

```
    class Meta:
```

```
        model = ESS2Bank
```

```
        fields = "__all__"
```

```
class ESS2RackSerializer(FlexFieldsModelSerializer):
```

```
    rack_module_fault = serializers.JSONField()
```

```
    class Meta:
```

```
        model = ESS2Rack
```

```
        fields = "__all__"
```

```
class ESS2PcsSerializer(FlexFieldsModelSerializer):
```

```
    class Meta:
```

```
        model = ESS2Pcs
```

```
        fields = "__all__"
```

```
class ESS2EtcSerializer(FlexFieldsModelSerializer):
```

```
    class Meta:
```

```
        model = ESS2Etc
```

```
        fields = "__all__"
```

- View 개발

- View는 클라이언트 요청을 처리하는 컨트롤러 역할을 한다. 주로 요청을 검증한 후 데이터베이스에 질의한 후 결과를 Django Model 객체에 담는다. 이후 Model 객체를 직렬화(Serialize)하고, 응답 처리한다. 에러 사항이 있으면 클라이언트에게 에러 코드와 메시지를 전달할 수 있도록 예외 처리한다. Django View 파일에 다음 내용을 정의한다.

ess/views.py - Django View 파일(일부 내용 생략)

```
import csv
from datetime import datetime
from datetime import timedelta
from django.db import connections, DataError
from django.http import StreamingHttpResponse
from rest_framework import status
from rest_framework.views import Response
from rest_framework.generics import ListAPIView, RetrieveAPIView
from .paginations import LargeResultsSetPagination
# This custom module have dynamic ess models, serializers, data_dates
from .ess_collections import (
    ESS_BANK,
    ESS_RACK,
    ESS_PCS,
    ESS_ETC,
    ESS_BANK_SERIALIZER,
    ESS_RACK_SERIALIZER,
    ESS_PCS_SERIALIZER,
    ESS_ETC_SERIALIZER,
)

TIME_BUCKET_TIMEZONE = "Asia/Seoul"

class Echo:
    """An object that implements just the write method of the file-like interface."""

    def write(self, value):
        """Write the value by returning it, instead of storing in a buffer."""
        return value

def dictfetchall(cursor):
    """Return all rows from a cursor as a dict"""
```

```

columns = [col[0] for col in cursor.description]
return [dict(zip(columns, row)) for row in cursor.fetchall()]

def get_csv_items(rows, pseudo_buffer, fieldnames):
    writer = csv.DictWriter(pseudo_buffer, fieldnames=fieldnames)
    yield writer.writeheader()

    for row in rows:
        yield writer.writerow(row)

# General ESS model list view

class ESSBankListView(ListAPIView):
    pagination_class = LargeResultsSetPagination

    def get_serializer_class(self):
        operating_site_id = self.kwargs["operating_site_id"]
        database = "ess" + str(operating_site_id)

        return ESS_BANK_SERIALIZER[database]

    def get_queryset(self):
        operating_site_id = self.kwargs["operating_site_id"]
        database = "ess" + str(operating_site_id)
        bank_id = self.kwargs["bank_id"]
        start_date = self.request.query_params.get("date")

        if start_date is not None:
            end_date = datetime.strptime(start_date, "%Y-%m-%d") + timedelta(days=1)
            queryset = (
                ESS_BANK[database]
                .objects.using(database)
                .filter(bank_id=bank_id, timestamp__gte=start_date, timestamp__lt=end_date)
            ).order_by("timestamp")
        else:
            queryset =
ESS_BANK[database].objects.using(database).filter(bank_id=bank_id).order_by("timestamp")

        return queryset

```

```

def get(self, request, *args, **kwargs):
    try:
        queryset = self.get_queryset()
        page = self.paginate_queryset(queryset)

        if page is not None:
            serializer = self.get_serializer(page, many=True)

            return self.get_paginated_response(serializer.data)

        serializer = self.get_serializer(queryset, many=True)

        return Response(serializer.data)
    except KeyError:
        return Response(
            {"code": "400", "exception type": "Key Error", "message": "올바른 요청 URL을 입력하세요.(operating_site_id)"},
            status=status.HTTP_400_BAD_REQUEST,
        )
    except ValueError:
        return Response(
            {"code": "400", "exception type": "Value Error", "message": "올바른 요청 파라미터를 입력하세요.(date)"},
            status=status.HTTP_400_BAD_REQUEST,
        )

class ESSRackListView(ListAPIView):
    pagination_class = LargeResultsSetPagination

    def get_serializer_class(self):
        operating_site_id = self.kwargs["operating_site_id"]
        database = "ess" + str(operating_site_id)

        return ESS_RACK_SERIALIZER[database]

    def get_queryset(self):
        operating_site_id = self.kwargs["operating_site_id"]
        database = "ess" + str(operating_site_id)
        bank_id = self.kwargs["bank_id"]
        start_date = self.request.query_params.get("date")

        if start_date is not None:

```



```

end_date = datetime.strptime(start_date, "%Y-%m-%d") + timedelta(days=1)
queryset = (
    ESS_RACK[database]
    .objects.using(database)
    .filter(bank_id=bank_id, timestamp__gte=start_date, timestamp__lt=end_date)
    .order_by("timestamp")
)
else:
    queryset =
ESS_RACK[database].objects.using(database).filter(bank_id=bank_id).order_by("timestamp")

return queryset

def get(self, request, *args, **kwargs):
    try:
        queryset = self.get_queryset()
        page = self.paginate_queryset(queryset)

        if page is not None:
            serializer = self.get_serializer(page, many=True)

            return self.get_paginated_response(serializer.data)

        serializer = self.get_serializer(queryset, many=True)

        return Response(serializer.data)
    except KeyError:
        return Response(
            {"code": "400", "exception type": "Key Error", "message": "올바른 요청 URL을 입력하세요.(operating_site_id)"},
            status=status.HTTP_400_BAD_REQUEST,
        )
    except ValueError:
        return Response(
            {"code": "400", "exception type": "Value Error", "message": "올바른 요청 파라미터를 입력하세요.(date)"},
            status=status.HTTP_400_BAD_REQUEST,
        )

class ESSPcsListView(ListAPIView):
    pagination_class = LargeResultsSetPagination

```

```

def get_serializer_class(self):
    operating_site_id = self.kwargs["operating_site_id"]
    database = "ess" + str(operating_site_id)

    return ESS_PCS_SERIALIZER[database]

def get_queryset(self):
    operating_site_id = self.kwargs["operating_site_id"]
    database = "ess" + str(operating_site_id)
    start_date = self.request.query_params.get("date")

    if start_date is not None:
        end_date = datetime.strptime(start_date, "%Y-%m-%d") + timedelta(days=1)
        queryset = (
            ESS_PCS[database]
            .objects.using(database)
            .filter(timestamp__gte=start_date, timestamp__lt=end_date)
            .order_by("timestamp")
        )
    else:
        queryset = ESS_PCS[database].objects.using(database).all().order_by("timestamp")

    return queryset

def get(self, request, *args, **kwargs):
    try:
        queryset = self.get_queryset()
        page = self.paginate_queryset(queryset)

        if page is not None:
            serializer = self.get_serializer(page, many=True)

            return self.get_paginated_response(serializer.data)

        serializer = self.get_serializer(queryset, many=True)

        return Response(serializer.data)
    except KeyError:
        return Response(
            {"code": "400", "exception type": "Key Error", "message": "올바른 요청 URL을 입력하세요.(operating_site_id)"},
            status=status.HTTP_400_BAD_REQUEST,
        )

```

```

except ValueError:
    return Response(
        {"code": "400", "exception type": "Value Error", "message": "올바른 요청 파라미
터 입력하세요.(date)"},
        status=status.HTTP_400_BAD_REQUEST,
    )

class ESSEtcListView(ListAPIView):
    pagination_class = LargeResultsSetPagination

    def get_serializer_class(self):
        operating_site_id = self.kwargs["operating_site_id"]
        database = "ess" + str(operating_site_id)

        return ESS_ETC_SERIALIZER[database]

    def get_queryset(self):
        operating_site_id = self.kwargs["operating_site_id"]
        database = "ess" + str(operating_site_id)
        start_date = self.request.query_params.get("date")

        if start_date is not None:
            end_date = datetime.strptime(start_date, "%Y-%m-%d") + timedelta(days=1)
            queryset = (
                ESS_ETC[database]
                .objects.using(database)
                .filter(timestamp__gte=start_date, timestamp__lt=end_date)
                .order_by("timestamp")
            )
        else:
            queryset = ESS_ETC[database].objects.using(database).all().order_by("timestamp")

        return queryset

    def get(self, request, *args, **kwargs):
        try:
            queryset = self.get_queryset()
            page = self.paginate_queryset(queryset)

            if page is not None:
                serializer = self.get_serializer(page, many=True)

```

```

        return self.get_paginated_response(serializer.data)

    serializer = self.get_serializer(queryset, many=True)

    return Response(serializer.data)
except KeyError:
    return Response(
        {"code": "400", "exception type": "Key Error", "message": "올바른 요청 URL을 입  
력하세요.(operating_site_id)",
        status=status.HTTP_400_BAD_REQUEST,
    )
except ValueError:
    return Response(
        {"code": "400", "exception type": "Value Error", "message": "올바른 요청 파라미  
터를 입력하세요.(date)",
        status=status.HTTP_400_BAD_REQUEST,
    )

# General ESS model detail list view

class ESSRackDetailView(ListAPIView):
    pagination_class = LargeResultsSetPagination

    def get_serializer_class(self):
        operating_site_id = self.kwargs["operating_site_id"]
        database = "ess" + str(operating_site_id)

        return ESS_RACK_SERIALIZER[database]

    def get_queryset(self):
        operating_site_id = self.kwargs["operating_site_id"]
        database = "ess" + str(operating_site_id)
        bank_id = self.kwargs["bank_id"]
        rack_id = self.kwargs["rack_id"]
        start_date = self.request.query_params.get("date")

        if start_date is not None:
            end_date = datetime.strptime(start_date, "%Y-%m-%d") + timedelta(days=1)
            queryset = (
                ESS_RACK[database]
                .objects.using(database)

```

```

        .filter(bank_id=bank_id, rack_id=rack_id, timestamp__gte=start_date,
timestamp__lt=end_date)
        .order_by("timestamp")
    )
else:
    queryset = (
        ESS_RACK[database]
        .objects.using(database)
        .filter(bank_id=bank_id, rack_id=rack_id)
        .order_by("timestamp")
    )

    return queryset

def get(self, request, *args, **kwargs):
    try:
        queryset = self.get_queryset()
        page = self.paginate_queryset(queryset)

        if page is not None:
            serializer = self.get_serializer(page, many=True)

            return self.get_paginated_response(serializer.data)

        serializer = self.get_serializer(queryset, many=True)

        return Response(serializer.data)
    except KeyError:
        return Response(
            {"code": "400", "exception type": "Key Error", "message": "올바른 요청 URL을 입
력하세요.(operating_site_id)"},
            status=status.HTTP_400_BAD_REQUEST,
        )
    except ValueError:
        return Response(
            {"code": "400", "exception type": "Value Error", "message": "올바른 요청 파라미
터를 입력하세요.(date)"},
            status=status.HTTP_400_BAD_REQUEST,
        )

# ESS operating data(ESS model) download view

```

```

class ESSOperatingDataDownloadView(RetrieveAPIView):
    def get(self, request, *args, **kwargs):
        try:
            operating_site_id = kwargs["operating_site_id"]
            database = "ess" + str(operating_site_id)
            start_time_query_param = request.query_params.get("start-time")
            start_time = datetime.strptime(start_time_query_param, "%Y-%m-%dT%H:%M:%S")
            end_time_query_param = request.query_params.get("end-time")
            end_time = datetime.strptime(end_time_query_param, "%Y-%m-%dT%H:%M:%S")
            data_types = {
                "bank": ESS_BANK[database],
                "rack": ESS_RACK[database],
                "pcs": ESS_PCS[database],
                "etc": ESS_ETC[database],
            }
            data_type = kwargs["data_type"]
            queryset = (
                data_types[data_type]
                .objects.using(database)
                .filter(timestamp__gte=start_time, timestamp__lte=end_time)
                .order_by("timestamp")
            )
            fieldnames = list(queryset.values()[0].keys())

            return StreamingHttpResponse(
                (get_csv_items(queryset.values(), Echo(), fieldnames)),
                content_type="text/csv",
                headers={"Content-Disposition": "attachment; filename='" + data_type + ".csv'"},
            )
        except KeyError:
            return Response(
                {"code": "400", "exception type": "Key Error", "message": "올바른 요청 URL을 입력하세요.(data_type)"},
                status=status.HTTP_400_BAD_REQUEST,
            )
        except ValueError:
            return Response(
                {
                    "code": "400",
                    "exception type": "Value Error",
                    "message": "올바른 요청 파라미터를 입력하세요.(time 형식은 'YYYY-MM-DDThh:mm:ss' 입니다.)",
                }
            )

```

```

    },
    status=status.HTTP_400_BAD_REQUEST,
)
except TypeError:
    return Response(
        {"code": "400", "exception type": "Type Error", "message": "필수 요청 파라미터를  
입력하세요.(start-time, end-time)"},
        status=status.HTTP_400_BAD_REQUEST,
    )
except IndexError:
    return Response(
        {"code": "404", "exception type": "Index Error", "message": "해당 데이터를 찾을  
수 없습니다."},
        status=status.HTTP_404_NOT_FOUND,
    )

```

- URL Dispatcher 개발

- URL Dispatcher는 URL 구성을 디자인하고, 클라이언트 요청에서 URL 해석을 통해 관련 'View'에 요청을 전달하는 역할을 한다. Django URL Dispatcher 파일에 다음 내용을 정의한다.

ess/urls.py - Django URL Dispatcher 파일(일부 내용 생략)

```

from django.urls import include, path
from rest_framework.routers import DefaultRouter
from .views import (
    ESSBankListView,
    ESSRackListView,
    ESSPcsListView,
    ESSEtcListView,
    ESSRackDetailView,
    ESSOperatingDataDownloadView,
)

urlpatterns = [
    # General ESS model list url
    path("operating-sites/<int:operating_site_id>/banks/<int:bank_id>/", ESSBankListView.as_view(),
    name="bank-list"),
    path(
        "operating-sites/<int:operating_site_id>/banks/<int:bank_id>/racks/",
        ESSRackListView.as_view(),
        name="rack-list",
    ),

```

```
path("operating-sites/<int:operating_site_id>/pcs/", ESSPcsListView.as_view(), name="pcs-list"),
path("operating-sites/<int:operating_site_id>/etc/", ESSEtcListView.as_view(), name="etc-list"),
# General ESS model detail list url
path(
    "operating-sites/<int:operating_site_id>/banks/<int:bank_id>/racks/<rack_id>/",
    ESSRackDetailView.as_view(),
    name="rack-detail-list",
),
path("download/operating-sites/<int:operating_site_id>/<data_type>/",
ESSOperatingDataDownloadView.as_view()),
]
```

o 웹 서비스 배포

- Docker를 활용해 웹 서비스를 배포한다. Docker는 애플리케이션의 실행을 위해 운영 체제에서 격리된 컨테이너 가상 환경을 제공한다. Docker 컨테이너 안에 웹 서비스를 구축한다. Django 웹 서버는 단독으로 실행할 수 있지만 성능이 좋지 않아 운영 환경에서는 ‘Gunicorn’ 과 같은 Python WSGI(Web Server Gateway Interface) HTTP Server 웹 서버와 함께 결합하여 실행한다. Docker 이미지 생성 파일에 내용을 정의한다.

Dockerfile - Docker 이미지 생성 파일

```
FROM python:3.8

# This prevents Python from writing out pyc files
ENV PYTHONDONTWRITEBYTECODE 1

# This keeps Python from buffering stdin/stdout
ENV PYTHONUNBUFFERED 1

WORKDIR /app

COPY requirements.txt /app/

RUN pip install -r requirements.txt

COPY . /app/

CMD ["gunicorn", "--workers=3", "--timeout=300", "--bind", ":8002", "config.wsgi"]
```

□ Test 및 결과

o Test

- Test는 브라우저에서 위의 API 예시 URL 입력을 통해 결과를 확인한다.

o 결과

- API Call

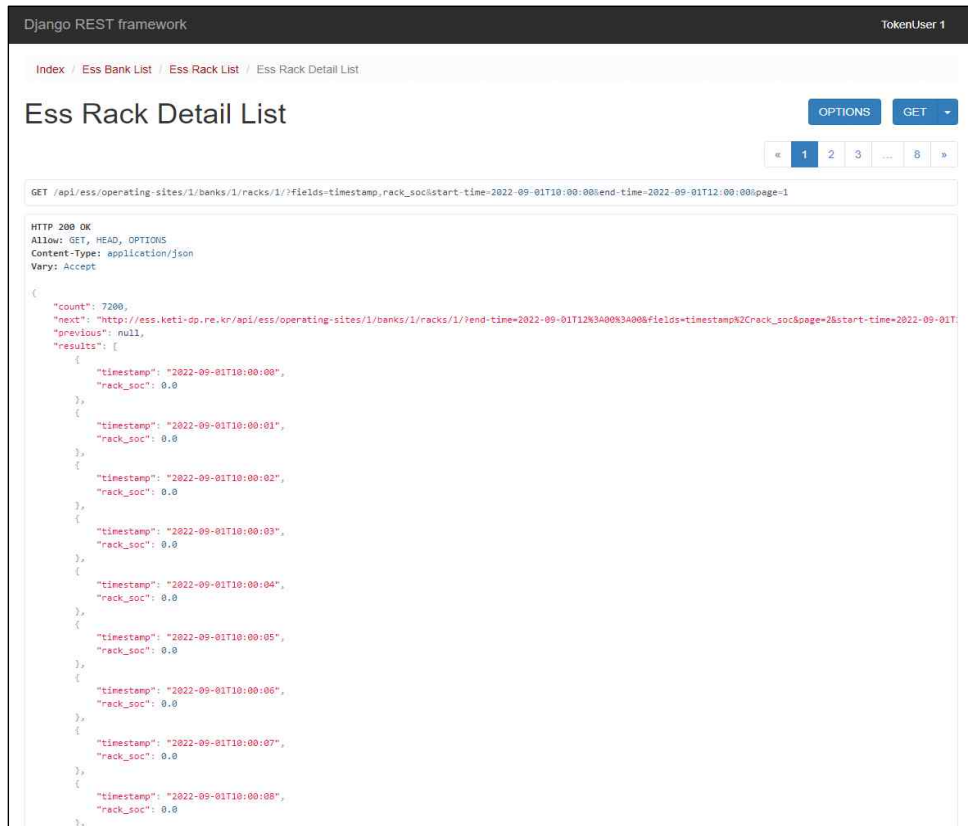
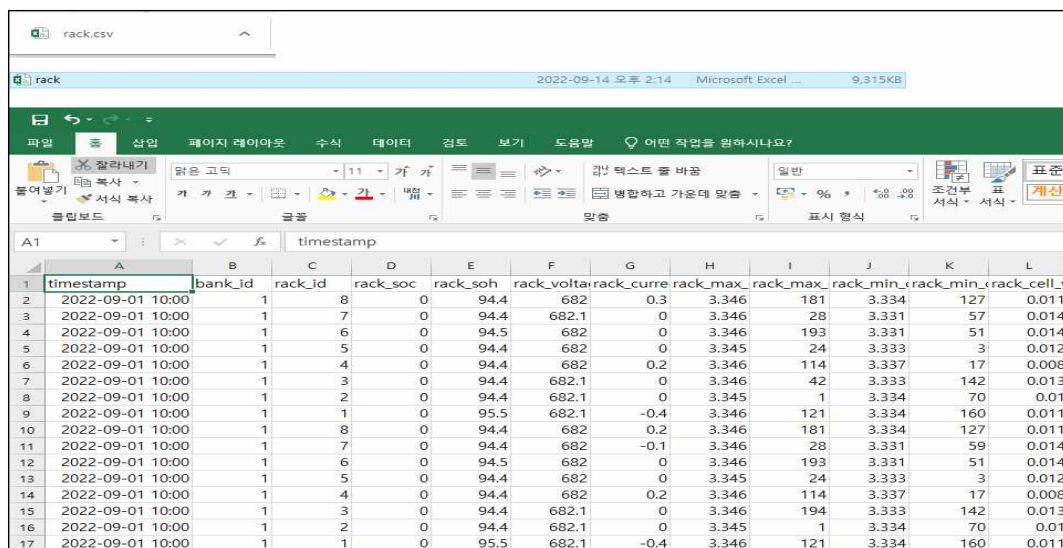


그림 6 브라우저에서 확인하는 API Call 결과

- 다운로드



	A	B	C	D	E	F	G	H	I	J	K	L
	timestamp	bank_id	rack_id	rack_soc	rack_soh	rack_voltage	rack_current	rack_max_voltage	rack_max_current	rack_min_voltage	rack_min_current	rack_cell_voltage
1	2022-09-01 10:00	1	8	0	94.4	682	0.3	3.346	181	3.334	127	0.011
2	2022-09-01 10:00	1	7	0	94.4	682.1	0	3.346	28	3.331	57	0.014
3	2022-09-01 10:00	1	6	0	94.5	682	0	3.346	193	3.331	51	0.014
4	2022-09-01 10:00	1	5	0	94.4	682	0	3.345	24	3.333	3	0.012
5	2022-09-01 10:00	1	4	0	94.4	682	0.2	3.346	114	3.337	17	0.008
6	2022-09-01 10:00	1	3	0	94.4	682.1	0	3.346	42	3.333	142	0.013
7	2022-09-01 10:00	1	2	0	94.4	682.1	0	3.345	1	3.334	70	0.01
8	2022-09-01 10:00	1	1	0	95.5	682.1	-0.4	3.346	121	3.334	160	0.011
9	2022-09-01 10:00	1	8	0	94.4	682	0.2	3.346	181	3.334	127	0.011
10	2022-09-01 10:00	1	7	0	94.4	682	-0.1	3.346	28	3.331	59	0.014
11	2022-09-01 10:00	1	6	0	94.5	682	0	3.346	193	3.331	51	0.014
12	2022-09-01 10:00	1	5	0	94.4	682	0	3.345	24	3.333	3	0.012
13	2022-09-01 10:00	1	4	0	94.4	682	0.2	3.346	114	3.337	17	0.008
14	2022-09-01 10:00	1	3	0	94.4	682.1	0	3.346	194	3.333	142	0.013
15	2022-09-01 10:00	1	2	0	94.4	682.1	0	3.345	1	3.334	70	0.01
16	2022-09-01 10:00	1	1	0	95.5	682.1	-0.4	3.346	121	3.334	160	0.011
17	2022-09-01 10:00	1	1	0	95.5	682.1	-0.4	3.346	121	3.334	160	0.011

그림 7 브라우저에서 다운로드 받은 결과

(위: 브라우저에서 파일 다운로드, 중간: 폴더 파일, 아래: CSV 결과)