

스크립트 프로그래밍 미니 프로젝트

REST API와 라이브러리를 이용한 데이터 시각화 웹 대시보드

학번: 2020214597
이름: 김광래

< 목차 >

1. 프로젝트 개요

- 1) 프로젝트 배경 및 목적
- 2) 요구 기능 및 특징

2. 시스템 아키텍처

- 1) 시스템 구성도와 시퀀스
- 2) 기술 스택

5. 개발 환경

1) 개발 도구 및 실험 환경

- Visual Studio Code
- Termius
- XAMPP
- Sequel Pro
- Ubuntu MATE 22.04 LTS
- Postman

2) 모듈 및 라이브러리

- 사용된 모듈의 기능 및 역할
- 사용된 외부 라이브러리의 기능 및 역할

6. 데이터베이스 및 API

- 1) 데이터베이스 개요 및 테이블 구조
- 2) API 엔드포인트 목록 및 기능

7. 데이터 시각화

- 1) 사용된 시각화 라이브러리 및 도구
- 2) 시각화된 데이터 설명

8. 시스템 모니터링

- 1) 사용된 모니터링 도구 및 라이브러리
- 2) 모니터링 항목

9. 테스트 및 배포

- 1) 테스트 시나리오 및 결과
- 2) 배포 과정 및 환경 설정

10. 성과 및 과제

11. 참고 자료

- 1) 전체 프로그램 코드 및 관련 참고 자료
- 2) 별첨 자료

1. 프로젝트 개요

1) 프로젝트 배경 및 목적

본 프로젝트는 '스크립트 프로그래밍' 교과목의 미니 프로젝트 과제 수행 및 현재 서비스 중인 DRM(Digital Rights Management) 시스템의 효율적인 관리를 위해 구상되었다.

먼저, 기존에 사용중이던 DRM은 Discord 메신저의 봇을 통해 명령어를 수신하고, 처리하는 방식이었다. 따라서 해당 서비스를 사용하기 위해 명령어를 숙지해야 했으며, 직관적인 결과를 기대하기 힘들었다. 또한 사용자 및 라이센스 관리 측면에서 아래와 같은 불편이 있었다.

- 리소스의 관리 및 접근의 어려움
- 라이센스 서버 상태 확인의 어려움
- 시스템 점검시 라이센스 접근 불가
- 사용자 현황 및 관리의 어려움

따라서 위와 같은 문제를 해결하기 위한 방안으로 웹페이지를 이용하여 서버와 라이센스, 그리고 리소스 및 고객의 현황을 쉽게 확인하고 관리할 수 있는 대시보드 시스템을 구상했다.

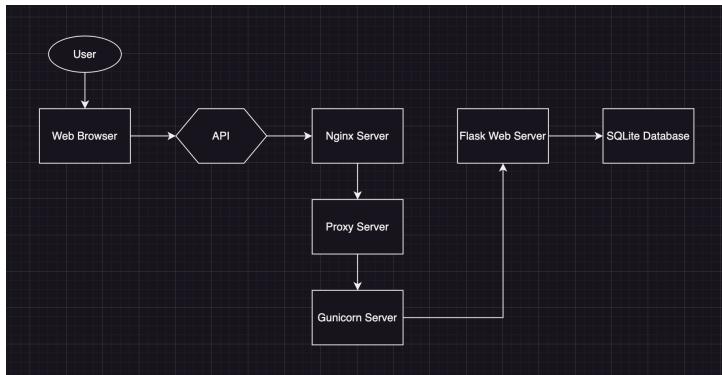
2) 요구 기능 및 특징

라이센스와 리소스, 사용자를 효과적으로 관리하기 위한 대시보드는 관련 지식이 없어도 쉽게 현황을 파악할 수 있어야 하고, 라이센스를 인증하고 파일을 제공받기 위한 서버가 정상 작동중인지 파악할 수 있어야 한다. 또한 관리자는 사용자가 API에 접근하는 시간대를 피해 서비스를 점검할 수 있도록 관련 데이터를 확인할 수 있어야 한다. 따라서 효과적인 대시보드를 위해 아래와 같이 요구되는 기능을 구성했다.

- ① 시스템 성능 및 서버 트래픽 모니터링
- ② 라이센스를 사용중인 등록 고객 수
- ③ 총 리소스 판매량
- ④ 등록된 라이센스의 수
- ⑤ 총 리소스의 수
- ⑥ 라이센스에 할당된 리소스 수
- ⑦ 더미 리소스의 수
- ⑧ 리소스별 실제 사용량 집계 그래프
- ⑨ 시간대별 라이센스 및 다운로드 서버 접근 빈도 그래프

2. 시스템 아키텍처

1) 시스템 구성도와 시퀀스



<시스템 구성도 세부 사항>

- ① **User**: 사용자는 웹 브라우저를 통해 시스템에 접근할 수 있다.
- ② **Web Browser**: 사용자가 API 요청을 보낸다.
- ③ **API**: API는 Nginx서버로 요청을 전달한다.
- ④ **Nginx Server**: Nginx서버는 요청을 프록시 서버로 전달한다.
- ⑤ **Proxy Server**: 프록시 서버는 요청을 Gunicorn 서버로 전달한다.
- ⑥ **Gunicorn Server**: Gunicorn 서버는 요청을 Flask 웹 서버로 전달한다.
- ⑦ **Flask Web Server**: Flask 웹 서버는 요청을 처리하고 SQLite 데이터베이스와 상호작용한다.
- ⑧ **SQLite Database**: 데이터베이스는 필요한 데이터를 저장하고 제공한다.

<시퀀스>

- ① **User**: 웹 브라우저를 통해 대시보드 웹페이지에 접속
- ② **Web Browser**: HTTP GET 요청을 Nginx 서버로 보냄
- ③ **Nginx Server**: 요청을 프록시 서버로 전달
- ④ **Proxy Server**: Gunicorn 서버로 요청 전달
- ⑤ **Gunicorn Server**: 요청을 Flask 웹 서버로 전달
- ⑥ **Flask Web Server**: 요청 처리, 필요한 경우 쿼리를 SQLite 데이터베이스에 보냄
- ⑦ **SQLite Database**: 쿼리 결과를 반환
- ⑧ **Flask Web Server**: 데이터를 처리하고 결과를 JSON 형식으로 변환하여 Gunicorn 서버로 반환
- ⑨ **Gunicorn Server**: 결과를 프록시 서버로 반환
- ⑩ **Proxy Server**: 결과를 Nginx 서버로 반환
- ⑪ **Nginx Server**: 웹 브라우저로 결과 반환
- ⑫ **Web Browser**: 데이터를 시각화하여 사용자에게 표시

2) 기술 스택

대시보드를 표현하기 위해 사용한 채택한 기술들은 다음과 같다.

1. Flask

기존의 불편한 과정을 간편화하고, 서비스를 관리하기에 적합한 데이터를 표현하기 위해 접근성을 중요하게 생각했다. 따라서 대시보드는 파이썬으로 빌드된 응용 프로그램이 아닌 접근이 용이한 웹 페이지를 통해 제작하는 것이 더 용이했다. Flask는 파이썬으로 작성된 마이크로 웹 프레임워크로서, 웹 어플리케이션을 빠르고 간편하게 개발할 수 있는 기술이다.

2. Bootstrap

대시보드는 접근성이 용이한 웹 페이지로 제작됨과 동시에, 여러 플랫폼에서 효율적으로 데이터를 표시하고 관리할 수 있게 해야했다. 따라서 사용하는 기기에 따라 맞춤형 UI를 표시해주어야 하는 시스템 특성상 Bootstrap을 이용한 개발이 짧은 개발 기간과 주제에 알맞았다. Bootstrap은 자주 사용되는 UI의 CSS, JavaScript 컴포넌트와 사전 정의된 클래스를 사용할 수 있어 적은 코드로 빠르게 UI를 구현할 수 있으며 주요 브라우저와 호환되도록 설계된 기술이다.

3. Gunicorn

Python WSGI(Web Server Gateway Interface) HTTP 서버인 Gunicorn은 파이썬 어플리케이션과 웹 서버 간의 인터페이스를 제공해 높은 호환성을 가지고 있다. 앞서 사용된 Flask를 프로덕션 환경에서 안정적으로 사용할 수 있으며, 설정이 간단하고 유연하다. 또한 시스템에 맞는 워커 프로세스를 사용해 병렬 처리가 가능해 더 많은 요청과 태스크를 동시에 처리할 수 있다.

4. Nginx, Reverse Proxy

Nginx는 HTTP 및 리버스 프록시 서버로, 웹 서버로부터 클라이언트 요청을 받아들이고, 이를 적절한 백엔드 서버로 전달하는 역할을 한다. 이를 통해 서버 부하 분산, 보안 강화, 정적 파일 제공 등의 기능을 수행한다. 이는 Gunicorn과 함께 사용해 효율적이고 안전한 웹 서비스 환경을 구축할 수 있는 기술이다. 본 프로젝트에서는 도메인을 통해 요청한 정보를 Nginx에서 로컬호스트 Gunicorn 서버로 바인딩하는 역할로 사용되었다.

5. SQLite

대시보드 내 데이터의 기반이 될 DRM은 SQLite3을 사용하고 있는데, 단일 파일로 활용하는 데이터베이스이기 때문에, 가볍고 빨라 본 서비스처럼 Odroid C4와 같은 싱글보드 컴퓨터에서 사용하기에 적합했다. 따라서 웹페이지 또한 동일한 서버에서 호스트하기 위해 SQLite를 사용했다.

6. REST API(Representational State Transfer API)

REST API는 리소스 중심의 설계와 HTTP 표준을 따르는 API로, 다양한 클라이언트와의 통신이 쉽다. 또한 각 요청이 독립적으로 처리되고, 서버에 이전 요청의 상태를 저장하지 않아 서버의 부담을 줄일 수 있어 대시보드를 개발하는데 사용하기에 적합한 기술이다.

7. Data Visualization

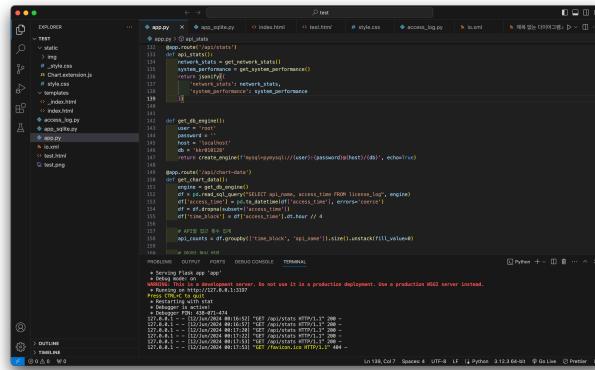
대시보드에서 데이터를 시각화하기 위해 Pandas와 Matplotlib를 사용했다. Pandas는 데이터 분석을 위한 라이브러리로, 데이터 조작 및 분석에 최적화되어 있다. Matplotlib는 다양한 유형의 차트를 쉽게 생성할 수 있는 라이브러리로, 데이터를 시각적으로 표현하는 데 사용되었다. 이들은 복잡한 데이터도 직관적으로 이해할 수 있게 도와준 기술이다.

5. 개발 환경

1) 개발 도구 및 실험 환경

• Visual Studio Code

다양한 확장기능과 작업환경 커스텀이 가능하고, 다양한 종류의 파일을 다루기에 적합한 VSC를 사용해 코드를 작성했다.



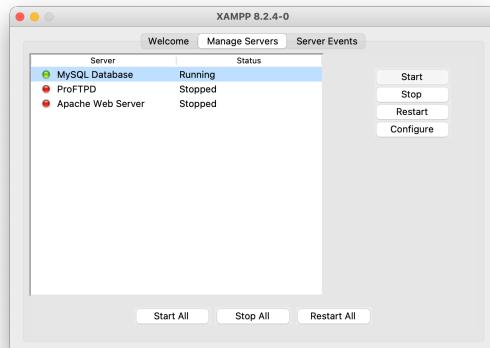
• Termius

다양한 플랫폼에서 SSH, SFTP, Snippet 기능을 제공하는 프로그램으로, 배포할 서버로의 원격 접속 및 관리를 위해 사용되었다.



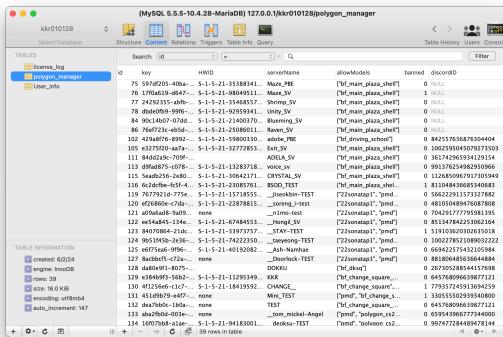
• XAMPP

Apache, MySQL 등을 포함하는 웹 서버 솔루션 스택 패키지로, 개발과 테스트 환경의 운영 체제에서 웹 서버를 쉽게 설정 및 관리하기 위해 사용했다.



• Sequel Pro 와 MySQL

개발 및 테스트 환경에서 GUI를 이용해 데이터베이스를 조작, 관리하기 위해 사용된 MacOS 전용 MySQL 프로그램으로, 기능 구현을 쉽게 하기 위해 사용되었다. MySQL은 배포 과정에서 서버에서 사용 중인 SQLite3과 높은 호환성을 가졌다.



• Ubuntu MATE 22.04 LTS

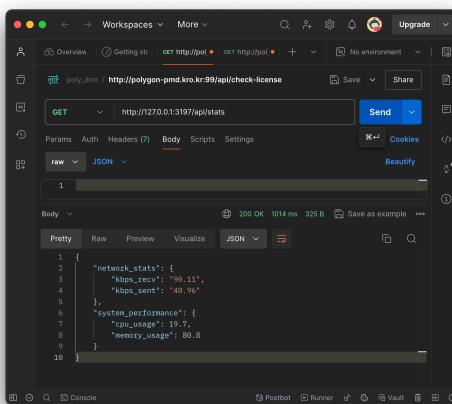
개발한 작업물이 배포될 환경으로, 싱글 보드 컴퓨터인 Hard Kernel사의 Odroid C4에 설치된 운영체계이다. 개발 환경인 MacOS와 대부분의 명령어 및 작업의 호환성이 매우 높다.



Odroid C4

• Postman

웹페이지를 통한 API 통신을 위해 해당 API가 올바르게 동작하는지 디버깅하기 위해 사용한 툴로, 요청과 응답을 확인할 수 있어 API 개발을 쉽고 빠르게 구현할 수 있다.



2) 모듈 및 라이브러리

- 사용된 모듈의 기능 및 역할

- ① **flask**: 웹 서버 구축 및 REST API 구현
- ② **pymysql**: 테스트 개발 과정에서 MySQL 데이터베이스와의 연결 및 상호작용(데이터 저장 및 조회)
- ③ **sqlite3**: 배포될 Ubuntu 환경에서의 로컬 데이터베이스 파일 관리 용도로 사용
- ④ **json**: API를 이용해 응답을 처리하는 과정에서의 데이터 교환 및 객체 변환을 담당함
- ⑤ **os**: 호스트의 운영 체제와 상호작용하며, 파일 경로 및 디렉토리 구조를 다룸
- ⑥ **psutil**: 시스템 및 프로세스 유ти리티 모듈로, 시스템 성능 데이터를 수집함
- ⑦ **pandas**: 데이터 분석 및 조작 라이브러리로, 데이터베이스에서 조회한 데이터를 처리하는 데 사용함
- ⑧ **sqlalchemy**: 데이터베이스와의 연결 관리 및 쿼리 실행, 데이터베이스 엔진 생성 및 SQLite 접근
- ⑨ **numpy**: 방사형 차트의 각도 계산 등의 수치 데이터를 처리하는 데 사용함
- ⑩ **matplotlib**: 데이터 시각화 라이브러리로, 차트와 그래프를 디버그용 로컬 이미지로 생성하는 데 사용함
- ⑪ **datetime**: 로그의 타임스탬프를 생성 및 처리하기 위해 사용함

- 사용된 외부 라이브러리의 기능 및 역할

① Chart.js

데이터 시각화를 위해 사용된 JavaScript 라이브러리로, 다양한 차트를 쉽게 만들 수 있도록 도와준다. 본 프로젝트에서는 바, 레이더 차트를 구현하는 데 사용되었다. Chart.js는 간단한 설정으로 다양한 차트를 만들 수 있어 대시보드에 필요한 시각적 요소를 빠르게 구현할 수 있었다. 또한, 다양한 옵션 파라미터와 반응형 디자인을 지원해 다양한 화면 크기에서도 일관된 차트 표현이 가능했다.

② Bootstrap

웹 페이지의 접근성을 높이기 위해 사용된 프론트엔드 프레임워크로, 다양한 기기에서 일관된 UI를 제공하는 데 도움을 준다. Bootstrap은 미리 정의된 CSS 및 JavaScript 컴포넌트를 제공해, 적은 코드로 빠르게 웹 페이지의 디자인을 구현할 수 있다. 이는 대시보드의 개발 시간을 단축시키고, 다양한 플랫폼에서 호환성을 보장할 수 있었다.

6. 데이터베이스 및 API

1) 데이터베이스 개요 및 테이블 구조

본 프로젝트의 테스트 환경에서는 kkr010128, 배포 환경에서는 database.db 데이터베이스를 사용하며, 주요 테이블로는 license_log, polygon_manager가 있다.

테이블 구조는 아래와 같다.

① license_log 테이블:

이 테이블은 API 요청 로그를 저장한다. 각 로그에는 요청된 API 이름과 접근 시간이 기록된다.

```
id: INT, AUTO_INCREMENT, PRIMARY KEY  
api_name: VARCHAR(50)  
access_time: DATETIME
```

② polygon_manager 테이블:

이 테이블은 각 사용자의 라이센스 정보를 관리한다. 각 사용자는 고유한 key(발급한 라이센스)를 가지고 있으며, 귀속 대상 호스트의 HWID, 허용된 리소스 모델과 사용자 차단 여부, DiscordID 값과 같은 추가 정보를 가지고 있다.

```
id: INT, AUTO_INCREMENT, PRIMARY KEY  
key: VARCHAR  
HWID: VARCHAR  
serverName: VARCHAR  
allowModels: JSON  
banned: BOOLEAN  
discordID: VARCHAR
```

2) API 엔드포인트 목록 및 기능

- GET /api/chart-data

기능: 라이센스 로그 데이터를 가져와 시간대별 API 호출 횟수를 반환한다.

응답: JSON 형식의 데이터로, 시간대와 각 API의 호출 횟수를 포함한다.

```
@app.route('/api/chart-data')
def get_chart_data():
    engine = get_db_engine()
    df = pd.read_sql_query("SELECT api_name, access_time FROM license_log", engine)
    df['access_time'] = pd.to_datetime(df['access_time'], errors='coerce')
    df = df.dropna(subset=['access_time'])
    df['time_block'] = df['access_time'].dt.hour // 4

    # API별 접근 횟수 카운트
    api_counts = df.groupby(['time_block', 'api_name']).size().unstack(fill_value=0)

    # 데이터 형식 변경
    labels = ['0-4', '4-8', '8-12', '12-16', '16-20', '20-24']
    data = {api: api_counts[api].reindex(range(6), fill_value=0).tolist() for api in api_counts.columns}

    return jsonify({'labels': labels, 'data': data})
```

- GET /api/stats

기능: 시스템의 성능 및 네트워크 통계를 반환한다.

응답: JSON 형식의 데이터로, CPU 사용량과 메모리 사용량, 송신 및 수신 속도를 포함한다.

```
@app.route('/api/stats')
def api_stats():
    network_stats = get_network_stats()
    system_performance = get_system_performance()
    return jsonify({
        'network_stats': network_stats,
        'system_performance': system_performance
    })
```

7. 데이터 시각화

1) 사용된 시각화 라이브러리 및 도구

본 프로젝트에서는 Chart.js와 Matplotlib를 이용해 데이터베이스에서 정보를 가져와 시각화를 진행했다. Chart.js는 실제 배포될 웹 페이지에서 사용중인 리소스의 개수를 막대형 차트로 표현했고, Matplotlib으로는 시간대별 API 호출 횟수를 방사형 차트로 표현했다. 아래는 방사형 차트를 생성하기 위한 코드다.

```
# 방사형 차트 생성 함수
def generate_radar_chart():
    conn = pymysql.connect(**db_config)
    df = pd.read_sql_query("SELECT api_name, access_time FROM license_log", conn)
    df['access_time'] = pd.to_datetime(df['access_time'])
    df['time_block'] = df['access_time'].dt.hour // 4

    # API별 접근 횟수 집계
    api_counts = df.groupby(['time_block', 'api_name']).size().unstack(fill_value=0)

    # 방사형 차트 생성
    labels = ['0-4', '4-8', '8-12', '12-16', '16-20', '20-24']
    num_vars = len(labels)

    angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()
    angles += angles[:-1]

    fig, ax = plt.subplots(figsize=(12, 8), subplot_kw=dict(polar=True))

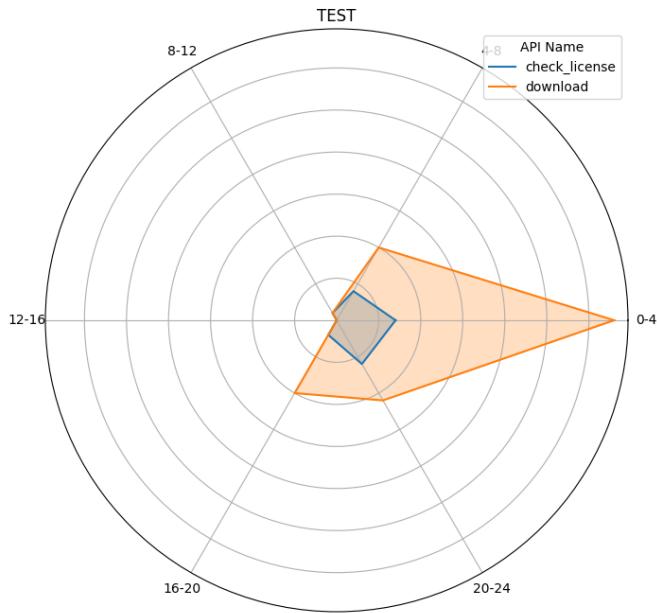
    for api in api_counts.columns:
        values = api_counts[api].reindex(range(6), fill_value=0).tolist()
        values += values[:-1]
        ax.plot(angles, values, label=api)
        ax.fill(angles, values, alpha=0.25)

    ax.set_yticklabels([])
    ax.set_xticks(angles[:-1]) # 마지막 각도를 제외하여 길이를 맞춤
    ax.set_xticklabels(labels)

    plt.title('TEST')
    plt.legend(title='API Name')
    plt.savefig('test.png') # 파일 저장 경로 변경
    conn.close()
```

1. license_log 테이블 내 api_name과 access_time 칼럼 데이터를 pandas데이터프레임으로 변환한다.
2. access_time은 to_datetime을 사용해 datetime 형식으로 변환한다.
3. 생성할 차트의 가독성을 위해 변환한 datetime을 6개의 시간대로 나눈다.
4. groupby를 사용해 time_block과 api_name을 기준으로 접근 횟수를 집계하고, 빈 값은 0으로 채운다. 채우지 않을 경우 차트에 표시할 데이터가 없어 오류가 생겼다.
5. 차트를 그리기 위해 labels와 num_vars를 설정했으며, 각각 시간 블록과 라벨의 개수를 의미한다.
6. numpy로 라벨에 대한 각도를 계산하고, angles += angles[:-1]로 마지막 각과 첫 각을 일치시킨다.
7. 전체 그래프를 담는 컨테이너인 figure 객체를 생성하고, polar=true로 방사형으로 생성한다.
8. for문을 이용해 각 API별 데이터를 차트에 추가하고 ax.plot과 ax.fill로 각도에 따라 데이터를 그린다.
9. y축을 제거하고 라벨을 x축과 동일하게 설정한다.
10. 차트를 test.png 파일로 저장한다.

2) 시각화된 데이터 설명



시각화된 데이터는 위와 같은 모습의 이미지로 생성되었고, 해당 데이터는 데이터 테이블 내에 기록된 check-license와 download API의 접근 로그 데이터를 기반으로 그려졌다. 이 차트를 통해 서비스를 사용하는 사용자들이 라이센스를 인증하고 다운로드 받는 시간대를 파악할 수 있었고, 이로 인해 이용 빈도가 상대적으로 적은 구간과 빈도가 높은 구간을 확인할 수 있었다. 이 데이터는 서비스의 점검 또는 업데이트가 요구될 때 사용자들의 불편을 최소화 할 수 있는 시간대를 찾는 근거를 마련하는 데이터로 사용할 수 있다.

8. 시스템 모니터링

1) 사용된 모니터링 도구 및 라이브러리

- Psutil

psutil은 시스템 성능 및 자원 사용을 모니터링하기 위한 라이브러리다. 본 프로젝트에서는 해당 모듈을 사용해 CPU, RAM, 트래픽 등의 시스템 성능 데이터를 수집했다. psutil.cpu_percent(interval=1) 메서드를 통해 CPU 사용률을, psutil.virtual_memory() 메소드를 통해 메모리 사용률을, psutil.net_io_counters() 메소드를 통해 네트워크 입출력을 모니터링했다.

- OS 모듈:

OS 모듈은 운영 체제와 상호 작용하는 데 사용된다. 이 프로젝트에서는 디렉토리 파일을 모니터링하고 관리하기 위해 사용했다. os.path.exists(directory) 메서드를 사용하여 디렉토리의 존재 여부를 확인하고, os.listdir(directory) 메소드를 사용하여 디렉토리 내 파일 및 폴더 목록을 가져왔다. 이를 통해 특정 디렉토리의 파일 수를 세고, 디렉토리의 변화 상태를 모니터링했다.

- SQL:

SQL을 사용하여 데이터베이스 내 리소스 및 고객 정보를 모니터링했다. 데이터베이스 쿼리를 통해 리소스 및 고객 정보를 실시간으로 조회하고, 이를 기반으로 모니터링 데이터를 수집했다. 예를 들어, 고객 테이블의 내용을 주기적으로 조회하여 새로운 고객 등록 또는 고객 정보 변경 여부를 모니터링했다.

2) 모니터링 항목

- CPU 사용률:

CPU 사용률은 시스템의 프로세서가 현재 얼마나 사용되고 있는지를 나타낸다.

`psutil.cpu_percent(interval=1)` 메서드를 사용하여 1초 간격으로 CPU 사용률을 측정했다.

- 메모리 사용률:

메모리 사용률은 시스템의 RAM이 얼마나 사용되고 있는지를 나타낸다.

`psutil.virtual_memory()` 메서드를 사용하여 전체 메모리 중 사용 중인 메모리의 비율을 확인했다.

- 네트워크 입출력:

네트워크 입출력은 네트워크 인터페이스를 통해 주고받는 데이터의 양을 측정한다.

`psutil.net_io_counters()` 메서드를 사용하여 이전 측정값과의 차이를 계산하여 초당 전송된 바이트 수 및 수신된 바이트 수를 모니터링했다.

- 디렉토리 파일 모니터링:

특정 디렉토리의 파일 수를 모니터링했다.

`os.path.exists(directory)` 메서드를 사용하여 디렉토리의 존재 여부를 확인하고, `os.listdir(directory)` 메서드를 사용하여 디렉토리 내 파일 및 폴더 목록을 가져왔다.

- SQL을 사용한 리소스 및 고객 모니터링:

데이터베이스 내 리소스 및 고객 정보를 모니터링했다.

주기적으로 데이터베이스 쿼리를 실행하여 리소스 및 고객 테이블의 변화를 감지하고, 이를 기반으로 모니터링 데이터를 수집했다.

이와 같이 시스템 모니터링 도구 및 라이브러리를 활용하여 대시보드에 실시간으로 시스템 성능 데이터를 시각화하고, 리소스와 사용자를 효과적으로 관리할 수 있도록 구성했다.

9. 테스트 및 배포

1) 테스트 시나리오 및 결과

목표: 프로젝트의 기능과 성능을 검증하기 위해 다양한 테스트 시나리오를 수행했다.

테스트 환경: macOS

테스트 도구: 브라우저, Postman, Visual Studio Code

시스템 성능 테스트

시나리오:

1. CPU 및 메모리 사용률 확인: 성능 카드를 통해 실시간 CPU 및 메모리 사용률을 확인한다.
2. 네트워크 트래픽 확인: 네트워크 트래픽 카드를 통해 실시간 네트워크 입출력 상황을 확인한다.

예상 결과:

1. CPU 및 메모리 사용률이 실시간으로 정확하게 표시된다.
2. 네트워크 트래픽이 실시간으로 정확하게 표시된다.

결과:

1. CPU 및 메모리 사용률이 예상대로 정확하게 표시되었다.
2. 네트워크 트래픽이 예상대로 정확하게 표시되었다.

데이터베이스 테스트

시나리오:

1. Create: 새로운 데이터 레코드를 데이터베이스에 추가한다.
2. Read: 데이터베이스에서 데이터를 읽어와 대시보드에 표시한다.
3. Update: 기존 데이터 레코드를 수정하고 변경 사항을 반영한다.

예상 결과:

1. 데이터가 성공적으로 추가된다.
2. 데이터가 정확하게 읽어와진다.
3. 데이터가 성공적으로 수정된다.

결과:

1. 데이터가 성공적으로 추가되었다.
2. 데이터가 정확하게 읽어와졌다.
3. 데이터가 성공적으로 수정되었다.

REST API 테스트

시나리오:

1. API 엔드포인트 호출: Postman을 사용하여 다양한 API 엔드포인트를 호출하고 응답을 확인한다.
 - /api/stats: 시스템 성능 및 네트워크 통계를 가져온다.
 - /api/chart-data: 차트에 필요한 데이터를 가져온다.

예상 결과:

1. 모든 API 엔드포인트가 예상대로 동작하며, 올바른 데이터를 반환한다.

결과:

1. 모든 API 엔드포인트가 예상대로 동작하며, 올바른 데이터를 반환하였다.

데이터 시각화 테스트

시나리오:

1. 차트 데이터 로딩: 대시보드에서 차트 데이터를 로딩하고 정확하게 표시한다.
2. 차트 업데이트: 실시간으로 차트 데이터를 업데이트하고 새로고침 해 변경 사항을 반영한다.

예상 결과:

1. 차트 데이터가 정확하게 로딩되고 표시된다.
2. 차트 데이터가 실시간으로 업데이트된다.

결과:

1. 차트 데이터가 정확하게 로딩되고 표시되었다.
2. 차트 데이터가 실시간으로 업데이트되었다.

사용자 인터페이스 테스트

시나리오:

1. 페이지 로드: 웹 브라우저에서 대시보드를 로드하고 모든 UI 요소가 제대로 표시되는지 확인한다.
2. 반응형 디자인: 다양한 화면 크기에서 대시보드의 레이아웃과 UI 요소가 제대로 반응하는지 확인한다.

예상 결과:

1. 모든 UI 요소가 제대로 표시된다.
2. 대시보드가 다양한 화면 크기에서 적절하게 반응한다.

결과:

1. 모든 UI 요소가 제대로 표시되었다.
2. 대시보드가 다양한 화면 크기에서 적절하게 반응하였다.

테스트 결과: 모든 테스트 시나리오가 성공적으로 수행되었으며, 프로젝트의 기능과 성능이 예상대로 동작함을 확인하였다.

추가 검토 사항: 과제 제출 목적이 아닌 실제 서비스를 목적으로 사용할 경우, 데이터 보안 및 인증서 등 보안 및 일부 예외 처리 로직에 대한 추가적인 검토가 필요할 수 있다. 그러나 전체적인 시스템은 안정적으로 동작하고 있으며, 개발 요구사항을 충족하는 것으로 판단된다.

2) 배포 과정 및 환경 설정

테스트를 마친 코드는 서비스를 제공하는 서버 환경에 맞추기 위해 MySQL 관련 구문을 SQLite3으로 변경하고, 데이터베이스와 테이블 명을 동기화했다. 그 후 Termius SFTP를 사용해 서버로 접속, /home/odroid/Desktop/DashBoard 디렉토리에 프로젝트 파일을 업로드 했다. 그 후 내부 코드에서 DRM 디렉토리인 /home/odroid/Desktop/Resource-Manager 폴더로 연결해주었고, 라이센스 접근시 로깅하는 access_log.py를 DRM의 디렉토리로 옮겨 모듈을 정상적으로 로딩할 수 있도록 설정해주었다.

그 후 pip install gunicorn을 진행하고,
cd Desktop/DashBoard,
gunicorn -w 3 -b 127.0.0.1:3197 app:app을 사용해 3197 포트로 로컬 서버를 바인딩했다.

다음으로 외부와의 프록시를 연결하기 위해 sudo apt install nginx를 진행하고,
sudo nano etc/nginx/sites-available/dashboard을 사용해 다음과 같이 설정 파일을 작성했다.

```
server {
    listen 3197;
    server_name polygon-dash.kro.kr;
    location / {
        proxy_pass http://127.0.0.1:3197;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
    location /static {
        alias /home/odroid/Desktop/DashBoard/static;
    }
}
```

마지막으로, 네트워크 관리 페이지(192.168.0.1)에 접속해 3197로 내/외부 포트를 포워딩 해주었다.

이 과정은 웹 브라우저를 통해 접속하는 사용자를 nginx 프록시 엔진을 거쳐 gunicorn 서버로 이동, 그 후 웹 Flask 서버를 통해 데이터베이스와 통신할 수 있게 해준다.

10. 성과 및 과제

본 프로젝트에서 요구했던 모든 기능들은 정상 작동했고, 배포 또한 문제 없이 진행됐다. 이를 통해 서비스 관리자는 사용자들의 리소스와 서버 상태, 관련 정보 등을 쉽게 접근해 파악할 수 있는 성과를 달성하게 되었다. 프로젝트는 성공적이었으나, 실제 서비스에 앞서 관리자 접근 권한과 로그인, 기타 부가 기능 개발 및 SSL 인증서 연결이 요구될 것으로 보인다. 또한 커맨드 및 SQL injection 등의 보안 솔루션도 고려해야 될 것이라 생각된다.

11. 전체 프로그램 코드 및 관련 참고 자료

1) app.py(sqlite3 version)

```
⚡ app_sqlite.py > ...
1  from flask import Flask, render_template, jsonify
2  import json
3  import os
4  import psutil
5  import pandas as pd
6  import sqlite3
7  from sqlalchemy import create_engine
8
9  app = Flask(__name__)
10
11 previous_net_io = psutil.net_io_counters()
12
13 # 데이터베이스 연결 및 데이터 가져오기 함수
14 def get_data():
15     try:
16         conn = sqlite3.connect('database.db')
17         conn.row_factory = sqlite3.Row
18         cursor = conn.cursor()
19         cursor.execute('SELECT * FROM polygon_manager')
20         rows = cursor.fetchall()
21         conn.close()
22         return [dict(row) for row in rows]
23     except sqlite3.Error as e:
24         print(f"Connection Failed: {e}")
25     return []
26
27 def get_model_counts(data):
28     all_models = []
29     for row in data:
30         models = json.loads(row['allowModels'])
31         all_models.extend(models)
32     model_counts = {model: all_models.count(model) for model in set(all_models)}
33     return model_counts
34
35 def get_user_counts(data):
36     discord_ids = set()
37     for row in data:
38         discord_ids.add(row['discordID'])
39     return len(discord_ids)
40
41 def get_folder_count(directory):
42     if not os.path.exists(directory):
43         print(f"Directory does not exist: {directory}")
44     return 0
45     try:
46         return len([name for name in os.listdir(directory) if os.path.isdir(os.path.join(directory, name))])
47     except FileNotFoundError as e:
48         print(f"Error: {e}")
49     return 0
50
51 def get_dict_model_counts(data):
52     all_models = []
53     for row in data:
54         models = json.loads(row['allowModels'])
55         all_models.extend(models)
56     dict_model_counts = {model: all_models.count(model) for model in set(all_models) if all_models.count(model) > 1}
57     return dict_model_counts
58
59 def get_network_stats():
60     global previous_net_io
61     current_net_io = psutil.net_io_counters()
62     bytes_sent_per_sec = current_net_io.bytes_sent - previous_net_io.bytes_sent
63     bytes_recv_per_sec = current_net_io.bytes_recv - previous_net_io.bytes_recv
64     previous_net_io = current_net_io
65
66     # Convert to kbps
67     bits_sent_kbps = (bytes_sent_per_sec * 8) / 1_000
68     bits_recv_kbps = (bytes_recv_per_sec * 8) / 1_000
69
```

```

70     return {
71         'kbps_sent': f'{bits_sent_kbps:.2f}',
72         'kbps_recv': f'{bits_recv_kbps:.2f}'
73     }
74
75     def get_system_performance():
76         cpu_usage = psutil.cpu_percent(interval=1)
77         memory_info = psutil.virtual_memory()
78         memory_usage = memory_info.percent
79         return {
80             'cpu_usage': cpu_usage,
81             'memory_usage': memory_usage
82         }
83
84     @app.route('/')
85     def index():
86         data = get_data()
87         model_counts = get_model_counts(data)
88         model_counts_json = json.dumps(model_counts)
89         user_counts = get_user_counts(data)
90         dict_model_counts = get_dict_model_counts(data)
91
92         # 웬만하면 절대 경로 사용
93         model_directory_path = os.path.abspath("/home/odroid/Desktop/Resource-Manager/models")
94         dummy_directory_path = os.path.abspath("/home/odroid/Desktop/Resource-Manager/models_dummy")
95
96         # 현재 작업 디렉토리 로그로 출력
97         print(f"Current working directory: {os.getcwd()}")
98         print(f"Model directory path: {model_directory_path}")
99         print(f"Dummy directory path: {dummy_directory_path}")
100
101        uploaded_rsc_cnt = get_folder_count(model_directory_path) # 디렉토리 내 리소스 총 개수
102        dummy_rsc_cnt = get_folder_count(dummy_directory_path) # 디렉토리 내 더미리소스 총 개수
103        license_cnt = len(data) # 등록된 라이센스 개수
104        license_rsc = len(model_counts) # 라이센스에 등록된 리소스 개수(중복 제외)
105        assigned_rsc = sum(model_counts.values()) # 라이센스에 할당 된 리소스 총 개수(중복 허용)
106
107        network_stats = get_network_stats()
108        system_performance = get_system_performance()
109
110        return render_template('index.html',
111                               model_counts=model_counts_json,
112                               uploaded_rsc_cnt=uploaded_rsc_cnt,
113                               dummy_rsc_cnt=dummy_rsc_cnt,
114                               license_cnt=license_cnt,
115                               license_rsc=license_rsc,
116                               assigned_rsc=assigned_rsc,
117                               network_stats=network_stats,
118                               system_performance=system_performance,
119                               user_counts=user_counts,
120                               dict_model_counts = json.dumps(dict_model_counts)
121
122                           )
123
124     @app.route('/api/stats')
125     def api_stats():
126         network_stats = get_network_stats()
127         system_performance = get_system_performance()
128         return jsonify({
129             'network_stats': network_stats,
130             'system_performance': system_performance
131         })
132
133
134     def get_db_engine():
135         return create_engine('sqlite:///database.db', echo=True)

```

```
136
137     @app.route('/api/chart-data')
138     def get_chart_data():
139         engine = get_db_engine()
140         df = pd.read_sql_query("SELECT api_name, access_time FROM license_log", engine)
141         df['access_time'] = pd.to_datetime(df['access_time'], errors='coerce')
142         df = df.dropna(subset=['access_time'])
143         df['time_block'] = df['access_time'].dt.hour // 4
144
145         # API별 접근 횟수 집계
146         api_counts = df.groupby(['time_block', 'api_name']).size().unstack(fill_value=0)
147
148         # 데이터 형식 변경
149         labels = ['0-4', '4-8', '8-12', '12-16', '16-20', '20-24']
150         data = {api: api_counts[api].reindex(range(6), fill_value=0).tolist() for api in api_counts.columns}
151
152         return jsonify({'labels': labels, 'data': data})
153
154     if __name__ == '__main__':
155         app.run(debug=True, port=3197)
```

2) license_log.py

```
access_log.py > generate_radar_chart
1  import pymysql
2  import pandas as pd
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from datetime import datetime
6
7  # MySQL 연결 설정
8  db_config = {
9      'host': 'localhost',
10     'user': 'root',
11     'password': '', # 비밀번호가 없는 경우
12     'database': 'kkkr010128'
13 }
14
15 # 로그 테이블 생성 함수
16 def create_log_table():
17     conn = pymysql.connect(**db_config)
18     cursor = conn.cursor()
19     cursor.execute("""
20         CREATE TABLE IF NOT EXISTS license_log (
21             id INT AUTO_INCREMENT PRIMARY KEY,
22             api_name VARCHAR(50),
23             access_time DATETIME
24         )
25     """)
26     conn.commit()
27     conn.close()
28
29 # log_down 함수
30 def log_down():
31     conn = pymysql.connect(**db_config)
32     cursor = conn.cursor()
33     access_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
34     cursor.execute("INSERT INTO license_log (api_name, access_time) VALUES (%s, %s)", ('download', access_time))
35     conn.commit()
36     conn.close()
37
38 # log_check 함수
39 def log_check():
40     conn = pymysql.connect(**db_config)
41     cursor = conn.cursor()
42     access_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
43     cursor.execute("INSERT INTO license_log (api_name, access_time) VALUES (%s, %s)", ('check_license', access_time))
44     conn.commit()
45     conn.close()
46
```

```

46
47 # 방사형 차트 생성 함수
48 def generate_radar_chart():
49     conn = pymysql.connect(**db_config)
50     df = pd.read_sql_query("SELECT api_name, access_time FROM license_log", conn)
51     df['access_time'] = pd.to_datetime(df['access_time'])
52     df['time_block'] = df['access_time'].dt.hour // 4
53
54     # API별 접근 횟수 집계
55     api_counts = df.groupby(['time_block', 'api_name']).size().unstack(fill_value=0)
56
57     # 방사형 차트 생성
58     labels = ['0~4', '4~8', '8~12', '12~16', '16~20', '20~24']
59     num_vars = len(labels)
60
61     angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()
62     angles += angles[:1]
63
64     fig, ax = plt.subplots(figsize=(12, 8), subplot_kw=dict(polar=True))
65
66     for api in api_counts.columns:
67         values = api_counts[api].reindex(range(6), fill_value=0).tolist()
68         values += values[:1]
69         ax.plot(angles, values, label=api)
70         ax.fill(angles, values, alpha=0.25)
71
72     ax.set_yticklabels([])
73     ax.set_xticks(angles[:-1]) # 마지막 각도를 제외하여 길이를 맞춤
74     ax.set_xticklabels(labels)
75
76     plt.title('TEST')
77     plt.legend(title='API Name')
78     plt.savefig('test.png') # 파일 저장 경로 변경
79     conn.close()
80
81     conn = pymysql.connect(**db_config)
82     cursor = conn.cursor()
83     cursor.executemany("INSERT INTO license_log (api_name, access_time) VALUES (%s, %s)", test_data)
84     conn.commit()
85     conn.close()
86
87 if __name__ == "__main__":
88     create_log_table()
89     log_down() # 테스트를 위해 호출
90     log_check() # 테스트를 위해 호출
91     # insert_test_data()
92     generate_radar_chart()

```

3) index.html (코드가 길어 JS 부분만 삽입하며, 전체 코드는 파일로 첨부하겠습니다.)

```
<script> // 실시간 서버 정보 반영
  function updateStats() {
    fetch('/api/stats')
      .then(response => response.json())
      .then(data => {
        document.getElementById('traffic').innerHTML = `Down: ${data.network_stats.kbps_recv}<br>Up: ${data.network_stats.kbps_sent}`;
        document.getElementById('performance').innerHTML = `CPU: ${data.system_performance.cpu_usage}%<br>RAM: ${data.system_performance.memory_usage}%`;
      })
      .catch(error => console.error('Error fetching stats:', error));
  }
  setInterval(updateStats, 3000); // 2초 간격으로 갱신
</script>
<script>
  document.addEventListener('DOMContentLoaded', function () {
    // 서버에서 전달된 JSON 데이터 파싱
    var dictModelCounts = JSON.parse('{{ dict_model_counts|safe }}');
    var labels = Object.keys(dictModelCounts); // 레이블(키) 추출
    var dataValues = Object.values(dictModelCounts); // 데이터 값 추출

    var ctx = document.getElementById('chart1').getContext('2d'); // 차트 영역 context 불러옴

    // 차트 데이타를 설정합니다.
    var data = {
      labels: labels, // dict_model_counts의 키를 레이블로 설정
      datasets: [
        {
          label: '리소스 사용량',
          data: dataValues, // dict_model_counts의 값을 데이터로 설정
          backgroundColor: 'rgba(251, 99, 64, 0.8)',
          radius: 50,
          borderWidth: 2,
          borderRadius: 100,
          borderSkipped: false,
          barPercentage: 0.7,
          hoverBackgroundColor: 'rgba(251, 99, 64, 1)',
          hoverBorderColor: 'rgba(251, 99, 64, 0.5)'
        }
      ];
    };

    // 차트 옵션을 설정합니다.
    var options = {
      responsive: true,
      maintainAspectRatio: false,
      indexAxis: 'x', // 가로 막대 차트로 전환
      scales: {
        x: {
          beginAtZero: true,
          grid: {drawOnChartArea: false}, // x 축 그리드 숨김
          ticks: {color: 'white', font: {size: 8}}, // y축 라벨 색상 흰색, 폰트 사이즈 조정
        },
        y: {
          max: 30, // 차트 내 최댓값 표시 제한
          ticks: {
            color: 'white', // y축 라벨의 색상을 흰색으로 설정
            font: {size: 8}
          }
        },
      },
      plugins: {
        legend: {display: false}, // 라벨을 표시하지 않음
      },
    };
    // 차트 생성
    var myBarChart = new Chart(ctx, {
      type: 'bar', // 차트 유형 (bar, line, pie 등)
      data: data,
      options: options,
      //plugins: [ChartDataLabels],
    });
  });
</script>
```

```
<script>
    async function fetchData() {
        const response = await fetch('/api/chart-data');
        const result = await response.json();
        return result;
    }

    async function createChart() {
        const data = await fetchData();
        const ctx = document.getElementById('chart2').getContext('2d');
        const colors = [
            {backgroundColor: 'rgba(54, 162, 235, 0.2)', borderColor: 'rgba(54, 162, 235, 1)', pointBackgroundColor: 'rgba(54, 162, 235, 1')},
            {backgroundColor: 'rgba(255, 99, 132, 0.2)', borderColor: 'rgba(255, 99, 132, 1)', pointBackgroundColor: 'rgba(255, 99, 132, 1')}

        ];
        new Chart(ctx, {
            type: 'line',
            data: {
                labels: data.labels,
                datasets: Object.keys(data.data).map((api, index) => ({
                    label: api,
                    data: data.data[api],
                    fill: true,
                    backgroundColor: colors[index % 2].backgroundColor,
                    borderColor: colors[index % 2].borderColor,
                    pointBackgroundColor: colors[index % 2].pointBackgroundColor,
                    tension: 0.4 // 곡선의 부드러움을 조절하는 속성 (0으로 설정하면 직선, 1로 설정하면 매우 부드러운 곡선)
                }))
            },
            options: {
                scales: {
                    r: {
                        angleLines: {
                            color: 'white' // 그리드 선 색상
                        },
                        grid: {
                            color: 'white' // 그리드 색상
                        },
                        pointLabels: {
                            color: 'white' // 라벨 색상
                        },
                        ticks: {
                            display: false // 척도 숨김
                        }
                    },
                    plugins: {
                        legend: {
                            display: false // 범례 숨김
                        }
                    }
                }
            }
        });
    }

```

본 프로젝트에 사용된 DRM *별첨 자료와 동일

https://velog.io/@kk_0128_/drm

배포된 프로젝트

<http://polygon-dash.kro.kr:3197>

Chat.js Document

<https://www.chartjs.org/docs/latest/samples/information.html>

Bootstrap Document

<https://getbootstrap.com/docs/5.3/getting-started/introduction/>

SQLite3 Tutorial

<https://docs.python.org/3/library/sqlite3.html>

Matplotlib Tutorial

https://matplotlib.org/stable/gallery/specify_plots/radar_chart.html

SQLAlchemy Document

<https://docs.sqlalchemy.org/en/20/orm/quickstart.html>

Pandas Document

https://pandas.pydata.org/docs/reference/api/pandas.read_sql_query.html#pandas-read-sql-query

https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html#pandas-to-datetime

그 외 참고 문헌

<https://velog.io/@baeyuna97/SQLAlchemy-사용하기>