2023-2 창의적통합설계 최종 보고서

FOSSLight Hub CLI & Web

FOSSLight Hub 프로그래밍 인터페이스 및 사용자 친화적 Web 개발

2016-10399 최덕경

2016-19965 김성은

2016-10586 한재훈

엄현상 교수님

민경선 담당자님 (LG Electronics, Open Source Task)

Table of Contents

1. Abstract

본 프로젝트는 LG 전자가 이미 개발하고 출시한 FOSSLight Hub 라는 웹 서비스의 기능을 확장하기 위한 추가 소프트웨어를 개발하는 것을 주된 목적으로 삼는다. FOSSLight Hub 는 프로젝트에 속한 오픈 소스 소프트웨어의 목록을 관리하고 그것과 관련된 각종 유용한 기능들을 간편하게 활용할 수 있도록 고안된 웹 서비스이다. 이때 기존의 웹 서비스에서만 사용할 수 있던 기능들을 커맨드 라인에서도 활용할 수 있도록 CLI 를 개발하여 제공하는 것이 첫 번째 과제이고, 관리자의 역할까지 고려된 기존의 웹 서비스와 달리 사용자의 역할만 고려한 사용자 친화적 웹 서비스를 추가로 개발하여 제공하는 것이 두 번째 과제이다. 더불어, 이러한 과제들을 진행하는 데 필요한 각종 API 또한 그동안 유지보수가 온전히 이뤄지고 있지 않았으므로 API 의 전체적인 리뉴얼까지 함께 진행하기로 한다.

CLI 의 개발은 기존 웹 서비스의 기능을 프로그래밍 인터페이스로도 접근할 수 있는 환경을 제공함으로써 CI/CD 와 같은 자동화 시스템 구축에 활용될 수 있는 확장성과 유연성을 제공할 것이며, 필요시 전체적으로 리뉴얼이 이뤄진 API 를 활용하여 각자에게 필요한 커스텀 소프트웨어를 개발할 수 있는 가능성도 열 것이다. 또한, 사용자 친화적 웹 서비스의 개발은 기존 웹 서비스의 불편하고 복잡한 UI & UX 를 벗어나 오로지 사용자에게만 필요한 기능 위주의 UI & UX 를 제공함으로써 많은 사용자에게 크나큰 사용성의 개선을 가져다줄 것이다. 마지막으로, 기존의 FOSSLight Hub 에 이러한 소프트웨어들이 추가되어 상용화된다면 오픈 소스 소프트웨어 관리의 중요성에 관한 인식도 전보다 더 강조되어 건강한 개발 생태계가 구축되는 효과도 기대해 볼 수 있을 것이다.

2. Introduction

FOSSLight Hub 는 LG 전자에서 개발하여 출시한 웹 서비스로, 프로젝트별로 오픈 소스 소프트웨어(Open Source Software, OSS) 목록을 관리하며 각 OSS 에 대해 저작권, 라이선스, 보안 취약점 등의 정보를 파악 후 그에 걸맞은 적절한 행동을 취하고 SBOM(Software Bill Of Materials)을 생성하는 프로세스, 즉 OSC(Open Source Compliance) 프로세스를 도와주는 서비스이다.

그런데 사실 생각보다 상당수의 개발자에게 OSC 프로세스는 익숙지 않은 개념이다. 특별히 OSS 의 사용 규정에 관한 엄격한 내규가 있는 기업에서 근무한 경험이 없는 이상, 여러 라이브러리를 가져다 사용하며 개발한 경험만 있는 일반적인 개발자에게 OSC 프로세스는 매우 낯선 개념이다. 이해를 돕기 위해 OSC 프로세스를 한 줄로 요약하자면, "법적인 리스크를 최소화하며 각종 OSS 를 효과적으로 사용하기 위한 프로세스"이다. 프로젝트 담당자는 해당 프로젝트에 속한 OSS 의 목록을 정확히 파악하고, 각 OSS 의 저작권뿐만 아니라 각 OSS 에 걸린 라이선스들에서 정한 규칙들까지 반드시 준수하도록 심혈을 기울여 관리해야 한다. 라이선스도 여러 가지가 존재하는데, 그중에서 어떤 라이

선스는 반드시 소스 코드 또는 고지문을 공개해야 한다는 등의 '의무사항'이 있기도 하고, 상업적 이용 또는 소스 코드의 수정을 금지한다는 등의 '제한사항'이 있기도 하다. 현실적으로 작은 기업이나 개인의 경우에는 이러한 규칙들을 조금 준수하지 못해도 위험을 피해 갈 가능성이 있지만, 조금이라도 규모가 있는 사업에서는 법적인 제재를 당할 소지가 다분하기 때문에 이러한 규칙들을 면밀히 파악하고 준수해야 하고 소프트웨어 자재 명세서에 해당하는 SBOM 까지 제대로 생성해야 한다. 또한, 어떤 OSS 에는 몇몇 보안 취약점들이 발견되기도 하는데, 그런 정보들을 꼼꼼히 확인하지 않고 사용하는 경우 그에 따른 보안 위협에 노출될 가능성이 있다. 따라서 법적인 리스크를 떠나서 각 OSS 의 보안 취약점도 제대로 파악하여 사용에 주의를 기울여야 한다.

하지만 현실적으로 규모가 커지면 커질수록 해당 프로젝트에 속한 OSS 의 목록을 직접 파악하는 것은 매우 부담스러운 일이 되고, 각 OSS 의 저작권, 라이선스, 보안 취약점 등과 같은 정보를 일일이 파악하는 것은 훨씬 더 부담스러운 일이 된다. 매번 고지문이나 SBOM 을 직접 생성하는 것도 상당히 번거로울 것이며, 무엇보다 이러한 과정들이 어떤 플랫폼에서 체계적으로 이뤄지지 않는다면 관리 체계가 부실해질 가능성이 크다. 또한, 각 OSS 별로 주의해야 하는 보안 취약점을 하나하나 확인하는 것 또한 상당히 머리 아픈 일이라는 사실을 부정하는 사람은 없을 것이다. 이렇듯 OSC 프로세스는 추구하는 목적에 비해 현실에서 감당해야 하는 어려움이 크고, 이러한 이유로 많은 개발자가 OSC 프로세스를 제대로 인지하지 못하고 있는 것으로 보인다.

이러한 문제를 해결하기 위한 서비스가 바로 FOSSLight Hub 이다. 이는 관리자와 사용자의 역할을 구분한다. 관리자는 데이터베이스에 OSS 및 라이선스의 정보를 채우고, 프로젝트 단위로 OSC 프로세스를 철저하게 관리한다. OSS 의 목록은 FOSSLight Scanner 라는 패키지를 이용해 자동으로 분석할 수도 있고 직접 기입할 수도 있는데, 이러한 과정을 'Identification'이라고 부른다. 이 과정을 통해 각 OSS 의 저작권, 라이선스, 보안 취약점 등을 간편하게 파악하여 SBOM 을 생성할 수 있다. 다음으로 'Packaging' 과정에서는 앞서 기입한 OSS 중 소스 코드 공개 의무가 있는 것들을 자동으로 읽어와 보여주며 해당 소스 코드를 업로드할 수 있게 하고, 이를 바탕으로 고지문도 쉽게 생성할 수 있다. 반면에 관리자가 아닌 사용자는 직접적으로 데이터베이스에 OSS 및 라이선스의 정보를 채우거나 프로젝트를 관리하는 권한이 없다. 대신, Self-Check 프로젝트라는 것을 따로 만들어서, 앞서 설명한 프로젝트에서의 OSC 프로세스를 스스로 먼저 체크해 볼 수 있도록 몇몇 기능들을 제공한다.

그렇다면 기존의 FOSSLight Hub 를 활용할 때 발생할 수 있는 추가적인 요구사항이나 문제 상황으로는 무엇이 있을까. 다음 장부터는 기존의 FOSSLight Hub 를 둘러싼 여러 가지 문제 상황 및 달성 과제에 관해 이야기하고, 이어서 해당 달성 과제들을 어떤 방향으로 접근할지 면밀히 짚어본다. 그다음으로는 개발과 협업에 사용할 각종 기술 스택 및 툴과 전체적으로 디자인한 아키텍처에 관해 이야기하고, 실제로 개발을 진행하기에 앞서 설계한 스펙은 어떠한지 상세히 설명한다. 그리고 해당 스펙

에 따라 실제로 개발을 진행한 구현 항목과 이러한 개발에 따른 성취가 무엇인지 정리하고 결론과 함께 본 보고서를 마무리한다. 사용 가이드 문서는 부록에서 따로 제공한다.

3. Problems and Goal

기존의 FOSSLight Hub 에서 첫 번째로 문제가 되는 지점은 바로 CLI(Command Line Interface)의 부재이다. CLI 의 부재가 문제가 되는 이유는, CLI 가 제공하는 여러 장점을 얻을 수 없기 때문이다. 그렇다면 CLI 의 장점으로는 무엇이 있을까.

① 우선 가장 중요한 장점은 커맨드 라인에서 동일한 기능을 활용할 수 있는 프로그래밍 인터페이스가 제공된다는 점이다. 이는 사용자가 직접 웹으로만 활용할 수 있던 기능들을 프로그래밍할 수 있게 된다는 사실을 의미한다. 따라서 CI/CD 와 같은 자동화 시스템을 구축할 때 활용하면 OSC 프로세스에 수반되는 여러 종류의 휴먼 에러를 자동으로 감지할 수 있고, 필요시에는 CLI 를 활용한 병렬 작업 환경을 구축하는 등 더욱 효율적인 개인 작업 환경을 구축할 수도 있다. CLI 가 없다면 이러한 확장성과 유연성을 활용할 수 없다.

② 다음으로, 웹의 GUI 보다 터미널 환경이 더 익숙한 개발자에게 사용성의 개선을 가져다주는 이점이 있다. 실제로 FOSSLight Hub 의 사용자가 주로 특정 프로젝트를 진행하는 개발자라는 사실을 고려한다면 이는 실제로 더 큰 장점으로 다가올 수 있다.

③ 마지막으로, 웹을 기반으로 하지 않기 때문에 브라우저의 렌더링 메커니즘에 따른 성능 하락 요소가 없고, 브라우저의 버전이나 구현체에 영향을 받지 않으며, 웹 기반의 각종 해킹 공격(XSS, CSRF 등)에 안전해진다는 장점이 있다.

따라서 본 프로젝트의 첫 번째 달성 과제는 **FOSSLight Hub CLI** 를 개발하여 제공함으로써 위에서 언급한 CLI 의 여러 장점을 획득하는 것이다. 즉, 자동화 시스템 등에 활용할 수 있는 커맨드 라인 언어를 갖추고, 터미널이 편한 개발자들의 취향을 저격하고, 웹과 완전히 분리되어 더 경량화된 프로그래밍 인터페이스를 제공하는 것이 본 프로젝트의 첫 번째 목표라고 할 수 있다.

기존의 FOSSLight Hub 에서 두 번째로 문제가 되는 지점은 바로 기존 웹의 사용성에 있다. 구체적으로 기존 웹의 사용성에서 문제가 되는 부분들을 정리하자면 다음과 같다.

① FOSSLight Hub 는 관리자와 사용자의 역할이 구분되어 있는데, 두 역할을 모두 고려하여 설계된 나머지 대다수의 사용자에게는 전혀 필요 없는 기능들이 주를 이루고 있어 처음 방문하는 사용자의 입장에서는 해당 웹의 복잡도에 압도당하기 쉽다. 또한, 각종 워딩이 분명한 느낌을

주지 못하고 여러 기능이 곳곳에 중복으로 존재하는 Redundancy 문제가 있어 OSC 프로세스를 한 번에 이해하기란 쉽지 않아 보인다. 이는 곧 UX 의 문제라고 볼 수 있다.

② 다음으로, UI 자체도 충분히 문제가 되는 부분이다. 실제로 LG 전자에서도 UI 만 개선하여 새로운 웹을 출시하려고 했을 정도로, 전체적으로 매우 노후화되고 최근 트렌드에서 벗어난 구식의 디자인을 갖추고 있다. 그리고 무엇보다 강조하고 싶은 문제점은 모바일 환경을 전혀 고려하지 않았다는 점이다. 사실상 모바일은 배제한 UI 를 구축했기에, 반드시 PC 환경에서만 사용해야 한다는 치명적인 약점을 지니고 있다.

③ 마지막으로, 최근 트렌드에서는 사실상 거의 사용하지 않는 기술 스택(JSP)으로 웹이 구현되어 있어 유지보수가 쉽지 않고, 페이지 리로드가 없는 SPA(Single Page Application)를 구현한 듯 보이지만 사실 몇 번만 뒤로가기를 해보더라도 알 수 있듯이 네비게이션 동작이 제대로 구현되어 있지 않다. 별거 아닌 듯 보여도 뒤로가기가 제대로 동작하지 않는 문제는 사용자에게 순간적으로 굉장한 불편함을 제공할 수 있는 치명적인 단점이다.

따라서 본 프로젝트의 두 번째 달성 과제는 **User-friendly FOSSLight Hub** 를 추가로 개발하여 제공함으로써 위에서 언급한 기존 웹의 여러 문제점을 해소하는 것이다. 즉, 기존의 복잡하고 노후화된 UI & UX 를 벗어나 오로지 사용자에게만 필요한 기능 위주의 UI & UX 를 갖추고, 모바일 환경까지 지원하며, 앞으로도 유지보수하기 수월한 최신 기술 기반의 새로운 웹을 구축하는 것이 본 프로젝트의 두 번째 목표라고 할 수 있다.

4. Approach

본 프로젝트에서 앞서 소개한 두 개의 달성 과제를 진행하기 위해 첫 번째로 필요하다고 판단한 부분은 바로 REST API 를 정리하고 개선하는 작업이다. 기존의 FOSSLight Hub 서버에는 Open API 와 Closed API 가 구현되어 있는데, CLI 를 개발하기 위해서는 기존의 Open API 가 필요하고 User-friendly Hub 를 개발하기 위해서는 기존의 Closed API 가 필요하다. 하지만 두 종류의 API 모두 유지보수가 잘 되고 있지 않아 사용성이나 일관성이 현저히 무너져 있다. 따라서 각 소프트웨어의 개발에 앞서 다음과 같이 두 부류로 나눠 REST API 개발 작업을 진행하기로 한다.

① CLI 의 개발을 위해, 기존의 Open API 에 해당하는 v1 버전을 리뉴얼하여 v2 버전을 개발한다. 이는 각 API 가 RESTful 규칙을 지키고 파라미터/응답 형식의 일관성을 갖추도록 수정하고 내부 구현을 다듬는 작업뿐만 아니라 CLI 개발에 필요한 새로운 API 를 신설하는 작업까지 포함

한다. 그리고 이러한 v2 버전의 Open API 는 CLI 개발에만 사용되고 끝나지 않고 추후 다른 소프트웨어의 개발을 위한 재료로 활용될 수 있는 확장성까지 제공할 것이다.

② User-friendly Hub 의 개발을 위해, 기존의 Closed API 를 정리하고 확장한다. 이 또한 마찬가지로 각 API 가 RESTful 규칙을 지키고 파라미터/응답 형식의 일관성을 갖추도록 수정하고 내부 구현을 다듬는 작업뿐만 아니라 CLI 개발에 필요한 새로운 API 를 신설하는 작업까지 포함한다. 다만 이는 외부에 공개하는 Open API 가 아니므로 토큰 기반의 인증을 사용하는 Open API 와 달리 기존과 동일하게 세션 기반의 인증을 사용해야 한다.

다음으로, REST API 의 정리 및 개선 작업이 완료되었다는 가정하에 본 프로젝트에서 달성 과제로 삼았던 두 종류의 소프트웨어 개발을 병렬적으로 진행하도록 한다. 그중에서 먼저 CLI 의 개발과 관련하여 굵직하게 생각한 접근 방향은 다음과 같다.

① 사용성과 확장성을 고려하여 프로그래밍 인터페이스를 고안한다. CI/CD 와 같은 자동화 시스템에 활용될 것을 염두에 두고, 최대한 유연하게 다양한 기능을 활용할 수 있는 편리한 인터페이스를 설계해야 한다. 경직된 인터페이스를 제공한다면 스크립트 환경에서 사용할 수 있는 기능 또한 몇 가지로 제한될 것이다. 한편, 이를 위해서는 REST API 개발 담당자와 CLI 개발 담당자의 원활한 소통이 중요하므로 협업 과정에도 주의를 기울이도록 해야 한다.

② 자체적으로 Scanner 를 탑재한다. 실제로 FOSSLight Hub 를 이용할 때 OSS 의 목록은 일일이 기입하는 것보다 FOSSLight Scanner 라는 패키지를 이용하여 분석한 결과를 업로드하는 경우가 더 흔하다. 따라서 CLI 자체에 Scanner 를 탑재하여 이러한 과정을 자동화한다면 CLI 만으로도 통합적인 서비스 이용이 가능하여 사용성이 증폭될 것이다.

마지막으로, User-friendly Hub 의 개발에 있어 중요하게 생각한 구현 방향을 정리하자면 다음과 같다. 웹의 경우에는 절대적으로 개발 규모가 큰 편이라 긴 호흡으로 개발을 가져가야 하기에 이러한 방향성을 끝까지 놓치지 않고 챙겨주는 것이 중요하다.

① 직관적이고 세련된 UI & UX 를 기획한다. 관리자가 아닌 사용자가 사용하기 쉽도록 간편한 UI & UX 를 설계해야 하기에, 사용자에게 필요한 기능들만 남기되 각 기능이 중복으로 배치되지 않도록 주의를 기울이고, OSC 프로세스를 직관적으로 이해하기 쉽도록 전체적인 워딩 및 디자인을 구상한다. 또한, 모바일 환경까지 지원하는 최신 트렌드의 세련된 UI 를 디자인함과 동시에 최신 기술 스택으로 올바른 SPA 를 구현하여 사용성을 증폭시키도록 한다.

② 기존의 FOSSLight Hub 와 마찬가지로 Docker Compose 실행 환경을 제공한다. User-friendly Hub 를 별도로 설치할 필요 없이 FOSSLight Hub 만 설치하면 되도록 기존과 동일한 저장소에서 작업을 진행하고, FOSSLight Hub 의 실행 가이드에 명시되어 있듯이 'docker-compose up'

명령어만 수행해도 별도의 포트로 User-friendly Hub 가 실행되도록 환경을 구성한다. FOSS-Light Hub 는 자체적으로 호스팅을 할 수 있도록 Docker 환경을 제공하기에, 그 기조와 걸맞은 환경을 구성하는 것이 바람직할 것이다.

5. Development Environment

5-1. Technology Stacks

① FOSSLight Hub REST API (Backend)

백엔드는 기존의 FOSSLight Hub 처럼 Java 11 기반으로 Spring 프레임워크를 사용하고, 의존성은 Gradle 로 관리한다. 데이터베이스는 MariaDB 를 사용하며, Hikari Connection Pool 을 통해 Spring 과 연결한다. 한편, 수많은 의존성을 표현하기 위해 스키마가 상당히 복잡하게 설계되어 있기 때문에, 맵핑 목적으로는 (hibernate 등의 ORM 이 아닌) persistent 프레임워크인 iBATIS 를 사용한다. API 엔드 포인트의 문서를 자동 생성하기 위한 툴로는 Swagger 를 사용한다. CI/CD 는 GitHub Action 으로 관리되고 있는데, JUnit 으로 작성된 테스트 코드를 자동으로 실행하여 PR 을 확인하거나, 병합된 커밋의 코드를 이미지로 빌드하여 Docker Hub 에 푸시하는 역할을 수행한다. 이렇게 빌드된 이미지는 Docker Compose 에 의해 실행된다.

② FOSSLight Hub CLI

CLI 의 개발에는 생산성과 호환성이 좋고 대다수의 개발자에게 친숙하여 유지보수하기에도 편리한 Python 을 사용한다. 그리고 click 라이브러리를 사용하여 CLI 의 입력 파라미터와 명령어 스키마를 관리한다. 비슷한 라이브러리로는 구글에서 만든 fire, fire 와 비슷한 typer 등이 있는데, 이것들은 직관적이고 사용하기 쉽지만 help text, 명령어 그룹화 등의 세세한 부분을 제대로 지원하지 않아서 click 을 사용하기로 한다. click 은 데코레이터 기반으로 명령어와 입력 파라미터의 계층 구조를 관리하고, help text 와 같이 CLI 에 필요한 각종 기능을 간편하게 구현할 수 있도록 돕는다. 서버와 통신하기 위한 라이브러리로는 가장 흔히 사용되는 requests 를 사용하고, 모듈 간 데이터 전달에 사용할 DTO(Data Translation Object)를 정의할 때는 dataclasses 라이브러리를 사용한다. dataclasses 를 사용하면 직접 클래스를 작성하는 것보다 더 짧게 코드를 작성할 수 있고, 타입 명시를 통해 직관성을 높일 수 있다.

③ User-friendly FOSSLight Hub

기존의 FOSSLight Hub 는 JSP 를 기반으로 웹이 구현되어 있는데 현재는 해당 기술 스택을 다루는 개발자가 많지 않아 앞으로 유지보수하기 어렵고, SPA 구현이 잘 되어 있지 않아 네비게이션도 올바르게 동작하지 않는다. 따라서 본 프로젝트에서 User-friendly Hub 를 개발할 때는 프론트엔드에서 압도적인 점유율을 지니는 React 프레임워크를 사용한다(조금 더 구체적으로는 Next 13). 이는 점유율이 높은 만큼 유지보수하기 좋고 개발이나 디버깅과 관련한 이슈를 찾기에도 좋다. 또한, 정적 타입 시스템을 통해 코드의 안정성을 확보하기 위해 TypeScript 를 사용한다. 전역 상태 관리 라이브러리로는 필요한 기능만 최소한으로 담고 있고 사용 방법이 매우 직관적인 recoil 을 사용하여 생산성을 높이고, CSS 파일을 따로 두지 않고 유틸리티 클래스로 디자인하는 tailwindcss 라이브러리를 사용하여 생산성을 높임과 동시에 프로젝트의 파일 구조를 간소화한다. 클라이언트 사이드의 API 호출부에서는 axios 라이브러리를 기반으로 react-query 라는 라이브러리를 사용하여 캐싱, 새로고침, 로딩, 에러 처리 등을 간편하게 구현하고, 단순한 입력을 구현할 때는 React 자체의 지역 상태로 직접 구현하되 복잡한 입력을 구현할 때는 입력과 관련된 여러 기능을 이미 구현하여 추상화해 둔 react-hook-form 라이브러리를 활용한다. 그밖에 날짜와 관련된 데이터를 처리하는 dayjs 라이브러리, 모바일 환경 감지를 위한 react-responsive 라이브러리 등과 같이 개발하는 과정에 필요한 사소한 라이브러리들은 대중성, 안정성, 사용성에 기반하여 그때그때 선택하여 사용한다. 마지막으로, User-friendly Hub 또한 기존 FOSSLight Hub 와 마찬가지로 동일한 Docker 환경 안에서 빌드되도록 하되 포트만 다르게 설정한다.

5-2. Collaborative Tools

본 프로젝트의 구성원은 총 3 명이며 REST API, CLI, User-friendly Hub 의 개발을 하나씩 나눠서 담당한다(10. Division of Work 참고). 하지만 세 과제가 완전히 독립적이지 않고, 특히나 REST API 를 중심으로 소통해야 하는 일이 잦고, 요구사항에 관한 논의를 위해 LG 전자의 담당자님과 소통해야 하는 경우도 있다. 따라서 구성원끼리의 기본적인 소통은 카카오톡이나 구두로 진행하되, 공식적인 내용의 논의나 LG 전자 담당자님과의 소통이 필요한 경우에는 미리 개설해둔 공동의 Slack 채널을 활용하여 그곳에서 의사소통을 진행하기로 한다.

또한, 버전 관리 시스템으로는 Git 을 사용하고, 효율적인 협업과 작업의 투명성을 확보하기 위해 GitHub 을 사용한다. 각 작업은 GitHub 의 이슈(Issue) 기능으로 관리하였고, 대기업에 제공하는 서비스인 만큼 각 작업의 기록을 체계적인 형태로 남기고자 다음과 같이 최소한의 Git 사용 규칙까지 합의한다.

① Repository

A. FOSSLight Hub CLI: LG 전자에서 새로 생성한 'fosslight_cli' 저장소에서 작업한다.

B. User-friendly FOSSLight Hub: 기존 FOSSLight Hub 의 저장소('fosslight')를 Fork 하고, 루트 디렉터리에 '/lite' 디렉터리를 생성하여 그 안에서 작업한다.

② Branch

A. main 브랜치에서 snu 브랜치를 생성한다.

B. snu 브랜치에서 각자의 feature 브랜치를 생성하여 작업 후 snu 브랜치로 병합한다.

C. feature 브랜치의 이름은 다음과 같은 규칙으로 짓는다.

EX 1) cdg/feature/#12#34 → 최덕경 담당 작업, #12 이슈와 #34 이슈
EX 2) kse/feature/#56 → 김성은 담당 작업, #56 이슈
EX 3) any/feature/#78 → 공동 작업, #78 이슈

D. 최종적으로 개발이 마무리되면 snu 브랜치를 main 브랜치로 병합한다.

③ Commit

A. 커밋 메시지는 영어로 작성하며, 대문자와 동사로 시작한다.

B. 커밋 메시지의 맨 앞에는 태그(Add, Remove, Improve, Fix 중 하나)를 붙인다.

C. (필요시) 상세 내용은 개행을 두 번 하고 '-' 기호를 사용해 리스트 형식으로 작성한다.

④ Pull Request (PR)

A. 커밋 태그와 동일하게 라벨(Add, Remove, Improve, Fix 중 하나)을 붙인다.

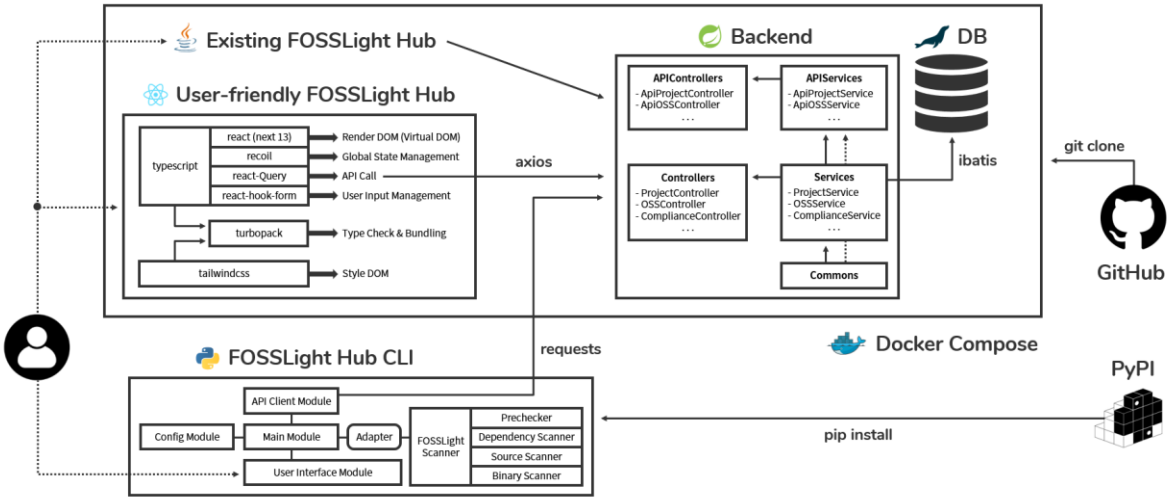
B. 제목은 커밋 메시지와 규칙이 동일하되, 맨 끝에 이슈 번호를 붙인다.

C. (필요시) 상세 내용은 영어로 작성한다.

D. 병합 커밋 메시지는 디폴트 값을 그대로 사용한다.

6. Architecture

6-1. Architecture Diagram



6-2. Architecture Description

① FOSSLight REST API (Backend)

Spring 백엔드에서는 Service 와 Controller 가 비즈니스 로직의 큰 축을 담당한다. Controller 에서는 API 를 구현하는데, 크게 세 부류로 나뉜다. 하나는 기존의 FOSSLight Hub 에서 사용하는 Closed API 를 위한 Controller 이고, 다른 하나는 Open API 를 위한 ApiController, 마지막 하나는 User-friendly Hub 에서 사용하는 Closed API 를 위한 LiteController 이다. 예를 들어, 기존 FOSSLight Hub 에서 OSS 의 목록을 불러오는 API 는 OssController 에서 구현되지만, Open API 는 ApiOssController 에서 구현되며 User-friendly Hub 에서 사용하는 Closed API 는 LiteOssController 에서 구현된다. Controller 에서 비즈니스 로직의 실행을 위해 호출하는 메서드들은 Service 에서 구현되는데, 이는 Service 와 ApiService 로 나뉜다. 다만 Service 중에는 파일의 업로드를 담당하는 FileService, 사용자의 정보를 가져오는 T2UserService 등도 있는데, 이들은 Service 와 ApiService 의 구분 없이 공통으로 사용된다. Service 는 iBATIS 를 통해 MariaDB 데이터베이스로부터 정보를 가져오며, 필요한 SQL 쿼리들은 Mapper 파일(.xml)에서 구현된다. 이러한 SQL 쿼리들은 iBATIS 를 통해 Repository 에 정의된 인터페이스들의 메서드들에 맵핑되며, Service 에서 호출된다.

② FOSSLight Hub CLI

먼저, Main 모듈과 User Interface 모듈은 명령어 스키마를 정의하고, 사용자의 입력을 분석하여 해당 명령어에 대응하는 함수가 호출되도록 한다. 각 함수는 다른 모듈(API Client 모듈, Config 모듈, Adapter 모듈)을 호출하여 사용자의 요청을 처리한다. 그리고 API Client 모듈은 서버와의 통신을 담당하는 모듈로서 API 스키마를 관리하고 서버로의 API 호출을 담당하며, Config 모듈은 서버 URL, 토큰 값 등의 각종 설정을 관리하는 모듈로서 파일을 대상으로 설정값을 읽고 쓰는 역할을 담당한다. 마지막으로, Adapter 모듈은 FOSSLight Scanner 를 연동하기 위한 모듈로서 프로젝트 디렉터리의 경로와 Scanning 옵션을 입력 파라미터로 받아서 적절한 동작을 수행한다.

③ User-friendly FOSSLight Hub

React 개발을 목적으로 JavaScript 의 문법을 확장하여 정의한 JSX 문법에 TypeScript 의 정적 타입 시스템을 첨가한 TSX 문법으로 코드를 작성하면, Next 13 의 turbopack 번들러에 의해 타입 체크와 번들링이 수행되어 브라우저에서 실행할 수 있는 정적 파일들이 생성된다. 순수 React 는 자체 서버가 따로 없어 번들링된 정적 파일들을 웹 서버가 참조할 수 있는 특정 경로에 배치하는 것이 전부지만, 본 프로젝트에서는 React 프레임워크를 래핑하여 서버 사이드의 기능을 추가로 지원하는 Next 프레임워크를 사용하기 때문에 번들링된 정적 파일들을 직접 제공하는 자체 서버가 존재한다. 이때 TSX 문법으로 작성되는 각 컴포넌트에서는 tailwindcss 의 유틸리티 클래스를 이용하여 렌더링되는 DOM 을 디자인한다. 브라우저가 React 를 기반으로 번들링된 정적 파일들을 실행하면 메모리에만 상주하는 Virtual DOM 을 기반으로 DOM 이 화면에 렌더링된다. 사용자의 입력은 React 자체가 제공하는 Hook 기반의 지역 상태 또는 react-hook-form 라이브러리를 통해 구현하고, 여러 컴포넌트가 공유해야 하는 뷰포트, 로딩 중 등의 상태는 recoil 라이브러리의 전역 상태 관리 메커니즘으로 구현한다. 그리고 API 호출은 axios 와 react-query 라이브러리를 활용하여 구현하되, 자주 사용되는 패턴을 고려하여 useAPI()라는 Custom Hook 을 직접 정의하여 사용하도록 한다.

7. Specifications

7-1. FOSSLight Hub REST API (Backend)

① Modules

공통 모듈들은 oss.fosslight.*에 정의되어 있고, Open API, User-friendly Hub 를 위한 API 및 해당 부분에서만 의존성을 가지는 모듈들은 oss.fosslight.api.*에서 구현된다. 사용자에게 공개된 엔드 포인트들을 구현한 Controller 들은 oss.fosslight.api.controller 에서 정의되며, 기존의 FOSSLight Hub 를 위한 API, Open API, User-friendly Hub 를 위한 API 의 엔드 포인트들이 정의된다. Open API 들은 oss.fosslight.api.controller.v1, oss.fosslight.api.controller.v2 에서 구현되며, User-friendly Hub 에서 쓰는 API 는 oss.fosslight.api.controller.lite 에서 구현된다. 해당 Controller 들에서는 파라미터의 유효성 검사 및 에러 처리 로직이 정의된다. 유효성 검사와 에러 처리를 거친 후, Controller 은 요구되는 로직에 따라 Service 에서 구현된 메서드를 호출한다. Service 는 앞서 언급했듯 엔드 포인트에 따라 Service 와 ApiService 로 나뉘며, 그 안에서 다시 데이터에 따라 OssService, VulnerabilityService 등으로 나뉜다. Service 에서는 실제 비즈니스 로직의 실행과 iBATIS 로 맵핑된 메서드의 호출이 이뤄진다.

② Database

데이터베이스에는 OSS, 라이선스, 보안 취약점, 프로젝트, 사용자 등과 관련된 스키마가 있다. 주요 정보들은 PROJECT_MASTER, OSS_MASTER 등 *_MASTER 에 정의되어 있으며, 세부 정보, 1:N 이나 N:N 관계를 가지는 정보들은 PROJECT_WATCHER, OSS_COMPONENTS 등의 이름을 가진 테이블에 저장된다. 예를 들어, PROJECT_MASTER 테이블은 PROJECT_MODEL, PROJECT_WATCHER, PROJECT_PARTNER_MAP 등이 참조하고 있어 3rd Party 프로그램, 프로젝트의 Watcher 등을 관리하게 된다. 이러한 테이블에 저장된 정보가 코드 내에서 사용될 때는 oss.fosslight.domain 에 구현되어 있는 DTO, Bean 들을 사용한다. 이곳에 구현되어 있는 자료구조들은 주로 iBATIS Mapper 의 출력 형식으로 사용되며, 일부는 iBATIS 쿼리의 입력 Map 을 정의하는 데 쓰이기도 한다. 기존 코드에서 몇몇 DTO 의 경우, 쿼리의 입력 기능과 API 의 출력을 표현하는 기능을 동시에 맡기도 한다. 그러나 이러한 경우 쿼리에서만 사용되는 플래그의 값들마저 출력의 데이터 타입에 포함되는 문제가 있기 때문에, 새로 구현하는 코드에서는 프론트엔드에서 요구하는 자료구조에 맞춰서 출력용 DTO 를 새롭게 정의한다. NVD_DATA 에는 National Vulnerability Database 의 API 에서 갱신되는 보안 취약점 정보를 ScheduleWorkerTask 의 cronjob 으로 sync 한다. NVD API 에서 제공하는 정보들은 product 열을 통해 OSS 를 참조하도록 하여, 프로젝트와 OSS 의 보안 취약점을 확인하는 용도로 사용한다. VulnerabilityMap-

per.xml 파일에서는 이러한 관계를 이용하여 쿼리의 대상이 되는 OSS 의 정보로 관련된 보안 취약점을 가져온다.

③ Security

보안의 경우 호출하는 대상에 따라 프론트엔드에서 호출하는 경우 세션 토큰을, Open API 의 경우 헤더 토큰을 이용해서 인증을 진행한다. 프론트엔드에서 호출하는 API 는 "/api/lite/*", Open API 는 "/api/v1, /api/v2"로, URI 로 구분 가능하므로 인증의 방식 또한 엔드 포인트의 URI 에 따라 달라지게 구현한다. JSP 프론트엔드와 User-Friendly Hub 는 Origin 또한 다르므로, out-of-the-box 로 지원되는 WebSecurityConfigurerAdapter 에서는 로그인 API 과 세션 토큰을 위한 로직만 구현하고, User-Friendly Hub 에서 CORS 관련 로직과 인증 관련 로직은 servlet filter 로 구현한다. 반면, 사용자에게 공개되는 Open API 는 앞서 서술한 WebSecurityConfigurerAdapter 의 기능을 통해 헤더에 포함된 세션 토큰을 사용하여 인증을 진행하도록 한다. 해당 코드들은 CorsFilter.java 에서 구현한다.

④ Controllers

새로 구현되는 API 중, v2 API 의 경우 기존 v1 에서 지원한 API 의 기능들, 그리고 CLI 에서 필요로 하는 기능들을 위주로 구현한다. 기존에 지원하던 기능들은 크게 OSS, 프로젝트, 3rd Party, 보안 취약점의 조회와 등록이 있으며, 그 외에도 라이선스의 조회, OSS 의 보안 취약점 조회 등의 기능이 있다. 기존에 존재하는 기능들의 경우 엔드 포인트들에서 공통으로 사용하는 형식이 REST API 의 Best Practice 에서 먼 부분, 반환 값이 HTTP 표준과 맞지 않는 부분이 있었기에 이런 부분들을 개선하는 데에 초점을 맞추기로 한다. 반환 값이 기존에서 공통으로 사용하던 로직과 다른 부분이 있었기에 v2 Controller 들에서 사용하는 Service 인 RestResponseService 을 구현한다. 내부적으로 정의된 에러 메시지를 통해 HTTP 응답을 빌드해주는 기능 또한 포함하며, 이는 User-friendly Hub 의 ResponseEntity 생성을 위해서도 사용된다. 또한, 스프링 자체적인 에러 처리 동작에서 v1 과 v2 를 분리하기 위해, v1 에서 공통으로 에러 처리를 담당하던 ResponseEntityExceptionHandler 대신 v2 전용 핸들러를 구현한 뒤 v2 API 의 Controller 에서 에러가 날 경우 이 핸들러를 먼저 사용하고, 기존 핸들러는 Fallback 으로 사용되게 한다. 해당 부분은 APIV2ExceptionAdvice 에서 구현된다. User-friendly Hub 전용 API 의 경우, 새로운 화면에서 필요로 하는 목록 조회 API 와 검색 API 를 구현한다. 구현한 기능은 크게 OSS, 라이선스, 보안 취약점의 검색과 조회, 대시보드 정보 조회, Self-Check 프로젝트 조회, 그리고 전체 검색 기능이 있다. 목록 조회 기능에서는 기본적으로 페이징과 각 열에 대한 정렬 기능을 구현하되 정렬 기준과 열을 추가하고, 엑셀 파일을 다운로드하기 위한 API 까지 구현한다. 또한, 검색 자동완성 등의 편의성 기능을 위한 간소화된 API 또한 부가적으로 구현한다.

⑤ Services

Service 들은 앞서 언급한 API 엔드 포인트인 Controller 에서 비즈니스 로직의 실행을 위한 Service 들이 있으며, 그 외에 공통적인 로직을 실행하는 Service 들이 있다. 공통적으로 사용되는 로직을 실행하는 Service 들의 예시로는 다음과 같은 것들이 있다.

(1) T2UserService: 사용자의 역할(Role) 확인 및 수정, 인증 담당.
(2) FileService: FOSSLight Scanner 출력값인 XML 파일 등의 파일 업로드, 조회 및 관리.
(3) ResponseService: Service 출력값으로 JSP 프론트엔드에 보내는 Result entity 생성. (여기서 사용되는 Result entity 는 자체적으로 정의된 vo)
(4) RestResponseService: Service 출력값 또는 HTTP 상태 코드로 ResponseEntity 생성.
(5) MailService: 사용자에게 메일 전송. (EX. User-friendly Hub 에서 잘못된 라이선스 정보가 포함되어 있어 고지문을 생성할 수 없을 때)
(6) SystemConfigurationService: 시스템의 설정 관리.

또한 User-friendly Hub 에서 사용하기 위한 ApiVerificationService 와 같은 새로운 ApiService 들 또한 정의한다.

⑥ Models

기존의 JSP 프론트엔드에 대응하는 API 와 비교했을 때 입력 형식이 바뀌었으므로, 기존 입력/출력의 통일된, 즉 ResponseEntity 와 @ModelAttribute 에 같은 DTO 를 사용하던 방식 대신 개별적인 자료구조를 사용하여 구현한다. 변수의 경우 프론트엔드에서 자연스러운 포맷/자료형이 SQL 쿼리에서 다른 경우가 많으므로, 이러한 부분은 DTO 내부의 setter 에 구현하여 @ModelAttribute 에 할당될 때 자동으로 변환하도록 한다. 예를 들어, User-friendly Hub 에서 OSS 목록을 불러올 때는 oss.fosslight.api.dto 의 ListOssDto.Request 를 입력으로 주고, ListOssDto.Result 의 형태로 결과를 받도록 한다.

7-2. FOSSLight Hub CLI

① Modules

API Client 모듈, Config 모듈, Adapter 모듈은 각각 client.py, config.py, scanner.py 파일에서 작성된다. 또한, Main, User Interface 모듈은 main.py, commands/*.py 에 구현되어 있고, 명령어 단위로 파일을 분리한다. 예를 들어, commands/create.py 파일에서는 create project,

create selfcheck 등의 명령어를 정의하고, commands/search.py 파일에서는 search project, search oss 등의 명령어를 정의한다.

② Flow

사용자가 명령어를 입력하면, main 함수와 commands/*.py 에 정의된 데코레이터들에 의해 명령어가 분석되고 라우팅되어 사용자가 입력한 명령어에 해당하는, commands/*.py 파일에 정의된 적절한 함수가 호출되어 명령어를 처리한다. 한편, 각 함수에서는 다른 모듈들을 적절히 사용하여 명령어를 처리한다. 예를 들어 create project 커맨드를 입력한 경우, commands/*.py 파일에 정의된 create_project 함수가 호출되고 create_project 함수는 Config 모듈에서 서버 정보를 읽어와서 API Client 모듈에 입력하고, API Client 모듈을 이용하여 프로젝트 생성 API 를 호출한다. create_project 함수는 API 호출 결과를 받아서 적절히 분석하고, 사용자에게 실행 결과를 보여준다.

③ Scheme

```
fosslight-cli [command] [resource name] ([sub-resource-name]) [parameters ...]
```

ⓐ command: 수행하려는 동작을 지정한다.

- create
- update
- get
- export
- apply
- compare

ⓑ resource-name: 리소스 이름을 지정한다.

- project
- selfCheck
- config
- partner
- oss
- license
- vulnerability
- maxVulnerability
- yaml

ⓒ sub-resource-name: (일부 명령에서 필요) 하위 리소스의 이름을 지정한다.

- EX 1) fosslight-cli get project list
- EX 2) fosslight-cli update project bin
- EX 3) fosslight-cli get project models

ⓓ parameters: 입력 파라미터들을 지정한다. (필수 or 선택)

④ Advanced Commands

프로젝트의 새로운 버전을 배포할 때마다 웹에서 프로젝트를 생성하고, Scanner 를 이용하여 소스 코드를 스캔하고, 스캔한 결과를 업로드한 뒤 BOM 파일을 다운로드하는 Flow 를 진행한다. 이 과정을 쉽게 자동화할 수 있도록, 그리고 각 프로젝트 관리를 웹에 굳이 들어가지 않고 프로젝트에 파일 하나를 생성해서 진행할 수 있도록 아래의 고도화 명령어를 추가하였다.

ⓐ Scan & Update: Scanner 의 실행 시간이 30 초 이상 소요되기 때문에 스캔이 끝나는 것을 사용자가 기다렸다가 업로드하는 것은 사용성이 좋지 않다. 이 과정을 쉽게 만들어주는 명령어는 다음과 같다. 해당 디렉토리를 스캔하고 결과까지 업로드해주기 때문에 원래처럼 기다릴 필요가 없어진다.

```
fosslight-cli update project scan -prjId 3 -dir "path/to/project"
```

ⓑ Apply: 여기서 더 나아가 프로젝트 생성부터 스캔, 스캔 결과 업로드를 한 번에 처리해주는 명령어다. 이 Flow 에서만 사용 가능한 명령어는 아니고, 다른 Flow 들도 자동화할 수 있도록 확장 가능한 개념으로 설계하였다.

```yaml
# create-project.yaml
kind: createProject
parameters:
  prjName: exampleProject
  prjVersion: ${PRJ_VERSION}
  osType: Linux
  distributionType: "General Model"
  networkServerType: N
  priority: P1
update:
  modelFile:
    modelReport: "ModelList.xlsx"
scan:
  dir: "../"
```

위의 YAML 파일을 각 프로젝트마다 관리하고, 새로운 프로젝트 버전이 나올 때마다 `fosslight-cli apply create-project.yaml` 명령어를 실행하면 생성부터 분석 파일 업로드까지 진행된다.

kind 값이 어떤 동작과 Flow 를 수행할 것인지를 결정한다. 필요하다면 export 까지도 진행하도록 추가할 수 있고, 혹은 스크립트를 따로 작성해서 `fosslight-cli export project ...` 명령어를 뒤이어서 호출해줘도 된다.

7-3. User-friendly FOSSLight Hub

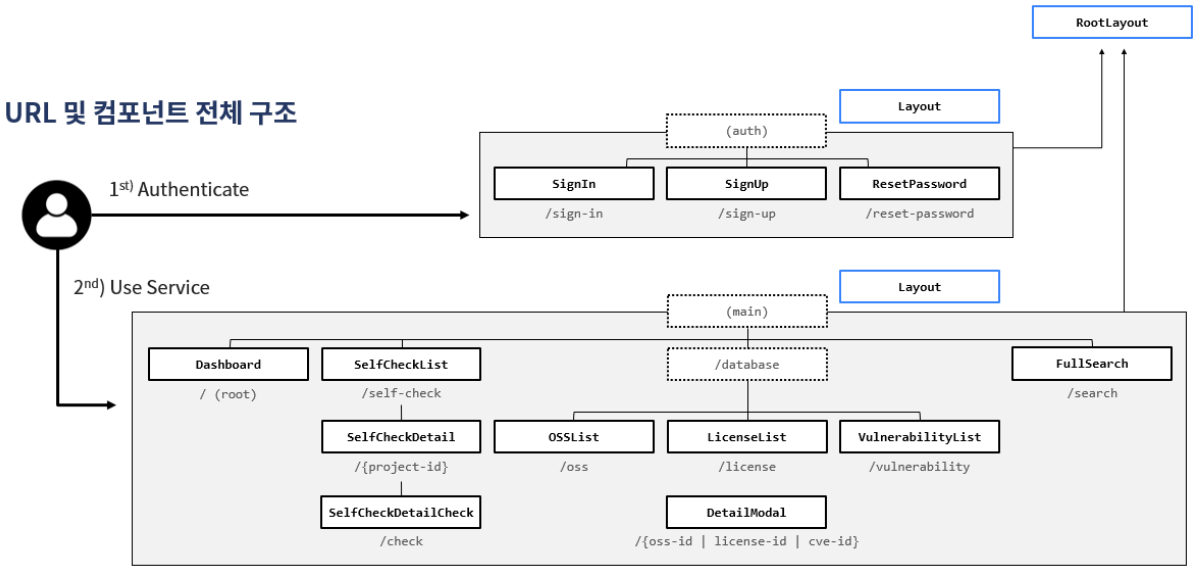
User-friendly Hub 를 개발할 때 가장 중요한 부분은 사용자에게 필요한 기능을 중심으로 복잡도를 낮추고 직관성을 높이는 인터페이스를 설계하는 것이다. LG 전자 측에서 전달한 공식적인 요구사항의 골자는 다음과 같이 두 가지이다. 하나는 본인의 프로젝트에 속한 OSS 의 목록을 기입하여 각 OSS 및 라이선스의 정보를 빠르게 파악하고 필요한 고지문을 쉽게 생성할 수 있도록 하는 Self-Check 프로젝트 페이지이고, 다른 하나는 필요한 정보를 쉽고 빠르게 얻을 수 있도록 기획된 OSS, 라이선스, 보안 취약점의 정보 검색 기능이다.

여기서 Self-Check 프로젝트 페이지의 경우 몇 가지 추가 요구사항을 전달 주셨는데, 기존에는 없던 Packaging 과정을 추가하는 것(이를 위해서는 업로드된 Package 파일을 저장하기 위한 필드 선언이 필요), 목록 페이지에서 고지문을 바로 다운로드할 수 있도록 하는 것, 그리고 유효하지 않은 라이선스가 기입된 경우 고지문 미리보기/다운로드를 막고 관리자에게 이메일을 보낼 수 있도록 하는 것이 바로 그것에 해당한다.

이 외에도 좋은 아이디어가 있다면 자율적으로 기능을 추가해도 된다고 하셨기에, 우리는 대시보드 페이지와 전체 검색 기능을 추가로 떠올렸고 더불어 모바일 사용성의 확보를 위해 반응형 디자인까지 진행하기로 한다.

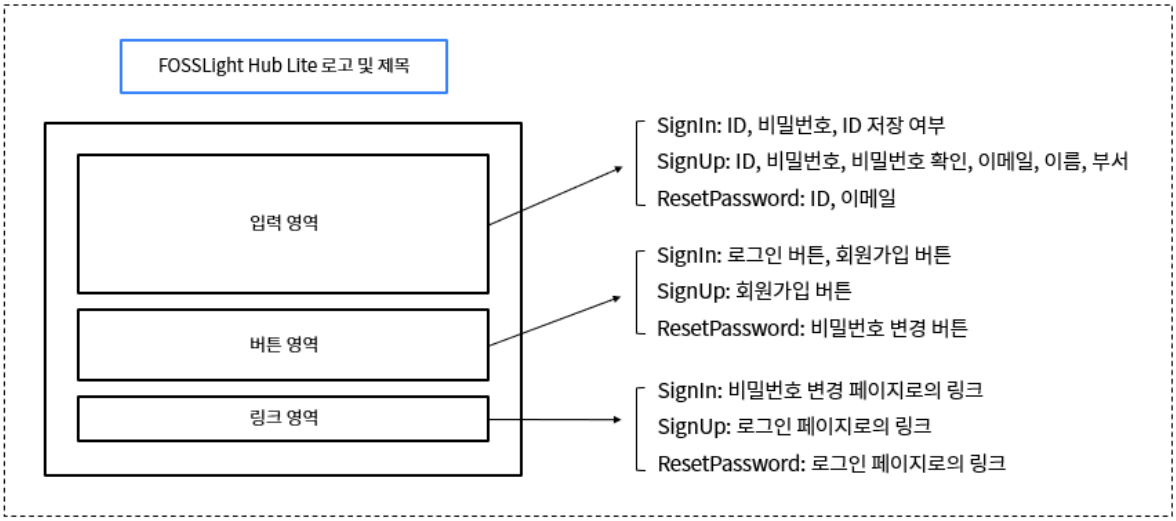
구체적으로 User-friendly Hub 의 개발에 사용된 스펙을 정리하면 다음과 같다.

① URL 및 컴포넌트 전체 구조



Next 프레임워크는 URL 구조를 파일 시스템의 구조와 거의 비슷하게 구축할 수 있는 시스템을 제공하기 때문에, 이에 근거하여 URL 및 컴포넌트의 전체 구조를 위와 같이 설계한다. (auth)는 인증 관련 페이지들을 가상으로 묶는 디렉터리, (main)은 나머지 페이지들을 가상으로 묶는 디렉터리이다. 이때 가상이라 함은 디렉터리는 존재하지만 실제로 그 디렉터리의 이름이 URL 에 맵핑되지는 않는다는 뜻이다. 점선은 디렉터리를, 실선은 컴포넌트를 의미한다. 각 컴포넌트에 대한 자세한 스펙은 이어지는 설명에서 다룬다.

② (auth) Layout / SignIn, SignUp, ResetPassword 컴포넌트 상세 구조

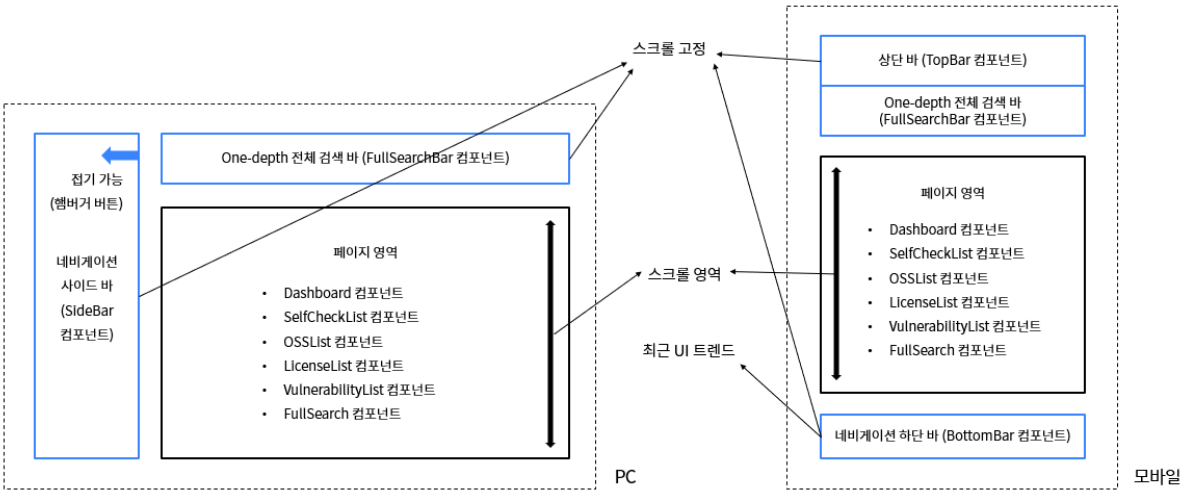


■ Layout 컴포넌트의 구현부 (페이지 공통)

SignIn, SignUp, ResetPassword 는 각각 로그인, 회원가입, 비밀번호 변경 페이지에 해당하는 컴포넌트이고, 하늘색 영역에 해당하는 공통 디자인은 Layout 컴포넌트에서 구현한다. 페이지

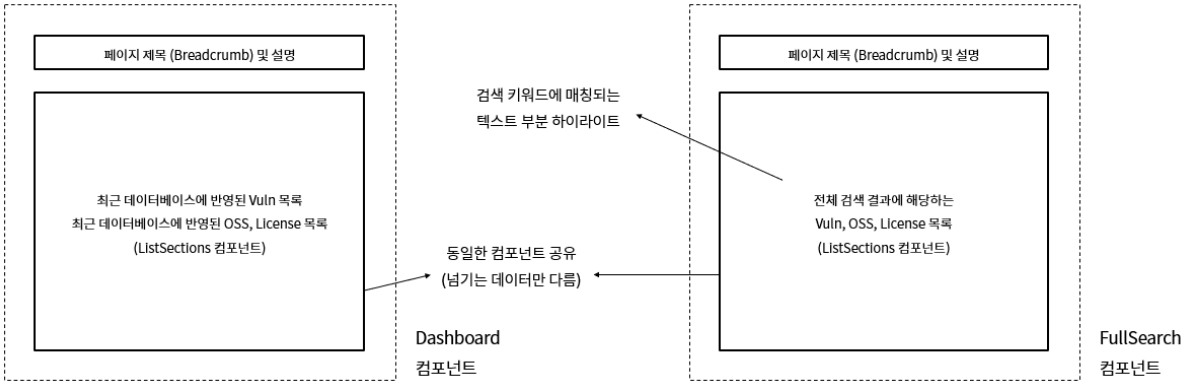
별로 그 역할에 맞는 입력, 버튼, 그리고 다른 페이지로의 링크가 조금씩 다를 뿐(위 그림 참조) 구조나 디자인은 사실상 거의 동일하다.

③ (main) Layout 컴포넌트 상세 구조



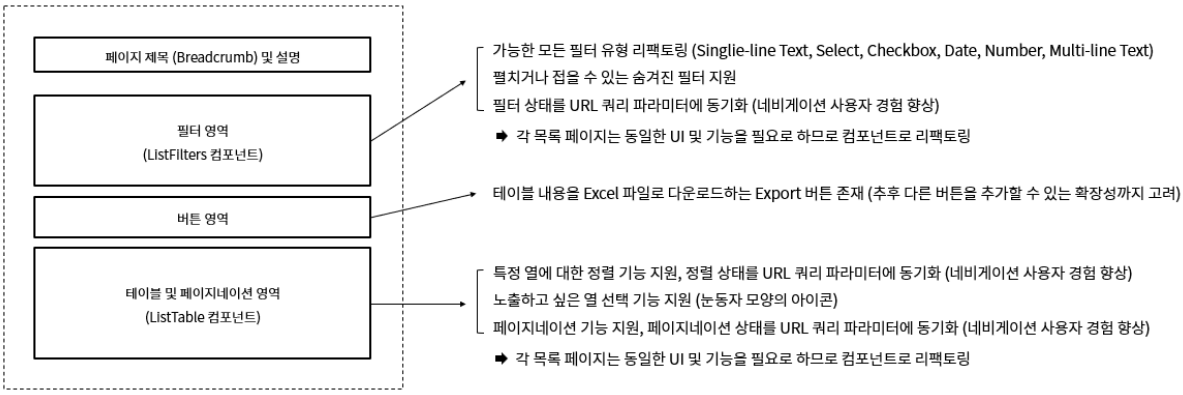
Layout 컴포넌트는 로그인, 회원가입, 비밀번호 변경 페이지를 제외한 나머지 모든 페이지는 공통 디자인 및 로직(하늘색 영역)을 구현하며, 모바일 사용성도 확보하기 위해 PC/모바일 환경을 구분한다. PC 환경에서는 햄버거 버튼을 통해 접거나 펼칠 수 있는 네비게이션 전용 사이드 바가 좌측에 있고, One-depth 검색을 위한 전체 검색 바가 상단에 있다. 둘 다 스크롤과 무관하게 위치가 고정되고, 나머지 영역은 스크롤 되는 페이지의 영역이다. 그리고 모바일 환경에서는 마찬가지로 스크롤과 무관하게 위치가 고정되어 있는 상단 바, 전체 검색 바, 하단 바가 있고, 나머지 영역은 스크롤 되는 페이지의 영역이다. 상단 바에는 웹사이트의 로고 및 제목, 프로필 정보를 확인할 수 있는 아이콘이 있고, 전체 검색 바는 PC 환경과 동일하며, 하단 바는 소메뉴들을 펼쳐볼 수 있는 대메뉴로 구성된 네비게이션 바이다. 하단 바는 최근 모바일 애플리케이션 시장에서 가장 트렌디하게 사용되는 UI 이다.

④ (main) Dashboard, FullSearch 컴포넌트 상세 구조



대시보드 페이지에서는 최근에 데이터베이스에 반영(추가 또는 수정)된 보안 취약점, OSS, 라이선스를 한 번에 확인할 수 있도록 한다. 그리고 전체 검색 결과 페이지도 대시보드 페이지와 거의 동일한 UI 로 보안 취약점, OSS, 라이선스들을 표시하도록 한다. 이는 개발 시 컴포넌트를 재활용할 수 있을 뿐 아니라 일관된 UI 가 사용자에게 직관적인 사용성을 제공하기 때문이다. 또한, 전체 검색 결과 페이지에서는 검색 키워드에 해당하는 부분을 노란색으로 하이라이트 처리한다.

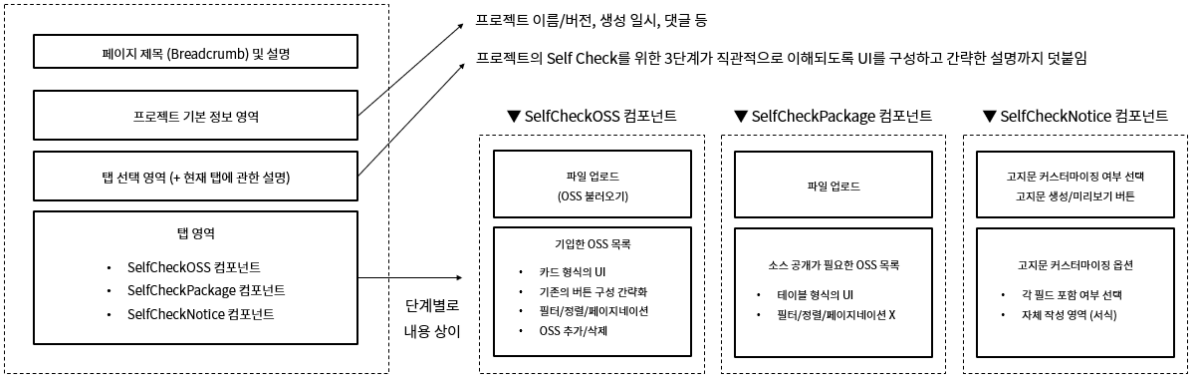
⑤ (main) SelfCheckList, OSSList, LicenseList, VulnerabilityList 컴포넌트 상세 구조



Self-Check 프로젝트, OSS, 라이선스, 보안 취약점의 목록 페이지는 동일한 UI 와 기능을 갖도록 설계함으로써 재활용 가능한 컴포넌트들을 정의하여 생산성과 유지보수성을 높일 수 있게 한다. 이 또한 실제로 각 목록 페이지가 UI 나 기능에 있어 일관성이 없다면 사용자에게 혼란을 줄 수 있다는 점을 고려한 것이다. ListFilters 컴포넌트는 검색 필터 영역을 구현하며, 필터의 상태를 URL 쿼리 파라미터에 동기화하여 새로고침이나 뒤로가기 등의 경우에도 적용된 필터 정보가 사라지지 않도록 한다. 그리고 ListTable 컴포넌트는 테이블 및 페이지네이션 영역을 구현하며, 정렬 및 페이지네이션 기능을 지원하도록 한다. 이때 정렬과 페이지네이션의 상태 또한

URL 쿼리 파라미터에 동기화하여 마찬가지의 목적을 달성한다. 테이블의 데이터는 필터, 정렬, 페이지네이션의 상태가 변할 때마다 새로 불러오도록 한다.

⑥ (main) SelfCheckDetail 컴포넌트 상세 구조



Self-Check 를 목적으로 만들어진 프로젝트의 상세 정보를 조회할 수 있는 페이지로, 프로젝트의 기본 정보를 확인할 수 있을 뿐 아니라 Self-Check 를 위한 3 단계를 직관적인 UI 의 탭으로 구성하여 각 단계에서 사용자가 진행해야 하는 작업을 분명히 한다. 각 탭의 영역은 별도의 컴포넌트로 구현한다. OSS 탭의 영역에서는 OSS 목록의 간편 기입을 위한 파일 업로드 기능을 지원하고 지금까지 기입한 모든 OSS 의 목록을 표시한다. 이때 OSS 의 목록은 직관적인 UI 를 위해 카드 형식으로 구성하며, 간단한 필터, 정렬, 페이지네이션 기능과 개별 OSS 추가/삭제 기능까지 구현하도록 한다. Package 탭의 영역에서는 소스 공개가 필요한 OSS 의 목록을 표시하고, 그것들에 해당하는 소스 파일을 업로드할 수 있도록 한다. 마지막으로, Notice 탭의 영역에서는 사용자가 고지문 커스터마이징 여부를 선택하고 그것에 따라 원하는 고지문을 생성하거나 미리보기할 수 있도록 한다. 사용자는 이러한 Self-Check 의 3 단계를 차례대로 진행함으로써 현재 진행 중인 프로젝트에서 OSC 관련 이슈를 간편하게 파악할 수 있다.

8. Implementation (Solutions)

8-1. FOSSLight Hub REST API

백엔드 측에서는 크게 Open API, 그리고 User-friendly Hub 두 종류의 엔드 포인트를 구현해야 했다. Open API 의 경우 기존의 v1 API 의 설계를 RESTful 하게 고치는 것과 함께 새로운 기능을 구현하는데 초점을 맞추었고, User-friendly Hub 는 기존 웹에서 AJAX 로 사용하기 위해 개발된 엔드 포인트들을 User-friendly Hub 의 스펙에 맞춘 형태로 개발하는 것이 주된 과제였다. 백엔드에서 사용하는

스키마의 복잡도가 높고 단순해 보였던 기능도 수많은 Service 들이 상호작용하는 경우가 많았기에 이러한 시스템을 이해한 것을 바탕으로 새 로직을 작성하는 것이 주요 관건이었다.

① 엔드 포인트 (Open API)

Open API 의 경우 기존의 기능들을 구현함과 동시에, CLI 로 자동화를 할 때 유용한 기능들을 추가하는 방향으로 개발하였다. 또한, 기존 기능을 옮기면서, 기존의 엔드 포인트들중 기능이 합쳐질 수 있다고 판단되는 경우 하나의 엔드 포인트로 합치기도 하였다. 예를 들어, Download URL 로 OSS 를 검색하는 GET /api/v1/downloadlocation_search API 와 전반적인 OSS 를 이름, 버전 등으로 검색하는 GET /api/v1/oss_search API 의 경우 하나로 구현하는 편이 직관적이라 판단하여 GET /api/v2/oss API 로 통합하였다.

또한, 엔드 포인트의 주소도 계층 구조가 더 잘 나타나도록 수정하였다. 예를 들어, POST /api/v1/prj_bom_export API 는 POST /api/v2/projects/{id}/bom/export API 로 수정했다.

마지막으로, 기존 API 의 경우 잘못된 값을 사용하거나 요청이 잘못되었을 경우 원인이 명확하게 드러나지 않는 경우가 있어, 이를 보여주기 위한 기능도 추가하였다. 앞서 서술한 ApiV2ExceptiopnAdvice 또한 이중 하나이며, 여기에는 주로 Authentication 을 위한 기능을 구현하였다. 해당 코드는 다음과 같다.

```
@ExceptionHandler(CUserNotFoundException.class)
    @ResponseStatus(HttpStatus.UNAUTHORIZED)
    protected ResponseEntity<Map<String, Object>> userNot-
Found(HttpServletRequest request, CUserNotFoundException e) {
        return responseService.errorResponse(HttpStatus.UNAUTHORIZED
                , CoCodeManag-
er.getCodeString(CoConstDef.CD_OPEN_API_MESSAGE, CoCon-
stDef.CD_OPEN_API_USER_NOTFOUND_MESSAGE));
    }

    @ExceptionHandler(CSigninFailedException.class)
    @ResponseStatus(HttpStatus.UNAUTHORIZED)
    protected ResponseEntity<Map<String, Object>>
emailSignInFailed(HttpServletRequest request, CSigninFailedException e)
{
        return responseService.errorResponse(HttpStatus.UNAUTHORIZED,
                CoCodeManag-
er.getCodeString(CoConstDef.CD_OPEN_API_MESSAGE, CoCon-
stDef.CD_OPEN_API_SIGNIN_FAILED_MESSAGE));

    }
```

그 외에 추가로 구현한 엔드 포인트의 경우, CLI 의 자동화에 도움이 되는 기능, 예를 들어 고지문 다운로드 기능을 추가하였다. 고지문의 경우 최근에 회사 이름 등 고지문의 생성에 필

요한 정보들을 OSS_NOTICE 테이블에 저장해 두는데, 최근 수정한 값을 들고 온 다음 Velocity template 을 통해 해당 값들과 프로젝트의 OSS/라이선스에 맞는 고지문를 생성해주는 방식이다.

새로 추가된 엔드 포인트들은 v1 과 마찬가지로 Swagger 로 문서화가 되도록 하였다. 새로운 API 들은 SwaggerConfig 에서 새로운 v2 그룹으로 분류되게 하였다. 또한, 기존의 파라미터 설명 중 예시가 직관적이지 않거나 백엔드 코드를 보지 않고는 어떤 용도인지 알기 힘든 것들이 많았기에 이런 경우 설명을 추가해주었다.

② 엔드 포인트 (User-friendly Hub)

User-friendly Hub 를 위한 엔드 포인트들에서는 기존 웹에서 사용하던 기능과 호환되면서도, 추가된 검색 파라미터 (후술할) 달라진 반환 값을 반환하기 위한 로직을 작성하였다. 구현한 Controller 들은 다음과 같다.

- LiteDashboardController
- LiteLicenseController
- LiteOssController
- LiteSelfCheckController
- LiteUserController
- LiteVulnerabilityController

각 Controller 들은 이름에 묘사된 대로 OSS, 라이선스 등의 자료구조를 다루며, 해당 자료구조의 검색, 상세 정보 조회, 엑셀 파일로 내보내기 기능 등이 구현되어 있다. 리스팅 기능의 경우, 입력받는 파라미터들을 이용해서 프로젝트에서 사용하는 iBATIS 를 이용하여 스펙에 맞는 SQL 쿼리로 바꾸는 것이 예상 외로 까다로운 작업이었다. 예를 들어, 기존 로직에서 OSS 를 조회하는 데 사용하는 쿼리의 경우 약 200 줄 길이의 쿼리로 구현되어 있는데, iBATIS 특성상 런타임 전에는 문제를 찾을 방법이 없었기에 MySQL 자체의 로깅 기능을 켜고 이를 활용하여 디버깅을 하는 게 최선이었다.

또한, iBATIS Mapper 의 경우 기존 코드에선 동일한 일을 하는 경우 중복된 코드가 많은 편이었다. 예를 들어, 특정 파라미터를 검색 값으로 삼아 row 를 가져오는 쿼리와 해당 파라미터에 일치하는 row 의 수를 세는 쿼리의 경우, WHERE 절의 조건이 완전히 일치해야만 한다. 관리의 용의성을 위해 추가된 쿼리에서 이런 일치해야만 하는 조건문과 Paging 과 같이 동일하게 실행되는 로직의 경우 <sql> 태그로 모듈화하여 따로 분리해 두었다. 예를 들어, Self-Check 를 가져오는 쿼리는 다음과 같다.

```
<resultMap id="SelfCheckDtoMap" type="oss.fosslight.api.dto.SelfCheckDto">
  <result property="projectId" column="prj_id"/>
  <result property="projectName" column="prj_name"/>
  <result property="projectVersion" column="prj_version"/>
  <result property="created" column="created_date"/>
  <result property="modified" column="modified_date"/>
  <result property="notice" column="notice_file_id"/>
  <result property="report" column="review_report_files_id"/>
</resultMap>
<select id="selectSelfCheckList" resultMap="SelfCheckDtoMap">
  SELECT * FROM
  <include refid="selfCheckQuery" />
  WHERE 1=1
  <include refid="paging"/>
</select>
```

위 예시의 경우 쿼리는 Mapping + Query + Paging(정렬 및 Limit 기능, 리스팅 공통) 세 단계로 분리된다.

다만 이러한 iBATIS 맵핑에 어떤 파라미터가 추가될지 모르며 두 쿼리의 행동에 관한 요구사항이 달라질지도 모르는 것이기에, 이러한 추상화는 '비슷하기에 추상화 할 수 있는 로직'보단 '반드시 일치해야만 하는 쿼리일 때' 사용하도록 하였다.

User-friendly Hub 에서 새로 추가된 기능을 위한 API 또한 구현해야 했다. 예를 들어, 전체 검색에 필요한 API, 대시보드 데이터의 로딩을 위한 API 등이 필요했다. 또한, 이번 User-friendly Hub 에서 Self-Check 과정에 기능을 확장하게 되었기에 이를 위해 필요한 API 또한 구현하였다. 먼저 소스 코드 공개가 필요한 경우를 위해 소스 코드(패키지) 파일을 업로드하는 기능을 구현하였는데, 이를 위해 Self-Check 관련 정보를 저장하는 PRE_PROJECT_MASTER 테이블에 해당 정보를 저장하는 컬럼을 추가하게 되었다. 파일이 업로드되는 경우 파일은 T2_FILE 의 형태로 저장되며, 이에 대한 레퍼런스를 필요로 하는 테이블에서 지니고 있게 된다. 그리고 Self-Check 프로젝트의 고지문 생성 과정에서 수정이 필요한 라이선스가 있는 경우 관리자에게 이메일을 보내는 기능도 추가가 되었는데, 이 또한 Open API 에서 추가한 고지문 API 와 마찬가지로 Velocity 로 생성한 이메일을 JavaMailSender 을 이용해 발송하는 방식이었다. 이메일의 경우 발송을 테스트가 까다로웠기에, JUnit 으로 테스트를 작성 후 @TestPropertySource 로 바꾼 SMTP configuration 값들을 읽어들여 테스트를 가능하게 하면서도 SMTP Credential 값들이 Git 에 포함되지 않도록 관리 하였다.

③ Data Structure

기존의 백엔드에서 내려주는 반환 값들은 DB 쿼리의 입력값, API 의 파라미터, API 의 반환 값들을 겸용하고 있었기에 데이터가 Over-fetching 되는 케이스가 많았고, 백여 개의 필드 중 개

발 및 디버깅 중에도 입력하거나 읽어와야 하는 필드가 무엇인지 파악하는 것이 힘들 때가 많았다. 하나의 데이터 클래스 안의 100 개가 넘어가는 필드를 변수명으로 용도를 알아내는 것은 불가능했으므로 디버깅 포인트를 사용하여 수정이 되는 필드를 찾아내야 했으므로 이러한 점이 개발에서 굉장히 어려운 점이었다. 다행히 User-friendly Hub 를 위한 API 의 경우 기존 FOSSlight hub 와 개별적인 웹이기에 엔드 포인트를 구현하는 package 를 분리할 수 있었고, 기존 엔드 포인트에서 사용하는 데이터 클래스에 의존성을 가지지 않는 환경을 구성할 수 있었다. 이런 환경을 마련한 뒤, 새로운 API 를 위한 DTO 들을 구현해 API 쿼리, API 에 따른 적절한 반환 값 들을 구현하였다. OSS 리스트 API 의 입력에 사용되는 클래스의 예시는 다음과 같다.

ⓐ 기존 OSS

```
public class OssMaster extends ComBean implements Serializable {
  private String ossName;
  private String ossNameTemp;
  private String schOssName;
  private String ossNameVerStr;
  private String ossNickname;
  private String[] ossNicknames;
  private String[] ossNicknameArr;
  private String[] ossNames;
  // 그 외 106 개의 필드들
}
```

ⓑ User-friendly Hub 전용 입력값

```
public class ListOssDto {
 // …
 @Setter
 @Getter
 @Builder(toBuilder = true)
 public static class Request extends Paging {
    private String ossName;
    private String ossId;
    @Builder.Default
    private Boolean ossNameExact = false;
    private String licenseName;
    @Builder.Default
    private Boolean licenseNameExact = false;
    private String url;
    @Builder.Default
    private Boolean urlExact = false;
    private String description;
    private String copyright;
    private String deactivate;
    private String ossType;
    private String licenseType;
    private String creator;
    private String createdFrom;
```

```
    private String createdTo;
    private String modifier;
    private String modifiedFrom;
    private String modifiedTo;
 }
}
```

ⓒ User-friendly Hub 전용 출력값

```
// ListOssDto.java
public class ListOssDto {
 // ...
 @Builder
 public static class Result {
  public List<OssDto> list;
  public int totalCount;
 }
}

// OssDto.java
@Data
public class OssDto implements ExcelData {
    String ossId;
    String ossType;
    String ossName;
    String ossVersion;
    String licenseName;
    String licenseType;
    String downloadUrl;
    String homepageUrl;
    String description;
    String cveId;
    String cvssScore;
    String creator;
    String created;
    String modifier;
    String modified;
    List<Character> obligations;
    String copyright;
    String nicknames;
    String attribution = "";
    Boolean exclude = false;
   // 그 외 Getter/Setters
}
```

ⓓ iBATIS 쿼리

```
<resultMap id="OssDtoMap" type="oss.fosslight.api.dto.OssDto">
    <result property="downloadUrl" column="download_location"/>
    <result property="homepageUrl" column="homepage"/>
    <result property="created" column="created_date"/>
    <result property="obligations" column="obligation_type"/>
    <result property="modified" column="modified_date"/>
```

```
    <result property="description" column="summary_description"/>
    <result property="nicknames" column="oss_nickname"/>
</resultMap>
<select id="selectOssList" parameter-
Type="oss.fosslight.api.dto.ListOssDto$Request" resultMap="OssDtoMap">
    SELECT
    T1.OSS_ID
    , T1.OSS_NAME
    , T1.OSS_VERSION
    , T1.LICENSE_DIV
    , T1.HOMEPAGE
    <choose>
        <when test="searchFlag">
            , SUBSTRING_INDEX(T1.DOWNLOAD_LOCATION, ',', 1) AS DOWN-
LOAD_LOCATION
        </when>
        <otherwise>
            , IFNULL((SELECT GROUP_CONCAT(D.DOWNLOAD_LOCATION ORDER BY
D.SORT_ORDER ASC)
            FROM OSS_DOWNLOADLOCATION D
            WHERE D.OSS_ID = T1.OSS_ID), T1.DOWNLOAD_LOCATION) AS DOWN-
LOAD_LOCATION
        </otherwise>
    </choose>
    , T1.SUMMARY_DESCRIPTION
    , T1.ATTRIBUTION
    , IFNULL((SELECT USER_NAME FROM T2_USERS WHERE T1.CREATOR = USER_ID),
T1.CREATOR) AS CREATOR
    , IFNULL((SELECT USER_NAME FROM T2_USERS WHERE T1.MODIFIER = USER_ID),
T1.MODIFIER) AS MODIFIER
    , T1.CVSS_SCORE
    , T1.CVE_ID
    , T1.CREATED_DATE
    , T1.MODIFIED_DATE
    , T3.LICENSE_NAME
    , T1.OBLIGATION_TYPE
    , SUB1.OSS_TYPE AS OSS_TYPE
    , IFNULL((SELECT CD_DTL_NM FROM T2_CODE_DTL WHERE CD_NO= '201' AND
CD_DTL_NO = T1.LICENSE_TYPE), T5.CD_DTL_NM )
    AS LICENSE_TYPE
    , (SELECT GROUP_CONCAT(OSS_NICKNAME SEPARATOR '|') FROM OSS_NICKNAME
WHERE OSS_NAME = T1.OSS_NAME) AS
    OSS_NICKNAME
    , T1.COPYRIGHT
    , T1.ATTRIBUTION
    , T1.DEACTIVATE_FLAG
    FROM OSS_MASTER T1
    INNER JOIN OSS_LICENSE_DECLARED T2 ON T1.OSS_ID = T2.OSS_ID
    INNER JOIN LICENSE_MASTER T3 ON T2.LICENSE_ID = T3.LICENSE_ID AND
T3.USE_YN = 'Y'
    INNER JOIN (SELECT OSS_ID, CONCAT(IF(MULTI_LICENSE_FLAG = 'N', '0',
'1'), IF(DUAL_LICENSE_FLAG = 'N', '0', '1'),
    IF(VERSION_DIFF_FLAG = 'N', '0', '1') ) AS OSS_TYPE FROM
OSS_MASTER_LICENSE_FLAG ) SUB1 ON T1.OSS_ID =
```

```
    SUB1.OSS_ID
    LEFT OUTER JOIN T2_CODE_DTL T5 ON T3.LICENSE_TYPE = T5.CD_DTL_NO AND
T5.CD_NO = '201' /*Code Table*/
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(url)">
        LEFT OUTER JOIN OSS_DOWNLOADLOCATION DOWN ON T1.OSS_ID =
DOWN.OSS_ID
    </if>
    WHERE T1.USE_YN = 'Y'
    /*검색 쿼리 시작*/
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(ossName)">
        AND (
        <choose>
            <when test="ossNameExact">
                T1.OSS_NAME = #{ossName}
            </when>
            <otherwise>
                T1.OSS_NAME LIKE CONCAT('%',REGEXP_REPLACE(#{ossName},
'_', '\\\\_'),'%')
            </otherwise>
        </choose>
        OR EXISTS (SELECT *
        FROM OSS_NICKNAME A1
        WHERE A1.OSS_NAME = T1.OSS_NAME
        <choose>
            <when test="ossNameExact">
                AND A1.OSS_NICKNAME = #{ossName}
            </when>
            <otherwise>
                AND A1.OSS_NICKNAME LIKE CON-
CAT('%',REGEXP_REPLACE(#{ossName}, '_', '\\\\_'),'%')
            </otherwise>
        </choose>
        )
        )
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(licenseName)">
        AND (
        <choose>
            <when test="licenseNameExact">
                UPPER(T3.LICENSE_NAME) = UPPER(#{licenseName})
            </when>
            <otherwise>
                UPPER(T3.LICENSE_NAME) LIKE UP-
PER(CONCAT('%',REGEXP_REPLACE(#{licenseName}, '_', '\\\\_'),'%'))
            </otherwise>
        </choose>
        OR
        UPPER(T3.LICENSE_NAME) IN (
        select UPPER(LICENSE_NAME)
        from LICENSE_NICKNAME
        where
        <choose>
            <when test="licenseNameExact">
                UPPER(LICENSE_NICKNAME) = UPPER(#{licenseName})
```

```
            </when>
            <otherwise>
                UPPER(LICENSE_NICKNAME) LIKE UP-
PER(CONCAT('%',REGEXP_REPLACE(#{licenseName}, '_', '\\\\_'),'%'))
            </otherwise>
        </choose>
        )
        OR
        <choose>
            <when test="licenseNameExact">
                UPPER(T3.SHORT_IDENTIFIER) = UPPER(#{licenseName})
            </when>
            <otherwise>
                UPPER(T3.SHORT_IDENTIFIER) LIKE UP-
PER(CONCAT('%',REGEXP_REPLACE(#{licenseName}, '_', '\\\\_'),'%'))
            </otherwise>
        </choose>
        )
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(description)">
        AND REPLACE(REPLACE(REPLACE(IFNULL(T1.SUMMARY_DESCRIPTION,
''),'\r\n', ' '),'\n', ' '), ' ', '') LIKE
        CONCAT('%',REPLACE(REPLACE(REPLACE(#{description},'\r\n', '
'),'\n', ' '), ' ', ''),'%')
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(url)">
        AND
        <choose>
            <when test="urlExact">
                (
                SUBSTRING_INDEX(SUBSTRING_INDEX(T1.HOMEPAGE, '//', -1),
'www.', -1) = #{url} OR
                SUBSTRING_INDEX(SUBSTRING_INDEX(T1.HOMEPAGE, '//', -1),
'www.', -1) = CONCAT(#{url},"/")
                OR SUBSTRING_INDEX(SUBSTRING_INDEX(T1.DOWNLOAD_LOCATION,
'//', -1), 'www.', -1) = #{url} OR
                SUBSTRING_INDEX(SUBSTRING_INDEX(T1.DOWNLOAD_LOCATION,
'//', -1), 'www.', -1) = CONCAT(#{url},"/")
                OR SUBSTRING_INDEX(SUBSTRING_INDEX(DOWN.DOWNLOAD_LOCATION,
'//', -1), 'www.', -1) = #{url} OR
                SUBSTRING_INDEX(SUBSTRING_INDEX(DOWN.DOWNLOAD_LOCATION,
'//', -1), 'www.', -1) = CONCAT(#{url},"/")
                )
            </when>
            <otherwise>
                (
                T1.HOMEPAGE LIKE CONCAT('%',REGEXP_REPLACE(#{url}, '_',
'\\\\_'),'%')
                OR T1.DOWNLOAD_LOCATION LIKE CON-
CAT('%',REGEXP_REPLACE(#{url}, '_', '\\\\_'),'%')
                OR DOWN.DOWNLOAD_LOCATION LIKE CON-
CAT('%',REGEXP_REPLACE(#{url}, '_', '\\\\_'),'%')
                )
            </otherwise>
```

```
        </choose>
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(copyright)">
        AND (
        REPLACE(REPLACE(REPLACE(IFNULL(T2.OSS_COPYRIGHT, ''),'\r\n', '
'),'\n', ' '), ' ', '') LIKE
        CONCAT('%',REPLACE(REPLACE(REPLACE(#{copyright},'\r\n', ' '),'\n',
' '), ' ', ''),'%')
        OR
        REPLACE(REPLACE(REPLACE(IFNULL(T1.COPYRIGHT, ''),'\r\n', '
'),'\n', ' '), ' ', '') LIKE
        CONCAT('%',REPLACE(REPLACE(REPLACE(#{copyright},'\r\n', ' '),'\n',
' '), ' ', ''),'%')
        )
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(creator)">
        AND T1.CREATOR LIKE CONCAT('%',#{creator},'%')
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(modifier)">
        AND T1.MODIFIER LIKE CONCAT('%',#{modifier},'%')
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(createdFrom)">
        AND DATE_FORMAT(T1.CREATED_DATE,'%Y-%m-%d') <![CDATA[>=]]> #{cre-
atedFrom}
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(createdTo)">
        AND DATE_FORMAT(T1.CREATED_DATE,'%Y-%m-%d') <![CDATA[<=]]> #{cre-
atedTo}
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(modifiedFrom)">
        AND DATE_FORMAT(T1.MODIFIED_DATE,'%Y-%m-%d') <![CDATA[>=]]> #{mod-
ifiedFrom}
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(modifiedTo)">
        AND DATE_FORMAT(T1.MODIFIED_DATE,'%Y-%m-%d') <![CDATA[<=]]> #{mod-
ifiedTo}
    </if>
    <if test="deactivate != null">
        AND T1.DEACTIVATE_FLAG = #{deactivate}
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(licenseType)">
        AND T1.LICENSE_TYPE = #{licenseType}
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(ossType)">
        AND
        (
        FALSE
        <if test="@oss.fosslight.util.StringUtil@contains(ossType, 'N')">
            OR OSS_TYPE LIKE '000'
        </if>
        <if test="@oss.fosslight.util.StringUtil@contains(ossType, 'M')">
            OR OSS_TYPE LIKE '1__'
        </if>
        <if test="@oss.fosslight.util.StringUtil@contains(ossType, 'D')">
```

```
            OR OSS_TYPE LIKE '_1_'
        </if>
        <if test="@oss.fosslight.util.StringUtil@contains(ossType, 'V')">
            OR OSS_TYPE LIKE '__1'
        </if>
        )
    </if>
    <if test="!@oss.fosslight.util.StringUtil@isEmpty(cvssScore)">
        <![CDATA[
         AND CAST(T1.CVSS_SCORE AS FLOAT ) >= CAST(#{cvssScore} AS FLOAT )
         ]]>
    </if>
    <if test="versionCheck">
        GROUP BY T1.OSS_NAME
    </if>
    <include refid="orderby"/>
    <include refid="limitPage"/>
</select>
```

위의 기존 OSS 클래스의 ossName 과 같이 하나의 값을 여러 형태로 나타내야 하는 경우, 새로 구현한 로직에서는 따로 값을 저장하기 보단 Getter/Setter 을 활용하여 일관된 하나의 필드로 저장해 두도록 구현하였다.

필드들의 이름에서 보이듯 기존의 통합된 클래스의 경우 모든 곳에서 사용하도록 구현하였기에 범용성을 확보하였으나, 그로 인해 용도를 구분하기 어려운 필드들이 많이 생긴다는 단점이 있었다. 모듈이 분리된 새로운 API 의 경우 기존의 데이터 클래스를 반드시 사용할 필요가 없었고, 기존 정의된 클래스를 사용하기엔 개발 용이성에서 얻는 불이익이 크다고 생각했기에 새로운 클래스를 정의하며, 최대한 필요한 필드만 포함하도록 했다. 또한, Lombok 라이브러리의 @Builder 기능을 사용하여 하나씩 값을 할당하거나, 인스턴스를 복제한 뒤 값을 바꿔야 할 때 사용하기 용이하게 구현하였다.

또한, 이러한 테이블로 출력되는 데이터를 표현하는 클래스들의 경우 엑셀 파일로 내보기를 하는 기능을 구현한 경우가 많았기에, 해당 기능을 사용할 때 엑셀 데이터를 생성하는 XSSWorkbook 에 넣어주는 데이터를 일일히 ExcelDownloadUtil.java 에 명시해 주는 대신 기존 데이터 클래스 내부에 row 로 변환되는 양식을 명시해줄 수 있는 interface 인 ExcelData 를 추가하였고, OssDto 도 이를 implement 한 클래스로서 구현하였다. 해당 클래스는 다음과 같은 코드로 엑셀 데이터로 변환한다.

```
private static <T extends ExcelData> String makeExcelDataId(List<T> dataL-
ist, String type) throws Exception {
  Workbook wb = null;
  Sheet sheet = null;
  FileInputStream inFile = null;
```

```
  try {
     inFile = new FileInputStream(downloadpath + File.separator + type +
".xlsx");
     try {
       wb = new XSSFWorkbook(inFile);
     } catch (IOException e) {
       log.error(e.getMessage());
     }
     sheet = wb.getSheetAt(0);
     wb.setSheetName(0, type);
     List<String[]> rows = dataL-
ist.stream().map(ExcelData::toRow).collect(Collectors.toList());
     makeSheet(sheet, rows);
  } catch (FileNotFoundException e) {
     log.error(e.getMessage(), e);
  } finally {
     if (inFile != null) {
       try {
         inFile.close();
       } catch (Exception ignored) {
       }
     }
  }

  return makeExcelFileId(wb, type);
}
```

④ Services

앞서 언급했듯 Service 의 경우 기존 v1 API, v2 API, 그리고 User-friendly Hub 에서 사용하는 API 들에서 필요한 기능들을 APIService 에서 구현하였다, 예를 들어, OSS 관련 기능의 경우 ApiOssService 에서 구현하였다. 기본적으로 Controller 는 Service 에, Service 는 Mapper 에, ApiController 는 ApiService/Service 에, ApiService 는 Service/Mapper/ApiMapper 에 의존성을 가지는 구조이므로, 이렇게 트리 형태의 의존성을 가지도록 구현함으로서 'Api'에 소속한 컴포넌트들은 기존 로직과의 의존성에서 비교적 자유로워지고, 기존 웹과의 로직이 조금 다른 기능을 구현해야 할 때 혼동을 줄일 수 있게 된다. 추가되는 로직들은 기존에 존재하던 ApiService 들 중 대응되는 클래스에서 새로운 메서드로 구현되었으며, Self-Check 와 같은 기능에서 필요한 Verification 의 경우 기존에는 비슷한 과정을 다루는 Service 가 없었기에 새로운 ApiVerificationService 에 package 파일 업로드, 삭제 등과 같이 추가로 필요한 로직을 구현하게 되었다.

8-2. FOSSLight Hub CLI

CLI 개발 배경은 다음과 같다. 사용자가 웹에서 수행해야 하는 반복 작업이 많았다. 이로 인해 자동화의 필요성이 컸지만, 자동화 개발 및 유지보수에 오버헤드가 크게 들어간다는 문제가 있었다. Open API 가 존재하여 자동화 스크립트 개발이 가능하지만, 각 사용자가 서로 다른 형태의 스크립트를 작성하여 사용하여 동일한 문제를 각자 해결하는 비효율성이 존재한다. 예를 들어 인증 토큰, 서버 정보 저장이나 API 사용 인터페이스 구축 등의 기본적으로 처리해야 하는 문제를 모두 각자의 방식으로 해결하고 있는 것이다. 만약 Open API 스펙이 바뀌면 문제는 더 커진다. 사용자들은 변경된 스펙에 맞춰서 수정하는 작업을 진행해야 하고, 번거로움 때문에 서비스 이용을 포기하는 사용자도 발생할 것이다. 이러한 문제를 해결하고 쉽게 자동화 가능한 환경을 제공하는 것이 CLI 개발의 목적이다.

먼저 Open API 접근성이 떨어지는 점을 개선하기 위하여, 인증 정보, 서버 정보를 쉽게 관리할 수 있도록 명령어를 추가하였고, 각 Open API 에 대응되는 명령어를 추가하여 Open API 호출을 위해 따로 스크립트를 작성하지 않아도 되도록 개선하였다.

오픈 소스 프로젝트이기 때문에 Open API 스펙이 바뀌더라도 한 명만 대응하면 모두 잘 작동할 것이다. YAML 파일을 통해 자동화 Flow 동작을 실행할 수 있는 Apply 명령어 인터페이스를 정의하였다. YAML 파일에 입력된 kind 값으로 어떤 Flow 를 진행할지 결정한다. 기존에는 각 사용자가 각자 자동화 스크립트를 작성하였다면, 이제는 이 인터페이스로 특정 Flow 를 진행하는 새로운 kind 들을 추가해나가며 동일한 문제를 겪는 다른 사용자가 스크립트를 직접 작성하지 않아도 되도록 비효율을 개선할 수 있을 것으로 기대한다. 예시로 프로젝트 생성, 스캔, 결과 업로드 Flow 를 수행하는 create-Project kind 를 추가해두었다.

아래의 코드는 createProject kind 를 구현한 것이다. 다른 사용자도 이와 비슷하게 kind 를 추가할 수 있다.

```
class Kind:
    CREATE_PROJECT = 'createProject'
    choices = [CREATE_PROJECT]
```

```
def create_project(data: dict):
    service = ProjectService()

    # check schema
    required = [
        "parameters",
        "parameters.prjName",
        "parameters.osType",

        "update.models.modelListToUpdate",
        "update.modelFile.modelReport",
        "update.watchers.emailList",
```

```
        "scan.dir",
    ]
    for field in required:
        keys = field.split(".")
        if len(keys) == 1:
            assert _has_key(data, keys), f"SchemaError: createPro-ject.{field}
required"
        else:
            if _has_key(data, keys[:-1]):
                assert _has_key(data, keys), f"SchemaError: createPro-
ject.{field} required"

    # create
    prjId = service.create(**data["parameters"])
    display_text(f"Project created ({prjId})")

    # update
    if update := data["update"]:
        if models := update.get("models"):
            service.update_models(prjId=prjId, model-
ListToUpdate=models["modelListToUpdate"])
        if model_file := update.get("modelFile"):
            service.update_model_file(prjId=prjId, modelRe-
port=model_file["modelReport"])
        if watchers := update.get("watchers"):
            service.update_watchers(prjId=prjId, email-
List=watchers["emailList"])
        display_text(f"Project updated")

    # scan
    if scan := data.get("scan"):
        service.scan(prjId=prjId, dir=scan["dir"])
    return prjId
```

이를 사용하려면 아래와 같은 YAML 파일을 작성하고, fosslight-cli apply create_project.yaml 명령어를 사용하면 된다.

YAML Copy Caption

```yaml
# create_project.yaml
kind: createProject
parameters:
  prjName: test-project
  prjVersion: 1
  osType: Linux
  distributionType: "General Model"
  networkServerType: N
  priority: P1
update:
  models:
    modelListToUpdate: "ASDF|AV/Car/Security > AV|20201010"
scan:
  dir: "~/data/simpleProject"
```

Open API 를 사용하기 어려웠던 부분 중 하나는 입력 파라미터를 넘기는 부분이었다. 프로젝트 생성 API 를 예로 들면, osType 의 종류로 Linux, Windows 등이 있는데 각각의 이름에 대응되는 코드 값을 전송해야 한다. 예를 들어, Linux 를 선택하고 싶다면 100 을 보내야 하고, Windows 를 선택하고 싶다면 200 을 보내야 하는 것이다. 만약 코드가 정해져 있다면 값을 외우거나 로컬 변수로 관리해서 처리가 가능하겠지만, 이 코드들은 동적으로 웹페이지에서 추가 제거할 수 있는 값이라 쉽게 처리하기 어렵다.

이러한 문제 때문에 CLI 로 API 를 호출할 수 있게 되었더라도 입력 파라미터를 찾는 부분에서 허들이 생기게된다. 이를 해결하기 위해 LG 측에서 제안했던 방법은 Interactive CLI 형태로 선택지를 유저에게 보여주는 방식이었는데, 자동화에 사용될 것을 고려하면 Interactive CLI 는 적합하지 않았기 때문에 기각하였다. 대안으로 찾은 방법이 유저에게는 Linux 로 입력을 받고, 동적으로 서버에서 코드 맵핑 값을 불러와서 100 으로 변환하여 서버로 보내는 방법을 선택하였다. 여기에 더해서 유저의 입력이 잘못되었을 때 입력 가능한 선택지 목록을 출력해주어 어떤 값을 사용해야 하는지 쉽게 알 수 있게 하였다.

아래의 코드는 유저가 입력한 코드를 받아서 해당 값을 변환하고, 유효하지 않으면 선택 가능한 목록과 함께 오류를 출력하는 코드이다. 유저가 처음부터 100 과 같은 실제 값을 입력한 경우에는 해당 값을 그대로 사용하고, Linux 와 같은 값을 입력했다면 서버에서 코드 목록을 불러와 이를 통해 변환을 거친 후 사용한다.

```python
from src.client import get_api_client
from src.enums import CodeType


# get the code value used for API calls
def get_code_value(name: str, code_type: CodeType) -> str:
    client = get_api_client()
    response = client.get_code(codeType=code_type)
    code_list = response.json()['content']
    # To make it case-insensitive using upper()
    mapping = {x["cdDtlNm"].upper(): x["cdDtlNo"] for x in code_list}

    # name is already code value
    if name in mapping.values():
        return name

    # convert name to code value
    value = mapping.get(name.upper())
    if value is not None:
        return value
    raise ValueError(f"{CodeType.get_display_name(code_type)} - '{name}' is not a valid code.\n- available code inputs: {', '.join(mapping.keys())}")
```

아래의 코드는 프로젝트 생성 함수로, 유저가 입력한 코드 값들을 get_code_value 함수를 이용하여 변환하고 API Client 모듈로 넘기고 있다.

```
def create(
    self,
    prjName,
    osType,
    distributionType,
    networkServerType,
    priority,
    osTypeEtc=None,
    prjVersion=None,
    publicYn=None,
    comment=None,
    userComment=None,
    watcherEmailList=None,
    modelListToUpdate=None,
    modelReportFile=None,
) -> str:
    response = get_api_client().create_project(
        prjName=prjName,
        osType=get_code_value(osType, CodeType.OS_TYPE),
        distributionType=get_code_value(distributionType, Code-
Type.DISTRIBUTION_TYPE),
        networkServerType=networkServerType,
        priority=get_code_value(priority, CodeType.PRIORITY),
        osTypeEtc=osTypeEtc,
        prjVersion=prjVersion,
        publicYn=publicYn,
        comment=comment,
        userComment=userComment,
        watcherEmailList=watcherEmailList,
        modelListToUpdate=modelListToUpdate,
        modelReportFile=modelReportFile,
    )
    check_response(response)
    return response.json()['prjId']
```

8-3. User-friendly FOSSLight Hub

User-friendly Hub 를 개발하는 본질적인 목적은 크게 두 가지이다. 하나는 사용자의 역할만을 고려하여 더 나은 UI & UX 를 구축하는 것이고, 다른 하나는 트렌드에 맞고 유지보수성이 좋은 기술 스택을 활용하여 SPA 웹사이트를 올바르게 구현하는 것이다. 즉, 첫 번째 관점은 UI & UX 에 관한 것이고, 두 번째 관점은 구현(Implementation)에 관한 것이다. 지금부터 각 관점별로 어떤 철학과 접근 방법을 통해 우리의 달성 과제를 해결하였는지 설명한다.

① UI & UX

1) 깔끔한 디자인의 시작은 바로 색상이다. 무작정 여러 색상을 사용하게 되면 지저분한 느낌을 지울 수 없다. 따라서 User-friendly Hub 에서 사용할 몇 가지의 대표적인 브랜드 색상들을 정하고, 특별한 경우를 제외하고는 해당 색상들을 이용하여 디자인을 진행하였다.

```
{
  charcoal: '#34393f',
  semicharcoal: '#68727e',
  crimson: '#b02a42',
  semiwhite: '#eeeeee',
  semiblack: '#222222',
  gray: '#bbbbbb',
  semigray: '#cccccc',
  darkgray: '#aaaaaa'
}
```

2) 기존 웹은 관리자의 역할까지 고려한 나머지 메뉴들이 너무 많고, 단순히 메뉴 개수를 떠나서도 관련된 메뉴들이 묶여있는 구조도 아니라 각 메뉴가 어떤 기능을 하는지 단번에 파악하는 게 쉽지 않다. 따라서 User-friendly Hub 에서는 사용자에게 필요한 메뉴들만을 남기고, 그중에서도 OSS, 라이선스, 보안 취약점은 데이터베이스에 등록된 OSC 관련 정보라는 공통점을 갖고 있으므로 Database 라는 하나의 대메뉴로 묶어서 정리하였다. 따라서 사용자의 입장에서는 메인 페이지에 해당하는 대시보드를 제외하면, 크게 Self-Check 를 위한 페이지들과 Database 관련 페이지들로 나뉜다는 것을 쉽게 파악할 수 있다. 더불어, 각 페이지는 공통적으로 상단에 페이지의 위치와 제목을 나타내는 Breadcrumb 를 표시하고 그 바로 아래에 해당 페이지의 역할에 관한 짧은 설명문을 표시하였다. 이를 통해 사용자는 웹사이트의 전체적인 구조를 빠르게 파악하고 각 페이지의 역할을 쉽게 이해할 수 있다. 한편, 모바일 사용성까지 지원하는 것을 목표로 하기 때문에 모바일에서는 메뉴들로 이뤄진 하단 바를 통해 네비게이션을 수행할 수 있도록 하였다. 하단 바는 모바일 앱 시장에서 최근에 가장 많이 채택되는 UI 로, 추후 앱으로의 발전 가능성까지 고려했을 때 더 좋은 사용자 경험을 가져다줄 수 있는 확장성을 갖고 있다.

3) 기존 웹에서는 각 데이터가 무엇을 의미하는지 한 번에 이해하기 어려운 워딩과 UI 를 갖추고 있다. 예를 들어, OSS 에는 두 종류의 URL 필드가 있는데 하나는 Download Location 이라고 불리고 다른 하나는 Homepage 라고 부른다. 둘 다 URL 이지만 하나만 Location 이라고 부르고 있기 때문에 통일성을 해치고, 더군다나 라이선스에도 OSS 의 Homepage 와 거의 동일한 필드가 있는데 이름이 Homepage 가 아닌 Website 이다. 또한, 같은 필드도 종종 때에 따라 다르게 표현되는데, OSS 에서 Description 이라고 부르는 것이 다른 곳에서는 Summary Description 이라고 불리는 것이 하나의 예시이다. 마지막으로, 의미적으로 관련된 필드들끼리 그룹화를 하지 않고 일방적으로 모든 필드를 나열하는 식이기 때문에 보는 사람 입장에서는 필드 정보를 직관

적으로 확인하기 쉽지 않다. 따라서 User-friendly Hub 에서는 최대한 비슷한 필드를 같은 레벨의 워딩으로 통일하고(EX. Download Location → Download URL, Homepage → Homepage URL, Website → Homepage URL), 때에 따라 표현하는 방식이 최대한 달라지지 않도록 하여(EX. Summary Description → Description) 같은 필드를 같은 필드로 이해할 수 있게 하였다. 또한, 각 필드의 의미를 고려하여 비슷한 필드끼리 뭉치도록 필드들을 배치하고(목록 페이지 내 테이블 열의 순서, 상세 팝업에서 필드의 배치 순서) 구분선 등의 UI 를 적극 활용하여(상세 팝업) 각 필드의 의미를 직관적으로 이해할 수 있게 하였다.

4) Self-Check 상세 페이지의 경우, 사용자에게 필요한 기능을 위주로 남기고 각 기능을 최대한 직관적으로 이해할 수 있는 UI & UX 를 고안하였다. 프로젝트의 기본 정보가 내용에 비해 너무 많은 공간을 차지하고 있고 내용이 눈에 띄지 않으므로 해당 섹션의 크기를 줄이되 각 내용이 눈에 띄도록 디자인을 수정하고, 사용자 입장에서는 불필요한 Watcher 설정 기능을 삭제하였다. 그리고 기존 웹에서는 프로젝트의 삭제 버튼이 OSS 목록의 UI 에 배치되어 있는데, 이는 OSS 목록의 삭제로 오인한 사용자가 프로젝트 자체를 삭제할 위험성이 있다. 따라서 [Delete] 버튼을 [Edit] 버튼과 나란히 둠으로써 해당 버튼들이 프로젝트를 대상으로 한 수정/삭제 기능임을 명확히 하였다. 다음으로, Self-Check 를 위한 3 단계 프로세스가 직관적으로 이해될 수 있도록 화살표를 이용한 UI 로 각 단계의 탭을 구성하고, 각 탭의 이름은 OSS, Package, Notice 로 간결하게 변경하며, 각 탭을 선택할 때마다 바로 아래에 해당 탭의 역할에 관한 짧은 설명문을 표시하였다. 이를 통해 사용자는 Self-Check 프로세스가 어떤 절차로 이뤄져 있는지, 그리고 각 절차에서 수행해야 하는 일이 무엇인지 빠르게 파악할 수 있다.

5) Self-Check 상세 페이지의 OSS 탭에 해당하는 UI 는 굉장히 복잡한데, 최대한 각 기능을 쉽게 이해할 수 있도록 필요한 최소한의 기능만을 남기고 중복된 기능은 제거하였다. 한 프로젝트에 속한 OSS 의 개수가 수만 개의 단위는 아니기 때문에 API 는 해당 프로젝트의 모든 OSS 를 반환하도록 단순화하고, 프론트엔드에서 필터링, 정렬, 페이지네이션을 수행하였다. 이를 통해 서버의 코드를 간소화할 수 있을 뿐 아니라 이미 한 번에 가져온 데이터를 가지고 액션을 수행하기 때문에 액션의 반응 속도도 훨씬 더 빨라지는 효과를 얻었다. 기존 웹의 [Reset] 버튼은 전체 선택과 삭제 버튼을 활용하여 비슷하게 흉내낼 수 있는 중복 기능이므로 삭제하고, OSS 와 라이선스를 검사하는 두 개의 버튼은 하나의 [Check] 버튼으로 합쳐서 버튼의 개수를 최소화했다. 그리고 [Export] 버튼은 다른 페이지에 위치한 [Export] 버튼과 동일한 UI 를 갖도록 함으로써 비슷한 기능을 수행하는 버튼임을 명확히 했다. 그리고 기존 웹에서는 수정이 즉시 가능한 Writable 테이블의 UI 를 제공하고 있는데, 좁은 테이블에 많은 데이터와 Write 기능까지 넣다 보니 전체적으로 테이블이 매우 복잡하다는 느낌을 지울 수 없었다. 따라서 깔끔한 UI 에 최적화되어 있는 카드 UI 를 통해 각 OSS 의 정보를 보여주고, 추가나 수정이 필요한 경우에는

별도의 팝업을 띄워서 Write 작업을 수행할 수 있도록 변경하였다. 또한, 기존에는 OSS 탭에 처음 들어왔을 때도 [Save]를 눌러 저장하지 않으면 [Check OSS Name] 등의 기능이 수행되지 않은 불편함이 있는데, 초기 상태라면 변동사항이 없으므로 [Save]를 하지 않아도 되는 것이 더 자연스럽다. 하여 이러한 부분도 함께 수정하였다.

6) Self-Check 상세 페이지의 Notice 탭에서는 고지문의 다운로드 및 미리보기가 가능한데, 기존 웹에서는 기본적으로 선택되어 있는 파일 형식이 보이지 않을 뿐 아니라 파일 형식을 선택하는 Select 바가 미리보기 버튼과 다운로드 버튼의 사이에 위치해서 해당 Select 바가 어떤 기능에 매칭되는지 혼란을 야기하였다. 따라서 Select 바를 삭제하고 [Download] 버튼을 클릭했을 때 별도의 팝업을 통해 다운로드하고자 하는 파일의 형식을 선택할 수 있도록 개선하였다.

7) Self-Check, OSS, 라이선스, 보안 취약점의 목록 페이지는 공통의 UI 와 기능을 공유하도록 함으로써 목록 페이지로서의 정체성을 사용자가 인식할 수 있도록 했다. 기존 웹에서 목록 페이지의 테이블은 열의 너비를 효율적으로 활용하지 못하여 화면이 조금만 좁아져도 내용이 많이 무너지는 단점이 있다. 따라서 보고 싶은 열만 볼 수 있도록 특정 열을 선택하는 열 선택 기능을 배치함과 동시에 반응형을 고려하여 열의 너비가 부족한 경우 가로 스크롤이 생기도록 하고, 너무 자세한 정보는 테이블이 아닌 열을 클릭했을 때 뜨는 상세 팝업을 통해 확인할 수 있도록 간소화했다. 이때 상세 팝업은 그 자체로 하나의 URL 에 맵핑되게 함으로써 뒤로 가기를 통해 닫을 수 있는 편의까지 제공하도록 한다. 이밖에도 필터링, 정렬, 페이지네이션 등의 기능도 동일한 UI & UX 를 공유하도록 함으로써 한 페이지만 익숙해지면 나머지 페이지도 금방 익숙해질 수 있도록 했다. 비슷한 아이디어로, 대시보드 페이지와 전체 검색 결과 페이지 또한 각 항목을 보여주는 목록 UI & UX 를 동일하게 구성함으로써 사용자가 웹사이트의 구조를 최대한 쉽게 파악할 수 있도록 했다.

8) 대시보드 페이지에서는 가장 최근에 데이터베이스에 반영된 보안 취약점, OSS, 라이선스의 정보를 한눈에 볼 수 있도록 했다. 만약 프로젝트에 속한 특정 OSS 의 취약점이 최근에 발견되었다면 대시보드를 통해 그 소식을 빠르게 확인하고 대응할 수 있게 되며, 아직 등록되지 않은 OSS 나 라이선스가 등록되었다는 사실도 빠르게 접해볼 수 있게 된다.

9) OSS, 라이선스, 보안 취약점의 정보를 One-depth 로 검색할 수 있는 전체 검색 기능을 구현하였다. 보안 취약점의 경우 CVE ID 를 기준으로, OSS 의 경우 이름과 닉네임을 기준으로, 라이선스의 경우 이름, 닉네임, 그리고 약칭을 기준으로 필터링된다. 검색 키워드에 매칭되는 부분은 노란 형광색의 하이라이트로 표시했다. 검색 결과도 항목별로 최대 5 개로 제한하기 때문에 검색 속도가 빠르다. 따라서 필요한 정보만 빠르게 찾고 싶은 사용자에게 최적화된 기능이다.

② Implementation

1) 프론트엔드에서 최근에 가장 트렌디하게 사용되고 있는 Next.js 프레임워크를 활용하여 SPA 웹사이트를 구현하였다. 따라서 SPA 의 핵심이라고 할 수 있는 클라이언트 사이드 네비게이션을 직접 구현할 필요가 없어진다. 기존 웹은 클라이언트 사이드 네비게이션을 직접 구현한 듯 보이긴 하지만 각 방문 기록을 스택에 남기지 않기 때문에 뒤로가기를 하는 순간 해당 웹사이트 자체를 떠나게 된다. Next.js 를 이용한 웹사이트의 구현은 그러한 문제를 해결하였다.

2) CSS 라이브러리를 최소한으로 사용하고 TailwindCSS 를 활용함으로써 별도의 CSS 파일을 만들지 않도록 했다. CSS 라이브러리에 전적으로 의존하면 불필요한 CSS 코드가 남아 코드의 품질이 떨어질 가능성이 있다. 따라서 직접 구현하기 매우 번거로운 몇몇 컴포넌트(EX. toggle) 등을 제외하고 대부분의 CSS 를 직접 구현함으로써 코드의 품질을 확보했다. 대신, 자주 사용되는 UI 는 컴포넌트로 리팩토링함으로써 재활용성을 극대화했다.

3) 기존 웹에서 지원하지 않던 반응형 디자인을 구현함으로써 모바일과 같이 뷰포트가 작은 장치에서도 User-friendly Hub 를 사용할 수 있게 했다. 이 때문에 CSS 의 작업량은 두 배가 되지만 스마트폰의 사용 시간이 굉장히 많은 걸 감안했을 때는 충분히 시간을 들일 만했다. 기본적으로 뷰포트의 변화에 따른 디자인의 변화는 TailwindCSS 의 media 쿼리를 이용하여 구현하되, 768px 을 기준으로 변하는 웹사이트의 전체적인 레이아웃은 useMediaQuery() Hook 을 활용하여 뷰포트를 감지하고 분기 처리하였다. 다음은 /app/(main)/layout.tsx 파일의 코드이다.

```
export default function Layout({ children }: { children: Re-
act.ReactNode }) {
  const [user, setUser] = useRecoilState(userState);
  const [view, setView] = useRecoilState(viewState);
  const loading = useRecoilValue(loadingState);
  const [isSideBarShown, setIsSideBarShown] = useState(true);
  const userLoadingRef = useRef<HTMLDivElement>(null);
  const router = useRouter();
  const isMobile = useMediaQuery({ maxWidth: 768 });
  const isSubwindow = typeof window !== 'undefined' && Boole-
an(window.opener);

  ...

  useEffect(() => {
    if (isMobile) setView('mobile');
    else setView('pc');
  }, [setView, isMobile]);

  // Wait until detecting appropriate view
  if (view === 'none') return null;

  return (
```

```
    <>
      {/* Wait until detecting logined user */}
      <div
        className={clsx(
          'fixed center flex flex-col items-center z-[1000] transition-
opacity duration-200',
          user ? 'opacity-0' : 'opacity-100'
        )}
        ref={userLoadingRef}
      >
        <div className="mb-4 text-lg font-bold">Loading . . .</div>
        <Loading />
      </div>

      {/* Main content */}
      <main
        className={clsx(
          'min-h-screen transition-opacity duration-200',
          view === 'pc' && 'flex',
          user ? 'opacity-100' : 'opacity-0'
        )}
      >
        {/* Left navigation bar (PC) */}
        {view === 'pc' && !isSubwindow && <SideBar
isShown={isSideBarShown} />}

        {/* Page body */}
        <div className={clsx(view === 'pc' && 'flex-1 min-w-0', loading &&
'opacity-50')}>
          {/* Areas fixed at the top of the page */}
          {!isSubwindow && (
            <div
              className={clsx(
                'sticky top-0 bg-white z-10',
                view === 'pc' && 'flex flex-col gap-y-10 pt-4 px-4 pb-4'
              )}
            >
              {/* Hamburger button (PC) or Top bar (Mobile) */}
              {view === 'pc' ? (
                <button
                  className="w-6 h-6 text-xl text-charcoal"
                  onClick={() => setIsSideBarShown(!isSideBarShown)}
                >
                  <i className="fa-solid fa-bars" />
                </button>
              ) : (
                <TopBar />
              )}

              {/* Full search bar */}
              <FullSearchBar />
            </div>
          )}
```

```
          {/* Page content */}
          <div
            className={clsx(
              'pt-4 mx-4 overflow-x-auto transition-opacity duration-300
no-scrollbar',
              view === 'pc' ? 'pb-8' : 'pt-4 pb-24'
            )}
          >
            {children}
          </div>
        </div>

        {/* Bottom navigation bar (Mobile) */}
        {view === 'mobile' && !isSubwindow && <BottomBar />}

        {/* Modal for detail view */}
        <DetailModal />

        {/* Loading */}
        {loading && (
          <div className="fixed center">
            <Loading />
          </div>
        )}
      </main>
    </>
  );
}
```

4) 런타임 에러를 최대한 줄이고 정적인 타입 검사를 통해 안정적인 코드 베이스를 구축하기 위해, TypeScript 의 타입 시스템을 적극 활용하였다. 개선된 UI & UX 를 구현하는 것만큼 새로운 기술 스택의 유지보수성을 최대한 확보하는 것도 중요하다고 생각했고, 이에 따라 any 타입은 최대한 배제하고 각 페이지에서 요구하는 주요 데이터(Self-Check, OSS, 라이선스, 보안 취약점 등)의 타입을 명확히 선언하여 프로그래밍 과정에서의 실수를 줄이고 또 한편으로는 TypeScript 의 자동 완성 기능을 활용하여 프로그래밍 생산성까지 함께 증진시켰다. 다음은 전역으로 각 데이터의 타입을 선언한 @types/globals.d.ts 파일의 내용이다.

```
declare global {
  namespace Common {
    type OSSTypes = Record<string, { name: string; desc: string; color:
string }>;
  }

  namespace Nav {
    interface RootMenu {
      name: string;
      icon: string;
    }

    interface Menu {
```

```
      name: string;
      icon: string;
      path: string;
      sub?: {
        name: string;
        path: string;
      }[];
    }
  }

  namespace List {
    interface Filter {
      label: React.ReactNode;
      name: string;
      type: 'char' | 'char-exact' | 'select' | 'checkbox' | 'date' | 'num-
ber' | 'text';
      options?: { label: React.ReactNode; value: string }[];
    }

    interface Column {
      name: string;
      sort: string;
    }

    interface SelfCheck {
      projectId: string;
      projectName: string;
      projectVersion: string;
      ossCount: number;
      packages: SelfCheck.PackageFile[];
      cveId: string;
      cvssScore: string;
      created: string;
    }

    interface OSS {
      ossId: string;
      ossName: string;
      ossVersion: string;
      ossType: string;
      licenseName: string;
      licenseType: string;
      obligations: string[];
      downloadUrl: string;
      homepageUrl: string;
      description: string;
      cveId: string;
      cvssScore: string;
      creator: string;
      created: string;
      modifier: string;
      modified: string;
    }
```

```
    interface License {
      licenseId: string;
      licenseName: string;
      licenseIdentifier: string;
      licenseType: string;
      obligations: string[];
      restrictions: string[];
      homepageUrl: string;
      description: string;
      creator: string;
      created: string;
      modifier: string;
      modified: string;
    }

    interface Vuln {
      ossName: string;
      ossVersion: string;
      vendor: string;
      cveId: string;
      cvssScore: string;
      summary: string;
      modified: string;
    }
  }

  namespace SelfCheck {
    interface Basics {
      projectName: string;
      projectVersion: string;
      created: string;
      comment: string;
    }

    interface Edit {
      projectId: string;
      projectName: string;
      projectVersion: string;
      comment: string;
    }

    interface Tab {
      name: 'OSS' | 'Package' | 'Notice';
      title: string;
      description: string;
    }

    interface OSSFile {
      fileId: string;
      fileSeq: string;
      logiNm: string;
      orgNm: string;
      created: string;
      state?: 'add' | 'delete';
```

```
}

interface OSSLicense {
  licenseId: string | null;
  licenseName: string;
}

interface OSS {
  gridId: string;
  ossId: string | null;
  ossName: string;
  ossVersion: string;
  obligations: string[];
  vuln: boolean;
  cveId: string;
  cvssScore: string;
  licenses: OSSLicense[];
  path: string;
  userGuide: string;
  copyright: string;
  restrictions: string;
  downloadUrl: string;
  homepageUrl: string;
  exclude: boolean;
  changed?: 'add' | 'edit';
}

type OSSValidMap = Record<string, Record<string, string>>;

interface EditOSS {
  gridId: string;
  ossName: string;
  ossVersion: string;
  licenses: OSSLicense[];
  path: string;
  copyright: string;
  downloadUrl: string;
  homepageUrl: string;
}

interface OSSCheck {
  gridIds: string[];
  downloadUrl: string;
  before: { value: string; msg?: string };
  after: { value: string; msg?: string };
}

interface LicenseCheck {
  gridId: string;
  ossName: string;
  ossVersion: string;
  downloadUrl: string;
  before: { value: string[]; msg?: string };
  after: { value: string[]; msg?: string };
```

```
    }

    interface PackageFile {
      fileId: string;
      fileSeq: string;
      logiNm: string;
      orgNm: string;
      created: string;
    }

    interface PackageOSS {
      ossId: string;
      ossName: string;
      ossVersion: string;
      licenseName: string;
      downloadUrl: string;
      homepageUrl: string;
    }
  }

  namespace ListSection {
    interface Vuln {
      ossName: string;
      ossVersion: string;
      cveId: string;
      cvssScore: string;
      summary: string;
      modified: string;
    }

    interface OSS {
      ossId: string;
      ossName: string;
      ossVersion: string;
      licenseName: string;
      obligations: string[];
      cveId: string;
      cvssScore: string;
      created: string;
      modified: string;
    }

    interface License {
      licenseId: string;
      licenseName: string;
      licenseIdentifier: string;
      obligations: string[];
      restrictions: string[];
      created: string;
      modified: string;
    }
  }

  namespace Detail {
```

```
interface OSSLicense {
  licenseId: string;
  licenseName: string;
  licenseIdentifier: string;
  comb: '' | 'AND' | 'OR';
}

interface OSSVuln {
  cveId: string;
  cvssScore: string;
  summary: string;
}

interface OSS {
  ossName: string;
  ossNicknames: string[];
  ossVersion: string;
  ossType: string;
  licenses: OSSLicense[];
  licenseType: string;
  obligations: string[];
  downloadUrl: string;
  homepageUrl: string;
  description: string;
  copyright: string;
  attribution: string;
  vulnerabilities: OSSVuln[];
  deactivate: boolean;
  creator: string;
  created: string;
  modifier: string;
  modified: string;
}

interface License {
  licenseName: string;
  licenseNicknames: string[];
  licenseIdentifier: string;
  licenseType: string;
  obligations: string[];
  restrictions: string[];
  homepageUrl: string;
  description: string;
  licenseText: string;
  attribution: string;
  creator: string;
  created: string;
  modifier: string;
  modified: string;
}

interface VulnOSS {
  ossName: string;
  ossVersion: string;
```

```
    }

    interface Vuln {
      cveId: string;
      cvssScore: string;
      summary: string;
      modified: string;
      oss: VulnOSS[];
    }
  }
}

export {};
```

5) 기존 웹의 로그인 세션과 Closed API 를 그대로 이식하기 위해서 세션 기반의 인증을 동일하게 따르도록 하였다. 만약 Open API 에서 다루는 것과 같이 JWT 등의 토큰을 기반으로 한 인증을 사용한다고 하면 기존 웹에서 쓰던 모든 Closed API 들을 그 인증 방식으로 변경해야 할 뿐 아니라, 기존 웹의 로그인 세션과 User-friendly Hub 의 로그인 세션이 분리되므로 사용자 입장에서는 최악의 경우 두 번이나 로그인해야 하는 불편함이 생길 수도 있다. 따라서 기존 웹에 대한 하위호환성을 보장하기 위해 세션 ID 기반의 인증을 그대로 채택하였다.

6) 앞서 UI & UX 파트에서 언급했듯 Self-Check, OSS, 라이선스, 보안 취약점의 목록 페이지는 공통의 UI 와 기능을 공유하도록 하였는데, 대표적으로 필터, 정렬, 열 선택기, 페이지네이션 등의 기능이 바로 그것에 해당한다. 이것들 중에서 필터는 ListFilters 컴포넌트로 리팩토링하였고, 정렬, 열 선택기, 페이지네이션은 테이블의 UI 자체를 리팩토링한 ListTable 컴포넌트에 포함시켰다. 다음은 순서대로 ListFilters 컴포넌트와 ListTable 컴포넌트의 코드이다.

```
function renderFilters(filters: List.Filter[], form: UseFormReturn) {
  const labelClass = 'flex-shrink-0 w-20 lg:w-24';
  const inputClass = 'px-2 py-1 border border-darkgray outline-none';

  return filters.map((filter) => {
    // Single-line text
    if (filter.type === 'char' || filter.type === 'char-exact') {
      return (
        <div key={filter.name} className="flex items-start gap-x-4">
          <div className={labelClass}>{filter.label}</div>
          <div className="flex-1">
            <div className="flex gap-x-2">
              <input className={clsx('flex-1 w-0', inputClass)}
{...form.register(filter.name)} />
              {filter.type === 'char-exact' && (
                <label className="flex items-center gap-x-1.5">
                  <input
                    className="w-4 h-4"
                    type="checkbox"
                    {...form.register(`${filter.name}Exact`)}
```

```
                  />
                  Exact
                </label>
              )}
            </div>
          </div>
        </div>
      );
    }

    // Select
    if (filter.type === 'select') {
      return (
        <div key={filter.name} className="flex items-start gap-x-4">
          <div className={labelClass}>{filter.label}</div>
          <div className="flex-1">
            <select className={clsx('w-full', inputClass)}
{...form.register(filter.name)}>
              <option value="">(Select)</option>
              {filter.options?.map((option) => (
                <option key={option.value} value={option.value}>
                  {option.label}
                </option>
              ))}
            </select>
          </div>
        </div>
      );
    }

    // Checkbox
    if (filter.type === 'checkbox') {
      return (
        <div key={filter.name} className="flex items-start gap-x-4">
          <div className={labelClass}>{filter.label}</div>
          <div className="flex-1">
            <div className="flex flex-wrap gap-x-4 gap-y-1">
              {filter.options?.map((option) => (
                <label key={option.value} className="flex items-center
gap-x-1.5">
                  <input
                    className="w-4 h-4"
                    type="checkbox"
                    value={option.value}
                    {...form.register(filter.name)}
                  />
                  {option.label}
                </label>
              ))}
            </div>
          </div>
        </div>
      );
    }
```

```
    // Date
    if (filter.type === 'date') {
      return (
        <div key={filter.name} className="flex items-start gap-x-4">
          <div className={labelClass}>{filter.label}</div>
          <div className="flex-1">
            <div className="flex items-center gap-x-2">
              <div className="relative flex-1 w-0">
                <input
                  className={clsx('w-full invisible', inputClass)}
                  type="date"
                  {...form.register(`${filter.name}From`)}
                />
                <div
                  className={clsx('absolute inset-0', inputClass)}
                  onClick={() => {
                    const input = docu-
ment.querySelector(`[name=${filter.name}From]`);
                    (input as HTMLInputElement).showPicker();
                  }}
                >
                  {(() => {
                    const date = dayjs(form.watch(`${filter.name}From`));
                    if (!date.isValid()) return '';
                    return date.format('YYYY.MM.DD');
                  })()}
                </div>
              </div>
              ~
              <div className="relative flex-1 w-0">
                <input
                  className={clsx('w-full invisible', inputClass)}
                  type="date"
                  {...form.register(`${filter.name}To`)}
                />
                <div
                  className={clsx('absolute inset-0', inputClass)}
                  onClick={() => {
                    const input = docu-
ment.querySelector(`[name=${filter.name}To]`);
                    (input as HTMLInputElement).showPicker();
                  }}
                >
                  {(() => {
                    const date = dayjs(form.watch(`${filter.name}To`));
                    if (!date.isValid()) return '';
                    return date.format('YYYY.MM.DD');
                  })()}
                </div>
              </div>
            </div>
          </div>
        </div>
```

```
      );
    }

    // Number
    if (filter.type === 'number') {
      return (
        <div key={filter.name} className="flex items-start gap-x-4">
          <div className={labelClass}>{filter.label}</div>
          <div className="flex-1">
            <input
              className={clsx('w-full', inputClass)}
              type="number"
              {...form.register(filter.name)}
            />
          </div>
        </div>
      );
    }

    // Multi-line text
    return (
      <div key={filter.name} className="flex items-start gap-x-4">
        <div className={labelClass}>{filter.label}</div>
        <div className="flex-1">
          <textarea
            className={clsx('block w-full resize-none', inputClass)}
            rows={3}
            {...form.register(filter.name)}
          />
        </div>
      </div>
    );
  });
}

export default function ListFilters({
  form,
  filters
}: {
  form: UseFormReturn;
  filters: { default: List.Filter[]; hidden: List.Filter[] };
}) {
  const { default: defaultFilters, hidden: hiddenFilters } = filters;

  const loading = useRecoilValue(loadingState);
  const [areHiddenFiltersShown, setAreHiddenFiltersShown] = us-
eState(false);
  const router = useRouter();
  const pathname = usePathname();
  const queryParams = useSearchParams();
  const filtersQueryParam = queryParams.get('f') || '';

  // Reflect states on URL query parameters (state -> URL)
  function setFilters(filterParams: FieldValues) {
```

```
    if (loading) {
      return;
    }

    const urlQueryParams = new URLSearchParams(queryParams);

    const f = stringifyFilters(filterParams);
    if (f) {
      urlQueryParams.set('f', f);
    } else {
      urlQueryParams.delete('f');
    }
    urlQueryParams.delete('p');

    router.push(`${pathname}?${urlQueryParams.toString()}`, { scroll:
false });
  }

  // Reflect URL query parameters on states (URL -> state)
  useEffect(() => {
    const f = stringifyFilters(form.watch());
    if (f !== filtersQueryParam) {
      form.reset(parseFilters(filtersQueryParam, filters), { keepDefault-
Values: true });
    }
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [filtersQueryParam, form]);

  return (
    <form
      className="relative w-[calc(100%-4px)] shadow-box"
      onSubmit={form.handleSubmit(setFilters)}
    >
      {/* Default filters */}
      <div className="grid grid-cols-1 gap-x-8 gap-y-2 text-sm lg:grid-
cols-2">
        {renderFilters(defaultFilters, form)}
      </div>

      {/* Hidden filters (If exists) */}
      {hiddenFilters.length > 0 && (
        <>
          <div
            className={clsx(
              'overflow-y-hidden transition-[max-height] duration-500',
              areHiddenFiltersShown ? 'max-h-96' : 'max-h-0'
            )}
          >
            <hr className="my-3 border-semigray" />
            <div className="grid grid-cols-1 gap-x-8 gap-y-2 text-sm
lg:grid-cols-2">
              {renderFilters(hiddenFilters, form)}
            </div>
          </div>
```

```
          <button
            className="absolute top-full center-x px-2 py-0.5 border-x
border-b border-darkgray rounded-b outline-none text-xs"
            type="button"
            onClick={() => setAreHiddenFil-
tersShown(!areHiddenFiltersShown)}
          >
            {areHiddenFiltersShown ? 'Hide' : 'Expand'} 
            {areHiddenFiltersShown ? (
              <i className="fa-solid fa-chevron-up" />
            ) : (
              <i className="fa-solid fa-chevron-down" />
            )}
          </button>
        </>
      )}

      {/* Search button */}
      <div className="mt-2 text-right">
        <button className="px-2 py-0.5 crimson-btn" disabled={loading}>
          Search
        </button>
      </div>
    </form>
  );
}
```

```
function generatePagination(currPage: number, lastPage: number) {
  const pageCandidates = new Set([
    ...[currPage - 1, currPage, currPage + 1],
    ...[1, 2],
    ...[lastPage - 1, lastPage]
  ]);
  const pages: number[] = [];

  Array.from(pageCandidates)
    .filter((page) => page >= 1 && page <= lastPage)
    .sort((a, b) => a - b)
    .forEach((page, idx, arr) => {
      if (idx > 0 && page - arr[idx - 1] > 1) {
        pages.push(-1);
      }

      pages.push(page);
    });

  return pages;
}

export default function ListTable({
  rows,
  columns,
  checkbox,
  currentSort,
  pagination,
```

```
  hideColumnSelector = false,
  render,
  onClickRow
}: {
  rows: any[];
  columns: List.Column[];
  checkbox?: {
    name: string;
    checkedList: number[];
    setCheckedList: (checkedList: number[]) => void;
    disabledList: number[];
  };
  currentSort?: string;
  pagination?: { totalCount: number; currentPage: number; countPerPage:
number };
  hideColumnSelector?: boolean;
  render: (row: any, column: string) => React.ReactNode;
  onClickRow?: (row: any) => void;
}) {
  const view = useRecoilValue(viewState);
  const loading = useRecoilValue(loadingState);
  const [isColumnSelectorShown, setIsColumnSelectorShown] = us-
eState(false);
  const [isColumnShown, setIsColumnShown] = useState(
    Object.fromEntries(columns.map((column) => [column.name, true]))
  );

  const router = useRouter();
  const pathname = usePathname();
  const queryParams = useSearchParams();

  function setSort(sort: string) {
    if (currentSort === undefined || !sort || loading) {
      return;
    }

    let newSortInfo: string[] | null = [sort, 'asc'];

    if (currentSort.includes('-')) {
      const [k, d] = currentSort.split('-');
      if (k === sort) {
        if (d === 'asc') newSortInfo = [sort, 'dsc'];
        else if (d === 'dsc') newSortInfo = null;
      }
    }

    const urlQueryParams = new URLSearchParams(queryParams);

    if (newSortInfo) {
      urlQueryParams.set('s', `${newSortInfo[0]}-${newSortInfo[1]}`);
    } else {
      urlQueryParams.delete('s');
    }
```

```
    router.push(`${pathname}?${urlQueryParams.toString()}`, { scroll:
false });
  }

  function setPage(page: number) {
    if (!pagination || page === pagination.currentPage || loading) {
      return;
    }

    const urlQueryParams = new URLSearchParams(queryParams);

    if (page > 1) {
      urlQueryParams.set('p', String(page));
    } else {
      urlQueryParams.delete('p');
    }

    router.push(`${pathname}?${urlQueryParams.toString()}`, { scroll:
false });
  }

  useEffect(() => {
    function handleClickOutsideColumnSelector(e: MouseEvent) {
      if (e.target && !(e.target as Element).closest('.column-selector'))
{
        setIsColumnSelectorShown(false);
      }
    }

    document.addEventListener('mousedown', handleClickOutsideColumnSelec-
tor);
    return () => {
      document.removeEventListener('mousedown', handleClickOutsideCol-
umnSelector);
    };
  }, []);

  let validCheckList = rows.map((_, idx) => Number(idx));
  if (checkbox && checkbox.disabledList.length > 0) {
    validCheckList = validCheckList.filter((idx) => !check-
box.disabledList.includes(idx));
  }

  return (
    <>
      <div className="relative overflow-x-auto no-scrollbar">
        <table className={clsx('w-full text-sm', rows.length === 0 &&
'min-h-[160px]')}>
          {/* Columns */}
          <thead>
            <tr className="border-b-2 border-charcoal/80 text-left
whitespace-nowrap">
              {checkbox && (
                <th className="p-2">
```

```
                  <input
                    type="checkbox"
                    checked={
                      validCheckList.length > 0
                        ? validCheckList.length ===
                          checkbox.checkedList.filter((idx) => !check-
box.disabledList.includes(idx))
                            .length
                        : false
                    }
                    onChange={(e) => {
                      if (e.target.checked) {
                        checkbox.setCheckedList(
                          checkbox.disabledList
                            .filter((idx) => check-
box.checkedList.includes(idx))
                            .concat(validCheckList)
                        );
                      } else {
                        checkbox.setCheckedList(
                          checkbox.checkedList.filter((idx) => check-
box.disabledList.includes(idx))
                        );
                      }}
                  />
                </th>
              )}
              {columns
                .filter((column) => isColumnShown[column.name])
                .map((column) => (
                  <th key={column.name} className="p-2">
                    <button
                      className="flex gap-x-2"
                      onClick={() => setSort(column.sort)}
                      disabled={currentSort === undefined || !column.sort
|| loading}
                    >
                      {column.name}

                      {/* Sorting */}
                      {currentSort !== undefined && column.sort && (
                        <span className="relative inline-block w-2">
                          {(() => {
                            let [up, down] = [false, false];

                            if (currentSort.includes('-')) {
                              const [k, d] = currentSort.split('-');
                              if (k === column.sort) {
                                if (d === 'asc') up = true;
                                else if (d === 'dsc') down = true;
                              }
                            }
```

```
                            return (
                              <>
                                <i
                                  className={clsx(
                                    'absolute inset-0 pt-1 fa-solid fa-
sort-up',
                                    !up && 'text-semigray'
                                  )}
                                />
                                <i
                                  className={clsx(
                                    'absolute inset-0 pt-1 fa-solid fa-
sort-down',
                                    !down && 'text-semigray'
                                  )}
                                />
                              </>
                            );
                          })()}
                        </span>
                      )}
                    </button>
                  </th>
                ))}
              {!hideColumnSelector && (
                <th className="column-selector relative w-8 p-2">
                  <button onClick={() => setIsColumnSelector-
Shown(!isColumnSelectorShown)}>
                    <i className="fa-solid fa-eye" />
                  </button>
                  {isColumnSelectorShown && (
                    <div className="absolute top-full right-0 flex flex-
col gap-y-1.5 p-3 mt-0.5 bg-white border-x border-b border-darkgray round-
ed-b shadow-[-2px_2px_4px_0_rgba(0,0,0,0.2)]">
                      {columns.map((column) => (
                        <label key={column.name} className="flex justify-
end items-center gap-x-2">
                          {column.name}
                          <input
                            type="checkbox"
                            checked={isColumnShown[column.name]}
                            onChange={(e) => {
                              const { checked } = e.target;
                              const checkedCnt = Ob-
ject.values(isColumnShown).filter((isShown) =>
                                Boolean(isShown)
                              ).length;

                              if (!checked && checkedCnt <= 3) {
                                return;
                              }

                              setIsColumnShown({
                                ...isColumnShown,
```

```
                                [column.name]: checked
                              });
                            }}
                          />
                        </label>
                      ))}
                    </div>
                  )}
                </th>
              )}
            </tr>
          </thead>

          {/* Rows */}
          <tbody>
            {rows.map((row, idx) => (
              <tr
                key={idx}
                className={clsx(
                  'border-b border-semigray',
                  onClickRow && 'cursor-pointer hover:opacity-80'
                )}
                onClick={() => onClickRow && onClickRow(row)}
              >
                {checkbox && (
                  <td className="px-2 py-1.5">
                    <input
                      className={checkbox.name}
                      type="checkbox"
                      checked={checkbox.checkedList.includes(idx)}
                      disabled={checkbox.disabledList.includes(idx)}
                      value={idx}
                      onChange={(e) => {
                        if (e.target.checked) {
                          check-
box.setCheckedList([...checkbox.checkedList, idx]);
                        } else {
                          const sliceIdx = check-
box.checkedList.findIndex((i) => i === idx);
                          checkbox.setCheckedList([
                            ...checkbox.checkedList.slice(0, sliceIdx),
                            ...checkbox.checkedList.slice(sliceIdx + 1)
                          ]);
                        }
                      }}
                    />
                  </td>
                )}
                {columns
                  .filter((column) => isColumnShown[column.name])
                  .map((column) => (
                    <td key={column.name} className="px-2 py-1.5">
                      {render(row, column.name)}
                    </td>
```

```
                  ))}
                {!hideColumnSelector && <td></td>}
              </tr>
            ))}
          </tbody>
        </table>

        {/* When there are no rows */}
        {rows.length === 0 && (
          <div className="absolute center text-darkgray">There are no en-
tries.</div>
        )}
      </div>

      {/* Pagination */}
      {view !== 'none' && pagination && (
        <div
          className={clsx(
            'flex mt-4 items-center',
            view === 'pc' ? 'justify-between' : 'flex-col-reverse gap-y-4'
          )}
        >
          <div className="text-darkgray">
            {rows.length} entries (total {insertCom-
mas(pagination.totalCount)} entries)
          </div>
          <div className="flex items-center gap-x-2">
            {generatePagination(
              pagination.currentPage,
              Math.max(Math.ceil(pagination.totalCount / pagina-
tion.countPerPage), 1)
            ).map((page, idx) => {
              if (page === -1) {
                return <i key={`${page}${idx}`} className="fa-solid fa-
ellipsis" />;
              }

              return (
                <button
                  key={page}
                  className={clsx(
                    'px-2 py-0.5 border',
                    page === pagination.currentPage
                      ? 'bg-charcoal border-charcoal text-semiwhite'
                      : 'border-darkgray'
                  )}
                  onClick={() => setPage(page)}
                  disabled={page === pagination.currentPage || loading}
                >
                  {page}
                </button>
              );
            })}
          </div>
```

```
        </div>
      )}
    </>
  );
}
```

위 두 개의 컴포넌트에서 가장 집중적으로 구현을 진행한 부분은 바로 URL 쿼리 파라미터와의 동기화 로직이다. 사실 필터, 정렬, 페이지네이션을 React 상태로 구현하는 것은 그렇게까지 어려운 것은 아니지만, 오로지 React 상태로만 구현할 경우 URL 쿼리 파라미터에 현재 적용된 필터, 정렬, 페이지네이션의 정보가 남지 않기 때문에 새로고침을 하거나 누군가에게 해당 URL을 공유할 때 그러한 정보가 유지되지 않는다. 즉, 새로고침을 하면 필터, 정렬, 페이지네이션이 전혀 적용되지 않은 상태로 리로드된다는 것이다. 따라서 적용된 필터, 정렬, 페이지네이션 정보를 어떻게든 시시각각 URL 쿼리 파라미터에 연동을 시켜서 이러한 문제를 해결하였다. 결론적으로, 기존 웹에서는 애초에 SPA 구현 자체가 제대로 되어 있지 않아서 새로고침 시 보고 있던 페이지마저 떠나지만, User-friendly Hub 에서는 보고 있던 페이지가 그대로 유지될 뿐 아니라(Next.js 의 라우팅 메커니즘에 근간) 이미 적용했던 필터, 정렬, 페이지네이션의 정보가 그대로 유지된다. 따라서 사용자 입장에선 크나큰 UX 의 개선을 가져다 줄 수 있다. 이는 React 상태가 변할 때마다 URL 쿼리 파라미터를 변경하고, URL 쿼리 파라미터가 변할 때마다 React 상태를 변경하는 양방향 연동을 구현함으로써 얻을 수 있는 이점이다. 생각보다 많은 사용자는 새로고침이나 뒤로가기 동작을 무의식적으로 많이 수행하기 때문에, 이러한 네비게이션 동작에 대한 기존 상태의 유지 메커니즘은 섬세하게 다뤄져야 한다.

비슷한 맥락으로, OSS, 라이선스, 보안 취약점의 상세 팝업 또한 그 자체가 하나의 URL 에 맵핑되어 뒤로가기를 하면 해당 팝업이 꺼지고 앞으로가기를 하면 다시 팝업이 켜지고, 팝업이 켜진 상태에서 새로고침을 하면 팝업이 유지되도록 하였다. 다음은 해당 동작을 구현한 DetailModal 컴포넌트의 코드이다.

```
export default function DetailModal() {
  const [dataType, setDataType] = useState('');
  const router = useRouter();
  const queryParams = useSearchParams();
  const modalType = queryParams.get('modal-type') || '';
  const modalId = queryParams.get('modal-id') || '';

  useEffect(() => {
    if (!modalType || !modalId) {
      return;
    }

    setDataType(modalType);
  }, [modalType, modalId]);
```

```
  return (
    <Modal show={Boolean(modalType && modalId)} onHide={() => rout-
er.back()} size="lg">
      {(() => {
        if (dataType === 'oss') return <DetailModalOSS modalId={modalId}
/>;
        if (dataType === 'license') return <DetailModalLicense
modalId={modalId} />;
        if (dataType === 'vuln') return <DetailModalVuln modalId={modalId}
/>;
        return null;
      })()}
    </Modal>
  );
}
```

7) Next.js 기반의 User-friendly Hub 를 Docker 환경에 귀속시켰다. 즉, docker-compose up 명령어를 통해 User-friendly Hub 도 함께 실행될 수 있도록 설정하였다. 다음은 루트 디렉토리에 위치한 docker-compose.yml 파일에 추가한 코드이다.

```
lite:
  build:
    context: ./lite
    dockerfile: ./Dockerfile
    args:
      - PRODUCT_MODE=${PRODUCT_MODE:-false}
  container_name: fosslight_lite
  depends_on:
    - db
  ports:
    - "3000:3000"
```

db 컨테이너가 실행된 다음 실행되도록 설정하고, 컨테이너의 3000 번 포트가 호스트의 3000 번 포트와 연결되도록 맵핑하였다. 이미지는 /lite/Dockerfile 파일에 따라 빌드하도록 하고, 빌드된 이미지로 실행하는 컨테이너의 이름은 fosslight_lite 로 지정했다. 이때 동일한 디렉토리의 .env 파일로부터 읽은 PRODUCT_MODE 의 값(default: false)을 Dockerfile 에 전달함으로써 이미지를 빌드할 때 개발 환경인지 배포 환경인지 구분할 수 있도록 했다. 이렇게 설정한 이유는 빌드 타임에 환경 변수를 읽어 코드를 빌드하는 Next.js 의 특성 때문이다. /lite 디렉토리에 위치한 Dockerfile 의 내용은 다음과 같다.

```
FROM node:18

# Copy Source Codes
COPY . /data/fosslight
WORKDIR /data/fosslight

# Install Node.js Packages
RUN npm install
```

```
# Set Environment Variable for Build
ARG PRODUCT_MODE
ENV NEXT_PUBLIC_PRODUCT_MODE=$PRODUCT_MODE

# Bundle Source Codes
RUN npm run build

# Run Front-end Server
CMD ["npm", "run", "start"]
```

여기서 PRODUCT_MODE 가 바로 docker-compose.yml 파일에서 .env 파일의 내용을 읽어 전달해준 값이다. 이를 통해 Next.js 가 정적 파일을 빌드(npm run build)하기 전에 적절한 환경 변수를 읽어서 설정할 수 있도록 했다. 빌드가 끝나고 나면 서버를 실행(npm run start)한다.

빌드 전에 NEXT_PUBLIC_PRODUCT_MODE 환경 변수를 설정해줬기 때문에, Next.js 에서는 이 값을 바탕으로 API 를 호출할 때 사용할 서버의 다음과 같이 Origin 을 결정할 수 있게 된다.

```
export const serverOrigin =
  process.env.NEXT_PUBLIC_PRODUCT_MODE === 'true'
    ? 'https://fosslight.org'
    : 'http://localhost:8180';
```

그리고 이를 바탕으로 직접 정의한 useAPI() Hook 은 다음과 같다. API 가 호출되는 패턴을 고려하여 중복되는 코드가 최소화되도록 리팩토링하였다. API 를 호출하기 위한 URL 을 얻을 때 앞서 정의한 serverOrigin 변수의 값을 사용한다.

```
export function useAPI(
  method: 'get' | 'post' | 'put' | 'patch' | 'delete',
  path: string,
  config?: {
    onStart?: () => void;
    onSuccess?: (res: any) => void;
    onError?: (err: unknown) => void;
    onFinish?: () => void;
    type?: 'json' | 'file';
  }
) {
  const mutationOptions: any = {};
  if (config?.onStart) {
    mutationOptions.onMutate = config.onStart;
  }
  if (config?.onSuccess) {
    mutationOptions.onSuccess = config.onSuccess;
  }
  if (config?.onError) {
    mutationOptions.onError = config.onError;
  }
  if (config?.onFinish) {
```

```
    mutationOptions.onSettled = config.onFinish;
  }

  const url = `${serverOrigin}${path}`;

  const mutation = useMutation(async (data: { params?: any; body?: any } |
null) => {
    const requestConfig: AxiosRequestConfig = { method, url, withCreden-
tials: true };

    if (data?.params) {
      requestConfig.params = data.params;
      requestConfig.paramsSerializer = (params) => qs.stringify(params,
{ arrayFormat: 'repeat' });
    }

    if (data?.body && method !== 'get') {
      if (config?.type === 'json') {
        requestConfig.data = data.body;
        requestConfig.headers = { 'Content-Type': 'application/json' };
      } else if (config?.type === 'file') {
        requestConfig.data = data.body;
        requestConfig.headers = { 'Content-Type': 'multipart/form-data' };
      } else {
        requestConfig.data = qs.stringify(data.body, { arrayFormat: 're-
peat' });
      }
    }

    return axios(requestConfig);
  }, mutationOptions);

  const api: typeof mutation & {
    execute: typeof mutation.mutate;
    executeAsync: typeof mutation.mutateAsync;
  } = {
    ...mutation,
    execute: mutation.mutate,
    executeAsync: mutation.mutateAsync
  };

  return api;
}
```

마지막으로, User-friendly Hub 를 개발하면서 어려움을 겪었던 부분은 다음과 같다.

먼저, 가장 많이 시간을 쓰고 어려움을 겪었던 부분은 기존 웹에 대한 이해와 이를 바탕으로 한 역기획에 있었다. 사실상 스펙이나 요구사항이라는 것이 명확히 있지 않고, 그저 사용자의 Flow 와 기존 웹의 동작을 정확히 이해하고 이를 바탕으로 사용자가 필요로 하는 부분을 파악해서 새로운 기술 스택을 통해 더 개선된 버전의 웹사이트를 신설하는 것이 목표였는데, 기본적으로 DB 스키마가 굉장히 복잡하게 설계되어 있고 과거의 기술 스택으로 작성된 웹과 서버의 코드를 빠르게 이해하는 것도 어

려웠다. 디자인 자체가 본업이 아니다 보니 디자인 기획 자체에도 많은 심력이 소모되었고, 실제로 대기업이 사용할 하나의 웹사이트를 한 학기 동안 만들어야 한다는 것이 굉장히 부담스럽게 느껴졌다.

다음으로, 주요 API 의 경우 API 담당자가 깔끔하게 작업해주었기 때문에 프론트엔드와 연결하는 것에 큰 어려움이 없었지만, 현실적으로 이미 존재하던 거대한 웹사이트를 새로운 기술 스택으로 옮기는 과정에서 기존의 웹이 사용하던 수많은 Closed API 들을 API 담당자가 수정하는 것은 불가능했기 때문에 일부 Closed API 들은 기존 웹이 쓰던 방식을 역으로 이해해서 사용해야만 했다. 하지만 앞서 언급했듯이 DB 스키마가 상당히 복잡한데다가 한 개의 쿼리가 몇백 줄에 달했고 심지어 오늘날 사용하는 기술 스택이 아니었기 때문에 코드를 해석하는 과정이 마치 암호 해독과 같이 느껴졌다. 그래서 일부 API 는 기존 웹에서 어떻게 호출하고 있는지를 역으로 분석해서 그 의미를 추론해야 하는 경우도 있었다. 아마 User-friendly Hub 를 개발하는 데 들인 시간 중에서 전체적인 UI & UX 를 기획하는 시간 다음으로 가장 많은 시간을 쓴 부분일 것이다.

또한, 앞서 말한 원칙에 따라 CSS 라이브러리를 최소한으로 사용하려고는 했지만 팝업(모달)과 자동 완성 입력 등의 UI 까지 직접 구현하기에는 시간적으로 모자랄 것 같아 관련 라이브러리를 찾았는데, react-hook-form 과 TypeScript 와 제대로 호환되는 라이브러리를 찾지 못해 직접 구현할 수밖에 없었다. 원래는 라이브러리를 활용하여 구현하는 것을 갑자기 직접 구현하려고 하니 불가능한 수준까지는 아니었지만 굉장히 낯설었고 또 생각보다 고려해야 할 케이스가 너무나도 많아서 시간을 꽤나 많이 잡아 먹었다. 또, 기존 웹에서 사용하던 서식 라이브러리인 CKEditor 를 이식하려고 했는데, 마찬가지로 TypeScript 와 올바르게 호환되지 않아서 호환성을 맞추기 위해 적지 않은 시간을 소모하였다.

9. Achievement

9-1. FOSSLight Hub REST API

Open API 의 경우 v2 API 를 개발함으로써 각 엔드 포인트를 좀 더 직관적으로 바꾸었고, 파라미터의 혼동을 줄였으며 기능을 확장하였다. 에러 처리와 ExceptionAdvice 에서 catch 하는 경우의 수를 늘려서 사용자 입장에서 개발과 API 의 활용에 있어서 필요한 정보를 더 쉽게 얻을 수 있도록 하였다. 정보가 늘어난 API 를 사용할 경우 CLI 등의 연동 서비스의 개발에도 도움이 될 수 있을 것이다.

User-friendly Hub 측의 API 의 경우 용도에 따라 필요한 정보만을 데이터 클래스 내에 정의하는 방향성을 채택했으므로, 추후 기능이 확장된다고 해도 데이터 클래스의 복잡도가 늘어나는 정도를 어느 정도 완화할 수 있을 것이다.

9-2. FOSSLight Hub CLI

CLI 개발을 통해 Open API 를 쉽게 사용할 수 있게 되었고, 거기에 더해 Scanner 와 결합한 명령어와 YAML 파일을 읽어서 프로젝트를 생성해주는 명령어를 활용한다면 기존에 수동으로 진행하던 반복 작업들을 자동화할 수 있게 된다. 이는 업무의 효율성을 증가시키며, 실수할 가능성을 줄여준다.

만약 여러 프로젝트를 동시에 진행한다면, 분석 파일이나 모델 파일 등 여러 파일들을 프로젝트별로 잘 나눠서 관리해야 하는 등 신경써야 할 부분이 많다. CLI 의 apply 기능을 사용하면 YAML 파일로 프로젝트의 명세, 버전, 모델 정보 등을 관리할 수 있고, 자동으로 생성, 스캔, 업로드가 진행되기 때문에 사용자가 신경써야 하는 부분이 엄청나게 줄어든다. 여기에 더해서 CLI 의 API 들을 같이 조합한 스크립트를 작성하거나 CI/CD 를 활용한다면 더 효과적으로 자동화할 수 있게 된다.

결론적으로 CLI 를 도입하여 웹의 오버헤드를 극단적으로 줄일 수 있게 되었다. 이는 사용자들의 진입 장벽을 낮추는 효과가 있으며, 서비스 사용자 수 증가에 긍정적인 영향을 줄 것으로 기대된다.

9-3. User-friendly FOSSLight Hub

User-friendly Hub 의 개발에 따라 얻게 되는 이점을 정리하면 다음과 같다.

첫째, 기존에 제대로 동작하지 않던 클라이언트 사이드 네비게이션을 제대로 동작하도록 수정했다. 기존에는 클라이언트 사이드에서 페이지 이동이 이뤄지기는 해도 방문 기록을 스택에 쌓지 않고 URL 이 변하지 않기 때문에 새로고침을 하면 보고 있던 페이지가 날아가고 뒤로가기를 하면 아예 해당 웹사이트에서 이탈하게 되는 문제가 있었다. 그러나 Next.js 기반의 SPA 웹사이트를 구현함에 따라 각 네비게이션이 방문 기록을 쌓고 URL 을 변경시키기 때문에 새로고침이나 뒤로가기 시에도 사용자가 의도하고 예상한 대로 네비게이션이 동작하게 된다. 또한 목록 페이지에서 필터, 정렬, 페이지네이션 결과도 URL 쿼리 파라미터에 반영하였고, OSS, 라이선스, 보안 취약점의 상세 팝업은 그 자체로 URL 과 맵핑되도록 했기 때문에 실수로 뒤로가기나 새로고침을 하더라도 원하지 않는 네비게이션이 이뤄지지 않도록 하는 메커니즘을 구축하였다.

둘째, Next.js 13 과 TypeScript 를 중심으로 한 최신 트렌드의 기술 스택들을 기반으로 웹사이트를 신설하였기 때문에, 미래를 내다보았을 때 유지보수의 가능성이 훨씬 더 확보되는 장점이 있다. 한 학기 동안 기존 웹의 코드를 살펴본 결과 너무나도 오래된 기술의 코드라서 오랫동안 유지보수를 하기는 어렵다는 생각이 강하게 들었고, 그렇기에 이번에 우리의 결과물은 앞으로도 꽤나 오랫동안 유지되고 상용화될 수 있는 가능성을 연다.

셋째, 모바일과 같이 뷰포트가 좁은 장치에서도 무너지지 않도록 반응형 UI 를 구현하였다. 이때 단순히 뷰포트의 크기에 따라 각 요소를 늘리고 줄이는 정도에서 그치는 것이 아니라, 뷰포트가 768px 이하가 되면 아예 레이아웃 자체를 모바일 친화적으로 바꿔버리기 때문에 모바일에서 사용하는 것이 사실상 불가능했던 기존 웹에 비하면 엄청난 사용성의 개선을 가져다준다. 모바일 친화적 레이아웃은 마치 네이티브 앱을 사용할 때와 비슷한 느낌을 받도록 하는 UI 를 제공함으로써 사용자로 하여금 모바일로도 해당 웹사이트를 편리하게 사용할 수 있도록 돕는다.

넷째, 기존 웹의 설치 방법과 사실상 완전히 호환되도록 환경을 구축하였고 기존 웹의 로그인 세션이 남아 있다면 User-friendly Hub 에서도 해당 세션을 그대로 사용할 수 있도록 지원하기 때문에, 지금까지 기존 웹을 사용하던 사람의 입장에서는 별다른 추가적인 설정 없이도 User-friendly Hub 를 활용할 수 있다는 장점이 있다. 물론 기존 웹을 완전히 대체하는 것은 아니고 웹사이트의 역할 자체가 조금 다르긴 하지만, 이를 사용하기 위한 별도의 설정 과정이 많았다면 User-friendly Hub 를 받아들이는 데 거부감이 생기기 마련이었을 것이다.

다섯째, Self-Check 관련 페이지들의 UI & UX 를 전체적으로 직관적으로 바꾸고 복잡한 기능을 단순화하여 정리하였으며, 요구사항에 따라 새로운 기능까지 추가로 구현하였다. User-friendly Hub 에서는 Self-Check 상세 페이지에서 불필요한 Watcher 설정 영역을 삭제하였고, OSS 탭에 존재하는 많은 버튼들을 최대한 단순화하여 세 개의 버튼만을 남겼고, 잘못된 곳에 위치해 있던 프로젝트의 삭제 버튼을 올바른 곳으로 옮겼으며, 무엇보다 OSS 목록의 Writable 테이블 UI 를 Read-only 에 최적화된 카드 UI 로 바꾸고 추가/수정 기능은 팝업으로 구현하여 조회 및 추가/수정 작업을 독립적으로 분리하였다. 그리고 회사 측에서 추가로 요구했듯이 OSS 단계와 Notice 단계의 사이에 Package 단계를 추가하여 패키지(소스 코드) 파일을 업로드하고 다운로드할 수 있는 기능을 신설하였고, OSS 의 목록에서 라이선스가 잘못 기입된 경우 고지문 생성을 하지 못하도록 막고 필요시 관리자에게 이메일을 발송하는 기능까지도 추가로 구현하였다.

여섯째, Self-Check, OSS, 라이선스, 보안 취약점의 목록 페이지에 있는 테이블의 UI 가 무너지지 않도록 수정하였고 모바일과 같이 뷰포트가 좁은 환경에서도 테이블의 정보를 잘 조회할 수 있도록 가로 스크롤과 열 선택 기능을 추가하였다. 또한, 상세 팝업을 별도로 구현하였기 때문에 지나치게 상세한 정보는 테이블에서 제외함으로써 테이블 내 각 열의 너비도 최적화를 할 수 있었다.

일곱째, OSS, 라이선스, 보안 취약점의 상세 팝업을 별도로 구현함으로써 각 항목의 상세 정보를 확인하는 UI 가 명확하게 제공되지 않던 기존 웹의 문제점을 해소하였다. 기존 웹에서 OSS 와 라이선스는 수정 페이지가 곧 상세 페이지의 역할을 하였기에 수정 기능을 고려한 나머지 상세 페이지라고 하기에는 다소 불필요하고 복잡한 UI 가 많았고, 연관된 정보들끼리 잘 묶여있지 않아 익숙한 사람이 아니라면 필요한 필드를 확인하는 데 불편하였다. 그리고 보안 취약점은 상세 페이지라고 할 만한 것이

별도로 존재하지 않았기 때문에 특정 보안 취약점의 정보를 확인하는 것이 쉽지 않았다. 그러나 User-friendly Hub 에서는 아예 각 항목의 상세 팝업을 별도로 구현하였기 때문에 이러한 문제점들이 전부 해소되고 상세 정보 조회를 위한 UX 가 통일되었다. 오로지 상세 정보 조회용이기 때문에 UI 도 오로지 조회용으로만 최적화하였고, 연관된 정보들끼리 잘 그룹화하여 필요한 정보를 빠르고 쉽게 찾을 수 있도록 하였다. 또한, 상세 팝업은 꼭 해당 항목의 목록 페이지가 아니더라도 해당 항목의 정보가 노출되는 페이지라면 어디에서든 열어볼 수 있도록 하였다.

여덟째, 요구사항은 아니었지만 빠르게 필요한 정보만을 찾고자 하는 사용자가 충분히 있을 수 있다는 가정을 하고 One-depth 전체 검색 기능을 구현하였다. 이에 따라 기존 웹에서는 할 수 없던, 한 번에 바로 필요한 OSS, 라이선스, 보안 취약점의 정보를 검색할 수 있는 루트가 열리게 되었다. 또한, 마찬가지로 추가로 구현한 대시보드에서는 최근 데이터베이스에 반영된 보안 취약점, OSS, 라이선스의 정보를 조회할 수 있기 때문에 User-friendly Hub 에 접속할 때마다 최신 업데이트 소식을 빠르게 접해볼 수 있게 된다. 본인 프로젝트에 속한 특정 OSS 의 보안 취약점이 발견되었다는 소식을 최대한 빠르게 알 수 있다면 이에 대한 대응도 빨라지기 때문에 더 안전해질 것이다.

10. Division of Work

항목	담당자
REST API 설계 및 개발	김성은 (설계 보조: 최덕경, 한재훈)
CLI 설계 및 개발	한재훈 (설계 보조: 김성은, 최덕경)
User-friendly Hub 설계 및 개발	최덕경 (설계 보조: 김성은)

11. Conclusion

지금 이 시점에도 셀 수 없을 만큼 많은 개발 프로젝트가 진행되고 있을 오늘날, 얼마나 많은 개발자가 오픈 소스 소프트웨어를 올바르게 사용하고 있는지는 확신하기 어렵다. 사실 현실적으로는 저작권이나 라이선스를 엄밀히 준수해서 개발하는 사람보다는 그렇지 않은 사람이 더 많을 것 같으니 말이다. 그렇기에 FOSSLight Hub 와 같은 오픈 소스 소프트웨어 관리 도구가 더 많이 유명해지고 활성화되어야 한다는 생각이 든다.

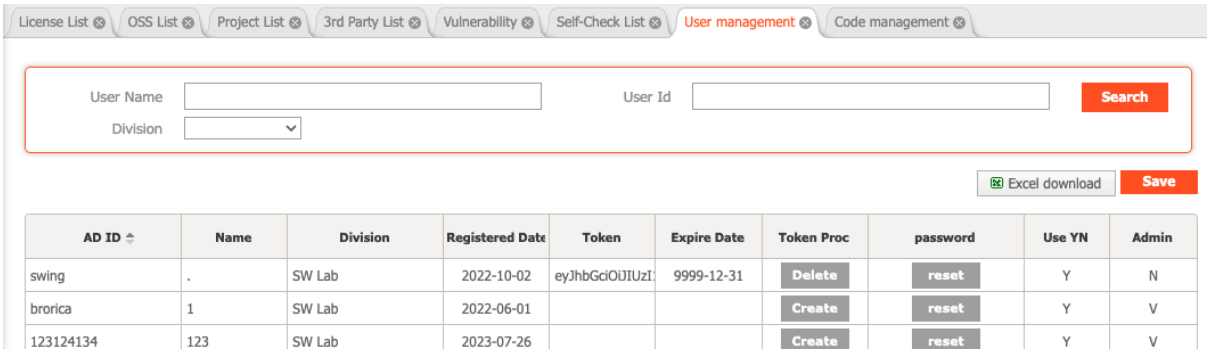
우리는 이번 한 학기 동안 그러한 건강한 개발 생태계를 구축하기 위한 하나의 발판으로서 프로그래밍 인터페이스를 제공하는 CLI 와 사용자 전용으로 특화된 새로운 기술 스택 기반의 User-friendly Hub 를 새롭게 개발하였다. 원래는 CLI 개발만이 요구사항이었던 처음과 달리 무거운 웹사이트를 하

나 신설하는 것까지 일을 늘린 것에 대해 작업량의 부담 때문에 솔직히는 후회도 많이 했지만, 이러한 결과물이 결과론적으로는 대기업에서 실제로 운영하는 서비스로 활용이 된다는 점과 또 이것이 장기적으로는 건강한 OSC 생태계를 구축할 수도 있다는 점을 생각했을 때는 굉장한 보람과 뿌듯함을 느끼게 되는 듯하다. 많이 욕심을 내고 노력한 만큼 우리의 결과물이 적극적으로 상용화되고 흥미로운 관심사가 되어 오픈 소스 소프트웨어의 생태계 전반에 조금이나마 기여가 되었으면 좋겠다.

◆ [Appendix] User Manual (Guide)

A. FOSSLight Hub REST API

API 를 실행하기 위해선 먼저 Authorization token 이 필요하다. 해당 토큰은 기존 FOSSLight Hub 의 User management 페이지에서 확인 가능하다.



Token 을 얻었다면 요청을 호출할 때 Authorization header 에 token 을 추가함으로서 API 들의 접근 권한을 얻게 된다. 단, Project 의 정보 수정과 같은 기능일 경우, 해당 Project 의 소유자, 혹은 watcher 의 token 으로만 권한을 얻을 수 있다. 요청 실행 예시는 다음과 같다.

```
curl
-H "Authorization: {TOKEN}"
-X GET "http://{SERVER_URL}/api/v2/codes?codeType=OS"

# {"content":[{"cdDtlNo":"100","cdDtlNm":"Linux"}, {"cdD-
tlNo":"200","cdDtlNm":"Windows"}, {"cdDtlNo":"300","cdDtlNm":"MacOS"}, {"cdD-
tlNo":"999","cdDtlNm":"ETC"}]}
```

엑셀 파일과 같이 파일을 다운로드 받는 경우 다음과 같이 실행 가능하다.

```
curl
-H "Authorization: {TOKEN}"
-o notice.html "http://{SERVER_URL}/api/v2/projects/10/notice"
```

사용 가능한 API 목록은 다음과 같다. (더 구체적인 파라미터 정보는 Swagger 에서 v2 버전을 선택하면 확인할 수 있다.)

① OSS & License

- GET /api/v2/licenses (라이선스를 검색한다.)
- GET /api/v2/oss (OSS 를 검색한다.)
- POST /api/v2/oss (OSS 를 생성한다. 이름 버전, 라이선스가 필수로 입력되어야 한다.)

② **3rd Party**

- GET /api/v2/partners (3rd Party 를 검색한다.)
- PUT /api/v2/partners/{id}/watchers (3rd Party 에 Watcher 를 추가한다. 이메일의 유효성도 함께 검사한다.)

③ **Project**

- GET /api/v2/projects (프로젝트를 검색한다.)
- POST /api/v2/projects (프로젝트를 생성한다.)
- GET /api/v2/projects/models (프로젝트에 해당하는 모델 정보를 가져온다. 고지할 문서의 종류, 필수로 필요한 운영체제 정보 등의 상수 값은 GET /api/v2/codes 로 확인할 수 있다.)
- PUT /api/v2/projects/{id}/bin (Scanner 의 결과물인 Binary Report 를 업로드한다. 프로젝트의 정보를 추가하는 것이므로 프로젝트의 수정 권한이 있어야 한다. 이는 후술할 다른 업로드/정보 수정 API 도 마찬가지이다.)
- GET /api/v2/projects/{id}/bom/compare-with/{compareId} (입력받은 두 프로젝트의 BOM 을 비교한 결과를 출력한다.)
- GET /api/v2/projects/{id}/bom/export (BOM 을 파일로 다운로드한다.)
- GET /api/v2/projects/{id}/bom/json (BOM 을 JSON 형식으로 출력한다.)
- PUT /api/v2/projects/{id}/models (프로젝트에 모델을 추가한다. 모델 입력은 이름, 카테고리, yyyyMMdd 포맷의 출시 일자를 포함한 문자열 리스트로 받는다.)
- PUT /api/v2/projects/{id}/models/upload (프로젝트에 모델 정보가 담긴 엑셀 파일을 업로드하여 모델을 추가한다.)
- GET /api/v2/projects/{id}/notice (생성된 고지문이 있다면 HTM 형식으로 다운로드한다.)
- POST /api/v2/projects/{id}/packages (공개가 필요한 소스 코드의 파일을 업로드한다.)
- POST /api/v2/projects/{id}/src (Scanner 의 결과물인 Source Report 파일을 업로드한다.)
- POST /api/v2/projects/{id}/watchers (프로젝트에 Watcher 를 추가한다. 3rd Party 와 마찬가지로 이메일의 유효성도 함께 검사한다.)

④ **Vulnerability**

- GET /api/v2/max-vulnerabilities (특정 OSS 의 최대 CVSS 점수를 출력한다.)
- GET /api/v2/vulnerabilities (보안 취약점을 검색한다.)

⑤ **SelfCheck**

- POST /api/v2/selfchecks (SelfCheck 프로젝트를 생성한다.)
- GET /api/v2/selfchecks/{id} (SelfCheck 프로젝트를 가져온다.)
- GET /api/v2/selfchecks/{id}/export (SelfCheck 프로젝트를 Export 한다.)
- PUT /api/v2/selfchecks/{id}/report (SelfCheck 프로젝트에서 Report 를 가져온다.)
- PUT /api/v2/selfchecks/{id}/watchers (SelfCheck 프로젝트에 Watcher 를 추가한다.)

⑥ **Code**

- GET /api/v2/codes (각종 상수 값의 코드 정보를 검색한다. 운영체제 종류, 고지문 종류와 같이 몇 가지 상수로 분류되는 입력값은 이 API 로 조회할 수 있다.)

B. FOSSLight Hub CLI

GitHub: https://github.com/hjcdg1/fosslight/

① 요구사항

Python 3.8+

② 설치

pip install fosslight_cli

③ 실행

터미널에서 fosslight-cli 명령을 실행하려면 다음의 구문을 사용한다.

```
fosslight-cli [command] [resource name] ([sub-resource-name]) [parameters ...]
```

ⓐ command: 수행하려는 동작을 지정한다.

- create
- update
- get
- export
- apply
- compare

ⓑ resource-name: 리소스 이름을 지정한다.

- project
- selfCheck
- config
- partner
- oss
- license
- vulnerability
- maxVulnerability
- yaml

ⓒ sub-resource-name: (일부 명령에서 필요) 하위 리소스의 이름을 지정한다.

- EX 1) fosslight-cli get project list
- EX 2) fosslight-cli update project bin
- EX 3) fosslight-cli get project models

ⓓ parameters: 입력 파라미터들을 지정한다. (필수 or 선택)

④ 명령어

명령어	구문	설명
create		리소스 생성
	fosslight-cli create project --prjName TEXT Name of the Project [required] --osType TEXT OS type of the Project [required] --distributionType TEXT [required] --networkServerType TEXT [required] --priority TEXT [required] --osTypeEtc TEXT --prjVersion TEXT --publicYn TEXT --comment TEXT --userComment TEXT --watcherEmailList TEXT --modelListToUpdate TEXT --modelReportFile TEXT	프로젝트 생성
	fosslight-cli create selfCheck --prjName TEXT Name of the Project [required] --prjVersion TEXT Version of the Project	self-check 생성
update		리소스 수정
	fosslight-cli update project watchers --prjId TEXT project id [required] --emailList TEXT watcher emailList [required]	프로젝트 watcher 변경

	fosslight-cli update project models --prjId TEXT project id [required] --modelListToUpdate TEXT [required]	프로젝트 모델 목록 수정
	fosslight-cli update project modelFile --prjId TEXT project id [required] --modelReport TEXT [required]	모델 파일을 이용하여 프로젝트 모델 목록 수정
	fosslight-cli update project scan --prjId TEXT project id [required] --dir TEXT project directory path [required]	프로젝트 디렉토리를 fosslight scanner 를 이용하여 분석한 후 bin, src 파일 업로드
	fosslight-cli update project bin --prjId TEXT project id [required] --ossReport TEXT --binaryTxt TEXT --comment TEXT --resetFlag TEXT	프로젝트 bin 파일 업로드
	fosslight-cli update project src --prjId TEXT project id [required] --ossReport TEXT --comment TEXT --resetFlag TEXT	프로젝트 src 파일 업로드
	fosslight-cli update project package --prjId TEXT project id [required] --packageFile TEXT [required] --verifyFlag TEXT	프로젝트 package 파일 업로드
	fosslight-cli update selfCheck report --selfCheckId TEXT selfCheck id [required] --ossReport TEXT --resetFlag TEXT	self-check report 파일 업로드

	fosslight-cli update selfCheck watchers --selfCheckId TEXT selfCheck id [required] --emailList TEXT [required]	self-check watcher 변경
	fosslight-cli update partner watchers --partnerId TEXT partner id [required] --emailList TEXT [required]	partner watcher 변경
	fosslight-cli update config -s, --server TEXT Server url -t, --token TEXT Account token	서버, 인증 토큰 설정
get		리소스 가져오기
	fosslight-cli get project list --createDate TEXT --creator TEXT --division TEXT --modelName TEXT --prjIdList TEXT --status TEXT --updateDate TEXT	프로젝트 목록 출력
	fosslight-cli get project models --prjIdList TEXT	프로젝트 모델 목록 출력
	fosslight-cli get license list --licenseName TEXT license name [required]	라이센스 목록 출력
	fosslight-cli get oss list --ossName TEXT oss name [required] --ossVersion TEXT oss version --downloadLocation TEXT download location	oss 목록 출력
	fosslight-cli get partner list --createDate TEXT --creator TEXT --division TEXT	3rd party 목록 출력

	--partnerIdList TEXT --status TEXT --updateDate TEXT	
	fosslight-cli get config	설정된 서버, 인증 토큰 출력
	fosslight-cli get code --codeType TEXT code type [required] --detailValue TEXT detail value	코드 정보 출력
	fosslight-cli get maxVulnerability --ossName TEXT oss name [required] --ossVersion TEXT oss version	max vulnerability 출력
	fosslight-cli get vulnerability --cveId TEXT cve id --ossName TEXT oss name --ossVersion TEXT oss version	vulnerability 정보 출력
export		리소스(주로 파일) 가져오기
	fosslight-cli export project bom --prjId TEXT project id [required] --mergeSaveFlag TEXT mergeSaveFlag -o, --output TEXT output file path	프로젝트 bom 파일 다운로드
	fosslight-cli export project bomJson --prjId TEXT project id [required]	프로젝트 bom 정보를 json 으로 출력
	fosslight-cli export project notice --prjId TEXT project id [required] -o, --output TEXT output file path	프로젝트 고지문 파일 다운로드
	fosslight-cli export selfCheck --selfCheckId TEXT selfCheck id [required]	self-check export
compare		리소스 비교

	fosslight-cli compare project bom --prjId TEXT [required] --compareId TEXT [required]	두 프로젝트의 bom 을 비교
apply		파일에 정의된 동작을 수행
	fosslight-cli apply yaml -f, --file TEXT yaml file path [required]	yaml 파일을 입력으로 받아 적절한 동작 수행

C. User-friendly FOSSLight Hub

GitHub: https://github.com/hjcdg1/fosslight/

① 요구사항

Docker, Docker-compose, Node.js 16+

② 설치 및 실행

FOSSLight Hub 에서 소스 코드를 다운로드하고, Docker Compose 를 이용하여 이미지를 빌드하고 컨테이너를 실행합니다. (docker-compose up --build)

③ 배포 환경 설정

실제로 서비스를 운영하기 위해 배포(Production) 환경을 설정하고 싶은 경우, 루트 디렉토리(docker-compose.yml 파일이 위치한 디렉토리)에 .env 파일을 생성하고 해당 파일 안에 PRODUCT_MODE=true 환경 변수를 정의해야 합니다. 그리고 Lite 웹이 기존 도메인의 하위 도메인을 사용한다고 가정할 때, 웹 서버에서 해당 하위 도메인으로부터 오는 요청들의 경우 3000 번 포트의 컨테이너(= Lite 웹 컨테이너)로 포워딩하고 나머지 요청들은 8180 번 포트의 컨테이너로 포워딩하도록 설정 파일을 수정해야 합니다. 마지막으로, Lite 웹 컨테이너는 배포 환경을 고려하여 설계되어 있기 때문에, 개발 환경에서는 Lite 웹 컨테이너를 끄고 /lite 디렉토리에서 직접 개발 서버를 실행하여(npm run dev) 소스 코드의 수정이 즉각 반영되는 Hot Re-loading 기능을 활용하는 것을 권장합니다.

④ 로그인 및 회원가입

로그인이 되어 있지 않은 사용자는 웹사이트에 접속 시 로그인 페이지로 이동합니다.

1) 로그인 페이지 (/sign-in)

ID 와 비밀번호를 입력하면 로그인을 할 수 있습니다. 기본적으로 관리자 계정(ID: admin, 비밀번호: admin)이 제공되며, 별도의 계정이 필요한 경우 [Sign Up] 버튼을 클릭하여 회원가입 페이지로 이동할 수 있습니다.

2) 회원가입 페이지 (/sign-up)

ID, 비밀번호, 비밀번호 확인, 이메일, 이름을 입력하면 회원가입을 할 수 있습니다. 부서(Division)의 경우 반드시 선택할 필요는 없으며, 필요한 경우 선택하면 됩니다. 동일한 ID

의 계정이 존재하는 경우 회원가입에 실패합니다. 이미 회원가입을 한 계정을 갖고 있는 경우 [Sign Up] 버튼 아래에 위치한 링크를 클릭하여 로그인 페이지로 이동할 수 있습니다.

⑤ 웹사이트 전체 구조

PC 화면(> 768px)의 경우, 로그인한 계정의 기본 정보(이름, 이메일)와 웹사이트의 메뉴들을 보여주는 사이드 바가 좌측에 위치합니다. 현재 보고 있는 페이지에 따라 해당하는 메뉴가 하이라이트로 표시됩니다. 사이드 바 옆의 햄버거 버튼을 클릭하면 사이드 바를 접거나 펼칠 수 있습니다. 상단의 전체 검색 바를 이용하면 원하는 보안 취약점, OSS, 또는 라이선스를 빠르게 찾아볼 수 있습니다.

모바일 화면(≤ 768px)의 경우, 우측 상단의 프로필 아이콘을 클릭하여 로그인한 계정의 기본 정보(이름, 이메일)를 확인할 수 있고 하단 바에 웹사이트의 메뉴들이 위치합니다. 상단의 전체 검색 바를 이용하면 원하는 보안 취약점, OSS, 또는 라이선스를 빠르게 찾아볼 수 있습니다.

⑥ 대시보드 및 전체 검색

대시보드와 전체 검색 페이지는 공통의 UI 를 갖고 있으며, 특정 조건에 맞는 보안 취약점, OSS, 라이선스의 목록을 한눈에 확인할 수 있습니다. 섹션별로 최대 5 개의 항목만 표시하므로, 더 많은 항목을 확인하고 싶은 경우 [show more here] 링크를 클릭합니다. 각 항목의 아래에는 해당 항목이 데이터베이스에 반영(추가 또는 수정)된 날짜 및 시각이 표시됩니다.

각 보안 취약점 항목에는 위험도를 나타내는 CVSS 점수, 해당 보안 취약점이 발견된 OSS 의 이름/버전, 해당 보안 취약점의 고유한 ID 인 CVE ID, 그리고 해당 보안 취약점의 상세 내용이 표시됩니다. CVSS 점수를 클릭하면 보안 취약점 상세 팝업이 떠서 해당 보안 취약점에 관한 더 자세한 정보를 확인할 수 있습니다.

각 OSS 항목에는 해당 OSS 의 이름/버전, (의무사항이 있는 경우) 고지문 또는 소스 코드 공개 의무사항, (보안 취약점이 발견된 경우) 가장 위험도가 높은 보안 취약점의 CVSS 점수, 그리고 해당 OSS 에 적용된 라이선스들의 약칭이 표시됩니다. OSS 의 이름/버전을 클릭하면 OSS 상세 팝업이 떠서 해당 OSS 에 관한 더 자세한 정보를 확인할 수 있습니다.

각 라이선스 항목에는 해당 라이선스의 이름/약칭, (의무사항이 있는 경우) 고지문 또는 소스 코드 공개 의무사항, 그리고 (제한사항이 있는 경우) 해당 라이선스에서 강제하는 제한사항들이 표시됩니다. 라이선스의 이름/약칭을 클릭하면 라이선스 상세 팝업이 떠서 해당 라이선스에 관한 더 자세한 정보를 확인할 수 있습니다.

1) 대시보드 (/)

메인 페이지에 해당하는 페이지입니다. 가장 최근에 데이터베이스에 반영(추가 또는 수정)된 보안 취약점, OSS, 그리고 라이선스의 목록을 한눈에 확인할 수 있습니다.

2) 전체 검색 (/search?keyword={keyword})

상단의 전체 검색 바를 이용하여 검색을 한 결과에 해당하는 페이지입니다. 보안 취약점의 경우 CVE ID 를 기준으로 검색이 이뤄지고, OSS 의 경우 이름, 닉네임을 기준으로 검색이 이뤄지고, 라이선스의 경우 이름, 닉네임, 약칭을 기준으로 검색이 이뤄집니다. 검색 키워드에 매칭되는 부분은 노란 형광색의 하이라이트로 표시됩니다.

⑦ Self-Check

리뷰 과정을 거치지 않고, 스스로 먼저 프로젝트 단위로 검토할 OSS 에 대한 라이선스, 보안 취약점 등의 정보를 확인해보고 이를 바탕으로 소스 코드 파일을 업로드하거나 고지문을 만들어 볼 수 있는 메뉴입니다.

1) Self-Check 목록 (/self-check)

Self-Check 를 위해 생성한 프로젝트들의 목록으로, 로그인한 계정이 생성한 프로젝트들만 표시합니다. 이 페이지 또한 ⑧에서 설명할 목록 페이지의 공통적인 UI 와 기능을 그대로 갖습니다. [Create Project] 버튼으로 Self-Check 프로젝트를 생성할 수 있습니다.

ID: Self-Check 프로젝트를 식별하는 고유한 ID 로, 버전만 다른 것도 다른 프로젝트로 취급합니다.

Name: Self-Check 프로젝트의 이름을 의미합니다.

Version: Self-Check 프로젝트의 버전을 의미합니다.

Report: Self-Check 프로젝트의 OSS 목록에 해당하는 리포트를 엑셀 파일로 다운로드할 수 있는 버튼으로, OSS 목록이 존재하는 경우에만 노출됩니다.

Packages: Self-Check 프로젝트에 업로드된 소스 코드 파일들을 다운로드할 수 있는 버튼들로, 업로드된 소스 코드 파일이 존재하는 경우에만 노출됩니다.

Notice: Self-Check 프로젝트의 OSS 목록을 바탕으로 고지문을 다운로드할 수 있는 버튼으로, OSS 목록이 존재하는 경우에만 노출됩니다. 라이선스가 잘못 기입된 OSS 가 존재하는

경우 고지문을 다운로드할 수 없다는 경고 팝업이 뜨며, 해당 팝업에서 이메일 발송 버튼을 클릭하면 관리자에게 이메일 알림을 발송할 수 있습니다.

Vuln: Self-Check 프로젝트의 OSS 목록에서 발견되는 보안 취약점 중 가장 위험도가 높은 보안 취약점의 CVSS 점수를 표시합니다.

Create: Self-Check 프로젝트를 생성한 날짜를 의미합니다.

2) Self-Check 상세 (/self-check/{project-id})

Self-Check 프로젝트의 기본 정보(이름, 버전, 생성 날짜, 코멘트)를 확인할 수 있고, [Edit] 버튼을 통해 해당 기본 정보를 수정하거나 [Delete] 버튼을 통해 해당 프로젝트를 삭제할 수 있습니다. 그리고 그 아래부터는 Self-Check 프로세스에 해당하는 3 단계가 탭으로 구성되어 있습니다. 기본적으로는 각 단계를 순차적으로 거치는 것이 원칙이며, 각 단계에 대한 설명은 다음과 같습니다.

2-1) OSS 탭

해당 프로젝트에 속한 OSS 의 목록을 기입합니다. 좌측 상단의 [+] 버튼을 클릭하여 수동으로 기입할 수도 있고, FOSSLight Scanner 의 분석 결과에 해당하는 리포트 파일을 업로드함으로써 한 번에 여러 OSS 를 기입할 수도 있습니다. 삭제하고 싶은 OSS 를 선택하고 좌측 상단의 휴지통 아이콘을 클릭하면 삭제가 되며, 각 OSS 의 연필 아이콘을 클릭하면 OSS 의 정보를 수정할 수 있습니다. 이러한 OSS 의 추가/수정/삭제 변동사항이 있는 경우 'OSS List' 텍스트 옆에 빨간 점이 표시되며, [Save] 버튼을 클릭하면 그러한 변동사항을 한 번에 저장할 수 있습니다.

OSS 의 이름/버전 또는 라이선스가 잘못 기입된 경우, 해당하는 부분에 빨간색의 경고 아이콘이 표시됩니다. 해당 아이콘에 마우스를 갖다 대면 어떤 종류의 경고인지 확인할 수 있습니다.

OSS 목록의 상단에 위치한 검색/정렬 기능을 사용하면 원하는 OSS 를 검색하거나 OSS 목록을 특정 기준으로 정렬할 수 있습니다. 그리고 [Export] 버튼을 클릭하면 Self-Check 목록 페이지에서 Report 열의 버튼을 클릭했을 때와 마찬가지의 리포트(엑셀 파일)를 다운로드할 수 있습니다.

[Check] 버튼을 클릭하면 OSS 의 이름이 잘못 기입된 경우 또는 라이선스들이 잘못 기입된 경우를 확인할 수 있습니다. OSS 의 이름은 동일한 Download URL 을 갖는 OSS 가 데이터베이스에 등록되어 있을 때, 라이선스들은 동일한 OSS 이름/버전에 매

칭된 라이선스들이 데이터베이스에 등록되어 있을 때 검사가 이뤄집니다. 원하는 OSS 를 체크하고 [Change OSS Name] 버튼을 클릭하면 자동으로 해당하는 OSS 의 이름이 일괄 수정되며, 원하는 라이선스를 체크하고 [Change Licenses] 버튼을 클릭하면 자동으로 해당하는 OSS 의 라이선스들이 일괄 수정됩니다.

2-2) Package 탭

OSS 탭에서 저장([Save] 버튼)이 이뤄진 경우에만 이 탭으로 이동할 수 있습니다. 소스 코드 공개가 필요한 OSS 의 목록을 표시하며, 해당 목록에 표시된 OSS 의 소스 코드 파일을 업로드해볼 수 있습니다. zip, tar.gz, gz 파일 형식만 지원합니다.

2-3) Notice 탭

OSS 탭에서 저장([Save] 버튼)이 이뤄진 경우에만 이 탭으로 이동할 수 있습니다. 사용자가 고지문 커스터마이징 여부를 선택하고 그것에 따라 원하는 고지문을 생성하거나 미리보기할 수 있습니다. 단, OSS 탭에서 라이선스가 잘못 기입된 경우(라이선스 부분에 경고 아이콘이 표시된 경우)에는 고지문의 생성 및 미리보기가 불가능하며, 이때는 경고 팝업이 뜨고 해당 팝업에서 이메일 발송 버튼을 클릭하여 관리자에게 이메일 알림을 발송할 수 있습니다.

⑧ Database

데이터베이스에 등록되어 있는 OSS, 라이선스, 보안 취약점들의 정보를 확인할 수 있는 메뉴입니다. 기본적으로 각 항목(OSS, 라이선스, 보안 취약점)의 목록 페이지는 공통적으로 검색을 위한 필터링 기능, 특정 열을 기준으로 한 정렬 기능(해당 열 클릭), 원하는 열만 표시하기 위한 열 선택 기능(눈동자 모양의 아이콘), 목록의 내용을 엑셀 파일로 다운로드할 수 있는 기능([Export] 버튼)을 갖고 있습니다. 또한 각 열을 클릭하는 경우 해당 항목의 상세 팝업이 떠서 해당 항목에 관한 더 자세한 정보를 확인할 수 있습니다.

1) OSS 목록/상세 (/database/oss)

ID: OSS 를 식별하는 고유한 ID 로, 버전만 다른 것도 다른 OSS 로 취급합니다.

Name: OSS 의 이름을 의미합니다.

Nickname: OSS 의 닉네임으로, 한 OSS 가 여러 닉네임을 가질 수 있습니다.

Version: OSS 의 버전을 의미합니다.

Type: 적용된 라이선스들의 종류에 따라 결정되며, M(Multi License), D(Dual License), V(Version different License) 플래그가 설정됩니다. 각각의 의미는 기존 FOSSLight Hub 와 동일합니다.

License Type: 적용된 라이선스들의 종류에 따라 결정되며, Permissive, Weak Copyleft, Copyleft, Proprietary, Proprietary Free 중 하나로 설정됩니다. 각각의 의미는 기존 FOSS-Light Hub 와 동일합니다.

License Name: 적용된 라이선스들의 이름으로, 목록의 테이블에는 약칭으로 표시합니다.

Obligation: 적용된 라이선스들의 종류에 따라 결정되며, 고지문 또는 소스 코드 공개 의무사항을 의미합니다.

Download URL: OSS 를 다운로드하기 위한 URL 을 의미합니다.

Homepage URL: OSS 의 공식 홈페이지에 해당하는 URL 을 의미합니다.

Description: OSS 에 관한 설명을 의미합니다.

Copyright: OSS 에 적용된 저작권의 내용을 의미합니다.

Attribution: 고지문 발행 시 별도로 포함해야 하는 내용을 의미합니다.

Vulnerability: OSS 에서 발견된 보안 취약점들의 목록으로, 목록의 테이블에는 그중에서 가장 위험도가 높은 보안 취약점의 CVSS 점수만 표시합니다.

Create/Modify: OSS 의 등록/수정 정보(날짜, 행위 주체)를 의미합니다.

2) 라이선스 목록/상세 (/database/license)

ID: 라이선스를 식별하는 고유한 ID 를 의미합니다.

Name: 라이선스의 이름을 의미합니다.

Nickname: 라이선스의 닉네임으로, 한 라이선스가 여러 닉네임을 가질 수 있습니다.

(SPDX) Identifier: 라이선스의 약칭으로, SPDX 표기 방식을 따릅니다.

Type: Permissive, Weak Copyleft, Copyleft, Proprietary, Proprietary Free 중 하나로 설정됩니다. 각각의 의미는 기존 FOSSLight Hub 와 동일합니다.

Obligation: 고지문 또는 소스 코드 공개 의무사항을 의미합니다.

Restriction: 라이선스에서 강제하는 제한사항으로, 종류로는 Non-Commercial Use Only, Network Redistribution, Restricted Modifications, Platform Deployment Restriction, Prohibited Purpose, Specification Constraints, Restricted Redistribution, Common Clause Restriction 이 있습니다. 각각의 의미는 기존 FOSSLight Hub 와 동일합니다.

Homepage URL: 라이선스의 공식 홈페이지에 해당하는 URL 입니다.

Description: 라이선스에 관한 설명으로, OSS 사용 시의 주의사항을 나타냅니다.

License Text: 라이선스의 원문을 의미합니다.

Attribution: 고지문 발행 시 별도로 포함해야 하는 내용을 의미합니다.

Create/Modify: 라이선스의 등록/수정 정보(날짜, 행위 주체)입니다.

3) 보안 취약점 목록/상세 (/database/vulnerability)

OSS: 보안 취약점이 발견된 OSS 들의 이름/버전을 의미합니다. 하나의 보안 취약점은 여러 개의 OSS 에서 발견될 수 있으며, 목록의 테이블에는 (OSS, 보안 취약점)의 모든 쌍이 표시됩니다.

CVE ID: 보안 취약점을 식별하는 고유한 ID 를 의미합니다.

CVSS Score: 보안 취약점의 위험도를 의미하는 점수입니다.

Summary: 보안 취약점에 관한 요약된 설명을 의미합니다.

Modify: 보안 취약점이 등록/수정된 날짜를 의미합니다. 보안 취약점의 정보는 매일 자동으로 NVD Data Feed 에서 다운로드되어 데이터베이스에 반영됩니다.