## 1. Executive Summary & Project Vision

### 1.1. 프로젝트 비전

본 프로젝트의 최종 목표는 명령줄 인터페이스(CLI)에 대한 깊은 지식 없이도 네트워크 엔지니어가 Cisco 기반의 스위치, 라우터, 무선 LAN 컨트롤러(WLC) 등 다양한 장비의 구성을 **표준화되고, 일관되며, 시각적으로** 관리할 수 있는 전문 데스크톱 애플리케이션을 구축하는 것입니다.

이 애플리케이션은 사용자가 GUI를 통해 원하는 설정값을 입력하면, 내부적으로 대상 장비의 OS 유형(IOS-XE, NX-OS 등)에 맞는 **Ansible 플레이북을 동적으로 생성**하고, 이를 통해 구성 배포를 자동화하는 것을 핵심 기능으로 합니다. 이를 통해 수동 구성 작업에서 발생하는 인적 오류를 최소화하고, 조직의 표준 구성 정책을 모든 장비에 일관되게 적용하여 네트워크의 안정성과 보안을 극대화하는 것을 목표로 합니다.

### 1.2. 핵심 전략

제공된 표준 구성 문서(CSV)는 파싱의 대상이 아니라, 전문가의 업무 흐름과 지식이 담긴 **설계 청사진**으로 활용합니다. 문서에 명시된 모듈 -> 서브모듈 계층 구조를 심도 있게 분석하여, 이를 애플리케이션의 UI 구조와 플레이북 생성 로직에 직접 반영합니다.

## 2. 현재 프로젝트 상황 및 리포지토리 구조

### 2.1. 현재 상황 분석

우리는 현재 프로젝트의 기본 골격을 갖춘 초기 단계에 있습니다.

* **UI 프레임워크:** PySide6를 기반으로 한 기본 애플리케이션 창(QMainWindow)이 구현되어 있습니다.
* **핵심 아키텍처:** UI(ui), 핵심 로직(core), Ansible 자원(ansible)으로 분리된 기본적인 3계층 구조가 수립되어 있습니다.
* **실행 엔진:** Windows 환경에서의 원활한 개발을 위해, 실제 Ansible을 호출하지 않는 **모의(Mock) 실행 엔진** (ansible\_engine.py)이 구현되어 있습니다. 이는 생성된 플레이북의 내용을 로그 창에 출력하여 즉각적인 피드백을 제공합니다.
* **과제:** 현재 UI는 기능 확장을 고려하지 않은 임시 구조이며, 플레이북 생성 로직 또한 초기 단계에 머물러 있습니다. 이제 이 기반 위에 표준 문서의 지식을 체계적으로 이식하여 전문적인 애플리케이션으로 발전시켜야 합니다.

### 2.2. 최종 리포지토리 구조

향후 개발은 아래의 구조를 기준으로 진행됩니다. CSV 파싱 관련 스크립트와 폴더는 제거되고, 전문가의 지식이 담긴 정적 UI와 지능적인 플레이북 생성기에 집중합니다.

| 경로 | 설명 |
| --- | --- |
| ansible\_config\_editor/ | 메인 애플리케이션 소스 코드 디렉토리. |
| ├── main.py | 애플리케이션 진입점. MainWindow를 초기화하고 실행합니다. |
| ├── ui/ | GUI 관련 모듈 (View). |
| │ └── main\_window.py | **(핵심 개발 영역)** QMainWindow를 상속받아, 표준 문서의 계층 구조를 반영한 **정적 다중 탭 UI**를 구현합니다. 모든 사용자 입력을 수집하는 역할을 담당합니다. |
| ├── core/ | 핵심 애플리케이션 로직 (Controller & Model). |
| │ ├── playbook\_manager.py | **(핵심 개발 영역)** ConfigManager 클래스를 통해, GUI에서 수집된 구조화된 데이터로부터 OS 유형에 맞는 Ansible 플레이북을 동적으로 생성합니다. |
| │ └── ansible\_engine.py | **(유지)** Windows 호환성을 위한 모의(Mock) 엔진. 개발 중 플레이북 생성 결과를 즉시 확인하는 데 사용되며, 최종 단계에서 실제 실행 엔진으로 교체될 수 있습니다. |
| ansible/ | Ansible 관련 파일 디렉토리. |
| ├── inventory/ | 대상 장비 목록을 정의하는 hosts.ini 파일 위치. |
| ├── playbooks/ | (참고용) discover\_device.yml 등 정적 플레이북 위치. |
| requirements.txt | PySide6, pyyaml 등 Python 패키지 의존성을 명시합니다. |

## 3. 아키텍처 상세 설계

애플리케이션의 성공은 견고하고 확장 가능한 아키텍처에 달려 있습니다. 우리는 '관심사 분리(Separation of Concerns)' 원칙에 따라 시스템을 세 가지 핵심 구성 요소로 설계합니다.

### 3.1. UI 계층 (ui/main\_window.py)

* **역할:** 사용자로부터 모든 구성 데이터를 입력받는 인터페이스입니다.
* **설계 철학:** 제공된 표준 구성 문서의 모듈 -> 서브모듈 계층 구조를 심도 있게 분석하여, 전문가의 작업 흐름에 맞는 **정적 다중 탭(Multi-Tab) UI**를 설계합니다. 이는 사용자가 마치 잘 정리된 기술 문서를 보며 작업하는 듯한 직관적인 경험을 제공합니다.
  + **메인 탭 (Modules):** Global, Interface, VLAN, Routing, HA, Security 등 표준 문서의 최상위 모듈을 QTabWidget으로 구현하여 기능별로 명확히 분리합니다.1
  + **그룹 박스 (Sub-modules):** 각 탭 내부에서는 Hostname, Logging, AAA, vPC 등 하위 모듈을 QGroupBox로 구현하여 관련 설정들을 시각적으로 그룹화합니다.1
  + **입력 위젯:** 각 설정 항목의 특성에 맞춰 QLineEdit(단일 행), QPlainTextEdit(여러 행), QCheckBox(On/Off), QComboBox(선택), QTableWidget(목록형 데이터) 등 최적의 위젯을 사용합니다.
* **데이터 흐름:** 사용자가 '구성 적용' 버튼을 클릭하면, UI 계층은 모든 탭과 위젯의 현재 상태를 읽어 구조화된 단일 Python 딕셔너리(Dictionary) 형태로 core 계층에 전달합니다.

### 3.2. 핵심 로직 계층 (core/playbook\_manager.py)

* **역할:** 애플리케이션의 "두뇌"입니다. UI로부터 받은 데이터와 사용자가 선택한 OS 유형을 조합하여 실제 Ansible 플레이북을 생성하는 모든 지능적인 작업을 처리합니다.
* **설계 철학:** ConfigManager 클래스는 **OS 유형에 따른 조건부 로직**을 사용하여 플레이북을 동적으로 생성합니다.
  + **입력:** os\_type (예: "L3\_IOS-XE") 문자열과 UI에서 수집된 config\_data 딕셔너리를 입력받습니다.
  + **처리:** \_generate\_global\_tasks, \_generate\_interface\_tasks 등 모듈별 헬퍼 함수를 호출합니다. 각 함수 내부에서는 if "IOS-XE" in os\_type: 또는 if "NX-OS" in os\_type: 과 같은 조건문을 사용하여 다음과 같은 OS별 차이를 처리합니다.
    - **기능 활성화:** NX-OS의 feature lacp 1 와 같이 특정 OS에서만 필요한 명령어를 추가합니다.
    - **모듈 선택:** OS에 맞는 Ansible 모듈(예: cisco.ios.ios\_config 2 vs  
      cisco.nxos.nxos\_config)을 선택합니다.
    - **구문 차이:** vrf forwarding Mgmt-intf (IOS-XE) 1 vs  
      vrf member management (NX-OS) 1 와 같은 명령어 구문의 차이를 반영합니다.
  + **출력:** Ansible이 즉시 실행할 수 있는 완전한 플레이북 구조(Python 딕셔너리)를 반환합니다.

### 3.3. 실행 계층 (core/ansible\_engine.py)

* **역할:** 생성된 플레이북을 대상 장비에서 실행하는 역할을 담당합니다.
* **현재 상태 (모의 실행):** Windows 환경에서의 개발 편의성을 위해, 현재는 실제 Ansible을 실행하지 않는 **모의(Mock) 엔진**으로 구현되어 있습니다. 이 엔진은 생성된 플레이북의 내용을 로그 창에 출력하여, 사용자가 UI 입력이 어떻게 플레이북으로 변환되는지 즉시 확인할 수 있게 해줍니다.
* **향후 계획:** 실제 배포 단계에서는 이 모듈을 ansible-runner 라이브러리를 사용하는 실제 실행 엔진으로 교체할 수 있습니다. (Windows에서 실행 시 WSL 환경 필요) 3

## 4. 개발 로드맵: 단계적 기능 확장 계획

현재 우리는 견고한 아키텍처의 기반을 다졌습니다. 앞으로 다음 단계를 통해 애플리케이션을 완성해 나갈 것입니다.

### 1단계: 핵심 모듈 상세 구현 (현재 ~ 3개월)

현재 UI의 여러 탭이 "구현 예정" 상태입니다. 이 단계에서는 표준 문서를 집중적으로 참고하여 가장 중요하고 복잡한 모듈들의 UI와 플레이북 생성 로직을 완성합니다.

* **Interface 모듈:**
  + 물리적 인터페이스, Port-Channel, SVI(VLAN Interface)를 구분하여 설정할 수 있는 UI 구현.
  + Access 모드와 Trunk 모드 선택에 따라 관련 옵션(Access VLAN, Allowed VLANs 등)이 동적으로 활성화/비활성화되는 UI 로직 추가.
  + spanning-tree portfast edge 1,  
    udld port aggressive 1 등 인터페이스별 세부 옵션 추가.
* **VLAN 모듈:**
  + VLAN ID와 이름을 목록으로 관리하는 QTableWidget 구현.
  + SVI 생성 시 IP 주소, VRRP, DHCP Helper-address 1 등 상세 설정 UI 추가.
* **Routing 모듈:**
  + Static, OSPF, BGP 등 프로토콜별 하위 탭 구현.
  + OSPF 설정 시 Process ID, Router ID, 네트워크 선언, 재분배(Route-map) 등 전문가용 옵션 제공.1
* **HA (고가용성) 모듈:**
  + OS 선택에 따라 StackWise Virtual (IOS-XE) 1 또는  
    vPC (NX-OS) 1 설정 UI가 조건부로 표시되도록 구현.

### 2단계: 고급 기능 및 편의성 향상 (3 ~ 6개월)

핵심 기능이 완성되면, 사용자의 편의성을 높이고 실제 운영 환경에 필요한 고급 기능을 추가합니다.

* **구성 저장 및 불러오기:** 현재 GUI에 입력된 모든 값을 단일 프로젝트 파일(예: .yaml 또는 .json)로 저장하고, 나중에 다시 불러와 작업을 이어갈 수 있는 기능 구현.
* **변수 관리 시스템:** `` 1 와 같이 반복적으로 사용되는 변수(사이트 코드, 건물, 층 등)를 별도의 UI에서 정의하고, 각 설정 필드에서 참조할 수 있는 기능 추가.
* **입력 유효성 검사:** IP 주소, VLAN 범위, ACL 규칙 등의 입력 형식이 올바른지 실시간으로 검증하여 사용자 실수를 방지.
* **"구성 가져오기" 기능:** cisco.ios.ios\_facts 5 와 같은 Ansible 모듈을 활용하여, 기존 장비의 현재 설정을 읽어와 GUI에 자동으로 채워주는 기능 구현.

### 3단계: 실제 배포 및 운영 준비 (6개월 이후)

애플리케이션을 실제 운영 환경에서 사용할 수 있도록 최종 단계를 진행합니다.

* **실제 Ansible 엔진 연동:** ansible\_engine.py를 ansible-runner 3를 사용하는 실제 실행 모듈로 교체하고, Windows 사용자를 위한 WSL 환경 설정 가이드 제공.
* **실행 전 미리보기 (Dry-Run):** Ansible의 --check 모드를 활용하여, 실제 구성을 적용하기 전에 어떤 변경이 발생할지 미리 보여주는 기능 추가.
* **상세 로깅 및 이력 관리:** 모든 실행 작업(누가, 언제, 어떤 장비에, 어떤 설정을 적용했는지)을 로그 파일이나 데이터베이스에 기록하여 감사 추적 기능 제공.
* **패키징 및 배포:** PyInstaller와 같은 도구를 사용하여 애플리케이션을 단일 실행 파일로 패키징하여, Python이나 Ansible이 설치되지 않은 환경에서도 쉽게 사용할 수 있도록 배포.

이 로드맵을 통해 우리는 단순한 플레이북 편집기를 넘어, Cisco 네트워크 환경 전반의 구성 관리를 혁신하는 강력하고 안정적인 전문 자동화 플랫폼을 구축할 수 있을 것입니다.

#### 참고 자료

1. Coupang Standard Configuration\_Ringnet\_ver 250905.xlsx
2. cisco.ios.ios\_config module – Module to manage configuration sections. — Ansible Community Documentation, 9월 24, 2025에 액세스, <https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_config_module.html>
3. Using Runner as a Python Module Interface to Ansible, 9월 24, 2025에 액세스, <https://ansible.readthedocs.io/projects/runner/en/stable/python_interface/>
4. ansible-runner Documentation - Read the Docs, 9월 24, 2025에 액세스, <https://readthedocs.org/projects/ansible-runner/downloads/pdf/latest/>
5. cisco.ios.ios\_facts module – Module to collect facts from remote devices. — Ansible Community Documentation, 9월 24, 2025에 액세스, <https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_facts_module.html>