**✅ Module 1: Project Setup & Virtual Environment**

**🧠 What is this module about?**

This module lays the **foundation** of your entire automation project. Just like you can’t build a house without a strong base, you can’t build a Python-based automation system without:

1. A clean project structure
2. An isolated environment to run your code
3. Version control for collaboration and history tracking

**🔍 1. What is a Virtual Environment and Why is it Needed?**

**💡 Real-life Analogy:**

Imagine you are a chef who works in multiple restaurants. Each restaurant has different ingredients and tools. To avoid mixing up recipes, each kitchen keeps its tools and ingredients separate.  
**In software, a virtual environment is your isolated kitchen.**

**🧪 Technical Explanation:**

* A virtual environment is a **self-contained directory** that has its own Python executable and libraries.
* It prevents **dependency conflicts** between different projects.

**✅ Benefits:**

| **Feature** | **Benefit** |
| --- | --- |
| Isolation | One project's libraries won’t break another’s |
| Portability | Easily share project with others via requirements.txt |
| Control | You decide what versions of packages are used |

**🔧 2. How to Create a Virtual Environment (Step-by-step)**

**✅ Step 1: Create the project directory**

bash

CopyEdit

mkdir network-api-testing

cd network-api-testing

**✅ Step 2: Initialize a virtual environment**

bash

CopyEdit

python -m venv venv

* python -m venv: Python’s built-in tool for creating virtual environments
* venv: The folder name where the environment will be stored

You will now see a folder structure like:

CopyEdit

network-api-testing/

├── venv/

**✅ Step 3: Activate the virtual environment**

* On macOS/Linux:

bash

CopyEdit

source venv/bin/activate

* On Windows:

bash

CopyEdit

venv\Scripts\activate

Once activated, your terminal will show something like:

bash

CopyEdit

(venv) user@machine:~/network-api-testing$

**📦 3. Installing Required Python Packages**

At this point, you’ll start installing the tools required for your project:

bash

CopyEdit

pip install requests pytest pytest-html jsonschema

**✅ What do these do?**

| **Package** | **Purpose** |
| --- | --- |
| requests | Makes HTTP calls to Cisco SD-WAN APIs |
| pytest | Test framework for running API test cases |
| pytest-html | Generates visual HTML reports from test results |
| jsonschema | Validates API response structure against a JSON schema |

You’ll later freeze these dependencies:

bash

CopyEdit

pip freeze > requirements.txt

Now your directory will contain:

CopyEdit

requirements.txt

This file helps you or anyone else reproduce the exact same environment later with:

bash

CopyEdit

pip install -r requirements.txt

**🗂️ 4. Setting Up Project Folder Structure**

We start organizing the project like this:

graphql

CopyEdit

network-api-testing/

├── venv/ # Virtual environment folder

├── auth.py # Handles authentication with the API

├── api\_helper.py # Reusable API call functions

├── test\_inventory.py # Pytest test cases

├── requirements.txt # Python dependencies

As you proceed with later modules, this will expand to include:

* utils/ for helpers
* schemas/ for JSON schema files
* .github/workflows/ for CI/CD setup

**🔁 5. Using Git for Version Control**

**💡 Why use Git?**

* Tracks every change you make in code
* Allows you to collaborate with others
* Helps you roll back if something breaks

**✅ Commands used:**

bash

CopyEdit

git init # Start version control

git add . # Stage all files

git commit -m "Initial commit" # Save your changes

You then push this project to GitHub using:

bash

CopyEdit

git remote add origin https://github.com/yourname/network-api-testing.git

git push -u origin main

**🏁 Summary of Module 1**

| **Component** | **Explanation** | **Benefit** |
| --- | --- | --- |
| Virtual Environment | Isolated Python workspace | Prevents dependency conflicts |
| Folder Structure | Organized codebase | Scalable project setup |
| Dependency Management | Via requirements.txt | Reproducible builds |
| Git Integration | Tracks your work | Safe code versioning |
| Ready for CI/CD | Lays groundwork for automation | Enables Module 10: GitHub Actions |

**🎯 Outcome After Module 1:**

You are ready to start interacting with the Cisco SD-WAN API with:

* Clean Python environment ✅
* Organized code folder ✅
* Tracked version control via GitHub ✅

**✅ Module 2: API Authentication and Device Inventory Fetch (Cisco SD-WAN Sandbox)**

**🔰 Objective (Layman's Terms)**

We want to:

* Log in to Cisco’s SD-WAN system **programmatically** (using Python).
* Fetch a list of **network devices** (routers, switches, etc.) from the cloud dashboard.
* This is like logging into a website and getting a table of all devices — but with **code**.

**🧠 What Are We Learning?**

* How to **authenticate** to a REST API using **session-based login** (not just simple tokens).
* How to use requests to make real API calls to **Cisco vManage** SD-WAN dashboard.
* Parse and print **inventory JSON** (list of all network devices managed by Cisco SD-WAN).
* Learn basics of **API endpoints**, **session cookies**, and **GET requests**.

**🔧 Technical Concepts Introduced**

| **Concept** | **Explanation** |
| --- | --- |
| requests.session() | Keeps cookies alive across requests (just like browsers). |
| /j\_security\_check | Cisco SD-WAN's login endpoint (takes username/password). |
| session.get() | Used to call another API after logging in — to get the inventory. |
| verify=False | Skips SSL cert checks (only okay in sandbox/test environments). |
| json() | Converts API responses to Python dictionaries for easy access. |
| Exception Handling | Raises errors if login fails (helps debug authentication issues). |

**🧱 What We Built**

**🔹 get\_inventory.py**

This Python script does three things:

1. **Authenticates** to Cisco vManage using j\_username and j\_password via POST to /j\_security\_check.
2. **Checks** that login was successful by verifying the presence of a cookie named JSESSIONID.
3. **Fetches** the device inventory by sending a GET request to:

bash

CopyEdit

https://sandbox-sdwan-2.cisco.com/dataservice/device

1. **Prints out** a list of devices (e.g., vEdge, vSmart, WAN edge routers).

**🔐 Authentication Flow**

pgsql

CopyEdit

POST /j\_security\_check → Get JSESSIONID cookie → Use session to send authenticated GET

* This mimics **logging into the Cisco SD-WAN dashboard manually**, but fully automated.

**📁 Files Created**

| **File** | **Purpose** |
| --- | --- |
| get\_inventory.py | Main driver script for login + device fetch |
| auth.py (moved later to separate) | Was inline here originally; later modularized |

**🧪 Example Output (JSON Extract)**

json

CopyEdit

{

"deviceId": "10.10.22.33",

"host-name": "vEdge1",

"device-type": "vedge",

"system-ip": "10.0.0.1"

}

You fetch **real hardware/software details** from a Cisco cloud dashboard — valuable data for monitoring, testing, or automation.

**💡 Why Is This Module Important?**

| **Reason** | **Impact** |
| --- | --- |
| ✅ Foundation | Authentication is **step 1** of any secure API integration. |
| ✅ Real-world | Uses a **real Cisco sandbox**, not dummy data. |
| ✅ Reusable | Same auth logic reused across all future tests. |
| ✅ Testable | Can now **automate validation** of network devices — core to SDET + DevNet roles. |

**🧰 Technologies Used**

| **Tool** | **Use** |
| --- | --- |
| Python | Programming language |
| requests | HTTP library for API calls |
| Cisco SD-WAN Sandbox | Real API platform |
| JSON | Data format used for communication |
| Terminal | Running scripts manually or in CI |

**🎯 Outcomes**

You can now:

* Log in to Cisco SD-WAN with code.
* Query inventory data using secure API sessions.
* Understand what APIs are, how to authenticate, and how to consume endpoints — like a DevNet professional.