# InTune Device Compliance Reporter

Contents

[InTune Device Compliance Reporter 1](#_Toc770849444)

[Introduction 1](#_Toc1351719846)

[1. Initial Setup 1](#_Toc134948029)

[1.1. Local Repository 1](#_Toc644524068)

[1.2. Version Control 3](#_Toc2055972151)

[2. Requirement Gathering 5](#_Toc34451683)

[2.1. Business Problem 5](#_Toc2140610608)

[2.2. Objective 7](#_Toc236596952)

[2.3. Stakeholder Engagement 7](#_Toc367454888)

[2.4. Functional Requirements 7](#_Toc617228680)

[2.5. Technical Requirements 8](#_Toc102070695)

[2.6. Constraints and Assumptions 8](#_Toc22224706)

[3. Analysis and Initial Design 8](#_Toc2068401745)

[3.1. Introduction 8](#_Toc1092463356)

[3.2 MoSCoW 8](#_Toc1630858078)

[3.3. Risks and Constraints 9](#_Toc86750437)

[3.4. Identify System Inputs, Processes, and Outputs 10](#_Toc172718059)

[3.5. High-Level Architecture 10](#_Toc881472053)

[3.6. Database Design 11](#_Toc769629749)

[3.7 Chosen Technologies 12](#_Toc948262663)

[4. Development 12](#_Toc1604946304)

[4.1. Project Setup and Environment 12](#_Toc123899526)

[4.2. API Connection to Microsoft Graph 13](#_Toc889823971)

[4.3 Storing Credentials Securely with .env and .gitignore 15](#_Toc131578176)

[4.4. Fetching and Parsing Device Data 17](#_Toc1999295276)

[4.5. Writing Data to SQLite 17](#_Toc1612523462)

[4.6. Generating Compliance Reports 17](#_Toc893051962)

[4.7. Optional Enhancements (e.g. Notifications or Scheduling) 17](#_Toc2051821289)

## Introduction

This project, *Intune Device Compliance Reporter*, was initiated as part of the **A1.4 Apply** stage of my Level 4 Software Engineer apprenticeship with QA. It aims to automate the process of retrieving, processing, and reporting on device compliance data for Windows and iOS devices managed via Microsoft Intune- a cloud-based endpoint management solution used by Assura to secure and monitor company devices.

Currently, reporting on device compliance is a **manual, time-consuming task** performed periodically by the IT team. This project will reduce administrative overhead, improve accuracy, and provide timely insights to enable faster IT triage.

My goal is to develop a maintainable, testable, and structured Python-based tool that retrieves compliance data via the Microsoft Graph API, stores it in an SQL database, and outputs clear reports for IT analysis.

The project directly benefits my organisation while also allowing me to evidence key apprenticeship learning outcomes:

* **S1:** Writing logical and maintainable code
* **S4:** Unit testing and debugging
* **S7:** Structured problem solving
* **B4:** Collaborating with internal teams
* **B10:** Continuing professional development through real-world tooling and automation

## 1. Initial Setup

### 1.1. Local Repository

To begin, I designed a clean and modular folder structure to support separation of concerns during development. Each folder has a clearly defined role.

A screenshot of a computer

AI-generated content may be incorrect.

#### Folder-by-Folder Review

|  |  |
| --- | --- |
| **Folder** | **Purpose** |
| Database/ | Storage for my .db file (SQLite) and SQL schema files. |
| Docs/ | Great for requirement gathering notes, design diagrams, and SDLC write-up drafts. |
| Reports/ | Ideal for storing auto-generated output: CSVs, charts (from matplotlib), or summary logs. |
| Screenshots/ | Useful to evidencing progress along the way (e.g., API calls, testing, terminal outputs, debugging, stakeholder communication). |
| Scripts/ | My code will live here with separate modules for fetching, transforming, storing, etc. |
| Tests/ | Where I will write unit tests testing and debug my code. Could use pytest at a later point. |

#### README.md File Creation

A markdown-based README file was added to the root of the project as well as to each sub-folder. It includes a brief overview of the project’s purpose (or folder’s purpose), the technologies used, and how the solution aligns with my apprenticeship learning outcomes (S1, S4, S7, B4, B10). This also serves as documentation for anyone reviewing the code or assessing the project via GitHub.

A screenshot of a computer program

AI-generated content may be incorrect.

### 1.2. Version Control

To manage this project and demonstrate good version control practices, Git and GitHub were used. Git was used locally to initialise and track changes to the project, while GitHub served as the remote repository, allowing for secure cloud-based backup and versioning.

As part of this project:

* The default branch was renamed to **main**, following modern Git conventions.
* Commits were made regularly using **clear, meaningful messages** to describe changes.
* Each core folder (e.g. Docs, Scripts, Tests) includes a placeholder README.md file to ensure they are tracked from the start.

#### GitHub Repository

I created a new **private GitHub repository** titled **intune-device-compliance-reporter**. I made the repository private as it will include sensitive information such as client secrets from Azure.

A screenshot of a computer

AI-generated content may be incorrect.

The repository was created successfully and provided with me the HTTPS address to include in my Git commands.

A screenshot of a computer

AI-generated content may be incorrect.

#### Connecting Local Repository to GitHub

The newly created repository was linked to my local folder using Git, and the initial commit (including the full folder structure and README) was pushed successfully. This ensures all future code changes are tracked and version controlled, supporting maintainability and collaboration.

To do this, first, I navigated to the Root folder and opened Terminal (PowerShell). A screenshot of a computer

AI-generated content may be incorrect.

I then executed the following Git command block. I have added clear inline comments to explain what each line does and why it's useful.

**# Initialise a new Git repository in the current local folder**

git init

**# Stage (add) all files and folders in the current directory for the next commit**

git add .

**# Create your first commit with a message describing the changes**

git commit -m "Initial commit with folder structure and README"

**# Rename the default branch from 'master' to 'main' (main is now industry standard)**

git branch -M main

**# Link your local repo to a remote GitHub repository**

git remote add origin https://github.com/lukebryson/intune-device-compliance-reporter.git

**# Push your local 'main' branch to GitHub and set it as the default upstream branch**

git push -u origin main

A screenshot of a computer

AI-generated content may be incorrect.While this approach gave me a deeper understanding of the Git workflow, later on in this project I will leverage Visual Studio Code’s built-in Source Control features as it will allow for quicker visual management of changes, commits, branches, and synchronisation with GitHub, all within a single interface.

Finally, I tested it had worked by navigating back to my GitHub repo page and confirming that I was able to see the folders and my README file.

A screenshot of a computer

AI-generated content may be incorrect.

## 2. Requirement Gathering

### 2.1. Business Problem

Currently, the process of extracting device compliance data from Microsoft Intune is **manual, repetitive, and inefficient**, relying on IT staff to log in, navigate the portal, and export reports manually. This slows down triage efforts, limits visibility of non-compliant devices, and introduces risk through inconsistent reporting intervals.

#### Current Process (Manual Reporting via Intune)

The current process for checking device compliance is entirely manual. It typically involves the following steps:

1. Log into the **Microsoft Intune Admin Center** at <endpoint.microsoft.com>.
2. Navigate to **Devices > All devices**.

A screenshot of a computer

AI-generated content may be incorrect.

1. Select Windows or iOS from the sidebar on the left.

A black screen with white text

AI-generated content may be incorrect.

1. Select Columns to be shown, typically:

A screenshot of a computer

AI-generated content may be incorrect.

* Device Name (primary)
* Compliance (default)
* Free storage
* IMEI
* Last check-in
* Model
* OS Version
* Phone number
* Primary user display name
* Primary user email address
* Serial Number
* Total Storage

1. Manually export the data to CSV or Excel.

A screenshot of a computer

AI-generated content may be incorrect.

1. Open the downloaded zip file and open the CSV file contained within.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Manually process the data by:

* Removing the Device ID column- this is not necessary
* Converting all ‘Last check-in’ data in column C (under cell C1) to **Short Date** format.
* Creating a new header in cell K1 titled **Storage Remaining**.
* Storage Remaining should use the formula =**J2/I2**, be formatted as a percentage and applied to all cells in Column K (under K1) to show the storage remaining % for each device; devices with less than 10% storage remaining should be highlighted.
* Apply filters to all headers (A1- K1).
* Remove any devices that have not checked in for 3 months +.

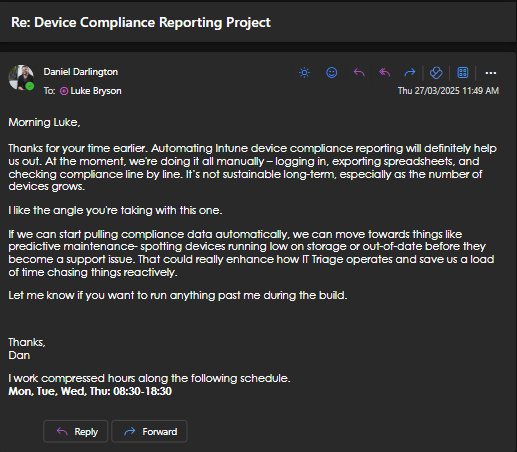
This process is time-consuming and does not support historical trend analysis or automated alerts. Due to its time-consuming nature, it is also only run on a monthly basis- I would like to change this to run weekly.

### 2.2. Objective

To design and build a Python-based tool that automates the retrieval, storage, and reporting of compliance data for Windows and iOS devices managed by Microsoft Intune, providing consistent and timely insights.

### 2.3. Stakeholder Engagement

To shape the requirements and ensure the project delivers value, I consulted with our Head of Digital & Transformation and IT Support Manager regarding reporting needs and process gaps and discussed report structure and compliance categories.



*Figure A.1 – Email from IT Support Manager confirming manual process and value of predictive maintenance.*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure A.2 – Email from Head of Digital Transformation with initial reporting requirements and strategic alignment.*

The project scope and plan were reviewed and approved by my Digital Learning Consultant (DLC) at QA.

### 2.4. Functional Requirements

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| Retrieve Compliance Data | Use Microsoft Graph API to fetch device compliance data (platform, status, last check-in) |
| Store Data | Insert structured records into a SQL database for long-term reference |
| Generate Reports | Provide summaries such as number of compliant vs non-compliant devices, platform breakdown |
| Automate | (Optional) Schedule periodic execution to keep data up to date |
| Alerting | (Stretch goal) Email notifications if non-compliance thresholds are exceeded |

### 2.5. Technical Requirements

|  |  |
| --- | --- |
| **Area** | **Technology / Decision** |
| Language | Python 3 |
| API Access | Microsoft Graph API with OAuth 2.0 |
| Data Storage | SQLite (lightweight, file-based SQL database) |
| Data Processing | pandas for transformation and aggregation |
| Visual Reporting | matplotlib or seaborn for charts (if time allows) |
| Security | Use .env file to store sensitive API credentials securely |

### 2.6. Constraints and Assumptions

* API access is assumed to be pre-authorised for device compliance endpoints.
* The script will run in a local or controlled internal environment with access to the Graph API.
* Delivery time is capped at ~30 hours across 4 weeks, with 7.5 hours per week.

With a clear understanding of the current challenges, requirements, and limitations, I was ready to begin the Analysis and Design planning phase.

## 3. Analysis and Initial Design

### 3.1. Introduction

Before starting development, I analysed the functional and non-functional requirements gathered from stakeholders, along with the technical constraints and project scope. This helped me structure the solution around high-priority features, identify potential blockers early, and align the development effort to real business needs.

### 3.2 MoSCoW

To prioritise the project requirements, I applied the MoSCoW method, a well-known Agile technique used to categorise features based on their importance to project success:

* Must Have – Essential features required for the solution to function
* Should Have – Important features that add value but aren't critical
* Could Have – Nice-to-have additions if time and resources permit
* Won’t Have – Explicitly excluded from this phase or project scope

This helped me focus on delivering a Minimum Viable Product (MVP) while remaining realistic about time constraints and project scope.

|  |  |
| --- | --- |
| Priority | Requirement |
| Must Have | Fetch device compliance data from Microsoft Intune using the Graph API |
| Must Have | Store the data in a SQL database for later reference and reporting |
| Must Have | Generate summary reports showing compliant vs non-compliant devices |
| Should Have | Filter reports by platform (Windows/iOS) |
| Should Have | Support trend analysis by storing historical data |
| Could Have | Automated email notifications for non-compliant devices |
| Won’t Have | User interface or web dashboard (out of scope for this phase) |

### 3.3. Risks and Constraints

During analysis, I also identified potential risks and technical constraints that could affect development or project delivery. These are documented below with mitigation plans.

|  |  |  |
| --- | --- | --- |
| Risk/Constraint | Impact | Mitigation |
| API Authentication | OAuth2 token access may require approval or permissions | Request delegated access and test in a secure environment |
| Rate Limiting | Frequent API calls could be throttled | Batch requests sensibly and cache where possible |
| Data Sensitivity | Compliance data may contain identifiers | Avoid storing PII; use secure storage and logging |
| Time Constraints | Project must be completed in 4 weeks (~30 hours) | Time-boxed features and focused Minimum Viable Product delivery |
| No UI Planned | Project is CLI and reporting-based only | Stakeholders aware this is a backend/reporting tool |

### 3.4. Identify System Inputs, Processes, and Outputs

To design the application effectively, I analysed the expected data inputs, the internal processes the system must perform, and the desired outputs for end users. This Input-Process-Output (IPO) model helps ensure the solution is well structured, maintainable, and aligned with user and business needs.

#### Inputs

* Microsoft Intune device compliance data retrieved via the Microsoft Graph API
* Authentication credentials (OAuth2 access token)
* Configuration parameters (e.g., platform filters, date range, environment variables)
* Historical compliance records (for trend analysis, if available)

#### Processes

* Authenticate with the Microsoft Graph API using secure credentials/ client secret
* Fetch device compliance data and parse the response
* Transform and clean the data using pandas (e.g., convert timestamps, standardise values)
* Insert the structured data into a SQL database (SQLite)
* Run summary queries to calculate compliance rates, trends, and other metrics
* Optionally trigger automated notifications for non-compliance thresholds

#### Outputs

* Tabular compliance summary reports (e.g., device ID, status, last check-in date)
* Aggregate statistics (e.g., number of non-compliant devices per OS)
* Visual charts (e.g., bar charts showing compliance over time)
* (Optional) Email alerts or log messages for internal IT reporting  
  Persistent record storage for historical analysis and audit

### 3.5. High-Level Architecture

#### System Architecture Diagram

The following diagram shows a high-level overview of the components that make up the solution and how they interact. It includes the Python scripts for data retrieval and processing, Microsoft Graph API as the external data source, the SQLite database for local storage, and the reporting layer used to generate outputs for the IT team. This architecture supports a modular, testable, and scalable design that can evolve over time as our needs grow.

A blue cylinder with white text

AI-generated content may be incorrect.

#### Data Flow Diagram (DFD)

A group of blue rectangular objects with text

AI-generated content may be incorrect.This DFD shows how data moves through the system from external source (Microsoft Graph API) through internal processing, storage, and reporting workflows initiated by the relevant member of IT. The diagram reflects a clean, modular structure to support maintainability and testability.

### 3.6. Database Design

This section outlines the design of the database that will store compliance data retrieved from Microsoft Intune. A clear and logical schema is essential for ensuring that data is stored efficiently, is easy to query, and supports meaningful reporting and analysis. This database design also contributes to the maintainability and scalability of the overall system, aligning with KSB S1 – Create logical and maintainable code.

#### Table: device\_compliance

A single table named device\_compliance has been designed to hold the retrieved data from the Graph API. This table includes the most critical fields required for reporting on device compliance and monitoring trends over time.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| serial\_number | TEXT (PK) | Unique identifier for each device |
| device\_name | TEXT | Hostname or display name of the device |
| user\_principal\_name | TEXT | Email or username of the device owner |
| os\_type | TEXT | Operating system type (e.g. Windows, iOS) |
| compliance\_state | TEXT | Compliance status (Compliant, Non-compliant, Unknown) |
| last\_check\_in | TEXT | Date/time the device last checked in |
| storage\_space\_gb | REAL | Available device storage space (if available) |
| timestamp\_recorded | TEXT or DATE | Timestamp of when this record was written to the DB |

The serial number is used as the **primary key** to ensure data integrity and prevent duplicate entries.

#### Entity Relationship Diagram (ERD)

I decided that an ERD was not necessary as my design only involves one table and the table structure I have written (with clear fields, types, and descriptions) already meets the expectations for demonstrating structured database design. My decision also aligns with the “YAGNI” principle (You Ain’t Gonna Need It)- a software development practice that encourages developers to only implement features when they are absolutely necessary.

### 3.7 Chosen Technologies

This section explains the technologies selected for this project and provides a rationale for each, alongside an evaluation of possible alternatives. The chosen tools enable me to develop a solution that is maintainable, testable, and aligned with both the project's requirements and my current technical skillset.

#### Chosen Tools and Rationale

|  |  |  |  |
| --- | --- | --- | --- |
| **Technology** | **Reasoning** | **Alternatives** | **Trade-offs** |
| **Python** | Easy to learn, excellent for scripting and APIs, strong library ecosystem (e.g. requests, pandas) | Java, C# | Slower than compiled languages; whitespace-sensitive |
| **SQLite** | Lightweight, no server required, ideal for local data storage; integrates easily with Python | PostgreSQL, MySQL | Not ideal for large-scale or multi-user systems |
| **Microsoft Graph API** | Official method to retrieve Intune data; secure and well-documented | Manual Intune exports | Requires authentication setup and API learning curve |
| **Pandas / Matplotlib** | Efficient tools for data manipulation and visualisation in Python | Excel, Power BI, seaborn | May require additional learning for complex plotting |
| *(Optional)* **Task Scheduler / Cron** | Enables scheduled automation of scripts | Manual script execution | Platform dependent; needs basic config |

## 4. Development

### 4.1. Project Setup and Environment

To begin development, I ensured my local environment was fully configured for Python development, version control, and structured project management. This reflects best practices in professional software engineering and supports maintainable code.

* **IDE Used:** I used Visual Studio Code for its Python support, Git integration, and terminal access.
* **Folder Structure:** As defined in Section 1, I created a modular folder structure (Scripts, Docs, Reports, etc.) to separate concerns and support scalability.
* **Version Control:** The project was tracked using Git from the beginning. All changes were committed with meaningful messages, and the repository was hosted privately on GitHub.
* **Python Environment:** I used the system-wide Python 3.x installation, with built-in modules (sqlite3, json, os) and additional packages (requests, pandas, matplotlib) installed via pip.

📸 *Include screenshots of VS Code open with your repo, Git commit history, or terminal running the script.*

This setup phase allowed me to begin development with confidence, knowing that the core tooling was in place and aligned with the design decisions laid out in earlier sections.

### 4.2. API Connection to Microsoft Graph

The first development task was establishing a secure connection to the **Microsoft Graph API**, which is the recommended method for retrieving compliance and device management data from Microsoft Intune.

This step reflects structured development and debugging processes and required analysis of authentication mechanisms and endpoint selection.

To configure API access, I:

* Logged into Azure, navigated to **App registrations** and selected **New registration** before entering a name and selecting **Register**.  
  A screenshot of a computer

  AI-generated content may be incorrect. **A screenshot of a computer

  AI-generated content may be incorrect.**
* Granted it the required Graph API permissions (**DeviceManagementManagedDevices.Read.All**)  
    
  A screenshot of a computer

  AI-generated content may be incorrect.
* This is the description of **DeviceManagementManagedDevices.Read.All**  
  A screenshot of a computer

  AI-generated content may be incorrect.

#### Authentication Approach

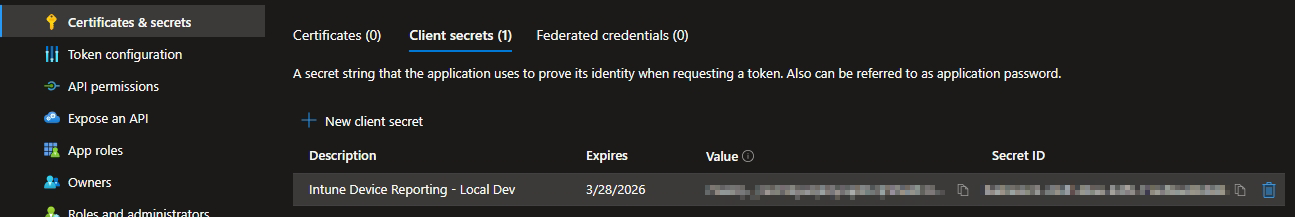
Microsoft Graph uses OAuth 2.0 for secure access. I selected **Application permissions** as this will allow it to run as a background service without a signed-in user.

A screenshot of a computer

AI-generated content may be incorrect.

#### Client Secret

A client secret was generated during the Microsoft Azure app registration process to authenticate against the Microsoft Graph API. A **client secret** is essentially a password used by your application to authenticate against Microsoft’s identity platform. It's used in combination with your **Client ID** and **Tenant ID** when requesting access tokens.



The secret was set to expire in 12 months, following best practices for development environments. This expiry window balances security with convenience and ensures the application’s credentials are rotated regularly. The expiry date has been documented to prevent unexpected failures- When a client secret **expires**, your application will **fail to authenticate**, meaning:

* You will **not be able to retrieve tokens** from Microsoft Graph
* Any scripts, integrations, or automations depending on it will **stop working**
* You will need to **create a new secret** and update your code

### 4.3 Storing Credentials Securely with .env and .gitignore

To ensure sensitive information such as API credentials is stored securely and not exposed in source control, I used a .env file in combination with a .gitignore file. This approach follows industry best practices and contributes toward professional software engineering standards.

#### Step 1: Install python-dotenv

#### Step 2: Creating a .env File

I created a new file named .env in the root of my local project folder. This file contains key-value pairs used to store my confidential Microsoft Graph API credentials securely:

CLIENT\_ID=my-client-id-here

TENANT\_ID=my-tenant-id-here

CLIENT\_SECRET\_ID=my-client-secret-id-here

CLIENT\_SECRET\_VALUE=my-clien-secret-value-here

Each variable is declared using the format KEY=VALUE, and these will be programmatically loaded in Python using the dotenv library during runtime.

#### Step 3: Adding .env to my .gitignore file

To ensure the .env file is never uploaded to GitHub, I created a **.gitignore** file in the root directory and added the following line:

# Environment variables

.env

This tells Git to **exclude the .env file from version control**, protecting sensitive credentials from being exposed in the public or private repository. When using Visual Studio Code, this also ensures GitHub desktop and Source Control panes do not accidentally stage this file for commit.

#### Step 4: Load the Environment Variables in Your Python Script

I created a file inside the **/Scripts** folder called **auth.py**. The Python script was designed to handle **authentication** by loading the .env file and retrieving the access token.

# auth.py

import os

from dotenv import load\_dotenv

import requests

# Load variables from .env file

load\_dotenv()

# Get credentials from environment variables

TENANT\_ID = os.getenv("TENANT\_ID")

CLIENT\_ID = os.getenv("CLIENT\_ID")

CLIENT\_SECRET\_VALUE = os.getenv("CLIENT\_SECRET")

# Construct the token request

url = f"https://login.microsoftonline.com/{TENANT\_ID}/oauth2/v2.0/token"

headers = {"Content-Type": "application/x-www-form-urlencoded"}

data = {

    "client\_id": CLIENT\_ID,

    "scope": "https://graph.microsoft.com/.default",

    "client\_secret": CLIENT\_SECRET,

    "grant\_type": "client\_credentials"

}

# Send the request and print the access token

response = requests.post(url, headers=headers, data=data)

print(response.json())  # Access token output

pip install requests

A screenshot of a computer

AI-generated content may be incorrect.

.venv created

Added the following to the .gitignore file

# Ignore virtual environment folder

.venv/

Activated

### 4.4. Fetching and Parsing Device Data

### 4.5. Writing Data to SQLite

### 4.6. Generating Compliance Reports

### 4.7. Optional Enhancements (e.g. Notifications or Scheduling)