# InTune Device Compliance Reporter

Contents

[InTune Device Compliance Reporter 1](#_Toc193976241)

[Introduction 1](#_Toc193976242)

[Initial Setup 2](#_Toc193976243)

[Local Folder 2](#_Toc193976244)

[README.md File Creation 3](#_Toc193976245)

[GitHub Repository 3](#_Toc193976246)

[Linking to Local Project 5](#_Toc193976247)

[Requirement Gathering 6](#_Toc193976248)

[1. Business Problem 6](#_Toc193976249)

[2. Objective 8](#_Toc193976250)

[3. Stakeholder Engagement 8](#_Toc193976251)

[4. Functional Requirements 9](#_Toc193976252)

[5. Technical Requirements 9](#_Toc193976253)

[6. Constraints and Assumptions 10](#_Toc193976254)

[Analysis and Initial Design 10](#_Toc193976255)

[MoSCoW 10](#_Toc193976256)

[Risks and Constraints 11](#_Toc193976257)

[Identify System Inputs, Processes, and Outputs 11](#_Toc193976258)

[Design High-Level Architecture 12](#_Toc193976259)

[System Architecture Diagram 12](#_Toc193976260)

[Data Flow Diagram 12](#_Toc193976261)

[Class Diagram 12](#_Toc193976262)

[ERD (Entity Relationship Diagram) 12](#_Toc193976263)

[Choose Technologies and Justify 12](#_Toc193976264)

## 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 and support compliance with internal policies.

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.

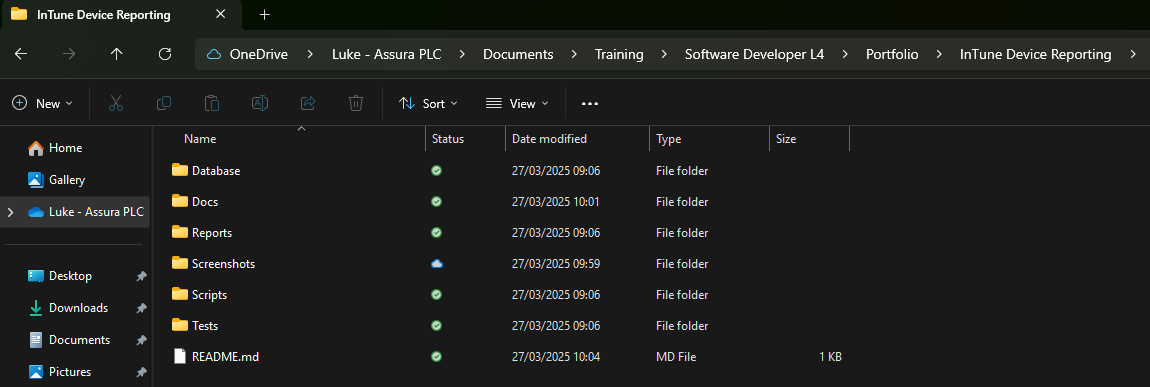
The project directly benefits my organisation while also allowing me to evidence key apprenticeship learning outcomes:

* **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

## Initial Setup

### Local Folder

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 was taken and stored in the /Screenshots folder for evidence, alongside a short folder-by-folder description included below.



#### 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.

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 allow me to link it to my local folders.  
  
A screenshot of a computer

AI-generated content may be incorrect.

### Linking to Local Project

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.

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- the folders must have contents which is why I gave each a README file**

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

Finally, I confirmed it was working by navigating back to my GitHub repo page and confirmed I was able to see my folders (Docs, Scripts, etc.) and my README.md.

A screenshot of a computer

AI-generated content may be incorrect.

## Requirement Gathering

### 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 +.

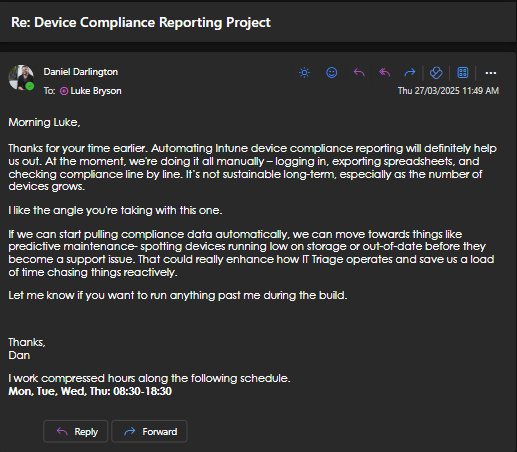
This process is time-consuming and does not support historical trend analysis or automated alerts. Due to its time-consuming nature, it is also only run on a monthly basis- I would like to change this to run weekly.

### 2. 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.

### 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.

### 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 |

### 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 |

### 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.

## Analysis and Initial Design

### 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.

### 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) |

### 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 |

### 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

### Design 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

Show how data moves between components.

Class Diagram

(Optional, if you apply OOP)

Useful if you want to include Java or Python classes.

### ERD (Entity Relationship Diagram)

Show how your data tables relate.

These will become key figures in your project report and evidence design competency (K11, S8, S9).

### Choose Technologies and Justify

Briefly explain and justify your technology stack:

* Python for scripting/API
* SQL for storage/querying
* Pandas/matplotlib for analysis/reporting
* Optional: Email reporting with smtplib, scheduling with cron or Task Scheduler

Use a comparison matrix or rationale list:

Use a comparison matrix or rationale list:

|  |  |  |  |
| --- | --- | --- | --- |
| **Technology** | **Why Chosen** | **Alternatives** | **Trade-offs** |
| Python | Ease of API integration | Java, C# | Slower execution than compiled languages |
| PostgreSQL | Powerful SQL with cloud support | MySQL, SQLite | More complex setup |

*This contributes to KSBs: S1, S4, K11 and will show critical evaluation for Distinction.*