University of Glasgow | School of Computing Science

Level 3 Project Case Study Dissertation

Cisco Meraki Data Analysis Web App

Group CS11

Ben Lynch
Fraser Dale
Jake Haakanson
Johnathan Dominick Sciallo
Ruofan Guo

April 12, 2021

Abstract

This document is a case study of the software development project Cisco Meraki Dashboard - a web application created to demonstrate the applications of the Cisco Meraki APIs to the client Cisco as part of the Team Project 3 coursework.

The team describe the development cycles of the project through specific milestones and challenges they encountered, software practices they used and the final product itself.

The conclusion demonstrates all key learning achievements. The team developed a functional web app meeting the requirements of the client and implementing the various technologies provided for their use, whilst recording any delays due to team behaviours and delays outwith their control.

Education Use Consent

We hereby give our permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format.

1 Introduction

The following document describes the development process of group CS11's level three team project. Cisco representatives asked the team to devise 2 phases for the project, with their aim being to observe how they used the provided APIs and if they could present the information in an interesting way. The main goals of the project were separated into two phases described in Section 2.2, where the first phase was used to establish a platform that incorporated the client's provided hardware (detailed in Section 4.3.3) and software functionality, and the second phase built upon this in order to create the unique functionality of a COVID tracker detailed below within the system.

The team decided that for the first phase of the project they would build upon the Cisco Meraki Dashboard[1] in order to provide network insights to non-technical users such as management, the team agreed that a web app was an appropriate format for implementing functionality and displaying the necessary information.

For the second phase, the team looked into the capabilities of the hardware provided for the project in order to implement unique functionality for the web app developed during the first phase. The team used the COVID-19 global pandemic and the hardware functionality as motivation to develop a "COVID Tracker", in which the system works to ensure that users of the network are adhering to social distancing guidelines using alerts that are displayed to system administrators.

The following also outlines the background of the case study in section 2, details of interactions with the client in section 3, information about the technologies employed by the team including hardware and software in section 4, documentation about how the team coped with implementing automated tests for the system in section 5. Finally, section 6 concludes the document, detailing what the team has learned throughout development of the project about the software engineering process.

2 Case Study Background

2.1 Client

Cisco Systems is a California-based multinational technology company well known for its contributions to networking technology [9]. Meraki, a cloud-managed IT company, was acquired by Cisco Systems in 2012 as part of their cloud networking group. Cisco Meraki's cloud architecture allows users and businesses to create and deploy secure and scalable networks that can be managed from anywhere via the internet [1]. The

[1]The Cisco Meraki Dashboard, described in Section 2, is an online portal allowing system administrators to manage their networks via the cloud.

group from Cisco that the team worked with consisted of a marketing team, with a mix of software engineers and marketing personnel.

2.2 Project

Cisco Meraki allows its users to manage and interact with their established networks via an online dashboard. This is an effective approach to network management for technologically-inclined users, however, the Meraki dashboard does not provide much accessibility to users who are not as experienced with the technology. This inspired the team to initially develop a more appropriate method utilizing a simple and easy to use interface which displays basic information for non-technical users to gain insights into their network.

While there were no solid initial requirements from the client as to what the finished product should be, the client specified an approach to development which the team should follow in order to create a product that the team would have free roam over the chosen implementation and functionality of the final product. The client also specified that the prominent features of the product should be the features the team felt would be useful or interesting to have inside an application which had access to the data provided by their APIs, rather than a strict set of features the client needed to have. The stages of development the client advised us to follow were:

- The first phase of the project was to establish a web app as an analytical dashboard in order to provide analysis of network data to non-technical users.
- The second phase of the project was to use the first phase in order to develop the application further and to provide more useful insights to users in different contexts.

The team worked closely with the client in order to realise and implement the functionality of the system as the project developed. Although Cisco stated from the beginning of project development that they had no desire in being delivered a product that would be used in practice, the interest was in creating a new way to use the data supplied by their API in an interesting way.

2.3 Finished Software Product

2.3.1 First Phase

As described in Section 2.2, during the first phase of the project the team decided to create a web app as an analytical dashboard to provide detailed network information to non-technical users, the technologies used to implement this project are discussed in Section 4. In order to interface with the client's hardware, the team used the provided Cisco Meraki Dashboard API in order to obtain relevant information about the network and provide it to the user in a non-technical fashion, conveyed through the use of a simple and easy to navigate user interface which displayed the relevant data in an easy to understand way. The first phase of the project ended with the development of a general "overview" page of the web app, providing users with an overview of the details of their organisations and the networks belonging to them. With this, the team also developed other functionality, such as methods to retrieve details of devices connected to individual networks, as well as login and registration to facilitate personalised API key usage.

Near the end of this phase of development, members of the team eventually gained access to Client hardware which needed to be set up and integrated with the Client API. Gaining access to this hardware, which is detailed in Section 4.3.3, broadened the scope of what the team were able to create as there was now new information being retrieved that could be utilized. Cisco also asked the team to decide the time-frame for the two phases they had proposed, which the team suggested should be Christmas for phase 1, and the final presentation day for phase 2.

2.3.2 Second Phase

After establishing the first phase of the project the team decided that, due to the COVID-19 global pandemic, the second phase of the project should attempt to ensure that users of the network are adhering to social distancing restrictions. The team's decision was also influenced by the functionality of the client's hardware, as the Cisco Meraki access points provide functionality for geolocation of devices on the network via an external listener, and the Cisco Meraki cameras provide functionality for real-time human-detection via an MQTT broker[2].

The final implementation of the second phase was a "COVID Tracker" alerts system, in which network administrators are notified of social distancing breaches. These breaches were separated into two different categories, breaches captured by access points using device locations, and breaches captured by cameras. The access points mentioned are hardware which was included in the Client hardware distributed to the team to be used over the course of the project, these access points along with camera equipment were the main information collecting devices used. These pieces of hardware were used separately to try and detect these breaches, in the case of the access points, by monitoring device locations, and in the case of the cameras, by monitoring physical activity through motion detection. These breaches once detected would create an alert which would display, in the case of the breach being detected by an access point, a time, location and the offending devices, and in the case of a camera detecting a breach, the time and a link to the image captured by the camera.

[2]MQTT or Message Queuing Telemetry Transport is a publish-subscribe protocol, where a device, in this case the Cisco Meraki camera, publishes information to a broker which is then received by a subscriber device of the broker.

3 Client Interactions

3.1 Requirements Gathering

The team were tasked with gathering client requirements at each monthly customer day meeting[3]. The team initially felt that this was particularly difficult to navigate due to the lack of project goals, as the client preferred that the team decided on the direction of the project. Cisco were very helpful and suggestive once the team had ideas and features to implement, but were very clear that initial direction and requirements should be constructed by the team first, and then commented on and evaluated by them second.

Before each customer day, the team referred to the set of completed tasks over the course of the previous sprint in order to determine new tasks for the next sprint, these new tasks were then discussed with the client during the meeting to allow them to provide any input into how they would like to see certain functionality implemented. Task priorities were also discussed during these meetings, and this process is described in section 3.2.

In the first customer day presentation the team constructed user stories to identify and justify requirements for phase 1 features, receiving feedback on the accuracy of the stories and the usefulness of the requirements identified at the end of the presentation.

As well as the team deciding on the new requirements of the project, the client also described functionality to the team that they would like to see implemented. This functionality largely related to use of the client's equipment, such as suggestions on how the team could implement certain hardware functionality into the existing web app.

The team used planning poker at multiple stages of development, an example scenario would be whether to prioritise integrating the MV sense API during phase 2 or the algorithm to calculate distance between two coordinates for devices obtained from the router. In this example the team would prioritise the API implementation, as it is a larger task and the other tasks relevance is dependant upon the API implementation.

3.2 Prioritising Requirements

Priorities were first determined by the team before the customer day meetings during the team's requirements analysis as described in section 3.1. The team discussed the priorities of tasks as they decided on the new requirements of the project, priorities were determined based on the length of time the task would take and the implications of a failed task and how it would affect the project, some tasks also affected the

[3]Customer days indicated the end of a sprint, and allowed the team to meet with the customer in order to demonstrate the progress made over the course of the sprint. It also allowed the team to discuss new requirements and their priorities with the customer before the start of the next sprint.

priorities of other tasks as the team felt that it was important to implement certain functionality sooner.

During the customer day meeting the team would discuss the earlier determined priorities with the client, this allowed them to provide any insight into how they felt the development of a feature would impact the project, and allowed the team to adjust any priorities based on these discussions. The discussions also allowed the team to review any of the priorities they had determined and adjust as necessary.

An activity that helped determine some task priorities was "Planning Poker" in which different cards are played by a team member representing how long they feel the task will take. This activity stimulated discussion amongst the team members and allowed them to agree upon an appropriate priority for a given task.

Another activity that the team utilised in the very early stages of the project was MoSCoW prioritisation. With this method, each team member categorised the larger tasks related to the overall goals of the project into four categories labelled "Must have", "Should have", "Could have", and "Would have" to indicate how they felt the task would contribute to the final product. The activity and accompanying discussion enabled the team to prioritise certain aspects of the project early on, aiding in future discussions about later tasks.

4 Technologies

Throughout the project, the team was introduced to several new technologies necessary to implement the functionality for the client's requirements. Some technologies were chosen by the team in order to implement the required functionality, and some technologies were provided by the client. The technologies used throughout the project include Meraki's Dashboard REST and MQTT APIs, a team written middleman REST API, GitLab, Python3 and the Django Python library, Amazon AWS, and the client's hardware which included Cisco Meraki MR30H access points and Cisco Meraki MV12 Cameras.

Django, described in section 4.2, was chosen due to its quick and easy-to-deploy nature, allowing the team to rapidly develop the base web app on which to build the first phase of the project. The team also made this decision based on the fact that all team members had prior experience with using Django. The team also noted that due to the open source nature of Django lead to an available and powerful resource with a lot of the complicated work done as well as security concerns.

The team also benefited from the fact that they all had a good amount of experience with the Python programming language, detailed in section 4.1, between them, this coupled with the Django framework would allow the team to rapidly develop minor parts of the project in order to spend more time developing the more vital functionality.

4.1 Python3

Python is a high-level, interpreted, general purpose programming language [6]. The team made the decision to use the Python programming language due to the fact that all team members had an amount of previous experience programming in Python. The decision was also made due to the fact that the Cisco Meraki Dashboard API had a published Python library allowing the team to interface with the Cisco Meraki hardware within Python code, and also due to the fact that the team opted to use the Django web development framework, described in section 4.2, in order to develop the base web app from which to develop the rest of the project functionality.

4.2 Django 3.1.2

Django[2] is a high-level Python-based web development framework designed for rapid development of web applications[3]. The team was also motivated to choose the Django framework as the Cisco Meraki Dashboard API, detailed in section 4.3.1, has a published Python library for ease of interaction with Cisco Meraki hardware. The team chose to use Django due to their familiarity with the framework, however, in hindsight the strict rule sets and limitations of Django lead to difficulties in front and back end communication and final deployment, in future the team would use either the Python Flask framework or a different API framework with more freedom.

4.3 Client Technologies

4.3.1 Client APIs

Throughout the first phase of the project, the team made use of the Meraki Dashboard API, a REST API that allows users to manage and monitor Meraki networks[12]. The team made use of the Meraki Dashboard API Python library in order to interface with the API within the existing Python code used to develop the web app, this API allowed the team to obtain information about Meraki networks, such as organisations, networks belonging to organisations, and individual devices connected to these networks. The goals of the first phase of the project were dependant on this API, as the functionality required analysis of the data collected by these network components, and therefore the team were required to quickly adapt to interfacing with the new technology.

The second phase of the project required the team to interface with the more in-depth functionality of the client's hardware and the team were introduced to two new APIs, the Cisco Meraki Scanning API that enabled integration of the access point's geolocation functionality[16], and the Cisco Meraki MV (or Meraki Vision) Sense API, an MQTT API that allowed integration of live camera motion detection functionality[14]. Again, the functionality throughout this phase of the project depended heavily on these technologies and it was required that the team quickly became proficient in interfacing with the APIs.

The team found that it was necessary to set up external functionality in the form of listening and broker servers in order to interact with these more advanced APIs. The access points would continuously push updates about device locations to a listening server which would then calculate distances and provide appropriate updates to the web app when necessary, and a listening server was set up using Google Cloud Services, a cloud computing infrastructure that allows users to easily manage server functionality via an online portal[10].

The MV Sense API required an MQTT broker server for the cameras to publish updates to in order for subscribers, in this case the web app, to receive these updates, MQTT is a lightweight publish-subscribe messaging protocol used heavily for the "Internet of Things"[18]. The team decided that due to the prototype nature of the project in its current stage, it was not appropriate to set up an MQTT broker, and instead opted to use the Mosquitto Test Server, a public MQTT broker hosted by Eclipse Mosquitto[17].

4.3.2 Paired Programming

The team practised paired programming when interacting with new technologies as a method to better understand them as a group, the team also used paired programming if one team member required further guidance. The team reflected that the efficiency of development and the quality of code was greatly improved by paired programming.

4.3.3 Client Hardware

The team were provided with three Cisco Meraki MR30H cloud managed access points and two Cisco Meraki MV12 mini dome cameras. However, these items of hardware were not supplied for several months after the start of development, this gave the team plenty of time to develop the basic features of the application that did not require the specialised hardware. This also made it so that the team had plenty of time to understand the APIs, but without significant practical experience in actually using the hardware, halting the development of more in depth features through either not having access to necessary hardware at the time or not having relevant experience in using them when they became available.

The MR30H access point is a dual-band 802.11ac wireless access point managed via the Cisco Meraki Dashboard, mentioned in section 1, providing good performance with high security[13]. The Cisco Meraki MR30H access points allowed the team to implement much of the functionality seen in the final product, such as retrieving all devices connected to the network and measuring distances between devices through geolocation. Geolocation of devices connected to the network is performed through triangulation of the device's position using at least two access points.

The MV12 mini dome camera is a ”smart camera” capable of providing advanced analytics, object detection and identification, and motion tracking[15]. The team made use of the MV12 cameras throughout the second phase of the project, in which they developed an alerts system. The MV12 provides functionality for motion detection and object identification, therefore the team were required to interface with this functionality in order to incorporate it into the project. Interfacing with the cameras was performed using an MQTT broker alongside the Cisco Meraki MV Sense API as described in section 4.3.1, and this functionality was then incorporated into the alerts system through retrieval of camera images via the MV Sense REST API.

4.4 Version Control

4.4.1 Feature Branching Strategies

Feature branching was a strategy employed by the team in order to allow us to concurrently develop new features, new feature branches would be created off of the development branches so multiple team members could create new features simultaneously and integrate these into a development branch that would not affect master. [20]Feature branches would be created based off of issues added to the git lab page, and could be worked on by team members independently of the main development branch. The use of feature branches reduced the conflicts that would have emerged with multiple team members working off of the same branch to add features, meaning that when it came time to integrate the code into a branch like development there would be a lower risk of conflicts. This was useful for features that took little time to develop, perhaps in a day or a couple of hours, this way those features could be created and merged without having problems with delayed integration.

4.4.2 Issue Tracking and Task Management

Issue tracking for the project was primarily handled via the use of the GitLab integrated issue tracker, because this allowed the team to create branches for each issue and work on them individually decided before merging the results back into the development branch. The team decided on issues and brought them to the customer during the customer days where they agreed priorities based on what the customer wanted and the team thought was feasible. After each retrospective, the team chose which tasks each member would handle based on the areas that they were most comfortable with or they wanted to learn more about. The priorities were reflected in the SCM, Gitlab.

Tasks could be assigned to individuals and tracked throughout their life cycle as well as submitted for code reviews before merging into the development and then master branch.

4.4.3 Code Reviews

Whilst initially the team did not take advantage of code reviews, as they got more comfortable with Gitlab they made use of the code review system to improve each others developments. The team also made use of code reviews when a member was unable to progress with their current development and a review would direct the other team member in the correct direction. The team also used code reviews on large new features in the later half of the project.

4.4.4 Documenting Project Progress and Meetings Using a Collaborative Wiki

Project progress was documented throughout the course of the project on the GitLab collaborative wiki. Pages were added to the wiki for each meeting with the customer which occurred on a weekly basis during the first phase of the project, and on a bi-weekly basis during the second phase.

Each customer day indicated the end of a sprint, and a page was added to the wiki for each customer day detailing the feedback from both the customer and the marker that attended the meeting. The wiki page also detailed the team's retrospective, the team opted to use the liked, learned, lacked, longed for approach, as well as a comprehensive list of all tasks completed during the sprint including issues, features, bugs, and any other notes the team felt were worth mentioning. Using regular retrospectives helped keep the team on tack and focused while forcing us to continually reevaluate our strategies to ensure they were constantly on track with making meaningful progress on the application. This evaluation came in the form of each team members asking themselves the questions on what the liked, lacked, learned, longed for, helping the team to think about the progress they were making and how it could be improved.

5 Testing

The team made use of Gitlab CI/CD, described in section 5.1, in order to carry out automated unit testing and static analysis of all code. The team opted to use Pylint for static analysis of all Python code described in section 5.3. Unit tests are a large part of Django development and the team made use of the existing functionality in order to implement tests as described in section 5.2.

5.1 Gitlab-CI Runner

Gitlab CI/CD is an integrated Gitlab tool that enforces continuous methodologies, an approach to software development characterised by the continuous development, testing, and deployment of code[7], to ensure that all code is continuously tested upon being pushed to the repository before it is possible to merge into any main branches[8].

The Gitlab runner handled the process of setting up a virtual environment, downloading any necessary Python libraries in order to run the code, and executing both the Django unit tests and the Pylint static analysis. Gitlab would indicate the success or failure of each stage of the pipeline allowing the team to make the necessary changes in order to ensure that the code was fully functional and bug-free before they could merge any changes into the development branch. This also allowed the team to merge changes efficiently, as many error causing conflicts would be flagged to the team.

The team did not encounter any issues when setting up the CI/CD runner, and did not experience many issues while using it over the course of the project. The most common, if not only, issue the team encountered was the slow execution of code on the CI/CD runner, the team believe that this was due to the fact that most other teams would be using the same runner at around the same time leading to slow run times for all users.

5.2 Unit Tests

As described briefly in section 5, the team made use of the existing Django unit testing functionality. Unit testing functionality is provided with Django and allows developers to automatically detect bugs and erroneous code with a user-defined test suite[4].

It was very important to implement unit testing to cover all newly developed functions in the project, because this allows the team to quickly resolve all detected errors or problems, and is a guarantee for high-quality code in software development. The unit test part includes tests for several functions of the web app, the most basic ones: registration, login, and API. It also includes testing of some details, such as the duplicate username/email, alert and overview interface.

Unit testing also enables the team to discover any negative effects of new development on the earlier functionality of the project. For example, the improvement of the registration and login interface, and the repeated testing of the API.

In hindsight the team felt that although implementing unit tests was a benefit the the overall project, and having a fully developed test suite for all implemented function in the application would have aided in the development process and quality of the code base, there were some flaws with our test implementation process that had a negative impact on development. As unit tests were not implemented from the start of development there was an initial load of tests that had to be added, instead of adding tests as the corresponding features were first created, this meant that adding these features took longer than expected. This also resulted in several tests having to be remade and condensed into larger, more efficient tests after implementing many smaller tests first. A result of this was that new features continued to be added without corresponding tests being made, exemplifying the initial problem of having a backlog of tests not created. In hindsight the team feels that if they were to start again a steady flow of tests being created would be ensured, so that all features would have a test created for them as they were implemented, perhaps by having a dedicated member of the team for unit testing or having the member of the team that created the feature write a test for it alongside their development.

5.3 Static Analysis

The team made use of the Pylint Static Analyser. Pylint is a tool used for error detecting and the enforcement of standards in code through code smells[11]. The execution of Pylint static analysis was handled by the GitLab Runner as described in section 5.1, and as a result all new code pushed to the repository was analysed for errors and poorly formed code.

The team found that this aided them throughout development of the project, not only as the enforced structure of the code allowed for clearer understanding between team members, but also that it encouraged the team to write more well-formed code that adhered to the PEP-8 Style Guide, documented conventions for the development of Python code[5]. It allowed the initially different coding styles and practises of the team to be consistent throughout the development process.

The team observed that a significant amount of Django conventions do not adhere to PEP-8, and therefore violate Pylint's rules. These violations were flagged by the team as false positives and it was necessary for the team to disable the detection of certain violations of code style in order for the Django code to pass inspection.

5.4 Bug Handling

The team communicated the detection and removal of bugs using the built-in GitLab issue tracker mentioned in section 4.4.2. Most bugs detected by the team members were related to improper use of developed functionality, such as providing unexpected inputs within the web app itself.

A number of other bugs were detected relating to the libraries that the team were using while developing the project, and while some of these were resolvable there were some minor bugs detected that relate to the interaction of these libraries with the Python programming language.

Another Type of bug the team encountered were those specific to the Cisco APIs that could be difficult to resolve due to their nature. These bugs arose from rate limiting on our API key that led to some small issues at first, but however became more apparent when the size and scope of the application increased. The only way to fix this was to contact Cisco support and open a support ticket to get the issue resolved, however this halted development of the project for several weeks and the team had to work around this rate limiting in order to continue to make progress.

6 Conclusions

The project marked the team's first large scale collaborative project as well as their first use of such wide scope APIs as those provided by Cisco, and posed a significant challenge in learning and using such professional and far-reaching technologies. A significant portion of time during the first weeks of the project was used to understand the provided Postman[4] demo of the dashboard API and learning how to request only what the team required from the comprehensive details the API provided for even simple calls. The team also had to deal with delivery delays caused by the client's need to deliver to the university first and then for the team to collect the devices themselves, as well as those caused by the COVID-19 pandemic. To overcome these factors the team had to learn how to prioritise the API familiarisation while the delay was in place, as well as adapt to learning to use equipment they did not possess via documentation and in some cases assumption. This gave them practical experience with real industry delays and cross organisation mediation and communication.

A powerful driving force behind the progress the team made in development was the use of source code management systems like those present in GitLab. This was a jump from the teams collective experience of using git, but the experience of using a more powerful system with more features to better organise the development process. This experience can help the team in future large scale projects where concurrent development and project management are imperative.

The teams key learning achievements from the development process of the team project included but were not limited to; professional agile practices including retrospectives, stand-ups, and paired programming as well as code reviews and use of an SCM system. The use of these good practices heavily aided the teams ability to complete all requirements created and maintain time and order between customer meetings.

This was the first time that the team had worked in a remote environment for the full duration of a project, including all interactions with the customer. The team reflected that the existing agile skills that they already had were, reasonably, straightforward to implement in an online environment, and the likes of CI/CD was generally the same. The team reflected in their retrospectives and in general that team meetings and general team communication was difficult at and this would be less of an issue in person, however, the meetings with the customer were always well attended and presented. The skills that were learnt and developed, by the team, from a remote environment will be useful and valuable in future professional endeavours.

The challenges that the team encountered were generally outwith their control, however, the handling of such issues improved as the project progressed. The issue concerning lack of requirements from the client that lead to slow progress initially but they worked with the client continuously to overcome this. Despite working under an agile practice, concrete requirements are not essential [19], as demonstrated in this document.

[4]Postman is a collaborative platform that allows developers to create and publish APIs[21]

References

[1] J. Brejcha. 10 things you need to know about cisco meraki, 2014. https://gblogs.cisco.com/uki/10-things-you-need-to-know-about-cisco-meraki/.

[2] Carl Burch. Django, a web framework using python: Tutorial presentation. *Journal of Computing Sciences in Colleges*, 25(5):154–155, 2010. https://dl.acm.org/doi/abs/10.5555/1747137.1747166.

[3] Django Software Foundation. Django web framework. https://www.djangoproject.com/.

[4] Django Software Foundation. Testing in django. https://docs.djangoproject.com/en/3.1/topics/testing/.

[5] Python Software Foundation. Pep 8 – style guide for python code. https://www.python.org/dev/peps/pep-0008/.

[6] Python Software Foundation. Python 3.8.8 documentation. https://docs.python.org/3.8/.

[7] GitLab Inc. Ci/cd concepts. https://docs.gitlab.com/ee/ci/introduction/index.html.

[8] GitLab Inc. Gitlab ci/cd. https://docs.gitlab.com/ee/ci/.

[9] R. Lewis. Cisco systems, 2017. https://www.britannica.com/topic/Cisco-Systems-Inc.

[10] Google LLC. Google cloud overview. https://cloud.google.com/docs/overview.

[11] Logilab. Pylint user manual. http://pylint.pycqa.org/en/latest/.

[12] Cisco Meraki. Meraki dashboard api. https://developer.cisco.com/meraki/api-v1/#!introduction/meraki-dashboard-api.

[13] Cisco Meraki. Mr30h datasheet. https://meraki.cisco.com/product-collateral/mr30h-datasheet/?file.

[14] Cisco Meraki. Mv sense. https://meraki.cisco.com/lib/pdf/meraki_datasheet_mv_sense.pdf.

[15] Cisco Meraki. Mv12 cloud-managed smart cameras. https://meraki.cisco.com/product-collateral/mv12-series-datasheet/?file.

[16] Cisco Meraki. Scanning api. https://documentation.meraki.com/MR/Monitoring_and_Reporting/Scanning_API.

[17] Eclipse Mosquitto. Mosquitto test server. https://test.mosquitto.org/.

[18] OASIS. Mqtt version 5.0, 2019. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[19] Z. Racheva, M. Daneva, K. Sikkel, A. Herrmann, and R. Wieringa. Do we know enough about requirements prioritization in agile projects: Insights from a case study, 2010. 18th IEEE International Requirements Engineering Conference, Sydney, NSW, Australia, 2010, pp. 147-156, doi: 10.1109/RE.2010.27.

[20] Konrad Schneid. Branching strategies for developing new features within the context of continuous delivery. In *CSE@ SE*, pages 28–35. Citeseer, 2017.

[21] Postman Team. What is postman? `https://www.postman.com/`.