Universitat Politècnica de Catalunya

Automating the configuration of networking devices with Python

A Master's Thesis submitted to the Faculty of the Escola Técnica Superior d'Enginyeria de Telecomunicació de Barcelona

by

Alberto Simón Fernández

In partial fulfillment of the requirements for the Master's degree in Telecommunications Engineering

supervised by
Prof. Marcos Postigo Boix

Barcelona, May 22th, 2020

Abstract

The networking team in ITNow needed to automate the configuration and massive data extraction of specific networking devices. In this thesis a web portal running the Django framework alongside some Python scripts was developed to fulfill this need. The operation and structure of the web portal are described including some examples and the scripts that run beneath are explained in detail.

List of Figures

Contents

Chapter 1

Introduction

1.1 Statement of purpose

The time where networking engineers log into networking equipment to manually input commands to configure devices or log into servers to manually configure one by one a list of devices/networks is coming to an end. More and more companies are pushing forward in automation as they see that every hour put into automation translates to a lot of hours of work saved.

Automating these tasks with some well done programming logic allows the configuration of hundreds of devices in minutes, removes the possibility of miss-configurations originated from human error, allows logging of configuration changes and has the advantage that it makes the configuration technical details transparent to the user who is going to start the automation process. For example, a company could subcontract an operations team that lacks the technical networking knowledge and just by providing them a certain set of inputs they could configure for X Access Points in Y networks an specific SSID with the desired parameters. The given inputs could be introduced by them in a web application and the underlying programming code would do the rest. In the end this translates to a very fast and reliable configuration that does not need to be done by the networking engineers. They can invest this extra time in other tasks. But automation not only does wonders in terms of configuration, its also great for monitoring the status of networks/devices/ports, obtain wireless health information and any other information that can be extracted from the end systems.

One can periodically query for information and present all this information in one screen, allowing anyone to have a big picture of the state of the system. By gathering and processing the data from various sources, information that would not be possible to get otherwise can be obtained. This information can be used to gain knowledge and with this knowledge engineers can improve network performance, deduce possible sources that are causing a certain problem in the network, etc.

It's important to take into consideration that recurring daily/weekly tasks that require the gathering of information are great candidates to automate. With an automation that queries the needed data and does some processing the required information can be quickly obtained and presented to the engineer saving him/her from manually logging into many devices, checking certain configuration lines, etc.

In the end, automating everyday networking tasks whether to massively configure or to quickly gather useful information, provides a positive Return of Investment (ROI) [1].

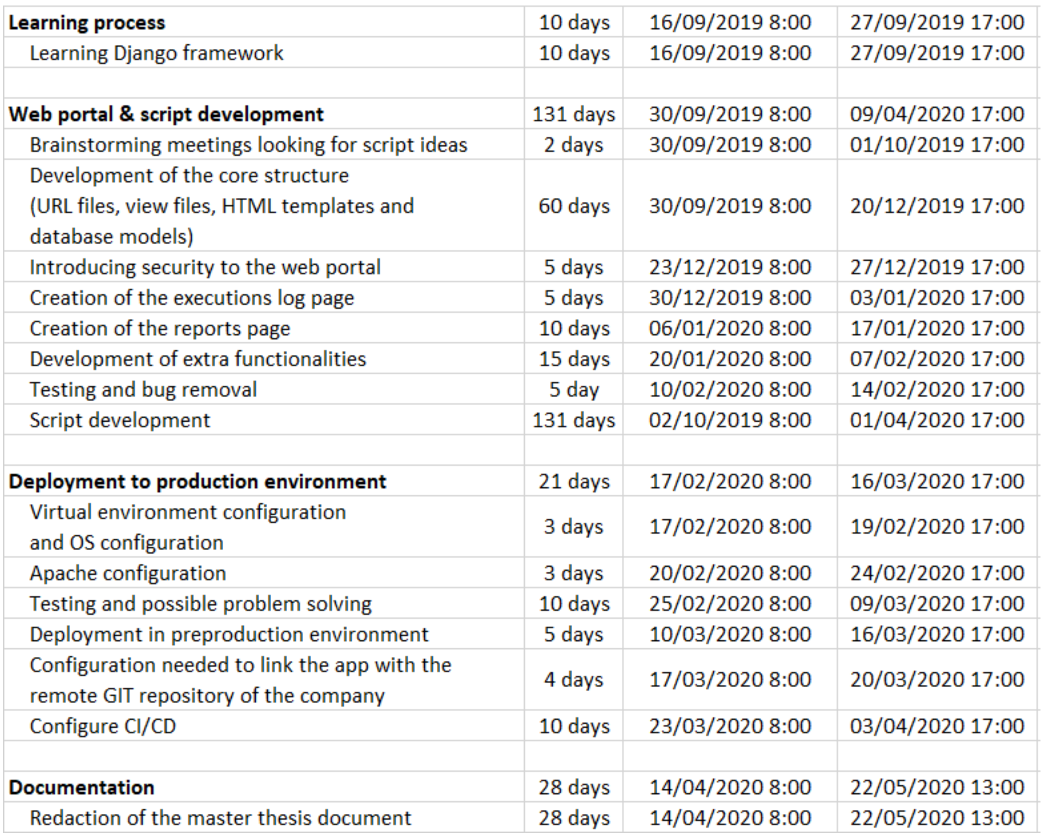
1.2 Requirements and specifications

The main requirements were clear from the start of the project:

- A publicly available web portal where the end users can call the automation scripts from a simple yet elegant frontend.
- Script integration in an Application Program Interface (API) so not only the portal is able to call this API but other systems in the company can also benefit from the scripts.
- Load balance the web portal between the given two UNIX servers using a Virtual IP Address (VIPA).
- Setup a pre-production environment where new scripts and changes in the platform are tested so no impact is made over the stable production environment.
- Link the application with a remote repository located inside the local area network of the company to make deployment changes easy and fast.

1.3 Work plan

The work plan was established by me at the start of the project. The work plan is shown in Fig. 1.1. All the tasks, its time duration and the correlation between tasks can be seen in Fig. 1.2



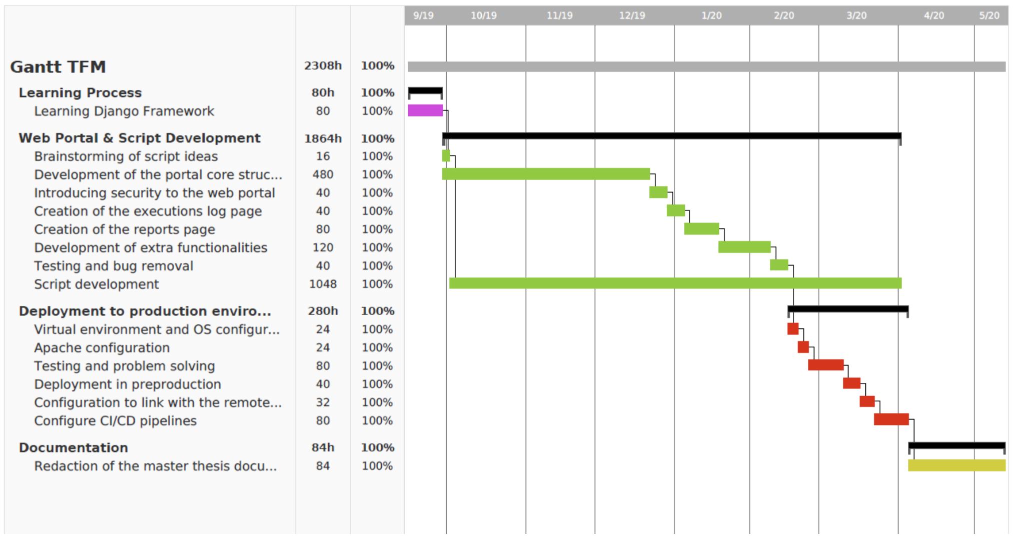
Learning process	10 days	16/09/2019 8:00	27/09/2019 17:00
Learning Django framework	10 days	16/09/2019 8:00	27/09/2019 17:00
Web portal & script development	131 days	30/09/2019 8:00	09/04/2020 17:00
Brainstorming meetings looking for script ideas	2 days	30/09/2019 8:00	01/10/2019 17:00
Development of the core structure (URL files, view files, HTML templates and database models)	60 days	30/09/2019 8:00	20/12/2019 17:00
Introducing security to the web portal	5 days	23/12/2019 8:00	27/12/2019 17:00
Creation of the executions log page	5 days	30/12/2019 8:00	03/01/2020 17:00
Creation of the reports page	10 days	06/01/2020 8:00	17/01/2020 17:00
Development of extra functionalities	15 days	20/01/2020 8:00	07/02/2020 17:00
Testing and bug removal	5 day	10/02/2020 8:00	14/02/2020 17:00
Script development	131 days	02/10/2019 8:00	01/04/2020 17:00
Deployment to production environment	21 days	17/02/2020 8:00	16/03/2020 17:00
Virtual environment configuration and OS configuration	3 days	17/02/2020 8:00	19/02/2020 17:00
Apache configuration	3 days	20/02/2020 8:00	24/02/2020 17:00
Testing and possible problem solving	10 days	25/02/2020 8:00	09/03/2020 17:00
Deployment in preproduction environment	5 days	10/03/2020 8:00	16/03/2020 17:00
Configuration needed to link the app with the remote GIT repository of the company	4 days	17/03/2020 8:00	20/03/2020 17:00
Configure CI/CD	10 days	23/03/2020 8:00	03/04/2020 17:00
Documentation	28 days	14/04/2020 8:00	22/05/2020 13:00
Redaction of the master thesis document	28 days	14/04/2020 8:00	22/05/2020 13:00

Figure 1.1: Work plan

1.4 Project team structure & project status

This project has been supervised by my work supervisor, David Salvador Rodrigo and by the project's tutor, Prof. Marcos Postigo-Boix. My work supervisor helped me with the administrative issues I found inside the company and other work related issues like scheduling meetings with the engineering teams when needed.

This project was started and finished by me. I was contracted to develop a web portal where automation processes could run underneath and this project is the result of 8 months of developing. For a few months I worked alongside a small group of trainees. They helped in the creation of some of the scripts which were code reviewed and approved by me.



Gantt TFM	**2308h**	**100%**
Learning Process	**80h**	**100%**
Learning Django Framework	80	100%
Web Portal & Script Development	**1864h**	**100%**
Brainstorming of script ideas	16	100%
Development of the portal core struc...	480	100%
Introducing security to the web portal	40	100%
Creation of the executions log page	40	100%
Creation of the reports page	80	100%
Development of extra functionalities	120	100%
Testing and bug removal	40	100%
Script development	1048	100%
Deployment to production enviro...	**280h**	**100%**
Virtual environment and OS configur...	24	100%
Apache configuration	24	100%
Testing and problem solving	80	100%
Deployment in preproduction	40	100%
Configuration to link with the remote...	32	100%
Configure CI/CD pipelines	80	100%
Documentation	**84h**	**100%**
Redaction of the master thesis docu...	84	100%

Figure 1.2: Gantt diagram for this project

1.5 Deviations and incidents

While the web portal development and the script development went as planned and approximately followed the theoretical finish dates specified in the initial work plan, two main problems arose when deploying from the local environment towards the production environment which translated in more days of work than the predicted ones.

1.5.1 Problems when deploying to the production environment

One of the problems was the connectivity issues that the machine had. The machine did not have connectivity towards:

- Certain network devices where the scripts needed to connect
- Certain servers where the scripts connected to gather information
- The remote GIT repository
- The Python Package Index (PyPI) [2]

The network devices were missing Access Control List (ACL) rules and the servers, the remote GIT repository and PyPI were not accessible due to strict firewall rules.

Since the machine had this very limited access to Internet, in order to install the needed Python modules, a simple pip install was not possible. The workaround was to download the modules from the Python Package Index and manually install them. This translated to a big problem, the famous “dependency hell” where some modules needed other minor modules to work. Furthermore, some modules had to be downgraded in version for some installations to work. This downgrade also supposed some modifications in the portal code itself.

The other problem was a bureaucratic one. New apps in the company had to be approved by a committee. This committee was rather strict, and the process advanced at a very slow rate. In the end, the committee accepted the app and the firewall rules were updated to connect to the remote GIT repository.

Chapter 2

State of the art

2.1 The evolution of web developing

At first, web apps were nothing more than a bunch of HTML, CSS and JavaScript files put together, related between them. Before frameworks, web apps were already completely customizable and dynamic using proper CSS and JavaScript. A good developer was able to make great web apps if he/she had enough skill/knowledge.

In the present day, frameworks have appeared and taking into consideration that they do not improve what the user ends up seeing and his/her interactions with the frontend, which in the end is the final objective, then one might ask why are they widely used nowadays [3].

2.2 Frameworks and why are they used

Frameworks have been adopted by the vast majority of web developers because they provide:

- Modularity

  As the app grows larger, code needs to be well structured in folders and files depending on what the code does. In the past, large apps suffered when the app grew, there were scalability problems as the number of JavaScript and CSS files increased rapidly and there were lots of repeated code between files. By providing a defined structure, a certain piece of code can be easily searched for. If we take as an example the Django framework [4], Django structures the code in a very specific way. The typical Django app structure can be seen in figure 2.1.

Inside the models.py file the database models are defined to make queries to the database unrelated to the specific database used in the project, inside views.py file, the logic to retrieve and process data when the user demands it is implemented, inside urls.py the routing of the app is established, inside the templates folder there are all the .html files where the views.py file send the obtained data to render it, etc.

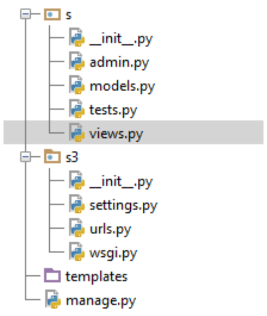


Figure 2.1: Typical Django project structure

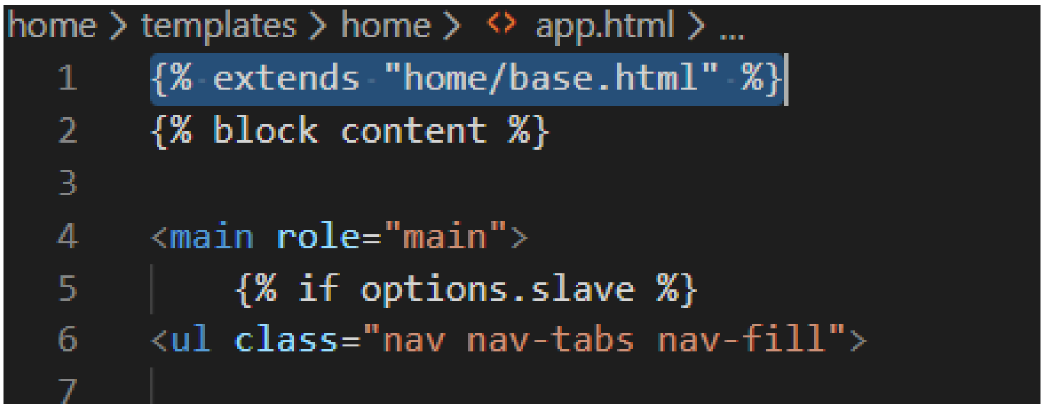
- Faster development

  Frameworks provide out of the box functionalities by simply calling already built-in functions/methods. By just reading the framework documentation and learning how to use them, the developer can rapidly incorporate functionalities that otherwise would be hard to implement and also very time consuming. Examples include authentication functions, session management functions, database operation functions, form validation functions and functions to provide security against malicious attackers.

- Security

  Most frameworks include tools to protect against the most common malicious attacks which are Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF) and SQL Injection [5]. To give an example, in Django, all the HTTP requests that the user sends include a CSRF token. In the code you can include a short sentence that will not load specific functionalities if the user does not have a valid CSRF token. This protects the application from CSRF attacks. The short sentence can be seen in figure 2.2.

Figure 2.2: Sentence to check that the user has a valid CSRF token in Django


home > templates > home > app.html > ...

```
{% extends "home/base.html" %}
{% block content %}

<main role="main">
    {% if options.slave %}
<ul class="nav nav-tabs nav-fill">
```

Figure 2.3: Example of code extension in the Django framework

- Code extension

  Most frameworks allow to extend some piece of code which will be used in several other files. This ensures that there is no repeated code and any change in that code translates to all instances that use that code. In Django you can extend code by simply using one sentence as shown in figure 2.3.

- Easier code legibility

  Since the code uses well defined standard functions and a certain specific structure, it is easier to pick up by someone who is new to the code itself but knows how the framework works.

2.3 The importance of version control

Aside from using frameworks, nowadays it is common practice to implement version control linking the whole web application with a remote repository like GitHub [6]. Version control allows developers to have multiple versions of code in case they want to roll back to a previous working code. Having a remote repository also allows to publish newer versions of the web application once they are ready directly into the production environment just with a simple command. Version control also makes it so the developer does not end having several local copies of the code corresponding to different versions.

2.4 The CI/CD pipeline

Once the web app is linked with the remote repository, the last trend in the DevOps world is to implement a CI/CD pipeline [7] which essentially is an automated process that triggers when new code is published in the remote repository. This process initiates code builds, runs some tests and finally if everything is fine, deploys the code automatically in the production environment. With this, the developers can ensure that nothing will break in production and new functionalities are served as soon as possible to the client.

Chapter 3

Project development

3.1 Brainstorming of script ideas

The first step in the process was to determine what to automate based in several considerations. In order to define this, many brainstorming meetings were done with the LAN & WiFi networking team. In these meetings a lot of ideas were brought to the table and the networking team established a priority in development order. The maximum priority was given to automations needed for the completion of some of their current projects which deadlines were soon and that required massive configuration deployments in networking equipment.

Aside from that, the priority was given to the automations that would overall save more time. The networking team has many recurrent tasks that are done daily/weekly and that at the end of the year represent a lot of time. The objective in many automations is to reduce the time consumed in doing these recurrent tasks.

It's also important to mention that not all the development petitions were established in this early brainstorming meetings and the needs for new automations were constantly being reported as the networking team had new ideas or received new urgent projects.

After receiving the wish list from the networking team, the petitions were analyzed in order to determine if they were technically feasible to implement in code, how would be the best way to do it and finally if the petition would result in a ROI greater than 1. A Return of Investment in this scenario represents the ratio between the gain in number of hours with respect to manually doing the procedure and the hours used to develop the script. If the script reduces the time with respect to doing it manually by 30 hours but it's very hard to develop and it can take approximately two weeks to develop then it has a ROI lower than 1 and considerations to discard its development should be made.

3.2 Programming language and framework decision

The company pushes forward to establish Python as the main programming language for its scripts and thus Python was the selected programming language for the development of the scripts. While I'm fluent writing Python code, it's important to mention that it was not chosen by me, it was imposed by the company standards. Following this, an analysis of three big Python frameworks was done.

3.2.1 Django

Django is a high-level web framework which is ideal to develop web apps with Python as the backend [4]. Django is one the best frameworks out there for many reasons:

- Provides fast app development. It simplifies much of the hassle of traditional web development which in the end translates to a quicker app completion time.
- It's a secure framework. Django has built-in modules to provide authentication and session management. Its also easy to implement protection against XSS, CSRF and SQL injection [5].
- It's scalable. Sites like Instagram, Pinterest and Bitbucket receive hundreds of thousands of daily requests and have Django as their framework of choice.
- It's open source, same as Python.
- It has high-quality documentation and the biggest community out of the Python frameworks so there is a lot of information online.
- Its easy to build an API using it.
- Has a prebuilt administration page that can be used to easily make changes in the database.

3.2.2 Flask

Flask is the second most popular Python framework just behind Django [8]. Since it's a very lightweight and minimalist framework, some people fit it inside the category of "Microframework". Its main positive qualities are:

- It's simple and fast to develop in.
- It's an extremely lightweight framework.
- Uses Non-relational databases, which are becoming very popular nowadays because of its simplicity.
- It's very easy to build APIs with Flask.

3.2.3 Pyramid

Pyramid is a minimalist, fast, lightweight and easy to use framework [9]. As Flask, it is considered a "Microframework". It was one of the first frameworks to be compatible with Python 3. Its great for very small applications since it's a simple framework. The community while small is quickly developing. Every Pyramid release has 100% statement coverage and 95% decision/condition coverage. The documentation is complete and friendly to newcomers.

3.3 Analysis of the pros and cons of each Python framework

After analyzing the pros/cons of each option, the winner was clear. Django was one step ahead in many areas. The only main positive points that Flask and Pyramid had with respect to Django were the simplicity and the lightness. These two positive points are far overweight by the advantages that Django offers:

- The app will grow and receive a lot of requests, so it needs a framework that scales very well and Django has the better scalability.
- Having good security is mandatory and Django prebuilt security modules are easy to implement and provide great security mechanisms.
- Django has the biggest community among the Python frameworks. There is a lot of information published online. Solutions to problems and answers to questions are easily found with a quick search. This greatly helps in the developing process.

3.4 Django learning topics

After deciding that Python would be the main programming language and Django would be the framework, the process of learning Django started [10]. Besides knowing Python, in order to develop the web portal, there were many things to learn related to Django:

- How the URL routing between views work.
- How to work with the SQLite database.
- How to implement the API in Django.
- How to authenticate users and create sessions.

3.5 Structure of the Django Web Application

When the basics were clear, the development of the web portal started in the local machine i.e. the company laptop. First the core structure of the web portal was defined. This core structure can be seen in figure 3.1. It's important to note that the whole Django application is on the same physical server. When the user sends a request to execute a script with certain inputs,

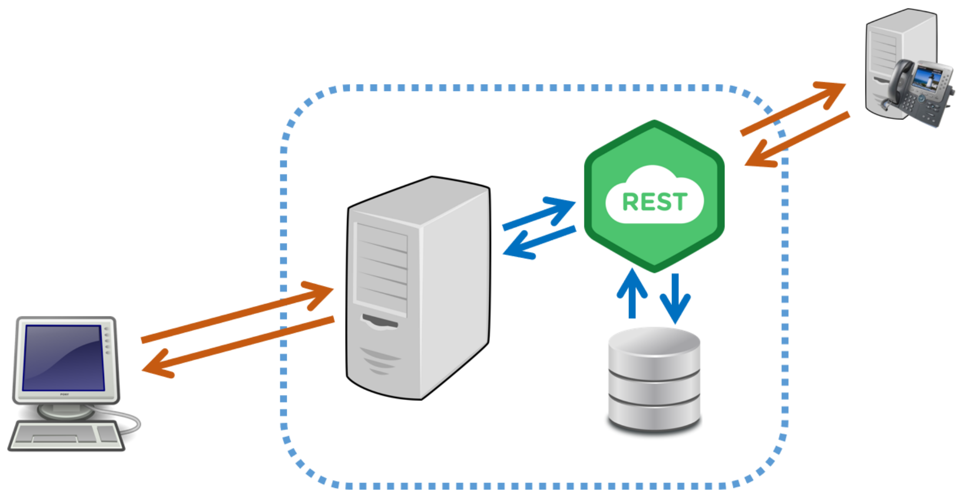


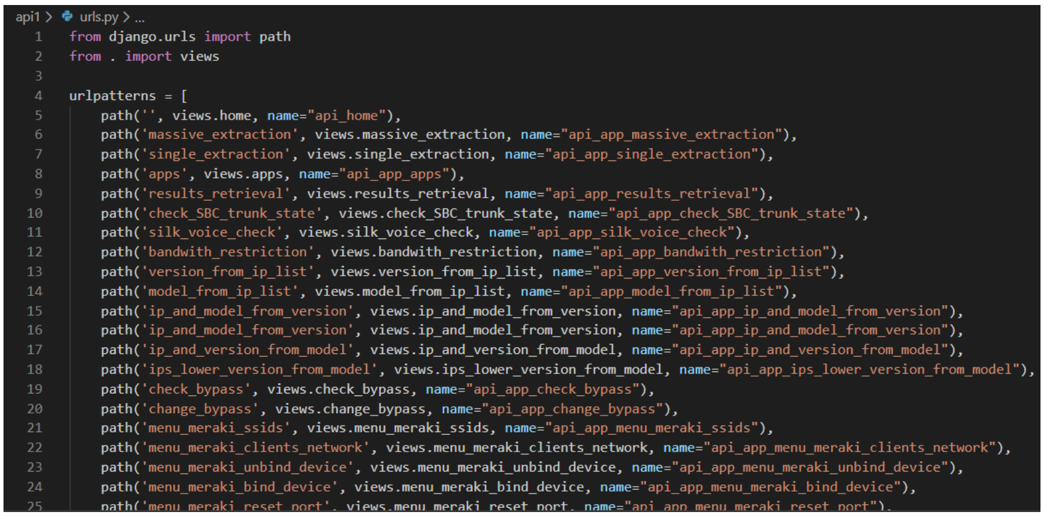
Figure 3.1: Django web portal core structure.

the Django app receives the request and sends a new request with all the information that the user introduced directly towards the API which receives it. The API interacts with the database if any previous data is needed before script execution. When the script finishes execution, the result is stored in the database and sent back to the frontend for the user to see.

The Django application is composed by many things, but the most important ones are the url.py files, the views.py files, the .html template files and the database.

3.6 The url.py files

These files define the URL routing of the application. URLs that match with the URL patterns described in the urls.py file are sent to the corresponding function in the views.py file. Matching is done from top to bottom in the urls.py file. While URL exact matching was implemented in this project, Django also allows the use of regular expression matching. In figure 3.2, the URLs of the API can be seen, each URL maps to the function in the api1/views.py file that ends up executing the script.

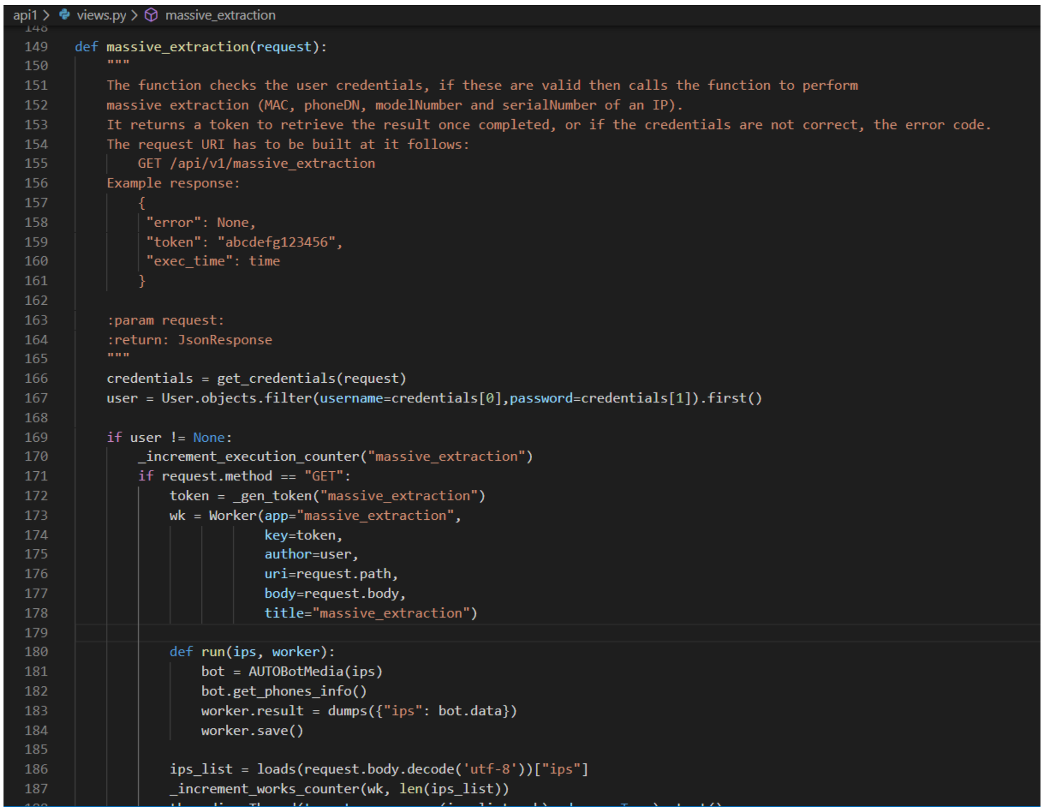


```python
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name="api_home"),
    path('massive_extraction', views.massive_extraction, name="api_app_massive_extraction"),
    path('single_extraction', views.single_extraction, name="api_app_single_extraction"),
    path('apps', views.apps, name="api_app_apps"),
    path('results_retrieval', views.results_retrieval, name="api_app_results_retrieval"),
    path('check_SBC_trunk_state', views.check_SBC_trunk_state, name="api_app_check_SBC_trunk_state"),
    path('silk_voice_check', views.silk_voice_check, name="api_app_silk_voice_check"),
    path('bandwith_restriction', views.bandwith_restriction, name="api_app_bandwith_restriction"),
    path('version_from_ip_list', views.version_from_ip_list, name="api_app_version_from_ip_list"),
    path('model_from_ip_list', views.model_from_ip_list, name="api_app_model_from_ip_list"),
    path('ip_and_model_from_version', views.ip_and_model_from_version, name="api_app_ip_and_model_from_version"),
    path('ip_and_model_from_version', views.ip_and_model_from_version, name="api_app_ip_and_model_from_version"),
    path('ip_and_version_from_model', views.ip_and_version_from_model, name="api_app_ip_and_version_from_model"),
    path('ips_lower_version_from_model', views.ips_lower_version_from_model, name="api_app_ips_lower_version_from_model"),
    path('check_bypass', views.check_bypass, name="api_app_check_bypass"),
    path('change_bypass', views.change_bypass, name="api_app_change_bypass"),
    path('menu_meraki_ssids', views.menu_meraki_ssids, name="api_app_menu_meraki_ssids"),
    path('menu_meraki_clients_network', views.menu_meraki_clients_network, name="api_app_menu_meraki_clients_network"),
    path('menu_meraki_unbind_device', views.menu_meraki_unbind_device, name="api_app_menu_meraki_unbind_device"),
    path('menu_meraki_bind_device', views.menu_meraki_bind_device, name="api_app_menu_meraki_bind_device"),
    path('menu_meraki_reset_port', views.menu_meraki_reset_port, name="api_app_menu_meraki_reset_port"),
```

Figure 3.2: Example of url.py file.

3.7 The views.py files

The functions on a views.py file are called when the given URL sent by the user matches the corresponding URL pattern in the urls.py file. Parameters sent via HTTP call enter the corresponding function via parameters or request body. In the views.py file of the API, the function executes the script code with the given input parameters. When the script code execution ends, the result is returned to the views function and passed as context to the corresponding .html file to show the results to the user that executed the script. An example of a function in a views.py file can be seen in figure 3.3.

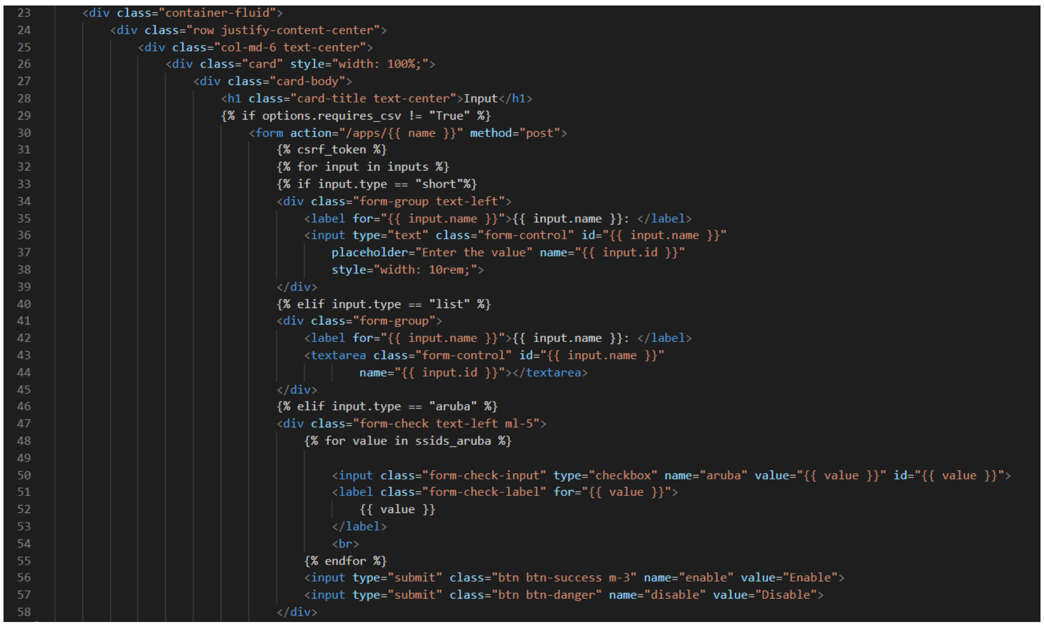


```python
def massive_extraction(request):
    """
    The function checks the user credentials, if these are valid then calls the function to perform
    massive extraction (MAC, phoneDN, modelNumber and serialNumber of an IP).
    It returns a token to retrieve the result once completed, or if the credentials are not correct, the error code.
    The request URI has to be built at it follows:
        GET /api/v1/massive_extraction
    Example response:
        {
         "error": None,
         "token": "abcdefg123456",
         "exec_time": time
        }

    :param request:
    :return: JsonResponse
    """
    credentials = get_credentials(request)
    user = User.objects.filter(username=credentials[0],password=credentials[1]).first()

    if user != None:
        _increment_execution_counter("massive_extraction")
        if request.method == "GET":
            token = _gen_token("massive_extraction")
            wk = Worker(app="massive_extraction",
                        key=token,
                        author=user,
                        uri=request.path,
                        body=request.body,
                        title="massive_extraction")

            def run(ips, worker):
                bot = AUTOBotMedia(ips)
                bot.get_phones_info()
                worker.result = dumps({"ips": bot.data})
                worker.save()

            ips_list = loads(request.body.decode('utf-8'))["ips"]
            _increment_works_counter(wk, len(ips_list))
```

Figure 3.3: Example of views.py file.

3.8 The .html template files

In the views.py function, Django renders the corresponding .html template file with a certain context. The context is in JavaScript Object Notation format (JSON) [11] and it is sent to the .html. Data in the context is displayed in the .html if the .html is properly parameterized. An example of .html with the syntax code on how to access the context data is shown in figure 3.4.



```html
    <div class="container-fluid">
        <div class="row justify-content-center">
            <div class="col-md-6 text-center">
                <div class="card" style="width: 100%;">
                    <div class="card-body">
                        <h1 class="card-title text-center">Input</h1>
                        {% if options.requires_csv != "True" %}
                            <form action="/apps/{{ name }}" method="post">
                                {% csrf_token %}
                                {% for input in inputs %}
                                {% if input.type == "short"%}
                                <div class="form-group text-left">
                                    <label for="{{ input.name }}">{{ input.name }}: </label>
                                    <input type="text" class="form-control" id="{{ input.name }}"
                                        placeholder="Enter the value" name="{{ input.id }}"
                                        style="width: 10rem;">
                                </div>
                                {% elif input.type == "list" %}
                                <div class="form-group">
                                    <label for="{{ input.name }}">{{ input.name }}: </label>
                                    <textarea class="form-control" id="{{ input.name }}"
                                            name="{{ input.id }}"></textarea>
                                </div>
                                {% elif input.type == "aruba" %}
                                <div class="form-check text-left ml-5">
                                    {% for value in ssids_aruba %}

                                        <input class="form-check-input" type="checkbox" name="aruba" value="{{ value }}" id="{{ value }}">
                                        <label class="form-check-label" for="{{ value }}">
                                            {{ value }}
                                        </label>
                                        <br>
                                    {% endfor %}
                                    <input type="submit" class="btn btn-success m-3" name="enable" value="Enable">
                                    <input type="submit" class="btn btn-danger" name="disable" value="Disable">
                                </div>
```

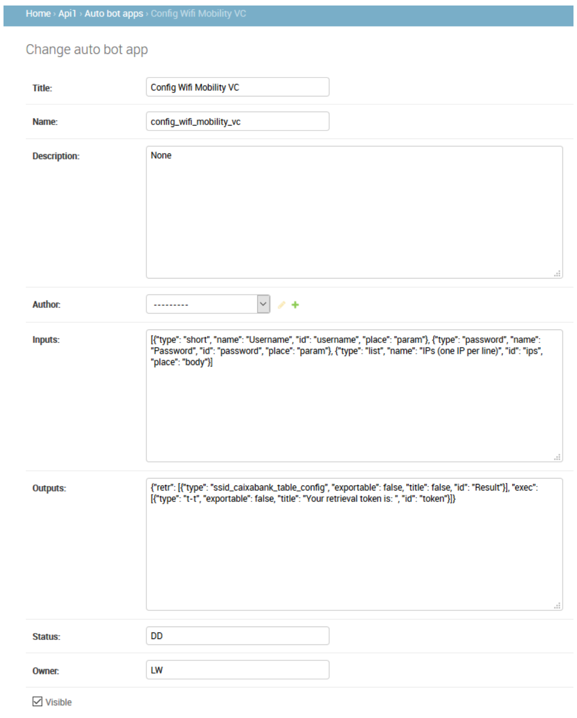
Figure 3.4: Example of .html file.

3.9 The database

The web portal uses a SQLite database [12]. This database contains three models which are defined in the models.py file, the app model, the worker model and the credentials model.

The app model defines the scripts. There is one record in the database for each script. It defines the inputs, outputs, name, etc. The worker model defines a particular execution of a script, like an execution log. It defines the ID which can be used to retrieve the result, the script name and the result of the execution. The credentials model stores credentials which are previously encrypted. With this, some scripts can get the credentials needed to work without user input and without directly referencing the credentials in the code.

The database can be managed directly through a Graphical User Interface (GUI) in Django. This GUI can only be accessed by admin users. Examples of records of the app model and the worker model seen in this GUI are shown in figures 3.5 and 3.6 respectively.



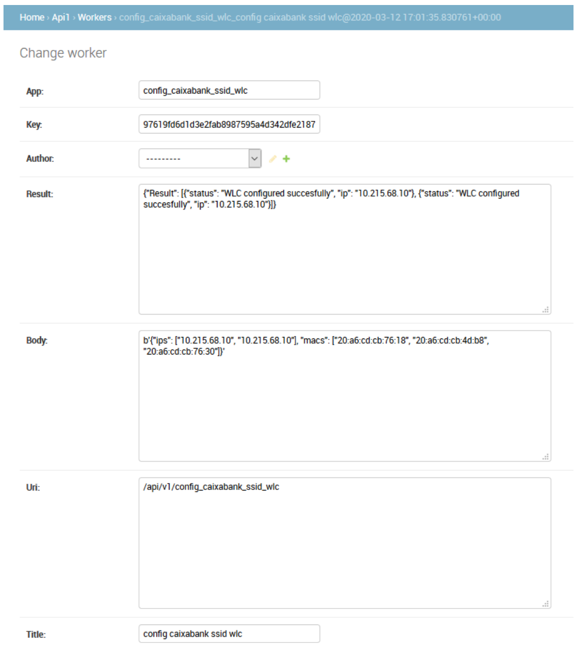
Home › Api1 › Auto bot apps › Config Wifi Mobility VC

Change auto bot app

Title:	Config Wifi Mobility VC
Name:	config_wifi_mobility_vc
Description:	None
Author:	---------
Inputs:	[{"type": "short", "name": "Username", "id": "username", "place": "param"}, {"type": "password", "name": "Password", "id": "password", "place": "param"}, {"type": "list", "name": "IPs (one IP per line)", "id": "ips", "place": "body"}]
Outputs:	{"retr": [{"type": "ssid_caixabank_table_config", "exportable": false, "title": false, "id": "Result"}], "exec": [{"type": "t-t", "exportable": false, "title": "Your retrieval token is: ", "id": "token"}]}
Status:	DD
Owner:	LW

☑ Visible

Figure 3.5: Instance of the app model. It defines a script.



Home › Api1 › Workers › config_caixabank_ssid_wlc_config caixabank ssid wlc@2020-03-12 17:01:35.830761+00:00

Change worker

App:	config_caixabank_ssid_wlc
Key:	97619fd6d1d3e2fab8987595a4d342dfe2187
Author:	---------
Result:	{"Result": [{"status": "WLC configured succesfully", "ip": "10.215.68.10"}, {"status": "WLC configured succesfully", "ip": "10.215.68.10"}]}
Body:	b'{"ips": ["10.215.68.10", "10.215.68.10"], "macs": ["20:a6:cd:cb:76:18", "20:a6:cd:cb:4d:b8", "20:a6:cd:cb:76:30"]}'
Uri:	/api/v1/config_caixabank_ssid_wlc
Title:	config caixabank ssid wlc

Figure 3.6: Instance of the worker model. It defines an execution log.

3.10 Workflow of the Django Web Portal

When the user tries to access the URL of the web portal, it is presented with the login page (figure 3.7). User creation by new users is not allowed for security reasons. After logging, the user is redirected towards the main

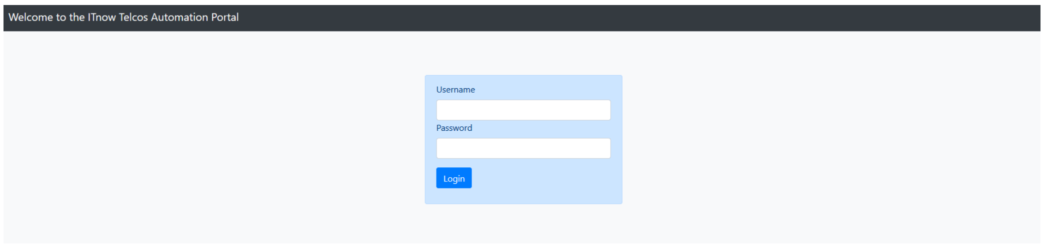


Figure 3.7: Login page of the automation web portal.

URL of the portal and is presented with the home page where a catalog with all the scripts is shown (figure 3.8). The routing towards the home page is done in the home/urls.py file, which points to the home/views.py file. Before rendering the home.html, the available scripts are obtained from the records of the app model in the database. This is done with a call to the API with the base URL "/api/v1/apps". Once the scripts and all of their info is retrieved, the home.html is rendered. Each script is allocated inside a box and all the scripts can be seen by scrolling down in the browser.

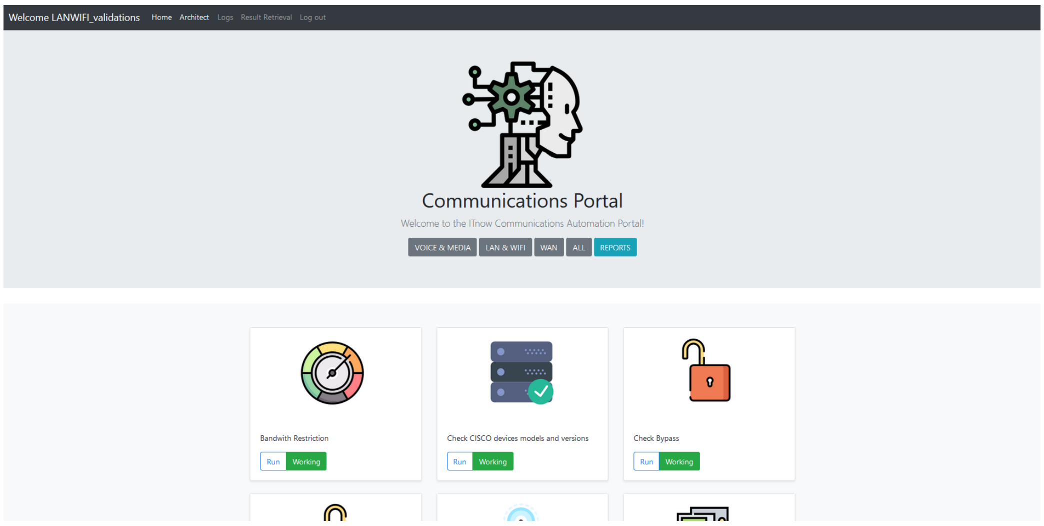
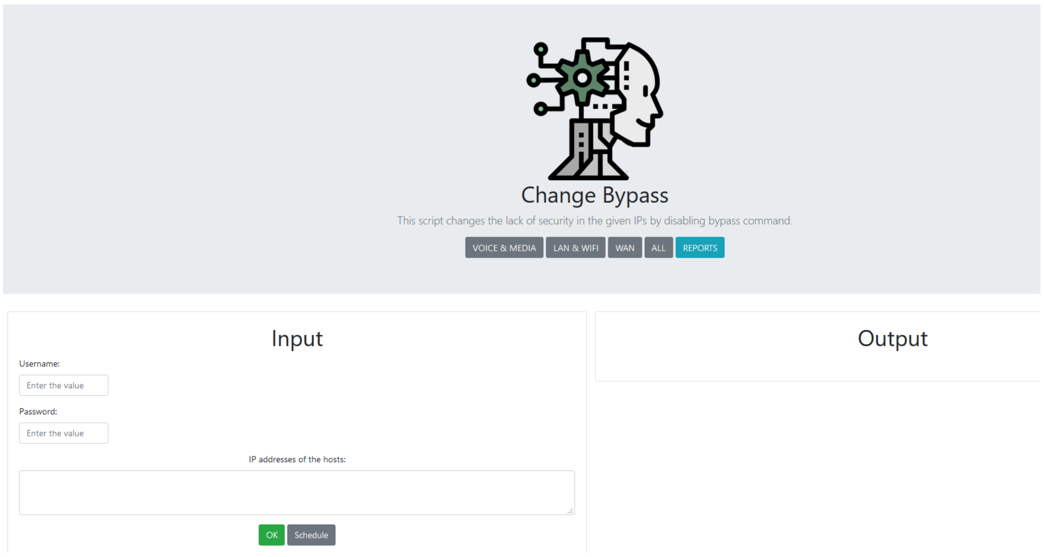


Figure 3.8: Home page of the automation web portal.

When a user clicks on the "Run" button of a certain script, the web portal shows him/her the .html page for that particular script, with its own specific inputs. An example can be seen on figure 3.9. To achieve this, the system sends an HTTP GET request to the home/urls.py file, with base URL "/apps/app_name" which calls the "app" function in home/views.py. Inside this function, the information of the script is retrieved using the "app_name" sent in the request, retrieving the script details from a global variable filled with the details of all scripts. With this, the web portal obtains the title and the inputs for that particular script.


Change Bypass

This script changes the lack of security in the given IPs by disabling bypass command.

Username: Enter the value

Password: Enter the value

IP addresses of the hosts:

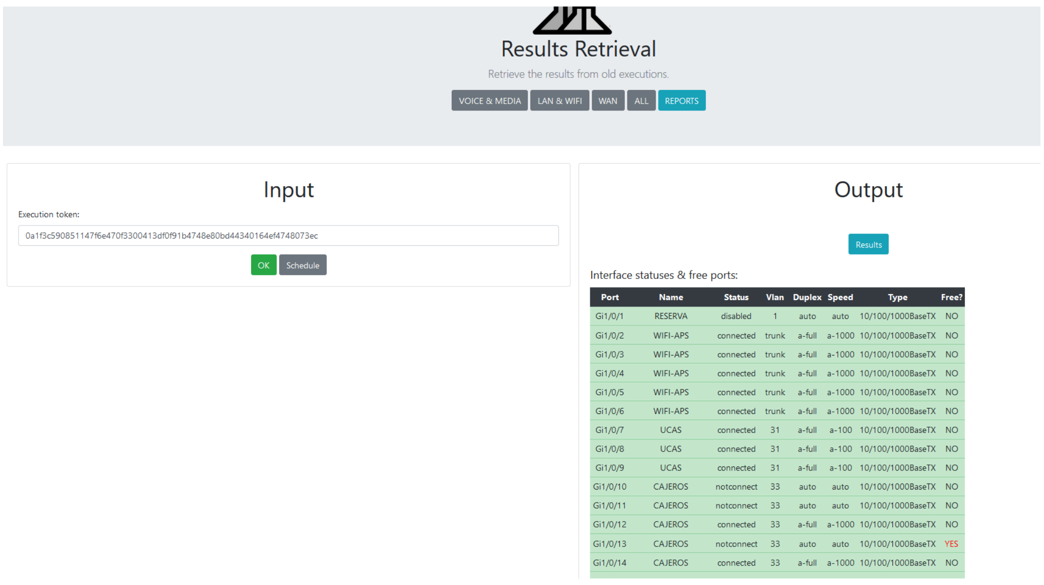
Figure 3.9: Script page for the "Change Bypass" script.

When the user fills the input form and clicks the "OK" button, an HTTP POST is sent to the home/urls.py which points to the "app" function in home/views.py file. Inside the function, the request is detected as a POST request. The data from the inputs form is obtained from the request and a new request is sent to the API with the data from the inputs form as parameters/body of this new request. This new request is sent to the base URL "/api/v1/app_name" which points to the corresponding function in the api1/views.py file. This function executes the script itself which is located in a separate file since routing code is separated from pure script code.

Once the script code finishes execution, the result data is returned to the API view function. In the same way, the view function returns the result data to the home view function in which is manipulated to match the expected JSON format and used to render the appropriate .html file to show the results to the user in the outputs section of the .html file.

Besides the results, a token is also generated and returned to retrieve data for that particular execution at any point in time. This token can be used in the "Results Retrieval" section which can be accessed from the navbar. The token is generated by hashing a certain fixed string with the current timestamp with microsecond precision. By doing it this way, it's not possible to cleverly manufacture a token to retrieve the results of a certain execution.

It's important to mention that while there are scripts that are able to return results in a few seconds there are others that take a long time to return them. For those scripts, the execution is threaded in order to not hang the portal while the script is running and only the token is returned. While the script is running if the user tries to retrieve the results with the token, the frontend displays a message telling him/her that the results are not yet ready. When the results are ready, the user can retrieve them running the "Results Retrieval" application and giving the token as input. An example can be seen in figure 3.10.



Execution token: 0a1f3c590851147f6e470f3300413df0f91b4748e80bd44340164ef4748073ec

Interface statuses & free ports:

Port	Name	Status	Vlan	Duplex	Speed	Type	Free?
Gi1/0/1	RESERVA	disabled	1	auto	auto	10/100/1000BaseTX	NO
Gi1/0/2	WIFI-APS	connected	trunk	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/3	WIFI-APS	connected	trunk	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/4	WIFI-APS	connected	trunk	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/5	WIFI-APS	connected	trunk	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/6	WIFI-APS	connected	trunk	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/7	UCAS	connected	31	a-full	a-100	10/100/1000BaseTX	NO
Gi1/0/8	UCAS	connected	31	a-full	a-100	10/100/1000BaseTX	NO
Gi1/0/9	UCAS	connected	31	a-full	a-100	10/100/1000BaseTX	NO
Gi1/0/10	CAJEROS	notconnect	33	auto	auto	10/100/1000BaseTX	NO
Gi1/0/11	CAJEROS	notconnect	33	auto	auto	10/100/1000BaseTX	NO
Gi1/0/12	CAJEROS	connected	33	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/13	CAJEROS	notconnect	33	auto	auto	10/100/1000BaseTX	YES
Gi1/0/14	CAJEROS	connected	33	a-full	a-1000	10/100/1000BaseTX	NO

Figure 3.10: Result retrieval of an execution of the interface status & free ports script.

The "Results Retrieval" application uses the token given in the inputs form to perform a query to the database. It searches for the record of the worker model that has the value of the given token and returns the results of the execution.

Once the core functionality of the Django web portal was ready, other important additions were developed.

3.11 Securing the web portal

After having ready the core functionality of the web portal, the next mandatory development was to make the web portal secure.

First of all, the views were protected from not logged in users. Then the API was protected from individuals without the proper authentication credentials. Django comes with a pretty good built-in user authentication system. The "django.contrib.auth" model together with the "login_required" decorator was used to protect the views. Basic Authentication together with HTTPS was used to protect the API from unauthorized individuals.

Form inputs were sanitized with query parameters before being sent to the database to avoid SQL injection and CSRF tokens were included to avoid CSRF attacks. XSS attacks are also avoided in the framework by default since Django templates escape specific characters which are particularly dangerous to HTML.

3.12 Logs page

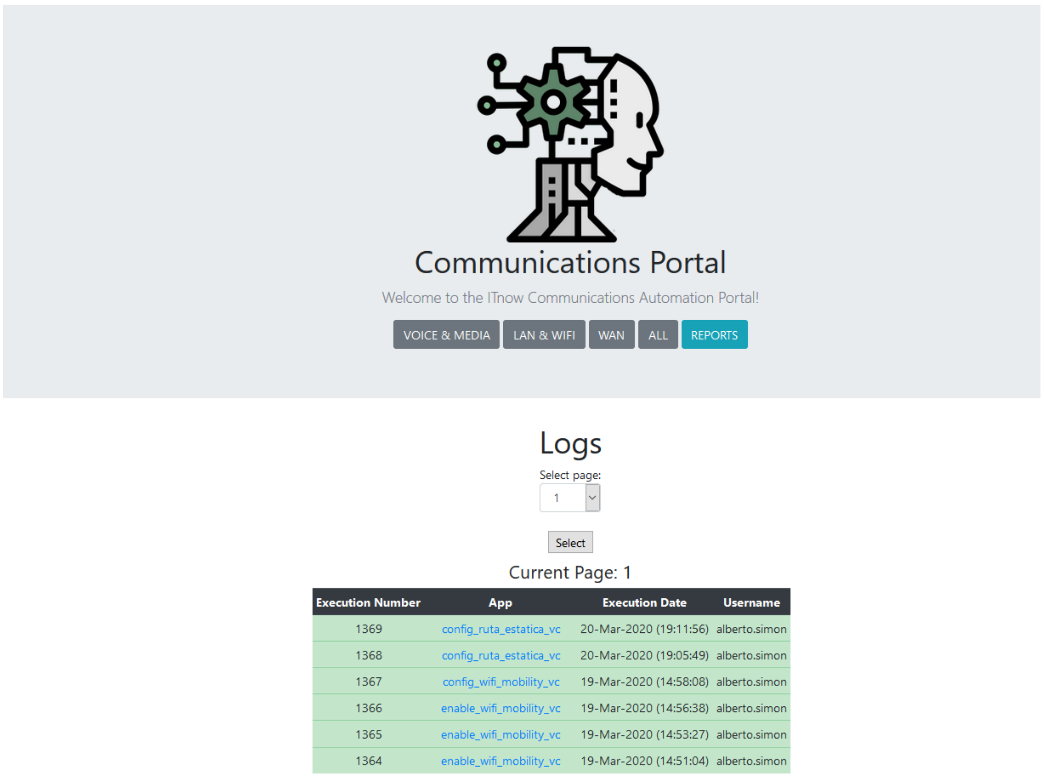
After securing the web portal, an execution logs page was developed. The link to the logs page was incorporated to the navbar and when requested, the "logs" function from the home/views.py file is called. Inside this function a call to the API is sent requesting the execution logs. The page number is sent as a parameter to provide pagination. It defines which records are retrieved. Inside this function the database is queried, and records of the worker model are retrieved. The logs page can be seen in figure 3.11.

The result of the executions is not shown in the logs page itself. When clicking the name of a particular execution, a call to the "Results retrieval" application is made, together with the execution token and the "Results retrieval" page is rendered with the result of that particular execution.

3.13 Reports page

Another big functionality that was added was the reports page. The reports page allows the user to see through some graphs, the number of executions and works that any script/group of scripts has done between two points in time. Executions of a script are the number of times the script has been executed. Works are the number of devices the device has configured/queried for data.

Each script has a field in its database model that defines how many hours it takes to manually do the job for one device. By querying the database and



Execution Number	App	Execution Date	Username
1369	config_ruta_estatica_vc	20-Mar-2020 (19:11:56)	alberto.simon
1368	config_ruta_estatica_vc	20-Mar-2020 (19:05:49)	alberto.simon
1367	config_wifi_mobility_vc	19-Mar-2020 (14:58:08)	alberto.simon
1366	enable_wifi_mobility_vc	19-Mar-2020 (14:56:38)	alberto.simon
1365	enable_wifi_mobility_vc	19-Mar-2020 (14:53:27)	alberto.simon
1364	enable_wifi_mobility_vc	19-Mar-2020 (14:51:04)	alberto.simon

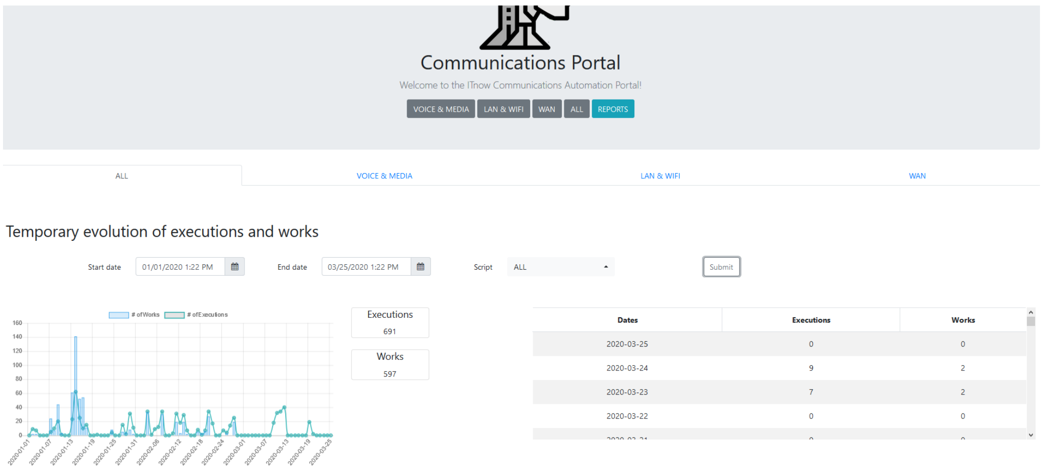
Figure 3.11: Logs page.

retrieving the worker records between two points in time, the total number of executions and works for that time period can be obtained. The graphs are then populated with that data.

Furthermore, by assigning for each script the manual hours that represent a work, total time saved by automation can be obtained and Return of Investment calculations can be done. In figure 3.12, the total number of executions/works for a group of scripts in a given time interval is shown. In figure 3.13, the total number of hours saved by automation for a group of scripts in a given time is shown.

3.14 Making the upload of new Python scripts transparent to new developers

The last and also biggest addition to the portal was the “Architect”. The ”Architect” is a system that through some forms and code in the backend, allows the upload of a Python script and adapts the code to the platform



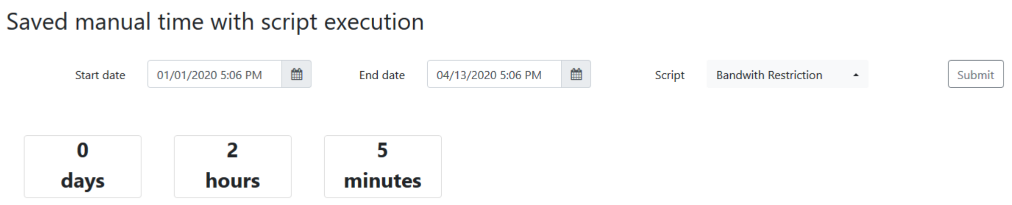
Temporary evolution of executions and works

Start date	01/01/2020 1:22 PM	End date	03/25/2020 1:22 PM	Script	ALL

Executions	691
Works	597

Dates	Executions	Works
2020-03-25	0	0
2020-03-24	9	2
2020-03-23	7	2
2020-03-22	0	0

Figure 3.12: Reports page. Number of executions/works over time.



Saved manual time with script execution

Start date	01/01/2020 5:06 PM	End date	04/13/2020 5:06 PM	Script	Bandwith Restriction

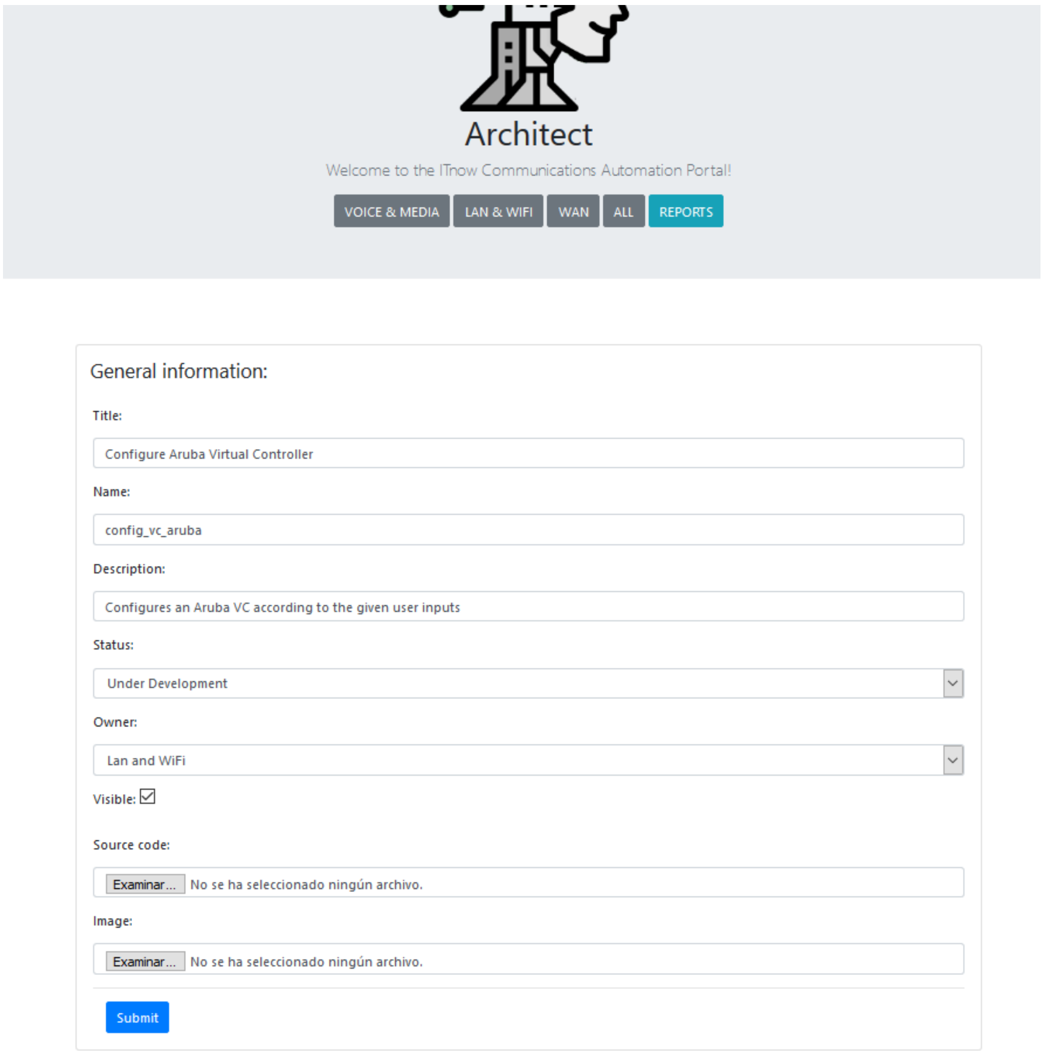
0 days	2 hours	5 minutes

Figure 3.13: Reports page. Total time saved.

standards. By putting the raw script code in a certain template, the "Architect" adapts it, searching for some specific comments with a certain format in the template and directly substitutes them for some code.

When calling the "Architect" through the frontend, some forms are presented. The forms ask for the inputs/outputs of the script and also for the source code of the script (the template). After correctly filling the forms, the "Architect" creates the script record in the app model of the database with the correct input/output format that the platform wants and it also modifies the template to add the platform specific code, like for example the code needed for storing the result of the script in the database.

This allows someone with no Django knowledge and/or platform knowledge to publish a script in the platform with just some small considerations in the template itself. One of the forms of the "Architect" is shown in figure 3.14.



General information:

Title: Configure Aruba Virtual Controller

Name: config_vc_aruba

Description: Configures an Aruba VC according to the given user inputs

Status: Under Development

Owner: Lan and WiFi

Visible: ☑

Source code: No se ha seleccionado ningún archivo.

Image: No se ha seleccionado ningún archivo.

Figure 3.14: First form of the "Architect". Used to create the script record in the database.

3.15 Moving the web portal to the production environment

Once all the desired functionality was added and working in the local environment, the task to move the portal to a production environment began. The application was deployed in a RedHat 7.7 Linux machine.

One of the big problems that arose when transferring the portal to production was the lack of superuser rights. The administrators of the machine were the “BACKENDS LAN-WAN” engineering team. A lot of coordination and working alongside them was needed in order to transfer the portal in a reasonable time.

The first objective was to successfully run without errors the Django application in the server. First of all, in order to not interfere with the Python modules already deployed in the production server, a virtual environment [13] was created to install inside it the specific modules needed for the Django application to work. Since the machine had very limited internet access due to strict firewall rules, the modules had to be manually installed, a simple pip install could not be done.

When all the modules were finally installed inside the virtual environment, writing rights inside a directory tree were given to me to upload all the Django files. After uploading the files, the Django application ran without errors.

The next step was to deploy the Django application using Apache Web Server. The configuration had to be done by the ”BACKENDS LAN-WAN” team and only certain inputs were needed from my side, like the URL where it will be available, the port where the Django app listens (port 8000 in this case) and a petition to the security team asking for the generation of a valid SSL certificate to configure SSL in Apache. After all these procedures and some minor modifications in the settings.py file, the Django application was accessible through the URL. Following this, the testing of all the functionalities began. Some minor code changes were needed for it to work like in the local environment.

After the Django app was ready in the production environment, the deployment on another production server and in the preproduction server began. The deployment in the second production server provides redundancy and allows the configuration of an active/passive VIPA to ensure that the Django web portal has 100% uptime. If the primary production server goes down, the URL then points to the secondary production server. All the VIP configuration was done by the corresponding team in the company and I only needed to provide some details about the application/servers to them.

The pre-production server allows the testing of new scripts and also new portal functionalities without affecting the production server. The configuration of these two servers went a lot smoother after knowing what had to be done and how to solve the problems that arise.

3.16 GIT implementation

GIT is a must in any serious development project, and this is no exception. It provides fast code deployment, versioning and allows branching. Having it installed in the local development machine and in the production environment allows to easily and quickly deploy into the production environment the already tested code in the local development environment.

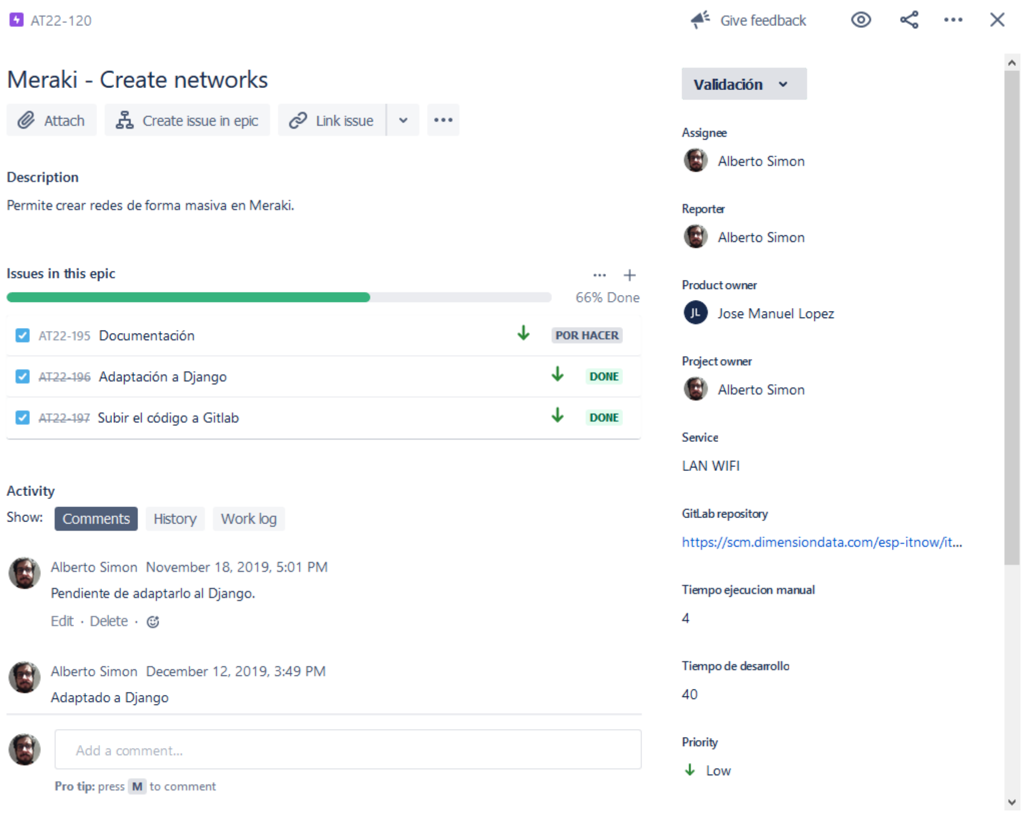
Versioning provides the option to roll back to previous code versions in the case that some unknown error appears in a newer version. In this project a master branch with stable code was used to publish the web portal to the production environment while stable new developments that were being tested were placed in the main development branch. A branch was created for each new functionality/script and when ready, that branch was merged with the main development branch. When the development branch is fully tested, the code was merged into the master branch.

3.17 Task tracking of the project

Jira is the software tool that was used to track all the issues related with this master's degree thesis [14] like the development of each individual script, the preparations needed to properly run Django in the machine (like proxy configurations, firewall configurations, VIPA configuration, installation of modules/dependencies, etc.), the developing of the framework itself, etc.

Jira allows the tracking of the project tasks, information about them, etc. Task fields are completely customizable and the same goes with task status workflows, the Kanban board, etc. If periodically filled, one can recap from one day to another the status and details of any task. This software was used to report the status of the project to the cotutor of the project. Jira also comes with some prebuilt automation tools that allow automatic workflow transitions, automatic subtask creation, etc. Those automations proved to be very useful.

In the end, a Jira project tailored to the project needs was configured and the time invested configuring the Jira platform was time well invested. A Jira task for this project can be seen in figure 3.15.


AT22-120

Give feedback

Meraki - Create networks

Attach | Create issue in epic | Link issue

Validación

Description

Permite crear redes de forma masiva en Meraki.

Issues in this epic

66% Done

AT22-195	Documentación	POR HACER
~~AT22-196~~	Adaptación a Django	DONE
~~AT22-197~~	Subir el código a Gitlab	DONE

Activity

Show: Comments | History | Work log

Alberto Simon November 18, 2019, 5:01 PM
Pendiente de adaptarlo al Django.
Edit · Delete ·

Alberto Simon December 12, 2019, 3:49 PM
Adaptado a Django

Add a comment...

Pro tip: press M to comment

Assignee	Alberto Simon
Reporter	Alberto Simon
Product owner	Jose Manuel Lopez
Project owner	Alberto Simon
Service	LAN WIFI
GitLab repository	https://scm.dimensiondata.com/esp-itnow/it...
Tiempo ejecucion manual	4
Tiempo de desarrollo	40
Priority	Low

Figure 3.15: A task in the Jira project.

Chapter 4

Results

In this chapter, the scripts inside the web portal will be discussed in detail and the execution results for each one of them will be shown.

4.1 Massive extraction of phone information

4.1.1 Inputs

- A list containing the IPs of the CISCO phones.

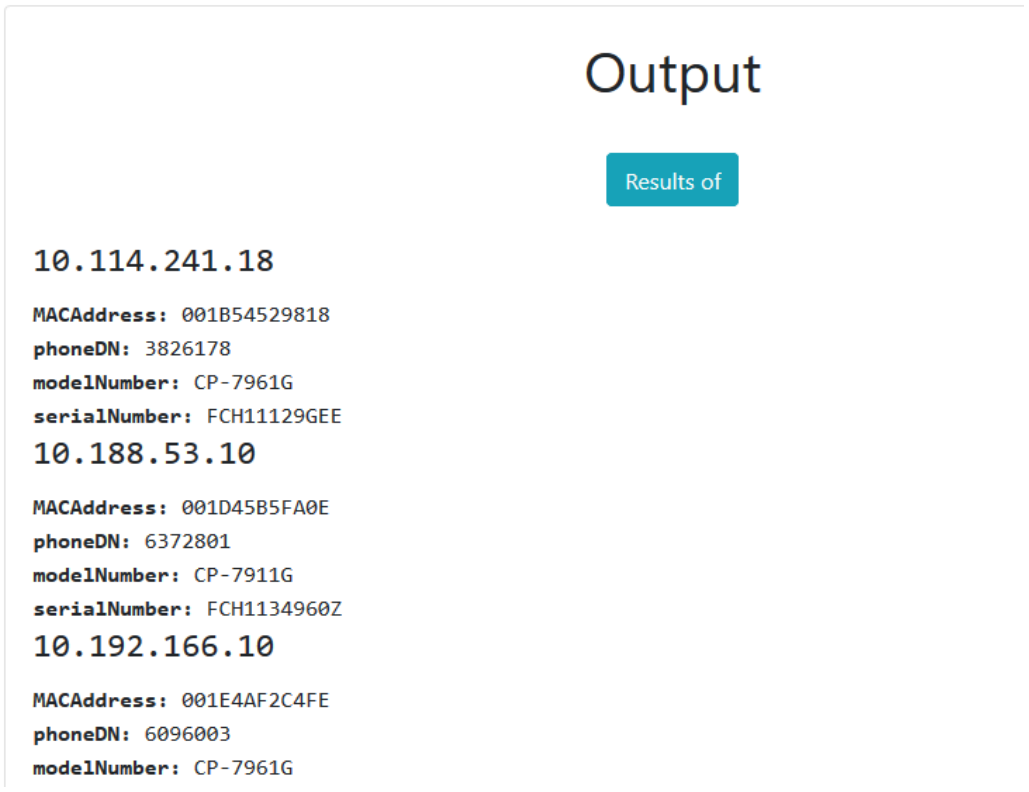
4.1.2 Outputs

- A dictionary containing the MAC address, the directory number, the model number and the serial number of the CISCO phone.

The voice & media engineering team wanted to extract certain information from a rather large subset of CISCO phones to put them in their inventory and to use some of that data to fill in some reports. The phones are rather new and have a built-in API that allows to query them for information via HTTP calls [15].

Since the needed information was possible to retrieve with an API call, a script was made that cycles through the list of CISCO phones and sends an HTTP call to every single one of them to retrieve the desired information. The “Content-Type” header had to be specified in the HTTP GET request with value ”application/x-www-form-urlencoded”. The requests Python module was used to send the HTTP request.

The phones respond in Extensible Markup Language (XML) format and to further process the data, the xmltodict Python module was used to convert the data from XML to a dictionary. The dictionary contains the MAC address, the directory number, the model number and the serial number of the CISCO phone. This dictionary is then passed to the context of the corresponding view to render the obtained results in the frontend. (Figure 4.1)



```
10.114.241.18
MACAddress: 001B54529818
phoneDN: 3826178
modelNumber: CP-7961G
serialNumber: FCH11129GEE
10.188.53.10
MACAddress: 001D45B5FA0E
phoneDN: 6372801
modelNumber: CP-7911G
serialNumber: FCH1134960Z
10.192.166.10
MACAddress: 001E4AF2C4FE
phoneDN: 6096003
modelNumber: CP-7961G
```

Figure 4.1: Execution results for the Massive Extraction script.

4.2 Check Session Border Controller SIP agents state & statistics

4.2.1 Inputs

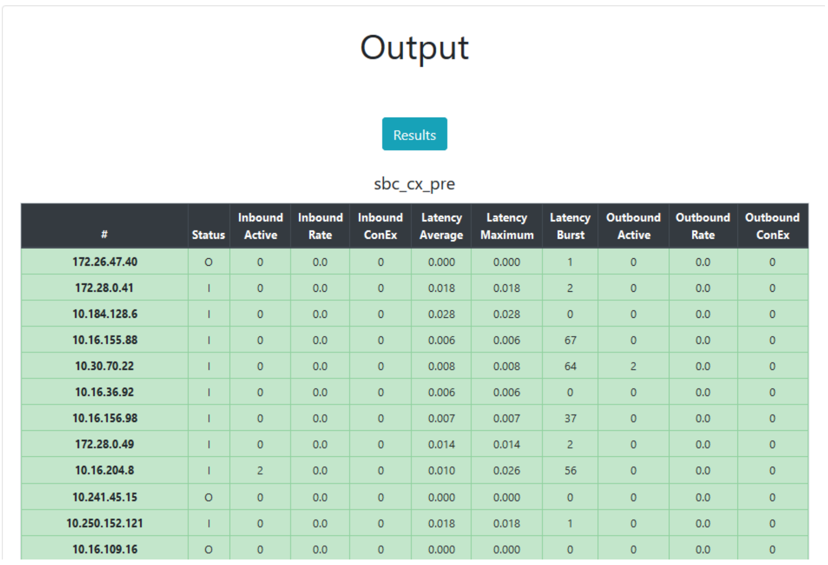
- A list containing the IPs of Session Border Controllers (SBCs)
- The credentials to be able to log into them via SSH
- An identifier which is used to compare the results of two different executions that share the same identifier

4.2.2 Outputs

- A dictionary that contains for each SBC, its SIP Agents and all its information.

The voice & media engineering team wanted to quickly compare the statistics and status of the SIP agents in a list of SBCs [16] between two different points in time. The problem they had is that when the firewall engineering team makes some changes in the network, sometimes some SIP proxies, SIP gateways or SIP endpoints lose connectivity, or their performance is affected.

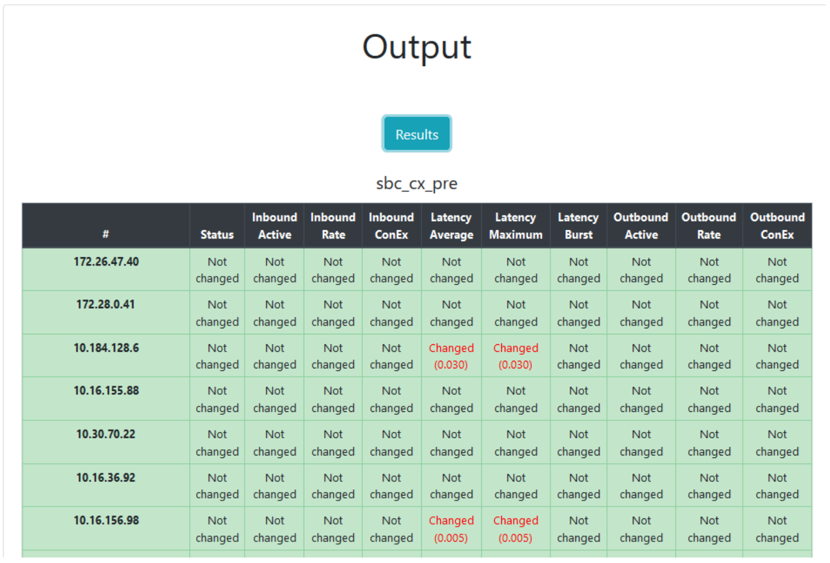
These firewall changes are done midnight because the changes might affect critical equipment in production which can impact service in the clients. With this script, the voice & media team creates a snapshot containing the SIP agents statistics the evening before the firewall changes (Figure 4.2) and then, executing the script again the morning after the firewall changes, a comparison of the two executions is made, showing the comparison results in a table and highlighting the differences found between the two executions (Figure 4.3)



sbc_cx_pre

#	Status	Inbound Active	Inbound Rate	Inbound ConEx	Latency Average	Latency Maximum	Latency Burst	Outbound Active	Outbound Rate	Outbound ConEx
172.26.47.40	O	0	0.0	0	0.000	0.000	1	0	0.0	0
172.28.0.41	I	0	0.0	0	0.018	0.018	2	0	0.0	0
10.184.128.6	I	0	0.0	0	0.028	0.028	0	0	0.0	0
10.16.155.88	I	0	0.0	0	0.006	0.006	67	0	0.0	0
10.30.70.22	I	0	0.0	0	0.008	0.008	64	2	0.0	0
10.16.36.92	I	0	0.0	0	0.006	0.006	0	0	0.0	0
10.16.156.98	I	0	0.0	0	0.007	0.007	37	0	0.0	0
172.28.0.49	I	0	0.0	0	0.014	0.014	2	0	0.0	0
10.16.204.8	I	2	0.0	0	0.010	0.026	56	0	0.0	0
10.241.45.15	O	0	0.0	0	0.000	0.000	0	0	0.0	0
10.250.152.121	I	0	0.0	0	0.018	0.018	1	0	0.0	0
10.16.109.16	O	0	0.0	0	0.000	0.000	0	0	0.0	0

Figure 4.2: First execution results for the Check Session Border Controller SIP agents state & statistics script.



sbc_cx_pre

#	Status	Inbound Active	Inbound Rate	Inbound ConEx	Latency Average	Latency Maximum	Latency Burst	Outbound Active	Outbound Rate	Outbound ConEx
172.26.47.40	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed
172.28.0.41	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed
10.184.128.6	Not changed	Not changed	Not changed	Not changed	Changed (0.030)	Changed (0.030)	Not changed	Not changed	Not changed	Not changed
10.16.155.88	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed
10.30.70.22	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed
10.16.36.92	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed	Not changed
10.16.156.98	Not changed	Not changed	Not changed	Not changed	Changed (0.005)	Changed (0.005)	Not changed	Not changed	Not changed	Not changed

Figure 4.3: Comparison results for the Check Session Border Controller SIP agents state & statistics script.

4.3 Check Alcatel SILK Phone Status

4.3.1 Inputs

- The IP of the server that contains the information of the phones
- The credentials to be able to log into the server via SSH
- An identifier which is used to compare the results of two different executions that share the same identifier

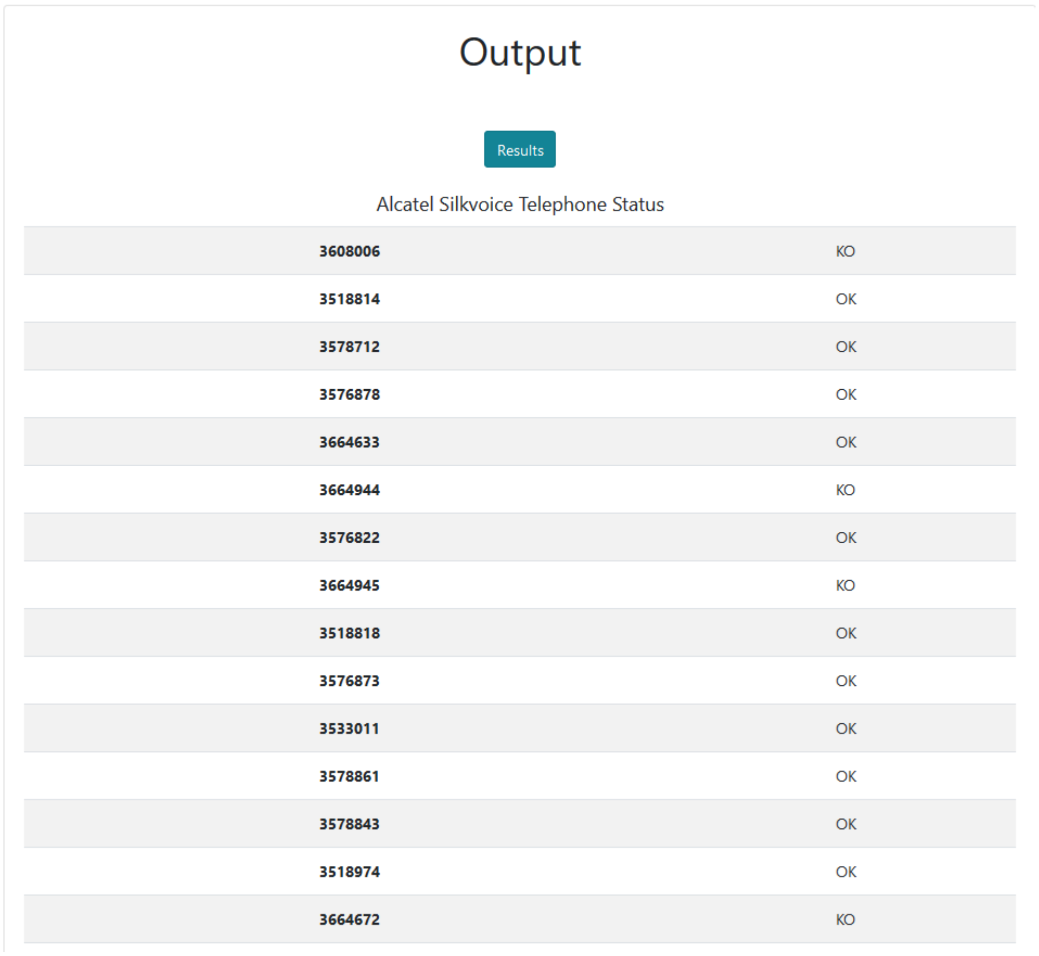
4.3.2 Outputs

- A dictionary containing all the registered phones and its status, this status can be either "OK" or "KO"

The voice & media engineering team wanted to compare the status of some Alcatel SILK phones before and after the firewall engineering team makes some changes in the network.

This script much like the "Check Session Border Controller SIP Agents State & Statistics" script, performs a comparison between two executions that share the same identifier. Since the current status can be obtained from an SSH accessible server, the script logs into the machine using the pexpect

Python module, extracts the info, parses it into a well formatted dictionary and displays the result in a table in the frontend. (Figure 4.4)



Output

Alcatel Silkvoice Telephone Status

3608006	KO
3518814	OK
3578712	OK
3576878	OK
3664633	OK
3664944	KO
3576822	OK
3664945	KO
3518818	OK
3576873	OK
3533011	OK
3578861	OK
3578843	OK
3518974	OK
3664672	KO

Figure 4.4: Execution results for the Check Alcatel SILK Phone Status script.

If the script is executed a second time with the same identifier, the script queries the backend database to get the first execution dictionary and compares both dictionaries, generating a new dictionary with the comparison results. Then the comparison dictionary is passed to the view and displayed as a table showing if any phone has changed status, highlighting the change.

4.4 Bandwidth Restriction

4.4.1 Inputs

- The IPs of the Aruba Virtual Controllers where the bandwidth restriction configuration commands will be sent
- The downstream and upstream max allowed speed in Mb/s
- The credentials to log into the Virtual Controller

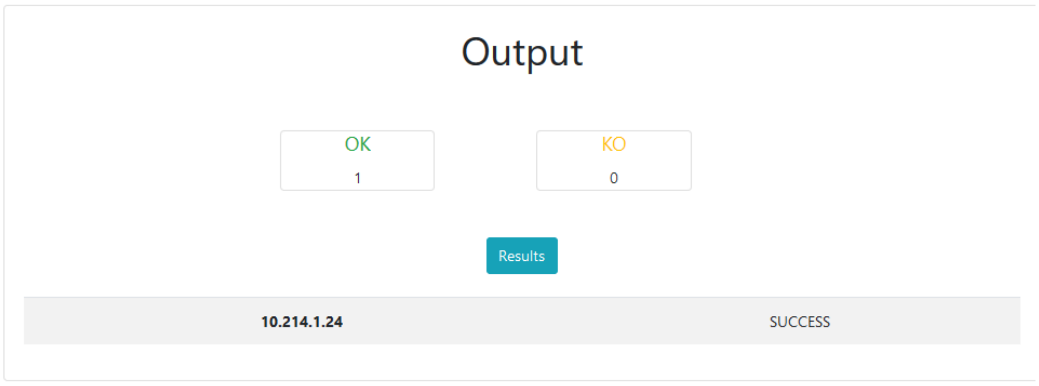
4.4.2 Outputs

- A dictionary containing for each IP the status of the configuration. The configuration status can be either "SUCCESS" or "FAIL"

The LAN & WiFi engineering team needed to apply a bandwidth restriction configuration to a lot of Aruba Virtual Controllers (VCs) [17] for a project with imminent deadline. Since the change had to be done outside of normal working hours because the VCs are in a production environment and the bash scripts that automate configuration deployment that the LAN & WiFi engineering team has do not work for Aruba Virtual Controllers, the necessity of a custom-tailored script to massively configure the VCs arose.

The script logs into the VCs via SSH using the pexpect Python module with the given credentials. The script has into consideration the multiple possible prompts of those devices. The pexpect module much like Expect, waits for the prompt to be ready by the destination device in order to send the next command. By doing it this way, pexpect ensures that the command can be processed by the destination device.

Once logged in, the script launches the respective configuration commands. If any error occurs mid execution in a certain IP, the code saves a "KO" for that IP inside the results dictionary denoting that most likely the configuration was not applied. Otherwise, an "OK" is saved for that IP inside the results dictionary. For each "OK" and "KO" the respective counter is increased. The results dictionary also contains the number of devices that are "OK" i.e. correctly configured and the number of devices that are "KO" i.e. most likely not configured. The result dictionary is passed to the corresponding view and formatted to a table in the HTML itself. (figure 4.5)



10.214.1.24 SUCCESS

Figure 4.5: Execution results for the Bandwidth Restriction script.

4.5 Show Interface Status & Free Ports

4.5.1 Inputs

- The IP of the CISCO switch
- The credentials needed to log into it

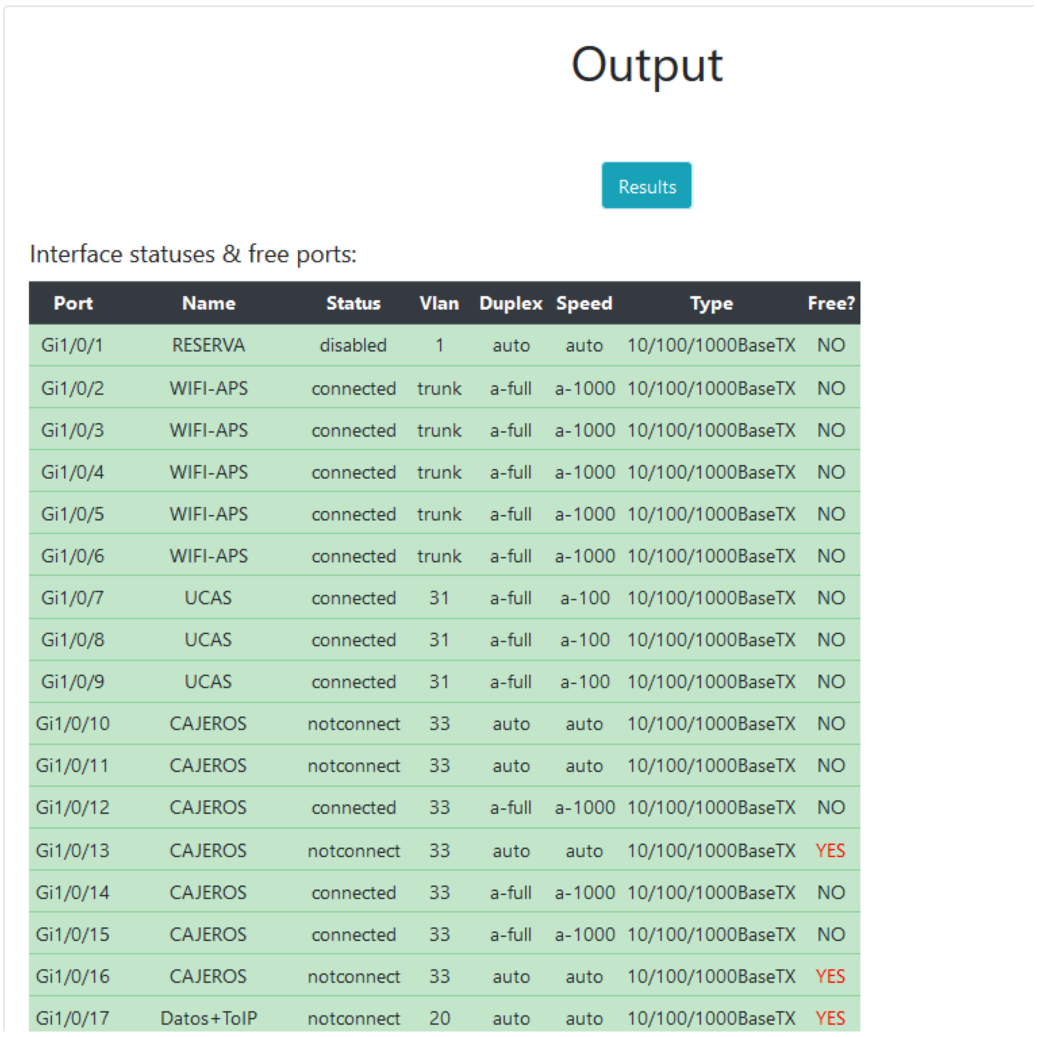
4.5.2 Outputs

- A dictionary containing for each port if its free and all its information.

The necessity of this script developed from the fact that when a technician goes to a switch and wants to connect to a certain port, he/she needs to know if that port is free momentarily or it has not been used in a long time. If it's just free momentarily and they plug in, problems with clients might arise. To solve this problem, a script was developed that besides showing the information of all the ports (which is useful to determine in which port between the free ports the technician should connect), it also shows and highlights the free ports.

To determine if a port has been free for a long time, the script connects via SSH using the *pexpect* module and then sends the “show interface status” command to gather all the port information and the port names. The “show interface status command” does not show the ports that are free, in order to do that, using the obtained port names, for each port the command “show interface [port_name]” command is sent. This command outputs certain text information that can be used in conjunction with some logic to determine if the port has been free for a long time. The established criteria in the script considers that if the command returns “last input never” and “last output never” then the port can be considered free for a long time.

Once determined, the value “YES” or “NO” is appended to a list that contains all the other information about that particular port. A list is created and appended for each port. A list of lists is generated and put inside a dictionary to pass that data to the corresponding view. Inside the HTML the list of lists is iterated to fill a table with all the information (Figure 4.6).



Interface statuses & free ports:

Port	Name	Status	Vlan	Duplex	Speed	Type	Free?
Gi1/0/1	RESERVA	disabled	1	auto	auto	10/100/1000BaseTX	NO
Gi1/0/2	WIFI-APS	connected	trunk	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/3	WIFI-APS	connected	trunk	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/4	WIFI-APS	connected	trunk	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/5	WIFI-APS	connected	trunk	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/6	WIFI-APS	connected	trunk	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/7	UCAS	connected	31	a-full	a-100	10/100/1000BaseTX	NO
Gi1/0/8	UCAS	connected	31	a-full	a-100	10/100/1000BaseTX	NO
Gi1/0/9	UCAS	connected	31	a-full	a-100	10/100/1000BaseTX	NO
Gi1/0/10	CAJEROS	notconnect	33	auto	auto	10/100/1000BaseTX	NO
Gi1/0/11	CAJEROS	notconnect	33	auto	auto	10/100/1000BaseTX	NO
Gi1/0/12	CAJEROS	connected	33	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/13	CAJEROS	notconnect	33	auto	auto	10/100/1000BaseTX	YES
Gi1/0/14	CAJEROS	connected	33	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/15	CAJEROS	connected	33	a-full	a-1000	10/100/1000BaseTX	NO
Gi1/0/16	CAJEROS	notconnect	33	auto	auto	10/100/1000BaseTX	YES
Gi1/0/17	Datos+ToIP	notconnect	20	auto	auto	10/100/1000BaseTX	YES

Figure 4.6: Execution results for the Show Interface Status & Free Ports script.

4.6 Check Bypass

4.6.1 Inputs

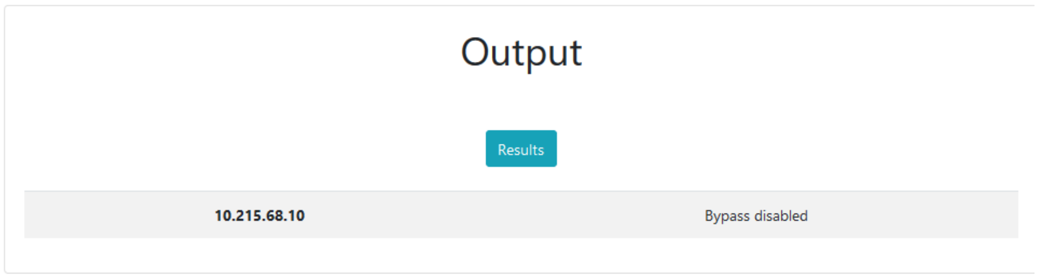
- The list of IPs of the Wireless LAN Controllers (WLCs)
- The credentials to log into these devices

4.6.2 Outputs

- A dictionary specifying for each WLC its bypass status which can be "Bypass enabled", "Bypass disabled" or "Connection Failed"

The LAN & WiFi engineering team found that some Wireless LAN Controllers (WLCs) [18] had the "enable bypass" option activated, that means that anyone that could login into the WLC could bypass the password prompt needed to access the "enable" mode and directly enter the "enable" mode which gives full privileges. This is obviously a security hole and they wanted to patch it, but they didn't know which of the WLCs had it activated and which not. Checking manually was an option but it was not a very efficient one since there are a lot of WLCs in the network, so they suggested the development of a script to automatically check and report all the devices that had the "enable bypass" option activated.

The script sequentially logs via SSH into all the WLCs using pexpect and sends the command "show run | i enable" to determine if the bypass is active, then it updates a dictionary with the status of the bypass for that. The dictionary is either filled with "Bypass enabled", "Bypass disabled" or "Connection Failed". Once all the IPs have been covered, the dictionary is sent to the view and the result shown in the frontend (Figure 4.7).



10.215.68.10	Bypass disabled

Figure 4.7: Execution results for the Check Bypass script.

4.7 Change Bypass

4.7.1 Inputs

- The list of IPs of the Wireless LAN Controllers (WLCs)
- The credentials to log into these devices

4.7.2 Outputs

- A dictionary with as many keys as Wireless LAN Controllers, the key names are the IPs of the WLCs, and the values of those keys are either “Bypass disabled successfully” or “Not possible to change bypass”

Once the engineers knew with the “Check Bypass” script which WLCs had the “enable bypass” option, what was left was to remove that option. A script was created for that purpose.

The script was tailored to log into the WLCs of Aruba, detect the specific prompts and input the commands needed to remove the “enable bypass” option. If all the commands are sent and no exception occurs in the code execution, then the script considers that the configuration commands were applied, and the results dictionary is updated for that IP with the value “Bypass disabled successfully”. If the script is not able to connect, or some timeout occurs midway execution, or the destination terminal outputs some parse error or an exception is thrown, the dictionary is updated for that IP with “Not possible to change bypass”. Once all the IPs have been covered, the dictionary is sent to the view and the result shown in the frontend (Figure 4.8)

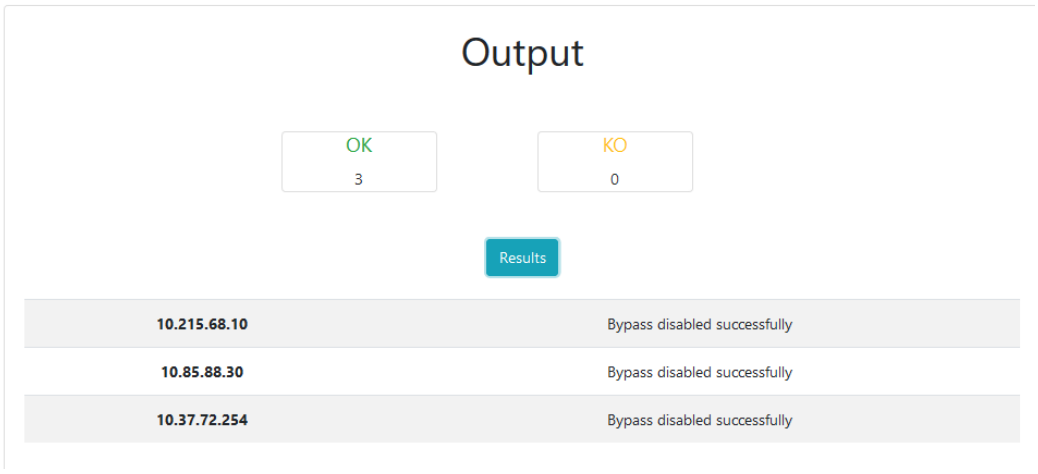
4.8 Virtual Controller Configuration

4.8.1 Inputs

- The IP of the VC and a list of input parameters that modify the configuration template that is going to be applied
- The credentials to log into the device

4.8.2 Outputs

- A dictionary with only one key/value pair. The key name is “status” and the key value can be either “Master not found in the IP range”,


Output

OK	KO
3	0

Results

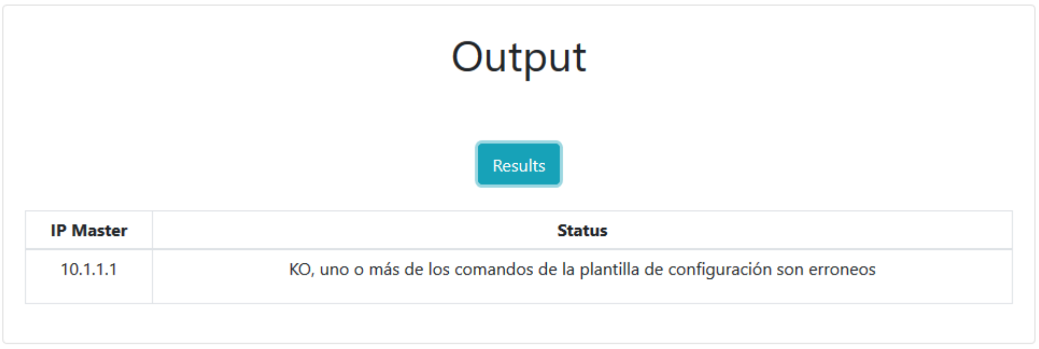
10.215.68.10	Bypass disabled successfully
10.85.88.30	Bypass disabled successfully
10.37.72.254	Bypass disabled successfully

Figure 4.8: Execution results for the Change Bypass script.

“KO, one or more commands were not successfully applied” or “OK, configuration applied successfully”

One common task in the LAN & WiFi engineering team is to configure new Aruba Virtual Controllers. The manual process is quite slow because they must find which IP was given inside of the IP range they had specified, and then manually input the configuration commands which are not always the same. The VC configuration depends on input parameters.

The script first searches for the IP where the master Access Point is located. It tries to sequentially log into the IPs inside the IP range using the pexpect module and if there is no timeout then it sends the command “show election statistics”. If the master AP is not found inside the IP range then the dictionary is updated with “Master not found in the IP range”. If “master” is in the response of the command, then the master AP can be found in that IP. The script then starts sending the configuration commands. If there is no timeout and all the commands are correctly sent and none of them return the response “% Parse error” then the dictionary is updated with the value “OK, configuration applied successfully”. If an error occurs midway execution like a timeout or error response, then the dictionary is updated with “KO, one or more commands were not successfully applied”. Once the dictionary is filled, it is returned to the view and displayed in a table in the frontend (Figure 4.9).



IP Master	Status
10.1.1.1	KO, uno o más de los comandos de la plantilla de configuración son erroneos

Figure 4.9: Execution results for the Virtual Controller Configuration script when one or more of the commands in the given template is incorrect and triggers a parse error in the VC.

4.9 Meraki Menu

The Meraki Menu consists of multiple different scripts which can be called from the frontend of the portal that interact with the CISCO Meraki Cloud using the "Dashboard API" of Meraki. Meraki is a network management platform available in the cloud [19]. It is property of CISCO. The necessity of developing the Meraki menu was proposed for three different reasons:

- Even though all operations can be done in the online Meraki Dashboard, if write permissions are given to an individual, that individual can perform ALL write operations and that's dangerous in terms of security.
- Some operations need to be done massively and doing them manually in the online Meraki Dashboard is extremely slow, specially because the pages render very slowly.
- Gathering certain information is only possible by querying multiple times the API and processing the obtained data. The online Meraki Dashboard does not directly give some information that the company considers useful.

Before the development of the scripts, an extensive investigation was made on how the API works and which were the appropriate calls to obtain the desired results for each script. The API is RESTful and uses API keys as authentication. The API key value needs to be put inside a "X-Cisco-Meraki-API-Key" request header to authenticate the calls. Each API key is linked to a user and has access to the organizations that user has access. The requests Python module was used to create all the requests needed in the scripts.

The Meraki Menu consists of 7 different scripts:

- Get the radiated SSIDs in a network

  This script returns a list with the SSIDs that are being currently radiated in a certain network. The input of the script is the name of the network in Meraki.

  To retrieve the desired information, two HTTP requests need to be sent to the API. One is a HTTP GET to:

  `https://dashboard.meraki.com/api/v0/organizations/[meraki_org]/networks`

  This call retrieves all the networks in the organization. Then searching the name of the network given in the input form in all of the returned records, the ID for that network is obtained.

  With that ID then the call to get the radiated SSIDs is done. The base URL for that call is:

  `https://dashboard.meraki.com/api/v0/networks/[network_id]/ssids`

  The returning data is formatted, stored in a dictionary and presented in the frontend.

- Get the clients information in a network

  This script returns the information of all the clients in a network. IP, MAC, device information and network usage data is retrieved among other information from this script. The name of the network in Meraki is needed as input.

  Three calls are needed to retrieve this information. First, by doing the same procedure as the one stated in the SSIDs script, the ID of the network is retrieved. After getting the ID, a call to get the list of serial numbers is sent. The base URL for that call is:

  `https://dashboard.meraki.com/api/v0/networks/[network_id]/devices`

  Then for each serial number, a call needs to be made to get the clients of that device and its information. The base URL for that call is:

`https://dashboard.meraki.com/api/v0/devices/[serial_number]/clients?timespan=180`

The timespan value is in seconds and ensures that the clients returned by the call were seen in the time interval between now and 180 seconds ago. All the clients information is stored in a dictionary and directly sent to the frontend.

- Unbind a device from a network

  This script allows the unbinding of a device from a network. There are two main use cases for this script, one is when the device stops working and needs to be removed or when the device wants to be moved to another network. To perform this operation, the serial number of the device is needed.

  Only two calls are needed to perform this operation. The call to get the network ID and a POST call to:

  `https://dashboard.meraki.com/api/v0/networks/[network_id]/devices/[serial_number]/remove`

  Depending on the response of this last call, the result of the operation is notified in the frontend.

- Bind a device to a network

  This script allows the binding of a device to a network and then binds it to a switch profile if the device is a switch. In Meraki, the switches need a switch profile to retrieve the specific configuration of each of its ports. To perform these two operations, the serial number of the device and the ID of the switch profile are needed as inputs.

  For a switch, the script sends three calls, the one to get the network ID, a POST call to:

  `https://dashboard.meraki.com/api/v0/networks/[network_id]/devices/claim`

  And another POST call to:

  `https://dashboard.meraki.com/api/v0/networks/[network_id]/devices/[serial_number]`

  For the claim call, the serial number needs to be sent in the body of the call and for the last call, switch profile ID needs to be in the body of the call to bind the switch to a specific switch profile.

- Create networks

  This script allows to massively create networks in Meraki, making the creation process a lot faster than doing it manually in the online Meraki Dashboard. A list of network names is needed as input. The following POST call is the one used to create a network:

  `https://dashboard.meraki.com/api/v0/organizations/[org_name]/networks`

  The organization name must be specified in the URL and some parameters must be specified in the body of the call, like the name of the network and the network type. A dictionary is updated for each network specifying if the network was created successfully or not depending on the response of each POST call.

- Reset the port of a switch

  This script allows the reset of a specific port in a Meraki switch. The serial number of the switch and the port number are needed as inputs. Two POST calls are needed to perform the reset, one to bring down the port and another to bring it back up. The two calls use the same URL which is:

  `https://dashboard.meraki.com/api/v0/devices/[serial_number]/switchPorts/[port_number]`

  In one call, the body has the enabled parameter as “False” and in the other the body has the enabled parameter as “True”. Depending on the response of these two calls, a dictionary is updated indicating the result of the reset operation and send to the frontend to notify the user.

- Get the number of tunnels created in an organization

  The Meraki MX devices which act as firewalls in Meraki have a limited maximum number of tunnels that can be active at the same time. If the connections are configured to be tunneled then for each AP and SSID, a tunnel is created. This script sends the minimum number of calls needed to count the total number of tunnels active in the present moment. This is useful information because when the total number of active concurrent tunnels is near the MX tunnel limit, then actions must be taken.

  The script first sends a GET call to get all the network:

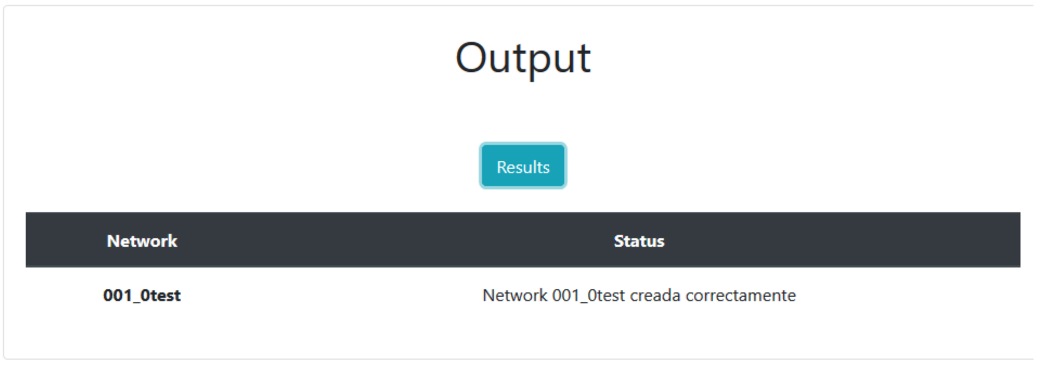
  `https://dashboard.meraki.com/api/v0/organizations/[org_id]/networks`

For each network, if the network has a template with SSIDs that create a tunnel, then the ID of the network is added in a list. Then the networks in this list are queried with the GET call:

`https://dashboard.meraki.com/api/v0/networks/[serial_number]/devices`

From this call the number of APs for each network is extracted. Since each AP creates a tunnel towards the MX, each AP increments a counter. At the end of the script execution, the value of this counter is retrieved and sent to the frontend.

An execution for the create networks script can be seen in Figure 4.10



Network	Status
001_0test	Network 001_0test creada correctamente

Figure 4.10: Execution results for the Meraki create networks script.

4.10 ACL Config

4.10.1 Inputs

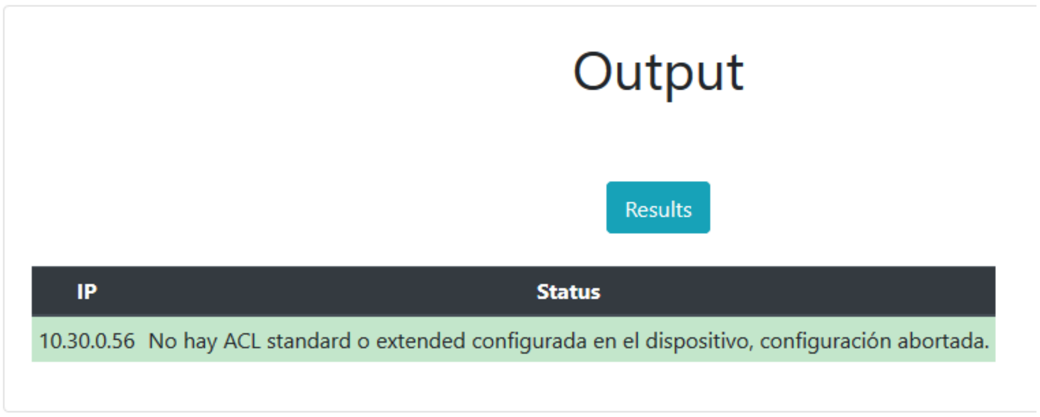
- The list of IPs of the switches where the configuration will be applied
- The list of IPs to add in the ACL of the device and the credentials to log into the devices

4.10.2 Outputs

- A dictionary that contains for each IP the status of the configuration which can be “Could not connect to the device”, “Timeout mid execution of the commands” or “Configuration applied correctly”

An update of the Access Control List (ACL) of a lot of CISCO switches was needed. The bash script to configure CISCO switches that the LAN & WiFi engineering team had was not good enough for this task because the configuration that needed to be deployed was dependent on the properties of the ACL of each switch, so the creation of a new script was needed.

The script logs into sequentially into all of the CISCO switches and for each of them sends the command "show run | i ACCES-SSH". The response to this command determines if the ACL is of type standard or extended. Once the type is determined, the appropriate configuration commands are sent. If there is no error mid execution and there are no timeouts then a dictionary updated with status "Configuration applied correctly", otherwise depending on what happened, the dictionary is updated with the appropriate status. Once the script has executed in all of the IPs, the completely updated dictionary is returned and shown in the frontend in a table, reporting how the configuration went for each switch (Fig. 4.11)



IP	Status
10.30.0.56	No hay ACL standard o extended configurada en el dispositivo, configuración abortada.

Figure 4.11: Execution results for the ACL config script when there is no ACL configured in the switch.

4.11 Backup of CISCO switches

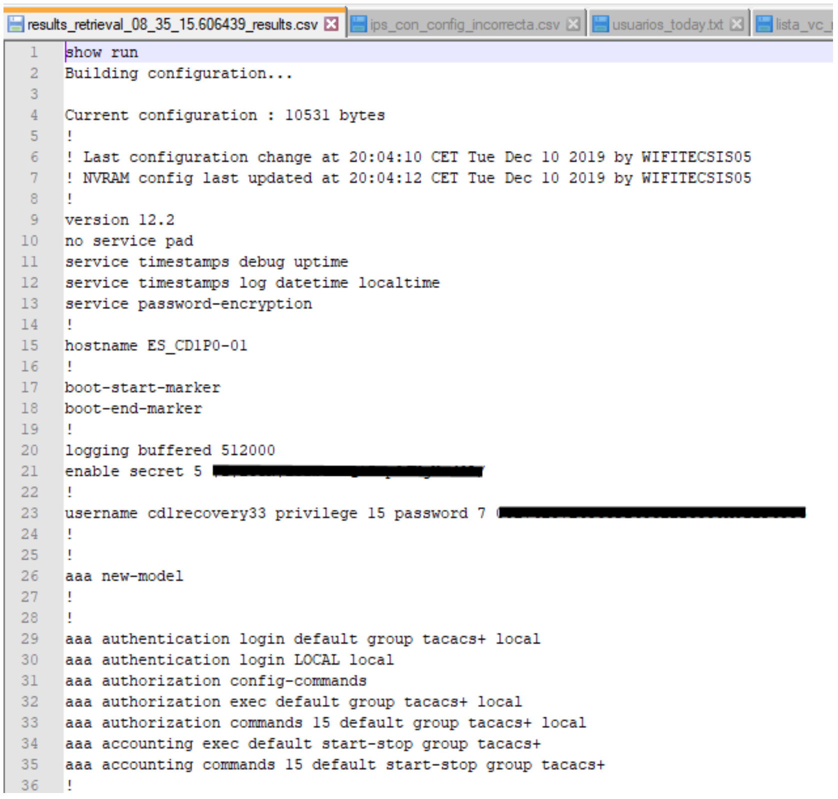
4.11.1 Inputs

- The hostname of the device which can be selected from a dropdown that has all the devices

4.11.2 Outputs

- A button that allows the export of the configuration of the device

To easily and quickly retrieve the backup of any CISCO switch, two scripts were developed. One of the scripts runs in a Command Run On (CRON) and is the script that logs into all of the CISCO switches which are specified in a .txt file. This script logs into the devices using the pexpect module, sends some show commands including “show run”, appends the results of those commands in a list and writes the results in a .txt file. The script upon its daily execution, checks if there is any change in the configuration with respect to the previous execution and if so, overwrites the file. The overwrite for a switch is not made if the script cannot connect or an exception rises midway execution. The other script, the one that runs when ordered via the web portal, reads the content of the specific configuration .txt file and allows the user to download the file. An example of an exported backup file is shown in Figure 4.12.


results_retrieval_08_35_15.606439_results.csv
ips_con_config_incorrecta.csv
usuarios_today.txt
lista_vc_

```
show run
Building configuration...

Current configuration : 10531 bytes
!
! Last configuration change at 20:04:10 CET Tue Dec 10 2019 by WIFITECSIS05
! NVRAM config last updated at 20:04:12 CET Tue Dec 10 2019 by WIFITECSIS05
!
version 12.2
no service pad
service timestamps debug uptime
service timestamps log datetime localtime
service password-encryption
!
hostname ES_CD1P0-01
!
boot-start-marker
boot-end-marker
!
logging buffered 512000
enable secret 5 [redacted]
!
username cdlrecovery33 privilege 15 password 7 [redacted]
!
!
aaa new-model
!
!
aaa authentication login default group tacacs+ local
aaa authentication login LOCAL local
aaa authorization config-commands
aaa authorization exec default group tacacs+ local
aaa authorization commands 15 default group tacacs+ local
aaa accounting exec default start-stop group tacacs+
aaa accounting commands 15 default start-stop group tacacs+
!
```

Figure 4.12: An example of an exported backup generated by the backup script.

4.12 Check CUCM server

4.12.1 Inputs

- Proxy credentials to be able to access the server were the Cisco Unified Communications Manager is deployed
- The credentials to query the CUCM API
- An identifier to compare results with previous executions

4.12.2 Outputs

- A dictionary that contains for each phone if its status is "Registered" or "Unregistered". If the execution uses an identifier previously used in another execution then the results are compared, and the status can either be "Changed" or "Unchanged"

Similar to the scenario with the "Check Session Border Controller SIP Agents State & Statistics" script, the voice & media engineering team wanted with this script to quickly know after an operation that changes firewall settings in the network, if any CISCO phone had changed state.

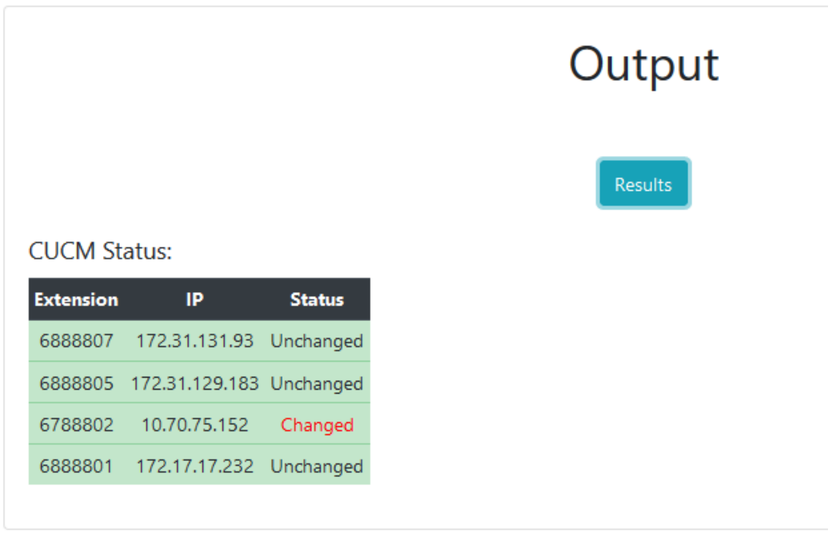
The script queries the Cisco Unified Communications Manager API which is a SOAP API. The CUCM is a Call Manager that provides reliable, secure, scalable, and manageable call control and session management [20]. An HTTP POST request with a well formatted XML in the body of the request is needed to retrieve the desired information. The requests Python module was used to create this request. Proxy configuration code was needed to contact the server. The SOAP API uses Basic Auth in conjunction with the CUCM credentials.

If the script is executed with the same identifier as the first execution, the results of the database are compared with the results of the second execution and the dictionary is updated with the comparison results. The comparison dictionary has for each phone one of two statuses, "Changed" or "Unchanged". This can be seen in Figure 4.13.

4.13 Check VCS phone status

4.13.1 Inputs

- A list of IPs where the CISCO Video Communication Servers (VCS) are located



CUCM Status:

Extension	IP	Status
6888807	172.31.131.93	Unchanged
6888805	172.31.129.183	Unchanged
6788802	10.70.75.152	Changed
6888801	172.17.17.232	Unchanged

Figure 4.13: Comparison of the phone statuses done by the CUCM script.

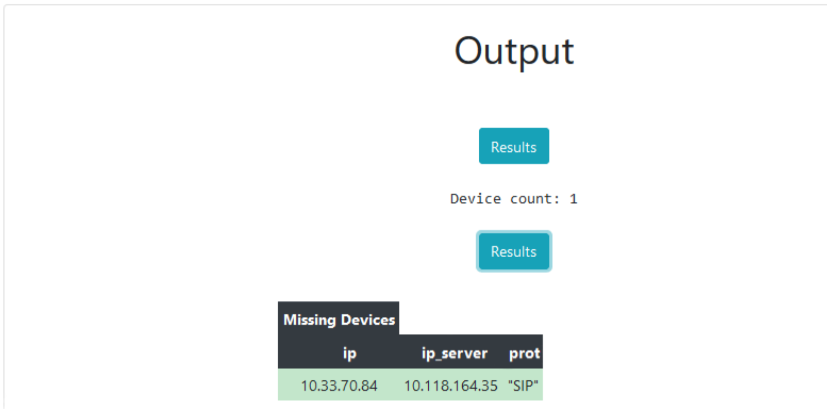
- The credentials to log into the servers
- An identifier to compare results with previous executions

4.13.2 Outputs

- A dictionary containing the CISCO phones found in the VCS and all the information of each phone. This information is the IP, the IP of the server where it was found, the protocol that uses (SIP or H323) and the telephone number.

Like previous scripts, this script allows to compare the results between two executions with the same identifier. The purpose of this script is to see after firewall changes, if any CISCO phone is missing from the Video Communication Servers (VCS). A VCS performs the functions of a Call Manager like the CUCM but for telepresence devices [21].

The script sequentially logs into the VCS's with the pexpect module and sends the “xstatus registrations” command to every single one of them. The response of the command is a string and is concatenated for each VCS. After that, parsing is done with some regular expressions and the information of all the found devices is stored in a dictionary. If it's the first execution for the given identifier, the dictionary is directly sent to the frontend and displayed in a table. Otherwise the second execution dictionary is compared with the first one which is stored in database. The comparison dictionary contains the phones that are missing in the second execution and its details. An execution example showing the missing devices and its details is shown in Figure 4.14.


Output

Results

```
Device count: 1
```

Results

Missing Devices		
ip	ip_server	prot
10.33.70.84	10.118.164.35	"SIP"

Figure 4.14: Comparison showing the missing phones in each VCS server.

4.14 Check OCCAS status & EOM status

4.14.1 Inputs

- The credentials to authenticate in the proxy that connects to the OCCAS & EOM servers
- The API key of the EOM server

4.14.2 Outputs

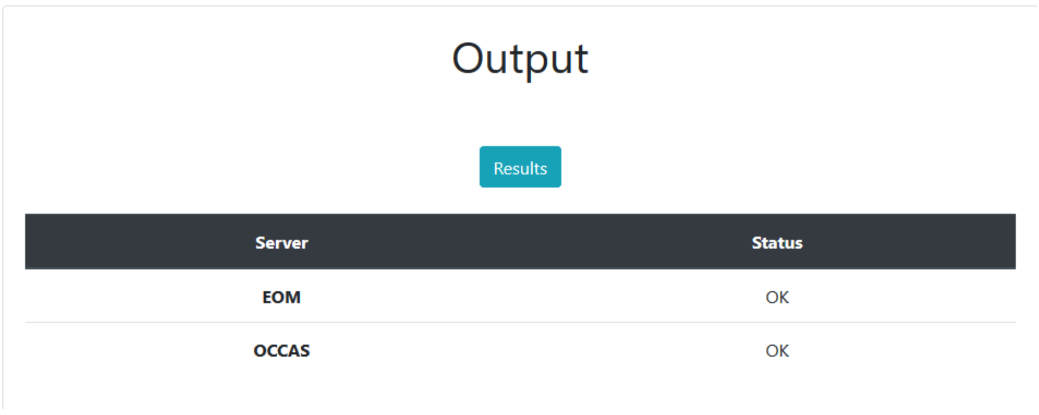
- A dictionary showing the status of the OCCAS server and the EOM server.

The purpose of this script is to quickly determine the status of two critical servers, the Oracle Communications Converged Application Server (OCCAS) and the Oracle Enterprise Operations Monitor (EOM). This task is done manually by the voice & media engineering team every single day and automating the process saves some time in the long run.

The script first sends a SOAP call to the OCCAS API. The body of the SOAP call is filled with an XML that orders the OCCAS server to initiate a call from one phone to another phone. Depending on the response of the SOAP call, the status of the OCCAS server can be determined and the dictionary with the results is properly updated.

If the call between the two phones is done, the script waits a fixed amount of time to ensure that the call is registered in the EOM server. Then to check if the EOM is working properly, a call to the RESTful API of the EOM is sent. The API demands the API key sent as a header for authorization purposes. The body of the call contains some filters, specifically the source and destination phone numbers and a starting timestamp. If the response to this API call with the filter returns a record, the status of the EOM server can be considered OK.

If the OCCAS server did not trigger the call between the two phones then the status of the EOM server is set as "TBD" as it could not be determined. Finally, the dictionary is updated with the results and sent to the frontend in which is displayed as a table (Figure 4.15)



Server	Status
EOM	OK
OCCAS	OK

Figure 4.15: Execution results for the Check OCCAS status & EOM status script.

4.15 Wireless Client Isolation

4.15.1 Inputs

- A list containing Virtual Controllers or Wireless LAN Controllers
- The credentials to log into them
- A textbox to specify whether the list contains VCs or WLCs

4.15.2 Outputs

- A list of dictionaries. Each dictionary contains the IP of the VC or WLC and the status which can be "Success" or "Fail".

For security reasons the deployment of wireless client isolation configuration among the Aruba Virtual Controllers and Wireless LAN Controllers was needed for an urgent project and since this configuration was not applied in the initial configuration, a script was developed to quickly configure them. Wireless client isolation configuration ensures that other users cannot see or send traffic between them, i.e. they are isolated between them.

The script first checks whether the IPs inside the list of IPs correspond to VCs or WLCs. The configuration commands differ in each scenario. Then it logs into every single IP in the list and sends a command to retrieve the list of radiated SSIDs. If it's a VC, the sent command is "show summary", if it's a WLC, it is "show profile-list wlan virtual-ap". The list of SSIDs is needed since the wireless client isolation configuration needs to be applied at global level and also at SSID level.

After retrieving the list, the appropriate configuration commands are sent. Finally, the dictionary for that device is updated with the resulting configuration status, if the connection was possible and no timeouts occurred then the status becomes "Success", otherwise its "Fail". After configuring all the devices, the dictionary with the status of all devices is sent to the frontend and displayed in a table. In Figure 4.16 an example showing the commands sent for the configuration of a VC can be seen.

4.16 Techspec of CISCO switches

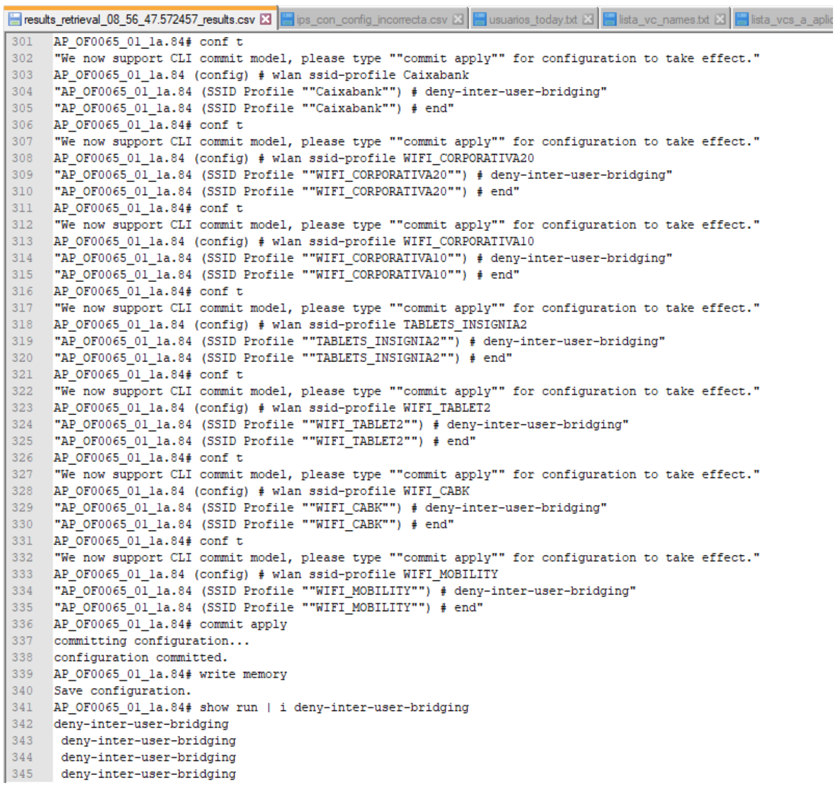
4.16.1 Inputs

- A list of IPs of CISCO switches
- The credentials to log into them

4.16.2 Outputs

- A dictionary with as many keys as IPs in the list. The value of each key is a string detailing the security issues found in the configuration of the device.

The company specifies some security minimums that the configuration of all the CISCO switches must satisfy. For example, any local user with privilege 15 must have a MD5 encrypted password. Since the configuration of the CISCO switches is not standardized and thus not homogeneous among all devices, some devices do not comply with the security standards. The necessity of developing this script came from the LAN & WiFi engineering


results_retrieval_08_56_47.572457_results.csv
ips_con_config_incorrecta.csv
usuarios_today.txt
lista_vc_names.txt
lista_vcs_a_aplic

```
AP_OF0065_01_la.84# conf t
"We now support CLI commit model, please type ""commit apply"" for configuration to take effect."
AP_OF0065_01_la.84 (config) # wlan ssid-profile Caixabank
"AP_OF0065_01_la.84 (SSID Profile ""Caixabank"") # deny-inter-user-bridging"
"AP_OF0065_01_la.84 (SSID Profile ""Caixabank"") # end"
AP_OF0065_01_la.84# conf t
"We now support CLI commit model, please type ""commit apply"" for configuration to take effect."
AP_OF0065_01_la.84 (config) # wlan ssid-profile WIFI_CORPORATIVA20
"AP_OF0065_01_la.84 (SSID Profile ""WIFI_CORPORATIVA20"") # deny-inter-user-bridging"
"AP_OF0065_01_la.84 (SSID Profile ""WIFI_CORPORATIVA20"") # end"
AP_OF0065_01_la.84# conf t
"We now support CLI commit model, please type ""commit apply"" for configuration to take effect."
AP_OF0065_01_la.84 (config) # wlan ssid-profile WIFI_CORPORATIVA10
"AP_OF0065_01_la.84 (SSID Profile ""WIFI_CORPORATIVA10"") # deny-inter-user-bridging"
"AP_OF0065_01_la.84 (SSID Profile ""WIFI_CORPORATIVA10"") # end"
AP_OF0065_01_la.84# conf t
"We now support CLI commit model, please type ""commit apply"" for configuration to take effect."
AP_OF0065_01_la.84 (config) # wlan ssid-profile TABLETS_INSIGNIA2
"AP_OF0065_01_la.84 (SSID Profile ""TABLETS_INSIGNIA2"") # deny-inter-user-bridging"
"AP_OF0065_01_la.84 (SSID Profile ""TABLETS_INSIGNIA2"") # end"
AP_OF0065_01_la.84# conf t
"We now support CLI commit model, please type ""commit apply"" for configuration to take effect."
AP_OF0065_01_la.84 (config) # wlan ssid-profile WIFI_TABLET2
"AP_OF0065_01_la.84 (SSID Profile ""WIFI_TABLET2"") # deny-inter-user-bridging"
"AP_OF0065_01_la.84 (SSID Profile ""WIFI_TABLET2"") # end"
AP_OF0065_01_la.84# conf t
"We now support CLI commit model, please type ""commit apply"" for configuration to take effect."
AP_OF0065_01_la.84 (config) # wlan ssid-profile WIFI_CABK
"AP_OF0065_01_la.84 (SSID Profile ""WIFI_CABK"") # deny-inter-user-bridging"
"AP_OF0065_01_la.84 (SSID Profile ""WIFI_CABK"") # end"
AP_OF0065_01_la.84# conf t
"We now support CLI commit model, please type ""commit apply"" for configuration to take effect."
AP_OF0065_01_la.84 (config) # wlan ssid-profile WIFI_MOBILITY
"AP_OF0065_01_la.84 (SSID Profile ""WIFI_MOBILITY"") # deny-inter-user-bridging"
"AP_OF0065_01_la.84 (SSID Profile ""WIFI_MOBILITY"") # end"
AP_OF0065_01_la.84# commit apply
committing configuration...
configuration committed.
AP_OF0065_01_la.84# write memory
Save configuration.
AP_OF0065_01_la.84# show run | i deny-inter-user-bridging
deny-inter-user-bridging
 deny-inter-user-bridging
 deny-inter-user-bridging
 deny-inter-user-bridging
```

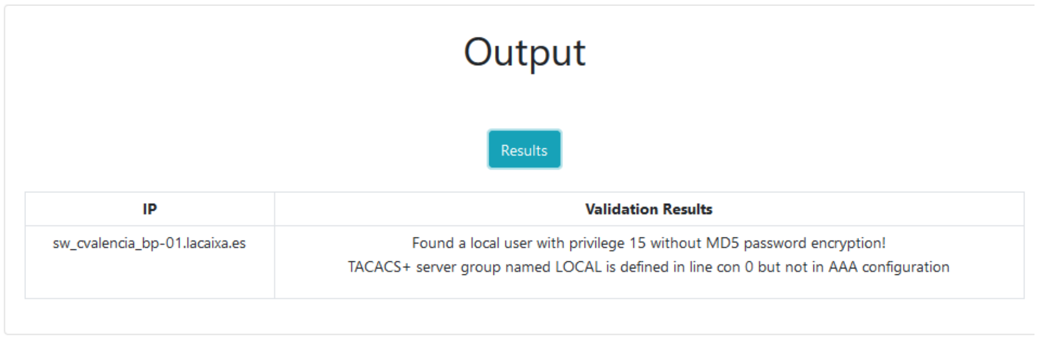
Figure 4.16: Wireless Client Isolation configuration commands for a VC

team. They wanted to know which CISCO switches do not comply and why they don't. The script sequentially logs into the CISCO switches and sends the "show run" command. The response is then processed through some logic and regular expressions to determine the security issues found in the configuration. The script checks that:

- The AAA configuration is correct
- The local users with privilege 15 and privilege 0 have a MD5 encrypted password with a minimum length.
- If the configured SNMP is not SNMPv3, then the SNMP community string must include numbers, letters, special characters and should be 14 or more characters long.
- Telnet is disabled
- All line configurations use password and are related to the AAA configuration.

- An Access Control List exists for the device and has at least an IP.
- Some form of DDoS protection is enabled.
- A banner indicating the prosecution of anyone that does wrong things to the device is configured.

The dictionary is updated with the security issues found for that switch. After cycling through all switches, the dictionary is returned to the frontend and displayed in a table (Figure 4.17).



IP	Validation Results
sw_cvalencia_bp-01.lacaixa.es	Found a local user with privilege 15 without MD5 password encryption! TACACS+ server group named LOCAL is defined in line con 0 but not in AAA configuration

Figure 4.17: Execution results for the Techspec of CISCO switches script.

4.17 Morning Check LAN & WiFi

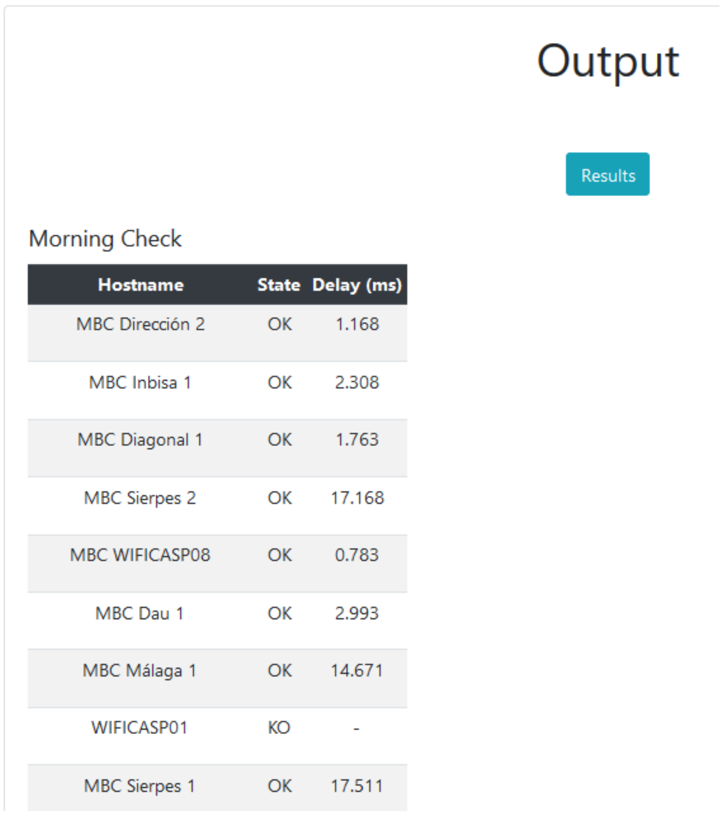
4.17.1 Inputs

- No user inputs are required for this script.

4.17.2 Outputs

- A dictionary that contains for each server/device its status and latency

Every day the LAN & WiFi engineering team checks the status of a list of servers/devices which are critical. They want to know if they are up and their latency. This script was created to automate this status check process. The script sends pings to the servers/devices in the list. If the pings don't timeout then the server/device is responding, and the dictionary is updated with an "OK" for that server/device. The dictionary is also updated with the observed latency in the ping. If there is a timeout, then its updated with a "KO". This is done for all the IPs in the list and once finished, the resultant dictionary is sent to the frontend and displayed in a table (Figure 4.18).



Morning Check

Hostname	State	Delay (ms)
MBC Dirección 2	OK	1.168
MBC Inbisa 1	OK	2.308
MBC Diagonal 1	OK	1.763
MBC Sierpes 2	OK	17.168
MBC WIFICASP08	OK	0.783
MBC Dau 1	OK	2.993
MBC Málaga 1	OK	14.671
WIFICASP01	KO	-
MBC Sierpes 1	OK	17.511

Figure 4.18: Execution results for the Morning Check script.

4.18 Check CISCO devices models & versions

4.18.1 Inputs

- A list of CISCO switches
- The credentials needed to log into them.
- Depending on the script, a specific model or software version must be given to filter the obtained results.

4.18.2 Outputs

- A dictionary with as many keys as IPs of servers/devices. The value of each key is a list of two elements. The first element is the status of the server/device which can be "OK" or "KO". The second element is the average latency shown when pinging the server/device.

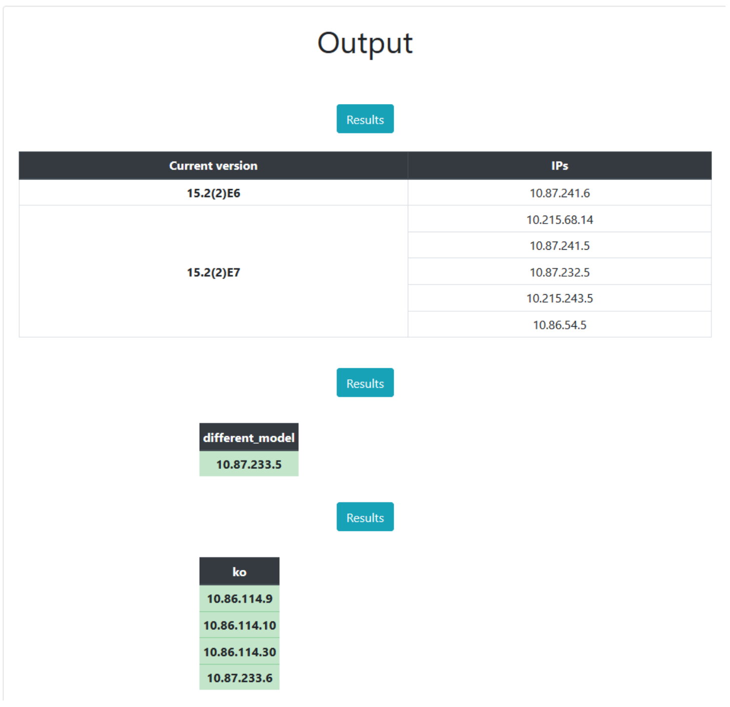
The LAN & WiFi engineering team wanted to know the models and software versions of a big list of CISCO switches to update their inventory and to check which switches are or will be soon out of support. A script was developed to gather the model and software version of a list of CISCO devices and process the information in five different ways to fulfill the needs of the client. The script allows to:

- Retrieve and show the software version of a list of IPs of CISCO switches.
- Retrieve and show the model of a list of IPs of CISCO switches.
- Retrieve the models of switches of a given software version.
- Retrieve the software versions of switches of a given model.
- Show the switches of a given model that have a lower software version than the one given.

The user can choose which results to see depending on the selected view in the frontend. Depending on the view, the user has to input the software version or/and the switch model.

The output of the scripts is dependent on the particular script but its always a dictionary containing either the model, the software version or both for each CISCO switch. The scripts send the "show version" command and parse the response of the command. The parsing is different for each CISCO OS firmware and the logic considers the known response formats to retrieve the version and model from the response string. The obtained model and/or software version updates a dictionary which is send to the frontend to display the data to the user.

An example execution that returns the IPs of the devices that have a different model than the one that the user submitted in the input form, is shown in Figure 4.19



Current version	IPs
15.2(2)E6	10.87.241.6
15.2(2)E7	10.215.68.14
	10.87.241.5
	10.87.232.5
	10.215.243.5
	10.86.54.5

different_model
10.87.233.5

ko
10.86.114.9
10.86.114.10
10.86.114.30
10.87.233.6

Figure 4.19: Retrieving the IPs of the devices that have a different model than the one specified by the user in the input form.

Chapter 5

Conclusions

In this project, the most important results are the scripts themselves that run beneath the automation portal. Those scripts are the big part of this project since without them, the automation portal would be useless. The scripts contain the logic behind the specific configuration of commercial networking devices and are the ones that provide the main utility. Besides the automation scripts themselves, the reports section and the "Architect" are the big functionalities of this project, while not being the base of the portal, they provide a lot of utility. The reports section allows someone to see which scripts are the ones that users execute more, when they were executed and most importantly, the time saved by each script.

By observing the obtained data in the reports section, it is possible to conclude that the development of a web portal that automates configuration processes, results in a positive return of investment for the company when the number of networking devices to configure is big. This is because not only initial configuration is needed, there is also many miss-configurations and many missing configurations in the device park. Previous to the automation portal, the networking team were accessing manually to the devices and copy pasting the configuration by hand, which should not be a thing in the present days.

The "Architect" adapts a template containing a Python script and performs code substitutions to adapt the code to the platform standards. It also creates the script instance in the database and fills it with the appropriate input/output format. With this, the user can quickly import any Python script inside the platform by just knowing how to fill the given template.

Its also important to mention that like the analysis pointed beforehand, Django proved to be a great framework to quickly develop an scalable and secure web application.

Working inside a big company proved difficult to accomplish certain tasks which should be easier in less closed scenarios. There was too many unnecessary bureaucracy and the firewall rules in the inner network made some tasks that at first glance seemed easy, difficult tasks.

The main problems I had with this project appeared while deploying the automation portal to production. The installed SQLite version in the Linux server was old and not compatible with Django, then I wasn't able to download the Python modules from the online PIP repository because of firewall rules and finally the reports page was loading with errors.

To solve the SQLite issue I downgraded the Django version and adapted part of the code to the older version. To install the Python modules I had to manually download and install the modules and their dependencies from PyPI which resulted in experiencing the famous "dependency hell" [22]. Finally, the Pandas Python module had to be downgraded to be compatible with the downgraded Django version.

5.1 Future Work

Due to a time limitation generated by the previously discussed problems, CI/CD pipelines were not possible to implement and should be the first addressed matter in future work. A well configured CI/CD pipeline can provide automatic code testing and code deployment into production. When the code is pushed to the remote repository, it is tested and if everything is fine, it is directly deployed into production.

Bibliography

[1] Investing Answer. ROI - Return on Investment, 2020. https://investinganswers.com/dictionary/r/return-investment-roi.

[2] PyPI. Find, install and publish Python packages with the Python Package Index., 2020. https://pypi.org.

[3] Eric Normand. Why do we use web frameworks?, 2020. https://lispcast.com/why-web-frameworks.

[4] Django. Django - The web framework for perfectionists with deadlines., 2020. https://www.djangoproject.com.

[5] Rapid7. Web Application Vulnerabilities. A look at how attackers target web apps., 2020. https://www.rapid7.com/fundamentals/web-application-vulnerabilities.

[6] John D. Blischak, Emily R. Davenport, and Greg Wilson. A quick introduction to version control with git and github. *PLOS Computational Biology*, 12(1):1–18, 01 2016. doi: 10.1371/journal.pcbi.1004668. URL https://doi.org/10.1371/journal.pcbi.1004668.

[7] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943, 2017. ISSN 2169-3536. doi: 10.1109/access.2017.2685629. URL http://dx.doi.org/10.1109/ACCESS.2017.2685629.

[8] Flask. Flask, web development, one drop at a time., 2020. https://flask.palletsprojects.com/en/1.1.x/#.

[9] Pyramid. Pyramid, The Start Small, Finish Big Stay Finished Framework, 2020. https://trypyramid.com.

[10] Django. Getting started with Django, 2020. https://www.djangoproject.com/start.

[11] Squarespace. What is JSON?, 2020. https://developers.squarespace.com/what-is-json.

[12] SQLite. What Is SQLite?, 2020. https://www.sqlite.org/index.html.

[13] GeeksforGeeks. Python Virtual Environment — Introduction, 2020. https://www.geeksforgeeks.org/python-virtual-environment.

[14] Atlassian. What is Jira used for, 2020. https://www.atlassian.com/es/software/jira/guides/use-cases/what-is-jira-used-for.

[15] CISCO. Cisco Unified IP Phone Services Application Development Notes for Cisco Unified Communications Manager and Multiplatform Phones, 2020. https://www.cisco.com/c/en/us/td/docs/voice_ip_comm/cuipph/all_models/xsi/9-1-1/CUIP_BK_P82B3B16_00_phones-services-application-development-notes/CUIP_BK_P82B3B16_00_phones-services-application-development-notes_chapter_0110.html.

[16] Mitel. What is a Session Border Controller, 2020. https://www.mitel.com/es-es/caracteristicas-y-beneficios/session-border-controller-sbc.

[17] Aruba. Virtual Controller Overview, 2020. https://www.arubanetworks.com/techdocs/Instant_40_Mobile/Advanced/Content/UG_files/virtual_controller/Master_Election_Protocol.htm.

[18] CISCO. What Is a WLAN Controller?, 2020. https://www.cisco.com/c/en/us/products/wireless/wireless-lan-controller/what-is-wlan-controller.html.

[19] Jiri Brejcha. 10 Things You Need To Know About Cisco Meraki, 2020. https://gblogs.cisco.com/uki/10-things-you-need-to-know-about-cisco-meraki.

[20] CISCO. CUCM - Enterprise unified communications and collaboration, 2020. https://www.cisco.com/c/en/us/products/unified-communications/unified-communications-manager-callmanager/index.html.

[21] CISCO. Cisco TelePresence Video Communication Server, 2020. https://www.cisco.com/c/en/us/products/unified-communications/telepresence-video-communication-server-vcs/index.html.

[22] KNEWTON. The Nine Circles of Python Dependency Hell, 2020. https://medium.com/knerd/the-nine-circles-of-python-dependency-hell-481d53e3e025.