



Μεταπτυχιακή Διατρίβη
Πρόγραμμα Μεταπτυχιακών Σπουδών
ΠΜΣ Πληροφορική

Τίτλος Διατριβής	Ανάπτυξη Εφαρμογής για παραμετροποίηση δικτύου με το Django Framework Application Development for Network Configuration with the Django Framework
Ονοματεπώνυμο Φοιτητή	Ιάσονας Σιμώτας
Πατρώνυμο	Παντελής
Αριθμός Μητρώου	ΜΠΠΛ21069
Επιβλέπων	Δουληγέρης Χρήστος, Καθηγητής

Επιβλέπων: Δουληγέρης Χρήστος
Καθηγητής ΠΑΠΕΙ

ΤΡΙΜΕΛΗΣ ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

Εγκρίθηκε από την κάτωθι τριμελή επιτροπή την 1^η Μαΐου 2025.

Όνομα Επώνυμο
Καθηγητής

Όνομα Επώνυμο
Καθηγητής

Όνομα Επώνυμο
Αναπληρωτής Καθηγητής

Ιάσονας Σιμώτας
Πτυχιούχος Μεταπτυχιακού ΠΜΣ Πληροφορικής

Copyright © 'Όνομα Επώνυμο, 2024
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ' ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Πανεπιστήμιου Πειραιά.

1 Περίληψη

Abstract

This thesis focuses on the development of an application for the configuration of network devices, using Django and Python libraries for automation. The application simplifies the configuration process through a user-friendly interface and automated procedures. DevOps principles like CI/CD are applied, using technologies such as Netmiko and GNS3 for testing in real-world conditions. The thesis presents the development steps, technological challenges, and solutions adopted, offering a practical and scalable application.

Περίληψη

Η πυχιακή εργασία αφορά την ανάπτυξη μιας εφαρμογής για την παραμετροποίηση δικτυακών συσκευών, χρησιμοποιώντας το Django και βιβλιοθήκες Python για αυτοματοποίηση. Η εφαρμογή απλοποιεί τη διαδικασία παραμετροποίησης μέσω ενός φιλικού περιβάλλοντος και αυτοματοποιημένων διαδικασιών. Εφαρμόζονται αρχές DevOps όπως CI/CD, με τη χρήση τεχνολογιών όπως το Netmiko και το GNS3 για δοκιμές σε ρεαλιστικές συνθήκες. Παρουσιάζονται τα βήματα ανάπτυξης, οι τεχνολογικές προκλήσεις και οι λύσεις που υιοθετήθηκαν, προσφέροντας μια πρακτική και επεκτάσιμη εφαρμογή.

1.1 Συνοπτική Περιγραφή της Εργασίας

Η παρούσα πτυχιακή εργασία αποσκοπεί στην ανάπτυξη μιας σύγχρονης εφαρμογής για την παραμετροποίηση δικτυακών συσκευών, αξιοποιώντας τις δυνατότητες που προσφέρουν τα σύγχρονα τεχνολογικά εργαλεία, πρότυπα και αρχές DevOps. Η επιλογή του θέματος βασίζεται στη διαρκώς αυξανόμενη ανάγκη για αυτοματοποιημένες, επεκτάσιμες και αποδοτικές λύσεις διαχείρισης, ειδικά σε περιβάλλοντα με υψηλή κλίμακα και πολυπλοκότητα.

Η εφαρμογή αναπτύχθηκε με τη χρήση του Django framework της γλώσσας προγραμματισμού Python, προσφέροντας ένα ολοκληρωμένο backend σύστημα για τη διαχείριση και παραμετροποίηση δικτυακών συσκευών.

Η ανάπτυξη υιοθέτησε τις αρχές του DevOps, διασφαλίζοντας τη συνεχή ενσωμάτωση και παράδοση, καθώς και την αποτελεσματική παρακολούθηση και υποστήριξη της εφαρμογής. Παρόλο που η εργασία δεν περιλαμβάνει τη ρύθμιση εργαλείων όπως το Jenkins ή το GitHub Actions για τη συνεχή παράδοση – καθώς αυτό γίνεται με μη αυτοματοποιημένο τρόπο στη δική μας περίπτωση – οι ενσωματωμένες πρακτικές του DevOps για τη συνεχή ενσωμάτωση επιτρέπουν την αυτοματοποίηση της ροής εργασιών. Επιπλέον, διευκολύνουν την άμεση ενημέρωση της ομάδας για αλλαγές στον κώδικα, ενισχύοντας τη συνεργασία και τη διαφάνεια μέσω της χρήσης του GitHub.

Κατά την υλοποίηση και δοκιμή της εφαρμογής χρησιμοποιήθηκαν εργαλεία όπως το Docker, που εξασφαλίζει φορητότητα και σταθερότητα μέσω της ανάπτυξης και διαχείρισης containers. Παράλληλα η εισαγωγή του κυβερνήτη ως τεχνολογία διαχείρισης κοντέινερς αξιοποιήθηκε για να επιδείξει πώς μια εφαρμογή μπορεί να λειτουργεί αποτελεσματικά σε ένα τέτοιο περιβάλλον, αναδεικνύοντας την επεκτασιμότητα και την ευελιξία της σε πραγματικές συνθήκες.

Η εφαρμογή δοκιμάστηκε σε προσομοιωμένο περιβάλλον GNS3, όπου αναπαραστάθηκαν διαφορετικές συνθήκες δικτύου για την αξιολόγηση της λειτουργικότητας και της απόδοσής της.

Επιπλέον, αξιοποιήθηκαν τεχνολογίες όπως τα RESTful APIs και το πρωτόκολλο SSH για τη διασύνδεση με δικτυακές συσκευές, ενώ η αρχιτεκτονική της εφαρμογής βασίστηκε σε μικρούπηρεσίες για να διασφαλιστεί η ευελιξία και η επεκτασιμότητα.

Η ανάπτυξη περιλάμβανε τη δημιουργία ενός φιλικού προς τον χρήστη περιβάλλοντος για την εισαγωγή και επεξεργασία ρυθμίσεων, καθώς και την ενσωμάτωση εργαλείων για την αυτοματοποιημένη εκτέλεση εντολών. Ιδιαίτερη έμφαση δόθηκε στις τεχνολογίες containerization, οι οποίες επιτρέπουν την εύκολη ανάπτυξη της εφαρμογής σε διαφορετικά περιβάλλοντα.

Συνολικά, η εργασία συνδυάζει πρακτικές αυτοματοποίησης και DevOps με τη χρήση σύγχρονων τεχνολογικών εργαλείων, δημιουργώντας ένα ολοκληρωμένο πλαίσιο διαχείρισης δικτυακών συσκευών. Η εφαρμογή φιλοδοξεί να αποτελέσει ένα πολύτιμο εργαλείο για προγραμματιστές και διαχειριστές δικτύων, συμβάλλοντας σημαντικά στον τομέα της αυτοματοποίησης και της παραμετροποίησης δικτύων.

2 Αναγνώριση και Ευχαριστίες

Η ολοκλήρωση της παρούσας διπλωματικής εργασίας αποτελεί έναν σταθμό που δεν θα ήταν εφικτός χωρίς την υποστήριξη και την ενθάρρυνση ορισμένων ανθρώπων, στους οποίους θα ήθελα να εκφράσω την βαθιά μου ευγνωμοσύνη.

Πρώτα και κύρια, θα ήθελα να ευχαριστήσω την οικογένειά μου, τους φίλους και την κοπέλα μου για τη συνεχή τους υποστήριξη. Η ενθάρρυνση και η καθοδήγησή τους ήταν αστείρευτες πηγές έμπνευσης και δύναμης. Με βοήθησαν να παραμείνω επικεντρωμένος στους στόχους μου, ακόμα και όταν οι δυσκολίες και οι προκλήσεις πολλαπλασιάζονταν. Χάρη σε αυτούς, κατάφερα να διατηρήσω την αισιοδοξία και την αντοχή που απαιτούνταν για να φέρω εις πέρας αυτή την προσπάθεια.

Παρά το γεγονός ότι οι επαγγελματικές μου υποχρεώσεις ήταν συχνά απαιτητικές και πολλές φορές δεν μου άφηναν τον χρόνο που ήθελα για την ενασχόληση με τη διπλωματική μου, κατάφερα να αντλήσω από την εργασιακή μου εμπειρία πολύτιμα εφόδια. Η επαγγελματική μου πορεία ως μηχανικός δικτύωσης και λογισμικού στον τομέα των τηλεπικοινωνιών με βοήθησε να κατανοήσω καλύτερα τις έννοιες και τις τεχνολογίες που μελετήθηκαν. Επιπλέον, αυτή η εμπειρία αποτέλεσε σημαντικό εργαλείο για τη σύνθεση, την ανάλυση και την εμβάθυνση στις τεχνικές πτυχές του έργου.

Η χρονιά που ξεκίνησα το μεταπτυχιακό μου πρόγραμμα, το 2021, συνέπεσε με μια από τις πιο γόνιμες περιόδους της ακαδημαϊκής και επαγγελματικής μου ζωής. Με δύο χρόνια εμπειρίας στον τομέα της μηχανικής δικτύων, είχα ήδη τη βάση για να διευρύνω τις γνώσεις μου και να εξελίξω την αντίληψή μου γύρω από τις τεχνολογικές εξελίξεις στις τηλεπικοινωνίες. Η αγάπη μου για τον κλάδο αυτό αποτέλεσε το κύριο κίνητρο για την απόφαση να συνεχίσω τις σπουδές μου και να ασχοληθώ με την παρούσα εργασία. Αφορμή για την ιδέα αποτέλεσε μία δουλειά ενός μηχανικού η οποία με ενθουσίασε και θέλησα να την πάω ένα βήμα παρακάτω(1).

Μέσα από τη διαδικασία συγγραφής της διπλωματικής, απέκτησα όχι μόνο γνώσεις σε θεωρητικό επίπεδο αλλά και πρακτικές δεξιότητες που εμπλούτισαν την επαγγελματική μου ταυτότητα. Η εμπειρία αυτή συνδύασε την τεχνική μου κατάρτιση με τη θεωρητική ανάλυση, επιτρέποντάς μου να αναπτύξω την ικανότητα να αντιμετωπίζω περίπλοκα προβλήματα με δημιουργική και κριτική σκέψη.

Αναγνωρίζω ότι οι σπιγμές αβεβαιότητας και πίεσης, που πολλές φορές συνδυάζονταν με τις επαγγελματικές απαιτήσεις, υπήρξαν ιδιαίτερα δύσκολες. Ωστόσο, αυτές οι προκλήσεις με δύναξαν την αξία της αποτελεσματικής διαχείρισης χρόνου και της υπομονής. Μέσα από αυτές τις δυσκολίες, έμαθα να προτεραιοποιώ τις υποχρεώσεις μου και να εργάζομαι με συνέπεια.

Δεν μπορώ να παραλείψω την πολύτιμη συμβολή των συναδέλφων και των φίλων μου. Η κατανόηση και η υποστήριξή τους υπήρξαν ανεκτίμητες. Η υπομονή και η ενθάρρυνσή τους, ειδικά σε περιόδους έντονης πίεσης, μου έδωσαν τη δυνατότητα να διατηρήσω την ισορροπία μου και να ολοκληρώσω αυτό το έργο με επιτυχία. Μέσα από αυτή την εμπειρία, συνειδητοποίησα τη σημασία της συνεργασίας και της αλληλούποστήριξης, για τις οποίες τους ευχαριστώ από καρδιάς.

Η εργασία αυτή δεν είναι απλώς μια ακαδημαϊκή ολοκλήρωση αλλά ένα προσωπικό και επαγγελματικό επίτευγμα που ελπίζω να αποτελέσει εφαλτήριο για περαιτέρω ανάπτυξη και συνεισφορά στον χώρο της τεχνολογίας και των τηλεπικοινωνιών.

Περιεχόμενα

1 Περίληψη	1
1.1 Συνοπτική Περιγραφή της Εργασίας	2
2 Αναγνώριση και Ευχαριστίες	3
3 Εισαγωγή	9
3.1 Στόχοι του έργου	9
3.2 Περιγραφή προβλήματος και λύσης	9
3.3 Τεχνολογίες που χρησιμοποιήθηκαν και γιατί	10
3.4 Διάρθρωση της παρούσας εργασίας	11
4 Θεωρητικό υπόβαθρο	13
4.1 Αυτοματοποίηση Διαχείρισης Δικτύου	13
4.2 Web Framework- Django	14
4.3 Βιβλιοθήκες της Python για Network automation	14
4.3.1 Paramiko	14
4.3.2 Netmiko	15
4.3.3 Napalm	15
4.4 Devops και Διαδικασίες Αυτοματισμού	16
4.4.1 Συνεχής Ενσωμάτωση και Παράδοση (CI/CD)	16
4.4.2 Διαχείρηση εκδόσεων (Version Control) με Git και GitHub	16
4.4.3 Containers και Docker	17
4.4.4 Kubernetes και Container Orchestration	17
4.5 Προσομοίωση και Εικονικά Περιβάλλοντα	17
4.5.1 GNS3 και Εικονικά Δίκτυα	17
4.5.2 Πρόγραμμα εικονοποίησης για το GNS3 VM-VirtualBox	18
5 Σχεδίαση και Ανάλυση	21
5.1 Απαιτήσεις Εφαρμογής Διαχείρισης Δικτύου.	21
5.1.1 Λειτουργικές απαιτήσεις	21
5.2 Αρχιτεκτονική της εφαρμογής	21
5.2.1 Μοντέλο MVC (Model-View-Controller)	22
5.2.2 Django MTV	23
5.2.3 Χρήση του MTV στην εφαρμογή	23
5.3 Εικονικό Περιβάλλον Δικτύου GNS3-Testbed	24

5.3.1	Σχεδίαση Τοπολογίας Δικτύου	25
5.3.2	Προσομοίωση Συσκευών Cisco	30
6	Υλοποίηση της εφαρμογής	35
6.1	Ανάπτυξη με το Django framework	35
6.1.1	Δημιουργία Models	35
6.1.2	Views και Urls	35
6.1.3	User Interface (Templates)	37
6.2	REST API Integration και διασύνδεση με το SSH protocol	38
7	Επίδειξη της εφαρμογής(Application Demo)	41
7.1	Εισαγωγή-Η λογική της λειτουργίας της εφαρμογής Django	41
7.2	Η αρχική σελίδα της εφαρμογής	42
7.3	Devices	43
7.4	Στατιστικά της συσκευής	45
7.5	Στατιστικά της διεπαφής	47
7.6	Backup της συσκευής	48
7.7	Διαμόρφωση διεύθυνσης IP	49
8	Containerization και Deployment	53
8.1	Containerization με Docker	53
8.1.1	Δημιουργία Docker Image	53
8.2	Deployment με Kubernetes	56
8.2.1	Δημιουργία Kubernetes manifest files	57
8.2.2	Πρόσβαση στο Django pod	59
8.3	Use cases με Kubernetes	60
9	Συμπέρασματα και Μελλοντική Εργασία	63
9.1	Συμπεράσματα	63
9.2	Προκλήσεις και Μαθήματα	63
9.3	Μελλοντική Εργασία και Επέκταση Λειτουργικότητας	63
10	Παράρτημα	65
10.1	Σχεδίαση και Διάγραμμα τοπολογίας δικτύου	65
10.2	Οδηγός χρήσης της εφαρμογής	65
10.3	Εκκίνηση της εφαρμογής	65
10.3.1	Τοπικά	65
10.3.2	Docker Container	66
10.3.3	Access της εφαρμογής μέσα από το pod	66
10.3.4	Δημιουργία αντικειμένου-Προσθήκη συσκευής	66
10.4	Κώδικας	67
11	Βιβλιογραφία	69

Κατάλογος Σχημάτων

4.1	Virtualization Γενική αρχιτεκτονική	19
4.2	Virtualization αρχιτεκτονική που ακολουθήθηκε στην εφαρμογή	20
4.3	Εφαρμογή Virtualbox	20
5.1	MVC μοντέλο	23
5.2	Διαγραμματική απεικόνιση της εφαρμογής	24
5.3	Local PC-GNS3VM-CISCO IOS Connection Architecture	26
5.4	CLOUD NODE With CISCO Router Device Point to Point Connection	27
5.5	CLOUD NODE Network Interface Configuration	27
5.6	GNS3 VM κάρτα δικτύου-Γεφυρωμένη κάρτα	28
5.7	Προσαρμογέας δικτύου για την εικονική μηχανή GNS3 VM	28
5.8	Διεπαφή δικτύου στο περιβάλλον των Windows	29
5.9	Παραμετροποίηση στη δικτυακή συσκευή	29
5.10	Κατανομή IP διευθύνσης	29
5.11	IP connectivity test	30
5.12	Import appliance	31
5.13	Install appliance	31
5.14	Appliance	32
5.15	filename configuration	32
5.16	Browse Devices	33
5.17	ssh and credentials requirements	34
6.1	Μοντέλα συσκευών	35
6.2	Αρχείο urls.py	36
6.3	Αρχείο views.py	37
6.4	Παράδειγμα html αρχείου	38
6.5	Παράδειγμα wireshark	39
6.6	Παράδειγμα wireshark	39
6.7	Εμφάνιση απάντησης συσκευής τυπωμένη στο τερματικό	40
7.1	Σνάρτηση firstPage στο αρχείο views.py	41
7.2	Urls δρομολόγηση για την αρχική σελίδα.	41
7.3	Urls.py αρχείο	42
7.4	Έξοδος τερματικού κατά την εκκίνηση του Django server	42
7.5	Αρχική Σελίδα της εφαρμογής	43

7.6	Αρχικοποίηση της συσκευής μέσω της κλάσης Device	43
7.7	Controller-Device Interfaces	44
7.8	Interfaces status	45
7.9	Στατιστικά Συσκευής	46
7.10	Στατιστικά Συσκευής	47
7.11	Στατιστικά διεπαφής αρχική σελίδα	48
7.12	Στατιστικά διεπαφής	48
7.13	Get Backup of Running config	49
7.14	Επιστροφή Τρέχων Διαμόρφωση	49
7.15	IP Διαμόρφωση	50
7.16	Επιστροφή επιβεβαίωσης επιτυχούς ρύθμισης IP	50
7.17	Μεταβολή της IP διεύθυνσης λόγω διαμόρφωσης	51
8.1	Dockerfile	54
8.2	Docker build-Δημιουργία του κοντεινέρ	55
8.3	Dockerfile Path	55
8.4	Αρχείο requirements.txt	56
8.5	Docker image list	56
8.6	Minikube deployment	58
8.7	Spec του YAML file	58
8.8	Manifest for Django pod	59
8.9	Πρόσβαση στο Django pod μέσα από τη λειτουργία Port forward	60
8.10	Deployment yaml for Django pod	61
10.1	Network topology design	65
10.2	GUI Login	66
10.3	GUI Login second page	67
10.4	Add device page	67
10.5	github repo	68

3 Εισαγωγή

3.1 Στόχοι του έργου

Ο πρωταρχικός σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη μιας ολοκληρωμένης εφαρμογής που ενσωματώνει τις δυνατότητες της αυτοματοποίησης δικτύων και της ανάπτυξης εφαρμογών Ιστού. Η μελέτη επικεντρώνεται στη χρήση σύγχρονων τεχνολογών, συμπεριλαμβανομένων του Kubernetes, του Django και της γλώσσας προγραμματισμού Python. Ο βασικός στόχος αυτής της διπλωματικής εργασίας είναι η ανάπτυξη μιας ολοκληρωμένης εφαρμογής που ενσωματώνει τις δυνατότητες της αυτοματοποίησης δικτύων και της ανάπτυξης εφαρμογών Ιστού. Η εργασία εστιάζει στη χρήση σύγχρονων εργαλείων όπως το Kubernetes, το Django και την γλώσσα προγραμματισμού Python.

Συγκεκριμένα, η εφαρμογή που αναπτύχθηκε επιτρέπει την αυτοματοποιημένη διαχείριση δικτυακών συσκευών μέσω ενός γραφικού περιβάλλοντος. Η υλοποίηση βασίστηκε στο Django, το οποίο χρησιμοποιήθηκε ως το βασικό backend framework. Χάρη στο Django, δημιουργήθηκαν οι απαραίτητες συναρτήσεις και ένα απλό frontend περιβάλλον, επιτρέποντας την εστίαση στη λειτουργικότητα της εφαρμογής, αντί της εμφάνισης προς τον χρήστη. Για τη δοκιμή και αξιολόγηση της εφαρμογής, αξιοποιήθηκε το GNS3, ένα εργαλείο προσομίωσης δικτυακών συσκευών. Το GNS3 επέτρεψε την προσομοίωση ενός πραγματικού δικτυακού περιβάλλοντος, διευκολύνοντας τις δοκιμές και την επικοινωνία του Django service με εικονικές συσκευές. Χωρίς το GNS3, η διαδικασία αυτή θα απαιτούσε φυσικό εξοπλισμό, γεγονός που θα καθιστούσε τις δοκιμές ιδιαίτερα απαιτητικές τόσο σε κόστος όσο και σε πολυπλοκότητα, λόγω της ανάγκης για αγορά και της δικτυακή ενσωμάτωσή του.

Όταν η εφαρμογή έφτασε σε ένα ικανοποιητικό επίπεδο ανάπτυξης, ενσωματώθηκαν Cloud Native τεχνολογίες, προκειμένου να ακολουθήσει τις σύγχρονες τάσεις στον χώρο της Πληροφορικής και των δικτύων. Για την υλοποίηση αυτής της προσέγγισης, δημιουργήθηκε σε ένα Linux περιβάλλον το κατάλληλο περιβάλλον εκτέλεσης, ώστε η εφαρμογή να μετατραπεί σε Image και μετά container μέσα σε ένα Kubernetes pod. Στη συνέχεια, η διαχείρισή της έγινε μέσω του Kubernetes, το οποίο επιτρέπει την εύκολη ανάπτυξη, συντήρηση και κλιμάκωση των υπηρεσιών.

3.2 Περιγραφή προβλήματος και λύσης

Η διαχείριση δικτυακών υποδομών έχει γίνει εξαιρετικά περίπλοκη λόγω του μεγέθους και της πολυπλοκότητας των σύγχρονων δικτύων. Η χειροκίνητη διαχείριση αυτών των υποδομών είναι χρονοβόρα και επιρρεπής σε σφάλματα διαδικασία, ενώ δεν μπορεί να ανταποκριθεί

επαρκώς στις αυξανόμενες απαιτήσεις για ευελιξία, ταχύτητα και αξιοπιστία. Τα παραδοσιακά μοντέλα διαχείρισης συσκευών απαιτούν εξειδικευμένες γνώσεις, καθιστώντας δύσκολη την προσαρμογή στις ταχέως μεταβαλλόμενες συνθήκες. Όταν μιλάμε για τα παραδοσιακά μοντέλα διαχείρισης συσκευών, αναφερόμαστε στη χειροκίνητη διαδικασία μέσω της οποίας οι μηχανικοί πραγματοποιούσαν αλλαγές και ρυθμίσεις στις υποδομές. Αυτή η προσέγγιση απαιτούσε άμεση ανθρώπινη παρέμβαση, καθιστώντας τη διαδικασία πιο χρονοβόρα, επιρρεπή σε σφάλματα και δύσκολα διαχειρίσιμη σε μεγάλης κλίμακας περιβάλλοντα.

Αν και η εφαρμογή μας δεν περιλαμβάνει ιδιαίτερα σύνθετους μηχανισμούς αυτοματοποιημένης διαχείρισης, στις μεγάλες επιχειρήσεις σύγχρονα εργαλεία αυτού του τύπου χρησιμοποιούνται για την υλοποίηση πιο περίπλοκων διαδικασιών. Παρόλα αυτά, η προτεινόμενη λύση συμβάλλει στην αντιμετώπιση παρόμοιων προκλήσεων, μειώνοντας σημαντικά την πιθανότητα ανθρώπινου σφάλματος και βελτιώνοντας την αποδοτικότητα των δικτυακών διαχειριστικών εργασιών.

Η λύση που προτείνεται στην παρούσα εργασία περιλαμβάνει την υλοποίηση μιας εφαρμογής που αξιοποιεί ένα Django backend για την παραμετροποίηση των δικτυακών συσκευών με στόχο να μειώσει την ανθρώπινη παρέμβαση, να εξαλείψει επαναλαμβανόμενες εργασίες και να ενισχύσει τη δυνατότητα λήψης αποφάσεων σε πραγματικό χρόνο. Η διαχείριση δικτυακών συσκευών απαιτεί εξειδικευμένες γνώσεις και εξοικείωση με το περιβάλλον γραμμής εντολών (Command Line Interface - CLI). Ωστόσο, με τη χρήση μιας εφαρμογής που αυτοματοποιεί ή απλοποιεί τις διαδικασίες διαχείρισης, η ανάγκη για αυτή την εξειδίκευση περιορίζεται σημαντικά, καθώς απαιτείται μόνο η κατανόηση θεμελιωδών δικτυακών εννοιών. Συνεπώς, η ανάγκη εκπαίδευσης του ανθρώπινου δυναμικού σε εξειδικευμένα αντικείμενα μειώνεται, καθιστώντας τη διαδικασία εκμάθησης λιγότερο χρονοβόρα και απαιτητική.

Μέσα από τη χρήση διαφόρων βιβλιοθηκών Python, των εργαλείων και βιβλιοθηκών Django, και της τεχνολογίας Kubernetes, επιτυγχάνεται η κεντρικοποιημένη διαχείριση και η δυναμική προσαρμογή της εφαρμογής σύμφωνα με τις ανάγκες του οργανισμού. Η δυνατότητα δυναμικής προσαρμογής της εφαρμογής δεν υλοποιήθηκε στο πλαίσιο της παρούσας διπλωματικής εργασίας, ωστόσο αναφέρεται ως μια προοπτική που θα μπορούσε να αξιοποιηθεί αν αυτή η εφαρμογή υλοποιούνταν σε ένα πραγματικό και μεγαλύτερο περιβάλλον. Σε περιπτώσεις αυξημένης ζήτησης για υπηρεσίες, η ανάγκη για επέκταση της εφαρμογής θα μπορούσε να καλυφθεί αυτόματα, εξασφαλίζοντας αποδοτικότητα και ευελιξία. Στο Kubernetes, αυτό επιτυγχάνεται μέσω του Horizontal Pod Autoscaling, το οποίο επιτρέπει την προσαρμογή της εφαρμογής ανάλογα με τους διαθέσιμους πόρους και τις απαιτήσεις για ζήτηση υπηρεσίας.

3.3 Τεχνολογίες που χρησιμοποιήθηκαν και γιατί

Η παρούσα εργασία επικεντρώνεται στην ανάπτυξη μιας εφαρμογής για τη διαχείριση δικτυακών υποδομών, χρησιμοποιώντας σύγχρονες τεχνολογίες και εργαλεία. Η Python α-

ποτέλεσε τη βασική γλώσσα προγραμματισμού, χάρη στην ευκολία της στη σύνταξη κώδικα, αλλά και στη μεγάλη συλλογή βιβλιοθηκών που προσφέρει, ειδικά για δικτυακές εφαρμογές και αυτοματοποίηση. Μέσω της ενσωμάτωσης πρωτοκόλλων όπως το SSH και το REST, καταφέραμε να επιτύχουμε την αποτελεσματική διαχείριση συσκευών και τη διεκπεραίωση κρίσιμων λειτουργιών. Η Python χρησιμοποιήθηκε κυρίως για την αυτοματοποίηση διαδικασιών, την ανάπτυξη scripts που παρακολουθούν τη λειτουργία των συσκευών, καθώς και για την υλοποίηση API που ενισχύουν τη διαλειτουργικότητα της εφαρμογής.

Για την ανάπτυξη του backend, επιλέξαμε το Django, το οποίο ξεχωρίζει για την ευελιξία, την ασφάλεια και την ταχύτητα ανάπτυξης που προσφέρει. Το Django αξιοποιήθηκε για τη δημιουργία της κεντρικής διεπαφής διαχείρισης των δικτύων, επιτρέποντας την επεξεργασία δεδομένων και την παροχή δυναμικών υπηρεσιών στους χρήστες. Με τη χρήση αυτού του ισχυρού πλαισίου, μπορέσαμε να διαχειριστούμε δεδομένα με τρόπο ασφαλή και αξιόπιστο, ενώ παράλληλα υποστηρίξαμε την ταχύτερη ανάπτυξη της εφαρμογής.

Για τη διαχείριση των microservices και την εξασφάλιση της κλιμακωσιμότητας της εφαρμογής, υιοθετήσαμε το Kubernetes. Η χρήση του Kubernetes μας έδωσε τη δυνατότητα να διαχειριστούμε κοντέινερ που φιλοξενούσαν τα microservices, επιτρέποντας τη δυναμική ανάπτυξη, την αποτελεσματική κατανομή πόρων και τη συντήρηση της εφαρμογής. Αυτή η προσέγγιση εξασφάλισε ότι η εφαρμογή θα μπορούσε να προσαρμοστεί σε αυξημένες απαιτήσεις φορτίου, και αυξημένη ζήτηση για υπηρεσία.

Ένα ιδιαίτερα σημαντικό στοιχείο της εργασίας ήταν η δυνατότητα δοκιμής της εφαρμογής σε περιβάλλοντα που προσδομοίωνουν πραγματικές συνθήκες. Για τον σκοπό αυτό, δημιουργήσαμε εικονικά δίκτυα χρησιμοποιώντας το εργαλείο GNS3 (Graphical Network Simulator-3), σε συνδυασμό με Cisco Images και VirtualBox. Αυτή η υποδομή επέτρεψε την εξομοίωση πραγματικών δικτυακών συσκευών, διευκολύνοντας την ανίχνευση και την επίλυση προβλημάτων πριν από την εφαρμογή του συστήματος σε πραγματικά δίκτυα. Η διαδικασία αυτή αποδειχθήκε καθοριστική, καθώς μας επέτρεψε να βελτιώσουμε την αξιοπιστία και τη λειτουργικότητα της εφαρμογής, μειώνοντας σημαντικά τον κίνδυνο αποτυχίας σε πραγματικές συνθήκες.

3.4 Διάρθρωση της παρούσας εργασίας

Η παρούσα πτυχιακή εργασία αποτελεί το αποτέλεσμα μιας συστηματικής και επίμονης προσπάθειας, με στόχο την ανάπτυξη μιας ολοκληρωμένης εφαρμογής, αξιοποιώντας σύγχρονες τεχνολογίες ανάπτυξης λογισμικού. Μέσα από αυτήν την εργασία, επιχειρείται η αναλυτική παρουσίαση όλων των σταδίων της διαδικασίας ανάπτυξης, από τη θεωρητική θεμελίωση έως την τελική υλοποίηση και αξιολόγηση.

Στο πρώτο κεφάλαιο γίνεται ένας πρόλογος της εργασίας ενώ στο τρίτο παρουσιάζεται η εισαγωγή στο θέμα της πτυχιακής εργασίας. Αναλύονται οι στόχοι, το γενικό πλαίσιο και η σπουδαιότητα της εφαρμογής. Γίνεται μια σύντομη αναφορά στο πρόβλημα που επιχειρείται να επιλυθεί, καθώς και στη συνεισφορά της εργασίας στην ευρύτερη επιστημονική κοινότητα.

Το τέταρτο κεφάλαιο επικεντρώνεται στο θεωρητικό υπόβαθρο, παρουσιάζοντας τις βασικές αρχές, τα μοντέλα και τις τεχνολογίες που υποστηρίζουν την ανάπτυξη της εφαρμογής. Αναλύονται οι απαιτήσεις του έργου και περιγράφονται τα εργαλεία που χρησιμοποιήθηκαν.

Στο πέμπτο κεφάλαιο περιγράφεται η σχεδίαση της εφαρμογής. Παρουσιάζεται η μεθοδολογία ανάπτυξης, οι αρχιτεκτονικές επιλογές και οι σχεδιαστικές αποφάσεις που ελήφθησαν. Επιπλέον, αναλύεται η λογική δομή της εφαρμογής μέσω διαγραμμάτων και περιγράφονται οι λειτουργικές και μη λειτουργικές απαιτήσεις.

Το έκτο κεφάλαιο καλύπτει την υλοποίηση της εφαρμογής. Περιγράφονται βήμα προς βήμα τα επιμέρους στάδια ανάπτυξης, οι τεχνολογίες που χρησιμοποιήθηκαν, καθώς και οι προκλήσεις που προέκυψαν. Παρουσιάζονται, επίσης, τα κύρια χαρακτηριστικά της εφαρμογής και οι λειτουργίες που υλοποιήθηκαν ενώ γίνεται και μια εκτεταμένη παρουσίαση του testbed.

Στο έβδομο κεφάλαιο γίνεται επίδειξη της εφαρμογής. Παρουσιάζεται η λειτουργία της εφαρμογής μέσα από παραδείγματα χρήσης, καθώς και τα αποτελέσματα που επιτεύχθηκαν.

Το όγδοο κεφάλαιο εστιάζει στη διαδικασία containerization και στην ανάπτυξη της εφαρμογής σε περιβάλλοντα κυβερνήτη. Εξηγείται η επιλογή της συγκεκριμένης τεχνολογίας, τα πλεονεκτήματά της και τα βήματα που ακολουθήθηκαν για την ολοκλήρωση της διαδικασίας μετατροπής Image,container μέχρι και την υλοποίηση κυβερνήτη.

Στο ένατο κεφάλαιο παρουσιάζονται τα συμπεράσματα που προέκυψαν από την εκπόνηση της εργασίας. Αξιολογείται η απόδοση της εφαρμογής, καταγράφονται τα διδάγματα που αντλήθηκαν από τη διαδικασία ανάπτυξης, ενώ παράλληλα προτείνονται ιδέες για μελλοντικές βελτιώσεις και επεκτάσεις.

Η εργασία αυτή στοχεύει να προσφέρει μια ολοκληρωμένη εικόνα της ανάπτυξης μιας σύγχρονης εφαρμογής, συνδυάζοντας θεωρία, σχεδιασμό, υλοποίηση και τεχνολογίες αιχμής, όπως το containerization, σε ένα ενιαίο, δομημένο πλαίσιο.

4 Θεωρητικό υπόβαθρο

4.1 Αυτοματοποίηση Διαχείρισης Δικτύου

Η αυτοματοποίηση δικτύου δεν περιορίζεται μόνο στη διαμόρφωση συσκευών. Αντιθέτως, το πιο σημαντικό μέρος της αυτοματοποίησης δικτύου, που συμβάλλει στη μείωση των ανθρώπινων σφαλμάτων, είναι η δυνατότητα που παρέχει στους διαχειριστές να αυτοματοποιούν διαδικασίες για τη διενέργεια ελέγχων συμμόρφωσης και επικύρωσης της τρέχουσας διαμόρφωσης ή οποιασδήποτε διαμόρφωσης πρόκειται να εφαρμοστεί.

Αυτό έχει ως αποτέλεσμα τη μείωση του χρόνου υλοποίησης των αλλαγών στο δίκτυο και του κινδύνου διακοπής ή διατάραξης της υπηρεσίας, ενώ ελαχιστοποιεί την πιθανότητα ανθρώπινου λάθους και διασφαλίζει την ευθυγράμμιση με τις πολιτικές του δικτύου. Μία ακόμη διαδικασία που μπορεί να αυτοματοποιηθεί είναι η επίλυση προβλημάτων. Όταν προκύπτει κάποιο πρόβλημα στο δίκτυο, το πρώτο βήμα για την αντιμετώπισή του είναι η συλλογή πληροφοριών. Η συλλογή πληροφοριών από κάθε συσκευή μπορεί να είναι χρονοβόρα και περίπλοκη, κάτι που είναι κρίσιμο, διότι συνήθως, στο μεταξύ, το δίκτυο ή ένα μέρος του παραμένει εκτός λειτουργίας.

Με τη χρήση της αυτοματοποίησης δικτύου, μπορούμε να αυτοματοποιήσουμε τις εντολές που απαιτούνται για τη συλλογή των απαραίτητων πληροφοριών για την επίλυση προβλημάτων και να έχουμε πρόσβαση σε αυτές σε πραγματικό χρόνο. Η προγραμματιστική συλλογή αυτών των πληροφοριών επιπρέπει και τον έλεγχο τους σε πραγματικό χρόνο.

Η παρούσα διπλωματική εργασία βασίστηκε σε μια συναφή μελέτη (<https://github.com/dmfigol/network-programmability-stream>), η οποία υλοποιήθηκε από έναν μηχανικό που ανέπτυξε μια παρόμοια εφαρμογή χρησιμοποιώντας το Django. Στην εργασία του, ο δημιουργός εστίασε στην υλοποίηση βασικών λειτουργιών, αξιοποιώντας ένα διαφορετικό περιβάλλον, καθώς είχε πρόσβαση σε εικονικές συσκευές. Συνεπώς, δεν απαιτήθηκε η χρήση του GNS3 για δοκιμές.

Οι κύριες λειτουργίες που ανέπτυξε περιλάμβαναν την καταγραφή των συσκευών και τη διαμόρφωση διευθύνσεων IP. Αξίζει να σημειωθεί ότι, λόγω της επαγγελματικής μου εμπειρίας στην Ericsson, έχω έρθει σε επαφή με πιο προηγμένα συστήματα αυτού του τύπου, όπως το ENM (Ericsson Network Manager)(2). Το συγκεκριμένο σύστημα παρέχει εξειδικευμένες λύσεις για τη διαχείριση εξαιρετικά πολύπλοκων τηλεπικοινωνιακών υποδομών, όπου η διαχείριση, η παραμετροποίηση και η παρακολούθηση των δικτυακών συσκευών είναι κρίσιμης σημασίας.

Ο έλεγχος πληροφοριών σε πραγματικό χρόνο και η άμεση λήψη αποφάσεων, όταν μεταβάλλεται μια κρίσιμη παράμετρος, όπως το MTU(3), αποτελεί μία σημαντική πτυχή της

αυτοματοποιημένης διαχείρισης δικτύων, γνωστή ως αυτοματοποιημένη παρακολούθηση. Το MTU (Maximum Transmission Unit) καθορίζει το μέγιστο μέγεθος των δεδομένων που μπορούν να μεταφερθούν σε ένα πακέτο μέσω ενός συγκεκριμένου δικτυακού μέσου. Η συνεχής παρακολούθηση τέτοιων παραμέτρων μπορεί να συμβάλει στην έγκαιρη ανίχνευση πιθανών προβλημάτων, όπως η δυσλειτουργία ενός δικτυακού interface. Ακόμη περισσότερο, η αυτοματοποιημένη παρακολούθηση παίζει καθοριστικό ρόλο στην πρόληψη βλαβών που προκύπτουν από αποτυχίες υλικού, επιτρέποντας την έγκαιρη διάγνωση και την ελαχιστοποίηση του χρόνου διακοπής λειτουργίας του δικτύου. Στην παρούσα διπλωματική εργασία δεν υλοποιήθηκε αυτοματοποιημένη παρακολούθηση αλλά αυτοματοποιημένη συλλογή πληροφοριών.

4.2 Web Framework- Django

Το Django είναι ένα backend framework το οποίο βασίζεται στη γλώσσα προγραμματισμού Python. Το Django βοηθάει στο να στηθεί γρήγορα μία εφαρμογή ανάπτυξης ιστού έτσι ώστε να μπορούμε να εστιάσουμε στη σύνταξη της εφαρμογής και της λειτουργικότητάς της χωρίς να χρειάζεται να ανακαλύψουμε ξανά τον τροχό. Είναι δωρεάν και ανοιχτού κώδικα. Ορισμένες από τις πιο μεγάλες εταιρίες στον πλανήτη χρησιμοποιούν την ικανότητα του να κλιμακώνεται γρήγορα και με ευελιξία για να ανταποκρίνεται στις μεγαλύτερες απαιτήσεις κίνησης. Στη δικιά μας περίπτωση χρησιμοποιήθηκε το συγκεκριμένου Framework γιατί θα μας έδινε τη δυνατότητα να φτιάξουμε μία εφαρμογή με μεγάλη επεκτασιμότητα και παράλληλα να μπορέσουμε να ενσωματώσουμε μέσα διαφορετικές τεχνολογίες.

Παράλληλα με το Django χρησιμοποιήθηκαν έτοιμες βιβλιοθήκες της Python προκειμένου να μπορέσουν να εκτελεστούν βασικές λειτουργίες της εφαρμογής όπως τα πρωτόκολλα επικοινωνίας. Στην ενότητα 4.3 παρουσιάζονται οι βιβλιοθήκες που χρησιμοποιήθηκαν και κάποια βασικά χαρακτηριστικά τους.

Η εφαρμογή βασίστηκε στην αρχιτεκτονική REST, αξιοποιώντας τη λογική των HTTP requests και responses. Όταν ο χρήστης εισάγει ένα URL, όπως <http://127.0.0.1:8000/manage/>, το Django δρομολογεί το αίτημα μέσω του urls.py και επιστρέφει το αντίστοιχο περιεχόμενο, όπως το base1.html, μέσω του views.py.

Η χρήση REST επιτρέπει οργανωμένη επικοινωνία μεταξύ frontend και backend, ενώ η αξιοποίηση έτοιμων Python βιβλιοθηκών διευκολύνει βασικές λειτουργίες, όπως η διαχείριση των πρωτοκόλλων επικοινωνίας. Περισσότερες λεπτομέρειες για τις βιβλιοθήκες και τη δρομολόγηση παρέχονται στα επόμενα κεφάλαια.

4.3 Βιβλιοθήκες της Python για Network automation

4.3.1 Paramiko

Το Paramiko(4) είναι μια βιβλιοθήκη της Python που υλοποιεί το πρωτόκολλο SSH έκδοσης 2 σε Python, παρέχοντας λειτουργικότητα τόσο πελάτη όσο και διακομιστή. Η Paramiko βασίζεται στην κρυπτογραφία για τη λειτουργία κρυπτογράφησης, η οποία χρησιμοποιεί επεκτάσεις

C και Rust. Οποιαδήποτε συσκευή που μπορεί να ρυθμιστεί μέσω SSH μπορεί επίσης να ρυθμιστεί από την Python με σενάρια με τη χρήση αυτής της μονάδας.

4.3.2 Netmiko

Το Netmiko(5) είναι μια βιβλιοθήκη Python ανοικτού κώδικα η οποία μπορεί να υποστηρίξει πολλές διαφορετικές συσκευές διαφορετικών προμηθευτών(Cisco,Juniper,Arista και άλλοι) που σημαίνει ότι πολλές συσκευές μπορούν να ρυθμιστούν από την python χρησιμοποιώντας το Netmiko. Το framework αυτό χρησιμοποιήθηκε στην εργασία γιατί απλοποιεί τη σύνδεση με συσκευές μέσω SSH, αφαιρώντας την πολυπλοκότητα στη διαχείριση διαφορετικών πρωτοκόλλων και τύπων συσκευών.

Το Netmiko περιλαμβάνει επίσης ενσωματωμένες λειτουργίες για την εκτέλεση εντολών και τη διαχείριση ρυθμίσεων, επιταχύνοντας την ανάπτυξη λύσεων. Υποστηρίζει ασύγχρονες λειτουργίες για αποτελεσματική διαχείριση πολλαπλών ταυτόχρονων συνδέσεων, κάπι που είναι κρίσιμο για μεγάλης κλίμακας δίκτυα. Επιπλέον, έχει ισχυρή κοινότητα και τακτικές ενημερώσεις, διευκολύνοντας την υποστήριξη και την επίλυση προβλημάτων. Τόσο το Paramiko όσο και το Netmiko αποτελούν εναλλακτικές επιλογές για συσκευές που δεν υποστηρίζουν APIs επειδή βασίζεται στο SSH πρωτόκολλο.

4.3.3 Napalm

Το NAPALM(6) (Network Automation and Programmability Abstraction Layer with Multivendor support) είναι μια βιβλιοθήκη Python που υλοποιεί ένα σύνολο λειτουργιών για την αλληλεπίδραση με διαφορετικά λειτουργικά συστήματα συσκευών δικτύου χρησιμοποιώντας ένα ενοποιημένο API. Το NAPALM υποστηρίζει διάφορες μεθόδους σύνδεσης με τις συσκευές, χειρισμού των ρυθμίσεων ή ανάκτησης δεδομένων. Το Napalm συνεπώς είναι μια βιβλιοθήκη Python που παρέχει ένα API (Application Programming Interface) για την εργασία με συσκευές δικτύου. Έχει σχεδιαστεί για να απλοποιεί την αυτοματοποίηση και τη διαχείριση του δικτύου με την αφάίρεση των υποκείμενων λεπτομερειών που σχετίζονται με τον εκάστοτε προμηθευτή και την παροχή μιας συνεπούς διεπαφής σε διαφορετικά δίκτυα. συσκευών.

Το Napalm επιτρέπει στους μηχανικούς και τους διαχειριστές δικτύων να αυτοματοποιούν κοινές εργασίες διαχείρισης δικτύου, όπως η διαμόρφωση, η παροχή, η παρακολούθηση και η αντιμετώπιση προβλημάτων. Υποστηρίζει πολλούς προμηθευτές συσκευών δικτύου, συμπεριλαμβανομένων των Cisco, Juniper, Arista και Huawei. Το Napalm παρέχει ένα σύνολο κοινών λειτουργιών που μπορούν να εκτελεστούν σε συσκευές δικτύου, όπως η ανάκτηση πληροφοριών διαμόρφωσης, η εφαρμογή διαμόρφωσης αλλαγών, έλεγχος στατιστικών στοιχείων διασύνδεσης και συλλογή πληροφοριών τοπολογίας δικτύου παρέχει επίσης χαρακτηριστικά όπως υποστήριξη επαναφοράς, επικύρωση διαμόρφωσης αλλαγών διαμόρφωσης, και σύγκριση των διαφορών διαμόρφωσης μεταξύ συσκευών.

Το Napalm μπορεί να συνδυαστεί με βιβλιοθήκες Python όπως οι Netmiko, Paramiko και Ansible για τη δημιουργία σύνθετων ροών εργασίας αυτοματισμού δικτύου. Μπορεί επίσης

να ενσωματωθεί με δημοφιλή εργαλεία παρακολούθησης δικτύου, όπως το Prometheus και το Grafana, για την παρακολούθηση της απόδοσης του δικτύου σε πραγματικό χρόνο. Η συνάρτηση get network driver της βιβλιοθήκης NAPALM χρησιμοποιείται στην εφαρμογή μας προκειμένου το User Interface του Django να αποκτήσει έναν οδηγό δικτύου που επιτρέπει την αλληλεπίδραση με τις συσκευές της Cisco μέσω ενός ενιαίου API. Αυτό διευκολύνει την αυτοματοποίηση και τη διαχείριση των συσκευών χρησιμοποιώντας το πρωτόκολλο SSH.

4.4 Devops και Διαδικασίες Αυτοματισμού

4.4.1 Συνεχής Ενσωμάτωση και Παράδοση (CI/CD)

Η τελευταία τάση στον κόσμο του DevOps είναι η υλοποίηση ενός αγωγού CI/CD, ο οποίος ουσιαστικά είναι μια αυτοματοποιημένη διαδικασία που ενεργοποιείται όταν νέος κώδικας δημοσιεύεται στο απομακρυσμένο αποθετήριο. Αυτή η διαδικασία ξεκινάει τη δημιουργία κώδικα, εκτελεί κάποιες δοκιμές και τέλος, αν όλα είναι εντάξει, αναπτύσσει αυτόμata τον κώδικα στο περιβάλλον παραγωγής. Με αυτόν τον τρόπο, οι προγραμματιστές μπορούν να διασφαλίσουν ότι τίποτα δεν θα χαλάσει στην παραγωγή και οι νέες λειτουργικότητες εξυπηρετούνται το συντομότερο δυνατό στους πελάτη. Στην περίπτωσή μας τόσο η διπλωματική εργασία (Latex) όσο και η εφαρμογή υλοποιήθηκαν με αυτή τη λογική. Τα βήματα όμως από τη δημιουργία του Docker Image μέχρι το Deployment στο Kubernetes επίπεδο γίνανε manually προκειμένου να καταλάβουμε σε βάθος τα βήματα που ακολουθούνται με αυτοματοποιημένο τρόπο μέσα από εργαλεία όπως το Jenkins και το GitLab CI/CD. Για τη δικιά μας εφαρμογή δε θεωρήθηκε απαραίτητη η χρήση εργαλείων CI/CD τέτοιων όπως για παράδειγμα το GitHub Actions καθώς ήταν επιλογή μας το παραπάνω με σκοπό την εξοικείωση με το Kubernetes/Containerization της DevOps κουλτούρας.

4.4.2 Διαχείρηση εκδόσεων (Version Control) με Git και GitHub

Το Git(7) είναι ένα σύγχρονο σύστημα ελέγχου εκδόσεων (γνωστό και ως σύστημα διαχείρισης αναθεωρήσεων ή πηγαίου κώδικα), σχεδιασμένο με έμφαση στην ταχύτητα, την ακεραιότητα των δεδομένων και την υποστήριξη κατανεμημένων, μη γραμμικών ροών εργασίας. Στην παρούσα διπλωματική εργασία, θα αξιοποιήσουμε το Git για να διασφαλίσουμε την ορθή διαχείριση των εκδόσεων του λογισμικού, τόσο κατά τη διάρκεια της ανάπτυξης όσο και για την πιθανή μελλοντική χρήση και εξέλιξη της δουλειάς μας. Το GIT συνεπώς είναι απαραίτητο σε κάθε σοβαρό έργο ανάπτυξης, και αυτό δεν αποτελεί εξαίρεση. Παρέχει γρήγορη ανάπτυξη κώδικα, έκδοση και επιτρέπει διακλαδώσεις. Έχοντας το εγκατεστημένο τόσο στον τοπικό υπολογιστή ανάπτυξης όσο και στο περιβάλλον παραγωγής, διευκολύνει και επιταχύνει τη διαδικασία ανάπτυξης στο παραγωγικό περιβάλλον. Το Git παρέχει τη δυνατότητα επαναφοράς σε προηγούμενες εκδόσεις κώδικα σε περίπτωση κάποιο άγνωστο σφάλμα εμφανιστεί σε μια νεότερη έκδοση.

4.4.3 Containers και Docker

Το Docker είναι μια πλατφόρμα που επιτρέπει τη δημιουργία, τη διανομή και την εκτέλεση εφαρμογών μέσα σε ελαφριά, απομονωμένα "κοντέινερ" (containers). Τα κοντέινερ περιλαμβάνουν ό,τι χρειάζεται μια εφαρμογή για να τρέξει, όπως κώδικα, βιβλιοθήκες και εξαρτήσεις, διασφαλίζοντας ότι θα λειτουργεί ομοιόμορφα ανεξάρτητα από το περιβάλλον στο οποίο εκτελείται. Με αυτόν τον τρόπο διευκολύνεται η διαχείριση και η μεταφορά εφαρμογών από τον έναν υπολογιστή ή διακομιστή στον άλλον. Στη διπλωματική μας χρησιμοποιήθηκε το docker daemon(8) του linux

4.4.4 Kubernetes και Container Orchestration

Ο κυβερνήτης είναι ο διαχειριστής των με απλά λόγια ο διαχειριστής των containers. Είναι μια πλατφόρμα ανοικτού κώδικα για τη διαχείριση φορτίων εργασίας και υπηρεσιών που περιέχουν containers, η οποία διευκολύνει τόσο τη δηλωτική διαμόρφωση όσο και την αυτοματοποίηση. Διαθέτει ένα μεγάλο, ταχέως αναπτυσσόμενο οικοσύστημα. Οι υπηρεσίες, η υποστήριξη και τα εργαλεία του Kubernetes είναι ευρέως διαθέσιμα. Για το σκοπό αυτό χρησιμοποιήσαμε ως διαχειριστή των containers το minikube.(9)

4.5 Προσομοίωση και Εικονικά Περιβάλλοντα

4.5.1 GNS3 και Εικονικά Δίκτυα

Το GNS3(10) Είναι ένα εργαλείο προσομοίωσης δικτύων ανοικτού κώδικα που επιτρέπει στους χρήστες να προσομοιώσουν σύνθετες τοπολογίες δικτύων στους υπολογιστές τους. Μηχανικοί δικτύων και φοιτητές το χρησιμοποιούν ευρέως για να μάθουν και να εξασκηθούν σε έννοιες δικτύωσης, να δοκιμάσουν διαμορφώσεις δικτύου και να δημιουργήσουν εικονικά περιβάλλοντα δικτύου.

Το GNS3 υποστηρίζει διάφορες συσκευές δικτύου, όπως δρομολογητές, μεταγωγείς και τείχη προστασίας από διάφορους προμηθευτές, συμπεριλαμβανομένων των Cisco, Juniper, Nokia και άλλων. Επιτρέπει στους χρήστες να προσομοιώσουν διάφορα σενάρια και διαμορφώσεις δικτύου και να δοκιμάσουν τη συμπεριφορά των συσκευών δικτύου σε ένα ελεγχόμενο περιβάλλον. Το πλεονέκτημα του GNS3 σε σχέση με άλλες εφαρμογές όπως το Packet tracer είναι ότι το GNS3 μπορεί να σηκώσει πραγματικά images άρα πραγματικό λογισμικό (IOS) συνεπώς οι λειτουργίες προσομοίωσης είναι πιο ρεαλιστικές.

Στην παρούσα διπλωματική εργασία χρησιμοποιούνται δύο τεχνολογίες του GNS3. Το GNS3 λογισμικό και το GNS3 VM. Τα δύο αυτά εργαλεία λογισμικού μας βοηθάνε στο κομμάτι της προσομείωσης των δικτυακών συσκευών. Περισσότερες λεπτομέριες για τη δημιουργία του testbed περιγράφονται στο κεφάλαιο 5.3

Το IOU (Cisco IOS on UNIX) είναι μια εικονική έκδοση του λογισμικού IOS της Cisco που μπορεί να χρησιμοποιηθεί για σκοπούς προσομοίωσης και δοκιμής δικτύου. Επιτρέπει στους μηχανικούς δικτύου να δημιουργούν εικονικές τοπολογίες δικτύου και να εξασκούνται σε

διάφορες εργασίες δικτύου, όπως η διαμόρφωση δρομολογητών και μεταγωγέων, χωρίς να απαιτείται φυσικό υλικό.

Το IOS μπορεί να τρέχει κατευθείαν πάνω σε υλικό, δηλαδή μπορεί να εγκατασταθεί απευθείας πάνω στα router και στα switches . Το IOU είναι λογισμικό το οποίο μπορεί να τρέξει το λογισμικό (IOS) ακόμα και σε PC προσομοιώνοντας με αυτόν τον τρόπο λογισμικό δικτυακών συσκευών χωρίς την ανάγκη ύπαρξης συγκεκριμένου υλικού. Στην εφαρμογή μας χρησιμοποιήθηκαν τέτοιες εικονικές εκδόσεις προκειμένου να προσομειώσουμε τις Cisco συσκευές καθώς δεν υπήρχε διαθέσιμο πειραματικό υλικό τέτοιο που να μας επιτρέψει την προσομείωση IOS. Τα CISCO IOU είναι εύκολα στην εγκατάσταση και την προσομοίωσή τους συνεπώς κάνουν και τη δημιουργία του testbed ευκολότερη.

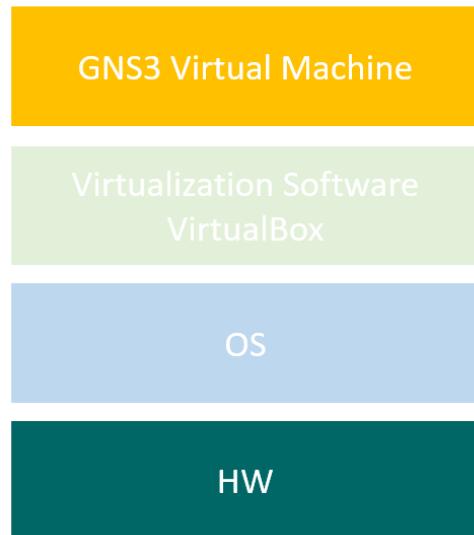
4.5.2 Πρόγραμμα εικονοποίησης για το GNS3 VM-VirtualBox

Στην επιστήμη της πληροφορικής, η εικονικοποίηση virtualization είναι ένας ευρύς όρος των υπολογιστικών συστημάτων που αναφέρεται σε έναν μηχανισμό αφαιρεσης, στοχευμένο στην απόκρυψη λειτουρειών της υλοποίησης και της κατάστασης ορισμένων υπολογιστικών πόρων από πελάτες των πόρων αυτών (π.χ. εφαρμογές, άλλα συστήματα, χρήστες κλπ). Στη συγκεκριμένη εργασία, η εικονικοποίηση αξιοποιήθηκε για τη δημιουργία του εργαστηριακού πειριβάλλοντος. Το GNS3 χρησιμοποιήθηκε σε συνδυασμό με το VirtualBox, το οποίο λειτουργεί ως hypervisor τύπου 2. Ένας hypervisor τύπου 2 είναι ένα λογισμικό που εγκαθίσταται πάνω σε ένα υπάρχον λειτουργικό σύστημα και επιτρέπει την εκτέλεση εικονικών μηχανών

To Oracle VM VirtualBox(11) ή VirtualBox (πρώην Sun VirtualBox, Sun xVM VirtualBox και Innotek VirtualBox) είναι υπερεπόπτης ανοιχτού κώδικα για υπολογιστές x86 που αναπτύσσεται από την Oracle Corporation. Αναπτύχθηκε αρχικά από την Innotek GmbH και αποκτήθηκε από τη Sun Microsystems το 2008, η οποία εξαγοράστηκε από την Oracle το 2010.

To VirtualBox μπορεί να εγκατασταθεί σε διάφορα λειτουργικά συστήματα, συμπεριλαμβανόμενων των Linux, macOS, Windows, Solaris και OpenSolaris. Υπάρχουν επίσης μεταφορές για το FreeBSD και το Genode. Υποστηρίζει τη δημιουργία και τη διαχείριση εικονικών μηχανών που εκτελούν εκδόσεις και παραλλαγές των Microsoft Windows, Linux, BSD, Solaris, Haiku, OSx86 και άλλα, καθώς και περιορισμένη εικονικοποίηση macOS. Για ορισμένα λειτουργικά συστήματα είναι διαθέσιμο ένα πακέτο "Guest Additions" από μηχανές συσκευών και εφαρμογές συστήματος που συνήθως βελτιώνει την απόδοση, ειδικά των γραφικών, επίσης δίνει την δυνατότητα στον χρήστη να μεταφέρει αρχεία ή κείμενο από μία εικονική μηχανή στον υπολογιστή του χρήστη και να αυξήσει την ανάλυση του παράθυρου της μηχανής. Η γενική αρχιτεκτονική παρουσιάζεται παρακάτω στο σχήμα 4.1

Με τη χρήση του VirtualBox, δημιουργήθηκε ένα εικονικό περιβάλλον όπου το GNS3 μπορούσε να τρέξει τις προσομοιώσεις δικτύου, χωρίς την ανάγκη φυσικού εξοπλισμού. Αυτό μας έδωσε τη δυνατότητα να δοκιμάσουμε την εφαρμογή μας και να επαληθεύσουμε τη λειτουργικότητά της σε ένα ελεγχόμενο περιβάλλον, εξασφαλίζοντας ρεαλιστικά σενάρια προσομοίωσης. Οι παρακάτω εικόνες βοηθούν στην κατανόηση της γενικής και ειδικής αρχιτεκτονικής που χρησιμοποιήθηκε



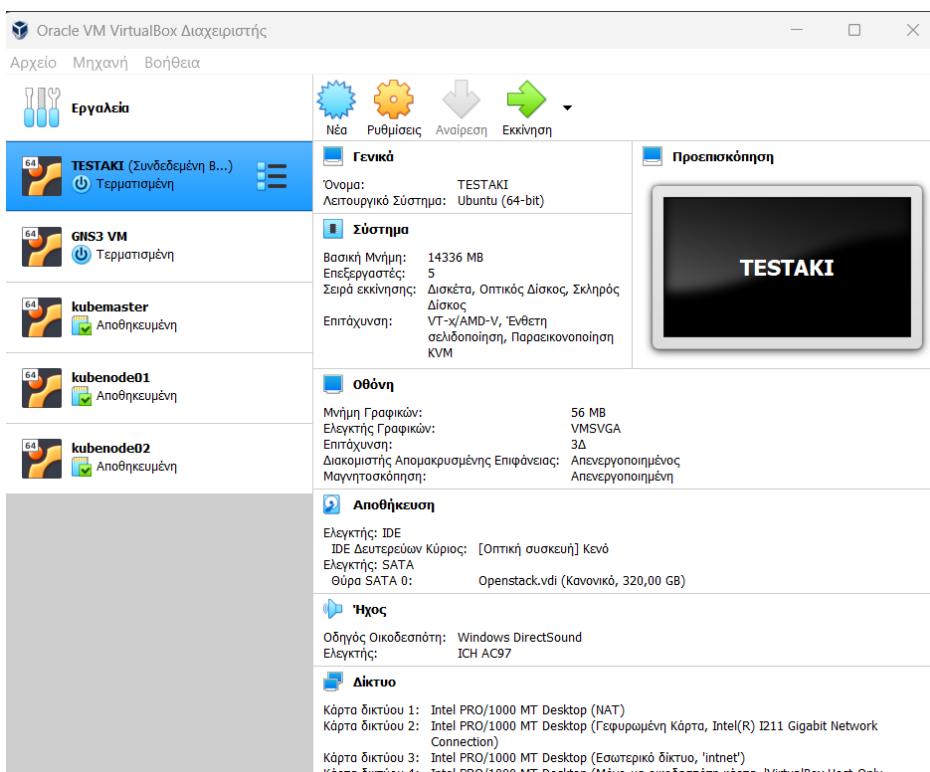
Σχήμα 4.1. Virtualization Γενική αρχιτεκτονική

To VirtualBox χρησιμοποιήθηκε συνεπώς προκειμένου να σηκώσουμε το GNS3 VM. Οι developers του GNS3 συνιστούν για τις περισσότερες περιπτώσεις την εγκατάσταση και του GNS3 VM όταν χρησιμοποιούμε Windows OS(12). Συνεπώς το GNS3 VM ήταν σημαντικό να εγκατασταθεί καθώς αποτελούσε συστημικό στοιχείο του GNS3 λογισμικού. Η αρχιτεκτονική της εικονοποίησης μπορεί να περιγραφεί με το σχήμα 4.2 με έναν πιο ειδικό τρόπο. Στο πρώτο επίπεδο βρίσκεται το υλικό, στο δεύτερο επίπεδο είναι το λειτουργικό σύστημα windows. Το επόμενο επίπεδο αποτελείται από το λογισμικό VirtualBox, το οποίο λειτουργεί ως hypervisor τύπου B, παρέχοντας τη δυνατότητα δημιουργίας και διαχείρισης εικονικών μηχανών. Η εικονική μηχανή GNS3 VM εκτελείται εντός του VirtualBox και χρησιμεύει ως το περιβάλλον φιλοξενίας των εικόνων του λειτουργικού συστήματος Cisco IOS, οι οποίες εκτελούνται πάνω σε αυτήν, επιτρέποντας την προσομοίωση και διαχείριση δικτυακών τοπολογιών με αποδοτικό και κλιμακούμενο τρόπο.

Στην εικόνα 4.3 μπορούμε να δούμε το περιβάλλον του VirtualBox και την εικονική μηχανή GNS3 VM.



Σχήμα 4.2. Virtualization αρχιτεκτονική που ακολουθήθηκε στην εφαρμογή



Σχήμα 4.3. Εφαρμογή Virtualbox

5 Σχεδίαση και Ανάλυση

5.1 Απαιτήσεις Εφαρμογής Διαχείρισης Δικτύου.

5.1.1 Λειτουργικές απαιτήσεις

Μια σειρά από λειτουργίες κρίθηκαν απαραίτητες και υλοποιηθήκαν στην παρούσα εφαρμογή. Έτσι:

- Η εφαρμογή διαχείρισης του δικτύου θα πρέπει να παρέχει στον χρήστη ένα γραφικό περιβάλλον που θα του επιτρέπει να αλληλεπιδρά εύκολα και αποδοτικά με το σύστημα.
- Ο χρήστης θα πρέπει να έχει την δυνατότητα να παρακολουθεί την κατάσταση μιας συγκεκριμένης διεπαφής δικτύου, βλέποντας αν λειτουργεί σωστά ή αν υπάρχουν προβλήματα.
- Επιπλέον, θα πρέπει να έχει τη δυνατότητα να βλέπει αναλυτικά στατιστικά στοιχεία για μια συγκεκριμένη δικτυακή συσκευή, όπως η χρήση δεδομένων, η ταχύτητα σύνδεσης ή τυχόν σφάλματα, αλλά και για συγκεκριμένες διεπαφές του δικτύου, ώστε να κατανοεί τη λειτουργία τους σε βάθος.
- Η εφαρμογή θα επιτρέπει επίσης τη δημιουργία αντιγράφου ασφαλείας (backup) της τρέχουσας ρύθμισης του δικτύου, για να μπορεί ο χρήστης να επαναφέρει τη ρύθμιση αν χρειαστεί. Ουσιαστικά θα δίνεται η δυνατότητα στο χρήστη να πάρει σε ένα text αρχείο το running-config. Το restoration του backup θα γίνεται όμως χειροκίνητα από το χρήστη καθώς δεν υλοποιήθηκε στην εφαρμογή λειτουργία αυτόματου restoration.
- Τέλος, θα δίνεται η δυνατότητα αλλαγής της διεύθυνσης IP μιας επιλεγμένης δικυακής εφαρμογής, εξασφαλίζοντας μεγαλύτερη ευελιξία στη διαχείριση του δικτύου.

Όλες αυτές οι λειτουργίες έχουν σχεδιαστεί για να διευκολύνουν τη διαχείριση και την παρακολούθηση του δικτύου, ακόμη και από χρήστες χωρίς εξειδικευμένες γνώσεις.

5.2 Αρχιτεκτονική της εφαρμογής

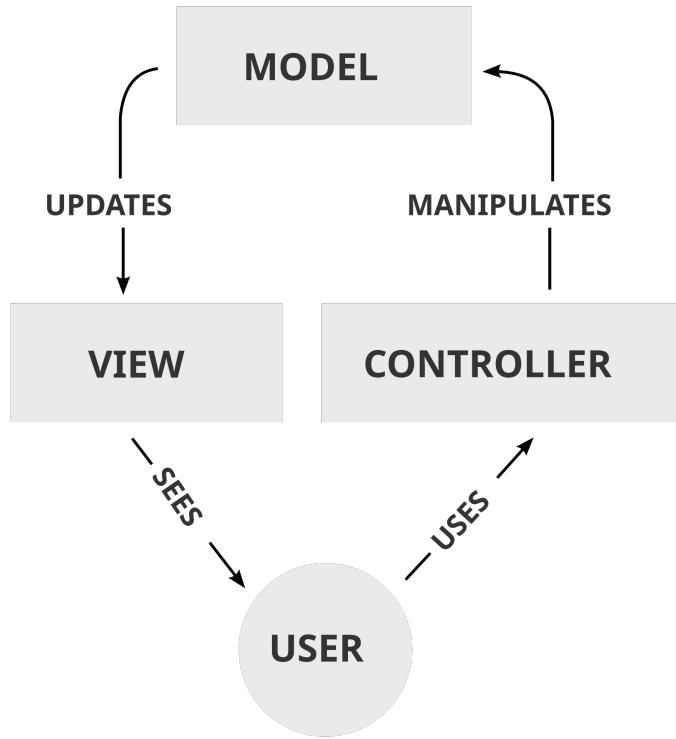
Στην αρχή, οι εφαρμογές ιστού δεν ήταν τίποτα περισσότερο από ένα σύνολο αρχείων HTML, CSS και javascript που ήταν συνδεδεμένα μεταξύ τους. Ένας καλός προγραμματιστής ήταν σε θέση να φτιάξει σπουδαίες εφαρμογές ιστού αν αυτός/αυτή είχε αρκετές δεξιότητες/γνώσεις.

Στη σύγχρονη εποχή, η χρήση των frameworks έχει διαδοθεί ευρέως, και αν και δεν επηρεάζουν άμεσα την τελική εμπειρία του χρήστη ή τις αλληλεπιδράσεις του με το frontend, παραμένουν ευρέως χρησιμοποιούμενα λόγω των πλεονεκτημάτων που προσφέρουν στην ανάπτυξη εφαρμογών. Παρόμοιες δουλειές με την παρούσα εργασία υπάρχουν και σε άλλες διπλωματικές εργασίες καθώς και σε μη διπλωματικές εργασίες. Μηχανικοί από όλο τον κόσμο ασχολούνται με την αυτοματοποίηση συστημάτων και τη δημιουργία κώδικα που να αυτοματοποιεί συσκευές/συστήματα.

Λαμβάνοντας υπόψη προηγούμενες σχετικές προσπάθειες, πραγματοποιήθηκε ανασκόπηση της υπάρχουσας βιβλιογραφίας με σκοπό την ανάπτυξη μιας εφαρμογής προσαρμοσμένης στα σύγχρονα δεδομένα. Ιδιαίτερη έμφαση δόθηκε στην ενσωμάτωση των πλέον πρόσφατων τεχνολογιών αιχμής, όπως η Cloud Native αρχιτεκτονική, ώστε να διασφαλιστεί η βέλτιστη απόδοση και προσαρμοστικότητα του συστήματος. Στη συνέχεια γίνεται προσπάθεια να δωθεί εκτενής ανάλυση στο πως λειτουργεί η εφαρμογή καθώς και στην αλληλεπίδρασή της με τα συνεργαζόμενα συστήματα.

5.2.1 Μοντέλο MVC (Model-View-Controller)

Το μοτίβο MVC(13) (Model-View-Controller σχήμα 5.1) είναι ένα αρχιτεκτονικό πρότυπο ανάπτυξης λογισμικού που έχει ως στόχο τον διαχωρισμό της παρουσίασης των δεδομένων από τη λογική που διέπει τη διαχείριση των αλληλεπιδράσεων του χρήστη. Συγκεκριμένα, το "Model" αναλαμβάνει τη διαχείριση των δεδομένων και της επιχειρησιακής λογικής, το "View" είναι υπεύθυνο για την παρουσίαση των δεδομένων στον χρήστη, ενώ το "Controller" χειρίζεται την αλληλεπίδραση του χρήστη και τη ροή των δεδομένων μεταξύ του Model και του View. Χάρη στη δομή του, το MVC προσφέρει μεγαλύτερη ευελιξία, επεκτασιμότητα και καλύτερη οργάνωση του κώδικα, καθιστώντας το ιδανικό για σύνθετες εφαρμογές. Γι' αυτόν τον λόγο, όλα τα κορυφαία frameworks για την ανάπτυξη εφαρμογών web, όπως το Django, είναι βασισμένα σε αυτό το μοτίβο.



Σχήμα 5.1. MVC μοντέλο

5.2.2 Django MTV

Παρόλο που το Django ακολουθεί το μοτίβο MVC, προτιμά να χρησιμοποιεί τη δική του λογική στην υλοποίηση. Το framework αναλαμβάνει το Controller μέρος του MVC και αφήνει τα περισσότερα από τα "καλά" να γίνονται στο Model-Template-View (MTV). Αυτός είναι ο λόγος που το Django συχνά αναφέρεται ως MTV framework.

5.2.3 Χρήση του MTV στην εφαρμογή

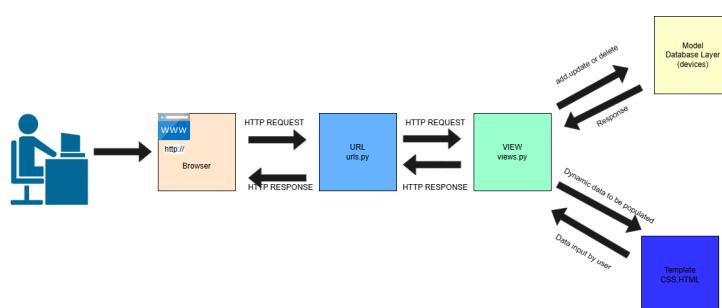
Η εφαρμογή διαχείρισης δικτύου έχει υλοποιηθεί με βάση το αρχιτεκτονικό μοτίβο MVC (Model-View-Controller), το οποίο εξασφαλίζει τη σαφή οργάνωση της λειτουργικότητας της εφαρμογής σε τρία επίπεδα: το μοντέλο για τη διαχείριση των δεδομένων, τα πρότυπα για την παρουσίαση των πληροφοριών στον χρήστη και τις προβολές για τη διασύνδεση των χρηστών με τα δεδομένα και την παρουσίασή τους. Το μοντέλο αποτελεί τον πυρήνα της εφαρμογής και διαχειρίζεται τα δεδομένα που σχετίζονται με τις δικτυακές συσκευές.

Το βασικό μοντέλο-αντικείμενο που χρησιμοποιείται εδώ είναι το "Device", το οποίο αναπαριστά συσκευές όπως δρομολογητές και switches, αποθηκεύοντας πληροφορίες όπως η διεύθυνση του host, το όνομα του χρήστη, την πλατφόρμα της συσκευής και τον κωδικός πρόσβασης. Το "Device" είναι συνδεδεμένο με τη βάση δεδομένων της εφαρμογής και παρέχει τις βασικές λειτουργίες για την εισαγωγή, ανάγνωση, επεξεργασία και διαγραφή δεδομένων. Αυτή η δομή επιτρέπει την κεντρική διαχείριση όλων των δεδομένων που απαιτούνται για την ομαλή λειτουργία της εφαρμογής.

Για την παρουσίαση των δεδομένων στον χρήστη, χρησιμοποιούνται αρχεία HTML ως πρότυπα. Αυτά τα αρχεία λειτουργούν ως δυναμικές σελίδες που δημιουργούνται με τη γλώσσα Django Template, εξασφαλίζοντας την απεικόνιση των δεδομένων με τρόπο φιλικό προς τον χρήστη. Για παράδειγμα, τα πρότυπα χρησιμοποιούνται για την εμφάνιση της λίστας των συσκευών, παρέχοντας λεπτομέρειες για κάθε μία από αυτές, καθώς και για την προβολή στατιστικών ή ρυθμίσεων. Επιπλέον, τα πρότυπα προσαρμόζονται ανάλογα με τα δεδομένα που αντλούνται από το μοντέλο, ώστε να παρέχουν ενημερωμένες πληροφορίες, όπως αποτελέσματα εκτέλεσης εντολών ή την κατάσταση μιας συσκευής.

Οι προβολές της εφαρμογής (views.py) είναι συναρτήσεις Python που συνδέουν τα αιτήματα των χρηστών με τα δεδομένα και την παρουσίαση. Όταν ένας χρήστης κάνει μια ενέργεια, όπως το να ζητήσει την προβολή μιας λίστας συσκευών, η αντίστοιχη προβολή λαμβάνει το αίτημα, επικοινωνεί με το μοντέλο για να ανακτήσει τα δεδομένα και στη συνέχεια τα περνάει στο κατάλληλο πρότυπο. Όλες οι λειτουργικές δυνατότητες της εφαρμογής μας υλοποιούνται μέσα στη σελίδα views.py. Το πρότυπο δημιουργεί τη σελίδα HTML που θα επιστραφεί στον χρήστη, περιέχοντας όλες τις πληροφορίες που ζήτησε. Για παράδειγμα, αν ένας χρήστης θέλει να δει τα στατιστικά μιας συγκεκριμένης συσκευής, η προβολή θα αντλήσει τα απαραίτητα δεδομένα από το μοντέλο και θα τα επιστρέψει στον χρήστη μέσα από ένα προσαρμοσμένο πρότυπο.

Με τον τρόπο αυτό, το μοτίβο MVC εξασφαλίζει την αποτελεσματική λειτουργία της εφαρμογής, διαχωρίζοντας τις ευθύνες ανάμεσα στη διαχείριση των δεδομένων, την παρουσίαση τους και την επικοινωνία με τον χρήστη. Η οργάνωση αυτή καθιστά την εφαρμογή ευέλικτη και εύκολη στη συντήρηση, ενώ παράλληλα βελτιώνει την εμπειρία χρήσης, παρέχοντας ένα λειτουργικό και φιλικό περιβάλλον διαχείρισης δικτυακών συσκευών. Στο σχήμα 5.2 παρουσιάζεται διάγραμμα απεικόνισης της βασικής αρχιτεκτονικής της εφαρμογής.



Σχήμα 5.2. Διαγραμματική απεικόνιση της εφαρμογής

5.3 Εικονικό Περιβάλλον Δικτύου GNS3–Testbed

Το τοπικό περιβάλλον ανάπτυξης πάνω στο οποίο δοκιμάστηκε η εφαρμογή είναι Linux, Ubuntu 22.04.2 LTS η οποία εικονοποιήθηκε πάνω σε λειτουργικό Windows ως WSL2(14). Το Windows Subsystem for Linux version 2(14) είναι μια τεχνολογία της Microsoft που επιτρέπει στους χρήστες Windows να τρέχουν Linux περιβάλλοντα απευθείας στο λειτουργικό σύστημα Windows, χωρίς την ανάγκη για εξομοιωτές ή εικονικές μηχανές. Είναι η δεύτερη έκδοση

του Windows Subsystem for Linux και αποτελεί σημαντική βελτίωση σε σχέση με την πρώτη έκδοση WSL1. Το WSL 2 χρησιμοποιεί την τεχνολογία εικονικοποίησης (virtualization) για να τρέχει έναν πραγματικό πυρήνα Linux μέσα σε μια ελαφριά εικονική μηχανή βοηθητικών λειτουργιών (utility VM). Αυτό επιτρέπει στο WSL 2 να προσφέρει καλύτερη απόδοση, πλήρη συμβατότητα με τις λειτουργίες Linux και πρόσβαση σε εργαλεία όπως το Docker. Η εγκατάσταση του WSL2 περιγράφεται αναλυτικά στο άρθρο Windows Subsystem for Linux 2 (WSL2): The Complete Tutorial for Windows 10 and 11 (15)

Το WSL2 αξιοποιήθηκε διότι παρέχει τη δυνατότητα λειτουργίας ενός περιβάλλοντος Linux μέσα σε έναν υπολογιστή με λειτουργικό σύστημα Windows, χωρίς να απαιτείται η χρήση ενός Type B Hypervisor. Αυτό καθιστά τη διαδικασία πιο ελαφριά και αποδοτική, επιτρέποντας τη δημιουργία και εκτέλεση του Django server και τη διαχείριση των απαιτήσεων που περιλαμβάνονται στο αρχείο requirements.txt με έναν πιο οργανωμένο και σταθερό τρόπο.

Το αρχείο requirements.txt(16) είναι ένα αρχείο το οποίο περιέχει τις βιβλιοθήκες της Python που θα χρησιμοποιήσουμε στην εφαρμογή μας. Είναι ένα text αρχείο το οποίο περιέχει όλα εκείνα τα στοιχεία του λογισμικού τα οποία είναι απαραίτητα προκειμένου να τρέξει η εφαρμογή μας. Με τη βοήθεια του package manager της Python το pip εγκαθίστανται εύκολα με μία εντολή όλες τις βιβλιοθήκες που περιγράφονται μέσα σε αυτό το text αρχείο. Με την εντολή αυτή pip install -r requirements.txt εγκαθισταται αυτόματα όλο το απαραίτητο λογισμικό για την εκτέλεση της εφαρμογής.

Η εξοικείωσή με περιβάλλοντα Unix αποτέλεσε καθοριστικό παράγοντα για την επιλογή δημιουργίας αυτού του περιβάλλοντος, καθώς το WSL2 προσφέρει μια εμπειρία χρήσης που μοιάζει πολύ με αυτή ενός πλήρους Linux συστήματος. Αυτή η προσέγγιση επέτρεψε τη χρήση γνωστών εργαλείων και πρακτικών ανάπτυξης, ενώ ταυτόχρονα αξιοποίησε τη συμβατότητα του Windows οικοσυστήματος το οποίο χρειάστηκε για την εγκατάσταση του GNS3. Έτσι, επιτεύχθηκε η δημιουργία ενός τοπικού περιβάλλοντος ανάπτυξης που συνδυάζει την ευελιξία και τη σταθερότητα των Unix συστημάτων με την ευκολία πρόσβασης που παρέχει το λειτουργικό σύστημα Windows.

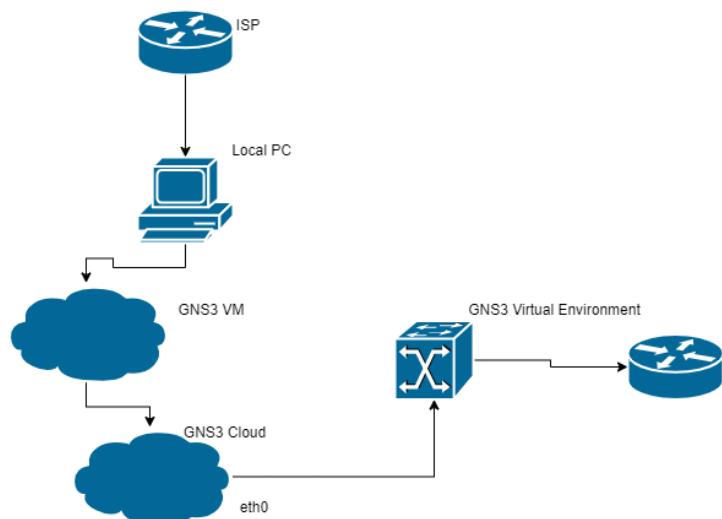
Το γεγονός ότι το WSL2 αποτελεί ένα Linux περιβάλλον που εκτελείται πάνω σε λειτουργικό σύστημα Windows είχε καθοριστική σημασία για την παρούσα υλοποίηση. Αυτό οφείλεται στο ότι τόσο το GNS3 όσο και το VirtualBox εγκαταστάθηκαν στο Windows περιβάλλον, ενώ το WSL2 αξιοποιήθηκε αποκλειστικά για την ανάπτυξη της εφαρμογής Django. Αυτή η διάκριση στη χρήση των εργαλείων επέτρεψε την ομαλή ενσωμάτωση του Linux περιβάλλοντος στο συνολικό σύστημα, διατηρώντας παράλληλα τη συμβατότητα με τις απαιτήσεις της προσδομοίωσης δικτύου.

5.3.1 Σχεδίαση Τοπολογίας Δικτύου

Η ανάπτυξη του Testbed βασίστηκε στη λογική ότι οι συσκευές που εκτελούνται στο GNS3 VM θα πρέπει να μπορούν να αλληλεπιδρούν με το τοπικό περιβάλλον WSL2(17). Για να επιτευχθεί αυτός ο στόχος, ακολουθήθηκε μια συγκεκριμένη μεθοδολογία, η οποία περιλαμβάνει διαδοχικά βήματα. Το πρώτο και θεμελιώδες βήμα ήταν η εγκατάσταση των απαραίτητων

λογισμικών εργαλείων, τα οποία έχουν ήδη παρουσιαστεί στο θεωρητικό υπόβαθρο. Η διαδικασία δημιουργίας του testbed αποδείχθηκε ιδιαίτερα απαιτητική, γι' αυτό και θα εξηγήσουμε ορισμένες βασικές αρχές που διέπουν τη λειτουργία του, ώστε να γίνει πιο κατανοητή. Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, έχουμε ήδη εγκαταστήσει το GNS3, το GNS3 VM στο VirtualBox, και πλέον είμαστε έτοιμοι να διαμορφώσουμε το περιβάλλον όπου οι συσκευές της Cisco, που εκτελούνται στο GNS3 VM, θα μπορούν να επικοινωνούν με το WSL2, το οποίο αποτελεί και το βασικό λειτουργικό σύστημα του πειραματικού μας περιβάλλοντος.

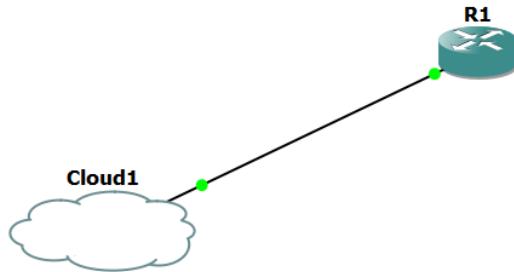
Λόγω των περιορισμένων υπολογιστικών πόρων που διατίθενται στο πλαίσιο αυτής της διπλωματικής εργασίας, η τοπολογία που χρησιμοποιήθηκε είναι απλή, όπως φαίνεται και στο Σχήμα 5.3. Συγκεκριμένα, η δομή του testbed περιλαμβάνει πρώτον έναν κόμβο Cloud, ο οποίος επιτρέπει τη δικτυακή αλληλεπίδραση μεταξύ των συσκευών του GNS3 VM και του WSL2 και μία συσκευή δοκιμών, που χρησιμοποιείται για την αξιολόγηση της συνδεσιμότητας και της επικοινωνίας στο προτεινόμενο περιβάλλον. Επιλέχθηκε αυτή η τοπολογία διότι με το ελάχιστο δυνατό υπολογιστικό κόστος παρέχεται το απαραίτητο περιβάλλον δοκιμών. Με αυτήν την προσέγγιση, διασφαλίζεται η σωστή λειτουργία του testbed, καθιστώντας εφικτή τη μελέτη και την ανάλυση της επικοινωνίας μεταξύ των εικονικών και των φυσικών συσκευών.



Σχήμα 5.3. Local PC-GNS3VM-CISCO IOS Connection Architecture

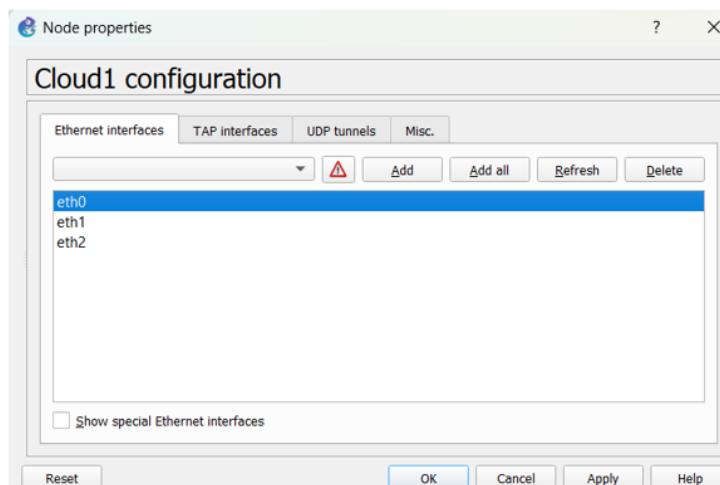
Το GNS3 cloud node μας δίνει τη δυνατότητα να συνδέσουμε τις συσκευές μας που τρέχουν στο GNS3 VM με το τοπικό μας δίκτυο. Για να γίνει αυτό ρυθμίζουμε το cloud node έτσι ώστε το eth0 interface του να δώσει ip διεύθυνση μέσα από το DHCP πρωτόκολλο στα devices μας. Με αυτόν τον τρόπο έχουμε απευθείας πρόσβαση σε αυτά μέσα από το τοπικό μας δίκτυο.

Για την παραμετροποίηση του Cloud Node, ρυθμίζουμε το interface που συνδέεται με τη συσκευή, ακολουθώντας την παρακάτω διαδικασία. Αρχικά, συνδέουμε το interface eth0 του Cloud Node με το αντίστοιχο eth0 της συσκευής, όπως φαίνεται στην 5.4 εικόνα.



Σχήμα 5.4. CLOUD NODE With CISCO Router Device Point to Point Connection

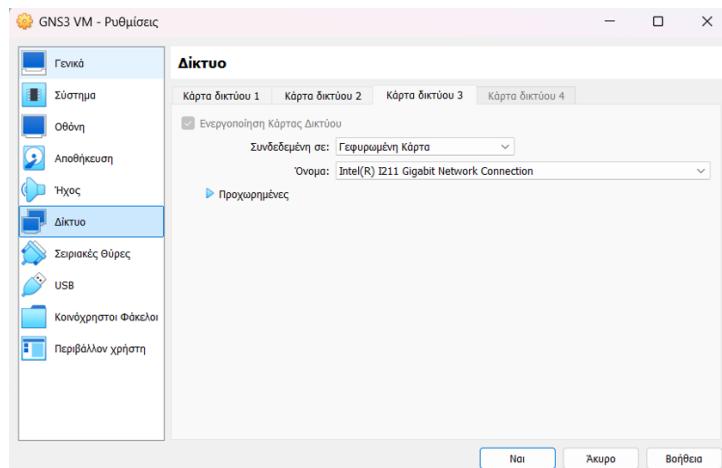
Η σύνδεση πραγματοποιείται μέσω της επιλογής του interface από το Configuration Menu του Cloud Node. Συγκεκριμένα, κάνουμε δεξί κλικ στο Cloud Node, επιλέγουμε "Παραμετροποίηση" και στη συνέχεια προσθέτουμε το eth0 ως τη διεπαφή επικοινωνίας του τοπικού δικτύου με τον διακομιστή όπως φαίνεται στην εικόνα 5.5.



Σχήμα 5.5. CLOUD NODE Network Interface Configuration

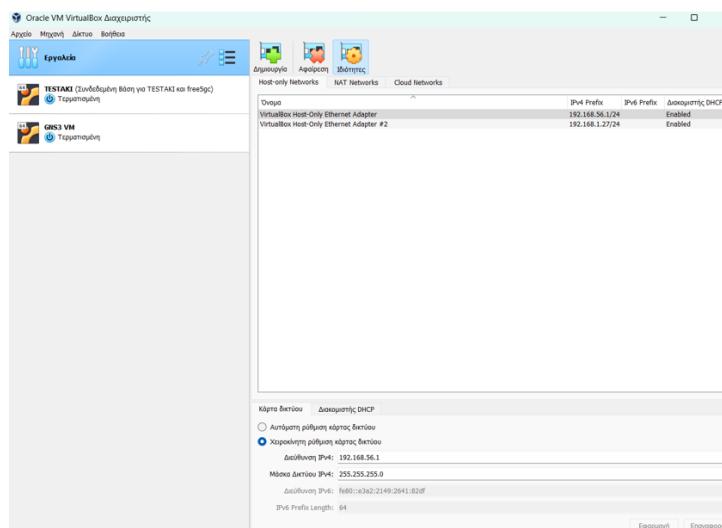
Η επιλογή του eth0 είναι καθοριστικής σημασίας, καθώς στο GNS3 VM το συγκεκριμένο interface είναι ήδη προρυθμισμένο για αυτήν τη λειτουργία, διασφαλίζοντας έτσι την ορθή επικοινωνία του τοπικού δικτύου με τον διακομιστή. Γιατί όμως επιλέγουμε το eth0 στο σχήμα 5.4 και όχι το eth1 ή το eth2.

Επιλέγουμε το eth0 στο Cloud node διότι αυτό ανήκει στο δίκτυο της Γεφυρωμένης Κάρτας Δικτύου που επιλέξαμε για το GNS3 VM. Στην εικόνα παρακάτω(Σχήμα 5.6) βλέπουμε την κάρτα δικτύου του GNS3 VM.



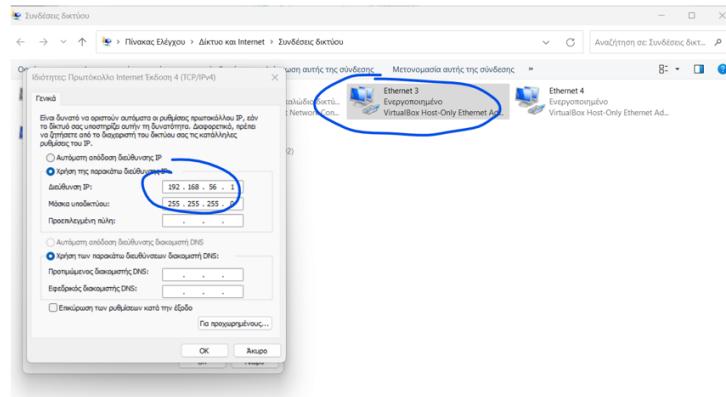
Σχήμα 5.6. GNS3 VM κάρτα δικτύου-Γεφυρωμένη κάρτα

Προκειμένου να φτιάξουμε την κάρτα δικτύου αυτή μέσα από τα εργαλεία του Vbox πιατάμε Δημιουργία Host-only Networks και ρυθμίζουμε την κάρτα δικτύου όπως φαίνεται στο σχήμα 5.7. Στο σχήμα φαίνεται ότι όλες οι συσκευές μας θα πρέπει να παίρνουν IP διεύθυνση από το δίκτυο 192.168.56.0/24



Σχήμα 5.7. Προσαρμογέας δικτύου για την εικονική μηχανή GNS3 VM

Αφού φτιάξουμε την κάρτα δικτύου του VBox αυτή μετά φαίνεται αυτόματα στις συνδέσεις δικτύου του υπολογιστή μας όπως στην εικόνα 5.8.



Σχήμα 5.8. Διεπαφή δικτύου στο περιβάλλον των Windows

Προκειμένου τώρα οι συσκευές αυτές να μπορέσουν να μιλήσουν με το τοπικό μας δίκτυο θα πρέπει να τους ορίσουμε IP διευθύνσεις. Συνεπώς, με αυτήν την παραμετροποίηση, όταν η δικτυακή συσκευή παραμετροποιηθεί ώστε να αποκτά IP διεύθυνση από το τοπικό δίκτυο, θα είναι δυνατή η επικοινωνία με τις συσκευές του δικτύου για λόγους δοκιμών. Προκειμένου να πάρουν οι συσκευές IP διεύθυνση από το τοπικό δίκτυο δηλαδή από το εικονικό interface του Vbox που δημιουργήσαμε παραπάνω(192.168.56.0/24) κάναμε την παρακάτω παραμετροποίηση. Στο σχήμα 5.9 φαίνεται η παραμετροποίηση στην δικτυακή συσκευή.

```
R1#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
R1(config)#interface fa
R1(config)#interface fastEthernet0/0
R1(config-if)#ip add
R1(config-if)#ip address dhcp
R1(config-if)#ip address 192.168.56.1 255.255.255.0
R1(config-if)#shut
R1(config-if)#no shu
R1(config-if)#no shutdown
R1(config-if)#end
R1#
R1#
```

Σχήμα 5.9. Παραμετροποίηση στη δικτυακή συσκευή

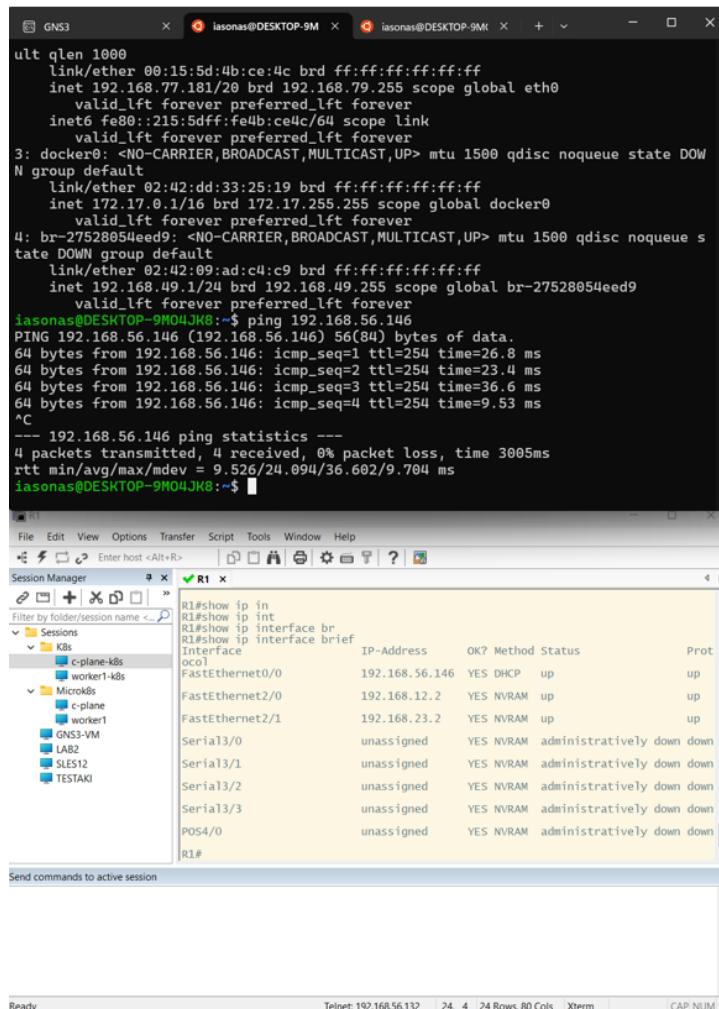
Στο σχήμα 5.10 φαίνεται η συσκευή να πάρνει σωστά IP διευθύνση από το 192.168.56.0/24.

Interface	IP-Address	OK?	Method	Status	Prot
FastEthernet0/0	192.168.56.4	YES	DHCP	up	up
R1#					

Σχήμα 5.10. Κατανομή IP διευθύνσης

Αφού το interface πήρε IP διεύθυνση ελέγχουμε στο τέλος ότι μπορούμε να επικοινωνήσουμε με τη συσκευή όπως στο σχήμα 5.11. Στα σχήματα οι IP διευθύνσεις μπορεί να είναι διαφορετικές καθώς δοκιμάστηκαν σε διαφορετική φάση. Η ουσία όμως είναι ότι όλες θα πάρνουν IP διεύθυνση από το δίκτυο 192.168.56.0/24 το οποίο θα αποτελεί και το τελικό δίκτυο πάνω στο οποίο θα αλληλεπιδρούν ο Django Server με το GNS3 testbed. Το δίκτυο αυτό είναι κοινό για όλα τα nodes που βλέπουμε στο Σχήμα 5.3 το οποίο σχεδιάστηκε

προκειμένου να γίνει κατανοητή η σχεδίαση.



The screenshot shows the GNS3 application interface. The top window is a terminal window titled 'GNS3' showing network configuration and ping results:

```

ult qlen 1000
link/ether 00:15:5d:4b:ce:4c brd ff:ff:ff:ff:ff:ff
inet 192.168.77.181/20 brd 192.168.79.255 scope global eth0
    valid_lft forever preferred_lft forever
inet6 fe80::215:5dff:fe4b:ce4c/64 scope link
    valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
N group default
    link/ether 02:42:dd:33:25:19 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
4: br-27528054eed9: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:09:ad:c4:c9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.49.1/24 brd 192.168.49.255 scope global br-27528054eed9
        valid_lft forever preferred_lft forever
jasonas@DESKTOP-9M04JK8:~$ ping 192.168.56.146
PING 192.168.56.146 (192.168.56.146) 56(84) bytes of data.
64 bytes from 192.168.56.146: icmp_seq=1 ttl=254 time=26.8 ms
64 bytes from 192.168.56.146: icmp_seq=2 ttl=254 time=23.4 ms
64 bytes from 192.168.56.146: icmp_seq=3 ttl=254 time=36.6 ms
64 bytes from 192.168.56.146: icmp_seq=4 ttl=254 time=9.53 ms
^C
--- 192.168.56.146 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 9.526/24.094/36.602/9.704 ms
jasonas@DESKTOP-9M04JK8:~$ 

```

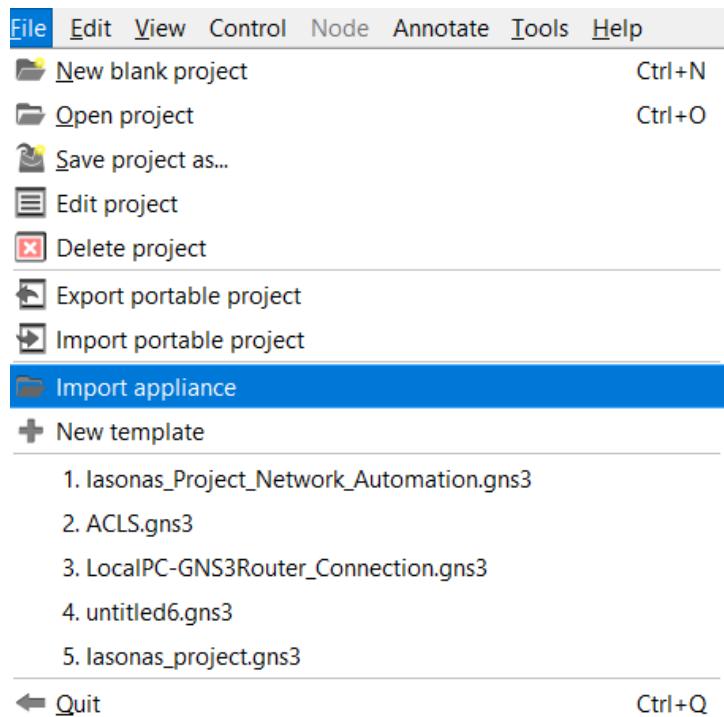
The bottom window is the 'Session Manager' titled 'R1 x'. It lists network interfaces and their configurations:

Interface	IP-Address	OK?	Method	Status	Prot
FastEthernet0/0	192.168.56.146	YES	DHCP	up	up
FastEthernet2/0	192.168.12.2	YES	NVRAM	up	up
FastEthernet2/1	192.168.23.2	YES	NVRAM	up	up
Serial1/0	unassigned	YES	NVRAM	administratively down	down
Serial1/1	unassigned	YES	NVRAM	administratively down	down
Serial1/2	unassigned	YES	NVRAM	administratively down	down
Serial1/3	unassigned	YES	NVRAM	administratively down	down
POS4/0	unassigned	YES	NVRAM	administratively down	down

Σχήμα 5.11. IP connectivity test

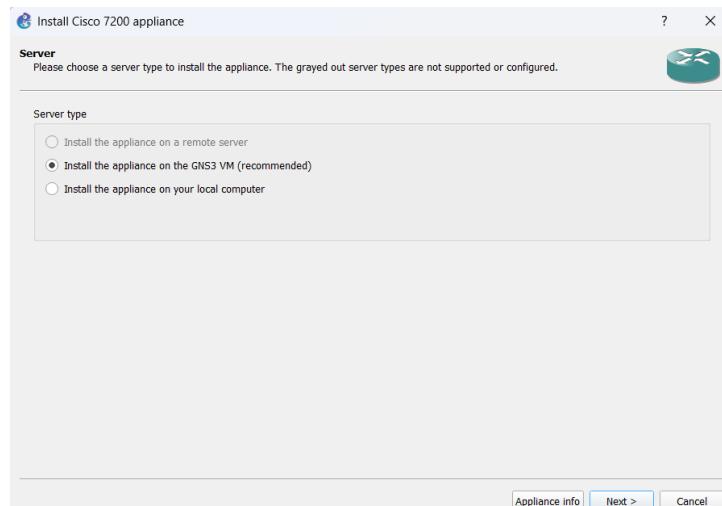
5.3.2 Προσομοίωση Συσκευών Cisco

Προκειμένου να προσομοιώσουμε συσκευές της Cisco το πρώτο βήμα είναι να κατεβάσουμε συγκεκριμένο appliance από το GNS3 marketplace. Αφού το κατεβάσουμε το εισάγουμε στο GNS3 με τον εξής τρόπο: (εικόνα 5.12)



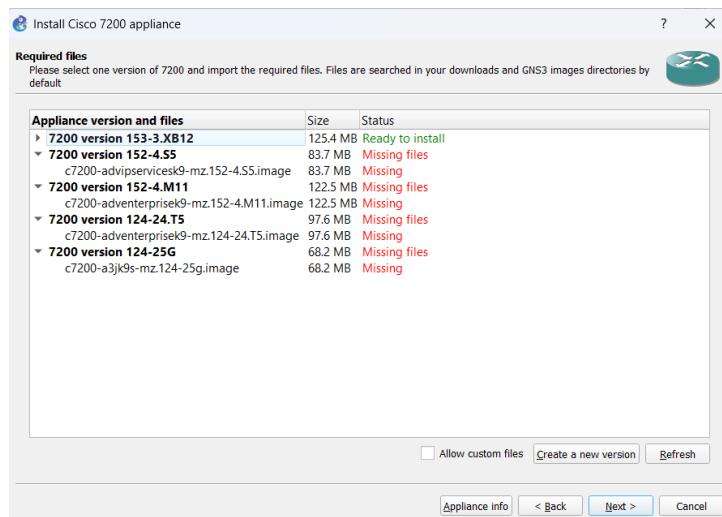
Σχήμα 5.12. Import appliance

Ακολούθως πατάμε Import appliance και επιλέγουμε το αρχείο που κατεβάσαμε. Μετά επιλέγουμε Install appliance on the GNS3 VM όπως φαίνεται στο σχήμα 5.13.



Σχήμα 5.13. Install appliance

Η επιλογή αυτή θα μας μεταφέρει στο σχήμα 5.14.

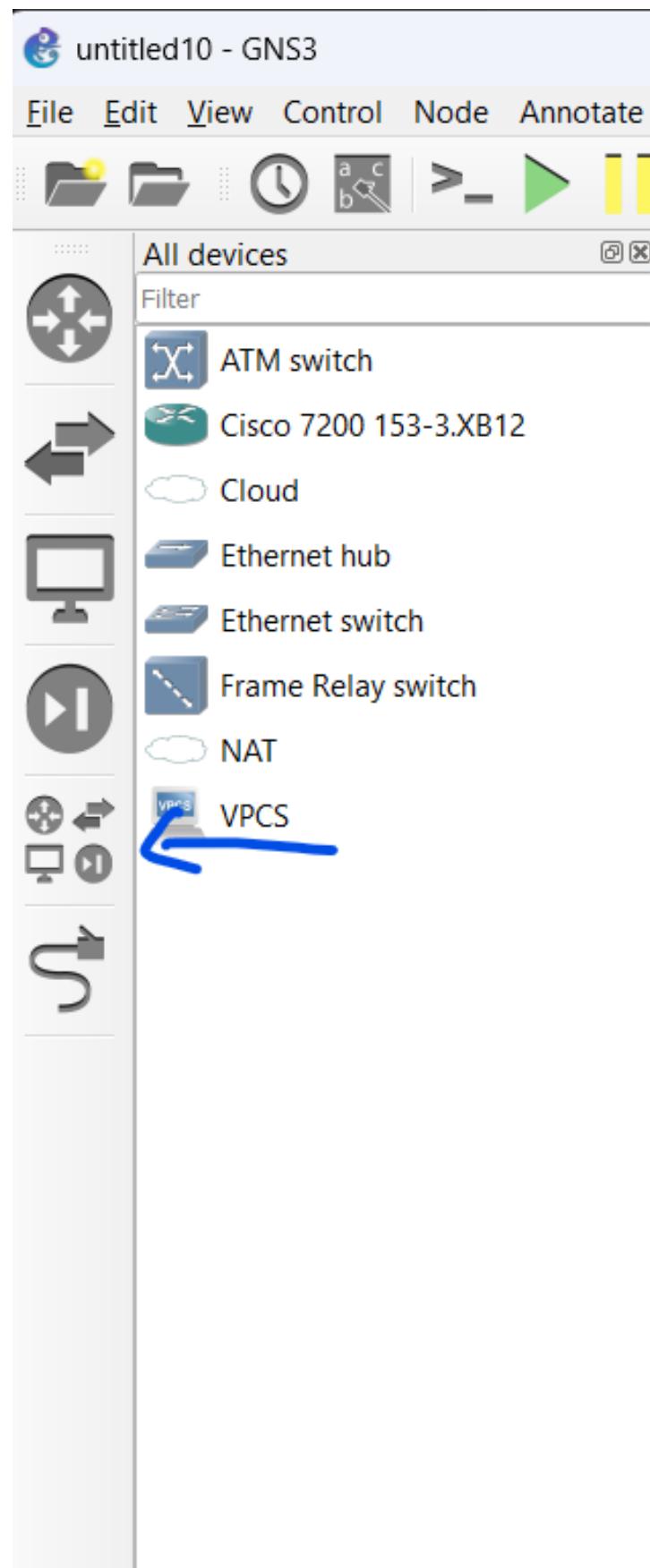
**Σχήμα 5.14. Appliance**

Στη συνέχεια, αρκεί να αντιστοιχιστεί το filename του appliance με το image που διαθέτουμε, προκειμένου να εισαχθεί η συσκευή. Για να εντοπιστεί το filename στο appliance, ανοίγεται το αρχείο με énava editor και το filename τροποποιείται αναλόγως, όπως φαίνεται στην εικόνα 5.15(18).

```
"images": [
    {
        "filename": "c7200-adventerprisek9-mz.153-3.XB12.image",
        "version": "153-3.XB12",
        "md5sum": "3d234a3793331c972776354531f87221",
        "filesize": 131471340
    },
]
```

Σχήμα 5.15. filename configuration

Μετά το τέλος της διαδικασίας η συσκευή θα έχει προστεθεί και μπορούμε να την δούμε στην επιλογή **Browse all devices**(σχήμα 5.16). Συνεπώς θα μπορούμε να την προσθέσουμε σε τοπολογία και να την κάνουμε να δουλέψει. Για να μπορέσει η εφαρμογή μας να επικοινωνήσει με οποιαδήποτε συσκευή προστίθεται μέσα στο εικονικό περιβάλλον δοκιμών (αλλά και πραγματική δικτυακή συσκευή που συνδέεται στο δίκτυό μας) θα πρέπει να έχουν ρυθμιστεί ορισμένες παράμετροι σε αυτή τη συσκευή. Συγκεκριμένα στην παραπάνω συσκευή Cisco και μέσα στο περιβάλλον διαχείρισής της, πρέπει να ρυθμιστούν το username, password, secret καθώς και να ενεργοποιηθεί το ssh service προκειμένου να είναι δυνατή η σύνδεση στην συσκευή μέσω του API της εφαρμογής μας.



Σχήμα 5.16. Browse Devices

Στην εικόνα 5.17 φαίνεται τι πρέπει να έχει υλοποιηθεί ως προυπόθεση στην εφαρμογή. Φυσικά θεωρούμε δεδομένο ότι έχουν γίνει όλα τα προηγούμενα βήματα καθώς το παρακάτω αφορά μόνο το χρήστη και τον κωδικό του προκειμένου να υλοποιήσει την SSH σύνδεση.

```
!
username iasonas password 0 ericsson
!
redundancy
!
!
ip tcp synwait-time 5
ip ssh version 2
```

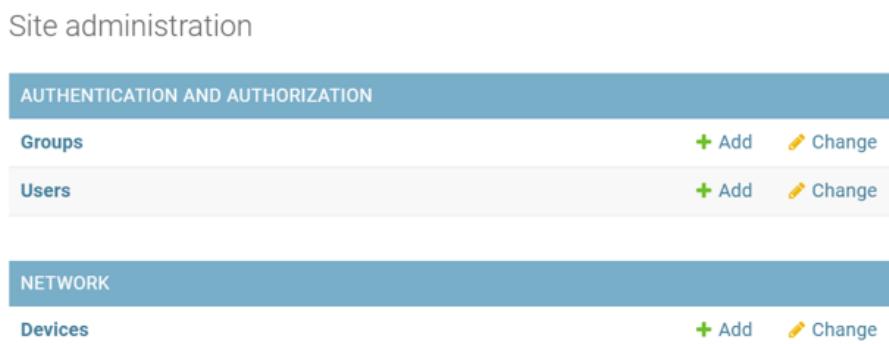
Σχήμα 5.17. ssh and credentials requirements

6 Υλοποίηση της εφαρμογής

6.1 Ανάπτυξη με το Django framework

6.1.1 Δημιουργία Models

Η διαδικτυακή πύλη χρησιμοποιεί μια βάση δεδομένων SQLite. Αυτή η βάση δεδομένων περιέχει τρία μοντέλα τα οποία ορίζονται στο αρχείο `models.py`. Στην εικόνα 6.1 φαίνονται τα μοντέλα αυτά.



Σχήμα 6.1. Μοντέλα συσκευών

Η διαδικτυακή πύλη περιέχει την κλάση `Device`, η οποία δέχεται ως ορίσματα το όνομα, τη διεύθυνση IP, το όνομα χρήστη, τον κωδικό πρόσβασης, τον κρυφό κωδικό και το μοντέλο της συσκευής. Για κάθε μία από αυτές τις συσκευές που δημιουργούμε, καταγράφεται μια αντίστοιχη εγγραφή στη βάση δεδομένων. Το αντικείμενο που δημιουργείται δηλαδή η συσκευή προς δοκιμή περιλαμβάνει πεδία που αξιοποιούνται από τις συναρτήσεις του αρχείου `views.py`. Με αυτόν τον τρόπο, τα πεδία του αντικειμένου αποθηκεύονται και χρησιμοποιούνται ως ορίσματα μέσα στις συναρτήσεις, διευκολύνοντας τη διαχείριση και την επεξεργασία των δεδομένων της συσκευής.

6.1.2 Views και Urls

Τα αρχεία `Urls` καθορίζουν τη δρομολόγηση των URL της εφαρμογής. Οι διευθύνσεις URL που αντιστοιχούν στα μοτίβα που περιγράφονται στο αρχείο `urls.py` προωθούνται στην αντίστοιχη συνάρτηση στο αρχείο `views.py`. Η αντιστοίχιση πραγματοποιείται σειριακά από πάνω προς τα κάτω στο αρχείο `urls.py` όπως φαίνεται στη εικόνα 6.2.

```
urlpatterns = [
    path('', views.firstPage),
    path('manage/',views.index, name="manage"),
    path('manage2/',views.index2, name="manage2"),
    path('manage3/',views.index3, name="manage3"),
    path('device_statistics/<int:device_id>', views.get_interface_statistics, name="device_statistics"),
    path('interface_statistics/<int:device_id>', views.get_interfaces_counters, name="interface_statistics"),
    path('device/<int:device_id>', views.get_device_stats, name="device"),
    path('execute_script/', views.execute_script_on_remote, name='execute_script'),
    path('manage4/',views.index4, name="manage4"),
    path('manage5/',views.index5, name="manage5"),
    path('running_config/<int:device_id>', views.get_running_config, name="running_config"),
    path('show_running_config/<int:device_id>/', views.show_running_config, name='show_running_config'),
]
```

Σχήμα 6.2. Αρχείο urls.py

Παρόλο που σε αυτό το έργο υλοποιήθηκε ακριβής αντιστοίχιση των URL, το Django παρέχει τη δυνατότητα χρήσης ταυτοποίησης μέσω κανονικών εκφράσεων. Στην εικόνα που ακολουθεί, παρουσιάζονται οι διεύθυνσεις URL του API, όπου κάθε διεύθυνση αντιστοιχεί σε μια συνάρτηση στο αρχείο api1/views.py, η οποία καταλήγει στην εκτέλεση του αντίστοιχου σεναρίου

Οι συναρτήσεις σε ένα αρχείο views.py καλούνται όταν η δεδομένη διεύθυνση URL που αποστέλλεται από το χρήστη ταιριάζει με το αντίστοιχο μοτίβο URL στο αρχείο urls.py. Παράμετροι που αποστέλλονται μέσω κλήσης HTTP εισέρχονται στην αντίστοιχη συνάρτηση μέσω παραμέτρων ή σώματος αίτησης. Στο αρχείο views.py του API, η συνάρτηση εκτελεί το σενάριο κώδικα με τις δεδομένες παραμέτρους εισόδου. Όταν τελειώσει η εκτέλεση του κώδικα δέσμης ενεργειών, το αποτέλεσμα επιστρέφεται στη συνάρτηση views και μεταβιβάζεται ως πλαίσιο στο αντίστοιχο αρχείο .html για την εμφάνιση των αποτελεσμάτων στον χρήστη που εκτέλεσε το σενάριο. Ένα παράδειγμα τριών διαφορετικών συνάρτησεων σε ένα αρχείο views.py μπορείτε να δείτε στο σχήμα 6.3.

```

def index3(request: HttpRequest) -> HttpResponse:
    devices = Device.objects.all()
    context = {
        'title': 'Interface Statistics',
        'devices': devices
    }
    return render(request, 'index3.html', context)

def index4(request: HttpRequest) -> HttpResponse:
    devices = Device.objects.all()
    context = {
        'title': 'Backup running config',
        'devices': devices
    }
    return render(request, 'index4.html', context)

def index5(request: HttpRequest) -> HttpResponse:
    devices = Device.objects.all()
    context = {
        'title': 'Backup running config',
        'devices': devices
    }
    return render(request, 'index5.html', context)

```

Σχήμα 6.3. Αρχείο views.py

6.1.3 User Interface (Templates)

Σε κάθε συνάρτηση του αρχείου views.py, το Django αποδίδει το αντίστοιχο πρότυπο .html αρχείο με ένα συγκεκριμένο πλαίσιο. Η λογική είναι ότι κάθε HTTP Request που δέχεται όταν ο χρήστης επιδρά με την εφαρμογή απαντιέται με ένα αντίστοιχο HTTP Response. Τα δεδομένα στο πλαίσιο εμφανίζονται στο .html μέσα από το HTTP Response, εάν το .html έχει παραμετροποιηθεί κατάλληλα. Ένα παράδειγμα .html με τον συντακτικό κώδικα για τον τρόπο πρόσβασης στα δεδομένα του πλαισίου παρουσιάζεται στην εικόνα 6.4.

```

</style>
</head>
<body>
  <div class="container">
    <h1>{{ title }}</h1>
    <h2>Devices</h2>
    <table>
      <tr>
        <th>Id</th>
        <th>Name</th>
        <th>Host</th>
      </tr>
      <tr>
        <td>result</td>
        <td><a href="{% url 'device' device.id %}">{{ device.name }}</a></td>
        <td>{{ device.host }}</td>
      </tr>
    </table>
  </div>
</body>
</html>

```

Σχήμα 6.4. Παράδειγμα html αρχείου

Έτσι όλη η διαθέσιμη λειτουργικότητα που παρέχεται στον χρήστη της εφαρμογής διαχείρισης δικτύου υλοποιείται με την χρήση URLs (κλήσεις συναρτήσεων και html αρχείων). Έτσι για παράδειγμα για την λειτουργία Statistics :

- Ο χρήστης επισκέπτεται το URL /manage2.
- Ο ορισμός του URL στο urls.py κατευθύνει το αίτημα στη συνάρτηση index2 στο views.py.
- Με αυτό τον τρόπο ο χρήστης κάνει ένα HTTP Request. Η συνάρτηση index2 συλλέγει τα δεδομένα (context-data) που χρειάζονται για την προβολή.
- Η συνάρτηση index2 επιστρέφει τα δεδομένα στη σελίδα index2.html επιστρέφοντας το μέσα από ένα HTTP Response.
- Το template index2.html λαμβάνει τα δεδομένα και τα εμφανίζει στον χρήστη.

6.2 REST API Integration και διασύνδεση με το SSH protocol

To REST API Integration διαδραματίζει κρίσιμο ρόλο στη διπλωματική εργασία, καθώς επιτρέπει την αλληλεπίδραση μεταξύ των διαφόρων συστημάτων και την ανταλλαγή δεδομένων μέσω του διαδικτύου. To REST (Representational State Transfer) είναι μια αρχιτεκτονική προσέγγιση που χρησιμοποιεί τα πρωτόκολλα HTTP για την εκτέλεση λειτουργιών όπως η ανάκτηση, η δημιουργία, η ενημέρωση και η διαγραφή δεδομένων. Με την υλοποίηση REST APIs, καθίσταται δυνατή η διαλειτουργικότητα της εφαρμογής με άλλες υπηρεσίες, διευκολύνοντας τη σύνθεση πολύπλοκων λειτουργιών από διάφορες πηγές. Στο πλαίσιο της διπλωματικής, το REST API χρησιμοποιείται για την διαχείριση δικτυακών συσκευών, παρακολούθηση στατιστικών, εκτέλεση εντολών, διασφαλίζοντας την ασφάλεια και την αποδοτικότητα της επικοινωνίας μέσω διαφανών και επεκτάσιμων τεχνολογιών. Το REST δεν

χρησιμοποιείται για την επικοινωνία της εφαρμογής και των συσκευών αλλά εσωτερικά ως μηχανισμός δρομολόγησης προκειμένου η εφαρμογή να μπορεί να φέρει στον χρήστη το σωστό αποτέλεσμα. Η διαχείρηση των δικτυακών συσκευών γίνεται με το πρωτόκολλο SSH.

Προσπαθώντας μέσα από το wireshark να πιάσουμε την κίνηση μεταξύ WSL και δικτυακής συσκευής κάναμε το παρακάτω πείραμα προκειμένου να κατανοήσουμε πως επικοινωνεί η εφαρμογή με τις συσκευές. Πατάμε το κουμπί R1 Testing όπως φαίνεται στο σχήμα 6.5. και στο Wireshark κάνουμε capture το interface eth0 του WSL προκειμένου να δούμε τι γίνεται όταν εμείς πατάμε το κουμπί για μια λειτουργία. Τα κουμπιά έχουν σχεδιαστεί έτσι ώστε, με το πάτημά τους, να σε κατευθύνουν μέσω συνδέσμου (link) στη λειτουργία που επιθυμείς να εκτελέσεις.

CONTROLLER		
Device Interfaces		
Id	Name	Host
1	R1	192.168.2.9
2	R4	192.168.56.152
3	Router_Ericsson	192.168.56.120
4	R2_Ericsson	192.168.2.15
5	R3	192.168.56.123
6	R1_testing	192.168.56.146

Σχήμα 6.5. Παράδειγμα wireshark

Capturing from vEthernet (WSL (Hyper-V firewall))						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.77.181	192.168.56.146	TCP	74	45479 → 22 [SYN] Seq=0 Win=65340 Len=0 MSS=1452 SACK_PERM=1 TSval=4003428849 TSecr=0 WS=128
2	0.023672	192.168.56.146	192.168.77.181	TCP	58	22 → 45479 [SYN, ACK] Seq=9 Ack=1 Win=4128 Len=0 MSS=1452
3	0.023024	192.168.77.181	192.168.56.146	TCP	54	45470 → 22 [ACK] Seq=1 Ack=1 Win=65340 Len=0
4	0.023199	192.168.77.181	192.168.56.146	SSHv2	78	Client: Protocol (SSH-2.0-paramiko_3.5.0)
5	0.067436	192.168.56.146	192.168.77.181	TCP	54	22 → 45479 [ACK] Seq=1 Ack=25 Win=4104 Len=0
6	0.067536	192.168.56.146	192.168.77.181	SSHv2	73	Server: Protocol (SSH-2.0-Cisco_3.5.0)
7	0.067714	192.168.77.181	192.168.56.146	TCP	54	45470 → 22 [ACK] Seq=25 Ack=20 Win=65321 Len=0
8	0.068266	192.168.77.181	192.168.56.146	SSHv2	1342	Client: Key Exchange Init
9	0.086762	192.168.77.181	192.168.56.146	SSHv2	398	Server: Key Exchange Init
10	0.087279	192.168.77.181	192.168.56.146	SSHv2	78	Client: Diffie-Hellman Group Exchange Request
11	0.119804	192.168.77.181	192.168.56.146	SSHv2	334	Server: Diffie-Hellman Group Exchange Group
12	0.136551	192.168.77.181	192.168.56.146	SSHv2	326	Client: Diffie-Hellman Group Exchange Init
13	0.290326	192.168.77.181	192.168.56.146	TCP	54	22 → 45479 [ACK] Seq=6444 Ack=6409 Win=4128 Len=0
14	0.555699	192.168.77.181	192.168.56.146	TCP	614	22 → 45479 [ACK] Seq=6444 Ack=6409 Win=4128 Len=560 [TCP segment of a reassembled PDU]
15	0.555895	192.168.77.181	192.168.56.146	SSHv2	70	Server: Diffie-Hellman Group Exchange Reply
16	0.555831	192.168.77.181	192.168.56.146	SSHv2	70	Server: New Keys
17	0.556023	192.168.77.181	192.168.56.146	TCP	54	45470 → 22 [ACK] Seq=1609 Ack=1236 Win=64105 Len=0
18	0.556344	192.168.77.181	192.168.56.146	SSHv2	76	Client: New Keys
19	0.582335	192.168.77.181	192.168.56.146	TCP	54	22 → 45479 [ACK] Seq=1236 Ack=1625 Win=4112 Len=0
20	0.809597	192.168.77.181	192.168.56.146	SSHv2	186	Client: Encrypted packet (len=52)
21	0.843036	192.168.77.181	192.168.56.146	SSHv2	106	Server: Encrypted packet (len=52)
22	0.844374	192.168.77.181	192.168.56.146	SSHv2	138	Client: Encrypted packet (len=84)
23	0.877768	192.168.77.181	192.168.56.146	SSHv2	99	Server: Encrypted packet (len=36)
24	0.878742	192.168.77.181	192.168.56.146	SSHv2	122	Client: Encrypted packet (len=68)
25	0.926645	192.168.77.181	192.168.56.146	SSHv2	106	Server: Encrypted packet (len=52)
26	0.921334	192.168.77.181	192.168.56.146	SSHv2	138	Client: Encrypted packet (len=84)
27	0.956454	192.168.77.181	192.168.56.146	SSHv2	90	Server: Encrypted packet (len=36)
28	0.951016	192.168.77.181	192.168.56.146	SSHv2	106	Client: Encrypted packet (len=52)
29	0.993458	192.168.77.181	192.168.56.146	SSHv2	99	Server: Encrypted packet (len=36)
30	0.993550	192.168.77.181	192.168.56.146	SSHv2	106	Client: Encrypted packet (len=52)
31	0.994082	192.168.77.181	192.168.56.146	TCP	54	45470 → 22 [ACK] Seq=1965 Ack=1500 Win=64053 Len=0
32	0.994136	192.168.77.181	192.168.56.146	SSHv2	106	Client: Encrypted packet (len=52)
33	1.005846	192.168.77.181	192.168.56.146	SSHv2	106	Server: Encrypted packet (len=52)

Σχήμα 6.6. Παράδειγμα wireshark

Αυτή η ακολουθία του σχήματος 6.6 αντιπροσωπεύει μια τυπική, ασφαλή σύνδεση SSH. Στο αρχείο `pcap` που καταγράψαμε, αποτυπώνεται η διαδικασία εγκαθίδρυσης μιας σύνδεσης TCP (SYN, SYN/ACK, ACK). Συγκεκριμένα, στο πρώτο πακέτο παρατηρούμε το SYN, το οποίο αποστέλλεται προς τη συσκευή. Στη συνέχεια, η συσκευή αποκρίνεται με SYN/ACK, επιβεβαιώνοντας την επικοινωνία. Τέλος, στο τρίτο πακέτο, απαντάμε με ACK, ολοκληρώνοντας έτσι τη διαδικασία σύναψης της σύνδεσης.

Μετά ακολουθεί η διαπραγμάτευση πρωτοκόλλου SSH και στο τέλος η έναρξη της κρυπτογραφημένης επικοινωνίας την οποία δε μπορούμε να δούμε. Όμως μπορούμε να δούμε τι επιστρέφεται από το cli του Django server στην εικόνα 6.7.

Σχήμα 6.7. Εμφάνιση απάντησης συσκευής τυπωμένη στο τερματικό

7 Επίδειξη της εφαρμογής(Application Demo)

7.1 Εισαγωγή-Η λογική της λειτουργίας της εφαρμογής Django

Όταν τρέχουμε τον διακομιστή της εφαρμογής η εφαρμογή ιστού είναι διαθέσιμη από οποιοδήποτε φυλλομετρητή. Η αρχική σελίδα στη εφαρμογή Django δηλώνεται στο πλαίσιο του Django run server ώς μία συνάρτηση της Python η οποία δέχεται ένα http request και σαν απάντηση επιστρέφει την αρχική σελίδα(index.html).

Αυτό γίνεται μέσα από δύο κύριως μηχανισμούς. Η συνάρτηση firstPage ορίζεται στο αρχείο views.py της εφαρμογής. Πλέονει ως όρισμα ένα αντικείμενο HttpResponseRedirect (την αίτηση του χρήστη) και επιστρέφει ένα HttpResponseRedirect μέσω της render(σχήμα 7.1):

```
def firstPage(request: HttpRequest) -> HttpResponseRedirect:
    return render(request, 'index.html')
```

Σχήμα 7.1. Συνάρτηση firstPage στο αρχείο views.py

Η λίστα urlpatterns συνδέει τη διεύθυνση URL με τη συνάρτηση firstPage. Το κενό string ('') σημαίνει ότι αυτή η διαδρομή αντιστοιχεί στη ρίζα του ιστότοπου (π.χ., http://localhost:8000/). Όταν ένας χρήστης επισκέπτεται τη ρίζα, η συνάρτηση firstPage εκτελείται και επιστρέφει την HTML σελίδα index.html. Στο σχήμα 7.2 το σημείο στο urls.py όπου γίνεται η δρομολόγηση.

```
urlpatterns = [
    path('', views.firstPage),
```

Σχήμα 7.2. Urls δρομολόγηση για την αρχική σελίδα.

Μέσα από αυτή την αρχική σελίδα μπορούμε να κατευθυνθούμε σε οποιαδήποτε επιλογή εμείς θέλουμε. Στο Django καθοριστικός παράγοντας στην ευκολία με την οποία μπορείς να κτίσεις μια εφαρμογή από την αρχή είναι ο τρόπος που χειρίζεται τη δρομολόγηση των διαφόρων σελίδων. Αυτό γίνεται μέσα από το urls.py

Αυτή η ρύθμιση καθορίζει πώς θα δρομολογούνται οι αιτήσεις HTTP προς συγκεκριμένες λειτουργίες (views) στην εφαρμογή Django. Με τη χρήση των δυναμικών παραμέτρων, μπορούμε συνεπώς να δημιουργήσουμε πιο ευέλικτες διαδρομές URL που μπορούν να χειρίζονται ποικίλες καταστάσεις.

Κάθε ένα από αυτά τα paths που φαίνονται και στην εικόνα 7.3 είναι υπεύθυνα για την ανακατεύθυνση των διευθύνσεων και τη σωστή δρομολόγησή τους ώστε να δώσουν σαν απάντηση κάθε φορά τα σωστά δεδομένα.

```
urlpatterns = [
    path('', views.firstPage),
    path('configure-ip/', views.configure_ip, name='configure_ip'),
    path('manage/', views.index, name='manage'),
    path('manage2/', views.index2, name='manage2'),
    path('manage3/', views.index3, name='manage3'),
    path('device_statistics/<int:device_id>', views.get_interface_statistics, name="device_statistics"),
    path('interface_statistics/<int:device_id>', views.get_interfaces_counters, name="interface_statistics"),
    path('device/<int:device_id>', views.get_device_stats, name="device"),
    path('execute_script/', views.execute_script_on_remote, name='execute_script'),
    path('manage4/', views.index4, name='manage4'),
    path('manage5/', views.index5, name='manage5'),
    path('running_config/<int:device_id>', views.get_running_config, name="running_config"),
    path('show_running_config/<int:device_id>', views.show_running_config, name='show_running_config'),
```

Σχήμα 7.3. Urls.py αρχείο

7.2 Η αρχική σελίδα της εφαρμογής

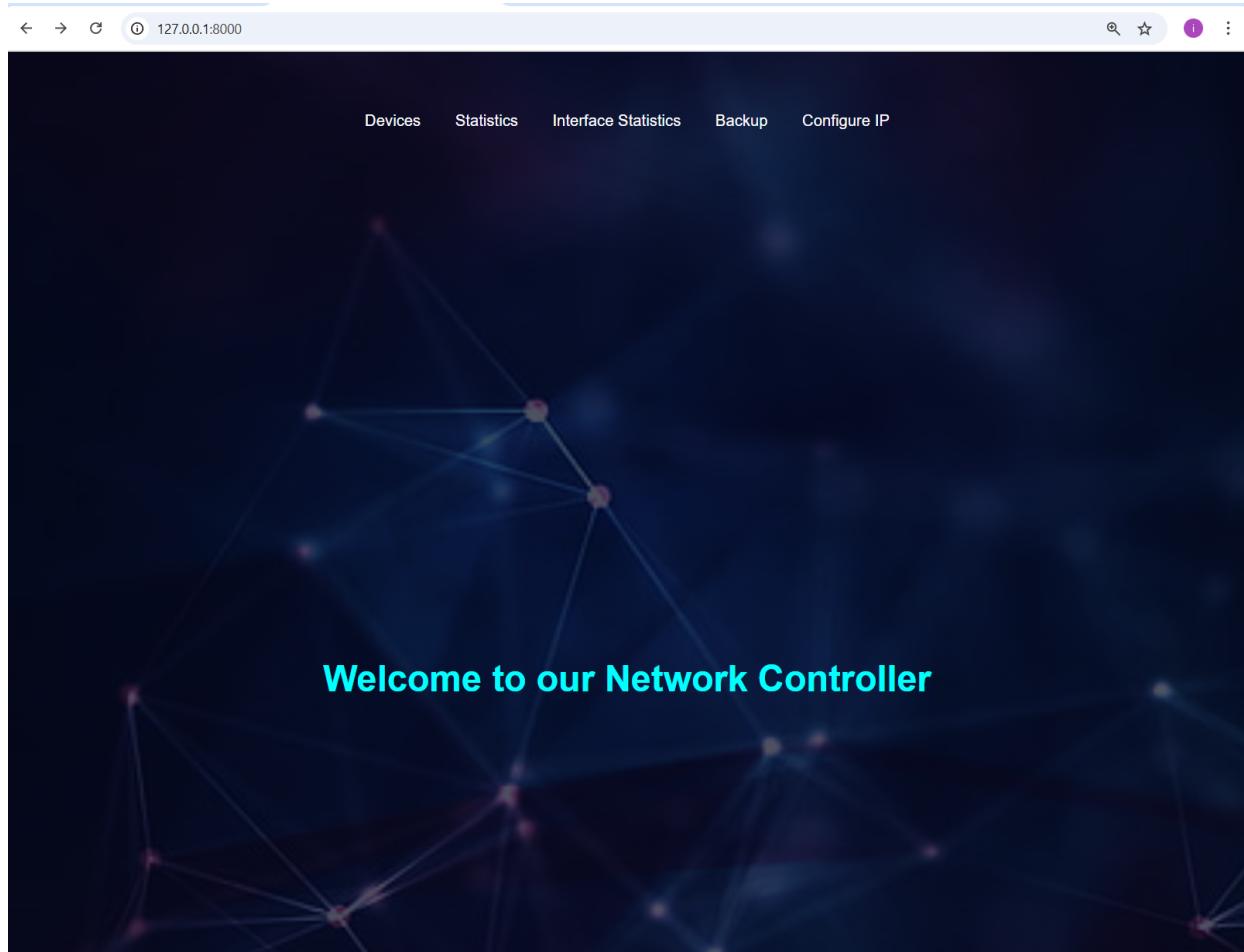
Προκειμένου να τρέξει ο Django server τρέχουμε την εντολή python3 manage.py runserver η οποία βρίσκεται στην εικόνα 7.4.

```
^Ciasonas@DESKTOP-9M04JK8:~/SDN_Django_framework_for_implementation_ne
o twork_service_configuration_application$ python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
October 21, 2024 - 15:53:58
Django version 3.2.4, using settings 'SDN_Django_framework_for_impleme
ntation_network_service_configuration_application.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Σχήμα 7.4. Έξοδος τερματικού κατά την εκκίνηση του Django server

Ο εξυπηρετητής τρέχει σαν διεργασία στο λειτουργικό και είναι διαθέσιμος στη διεύθυνση <http://127.0.0.1:8000/>. Εάν εισάγουμε αυτή τη διεύθυνση σε έναν φυλλομετρητή της επιλογής μας, θα ανακατευθυνθούμε στην αρχική σελίδα, η οποία θα παρουσιαστεί στον χρήστη όπως στην εικόνα 7.5. Το εμπρόσθιο τμήμα της εφαρμογής(frontend) είναι γραμμένο με HTML,CSS ενώ το οπίσθιο τμήμα(backend) είναι γραμμένο σε Python.



Σχήμα 7.5. Αρχική Σελίδα της εφαρμογής

7.3 Devices

Προκειμένου να δημιουργήσουμε μία νέα συσκευή η παρακάτω κλάση κώδικα μας βοηθάει στο να γίνει όπως στο σχήμα 7.6.

```
class Device(models.Model):
    name = models.CharField(max_length=100)
    host = models.CharField(max_length=70)
    username = models.CharField(max_length=100)
    password = models.CharField(max_length=100, blank=True)
    secret = models.CharField(max_length=100, blank=True)
    device_type = models.CharField(
        max_length=30, choices=(("router", "Router"), ("switch", "Switch"), ("firewall", "Firewall")), blank=True
    )

    platform = models.CharField(
        max_length=30, choices=(("cisco_ios", "Cisco IOS"), ("cisco_iosxe", "Cisco IOS XE")), blank=True
    )
```

Σχήμα 7.6. Αρχικοποίηση της συσκευής μέσω της κλάσης Device

Αφού λοιπόν πατήσουμε το κουμπί Devices αυτό θα μας δρομολογήσει στο αντίστοιχο HTML link, το οποίο θα μας εμφανίσει μια άλλη σελίδα αυτή του σχήματος 7.7.

Αποτέλεσμα του πίνακα του σχήματος 7.7 είναι όλες οι συσκευές οι οποίες είναι στη βάση δεδομένων μας όπου ο χρήστης πρόσθεσε προκειμένου να μπορεί να αναδράσει

με αυτές. Το πως προσθέτουμε συσκευές αναλύεται στο παρότρημα.

CONTROLLER		
Device Interfaces		
Id	Name	Host
1	R1	192.168.2.9
2	R4	192.168.56.152
3	Router_Ericsson	192.168.56.120
4	R2_Ericsson	192.168.2.15
5	R3	192.168.56.123
6	R1_testing	192.168.56.146

Σχήμα 7.7. Controller-Device Interfaces

Με το πάτημα ενός από αυτά τα κουμπιά(σχήμα 7.7), επιστρέφεται μια σελίδα HTML που εμφανίζει ποια Interfaces είναι UP και ποια είναι enabled (Σχήμα 7.8).

Interface name	UP	Enabled
FastEthernet0/0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
FastEthernet2/0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
FastEthernet2/1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Serial3/0	<input type="checkbox"/>	<input type="checkbox"/>
Serial3/1	<input type="checkbox"/>	<input type="checkbox"/>
Serial3/2	<input type="checkbox"/>	<input type="checkbox"/>
Serial3/3	<input type="checkbox"/>	<input type="checkbox"/>
POS4/0	<input type="checkbox"/>	<input type="checkbox"/>

Σχήμα 7.8. Interfaces status

7.4 Στατιστικά της συσκευής

Η λειτουργία του κουμπιού Statistics βασίζεται στη συνάρτηση `get_interface_statistics`, η οποία είναι υπεύθυνη για τη σύνδεση σε μια δικτυακή συσκευή, την απόκτηση στατιστικών πληροφοριών σχετικά με τις διεπαφές της, και την παρουσίαση αυτών των πληροφοριών σε μια ιστοσελίδα μέσω ενός προτύπου HTML. Η συνάρτηση αυτή δέχεται δύο παραμέτρους: το δίτημα του χρήστη (`request`) και το αναγνωριστικό της συσκευής (`device_id`), το οποίο χρησιμοποιείται για να εντοπίσει τη συγκεκριμένη συσκευή από τη βάση δεδομένων.

Για να εντοπίσει τη συσκευή, παρουσιάζεται μία λίστα από το User Interface όπως και στην εικόνα (Devices) 7.9.

The screenshot shows a Django application interface. At the top, there is a header bar with the title "Devices". Below the header, there is a table listing seven devices with columns for "Id", "Name", and "Host". The table data is as follows:

Id	Name	Host
1	R1	192.168.2.9
2	R4	192.168.56.152
3	Router_Ericsson	192.168.56.120
4	R2_Ericsson	192.168.2.15
5	R3	192.168.56.123
6	R1_testing	192.168.56.146
7	containerlab	172.20.20.2

Below the table, there is another section titled "Device Statistics".

Σχήμα 7.9. Στατιστικά Συσκευής

Πατώντας πάνω σε κάθε μια από τις συσκευές που έχουμε ορίσει και εμφανίζονται ως links εμφανίζει τα παρακάτω στατιστικά αντίστοιχα. Όπως στην εικόνα 7.10.

R2_ERICSSON INTERFACE STATISTICS	
Category	Value
cpu	<ul style="list-style-type: none"> 0: {"%usage": 1.0}
memory	<ul style="list-style-type: none"> used_ram: 49537588 available_ram: 386419564
temperature	<ul style="list-style-type: none"> invalid: {'is_alert': False, 'is_critical': False, 'temperature': -1.0}
power	<ul style="list-style-type: none"> invalid: {'status': True, 'output': -1.0, 'capacity': -1.0}
fans	<ul style="list-style-type: none"> invalid: {'status': True}

Σχήμα 7.10. Στατιστικά Συσκευής

7.5 Στατιστικά της διεπαφής

Η συνάρτηση επιτρέπει τη σύνδεση σε μια δικτυακή συσκευή, την ανάκτηση των στατιστικών των διεπαφών της και την παρουσίαση αυτών των δεδομένων σε μια ιστοσελίδα. Αυτό επιτυγχάνεται επιλέγοντας Interface Statistics από το αρχικό μενού. Αυτό μας δρομολογεί στο HTML page του σχήματος 7.11.

The screenshot shows a web browser window with the URL `127.0.0.1:8000/manage3/`. The main content area is titled "Interfaces Statistics". Above it, there is a table titled "Devices" with columns "ID", "Name", and "Host". The data in the table is as follows:

ID	Name	Host
1	R1	192.168.2.9
2	R4	192.168.56.152
3	Router_Ericsson	192.168.56.120
4	R2_Ericsson	192.168.2.15
5	R3	192.168.56.123
6	R1_testing	192.168.56.146
7	latest	192.168.56.4

Σχήμα 7.11. Στατιστικά διεπαφής αρχική σελίδα

Επιλέγοντας μια από τις συσκευές εμφανίζεται η αντίστοιχη σελίδα με τα στατιστικά των διεπαφών της όπως φαίνονται στο σχήμα 7.12.

The screenshot shows a web browser window with the URL `127.0.0.1:8000/interface_statistics/6`. The main content area is titled "Interface Rx Unicast Packets Rx Octets". The data in the table is as follows:

Interface	Rx Unicast Packets	Rx Octets
FastEthernet0/0	12217	1138269
FastEthernet2/0	0	0
FastEthernet2/1	0	0
Serial3/0	0	0
Serial3/1	0	0
Serial3/2	0	0
Serial3/3	0	0
POS4/0	0	0

Σχήμα 7.12. Στατιστικά διεπαφής

7.6 Backup της συσκευής

Προκειμένου να σώσουμε τη διαμόρφωση της συσκευής πατάμε το κουμπί backup της αρχικής σελίδας. Το κουμπί αυτό μας κατευθύνει στη σελίδα του σχήματος 7.13.

ID	Name	Host
1	R1	192.168.2.9
2	R4	192.168.56.152
3	Router_Ericsson	192.168.56.120
4	R2_Ericsson	192.168.56.4
5	R3	192.168.56.123
6	R1_testing	192.168.56.146
7	containerlab	172.20.20.2

Get Backup of Running config

Σχήμα 7.13. Get Backup of Running config

Πατώντας τη συσκευή που θέλουμε να πάρουμε το running config της μας επιστρέφεται η τρέχων διαμόρφωση σε HTML μορφή(σχήμα 7.14).

Σχήμα 7.14. Επιστροφή Τρέχων Διαμόρφωση

7.7 Διαμόρφωση διεύθυνσης IP

Για την αλλαγή της IP διεύθυνσης, επιλέγεται το κουμπί Configure IP στην αρχική σελίδα. Αυτή η επιλογή ανακατευθύνει στη σελίδα HTML που εμφανίζεται στο Σχήμα 7.15. Η διαμόρφωση της IP διεύθυνσης πραγματοποιείται με την εισαγωγή των παρακάτω στοιχείων, όπως φαίνεται στο Σχήμα 7.15: hostname, username, password, Interface, ipaddress και subnetmask. Με την παροχή αυτών των στοιχείων, επιτυγχάνεται η παραμετροποίηση της IP διεύθυνσης σε ένα συγκεκριμένο interface και η αλλαγή της διεύθυνσης.

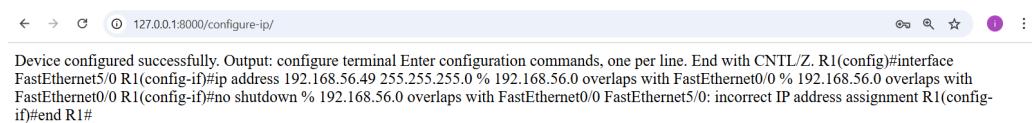
The screenshot shows a web browser window with the URL `127.0.0.1:8000/configure-ip/`. The title of the page is **Configure IP on Cisco Device**. The form contains the following fields:

- Device Hostname: `192.168.56.4`
- Username: `jasonas`
- Password: `*****`
- Interface: `FastEthernet5/0`
- IP Address: `192.168.56.49`
- Subnet Mask: `255.255.255.0`

At the bottom of the form is a blue **Configure** button.

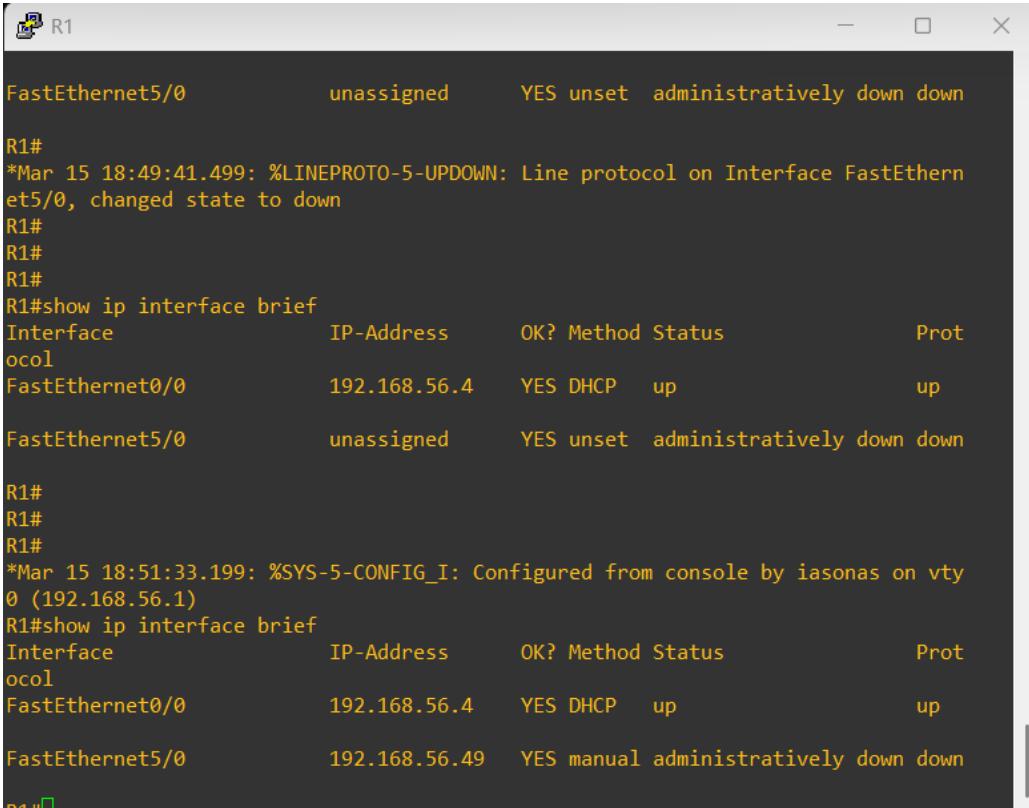
Σχήμα 7.15. IP Διαμόρφωση

Εισάγοντας τα στοιχεία που εμφανίζονται στο Σχήμα 7.15, εφόσον η διαμόρφωση είναι επιτυχής, επιστρέφεται η σελίδα HTML του Σχήματος 7.16.



Σχήμα 7.16. Επιστροφή επιβεβαίωσης επιτυχούς ρύθμισης IP

Στο σχήμα 7.17 φαίνεται η διαφορά τυπώνοντας στο CLI τις IP διευθύνσεις πριν και μετά.



```
R1
FastEthernet5/0      unassigned      YES unset administratively down down
R1#
*Mar 15 18:49:41.499: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet5/0, changed state to down
R1#
R1#
R1#
R1#show ip interface brief
Interface          IP-Address      OK? Method Status      Prot
ocol
FastEthernet0/0    192.168.56.4   YES DHCP   up           up
FastEthernet5/0    unassigned      YES unset administratively down down
R1#
R1#
R1#
*Mar 15 18:51:33.199: %SYS-5-CONFIG_I: Configured from console by iasonas on vty 0 (192.168.56.1)
R1#show ip interface brief
Interface          IP-Address      OK? Method Status      Prot
ocol
FastEthernet0/0    192.168.56.4   YES DHCP   up           up
FastEthernet5/0    192.168.56.49  YES manual administratively down down
R1#
```

Σχήμα 7.17. Μεταβολή της IP διεύθυνσης λόγω διαμόρφωσης

8 Containerization και Deployment

8.1 Containerization με Docker

8.1.1 Δημιουργία Docker Image

Προκειμένου να μπορέσουμε να χρησιμοποιήσουμε το Docker Desktop, θα πρέπει να έχουμε δημιουργήσει την εφαρμογή μας σε μορφή container. Για να το πετύχουμε αυτό, είναι απαραίτητο να δημιουργήσουμε το Dockerfile(Σχήμα 8.1), το οποίο ουσιαστικά μετατρέπει την εφαρμογή μας σε container. Προτού εξηγήσουμε τη διαδικασία δημιουργίας του Image, είναι σημαντικό να αναφέρουμε τον λόγο για τον οποίο πραγματοποιείται αυτή η διαδικασία. Οι Cloud Native microservices είναι σχεδιασμένες να λειτουργούν πάνω στην υποδομή του Kubernetes. Για να μπορέσει το Kubernetes να τις διαχειριστεί, πρέπει να γίνουν deploy σε αυτόν. Ο Kubernetes δεν αναγνωρίζει εφαρμογές, ούτε containers, ούτε Images αλλά μόνο pods. Τα pods περιέχουν τα containers, τα οποία ουσιαστικά αποτελούν το λογισμικό που δημιουργείται μέσω της διαδικασίας δημιουργίας του Image. Ο Kuβερνήτης δεν αναγνωρίζει τίποτα άλλο εκτός από τα pods. Επειδή, λοιπόν, ο Kubernetes αναγνωρίζει pods που ορίζονται μέσω αρχείων YAML, τα οποία αναλύονται εκτενέστερα στην παράγραφο 8.2.1, δημιουργούμε το Image με τέτοιο τρόπο ώστε να μπορεί να το αναγνωρίσει ο Kubernetes στο τοπικό Docker repository και να προχωρήσει στη δημιουργία του pod.

```

👉 Dockerfile
1  # Use an official Python runtime as the base image
2  FROM python:3.9
3
4  # Set the working directory in the container
5  WORKDIR /app
6
7  # Copy the requirements file and install dependencies
8  COPY requirements.txt /app/
9  RUN pip install --no-cache-dir --upgrade pip
10 RUN apt-get update \
11     && apt-get install -y libyaml-dev
12
13 RUN pip install --no-cache-dir -r requirements.txt
14
15 # Copy the Django project code to the container
16 COPY . /app/
17
18 # Set the PYTHONPATH environment variable to include the Django project directory
19 ENV PYTHONPATH=/app
20
21 # Expose the port on which the Django development server will run
22 EXPOSE 8000
23
24 # Run the Django development server
25 CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
26

```

Σχήμα 8.1. Dockerfile

To Dockerfile του σχήματος 8.1 δημιουργεί ένα περιβάλλον Python 3.9 για ένα έργο Django. Ορίζει το φάκελο εργασίας στον κατάλογο /app, εγκαθιστά τις απαιτούμενες εξαρτήσεις από το αρχείο requirements.txt(σχήμα 8.4) , ενημερώνει το pip, και εγκαθιστά βιβλιοθήκες συστήματος (π.χ. libyaml-dev). Αντιγράφει τον κώδικα του έργου Django στο κοντέινερ, θέτει τη μεταβλητή περιβάλλοντος PYTHONPATH, εκθέτει την θύρα 8000 για τον διακομιστή Django και εκκινεί την εφαρμογή με την εντολή runserver.

Η εντολή docker build(σχήμα 8.2) χρησιμοποιείται για τη δημιουργία μιας εικόνας Docker (Docker image) από ένα συγκεκριμένο αρχείο Dockerfile και τα αρχεία που περιλαμβάνονται στον φάκελο εργασίας (context).

Αυτή η διαδικασία πακετάρει τον κώδικα της εφαρμογής, τις εξαρτήσεις και τις ρυθμίσεις σε ένα απομονωμένο περιβάλλον. Με αυτόν τον τρόπο, η παραγόμενη εικόνα μπορεί να διαμορφωτεί και να εκτελεστεί με συνέπεια σε διαφορετικά συστήματα, εξασφαλίζοντας ομοιόμορφο περιβάλλον εκτέλεσης. Είναι σημαντικό εργαλείο για αυτοματοποίηση και ανάπτυξη σε συστήματα CI/CD. Για να φτιάξουμε το image τρέχουμε τη εντολή docker build -t djangothesis:v2(σχήμα 8.2). Για να μπορέσουμε να τρέξουμε την εντολή αυτή πρέπει να βρισκόμαστε στο path που βρίσκεται το requirements.txt και το manage.py. Η διαδρομή φαίνεται στο σχήμα 8.3 και είναι ουσιαστικά εκεί που βρίσκεται και το manage.py αρχείο.

```
root@DESKTOP-9M04JK8:/home/iasonas/SDN_Django_framework_for_implementation_network_service_configuration_application_iasonas# docker build -t djangothesis:v2 .
[*] Building 1.4s (13/13) FINISHED
    => [internal] load build definition from Dockerfile
    => [internal] transfering dockerfile: 743B
    => [internal] load metadata for docker.io/library/python:3.9
    => [auth] library/python:pull token for registry-1.docker.io
    => [internal] load .dockerrcignore
    =>> transferring context: 2B
    => [1/7] FROM docker.io/library/python:3.9@sha256:5ea663alc6ba266fdcac5949d1d2ea364ce30a2da92a3df95bb3c01437633
    => [internal] load build context
    =>> transferring context: 8.42kB
    => CACHED [2/7] WORKDIR /app
    => CACHED [3/7] COPY requirements.txt /app/
    => CACHED [4/7] RUN pip install --no-cache-dir --upgrade pip
    => CACHED [5/7] RUN apt-get update & apt-get install -y libyaml-dev
    => CACHED [6/7] RUN pip install --no-cache-dir -r requirements.txt
    => CACHED [7/7] COPY . /app/
    => exporting to image
    =>> exporting layers
    =>> writing image sha256:24ff0404f1f1162755271222535d264ee8c8d47d083a6fb0f68e2ac94f5a1f2
    =>> naming to docker.io/library/djangothesis:v2
root@DESKTOP-9M04JK8:/home/iasonas/SDN_Django_framework_for_implementation_network_service_configuration_application_iasonas#
```

Σχήμα 8.2. Docker build-Δημιουργία του κοντείνερ

```
root@DESKTOP-9M04JK8:/home/iasonas/SDN_Django_framework_for_implementation_network_service_configuration_application_iasonas# ls -ltrh
total 1.8M
-rwxr-xr-x 1 root root 737 Oct 28 11:41 manage.py
-rw-r--r-- 1 root root 369 Oct 28 11:41 create3ns.sh
-rw-r--r-- 1 root root 31K Oct 28 11:41 controller.png
-rw-r--r-- 1 root root 13K Oct 28 11:41 connection.png
-rw-r--r-- 1 root root 1.3M Oct 28 11:41 'Screenshot 2023-07-24 123620.png'
-rw-r--r-- 1 root root 229K Oct 28 11:41 'Screenshot 2023-05-25 145641.png'
drwxr-xr-x 3 root root 4.0K Oct 28 11:41 SDN_Django_framework_for_implementation_network_service_configuration_application
-rw-r--r-- 1 root root 485 Oct 28 11:41 README.md
-rw-r--r-- 1 root root 332 Oct 28 11:41 Dockerfile_backup
-rw-r--r-- 1 root root 784 Oct 28 11:41 Dockerfile
-rw-r--r-- 1 root root 1.9K Oct 28 11:41 Configuration.txt
drwxr-xr-x 2 root root 4.0K Oct 28 11:41 yaml
-rw-r--r-- 1 root root 393 Oct 28 11:41 user.name
-rw-r--r-- 1 root root 45K Oct 28 11:41 statistics.png
-rw-r--r-- 1 root root 527 Oct 28 11:41 requirements1.txt
-rw-r--r-- 1 root root 530 Oct 28 11:41 requirements.txt
drwxr-xr-x 6 root root 4.0K Oct 28 11:41 network
-rw-r--r-- 1 root root 136K Mar 15 17:48 db.sqlite3
drwxr-xr-x 2 root root 4.0K Mar 15 17:45 configs
root@DESKTOP-9M04JK8:/home/iasonas/SDN_Django_framework_for_implementation_network_service_configuration_application_iasonas# pwd
/home/iasonas/SDN_Django_framework_for_implementation_network_service_configuration_application_iasonas
root@DESKTOP-9M04JK8:/home/iasonas/SDN_Django_framework_for_implementation_network_service_configuration_application_iasonas#
```

Σχήμα 8.3. Dockerfile Path

```
root@DESKTOP-9M04JK8:/home/iasonas/SDN_Django_framework_for_implementation_network_service_configuration_application_iasonas# cat requirements.txt
asn1crypto==0.24.0
bcrypt==3.1.4
certifi==2018.8.24
chardet==3.0.4
cryptography==2.3.1
Django==3.2.4
enum34==1.1.6
future==0.16.0
idna==2.7
ipaddress==1.0.22
Jinja2==2.10
junos-eznc==2.2.0
napalm==2.3.2
ncclient==0.6.3
netaddr==0.7.19
netmiko==2.1.7
paramiko==2.4.2
pyasn1==0.4.4
pycparser==3.19
pyeapi==0.8.2
pyIOSXR==0.53
PyNaCl==1.3.0
pyNexos==0.0.3
pyserial==3.4
pytz==2018.5
PyYAML==5.4.1
requests==2.19.1
scp==0.11.0
selectors==2.0.1
six==1.11.0
textfsm==0.4.1
urllib3==1.23
pysnmp==4.4.12
celery==5.3.0
MarkupSafe==2.0.1

root@DESKTOP-9M04JK8:/home/iasonas/SDN_Django_framework_for_implementation_network_service_configuration_application_iasonas#
```

Σχήμα 8.4. Αρχείο requirements.txt

Τώρα που έγινε build το image είναι διαθέσιμο τοπικά(σχήμα 8.5) με το όνομα djangothesis και tag v2 όπως αυτό ορίστηκε από την εντολή docker build(σχήμα 8.2).

```
root@DESKTOP-9M04JK8:/home/iasonas/SDN_Django_framework_for_implementation_network_service_configuration_application_iasonas# docker image list
REPOSITORY          TAG      IMAGE ID      CREATED     SIZE
djangothesis        v2      24ff0404f1f1   12 minutes ago  1.13GB
vnetlab/cisco_ios   17.12.01  70c8af994324   4 months ago  704MB
iasonasi/django_thesis    latest   851c7a4ba667   4 months ago  1.13GB
public.ecr.aws/docker/library/debian  bookworm-slim  086cc0a12e56   4 months ago  74.8MB
ubuntu              20.04    6013ae1a63c2   5 months ago  72.8MB
ghcr.io/nokia/srlinux  latest   f23957871f9f   6 months ago  2.66GB
gcr.io/k8s-minikube/kicbase  v0.0.39  67a4b1138d2d   23 months ago  1.05GB
root@DESKTOP-9M04JK8:/home/iasonas/SDN_Django_framework_for_implementation_network_service_configuration_application_iasonas#
```

Σχήμα 8.5. Docker image list

8.2 Deployment με Kubernetes

Τα τελευταία χρόνια, παρατηρείται ραγδαία αύξηση στον τομέα της πληροφορικής, με την εμφάνιση και εξάπλωση νέων εννοιών, όπως ο κυβερνήτης και τα microservices. Ένας βασικός παράγοντας που συνέβαλε στην εισαγωγή αυτών των τεχνολογιών είναι η ικανότητα εικονικοποίησης του λειτουργικού συστήματος, καθώς και η δυνατότητα εκτέλεσης εφαρμογών ως κοντέινερ. Αυτές οι τεχνολογίες επιτρέπουν την απομόνωση και τη διαχείριση εφαρμογών με μεγαλύτερη ευελιξία και αποτελεσματικότητα, κάπι που έχει οδηγήσει σε σημαντικές αλλαγές στον τρόπο ανάπτυξης και λειτουργίας των σύγχρονων υποδομών λογισμικού.

Τα κοντέινερ είναι ένας καλός τρόπος για να ομαδοποιήσουμε και να εκτελέσουμε τις εφαρμογές μας. Σε ένα περιβάλλον παραγωγής, πρέπει να διαχειριστούμε τα κοντέινερ που εκτελούν τις εφαρμογές και να διασφαλίσουμε ότι δεν υπάρχει χρόνος διακοπής λειτουργίας. Για παράδειγμα, εάν ένα κοντέινερ πέσει κάτω, ένα άλλο κοντέινερ πρέπει να ξεκινήσει.

Το Kubernetes παρέχει ένα πλαίσιο για την εκτέλεση κατανεμημένων συστημάτων με μεγάλη ευχέρεια. Αναλαμβάνει τη διαχείριση της κλιμάκωσης και της ανθεκτικότητας (failover)

μίας εφαρμογής, προσφέροντας παράλληλα διάφορα αναπτυξιακά μοτίβα και άλλες χρήσιμες δυνατότητες. Δεν είναι στόχος να αναλυθεί με κάθε λεπτομέρεια η λειτουργία του κυβερνήτη, καθώς κάπι τέτοιο απαιτεί μια ολόκληρη διπλωματική εργασία, αλλά θα παρουσιαστούν τα βασικά χαρακτηριστικά του που χρησιμοποιήθηκαν στην πράξη για την υλοποίηση της διπλωματικής εργασίας.

Στη διπλωματική αυτή εργασία, χρησιμοποιείται ένα τοπικό περιβάλλον ανάπτυξης με Kubernetes μέσω του Minikube και του WSL2 (Windows Subsystem for Linux 2) για την ανάπτυξη και δοκιμή της εφαρμογής Django.

Το Kubernetes είναι μια δημοφιλής πλατφόρμα ενορχήστρωσης κοντέινερ, που επιτρέπει την αυτόματη διαχείριση και κλιμάκωση εφαρμογών σε περιβάλλοντα παραγωγής, ενώ το Minikube προσφέρει τη δυνατότητα εκκίνησης ενός τοπικού Kubernetes cluster. Με τον τρόπο αυτό, επιτυγχάνεται η δημιουργία ενός ασφαλούς, απομονωμένου περιβάλλοντος δοκιμών, το οποίο προσομοιώνει ένα πλήρες cluster, χωρίς την ανάγκη πρόσθετης υποδομής cloud.

Χάρη στο WSL2, το οποίο επιτρέπει την εκτέλεση Linux πυρήνα απευθείας στα Windows, εξασφαλίζεται ευκολία στη διαχείριση του cluster και της εφαρμογής Django, ενώ η χρήση εργαλείων όπως το kubectl καθιστά εύκολη την παρακολούθηση και τον έλεγχο των pods και υπηρεσιών. Αυτό το περιβάλλον προσφέρει μια ολοκληρωμένη εμπειρία ανάπτυξης και δοκιμής, βοηθώντας στην κατανόηση των αρχών του Kubernetes και διευκολύνοντας τη μετάβαση της εφαρμογής σε μεγαλύτερα production περιβάλλοντα

Το Minikube είναι ένα εργαλείο που απλοποιεί την εκτέλεση και διαχείριση ενός τοπικού Kubernetes cluster στον υπολογιστή σας, ειδικά σχεδιασμένο για περιβάλλοντα ανάπτυξης και δοκιμών. Σας επιτρέπει να ξεκινήσετε ένα Kubernetes cluster με ένα μόνο κόμβο (ή ακόμα και πολλούς σε ορισμένες περιπτώσεις) χρησιμοποιώντας εικονικοποίηση μέσω WSL, Docker, ή Hypervisor. Ο κύριος σκοπός του Minikube είναι να παρέχει ένα περιβάλλον Kubernetes με όλες τις βασικές δυνατότητες του Kubernetes αλλά χωρίς την πολυπλοκότητα που θα απαιτούσε η διαχείριση ενός cluster σε παραγωγικό περιβάλλον.

Επιπλέον, το Minikube διαθέτει ενσωματωμένα εργαλεία, όπως τη δυνατότητα να παρακολουθείτε και να διαχειρίζεστε τον πίνακα ελέγχου του Kubernetes, να δημιουργείτε pods, deployments, και services(19), και να παρακολουθείτε τα logs των εφαρμογών σας, ενώ επιτρέπει επίσης εύκολη σύνδεση με εργαλεία όπως το kubectl για πλήρη πρόσβαση στη διαχείριση του cluster. Αυτό το καθιστά ιδανικό για προγραμματιστές που θέλουν να πειραματιστούν με Kubernetes, να κάνουν δοκιμές εφαρμογών, ή να αναπτύξουν μικροϋπηρεσίες τοπικά χωρίς να απαιτείται η πολυπλοκότητα ενός πλήρους cluster όπως το περιβάλλον της παρούσας διπλωματικής εργασίας. Παρακάτω μπορούμε να δούμε πόσο εύκολα μπορούμε να ξεκινήσουμε ένα τοπικό περιβάλλον κυβερνήτη(σχήμα 8.5).

8.2.1 Δημιουργία Kubernetes manifest files

Στο πλαίσιο της ανάπτυξης της εφαρμογής, χρησιμοποιήθηκε το Kubernetes για τη δημιουργία και διαχείριση ενός pod που φιλοξενεί την εφαρμογή Django. Το αρχείο διαμόρφωσης YAML (pod1.yml) που δημιουργήθηκε, ακολουθεί τη βασική δομή του Kubernetes, ορίζοντας

```
iasonas@DESKTOP-9M04JK8:~$ minikube start
minikube v1.34.0 on Ubuntu 22.04 (amd64)
Kubernetes 1.31.0 is now available. If you would like to upgrade, specify: --kubernetes-version=v1.31.0
Using the docker driver based on existing profile
Starting "minikube" primary control-plane node in "minikube" cluster
Pulling base image v0.0.45 ...
docke "minikube" container is missing, will recreate.
Creating docker container (CPUs=2, Memory=3900MB) ...
! Image was not built for the current minikube version. To resolve this you can delete and recreate your minikube cluster using the latest images. Expected minikube version: v1.30.1 -> Actual minikube version: v1.34.0
Preparing Kubernetes v1.26.3 on Docker 23.0.2 ...
  • Generating certificates and keys ...
  • Booting up control plane ...
  • Configuring RBAC rules ...
Configuring bridge CNI (Container Networking Interface) ...
Verifying Kubernetes components...
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: storage-provisioner, default-storageclass
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
iasonas@DESKTOP-9M04JK8:~$ kubectl get pods -A
NAMESPACE     NAME           READY   STATUS    RESTARTS   AGE
kube-system   coredns-787d4945fb-2kwmn   1/1    Running   0          11s
kube-system   coredns-787d4945fb-htlz6   1/1    Running   0          11s
kube-system   etcd-minikube            1/1    Running   0          27s
kube-system   kube-apiserver-minikube  1/1    Running   0          26s
kube-system   kube-controller-manager-minikube  1/1    Running   0          25s
kube-system   kube-proxy-c4rrz        1/1    Running   0          11s
kube-system   kube-scheduler-minikube  1/1    Running   0          26s
kube-system   storage-provisioner      1/1    Running   0          23s
```

Σχήμα 8.6. Minikube deployment

τον τύπο πόρου ως Pod και εκκωφάρωντας μεταδεδομένα όπως το όνομα djangotestapp. Στην ενότητα spec, ορίζεται ένα container το οποίο χρησιμοποιεί την εικόνα iasonasi/djangotestapp:latest και ακούει στη θύρα 8000, η οποία είναι η προεπιλεγμένη θύρα της εφαρμογής Django. Παρακάτω το spec του yaml file.(σχήμα 8.7)

```
spec:
  containers:
    - name: djangotestapp
      image: iasonasi/djangotestapp:latest
      ports:
        - containerPort: 8000
```

Σχήμα 8.7. Spec του YAML file

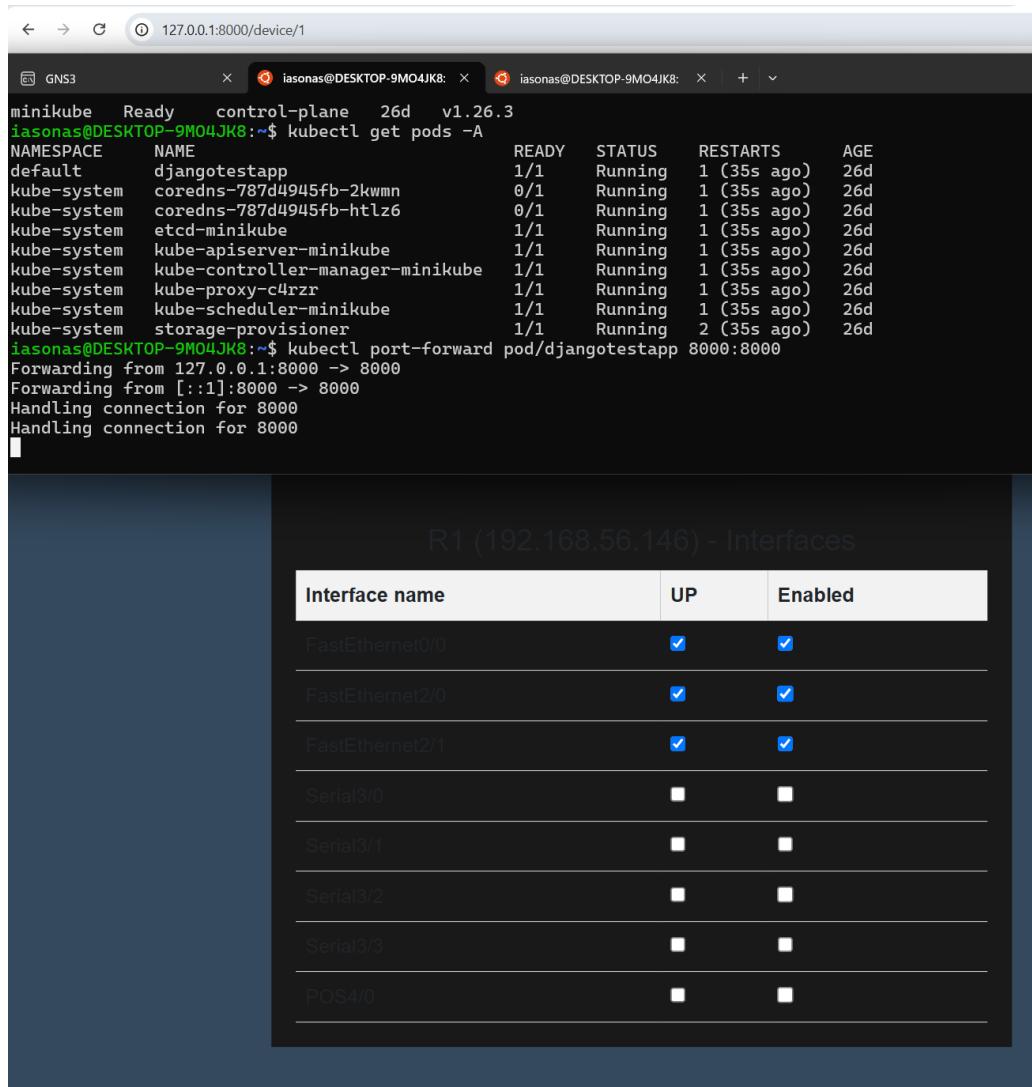
Αυτό το παράδειγμα αποδεικνύει τη σημασία της χρήσης του Kubernetes YAML syntax για την αυτοματοποιημένη ανάπτυξη και διαχείριση containerized εφαρμογών. Μέσω αυτής της διαδικασίας, η εφαρμογή μπορεί να επεκταθεί εύκολα σε διάφορα περιβάλλοντα και να κλιμακωθεί ανάλογα με τις ανάγκες. Το συγκεκριμένο αρχείο YAML επιτρέπει στο Kubernetes να εκτελέσει και να διαχειριστεί το pod με τρόπο ανεξάρτητο από το υποκείμενο σύστημα, εξασφαλίζοντας επαναληψιμότητα και δυνατότητα μεταφοράς του συστήματος σε διαφορετικές υποδομές. Στην παρακάτω εικόνα φαίνεται το YAML file που χρησιμοποιήθηκε(σχήμα 8.8).

```
iasonas@DESKTOP-9M04JK8:~$ 
iasonas@DESKTOP-9M04JK8:~$ 
iasonas@DESKTOP-9M04JK8:~$ kubectl get pods -A
NAMESPACE     NAME           READY   STATUS    RESTARTS   AGE
default       djangotestapp   1/1     Running   0          15m
kube-system   coredns-787d4945fb-2kwmn   1/1     Running   0          16m
kube-system   coredns-787d4945fb-htlz6   1/1     Running   0          16m
kube-system   etcd-minikube   1/1     Running   0          16m
kube-system   kube-apiserver-minikube   1/1     Running   0          16m
kube-system   kube-controller-manager-minikube   1/1     Running   0          16m
kube-system   kube-proxy-c4rzs   1/1     Running   0          16m
kube-system   kube-scheduler-minikube   1/1     Running   0          16m
kube-system   storage-provisioner   1/1     Running   1 (15m ago)   16m
iasonas@DESKTOP-9M04JK8:~$ 
iasonas@DESKTOP-9M04JK8:~$ 
iasonas@DESKTOP-9M04JK8:~$ cat SDN_Django_framework_for_implementation_network_service_configuration_application/yaml/pod1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: djangotestapp
spec:
  containers:
  - name: djangotestapp
    image: iasonasi/djangotestapp:latest
    ports:
    - containerPort: 8000
iasonas@DESKTOP-9M04JK8:~$
```

Σχήμα 8.8. Manifest for Django pod

8.2.2 Πρόσβαση στο Django pod

Για να αποκτήσουμε πρόσβαση στην εφαρμογή Django που τρέχει στο pod του Kubernetes, μπορούμε να χρησιμοποιήσουμε την εντολή kubectl port-forward pod/djangotestapp 8000:8000. Με την εντολή kubectl port-forward θα μπορέσουμε μέσα από έναν browser να έχουμε πρόσβαση στην εφαρμογή.



Σχήμα 8.9. Πρόσβαση στο Django pod μέσα από τη λειτουργία Port forward

Στη συνέχεια, η εφαρμογή είναι προσπελάσιμη στην διεύθυνση <http://localhost:8000>. Όλες οι λειτουργίες θα πρέπει να μπορούν να εκτελεστούν, όπως και επιβεβαιώνεται από την εικόνα στο σχήμα 8.9.

8.3 Use cases με Kubernetes

Η ανάπτυξη μιας εφαρμογής σε περιβάλλον Kubernetes στοχεύει στη μετατροπή της σε μικροϋπηρεσία, προσφέροντας σημαντικά πλεονεκτήματα. Αυτή η αρχιτεκτονική διευκολύνει την κλιμάκωση μέσω των μηχανισμών του Kubernetes, όπως το Horizontal Pod Autoscaler (HPA), το οποίο επιτρέπει την αυτόματη προσαρμογή των πόρων της εφαρμογής προσθέτοντας επιπλέον replica. Στο πλαίσιο της παρούσας διπλωματικής εργασίας, η υλοποίηση περιορίστηκε στη δημιουργία pod. Παρόλα αυτά, η μετατροπή της εφαρμογής σε μικροϋπηρεσία διευκολύνει την αξιοποίηση των δυνατοτήτων του Kubernetes, καθώς η ενσωμάτωση οποιουδήποτε αντικειμένου, όπως το HPA, πραγματοποιείται εύκολα μέσω της κατάλληλης διαμόρφωσης σε YAML file.

Η χρήση Kubernetes στη συγκεκριμένη εφαρμογή αποσκοπεί στην αύξηση της ευελιξίας και της προσαρμοστικότητάς της σε περιόδους υψηλής ζήτησης. Ενώ μια μονολιθική εφαρμογή παρουσιάζει περιορισμούς στην επέκταση, η αρχιτεκτονική μικροϋπηρεσιών επιτρέπει δυναμική κλιμάκωση, διασφαλίζοντας τη βέλτιστη λειτουργία της εφαρμογής σε κάθε συνθήκη.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: django-app
  labels:
    app: django
spec:
  replicas: 2
  selector:
    matchLabels:
      app: django
  template:
    metadata:
      labels:
        app: django
    spec:
      containers:
        - image: iasonasi/djangotestapp:latest
          name: djangotestapp
      ports:
        - containerPort: 8000
          name: gunicornroot@DESKTOP-9M04JK8:/home/iasonas/SDN_Django_framework_for_implementation_network
```

Σχήμα 8.10. Deployment yaml for Django pod

9 Συμπέρασματα και Μελλοντική Εργασία

9.1 Συμπεράσματα

Η διπλωματική αυτή εργασία επικεντρώθηκε στην ανάπτυξη ενός περιβάλλοντος διαχείρισης δικτυακών συσκευών με χρήση της γλώσσας προγραμματισμού Python και του Django framework. Κατά τη διάρκεια αυτής της διαδικασίας, αναπτύχθηκαν και εφαρμόστηκαν διάφορες τεχνολογίες για την αυτοματοποίηση δικτυακών εργασιών, όπως η διαμόρφωση στοιχείων των δικτυακών συσκευών καθώς και η παρακολούθηση πληροφοριών αυτών. Το εργαλείο αναπτύχθηκε και δοκιμάστηκε κάνοντας χρήση ενός περιβάλλοντος δοκιμών δημιουργημένου με το GNS3 και με χρήση Cisco IOUs, που παρείχε την δυνατότητα ελέγχων της εφαρμογής κατά την διάρκεια της ανάπτυξής της.

9.2 Προκλήσεις και Μαθήματα

Οι προκλήσεις που αντιμετωπίστηκαν κατά τη διάρκεια της εργασίας περιελάμβαναν τη διαχείριση περιορισμών, όπως η εύρεση της κατάλληλης έκδοσης Cisco IOU, μια διαδικασία χρονοβόρα λόγω της μη δημόσιας διαθεσιμότητάς τους. Αυτό απαίτησε χρόνο για έρευνα και επίλυση τεχνικών προβλημάτων, οδηγώντας σε καθυστερήσεις, αλλά και σε πολύτιμα μαθήματα για τη διαχείριση κρίσιμων πόρων και την επιμονή στις δυσκολίες. Η εκμάθηση της Python υπήρξε εξίσου απαιτητική, καθώς απαιτούσε εμβάθυνση στη σύνταξη, στις δομές δεδομένων και στον προγραμματισμό. Οι δυσκολίες αυτές έγιναν ευκαιρίες κατανόησης της δύναμης της γλώσσας, ειδικά στον τομέα της αυτοματοποίησης. Η εμπειρία αυτή ανέδειξε τη σημασία της κριτικής σκέψης και της δημιουργίας ρεαλιστικών λύσεων.

Τέλος, η ανάπτυξη της εφαρμογής έδειξε πώς οι τεχνικές προκλήσεις μπορούν να μετατραπούν σε μαθήματα ζωής, όπως η διαχείριση χρόνου, η συνεργασία με εργαλεία ανοιχτού κώδικα και η ανάγκη για συνεχή ενημέρωση σε νέες τεχνολογίες. Η εργασία ανέδειξε τη σημασία της προσαρμοστικότητας και της δια βίου μάθησης στον τομέα των δικτύων

9.3 Μελλοντική Εργασία και Επέκταση Λειτουργικότητας

Η μελλοντική εργασία θα επικεντρωθεί στην περαιτέρω βελτίωση της λειτουργικότητας της εφαρμογής, ενσωματώνοντας επιπλέον πρωτόκολλα δικτύωσης και αυτοματοποίησης. Μια πιθανή κατεύθυνση είναι η ανάπτυξη μηχανισμών για την υποστήριξη SDN (δικτύωση οριζόμενη από λογισμικό), προκειμένου να ανταποκριθεί στις απαιτήσεις σύγχρονων δικτύων.

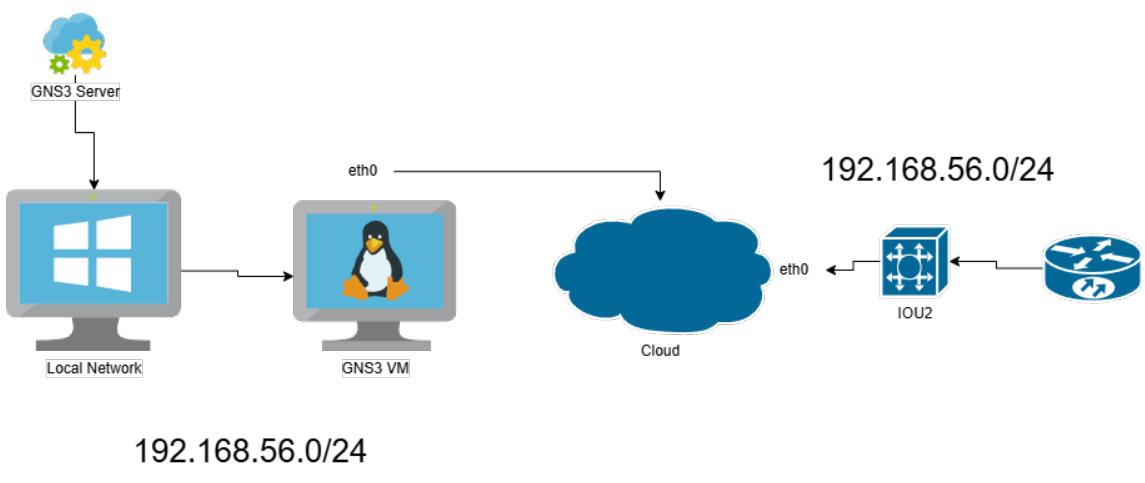
Επίσης, θα μπορούσαν να προστεθούν χαρακτηριστικά όπως η παρακολούθηση της απόδοσης του δικτύου σε πραγματικό χρόνο και η αυτοματοποίηση διαδικασιών αποκατάστασης προβλημάτων. Οι επεκτάσεις θα ενισχύσουν την ευελιξία και τη χρηστικότητα του εργαλείου.

Το περιβάλλον της εφαρμογής θα μπορούσε να είναι διαφορετικό. Μελλοντική εργασία μπορεί να αναπτυχθεί με τη βοήθεια καινούργιων περιβάλλοντων προσομοιώσης όπως το ContainerLab. Η περαιτέρω ανάπτυξη λειτουργιών της εφαρμογής, η δημιουργία καλυτερου User Experience και User Interface καθώς και η δημιουργία Testing Platform για την αυτοματοποίηση του testing της εφαρμογής θα μπορούσαν να αποτελέσουν ξεχωριστό επίσης θέμα για διπλωματική εργασία.

Τέλος, η μελλοντική έρευνα μπορεί να εξετάσει τη δυνατότητα διασύνδεσης με πλατφόρμες μηχανικής μάθησης, για την πρόβλεψη και αποτροπή πιθανών αποτυχιών, καθιστώντας την εφαρμογή ένα σύγχρονο εργαλείο αυτοματοποιημένης διαχείρισης δικτύου.

10 Παράρτημα

10.1 Σχεδίαση και Διάγραμμα τοπολογίας δικτύου



Σχήμα 10.1. Network topology design

10.2 Οδηγός χρήσης της εφαρμογής

10.3 Εκκίνηση της εφαρμογής

10.3.1 Τοπικά

Προκειμένου να εκκινήσουμε την εφαρμογή τοπικά ακολουθούμε τα παρακάτω βήματα:

- Βρισκόμαστε στο /home path.
- Κατεβάζουμε την εφαρμογή με την εντολή `git clone https://github.com/Iasimo92/SDN_Django_framework_for_implementation_network_service_configuration_application..`
- Μπαίνουμε στο φάκελο που κατεβάσαμε. Στο φάκελο αυτό βρίσκονται τα αρχεία `requirements.txt` και `manage.py`.
- Τρέχουμε την εφαρμογή με την εντολή `python3 manage.py runserver`.

10.3.2 Docker Container

Αφού έχει φτιαχτεί το image με βάση την εντολή docker build -t djangothesis:v2 όπως περιγράφηκε στην ενότητα 8.1.1 μπορούμε να τρέξουμε την εφαρμογή με την εντολή docker run. Τα βήματα που ακολουθούνται είναι τα παρακάτω:

- Ελέγχουμε με την εντολή docker image list ότι έχουμε το image που θέλουμε να τρέξουμε. Αυτή η εντολή φαίνεται στην εικόνα 8.5 της ενότητας 8.1.
- Τρέχουμε το container με την εντολή docker run -d -p 8000:8000 --name djangothesis_container djangothesis:v2.
- Η εφαρμογή τώρα είναι διαθέσιμη στο browser μας.

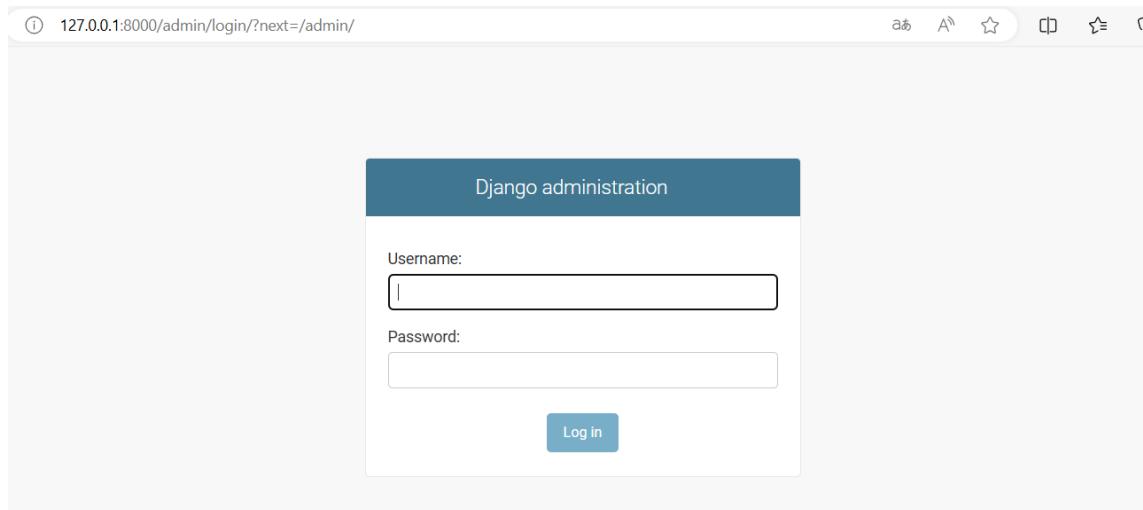
10.3.3 Access της εφαρμογής μέσα από το pod

Ο συγκεκριμένος τρόπος εξηγείται αναλυτικά στην ενότητα 8.2.2.

10.3.4 Δημιουργία αντικειμένου-Προσθήκη συσκευής

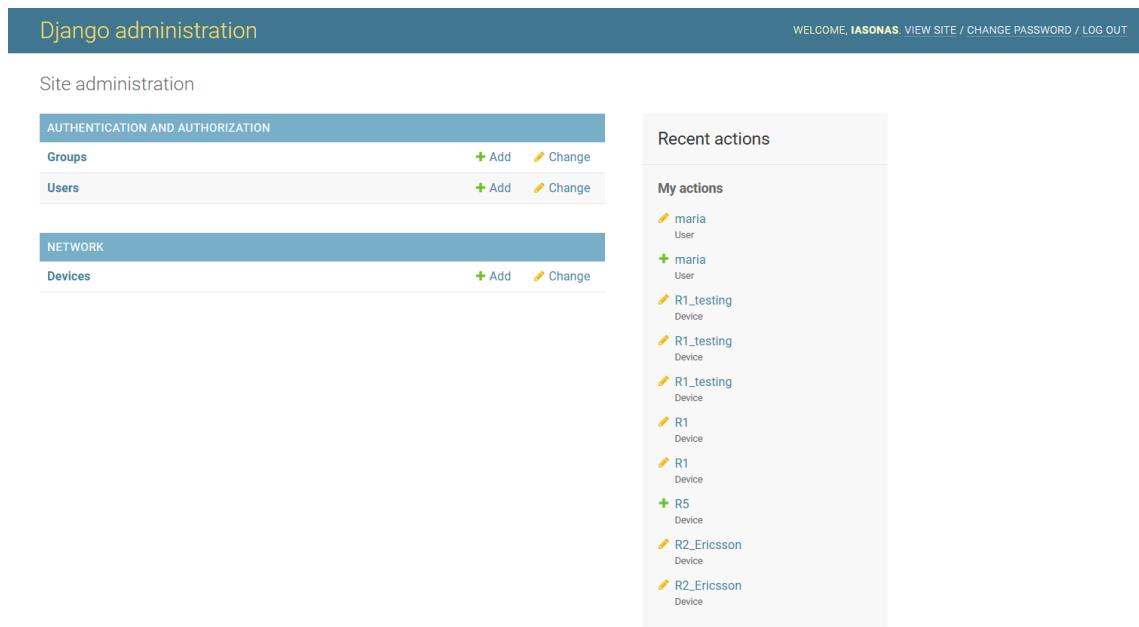
Προκειμένου να μπορούμε να διαχειριστούμε συσκευή θα πρέπει να την εισάγουμε ως αντικείμενο.

Ανοίγουμε το <http://127.0.0.1:8000/admin/> σε browser και εισάγουμε ως χρήστη iasonas και κωδικό ericsson.



Σχήμα 10.2. GUI Login

Αφού συνδεθούμε στο GUI επιλέγουμε Network, Devices και Add προκειμένου να εισάγουμε καινούργια συσκευή.



Σχήμα 10.3. GUI Login second page

Στη συνέχεια μας εμφανίζεται η παρακάτω εικόνα. Βάζουμε τα στοιχεία της συσκευής και πατάμε αποθήκευση.

The screenshot shows the 'Add device' form. The left sidebar has 'Devices' selected under 'NETWORK'. The main form fields are: Name (input field), Host (input field), Username (input field), Password (input field), Secret (input field), Device type (dropdown menu), and Platform (dropdown menu). At the bottom right are three buttons: 'Save and add another', 'Save and continue editing', and a large blue 'SAVE' button.

Σχήμα 10.4. Add device page

Αφού το κάνουμε αυτό όλες οι συσκευές που προσθέσαμε μπορούμε να τις δούμε σε όλες τις λειτουργίες της εφαρμογής. Οι λειτουργίες μπορούν να εκτελεστούν μια μια όπως έγινε στο application demo.

10.4 Κώδικας

Η εφαρμογή είναι ελεύθερη στη σελίδα: https://github.com/Iasimo92/SDN_Django_framework_for_implementation_network_service_configuration

application.

The screenshot shows a GitHub repository page for 'SDN_Django_framework_for_implementation_network_service_configuration_application'. The repository has 59 commits and 2 branches. The commits are listed below:

- lasimo92 changing requirements.txt celery version (13cd4ea - 2 months ago)
- SDN_Django_framework_for_implementation... Adding Configure IP address part (2 months ago)
- configs Adding Configure IP address part (2 months ago)
- network Adding Configure IP address part (2 months ago)
- yaml Adding the yaml first pods (last year)
- Configuration.txt Small changes (2 years ago)
- Dockerfile Dockerfile and requirements.txt update (last year)
- Dockerfile_backup Dockerfile and requirements.txt update (last year)
- README.md Update README.md (last year)
- Screenshot 2023-05-25 145641.png Add files via upload (last year)
- Screenshot 2023-07-24 123620.png Add files via upload (last year)
- connection.png Add files via upload (2 years ago)
- controller.png Add files via upload (last year)
- create3ns.sh Update create3ns.sh (last year)
- db.sqlite3 Adding Configure IP address part (2 months ago)
- manage.py initial commit (2 years ago)
- requirements.txt changing requirements.txt celery version (2 months ago)
- requirements1.txt Dockerfile and requirements.txt update (last year)
- statistics.png Add files via upload (last year)
- user.name Adding Configure IP address part (2 months ago)

About: No description, website, or topics provided.

Releases: No releases published. Create a new release

Packages: No packages published. Publish your first package

Languages: HTML 41.9%, Python 36.5%, CSS 19.9%, Dockerfile 1.1%, Shell 0.6%

Suggested workflows: Based on your tech stack

- Python Package using Anaconda: Create and test a Python package on multiple Python versions using Anaconda for package management.
- SLSA Generic generator: Generate SLSA compliant artifacts.

Σχήμα 10.5. github repo

11 Βιβλιογραφία

- 1 <https://github.com/dmfigol/network-programmability-stream>
- 2 <https://www.ericsson.com/en/portfolio/cloud-software-and-services/network-management-and-automation/ericsson-network-manager>
- 3 <https://www.cloudflare.com/learning/network-layer/what-is-mtu/>
- 4 <https://www.paramiko.org/>
- 5 <https://github.com/ktbyers/netmiko>
- 6 <https://napalm.readthedocs.io/en/latest/index.html>
- 7 <https://el.wikipedia.org/wiki/Git>
- 8 <https://docs.docker.com/engine/daemon/start/>
- 9 <https://minikube.sigs.k8s.io/docs/start/>
- 10 <https://docs.gns3.com/>
- 11 Virtual box <https://www.virtualbox.org/manual/>
- 12 <https://docs.gns3.com/docs/>
- 13 <https://developer.mozilla.org/en-US/docs/Glossary/MVC>
- 14 Wikipedia, τι είναι το Windows Subsystem for Linux
- 15 <https://www.sitepoint.com/wsl2/>
- 16 <https://www.freecodecamp.org/news/python-requirements-txt-explained/>
- 17 <https://docs.gns3.com/docs/using-gns3/advanced/connect-gns3-internet/>
- 18 <https://docs.gns3.com/docs/using-gns3/beginners/import-gns3-appliance/>
- 19 <https://kubernetes.io/docs/tutorials/hello-minikube/>