



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ  

---

UNIVERSITY OF PIRAEUS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ  
ΠΜΣ "ΠΛΗΡΟΦΟΡΙΚΗ"

# **Ανάπτυξη Εφαρμογής για παραμετροποίηση δικτύου με το Django Framework**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

**Ιάσωνας Σιμώτας**

Επιβλέπων: Δουληγέρης Χρήστος  
Καθηγητής ΠΑΠΕΙ

Αθήνα, Μήνας Έτος





ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ  
UNIVERSITY OF PIRAEUS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ  
ΠΜΣ "ΠΛΗΡΟΦΟΡΙΚΗ"  
ΤΟΜΕΑΣ

# Ανάπτυξη Εφαρμογής για παραμετροποίηση δικτύου με το Django Framework

ΔΙΠΛΩΜΑΤΙΚΗ

ΤΟΥ

**Ιάσωνας Σιμωτας**

Επιβλέπων: Δουληγέρης Χρήστος  
Καθηγητής ΠΑΠΕΙ

Εγκρίθηκε από την κάτωθι τριμελή επιτροπή την 1<sup>η</sup> Ιανουαρίου 2024.

---

Όνομα Επώνυμο  
Καθηγητής

---

Όνομα Επώνυμο  
Καθηγητής

---

Όνομα Επώνυμο  
Αναπληρωτής Καθηγητής

---

Ιάσωνας Σιμώτας

Πτυχιούχος Μεταπτυχιακού ΠΜΣ Πληροφορικής

Copyright © Όνομα Επώνυμο, 2023

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 Πανεπιστημίου Πειραιά.*

# Κεφάλαιο 1

## Πρόλογος

### 1.1 Περίληψη

Η πτυχιακή αυτή εργασία αφορά την ανάπτυξη μιας σύγχρονης εφαρμογής για την παραμετροποίηση δικτυακών συσκευών. Η εφαρμογή αξιοποιεί το πλαίσιο λογισμικού Django, ενσωματώνει βιβλιοθήκες Python για αυτοματοποίηση και για τη δοκιμή της γίνεται χρήση ενός εικονικού εργαστηρίου στο GNS3.

Στόχος της εφαρμογής είναι η απλοποίηση της διαδικασίας παραμετροποίησης δικτυακών συσκευών, παρέχοντας στο χρήστη ένα φιλικό περιβάλλον εργασίας που θα επιτρέπει την διαχείριση και την εφαρμογή ρυθμίσεων μέσω αυτοματοποιημένων διαδικασιών. Κατά την ανάπτυξη της εργασίας έχουν αξιοποιηθεί αρχές του DevOps όπως η συνεχής ενσωμάτωση και παράδοση CI/CD χρησιμοποιώντας μια απλοποιημένη λογική η οποία σε καμία περίπτωση δεν προσομοιάζει τα εργαλεία και την πολυπλοκότητα μιας μεγάλης σε κλίμακα εταιρείας. Χρησιμοποιήθηκαν τεχνολογίες όπως το Netmiko και το Paramiko για την επικοινωνία με τις συσκευές, καθώς και το GNS3 ως πλατφόρμα προσομοίωσης για την αξιολόγηση της εφαρμογής σε περιβάλλον πραγματικών συνθηκών. Μέσα από την εργασία παρουσιάζονται τα βήματα υλοποίησης της εφαρμογής, οι τεχνολογικές προκλήσεις που αντιμετωπίστηκαν και οι λύσεις που υιοθετήθηκαν. Έτσι, στην εργασία αυτή πέρα από την εφαρμογή που υλοποιήθηκε που συνδυάζει την πρακτικότητα και την επεκτασιμότητα, γίνεται και περιγραφή των διαδικασιών που ακολουθήθηκαν και αποτελούν τη βάση για την ανάπτυξη σύγχρονων εφαρμογών.

### 1.2 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 μικροϋπηρεσίες 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.

## **Κεφάλαιο 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.

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 στιγμές αβεβαιότητας και πίεσης, που πολλές φορές συνδυάζο-

νταν με τις επαγγελματικές απαιτήσεις, υπήρξαν ιδιαίτερα δύσκολες. Ωστόσο, αυτές οι προκλήσεις με δίδαξαν 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 αλληλοϋποστήριξης, 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

<b>1</b>	<b>Πρόλογος</b>	<b>5</b>
1.1	Περίληψη . . . . .	5
1.2	Συνοπτική Περιγραφή της Εργασίας . . . . .	5
<b>2</b>	<b>Αναγνώριση και Ευχαριστίες</b>	<b>7</b>
<b>3</b>	<b>Εισαγωγή</b>	<b>15</b>
3.1	Στόχοι του έργου . . . . .	15
3.2	Περιγραφή προβλήματος και λύσης . . . . .	16
3.3	Τεχνολογίες που χρησιμοποιήθηκαν και γιατί . . . . .	17
3.4	Διάρθρωση της παρούσας εργασίας . . . . .	18
<b>4</b>	<b>Θεωρητικό υπόβαθρο</b>	<b>19</b>
4.1	Web Framework και Βασικές Τεχνολογίες Ανάπτυξης . . . . .	19
4.1.1	Django Framework . . . . .	19
4.1.2	Διαχείριση εκδόσεων (Version Control) με Git και GitHub . . . . .	19
4.2	Αυτοματοποίηση Δικτύων και Διασύνδεση με Δικτυακές Συσκευές . . . . .	20
4.2.1	Αυτοματοποίηση Διαχείρισης Δικτύου . . . . .	20
4.3	Βιβλιοθήκες της Python για Network automation . . . . .	21
4.3.1	Paramiko . . . . .	21
4.3.2	Netmiko . . . . .	21
4.3.3	Napalm . . . . .	21
4.3.4	Cisco IOS και REST APIs για Δικτυακές Συσκευές της Cisco . . . . .	22
4.4	Devops και Διαδικασίες Αυτοματισμού . . . . .	23
4.4.1	Συνεχής Ενσωμάτωση και Παράδοση (CI/CD) . . . . .	23
4.4.2	Containers και Docker . . . . .	23
4.4.3	Kubernetes και Container Orchestration . . . . .	24
4.5	Προσομοίωση και Εικονικά Περιβάλλοντα . . . . .	24
4.5.1	GNS3 και Εικονικά Δίκτυα . . . . .	24
4.5.2	Πρόγραμμα εικονοποίησης για το GNS3 VM-VirtualBox . . . . .	24
<b>5</b>	<b>Σχεδίαση και Ανάλυση</b>	<b>29</b>
5.1	Απαιτήσεις Συστήματος . . . . .	29
5.1.1	Λειτουργικές απαιτήσεις . . . . .	29
5.2	Αρχιτεκτονική της εφαρμογής . . . . .	30

5.2.1	Μοντέλο MVC (Model-View-Controller) . . . . .	30
5.2.2	Django MTV . . . . .	30
5.2.3	Χρήση του MTV στην εφαρμογή . . . . .	31
5.3	Εικονικό Περιβάλλον Δικτύου GNS3 –Testbed . . . . .	32
5.3.1	Σχεδίαση Τοπολογίας Δικτύου . . . . .	33
5.3.2	Προσομοίωση Συσκευών Cisco . . . . .	34
<b>6</b>	<b>Υλοποίηση της εφαρμογής</b>	<b>37</b>
6.1	Ανάπτυξη με το Django framework . . . . .	37
6.1.1	Δημιουργία Models . . . . .	37
6.1.2	Views και Urls . . . . .	37
6.1.3	User Interface (Templates) . . . . .	38
6.2	REST API Integration και διασύνδεση με το SSH protocol . . . . .	39
6.3	Διασύνδεση με το GNS3 . . . . .	40
<b>7</b>	<b>Επίδειξη της εφαρμογής(Application Demo)</b>	<b>45</b>
7.1	Εισαγωγή-Η λογική της λειτουργίας της εφαρμογής Django . . . . .	45
7.2	Η αρχική σελίδα της εφαρμογής . . . . .	46
7.3	Devices . . . . .	47
7.4	Στατιστικά της συσκευής . . . . .	49
7.5	Στατιστικά της διεπαφής . . . . .	49
7.6	Backup της συσκευής . . . . .	50
7.7	Διαμόρφωση διεύθυνσης IP . . . . .	51
<b>8</b>	<b>Containerization και Deployment</b>	<b>53</b>
8.1	Containerization με Docker . . . . .	53
8.1.1	Δημιουργία Docker Image . . . . .	53
8.2	Deployment με Kubernetes . . . . .	55
8.2.1	Δημιουργία Kubernetes manifest files . . . . .	57
8.2.2	Πρόσβαση στο Django pod . . . . .	58
<b>9</b>	<b>Συμπέρασματα και Μελλοντική Εργασία</b>	<b>63</b>
9.1	Συμπεράσματα . . . . .	63
9.2	Προκλήσεις και Μαθήματα . . . . .	63
9.3	Μελλοντική Εργασία και Επέκταση Λειτουργικότητας . . . . .	64
<b>10</b>	<b>Παράρτημα</b>	<b>65</b>
10.1	Σχεδίαση και Διάγραμμα τοπολογίας δικτύου . . . . .	65
10.2	Οδηγός χρήσης της εφαρμογής . . . . .	65
10.2.1	Δημιουργία αντικειμένου-Προσθήκη συσκευής . . . . .	65
10.3	Κώδικας . . . . .	67
<b>11</b>	<b>Βιβλιογραφία</b>	<b>69</b>

11.1 Βιβλιογραφικές αναφορές . . . . .	69
--	----

# Κατάλογος Σχημάτων

4.1	Virtualization Γενική αρχιτεκτονική . . . . .	25
4.2	Virtualization Γενική αρχιτεκτονική . . . . .	26
4.3	Virtualbox . . . . .	27
5.1	MTV μοντέλο . . . . .	31
5.2	Διαγραμματική απεικόνιση της εφαρμογής . . . . .	32
5.3	Local PC-GNS3VM-CISCO IOS Connection Architecture . . . . .	34
5.4	Import appliance . . . . .	34
5.5	filename configuration . . . . .	35
5.6	ssh and credentials requirements . . . . .	35
6.1	url.py . . . . .	38
6.2	views.py . . . . .	38
6.3	Παράδειγμα html αρχείου . . . . .	39
6.4	Παράδειγμα wireshark . . . . .	40
6.5	Απάντηση συσκευής . . . . .	40
6.6	Παραμετροποίηση cloud . . . . .	41
6.7	Παραμετροποίηση eth0 . . . . .	42
6.8	Παραμετροποίηση DHCP . . . . .	42
6.9	IP connectivity test . . . . .	43
7.1	firstPage συνάρτηση . . . . .	45
7.2	Urls δρομολόγηση για την αρχική σελίδα. . . . .	45
7.3	Urls.py αρχείο . . . . .	46
7.4	Django run server . . . . .	46
7.5	Αρχική Σελίδα . . . . .	47
7.6	Αρχικοποίηση συσκευής . . . . .	48
7.7	Controller-Device Interfaces . . . . .	48
7.8	Interfaces . . . . .	49
7.9	Στατιστικά . . . . .	50
7.10	Στατιστικά διεπαφής . . . . .	51
7.11	Τρέχων Διαμόρφωση . . . . .	51
7.12	IP Διαμόρφωση . . . . .	52
7.13	IP Διαφορά . . . . .	52

8.1	Dockerfile . . . . .	54
8.2	Docker build-Δημιουργία του κοντεινερ . . . . .	55
8.3	Docker image list- . . . . .	55
8.4	Docker push- . . . . .	55
8.5	Minikube deployment . . . . .	57
8.6	Spec . . . . .	58
8.7	Manifest for Django pod . . . . .	58
8.8	Port forward traffic . . . . .	60
10.1	Network topology design . . . . .	65
10.2	GUI Login . . . . .	66
10.3	GUI Login second page . . . . .	66
10.4	Add device page . . . . .	67
10.5	github repo . . . . .	68



## Κεφάλαιο 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 τεχνολογιών, 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 προσομοίωσης 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 κεντριοποιημένη 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 αποτέλεσε 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 προσομοιώνουν 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 αποδείχθηκε 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.

## Κεφάλαιο 4

# Θεωρητικό υπόβαθρο

### 4.1 Web Framework και Βασικές Τεχνολογίες Ανάπτυξης

#### 4.1.1 Django Framework

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 Φραμεворκ 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 πρωτοκόλλα επικοινωνίας. Στο τμήμα 4.2.2 παρουσιάζονται οι βιβλιοθήκες που χρησιμοποιήθηκαν και κάποια βασικά χαρακτηριστικά τους.

#### 4.1.2 Διαχείριση εκδόσεων (Version Control) με Git και GitHub

Το Git 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 στον τοπικό υπολογιστή ανάπτυξης και στο περιβάλλον παραγωγής επιτρέπει την εύκολη και γρήγορη ανάπτυξη στο

περιβάλλον παραγωγής. τον ήδη δοκιμασμένο κώδικα στο τοπικό περιβάλλον ανάπτυξης. Η έκδοση παρέχει τη δυνατότητα επαναφοράς σε προηγούμενες εκδόσεις κώδικα σε περίπτωση που κάποιο άγνωστο σφάλμα εμφανιστεί σε μια νεότερη έκδοση.

## **4.2 Αυτοματοποίηση Δικτύων και Διασύνδεση με Δικτυακές Συσκευές**

### **4.2.1 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 προγραμματική συλλογή αυτών των πληροφοριών επιτρέπει και τον έλεγχο τους σε πραγματικό χρόνο.

Ο έλεγχος πληροφοριών σε πραγματικό χρόνο και η άμεση λήψη αποφάσεων, όταν μεταβάλλεται μια κρίσιμη παράμετρος, όπως το MTU, 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.

## **4.3 Βιβλιοθήκες της Python για Network automation**

### **4.3.1 Paramiko**

Το Paramiko 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 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, κάτι 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 (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 αφαίρεση 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. 14 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.

#### **4.3.4 Cisco IOS και REST APIs για Δικτυακές Συσκευές της Cisco**

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 προσομοιώσουμε τις Cisco συσκευές καθώς δεν υπήρχε διαθέσιμο πειραματικό υλικό τέτοιο που να μας επιτρέψει την προσομοίωση IOS. Τα CISCO IOU είναι εύκολα στην εγκατάσταση και την προσομοίωσή τους συνεπώς κάνουν και τη δημιουργία του testbed ευκολότερη.

Στο πλαίσιο της δημιουργίας της εφαρμογής αξιοποιήθηκε και το REST πρωτόκολλο. Ο λόγος που το χαρακτηρίζω ως REST είναι επειδή η διαδικασία αυτή βασίζεται στη θεμελιώδη αρχή του HTTP request-response μοντέλου: όταν ένας χρήστης εισάγει ένα URL στον browser, πραγματοποιείται ένα HTTP request προς τον server, ο οποίος στη συνέχεια επιστρέφει την κατάλληλη απάντηση, στην προκειμένη περίπτωση το κάθε συνάρτηση που υλοποιήσαμε στο `views.py` επιστρέφει ως HTTP response μια html σελίδα. Αυτό είναι ένας βασικός μηχανισμός της RESTful

αρχιτεκτονικής, καθώς επιτρέπει την ανταλλαγή δεδομένων μέσω σαφώς καθορισμένων διαδρομών (URLs) που δρομολογούνται από το Django.

Στην παρούσα διπλωματική εργασία αξιοποιήθηκαν URLs μέσα στην εφαρμογή Django. Μέσω των HTTP requests που πραγματοποιούσαμε από τον browser, μπορούσαμε να λαμβάνουμε πίσω την πληροφορία που επιθυμούσαμε. Στην προκειμένη περίπτωση, όταν αναφέρομαι στο REST, εννοώ ότι αν εισάγουμε στον browser ένα URL, όπως το `http://127.0.0.1:8000/manage/`, η εφαρμογή θα επεξεργαστεί το HTTP request και θα επιστρέψει ως απόκριση (HTTP response) το αρχείο `base1.html` το οποίο έχει οριστεί να επιστρέφεται στη συνάρτηση `index` του αρχείου `views.py`.

Περισσότερες λεπτομέρειες σχετικά με τη διαχείριση των URLs στη Django θα παρουσιαστούν σε επόμενο κεφάλαιο. Ωστόσο, η ουσία της αρχιτεκτονικής REST στην εφαρμογή μας έγκειται στο γεγονός ότι, δίνοντας συγκεκριμένα HTTP URLs στον browser, το Django επιστρέφει τα αντίστοιχα αποτελέσματα, ανάλογα με τη δρομολόγηση που έχει οριστεί στο αρχείο `urls.py`. Αυτό επιτρέπει την εύκολη και οργανωμένη επικοινωνία μεταξύ του frontend και του backend της εφαρμογής.

## **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 αυτόματα 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 περίπτωσης 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 Deploynnet 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 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 διευκολύνει τη

διαχείριση και τη μεταφορά εφαρμογών από τον έναν υπολογιστή ή διακομιστή στον άλλον. Στη διπλωματική μας χρησιμοποιήθηκε το docker daemon[13] του linux

#### **4.4.3 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.[14]

### **4.5 Προσομοίωση και Εικονικά Περιβάλλοντα**

#### **4.5.1 GNS3 και Εικονικά Δίκτυα**

Το GNS3 Είναι ένα εργαλείο προσομοίωσης δικτύων ανοικτού κώδικα που επιτρέπει στους χρήστες να προσομοιώνουν 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 προσομοιώνουν 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 2 τεχνολογίες του GNS3. Το GNS3 λογισμικό και το GNS3 VM. Τα δύο αυτά εργαλεία λογισμικού μας βοηθάνε στο κομμάτι της προσομοίωσης των δικτυακών συσκευών. Περισσότερες λεπτομέρειες για τη δημιουργία του testbed περιγράφονται στο κεφάλαιο 5.3

#### **4.5.2 Πρόγραμμα εικονοποίησης για το GNS3 VM-VirtualBox**

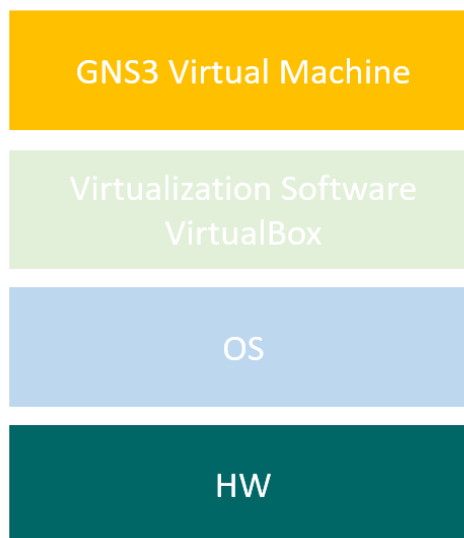
Στην επιστήμη της πληροφορικής, η εικονικοποίηση virtualization είναι ένας ευρύς όρος των υπολογιστικών συστημάτων που αναφέρεται σε έναν μηχανισμό αφαίρεσης, στοχευμένο στην απόκρυψη λεπτομερειών της υλοποίησης και της κατάστασης ορισμένων υπολογιστικών πόρων από πελάτες των πόρων αυτών (π.χ. εφαρμογές, άλλα συστήματα, χρήστες κλπ). Στη συγκεκριμένη εργασία, η εικονικοποίηση αξιοποιήθηκε για τη δημιουργία του εργαστηριακού περιβάλλοντος. Το GNS3 χρη-



σιμοποιήθηκε 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

Το Oracle VM VirtualBox 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.

Το VirtualBox μπορεί να εγκατασταθεί σε διάφορα λειτουργικά συστήματα, συμπεριλαμβανομένων 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.

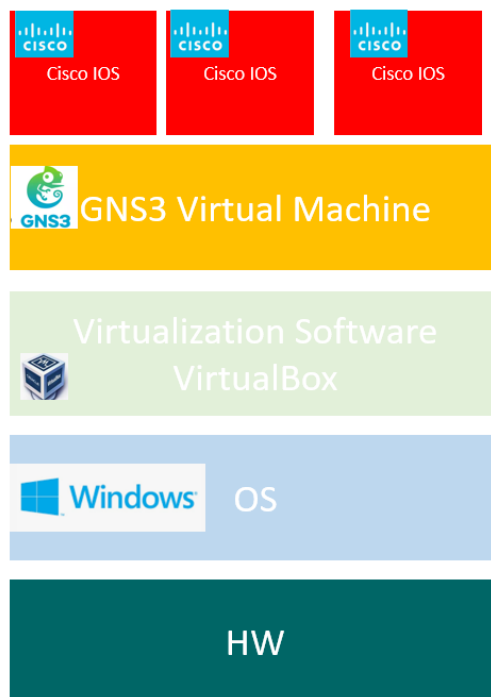


Σχήμα 4.1: Virtualization Γενική αρχιτεκτονική

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

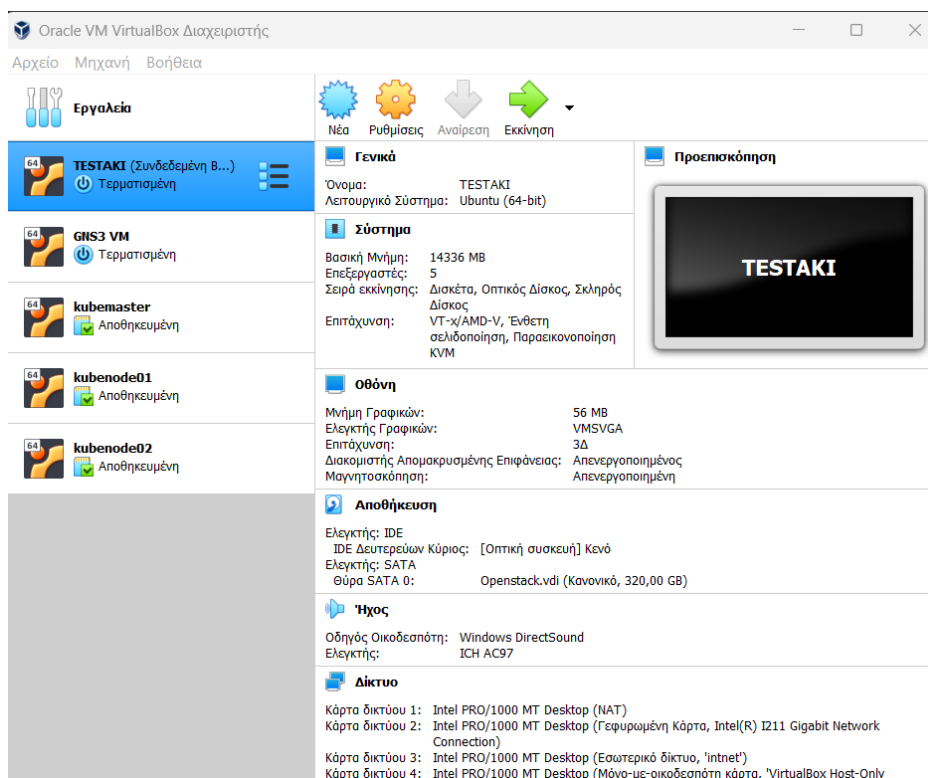
Το VirtualBox χρησιμοποιήθηκε συνεπώς προκειμένου να σηκώσουμε το GNS3

VM. Οι developers του GNS3 συνιστούν για τις περισσότερες περιπτώσεις την εγκατάσταση και του GNS3 VM όταν χρησιμοποιούμε Windows OS. 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 4.3 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.



Σχήμα 4.2: Virtualization Γενική αρχιτεκτονική

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



Σχήμα 4.3: Virtualbox



## Κεφάλαιο 5

# Σχεδίαση και Ανάλυση

### 5.1 Απαιτήσεις Συστήματος

#### 5.1.1 Λειτουργικές απαιτήσεις

Μια σειρά από λειτουργίες κρίθηκαν απαραίτητες και υλοποιήθηκαν 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
- Τέλος, θα δίνεται η δυνατότητα αλλαγής της διεύθυνσης IP μιας επιλεγμένης δικτυακής εφαρμογής, εξασφαλίζοντας μεγαλύτερη ευελιξία στη διαχείριση του δικτύου.

Όλες αυτές οι λειτουργίες έχουν σχεδιαστεί για να διευκολύνουν τη διαχείριση και την παρακολούθηση του δικτύου, ακόμα 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.

Στην εποχή μας, εμφανίστηκαν τα frameworks και λαμβάνοντας υπόψη ότι ουσιαστικά δεν βελτιώνουν αυτό που τελικά βλέπει ο χρήστης και τις αλληλεπιδράσεις του με το frontend, τότε θα μπορούσε κανείς να αναρωτηθεί γιατί χρησιμοποιούνται ευρέως στις μέρες μας. 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.

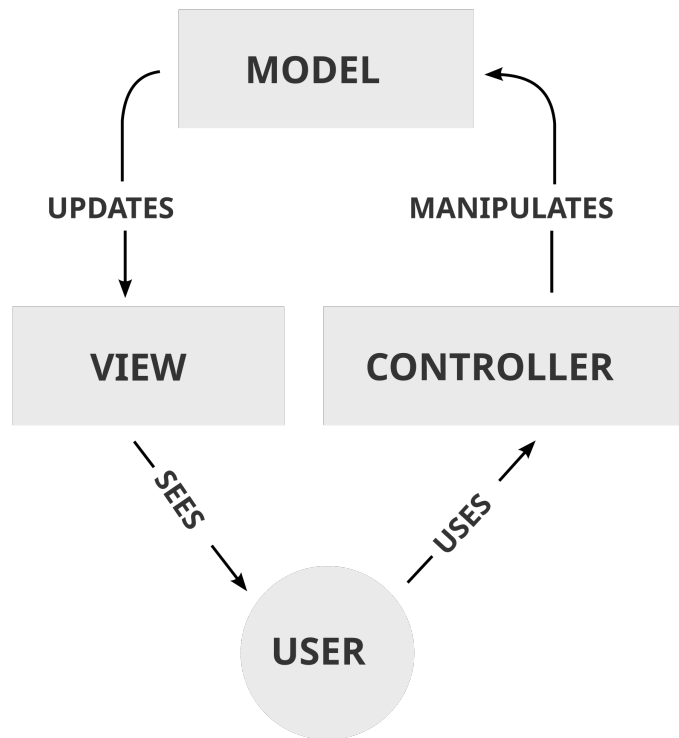
Με βάση άλλες τέτοιες προσπάθειες που έχουν γίνει στο παρελθόν εμείς συλλέξαμε την έως τώρα βιβλιογραφία και προσπαθήσαμε να φτιάξουμε μία τέτοια εφαρμογή η οποία όμως να βασίζεται στα τωρινά δεδομένα και να ενσωματώσουμε τις τελευταίες τεχνολογίες αιχμής όπως την Cloud Native αρχιτεκτονική. Στη συνέχεια γίνεται προσπάθεια να δοθεί 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[15] (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.

### 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.



Σχήμα 5.1: MTV μοντέλο

### 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.

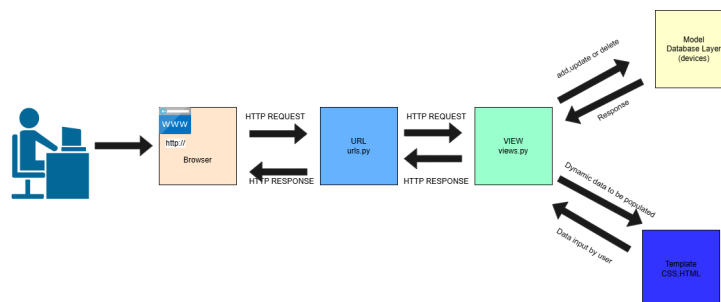
Το βασικό μοντέλο που χρησιμοποιείται εδώ είναι το Device, το οποίο αναπαριστά συσκευές όπως δρομολογητές και switches, αποθηκεύοντας πληροφορίες όπως η διεύθυνση του host, το όνομα χρήστη και ο κωδικός πρόσβασης, η πλατφόρμα της συσκευής και πρόσθετα στοιχεία ασφαλείας. Το Device είναι συνδεδεμένο με 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 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. 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 εύελικτη 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. Το Windows Subsystem for Linux version 2[11] 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 λει-



τουργιών (utility VM). Αυτό επιτρέπει στο WSL 2 να προσφέρει καλύτερη απόδοση, πλήρη συμβατότητα με τις λειτουργίες Linux και πρόσβαση σε εργαλεία όπως το Docker.

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 μπορούμε εύκολα να εγκαταστήσουμε με μία εντολή όλες τις βιβλιοθήκες που έχουμε βάλει μέσα σε αυτό το text αρχείο. Με την εντολή αυτή pip install -r requirements.txt εγκαθιστούμε αυτόματα όλο το απαραίτητο λογισμικό προκειμένου να τρέξει η εφαρμογή μας.

Η εξοικείωσή μου με περιβάλλοντα Unix έπαιξε καθοριστικό ρόλο στο να επιλέξω να φτιάξω το περιβάλλον αυτό, 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.

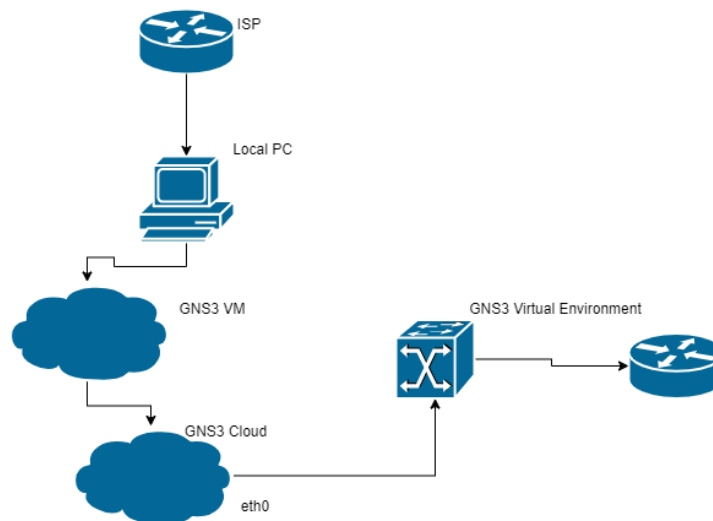
Το γεγονός ότι το WSL2 αποτελεί ένα Linux περιβάλλον που εκτελείται πάνω σε λειτουργικό σύστημα Windows είχε καθοριστική σημασία για την παρούσα υλοποίηση. Αυτό οφείλεται στο ότι τόσο το GNS3 όσο και το VirtualBox εγκαταστάθηκαν στο Windows περιβάλλον, ενώ το WSL2 αξιοποιήθηκε αποκλειστικά για την ανάπτυξη 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 προσομοίωσης δικτύου.

### **5.3.1 Σχεδίαση Τοπολογίας Δικτύου**

Η λογική την οποία χρησιμοποιήσαμε για τη δημιουργία του Testbed είναι ότι οι συσκευές οι οποίες τρέχουν πάνω στο GNS3 VM θα μπορέσουν να αλληλεπιδράσουν με το τοπικό WSL2 περιβάλλον. Προκειμένου να επιτευχθεί αυτός ο στόχος ακολουθήθηκαν βήματα. Το πρώτο βήμα είναι η εγκατάσταση βασικών προγραμμάτων τα οποία έχουν παρουσιαστεί στο Θεωρητικό υπόβαθρο.

Το πως μπορούν να επικοινωνήσουν συσκευές του GNS3 με το τοπικό περιβάλλον θα παρουσιαστεί σε επόμενο κεφάλαιο. Σε High Level overview η εφαρμογή

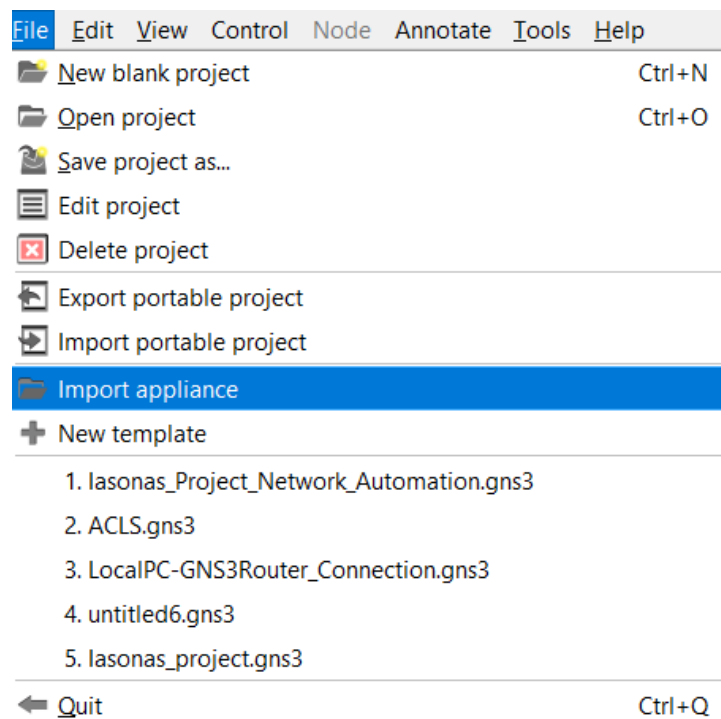
ακολουθεί την παρακάτω τοπολογία :



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

### 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 5.3)



Σχήμα 5.4: Import appliance

Ακολουθώντας πατάμε Install appliance on the GNS3 VM και ματσάροντας το filename του appliance με το image που έχουμε μας επιτρέπει να εισάγουμε τη

συσκευή.

Για να δούμε το filename στο appliance ανοίγουμε το αρχείο με έναν editor και αλλάζουμε το filename αντίστοιχα όπως στην εικόνα 5.4.

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

Σχήμα 5.5: filename configuration

Μετά το τέλος της διαδικασίας η συσκευή θα έχει προστεθεί και μπορούμε να την δούμε στην επιλογή Browse all devices. Συνεπώς θα μπορούμε να την προσθέσουμε σε τοπολογία και να την κάνουμε να δουλέψει.

Για να μπορέσουμε να δουλέψουμε με τη συσκευή θα πρέπει να έχουμε παραμετροποιήσει συγκεκριμένο username,password,secret καθώς και να τρέχει ssh service προκειμένου να μπορούμε να συνδεθούμε μέσα απο το API.

Στην παρακάτω εικόνα φαίνεται τι πρέπει να έχει υλοποιηθεί ως προϋπόθεση στην εφαρμογή.

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

Σχήμα 5.6: ssh and credentials requirements



## Κεφάλαιο 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`. Το αρχείο αυτό περιέχει μία κλάση `Device` η οποία δέχεται σαν ορίσματα το όνομα, την IP, το όνομα χρήστη, τον κωδικό, τον κρυφό κωδικό και το μοντέλο της συσκευής. Υπάρχει μία εγγραφή στη βάση δεδομένων ένα από αυτά τα αντικείμενα τα οποία εμείς τα δημιουργούμε. Το μοντέλο διαπιστευτηρίων αποθηκεύει τα διαπιστευτήρια τα οποία έχουν προηγουμένως κρυπτογραφημένα. Με αυτό, ορισμένες δέσμες ενεργειών μπορούν να λάβουν τα διαπιστευτήρια που απαιτούνται για να λειτουργήσουν χωρίς την είσοδο του χρήστη και χωρίς να γίνεται άμεση αναφορά στα διαπιστευτήρια στο κώδικα.

#### 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`. 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 διευθύνσεις 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

```
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 6.1: url.py

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 μιας συνάρτησης σε ένα αρχείο views.py μπορείτε να δείτε στο σχήμα παρακάτω

```
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 6.2: views.py

### 6.1.3 User Interface (Templates)

Στη συνάρτηση views.py, το Django αποδίδει το αντίστοιχο πρότυπο .html αρχείο με ένα συγκεκριμένο πλαίσιο. Το πλαίσιο είναι σε μορφή JavaScript Object Notation (JSON) και αποστέλλεται στο αρχείο .html. Τα δεδομένα στο πλαίσιο εμφανίζονται

στο .html, εάν το .html έχει παραμετροποιηθεί κατάλληλα. Ένα παράδειγμα .html με τον συντακτικό κώδικα για τον τρόπο πρόσβασης στα δεδομένα του πλαισίου παρουσιάζεται στην εικόνα παρακάτω

```
</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 6.3: Παράδειγμα html αρχείου

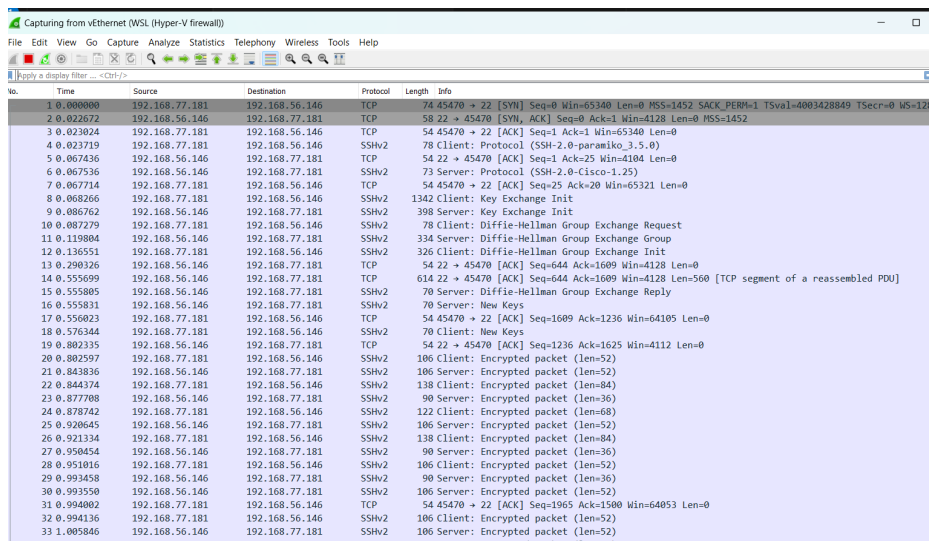
## 6.2 REST API Integration και διασύνδεση με το SSH protocol

Το REST API Integration διαδραματίζει κρίσιμο ρόλο στη διπλωματική εργασία, καθώς επιτρέπει την αλληλεπίδραση μεταξύ των διαφόρων συστημάτων και την ανταλλαγή δεδομένων μέσω του διαδικτύου. Το 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 ο χρήστης να μπορεί να φέρει στον χρήστη το σωστό αποτέλεσμα. Η διαχείριση 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 και στο Wireshark κάνουμε capture το interface eth0 του WSL προκειμένου να δούμε τι γίνεται όταν εμείς πατάμε το κουμπί για μια

λειτουργία. Στο παρακάτω trace φαίνεται τι γίνεται:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.77.181	192.168.56.146	TCP	74	45470 → 22 [SYN] Seq=0 Win=65536 Len=0 MSS=1452 SACK_PERM=1 TSval=4803428849 TSecr=0 WS=128
2	0.022672	192.168.56.146	192.168.77.181	TCP	58	22 → 45470 [SYN, ACK] Seq=0 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	0.023719	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 → 45470 [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-1.25)
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	192.168.56.146	192.168.77.181	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	192.168.56.146	192.168.77.181	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	192.168.56.146	192.168.77.181	TCP	54	22 → 45470 [ACK] Seq=644 Ack=1609 Win=4128 Len=0
14	0.555699	192.168.56.146	192.168.77.181	TCP	614	22 → 45470 [ACK] Seq=644 Ack=1609 Win=4128 Len=560 [TCP segment of a reassembled PDU]
15	0.555805	192.168.56.146	192.168.77.181	SSHv2	70	Server: Diffie-Hellman Group Exchange Reply
16	0.555831	192.168.56.146	192.168.77.181	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	0.576344	192.168.77.181	192.168.56.146	SSHv2	70	Client: New Keys
19	0.802135	192.168.56.146	192.168.77.181	TCP	54	22 → 45470 [ACK] Seq=1236 Ack=1625 Win=4112 Len=0
20	0.802597	192.168.77.181	192.168.56.146	SSHv2	106	Client: Encrypted packet (len=52)
21	0.843836	192.168.56.146	192.168.77.181	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	0.877708	192.168.56.146	192.168.77.181	SSHv2	90	Server: Encrypted packet (len=36)
24	0.878742	192.168.77.181	192.168.56.146	SSHv2	122	Client: Encrypted packet (len=68)
25	0.920645	192.168.56.146	192.168.77.181	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	0.950454	192.168.56.146	192.168.77.181	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	192.168.56.146	192.168.77.181	SSHv2	90	Server: Encrypted packet (len=36)
30	0.993550	192.168.77.181	192.168.56.146	SSHv2	106	Server: Encrypted packet (len=52)
31	0.994002	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	192.168.56.146	192.168.77.181	SSHv2	106	Server: Encrypted packet (len=52)

Σχήμα 6.4: Παράδειγμα wireshark

Αυτή η αλληλουχία αντιπροσωπεύει μια τυπική ασφαλή σύνδεση SSH. Εγκαθίδρυση TCP σύνδεσης (SYN,SYN/ACK,ACK). 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 απο το cli του Django server στην παρακάτω εικόνα.

```
[30/Nov/2024 19:37:18] "GET /static/style.css HTTP/1.1" 200 1625
{"FastEthernet0/0": {"is_enabled": True, "is_up": True, "description": "", "mac_address": "CA:02:09:97:08:08", "last_flapped": -1.0, "mtu": 1500, "speed": 100.0}, "FastEthernet2/0": {"is_enabled": True, "is_up": True, "description": "", "mac_address": "CA:02:09:97:08:08", "last_flapped": -1.0, "mtu": 1500, "speed": 100.0}, "FastEthernet2/1": {"is_enabled": True, "is_up": True, "description": "", "mac_address": "CA:02:09:97:08:08", "last_flapped": -1.0, "mtu": 1500, "speed": 100.0}, "Serial1/0": {"is_enabled": False, "is_up": False, "description": "", "mac_address": "", "last_flapped": -1.0, "mtu": 1500, "speed": 1.544}, "Serial1/1": {"is_enabled": False, "is_up": False, "description": "", "mac_address": "", "last_flapped": -1.0, "mtu": 1500, "speed": 1.544}, "Serial3/2": {"is_enabled": False, "is_up": False, "description": "", "mac_address": "", "last_flapped": -1.0, "mtu": 1500, "speed": 1.544}, "Serial3/3": {"is_enabled": False, "is_up": False, "description": "", "mac_address": "", "last_flapped": -1.0, "mtu": 1500, "speed": 1.544}, "POS4/0": {"is_enabled": False, "is_up": False, "description": "", "mac_address": "", "last_flapped": -1.0, "mtu": 4470, "speed": 135.0}}
[30/Nov/2024 19:37:30] "GET /device/6 HTTP/1.1" 200 5597
[30/Nov/2024 19:42:06] "GET / HTTP/1.1" 200 1303
[30/Nov/2024 19:42:06] "GET /static/style.css HTTP/1.1" 200 10755
[30/Nov/2024 19:42:06] "GET /static/plexus-atoms-neutrons-electrons-wallpaper-preview.jpg HTTP/1.1" 200 35128
```

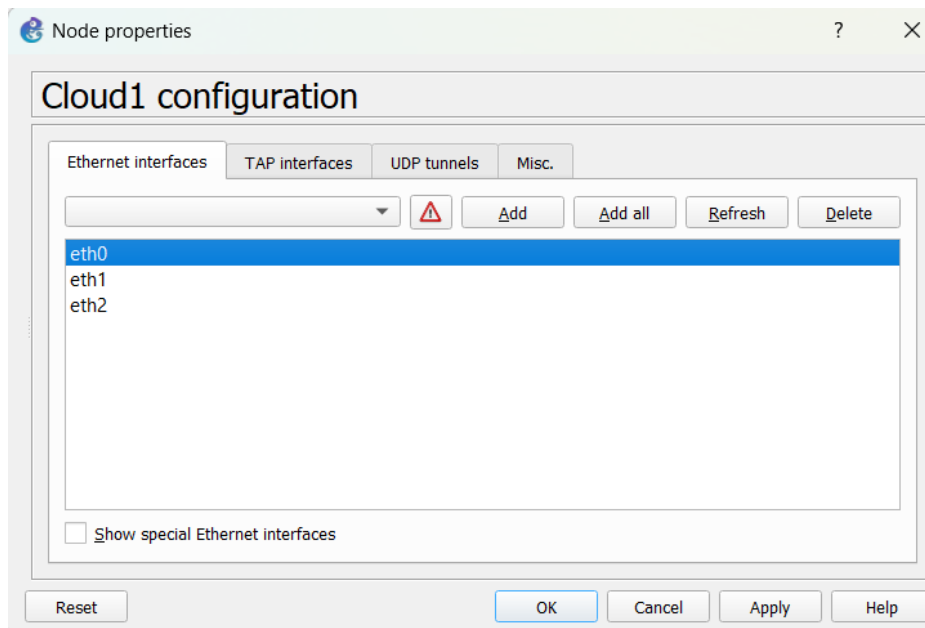
Σχήμα 6.5: Απάντηση συσκευής

## 6.3 Διασύνδεση με το GNS3

Παρακάτω θα παρουσιαστούν τα βήματα που ακολουθήθηκαν προκειμένου να μπορέσει να μιλήσει το τοπικό δίκτυο με το δίκτυο των εικονικών συσκευών. Προκειμένου να γίνει η σύνδεση της εφαρμογής με το GNS3 γίνανε αρκετές ρυθμίσεις.

Πρώτα σε επίπεδο συνδεσιμότητας. Σύνδεση μεταξύ IOU2 και Cloud. Το IOU2 είναι ένα Switch το οποίο θα μας βοηθήσει να έχουμε επαφή με L3 Devices αφού όλες οι συσκευές που ενώνονται μαζί του είναι στο ίδιο L2 network Το cloud αναπαριστά το τοπικό router μας. Είναι η επαφή μας με το τοπικό δίκτυο του GNS3 server. Λόγω αυτού θα μπορέσει ο διακομιστής που τρέχει στο WSL περιβάλλον να επικοινωνήσει με τις συσκευές στο GNS3. Παρατίθεται το cloud configuration:



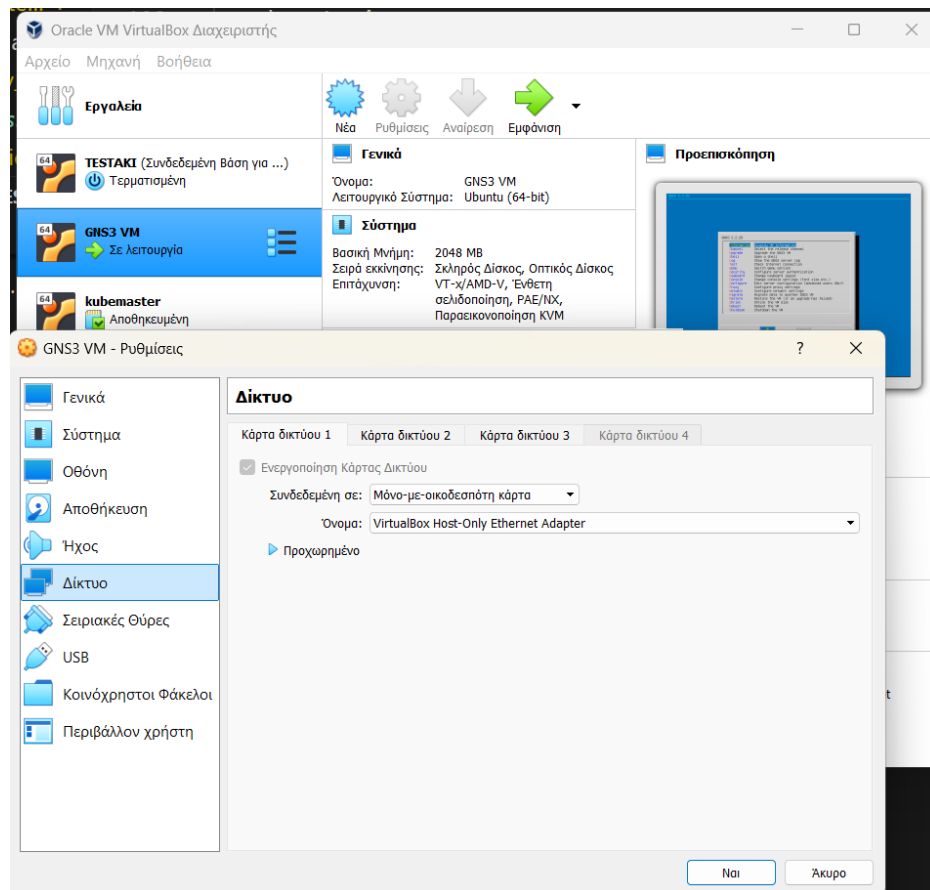


Σχήμα 6.6: Παραμετροποίηση cloud

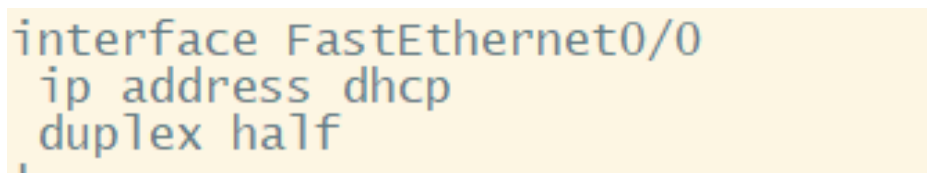
Δεξί κλικ και πατάμε παραμετροποίηση. Εκεί κάνουμε προσθήκη το eth0 ως διεπαφή επικοινωνίας του τοπικού δικτύου με τον διακομιστή. Το eth0 είναι κρίσιμης σημασίας επιλογή και ο λόγος που το επιλέγουμε είναι διότι στο GNS3 VM το eth0 έχει την παρακάτω παραμετροποίηση.

Συνεπώς επειδή αναπαριστά μια κάρτα δικτύου η οποία συνδέει την εικονική μας μηχανή με τον υπολογιστή στον οποίο εκτελείται αυτή. Συνεπώς οι συσκευές οι οποίες συνδέονται με το cloud θα έχουν απευθείας σύνδεση και με το τοπικό μας δίκτυο.

Προκειμένου τώρα οι συσκευές αυτές να μπορέσουν να μιλήσουν με το τοπικό μας δίκτυο θα πρέπει να τους ορίσουμε IP διευθύνσεις. Παρακάτω φαίνεται η παραμετροποίηση που έγινε στη συσκευή προκειμένου να πάρει IP διευθύνση από το τοπικό μας δίκτυο(192.168.56.0/24)



Σχήμα 6.7: Παραμετροποίηση eth0



Σχήμα 6.8: Παραμετροποίηση DHCP

Έλεγχος ότι μπορούμε να επικοινωνήσουμε με τις εικονικές συσκευές της Cisco

The image shows a GNS3 terminal window and a Session Manager window. The terminal window displays the configuration of a network interface (eth0) and a ping test to 192.168.56.146. The Session Manager window shows the network topology with various interfaces and their status.

**Terminal Output:**

```

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 s
tate 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
iasonas@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
iasonas@DESKTOP-9M04JK8:~$

```

**Session Manager:**

Filter by folder/session name <...>

- Sessions
  - K8s
    - c-plane-k8s
    - worker1-k8s
  - Microk8s
    - c-plane
    - worker1
  - GNS3-VM
  - LAB2
  - SLES12
  - TESTAKI

**Interface Status Table:**

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
Serial3/0	unassigned	YES	NVRAM	administratively down	down
Serial3/1	unassigned	YES	NVRAM	administratively down	down
Serial3/2	unassigned	YES	NVRAM	administratively down	down
Serial3/3	unassigned	YES	NVRAM	administratively down	down
POS4/0	unassigned	YES	NVRAM	administratively down	down

Send commands to active session

Ready Telnet: 192.168.56.132 24, 4 24 Rows, 80 Cols Xterm CAP NUM

Σχήμα 6.9: IP connectivity test



## Κεφάλαιο 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 ως μία συνάρτηση της Python η οποία δέχεται ένα http request και σαν απάντηση επιστρέφει την αρχική σελίδα(index.html).

Αυτό γίνεται μέσα από 2 κύριως μηχανισμούς. Η συνάρτηση firstPage ορίζεται στο αρχείο views.py της εφαρμογής. Παίρνει ως όρισμα ένα αντικείμενο HttpRequest (την αίτηση του χρήστη) και επιστρέφει ένα HttpResponse μέσω της ρενδερ:

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

Σχήμα 7.1: firstPage συνάρτηση

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. Παρακάτω 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 χτίσεις 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 παρακάτω στην εικόνα 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 η οποία βρίσκεται στην εικόνα 5.1.

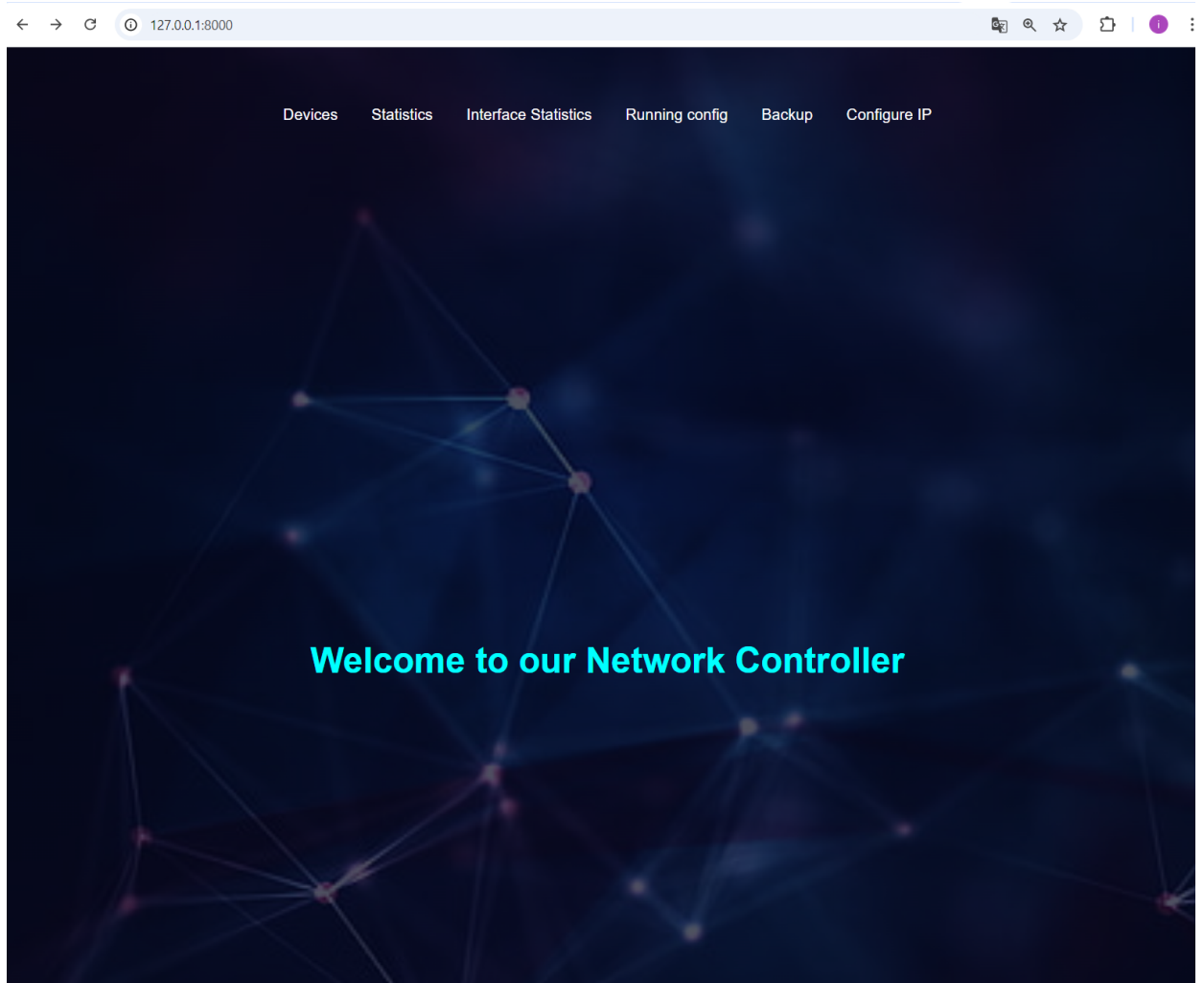
```
^Ciasonas@DESKTOP-9M04JK8:~/SDN_Django_framework_for_implementation_network_service_configuration_application$ python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

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

Σχήμα 7.4: Django run 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 5.2. Το εμπρόστιο τμήμα της εφαρμογής(frontend) είναι γραμμένο με HTML,CSS ενώ το οπίσθιο τμήμα(backend) είναι γραμμένο σε Python.



Σχήμα 7.5: Αρχική Σελίδα

### 7.3 Devices

Προκειμένου να δημιουργήσουμε μία νέα συσκευή η παρακατω κλάση κώδικα μας βοηθάει στο να γίνει όπως στο σχήμα 5.3

```

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 συσκευής

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 5.4.

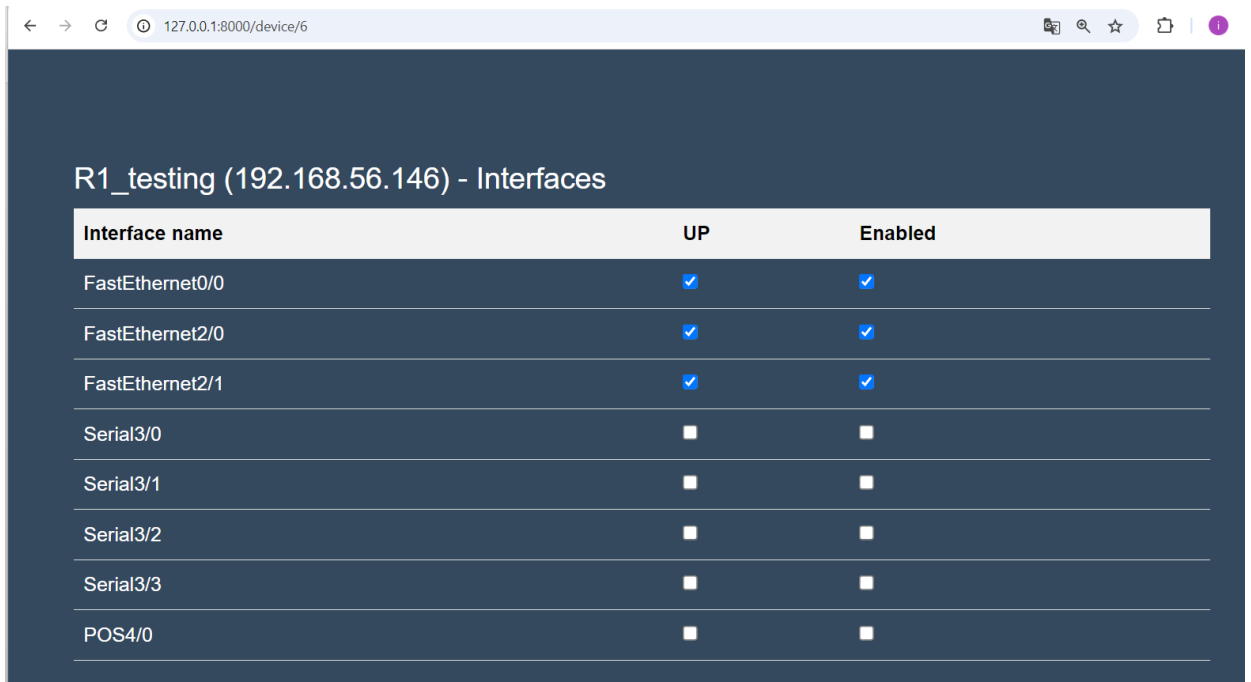
Αποτέλεσμα του πίνακα του σχήματος 5.4 είναι όλες οι συσκευές οι οποίες είναι στη βάση δεδομένων μας όπου ο χρήστης πρόσθεσε προκειμένου να μπορεί να αναδράσει με αυτές.

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

Πατώντας ένα από αυτά τα κουμπιά θα μπορέσουμε να πάρουμε το αποτέλεσμα που θέλουμε. Σε αυτή τη σελίδα μπορούμε να δούμε αν κάποια διεπαφή αν λειτουργεί ή όχι. (Σχήμα 5.5)





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

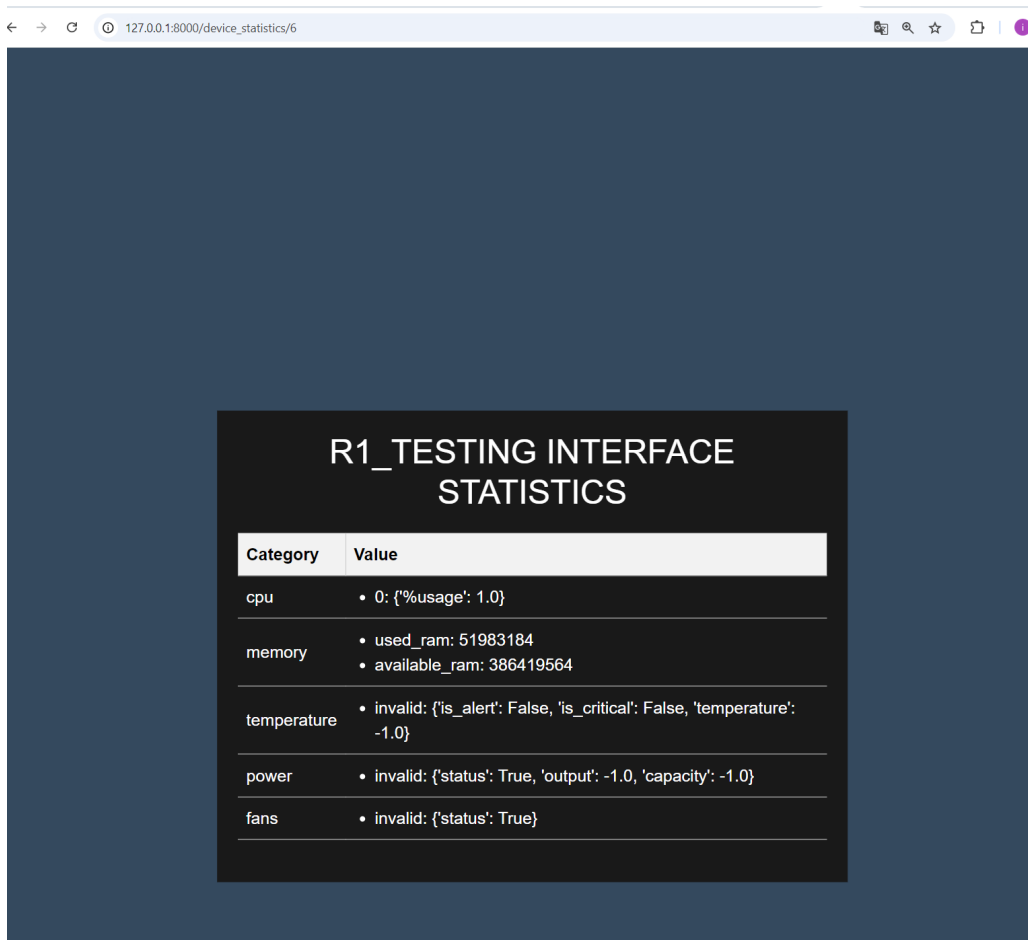
## 7.4 Στατιστικά της συσκευής

Η λειτουργία του κουμπιού αυτού 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 αίτημα 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 παραπάνω (Devices), και πατώντας πάνω της εμφανίζει τα παρακάτω στατιστικά:

## 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



Σχήμα 7.9: Στατιστικά

## 7.6 Backup της συσκευής

Προκειμένου να σώσουμε τη διαμόρφωση της συσκευής πατάμε το κουμπί backup και με αυτό πέρνουμε το παρακάτω αποτέλεσμα.

Ουσιαστικά είναι σε text αρχείο το λεγόμενο running config της συσκευής

Σχήμα 7.10: Στατιστικά διεπαφής

Σχήμα 7.11: Τρέχων Διαμόρφωση

Η διαμόρφωση IP διεύθυνσης γίνεται δίνοντας τα παρακάτω στοιχεία σαν είσοδο  
Και καταφέρνει και αλλάζει την διεύθυνση.

← → ↺ ⓘ 127.0.0.1:8000/configure-ip/

## Configure IP on Cisco Device

Device Hostname:

Username:

Password:

Interface:

IP Address:

Subnet Mask:

Σχήμα 7.12: IPΔιαμόρφωση

```
R1#show ip in
R1#show ip int
R1#show ip interface br
R1#show ip interface brief
Interface IP-Address OK? Method Status Protocol
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
Serial3/0 unassigned YES NVRAM administratively down down
Serial3/1 unassigned YES NVRAM administratively down down
Serial3/2 unassigned YES NVRAM administratively down down
Serial3/3 unassigned YES NVRAM administratively down down
POS4/0 unassigned YES NVRAM administratively down down
R1#
R1#show ip interface brief
Interface IP-Address OK? Method Status Protocol
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
Serial3/0 unassigned YES NVRAM administratively down down
Serial3/1 unassigned YES NVRAM administratively down down
Serial3/2 unassigned YES NVRAM administratively down down
Serial3/3 unassigned YES NVRAM administratively down down
POS4/0 unassigned YES NVRAM administratively down down
R1#
*Oct 21 17:54:15.803: %SYS-5-CONFIG_I: Configured from console by iasonas on vty0 (192.168.56.1)
R1#
*Oct 21 17:54:17.395: %SYS-5-CONFIG_I: Configured from console by iasonas on vty1 (192.168.56.1)
R1#
```

Σχήμα 7.13: IP Διαφορά

## Κεφάλαιο 8

# Containerization και Deployment

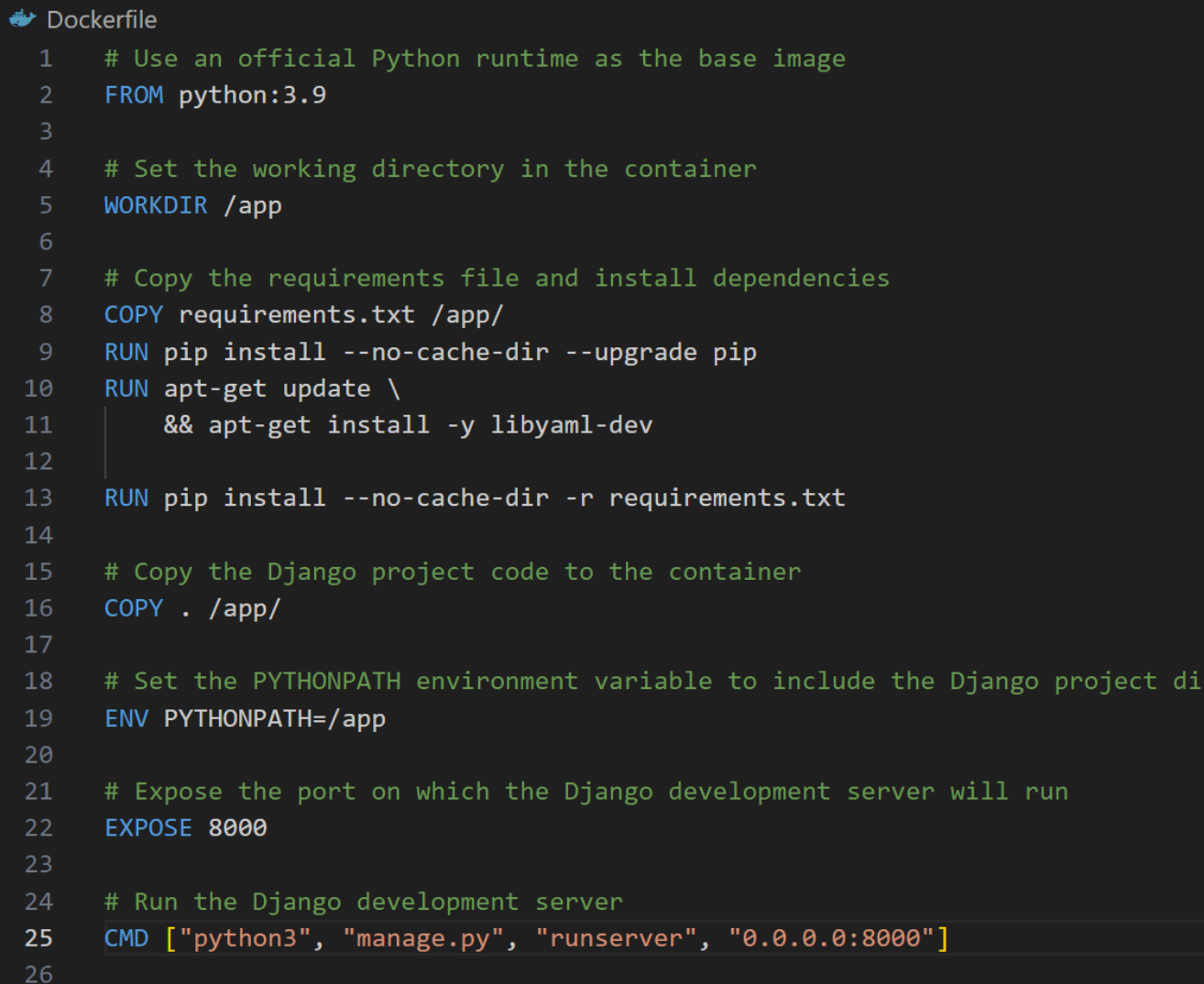
### 8.1 Containerization με Docker

#### 8.1.1 Δημιουργία Docker Image

Προκειμένου να μπορέσουμε να χρησιμοποιήσουμε το Docker Desktop θα πρέπει να έχουμε φτιάξει την εφαρμογή μας σαν κοντεινερ.

Για να γίνει αυτό θα πρέπει να φτιάξουμε το παρακάτω Dockerfile το οποίο ουσιαστικά είναι η εφαρμογή μας αλλά σε κοντεινερ.

Αυτό το Dockerfile 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, 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.

A screenshot of a Dockerfile with 26 lines of code. The code is color-coded: comments are green, FROM, COPY, and CMD are blue, and WORKDIR, RUN, and EXPOSE are green. The code sets up a Python 3.9 environment, installs dependencies from requirements.txt, copies the Django project code, sets the PYTHONPATH, and runs the Django development server on port 8000.

```
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 di
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

Η εντολή `docker build` 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 διαμοιραστεί 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 παρακάτω εντολή.

```

iasonas@DESKTOP-9M04JK8:~/SDN_Django_framework_for_implementation_network_service_configuration_application$ docker build -t django:v1 .
[+] Building 36.8s (12/12) FINISHED
=> [Internal] load build definition from Dockerfile
=> => transferring dockerfile: 743B
=> [Internal] load .dockerignore
=> => transferring context: 2B
=> [Internal] load metadata for docker.io/library/python:3.9
=> [1/7] FROM docker.io/library/python:3.9@sha256:ed8b9dd4e9f89c11f4bdb85a55f8c9f0e22796a298449380b15f627d9914
=> [Internal] load build context
=> => transferring context: 14.18kB
=> CACHED [2/7] WORKDIR /app
=> [3/7] COPY requirements.txt /app/
=> [4/7] RUN pip install --no-cache-dir --upgrade pip
=> [5/7] RUN apt-get update && apt-get install -y libyaml-dev
=> [6/7] RUN pip install --no-cache-dir -r requirements.txt
=> [7/7] COPY . /app/
=> exporting to image
=> exporting layers
=> writing image sha256:67c01544160e308aea5622d3de55488251b54e023a607e2d2fffcf4886e53788
=> naming to docker.io/library/django:v1

```

Σχήμα 8.2: Docker build-Δημιουργία του κοντεινερ

Θα πρέπει μετά να βάλουμε το image στο DockerDesktop και αφού γίνει αυτό θα μπορέσουμε να την τρέξουμε στην υποδομή του κυβερνήτη. Το ίδιο μπορεί να γίνει και χωρίς την εφαρμογή DockerDesktop αλλά τρέχοντας το Docker service απευθείας πάνω στο WSL2 όπως φαίνεται και στις παρακάτω εικόνες.

```

root@DESKTOP-9M04JK8:~# docker image list
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
vrnetlab/cisco_iol  17.12.01           70c8af994324       4 weeks ago        704MB
iasonasi/django_thesis  latest             851c7a4ba667       5 weeks ago        1.13GB
public.ecr.aws/docker/library/debian bookworm-slim      086cc0a12e56       6 weeks ago        74.8MB
ubuntu              20.04              6013ae1a63c2       7 weeks ago        72.8MB
ghcr.io/nokia/srlinux latest              f23957871f9f       2 months ago       2.66GB
gcr.io/k8s-minikube/kicbase v0.0.39            67a4b1138d2d       20 months ago     1.05GB
root@DESKTOP-9M04JK8:~#

```

Σχήμα 8.3: Docker image list-

Παρακάτω παρουσιάζεται πως μπορούμε να κάνουμε push την εικόνα σε αποθετήριο σε περίπτωση που θα μπορούσαμε να χρησιμοποιήσουμε το Docker Desktop.

```

iasonas@DESKTOP-9M04JK8:~/SDN_Django_framework_for_implementation_network_service_configuration_application$ docker push s
Using default tag: latest
The push refers to repository [docker.io/iasonasi/django_thesis]
4eb48115a042: Pushed
164a177ac4f1: Pushed
da802df85c96: Pushed
63dc518f902b: Pushed
48d045e2a1f0: Pushed
dac6f4a56d09: Pushed
274fbebada99: Pushed
69470c0633ea: Pushed
249a2754c71b: Pushed
cc6cc6c1c4bf: Pushed
7d98d813d54f: Pushed
ad1c7cfc347f: Pushing [=====] 138.4MB/211.3MB
7aad5092c3b: Pushed
0b41a54d942d: Pushed

```

Σχήμα 8.4: Docker push-

## 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 τεχνολο-

γιών 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, κάτι 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.

Έτσι έρχεται να σώσει την κατάσταση ο κυβερνήτης. Το Kubernetes σάς παρέχει ένα πλαίσιο για να εκτελείτε τα καταναμεμημένα συστήματα με ευελιξία. Φροντίζει για την κλιμάκωση και το failover για την εφαρμογή σας, παρέχει μοτίβα ανάπτυξης και πολλά άλλα. Δε θα αναλύσουμε με κάθε λεπτομέρεια τι ακριβώς είναι ο κυβερνήτης γιατί μία τέτοια προσπάθεια είναι μία διπλωματική από μόνη της αλλά θα προσπαθήσουμε να παρουσιάσουμε τα βασικά χαρακτηριστικά τα οποία χρησιμοποιήθηκαν πρακτικά στη διπλωματική.

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 ενорχήστρωσης 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, 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.

```
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=v
Using the docker driver based on existing profile
Starting "minikube" primary control-plane node in "minikube" cluster
Pulling base image v0.0.45 ...
docker "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
ster 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-c4rzr                      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 8.5: Minikube deployment

### 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 τον τύπο πόρου ως Pod και εκχωρώντας 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.

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

Σχήμα 8.6: Spec

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.

```
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-c4rzzr                     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:~$
iasonas@DESKTOP-9M04JK8:~$ cat SDN_Django_framework_for_implementation_network_service_configuration_app
od1.yml
apiVersion: v1
kind: Pod
metadata:
  name: djangotestapp
spec:
  containers:
  - name: djangotestapp
    image: iasonasi/djangotestapp:latest
    ports:
    - containerPort: 8000
iasonas@DESKTOP-9M04JK8:~$
```

Σχήμα 8.7: 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`.

Βρείτε το pod και με την εντολή `kubectl port-forward` θα μπορέσουμε μέσα από έναν browser να έχουμε πρόσβαση σε αυτήν.

127.0.0.1:8000/device/1

GNS3

iasonas@DESKTOP-9M04JK8: x

iasonas@DESKTOP-9M04JK8: x

minikube Ready control-plane 26d v1.26.3

iasonas@DESKTOP-9M04JK8:~\$ kubectl get pods -A

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	djangotestapp	1/1	Running	1 (35s ago)	26d
kube-system	coredns-787d4945fb-2kwmn	0/1	Running	1 (35s ago)	26d
kube-system	coredns-787d4945fb-htlz6	0/1	Running	1 (35s ago)	26d
kube-system	etcd-minikube	1/1	Running	1 (35s ago)	26d
kube-system	kube-apiserver-minikube	1/1	Running	1 (35s ago)	26d
kube-system	kube-controller-manager-minikube	1/1	Running	1 (35s ago)	26d
kube-system	kube-proxy-c4rzt	1/1	Running	1 (35s ago)	26d
kube-system	kube-scheduler-minikube	1/1	Running	1 (35s ago)	26d
kube-system	storage-provisioner	1/1	Running	2 (35s ago)	26d

iasonas@DESKTOP-9M04JK8:~\$ kubectl port-forward pod/djangotestapp 8000:8000

Forwarding from 127.0.0.1:8000 -> 8000

Forwarding from [::1]:8000 -> 8000

Handling connection for 8000

Handling connection for 8000

R1 (192.168.56.146) - Interfaces

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 8.8: Port forward traffic

Στη συνέχεια, επισκεφτείτε την εφαρμογή σας στον browser μέσω της διεύθυνσης `http://localhost:8000`. Θα πρέπει όλες οι λειτουργίες να μπορούν να εκτελεστούν κάτι το οποίο η παραπάνω εικόνα το αποδεικνύει.



## **Κεφάλαιο 9**

# **Συμπέρασματα και Μελλοντική Εργασία**

### **9.1 Συμπεράσματα**

Η διπλωματική αυτή εργασία επικεντρώθηκε στην ανάπτυξη ενός εργαλείου αυτοματοποίησης δικτύου με χρήση της γλώσσας προγραμματισμού Python. Κατά τη διάρκεια αυτής της διαδικασίας, καταφέραμε να αναπτύξουμε και να εφαρμόσουμε διάφορες τεχνολογίες για την αυτοματοποίηση δικτυακών εργασιών, όπως η διαμόρφωση συσκευών και η παρακολούθηση της απόδοσης του δικτύου. Το εργαλείο αναπτύχθηκε χρησιμοποιώντας σαν TestBed το GNS3 και το Cisco IOU, και παρέχει σημαντικά οφέλη στον τομέα της δικτύωσης.

Η εφαρμογή αυτή έχει σημαντική χρησιμότητα στον τομέα της δικτύωσης, κυρίως για οργανισμούς που αναζητούν λύσεις αυτοματοποίησης και διαχείρισης δικτύων με σκοπό τη βελτίωση της απόδοσης και την εξοικονόμηση χρόνου. Μπορεί να εφαρμοστεί σε διάφορα περιβάλλοντα και να επεκταθεί για να καλύψει νέες ανάγκες, όπως η ενσωμάτωση νέων πρωτοκόλλων και εργαλείων παρακολούθησης.

Όπως είπαμε στην αρχή οι τεχνολογίες Cloud Native δεν εισάχτην προκειμένου να γίνουν μετρήσεις ταχύτητας μεταξύ της εφαρμογής σε περιβάλλον τοπικό και σε περιβάλλον κυβερνήτη. Η εισαγωγή αυτής της τεχνολογίας έγινε προκειμένου και μόνο να μπορούμε να διαχειριστούμε την εφαρμογή ως προς τα πλαίσια της κλιμάκωσης και της φορητότητάς της. Συνεπώς δεν ήταν στόχος μας να παρουσιάσουν πολιτικές εκτενείς διαφορές και πειραματισμοί κάτι το οποίο από μόνο του θα μπορούσε να αποτελέσει ένα άλλο θέμα για διπλωματική διατριβή μεταπτυχιακού επιπέδου.

### **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 απαιτήσε

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 πάνω στο οποίο τεσταρίστηκε η εφαρμογή θα μπορούσε να είναι διαφορετικό. Μελλοντική εργασία μπορεί να αναπτυχθεί με τη βοήθεια καινούργιων περιβάλλοντων προσομοίωσης όπως το ContainerLab. Η περαιτέρω ανάπτυξη λειτουργιών της εφαρμογής, η δημιουργία καλύτερου 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.

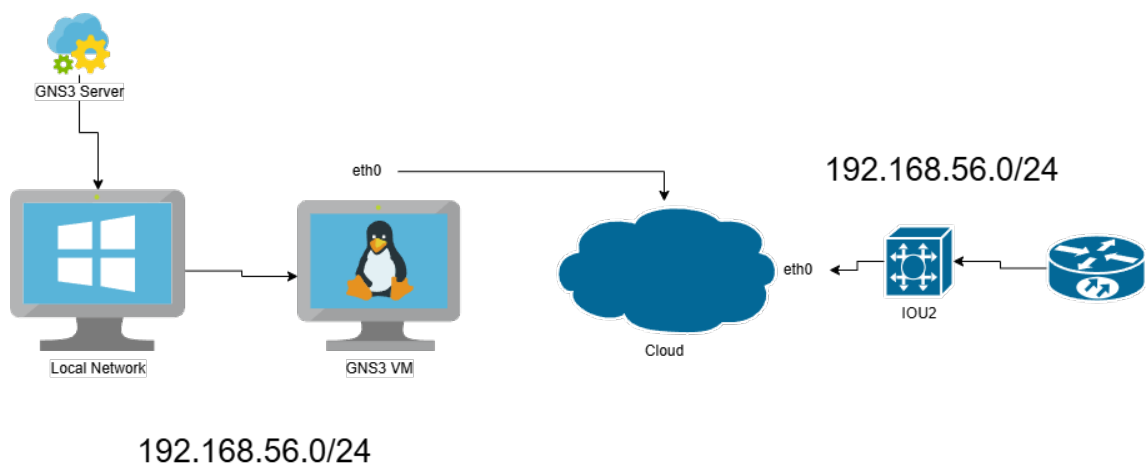
Τέλος, η μελλοντική έρευνα μπορεί να εξετάσει τη δυνατότητα διασύνδεσης με πλατφόρμες μηχανικής μάθησης, για την πρόβλεψη και αποτροπή πιθανών αποτυχιών, καθιστώντας την εφαρμογή ένα σύγχρονο εργαλείο αυτοματοποιημένης διαχείρισης δικτύου



## Κεφάλαιο 10

## Παράρτημα

### 10.1 Σχεδίαση και Διάγραμμα τοπολογίας δικτύου



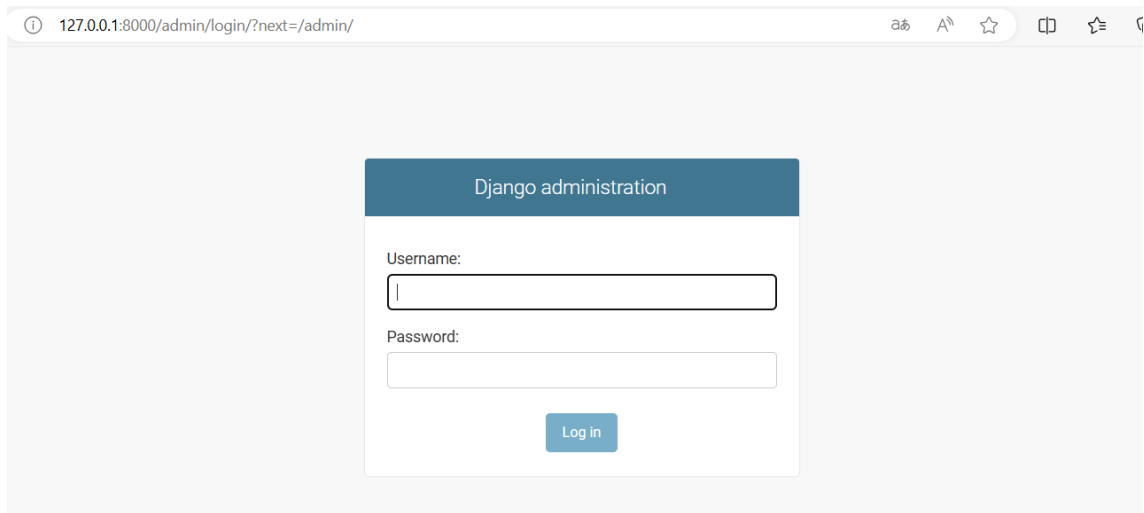
Σχήμα 10.1: Network topology design

### 10.2 Οδηγός χρήσης της εφαρμογής

#### 10.2.1 Δημιουργία αντικειμένου-Προσθήκη συσκευής

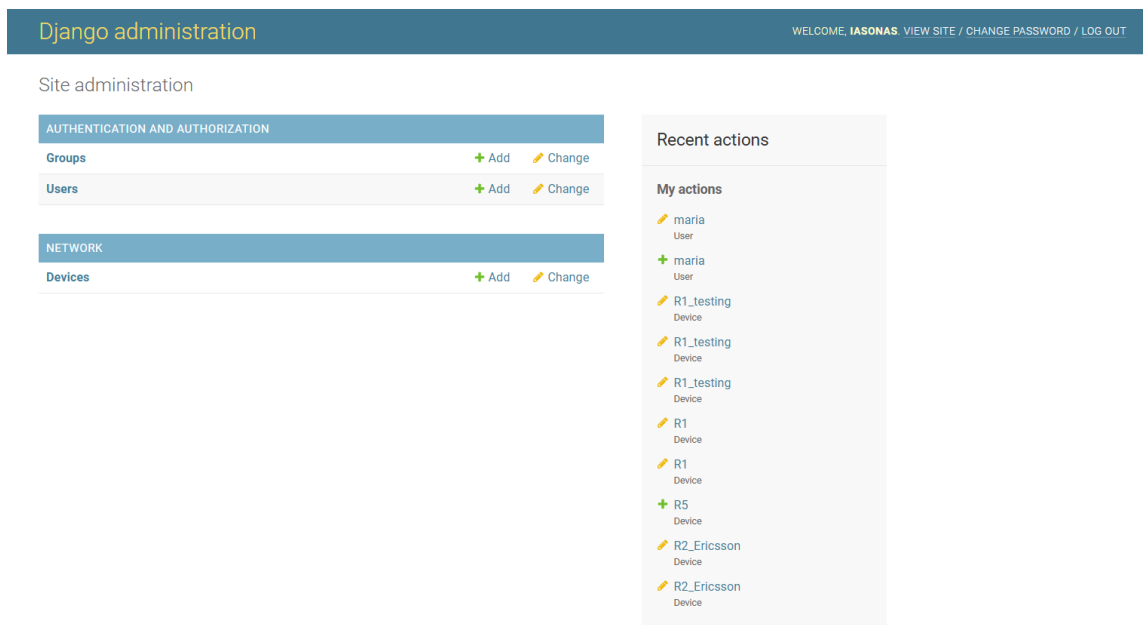
Προκειμένου να μπορούμε να διαχειριστούμε συσκευή θα πρέπει να την εισάγουμε ως αντικείμενο.

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.

Django administration

WELCOME, IASONAS. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > Network > Devices > Add device

**AUTHENTICATION AND AUTHORIZATION**

- Groups [+ Add](#)
- Users [+ Add](#)

**NETWORK**

- Devices [+ Add](#)

### Add device

Name:

Host:

Username:

Password:

Secret:

Device type:

Platform:

[Save and add another](#) [Save and continue editing](#) [SAVE](#)

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.

## 10.3 Κώδικας

Η εφαρμογή είναι ελεύθερη στη σελίδα :

The screenshot shows a GitHub repository page. The repository name is 'SDN\_Django\_framework\_for\_implementation\_network\_service\_configuration\_application' by user 'lasimo92'. The repository is public and has 59 commits. The file list includes folders for 'SDN\_Django\_framework\_for\_implementation...', 'configs', 'network', and 'yaml', and files for 'Configuration.txt', 'Dockerfile', 'Dockerfile\_backup', 'README.md', 'Screenshot 2023-05-25 145641.png', 'Screenshot 2023-07-24 123620.png', 'connection.png', 'controller.png', 'create3ns.sh', 'db.sqlite3', 'manage.py', 'requirements.txt', 'requirements1.txt', 'statistics.png', and 'user.name'. The right sidebar shows the 'About' section with no description, 'Releases' section with no releases published, 'Packages' section with no packages published, 'Languages' section showing a bar chart for HTML (41.9%), Python (36.5%), CSS (19.9%), Dockerfile (1.1%), and Shell (0.6%), and 'Suggested workflows' section with two workflows: 'Python Package using Anaconda' and 'SLSA Generic generator'.

SDN\_Django\_framework\_for\_implementation\_network\_service\_configuration\_application

lasimo92 / SDN\_Django\_framework\_for\_implementation\_network\_service\_configuration\_application

Code Issues Pull requests Actions Projects Wiki Security 95 Insights Settings

SDN\_Django\_framework\_for\_implementation\_network\_service\_configuration\_application Public Pin Unwatch 1 Fork 0 Star 0

main 2 Branches 0 Tags

Go to file Add file Code

lasimo92 changing requirements.txt celery version 13cd4ea · 2 months ago 59 Commits

File	Description	Time
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.

Readme Activity 0 stars 1 watching 0 forks

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 Configure  
Create and test a Python package on multiple Python versions using Anaconda for package management.

SLSA Generic generator Configure  
Create SLSA generic generator

Σχήμα 10.5: github repo

# Κεφάλαιο 11

## Βιβλιογραφία

### 11.1 Βιβλιογραφικές αναφορές

- 1 <https://kubernetes.io/docs/concepts/overview/>
- 2 Vs Code development environment
- 3 <https://el.wikipedia.org/wiki/>
- 4 Virtual box ή οποιονδήποτε type B hypervisor
- 5 <https://www.cloudflare.com/learning/network-layer/what-is-the-control-plane/>
- 6 <https://el.wikipedia.org/wiki/Git>
- 7 <https://kubernetes.io>
- 8 <https://github.com/dmfigol/network-programmability-stream>
- 9 <https://medium.com/@komalminhas.96/a-step-by-step-guide-to-build-and-push-your-own-docker-images-to-dockerhub-709963d4a8bc>
- 10 <https://repository.ihu.edu.gr/>, Network automation using python George Milios
- 11 Wikipedia, τι είναι το Windows Subsystem for Linux
- 12 [wikipedia.org/wiki/Modelviewcontroller](https://wikipedia.org/wiki/Modelviewcontroller)
- 13 <https://docs.docker.com/engine/daemon/start/>
- 14 <https://minikube.sigs.k8s.io/docs/start/>
- 15 <https://developer.mozilla.org/en-US/docs/Glossary/MVC>
- 16 <https://www.freecodecamp.org/news/python-requirements.txt-explained/>