

Πτυχιακή Εργασία



ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη διαδικτυακής εφαρμογής για την κατασκευή
και τον διαμοιρασμό Linux περιβαλλόντων

Πανακάκης Σταύρος
Π2018108

Επιβλέπουσα: Ελένη Χριστοπούλου

Κέρκυρα, Ιούνιος 2022

Τριμελής επιτροπή

Ελένη Χριστοπούλου, ΕΔΙΠ,
Ιόνιο Πανεπιστήμιο, Τμήμα Πληροφορικής

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο, Τμήμα Πληροφορικής

Βασίλειος Καρυώτης, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο, Τμήμα Πληροφορικής

Περίληψη

Σε ποικίλα εργαστηριακά μαθήματα πληροφορικής (π.χ. προγραμματισμός υπολογιστών, λειτουργικά συστήματα) απαιτείται πέρα από τη θεωρητική γνώση και η πρακτική εξάσκηση εν ώρα μαθήματος. Ένας συνηθισμένος τρόπος είναι το κάλεσμα των φοιτητών από τους καθηγητές, κάποιες μέρες πριν ή ακόμη και την ώρα του μαθήματος, για την δημιουργία του κατάλληλου προγραμματιστικού περιβάλλοντος στον προσωπικό τους υπολογιστή για τις ανάγκες του μαθήματος. Αυτό συνήθως απαιτεί εξειδικευμένες γνώσεις υπολογιστών, πρόσβαση στο ίντερνετ και προσωπικούς υπολογιστές με ικανοποιητικά χαρακτηριστικά. Η λύση σε αυτό το πρόβλημα είναι μία ιντερνετική εφαρμογή η οποία αφορά εκπαιδευτικά ιδρύματα πληροφορικής. Ο σκοπός είναι οι καθηγητές να δημιουργούν εικονικές μηχανές οι οποίες να περιλαμβάνουν το λογισμικό της επιλογής τους για τις ανάγκες ενός μαθήματος και οι φοιτητές να κλωνοποιούν αυτές τις εικονικές μηχανές απλώς πατώντας έναν σύνδεσμο.

Λέξεις - Κλειδιά: Linux, προγραμματισμός, εφαρμογή

Abstract

In computer lab workshops such as operating systems lab, students should use heavyweight or complex software which is extremely difficult to install. As a result, students create an appropriate programming environment on their personal computers on lesson time which is really time-consuming. One solution is the development a web-based application on which professors will be able to create personalized Linux environments in minutes and share them to their students just by using a link.

Keywords: Linux, coding, application

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους τους δικούς μου ανθρώπους.

Περιεχόμενα

Τριμελής επιτροπή	i
Περίληψη	ii
Abstract	ii
Ευχαριστίες	iii
Περιεχόμενα	iv
1 Εισαγωγή	2
1.1 Σκοπός - Κίνητρο	2
1.2 Συμβολή	2
1.3 Διάρθρωση της εργασίας	2
2 Βιβλιογραφική ανασκόπηση	4
2.1 Θεωρητικό υπόβαθρο	4
2.1.1 Τι είναι το Linux;	4
2.1.2 Τι είναι το Docker;	4
2.2 Συνεργατικές πλατφόρμες	5
2.3 Σχετικές πλατφόρμες	5
2.3.1 Play with Docker	5
2.3.2 CS50 Sandbox	6
2.4 Cloud computing	7
3 Αρχιτεκτονική	8
3.1 Προεπισκόπηση	8
3.2 Απαιτήσεις	9
3.3 Πυρήνας	9
3.3.1 Πελάτης	9
3.3.2 Διακομιστής	9
3.4 Αντίστροφος διακομιστής μεσολάβησης	9
3.5 Βάση δεδομένων	12
3.5.1 users	12
3.5.2 environments	12
3.5.3 urlshorteners	13
3.6 Docker engine	13
3.7 Προγραμματιστικό περιβάλλον	14
3.7.1 Πελάτης	14

3.7.2	Διακομιστής	15
3.7.3	Βάση δεδομένων	17
3.7.4	Περιγραφή της διεπαφής της εφαρμογής	17
4	Διεπαφή χρήστη	19
4.1	Προεπισκόπηση	19
4.2	Σελίδες	19
4.2.1	Ιστοσελίδα Παρουσίασης	19
4.2.2	Εγγραφή του χρήστη	20
4.2.3	Σύνδεση του χρήστη	21
4.2.4	Λίστα περιβαλλόντων	22
4.2.5	Δημιουργία περιβάλλοντος	25
4.2.6	Αλλαγή κωδικού	26
4.3	Διαδικασία χρήσης	26
4.4	Επαναχρησιμοποιούμενα Μέρη	27
4.4.1	Form	27
4.4.2	Sidebar	30
4.4.3	Head	33
4.4.4	Loader	34
4.4.5	Modal	35
5	Λειτουργικότητα της εφαρμογής	37
5.1	Προεπισκόπηση	37
5.2	Αρχιτεκτονική	37
5.3	User	38
5.3.1	Entity	38
5.3.2	Repository	39
5.3.3	Services	40
5.3.4	Handlers	41
5.4	Environment	44
5.4.1	Entity	44
5.4.2	Repository	44
5.4.3	Docker	46
5.4.4	Services	49
5.4.5	Handlers	50
5.5	URL Shortener	52
5.5.1	Entity	52
5.5.2	Repositories	53
5.5.3	Services	53
5.5.4	Handlers	54
6	Διαδικασίες	56
6.1	Continuous Integration	56
6.2	Deployment	59
6.2.1	Docker swarm	59
6.2.2	GitHub Actions	62
7	Συμπεράσματα	66
7.1	Μελλοντικές προοπτικές	66

Κατάλογος Σχημάτων

2.1	Λογότυπο του Linux	4
2.2	Στιγμιότυπο οθόνης από την εφαρμογή Play with Docker	6
2.3	Στιγμιότυπο οθόνης από την εφαρμογή CS50 Sandbox	7
3.1	Αρχιτεκτονική της εφαρμογής	8
3.2	Διάγραμμα οντοτήτων του πίνακα users	12
3.3	Διάγραμμα οντοτήτων του πίνακα environments	13
3.4	Διάγραμμα οντοτήτων του πίνακα urlshorteners	14
3.5	Απεικόνιση του λογισμικού Swagger	18
4.1	Σελίδα παρουσίασης	20
4.2	Σελίδα εγγραφής του χρήστη	20
4.3	Σελίδα εγγραφής του χρήστη με μήνυμα λάθους	21
4.4	Σελίδα σύνδεσης του χρήστη	22
4.5	Σελίδα με άδεια λίστα περιβαλλόντων	22
4.6	Σελίδα με λίστα περιβαλλόντων	23
4.7	Σελίδα κονσόλας	23
4.8	Σελίδα κοινοποίησης	24
4.9	Σελίδα διαγραφής	25
4.10	Σελίδα δημιουργίας περιβάλλοντος	25
4.11	Σελίδα αλλαγής κωδικού	26
4.12	Διάγραμμα χρήσης	27
4.13	Μέρος Form	28
4.14	Μέρος Input	29
4.15	Μέρος Error	30
4.16	Μέρος Button	30
4.17	Sidebar Component	31
4.18	Button Component	32
4.19	Μέρος SubButton	33
4.20	Μέρος BottomButton	33
4.21	Μέρος Loader	35
4.22	Μέρος Modal	36
5.1	Clean architecture	38
6.1	Continuous integration pipeline στο pull request	58
6.2	Continuous integration pipeline για το deployment της εφαρμογής	65

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός - Κίνητρο

Τα εργαστηριακά μαθήματα του τμήματος τα οποία περιλαμβάνουν την εγκατάσταση προγραμματιστικών εργαλείων ή ακόμη και την εγκατάσταση των λειτουργικών συστημάτων τύπου Linux[1] είναι κάτι το οποίο δυσκόλευε ιδιαίτερα τους φοιτητές και χάνεται αρκετός χρόνος μέχρι να έρθει η ώρα η οποία οι φοιτητές θα χρησιμοποιήσουν αυτά τα εργαλεία λόγω του ότι καθυστερεί η εγκατάστασή τους. Επιπλέον, κάποια από αυτά τα εργαλεία απαιτούν μεγάλη υπολογιστική δύναμη κάτι το οποίο πολλοί φοιτητές δεν έχουν την οικονομική άνεση να την αποκτήσουν. Αυτό το πρόβλημα ήταν η αφορμή για την δημιουργία της εφαρμογής Nephela Labs η οποία δημιουργήθηκε για να το λύσει.

1.2 Συμβολή

Η δημιουργία της συγκεκριμένης εφαρμογής θα λύσει το πρόβλημα που αναφέρθηκε παραπάνω με έναν πολύ εύκολο τρόπο. Η ιδέα έχει ως εξής. Είναι μια διαδικτυακή εφαρμογή όπου οι καθηγητές θα μπορούν να δημιουργήσουν Linux[1] περιβάλλοντα με τα πακέτα της επιλογής τους απλώς συμπληρώνοντας μια φόρμα και θα μπορούν να τα στείλουν στους φοιτητές τους μόνο με ένα link. Έτσι οι φοιτητές θα μπορούν να τα κλωνοποιήσουν στο δικό τους λογαριασμό και να έχουν ο καθένας το προσωπικό του σύστημα Linux[1] με τα πακέτα που απαιτούνται στο κάθε μάθημα χωρίς να χρειαστεί να εγκαταστήσουν τίποτα εφόσον το τεματικό υπάρχει στον φυλλομετρητή των χρηστών.

1.3 Διάρθρωση της εργασίας

Στα παρακάτω κεφάλαια θα υπάρχει η περιγραφή για όλα τα βήματα από τον σχεδιασμό μέχρι και την τελική υλοποίηση της εφαρμογής. Αρχικά, θα υπάρχει αναφορά στο θεωρητικό υπόβαθρο της εφαρμογής. Στο επόμενο κεφάλαιο θα υπάρχει η επεξήγηση της αρχιτεκτονικής της εφαρμογής όπου θα αναφέρονται αναλυτικά όλα τα μέρη της εφαρμογής σε υψηλό επίπεδο. Για παράδειγμα ο πυρήνας της όπου θα εξηγείται η επικοινωνία πελάτη-διακομιστή, ο αντίστροφος διακομιστής μεσολάβησης, η βάση δεδομένων, το Docker Engine [9] και ο τρόπος

όπου δημιουργήθηκε το προγραμματιστικό περιβάλλον. Έπειτα θα γίνει η επεξήγηση της διεπαφής της εφαρμογής. Σε αυτό το κεφάλαιο θα γίνει περιγραφή όλων των σελιδών της πλατφόρμας, όλων των εναχρησιμοποιούμενων μερών που δημιουργήθηκαν καθώς και η διαδικασία χρήσης της εφαρμογής. Μετά, ακολουθεί η λειτουργικότητα της εφαρμογής όπου γίνεται επεξήγηση του τρόπου με τον οποίο λειτουργεί η πλατφόρμα σε χαμηλό επίπεδο. Για παράδειγμα, περιγραφή μεθόδων του κώδικα καθώς και της διεπαφής προγραμματισμού. Επιπλέον, στο επόμενο κεφάλαιο θα αναλυθούν διαδικασίες οι οποίες υποστηρίζουν την ανάπτυξη της εφαρμογής και το πως ο κώδικας καταλήγει από την τοπική εγκατάσταση στην παραγωγή χωρίς να χρειαστεί να σταματήσει ο διακομιστής. Τέλος, υπάρχουν τα συμπεράσματα καθώς και οι μελλοντικές προοπτικές.

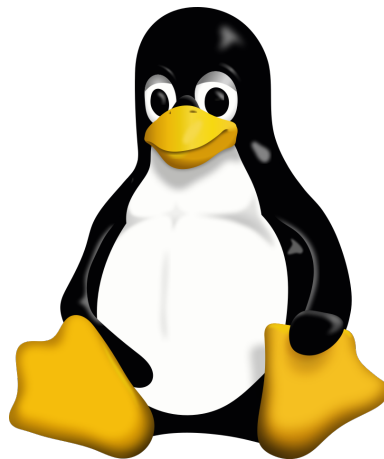
Κεφάλαιο 2

Βιβλιογραφική ανασκόπηση

2.1 Θεωρητικό υπόβαθρο

2.1.1 Τι είναι το Linux;

Το Linux[1] [Σχήμα 2.1] είναι μια "βάση" ενός λειτουργικού συστήματος ανοιχτού κώδικα όπου οι χρήσεις του ποικίλουν. Πιο συγκεκριμένα, χρησιμοποιείται από διακομιστές μέχρι και θερμοστάτες. Δημιουργήθηκε τον Linus Torvalds το 1991 και είναι βασισμένο στο λειτουργικό σύστημα Unix[2]. Όπως αναφέρθηκε προηγουμένως, επειδή είναι η "βάση" για ένα λειτουργικό σύστημα, πολλές διανομές έχουν δημιουργηθεί βασισμένα σε αυτό. Κάποια παραδείγματα είναι το Debian[3], Ubuntu[4], Linux Mint[5], Manjaro[6], Arch[7]. Στο Nephela Labs χρησιμοποιήθηκε η διανομή Ubuntu[4] τόσο στην φιλοξενία της εφαρμογής όσο και στην δημιουργία περιβαλλόντων.



Σχήμα 2.1: Λογότυπο του Linux

2.1.2 Τι είναι το Docker;

Το Docker[9] είναι μια open source πλατφόρμα η οποία χρησιμοποιείται για πακετάρισμα λογισμικού με σκοπό της ανεξαρτητοποίησης του από την υποδομή ενός συστήματος. Αυτό είναι χρήσιμο διότι μπορούν να δημιουργηθούν πολλαπλά

πλήρως ολοκληρωμένα λειτουργικά συστήματα σε μια υποδομή. Ο τρόπος που λειτουργεί είναι ότι εκμεταλλεύεται την τεχνολογία των containers του πυρήνα του Linux[1]. Ο λόγος που το Nephelē Labs χρησιμοποίησε το Docker[9] είναι τόσο για την δημιουργία περιβαλλόντων των χρηστών, όσο και για την ανάπτυξή του στον διακομιστή.

2.2 Συνεργατικές πλατφόρμες

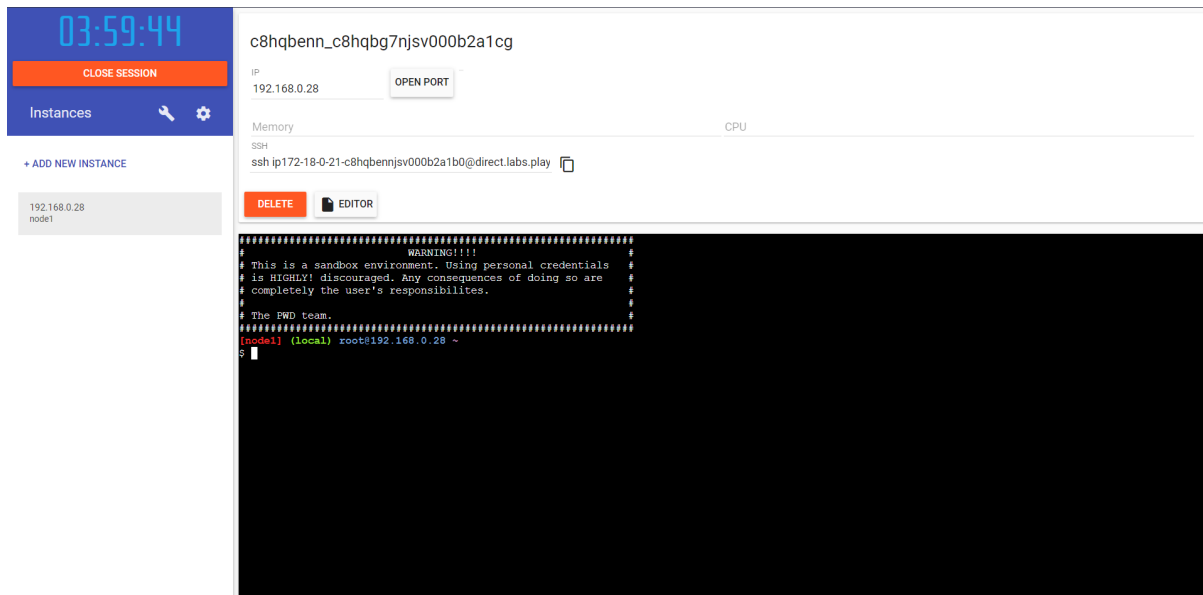
Στον τεχνολογικό χώρο υπάρχει η ανάγκη συνεργασίας πολλών ατόμων για την ολοκλήρωση ενός έργου ή ενός προγράμματος. Τα προβλήματα τα οποία υπάρχουν πολλές φορές δημιουργούνται από γεωγραφικές ή και πληθυσμιακές δυσκολίες. Άτομα τα οποία βρίσκονται σε διαφορετικά μέρη του κόσμου ακόμα και εάν καταφέρουν να ανταλλάξουν πληροφορία, δεν είναι σίγουρο πως θα μπορούν να το κάνουν με αμεσότητα, σε ολόκληρη την ομάδα με την οποία συνεργάζονται. Για τον λόγο αυτό, είναι σημαντική η δημιουργία και η ανάπτυξη των συνεργατικών πλατφορμών (collaborative working platforms). [35] [36]

Μια συνεργατική πλατφόρμα μπορεί να βοηθά τις ομάδες να επιτύχουν συγκεκριμένους στόχους ή να λύσουν επιχειρηματικά προβλήματα μέσω της διαχείρισης εγγράφων, της κοινής χρήσης ιδεών και της διαχείρισης εργασιών, κάνοντας την πιο αποδοτική και πιο παραγωγική. Πολλές συνεργατικές πλατφόρμες χρησιμοποιούνται από Πανεπιστημιακά ιδρύματα, εταιρίες με υπαλλήλους που χρησιμοποιούν το μοντέλο της εξ αποστάσεως εργασίας ή ακόμα και συνέδρια με διαδραστικό χαρακτήρα.

2.3 Σχετικές πλατφόρμες

2.3.1 Play with Docker

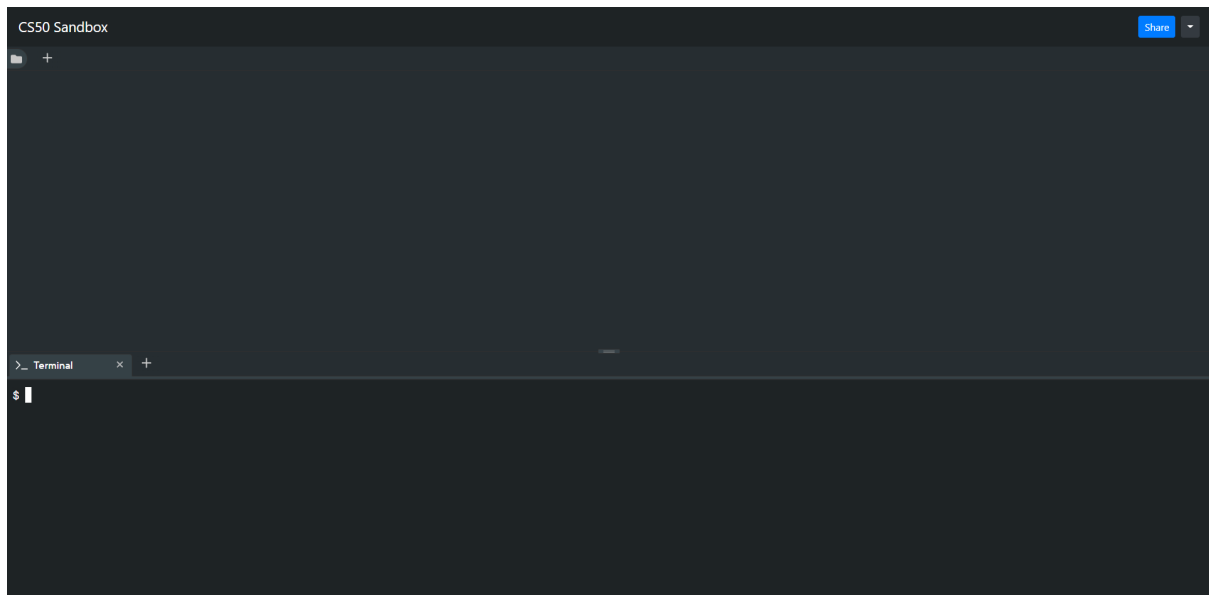
Το Play with Docker[8] [Σχήμα 2.2] είναι μια διαδικτυακή πλατφόρμα ανοιχτού κώδικα η οποία περιέχει ένα τερματικό και επιτρέπει στους χρήστες της να χρησιμοποιήσουν την τεχνολογία Docker[9] στον φυλλομετρητή τους χωρίς να χρειάζεται να κατεβάσουν ξεχωριστό λογισμικό στον υπολογιστή τους. Ο τρόπος που αυτή η πλατφόρμα επέτρεπε στον χρήστη να έχει ένα απόλυτα λειτουργικό τερματικό απλώς πατώντας ένα κουμπί αποτέλεσε έμπνευση για την δημιουργία της πλατφόρμας Nephelē Labs. Οι πλατφόρμες μεταξύ τους έχουν πολλές ομοιότητες και τόσο ως τον τρόπο λειτουργίας τους όσο και στην εμπειρία που προσφέρουν στον χρήστη. Και στις δύο περιπτώσεις υπάρχει ένα περιβάλλον Linux[1] μέσα σε έναν Browser. Οι διαφορές τους είναι ότι το Play with Docker[8] εξιδικεύεται στην τεχνολογία Docker[9] ενώ το Nephelē Labs δίνει στον χρήστη ένα πιο γενικό περιβάλλον και την δυνατότητα να το μοιραστεί.



Σχήμα 2.2: Στιγμιότυπο οθόνης από την εφαρμογή Play with Docker

2.3.2 CS50 Sandbox

Η πλατφόρμα CS50 Sandbox[10] [Σχήμα 2.3] είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης το οποίο υπάρχει στον φυλλομετρητή. Δημιουργήθηκε από την ομάδα CS50[11] στο πανεπιστήμιο Harvard. Η συγκεκριμένη πλατφόρμα αποτέλεσε επίσης έμπνευση για το Nephele Labs διότι είναι όμοιες σαν πλατφόρμες. Το CS50 Sandbox παρέχει ένα ολοκληρωμένο περιβάλλον προγραμματισμού ενώ το Nephele Labs ένα Linux[1] περιβάλλον. Το πρώτο δημιουργήθηκε με σκοπό να επιτρέψει στους χρήστες να εξασκήσουν τις προγραμματιστικές γνώσεις τους έχοντας μόνο έναν φυλλομετρητή ενώ το δεύτερο για να εξασκήσουν τις γνώσεις τους στο Linux[1]. Τέλος, σε αντίθεση με το Play with Docker[8], το CS50 Sandbox[10] επιτρέπει στους χρήστες του να μοιραστούν τα προγραμματιστικά τους περιβάλλοντα, όπως ακριβώς το Nephele Labs.



Σχήμα 2.3: Στιγμιότυπο οθόνης από την εφαρμογή CS50 Sandbox

2.4 Cloud computing

Το μεγάλο μέγεθος της πληροφορίας που υπάρχει στο διαδίκτυο και η ανάγκη ψηφιοποίησης της ανέδειξε μία νέα ανάγκη στο χώρο της πληροφορικής. Ο μεγάλος αυτός όγκος γνώσης και πληροφορίας θα πρέπει να υπάρχει στο διαδίκτυο για εύκολο διαμοιρασμό και εύκολη πρόσβαση ανά πάσα στιγμή με την δυνατότητα πρόσβασης από μεγάλο πλήθος χρηστών. Για τον λόγο αυτό δημιουργήθηκε το cloud computing [37] [38]. Στην σημερινή εποχή, το cloud computing έχει πολλές χρήσεις όπως, αποθήκευση αρχείων κειμένου, ήχου ή εικόνων, χρήση σε εκπαιδευτικά περιβάλλοντα για την αποστολή και πρόσβαση σε εργασίες μαθητών ή φοιτητών ακόμα και την φιλοξενία εφαρμογών και ιστοτόπων.

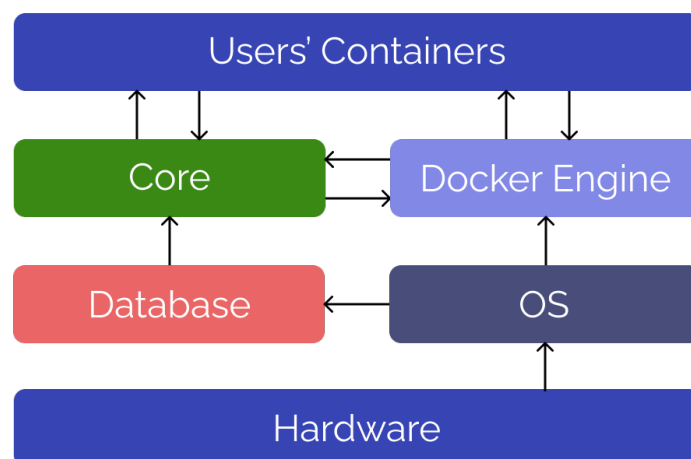
Κεφάλαιο 3

Αρχιτεκτονική

3.1 Προεπισκόπηση

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε ποικιλία τεχνολογιών και εργαλείων με σκοπό τη βέλτιστη λειτουργία και επέκτασή της. Αρχικά, τα μέρη της εφαρμογής [Σχήμα 3.1] είναι τρία:

- Πυρήνας: αποτελεί την διεπαφή του χρήστη και τη λειτουργικότητα της εφαρμογής
- Βάση δεδομένων: χρησιμοποιείται για την αποθήκευση των δεδομένων των χρηστών
- Docker Engine [9] : χρησιμοποιείται για τη δημιουργία των περιβαλλόντων Linux[1].



Σχήμα 3.1: Αρχιτεκτονική της εφαρμογής

3.2 Απαιτήσεις

Σε ένα τέτοιο σύστημα οι απαιτήσεις του χρήστη είναι αρκετές. Ο χρήστης έχει την ανάγκη να δημιουργεί Linux[1] περιβάλλοντα σε μικρό χρόνο, να μπορεί να τα διαμοιράζεται γρήγορα αλλά και να τρέχει όποια λειτουργία επιθυμεί σε αυτά. Ο λόγος είναι για να μοιάζει όσο περισσότερο γίνεται σε τοπικό περιβάλλον Linux[1]. Αυτό απαιτεί συγκεκριμένα χαρακτηριστικά αλλά θέτει και σημαντικούς περιορισμούς οι οποίοι δεν μας επιτρέπουν να καθορίσουμε με ακρίβεια τις απαιτήσεις της πλατφόρμας. Πιο συγκεκριμένα, οι απαιτήσεις ενός τέτοιου συστήματος αυξάνονται αναλογικά με την αύξηση των χρηστών του. Αυτό συμβαίνει διότι ο κάθε χρήστης έχει την ανάγκη να έχει συγκεκριμένα χαρακτηριστικά στο περιβάλλον του. Για παράδειγμα, ας υποθέσουμε ότι η εφαρμογή θα χρησιμοποιηθεί για ένα σεμινάριο 20 ατόμων όπου γίνεται εκμάθηση Linux[1] και ο κάθε χρήστης δεν χρειάζεται να τρέχει χρονοβόρες διαδικασίες. Εάν ο κάθε χρήστης χρειάζεται 128MB μνήμη RAM τότε το σύστημα θα χρειαστεί τουλάχιστον 20 x 128MB μνήμη RAM. Αυτό θα ρυθμίζεται αναλόγως την χρήση που θα κάνει η κάθε ομάδα χρηστών στην εφαρμογή.

3.3 Πυρήνας

Ο πυρήνας της εφαρμογής έχει σχεδιαστεί με την αρχιτεκτονική πελάτη-διακομιστή.

3.3.1 Πελάτης

Ο πελάτης είναι η βασική διεπαφή της εφαρμογής η οποία είναι προσβάσιμη από τον φυλλομετρητή του εκάστοτε χρήστη της. Ο λόγος που επιλέχθηκε μια διεπαφή μέσω φυλλομετρητή και όχι κάποια διεπαφή μέσω επιφάνειας εργασίας ήταν για την ευκολία πρόσβασης του χρήστη διότι δεν είναι απαραίτητη η εγκατάσταση της εφαρμογής. Μελλοντικά, όπως θα αναφερθεί παρακάτω θα μπορούσε να γίνει μια προέκταση της εφαρμογής σε κινητά ή tablet μέσω ξεχωριστής εφαρμογής. Η χρήση του πελάτη είναι η επικοινωνία με τον διακομιστή και η εμφάνιση μηνυμάτων και πληροφοριών στον χρήστη.

3.3.2 Διακομιστής

Ο διακομιστής είναι η βασική λειτουργικότητα της εφαρμογής. Ο πελάτης της εφαρμογής επικοινωνεί με τον διακομιστή ο οποίος επεξεργάζεται την πληροφορία, την προωθεί στο Docker Engine[9] και στην βάση δεδομένων και αναλόγως την εκάστοτε λειτουργία επιστρέφει τα αποτελέσματα στον πελάτη της εφαρμογής.

3.4 Αντίστροφος διακομιστής μεσολάβησης

Ο αντίστροφος διακομιστής μεσολάβησης (nginx [12]) χρησιμοποιείται με ποικίλους τρόπους. Αρχικά, είναι υπεύθυνος για την ανάθεση των ονομάτων ιστοσελίδας nephelabs.com και www.nephelabs.com στον πελάτη της εφαρμογής και του ονόματος api.nephelabs.com στον διακομιστή. Επίσης, χρησιμοποιείται για την

ανάθεση πιστοποιητικού SSL μέσω του λογισμικού Let's encrypt[13] και στον πελάτη και στον διακομιστή με σκοπό την ασφάλή μεταφορά της πληροφορίας. Επιπλέον, προσθέτει άλλο ένα βήμα ασφάλειας διότι ανακατευθύνει τις συνδέσεις HTTP σε συνδέσεις HTTPS. Τέλος, χρησιμοποιείται ως ενδιάμεσο βήμα μεταξύ του πελάτη και του διακομιστή της εφαρμογής. Πιο συγκεκριμένα, λαμβάνει τα μηνύματα του πελάτη και τα προωθεί στα αντίγραφα του διακομιστή. Παρακάτω, ακολουθούν οι ρυθμίσεις. Κάποια σημεία έχουν παραληφθεί για λόγους ασφαλείας.

```
events {
    worker_connections 1024;
}

http {
    server {
        listen 80 default_server;
        return 444;
    }

    server {
        listen 80;
        server_name nephelelabs.com www.nephelelabs.com;
        return 301 https://$server_name$request_uri;
    }

    server {
        listen 443 ssl;
        server_name nephelelabs.com www.nephelelabs.com;

        ssl_certificate /etc/letsencrypt/live/nephelelabs.com/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/nephelelabs.com/privkey.pem;

        ssl_session_cache shared:le_nginx_SSL:10m;
        ssl_session_timeout 1440m;
        ssl_session_tickets off;

        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_prefer_server_ciphers off;

        ssl_ciphers "...";

        location / {
            proxy_pass http://frontend:3000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
```

```

    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Port $server_port;
}

location ~~ /.well-known/acme-challenge/ {
    root /var/www/html;
}

server {
    listen 80;
    server_name api.nephelelabs.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl;
    server_name api.nephelelabs.com;

    ssl_certificate /etc/letsencrypt/live/api.nephelelabs.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/api.nephelelabs.com/privkey.pem;

    ssl_session_cache shared:le_nginx_SSL:10m;
    ssl_session_timeout 1440m;
    ssl_session_tickets off;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers off;

    ssl_ciphers "...

    location / {
        proxy_pass http://backend:8000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Port $server_port;
    }

    location ~~ /.well-known/acme-challenge/ {
        root /var/www/html;
    }
}

```

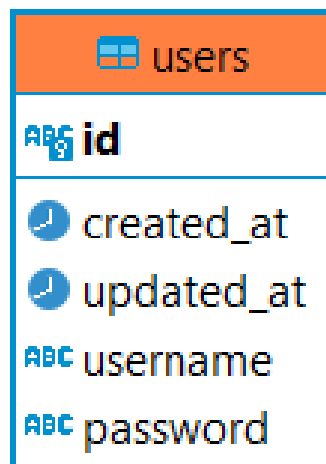
3.5 Βάση δεδομένων

Το RDBMS που χρησιμοποιήθηκε ήταν η MySQL[14]. Ο λόγος που χρησιμοποιήθηκε ήταν λόγω του ότι είναι σχεσιακή βάση δεδομένων, έχει μεγάλη κοινότητα υποστήριξης και την υποστήριξη της εταιρίας Oracle[15]. Οι πίνακες της βάσης δεδομένων που δημιουργήθηκαν ήταν οι ακόλουθοι:

1. users [Σχήμα 3.2]
2. environments [Σχήμα 3.3]
3. urlshorteners [Σχήμα 3.4]

3.5.1 users

```
CREATE TABLE `users` (  
  `id` varchar(191) NOT NULL,  
  `created_at` datetime(3) DEFAULT NULL,  
  `updated_at` datetime(3) DEFAULT NULL,  
  `username` longtext,  
  `password` longtext,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```



Σχήμα 3.2: Διάγραμμα οντοτήτων του πίνακα users

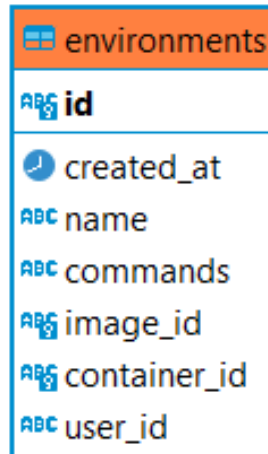
3.5.2 environments

```
CREATE TABLE `environments` (  
  `id` varchar(191) NOT NULL,  
  `created_at` datetime(3) DEFAULT NULL,  
  `name` longtext NOT NULL,
```

```

`commands` longtext NOT NULL,
`image_id` varchar(191) DEFAULT NULL,
`container_id` varchar(191) DEFAULT NULL,
`user_id` varchar(200) DEFAULT NULL,
PRIMARY KEY (`id`),
UNIQUE KEY `image_id` (`image_id`),
UNIQUE KEY `container_id` (`container_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

```



Σχήμα 3.3: Διάγραμμα οντοτήτων του πίνακα environments

3.5.3 urlshorteners

```

CREATE TABLE `url_shorteners` (
  `id` varchar(191) NOT NULL,
  `created_at` datetime(3) DEFAULT NULL,
  `name` longtext NOT NULL,
  `commands` longtext NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

```

3.6 Docker engine

Το Docker engine[9] χρησιμοποιήθηκε για πολλούς και διαφορετικούς λόγους. Αρχικά, χρησιμοποιήθηκε για την δημιουργία τοπικού προγραμματιστικού περιβάλλοντος το οποίο θα αναλυθεί περισσότερο παρακάτω. Επιπλέον, χρησιμοποιήθηκε με σκοπό την επικοινωνία του με τον διακομιστή και την δημιουργία των environments. Πιο συγκεκριμένα, ένα environment είναι ένα Docker Container [16] το οποίο βασίζεται σε ένα Docker Image [17]. Τέλος, χρησιμοποιήθηκε για την της εφαρμογής στην παραγωγή το οποίο θα αναλυθεί επίσης σε ένα διαφορετικό κεφάλαιο.



Σχήμα 3.4: Διάγραμμα οντοτήτων του πίνακα urlshorteners

3.7 Προγραμματιστικό περιβάλλον

Για την ανάπτυξη της εφαρμογής υπήρχε η ανάγκη για την δημιουργία ενός προγραμματιστικού περιβάλλοντος το οποίο θα είναι όσο περισσότερο όμοιο γίνεται με το πραγματικό περιβάλλον στον διακομιστή της πραγματικής εφαρμογής. Με αφορμή τη συγκεκριμένη ανάγκη χρησιμοποιήθηκε το Docker[9]. Δημιουργήθηκε ένα αρχείο τύπου Docker Compose[18] το οποίο ενορχηστρώνει όλες τις υπηρεσίες που απαιτούνται για την εκτέλεση της εφαρμογής.

3.7.1 Πελάτης

Η υπηρεσία του πελάτη της εφαρμογής ονομάστηκε frontend και χρησιμοποιήθηκε ένα προσαρμοσμένο Docker Image[17] για τη δημιουργία του. Επιπλέον, χρησιμοποιήθηκε μια μεταβλητή περιβάλλοντος η οποία αναφέρει στο πρόγραμμα αν η εφαρμογή βρίσκεται σε τοπικό περιβάλλον ή σε περιβάλλον παραγωγής. Επίσης, η εφαρμογή ακούει στην πόρτα 3000.

```
// docker-compose.yml
frontend:
  build:
    context: ../web
    dockerfile: ../build/images/frontend.dev.Dockerfile
  environment:
    NEXT_PUBLIC_DEBUG: ${DEBUG:-true}
  ports:
    - "3000:3000"
  volumes:
    - ../web:/usr/src/app
    - /usr/src/app/node_modules
```

Το προσαρμοσμένο Docker Image[17] που δημιουργήθηκε χρησιμοποιεί ως βάση την fermium[19] η οποία είναι μια έκδοση της Node[20] η οποία υποστηρίζεται για ένα μεγάλο χρονικό διάστημα, κάνει εγκατάσταση τα απαραίτητα πακέτα και τρέχει την εφαρμογή τοπικά.

```
// frontend.dev.Dockerfile
FROM node:fermium-alpine3.11

WORKDIR /usr/src/app

COPY . .

RUN yarn

CMD ["yarn", "dev"]
```

3.7.2 Διακομιστής

Η υπηρεσία του διακομιστή της εφαρμογής ονομάστηκε backend και χρησιμοποιήθηκε επίσης ένα προσαρμοσμένο Docker Image[17] για τη δημιουργία της. Επιπλέον, ο διακομιστής ακούει στην πόρτα 8000. Επίσης, χρησιμοποιήθηκαν κάποιες μεταβλητές περιβάλλοντος οι οποίες είναι τα στοιχεία αυθεντικοποίησης της βάσης δεδομένων, η έκδοση του βασικού Docker Image[17] των environments και το μυστικό κλειδί το οποίο χρησιμοποιείται για την παραγωγή συμβόλων αυθεντικοποίησης τα οποία θα αναλυθούν σε ένα παρακάτω κεφάλαιο.

```
// docker-compose.yml
backend:
build:
  context: ../
  dockerfile: ./build/images/backend.dev.Dockerfile
ports:
  - "8000:8000"
environment:
  BASE_IMAGE_VERSION: ${BASE_IMAGE_VERSION:-1.0.0}
  DEBUG: ${DEBUG:-true}
  DATABASE_NAME: ${DATABASE_NAME:-localdb}
  DATABASE_USER: ${DATABASE_USER:-root}
  DATABASE_PASSWORD: ${DATABASE_PASSWORD:-root}
  DATABASE_HOST: ${DATABASE_HOST:-db}
  DATABASE_PORT: ${DATABASE_PORT:-3306}
  DSN: root:root@tcp(db:3306)/localdb
  SECRET_KEY: ${SECRET_KEY:-secret}
volumes:
  - ../:/app/nephelelabs
  - /var/run/docker.sock:/var/run/docker.sock
depends_on:
  - db
```

Το προσαρμοσμένο Docker Image[17] που δημιουργήθηκε χρησιμοποιεί ως βάση το λειτουργικό σύστημα Alpine Linux[21] με εγκατεστημένη την έκδοση 1.16 της Go[22]. Επιπλέον, έγινε εγκατάσταση συγκεκριμένων πακέτων με σκοπό να καταστεί δυνατή η εκτέλεση των tests μέσα στο συγκεκριμένο Docker Container[16]. Τέλος, έγινε η εγκατάσταση του λογισμικού Air[23] με σκοπό την γρήγορη επαναφόρτωση της εφαρμογής σε τοπικό επίπεδο.

```
// backend.dev.Dockerfile
FROM golang:1.16-alpine3.14

WORKDIR /app/nephelelabs

# For testing purposes inside container
RUN apk add gcc musl-dev

COPY go.mod .
COPY go.sum .

ENV DEBUG=true

RUN go mod tidy
RUN go mod vendor

COPY air.toml .
COPY domain ./domain
COPY environments ./environments
COPY pkg ./pkg
```

```
COPY urlshortener ./urlshortener
COPY users ./users
COPY main.go .

# Install air for hot reload purposes
RUN go get github.com/cosmtrek/air

# Run with hot reload
CMD ["air", "-c", "air.toml"]
```

3.7.3 Βάση δεδομένων

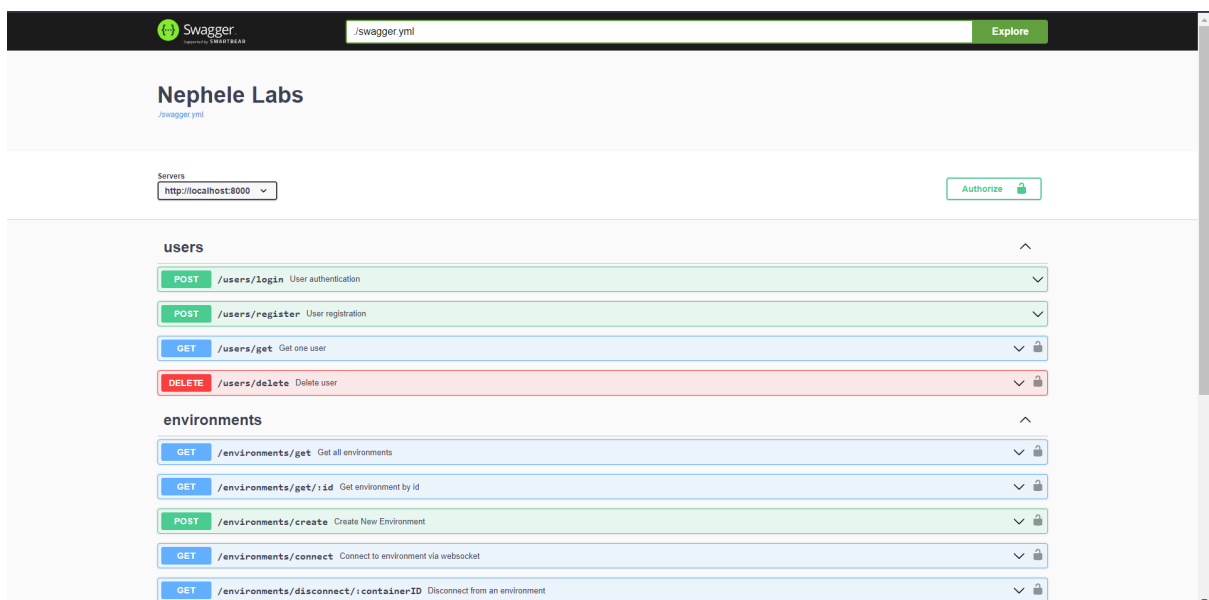
Η υπηρεσία της βάσης δεδομένων χρησιμοποιεί το Docker Image[17] με όνομα MySQL[14] και την έκδοση 8.0. Επιπλέον, ακούει στην πόρτα 3306 και χρησιμοποιεί κάποιες μεταβλητές περιβάλλοντος για την δημιουργία κωδικού χρήστη και ονόματος βάσης.

```
// docker-compose.yml
db:
  image: mysql:8.0
  command: --default-authentication-plugin=mysql_native_password
  volumes:
    - ../mysql_data:/var/lib/mysql
  environment:
    MYSQL_ROOT_PASSWORD: ${DATABASE_PASSWORD:-root}
    MYSQL_DATABASE: ${DATABASE_NAME:-localdb}
  ports:
    - 3306:3306
```

3.7.4 Περιγραφή της διεπαφής της εφαρμογής

Η τελευταία υπηρεσία που δημιουργήθηκε αφορά την περιγραφή της διεπαφής της εφαρμογής. Χρησιμοποιήθηκε το Docker Image[17] του λογισμικού Swagger[24] [Σχήμα 3.5] και ακούει στην πόρτα 4000. Οι ρυθμίσεις του λογισμικού ακολουθούν παρακάτω.

```
// docker-compose.yml
swagger:
  image: swaggerapi/swagger-ui
  volumes:
    - ../swagger:/swagger
  ports:
    - 4000:8080
  environment:
    SWAGGER_JSON: /swagger/swagger.yml
```



Σχήμα 3.5: Απεικόνιση του λογισμικού Swagger

Κεφάλαιο 4

Διεπαφή χρήστη

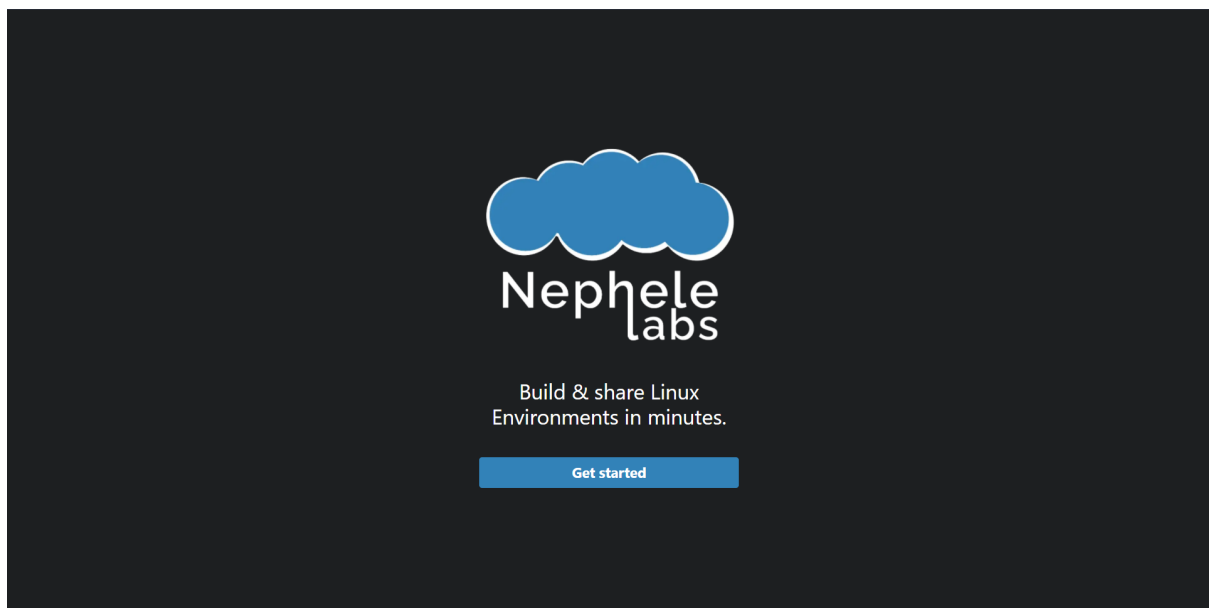
4.1 Προεπισκόπηση

Η γλώσσα στην οποία γράφτηκε η διεπαφή της εφαρμογής είναι η TypeScript[25]. Πιο συγκεκριμένα, χρησιμοποιήθηκε η βιβλιοθήκη Reactjs[26] μέσω της εργαλειοθήκης Nextjs[27]. Πριν την τελική υλοποίηση, έγινε δοκιμή και με την εργαλειοθήκη Sapper[28] της Svelte[29] και με HTML υποδείγματα. Ο λόγος που τελικά επιλέχθηκαν οι συγκεκριμένες τεχνολογίες ποικίλουν. Αρχικά, η TypeScript[25] επιλέχθηκε διότι πρακτικά είναι μια μορφή της JavaScript με τύπους μεταβλητών όπου η JavaScript είναι η γλώσσα προγραμματισμού που υποστηρίζουν οι φυλλομετρητές. Έπειτα, ο λόγος που επιλέχθηκε η Reactjs[26] σαν βιβλιοθήκη ήταν η μεγάλη κοινότητα προγραμματιστών οι οποίοι ασχολούνται με την συγκεκριμένη βιβλιοθήκη. Αυτό σημαίνει πως υπήρχαν ταυτόχρονα εύκολες και αποδοτικές λύσεις στα περισσότερα προβλήματα της εφαρμογής. Εκτός, από τη λογική της διεπαφής της εφαρμογής, έπρεπε να γίνει και η επιλογή για την εμφάνιση της. Αντί για τον κλασσικό τρόπο της CSS επιλέχθηκε η εργαλειοθήκη Tailwind CSS[30] για την ευκολία και την ταχύτητα συγγραφής του.

4.2 Σελίδες

4.2.1 Ιστοσελίδα Παρουσίασης

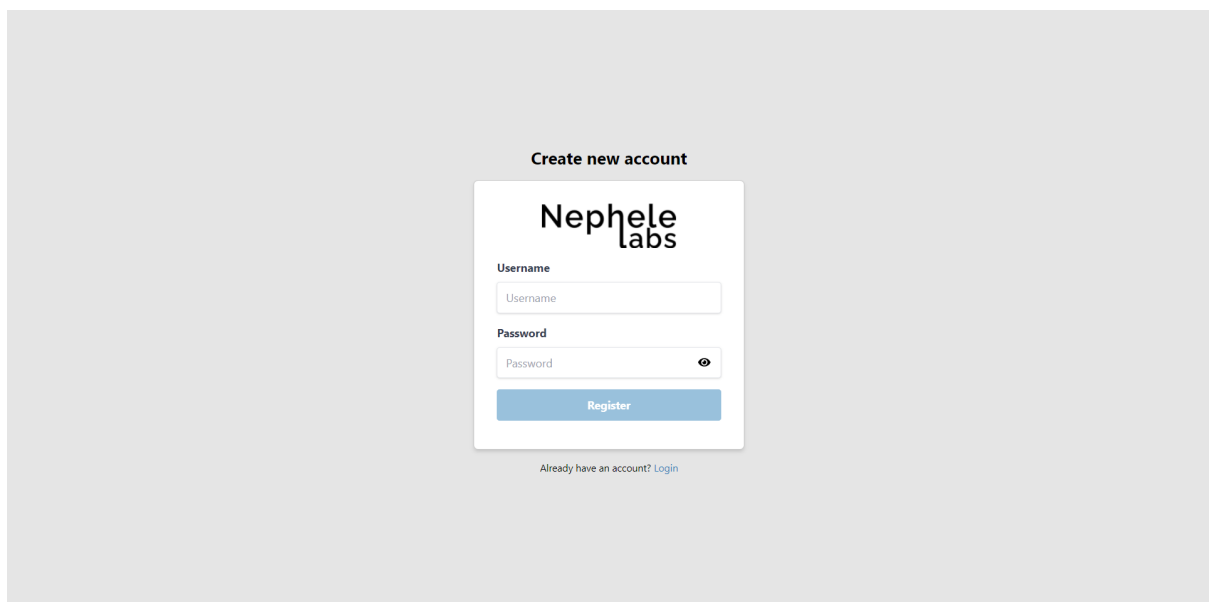
Η πρώτη σελίδα η οποία βλέπει ο χρήστης όταν ανοίξει την εφαρμογή είναι η σελίδα παρουσίασής της [Σχήμα 4.1]. Αρχικά, είναι πολύ μινιμαλιστική εφόσον περιλαμβάνει απλά το λογότυπο και το σύνθημα της εφαρμογής. Ο χρήστης έχει μόνο μια επιλογή σε αυτή τη σελίδα η οποία είναι το κουμπί Get Started όπου μεταφέρει τον χρήστη στην σελίδα εγγραφής.



Σχήμα 4.1: Σελίδα παρουσίασης

4.2.2 Εγγραφή του χρήστη

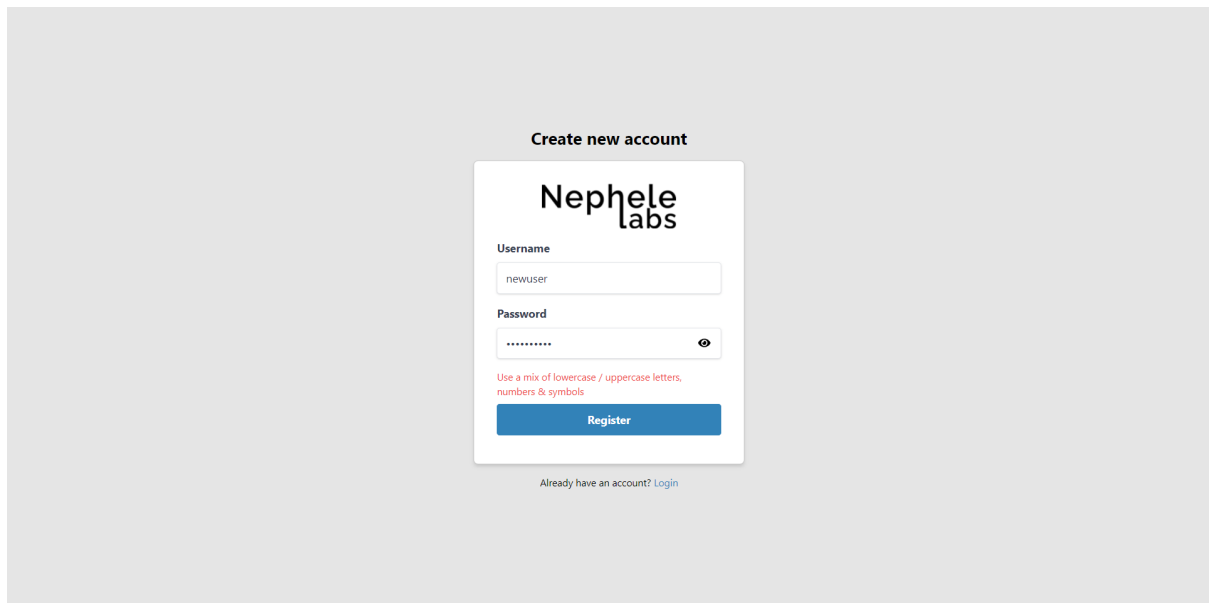
Η συγκεκριμένη σελίδα αφορά την εγγραφή του χρήστη στην εφαρμογή. [Σχήμα 4.3] Πιο συγκεκριμένα, ο χρήστης συμπληρώνει το όνομα χρήστη και τον κωδικό της επιλογής του και σε περίπτωση που δεν υπάρχει κάποιο λάθος μεταφέρεται στην επόμενη σελίδα όπου είναι η σελίδα σύνδεσης [Σχήμα 4.4].



Σχήμα 4.2: Σελίδα εγγραφής του χρήστη

Σε περίπτωση που υπάρχει κάποιο λάθος (π.χ. ότι ο κωδικός δεν είναι έγκυρος ή ότι το όνομα χρήστη υπάρχει) επιστρέφεται το μήνυμα λάθους [Σχήμα 4.3]. Σε

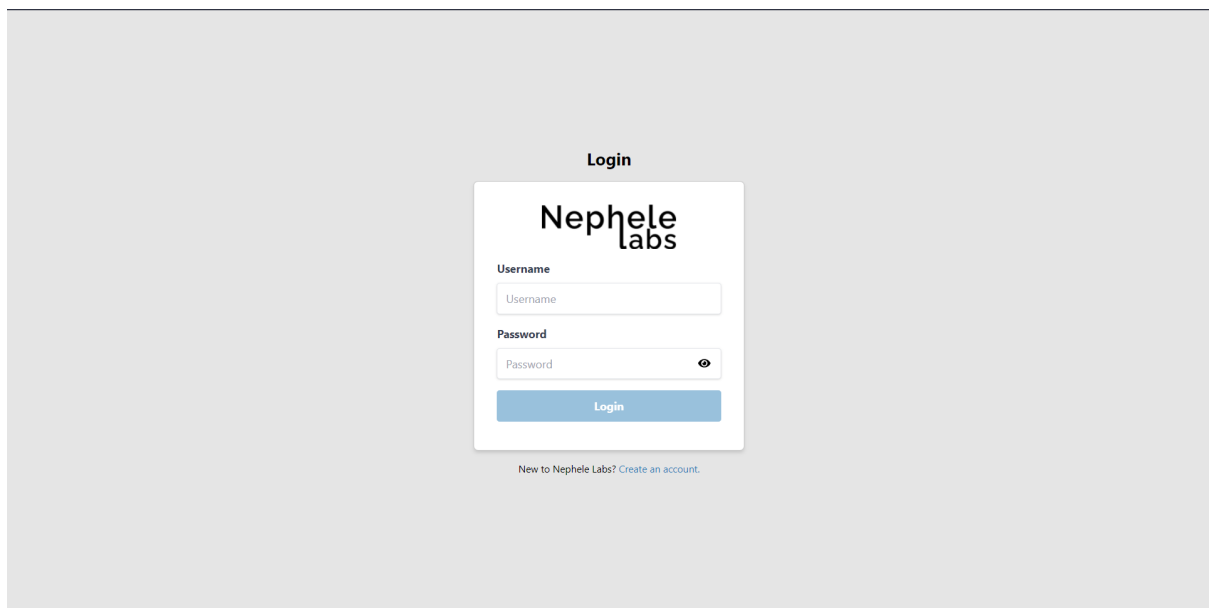
περίπτωση που ο χρήστης έχει ήδη λογαριασμό, μπορεί να πατήσει τον σύνδεσμο κάτω από την φόρμα με σκοπό να μεταφερθεί απευθείας στην σελίδα σύνδεσης [Σχήμα 4.4].



Σχήμα 4.3: Σελίδα εγγραφής του χρήστη με μήνυμα λάθους

4.2.3 Σύνδεση του χρήστη

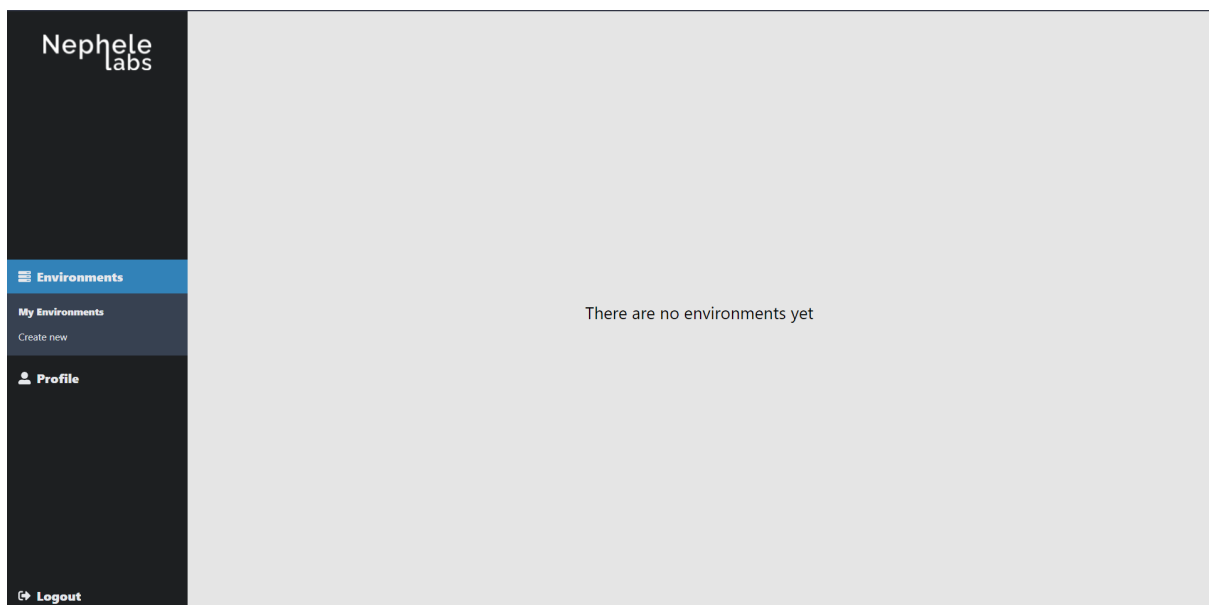
Η συγκεκριμένη σελίδα αφορά την σύνδεση του χρήστη [Σχήμα 4.4]. Αρχικά, μοιάζει πάρα πολύ με την εγγραφή του χρήστη αλλά έχει κάποιες σημαντικές διαφορές. Ο χρήστης χρησιμοποιεί αυτή τη σελίδα για την αυθεντικοποίηση του και όχι για την εγγραφή του στην υπηρεσία. Τα στοιχεία που πρέπει να συμπληρώσει είναι το όνομα χρήστη και ο κωδικός του. Σε περίπτωση που αυτά τα στοιχεία είναι σωστά, γίνεται ανακατεύθυνση στην σελίδα με τα περιβάλλοντα του. Σε περίπτωση που υπάρχει κάποιο πρόβλημα (π.χ. όνομα χρήστη το οποίο δεν υπάρχει ή λάθος κωδικός) τότε εμφανίζεται μήνυμα λάθους.



Σχήμα 4.4: Σελίδα σύνδεσης του χρήστη

4.2.4 Λίστα περιβαλλόντων

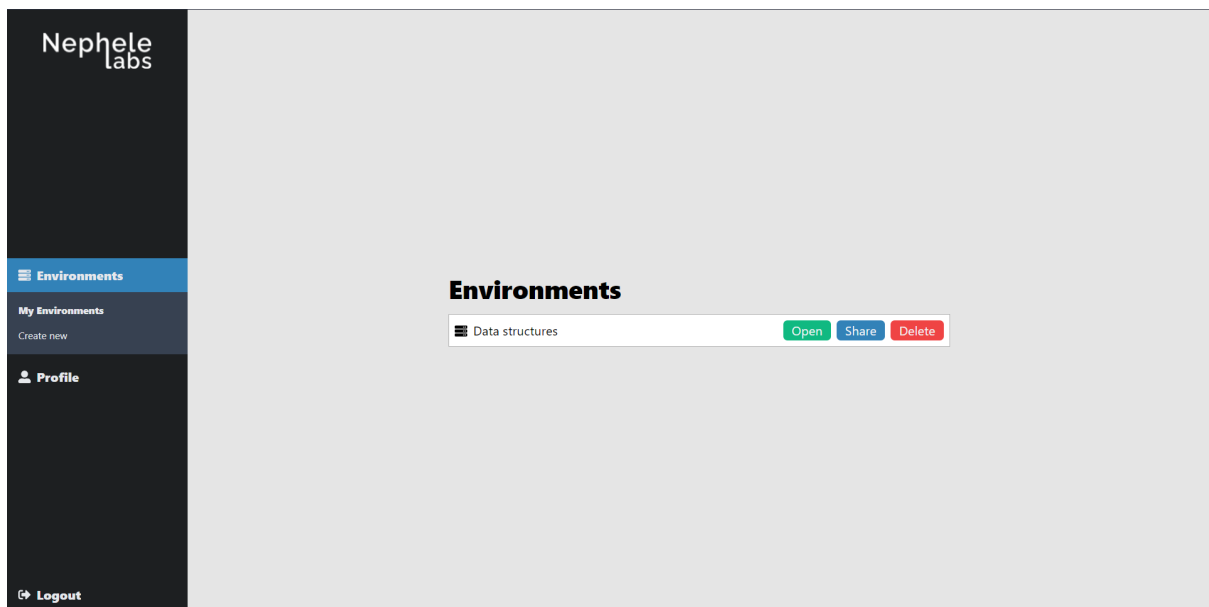
Αυτή η σελίδα, είναι η αρχική σελίδα του ταμπλό του χρήστη. Είναι μια σελίδα που δείχνει μια λίστα με τα περιβάλλοντα του χρήστη. Σε περίπτωση που ο χρήστης δεν έχει δημιουργήσει κανένα περιβάλλον, φαίνεται μήνυμα που τον ενημερώνει [Σχήμα 4.5].



Σχήμα 4.5: Σελίδα με άδεια λίστα περιβαλλόντων

Σε περίπτωση που υπάρχει τουλάχιστον ένα δείχνει το όνομα του περιβάλλοντος και τις επιλογές που έχει για αυτό [Σχήμα 4.6]. Παρακάτω, ακολουθεί η περιγραφή

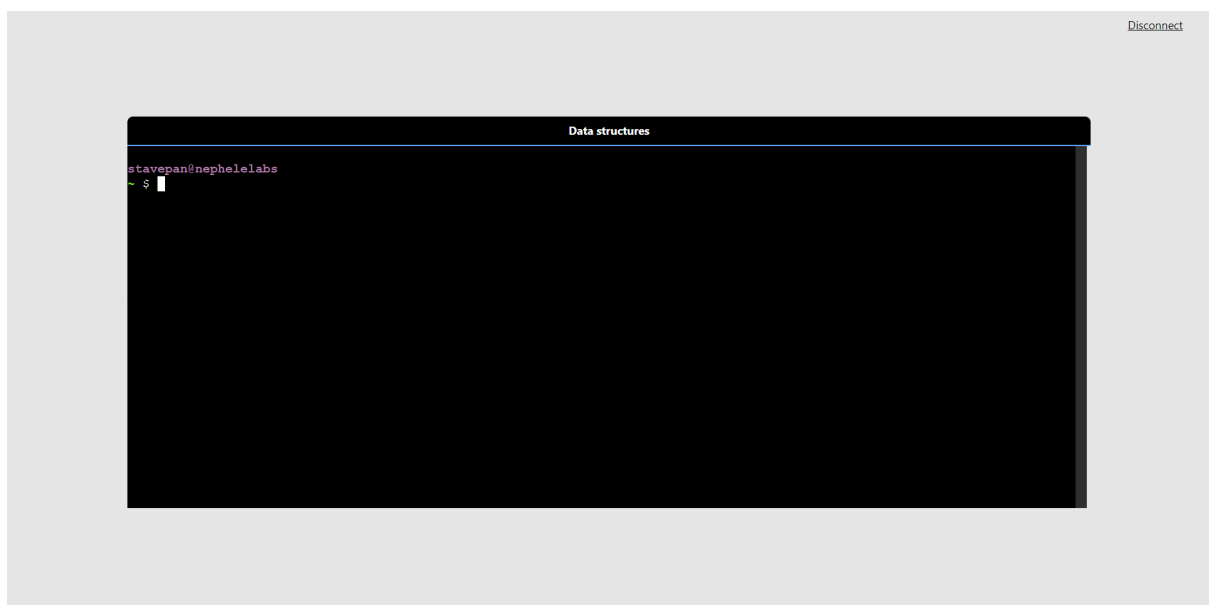
των επιλογών.



Σχήμα 4.6: Σελίδα με λίστα περιβαλλόντων

Άνοιγμα του περιβάλλοντος

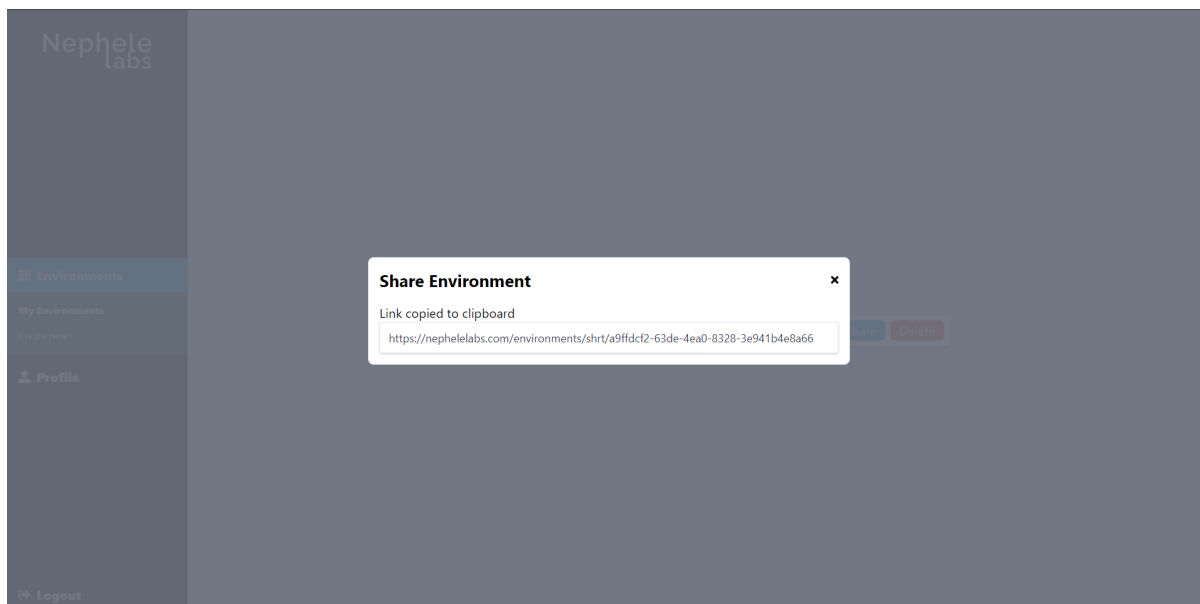
Σε περίπτωση που ο χρήστης πατήσει το κουμπί Open γίνεται ανακατεύθυνση στην σελίδα της κονσόλας [Σχήμα 4.7]. Η σελίδα της κονσόλας περιλαμβάνει ένα πλήρως λειτουργικό τερματικό όπου ο χρήστης αλληλεπιδρά με το περιβάλλον του.



Σχήμα 4.7: Σελίδα κονσόλας

Κοινοποίηση

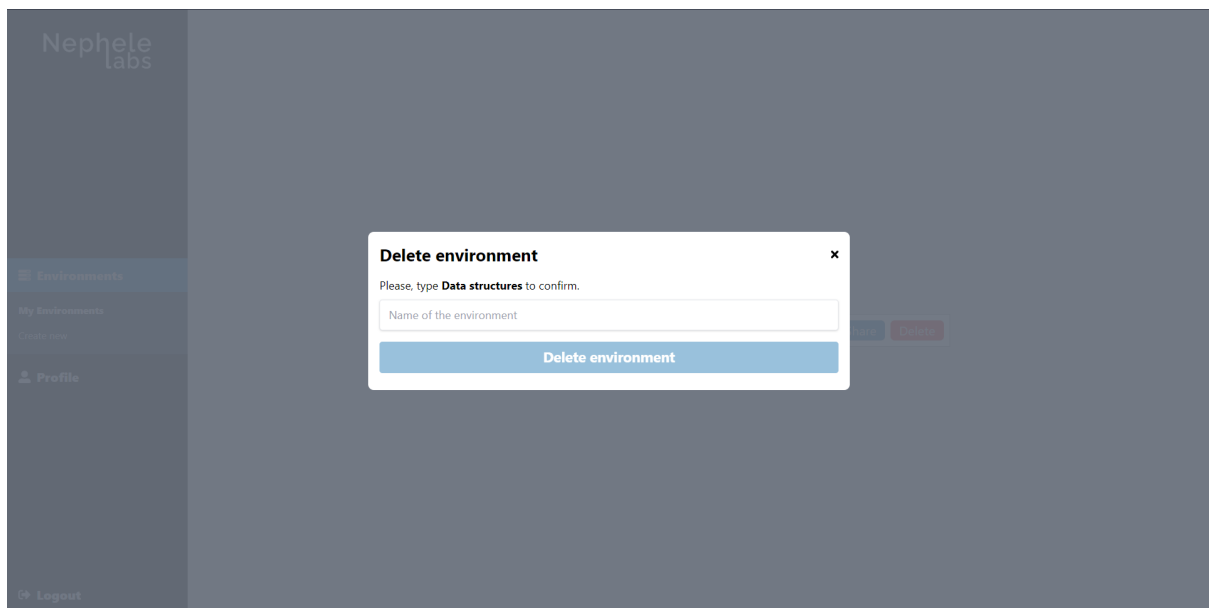
Σε περίπτωση που ο χρήστης πατήσει το κουμπί Share, θα ανοίξει ένα αναδυόμενο παράθυρο στην σελίδα όπου θα του εμφανίζει έναν σύνδεσμο [Σχήμα 4.8]. Όποιος χρήστης έχει στην κατοχή του τον συγκεκριμένο σύνδεσμο, έχει τη δυνατότητα να κλωνοποιήσει το περιβάλλον.



Σχήμα 4.8: Σελίδα κοινοποίησης

Διαγραφή

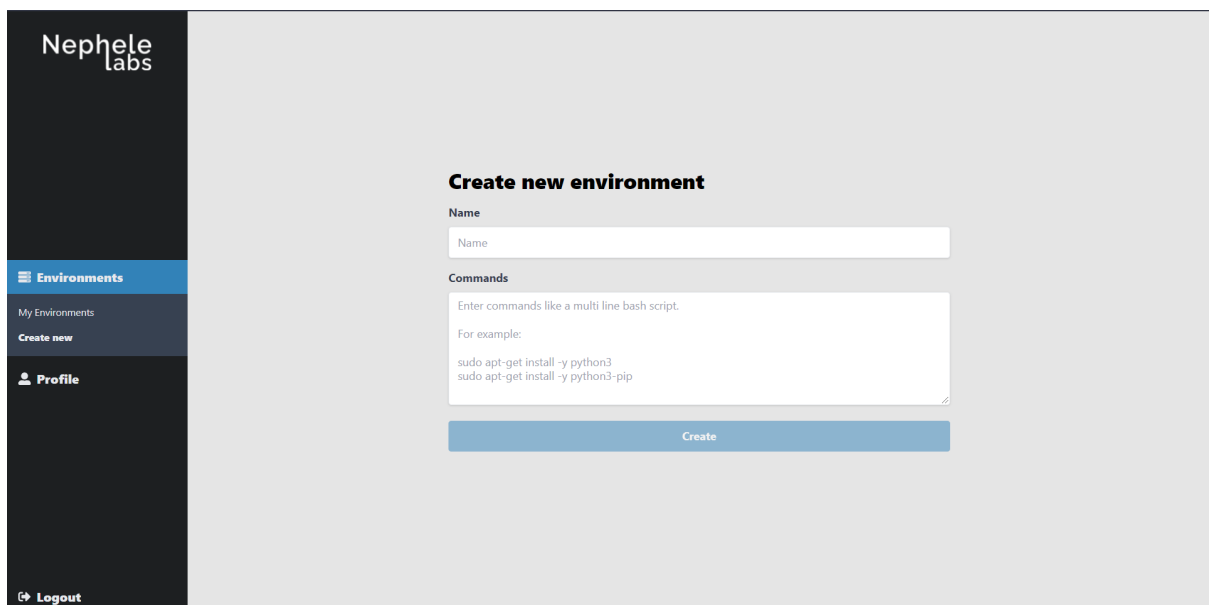
Σε περίπτωση που ο χρήστης πατήσει το κουμπί Delete, θα ανοίξει επίσης ένα αναδυόμενο παράθυρο [Σχήμα 4.9]. Ο χρήστης θα καλείται να συμπληρώσει το όνομα του περιβάλλοντος που επιθυμεί να διαγράψει. Σε περίπτωση που είναι σωστό, το κουμπί της επιβεβαίωσης θα ενεργοποιηθεί.



Σχήμα 4.9: Σελίδα διαγραφής

4.2.5 Δημιουργία περιβάλλοντος

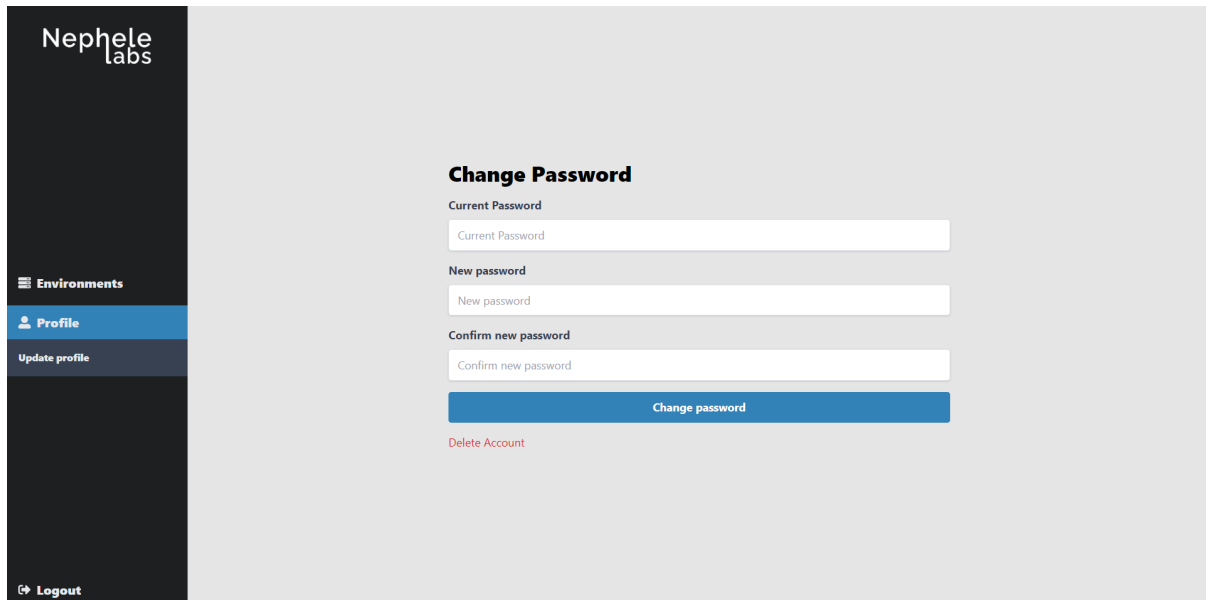
Η σελίδα που είναι υπεύθυνη για την δημιουργία ενός περιβάλλοντος περιλαμβάνει μια φόρμα με τα βασικά στοιχεία [Σχήμα 4.10]. Ο χρήστης καλείται να συμπληρώσει το όνομα και τις εντολές της επιλογής του. Όσον αφορά το πεδίο των εντολών, ο χρήστης θα πρέπει να γράψει τις εντολές όπως γράφεται ένα τμήμα κώδικα σε γλώσσα Bash[31]. Εάν η δημιουργία είναι επιτυχημένη τότε γίνεται ανακατεύθυνση στην σελίδα με τη λίστα των περιβαλλόντων [Σχήμα 4.6]. Σε περίπτωση που δεν είναι, επιστρέφεται απλά μήνυμα λάθους στην φόρμα.



Σχήμα 4.10: Σελίδα δημιουργίας περιβάλλοντος

4.2.6 Αλλαγή κωδικού

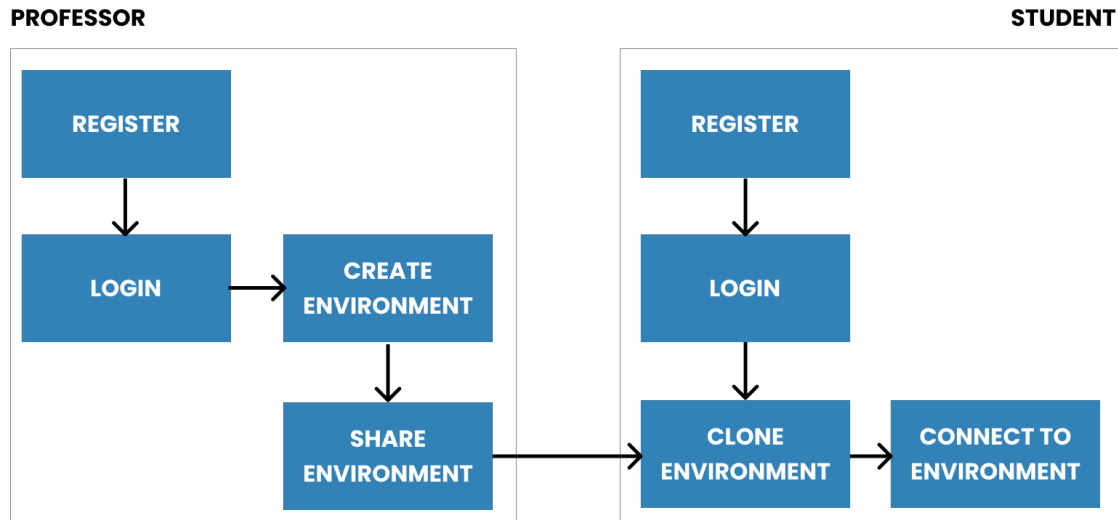
Αυτή η σελίδα περιλαμβάνει μια φόρμα όπου ο χρήστης μπορεί να χρησιμοποιήσει για να αλλάξει τον κωδικό του [Σχήμα 4.11]. Συμπληρώνοντας τον παλιό του κωδικό και δύο φορές τον καινούριο που επιθυμεί του εμφανίζεται το ανάλογο μήνυμα.



Σχήμα 4.11: Σελίδα αλλαγής κωδικού

4.3 Διαδικασία χρήσης

Οι δύο βασικές περσόνες της εφαρμογής είναι ο καθηγητής και ο φοιτητής. Η διαδικασία χρήσης [Σχήμα 4.12] ξεκινάει ως εξής. Αρχικά, πρέπει να δημιουργηθεί λογαριασμός στην πλατφόρμα και από τις δύο περσόνες όπως επίσης και να συνδεθούν με τα μοναδικά τους στοιχεία. Η διαδικασία συνεχίζεται με τον καθηγητή να δημιουργεί το περιβάλλον της επιλογής του. Μετά από τη δημιουργία του, δημιουργεί ένα μοναδικό σύνδεσμο τον οποίο αποστέλνει στους φοιτητές. Έπειτα, εφόσον οι φοιτητές έχουν δημιουργήσει λογαριασμό και είναι συνδεδεμένοι στην πλατφόρμα, επισκέπτονται τον σύνδεσμο που τους στάλθηκε από τον καθηγητή και έτσι κλωνοποιούν το περιβάλλον. Τέλος, συνδέονται στο περιβάλλον και αλληλεπιδρούν με αυτό.



Σχήμα 4.12: Διάγραμμα χρήσης

4.4 Επαναχρησιμοποιούμενα Μέρη

Πολλές σελίδες έχουν πολλά κοινά σημεία μεταξύ τους, οπότε για την οργάνωση του κώδικα δημιουργήθηκαν επαναχρησιμοποιούμενα μέρη. Πιο συγκεκριμένα είναι τα παρακάτω:

1. Form

- Input
- Button
- Error

2. Sidebar

- Button
- Bottom Button
- Sub Button

3. Head

4. Loader

5. Modal

4.4.1 Form

Το μέρος Form [Σχήμα 4.13] δημιουργήθηκε διότι υπήρχε η ανάγκη για πολλαπλές φόρμες στην εφαρμογή. Πιο συγκεκριμένα, στις σελίδες σύνδεσης, δημιουργίας λογαριασμού, δημιουργία περιβάλλοντος και αλλαγής κωδικού πρόσβασης.

Username

Password



Register

Σχήμα 4.13: Μέρος Form

Σε αυτό το μέρος δημιουργήθηκαν ακόμη τρία μικρότερα μέρη τα οποία είναι τα εξής:

- Input
- Button
- Error

Input

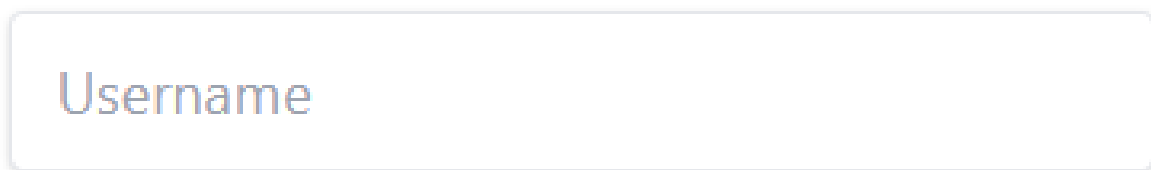
Το μέρος Input [Σχήμα 4.14] είναι το κομμάτι μιας φόρμας το οποίο δέχεται είσοδο (π.χ. HTML input). Τα props τα οποία δέχεται είναι τα εξής:

```
Interface Props contains  
  name: string  
  type: "text" | "password"  
  placeholder: string  
  label?: string  
  as?: "input" | "select" | "textarea"  
  className?: string  
  value?: string  
  defaultValue?: string  
end
```

Το JSX το οποίο επιστέφει είναι το εξής:

```
<div className="mb-4">
  <label className="block text-gray-700 text-md font-bold mb-2"
    htmlFor={name}>{label}</label>
  <Field
    as={!as? ("input"): (as)}
    className={
      "focus:outline-none focus:ring-2 focus:ring-blue-450 shadow
        appearance-none border rounded h-11 w-full py-2 px-3 text-gray-700
        leading-tight focus:outline-none focus:shadow-outline "
      + className
    }
    type={type}
    name={name}
    placeholder={placeholder}
    value={value}
    defaultValue={defaultValue}
  />
</div>
```

Username



Σχήμα 4.14: Μέρος Input

Error

Το μέρος Error [Σχήμα 4.15] είναι ένα στοιχείο HTML div το οποίο είναι με κόκκινο χρώμα και εμφανίζεται στις φόρμες σε περίπτωση λάθους. Τα props τα οποία δέχεται είναι τα εξής:

```
Interface Props contains
| message: string
end
```

Το JSX το οποίο επιστέφει είναι το εξής:

```
<div className="text-red-500 text-sm mb-2">{message}</div>
```

Password is required

Σχήμα 4.15: Μέρος Error

Button

Το μέρος Button [Σχήμα 4.16] είναι ένα στοιχείο HTML button το οποίο είναι με μπλε χρώμα και χρησιμοποιείται για την υποβολή της εκάστοτε φόρμας. Ο λόγος που έχει δημιουργηθεί είναι επειδή υπήρχε η ανάγκη για επαναλαμβανόμενη χρήση του αλλά με διαφορετικό όνομα και με δυνατότητα απενεργοποίησης. Τα props τα οποία δέχεται είναι τα εξής:

```
Interface Props contains
|   name: string
|   disabled: boolean
end
```

Το JSX το οποίο επιστέφει είναι το εξής:

```
<button
  type="submit"
  className="disabled:opacity-50 disabled:cursor-default w-full h-11
    bg-blue-button text-white font-bold py-2 px-4 rounded
    focus:outline-none focus:shadow-outline"
  disabled={disabled}
>
  {name}
</button>
```



Σχήμα 4.16: Μέρος Button

4.4.2 Sidebar

Το μέρος Sidebar [Σχήμα 4.17] δημιουργήθηκε διότι χρησιμοποιείται σε όλες τις σελίδες στο ταμπλό του χρήστη. Πιο συγκεκριμένα, στις σελίδες της λίστας των

περιβαλλόντων, στην αλλαγή στοιχείων του χρήστη και στη δημιουργία περιβάλλοντος.



Σχήμα 4.17: Sidebar Component

Λόγω του Sidebar μέρους δημιουργήθηκαν ακόμη τρία μικρότερα μέρη:

- Button
- Sub Button
- Bottom Button

Button

Το μέρος Button [Σχήμα 4.18] είναι το μεγαλύτερο κουμπί που υπάρχει στο Sidebar και χρησιμοποιείται για λόγους κατηγοριοποίησης των λειτουργιών της εφαρμογής. Για παράδειγμα, υπάρχει το κουμπί Environments το οποίο έχει τις επιλογές για λίστα και δημιουργία περιβαλλόντων. Τα props τα οποία δέχεται είναι τα εξής:

```
Interface Props contains  
  name: string  
  icon: IconDefinition  
  link: string  
end
```

Το JSX το οποίο επιστέφει είναι το εξής:

```
<div>  
  <Link href={link}>  
    <a  
      className={`flex flex-row text-lg items-center h-12 w-64 pl-4 mt-2  
        ${active? "bg-blue-button cursor-default" : ""}`}  
    >
```

```

    <FontAwesomeIcon
      className="text-gray-200"
      icon={icon}
    />
    <span className="text-gray-200 font-black pl-2">{name}</span>
  </a>
</Link>
{active? (
  <div className="flex flex-col">
    {children}
  </div>
):(null)}
</div>

```



Σχήμα 4.18: Button Component

SubButton

Το μέρος SubButton [Σχήμα 4.19] χρησιμοποιείται για την δημιουργία των μικρότερων κουμπιών κάτω από τις βασικές κατηγορίες των λειτουργιών της εφαρμογής. Για παράδειγμα, όταν το μέρος Button έχει το όνομα Environments, τα SubButtons θα έχουν όνομα "Δημιουργία περιβάλλοντος" και "Λίστα περιβαλλόντων". Τα props τα οποία δέχεται είναι τα εξής:

```

Interface Props contains
  | name: string
  | link: string
  | active: boolean
end

```

Το JSX το οποίο επιστέφει είναι το εξής:

```

<div className={"cursor-default bg-gray-700 text-gray-200 text-sm pl-4 pb-4 first:py-4 "} >
  <Link href={link}>
    <a className={active?("font-black"):( "hover:text-white")}>
      <span className={` ${active? ("cursor-default") : ("cursor-pointer")} `}>{name}</span>
    </a>
  </Link>
</div>

```

My Environments

Σχήμα 4.19: Μέρος SubButton

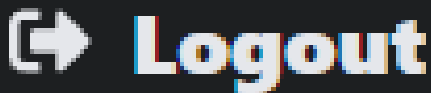
BottomButton

Το μέρος BottomButton [Σχήμα 4.20] έχει αρκετές ομοιότητες με το μέρος Button. Ο λόγος που δημιουργήθηκε ήταν η ανάγκη που υπήρχε για ένα δεύτερο μενού στο κάτω μέρος της πλαϊνής μπάρας. Τα props τα οποία δέχεται είναι τα εξής:

```
Interface Props contains
  name: string
  icon: IconDefinition
  onClick: () => void
end
```

Το JSX το οποίο επιστέφει είναι το εξής:

```
<div
  className="cursor-pointer flex flex-row text-lg items-center h-12 w-64 pl-4"
  onClick={onClick}
>
  <FontAwesomeIcon
    className="text-gray-200"
    icon={icon}
  />
  <span className="text-gray-200 font-black pl-2">{name}</span>
</div>
```



Σχήμα 4.20: Μέρος BottomButton

4.4.3 Head

Το μέρος Head δημιουργήθηκε διότι υπήρχε η ανάγκη να μπουν όλα τα μεταδεδομένα της εφαρμογής και να επαναχρησιμοποιούνται σε όλες τις σελίδες αλλάζοντας μόνο τις πληροφορίες που διαφέρουν από σελίδα σε σελίδα. Σε αυτό το στάδιο

της εφαρμογής είναι μόνο ο τίτλος της κάθε σελίδας αλλά στο μέλλον πιθανόν να είναι περισσότερα. Τα props που δέχεται είναι τα εξής:

```
Interface Props contains  
| title?: string  
end
```

Το JSX το οποίο επιστέφει είναι το εξής:

```
<NextHead>  
  {!title? (  
    <title>Nephele Labs</title>  
  ): (  
    <title>{title + " | Nephele Labs"}</title>  
  )  
}  
  <link rel="shortcut icon" href="/images/favicon.png" />  
  <meta name="viewport" content="initial-scale=1.0, width=device-width" />  
</NextHead>
```

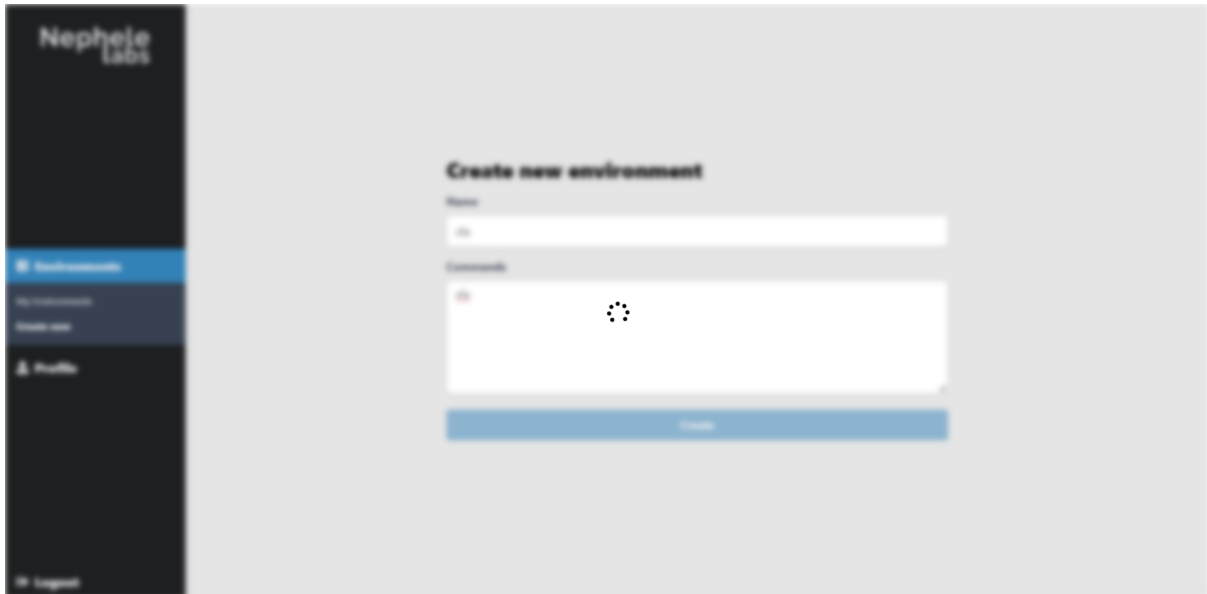
4.4.4 Loader

Το μέρος Loader [Σχήμα 4.21] δημιουργήθηκε με σκοπό τη βελτίωση της εμπειρίας του χρήστη. Πιο συγκεκριμένα χρησιμοποιείται σε όλες τις διαδικασίες οι οποίες είναι χρονοβόρες για να δείχνει στους χρήστες ότι υπάρχει πρόοδος στα αιτήματα τους και ότι δεν έχει κολλήσει η εφαρμογή. Τα props που δέχεται είναι τα εξής:

```
Interface Props contains  
| active: boolean  
end
```

Το JSX το οποίο επιστέφει είναι το εξής:

```
<div>  
  <div  
    className={` ${active? "filter blur-sm" : ""} `}  
  >  
    {children}  
  </div>  
  <FontAwesomeIcon  
    className={` ${active? "absolute top-1/2 left-1/2 -translate-x-1/2  
      -translate-y-1/2 animate-spin" : "hidden"} `}  
    icon={faSpinner}  
    size='2x'  
  />  
</div>
```



Σχήμα 4.21: Μέρος Loader

4.4.5 Modal

Το τελευταίο μέρος της εφαρμογής είναι το Modal [Σχήμα 4.22]. Το Modal είναι ένα αναδυόμενο παράθυρο το οποίο χρησιμοποιείται για αρκετές λειτουργίες της εφαρμογής. Για παράδειγμα, για την διαγραφή ή την κοινοποίηση ενός περιβάλλοντος. Τα props τα οποία δέχεται είναι τα εξής:

Interface Props contains

```

  title: string
  isOpen: boolean
  onClose: () => void

```

end

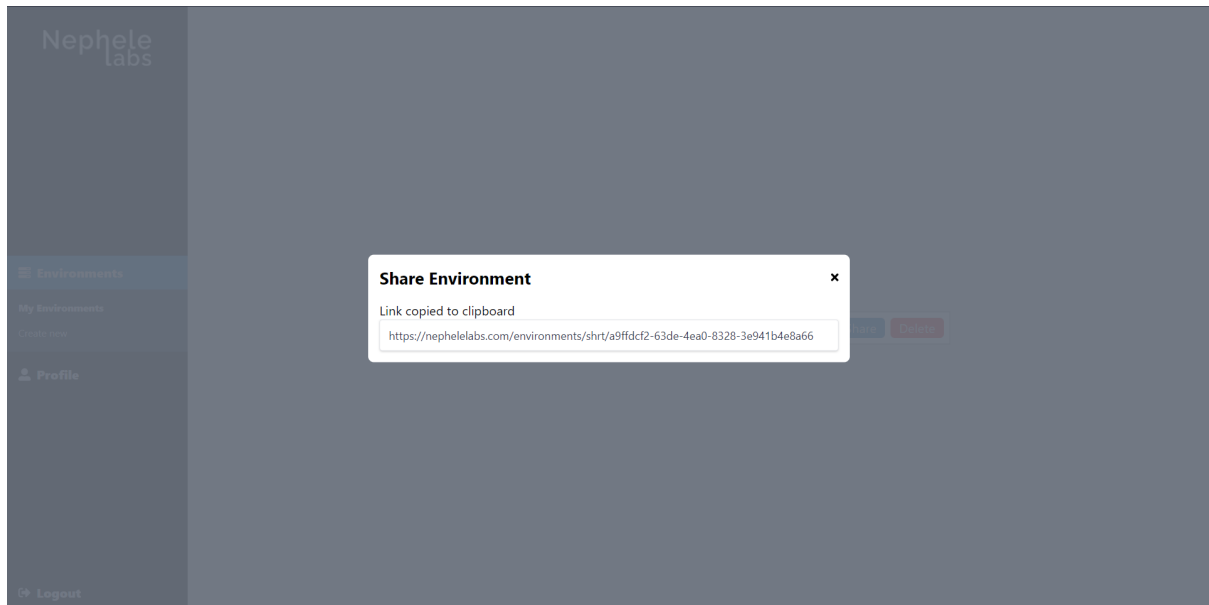
Το JSX το οποίο επιστέφει είναι το εξής:

```

<ReactModal
  isOpen={isOpen}
  onRequestClose={onClose}
  className="inset-0 m-auto bg-gray-700 bg-opacity-70 absolute"
>
  <div
    className="bg-white p-4 m-auto rounded-lg w-2/5 top-1/2 -translate-y-1/2
      transform relative"
  >
    <div className="flex justify-between items-center">
      <div className="text-2xl font-bold">{title}</div>
      <FontAwesomeIcon
        className="cursor-pointer"
        icon={faTimes}
        onClick={onClose}
      />
    </div>
  </div>

```

```
    </div>
    {children}
  </div>
</ReactModal>
```



Σχήμα 4.22: Μέρος Modal

Κεφάλαιο 5

Λειτουργικότητα της εφαρμογής

5.1 Προεπισκόπηση

Η γλώσσα στην οποία γράφτηκε η λειτουργικότητα της εφαρμογής είναι η Go.[22] Ο λόγος που επιλέχθηκε η συγκεκριμένη γλώσσα είναι η απλότητα της, η ταχύτητά της και το γεγονός ότι το Docker engine[9] το οποίο χρησιμοποιείται από την εφαρμογή είναι γραμμένο στην ίδια γλώσσα. Αυτό έγινε για να αποφευχθούν τυχόν προβλήματα συμβατότητας μεταξύ του Docker engine[9] και της εφαρμογής.

5.2 Αρχιτεκτονική

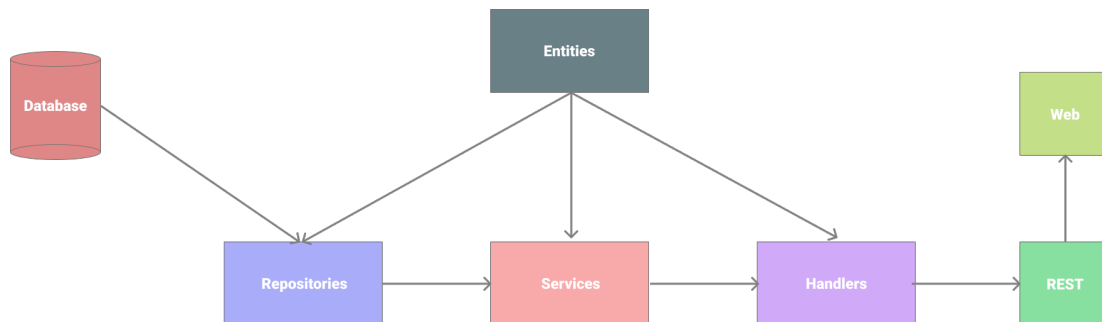
Η αρχιτεκτονική της εφαρμογής είναι εμπνευσμένη από το βιβλίο "Clean Architecture" του Robert C. Martin. [32] Ο βασικότερος λόγος για αυτή την επιλογή είναι ο κανόνας τον εξαρτήσεων. Αυτός ο κανόνας εξηγεί πως η κάθε λειτουργία είναι υπεύθυνη μόνο για τον εαυτό της. Αυτό ευνοεί την εύκολη δημιουργία unit tests για κάθε λειτουργία και συνεπώς την εύκολη και γρήγορη πρόσθεση νέων λειτουργιών ή την αναδιαμόρφωση των ήδη υπάρχοντων με μεγαλύτερη ασφάλεια. Η συγκεκριμένη αρχιτεκτονική [Σχήμα 5.1] χωρίζεται σε τέσσερα μέρη:

- Entities
- Services - Use Cases
- Repositories
- Handlers

Τα Entities είναι στην ουσία αντικείμενα πάνω στα οποία βασίζονται οι μέθοδοι της εφαρμογής. Για παράδειγμα, σε μια εφαρμογή ερωτήσεων - απαντήσεων, τα entities θα μπορούσε να είναι οι ερωτήσεις και οι απαντήσεις. Το κάθε entity έχει τα δικά του services, repositories και handlers. Στην συγκεκριμένη περίπτωση, τα entities είναι τα παρακάτω:

- User
- Environment
- URL Shortener

Τα Services - Use Cases είναι τα business rules της εφαρμογής. Πιο συγκεκριμένα, είναι οι περιπτώσεις χρήσης όπου χρησιμοποιούνται από τα entities και χρησιμοποιούν τα repositories. Είναι χρήσιμα διότι μπορούν να αλλάζουν ανάλογα με τις ανάγκες της εφαρμογής χωρίς να επηρεάζονται από εξωτερικούς παράγοντες όπως η βάση δεδομένων. Τα repositories χρησιμοποιούνται από τα services και χρησιμοποιούν εξωτερικές διεργασίες όπως η βάση δεδομένων. Αυτό συμβαίνει για να μπορούν να αλλάζουν εύκολα οι εξωτερικές διεργασίες χωρίς να χρειάζεται να επηρεαστούν οι περιπτώσεις χρήσης της εφαρμογής. Οι handlers χρησιμοποιούν τα services της εφαρμογής και είναι υπεύθυνοι για το κομμάτι της επικοινωνίας της εφαρμογής. Σε αυτό το layer δημιουργείται το REST API όπου χρησιμοποιείται από τη διεπαφή του χρήστη. Παρακάτω θα γίνει η περιγραφή των entities και των services και των repositories που χρησιμοποιούν.



Σχήμα 5.1: Clean architecture

5.3 User

5.3.1 Entity

Το πρώτο entity είναι ο χρήστης της εφαρμογής και περιγράφεται ως εξής:

```

Struct User contains
  string ID;
  string Username;
  string Password;
end
  
```

Το ID και το username είναι τα αναγνωριστικά του κάθε χρήστη και το password υπάρχει για την ασφάλεια και την αυθεντικοποίηση του.

5.3.2 Repository

Παρακάτω υπάρχει το Interface το οποίο περιγράφει το Repository των χρηστών της εφαρμογής.

Interface Repository contains

```
Insert(user User) error;  
Exists(user User) (bool, error);  
Authorize(username string, password string) (*User, error);  
GetByID(userID string) (User, error);  
ChangePassword(id string, oldPassword string, newPassword string,  
    confirmNewPassword string) error;  
Delete(ID string) error;  
end
```

Insert

Η μέθοδος Insert παίρνει σαν παράμετρο το αντικείμενο User και αποθηκεύει τα στοιχεία του αντικειμένου στην βάση δεδομένων. Στην περίπτωση που η αποθήκευση είναι επιτυχής επιστρέφει την τιμή null αλλιώς επιστρέφει το error που έλαβε από τη βάση δεδομένων.

Exists

Η μέθοδος Exists παίρνει σαν παράμετρο το αντικείμενο User και ελέγχει στην βάση δεδομένων αν υπάρχει χρήστης με το ίδιο username με αυτό που έχει το αντικείμενο User. Στην περίπτωση που υπάρχει επιστρέφει την τιμή True και ένα error το οποίο ενημερώνει ότι ο χρήστης υπάρχει ήδη στην βάση δεδομένων. Σε αντίθετη περίπτωση επιστρέφει την τιμή False και την τιμή null.

Authorize

Η μέθοδος Authorize παίρνει σαν παραμέτρους την τιμή username και την τιμή password. Αρχικά, κάνει μια αναζήτηση στην βάση δεδομένων βάσει του username. Στην περίπτωση που δεν βρεθεί κάποιος χρήστης επιστρέφει τις τιμές null και ένα error το οποίο ενημερώνει ότι δεν βρέθηκε χρήστης με το συγκεκριμένο username. Σε αντίθετη περίπτωση συγκρίνει το password που πήρε σαν παράμετρο με το hash στην βάση δεδομένων. Εάν είναι το ίδιο, τότε επιστρέφει τα στοιχεία του χρήστη και την τιμή null, αντίθετα επιστρέφει την τιμή null και ένα error το οποίο αναγράφει πως ο κωδικός που έλαβε σαν παράμετρο είναι λάθος.

GetByID

Η μέθοδος GetByID παίρνει σαν παράμετρο ένα ID και κάνει αναζήτηση στην βάση δεδομένων των χρηστών με βάση αυτό το ID. Σε περίπτωση που βρεί κάποιον χρήστη τον επιστρέφει μαζί με μια δεύτερη τιμή null. Σε αντίθετη περίπτωση επιστρέφει null και ένα error το οποίο ενημερώνει πως δεν βρέθηκε χρήστης με αυτό το ID.

ChangePassword

Η μέθοδος ChangePassword παίρνει σαν παραμέτρους το ID του χρήστη, τον παλιό του κωδικό και άλλες δύο τιμές που περιλαμβάνουν τον καινούριο του κωδικό. Αυτό που κάνει αρχικά είναι το να κάνει αναζήτηση στην βάση δεδομένων των χρηστών βάσει το ID που πήρε σαν παράμετρο. Στην περίπτωση που η βάση δεδομένων δεν βρει κάποιον χρήστη με το συγκεκριμένο ID, η μέθοδος επιστρέφει ένα error ενημερώνοντας για την κατάσταση. Σε αντίθετη περίπτωση, γίνεται σύγκριση του παλιού κωδικού του χρήστη με το hash που υπάρχει στην βάση δεδομένων. Αν η ταυτοποίηση είναι επιτυχής γίνεται ενημέρωση του κωδικού στην βάση δεδομένων και επιστρέφεται η τιμή null, αλλιώς επιστρέφεται ένα error το οποίο αναγράφει την αδυναμία ταυτοποίησης.

Delete

Η μέθοδος Delete παίρνει σαν παράμετρο το ID ενός χρήστη. Αυτό που συμβαίνει είναι πως διαγράφει τον χρήστη που υπάρχει στην βάση δεδομένων με το συγκεκριμένο ID. Σε περίπτωση που υπάρχει κάποιο πρόβλημα, ενημερώνει επιστρέφοντας ένα error περιγράφοντας το πρόβλημα, αλλιώς επιστρέφει την τιμή null.

5.3.3 Services

Παρακάτω υπάρχει το Interface το οποίο περιλαμβάνει μια λίστα με τα Services των χρηστών της εφαρμογής.

Interface Services contains

```
Register(user User) error;  
Login(username string, password string) (string, error);  
GetByID(userID string) (User, error);  
ChangePassword(id string, oldPassword string, newPassword string,  
    confirmNewPassword string) error;  
Delete(ID string) error;
```

end

Register

Αρχικά, η μέθοδος Register παίρνει σαν παράμετρο έναν χρήστη. Έπειτα, ελέγχει αν η μορφή των στοιχείων του είναι έγκυρη. Πιο συγκεκριμένα, ελέγχει αν το username είναι μόνο μια λέξη, ότι δεν είναι κενό και ότι δεν περιέχει κάποιον ιδιαίτερο χαρακτήρα. Εκτός από αυτό, ελέγχει αν το password είναι πάνω από οχτώ χαρακτήρες, αν περιλαμβάνει και μικρούς αλλά και κεφαλαίους χαρακτήρες, όπως επίσης αριθμούς και ιδιαίτερους χαρακτήρες. Στην περίπτωση που δεν ισχύουν οι παραπάνω προϋποθέσεις, η μέθοδος επιστρέφει error. Σε αντίθετη περίπτωση, ελέγχει αν ο χρήστης υπάρχει ήδη βάσει του username του καλώντας τη μέθοδο Exists από το repository. Στην περίπτωση που υπάρχει ήδη επιστρέφει error. Αν δεν έχει προκύψει κάποιο πρόβλημα ως εδώ, η μέθοδος παράγει ένα μοναδικό ID για να το δώσει στον χρήστη και γίνεται hashed ο κωδικός. Τέλος, καλείται η μέθοδος Insert από το repository με σκοπό να εισέρθει ο χρήστης στη

βάση δεδομένων. Εάν έχει προκύψει κάποιο πρόβλημα, η συνάρτηση επιστρέφει error, αλλιώς επιστρέφει null.

Login

Η μέθοδος Login είναι υπεύθυνη για την αυθεντικοποίηση του χρήστη. Αρχικά, παίρνει σαν παράμετρο το username και το password του χρήστη και καλεί την μέθοδο Authorize από το repository του χρήστη. Εάν η αυθεντικοποίηση δεν είναι επιτυχής, η μέθοδος επιστρέφει ένα κενό string και το error που επέστρεψε η μέθοδος Authorize. Στην περίπτωση που είναι επιτυχής, ακολουθεί την μέθοδο αυθεντικοποίησης με JSON Web Tokens. [33] Τα claims που αποθηκεύονται στο Token είναι το ID και το username του χρήστη. Αυτό συμβαίνει για να είναι διαθέσιμες αυτές οι τιμές στις υπόλοιπες μεθόδους που θα έχουν πρόσβαση στο Token.

GetByID

Η μέθοδος GetByID παίρνει σαν παράμετρο το ID του χρήστη και καλεί απευθείας την ομόνυμη μέθοδο από το repository του χρήστη και επιστρέφει τις ίδιες τιμές.

ChangePassword

Η μέθοδος ChangePassword παίρνει σαν παραμέτρους το ID του χρήστη, τον παλιό κωδικό του και άλλες δυο παραμέτρους με τον καινούριο του κωδικό. Αρχικά, ελέγχει αν οι δυο παράμετροι με τον καινούριο κωδικό έχουν την ίδια τιμή. Στην περίπτωση που αυτή η συνθήκη δεν ισχύει επιστρέφει μήνυμα λάθους με αυτή την πληροφορία. Σε αντίθετη περίπτωση, ελέγχει αν ο παλιός και ο νέος κωδικός είναι ίδιοι. Εάν ισχύει, επιστρέφει μήνυμα λάθους. Έπειτα, συνεχίζει ελέγχοντας αν ο νέος κωδικός είναι έγκυρος. Πιο συγκεκριμένα, ο κωδικός θα πρέπει να πληρεί τα κριτήρια που αναφέρθηκαν στην μέθοδο Register. Εάν δεν πληρούνται, επιστρέφεται μήνυμα λάθους. Τέλος, ο νέος κωδικός γίνεται hashed και καλείται η ομόνυμη μέθοδος από το repository του χρήστη με σκοπό να ενημερωθεί ο κωδικός του χρήστη στη βάση δεδομένων.

Delete

Η μέθοδος Delete παίρνει σαν παράμετρο το ID του χρήστη και είναι υπεύθυνη για να τον διαγράψει από την εφαρμογή. Αρχικά, καλεί μια μέθοδο η οποία θα αναλυθεί σε επόμενο κεφάλαιο όπου διαγράφει όλα τα περιβάλλοντα του χρήστη. Σε περίπτωση λάθους επιστρέφει το λάθος. Έπειτα, καλείται η ομόνυμη μέθοδος από το repository του χρήστη με σκοπό να διαγραφτεί ο χρήστης από τη βάση δεδομένων.

5.3.4 Handlers

Οι handlers του χρήστη είναι υπεύθυνοι για τη δημιουργία του REST API της εφαρμογής με σκοπό να είναι εφικτή η επικοινωνία της διεπαφής του χρήστη με τον server της εφαρμογής. Συγκεκριμένα, οι handlers του χρήστη έχουν το

πρόθημα /users. Παρακάτω, θα γίνει η αναλυτική περιγραφή του κάθε handler ξεχωριστά.

Middleware

Το middleware είναι κάποιες συναρτήσεις οι οποίες καλούνται πριν από κάποιες συγκεκριμένες μεθόδους που αφορούν τους handlers των χρηστών. Στην συγκεκριμένη περίπτωση αυτές οι συναρτήσεις είναι δύο:

- CORS
- Authentication

Η συνάρτηση που αφορά το CORS φροντίζει έτσι ώστε η εφαρμογή να δέχεται requests μόνο από συγκεκριμένες IP και το περιεχόμενο των requests να είναι συγκεκριμένο. Αυτό συμβαίνει κυρίως για λόγους ασφαλείας. Αυτή η συνάρτηση καλείται για όλους τους handlers. Η συνάρτηση που αφορά το Authentication καλείται για συγκεκριμένους handlers όπου θα αναφερθούν παρακάτω. Αυτό που κάνει είναι το να ελέγχει το token του χρήστη που έρχεται μαζί με το request αν υπεγράφη από τον συγκεκριμένο server και αν είναι έγκυρο. Στην περίπτωση που δεν είναι έγκυρο, εμποδίζει την είσοδο του request στον handler που αφορά.

Login

POST /users/login

Body:

username: string
password: string

Returns

200 -> token: string
401 -> error: string

Η μέθοδος Login συνδέεται με το παραπάνω endpoint και δέχεται μόνο POST Requests. Οι παράμετροι της είναι το username και το password ενός χρήστη σε μορφή JSON. Αυτά τα δεδομένα αποκωδικοποιούνται και παιρνιούνται σαν παράμετρο στην μέθοδο Login από τα services του χρήστη. Εάν αυτό το service επιστρέψει λάθος, τότε το handler θα επιστρέψει σε μορφή JSON ένα error με τον κωδικό 401. Εάν τα στοιχεία του χρήστη είναι έγκυρα, τότε θα επιστραφεί σε μορφή JSON το JSON Web Token [33] με κωδικό 200.

Register

POST /users/register

Body:

username: string
password: string

Returns

200 -> No body

400 -> error: string

Η μέθοδος Register συνδέεται με το παραπάνω endpoint και δέχεται επίσης μόνο POST Requests. Οι παράμετροι ξανά είναι το username και το password του χρήστη. Ακολουθεί η ίδια διαδικασία που αφορά την αποκωδικοποίηση. Έπειτα, καλείται η μέθοδος Register από τα services του χρήστη. Στην περίπτωση που υπάρχει κάποιο πρόβλημα, επιστρέφεται κάποιο error σε μορφή JSON μαζί με τον κωδικό 400, ενώ εάν η εγγραφή του χρήστη ήταν επιτυχής, επιστρέφεται ο κωδικός 200.

Get

GET /users/get

Returns

200 -> username: string, password: string

400 -> error: string

401 -> No Body

Η μέθοδος Get δέχεται μόνο GET Requests και καλείται όταν το request γίνει στο παραπάνω endpoint. Σε αυτή τη μέθοδο, χρησιμοποιείται το Authentication Middleware που αναφέρθηκε παραπάνω. Αυτό συμβαίνει γιατί ο σκοπός της μεθόδου είναι το να επιστρέψει πίσω τον αυθεντικοποιημένο χρήστη. Αυτό το καταφέρνει καλώντας την μέθοδο GetByID από τα services του user. Σε περίπτωση που επιστρέψει λάθος, η μέθοδος απαντά με κάποιο error σε μορφή JSON μαζί με τον κωδικό 400. Στην περίπτωση που ο χρήστης δεν έχει ταυτοποιηθεί, μπλοκάρεται το request με κωδικό 401. Τέλος, στην περίπτωση που δεν προκύψει κάποιο πρόβλημα επιστρέφει το username και τον κρυπτογραφημένο κωδικό του χρήστη μαζί με τον κωδικό 200.

Delete

DELETE /users/delete

Returns

200 -> No Body

400 -> error: string

401 -> No Body

Η μέθοδος Delete δέχεται αποκλειστικά DELETE Requests και καλείται στο endpoint που αναφέρθηκε παραπάνω. Για να καλεστεί επιτυχώς, θα πρέπει να περάσει από το middleware που αφορά την αυθεντικοποίηση του χρήστη. Αυτό που κάνει είναι το να καλέσει την μέθοδο Delete από τα services του χρήστη και αναλόγως το αποτέλεσμα της, να επιστρέψει τους κωδικούς που αναγράφονται παραπάνω. Πιο συγκεκριμένα, τον κωδικό 200 για επιτυχή διαγραφή του χρήστη, τον κωδικό 401 στην περίπτωση που η ταυτοποίηση του χρήστη ήταν αδύνατη ή τον κωδικό 400 σε περίπτωση που υπάρχει κάποιο λάθος, το οποίο θα επιστραφεί και αυτό σε μορφή JSON.

ChangePassword

POST /users/change-password

Body:

```
oldPassword: string
newPassword: string
confirmNewPassword: string
```

Returns

```
200 -> No body
400 -> error: string
401 -> No body
```

Ο handler `ChangePassword` καλείται όταν γίνεται request στο παραπάνω endpoint και δέχεται μόνο POST Requests. Για να καλεστεί η μέθοδος με επιτυχία, πρέπει ο χρήστης που την καλεί να είναι ταυτοποιημένος. Το σώμα που δέχεται είναι ο παλιός κωδικός του χρήστη, και ο καινούριος του σε δύο διαφορετικές παραμέτρους σε μορφή JSON. Μετά από την αποκωδικοποίηση του σώματος, οι ίδιοι παράμετροι χρησιμοποιούνται για να καλεστεί η ομώνυμη μέθοδος από τα services του χρήστη. Στην περίπτωση που υπάρχει κάποιο πρόβλημα με τους κωδικούς επιστρέφεται λάθος πίσω στον χρήστη σε μορφή JSON με τον κωδικό 400. Αντίθετα, επιστρέφεται ο κωδικός 200 για να ενημερώσει τον χρήστη ότι η αλλαγή του κωδικού του έγινε με επιτυχία. Αν το request δεν περάσει την ταυτοποίηση, μπλοκάζεται με την τιμή 401.

5.4 Environment

5.4.1 Entity

Το επόμενο entity είναι τα περιβάλλοντα των χρηστών και περιγράφονται ως εξής:

Struct Environment contains

```
string ID;
string Name;
string Commands;
string ImageID;
string ContainerID;
string UserID;
```

end

Το ID είναι το αναγνωριστικό του κάθε περιβάλλοντος. Το Name και τα Commands είναι τα πεδία που συμπλήρωσε ο χρήστης για να το δημιουργήσει. Τα ImageID και τα ContainerID χρησιμοποιούνται για τη συσχέτιση μεταξύ της βάσης δεδομένων και του Docker. Τέλος, το UserID χρησιμοποιείται σαν foreign κλειδί μεταξύ του χρήστη και των περιβαλλόντων.

5.4.2 Repository

Παρακάτω, ακολουθεί το Interface το οποίο περιγράφει το Repository των περιβαλλόντων της εφαρμογής.

Interface Repository contains

```
GetByID(id string) Environment;  
GetByUserID(userID string) []Environment;  
Insert(environment Environment) error;  
Delete(environment Environment) error;  
CheckIfUserOwnsContainer(userID string, containerID string) (bool,  
    error);  
CountByUserID(userID string) (int, error);  
end
```

GetByID

Η μέθοδος GetByID παίρνει σαν παράμετρο το ID ενός περιβάλλοντος και βάσει αυτού, κάνει αναζήτηση στην βάση δεδομένων. Στην περίπτωση που βρεθεί ένα περιβάλλον με το συγκεκριμένο αναγνωριστικό το επιστρέφει, αλλιώς επιστρέφει ένα κενό αντικείμενο.

GetByUserID

Η μέθοδος GetByUserID μοιάζει αρκετά με την παραπάνω μέθοδο. Η πρώτη τους διαφορά είναι πως στην συγκεκριμένη περίπτωση παίρνει σαν παράμετρο το ID του χρήστη και κάνει αναζήτηση στην βάση δεδομένων μέσω αυτού. Η τελευταία τους διαφορά είναι πως υπάρχει η δυνατότητα να επιστραφούν πολλά περιβάλλοντα εφόσον ένας χρήστης μπορεί να έχει δημιουργήσει πολλά περιβάλλοντα.

Insert

Η μέθοδος Insert παίρνει σαν παράμετρο ένα περιβάλλον και το κάνει εισαγωγή στην βάση δεδομένων. Στην περίπτωση που υπάρξει κάποιο πρόβλημα, θα επιστραφεί μήνυμα λάθους. Σε αντίθετη περίπτωση, θα επιστραφεί η τιμή null.

Delete

Η μέθοδος Delete παίρνει επίσης σαν παράμετρο ένα περιβάλλον και το διαγράφει από τη βάση δεδομένων. Στην περίπτωση λάθους το επιστρέφει, αλλιώς επιστρέφει την τιμή null.

CheckIfUserOwnsContainer

Η μέθοδος CheckIfUserOwnsContainer παίρνει σαν παραμέτρους το ID του χρήστη και το ID του container. Γίνεται αναζήτηση στην βάση δεδομένων βάσει αυτών των στοιχείων. Στην περίπτωση που βρεθεί τουλάχιστον ένα περιβάλλον επιστρέφεται η τιμή true και η τιμή null. Αυτό σημαίνει ότι ο συγκεκριμένος container ανήκει στον χρήστη. Σε αντίθετη περίπτωση ή στην περίπτωση λάθους επιστρέφεται η τιμή false μαζί με μήνυμα λάθους.

CountByUserID

Η μέθοδος CountByUserID παίρνει σαν παράμετρο το ID του χρήστη και κάνει αναζήτηση στην βάση δεδομένων για να βρει τον αριθμό των περιβαλλόντων που έχει δημιουργήσει ο χρήστης. Σε περίπτωση λάθους επιστρέφει την τιμή 0 μαζί με το μήνυμα λάθους. Σε αντίθετη περίπτωση επιστρέφει τον αριθμό των περιβαλλόντων του συγκεκριμένου χρήστη μαζί με την τιμή null.

5.4.3 Docker

Το συγκεκριμένο entity έχει μια ιδιαιτερότητα σε σχέση με τα υπόλοιπα. Εκτός από τα Repository, Services και Handlers υπάρχει και το Docker. Είναι μια σειρά από βοηθητικές μεθόδους όπου δημιουργήθηκαν με σκοπό την ευκολία του Testing, τον πιο καθαρό κώδικα και την επικοινωνία της εφαρμογής με το Docker. Παρακάτω, ακολουθεί το Interface το οποίο περιγράφει το Docker client των περιβαλλόντων της εφαρμογής.

Interface Docker contains

```
BuildImage(commands string, username string) (string, error);
CreateContainer(image string) (string, error);
AttachContainer(containerID string) (types.HijackedResponse, error);
StartContainer(containerID string) error;
StopContainer(containerID string) error;
ResizeContainer(containerID string, width uint, height uint) error;
RemoveImage(imageID string) error;
RemoveContainer(containerID string) error;
end
```

BuildImage

Η μέθοδος BuildImage είναι υπεύθυνη για την δημιουργία του Docker Image που θα χρησιμοποιούν τα περιβάλλοντα των χρηστών. Αρχικά, το Image το οποίο θα δημιουργηθεί βασίζεται ήδη σε ένα Image που δημιουργήθηκε για τους σκοπούς της εφαρμογής με όνομα nperhelelabs/nperhelelabs. Παρακάτω, ακολουθεί το Dockerfile του Base Image.

```
FROM ubuntu:20.04

RUN apt-get update

RUN apt-get install -y sudo git wget curl fonts-powerline vim zsh

RUN echo "root:pass" | chpasswd

RUN useradd -ms /bin/bash newuser
RUN usermod -aG sudo newuser

RUN echo "newuser:pass" | chpasswd

USER newuser
```

```

WORKDIR /home/newuser

RUN wget https://github.com/ohmyzsh/ohmyzsh/raw/master/tools/install.sh -O -
    | zsh || true

RUN rm -rf /home/newuser/.oh-my-zsh/themes/*

COPY nephele.zsh-theme /home/newuser/.oh-my-zsh/themes

RUN git clone https://github.com/zsh-users/zsh-autosuggestions
    ${ZSH_CUSTOM:~/.oh-my-zsh/custom}/plugins/zsh-autosuggestions

RUN git clone https://github.com/zsh-users/zsh-syntax-highlighting.git
    ${ZSH_CUSTOM:~/.oh-my-zsh/custom}/plugins/zsh-syntax-highlighting

COPY .vimrc .
COPY .zshrc .

ENTRYPOINT [ "/bin/zsh" ]

```

Αρχικά, το Base Image βασίζεται στο Ubuntu 20.04. Έπειτα, κάνει update όλα τα repositories της διανομής και εγκαθιστά κάποια βασικά πακέτα. Εκτός από αυτό, δημιουργεί κωδικό για τον root user και δημιουργεί έναν νέο user με όνομα newuser. Τέλος, γίνεται αντικατάσταση του bash με το zsh και δημιουργεί ένα συγκεκριμένο configuration για το vim και το zsh.

Έχοντας σαν βάση αυτό το Image, η μέθοδος BuildImage χρησιμοποιεί το παρακάτω Dockerfile για τη δημιουργία του Image.

```

FROM nephelelabs/nephelelabs:${IMAGE_VERSION}

USER root

RUN usermod -l ${USERNAME} newuser
RUN usermod -d /home/${USERNAME} -m ${USERNAME}
RUN groupmod -n ${USERNAME} newuser
RUN usermod -u 2000 ${USERNAME}

WORKDIR /home/${USERNAME}

RUN rm -rf /home/newuser

RUN echo "%${USERNAME} ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

USER ${USERNAME}

RUN ${COMMANDS}

```

Οι μεταβλητές commands και username αναφέρονται στις παραμέτρους της μεθόδου. Αυτό που συμβαίνει στο συγκεκριμένο Dockerfile είναι πως γίνεται μετανο-

μασία του newuser στο όνομα του χρήστη και του επιτρέπει να χρησιμοποιεί την εντολή sudo χωρίς κωδικό. Τέλος, καλεί τα commands που συμπλήρωσε ο χρήστης στην φόρμα. Στην περίπτωση που υπάρχει κάποιο λάθος επιστρέφεται πίσω ένα κενό string μαζί με το μήνυμα λάθους. Σε αντίθετη περίπτωση επιστρέφεται το ID του Image μαζί με την τιμή null.

CreateContainer

Η μέθοδος CreateContainer παίρνει σαν παράμετρο το ID ενός Image και είναι υπεύθυνη να δημιουργήσει ένα container στο σύστημα με βάση αυτό το Image. Σε περίπτωση λάθους, επιστρέφει ένα κενό string μαζί με το μήνυμα λάθους, ενώ σε αντίθετη περίπτωση επιστρέφει το ID του container μαζί με την τιμή null.

AttachContainer

Η μέθοδος AttachContainer παίρνει σαν παράμετρο το ID του Container και δημιουργεί μια σύνδεση στο συγκεκριμένο container. Εάν προκύψει μήνυμα λάθους, επιστρέφεται μια κενή σύνδεση container μαζί με την τιμή λάθους. Σε αντίθετη περίπτωση επιστρέφεται μια μεταβλητή με τις πληροφορίες σύνδεσης του container μαζί με την τιμή null.

StartContainer

Η μέθοδος StartContainer παίρνει σαν παράμετρο το ID του Container και ξεκινάει το συγκεκριμένο container. Σε περίπτωση λάθους επιστρέφει το μήνυμα λάθους, αλλιώς την τιμή null.

StopContainer

Η μέθοδος StopContainer ακολουθεί ακριβώς την ίδια διαδικασία με την StartContainer με μόνη διαφορά ότι αντί να ξεκινάει το container, το σταματάει.

ResizeContainer

Η μέθοδος ResizeContainer χρησιμοποιείται για να αλλάζει το μέγεθος ενός container. Τόσο το container όσο και το τελικό μέγεθος, το παίρνει από τις παραμέτρους της. Όσον αφορά την διαχείριση λάθους, ακολουθεί την ίδια διαδικασία με τις δύο παραπάνω μεθόδους.

RemoveImage

Η μέθοδος RemoveImage παίρνει σαν παράμετρο το ID ενός Docker Image και είναι υπεύθυνη για την διαγραφή του. Σε περίπτωση λάθους το επιστρέφει, αλλιώς επιστρέφει null

RemoveContainer

Η μέθοδος RemoveContainer κάνει την ίδια διαδικασία με την παραπάνω μέθοδο, με μόνη διαφορά ότι συμβαίνει σε Container και όχι σε Image.

5.4.4 Services

Παρακάτω, ακολουθεί το Interface το οποίο περιγράφει τα Services των περιβαλόντων της εφαρμογής.

Interface Services contains

```
GetByUserID(userID string) []Environment;  
GetByID(id string) Environment;  
Create(environment Environment, username string) error;  
Connect(containerID string) (types.HijackedResponse, error);  
Delete(uuid string) error;  
CheckIfUserOwnsContainer(userID string, containerID string) (bool,  
    error);  
DeleteAllByUserID(userID string) error;
```

end

GetByUserID

Η μέθοδος GetByUserID παίρνει σαν παράμετρο το ID του χρήστη και τέλος καλεί και επιστρέφει την ομώνυμη μέθοδο από το Repository των environments.

GetByID

Η μέθοδος GetByID παίρνει σαν παράμετρο το ID του environment και καλεί επίσης την ομώνυμη μέθοδο από το repository των environments.

Create

Η μέθοδος Create παίρνει σαν παράμετρο ένα environment και το username του χρήστη και είναι υπεύθυνη για την δημιουργία ενός περιβάλλοντος. Αρχικά, καλεί τις μεθόδους BuildImage και CreateContainer από τις βοηθητικές συναρτήσεις για το docker. Έπειτα, κάνει generate ένα καινούριο uuid για να είναι μοναδικό. Τέλος, καλεί την μέθοδο Insert από το repository των environments. Σε περίπτωση που υπάρχει λάθος στην όλη διαδικασία η μέθοδος επιστρέφει το μήνυμα λάθους. Σε αντίθετη περίπτωση επιστρέφεται η τιμή null.

Connect

Η μέθοδος Connect παίρνει σαν παράμετρο το ID ενός container και είναι υπεύθυνη για την σύνδεση στον συγκεκριμένο container. Αρχικά, καλεί την μέθοδο StartContainer από τις βοηθητικές μεθόδους για το docker με σκοπό να ξεκινήσει το ζητούμενο container. Τέλος καλεί την μέθοδο AttachContainer για να συνδεθεί απευθείας στο container. Σε περίπτωση που δεν υπάρχει κάποιο λάθος επιστρέφεται η σύνδεση του container και η τιμή null. Σε αντίθετη περίπτωση επιστρέφεται μια άδεια σύνδεση και το μήνυμα λάθους.

Delete

Η μέθοδος Delete είναι υπεύθυνη για την διαγραφή ενός περιβάλλοντος και παίρνει σαν παράμετρο το ID του περιβάλλοντος. Αρχικά, καλεί την μέθοδο GetByID

από τα repository των environments με σκοπό να πάρει όλες τις πληροφορίες για το συγκεκριμένο περιβάλλον. Έπειτα, καλεί την μέθοδο Delete από τα ίδια repositories για να γίνει η διαγραφή από την βάση δεδομένων. Όταν τελειώσει αυτή η διαδικασία, καλούνται οι μέθοδοι RemoveContainer και RemoveImage για να διαγραφεί το περιβάλλον από το σύστημα. Σε περίπτωση λάθους, επιστρέφεται το μήνυμα, αλλιώς η τιμή null.

CheckIfUserOwnsContainer

Η μέθοδος CheckIfUserOwns container καλεί και επιστρέφει την ομώνυμη μέθοδο από το repository των environments.

DeleteAllByUser

Η μέθοδος DeleteAllByUser είναι υπεύθυνη να διαγράψει όλα τα environments ενός χρήστη. Αρχικά, παίρνει σαν παράμετρο το ID ενός χρήστη και καλεί την μέθοδο GetByUserID από τα repositories των environments για να πάρει όλες τις πληροφορίες για όλα τα environments του χρήστη. Για κάθε ένα από αυτά καλείται η συνάρτηση Delete από τα repositories των περιβαλλόντων για να τα διαγράψει. Σε περίπτωση λάθους επιστρέφεται, αλλιώς επιστρέφεται η τιμή null.

5.4.5 Handlers

Οι handlers των environments δουλεύουν με τον ίδιο τρόπο με τους handlers των χρηστών. Πιο συγκεκριμένα, είναι υπεύθυνοι για την δημιουργία του REST API και έχει ως σκοπό την επικοινωνία της διεπαφής με τη λειτουργικότητα της εφαρμογής. Εκτός από αυτό, χρησιμοποιούνται τα ίδια middleware, δηλαδή το CORS και το Authentication. Το πρόθημα των handlers είναι /environments. Παρακάτω θα γίνει η περιγραφή των handlers.

GetByUserID

GET /environments/get

Returns

200 -> environments: Environment[]
401 -> error: string

Η μέθοδος GetByUserID δέχεται μόνο GET Requests και καλείται όταν γίνει request στο παραπάνω endpoint. Σε περίπτωση που ο χρήστης είναι αυθεντικοποιημένος επιστρέφονται όλα τα περιβάλλοντα του καλώντας την ομώνυμη μέθοδο από τα services των περιβαλλόντων. Σε αντίθετη περίπτωση επιστρέφεται μήνυμα λάθους με κωδικό 401.

GetByID

GET /environments/get/id

Returns

200 -> environment: Environment

401 -> error: string

Η μέθοδος GetByID δέχεται μόνο GET Requests και καλείται στο endpoint που αναφέρθηκε παραπάνω. Σε περίπτωση που ο χρήστης είναι αυθεντικοποιημένος επιστρέφεται το περιβάλλον που έχει το ID που αναφέρεται στο endpoint καλώντας την ομώνυμη μέθοδο από τα services των environments. Σε αντίθετη περίπτωση επιστρέφεται μήνυμα λάθους με κωδικό 401.

Create

POST /environments/create

Body:

environment: Environment

Returns

200 -> No Body

400 -> error: string

401 -> error: string

Η μέθοδος Create καλείται όταν γίνει POST Request στο παραπάνω endpoint. Το body που δέχεται είναι μια παράμετρος με όνομα environment και τύπο Environment. Σε περίπτωση που το body δεν είναι έγκυρο επιστρέφεται μήνυμα λάθους με κωδικό 400. Έπειτα, καλείται η μέθοδος Create από τα services των environments. Σε περίπτωση λάθους σύμβαίνει η ίδια διαχείριση λάθους. Σε περίπτωση που ο χρήστης δεν είναι αυθεντικοποιημένος επιστρέφεται μήνυμα λάθους με τιμή 401. Σε περίπτωση που δεν υπάρχει λάθος, επιστρέφεται ο κωδικός 200.

Connect

GET /environments/connect

Η μέθοδος Connect είναι περισσότερο περίπλοκη από τις παραπάνω. Αρχικά χρησιμοποιούνται WebSockets με σκοπό την επικοινωνία πελάτη-διακομιστή σε πραγματικό χρόνο. Αυτό είναι απαραίτητο γιατί αυτό το endpoint είναι υπεύθυνο για την σύνδεση του χρήστη με το περιβάλλον του και προσομοιώνει μια πραγματική σύνδεση ενός χρήστη με ένα τερματικό. Αρχικά, δέχεται δύο query variables. Η πρώτη είναι το ID του container στον οποίο θα συνδεθεί και το JWT token για λόγους αυθεντικοποίησης. Η πρώτη μέθοδος που καλείται είναι η CheckIfUserOwnsContainer από τα services των environments με σκοπό να γίνει ο έλεγχος αν ο χρήστης που προσπαθεί να συνδεθεί είναι ο ίδιος που κατέχει το περιβάλλον. Σε περίπτωση που δεν είναι επιστρέφεται μήνυμα λάθους και κλείνει η σύνδεση. Έπειτα, καλείται η μέθοδος Connect από τα services των environments με σκοπό να γίνει η σύνδεση. Σε περίπτωση λάθους, επιστρέφεται το λάθος. Όταν γίνει η σύνδεση γίνεται μια παράλληλη δραστηριότητα όπου ο διακομιστής διαβάζει το input του χρήστη και ταυτόχρονα στέλνει το output όταν χρειάζεται. Στην ουσία προωθεί το input του χρήστη στο Docker container και την απάντηση του Docker container στον χρήστη. Παρόλα αυτά, το input που δέχεται έχει συγκεκριμένη μορφή. Πιο συγκεκριμένα δέχεται μια παράμετρο Event όπου μπορεί να έχει είτε την τιμή Key είτε την τιμή Resize. Επίσης δέχεται μια

παράμετρο Body. Στην περίπτωση που το event είναι το Key, το body δέχεται το κουμπί που πατήσε ο χρήστης στο τερματικό. Στην περίπτωση που το event είναι το Resize το body είναι οι διαστάσεις του container. Σε κάθε περίπτωση αυτά τα μηνύματα προωθούνται στο container.

Disconnect

GET /environments/disconnect/container_id

Returns

200 -> No Body
400 -> error: string
401 -> error: string

Η μέθοδος Disconnect δέχεται σαν παράμετρο το ID ενός container στο endpoint της και δέχεται μόνο GET Requests. Στην περίπτωση που ο χρήστης δεν είναι αυθεντικοποιημένος επιστρέφεται μήνυμα λάθους μαζί με την τιμή 401. Σε αντίθετη περίπτωση, καλείται η μέθοδος StopContainer από τις βοηθητικές μεθόδους του docker. Σε περίπτωση λάθους επιστρέφεται το μήνυμα μαζί με τον κωδικό 400. Σε αντίθετη περίπτωση, επιστρέφεται ο κωδικός 200.

Delete

DELETE /environments/delete/id

Returns

200 -> environment: Environment
400 -> error: string
401 -> error: string

Η μέθοδος Delete δέχεται σαν παράμετρο το ID ενός περιβάλλοντος στο endpoint της και είναι υπεύθυνη για την διαγραφή του. Σε περίπτωση που ο χρήστης δεν είναι αυθεντικοποιημένος επιστρέφεται μήνυμα λάθους μαζί με την τιμή 401. Σε αντίθετη περίπτωση καλείται η μέθοδος Delete από τα services των environments. Σε περίπτωση λάθους επιστρέφεται το λάθος με τον κωδικό 400. Σε αντίθετη περίπτωση, επιστρέφεται ο κωδικός 200.

5.5 URL Shortener

5.5.1 Entity

Το τελευταίο entity είναι το Url Shortener και περιγράφεται ως εξής.

Struct UrlShortener contains

```
| ID string;  
| Name string;  
| Commands string;  
end
```

5.5.2 Repositories

Παρακάτω, ακολουθεί το Interface το οποίο περιγράφει το Repository του Url Shortener της εφαρμογής.

```
Interface Repository contains  
| Encode(urlshortener UrlShortener) error;  
| Decode(id string) (UrlShortener, error);  
end
```

Encode

Η μέθοδος Encode παίρνει σαν παράμετρο ένα αντικείμενο τύπου Url Shortener και είναι υπεύθυνη για να δημιουργήσει ένα αντικείμενο ίδιου τύπου στην βάση δεδομένων. Στην περίπτωση λάθους επιστρέφει το λάθος, αλλιώς επιστρέφει την τιμή null.

Decode

Η μέθοδος Decode παίρνει σαν παράμετρο το Id ενός αντικειμένου τύπου Url Shortener και είναι υπεύθυνη για να κάνει αναζήτηση στην βάση δεδομένων το αντικείμενο που έχει το συγκεκριμένο Id. Σε περίπτωση που βρεθεί επιστρέφει αυτό το αντικείμενο και την τιμή null. Σε αντίθετη περίπτωση επιστρέφει ένα κενό αντικείμενο και την το μήνυμα λάθους ότι δεν βρέθηκε.

5.5.3 Services

Παρακάτω ακολουθεί το Interface το οποίο περιγράφει τα Services του Url shortener της εφαρμογής.

```
Interface Service contains  
| Encode(urlshortener UrlShortener) error;  
| Decode(id string) (UrlShortener, error);  
end
```

Encode

Η μέθοδος Encode παίρνει σαν παράμετρο ένα αντικείμενο τύπου Url Shortener και είναι υπεύθυνη για την κωδικοποίηση ενός περιβάλλοντος σε ένα μικρό url. Αυτό που συμβαίνει είναι πως παράγεται ένα id και έπειτα καλείται η ομώνυμη μέθοδος από το Repository του Url Shortener. Σε περίπτωση λάθους επιστρέφεται ένα κενό string μαζί με την τιμή λάθους. Σε αντίθετη περίπτωση επιστρέφεται το Id του Url Shortener μαζί με την τιμή null.

Decode

Η μέθοδος Decode παίρνει σαν παράμετρο το id ενός Url Shortener αντικειμένου και είναι υπεύθυνη για την αποκωδικοποίηση του Id σε Url Shortener αντικείμενο. Αρχικά, καλεί την ομώνυμη μέθοδο από τα Repositories του Url Shortener. Σε περίπτωση που δεν υπάρχει λάθος επιστρέφει το αντικείμενο με το συγκεκριμένο

Id μαζί με την τιμή null. Σε αντίθετη περίπτωση επιστρέφει ένα κενό αντικείμενο τύπου Url Shortener μαζί με το μήνυμα λάθους.

5.5.4 Handlers

Οι handlers του Url Shortener δουλεύουν με τον ίδιο τρόπο που δουλεύουν και οι παραπάνω handlers. Πιο συγκεκριμένα, δημιουργούν το REST API το οποίο θα χρησιμοποιήσει ο πελάτης με σκοπό να έχει πρόσβαση στα Services του Url Shortener. Επίσης, χρησιμοποιεί τα ίδια middleware με τους handlers των περιβαλλόντων της εφαρμογής, δηλαδή το CORS και το Authentication. Τέλος, το πρόθημα των συγκεκριμένων handlers είναι /shrt. Παρακάτω, ακολουθεί η περιγραφή των handlers.

Encode

Encode

POST /shrt/encode

Body:

name: string
commands: string

Returns

200 -> id: string
400 -> error: string
401 -> error: string

Η μέθοδος Encode δέχεται μόνο POST Requests και καλείται όταν γίνεται request στο παραπάνω endpoint. Αρχικά, η μέθοδος αυτή επίσης δέχεται σαν body το όνομα και τις εντολές ενός περιβάλλοντος. Σε περίπτωση που υπάρχει πρόβλημα αυθεντικοποίησης επιστρέφεται μήνυμα λάθους μαζί με τον κωδικό 401. Σε αντίθετη περίπτωση καλείται η ομώνυμη μέθοδος από τα Services του Url Shortener. Εάν υπάρχει λάθος επιστρέφεται το μήνυμα λάθους μαζί με τον κωδικό 400. Σε περίπτωση που δεν υπάρχει λάθος επιστρέφεται το id του συγκεκριμένου αντικείμενου τύπου Url Shortener μαζί με τον κωδικό 200.

Decode

GET /shrt/decode/:id

Returns

200 -> environment: Environment
400 -> error: string
401 -> No Body

Η μέθοδος Decode καλείται όταν γίνει request στο παραπάνω endpoint, δέχεται μόνο GET requests και απαιτεί και το Url variable με όνομα ID. Σε περίπτωση που ο χρήστης δεν είναι αυθεντικοποιημένος επιστρέφεται ο κωδικός 401 και ακυρώνεται το request. Σε αντίθετη περίπτωση καλείται η ομώνυμη μέθοδος από τα Services του Url Shortener. Αναλόγως το αποτέλεσμα της συγκεκριμένης μεθόδου

είτε επιστρέφεται μήνυμα λάθους με κωδικό 400, είτε ένα περιβάλλον σε μορφή JSON με κωδικό 200.

Κεφάλαιο 6

Διαδικασίες

Υπάρχουν κάποιες διαδικασίες οι οποίες ήταν απαραίτητες να εφαρμοστούν για την επιτυχή εκτέλεση της εφαρμογής. Πιο συγκεκριμένα, είναι οι παρακάτω

1. Continuous Integration
2. Deployment

6.1 Continuous Integration

Για το version control της εφαρμογής χρησιμοποιήθηκε το λογισμικό Git και η ιντερνετική εφαρμογή GitHub. Το βασικό κλαδί της εφαρμογής είναι το κλαδί με όνομα main. Είναι πολύ σημαντικό ο κώδικας που υπάρχει σε αυτό το κλαδί να είναι απόλυτα λειτουργικός, καθαρός και τεσταρισμένος. Για αυτόν τον λόγο χρησιμοποιήθηκε continuous integration [Σχήμα 6.1] μέσω της εφαρμογής GitHub Actions. Κάθε φορά που γίνεται ένα pull request από ένα κλαδί στο κλαδί main η διαδικασία έχει ως εξής. Αρχικά, γίνεται clone το αποθετήριο σε έναν απομακρυσμένο server της εταιρίας GitHub, γίνεται η εγκατάσταση όλων των απαραίτητων πακέτων της εφαρμογής, γίνεται η εγκατάσταση της βάσης δεδομένων MySQL, γίνεται compile ο κώδικας, τρέχουν όλα τα tests και όλοι οι linters και formatters για να ελέγξουν την καθαρότητα του κώδικα.

Οι ρυθμίσεις για τον διακομιστή ακολουθούν παρακάτω:

```
name: ci-backend

on:
  pull_request:
    paths-ignore:
      - '.gitignore'
      - 'air.toml'
      - 'Dockerfile'
      - 'docker-compose.yml'
      - '**.md'
      - 'web/**'
  branches:
    - '*'
```

```

jobs:
  build:
    runs-on: ubuntu-20.04
    env:
      DEBUG: true
      SECRET_KEY: random
      DATABASE_NAME: localdb
      DATABASE_USER: root
      DATABASE_PASSWORD: root
      DATABASE_HOST: localhost
      DATABASE_PORT: 3306
      DSN: root:root@tcp(localhost:3306)/localdb
    steps:
      - name: Check out code
        uses: actions/checkout@v2

      - name: Setup Go
        uses: actions/setup-go@v2
        with:
          go-version: 1.16

      - name: Install dependencies
        run : |
          go mod download

      - name: Set up MySQL
        run: |
          sudo /etc/init.d/mysql start
          mysql -e 'CREATE DATABASE ${env.DATABASE_NAME};' -u${env.DATABASE_USER} -p${env.DATABASE_PASSWORD}

      - name: Build / Test
        run : |
          go build
          go test ./...

      - name: golangci-lint
        uses: golangci/golangci-lint-action@v2
        with:
          version: latest
          skip-go-installation: true
          args: --issues-exit-code=0

```

Οι ρυθμίσεις για τον πελάτη ακολουθούν παρακάτω:

```

name: ci-frontend

```

```

on:
  pull_request:
    paths:
      - 'web/**'

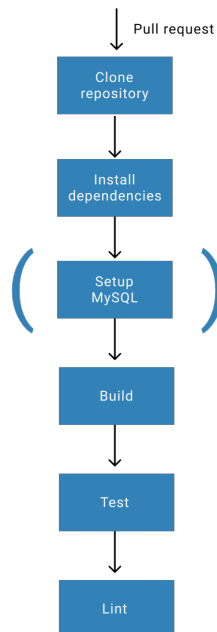
```

```
branches:
  - '*'

jobs:
  build:
    runs-on: ubuntu-20.04
    steps:
      - name: Check out code
        uses: actions/checkout@v2

      - name: Setup Nodejs
        uses: actions/setup-node@v2
        with:
          node-version: 'lts/fermium'

      - name: Install / Build / Lint / Test
        run : |
          yarn
          yarn build
          yarn lint
          yarn test
        working-directory: web
```



Σχήμα 6.1: Continuous integration pipeline στο pull request

6.2 Deployment

Το τελικό στάδιο της εφαρμογής είναι το να μεταφερθεί από το τοπικό προγραμματιστικό περιβάλλον στην παραγωγή με σκοπό να έχουν την δυνατότητα οι χρήστες να την χρησιμοποιήσουν. Η εφαρμογή απευθύνεται σε παγκόσμιο κοινό το οποίο σημαίνει πως πιθανόν να χρησιμοποιείται όλες τις ώρες της ημέρας και είναι πολύ βασικό όταν βγαίνει μια νέα έκδοση να μην πετάει έξω τους χρήστες της αλλά να κάνει την αναβάθμιση χωρίς να την καταλάβουν οι χρήστες και να υπάρχει zero-downtime. Ο τρόπος που χρησιμοποιήθηκε για την επίτευξη αυτού του σκοπού είναι ο orchestrator Docker Swarm[34] και η εφαρμογή GitHub Actions.

6.2.1 Docker swarm

Για την εκτέλεση του docker swarm δημιουργήθηκε ένα αρχείο με τις ρυθμίσεις του orchestrator και την δήλωση των απαραίτητων υπηρεσιών για την εκτέλεση της εφαρμογής. Η μόνη εξαίρεση είναι η βάση δεδομένων όπου η εγκατάσταση της έγινε εκτός του orchestrator.

Διακομιστής

Η υπηρεσία του διακομιστή της εφαρμογής ονομάστηκε backend και χρησιμοποιεί ένα προσαρμοσμένο Docker Image το οποίο θα αναλυθεί παρακάτω. Επιπλέον, γίνεται mount το socket του Docker διότι είναι απαραίτητο να λειτουργεί το Docker Engine μέσα στο container της εφαρμογής. Επίσης δηλώθηκαν κάποιες μεταβλητές περιβάλλοντος οι οποίες αναφέρουν εάν η εφαρμογή βρίσκεται σε τοπικό περιβάλλον και την έκδοση της βασικής Docker Image των Linux περιβαλλόντων. Επιπροσθέτως, υπάρχουν κάποια μυστικά, όπως τα credentials της βάσης δεδομένων και το κλειδί για την υπογραφή των authorization tokens τα οποία περνάνε στο πεδίο secrets για να μην είναι σε φανερό σημείο του διακομιστή και είναι ευάλωτα. Τέλος, έχει δηλωθεί ότι θα υπάρχουν 2 replicas για τον server της εφαρμογής το οποίο θα βοηθήσει στο zero-downtime deployment.

```
backend:
  image: ${BACKEND_IMAGE}
  volumes:
    - ../:/app/nephelelabs
    - /var/run/docker.sock:/var/run/docker.sock
  deploy:
    mode: replicated
    replicas: 2
    update_config:
      order: start-first
      failure_action: rollback
      delay: 5s
  secrets:
    - dsn
    - secret_key
  environment:
```

```
BASE_IMAGE_VERSION: 1.0.0
DEBUG: "false"
```

Το Docker Image που χρησιμοποιείται από τον orchestrator χρησιμοποιεί ως βάση το λειτουργικό σύστημα Alpine με την γλώσσα Go εγκατεστημένη με σκοπό να εγκαταστήσει τις απαραίτητες προϋποθέσεις της εφαρμογής και να την κάνει compile. Τέλος, κάνει αντιγραφή και επικόλληση το εκτελέσιμο αρχείο σε ένα άδαιο λειτουργικό σύστημα Alpine, έτσι ώστε το τελικό image να είναι μόνο λίγα mb.

```
#syntax=docker/dockerfile:1.2
FROM golang:1.16-alpine3.14 as builder

WORKDIR /app/nephelelabs

ENV GO111MODULE=on

COPY go.mod .
COPY go.sum .

RUN go mod download

COPY domain ./domain
COPY environments ./environments
COPY pkg ./pkg
COPY urlshortener ./urlshortener
COPY users ./users
COPY main.go .

RUN go build -o nephelelabs

FROM alpine:3.14

USER root

WORKDIR /root/

EXPOSE 8000

COPY --from=builder /app/nephelelabs/nephelelabs /root/nephelelabs

CMD [ "/root/nephelelabs" ]
```

Πελάτης

Η υπηρεσία του πελάτη της εφαρμογής ονομάστηκε frontend και χρησιμοποιεί ένα προσαρμοσμένο Docker Image το οποίο θα αναλυθεί παρακάτω. Επιπλέον, έχει δηλωθεί ότι θα υπάρχουν 2 replicas για τον frontend πελάτη το οποίο θα βοηθήσει στο zero-downtime deployment.

```
frontend:
  image: ${FRONTEND_IMAGE}
  volumes:
    - ../web:/usr/src/app
    - /usr/src/app/node_modules
    - /usr/src/app/.next
  deploy:
    mode: replicated
    replicas: 2
    update_config:
      order: start-first
      failure_action: rollback
      delay: 5s
```

Το Docker Image του πελάτη χρησιμοποιεί ως βάση την *fermium* η οποία είναι μια Long-term support έκδοση της Node. Έπειτα, γίνεται η εγκατάσταση των βασικών πακέτων που χρειάζεται η εφαρμογή, γίνεται *expose* η πόρτα 3000 και ξεκινάει η εφαρμογή.

```
FROM node:fermium-alpine3.11

RUN apk add --no-cache libc6-compat

WORKDIR /usr/src/app

COPY package.json yarn.lock ./

RUN yarn install --frozen-lockfile

COPY ./ ./

RUN yarn build && yarn install --production --ignore-scripts --prefer-offline

ENV NODE_ENV=production

EXPOSE 3000

ENV NEXT_TELEMETRY_DISABLED=1

CMD ["yarn", "start"]
```

Αντίστροφος διακομιστής μεσολάβησης

Η τελευταία υπηρεσία της εφαρμογής είναι ο αντίστροφος διακομιστής μεσολάβησης όπου χρησιμοποιήθηκε ως βάση το λειτουργικό σύστημα Alpine με τον διακομιστή *nginx* εγκατεστημένο. Επιπλέον, χρησιμοποιήθηκαν 2 replicas οι οποίες θα βοηθήσουν στο zero-downtime deployment. Επίσης, έγιναν mount τα paths όπου υπάρχουν οι ρυθμίσεις και τα SSL πιστοποιητικά. Τέλος, έγιναν *expose* τα ports

80 και 443.

```
nginx:
  image: nginx:1.21.3-alpine
  volumes:
    - ../nginx/nginx.conf:/etc/nginx/nginx.conf
    - /var/www/html:/var/www/html
    - /var/lib/letsencrypt:/var/lib/letsencrypt
    - /etc/letsencrypt:/etc/letsencrypt/
  deploy:
    mode: replicated
    replicas: 2
    update_config:
      order: start-first
      failure_action: rollback
      delay: 5s
  ports:
    - "80:80"
    - "443:443"
  depends_on:
    - frontend
    - backend
```

6.2.2 GitHub Actions

Όπως αναφέρθηκε προηγουμένως, κάθε φορά που περνάει κώδικας στο κλαδί `main` της εφαρμογής είναι καθαρός, τεσταρισμένος και έτοιμος να περάσει στην παραγωγή. Κάθε φορά που βγαίνει ένα νέο `Git tag` τότε ξεκινάει η εξής διαδικασία [Σχήμα 6.2] η οποία στέλνει τον κώδικα στην παραγωγή. Αρχικά, γίνεται `clone` το repository σε έναν απομακρυσμένο server του GitHub. Έπειτα, γίνεται εγκατάσταση του Docker. Μετά, γίνεται σύνδεση στο DockerHub το οποίο είναι το registry στο οποίο αποθηκεύονται τα Docker Images της εφαρμογής. Επιπλέον, τα Docker Images της εφαρμογής γίνονται `build` και `push` στο DockerHub. Τέλος, γίνεται μια σύνδεση με `SSH` στον server της εφαρμογής και τρέχει το `command` το οποίο εκτελεί τον orchestrator `Docker swarm`.

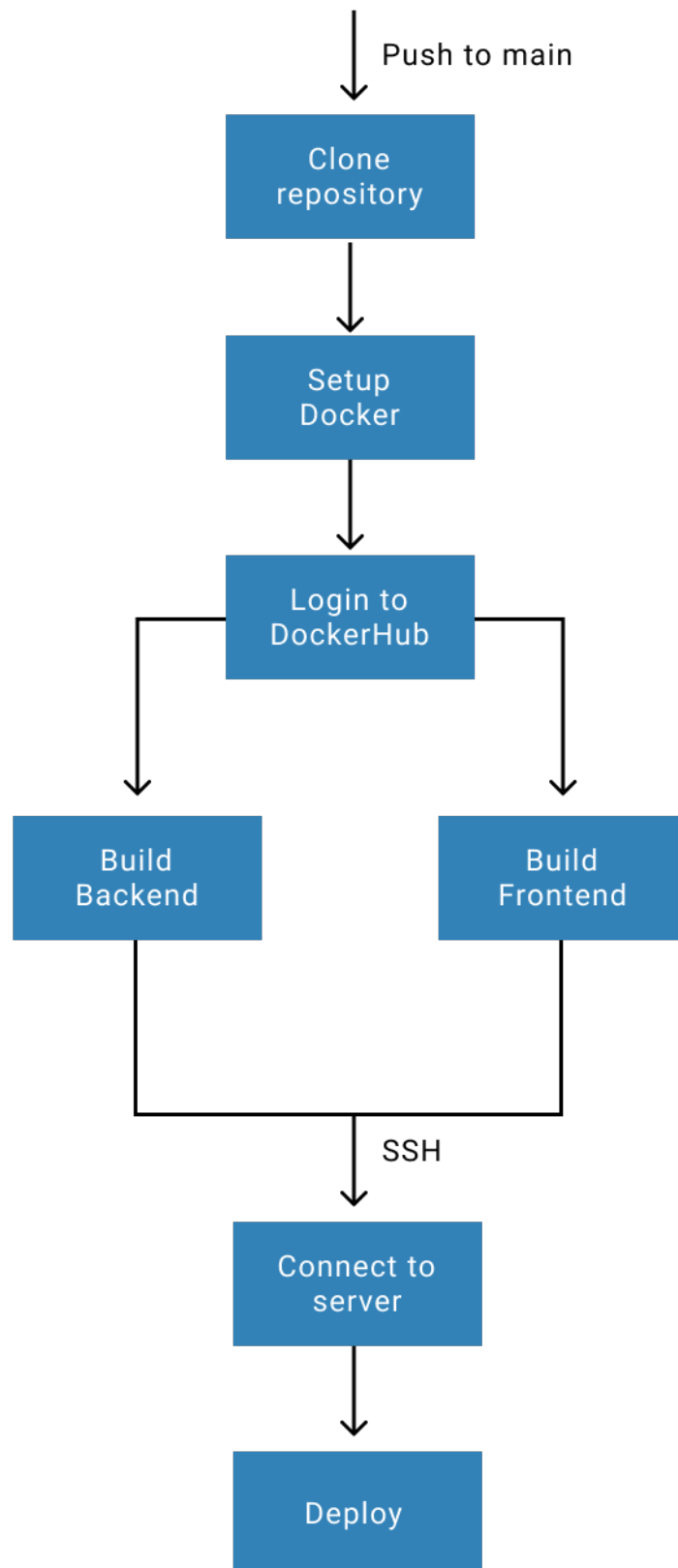
```
name: deploy

on:
  push:
    tags:
      - 'v*'

jobs:
  deploy:
    runs-on: ubuntu-20.04
    steps:
      - name: Check out code
        uses: actions/checkout@v2
```

- name: Set up QEMU
uses: docker/setup-qemu-action@v1
- name: Set up Docker Buildx
uses: docker/setup-buildx-action@v1
- name: Login to DockerHub
uses: docker/login-action@v1
with:
 username: \${{ secrets.DOCKERHUB_USERNAME }}
 password: \${{ secrets.DOCKERHUB_TOKEN }}
- name: Docker meta for backend
id: backend_meta
uses: docker/metadata-action@v3
with:
 images: nephelelabs/backend
 tags: |
 type=semver,pattern={{version}}
- name: Build and push backend
uses: docker/build-push-action@v2
with:
 context: .
 file: ./build/images/backend.prod.Dockerfile
 push: true
 tags: \${{ steps.backend_meta.outputs.tags }}
 secrets: |
 "dsn=\${{ secrets.DSN }}"
- name: Docker meta for frontend
id: frontend_meta
uses: docker/metadata-action@v3
with:
 images: nephelelabs/frontend
 tags: |
 type=semver,pattern={{version}}
- name: Build and push frontend
uses: docker/build-push-action@v2
with:
 context: web
 file: ./build/images/frontend.prod.Dockerfile
 push: true
 tags: \${{ steps.frontend_meta.outputs.tags }}
- name: Deploy
uses: appleboy/ssh-action@master
with:
 host: \${{ secrets.SSH_HOST }}

```
username: ${ secrets.SSH_USERNAME }}
key: ${ secrets.SSH_KEY }}
script: |
  cd nephelelabs
  git pull
  cd build
  export BACKEND_IMAGE=${ steps.backend_meta.outputs.tags }}
  export FRONTEND_IMAGE=${ steps.frontend_meta.outputs.tags }}
  docker stack deploy -c docker-compose.prod.yml nephelelabs
```



Σχήμα 6.2: Continuous integration pipeline για το deployment της εφαρμογής

Κεφάλαιο 7

Συμπεράσματα

Στο σημείο αυτό η εφαρμογή έχει φτάσει σε ένα τελικό στάδιο. Κάποιος χρήστης μπορεί να δημιουργήσει λογαριασμό και να συνδεθεί στην πλατφόρμα. Ένας καθηγητής έχει τη δυνατότητα να δημιουργήσει περιβάλλοντα και να τα διαμοιραστεί με τους φοιτητές του. Οι φοιτητές θα μπορούν να πραγματοποιήσουν τις διαδικασίες τις επιλογές τους μέσα στα περιβάλλοντα σε πραγματικό χρόνο χωρίς προβλήματα με ασφαλή τρόπο. Η εφαρμογή είναι στο στάδιο όπου πρέπει να γίνει η πρώτη δοκιμή. Το ιδανικό θα ήταν να γίνει αυτή δοκιμή είτε σε κάποιο εργαστήριο εκμάθησης Linux είτε σε κάποιο σεμινάριο με περιορισμένο αριθμό ατόμων. Ο λόγος είναι το ότι με την αύξηση των ατόμων, αυξάνεται ταυτόχρονα και το bandwidth που απαιτείται για την ομαλή εκτέλεση της εφαρμογής. Μετά από αυτό το σεμινάριο όπου θα ήταν δυνατόν να γίνουν μετρήσεις για τον τρόπο που χρησιμοποιείται η εφαρμογή, θα ήταν σε θέση να τρέξει με ακαθόριστο αριθμό χρηστών όπου ανάλογα με τον αριθμό των χρηστών θα γινόταν και η κατάλληλη αύξηση των χαρακτηριστικών του διακομιστή δυναμικά.

7.1 Μελλοντικές προοπτικές

Μετά την δοκιμή της και την διόρθωση των σφαλμάτων που ενδεχομένως να έχει και μετά από την ανατροφοδότηση των χρηστών θα μπορούσαν να προστεθούν επιπλέον λειτουργίες οι οποίες θα τους εξυπηρετούν. Τέλος, θα μπορούσαν να δημιουργηθούν νέοι πελάτες οι οποίοι θα ήταν ή για χρήση μέσω tablet ή για χρήση απευθείας μέσω τερματικού σε περίπτωση που κάποιοι χρήστες διευκολύνονται να χρησιμοποιούν την εφαρμογή με αυτόν τον τρόπο.

Βιβλιογραφία

- [1] Linux, "Linux" <https://www.linux.com/what-is-linux/> [Online; Accessed 21-05-2022]
- [2] The UNIX System, "UNIX SYSTEM" <https://unix.org/> [Online; Accessed 21-05-2022]
- [3] Debian, "The Universal Operating System" <https://debian.org/> [Online; Accessed 21-05-2022]
- [4] Canonical, "Ubuntu 20.04" <https://ubuntu.com> [Online; Accessed 10-Nov-2021]
- [5] Linux Mint, "Linux Mint" <https://www.linuxmint.com/> [Online; Accessed 10-Nov-2021]
- [6] Manjaro, "Enjoy the simplicity" <https://manjaro.org/> [Online; Accessed 10-Nov-2021]
- [7] Arch Linux, "Arch Linux" <https://www.archlinux.org/> [Online; Accessed 10-Nov-2021]
- [8] Play with docker, "Play with docker" <http://play-with-docker.com/> [Online; Accessed 10-Nov-2021]
- [9] Docker inc., "Docker Engine" <https://docs.docker.com/engine/> [Online; Accessed 10-Nov-2021]
- [10] CS50 Sandbox, "CS50 Sandbox" <https://sandbox.cs50.io/> [Online; Accessed 10-Nov-2021]
- [11] CS50, "CS50" <https://cs50.harvard.edu/> [Online; Accessed 10-Nov-2021]
- [12] NGINX, "Advanced Load Balancer, Web Server, and Reverse Proxy" <https://nginx.com/> [Online; Accessed 10-Nov-2021]
- [13] Let's encrypt, "Let's encrypt" <https://letsencrypt.org/> [Online; Accessed 10-Nov-2021]
- [14] Oracle, "MySQL" <https://mysql.com/> [Online; Accessed 10-Nov-2021]
- [15] Oracle, "Oracle" <https://oracle.com/> [Online; Accessed 10-Nov-2021]
- [16] Docker inc., "What is a Container?" <https://www.docker.com/resources/what-container/> [Online; Accessed 10-Nov-2021]

- [17] Docker inc., "Docker images" <https://docs.docker.com/engine/reference/commandline/images/> [Online; Accessed 10-Nov-2021]
- [18] Docker inc., "Overview of Docker Compose" <https://docs.docker.com/compose/> [Online; Accessed 10-Nov-2021]
- [19] Nodejs, "Fermium LTS" <https://github.com/nodejs/docker-node/blob/4e8fe34165d79044d7ea909021ccc0de3e3b4c6d/14/alpine3.14/Dockerfile> [Online; Accessed 10-Nov-2021]
- [20] Nodejs, "Node" <https://nodejs.org/en/> [Online; Accessed 10-Nov-2021]
- [21] Alpine Linux, "Alpine Linux" <https://www.alpinelinux.org/> [Online; Accessed 10-Nov-2021]
- [22] Google, "Go" <https://golang.org/> [Online; Accessed 10-Nov-2021]
- [23] Air, "Live reload for Go apps" <https://github.com/cosmtrek/air> [Online; Accessed 10-Nov-2021]
- [24] Swagger, "API Documentation and Design Tools for Teams" <https://swagger.io/> [Online; Accessed 10-Nov-2021]
- [25] TypeScript, "JavaScript With Syntax For Types." <https://www.typescriptlang.org/> [Online; Accessed 10-Nov-2021]
- [26] Reactjs, "A JavaScript library for building user interfaces" <https://reactjs.org/> [Online; Accessed 10-Nov-2021]
- [27] Nextjs, "The React Framework" <https://nextjs.org/> [Online; Accessed 10-Nov-2021]
- [28] Sapper, "The next small thing in web development" <https://sapper.svelte.dev/> [Online; Accessed 10-Nov-2021]
- [29] Svelte, "Cybernetically enhanced web apps" <https://svelte.dev/> [Online; Accessed 10-Nov-2021]
- [30] Tailwind CSS, "Rapidly build modern websites without ever leaving your HTML." <https://tailwindcss.com/> [Online; Accessed 10-Nov-2021]
- [31] Bash, "GNU Project - Free Software Foundation" <https://www.gnu.org/software/bash/> [Online; Accessed 10-Nov-2021]
- [32] Martin, R., 2018. Clean Architecture. Boston Columbus Indianapolis New York San Francisco Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo: Prentice Hall.
- [33] Auth0 "JWT" <https://jwt.io/> [Online; Accessed 11-Nov-2021]
- [34] Docker "Swarm" <https://docs.docker.com/engine/swarm/> [Online; Accessed 11-Nov-2021]

- [35] Zagalsky, Alexey, et al. "The emergence of github as a collaborative platform for education." Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing. 2015.
- [36] Consoli, Domenico. "A Collaborative Platform to Support the Enterprise 2.0 in Active Interactions with Customers." *Informatica Economica* 16.3 (2012).
- [37] Dillon, Tharam, Chen Wu, and Elizabeth Chang. "Cloud computing: issues and challenges." 2010 24th IEEE international conference on advanced information networking and applications. Ieee, 2010.
- [38] Wang, Lizhe, et al. "Cloud computing: a perspective study." *New generation computing* 28.2 (2010): 137-146.