ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Χρήση μεθοδολογιών και εργαλείων devops στην ανάπτυξη και την παραγωγική λειτουργία web εφαρμογών

Πτυχιακή εργασία

Παναγιώτης Μπέλλιας

[ΠΗΓΗ: https://s27389.pcdn.co/wp-content/uploads/2019/05/ultimate-guide-devops-e1558342120973-931x440.jpg.optimal.jpg]

Αθήνα, 2022

HAROKOPIO UNIVERSITY
SCHOOL OF DIGITAL TECHNOLOGY
DEPARTMENT OF INFORMATICS AND TELEMATICS

Use of devops methodologies and tools in development and production environment of web applications

Bachelor thesis

Panagiotis Bellias

[SOURCE: https://s27389.pcdn.co/wp-content/uploads/2019/05/ultimate-guide-devops-e1558342120973-931x440.jpg.optimal.jpg]

Athens, 2022

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Τριμελής Εξεταστική Επιτροπή

Τσαδήμας Ανάργυρος (Επιβλέπων)
Ε.ΔΙ.Π., Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο

Κουσιουρής Γεώργιος
Δ.Ε.Π., Επίκουρος Καθηγητής,
Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο

Δίου Χρήστος
Δ.Ε.Π., Επίκουρος Καθηγητής,
Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο

Ο Παναγιώτης Μπέλλιας

δηλώνω υπεύθυνα ότι:

1) Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

2) Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

3) Όπου υφίστανται δικαιώματα άλλων δημιουργών έχουν διασφαλιστεί όλες οι αναγκαίες άδειες χρήσης ενώ το αντίστοιχο υλικό είναι ευδιάκριτο στην υποβληθείσα εργασία.

Η παρούσα πτυχιακή εργασία είναι αφιερωμένη στην οικογένειά μου, αλλά κυρίως στη γιαγιά μου Χριστίνα και στον παππού μου Γεώργιο, τους οποίους αγαπούσα πολύ και έφυγαν από τη ζωή κατά τη διάρκεια των σπουδών μου. Θα ήθελαν πολύ να προλάβουν να με δουν να ορκίζομαι ως νέος πτυχιούχος.

Μην αφήσεις ποτέ να σβήσει
η φλόγα της ελπίδας μέσα σου.
Χωρίς ελπίδα δε ζει το όνειρο…

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα Πτυχιακή Εργασία εκπονήθηκε στα πλαίσια του προπτυχιακού προγράμματος σπουδών του Τμήματος Πληροφορικής και Τηλεματικής, του Χαροκοπείου Πανεπιστημίου, υπό την επίβλεψη του κ. Τσαδήμα Ανάργυρου.

Αρχικά, θα ήθελα να ευχαριστήσω τους γονείς μου για τη συνεχή υποστήριξή τους καθ' όλη τη διάρκεια της φοίτησης προκειμένου να πετύχω τον απώτερο στόχο, αυτόν της αποφοίτησης και της απόκτησης των κατάλληλων γνώσεων. Η συμπαράσταση και η εμπιστοσύνη που έδειξαν στο πρόσωπό μου αποτελεί καθοριστικό παράγοντα αυτής της επιτυχίας.

Την ειλικρινή μου ευγνωμοσύνη θα ήθελα να δείξω στους φίλους και συμφοιτητές μου όπως και το υπόλοιπο προσωπικό του πανεπιστημίου που αποτέλεσαν και εκείνοι ένα επιπρόσθετο κίνητρο με τις παροτρύνσεις και συμβουλές τους σε όλη τη διάρκεια αυτών των χρόνων γεγονός το οποίο εκτιμώ ιδιαιτέρως.

Τέλος, θα ήθελα να ευχαριστήσω βαθύτατα τον κ. Τσαδήμα Ανάργυρο, μέλος Ε.ΔΙ.Π. του Τμήματος,για την έμπνευση που μου έδωσε για αυτήν την εργασία μέσα από τη διδασκαλία του, τη δυνατότητα που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα στο χώρο της Πληροφορικής και την στήριξη και τις ουσιώδεις συμβουλές, κατευθύνσεις που μου παρείχε κατά την εκπόνηση της. Με βοήθησε να εμπλουτίσω το γνωστικό πεδίο στο αντικείμενο της Πληροφορικής και κυρίως να με εξοικειώσει με νέες τεχνολογίες και μεθόδους, αναπτύσσοντας μεταξύ μας άριστη συνεργασία. Επίσης, θα ήθελα να ευχαριστήσω τα μέλη της εξεταστικής επιτροπής, κ. Κουσιουρή Γεώργιο, Επίκουρο Καθηγητή στο γνωστικό αντικείμενο Τεχνολογίες και Απόδοση Υπηρεσιοστρεφών Εφαρμογών και Υποδομών, μέλος ΔΕΠ και τον κ. Δίου Χρήστο, Επίκουρο Καθηγητή στο γνωστικό αντικείμενο Τεχνητή Νοημοσύνη και Μηχανική Μάθηση, μέλος ΔΕΠ, για τη πολύτιμη βοήθεια τους στην περάτωση των σπουδών μου. Πληροφορίες για τους κ. Κουσιουρή και κ. Δίου στη σελίδα [professors-info].

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Περίληψη

Ο σκοπός της παρούσας πτυχιακής εργασίας είναι η ανάπτυξη, υλοποίηση και παραμετροποίηση ενός διαδικτυακού συστήματος διαχείρισης αιτήσεων συστατικών επιστολών που θα διευκολύνει φοιτητές, καθηγητές αλλά και εξωτερικούς φορείς όπως πανεπιστήμια ή εταιρείες (που χρειάζονται συστατική επιστολή προκειμένου να δεχτούν ή να προσλάβουν ένα φοιτητή ή απόφοιτο) να αυτοματοποιήσουν διάφορες διαδικασίες. Θα εξοικειωθούμε με συναφείς τεχνολογίες πληροφορικής σχετικά με την σχεδίαση, ανάπτυξη, αυτοματοποίηση, εγκατάσταση και συντήρηση λογισμικού διαθέσιμου στο web.

Το αντικείμενο της εργασίας είναι η εγκατάσταση και παραμετροποίηση του συστήματος προκειμένου να είναι διαθέσιμο στο διαδίκτυο αλλά και να μπορεί να βελτιώνεται εύκολα με αυτοματοποιημένο τρόπο. Οι διαδικασίες για την επίτευξη των παραπάνω ονομάζονται συχνά development operations (DevOps).

Η μεθοδολογία που ακολουθήθηκε είναι η παρακάτω. Ξεκινάμε με τη σχεδίαση του λογισμικού και τις απαιτήσεις των δυνητικών χρηστών. Συνεχίζουμε με την επιλογή των κατάλληλων προγραμματιστικών γλωσσών και των αντίστοιχων frameworks (έτοιμες προγραμματιστικές βιβλιοθήκες προς επαναχρησιμοποίηση) και αναλαμβάνουμε την υλοποίηση της εφαρμογής σύμφωνα με τις απαιτήσεις και τη σχεδίαση που προηγήθηκε.

Στην πορεία ερευνήθηκε ο τρόπος του dockerization του συστήματος αλλά και ο τρόπος αυτοματοποίησης της εγκατάστασης αυτού. Εξετάστηκαν οι εναλλακτικές τεχνολογίες docker για virtualization και containerization, όπως το Github Container Registry. Το σημαντικότερο λογισμικό που χρησιμοποιήθηκε στα πλαίσια της DevOps διαδικασίας είναι ο Κυβερνήτης (Kubernetes), το οποίο διαχειρίζεται containers σαν αντικείμενα λογισμικού συνδεδεμένα και με άλλα έτοιμα λογισμικά που υπάρχουν ή βάσεις δεδομένων. Το επόμενο κομμάτι της διαδικασίας είναι η συνεχής ενσωμάτωση, ανάπτυξη και συντήρηση της εφαρμογής χρησιμοποιώντας CI/CD servers.

Τέλος, κάνουμε μια αναφορά και δοκιμή σε κάποια εργαλεία διάγνωσης προβλημάτων αλλά και επίβλεψης λογισμικού (monitoring - logging tools). Για την εκπόνηση της παρούσας εργασίας και την εκτέλεση της περιγραφείσας διαδικασίας χρησιμοποιήθηκαν εργαλεία version control όπως github, και ιδεατές μηχανές (virtual machines - VMs) που ανήκουν στις υποδομές του τμήματος Πληροφορικής και Τηλεματικής του Χαροκοπείου Πανεπιστημίου Αθηνών, αλλά και σε cloud παρόχους (βλ. Microsoft Azure και Google Cloud Platform).

Λέξεις κλειδιά: [Διαχείριση συστατικών επιστολών, Λογισμικό, Αυτοματοποίηση, DevOps, Kubernetes]

Abstract

The purpose of this thesis is the development, implementation and configuration of a web reference letter request management system that will facilitate students, faculty and external bodies such as universities or companies (that need a reference letter to accept or hire a student or graduate) automating various procedures. We will be familiar with IT technologies related to the design, development, automation, installation and maintenance of software available on the web.

The object of the thesis is the installation and configuration of the system in order to be available on the internet but also to be able to be easily improved in an automated way. The processes to achieve the above are often called development operations (DevOps).

The methodology followed is as follows. We start with the design of the software and the requirements of the potential users. We continue with the selection of the appropriate programming languages and the corresponding frameworks (ready programming libraries for reuse) and we undertake the implementation of the application according to the requirements and the previous design.

In the course, the way of dockerization of the system was investigated, as well as the way of automation of installation of its. Alternative docker technologies for virtualization and containerization, such as the Github Container Registry, were examined. The most important software used in the DevOps process is the Kubernetes, which manages containers as software objects linked to other existing software or databases. The next part of the process is the continuous integration, development and maintenance of the application using CI / CD servers.

Finally, we make a report and test on some problems of diagnostics and software monitoring (monitoring - logging tools). For the elaboration of the present thesis and the execution of the described process, version control tools such as github, and virtual machines (VMs) belonging to the infrastructure of the Department of Informatics and Telematics of Harokopio University of Athens, but also to cloud providers were used (see Microsoft Azure and Google Cloud Platform).

Keywords: [Github, Software, Docker, Jenkins, Kubernetes]

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ/ΑΚΡΩΝΥΜΙΑ

DevOps	Development Operations
Dev	Development
IT	Information Technology
SDLC	Systems Development Life Cycle
KPI	Key Performance Indicator
CI/CD	Continuous integration / Continuous Delivery - Deployment
UML	Unified Modeling Language
VM	Virtual Machine (Ιδεατή - Εικονική Μηχανή)
fqdn	Fully Qualified Domain Name
ASGI	Asynchronous Server Gateway interface
JS	JavaScript
HTML	HyperText Markup Language
npm	Node Package Manager
VSCode	Visual Studio Code
IDE	Integrated Development Environment
PostgreSQL	Postgre Structured Query Language
URL	Uniform Resource Locator
HTTP/HTTPS	Hypertext Transfer Protocol / Hypertext Transfer Protocol Secure
IP	Internet Protocol
TCP	Transmission Control Protocol
YAML	Yet Another Markup Language
SSH	Secure Shell (protocol)
CWI	Centrum Wiskunde & Informatica
FastCGI	Fast Common Gateway Interface
CPU	Central Processing Unit
INI	Initialize/Initialization/Configuration file
CRUD	Create, Read, Update and Delete
REST API	REpresentational State Transfer Application Programming Interface
DNS	Domain Name Server

ΚΕΦ. 1: Εισαγωγή

Ένας από τους κύριους στόχους του DevOps είναι να γεφυρώσει το χάσμα μεταξύ Business, Dev και IT με στόχο να μετατρέψει το SDLC σε στρατηγικό πόρο. Ωστόσο, για να επιτευχθεί αυτός ο στόχος, είναι επιτακτική ανάγκη να επιλεγούν τα σωστά εργαλεία. Τα σωστά εργαλεία πραγματοποιούν όλα τα πλεονεκτήματα του DevOps βελτιώνοντας την παραγωγικότητα, αυξάνοντας τη διαφάνεια, μειώνοντας το χρόνο για τα προϊόντα και τις υπηρεσίες να βγουν στην αγορά, βελτιώνοντας τους KPI, όπως μετρήσεις απόδοσης κώδικα και αξιοπιστία, ενώ ουσιαστικά επιταχύνεται η διαδικασία δημιουργίας αξίας. Η ολοκληρωμένη λίστα των εργαλείων DevOps είναι ένα βήμα προς την κατανόηση αυτού του τοπίου. Ωστόσο, πρέπει να έχετε κατά νου ότι η απόκτηση λειτουργικής ωριμότητας δεν είναι κάτι που μπορεί να επιτευχθεί εν μία νυκτί, ακόμη και μετά την εφαρμογή των εργαλείων DevOps. Θα επιτρέψει να παρέχεται καλύτερο λογισμικό και να διασφαλίζεται βελτιωμένη διαφάνεια σε ολόκληρη την αλυσίδα αξίας. Αυτά τα εργαλεία DevOps από μόνα τους δεν οδηγούν σε μετασχηματισμό, είναι απλώς ένα κρίσιμο βήμα για να συμμορφώνεστε με τις οδηγίες Agile/DevOps, ώστε να μπορείτε να εστιάσετε στη βελτίωση της ταχύτητας και της ακρίβειας και να ενεργοποιηθεί η συνεργασία μεταξύ ομάδων.

Στόχος της πτυχιακής εργασίας είναι το ηλεκτρονικό σύστημα διαχείρισης συστατικών επιστολών να αξιοποιεί τη μέθοδο του DevOps και να διευκολύνει την καθημερινότητα στην επαγγελματική ζωή των φοιτητών και των καθηγητών αλλά και υπαλλήλων σε πανεπιστήμια και εταιρείες ώστε με τη βοήθεια των προγραμματιστών να αυτοματοποιούν διαδικασίες, να πετυχαίνουν συνεχή ενσωμάτωση και προώθηση σε περιβάλλον παραγωγής (CD) του συστήματος μετά από κάθε αλλαγή προσαρμοσμένη στις ανάγκες τους, να έχουν δυνατότητα παραμετροποίησης, επεκτασιμότητας του συστήματος σε διαφορετικά περιβάλλοντα εκτέλεσης χωρίς τα προβλήματα της αναχρονιστικής μεθόδου προγραμματισμού.

Όσον αφορά τη δομή της παρούσας πτυχιακής εργασίας έχουμε τα παρακάτω κεφάλαια.

Στο κεφάλαιο 2 γίνεται παρουσίαση της ιδέας, της σχεδίασης και των χαρακτηριστικών του συστήματος για να καταλάβουμε πώς θα βοηθήσει την ακαδημαϊκή και όχι μόνο κοινότητα. Η παρουσίαση της σχεδίασης και της αρχιτεκτονικής του συστήματος γίνεται με τη χρήση UML διαγραμμάτων.

Στο κεφάλαιο 3 περιγράφονται οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη του συγκεκριμένου διαδικτυακού συστήματος και τις συνιστώσες (components) αυτού.

Στο κεφάλαιο 4 βλέπουμε κάποια επιπλέον components, τον τρόπο παραμετροποίησης της συστήματος ώστε να μπορεί να εγκατασταθεί σε servers και να γίνεται public available και παρουσιάζονται υποστηριζόμενες λειτουργίες. Επιπλέον, κάνουμε αναφορά σε τρόπους και εργαλεία αυτοματοποιημένης δοκιμής του λογισμικού και εφαρμογές πάνω στο σύστημα που περιγράφεται.

Στο κεφάλαιο 5 παρουσιάζεται ο τρόπος του containerization και virtualization εφαρμογών με χρήση του docker και του GitHub Container Registry αλλά και πως γίνεται η ανίχνευση τρωτών σημείων μέσα στα containers του συστήματος.

Στο κεφάλαιο 6 εξηγείται πώς μπορεί να αυτοματοποιηθεί η εγκατάσταση κώδικα λογισμικού σε VMs ή servers χρησιμοποιώντας την Ansible, τα υπάρχοντα docker services, και αντίστοιχο κώδικα (ansible galaxy roles).

Στο κεφάλαιο 7 βλέπουμε πώς μπορεί να χρησιμοποιηθεί ένας CI/CD server (jenkins) για το συνεχή έλεγχο και τη διανομή του συστήματος στους τελικούς χρήστες με κάθε αλλαγή που μπορεί να γίνει στον κώδικα.

Στο κεφάλαιο 8 γίνεται χρήση του λογισμικού kubernetes και επεξήγηση της έννοιας του helm. Επίσης αναλύουμε τον τρόπο που ρυθμίζουμε fqdn (fully qualified domain name) για την εφαρμογή μας.

Στο κεφάλαιο 9 παρουσιάζονται τα σενάρια χρήσης του συστήματος.

Στο κεφάλαιο 10 βγάζουμε χρήσιμα συμπεράσματα που ενδέχεται να χρησιμοποιηθούν σε μελλοντικές επεκτάσεις του θέματος.

Τέλος, γίνεται παράθεση της βιβλιογραφίας για όλη τη βοήθεια που προσέφεραν στην εκπόνηση της παρούσας πτυχιακής εργασίας και παραρτημάτων με τον κώδικα που βοήθησε στην ανάπτυξη του συστήματος.

ΚΕΦ. 2: Ανάλυση & Σχεδίαση Διαδικτυακού Συστήματος

2.1 Εισαγωγή

Στο συγκεκριμένο κεφάλαιο θα παρουσιαστεί η ανάλυση των απαιτήσεων των χρηστών και των χαρακτηριστικών που θα θέλαμε να καλύπτει το σύστημα. Γίνεται αναφορά στους ρόλους της εφαρμογής και αναδεικνύεται η σημασία της χρήσης DevOps εργαλείων για την αρχιτεκτονική του συστήματος.

2.2 Ανάλυση Απαιτήσεων & Χαρακτηριστικά Συστήματος

Το σύστημα που θέλουμε να αναπτύξουμε και να παραμετροποιήσουμε διαχειρίζεται συστατικές επιστολές και υποστηρίζει τους εξής ρόλους: φοιτητής (student), καθηγητής (teacher) και διαχειριστής συστήματος (admin). Σχεδιάζεται για να εξυπηρετεί την ακαδημαϊκή κοινότητα και κάθε πανεπιστήμιο.

Ο φοιτητής πρέπει να μπορεί να κάνει αίτηση σε έναν από τους διαθέσιμους εγγεγραμμένους στο σύστημα καθηγητές για συστατική επιστολή για να αποσταλεί είτε σε κάποιον φορέα εργασίας (εταιρεία), είτε σε κάποιο πανεπιστήμιο για μεταπτυχιακό πρόγραμμα σπουδών.

Ο καθηγητής πρέπει:

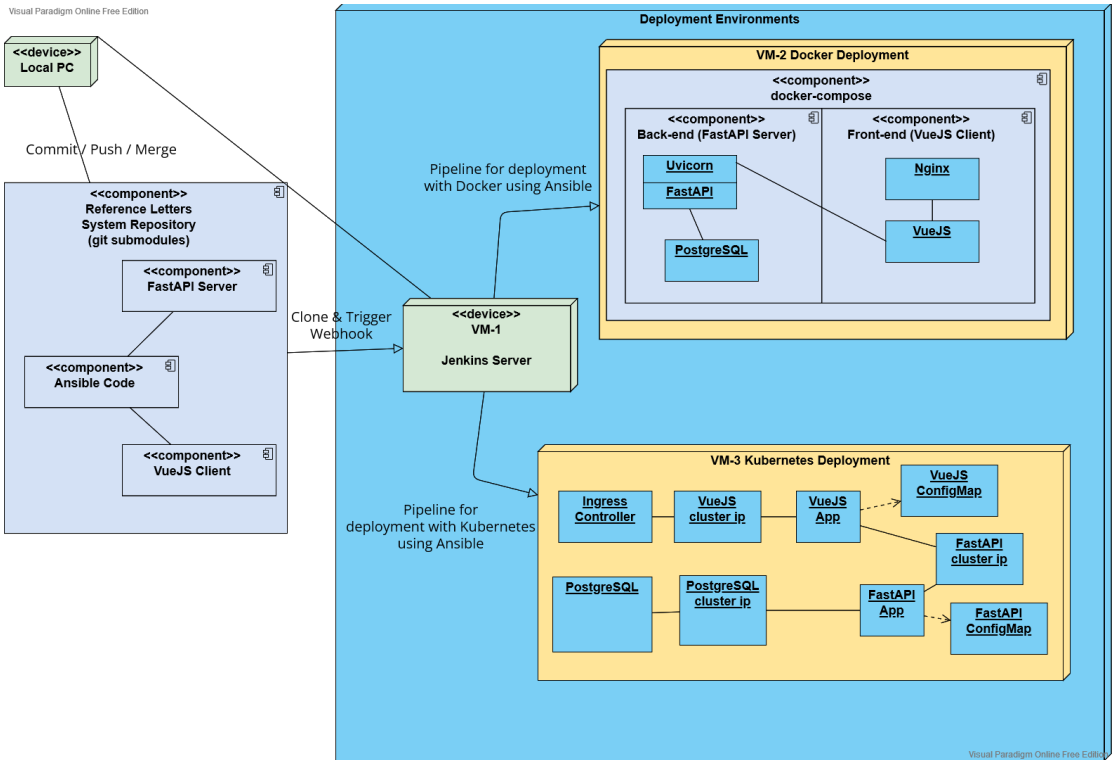
1) Να μπορεί να δει πληροφορίες για τις αιτήσεις που τον αφορούν και να αποφασίζει αν τις αποδέχεται ή αν τις απορρίπτει.
2) Εφόσον αποφάσισε να την αποδεχτεί, θα υποβάλλει το σχετικό κείμενο της επιστολής και θα αποστέλλεται ενημέρωση στον εξωτερικό φορέα ή πανεπιστήμιο.
3) Θα ενημερώνεται για νέες αιτήσεις που απευθύνονται σε αυτόν.

Στο σύστημα, θα υπάρχει και ο διαχειριστής ο οποίος θα έχει πρόσβαση σε όλα τα δεδομένα αιτήσεων και χρηστών. Θα μπορεί επίσης να δει στατιστικά σχετικά με τις αιτήσεις συστατικής επιστολής και της πορείας αυτών.

Κάποια από τα επιθυμητά χαρακτηριστικά που θα έχει το σύστημα είναι η γραφική απεικόνιση στατιστικών και η παροχή φόρμας επικοινωνίας. Ακόμη, θα θέλαμε:

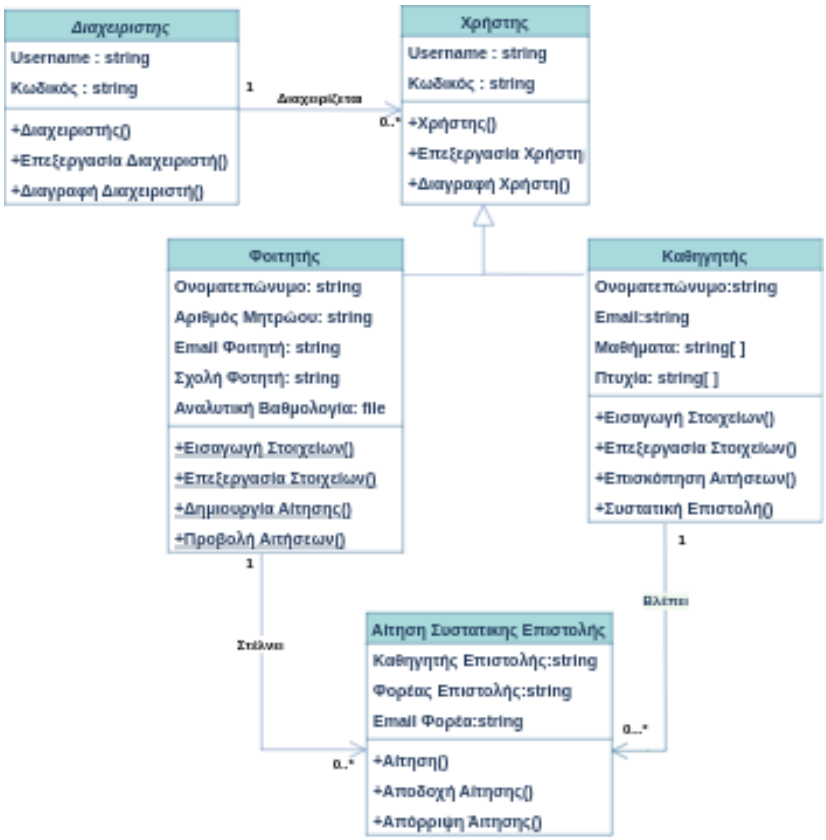
1) να παρέχεται γραμμή αναζήτησης για αιτήσεις και καθηγητές,
2) να πετύχουμε ανεκτικότητα σε λάθος δεδομένα εισόδου ή σε ελαττώματα άλλου λογισμικού που χρησιμοποιεί,
3) να μπορεί να χρησιμοποιηθεί με docker και kubernetes για τις ανάγκες της εργασίας,
4) εύκολη παραμετροποίηση τιμών του συστήματος (όπως σύνδεση με βάση, αποδεκτοί hosts κ.ά.), κάτι που θεωρείται αυτονόητο κατά τη διάρκεια ανάπτυξης μιας διαδικτυακής εφαρμογής.

2.3 Σχεδιαγράμματα



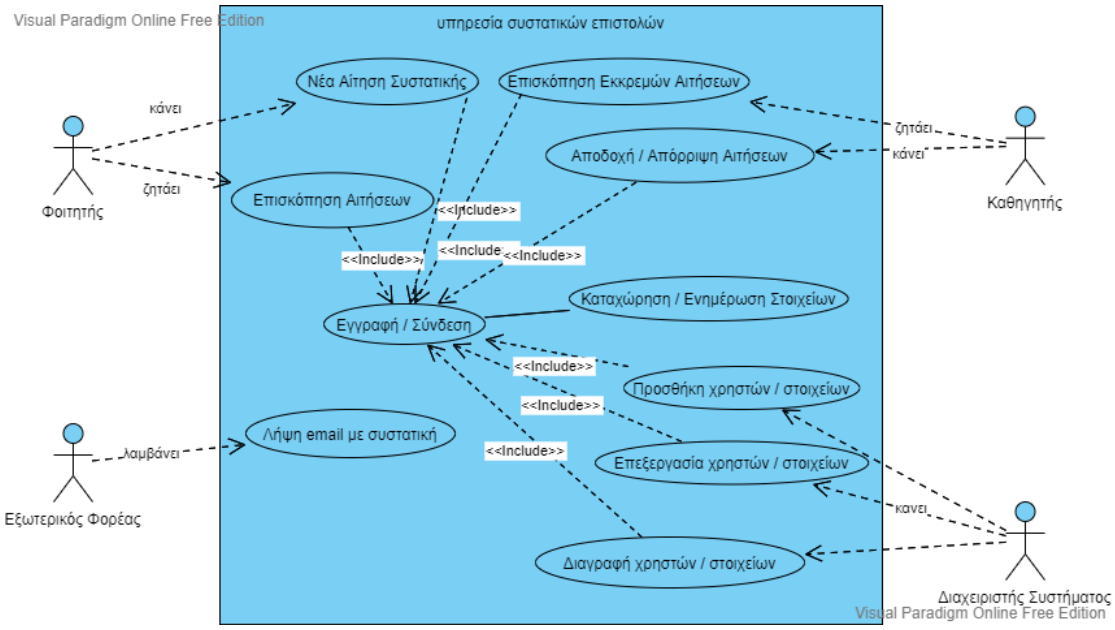
Σχήμα 1 Αρχιτεκτονική Συστήματος

Στο παρόν διάγραμμα αρχιτεκτονικής βλέπουμε όλους τους τρόπους όπου μπορεί να γίνει deploy το σύστημά μας. Παρατηρούμε δηλαδή τα εργαλεία devops, τις τεχνολογίες που χρησιμοποιούνται σε κάθε τρόπο deployment, τον CI/CD Jenkins server που στοχεύει στην αυτοματοποίηση της όλης διαδικασίας, ενδεικτικά τα VM (εικονικές μηχανές) όπου μπορεί να γίνει deploy το σύστημα και τα τρία αποθετήρια κώδικα (ansible code repository, backend fastapi repository, frontend vuejs repository) όπου περιέχεται ο κώδικας όλης της εργασίας. Στους servers - VMs και στον τοπικό προσωπικό υπολογιστή μας, όπου επιτυγχάνεται σύνδεση, επικοινωνία και deploy του συστήματος χρησιμοποιείται το πρωτόκολλο SSH με αντίστοιχα κλειδιά που δημιουργούνται. Έχοντας στόχο το deployment του συστήματος απο το VM που περιέχει τον Jenkins server στα υπόλοιπα και να δούμε την εφαρμογή μας τοπικά, χρησιμοποιούνται jenkins pipelines.



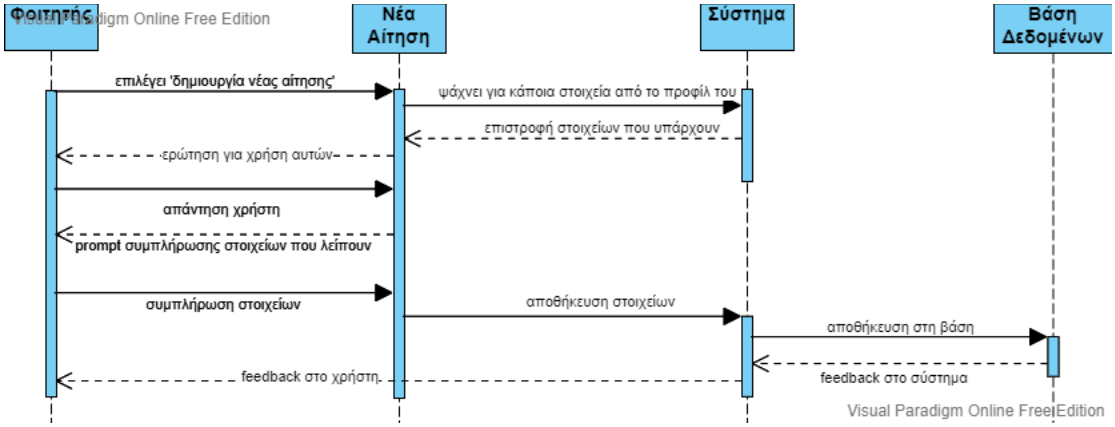
Σχήμα 2 Class Διάγραμμα

Ο διαχειριστής του συστήματος μπορεί να επεξεργαστεί τα δικά του στοιχεία ή των υπόλοιπων χρηστών. Οι χρήστες μπορούν να επεξεργαστούν τα δικά τους στοιχεία. Χρήστης μπορεί να είναι ένας καθηγητής ή ένας φοιτητής. Οι φοιτητές μπορούν να στέλνουν αιτήσεις ή να βλέπουν την εξέλιξή τους. Οι καθηγητές βλέπουν τις αιτήσεις των φοιτητών και στέλνει κείμενο συστατικής επιστολής εφόσον την αποδέχεται. Στην αίτηση αναγράφεται και το email του φορέα ώστε σε περίπτωση αποδοχής να σταλεί το κείμενο συστατικής επιστολής αυτόματα στον εξωτερικό φορέα στον οποίο απευθύνεται ο φοιτητής.



Σχήμα 3 Use Case Διάγραμμα

Ο φοιτητής κάνει νέα αίτηση συστατικής επιστολής και ζητάει επισκόπηση όλων των αιτήσεών του. Ο καθηγητής ζητάει επισκόπηση των εκκρεμών αιτήσεων, αποδοχή ή απόρριψή τους και ανεβάζουν συστατική επιστολή όταν αποδέχονται την αίτηση. Ο εξωτερικός φορέας λαμβάνει συστατικές επιστολές. Ο διαχειριστής μπορεί να προσθέσει, να τροποποιήσει ή να διαγράψει χρήστες και δεδομένα τους.



Σχήμα 4 Sequence Διάγραμμα

Ο φοιτητής επιλέγει τη λειτουργία της δημιουργίας νέας αίτησης από την αρχική σελίδα. Το σύστημα ψάχνει για καταχωρημένα στοιχεία στο σύστημα που μπορεί να χρησιμοποιήσει και επιστρέφει στο φοιτητή τα στοιχεία που υπάρχουν για να συμφωνήσει ο φοιτητής για τη χρήση αυτών. Για ό,τι στοιχείο λείπει (είτε δεν υπάρχει στο σύστημα, είτε είναι καινούριο) συμπληρώνει τα αντίστοιχα πεδία ο φοιτητής. Η νέα αίτηση αποθηκεύεται στη βάση και το σύστημα δίνει feedback στο φοιτητή για επιτυχή δημιουργία και αποθήκευση νέας αίτησης.

ΚΕΦ.3: Τεχνολογίες του Διαδικτυακού Συστήματος

3.1 Python - FastAPI Framework

Η Python είναι μία διερμηνευόμενη δυναμική γλώσσα προγραμματισμού υψηλού επιπέδου και υποστηρίζει το διαδραστικό και το αντικειμενοστραφές μοντέλο. Δημιουργήθηκε από τον Ολλανδό Γκίντο βαν Ρόσσουμ στο ερευνητικό κέντρο Centrum Wiskunde & Informatica (CWI) το 1989 και κυκλοφόρησε για πρώτη φορά το 1991.

Το FastAPI είναι τελευταία αρκετά δημοφιλές ανάμεσα σε άλλα web frameworks για backend development και έχει γραφτεί σε γλώσσα python. Ένα web framework είναι ένα σύνολο από στοιχεία που μας βοηθά να αναπτύξουμε ιστοσελίδες πιο γρήγορα και εύκολα. Το FastAPI προσφέρει διάφορα σημαντικά χαρακτηριστικά όπως είναι οι application servers (βλ. uvicorn), οι οποίοι διακομιστές διαβάζουν μηνύματα και αποκρίνονται με ιστοσελίδες. Γι' αυτό μας βοηθάει στην αποστολή περιεχομένου ώστε να εμφανιστεί η ιστοσελίδα. Βασίζεται στο μοντέλο του single-page application, δηλαδή έχουμε ένα προγραμματιστικό αρχείο όπου δηλώνουμε τα endpoints που θέλουμε να εξυπηρετούν τον client (πελάτη) που θα καλέσει τη FastAPI εφαρμογή μας. Φυσικά χάρη στις βασικές αρχές της python μπορούμε για μεγαλύτερη οργάνωση του κώδικα που συγγράφουμε να χωρίζουμε την πληροφορία σε περισσότερα αρχεία (είτε ανά κατηγορία endpoint, είτε επειδή θέλουμε να ξεχωρίσουμε πχ τον κώδικα που μας βοηθάει στη σύνδεση με μια βάση δεδομένων ή ένα third-party λογισμικό).

Για να έχουμε τη δυνατότητα δημιουργίας μιας FastAPI εφαρμογής είναι απαραίτητη προϋπόθεση να έχουμε εγκατεστημένη κάποια έκδοση της Python. Έπειτα, πρέπει να εκτελέσουμε κάποιες εντολές προκειμένου να γίνει η εγκατάσταση του FastAPI framework και ότι αυτό χρειάζεται:

```
virtualenv fvenv -p python3
```

Εικόνα 1 Δημιουργία Εικονικού Περιβάλλοντος

Το εικονικό περιβάλλον (virtual environment) είναι απαραίτητο επειδή δημιουργεί ένα απομονωμένο περιβάλλον, ένα φάκελο δηλαδή που περιέχει όλα τα απαραίτητα εκτελέσιμα που χρησιμοποιούν τα πακέτα της εφαρμογής.

```
source fvenv/bin/activate
```

Εικόνα 2 Ενεργοποίηση Εικονικού Περιβάλλοντος

Ύστερα πρέπει να εγκαταστήσουμε το FastAPI με την παρακάτω εντολή:

```
pip install fastapi
```

Εικόνα 3 Εγκατάσταση FastAPI

Ακόμη θα χρειαστούμε έναν ASGI server, για production όπως ο Uvicorn (ή εναλλακτικά ο Hypercorn).

```
pip install "uvicorn[standard]"
```

Εικόνα 4 Εγκατάσταση Uvicorn Application Server

Για να αρχικοποιήσουμε το project μας δημιουργούμε έναν φάκελο με ένα όνομα της προτίμησής μας (π.χ. ref_letters για να ταιριάζει με το θέμα του συστήματος που θα αναπτύξουμε) και μέσα εκεί ένα αρχείο με τίτλο main.py το οποίο αργότερα θα το καλέσουμε χρησιμοποιώντας τον uvicorn server. Έχουμε τον παρακάτω απλό κώδικα για δοκιμή τοπικά.

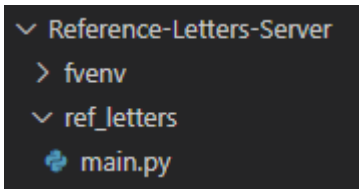
```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"greetings": "Welcome to FastAPI Python"}
```

Εικόνα 5 Ενδεικτικός Κώδικας FastAPI

Σε αυτή τη φάση η δομή του FastAPI project θα πρέπει να είναι έτσι:



Εικόνα 6 Δομή FastAPI Project

Όσον αφορά τη σύνδεση με βάση δεδομένων δημιουργούμε ένα αρχείο με τίτλο db.py όπου δηλώνουμε κάποια πράγματα που φαίνονται παρακάτω χρησιμοποιώντας το sqlalchemy dependency και επίσης τους πίνακες που θέλουμε αναλόγως των αναγκών της εφαρμογής.

Βλέπουμε να γίνονται αρχικοποιήσεις σε κάποια objects που σχετίζονται με σύνδεση σε βάσεις δεδομένων. Έχουμε και το engine του sqlalchemy όπου δημιουργεί τους πίνακες στη βάση εφόσον τρέξουμε για πρώτη φορά την εφαρμογή μας. Το DATABASE_URL είναι ενδεικτικό και μπορούμε να το περνάμε στον κώδικα παραμετρικά χρησιμοποιώντας το module dotenv. Παρακάτω βλέπετε τον κώδικα με τον οποίο μπορούμε να αντικαταστήσουμε τη γραμμή που δηλώνει το DATABASE_URL κάνοντας και κάποιες επιπλέον εισαγωγές βιβλιοθηκών.

```python
DATABASE_URL = os.getenv("DATABASE_URL", default="sqlite+aiosqlite:///./test.db")   # Declare database url
Base: DeclarativeMeta = declarative_base()
engine = create_async_engine(DATABASE_URL)
async_session_maker = sessionmaker(engine, class_=AsyncSession, expire_on_commit=False)

class Teacher(Base):
    __tablename__ = "teacher"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String)
    description = Column(String)
    user_username = Column(String, ForeignKey("user.username"))
    user = relationship("User")

class Student(Base):
    __tablename__ = "student"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String)
    school = Column(String)
    school_id = Column(String)
    grades_url = Column(String)
    user_username = Column(String, ForeignKey("user.username"))
    user = relationship("User")

class ReferenceLetterRequest(Base):
    __tablename__ = "reference_letter_request"
    id = Column(Integer, primary_key=True, index=True)
    teacher_id = Column(Integer, ForeignKey("teacher.id"))
    student_id = Column(Integer, ForeignKey("student.id"))
    carrier_name = Column(String)
    carrier_email = Column(String)
    status = Column(String)
    text = Column(String)

class User(SQLAlchemyBaseUserTableUUID, Base):
    username = Column(String, unique=True)
    email = Column(String)
    full_name = Column(String)
    student = Column(Boolean)
    teacher = Column(Boolean)
    admin = Column(Boolean)
    disabled = Column(Boolean)

async def create_db_and_tables():
    async with engine.begin() as conn:
        await conn.run_sync(Base.metadata.create_all)

async def get_async_session() -> AsyncGenerator[AsyncSession, None]:
    async with async_session_maker() as session:
        yield session

async def get_user_db(session: AsyncSession = Depends(get_async_session)):
    yield SQLAlchemyUserDatabase(session, User)
```

Εικόνα 7 Σύνδεση Project Με Βάσεις Δεδομένων

```python
import os
from dotenv import load_dotenv
load_dotenv(verbose=True)

DATABASE_URL = os.getenv("DATABASE_URL", default="sqlite+aiosqlite:///./test.db")   # Declare database url
```

Εικόνα 8 Παραμετροποίηση Τιμών με dotenv

Αυτό επιβάλλεται να χρησιμοποιείται γενικά στις εφαρμογές για απόκρυψη ευαίσθητων τιμών από ένα αποθετήριο κώδικα που πιθανώς έχουμε την εφαρμογή μας. Για να δουλέψει αυτό δημιουργούμε επίσης μέσα στο φάκελο ref_letters, δύο αρχεία. Ένα .env (όπου δεν το ανεβάζουμε σε repository - μπορούμε να το προσθέσουμε πχ στο .gitignore) και ένα .env.example όπου έχουμε ενδεικτικές τιμές και ο χρήστης πρέπει να το αντιγράψει σε .env και να αλλάξει τις τιμές με βάση τις δικές του ανάγκες. Επίσης, κάτι που παραλείψαμε προηγουμένως είναι πως πρέπει να συλλέξουμε σε ένα αρχείο οτιδήποτε χρησιμοποιούμε ως dependency στο virtual environment. Αυτό γίνεται όταν είμαστε στον root folder του project με την παρακάτω εντολή.

```
pip freeze -l > requirements.txt
```

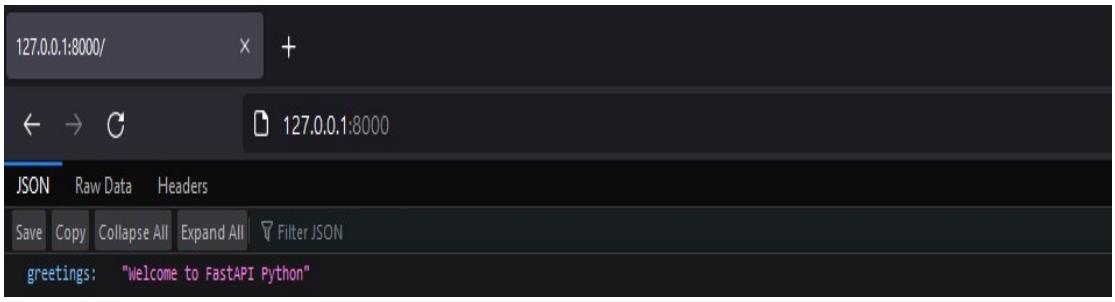
Εικόνα 9 Συλλογή Των Dependencies Του Project

Εν τέλει, προκειμένου να τρέξουμε τοπικά την εφαρμογή μας μέσα από τον uvicorn application server τρέξαμε:

```
uvicorn ref_letters.main:app --reload
```

Εικόνα 10 Εκτέλεση FastAPI Εφαρμογής Τοπικά

Για να δούμε την εφαρμογή μας στον browser μας, πληκτρολογούμε στη γραμμή αναζήτησης: http://127.0.0.1:8000/



Εικόνα 11 Επιτυχής Εκτέλεση Της Εφαρμογής

3.2 Javascript - VueJS Framework

Η JavaScript (JS) είναι διερμηνευμένη γλώσσα προγραμματισμού για ηλεκτρονικούς υπολογιστές. Αρχικά αποτέλεσε μέρος της υλοποίησης των φυλλομετρητών Ιστού, ώστε τα σενάρια από την πλευρά του πελάτη (client-side scripts) να μπορούν να επικοινωνούν με τον χρήστη, να ανταλλάσσουν δεδομένα ασύγχρονα και να αλλάζουν δυναμικά το περιεχόμενο του εγγράφου που εμφανίζεται. Δημιουργήθηκε αρχικά από τον Brendan Eich της εταιρείας Netscape με την επωνυμία Mocha. Αργότερα, η Mocha μετονομάστηκε σε LiveScript, και τελικά σε JavaScript, κυρίως επειδή η ανάπτυξή της επηρεάστηκε περισσότερο από τη γλώσσα προγραμματισμού Java. LiveScript ήταν το επίσημο όνομα της γλώσσας όταν για πρώτη φορά κυκλοφόρησε στην αγορά σε βήτα (beta) εκδόσεις με το πρόγραμμα περιήγησης στο Web, Netscape Navigator εκδοχή 2.0 τον Σεπτέμβριο του 1995. Η LiveScript μετονομάστηκε σε JavaScript σε μια κοινή ανακοίνωση με την εταιρεία Sun Microsystems στις 4 Δεκεμβρίου, 1995, όταν επεκτάθηκε στην έκδοση του προγράμματος περιήγησης στο Web, Netscape εκδοχή 2.0B3. [1]

Το Vue.js διαθέτει μια σταδιακά προσαρμόσιμη αρχιτεκτονική που εστιάζει στη δηλωτική απόδοση και τη σύνθεση στοιχείων. Η βασική βιβλιοθήκη εστιάζει μόνο στο επίπεδο προβολής. Οι προηγμένες δυνατότητες που απαιτούνται για πολύπλοκες εφαρμογές όπως η δρομολόγηση, η διαχείριση κατάστασης και η κατασκευή εργαλείων προσφέρονται μέσω επίσημα συντηρούμενων βιβλιοθηκών και πακέτων υποστήριξης. Το Vue.js επιτρέπει την επέκταση του HTML με χαρακτηριστικά HTML που ονομάζονται οδηγίες. Οι οδηγίες προσφέρουν λειτουργικότητα σε εφαρμογές HTML και έρχονται είτε ως ενσωματωμένες είτε ως οδηγίες που ορίζονται από το χρήστη. Δημιουργήθηκε από τον Evan You αφού εργάστηκε για την Google χρησιμοποιώντας το AngularJS σε πολλά έργα. Αργότερα συνόψισε τη διαδικασία σκέψης του: "Σκέφτηκα, τι θα γινόταν αν μπορούσα απλώς να εξαγάγω το μέρος που μου άρεσε πολύ στο Angular και να δημιουργήσω κάτι πολύ ελαφρύ." Το Vue κυκλοφόρησε για πρώτη φορά τον επόμενο Φεβρουάριο, το 2014. [2]

Για να έχουμε τη δυνατότητα δημιουργίας μιας VueJS εφαρμογής είναι απαραίτητη προϋπόθεση να έχουμε εγκατεστημένη κάποια έκδοση του NodeJS και του npm. Έπειτα, πρέπει να εκτελέσουμε κάποιες εντολές προκειμένου να γίνει η εγκατάσταση του VueJS framework και ότι αυτό χρειάζεται:

```
npm install -g vue-cli
```

Εικόνα 12 Εγκατάσταση VueJS CLI

Το VueJS CLI βελτιώνει την εμπειρία του προγραμματιστή μιας VueJS εφαρμογής και αυτοματοποιεί αρκετά κοινότοπα πράγματα που θέλει να κάνει.

```
vue init webpack your-vue-app-name
```

Εικόνα 13 Δημιουργία Νέου VueJS Project

Δημιουργούμε χρησιμοποιώντας το package webpack ένα νέο VueJS project με όνομα που ορίζουμε εμείς.

```
? Project name your-vue-app-name
? Project description A Vue.js project
? Author Jordan Hudgens <jordan@devcamp.com>
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Set up unit tests No
? Setup e2e tests with Nightwatch? No
? Should we run `npm install` for you after the project has been created? (recommended) npm
```

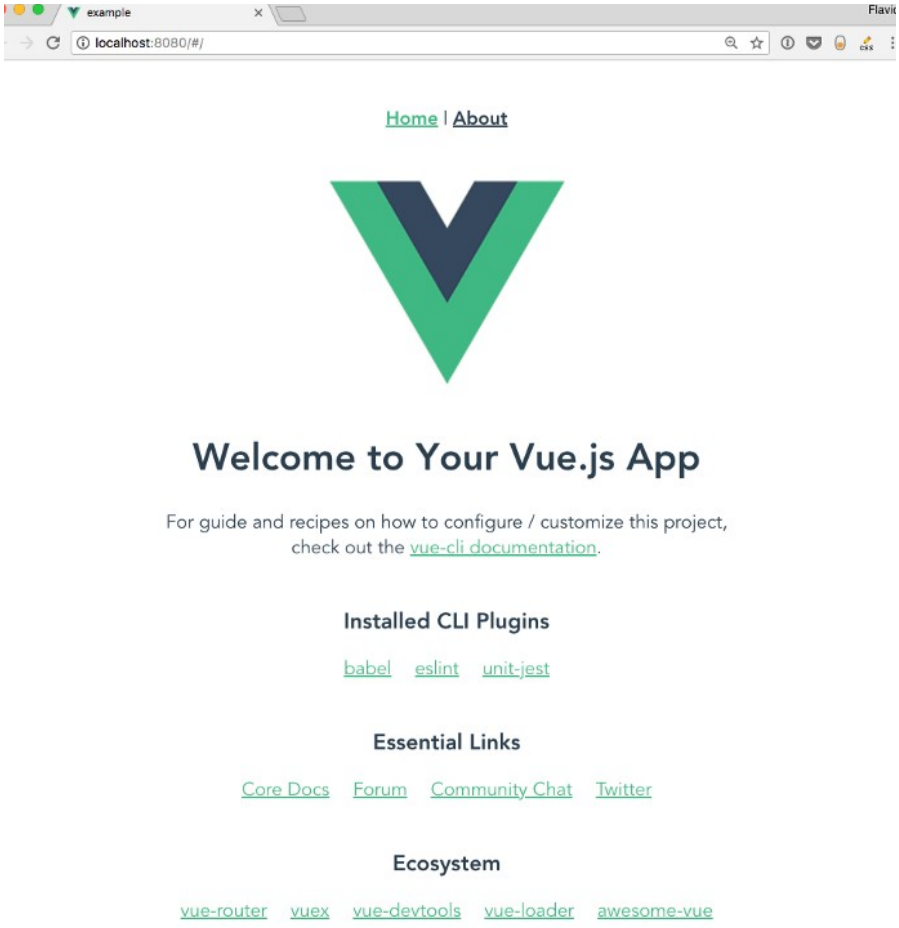
Εικόνα 14 Ρύθμιση / Αρχικοποίηση Project

Έπειτα, υπάρχουν διάφορες ερωτήσεις που πρέπει να απαντήσουμε για να ρυθμίσουμε και να αρχικοποιήσουμε το project μας. Παραπάνω βλέπετε τις ερωτήσεις που είναι πολύ πιθανό να σας κάνει και ενδεικτικές απαντήσεις που μπορεί να διαφέρουν από project σε project αναλόγως των αναγκών.

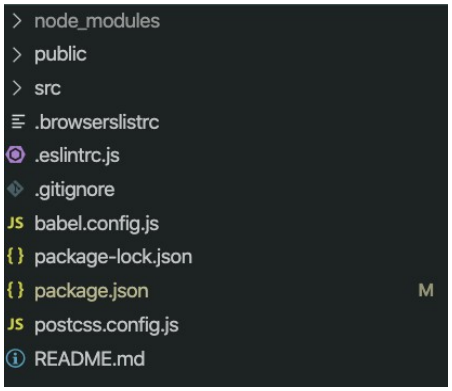
```
npm run dev
```

Εικόνα 15 Έναρξη Development Server

Πρώτα, εφόσον βλέπετε ότι υπάρχουν errors στο project χρειάζεται να εκτελέσετε την εντολή `npm install` και έπειτα να ξεκινήσετε με την παραπάνω εντολή τον development server για να δείτε την προεπιλεγμένη αρχική σελίδα της εφαρμογής. Μπορείτε να τη δείτε εδώ εφόσον ξεκίνησε ο development server: http://localhost:8080/



Εικόνα 16 Επιτυχής Εκτέλεση VueJS Εφαρμογής (Πηγή: https://www.freecodecamp.org/news/learn-how-to-use-the-vue-js-cli-8349fb23a566/)



Εικόνα 17 Ενδεικτική Δομή VueJS Project

Βλέπουμε μια ενδεικτική δομή ενός project που έχει δημιουργηθεί ακολουθώντας τα παραπάνω βήματα που προαναφέραμε.

```
data(){
  return {
    rl_requests: [],
    errors: [],
    backend: process.env.VUE_APP_BACKEND_URL
  }
},
```

Εικόνα 18 Παραμετροποίηση Με dotenv

Όπως και στην backend εφαρμογή θα χρησιμοποιήσουμε τη δυνατότητα να αποκρύπτουμε ευαίσθητες πληροφορίες από τον κώδικα, όπως, για παράδειγμα, είναι το URL στο οποίο είναι διαθέσιμο το REST API που θέλουμε να καλέσουμε. Στην εικόνα φαίνεται πως μπορούμε να αναθέσουμε στη μεταβλητή backend μια αντίστοιχη τιμή διαβάζοντας ένα .env αρχείο. Παρόμοια με πριν, ο χρήστης / προγραμματιστής που θα το τρέξει πρέπει να αντιγράψει το .env.example που θα βρει στο root path του project σε .env και να αλλάξει τις τιμές ανάλογα με τις δικές του ανάγκες και το περιβάλλον στο οποίο θα τρέξει η εφαρμογή. Το παράδειγμα της εικόνας είναι κομμάτι κώδικα που υπάρχει στο javascript κομμάτι του vue component και αρχικοποιεί όλα τα δεδομένα που θα χειριστούμε ή θα χρησιμοποιήσουμε σαν αρχικές τιμές.

Στην παρούσα τεχνολογία η παραμετροποίηση με dotenv δε χρειάζεται κάποια εγκατάσταση κώδικα τρίτου, καθώς το παρέχει αυτόματα. Με αυτόν τον τρόπο επίσης μπορούμε να καλέσουμε ένα υπάρχον δοκιμαστικό REST API για να δούμε πως όλα δουλεύουν με σωστό τρόπο, όπως είναι το [JSONPlaceholder].

3.3 VSCode IDE

Το Visual Studio Code, που συνήθως αναφέρεται και ως VS Code, είναι ένα πρόγραμμα επεξεργασίας πηγαίου κώδικα που δημιουργήθηκε από τη Microsoft για Windows, Linux και macOS. Οι δυνατότητες περιλαμβάνουν υποστήριξη για εντοπισμό σφαλμάτων, επισήμανση σύνταξης, έξυπνη συμπλήρωση κώδικα, αποσπάσματα, ανακατασκευή κώδικα και ενσωματωμένο Git. Οι χρήστες μπορούν να αλλάξουν το θέμα, τις συντομεύσεις πληκτρολογίου, τις προτιμήσεις και να εγκαταστήσουν επεκτάσεις που προσθέτουν επιπλέον λειτουργίες.

Στο Stack Overflow 2021 Developer Survey, το Visual Studio Code κατατάχθηκε ως το πιο δημοφιλές εργαλείο περιβάλλοντος προγραμματιστών, με το 70% των 82.000 ερωτηθέντων να αναφέρει ότι το χρησιμοποιεί.

Το Visual Studio Code είναι ένα πρόγραμμα επεξεργασίας πηγαίου κώδικα που μπορεί να χρησιμοποιηθεί με μια ποικιλία γλωσσών προγραμματισμού, όπως Java, JavaScript, Go, Node.js, Python, C++ και Fortran. Βασίζεται στο Electron framework, που χρησιμοποιείται για την ανάπτυξη εφαρμογών Web Node.js που εκτελούνται στη μηχανή διάταξης Blink. Το Visual Studio Code χρησιμοποιεί το ίδιο στοιχείο επεξεργασίας (με την κωδική ονομασία "Monaco") που χρησιμοποιείται στο Azure DevOps (παλαιότερα ονομαζόταν Visual Studio Online και Visual Studio Team Services). [3]

3.4 SQLite - DB Browser for SQLite, PostgreSQL - PgAdmin

Το SQLite είναι μια σχεσιακή βάση δεδομένων συμβατή με SQL. Σε αντίθεση με άλλα συστήματα που βασίζονται σε SQL όπως MySQL και PostgreSQL, το SQLite δεν χρησιμοποιεί αρχιτεκτονική πελάτη-διακομιστή (client-server). Ολόκληρο το λογισμικό περιέχεται σε μια βιβλιοθήκη C, η οποία είναι ενσωματωμένη σε εφαρμογές.

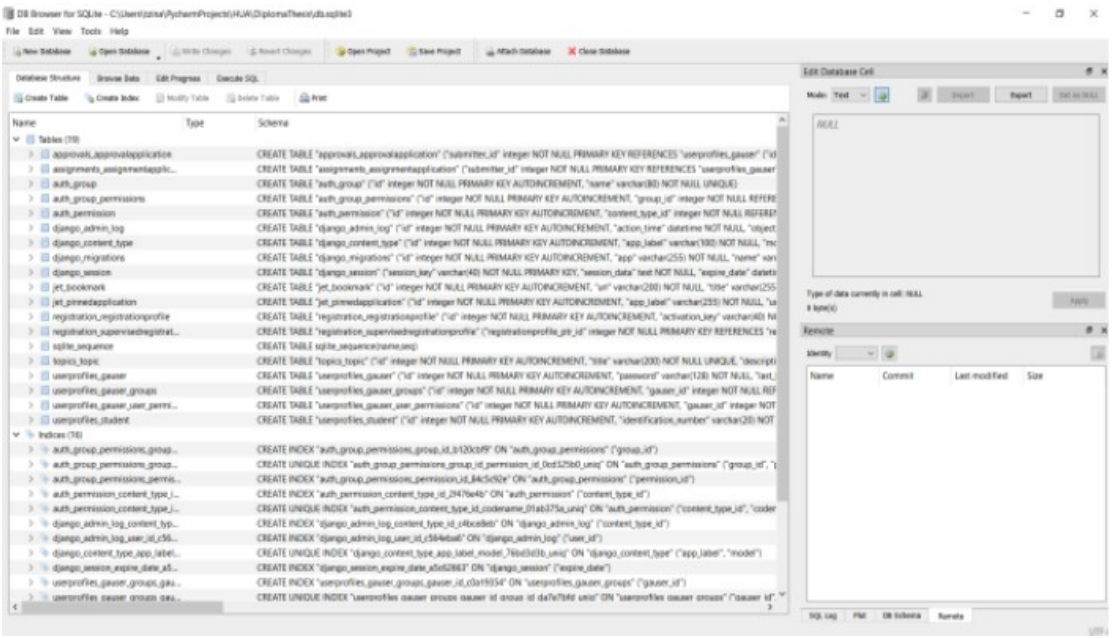
Το SQLite επικεντρώνεται στην παροχή μιας ισχυρής βάσης δεδομένων συμβατής με SQL χωρίς εξαρτήσεις. Όπως υποδηλώνει το όνομα, είναι ένα ελαφρύ λογισμικό που μπορεί να τρέξει σχεδόν σε οτιδήποτε υποστηρίζει C και μόνιμο χώρο αποθήκευσης αρχείων. Οι δεσμεύσεις είναι διαθέσιμες για τις πιο δημοφιλείς γλώσσες προγραμματισμού υψηλού επιπέδου. Δεδομένου ότι οι βάσεις δεδομένων SQLite είναι κανονικά αρχεία, είναι εξαιρετικά φορητές και δημιουργούνται εύκολα αντίγραφα ασφαλείας. Επίσης, η έλλειψη ενός στοιχείου διακομιστή καθιστά το SQLite πολύ πιο εύκολο να ρυθμιστεί, ακόμα κι αν δεν βρισκόμαστε σε ένα πλήρες περιβάλλον ανάπτυξης (production).

Το SQLite λειτουργεί καλύτερα όταν θέλουμε να συνδυάσουμε τις ισχυρές δυνατότητες αναζήτησης και αποθήκευσης του SQL με την ευκολία χρήσης της συμβατικής πρόσβασης στο σύστημα αρχείων. Σε αυτά τα σενάρια, προσφέρει καλύτερη απόδοση και ανθεκτικότητα σε σύγκριση με τις κανονικές λειτουργίες ανάγνωσης και εγγραφής.

Το σύστημα λειτουργεί καλά σε περιβάλλοντα όπου οι τελικοί χρήστες δεν πρέπει ποτέ να γνωρίζουν την ύπαρξη της βάσης δεδομένων και δεν απαιτεί συντήρηση ή διαχείριση, καθιστώντας το ιδανικό για εφαρμογές για κινητά και συσκευές IoT.

Συνοπτικά, η απλότητα, η φορητότητα και η αξιοπιστία τον καθιστούν το προτιμώμενο σύστημα αποθήκευσης για σύγχρονα λειτουργικά συστήματα και ενσωματωμένες πλατφόρμες. Δεν καταναλώνει σχεδόν καθόλου πόρους, είναι παραμετροποιήσιμο και εύκολο στη χρήση για προγραμματιστές λειτουργώντας ‘αόρατα’ για τους τελικούς χρήστες.

Το DB Browser for SQLite είναι μία διαδραστική, ανοικτού κώδικα τεχνολογία που χρησιμοποιείται για να σχεδιάσει, να δημιουργήσει και να επεξεργαστεί αρχεία βάσεων δεδομένων συμβατά με SQLite. Αυτή η τεχνολογία επιτρέπει στους χρήστες να αλληλεπιδρούν με δεδομένα σε ένα περιβάλλον βασισμένο στα φύλλα.



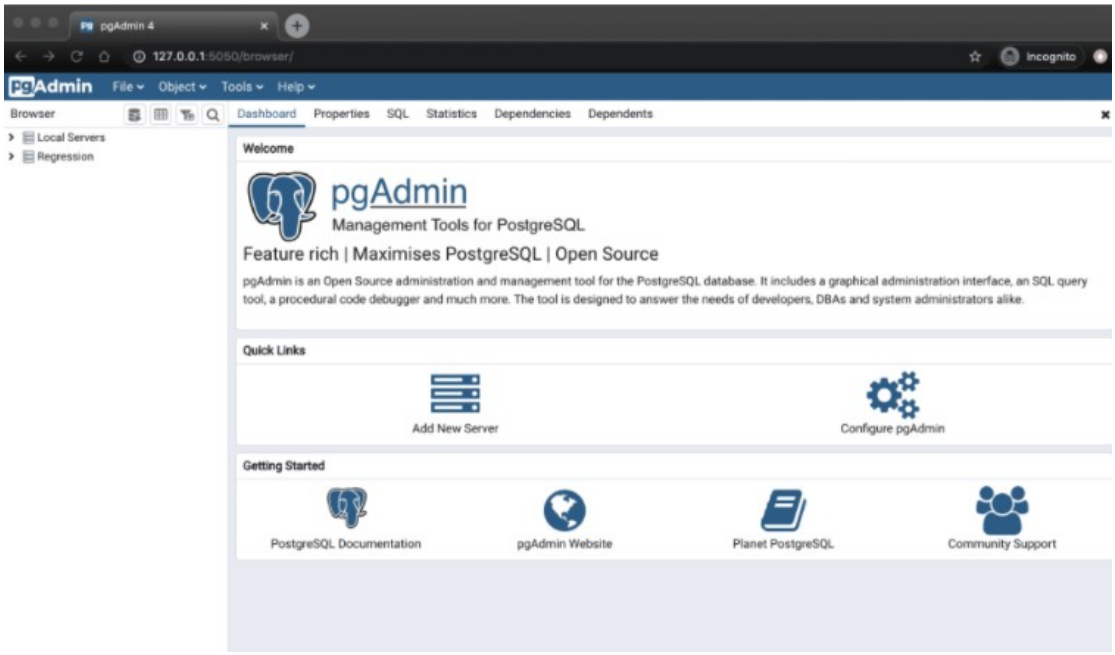
Εικόνα 19 DB Browser for SQLite

Όσον αφορά την PostgreSQL, είναι μια σχεσιακή βάση δεδομένων ανοικτού κώδικα με πολλές δυνατότητες. Η ανάπτυξη της διαρκεί ήδη πάνω από δύο δεκαετίες και βασίζεται σε μια αποδεδειγμένα καλή αρχιτεκτονική η οποία έχει δημιουργήσει μια ισχυρή αντίληψη των χρηστών της γύρω από την αξιοπιστία, την ακεραιότητα δεδομένων και την ορθή λειτουργία.

Η PostgreSQL δεν ακολουθεί την αρχιτεκτονική client – server. Αντιθέτως, η βάση δεδομένων PostgreSQL είναι ενσωματωμένη στην εφαρμογή που έχει πρόσβαση στη βάση δεδομένων. Η εφαρμογή αλληλεπιδρά με τη βάση δεδομένων που διαβάζει και γράφει απευθείας από τα αρχεία βάσης δεδομένων που είναι αποθηκευμένα στο δίσκο.

Η PostgreSQL δεν έχει η ίδια κάποιο γραφικό περιβάλλον διεπαφής με τον χρήστη. Για την αλληλεπίδραση με την βάση δεδομένων χρησιμοποιείται ένα πρόγραμμα χρήστη (client) το PgAdmin που παρέχεται και εγκαθίσταται μαζί με την PostGreSQL. Το PgAdmin και έχει τα παρακάτω χαρακτηριστικά :

1. Είναι ένα ολοκληρωμένο σύστημα σχεδιασμού και διαχείρισης βάσεων δεδομένων και αποτελεί το μέσο αλληλεπίδρασης της βάσης δεδομένων με το χρήστη.
2. Είναι γραμμένο σε C++ και είναι δυνατό να χρησιμοποιηθεί σε περιβάλλοντα Linux, FreeBSD, Solaris, Mac OS X και Windows για τη διαχείριση της PostgreSQL.
3. Προσφέρει ένα απλό γραφικό περιβάλλον για την ανάπτυξη πολύπλοκων βάσεων δεδομένων μέσω της διατύπωσης ερωτημάτων σε SQL, με στόχο την απλούστευση των διαδικασιών για το χρήστη
4. Είναι ελεύθερο λογισμικό και δεν απαιτεί επιπλέον προγράμματα για την επικοινωνία με τον διακομιστή της βάσης δεδομένων.



Εικόνα 20 PgAdmin

3.5 Uvicorn, Nginx

Όταν ένα διαδικτυακό σύστημα αποτελείται από διαφορετικές εφαρμογές και τρέχει στην παραγωγή (production) συνήθως χρειάζονται δύο βασικά σημεία για να εμφανιστεί. Το πρώτο είναι ένας web server (π.χ. nginx) όπου θα συνδεθεί με την frontend εφαρμογή και το δεύτερο είναι ένας asgi web application server (π.χ. uvicorn) όπου εξυπηρετεί την backend εφαρμογή και την κάνει διαθέσιμη στη frontend. Και βέβαια χρειάζονται οι εφαρμογές μας και οι εξαρτήσεις τις οποίες έχουν.

Το Uvicorn είναι μια υλοποίηση ASGI web server για την Python. Μέχρι πρόσφατα, η Python δεν είχε μια ελάχιστη διεπαφή διακομιστή/εφαρμογής χαμηλού επιπέδου για ασύγχρονα frameworks. Η προδιαγραφή ASGI καλύπτει αυτό το κενό και σημαίνει ότι τώρα είμαστε σε θέση να αρχίσουμε να χτίζουμε ένα κοινό σύνολο εργαλείων που μπορούν να χρησιμοποιηθούν σε όλα τα ασύγχρονα frameworks. Το Uvicorn υποστηρίζει επί του παρόντος HTTP/1.1 και WebSockets.

Ο web server αποδέχεται αιτήματα, φροντίζει για μία γενική λογική στο domain και χειρίζεται συνδέσεις βασισμένες στα http/https.

Όσον αφορά τους web servers, συναντάμε συχνά τον όρο web hosting (ή διακομιστή/web server). Σε αυτήν την εποχή της πληροφορίας βρίσκουμε πολλές παραλλαγές υπηρεσιών web hosting ειδικά από τις Ηνωμένες Πολιτείες (ΗΠΑ), όπου η τεχνολογία του Διαδικτύου συνεχίζει να αναπτύσσεται. Για να εκτελέσουμε ένα σύστημα web hosting, χρειαζόμαστε μια συσκευή web server. Μεταξύ των πολλών λογισμικών που χρησιμοποιούν οι προγραμματιστές, υπάρχει και το διάσημο Nginx. Είναι λογισμικό web server που κυκλοφόρησε ως ανοιχτή πηγή. Εκτός από γνωστό ως web server, το Nginx είναι επίσης γνωστό ως αντίστροφος διακομιστής μεσολάβησης, HTTP cache, και εξισορρόπησης φορτίου. Πολλές εταιρείες βασισμένες σε τεχνολογία υπολογιστών μεγάλης κλίμακας σε όλο τον κόσμο επιλέγουν να χρησιμοποιήσουν web server. Μεταξύ των ονομάτων αυτών είναι οι Google, Twitter, Facebook.

Το Nginx λειτουργεί ως web server, δηλαδή προσομοιώνει μια συσκευή υπολογιστή ως μηχανή παροχής υπηρεσιών σελίδας στο διαδίκτυο. Τα χαρακτηριστικά του Nginx τα οποία μπορούν να χρησιμοποιηθούν για την υποστήριξη της απόδοσης του διαδικτύου είναι:

Υποστήριξη IPv6 (δικτυακό πρότυπο). Το IPv6 έχει διαφορετική δομή διευθύνσεων από το IPv4, το οποίο έχει μήκος 128-bit και είναι γραμμένο με 8 δεκαεξαδικων ομάδες.

Εξισορρόπηση φορτίου. Η εξισορρόπηση φορτίου είναι μια τεχνική για την κατανομή φορτίων κίνησης σε δύο ή περισσότερες γραμμές σύνδεσης με ισορροπημένο τρόπο.

Υποστήριξη FastCGI με διαδικασίες cache. Το FastCGI (Fast Common Gateway Interface) είναι ένα πρωτόκολλο δυαδικό για σύνδεση προγραμμάτων (για παράδειγμα web browser) με web server διαδραστικά.

Websockets. Το WebSocket είναι ένα πρωτόκολλο επικοινωνίας υπολογιστή, η λειτουργία του είναι να παρέχει σύνδεση μέσων επικοινωνίας πλήρως αμφίδρομη μέσω μιας σύνδεσης TCP (Transmission Control Protocol).

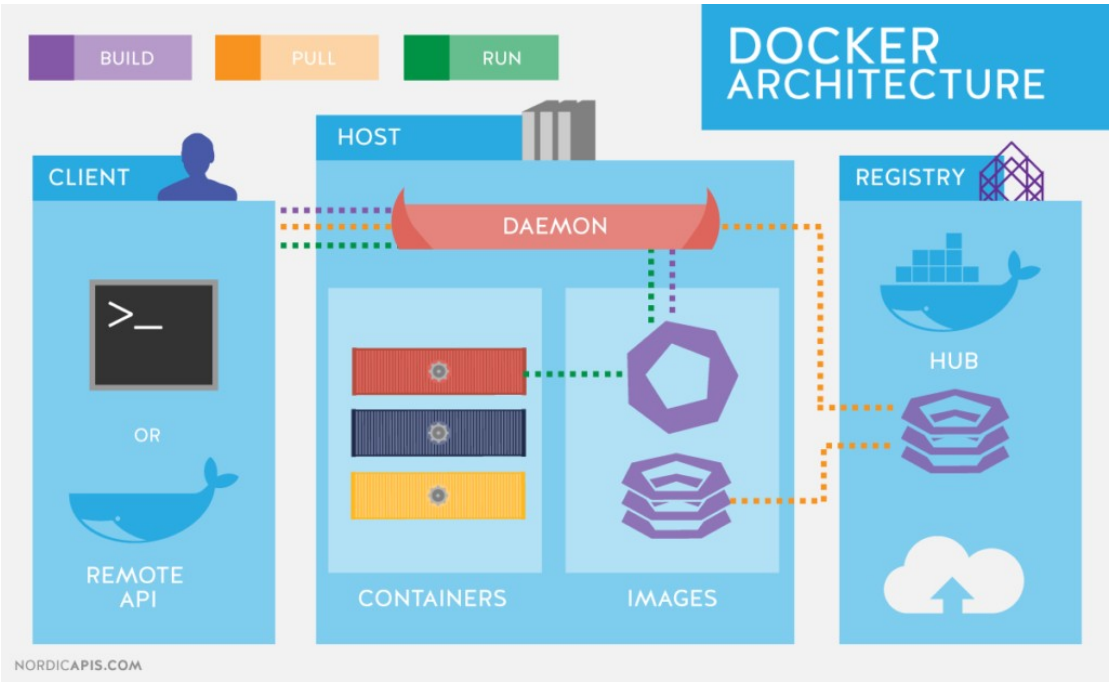
Διαχειρίζεται στατικά αρχεία, αρχεία ευρετηρίου και αυτόματη ευρετηρίαση.

3.6 Docker, Ansible, Jenkins, Kubernetes

Η ανάπτυξη εφαρμογών, σήμερα, απαιτεί πολλά περισσότερα από κώδικα. Οι πολλαπλές γλώσσες, τα πλαίσια, οι αρχιτεκτονικές και οι διεπαφές μεταξύ εργαλείων για κάθε στάδιο του κύκλου ζωής δημιουργούν τεράστια πολυπλοκότητα. Το Docker απλοποιεί και επιταχύνει τη ροή αυτή, ενώ δίνει στους προγραμματιστές την ελευθερία να καινοτομούν με την επιλογή εργαλείων, στοίβας εφαρμογών και περιβαλλόντων ανάπτυξης για κάθε έργο.

Το Docker, λοιπόν, είναι ένα λογισμικό ανοικτού κώδικα που αυτοματοποιεί την ανάπτυξη εφαρμογών λογισμικού μέσα σε containers (βλ. παρακάτω) παρέχοντας ένα πρόσθετο επίπεδο αφαίρεσης και αυτοματοποίησης σε επίπεδο λειτουργικού συστήματος στο Linux.

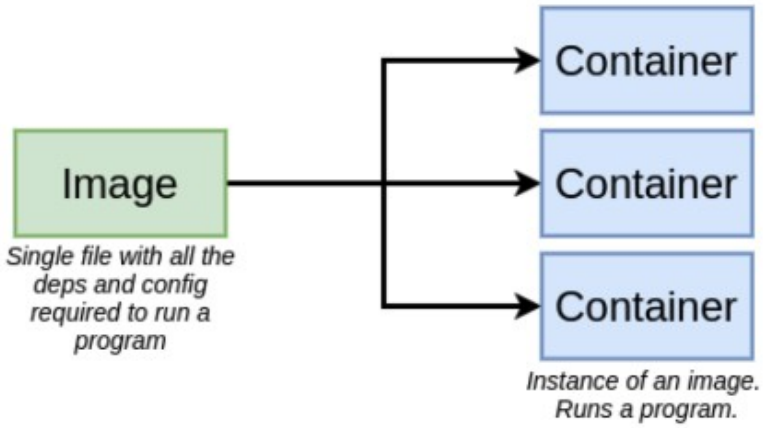
Για εκατομμύρια προγραμματιστές σήμερα, το Docker είναι το βασικό πρότυπο για τη δημιουργία και την κοινή χρήση εφαρμογών με containers - από την επιφάνεια εργασίας, μέχρι το cloud.



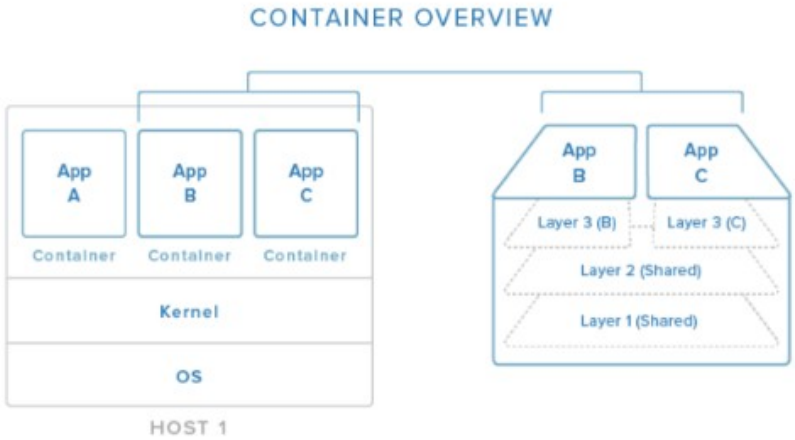
Εικόνα 21. Αρχιτεκτονική Docker [Πηγή: https://nordicapis.com/api-driven-devops-spotlight-on-docker/]

Ένα container είναι μια μονάδα λογισμικού που πακετάρει τον κώδικα και όλα τα αρχεία του, ώστε η εφαρμογή να εκτελείται γρήγορα και αξιόπιστα από το ένα υπολογιστικό περιβάλλον στο άλλο. Ένα container image είναι ένα ελαφρύ, αυτόνομο, εκτελέσιμο πακέτο λογισμικού που περιλαμβάνει όλα όσα απαιτούνται για την εκτέλεση μιας εφαρμογής: κώδικα, χρόνο εκτέλεσης, εργαλεία συστήματος, βιβλιοθήκες συστήματος και ρυθμίσεις.

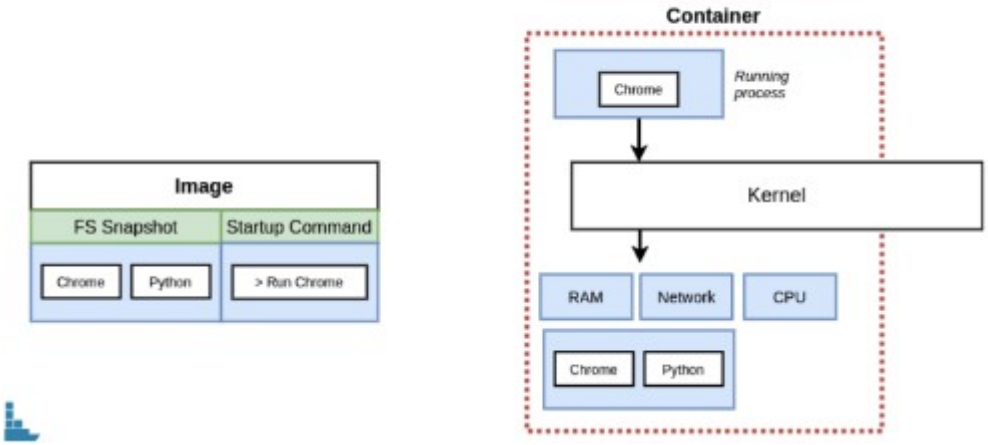
Τα images γίνονται containers κατά την εκτέλεση και στην περίπτωση των containers - τα images γίνονται containers όταν εκτελούνται στην Docker Engine. Είναι διαθέσιμο τόσο για εφαρμογές που βασίζονται σε Linux όσο και για εφαρμογές που βασίζονται σε Windows, το λογισμικό που περιέχει containers θα εκτελείται πάντα το ίδιο, ανεξάρτητα από την υποδομή. Τα containers απομονώνουν το λογισμικό από το περιβάλλον του και διασφαλίζουν ότι λειτουργεί ομοιόμορφα παρά τις διαφορές, για παράδειγμα, μεταξύ ανάπτυξης και παραγωγής.



Εικόνα 22 Images - Containers



Εικόνα 23 Επίπεδο των containers



Εικόνα 24 Από image σε container

```
$ docker run hello-world

Hello from Docker.
This message shows that your installation appears to be working correctly.
...
```

Εικόνα 25 Εκτέλεση container με βάση ένα image

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
```

Εικόνα 26 Εμφάνιση των containers

Ένα Dockerfile είναι ένα απλό αρχείο κειμένου που περιέχει μια λίστα εντολών που καλείται κατά τη δημιουργία μιας εικόνας. Είναι δηλαδή ένας απλός τρόπος αυτοματοποίησης της διαδικασίας δημιουργίας εικόνας. Οι εντολές που γράφουμε σε ένα Dockerfile είναι παρόμοιες με εντολές που χρησιμοποιούμε στο Linux. Αυτό σημαίνει ότι δεν χρειάζεται να γνωρίζει κάποιος ειδική σύνταξη προκειμένου να δημιουργήσει ένα Dockerfile. Μια καλή πρακτική είναι να προσθέσουμε και εντολές που μεταβιβάζουν τον έλεγχο σε έναν νέο χρήστη αντί να τρέχουν από τον root χρήστη του συστήματος. Ένα τέτοιο παράδειγμα βλέπουμε στην παρακάτω εικόνα.

```
FROM python:3
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/
```

Εικόνα 27 Σύνταξη Dockerfile [Πηγή: https://docs.docker.com/samples/django/]

Η εντολή docker build αναλαμβάνει τη δημιουργία ενός Docker image από ένα Dockerfile. Παρακάτω μπορούμε να δούμε τη δημιουργία ενός image. Πριν εκτελέσουμε την εντολή θα πρέπει να αντικαταστήσουμε το όνομα χρήστη με το δικό μας, το οποίο θα πρέπει να είναι το ίδιο με αυτό που φτιάχνουμε κατά την εγγραφή μας στο Docker hub ή σε όποιο container registry έχουμε σκοπό να χρησιμοποιήσουμε (όπως το Github Container Registry που θα δούμε αργότερα). Το Docker hub είναι ένα registry στο οποίο υπάρχουν images. Η εντολή docker build είναι αρκετά απλή καθώς παίρνει μια τοποθεσία του καταλόγου που περιέχει το αρχείο Docker, ένα προαιρετικό όνομα ετικέτας με -t και αν το Dockerfile μας έχει διαφορετικό όνομα το δηλώνουμε με -f και το όνομά του.

```
docker build . \
-t panagiotis-bellias-it21871/ref-letters-fastapi-server:latest \
-f nonroot.Dockerfile \
```

Εικόνα 28 Δημιουργία image για Docker Hub

```
docker build . \
-t ghcr.io/panagiotis-bellias-it21871/ref-letters-fastapi-server:latest \
-f nonroot.Dockerfile
```

Εικόνα 29 Δημιουργία image για Github Container Registry

Αφού δημιουργηθεί το image αξιοποιούμε το docker compose, ένα εργαλείο το οποίο εγκαθίσταται μαζί με το docker και μας βοηθά ώστε να εκτελέσουμε πολλαπλά containers την ίδια στιγμή, αυτοματοποιώντας τη διαδικασία. Γι' αυτό φτιάχνουμε ένα αρχείο docker-compose.yml. Αυτό περιέχει μία λίστα από containers, αντιστοιχίσεις πορτών (port mapping) και το πώς θα γίνει το χτίσιμο.

Προκειμένου να γίνει εκτέλεση γράφουμε την εντολή docker-compose up.

Η Ansible είναι λογισμικό αυτοματοποίησης διεργασιών πληροφορικής, το οποίο αυτοματοποιεί την παροχή cloud (cloud provisioning), τη διαχείριση παραμετροποιήσεων (configuration management), την ανάπτυξη εφαρμογών και συστημάτων, και την ενορχήστρωση εντός των υπηρεσιών. Βασίζεται στη γλώσσα προγραμματισμού Python και είναι σχεδιασμένη για πολυεπίπεδες αναπτύξεις. Μπορεί να περιγράψει τον τρόπο με τον οποίο όλα τα συστήματα και λογισμικά σχετίζονται μεταξύ τους, αντί να κάνουμε διαχείριση ενός συστήματος τη φορά.

Η Ansible τρέχει σε πολλά συστήματα βασισμένα στο Unix, όπως είναι τα Linux και τα Mac OS X ενώ μπορεί να παραμετροποιήσει και συστήματα Windows. Γράφτηκε από τον Michael DeHaan και αποκτήθηκε από την εταιρεία RedHat το 2015. Χρησιμοποιεί μία πολύ απλή γλώσσα, τη YAML, που τη χρησιμοποιούμε στην ανάπτυξη Ansible Playbooks τα οποία επιτρέπουν να περιγράφουμε τις εργασίες αυτοματοποίησης που θέλουμε να πετύχουμε.

Λειτουργεί με σύνδεση στους κόμβους (servers ή VMs) και προώθηση σε αυτούς μικρών προγραμμάτων - ενοτήτων που ονομάζουμε Ansible Modules. Η Ansible εκτελεί αυτά τα modules (μέσω SSH πρωτοκόλλου σύνδεσης που θα δούμε) και τα αφαιρεί όταν ολοκληρώσουν την εργασία τους.

Η αναπαράσταση των μηχανημάτων που διαχειρίζεται γίνεται με τη χρήση αρχείου INI όπου ομαδοποιεί τα διαχειριζόμενα μηχανήματα με τον τρόπο που επιθυμούμε, όπως φαίνεται στην εικόνα 27.

```
# You first must have the corresponding ~/.ssh/config with the names for the hosts below
# matching these in that file referring to the Host
jenkins_server:
    hosts:
        jenkins-vm

docker_group:
    hosts:
        docker-vm

k8s_group:
    hosts:
        k8s-vm
```

Εικόνα 30 Δομή Του Inventory File

Τα playbooks έχουν την ικανότητα με λεπτομέρεια να ενορχηστρώνουν πολλαπλά σημεία της τοπολογίας της υποδομής, με λεπτομερή έλεγχο σχετικά με το πλήθος των servers που θα υπάρχουν κάθε φορά. Εδώ η Ansible αρχίζει να αποκτά περισσότερο ενδιαφέρον.

Η προσέγγιση της Ansible για το orchestration είναι προσέγγιση απλότητας καθώς ο κώδικας που χρησιμοποιούμε για αυτοματοποίηση έχει νόημα για καιρό αφού δε χρειάζεται να θυμόμαστε την ειδική σύνταξή του ή τα χαρακτηριστικά του. Στην περίπτωσή μας η Ansible θα μας βοηθήσει να αυτοματοποίησουμε το dockerization του συστήματος και το deployment αυτού αλλά και το deployment σε kubernetes περιβάλλον

```
---
- hosts: webservers
serial: 5 # update 5 machines at a time
roles:
- common
- webapp
```

Εικόνα 31. Δομή Playbook [Πηγή: https://www.ansible.com/overview/how-ansible-works]

Ο Jenkins είναι δρομολογητής αυτοματοποίησης ανοιχτού κώδικα. Βοηθάει στην αυτοματοποίηση κομματιών της ανάπτυξης λογισμικού σχετικά με το χτίσιμο, τη δοκιμή, και την ανάπτυξη (build-test-deploy).

Ο Jenkins διαχειρίζεται και ελέγχει τις διαδικασίες παράδοσης λογισμικού σε ολόκληρο τον κύκλο ζωής, συμπεριλαμβανομένης της κατασκευής, της τεκμηρίωσης, της δοκιμής, του σταδίου, της ανάπτυξης, της στατικής ανάλυσης κώδικα. Μπορούμε να ρυθμίσουμε το Jenkins ώστε να παρακολουθεί τυχόν αλλαγές κώδικα σε μέρη όπως το GitHub, το Bitbucket ή το GitLab και να κάνει αυτόματα μια κατασκευή (build) με εργαλεία όπως το Maven και το Gradle.

Η συνεχής ενσωμάτωση (Continuous integration/CI) είναι μια φιλοσοφία κωδικοποίησης και ένα σύνολο πρακτικών που οδηγούν τις ομάδες ανάπτυξης να εφαρμόζουν μικρές αλλαγές και να ελέγχουν συχνά τα αποθετήρια ελέγχου μετατροπής κώδικα. Επειδή οι περισσότερες σύγχρονες εφαρμογές απαιτούν την ανάπτυξη κώδικα σε διαφορετικές πλατφόρμες και εργαλεία, χρειάζεται ένας μηχανισμός για την ενσωμάτωση και την επικύρωση των αλλαγών της.

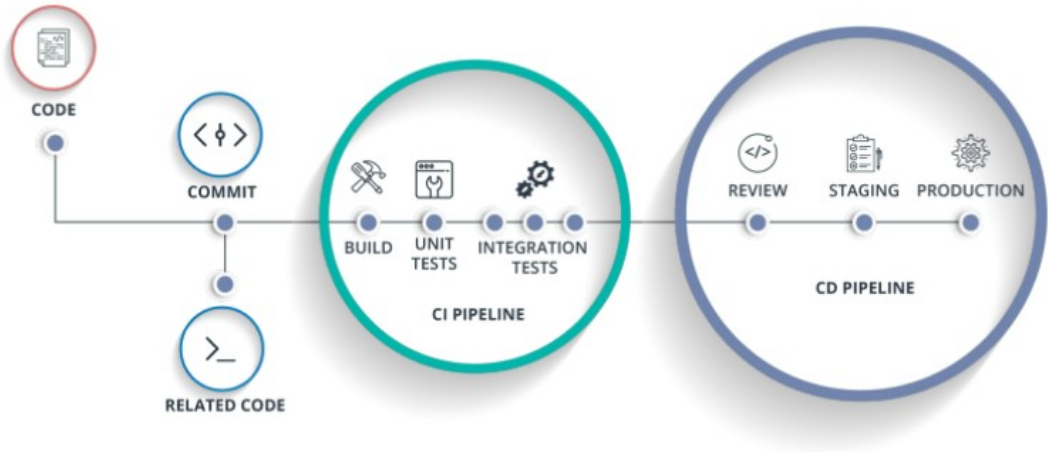
Η συνεχής παράδοση (Continuous Delivery/CD) συνεχίζει από εκεί που τελειώνει η συνεχής ενσωμάτωση. Το CD αυτοματοποιεί την παράδοση των εφαρμογών σε επιλεγμένα περιβάλλοντα υποδομής. Οι περισσότερες ομάδες εργάζονται σε πολλαπλά περιβάλλοντα, όπως περιβάλλοντα ανάπτυξης και δοκιμών, και το CD εξασφαλίζει ότι υπάρχει ένας αυτοματοποιημένος τρόπος για την προώθηση των αλλαγών κώδικα σε αυτά.

Η συνεχής ενσωμάτωση και η συνεχής παράδοση απαιτούν διαρκείς δοκιμές, επειδή ο στόχος είναι να παραδίδονται ποιοτικές εφαρμογές και κώδικας στους χρήστες. Οι συνεχείς δοκιμές συχνά υλοποιούνται ως ένα σύνολο αυτοματοποιημένων δοκιμών παλινδρόμησης, επιδόσεων και άλλων δοκιμών που εκτελούνται στο ευρέως γνωστό CI/CD pipeline (αγωγός).

Το pipeline είναι ένα σύνολο αυτοματοποιημένων διαδικασιών που επιτρέπουν στους προγραμματιστές να μεταγλωττίζουν, να χτίζουν και να αναπτύσσουν αξιόπιστα και αποτελεσματικά τον κώδικά τους σε πλατφόρμες παραγωγής. Δεν υπάρχει κανένας αυστηρός κανόνας που να ορίζει πώς πρέπει να μοιάζει ένα pipeline και τα εργαλεία που πρέπει να χρησιμοποιεί, ωστόσο τα πιο συνηθισμένα στοιχεία ενός pipeline είναι τα εξής: χτίσιμο αυτοματοποίησης/συνεχούς ενσωμάτωσης, αυτοματοποίηση δοκιμών και αυτοματοποίηση ανάπτυξης.

Ένα pipeline αποτελείται γενικά από ένα σύνολο εργαλείων που συνήθως αναλύονται στις εξής κατηγορίες:

- Έλεγχος πηγής (Source Control)
- Εργαλεία κατασκευής (Build tools)
- Containerisation (πληροφορίες παρακάτω)
- Διαχείριση παραμετροποίησης (Configuration Management)
- Παρακολούθηση (Monitoring)



Εικόνα 32. CI/CD Pipeline [Πηγή: https://medium.com/@singlanitish29/automated-ci-cd-pipeline-8124eb8dc136]

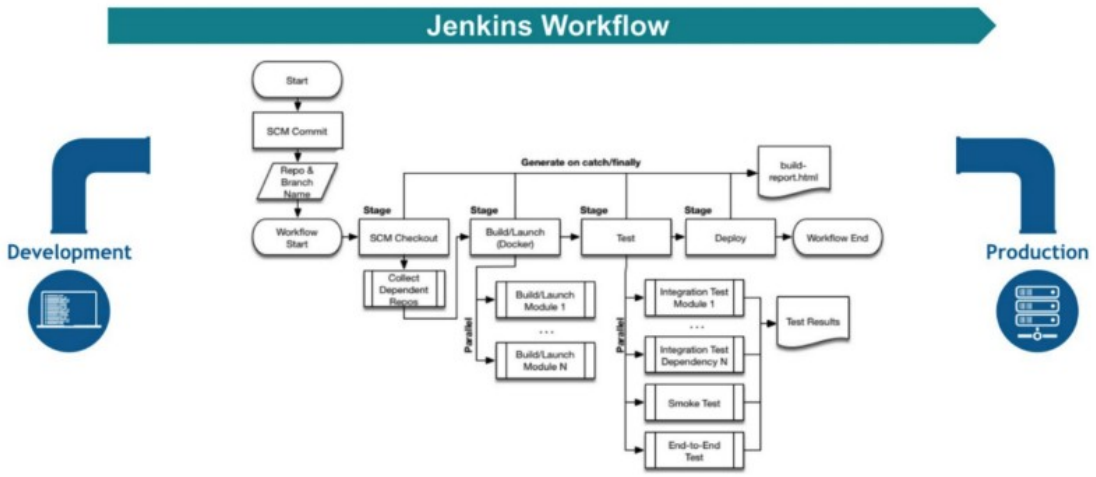
Τα συστατικά (components) ενός CI/CD pipeline είναι:

- Αποθετήριο κώδικα (π.χ. github)
- Αποθετήριο εικόνων (π.χ. Docker Hub, GitHub Container Registry)
- Περιβάλλον CI/CD (π.χ. Jenkins)
- Περιβάλλον(τα) ανάπτυξης (π.χ. VMs, Kubernetes Cluster)

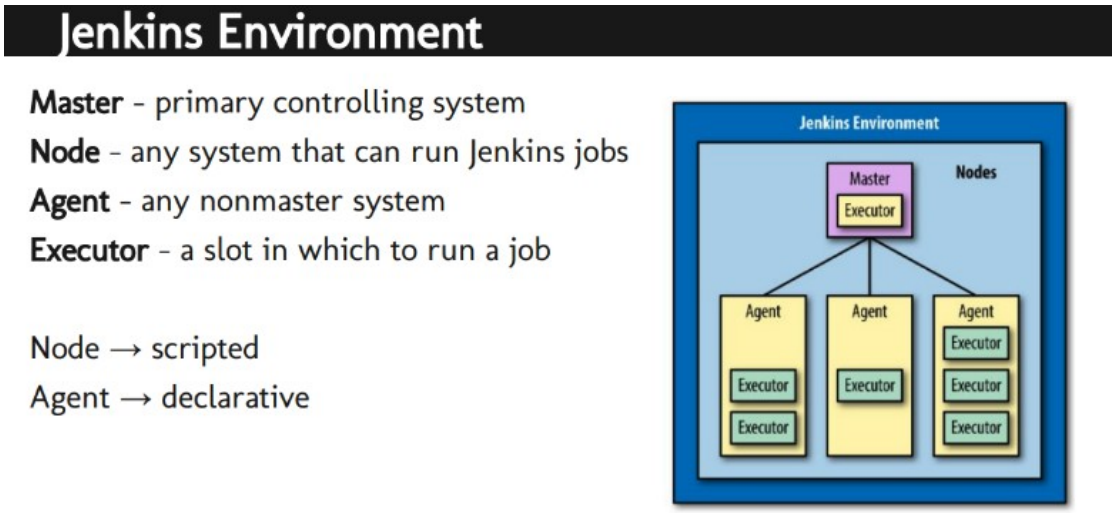
Παρακάτω απεικονίζεται η σύνταξη και η δομή ενός pipeline. Ένα stage στον Jenkins περιγράφει τη δουλειά που γίνεται κάθε φορά σε ένα συγκεκριμένο κομμάτι του κώδικα. Τα steps δείχνουν κάθε βήμα που ακολουθείται μέσα σε ένα stage. [1]

```
pipeline {
    agent any
    stages {
        stage('Example Build') {
            steps {
                echo 'Hello World'
            }
        }
        stage('Example Deploy') {
            when {
                branch 'production'
                environment name: 'DEPLOY_TO', value: 'production'
            }
            steps {
                echo 'Deploying'
            }
        }
    }
}
```

Εικόνα 33. Σύνταξη Pipeline



Εικόνα 34. Ροή Jenkins



Εικόνα 35. Περιβάλλον Jenkins

Στο επόμενο στάδιο έχουμε το Kubernetes. Το Kubernetes, γνωστό και ως K8s, είναι ένα σύστημα ανοικτού κώδικα για την αυτοματοποίηση της ανάπτυξης, της κλιμάκωσης και της διαχείρισης εφαρμογών με container.

Ομαδοποιεί τα containers που αποτελούν μια εφαρμογή σε λογικές μονάδες για εύκολη διαχείριση.

Το Kubernetes επιτρέπει να ενορχηστρώσουμε ένα σύνολο/συστάδα/cluster εικονικών μηχανών και να προγραμματίσουμε την εκτέλεση containers σε αυτές τις εικονικές μηχανές με βάση τους διαθέσιμους υπολογιστικούς πόρους και τις απαιτήσεις πόρων κάθε container. Τα containers ομαδοποιούνται σε pods, τη βασική λειτουργική μονάδα του Kubernetes. Τα containers και τα pods μπορούν να κλιμακωθούν στην επιθυμητή κατάσταση και είμαστε σε θέση να διαχειριστούμε τον κύκλο ζωής τους για να διατηρούμε τις εφαρμογές σας σε λειτουργία.

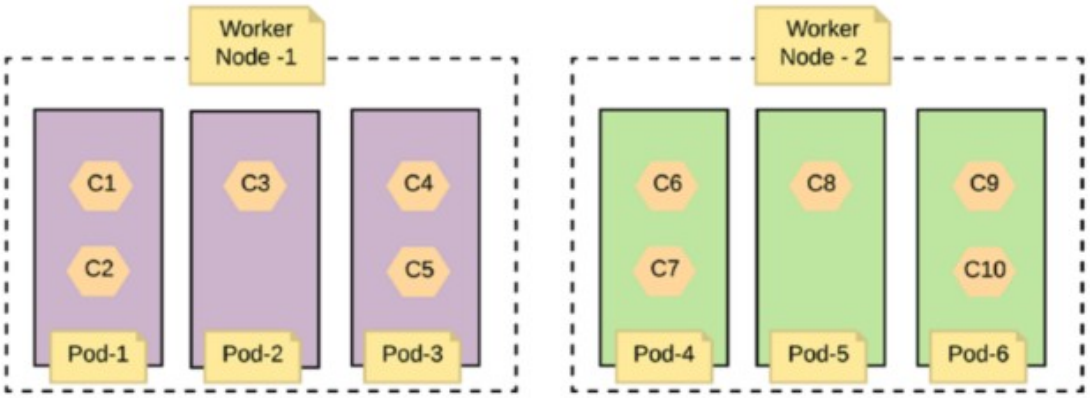
Το Kubernetes προσφέρει ορισμένα πολύ σημαντικά πλεονεκτήματα τα οποία αφορούν την αρχιτεκτονική του. Υποστηρίζει την επεκτασιμότητα δηλαδή μπορεί να κλιμακώσει τη δημιουργία pods ανάλογα με τη χρήση της CPU, την υψηλή διαθεσιμότητα και σε επίπεδο εφαρμογής αλλά και σε επίπεδο υποδομής, την ασφάλεια πολλαπλών επιπέδων (εφαρμογής, δικτύου κτλ) και τέλος τη φορητότητα.

Κάποιος, ωστόσο μπορεί να αναρωτηθεί πώς διαφοροποιείται σε σχέση με το Docker. Εδώ αξίζει να αναφέρουμε ότι μια θεμελιώδης διαφορά μεταξύ του Kubernetes και του Docker είναι ότι το Kubernetes προορίζεται να εκτελείται σε ένα cluster, ενώ το Docker εκτελείται σε έναν μόνο κόμβο. Το Kubernetes είναι πιο εκτεταμένο από το Docker και προορίζεται να συντονίζει συστάδες κόμβων σε κλίμακα παραγωγής με αποτελεσματικό τρόπο. Τα pods του Kubernetes (μονάδες προγραμματισμού που μπορούν να περιέχουν ένα ή περισσότερα containers στο οικοσύστημα Kubernetes) κατανέμονται μεταξύ των κόμβων για να παρέχουν υψηλή διαθεσιμότητα.

Μπορούμε επομένως να αντιληφθούμε τις τεράστιες δυνατότητες που παρουσιάζονται από τον συνδυασμό του Docker και του Kubernetes σε μια εφαρμογή.

Η από κοινού χρήση τους μας βοηθά:

1) Να κάνουμε την υποδομή πιο ισχυρή και την εφαρμογή πιο υψηλής διαθεσιμότητας. Η εφαρμογή θα παραμείνει σε λειτουργία, ακόμη και αν κάποιοι από τους κόμβους τεθούν εκτός λειτουργίας.
2) Να κάνουμε την εφαρμογή πιο επεκτάσιμη. Αν η εφαρμογή αρχίσει να δέχεται πολύ μεγαλύτερο φορτίο και πρέπει να επεκταθεί για να μπορέσουμε να παρέχουμε καλύτερη εμπειρία στους χρήστες, δημιουργούμε περισσότερα containers ή προσθέτουμε περισσότερους κόμβους στο cluster.



Εικόνα 36. Pods και Containers

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: croc-hunter
  labels:
    app: croc-hunter
spec:
  containers:
    - name: croc-hunter
      image: quay.io/lachie83/croc-hunter
      ports:
        - containerPort: 8080
```

Εικόνα 37. Σύνταξη Pod

Εκτέλεση / deployment ενός pod:

```
kubectl apply -f <pod-file>.yaml
```

Εικόνα 38. Δημιουργία deployment

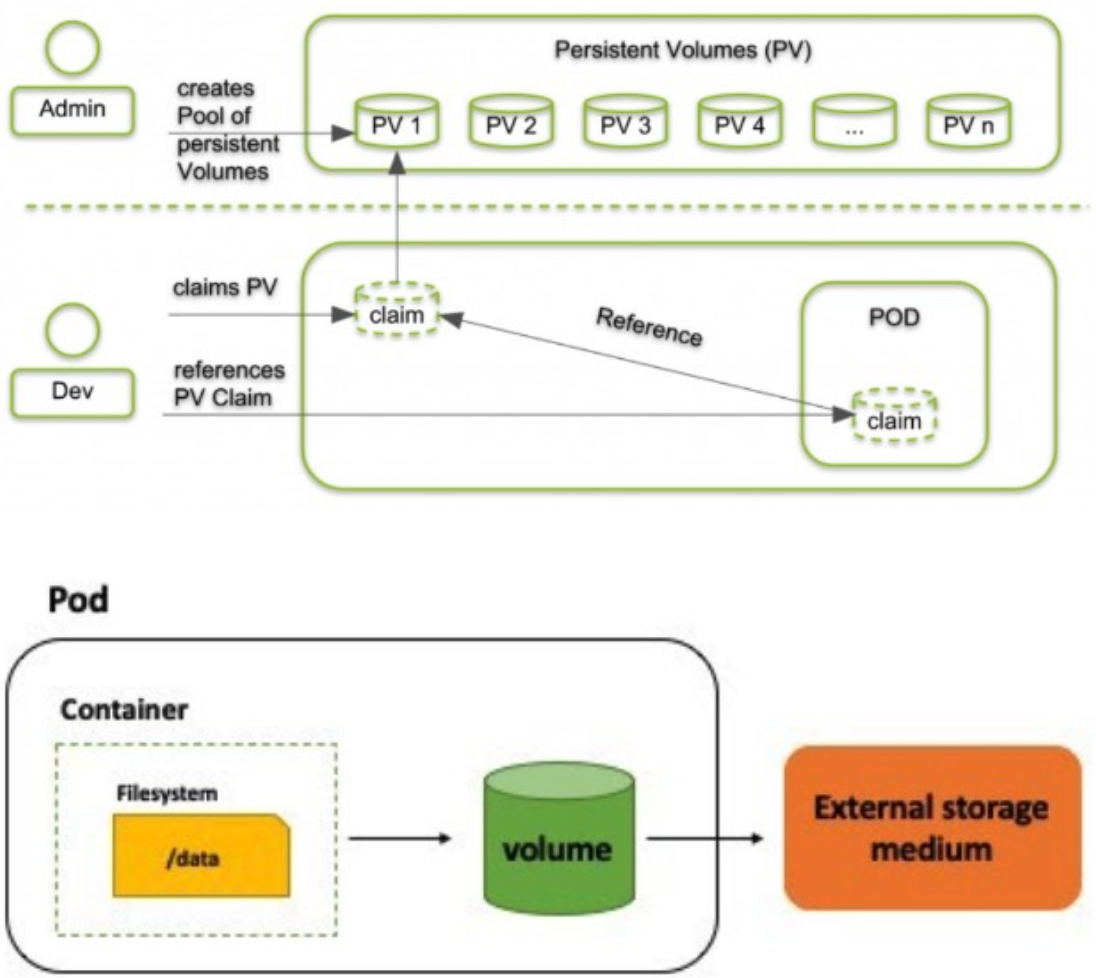
Αφού είδαμε τα pods, αξίζει να σημειώσουμε το τρόπο που γίνεται η προσπέλαση τους. Αυτό γίνεται με ένα Service, μια αφαιρετικότητα δηλαδή που ορίζει ένα σύνολο από pods και κανόνες με τους οποίους μπορούμε να τα προσπελάσουμε. Το σύνολο των pods που 'στοχεύει' ένα service καθορίζεται από ένα selector.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: db-service
  kind: ClusterIP
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Εικόνα 39. Σύνταξη Service

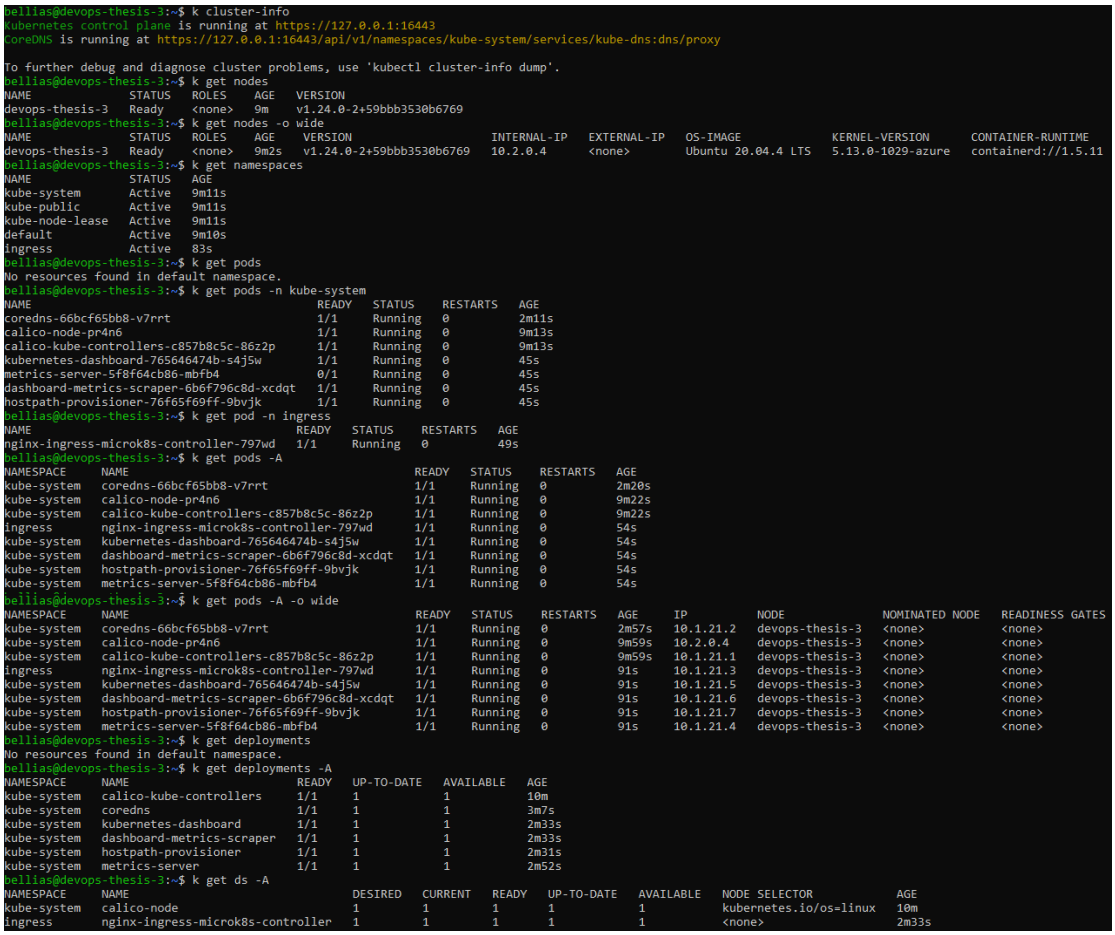
Περισσότερες πληροφορίες σχετικά με τα services και για τους τύπους τους, βρίσκονται εδώ. [4]

Στο περιβάλλον του kubernetes θα συναντήσουμε αρκετές ακόμα ορολογίες που αφορούν την εκτέλεση των εφαρμογών. Στα πλαίσια της παρούσας πτυχιακής εργασίας, μερικές από τις ορολογίες που θα ήταν χρήσιμο να αναφερθούν είναι ο Ingress controller, ένα pod που ουσιαστικά επιτρέπει την επικοινωνία της εφαρμογής με τον 'έξω κόσμο' λειτουργώντας δηλαδή σαν proxy server για να δούμε την εφαρμογή, τα volumes τα οποία είναι χώροι όπου αποθηκεύονται δεδομένα σχετικά με τα deployments, τα secrets στα οποία αποθηκεύονται key-value ζεύγη π.χ για τη σύνδεση με τη βάση μας, και τα configmaps τα οποία περιέχουν key-value ζεύγη όσον αφορά την παραμετροποίηση της εφαρμογής μας (π.χ .env αρχεία). [5]

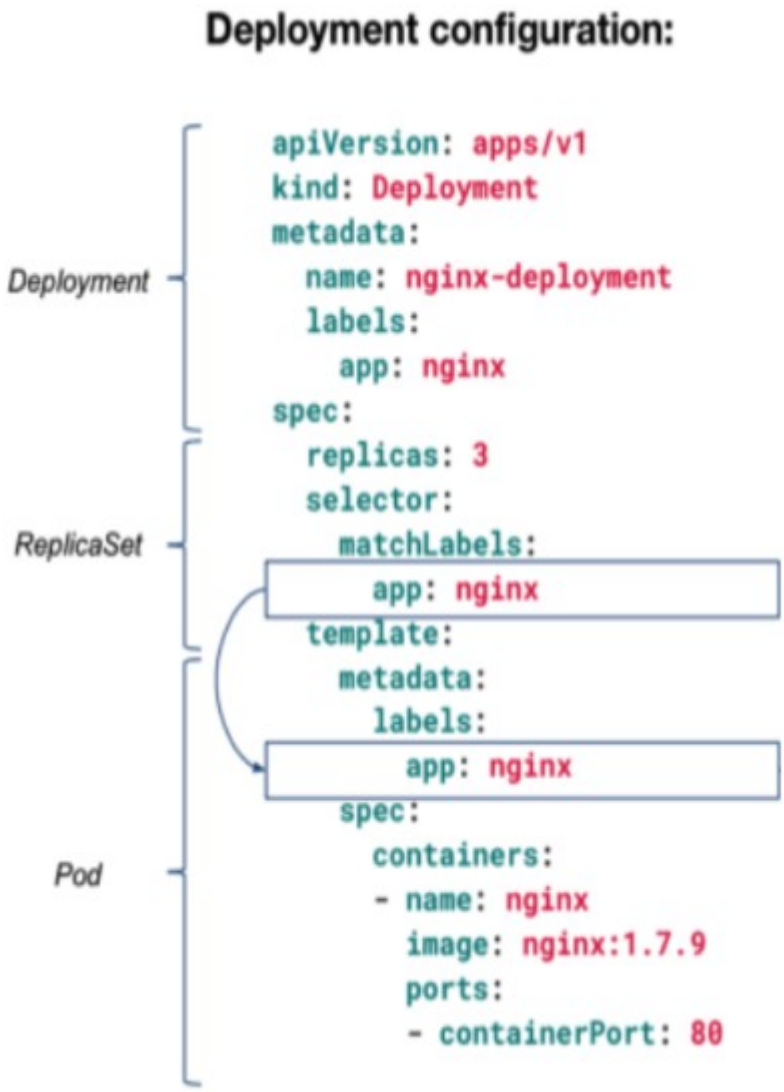


Εικόνα 40. Volumes

```
kubectl  get       po
         describe  deployment
         delete    svc          -n NAMESPACE   -o wide
                   cm                          -o yaml
                   secrets
                   nodes
```



Εικόνα 41. Βασικές Εντολές Kubernetes και Ενδεικτικά Αποτελέσματα



Εικόνα 42. Σύνταξη Ενός Deployment

ΚΕΦ.4: Υλοποίηση και Παραμετροποίηση Συστήματος

4.1 Βασική Δομή - Παραδοχές

Στα πλαίσια του βασικού συστήματος θα ήταν χρήσιμο να αναφερθούν επιπλέον ορισμένες παραδοχές που έχουν γίνει και αφορούν την υλοποίηση:

1. Υπάρχουν 2 φόρμες sign up. Μία για τους students και μία για τους teachers.
2. Αν κάποιος επιθυμεί να κάνει register ως student, καταχωρείται αυτόματα στο σύστημα ως student αφού κάνει sign up (σε διαφορετική sign up form).
3. Ο teacher μπορεί να δει λίστα με τα reference letter requests που τον αφορούν, να δει πληροφορίες σχετικά με αυτά και να τα αποδεχθεί ή να τα απορρίψει.
4. Ο teacher προκειμένου να αποδεχθεί ένα reference letter request πρέπει να ανεβάσει pdf με το σχετικό κείμενο ή απλά να το γράψει επί τόπου.
5. Ο student μπορεί να δει λίστα με τα reference letter requests που έχει κάνει, να δει την κατάστασή τους (approved/declined/pending) ή να τα ανακαλέσει / διαγράψει.
6. Ο student μπορεί να κάνει reference letter request συμπληρώνει τα στοιχεία του και το λόγο για τον οποίο θέλει να λάβει συστατική επιστολή.
7. Ο admin μπορεί να επεξεργαστεί τα στοιχεία των χρηστών του συστήματος αλλά και να επιβλέπει τα στοιχεία των αιτήσεων συστατικών επιστολών.

4.2 Αυθεντικοποίηση Χρηστών

Κάθε σύγχρονη διαδικτυακή εφαρμογή απαιτεί αυθεντικοποίηση των χρηστών και αποθήκευση των δεδομένων τους αφενός έχοντας σκοπό την ασφάλεια των δεδομένων και τη διαχείριση αυτών από τους πιο κατάλληλους χρήστες ανά περίπτωση και αφετέρου για βελτίωση της εμπειρίας του χρήστη όσον αφορά σε τυχόν recommendations που κάνει το σύστημα. Στην παρούσα πτυχιακή εργασία μάς αφορά να “ασφαλίσουμε” το διαδικτυακό σύστημά μας προκειμένου να προστατεύονται τα δεδομένα μας και οι χρήστες να έχουν συγκεκριμένα δικαιώματα ανάλογα με το ρόλο τους. Οπότε προκειμένου να επιτευχθεί αυτό θα “ασφαλίσουμε” την backend εφαρμογή και το REST API που παρέχει με το αντίστοιχο module. Η frontend εφαρμογή καλώντας την backend εφαρμογή και το REST API θα παρέχει στο χρήστη εγγραφή και σύνδεση στο σύστημα με συγκεκριμένο ρόλο και διαχείριση των αιτήσεων συστατικών επιστολών ανάλογα το ρόλο του. Επίσης θα παρέχεται λογαριασμός διαχειριστή όπου ο εν λόγω χρήστης θα μπορεί να διαχειρίζεται τα δεδομένα των χρηστών και των αιτήσεων συστατικών επιστολών.

4.2.1 Διαδικασία Ενσωμάτωσης στην FastAPI Εφαρμογή

Προκειμένου να ενσωματώσουμε υπηρεσίες αυθεντικοποίησης στην backend fastapi εφαρμογή μας ξεκινήσαμε κάνοντας εγκατάσταση το αντίστοιχο module όπως φαίνεται στην εικόνα 39. Επειδή χρησιμοποιούμε ήδη το sqlalchemy module για σύνδεση με τη βάση δεδομένων μας κάνουμε εγκατάσταση το αντίστοιχο module. [6]

```
pip install 'fastapi-users[sqlalchemy]'
```

Εικόνα 43. Εγκατάσταση του fastapi-users module στην python

Χρειάζεται να εγκατασταθεί το αντίστοιχο πρόγραμμα οδήγησης asyncio. Κοινές επιλογές είναι:

- Για PostgreSQL: pip install asyncpg
- Για SQLite: pip install aiosqlite

Παραδείγματα DB_URL είναι:

```
PostgreSQL: engine = create_engine
('postgresql+asyncpg://user:password@host:port/name')
SQLite: engine = create_engine ('sqlite+aiosqlite:///name.db')
```

Περισσότερα σχετικά με τη διαδικασία αυτή της ενσωμάτωσης της βιβλιοθήκης στη fastapi εφαρμογή μπορείτε να βρείτε εδώ. [7]

Όσον αφορά τη διαδικασία που ακολουθεί ο χρήστης ώστε να χρησιμοποιήσει με ασφάλεια το διαδικτυακό σύστημα, έχουμε:

- Εγγραφή στο σύστημα.
- Επιβεβαίωση email για επαλήθευση λογαριασμού χρήστη.
- Σύνδεση στο σύστημα με credentials. Εδώ επιστρέφεται το access json web token (jwt) όπου χρησιμοποιείται για την πρόσβαση των χρηστών στις λειτουργίες του συστήματος.
- Επισκόπηση προφίλ, δηλαδή των στοιχείων του.
- Ενημέρωση προφίλ, δηλαδή αλλαγή ενός ή περισσοτέρων στοιχείων του.
- Αποσύνδεση από το σύστημα.

4.2.2 Διαδικασία Ενσωμάτωσης στην VueJS Εφαρμογή

Το REST API που ρυθμίστηκε πριν λίγο με χρήση του fastapi-users module μπορούμε να το καλέσουμε μέσω της VueJS εφαρμογής κάνοντας χρήση του axios module. Επίσης για να παρέχεται πρόσβαση στις λειτουργίες του συστήματος χωρίς να πρέπει ο χρήστης να κάνει συνέχεια σύνδεση, χρησιμοποιείται το local storage του vuejs για να αποθηκεύεται ή να προσπελαύνεται το access token για να χρησιμοποιείται στα requests που γίνονται στην backend εφαρμογή.

Περισσότερα εδώ. [8]

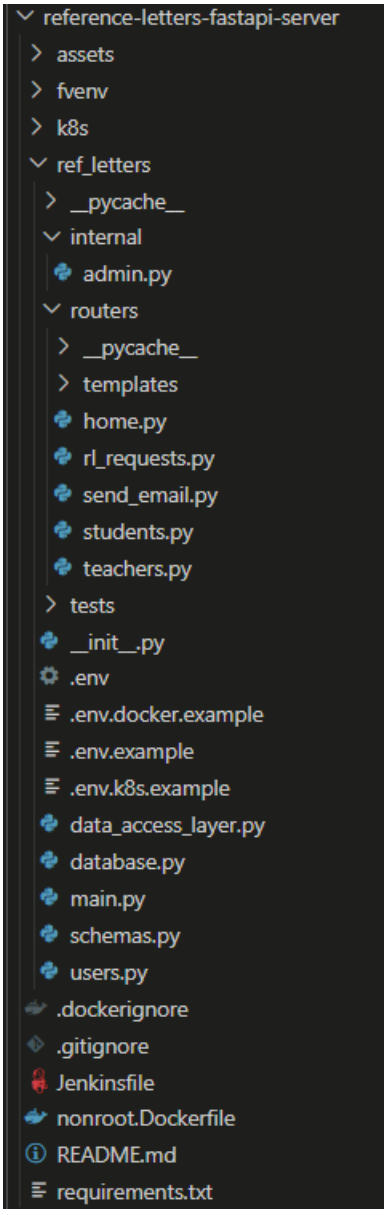
4.3 Backend Εφαρμογή - FastAPI

Το FastAPI είναι ένα Web Framework για την ανάπτυξη REST API στην Python. Το FastAPI βασίζεται σε Pydantic και υποδείξεις τύπου για την επικύρωση, τη σειριοποίηση και την αποσειριοποίηση δεδομένων και την αυτόματη δημιουργία εγγράφων OpenAPI.

Υποστηρίζει πλήρως τον ασύγχρονο προγραμματισμό και μπορεί να τρέξει με το Uvicorn και το Gunicorn. Για να βελτιωθεί η φιλικότητα προς τους προγραμματιστές, εξετάστηκε η υποστήριξη του συντάκτη από τις πρώτες ημέρες του έργου.

Το σύστημα διαχείρισης συστατικών επιστολών, στην παρούσα πτυχιακή εργασία, αποτελείται από δύο εφαρμογές. Η μία περιγράφεται εδώ και χρησιμοποιεί το FastAPI ώστε να παρέχει ένα API (Application Programming Interface) όπου θα επικοινωνεί με τη δεύτερη εφαρμογή η οποία θα εξυπηρετεί τον τελικό χρήστη. Αυτό το API θα παρέχει δεδομένα σχετικά με αιτήσεις για συστατικές επιστολές, και με τους χρήστες του συστήματος. Οι ρόλοι αυτών των χρηστών είναι student, teacher και admin. Το βασικό προγραμματιστικό αρχείο είναι το main.py, το οποίο καλεί άλλα αρχεία γραμμένα σε python και όχι μόνο ανάλογα με τις ανάγκες της εφαρμογής. Αυτά έχουν να κάνουν κυρίως με τη σύνδεση στη βάση δεδομένων που έχουμε διαθέσιμη και στις κλάσεις που αναπαριστούν τα δεδομένα που θέλουμε να χειριστούμε. Έχουμε και αρχείο με τις παραμέτρους σύνδεσης στη βάση δεδομένων, οι οποίες αλλάζουν αναλόγως του περιβάλλοντος εγκατάστασης της εφαρμογής αλλά και με τις παραμέτρους σύνδεσης με άλλες υπηρεσίες που θα δούμε παρακάτω και μας βοηθούν στη σωστή διαχείριση όλων των δεδομένων του συστήματος.

Προκειμένου να οργανώσουμε καλύτερα τη δομή του κώδικα στην εφαρμογή μας χρησιμοποιούμε ένα module του fastapi που λέγεται APIRouter και μας επιτρέπει να χωρίζουμε τον κώδικά μας με τον οποίο έχουμε δηλώσει τα endpoints μας σε περισσότερα αρχεία ανάλογα με το τι διαχειρίζονται και να καλούμε αυτά τα διαφορετικά αρχεία στο κύριο αρχείο μας main.py. Στο σύστημα που καλούμαστε να αναπτύξουμε μπορούμε να πούμε πως θα έχουμε ένα αρχείο για τα endpoints που διαχειρίζονται τις αιτήσεις συστατικών επιστολών, άλλο αρχείο για αυτά που διαχειρίζονται φοιτητές, άλλο για αυτά που διαχειρίζονται καθηγητές και άλλο για τα endpoints που χρησιμοποιεί ο διαχειριστής του συστήματος.



Εικόνα 44. Δομή FastAPI Εφαρμογής

```python
app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=origin,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(home.router,tags=["/"])
app.include_router(rl_requests.router,tags=["rl_requests"])
app.include_router(students.router,tags=["students"])
app.include_router(teachers.router,tags=["teachers"])

app.include_router(
    fastapi_users.get_users_router(UserRead, UserUpdate, requires_verification=True),
    prefix="/users",
    tags=["users"]
)

app.include_router(
    fastapi_users.get_auth_router(auth_backend, requires_verification=True),
    prefix="/auth/jwt",
    tags=["auth"]
)
app.include_router(
    fastapi_users.get_register_router(UserRead, UserCreate),
    prefix="/auth",
    tags=["auth"]
)
app.include_router(
    fastapi_users.get_verify_router(UserRead),
    prefix="/auth",
    tags=["auth"],
)
```

Εικόνα 45. Κώδικας κλήσης των routers

Συνοπτικά αναφέρονται τα endpoints που έχουν αναπτυχθεί σχετικά με τη διαχείριση των αιτήσεων για συστατικές επιστολές. Ακολουθείται το μοντέλο CRUD (Create, Read, Update, Delete)

Prefix: /api

URL Path	**HTTP Method**	**Description**	**Data**
/rl_requests	GET	Ανάκτηση όλων των αιτήσεων	Όλες οι αποθηκευμένες αιτήσεις στη βάση
/rl_requests	POST	Δημιουργία νέας αίτησης	Στοιχεία νέας αίτησης
/rl_requests/ {rl_request_id}	GET	Ανάκτηση της αίτησης με το δοθέν id	Μία αίτηση με id το δοθέν id
/rl_requests/ {rl_request_id}	PUT	Ενημέρωση της αίτησης με το δοθέν id	Μία αίτηση με id το δοθέν id
/rl_requests/s/ {student_id}	GET	Ανάκτηση των αιτήσεων του φοιτητή με το δοθέν id	Οι αποθηκευμένες αιτήσεις ενός φοιτητή
/rl_requests/t/ pending/ {teacher_id}	GET	Ανάκτηση των εκκρεμών αιτήσεων που αφορούν τον καθηγητή με το δοθέν id	Οι εκκρεμείς αιτήσεις που αφορούν συγκεκριμένο καθηγητή
/rl_requests/t/ {rl_request_id}/ approve	PUT	Αποδοχή μιας συγκεκριμένης αίτησης	ID αίτησης, κείμενο αποδοχής της αίτησης
/rl_requests/t/ {rl_request_id}/ decline	DECLINE	Απόρριψη μιας συγκεκριμένης αίτησης	ID αίτησης

Πίνακας 1. Endpoints Διαχείρισης Αιτήσεων Συστατικών Επιστολών

Επίσης αναφέρονται τα endpoints που έχουν αναπτυχθεί σχετικά με τη διαχείριση των στοιχείων των φοιτητών και καθηγητών. Ακολουθείται και πάλι το μοντέλο CRUD (Create, Read, Update, Delete)

Prefix: /api

URL Path	Method	Description	Data
/students	GET	Ανάκτηση όλων των φοιτητών	Όλοι οι αποθηκευμένοι φοιτητές στη βάση
/students	POST	Καταχώρηση νέου φοιτητή	Στοιχεία νέου φοιτητή
/students/ {student_id}	GET	Ανάκτηση του φοιτητή με το δοθέν id	Ένας φοιτητής με id το δοθέν id
/students/ {student_id}	PUT	Ενημέρωση του φοιτητή με το δοθέν id	Ένας φοιτητής με id το δοθέν id
/teachers	GET	Ανάκτηση όλων των καθηγητών	Όλοι οι αποθηκευμένοι καθηγητές στη βάση
/teachers	POST	Καταχώρηση νέου καθηγητή	Στοιχεία νέου καθηγητή
/teachers/ {teacher_id}	GET	Ανάκτηση του καθηγητή με το δοθέν id	Ένας καθηγητής με id το δοθέν id
/teachers/ {teacher_id}	PUT	Ενημέρωση του καθηγητή με το δοθέν id	Ένας καθηγητής με id το δοθέν id

Πίνακας 2. Endpoints Διαχείρισης Στοιχείων Φοιτητών / Καθηγητών

4.4 Frontend Εφαρμογή - VueJS

Το VueJS είναι ένα Web Framework για την ανάπτυξη εφαρμογών user interface σε Javascript. Χρησιμοποιεί συνήθως το axios module προκειμένου να έχει τη δυνατότητα να καλέσει web services και REST APIs.

Με σκοπό να παραμετροποιηθεί στις ανάγκες μας χρησιμοποιούμε το δημοφιλέστερο package manager που υπάρχει για Javascript, το npm όπου βασίζεται στο nodejs.

Η backend εφαρμογή όπως είπαμε χρησιμοποιεί το FastAPI ώστε να παρέχει ένα API (Application Programming Interface) όπου θα επικοινωνεί με την εφαρμογή που θα αναλύσουμε εδώ και η οποία θα εξυπηρετεί τον τελικό χρήστη. Με χρήση του axios module θα παίρνουμε δεδομένα από την backend εφαρμογή και το αντίστοιχο διαθέσιμο REST API σχετικά με αιτήσεις για συστατικές επιστολές, και με τους χρήστες του συστήματος. Οι ρόλοι αυτών των χρηστών είναι student, teacher και admin. Η προγραμματιστική αρχιτεκτονική του project βασίζεται σε components και views που επικοινωνούν μεταξύ τους. Συνήθως όλα αυτά τα αρχεία έχουν κατάληξη .vue και αποτελούνται από τρεις ενότητες. Η πρώτη είναι απλή HTML όπου δηλώνουμε τι στοιχεία θα φαίνονται στο χρήστη. Η δεύτερη είναι το CSS κομμάτι που αλλάζουμε αυτά τα HTML στοιχεία όσον αφορά την εμφάνιση. Τέλος έχουμε τη JavaScript όπου μπαίνει η λογική της εφαρμογής η οποία χειρίζεται τα δεδομένα και τα παρουσιάζει στον τελικό χρήστη.

Τα components και URL paths είναι:

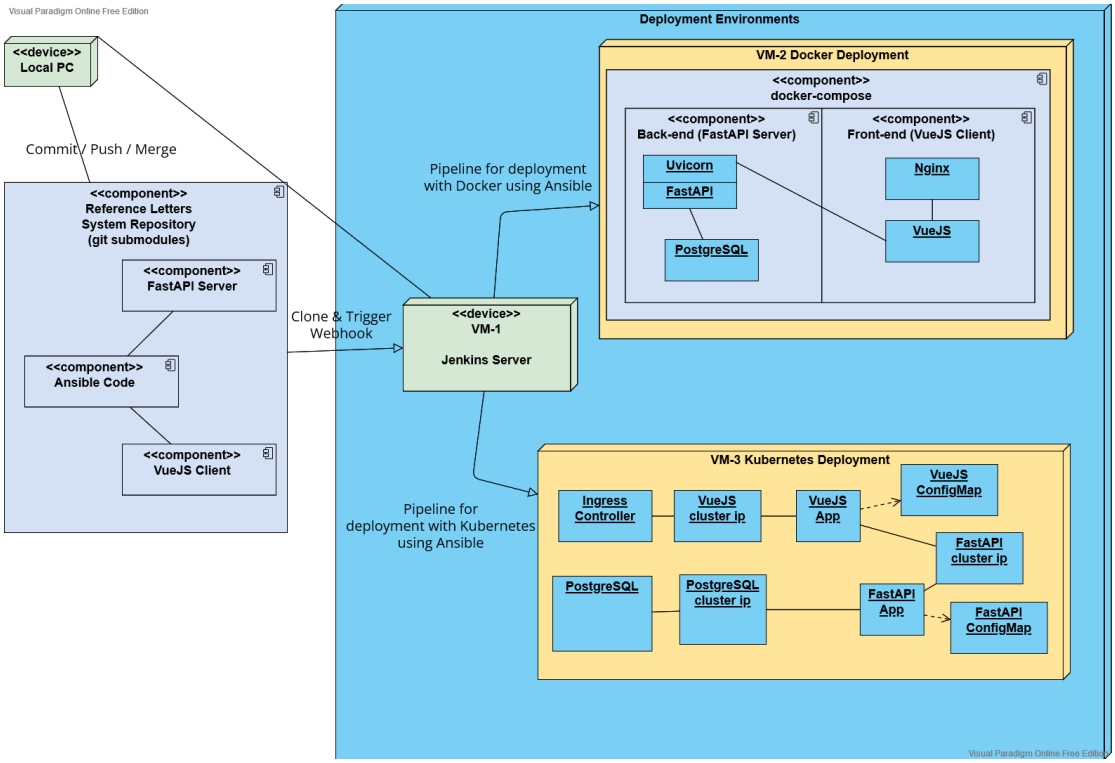
- HeaderPage: Η επικεφαλίδα κάθε σελίδας της εφαρμογής όπου δίνει πληροφορίες για την κατάσταση του χρήστη και λειτουργίες σύνδεσης / εγγραφής χρήστη.
- AboutView (/about): Πληροφορίες Εφαρμογής – Documentation
- AddReferenceLetterRequest: Δημιουργία συστατικής επιστολής
- AddStudent: Δημιουργία φοιτητή
- HomeView (/home): Αρχική σελίδα που περιέχει τα δεδομένα του συστήματος ανάλογα με την κατάσταση του χρήστη
- LoginView (/login): Σύνδεση του χρήστη
- OwnerDetails: Τα στοιχεία μας προς ενημέρωση του χρήστη.
- ReferenceLetterRequestsDetails (/reference-letter-request-details/:id): Λεπτομέρειες μιας αίτησης συστατικής επιστολής.
- ReferenceLetterRequestList: Λίστα με τις αιτήσεις συστατικών επιστολών
- SignUpLoginView (/signup): Επιλογή τύπου χρήστη
- SignUpStudentView (/signup/student): Εγγραφή νέου φοιτητή
- SignUpTeacherView (/signup/teacher): Εγγραφή νέου καθηγητή
- VerifyEmailToken (/verify_token/:token): Ανάκτηση token για επιβεβαίωση email.

4.5 Εργαλεία DevOps

Στο σύστημα μας και στις εφαρμογές μας συμπεριλαμβάνονται κάποια αρχεία που είναι χρήσιμα στην παραμετροποίηση με τα DevOps εργαλεία που θα αναλυθούν στη συνέχεια.

Πρόκειται είτε για αρχεία Jenkinsfile τα οποία αφορούν την ενσωμάτωση του κώδικα στον Jenkins server και τον τρόπο που θα γίνει το deployment της εφαρμογής μέσω αυτού σε κάποια εικονική μηχανή είτε για τα αρχεία docker-compose.yml και Dockerfile που περιγράφουν τον τρόπο και τις προϋποθέσεις ώστε να γίνει deploy το σύστημα σε εικονική μηχανή με τη χρήση του Docker, είτε για το φάκελο k8s που βρίσκεται σε κάθε εφαρμογή μας και περιέχει αρχεία σχετικά με τη διαδικασία του deployment σε εικονική μηχανή με τη χρήση του Kubernetes.

Το deployment όπου κάνουμε χρήση της Ansible περιγράφεται στη συνέχεια του κεφαλαίου καθώς στα πλαίσια της παρούσας πτυχιακής εργασίας δημιουργήθηκε ένα ακόμα αποθετήριο κώδικα πέρα από αυτά του βασικού συστήματος ώστε να εξυπηρετήσουμε τον σκοπό αυτό με μεγαλύτερη ευκολία. Χρησιμοποιείται για την αυτοματοποίηση της διαδικασίας του deployment με χρήση docker τεχνολογιών και όχι για το παραδοσιακό deployment σε κάποιον/κάποιους servers. Επίσης αυτοματοποιεί και τη διαδικασία του deployment με χρήση kubernetes κάνοντας χρήση του αντίστοιχου module.



Σχήμα 1 Αρχιτεκτονική Συστήματος

ΚΕΦ.5: Ενσωμάτωση του Docker και του Docker Compose

5.1 Εγκατάσταση Docker

Για να κάνουμε εγκατάσταση το Docker τρέχουμε τις παρακάτω εντολές (linux-based περιβάλλον):

```
sudo apt-get update

sudo apt-get install curl apt-transport-https ca-certificates software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

sudo apt update
apt-cache policy docker-ce
sudo apt install docker-ce

sudo systemctl status docker
```

Εικόνα 46. Εγκατάσταση Docker

5.2 Ενσωμάτωση του Docker Compose

Ένας γρήγορος και αξιόπιστος τρόπος να τρέξουμε πολλές εφαρμογές που πρέπει να επικοινωνήσουν μεταξύ τους για να έχουμε ένα διαδικτυακό σύστημα είναι να χρησιμοποιήσουμε το docker compose. Για να εγκατασταθεί τρέχουμε τις εξής εντολές:

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.28.5/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
docker-compose --version
```

Εικόνα 47. Εγκατάσταση Docker Compose

Κάνοντας εγκατάσταση σύμφωνα με τις παραπάνω εντολές θα χρειάζεται κάθε φορά που τρέχουμε docker εντολές να τις τρέχουμε σαν διαχειριστές (root χρήστες). Για να το αλλάξουμε αυτό μπορούμε να τρέξουμε τις παρακάτω δύο εντολές για να είμαστε στο docker group χρηστών.

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

Εικόνα 48. Προσθήκη χρήστη στο docker group

Προκειμένου να κάνουμε deploy το σύστημά μας στην εικονική μηχανή αξιοποιώντας το docker και το docker-compose έχουμε δημιουργήσει συνολικά τρία αρχεία, ένα στο project της frontend εφαρμογής, το Dockerfile και δύο στο project της backend εφαρμογής, το docker-compose.yml και το fastapi.nonroot.Dockerfile. Σε ένα Dockerfile ορίζεται τι χρειάζεται μία εφαρμογή για να τρέξει σε περιβάλλον docker μέσα από βήματα που εκτελούνται σειριακά. Συγκεκριμένα, στα Dockerfile του συστήματός μας ορίζουμε:

1. μία έκδοση της γλώσσας προγραμματισμού που χρησιμοποιούμε για να γίνει install (backend: python, frontend: node για javascript)
2. ένα working directory
3. αλλαγή σε non root χρήστη για τα περισσότερα από τα επόμενα βήματα ώστε να έχουμε ενός είδους ασφάλειας (εάν χρειάζεται και το επιθυμούμε - στο frontend δε θεωρούμε πως χρειάζεται)
4. τον κατάλογο που πρέπει να γίνει copy και περιέχει τα αρχεία που θέλουμε
5. τα requirements-dependencies (βιβλιοθήκες-εξαρτήσεις) που πρέπει να γίνουν install (backend: fastapi κλπ, frontend: vuejs κλπ)
6. την port που θέλουμε να γίνει expose η εφαρμογή, για να είναι ορατή από άλλες εφαρμογές ή μέσω της public ip της εικονικής μηχανής
7. την εντολή όταν ξεκινήσει το container για να γίνει expose αξιοποιώντας application server (π.χ. uvicorn) ή web server (π.χ. nginx)

```
# Use an existing docker image as a base
FROM python

# Change working directory to /usr/data. This is where we'll put the requirements.txt file and the app directory.
WORKDIR /usr/data

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

RUN groupadd appuser &&  useradd -g appuser -d  /usr/data -M appuser

ENV PATH=$PATH:/usr/data/.local/bin

# Copy the file with the requirements
COPY ./requirements.txt ./

# Copy main.py file
COPY ./ref_letters app

# Install the package dependencies in the requirements file.
RUN pip install -r requirements.txt

# change permission on workdir
RUN chown -R appuser:appuser /usr/data

USER appuser:appuser

EXPOSE 8000/tcp
# Tell what to do when it starts as a container
CMD ["uvicorn","app.main:app","--host","0.0.0.0","--port","8000"]
```

Εικόνα 49. non-root Dockerfile της backend εφαρμογής

```
# build stage
FROM node:lts-alpine as build-stage
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# production stage
FROM nginx:stable-alpine as production-stage
COPY --from=build-stage /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Εικόνα 50. Dockerfile της frontend εφαρμογής

Στο αρχείο docker-compose.yml ορίζουμε κάποια services τα οποία χρειάζεται το σύστημα για να εκτελεστεί και συγκεκριμένα τη βάση δεδομένων με τις παραμέτρους σύνδεσης, την υπηρεσία αυθεντικοποίησης χρηστών με τις παραμέτρους σύνδεσης και τις βασικές εφαρμογές διαχείρισης αιτήσεων συστατικών επιστολών έχοντας ορίσει τα αντίστοιχα Dockerfile που πρέπει να διαβαστεί, τα αρχεία με τις μεταβλητές περιβάλλοντος, volumes που χρειαζόμαστε για αποθήκευση δεδομένων, τα ports όπου θα τρέχουν οι βασικές εφαρμογές μας.

Οπότε, τώρα έχοντας ένα docker-compose.yml (ή .yaml) αρχείο κάπου (συνήθως στο root folder της εφαρμογής μας) μπορούμε να τρέξουμε `docker-compose up --build` (το --build δηλώνει πως τα images που φτιάχνουμε εμείς πρέπει να γίνουν build από την αρχή για εντοπισμό τυχόν αλλαγών που υπάρχουν στον κώδικα). Σε αυτή τη φάση, θα βλέπετε και όλα τα logs αυτών των components του συστήματος που έχουμε δηλώσει εφόσον έχουν πάει όλα καλά.

Να αναφερθεί πως στην παρούσα πτυχιακή εργασία χρησιμοποιείται το github ως αποθετήριο κώδικα και εκεί αποθηκεύεται ο κώδικάς μας. Το docker-compose.yml το έχουμε στο γενικό repository του συστήματος, όπου (με τη χρήση των git submodules [9]) έχουμε έναν υποφάκελο που αναφέρεται στο repository της backend εφαρμογής μας και έναν υποφάκελο που αναφέρεται στο repository της frontend εφαρμογής μας. Μέσα στο docker-compose.yml το context δηλώνεται στη backend εφαρμογή ως reference-letters-fastapi-server, δηλαδή ο φάκελος όπου αναφέρεται στο repository της backend εφαρμογής και στη frontend εφαρμογή δηλώνεται ως reference-letters-vuejs-client, δηλαδή ο φάκελος όπου αναφέρεται στο repository της frontend εφαρμογής που περιέχει τον αντίστοιχο κώδικα που χρειάζεται το σύστημα.

```yaml
version: '3'

services:

  database:
    container_name: database
    image: postgres:14
    expose:
      - 5432
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - FASTAPI_DB_USER=bellias
      - FASTAPI_DB_PASS=pass123
      - FASTAPI_DB_NAME=reference_letters_data
    healthcheck:
      test: [ "CMD", "pg_isready", "-q", "-d", "postgres", "-U", "postgres" ]
      timeout: 30s
      interval: 30s
      retries: 3
    volumes:
      - 'dj_postgres_data:/var/lib/postgresql/data/'
      - './assets/init_db:/docker-entrypoint-initdb.d/'
    restart:
      always
```

```yaml
  backend:
    container_name: backend
    build:
      context: reference-letters-fastapi-server
      dockerfile: nonroot.Dockerfile
    image: ghcr.io/panagiotis-bellias-it21871/ref-letters-fastapi-server
    command: uvicorn app.main:app --reload --workers 1 --host 0.0.0.0 --port 8000
    ports:
      - "8000:8000"
    volumes:
      - ./reference-letters-fastapi-server/ref_letters:/usr/data/app
    env_file:
      - reference-letters-fastapi-server/ref_letters/.env
    healthcheck:
      test:
        - CMD
        - wget -S --spider http://localhost:8000/docs
      interval: 30s
      timeout: 30s
      retries: 3
    depends_on:
      - database
    restart:
      always

  frontend:
    container_name: frontend
    build:
      context: reference-letters-vuejs-client
      dockerfile: Dockerfile
    image: ghcr.io/panagiotis-bellias-it21871/ref-letters-vuejs-client
    volumes:
      - './reference-letters-vuejs-client:/app'
      - '/app/node_modules'
    env_file:
      - reference-letters-vuejs-client/.env
    ports:
      - "80:80"
      # - "443:443"
    depends_on:
      - backend

volumes:
  dj_postgres_data:
```

Εικόνα 51. docker compose

5.3 Github Container Registry

Χρησιμοποιώντας το container registry του Github, μπορούμε να αποθηκεύσουμε και να διαχειριστούμε Docker images. Χρησιμοποιεί το package namespace https://ghcr.io

Για έλεγχο ταυτότητας στο container registry σε workflow ενός CI/CD server, χρησιμοποιήστε το GITHUB_TOKEN για την καλύτερη ασφάλεια και εμπειρία.

1. Δημιουργήστε ένα νέο προσωπικό access token (PAT) με τα κατάλληλα πεδία για τα tasks που θέλετε να ολοκληρώσετε. Εάν ο οργανισμός σας απαιτεί SSO, πρέπει να ενεργοποιήσετε το SSO για το νέο σας token.
    a. Επιλέξτε το πεδίο read:packages για λήψη container images και ανάγνωση των μεταδεδομένων τους.
    b. Επιλέξτε το πεδίο write:packages για λήψη και αποστολή container images και ανάγνωση και εγγραφή των μεταδεδομένων τους.
    c. Επιλέξτε το πεδίο delete:packages για να διαγράψετε container images.

    Για περισσότερες πληροφορίες, βλέπε εδώ [10]

2. Αποθηκεύστε το PAT σας. Συνίσταται να αποθηκεύσετε το PAT σας ως μεταβλητή περιβάλλοντος.

    ```
    export CR_PAT=YOUR_TOKEN
    ```

3. Χρησιμοποιώντας το CLI για τον τύπο του container σας, συνδεθείτε στην υπηρεσία container registry στη διεύθυνση ghcr.io

```
echo $CR_PAT | docker login ghcr.io -u USERNAME --password-stdin
> Login Succeeded
```

Για να μπορούμε να κάνουμε push ένα image ας δούμε πως γίνεται σε παράδειγμα.

```
docker push ghcr.io/OWNER/IMAGE_NAME:latest
```

Ή συγκεκριμένη έκδοση του image.

```
docker push ghcr.io/OWNER/IMAGE_NAME:2.5
```

Όταν γίνεται publish για πρώτη φορά ένα package, η προεπιλεγμένη ορατότητα είναι ιδιωτική. Για να αλλάξετε την ορατότητα ή να ορίσετε δικαιώματα πρόσβασης, ανατρέξτε στην ενότητα [11].

Περισσότερα εδώ [12]

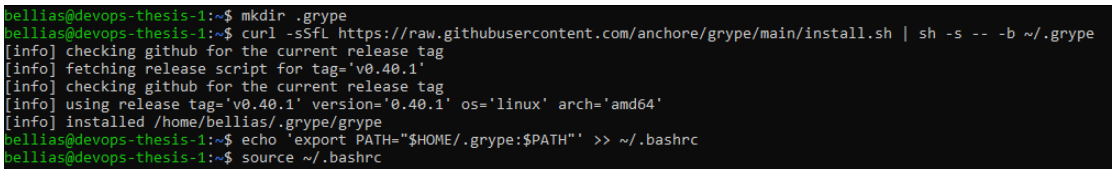
5.4 Security Scanning

Προκειμένου να ελέγξουμε τα docker images που φτιάχνουμε για τυχόν ευπάθειες μπορούμε να χρησιμοποιήσουμε το σαρωτή ευπαθειών grype που μπορεί να εφαρμοστεί και σε συστήματα αρχείων (file systems).

Εικόνα 52. Λογότυπο Grype

Τα χαρακτηριστικά του Grype είναι τα εξής:

- Σαρώνει τα περιεχόμενα ενός container image ή ενός συστήματος αρχείων για να βρείτε γνωστά τρωτά σημεία.
- Βρίσκει ευπάθειες για μεγάλα πακέτα λειτουργικών συστημάτων, όπως είναι το Alpine, το BusyBox, το CentOS, το Debian, τα Oracle Linux, τα Ubuntu
- Βρίσκει τρωτά σημεία για πακέτα ειδικά για κάποια γλώσσα προγραμματισμού όπως Java (JAR, WAR, EAR, JPI, HPI), JavaScript (NPM, Yarn) και Python (Egg, Wheel, Poetry, requirements.txt/setup.py files)
- Υποστηρίζει docker images



Εικόνα 53. Εγκατάσταση Grype

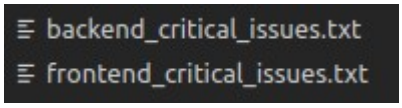
Με σκοπό να κατεβάσουμε το grype και να το δοκιμάσουμε τρέχουμε τις εντολές που βλέπετε στην εικόνα (περιβάλλον linux-based) βάζοντας το δικό σας path προορισμού όπου θέλετε να αποθηκευτεί το grype (ή που δημιουργήσατε για τον σκοπό αυτό, όπως φαίνεται στην εικόνα).

Το grype θα το χρησιμοποιούμε όταν θα θέλουμε να τρέξουμε το σύστημα σε Docker περιβάλλον. Θα δοκιμάζουμε να κάνουμε build τα docker images μας και αμέσως θα τρέχουμε το grype για να βλέπουμε τα ευαίσθητα σημεία του container. Αν όλα είναι εντάξει και δεν έχουμε κρίσιμα ή υψηλής σημασίας τρωτά σημεία συνεχίζουμε με το είδος του deployment που θέλουμε να κάνουμε. Αυτό μπορεί να είναι είτε να σηκώσουμε το σύστημα με docker-compose είτε να κάνουμε push τα images για να τα χρησιμοποιήσουμε σε ένα kubernetes cluster.

Για να δούμε πόσα και τι σοβαρότητας είναι τα τρωτά σημεία ενός container μπορούμε να χρησιμοποιήσουμε τις παρακάτω εντολές και να δημιουργήσουμε αρχεία όπου θα κρατάνε την πληροφορία αυτή ανά image και ανά σοβαρότητα.

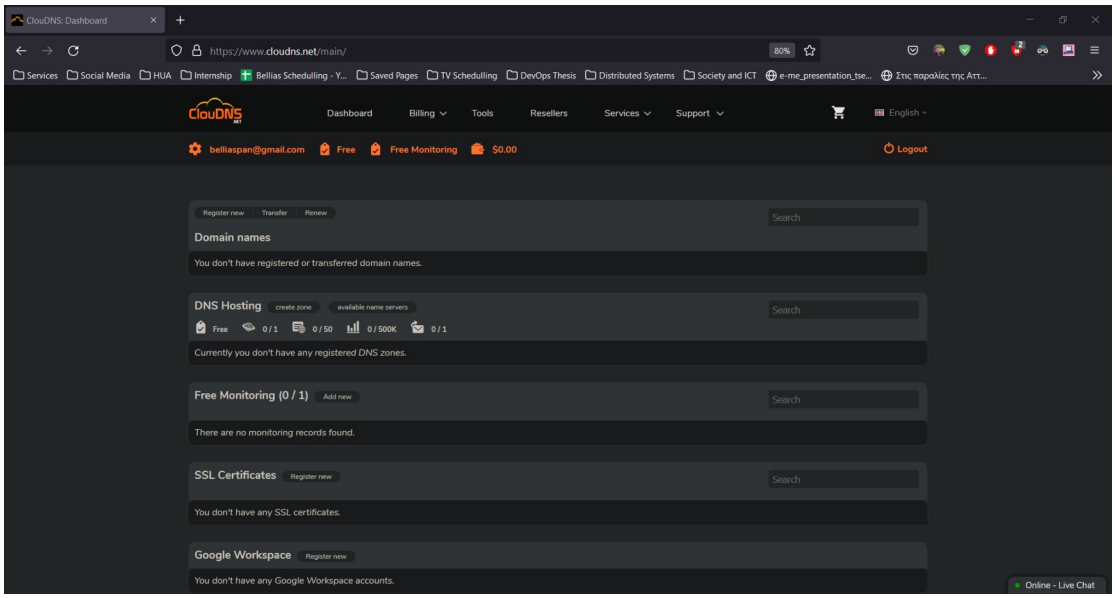
```
grype ghcr.io/panagiotis-bellias-it21871/ref-letters-fastapi-server > backend_image_grype_logs.txt
grype ghcr.io/panagiotis-bellias-it21871/ref-letters-vuejs-client > frontend_image_grype_logs.txt
cat backend_image_grype_logs.txt | grep 'Critical' > backend_critical_issues.txt
cat frontend_image_grype_logs.txt | grep 'Critical' > frontend_critical_issues.txt
```

Έτσι, βλέπουμε παρακάτω τα αντίστοιχα αρχεία που περιέχουν critical, δηλαδή κρίσιμης σημασίας, ευπάθειες για κάθε component που έχουμε, δηλαδή τη backend εφαρμογή μας και τη frontend εφαρμογή μας.

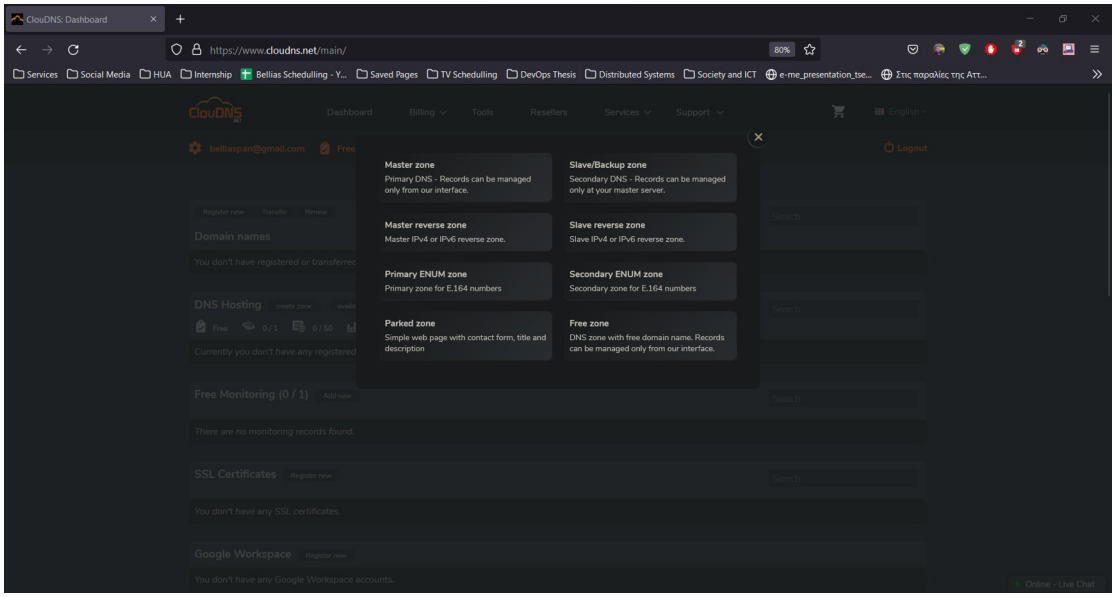


Εικόνα 54. Αρχεία Ανίχνευσης Ευπαθειών Docker Images

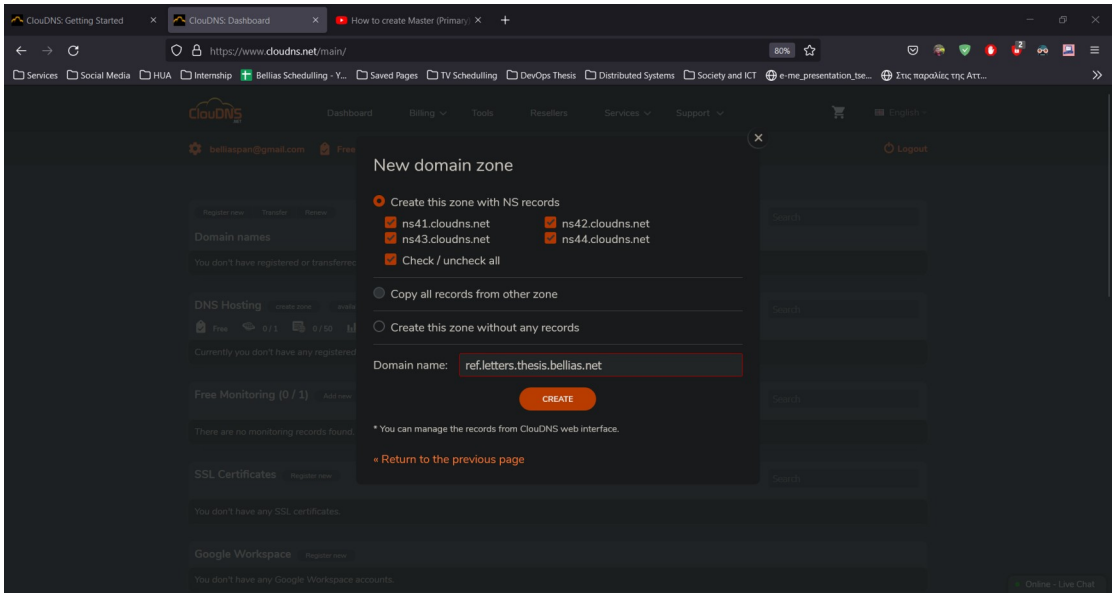
5.5 Ρύθμιση Domain Name & Εγκατάσταση SSL Certificates - HTTPS Περιβάλλον



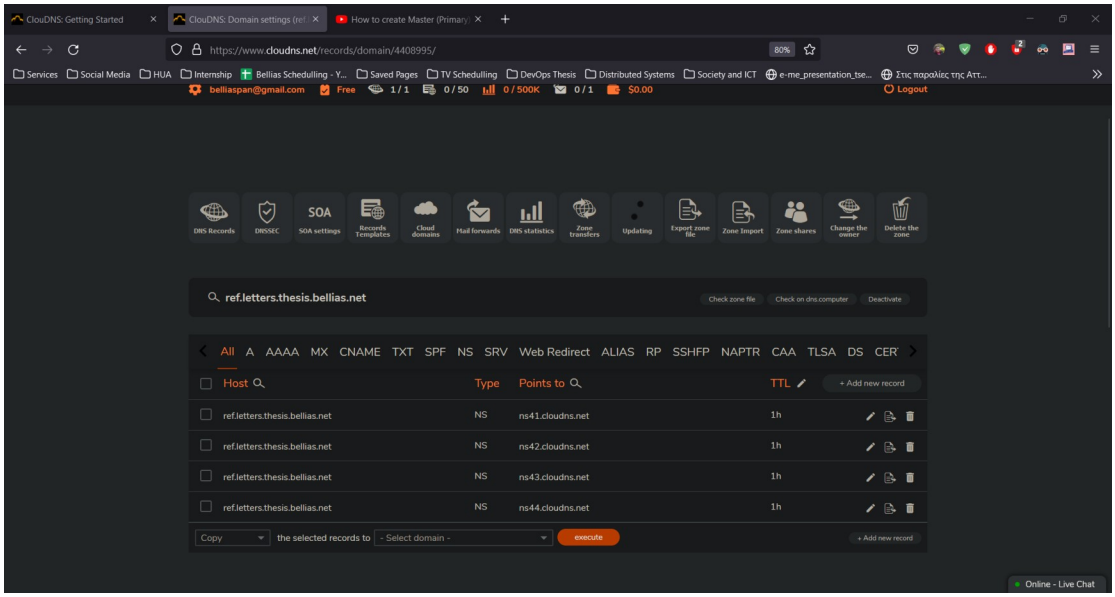
Εικόνα 55. Αρχική Σελίδα ClouDNS



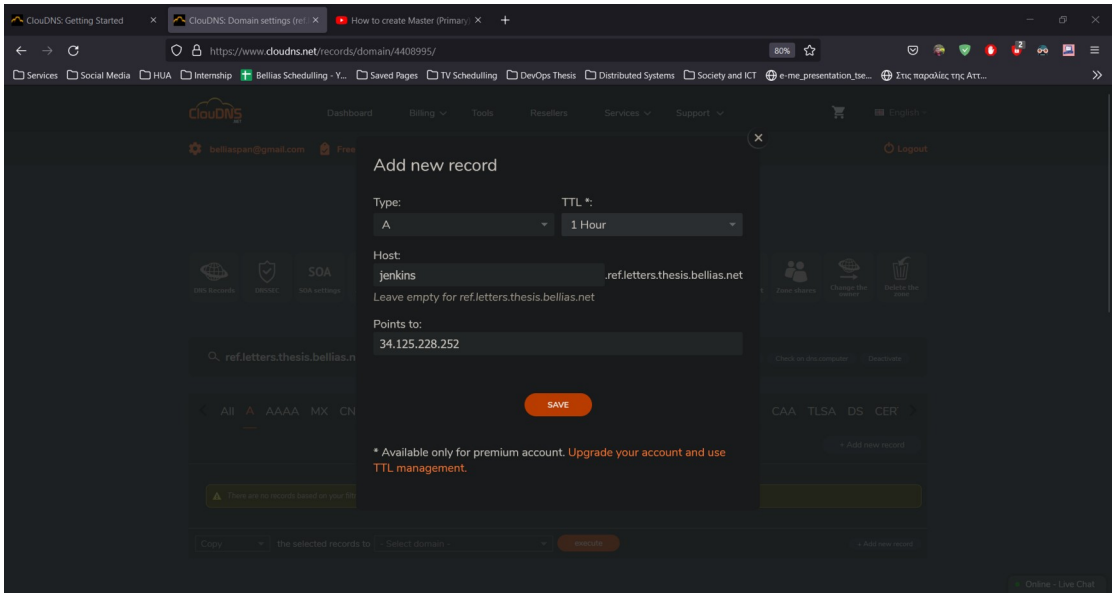
Εικόνα 56. Επιλογή DNS Zone



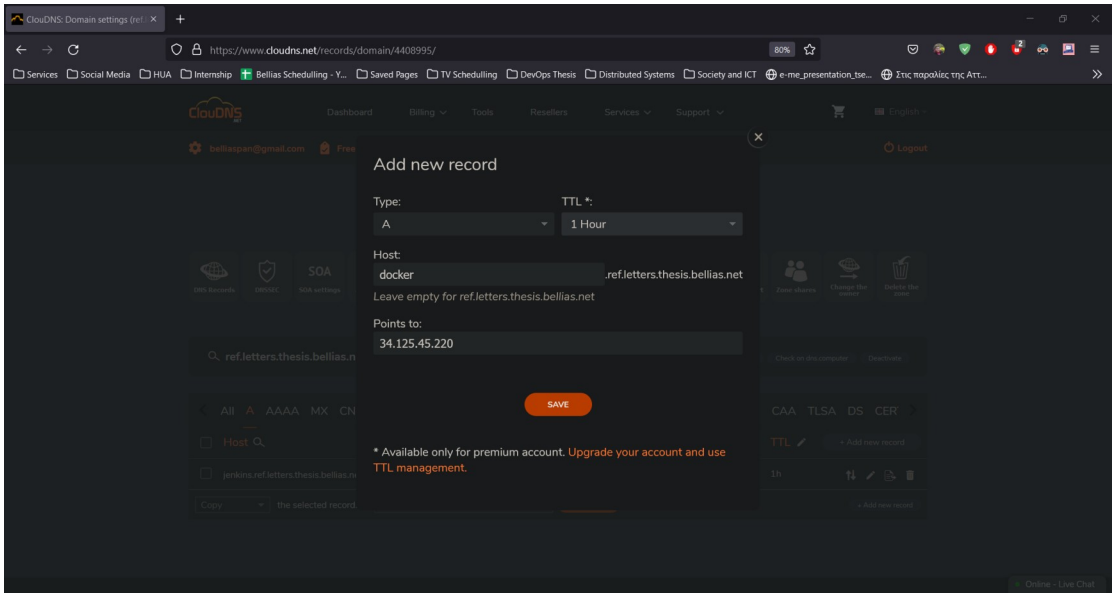
Εικόνα 57. Επιλογή Ονόματος για το DNS Zone



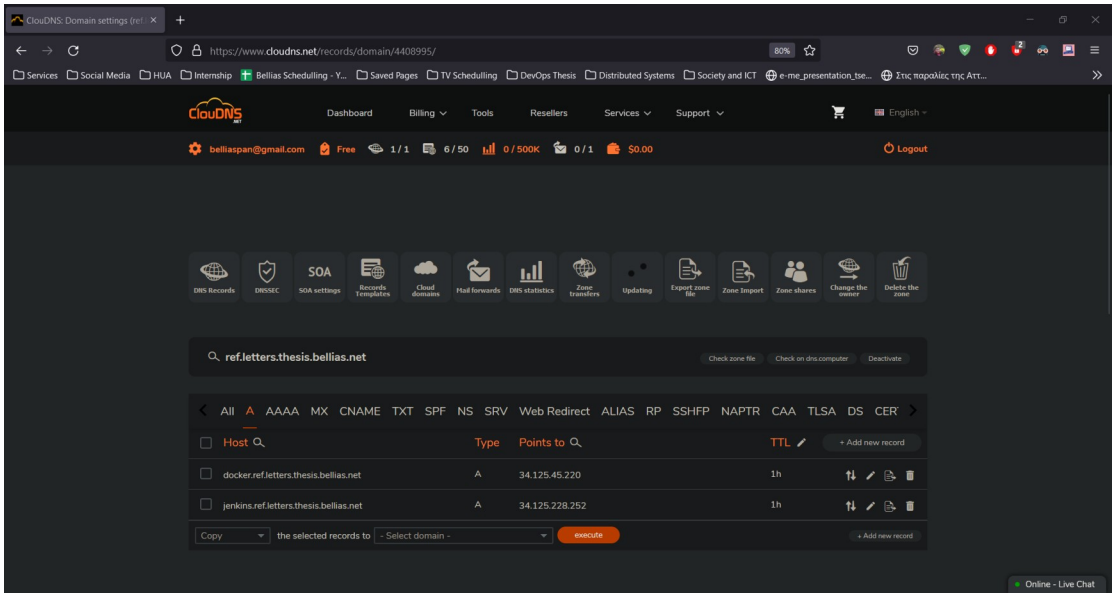
Εικόνα 58. Επιτυχής Δημιουργία DNS Zone



Εικόνα 59. Δημιουργία A Record για Jenkins VM



Εικόνα 60. Δημιουργία A Record για Docker VM

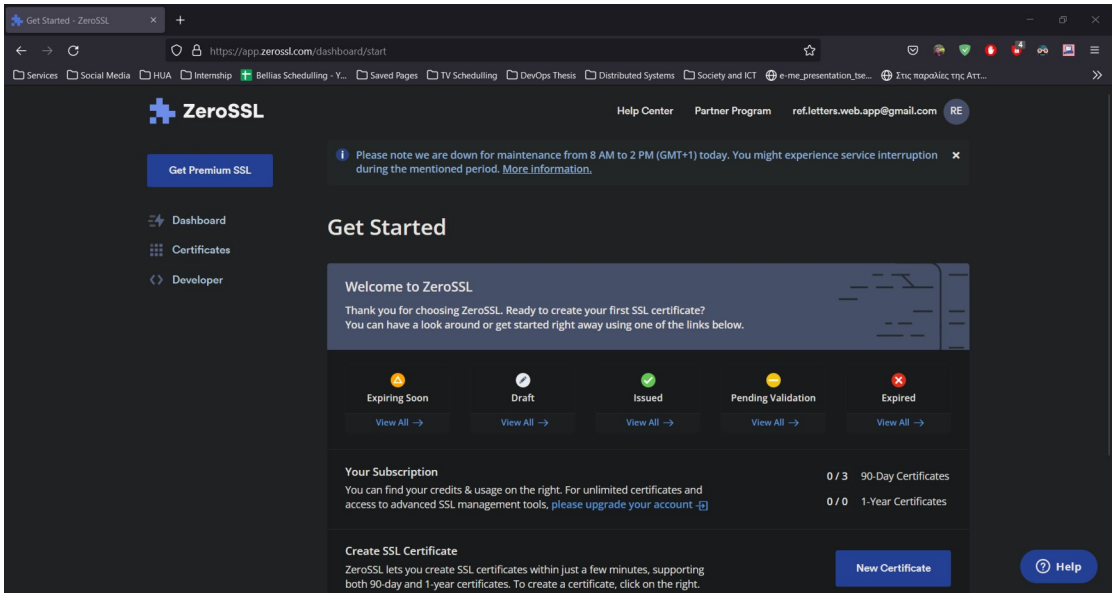


Εικόνα 61. Επιτυχής Δημιουργία A Record

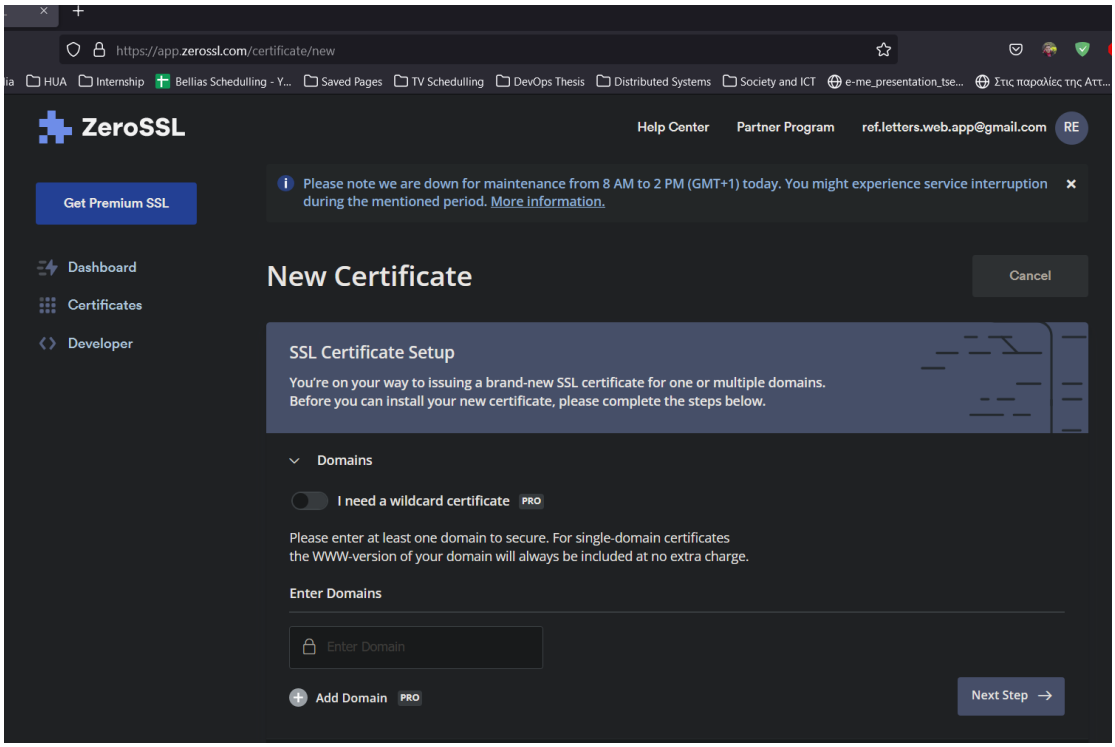
Στις παραπάνω εικόνες βλέπουμε πώς μπορούμε να φτιάξουμε DNS records για τα VM μας, τα οποία θα φιλοξενήσουν την εφαρμογή μας με διαφορετικό τρόπο το καθένα. Σε επόμενο κεφάλαιο θα χρησιμοποιήσουμε μια εναλλακτική και γρηγορότερη υπηρεσία DNS με στόχο να δημιουργήσουμε domain name για το VM που χρησιμοποιεί Kubernetes Cluster. Οπότε τα domain name που έχουν ρυθμιστεί είναι:

- jenkins.ref.letters.thesis.bellias.net → για το VM που έχει τον jenkins server
- docker.ref.letters.thesis.bellias.net → για το VM που χρησιμοποιεί docker

Όσον αφορά τη δημιουργία https περιβάλλοντος μπορούμε να επισκεφθούμε το https://zerossl.com/ και να φτιάξουμε εκεί λογαριασμό.

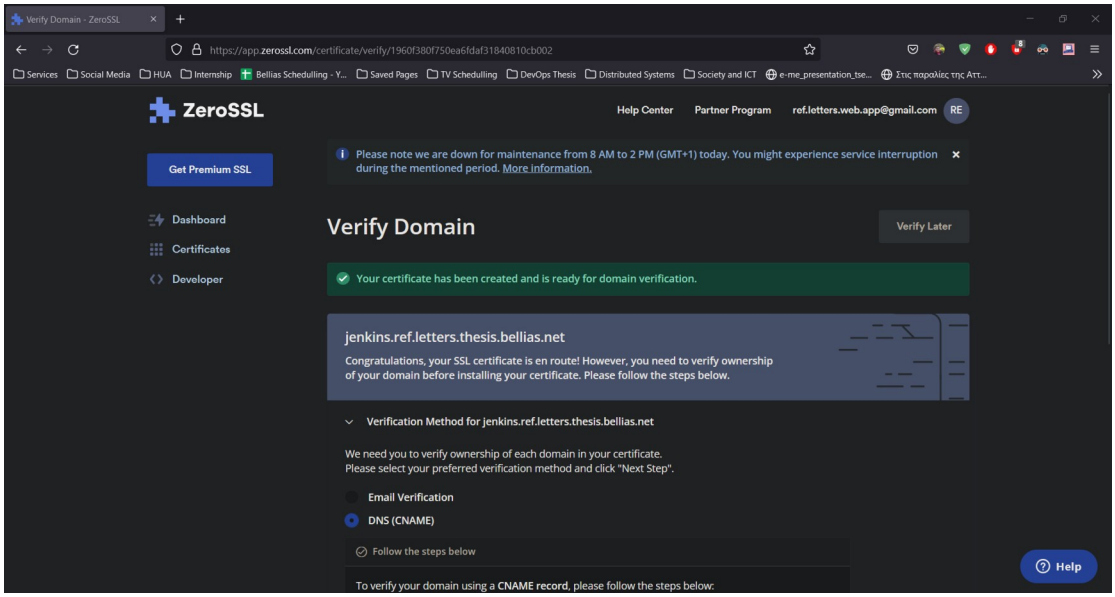


Εικόνα 62. Αρχική Σελίδα FreeSSL



Εικόνα 63. Δημιουργία SSL Certificate

Χρειάζεται επίσης να επαληθεύσουμε το domain name πηγαίνοντας στο cloudns.cl και προσθέτοντας το CNAME record σύμφωνα με τις οδηγίες. Μετά απλά κατεβάζουμε τα πιστοποιητικά για να τα ανεβάσουμε στον server μας.



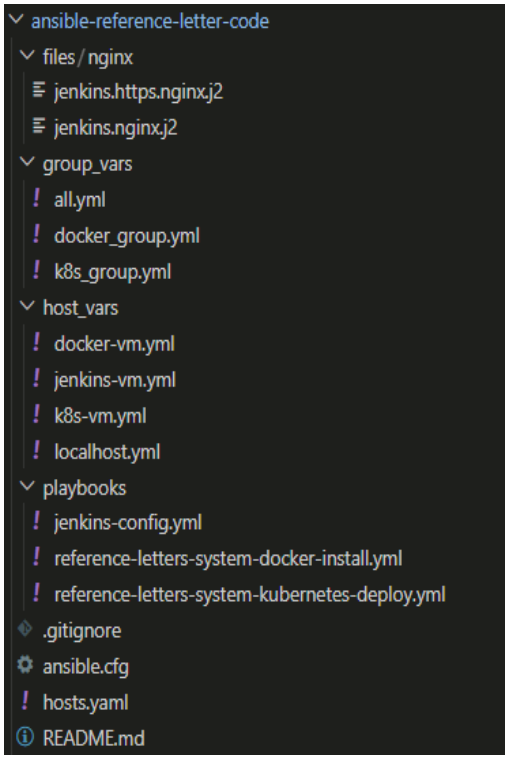
Εικόνα 64. Επαλήθευση Domain μέσω DNS (CNAME)

ΚΕΦ.6: Ενσωμάτωση της Ansible

6.1 Ansible Repository

Σκοπός μας στην παρούσα πτυχιακή εργασία είναι η δυνατότητα αυτοματοποίησης, ανάπτυξης και εκτέλεσης του συστήματος διαχείρισης συστατικών επιστολών σε διαφορετικά περιβάλλοντα (βλ. εικονικές μηχανές, οι εικονικές μηχανές που χρησιμοποιήθηκαν είναι από τον πάροχο Microsoft). Αυτό καθίσταται εφικτό αξιοποιώντας την Ansible.

Για την επίτευξη αυτού του σκοπού δημιουργήθηκε και παραμετροποιήθηκε ένα νέο project (Ansible-Reference-Letter-Code). Σε αυτό το project έχουν οριστεί μεταβλητές περιβάλλοντος (π.χ. host, στοιχεία σύνδεσης με βάση δεδομένων κ.ά.), και κυρίως τα playbooks όπου περιέχονται τα βήματα που πρέπει να γίνουν ώστε να εγκατασταθεί και να εκτελεστεί κάθε μία εφαρμογή ξεχωριστά αλλά και όλο το σύστημα μαζί σε μια εικονική μηχανή.



Εικόνα 65. Δομή Ansible Project

6.2 Σύνδεση SSH

Προκειμένου να συνδεθούμε στα VM που έχουμε για να γίνουν τα deployments του διαδικτυακού συστήματος ρυθμίζουμε κατάλληλα το ~/.ssh/config αρχείο που έχουμε τοπικά στο μηχάνημά μας. Έπειτα απλά μπορούμε να αναφερόμαστε στα VM μας μέσα από το hosts.yaml αρχείο με τα ονόματα που έχουμε δώσει στο κάθε VM.

```
Host jenkins-vm
        HostName <DNS-RECORD-OR-PUBLIC-IP>
        User belliaspan
        IdentityFile ~/.ssh/id_rsa_jenkins

Host docker-vm
        HostName <DNS-RECORD-OR-PUBLIC-IP>
        User belliaspan
        IdentityFile ~/.ssh/id_rsa_docker

Host k8s-vm
        HostName <DNS-RECORD-OR-PUBLIC-IP>
        User belliaspan
        IdentityFile ~/.ssh/id_rsa_k8s
```

Εικόνα 66. Ενδεικτικό ~/.ssh/config αρχείο

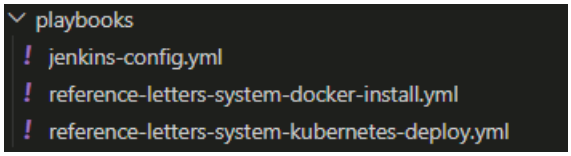
```
# You first must have the corresponding ~/.ssh/config with the names for the hosts below
# matching these in that file referring to the Host
jenkins_server:
    hosts:
        jenkins-vm

docker_group:
    hosts:
        docker-vm

k8s_group:
    hosts:
        k8s-vm
```

Εικόνα 67. Ενδεικτικό hosts.yaml αρχείο

6.3 Playbooks



Εικόνα 68. Ansible playbooks

Εδώ φαίνονται τα playbooks που χρησιμοποιούνται για να γίνουν τα deployments του συστήματος.

Αναλυτικότερα:

- jenkins-config.yml: Χρησιμοποιείται για να παραμετροποιήσει τον Jenkins server ώστε να τρέχει μέσω nginx web server και σε https περιβάλλον.
- reference-letters-system-docker-install.yml: Χρησιμοποιείται για να κατεβάσει από το github τον κώδικα του συστήματος, να ρυθμίσει τις μεταβλητές περιβάλλοντος και να εγκαταστήσει το σύστημα σε περιβάλλον docker με χρήση docker-compose, με nginx παραμετροποίηση και ssl certificates για να τρέχει με https.
- reference-letters-system-kubernetes-deploy.yml: Χρησιμοποιείται για να κατεβάσει από το github τα αρχεία που θα χρησιμοποιήσει το kubernetes API, να ρυθμίσει τις μεταβλητές περιβάλλοντος και να εγκαταστήσει το σύστημα σε kubernetes cluster με χρήση των docker images που είναι διαθέσιμα στο github container registry.

ΚΕΦ.7: Ενσωμάτωση του Jenkins Server

7.1 Jobs - Pipelines

Προκειμένου να επιτύχουμε συνεχή ενσωμάτωση και συνεχή παράδοση του διαδικτυακού μας συστήματος στα περιβάλλοντα παραγωγής που έχουμε διαθέσιμα χρησιμοποιούμε τον Jenkins Server, όπου αναφέρθηκε και προηγουμένως.

Είναι ένα εργαλείο αυτοματοποίησης το οποίο υποστηρίζει την τεχνική του CI/CD, δηλαδή τη συνεχή ενσωμάτωση των αλλαγών που γίνονται στον κώδικα πραγματοποιώντας build και tests και την προώθηση τους σε περιβάλλον production. Ο Jenkins server, λοιπόν λειτουργεί σαν συνδετικός κρίκος ανάμεσα στο αποθετήριο του κώδικα μας και στα περιβάλλοντα που θα γίνει deploy η εφαρμογή. Εγκαθίσταται σε μία εικονική μηχανή και εφόσον έχουν παραμετροποιηθεί σωστά τα credentials σύνδεσης τόσο με τα αποθετήρια κώδικα όσο και με τις εικονικές μηχανές τότε μπορεί να υπηρετήσει αποτελεσματικά το ρόλο για τον οποίο χρησιμοποιείται.

Απαραίτητη προϋπόθεση είναι για κάθε περιβάλλον που θέλουμε να κάνουμε την εφαρμογή μας deploy, να έχουμε το αντιστοιχο Jenkinsfile. Έτσι θα μπορεί ο Jenkins (δηλώνοντας το όνομα του Jenkinsfile σε ένα pipeline) να βρίσκει το αρχείο με τις οδηγίες εγκατάστασης.

Φυσικά αναγκαία είναι και η εγκατάστασή του σε κάποιο φυσικό server ή σε κάποια ιδεατή μηχανή (virtual machine) ώστε να είναι διαθέσιμος μέσα από μια public ip που θα χρειαστούμε για να ρυθμίσουμε την επικοινωνία του με το github repository.

Περισσότερα σχετικά με την εγκατάσταση και την παραμετροποίηση ενός Jenkins Server θα βρείτε εδώ. [13]

7.2 Διαχείριση Μεταβλητών Περιβάλλοντος για Παραμετροποίηση

Υπάρχουν τιμές, όπως έχουμε ήδη πει, που δε θέλουμε να υπάρχουν μέσα στον κώδικα και να είναι ορατές από οποιονδήποτε έχει πρόσβαση σε αυτόν (π.χ. public repository). Αυτές τις τιμές τις περνάμε σε μεταβλητές περιβάλλοντος και τις χρησιμοποιούμε παραμετρικά. Αυτό αντίστοιχα γίνεται και σε ένα CI/CD εργαλείο όπως είναι ο jenkins server.

Ο Jenkins παρέχει, μαζί με άλλα, τα λεγόμενα secret texts όπου σε κάθε ένα που φτιάχνουμε αποθηκεύουμε το όνομα της μεταβλητής και την τιμή της. Πρόσβαση έχουμε μέσα από το pipeline μας, όπου στην περίπτωσή μας είναι το Jenkinsfile, το οποίο κάνει install όλο το διαδικτυακό σύστημά μας. Επίσης υπάρχει αντίστοιχος τρόπος να αποθηκεύουμε ssh κλειδιά για σύνδεση σε VM ή server όπου θέλουμε να κάνουμε deploy το κώδικά μας ή να τρέξουμε κάποιο ansible playbook.

Στο README.md αρχείο του github repository μπορείτε να βρείτε πίνακα με τις μεταβλητές που πρέπει να γίνουν configure για να γίνεται deploy το σύστημά μας μαζί με ενδεικτικές τιμές για αυτές.

ΚΕΦ.8: Ενσωμάτωση του Kubernetes

8.1 Σύνδεση με Kubernetes Cluster

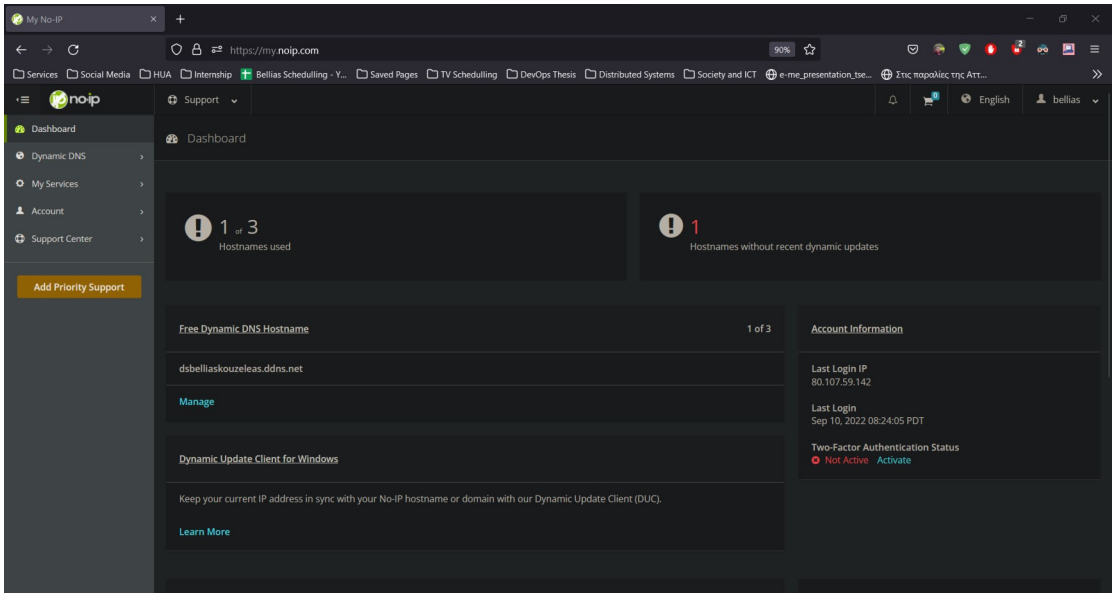
Προκειμένου να εγκαταστήσουμε το project μας σε Kubernetes Cluster, πρέπει πρώτα να συνδεθούμε στο VM όπου αυτό τρέχει έτσι ώστε να ρυθμίσουμε τη σύνδεση μεταξύ αυτού και του τοπικού υπολογιστή μας ή του Jenkins server όπου χρησιμοποιούμε.

Πληροφορίες σχετικά με την εγκατάσταση ενός δοκιμαστικού kubernetes cluster, του microk8s, βρίσκετε εδώ. [14]

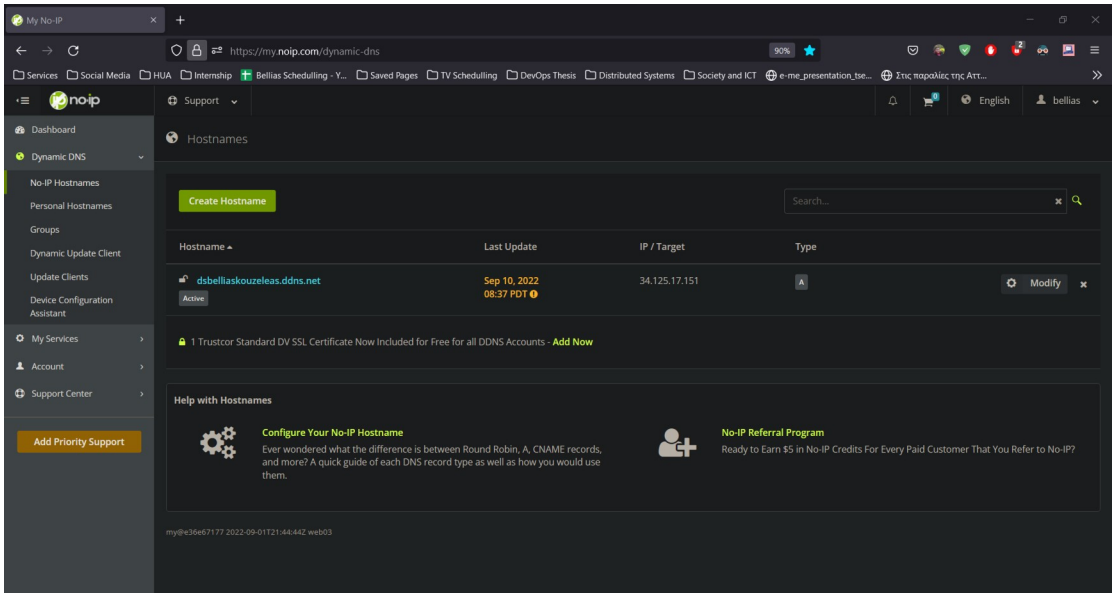
Για τη δική μας ευκολία, μπορούμε να προσθέσουμε ένα συνώνυμο εντολής (alias) στο φλοιό μας ώστε να μη γράφουμε μεγάλες εντολές συνέχεια. Για να χρησιμοποιήσουμε το CLI εργαλείο που περιέχεται στο microk8s, το kubectl μπορούμε να κάνουμε `echo "alias k='microk8s kubectl'" >> ~/.profile` και να επανασυνδεθούμε στο VM για να εφαρμοστούν οι αλλαγές στο φλοιό.

8.2 Ρύθμιση Domain Name

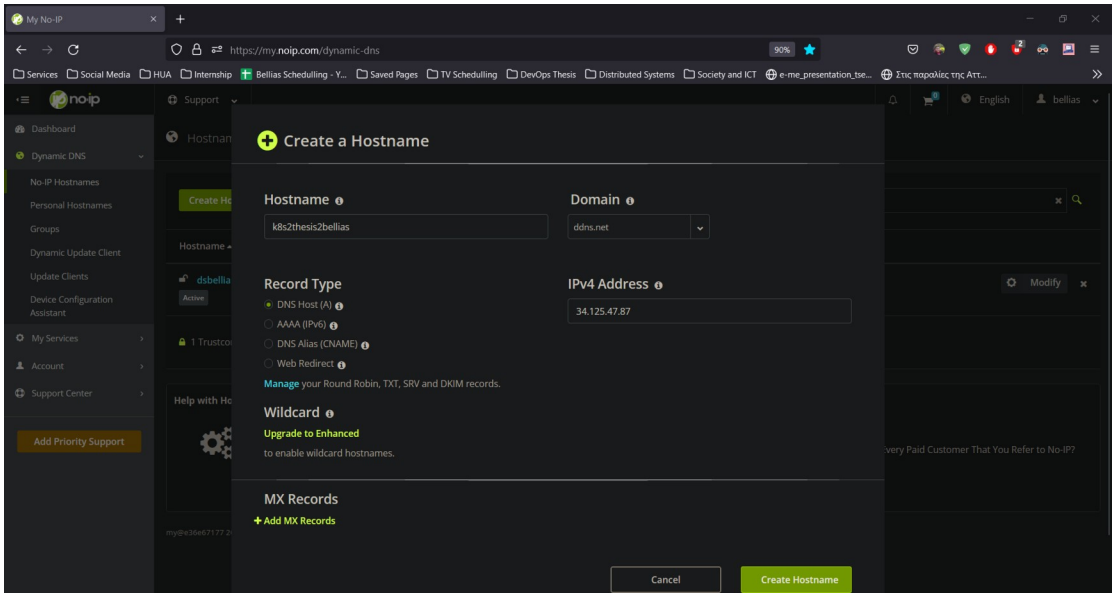
Στο περιβάλλον του kubernetes deployment θέλουμε επίσης να μπορούμε να έχουμε πρόσβαση στο user interface της εφαρμογής μας χρησιμοποιώντας domain name. Γι' αυτό χρησιμοποιούμε την online υπηρεσία No-IP όπου μπορούμε εύκολα και γρήγορα να ρυθμίσουμε ένα domain name για την IP διεύθυνση του VM μας.



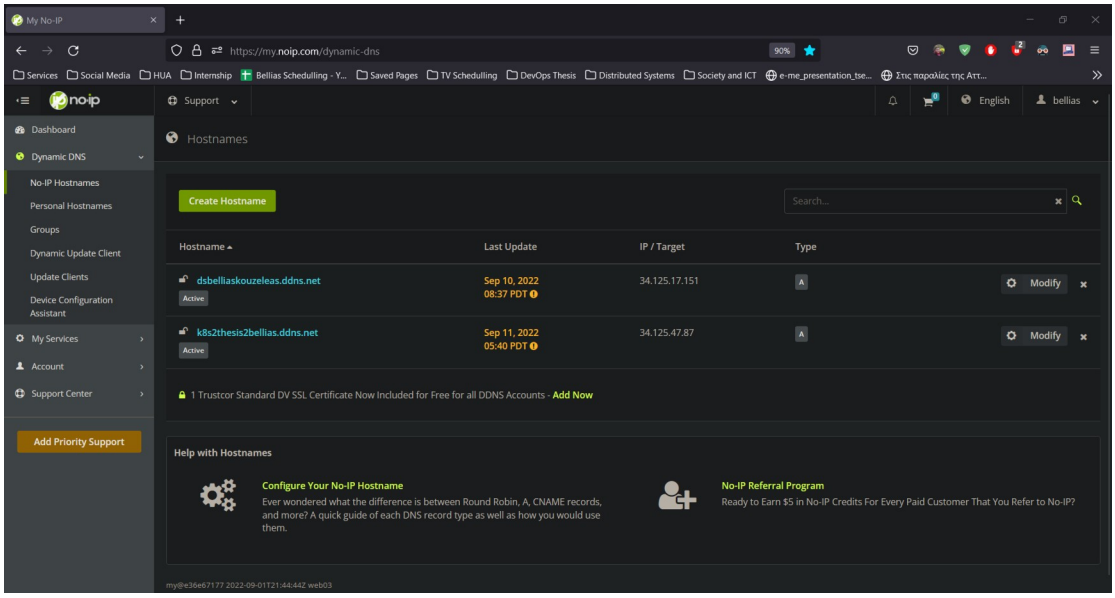
Εικόνα 69. noip.com Dashboard



Εικόνα 70. Υπάρχοντα Hostnames



Εικόνα 71. Δημιουργία Hostname στο NoIP



Εικόνα 72. Επιτυχής Δημιουργία Hostname στο NoIP

Προκειμένου να χρησιμοποιήσουμε domain name μέσω του https://noip.com/ απλά κάνουμε login ή φτιάχνουμε λογαριασμό αν δεν έχουμε ήδη και δημιουργούμε ένα νέο hostname. Οπότε για το VM μας όπου χρησιμοποιεί kubernetes cluster έχουμε ρυθμίσει ως domain name το k8s2thesis2bellias.ddns.net

8.3 Περιβάλλον Kubernetes

Για να μπορούμε να τρέξουμε εντολές kubectl πρέπει να είμαστε στο αντίστοιχο group οπότε χρειάζεται να τρέξουμε τις εξής εντολές για να προστεθούμε στο microk8s group και να ενεργοποιήσουμε κάποια plugins που χρειαζόμαστε.

```
sudo usermod -a -G microk8s $USER
sudo chown -f -R $USER ~/.kube
microk8s enable dns dashboard storage ingress
microk8s status
```

Τώρα είμαστε στον υπολογιστή μας (τοπικά) και τρέχουμε τις παρακάτω εντολές. Σε λίγο θα δούμε και την περίπτωση όπου χρησιμοποιούμε jenkins server.

```
# VM's terminal
k config view --raw > kube-config
cat kube-config

# Local terminal
mkdir ~/.kube
scp <host-name>:/home/$USER/kube-config ~/.kube/config
```

Τέλος, πρέπει να επεξεργαστούμε το αρχείο ~/.kube/config για να αντικαταστήσουμε την IP με την public IP του host ή VM μας και την γραμμή με το certificate στο block 'clusters' με τη γραμμή `insecure-skip-tls-verify: true`. Προσοχή, όχι σε πραγματικά production περιβάλλοντα όπως θα δούμε στη συνέχεια. Τώρα μπορούμε να τρέξουμε μια από τις βασικές kubectl εντολές `kubectl get pods` για να βεβαιωθούμε ότι έχουμε σύνδεση με το cluster.

Στην περίπτωση όπου είμαστε σε έναν jenkins server και θέλουμε να συνδεθούμε στο kubernetes cluster μας, τρέχουμε τις παρακάτω εντολές. Προτείνεται να κάνετε fork τα github repositories για να μπορείτε να παραμετροποιήσετε πράγματα εφόσον χρειαστεί.

```
# Jenkins terminal
sudo su
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
su jenkins
cd

# Local terminal
scp ~/.kube/config <jenkins-vm-name>:/tmp/config

# Jenkins terminal
mkdir -p .kube
cp /tmp/config ~/.kube/
```

Είτε χειροκίνητα, είτε μέσω jenkins server με χρήση Jenkinsfile και secret texts έχουμε να κάνουμε τα παρακάτω. Ο κώδικας της κάθε εφαρμογής βρίσκεται σε φάκελο με όνομα k8s οπότε θα αλλάξουμε φάκελο αρκετές φορές για να αναφερθούμε κάθε φορά στον κώδικα που χρειαζόμαστε. Ο κώδικας βρίσκεται σε μορφή yaml.

Προσοχή, πρέπει να έχετε διαθέσιμες τις αντίστοιχες docker images στο github container registry όπως είπαμε νωρίτερα καθώς οι deployment οντότητες τις χρησιμοποιούν. Μπορείτε να δείτε και το αντίστοιχο Jenkinsfile που βρίσκεται στο κεντρικό git repository. Φυσικά, πρέπει να έχετε εγκατεστημένο το docker είτε στο τοπικό σας μηχάνημα, είτε στον jenkins server. Στα README.md αρχεία των επιμέρους εφαρμογών βρίσκετε περισσότερες πληροφορίες για την docker image καθε μιας εφαρμογής. Βέβαια μεγαλύτερη απόδοση δίνει η χρήση του αντίστοιχου ansible playbook που περιγράφεται παρακάτω.

```
# Secret (for the postgresql database)
kubectl create secret generic pg-users \
--from-literal=PGUSER=<put user name here> \
--from-literal=PGPASSWORD=<put password here>


cd reference-letters-fastapi-server
```

```
# Config Map (for .env variables)
cp ref_letters/.env.k8s.example ref_letters/.env
nano ref_letters/.env # change to the correct values
kubectl create configmap fastapi-config --from-env-file=ref_letters/.env
cd k8s
```

```
# Persistent Volume Claim
kubectl apply -f db/postgres-pvc.yaml
# Deployments
kubectl apply -f db/postgres-deployment.yaml
kubectl apply -f fastapi/fastapi-deployment.yaml
# Services (Cluster IPs)
kubectl apply -f db/postgres-clip.yaml
kubectl apply -f fastapi/fastapi-clip.yaml
# Take Access inside postgres pod to initialize database
kubectl exec -it <POD-NAME> ./../assets/init_db/*

cd ../..
cd reference-letters-vuejs-client

# Config Map (for .env variables)
cp .env.k8s.example .env
nano .env # change to the correct values
kubectl create configmap vuejs-config --from-env-file=.env

cd k8s
# Deployments
kubectl apply -f vuejs/vuejs-deployment.yaml
# Services (Cluster IPs)
kubectl apply -f vuejs/vuejs-clip.yaml
# Ingress (For just HTTP - Edit file changing host to your own dns name)
kubectl apply -f vuejs/vuejs-ingress.yaml
```

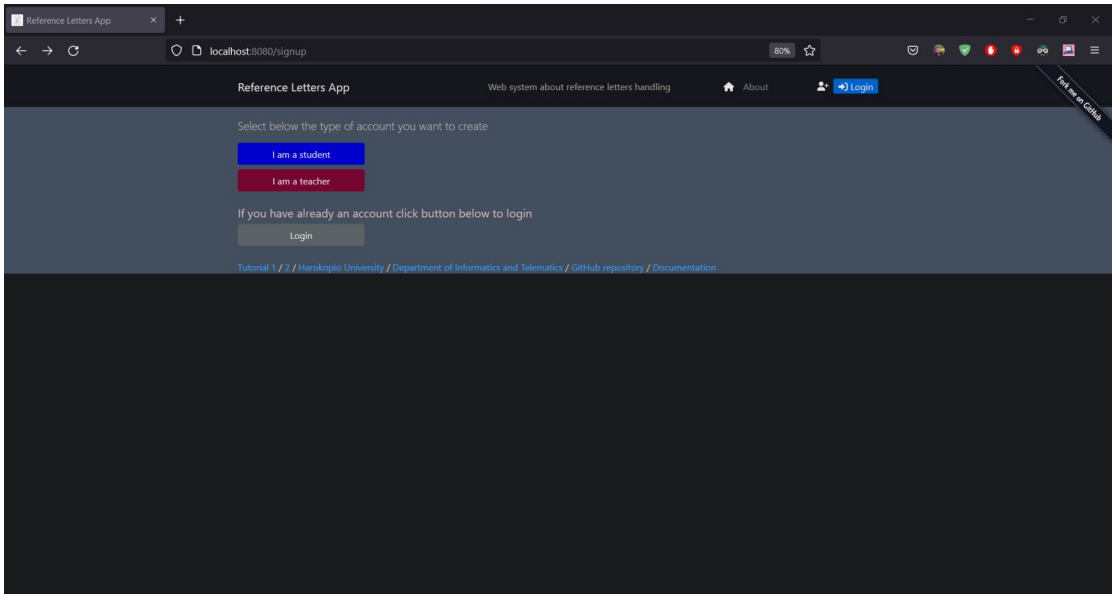
Για να διορθώσουμε τις τιμές στα .env αρχεία των εφαρμογών μπορούμε να χρησιμοποιήσουμε λίγο και την Ansible τρέχοντας ανάλογο playbook όπου θα το βρείτε στο ansible repository. Χρησιμοποιείται επίσης από τον Jenkins και το Jenkinsfile. Δείτε εδώ περισσότερα. [15]

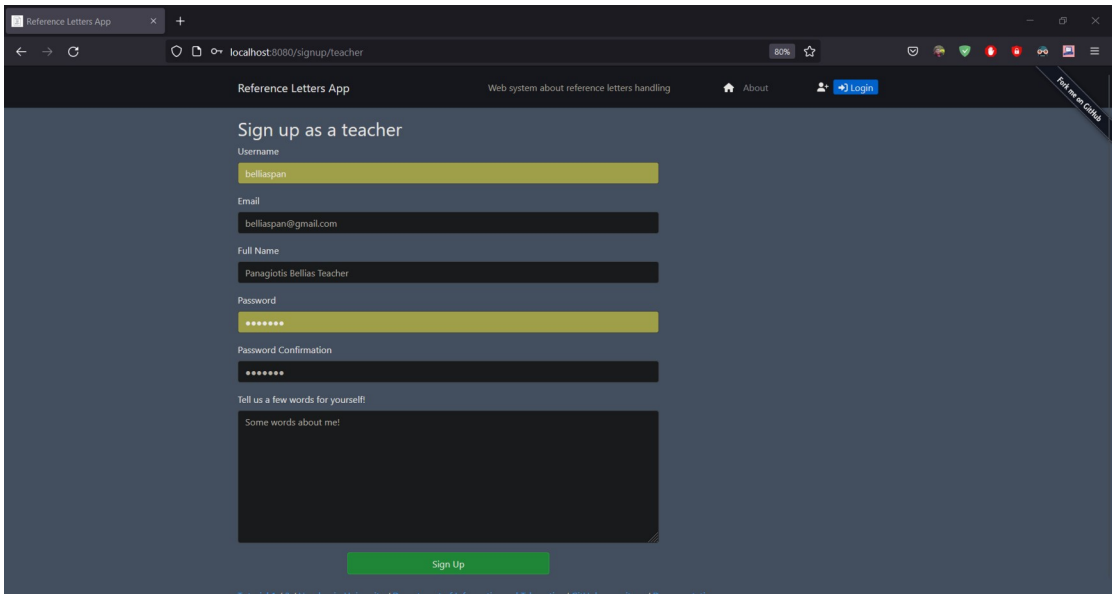
8.4 Εγκατάσταση SSL Certificates - HTTPS Περιβάλλον

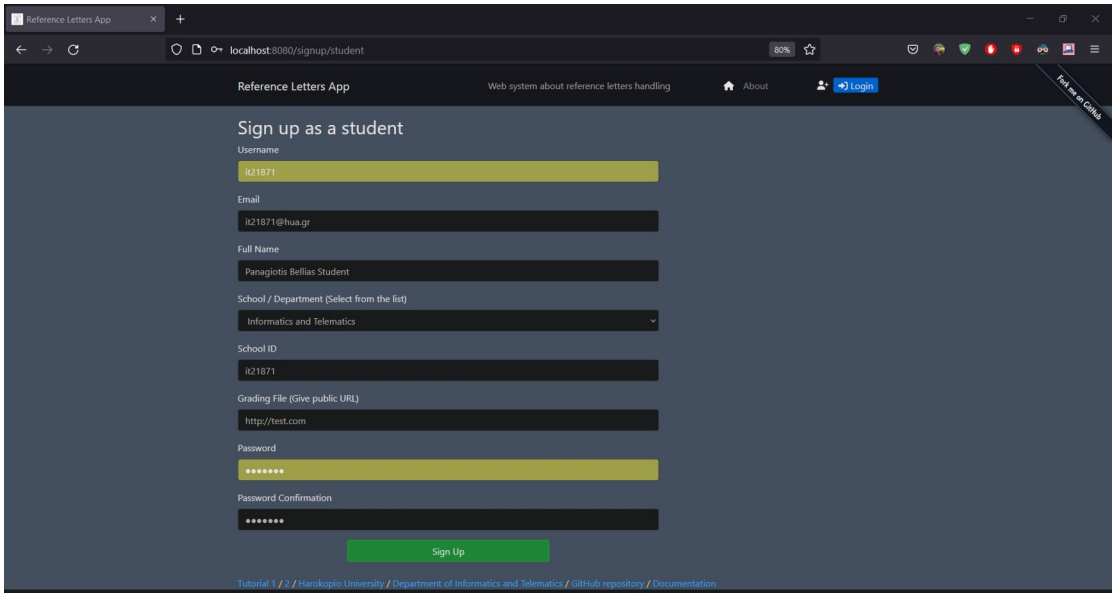
Για να ρυθμίσουμε https περιβάλλον στο kubernetes cluster μπορούμε να χρησιμοποιήσουμε το certbot και ακολουθώντας τα βήματα στο παρακάτω link μπορούμε εύκολα να πετύχουμε το στόχο μας. [16]

ΚΕΦ.9: Σενάρια Χρήσης

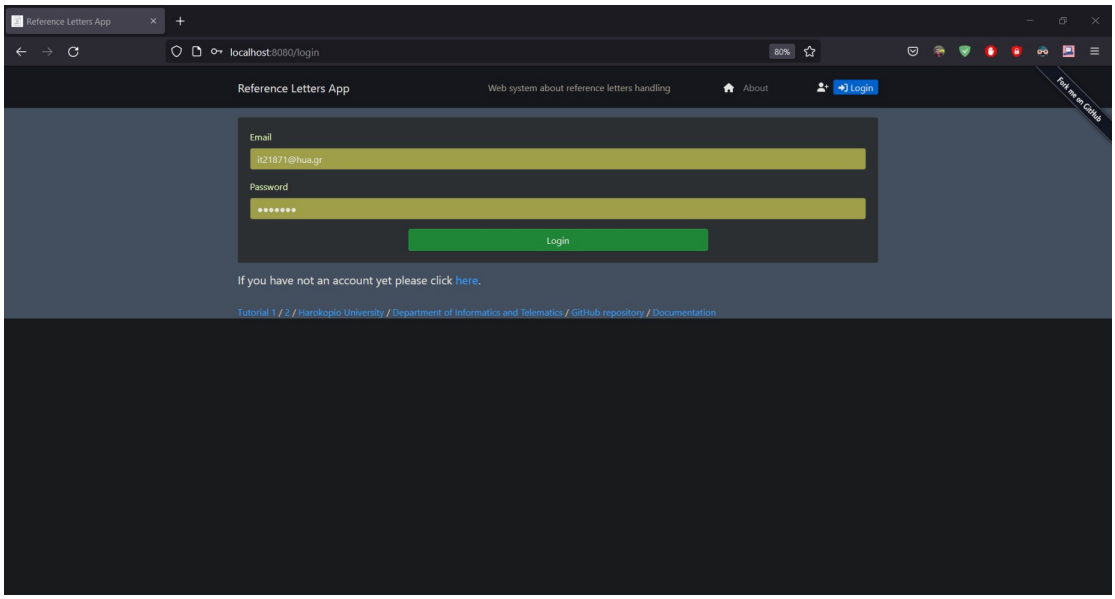
9.1 Σελίδα Εγγραφής & Σύνδεσης





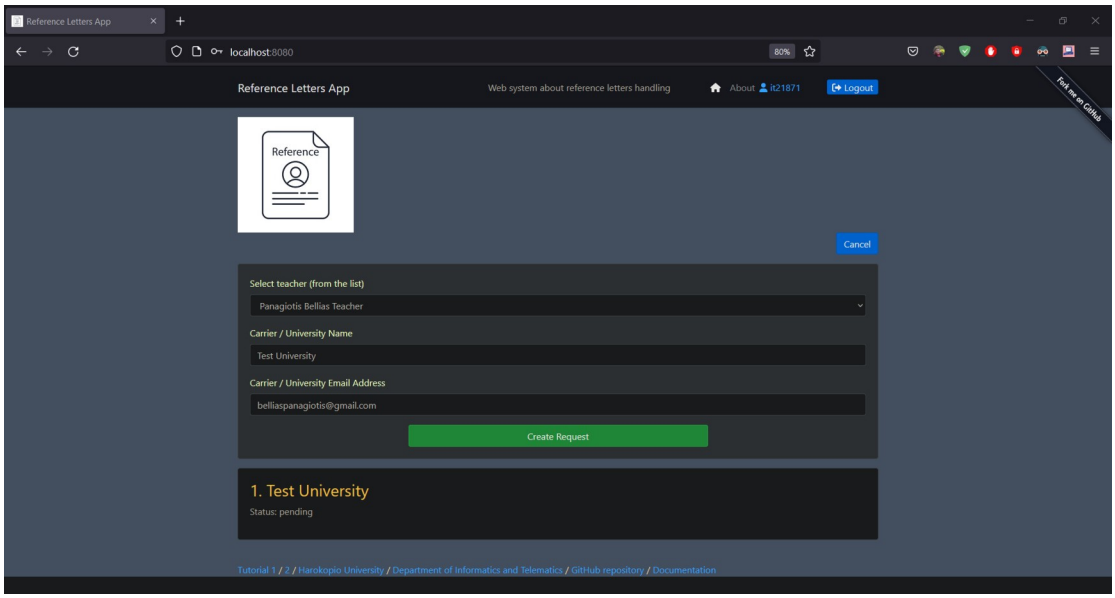


Εικόνα 73. Σελίδες Εγγραφής

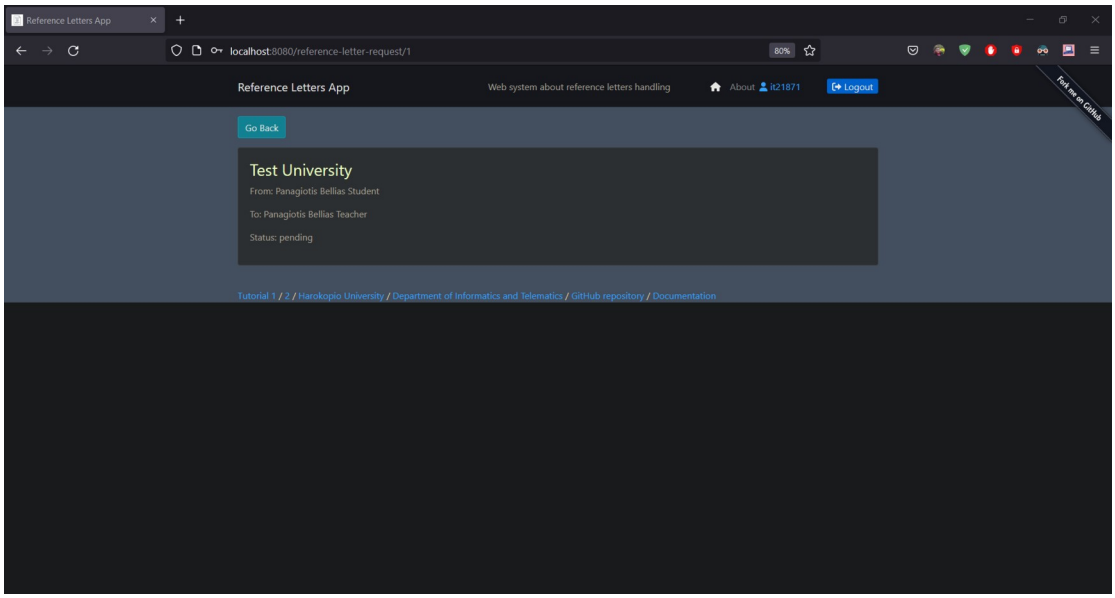


Εικόνα 74. Σελίδα Σύνδεσης

9.2 Λειτουργίες Χρηστών



Εικόνα 75. Αρχική Σελίδα Με Λίστα Των Αιτήσεων



Εικόνα 76. Επισκόπηση μίας συγκεκριμένης αίτησης

ΚΕΦ.10: Συμπεράσματα και Μελλοντικές Κατευθύνσεις

10.1 Συμπεράσματα

Μέσα από την παρούσα πτυχιακή εργασία και την ανάπτυξη του διαδικτυακού συστήματος διαχείρισης συστατικών επιστολών μπορούμε να καταλάβουμε πως ένα κομμάτι λογισμικού δεν περιορίζεται σε καμία περίπτωση στα πλαίσια της ανάπτυξής της.

Οι κύριοι στόχοι που επιτεύχθηκαν κατά την εκπόνηση της παρούσας πτυχιακής εργασίας ήταν:

a) η παρουσίαση μίας σύγχρονης και ραγδαία αναπτυσσόμενης τεχνολογικής τάσης και η κατανόηση του κύκλου ζωής των εφαρμογών (DevOps)
b) η ανάπτυξη μιας εφαρμογής όπου μπορεί να χρησιμοποιηθεί από την ακαδημαϊκή κοινότητα
c) η εξοικείωση με τεχνολογίες αυτοματοποίησης, συνεχούς ενσωμάτωσης των αλλαγών που γίνονται στον κώδικα και προώθηση των αλλαγών σε production περιβάλλον. (Με τη χρήση jenkins server)
d) η εξοικείωση με τεχνολογίες διαχείρισης παραμετροποιήσεων και ενορχήστρωσης και η αξιοποίηση αυτών για να εγκατασταθεί η εφαρμογή σε production περιβάλλον. (Με χρήση ansible)
e) η εξοικείωση με τεχνολογίες βασισμένες στη λογική των containers και της απομόνωσης της εφαρμογής από το υπόλοιπο λογισμικό που λειτουργεί στην υποδομή. (Με χρήση docker)
f) η εξοικείωση με τεχνολογίες βασισμένες στη λογική διαχείρισης των containers σε διαφορετικούς κόμβους διασφαλίζοντας την ομαλή λειτουργία της εφαρμογής. (Επιτυγχάνεται με τη χρήση kubernetes)
g) η διαδικασία ενσωμάτωσης των παραπάνω τεχνολογιών στην εκδοχή της εφαρμογής μας που έγινε με επιτυχία
h) η κατανόηση του τρόπου εκτέλεσης της εφαρμογής σε εικονικές μηχανές (virtual machines – VMs) αξιοποιώντας τις προαναφερόμενες τεχνολογίες
i) η διασύνδεση πολλαπλών κόμβων

10.2 Μελλοντικές Κατευθύνσεις

Η παρούσα διαδικτυακή εφαρμογή καλύπτει σε έναν ικανοποιητικό βαθμό τις απαιτήσεις που είχαν οι χρήστες κατά τη σχεδίαση της αλλά και την ενσωμάτωση τεχνολογιών βασισμένων στην DevOps κουλτούρα όπου αφορούν τον προγραμματιστή της εφαρμογής. Φυσικά, σε όλες τις εφαρμογές, συνεπώς και στην παρούσα εφαρμογή, υπάρχουν προοπτικές βελτίωσης τόσο ως προς την ανάπτυξη της εφαρμογής αυτής καθ' αυτής αλλά και ως προς την ενσωμάτωση περισσότερων τεχνολογιών DevOps.

Θα μπορούσαν να γίνουν βελτιώσεις όσον αφορά το security enhancement του συστήματος. Μία τεχνολογία που βοηθάει σε αυτό και διευκολύνει την επικοινωνία διαφορετικών components με ασφαλή τρόπο είναι το Keycloak. Το Keycloak είναι ένα προϊόν λογισμικού ανοιχτού κώδικα που επιτρέπει την απλή σύνδεση με διαχείριση ταυτότητας και πρόσβασης που στοχεύει σε σύγχρονες εφαρμογές και υπηρεσίες. Από τον Μάρτιο του 2018, αυτό το κοινοτικό έργο WildFly βρίσκεται υπό τη διαχείριση της Red Hat που το χρησιμοποιεί ως το upstream έργο για το προϊόν RH-SSO. [17] [18]

Κάτι ακόμα που θα ήταν πολύ εξυπηρετικό και για τον προγραμματιστή αλλά και για τον χρήστη είναι η ενσωμάτωση του MinIO. Το MinIO είναι ένας χώρος αποθήκευσης αντικειμένων υψηλής απόδοσης που κυκλοφορεί υπό την άδεια GNU Affero General Public License v3.0. Είναι συμβατό με API με την υπηρεσία αποθήκευσης cloud Amazon S3. Μπορεί να χειριστεί μη δομημένα δεδομένα, όπως φωτογραφίες, βίντεο, αρχεία καταγραφής, αντίγραφα ασφαλείας και εικόνες κοντέινερ με (προς το παρόν) το μέγιστο υποστηριζόμενο μέγεθος αντικειμένου των 5 TB. [19] Στην παρούσα εφαρμογή θα εξυπηρετούσε σίγουρα τη λειτουργία ανεβάσματος του αρχείου βαθμολογιών φοιτητή που κάνει ο φοιτητής κατά την εγγραφή του στο σύστημα ή κατά την ενημέρωση του προφίλ του. [20]

Τέλος, δεν θα μπορούσε να παραληφθεί ενδεχόμενη επέκταση που να αφορά το κομμάτι του (όσο το δυνατόν) καλύτερου testing της εφαρμογής. Προκειμένου μία εφαρμογή να προωθηθεί σε περιβάλλον production, είναι βαρύνουσας σημασίας να έχουν προβλεφθεί και υλοποιηθεί επαρκή σενάρια που θα αφορούν τη δοκιμή και τη συμπεριφορά της προτού παραδοθεί στον πελάτη για κανονική χρήση. Εργαλεία που μπορούν να βοηθήσουν σε αυτό είναι το pytest, το cypress [21] και το tavern [22].

ΒΙΒΛΙΟΓΡΑΦΙΑ

[professors-info] https://dit.hua.gr/index.php/el/department/faculty-a-staff/faculty-members
[1] https://el.wikipedia.org/wiki/JavaScript
[2] https://en.wikipedia.org/wiki/Vue.js
[JSONPlaceholder] https://jsonplaceholder.typicode.com/
[3] https://en.wikipedia.org/wiki/Visual_Studio_Code
[4] https://kubernetes.io/docs/concepts/services-networking/service/
[5] https://kubernetes.io/docs/home/
[6] https://fastapi-users.github.io/fastapi-users/10.1/configuration/databases/sqlalchemy/
[7] https://fastapi-users.github.io/fastapi-users/10.1/configuration/overview/
[8] https://fastapi-users.github.io/fastapi-users/10.1/usage/flow/
[9] https://www.vogella.com/tutorials/GitSubmodules/article.html
[10] https://docs.github.com/en/github/authenticating-to-github/creating-a-personal-access-token-for-the-command-line
[11] https://docs.github.com/en/packages/learn-github-packages/configuring-a-packages-access-control-and-visibility
[12] https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry
[13] https://www.jenkins.io/doc/book/installing/linux/
[14] https://ubuntu.com/tutorials/install-a-local-kubernetes-with-microk8s#2-deploying-microk8s
[15] https://github.com/panagiotis-bellias-it21871/ansible-reference-letter-code#k8s
[16] https://runnable.com/blog/how-to-use-lets-encrypt-on-kubernetes
[17] https://en.wikipedia.org/wiki/Keycloak
[18] https://www.packtpub.com/mapt/book/web_development/9781786463630/12/ch12lvl1sec75/introducing-red-hat-sso
[19] https://docs.min.io/docs/minio-server-limits-per-tenant.html
[20] https://en.wikipedia.org/wiki/MinIO
[21] https://www.cypress.io/blog/2021/04/06/getting-start-with-cypress-component-testing-vue-2-3/
[22] https://tavern.readthedocs.io/en/latest/
[23] https://sabuhish.github.io/fastapi-mail/example/#using-jinja2-html-templates

[24] https://noip.com/

[other-thesis] https://estia.hua.gr/browse/25671

Παράρτημα Α’ - Βοηθητικός Κώδικας

- https://towardsdatascience.com/build-an-async-python-service-with-fastapi-sqlalchemy-196d8792fa08

Παράρτημα Β’ - Αποθετήρια Κώδικα Εργασίας

- https://github.com/panagiotis-bellias-it21871/reference-letters-system.git
- https://github.com/panagiotis-bellias-it21871/reference-letters-fastapi-server.git
- https://github.com/panagiotis-bellias-it21871/reference-letters-vuejs-client.git
- https://github.com/panagiotis-bellias-it21871/ansible-reference-letter-code.git