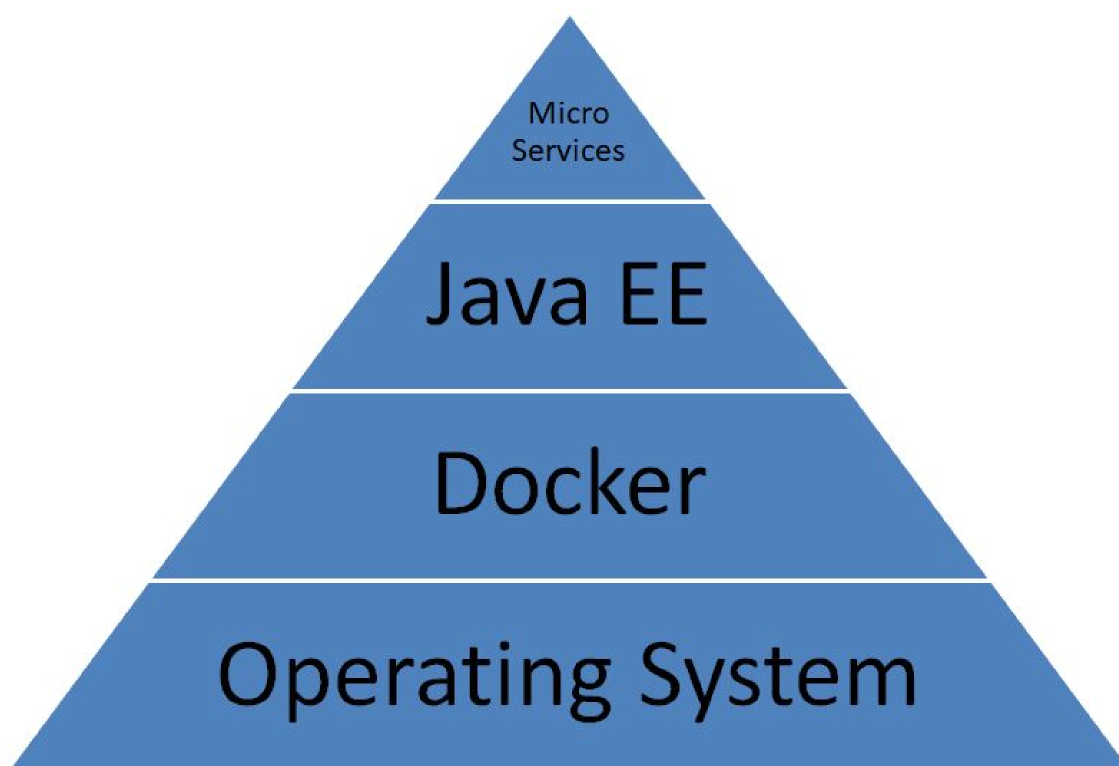
ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη Διαδικτυακών Εφαρμογών με την Αρχιτεκτονική των Microservices με την χρήση της Java EE σε περιβάλλον Docker



Του φοιτητή
Μπουραντά Γεώργιου
Αρ. Μητρώου: 123903

Επιβλέπων καθηγητής
Γιακουστίδης Κωνσταντίνος

Θεσσαλονίκη 2018

Ευχαριστίες **5**

Abstract **6**

Ευρετήριο σχημάτων **7**

1. Εισαγωγή **8**

2. Virtualization **9**
2.1 Εισαγωγή 9
2.2 Hypervisor Virtualization 9
2.3 Container Virtualization 10

3. Containerization **11**
3.1 Εισαγωγή 11
3.2 Τι είναι το runC ; 11
3.3 Containerization και runC 11
3.3.1 Containerization και runC 11
3.3.2 Namespaces 12
3.3.3 Control groups (cgroups) 12

4. Docker **12**
4.1 Εισαγωγή 12
4.2 Πλατφόρμες υποστήριξης του Docker 13
4.3 Δυνατότητες 13
4.4 Οι βασικές τεχνολογίες 14
4.4.1 Kernel Namespaces 14
4.4.2 Control groups (cgroups) 15
4.4.3 Union File System 15
4.5 Docker Engine 15
4.6.2 Ο δαίμονας dockerd 16
4.6.3 Ο docker πελάτης 16
4.6.4 Αποθετήριο Docker 16
4.7 Docker Objects 17
4.7.1 Images 17
4.7.2 Containers 17
4.7 Dockerfile 18

5. Μονολιθικές Εφαρμογές **19**
5.1 Εισαγωγή 19
5.2 Αρχιτεκτονική Μονολιθικών Εφαρμογών 19
5.2.1 Αρθρωτός Μονόλιθος 19
5.2.2 Κατανεμημένος Μονόλιθος 20
5.2.3 Runtime Μονόλιθος 21
5.3 Πλεονεκτήματα των μονολιθικών εφαρμογών 21

5.4 Μειονεκτήματα Μονολιθικό Εφαρμογών 22
5.4.1 Περιορισμένη Ευελιξία 22
5.4.2 Μείωση παραγωγικότητας 22
5.4.3 Δύσκολη δομή ομάδας 22
5.4.4 Μακροπρόθεσμη δέσμευση 23
5.4.5 Περιορισμένη δυνατότητα κλιμάκωσης 23
5.5 Αρχιτεκτονική SOA 23
5.5.1 Πρότυπο Enterprise Service Bus 24

6. Microservices Εφαρμογές 25
6.1 Εισαγωγή 25
6.2 Μοντέλο Scale Cude 25
6.2.1 Κλιμάκωση κατά τον άξονα X 26
6.2.2 Κλιμάκωση κατά τον άξονα Z 26
6.2.3 Κλιμάκωση κατά τον άξονα Y 26
6.3 Κοινόχρηστες βιβλιοθήκες 27
6.4 Χαρακτηριστά Microservices 27
6.4.1 Μοναδική Ευθύνη 27
6.4.2 Αυτονομία 27
6.4.3 Ανομοιογένεια 28
6.4.4 Ελαστικότητα 28
6.4.5 Κλιμάκωση 28
6.4.6 Εύκολη εγκατάσταση 28
6.4.7 Επαναχρησιμοποίηση 28
6.4.8 Αντικατάσταση 29
6.4.9 Μικρές Ομάδες 29

7. Java EE 29
7.1 Εισαγωγή 29
7.2 Ιστορική Αναδρομή 29
7.3 Πως γίνεται η ανάπτυξη της πλατφόρμας; 30
7.4 Τι είναι η Java EE; 30
7.5 Τι είναι ο Application Server; 30
7.6 Μοντέλο Ανάπτυξης Java EE Εφαρμογών 31
7.7 Αρχιτεκτονική 32
7.7.1 Java EE Components 32
7.7.2 Java EE Containers 32
7.7.3 Java EE Modules 33
7.8 Java EE Τεχνολογίες 34
7.8.1 CDI – Context and Dependency Injection 34
7.8.1.1 Ορισμός CDI 34
7.8.1.2 Ορισμός Java Beans 34
7.8.2 EJB3 – Enterprise JavaBeans 34

7.8.2.1 Session Beans 35
7.8.2.1.1 Stateful Session Beans 35
7.8.2.1.2 Stateless Session Beans 35
7.8.2.1.3 Singleton Session Beans 35
7.8.2.2 Entity JavaBeans 36
7.8.3 Συναλλαγές 36
7.8.3.1 Χαρακτηριστικά Συναλλαγών 36
7.8.3.2 Container-Managed Transactions 37
7.8.3.3 Bean-Managed Transactions (BMT) 38
7.8.3.4 Επίπεδα Απομόνωσης Συναλλαγών 38

8. Υλοποίηση 39
8.1 Εισαγωγή 39
8.2 Αρχιτεκτονική Εφαρμογής 39
8.3 Αρχιτεκτονική βάσης 42
8.4 Αυθεντικοποίηση και διαχείριση ταυτότητας 44
8.5 Web Server και Reverse Proxy 44
8.6 Υλοποίηση Microservices 46
8.6.1 Ρυθμίσεις βάσης δεδομένων 46
8.6.2 Ρυθμίσεις Web Services 47
8.6.3 Ρυθμίσεις ασφαλείας 47
8.6.4 Phones microservice 49
8.6.5 Users microservice 52
8.6.6 Orders microservice 58
8.7 Ηλεκτρονικό κατάστημα 62
8.7.1 Angular 62
8.7.2 Typescript 62
8.7.3 Πελάτης RestClient 63
8.7.4 Μακέτα εφαρμογής 65

10. Βιβλιογραφία 66

Ευχαριστίες

Με την ολοκλήρωση της παρούσας πτυχιακής εργασίας, θέλω να πω ένα μεγάλο και εγκάρδιο ευχαριστώ, στους δύο ήρωες της καθημερινότητας μου, τους γονείς μου, τον Μπουραντά Χαρίλαο και την Λαδικού Ελευθερία, που με στηρίζουν ηθικά όλα αυτά τα χρόνια, δίνοντάς μου κουράγιο να προχωρώ και να υπερπηδώ κάθε εμπόδιο.

Επιπροσθέτως θα ήθελα να ευχαριστήσω όλους τους καθηγητές του ΑΤΕΙ Μηχανικών Πληροφορικής Θεσσαλονίκης που μου έδωσαν τα κατάλληλα κίνητρα, τις απαραίτητες γνώσεις, τα εφόδια και την αγάπη για εξερεύνηση, στο ταξίδι της πληροφορίας.

Κυρίως ευχαριστώ τον επιβλέπων καθηγητή κ.Γιακουστίδη Κωνσταντίνο για την αμέριστη και ουσιαστική βοήθεια αλλά και καθοδήγηση που μου παρείχε όχι μόνο κατά τη διάρκεια συγγραφής της πτυχιακής εργασίας αλλά όλα τα χρόνια φοίτησής μου.

Τέλος, θέλω να σας διαβεβαιώσω ότι η ολοκλήρωση της πτυχιακής μου εργασίας αλλά και η ολοκλήρωση της προπτυχιακής μου εκπαίδευσης είναι μόνο η αρχή των μακροπρόθεσμων στόχων μου.

Σας ευχαριστώ και πάλι,
Μπουραντάς Γεώργιος

Abstract

In the reality of the 21st century, people are solving their daily problems through electronic services.The majority of them demand faster, safer and cheaper electronic services.In such world, where the demand is so high, programmers and corporations are looking for solutions to serve the increased demand.While using the most common and old technologies, is quite difficult and tricky to develop and maintain such services.A solution to such problem, could be to divide the original problem, into smaller problems and each problem get solved by a small group of people.This can help programmers, teams and corporations to easily focus on solving customer's demands and at the same time being extremely productive.

This paper explains the meaning behind of virtualization and containerization technologies and software architectural designs such as monolith applications and microservices.In addition, it explains in easy steps, how technologies such as Docker and Java EE can cooperate and provides the information needed to get started to develop services using the microservices architecture.

Ευρετήριο σχημάτων

Σχήμα 2.1 Είδη Hypervisor 9
Σχήμα 2.2 Πολλαπλές εφαρμογές που τρέχουν στο ίδιο λειτουργικό σύστημα 10
Σχήμα 4.1 Αρχιτεκτονική Docker 15
Σχήμα 4.2 Αρχιτεκτονική Docker Engine 16
Σχήμα 4.3 Κύκλος ζωής ενός container 18
Σχήμα 5.1 Αρθρωτές Εφαρμογές 20
Σχήμα 5.2 Κατανεμημένες Εφαρμογές 20
Σχήμα 5.3 Runtime Εφαρμογές 21
Σχήμα 5.4 Αρχιτεκτονική Enterprise Service Bus 24
Σχήμα 6.1 Αρχιτεκτονική Scale Cube 25
Σχήμα 8.1 Αρχιτεκτονική Εφαρμογής 41
Σχήμα 8.2 Αρχιτεκτονική Βάσης Δεδομένων 42
Σχήμα 8.3 Αρχιτεκτονική Typescript 63

1. Εισαγωγή

Αυτή η πτυχιακή εργασία έχει σκοπό την μελέτη και παρουσίαση σύγχρονων τεχνολογιών που χρησιμοποιούνται στην ανάπτυξη εφαρμογών διαδικτύου όπως το περιβάλλον Docker και η Java EE πλατφόρμα.Επίσης εξηγεί σε απλά βήματα πως οι δύο προαναφερθείσες τεχνολογίες μπορούν να συνδυαστούν σε μια microservices αρχιτεκτονική.Με τη σωστή χρήση των παραπάνω τεχνολογιών μπορεί να γίνει ανάπτυξη εφαρμογής που αποτελείται από πολλές ανεξάρτητες υπηρεσίες που έχουν τον δικό τους κύκλο ζωής.

Στο 2ο κεφάλαιο γίνεται μια σύντομη περιγραφή της έννοιας του Virtualization.Επίσης αναλύονται οι τύποι του Virtualization καθώς και τα πλεονεκτήματα του κάθε τύπου.

Στο 3ο κεφάλαιο συνεχίζεται η ανάλυση ενός άλλου είδους Virtualization γνωστό και ως Containerization.Εξηγείται πως αυτό θέτει τα σημερινά θεμέλια για την δημιουργία της Docker πλατφόρμας αλλά και το σημαντικό ρόλο που έχει στην αρχιτεκτονική των microservices.

Στο 4ο κεφάλαιο εξερευνούμε την Docker πλατφόρμα και εξηγούμαι τα βασικά συστατικά απ τα οποία αποτελείται.

Στο 5ο κεφάλαιο κάνουμε μια εισαγωγή στις αρχιτεκτονικές μονολιθικών εφαρμογών ενώ στο 6ο κεφάλαιο αναλύουμε την αρχιτεκτονική των microservices και πως αυτή χρησιμοποιείται για την ανάπτυξη εφαρμογών.

Στο 7ο κεφάλαιο περιγράφουμε τα βασικά χαρακτηριστικά της Java EE πλατφόρμας και τις βασικές τεχνολογίες απ τις οποίες αποτελείται.Επιπροσθέτως εξηγούμε πως μια τέτοια σύγχρονη πλατφόρμα βοηθά στην ανάπτυξη εφαρμογών που χρησιμοποιούν την microservices αρχιτεκτονική.

Στο 8ο κεφάλαιο υλοποιείται μια εφαρμογή διαδικτύου που αναπαριστά ένα ηλεκτρονικό κατάστημα το οποίο χρησιμοποιεί όλες τις προαναφερθείσες τεχνολογίες για την καλύτερη κατανόηση τους.

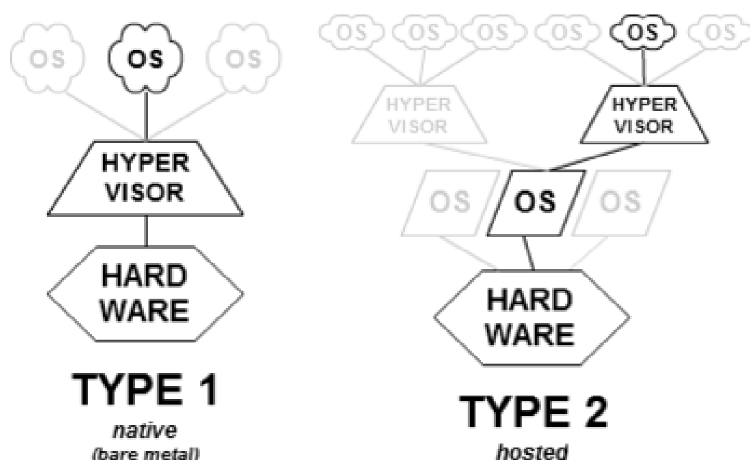
2. Virtualization

2.1 Εισαγωγή

Με την εξέλιξη της τεχνολογίας οι αποδόσεις του υλικού που χρησιμοποιούμε, για την δημιουργία ηλεκτρονικών υπολογιστών, βελτιώθηκε ραγδαία με αποτέλεσμα να μην εκμεταλλευόμαστε τις πλήρες δυνατότητες τους.Ένας απ’ τους τρόπους εκμετάλλευσης αυτού του πλεονασμού ταχύτητας είναι το virtualization.Ως virtualization ορίζουμε την διανομή των πόρων υλικού σε πολλαπλά εικονικά περιβάλλοντα με σκοπό την καλύτερη χρήση των διαθέσιμων πόρων.Σε αυτό το κεφάλαιο θα εξετάσουμε το virtualization που είτε βασίζεται στον hypervisor είτε στους containers.Το virtualization με τη χρήση του hypervisor βασίζεται στο λογισμικό το οποίο ονομάζεται hypervisor προσφέροντας ένα είδος αφαιρετικότητας των πόρων στις εικονικές μηχανές.Αντίθετα το virtualization που βασίζεται στους containers δημιουργεί διεργασίες που ονομάζονται containers οι οποίες είναι απομονωμένες η μία από την άλλη χρησιμοποιώντας ταυτόχρονα τον ίδιο πυρήνα του λειτουργικού συστήματος.

2.2 Hypervisor Virtualization

Το virtualization με την χρήση του hypervisor μας δίνει την δυνατότητα να φιλοξενήσουμε ολόκληρες εικονικές μηχανές που περιλαμβάνουν ένα ολοκληρωμένο λειτουργικό σύστημα μαζί με τον πυρήνα του.Το hypervisor είναι και αυτό ένα λογισμικό,όπως βλέπουμε και στο σχήμα 2.1 υπάρχουν δύο είδη hypervisors.Το Type 1 που είναι γνωστό ως “bare metal hypervisor” το οποίο εκτελείται άμεσα στο υλικό του ηλεκτρονικού υπολογιστή και το Type 2 που είναι γνωστό ως “hosted hypervisor” το οποίο φιλοξενείται σε ένα λειτουργικό σύστημα.Και στις δύο περιπτώσεις το επίπεδο ασφαλείας είναι αρκετά υψηλό γιατί η κεντρική μονάδα επεξεργασίας παρέχει ειδικές οδηγίες για την απομόνωση υλικού και μόνο το hypervisor προσφέρει μια μικρή επιφάνεια επίθεσης.



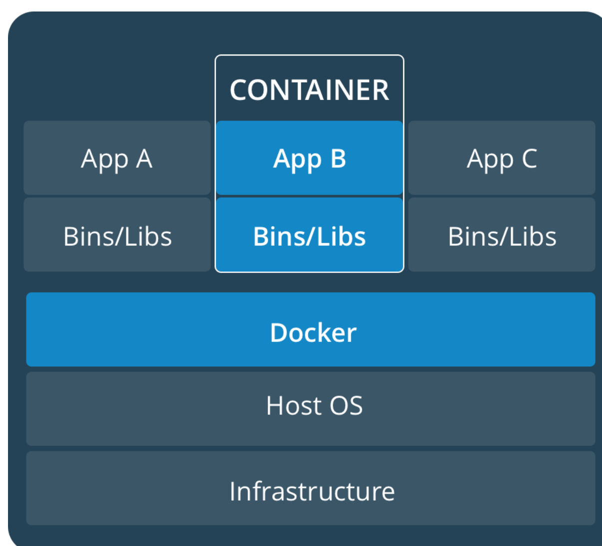
Σχήμα 2.1 Είδη Hypervisor

Type 1 : Xen, Oracle VM Server SPARC/x86,Microsoft Hyper-V και VMware ESX/ESXi

Type 2 : VMware Workstation, VMware Player, VirtualBox, Parallels Desktop και QEMU

2.3 Container Virtualization

Το virtualization με την χρήση container γνωστό ως containerization χρησιμοποιεί δυνατότητες του πυρήνα για να απομονώνει διεργασίες την μία από την άλλη.Το περιβάλλον απομόνωσης ονομάζεται container.Οι πιο βασικές τεχνολογίες που χρησιμοποιούνται για την δημιουργία τους είναι το kernel namespaces, το cgroups ή το root capabilities.Οι containers χρησιμοποιούν τον πυρήνα του λειτουργικού συστήματος οπότε δεν χρειάζεται η χρήση του hypervisor.Σε σύγκριση του virtualization με τη χρήση hypervisor οι απαιτήσεις σε πόρους είναι λιγότερη επειδή δεν υπάρχει ανάγκη να φορτωθούν το λειτουργικό σύστημα και ο πυρήνας που θα λειτουργούσαν ως ενδιάμεσος.Το containerization προσφέρει υψηλότερες αποδώσεις και ταχύτητες εκκίνησης διότι οι εφαρμογές εκτελούνται απευθείας στον πυρήνα του λειτουργικού συστήματος απ το οποίο φιλοξενούνται.Οι υψηλές αποδώσεις και η ταχύτητα είναι απ τους βασικούς λόγους που προτιμάται virtualization με containers έναντι με hypervisor.



Σχήμα 2.2 Πολλαπλές εφαρμογές που τρέχουν στο ίδιο λειτουργικό σύστημα

3. Containerization

3.1 Εισαγωγή

Η γενική ιδέα του containerization δεν δημιουργήθηκε από το Docker αλλά υπάρχει εδώ και αρκετά χρόνια.Πιο συγκεκριμένα η πρώτη εμφάνιση του γίνεται στο λειτουργικό σύστημα FreeBSD με την ονομασία Jails το 2000,μετά από τέσσερα χρόνια,δηλαδή το 2004 εμφανίστηκε το Solaris Zones για το λειτουργικό σύστημα Solaris και τελικά το 2008 κάνει την εμφάνιση του, το LXC το οποίο υποστηρίζεται από οποιοδήποτε Linux διανομή που χρησιμοποιεί τον πυρήνα του Linux από την έκδοση 2.6.24 και έπειτα.Τέλος η πρώτη έκδοση του Docker ήταν τον Μάρτιο του 2013 το όποιο βασιζόταν στις αρχές του LXC αλλά αργότερα δημιούργησε την δικιά του υλοποίηση με ονομασία runC.

3.2 Τι είναι το runC ;

Τον Ιούνιο του 2015 η Docker Inc μαζί με τους συνεργάτες της δημιουργούν έναν οργανισμό ανοιχτού κώδικα με ονομασία Open Container Initiative.Σκοπός του οργανισμού είναι να καθορίσει τις προδιαγραφές που πρέπει να έχουν οι containers και τα images ανεξαρτήτους λειτουργικού συστήματος.

Αποτέλεσμα αυτού ήταν η δημιουργία δυο προδιαγραφών με ονόματα Runtime Specification (runtime-spec) και Image Specification (image-spec).Βασισμένο στις προδιαγραφές του Runtime Specification γίνεται ανάπτυξη του runC.Το runC είναι υπεύθυνο για τον κύκλο ζωής ενός container όπως το ξεκίνημα, το σταμάτημα, η παύση αλλά και η καταστροφή του.

3.3 Containerization και runC

Για να καταλάβουμε καλύτερα τι είναι ένας container πρέπει να αναλύσουμε τα βασικά στοιχεία του containerization που βασίζεται στο runC και τις τεχνολογίες που το απαρτίζουν όπως το cgroups και namespaces.

3.3.1 Containerization και runC

Το runC είναι ένα λογισμικό για virtualization, σε επίπεδο λειτουργικού συστήματος, το οποίο μας δίνει την δυνατότητα δημιουργίας, για ένα ή πολλαπλά αυτόνομα και απομονωμένα εικονικά περιβάλλοντα σε ένα μηχάνημα, τα οποία είναι απομονωμένα μεταξύ τους, αλλά και από το λειτουργικό σύστημα στο οποίο φιλοξενούνται.Με το runC ο επεξεργαστής του οικοδεσπότης έχει την ικανότητα να διαχωρίσει και απομονώσει πιο καλά τους πόρους τους συστήματος με την χρήση των namespaces και cgroups τεχνολογιών.Η απομόνωσης που έχει ο κάθε container

είναι σε πολλά επίπεδα όπως στην χρήση της κεντρικής μονάδας επεξεργασίας, της μνήμης τυχαίας προσπέλασης, το αποθηκευτικό μέσο και τους πόρους δικτύου.

3.3.2 Namespaces

Τα Namespaces είναι μια τεχνολογία η οποία εμπεριέχεται στο πυρήνα Linux.Σκοπός της είναι να απομονώνει και να δημιουργεί εικονικούς πόρους για διεργασίες.Η απομόνωση γίνεται μεταξύ του λειτουργικού συστήματος από τις διεργασίες αλλά και τις διεργασίες μεταξύ τους.Οι πόροι που μπορούν να απομονωθούν περιλαμβάνουν τις διεργασίες, τα hostname, τους χρήστες, τις διαδικτυακές συσκευές, την επικοινωνία μεταξύ διεργασιών και το σύστημα αρχείων.

3.3.3 Control groups (cgroups)

Όπως και τα namespaces, έτσι και τα cgroups ή αλλιώς control groups είναι μια τεχνολογία η οποία εμπεριέχεται στο πυρήνα Linux.Έχει την δυνατότητα να διαμοιράζει και προαιρετικά να περιορίζει την χρήση πόρων όπως τη κεντρική μονάδα επεξεργασίας, τη μνήμης τυχαίας προσπέλασης, το αποθηκευτικό μέσο και τους πόρους δικτύου.

4. Docker

4.1 Εισαγωγή

Η πρώτη ονομασία του οικοσυστήματος του Docker ήταν dotCloud αλλά άλλαξε σε Docker, όταν έγινε λογισμικό ανοιχτού κώδικα το 2013.Η ανάπτυξή αυτού γίνεται στη γλώσσα προγραμματισμού Go.Κάνει χρήση του container virtualization προσθέτοντας ένα επίπεδο αφαιρετικότητας στο μηχανισμό αυτό.Βασίζεται στο λογισμικό runC το οποίο του δίνει την δυνατότητα δημιουργίας container δηλαδή να απομονώνει τις διεργασίες την μία από την άλλη και ταυτόχρονα να ελέγχει τους πόρους τους.

Το Docker είναι μια πλατφόρμα από διαφορετικές εφαρμογές για ανάπτυξη, διανομή και εκτέλεση εφαρμογών σε ένα απομονωμένο περιβάλλον το οποίο ονομάζεται container.Οι containers μπορούν να εκτελούνται απομονωμένα στο ίδιο λειτουργικό σύστημα χωρίς τη χρήση του hypervisor.

Η απομόνωση και η ασφάλεια, μας δίνει την δυνατότητα να εκτελούμε πολλούς containers ταυτόχρονα σε έναν οικοδεσπότη.Οι containers δεν απαιτούν πολλούς πόρους διότι δεν χρειάζεται το λειτουργικό σύστημα να φορτώσει τον hypervisor αλλά να εκτελούνται κατευθείαν από τον πυρήνα του λειτουργικού συστήματος στον οποίο φιλοξενούνται.

Το οικοσύστημα του Docker προσφέρει διάφορες υπηρεσίες όπως φαίνονται παρακάτω

- Docker Engine - ο πυρήνας του οικοσυστήματος Docker
- Docker Compose - εκτέλεση πολλαπλών container με χρήση ενός αρχείου
- Docker Swarm - ενορχήστρωση των container σε μεγάλη κλίμακα
- Docker Registry - αποθήκη με Docker Images
- Universal Control Plane - διαχείριση των containers σε επιχειρίσεις
- Docker Secrets - διαχείριση ευαίσθητων πληροφορίων
- Docker Content Trust - αποθήκη και επικαιροποίηση των Docker Images

4.2 Πλατφόρμες υποστήριξης του Docker

Το Docker υποστηρίζεται από της παρακάτω πλατφόρμες :

- Σε οποιαδήποτε διανομή Linux με έκδοση πυρήνα 3.10 και έπειτα σε x86-64 και ARM επεξεργαστές.
- Στα λειτουργικά συστήματα Windows Server 2016 και Windows 10 αλλά μόνο σε x86-64 επεξεργαστές.
- Σε cloud υποδομές όπως Amazon EC2, Google Compute Engine, Microsoft Azure, Rackspace

4.3 Δυνατότητες

Ένας από τους βασικούς λόγους που το Docker έχει αποκτήσει τόσο δημοτικότητα σε αντίθεση με άλλες τεχνολογίες containerization είναι ότι, περιλαμβάνει τις χαμηλού επιπέδου δυνατότητες όπως cgroups και namespaces αλλά προσθέτει τις υψηλού επιπέδου δυνατότητες που θα αναφέρουμε παρακάτω.

1) Το Docker Engine κάνει δυνατή την δημιουργία containers ανεξάρτητα απ το λειτουργικό σύστημα στο οποίο θα φιλοξενηθούν.Αποτέλεσμα αυτού είναι δυνατή η μεταφορά των containers από λειτουργικό, σε λειτουργικό χωρίς προβλήματα.Η δημιουργία containers με άλλη τεχνολογία κάνουν τους containers να είναι εξαρτημένοι από τις ρυθμίσεις του λειτουργικού συστήματος στο οποίο φιλοξενούνται.Το Docker προσθέτει ένα είδος αφαιρετικότητας στις συσκευές αποθήκευσης, δικτύου και λειτουργικού συστήματος ώστε να επιτύχει αυτή τη φορητότητα.
2) Το Docker είναι βελτιστοποιημένο για την ανάπτυξη εφαρμογών παρά την δημιουργία εικονικών μηχανών.Αυτό αντικατοπτρίζεται στην φιλοσοφία , το σχεδιασμό αλλά και στις οδηγίες χρήσεις ως προς την προσφερόμενη διεπαφή προγραμματισμού εφαρμογών.Σε αντίθεση με άλλες τεχνολογίες που

εστιάζουν στην δημιουργία ελαφριών εικονικών μηχανών που έχουν γρήγορη εκκίνηση και παράλληλα χρειάζονται λίγους πόρους σε μνήμη.

3) Δίνει εργαλεία στους προγραμματιστές όπως την αυτοματοποίηση της δημιουργίας ενός container και πλήρη έλεγχο στις εξαρτήσεις τις εφαρμογής με τη χρήση κώδικα σε ένα απλό αρχείο κειμένου που ονομάζεται Dockerfile.
4) Έχει ενσωματωμένο σύστημα ελέγχου έκδοσης container (π.χ. σαν git), ελέγχοντας τις διαφορές μεταξύ εκδόσεων, παράδοση νέων εκδόσεων αλλά και επιστροφή σε παλαιότερη έκδοση.Με το σύστημα ελέγχου έκδοσης είναι δυνατό να γίνει ενημέρωση σε μια νέα έκδοση εφαρμόζοντας στη παλιά μόνο τις διαφορές που έχει με την καινούργια.
5) Επαναχρησιμοποίησή των containers ως βάση για την δημιουργία νέων containers τα οποία θα έχουν τον δικό τους κύκλο ζωής.
6) Ιδιωτικό είτε δημόσιο αποθετήριο για την δημοσιοποιήσει και διαμοίραση των Dockerfile αρχείων ώστε άλλοι προγραμματιστές να επωφελούνται χρησιμοποιώντας τα.
7) Κατασκευαστές δημιουργούν ένα οικοσύστημα προσανατολισμένο γύρω από το Docker όπου υποστηρίζουν και επεκτείνουν τις δυνατότητες του.

4.4 Οι βασικές τεχνολογίες

4.4.1 Kernel Namespaces

Για να επιτευχθεί η δημιουργία ενός απομονωμένου container το Docker χρησιμοποιεί την τεχνολογία που ονομάζεται namespaces.Όταν δημιουργούμε έναν container το Docker δημιουργεί ένα σύνολο από namespaces για τον συγκεκριμένο container.Το κάθε namespace παρέχει το δικό του επίπεδο απομόνωσης.Κάθε συστατικό κομμάτι ενός container εκτελείται σε διαφορετικό namespace και η πρόσβασή του περιορίζεται σε αυτό το namespace.Όλα τα επίπεδα απομόνωσης έχουν ως αποτέλεσμα οι διεργασίες στις οποίες εκτελούνται οι containers να μην έχουν την δυνατότητα πρόσβασης ή παραποίησης σε άλλες διεργασίες.Η κάθε διεργασία έχει την δικιά της ξεχωριστή εικόνα για το σύστημα.Το Docker σε λειτουργικό σύστημα Linux χρησιμοποιεί τα παρακάτω namespaces :

- **PID (Process ID) Namespace** είναι υπεύθυνο για την απομόνωση των διεργασιών.
- **NET (Networking) Namespace** διαχειρίζεται τις διαδικτυακές συσκευές.
- **IPC (InterProcess Communication) Namespace** διαχειρίζεται την επικοινωνία μεταξύ διεργασιών.
- **MNT (Mount) Namespace** διαχειρίζεται τις συσκευές, το σύστημα αρχείων και τα μέσα αποθήκευσης.
- **UTS (Unix Timesharing System) Namespace** διαχείριση των host και domain ονομάτων.

4.4.2 Control groups (cgroups)

Το Docker χρησιμοποιεί τα control groups ώστε, να κάνει διαμερισμό τους πόρους του οικοδεσπότη, μεταξύ των containers και προαιρετικά να τους θέσει όρια και περιορισμούς όπως στη χρήστη της κεντρικής μονάδας επεξεργασίας,της μνήμης τυχαίας προσπέλασης,του αποθηκευτικού μέσου και τους πόρους δικτύου.

4.4.3 Union File System

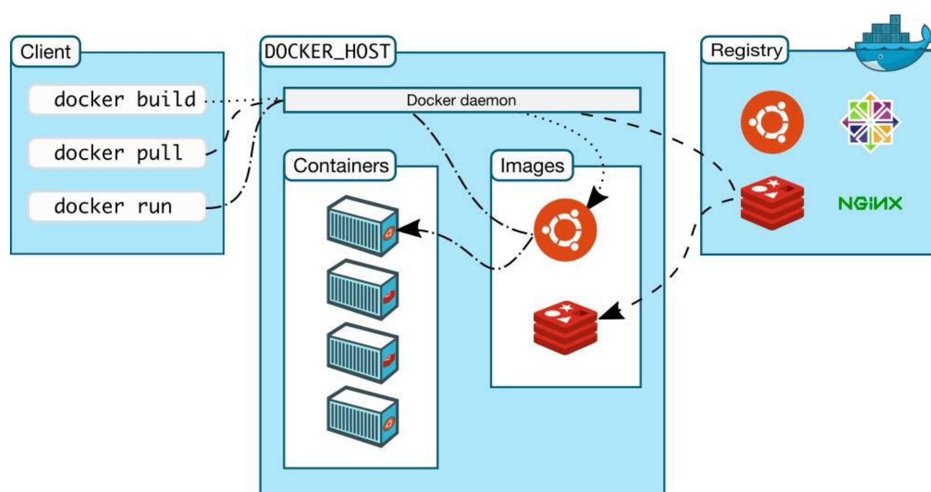
Το Union File System είναι σύστημα αρχείων που λειτουργεί δημιουργώντας στρώσεις οι οποίες είναι μόνο για ανάγνωση με αποτέλεσμα και είναι ένα ελαφρύ και γρήγορο σύστημα αρχείων.Αν θέλουμε να δημιουργήσουμε ένα νέο αρχείο στο container τότε το Docker φτιάχνει μια ακόμα στρώση σε αυτό το σύστημα αρχείων.

4.5 Docker Engine

Το Docker Engine είναι ο πυρήνας του οικοσυστήματος του Docker.Είναι μια εφαρμογή η οποία χρησιμοποιεί αρχιτεκτονική πελάτη - εξυπηρετητή και αποτελείται από τρία συστατικά μέρη.

1) Ο εξυπηρετητής είναι ένας δαίμονας που εκτελείται μακροπρόθεσμα και ονομάζεται dockerd.
2) Το Rest API που παρέχεται από τον dockerd δαίμονα ο οποίος ορίζει διεπαφές τις οποίες τις χρησιμοποιούν προγράμματα τα οποία θέλουν να επικοινωνήσουν με το δαίμονα ώστε να του δώσουν εντολές προς εκτέλεση.
3) Ένα πρόγραμμα πελάτη γραμμής εντολών το οποίο χρησιμοποιεί το Rest API του dockerd δαίμονα ώστε επικοινωνήσει με το δαίμονα.

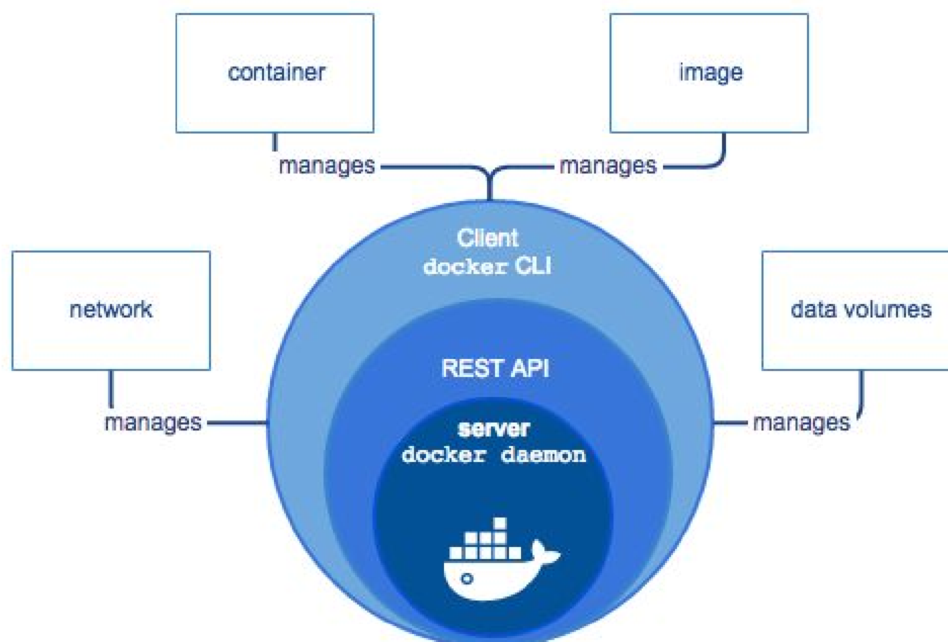
Ο δαίμονας και ο πελάτης Docker μπορεί να φιλοξενούνται στο ίδιο μηχάνημα ή είναι δυνατόν να συνδεθούμε με τον Docker πελάτη σε Docker δαίμονα ο οποίος βρίσκεται σε άλλο μηχάνημα.



Σχήμα 4.1 Αρχιτεκτονική Docker

4.6.2 Ο δαίμονας dockerd

Ο dockerd δαίμονας εκτελείται στον οικοδεσπότη με δικαιώματα διαχειριστή.Αναμένει να λάβει αιτήματα τα οποία περιέχουν εντολές και χρησιμοποιώντας το runC λογισμικό δημιουργεί images, containers, δίκτυα και μέσα αποθήκευσης εκτελώντας αυτές τις εντολες.Για το χτίσιμο images πρέπει πρώτα να λάβει ένα Dockerfile αρχείο που περιέχει τα βήματα δημιουργίας του image.



Σχήμα 4.2 Αρχιτεκτονική Docker Engine

4.6.3 Ο docker πελάτης

Η εφαρμογή πελάτη του docker είναι ο κυρίως τρόπος επικοινωνίας με έναν ή πολλούς dockerd δαίμονες.

4.6.4 Αποθετήριο Docker

Η docker πλατφόρμα προσφέρει υπηρεσία για την διαμοίραση των images.Τα images αποθηκεύονται σε ένα αποθετήριο το οποίο συνήθως είναι προσβάσιμο από άτομα τα οποία δεν έχουν εξουσιοδότηση.Το docker προσφέρει τρία είδη αποθετηρίων τα οποία είναι το Docker Hub, το Docker Cloud και το Docker Trusted Registry.Τα δύο πρώτα είναι δημόσια αποθετήρια όπου όλοι μπορούμε έχουμε πρόσβαση και μπορούμε να τα χρησιμοποιήσουμε.Οι εργοστασιακές ρυθμίσεις του Docker όταν αναζητούν κάποιο image το αναζητούν στο Docker Hub.Επίσης το Docker προσφέρει την δημιουργία ιδιωτικού αποθετήριο το οποίο μπορούμε να

χρησιμοποιήσουμε με τη χρήση της υπηρεσία Docker Datacenter (DDC).Εντολές όπως το docker run και docker pull αναζητούν τα images στο αποθετήριο που έχουμε ρυθμίσει το Docker να αναζητεί.Τέλος με την εντολή docker push μπορούμε να ανεβάσουμε δικά μας images μας στο αποθετήριο το οποίο το έχουμε ρυθμίσει.

4.7 Docker Objects

4.7.1 Images

Τα Docker Images είναι αρχεία μόνο προς ανάγνωση τα οποία περιέχουν σύστημα αρχείων, αρχεία ρυθμίσεων και εντολές με τα βήματα δημιουργίας ενός container.Για την δημιουργία images χρησιμοποιούμε αρχεία που ονομάζονται Dockerfiles.Τα Dockerfiles περιέχουν απλές εντολές στις οποίες περιγράφουν τα βήματα με τα οποία θα δημιουργηθεί ένα image.Ένα Dockerfile αρχείο σε συνδυασμό με την “docker build” εντολή δημιουργεί ένα image.Απαραίτητη παράμετρος στην εντολή “docker build” είναι το όνομα που θα έχει το image.Το αποτέλεσμα της “docker build” εντολής είναι ένα αρχείο μόνο προς ανάγνωση με πολλά επίπεδα το οποίο αναπαριστά το Dockerfile.Η κάθε εντολή απ το Dockerfile προσθέτει στο τελικό αρχείο ένα επιπλέον επίπεδο.Το κάθε επίπεδο αποθηκεύεται ξεχωριστά στην μνήμη cache.Όταν στο Dockerfile γίνονται αλλαγές με σκοπό να προσθέσουμε καινούργιες δυνατότητες ή να διορθώσουμε λάθη, τότε μόνο τα στρώματα τα οποία αλλάχτηκαν θα δημιουργηθούν εκ νέου ενώ για αυτά που παρέμειναν ίδια θα χρησιμοποιήσει τα στρώματα που είναι στην μνήμη cache.Για αυτό το λόγο τα images είναι μικρά, ελαφριά και γρήγορα όταν συγκρίνονται με hypervisor virtualization τεχνολογίες.

4.7.2 Containers

Ο container είναι ένα image το οποίο βρίσκεται σε εκτέλεση έχοντας τον δικό του κύκλο ζωής.Τα πιο συνηθισμένα στάδια στον κύκλο ζωής του είναι η δημιουργία του, η εκτέλεση του, η παύση του, το σταμάτημα του και η διαγραφή του.Με τη χρήση του docker πελάτη εκτός απ το να συνδεθούμε σε έναν container, μπορούμε να επέμβουμε στον κύκλο ζωής του.

Η ιδιομορφία των containers είναι ότι από κατασκευής τους δεν έχουν μόνιμη αποθήκευση με αποτέλεσμα όταν σταματούν να λειτουργούν, να χάνονται ότι αλλαγές έγιναν.Γενικότερα οι containers είναι απομονωμένοι από τους υπόλοιπους containers αλλά και από τον ηλεκτρονικό υπολογιστή στον οποίο φιλοξενούνται.Ωστόσο κατά την διάρκεια της δημιουργίας τους μπορούμε να ορίσουμε με παραμέτρους το πόσο απομονωμένος θα είναι ένας container.Όπως τι είδους αποθήκευσή θα έχει, σε ποιο δίκτυο θα ανήκει και εάν θα είναι απομονωμένος από άλλους containers ή/και από τη συσκευή στην οποία φιλοξενείται.

Σχήμα 4.3 Κύκλος ζωής ενός container

4.7 Dockerfile

Το Dockerfile είναι ένα απλό αρχείο κειμένου το οποίο περιέχει διάφορες εντολές για την δημιουργία images.Κάθε εντολή δημιουργεί ένα νέο επίπεδο στο σύστημα αρχείων του image.Το Docker Engine είναι υπεύθυνο για την ανάλυση και ερμηνεία των εντολών για την δημιουργία του image.Οι εντολές μπορούν για παράδειγμα να δημιουργήσουν ένα image απ την αρχή ή να κληρονομήσουν κάποιο άλλο, να ενεργοποιήσουν διαδικτυακές συσκευές του container, να τρέξουν εντολές είτε να προσθέσουμε μόνιμη αποθήκευση στον container.

Εάν μια εφαρμογή αποτελείται από ανεξάρτητα συστατικά στοιχεία, συνηθισμένη πρακτική είναι να δημιουργούμε διαφορετικά Dockerfiles αντί για ένα.Με αυτό το τρόπο μπορούμε να κάνουμε επαναχρησιμοποίηση των containers και να γίνει πιο εύκολη η επέκταση της εφαρμογής σε μεγάλη κλίμακα.

Με την εντολή FROM δηλώνουμε ότι το image που θα δημιουργηθεί είτε θα κληρονομήσει ένα άλλο image είτε να δημιουργήσει ένα εκ νέου.Για να δηλώσουμε ότι θα δημιουργηθεί ένα εκ νέου αρκεί να ορίσουμε την παράμετρο scratch στην εντολή FROM.

```
FROM ubuntu
CMD ls
```

Στο παρακάτω παράδειγμα το image κληρωνομεί το ubuntu image και τρέχει την εντολή ls.

5. Μονολιθικές Εφαρμογές

5.1 Εισαγωγή

Εφαρμογές όπου, με την πάροδο του χρόνου,ο πηγαίος τους κώδικας μεγαλώνει, αυξάνοντας την πολυπλοκότητα των εφαρμογών, στις οποίες για κάθε μικρή αλλαγή απαιτείται να γίνει μεταγλώττιση ολόκληρη η εφαρμογή και να τοποθετηθεί η νέα έκδοση στους διακομιστές, ονομάζονται μονολιθικές εφαρμογές.

5.2 Αρχιτεκτονική Μονολιθικών Εφαρμογών

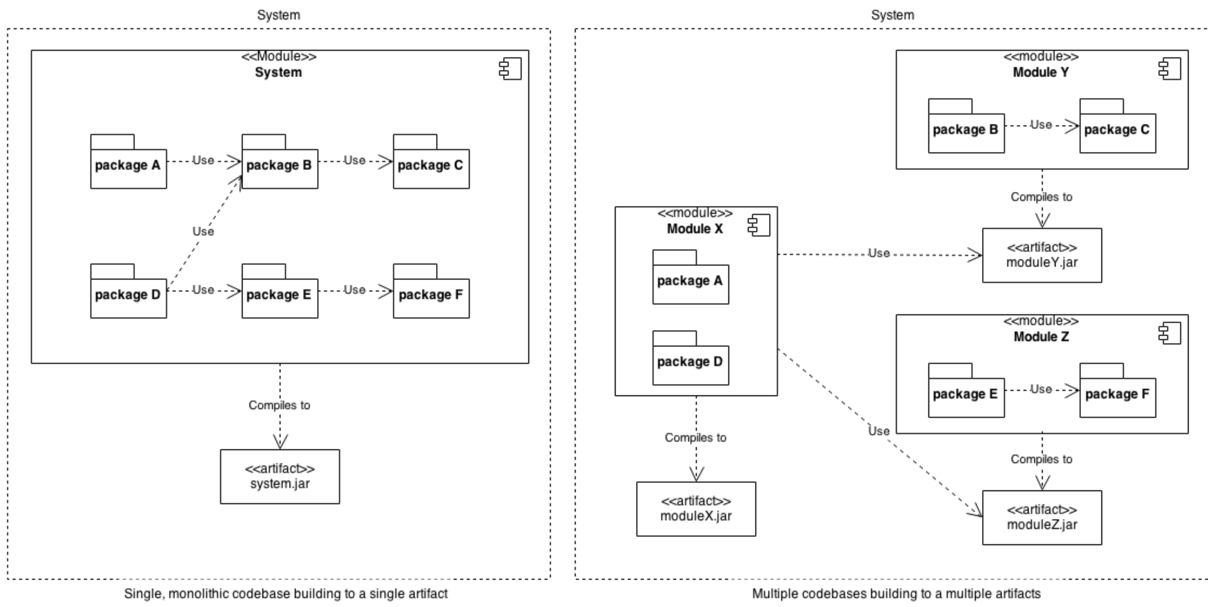
Ένα από τα πιο βασικά χαρακτηριστικά των μονολιθικών εφαρμογών είναι ότι ακόμα και αν αναπτυχθούν με αρθρωτές αρχιτεκτονικές παραμένουν μονολιθικές εφαρμογές διότι για να αξιοποιηθούν τα αρθρωτά τμήματα πρέπει όλα τα τμήματα της εφαρμογής να βρίσκονται στην ίδια έκδοση δηλαδή, όλα τα τμήματα της εφαρμογής να έχουν τον ίδιο κύκλο ζωής.Παρακάτω εξηγούμε τρεις τύπους διαφορετικών μονολιθικών εφαρμογών.

5.2.1 Αρθρωτός Μονόλιθος

Ο πηγαίος κώδικας μιας αρθρωτής μονολιθικής εφαρμογή είναι μοναδικός και αποτελείται από τμήματα κώδικα τα οποία έχουν εξαρτήσεις μεταξύ τους.Σε αυτή την περίπτωση το παραγόμενο αποτέλεσμα μετά τη μεταγλώττιση μιας αρθρωτής εφαρμογής είναι μόνο ένα αρχείο το οποίο αποτελεί και ολόκληρη την εφαρμογή.

Όπως βλέπουμε και στην παρακάτω εικόνα, αριστερά είναι μια αρθρωτή μονολιθική εφαρμογή στην οποία δεν διακρίνονται ξεκάθαρα τα τμήματα απ τα οποία αποτελείται.Το παραγόμενο αποτέλεσμα αυτού είναι μόνο ένα αρχείο.

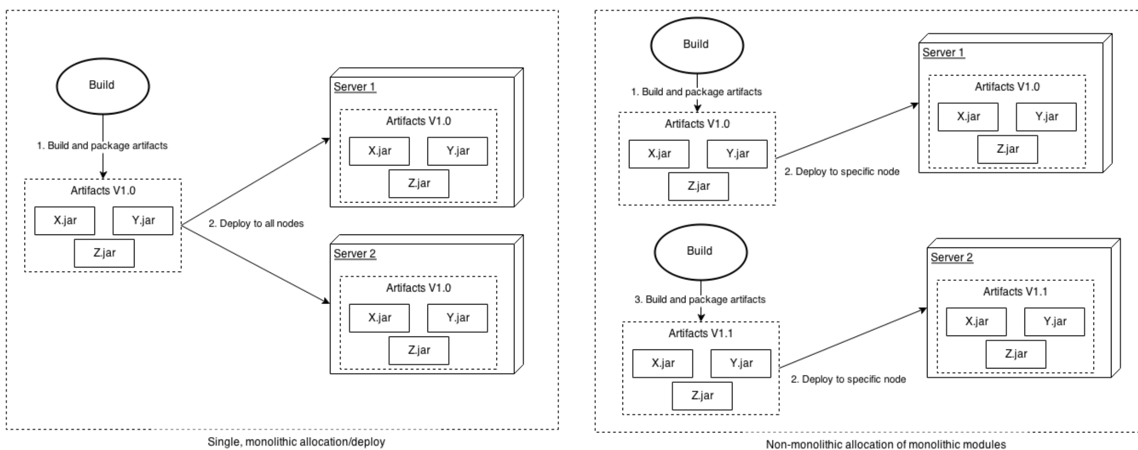
Στην παρακάτω εικόνα δεξιά φαίνεται ξεκάθαρα μια μη μονολιθική εφαρμογή η οποία οργανώνεται σε πολλαπλούς πηγαίους κώδικές και το παραγόμενο αποτέλεσμα μετά τη μεταγλώττιση μια τέτοιας εφαρμογής είναι διαφορετικά αρχεία τα οποία έχουν εξαρτήσεις το ένα από το άλλο.



Σχήμα 5.1 Αρθρωτές Εφαρμογές

5.2.2 Κατανεμημένος Μονόλιθος

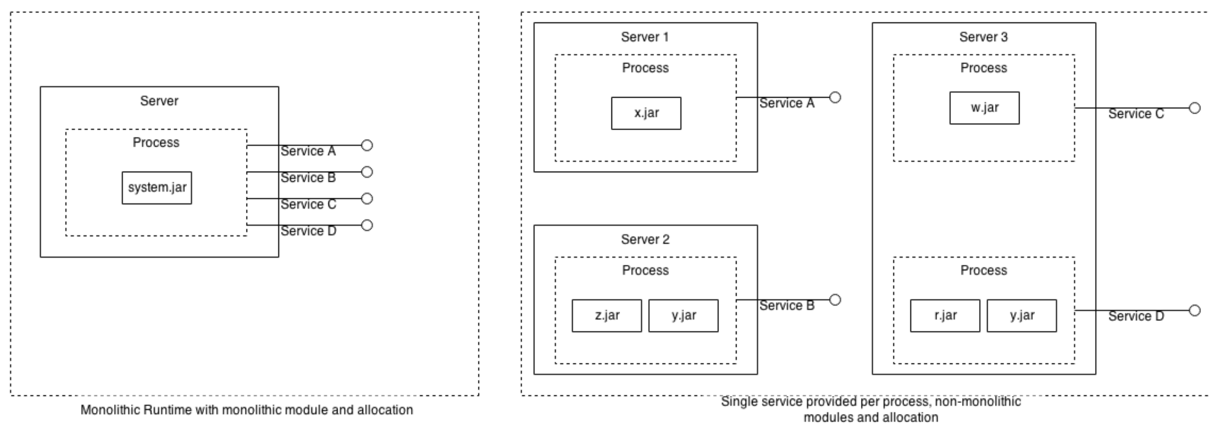
Ένας κατανεμημένος μονόλιθος αποτελείται από μια ή περισσότερες εφαρμογές οι οποίες είτε είναι μονόλιθοι είτε και όχι, αυτό όμως που έχει όμως σημασία είναι ότι το παραγόμενο αποτέλεσμα μετά τη μεταγλώττιση των εφαρμογών θεωρείται ότι ανήκει στην ίδια έκδοση.Το σύστημα για να είναι λειτουργικό πρέπει όλοι οι εξυπηρετητές να εκτελούν την ίδια έκδοση της εφαρμογής.Αυτό φαίνεται ξεκάθαρα στην παρακάτω εικόνα αριστερά όπου η ίδια έκδοση της εφαρμογής τοποθετείται στους εξυπηρετητές.Συνήθως η τοποθέτηση στους εξυπηρετητές γίνεται με το κλείσιμο ολόκληρου του συστήματος, την τοποθέτηση της καινούργιας έκδοσης της εφαρμογής και επανεκκίνηση των εξυπηρετητών.Στην παρακάτω εικόνα δεξιά η εφαρμογή είναι μια μη κατανεμημένη μονολιθική εφαρμογή όπου οι εξυπηρετητές μπορούν να φιλοξενήσουμε διαφορετικές εκδόσεις της εφαρμογής.Για την τοποθέτηση διαφορετικής έκδοσης στους εξυπηρετητές δεν είναι απαραίτητο κλείσει όλο το σύστημα παρά μόνο ο εξυπηρετητής που θα δεχτεί την ενημέρωση.



Σχήμα 5.2 Κατανεμημένες Εφαρμογές

5.2.3 Runtime Μονόλιθος

Το παραγόμενο αποτέλεσμα μετά τη μεταγλώττιση μιας στατικής runtime μονολιθική εφαρμογής είναι ένα αρχείο το οποίο εκτελείται σε μόνο μια διεργασία η οποία έχει πολλές εξωτερικές εξαρτήσεις και αναλόγως σε ποιον εξυπηρετητή τοποθετείται παρέχει και τις αντίστοιχες υπηρεσίες.Όπως βλέπουμε στην εικόνα αριστερά ο εξυπηρετητής φιλοξενεί μία runtime αρθρωτή και κατανεμημένη μονολιθική εφαρμογή παρέχοντας όλες τις υπηρεσίες της εφαρμογής.Στην δεύτερη περίπτωση δεξιά έχουμε διαφορετικούς εξυπηρετητές όπου προσφέρουν επιμέρους υπηρεσίες μιας μη αρθρωτής και κατανεμημένης μονολιθικής εφαρμογής.



Σχήμα 5.3 Runtime Εφαρμογές

5.3 Πλεονεκτήματα των μονολιθικών εφαρμογών

Παρότι ο όρος “μονολιθικές εφαρμογές” μας προϊδεάζει για κάτι κακό, αντιθέτως τέτοιους είδους εφαρμογές έχουν και πλεονεκτήματα τα οποία παρουσιάζονται παρακάτω.

1) Είναι εύκολη η ανάπτυξη μονολιθικών εφαρμογών επειδή πολλές πλατφόρμες έχουν δημιουργηθεί για αυτό το σκοπό.
2) Είναι πιο εύκολη η δημιουργία περιβάλλοντος εκτέλεσης της εφαρμογής στον ηλεκτρονικό υπολογιστή του προγραμματιστή.
3) Η διαδικασία τοποθέτησης της εφαρμογής σε εξυπηρετητές είναι πολύ απλή και εύκολη.Συνήθως επιτυγχάνεται με την απλή μεταφορά του τελικού αρχείου της εφαρμογής σε έναν κατάλογο στον εξυπηρετητή.
4) Η εφαρμογή μπορεί εύκολα να επεκταθεί σε μεγάλη κλίμακα απλά τοποθετώντας την σε πολλούς εξυπηρετητές πίσω από έναν load balancer ο οποίος θα μοιράζει την εισερχόμενη κίνηση στους εξυπηρετητές.
5) Διαφορετικές ομάδες εργάζονται και μοιράζονται τον ίδιο πηγαίο κώδικα.

5.4 Μειονεκτήματα Μονολιθικό Εφαρμογών

Με την πάροδο του χρόνου οι απαιτήσεις για τις παρεχόμενες υπηρεσίες μιας εφαρμογής αυξάνονται με αποτέλεσμα ο πηγαίος τους κώδικας να αυξάνει σε μέγεθος και παράλληλα να αυξάνουν και τα μέλη της ομάδας ανάπτυξης τους. Κάτω από αυτές τις προϋποθέσεις η μονολιθική αρχιτεκτονική αντιμετωπίζει τα προβλήματα που αναφέρονται παρακάτω.

5.4.1 Περιορισμένη Ευελιξία

Στις μονολιθικές εφαρμογές είναι δύσκολο να διαχειριστούμε τμήματα κώδικα ειδικά όταν δουλεύουν διαφορετικές ομάδες στα ίδια τμήματα.Ο πηγαίος κώδικας της εφαρμογής είναι μοναδικός όπου ακόμα και οι μικρές αλλαγές μπορούν να πυροδοτήσουν αλλαγές σε άλλα εξαρτώμενα σημεία στον κώδικα τα οποία δεν έχουμε προβλέψει.Για την τοποθέτηση μιας νέας έκδοσης στους εξυπηρετητές ακόμα και εάν έχουμε προσθέσει μόνο μικρές αλλαγές στον πηγαίο κώδικα, αναγκάζουν να γίνει ξανά η μεταγλώττιση της και να ακολουθηθεί η διαδικασία της τοποθέτησης στους εξηπηρετητές ξανά.Όλα τα παραπάνω έχουν ως αποτέλεσμα οι νέες εκδόσεις να τοποθετούνται στους εξυπηρετητές με αργό ρυθμό επηρεάζοντας την ευελιξία ανάπτυξης της εφαρμογής.

5.4.2 Μείωση παραγωγικότητας

Νέοι προγραμματιστές που προσθέτεται στην ομάδα ανάπτυξης της εφαρμογής έρχονται αντιμέτωποι με έναν αρκετά μεγάλο πηγαίο κώδικα όπου πολλές φορές τους εκφοβίζει με αποτέλεσμα να δυσκολεύονται στην κατανοήσει του.Η δομή του κώδικα έχει καθοριστικό ρόλο, στην κατανόηση ή μη σημαντικών τμημάτων της εφαρμογής.Όλα τα παραπάνω έχουν ως αποτέλεσμα να καθυστερούν την ανάπτυξη της εφαρμογής, τις τεχνικές εξασφάλισης της ποιότητας του κώδικα και την τοποθέτηση της στους εξυπηρετητές.

5.4.3 Δύσκολη δομή ομάδας

Ο διαχωρισμός της ομάδας ανάπτυξης σε υποομάδες καθώς και η ανάθεση καθηκόντων στα μέλη της κάθε υποομάδας δυσκολεύει.Οι πιο συνηθισμένοι τρόποι για τη διαίρεση των ομάδων είναι είτε με βάση την τεχνολογία που θα χρησιμοποιήσει η κάθε υποομάδα είτε τη γεωγραφική θέση της ομάδας.Ωστόσο, ο διαχωρισμός είτε με τεχνολογία είτε με γεωγραφική θέση ενδέχεται να μην είναι κατάλληλος σε όλες τις περιπτώσεις.Σε κάθε περίπτωση, η επικοινωνία μεταξύ των ομάδων μπορεί να είναι δύσκολη και αργή.Επιπλέον, είναι δύσκολο να οριστεί ως ιδιοκτήτης κάποια ομάδα, για θέματα όπως η ανάπτυξη μιας υπηρεσίας και η τοποθέτηση της νέας έκδοσης της υπηρεσίας στους εξυπηρετητές.Όταν η εφαρμογή παρουσιάσει κάποιο πρόβλημα, υπάρχει πάντα σύγχυση ποιος θα πρέπει να βρει

και να λύσει το πρόβλημα.Είτε υπεύθυνη για το πρόβλημα είναι η ομάδα που αναλαμβάνει την τοποθέτηση της εφαρμογής στους εξυπηρετητές είτε η τελευταία ομάδα που πρόσθεσε νέο κώδικα στην εφαρμογή.Η κατάλληλη δομή της ομάδας και η ιδιοκτησία μιας υπηρεσίας είναι πολύ σημαντικές για την ευελιξία.

5.4.4 Μακροπρόθεσμη δέσμευση

Η τεχνολογία που χρησιμοποιείται για την ανάπτυξη μιας εφαρμογής επιλέγεται κατά την ανάλυση των απαιτήσεων, λαμβάνοντας υπόψη την ωριμότητα της τρέχουσας τεχνολογίας εκείνη την εποχή.Όλες οι ομάδες του έργου πρέπει να χρησιμοποιήσουν τις ίδιες τεχνολογίες καθ 'όλη τη διάρκεια του κύκλου ζωής της εφαρμογής.Ωστόσο, είναι γνωστό ότι η τεχνολογία εξελίσσεται σε γρήγορους ρυθμούς.Έτσι, οι τεχνολογίες που επιλέχτηκαν κατά την ανάλυση απαιτήσεων μπορεί να είναι ξεπερασμένες και οι νέες τεχνολογίες να προσφέρουν καλύτερες λύσεις στα προβλήματα που λύνει η εφαρμογή.Στις μονολιθικές εφαρμογές, είναι πολύ δύσκολο και συνήθως επώδυνο να αλλάξουμε τις τεχνολογίες που χρησιμοποιήθηκαν σε άλλες νεότερες που λύνουν πιο εύκολα τα ίδια προβλήματα.

5.4.5 Περιορισμένη δυνατότητα κλιμάκωσης

Η κλιμάκωση μιας μονολιθικής εφαρμογής μπορεί να πραγματοποιηθεί με δύο τρόπους.Ο πρώτος τρόπος είναι τοποθετώντας την σε πολλούς εξυπηρετητές πίσω από έναν load balancer ο οποίος θα μοιράζει την εισερχόμενη κίνηση στους εξυπηρετητές.Ο δεύτερος τρόπος είναι όπως στην προηγούμενη περίπτωση, αλλά αυτή τη φορά θα γίνεται διαμέριση της πρόσβασης στη βάση δεδομένων αντί του αιτήματος του χρήστη.Και οι δύο προσεγγίσεις κλιμάκωσης βελτιώνουν την διαθεσιμότητα της εφαρμογής και τον αριθμό ταυτόχρονων χρηστών που μπορούν να εξηπηρετησούν.Ωστόσο,η απαίτηση όσον αφορά την κλιμάκωση για κάθε συστατικό ξεχωριστά μπορεί να είναι διαφορετική και δεν μπορεί να αντιμετωπιστεί χρησιμοποιώντας αυτήν την προσέγγιση. Επίσης, η πολυπλοκότητα μια μονόλιθου εφαρμογής παραμένει η ίδια επειδή τοποθετούμε την ίδια εφαρμογή σε πολλούς εξυπηρετητές.Οπότε εάν υπάρχει ένα πρόβλημα σε ένα στοιχείο, το ίδιο πρόβλημα μπορεί να επηρεάσει όλους τους διακομιστές που εκτελούν τα αντίγραφα της εφαρμογής.

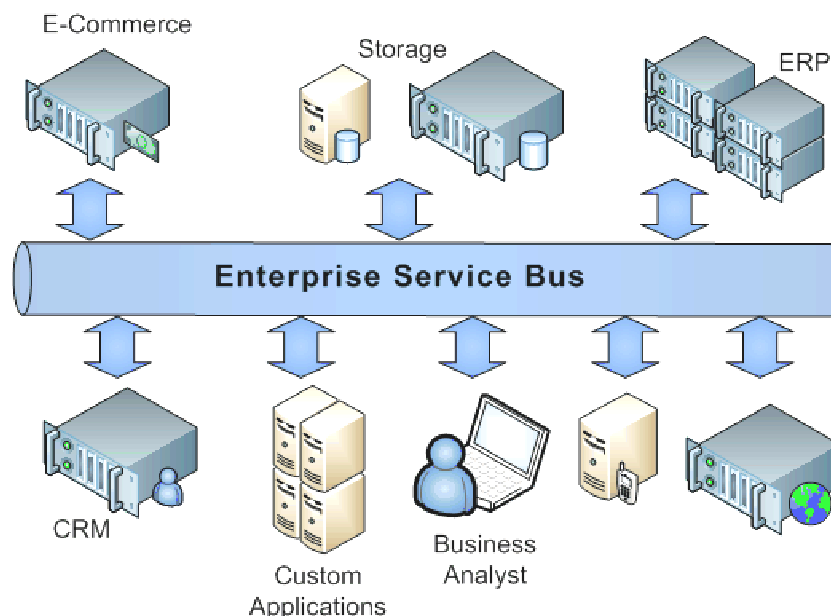
5.5 Αρχιτεκτονική SOA

Οι εφαρμογές που ως αρχιτεκτονική χρησιμοποιούν Service Oriented Architecture (SOA) αποτελούνται από αυτόνομες, διαλειτουργικές και επαναχρησιμοποιήσιμες υπηρεσίες.Οι υπηρεσίες συνήθως κατασκευάζονται με πρωτόκολλα διαδικτύου όπως το SOAP (Simple Object Access Protocol).Το SOA προσεγγίζει την λύση επιχειρηματικών προβλημάτων πιο αφαιρετικά με την δημιουργία λύσεων οι οποίες μπορούν να επαναχρησιμοποιήθουν από άλλες υπηρεσίες.Μια SOA υπηρεσία έχει

διπλό ρόλο, δηλαδή μπορεί να είναι δέκτης και παραλήπτης υπηρεσιών.Μια υπηρεσία ως δέκτης ενημερώνει τους παραλήπτες με πληροφορίες σχετικά με τις προσφερόμενες υπηρεσίες της.Στην SOAP τεχνολογία αυτή η ενημέρωση γίνεται μέσω του WSDL (Web Services Description Language) αρχείου.Παρομοίως μια υπηρεσίας ως παραλήπτης υπηρεσιών μπορεί να αναζητήσει τις προσφερόμενες υπηρεσίες και να διαλέξει την καταλληλότερη στέλνοντας ένα αίτημα στο δίκτυο.Από την άλλη λόγο της επεξεργασίας δεδομένων που αφορούν την αναζήτηση για τις προσφερόμενες υπηρεσίες αυξάνουν την κίνηση στο δίκτυο μειώνοντας την απόδοση της εφαρμογής.Για την αύξηση της απόδοσης, οι αυτόνομες υπηρεσίες είναι κατανεμημένες σε πολλούς διακομιστές επικοινωνώντας ασύχρονα.Τέλος λόγο της κατανεμημένης αρχιτεκτονικής η ασφάλεια των εφαρμογών SOA καθίσταται πολύ πιο πολύπλοκη.

5.5.1 Πρότυπο Enterprise Service Bus

Το Enterprise Service Bus (ESB) είναι ένα αρχιτεκτονικό πρότυπο που συνήθως αναφέρεται ως εφαρμογή SOA.Λειτουργεί ως κέντρο επικοινωνίας στο SOA και παρέχει ένα στρώμα ενοποίησης για υπηρεσίες.Το κυρίως όφελος από τη χρήση του ESB είναι η μειωμένη εξάρτηση μεταξύ των υπηρεσιών, καθώς επικοινωνούν μόνο μέσω του ESB το οποίο φροντίζει για τη διαβίβαση των αιτήσεων στον σωστό προορισμό.Αυτό είναι ιδιαίτερα χρήσιμο όταν ένα σύστημα αποτελείται από ένα μεγάλο αριθμό υπηρεσιών, όπου η διαχείριση των συνδέσεων από σημείο σε σημείο μεταξύ τους θα είναι ήταν δύσκολη.Από την άλλη πλευρά, η επικοινωνία μέσω της ESB εισάγει κάποια επιβάρυνση στην επικοινωνία το οποίο μπορεί τελικά να γίνει εμπόδιο.



Σχήμα 5.4 Αρχιτεκτονική Enterprise Service Bus

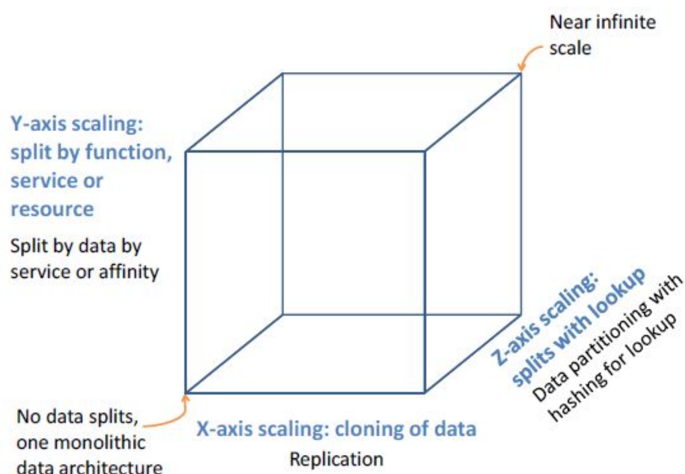
6. Microservices Εφαρμογές

6.1 Εισαγωγή

Η αρχιτεκτονική των microservices είναι μια προσέγγιση της SOA αρχιτεκτονικής αλλά αντί για τις μεγάλες εφαρμογές που παρέχουν πολλές υπηρεσίες αντικαθιστούν την κάθε υπηρεσία με μια εφαρμογή.Η κάθε εφαρμογή είναι μικρή, ελαφριά, κατανεμημένη, αυτόνομη, αντικαταστάσιμη, συντηρίσιμη, αναβαθμίσιμη, με τον δικό της κύκλο ζωής και πηγαίο κωδικά, η οποία επικεντρώνεται στο να προσφέρει μόνο μια υπηρεσία.Η κάθε μία εκτελείται σε ξεχωριστή διεργασία και επικοινωνούν μεταξύ τους με ελαφριά πρωτόκολλα διαδικτύου όπως είναι το REST.Λόγο της δομής των υπηρεσιών διασφαλίζεται πιο εύκολα η ποιότητα της κάθε υπηρεσίας, η διαθεσιμότητα της, η τοποθέτηση της στους εξυπηρετητές και η κλιμάκωση της σε πολλούς χρήστες.Παρόλα αυτά υπάρχει αύξηση στην πολυπλοκότητα της αρχιτεκτονικής εξαιτίας της επικοινωνίας μέσω δικτύου.Η ανεξαρτησία μεταξύ των υπηρεσιών οι οποίες έχουν τον δικό τους κύκλο ζωής αλλά και η επικοινωνίας μεταξύ τους που γίνεται μόνο μέσω δικτύου δίνει την δυνατότητα να χρησιμοποιηθούν διαφορετικές τεχνολογίες για την ανάπτυξη τους.Η αρχιτεκτονική των microservices είναι μια προσέγγιση για την διαμέριση μιας μονολιθικής εφαρμογής σε μικρότερα τμήματα τα οποία είναι ανεξάρτητα και λειτουργούν όλα μαζί ως μία εφαρμογή.

6.2 Μοντέλο Scale Cude

Τα microservices χρησιμοποιούν κατά κόρον το μοντέλο scale cube το οποίο έρχεται να δώσει λύσεις σε προβλήματα όπως η ευελιξία, η παραγωγικότητα και η κλιμάκωση.Το scale cube ορίζει την κλιμάκωση μιας εφαρμογής σε τρεις διαστάσεις.Η κλιμάκωση μπορεί γίνει σε οποιαδήποτε από τις τρεις κατευθύνσεις αναλόγως τις απατήσεις του συστήματος.Παρακάτω αναλύουμε το τι είδους κλιμάκωση προσφέρει η κάθε κατεύθυνση.



Σχήμα 6.1 Αρχιτεκτονική Scale Cube

6.2.1 Κλιμάκωση κατά τον άξονα X

Αυτό το είδους κλιμάκωσης επιτυγχάνετε αντιγράφοντας την εφαρμογή και τα δεδομένα της σε πολλούς διακομιστές.Τα εισερχόμενα αιτήματα επικοινωνούν με έναν load balancer ο οποίος διαμοιράζει τα αιτήματα σε όλους τους διακομιστές.Όλοι οι διακομιστές εκτελούν την πιο πρόσφατη έκδοση της εφαρμογής η οποία έχει πρόσβαση σε όλα τα απαιτούμενα δεδομένα με αποτέλεσμα να μην υπάρχει πρόβλημα ποιος διακομιστής θα εξυπηρετήσει το εισερχόμενο αίτημα.Το ζητούμενο αποτέλεσμα είναι να εξυπηρετούμε όσο το δυνατόν περισσότερα αιτήματα ανά πάσα στιγμή.Καθώς αυξάνεται ο αριθμός των αιτημάτων είναι εύκολο να γίνει κλιμάκωση κατά τον άξονα X απλά προστέθοντας επιπλέον κλώνους της εφαρμογής και των δεδομένων.Το αρνητικό σε αυτό το είδος της κλιμάκωσης είναι ότι, καθώς αυξάνονται τα δεδομένα της εφαρμογής, η κλιμάκωση της γίνεται όλο και πιο δύσκολη.Ένα άλλο πρόβλημα είναι ότι, η μεγάλη διακύμανση στη συχνότητα που τα εισερχόμενα αιτήματα παραλαμβάνονται, δεν βοηθούν στην κλιμάκωση όταν αυτά κατανέμονται στους διακομιστές με τον ίδιο τρόπο.

6.2.2 Κλιμάκωση κατά τον άξονα Z

Η κλιμάκωση στον άξονα Z, επιτυγχάνεται διαχωρίζοντας τα εισερχόμενα αιτήματα, βασισμένα σε διάφορα κριτήρια ή πληροφορίες σχετικά με τον αιτούντα.Σε αντίθεση με την κλιμάκωση στον άξονα X, οι εξυπηρετητές είναι υπεύθυνοι για την εξυπηρέτηση διαφορετικών αιτημάτων.Δηλαδή όλοι οι διακομιστές δεν μπορούν να εξυπηρετήσουν τα ίδια τύπου αιτημάτων.Σε μερικούς από τους διακομιστές έχουν προστεθεί επιπλέον δυνατότητες.Με τη χρήση αυτής της κλιμάκωσης, σε ορισμένες ομάδες πελατών μπορούν να παρέχονται πρόσθετες λειτουργίες, δοκιμάζοντας νέες δυνατότητες σε ένα μικρο σύνολο πελατών μειώνοντας τον κίνδυνο λάθους.Η κλιμάκωση του άξονα Z βοηθά στην απομόνωση σφαλμάτων και στην επεκτασιμότητα των συναλλαγών.

6.2.3 Κλιμάκωση κατά τον άξονα Y

Η κλιμάκωση κατά τον άξονα Y διασπά την εφαρμογή σε μικρότερα τμήματα που ονομάζονται υπηρεσίες.Ο διαχωρισμός γίνεται, είτε με βάση τον τύπων τον δεδομένων, είτε με βάση με την επεξεργασία που εκτελείται στα δεδομένα, είτε με τον συνδυασμό και των δύο.Στον άξονας Y δεν αντιγράφουμε ολόκληρη την εφαρμογή και τα δεδομένα όπως στο άξονα X και Z αλλά αντιγράφουμε συγκεκριμένες λειτουργίες στους διακομιστές με την μορφή υπηρεσιών.Το κυρίως πλεονέκτημα αυτής της κλιμάκωσης είναι ότι κάθε αίτημα κλιμακώνεται και αντιμετωπίζεται διαφορετικά ανάλογα με την αναγκαιότητά του.Με την διάσπαση της λογικής και των δεδομένων της εφαρμογής, οι προγραμματιστές μπορούν να εστιάσουν και να εργαστούν σε ένα μικρό τμήμα κάθε φορά αυξάνοντας την παραγωγικότητα καθώς και την ευκινησία.Τα σφάλματα σε μια υπηρεσία είναι

απομονωμένα και αντιμετωπίσιμα χωρίς να επηρεαστεί η υπόλοιπη εφαρμογή.Ωστόσο, η κλιμάκωση κατά μήκος του άξονα Y μπορεί να είναι δαπανηρή σε σύγκριση με την κλιμάκωση σε άλλες διαστάσεις.

6.3 Κοινόχρηστες βιβλιοθήκες

Οι βιβλιοθήκες είναι απ τους πιο συνηθισμένους τρόπους για την διαμοίραση έτοιμων λύσεων ανάμεσα σε υπηρεσίες και ομάδες.Για τις γλώσσες προγραμματισμού αυτό είναι ένα πλεονέκτημα, ωστόσο οι κοινόχρηστες βιβλιοθήκες στην αρχιτεκτονική των microservices, δεν παρέχουν τεχνολογική ετερογένεια.Η ανεξάρτητη ανάπτυξη, κλιμάκωση και συντήρηση εφαρμογών μπορεί να επιτευχθεί μόνο εάν οι βιβλιοθήκες είναι δυναμικά συνδεδεμένες.Σε περιπτώσεις όπου οι εφαρμογές δεν είναι δυναμικές συνδεδεμένες, οποιαδήποτε μικρή αλλαγή στη βιβλιοθήκη, οδηγεί σε αναδιάταξη ολόκληρης της εφαρμογής και την ανάγκη επανατοποθέτησης της εφαρμογής στους διακομιστές.Οι κοινόχρηστες βιβλιοθήκες είναι μια μορφή σύζευξης και πρέπει να αποφεύγεται.Η αποσύνθεση μιας εφαρμογής σε μεμονωμένες υπηρεσίες έχει πλεονεκτήματα, όπως την ευέλικτη, την ανεξάρτητη ανάπτυξη, την κλιμάκωση, την συντήρηση και την τοποθέτηση της εφαρμογής στους διακομιστές εύκολη ανάθεσή αρμοδιοτήτων στις ομάδες.Τέλος, τα microservices χρησιμοποιούν την τεχνική αποσύνθεσης που προτείνεται από scale cube μοντέλο.

6.4 Χαρακτηριστά Microservices

6.4.1 Μοναδική Ευθύνη

Οι υπηρεσίες στην αρχιτεκτονική των microservices επικεντρώνονται στο να προσφέρουν συγκεκριμένες λύσεις.Η κάθε υπηρεσία, έχει τον δικό της πηγαίο κώδικα, ο οποίος είναι μικρός σε έκταση, και έχει τον δικό της κύκλο ζωής.Το γεγονός ότι τα microservices είναι μικρά και δεν εξαρτώνται το ένα από το άλλο, δεν τα εμποδίζει στο να μεγαλώνουν πολύ.Για την προσθήκη μιας εντελώς νέας λειτουργικότητας, συνήθως θα δημιουργηθεί μια νέα υπηρεσία.Τέλος, τα microservices, σε αντίθεση με τους μονόλιθους, μπορούν να έχουν πιο εύκολα μια αρθρωτή δομή επειδή, δεν υπάρχει εξάρτηση σε επίπεδο πηγαίου κώδικα αλλά, επικοινωνούν μεταξύ τους απομακρυσμένα μέσω δικτύου.

6.4.2 Αυτονομία

Στην αρχιτεκτονική των microservices οι υπηρεσίες εκτελούνται σε διαφορετικές διεργασίες και διακομιστές.Λόγου ότι η επικοινωνία μεταξύ τους, πραγματοποιείται μόνο μέσω του δικτύου(δημιουργώντας κάποια επιβάρυνση), αποφεύγουμε την στενή σύζευξη μεταξύ των υπηρεσιών, διευκολύνοντας στη κατανόηση των τμημάτων απ τα οποία αποτελείται ολόκληρη η εφαρμογή.Σε αντίθεση με άλλες αρχιτεκτονικές, η κάθε υπηρεσία χρησιμοποιεί την δικιά της βάση δεδομένων

διατηρώντας την ακεραιότητα των δεδομένων.Καμία άλλη υπηρεσία δεν μπορεί να παραποιήσει απευθείας τα δεδομένα, παρά μόνο μέσω της υπηρεσίας.Η υπηρεσία χρησιμεύει ως πύλη πρόσβασης για άλλες υπηρεσίες που επιθυμούν να χρησιμοποιούσουν τα δεδομένα της υπηρεσίας.

6.4.3 Ανομοιογένεια

Η ανεξαρτησία μεταξύ των υπηρεσιών δίνει την δυνατότητα, να επιλεχθούν διαφορετικές τεχνολογίες, κατάλληλες για την κάθε υπηρεσία, με σκοπό την βέλτιστη λύση των προβλημάτων.Ο μόνος περιορισμός των υπηρεσιών είναι ότι πρέπει να προσφέρουν μία διεπαφή προγραμματισμού η οποία θα χρησιμοποιείται ως μέσω επικοινωνίας των υπηρεσιών και δεν θα εξαρτάται από την γλώσσα προγραμματισμού.Μπορούν να χρησιμοποιηθούν διαφορετικές γλώσσες προγραμματισμού αλλά και διαφορετικές βάσεις δεδομένων.

6.4.4 Ελαστικότητα

Τα microservices τοποθετούνται σε πολλούς διακομιστές και η επικοινωνία μεταξύ τους γίνεται απομακρυσμένα μέσω δικτύου.Οι υπηρεσίες σχεδιάζονται με τέτοιο τρόπο ώστε να απομονώνονται τα λάθη και να μην επηρεάζουν τις υπόλοιπες υπηρεσίες.Η ομάδα που είναι υπεύθυνη της λανθάνουσας υπηρεσίας πρέπει να ανταποκριθεί ώστε διορθωθεί το πρόβλημα.

6.4.5 Κλιμάκωση

Η κάθε υπηρεσία της εφαρμογής μπορεί να κλιμακωθεί ανεξάρτητα από τις υπόλοιπες ανάλογα με τις ανάγκες της.Οι υπηρεσίες που έχουν περισσότερη κίνηση μπορεί να διανεμηθούν περισσότερες φορές στους διακομιστές και αυτές που έχουν λιγότερη κίνηση να διανεμηθούν σε λιγότερους διακομιστές.

6.4.6 Εύκολη εγκατάσταση

Η τοποθέτηση των υπηρεσιών στους διακομιστές αυτοματοποιείται και μπορεί να ολοκληρωθεί χωρίς ανθρώπινη παρέμβαση.Η ανεξαρτησία μεταξύ των υπηρεσιών, που έχουν με τον δικό τους κύκλο ζωής και η αυτοματοποίηση κάνουν πιο εύκολη και γρήγορη την συνεχή τροποποίηση και ανάπτυξη της εφαρμογής ελαχιστοποιώντας τους κινδύνους λάθους.

6.4.7 Επαναχρησιμοποίηση

Η επαναχρησιμοποίηση των υπηρεσιών επιτυγχάνεται με την διαμοίραση λειτουργικότητας μέσω διαδικτυακών διεπαφών.Με αυτό το τρόπο επιτυγχάνεται η επαναχρησιμοποίηση μιας υπηρεσίας με ποικίλους τρόπους.Οι μέθοδοι της υπηρεσίας μπορούν να καλούνται από μια άλλη υπηρεσία από οπουδήποτε στο κόσμο.

6.4.8 Αντικατάσταση

Ο πηγαίος κώδικας των υπηρεσιών, αναπτύσσεται με την χρήση διεπαφών, οι οποίες ορίζουν την λειτουργικότητα των υπηρεσιών.Οι διεπαφές και ο μικρός σε έκσταση πηγαίος κώδικας, επιτρέπει στους προγραμματιστές να αντικαταστήσουν την υλοποίηση μια υπηρεσίας, άπλα αναπτύσσοντας την υπηρεσία βασιζόμενες στις διεπαφές, προσφέροντας ακριβώς τις ίδιες λειτουργίες.Κατά το χρόνο της μεταγλώττισης τους δεν υπάρχουν εξαρτήσεις μεταξύ των υπηρεσιών παρά μόνο στο χρόνο εκτέλεσης.Αυτό δίνει την δυνατότητα τοποθέτησης των υπηρεσιών στους διακομιστές, ανεξάρτητα από τις άλλες υπηρεσίες, μειώνοντας την ανάγκη τοποθέτησης όλων των υπηρεσιών ταυτόχρονα στους διακομιστές.

6.4.9 Μικρές Ομάδες

Οι υπηρεσίες οργανώνονται με σκοπό να λύνουν συγκεκριμένα επιχειρηματικά προβλήματα.Η δομή της κάθε ομάδα αλλά και ο αριθμός των ομάδων είναι ένα αντίγραφο της επικοινωνιακής δομής της επιχείρησης.Η κάθε ομάδα είναι ιδιοκτήτης της κάθε υπηρεσία οι οποίες έχουν τον δικό τους κύκλο ζωής.Αυτό σημαίνει ότι έχουν τον πλήρη έλεγχο της υπηρεσίας από το σχεδιασμό, την ανάπτυξη και την αντιμετώπιση προβλημάτων.Είναι αυτοί που αντιμετωπίζουν τα σφάλματα της υπηρεσίας και παίρνουν τις αποφάσεις για την υλοποίησης της υπηρεσίας.

7. Java EE

7.1 Εισαγωγή

Σε αυτό το κεφάλαιο κάνουμε μια εισαγωγή στις βασικές τεχνολογίες της Java EE.Αρχικά γίνεται μια ιστορική αναδρομή της πλατφόρμας, δίνεται ο ορισμός της και παράλληλα αναλύουμε την αρχιτεκτονική της αναφέροντας τις βασικές της τεχνολογίες απ τις οποίες αποτελείται.

7.2 Ιστορική Αναδρομή

Η πρώτη επίσημη κυκλοφορία της Java EE πλατφόρμας ήταν το 1999 με αρχική της ονομασία “Java 2 Platform, Enterprise Edition” ή αλλιώς με την κωδική ονομασία “J2EE”.Τον Μάιο του 2005 από την έκδοση 1.5 μέχρι και την έκδοση 1.8 άλλαξε το όνομά της σε “Java Platform, Enterprise Edition” με την κωδική ονομασία “Java EE”.Τον Ιούνιο του 2013 είναι πλέον στην έκδοση 1.7 και αποτελείται από 40 προδιαγραφές.Η Oracle τον Σεπτέμβριο του 2017 ανακοίνωσε την νέα έκδοση της, την 1.8 ανακοινώνοντας παράλληλα και την δωρεά της στο Eclipse Foundation.Για να επιτευχθεί η δωρεά της, το Eclipse Foundation δημιούργησε ένα νέο έργο το οποίο θα φιλοξενήσει την Java EE και άλλες τεχνολογίες του οικοσυστήματος

της.Αυτό το έργο ονομάστηκε “Eclipse Enterprise for Java” ή αλλιώς με το ακρωνύμιο “EE4J”.Κατά την διάρκεια εγγραφής αυτής της πτυχιακής εργασίας βρίσκεται σε ανάπτυξη η Java EE κάτω από την εποπτεία του EE4J όπου θα ερωτηθεί εκ νέου αν θα πρέπει να αλλάξει το όνομα της Java EE και ποιο θα είναι αυτό.Είναι γεγονός ότι η πλατφόρμα αναπτύσσεται σε γρήγορους ρυθμούς όπου η τρέχουσα έκδοση της Java EE είναι η όγδοη.Ωστόσο οι πληροφορίες στα επόμενα κεφάλαια αναφέρονται στην έβδομη έκδοση,οι οποίες θα παραμείνουν έγκυρες και στις επόμενες εκδόσεις της ή μπορεί να χρειαστούν μόνο μικρές αλλαγές.

7.3 Πως γίνεται η ανάπτυξη της πλατφόρμας;

Μέχρι πρότινος η ανάπτυξη της Java EE γινόταν μέσω της Java Community Process (JCP) η οποία είναι υπεύθυνη για όλες τις Java τεχνολογίες.Η JCP αποτελείται από ειδικούς και από μέλη της κοινότητας τα οποία έχουν γνώσεις σε τεχνολογικά θέματα.Οι ειδικοί πάνω σε διάφορες τεχνολογίες δημιουργούν προτάσεις για προδιαγραφές οι οποίες ονομάζονται Java Specification Requests (JSRs).Ο ρόλος της κοινότητας είναι να συμβάλει ότι αυτές οι προδιαγραφές είναι σταθερές και ότι υπάρχει συμβατότητα μεταξύ διαφορετικών πλατφορμών.Πλέον υπεύθυνη για την ανάπτυξη της είναι το Eclipse Foundation (EE4J) όπου επιτρέπει την ανάπτυξη των προδιαγραφών κάνοντας χρήση πιο ευέλικτων διαδικασιών, πιο ευέλικτης αδειοδότησης και πιο ανοιχτή διαδικασία διακυβέρνησης για την εξέλιξη της πλατφόρμας.

7.4 Τι είναι η Java EE;

Η Java EE είναι ένα σύνολο από προδιαγραφές όπου επεκτείνουν την Java SE με σκοπό την απλούστερη ανάπτυξη πολύπλοκων, ασφαλών, φορητών και κατανεμημένων εφαρμογών διαδικτύου σε μεγάλη κλίμακα θεσπίζοντας ταυτόχρονα έναν κοινό θεμέλιο για τα διάφορα δομικά στατιστικά της πλατφόρμας.Η υλοποίηση των προδιαγραφών στον τελικό χρήστη έρχεται με την μορφή των Application Servers.Οι Application Servers προσφέρουν λύσεις μέσα από την χρήση APIs στα πιο συνηθισμένα προβλήματα που συναντούν οι προγραμματιστές κατά την διάρκεια ανάπτυξης εφαρμογών διαδικτύου όπως στη χρήση των σχεδιαστικών προτύπων κάνοντας πιο εύκολη τη χρήση των πιο βέλτιστων λύσεων που είναι κοινά αποδεκτές.

7.5 Τι είναι ο Application Server;

Οι Application Servers υλοποιούν το σύνολο από τις προδιαγραφές που ορίζει η Java EE και αποτελούνται από components οι οποίοι ενθυλακώνονται μέσα σε containers(όχι docker containers).Οι Application Servers δημιουργούν ένα επίπεδο αφαιρετικότητας μεταξύ του λειτουργικού συστήματος και των εφαρμογών που

αναπτύσσουμε προσφέροντας λειτουργίες που χρησιμοποιούνται συχνά σε επιχειρησιακές εφαρμογές.Ως επιχειρησιακές εφαρμογές ορίζουμε τις εφαρμογές όπου απαιτείται να εγγυηθούν για την αξιοπιστία, την προσβασιμότητα, την αποτελεσματικότητα και την ασφάλεια τους εξυπηρετώντας ταυτόχρονα πολλούς χρήστες.

7.6 Μοντέλο Ανάπτυξης Java EE Εφαρμογών

Το μοντέλο της πλατφόρμας ορίζει μια αρχιτεκτονική για την ανάπτυξη εφαρμογών ως εφαρμογών πολλών στρωμάτων διαχωρίζοντας την εργασία ανάπτυξης σε δύο μέρη

- Στην επιχειρηματική λογική και στην λογική παρουσίασης οι οποίες αναπτύσσονται από τους προγραμματιστές.
- Στις προσφερόμενες υπηρεσίες συστήματος από την Java EE πλατφόρμα.

Ο προγραμματιστής κάνοντας χρήση των προσφερόμενων υπηρεσιών της πλατφόρμας μπορεί να προσφέρει λύσεις για προβλήματα που αντιμετωπίζουμε κατά την ανάπτυξη εφαρμογών πολλαπλών στρωμάτων.

Ο σαφής διαχωρισμός της αναπτυσσόμενης εφαρμογής από την υποδομή στην οποία φιλοξενείται δημιουργούν επιπλέον πλεονεκτήματα τα οποία είναι τα εξής

1. **Ακεραιότητα δεδομένων και κώδικα**
   Η αναπτυσσόμενη εφαρμογή δεν περιλαμβάνει κώδικα από τις υποδομές με αποτέλεσμα οι ενημερώσεις των εφαρμογών να είναι εγγυημένες, ελαχιστοποιώντας τον κίνδυνο ασυμβατότητας παλιάς και νέας έκδοσης.
2. **Ασφάλεια**
   Έχοντας ένα κεντρικό σημείο πρόσβασης στα δεδομένα (π.χ. βάση δεδομένων) μειώνει την ευθύνη αυθεντικοποίησης σε δυνητικά μη ασφαλές εφαρμογές πελάτη.
3. **Απόδοση**
   Περιορίζοντας την επισκεψιμότητα του πελάτη-διακομιστή, βελτιώνει την απόδοση των μεγάλων εφαρμογών
4. **Συναλλαγές**
   Ο διακομιστής κάνει πολύπλοκη παραγωγή κώδικα, ενώ οι προγραμματιστές μπορούν να επικεντρωθούν στην εγγραφή επιχειρησιακής λογικής.

7.7 Αρχιτεκτονική

Το μοντέλο αρχιτεκτονικής που ακολουθεί η Java EE είναι βασισμένο σε components όπου το κάθε component προσφέρει διαφορετικές υπηρεσίες.Η Java EE παρέχει αυτές τις υπηρεσίες με την μορφή containers(όχι docker containers) τα οποία είναι περιβάλλοντα φιλοξενίας και εκτέλεσής των components.Οι containers είναι σε θέση να λειτουργούν ανεξαρτήτως πλατφόρμας κάνοντας εύκολη την ανάπτυξη εφαρμογών χωρίς να χρειάζεται να αναπτύξουμε αυτές τις υπηρεσίες οι ίδιοι.

7.7.1 Java EE Components

Τα Java EE Components είναι αυτόνομες ανεξάρτητες οντότητες λογισμικού οι οποίες μπορούν να προσφέρουν διαχείριση συναλλαγών, πολυνηματικό προγραμματισμό, διαχείριση πόρων και άλλων λειτουργιών χαμηλού επιπέδου.Η κάθε οντότητα επικοινωνεί με τις υπόλοιπες και κάθε μία περιλαμβάνει τις δικές της κλάσεις και αρχεία.Οι προδιαγραφές της Java EE ορίζουν τα ακόλουθα Java EE Components.

- Εφαρμογές πελάτη όπως Java Swing,JavaFX και Applets.
- Εφαρμογές Server όπως Java Servlet, JavaServer Faces (JSF) ,JavaServer Pages (JSP) και Enterprise Java Beans (EJB)

Οι διαφορές των Java EE Components και των κλασσικών προγραμμάτων σε Java είναι ότι τα Java EE Components χρησιμοποιούνται ώστε να δημιουργήσουν Java EE εφαρμογές οι οποίες επαληθεύονται ώστε να είναι σωστά διαμορφωμένες και να ακολουθούν τις προδιαγραφές της Java EE οι οποίες φιλοξενούνται, εκτελούνται και διαχειρίζονται από τους Application Servers.

7.7.2 Java EE Containers

Τα containers είναι δυναμικά περιβάλλοντα φιλοξενίας των components προσφέροντάς τους έτοιμες υπηρεσίες για την αλληλεπίδραση μεταξύ τους αλλά και την επικοινωνία με λειτουργικότητα χαμηλότερου επιπέδου.Για την φιλοξενία και εκτέλεση των components η Java EE ορίζει στις προδιαγραφές της τα παρακάτω containers.

- **EJB Container** το οποίο διαχειρίζεται την εκτέλεση των Enterprise Java Beans.
- **Web Container** το οποίο διαχειρίζεται την εκτέλεση των Web Pages, Servlets, Java Server Pages, Java Server Faces και μερικά από τα EJB Components.
- **Application Client Container** το οποίο διαχειρίζεται την εκτέλεση των προγραμμάτων πελάτη τα οποία εκτελούνται στη μεριά του πελάτη.
- **Applet Container** το οποίο διαχειρίζεται την εκτέλεση των Applets.Εκτελείται στην μεριά του πελάτη μέσα από έναν φυλλομετρητή ο οποίος έχει και το αντίστοιχο Java Plugin.

Τα containers έχουν δυναμικές διεπαφές ώστε κατά την διάρκεια συναρμολόγησης τους να ρυθμίζονται ανάλογα με τις απαιτήσεις της εκάστοτε εφαρμογής.Οι ρυθμίσεις που ορίζονται αφορούν το κάθε component της Java EE και μπορούν να τροποποιήσουν τις παρεχόμενες υπηρεσίες των Java EE Servers.Μερικές από τις ρυθμίσεις περιλαμβάνουν πληροφορίες σχετικά με την ασφάλεια, τον κύκλο ζωής των EJBs και Servlets, την σύνδεση με την βάση δεδομένων και την διαχείριση πόρων, διαχείριση συναλλαγών.Πριν την εκτέλεση μιας Java EE εφαρμογής πρέπει να συναρμολογηθεί σε ένα ή περισσότερα Java EE Modules και να τοποθετηθούν στα αντίστοιχα containers για την εκτέλεση τους.

7.7.3 Java EE Modules

Το παραγόμενο αποτέλεσμα μετά την ανάπτυξη μιας Java EE εφαρμογής είναι τα Java EE Modules.Τα Java EE Modules είναι τα τελικά αρχεία που τοποθετούνται στους αντίστοιχους containers προς εκτέλεση.Αποτελούνται από ένα ή περισσότερα components των αντίστοιχων containers και προαιρετικά περιλαμβάνουν ένα αρχείο με οδηγίες για τη σωστή ρύθμισής τους που ονομάζεται Deployment Descriptor.Οι προδιαγραφές της Java EE ορίζει τα παρακάτω modules.

- **EJB Modules** εμπεριέχουν τα EJBs και προαιρετικά ένα EJB Deployment Descriptor. Συσκευάζονται σε ένα αρχείο με κατάληξη .JAR.
- **Web Modules** εμπεριέχουν τα Servlet, Web αρχεία πολυμέσων και προαιρετικά ένα Web Deployment Descriptor.Συσκευάζονται σε ένα JAR αρχείο το οποίο έχει κατάληξη .WAR (Web Archive).
- **Application Client Modules** περιέχουν class αρχεία και προαιρετικά ένα application client deployment descriptor.Συσκευάζονται σε ένα αρχείο με κατάληξη .JAR.
- **Resource Adapter Modules** περιέχουν αρχεία που έχουν interfaces, classes, native libraries και προαιρετικά ένα resource adapter deployment descriptor.Συσκευάζονται σε ένα JAR αρχείο το οποίο έχει κατάληξη .RAR.

Οι συχνά χρησιμοποιημένες μορφές των Java EE Modules είναι τα WAR Modules και EAR Modules.Τα EAR Modules μπορούν να εμπεριέχουν όλα τα υπόλοιπα modules.

7.8 Java EE Τεχνολογίες

Η Java EE αποτελείται από πολλές τεχνολογίες.Παρακάτω θα αναφέρουμε τρεις απ τις πιο βασικές που χρησιμοποιούνται για την ανάπτυξη διαδικτυακών εφαρμογών.

7.8.1 CDI – Context and Dependency Injection

Η CDI τεχνολογία είναι απ τις βασικότερες τεχνολογίες της Java EE πλατφόρμας διότι είναι ο συνδετικός κρίκος όπου πολλές άλλες τεχνολογίες της πλατφόρμας στηρίζονται σε αυτή.Η CDI τεχνολογία έχει πολλές ευρύτερες χρήσεις επιτρέποντας στους προγραμματιστές να συνδυάσουν διάφορα και πολύπλοκα κομμάτια κώδικα με έναν απλό αλλά ασφαλή τρόπο.

7.8.1.1 Ορισμός CDI

- **Context**
  Η δυνατότητα διαχείρισης του κύκλου ζωής και των αλληλεπιδράσεων μεταξύ διαφορετικών τμημάτων της εφαρμογής.
- **Dependency injection**
  Η δυνατότητα να επεκτείνουμε τις δυνατότητες ενός τμήματος μια εφαρμογής με ασφαλή τρόπο διαλέγοντας την υλοποίηση που θα χρησιμοποιηθεί για την επέκταση.

7.8.1.2 Ορισμός Java Beans

Οι προδιαγραφές για τα Managed Beans ορίζουν τα beans ως αντικείμενα τα οποία διαχειρίζονται από τους containers.Είναι γνωστά και με το ακρωνύμιο POJOS (Plain Old Java Objects).Τα beans υποστηρίζουν ένα μικρό σύνολο από βασικών λειτουργιών όπως το resource injection, lifecycle callbacks και interceptors.Σχεδόν κάθε κλάση στην Java που έχει έναν δομητή χωρίς παραμέτρους είναι Java Bean.Οι παραπάνω λειτουργίες χρησιμοποιούνται κατά κόρον σε όλες οι υπόλοιπες προδιαγραφές της Java EE όπως τα EJBs και το CDI.

7.8.2 EJB3 – Enterprise JavaBeans

Τα enterprise java beans (EJBs) είναι java ee components τα οποία απλοποιούν την ανάπτυξη μεγάλων και κατανεμημένων εφαρμογών διαδικτύου.Ο EJB container είναι υπεύθυνος για να προσφέρει υπηρεσίες συστήματος στους προγραμματιστές.Οι προγραμματιστές επικεντρώνονται στο να λύσουν τα επιχειρηματικά προβλήματα με τη χρήση των EJBs.Στα EJBs ενθυλακώνουμε την επιχειρηματική λογική της εφαρμογής μας θέτοντας της ρυθμίσεις όπως η ασφάλεια και οι συναλλαγές.Υπάρχουν δύο τύποι από EJBs αυτά είναι τα session beans και τα entity beans.Τα entity beans σε αντίθεση με τα session beans κρατούν την κατάσταση τους και την συμπεριφορά τους.

7.8.2.1 Session Beans

Τα session beans ενθυλακώνουν σε μεθόδους επιχειρηματική λογική στην οποία μπορούμε να έχουμε πρόσβαση μέσω εφαρμογών πελάτη.Οι προδιαγραφές για τα EJB ορίζουν τρία είδη session beans.

7.8.2.1.1 Stateful Session Beans

Η επιχειρηματική λογική, που έχει ενθυλακωθεί σε stateful beans, έχουν την ιδιομορφία ότι υπάρχει αντιστοίχηση ένα προς ένα με το πρόγραμμα πελάτη που επικοινωνεί.Αυτό σημαίνει ότι το εκάστοτε stateful bean δεν μοιράζεται με άλλους πελάτες και εξυπηρετεί μόνο τον συγκεκριμένο πελάτη.Σε όλη την διάρκεια ζωής του stateful bean, εκτός απ όταν καταστραφεί, η κατάσταση του διατηρείται.Όταν ο πελάτης, που χρησιμοποιεί το εκάστοτε bean, τερματίσει την επικοινωνία, τότε τελειώνει ο κύκλος ζωής του stateful bean και καταστρέφεται.

7.8.2.1.2 Stateless Session Beans

Στην περίπτωση όπου η επιχειρηματική λογική ενθυλακώνεται σε stateless beans, ο application server δημιουργεί μια ομάδα από διαφορετικά stateless beans, τα οποία είναι έτοιμα να εξυπηρετήσουν τα εισερχόμενα αιτήματα.Κάθε φορά που εφαρμογές πελάτη καλούν μεθόδους των stateless beans, τότε διαλέγεται τυχαία ένα stateless bean, που θα εξυπηρετήσει αυτό το αίτημα.Σε αντίθεση με τα stateful beans τα stateless beans δεν διατηρούν την κατάσταση τους.Όταν εφαρμογές πελάτη καλούν μεθόδους σε stateless beans, οι μεταβλητές του bean μπορεί να περιέχουν τιμές, οι οποίες αφορούν την εκάστοτε εφαρμογή πελάτη και διατηρούν αυτές τις τιμές μόνο για την διάρκεια αυτής της κλήσης.Επειδή δεν υπάρχει σχέση ένα προς ένα όπως στα stateful beans, τα stateless beans προσφέρουν καλύτερη κλιμάκωση για εφαρμογές όπου πολλοί πελάτες πρέπει να εξυπηρετηθούν ταυτόχρονα.Τυπικά μια εφαρμογή απαιτεί λιγότερα stateless beans από ότι stateful beans για την εξυπηρέτηση τον ίδιων αριθμό πελατών.

7.8.2.1.3 Singleton Session Beans

Σε εφαρμογές όπου θέλουμε να ενθυλακώσουμε επιχειρηματική λογική μόνο σε ένα σημείο χρησιμοποιούμε singleton beans.Τα singleton beans διατηρούνται κατά όλη την διάρκεια ζωής της εφαρμογής,δηλαδή έχουν τον ίδιο κύκλο ζωής της εφαρμογής και μοιράζονται ανάμεσα σε πολλούς πελάτες ταυτόχρονα.Μοιάζουν πολύ με τα stateless beans αλλά διαφέρουν στο ότι υπάρχει μόνο ένα singleton bean ανά εφαρμογή.Αν και εξυπηρετούν πολλούς πελάτες, η κατάσταση τους διατηρείται.Δεν είναι απαραίτητο να διατηρούν την κατάσταση τους εάν ο application server τερματίσει απροειδοποίητα.Στα singleton beans μπορούμε να ορίσουμε ενέργειες αρχικοποίησης και καθαρισμού όταν η εφαρμογή ξεκινά και τερματίζει αντίστοιχα.

7.8.2.2 Entity JavaBeans

Ένα entity bean αναπαριστά επιχειρηματικά δεδομένα τα οποία μπορούν να αποθηκευθούν μόνιμα.Χρησιμοποιούνται στην JPA για αντιστοίχηση αντικειμένων σε δεδομένα που αποθηκεύονται στη βάση δεδομένων.Οι διαφορές με τα sessions beans είναι οι εξής.

- **Persistence**
  Η κατάσταση ενός entity bean είναι αποθηκεύσιμη.
- **Shared access**
  Έχουν πρόσβαση πολλοί πελάτες.
- **Primary key**
  Κάθε entity bean αναγνωρίζεται μοναδικά.
- **Relationships**
  Υπάρχουν σχέσεις μεταξύ των entity beans.

7.8.3 Συναλλαγές

Οι συναλλαγές παίζουν καθοριστικό ρόλο στην ανάπτυξη μιας Java EE εφαρμογής η οποία χρησιμοποιεί βάση δεδομένων και την τεχνολογία των EJBs.Στην Java EE ορίζουμε δύο τύπους συναλλαγών τις "Container Managed" συναλλαγές και τις "Bean Managed" συναλλαγές.Πριν εξηγήσουμε τους τύπους συναλλαγών της Java EE, πρώτα θα εξηγήσουμε τα χαρακτηριστικά των συναλλαγών.

7.8.3.1 Χαρακτηριστικά Συναλλαγών

- **Ατομικότητα**
  Η ατομικότητα ορίζει ότι εάν σε μια συναλλαγή υπάρξει κάποιο λάθος και δεν εκτελεστεί σωστά η συναλλαγή, τα βήματα της συναλλαγής που έχουν γίνει μέχρι εκείνη την στιγμή θα ακυρωθούν όλα.
- **Συνέπεια**
  Η κατάσταση των εγγραφών στην βάση δεδομένων αλλάζουν από την μία συνεπή κατάσταση στην άλλη.
- **Απομόνωση**
  Η απομόνωση ορίζει την ορατότητά στα δεδομένα που θα έχουν παράλληλες και μη συναλλαγές και οι αλλαγές τις μίας συναλλαγής δεν επηρεάζει την ορατότητα των υπόλοιπον συναλλαγών.
- **Μονιμότητα**
  Η μονιμότητα εγγυάται στον προγραμματιστή ότι όταν τελειώσει μια συναλλαγή επιτυχώς τότε τα αποτελέσματα της δεν θα χαθούν.

7.8.3.2 Container-Managed Transactions

Στην περίπτωση που χρησιμοποιούμε container managed transactions (CMT) υπεύθυνος για τις συναλλαγές είναι ο EJB container.Κατά την ανάπτυξη εφαρμογής με CMT συναλλαγές ο κύκλος ζωής των συναλλαγών διαχειρίζονται αυτόματα.Στον πηγαίο κώδικα δεν ορίζουμε την αρχή και το τέλος μιας συναλλαγής αλλά αυτό γίνεται αυτόματα.Ο container αρχίζει την συναλλαγή κάθε φορά που ξεκινά μια μέθοδος και ολοκληρώνει την συναλλαγή λίγο πριν κάνει έξοδο απ την μέθοδο.Η κάθε μέθοδος μπορεί να συσχετιστεί με μόνο μια συναλλαγή.Η διαχείριση των συναλλαγών είναι αυτόματη αλλά μπορούμε να ρυθμίσουμε με έξι τρόπους που παρουσιάζουμε παρακάτω.Με τη χρήση του annotation @TransactionAttribute(TransactionAttributeType.TYPE) ορίζουμε στις παρενθέσεις τον τύπο της συναλλαγής.

- **MANDATORY**
  Μια μέθοδος χαρακτηρισμένη ως mandatory εγγυάται ότι θα εκτελεστεί η μέθοδος σε μια συναλλαγή.Εάν καλεστεί εκτός συναλλαγή θα παρουσιάσει σφάλμα κατά την κλήση της.
- **REQUIRED**
  Μια required μέθοδος είναι σας μια μέθοδο χαρακτηρισμένη ως mandatory, μόνο που σε αυτή την περίπτωση δε θα παρουσιαστεί σφάλμα αλλά ο container θα ξεκινήσει μια νέα συναλλαγή εάν η μέθοδος δεν καλεστεί από συναλλαγή.
- **REQUIRES_NEW**
  Μια requires_new μέθοδος είναι σαν σαν μια μέθοδο χαρακτηρισμένη ως required αλλά οι συναλλαγές που εκτελούνται παράλληλα εκείνη την χρονική περίοδο θα σταματήσουν ώστε να δημιουργηθεί και να ολοκληρωθεί αυτή η νέα συναλλαγή.
- **NEVER**
  Μια μέθοδος χαρακτηρισμένη ως never εγγυάται ότι δεν θα εκτελεστεί ποτέ σε μια συναλλαγή.Ώσοστο αν καλεστεί μέσα από συναλλαγή θα παρουσιάσει σφάλμα.
- **NOT_SUPPORTED**
  Μια μέθοδος χαρακτηρισμένη ως not_supported εγγυάται ότι δεν θα εκτελεστεί σε κάποια συναλλαγή.Εαν καλεστεί μέσα από μια συναλλαγή τότε ο container θα σταματήσει την συναλλαγή θα εκτελέσει την not_supported μέθοδο και θα επιστρέψει την συναλλαγή που σταμάτησε προηγουμένως στην κανονική της ροή.
- **SUPPORTS**
  Μια μέθοδος χαρακτηρισμένη ως supports θα χρησιμοποιήσει το είδος τις συναλλαγής από τον αιτούντα της.Οι supports μεθόδοι εκτελούνται είτε σε συναλλαγές είτε όχι αναλόγως τον τύπο συναλλαγής του αιτούντα.

Στις συναλλαγές που διαχειρίζεται ο container υπάρχουν δύο τρόποι στο να γίνει επαναφορά στις αλλαγές που έχουν γίνει μέσα σε συναλλαγές.Ο ένας τρόπος είναι εάν το σύστημα παρουσιάσει σφάλμα και ο δεύτερος εάν καλέσουμε την μέθοδο “setRollbackOnly” που ενημερώνουμε τον container να κάνει επαναφορά την συναλλαγή.

7.8.3.3 Bean-Managed Transactions (BMT)

Στην περίπτωση που χρησιμοποιούμε bean managed transactions (BMT) η διαχείριση των συναλλαγών γίνεται από τον προγραμματιστή με την συγγραφή κώδικα.Ο προγραμματιστής κάνει χρήστη των JTA Transactions για την διαχείριση των συναλλαγών μέσω του transaction manager που προσφέρεται από τον application server.Οι πιο βασικές μέθοδοι για την διαχείριση των JTA συναλλαγών είναι οι παρακάτω.Η μέθοδος “UserTransaction.begin()”, ξεκινά μια συναλλαγή και την συσχετίζει στο νήμα στο οποίο εκτελείται.Ο προγραμματιστής δεν μπορεί να δημιουργήσει εμφωλευμένες συναλλαγές με τη χρήση της παραπάνω μεθόδου.Η μέθοδος “UserTransaction.commit()” ολοκληρώνει μια συναλλαγή εφαρμόζοντας τις αλλαγές που έχουν γίνει κατά την διάρκεια της συναλλαγής.Κατά την διάρκεια που καλεστεί η παραπάνω μέθοδος, εάν η συναλλαγή χρειάζεται να κάνει επαναφορά τότε ο transaction manager παρουσιάζει σφάλμα.Με την μέθοδο “UserTransaction.rollback()” οι αλλαγές που έχουν γίνει σε μία συναλλαγή επαναφέρονται στην αρχική τους κατάσταση και δεν αποθηκεύονται μόνιμα οι αλλαγές που έχουν γίνει κατά την διάρκεια της συναλλαγής.Αφού εκτελεστεί η παραπάνω μέθοδος τότε σταματά πλέον να υφίσταται η συσχέτιση μεταξύ της συναλλαγής και στο νήμα στο οποίο εκτελείται.

7.8.3.4 Επίπεδα Απομόνωσης Συναλλαγών

Είτε χρησιμοποιούμε CMT είτε BMT ο προγραμματιστής πρέπει να γνωρίζει τον τύπο της απομόνωσης κάθε φορά.Ο τύπος απομόνωσης ορίζει το πως μια συναλλαγή είναι απομονωμένη από μια άλλη.Παρακάτω αναφέρουμε τρεις διαφορετικούς τύπους απομόνωσης.

- **Dirty reads**
  Συναλλαγές διαβάζουν δεδομένα τα οποία έχουν γραφτεί από άλλες συναλλαγές πριν αυτά τα δεδομένα αποθηκευθούν μόνιμα.
- **Non-repeatable reads**
  Συναλλαγές διαβάζουν δεδομένα τα οποία έχουν αλλάξει από άλλη συναλλαγή και είναι διαφορετικά από την πρώτη φορά που η πρώτη συναλλαγή διάβασε τα δεδομένα.
- **Phantom reads**
  Συναλλαγές εκτελούν ένα ερώτημα το οποίο επιστρέφει ένα σύνολο από αποτελέσματα, ταυτόχρονα μια άλλη συναλλαγή κάνει αλλαγές και η πρώτη συναλλαγή επαναλαμβάνει το ίδιο ερώτημα παραλαμβάνοντας διαφορετικά αποτελέσματα.

8. Υλοποίηση

8.1 Εισαγωγή

Στα πλαίσια κατανόησης των τεχνολογιών που αναφέρθηκαν στα προηγούμενα κεφάλαια αλλά και στο τρόπος με τον οποίο μπορούν να συνδυαστούν δημιουργήθηκε μία εφαρμογή διαδικτύου.Η εφαρμογή διαδικτύου που υλοποιούμε είναι ένα ηλεκτρονικό κατάστημα που εμπορεύεται κινητά τηλέφωνα.

8.2 Αρχιτεκτονική Εφαρμογής

Ο ηλεκτρονικός υπολογιστής που υλοποιήθηκε η εφαρμογή έχει τοπική διεύθυνση διαδικτύου την 192.168.10.9.Επίσης τα εργαλεία που εγκαταστάθηκαν τα παρακάτω

- Λειτουργικό σύστημα Windows 10 x64
- Docker Community Edition 17.12.0-ce-win47 (15139)
- MySQL 5.7

Οι απαιτήσεις του καταστήματος είναι οι αυτές που αναφέρονται στον παρακάτω πίνακα.Για κάθε μια από αυτές τις απαιτήσεις, με τη χρήση της Java EE, θα δημιουργηθούν τρία microservices με τα εξής ονόματα.Να σημειωθεί ότι το όνομα κάθε υπηρεσίας έχει καθοριστικό ρόλο για την σωστή αναζήτηση των υπηρισεών.

Απαιτήσεις	Όνομα Service
Διαχείριση πελατολογίου	users
Αποθήκη κινητών	orders
Παραγγελία κινητού	phones

Όλες οι παραπάνω λειτουργίες ενθυλακώνονται σε δύο docker containers με ονόματα “sso” και “phone”.Λόγο της αρχιτεκτονικής των microservices είναι αναγκαίο οι δύο containers να επικοινωνούν μεταξύ τους μέσω δικτύου.Για να επιτευχθεί το παραπάνω θα δημιουργήσουμε ένα δίκτυο με το όνομα “phone” και κατά την δημιουργία των containers θα τους αναθέσουμε τους containers στο παραπάνω δίκτυο.Η δημιουργία του “phone” δικτύου δημιουργείται με την εντολή “**docker network create phone”.**Πλέον το “phone” δίκτυο δημιουργήθηκε και είμαστε σε θέση να δημιουργήσουμε τα docker images απ τα οποία θα δημιουργηθούν τα containers.Οι δύο εντολές για την δημιουργία των δύο docker images είναι οι παρακάτω.

- **docker build -t phone/sso -f ./docker/keycloak/Dockerfile .**
- **docker build -t phone/server -f ./docker/wildfly/Dockerfile .**

Όπως βλέπουμε για την δημιουργία των images χρησιμοποιούμε την build εντολή από την εφαρμογή πελάτη docker.Ως παραμέτρους ορίζουμε την διαδρομή (-f) στην οποία θα αναζητήσει το Dockerfile το οποίο περιέχει τα βήματα δημιουργίας των images.Υποχρεωτική παράμετρος είναι το όνομα των images την οποία ορίζουμε με την (-t) παράμετρο.Εφόσον έχουν εκτελεστεί οι δύο παραπάνω εντολές δημιουργούνται δύο docker images με τα ονόματα "**phone/sso"** και "**phone/server"**.

Το πρώτο image είναι το "**phone/sso"** και φιλοξενεί το Keycloak.Το Keycloak είναι μία πλατφόρμα για αυθεντικοποίηση χρηστών και microservices.Η έκδοση του Keycloak που κάνουμε χρήση είναι η 3.3.0.CR2.Το δεύτερο image είναι το "**phone/server"**, φιλοξενεί το Wildfly που είναι ένας Java EE Application Server, έναν nginx διακομιστή που ο ρόλος του είναι ως reverse proxy στα microservices που φιλοξενούνται στον Wildfly, την Keycloak υπηρεσία αλλά και την διαδικτυακή πύλη απ την οποία πελάτες του καταστήματος θα κάνουν αγορές η οποία κατασκευάστηκε με την χρήση της Angular πλατφόρμας έκδοση 4.4.3.Για την φιλοξενία των microservices δεν χρησιμοποιήσαμε τρεις διαφορετικούς application servers αλλά μόνο έναν.Κάθε microservice θα αναγνωρίζεται μοναδικά από το όνομα του.Θα μπορούσαμε να έχουμε τρεις διαφορετικούς application servers όπου ο καθένας θα φιλοξενούσε μόνο ένα microservice και το κάθε microservice θα αναγνωριζόταν μοναδικά από το όνομα του.Ώστοσο για λόγους ευκολίας τοποθετήσαμε στον ίδιο application server τα microservices.

Πλέον τα images δημιουργήθηκαν και το επόμενο βήμα είναι να δημιουργήσουμε τα κατάλληλα containers χρησιμοποιώντας τα images που μόλις δημιουργήθηκαν.Οι εντολές για την δημιουργία των δύο containers είναι οι παρακάτω.

- **docker run -d --add-host database:192.168.10.9 --net phone --name sso phone/sso**
- **docker run -d -p 80:80 --add-host database:192.168.10.9 --net phone --name phone phone/server**

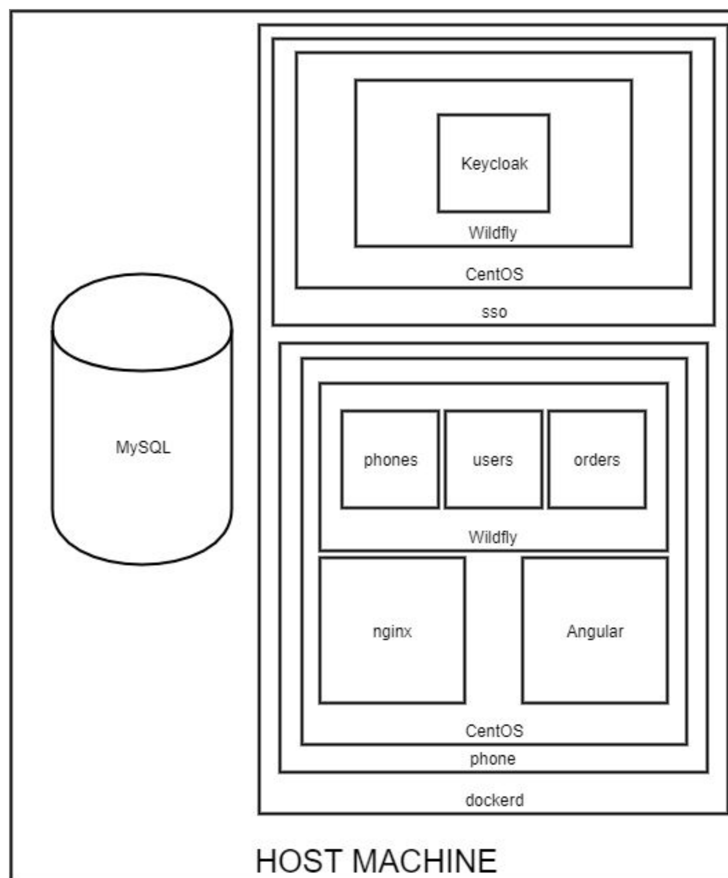
Στον πηγαίο κώδικα των υπηρεσιών έχουμε ορίσει ένα όνομα τομέα "database" που χρησιμοποιείται για την σύνδεση με την βάση δεδομένων.Δυστυχώς αυτό το όνομα τομέα δεν υπάρχει και κατά την διάρκεια αναζήτησης αυτό του ονόματος τομέα θα παρουσιαστεί σφάλμα.Για την αποφυγή του σφάλματος αλλά και για να μην ορίζουμε στον πηγαίο κώδικα την διεύθυνση δικτύου, κάνουμε αντιστοίχηση δυναμικά το όνομα τομέα "database" σε μια διεύθυνση δικτύου.Με την παράμετρο "--add-host database:192.168.10.9" κάνουμε αντιστοίχηση το όνομα τομέα με μια διεύθυνση δικτύου.

Για να υπάρχει επικοινωνία μέσω δικτύου μεταξύ των containers πρέπει ανήκουν στο ίδιο δίκτυο.Με την παράμετρο “--net phone” ορίζουμε ότι τα δύο containers ανήκουν στο δίκτυο “phone” το οποίο είχαμε δημιουργήσει προηγουμένως.

Με την παράμετρο “--name” ορίζουμε τα ονόματα των container και με την αμέσως επόμενη παράμετρο ορίζουμε το image απ το οποίο θα δημιουργηθούν τα containers.

Η επικοινωνία με τα microservices αλλά και με την Keycloak υπηρεσία γίνεται μέσω του πρωτοκόλλου HTTP.Έχουμε ορίσει ότι για τέτοιου είδους επικοινωνία είναι ικανός μόνο ο “phone” container.Με την παράμετρο “-p 80:80” κάνουμε αντιστοίχηση την πόρτα 80 του phone container με την πόρτα 80 τους μηχανήματος στο οποίο φιλοξενείται.

Θα χρησιμοποιηθεί μόνο μια βάση δεδομένων γιατί όποιος προκύπτει και στην επόμενη παράγραφο οι πίνακες (της βάσης) των microservces δεν έχουν σχέση μεταξύ τους οπότε δεν υπάρχει ανάγκη για διαφορετική βάση δεδομένων.Όλες οι πληροφορίες για τα κινητά τηλέφωνα είναι διαθέσιμες στην διεύθυνση https://www.kaggle.com/arwinneil/gsmarena-phone-dataset/data σε CSV μορφή.Για εισαγωγή όλων των δεδομένων δημιουργήθηκε ένας μετατροπέας (PhoneDataConvert) ο οποίος μετατρέπει τα δεδομένα από CSV σε SQL ερωτηματα.



Σχήμα 8.2 Αρχιτεκτονική Εφαρμογής

8.3 Αρχιτεκτονική βάσης

Όπως είπαμε και σε προηγούμενη ενότητα για λόγους ευκολίας κάναμε χρήση μόνο μίας βάσης δεδομένων.Όπως βλέπουμε και στο σχήμα 8.2 δεν υπάρχουν σχέσεις μεταξύ των πινάκων οπότε δεν υφίσταται θέμα για την ύπαρξη μιας βάσης δεδομένων.Ο κώδικας δημιουργίας των παρακάτω πινάκων βρίσκεται στο αρχείο "V1__Table_Initialization.sql".Η βάση δεδομένων αποτελείται από τρεις πίνακες τον phone_users, phone_phones και phone_orders.

Σχήμα 8.2 Αρχιτεκτονική Βάσης Δεδομένων

phone_users	Ο πίνακας αναπαριστά έναν χρήστη του συστήματος
Πεδία	Επεξήγηση
ID	Πρωτεύον κλειδί που αναγνωρίζει μοναδικά έναν χρήστη
USER_FIRSTNAME	Το όνομα του χρήστη
USER_LASTNAME	Το επώνυμο του χρήστη
EMAIL	Το email του χρήστη
TELEPHONE	Το κινητό ή σταθερό του χρήστη
ADDRESS	Η διεύθυνση αποστολής των παραγγελιών
KEYCLOAK_ID	Ένα κλειδί που λειτουργεί ως ξένο για την αντιστοίχησης του χρήστη του microservice με την υπηρεσία αυθεντικοποίησης

Επεξήγηση πεδίων του πίνακα phone_users

phone_phones	Ο πίνακας αναπαριστά την αποθήκη του καταστήματος
Πεδία	Επεξήγηση
ID	Πρωτεύον κλειδί που αναγνωρίζει μοναδικά ένα κινητό
PHONE_NAME	Το όνομα του κινητού
PHONE_BRAND	Το μοντέλο του κινητού
PHONE_CPU	Ο επεξεργαστής του κινητού
PHONE_SCREEN_SIZE	Το μέγεθος οθόνης του κινητού
PHONE_RELEASE_DATE	Η ημερομηνία έκδοσης του κινητού
PHONE_PRICE	Το κόστος του κινητού
PHONE_IMAGE	Ένας σύνδεσμος που έχει το κινητό ως εικόνα

Επεξήγηση πεδίων του πίνακα phone_phones

phone_orders	Ο πίνακας αναπαριστά τις παραγγελίες του καταστήματος
Πεδία	Επεξήγηση
ID	Πρωτεύον κλειδί που αναγνωρίζει μοναδικά μια παραγγελία
USER_ID	Ένα κλειδί που λειτουργεί ως ξένο για την αντιστοίχησης της παραγγελίας με χρήστη του phone_users
ITEMS	Κλειδιά χωρισμένα με κόμμα που λειτουργούν ως ξένα για αντιστοίχιση με τα κινητά του phone_phones
INPROGRESS	Boolean μεταβλητή που υποδεικνύει την ότι η παραγγελία βρίσκεται σε εξέλιξη
COMPLETED	Boolean μεταβλητή που υποδεικνύει την ότι η παραγγελία έχει ολοκληρωθεί
CANCELLED	Boolean μεταβλητή που υποδεικνύει την ότι η παραγγελία βρίσκεται έχει ακυρωθεί
TOTAL_PRICE	Το τελικό κόστος της παραγγελίας

Επεξήγηση πεδίων του πίνακα phone_orders

Όπως είδαμε και παραπάνω τα πεδία USER_ID και ITEMS του πίνακα phone_orders και το πεδίο KEYCLOAK_ID του πίνακα phone_users λειτουργούν ως ξένα κλειδιά αλλά δεν είναι.Τα microservices θα χρησιμοποιούν αυτά τα κλειδιά για την συσχέτιση πληροφοριών.

8.4 Αυθεντικοποίηση και διαχείριση ταυτότητας

Το πρώτο βήμα αφού εκτελεστούν οι docker containers είναι να εισάγουμε τις ρυθμίσεις στο Keycloak απ το αρχείο realm-export.json.Αυτές οι ρυθμίσεις περιλαμβάνουν την δημιουργία ενός καινούργιου realm που ονομάστηκε “sso”.Ως realm ορίζουμε μια πολική ασφαλείας για τις εφαρμογές μας.Η πολική περιλαμβάνει χρήστες, ρόλους και ομάδες.Για την σωστή λειτουργία της εφαρμογής, στο sso realm δημιουργήσαμε έναν πελάτη με το όνομα sso.Ως πελάτες το keycloak ορίζει έμπιστες εφαρμογές διαδικτύου ή web services.O sso πελάτης έχει από κατασκευής του τον “uma_protection” ρόλο.Τον “uma_protection” ρόλο έχει οποιοδήποτε εγγεγραμμένος χρήστης στο keycloak.Ο ρόλος “admin” προστέθηκε και αντιπροσωπεύει τον διαχειριστή του καταστήματος.

8.5 Web Server και Reverse Proxy

Ο nginx server λειτουργεί ως reverse proxy για τα microservices αλλά και ως web server για το ηλεκτρονικό κατάστημα.Εφαρμογές πελάτη μπορούν να επικοινωνήσουν με τα microservices και τον keycloak server μόνο μέσω του nginx.

```
http {
   upstream keycloak_server {
       server  sso:8080;
   }
   upstream wildfly_server {
       server  localhost:8080;
   }
server {
   listen 80;
   location ~ /(orders|users|phones)/resources(.*) {
       proxy_pass http://wildfly_server;
       proxy_http_version 1.1;
       proxy_set_header Host                $host;
       proxy_set_header X-Real-IP           $remote_addr;
       proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
       proxy_set_header X-Forwarded-Proto  $scheme;
   }
```

```
    location /auth/ {
        proxy_pass http://keycloak_server;
        proxy_http_version 1.1;
        proxy_set_header Host               $host;
        proxy_set_header X-Real-IP          $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto  $scheme;
    }
    location / {
        root /phone;
        index index.html;
        include /etc/nginx/mime.types;
        try_files $uri $uri/ /index.html;
     }
    }
}
```

Ρυθμίσεις αρχείου nginx.conf

Ο nginx εκτελείται στον docker container phone στον οποίο φιλοξενούνται η angular εφαρμογή και ο application server που φιλοξενει τα microservices.Στο αρχείο nginx.conf όπως παρουσιάζεται και παραπάνω βλέπουμε τις ρυθμίσεις του nginx server.Αρχικά με την λέξη κλειδί upstream ορίσαμε δύο ομάδες από servers.Η κάθε ομάδα περιέχει μόνο ένα server.Η πρώτη ομάδα είναι ο “keycloak_server” και η δεύτερη η “wildfly_server”.Η πρώτη ομάδα έχει έναν server με όνομα τομέα “sso:8080” και η δεύτερη έχει έναν server με όνομα τομέα “localhost:8080”.Στην πόρτα 8080 αναμένει ερωτήματα ο application server.Το όνομα τομέα “sso” δεν είναι υπαρκτό αλλά το docker θα αντιστοιχήσει το όνομα τομέα “sso” με την διεύθυνση δικτύου του container “sso”.Η αντιστοίχηση γίνετε με βάση το όνομα τομέα και τον container που έχει το ίδιο όνομα “sso”.Ο server στην δεύτερη ομάδα έχει ως διεύθυνση δικτύου το “localhost” το οποίο δηλώνει ότι ο server είναι στον ίδιο docker container(phone).Για να έχουμε πρόσβαση στα microservices, στην Keycloak υπηρεσία αλλά και την δικτυακή πύλη (angular) ορίσαμε με τις παρακάτω διαδρομές με την λέξη κλειδί “location”.Στην πρώτη διαδρομή ορίσαμε με τη χρήση regural expression ποιες διαδρομές θα εξυπηρετούνται από τον wildfly_server που βρίσκεται στον ίδιο docker container με τον nginx server.Στην δεύτερη διαδρομή ορίσαμε ότι οι διαδρομές που ξεκινούν με “/auth/” θα εξυπηρετούνται μέσω του “keycloak_server”.Στην πρώτη και στην δεύτερη περίπτωση ο nginx λειτουργεί απλά ως reverse proxy.Τέλος, τα αιτήματα που απευθύνονται στην ρίζα “/” αφορούν την angular εφαρμογή και εξυπηρετούνται από τον ίδιο nginx server.

8.6 Υλοποίηση Microservices

Για την υλοποίηση των microservices θα χρησιμοποιήσουμε την Java EE.Όλα τα microservices τα οποία έχουμε σχεδιάσει απαιτούν να έχουν πρόσβαση στην βάση δεδομένων.Οι ρυθμίσεις για την πρόσβαση στη βάση βρίσκονται στο αρχείο persistence.xml τις οποίες θα αναφέρουμε στην 8.6.1 ενότητα.Το persistence.xml είναι ίδιο σε όλα τα microservices.Τέλος όλα τα micrservices είναι προσβάσιμα με τη χρήση web services όποτε οι ρυθμίσεις των web services θα τις αναφέρουμε στην 8.6.2 ενότητα.

8.6.1 Ρυθμίσεις βάσης δεδομένων

Για τις ρυθμίσεις πρόσβασης στη βάση δεδομένων από την Java EE κάνουμε χρήστη του αρχείου persistence.xml.Παρακάτω βλέπουμε έναν πίνακα με τα στοιχεία του persistence.xml και την επεξήγηση τους.Το στοιχείο “<peristence-unit>” εμπεριέχει τα στοιχεία σύνδεσης σε μια βάση δεδομένων.Εαν η εφαρμογή μας απαιτεί την σύνδεση σε παραπάνω από μια βάσεις τότε το μόνο που έχουμε να κάνουμε είναι να προσθέσουμε ακόμα ένα στοιχείο “<peristence-unit>”.Στην περίπτωση που υπάρχουν παραπάνω από ένα “<peristence-unit>” στοιχεία τότε πρέπει κάθε να ορίσουμε στον κώδικα με τη χρήση annotation πιο “<peristence-unit>” θέλουμε να χρησιμοποιήσουμε.

Πεδία	Επεξήγηση
provider	Δηλώνουμε ότι θα χρησιμοποιηθεί το Hibernate
jta-data-source	Ορίζουμε πιο database pool θα χρησιμοποιήσουμε
javax.persistence.jdbc.user	Το όνομα χρήστη της βάσης δεδομένων
javax.persistence.jdbc.password	Ο κωδικός της βάσης δεδομένων
javax.persistence.jdbc.url	Το url σύνδεσης στη βάση δεδομένων.Στο “database” domain θα δώσουμε τιμή κατά την δημιουργία του Docker container
javax.persistence.jdbc.driver	Ορίζουμε τον JDBC οδηγό για την επικοινωνία της Java με την MySQL βάση
hibernate.dialect	Δηλώνουμε στο Hibernate σε τι είδους βάση θα συνδεθούμε

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="KeycloakDS" transaction-type="JTA">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>java:jboss/datasources/KeycloakDS</jta-data-source>
        <properties>
            <property name="javax.persistence.jdbc.user" value="admin"/>
            <property name="javax.persistence.jdbc.password"
value="admin"/>
            <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://database:3306/phone"/>
            <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver"/>
           <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL5Dialect"/>
        </properties>
    </persistence-unit>
</persistence>
```

Ρυθμίσεις αρχείου persistence.xml

8.6.2 Ρυθμίσεις Web Services

Όλα τα microservices που δημιουργήσαμε έχουν την κλάση JAXRSConfiguration.Για την ενεργοποίηση των web services επεκτείνουμε την κλάση JAXRSConfiguration με την Application κλάση από το πακέτο “javax.ws.rs.core”.Με το annotation @ApplicationPath ορίζουμε την διαδρομή στην οποία θα είναι διαθέσιμα τα web services.Με αυτή την ρύθμιση όλα τα microservices είναι διαθέσιμα στη παρακάτω διαδρομή διαδικτύου “http://192.168.10.9/resources/”.

```java
import javax.ws.rs.core.Application;
@ApplicationPath("resources")
public class JAXRSConfiguration extends Application {}
```

Ρυθμίσεις αρχείου JAXRSConfiguration.java

8.6.3 Ρυθμίσεις ασφαλείας

Οι ρυθμίσεις ασφαλείας των microservices βρίσκεται στο αρχέιο web.xml.Αρχικά σε αυτά τα αρχεία ορίζουμε τον τύπο ασφάλειας που θα χρησιμοποιήσουμε και έπειτα τι θα ασφαλίσουμε, δηλαδή ποιες μέθοδοι θα έχουν πρόσβαση και από ποίους.Όπως

φαίνεται και παρακάτω, ο τύπος ασφαλείας που ρυθμίζουμε τα microservices να χρησιμοποιούν είναι η keycloak υπηρεσία με πολιτική ασφαλείας το sso realm.

```
<login-config>
        <auth-method>KEYCLOAK</auth-method>
        <realm-name>sso</realm-name>
</login-config>
```

Ρυθμίσεις αρχείου web.xml

Άλλη μια προϋπόθεση για να επικοινωνούν με ασφάλεια τα microservices είναι να προσθέσουμε το αρχείο keycloak.json που μας παρείχε ο keycloak διαδομιστής στο κάθε microservice.

Επίσης στο web.xml θέτουμε τους ρόλους που μπορεί να έχουν οι χρήστες της εφαρμογής.Οι δύο ρόλοι που έχουμε ορίσει είναι ένας απλός αυθεντικοποιημένος χρήστης(uma_protection) και ο δεύτερος είναι ένας χρήστης με δυνατότητες διαχειριστή(admin).Όλα τα microservices έχουν και τους δύο ρόλους έκτος από το phones microservice που έχει μόνο τον ρόλο uma_protection.Παρακάτω βλέπουμε πως ορίζουμε στο web.xml τους ρόλους που αναφέραμε.

```
<security-role>
        <role-name>uma_protection</role-name>
</security-role>
<security-role>
        <role-name>admin</role-name>
</security-role>
```

Ρυθμίσεις αρχείου web.xml

Πιο συγκεκριμένα στο phones microservice ορίζουμε ότι πρόσβαση θα έχουν μόνο αυθεντικοποιημένοι χρήστες.Παρακάτω βλέπουμε τις συγκεκριμένες ρυθμίσεις

```
<security-constraint>
        <web-resource-collection>
            <web-resource-name>All Resources</web-resource-name>
            <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>uma_protection</role-name>
        </auth-constraint>
</security-constraint>
```

Ρυθμίσεις αρχείου web.xml

Στο orders microservice ορίζουμε ότι, είτε απλοί χρήστες είτε διαχειριστές θα έχουν πρόσβαση στα web services.Το μόνο που έχουμε να αλλάξουμε στις παραπάνω ρυθμίσεις είναι να προσθέσουμε και τον ρόλο του διαχειριστή όπως φαίνεται και παρακάτω.

```
<auth-constraint>
            <role-name>uma_protection</role-name>
            <role-name>admin</role-name>
</auth-constraint>
```

Ρυθμίσεις αρχείου web.xml

Στην περίπτωση του users microservice εκτός από αυθεντικοποιημένους χρήστες θέλουμε να έχουν πρόσβαση στην υπηρεσία εγγραφής χρήστη, χρήστες που δεν είναι αυθεντικοποιημένοι.Για να ορίσουμε το παραπάνω ορίζουμε ποια διαδρομή δεν θέλουμε να έχει αυθεντικοποίηση παραλείποντας το στοιχείο “<auth-constraint>” στο στοιχείο <security-constraint>.

```
<security-constraint>
        <display-name>Register EndPoint</display-name>
        <web-resource-collection>
            <web-resource-name>Register Rest
EndPoint</web-resource-name>
            <url-pattern>/resources/users</url-pattern>
        </web-resource-collection>
        <!-- OMIT auth-constraint -->
</security-constraint>
```

Ρυθμίσεις αρχείου web.xml

8.6.4 Phones microservice

Όπως αναφέραμε και παραπάνω το phone microservice αναπαριστά την αποθήκη του καταστήματος.Όλα τα δεδομένα για τα κινητά τηλέφωνα έχουν ήδη εισαγωγή στην βάση δεδομένων και σκοπός αυτού του service είναι να παρέχει πληροφορίες σχετικά με τα κινητά που έχει η αποθήκη.

Παρακάτω βλέπουμε την κλάση Phone η οποία αναπαριστά ένα κινητό τηλέφωνο.Στο τέλος της κλάσης παραλείπετε κώδικας που ουσιαστικά είναι μέθοδοι όπως setters και getters.Με την βοήθεια των annotations ορίζουμε επιπλέον πληροφορίες σχετικά την κλάση Phone.Με το annotation @Entity ορίζουμε ότι η κλάση Phone είναι ένα bean που σχετίζεται με την βάση δεδομένων και η κατάσταση του είναι μόνιμα αποθηκεύσιμη.Το annotation @Table ορίζει ότι η JPA θα συσχετίζει την κλάση Phone με τον πίνακα που ορίζει ως παράμετρο δηλαδή το “PHONE_PHONES”, διαφορετικά θα προσπαθούσε να την συσχετίσει με τον πίνακα Phone δηλαδή το όνομα που έχει η κλάση και το οποίο θα δημιουργούσε πρόβλημα.

Η συσχέτιση της κλάσης με την βάση δεδομένων δεν αφορά μόνο το όνομα του πίνακα αλλά τα πεδία που έχει ο πίνακας.Τα πεδία της κλάσης Phone γίνονται αντιστοίχηση με τα πεδία του πίνακα της βάσης.Από κατασκευής η αντιστοίχηση γίνεται μέσω το όνομα των μεταβλητών.Για να αλλάξουμε τον τρόπο της αντιστοίχησης πρέπει να ορίσουμε τα πεδία της κλάσης με το annotation @Column και να ορίσουμε το όνομα του πεδίου με το οποίο θα γίνει η αντιστοίχηση στην βάση δεδομένων.Ιδιαίτερη περίπτωση είναι το πεδίο της κλάσης ID διότι εκτός από το @Column annotation έχει ακόμα δυο επιπλέον annotations.Το @Id ορίζει ότι το πεδίο ID δεν είναι απλός ένα πεδίο του πίνακα της βάσης αλλά είναι και το πρωτεύων κλειδί του πίνακα Phone.Επίσης απ τον κώδικα της βάσης βλέπουμε ότι το πεδίο ID είναι autoincrement αυτό το δηλώνουμε στην JPA μέσω του @GeneratedValue.Το @NamedQueries ορίζει δυο SQL ερωτήματα σε JPQL.Το πρώτο αφορά ερώτημα που επιστρέφει όλα τα κινητά που έχει η βάση δεδομένων.Το δεύτερο αφορά ερώτημα όπου κάθε φορά επιστρέφει μόνο ένα κινητό από την βάση.Έφοσον έχει γίνει η συσχέτιση της κλάσης Phone με τον πίνακα της βάσης PHONE_PHONES και έχουν δημιουργηθεί και τα αντίστοιχα ερωτήματα το μόνο που μένει είναι να καλεστούν μέσω τον web services τα αντίστοιχα JPQL ερωτήματα και να επιστραφεί απάντηση η οποία έχει την μορφή JSON.Για την μετατροπή κάποιας κλάσης σε άλλη μορφή χρησιμοποιήσουμε το @XmlRootElement annotation.Με το @XmlAccesoorType ορίζουμε ότι η μετατροπή δεν θα γίνει χρησιμοποιώντας getters αλλά θα χρησιμοποιηθεί reflection ώστε να γίνει η μετατροπή.

```
@Entity
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
@Table(name = "PHONE_PHONES")
@NamedQueries({
    @NamedQuery(query = "SELECT p FROM Phone p",name = "Phone.findAll"),
    @NamedQuery(query = "SELECT p FROM Phone p where p.ID=:id", name =
"Phone.findById")
})
public class Phone {

    @Id
    @Column(name = "ID")
    @GeneratedValue
    private long ID;
    @Column(name = "PHONE_NAME")
    private String Name;
    @Column(name = "PHONE_BRAND")
    private String Brand;
```

```
    @Column(name = "PHONE_CPU")
    private String Cpu;
    @Column(name = "PHONE_SCREEN_SIZE")
    private String ScreenSize;
    @Column(name = "PHONE_RELEASE_DATE")
    private String Date;
    @Column(name = "PHONE_PRICE")
    private String Price;
    @Column(name = "PHONE_IMAGE")
    private String Image;
   //<output omitted>
}
```

Κώδικας αρχείου Phone.java

Όπως αναφέραμε σκοπός του service είναι να δίνουμε πληροφορίες με τα κινητά στα υπόλοιπα microservices για αυτό το σκοπό δημιουργήσαμε ένα web service.

```
@Path("phones")
@Stateless
@Produces(MediaType.APPLICATION_JSON + ";charset=UTF-8")
public class PhoneResource {

    @PersistenceContext
    EntityManager em;

    @GET
    @RolesAllowed("uma_protection")
    public List<Phone> getAllPhones() {
        return em.createNamedQuery("Phone.findAll",
Phone.class).getResultList();
    }

    @GET
    @Path("{id}")
    @RolesAllowed("uma_protection")
    public Phone getPhoneById(@PathParam("id") Long id) {
        TypedQuery<Phone> findPhoneById =
em.createNamedQuery("Phone.findById", Phone.class);
        findPhoneById.setParameter("id", id);
        return findPhoneById.getSingleResult();
    }
}
```

Κώδικας αρχείου PhoneResource.java

Με την κλάση JAXRSConfiguration δηλώσαμε ότι η εφαρμογή μας είναι προσβάσιμη με την χρήση web services.Όπως παρουσιάζεται και παραπάνω, με το annotation @Path δηλώνουμε ότι η κλάση PhoneResource θα είναι προσβάσιμη μέσω web

services και πιο συγκεκριμένα από την διαδρομη “resources/phones”.Η κλάση PhoneResource είναι ένα EJB δηλαδή είναι ένα Stateless Session Bean και μερικές απ τις δυνατότητες του είναι ότι μπορεί να γίνει pooled και μπορεί να συνδεθεί στην βάση δεδομένων με δυνατότητες συναλλαγών.Το annotation @Produces δηλώνει ότι το αποτέλεσμα τον μεθόδων θα μετατραπεί σε JSON μορφή.Πρόσβαση στη βάση δεδομένων γίνεται μέσω του EntityManager τον οποίο κάνουμε inject με τη χρήση του @PersistenceContext.Οι μέθοδοι getAllPhones() και getPhoneById() είναι προσβάσιμες με τη χρήση αιτήματος GET δηλώνοντάς το με τη χρήση του annotation @GET.Όταν κάνουμε αίτημα GET στην διαδρομή “resources/phones”, τότε καλείται η μέθοδος getAllPhones() η οποία χρησιμοποιεί τον EntityManager για να εκτελέσει το NamedQuery "Phone.findAll" και να επιστραφεί πίσω στον αιτούντα μια λίστα σε μορφή JSON που περιέχει πληροφορίες με όλα τα κινητά.Η getPhoneById() δέχεται παράμετρο το ID ενός κινητού και επιστρέφει αυτό το κινητό με μορφή JSON.Στο annotation @Path ορίζουμε μια διαδρομή απ την οποία πρέπει να εξάγουμε την πληροφορία για το ID.Για να είμαστε σε θέση να εξάγουμε πληροφορία από την διαδρομή πρέπει να τοποθετήσουμε κάποιο κομμάτι της διαδρομής σε άγκιστρα.Δηλαδη με το “@Path("{id}")” είμαστε στη θέση να ορίσουμε την παράμετρο της μεθοδού getPhoneBy().Για να ορίσουμε ποια παράμετρος θα πάρει τιμή και από που το δηλώνουμε με τη χρήση του annotation @PathParam("id").Και οι δυο παραπάνω μέθοδοί έχουν το annotation @RolesAllowed("uma_protection") με αυτό το τρόπο ορίζουμε ότι αυτές οι μέθοδοι θα έχουν πρόσβαση χρήστες ή εφαρμογές που έχουν τον ρόλο “uma_protection”.

8.6.5 Users microservice

Όπως αναφέραμε και παραπάνω το users microservice αναπαριστά τους χρήστες του καταστήματος και σκοπός αυτής της υπηρεσίες είναι να προσφέρει πληροφορίες και λειτουργίες σχετικά με τους χρήστες του συστήματος.Τα χαρακτηριστικά της κλάσης User είναι παρόμοια με την κλάση Phone.Οπότε παρακάτω θα περιγράψουμε μόνο τα επιπλέον χαρακτηριστικά που έχει η κλάση User.Η κλάση User αναπαριστά ένα χρήστη του συστήματος.Το NamedQuery “User.findByKeycloakID” δέχεται ως παράμετρο το ID του χρήστη που είναι κατοχυρωμένος στην βάση δεδομένων και αναζητά αν υπάρχει καταχωρημένος χρήστης με αυτό το ID και τον επιστρέφει.Το ID του χρήστη βλέπουμε ότι είναι πρωτεύων κλειδί και με το strategy που ορίζουμε στο “@GeneratedValue” ορίζουμε ότι αυτό το πεδίο αυξάνεται κατά ένα από την βάση δεδομένων.Με το “@Transient” στα πεδία “Username” και “Password” ορίζουμε όταν γίνεται η μετατροπή ενός User αντικειμένου σε JSON τα πεδία “Username” και “Password” να μην συμμετέχουν στο τελικό αποτέλεσμα.

```
@Entity
@XmlRootElement
```

```
@XmlAccessorType(XmlAccessType.FIELD)
@Table(name = "PHONE_USERS")
@NamedQueries({
    @NamedQuery(name = "User.findByKeycloakID", query = "Select u from
User u where u.KeycloakID=:id")
})
public class User {
    @Id
    @Column(name = "ID")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Transient
    private String Username;
    @Transient
    private String Password;
    @Column(name = "USER_FIRSTNAME")
    private String Firstname;
    @Column(name = "USER_LASTNAME")
    private String Lastname;
    @Column(name = "TELEPHONE")
    private String Telephone;
    @Column(name = "ADDRESS")
    private String Address;
    @Column(name = "EMAIL")
    private String Email;
    @Column(name = "KEYCLOAK_ID")
    private String KeycloakID;
    public User() {
    }
    public User(String firstName, String lastName, String id) {
        this.Firstname = firstName;
        this.Lastname = lastName;
        this.KeycloakID = id;
    }
   //output omitted
}
```

Κώδικας αρχείου User.java

Η κλάση “RealmResourceProvider” είναι ένα managed bean το οποίο χρησιμοποιώντας την βιβλιοθήκη του Keycloak παράγει ένα “RealmResource” με το οποίο έχουμε πρόσβαση στην υπηρεσία του Keycloak.

```
public class RealmResourceProvider {
```

```java
    //It's user's UUID
    private String clientSecret;
    private Keycloak keycloak;
    private RealmResource realm;

    @Produces
    public RealmResource masterRealm() {
        this.clientSecret = "ff88fe59-3040-4fcd-88d0-9236c89072d4";
        this.keycloak = KeycloakBuilder.builder()
                .serverUrl("http://sso:8080/auth")
                .realm("master")
                .clientId("admin-cli")
                .clientSecret(clientSecret)
                .username("admin")
                .password("admin")
                .build();
        this.realm = this.keycloak.realm("sso");
        return this.realm;
    }
}
```

Κώδικας αρχείου RealmResourceProvider.java

Η κλάση “UsersManager” είναι ένα Stateless Session Bean προσφέροντας τις υπηρεσίες που παρουσιάζονται παρακάτω.Αρχικά να αναφέρουμε ότι έκτος απ τον EntityManager που γίνεται inject, γίνεται inject ένα Instance του RealmResource.Ως Instance ορίζουμε ότι σε κάθε νέο αίτημα το συγκεκριμένο πεδίο του stateless bean θα γίνεται αρχικοποίηση κάθε φορά.Στη μέθοδο “getAllUsers()” χρησιμοποιώντας το πεδίο realmInstance έχουμε πρόσβαση στους χρήστες που είναι αποθηκευμένοι στην βάση δεδομένων της Keycloak υπηρεσίας.Κάνουμε ένα ερώτημα για να παραλάβουμε όλους τους χρήστες που βρίσκονται στο “sso” realm, τους μετατρέπουμε σε αντικείμενα της κλάσης User προσθέτοντας τους σε μια λίστα και επιστρέφοντας την τελική λίστα.Η μέθοδος “registerUser” δέχεται ένα αντικείμενο User.Έαν το πεδίο του αντικειμένου KeycloakID έχει ήδη αρχικοποιηθεί τότε καλεί την μέθοδο edit.Εάν όχι τότε κάνει εγγραφή τον χρήστη στην Keycloak υπηρεσία και έπειτα κάνει εγγραφή τον χρήστη στον πίνακα User του user microservice προσθέτοντας του, το KeycloakID που του επιστράφηκε από την εγγραφή του στην Keycloak υπηρεσια.Η μέθοδος “editPassword” δέχεται ως παράμετρο ένα UUID, ένα αντικείμενο UserRepresentaiton, που αναπαριστά το χρήστη στην Keyclaok υπηρεσία, και το password το οποίο θα αλλάξουμε.Η μέθοδος “editUser” δέχεται ένα αντικείμενο User το οποίο ενημερώνει τα στοιχεία του χρήστη στον πίνακα phone_users αλλά και στην βάση δεδομένων της Keycloak υπηρεσίας.Η μέθοδος “getCreatedId” δέχεται ως παράμετρο την απάντηση από την εγγραφή του χρήστη

στην Keycloak υπηρεσία επεξεργάζεται την απάντηση και επιστρέφει το UUID του χρήστη.Η μέθοδος “getUser” δέχεται το UUID σε μορφή string, χρησιμοποιώντας τον EntityManasger κάνει το NamedQuery “User.findByKeycloakID” βρίσκει τον χρήστη που έχει ως KeycloakID το UUID που πέρασε ως παράμετρος και τον επιστρέφει.

```
@Stateless
public class UsersManager {
    @PersistenceContext
    EntityManager em;
    @Inject
    private Instance<RealmResource> realmInstance;
    public List<User> getAllUsers() {
        List<UserRepresentation> results =
this.realmInstance.get().users().search(null);
        List<User> users = new ArrayList<>();
        results.forEach((user) -> users.add(new
User(user.getFirstName(), user.getLastName(), user.getId())));
        return users;
    }
    public User registerUser(User requestedUser) {
        System.out.println(requestedUser);
        if (requestedUser.getKeycloakID() != null) {
            return this.editUser(requestedUser);}
        UserRepresentation user = new UserRepresentation();
        user.setEmail(requestedUser.getEmail());
        user.setUsername(requestedUser.getUsername());
        user.setFirstName(requestedUser.getFirstname());
        user.setLastName(requestedUser.getLastname());
        user.setEnabled(true);
        Response response =
this.realmInstance.get().users().create(user);
        System.out.println("user created.... verify in keycloak!");
        String userId = getCreatedId(response);
        this.editPassword(userId, user, requestedUser.getPassword());
        System.out.println("role created.");
        requestedUser.setKeycloakID(userId);
        User newUser = this.em.merge(requestedUser);
        newUser.setUsername(requestedUser.getUsername());
        return newUser;
    }
    private void editPassword(String userId, UserRepresentation user,
String Passowrd) {
        UserResource userResource =
this.realmInstance.get().users().get(userId);
```

```
        userResource.update(user);
        CredentialRepresentation newCredential = new
CredentialRepresentation();
        newCredential.setType(CredentialRepresentation.PASSWORD);
        newCredential.setValue(Passowrd);
        newCredential.setTemporary(false);
        userResource.resetPassword(newCredential);
    }
    private User editUser(User requestedUser) {
        UserResource userResource =
this.realmInstance.get().users().get(requestedUser.getKeycloakID());
        UserRepresentation user = userResource.toRepresentation();
        user.setFirstName(requestedUser.getFirstname());
        user.setLastName(requestedUser.getLastname());
        user.setEmail(requestedUser.getEmail());
        if (requestedUser.getPassword() != null) {
            this.editPassword(requestedUser.getKeycloakID(), user,
requestedUser.getPassword());
        }
        userResource.update(user);
        requestedUser.setUsername(user.getUsername());
        this.em.merge(requestedUser);
        return requestedUser;
    }
    private String getCreatedId(Response response) {
        URI location = response.getLocation();
        if (!response.getStatusInfo().equals(Response.Status.CREATED)) {
            Response.StatusType statusInfo = response.getStatusInfo();
            throw new WebApplicationException("Create method returned
status "
                    + statusInfo.getReasonPhrase() + " (Code: " +
statusInfo.getStatusCode() + "); expected status: Created (201)",
response);
        }
        if (location == null) {return null;}
        String path = location.getPath();
        return path.substring(path.lastIndexOf('/') + 1);
    }
    public User getUser(String id) {
        TypedQuery<User> findUser =
this.em.createNamedQuery("User.findByKeycloakID", User.class);
        System.out.println("id = " + id);
        findUser.setParameter("id", id);
        return findUser.getSingleResult();
    }
```

```
}
```

Κώδικας αρχείου UsersManager.java

Η κλάση “UsersResource” ενθυλακώνει τις υπηρεσίας της “UsersManager” για την πρόσβαση αυτών μέσω web services.Οι υπηρεσίες είναι διαθέσιμες κάτω από το “resources/users”.Τα περισσότερα στοιχεία είναι ίδια με το PhoneResource οπότε θα αναφέρουμε μόνο τις διαφορές.Αρχικά κάνουμε inject το SessionContext το οποίο αναπαριστά την συνέδρια που εξυπηρετεί έναν χρήστη.Η αρχικοποίηση αυτού γίνετε σε κάθε εισερχόμενο αίτημα.Η μέθοδος “getAllUsers()” κάνοντας χρήση του “UserManager” επιστρέφει μια λίστα με όλους του χρήστες του συστήματος.Πρόσβαση σε αυτή μέθοδο έχουν χρήστες με “admin” ρόλο.Η μέθοδος “getUser()” είναι διαθέσιμη στην διαδρομή “resources/users/id”.Κάνει χρήση την κλάση “UserManager” και επιστρέφει τον αυθεντικοποιημένο χρήστη.Ο αυθεντικοποιημένος χρήστης αναγνωρίζεται από το SessionContext.Στη μέθοδο “registerUser” έχουν πρόσβαση μη αυθεντικοποιημένοι χρήστες, αυτό είναι αναγκαίο ώστε κάποιος πελάτης να είναι σε θέση να κάνει εγγραφή στο ηλεκτρονικό κατάστημα.Για να πετύχουμε τον παραπάνω σκοπό ορίζουμε πάνω από την μέθοδο το annotation @PermitAll.

```
@Path("users")@Stateless
@Produces(MediaType.APPLICATION_JSON + ";charset=UTF-8")
@Consumes(MediaType.APPLICATION_JSON + ";charset=UTF-8")
public class UsersResource {
    @Inject
    UsersManager usersManager;
    @Resource
    SessionContext ctx;
    @GET
    @RolesAllowed("admin")
    public List<User> getAllUsers() {
        return this.usersManager.getAllUsers();
    }
    @GET
    @Path("id")
    @RolesAllowed("uma_protection")
    public List<User> getUser() {
        User user = this.usersManager.getUser(this.getUserUUID());
        List<User> wrapper = new ArrayList<>();
        wrapper.add(user);
        return wrapper;
    }
    @POST
    @PermitAll
    public User registerUser(User requestedUser) {
```

```
        return this.usersManager.registerUser(requestedUser);
    }
    private String getUserUUID() {
        return ctx.getCallerPrincipal().getName();
    }
}
```

Κώδικα αρχείου UsersResource.java

8.6.6 Orders microservice

Όπως αναφέραμε και παραπάνω το orders microservice αναπαριστά τις παραγγελίες του καταστήματος και σκοπός αυτής της υπηρεσίες είναι να προσφέρει πληροφορίες και λειτουργίες σχετικά με τις παραγγελίες του συστήματος.Για την αποφυγή της χρήσης κοινόχρηστων βιβλιοθηκών έχουμε ορίσει ξανά την κλάση User και την κλάση Phone οι οποίες περιγράφουν ένα χρήστη του συστήματος και ένα κινητό τηλέφωνο αντίστοιχα.Όπως βλέπουμε παρακάτω οι κλάσεις User και Phone δεν έχουν annotation που να σχετίζονται με την αντιστοίχηση με τα πεδία της βάσης δεδομένων αλλά υπάρχουν annotation μόνο για την μετατροπή τους.

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class User {
    private String FirstName;
    private String LastName;
//setters and getters
}
```

Κώδικας αρχείου User.java

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Phone {
    private String Name;
    private String Brand;
    private String Cpu;
    private String ScreenSize;
    private String Date;
    private String Price;
    private String Image;
//setters and getters
}
```

Κώδικας αρχείου Phone.java

```
@Entity
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
```

```
@Table(name = "PHONE_ORDERS")
@NamedQueries({
    @NamedQuery(query = "SELECT o FROM Order o", name =
"Order.findAll"),
    @NamedQuery(query = "SELECT o FROM Order o where o.ID=:id", name =
"Order.findById"),
    @NamedQuery(query = "SELECT o FROM Order o where o.ID=:id AND
o.UserId=:userId", name = "Order.findByIdAndUserId"),
    @NamedQuery(query = "SELECT o FROM Order o where o.UserId=:userId",
name = "Order.ByUserId"),
})
public class Order {

    @Id
    @Column(name = "ID")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long ID;
    @Column(name = "USER_ID")
    private String UserId;
    @Transient
    private User User;
    @Transient
    private List<Phone> Phones;
    @Column(name = "ITEMS")
    private String Items;
    @Column(name = "INPROGRESS")
    private boolean InProgress;
    @Column(name = "COMPLETED")
    private boolean Completed;
    @Column(name = "CANCELLED")
    private boolean Cancelled;
    @Column(name = "TOTAL_PRICE")
    private BigDecimal TotalPrice;

    public Order() {
    }
    @PrePersist
    public void PrePersist() {
        this.InProgress = true;
    }
//setters and getters
}
```

Κώδικας αρχείου Order.java

Τα χαρακτηριστικά της κλάσης Order είναι παρόμοια με την κλάση κλάση Phone του phones microservice και την κλάση User του users microservice.Οπότε παρακάτω

θα περιγράψουμε μόνο τα επιπλέον χαρακτηριστικά που έχει η κλάση Order.Η Order αναπαριστά μια παραγγελία του συστήματος.Το πρώτο NamedQuery “Order.findAll” επιστρέφει όλες τις παραγγελίες που έχει ο πίνακας “PHONE_ORDERS”.Το NamedQuery “Order.findById” επιστρέφει μόνο μια παραγγελία η οποία ταιριάζει στην παράμετρο “id” που τοποθετεί ο χρήστης.Το “Order.findByIdAndUserId” αναζητεί την παραγγελία με συγκεκριμένο “id” του χρήστη με το “userId”.Η “Order.ByUserId” επιστρέφει όλες τις παραγγελίες του χρήστη με id ίσο με το “userId”.Στη μέθοδο PrePersist() ορίσαμε το annotation @PrePersist.Με αυτό το τρόπο ορίζουμε την JPA πριν αποθηκεύσει την κατάσταση αυτού του αντικειμένου να εκτελέσει αυτή την μέθοδο.Όπως βλέπουμε όταν γίνεται καταχώριση μιας παραγγελίες η κατάσταση “InProgress” γίνεται αληθής, δηλαδή η παραγγελία βρίσκεται σε εξέλιξη.

```
public class RealmResourceProducer {

    private String clientSecret;
    private Keycloak keycloak;
    private RealmResource realm;

    @Produces
    public String keycloakToken() {

        this.clientSecret = "6f50c1c1-46db-43a5-88cc-37d098798b48";
        this.keycloak = KeycloakBuilder.builder()
                .serverUrl("http://localhost/auth")
                .realm("sso")
                .clientId("sso")
                .clientSecret(clientSecret)
                .username("admin")
                .password("admin").grantType("password")
                .build();

        return this.keycloak.tokenManager().getAccessToken().getToken();
    }
}
```

Κώδικας αρχείου RealmResourceProducer.java

Όπως παρατηρούμε και παραπάνω η μέθοδος keycloakToken της κλάση RealmResourceProducer, επικοινωνεί με την Keycloak υπηρεσία αιτώντας ένα κλειδί που θα χρησιμοποιηθεί για την ασφαλή επικοινωνία μεταξύ των microservices.

```
@Stateless
public class OrderManager {
    @PersistenceContext
```

```
    EntityManager em;
    @Resource
    SessionContext ctx;
    @Inject
    String KeycloakToken;
    @Inject
    RealmResource rr;
    //public Order save(Order order){}
    //public void updateOrderStatus(Long orderId, String status) {}
    //private Order findOrderById(Long orderId) {}
    //private String getUserUUID() {}
    //public List<Order> getAllOrders() {}
    //public Order getOrderById(@PathParam("id") Long id) {}
    private BigDecimal getItemPrice(long id) {
        String link = "http://localhost:8080/phones/resources/phones/" +
id;
        WebTarget target = ClientBuilder.newClient().target(link);

        StringReader stringReader = new
StringReader(target.request(MediaType.APPLICATION_JSON).header("Authoriz
ation", "Bearer " + this.KeycloakToken).get(String.class));
        try (JsonReader jsonReader = Json.createReader(stringReader)) {
            String TotalPrice =
jsonReader.readObject().getString("Price");
            return new BigDecimal(Integer.parseInt(TotalPrice));
        }
    }
    private JsonObject getItem(long id) {
        String link = "http://localhost:8080/phones/resources/phones/" +
id;
        WebTarget target = ClientBuilder.newClient().target(link);
        StringReader stringReader = new
StringReader(target.request(MediaType.APPLICATION_JSON).header("Authoriz
ation", "Bearer " + this.KeycloakToken).get(String.class));
        try (JsonReader jsonReader = Json.createReader(stringReader)) {

            return jsonReader.readObject();
        }
    }
 //   public List<Order> getAllOrdersByUserId(String requestedUserId) {}
//    private boolean isAdmin(String userId) {}
}
```

Κώδικας αρχείου OrderManager.java

Η κλάση OrderManager εκτελεί παρόμοιες λειτουργίες με τις κλάσεις UserManager και PhoneManager οπότε θα αναφέρουμε μόνο αυτά που έχουν ενδιαφέρον για αυτή

την κλάση.Οι μέθοδοι “getItem” και “getItemPrice” έχουν ιδιαίτερο ενδιαφέρον διότι είναι οι μοναδικές που επικοινωνούν με το phones microservice μέσω δικτύου.Για να είναι εφικτή η επικοινωνία και οι δύο μέθοδοι κάνουν αίτημα στη Keycloak υπηρεσία για το Authorization Bearer Token ώστε το phones microservice να μπορέσει να αυθεντικοποίηση το αίτημα ότι προέρχεται από έμπιστη πηγή.Οι μεθόδοι “getItem” και η “getItemPrice” δέχονται ως παράμετρο ένα id κινητού.Οι παραπάνω μέθοδοι εκτελούν αίτημα στο phones microservice για να λάβουν πληροφορίες σχετικά με το κινητό τηλέφωνο που φέρει αυτό το id.Τελικά η μέθοδος “getItem” επιστρέφει το κινητό σε JSON μορφή ενώ η μέθοδος “getItemPrice” επιστρέφει την τιμή του κινητού.Η κλάση OrdersResource είναι παρόμοια με την PhonesResource και UsersResource.Ο ρόλος της OrdersResource είναι να ενθυλακώνει τις λειτουργίες της κλάσης OrderManager και να τις προσφέρει σε άλλες υπηρεσίες μέσω web services.

8.7 Ηλεκτρονικό κατάστημα

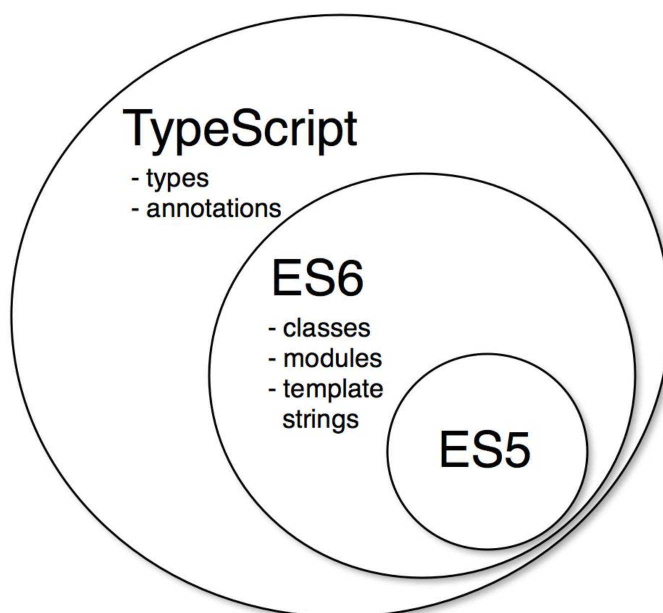
Για την υλοποίηση του ηλεκτρονικού καταστήματος χρησιμοποιήθηκε η Angular πλατφόρμα.Όπως ορίσαμε και παραπάνω ένας πελάτης του καταστήματος είναι σε θέση να αγοράζει κινητά της αρέσκεια του.Στην εφαρμογή υλοποιήθηκαν οι εξής δυνατότητες, η εγγραφή του χρήστη στο κατάστημα,προσθήκη προϊόντων στο καλάθι του καταστήματος, η εμφάνιση τον προϊόντων της αποθήκης και η εμφάνιση των παραγγελιών.Όλες αυτές οι υπηρεσίες προσφέρονται από τα microservices που υλοποιήσαμε προηγουμένος.Σκόπος αυτής της εφαρμογής είναι η επικοινωνία με τα microservices και παρουσιασής των παραπάνω λειτουργιών με γραφικό τρόπο.Με την κλάση RestClient θα εκτελούμε ερωτήματα στα microservices και με την angular πλατφόρμα θα τα παρουσιάσουμε.

8.7.1 Angular

Η πλατφόρμα της Angular κάνει εύκολη, γρήγορη και ασφαλή την ανάπτυξη εφαρμογών πελάτη διαδικτύου και ένας απ τους παράγοντες που πετυχαίνει όλα τα παραπάνω είναι λόγο της γλώσσας προγραμματισμού που χρησιμοποιηθεί που είναι η Typescript.Η πλατφόρμα συνδυάζει declarative templates, dependency injection και end-to-end εργαλεία περιλαμβάνοντας τις καλύτερες πρακτικές για την ανάπτυξη τέτοιου είδους εφαρμογών.Με τις παραπάνω δυνατότητες μπορούμε να δημιουργήσουμε εφαρμογές πελάτη που είναι ικανές ναι λειτουργούν σε κινητά τηλέφωνα και προσωπικούς υπολογιστές.Απ τα βασικά χαρακτηριστικά της Angular είναι το Dependency Injection.Η λογική της χρήσης του είναι ίδια με αυτή της Java EE.Δηλαδή,η παροχή πολύπλοκων υπηρεσιών με πολύ απλό, εύκολο και ασφαλή τρόπο.Προσφέρει συγχρονισμό δεδομένων μεταξύ κλάσεων και του DOM το οποίο ονομάζεται data binding.

8.7.2 Typescript

Η Microsoft κάνει ανάπτυξη της ανοιχτού κώδικα γλώσσας προγραμματισμού Typescript.Η Typescript είναι μια script γλώσσα και αναπτύσσεται στην JavaScript γλώσσα.Βασικό της χαρακτηριστικό είναι ότι είναι αρκετά ίδια με άλλες γλώσσες αντικειμενοστραφούς γλώσσες προγραμματισμού όπως η Java.Η Typesrcipt χρησιμοποιείται για την ανάπτυξη JavaScript εφαρμογών είτε στην μεριά του πελάτη είτε στην μεριά του διακομιστή.Ωστόσο οι angular εφαρμογές μπορούν να δημιουργηθούν με γλώσσες προγραμματισμού όπως η Typescript, η ES5, η ES6 και η Dart.



8.3 Αρχιτεκτονική Typescript

8.7.3 Πελάτης RestClient

Η επικοινωνία της angular εφαρμογής, με τα microservices που υλοποιήσαμε, γίνεται με REST.Τα microservices δέχονται αιτήματα μόνο από έμπιστες πηγές.Για να μπορέσει η επικοινωνία να είναι ασφαλής προστέθηκαν στην εφαρμογή τρία αρχεία.Το keycloak.js αρχείο το οποίο είναι υπεύθυνο για την αυθεντικοποίηση με τα microservices το οποίο ρυθμίζεται από το αρχείο keycloak.json.Το τρίτο αρχείο είναι ένας πελάτης με όνομα RestClient.Ο πελάτης RestClient χρησιμοποιεί το keycloak.js ώστε να κάνει αιτήματα στα microservices τα οποία είναι σε θέση να αυθεντικοποιηθούν τα εισαρχόμενα αιτήματα.Επίσης το RestClient μπορεί να κάνει αιτήματα τα οποία δεν χρησιμοποιούν το keycloak.js.

```
@Injectable()
```

```
export class RestClient {

  //<output omitted>

   get(PATH: string): any {
       return this.KeycloakService.getToken().then(token =>
this.getRequest(token, PATH));
   }

   private getRequest(token, PATH): Observable<any> {
       return this.http.get(PATH, <RequestOptionsArgs>{
           headers: new Headers({
               'Authorization': 'Bearer ' + token
           })
       })
           .map(this.extractData)
           .catch(this.handleError);
   }
   post(PATH: string, Body): any {
       return this.KeycloakService.getToken().then(token =>
this.postRequest(token, PATH, Body));
   }
   private postRequest(token, PATH, bodyData): Observable<any> {
       let headers = new Headers({
           'Accept': 'application/json, text/plain, */*',
           'Content-Type': 'application/json'
       });
       if (token != null) {
           headers.append("Authorization", 'Bearer ' + token)
       }
       return this.http.post(PATH, bodyData, <RequestOptionsArgs>{
           headers: headers
       }).map(this.extractData)
           .catch(this.handleError);
   }
   postNoAuth(PATH: string, Body): any {
       return new Promise((resolve, reject) => this.postRequest(null, PATH,
Body).subscribe(result => resolve(result)));
   }
//<output omitted>
}
```

Κώδικας αρχείου RestClient.ts

Απ τον παραπάνω κώδικα μπορούμε να διακρίνουμε ότι ο RestClient έχει την δυνατότητα να εκτελέσει αυθεντικοποιημένα GET και POST αιτήματα αλλά και μη αυθεντικοποιημένα POST αιτήματα.Η κλάση KeycloakService ενθυλακώνει τις λειτουργίες της keycloak.js βιβλιοθήκης.Για να είναι σε θέση ο RestClient να κάνει αυθεντικοποιημένα GET και POST αιτήματα είναι αναγκαστικό ένα JWT κλειδί.Με

την μέθοδο “getToken()” της KeycloakService κλάσης αιτούμαστε ένα token το οποίο τοποθετούμε στα headers του http αιτήματος ως “'Authorization': 'Bearer ' + token”.Η μέθοδο “get” δέχεται ως παράμετρο την διαδρομή που θα κάνουμε αίτημα ενώ στις μεθόδους “post” δέχεται επιπλέον παράμετρο το σώμα του αιτήματος post.Ο RestClient έχει την δυνατότητα να κάνει post αιτήματα τα οποία δεν χρειάζονται αυθεντικοποίηση.Σε όλες τις παραπάνω περιπτώσεις το αποτέλεσμα του αιτήματος μετατρέπεται από String σε JSON.

8.7.4 Μακέτα εφαρμογής

Παρακάτω φαίνεται η μακέτα της εφαρμογής.Το πρώτο βασικό συστατικό στη παραπάνω μακέτα είναι το top-menu το οποίο είναι ένα στατικό μενού.Το δεύτερο δυναμικό στοιχείο της εφαρμογής είναι το router-outlet.Στο στοιχείο router-outlet αναλόγως με τις ενέργειες του χρήστη αντικαταστατaί κάθε φορά το γραφικό περιβάλλον σε αυτό το στοιχείο.Όλες οι λειτουργίες της εφαρμογής χρησιμοποιούν τον RestClient για να κάνουν αιτήματα στα microservices και το αποτέλεσμα των αιτημάτων επιστρέφεται σε γραφικό περιβάλλον στο router-outlet.

```
<div class="ui-g">
   <div class="ui-g-12">
       <top-menu></top-menu>
   </div>
   <div class="ui-g-12">
       <router-outlet></router-outlet>
   </div>
   <div class="ui-g-12">
       <strong>phone.js</strong>
   </div>
</div>
<loading *ngIf="isLoading" style="z-index:1000;position: fixed;height:
100%;text-align: center;top: 0;right:0;left:0;background-color:
#f3f3f3;width: 100%;"></loading>
<p-confirmDialog header="Confirmation" icon="fa fa-question-circle"
responsive="true"></p-confirmDialog>
```

10. Βιβλιογραφία

1. Virtualization https://en.wikipedia.org/wiki/Virtualization
2. Hypervisor - https://en.wikipedia.org/wiki/Hypervisor
3. Operating system level virtualization - https://en.wikipedia.org/wiki/Operating-system-level_virtualization
4. Linux namespaces - https://en.wikipedia.org/wiki/Linux_namespaces
5. cgroups - https://en.wikipedia.org/wiki/Cgroups
6. Containerization - https://en.wikipedia.org/wiki/Containerization
7. Docker (software) - https://en.wikipedia.org/wiki/Docker_(software)
8. Docker frequently asked questions - https://docs.docker.com/engine/faq
9. The Docker platform overview - https://docs.docker.com/engine/docker-overview
10. Docker Engine - https://docs.docker.com/engine/
11. Monolithic application - https://en.wikipedia.org/wiki/Monolithic_application
12. What is a Monolith - http://www.codingthearchitecture.com/2014/11/19/what_is_a_monolith.html
13. Microservices - https://en.wikipedia.org/wiki/Microservices
14. The Scale Cube - http://microservices.io/articles/scalecube.html
15. Service-oriented architecture - https://en.wikipedia.org/wiki/Service-oriented_architecture
16. Microservices a definition of this new architectural term - https://martinfowler.com/articles/microservices.html
17. Microservice Trade-Offs - https://martinfowler.com/articles/microservice-trade-offs.html
18. Microservices and Data Architecture – Who Owns What Data? - https://genehughson.wordpress.com/2014/06/20/microservices-and-data-architecture-who-owns-what-data/
19. The Art of Scale: Microservices, The Scale Cube and Load Balancing - https://devcentral.f5.com/articles/the-art-of-scale-microservices-the-scale-cube-and-load-balancing
20. Microservices: Decomposing Applications for Deployability and Scalability - https://www.infoq.com/articles/microservices-intro
21. Java Platform, Enterprise Edition - https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition
22. Java Community Process - https://en.wikipedia.org/wiki/Java_Community_Process
23. Java Platform, Enterprise Edition , The Java EE Tutorial Release 7 - https://docs.oracle.com/javaee/7/JEETT.pdf
24. Java Platform, Enterprise Edition (Java EE) - http://www.theserverside.com/definition/J2EE-Java-2-Platform-Enterprise-Edition