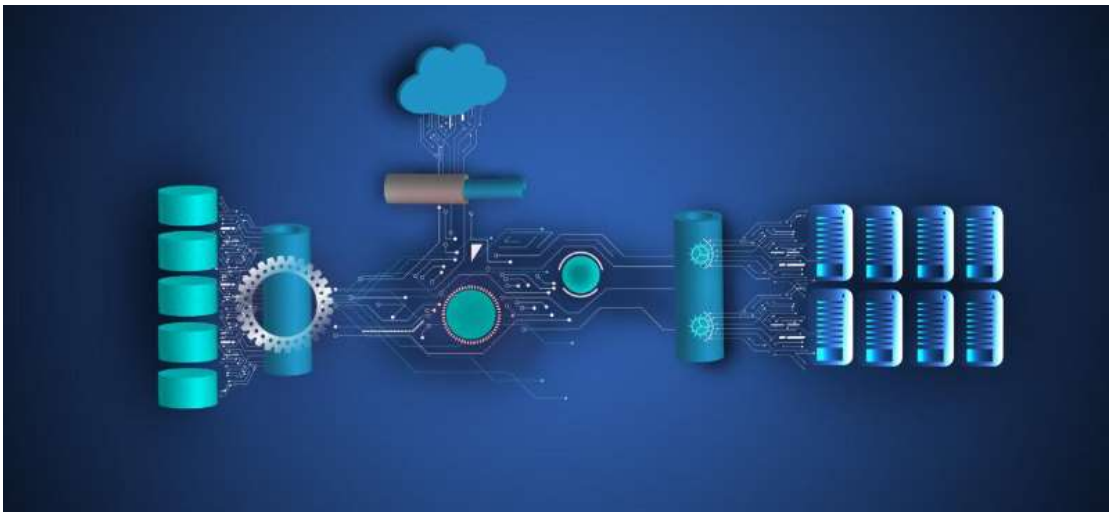
# ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗ
ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ ΣΤΗ ΔΙΟΙΚΗΣΗ ΕΠΙΧΕΙΡΗΣΕΩΝ


**ΜΕΤΑΒΑΣΗ ΜΙΑΣ ΔΙΑΔΙΚΤΥΑΚΗΣ ΓΕΩΧΩΡΙΚΗΣ ΕΦΑΡΜΟΓΗΣ ΑΠΟ ΤΟ ΜΟΝΟΛΙΘΙΚΟ ΜΟΝΤΕΛΟ, ΣΕ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΜΙΚΡΟΫΠΗΡΕΣΙΩΝ**

Μεταπτυχιακή εργασία


**Δαμαλά Θωμαή**

Αθήνα, 2022

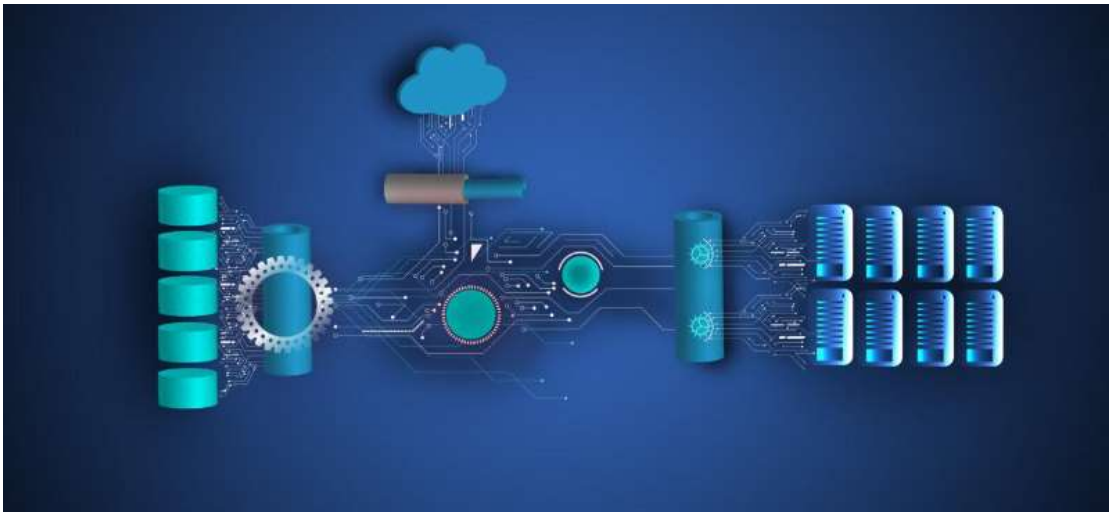
# HAROKOPIO UNIVERSITY

SCHOOL OF DIGITAL TECHNOLOGY
DEPARTMENT OF INFORMATICS AND TELEMATICS
POSTGRADUATE PROGRAMME IN INFORMATICS AND TELEMATICS
INFORMATION SYSTEMS IN BUSINESS ADMINISTRATION


**LOCATION BASED WEB APPLICATION: FROM MONOLITH TO MICROSERVICES APPROACH**

Master Thesis


**Damala Thomai**

Athens, 2022

# ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗ
ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ ΣΤΗ ΔΙΟΙΚΗΣΗ ΕΠΙΧΕΙΡΗΣΕΩΝ

**Τριμελής Εξεταστική Επιτροπή**

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

**Δαλάκας Βασίλειος**
**Ε.ΔΙ.Π, Τμήμα Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο Αθηνών**

**Τσερπές Κωνσταντίνος**
**Αναπληρωτής Καθηγητής, Τμήμα Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο Αθηνών**

Η Δαμαλά Θωμαή δηλώνω υπεύθυνα ότι:

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

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

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

# Σελίδα για Αφιέρωση

στο γιο μου, Νικόλα
το σπουδαιότερο επίτευγμα της ζωής μου

## ΕΥΧΑΡΙΣΤΙΕΣ

Η υψηλή κατάρτιση των διδασκόντων και η πρόθεσή τους να μεταλαμπαδεύσουν τις γνώσεις τους σε νέους ανθρώπους χρήζει ιδιαίτερης μνείας.

Θα ήθελα να ευχαριστήσω όλο το διδακτικό και διοικητικό προσωπικό του Χαροκοπείου Πανεπιστημίου Αθηνών, που με την αφοσίωση στο καθήκον, την επιμέλεια και την προθυμία τους άμβλυναν τον δύσκολο δρόμο της κατάκτησης της γνώσης, μετατρέποντας την πορεία αυτή σε ένα μαγικό ταξίδι.

## ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

## Περίληψη στα Ελληνικά

Σκοπός της παρούσας διπλωματικής εργασίας είναι η μελέτη των διαφόρων αρχιτεκτονικών των πληροφοριακών συστημάτων, η αναφορά των ιδιαίτερων χαρακτηριστικών τους, καθώς και των πλεονεκτημάτων και μειονεκτημάτων αυτών, και τέλος η επιλογή της σωστής αρχιτεκτονικής κατά το σχεδιασμό ενός πληροφοριακού συστήματος ανάλογα με το είδος της υπό ανάπτυξη εφαρμογής.

Σκοπός της παραπάνω κατανόησης είναι η μελέτη μετάπτωσης μιας μονολιθικής εφαρμογής γεωχωρικών δεδομένων από το μονολιθικό μοντέλο σε μικροϋπηρεσίες.

Πέραν της ερευνητικής βιβλιογραφικής εργασίας, η εκπόνηση της παρούσας διπλωματικής εργασίας οδηγεί στην ανάπτυξη τεχνικών δεξιοτήτων, καθώς στήθηκε Linux λειτουργικό σύστημα, έγινε η εγκατάσταση της μονολιθικής εφαρμογής με όλες τις απαραίτητες παραμετροποιήσεις και τέλος η μελέτη του Django Framework της Python.

Προκειμένου να ολοκληρωθεί η ανάλυση και η καταγραφή της υφιστάμενης υλοποίησης, καλλιεργήθηκαν δεξιότητες όπως η ικανότητα της ανάλυσης των απαιτήσεων και η απεικόνιση της υφιστάμενης και της προτεινόμενης υλοποίησης υπό το πρίσμα ενός αρχιτέκτονα συστημάτων.



**Λέξεις κλειδιά:** Μονολιθική εφαρμογή, Μικροϋπηρεσίες, Αρχιτεκτονικές Πληροφοριακών Συστημάτων

## Abstract

The purpose of this thesis is the study of the various architectures of the information systems, the reference of their particular characteristics, as well as their advantages and disadvantages, and finally the choice of the right architecture when designing an information system depending on the type of the application.

The purpose of the above understanding is to design the migration of a geospatial application from monolith to microservices model.

In addition to the bibliographic research work, the preparation of this thesis leads to the development of technical skills, as a Linux operating system was set up, the monolithic application with all the necessary parameters was installed and finally the study of Python's Django Framework was conducted.

In order to complete the analysis and documentation of the existing implementation, skills such as the ability to analyse requirements and visualize the existing and proposed implementation from the perspective of a systems' architect were cultivated.



**Keywords:** Monolithic application, Microservices, Information Systems' Architectures

# ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

# ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ



# ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ - ΔΙΑΓΡΑΜΜΑΤΩΝ

# ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ/ΑΚΡΩΝΥΜΙΑ

| | |
|---|---|
| ΤΠΕ | Τεχνολογίες Πληροφορικής και Τηλεπικοινωνιών |
| | |
| AMQP | Advanced Messaging Queue Protocol |
| AOP | Aspect-Oriented Programming |
| BDUF | Big Design Upfront |
| CI/CD | Continuous Integration / Continuous Delivery |
| CNCF | Cloud Native Computing Foundation |
| DDD | Domain Driven Design |
| DRY | Don’t repeat yourself |
| EAI | Enterprise Application Integration |
| ESB | Enterprise Service Bus |
| IDE | Integrated Development Environment |
| IDP | Third-party Identity Provider |
| ISP | Interface Segregation Principle |
| LoD | Law of Demeter |
| MSA | Microservice Architecture |
| OS | Operating System |
| OWS | Open Web Service |
| RP | Relying Party |
| SPA | Single-Page App |
| SRP | Single Responsibility Principle |
| SoC | Separation of Concerns |
| URI | Uniform Resource Identifier |
| YAGNI | You ain’t gonna need it |

# ΕΙΣΑΓΩΓΗ

Ένα πληροφοριακό σύστημα είναι ένα σύνολο από διασυνδεδεμένες συσκευές, εφαρμογές και πόρους. Η αρχιτεκτονική είναι ένα μοντέλο που ορίζει τη δομή, τη συμπεριφορά και τις όψεις ενός πληροφοριακού συστήματος (Jaakkola, 2010).

Σύμφωνα με τον Kruchten (1999; 2006;) και Kruchten et al. (2008): "*Η αρχιτεκτονική ενός πληροφοριακού συστήματος εμπεριέχει το σύνολο των σημαντικών αποφάσεων σχετικά με την οργάνωση του συστήματος που περιλαμβάνουν την επιλογή των δομικών στοιχείων και των διασυνδέσεών του, τη συμπεριφορά του, όπως ορίζεται στη συνεργασία μεταξύ των συστατικών του, τη σύνθεση αυτών των διαρθρωτικών συστατικών και των στοιχείων συμπεριφοράς σε μεγαλύτερα υποσυστήματα με ένα αρχιτεκτονικό στυλ που καθοδηγεί αυτή την οργάνωση. Επίσης, η αρχιτεκτονική ενός πληροφοριακού συστήματος εμπεριέχει αποφάσεις σχετικά με τη λειτουργικότητα (functionality), τη χρηστικότητα (usability), την ανθεκτικότητα (resilience), τις επιδόσεις (performance), την επαναχρησιμοποίηση (reuse), τον εύληπτο χαρακτήρα (comprehensibility), τους οικονομικούς και τους τεχνολογικούς περιορισμούς (economic and technology constraints), αλλά και την αισθητική του συστήματος (aesthetics)*".

Η αρχιτεκτονική ενός πληροφοριακού συστήματος αποτελεί τη γέφυρα μεταξύ των επιχειρηματικών και των τεχνικών απαιτήσεων, η οποία επιτυγχάνεται με την καταγραφή και αντιστοίχιση των περιπτώσεων χρήσης του συστήματος με τις τεχνικές λύσεις που τις υλοποιούν. Συνεπώς, στόχος της αρχιτεκτονικής θα πρέπει να είναι ο προσδιορισμός των απαιτήσεων που επηρεάζουν τη δομή της τεχνικής λύσης και γενικότερα του πληροφοριακού συστήματος. Μια καλή αρχιτεκτονική, επομένως, μειώνει τους επιχειρηματικούς κινδύνους που συνδέονται με τη δημιουργία του πληροφοριακού συστήματος. Επιπλέον, μια ευέλικτη σχεδίαση επιτρέπει την αποτελεσματική διαχείριση των αλλαγών που θα προκύψουν κατά τη διάρκεια ζωής του πληροφοριακού συστήματος, οι οποίες μπορεί να προέρχονται είτε από αλλαγές στο λογισμικό συστημάτων, είτε από αλλαγές στο hardware, είτε από αλλαγές των αναγκών της επιχείρησης (Bass et al., 2012). Τέλος, η αρχιτεκτονική ομογενοποιεί το σύνολο, δίνοντας δομή στο πληροφοριακό σύστημα (Garlan, 1994).

# Κεφ.1 Αρχιτεκτονική και σχεδιασμός πληροφοριακών συστημάτων

Η αρχιτεκτονική ενός πληροφοριακού συστήματος λαμβάνει υπόψη τις ακόλουθες απαιτήσεις (Microsoft Patterns & Practices Team, 2009):

- Τις επιχειρησιακές διεργασίες που θα υποστηρίξει το πληροφοριακό σύστημα.
- Τις υπάρχουσες υποδομές Τεχνολογιών Πληροφορικής και Τηλεπικοινωνιών (ΤΠΕ) της επιχείρησης.
- Τις ανάγκες που θα δημιουργήσει το πληροφοριακό σύστημα. Για παράδειγμα, ανάγκες σε ποιότητα και διαφύλαξη δεδομένων, ανάγκες σε ασφάλεια, σε διαχειρισιμότητα και σε επεκτασιμότητα.
- Τα εμπλεκόμενα μέρη στη διαμόρφωση της αρχιτεκτονικής και τους χρήστες του συστήματος.
- Την εμβέλεια της επιχείρησης. Η εμβέλεια αφορά στη γεωγραφική έκταση στην οποία δραστηριοποιείται η επιχείρηση και επηρεάζει εκτός από το μέγεθος, διάφορες παραμέτρους του συστήματος (π.χ. χρησιμοποιούμενες γλώσσες, ζώνες ώρας).
- Τα δεδομένα που παράγονται και τηρούνται στο σύστημα.

## 1.1 Βασικές Αρχές Σχεδιασμού - Key Design Principles

Κατά τον σχεδιασμό μιας εφαρμογής θα πρέπει να ληφθούν υπόψη οι παρακάτω “Βασικές Αρχές Σχεδιασμού” που θα βοηθήσουν τον αρχιτέκτονα-σχεδιαστή στη δημιουργία μιας αρχιτεκτονικής που να ακολουθεί αποδεδειγμένες αρχές, προκειμένου να ελαχιστοποιηθεί το κόστος και οι απαιτήσεις συντήρησης και ταυτόχρονα να επιτυγχάνονται η χρηστικότητα και η επεκτασιμότητα:

- **Διαχωρισμός ενδιαφερόντων (Separation of concerns):** Διαχωρισμός της εφαρμογής σε ξεχωριστές λειτουργίες με όσο το δυνατόν λιγότερη επικάλυψη λειτουργικότητας. Σημαντικός παράγοντας για την επίτευξη υψηλής συνοχής και χαμηλής σύζευξης είναι η ελαχιστοποίηση των σημείων αλληλεπίδρασης. Ωστόσο, ο διαχωρισμός της λειτουργικότητας σε λανθασμένα όρια μπορεί να οδηγήσει σε υψηλή σύζευξη και πολυπλοκότητα μεταξύ των χαρακτηριστικών, παρόλο που η περιεχόμενη λειτουργικότητα σε ένα χαρακτηριστικό δεν επικαλύπτεται σημαντικά (Aksit et al., 2001; Ernst, 2003).
- **Αρχή της Μοναδικής Αρμοδιότητας (SRP: Single Responsibility Principle):** Κάθε στοιχείο ή λειτουργική μονάδα θα πρέπει να είναι υπεύθυνη μόνο για ένα συγκεκριμένο χαρακτηριστικό ή λειτουργικότητα ή συνάθροιση συνεκτικών λειτουργιών (Martin, 2018; Noback, 2018).

- **Αρχή της Ελάχιστης Γνώσης (Principle of Least Knowledge / LoD: Law of Demeter):** Ένα στοιχείο ή αντικείμενο δεν πρέπει να γνωρίζει τις εσωτερικές λεπτομέρειες άλλων στοιχείων ή αντικειμένων (Minsky et al., 1996, Vernon, 2015).
- **Μια μοναδική πηγή για κάθε στοιχείο (DRY: Don't repeat yourself):** Στόχος αυτής της αρχής ανάπτυξης λογισμικού είναι η μείωση της επανάληψης μοτίβων, αντικαθιστώντας το με αφαιρέσεις ή χρησιμοποιώντας κανονικοποίηση δεδομένων για την αποφυγή πλεονασμών. Κάθε λειτουργικότητα θα πρέπει να υλοποιείται μόνο σε ένα στοιχείο. Η λειτουργικότητα δεν πρέπει να αντιγράφεται σε κανένα άλλο στοιχείο. Όταν η αρχή αυτή εφαρμόζεται με επιτυχία, μια τροποποίηση οποιουδήποτε μεμονωμένου στοιχείου ενός συστήματος δεν απαιτεί αλλαγή σε άλλα λογικά άσχετα στοιχεία (Sciore, 2018).
- **Ελαχιστοποίηση του αρχικού σχεδιασμού (Minimize upfront design):** Ο αρχικός σχεδιασμός θα πρέπει να περιλαμβάνει μόνο ό,τι είναι απαραίτητο. Σε ορισμένες περιπτώσεις, μπορεί να χρειαστεί ο εκ των προτέρων ολοκληρωμένος σχεδιασμός και η δοκιμή του κόστους ανάπτυξης αν μια αποτυχία στο σχεδιασμό αποδεικνύεται να είναι μελλοντικά πολύ υψηλή. Σε άλλες περιπτώσεις, ειδικά στις περιπτώσεις της ευέλικτης ανάπτυξης (agile development), συνίσταται η αποφυγή της μεγάλης εκ των προτέρων σχεδίασης (BDUF: Big Design Upfront). Εάν οι απαιτήσεις της εφαρμογής είναι ασαφείς ή εάν υπάρχει πιθανότητα ο σχεδιασμός να εξελιχθεί με την πάροδο του χρόνου, δεν θα πρέπει να καταβάλλεται πρόωρα, μεγάλη προσπάθεια σχεδίασης. Αυτή η αρχή είναι γνωστή και ως «Δεν θα το χρειαστείς» (YAGNI: You ain't gonna need it) (Beck et al., 2004).

Κατά το σχεδιασμό μιας εφαρμογής ή ενός συστήματος, ο στόχος ενός αρχιτέκτονα λογισμικού είναι η ελαχιστοποίηση της πολυπλοκότητας μέσω του διαχωρισμού του σχεδίου σε διαφορετικούς τομείς ενδιαφέροντος. Για παράδειγμα, η διεπαφή χρήστη (UI), η επιχειρηματική επεξεργασία (business processing) και η πρόσβαση σε δεδομένα (data access) αντιπροσωπεύουν όλα διαφορετικούς τομείς ενδιαφέροντος. Σε κάθε περιοχή, τα υπό σχεδίαση στοιχεία θα πρέπει να επικεντρώνονται σε αυτήν τη συγκεκριμένη περιοχή και δεν θα πρέπει να συνδυάζουν κώδικα από άλλους τομείς ενδιαφέροντος. Έτσι, τα στοιχεία επεξεργασίας διεπαφής χρήστη δεν θα πρέπει να περιλαμβάνουν κώδικα που έχει άμεση πρόσβαση σε μια πηγή δεδομένων, αλλά θα πρέπει να χρησιμοποιούν είτε επιχειρησιακά στοιχεία είτε στοιχεία πρόσβασης δεδομένων για την ανάκτηση των δεδομένων.

Θα πρέπει επίσης να ολοκληρωθεί μια αποτίμηση κόστους/αξίας για την επένδυση της εφαρμογής ή του συστήματος. Γενικά, θα πρέπει να εξεταστούν τα λειτουργικά όρια τόσο από επιχειρηματική όσο και από οικονομική άποψη.

Οι παρακάτω οδηγίες υψηλού επιπέδου βοηθούν στην εξέταση ενός μεγάλου φάσματος παραγόντων που μπορούν να επηρεάσουν την ευκολία του σχεδιασμού, της ανάπτυξης, της δοκιμής και της διατήρησης μιας εφαρμογής ή ενός συστήματος (Microsoft Patterns & Practices Team, 2009).

## 1.2 Πρακτικές Σχεδιασμού - Design Practices

- **Διατήρηση των μοτίβων σχεδίασης σε κάθε στρώμα**. Μέσα σε ένα λογικό επίπεδο και όπου είναι δυνατόν, ο σχεδιασμός των επιμέρους εξαρτημάτων θα πρέπει να είναι συνεπής για μια συγκεκριμένη λειτουργία. Για παράδειγμα, εάν επιλεγεί το μοτίβο Table Data Gateway για τη δημιουργία ενός αντικειμένου που λειτουργεί ως πύλη σε πίνακες μιας βάση δεδομένων, δεν θα πρέπει να συμπεριληφθεί ένα άλλο μοτίβο όπως το Repository, το οποίο χρησιμοποιεί διαφορετικό πρότυπο (paradigm) για την πρόσβαση στα δεδομένα. Ωστόσο, δύναται να χρησιμοποιηθούν διαφορετικά μοτίβα για εργασίες σε ένα επίπεδο που έχει μεγάλη ποικιλία στις απαιτήσεις, όπως μια εφαρμογή που περιέχει λειτουργίες επιχειρηματικών συναλλαγών (transactions) και αναφορές (reports) (Krause, 2021; Microsoft Patterns & Practices Team, 2009).
- **Αποφυγή επανάληψης της λειτουργικότητα σε μια εφαρμογή**. Θα πρέπει να υπάρχει μόνο ένα στοιχείο που παρέχει μια συγκεκριμένη λειτουργικότητα και αυτή η λειτουργικότητα δεν θα πρέπει να αντιγράφεται σε κανένα άλλο στοιχείο. Αυτό καθιστά τα εξαρτήματα συνεκτικά και διευκολύνει στη βελτιστοποίηση των στοιχείων στην περίπτωση που αλλάξει μια συγκεκριμένη λειτουργία ή λειτουργικότητα. Ο διπλασιασμός της λειτουργικότητας σε μια εφαρμογή μπορεί να δυσκολέψει την εφαρμογή αλλαγών, να μειώσει τη σαφήνεια και να δημιουργήσει πιθανές ασυνέπειες (Visser, 2016).
- **Προτίμηση της σύνθεσης από την κληρονομικότητα**. Όπου είναι δυνατόν, θα πρέπει να προτιμάται η σύνθεση έναντι της κληρονομικότητας κατά την επαναχρησιμοποίηση της λειτουργικότητας, επειδή η κληρονομικότητα αυξάνει την εξάρτηση μεταξύ γονικών και θυγατρικών κλάσεων, περιορίζοντας έτσι την επαναχρησιμοποίηση των θυγατρικών κλάσεων. Η πρακτική αυτή μειώνει επίσης τις ιεραρχίες κληρονομικότητας, οι οποίες μπορεί ενίοτε να γίνουν πολύ δύσκολες στη διαχείρισή τους (Gamma et al., 1995).
- **Καθιέρωση ενός στυλ ανάπτυξης κώδικα που περιλαμβάνει συμβάσεις ονομασίας (naming conventions).** Για την ευκολία του σχεδιασμού και κατ' επέκταση για την πιο εύκολη ανάπτυξη του κώδικα κάθε οργανισμός θα πρέπει να έχει καθιερώσει πρότυπα στυλ κωδικοποίησης και ονομασίας. Στην περίπτωση που αυτά δεν έχουν προβλεφθεί θα πρέπει να θεσπιστούν κοινά πρότυπα. Η πρακτική αυτή παρέχει ένα συνεπές μοντέλο που διευκολύνει τα μέλη της ομάδας να ελέγξουν τον κώδικα των άλλων μελών της ομάδας (code review), γεγονός που οδηγεί σε καλύτερη συντήρηση (maintainability) της εφαρμογής ή του συστήματος (Martin, 2018).
- **Διατήρηση της ποιότητας του συστήματος χρησιμοποιώντας αυτοματοποιημένες τεχνικές ελέγχου κατά την ανάπτυξη.** Χρησιμοποιώντας δοκιμές μονάδων (unit testing) και άλλες αυτοματοποιημένες τεχνικές ανάλυσης ποιότητας (Quality Analysis Techniques), όπως ανάλυση εξάρτησης και ανάλυση στατικού κώδικα, κατά την ανάπτυξη, αυξάνεται η ποιότητα του κώδικα εν γένει. Ο καθορισμός σαφών μετρήσεων συμπεριφοράς και

απόδοσης για τα επιμέρους στοιχεία και τα υποσυστήματα καθώς και η χρησιμοποίηση αυτοματοποιημένων εργαλείων διασφάλισης ποιότητας κατά τη διαδικασία ανάπτυξης (development process) διασφαλίζει ότι οι αποφάσεις τοπικού σχεδιασμού και ανάπτυξης δεν επηρεάζουν αρνητικά τη συνολική ποιότητα του συστήματος (Axelrod, 2018).

- **Λαμβάνοντας υπόψη τη λειτουργικότητα της εφαρμογής.** Προσδιορισμός των μετρήσεων και των λειτουργικών δεδομένων που απαιτούνται από την υποδομή της πληροφορικής για τη διασφάλιση τόσο της αποτελεσματικής ανάπτυξης όσο και τη λειτουργία της εφαρμογής. Ο σχεδιασμός των στοιχείων και των υποσυστημάτων της εφαρμογής με σαφή κατανόηση των επιμέρους λειτουργικών απαιτήσεων διευκολύνει σημαντικά τη συνολική ανάπτυξη και λειτουργία (Microsoft Patterns & Practices Team, 2009).

## 1.3 Επίπεδα Εφαρμογών - Application Layer

- **Διαχωρισμός των τομέων ενδιαφέροντος.** Διαχωρισμός της εφαρμογής σε διακριτά χαρακτηριστικά ώστε να αλληλεπικαλύπτονται σε λειτουργικότητα όσο το δυνατόν λιγότερο. Το κύριο πλεονέκτημα αυτής της προσέγγισης είναι ότι ένα χαρακτηριστικό ή λειτουργικότητα μπορεί να βελτιστοποιηθεί ανεξάρτητα από άλλα χαρακτηριστικά ή λειτουργικότητες. Επιπλέον, εάν ένα χαρακτηριστικό αποτύχει, δεν θα προκαλέσει την αποτυχία και άλλων λειτουργιών ώστε να μπορούν να εκτελεστούν ανεξάρτητα το ένα από το άλλο. Αυτή η προσέγγιση βοηθά επίσης στο να γίνει η εφαρμογή ευκολότερη στην κατανόηση και στο σχεδιασμό, και διευκολύνει στη διαχείριση πολύπλοκων αλληλοεξαρτώμενων συστημάτων (Martin, 2018; Sommervillle 2016).
- **Σαφής καθορισμός της δι-επικοινωνίας των επίπεδων.** Επιτρέποντας σε κάθε επίπεδο μιας εφαρμογής να επικοινωνεί ή να έχει εξαρτήσεις από όλα τα άλλα επίπεδα, δημιουργείται μια λύση που είναι δύσκολη στην κατανόηση και στη διαχείριση. Θα πρέπει να ληφθούν σαφείς αποφάσεις σχετικά με τις εξαρτήσεις των επιπέδων καθώς και τη ροή των δεδομένων μεταξύ τους (Black, 1996).
- **Εφαρμογή της αφαιρετικότητας για την εφαρμογή της χαλαρής σύζευξης μεταξύ των στρωμάτων.** Αυτό μπορεί να επιτευχθεί είτε ορίζοντας στοιχεία διεπαφής, όπως μια πρόσοψη (façade), με γνωστές εισόδους και εξόδους που μετατρέπουν τα αιτήματα (requests) σε μια μορφή κατανοητή από τα στοιχεία εντός του επιπέδου, είτε με τη χρήση διαφόρων τύπων διεπαφών ή αφηρημένων βασικών κλάσεων για τον καθορισμό ενός κοινού συμβολαίου διαδραστικότητας (Microsoft Patterns & Practices Team, 2009).
- **Ανάμειξη μόνο ίδιων τύπων εξαρτημάτων σε κάθε λογικό επίπεδο (logical layer).** Αρχικά θα πρέπει να εντοπιστούν οι διαφορετικές περιοχές ενδιαφέροντος και να ομαδοποιηθούν σε λογικά επίπεδα. Για παράδειγμα, το επίπεδο διεπαφής χρήστη (UI) δεν θα πρέπει να περιέχει στοιχεία επιχειρησιακής επεξεργασίας (business processing), αλλά θα πρέπει να περιέχει στοιχεία που χρησιμοποιούνται για τον χειρισμό των εισροών χρήστη (input data)

και την επεξεργασία αιτημάτων χρήστη (user requests) (Martin, 2018; Microsoft Patterns & Practices Team, 2009; Sommervillle 2016).

- **Διατήρηση της μορφής δεδομένων σε ένα επίπεδο ή σε ένα στοιχείο.** Η ανάμειξη διαφόρων μορφών δεδομένων θα κάνει την εφαρμογή πιο δύσκολη στην υλοποίηση (implement), επέκταση (extend) και συντήρηση (maintenance). Κάθε φορά που είναι αναγκαία η μετατροπή των δεδομένων από μια μορφή σε άλλη, απαιτείται η εφαρμογή κώδικα μετάφρασης για την εκτέλεση της λειτουργίας και κατά συνέπεια η επιβάρυνση με έμμεσα έξοδα επεξεργασίας (Loshin, 2010; Microsoft Patterns & Practices Team, 2009).

## 1.4 Στοιχεία, Ενότητες και Λειτουργίες

- **Ένα στοιχείο ή ένα αντικείμενο δεν πρέπει να βασίζεται σε εσωτερικές λεπτομέρειες άλλων στοιχείων ή αντικειμένων.** Κάθε στοιχείο ή αντικείμενο πρέπει να καλεί μια μέθοδο ενός άλλου αντικειμένου ή στοιχείου και αυτή η μέθοδος θα πρέπει να έχει πληροφορίες σχετικά με τον τρόπο επεξεργασίας του αιτήματος και, εφόσον απαιτείται, τον τρόπο δρομολόγησης του σε κατάλληλα υποστοιχεία ή άλλα στοιχεία. Με αυτή την πρακτική μια εφαρμογή καθίσταται πιο εύκολα διατηρήσιμη (maintainable) και προσαρμόσιμη (adaptable).
- **Η λειτουργικότητα ενός εξαρτήματος-στοιχείου δεν θα πρέπει να υπερφορτώνεται.** Για παράδειγμα, ένα στοιχείο επεξεργασίας διεπαφής χρήστη δεν θα πρέπει να περιέχει κώδικα πρόσβασης δεδομένων. Τα υπερφορτωμένα εξαρτήματα έχουν συχνά πολλές λειτουργίες και ιδιότητες παρέχοντας επιχειρηματική λειτουργικότητα σε συνδυασμό με λειτουργίες εγκάρσιας τομής, όπως η καταγραφή (logging) και ο χειρισμός εξαιρέσεων (error handling). Το αποτέλεσμα είναι ένα σχέδιο πολύ επιρρεπές σε σφάλματα (error prone) και πολύ δύσκολο να διατηρηθεί.
- **Κατανόηση της επικοινωνίας μεταξύ των εξαρτημάτων-στοιχείων.** Αυτό απαιτεί την κατανόηση των σεναρίων τοποθέτησης του κώδικα της εφαρμογής στην παραγωγή (deployment process). Θα πρέπει να καθοριστεί εάν όλα τα στοιχεία θα εκτελούνται εντός της ίδιας διεργασίας ή εάν θα πρέπει να υποστηρίζεται η επικοινωνία μεταξύ διαφόρων φυσικών ορίων ή διαφορετικών ορίων διεργασίας (Microsoft Patterns & Practices Team, 2009).
- **Διαχωρισμός του κώδικα διασταυρούμενων συμφερόντων (crosscutting code) από την επιχειρηματική λογική της εφαρμογής.** Ο κώδικας διασταυρούμενων συμφερόντων αναφέρεται σε κώδικα που σχετίζεται με την ασφάλεια, τις επικοινωνίες ή τη λειτουργική διαχείριση, όπως η καταγραφή (logging) και η ενορχήστρωση (instrumentation). Η ανάμειξη του κώδικα που υλοποιεί τις παραπάνω λειτουργίες με την επιχειρηματική λογική μπορεί να οδηγήσει σε ένα σχέδιο που είναι δύσκολο να επεκταθεί και να διατηρηθεί. Θα πρέπει να εξετάζεται το ενδεχόμενο χρήσης πλαισίων (frameworks) και τεχνικών, όπως ο

προγραμματισμός προσανατολισμένος σε όψεις, (AOP: Aspect-Oriented Programming) που μπορούν να βοηθήσουν στη διαχείριση εγκάρσιων θεμάτων (Ingeno, 2018).

- **Καθορισμός σαφών συμβολαίων για τα εξαρτήματα.** Τα στοιχεία (components), οι ενότητες (modules) και οι λειτουργίες (functions) θα πρέπει να ορίζουν μια προδιαγραφή σύμβασης ή διεπαφής που να περιγράφει με σαφήνεια τη χρήση και τη συμπεριφορά τους. Το συμβόλαιο πρέπει να περιγράφει πως άλλα στοιχεία μπορούν να έχουν πρόσβαση στην εσωτερική λειτουργικότητα του στοιχείου, της μονάδας ή της λειτουργίας, καθώς και τη συμπεριφορά αυτής της λειτουργικότητας όσον αφορά τις προκαταρκτικές συνθήκες (pre-conditions), τις μετασυνθήκες (post-conditions), τις παρενέργειες (side effects), τις εξαιρέσεις (exceptions) και τα χαρακτηριστικά απόδοσης (performance characteristics) (Liu et al., 2004; Microsoft Patterns & Practices Team, 2009).

## 1.5 Βασικά ζητήματα Σχεδιασμού - Key Design Considerations

Παρακάτω περιγράφονται οι σημαντικές αποφάσεις που πρέπει να λαμβάνονται υπόψη επαναληπτικά κατά τον αρχιτεκτονικό σχεδιασμό και οι οποίες βοηθούν στη διασφάλιση ότι ο αρχιτέκτονας-σχεδιαστής δεν έχει ξεφύγει από το βασικό πλαίσιο κατά την έναρξη και την ανάπτυξη του αρχιτεκτονικού σχεδίου. Οι κύριες αποφάσεις, που περιγράφονται συνοπτικά στις ακόλουθες ενότητες, είναι (Ingeno, 2018; Martin, 2018; Sommerville, 2016):

- Προσδιορισμός του τύπου της εφαρμογής
- Καθορισμός της στρατηγικής εγκατάστασης στην παραγωγή
- Προσδιορισμός των κατάλληλων τεχνολογιών
- Προσδιορισμός των ποιοτικών χαρακτηριστικών
- Προσδιορισμός των διασταυρούμενων συμφερόντων

### 1.5.1 Προσδιορισμός του τύπου της εφαρμογής

Η επιλογή του κατάλληλου τύπου εφαρμογής είναι το βασικό μέρος της διαδικασίας σχεδιασμού μιας εφαρμογής. Η επιλογή της αρχιτεκτονικής διέπεται από τις συγκεκριμένες απαιτήσεις και τους περιορισμούς στις υποδομές. Πολλές εφαρμογές πρέπει να υποστηρίζουν διαφορετικούς τύπους πελατών και μπορεί να κάνουν χρήση περισσότερων του ενός βασικών αρχέτυπων. Οι συνηθέστεροι βασικοί τύποι εφαρμογών είναι οι ακόλουθοι (Ingeno, 2018; Martin, 2018; Sommerville, 2016):

- Εφαρμογές σχεδιασμένες για φορητές συσκευές.
- Πλούσιες εφαρμογές πελάτη (rich client applications) σχεδιασμένες να εκτελούνται κυρίως στον υπολογιστή του πελάτη.

- Πλούσιες εφαρμογές Διαδικτύου, οι οποίες υποστηρίζουν σενάρια εμπλουτισμένων διεπαφών χρήστη και πολυμέσων.
- Εφαρμογές υπηρεσιών που έχουν σχεδιαστεί για να υποστηρίζουν την επικοινωνία μεταξύ χαλαρά συζευγμένων στοιχείων.
- Εφαρμογές ιστού σχεδιασμένες να εκτελούνται κυρίως στον διακομιστή με πλήρως συνδεδεμένα σενάρια.

Πιο εξειδικευμένοι τύποι εφαρμογών είναι οι ακόλουθοι:

- Εφαρμογές και υπηρεσίες φιλοξενούμενες και βασισμένες στο cloud.
- Εφαρμογές Office Business (OBA) που ενσωματώνουν τεχνολογίες Microsoft Office και διακομιστή Microsoft.
- Εφαρμογές SharePoint Line of Business (LOB) που παρέχουν πρόσβαση σε στυλ πύλης (portal) για πρόσβαση σε πληροφορίες και λειτουργίες των επιχειρήσεων.

Φυσικά η λίστα με τα αρχέτυπα εφαρμογών είναι ανεξάντλητη και όσο πιο έμπειρος είναι ένας αρχιτέκτονας συστημάτων τόσο περισσότερους τύπους γνωρίζει (Microsoft Patterns & Practices Team, 2009).

### 1.5.2 Καθορισμός της στρατηγικής εγκατάστασης στην παραγωγή - Deployment Strategy

Η εφαρμογή μπορεί να εγκατασταθεί σε διάφορα περιβάλλοντα, το καθένα με το δικό του συγκεκριμένο σύνολο περιορισμών, όπως φυσικό διαχωρισμό στοιχείων σε διαφορετικούς διακομιστές (physical separation of components across different servers), περιορισμό στα πρωτόκολλα δικτύωσης (limitation on networking protocols), διαμορφώσεις τείχους προστασίας και δρομολογητή (firewall and router configurations) και άλλα. Υπάρχουν πολλά κοινά πρότυπα εγκατάστασης, τα οποία περιγράφουν τα οφέλη και τις εκτιμήσεις για μια σειρά κατανεμημένων και μη σεναρίων. Θα πρέπει να ζυγιστούν οι απαιτήσεις της εφαρμογής σε σχέση με τα μοτίβα που μπορούν να υποστηριχθούν από το υλικό (hardware) καθώς και τους περιορισμούς που ασκεί το περιβάλλον στις επιλογές εγκατάστασης. Όλοι οι παραπάνω παράγοντες θα επηρεάσουν τον αρχιτεκτονικό σχεδιασμό (Ingeno, 2018; Martin, 2018; Microsoft Patterns & Practices Team, 2009).

### 1.5.3 Καθορισμός των κατάλληλων τεχνολογιών

Οι βασικοί παράγοντες που επηρεάζουν την επιλογή τεχνολογιών των εφαρμογών και των συστημάτων είναι ο τύπος της εφαρμογής-συστήματος, οι προτιμώμενες επιλογές για την τοπολογία εγκατάστασης και το αρχιτεκτονικό στυλ που έχει επιλεγεί. Η επιλογή των τεχνολογιών επηρεάζεται επίσης από τις πολιτικές του οργανισμού, τους περιορισμούς στις υποδομές, τους διαθέσιμους πόρους καθώς και τις δεξιότητες του προσωπικού (Ingeno, 2018; Martin, 2018; Microsoft Patterns & Practices Team, 2009).

### 1.5.4 Καθορισμός των ποιοτικών χαρακτηριστικών

Τα χαρακτηριστικά ποιότητας, όπως η ασφάλεια, η απόδοση και η χρηστικότητα, μπορούν να χρησιμοποιηθούν προκειμένου να εστιάσει ο αρχιτέκτονας τη σκέψη του στα κρίσιμα προβλήματα που καλείται να λύσει το αρχιτεκτονικό σχέδιο. Ανάλογα με τις απαιτήσεις, ίσως χρειαστεί να ληφθούν υπόψη όλα τα παραπάνω χαρακτηριστικά ποιότητας ή ενδεχομένως να χρειαστεί να εξεταστεί μόνο ένα υποσύνολο. Για παράδειγμα, κάθε σχέδιο εφαρμογής πρέπει να λαμβάνει υπόψη την ασφάλεια και την απόδοση, αλλά δεν χρειάζεται κάθε σχέδιο να λαμβάνει υπόψη τη διαλειτουργικότητα ή την επεκτασιμότητα. Η κατανόηση των απαιτήσεων και των σεναρίων εγκατάστασης ανάπτυξης, θα καθορίσουν και τα σημαντικά χαρακτηριστικά ποιότητας. Σε κάθε περίπτωση θα πρέπει να ληφθεί υπόψη ότι τα χαρακτηριστικά ποιότητας ενδέχεται να έρχονται σε σύγκρουση. Για παράδειγμα, η ασφάλεια απαιτεί συχνά μια αντιστάθμιση με την απόδοση ή τη χρηστικότητα (Ingeno, 2018; Martin, 2018; Microsoft Patterns & Practices Team, 2009).

Οι ακόλουθες οδηγίες θα πρέπει να ληφθούν υπόψη κατά το σχεδιασμό με εξέταση των ποιοτικών χαρακτηριστικών:

- Τα χαρακτηριστικά ποιότητας είναι ιδιότητες συστήματος που είναι ξεχωριστές από τη λειτουργικότητα του συστήματος.
- Από τεχνική άποψη, η εφαρμογή χαρακτηριστικών ποιότητας μπορεί να διαφοροποιήσει ένα καλό από ένα κακό σύστημα.
- Υπάρχουν δύο τύποι χαρακτηριστικών ποιότητας: αυτά που μετρώνται κατά το χρόνο εκτέλεσης και αυτά που μπορούν να εκτιμηθούν μόνο μέσω επιθεώρησης.
- Ανάλυση των συμβιβασμών - αντισταθμισμάτων μεταξύ των χαρακτηριστικών ποιότητας.

Οι κάτωθι ερωτήσεις πρέπει να τίθενται όταν εξετάζονται τα χαρακτηριστικά ποιότητας (Microsoft Patterns & Practices Team, 2009):

- Ποια είναι τα βασικά χαρακτηριστικά ποιότητας που απαιτούνται από την εφαρμογή; Αυτά θα πρέπει να προσδιοριστούν ως μέρος της διαδικασίας σχεδιασμού.
- Ποιες είναι οι βασικές απαιτήσεις για την αντιμετώπιση αυτών των χαρακτηριστικών; Είναι πράγματι μετρήσιμα;
- Ποια είναι τα κριτήρια αποδοχής που θα υποδεικνύουν ότι πληρούνται οι προϋποθέσεις;

### 1.5.5 Προσδιορισμός των διασταυρούμενων συμφερόντων

Τα διασταυρούμενα συμφέροντα αντιπροσωπεύουν βασικούς τομείς του σχεδιασμού που δεν σχετίζονται με ένα συγκεκριμένο επίπεδο της εφαρμογής. Για παράδειγμα, θα πρέπει να εξεταστεί το ενδεχόμενο εφαρμογής κεντρικών ή κοινών λύσεων για τα ακόλουθα (Ingeno, 2018; Microsoft Patterns & Practices Team, 2009):

- Ένας μηχανισμός καταγραφής (logging mechanism) που επιτρέπει σε κάθε επίπεδο να συνδέεται σε ένα κοινό κατάστημα ή να καταγράφει ξεχωριστά καταστήματα με τέτοιο τρόπο ώστε τα αποτελέσματα να μπορούν να συσχετιστούν στη συνέχεια.
- Ένας μηχανισμός για έλεγχο ταυτότητας (authentication) και εξουσιοδότηση (authorization) που μεταβιβάζει ταυτότητες σε πολλαπλά επίπεδα για να επιτρέπεται η παραχώρηση πρόσβασης σε πόρους.
- Ένα πλαίσιο διαχείρισης εξαιρέσεων (exception management framework) που θα λειτουργεί τόσο μέσα σε κάθε επίπεδο, όσο και κατά μήκος όλων των επιπέδων καθώς οι εξαιρέσεις διαδίδονται μέσα στα όρια του συστήματος.
- Μια προσέγγιση επικοινωνίας (communication approach) που θα χρησιμοποιηθεί για την επικοινωνία μεταξύ των επιπέδων.
- Μια κοινή υποδομή προσωρινής αποθήκευσης (caching infrastructure) για την αποθήκευση προσωρινών δεδομένων τόσο στο επίπεδο παρουσίασης (presentation layer), όσο και στο επίπεδο επιχειρηματικής λογικής (business layer) αλλά και στο επίπεδο πρόσβασης δεδομένων (data access layer).

Η ακόλουθη λίστα περιγράφει ορισμένα από τα βασικά διασταυρούμενα συμφέροντα που πρέπει να ληφθούν υπόψη κατά την αρχιτεκτονική των εφαρμογών (Ingeno, 2018; Microsoft Patterns & Practices Team, 2009):

- **Αυθεντικοποίηση (Authentication).** Καθορισμός του τρόπου ελέγχου ταυτότητας των χρηστών και μεταβίβασης ελεγμένων ταυτοτήτων στα διάφορα επίπεδα.
- **Εξουσιοδότηση (Authorization).** Διασφάλιση της κατάλληλης εξουσιοδότησης με την κατάλληλη ευαισθησία εντός κάθε επιπέδου και κατά μήκος των ορίων εμπιστοσύνης.
- **Καταγραφή και ενορχήστρωση (Logging and instrumentation).** Οργάνωση όλων των κρίσιμων συμβάντων για την καταγραφή επαρκών λεπτομερειών τους, χωρίς να συμπεριληφθούν ευαίσθητες πληροφορίες.
- **Διαχείριση εξαιρέσεων (Exception management).** Καθορισμός και διαχείριση των εξαιρέσεων σε λειτουργικά, λογικά και φυσικά όρια. Αποφυγή της αποκάλυψης ευαίσθητων πληροφοριών στους τελικούς χρήστες.
- **Επικοινωνία (Communication).** Επιλογή κατάλληλων πρωτοκόλλων, ελαχιστοποίηση των κλήσεων μέσω του δικτύου και προστασία των ευαίσθητων δεδομένων που περνούν μέσα από το δίκτυο.
- **Προσωρινή αποθήκευση (Caching).** Προσδιορισμός της πληροφορίας που θα πρέπει να αποθηκευτεί προσωρινά καθώς και του μέσου αποθήκευσης αυτής, προκειμένου να βελτιωθεί η απόδοση και η ανταπόκριση της εφαρμογής.

# Κεφ.2. Αρχιτεκτονικές Μοντέρνων Εφαρμογών (Architecting Modern Applications)

Κάθε εφαρμογή για να είναι κατανοητή στους εμπλεκόμενους χρήστες (σχεδιαστές, προγραμματιστές, συντηρητές, διαχειριστές) απαιτείται να ακολουθεί μια συγκεκριμένη αρχιτεκτονική.

Σύμφωνα με τον κατά Bass (2013) ορισμό: "*Η αρχιτεκτονική εφαρμογών είναι η δομή ή οι δομές μιας εφαρμογής που περιλαμβάνουν τις προγραμματιστικές οντότητες (software components) που τη συνθέτουν, τις εξωτερικές ιδιότητες των οντοτήτων αυτών, καθώς και τη μεταξύ τους διασύνδεση. Η αρχιτεκτονική εφαρμογών περιλαμβάνει τις αποφάσεις για τη δομή ή τις δομές της εφαρμογής, καθώς και τις αλληλεπιδράσεις μεταξύ των δομών αυτών. Η αρχιτεκτονική επομένως καθορίζει την ανάπτυξη, την υποστήριξη και τη συντήρηση της εφαρμογής*".

Οι μοντέρνες εφαρμογές που αξιοποιούν το υπολογιστικό νέφος έχουν διαφορετικές απαιτήσεις και περιορισμούς από εκείνες που αναπτύσσονταν στο παρελθόν. Καινούργια αρχιτεκτονικά μοντέλα και πρότυπα έχουν υιοθετηθεί για να υποστηρίξουν τις νέες ανάγκες (Ingeno, 2018).

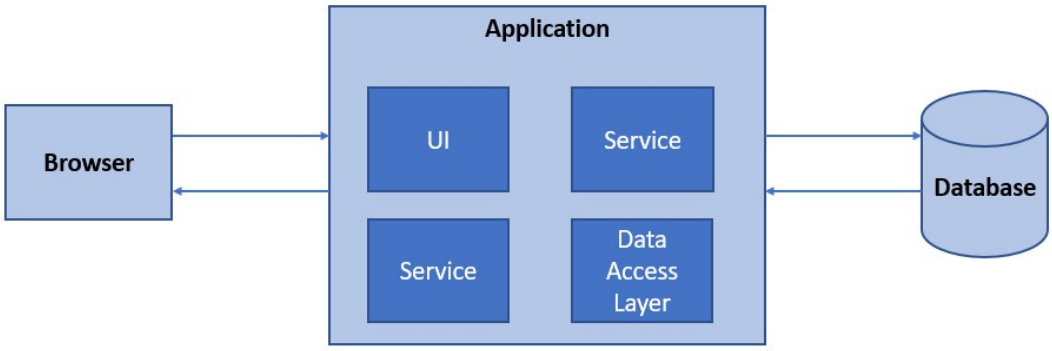
Παρακάτω αναλύονται ορισμένα από τα αρχιτεκτονικά μοτίβα του παρελθόντος, ώστε να γίνει κατανοητή η μετάβαση στις νέες αρχιτεκτονικές ανάπτυξης λογισμικού. Επίσης, μελετώνται οι νέες αρχιτεκτονικές ώστε να γίνει κατανοητή η ανάγκη μετάβασης σε αυτές.

Πιο συγκεκριμένα αναλύονται τα κάτωθι αρχιτεκτονικά πρότυπα:

- Μονολιθική αρχιτεκτονική (Monolithic architecture)
- Υπηρεσιοστρεφής αρχιτεκτονική (Service Oriented Architecture - SOA)
- Αρχιτεκτονική μικροϋπηρεσιών (Microservices architecture - MSA)
- Αρχιτεκτονική χωρίς διακομιστή (Serverless architecture)
- Αρχιτεκτονική υπολογιστικού νέφους (Cloud-native architecture)

## 2.1 Μονολιθική Αρχιτεκτονική (Monolithic architecture)

Το μονολιθικό λογισμικό είναι σχεδιασμένο να είναι αυτόνομο, καθώς στην μονολιθική αρχιτεκτονική η υπό ανάπτυξη εφαρμογή σχεδιάζεται ως μια ενιαία μονάδα λογισμικού. Τα στοιχεία του προγράμματος είναι διασυνδεδεμένα και αλληλοεξαρτώμενα σε μια αυστηρά συζευγμένη δομή, όπου κάθε στοιχείο και κομμάτι λογισμικού πρέπει να είναι παρόν ώστε να εκτελεστεί ο κώδικας (Havey, 2008).




Εικ. 1. Μονολιθική εφαρμογή

Τα διάφορα στοιχεία που συνιστούν την εφαρμογή, όπως η διεπαφή χρηστών (user interface), η επιχειρηματική λογική (business logic), η αυθεντικοποίηση (authorization), η καταγραφή συμβάντων (logging), η πρόσβαση σε βάσεις δεδομένων (database access) είναι όλα κομμάτια της ενιαίας, σφιχτά δεμένης εφαρμογής (Ingeno, 2018).

### 2.1.1 Πλεονεκτήματα της μονολιθικής αρχιτεκτονικής

Παρά τα όποια φανερά μειονεκτήματα χρήσης αυτού του τύπου αρχιτεκτονικής, προτείνεται η χρήση της, στην περίπτωση που η ανάπτυξη αφορά σε μια σχετικά μικρή υλοποίηση (Mäkitalo, 2018). Η μείωση της συσχέτισης διαφορετικών μηχανημάτων που επιτυγχάνεται με την εκτέλεση του κώδικα σε ένα μόνο μηχάνημα, οδηγεί σε καλύτερη απόδοση του συστήματος. Ωστόσο, η απόδοση μειώνεται καθώς η εφαρμογή μεγαλώνει και προστίθενται νέα χαρακτηριστικά σε αυτή (Saransig et al., 2019).

Επιπλέον, τέτοιου τύπου εφαρμογές είναι πιο εύκολο να αναπτυχθούν, να τεσταριστούν και να εγκατασταθούν (deploy) στην παραγωγή μιας και ο κώδικας βάσης (codebase) είναι μικρός και εύκολα διαχειρίσιμος (Bajaj et al., 2021).

Ειδικά όσον αφορά τον έλεγχο της εφαρμογής (test) η απλοποίηση έγκειται στο γεγονός ότι λιγότερα ξεχωριστά - εξωγενή στοιχεία θα πρέπει να ληφθούν υπόψη και να τεσταριστούν (Ingeno, 2018). Συνεπώς ο έλεγχος ενοποίησης (integration testing) είναι απλούστερος στην διεξαγωγή του για τις μονολιθικές εφαρμογές.

### 2.1.2 Μειονεκτήματα της μονολιθικής αρχιτεκτονικής

Παρόλο που το μοντέλο της μονολιθικής αρχιτεκτονικής μπορεί κάλλιστα να υιοθετηθεί για κάποιους τύπους εφαρμογών, όπως αναφέρθηκε παραπάνω συνίσταται κυρίως σε περιπτώσεις ανάπτυξης μικρών εφαρμογών, με μικρό κώδικα βάσης (codebase) και σχετικά μικρή

επιχειρηματική λογική (business logic). Καθώς οι εφαρμογές μεγαλώνουν σε όγκο και πολυπλοκότητα εντοπίζεται πληθώρα μειονεκτημάτων (Ingeno, 2018).

Εξαιτίας του γεγονότος, ότι όλη η επιχειρηματική λογική της εφαρμογής βρίσκεται σε ένα κοινό κώδικα βάσης, είναι πολύ δύσκολο για τα νέα μέλη μιας ομάδας ανάπτυξης να γίνουν γρήγορα παραγωγικά, αφού θα πρέπει να κατανοήσουν τις απαιτήσεις της εφαρμογής στο σύνολό της πριν αρχίσουν να γράφουν κώδικα για κάποια νέα υπο-ανάπτυξη απαίτηση (Ingeno, 2018; Mäkitalo, 2018).

Οι μεγάλες και σύνθετες εφαρμογές απαιτούν πολλές ομάδες ανάπτυξης και επιμερισμό ευθυνών. Σε μια μονολιθική εφαρμογή η ανάπτυξη κώδικα είναι δυνατό να επηρεάζει την εργασία που εκτελείται από άλλη ομάδα, καθώς οι προσθήκες και οι αλλαγές γίνονται πάνω σε κοινό κώδικα (Ingeno, 2018; Mella et al., 2019). Καθώς οι εφαρμογές μεγαλώνουν και αναπτύσσονται είναι δύσκολο να συντηρηθούν αφού η αλλαγή σε κάποιο σημείο της εφαρμογής επηρεάζει άλλα κομμάτια της (Bajaj et al., 2021; Mäkitalo, 2018; Pradhan et al., 2020).

Επιπλέον, τόσο η ανάπτυξη του κώδικα όσο και η συντήρησή του σε ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE: Integrated Development Environment) μπορεί να προκαλέσει εκνευρισμό στους προγραμματιστές εξαιτίας των καθυστερήσεων κατά την έναρξη της εφαρμογής, το φόρτωμα του κώδικα ακόμη και της εκτέλεσής του (Ingeno, 2018).

Οι μονολιθικές εφαρμογές απαιτούν την δέσμευση σε μια τεχνολογία και σε μια γλώσσα ανάπτυξης λογισμικού (Pradhan et al., 2020). Ο ενιαία συντηρούμενος κώδικας δεν μπορεί να υποστηρίξει την πολυγλωσσία (polyglot programming) και τα πλεονεκτήματα που αυτή επιφέρει, ούτε την χρήση διαφορετικών τεχνολογιών ανάλογα με την εκάστοτε ανάγκη (Ingeno, 2018).

Η επεκτασιμότητα (scalability) των μονολιθικών εφαρμογών δύναται να είναι μόνο οριζόντια με την αναπαραγωγή περισσοτέρων ίδιων μονάδων κώδικα (instances) να τρέχουν πίσω από έναν δρομολογητή φορτίου (load-balancer) (Al-Debagy, 2018; Gos et al., 2020; Mella et al., 2019; Pradhan et al., 2020).

Η αξιοπιστία (reliability) τέτοιου τύπου εφαρμογών είναι μειωμένη καθώς ένα σφάλμα/δυσλειτουργία επιφέρει προβλήματα σε όλη την εφαρμογή (Pradhan et al., 2020). Τέλος, η διαθεσιμότητα (availability) είναι επίσης χαμηλότερη σε σχέση με τις εφαρμογές που αναπτύσσονται σύμφωνα με άλλες αρχιτεκτονικές, αφού κάθε εγκατάσταση κώδικα στην παραγωγή απαιτεί το redeployment όλης της εφαρμογής, οδηγώντας σε καθολικό, έστω και για μικρό χρονικό διάστημα, χρόνο μη λειτουργίας της εφαρμογής (downtime) (Al-Debagy, 2018).

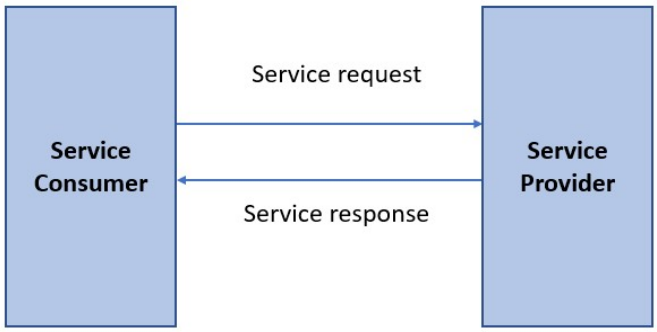
Τέλος, το γεγονός ότι κάθε αλλαγή στον κώδικα απαιτεί την εγκατάσταση (deployment) στην παραγωγή νέας έκδοσης λογισμικού για το σύνολο της εφαρμογής, μεγιστοποιείται η απόκλιση μεταξύ υλοποίησης (development) και παραγωγικής λειτουργίας (production), θέτοντας εμπόδια στην συνεχή ανάπτυξη (continuous deployment) και στην ευκινησία (agility) τόσο του τμήματος ανάπτυξης λογισμικού όσο και του ίδιου του οργανισμού εν γένει (Ingeno, 2018; Pradhan et al., 2020).

## 2.2 Υπηρεσιοστρεφής Αρχιτεκτονική (Service Oriented Architecture - SOA)

Η υπηρεσιοστρεφής αρχιτεκτονική συνίσταται σε ένα αρχιτεκτονικό μοντέλο ανάπτυξης λογισμικού, συστημάτων και εφαρμογών με την δημιουργία χαλαρά διασυνδεδεμένων υπηρεσιών (services) οι οποίες διαλειτουργούν μεταξύ τους (interoperable) (Esfandyari et al., 2013; Sheikh et al., 2011), έχουν καλά καθορισμένες διεπαφές (interfaces) προκειμένου να επικοινωνήσει κάποιο άλλο κομμάτι λογισμικού με αυτές και ακολουθούν ένα προτυποποιημένο πρωτόκολλο επικοινωνίας (Sheikh et al., 2011; Alanazi et al.,2019).

Με την συνεργασία των υπηρεσιών αυτών μπορεί να επιτευχθεί η αυτοματοποίηση των επιχειρησιακών διαδικασιών (Beydoun et al., 2013). Μια υπηρεσία είναι κομμάτι μιας εφαρμογής ή ενός συστήματος το οποίο επιτελεί ένα πολύ συγκεκριμένο ρόλο και ολοκληρώνει επίσης ένα πολύ συγκεκριμένο έργο, παρέχοντας λειτουργικότητα σε άλλα τμήματα της ίδιας ή άλλων εφαρμογών (Ingeno, 2018).

Κάθε τμήμα του λογισμικού επικεντρώνεται στην επίλυση ενός ξεχωριστού ζητήματος επιτυγχάνοντας το “Διαχωρισμό Ενδιαφερόντων” (SoC: Separation of Concerns)· τον επιμερισμό δηλαδή, ενός συστήματος σε αυτόνομα μέρη, όσον αφορά τη λειτουργικότητα που επιτελεί το καθένα από αυτά (Ingeno, 2018). Τα τμήματα αυτά είναι επαναχρησιμοποιήσιμα (Sheikh et al., 2011) και διανεμημένα κατά τέτοιο τρόπο ώστε να βελτιώνεται η ποιότητα όλου του λογισμικού στο σύνολό του, μιας και αυτό αποτελείται από πιο εύκολα διαχειρίσιμες και συντηρήσιμες μονάδες (Beydoun et al., 2013; Esfandyari et al., 2013; Alanazi et al.,2019). Κάθε υπηρεσία αναπτύσσει ένα πολύ συγκεκριμένο κομμάτι υλοποίησης και λογικής. Αυτό το κομμάτι είναι υπεύθυνο να επιτελεί ένα έργο, μια διαδικασία ή μια υπο-διαδικασία (Esfandyari et al., 2013). Οι υπηρεσίες μπορεί να διαφέρουν σε μέγεθος και δύναται μια υπηρεσία να απαιτεί την σύνθεση πολλών μικρότερων υπηρεσιών προκειμένου να ολοκληρώσει το έργο της (Al-Kofahi, 2011).




Εικ. 2. Βασική λειτουργικότητα της SOA αρχιτεκτονικής

Αυτό που καθιστά την υπηρεσιοστρεφή αρχιτεκτονική διαφορετική από άλλες αρχιτεκτονικές διανεμημένων υπηρεσιών είναι ο τρόπος σχεδιασμού των κεντρικών/βασικών στοιχείων της (Ingeno, 2018; Sheikh et al., 2011).

### 2.2.1 Πλεονεκτήματα της αρχιτεκτονικής SOA

Υπάρχει πλήθος πλεονεκτημάτων που απορρέουν από την χρήση της SOA αρχιτεκτονικής:

- Βελτιστοποίηση της ευθυγράμμισης της τεχνολογίας και των επιχειρηματικών απαιτήσεων
- Ενίσχυση της ενοποίησης σε έναν οργανισμό
- Δυνατότητα επιλογής διαφορετικών παρόχων
- Αύξηση της εσωστρεφούς διαλειτουργικότητας
- Υποστήριξη των agile μεθοδολογιών ανάπτυξης λογισμικού

#### 2.2.1.1 Βελτιστοποίηση της ευθυγράμμισης της τεχνολογίας και των επιχειρηματικών απαιτήσεων

Η αλλαγή είναι μια σταθερά που θα πρέπει να αντιμετωπίσουν όλοι οι οργανισμοί και ενυπάρχει σε κάθε επιχείρηση εξαιτίας πολλών και διαφορετικών παραγόντων, όπως οι δυνάμεις της αγοράς, οι τεχνολογικές αλλαγές, οι νέες επιχειρηματικές ευκαιρίες, οι συγχωνεύσεις εταιριών κ.λπ. (Sheikh et al., 2011). Ανεξαρτήτως της πρωτογενούς αιτίας των αλλαγών, η SOA αρχιτεκτονική παρέχει στον οργανισμό την ευελιξία για την άμεση εφαρμογή και υιοθέτηση των νέων κανόνων μέσω της χρήσης υπηρεσιών που είναι χαλαρά διασυνδεδεμένες και αφαιρετικές. Όταν οι συνθήκες επιτάσσουν την αλλαγή, η εφαρμογή τους μπορεί εύκολα και γρήγορα να επιτευχθεί ώστε οι επιχειρηματικές απαιτήσεις να είναι ευθυγραμμισμένες με την τεχνολογία (Woods et al., 2006).

#### 2.2.1.2 Ενίσχυση της ενοποίησης σε έναν οργανισμό

Η ενοποίηση (federation) σε έναν οργανισμό επιτυγχάνεται όταν οι εφαρμογές λογισμικού και οι πόροι διαλειτουργούν με άριστο τρόπο διατηρώντας παράλληλα την αυτονομία τους (Beydoun et al., 2013). Η ενοποίηση δίνει σε έναν οργανισμό την ελευθερία να μην απαιτείται η αντικατάσταση όλων των εμπλεκόμενων μερών σε υπάρχοντα συστήματα όταν υπάρξει η ανάγκη για μια αλλαγή. Με την υιοθέτηση ενός κοινού, ανοιχτού, προδιαγεγραμμένου πλαισίου (framework), τα legacy και τα non-legacy συστήματα μπορούν να λειτουργήσουν καλά μεταξύ τους·¶ η δυνατότητα επιλογής αντικατάστασης ορισμένων μόνο συστημάτων ή υπο-συστημάτων βοηθάει στην σταδιακή μετάπτωση μεγάλων εφαρμογών (Sheikh et al., 2011).

#### 2.2.1.3 Δυνατότητα επιλογής διαφορετικών παρόχων

Ένα ακόμη πλεονέκτημα αυτής της αρχιτεκτονικής είναι η δυνατότητα επιλογής πολλών και διαφορετικών παρόχων. Παρόλο που η δυνατότητα αυτή δεν είναι αυτοσκοπός της υπηρεσιοστρεφούς αρχιτεκτονικής, το κέρδος με την χρήση πολλών και διαφορετικών παρόχων εσωτερικά σε μια εφαρμογή-σύστημα είναι μεγάλο, αφού επιτυγχάνεται η καλύτερη δυνατή λύση (best-of-breed solution) (Ingeno, 2018).

2.2.1.4 Αύξηση της εσωστρεφής διαλειτουργικότητας

Διαφορετικές υπηρεσίες μπορούν να συνεργαστούν για να συνθέσουν μια μεγαλύτερη προκειμένου να αυτοματοποιηθεί μια επιχειρηματική διαδικασία (Beydoun et al., 2013). Αυτό επιτρέπει την επαναχρησιμοποίηση των δεδομένων (Nils et al., 2010) και της επιχειρηματικής λογικής σε έναν οργανισμό και προσφέρει καλύτερα επίπεδα συνειδητοποίησης όσων αφορά τους στρατηγικούς στόχους και τις επιχειρησιακές ανάγκες.

2.2.1.5 Υποστήριξη των ευέλικτων μεθοδολογιών (agile) ανάπτυξης λογισμικού

Το γεγονός ότι πολύπλοκα συστήματα λογισμικού μπορούν να κατατμηθούν σε υπηρεσίες με μικρές, διαχειρίσιμες μονάδες λογικής ταιριάζει πολύ με τις ευέλικτες μεθοδολογίες ανάπτυξης λογισμικού. Η ανάπτυξη τέτοιων αυτόνομων μονάδων μπορεί να επιτευχθεί από προγραμματιστές με μικρή εμπειρία οι οποίοι μπορεί να μην έχουν εικόνα όλης της λειτουργικότητας του συστήματος και να μην έχουν εντρυφήσει στον επιχειρηματικό τομέα (business domain) (Ingeno, 2018).

### 2.2.2 Βασικά κλειδιά της υπηρεσιοστραφούς ανάπτυξης λογισμικού

Υπάρχουν κάποια βασικά κλειδιά στον υπηρεσιοστραφή προσανατολισμό σχεδιασμού λογισμικού (Ingeno, 2018; Sheikh et al., 2011; Alanazi et al.,2019). Αυτά είναι:

- Καθορισμένο συμβόλαιο της υπηρεσίας
- Χαλαρή διασυνδεσιμότητα των υπηρεσιών
- Αφαιρετική ανάπτυξη των υπηρεσιών
- Επαναχρησιμοποίηση των υπηρεσιών
- Ενορχήστρωση/διακυβέρνηση των υπηρεσιών
- Αυτονομία των υπηρεσιών
- Statelessness[1] των υπηρεσιών
- Φυλλομέτρηση (discoverability) των υπηρεσιών
- Σύνθεση των υπηρεσιών

Κάθε υπηρεσία θα πρέπει να έχει μια σαφώς καθορισμένη διεπαφή μαζί με την απαραίτητη περιγραφή της. Η υιοθέτηση πρωτοκόλλων και κανόνων επιτρέπει την διαλειτουργικότητα και εξυπηρετεί τον σκοπό των υπηρεσιών για εύκολη κατανόηση.

Οι χαλαρά διασυνδεδεμένες υπηρεσίες υπόκεινται σε αλλαγές ταχύτερα και ευκολότερα (Beydoun et al., 2013). Ελαχιστοποιώντας την εξάρτηση μιας υπηρεσίας με μια άλλη ή με άλλο σημείο του

[1] Η κατάσταση λειτουργίας ενός εξυπηρετητή κατά την οποία δεν διατηρεί καθόλου στοιχεία σχετικά με τα αιτήματα που λαμβάνει και τις αποκρίσεις που αποστέλλει.

κώδικα, επιτυγχάνεται η ελαχιστοποίηση του ρίσκου μη λειτουργίας όλης της εφαρμογής ή του συστήματος κατά την ανάπτυξη ή την αλλαγή της κάθε υπηρεσίας.

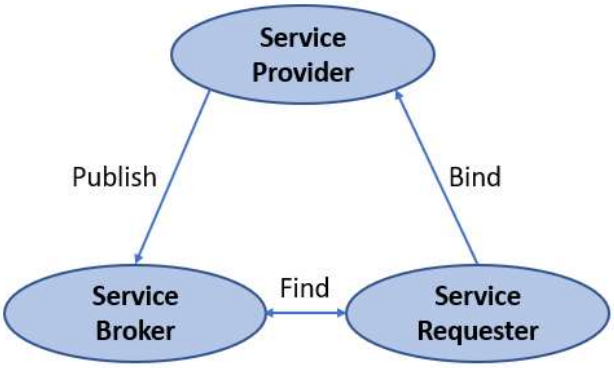
Η αφαιρετική ανάπτυξη (service abstraction) έχει ως σκοπό να αποκαλύψει μόνο εκείνη την πληροφορία της υπηρεσίας που είναι απαραίτητη για τη διαδραστικότητα με αυτήν. Η υλοποίηση θα πρέπει να γίνεται με τέτοιο τρόπο ώστε οι λεπτομέρειές της να βρίσκονται στο εσωτερικό, κρυφό κομμάτι της υπηρεσίας. Οι μετέπειτα αλλαγές που μπορεί να αφορούν στην υπηρεσία θα πρέπει να έχουν να κάνουν με την εσωτερική υλοποίηση, προκειμένου όλοι όσοι την καταναλώνουν να εξακολουθούν να τη χρησιμοποιούν χωρίς να χρειαστεί να αλλάξουν τον τρόπο κλήσης της (Ingeno, 2018).

Η επαναχρησιμοποίηση των υπηρεσιών οδηγεί σε αύξηση της παραγωγικότητας και στο κέρδος τόσο του κόστους όσο και του χρόνου υλοποίησης και συντήρησης (Nils et al., 2010). Η επαναχρησιμοποίηση είναι ένας παράγοντας που επιταχύνει τον χρόνο παράδοσης και το λανσάρισμα (release) της υπηρεσίας στην αγορά. Ενδεχομένως, η αποσύνθεση των έργων που επιτελεί ένα σύνθετο λογισμικό σε μικρότερες μονάδες υπηρεσιών να απαιτεί αρκετή ανάλυση και πολύπλοκη σκέψη. Ωστόσο, όταν τελικά επιτευχθεί η κατάτμηση, το κέρδος σε μακροπρόθεσμους όρους είναι μεγάλο. Τα συστήματα που χρησιμοποιούν πολλές μικρές αυτόνομες υπηρεσίες για την υποστήριξη των αναγκών έχει αποδειχθεί ότι είναι πιο ποιοτικά σε σχέση με τα μονολιθικά μοντέλα, αφού και ο έλεγχος τους είναι ευκολότερος (Ingeno, 2018).

Σε ένα διανεμημένο μοντέλο λογισμικού, η SOA αρχιτεκτονική αποτελείται συνήθως από 3 βασικά μέρη: Τον καταναλωτή της υπηρεσίας (αιτών), τον πάροχο της υπηρεσίας και τον διοργανωτή της υπηρεσίας (μεσίτης).

- Service requestor (Αιτών υπηρεσιών): Κόμβος δικτύου ο οποίος είναι υπεύθυνος για την ανακάλυψη και την κλήση των υπηρεσιών προκειμένου να παρέχει μια ολοκληρωμένη επιχειρηματική λύση. Η SOA αρχιτεκτονική αφήνει τις λεπτομέρειες ασφάλειας, το πρωτόκολλο μεταφοράς και τη δικτύωση, στην εκάστοτε υλοποίηση.
- Service provider (Πάροχος υπηρεσιών): Κόμβος δικτύου που είναι υπεύθυνος για την παροχή διεπαφής υπηρεσίας λογισμικού. Αυτός ο κόμβος μπορεί να αντιπροσωπεύει τις υπηρεσίες της επιχειρηματικής οντότητας ή τη διεπαφή επαναχρησιμοποιήσιμων υπηρεσιών υποσυστήματος.
- Enterprise Service Bus (ESB) (Μεσίτης υπηρεσιών): Κόμβος δικτύου που λειτουργεί ως κατάλογος ή αποθετήριο για τις δημοσιευμένες διεπαφές λογισμικού από παρόχους υπηρεσιών. Το ESB είναι ένα ενδιάμεσο λογισμικό (middleware) για τη διαχείριση των υπηρεσιών το οποίο βασίζεται σε πρότυπα αρχιτεκτονικής που υποστηρίζουν μια μεγάλη γκάμα μέσων μεταφοράς. Σε αντίθεση με την κοινή πεποίθηση, το ESB δεν βασίζεται αποκλειστικά στις υπηρεσίες Web, αλλά μπορεί επίσης να υλοποιείται βάσει του μοτίβου Enterprise Application Integration (EAI). Εν τέλει, είναι μια πλατφόρμα που χρησιμοποιείται

για να συνδυάζει υπηρεσίες ιστού, ανταλλαγές μηνυμάτων, έξυπνη δρομολόγηση, μετατροπές δεδομένων (Alanazi et al.,2019).




Εικ. 3. SOA μοντέλο

Οι υπηρεσίες θα πρέπει να σχεδιάζονται ώστε να είναι αυτόνομες και ανεξάρτητες από τα περιβάλλοντα στα οποία «τρέχουν». Όταν αυτές σχεδιάζονται με όσο το δυνατόν λιγότερη αλληλεπίδραση με τα runtime περιβάλλοντα τα οποία δεν μπορούν να ελέγξουν, έχουν καλύτερη απόδοση και αυξημένη αξιοπιστία (Sheikh et al., 2011).

Τα πρότυπα ανάπτυξης των υπηρεσιών θα πρέπει να προσανατολίζονται στην ελαχιστοποίηση της διαχείρισης της κατάστασής τους, καθώς και στον διαχωρισμό των δεδομένων αυτών από την ίδια την υπηρεσία. Οι υπηρεσίες καταναλώνουν λιγότερους πόρους όταν δεν διαχειρίζονται από μόνες τους την κατάστασή τους, όποτε δεν είναι πραγματικά απαραίτητο, γεγονός που τις καθιστά πιο αξιόπιστες στην διαχείριση των αιτημάτων κλήσης τους. Η εφαρμογή του statelessness προτύπου αυξάνει την επεκτασιμότητα (scalability) των υπηρεσιών και βελτιώνει την επαναχρησιμοποίησή τους (Ingeno, 2018; Sheikh et al., 2011).

Η φυλλομέτρηση των υπηρεσιών έχει να κάνει με την εύκολη εύρεσή τους, τόσο μέσω της αυτόματης αναζήτησής τους προγραμματιστικά, όσο και με την ανθρώπινη χειροκίνητη αναζήτηση (Esfandyari et al., 2013). Προκειμένου να επιτευχθεί αυτός ο στόχος, ο εκάστοτε προγραμματιστής θα πρέπει να φροντίσει ώστε κάθε υπηρεσία να συνοδεύεται από τα αντίστοιχα μεταδεδομένα (metadata).

Η ικανότητα σύνθεσης των υπηρεσιών προσφέρει σε ένα λογισμικό τη δυνατότητα επαναχρησιμοποίησης τους (Nils et al., 2010) και στον οργανισμό ένα από τα αδιαφιλονίκητα σημαντικότερα πλεονεκτήματα: οργανωτική και επιχειρησιακή ευελιξία (Beydoun et al., 2013).

## 2.3 Αρχιτεκτονική μικροϋπηρεσιών (Microservice architecture - MSA)

Η αρχιτεκτονική μικροϋπηρεσιών (MSA: microservice architecture) χρησιμοποιείται ως μοτίβο για την ανάπτυξη εφαρμογών που συνίστανται από μικρές, αυτόνομες, ανεξάρτητες και αυτοτελείς

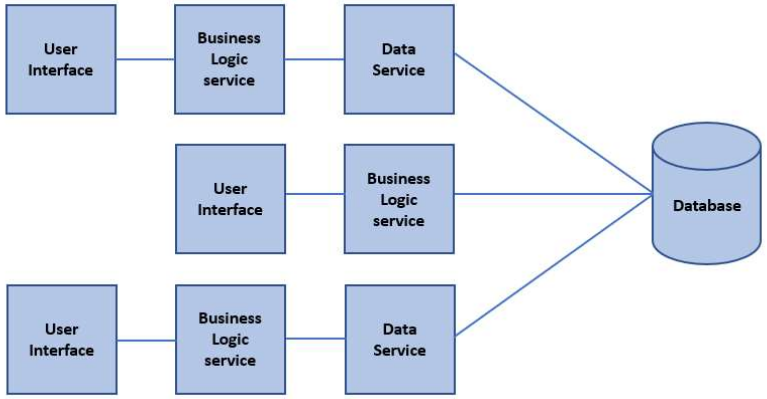
υπηρεσίες (Pradhan et al., 2020). Αυτές οι υπηρεσίες καθορίζονται από πολύ καλά ορισμένες διεπαφές που επικοινωνούν μεταξύ τους με ελαφριά πρωτόκολλα (lightweight protocols) (Viggiato et al., 2018, August 14; Sampaio et al., 2017, September; Fritzsch, 2018).

Η αρχιτεκτονική αυτή άρχισε να υιοθετείται έχοντας ως σκοπό την ανάπτυξη μικρότερων ανεξάρτητων τμημάτων λογισμικού, τα οποία δεν θα επηρεάζουν την υπόλοιπη εφαρμογή (Shadija et al., 2017, September).

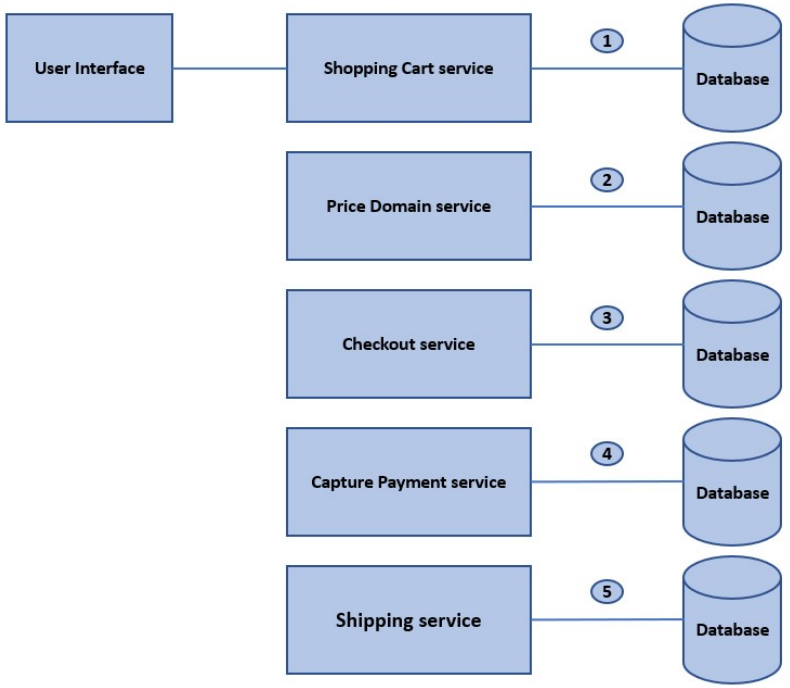
Παρόλο που η αρχιτεκτονική μικροϋπηρεσιών είναι μια παραλλαγή της υπηρεσιοστρεφούς αρχιτεκτονικής, υπάρχουν κάποιες βασικές διαφορές μεταξύ των δύο. Η πρώτη έχει χαρακτηριστεί ως "*SOA done right*" και έχει αναπτυχθεί έχοντας ως βασική ιδέα την παραδοσιακή SOA, χωρίς τα μειονεκτήματά της (Ingeno, 2018).

Αυτο έχει επιτευχθεί πρωτίστως με τη χρήση ESB-like λειτουργιών έναντι του καθαρού middleware επιχειρησιακού λεωφορείου (ESB) ως εργαλείου διαμεσολαβητή, η χρήση του οποίου μπορεί να οδηγήσει στον συγκεντρωτισμό της επιχειρησιακής λογικής. Αντί της ενορχήστρωσης της SOA αρχιτεκτονικής προτιμάται η χορογραφία με έξυπνα τελικά σημεία (endpoints) προκειμένου να διασφαλιστεί ότι διατηρείται η σχετική λογική και τα δεδομένα εντός των ορίων των υπηρεσιών, βοηθώντας με αυτό τον τρόπο να διατηρηθεί η συνεκτικότητα όσον αφορά την λειτουργικότητα και το μοντέλο (Shadija et al., 2017, September).

Στις παρακάτω εικόνες απεικονίζεται η τοπολογία μιας εφαρμογής υπηρεσιοστραφούς αρχιτεκτονικής και μιας εφαρμογής μικροϋπηρεσιών.




Εικ. 4. Παράδειγμα εφαρμογής που ακολουθεί τη SOA αρχιτεκτονική




Εικ. 5. Παράδειγμα εφαρμογής που ακολουθεί την MSA αρχιτεκτονική

### 2.3.1 Χαρακτηριστικά της αρχιτεκτονικής μικροϋπηρεσιών

Κάθε μικροϋπηρεσία υλοποιεί ένα πολύ μικρό κομμάτι της απαίτησης του συνόλου της εφαρμογής (Pradhan et al., 2020). Έτσι το κάθε micro-service τηρώντας περιορισμένη ευθύνη δραστηριότητας (scope) είναι πολύ εύκολο να αναπτυχθεί, να συντηρηθεί και να αλλάξει ή να αντικατασταθεί (Shadija et al., 2017, September).

Οι μικροϋπηρεσίες δεν είναι απλώς βιβλιοθήκες ή κομμάτια κώδικα. Έχουν επίσης στοιχεία που σχετίζονται με τα λειτουργικά συστήματα (OS: Operating System), τις πλατφόρμες (platforms) τις δομές και τα πλαίσια (frameworks) πάνω ή σύμφωνα με τα οποία εγκαθίστανται καθώς και με το περιβάλλον εκτέλεσης (run-time environment). Κατά τη διάρκεια του χρόνου εκτέλεσης καθεμία από αυτές λειτουργεί ως cloud VM ή Docker container (Docker,2022; Newman, 2015 ;Pradhan et al., 2020; Vayghana, 2020).

#### 2.3.1.1 Στοχευμένες υπηρεσίες προσανατολισμένες στον σχεδιασμό βάσει τομέα

Μέσω της σύνδεσης της υλοποίησης σε ένα εξελισσόμενο μοντέλο, είναι εύκολο να επιτευχθεί η ανάπτυξη λογισμικού με την προσέγγιση του σχεδιασμού βάσει τομέα (DDD: Domain Driven Design) (Ingeno, 2018).

#### 2.3.1.2 Διεπαφές

Μια μικροϋπηρεσία λειτουργεί ως μαύρο κουτί (black box) σε όσους θέλουν να την καταναλώσουν, κρύβοντας την πολυπλοκότητα και την υλοποίησή της και εκθέτοντας την λειτουργικότητά της μέσω της διεπαφής της (Ingeno, 2018; Pradhan et al., 2020).

#### 2.3.1.3 Αυτόνομες και ανεξάρτητες υπηρεσίες οργανωμένες γύρω από την επιχειρηματική διαθεσιμότητα (business capability)

Στην περίπτωση των πολυ-επιπέδων εφαρμογών (multi-tier applications) ο κώδικας και οι ομάδες ανάπτυξης οργανώνονται γύρω από τεχνικής άποψης λειτουργικές δομές, όπως η ομάδα ανάπτυξης λογισμικού της διεπαφής χρήστη (User Interface), η ομάδα ανάπτυξης της μονάδας της επιχειρηματικής λογικής (back-end layer), κ.λπ. (Ingeno, 2018; Shadija et al., 2017, September). Η τήρηση της Αρχής Διαχωρισμού των Διασυνδέσεων (ISP: Interface Segregation Principle) είναι αυτή που οριοθετεί τις μονάδες ανάπτυξης μιας πολυ-επίπεδης εφαρμογής (Pradhan et al., 2020).

Αυτό δεν συμβαίνει στην περίπτωση των μικροϋπηρεσιών οι οποίες δεν αναπτύσσονται σαν ένα επιπλέον κομμάτι επικοινωνίας (communication layer) μεταξύ των δομών της εφαρμογής, αλλά ως ένα κομμάτι μιας υπηρεσίας που ολοκληρωνει την επιχειρηματική ανάγκη με μια ή

περισσότερες μικροϋπηρεσίες. Για παράδειγμα η υπηρεσία “Διαχείριση Καλαθιού Αγοράς” συντίθεται από τα επιμέρους “Προσθήκη στο Καλάθι”, “Διαγραφή από το Καλάθι”, “΄Πήγαινε στο καλάθι”, “Διατήρηση προϊόντων Καλαθιού”, “Διαγραφή προϊόντων Καλαθιού” κ.λπ. (Shadija et al., 2017, September).

Έτσι όλος ο προσανατολισμός ανάπτυξης λογισμικού περιστρέφεται γύρω από την παράδοση προϊόντων - υπηρεσιών και όχι έργων (projects) λογισμικού (Shadija et al., 2017, September).

#### 2.3.1.4 Αυτόνομη αποθήκευση δεδομένων

Η δυνατότητα χρήσης διαφορετικής δομής αποθήκευσης δεδομένων σε κάθε μικροϋπηρεσία προωθεί την αυτονομία και την χαλαρή διασυνδεσιμότητα - ζεύξη των μικροϋπηρεσιών. Δεν είναι απαραίτητη η χρήση ξεχωριστής βάσης σε κάθε μια από αυτές, αλλά ενδεχομένως η χρήση ξεχωριστού πίνακα σε μια βάση (Ingeno, 2018).

Στην SOA αρχιτεκτονική ο πιο συνηθισμένος τρόπος συλλογής δεδομένων είναι με την χρήση των σχεσιακών βάσεων δεδομένων. Αντιθέτως στις μικροϋπηρεσίες γίνεται χρήση κυρίως no-SQL βάσεων και micro-SQL (Pradhan et al., 2020).

#### 2.3.1.5 Επικοινωνία με lightweight πρωτόκολλα

Για τη σύγχρονη επικοινωνία (synchronous communication) μεταξύ των συστημάτων, το REST είναι ένα από τα ευρέως διαδεδομένα πρωτόκολλα. Είναι συνηθισμένο το REST να χρησιμοποιεί την JSON (JavaScript Object Notation) σύνταξη για ανταλλαγή μηνυμάτων που είναι ένα ελαφρύ πρότυπο ανταλλαγής δεδομένων (Ingeno, 2018).

Στην περίπτωση που απαιτείται ασύγχρονη επικοινωνία μεταξύ των συστημάτων τότε υπάρχει η δυνατότητα χρήσης του AMQP (Advanced Messaging Queue Protocol) πρωτοκόλλου (Ingeno, 2018).

### 2.3.2 Πλεονεκτήματα της αρχιτεκτονικής των μικροϋπηρεσιών

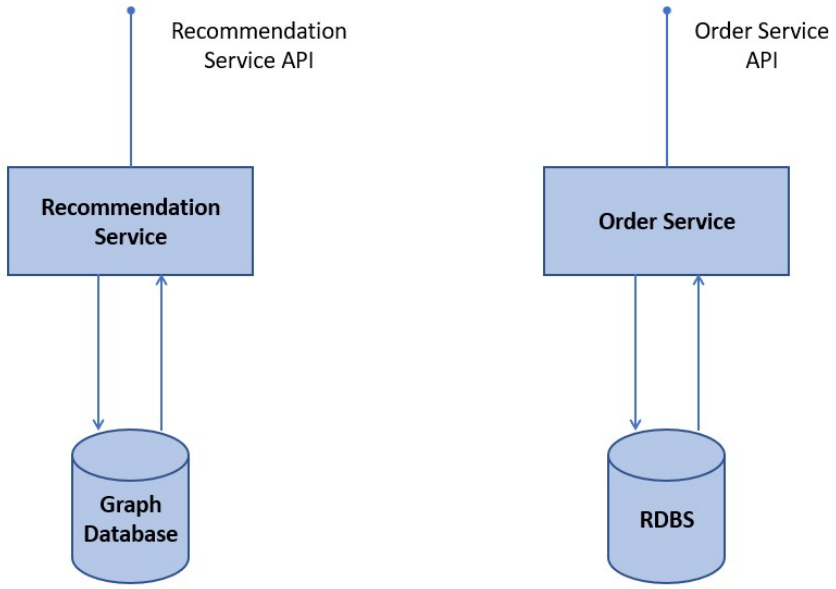
Τα βασικότερα πλεονεκτήματα της αρχιτεκτονικής των μικροϋπηρεσιών αναλύονται στις παρακάτω παραγράφους:

#### 2.3.2.1 Πολύγλωσσος προγραμματισμός

Η χρήση πολλών και διαφορετικών γλωσσών ανάπτυξης λογισμικού (polyglot programming) ανάλογα με το ποια γλώσσα προσφέρει την καλύτερη λύση για το υπό ανάπτυξη πρόβλημα είναι εφικτή στην αρχιτεκτονική μικροϋπηρεσιών (Ingeno, 2018).

2.3.2.2 Πολυποίκιλη αποθήκευση

Επίσης, ανάλογα με το είδος των δεδομένων που αποθηκεύονται και τον σκοπό που επιχειρείται να επιτευχθεί μέσω της χρήσης κάθε μικροϋπηρεσίας ενδέχεται να υπάρχει η απαίτηση για χρήση διαφορετικού τύπου βάσεων δεδομένων ή διαφορετικού μέσου αποθήκευσης όπως φαίνεται στην παρακάτω εικόνα.




Εικ. 6. Πολυποίκιλη - υβριδική αποθήκευση δεδομένων (Polyglot Persistence)

Μια μικροϋπηρεσία που αναπτύσσεται για να παρέχει συστάσεις προϊόντων βασισμένη σε προηγούμενες αγορές και αξιολογήσεις πελατών, θα πρέπει ιδανικά να κάνει χρήση μιας graph database.

2.3.2.3 Μικροϋπηρεσίες που διατηρούν ή όχι την κατάστασή τους

Υπαρχουν μικροϋπηρεσίες που λαμβάνουν ένα αίτημα (request), το επεξεργάζονται και αποστέλλουν πίσω την απάντηση (response) χωρίς να διατηρήσουν κανένα στοιχείο της κατάστασής τους κατά την ολοκλήρωση της όλης διαδικασίας (stateless microservices) και καθ' όλη τη διάρκεια της κλήσης τους.

Αντιθέτως, υπάρχουν άλλες μικροϋπηρεσίες που για να λειτουργήσουν απαιτείται η διατήρηση της κατάστασής τους. Αντί της εσωτερικής διατήρησης των διαφορετικών τιμών προτείνεται η αποθήκευση τους εξωτερικά, είτε σε μια σχεσιακή βάση δεδομένων (RDBMS), είτε σε μια μη σχεσιακή βάση δεδομένων (NoSQL), είτε στο σύννεφο (cloud storage).

Η αποθήκευση της κατάστασης μιας μικροϋπηρεσίας επικυρώνει την διαθεσιμότητα (availability), την αξιοπιστία (reliability), την κλιμάκωση-επεκτασιμότητα (scalability) και την συνέπεια (consistency) της υπηρεσίας (Ingeno, 2018; Vayghana, 2020).

2.3.2.4 Καλύτερη απομόνωση σφάλματος

Στην περίπτωση που μια μικροϋπηρεσία δεν λειτουργεί, αποφεύγεται ο κίνδυνος για την μη σωστή λειτουργία όλης της εφαρμογής, όπως συμβαίνει στην περίπτωση μιας μονολιθικής εφαρμογής. Με την απομόνωση σφάλματος (fault isolation) επιτυγχάνεται η διαθεσιμότητα άλλων λειτουργιών του συστήματος ακόμη και αν ένα ή περισσότερα μικρά κομμάτια του δεν είναι διαθέσιμα (Ingeno, 2018).

2.3.2.5 Ταχύτερους κύκλους εκδόσεων λογισμικού

Η αυτονομία που δίνει η κάθε μικροϋπηρεσία τόσο για την ανάπτυξή της όσο και για τον έλεγχο της λειτουργικότητας και της απόδοσής της έχει ως συνέπεια μια εφαρμογή ή ένα σύστημα να μπορεί να διαθέτει στην παραγωγή ταχύτερα νέες εκδόσεις (faster release cycles) με καινούργια, εμπλουτισμένη ή αλλαγμένη λειτουργικότητα.

2.3.2.6 Δυναμική επεκτασιμότητα

Σε αντίθεση με τις μονολιθικές εφαρμογές όπου η επεκτασιμότητα δύναται να είναι μόνο οριζόντια με την αναπαραγωγή περισσοτέρων ίδιων μονάδων κώδικα (instances) να τρέχουν πίσω από έναν δρομολογητή φορτίου (load-balancer) στην περίπτωση των μικροϋπηρεσιών δύναται η επανάληψη εκείνων μόνο των υπηρεσιών που παρουσιάζουν αυξημένη ζήτηση (dynamic scaling).

### 2.3.3 Μειονεκτήματα της αρχιτεκτονικής των μικροϋπηρεσιών

Ο έλεγχος της κάθε μικροϋπηρεσίας μπορεί τελικά να είναι πιο εύκολος, αλλά ο συνολικός έλεγχος όλης της παρεχόμενης λειτουργικότητας μέσω του συστήματος ή της εφαρμογής αποδεικνύεται σε μερικές περιπτώσεις δύσκολο εγχείρημα εξαιτίας των διανεμημένων επιμέρους μικροϋπηρεσιών.

Επιπροσθέτως, η αρχιτεκτονική αυτή επιφέρει μια πολυπλοκότητα καθώς θα πρέπει να μετριαστούν η ανοχή σε σφάλματα (fault tolerance), οι καθυστερήσεις του δικτύου (network latency), η δρομολόγηση των κινήσεων (load balancing) (Ingeno, 2018). Η απόδοση (performance) ενός MSA συστήματος θα πρέπει να ληφθεί σοβαρά υπόψιν κατά το σχεδιασμό καθώς είναι βασική προϋπόθεση για την επιτυχή επικοινωνία μεταξύ των μικροϋπηρεσιών (Amaral et al., 2015).

Η διαχείριση ορισμένων περιπτώσεων χρήσης που καλύπτουν περισσότερες από μια υπηρεσίες είναι δύσκολη και απαιτεί επικοινωνία και συνεργασία μεταξύ διαφορετικών ομάδων ανάπτυξης (Richardson, 2020).

Απαιτούνται προγραμματιστές με ένα ευρύ φάσμα δεξιοτήτων για τη διατήρηση και την προσαρμογή ενός συνόλου μικροϋπηρεσιών που χρησιμοποιούν μια ποικιλία γλωσσών (implementation languages) και πλαισίων υλοποίησης (frameworks) (Bozan et al., 2021).

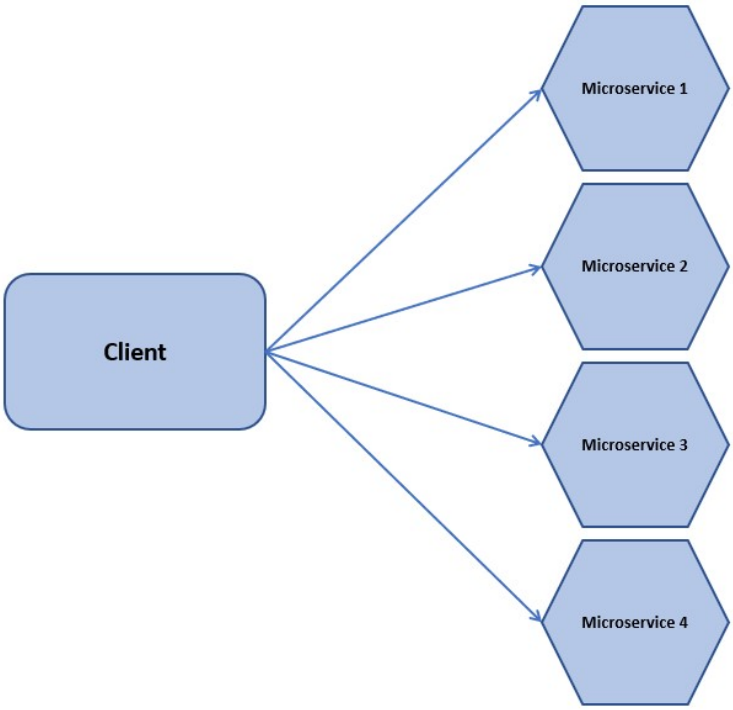
Η κατάτμηση (partitioning και decomposing) της εφαρμογής είναι ιδιαίτερα απαιτητική διαδικασία, η οποία μπορεί επιτυχώς να ολοκληρωθεί από έμπειρους αρχιτέκτονες πληροφοριακών συστημάτων οι οποίοι θα πρέπει επιπροσθέτως να διαθέτουν βαθιά γνώση των επιχειρησιακών διαδικασιών (Auera et al., 2021; Taibi et al, 2020).

Τέλος, καθώς οι μικροϋπηρεσίες αυξάνουν σε πλήθος απαιτείται η κατανάλωση περισσότερων πόρων μνήμης (Pradhan et al., 2020).

### 2.3.4 Πύλη API (API Gateway)

2.3.4.1 Άμεσο σχέδιο επαφής με την μικροϋπηρεσία - Direct design pattern

Η πιο βασική ρύθμιση για μια αρχιτεκτονική που βασίζεται σε μικροϋπηρεσίες είναι το μοτίβο σχεδίασης όπου μια εφαρμογή πελάτη υποβάλλει αιτήματα απευθείας στις μικροϋπηρεσίες, όπως φαίνεται στην παρακάτω εικόνα. Κάθε μικροϋπηρεσία έχει ένα δημόσιο τελικό σημείο (endpoint) με το οποίο μπορεί να επικοινωνήσει ο πελάτης (Adrian, 2019).




Εικ. 7. Direct design pattern

Αυτό το μοτίβο σχεδίασης είναι πολύ εύκολο στη ρύθμιση και αρκετά επαρκές για σχετικά μικρές εφαρμογές, προκαλεί εντούτοις αρκετές προκλήσεις που γίνονται όλο και πιο εμφανείς και ενοχλητικές καθώς η εφαρμογή μεγαλώνει σε μέγεθος και πολυπλοκότητα. Τα θέματα που προκύπτουν αφορούν σε (Adrian, 2019):

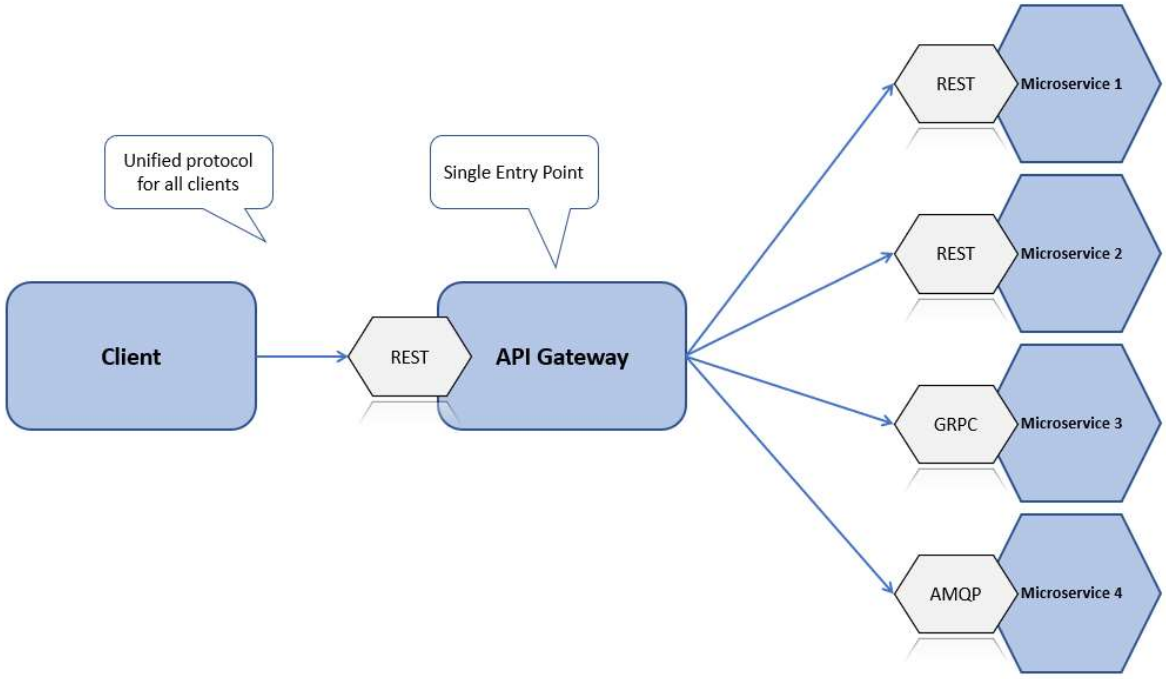
- Ζητήματα επιδόσεων (Performance issues)
- Ζητήματα επεκτασιμότητας (Scalability issues)
- Θέματα ασφάλειας (Security issues)
- Ζητήματα πολυπλοκότητας (Complexity issues)

2.3.4.2 API Gateway design pattern

Δεδομένου ότι οι μικροϋπηρεσίες συνιστώνται συνήθως για πολύπλοκες εφαρμογές, πρέπει να υπάρχουν πιο κλιμακούμενα μοτίβα με μεγαλύτερη επιχειρηματική ικανότητα και δυνατότητα εύκολης υποστήριξης πολλαπλών υπηρεσιών.

Το API Gateway ανεβάζει την επιχειρηματική ικανότητα ένα επίπεδο πιο πάνω. Όπως περιγράφεται στην παρακάτω εικόνα, παρέχει ένα επιπλέον στρώμα, ένα ενιαίο σημείο εισόδου μεταξύ μιας ομάδας μικροϋπηρεσιών και του στρώματος πρόσοψης. Αυτό το σχέδιο σχεδίασης αντιμετωπίζει όλες τις ανησυχίες που μόλις αναφέραμε, αποκρύπτοντας το τελικό σημείο των

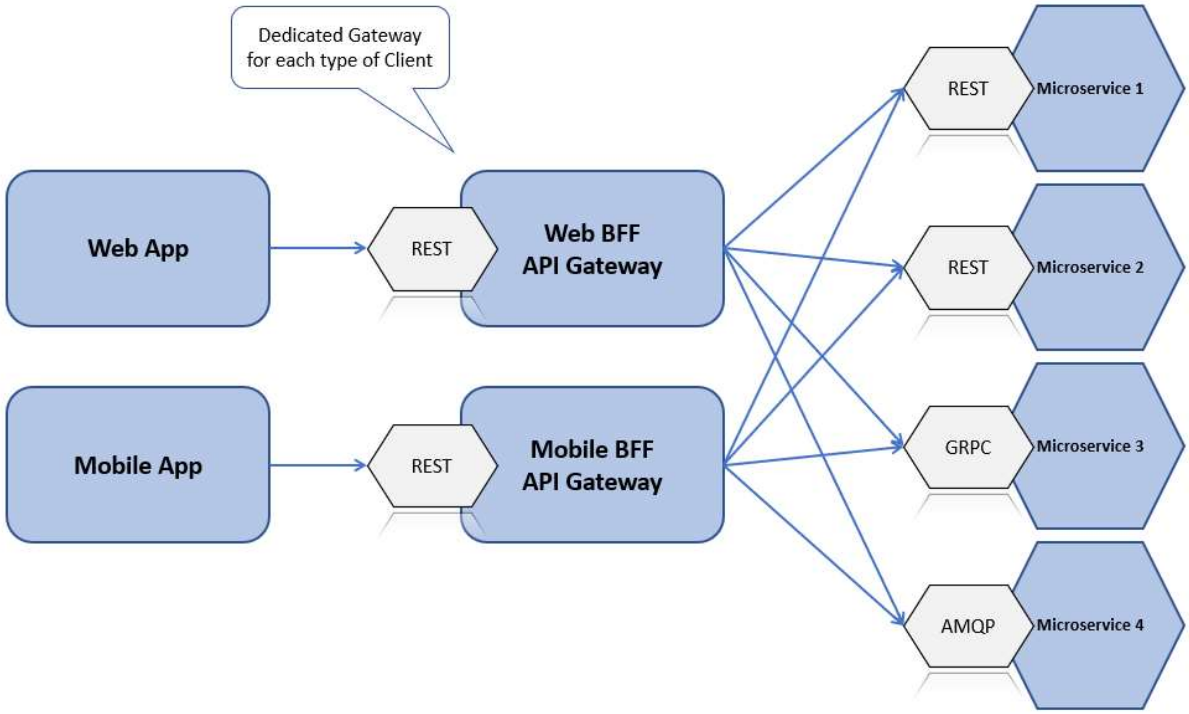
μικροϋπηρεσιών από το κοινό, αφαιρώντας τις αναφορές στις μικροϋπηρεσίες προς τον πελάτη και μειώνοντας τον λανθάνοντα χρόνο συγκεντρώνοντας πολλαπλές κλήσεις (Adrian, 2019).




Εικ. 8. Gateway design pattern

Ωστόσο, αυτό το μοτίβο σχεδιασμού μικροϋπηρεσιών εξακολουθεί να μην είναι ασφαλές από ζητήματα επεκτασιμότητας. Είναι περισσότερο από αρκετό όταν η αρχιτεκτονική περιστρέφεται γύρω από έναν πελάτη. Στην περίπτωση που υπάρχουν πολλές εφαρμογές πελατών, το μοτίβο σχεδίασης της πύλης API μπορεί τελικά να διογκωθεί, έχοντας απορροφήσει όλες τις διαφορετικές απαιτήσεις από διαφορετικές εφαρμογές πελατών. Τελικά, μπορεί πρακτικά να καταλήξει να γίνει μια μονολιθική εφαρμογή και να αντιμετωπίσει πολλά από τα ίδια προβλήματα που έχει να αντιμετωπίσει και το άμεσο μοτίβο (βλ. προηγούμενη παράγραφο).

Αν ένα σύστημα που βασίζεται σε μικροϋπηρεσίες μπορεί να έχει πολλούς πελάτες ή διαφορετικούς επιχειρηματικούς τομείς, θα πρέπει να εξεταστεί το ενδεχόμενο να χρησιμοποιηθεί το μοτίβο σχεδίασης Backend for Frontend, όπως παρουσιάζεται στην παρακάτω εικόνα (Adrian, 2019):




Εικ. 9. Backend for Frontend

Το BFF είναι ουσιαστικά μια παραλλαγή του μοτίβου API Gateway. Παρέχει, επίσης, ένα πρόσθετο επίπεδο μεταξύ των microservices και των πελατών, με τη διαφορά ότι αντί για ένα μοναδικό σημείο εισόδου, εισάγει πολλαπλές πύλες για κάθε πελάτη.

Με το BFF, μπορεί να προστεθεί ένα API προσαρμοσμένο στις ανάγκες κάθε πελάτη (client), αφαιρώντας μεγάλο μέρος της πολυπλοκότητας που προκαλείται από τη διατήρηση όλων των σημείων εισόδου σε ένα μέρος.

Αξίζει να αναφερθεί ότι αυτό το συγκεκριμένο μοτίβο μπορεί ακόμα να επεκταθεί για ιδιαίτερα περίπλοκες εφαρμογές, με τη δημιουργία διαφορετικών πυλών για συγκεκριμένους επιχειρηματικούς τομείς. Αυτό το μοντέλο είναι αρκετά ευέλικτο ώστε να ανταποκρίνεται σε σχεδόν κάθε τύπο κατάστασης που βασίζεται σε μικροϋπηρεσίες. Είναι τέλειο για συνεχή παράδοση αρχιτεκτονικής μικροϋπηρεσιών σε πραγματικά μεγάλη κλίμακα και παρέχει απίστευτες επιχειρηματικές δυνατότητες (Adrian, 2019).

#### 2.3.4.3 Διασταυρούμενα συμφέροντα της API πύλης

Σε όλες τις αρχιτεκτονικές που βασίζονται σε υπηρεσίες, υπάρχουν πολλές ανησυχίες που μοιράζονται μεταξύ όλων, ή των περισσότερων υπηρεσιών. Η αρχιτεκτονική που βασίζεται στις μικροϋπηρεσίες δεν αποτελεί εξαίρεση. Όπως έγινε κατανοητό με όσα αναφέρθηκαν σε προηγούμενες παραγράφους οι μικροϋπηρεσίες αναπτύσσονται σχεδόν μεμονωμένα. Οι

εγκάρσιες ανησυχίες (cross-cutting concerns) αντιμετωπίζονται από τα ανώτερα στρώματα στη στοίβα λογισμικού. Η πύλη API είναι ένα από αυτά τα επίπεδα. Ακολουθεί μια λίστα με διασταυρούμενα συμφέροντα που αντιμετωπίζουν οι εν λόγω πύλες (Payrott, 2015):

- Αυθεντικοποίηση (Authentication)
- Ασφάλεια μεταφορών (Transport security)
- Εξισορρόπηση φορτίου (Load-balancing)
- Αποστολή αιτήματος, συμπεριλαμβανομένης της ανοχής σφαλμάτων και της ανακάλυψης υπηρεσίας (Request dispatching)
- Επίλυση εξάρτησης (Dependency resolution)
- Μετασχηματισμοί μεταφορών (Transport transformations)

*2.3.4.3.1 Αυθεντικοποίηση (Authentication)*

Οι περισσότερες πύλες εκτελούν κάποιο είδος ελέγχου ταυτότητας για κάθε αίτημα (ή σειρά αιτημάτων). Σύμφωνα με κανόνες που είναι συγκεκριμένοι για κάθε υπηρεσία, η πύλη είτε δρομολογεί το αίτημα στις αιτούμενες μικροϋπηρεσίες είτε επιστρέφει έναν κωδικό σφάλματος. Σε πολλές περιπτώσεις το μήνυμα της επιστροφής σφάλματος είναι πολύ γενικό και ενδέχεται να μην περιέχει κάποιον κωδικό σφάλματος προκειμένου να μην αποκαλύπτεται στους clients η εσωτερική υλοποίηση της κάθε μικροϋπηρεσίας. Οι περισσότερες πύλες προσθέτουν πληροφορίες ελέγχου ταυτότητας στο αίτημα, αυτό επιτρέπει στις μικροϋπηρεσίες να εφαρμόζουν τη λογική του χρήστη όποτε απαιτείται (Payrott, 2015).

*2.3.4.3.2 Ασφάλεια (Transport security)*

Πολλές πύλες λειτουργούν ως ενιαίο σημείο εισόδου για ένα δημόσιο API. Σε τέτοιες περιπτώσεις, οι πύλες χειρίζονται την ασφάλεια μεταφοράς και στη συνέχεια αποστέλλουν τα αιτήματα είτε χρησιμοποιώντας διαφορετικό ασφαλές κανάλι είτε αφαιρώντας περιορισμούς ασφαλείας που δεν είναι απαραίτητοι μέσα στο εσωτερικό δίκτυο. Για παράδειγμα, για ένα RESTful HTTP API, μια πύλη μπορεί να εκτελέσει *τερματισμό ενός SSL πιστοποιητικού*: δημιουργείται μια ασφαλής SSL σύνδεση μεταξύ των clients και της πύλης και στη συνέχεια αποστέλλονται αιτήματα μεσολάβησης μέσω non-SSL συνδέσεων στις εσωτερικές υπηρεσίες (Payrott, 2015).

*2.3.4.3.3 Εξισορρόπηση φορτίου (Load-balancing)*

Σε σενάρια υψηλού φορτίου, οι πύλες μπορούν να διανέμουν αιτήματα μεταξύ των πολλαπλών instances της μικροϋπηρεσίας σύμφωνα με την προσαρμοσμένη λογική. Κάθε υπηρεσία μπορεί να έχει συγκεκριμένους περιορισμούς κλίμακας. Οι πύλες έχουν σχεδιαστεί για να

εξισορροπούν το φορτίο λαμβάνοντας υπόψη αυτούς τους περιορισμούς. Για παράδειγμα, ορισμένες υπηρεσίες ενδέχεται να κλιμακωθούν έχοντας πολλαπλές παρουσίες που εκτελούνται κάτω από διαφορετικά εσωτερικά τελικά σημεία. Οι πύλες μπορούν να αποστείλουν αιτήματα σε αυτά τα τελικά σημεία (endpoints) ή ακόμη και να ζητήσουν τη δυναμική εγκατάσταση περισσότερων τελικών σημείων, για να χειριστούν το φορτίο (Payrott, 2015).

*2.3.4.3.4 Αποστολή αιτήματος (Request dispatching)*

Ακόμη και σε σενάρια κανονικού φορτίου, οι πύλες μπορούν να παρέχουν προσαρμοσμένη λογική για την αποστολή αιτημάτων. Σε μεγάλες αρχιτεκτονικές, τελικά σημεία (endpoints) προστίθενται και αφαιρούνται καθώς οι ομάδες ανάπτυξης παρέχουν ή αφαιρούν μικροϋπηρεσίες ή νέες παρουσίες μικροϋπηρεσιών δημιουργούνται, όταν για παράδειγμα γίνονται αλλαγές στην τοπολογία. Οι πύλες ενδέχεται να λειτουργούν παράλληλα με διαδικασίες εγγραφής/ανακάλυψης υπηρεσιών ή βάσεις δεδομένων που περιγράφουν τον τρόπο αποστολής κάθε αιτήματος (βλ. παράγραφος “Αναζήτηση των διαθέσιμων μικροϋπηρεσιών”). Αυτό παρέχει εξαιρετική ευελιξία στις ομάδες ανάπτυξης. Επιπλέον, οι ελαττωματικές υπηρεσίες μπορούν να δρομολογηθούν σε εφεδρικές ή γενικές υπηρεσίες που επιτρέπουν στο αίτημα να ολοκληρωθεί αντί να αποτύχει πλήρως (Payrott, 2015).

*2.3.4.3.5 Επίλυση εξάρτησης (Dependency resolution)*

Καθώς οι μικροϋπηρεσίες ολοκληρώνουν πολύ συγκεκριμένες εργασίες, ορισμένες αρχιτεκτονικές που βασίζονται σε μικροϋπηρεσίες τείνουν να γίνονται «συνομιλούμενες»: για να εκτελέσουν χρήσιμη εργασία, πολλά αιτήματα πρέπει να αποστέλλονται σε πολλές διαφορετικές υπηρεσίες. Για λόγους ευκολίας και απόδοσης, οι πύλες ενδέχεται να παρέχουν προσόψεις / "εικονικά" τελικά σημεία, που εσωτερικά δρομολογούνται σε πολλές διαφορετικές μικροϋπηρεσίες (Payrott, 2015).

*2.3.4.3.6 Μετασχηματισμοί μεταφορών (Transport transformations)*

Όπως προαναφέρθηκε, οι μικροϋπηρεσίες αναπτύσσονται συνήθως μεμονωμένα και οι ομάδες ανάπτυξης έχουν μεγάλη ευελιξία στην επιλογή της πλατφόρμας ανάπτυξης. Αυτό μπορεί να οδηγήσει σε μικροϋπηρεσίες που επιστρέφουν δεδομένα και χρησιμοποιούν πρωτόκολλα που δεν είναι βολικά για τους πελάτες στην άλλη πλευρά της πύλης. Η πύλη πραγματοποιεί τους απαραίτητους μετασχηματισμούς, έτσι ώστε οι πελάτες να μπορούν ακόμα να επικοινωνούν με τις μικροϋπηρεσίες πίσω από αυτήν. Για παράδειγμα το μετασχηματισμό HTTP αιτημάτων σε AMQP ή GRPC πρωτόκολλα (Payrott, 2015).

### 2.3.5 Αναζήτηση των διαθέσιμων υπηρεσιών

Ο καταναλωτής μιας υπηρεσίας, είτε αυτή εκτίθεται μέσω μιας API πύλης είτε μέσω μιας άλλης υπηρεσίας, θα πρέπει να είναι σε θέση να ανακαλύψει τη θέση της. Σε ένα παραδοσιακό διανεμημένο μοντέλο, τα στοιχεία της υπηρεσίας που την καθιστούν προσβάσιμη και διαθέσιμη (IP address και Port) είναι γενικά στατικά και η υπηρεσία μπορεί να ανακαλυφθεί εύκολα. Για παράδειγμα, οι διαθέσιμες θέσεις θα μπορούσαν να βρεθούν μέσω της προσπέλασης ενός αρχείου παραμετροποίησης (configuration file).

Αντίθετα, στην περίπτωση των cloud-based εφαρμογών που χρησιμοποιούν μικροϋπηρεσίες, η εύρεση των θέσεών τους (service discovery) είναι πιο περίπλοκη εργασία. Ο αριθμός και η θέση των μικροϋπηρεσιών μεταβάλλεται δυναμικά στο σύννεφο. Σε αυτή την περίπτωση είναι απαραίτητος ένας κατάλογος υπηρεσιών (service registry) με τις διαθέσιμες υπηρεσίες και τις θέσεις τους.

#### 2.3.5.1 Μοντέλα διαχείρισης

Ο κατάλογος υπηρεσιών (service registry) παίζει ένα κεντρικό ρόλο στην αναζήτηση των διαθέσιμων υπηρεσιών και ανεξαρτήτως από τον τρόπο που υλοποιείται και διατίθεται θα πρέπει να είναι πάντοτε διαθέσιμος (high available) και σωστά ενημερωμένος (up to date).

Χάριν της ακρίβειας της πληροφορίας θα πρέπει οι υπηρεσίες να εγγράφονται και να διαγράφονται από τον κατάλογο. Αυτό δύναται να υλοποιηθεί είτε με το μοντέλο της αυτο-διαχείρισης, είτε με το μοντέλο της διαχείρισης μέσω τρίτων.

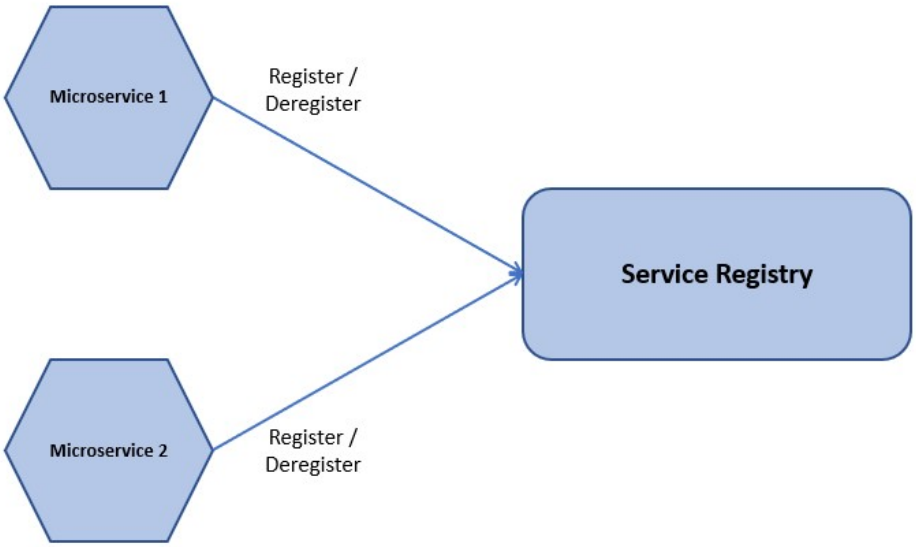
##### *2.3.5.1.1 Μοντέλο αυτο-διαχείρισης*

Σε αυτή την υλοποίηση κάθε μικροϋπηρεσία θα πρέπει να αναλαμβάνει να εγγράφεται και να διαγράφεται από μόνη της στον σχετικό κατάλογο υπηρεσιών. Η εγγραφή γίνεται αυτόματα με την έναρξη (start-up) του microservice. Περιοδικά και για την επιβεβαίωση της διαθεσιμότητάς τους, οι υπηρεσίες αποστέλλουν ένα heartbeat request προκειμένου να δηλώσουν ότι είναι διαθέσιμες (alive and responsive).

Στην περίπτωση που μια υπηρεσία δεν είναι διαθέσιμη, ενδέχεται να μην μπορεί να εκτελέσει τη διαγραφή της από τον κατάλογο.

Σε περίπτωση δε, που οι διαφορετικές μικροϋπηρεσίες χρησιμοποιούν διαφορετικές γλώσσες προγραμματισμού (programming languages) και διαφορετικές δομές ανάπτυξης (frameworks), απαιτείται να αναπτυχθεί λογική διαχείρισης από την εκάστοτε ομάδα ανάπτυξης για καθεμία από αυτές.

Αυτό το μοντέλο αυτο-διαχείρισης συστήνεται για μικρής κλίμακας εφαρμογές.




Εικ. 10. Self-registration pattern

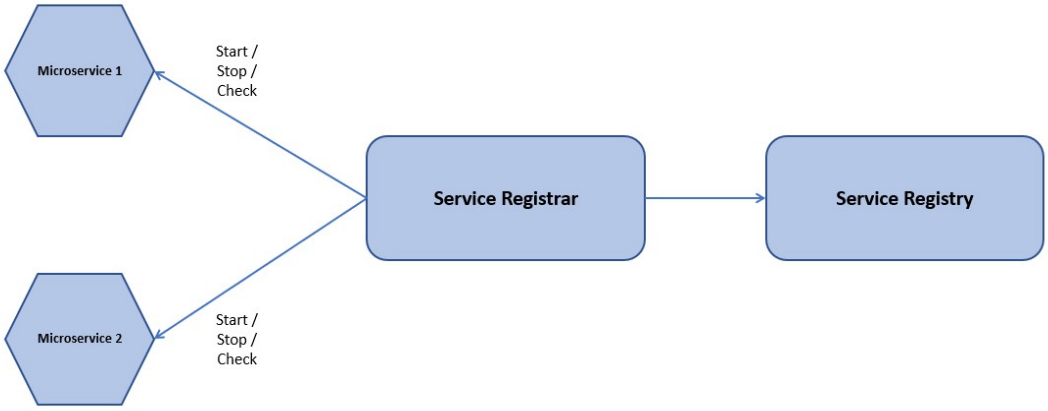
*2.3.5.1.2 Μοντέλο διαχείρισης μέσω τρίτων*

Στην περίπτωση χρήσης μοντέλου διαχείρισης μέσω τρίτων, ένα κομμάτι του συστήματος που συνήθως αναφέρεται ως ληξιαρχείο υπηρεσιών (service registrar), είναι αφοσιωμένο στην εγγραφή, διαγραφή και τον έλεγχο της διαθεσιμότητας των υπηρεσιών. Το κομμάτι αυτό είναι πολύ σημαντικό για την διαθεσιμότητα των υπηρεσιών για αυτό θα πρέπει να είναι πάντοτε διαθέσιμο (highly available).

Σε αυτή την υλοποίηση οι μικροϋπηρεσίες διαχωρίζονται από τον κατάλογο και υλοποιούν μόνο την βασική λειτουργικότητά τους, χωρίς να ασχολούνται με την εγγραφή τους και τη διαγραφή τους από τον κατάλογο.

Επίσης, να σημειωθεί ότι δεν απαιτείται πλέον να αναπτυχθεί λογική διαχείρισης της μικροϋπηρεσίας από την εκάστοτε ομάδα ανάπτυξης σε καθεμία από αυτές, σε αντίθεση με το μοντέλο της αυτο-διαχείρισης.

Το μειονέκτημα αυτού του μοντέλου είναι ότι το ληξιαρχείο των υπηρεσιών θα πρέπει να υλοποιηθεί σαν ξεχωριστό κομμάτι του συστήματος.




Εικ. 11. Third-party registration pattern

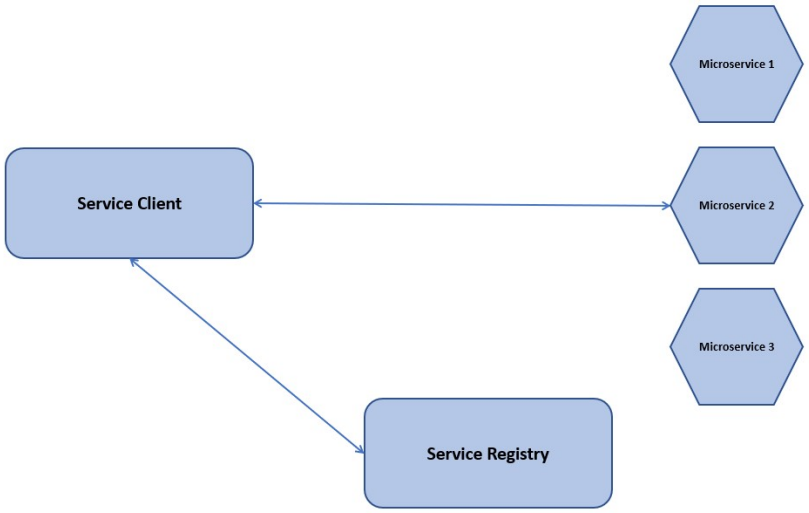
### 2.3.5.2 Τύποι διαχείρισης

Υπάρχουν δύο ξεχωριστοί τύποι διαχείρισης της εύρεσης των μικροϋπηρεσιών:

- Μοντέλο διαχείρισης από τον πελάτη (client-side discovery pattern)
- Μοντέλο διαχείρισης από τον εξυπηρετητή (server-side discovery pattern)

#### *2.3.5.2.1 Μοντέλο διαχείρισης από τον πελάτη*

Στην περίπτωση του μοντέλου διαχείρισης των υπηρεσιών από τον πελάτη (client-side discovery pattern) ο αιτών της υπηρεσίας, είτε πρόκειται για μια API πύλη, είτε πρόκειται για άλλη υπηρεσία, ρωτάει τον διαχειριστή των υπηρεσιών για τις διαθέσιμες υπηρεσίες όπως φαίνεται στην παρακάτω εικόνα.



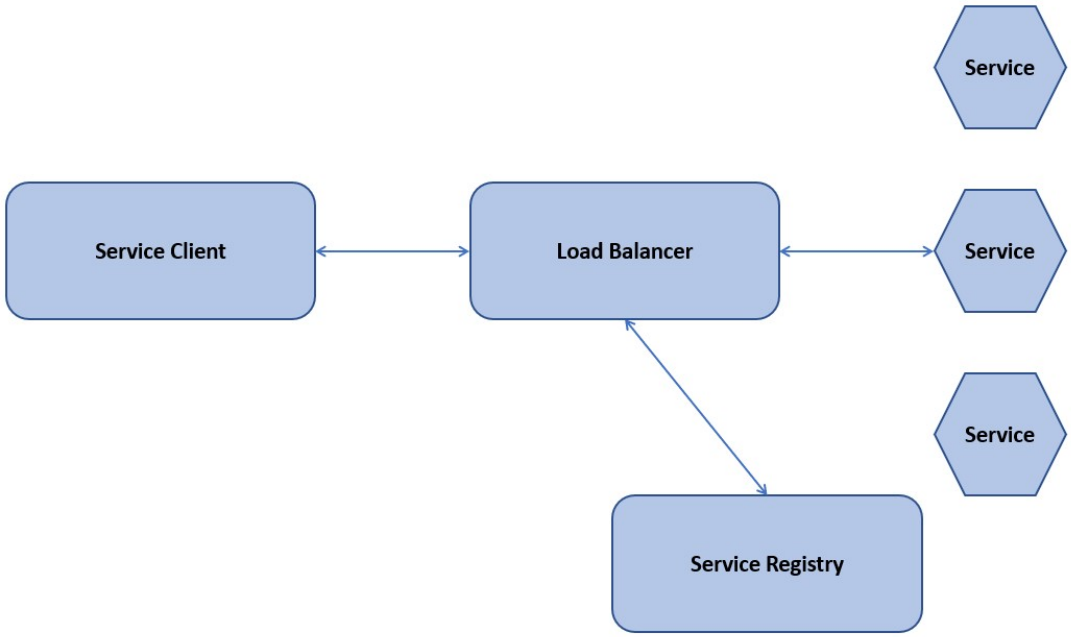

Εικ. 12. Client-side discovery pattern

Όταν ανακτηθεί ο κατάλογος με τις διαθέσιμες υπηρεσίες ο αιτών χρησιμοποιεί έναν αλγόριθμο δρομολογητή κίνησης (load balancing algorithm) για να επιλέξει μια από τις διαθέσιμες οντότητες (instances). Σε αυτή την περίπτωση ο αιτών αλληλεπιδρά με μια συγκεκριμένη οντότητα.

Σε αυτό το μοντέλο, υπάρχει αλληλεπίδραση (coupling) μεταξύ του αιτούντος της υπηρεσίας (client) και του διαχειριστή των υπηρεσιών (service registry). Σε οργανισμούς που εκμεταλλεύονται την χρήση πολλών και διαφορετικών γλωσσών προγραμματισμού (programming languages) και διαφορετικών δομών ανάπτυξης (frameworks) για κάθε μικροϋπηρεσία, απαιτείται, η λογική ανάκτησης της υπηρεσίας, να γραφτεί για καθεμία από αυτές.

*2.3.5.2.2 Μοντέλο διαχείρισης από τον εξυπηρετητή*

Στην περίπτωση του μοντέλου διαχείρισης των υπηρεσιών από τον εξυπηρετητή (server-side discovery pattern), ο αιτών της υπηρεσίας κάνει ένα αίτημα προς τον δρομολογητή (router) ο οποίος είναι τυπικά ένας δρομολογητής κίνησης (load balancer), όπως απεικονίζεται παρακάτω:




Εικ. 13. Server-side discovery pattern

Σε αυτό το μοντέλο, ο δρομολογητής κίνησης ρωτάει το διαχειριστή των υπηρεσιών για τη θέση των διαθέσιμων οντοτήτων των υπηρεσιών, στοιχείο που είναι υπεύθυνο για την δρομολόγηση των κλήσεων. Ο διαχειριστής υπηρεσιών μπορεί να είναι κομμάτι του ίδιου του δρομολογητή ή ένα ξεχωριστό στοιχείο.

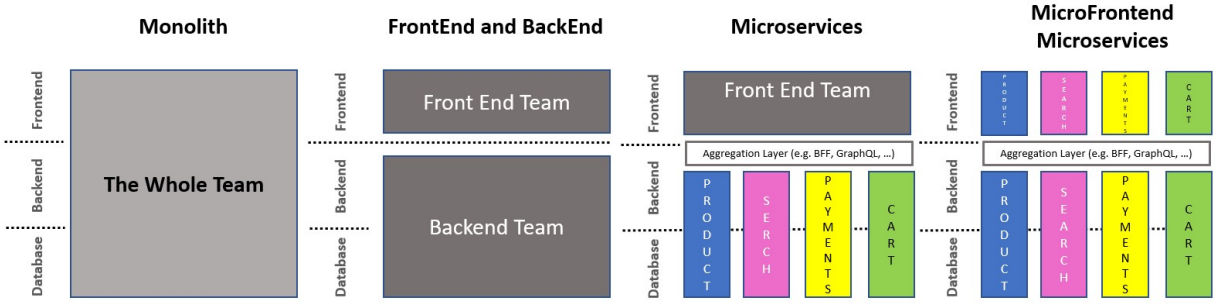
Σε αντίθεση με αυτό που συμβαίνει στο μοντέλο του διαχείρισης των υπηρεσιών από τον αιτούντα, ο κώδικας του είναι απλούστερος μιας και δεν αλληλεπιδρά απευθείας με τον διαχειριστή ούτε και θα πρέπει να υλοποιήσει αλγόριθμο δρομολόγησης κίνησης.

Τα μειονεκτήματα αυτής της λύσης είναι ότι θα πρέπει να υλοποιηθεί το στοιχείο που αφορά στην διασυνδεσιμότητα του δρομολογητή κίνησης με τον διαχειριστή των υπηρεσιών, εκτός αν παρέχεται από το σύννεφο (cloud), καθώς και ότι προστίθενται περισσότεροι δικτυακοί κόμβοι (internet hops) με ότι συνέπειες επιφέρει η προσθήκη αυτή -ακόμη ένα πιθανό σημείο αποτυχίας, καθυστερήσεις κ.λπ.-.

### 2.3.6 Micro Frontends

Η αρχιτεκτονική των μικροϋπηρεσιών ξεκινά συχνά με τη διαχείριση δεδομένων και λογικής από την πλευρά του διακομιστή (server-side), αλλά, σε πολλές περιπτώσεις, η διεπαφή χρήστη εξακολουθεί να αντιμετωπίζεται ως μονόλιθος. Ωστόσο, μια πιο προηγμένη προσέγγιση, που ονομάζεται micro frontends, δίνει τα πρότυπα για τον σχεδιασμό της διεπαφής χρήστη μιας εφαρμογής (UI) επίσης με βάση τις μικροϋπηρεσίες.

Αυτή η προσέγγιση δημιουργεί ένα σύνθετο περιβάλλον για το σύνολο της εφαρμογής όπου παράγονται μικροϋπηρεσίες στον διακομιστή και καταναλώνονται απο μικροϋπηρεσίες στη διεπαφή αντί απλώς να υπάρχει μια μονολιθική UI εφαρμογή που καταναλώνει τις μικροϋπηρεσίες του διακομιστή (Anil et al., 2021 September).




Εικ. 14. Micro Frontends με μικροϋπηρεσίες (Micro-architecture)

#### 2.3.6.1 Ανάγκη υιοθέτησης της αρχιτεκτονικής μικροϋπηρεσιών στο frontend

Ο όρος Micro Frontends πρωτοεμφανίστηκε στο ThoughtWorks Technology Radar στα τέλη του 2016. Επεκτείνει τις έννοιες των μικροϋπηρεσιών στον κόσμο του frontend.

Η τρέχουσα τάση είναι να δημιουργηθεί μια πλούσια σε χαρακτηριστικά και ισχυρή εφαρμογή προγράμματος περιήγησης, γνωστή και ως εφαρμογή μιας σελίδας, η οποία βρίσκεται πάνω από μια αρχιτεκτονική μικροϋπηρεσιών. Με την πάροδο του χρόνου το επίπεδο frontend, που

συχνά αναπτύσσεται από μια ξεχωριστή ομάδα, μεγαλώνει και γίνεται πιο δύσκολο να διατηρηθεί. Αυτός ο μονόλιθος ονομάζεται Frontend Monolith.

Η ιδέα πίσω από το micro frontends είναι η θεώρηση ενός ιστοτόπου ή μιας εφαρμογής Ιστού ως μια σύνθεση λειτουργιών που ανήκουν σε ανεξάρτητες ομάδες. Κάθε ομάδα έχει έναν ξεχωριστό τομέα επιχειρηματικής δραστηριότητας στον οποίο ειδικεύεται. Μια ομάδα δύναται να είναι πολλαπλά λειτουργική και να αναπτύσσει τα χαρακτηριστικά της από άκρο σε άκρο, από τη βάση δεδομένων έως τη διεπαφή χρήστη (Geers, 2020).

Αυτή η ιδέα δεν είναι νέα και έχει πολλά κοινά με την έννοια των Αυτοτελών Συστημάτων. Στο παρελθόν, προσεγγίσεις όπως αυτή ονομάζονταν Frontend Integration for Verticalised Systems. Σαφώς το Micro Frontends είναι πιο φιλικός και λιγότερο ογκώδης όρος (Geers, 2021).

#### 2.3.6.2 Πλεονεκτήματα χρήσης

Αντί να ορίζονται οι μικροπροοπτικές με όρους συγκεκριμένων τεχνικών προσεγγίσεων ή λεπτομερειών υλοποίησης, δίνεται έμφαση στα χαρακτηριστικά που προκύπτουν και στα οφέλη που προσφέρουν. Πιο αναλυτικά, με την υιοθέτηση του μοντέλου επιτυγχάνονται:

- **Σταδιακές αναβαθμίσεις**

Για πολλούς οργανισμούς αυτό είναι και το έναυσμα της υιοθέτησης του μοντέλου. Ο παλιός, μεγάλος, μπροστινός μονόλιθος αγκιστρώνεται από το τεχνολογικό stack του παρελθόντος ή από κώδικα γραμμένο υπό πίεση παράδοσης, και φτάνει ένα σημείο που μια συνολική επανεγγραφή είναι δελεαστική. Προκειμένου να αποφευχθεί ο κινδύνους μιας πλήρους επανεγγραφής, θα ήταν προτιμητέο να αποδομείται η παλιά εφαρμογή κομμάτι-κομμάτι και εν τω μεταξύ να συνεχίσει η παράδοση νέων δυνατοτήτων στους πελάτες χωρίς το βάρος του μονόλιθου.

Με αυτή την προσέγγιση παρέχεται μεγαλύτερη ελευθερία λήψης αποφάσεων κατά περίπτωση και για μεμονωμένα μέρη του προϊόντος και η δυνατότητα σταδιακών αναβαθμίσεων στην αρχιτεκτονική, τις εξαρτήσεις και στην εμπειρία του χρήστη. Εάν υπάρξει μια σημαντική αλλαγή στο κύριο πλαίσιο (framework), κάθε micro frontend μπορεί να αναβαθμιστεί όποτε έχει νόημα και αξία για το κομμάτι λειτουργικότητας το οποίο ολοκληρώνει, αντί του εξαναγκασμού της αναβάθμισης επί του συνόλου της εφαρμογής. Επίσης, διευκολύνεται ο πειραματισμός με νέες τεχνολογίες ή νέους τρόπους αλληλεπίδρασης, μιας και αυτή μπορεί να επιτευχθεί με ένα πιο απομονωμένο τρόπο από ό,τι σε ένα front end μονόλιθο (Jackson, 2019 June; Wanga et al, 2020).

- **Αποσυνδεδεμένος κώδικας βάσης**

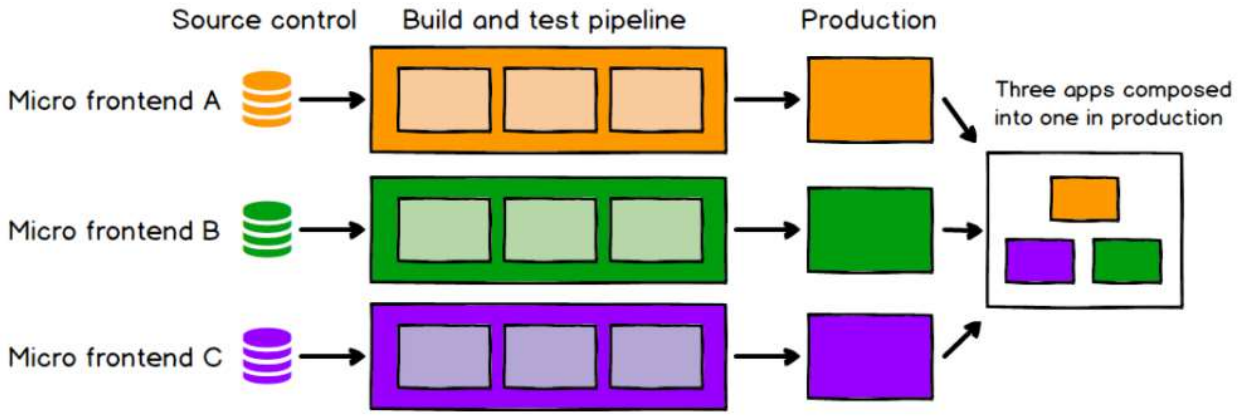
Ο πηγαίος κώδικας για κάθε μεμονωμένο micro frontend είναι εξ’ ορισμού πολύ μικρότερος από τον πηγαίο κώδικα μιας μεμονωμένης μονολιθικής διεπαφής. Αυτοί οι μικρότεροι κώδικες βάσης (codebases) τείνουν να είναι απλούστεροι και ευκολότεροι στην ανάπτυξη και στην

συντήρηση. Επιπλέον, αποφεύγεται η πολυπλοκότητα που προκύπτει από ακούσια και ακατάλληλη σύζευξη μεταξύ εξαρτημάτων που δεν θα έπρεπε να γνωρίζουν το ένα για το άλλο.

Φυσικά, μια ενιαία, υψηλού επιπέδου αρχιτεκτονική απόφαση, δεν αποτελεί εχέγγυο του καλού και καθαρού κώδικα. Η υιοθέτηση του μοντέλου δεν απαλλάσει τους αρχιτέκτονες και τους προγραμματιστές στο να καταβάλλουν προσπάθεια για την ποιότητά του. Το πλαίσιο βοηθάει ώστε να αποφευχθούν οι εύκολα κακές αποφάσεις (Jackson, 2019 June; Wanga et al, 2020).

- **Ανεξάρτητη ένταξη στην παραγωγή**

Ακριβώς όπως και με τις μικροϋπηρεσίες, η ανεξάρτητη δυνατότητα ένταξης στην παραγωγή των micro frontends είναι το κλειδί που μειώνει τον κίνδυνο για καθολικές αποτυχίες της εφαρμογής. Ανεξάρτητα από το πώς ή πού φιλοξενείται ο κώδικας διεπαφής, κάθε micro frontend θα πρέπει να έχει τη δική του γραμμή συνεχούς παράδοσης, η οποία ξεινάει από τη δημιουργία (development), συνεχίζει με τις δοκιμές (tests) και ολοκληρώνεται με την εγκατάσταση στην παραγωγή (deployment). Εάν ένα κομμάτι κώδικα / χαρακτηριστικό του micro frontend είναι έτοιμο να πάει στην παραγωγή, θα πρέπει αυτή η απόφαση να εξαρτάται από την ομάδα που το χτίζει και το συντηρεί, ανεξαρτήτως της τρέουσας κατάστασης των άλλων κώδικων βάσης (codebases) (Jackson, 2019 June; Wanga et al, 2020).

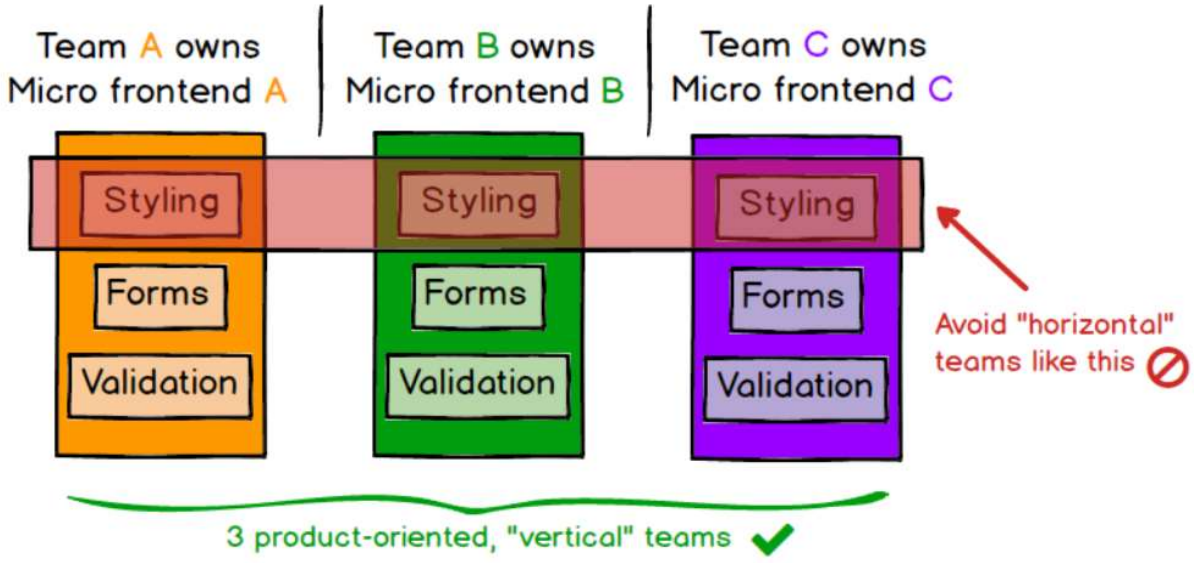



Εικ. 15. Κάθε Micro Frontend εντάσσεται στην παραγωγή αυτόνομα

- **Αυτόνομες ομάδες**

Οι ομάδες μπορούν να έχουν πλήρη κυριότητα όλων όσων χρειάζονται για να προσφέρουν αξία στους πελάτες, κάτι που τους επιτρέπει να κινούνται γρήγορα και αποτελεσματικά (Wanga et al, 2020). Για να λειτουργήσει αυτό, οι ομάδες θα πρέπει να σχηματιστούν γύρω από κάθετες τομές της επιχειρηματικής λειτουργικότητας και όχι γύρω από τεχνικές δυνατότητες. Ένας εύκολος τρόπος για την επίτευξη αυτής της προσέγγισης είναι η διαμόρφωση του τελικού προϊόντος με βάση το τι θα δουν οι τελικοί χρήστες, έτσι ώστε κάθε micro frontend να ενσωματώνει μία μόνο σελίδα της εφαρμογής και να ανήκει από άκρο σε άκρο σε μία μόνο

ομάδα. Αυτό φέρνει μεγαλύτερη συνοχή της δουλειάς των ομάδων από ό,τι αν οι ομάδες σχηματίζονταν γύρω από τεχνικά ή «οριζόντια» ζητήματα όπως το στυλ (style), οι φόρμες (forms) ή η επικύρωση (validation) (Jackson, 2019 June).




Εικ. 16. Κάθε Micro Frontend μπορεί να ανήκει σε ξεχωριστή ομάδα ανάπτυξης

Εν κατακλείδι, οι αρχιτεκτονικές micro frontend τείνουν να είναι απλούστερες, και επομένως πιο εύκολο να συντηρηθούν και να διαχειριστούν. Οι ανεξάρτητες ομάδες ανάπτυξης μπορούν να συνεργαστούν σε μια micro frontend εφαρμογή πιο εύκολα από ότι σε έναν UI μονόλιθο. Επιτυγχάνεται η ευκολία στις αναβαθμίσεις, μιας και οι εφαρμογές μπορούν να ενημερώνονται με πιο σταδιακό και μεμονωμένο τρόπο χωρίς τον κίνδυνο διάσπασης ενός άλλου τμήματος της εφαρμογής. Η επεκτασιμότητα επιτυγχάνεται πιο εύκολα. Τέλος, κάθε micro frontend έχει τη δική του αυτόνομη διοχέτευση CI/CD προσφέροντας την απομόνωση και την αυτονομία που απαιτείται για τη γρήγορη ανάπτυξη νέων απαιτήσεων (Singh, 2021).

### 2.3.7 SOA vs MSA

Ακολουθεί ένας συνοπτικός συγκριτικός πίνακας των χαρακτηριστικών γνωρισμάτων της υπηρεσιοστρεφούς και της αρχιτεκτονικής μικροϋπηρεσιών ο οποίος μπορεί να χρησιμοποιηθεί στην αρχιτεκτονική λήψης αποφάσεων κατά την υλοποίηση έργων και προϊόντων.

Πίν. 1. Σύγκριση της SOA αρχιτεκτονικής με την αρχιτεκτονική MSA

| Χαρακτηριστικά | SOA / Υπηρεσιοστρεφής αρχιτεκτονική | MSA / Αρχιτεκτονική Μικροϋπηρεσιών |
|---|---|---|
| Definition - Ορισμός | Χαλαρά διασυνδεδεμένες και διαλειτουργικές υπηρεσίες (focus on reusability). | Ανεξάρτητα αναπτυσσόμενες υπηρεσίες (focus on decoupling). |
| Services - Υπηρεσίες | Μια εφαρμογή αποτελείται από δυο ή τρεις υπηρεσίες. | Μια εφαρμογή μπορεί να αποτελείται από δεκάδες υπηρεσίες. |
| Development process - Μεθοδολογία ανάπτυξης λογισμικού | Μια νέα απαίτηση απαιτεί προσεκτικές αλλαγές-προσθήκες στον κώδικα. | Μια νέα απαίτηση ολοκληρώνεται εύκολα με μια ανεξάρτητη υπηρεσία. |
| Business units - Επιχειρηματικές μονάδες | Οι επιχειρηματικές μονάδες εξαρτώνται η μια από την άλλη. | Οι επιχειρηματικές μονάδες είναι ανεξάρτητες μεταξύ τους. |
| Business tasks - Επιχειρηματικές ανάγκες | Κάθε υπηρεσία ολοκληρώνει περισσότερες από μια επιχειρηματικές ανάγκες. | Κάθε μικροϋπηρεσία σχεδιάζεται για να επιτελεί μια επιχειρηματική ανάγκη. |
| Composition - Σύνθεση | Συχνά γίνεται διαμοιρασμός των στοιχείων λογισμικού (component sharing). | Τις περισσότερες φορές δεν γίνεται διαμοιρασμός των στοιχείων λογισμικού (component sharing). |
| Resources - Πόροι | Διαμοιρασμός πόρων από τις επιμέρους υπηρεσίες. | Οι υπηρεσίες λειτουργούν ανεξάρτητα η μια από την άλλη. |
| Data Storage - Αποθήκευση δεδομένων | Ενιαίο επίπεδο αποθήκευσης δεδομένων το οποίο μοιράζονται όλες οι υπηρεσίες (συνήθως RDBMS σχεσιακό μοντέλο). | Πολυποίκιλη αποθήκευση (SQL, NoSQL) ανάλογα με τις ανάγκες των υπηρεσιών το οποίο δύναται να είναι ξεχωριστό για κάθε υπηρεσία. |
| Technology Stack - Τεχνολογίες | Μικρός αριθμός διαφορετικών τεχνολογιών. | Μεγάλος αριθμός διαφορετικών τεχνολογιών. |

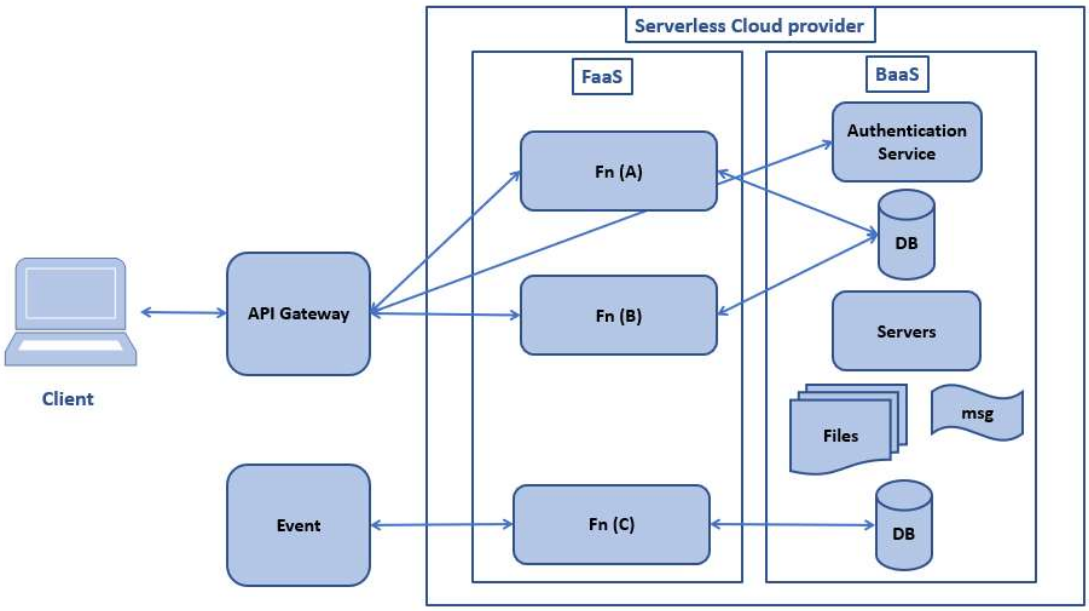
| | | |
|---|---|---|
| Message Protocols - Πρωτόκολλα ανταλλαγής μηνυμάτων | Υποστηρίζει πολλαπλά πρωτόκολλα ανταλλαγής μηνυμάτων. | Χρησιμοποιεί ελαφρά πρωτόκολλα όπως HTTP, REST, ή Thrift APIs. |
| Communication Center - Επικοινωνία Υπηρεσιών | Enterprise service bus (ESB). | API πύλες. |
| Software size - Μέγεθος λογισμικού | Το μέγεθος του λογισμικού είναι μεγαλύτερο από οποιοδήποτε συμβατικό λογισμικό. | Το μέγεθος του λογισμικού είναι μικρό για κάθε μικροϋπηρεσία. |
| Thread - Νήμα εκτέλεσης | Multi-threaded. | Single-threaded. |
| Software Development and IT operations - Ανάπτυξη και Διαχείριση Συστήματος / Εφαρμογής | Δεν καθορίζεται. | Υιοθέτηση ολοκληρωμένης διαχείρισης συστημάτων (DevOps) και αποκεντρωμένη συνεχής παράδοση (CI - Continuous Delivery). |
| Deployment process - Νέα έκδοση στην παραγωγή | Χρονοβόρα διαδικασία. | Η νέα έκδοση στην παραγωγή είναι απλή και λιγότερο χρονοβόρα διαδικασία. |
| Deployment - Φιλοξενία | Δεν καθορίζεται. | Container (Docker, Dropwizard), XaaS. |
| Scalability - Επεκτασιμότητα | Αρχιτεκτονική που δεν ευνοεί την επεκτασιμότητα. | Αύξηση της επεκτασιμότητας. |
| Governance and standards - Διακυβέρνηση και πρότυπα | Κοινή διακυβέρνηση και πρότυπα. | Χαλαρή διακυβέρνηση και ανεξάρτητα πρότυπα. |

## 2.4 Αρχιτεκτονική χωρίς διακομιστή (Serverless architecture)

Η αρχιτεκτονική χωρίς διακομιστές, ή αλλιώς serverless architecture, περιγράφει τις αρχιτεκτονικές όπου οι εταιρείες ή τα ενδιαφερόμενα μέρη αναθέτουν αποτελεσματικά τη διαχείριση δεδομένων από διακομιστές σε τρίτους. Οι ειδικοί επισημαίνουν ότι η αρχιτεκτονική χωρίς διακομιστές δεν σημαίνει ότι δεν υπάρχουν διακομιστές που να εμπλέκονται στη διαχείριση δεδομένων, απλώς σημαίνει ότι η εταιρεία απολύει την ευθύνη της διαχείρισης και

της φροντίδας των διακομιστών. Η αρχιτεκτονική χωρίς διακομιστή εκτελείται σε τεχνολογία νέφους, επιτρέποντας στους χρήστες να εκτελούν κώδικα για σχεδόν κάθε τύπο εφαρμογής ή υπηρεσίας backend με μηδενική διαχείριση (Hassan et al., 2021).

Στην παρακάτω εικόνα απεικονίζεται ένα σύστημα αρχιτεκτονικής χωρίς διακομιστή:




Εικ. 17.Serverless αρχιτεκτονική

Μια serverless πλατφόρμα μπορεί να προσφέρει ένα ή και τα δύο από τα μοντέλα Function as a Service (Faas) και Backend as a Service (BaaS) για την παροχή backend λύσεων μιας εφαρμογής.

### 2.4.1 Function-as-a-Service (FaaS)

Σε μια serverless λύση, ένα μικρό κομμάτι κώδικα, όπως είναι μια συνάρτηση, μπορεί να εκτελεστεί χρησιμοποιώντας μια εφήμερη υπολογιστική υπηρεσία για να παράξει ένα αποτέλεσμα. Η λειτουργικότητα αυτή είναι γνωστή ως Function as a Service (Faas). Με τον χαρακτηρισμό ‘εφήμερη’, ορίζεται ότι η υπηρεσία διαρκεί για ένα περιορισμένο χρονικό διάστημα· ο κώδικας εκτελείται σε έναν περιέκτη (container) ο οποίος απενεργοποιείται μετά την εκτέλεση της υπηρεσίας.

Οι συναρτήσεις εκκινούνται (trigger/invoke) εξαιτίας γεγονότων (events) ή HTTP requests. Όταν μια συνάρτηση ολοκληρώσει την επεξεργασία που είναι επιφορτισμένη να κάνει, είτε επιστρέφει την τιμή της σε αυτόν που την κάλεσε ή περνάει το αποτέλεσμα σε μια άλλη συνάρτηση που είναι μέρος της ροής εργασίας (workflow). Το αποτέλεσμα δύναται να είναι είτε ένα δομημένο αντικείμενο (HTTP response object), είτε μια μη δομημένη τιμή (string, integer).

Κάθε συνάρτηση θα πρέπει να ακολουθεί την Αρχή της Μοναδικής Αρμοδιότητας (SRP) και να είναι ταυτοδύναμη (idempotent), ήτοι πολλές εκτελέσεις της ίδιας συνάρτησης θα πρέπει να δίνουν το ίδιο αποτέλεσμα και δεν θα πρέπει να προκαλούνται δυσμενείς παρενέργειες. Επίσης, αντίγραφα της ίδιας συνάρτησης μπορεί να υπάρξουν σε περισσότερες από μια μηχανές προκειμένου να διασφαλιστεί η υψηλή διαθεσιμότητα. Τέλος, οι συναρτήσεις δύνανται να είναι είτε σύγχρονες (synchronous) είτε ασύγχρονες (asynchronous) ανάλογα με τις ανάγκες της εργασίας. Στην περίπτωση των ασύγχρονων συναρτήσεων η πλατφόρμα επιστρέφει ένα μοναδικό αναγνωριστικό (unique identifier) για τη διερεύνηση της ασύγχρονης κλήσης (Ingeno, 2018; Xu et al, 2021).

Πολύ σημαντικό ρόλο στην αρχιτεκτονική χωρίς διακομιστή λαμβάνει χώρα η πύλη διεπαφής προγραμματισμού της εφαρμογής (API Gateway). Η διεπαφή αυτή αποτελεί το σημείου εισόδου του συστήματος. Πρόκειται στην ουσία για έναν HTTP server, ο οποίος λαμβάνει αιτήματα (requests) από τους πελάτες (clients) και δρομολογεί (routes) τα αιτήματα αυτά στον κατάλληλο περιέκτη της σωστής συνάρτησης. Οι περιέκτες εκτελούν την συνάρτηση και επιστρέφουν την απάντηση στον πελάτη. Να σημειωθεί ότι οι περιέκτες δεν διατηρούν κανένα στοιχείο της κατάστασης (stateless, χωρίς κατάσταση) των πελατών (clients). Η μόνη πληροφορία που διατηρείται είναι η κατάσταση της συνεδρίας (session) (Hassan et al., 2021; Ingeno, 2018).

Οι μεγαλύτεροι πάροχοι υπολογιστικού νέφους διαθέτουν FaaS υπηρεσίες: Amazon Web Services Lambda, Microsoft Azure Functions, Google Cloud Functions και IBM Cloud Functions.

### 2.4.2 Backend-as-a-Service (BaaS)

Το Backend as a Service (BaaS) μοντέλο επιτρέπει στους προγραμματιστές να εκμεταλλευτούν υπηρεσίες εφαρμογών που παρέχονται από τρίτους φορείς παροχής υπηρεσιών (third party service applications). Αυτή η πρακτική μειώνει το χρόνο και το κόστος υλοποίησης αφού δεν απαιτείται η εσωτερική (in-house) ανάπτυξη και συντήρηση μιας λειτουργικότητας της εφαρμογής-συστήματος. Από τα παραπάνω είναι κατανοητό ότι στο FaaS μοντέλο η ομάδα ανάπτυξης θα πρέπει να γράψει τον κώδικα της κάθε συνάρτησης, ενώ στο BaaS μοντέλο γίνεται χρήση υπαρχόντων υπηρεσιών.

Παραδείγματα αυτού του μοντέλου είναι μια βάση δεδομένων, οι ειδοποιήσεις τύπου push (push notifications), η αποθήκευση αρχείων (file storage) και υπηρεσίες αυθεντικοποίησης (authentication service) (Hassan et al., 2021; Ingeno, 2018).

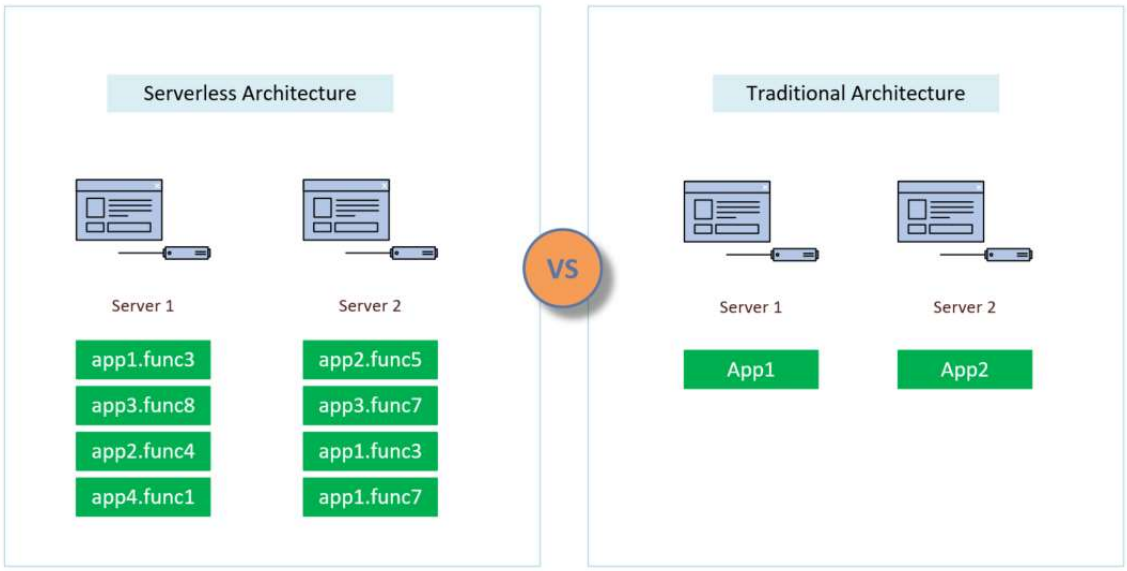
### 2.4.3 Πλεονεκτήματα αρχιτεκτονικής χωρίς διακομιστή

Ένα από τα βασικά πλεονεκτήματα του serverless computing είναι η χρέωση με βάση τους πόρους που χρησιμοποιήθηκαν. Συγκεκριμένα ο αδρανής χρόνος της εφαρμογής δεν προσμετράται στην τελική χρέωση. Κάποιες υλοποιήσεις του μοντέλου επιτρέπουν μάλιστα την

πλήρη αποδέσμευση των πόρων (scale to zero) όταν περιέλθει ένα προκαθορισμένο χρονικό διάστημα αδράνειας. Συνεπώς, ένα ισχυρό προτέρημα είναι και η κλιμάκωση των εφαρμογών σε όσα αντίγραφα χρειάζονται με αυτόματο τρόπο.

Η επεκτασιμότητα και η ευελιξία που προσφέρει η αρχιτεκτονική χωρίς διακομιστή αφαιρούν από έναν οργανισμό-εταιρία τη μη διαθεσιμότητα υπολογιστικών πόρων σε περιόδους αιχμής καθώς επίσης και το να έχει ανενεργούς διαθέσιμους πόρους σε περιόδους μη αιχμής. Η αφαίρεση του πονοκεφάλου της διαχείρισης των μηχανών επιτρέπει στους οργανισμούς να επικεντρωθούν στον κύριο επιχειρηματικό στόχο τους που είναι να παρέχουν καινοτόμες λύσεις στους πελάτες τους και νέα χαρακτηριστικά στις εφαρμογές και στα συστήματά τους. Ακόμη και μια μικρή ομάδα ανάπτυξης μπορεί να βγάλει γρήγορα ένα νέο προϊόν στην αγορά, αφού δεν θα ασχοληθεί καθόλου με θέματα υποδομών.

Η πολύγλωσση ανάπτυξη υπηρεσιών είναι και σε αυτή την τεχνολογία, όπως και στις μικροϋπηρεσίες εφικτή. Υπάρχουν βέβαια κάποιοι περιορισμοί ως προς τις γλώσσες προγραμματισμού και τις τεχνολογίες που προσφέρουν οι πάροχοι serverless λύσεων, ωστόσο το διαθέσιμο stack αυξάνεται συνεχώς (Ingeno, 2018).




Εικ. 18. Σύγκριση Serverless και παραδοσιακής αρχιτεκτονικής

### 2.4.4 Μειονεκτήματα αρχιτεκτονικής χωρίς διακομιστή

Η χρήση της αρχιτεκτονικής χωρίς διακομιστή παρουσιάζει κάποια μειονεκτήματα. Το βασικότερο όλων είναι ότι υπάρχει δυσκολία στην αποσφαλμάτωση (debugging) των εφαρμογών. Όταν υπάρχει συνεργατικότητα περισσότερων από μια συναρτήσεων για την ολοκλήρωση μιας εργασίας, μπορεί να είναι πολύ δύσκολο να εντοπιστεί η προέλευση ενός σφάλματος. Οι πάροχοι προσφέρουν μια σειρά από εργαλεία αποσφαλμάτωσης και

παρακολούθησης (monitoring), αλλά οι λύσεις που προσφέρονται είναι ακόμη σε πρώιμο στάδιο.

Θέματα ασφάλειας, εμπιστευτικότητας και απόδοσης προκύπτουν εξαιτίας της πολυμίσθωσης, μιας και η εκτέλεση συναρτήσεων για εφαρμογές διαφορετικών πελατών στην ίδια μηχανή μπορεί είτε να εκθέσει δεδομένα ή να μειώσει την ταχύτητα εκτέλεσης μια συνάρτησης.

Η αρχιτεκτονική αυτή 'δένει' την εφαρμογή με τον εκάστοτε πάροχο, και η αλλαγή του δεν είναι τελικά τόσο εύκολη όσο η μετακίνηση του κώδικα, αφού ο κάθε πάροχος διαθέτει άλλη μορφοποίηση για την λύση που προσφέρει και διαφορετικές μεθόδους νέας έκδοσης στην παραγωγή (deployment methods). Τέλος, το χτίσιμο μιας πολύπλοκης συναλλαγής (transaction) με πολλές επιμέρους συναρτήσεις ή με μια πολύ μεγάλη συνάρτηση που είναι δύσκολο να συντηρηθεί προσθέτουν πολυπλοκότητα στο σχεδιασμό και δυσκολία στην ανάπτυξη και την συντήρηση των εφαρμογών.

Εν κατακλείδι, η αρχιτεκτονική χωρίς διακομιστή μπορεί να χρησιμοποιηθεί υβριδικά μαζί με άλλες αρχιτεκτονικές για εκείνα τα σημεία της εφαρμογής που προσφέρει περισσότερα πλεονεκτήματα από τα μειονεκτήματα που την ταλανίζουν (Ingeno, 2018).

## 2.5 Αρχιτεκτονική υπολογιστικού νέφους (Cloud-native architecture)

Τα τελευταία χρόνια η χρήση τεχνολογιών υπολογιστικού νέφους έχει αυξηθεί σε σημαντικό βαθμό. Η παροχή υπηρεσιών που απαιτούν υπολογιστική ισχύ μετατοπίζεται από τη λογική αγοράς ενός προϊόντος και προσεγγίζει την έννοια της υπηρεσίας που παραδίδεται στους πελάτες μέσω του διαδικτύου από υπολογιστικά κέντρα ή αλλιώς νέφη (cloud data center).

Το υπολογιστικό νέφος αναφέρεται στην παροχή υπολογιστικής ισχύος σε πελάτες ως υπηρεσία. Ο πάροχος της υπηρεσίας δεσμεύει τους κατάλληλους πόρους που ζητήθηκαν και τους παρέχει στους πελάτες μέσω του διαδικτύου. Το υπολογιστικό νέφος (cloud computing) έχει τη δυνατότητα να προσφέρει υπηρεσίες σε πελάτες με διάφορα επίπεδα αφαίρεσης. Συγκεκριμένα, οι πάροχοι μπορούν να προσφέρουν από εικονικές μηχανές μέχρι ολοκληρωμένες υπηρεσίες για τους τελικούς χρήστες. Αυτό το μοντέλο παροχής υπηρεσιών ονομάζεται Everything as a Service (EaaS ή aaS ή XaaS). Με αυτόν τον τρόπο, ανάλογα με το ποιο μοντέλο εφαρμόζεται, ο πελάτης μπορεί να έχει στη διάθεση του μια εικονική μηχανή για να αναπτύξει τον κώδικα που επιθυμεί ή ενδεχομένως μια υπηρεσία που εκτελεί μια συγκεκριμένη εργασία και φιλοξενείται σε υποδομή υπολογιστικού νέφους (Duan et al., 2006).

Υπάρχουν διάφορες υλοποιήσεις για την παροχή υπηρεσιών υπολογιστικού νέφους ανάλογα με το μοντέλο και το είδος της παρεχόμενης υπηρεσίας. Βασικά μοντέλα παροχής υπηρεσιών υπολογιστικού νέφους αποτελούν τα: IaaS, PaaS και SaaS. Πιο αναλυτικά (Arundel, 2019; Badger, 2012; Igneno, 2018; Youssef, 2012):

- **IaaS (Infrastructure as a Service/Υποδομή ως υπηρεσία):**

Σε αυτό το μοντέλο παροχής υπηρεσιών, οι παρεχόμενοι πόροι είναι συνήθως εικονικές μηχανές (Virtual Machines), διακομιστές (servers), εξισορροπητές φόρτου (load balancers), δίκτυα ή containers. Τα βασικότερα προϊόντα αποτελούν οι εικονικές μηχανές και τα containers τα οποία είναι μια πιο ελαφριά εναλλακτική των εικονικών μηχανών. Στον πελάτη επομένως προσφέρεται η απαραίτητη υποδομή για την ανάπτυξη κώδικα ανεξαρτήτως λειτουργικού συστήματος και γλώσσας προγραμματισμού. Ταυτόχρονα η δέσμευση των πόρων αποκρύπτεται από τον πελάτη. Το μοντέλο IaaS παρέχει μεγάλη ευελιξία αφού ο πελάτης επιλέγει ελεύθερα πως θα χρησιμοποιήσει την υποδομή αλλά επιβαρύνεται με το στήσιμο και την προετοιμασία του περιβάλλοντος με το οποίο επιθυμεί να εργαστεί.

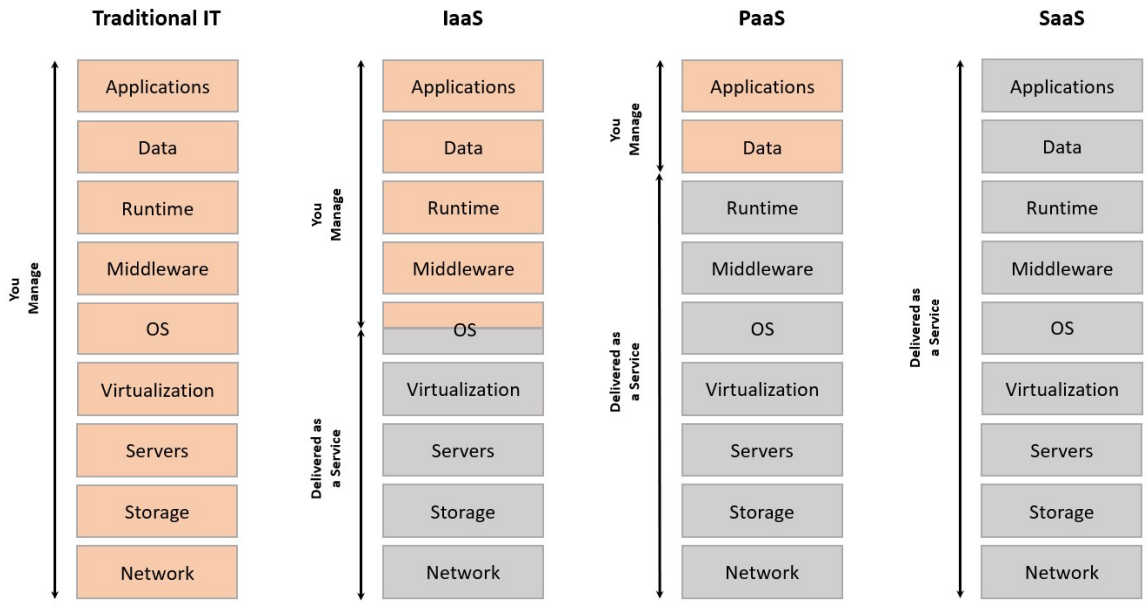
- **PaaS (Platform as a Service/Πλατφόρμα ως Υπηρεσία):**

Σε αυτό το μοντέλο παροχής υπηρεσιών, προσφέρονται στον πελάτη έτοιμα περιβάλλοντα εκτέλεσης προγραμματιστικών γλωσσών, βιβλιοθήκες κώδικα και άλλα εργαλεία. Βρίσκεται σε ένα υψηλότερο επίπεδο αφαίρεσης σε σχέση με το IaaS αφού πλέον ο πελάτης δεν ασχολείται με τη δημιουργία και οργάνωση δικτύων, λειτουργικών συστημάτων και περιβαλλόντων εκτέλεσης. Ο πελάτης λοιπόν περιορίζεται από τα προσφερόμενα περιβάλλοντα εκτέλεσης του παρόχου αλλά επιταχύνεται και διευκολύνεται η ανάπτυξη κώδικα.

- **SaaS (Software as a Service/Λογισμικό ως Υπηρεσία):**

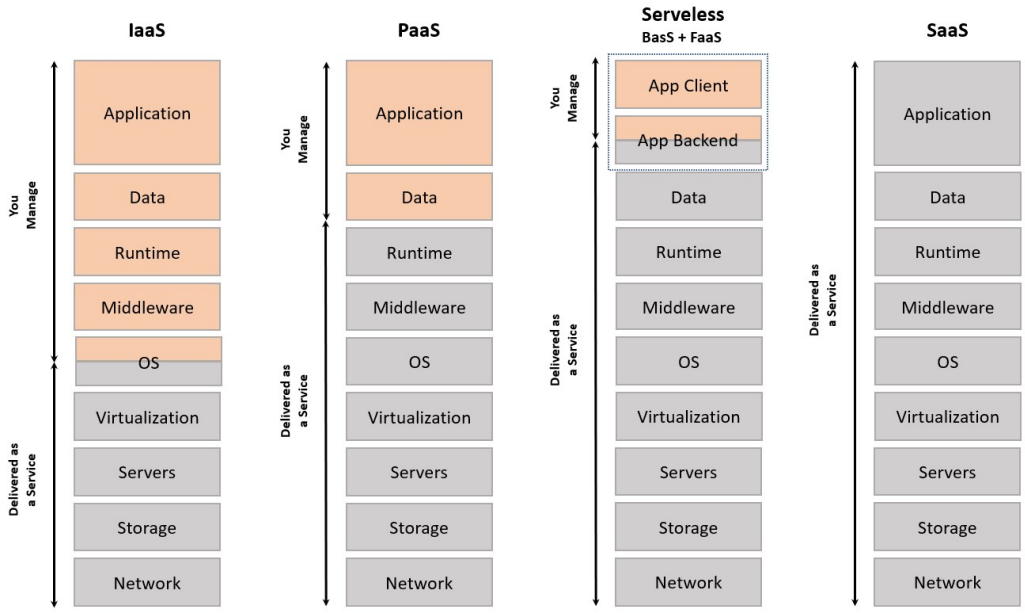
Σε αυτό το μοντέλο παροχής υπηρεσιών, προσφέρεται στον πελάτη μια εφαρμογή του παρόχου που τρέχει στο υπολογιστικό νέφος. Αποτελεί το μεγαλύτερο επίπεδο αφαίρεσης επειδή ο πελάτης δεν έχει πρόσβαση στην υποδομή που φιλοξενεί την εφαρμογή. Συνήθως πρόκειται για λογισμικό που χρεώνεται με βάση τη χρήση του ή με κάποια συνδρομή. Παραδείγματα του μοντέλου SaaS είναι υπηρεσίες αποθηκευτικού χώρου στο νέφος και σουίτες εφαρμογών γραφείου που είναι διαθέσιμες στο διαδίκτυο. Στο μοντέλο SaaS ο πελάτης δεν μπορεί να αναπτύξει δικό του κώδικα και η υλοποίηση της εφαρμογής εξαρτάται μόνο από τον πάροχο.

Στην παρακάτω εικόνα απεικονίζονται τα είδη της παρεχόμενη υπηρεσίας ανάλογα με το μοντέλο που έχει επιλεγεί. Είναι κατανοητό ότι σε ένα παραδοσιακό μοντέλο ανάπτυξης εφαρμογών όλοι οι πόροι θα πρέπει να βρεθούν και να διαχειριστούν από τον κάτοχο του λογισμικού.




Εικ. 19. “Cloud Stack": IaaS, PaaS, and SaaS VS Παραδοσιακές αρχιτεκτονικές

Στην παρακάτω εικόνα παρουσιάζεται η διαφορά μεταξύ των παραπάνω cloud λύσεων και της αρχιτεκτονικής χωρίς διακομιστή.




Εικ. 20. Serverless architecture VS IaaS, PaaS, and SaaS

### 2.5.1 Λόγοι μετάβασης στο cloud - Πλεονεκτήματα

Στα πρώτα χρόνια του υπολογιστικού νέφους, οι εταιρείες δίσταζαν να υιοθετήσουν τις εν λόγω τεχνολογίες. Εγείρονταν Θέματα όπως η απώλεια ελέγχου των υποδομών τους, η ασφάλεια των δεδομένων και η αξιοπιστία.

Από τότε, πολλές επιχειρήσεις αποφάσισαν τελικά να μεταφέρουν μέρος της IT τεχνολογίας τους στο Cloud. Ο ανταγωνισμός μεταξύ των παρόχων υπολογιστικού νέφους όπως είναι η Amazon, η Microsoft και η Google έχουν σημειώσει εξαιρετική ανάπτυξη. Εξάλλου, υπάρχουν πολλοί λόγοι για τους οποίους μια εταιρία επιλέγει να μεταφέρει τις εφαρμογές ή και τα δεδομένα της στο νέφος (Arundel, 2019; Ingeno, 2018).

#### 2.5.1.1 Μείωση κόστους

Η φιλοξενία υποδομών, εφαρμογών και δεδομένων στο νέφος μειώνει τις κεφαλαιουχικές δαπάνες (capital expenditure) με τη συρρίκνωση των δαπανών σε πάγια όπως είναι το λογισμικό (software) και το υλικό (hardware). Επιτυγχάνεται, επιπροσθέτως, μείωση των λειτουργικών δαπανών (operational expenditure) με τη συρρίκνωση των εξόδων του υποστηρικτικού προσωπικού στο IT καθώς και άλλων εξόδων όπως η ανάγκη για ηλεκτρισμό και ψύξη των μηχανημάτων (Arundel, 2019; Ingeno, 2018; Müller,2015; Sainia, 2019; Sether 2016).

#### 2.5.1.2 Μεγαλύτερη ευλιξία και επεκτασιμότητα

Το νέφος προσφέρει μεγαλύτερα επίπεδα ευελιξίας, αφού ο φόρτος εργασίας (workload) μπορεί εύκολα να προσαρμοστεί σε ξαφνικές διακυμάνσεις ζήτησης είτε προς τα πάνω είτε προς τα κάτω. Οι μεγάλοι πάροχοι υπολογιστικού νέφους έχουν παγκόσμια γεωγραφική κάλυψη προσφέροντας τους απαραίτητους πόρους την κατάλληλη στιγμή και από την κατάλληλη γεωγραφική θέση (Arundel, 2019; Ingeno, 2018; Müller,2015; Sainia, 2019; Sether 2016).

#### 2.5.1.3 Αυτόματες αναβαθμίσεις

Το καθήκον της αναβάθμισης του software των υποδομών καθώς και των νέων εκδόσεων ασφάλειας μετατίθενται στον πάροχο. Ακόμη και οι αναβαθμίσεις του υλικού, όπως μηχανήματα (servers), μνήμη (memory), υπολογιστική ισχύς (processing power), αποθηκευτικός χώρος (disk storage) αφορούν πλέον τους παρόχους. Τα υπολογιστικά κέντρα πραγματοποιούν σε τακτά διαστήματα αναβαθμίσεις στο υλικό (hardware) διασφαλίζοντας έτσι μεγαλύτερη αποτελεσματικότητα και απόδοση (Ingeno, 2018; Müller,2015; Sainia, 2019; Sether 2016).

#### 2.5.1.4 Disaster Recovery

Η δημιουργία αντιγράφων ασφαλείας (backup), η ανάκτηση δεδομένων (restore), η αποκατάσταση καταστροφών (disaster recovery) και η αδιάλειπτη λειτουργία (business

continuity) είναι σημαντικά ζητήματα που πρέπει να λαμβάνονται υπόψη κατά το σχεδιασμό και τη συντήρηση των εφαρμογών. Τα υπολογιστικά νέφη αναλαμβάνουν την ολοκλήρωση τέτοιων ζητημάτων πολύ εύκολα. Ειδικά όσον αφορά τις μικρότερες επιχειρήσεις τα θέματα αυτά είναι τεράστια πρόκληση και σε πολλές περιπτώσεις ανέφικτα στην υλοποίησή τους (Arundel, 2019; Ingeno, 2018; Müller,2015; Sainia, 2019).

### 2.5.2 Μειονεκτήματα του Cloud Computing

Όπως κάθε τεχνολογία έτσι και αυτή του υπολογιστικού νέφους διακρίνεται από μια σειρά μειονεκτημάτων. Παρακάτω αναφέρονται τα βασικότερα (Srinivasan, 2014), μερικά εκ των οποίων, εξαλείφονται με την εξέλιξη της τεχνολογίας:

#### 2.5.2.1 Θέματα Ασφάλειας

Το κύριο μέλημα με το cloud computing είναι η εύκολη πρόσβαση στα δεδομένα μέσω του Ιστού. Αν και στα θέματα ασφάλειας έχει δοθεί ιδιαίτερο βάρος από όλους τους παρόχους, το θέμα αυτό εξακολουθεί να είναι ένα από τα κριτήρια που οι εταιρείες είναι σκεπτικές όσον αφορά την μετάβαση στο cloud.

#### 2.5.2.2 Κίνδυνος απώλειας σύνδεσης στο Διαδίκτυο

Εάν δεν υπάρχει σύνδεση στο Διαδίκτυο, η πρόσβαση στα δεδομένα είναι αδύνατη.

#### 2.5.2.3 Ιδιοκτησία δεδομένων και Απόρρητο

Θέματα όπως η μη διαγραφή των δεδομένων μετά τη διακοπή της υπηρεσίας ή η χρήση τους από τους παρόχους βρίσκονται ψηλά στην ατζέντα των ερωτημάτων για την υιοθέτηση της εν λόγω τεχνολογίας.

#### 2.5.2.4 Περιορισμένη δυνατότητα προσαρμογών (customizations)

Η σε βάθος απαίτηση προσαρμογών και η ενοποίηση με τα τρέχοντα συστήματα των καθημερινών επιχειρηματικών λειτουργιών μιας εταιρίας μπορεί να μην είναι εν τέλει εφικτές.

#### 2.5.2.5 Διαθεσιμότητα

Η διαθεσιμότητα των δεδομένων και των εφαρμογών είναι μια ακόμη πρόκληση που απασχολεί όσους σκέφτονται μια πλήρη μετάβαση στο cloud.

### 2.5.3 Χαρακτηριστικά των cloud εφαρμογών

Μεταφέροντας μια εφαρμογή στο νέφος δεν σημαίνει ότι αυτή μετασχηματίζεται αυτόματα και σε cloud εφαρμογή. Ο οργανισμός **Cloud Native Computing Foundation (CNCF)** θεωρεί ότι μια εφαρμογή είναι μια εγγενής εφαρμογή σύννεφο (cloud-native application) όταν χρησιμοποιούνται τεχνολογίες ανοιχτού κώδικα (open source software stack) και η παραγόμενη εφαρμογή φιλοξενείται σε περιέκτες (containerized), ενορχηστρώνεται δυναμικά (dynamically orchestrated) και ακολουθεί την αρχιτεκτονική των μικροϋπηρεσιών (microservices-oriented).

#### 2.5.3.1 Φιλοξενία σε Περιέκτες

Οι Περιέκτες (containers) είναι ένα μέσο φιλοξενίας εφαρμογών. Είναι ελαφρού τύπου, ανεξάρτητες μονάδες λογισμικού. Η εφαρμογή, μαζί με όλες τις βιβλιοθήκες και τις εξαρτήσεις της ‘δένονται’ σε ένα αμετάβλητο πακέτο. Αυτό έχει ως αποτέλεσμα η εφαρμογή να μπορεί να τρέξει οπουδήποτε υποστηρίζονται οι περιέκτες και τελικά να εξαλείφονται απρόσμενα λάθη εξαιτίας της διαφοροποίησης των μηχανημάτων ή των περιβαλλόντων εκτέλεσης. Οι περιέκτες προσφέρουν απομόνωση στα διάφορα στοιχεία της εφαρμογής που τρέχουν σε καθέναν από αυτούς. Επιπλέον, οι περιέκτες μειώνουν τα σφάλματα των νέων εκδόσεων στην παραγωγή (deployments) (Igneno, 2018; Goniwada, 2021).

#### 2.5.3.2 Μικροϋπηρεσίες

Η φιλοξενία της εφαρμογής σε περιέκτη δεν είναι αρκετή ώστε να θεωρηθεί μια εφαρμογή ως εγγενής εφαρμογή σύννεφου. Θα πρέπει η αρχική εφαρμογή να σπάσει σε μικροϋπηρεσίες, σε μικρές δηλαδή, αυτόνομες, ανεξάρτητες υπηρεσίες (Igneno, 2018; Goniwada, 2021; Vayghana, 2020).

#### 2.5.3.3 Δυναμική ενορχήστρωση

Όπως έγινε σαφές στις δυο παραπάνω παραγράφους μια εγγενής εφαρμογή σύννεφου απαιτεί να ‘τρέχει’ σε πολλούς περιέκτες, οι οποίοι βρίσκονται σε πολλά και διαφορετικά μηχανήματα. Η διαδικασία συντονισμού των περιεκτών απαιτεί τη δυναμική τους ενορχήστρωση (dynamic orchestration). Το σύστημα θα πρέπει να ξεκινήσει την κατάλληλη στιγμή και να έχει τη δυνατότητα προσθήκης ή αφαίρεσης περιεκτών βασισμένο στη ζήτηση (demand) ή στην αποτυχία (failure launch) (Vayghana, 2020).

Υπάρχουν πολλοί και διαφορετικοί ενορχηστρωτές περιεκτών. Ο πιο διάσημος ενορχηστρωτής είναι ο Kubernetes, γνωστός και ως K8s (Kubernetes, 2022), ονομάστηκε έτσι εξαιτίας του γεγονότος ότι υπάρχουν 8 γράμματα μεταξύ του K και του S. Είναι ένας open source orchestrator που αρχικά αναπτύχθηκε από την Google. Άλλοι διάσημοι ενορχηστρωτές είναι οι Docker Swarm και Apache Mesos.

Οι πάροχοι υπηρεσιών νέφους παρέχουν και άλλους δικούς τους ενορχηστρωτές όπως για παράδειγμα ο Amazon Elastic Container Service (Amazon ECS) για τις υπηρεσίες Amazon Web Services (AWS). Η Amazon παρέχει και τον Amazon Elastic Container Service for Kubernetes (Amazon EKS) για την ενορχήστρωση των υπηρεσιών της με τον K8s. Επιπροσθέτως, προσφέρει και την τεχνολογία AWS Fargate που μπορεί να συνδυαστεί με την υπηρεσία Amazon ECS και EKS για την εκτέλεση containers (δοχείων) χωρίς την ανάγκη διαχείρισης εξυπηρετητών ή συστάδων από Amazon EC2 στιγμιότυπα (ιδεατές μηχανές).

Η Microsoft παρέχει υπηρεσίες νέφους (Microsoft Azure Container Service) που ενορχηστρώνονται μέσω του Kubernetes (Azure Kubernetes Service: AKS). Επίσης παρέχει υπηρεσίες νέφους που ενορχηστρώνονται με Mesosphere DC/OS.

Τέλος, και η Google παρέχει υπηρεσίες νέφους που ενορχηστρώνονται από τον K8s (Google Kubernetes Engine) (Igneno, 2018; Goniwada, 2021).

#### 2.5.3.4 Μηδενισμός χρόνου εκτός λειτουργίας

Για τις περισσότερες εφαρμογές, ο χρόνος λειτουργίας είναι ιδιαίτερα σημαντικός. Οποιαδήποτε διακοπή μπορεί να θεωρηθεί από τους πελάτες ως απογοήτευση ή ευκαιρία να μεταβούν σε έναν ανταγωνιστή. Επιπροσθέτως, για έναν ιστότοπο που περιλαμβάνει ηλεκτρονικό εμπόριο, ο χρόνος εκτός λειτουργίας μπορεί να σημαίνει χαμένες πωλήσεις.

Το No/Zero downtime αναφέρεται ως χαρακτηριστικό μη διακοπής της υπηρεσίας, δηλαδή μέγιστη επίτευξη της διαθεσιμότητας. Τέτοιοι υψηλοί στόχοι είναι πλέον εφικτοί στις υπηρεσίες νέφους με φιλοξενία των υπηρεσιών σε εναλλακτικές ζώνες και περιοχές διαθεσιμότητας.

Όλες αυτές οι απαιτήσεις θα αυξήσουν ενδεχομένως τον χρόνο λειτουργίας, αλλά θα φέρουν την εφαρμογή κοντά στο μηδενικό χρόνο διακοπής λειτουργίας.

Επιπροσθέτως, η κατάτμηση του συνόλου της εφαρμογής σε μικροϋπηρεσίες ανεξάρτητες μεταξύ τους διασφαλίζει ότι δεν θα υπάρξει ένα καθολικό downtime του συνόλου των υπηρεσιών (Igneno, 2018; Vayghana, 2020).

#### 2.5.3.5 Συνεχής παράδοση κώδικα

Ο υψηλός ανταγωνισμός αλλά και οι αυξανόμενες απαιτήσεις των πελατών οδηγεί σε μικρότερους κύκλους νέων εκδόσεων στην παραγωγή (release cycles). Είναι επιτακτική η ανάγκη πλέον, αντί των σημαντικών εκδόσεων (major releases) με πολλές αναβαθμίσεις της κάθε εφαρμογής που έβγαιναν στην παραγωγή μηνιαίως ή σε ετήσια βάση να απαιτείται οι εφαρμογές να αναβαθμίζονται με μεγαλύτερη συχνότητα (ημερήσια ή εβδομαδιαία βάση).

Μικρότεροι κύκλοι αναβάθμισης δίνουν τη δυνατότητα της γρήγορης ανατροφοδότησης με σημαντικές πληροφορίες από τους πελάτες. Η συνεχής παράδοση κώδικα βοηθάει την ομάδα

ανάπτυξης να κάνει γρήγορες προσαρμογές και βελτιώσεις στον κώδικα. Οι υπηρεσίες νέφους είναι σχεδιασμένες εκ των ων ουκ ανευ να υποστηρίζουν CI/CD διαδικασίες (Continuous Integration / Continuous Delivery) γεγονός που δίνει στον οργανισμό/εταιρία ανταγωνιστικό πλεονέκτημα (Igneno, 2018; Goniwada, 2021).

2.5.3.6 Υποστήριξη των εφαρμογών από διαφορετικές συσκευές

Οι χρήστες των μοντέρνων εφαρμογών χρησιμοποιούν τόσο κινητά (mobile devices), όσο υπολογιστές (desktop machines) και ταμπλέτες (tamplets) και περιμένουν κοινή εμπειρία χρήστη από όλες τις παραπάνω συσκευές. Η παραπάνω απαίτηση (support for a variety of devices) δύναται να επιτευχθεί με τις υπηρεσίες νέφους με τη διασφάλιση κοινών backend υπηρεσιών στις ανάγκες διαφορετικών front clients (Igneno, 2018).

# Κεφ.3 Μετάβαση από την μονολιθική αρχιτεκτονική στην αρχιτεκτονική μικροϋπηρεσιών

Η αποσύνθεση (decomposition) είναι ένα από τα πιο σύνθετα και δύσκολα καθήκοντα κατά την μετάπτωση των μονολιθικών εφαρμογών σε συστήματα που ακολουθείται η αρχιτεκτονική των μικροϋπηρεσιών και η ολοκλήρωσή τους εναπόκειται στις ικανότητες και την εμπειρία των αρχιτεκτόνων λογισμικού (Taibi et al, 2020, June).

## 3.1 Ανάγκη μετάβασης

Πριν την μετάβαση μιας μονολιθικής εφαρμογής σε εφαρμογή μικροϋπηρεσιών θα πρέπει να γίνει αξιολόγηση της ανάγκης μετάβασης.

Σύμφωνα με τα χαρακτηριστικά που παρατίθενται στον παρακάτω πίνακα είναι σαφές ότι η αρχιτεκτονική μικροϋπηρεσιών γίνεται ελκυστική όταν ο κώδικας βάσης (codebase) αρχίζει να αποκτά μεγάλες διαστάσεις. Αν η ομάδα χωλαίνει σε DevOps ικανότητες, τότε θα ήταν προτιμότερη η παραμονή στο μονολιθικό μοντέλο (Mäkitalo, 2018).

Στον παρακάτω πίνακα γίνεται μια σύγκριση βάσει μιας μεγάλης γκάμας χαρακτηριστικών μεταξύ της μονολιθικής αρχιτεκτονικής και της αρχιτεκτονικής μικροϋπηρεσιών.

Πίν. 2. Σύγκριση της μονολιθικής αρχιτεκτονικής με την αρχιτεκτονική MSA[2]

| Χαρακτηριστικά | Monoliths / Μονολιθικές εφαρμογές | MSA / Μικροϋπηρεσίες |
|---|---|---|
| Χρόνος διάθεσης στην αγορά (Time to market) | Σύντομος στην αρχή, αυξάνει καθώς μεγαλώνει ο κώδικας βάσης (codebase). | Καθυστερήσεις κατά την έναρξη της διαθεσιμότητας των εφαρμογών (roll-out), εξαιτίας των τεχνικών θεμάτων που πρέπει να αντιμετωπιστούν σε αυτή την αρχιτεκτονική. Ο χρόνος μειώνεται στις επόμενες εκδόσεις (deployments). |
| Αναδιάρθρωση κώδικα (Refactoring) | Δύσκολο να επιτευχθεί, καθώς οι αλλαγές δύναται να επηρεάζουν πολλά και διαφορετικά σημεία της λειτουργικότητας. | Εύκολη και ασφαλής καθώς οι αλλαγές περιέχονται μόνο σε ένα κομμάτι της λειτουργικότητας που περιέχεται στην μικροϋπηρεσία. |
| Νέα έκδοση στην παραγωγή (Deployment) | Ο κώδικας ολόκληρης της εφαρμογής θα πρέπει να ανέβει στην παραγωγή μετά από κάθε αλλαγή ή προσθήκη. | Ο κώδικας που βγαίνει στην παραγωγή αφορά μόνο μια υπηρεσία κάθε φορά ή όσες επηρεάζονται από την εκάστοτε απαίτηση. |
| Γλώσσα ανάπτυξης (Coding language) | Πολύ δύσκολο να μεταβληθεί. Απαιτείται να ξαναγραφτεί το σύνολο της εφαρμογής. | Η γλώσσα και τα εργαλεία ανάπτυξης μπορεί να είναι διαφορετικά ανά υπηρεσία. Επίσης, ο κώδικας που υποστηρίζει καθεμία από τις μικροϋπηρεσίες είναι τόσο μικρός που μπορεί εύκολα να αλλάξει η γλώσσα ανάπτυξής του μελλοντικά. |
| Κλιμάκωση - επεκτασιμότητα (Scaling) | Η κλιμάκωση μιας μονολιθικής εφαρμογής απαιτεί την έκδοση στην παραγωγή (deployment) πολλών αντιτύπων της ίδιας εφαρμογής. | Η κλιμάκωση μπορεί να γίνει ανά υπηρεσία. |
| DevOps ικανότητες (skills) | Δεν απαιτούνται ιδιαίτερες DevOps ικανότητες καθώς το εύρος των | Εμπλέκονται πολλές και διαφορετικές τεχνολογίες και |

[2] Αμφότερες οι αρχιτεκτονικές έχουν θετικά και αρνητικά (Mäkitalo, 2018)

| | | |
|---|---|---|
| | εμπλεκόμενων τεχνολογιών είναι περιορισμένο. | απαιτούνται DevOps skills από τα εμπλεκόμενα μέλη της ομάδας. |
| Κατανοησιμότητα (Understandability) | Όλη η λειτουργικότητα της εφαρμογής περιέχεται σε κοινό κώδικα βάσης, γεγονός που αυξάνει την πολυπλοκότητα και κατά συνέπεια την κατανόηση του. Η αλλαγή σε ένα σημείο ενδέχεται να επιφέρει αλλαγές και σε πολλά άλλα σημεία της εφαρμογής. | Ο κώδικας γίνεται εύκολα κατανοητός καθώς είναι αρθρωτός (modular) και οι υπηρεσίες αναπτύσσονται βάσει της Αρχής της Μοναδικής Αρμοδιότητας - SRP. |
| Απόδοση (Performance) | Οι διαθέσιμες τεχνολογίες μπορεί να μην υποστηρίζουν την βέλτιστη απόδοση. | Υπάρχει βελτιστοποίηση της απόδοσης μιας και για κάθε μικροϋπηρεσία δύναται να χρησιμοποιηθεί διαφορετική τεχνολογία ανάλογα με το πρόβλημα-ανάγκη που θα πρέπει να επιλυθεί. |

## 3.2 Στρατηγικές και Μεθοδολογίες μετάβασης

### 3.2.1 Στρατηγικές μετάβασης

Ο Christopher Grant, Senior Architect της Home Depot έθεσε το παρακάτω ερώτημα: “Αλλάζετε τροχούς κατά τη διάρκεια της οδήγησης;” Η φράση αυτή απεικονίζει τις προκλήσεις που αντιμετωπίζουν πολλοί οργανισμοί όταν σχεδιάζουν μια ανακατασκευή. Καίριο ερώτημα που τίθεται σε κάθε μετάβαση είναι το εξής: Ποια είναι η καλύτερη στρατηγική για τον εκσυγχρονισμό μιας εφαρμογής: Μια ολοκληρωτική αλλαγή που μπορεί να χαρακτηριστεί ως μια «μεγάλη έκρηξη» ή μια σταδιακή μετάβαση;

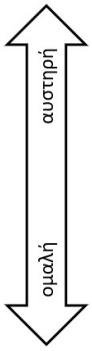
Λόγω των θεμελιωδών αλλαγών στη διαδικασία ανάπτυξης λογισμικού, η έγκαιρη συμμετοχή όλων των ενδιαφερομένων είναι ζωτικής σημασίας (Scheuch, 2015). Ο Haselböck αξιολογεί τη μετάβαση στις μικροϋπηρεσίες ως μια πολύπλοκη εργασία που θα αγγίξει σχεδόν όλες τις πτυχές ενός οργανισμού ανάπτυξης λογισμικού: σχεδιασμός, ανάπτυξη, υποδομή, συντήρηση και οργάνωση ομάδας (Haselböck et al., 2017).

Ο Scheuch (2015) επισημαίνει τρεις βασικές στρατηγικές για μια μετάβαση σε μικροϋπηρεσίες:

1. Υιοθέτηση ενός τυπικού λογισμικού από τρίτο προμηθευτή
2. Ανάπλαση με βάση το νέο αρχιτεκτονικό πρότυπο

3. Reengineering ως επαναληπτική προσέγγιση

Είναι λογικό να μπορούν να εφαρμοστούν και μικτές μορφές των παραπάνω προσεγγίσεων. Προερχόμενο από μια υπάρχουσα υλοποίηση, το παρακάτω σχήμα παραθέτει τέσσερις πιθανές στρατηγικές προς επιλογή, που κυμαίνονται από μια αυστηρή έως μια ομαλή προσέγγιση. Η ταξινόμηση έχει δημιουργηθεί από προτάσεις και αναφορές εμπειρίας όπως περιγράφεται παρακάτω.




α) Ολόκληρη η εφαρμογή ξαναγράφεται με βάση το νέο αρχιτεκτονικό στυλ.

β) "Big Bang": ορισμός της τελικής δομής και ‘σπρώξιμο’ του κώδικα στον τελικό προορισμό του.

γ) «Διαίρει και βασίλευε» – διαχωρισμός του κώδικα σε δύο κομμάτια και επανάληψη της διαδικασίας μέχρι την ολοκλήρωση της μετάβασης.

δ) Σταδιακή εξέλιξη του κώδικα προς τις μικροϋπηρεσίες.

Εικ. 21. Στρατηγικές για τις συνολικές διαδικασίες μετάβασης

Η προσέγγιση «big bang» (β) έχει ασκηθεί και τεκμηριωθεί στην εταιρεία SoundCloud (Stenberg, 2014).

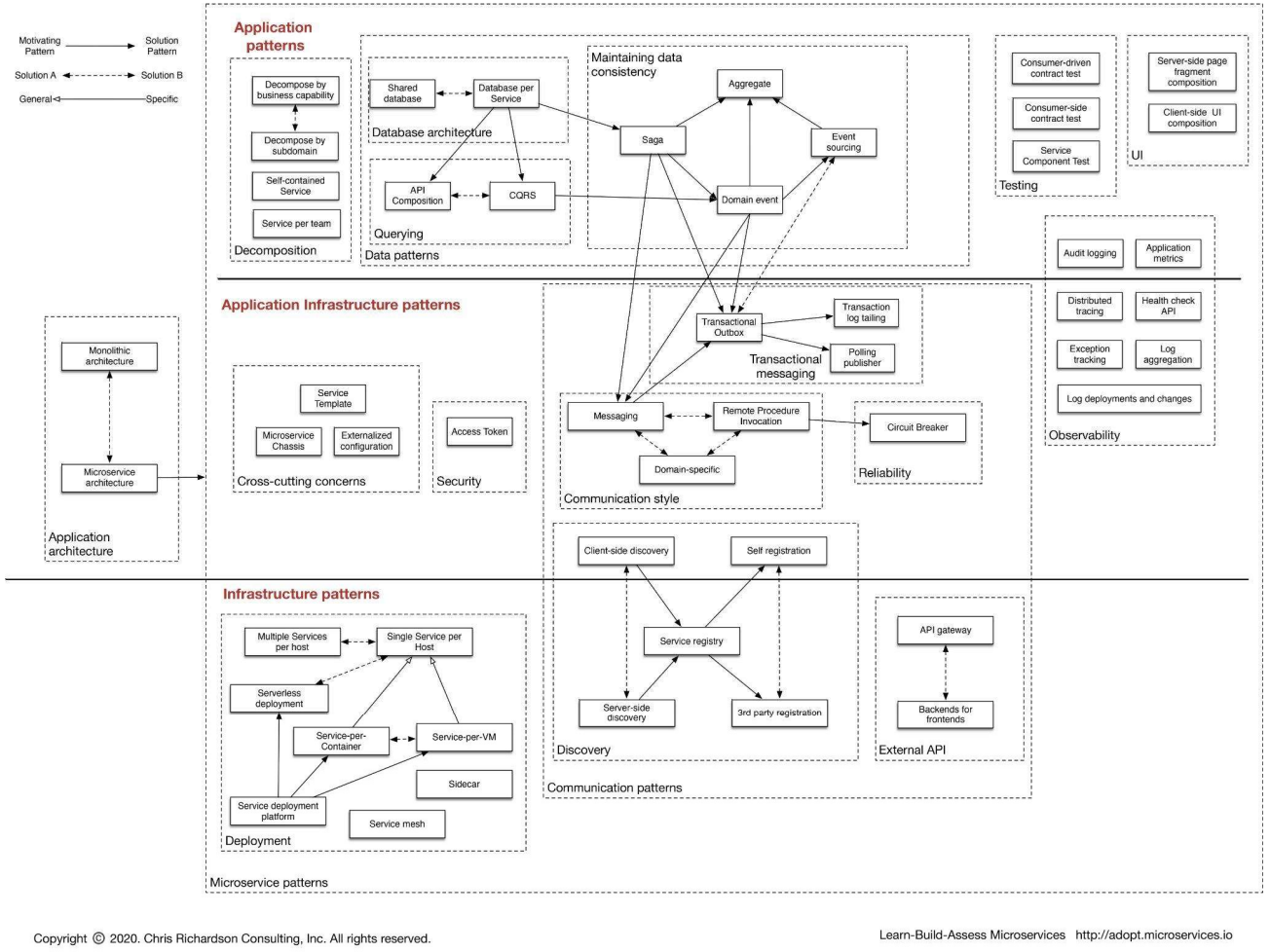
Μια σταδιακή προσέγγιση (δ) έχει χρησιμοποιηθεί από εταιρείες όπως το Netflix, το eBay, η Amazon (Kwiecien, 2019). Μέρη της υπάρχουσας μονολιθικής εφαρμογής ανακατασκευάζονται, νέες μονάδες αναπτύσσονται ως μικροϋπηρεσίες και ενσωματώνονται διαδοχικά.

### 3.2.2 Μεθοδολογίες μετάβασης

Ανεξάρτητα από τη στρατηγική μετάβασης που έχει επιλεγεί, ερωτήματα σχετικά με τον αριθμό, το μέγεθος και την αλληλεπίδραση των μικροϋπηρεσιών προκύπτουν πάντα αργά ή γρήγορα. Από την οπτική γωνία ενός αρχιτέκτονα λογισμικού, η αποσύνθεση σε μικροϋπηρεσίες μπορεί να θεωρηθεί ως το κύριο ζήτημα, καθώς θα καθορίσει τη συνολική επιτυχία μιας ανακατασκευής.

Η έλλειψη ενός καλά καθορισμένου αλγορίθμου που καθοδηγεί τη διαδικασία αποσύνθεσης το καθιστά «κάπως σαν τέχνη», όπως επισημαίνει ο Richardson (2020). Εάν εκτελεστεί λανθασμένα, μπορεί να οδηγήσει σε αρχιτεκτονικές που συνδυάζουν τα μειονεκτήματα και των δύο προτύπων, τόσο του μονολιθικού όσο και των μικροϋπηρεσιών.

Υπάρχουν πηγές που παρέχουν συμβουλές, τεχνικές και μοτίβα μετανάστευσης, αλλά τα εγκεκριμένα επίσημα μοντέλα και τα αυτοματοποιημένα εργαλεία είναι ακόμη λίγα, όπως αποκάλυψε η διενεργηθείσα βιβλιογραφική ανασκόπηση.

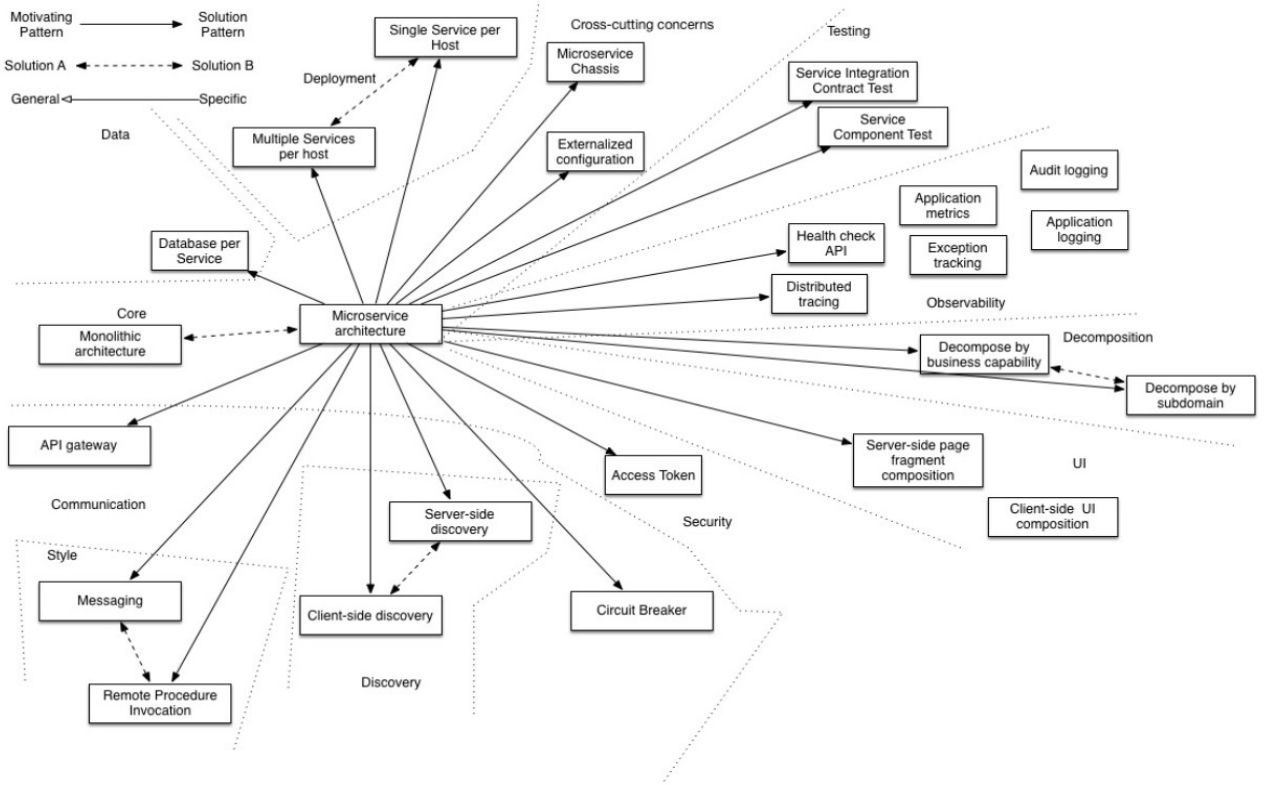



Εικ. 22. Οι απαρχές μιας γλώσσας προτύπων για αρχιτεκτονικές μικροϋπηρεσιών[3]

[3] *Πηγή: Richardson, 2020*

Στις επόμενες παραγράφους, θα συζητηθούν οι βασικές αρχές μιας διαδικασίας αποσύνθεσης, οδηγώντας σε μια εξέταση των διαθέσιμων προσεγγίσεων για την αποσύνθεση σε μικροϋπηρεσίες.

Ωστόσο, ο αρχιτέκτονας λογισμικού θα πρέπει πάντοτε να έχει κατά νου ότι η επιλογή της αρχιτεκτονικής μικροϋπηρεσιών προϋποθέτει την επιλογή μερικών από μια μεγάλη λίστα διαφορετικών μοτίβων προκειμένου να αντιμετωπίσει τις συνέπειες της απόφασής του, όπως φαίνεται στην παρακάτω εικόνα.




Εικ. 23. Μοτίβα για αρχιτεκτονικές μικροϋπηρεσιών [4]

[4] *Πηγή: Richardson, 2020*

### 3.2.3 Ανάπτυξη μεθοδολογιών

3.2.3.1 Κατά Amundsen - ποικίλα κριτήρια διαχωρισμού

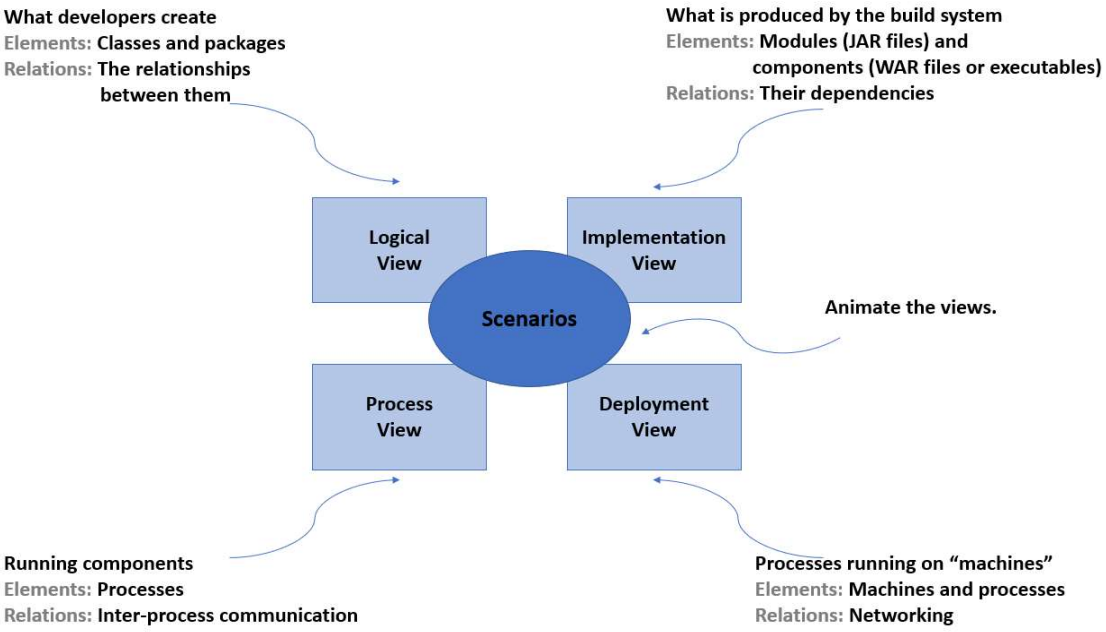
Υπάρχουν διάφοροι τρόποι για να χωρίσει ένας αρχιτέκτονας ένα σύστημα σε μικρότερα μέρη. Ένας τρόπος διαχωρισμού θα μπορούσε να είναι ο κάτωθι (Amundsen et al., 2016):

- Με βάση την *τεχνολογία υλοποίησης*, οι υπολογιστικά βαριές υπηρεσίες που είναι γραμμένες σε C μπορούν να διαχωριστούν από τα chatty συστατικά χρησιμοποιώντας το Node.js.
- Με βάση τη *γεωγραφία*, συμπεριλαμβανομένων συγκεκριμένων νομικών, εμπορικών ή πολιτιστικών απαιτήσεων.

Εκτός από την τεχνολογία υλοποίησης ή τη γεωγραφία, άλλα κριτήρια διαχωρισμού θα μπορούσαν να είναι: το αρχιτεκτονικό στυλ, διάφορες μη λειτουργικές απαιτήσεις, προσωπικές εμπειρίες ή εκπαίδευση.

3.2.3.2 Κατά Richardson - τέσσερις διαφορετικές προβολές

Ο Richardson (2017) χρησιμοποιεί την ιδέα του Kruntchen, ο οποίος προτείνει ένα μοντέλο περιγράφοντας τέσσερις διαφορετικές απόψεις μιας αρχιτεκτονικής λογισμικού: λογική, διεργασιών, φυσική και προβολή ανάπτυξης.




Εικ. 24. Τέσσερις διαφορετικές απόψεις μιας αρχιτεκτονικής λογισμικού: λογική, διεργασιών, φυσική και προβολή ανάπτυξης

- Λογική προβολή (Logical View)

Τα στοιχεία λογισμικού είναι εκείνα τα συστατικά που δημιουργούνται από τους προγραμματιστές. Στις αντικειμενοστρεφείς γλώσσες, αυτά τα στοιχεία είναι οι κλάσεις και τα πακέτα. Οι σχέσεις μεταξύ τους είναι οι σχέσεις μεταξύ των κλάσεων και των πακέτων, συμπεριλαμβανομένης της κληρονομικότητας, των συσχετίσεων και των εξαρτήσεων.

- Προβολή υλοποίησης (Implementation View)

Το παραγόμενο του υπό ανάπτυξη συστήματος. Αυτή η προβολή αποτελείται από λειτουργικές μονάδες, οι οποίες αντιπροσωπεύουν συσκευασμένο κώδικα, και στοιχεία, που είναι εκτελέσιμες ή deployable μονάδες οι οποίες στην συνέχεια αποτελούνται από μία ή περισσότερες μονάδες. Στην Java, μια λειτουργική μονάδα είναι ένα JAR αρχείο και ένα στοιχείο είναι συνήθως ένα WAR αρχείο ή ένα εκτελέσιμο JAR αρχείο. Οι σχέσεις μεταξύ τους περιλαμβάνουν σχέσεις εξάρτησης μεταξύ λειτουργικών μονάδων και σχέσεις σύνθεσης μεταξύ στοιχείων και μονάδων.

- Προβολή διεργασίας (Process View)

Τα στοιχεία κατά το χρόνο εκτέλεσης (runtime). Κάθε στοιχείο είναι μια διαδικασία και οι σχέσεις μεταξύ των διαδικασιών αντιπροσωπεύουν την επικοινωνία μεταξύ των διεργασιών.

- Ανάπτυξη (Deployment)

Αυτή η οπτική καταδεικνύει την συσχέτιση των διαδικασιών σε μηχανές. Τα στοιχεία σε αυτήν την προβολή αποτελούνται από (φυσικές ή εικονικές) μηχανές και τις διεργασίες τους. Οι σχέσεις μεταξύ των μηχανών αντιπροσωπεύουν τη δικτύωση. Αυτή η άποψη περιγράφει επίσης τη σχέση μεταξύ διεργασιών και μηχανών.

#### 3.2.3.3 Οι τρεις απόψεις διαχωρισμού κατά Hölttä-Ottoet et al

Οι Hölttä-Ottoet et al. (2013) διακρίνουν τρεις τύπους απόψεων στη μελέτη τους σχετικά με την αρχιτεκτονική αποσύνθεση στον κατασκευαστικό τομέα, οι οποίες είναι: φυσική (assembly decomposition), λειτουργική (functional) και βασισμένη σε υπηρεσίες (service-based).

- Φυσική αποσύνθεση - Assembly Decomposition

Το όνομα δηλώνει ότι αυτή η άποψη εξετάζει τα φυσικά μέρη ενός συστήματος, όπως τα μέρη ενός αυτοκινήτου. Εάν κάποιος σκοπεύει να επιθεωρήσει το προϊόν ενός ανταγωνιστή, αυτός θα ήταν πιθανώς ο προτιμώμενος τρόπος για να το αποσυνθέσει. Αντίστοιχα, στη Μηχανική Λογισμικού θα εξέταζε κανείς τις μονάδες που μπορούν να αναπτυχθούν. Ο πηγαίος κώδικας σχετίζεται με πακέτα (packages), κλάσεις (classes) και αρχεία (files). Ως εκ τούτου, μια αρχιτεκτονική λογισμικού που βασίζεται σε στοιχεία θα επέτρεπε άμεσα μια κατάτμηση στα στοιχεία της. Ωστόσο, ένα μόνο συστατικό μπορεί να εμπλέκεται σε πολλές λειτουργίες του προϊόντος, σε αντίθεση με την ακόλουθη άποψη.

- Λειτουργική Αποσύνθεση - Functional Decomposition

Αυτός ο τύπος λαμβάνει υπόψη τις λειτουργικές πτυχές του προϊόντος. Μια αποσύνθεση θα απαιτούσε τον προσδιορισμό των λειτουργικών μερών της. Αυτές μπορούν περαιτέρω να χωριστούν σε υπολειτουργίες μέχρι να επιτευχθεί το επιθυμητό επίπεδο ευαισθησίας. Στη Μηχανική Λογισμικού αυτά μπορεί να σχετίζονται με στοιχεία (components), κλάσεις (classes) και μεθόδους (methods). Μεμονωμένα στοιχεία που συμβάλλουν σε πολλές λειτουργίες μπορούν να απομονωθούν ως ξεχωριστές μονάδες ή να αντιγραφούν για κάθε λειτουργία.

- Αποσύνθεση με βάση την υπηρεσία

Η άποψη που βασίζεται στην υπηρεσία θα μπορούσε να βασίζεται στο εγχειρίδιο σέρβις ενός προϊόντος. Σε κάθε κεφάλαιο περιγράφεται μια επιχειρησιακή περιοχή ή μια πτυχή της υπηρεσίας, ενώ οι υπο-ενότητες των κεφαλαίων αναλύουν τις πτυχές σε συνδυασμένες ή μεμονωμένες περαιτέρω ενέργειες. Ομοίως, απεικονίζονται οι περιπτώσεις χρήσης (use cases) και οι απαιτήσεις (functional and non-functional requirements) τις εφαρμογής, δηλαδή οι λειτουργίες που το προϊόν λογισμικού είναι ικανό να εκτελέσει.

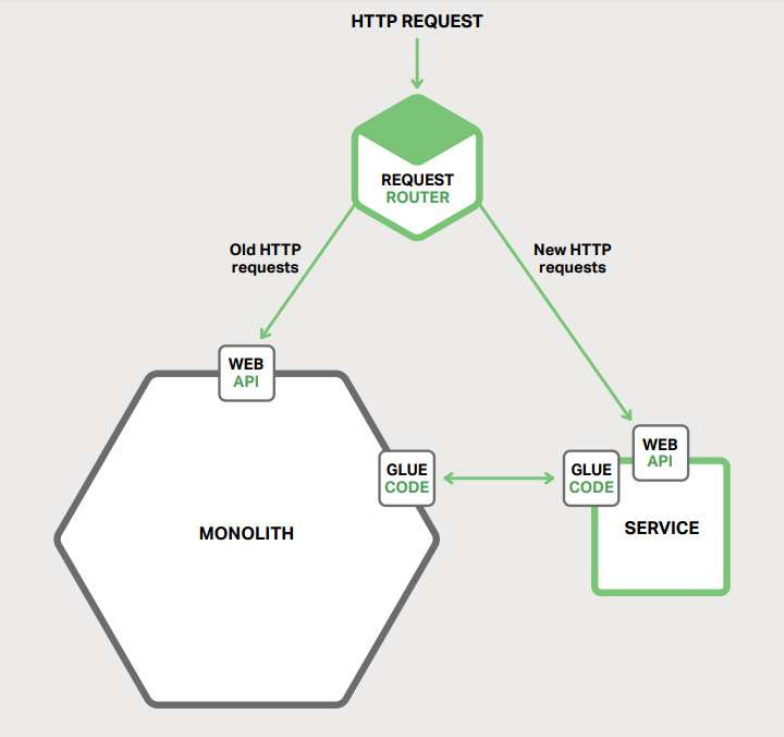
#### 3.2.3.4 Κατά Fowler - στραγγαλιστική εκδοχή

Ο Martin Fowler αναφέρεται σε αυτήν την εφαρμογή στρατηγικής εκσυγχρονισμού ως Application Strangler. Το όνομα προέρχεται από το στραγγαλιστικό κλήμα (ή στραγγαλιστικό σύκο) που βρίσκεται σε τροπικά δάση. Ένα στραγγαλιστικό κλήμα μεγαλώνει γύρω από ένα δέντρο για να φτάσει το φως του ήλιου πάνω από το κουβούκλιο του δάσους. Μερικές φορές, το δέντρο πεθαίνει, αφήνοντας ένα δέντρο σε σχήμα αμπέλου. Ο εκσυγχρονισμός των εφαρμογών ακολουθεί το ίδιο μοτίβο. Αυτό που προτείνεται εν τέλει είναι να δημιουργηθεί μια νέα εφαρμογή που αποτελείται από μικροϋπηρεσίες γύρω από την μονολιθική εφαρμογή, η οποία θα συρρικνωθεί και ίσως, τελικά, πεθάνει (Richardson & Smith, 2016).

##### *3.2.3.4.1 Μεθοδολογία συρρίκνωσης*

- **Διακοπή ανοίγματος τρυπών**

Κάθε νέα απαίτηση στην μονολιθική εφαρμογή προσομοιάζεται με μια τρύπα η υλοποίηση της οποίας απαιτεί την διακοπή του σκαψίματός της. Αυτή η συμβουλή θα πρέπει να ακολουθηθεί όταν έχει καταστεί αδύνατη η συντήρηση μιας μονολιθικής εφαρμογής. Βασική αρχή σε αυτή την προσέγγιση είναι να σταματήσει η διόγκωση του μονόλιθου. Ο νέος κώδικας θα πρέπει να τοποθετηθεί σε μια αυτόνομη μικροϋπηρεσια.




Εικ. 25. Εφαρμογή της νέας λειτουργικότητας ως ξεχωριστή υπηρεσία αντί της προσθήκης νέας ενότητας στο μονόλιθο

Εκτός από τη νέα υπηρεσία και το μονόλιθο παλαιού τύπου, υπάρχουν δύο άλλα νέα στοιχεία.

Το πρώτο είναι ένας δρομολογητής αιτημάτων (request router), ο οποίος χειρίζεται εισερχόμενα αιτήματα (HTTP). Ο δρομολογητής στέλνει τα αιτήματα που αντιστοιχούν στη νέα λειτουργικότητα στη νέα υπηρεσία, ενώ διευθύνει αιτήματα παλαιού τύπου στη μονολιθική εφαρμογή.

Το άλλο στοιχείο είναι ο κωδικός κόλλας (glue code), ο οποίος ενσωματώνει την υπηρεσία με το μονόλιθο. Μια υπηρεσία σπάνια μπορεί να λειτουργήσει μεμονωμένα και συχνά χρειάζεται πρόσβαση σε δεδομένα που ανήκουν στον μονόλιθο.

Ο κωδικός κόλλας, ο οποίος βρίσκεται είτε στο μονόλιθο, είτε στην μικροϋπηρεσία, είτε και στα δύο, είναι υπεύθυνος για την ενοποίηση δεδομένων. Η υπηρεσία χρησιμοποιεί τον κώδικα κόλλας για την ανάγνωση και εγγραφή δεδομένων που ανήκουν στον μονόλιθο.

Υπάρχουν τρεις στρατηγικές που μπορεί να χρησιμοποιήσει μια υπηρεσία για να αποκτήσει πρόσβαση στα δεδομένα του μονόλιθου:

1. Κλήση ενός απομακρυσμένου API που παρέχεται από τον μονόλιθο
2. Απευθείας πρόσβαση στη βάση δεδομένων του μονόλιθου

3. Η μικροϋπηρεσία διατηρεί το δικό του αντίγραφο των δεδομένων, το οποίο είναι συγχρονισμένο με τη βάση δεδομένων του μονόλιθου.

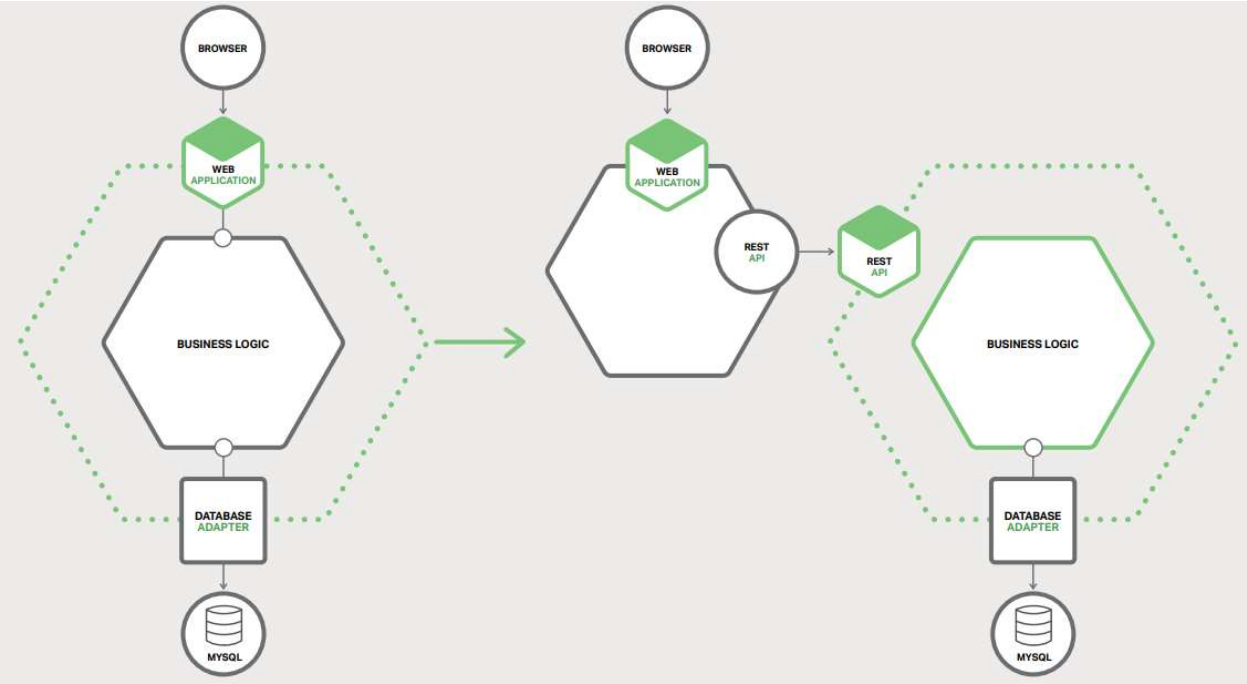
Ο κωδικός κόλλας ονομάζεται μερικές φορές στρώμα κατά της διαφθοράς (anti-corruption layer). Αυτό συμβαίνει επειδή ο κωδικός κόλλας αποτρέπει τη μόλυνση της υπηρεσίας, η οποία έχει το δικό της νέο μοντέλο τομέα (domain model) από το μοντέλο τομέα του κληροδοτημένου μονόλιθου.

- **Διαχωρισμός του Front-end και του Back-end**

Μια στρατηγική που συρρικνώνει τη μονολιθική εφαρμογή είναι να διαχωριστεί το επίπεδο παρουσίασης (presentation layer) από τα επίπεδα επιχειρηματικής λογικής (business layer) και πρόσβασης δεδομένων (data access layer). Μια τυπική εταιρική εφαρμογή αποτελείται από τουλάχιστον τρεις διαφορετικούς τύπους εξαρτημάτων:

- ❖ Επίπεδο παρουσίασης (presentation layer): Πρόκειται για στοιχεία που χειρίζονται HTTP αιτήματα και υλοποιούν είτε (REST) API διεπαφή ή διεπαφή χρήστη που βασίζεται σε HTML. Σε μια εφαρμογή που έχει εξελιγμένη διεπαφή χρήστη, το επίπεδο παρουσίασης είναι συχνά ένα σημαντικό σώμα κώδικα.
- ❖ Επιχειρησιακό επίπεδο λογικής (business layer): Πρόκειται για στοιχεία που αποτελούν τον πυρήνα της εφαρμογής και υλοποιούν τους επιχειρηματικούς κανόνες.
- ❖ Επίπεδο πρόσβασης δεδομένων (data access layer): Πρόκειται για στοιχεία που έχουν πρόσβαση σε στοιχεία υποδομής, όπως βάσεις δεδομένων ή μεσίτες μηνυμάτων (message brokers).

Συνήθως υπάρχει καθαρός διαχωρισμός μεταξύ της λογικής παρουσίασης στη μία πλευρά και της επιχειρηματικής και λογικής πρόσβασης δεδομένων από την άλλη. Αυτός ο διαχωρισμός αποτελεί και την φυσική ραφή κατά μήκος της οποίας μπορεί να χωριστεί ο μονόλιθος σε δύο μικρότερες εφαρμογές. Μία εφαρμογή περιέχει το επίπεδο παρουσίασης. Η άλλη εφαρμογή περιέχει τη λογική της επιχείρησης και της πρόσβασης δεδομένων. Μετά τη διάσπαση, η εφαρμογή της λογικής παρουσίασης πραγματοποιεί απομακρυσμένες κλήσεις στην εφαρμογή της επιχειρηματικής λογικής.




Εικ. 26. Διαχωρισμός μονολιθικής εφαρμογής σε περισσότερα layers

- **Εξαγωγή υπηρεσιών**

Η τρίτη στρατηγική ανακατασκευής είναι να μετατραπούν οι υπάρχουσες μονάδες εντός του μονόλιθου σε αυτόνομες μικροϋπηρεσίες. Κάθε φορά που ολοκληρώνεται η εξαγωγή μιας μονάδας και μετατρέπεται σε μικροϋπηρεσία, ο μονόλιθος συρρικνώνεται.

Μόλις ολοκληρωθεί η μετατροπή αρκετών μονάδων, ο μονόλιθος θα πάψει να είναι πρόβλημα. Είτε εξαφανίζεται εντελώς είτε γίνεται αρκετά μικρός ώστε να είναι απλώς μια άλλη υπηρεσία.

*3.2.3.4.2 Προτεραιοποίηση ενοτήτων που θα μετατραπούν σε υπηρεσίες*

Μια μεγάλη, πολύπλοκη μονολιθική εφαρμογή αποτελείται από δεκάδες ή εκατοντάδες ενότητες, οι οποίες είναι υποψήφιες προς εξαγωγή. Η προτεραιοποίηση μετατροπής του μετασχηματισμού των μονάδων είναι μια συχνή πρόκληση.

Μια καλή προσέγγιση είναι η έναρξη με μερικές ενότητες που είναι εύκολο να εξαχθούν. Αυτή η προσέγγιση προσφέρει εμπειρία με τις μικροϋπηρεσίες γενικά και τη διαδικασία εξαγωγής ειδικότερα.

Μετά από αυτό, θα πρέπει να εξαχθούν εκείνες οι ενότητες που προσφέρουν το μεγαλύτερο όφελος. Η μετατροπή μιας ενότητας σε υπηρεσία είναι συνήθως χρονοβόρα. Απαιτεί πρωτίστως ταξινόμηση των ενοτήτων που είναι ένα σημαντικό όφελος από μόνο του.

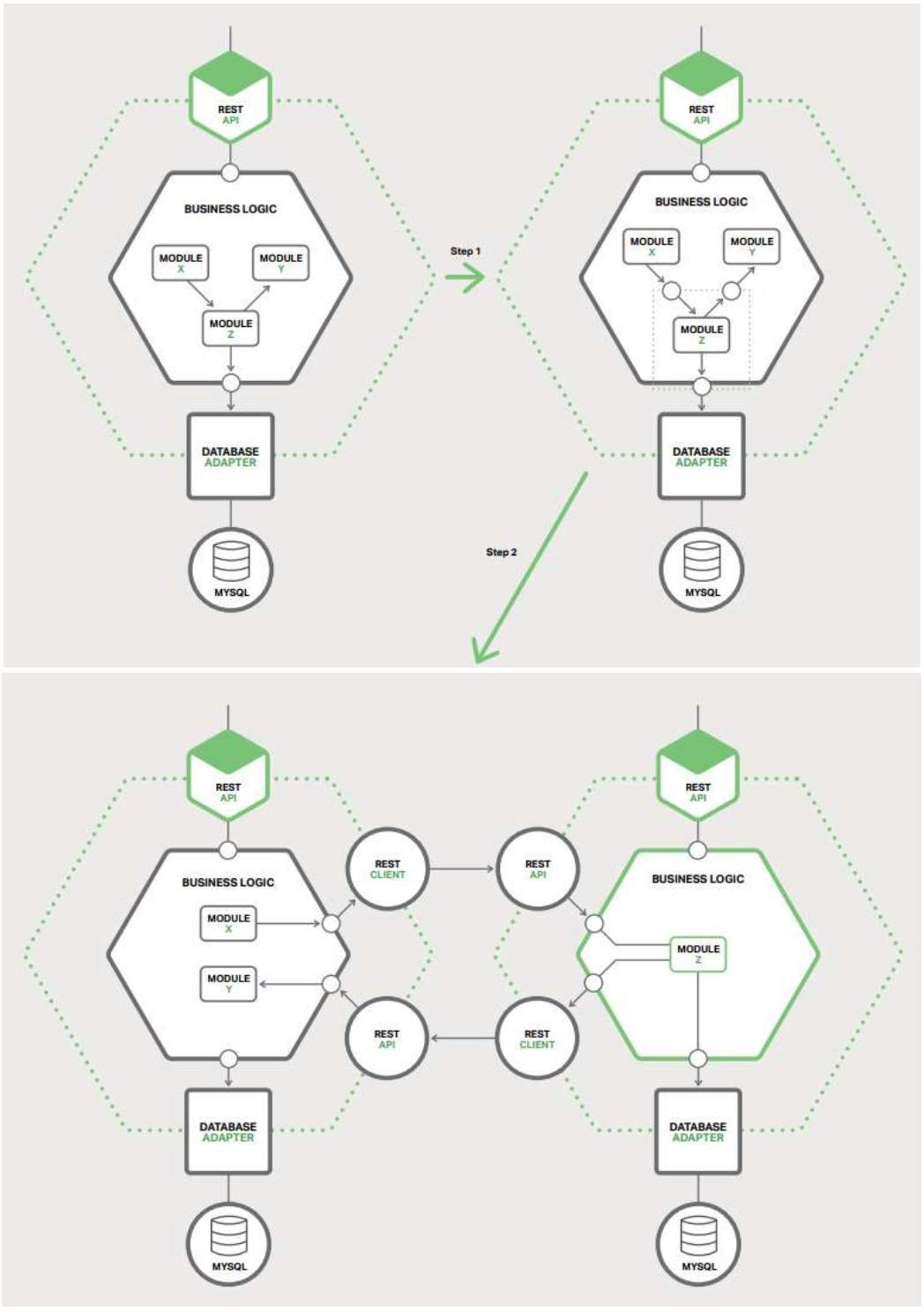
Επιπλέον, θα πρέπει κανείς να λάβει υπόψιν ότι είναι συνήθως ωφέλιμο να εξαχθούν πρώτα οι ενότητες που αλλάζουν συχνά.

Είναι επίσης ωφέλιμο να εξαχθούν πρωτίστως οι ενότητες που έχουν σημαντικές απαιτήσεις σε πόρους και είναι διαφορετικές από αυτές του υπόλοιπου μονόλιθου. Είναι χρήσιμο, για παράδειγμα, να γίνει πρώτα μικροϋπηρεσία μια ενότητα που έχει μια βάση δεδομένων στη μνήμη. Ομοίως, μπορεί να αξίζει τον κόπο να εξαγχθούν πρωτίστως ενότητες που υλοποιούν υπολογιστικά ακριβούς αλγόριθμους και μπορούν στη συνέχεια να αναπτυχθούν σε κεντρικούς υπολογιστές με πολλές CPU. Μετατρέποντας ενότητες με ιδιαίτερες απαιτήσεις πόρων σε υπηρεσίες, γίνεται ταυτόχρονα και η ίδια η μονολιθική εφαρμογή πολύ πιο εύκολη και λιγότερο δαπανηρή σε κλιμάκωση.

Κατά τον υπολογισμό εξαγωγής είναι χρήσιμο να αναζητηθούν αυτόνομες μονάδες με διακριτά όρια (γνωστές και ως ραφές). Αυτή η προσέγγιση καθιστά ευκολότερη και φθηνότερη τη μετατροπή των μονάδων σε υπηρεσίες. Ένα παράδειγμα ενός τέτοιου ορίου είναι μια ενότητα που επικοινωνεί με την υπόλοιπη εφαρμογή μέσω ασύγχρονων μηνυμάτων.

*3.2.3.4.3 Τρόπος εξαγωγής της υπηρεσίας*

Το πρώτο βήμα της εξαγωγής μιας ενότητας είναι ο ορισμός μιας διεπαφής μεταξύ της μικροϋπηρεσίας και του μονόλιθου. Πιθανότατα αυτό θα είναι ένα αμφίδρομο API, αφού ο μονόλιθος θα χρειαστεί δεδομένα που ανήκουν στην μικροϋπηρεσία και αντίστροφα. Είναι συχνά δύσκολο να εφαρμοστεί ένα τέτοιο API λόγω των μπερδεμένων εξαρτήσεων και των λεπτών μοτίβων αλληλεπίδρασης μεταξύ των ενοτήτων που θα διαχωριστούν με όλη την υπόλοιπη μονολιθική εφαρμογή. Συχνά θα χρειαστούν σημαντικές αλλαγές στον κώδικα για να σπάσουν αυτές οι εξαρτήσεις. Το παρακάτω σχήμα δείχνει μια τέτοια ανακατασκευή.




Εικ. 27. Μετατροπή ενότητας μονόλιθου σε μικροϋπηρεσία

3.2.3.5 Κατά Microsoft

Οποιαδήποτε στρατηγική μετάβασης και να ακολουθηθεί θα πρέπει να δίνεται η δυνατότητα στις ομάδες ανάπτυξης να αναδιαμορφώνουν σταδιακά την εφαρμογή σε μικρότερες υπηρεσίες, παρέχοντας παράλληλα τη συνέχιση των υπολοίπων υπηρεσιών στους τελικούς χρήστες (Microsoft; 2021, September 11).

Εδώ είναι η γενική προσέγγιση περιγράφεται με τα ακόλουθα βήματα:

1. Διακοπή προσθήκης λειτουργικότητας στο μονόλιθο.
2. Διαχωρισμός του front από το back-end.
3. Αποσύνδεση και αποσύνθεση του μονόλιθου σε μια σειρά μικροϋπηρεσιών.

Τα παραπάνω βήματα της γενικής προσέγγισης είναι δυνατόν να ολοκληρωθούν με διάφορες προσεγγίσεις. Παρακάτω αναλύονται οι διαφορετικές προσεγγίσεις:

- **Εφαρμογή του Domain Driven Design**

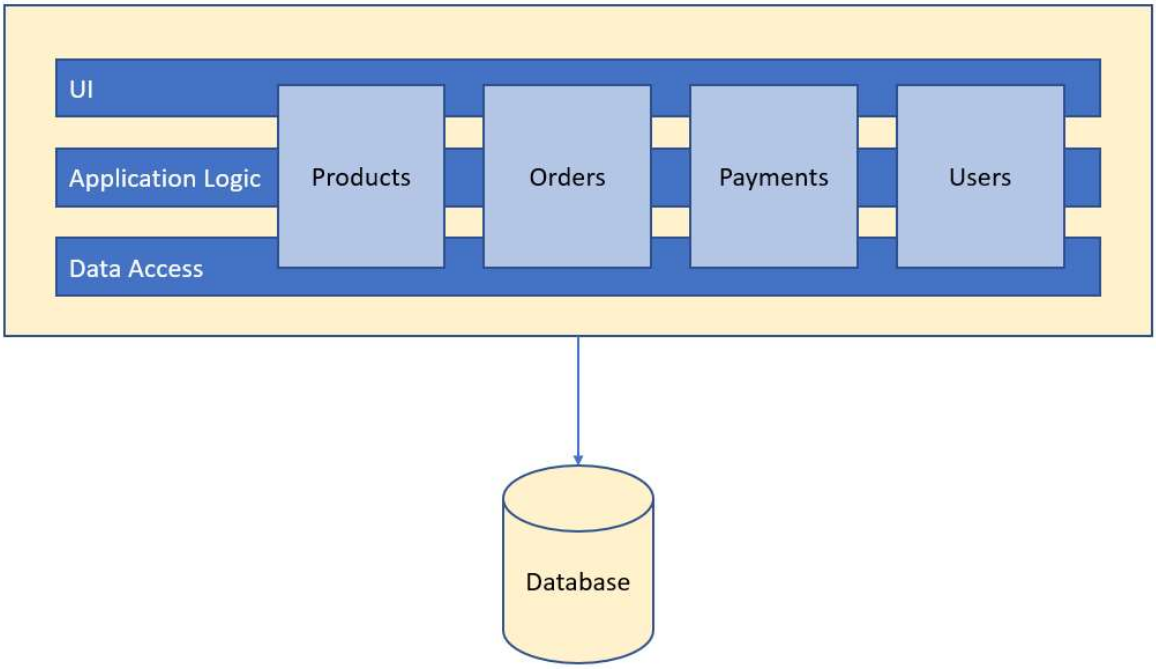
Για να διευκολυνθεί αυτή η αποσύνθεση, μια βιώσιμη προσέγγιση ανάπτυξης λογισμικού είναι η εφαρμογή των αρχών της σχεδίασης βάσει τομέα (DDD).

Το Domain Driven Design (DDD) είναι μια προσέγγιση ανάπτυξης λογισμικού που εισήχθη για πρώτη φορά από τον Eric Evans. Το DDD απαιτεί καλή κατανόηση του τομέα για τον οποίο θα γραφτεί η εφαρμογή. Οι απαραίτητες γνώσεις τομέα για τη δημιουργία της εφαρμογής βρίσκονται στα άτομα που την κατανοούν — τους ειδικούς τομέα.

Η προσέγγιση DDD μπορεί να εφαρμοστεί αναδρομικά σε μια υπάρχουσα εφαρμογή, ως ένας τρόπος για να ξεκινήσει η αποσύνθεση της εφαρμογής.

A. Θα πρέπει να υπάρχει ένα κοινό λεξιλόγιο που μοιράζονται και κατανοούν όλα τα ενδιαφερόμενα μέρη (stakeholders).
B. Έπειτα θα γίνει ο προσδιορισμός των ενοτήτων της μονολιθικής εφαρμογής που είναι υποψήφια προς αποσύνθεση και θα ονοματιστούν βάσει του κοινού λεξιλογίου.
C. Στη συνέχεια θα οριστούν τα μοντέλα τομέα (domain models) της μονολιθικής εφαρμογής. Το μοντέλο τομέα είναι ένα αφηρημένο μοντέλο του επιχειρηματικού τομέα (business domain).
D. Τέλος θα οριστούν τα σαφώς καθορισμένα περιβάλλοντα για τα μοντέλα. Ένα περιορισμένο περιβάλλον είναι το όριο εντός ενός τομέα όπου εφαρμόζεται ένα συγκεκριμένο μοντέλο τομέα. Απαιτούνται ρητά όρια με σαφώς καθορισμένα μοντέλα και ευθύνες.

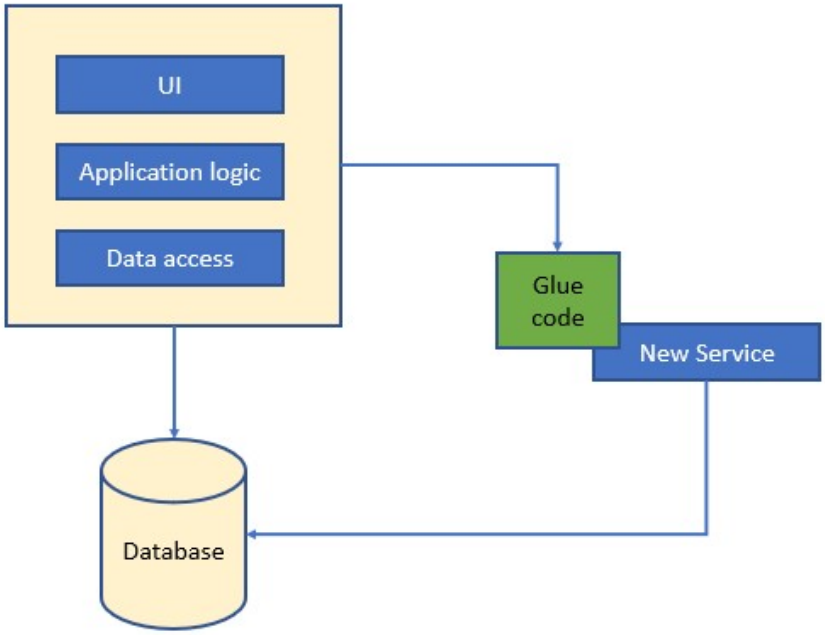
Τα οριοθετημένα περιβάλλοντα που προσδιορίζονται στο βήμα 4 είναι υποψήφια για αναδιαμόρφωση σε μικρότερες μικροϋπηρεσίες. Το παρακάτω διάγραμμα δείχνει τον υπάρχοντα μονόλιθο με τα οριοθετημένα περιβάλλοντα να επικαλύπτονται:




Εικ. 28. Οριοθετημένα περιβάλλοντα εντός μιας μονολιθικής εφαρμογής

- **Χρήση κώδικα κόλλας (στρώμα κατά της διαφθοράς)**

Ενώ εκτελείται η παραπάνω ερευνητική εργασία για την απογραφή της μονολιθικής εφαρμογής, μπορεί να προστεθεί νέα λειτουργικότητα εφαρμόζοντας τις αρχές του DDD ως ξεχωριστές μικροϋπηρεσίες. Ο "Κώδικας κόλλας" επιτρέπει στη μονολιθική εφαρμογή να διαμεσολαβεί κλήσεις στη νέα υπηρεσία για να αποκτήσει νέα λειτουργικότητα.




Εικ. 29. Κώδικας κόλλας στην μικροϋπηρεσία

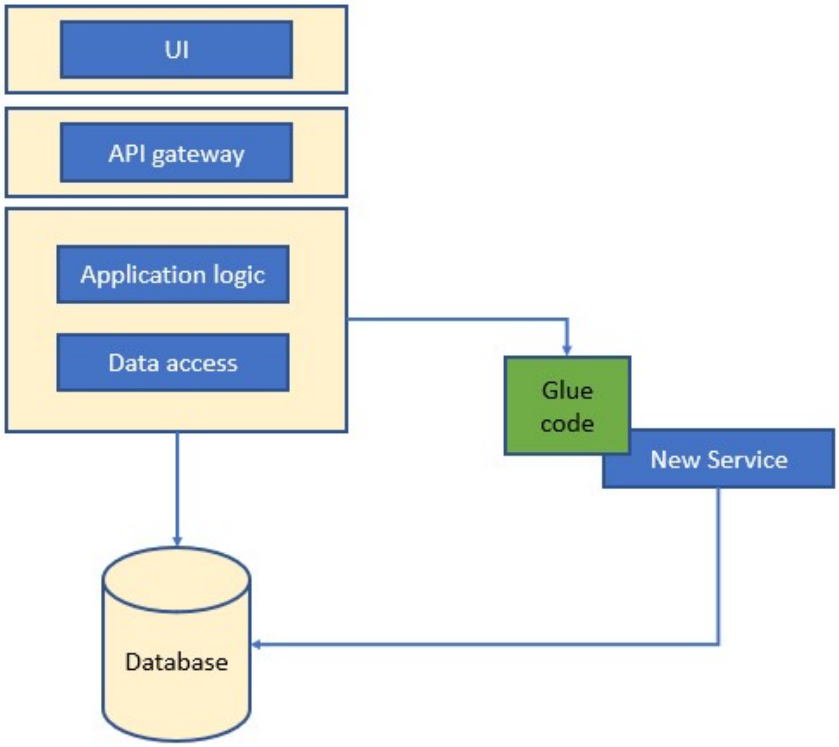
Ο κωδικός κόλλας (μοτίβο προσαρμογέα) λειτουργεί αποτελεσματικά ως στρώμα κατά της διαφθοράς, διασφαλίζοντας ότι η νέα υπηρεσία δεν μολύνεται από μοντέλα δεδομένων που απαιτούνται από τη μονολιθική εφαρμογή. Ο κωδικός κόλλας βοηθά στη μεσολάβηση των αλληλεπιδράσεων μεταξύ των δύο και διασφαλίζει ότι διαβιβάζονται μόνο τα δεδομένα που απαιτούνται από τη νέα υπηρεσία για να επιτραπεί η συμβατότητα.

Μέσω της διαδικασίας ανακατασκευής, οι ομάδες μπορούν να απογράφουν τη μονολιθική εφαρμογή για να εντοπίσουν υποψήφιες μονάδες για ανακατασκευή σε μικροϋπηρεσίες, ενώ παράλληλα καθιερώνουν τη νέα λειτουργικότητα με νέες υπηρεσίες.

- **Δημιουργία ενός επιπέδου παρουσίασης**

Το επόμενο βήμα είναι να διαχωρίστεί το επίπεδο παρουσίασης (presentation layer ή front-end) από το επίπεδο υποστήριξης (back-end). Σε μια παραδοσιακή n-tier εφαρμογή, το επιχειρηματικό επίπεδο (business layer) τείνει να περιέχει τα στοιχεία που είναι ο πυρήνας της εφαρμογής και παρέχουν τη λογική τομέα (domain logic). Αυτά τα χονδρόκοκκα APIs αλληλεπιδρούν με το επίπεδο πρόσβασης δεδομένων (data access layer) για την ανάκτηση δεδομένων μέσα από μια βάση. Τα APIs δημιουργούν ένα φυσικό όριο μεταξύ του επιπέδου παρουσίασης και της επιχειρηματικής λογικής και βοηθούν στην αποσύνθεση του επιπέδου παρουσίασης σε μια ξεχωριστή μονάδα.

Το παρακάτω διάγραμμα δείχνει το επίπεδο παρουσίασης (UI) χωρισμένο από τα επίπεδα λογικής και πρόσβασης δεδομένων.




Εικ. 30. Διαχωρισμός επιπέδου παρουσίασης από τις υπόλοιπες μονάδες

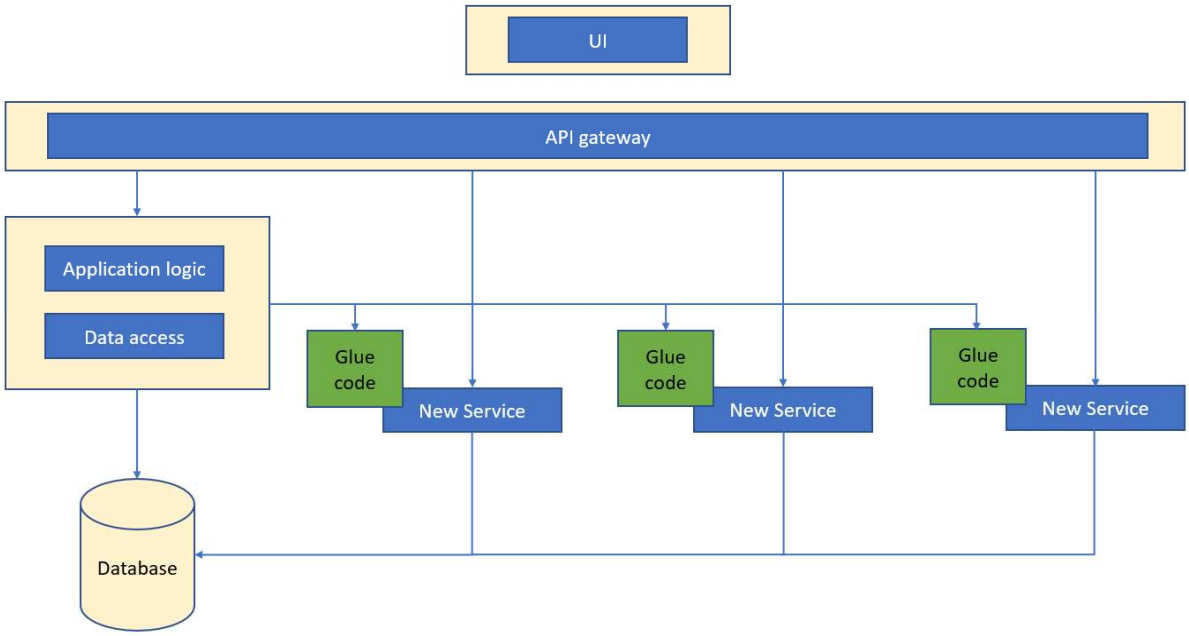
Αυτό το διάγραμμα εισάγει επίσης ένα άλλο επίπεδο, την πύλη API, που βρίσκεται μεταξύ του επιπέδου παρουσίασης και της επιχειρηματικής λογικής της εφαρμογής. Η πύλη API είναι ένα επίπεδο πρόσοψης που παρέχει μια συνεπή και ομοιόμορφη διεπαφή για την αλληλεπίδραση του επιπέδου παρουσίασης, ενώ επιτρέπει στις μεταγενέστερες υπηρεσίες να εξελίσσονται ανεξάρτητα, χωρίς να επηρεάζεται η εφαρμογή. Το API Gateway μπορεί να χρησιμοποιεί μια τεχνολογία όπως το Azure API Management και επιτρέπει στην εφαρμογή να αλληλεπιδρά με *RESTful* τρόπο.

Το επίπεδο παρουσίασης μπορεί να αναπτυχθεί σε οποιαδήποτε γλώσσα ή πλαίσιο στο οποίο έχει εξειδίκευση η ομάδα, όπως μια εφαρμογή μιας σελίδας (single page application) ή μια MVC εφαρμογή. Αυτές οι εφαρμογές αλληλεπιδρούν με τις μικροϋπηρεσίες μέσω της πύλης, χρησιμοποιώντας τυπικές HTTP κλήσεις.

- **Σταδιακή απόσυρση του μονόλιθου**

Σε αυτό το στάδιο, η ομάδα μπορεί να αρχίσει να ξεφλουδίζει τη μονολιθική εφαρμογή και να εξάγει σιγά-σιγά τις υπηρεσίες που έχουν δημιουργηθεί από τα οριοθετημένα περιβάλλοντα στο δικό της σύνολο μικροϋπηρεσιών. Οι μικροϋπηρεσίες μπορούν να εκθέσουν μια RESTful

διεπαφή για την αλληλεπίδραση του επιπέδου εφαρμογής, μέσω της API πύλης, με τον κώδικα κόλλας για επικοινωνία με το μονόλιθο σε συγκεκριμένες περιπτώσεις.




Εικ. 31. API πύλη και κώδικας κόλλας

Μετά την σταδιακή αφαίρεση λειτουργικότητας και κώδικα από τον μονόλιθο. τελικά θα συρρικνωθεί σε μέγεθος και πολυπλοκότητα, σε σημείο που να μην υφίσταται πλέον. Σε αυτό το στάδιο, το στρώμα κατά της διαφθοράς (κωδικός κόλλας) μπορεί να αφαιρεθεί με ασφάλεια.

- **Επόμενα βήματα**

Όταν η εφαρμογή έχει αποσυντεθεί σε συστατικές μικροϋπηρεσίες, καθίσταται δυνατή η χρήση σύγχρονων εργαλείων ενορχήστρωσης όπως το Azure DevOps για τη διαχείριση του κύκλου ζωής κάθε υπηρεσίας. Επίσης μπορούν να εφαρμοστούν CI/CD πρακτικές συνεχούς ολοκλήρωσης και συνεχούς παράδοσης ή ανάπτυξης.

# Κεφ.4 Γεωχωρικές τεχνολογίες

Οι Γεωχωρικές Τεχνολογίες (Geospatial Technologies), είναι ο κλάδος των επιστημών που αναφέρεται στη συλλογή, επεξεργασία, διαχείριση, απεικόνιση και ερμηνεία πληροφοριών με γεωγραφική αναφορά που αφορούν το φυσικό και ανθρωπογενές περιβάλλον.

Η “Γεωπληροφορική” (Geoinformatics), είναι η επιστήμη η οποία αξιοποιεί τη γεωγραφική πληροφορία και τις σύγχρονες τεχνολογίες που αναπτύσσει ο τομέας της πληροφορικής, έτσι ώστε να συγκεντρώσει, να αποθηκεύσει, να ενημερώσει, να διαχειριστεί, να επεξεργαστεί, να

αναλύσει, να οπτικοποιήσει και να παρουσιάσει προτάσεις σε προβλήματα που αφορούν τις επιστήμες του χώρου και των συναφών κλάδων (Reed et al., 2003).

Ορισμένα από τα βασικά εργαλεία που χρησιμοποιούνται στη Γεωπληροφορική είναι τα Συστήματα Γεωγραφικών Πληροφοριών, γνωστά και ως GIS (Geographic Information Systems), η Τεχνολογία Δορυφορικού Εντοπισμού Θέσης, γνωστή και ως GPS (Global Position System), οι Τεχνολογίες Ανάλυσης και Επεξεργασίας Αεροφωτογραφιών και Δορυφορικών Εικόνων, η τηλεμετρία (telemetry) και η τηλεπισκόπηση/τηλεανίχνευση (remote sensing). Ο συνδυασμός των προαναφερθεισών τεχνολογιών και όχι μόνο, καταφέρνει να παρουσιάσει ένα ολοκληρωμένο σύνολο χωρικής και περιγραφικής πληροφορίας, αλλά παρέχει την «γεωπληροφορία», με τη χρήση της οποίας δύναται να (McClain, 2022; Reed et al., 2003):

- εκτελεστούν χωρικά ερωτήματα,
- αναλυθούν γεωχωρικά δεδομένα,
- δημιουργηθούν χάρτες και μοντέλα και τέλος
- να ληφθούν καλύτερες αποφάσεις και να επιλεχθούν καλύτερες λύσεις σε γεωχωρικά προβλήματα.

Ο συνδυασμός των παραπάνω εργαλείων μπορεί να δημιουργήσει ένα συνδυαστικό εργαλείο, το οποίο επιτρέπει στους χρήστες να αποτυπώσουν μια περίληψη του πραγματικού κόσμου, να εκτελέσουν διαδραστικές ερωτήσεις χωρικού ή περιγραφικού χαρακτήρα (αναζητήσεις δημιουργούμενες από τον χρήστη), να αναλύσουν τα χωρικά δεδομένα (spatial data), να τα προσαρμόσουν και να τα αποδώσουν σε αναλογικά μέσα (εκτυπώσεις χαρτών και διαγραμμάτων) ή σε ψηφιακά μέσα (αρχεία χωρικών δεδομένων, διαδραστικοί χάρτες στο διαδίκτυο).

Η χαρακτηριστική δυνατότητα που παρέχει η Γεωπληροφορική (σύνδεση χωρική με περιγραφική πληροφορία), την καθιστά ως ένα μοναδικό εργαλείο συλλογής, επεξεργασίας και ανάλυσης δεδομένων, εισάγοντας την και σε επιχειρηματικούς τομείς όπως, το κτηματολόγιο, η πολεοδομία, εφαρμογές διαχείρισης υδάτων και δικτύων, χωροθέτησεις λειτουργιών, κινητής τηλεφωνίας, αναλύσεις οικονομικών μεγεθών και δημογραφικών δεδομένων, στρατιωτικές εφαρμογές κ.α.

## 4.1 Γεωχωρικές εφαρμογές

Ο GeoServer είναι ο πιο ευρέως χρησιμοποιούμενος και αναπτυγμένος γεωχωρικός διακομιστής ανοιχτού κώδικα στον κόσμο. Επιτρέπει τη δημοσίευση, τον μετασχηματισμό και την επεξεργασία γεωχωρικών δεδομένων από έναν μεγάλο αριθμό μορφών, μέσω ενός εκτεταμένου αριθμού υπηρεσιών που απευθύνονται στους χρήστες. Εκτός από αυτές τις out-of-

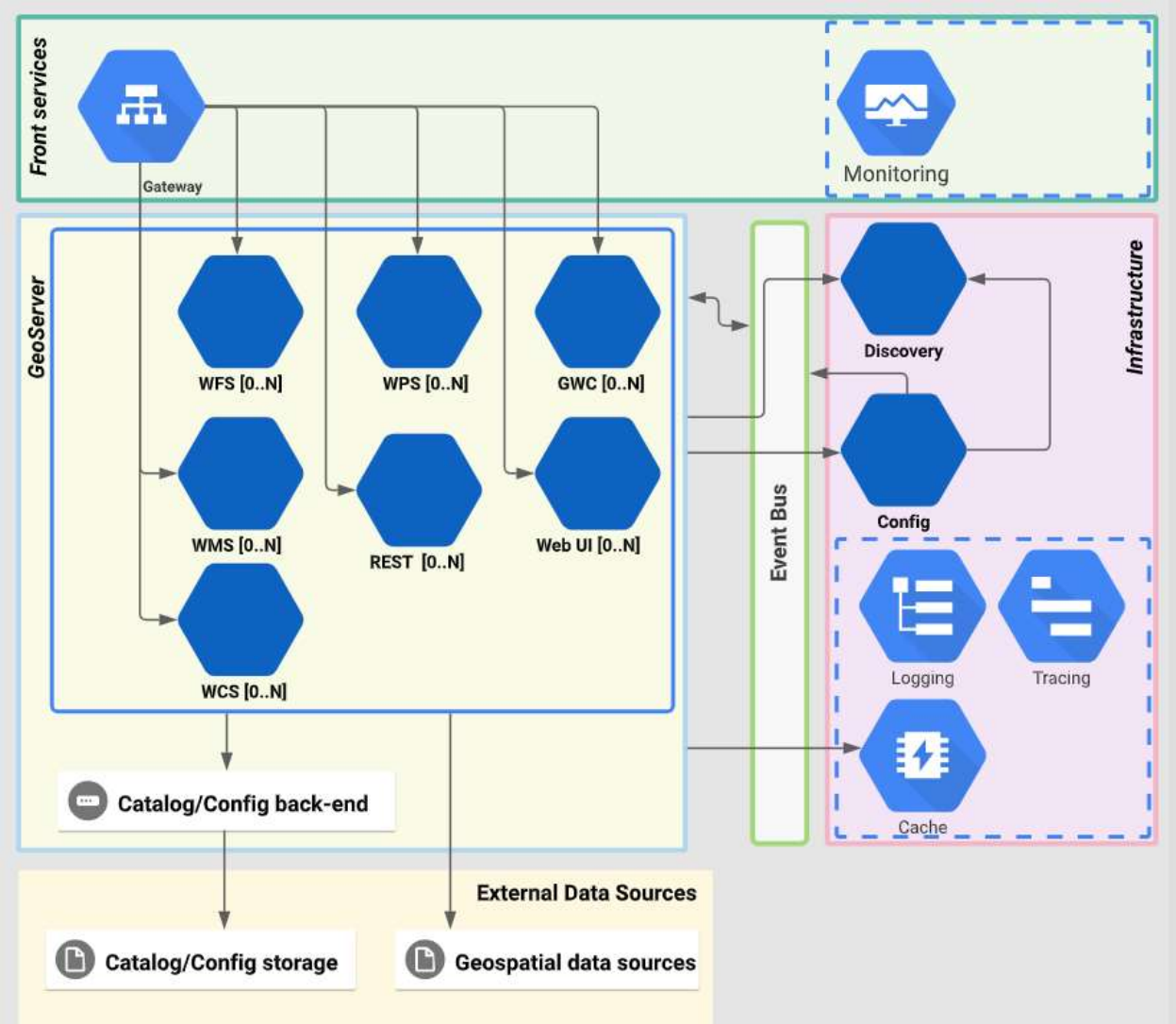
the-box δυνατότητες, μετράει επίσης περισσότερες από 40 υποστηριζόμενες επεκτάσεις, συν έναν μεγαλύτερο αριθμό πειραματικών επεκτάσεων (τα λεγόμενα "community modules").

Όντας μια μονολιθική εφαρμογή, η εγκατάσταση και η διαμόρφωση του GeoServer είναι πολύ εύκολη, καθώς όλα τα διαφορετικά εξαρτήματά του συνδυάζονται, προσφέροντας μια απλή, συνεκτική προσέγγιση στην παράδοση λογισμικού.

Οι υπεύθυνοι για την ανάπτυξη και τη συντήρηση του GeoServer αντιμετωπίζουν μερικές πολύ συνηθισμένες προκλήσεις όσον αφορά τη διαστασιολόγηση, τη διαμόρφωση, την παρακολούθηση της υγείας του συστήματος και τις διορθωτικές ενέργειες. Δυσκολίες που αυξάνονται καθώς πρέπει να εξασφαλιστεί μια ορισμένη ικανότητα διαχείρισης του φορτίου των αιτημάτων, της διαθεσιμότητας υπηρεσιών και της συνολικής απόδοσης.

Προκειμένου να ξεπεράσουν πολλά από τα προβλήματα/προκλήσεις που δημιουργούσε η μονολιθική υλοποίηση του GeoServer αναπτύχθηκε το έργο Cloud Native GeoServer. Ο Cloud Native GeoServer είναι κατασκευασμένος με στοιχεία λογισμικού GeoServer. Η cloud λύση (Cloud Native GeoServer, 2021) είναι ο GeoServer προσφερόμενος για χρήση στο cloud μέσω μικροϋπηρεσιών που είναι συνδεδεμένες σε docker. Η cloud υλοποίηση του γεωχωρικού διακομιστή διαχωρίζει τις γεωχωρικές υπηρεσίες και τις προσφορές API του GeoServer σε μεμονωμένα αναπτυσσόμενα στοιχεία μιας αρχιτεκτονικής που βασίζεται σε μικροϋπηρεσίες.

Τα εξαρτήματα προσαρμόζονται και/ή επεκτείνονται σε μια λειτουργική αποσύνθεση ανάλογα με την επιχειρηματική ικανότητα. Το αποτέλεσμα είναι ότι κάθε υπηρεσία (OWS υπηρεσία: Open Web Service) είναι διαθέσιμη ως αυτόνομη, μεμονωμένη και επεκτάσιμη μικροϋπηρεσία.




Εικ. 32. Αρχιτεκτονική Cloud Native Geoserver[5]

Ο Cloud Native GeoServer είναι ένα "αδελφό" έργο με τον GeoServer, που στοχεύει οργανισμούς και διαχειριστές συστημάτων αναζητώντας έναν σταθερό, καθορισμένο τρόπο παροχής δυνατοτήτων GeoServer σε ένα δυναμικό περιβάλλον, όπου κάθε επιχειρηματική ικανότητα μπορεί να ενεργοποιηθεί, να διαμορφωθεί, να διαστασιολογηθεί και να αναπτυχθεί ανεξάρτητα.

Η cloud λύση δεν αποτελεί επανεγγραφή του GeoServer. Αντίθετα, βασίζεται στα υπάρχοντα στοιχεία λογισμικού GeoServer, προσαρμόζοντάς τα ή/και επεκτείνοντάς τα, τροφοδοτώντας

[5] Cloud Native GeoServer, 2021

και συνεισφέροντας στην κύρια ανάπτυξη του GeoServer. Οι παραδοσιακές εγκαταστάσεις GeoServer εξακολουθούν να συνιστώνται για απλές λύσεις ανάπτυξης λογισμικού.

Αυτό το έργο επαναδιατυπώνει τον τρόπο με τον οποίο οι επιμέρους δυνατότητες συνδυάζονται, σε μια προσπάθεια να επιτευχθεί η λειτουργική αποσύνθεση από την επιχειρηματική λειτουργικότητα, που σημαίνει ότι κάθε υπηρεσία, όπως για παράδειγμα η διεπαφή ιστού, το REST API και πιθανώς άλλα στοιχεία όπως το υποσύστημα *Κατάλογος* και *Διαμόρφωση*, γίνονται αυτόνομες, μεμονωμένα αναπτυσσόμενες και επεκτάσιμες μικροϋπηρεσίες, χαλαρά δια-συνδεδεμένες μέσω μιας αρχιτεκτονικής που βασίζεται σε συμβάντα (events).

Αυτές οι εφαρμογές διαθέσιμες σε περιέκτες (containers) ενορχηστρώνονται είτε με Docker Swarm είτε με Kubernetes.

Οι στόχοι και τα οφέλη της μετάβασης του γεωχωρικού διακομιστή σε αρχιτεκτονική μικροϋπηρεσίων είναι πολλά, μερικά από τα οποία αναφέρονται στην παρακάτω λίστα.

- Δυνατότητα αξιολόγησης και παροχής κατευθυντήριων γραμμών για τη σωστή διαστασιολόγηση των υπηρεσιών με βάση τις ανάγκες της καθεμίας από αυτές σε πόρους και χαρακτηριστικά απόδοσης.
- Ανεξάρτητη εξέλιξη των υπηρεσιών.
- Εξωτερική, κεντρική διαμόρφωση των υπηρεσιών και των υπο-συστατικών τους.
- Η απομόνωση υπηρεσίας επιτρέπει στο σύστημα να συνεχίσει να λειτουργεί σε περίπτωση που κάποια συγκεκριμένη υπηρεσία δεν είναι διαθέσιμη.
- Δυνατότητα υλοποίησης αυτόματης κλιμάκωσης ανά υπηρεσία.
- Δυνατότητα υλοποίησης συνεχών ροών εργασίας παράδοσης.
- Διαφάνεια τοποθεσίας παρουσιών υπηρεσιών σε δυναμικό περιβάλλον. Μια υπηρεσία "πύλης" λειτουργεί ως ενιαίο σημείο εισόδου σε όλα τα αιτήματα, αποστέλλοντάς τα στις κατάλληλες μικροϋπηρεσίες σε μια αυτόματη διαμόρφωση εξισορρόπησης φορτίου.
- Κεντρική καταγραφή υπηρεσιών, ανίχνευση αιτημάτων και παρακολούθηση.

## 4.2 Βάσεις για γεωχωρικά δεδομένα

Για την δημιουργία και επεξεργασία των δεδομένων της εφαρμογής V2ofDjango χρησιμοποιούνται τα ακόλουθα εργαλεία.

### 4.2.1 PostgreSQL

Για την οργάνωση, αποθήκευση και διαχείριση των δεδομένων χρησιμοποιείται η PostgreSQL που είναι εγκατεστημένη σε Linux Server (PostgreSQL, 2021).

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

Μερικά από τα κύρια χαρακτηριστικά της είναι:

- ANSI SQL 89, 92, 93.
- 100% συμβατή με ACID και πλήρη υποστήριξη commit και rollback.
- Online αντίγραφα ασφαλείας: υψηλότερη ασφάλεια και διαθεσιμότητα των δεδομένων.
- Τύποι δεδομένων: numeric, decimal, smallint, integer, bigint, real, double, serial, char, varchar, bit, text, date, time, timestamp, interval, boolean, network address, geometric types και πολλά άλλα.
- Δυνατότητα δημιουργίας νέων τύπων δεδομένων από τους χρήστες.
- Αποθήκευση BLOBS (binary large objects), συμπεριλαμβανομένων αρχείων κειμένου, ήχου, εικόνων ή βίντεο.
- Πλήρης υποστήριξη συναρτήσεων συγκεντρωτικών αποτελεσμάτων (GROUP_BY) όπως COUNT, SUM, AVG, MIN, MAX, STDDEV and VARIANCE. Δυνατότητα δημιουργίας νέων συγκεντρωτικών συναρτήσεων εφόσον χρειαστεί.
- Υποστήριξη όλων των τύπων ενώσεων (cross, inner, outer, left, right, full, natural).
- Συναρτήσεις ορισμένες από τον χρήστη, οι οποίες μπορούν να γραφούν σε πολλές γλώσσες προγραμματισμού όπως C, SQL, PL/pgSQL, TCL, Perl, Python, Ruby.
- Περιβάλλον ανάπτυξης γλωσσών προγραμματισμού.
- Βιβλιοθήκη συναρτήσεων και τελεστών με ορισμένες προεγκατεστημένες συναρτήσεις όπως math, date/time, string, geometric, formating κ.α.
- Συναρτήσεις trigger μπορούν να οριστούν από οποιαδήποτε γλώσσα προγραμματισμού που υποστηρίζει ο server όπως C ή PL/pgQL.
- Προσωρινοί πίνακες οι οποίοι σβήνονται αυτόματα μετά το τέλος της συνόδου.
- Μοντέλο ασφαλείας ομάδας/χρήστη. Η πρόσβαση στο διακομιστή της βάσης δεδομένων μπορεί να περιορίζεται από το χρήστη, κεντρικό υπολογιστή ή μια βάση δεδομένων.
- Το μέγεθος του πίνακα και της βάσης δεδομένων είναι σχεδόν απεριόριστο. Απεριόριστες καταχωρήσεις και ευρετήρια ανά πίνακα.

### 4.2.1 PostGIS

Για την δημιουργία και επεξεργασία των γεωσυνόλων χρησιμοποιείται η επέκταση της PostgreSQL η PostGIS (PostGIS, 2021).

Η PostGIS ενεργοποιεί χωρικά την PostgreSQL, επιτρέποντάς της να χρησιμοποιηθεί σαν βάση δεδομένων σε Γεωγραφικά Συστήματα Πληροφοριών (GIS) και εφαρμογές διαδικτυακής χαρτογράφησης.

Η PostGIS είναι σταθερή, γρήγορη, συμβατή με τα διεθνή πρότυπα, παρέχει εκατοντάδες χωρικές συναρτήσεις και είναι η πιο δημοφιλής χωρική βάση δεδομένων ανοιχτού κώδικα. Η PostGIS χρησιμοποιείται από ποικίλους οργανισμούς ανά τον κόσμο, περιλαμβανομένων υψηλού ρίσκου κυβερνητικών υπηρεσιών και οργανισμών, αποθηκεύοντας terrabytes δεδομένων και εξυπηρετώντας εκατομμύρια διαδικτυακές κλήσεις ανά ημέρα.

Η διαχείριση της βάσης δεδομένων γίνεται μεταξύ άλλων μέσω των pgAdmin και phpPgAdmin. Η είσοδος και έξοδος δεδομένων παρέχεται από πληθώρα εργαλείων μετατροπής (shp2pgsql, pgsql2shp, ogr2ogr, dxf2postgis). Υπάρχουν πολλά λογισμικά GIS (desktop και διαδικτυακά) για επισκόπηση δεδομένων σε PostGIS.

Βασικά Χαρακτηριστικά της PostGIS είναι:

- Χωρικές λειτουργίες
- Εργαλεία χωρικής ανάλυσης όπως: Ζώνες επιρροής, ένωση, τομή, επίθεση, απόσταση και πολλά περισσότερα
- Ολοκλήρωση συναλλαγών ACID
- Χωρικοί κατάλογοι R-Tree
- Υποστήριξη πολλών ταυτόχρονων χρηστών
- Κλείδωμα σε επίπεδο γραμμής
- Δυνατότητα αντιγραφής
- Στεγανοποίηση
- Ασφάλεια βάσει ρόλων
- Χώροι πινάκων, σχήματα βάσης
- Υλοποιημένα Πρότυπα
- Συμβατή με τα πρότυπα του OGC (SFSQL)

# Κεφ.5 Ανάλυση απαιτήσεων

Η ανάλυση και η οπτικοποίηση των γεωχωρικών δεδομένων έγινε στο μονόλιθο με τη χρήση της Python και του Django Framework. Οι απαιτήσεις παραμένουν οι ίδιες τόσο στην παλιά (μονόλιθος) όσο και στη νέα εφαρμογή (μικροϋπηρεσίες). Στις επόμενες παραγράφους παρουσιάζονται:

- Σε μορφή πίνακα τα services (endpoints) που έχουν υλοποιηθεί για την ικανοποίηση των επιχειρηματικών αναγκών της εφαρμογής,
- Σε μορφή διαγραμμάτων τόσο εκείνα τα στοιχεία της εφαρμογής που είναι όμοια στις δυο αρχιτεκτονικές και αφορούν στην ικανοποίηση ίδιων επιχειρηματικών απαιτήσεων (συμπεριφορά συστήματος, ροές εργασίας, διαγράμματα ακολουθίας), όσο και εκείνα που διαφέρουν και αφορούν στην αρχιτεκτονική της εφαρμογής.

## 5.1 Υπηρεσίες

Στον παρακάτω πίνακα γίνεται η συσχέτιση μεταξύ των χρηστών και της αλληλεπίδρασής τους με το σύστημα, παραθέτοντας την υπηρεσία (service) του μονόλιθου που υλοποιεί την κάθε προσφερόμενη λειτουργικότητα.

Στην νέα αρχιτεκτονική κάθε μέθοδος θα αποτελεί μια μικροϋπηρεσία, οι οποίες ανά ομάδες όπως περιγράφεται στο έβδομο κεφάλαιο θα φιλοξενηθούν σε dockers.

Πίν. 3. Mapping Table: Users' Interactions & api services

| User | User Interaction | Method (in monolith) / Service (in microservices) | Method Type (Verb) |
|---|---|---|---|
| Admin & Web app user | Register | /users/{id} | put |
| Admin & Web app user | Login / View Profile | /users/{id} | get |
| Admin & Web app user | Update Profile | /users/{id} | patch |

| | | | |
|---|---|---|---|
| Admin & Web app user | Delete User | /users/{id} | delete |
| Admin | View Users | /users/ | get |
| Admin | Edit Users | /users/ | post |
| Admin & Web app user | View Generation of Network Technologies[6] | /myapp/networks/ | get |
| Admin & Web app user | View Mobile Network Operators (MNOs)[7] | /myapp/operators/ | get |
| Admin & Web app user | View Operating Systems of Mobile Devices | /myapp/osStats/ | get |
| Admin & Web app user | View Device Manufacturers / Vendors | /myapp/vendorStats/ | get |
| Admin & Web app user | View network type per operator | /myapp/providers/ | get |
| Admin & Web app user | View Campaigns | /myapp/campaigns/ | get |
| Admin & Web app user | View Measurements | /myapp/measurements/ | get |

[6] - (καμία τιμή), 2G, 3G, 4G

[7] Wind, Vodafone, Cosmote and Other

| | | | |
|---|---|---|---|
| Admin & Web app user | View level stats | /myapp/levelStats/ | get |
| Admin & Web app user | View level stats per network type | /myapp/levelStats/{network_type}/ | get |
| Admin & Web app user | View level stats for a period of time | /myapp/levelStats/{start} - {end}/ | get |
| Admin & Web app user | View level stats per network type for a period of time | /myapp/levelStats/{start} – {end}/{network_type}/ | get |
| Admin & Web app user | View uplink stats | /myapp/uplinkStats/ | get |
| Admin & Web app user | View uplink stats per network type | /myapp/uplinkStats/{network_type}/ | get |
| Admin & Web app user | View uplink stats for a period of time | /myapp/uplinkStats/{start} – {end}/ | get |
| Admin & Web app user | View uplink stats per network type for a period of time | /myapp/uplinkStats/{start} – {end}/{network_type}/ | get |
| Admin & Web app user | View downlink stats | /myapp/downlinkStats/ | get |
| Admin & Web app user | View downlink stats per network type | /myapp/downlinkStats/{network_type}/ | get |

| | | | |
|---|---|---|---|
| Admin & Web app user | View downlink stats for a period of time | /myapp/downlinkStats/{start} – {end}/ | get |
| Admin & Web app user | View downlink stats per network type for a period of time | /myapp/downlinkStats/{start} - {end}/{network_type}/ | get |

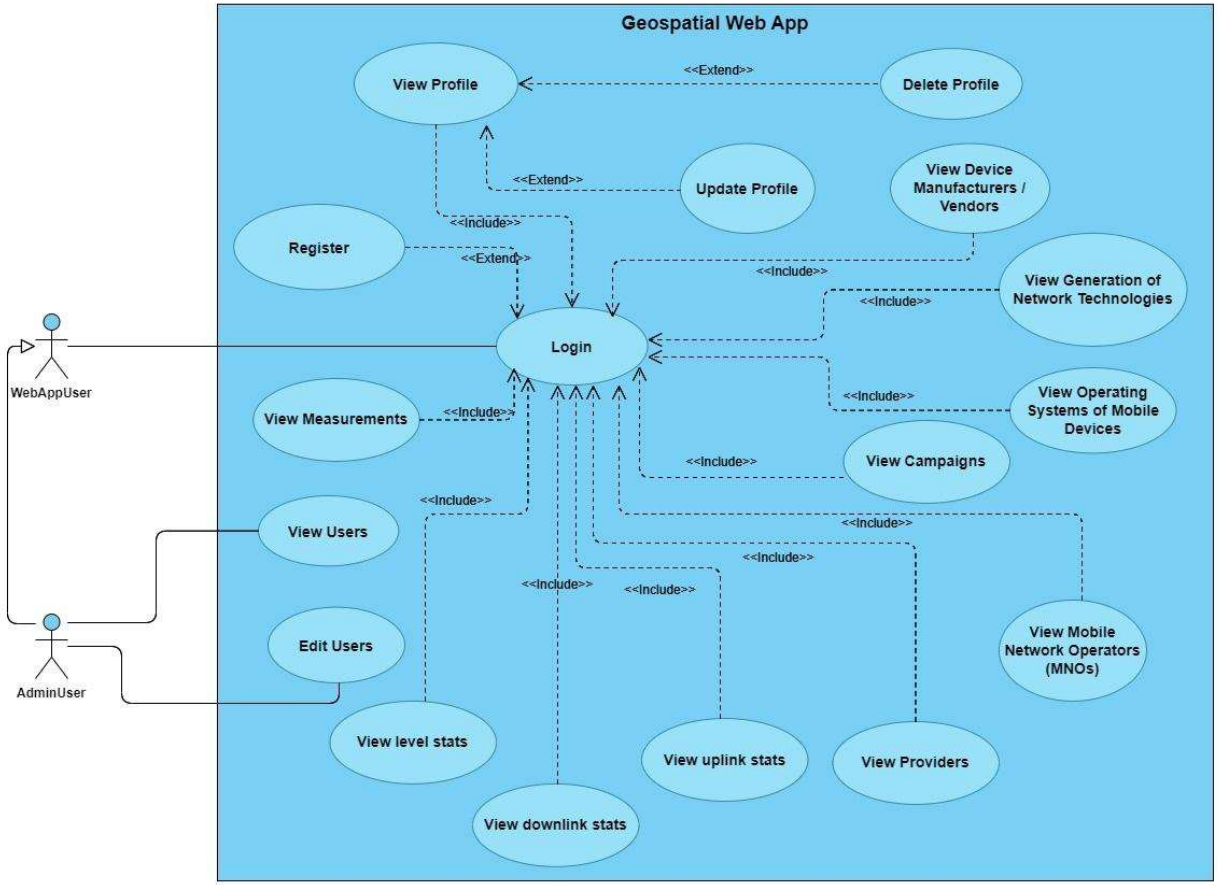
## 5.2 Διαγράμματα

### 5.2.1 Διάγραμμα περιπτώσεων χρήσης

Τα διαγράμματα περιπτώσεων χρήσης (use case diagrams) παρουσιάζουν το σύστημα όπως αυτό φαίνεται από τη θέση ενός εξωτερικού παρατηρητή και απεικονίζουν γραφικά τη συμπεριφορά του συστήματος. Στη φάση αυτή, της ανάλυσης της υπάρχουσας υλοποίησης, δημιουργήθηκε ένα τέτοιο διάγραμμα για να απεικονιστούν οι λειτουργικές απαιτήσεις του συστήματος και να γίνει κατανοητό πως το σύστημα έχει σχεδιαστεί να δουλεύει.

Στην παρακάτω απεικόνιση περιέχονται τα εξής στοιχεία:

- Actors (χρήστες ή ενεργούντες): οι αλληλεπιδρώντες με το σύστημα.
- Use cases (περιπτώσεις χρήσης): περιγράφουν τι κάνει το σύστημα/ποιες είναι οι λειτουργίες του.
- Interactions ή relationships (αλληλεπιδράσεις ή σχέσεις/συσχετίσεις): οι διάφοροι τύποι των σχέσεων ανάμεσα στους χρήστες και τις περιπτώσεις χρήσης.




Σχ. 1. Use-Case diagram

5.2.1.2 Περιγραφή περιπτώσεων χρήσης

Παρακάτω παρατίθενται οι πίνακες που περιγράφουν κάθε περίπτωση χρήσης του παραπάνω διαγράμματος.

Πίν. 4. Login Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **Login** |
|---|---|
| **Περιγραφή (Short Description):** | Σύνδεση χρήστη στην εφαρμογή |
| **Χρήστες (Actors):** | WebAppUser, AdminUser |
| **Έναυσμα (Trigger):** | 1) Καταχώρηση username και password ή<br>2) Επιλογή e-mail account για Login με google account ή<br>3) Καταχώρηση Git username για επιλογή Login με Git Credentials |
| **Προϋποθέσεις (Precondition):** | Εγγραφή χρήστη (Register) |
| **Μετασυνθήκες (Postcondition):** | Ο χρήστης έχει πρόσβαση στην εμφάνιση και την αλλαγή του προφίλ του καθώς και στις αναζητήσεις των δεδομένων των ζεύξεων.<br>Ειδικά για τον χρήστη Admin παρέχεται η επιπλέον δυνατότητα της εμφάνισης και της αλλαγής των στοιχείων όλων των χρηστών της εφαρμογής. |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Συνδεδεμένος στην εφαρμογή χρήστης |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Απόρριψη σύνδεσης χρήστη |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει το εικονίδιο Login<br>1.1. Ο χρήστης καταχωρεί το username και το password ή<br>1.2. Ο χρήστης επιλέγει Google account ή<br>1.3. Ο χρήστης επιλέγει Git account<br>2. Τα στοιχεία που έχουν υποβληθεί ελέγχονται για επαλήθευση |

| | |
|---|---|
| | 3. Ο χρήστης έχει τη δυνατότητα πλοήγησης στην εφαρμογή |
| **Included Cases:** | View Profile, View Generation of Network Technologies, View Mobile Network Operators (MNOs), View Operating Systems of Mobile Devices,View Device Manufacturers / Vendors, View Providers, View Campaigns, View uplink stats, View level stats, View downlink stats |
| **Extended Cases:** | Register |

Πίν. 5. Register Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **Register** |
|---|---|
| **Περιγραφή (Short Description):** | Εγγραφή χρήστη στην εφαρμογή |
| **Χρήστες (Actors):** | WebAppUser, AdminUser |
| **Έναυσμα (Trigger):** | 1) Καταχώρηση username και password ή<br>2) Επιλογή e-mail account για Sign up με google account ή<br>3) Καταχώρηση Git username για επιλογή Sign up με Git Credentials |
| **Προϋποθέσεις (Precondition):** | Καμία |
| **Μετασυνθήκες (Postcondition):** | Ο χρήστης μπορεί να συνδεθεί στην εφαρμογή |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Εγγεγραμμένος στην εφαρμογή χρήστης |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Απόρριψη εγγραφής χρήστη |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει το εικονίδιο Sign up<br>1.1. Ο χρήστης καταχωρεί το username και το password ή<br>1.2. Ο χρήστης επιλέγει Google account ή<br>1.3. Ο χρήστης επιλέγει Git account<br>2. Τα στοιχεία που έχουν υποβληθεί ελέγχονται για επαλήθευση<br>3. Ο χρήστης έχει τη δυνατότητα σύνδεσης στην εφαρμογή |

| | |
|---|---|
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 6. View Profile Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View Profile** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφάνιση στοιχείων χρήστη |
| **Χρήστες (Actors):** | WebAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή εμφάνιση στοιχείων προφίλ |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | Ο χρήστης μπορεί να μεταβάλλει τα στοιχεία του προφίλ του |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Εμφάνιση των στοιχείων του προφίλ του χρήστη |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Μη εμφάνιση των στοιχείων του προφίλ του χρήστη |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει το εικονίδιο “Εμφάνιση στοιχείων χρήστη”<br>2. Γίνεται ανάκτηση των στοιχείων του χρήστη από την βάση<br>3. Ο χρήστης βλέπει τα στοιχεία του προφίλ του στην διεπαφή χρήστη |
| **Included Cases:** | Login |
| **Extended Cases:** | Update Profile, Delete Profile |

Πίν. 7. Update Profile Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **Update Profile** |
|---|---|
| **Περιγραφή (Short Description):** | Μεταβολή στοιχείων χρήστη |
| **Χρήστες (Actors):** | WebAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή μεταβολής στοιχείων προφίλ |

| | |
|---|---|
| **Προϋποθέσεις (Precondition):** | Login, View Profile |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Μεταβολή των στοιχείων του προφίλ του χρήστη |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής μεταβολή των στοιχείων του χρήστη με την εμφάνιση κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει το εικονίδιο "Μεταβολή στοιχείων χρήστη"<br>2. Μετατρέπονται σε editable τα πεδία που είναι διαθέσιμα για αλλαγή<br>3. Ο χρήστης καταχωρεί τις νέες τιμές<br>4. Ο χρήστης επιλέγει καταχώριση<br>5. Γίνεται validation των νέων τιμών<br>6. Καταχωρούνται οι νέες τιμές στην βάση<br>7. Εμφανίζονται οι νέες τιμές στην οθόνη |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 8. Delete Profile Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **Delete Profile** |
|---|---|
| **Περιγραφή (Short Description):** | Διαγραφή χρήστη |
| **Χρήστες (Actors):** | WebAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή διαγραφής προφίλ |
| **Προϋποθέσεις (Precondition):** | Login, View Profile |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Διαγραφή του προφίλ του χρήστη. |
| **Καταστάσεις σφαλμάτων** | Ανεπιτυχής διαγραφή του προφίλ χρήστη με την εμφάνιση |

| | |
|---|---|
| **(Error situations - Failed End Condition):** | κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει το εικονίδιο “Διαγραφή προφίλ”<br>2. Καταχωρείται κατάλληλη ένδειξη στην βάση για μη διαθεσιμότητα του προφίλ<br>3. Ο χρήστης γίνεται Log out |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 9. View Users Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View Users** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφάνιση στοιχείων χρηστών |
| **Χρήστες (Actors):** | AdminUser |
| **Έναυσμα (Trigger):** | Εμφάνιση στοιχείων χρηστών |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | Edit Users |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Εμφάνιση των στοιχείων των χρηστών της εφαρμογής |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής εμφάνιση των στοιχείων των χρηστών της εφαρμογής με την εμφάνιση κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει το εικονίδιο “Εμφάνιση στοιχείων χρηστών”<br>2. Εμφανίζεται οθόνη με τα αναλυτικά στοιχεία όλων των χρηστών της εφαρμογής |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 10. Edit Users Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **Edit Users** |
|---|---|
| **Περιγραφή (Short Description):** | Μεταβολή στοιχείων χρηστών και διαγραφή χρηστών |
| **Χρήστες (Actors):** | AdminUser |
| **Έναυσμα (Trigger):** | Μεταβολή χρηστών |
| **Προϋποθέσεις (Precondition):** | Login, View Users |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Μεταβολή των στοιχείων των χρηστών της εφαρμογής ή διαγραφή τους |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής μεταβολή των στοιχείων των χρηστών της εφαρμογής ή της διαγραφής τους με την εμφάνιση κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει το εικονίδιο “Μεταβολή στοιχείων χρηστών”<br>2. Εμφανίζεται οθόνη με τα αναλυτικά στοιχεία όλων των χρηστών της εφαρμογής<br>3. Επιλέγει είτε:<br>3.1. Μεταβολή στοιχείων και στη συνέχεια καταχώριση των νέων στοιχείων<br>3.2. Διαγραφή χρήστη και καταχώριση της διαγραφής<br>4. Γινεται validation των νέων δεδομένων<br>5. Τα νέα δεδομένα καταχωρούνται στην βάση<br>6. Γίνεται επαναφόρτωση των στοιχείων της διεπαφής χρήστη με τα νέα δεδομένα |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 11. View Generation of Network Technologies Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View Generation of Network Technologies** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφανίζονται στην διεπαφή οι διαφορετικές τεχνολογίες που αφορούν στις γενιές των δικτύων |
| **Χρήστες (Actors):** | WepAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή από το μενού “Generation of Network Technologies” |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Εμφάνιση των στοιχείων των γενιών των δικτύων |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής εμφάνιση των στοιχείων των γενεών των δικτύων με την εμφάνιση κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει “Generation of Network Technologies”<br>2. Εμφανίζονται στην οθόνη τα αναλυτικά στοιχεία των δικτύων |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 12. View Mobile Network Operators (MNOs) Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View Mobile Network Operators (MNOs)** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφανίζονται στην διεπαφή οι φορείς εκμετάλλευσης δικτύου κινητής τηλεφωνίας |
| **Χρήστες (Actors):** | WepAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή από το μενού “Mobile Network Operators (MNOs)” |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση** | Εμφάνιση των φορέων εκμετάλλευσης των δικτύων κινητής |

| | |
|---|---|
| **(Successful End Condition):** | τηλεφωνίας |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής εμφάνιση των φορέων εκμετάλλευσης των δικτύων κινητής τηλεφωνίας με την εμφάνιση κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει “Mobile Network Operators (MNOs)”<br>2. Εμφανίζονται στην οθόνη τα αναλυτικά στοιχεία των φορέων κινητής τηλεφωνίας |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 13. View Operating Systems of Mobile Devices Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View Operating Systems of Mobile Devices** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφανίζονται στην διεπαφή τα διαφορετικά λειτουργικά συστήματα των συσκευών κινητής τηλεφωνίας |
| **Χρήστες (Actors):** | WepAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή από το μενού “Operating Systems of Mobile Devices” |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Εμφάνιση των λειτουργικών συστημάτων των συσκευών κινητής τηλεφωνίας |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής εμφάνιση των λειτουργικών συστημάτων των συσκευών κινητής τηλεφωνίας με την εμφάνιση κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει “Operating Systems of Mobile Devices”<br>2. Εμφανίζονται στην οθόνη τα αναλυτικά στοιχεία των λειτουργικών συστημάτων των συσκευών κινητής τηλεφωνίας |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 14. View Device Manufacturers / Vendors Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View Device Manufacturers / Vendors** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφανίζονται στην διεπαφή οι κατασκευαστές κινητών τηλεφώνων |
| **Χρήστες (Actors):** | WepAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή από το μενού “Device Manufacturers / Vendors” |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Εμφάνιση των στοιχείων των κατασκευαστών κινητών τηλεφώνων. |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής εμφάνιση των στοιχείων των κατασκευαστών κινητών τηλεφώνων με την εμφάνιση κατάλληλου μηνύματος. |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει “Device Manufacturers / Vendors”<br>2. Εμφανίζονται στην οθόνη τα αναλυτικά στοιχεία των κατασκευαστών κινητών τηλεφώνων. |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 15. View Providers Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View Providers** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφανίζονται στην διεπαφή τα στοιχεία των παρόχων των δικτύων της κινητής τηλεφωνίας |
| **Χρήστες (Actors):** | WepAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή από το μενού “Providers” |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Εμφάνιση των στοιχείων των παρόχων των δικτύων της κινητής τηλεφωνίας |

| | |
|---|---|
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής εμφάνιση των στοιχείων των παρόχων των δικτύων της κινητής τηλεφωνίας με την εμφάνιση κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει “Device Manufacturers / Vendors”<br>2. Εμφανίζονται στην οθόνη τα αναλυτικά στοιχεία των παρόχων των δικτύων της κινητής τηλεφωνίας |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 16. View Campaigns Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View Campaigns** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφανίζονται στην διεπαφή τα στοιχεία από τις καμπάνιες (περιοχές ενδιαφέροντος προς μέτρηση) |
| **Χρήστες (Actors):** | WepAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή από το μενού “Campaigns” |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Εμφάνιση των στοιχείων που αφορούν στις καμπάνιες |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής εμφάνιση των στοιχείων που αφορούν στις καμπάνιες με την εμφάνιση κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει “Campaigns”<br>2. Εμφανίζονται στην οθόνη τα αναλυτικά στοιχεία για τις καμπάνιες |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 17. View Measurements Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View Measurements** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφανίζονται στην διεπαφή όλα τα στοιχεία των μετρήσεων |
| **Χρήστες (Actors):** | WepAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή από το μενού “Measurements” |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Εμφάνιση των στοιχείων που αφορούν στις μετρήσεις |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής εμφάνιση των στοιχείων που αφορούν στις μετρήσεις με την εμφάνιση κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 1. Ο χρήστης επιλέγει “Measurements”<br>2. Εμφανίζονται στην οθόνη τα αναλυτικά στοιχεία για τις μετρήσεις |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 18. View level Stats Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View level stats** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφανίζονται στην διεπαφή των στοιχεία που αφορούν την ισχύ του σήματος των ζεύξεων (και σε συνδυασμό με τον τύπο του δικτύου και το εύρος ημερομηνίας) |
| **Χρήστες (Actors):** | WepAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή από το μενού “Level Stats” |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση** | Εμφάνιση των στοιχείων που αφορούν στα στοιχεία του συνόλου |

| | |
|---|---|
| **(Successful End Condition):** | της ανοδικής και της καθοδικής ζεύξης |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής εμφάνιση των στοιχείων που αφορούν στα στοιχεία των συνόλων των ανοδικών και των καθοδικών ζεύξεων με την εμφάνιση κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 3. Ο χρήστης επιλέγει “Level Stats”<br>4. Εμφανίζονται στην οθόνη τα αναλυτικά στοιχεία για το σύνολο της ζεύξης |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 19. View Uplink Stats Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View uplink stats** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφανίζονται στην διεπαφή των στοιχεία που αφορούν την ισχύ του σήματος των ανοδικών ζεύξεων (και σε συνδυασμό με τον τύπο του δικτύου και το εύρος ημερομηνίας) |
| **Χρήστες (Actors):** | WepAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή από το μενού “Uplink Stats” |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Εμφάνιση των στοιχείων που αφορούν στα στοιχεία των ανοδικών ζεύξεων |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής εμφάνιση των στοιχείων που αφορούν στα στοιχεία των ανοδικών ζεύξεων με την εμφάνιση κατάλληλου μηνύματος |
| **Περιγραφή ροής (Standard process):** | 5. Ο χρήστης επιλέγει “Uplink Stats”<br>6. Εμφανίζονται στην οθόνη τα αναλυτικά στοιχεία για τις ανοδικές ζεύξεις |
| **Included Cases:** | - |
| **Extended Cases:** | - |

Πίν. 20. View Downlink Stats Use Case

| **Ονομασία περίπτωσης χρήσης (Use Case Name ):** | **View downlink stats** |
|---|---|
| **Περιγραφή (Short Description):** | Εμφανίζονται στην διεπαφή των στοιχεία που αφορούν την ισχύ του σήματος των καθοδικών ζεύξεων (και σε συνδυασμό με τον τύπο του δικτύου και το εύρος ημερομηνίας) |
| **Χρήστες (Actors):** | WepAppUser, AdminUser |
| **Έναυσμα (Trigger):** | Επιλογή από το μενού “Downlink Stats” |
| **Προϋποθέσεις (Precondition):** | Login |
| **Μετασυνθήκες (Postcondition):** | - |
| **Επιτυχής ολοκλήρωση (Successful End Condition):** | Εμφάνιση των στοιχείων που αφορούν στα στοιχεία των καθοδικών ζεύξεων |
| **Καταστάσεις σφαλμάτων (Error situations - Failed End Condition):** | Ανεπιτυχής εμφάνιση των στοιχείων που αφορούν στα στοιχεία των καθοδικών ζεύξεων με την εμφάνιση κατάλληλου μηνύματος. |
| **Περιγραφή ροής (Standard process):** | 7. Ο χρήστης επιλέγει “Downlink Stats”<br>8. Εμφανίζονται στην οθόνη τα αναλυτικά στοιχεία για τις καθοδικές ζεύξεις |
| **Included Cases:** | - |
| **Extended Cases:** | - |

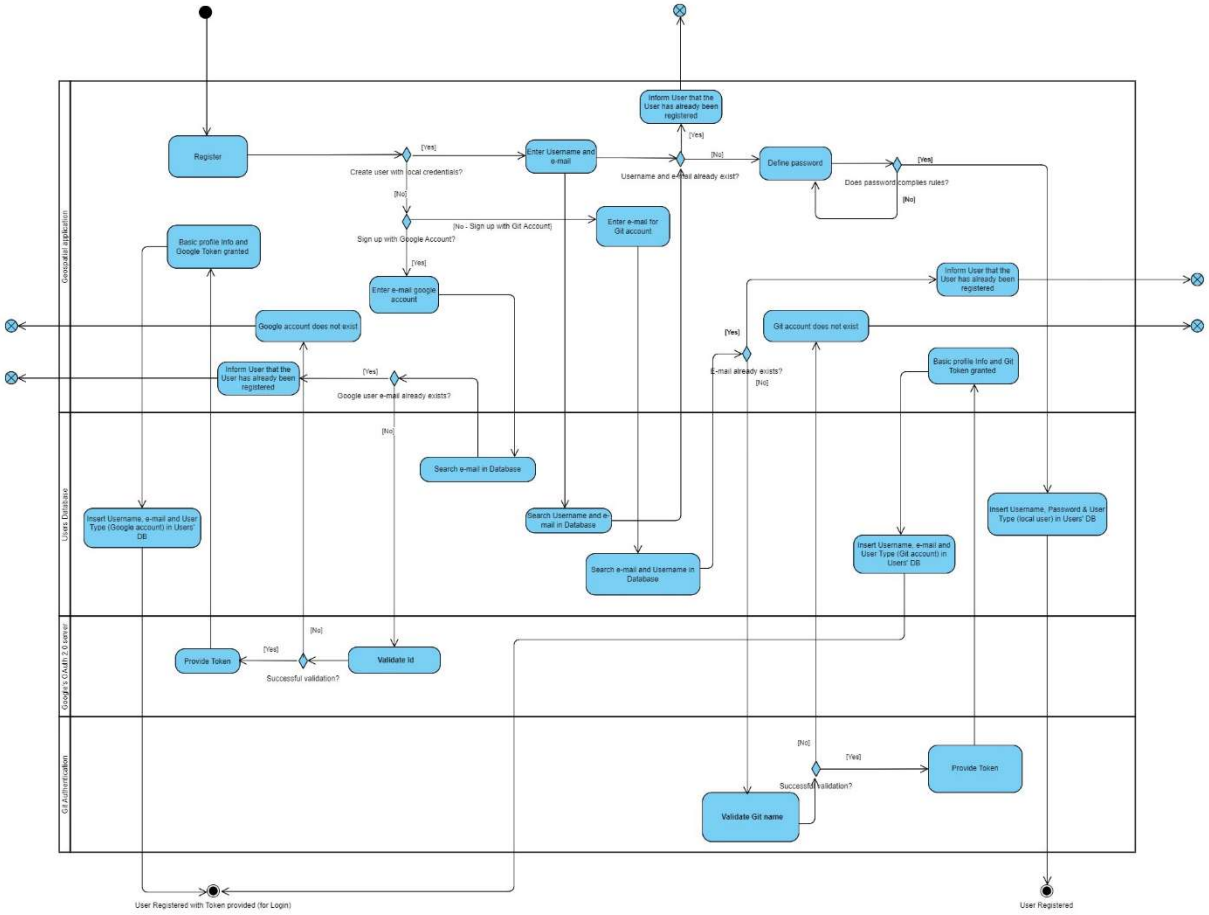
### 5.2.3 Διαγράμματα δραστηριότητας

Τα Διαγράμματα δραστηριότητας (Activity diagrams) είναι γραφικές αναπαραστάσεις των ροών εργασίας (workflows) και απεικονίζουν τις βηματικές δραστηριότητες του συστήματος. Είναι διαγράμματα που προορίζονται να μοντελοποιήσουν τόσο τις υπολογιστικές όσο και τις οργανωτικές διαδικασίες (“Διαγράμματα Δραστηριότητας”, 2017). Τα διαγράμματα δραστηριότητας παρουσιάζουν τη συνολική ροή του ελέγχου και δείχνουν την ακολουθία των δραστηριοτήτων που ακολουθούνται.

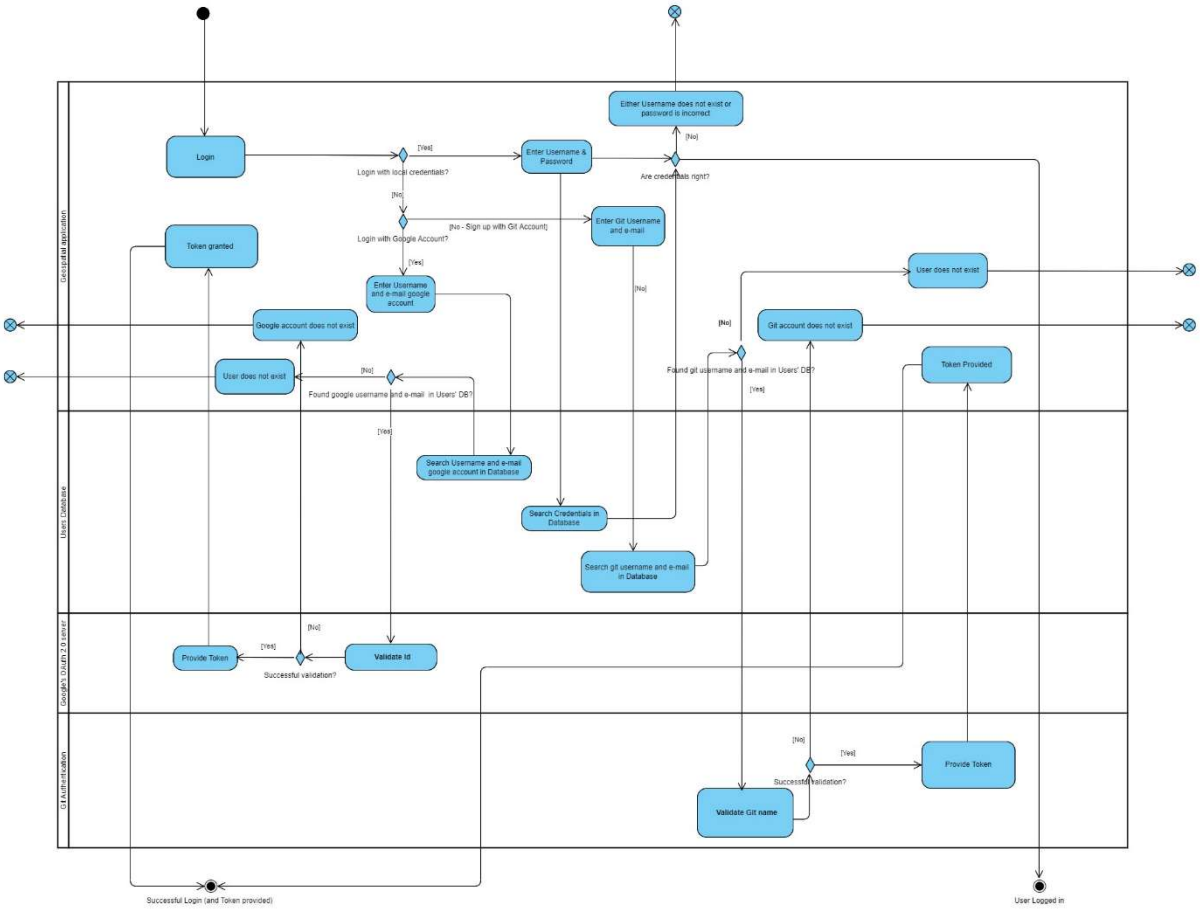
Στο παρακάτω σχήμα αναπαρίστανται οι ροές εργασίας της εφαρμογής για την εγγραφή και την είσοδο ενός χρήστη.

Να σημειωθεί ότι η δημιουργία Admin χρήστη υποστηρίζεται από το Django framework. Ο Admin πέραν όλης της λειτουργικότητας που έχει ο απλός χρήστης: ενημέρωση στοιχείων του

προφίλ του, αναζήτηση στοιχείων των ζεύξεων επιτελεί ακόμη δυο λειτουργίες: διαγραφή, ανανέωση χρηστών καθώς και τη δημιουργία άλλων διαχειριστών.




Σχ. 2. Activity diagram για Sign Up




Σχ. 3. Activity diagram για Login

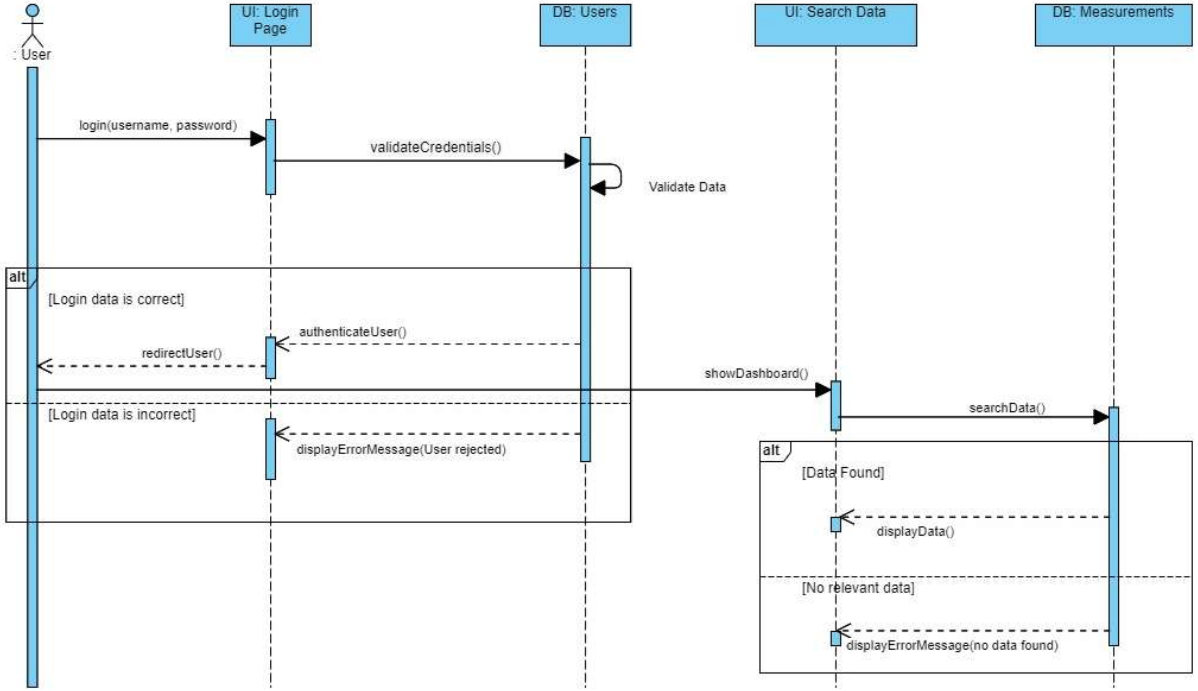
### 5.2.4 Διαγράμματα ακολουθίας

Ένα διάγραμμα ακολουθίας (sequence diagram) απεικονίζει την ακολουθία μηνυμάτων μεταξύ αντικειμένων σε μια αλληλεπίδραση και αποτελείται από μια ομάδα αντικειμένων που αντιπροσωπεύονται από γραμμές ζωής και τα μηνύματα που ανταλλάσσονται με την πάροδο του χρόνου κατά τη διάρκεια της αλληλεπίδρασης. Αυτά τα διαγράμματα ενίοτε αποκαλούνται και διαγράμματα συμβάντων (event diagrams) ή σενάρια συμβάντων (event scenarios).

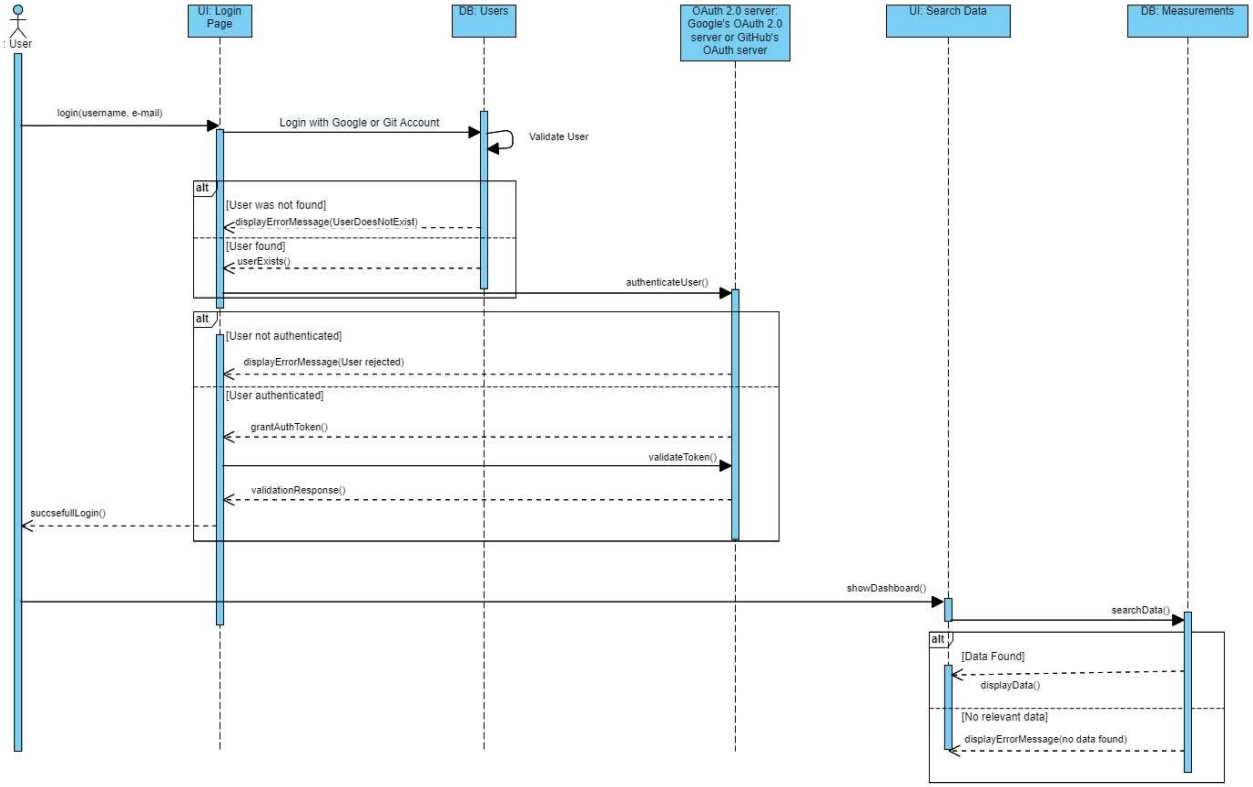
Προκειμένου να γίνουν πιο κατανοητά τα παρακάτω διαγράμματα ακολουθίας, κυρίως στο σημείο που αφορούν στην ταυτοποίηση χρήστη (authorization) από υπηρεσίες τρίτων καθώς και στην παροχή πρόσβασης σε υπηρεσίων τρίτων (authenticaion), θα πρέπει οι έννοιες που σχετίζονται με το OpenId 2.0 πρότυπο και OAuth 2.0 πρωτόκολλο να είναι οικείες στον αναγνώστη (βλ. παράγραφος 5.3).

Η ταυτοποίηση του χρήστη καθώς και η παροχή προσβάσεων σε υπηρεσίες της εφαρμογής (αλλαγή στοιχείων χρήστη και αναζήτηση στοιχείων των ζεύξεων) γίνεται είτε:

1) Με τη χρήση username, password που εκδίδονται και χρησιμοποιούνται από την ίδια την εφαρμογή και ολοκληρώνεται με τη ροή client credentials,
2) Με τη χρήση του Google Sign-In, το οποίο ενσωματώνει το OpenId πρωτόκολλο για το authentication και το OAuth 2.0 πρωτόκολλο για το authorization σε δεδομένα που αφορούν τον λογαριασμό του χρήστη. Τα scopes που έχουν δηλωθεί είναι τα: openId και profile (για εμφάνιση των βασικών στοιχείων του χρήστη) και email (για εμφάνιση των στοιχείων του λογαριασμού του ηλεκτρονικού ταχυδρομείου) και ο τύπος του flow είναι authorization code grant type (Google Developers, December 2021) ή
3) Με τη χρήση του Web application flow του GitHub's OAuth implementation που είναι τύπου authorization code grant type και τη δήλωση των scopes: repo και gist (GitHub Inc., 2022).




Σχ. 4. Διάγραμμα ακολουθίας για Login με local credentials και αναζήτηση δεδομένων για τις ζεύξεις




Σχ. 5. Διάγραμμα ακολουθίας για Login με Google Sign-In ή Git Login και αναζήτηση δεδομένων για τις ζεύξεις

## 5.3 Πρωτόκολλα Ελέγχου Ταυτότητας

Το OpenID είναι ένα ανοιχτό πρότυπο και αποκεντρωμένο πρωτόκολλο ελέγχου ταυτότητας που προωθείται από το μη κερδοσκοπικό OpenID Foundation. Επιτρέπει στους χρήστες τον έλεγχο ταυτότητας από συνεργαζόμενους ιστότοπους (γνωστούς ως εξαρτημένα μέρη ή RP: Relying Parties) χρησιμοποιώντας μια υπηρεσία παροχής ταυτότητας τρίτου μέρους (Third-party Identity Provider: IDP), εξαλείφοντας την ανάγκη των webmasters να παρέχουν τα δικά τους ad hoc συστήματα σύνδεσης και επιτρέποντας στους χρήστες να συνδεθούν σε πολλούς άσχετους μεταξύ τους ιστότοπους χωρίς να χρειάζεται να διαθέτουν ξεχωριστή ταυτότητα και κωδικό πρόσβασης για τον καθένα. Οι χρήστες δημιουργούν λογαριασμούς επιλέγοντας έναν πάροχο ταυτότητας OpenID Connect και στη συνέχεια χρησιμοποιούν αυτούς τους λογαριασμούς για να συνδεθούν σε οποιονδήποτε ιστότοπο αποδέχεται έλεγχο ταυτότητας OpenID. Αρκετοί μεγάλοι οργανισμοί εκδίδουν ή αποδέχονται OpenID στους ιστότοπούς τους.

Η τελική έκδοση του OpenID είναι το OpenID 2.0, οριστικοποιήθηκε και δημοσιεύτηκε τον Δεκέμβριο του 2007. Ο όρος OpenID μπορεί επίσης να αναφέρεται σε ένα αναγνωριστικό όπως καθορίζεται στο πρότυπο OpenID. Αυτά τα αναγνωριστικά έχουν τη μορφή ενός μοναδικού Uniform Resource Identifier (URI) και τα διαχειρίζεται κάποιος "πάροχος OpenID" που χειρίζεται τον έλεγχο ταυτότητας.

Το OAuth 2.0 πρωτόκολλο, το οποίο σημαίνει "Open Authorization", είναι ένα πρότυπο που έχει σχεδιαστεί για να επιτρέπει σε έναν ιστότοπο ή μια εφαρμογή να έχει πρόσβαση σε πόρους που φιλοξενούνται από άλλες εφαρμογές ιστού για λογαριασμό ενός χρήστη.

Το OAuth 2.0 είναι ένα ανοιχτό πρότυπο πλαίσιο εξουσιοδότησης για εξουσιοδότηση που βασίζεται σε διακριτικά πρόσβασης (tokens) στο διαδίκτυο. Με τη χρήση του πρωτοκόλλου δύναται η χρήση των πληροφοριών λογαριασμού ενός τελικού χρήστη από υπηρεσίες τρίτων, όπως το Facebook, η Google και το Twitter, χωρίς να εκτίθενται τα διαπιστευτήρια του λογαριασμού του χρήστη σε τρίτους. Λειτουργεί ως ενδιάμεσος για λογαριασμό του τελικού χρήστη, παρέχοντας στην υπηρεσία τρίτου ένα διακριτικό πρόσβασης που εξουσιοδοτεί την κοινή χρήση συγκεκριμένων πληροφοριών λογαριασμού. Η διαδικασία για την απόκτηση του διακριτικού ονομάζεται ροή εξουσιοδότησης (D. Hardt, October 2012).

Υπάρχουν 4 διαφορετικές ροές OAuth2 (grant types/flows):

- Client credential
- Resource owner password credential
- Authorization code
- Implicit

Κάθε ροή τύπου επιχορήγησης OAuth2 έχει τον ίδιο στόχο:

Τη λήψη του κλειδιού εξουσιοδότησης/διακριτικού πρόσβασης (authorization key/access token) για τον χρήστη, το οποίο αντιπροσωπεύει ένα σύνολο αδειών (permissions) και την εκτέλεση κάποιου κομματιού κώδικα για λογαριασμό του.

Η επίτευξη αυτού του στόχου είναι μια ροή 2 μερών και αναλύεται στα εξής 2 βήματα:

1. **Λήψη του διακριτικού πρόσβασης (Get Access Token)**

Απόκτηση του κλειδιού εξουσιοδότησης/διακριτικού πρόσβασης (authorization key/access token) του χρήστη από τον πάροχο OAuth, π.χ. Twitter

2. **Χρησιμοποίηση του διακριτικού πρόσβασης (Use Access Token)**

Χρησιμοποίηση του κλειδιού εξουσιοδότησης/διακριτικού πρόσβασης για την εκτέλεση κώδικα καλώντας ένα προστατευμένο τελικό σημείο API (a protected API endpoint) για λογαριασμό του χρήστη, π.χ., δημοσίευση ενός tweet.

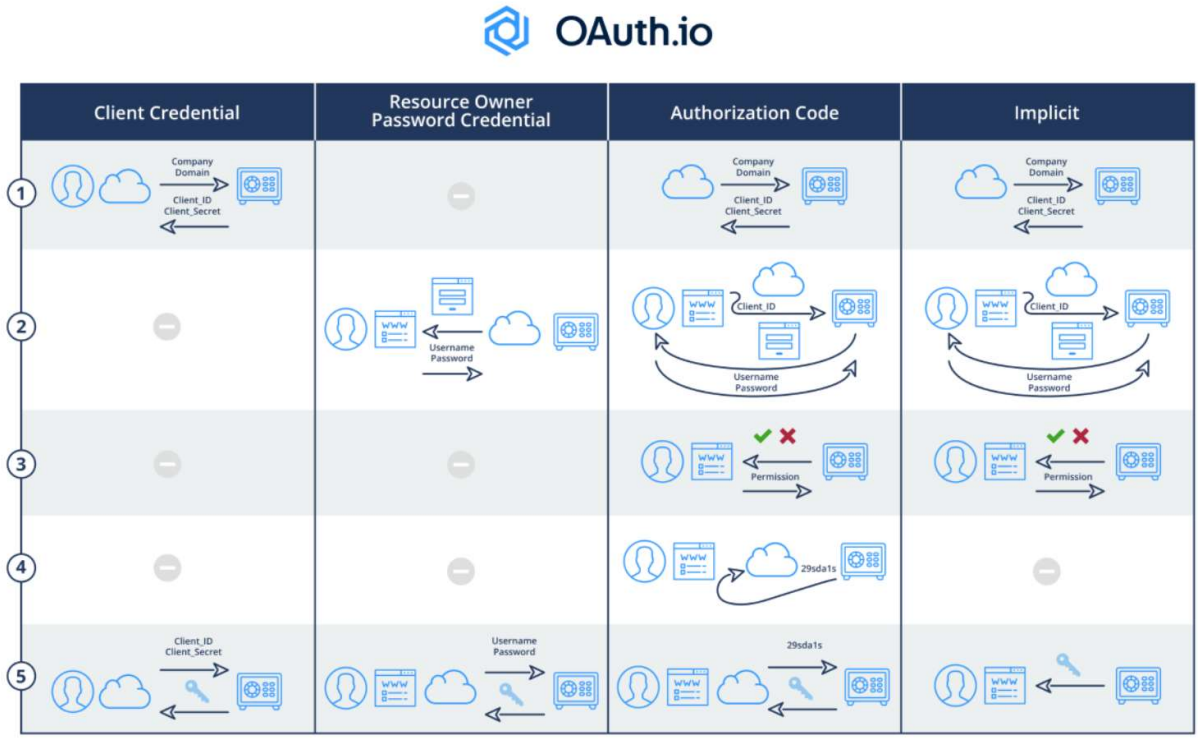
### 5.3.1 Διαφορές μεταξύ των διαγραμμάτων ροής τύπου επιχορήγησης OAuth2

Κάθε ροή τύπου επιχορήγησης OAuth2 διαφέρει μόνο στο πρώτο μέρος της κύριας ροής που αφορά στον τρόπο λήψης του διακριτικού πρόσβασης.

Κατ 'αρχήν, η ροή Get Access Token έχει 5 βήματα (όπως φαίνεται στο πιο κάτω διάγραμμα):

1. Προεγγραφή πελάτη -Εφαρμογή- με διακομιστή OAuth για τη λήψη του Αναγνωριστικού πελάτη/Μυστικό πελάτη (Pre-register Client -App- with OAuth Server to get Client ID/Client Secret.
2. Ο διακομιστής OAuth ελέγχει την ταυτότητα του χρήστη όταν κάνει κλικ στο κουμπί σύνδεσης κοινωνικής δικτύωσης της εφαρμογής, το οποίο επισημαίνεται με το αναγνωριστικό πελάτη (OAuth Server authenticates user when she clicks on the App’s social login button, which is tagged with Client ID).
3. Ο διακομιστής OAuth ζητά την άδεια χρήστη για να επιτρέψει στην εφαρμογή να εκτελεί κάτι για λογαριασμό της (OAuth Server solicits user permission to allow the App to perform something on her behalf).
4. Ο διακομιστής OAuth στέλνει μυστικό κώδικα στην εφαρμογή (OAuth Server sends secret Code to App).

5. Η εφαρμογή αποκτά διακριτικό κλειδιού/πρόσβασης από τον διακομιστή OAuth παρουσιάζοντας μυστικό κωδικό και μυστικό πελάτη (App acquires Key/Access Token from OAuth Server by presenting secret Code and Client Secret).




Εικ. 33. Διαφορές μεταξύ των διαγραμμάτων ροής τύπου επιχορήγησης OAuth2[8]

Σημειώσεις:
Το εικονίδιο “σύννεφο” αντιπροσωπεύει την εφαρμογή
Το εικονίδιο “www” αντιπροσωπεύει τον χρήστη/πρόγραμμα περιήγησης
Το εικονίδιο “χρηματοκιβώτιο” αντιπροσωπεύει τον διακομιστή OAuth
Το εικονίδιο το “απαγορευτικό” σημαίνει ότι το βήμα δεν απαιτείται στη ροή τύπου επιχορήγησης

#### 5.3.1.1 Client Credential Grant Type Flow

Η ροή τύπου επιχορήγησης “Διαπιστευτηρίων πελάτη” (Client Credential grant type flow) είναι εύκολα υλοποιήσιμη, αφού έχει δύο μόνο βήματα, αλλά απαιτεί από τον χρήστη να είναι ίδια η οντότητά του με την Εφαρμογή, καθώς ο χρήστης θα χρησιμοποιήσει το αναγνωριστικό πελάτη/μυστικό πελάτη της εφαρμογής για να ταυτοποιήσει τον εαυτό του κατά την επικοινωνία με το OAuth Διακομιστής στο Βήμα #5. Επομένως, η περίπτωση χρήσης για αυτόν

[8] **Πηγή**: OAuth.io, 2018, September

τον τύπο επιχορήγησης είναι περιορισμένη, αλλά ασφαλής καθώς το κλειδί/διακριτικό πρόσβασης παραδίδεται στην Εφαρμογή (backend), η οποία είναι καλά φυλαγμένη.

#### 5.3.1.2 Resource Owner Password Credential Grant Type Flow

Η ροή τύπου επιχορήγησης “Διαπιστευτηρίων κωδικού πρόσβασης κατόχου πόρων” (Resource Owner Password Credential grant type flow) είναι εύκολα υλοποιήσιμη με μόνο 2 βήματα, αλλά απαιτεί από τον χρήστη να παραδώσει το όνομα χρήστη/κωδικό πρόσβασης στην εφαρμογή στο Βήμα #2. Εκτός εάν η Εφαρμογή και ο Διακομιστής OAuth ανήκουν στην ίδια οντότητα, η παράδοση του ονόματος χρήστη/κωδικού πρόσβασης στην Εφαρμογή εξουσιοδοτεί την Εφαρμογή με όλα τα προνόμια ως Χρήστης, κάτι που αποτελεί κίνδυνο για την ασφάλεια.

#### 5.3.1.3 Authorization Code Grant Type Flow

Η ροή τύπου επιχορήγησης “Κωδικός εξουσιοδότησης” (Authorization Code grant type flow) είναι η πιο περίπλοκη, αφού διαθέτει και τα 5 βήματα λήψης κλειδιού και είναι επίσης η πιο ασφαλής, καθώς το κλειδί/διακριτικό πρόσβασης εκδίδεται μόνο στην εφαρμογή (backend), το οποίο και είναι καλά φυλαγμένο (Βήμα #5), μειώνοντας έτσι την επιφάνεια επίθεσης του συστήματος.

#### 5.3.1.4 Implicit Code Grant Type Flow

Η ροή τύπου “Σιωπηρής επιχορήγησης” (Implicit grant type flow) μοιάζει περισσότερο με τον Κώδικα εξουσιοδότησης, εκτός από το γεγονός ότι το Βήμα #4 δεν απαιτείται, δηλαδή, ο διακομιστής OAuth παραδίδει το κλειδί/διακριτικό πρόσβασης απευθείας στον χρήστη/πρόγραμμα περιήγησης. Αυτό αυξάνει μετρίως την επιφάνεια επίθεσης του συστήματος καθώς κλειδί/διακριτικό πρόσβασης είναι αποθηκευμένο στο πρόγραμμα περιήγησης, το οποίο είναι περισσότερο εκτεθειμένο στο διαδίκτυο από ό,τι στην εφαρμογή (backend). Αυτό συχνά μετριάζεται με την παροχή κλειδιού/κουπονιού πρόσβασης που δεν μπορεί να ανανεωθεί και έχει μικρότερη λήξη. Αυτή η ροή είναι απαραίτητη όταν η εφαρμογή δεν διαθέτει backend, όπως μια εφαρμογή μιας σελίδας (SPA: Single-page App) ή μια εγγενή εφαρμογή για κινητά.

### 5.3.2 Επιλογή του κατάλληλου τύπου επιχορήγησης

Συνοπτικά, η επιλογή του σωστού τύπου επιχορήγησης εξαρτάται από το:

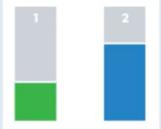
- Εάν οποιοδήποτε μέρος του κώδικά είναι ιδιωτικό, και μπορεί έτσι να αποθηκεύσει ένα μυστικό κλειδί (secret key)
- Επίπεδο εμπιστοσύνης μεταξύ χρήστη και εφαρμογής

Στον παρακάτω πίνακα απαριθμούνται οι προϋποθέσεις για την επιλογή ενός τύπου επιχορήγησης κατατάσσονται με βάση:

- Δυσκολία (1 έως 3, με το 3 να είναι πιο δύσκολο και πολύπλοκο)
- Ασφάλεια (1 έως 3, με το 3 να είναι πιο ασφαλές)

Πίν. 21. Συνθήκες επιλογής διαφορετικών τύπων επιχορήγησης

| Συνθήκη | Grant Type/Flow | Δυσκολία (πράσινο) vs. Ασφάλεια (μπλε) | Σημειώσεις |
|---|---|---|---|
| Εάν ο πελάτης και ο κάτοχος πόρων είναι η ίδια οντότητα. | Client credential |  | Όλος ο κώδικας βρίσκεται στο backend.<br>Ο κωδικός υποστήριξης είναι ιδιωτικός, επομένως μπορεί να κρατήσει το μυστικό κλειδί, το οποίο χρησιμοποιείται για την απόκτηση διακριτικού πρόσβασης. |
| Εάν η κύρια λογική της εφαρμογής βρίσκεται στο backend, ενώ το front-end είναι υπεύθυνο μόνο για την παρουσίαση. | Authorization code |  | Ο κωδικός υποστήριξης είναι ιδιωτικός, ώστε να μπορεί να κρατήσει το μυστικό κλειδί, το οποίο χρησιμοποιείται για την απόκτηση διακριτικού πρόσβασης. |

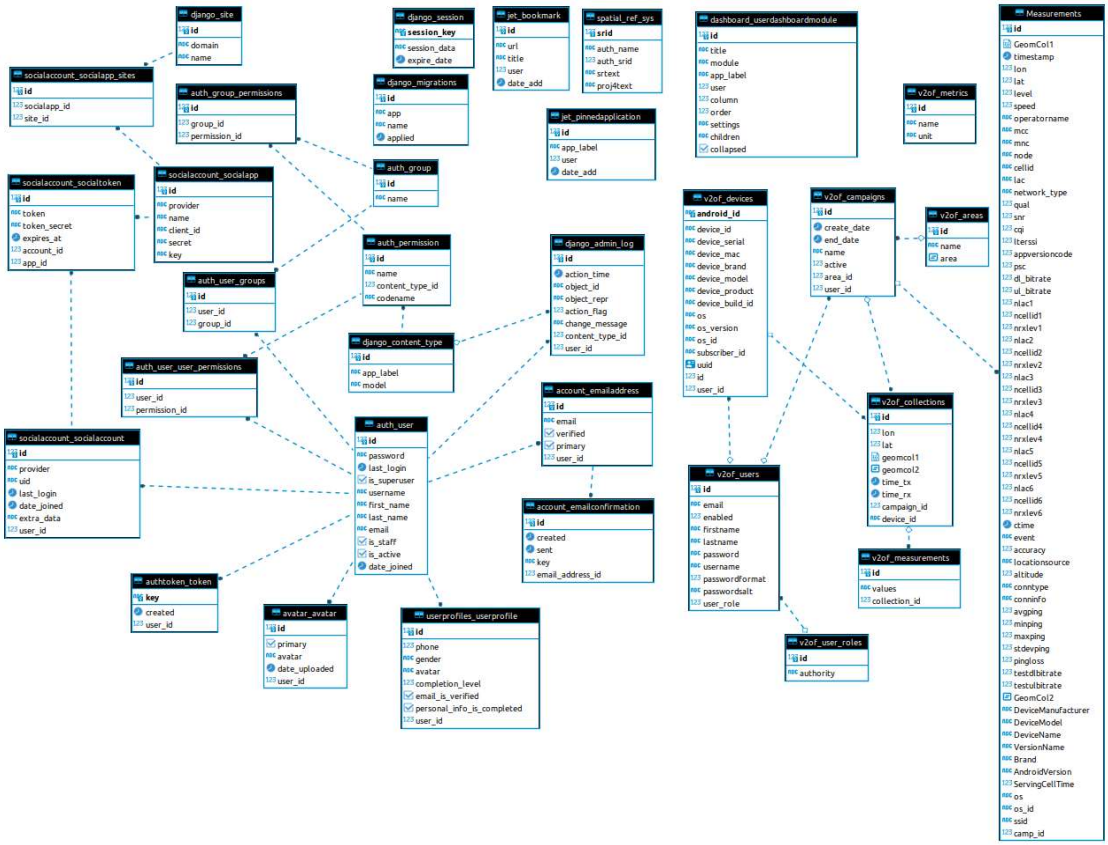
| | | | |
|---|---|---|---|
| Η εφαρμογή με την οποία αλληλεπιδρούν οι χρήστες σας δεν μπορεί να εκκινήσει πρόγραμμα περιήγησης Ιστού ούτε να υποστηρίξει ανακατεύθυνση [1]<br><br>Υπάρχει υψηλό επίπεδο εμπιστοσύνης μεταξύ χρήστη και εφαρμογής, π.χ., η εφαρμογή είναι το λειτουργικό σύστημα, το οποίο είναι ανοιχτού κώδικα, ή υψηλό επίπεδο εμπιστοσύνης μεταξύ της εφαρμογής και του παρόχου OAuth, π.χ. ο πάροχος OAuth κατασκευάζει την εφαρμογή. | Resource owner password credential |  | Ο χρήστης παραδίδει τα διαπιστευτήρια του κωδικού πρόσβασης στην εφαρμογή, κάτι που αποτελεί κίνδυνο ασφαλείας.<br><br>Η εφαρμογή μπορεί να ζητήσει πλήρη πρόσβαση σε ένα εύρος πόρων του χρήστη, αφού διαθέτει διαπιστευτήρια κωδικού πρόσβασης.<br><br>Συνήθως μόνο για εφαρμογές που εκτελούνται σε υλικό/λογισμικό ή εφαρμογές παλαιού τύπου. |
| Η εφαρμογή είναι εφαρμογή μιας σελίδας (SPA: single-page application) ή μια εγγενής εφαρμογή και αλληλεπιδρά με τον πόρο χρήστη στον πάροχο OAuth. | Implicit |  | Ο κωδικός της διεπαφής (front-end) αποκτά πρόσβαση στο διακριτικό πρόσβασης.<br><br>Το διακριτικό πρόσβασης στον κώδικα διεπαφής έχει πιθανότητα να παραβιαστεί, π.χ. όταν το πρόγραμμα περιήγησης Ιστού έχει μια τρύπα ασφαλείας που εκθέτει το διακριτικό πρόσβασης σε άλλους ιστότοπους που επισκέπτεται ο χρήστης. |

## Κεφ.6. Υφιστάμενη μονολιθική εφαρμογή

Παρακάτω απεικονίζονται, το αρχιτεκτονικό διάγραμμα της εφαρμογής σε υψηλό επίπεδο, και το E-R διάγραμμα της βάσης Measurements. Η βάση θα παραμείνει η ίδια και στην νέα αρχιτεκτονική (μικροϋπηρεσίες), συνεπώς δεν απαιτείται migration των δεδομένων.

## 6.1 E-R Diagram

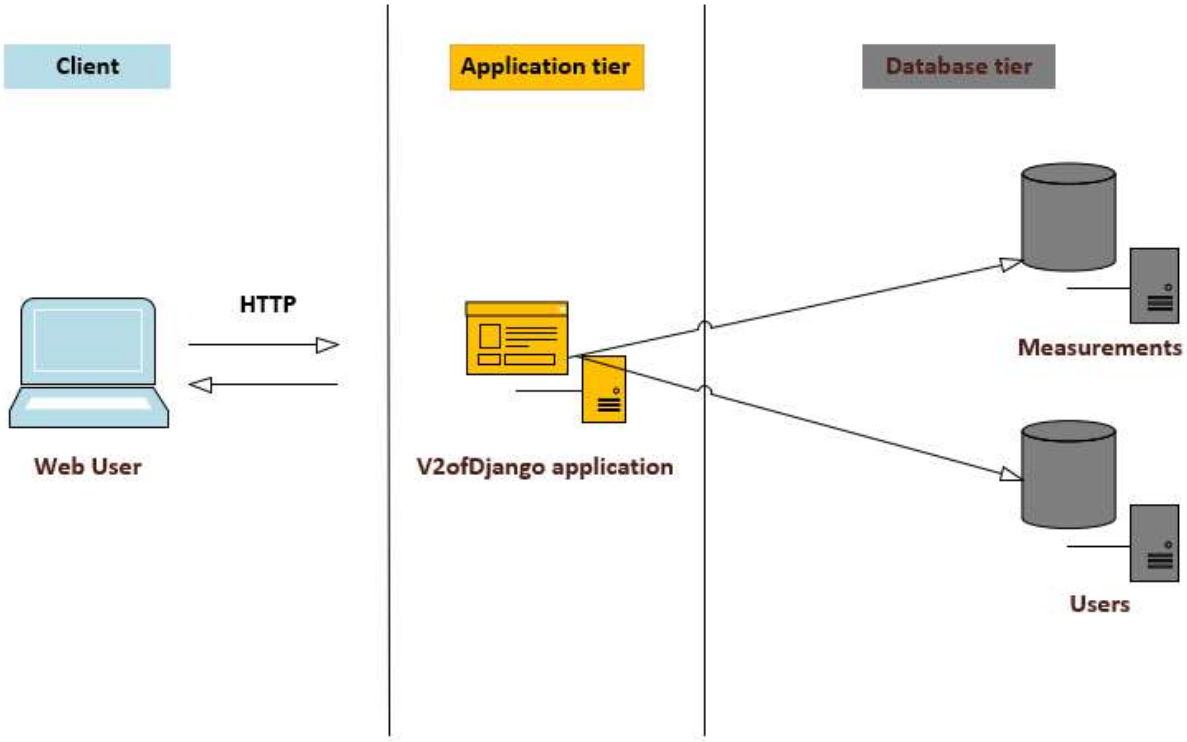
Το schema της βάσης Measurements δεν αλλάζει στην νέα υλοποίηση, συνεπώς το παρακάτω E-R διάγραμμα παραμένει το ίδιο στην εφαρμογή των μικροϋπηρεσιών.




Σχ. 6. E-R διάγραμμα της measurements βάσης

## 6.2 Layered Architecture Diagram

Όπως είναι φανερό και από την παρακάτω απεικόνιση τα στοιχεία της εφαρμογής που αφορούν στην παρουσίαση των δεδομένων (presentation layer), την επιχειρηματική λογική (business layer) καθώς και τα στοιχεία σύνδεσης με τη βάσης (data access layer) βρίσκονται όλα σε ένα deployable component, στο σφιχτά δεμένο μονόλιθο, ο οποίος είναι εγκατεστημένος στο application tier.




Σχ. 7.Αρχιτεκτονική απεικόνιση υπάρχουσας λύσης. Μονολιθική εφαρμογή

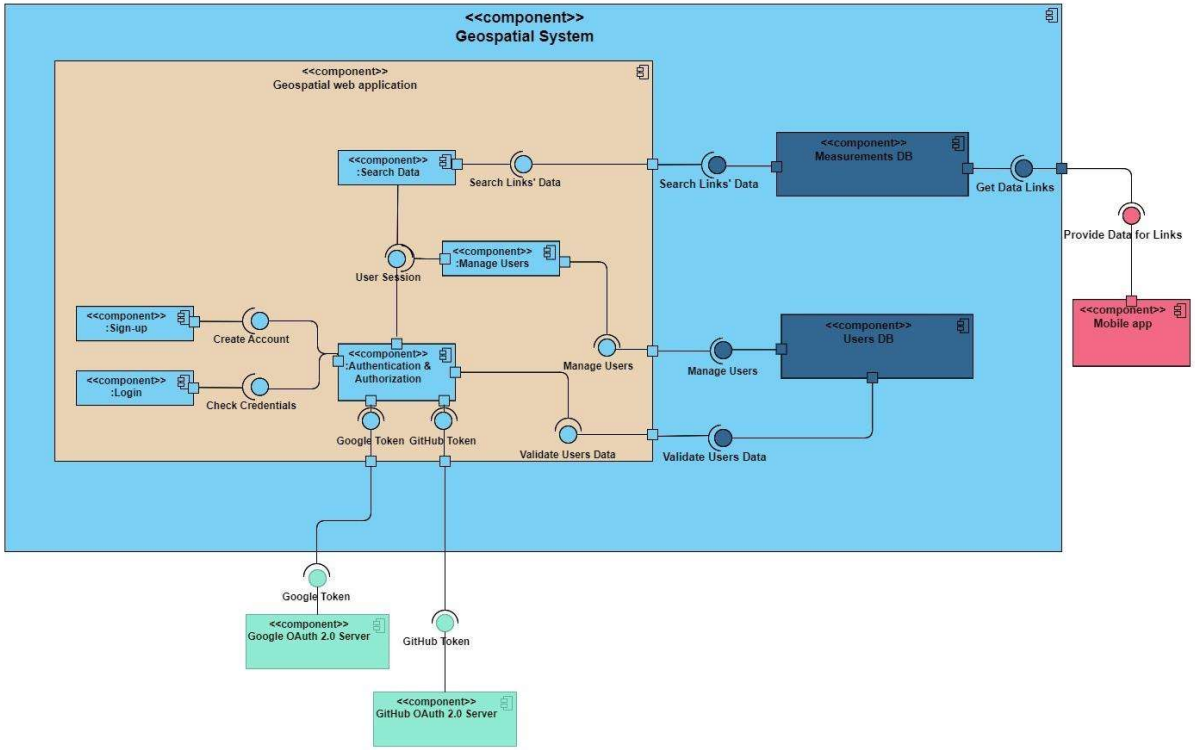
## 6.3 Διαγράμματα συνιστωσών

Τα διαγράμματα συνιστωσών παρέχουν τις δομές μοντελοποίησης που απαιτούνται για την απεικόνιση της φυσικής δομής του συστήματος. Περιέχουν πληροφορίες σχετικά με τα εκτελέσιμα αρχεία, τις βιβλιοθήκες που απαιτούνται για να εκτελεστεί κ.λπ.

- Οι συνιστώσες του πηγαίου κώδικα (εξαρτήσεις μεταγλώττισης) απεικονίζουν τις εξαρτήσεις μεταγλώττισης μεταξύ των αρχείων πηγαίου κώδικα
- Οι συνιστώσες εκτέλεσης (εξαρτήσεις κατά την εκτέλεση) δείχνουν την αντιστοιχία των κλάσεων σε βιβλιοθήκες (π.χ. αρχεία jar) καθώς και τις εξαρτήσεις με άλλες βιβλιοθήκες.

Υπάρχουν δύο τρόποι αναπαράστασης των εξαρτημάτων: *διαφανής (white box)* όπου φαίνεται η εσωτερική δομή των εξαρτημάτων και *αδιαφανής (black box)* που δε φαίνεται και φαίνονται μόνο οι δημόσιες διεπαφές του εξαρτήματος.

Το σύστημα αποτελείται από την γεωχωρική μονολιθική εφαρμογή και τις postgresql βάσεις Measurements και Users. Η βάση Measurements ενημερώνεται με δεδομένα από μια εξωσυστημική mobile εφαρμογή.




Σχ. 8. Διάγραμμα συνιστωσών της μονολιθικής εφαρμογής

# Κεφ.7 Υπό-ανάπτυξη εφαρμογή μικροϋπηρεσιών

Οι λειτουργικές απαιτήσεις του υπάρχοντος συστήματος, όπως προαναφέρθηκε δεν αλλάζουν στο νέο σύστημα που θα δημιουργηθεί.

## 7.1 Βήματα μετάβασης

Η αλλαγή αφορά μόνο στην αρχιτεκτονική του και στην τεχνολογία που θα πρέπει να χρησιμοποιηθεί. Το Django Framework δεν προορίζεται για μικροϋπηρεσίες, αλλά για μονολιθικά δεμένες εφαρμογές. Κατάλληλο Framework της Python για αυτή τη μετάβαση είναι ένα από τα:

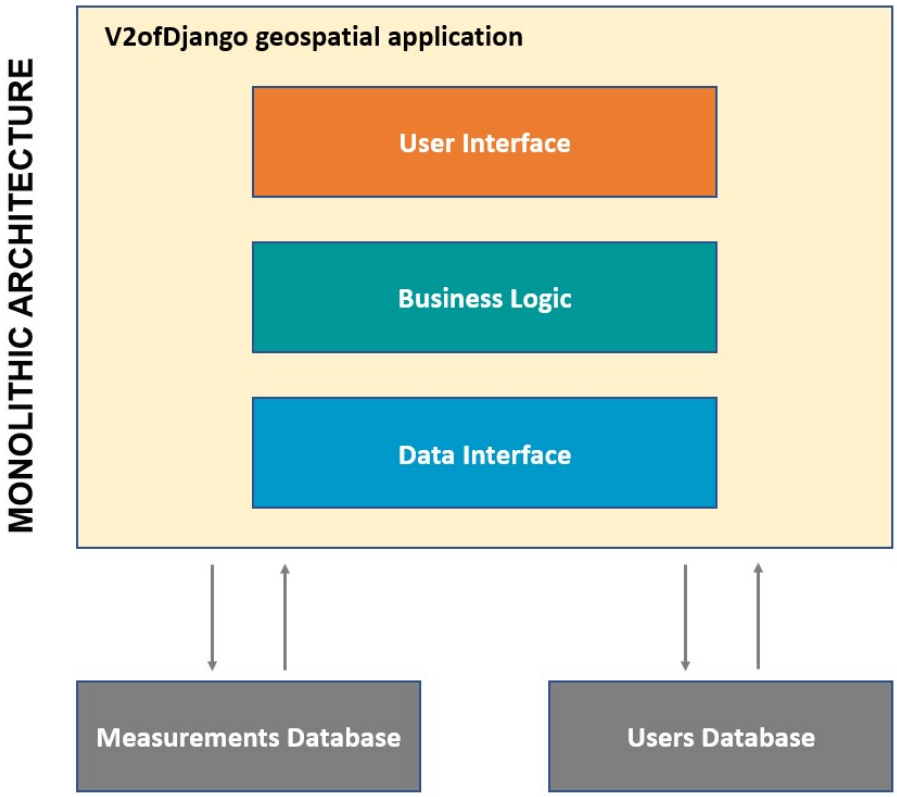
- Flask: Το πιο δημοφιλές Python Micro framework βασίζεται στα Jinja2 και Werkzeug
- Falcom: Διευκολύνει στη δημιουργία έξυπνων proxies, cloud APIs και app back-ends
- Bottle: Απλό, ελαφρύ και γρήγορο WSGI micro framework
- Nameko: Το καλύτερο ανάμεσα στα Python Microservices frameworks το οποίο επιτρέπει τους προγραμματιστές να επικεντρωθούν στην επιχειρηματική λογική της εφαρμογής (application logic).
- CherryPy: Ώριμο, Python object-oriented web framework

Επιπλέον, η γεωχωρική εφαρμογή δεν υποστηρίζει κάποια καίριας σημασίας λειτουργικότητα σε production περιβάλλον. Για αυτό το λόγο καταλληλότερη μεθοδολογία μετάβασης είναι να ξαναγραφτεί όλη η εφαρμογή με τη χρήση διαφορετικού framework της Python με το νέο αρχιτεκτονικό στυλ. Ωστόσο, ο διαχωρισμός της εφαρμογής σε περιέκτες όπως περιγράφεται παρακάτω έγινε βάσει της αποσύνθεσης σε υπηρεσίες κατά Hölttä-Ottoet et al.

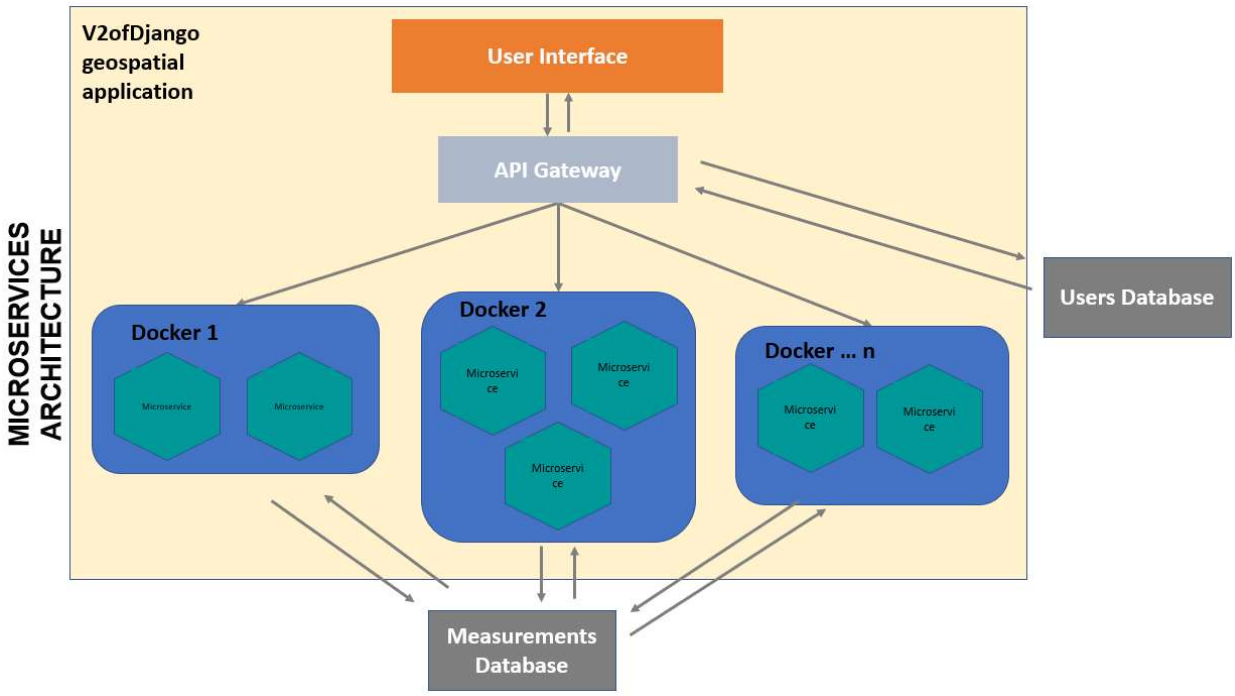
Στις παρακάτω εικόνες απεικονίζεται ένα high level της προσέγγισης διαχωρισμού. Οι μικροϋπηρεσίες θα μοιραστούν σε docker ανά ομάδες.

Οι βάσεις θα χρησιμοποιηθούν ως έχουν:

- Η **Users** θα περιέχει τα δεδομένα των χρηστών και
- Η **Measurements** τα στοιχεία που αφορούν στις μετρήσεις.




Σχ. 9. High level αρχιτεκτονική απεικόνιση της μονολιθικής εφαρμογής




Σχ. 10. High level αρχιτεκτονική απεικόνιση της εφαρμογής μικροϋπηρεσιών

Στην παράγραφο 7.1 περιγράφεται αναλυτικά η διαδικασία αποσύνθεσης της εφαρμογής σε μικροϋπηρεσίες, τα κριτήρια επιλογής της όλης διαδικασίας, καθώς και η ομαδοποίησή τους σε docker containers.

Η εφαρμογή των μικροϋπηρεσιών θα φιλοξενηθεί σε 4 dockers συνολικά.

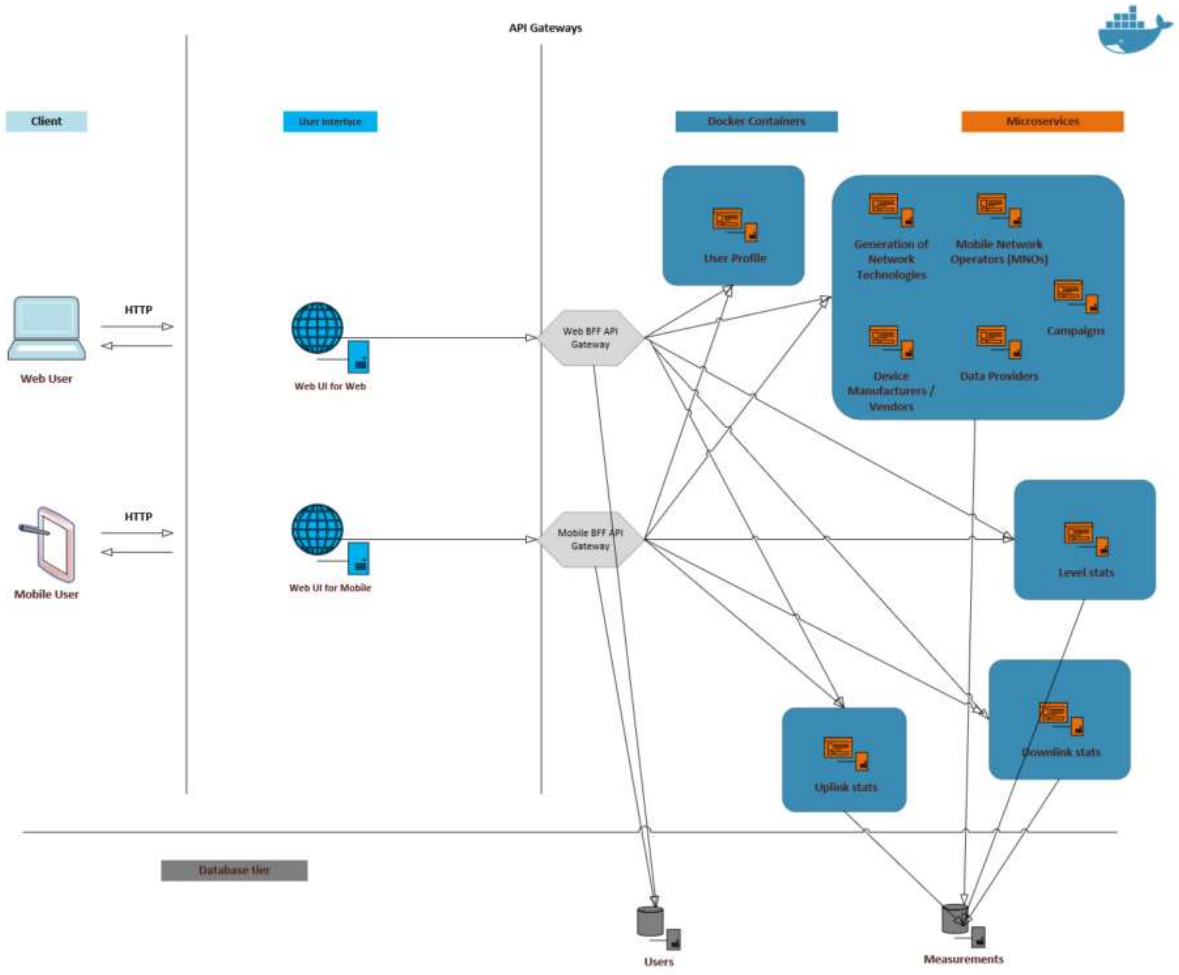
Στον πρώτο περιέκτη θα περιέχονται οι μικροϋπηρεσίες που αφορούν σε αναζήτηση δεδομένων των παρόχων, των τεχνολογιών, των δικτύων, των περιοχών ενδιαφέροντος προς μέτρηση (καμπάνιες), κ.ά.

Στο δεύτερο docker θα περιλαμβάνονται οι μικροϋπηρεσίες που αφορούν στην αναζήτηση δεδομένων της ζεύξης. Τα δεδομένα αφορούν στην ισχύ του σήματος, σε συνδυασμό με τον τύπο του δικτύου (2G, 3G, 4G) καθώς και το εύρος της ημερομηνίας.

Στον τρίτο περιέκτη θα περιλαμβάνονται οι μικροϋπηρεσίες που αφορούν στην αναζήτηση των δεδομένων της ανοδικής ζεύξης, και τέλος στον τέταρτο περιέκτη θα περιλαμβάνονται οι μικροϋπηρεσίες που αφορούν στην αναζήτηση δεδομένων της καθοδικής ζεύξης.

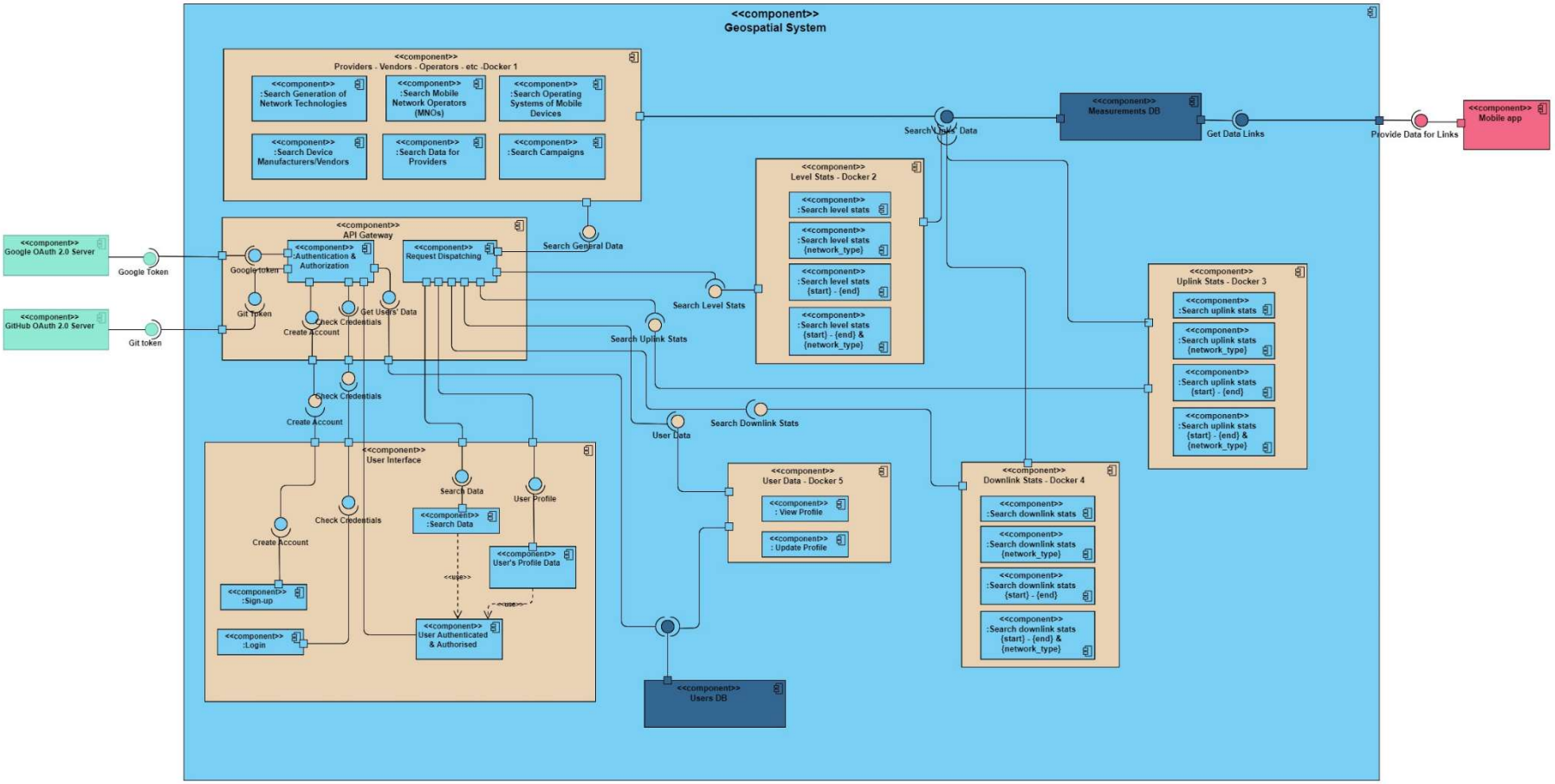
## 7.2 Σχεδίαση

### 7.2.1 Layered Architecture Diagram




Σχ. 11. Αρχιτεκτονική απεικόνιση προτεινόμενης λύσης. Microservices implementation

### 7.2.2 Component Diagram




Σχ. 12. Διάγραμμα συνιστωσών της εφαρμογής μικροϋπηρεσιών

Όπως φαίνεται και από το παραπάνω διάγραμμα προτείνεται η αποσύνθεση της μονολιθικής εφαρμογής στις εξής μονάδες που θα φιλοξενηθούν από τα αντίστοιχα Dockers ή από το VM:

Πίν. 22. Οι μικροϋπηρεσίες της γεωχωρικής εφαρμογής σε dockers

| Docker | Στοιχεία | Microservices |
|---|---|---|
| VM | front-end | View Data<br>View Users Profile |
| VM | API Gateway | Authenticate Users |
| Docker 1 | Providers - Vendors - Operators - etc | • Search Generation of Network Technologies<br>• Search Mobile Network Operators (MNOs)<br>• Search Operating Systems of Mobile Devices<br>• Search Device Manufacturers / Vendors<br>• Search Data for Providers<br>• Search Campaigns |
| Docker 2 | Level Stats | • Search level stats<br>• Search level stats {network_type}<br>• Search level stats {start} - {end}<br>• Search level stats {start} - {end} & {network_type} |
| Docker 3 | Uplink Stats | • Search uplink stats<br>• Search uplink stats {network_type} |

| | | |
|---|---|---|
| | | • Search uplink stats {start} - {end}<br>• Search uplink stats {start} - {end} & {network_type} |
| Docker 4 | Downlink Stats | • Search downlink stats<br>• Search downlink stats {network_type}<br>• Search downlink stats {start} - {end}<br>• Search downlink stats {start} - {end} & {network_type} |
| Docker 5 | User Profile | • View Profile<br>• Update Profile |
| VM | Postgresql Βάσεις | measurements, users |

Η αποσύνθεση των στοιχείων στις παραπάνω μικροϋπηρεσίες και η φιλοξενία τους σε docker ανά ομάδες βασίστηκε στα παρακάτω κριτήρια:

- Πλήθος requests χρηστών ανά μικροϋπηρεσία
- Κατηγοριοποίηση βάσει της επιχειρηματικής λογικής

Οι βάσεις δεν θα φιλοξενηθούν σε containers αλλά στο VM.

Το front-end δεν θα ακολουθήσει την micro-UI αρχιτεκτονική, αλλά θα τηρηθεί ως front-end μονόλιθος, όπως απεικονίστηκε και στο αρχιτεκτονικό διάγραμμα.

## 7.3 Διαδικασία εγκατάστασης υφιστάμενης υλοποίησης για την ανάλυσή της

Παρακάτω περιγράφονται τα βήματα για τη δημιουργία του Django Geospatial Web Application σε μια εικονική μηχανή, όπου έγινε η εγκατάσταση του λειτουργικού συστήματος Linux Ubuntu 20.04.4 LTS (Focal Fossa).

Η δημιουργία και τη διαχείριση της εικονικής μηχανής που στήθηκε έγινε με τη βοήθεια του υπερόπτη ανοιχτού κώδικα Oracle VM VirtualBox.

Για την ολοκλήρωση της εφαρμογής εκτελέστηκαν τα παρακάτω βήματα:

- **Εγκατάσταση της γλώσσας προγραμματισμού Python**

Έλεγχος της python έκδοσης που είναι εγκατεστημένη στο pc του developer:

**$ python3 -V**

*Python 3.8.10*

Στα περισσότερα Linux περιβάλλοντα η Python έρχεται προεγκατεστημένη. Στο link: https://www.python.org/downloads/ μπορούν να βρεθούν περισσότερες πληροφορίες για τα περιβάλλοντα εγκατάστασης και τις εκδόσεις της γλώσσας.

- **Δημιουργία folders μέσα στο οποίο θα γίνει η ανάπτυξη της εφαρμογής**

Με τη βοήθεια της mkdir εντολής δημιουργήθηκε το παρακάτω path στο Home directory

~/Projects/Thesis$

- **Δημιουργία κλώνου (αντιγράφου) από το ήδη υπάρχον αποθετήριο του κώδικα της μονολιθικής γεωχωρικής εφαρμογής που θα σπάσει σε microservices**

$ git clone https://github.com/mKorniotakis/V2ofDjango

- **Δημιουργία ενός εικονικού περιβάλλοντος (virtual environment)**

Μέσα στον φάκελο V2ofDjango (στο ίδιο επίπεδο με το mysite) δημιουργείται ένα εικονικό περιβάλλον με την εντολή:

**$ python3 -m venv msaVenv**

Έπειτα γίνεται η πλοήγηση μέσα σε αυτό και η ενεργοποίησή του σύμφωνα με τις εντολές:

**$cd msaVenv**

**$source bin/activate**

Μετά την ενεργοποίηση του εικονικού περιβάλλοντος η τρέχουσα γραμμή εντολών του bash αρχίζει με το λεκτικό **(msaVenv)**. Αυτό σημαίνει ότι ορίζεται αυτομάτως ως **Environmental Path** το μονοπάτι-διαδρομή του συγκεκριμένου εικονικού περιβάλλοντος μέσα από το οποίο γίνεται η ενεργοποίηση.

- **Εγκατάσταση των dependencies του Project**

Οι απαιτήσεις της εφαρμογής βρίσκονται στο requirements.txt αρχείο (στο ίδιο επίπεδο με το εικονικό περιβάλλον) και για την εγκατάστασή τους εκτελείται η εντολή:

**(msaVenv) $ pip3 install -r requirements.txt**

Από τα error που προέκυψαν έγιναν upgrade ή εγκαταστάθηκαν τα παρακάτω πακέτα (σε κάθε χρήστη ανάλογα με το τι είναι εγκατεστημένο στο λειτουργικό του αυτά οι ενημερώσεις δύναται να διαφέρουν):

**$ pip3 install wheel**

**$ sudo apt install libjpeg-dev zlib1g-dev**

- **Τρέξιμο εφαρμογής**

Θα πρέπει να γίνει η πλοήγηση μέσα στο mySite φάκελο και να τρέξει η παρακάτω εντολή:

**(msaVenv) python3 manage.py runserver**

Από το error που προκύπτει στο αρχείο settings.py μπήκε σε σχόλια η γραμμή 20 και έγινανα uncommented οι 21 και 18.

- **Επίλυση προβλημάτων**

Το επόμενο error αφορά στο GDAL_LIBRARY_PATH

**$ sudo apt-get install gdal-bin**

**$ sudo apt-get install libgdal-dev**

Για την επίλυση του επόμενου error χρειάστηκε να γίνει downgrade της έκδοσης του MarkupSafe στο virtual environment:

**(msaVenv) $ pip3 install MarkupSafe==2.0.1**

Εγκατάσταση της Postgresql σχεσιακής βάσης δεδομένων

**$ sudo apt install postgresql postgresql-contrib**

Κατά την εγκατάσταση, η Postgres έχει ρυθμιστεί ώστε να χρησιμοποιεί έλεγχο ταυτότητας, που σημαίνει ότι συσχετίζει ρόλους Postgres με έναν αντίστοιχο λογαριασμό συστήματος Unix/Linux. Εάν υπάρχει ένας ρόλος στο Postgres, ένα όνομα χρήστη Unix/Linux με το ίδιο όνομα μπορεί να εισέλθει ως αυτός ο ρόλος.

Η διαδικασία εγκατάστασης δημιούργησε έναν λογαριασμό χρήστη που ονομάζεται postgres που σχετίζεται με τον προεπιλεγμένο ρόλο Postgres. Υπάρχουν μερικοί τρόποι για να χρησιμοποιηθεί αυτός ο λογαριασμός για πρόσβαση στο Postgres. Ένας τρόπος είναι η μετάβαση στον λογαριασμό postgres του διακομιστή εκτελώντας την ακόλουθη εντολή:

**$ sudo -i -u postgres**

Στη συνέχεια, μπορεί κανείς να αποκτήσει πρόσβαση στην Postgres εκτελώντας:

**$ psql**

Αυτή η εντολή συνδέει το χρήστη στη γραμμή εντολών PostgreSQL και στη συνέχεια μπορεί να αλληλεπιδράσει με το σύστημα διαχείρισης της βάσης δεδομένων.

Για έξοδο από την προτροπή PostgreSQL, αρκεί η εκτέλεση της εντολής:

**postgres=# \q**

Η εντολή αυτή οδηγεί πίσω στη γραμμή εντολών του postgres Linux. Για επιστροφή στον κανονικό χρήστη του συστήματός σας, θα πρέπει να εκτελεστεί η εντολή εξόδου:

**postgres@server:~$ exit**

Ένας άλλος τρόπος για να συνδεθεί κάποιος στη γραμμή εντολών Postgres είναι να εκτελέσει την εντολή psql ως λογαριασμό postgres απευθείας με το sudo:

**$ sudo -u postgres psql**

Με αυτό τον τρόπο ο χρήστης συνδέεται απευθείας στην Postgres χωρίς το ενδιάμεσο κέλυφος bash.

Και πάλι, η έξοδος από τη διαδραστική περίοδο λειτουργίας Postgres ολοκληρώνεται με την εκτέλεση της εντολής:

**postgres=# \q**

- **Δημιουργία measurements βάσης**

**postgres=# create database measurements;**

Θα πρέπει να εγκατασταθεί το postgis extension της postgresql βάσης θα πρέπει να γίνουν τα δυο παρακάτω βήματα:

a) Να εγκατασταθούν τα παρακάτω postgis packages στο λειτουργικό και

**$ sudo apt install postgis postgresql-12-postgis-3**

**$ sudo apt install postgis postgresql-12-postgis-3-scripts**

b) Να εγκατασταθεί το postgis και το hstore extension στην measurements βάση.

**$ psql -d measurements;**

**measurements=# CREATE EXTENSION postgis;**

**CREATE EXTENSION**

**measurements=# CREATE EXTENSION hstore;**

**CREATE EXTENSION**

- **Propagate των μοντέλων των πινάκων στη βάση**

Για να γίνουν propagate οι Πίνακες στη βάση, θα πρέπει να τρέξει η εντολή migrate η οποία λαμβάνει υπόψη της όλα τα migrations τα οποία δεν έχουν εφαρμοστεί (το Django γνωρίζει ποια migrations έχουν εφαρμοστεί χρησιμοποιώντας έναν ειδικό πίνακα στην βάση δεδομένων υπό το όνομα django_migrations) και τα τρέχει ένα-ένα (στην ουσία, πραγματοποιεί ένα είδους συγχρονισμό μεταξύ των αλλαγών των μοντέλων και του schema της βάσης).

Επειδή στο project είναι stored τα αρχεία που έχουν να κάνουν με το migration, θα πρέπει να γίνουν τα εξής:

1) Delete τα files που είναι στο mysite/myapp/migrations
2) Αλλαγή του property managed = True σε όσα μοντέλα του models.py δεν αφορά σε Django admin tables
3) Τρέξιμο της εντολής:

**(msaVenv) $ python3 manage.py makemigrations (δύο φορές, σ.σ. Τη δεύτερη φορά θα φτιαχτούν τα migration files των πινάκων που είναι managed -το process θα πρέπει να εκτελεστεί σε καινούργιο virtual environment-).**

4) Και στη συνέχεια:

**(msaVenv) $ python3 manage.py migrate**

*Operations to perform:*

*Apply all migrations: admin, auth, contenttypes, sessions*

*Running migrations:*

*Applying contenttypes.0001_initial... OK*

*Applying auth.0001_initial... OK*

*Applying admin.0001_initial... OK*

*Applying admin.0002_logentry_remove_auto_add... OK*

*Applying admin.0003_logentry_add_action_flag_choices... OK*

*Applying contenttypes.0002_remove_content_type_name... OK*

*Applying auth.0002_alter_permission_name_max_length... OK*

*Applying auth.0003_alter_user_email_max_length... OK*

*Applying auth.0004_alter_user_username_opts... OK*

*Applying auth.0005_alter_user_last_login_null... OK*

*Applying auth.0006_require_contenttypes_0002... OK*

*Applying auth.0007_alter_validators_add_error_messages... OK*

*Applying auth.0008_alter_user_username_max_length... OK*

*Applying auth.0009_alter_user_last_name_max_length... OK*

*Applying auth.0010_alter_group_name_max_length... OK*

*Applying auth.0011_update_proxy_permissions... OK*

*Applying auth.0012_alter_user_first_name_max_length... OK*

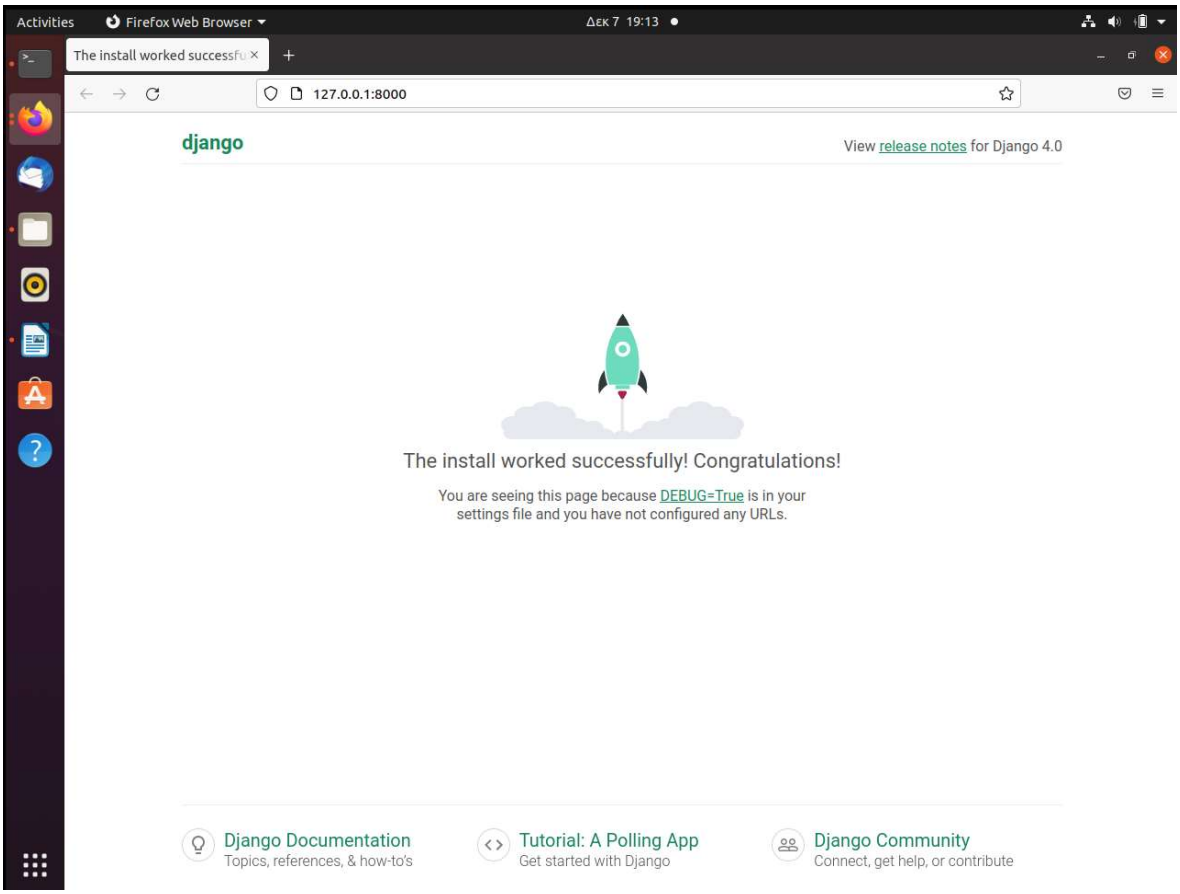
*Applying sessions.0001_initial... OK*

- **Ενεργοποίηση του django development server**

**(msaVenv) python3 manage.py runserver**

*Starting development server at http://127.0.0.1:8000/*

Στην διαδρομή (url) http://127.0.0.1:8000/ ενός φυλλομετρητή (browser) μπορεί κανείς να πληροφορηθεί ότι έχει εγκατασταθεί επιτυχώς ένα django project.




Εικ. 34. The default Django website

Ανοίγοντας σε ένα browser τα δυο παρακάτω urls, ο χρήστης έχει πλέον πρόσβαση στην εφαρμογή και στη διαχειριστική σελίδα της, αντίστοιχα:

*http://127.0.0.1:8000/myapp*

*http://127.0.0.1:8000/admin*

- **Δημιουργία superuser για πρόσβαση στην κονσόλα του admin.**

**(msaVenv) $python3 manage.py createsuperuser**

Username: admin.

Email address: admin@example.com

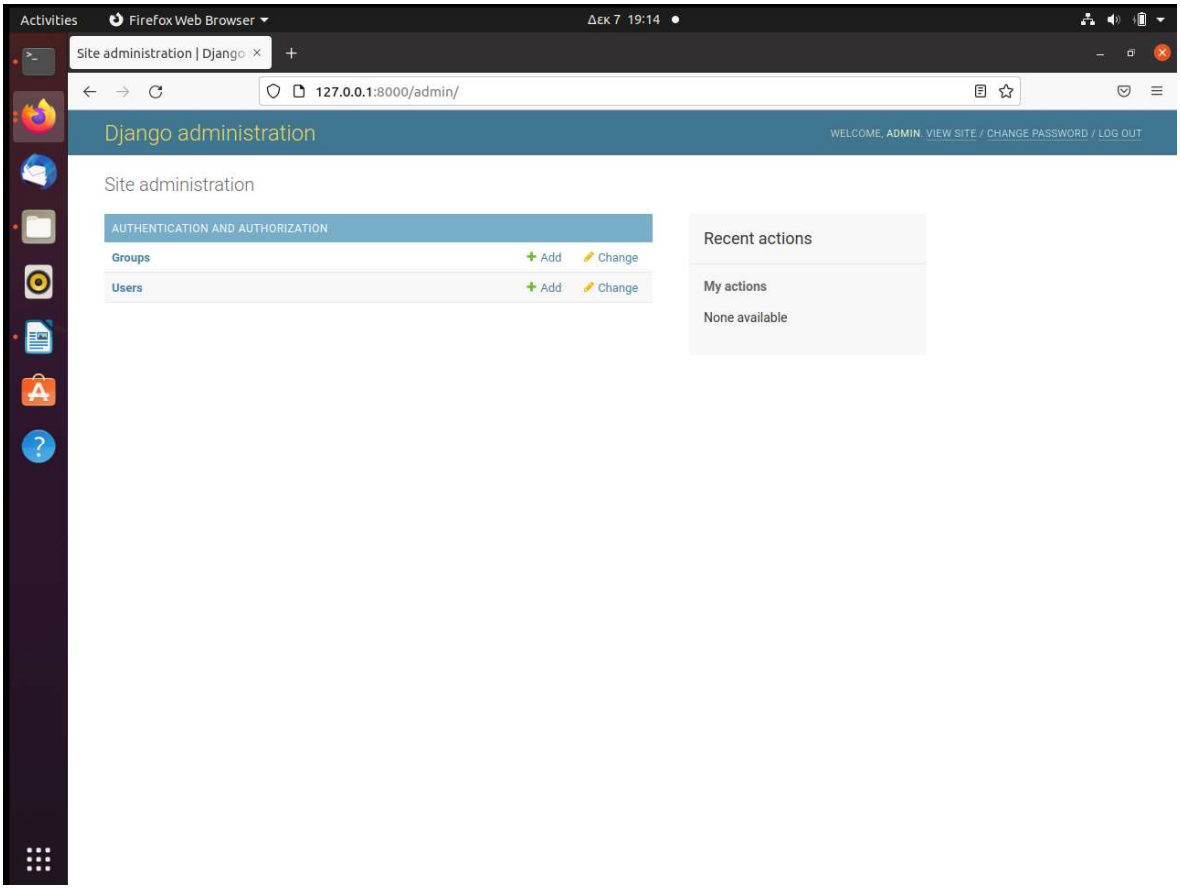
Password: ********

Password (again): ********

Superuser created successfully.

Ο χρήστης δημιουργείται στον πίνακα public.auth_user της measurements βάσης.

Επίσης ο χρήστης με τα credentials του superuser μπορεί να έχει πρόσβαση στην παρακάτω οθόνη:




Εικ. 35. The admin Django webpage

- **Μεταφορά του πηγαίου κώδικα σε άλλο αποθετήριο**

Μετά το επιτυχημένο τρέξιμο της εφαρμογής έγιναν τα κατάλληλα βήματα, ώστε ο πηγαίος κώδικας (sourcecode) να μεταφερθεί σε άλλο αποθετήριο στο σύστημα διαχείρισης εκδόσεων λογισμικού GitHub προκειμένου να γίνει μελλοντικά η κατάτμηση της εφαρμογής σε μικροϋπηρεσίες.

Ο κώδικας μπορεί πλέον να βρεθεί στο παρακάτω link:

**https://github.com/tDamala/msaGeospatialApp.git**

# ΒΙΒΛΙΟΓΡΑΦΙΑ


Adrian, A. (2019, September). *Design patterns in microservices for CTOs: API Gateway, Backend for Frontend and more*. Poland: The Software House. Retrieved from: https://tsh.io/blog/design-patterns-in-microservices-api-gateway-bff-and-more/

Al-Debagy, O., & Martinek, P., (2018). *A Comparative Review of Microservices and Monolithic Architectures*. [Symposium Paper].

Al-Kofahi, M. (2011, January). *Service Oriented Architecture (SOA) Security Models*.

Alanazi, S., Abdullah, N., Mohammed, A., & Al-wesabi, O. (2019). *Evaluation Approaches of Service Oriented Architecture (SOA) - A Survey*.

Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M., & Steinder, M. (2015). *Performance Evaluation of Microservices Architectures using Containers.*

Amundsen, M. et al. (2016). *Microservice Architecture*. O'Reilly.

Anil, N., Veloso, M., Parente, J., Mang, A., & Wenzel, M. (2021 September). *Creating composite UI based on microservices.* United States: Microsoft. Retrieved from: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/microservice-based-composite-ui-shape-layout

Aksit, M., Tekinerdogan, B., & Lodewijk, B. (2001 January). *The Six concerns for Separation of Concerns*. Conference: Workshop on Advanced Separation of Concerns (ECOOP 2001).

Arundel, J., & Domingus, J. (2019, March). *Cloud Native DevOps with Kubernetes Building, Deploying, and Scaling Modern Applications in the Cloud.* O'Reilly.

Axelrod, A. (2018). *Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects.* Apress.

Auera, F., Lenarduzzi, V., Feldererac, M., & Taibid, D. (2021, September). *From monolithic systems to Microservices: An assessment framework.* [Journal] Information and Software Technology, Vol. 137. Retrieved from: https://doi.org/10.1016/j.infsof.2021.106600.

Bajaj, D., Bharti, U., Goel, A., & Gupta, S. C. (2020). *Partial Migration for Re-architecting a Cloud Native Monolithic Application into Microservices and FaaS*.

Badger, L., Grance, T., Patt-Corner, R., & Voas, J. (2012, May). *Cloud Computing Synopsis and Recommendations.* Recommendations of the National Institute of Standards and Technology.

Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice.*

Beck, K., & Andres, C. (2004, September). *Extreme Programming Explained: Embrace Change.* 2nd Edition (The XP Series).

Beydoun, G., Xu, D., & Sugumaran, V. (2013). *Service Oriented Architectures (SOA) Adoption Challenges Service Oriented Architecture (SOA) Adoption Challenges.*

Black, D. U. (1996). *OSI: A Model for Computer Communications Standards.*

Bozan, K., Lyytinen, K., & Rose, M. G. (2021, January). *How to Transition Incrementally to Microservice Architecture,* [Journal] Communications of the ACM, Vol. 64 No. 1.

Cloud Native GeoServer (2022). *Dockerized GeoServer micro-services.* [Online]. Retrieved from: http://geoserver.org/geoserver-cloud/.

D. Hardt, Ed. (2012, October). The OAuth 2.0 Authorization Framework. Retrieved from: https://datatracker.ietf.org/doc/html/rfc6749.

Docker (2022). *Docker - build, ship, and run any app, anywhere*. [Online]. Retrieved from: https://www.docker.com/.

Duan, Y.,Feng, W., Zhou, X., Dong, L., Hu, Y., & Gao, H. (2016). *Exploring the Categories and Models of Everything as a Service (XaaS).* International Journal of Grid Distribution Computing, Vol. 8, No.6, (2015), pp.215-228. Retrieved from: http://dx.doi.org/10.14257/ijgdc.2015.8.6.21

Ernst, E. (2003 January). Separation of concerns.

Esfandyari, A., & Barati, M. (2012). *A new semantic service discovery in service oriented architecture (SOA)*.

Fritzsch, J. (2018). *From Monolithic Applications to Microservices: Guidance on Refactoring Techniques and Result Evaluation*. [Master Thesis].

Garlan, D., & Shaw, M. (1994 January). *An Introduction to Software Architecture.*

Gamma, E., Richard Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st Edition.

Geers, M. (2020). *Micro Frontends in Action.* Manning Publications.

Geers, M. (2021). *Micro Frontends extending the microservice idea to frontend development.* Retrieved from: https://micro-frontends.org/

GitHub, Inc., (2022). *Authorizing OAuth Apps.* Retrieved from: https://docs.github.com/en/developers/apps/building-oauth-apps/authorizing-oauth-apps.

Goniwada, R. S. (2021, October). *Cloud Native Architecture and Design: A Handbook for Modern Day Architecture and Design with Enterprise-Grade Examples*. O'Reilly.

Google Developers. (2021, December). *Using OAuth 2.0 to Access Google APIs.* Retrieved from: https://developers.google.com/identity/protocols/oauth2.

Gos, K., & Zabierowski, W. (2020, April). *The Comparison of Microservice and Monolithic Architecture*. [Conference Paper].

Haselböck, S., & Weinreich, R. (2017). *Decision guidance models for microservice monitoring. Proceedings - 2017 IEEE International Conference on Software Architecture Workshops.*

Hassan, S., Bahsoon, R., & Kazman, R. (2020, February 19). *Microservice transition and its granularity problem: A systematic mapping study*. [Survey Paper].

Havey, M. (2008). *SOA Cookbook - Design Recipes for Building Better SOA Processes*. Packt Publishing.

Hassan, B. H., Barakat, A. S., & Sarhan I. Q. (2021). *Survey on serverless computing. Journal of Cloud Computing: Advances, Systems and Applications*. Retrieved from: https://doi.org/10.1186/s13677-021-00253-7

Hölttä-Otto, K., Chiriac, N., Lysy, D., & Suh, E., S. (2013, July 13). *Architectural decomposition: The role of granularity and decomposition viewpoint*.

Ingeno, J. (2018, August). *Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts*. Packt Publishing Limited.

Jaakkola, H., & Thalheim, B. (2010). *Architecture-Driven Modelling Methodologies.*

*Jackson, C. (2019 June). Micro Frontends.* Retrieved from*:* https://martinfowler.com/articles/micro-frontends.html

Krause, R. (2021, January). *Maintain Consistency and Adhere to Standards.*

Kruchten, P. (1999 February). *The Software Architect.*

Kruchten, P. (2008 December). *What do software architects really do?* [Journal] Journal of Systems and Software, Vol. 81. Issue 12, Retrieved from: https://doi.org/10.1016/j.jss.2008.08.025

Kruchten, P., & Obbink, H. (2006 April). *The Past, Present and Future for Software Architecture*

Kubernetes. (2020). *Kubernetes documentation*. [Online]. Retrieved from: https://kubernetes.io/docs/home/.

Kwiecien, A. (2019, August 2).*10 companies that implemented the microservice architecture and paved the way for others.* Retrieved from: https://www.divante.com/blog/10-companies-that-implemented-the-microservice-architecture-and-paved-the-way-for-others.

Liu, Z., Jifeng, H., & Li, X. (2004, January). *Contract Oriented Development of Component Software.* In book: *Exploring New Frontiers of Theoretical Informatics.* DOI: 10.1007/1-4020-8141-3_28

Loshin, D. (2010, July). *Master Data Management.* Kaufmann.

McClain, P. B. (2022). *Python for Geospatial Data Analysis. Theory, Tools, and Practice for Location Intelligence.* O'Reilly.

Mäkitalo, N., & Mikkonen, T. (2018). *Challenges When Moving from Monolith to Microservice Architecture*.

Martin, C. R. (2018). *Clean Architecture: A Craftsman's Guide to Software Structure and Design.*

Mella, F. P., Márquez, G., & Astudillo, H. (2019, September). *Migrating from monolithic architecture to microservices: A Rapid Review*. [Conference Paper].

Microsoft Patterns & Practices Team (2009). *Microsoft® Application Architecture Guide, 2nd Edition (Patterns & Practices).* Retrieved from: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff650706(v=pandp.10)?redirectedfrom=MSDN

Microsoft (2021, September 11). *Migrate a monolith application to microservices using domain-driven design*. Retrieved from: https://docs.microsoft.com/en-us/azure/architecture/microservices/migrate-monolith.

Minsky, N., & Pal, P. (1996, February). *Imposing The Law of Demeter and Its Variations.*

Müller, S. D. (2015, January). *Benefits of Cloud Computing: Literature Review in a Maturity Model Perspective*. Article in Communications of the Association for Information Systems. DOI: 10.17705/1CAIS.03742

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*.

Nils, J., Beimborn, D., & Weitzel, T. (2010). *Investigating Adoption Determinants of Service-Oriented Architectures (SOA)*.

Noback, M. (2018 January). *The Single Responsibility Principle: Creating Reusable Software Components.*

OAuth.io (2018, June). Choose The Right OAuth2 Flow/Grant Types For Your App. [Online]. Retrieved from: https://blog.oauth.io/choose-oauth2-flow-grant-types-for-app/.

OAuth.io (2018, September). OAuth2 Introduction Through Flow Diagrams in 5-minutes. [Online]. Retrieved from: https://blog.oauth.io/introduction-oauth2-flow-diagrams/.

Payrott, S. (2015, September). API Gateway. An Introduction to Microservices, Part 2. Learn about API gateways and how they work in a microservice-based architecture. Retrieved from: https://auth0.com/blog/an-introduction-to-microservices-part-2-API-gateway/

PostGIS. (2021). *About PostGIS*. [Online]. Retrieved from: https://postgis.net/

PostgreSQL. (2021). *What is PostgreSQL?* [Online]. Retrieved from: https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/

Pradhan, R., & Dash, K. A. (2020). *An Overview of Microservices*.

Reed, A. P., & Ritz, J. (2003, January). *Geospatial Technology.*

Richardson, C. (2020). *Microservice Architecture.* Retrieved from*:* http://microservices.io/patterns.

Richardson, C. (2017). *Microservice Patterns. Manning.*

Richardson, C., & Smith, F. (2016). *Microservices From Design to Deployment*.

Sainia, C. H., & Dr. Upadhyayab, A., Khandelwal, K. M. (2019 January). *Benefits of Cloud Computing for Business Enterprises: A Review.* International Conference on Advancements in Computing & Management (ICACM-2019)

Sampaio, R. A. Jr., Kadiyalax, H., Hux, B., Steinbachery, J., Erwinz, T., Rosa, N., Beschastnikhx, I., & Rubinx, J. (2017, September). *Supporting Microservice Evolution*.

Saransig, A., & Leon, F. T. (2019). *Performance analysis of Monolithic and Microservice architectures - Containers technology*.

Scheuch, R. (2015). *Risikominimierung bei der Applikations - modernisierung mittels Microservices.*

Sciore, E. (2018, December). *Java Program Design: Principles, Polymorphism, and Patterns.* Apress.

Sether, A. (2016, June). *Cloud Computing Benefits*. Retrieved from: DOI: 10.13140/RG.2.1.1776.0880

Shadija, D., Rezai, M., & Hill, R. (2017, September). *Towards an Understanding of Microservices*.

Sheikh, Al. A. M., Aboalsamh, H., & Albarrak, I. A. (2011). *Migration of legacy applications and services to Service-Oriented Architecture (SOA)*.

Singh, B. B. (2021). *Micro Frontends: Reinventing UI In The Microservices World.* Retrieved from: https://www.velotio.com/engineering-blog/micro-frontends-reinventing-ui-in-the-microservices-world.

Srinivasan A. (2014, January). *Cloud Computing*. O'Reilly.

Stenberg, J. (2014). *Moving from a Monolith to Microservices at SoundCloud.* Retrieved from: https://www.infoq.com/news/2014/06/soundcloud-microservices.

Sommervillle, I. (2016). *Software Engineering.* Pearson.

Taibi, D., & Systä, K. (2020, June). *Decomposition and Metric-Based Evaluation Framework for Microservices*.

Vayghana, A. L., Saiedb, M. A., Toeroec, M., & Khendeka, F. (2020, Jully). *A Kubernetes controller for managing the availability of elastic microservice based stateful applications*. Retrieved from: https://doi.org/10.1016/j.jss.2021.110924

Vernon V. (2015 January). *Implementing Domain-Driven Design.*

Viggiato, M., Terra, R., Rocha, H., Valente, T. M., & Figueiredo, E. (2018, August 14). *Microservices in Practice: A Survey Study*.

Visser, J. (2016, February). *Building Maintainable Software.* O'Reilly.

Wanga, D., Yanga, D., Zhoua, H., Wanga, Y., Honga, D., Donga, Q., & Songc, S. (2020). *A Novel Application of Educational Management Information System based on Micro Frontends.*

Wolf, O. (2019). *Serverless Architecture in Short.* Retrieved from: https://specify.io/concepts/serverless-baas-faas

Xu, D., & Sun, Z. (2021, August). *An adaptive function placement in serverless computing*. Retrieved from: https://doi.org/10.1007/s10586-021-03506-x(0123456789().,-volV)(0123456789().,-volV)

Woods, D., & Mattern, T. (2006). *Enterprise SOA: Designing IT for Business Innovation.* O'Reilly.

Youssef, E. A. (2012, July). *Exploring Cloud Computing Services and Applications.* Journal of Emerging Trends in Computing and Information Sciences, Retrieved from: http://www.cisjournal.org, VOL. 3, NO. 6, July 2012.

Διαγράμματα Δραστηριότητας. (2017, Μαίου 6). Διαθέσιμο στο: https://el.wikipedia.org/wiki/Διαγράμματα_δραστηριότητας. Βικιπαίδεια.