



ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

Πτυχιακή Εργασία

**Ανάπτυξη Εφαρμογών με Language Models:
Συγκριτική Μελέτη Μονολιθικής και Microservices
Αρχιτεκτονικής**

Σίντι Βρουσάι

AM 8210017

Επιβλέπων:

Παναγιώτης Λουρίδας

Αθήνα, Ιούλιος 2025



Με τη συγχρηματοδότηση
της Ευρωπαϊκής Ένωσης



Πρόγραμμα
Ανθρώπινο Δυναμικό και
Κοινωνική Συνοχή

Περίληψη

Η παρούσα πτυχιακή εργασία εξετάζει τη δυνατότητα αξιοποίησης Μεγάλων Γλωσσικών Μοντέλων (LLMs), όπως τα GPT-4, Claude Sonnet και Gemini, για την αυτόματη παραγωγή λογισμικού σε διαφορετικές αρχιτεκτονικές: μονολιθική και microservices. Μέσω μιας εμπειρικής μελέτης, κάθε μοντέλο κλήθηκε να υλοποιήσει μία REST API εφαρμογή και στις δύο αρχιτεκτονικές προσεγγίσεις, ώστε να συγκριθούν τα αποτελέσματά τους. Η αξιολόγηση βασίστηκε τόσο σε ποιοτικά χαρακτηριστικά του παραγόμενου κώδικα όσο και σε μετρικές στατικής ανάλυσης μέσω του εργαλείου SonarQube.

Τα αποτελέσματα ανέδειξαν πως τα LLMs αποδίδουν ιδιαίτερα καλά σε μονολιθικές υλοποιήσεις, παρουσιάζοντας υψηλό επίπεδο κατανόησης της ζητούμενης δομής και παραγωγή καθαρού, οργανωμένου κώδικα. Αντιθέτως, στο πλαίσιο των microservices παρατηρήθηκαν περισσότερες προκλήσεις — κυρίως στον διαχωρισμό ευθυνών και στη σωστή οργάνωση των υπηρεσιών. Ωστόσο, τα μοντέλα απέδειξαν ότι μπορούν να ανταποκριθούν επαρκώς, εφόσον δοθεί κατάλληλη καθοδήγηση. Παράλληλα, πραγματοποιήθηκε και συγκριτική αξιολόγηση μεταξύ των μοντέλων, από την οποία προέκυψαν ανταγωνιστικά πλεονεκτήματα για το καθένα.

Περιεχόμενα

1	Εισαγωγή	2
1.	Περιγραφή Θέματος	2
2.	Συνεισφορά	2
3.	Μεθοδολογία	3
2	Βιβλιογραφική Επισκόπηση	6
1.	Εισαγωγή στα Μεγάλα Γλωσσικά Μοντέλα	6
1.1	Τι είναι τα Μεγάλα Γλωσσικά Μοντέλα;	6
1.2	Ιστορική Αναδρομή	7
1.3	GPT-4	7
1.4	Gemini 2.5	8
1.5	Claude Sonnet 4	8
2.	Εισαγωγή στις Αρχιτεκτονικές Λογισμικού	8
2.1	Εισαγωγή	8
2.2	Μονολιθική Αρχιτεκτονική	9
2.3	Microservices Αρχιτεκτονική	9
2.4	Συγκριτική ανάλυση στο Πλαίσιο REST API σε Java	10
2.5	Συσχέτιση με LLMs	11
3	Πρακτικό Μέρος	12
1.	Περιγραφή Εφαρμογής	12
2.	Ανάπτυξη Εφαρμογής	13

2..1	GPT-4	13
2..2	Gemini	19
2..3	Claude-Sonnet	26
4	Ανάλυση Αποτελεσμάτων	33
1.	Συγκριτική ανάλυση ανάπτυξης εφαρμογών με μονολιθική και microser- vices αρχιτεκτονική	33
1..1	Επίδραση της αρχιτεκτονικής στην απόδοση του GPT-4	34
1..2	Επίδραση της αρχιτεκτονικής στην απόδοση του Gemini 2.5	40
1..3	Επίδραση της αρχιτεκτονικής στην απόδοση του Claude Sonnet 4	45
2.	Συγκριτική ανάλυση της απόδοσης των LLMs	50
2..1	Σημεία σύγκλισης των μοντέλων	50
2..2	Σημεία διαφοροποίησης των μοντέλων	51
2..3	Σύγκριση Κόστους	51
5	Συμπεράσματα	53
1.	Επίδραση Αρχιτεκτονικής στις Δυνατότητες των LLMs	53
2.	Σύγκριση των LLMs	54
3.	Πρακτική Χρησιμότητα των Ευρημάτων	56
4.	Προτάσεις για Μελλοντική Έρευνα	56
A'	Υλικό	60

Κεφάλαιο 1

Εισαγωγή

1. Περιγραφή Θέματος

Τα τελευταία χρόνια, η ραγδαία ανάπτυξη των Μεγάλων Γλωσσικών Μοντέλων (LLMs) έχει ανοίξει νέους δρόμους στην ανάπτυξη λογισμικού, προσφέροντας εργαλεία ικανά να παράγουν, να βελτιστοποιούν και να ενσωματώνουν κώδικα με υψηλή ακρίβεια και ταχύτητα. Μοντέλα όπως το GPT-4 και το Code Llama έχουν αναδειχθεί ως ισχυροί συνεργάτες των προγραμματιστών, ενισχύοντας την αποδοτικότητα και την παραγωγικότητα τους σε ποικίλα έργα ανάπτυξης λογισμικού.

Παράλληλα, η επιλογή αρχιτεκτονικής αποτελεί κρίσιμο ζήτημα στον σχεδιασμό εφαρμογών, με τις μονολιθικές και *microservices* αρχιτεκτονικές να κυριαρχούν ως οι δύο βασικές προσεγγίσεις. Η μονολιθική αρχιτεκτονική, με την ενιαία δομή της, προσφέρει απλότητα και άμεση διαχείριση, ενώ η *microservices* αρχιτεκτονική επιτρέπει την αποκέντρωση και την ευελιξία μέσω ανεξάρτητων υπηρεσιών.

Η παρούσα μελέτη εστιάζει στη συγκριτική αξιολόγηση των LLMs στην ανάπτυξη μιας REST API εφαρμογής σε Java, η οποία θα υλοποιηθεί τόσο σε μονολιθική όσο και σε *microservices* αρχιτεκτονική. Στόχος είναι να αναλυθούν οι δυνατότητες των LLMs στην παραγωγή κώδικα και να διερευνηθεί κατά πόσο η αρχιτεκτονική επηρεάζει την απόδοσή τους. Για τη σύγκριση θα ληφθούν υπόψη παράγοντες όπως η ποιότητα του παραγόμενου κώδικα, η αποδοτικότητα της εφαρμογής και η διαχείριση σφαλμάτων, ώστε να εξαχθούν χρήσιμα συμπεράσματα για την αποτελεσματικότητα κάθε μοντέλου σε διαφορετικά αρχιτεκτονικά περιβάλλοντα.

2. Συνεισφορά

Η παρούσα εργασία επιδιώκει να συμβάλει στην αξιολόγηση της χρησιμότητας σύγχρονων LLMs στην ανάπτυξη λογισμικού με έμφαση στην σύγκριση μεταξύ

μονολιθικών και microservices αρχιτεκτονικών. Πιο συγκεκριμένα η συνεισφορά έγκειται στα εξής σημεία:

1. Πρακτική αξιολόγηση των LLMs (GPT-4, Claude Sonnet 4, Gemini 2.5) σε πραγματικό περιβάλλον ανάπτυξης εφαρμογής τύπου REST API, τόσο με μονολιθική όσο και με microservices αρχιτεκτονική.
2. Συστηματική σύγκριση των μοντέλων με βάση ποσοτικά και ποιοτικά κριτήρια που θα καθοριστούν στην συνέχεια.
3. Ανάδειξη των επιπτώσεων της αρχιτεκτονικής επιλογής (μονολιθική vs. microservices) στις δυνατότητες και τους περιορισμούς κάθε μοντέλου, προσφέροντας χρήσιμα συμπεράσματα για τη μελλοντική αξιοποίηση των LLMs στην ανάπτυξη λογισμικού.

Η εργασία καλύπτει ένα ερευνητικό κενό καθώς δεν υπάρχει εκτενής βιβλιογραφία που να εξετάζει τα LLMs συναρτήσει της αρχιτεκτονικής.

3. Μεθοδολογία

Η μεθοδολογία που ακολουθείται περιλαμβάνει μια εκτενή βιβλιογραφική επισκόπηση σχετικά με το θεωρητικό υπόβαθρο. Η μελέτη, ωστόσο, επικεντρώνεται στο πρακτικό κομμάτι. Τα βήματα που ακολουθούνται συνοψίζονται ως εξής:

1. Επιλογή των LLMs προς αξιολόγηση: Επιλέγονται τρία σύγχρονα και ευρέως χρησιμοποιούμενα μοντέλα, συγκεκριμένα το GPT-4, το Claude Sonnet 4 και το Gemini 2.5.
2. Καθορισμός εφαρμογής: Ορίζεται μια απλή και συνάμα λειτουργική εφαρμογή σε Java, που εκτελεί βασικές λειτουργίες αναζήτησης δεδομένων.
3. Υλοποίηση της εφαρμογής: Η ίδια εφαρμογή υλοποιείται δύο φορές για κάθε μοντέλο, μία φορά με μονολιθική αρχιτεκτονική και μία με αρχιτεκτονική microservices.
4. Αξιολόγηση απόδοσης μοντέλων με βάση την αρχιτεκτονική: Για κάθε μοντέλο αξιολογείται η απόδοσή του, χρησιμοποιώντας τα παρακάτω κριτήρια:
 - Κατανόηση αρχιτεκτονικής: Εξετάζεται κατά πόσο το μοντέλο αντιλαμβάνεται σωστά τη δομή και τις απαιτήσεις της ζητούμενης αρχιτεκτονικής (Μονολιθικής ή Microservices). Η αξιολόγηση γίνεται ποιοτικά με βάση την πρώτη απόκριση του μοντέλου και κατατάσσεται σε μία από τις εξής τιμές:
 - Σαφής: το μοντέλο υλοποιεί σωστά την δομή εξαρχής

- Κατόπιν επεξήγησης: απαιτούνται διευκρινίσεις για να αποδώσει σωστά
- Ανακριβής: το μοντέλο παρουσιάζει λανθασμένη αντίληψη της αρχιτεκτονικής και χρειάζεται ανθρώπινη παρέμβαση
- Οργάνωση και διαχωρισμός αρμοδιοτήτων: Αξιολογείται ο τρόπος που το LLM οργανώνει την δομή του κώδικα, ειδικά ως προς τον διαχωρισμό επιπέδων (controller, service, repository) ή μεταξύ υπηρεσιών στην περίπτωση των microservices. Εξετάζεται ποιοτικά αν εφαρμόζεται καθαρή αρχιτεκτονική και οι ρόλοι κατανέμονται σωστά. Η αξιολόγηση γίνεται στις εξής κατηγορίες:
 - Σαφής: υπάρχει ξεκάθαρη διαστρωμάτωση του συστήματος
 - Επαρκής: υφίσταται διαστρωμάτωση αλλά παρουσιάζονται κάποιες αδυναμίες
 - Ασαφής: ο ρόλος κάθε μονάδας δεν είναι ορθά κατανεμεμένος και υπάρχει σύγχυση
- Διαχείριση σφαλμάτων: Ελέγχεται η χρήση μηχανισμών εξαιρέσεων (try/catch blocks, custom exceptions) και αν οι αποκρίσεις σε σφάλματα είναι κατανοητές και στοχευμένες (HTTP status code). Η διαχείριση σφαλμάτων θα αξιολογηθεί ποιοτικά με βάση την πορεία της υλοποίησης και θα διακριθεί σε δύο κατηγορίες:
 - Ακριβής: ορθή και επαρκής διαχείριση σφαλμάτων
 - Ελλιπής: απουσία ή ανεπαρκής χειρισμός σφαλμάτων και εξαιρέσεων
- Πληρότητα υλοποίησης: Εξετάζεται η πληρότητα του κώδικα, δηλαδή αν δημιουργήθηκε λειτουργική υλοποίηση από την αρχή ή αν απαιτήθηκαν επεμβάσεις. Η κάθε υλοποίηση θα χαρακτηριστεί είτε Πλήρης από τα αρχικά στάδια είτε Πλήρης Κατόπιν καθοδήγησης.
- Χρονική απόδοση και διαδικασία debugging: Αξιολογείται η ταχύτητα με την οποία το LLM εντοπίζει και διορθώνει σφάλματα κατά τη διαδικασία ανάπτυξης. Η αξιολόγηση βασίζεται σε μετρήσιμα στοιχεία, όπως:
 - Αριθμός προτροπών (prompts) που απαιτήθηκαν για την επιτυχή διόρθωση.
 - Έκταση αλλαγών στον παραγόμενο κώδικα.

Η απόδοση κατατάσσεται ως εξής:

- Γρήγορη: ≤ 3 prompts, ελάχιστες επεμβάσεις στον κώδικα.
 - Μέτρια: 4–6 prompts, μεσαίου μεγέθους αλλαγές στον κώδικα.
 - Αργή: > 6 prompts, εκτεταμένες διορθώσεις ή επανασχεδιασμός υλοποίησης.
- Στατική ανάλυση με SonarQube: Η ποιότητα του παραγόμενου κώδικα αξιολογείται με βάση μετρικές στατικής ανάλυσης όπως bugs, code smells και complexity, όπως εντοπίζονται από το εργαλείο SonarQube. Αποτελούν ποσοτικά στοιχεία και θα παρουσιαστούν αυτούσια. Ωστόσο λαμβάνεται υπόψη και η σημαντικότητα των λαθών.

- Σύγκριση κόστους: Λαμβάνεται υπόψη το κόστος της ανάπτυξης καθώς δεν παρέχεται ελεύθερη πρόσβαση σε όλα τα μοντέλα. Είναι σημαντικό να αναφερθεί ότι το κόστος έχει νόημα να συγκριθεί μόνο ως προς τα διαφορετικά μοντέλα και όχι ως προς τις αρχιτεκτονικές καθώς συνήθως τα μοντέλα προσφέρονται με συνδρομές που επαρκούν να αναπτύξουν και τις 2 υλοποιήσεις κάθε φορά.
- 5. Καταγραφή, ανάλυση και σύγκριση αποτελεσμάτων: Παρουσιάζονται συγκριτικοί πίνακες και αναλύονται τα ποιοτικά και ποσοτικά ευρήματα.
- 6. Διατύπωση συμπερασμάτων και προτάσεων: Συνοψίζονται τα ευρήματα και διατυπώνονται προτάσεις για τη χρήση LLMs σε παρόμοιες περιπτώσεις ανάπτυξης εφαρμογών.

Η παραπάνω μεθοδολογία επιτρέπει μια αντικειμενική σύγκριση των μοντέλων και των αρχιτεκτονικών, αναδεικνύοντας την βέλτιστη αξιοποίηση LLMs στην ανάπτυξη λογισμικού.

Κεφάλαιο 2

Βιβλιογραφική Επισκόπηση

1. Εισαγωγή στα Μεγάλα Γλωσσικά Μοντέλα

1.1 Τι είναι τα Μεγάλα Γλωσσικά Μοντέλα;

Τα μεγάλα γλωσσικά μοντέλα (Large Language Models - LLMs) αποτελούν ένα σημαντικό κομμάτι της τεχνητής νοημοσύνης καθώς είναι προηγμένα συστήματα ικανά να κατανοήσουν, να παράξουν, να διορθώσουν και να βελτιώσουν φυσική γλώσσα με ιδιαίτερη ταχύτητα και ακρίβεια [1]. Οι εξελιγμένες δυνατότητές τους οφείλονται στον τεράστιο όγκο δεδομένων που έχουν προεκπαιδευτεί, τα οποία προέρχονται από διάφορες πηγές όπως βιβλία, άρθρα και διαλόγους. Για την επίτευξη αυτού του στόχου, καταληκτική είναι η συμβολή της χρήσης τεχνικών μηχανικής μάθησης και ιδιαίτερα βαθιάς μάθησης. Αποτέλεσμα αυτών των προσπαθειών είναι το γεγονός ότι τα σύγχρονα μοντέλα έχουν ποικίλες δυνατότητες καθώς μπορούν να απαντούν φυσικά ερωτήσεις, να συντάσσουν πλήρη κείμενα, να δημιουργούν περιλήψεις, να κατανοούν και να αλλάζουν το ύφος του κειμένου [2].

Όσον αφορά τον τομέα του προγραμματισμού, πλέον, αποτελούν σημαντικά εργαλεία στα χέρια των προγραμματιστών. Οι εργασίες που μπορούν να διεκπεραιώσουν ποικίλουν καθώς είναι σε θέση να παράγουν, επεξηγούν και βελτιστοποιούν κώδικα [3]. Μπορούν, επίσης, να εντοπίσουν σφάλματα στον κώδικα και να προτείνουν διορθώσεις. Το γεγονός ότι οι δυνατότητες που προαναφέρθηκαν μπορούν να αξιοποιηθούν σε ευρεία γκάμα γλωσσών προγραμματισμού και τεχνολογιών λογισμικού αυξάνει σε μεγάλο βαθμό την χρησιμότητά τους στον κλάδο.. Ωστόσο, πέραν των πλεονεκτημάτων τους, η χρήση τους συνοδεύεται με κάποιους περιορισμούς. Αρχικά, οι λύσεις που προτείνουν κάποιες φορές περιέχουν σφάλματα ή μη βέλτιστες πρακτικές, ιδιαίτερα σε πιο σύνθετες περιπτώσεις με υψηλή πολυπλοκότητα [4]. Παράλληλα, εκλείπει πραγματική κατανόηση του προβλήματος και των απαιτήσεων, γεγονός που οδηγεί συχνά στην παραγωγή ανακριβούς ή παραπλανητικού περιεχομένου. Ιδιαίτερα στον προγραμματισμό, τα μοντέλα είναι επιρρεπή σε φαινόμενα hallucinations, δηλαδή παραγωγής συντακτικά σωστού αλλά λειτουρ-

γικά λανθασμένου κώδικα, με σοβαρές επιπτώσεις στην αξιοπιστία των εφαρμογών [5]. Για τον λόγο αυτό, η ανθρώπινη επίβλεψη και κριτική σκέψη παραμένουν απαραίτητες.[6].

1.2 Ιστορική Αναδρομή

Αρχικά, τα LLMs αποτέλεσαν μια προσπάθεια της τεχνητής νοημοσύνης να επεξεργαστούν την φυσική γλώσσα, βασιζόμενοι στην στατιστική. Στην πορεία, εμφανίστηκαν οι όροι βαθιά μάθηση και νευρωνικά δίκτυα που αποτέλεσαν την πραγματική πρόοδο τους. Συγκεκριμένα, τα νευρωνικά δίκτυα είναι υπολογιστικά μοντέλα που μιμούνται τον τρόπο που λειτουργεί ο ανθρώπινος εγκέφαλος. Η βαθιά μάθηση είναι κλάδος της μηχανικής μάθησης και χρησιμοποιεί τεχνητά νευρωνικά δίκτυα πολλών στρώσεων (layers) για να επεξεργαστεί την πληροφορία που λαμβάνει σε πολλά επίπεδα .

Η εξέλιξη των LLMs ξεκίνησε δυναμικά το 2013, όπου η Google ανέπτυξε το μοντέλο Word2Vec, το οποίο αναπαράστούσε την σημασία των λέξεων με βάση τα συμφραζόμενα, μέσω διανυσματικών αναπαραστάσεων [7]. Το 2018, η Google ανέπτυξε το μοντέλο του BERT (Bidirectional Encoder Representations from Transformers), που θεωρήθηκε επαναστατικό καθώς μπορούσε να διαβάσει λέξεις και από τις δύο κατευθύνσεις[8]. Αυτή η δυνατότητα αύξησε ραγδαία την κατανόηση της φυσικής γλώσσας και έθεσε νέα θεμέλια στον κλάδο. Την ίδια χρονιά, η OpenAI ανέπτυξε το GPT (Generative Pre-trained Transformer), το οποίο βασίστηκε στους μετασχηματιστές και ήταν πρωτοπόρο, εισάγοντας την έννοια της μαζικής προεκπαίδευσης σε μεγάλο όγκο δεδομένων [9]. Τα επόμενα έτη, η OpenAI προχώρησε σε νέες εκδόσεις του όπως GPT-2, GPT-3, GPT-3.5, με το GPT-4 να αποτελεί την πιο εξελιγμένη του έκδοση.

Τα παραπάνω μοντέλα αποτέλεσαν ορόσημα για τον κλάδο, ο οποίος έχει εξελιχθεί ραγδαία τα τελευταία χρόνια, με νέα μοντέλα και καινοτόμες λειτουργίες να αναπτύσσονται συνεχώς. Στην παρούσα φάση, πολλές εταιρείες παγκοσμίως επικεντρώνονται στην ανάπτυξη μοντέλων που ασχολούνται με προγραμματιστικές λειτουργίες όπως η κατανόηση και συγγραφή κώδικα που προαναφέρθηκαν. Κάποια από αυτά τα μοντέλα είναι το GPT-4, το Claude Sonnet και το Gemini, τα οποία θα μελετηθούν στην συνέχεια της εργασίας [10, 11].

1.3 GPT-4

Το GPT-4(Generative Pre-trained Transformer 4) αναπτύχθηκε από την OpenAI και κυκλοφόρησε το 2023. Αποτελεί ένα από τα πιο προηγμένα συστήματα LLM, καθώς είναι η εξέλιξη του GPT-3, με αρκετά βελτιωμένες λειτουργίες στην κατανόηση και παραγωγή φυσικής γλώσσας. Χρησιμοποιείται ευρέως από την πλατφόρμα Chat-GPT, αλλά και σε πολλές άλλες εφαρμογές και περιβάλλοντα ανάπτυξης λογισμικού, όπου μπορεί να παράγει, να εξηγεί και να διορθώνει κώδικα σε διάφορες γλώσσες,

όπως Python, JavaScript, C++, HTML και άλλες. Επιπλέον είναι σε θέση να αναλύει απαιτήσεις, να προτείνει λύσεις και βελτιστοποιήσεις και να εντοπίζει λάθη [12].

1.4 Gemini 2.5

Το Gemini 2.5 αναπτύχθηκε από την Google DeepMind και κυκλοφόρησε το 2024. Αποτελεί ένα προηγμένο LLM και μια βελτιωμένη έκδοση της σειράς Gemini με ενισχυμένες δυνατότητες κατανόησης και ανάλυσης φυσικής γλώσσας, κώδικα και εικόνων. Το Gemini 2.5 χρησιμοποιείται ευρέως στην πλατφόρμα Google AI Studio. Παράλληλα, υποστηρίζεται η ενσωμάτωση του σε διάφορα εργαλεία όπως το Android Studio, παρέχοντας υποστήριξη στην συγγραφή, κατανόηση και βελτιστοποίηση κώδικα. Το μοντέλο, επίσης, έχει την δυνατότητα να επεξεργάζεται μεγάλα σύνολα δεδομένων, να αναλύει τεχνικές απαιτήσεις και να υποστηρίζει ενεργά την λήψη αποφάσεων βάση του περιβάλλοντος του [13].

1.5 Claude Sonnet 4

Το Claude Sonnet 4.0 αναπτύχθηκε από την Anthropic και κυκλοφόρησε το 2024. Αποτελεί ένα από τα πιο ισορροπημένα και προηγμένα μοντέλα της σειράς Claude 3. Παρέχει υψηλές αποδόσεις στην κατανόηση και παραγωγή τόσο φυσικής γλώσσας όσο και κώδικα αποτελώντας έναν δυνατό σύμμαχο στην ανάπτυξη λογισμικού. Παρέχεται κυρίως από την πλατφόρμα την Anthropic, αν και μέσω του API υποστηρίζεται απο μεγάλη γκάμα εφαρμογών όπως το Amazon Bedrock. Η σχεδιαστική του προσέγγιση επιτρέπει την ασφαλή επεξεργασία μεγάλου όγκου πληροφοριών, την κατανόηση περίπλοκων τεχνικών οδηγιών και την παραγωγή καθαρού, καλά δομημένου κώδικα [14].

2. Εισαγωγή στις Αρχιτεκτονικές Λογισμικού

2.1 Εισαγωγή

Η αρχιτεκτονική του λογισμικού αποτελεί καθοριστικό παράγοντα στην δομή ενός συστήματος καθώς και κρίσιμο παράγοντα της επιτυχίας του, ιδίως σε ό,τι αφορά την επεκτασιμότητα, τη συντηρησιμότητα, την ευκολία ανάπτυξης και τη δυνατότητα αυτοματοποίησης. Καθώς η ανάπτυξη λογισμικού εξελίσσεται, νέες τάσεις, ανάγκες και τεχνολογικά εργαλεία οδηγούν στην εμφάνιση και στην υιοθέτηση διαφορετικών αρχιτεκτονικών προσεγγίσεων. Στο πλαίσιο αυτής της εργασίας, εστιάζουμε στη συγκριτική μελέτη δύο κυρίαρχων προσεγγίσεων: της μονολιθικής και της microservices αρχιτεκτονικής, ειδικά στο περιβάλλον ανάπτυξης REST API σε Java.

2..2 Μονολιθική Αρχιτεκτονική

Η μονολιθική αρχιτεκτονική αποτελεί την πιο ιστορική και παραδοσιακή προσέγγιση της ανάπτυξης λογισμικού. Η βασική ιδέα βρίσκεται στην συγκέντρωση όλων των συστατικών μιας εφαρμογής (Frontend, Backend, πρόσβαση στην βάση δεδομένων, κα) σε ένα ενιαίο έργο.

Ένα από τα βασικά της πλεονεκτήματα είναι η απλότητα στην ανάπτυξη, τη δοκιμή και την υλοποίηση, καθώς όλα τα υποσυστήματα βρίσκονται στον ίδιο χώρο εκτέλεσης, επιτρέποντας άμεση επικοινωνία μεταξύ τους. Αυτό καθιστά ευκολότερη την εύρεση σφαλμάτων και την ενσωμάτωση αλλαγών. Παράλληλα, η μονολιθική προσέγγιση συχνά έχει καλύτερες επιδόσεις, επειδή όλα τα μέρη της εφαρμογής επικοινωνούν μεταξύ τους απευθείας, χωρίς να χρειάζεται να επικοινωνούν μέσω δικτύου όπως συμβαίνει σε πιο καταναμημένα συστήματα.

Ωστόσο, η μονολιθική δομή ενέχει σημαντικά μειονεκτήματα, ειδικά σε εφαρμογές μεγάλης κλίμακας ή ταχείας εξέλιξης. Καθώς το σύστημα μεγαλώνει, καθίσταται δύσκολη η συντήρηση, ενώ ακόμα και μικρές αλλαγές ενδέχεται να επηρεάσουν απρόβλεπτα άλλα μέρη της εφαρμογής. Επιπρόσθετα, ο περιορισμένος βαθμός επεκτασιμότητας, η δυσκολία στην υιοθέτηση νέων τεχνολογιών ανά υποσύστημα, καθώς και η αδυναμία ανεξάρτητης ανάπτυξης λειτουργιών αποτελούν σημαντικούς περιορισμούς της ευελιξίας της αρχιτεκτονικής [15, 16].

2..3 Microservices Αρχιτεκτονική

Η αρχιτεκτονική microservices βασίζεται στη λογική της διάσπασης μιας εφαρμογής σε μικρότερες, αυτόνομες υπηρεσίες, καθεμία από τις οποίες εκτελεί μια συγκεκριμένη λειτουργία και επικοινωνεί με τις υπόλοιπες μέσω κατάλληλων διεπαφών (όπως REST APIs).

Αυτή η προσέγγιση προσφέρει σημαντικά πλεονεκτήματα, όπως αυξημένη ευελιξία και ευκολία στην ανάπτυξη και συντήρηση του λογισμικού. Κάθε υπηρεσία μπορεί να αναπτυχθεί, να δοκιμαστεί και να βελτιστοποιηθεί ανεξάρτητα, κάτι που επιταχύνει την κυκλοφορία νέων χαρακτηριστικών. Επίσης, διευκολύνει την κλιμάκωση της εφαρμογής, καθώς μπορούν να αυξηθούν οι πόροι μόνο για τις υπηρεσίες που το απαιτούν [17].

Ωστόσο, η προσέγγιση αυτή εισάγει και νέες προκλήσεις. Η πολυπλοκότητα της υποδομής αυξάνεται, απαιτώντας λύσεις για τη διαχείριση της επικοινωνίας, του συντονισμού και της ασφάλειας μεταξύ των υπηρεσιών. Επιπλέον, η ύπαρξη πολλών διαφορετικών υπηρεσιών συνεπάγεται συχνά μεγαλύτερη ανάγκη για παρακολούθηση, logging και debugging, γεγονός που μπορεί να αυξήσει το κόστος και τον χρόνο ανάπτυξης. Παρά τις δυσκολίες, τα microservices παραμένουν ιδιαίτερα δημοφιλή σε μεγάλες εφαρμογές και οργανισμούς που επιζητούν ευελιξία και δυνατότητα συνεχούς ανάπτυξης [18, 19].

2..4 Συγκριτική ανάλυση στο Πλαίσιο REST API σε Java

Τα REST APIs (Representational State Transfer Application Programming Interfaces) αποτελούν έναν διαδεδομένο και αρκετά δημοφιλή τρόπο επικοινωνίας στα συστήματα λογισμικού. Η επικοινωνία γίνεται μέσω του πρωτοκόλλου HTTP (Hypertext Transfer Protocol), επιτρέποντας την αποστολή και λήψη δεδομένων μέσω καθορισμένων διαδρομών (endpoints). Σε περιβάλλοντα της Java, για την ανάπτυξη Rest Apis εφαρμογών, συχνά, επιλέγεται η χρήση του framework Spring Boot, καθώς περιέχει δομημένες μεθόδους για καλύτερη διαχείριση των αιτημάτων [20]. Η αρχιτεκτονική που επιλέγεται (μονολιθική ή microservices) επηρεάζει σημαντικά αρχικά την κατασκευή και μετέπειτα τις δυνατότητες ελέγχου, συντήρησης και επέκτασης [21].

Μονολιθική Προσέγγιση: Η δημιουργία όλων των endpoints πραγματοποιείται σε ένα ενιαίο αρχείο που περιλαμβάνει όλα τα συστατικά όπως τις οντότητες (entities), υπηρεσίες (services) και τους ελεγκτές (controllers).

Microservices Προσέγγιση: Κάθε επιμέρους λειτουργικότητα αποτελεί μια ξεχωριστή υπηρεσία (service) με δικό της REST API. Η επικοινωνία των υπηρεσιών αυτών συντελείται μέσω HTTP και απαιτούνται κάποιοι μηχανισμοί συντονισμού.

Παρακάτω ακολουθεί ένας συνοπτικός πίνακας με τα πλεονεκτήματα και τα μειονεκτήματα της κάθε προσέγγισης [22, 23].

Πίνακας 2.1: Σύγκριση Μονολιθικής και Microservices Αρχιτεκτονικής σε REST API με Java

Κριτήριο	Μονολιθική Αρχιτεκτονική	Microservices Αρχιτεκτονική
Απλότητα Ανάπτυξης	Υψηλή	Χαμηλότερη
Επεκτασιμότητα	Περιορισμένη	Υψηλή
Συντηρησιμότητα	Δύσκολη σε μεγάλα έργα	Ευκολότερη
Ελεγκσιμότητα (Testability)	Περιορισμένη	Υψηλή
Απόδοση	Υψηλή	Πιο αργή
Τεχνολογική Ευελιξία	Περιορισμένη	Υψηλή
Δυσκολία Υλοποίησης	Χαμηλή	Υψηλή
Χρήση σε Μικρές Εφαρμογές	Κατάλληλη	Υπερβολική πολυπλοκότητα

2..5 Συσχέτιση με LLMs

Οι δυνατότητες των LLMs διαφοροποιούνται ανάλογα με την αρχιτεκτονική που καλούνται να υλοποιήσουν. Από την μία, η μονολιθική αρχιτεκτονική διευκολύνει τις οδηγίες που θα δώσουμε στο μοντέλο, λόγω του συνεκτικού της χαρακτήρα. Ένα LLM μπορεί να κατανοήσει καλύτερα τις απαιτήσεις και να παράξει καλύτερα αποτελέσματα όταν έχει εικόνα όλου του έργου.

Αντιθέτως, όταν ένα σύστημα αναπτύσσεται με *microservices*, κάθε υπηρεσία είναι αυτόνομη και ανεξάρτητη. Τα LLMs αποδίδουν καλά στην ανάπτυξη κάθε επιμέρους υπηρεσίας καθώς αποτελούν μικρότερα και πιο στοχευμένα τμήματα του συστήματος. Παράλληλα, τέτοια μοντέλα παρέχουν σημαντική βοήθεια στην τεκμηρίωση, τους ελέγχους και στις βελτιστοποιήσεις του κώδικα, γεγονός αρκετά σημαντικό στην περίπτωση των *microservices* λόγω αυξημένης πολυπλοκότητας.

Σε γενικό πλαίσιο, τα LLMs μπορούν να προσαρμοστούν και στις δύο προσεγγίσεις, με την αποτελεσματικότητά τους να διαφέρει βεβαίως [24]. Σε πιο μικρές μονολιθικές εφαρμογές μπορούν να συνδράμουν στην ολιστική παραγωγή του κώδικα ενώ στις *microservices* βοηθούν στην επιμέρους παραγωγή των αυτόνομων μονάδων [4, 25]. Παρ' όλα αυτά, η χρήση των LLMs συνοδεύεται από περιορισμούς, όπως η πιθανότητα δημιουργίας *hallucinations* ή εσφαλμένων εξόδων κατά τη διαδικασία παραγωγής κώδικα, γεγονός που έχει επισημανθεί σε σύγχρονες μελέτες [26].

Κεφάλαιο 3

Πρακτικό Μέρος

1. Περιγραφή Εφαρμογής

Στο πλαίσιο της παρούσας μελέτης αναπτύχθηκε μία REST API εφαρμογή σε Java 17 χρησιμοποιώντας το Spring Boot framework, με σκοπό την ανάκτηση και παρουσίαση πληροφοριών σχετικά με ταινίες, αξιοποιώντας την εξωτερική IMDB REST API μέσω του RapidAPI.

Η εφαρμογή προσφέρει ένα REST endpoint το οποίο δέχεται ένα τίτλο ταινίας ως query parameter και επιστρέφει τα ακόλουθα πεδία σε μορφή JSON:

title: Τίτλος της ταινίας

year: Έτος κυκλοφορίας

runningTimeInMinutes: Διάρκεια προβολής σε λεπτά

leadActor: Πρωταγωνιστής

Για την αναζήτηση ταινιών, γίνεται χρήση του endpoint `/title/find` της εξωτερικής IMDB API. Η εφαρμογή επικοινωνεί με την API μέσω HTTP κλήσεων, επεξεργάζεται τα αποτελέσματα και επιστρέφει τις ζητούμενες πληροφορίες στον τελικό χρήστη με καθαρή και κατανοητή δομή.

Σε περίπτωση αποτυχίας της κλήσης προς την εξωτερική υπηρεσία ή σφάλματος στην επεξεργασία των δεδομένων, η εφαρμογή επιστρέφει κατάλληλα μηνύματα λάθους με κατανοητή περιγραφή, προκειμένου να βοηθήσει τον χρήστη να κατανοήσει τη φύση του προβλήματος.

Σε όλα τα μοντέλα δόθηκε το παραπάνω prompt δύο φορές, μια φορά με την οδηγία να αναπτύξουν την εφαρμογή με μονολιθική αρχιτεκτονική και την επόμενη με την αρχιτεκτονική `microservices`.

Αν η υλοποίηση της εφαρμογής γινόταν αποκλειστικά από εμένα, χωρίς τη βοήθεια Μεγάλων Γλωσσικών Μοντέλων, στη μονολιθική εκδοχή θα επέλεγα τη χρήση του προτύπου MVC, με σαφή διαχωρισμό μεταξύ controller, service και model, ακολουθώντας αρχές καθαρής αρχιτεκτονικής. Θα φρόντιζα επίσης για τη δημιουργία κατάλληλων μηχανισμών διαχείρισης εξαιρέσεων με custom exceptions και global exception handler για ομοιόμορφες αποκρίσεις. Στην περίπτωση της αρχιτεκτονικής microservices, θα προχωρούσα στον διαχωρισμό της εφαρμογής σε δύο ανεξάρτητα Spring Boot services: ένα για την επικοινωνία με το IMDB API και ένα για την παροχή των REST endpoints στον τελικό χρήστη. Τα services θα επικοινωνούσαν μεταξύ τους μέσω HTTP, με ιδιαίτερη προσοχή στη διαχείριση σφαλμάτων, στην τυποποίηση των απαντήσεων και στη σωστή ρύθμιση των διαδρομών και των πόρων του κάθε μικροϋπηρεσιακού συστήματος. Η προσέγγισή μου βασίζεται σε καθιερωμένες πρακτικές του κλάδου, και γι' αυτό η αξιολόγηση που ακολουθεί γίνεται συγκριτικά με αυτά τα πρότυπα.

2. Ανάπτυξη Εφαρμογής

2.1 GPT-4

2.1.1 GPT-4: Μονολιθική υλοποίηση

Πίνακας 3.1: Διάλογος μεταξύ Χρήστη και GPT-4 για την παραγωγή εφαρμογής

Ομιλητής	Περιεχόμενο
Χρήστης	Ζητήθηκε από το μοντέλο να αναπτύξει μια εφαρμογή, με την χρήση του prompt που καθορίστηκε προηγουμένως με μονολιθική αρχιτεκτονική
GPT-4	Δημιουργήθηκε μονολιθική εφαρμογή, με πλήρη κώδικα (pom.xml, controller, service, client, DTO, CLI).
Χρήστης	Ενημέρωση ότι υπάρχει σφάλμα καθώς από πολλές εξαρτήσεις της maven λείπουν οι κατάλληλες εκδόσεις
GPT-4	Διόρθωση σφάλματος εκδόσεων
Χρήστης	Ενημέρωση ότι το σύστημα εκτελείται κανονικά. Ωστόσο μετά την πρώτη επανάληψη, το σύστημα τερματίζεται ανεξαρτήτως από το εάν ο χρήστης θέλει να αναζητήσει νέα ταινία ή όχι, το οποίο αποτελεί λογικό λάθος
GPT-4	Εντοπισμός λάθους και διόρθωσή του

Παρακάτω, η Εικόνα 3.1 δείχνει την τελική δομή του project που ανέπτυξε το GPT-4 και καλύπτει τις απαιτήσεις της εκφώνησης.



Εικόνα 3.1: Δομή της μονολιθικής αρχιτεκτονικής με την χρήση GPT-4

- **MovieSearchApplication.java:** Η κύρια κλάση της εφαρμογής. Εκκινεί το Spring Boot και χειρίζεται είσοδο από τη γραμμή εντολών για αναζήτηση ταινιών.
- **MovieController.java:** Προσφέρει REST endpoint που δέχεται τίτλο ταινίας και επιστρέφει πληροφορίες σε JSON μορφή.
- **MovieService.java:** Περιέχει τη βασική λογική της εφαρμογής. Καλεί τον client για τα δεδομένα και τα επιστρέφει στον controller.
- **ImdbClient.java:** Εκτελεί HTTP κλήσεις προς το IMDB API μέσω RapidAPI και επεξεργάζεται τις αποκρίσεις JSON.

- **MovieResponse.java:** Αντικείμενο μεταφοράς δεδομένων με τα πεδία: τίτλος, έτος, διάρκεια και πρωταγωνιστής.
- **application.properties:** Αρχείο ρυθμίσεων για την εφαρμογή (π.χ. θύρα server, κλειδιά API).

2.1.2 GPT-4: Microservices υλοποίηση

Πίνακας 3.2: Διάλογος μεταξύ Χρήστη και GPT-4 για την παραγωγή εφαρμογής

Ομιλητής	Περιεχόμενο
Χρήστης	Ζητήθηκε από το μοντέλο να αναπτύξει μια εφαρμογή, με την χρήση του prompt που καθορίστηκε προηγουμένως με microservices αρχιτεκτονική
GPT-4	Δημιουργήθηκαν 2 services, ένα για την αναζήτηση ταινιών και ένα για την παροχή του endpoint, καθώς και τα υπόλοιπα σχετικά στοιχεία όπως pom.xml και application.properties.
Χρήστης	Ενημέρωση ότι η δομή που έχει δώσει δεν είναι ξακάθαρη καθώς ο κώδικας που είχε δώσει δεν ήταν ολοκληρωμένος, έλειπαν οι ονομασίες των packages
GPT-4	Διασαφήνιση της δομής
Χρήστης	Ενημέρωση ότι οι κλάσεις δεν είναι ολοκληρωμένες καθώς λείπουν τα απαραίτητα imports και dependencies
GPT-4	Εντοπισμός λάθους και διόρθωσή του
Χρήστης	Ενημέρωση ότι ο κώδικας στην πραγματικότητα δεν αποτελεί microservices εφαρμογή καθώς ενώ περιλαμβάνει 2 διαφορετικά modules, το κάθε module δεν τρέχει ξεχωριστά. Αντιθέτως, έχει χρησιμοποιήσει την ίδια λογική με την μονολιθική εφαρμογή, έχοντας χωρίσει τις κλάσεις στα δύο modules
GPT-4	Κατανόησε το σφάλμα, ξαναέστησε την εφαρμογή με σωστό τρόπο, αναπτύσσοντας 2 microservices που τρέχουν αυτόματα
Χρήστης	Ενημέρωση ότι ένα από τα δύο services δεν περιλαμβάνει την κλάση main ώστε να τρέξει
GPT-4	Εντοπισμός λάθους και διόρθωσή του
Χρήστης	Ενημέρωση ότι κάνει χρήση της βιβλιοθήκης UriUtils που δεν είναι συμβατό με την έκδοση του SpringBoot που χρησιμοποιείται
GPT-4	Εντοπισμός προβλήματος και επίλυση με εναλλακτική προσέγγιση

Παρακάτω, η Εικόνα 3.1 δείχνει την τελική δομή του project που ανέπτυξε το GPT-4 και καλύπτει τις απαιτήσεις της εκφώνησης.



Εικόνα 3.2: Δομή της microservices αρχιτεκτονικής με την χρήση GPT-4

► **movie-search-service**

Αυτή η μικροϋπηρεσία είναι υπεύθυνη για την αναζήτηση ταινιών μέσω της εξωτερικής υπηρεσίας IMDB, αξιοποιώντας την RapidAPI. Δέχεται έναν τίτλο ταινίας ως είσοδο μέσω REST endpoint, πραγματοποιεί HTTP αίτημα προς το IMDB API, επεξεργάζεται την απόκριση και επιστρέφει βασικές πληροφορίες για την ταινία (τίτλος, έτος, διάρκεια, πρωταγωνιστής) σε μορφή JSON.

► **MovieSearchServiceApplication.java**

Κύρια κλάση εκκίνησης του Spring Boot service. Εκκινεί τον server στη θύρα 8081.

► **SearchController.java**

REST controller που δέχεται τίτλο ταινίας, καλεί το IMDB API μέσω RapidAPI

και επιστρέφει βασικές πληροφορίες για την ταινία.

► **MovieDTO.java**

Κλάση μεταφοράς δεδομένων (Data Transfer Object) που περιλαμβάνει τίτλο, έτος, διάρκεια και πρωταγωνιστή.

► **application.properties**

Αρχείο ρυθμίσεων. Περιέχει τη θύρα (8081) και το API key για την πρόσβαση στη RapidAPI.

movie-api-service

► **MovieApiServiceApplication.java**

Εκτελέσιμη κλάση που περιλαμβάνει CLI. Ζητάει τίτλο ταινίας από τον χρήστη και καλεί την υπηρεσία αναζήτησης για να εμφανίσει τα αποτελέσματα.

► **application.properties**

Ρυθμίζει τη θύρα (8080) και το URL της υπηρεσίας αναζήτησης για την επικοινωνία μεταξύ των microservices.

2..2 Gemini

Gemini: Μονολιθική υλοποίηση

Πίνακας 3.3: Διάλογος μεταξύ Χρήστη και Gemini για την Επίλυση Προβλημάτων Εφαρμογής

Ομιλητής	Περιεχόμενο
Χρήστης	Ζητήθηκε από το μοντέλο να αναπτύξει μια εφαρμογή, με την χρήση του prompt που καθορίστηκε προηγουμένως με μονολιθική αρχιτεκτονική
Gemini	Δημιουργήθηκε μονολιθική εφαρμογή, με πλήρη κώδικα (pom.xml, controller, service, exception, DTO, application.properties).
Χρήστης	Ενημέρωση ότι λείπουν κάποια απαραίτητα dependencies
Gemini	Προσθήκη όλων των εξαρτήσεων
Χρήστης	Ενημέρωση για αναντιστοιχία τύπου στον κώδικα (Mono vs Mono<Optional>). Χρειάστηκαν 4 επαναλήψεις ώστε να διορθωθεί το λάθος
Gemini	Στο 4ο prompt δόθηκε σωστή λύση
Χρήστης	Ενημέρωση για μια εξαίρεση που δημιουργείται
Gemini	Εντοπισμός λάθους και διόρθωσή του
Χρήστης	Ενημέρωση ότι ενώ η εφαρμογή τρέχει δεν εμφανίζει τα πεδία που ζητούνται και κοινοποίηση json που λαμβάνει
Gemini	Εντοπισμός λάθους και διόρθωσή του
Χρήστης	Ενημέρωση ότι υπήρξε πρόοδος και εμφανίζει όλα τα πεδία εκτός από την διάρκεια της ταινίας
Gemini	Εντοπισμός λάθους και διόρθωσή του



Εικόνα 3.3: Δομή της μονολιθικής αρχιτεκτονικής με την χρήση Gemini

- **GeminiApplication.java:** Η κύρια κλάση εκκίνησης (entry point) της εφαρμογής Spring Boot. Περιέχει τη μέθοδο 'main' που εκκινεί τον server και φορτώνει όλα τα απαραίτητα beans.
- **MovieController.java:** Ένας REST Controller που χειρίζεται τα εισερχόμενα HTTP αιτήματα από τους πελάτες. Εκθέτει τα API endpoints για την αναζήτηση ταινιών.
- **ImdbApiException.java:** Μια custom εξαίρεση που χρησιμοποιείται για να σηματοδοτήσει σφάλματα που προκύπτουν κατά την επικοινωνία με το IMDb API.
- **MovieNotFoundException.java:** Μια custom εξαίρεση που δηλώνει ότι δεν βρέθηκε κάποια ταινία με βάση τα κριτήρια αναζήτησης.

- **ImdbApiResponse.java:** Το μοντέλο δεδομένων (Data Transfer Object) που χρησιμοποιείται για να χαρτογραφήσει την πλήρη δομή του JSON response που λαμβάνεται από το IMDb API.
- **MovieInfo.java:** Ένα απλοποιημένο μοντέλο δεδομένων. Αυτή η κλάση περιέχει μόνο τις βασικές πληροφορίες για μια ταινία (π.χ. τίτλος, έτος, διάρκεια, πρωταγωνιστής) που θέλουμε να χρησιμοποιήσουμε ή να εμφανίσουμε στην εφαρμογή μας.
- **ImdbApiService.java:** Η κλάση υπηρεσίας που περιέχει την επιχειρηματική λογική για την επικοινωνία με το εξωτερικό IMDb API, την ανάκτηση των δεδομένων και τον αρχικό μετασχηματισμό τους.
- **application.properties:** Το αρχείο ρυθμίσεων της εφαρμογής. Περιέχει παραμέτρους όπως το API key, το host και το URL του IMDb API, καθώς και άλλες ρυθμίσεις της Spring Boot εφαρμογής.

Gemini: Microservices υλοποίηση

Πίνακας 3.4: Διάλογος μεταξύ Χρήστη και Gemini για την Επίλυση Προβλημάτων Εφαρμογής

Ομιλητής	Περιεχόμενο
Χρήστης	Ζητήθηκε από το μοντέλο να αναπτύξει μια εφαρμογή, με την χρήση του prompt που καθορίστηκε προηγουμένως με microservices αρχιτεκτονική
Gemini	Δημιουργήθηκαν 2 services, ένα για την αναζήτηση ταινιών και ένα για την παροχή του endpoint, καθώς και ένα επιπλέον module για αλληλεπίδραση με τον χρήστη μέσω της γραμμής εντολών καθώς και τα υπόλοιπα σχετικά στοιχεία όπως pom.xml και application.properties.
Χρήστης	Ενημέρωση ότι λείπουν κάποια απαραίτητα dependencies
Gemini	Προσθήκη όλων των εξαρτήσεων
Χρήστης	Ο κώδικας έτρεξε. Ωστόσο, δεν επεξεργάζεται σωστά τα δεδομένα που λαμβάνει με αποτέλεσμα να εμφανίζει ότι δεν βρέθηκε κάποια ταινία.
Gemini	Ζητήθηκε από τον χρήστη να μοιραστεί τα logs του προγράμματος ώστε να εντοπίσει το λάθος. Από τα logs είδε την δομή του json που επιστρέφει το API και κατάλαβε ότι ο τρόπος που παίρνει τα δεδομένα είναι λάθος. Πρότεινε μια λύση.
Χρήστης	Ενημέρωση ότι η πρόταση του δεν επίλυσε το πρόβλημα
Gemini	Εναλλακτική πρόταση.
Χρήστης	Ενημέρωση ότι το σφάλμα διορθώθηκε εν μέρει. Ωστόσο το πρόγραμμα συνεχίζει να μην εμφανίζει σωστά την διάρκεια της ταινίας
Gemini	Επίλυση προβλήματος

```

my-movie-app/
├── pom.xml
├── imdb-api-client-service/
│   ├── src/main/java/com/example/imdbapiclient/
│   │   ├── ImdbApiClientServiceApplication.java
│   │   ├── controller/
│   │   │   └── ImdbApiController.java
│   │   ├── dto/
│   │   │   ├── ImdbMovieResponse.java
│   │   │   └── Movie.java
│   │   ├── exception/
│   │   │   ├── GlobalExceptionHandler.java
│   │   │   ├── ImdbApiException.java
│   │   │   └── MovieNotFoundException.java
│   │   └── service/
│   │       └── ImdbApiClientService.java
│   ├── src/main/resources/
│   │   └── application.properties
│   └── pom.xml
├── movie-cli-client/
│   ├── src/main/java/com/example/moviecli/
│   │   └── MovieCliApplication.java
│   └── pom.xml
└── movie-search-service/
    ├── src/main/java/com/example/moviesearch/
    │   ├── MovieSearchServiceApplication.java
    │   ├── controller/
    │   │   └── MovieController.java
    │   ├── dto/
    │   │   ├── ErrorResponse.java
    │   │   └── Movie.java
    │   ├── exception/
    │   │   ├── GlobalExceptionHandler.java
    │   │   └── ImdbApiClientException.java
    │   └── service/
    │       └── MovieService.java
    ├── src/main/resources/
    │   └── application.properties
    └── pom.xml

```

Εικόνα 3.4: Δομή της microservices αρχιτεκτονικής με την χρήση Gemini

imdb-api-client-service

Αυτό το microservice είναι υπεύθυνο για όλη την επικοινωνία με το εξωτερικό RapidAPI IMDB.

- ▶ **ImdbApiClientServiceApplication.java:** Η κύρια κλάση που εκκινεί αυτό το Spring Boot service.
- ▶ **ImdbApiController.java:** Παρέχει REST endpoints που επιτρέπουν σε άλλα services να ζητούν πληροφορίες ταινιών από το IMDB API μέσω HTTP.
- ▶ **ImdbApiClientService.java:** Ο πυρήνας της λογικής για την αλληλεπίδραση με το RapidAPI. Πραγματοποιεί τις πραγματικές κλήσεις στο IMDB API (μία για αναζήτηση, μία για λεπτομέρειες) και παρσάρει τις JSON απαντήσεις.
- ▶ **ImdbMovieResponse.java:** Ένα αντικείμενο μεταφοράς δεδομένων (DTO) που αντιστοιχεί στη δομή της απάντησης από το αρχικό endpoint αναζήτησης του IMDB API.
- ▶ **Movie.java:** Ένα DTO που αναπαριστά μια ταινία με τα βασικά της στοιχεία (τίτλος, έτος, διάρκεια, πρωταγωνιστής). Χρησιμοποιείται για τη μεταφορά δεδομένων εντός και μεταξύ των services.
- ▶ **GlobalExceptionHandler.java:** Αναλαμβάνει την κεντρική διαχείριση των εξαιρέσεων που μπορεί να συμβούν σε αυτό το service, επιστρέφοντας τυποποιημένες HTTP απαντήσεις σφάλματος.
- ▶ **ImdbApiException.java:** Μια προσαρμοσμένη εξαίρεση που σηματοδοτεί γενικά προβλήματα κατά την επικοινωνία με το RapidAPI IMDB.
- ▶ **MovieNotFoundException.java:** Μια προσαρμοσμένη εξαίρεση που πετιέται όταν το IMDB API δεν επιστρέψει αποτελέσματα για τον αναζητούμενο τίτλο ταινίας.

—

movie-cli-client

Αυτό το module είναι η εφαρμογή γραμμής εντολών που επιτρέπει στον χρήστη να αλληλεπιδρά με το σύστημα.

- ▶ **MovieCliApplication.java:** Η κύρια κλάση της CLI εφαρμογής. Διαχειρίζεται την είσοδο του χρήστη, καλεί το 'movie-search-service' για την αναζήτηση ταινιών και εμφανίζει τα αποτελέσματα στην κονσόλα.

—

movie-search-service

Αυτό το microservice λειτουργεί ως ο κεντρικός κόμβος για την αναζήτηση ταινιών, συντονίζοντας την επικοινωνία μεταξύ του CLI client και του IMDB API client.

- ▶ **MovieSearchServiceApplication.java:** Η κύρια κλάση που εκκινεί αυτό το Spring Boot service.
- ▶ **MovieController.java:** Παρέχει το βασικό REST endpoint ('/movies/search') στο οποίο καλεί ο CLI client για να ζητήσει αναζήτηση ταινίας.
- ▶ **MovieService.java:** Περιέχει την επιχειρηματική λογική για την αναζήτηση. Καλεί το 'imdb-api-client-service' (μέσω του 'ImdbApiController' endpoint του) για να λάβει τα δεδομένα της ταινίας.
- ▶ **ErrorResponse.java:** Ένα DTO που χρησιμοποιείται για την τυποποιημένη επιστροφή μηνυμάτων σφάλματος στους καταναλωτές του API.
- ▶ **Movie.java:** Το ίδιο DTO με αυτό στο 'imdb-api-client-service' για την αναπαράσταση της ταινίας.
- ▶ **GlobalExceptionHandler.java:** Παρέχει κεντρική διαχείριση εξαιρέσεων για αυτό το service, εξασφαλίζοντας συνεπείς αποκρίσεις σφάλματος.
- ▶ **ImdbApiClientException.java:** Μια προσαρμοσμένη εξαίρεση που υποδηλώνει προβλήματα κατά την επικοινωνία με το 'imdb-api-client-service'.

2.3 Claude-Sonnet

2.3.1 Claude-Sonnet: Μονολιθική υλοποίηση

Πίνακας 3.5: Διάλογος μεταξύ Χρήστη και Claude-Sonnet για την παραγωγή εφαρμογής

Ομιλητής	Περιεχόμενο
Χρήστης	Ζητήθηκε από το μοντέλο να αναπτύξει μια εφαρμογή, με την χρήση του prompt που καθορίστηκε προηγουμένως με μονολιθική αρχιτεκτονική
Claude-Sonnet	Δημιουργήθηκε μονολιθική εφαρμογή, με πλήρη κώδικα (pom.xml, controller, service, model, exception, CLI).
Χρήστης	Ενημέρωση ότι υπάρχει σφάλμα στην δήλωση μεταβλητών του application.yml
Claude-Sonnet	Διόρθωση σφάλματος δήλωσης μετά από 2 προσπάθειες
Χρήστης	Ενημέρωση ότι το πρόγραμμα δεν εκτελείται διότι λείπουν κάποιες εξαρτήσεις
Claude-Sonnet	Εντοπισμός λάθους και διόρθωσή του

Παρακάτω, η Εικόνα 3.5 δείχνει την τελική δομή του project που ανέπτυξε το Claude-Sonnet και καλύπτει τις απαιτήσεις της εκφώνησης.



Εικόνα 3.5: Δομή της μονολιθικής αρχιτεκτονικής με την χρήση Claude-Sonnet

- **MovieSearchApiApplication.java:** Η βασική κλάση της εφαρμογής Spring Boot. Εκκινεί την εφαρμογή και αποτελεί το σημείο εκκίνησης του συστήματος.

- ▶ **MovieSearchCLI.java:** Παρέχει διεπαφή γραμμής εντολών (CLI) για την αναζήτηση ταινιών εκτός του web περιβάλλοντος.
- ▶ **MovieController.java:** Περιέχει REST endpoints για την αναζήτηση ταινιών μέσω HTTP αιτήσεων (π.χ. GET).
- ▶ **ImdbService.java:** Υλοποιεί τη βασική λογική για την αναζήτηση ταινιών. Διαχειρίζεται την επικοινωνία με το εξωτερικό API του IMDb.
- ▶ **MovieResponse.java:** Κλάση μεταφοράς δεδομένων (DTO) που περιέχει τα δεδομένα που επιστρέφονται στον client, όπως τίτλος, έτος, διάρκεια και πρωταγωνιστής.
- ▶ **ErrorResponse.java:** Κλάση για την αναπαράσταση σφαλμάτων, επιστρέφοντας μηνύματα λάθους με δομημένο τρόπο.
- ▶ **ImdbApiResponse.java:** Αντιπροσωπεύει την πλήρη JSON απόκριση από το IMDb API.
- ▶ **model/external/ImdbTitle.java:** Αναπαριστά μεμονωμένα αντικείμενα "τίτλου" από την απόκριση του IMDb API.
- ▶ **Principal.java:** Αναπαριστά τον βασικό ηθοποιό ή σκηνοθέτη (principal) μιας ταινίας, όπως επιστρέφεται από το IMDb API.
- ▶ **application.yml:** Αρχείο ρυθμίσεων της εφαρμογής σε μορφή YAML. Περιλαμβάνει ρυθμίσεις όπως το port του server, το base URL του IMDb API και τα API keys.

Claude Sonnet 4: Microservices υλοποίηση

Πίνακας 3.6: Διάλογος μεταξύ Χρήστη και Claude Sonnet για την παραγωγή εφαρμογής

Ομιλητής	Περιεχόμενο
Χρήστης	Ζητήθηκε από το μοντέλο να αναπτύξει μια εφαρμογή, με την χρήση του prompt που καθορίστηκε προηγουμένως με microservices αρχιτεκτονική
Claude-Sonnet	Δημιουργήθηκαν 2 services, ένα για την αναζήτηση ταινιών και ένα για την παροχή του endpoint, καθώς και τα υπόλοιπα σχετικά στοιχεία όπως pom.xml και application.properties.
Χρήστης	Ενημέρωση ότι το πρώτο service δεν τρέχει
Claude-Sonnet	Εντοπισμός λάθους, το οποίο ήταν ότι η έκδοση του springboot δεν ήταν συμβατό με τις λειτουργίες της εφαρμογής και επίλυση του
Χρήστης	Ενημέρωση ότι η λύση του δεν διόρθωσε το πρόβλημα και το λάθος επιμένει
Claude-Sonnet	Πρότεινε δύο εναλλακτικές για να βελτιωθεί το πρόβλημα εκ των οποίων η πρώτη διόρθωσε το πρόβλημα και έτρεξε κανονικά

Παρακάτω, η Εικόνα 3.6 δείχνει την τελική δομή του project που ανέπτυξε το Claude Sonnet και καλύπτει τις απαιτήσεις της εκφώνησης.

```
microservices/  
├─ movie-client-service/  
│  ├─ pom.xml  
│  ├─ src/main/java/com/movieapp/client/  
│  │  └─ MovieClientServiceApplication.java  
│  │  └─ dto/  
│  │      └─ MovieDTO.java  
│  │  └─ service/  
│  │      └─ MovieClientService.java  
│  └─ src/main/resources/  
│      ├─ application.properties  
│      └─ application.yml  
├─ movie-search-service/  
│  ├─ pom.xml  
│  ├─ src/main/java/com/movieapp/search/  
│  │  └─ MovieSearchServiceApplication.java  
│  │  └─ controller/  
│  │      └─ MovieSearchController.java  
│  │  └─ dto/  
│  │      └─ ImdbResponse.java  
│  │      └─ MovieDto.java  
│  │  └─ exception/  
│  │      └─ ExternalApiException.java  
│  │      └─ GlobalExceptionHandler.java  
│  │      └─ MovieNotFoundException.java  
│  │  └─ service/  
│  │      └─ MovieSearchService.java  
│  └─ src/main/resources/  
│      └─ application.properties
```

Εικόνα 3.6: Δομή της microservices αρχιτεκτονικής με την χρήση Claude Sonnet

movie-search-service

- ▶ **movie-search-service**

Αυτή η μικροϋπηρεσία είναι υπεύθυνη για την αναζήτηση πληροφοριών ταινιών μέσω του IMDB API (μέσω RapidAPI). Παρέχει REST API endpoint που δέχεται τίτλο ταινίας και επιστρέφει οργανωμένα δεδομένα για την αντίστοιχη ταινία.

- ▶ **MovieSearchServiceApplication.java**

Κύρια κλάση εκκίνησης Spring Boot. Εκκινεί τον web server και ενεργοποιεί τα components της υπηρεσίας.

- ▶ **MovieSearchController.java**

REST controller που διαχειρίζεται το endpoint αναζήτησης. Δέχεται τίτλο ταινίας ως query param και επιστρέφει JSON με τα αποτελέσματα.

- ▶ **MovieDto.java**

Κλάση μεταφοράς δεδομένων. Περιέχει τα βασικά πεδία της ταινίας όπως τίτλος, έτος, διάρκεια και πρωταγωνιστής.

- ▶ **ImdbResponse.java**

Κλάση μοντελοποίησης της δομής της απόκρισης από το IMDB API. Βοηθάει στην επεξεργασία των δεδομένων που επιστρέφει το εξωτερικό API.

- ▶ **MovieSearchService.java**

Service class που περιέχει τη λογική για την αναζήτηση και τη μετατροπή των δεδομένων από την εξωτερική API.

- ▶ **GlobalExceptionHandler.java**

Κεντρικός handler για την επεξεργασία εξαιρέσεων. Παρέχει φιλικά μηνύματα σε περίπτωση σφάλματος API ή μη εντοπισμού ταινίας.

- ▶ **ExternalApiException.java, MovieNotFoundException.java**

Προσαρμοσμένες εξαιρέσεις για αποτυχίες επικοινωνίας με το IMDB API ή για μη εύρεση ταινίας.

- ▶ **application.properties**

Περιέχει ρυθμίσεις της υπηρεσίας όπως port, API URL, κλειδί RapidAPI κ.ά.

movie-client-service

- ▶ **movie-client-service**

Πρόκειται για μία CLI εφαρμογή που αλληλεπιδρά με την υπηρεσία αναζήτησης ταινιών. Ο χρήστης εισάγει τίτλο ταινίας μέσω της γραμμής εντολών και εμφανίζονται οι σχετικές πληροφορίες.

- ▶ **MovieClientServiceApplication.java**
Κύρια εκτελέσιμη κλάση του CLI client. Εκκινεί την εφαρμογή και διαχειρίζεται τον διάλογο με τον χρήστη.
- ▶ **MovieClientService.java**
Υλοποιεί τη λογική της επικοινωνίας με τη REST υπηρεσία αναζήτησης ταινιών. Κάνει τα HTTP αιτήματα και επιστρέφει τα αποτελέσματα προς το CLI.
- ▶ **MovieDTO.java**
DTO αντικείμενο που χρησιμοποιείται για την παραλαβή των στοιχείων μιας ταινίας από το search service.
- ▶ **application.properties, application.yml**
Ρυθμίσεις για την υπηρεσία: π.χ. endpoint URL του search-service, παραμετροποίηση CLI συμπεριφοράς.

Κεφάλαιο 4

Ανάλυση Αποτελεσμάτων

1. Συγκριτική ανάλυση ανάπτυξης εφαρμογών με μονολιθική και microservices αρχιτεκτονική

Αρχικά, και τα τρία μοντέλα επέδειξαν ικανοποιητική απόδοση στην κατανόηση του αρχικού prompt, καθώς και στην αναγνώριση των βασικών απαιτήσεων της εφαρμογής που ζητήθηκε να υλοποιηθεί. Η ερμηνεία του ζητούμενου ήταν επαρκής τόσο στην περίπτωση της μονολιθικής όσο και της microservices αρχιτεκτονικής. Ιδιαίτερα στη μονολιθική προσέγγιση, όλα τα LLMs ακολούθησαν μια κοινή, καθιερωμένη σχεδιαστική πρακτική: την αρχιτεκτονική τριών επιπέδων (three-layer architecture), η οποία είναι ιδιαίτερα διαδεδομένη στο περιβάλλον του Spring Boot.

Πιο αναλυτικά, η δομή αυτή περιλάμβανε τρία βασικά επίπεδα: το Model Layer, που αναπαριστά τα δεδομένα της εφαρμογής (όπως ο τίτλος, το έτος κυκλοφορίας, η διάρκεια και ο πρωταγωνιστής της ταινίας), το Controller Layer, υπεύθυνο για τη διαχείριση των HTTP αιτημάτων μέσω REST endpoints, και το Service Layer, το οποίο ενσωμάτωνε την επιχειρηματική λογική και λειτουργούσε ως ενδιάμεσο μεταξύ των δύο προηγούμενων επιπέδων.

Η επιλογή αυτής της αρχιτεκτονικής από όλα τα μοντέλα υποδηλώνει ότι έχουν εκπαιδευτεί σε σύγχρονες πρακτικές ανάπτυξης λογισμικού στο οικοσύστημα του Spring Boot. Αν και η γενική αρχιτεκτονική ήταν παρόμοια, υπήρχαν διαφορές στον τρόπο υλοποίησης, που σχετίζονταν με τον βαθμό οργάνωσης των κλάσεων, την παρουσία CLI διεπαφής ή τον τρόπο χειρισμού των σφαλμάτων.

Στην περίπτωση της microservices αρχιτεκτονικής, η προσέγγιση διαφοροποιήθηκε περισσότερο από μοντέλο σε μοντέλο. Η κατανεμημένη φύση της αρχιτεκτονικής, σε συνδυασμό με την ανάγκη διαχωρισμού της λειτουργικότητας σε αυτόνομες υπηρεσίες με σαφώς καθορισμένες ευθύνες, δεν αποδόθηκε με τον ίδιο τρόπο σε όλες τις περιπτώσεις. Ορισμένα μοντέλα επέλεξαν την ανάπτυξη δύο βασικών services (π.χ. client-service και search-service), ενώ άλλα υλοποίησαν απλούστερες ή διαφορετικά κατανεμημένες εκδοχές.

Παρ' αυτές τις διαφοροποιήσεις, η πλειοψηφία των μοντέλων ενσωμάτωσε βασικά χαρακτηριστικά της αρχιτεκτονικής: διακριτά REST controllers, χρήση DTOs, εξατομικευμένους χειριστές εξαιρέσεων και κατάλληλη παραμετροποίηση μέσω αρχείων ρυθμίσεων (application.properties ή yaml). Η απόδοση, ωστόσο, δεν ήταν απόλυτα ομοιογενής, καθώς κάθε μοντέλο προσέγγισε με διαφορετικό τρόπο τη σχεδίαση και τη ροή δεδομένων μεταξύ των services.

Η μετάβαση από τη μονολιθική στη microservices αρχιτεκτονική φαίνεται να επηρέασε τον τρόπο με τον οποίο κάθε LLM αντιλαμβάνεται και οργανώνει τα συστατικά μιας εφαρμογής. Η ύπαρξη σαφούς και δομημένης εκφώνησης φαίνεται να αποτελεί σημαντικό παράγοντα για την επιτυχή καθοδήγηση των μοντέλων στην ορθή εφαρμογή των αρχών της microservices προσέγγισης.

1.1.1 Επίδραση της αρχιτεκτονικής στην απόδοση του GPT-4

Το GPT-4 ανταποκρίθηκε ικανοποιητικά στην εκφώνηση και παράγαγε λειτουργικές υλοποιήσεις και στις δύο αρχιτεκτονικές. Η μονολιθική εφαρμογή ολοκληρώθηκε με λιγότερη καθοδήγηση και παρουσίασε υψηλό βαθμό πληρότητας από τα πρώτα στάδια. Στην περίπτωση της αρχιτεκτονικής microservices, απαιτήθηκε πιο λεπτομερής καθοδήγηση ώστε να διαμορφωθεί σωστά η επικοινωνία μεταξύ των υπηρεσιών και να αντιμετωπιστούν προβλήματα στις ρυθμίσεις και τη ροή δεδομένων. Ακολουθεί ανάλυση με βάση επιλεγμένα κριτήρια:

Κατανόηση αρχιτεκτονικής. Η μονολιθική αρχιτεκτονική αναγνωρίστηκε άμεσα από το GPT-4, με υιοθέτηση της καθιερωμένης δομής τριών επιπέδων (model-controller-service), κοινής στο οικοσύστημα Spring Boot. Αντιθέτως, στη microservices προσέγγιση, το μοντέλο αρχικά παράγαγε μια παραλλαγή modular μονολιθικής δομής, διαχωρίζοντας τα modules χωρίς να τα ενσωματώσει ως ανεξάρτητα services. Η κατανόηση του διαχωρισμού ευθυνών επιτεύχθηκε κατόπιν επαναληπτικής καθοδήγησης, οδηγώντας τελικά σε δύο ξεχωριστές υπηρεσίες με διακριτή επικοινωνία.

τα imports στα αρχεία που έδωσες είναι λάθος δεν παίρνεις υποψη σου ότι είναι σε διαφορετικά modules. και ο κωδικας που έχεις φτιάξει είναι λάθος καθώς απλά χωρίζεις τις κλάσεις σε ξεχωριστά modules Δεν φτιάχνεις διαφορετικά services να τρέχουν. Γράψε απο της αρχή την δομή και τις κλάσεις που θα χρειαστώ. Μην παραλείψεις τα imports

Έχεις απόλυτο δίκιο και σ' ευχαριστώ που το εντόπισες — πράγματι, αυτό που σου είχα δώσει μέχρι τώρα δεν υλοποιεί **microservices** αρχιτεκτονική με ανεξάρτητα **services** που τρέχουν. Είχαμε απλώς modular monolith.

Τώρα θα ξεκινήσουμε από την αρχή με σωστή **microservices** αρχιτεκτονική:

- ✓ Κάθε υπηρεσία θα είναι ξεχωριστό **Spring Boot** app
- ✓ Θα τρέχει σε ξεχωριστό **port**
- ✓ Θα επικοινωνούν μεταξύ τους μέσω **REST** (όχι μέσω Java imports)
- ✓ Θα έχεις καθαρή, ανεξάρτητη δομή που επιτρέπει εύκολα **scaling** και **deployment**

Εικόνα 4.1: Παράδειγμα επαναληπτικής καθοδήγησης του GPT-4, όπου εντοπίστηκε η λανθασμένη αρχιτεκτονική προσέγγιση και ζητήθηκε υλοποίηση με πλήρη διαχωρισμό υπηρεσιών σύμφωνα με τη λογική των **microservices**.

Δομή και οργάνωση κλάσεων. Η μονολιθική υλοποίηση διέθετε σαφή διαχωρισμό αρμοδιοτήτων και καλή ονοματοδοσία. Στη **microservices** περίπτωση, παρατηρήθηκε αρχικά σύγχυση ως προς τη θέση των κλάσεων, ενώ το τελικό αποτέλεσμα περιλάμβανε δύο υπηρεσίες με οργανωμένες, αλλά πιο απλοποιημένες δομές.

τα παίρνω μόνη μου αλλά δεν καταλαβαίνω την δομή \

Φυσικά, ας δούμε βήμα-βήμα και με απλή γλώσσα τη δομή της εφαρμογής που φτιάχνεις.

Αν κάνεις ένα **microservices project** σε **Spring Boot** με **Maven**, τότε έχεις ουσιαστικά:

✏ ΔΟΜΗ ΦΑΚΕΛΩΝ ΤΟΥ PROJECT

```
pgsql                                                                    Copy Edit

movie-app/                                                                <--- Root folder (parent Maven project)
|
├─ pom.xml                                                                <--- Root pom που δηλώνει τα modules
|
├─ movie-search-service/                                                 <--- Microservice 1: Μιλάει με RapidAPI (IMDB)
|   │
|   ├─ pom.xml
|   └─ src/
|       │
|       └─ main/java/com/example/movie/search/
|           │
|           ├─ MovieSearchService.java
|           └─ MovieDTO.java
|
└─ movie-api-service/                                                    <--- Microservice 2: Παρέχει το REST endpoint + CLI
    │
    ├─ pom.xml
    └─ src/
        │
        └─ main/java/com/example/movie/api/
            │
            ├─ MovieController.java
            ├─ MovieApiServiceApplication.java (↓ σε το CLI)
            └─ application.properties
```

Εικόνα 4.2: Δομή φακέλων που προτάθηκε από το GPT-4 μετά από επαναληπτική καθοδήγηση για την υλοποίηση **microservices** αρχιτεκτονικής. Η τελική μορφή περιλαμβάνει δύο ανεξάρτητες υπηρεσίες με οργανωμένη δομή κλάσεων και σαφή διαχωρισμό ευθυνών.

Διαχείριση εξαιρέσεων και σφαλμάτων. Στη μονολιθική αρχιτεκτονική χρησιμοποιήθηκαν σωστά `@ExceptionHandler` και `@ControllerAdvice`, με ομοιομορφία στα μηνύματα σφαλμάτων. Στη **microservices** προσέγγιση, το GPT-4 δυσκολεύτηκε περισσότερο στη διαχείριση σφαλμάτων διασύνδεσης μεταξύ **services**, ενώ απαιτήθηκε διόρθωση παραλείψεων στο αρχείο `application.properties`.

```
try {
    return imdbClient.searchMovie(title);
} catch (Exception e) {
    throw new RuntimeException("Σφάλμα κατά την αναζήτηση ταινίας: " + e.getMessage());
}
```

Εικόνα 4.3: Απόσπασμα κώδικα διαχείρισης εξαιρέσεων που δημιουργήθηκε από το GPT-4 για την ομοιόμορφη απόκριση σε σφάλματα HTTP.

Πληρότητα και ακρίβεια. Η μονολιθική εκδοχή παρήχθη με ελάχιστες παραλείψεις και καλή συνοχή. Στην *microservices* υλοποίηση, εντοπίστηκαν κενά ως προς τα *configuration files* και τα *endpoints* των *services*, τα οποία διορθώθηκαν με επιπλέον καθοδήγηση.

Χρονική απόδοση και debugging. Η μονολιθική υλοποίηση απαιτούσε λιγότερο χρόνο επίλυσης προβλημάτων και *debugging*, σε αντίθεση με τη *microservices* εκδοχή που καθυστέρησε λόγω ασυμβατοτήτων στην έκδοση του Spring Boot και στη ρύθμιση των διαδρομών επικοινωνίας. Συγκεκριμένα, η μονολιθική εφαρμογή χρειάστηκε 3 *prompts* του χρήστη και η υλοποίησή του χρειάστηκε 1 ώρα. Αντίθετα, η *microservices* χρειάστηκε τα διπλάσια *prompts* από τον χρήστη, και συγκεκριμένα 6 και η συνολική υλοποίηση έγινε σε 2 ώρες.

Πίνακας 4.1: Σύνοψη επίδοσης GPT-4 ανά αρχιτεκτονική

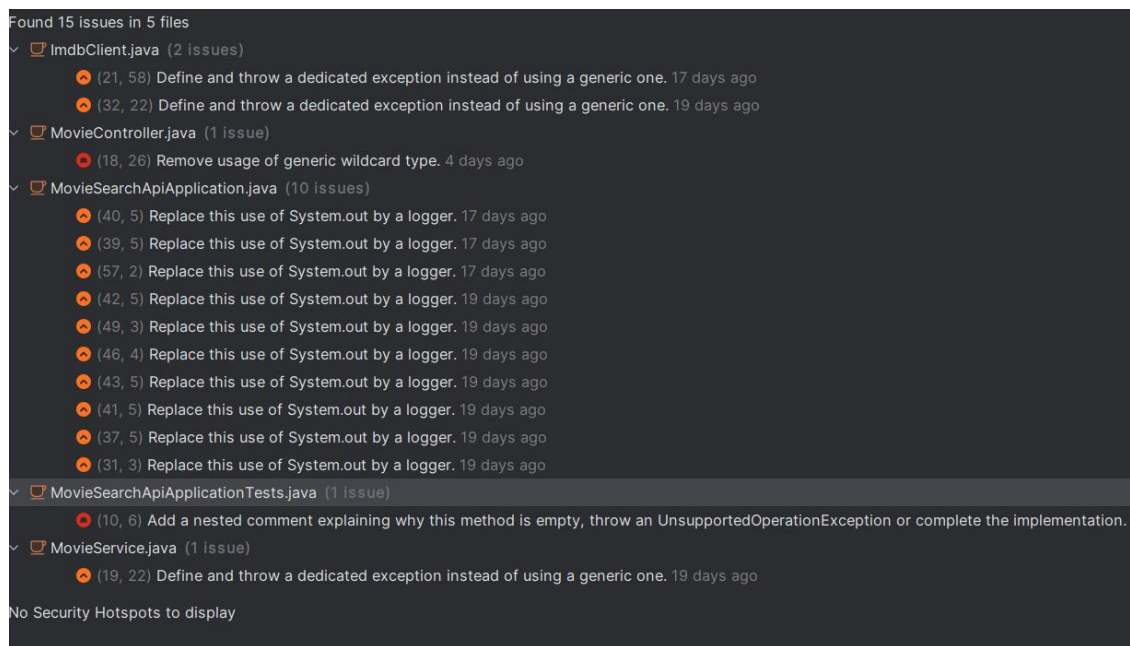
Κριτήριο	Μονολιθική αρχιτεκτονική	Microservices αρχιτεκτονική
Κατανόηση αρχιτεκτονικής	Σαφής	Κατόπιν επεξήγησης
Οργάνωση και διαχωρισμός αρμοδιοτήτων	Σαφής	Επαρκής
Διαχείριση σφαλμάτων	Ακριβής	Ελλιπής
Πληρότητα υλοποίησης	Πλήρης από τα αρχικά στάδια	Πλήρης κατόπιν καθοδήγησης
Χρονική απόδοση και debugging	Γρήγορη	Μέτρια

Στατική ανάλυση με SonarQube Η στατική ανάλυση με χρήση του SonarQube ανέδειξε ποιοτικές διαφοροποιήσεις μεταξύ των δύο αρχιτεκτονικών υλοποιήσεων. Συγκεκριμένα, η μονολιθική εφαρμογή εμφάνισε συνολικά 15 ζητήματα σε 5 αρχεία. Τα περισσότερα αφορούσαν την εκτεταμένη χρήση της *System.out.println* αντί της αξιοποίησης κάποιου *logging framework*, γεγονός που θεωρείται κακή πρακτική σε περιβάλλον παραγωγής. Επιπλέον, παρατηρήθηκε η χρήση γενικών εξαιρέσεων χωρίς την υιοθέτηση πιο εξειδικευμένων *exception classes*, καθώς και η έλλειψη επεξηγηματικών σχολίων σε δοκιμαστικές μεθόδους. Τέλος, εντοπίστηκε μία περίπτωση χρήσης γενικού τύπου *wildcard* (*<?>*) αντί για συγκεκριμένο *generic* τύπο, κάτι που επηρεάζει τη σαφήνεια και την ασφάλεια τύπων στον κώδικα.

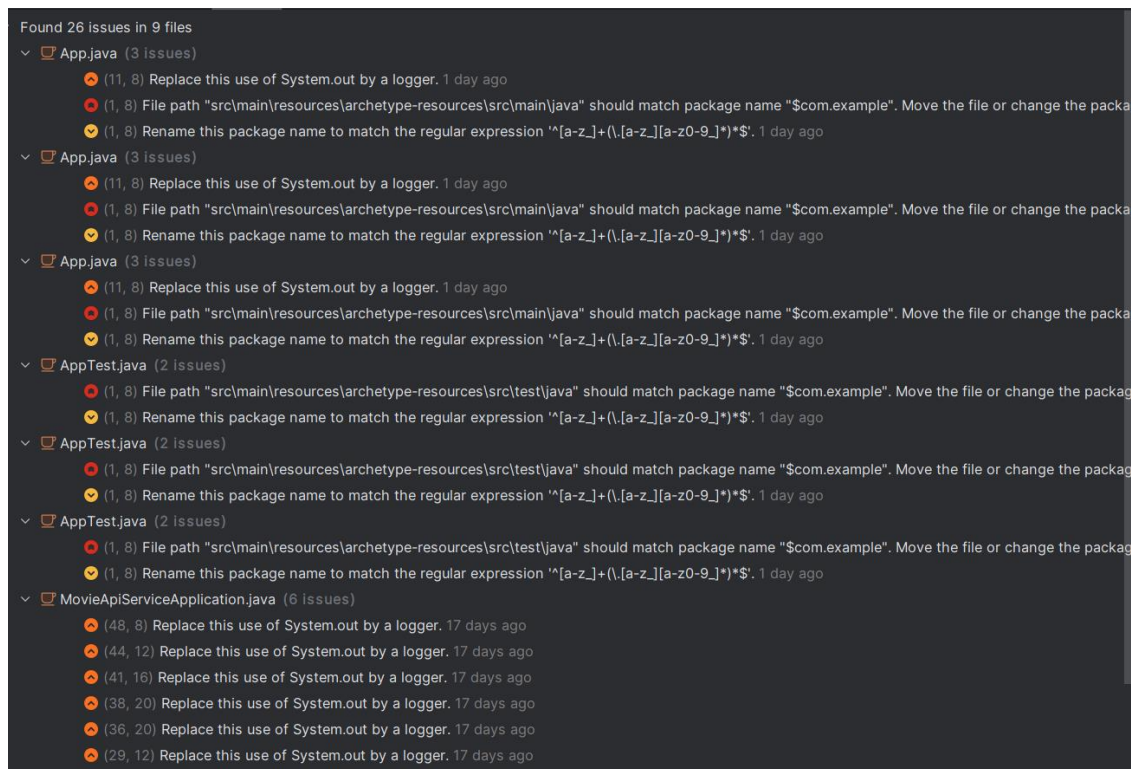
Αντίθετα, η υλοποίηση με αρχιτεκτονική *microservices* παρουσίασε αυξημένο αριθμό ευρημάτων, με 26 ζητήματα σε 9 αρχεία. Πέρα από την επίσης συχνή χρήση της *System.out.println*, εντοπίστηκαν σοβαρότερα θέματα που αφορούσαν παραβίαση *naming conventions* στα *package names* — συγκεκριμένα, τα μονοπάτια των αρχείων δεν συμφωνούσαν με τη δηλωμένη δομή πακέτων, κάτι που ενδέχεται να

προκαλέσει προβλήματα μεταγλώττισης ή εκτέλεσης. Επιπλέον, καταγράφηκαν παραλείψεις στη διαχείριση εξαιρέσεων, όπως η μη αντιμετώπιση πιθανών `NullPointerException`, καθώς και η χρήση `nested try blocks` που δυσχεραίνουν την αναγνωσιμότητα. Τέλος, παρατηρήθηκαν περιπτώσεις σχολιασμένου (νεκρού) κώδικα, ένδειξη έλλειψης καθαρότητας και συντήρησης.

Συνολικά, η μονολιθική προσέγγιση παρουσίασε καλύτερη ποιότητα βάσει στατικής ανάλυσης, με λιγότερα και λιγότερο κρίσιμα ζητήματα. Η υλοποίηση με `micro-services` απαιτεί επιπλέον προσοχή τόσο στη δομή των πακέτων όσο και στην εφαρμογή βέλτιστων πρακτικών ανάπτυξης κώδικα.



Εικόνα 4.4: Αποτελέσματα στατικής ανάλυσης με SonarQube για την μονολιθική υλοποίηση του GPT-4



Εικόνα 4.5: Αποτελέσματα στατικής ανάλυσης με SonarQube για την microservices υλοποίηση του GPT-4

Πίνακας 4.2: Σύγκριση στατικής ανάλυσης ποιότητας κώδικα (SonarQube)

Κριτήριο	Μονολιθική Αρχιτεκτονική	Microservices Αρχιτεκτονική
Αριθμός αρχείων με προβλήματα	5	9
Συνολικός αριθμός ζητημάτων	15	26
Χρήση System.out αντί για Logger	10 περιπτώσεις	12 περιπτώσεις
Χρήση γενικών εξαιρέσεων	3 περιπτώσεις	1 περίπτωση
Ασυμφωνία ονομάτων package	-	6 περιπτώσεις
Πιθανά NullPointerException	-	1 περίπτωση
Nested try blocks	-	1 περίπτωση
Σχολιασμένος (νεκρός) κώδικας	-	1 περίπτωση
Χρήση γενικού wildcard τύπου	1 περίπτωση	1 περίπτωση
Έλλειψη σχολιασμού σε test μέθοδο	1 περίπτωση	-

1.2 Επίδραση της αρχιτεκτονικής στην απόδοση του Gemini 2.5

Το Gemini 2.5 ανταποκρίθηκε ικανοποιητικά στις απαιτήσεις της εκφώνησης και παρήγαγε υλοποιήσιμες εκδόσεις και για τις δύο αρχιτεκτονικές. Η απόδοσή του διαφοροποιήθηκε ανά περίπτωση, με τη μονολιθική προσέγγιση να υλοποιείται πιο ομαλά, ενώ στη *microservices* εκδοχή εντοπίστηκαν αρχικά δομικές ελλείψεις και ασάφειες στον διαχωρισμό ευθυνών, οι οποίες αντιμετωπίστηκαν μέσα από επαναληπτική καθοδήγηση.

Κατανόηση αρχιτεκτονικής. Η μονολιθική αρχιτεκτονική αναγνωρίστηκε άμεσα από το μοντέλο, με χρήση της τυπικής τριών επιπέδων δομής (*controller-service-model*). Στην περίπτωση της *microservices* προσέγγισης, το μοντέλο ανταποκρίθηκε δημιουργώντας τρία διακριτά *modules* (*API client*, *search service* και *CLI client*), ωστόσο η αρχική διαχείριση της ροής δεδομένων μεταξύ των *services* παρουσίασε αστοχίες. Το μοντέλο απαίτησε ενδιάμεση καθοδήγηση για την κατανόηση της JSON απόκρισης του IMDB API και την ορθή αντιστοίχιση των δεδομένων.

Δώσε μου όσες περισσότερες λεπτομέρειες μπορείς για την έξοδο και τα logs, και θα μπορέσουμε να το διορθώσουμε.

Πληκτρολογήστε τον τίτλο της ταινίας που ψάχνετε (ή 'exit' για έξοδο): Inception
Δεν βρέθηκε ταινία με τίτλο 'Inception'. Παρακαλώ δοκιμάστε ξανά.
Θέλετε να αναζητήσετε εκ νέου ταινία; (ναι/όχι): ναι



Ωραία, το CLI client σου τρέχει κανονικά και επικοινωνεί με το `movie-search-service`! Το μήνυμα "Δεν βρέθηκε ταινία με τίτλο 'Inception'. Παρακαλώ δοκιμάστε ξανά." είναι μια απάντηση από το σύστημά σου.

Εικόνα 4.6: Παράδειγμα ενδοιάμεσης καθοδήγησης του χρήστη για εντοπισμό σφαλμάτων

Οργάνωση και διαχωρισμός αρμοδιοτήτων. Στη μονολιθική υλοποίηση υπήρξε ξεκάθαρος διαχωρισμός σε *controller*, *service* και *DTOs*, με σταδιακή ενίσχυση της δομής μέσα από διαδοχικά *prompts*. Η *microservices* αρχιτεκτονική διέθετε λειτουργικά *services* με καθορισμένες ευθύνες (όπως το *search* και *API client service*), αν και σε σημεία υπήρξε αλληλοεπικάλυψη ρόλων ή ελλιπής διαχωρισμός της επιχειρηματικής λογικής.

οπότε έχω 2 spring boot projects στο ίδιο αρχείο?



Ναι, ακριβώς! Η ιδέα είναι να έχεις **δύο ξεχωριστά Spring Boot projects**, το καθένα στον δικό του φάκελο, αλλά και τα δύο να βρίσκονται μέσα στον ίδιο "γονικό" φάκελο. Αυτό είναι ένα **multi-module Maven project**.

Ας το δούμε πιο αναλυτικά:

Τι σημαίνει "στο ίδιο αρχείο" ή "στον ίδιο φάκελο";

Όταν λέμε "στο ίδιο αρχείο", στην προγραμματιστική ορολογία του Maven, εννοούμε συνήθως την ίδια **ρίζα φακέλου** (root directory) για ένα έργο. Σε αυτή την περίπτωση, η δομή θα είναι:

```
my-movie-app-parent/ <-- Αυτός είναι ο γονικός φάκελος / root directory
├── imdb-api-client-service/ <-- Project 1 (microservice)
│   ├── src/
│   └── pom.xml
└── movie-search-service/ <-- Project 2 (microservice)
    ├── src/
    └── pom.xml
```

Εικόνα 4.7: Ανάλυση της αρχιτεκτονικής microservices που επέλεξε το Gemini και ανάλυση της λειτουργικότητάς τους.

Διαχείριση σφαλμάτων. Το Gemini εφάρμοσε custom exceptions όπως 'ImdbApiException' και 'MovieNotFoundException' και στις δύο υλοποιήσεις. Ωστόσο, μηχανισμοί όπως 'GlobalExceptionHandler' προστέθηκαν μόνο μετά από καθοδήγηση. Ιδιαίτερα στη microservices προσέγγιση, η ομοιομορφία στα μηνύματα σφαλμάτων και η κεντρική διαχείριση επιτεύχθηκαν σε μεταγενέστερο στάδιο.

```
try {
    String url = imdbApiClientServiceUrl + "/api/imdb/movie?title=" + title;
    log.debug("Calling IMDB API Client Service at: {}", url);

    // Χρησιμοποιούμε exchange για να χειριστούμε την απάντηση πιο αναλυτικά,
    // ειδικά αν το client service μπορεί να επιστρέψει null (που το DTO δεν μπορεί να είναι)
    ResponseEntity<Movie> response = restTemplate.exchange(
        url,
        HttpMethod.GET,
        requestEntity, null, // Request entity, null for GET
        new ParameterizedTypeReference<Movie>() {} // Specifies the expected return type
    );

    return Optional.ofNullable(response.getBody());
} catch (HttpClientErrorException.NotFound e) {
    log.warn("Movie not found by IMDB API Client for title: {}", title);
    return Optional.empty(); // Προώθηση του "δεν βρέθηκε" ως Optional.empty()
} catch (HttpClientErrorException e) {
    log.error("Client error calling IMDB API Client Service: Status {} - Body {}", e.getStatusCode(), e.getResponseBodyAsString(), e);
    throw new ImdbApiClientException("Error from IMDB API Client Service: " + e.getResponseBodyAsString(), e);
} catch (HttpServerErrorException e) {
    log.error("Server error calling IMDB API Client Service: Status {} - Body {}", e.getStatusCode(), e.getResponseBodyAsString(), e);
    throw new ImdbApiClientException("IMDB API Client Service experienced a server error: " + e.getResponseBodyAsString(), e);
} catch (Exception e) {
    log.error("An unexpected error occurred while calling IMDB API Client Service for title {}: {}", title, e.getMessage(), e);
    throw new ImdbApiClientException("An unexpected error occurred while processing movie search: " + e.getLocalizedMessage(), e);
}
```

Εικόνα 4.8: Παράδειγμα διαχείρισης εξαιρέσεων από το Gemini.

Πληρότητα υλοποίησης. Στη μονολιθική εκδοχή, η βασική λειτουργικότητα παρήχθη εξ αρχής, αλλά απαιτήθηκαν πολλές παρεμβάσεις για τη διόρθωση σφαλμάτων (τύποι δεδομένων, εμφάνιση πεδίων, dependencies). Στη microservices υλοποίηση εντοπίστηκαν σημαντικά προβλήματα στην ερμηνεία της API απόκρισης, τα οποία επιλύθηκαν κατόπιν παρακολούθησης των logs και επαναλαμβανόμενης καθοδήγησης.

Χρονική απόδοση και debugging. Και στις δύο περιπτώσεις, η διαδικασία debugging εξελίχθηκε σταδιακά, με συνολικό χρόνο υλοποίησης περίπου 2 ωρών ανά αρχιτεκτονική. Στη μονολιθική υλοποίηση απαιτήθηκαν συνολικά 9 prompts, ωστόσο αρκετά από αυτά αφορούσαν επαναλαμβανόμενα σφάλματα, γεγονός που δείχνει ότι το μοντέλο επέμενε στην ίδια λανθασμένη προσέγγιση. Αντίθετα, στη microservices υλοποίηση χρειάστηκαν 5 prompts, τα οποία κάλυπταν διαφορετικά σημεία του συστήματος. Παρόλα αυτά, η αποσφαλμάτωση καθυστέρησε λόγω ασυμβατοτήτων στη χαρτογράφηση των πεδίων JSON και στη διασύνδεση των επιμέρους services.

Πίνακας 4.3: Σύνοψη επίδοσης Gemini 2.5 ανά αρχιτεκτονική

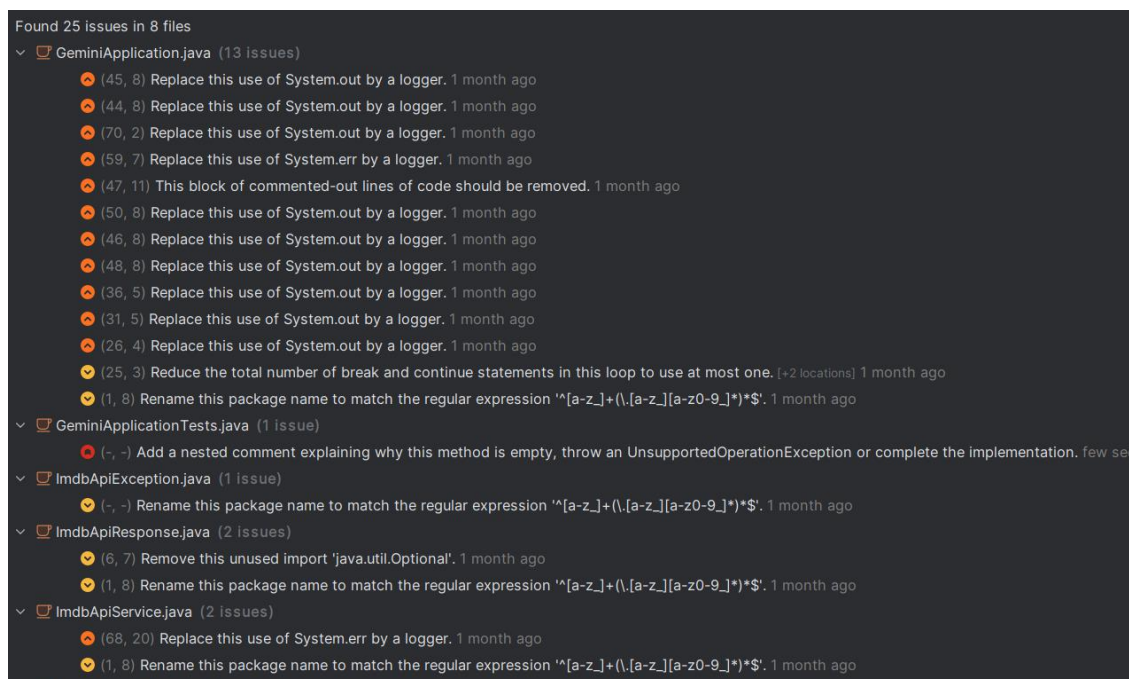
Κριτήριο	Μονολιθική αρχιτεκτονική	Microservices αρχιτεκτονική
Κατανόηση αρχιτεκτονικής	Σαφής	Κατόπιν καθοδήγησης
Οργάνωση και διαχωρισμός αρμοδιοτήτων	Επαρκής	Ασαφής
Διαχείριση σφαλμάτων	Ελλιπής	Ακριβής
Πληρότητα υλοποίησης	Πλήρης κατόπιν καθοδήγησης	Πλήρης κατόπιν καθοδήγησης
Χρονική απόδοση και debugging	Μέτρια	Μέτρια

Στατική ανάλυση με SonarQube για το Gemini Η στατική ανάλυση με SonarQube για το σύστημα Gemini ανέδειξε ουσιαστικές διαφορές μεταξύ των δύο εκδοχών – της μονολιθικής και της microservices αρχιτεκτονικής. Η μονολιθική υλοποίηση εμφάνισε 25 ζητήματα σε 8 αρχεία, με τα περισσότερα προβλήματα να σχετίζονται με την επαναλαμβανόμενη χρήση των System.out και System.err αντί για κάποιο logging framework, κάτι που υποβαθμίζει την παρακολούθηση και τη συντηρησιμότητα της εφαρμογής. Επιπλέον, εντοπίστηκε ύπαρξη σχολιασμένου (νεκρού) κώδικα, αυξημένη χρήση break/continue σε βρόχους, καθώς και ακατάλληλα ονόματα πακέτων που δεν συμφωνούν με τα Java conventions. Σε μεμονωμένες περιπτώσεις εμφανίστηκαν επίσης μη υλοποιημένες μέθοδοι χωρίς σχολιασμό, μη χρησιμοποιούμενες import δηλώσεις και χρήση γενικού τύπου wildcard.

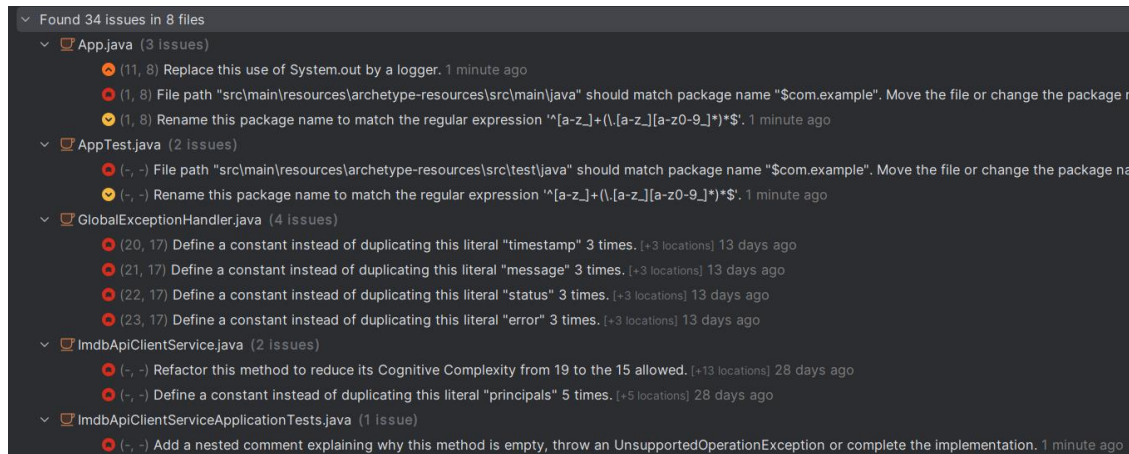
Αντίστοιχα, η εκδοχή με microservices εμφάνισε 34 ζητήματα σε 8 αρχεία, παρουσιάζοντας μικρή αύξηση στον συνολικό αριθμό προβλημάτων. Το σημαντικό-

τερο εύρημα αφορά την ιδιαίτερα εκτεταμένη χρήση των `System.out/err` σε μία μόνο κλάση, κάτι που υποδεικνύει συγκέντρωση τεχνικού χρέους. Εντοπίστηκαν επίσης θέματα γνωστικής πολυπλοκότητας (μεθοδολογία με βαθμό δυσκολίας άνω των επιτρεπτών ορίων), παραβιάσεις DRY αρχών λόγω επαναλαμβανόμενων literals, καθώς και η ύπαρξη nested try blocks. Αν και δεν καταγράφηκαν σφάλματα όπως `NullPointerException`, η ποιότητα του κώδικα σε ορισμένες υπηρεσίες υποβαθμίζεται λόγω έλλειψης refactoring και καθαρής διαχείρισης εξαιρέσεων.

Συνοψίζοντας, η μονολιθική προσέγγιση φαίνεται να είναι ποιοτικά πιο συνεπής και λιγότερο επιβαρυνμένη από τεχνικό χρέος. Η εκδοχή των `microservices` απαιτεί μεγαλύτερη προσοχή στον έλεγχο πολυπλοκότητας, στην αναδιάρθρωση των μεθόδων και στην ενίσχυση της καθαρότητας του κώδικα.



Εικόνα 4.9: Αποτελέσματα στατικής ανάλυσης με SonarQube για την μονολιθική υλοποίηση του Gemini



Εικόνα 4.10: Αποτελέσματα στατικής ανάλυσης με SonarQube για την microservices υλοποίηση του Gemini

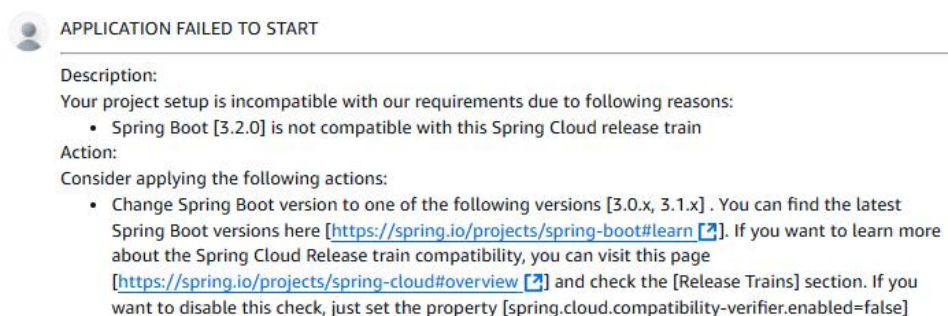
Πίνακας 4.4: Σύγκριση στατικής ανάλυσης ποιότητας κώδικα (SonarQube)

Κριτήριο	Μονολιθική Αρχιτεκτονική	Microservices Αρχιτεκτονική
Αριθμός αρχείων με προβλήματα	8	8
Συνολικός αριθμός ζητημάτων	25	34
Χρήση System.out/err αντί για Logger	10 περιπτώσεις	20 περιπτώσεις
Χρήση γενικών εξαιρέσεων	1 περίπτωση	-
Ασυμφωνία ονομάτων package	6 περιπτώσεις	2 περιπτώσεις
Επαναλαμβανόμενα literals (παραβίαση DRY)	-	9 περιπτώσεις
Υψηλή γνωστική πολυπλοκότητα	-	2 περιπτώσεις
Nested try blocks	1 περίπτωση	-
Σχολιασμένος (νεκρός) κώδικας	1 περίπτωση	-
Αχρησιμοποίητα imports	1 περίπτωση	-
Έλλειψη σχολιασμού σε test μέθοδο	1 περίπτωση	1 περίπτωση
Χρήση γενικού wildcard τύπου	1 περίπτωση	1 περίπτωση

1.3 Επίδραση της αρχιτεκτονικής στην απόδοση του Claude Sonnet 4

Το Claude Sonnet 4 ανταποκρίθηκε θετικά στις απαιτήσεις και των δύο αρχιτεκτονικών, παρουσιάζοντας σταθερή και στοχευμένη προσέγγιση από τα αρχικά στάδια. Ιδιαίτερα στη μονολιθική υλοποίηση, επέδειξε σαφή κατανόηση των απαιτήσεων και παρήγαγε μια λειτουργική εφαρμογή με ορθές σχεδιαστικές επιλογές. Στην περίπτωση της αρχιτεκτονικής *microservices*, η δομή και η διασύνδεση μεταξύ των υπηρεσιών αντιμετωπίστηκαν με μεγαλύτερη ακρίβεια σε σχέση με άλλες περιπτώσεις, παρόλο που απαιτήθηκαν τεχνικές διορθώσεις για την εκτέλεση της εφαρμογής.

Κατανόηση αρχιτεκτονικής. Η μονολιθική προσέγγιση υλοποιήθηκε με ακρίβεια, ακολουθώντας την καθιερωμένη αρχιτεκτονική τριών επιπέδων. Στην περίπτωση των *microservices*, το Claude Sonnet δημιούργησε δύο ανεξάρτητες υπηρεσίες (*search-service* και *CLI client*) με διακριτούς ρόλους και αρμοδιότητες. Παρά το ότι η αρχιτεκτονική κατανόηση ήταν γενικά σωστή, η αρχική εκτέλεση απέτυχε λόγω ασυμβατότητας στην έκδοση του Spring Boot, που εντοπίστηκε και διορθώθηκε μετά από καθοδήγηση.



Εικόνα 4.11: Ανίχνευση σφάλματος συμβατότητας κατά την αρχική εκτέλεση της εφαρμογής, λόγω ασυμβατότητας έκδοσης Spring Boot. Το μήνυμα σφάλματος κοινοποιήθηκε στο Claude Sonnet και οδήγησε σε επιτυχή διόρθωση μέσω επαναληπτικής καθοδήγησης.

Οργάνωση και διαχωρισμός αρμοδιοτήτων. Η οργάνωση των κλάσεων στη μονολιθική υλοποίηση ήταν ξεκάθαρη, με σαφή διαχωρισμό μεταξύ *controller*, *service*, *CLI* και *DTOs*. Στη *microservices* εκδοχή, κάθε *service* ανέλαβε συγκεκριμένη ευθύνη, με σαφείς ροές δεδομένων και ορισμένες κλάσεις να επαναχρησιμοποιούνται κατάλληλα (όπως τα *DTOs*). Η προσέγγιση ήταν δομημένη και κοντά στις βέλτιστες πρακτικές.

Διαχείριση σφαλμάτων. Και στις δύο υλοποιήσεις εφαρμόστηκαν μηχανισμοί διαχείρισης σφαλμάτων, όπως *custom exceptions* και δομημένες αποκρίσεις (*ErrorRes-*

ponse', 'GlobalExceptionHandler'). Παρότι αρχικά υπήρξαν αστοχίες στη διαμόρφωση του 'application.yml', αυτές επιλύθηκαν με δύο διορθώσεις και δεν παρατηρήθηκε σημαντική υστέρηση σε αυτόν τον τομέα.

```
private void handleWebClientException(WebClientResponseException e) {
    if (e.getStatusCode() == HttpStatus.NOT_FOUND) {
        try {
            @SuppressWarnings("unchecked")
            Map<String, String> errorBody = e.getResponseBodyAs(Map.class);
            if (errorBody != null && errorBody.containsKey("message")) {
                System.out.println("❌ " + errorBody.get("message") + "\n");
            } else {
                System.out.println("❌ Η ταινία δεν βρέθηκε.\n");
            }
        } catch (Exception ex) {
            System.out.println("❌ Η ταινία δεν βρέθηκε.\n");
        }
    } else if (e.getStatusCode() == HttpStatus.SERVICE_UNAVAILABLE) {
        System.out.println("❌ Η υπηρεσία δεν είναι διαθέσιμη προς το παρόν. Παρακαλώ δοκιμάστε αργότερα.\n");
    } else {
        System.out.println("❌ Σφάλμα υπηρεσίας: " + e.getStatusCode() + "\n");
    }
}
```

Εικόνα 4.12: Παράδειγμα διαχείρισης εξαιρέσεων από το Claude Sonnet.

Πληρότητα υλοποίησης. Η μονολιθική εφαρμογή παρήχθη με πλήρη δομή και σχεδόν άμεση πληρότητα. Στη *microservices* έκδοχή δημιουργήθηκαν όλες οι απαραίτητες υπηρεσίες, ενώ υπήρξε ικανοποιητικός χειρισμός της ροής των δεδομένων. Οι επιμέρους λειτουργίες όπως η επικοινωνία με το IMDb API και η CLI διασύνδεση ενσωματώθηκαν επιτυχώς στο τελικό σύστημα.

Χρονική απόδοση και debugging. Η επίλυση των σφαλμάτων και στις δύο υλοποιήσεις (μονολιθική και *microservices*) ήταν σχετικά γρήγορη, καθώς απαιτήθηκαν μόλις 3 prompts σε κάθε περίπτωση. Η μονολιθική εφαρμογή ολοκληρώθηκε σε περίπου 1 ώρα, ενώ η *microservices* έκδοση χρειάστηκε ελαφρώς περισσότερο χρόνο (1 ώρα και 20 λεπτά) λόγω μιας ασυμβατότητας στην έκδοση του Spring Boot, η οποία απαιτήσε δοκιμή εναλλακτικών λύσεων. Παρ' όλα αυτά, το Claude Sonnet παρείχε στοχευμένες και αποτελεσματικές απαντήσεις, οδηγώντας σε επιτυχή εκτέλεση και των δύο εκδόσεων.

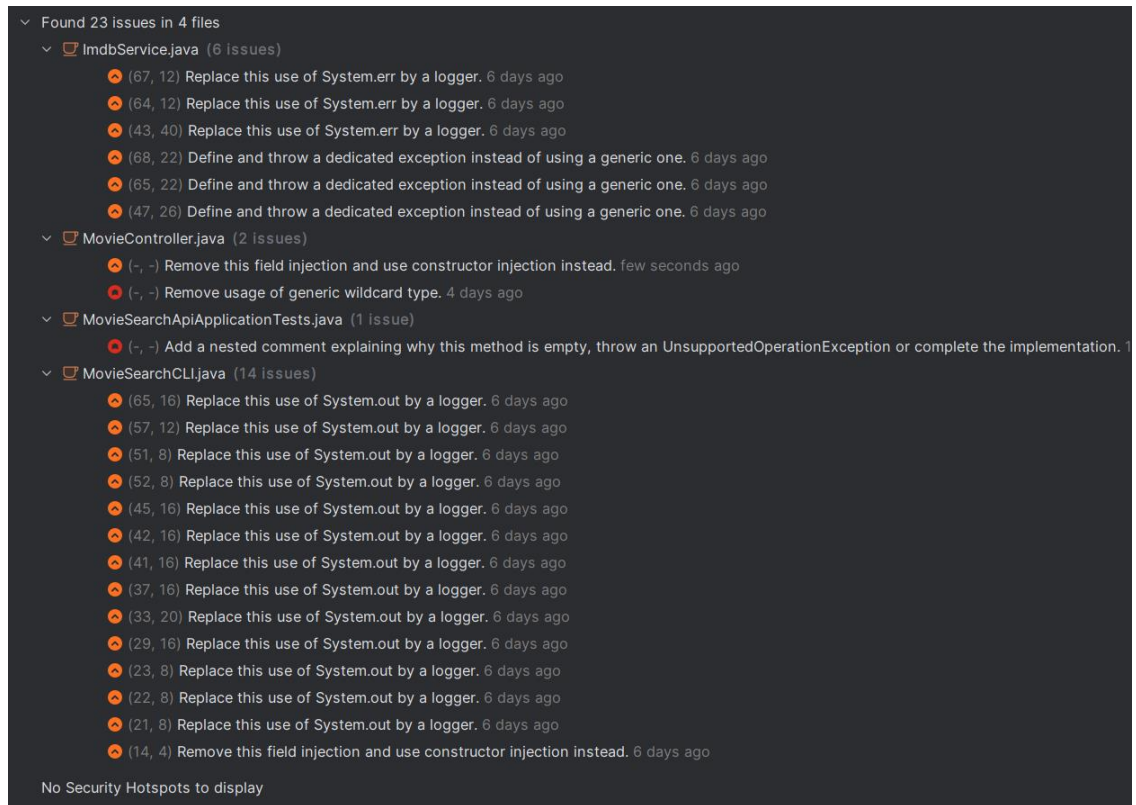
Πίνακας 4.5: Σύνοψη επίδοσης Claude Sonnet 4 ανά αρχιτεκτονική

Κριτήριο	Μονολιθική αρχιτεκτονική	Microservices αρχιτεκτονική
Κατανόηση αρχιτεκτονικής	Σαφής	Κατόπιν καθοδήγησης
Οργάνωση και διαχωρισμός αρμοδιοτήτων	Σαφής	Σαφής
Διαχείριση σφαλμάτων	Ακριβής	Επαρκής
Πληρότητα υλοποίησης	Πλήρης από τα αρχικά στάδια	Πλήρης κατόπιν καθοδήγησης
Χρονική απόδοση και debugging	Γρήγορη	Μέτρια

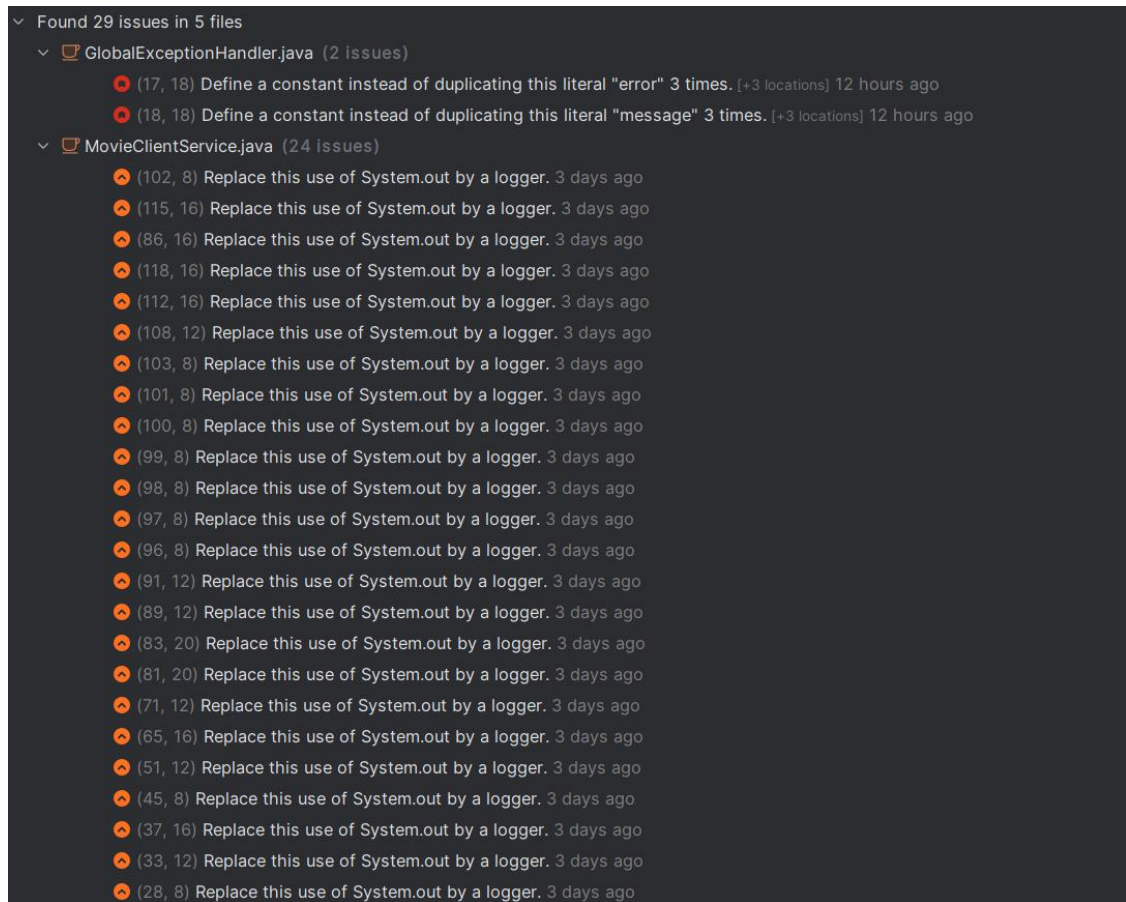
Στατική ανάλυση με SonarQube – Claude Η στατική ανάλυση με χρήση του SonarQube στην περίπτωση του Claude ανέδειξε διαφοροποιήσεις μεταξύ της μονολιθικής και της microservices αρχιτεκτονικής, τόσο σε πλήθος όσο και σε είδος ευρημάτων. Στη μονολιθική αρχιτεκτονική καταγράφηκαν 23 ζητήματα σε 4 αρχεία. Η πλειοψηφία των παρατηρήσεων αφορούσε την εκτεταμένη χρήση της System.out και System.err αντί για κατάλληλο μηχανισμό logging, γεγονός που θεωρείται αντιπαραγωγική πρακτική. Παράλληλα, επισημάνθηκαν αρκετές περιπτώσεις χρήσης γενικών εξαιρέσεων αντί για εξειδικευμένες, ενώ σημειώθηκαν και δομικές αδυναμίες, όπως η χρήση field injection αντί constructor injection, και η απουσία εξηγήσεων σε κενές test μεθόδους. Ένα επιπλέον ζήτημα που εντοπίστηκε ήταν η χρήση γενικού wildcard τύπου.

Αντιθέτως, η αρχιτεκτονική microservices του Claude παρουσίασε μεγαλύτερο αριθμό ευρημάτων: 29 ζητήματα σε 5 αρχεία. Τα περισσότερα προβλήματα εντοπίστηκαν στο MovieClientService.java, όπου η εκτεταμένη χρήση της System.out χωρίς logger κατέλαβε το μεγαλύτερο μέρος των ευρημάτων (24 περιπτώσεις). Παράλληλα, επισημάνθηκαν περιπτώσεις επαναλαμβανόμενων string literals που έπρεπε να αντικατασταθούν με σταθερές (constants), καθώς και η απουσία τεκμηρίωσης σε κενές μεθόδους δοκιμών. Αν και δεν παρατηρήθηκαν παραβιάσεις naming conventions ή ζητήματα τύπων, η υπερβολική χρήση System.out επιβαρύνει την αναγνωσιμότητα και τη συντηρησιμότητα του κώδικα.

Συμπερασματικά, αν και η μονολιθική υλοποίηση του Claude εμφάνισε ελαφρώς λιγότερα ζητήματα, η πλειονότητα αυτών ήταν δομικού και σχεδιαστικού χαρακτήρα. Από την άλλη, η υλοποίηση με microservices παρουσίασε περισσότερα αλλά πιο επιφανειακά προβλήματα, με έμφαση στη βελτιστοποίηση της εξόδου και της αναχρησιμοποίησης τιμών. Και οι δύο υλοποιήσεις παρουσιάζουν περιθώρια βελτίωσης, με έμφαση σε πρακτικές καθαρού και συντηρήσιμου κώδικα.



Εικόνα 4.13: Αποτελέσματα στατικής ανάλυσης με SonarQube για την μονολιθική υλοποίηση του Claude Sonnet 4



Εικόνα 4.14: Αποτελέσματα στατικής ανάλυσης με SonarQube για την microservices υλοποίηση του Claude Sonnet 4

Πίνακας 4.6: Σύγκριση στατικής ανάλυσης ποιότητας κώδικα (SonarQube) – Claude

Κριτήριο	Μονολιθική Αρχιτεκτονική	Microservices Αρχιτεκτονική
Αριθμός αρχείων με προβλήματα	4	5
Συνολικός αριθμός ζητημάτων	23	29
Χρήση <code>System.out/err</code> αντί για <code>Logger</code>	17 περιπτώσεις	24 περιπτώσεις
Χρήση γενικών εξαιρέσεων	3 περιπτώσεις	-
Constructor αντί για <code>field injection</code>	2 παρατηρήσεις για <code>field injection</code>	-
Απουσία τεκμηρίωσης σε <code>test</code> μέθοδο	1 περίπτωση	2 περιπτώσεις
Χρήση γενικού wildcard τύπου	1 περίπτωση	-
Επαναλαμβανόμενα <code>string literals</code> (χωρίς σταθερά)	-	3 περιπτώσεις

2. Συγκριτική ανάλυση της απόδοσης των LLMs

2.1 Σημεία σύγκλισης των μοντέλων

Παρά τις επιμέρους διαφοροποιήσεις στον τρόπο προσέγγισης, τα τρία μοντέλα (GPT-4, Gemini 2.5, Claude Sonnet 4) παρουσίασαν ορισμένα κοινά χαρακτηριστικά ως προς την κατανόηση και την υλοποίηση των απαιτήσεων της εκφώνησης.

Καταρχάς, και τα τρία μοντέλα αναγνώρισαν με επάρκεια τη βασική λειτουργικότητα της εφαρμογής και εφάρμοσαν σε μεγάλο βαθμό την τυπική αρχιτεκτονική τριών επιπέδων (`controller-service-model`) στη μονολιθική εκδοχή. Στη `microservices` προσέγγιση, αν και με διαφορετική ποιότητα και δομή, όλα τα μοντέλα κατάφεραν να υλοποιήσουν διακριτές υπηρεσίες με λειτουργική ροή δεδομένων.

Επιπλέον, και τα τρία μοντέλα προσαρμόστηκαν στην καθοδήγηση του χρήστη και ανταποκρίθηκαν με τροποποιήσεις στον αρχικό κώδικα. Εφάρμοσαν βασικές αρχές διαχείρισης εξαιρέσεων, χρησιμοποίησαν `DTOs` και ανέπτυξαν `RESTful endpoints`. Ακόμη και στον τομέα του `debugging`, όλα τα μοντέλα ακολούθησαν βήμα-βήμα ανατροφοδότηση για την επίλυση σφαλμάτων, προσαρμόζοντας ανάλογα τη λογική τους.

2..2 Σημεία διαφοροποίησης των μοντέλων

Παρά τη γενική σύγκλιση ως προς τη βασική δομή των υλοποιήσεων, υπήρξαν σημαντικές διαφοροποιήσεις που επηρεάζουν την ποιότητα και την αποτελεσματικότητα κάθε μοντέλου.

Το GPT-4 επέδειξε πολύ σταθερή και πληρέστατη υλοποίηση στη μονολιθική εκδοχή, με ελάχιστη ανάγκη καθοδήγησης και υψηλή αποδοτικότητα στο debugging. Αντίθετα, στη *microservices* αρχιτεκτονική, ήταν το μόνο που παρουσίασε αρχικά σύγχυση, καθώς παρήγαγε *modular* μονολιθικότητα, απαιτώντας επαναπροσδιορισμό της δομής.

Το Gemini 2.5 ξεκίνησε πιο αποσπασματικά και στις δύο περιπτώσεις, με αρχικές ελλείψεις στη δομή και στον διαχωρισμό ρόλων. Παρότι ενσωμάτωσε τελικά όλες τις απαιτούμενες λειτουργίες, χρειάστηκε επαναλαμβανόμενη καθοδήγηση για την κατανόηση της ροής δεδομένων και τη διαχείριση της JSON απόκρισης.

Το Claude Sonnet 4 παρουσίασε τη μεγαλύτερη συνοχή από τα αρχικά στάδια, κυρίως στη *microservices* εκδοχή. Παρήγαγε λειτουργικές υπηρεσίες με σαφή ροή, πέτυχε καλύτερη επαναχρησιμοποίηση κώδικα και είχε λιγότερες κρίσιμες παραβάσεις βάσει της στατικής ανάλυσης. Ωστόσο, εμφάνισε μικρές ασυμβατότητες στις ρυθμίσεις (π.χ. έκδοση Spring Boot), που αντιμετωπίστηκαν με καθοδήγηση.

Στον τομέα της στατικής ανάλυσης με SonarQube, το Claude εμφάνισε τα περισσότερα συνολικά ζητήματα, αλλά λιγότερης σοβαρότητας, σε σχέση με το Gemini που είχε ενδείξεις τεχνικού χρέους και γνωστικής πολυπλοκότητας. Το GPT-4 είχε την πιο ισορροπημένη στατική εικόνα, ειδικά στη μονολιθική του εκδοχή.

2..3 Σύγκριση Κόστους

Πέρα από τις δυνατότητες που προσφέρει ένα LLM, το κόστος αποτελεί έναν σημαντικό παράγοντα στην επιλογή του κατά την ανάπτυξη λογισμικού. Τα μοντέλα που χρησιμοποιήθηκαν στην παρούσα εργασία διαφέρουν ως προς το κόστος χρήσης τους, κάτι που πρέπει να ληφθεί υπόψη στη συγκριτική αξιολόγηση. Συγκεκριμένα:

- **GPT-4:** Το μοντέλο χρησιμοποιήθηκε μέσω της εφαρμογής ChatGPT, με κόστος 24 δολάρια ανά μήνα για απεριόριστη χρήση. Παρέχει επίσης δωρεάν πρόσβαση στο GPT-3.5, το οποίο όμως δεν επιλέχθηκε λόγω των περιορισμένων δυνατοτήτων του. Προσφέρεται και η δωρεάν χρήση του GPT-4 mini, ωστόσο ο περιορισμένος αριθμός *tokens* δεν επαρκούσε για τις ανάγκες της παρούσας μελέτης.
- **Gemini 2.5:** Χρησιμοποιήθηκε από την πλατφόρμα Google AI Studio, με δωρεάν πρόσβαση στο βασικό μοντέλο. Η Google προσφέρει και τη συνδρομητική έκδοση *Gemini Advanced* με κόστος 23 δολάρια ανά μήνα. Στην παρούσα

εργασία αξιοποιήθηκε αποκλειστικά η δωρεάν έκδοχή του μοντέλου.

- **Claude Sonnet 4.0:** Το μοντέλο αξιοποιήθηκε μέσω της υπηρεσίας *Amazon Bedrock* που διατίθεται στην πλατφόρμα AWS. Η χρέωση έγινε ανά token: 0,003 δολάρια για κάθε 1000 input tokens και 0,015 δολάρια για κάθε 1000 output tokens, με το συνολικό κόστος να ανέρχεται περίπου στα 15 δολάρια. Το Claude Sonnet είναι επίσης προσβάσιμο μέσω της πλατφόρμας της Anthropic με μηνιαία συνδρομή 18 δολαρίων ή ετήσια στα 90 δολάρια. Παρέχεται και περιορισμένη δωρεάν πρόσβαση, η οποία όμως δεν επαρκούσε για την έκταση της παρούσας εργασίας.

Πίνακας 4.7: Πραγματικό κόστος χρήσης των μοντέλων

Μοντέλο	Κόστος χρήσης
GPT-4 (ChatGPT Plus)	24 δολάρια (μηνιαία συνδρομή)
Gemini 2.5 (Google AI Studio)	0 δολάρια (δωρεάν έκδοση)
Claude Sonnet 4.0 (AWS Bedrock)	15 δολάρια (χρέωση ανά token)

Κεφάλαιο 5

Συμπεράσματα

1. Επίδραση Αρχιτεκτονικής στις Δυνατότητες των LLMs

Από την ανάλυση των αποτελεσμάτων προκύπτει ότι η μονολιθική αρχιτεκτονική διευκολύνει σημαντικά την κατανόηση και την παραγωγή κώδικα από τα LLMs. Και τα τρία μοντέλα παρήγαγαν δομές με καθαρό διαχωρισμό επιπέδων (controller, service, exception, DTO), ακολουθώντας το πρότυπο MVC. Οι περισσότερες αστοχίες εμφανίστηκαν σε λεπτομέρειες υλοποίησης και όχι σε ζητήματα αρχιτεκτονικής. Επιπλέον, η στατική ανάλυση με SonarQube στην μονολιθική εκδοχή εμφάνισε λιγότερα bugs και code smells, στοιχείο που ενισχύει την παρατήρηση πως η ενιαία δομή της εφαρμογής οδήγησε σε πιο συνεπή και ευανάγνωστο κώδικα.

Αντίθετα, στην microservices εκδοχή, τα LLMs εμφάνισαν μεγαλύτερη δυσκολία στο να κατανοήσουν τη διαμοιρασμένη ευθύνη μεταξύ υπηρεσιών. Συγκεκριμένα, εντοπίστηκαν προβλήματα τόσο στη ρύθμιση των routes επικοινωνίας, όσο και στη σωστή απομόνωση αρμοδιοτήτων. Ενδεικτικά, το GPT-4 δεν κατάλαβε εξαρχής ποια αρχιτεκτονική έπρεπε να ακολουθήσει, με αποτέλεσμα να απαιτηθεί επεξήγηση ώστε να μεταβεί από μονολιθική σε κατανεμημένη σχεδίαση. Παράλληλα, το Gemini φάνηκε να μην έχει ιδιαίτερα καλή αντίληψη της δομής που δημιουργεί, καθώς παρουσίασε συγκεχυμένο διαχωρισμό μεταξύ των services και των ρόλων τους, κάτι που οδήγησε σε ασαφή αρχιτεκτονική και απαίτησε περαιτέρω καθοδήγηση και αναδιαμόρφωση.

Η στατική ανάλυση του κώδικα έδειξε αυξημένη πολυπλοκότητα και περισσότερα σφάλματα στην εκδοχή των μικροϋπηρεσιών, με κύρια προβλήματα την ασυνέπεια στη χρήση εξαιρέσεων και την πλοκή των διασυνδέσεων μεταξύ των services. Παρά τις προκλήσεις, όλα τα μοντέλα κατάφεραν τελικά να παραγάγουν λειτουργικές microservices υλοποιήσεις, γεγονός που αποδεικνύει την ικανότητά τους να διαχειριστούν κατανεμημένες αρχιτεκτονικές, εφόσον λάβουν κατάλληλη καθοδήγηση. Ωστόσο, η διαφορά στην αρχική ποιότητα υλοποίησης σε σύγκριση με τη μονολιθική εκδοχή ήταν εμφανής, υποδεικνύοντας ότι η αυξανόμενη πολυπλοκότητα της αρχιτεκτονικής επηρεάζει τη συμπεριφορά και την αποδοτικότητα των

μοντέλων.

Τα παραπάνω ευρήματα υποδεικνύουν ότι η χρήση Μεγάλων Γλωσσικών Μοντέλων στην ανάπτυξη λογισμικού είναι πιο αποτελεσματική όταν πρόκειται για απλούστερες, ενιαίες αρχιτεκτονικές, όπως η μονολιθική. Στην πράξη, αυτό σημαίνει πως σε έργα με μικρότερο βαθμό πολυπλοκότητας, τα LLMs μπορούν να αξιοποιηθούν με ελάχιστη καθοδήγηση για τη γρήγορη παραγωγή λειτουργικού και δομημένου κώδικα. Στις αρχιτεκτονικές τύπου *microservices*, τα μοντέλα παρουσιάζουν δυσκολία κυρίως στη μεταξύ των υπηρεσιών επικοινωνία και στην οργάνωση του συστήματος ως ενιαίου συνόλου. Ωστόσο, στα επιμέρους *service modules*, αποδίδουν εξίσου καλά, επιδεικνύοντας ικανότητα στην παραγωγή επαναχρησιμοποιήσιμου και σαφώς διαχωρισμένου κώδικα. Η παρατήρηση αυτή ενισχύει την άποψη ότι τα LLMs αποτελούν ισχυρά εργαλεία υποστήριξης για τον προγραμματιστή, ειδικά όταν αξιοποιούνται με τη σωστή καθοδήγηση και σε περιβάλλοντα που απαιτούν υψηλό επίπεδο σχεδιαστικής πειθαρχίας. Αξιοσημείωτο είναι, τέλος, ότι σε όλες τις περιπτώσεις η διάρκεια υλοποίησης παρέμεινε εντός εύλογων χρονικών πλαισίων — από 1 έως 2 ώρες — γεγονός που καταδεικνύει τη δυναμική των LLMs ως εργαλεία επιτάχυνσης της ανάπτυξης, ακόμη και σε σύνθετα αρχιτεκτονικά περιβάλλοντα.

2. Σύγκριση των LLMs

Σε γενικό πλαίσιο, το Claude Sonnet ξεχώρισε ως το πιο συνεπές και αξιόπιστο μοντέλο, προσφέροντας ισορροπία μεταξύ ακρίβειας, πληρότητας και ταχύτητας απόκρισης. Παρ' όλα αυτά, το GPT-4 εμφάνισε τα καλύτερα αποτελέσματα στη στατική ανάλυση κώδικα με SonarQube, με λιγότερα bugs και code smells, γεγονός που υποδηλώνει υψηλή ποιότητα στην τελική παραγωγή. Ωστόσο, η αξιολόγηση των μοντέλων αποκτά ουσιαστικότερο νόημα όταν γίνεται διακριτά ανά αρχιτεκτονική, καθώς οι απαιτήσεις μεταξύ μονολιθικών και *microservices* υλοποιήσεων διαφέρουν σημαντικά.

Στη μονολιθική αρχιτεκτονική, τόσο το GPT-4 όσο και το Claude Sonnet παρουσίασαν πολύ καλή απόδοση, παράγοντας δομημένο, λειτουργικό κώδικα με ελάχιστες αστοχίες και με σαφή εφαρμογή του προτύπου MVC. Και τα δύο μοντέλα κατάφεραν να αντιληφθούν σωστά τη ζητούμενη αρχιτεκτονική από το αρχικό prompt, με το Claude να ξεχωρίζει για την πληρότητα της αρχικής υλοποίησης και το GPT-4 να υπερέχει ελαφρώς ως προς την ποιότητα κώδικα βάσει στατικής ανάλυσης. Αντίθετα, το Gemini φάνηκε να δυσκολεύεται περισσότερο, παρουσιάζοντας προβλήματα σε βασικά σημεία της υλοποίησης όπως η διαχείριση εξαιρέσεων και η ενσωμάτωση των εξαρτήσεων, με αποτέλεσμα να απαιτηθούν περισσότερες επεμβάσεις και καθοδήγηση για να ολοκληρωθεί η εφαρμογή. Παρ' όλα αυτά, κατάφερε τελικά να παραγάγει μια πλήρως λειτουργική μονολιθική εφαρμογή, έστω και με μεγαλύτερη προσπάθεια.

Στην περίπτωση της αρχιτεκτονικής *microservices*, το Claude Sonnet ήταν αυτό

που παρουσίασε την πιο σταθερή και αξιόπιστη απόδοση. Κατάφερε να οργανώσει αποτελεσματικά τις υπηρεσίες, να διαχωρίσει τους ρόλους με σαφήνεια και να ανταποκριθεί γρήγορα στα σφάλματα, επιδεικνύοντας ιδιαίτερη κατανόηση της πολυπλοκότητας της κατανεμημένης σχεδίασης. Το Gemini, αν και αρχικά εμφάνισε ασάφειες στη διασύνδεση των services και χρειάστηκε καθοδήγηση, ανταποκρίθηκε καλά στην επίλυση των τεχνικών προβλημάτων και παράγαγε ένα πλήρες αποτέλεσμα σε λογικό χρόνο. Αντίθετα, το GPT-4 παρουσίασε μεγαλύτερη δυσκολία στην κατανόηση της συνολικής δομής, καθυστερώντας να υιοθετήσει πλήρως το microservices μοντέλο, ενώ εμφάνισε και αδυναμίες στον σαφή διαχωρισμό των υπηρεσιών, γεγονός που το κατέταξε χαμηλότερα σε απόδοση για την αρχιτεκτονική αυτή.

Στην αρχιτεκτονική των microservices, το Claude Sonnet κατέγραψε την καλύτερη συνολική απόδοση, επιδεικνύοντας κατανόηση της λογικής του διαχωρισμού των υπηρεσιών και παράγοντας μια καθαρά δομημένη και λειτουργική υλοποίηση με σχετική ευκολία. Το Gemini ακολούθησε, εμφανίζοντας θετική ανταπόκριση στην απομόνωση των αρμοδιοτήτων ανά service και παρέχοντας επαρκή λύση, αν και με ορισμένες συγχύσεις ως προς την οργάνωση των modules και την επεξεργασία των δεδομένων. Το GPT-4 εμφάνισε τις μεγαλύτερες δυσκολίες, καθώς δεν κατάλαβε εξ αρχής την αρχιτεκτονική, οδηγώντας σε λανθασμένη αρχική προσέγγιση και ανάγκη για περαιτέρω διευκρινίσεις και καθοδήγηση. Παρά τις επιμέρους επιδόσεις, η στατική ανάλυση του κώδικα μέσω SonarQube ανέδειξε το GPT-4 ως το μοντέλο με τον πιο «καθαρό» και χαμηλής πολυπλοκότητας κώδικα, ακόμα και στη microservices εκδοχή, γεγονός που καταδεικνύει την τεχνική συνέπεια του μοντέλου ακόμη και όταν δεν κατανοεί πλήρως τη γενική αρχιτεκτονική.

Όσον αφορά το κόστος χρήσης των μοντέλων, το Gemini ξεχωρίζει ως η πιο προσιτή επιλογή, προσφέροντας αξιοπρεπείς επιδόσεις χωρίς οικονομική επιβάρυνση, κάτι που το καθιστά ιδιαίτερα ελκυστικό για προσωπικά ή ακαδημαϊκά έργα. Το Claude Sonnet, αν και είχε την καλύτερη συνολική απόδοση στην πλειονότητα των περιπτώσεων, βρίσκεται σε μέτριο εύρος κόστους, με συνδρομές που κρίνονται αποδεκτές για το επίπεδο υποστήριξης και ποιότητας που προσφέρει. Από την άλλη, το GPT-4 είναι η πιο δαπανηρή επιλογή, γεγονός που ενδέχεται να αποτελέσει περιοριστικό παράγοντα σε ευρύτερη κλίμακα. Ωστόσο, η υψηλή ποιότητα στατικής ανάλυσης και η συνέπεια στον παραγόμενο κώδικα επιβεβαιώνουν ότι το κόστος συνοδεύεται από αντίστοιχη αποδοτικότητα και τεχνική ακρίβεια.

Κάθε μοντέλο ανέδειξε διαφορετικά δυνατά σημεία κατά τη διάρκεια της μελέτης. Το Claude Sonnet ξεχώρισε για την ικανότητά του να κατανοεί πολύ καλά τις αρχιτεκτονικές απαιτήσεις, παρέχοντας συνεπείς, καθαρές και καλά οργανωμένες υλοποιήσεις, ιδιαίτερα σε περιβάλλοντα microservices. Το GPT-4, αν και πιο ακριβό, διακρίθηκε για τη σταθερότητα και την ποιότητα του παραγόμενου κώδικα, αποτυπώνοντας τις καλύτερες επιδόσεις στις στατικές αναλύσεις με SonarQube και επιδεικνύοντας αξιοπιστία στις λεπτομέρειες της υλοποίησης. Το Gemini, παρά το ότι εμφάνισε κάποια υστερήματα σε σύγκριση με τα άλλα δύο, παρουσίασε ικανοποιητική απόδοση τόσο σε μονολιθικά όσο και σε microservices περιβάλλοντα, με βασικό του πλεονέκτημα τη δωρεάν διαθεσιμότητα και την ταχύτητα απόκρισης,

που το καθιστούν μια επαρκή λύση για λιγότερο απαιτητικά σενάρια.

3. Πρακτική Χρησιμότητα των Ευρημάτων

Η πρακτική χρησιμότητα των αποτελεσμάτων αυτής της μελέτης έγκειται κυρίως στη δυνατότητα στρατηγικής ενσωμάτωσης των LLMs στην καθημερινή διαδικασία ανάπτυξης λογισμικού. Τα μοντέλα απέδειξαν ότι μπορούν να επιταχύνουν τη δημιουργία εφαρμογών, προσφέροντας πλήρως λειτουργικές υλοποιήσεις εντός περιορισμένου χρονικού πλαισίου (1–2 ώρες), γεγονός ιδιαίτερα σημαντικό σε εργασιακά ή εκπαιδευτικά περιβάλλοντα όπου η ταχύτητα υλοποίησης αποτελεί κρίσιμο παράγοντα. Σε μονολιθικές αρχιτεκτονικές, μπορούν να χρησιμοποιηθούν σχεδόν αυτούσια για παραγωγή αρχικού κώδικα ή scaffolding, ενώ σε microservices περιβάλλοντα —αν και απαιτούν καθοδήγηση— καταφέρνουν να ανταποκριθούν επαρκώς στα επιμέρους modules, προσφέροντας χρήσιμο κώδικα που μπορεί εύκολα να ενσωματωθεί στο σύστημα από έναν προγραμματιστή.

4. Προτάσεις για Μελλοντική Έρευνα

Η παρούσα μελέτη επικεντρώθηκε στη σύγκριση της απόδοσης των LLMs στην ανάπτυξη εφαρμογών REST API με μονολιθική και microservices αρχιτεκτονική, αξιοποιώντας ένα συγκεκριμένο σενάριο χρήσης. Στο μέλλον, θα ήταν ενδιαφέρον να επεκταθεί η αξιολόγηση σε πιο σύνθετες εφαρμογές, με πολλαπλές υπηρεσίες, βάσεις δεδομένων και μηχανισμούς ασφάλειας, ώστε να εξεταστεί η συμπεριφορά των LLMs σε πραγματικά περιβάλλοντα παραγωγής.

Επιπλέον, προτείνεται η διεύρυνση του φάσματος των μοντέλων που μελετώνται, εντάσσοντας ανοιχτού κώδικα λύσεις (όπως το StarCoder, Mistral, ή το LLaMA) για να εξεταστεί κατά πόσο είναι εφάμιλλες ή εάν υπολείπονται των εμπορικών μοντέλων ως προς την ακρίβεια και την ταχύτητα υλοποίησης.

Ιδιαίτερο ενδιαφέρον παρουσιάζει και η αξιολόγηση της μακροπρόθεσμης συντηρησιμότητας του κώδικα που παράγουν τα LLMs. Θα μπορούσε, για παράδειγμα, να εξεταστεί αν ο παραγόμενος κώδικας ενσωματώνει καλές πρακτικές επεκτασιμότητας, τεκμηρίωσης και modularity, μέσα από μια ανάλυση κατά τη διάρκεια εξέλιξης ενός έργου.

Τέλος, προτείνεται η εστίαση στην αλληλεπίδραση ανθρώπου–LLM σε συνεργατικό περιβάλλον. Μια μελλοντική μελέτη θα μπορούσε να διερευνήσει πώς αλλάζει η ποιότητα της υλοποίησης όταν οι προγραμματιστές συνεργάζονται με LLMs όχι μόνο ως “εργαλεία εντολών”, αλλά ως πραγματικούς συμπαραγωγούς κώδικα σε συνεχή ροή εργασίας.

Βιβλιογραφία

- [1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are Few-shot learners. *CoRR*, abs/2005.14165, 2020.
- [2] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2022.
- [3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé

- de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models trained on Code. *CoRR*, abs/2107.03374, 2021.
- [4] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. Evaluating Large Language Models in Class-Level Code Generation. New York, NY, USA, 2024. Association for Computing Machinery.
 - [5] Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi Li, and Yuchi Ma. Exploring and evaluating Hallucinations in LLM-powered Code Generation, 2024.
 - [6] Yue Gao, Xiaohong Hu, Yufeng Gao, Xin Xia, and Zhi Jin. The current challenges of Software Engineering in the era of Large Language Models. *arXiv preprint arXiv:2412.14554*, 2024.
 - [7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of Words and Phrases and their compositionality. In C. J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119. Curran Associates, Inc., 2013.
 - [8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2019.
 - [9] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. Technical report, OpenAI, 2018. Technical report.
 - [10] Xiang Chen, Xiaohong Hu, Yufeng Huang, et al. Deep learning-based Software Engineering: Progress, challenges, and opportunities. *Science China Information Sciences*, 68(11):111102, 2025.
 - [11] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on Large Language Models for Code Generation, 2024.
 - [12] OpenAI. GPT-4 technical report, 2023. Προσπελάστηκε στις 22 Απριλίου 2025.

- [13] Google DeepMind. Gemini 2.5 overview, 2024. Προσπελάστηκε στις 22 Απριλίου 2025.
- [14] Anthropic. Introducing the Claude 3 Model Family, 2024. Προσπελάστηκε στις 22 Απριλίου 2025.
- [15] Monolithic application vs Microservices Architecture. OpenLegacy Blog, 2024. Προσπελάστηκε στις 1 Μαΐου 2025.
- [16] Arjun Kamisetty, Deekshith Narsina, Marcus Rodriguez, Srinikhita Kothapalli, and Jaya Chandra Srikanth Gummadi. Microservices vs. Monoliths: Comparative analysis for scalable Software Architecture Design. *Engineering International*, 11(2):99–112, 2023.
- [17] 5 advantages of Microservices [+ disadvantages]. Atlassian, 2024. Προσπελάστηκε στις 1 Μαΐου 2025.
- [18] Microservices Architecture: Challenges and benefits. IBM Developer, 2023. Προσπελάστηκε στις 1 Μαΐου 2025.
- [19] Victor Velepucha and Pamela Flores. A survey on Microservices Architecture: Principles, patterns and migration challenges. *IEEE Access*, 11:88339–88358, 2023.
- [20] M. Mythily, A. Samson Arun Raj, and Iwin Thanakumar Joseph. An analysis of the significance of Spring Boot in the market. In *2022 International Conference on Inventive Computation Technologies (ICICT)*, pages 1277–1281, 2022.
- [21] Vinotech. Monolithic and Microservice Architectures in Spring Boot. Medium, 2024.
- [22] Matheus Felisberto. The trade-offs between Monolithic vs. Distributed Architectures, 2024.
- [23] Vincent Berry, Arnaud Castelltort, Benoit Lange, Joan Teriihoania, Chouki Tibermachine, and Catia Trubiani. Is it worth migrating a Monolith to Microservices? an experience report on Performance, Availability and Energy Usage. In *2024 IEEE International Conference on Web Services (ICWS)*, pages 944–954, 2024.
- [24] Khaled Sellami and Mohamed Aymen Saied. Contrastive learning-enhanced Large Language Models for Monolith-to-Microservice Decomposition, 2025.
- [25] Ansong Ni, Pengcheng Yin, Yilun Zhao, Martin Riddell, Troy Feng, Rui Shen, Stephen Yin, Ye Liu, Semih Yavuz, Caiming Xiong, Shafiq Joty, Yingbo Zhou, Dragomir Radev, and Arman Cohan. L2ceval: Evaluating language-to-code generation capabilities of large language models, 2023.
- [26] Li Zhong and Zilong Wang. Can LLM replace Stack Overflow? a study on robustness and reliability of Large Language Model Code Generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(19):21841–21849, 2024.

Παράρτημα Α΄

Υλικό

Το σύνολο του υλικού της πτυχιακής εργασίας είναι διαθέσιμο στο ακόλουθο αποθετήριο GitHub:

- **URL:** https://github.com/Sintivrounai/t8210017_thesis

Το αποθετήριο περιλαμβάνει τους εξής φακέλους:

- **Claude-Sonnet:** Τελικές υλοποιήσεις εφαρμογών που παρήχθησαν από το Claude Sonnet (μονολιθικές και microservices).
- **Claude-Monolith-Prompt-History & Claude-Microservices-Prompt-History:** Ιστορικό διαλόγων και εντολών για τις αντίστοιχες αρχιτεκτονικές του Claude Sonnet.
- **GPT-4:** Υλοποιήσεις του GPT-4.
- **GPT-Monolith-Prompt-History & GPT-Microservices-Prompt-History:** Αναλυτικό prompt history του GPT-4 για κάθε αρχιτεκτονική.
- **Gemini:** Υλοποιήσεις του μοντέλου Gemini.
- **Gemini-Monolith-Prompt-History & Gemini-Microservices-Prompt-History:** Καταγεγραμμένη επικοινωνία με το Gemini.
- **SonarQube_Results:** Περιλαμβάνει στιγμιότυπα και αρχεία με τα αποτελέσματα στατικής ανάλυσης κώδικα (bugs, code smells κ.ά.).
- **latex:** Αρχεία σύνταξης της πτυχιακής εργασίας σε μορφή LaTeX.



ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

ΜΕΡΟΣ Β

Αναφορά πρακτικής άσκησης

Επιβλέπων:

Μαριλένα Τασιούλα

Αθήνα, Ιούλιος 2025



Με τη συγχρηματοδότηση
της Ευρωπαϊκής Ένωσης



Πρόγραμμα
Ανθρώπινο Δυναμικό και
Κοινωνική Συνοχή

Περιεχόμενα

1	Εισαγωγή	1
2	Βασικά Χαρακτηριστικά πρακτικής	2
1.	Περιγραφή τμήματος	2
2.	Οργανωτική Δομή	2
3.	Βασικές Διαδικασίες	2
4.	Περιγραφή θέσης	3
5.	Δεξιότητες που απαιτήθηκαν	3
6.	Αναμενόμενα Αποτελέσματα του έργου μου	4
3	Έργα/Δραστηριότητες	4
4	Αποτελέσματα	5
1.	Go beyond payments	6
2.	Αλλάζω σύστημα θέρμανσης και θερμοσίφωνα	6
5	Ενημερωμένος Χρονικός Προγραμματισμός του Έργου	7
6	Δεξιότητες	9
7	Σχόλια-Παρατηρήσεις	10
1.	Εργασιακό περιβάλλον	10
2.	Λειτουργία του τμήματος	11
3.	Παράγοντες που αποτέλεσαν εμπόδιο στην πρακτική μου	12
4.	Παράγοντες που διευκόλυναν την πρακτική μου	12
5.	Γνώσεις και εμπειρία που αποκτήθηκε	13
6.	Προτάσεις για βελτίωση του τμήματος	14

1. Εισαγωγή

Η Grant Thornton International Ltd είναι μία από τις κορυφαίες οργανώσεις παροχής ελεγκτικών, φορολογικών και συμβουλευτικών υπηρεσιών παγκοσμίως, με παρουσία σε περισσότερες από 140 χώρες και με πάνω από 60.000 επαγγελματίες. Ιδρύθηκε το 1924 και από τότε έχει αναπτυχθεί σε έναν παγκόσμιο οργανισμό που προσφέρει υπηρεσίες υψηλής προστιθέμενης αξίας σε επιχειρήσεις όλων των μεγεθών — από μικρομεσαίες επιχειρήσεις έως και πολυεθνικές. Οι βασικοί τομείς στους οποίους δραστηριοποιείται περιλαμβάνουν:

- ▶ Ελεγκτικές υπηρεσίες (*Audit & Assurance*)
- ▶ Φορολογικές υπηρεσίες (*Tax*)
- ▶ Συμβουλευτικές υπηρεσίες (*Advisory*)
- ▶ *Risk Management*
- ▶ *Business Consulting & Digital Transformation*

Η φιλοσοφία της Grant Thornton βασίζεται στην προσωποποιημένη εξυπηρέτηση, τη διαφάνεια, την τεχνογνωσία, και τη συνεχή καινοτομία με στόχο να ενισχύσει τη βιώσιμη ανάπτυξη των πελατών της σε ένα μεταβαλλόμενο παγκόσμιο επιχειρηματικό περιβάλλον.

Η Grant Thornton Ελλάδας αποτελεί ένα από τα πλέον καταξιωμένα γραφεία του διεθνούς δικτύου Grant Thornton και δραστηριοποιείται στη χώρα για περισσότερα από 35 χρόνια. Διαθέτει γραφεία στην Αθήνα, τη Θεσσαλονίκη, την Πάτρα και την Ηράκλειο Κρήτης, απασχολώντας πάνω από 1.000 εργαζόμενους.

2. Βασικά Χαρακτηριστικά πρακτικής

1. Περιγραφή τμήματος

Στα πλαίσια του μαθήματος «Πρακτική Άσκηση - Πτυχιακή Εργασία», εκπόνησα την πρακτική μου στο τμήμα του Technology ως Junior Backend Engineer. ο τμήμα Technology αποτελεί έναν δυναμικά αναπτυσσόμενο τομέα που εντάσσεται στο ευρύτερο τμήμα Advisory και ειδικεύεται στην παροχή τεχνολογικών λύσεων και καινοτόμων υπηρεσιών λογισμικού σε πελάτες του ιδιωτικού και του δημόσιου τομέα. Στόχος του είναι να υποστηρίζει τις επιχειρήσεις στη διαδικασία του ψηφιακού μετασχηματισμού, παρέχοντας αξιόπιστα και σύγχρονα πληροφοριακά συστήματα προσαρμοσμένα στις ανάγκες κάθε πελάτη.

2. Οργανωτική Δομή

Το τμήμα Technology οργανώνεται σε επιμέρους ομάδες ανάλογα με την ειδικότητα και τα έργα. Οι βασικοί ρόλοι περιλαμβάνουν:

- ▶ Backend Engineers
- ▶ Frontend Engineers
- ▶ QA Engineers
- ▶ UI/UX Designers
- ▶ DevOps Engineers
- ▶ Data Engineers
- ▶ Project Managers
- ▶ Business Analysts

Η επικοινωνία μεταξύ των ομάδων είναι συνεχής, με έμφαση στη συνεργατικότητα και την agile προσέγγιση μέσω Scrum ή Kanban.

3. Βασικές Διαδικασίες

Οι κύριες διαδικασίες του τμήματος περιλαμβάνουν:

- ▶ Ανάλυση απαιτήσεων πελάτη (*requirements gathering*)
- ▶ Σχεδίαση και ανάπτυξη backend συστημάτων
- ▶ Δημιουργία *RESTful APIs*
- ▶ Έλεγχος ποιότητας (testing, code reviews)
- ▶ Συνεχής ολοκλήρωση και ανάπτυξη (*CI/CD*)
- ▶ Υποστήριξη και συντήρηση των εφαρμογών

4. Περιγραφή θέσης

Κατά τη διάρκεια της πρακτικής μου, εργάστηκα στη θέση του *Junior Backend Engineer*. Ήμουν μέλος μιας ομάδας ανάπτυξης που είχε ως ευθύνη την υλοποίηση και υποστήριξη *backend* υπηρεσιών για *web* εφαρμογές. Ειδικότερα, συμμετείχα σε εργασίες όπως:

- ▶ Σχεδίαση και ανάπτυξη *endpoints* σε Java με Spring Boot
- ▶ Διασύνδεση με βάσεις δεδομένων (κυρίως PostgreSQL)
- ▶ Δημιουργία και κατανάλωση *REST APIs*
- ▶ Συμμετοχή σε συναντήσεις *Sprint Planning* και καθημερινά *Standups*
- ▶ Έλεγχος κώδικα με χρήση εργαλείων όπως *SonarQube* και *JUnit*

5. Δεξιότητες που απαιτήθηκαν

Για την επιτυχή υλοποίηση των καθηκόντων μου, απαιτήθηκε συνδυασμός τεχνικών και οριζόντιων δεξιοτήτων, όπως:

- ▶ Πολύ καλή γνώση Java και Spring Boot
- ▶ Κατανόηση βασικών αρχών σχεδίασης λογισμικού (OOP, MVC, REST)
- ▶ Εξοικείωση με βάσεις δεδομένων (SQL)
- ▶ Εξοικείωση με τη χρήση Docker
- ▶ Git για *version control* και συνεργασία
- ▶ Αναλυτική σκέψη και επίλυση προβλημάτων
- ▶ Εξοικείωση με εργαλεία διαχείρισης έργου όπως το Jira για παρακολούθηση προόδου και οργάνωση εργασιών σε *agile* περιβάλλον

6. Αναμενόμενα Αποτελέσματα του έργου μου

Η πρακτική μου άσκηση ως *Junior Backend Engineer* είχε ως στόχο τη συμμετοχή μου στην ανάπτυξη και υλοποίηση λειτουργικών, αποδοτικών και επεκτάσιμων *backend* λύσεων για *web* εφαρμογές. Τα αναμενόμενα αποτελέσματα της συμμετοχής μου στο έργο περιλάμβαναν:

- ▶ Ανάπτυξη λειτουργικών *endpoints* με χρήση *Java* και *Spring Boot*, σύμφωνα με τις ανάγκες των πελατών και τις αρχές του καθαρού και επαναχρησιμοποιήσιμου κώδικα.
- ▶ Δημιουργία και κατανάλωση *RESTful APIs* για την υποστήριξη της επικοινωνίας μεταξύ *backend* και *frontend* συστημάτων.
- ▶ Διασύνδεση με βάσεις δεδομένων *PostgreSQL* και διαχείριση των μεταναστεύσεων (*schema migrations*) μέσω *Liquibase* για την ορθή εξέλιξη της βάσης δεδομένων κατά τη διάρκεια της ανάπτυξης.
- ▶ Συμμετοχή σε όλες τις φάσεις των *agile* διαδικασιών, με ενεργό ρόλο σε *Sprint Planning* και *Daily Standups*.
- ▶ Συμβολή στη διασφάλιση ποιότητας του κώδικα, μέσω στατικών ελέγχων με *SonarQube* και συγγραφής μονάδων δοκιμών (*unit tests*) με *JUnit*, προκειμένου να εξασφαλιστεί η αξιοπιστία και η σταθερότητα του λογισμικού.
- ▶ Εξοικείωση με σύγχρονες τεχνολογίες *DevOps*, όπως *Docker* και εργαλεία συνεχούς ολοκλήρωσης και ανάπτυξης (*CI/CD pipelines*).
- ▶ Αξιοποίηση του *Git* και του *Jira* για την παρακολούθηση της προόδου, την οργάνωση εργασιών και τη συνεργασία στο πλαίσιο *agile* μεθοδολογιών.

Συνολικά, τα αναμενόμενα αποτελέσματα αφορούσαν τόσο την τεχνική πρόοδο του έργου όσο και την απόκτηση ουσιαστικής επαγγελματικής εμπειρίας μέσα από τη συμμετοχή μου σε ένα σύγχρονο και απαιτητικό τεχνολογικό περιβάλλον.

3. Έργα/Δραστηριότητες

Κατά τη διάρκεια της συνέντευξης, είχα ενημερωθεί πως θα ενταχθώ αρχικά στην ευρύτερη ομάδα *backend* κατά τη φάση της εκπαίδευσης, και στη συνέχεια θα τοποθετούμουν σε συγκεκριμένο έργο. Τα βασικά στάδια της πρακτικής μου ήταν τα εξής:

- **Παρακολούθηση εκπαιδευτικού υλικού:** Στην αρχή της πρακτικής, το τμήμα ανθρώπινου δυναμικού (HR) μου παρείχε αναλυτική ενημέρωση για τη λειτουργία της εταιρείας, τα δικαιώματα και τις υποχρεώσεις μου, τα βασικά συστήματα καθώς και απαντήσεις σε απορίες. Στη συνέχεια, παρακολούθησα υποχρεωτικά εκπαιδευτικά προγράμματα που μου έδωσαν συνολική εικόνα της εταιρικής δομής και των πολιτικών της.
- **Ένταξη στην ομάδα του backend:** Μετά την ολοκλήρωση της αρχικής εκπαίδευσης, ξεκίνησα τεχνική κατάρτιση στο Spring Boot, μέσω ενός επιμορφωτικού προγράμματος. Στη συνέχεια, ανέπτυξα μια demo εφαρμογή με βασικές λειτουργίες όπως δημιουργία *endpoints*, κλήσεις *REST APIs*, διαχείριση βάσης δεδομένων και μηχανισμό *authentication* χρηστών.
- **Εξοικείωση με τη βασική βιβλιοθήκη Java:** Η ομάδα backend χρησιμοποιεί μια εσωτερική βιβλιοθήκη ως βάση για τα περισσότερα έργα. Μελετήθηκε η δομή της και στη συνέχεια μου ανατέθηκαν εργασίες όπως διόρθωση σφαλμάτων και προσθήκη λειτουργιών.
- **Ρύθμιση περιβάλλοντος εργασίας:** Με την υποστήριξη της ομάδας, προχώρησα στη ρύθμιση του τοπικού περιβάλλοντος ανάπτυξης με τεχνολογίες όπως *Java*, *Spring Boot*, *PostgreSQL*, *pgAdmin*, *Docker*, καθώς και χρήση *Git*. Εξασφαλίστηκε η ομαλή διασύνδεση με τις βάσεις και τα απαραίτητα *services*, ώστε να μπορώ να συμμετάσχω χωρίς καθυστερήσεις.
- **Συμμετοχή στο έργο *Go Beyond Payments* του gov.gr:** Το έργο βρισκόταν σε παραγωγή και επικεντρωνόταν στη συντήρηση. Μελέτησα τη ροή και την αρχιτεκτονική του συστήματος και πραγματοποίησα στατική ανάλυση με *SonarQube*, προτείνοντας βελτιώσεις για την ποιότητα του κώδικα.
- **Συμμετοχή στο έργο *Αλλάζω σύστημα θέρμανσης και θερμοσίφωνα*:** Το έργο βρισκόταν σε ενεργή ανάπτυξη και συμμετείχα σε δύο *sprints*. Υλοποίησα API κλήσεις και εργάστηκα με *Liquibase* για τις μεταναστεύσεις της βάσης δεδομένων.

4. Αποτελέσματα

Η εταιρεία στην οποία πραγματοποίησα την πρακτική μου άσκηση, διαθέτει ένα ευρύ φάσμα δραστηριοτήτων στον τομέα της ανάπτυξης λογισμικού και συμμετέχει σε πληθώρα έργων του δημοσίου και ιδιωτικού τομέα. Για τον λόγο αυτό, η ομάδα Technology είναι οργανωμένη σε διακριτές ομάδες έργου, ώστε κάθε ομάδα να επικεντρώνεται αποκλειστικά σε ένα συγκεκριμένο σύστημα ή πλατφόρμα, διασφαλίζοντας έτσι τη στοχευμένη ενασχόληση, τη συνεχή παρακολούθηση και την

ποιοτική υλοποίηση των απαιτήσεων κάθε έργου. Μέσα σε αυτό το πλαίσιο, κατά την πορεία της πρακτικής μου, εντάχθηκα σε δύο διαφορετικά έργα —το Go Beyond Payments και το «Αλλάζω σύστημα θέρμανσης και θερμοσίφωνα»— αναλαμβάνοντας συγκεκριμένες αρμοδιότητες στο backend σκέλος της ανάπτυξης, ανάλογα με τη φάση στην οποία βρισκόταν κάθε έργο.

1. Go beyond payments

Κατά τη διάρκεια της πρακτικής μου, ένα από τα βασικότερα έργα στα οποία συμμετείχα ήταν το Go Beyond Payments του gov.gr, το οποίο αποτέλεσε και το πρώτο μεγάλο έργο στο οποίο εργάστηκα ενεργά. Το συγκεκριμένο έργο ήταν ήδη σε φάση παραγωγής κατά την ένταξή μου στην ομάδα, και η κύρια δραστηριότητα της ομάδας επικεντρωνόταν στη συντήρηση, βελτιστοποίηση και διασφάλιση ποιότητας του backend κώδικα. Ο ρόλος μου αφορούσε κυρίως τη στατική ανάλυση του backend συστήματος με χρήση του εργαλείου SonarQube, με σκοπό την εντόπιση προβλημάτων όπως code smells, τεχνικό χρέος και δυνητικά μη ασφαλείς υλοποιήσεις. Η ανάλυση πραγματοποιήθηκε σε συνδυασμό με μελέτη της αρχιτεκτονικής και της δομής της εφαρμογής, προκειμένου να κατανοήσω τη ροή και τις βασικές συνιστώσες του συστήματος. Βασίστηκα σε αρχές καθαρού και επεκτάσιμου κώδικα (Clean Code και SOLID) και προχώρησα σε διορθώσεις και προτάσεις για βελτίωση του υπάρχοντος κώδικα, τις οποίες υπέβαλα μέσω Pull Requests. Κατά τη διάρκεια της δραστηριότητας αυτής, εργάστηκα περίπου 60 ώρες σε διάστημα δύο εβδομάδων, χωρίς να υπάρξουν καθυστερήσεις σε σχέση με τον αρχικό προγραμματισμό. Η βασική δυσκολία που αντιμετώπισα ήταν η εξοικείωση με μια μεγάλη και σύνθετη βάση κώδικα, δεδομένου ότι δεν είχα προηγούμενη εμπειρία με παραγωγικά έργα τέτοιας κλίμακας. Με την καθοδήγηση των πιο έμπειρων μελών της ομάδας και τη μεθοδική μελέτη του έργου, κατάφερα να ενταχθώ πλήρως και να συμβάλλω ουσιαστικά. Η δραστηριότητά μου είχε θετικό αντίκτυπο στη λειτουργία της επιχείρησης, καθώς βοήθησε στη μείωση του τεχνικού χρέους και στη βελτίωση της ποιότητας και συντηρησιμότητας του συστήματος, διευκολύνοντας την ομάδα στην προετοιμασία για μελλοντικά releases και επεκτάσεις της εφαρμογής.

2. Αλλάζω σύστημα θέρμανσης και θερμοσίφωνα

Το δεύτερο έργο στο οποίο συμμετείχα κατά τη διάρκεια της πρακτικής μου ήταν το «Αλλάζω σύστημα θέρμανσης και θερμοσίφωνα» του gov.gr, το οποίο βρισκόταν σε φάση ενεργής ανάπτυξης όσο ήμουν ενταγμένος στην ομάδα. Το έργο αυτό διεκπεραιωνόταν με βάση agile μεθοδολογία, μέσα από οργανωμένα sprints, και κατά τη διάρκειά μου σε αυτό πραγματοποιήθηκαν δύο κύκλοι ανάπτυξης που αφορούσαν τα επικείμενα releases της πλατφόρμας. Η συμβολή μου επικεντρώθηκε στην υλοποίηση συγκεκριμένων backend λειτουργιών, κυρίως μέσω κλήσεων REST

APIs, καθώς και στη διαχείριση μεταναστεύσεων της βάσης δεδομένων με χρήση του εργαλείου Liquibase, το οποίο χρησιμοποίησα για πρώτη φορά στο πλαίσιο αυτής της δραστηριότητας. Παρά την αρχική δυσκολία κατανόησης του εργαλείου και της λογικής του versioning των βάσεων δεδομένων, με την καθοδήγηση της ομάδας και μέσω hands-on παραδειγμάτων, κατάφερα να εξοικειωθώ και να παράδωσω λειτουργικά migrations, τα οποία ενσωματώθηκαν επιτυχώς στο κεντρικό αποθετήριο. Παράλληλα, συνέβαλα σε απλές μονάδες ελέγχου (unit tests) χρησιμοποιώντας JUnit, διασφαλίζοντας τη σταθερότητα των αλλαγών που υλοποιούσα. Συνολικά, εργάστηκα για περίπου 65 ώρες στο πλαίσιο του συγκεκριμένου έργου, τηρώντας πλήρως το χρονικό πλάνο των sprints χωρίς να παρουσιαστούν καθυστερήσεις. Η δραστηριότητά μου συνέβαλε στην ομαλή και έγκαιρη παράδοση λειτουργιών του backend, απαραίτητων για τη δημόσια διάθεση της εφαρμογής, ενώ ενίσχυσε τη σταθερότητα και την αξιοπιστία της βάσης δεδομένων μέσω ορθά δομημένων schema migrations. Το βασικότερο πρόβλημα που αντιμετώπισα ήταν η αρχική απειρία σε εργαλεία όπως το Liquibase, το οποίο όμως κατάφερα να μάθω και να εφαρμόσω αποτελεσματικά σε πραγματικό περιβάλλον, γεγονός που βελτίωσε σημαντικά τόσο τις τεχνικές μου δεξιότητες όσο και την αυτοπεποίθησή μου ως backend developer.

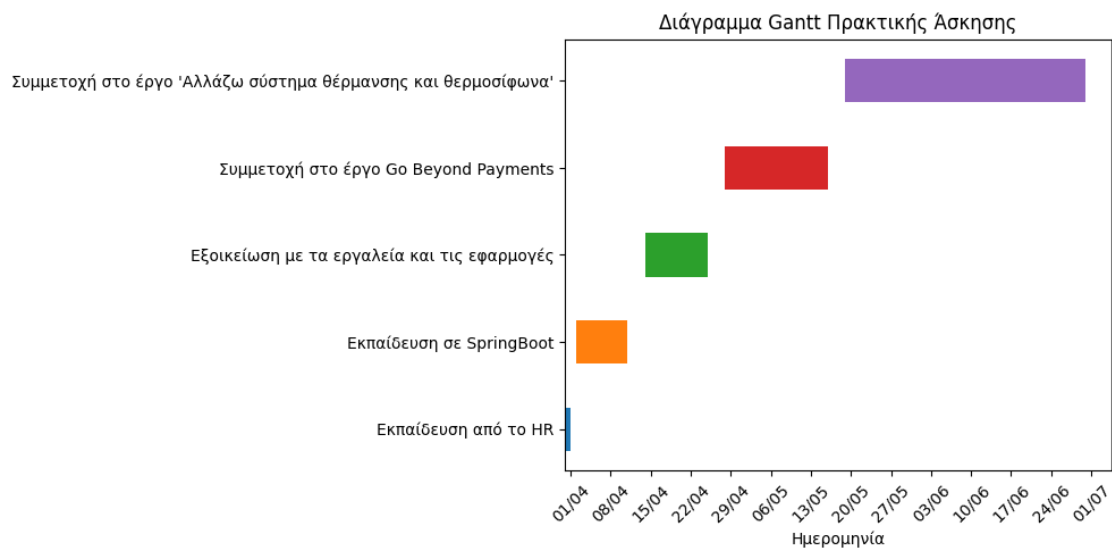
5. Ενημερωμένος Χρονικός Προγραμματισμός του Έργου

Παρακάτω, παρουσιάζεται ένας αναλυτικός πίνακας με τον αναλυτικό χρονικό ορίζοντα της πρακτικής και συγκεκριμένα με τι απασχολήθηκα ανά περιόδους. Η ημερομηνία έναρξης και λήξης επηρεάζονται από κάποια σαββατοκύριακα, αργίες και άδειες, ενώ η τελευταία στήλη δείχνει την πραγματική διάρκεια της κάθε δραστηριότητας. Είναι σημαντικό να αναφερθεί ότι οι δραστηριότητες δεν ήταν πάντα διακριτές. Για παράδειγμα, κατά την διάρκεια της συμμετοχής μου στο έργο των πληρωμών, μου ανατέθηκε να κάνω μια διόρθωση στην βασική βιβλιοθήκη. Ωστόσο, ο πίνακας είναι αρκετά έγκυρος καθώς λαμβάνει υπόψη του που απασχολήθηκα κατά βάση ανά περίοδο.

Δραστηριότητα	Ημερομηνία Έναρξης	Ημερομηνία Λήξης	Διάρκεια
Εκπαίδευση από το HR	31/3	1/4	2 ημέρες
Εκπαίδευση σε SpringBoot	2/4	4/4	3 ημέρες
Ανάπτυξη demo εφαρμογής σε SpringBoot	7/4	11/4	5 ημέρες
Εξοικείωση με τα κανάλια επικοινωνίας (Gitlab, Jira) και απόκτηση πρόσβασης στις εφαρμογές	14/4	17/4	3 ημέρες
Παρουσίαση από την ομάδα της βασικής βιβλιοθήκης που θα χρησιμοποιήσω	22/4	23/4	2 ημέρες
Ρύθμιση τοπικού περιβάλλοντος ανάπτυξης	24/4	25/4	2 ημέρες
Ένταξη στο Go Beyond Payments και εξοικείωση με το έργο	28/4	2/5	4 ημέρες
Στατική ανάλυση του Go Beyond Payments	2/5	16/5	11 ημέρες
Αλλαγή project και εξοικείωση με το «Αλλάζω σύστημα θέρμανσης και θερμοσίφωνα»	19/5	30/5	9 ημέρες
Ενεργός συμμετοχή στην ανάπτυξη του project	10/6	30/6	18 ημέρες

Πίνακας 5.1: Χρονολογική απεικόνιση δραστηριοτήτων κατά την πρακτική άσκηση

Η παρακάτω εικόνα παρουσιάζει σε μορφή διαγράμματος Gantt τη χρονολογική εξέλιξη των βασικών δραστηριοτήτων που υλοποιήθηκαν κατά τη διάρκεια της πρακτικής άσκησης. Κάθε γραμμή αντιστοιχεί σε μια συγκεκριμένη δραστηριότητα ή έργο, ενώ το μήκος της μπάρας δείχνει τη διάρκεια της υλοποίησής της στο αντίστοιχο χρονικό διάστημα.



Εικόνα 5.1: Διάγραμμα Gantt που παρουσιάζει τη χρονολογική απεικόνιση των βασικών δραστηριοτήτων κατά τη διάρκεια της πρακτικής άσκησης.

6. Δεξιότητες

Κατά τη διάρκεια της Πρακτικής Άσκησης, είχα την ευκαιρία να εφαρμόσω στην πράξη πολλές από τις γνώσεις και δεξιότητες που απέκτησα στο πλαίσιο του προγράμματος σπουδών του Τμήματος Διοικητικής Επιστήμης και Τεχνολογίας. Μέσα από την ενασχόλησή μου με πραγματικά έργα, ανέπτυξα τεχνικές και οργανωτικές ικανότητες, όπως η ανάπτυξη λογισμικού, ο σχεδιασμός βάσεων δεδομένων, η χρήση εργαλείων testing και στατικής ανάλυσης, η συνεργασία σε agile περιβάλλον και η διαχείριση εκδόσεων κώδικα. Ο παρακάτω πίνακας συνοψίζει ενδεικτικά τις δεξιότητες που αξιοποιήθηκαν και ενισχύθηκαν κατά την πρακτική μου εμπειρία.

Δεξιότητα	Σχετικές μέθοδοι	Παράδειγμα
Ανάπτυξη και συντήρηση λογισμικού	Java, Maven, Docker	Ανάπτυξη demo εφαρμογής και υλοποίηση backend λειτουργιών σε υπάρχοντα έργα με Java και Maven, χρήση Docker για εκτέλεση τοπικού περιβάλλοντος

Σχεδιασμός και ανάπτυξη βάσεων δεδομένων	MySQL, Postgres, Liquibase	Δημιουργία και διαχείριση βάσεων δεδομένων σε Postgres και υλοποίηση database migrations με Liquibase
Διαχείριση σχηματισμών λογισμικού	Git, Gitlab, Azure	Διαχείριση εκδόσεων μέσω Git και GitLab, δημιουργία branch, merge requests και χρήση Azure DevOps για κεντρική αποθήκευση κώδικα και συνεργασία
Εργαλεία στατικής ανάλυσης κώδικα	SonarQube, Checkstyle, SpotBugs	Στατική ανάλυση Java κώδικα στο έργο Go Beyond Payments με χρήση SonarQube και SpotBugs, εντοπισμός και βελτίωση code smells και violations coding standards
Δοκιμή λογισμικού (Testing)	JUnit, Mockito, Postman	Ανάπτυξη μονάδων ελέγχου με JUnit, χρήση Mockito για mocking και έλεγχος REST endpoints μέσω Postman
Χρονοπρογραμματισμός και διαχείριση εργασιών	Jira, Agile (Scrum/Kanban)	Συμμετοχή σε agile κύκλους ανάπτυξης με χρήση Jira για παρακολούθηση των εργασιών, συμμετοχή σε Sprint Planning και Daily Standups
Κατανόηση αρχών σχεδίασης λογισμικού	OOP, MVC, REST	Ανάπτυξη backend endpoints με Java Spring Boot ακολουθώντας αρχές MVC και RESTful αρχιτεκτονικής
Συνεχής ολοκλήρωση (CI/CD)	GitLab CI/CD, Azure Pipelines	Ενσωμάτωση κώδικα σε κοινό αποθετήριο και παρακολούθηση της αυτόματης διαδικασίας build και deploy μέσω CI/CD pipelines

7. Σχόλια-Παρατηρήσεις

1. Εργασιακό περιβάλλον

Οι συνθήκες εργασίας κατά τη διάρκεια της πρακτικής μου ήταν ιδιαίτερα ικανοποιητικές. Από την αρχή υπήρξε σεβασμός στα δικαιώματά μου, τόσο σε ό,τι αφορά την τήρηση του ωραρίου και των διαλειμμάτων, όσο και σε σχέση με τις επιπλέον παροχές που προσφέρει η εταιρία στους εργαζομένους της. Η ομάδα του ανθρώπι-

νου δυναμικού (HR) συνέβαλε ουσιαστικά στην ομαλή ένταξή μου, παρέχοντάς μου οργανωμένη υποστήριξη και βοήθεια σε οποιοδήποτε ζήτημα προέκυπτε. Παράλληλα, το κλίμα στην ομάδα στην οποία εντάχθηκα ήταν εξαιρετικό. Υπήρχε ειλικρινής συνεργασία και φιλική διάθεση μεταξύ των μελών, γεγονός που με έκανε να αισθανθώ μέρος της ομάδας από την πρώτη κιόλας ημέρα. Οι συνάδελφοί μου ήταν πάντα πρόθυμοι να απαντήσουν σε απορίες μου και να εξηγήσουν έννοιες ή διαδικασίες που δεν γνώριζα, αναγνωρίζοντας ότι πρόκειται για την πρώτη μου ουσιαστική εμπειρία στον χώρο της επαγγελματικής ανάπτυξης λογισμικού. Σε καμία περίπτωση δεν υπήρξαν υπερβολικές απαιτήσεις, αλλά αντίθετα δόθηκε έμφαση στην υποστήριξη, την καθοδήγηση και την απόκτηση εμπειρίας με πραγματικούς όρους.

2. Λειτουργία του τμήματος

Το τμήμα στο οποίο εντάχθηκα διαθέτει τον ρόλο του coordinator, γεγονός που ενισχύει σημαντικά την οργανωτική του λειτουργία. Ο coordinator αποτελεί βασικό σημείο αναφοράς για ζητήματα καθημερινής λειτουργίας, εξασφαλίζοντας κοινή ροή και συντονισμό της ομάδας, είτε αυτό αφορά την κατανομή των θέσεων εργασίας στον χώρο, είτε τη διαχείριση των αδειών, είτε διοικητικά και γραφειοκρατικά θέματα. Παρότι η ύπαρξη αυτού του ρόλου συμβάλλει θετικά στη δομή και τη συνέπεια του τμήματος, κατά την έναρξη της πρακτικής μου υπήρχαν ορισμένες προκλήσεις λόγω υποστελέχωσης, ειδικά στην ομάδα του backend.

Το μειωμένο ανθρώπινο δυναμικό είχε ως αποτέλεσμα οι εργαζόμενοι να συμμετέχουν ταυτόχρονα σε πολλαπλά έργα, κάτι που δυσκόλευε τον σαφή διαχωρισμό των ομάδων και επιβάρυνε τη ροή εργασίας. Η κατάσταση αυτή, αν και αναμενόμενη σε περιόδους μετάβασης, επηρέαζε την αποδοτικότητα και την εσωτερική επικοινωνία του τμήματος. Περίπου στα μέσα της πρακτικής μου, η εικόνα αυτή άρχισε να βελτιώνεται αισθητά, καθώς πραγματοποιήθηκαν νέες προσλήψεις για την ενίσχυση των ομάδων. Η ένταξη πολλών νέων μελών μέσα σε σύντομο χρονικό διάστημα προκάλεσε αρχικά μια προσωρινή έλλειψη συντονισμού, καθώς απαιτήθηκε χρόνος για την εκπαίδευσή τους και την πλήρη ενσωμάτωσή τους στις καθημερινές διαδικασίες. Ωστόσο, στον τελευταίο μήνα της πρακτικής μου, η κατάσταση είχε σταθεροποιηθεί, οι ομάδες είχαν δομηθεί με βάση τα επιμέρους έργα, και υπήρχε πλέον σαφής ροή και καταμερισμός αρμοδιοτήτων.

Σήμερα, που συνεχίζω να εργάζομαι στην εταιρία ως μόνιμος υπάλληλος, το τμήμα παρουσιάζει μια πολύ πιο σταθερή και οργανωμένη εικόνα, με σαφείς ρόλους, συντονισμένες ομάδες και διαδικασίες που λειτουργούν υποδειγματικά.

3. Παράγοντες που αποτέλεσαν εμπόδιο στην πρακτική μου

Κατά τη διάρκεια της πρακτικής μου υπήρξαν ορισμένοι παράγοντες που λειτούργησαν ως εμπόδια στην καθημερινή εξέλιξη και εκπαίδευσή μου, χωρίς όμως να ακυρώνουν τη συνολική θετική εμπειρία. Ένα από τα βασικά ζητήματα ήταν η υποστελέχωση του τμήματος, ειδικά κατά τους πρώτους μήνες, γεγονός που είχε ως αποτέλεσμα τα μέλη της ομάδας να είναι επιφορτισμένα με πολλαπλά έργα ταυτόχρονα. Αυτό επηρέασε τη διαθεσιμότητα χρόνου για ουσιαστική καθοδήγηση και εκπαίδευση, ειδικά στην αρχική μου φάση, όπου η εξοικείωση με τις τεχνολογίες και τις διαδικασίες ήταν πιο απαιτητική.

Ένας ακόμη παράγοντας που αποτέλεσε περιορισμό στην πλήρη αξιοποίηση της πρακτικής μου ήταν η έλλειψη εργασιών (tickets) σχεδιασμένων για αρχάριους. Επειδή τα περισσότερα έργα στα οποία συμμετείχε η ομάδα βρίσκονταν ήδη σε προχωρημένο στάδιο ανάπτυξης ή παραγωγής, τα διαθέσιμα tasks απαιτούσαν συνήθως βαθιά γνώση του κώδικα, εμπειρία με συγκεκριμένες τεχνολογίες ή εξοικείωση με πολύπλοκες επιχειρησιακές ροές. Αυτό είχε ως αποτέλεσμα να μην υπάρχουν πάντοτε εργασίες που να μπορούσαν να ανατεθούν σε ασκούμενους, ιδίως στην αρχή, όπου ο βαθμός εξοικείωσης με τα συστήματα και τις απαιτήσεις ήταν ακόμα περιορισμένος. Αν και υπήρχε πρόθεση από την ομάδα να με εντάξει ενεργά στα έργα, η φύση των διαθέσιμων tickets σε ορισμένες περιόδους δεν ευνοούσε τη σταδιακή ανάληψη ευθυνών, όπως θα συνέβαινε αν υπήρχαν πιο απλές, καθοδηγούμενες εργασίες κατάλληλες για εκπαίδευση.

Επιπλέον, σε περιόδους όπου υπήρχαν έντονα deadlines ή επείγουσες παραδόσεις, η προτεραιότητα δινόταν δικαιολογημένα στην ολοκλήρωση των παραδοτέων των έργων. Κατά συνέπεια, η δυνατότητα άμεσης επίβλεψης ή mentoring περιοριζόταν. Ωστόσο, ακόμα και σε αυτές τις συνθήκες, προσπαθούσα να παραμένω ενεργός/ή, να αξιοποιώ τον χρόνο για να μελετήσω, να εξερευνώ τον κώδικα και να προτείνω λύσεις όπου μπορούσα, ενώ όταν υπήρχε διαθέσιμος χρόνος από την ομάδα, γινόταν στοχευμένη υποστήριξη. Παρότι οι δυσκολίες αυτές επιβράδυναν σε ορισμένες φάσεις τον ρυθμό μάθησης, θεωρώ ότι με ενίσχυσαν στο να αναπτύξω μεγαλύτερη αυτονομία και προσαρμοστικότητα.

4. Παράγοντες που διευκόλυναν την πρακτική μου

Παρά τις όποιες δυσκολίες, υπήρξαν αρκετοί παράγοντες που διευκόλυναν την πρακτική μου και συνέβαλαν στην ομαλότερη προσαρμογή μου στο περιβάλλον εργασίας. Καταρχάς, το γεγονός ότι εντάχθηκα από την αρχή σε μια πραγματική ομάδα ανάπτυξης και όχι σε κάποιο απομονωμένο πλαίσιο εκπαίδευσης, μου έδωσε τη δυνατότητα να παρακολουθώ πώς λειτουργεί στην πράξη ένας κύκλος εργασιών σε agile περιβάλλον. Η συμμετοχή μου στα daily meetings, με βοήθησε να κατανοήσω τη ροή των εργασιών, την ορολογία που χρησιμοποιείται και τον τρόπο με τον οποίο συντονίζονται τα μέλη μιας ομάδας. Επιπλέον, η ύπαρξη τεχνικών εργαλείων

όπως το Jira, το Git και τα pipelines CI/CD αποτέλεσαν πολύτιμο «πεδίο παρατήρησης», καθώς μέσω αυτών μπόρεσα να κατανοήσω πώς οργανώνεται ένα έργο σε επίπεδο διαδικασιών.

Παρότι η συμμετοχή μου περιοριζόταν κυρίως στην παρακολούθηση ή σε μικρότερες εργασίες, υπήρχε διάθεση βοήθειας όταν ζητούσα διευκρινίσεις ή αντιμετώπιζα κάποιο τεχνικό πρόβλημα. Το ευρύτερο εργασιακό κλίμα ήταν ήρεμο, ευγενικό και υποστηρικτικό, κάτι που μου έδινε την άνεση να ρωτήσω χωρίς φόβο ότι θα εκτεθώ ή θα φανώ ανεπαρκής. Τέλος, σημαντική βοήθεια αποτέλεσε και η διάθεση για καθοδήγηση από ορισμένα μέλη της ομάδας, είτε μέσα από μικρές εξηγήσεις, είτε μέσω τεχνικών παραδειγμάτων ή έτοιμου κώδικα που μπορούσα να μελετήσω. Όλα αυτά, σε συνδυασμό με τον χρόνο που μου δόθηκε στην αρχή για εξοικείωση, δημιούργησαν ένα περιβάλλον στο οποίο ένιωσα ότι μπορώ να μάθω, να παρατηρήσω και να αρχίσω να χτίζω μια πιο ρεαλιστική εικόνα για το πώς λειτουργεί ένας επαγγελματικός τεχνολογικός οργανισμός.

5. Γνώσεις και εμπειρία που αποκτήθηκε

Η πρακτική μου άσκηση αποτέλεσε μια ουσιαστική εισαγωγή στον κόσμο της επαγγελματικής ανάπτυξης λογισμικού και μου επέτρεψε να αποκτήσω πολύτιμες γνώσεις τόσο σε τεχνικό όσο και σε λειτουργικό επίπεδο. Κατά τη διάρκεια της πρακτικής εξοικειώθηκα με τεχνολογίες όπως Java, Spring Boot, PostgreSQL, καθώς και με εργαλεία που χρησιμοποιούνται ευρέως στον κλάδο, όπως το Git για διαχείριση εκδόσεων, το Jira για οργάνωση εργασιών, το Liquibase για διαχείριση αλλαγών σε βάσεις δεδομένων και το SonarQube για στατική ανάλυση κώδικα. Παράλληλα, απέκτησα εμπειρία στη χρήση του JUnit για μονάδες ελέγχου και κατάλαβα τη σημασία του testing στην ποιότητα λογισμικού.

Πέρα από τις τεχνικές δεξιότητες, γνώρισα από κοντά τον τρόπο λειτουργίας μιας ομάδας που εργάζεται με βάση agile μεθοδολογίες, μέσα από τη συμμετοχή μου σε καθημερινά standups, τη χρήση ticketing συστημάτων και την τήρηση των sprint κύκλων. Έμαθα να κινούμαι σε πραγματικό περιβάλλον έργου, να ακολουθώ τις διαδικασίες, να παρακολουθώ την εξέλιξη του κώδικα και να κατανοώ πώς γίνεται η συνεργασία ανάμεσα σε διαφορετικούς ρόλους.

Η εμπειρία αυτή με βοήθησε να αναπτύξω την ικανότητα επίλυσης προβλημάτων, να εξασκηθώ στην αυτονομία και την προσαρμοστικότητα, και να κατανοήσω τι σημαίνει να συμμετέχεις ενεργά —έστω και σταδιακά— σε παραγωγικά έργα με πραγματικές απαιτήσεις. Συνολικά, η πρακτική μου με εφοδίασε με βάσεις πάνω στις οποίες μπορώ να εξελιχθώ επαγγελματικά, έχοντας πλέον μια πιο ρεαλιστική και πρακτική εικόνα του χώρου της τεχνολογίας.

6. Προτάσεις για βελτίωση του τμήματος

Η συνολική εμπειρία της πρακτικής μου ήταν θετική, ιδιαίτερα στο αρχικό της στάδιο, όπου υπήρχε σαφής οργάνωση και μεθοδικότητα. Η εισαγωγική εκπαίδευση, σε συνδυασμό με την ανάπτυξη μιας demo εφαρμογής, ήταν καλοδομημένη και με βοήθησε να εξοικειωθώ με το τεχνολογικό περιβάλλον της εταιρίας. Ωστόσο, μετά το πέρας αυτής της φάσης, υπήρξαν σημεία όπου η οργάνωση μπορούσε να είναι πιο λειτουργική. Πιο συγκεκριμένα, η μετάβαση από την εκπαιδευτική φάση στην πραγματική εμπλοκή με έργα έγινε με αρκετά ανοργάνωτο τρόπο. Τοποθετήθηκα σε ένα έργο που βρισκόταν ήδη σε στάδιο συντήρησης και σταθεροποίησης, κάτι που περιορίζει εκ των πραγμάτων τις δυνατότητες ενεργής συμμετοχής ενός αρχάριου.

Θα ήταν ιδιαίτερα χρήσιμο, πριν την ένταξη σε κάποιο έργο, να προηγείται μια αξιολόγηση της πολυπλοκότητας και του σταδίου στο οποίο βρίσκεται το έργο, ώστε οι ασκούμενοι να τοποθετούνται σε περιβάλλοντα που προσφέρουν περισσότερες ευκαιρίες για πρακτική εξάσκηση και ενεργή συμμετοχή. Ιδανικά, η πρώτη εμπλοκή με «ζωντανό» έργο θα μπορούσε να αφορά κάποιο πιο ενεργό στάδιο ανάπτυξης ή τουλάχιστον επιμέρους, καλά οριοθετημένες εργασίες (π.χ. μικρά tickets, τεχνικό documentation, testing), προσαρμοσμένες στο επίπεδο εμπειρίας κάθε ασκούμενου.

Ένα ακόμη σημείο που παρουσίασε δυσκολία ήταν η έλλειψη τεκμηριωμένου οδηγού για τη ρύθμιση του τοπικού περιβάλλοντος και τη βασική κατανόηση της λειτουργίας των έργων. Η απουσία ενός τέτοιου οδηγού είχε ως αποτέλεσμα να χρειάζεται είτε συνεχής βοήθεια από μέλη της ομάδας είτε χρονοβόρα αναζήτηση λύσεων, ακόμα και για βασικά βήματα όπως η εγκατάσταση εξαρτήσεων ή η σωστή εκτέλεση των υπηρεσιών. Θα ήταν ιδιαίτερα υποστηρικτικό να δημιουργηθεί ένας συνοπτικός αλλά λειτουργικός οδηγός τεκμηρίωσης ανά έργο ή τεχνολογία, με βασικά βήματα για τοπική εγκατάσταση, ενδεικτικές εντολές, περιβαλλοντικές μεταβλητές και γνωστά σημεία προσοχής. Ένας τέτοιος οδηγός θα μείωνε σημαντικά τον χρόνο προσαρμογής των νέων μελών και θα τους επέτρεπε να ξεκινήσουν την ενασχόληση με το έργο με μεγαλύτερη αυτονομία και αυτοπεποίθηση.