ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΕΛΕΓΧΟΣ ΑΠΟΔΟΣΗΣ ΚΑΙ ΜΕΘΟΔΟΙ ΒΕΛΤΙΩΣΗΣ: ΜΙΑ ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΓΙΑ ΤΗΝ ΕΝΙΣΧΥΣΗ ΤΗΣ ΑΠΟΔΟΣΗΣ ΕΝΟΣ ΣΥΣΤΗΜΑΤΟΣ»

Του φοιτητή
Ανθόπουλου Παναγιώτη
Αρ. Μητρώου: 154431

Επιβλέπων
Αντώνης Σιδηρόπουλος
Αναπληρωτής Καθηγητής

25/05/2025

Τίτλος Π.Ε: ΕΛΕΓΧΟΣ ΑΠΟΔΟΣΗΣ ΚΑΙ ΜΕΘΟΔΟΙ ΒΕΛΤΙΩΣΗΣ: ΜΙΑ ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΓΙΑ ΤΗΝ ΕΝΙΣΧΥΣΗ ΤΗΣ ΑΠΟΔΟΣΗΣ ΕΝΟΣ ΣΥΣΤΗΜΑΤΟΣ

Κωδικός Π.Ε.: 23248

Ονοματεπώνυμο φοιτητή: ΑΝΘΟΠΟΥΛΟΣ ΠΑΝΑΓΙΩΤΗΣ

Ονοματεπώνυμο εισηγητή: ΣΙΔΗΡΟΠΟΥΛΟΣ ΑΝΤΩΝΗΣ

Ημερομηνία ανάληψης Π.Ε.: 15/09/2023

Ημερομηνία περάτωσης Π.Ε.: 31/05/2025

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Πρόλογος

Η παρούσα πτυχιακή εργασία έχει ως σκοπό τη μελέτη και αξιολόγηση της απόδοσης ενός πληροφοριακού συστήματος μέσω της χρήσης σύγχρονων τεχνικών βελτιστοποίησης. Η ανάγκη για γρήγορη και αποδοτική επεξεργασία πληροφοριών αποτελεί κρίσιμο παράγοντα στην ανάπτυξη εφαρμογών, ιδίως όταν διαχειρίζονται μεγάλο όγκο δεδομένων και πολλαπλά αιτήματα ταυτόχρονα.

Η εργασία αυτή επιχείρησε να εντοπίσει τα σημεία συμφόρησης σε ένα υφιστάμενο σύστημα, να καταγράψει την απόδοσή του πριν και μετά τις βελτιώσεις, και να αναδείξει τις τεχνικές που οδήγησαν σε σημαντική μείωση των χρόνων απόκρισης. Μέσα από αυτή τη διαδικασία, αποκόμισα πολύτιμες γνώσεις τόσο στον τομέα της ανάλυσης απόδοσης συστημάτων όσο και στη χρήση εργαλείων όπως το JMeter και το Redis.

Περίληψη

Η παρούσα πτυχιακή εργασία επικεντρώνεται στη μελέτη και βελτιστοποίηση της απόδοσης ενός πληροφοριακού συστήματος διαχείρισης του ανθρώπινου δυναμικού μιας επιχείρησης. Βασικό αντικείμενο της έρευνας αποτέλεσε η αξιολόγηση των χρόνων απόκρισης σε επιλεγμένα API αιτήματα και η εφαρμογή τεχνικών που στοχεύουν στη βελτίωση της αποδοτικότητας του συστήματος. Αρχικά, καταγράφηκαν οι επιδόσεις του συστήματος πριν από οποιαδήποτε επέμβαση, μέσω εργαλείων όπως το Apache JMeter, το οποίο χρησιμοποιήθηκε για την προσομοίωση φόρτου και την εξαγωγή μετρήσεων.

Ανάμεσα στα κύρια προβλήματα που εντοπίστηκαν ήταν οι μεγάλοι χρόνοι απόκρισης σε αιτήματα με υψηλό όγκο δεδομένων. Ως απάντηση σε αυτά τα προβλήματα, υλοποιήθηκαν βελτιώσεις μέσω caching, φιλτραρίσματος δεδομένων και ανασχεδιασμού SQL ερωτημάτων. Η σύγκριση των αποτελεσμάτων πριν και μετά τις βελτιώσεις έδειξε σαφή μείωση στους χρόνους απόκρισης και γενικότερη αύξηση της απόδοσης, ειδικά για αιτήματα που αξιοποιούν προσωρινή αποθήκευση.

Ωστόσο, η μελέτη αναγνωρίζει ορισμένους περιορισμούς, όπως η χρήση στατικών δεδομένων και η απουσία δοκιμών σε συνθήκες υψηλού φορτίου πραγματικών χρηστών. Παρ' όλα αυτά, τα ευρήματα καταδεικνύουν τη σημασία της βελτιστοποίησης τόσο σε επίπεδο backend όσο και στη δομή των δεδομένων.

Η εργασία συμβάλλει στην κατανόηση τεχνικών απόδοσης σε web συστήματα και θέτει τις βάσεις για μελλοντικές έρευνες πάνω σε scalable αρχιτεκτονικές, έξυπνες στρατηγικές χρήσης προσωρινής μνήμης και δοκιμές σε cloud περιβάλλοντα.

Performance Testing and Optimization Methods: A Comparative Study for Enhancing System Efficiency

Panagiotis Anthopoulos

Abstract

This thesis focuses on the study and optimization of the performance of an HRM web application. The primary objective of the research was to evaluate response times of selected API requests and apply techniques aimed at improving the system's efficiency. Initially, performance metrics were recorded before any optimization interventions, using tools such as Apache JMeter to simulate load and collect response time measurements.

Among the main issues identified were high response times for data-heavy requests. To address these challenges, several optimizations were implemented, including caching, data filtering and restructuring of SQL queries. The comparison between the system's performance before and after the changes demonstrated a significant reduction in response times and an overall improvement in efficiency, particularly for requests leveraging caching mechanisms.

Nonetheless, the study acknowledges certain limitations, such as the use of static test data and the absence of load testing under real-world user traffic conditions. Despite these constraints, the findings highlight the importance of backend optimization and efficient data handling in improving application performance.

This thesis contributes to the broader understanding of performance techniques in web-based systems and lays the groundwork for future research on scalable architectures, advanced caching strategies, and testing in cloud-based environments.

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Απόδοση συστημάτων: Έννοιες και προσεγγίσεις

Στη σύγχρονη ανάπτυξη λογισμικού, η απόδοση των εφαρμογών ιστού έχει καταστεί κρίσιμος παράγοντας για τη διασφάλιση της ικανοποίησης των χρηστών και της επιχειρηματικής επιτυχίας. Καθώς οι εφαρμογές ιστού καλούνται να διαχειρίζονται αυξανόμενο αριθμό χρηστών, μεγαλύτερο όγκο δεδομένων και πιο σύνθετες εργασίες, η βελτιστοποίηση της απόδοσης είναι απαραίτητη για τη διατήρηση ανταγωνιστικού πλεονεκτήματος. Οι αργοί χρόνοι φόρτωσης, οι μη ανταποκρινόμενες διεπαφές χρήστη ή η διακοπή λειτουργίας του διακομιστή μπορεί να οδηγήσουν σε κακές εμπειρίες χρηστών, μειωμένη παραγωγικότητα και σημαντικές οικονομικές απώλειες. Μια μελέτη της Akamai διαπίστωσε ότι μια καθυστέρηση ακόμη και 100 χιλιοστών του δευτερολέπτου στο χρόνο φόρτωσης της σελίδας μπορεί να μειώσει τα ποσοστά μετατροπής έως και 7%, υπογραμμίζοντας την ανάγκη οι εφαρμογές ιστού να παρέχουν γρήγορες και αξιόπιστες επιδόσεις [1].

Πέρα από τη βελτίωση της εμπειρίας των χρηστών, η καλή απόδοση μιας εφαρμογής μειώνει τις απαιτήσεις σε υπολογιστικούς πόρους, γεγονός που συμβάλλει στη μείωση του κόστους διατήρησης. Αυτό γίνεται ιδιαίτερα εμφανές στις περιπτώσεις cloud hosting, όπου το κόστος είναι άμεσα συνδεδεμένο με τους πόρους που καταναλώνονται από την εφαρμογή σε πραγματικό χρόνο. Μια εφαρμογή με κακή απόδοση μπορεί να οδηγήσει σε υπερβολική κατανάλωση πόρων, δημιουργώντας έτσι ένα σημαντικό οικονομικό βάρος για την επιχείρηση. Επιπλέον, οι εφαρμογές που δεν είναι αποδοτικές μπορεί να επηρεάσουν την κλιμάκωση της επιχείρησης. Καθώς ο αριθμός των χρηστών αυξάνεται, η ανάγκη για επιπλέον πόρους μπορεί να μεγαλώνει δυσανάλογα αν η εφαρμογή δεν λειτουργεί αποτελεσματικά. Με την πάροδο του χρόνου, αυτό μπορεί να οδηγήσει σε δυσκολίες στην επέκταση της επιχείρησης και αύξηση του χρόνου διαχείρισης των πόρων, ενώ οι πελάτες ενδέχεται να βιώσουν καθυστερήσεις, επηρεάζοντας αρνητικά την εμπιστοσύνη τους στην επιχείρηση.

Σε αυτό το πλαίσιο, ο έλεγχος επιδόσεων αποτελεί ένα σημαντικό κομμάτι της ανάπτυξης λογισμικού, βοηθώντας τους προγραμματιστές να εντοπίσουν πιθανά σημεία συμφόρησης, να μετρήσουν τους χρόνους απόκρισης υπό διάφορες συνθήκες και να διασφαλίσουν ότι οι εφαρμογές πληρούν τα πρότυπα επεκτασιμότητας και αξιοπιστίας. Ενώ η λειτουργική δοκιμή επαληθεύει ότι μια διαδικτυακή εφαρμογή συμπεριφέρεται όπως αναμένεται, η δοκιμή επιδόσεων εστιάζει στις μη λειτουργικές απαιτήσεις, διασφαλίζοντας ότι το σύστημα μπορεί να διαχειριστεί αποτελεσματικά τα φορτία του πραγματικού κόσμου.

1.1.1 Μετρικές απόδοσης

Η αξιολόγηση της απόδοσης ενός συστήματος βασίζεται σε διάφορες μετρικές που επιτρέπουν την κατανόηση της αποδοτικότητας και της αποτελεσματικότητάς του. Οι βασικές μετρικές που χρησιμοποιούνται για την ανάλυση της απόδοσης περιλαμβάνουν:

- **Χρόνος απόκρισης:**

Ο χρόνος απόκρισης αναφέρεται στη χρονική διάρκεια που μεσολαβεί από την αποστολή ενός αιτήματος από τον χρήστη έως την επιστροφή της απάντησης από την εφαρμογή. Αποτελεί έναν από τους πλέον καθοριστικούς παράγοντες για την εμπειρία χρήστη, καθώς σημαντικές καθυστερήσεις μπορούν να μειώσουν την αποδοτικότητα και να οδηγήσουν σε αρνητική αντίληψη της ποιότητας του συστήματος. Ιδιαίτερα σε διαδικτυακές εφαρμογές, η αυξημένη καθυστέρηση στην απόκριση

συσχετίζεται με χαμηλότερα επίπεδα ικανοποίησης των χρηστών, ενώ ερευνητικά δεδομένα υποδεικνύουν ότι χρόνοι απόκρισης που υπερβαίνουν τα 12 δευτερόλεπτα ενδέχεται να προκαλέσουν σημαντική μείωση στη δέσμευση και την αλληλεπίδραση των χρηστών με την εφαρμογή.

Επιπλέον, ο χρόνος απόκρισης διαδραματίζει κρίσιμο ρόλο στη συνολική αντίληψη των χρηστών σχετικά με την αποδοτικότητα και τη χρηστικότητα της εφαρμογής. Ακόμη και σε περιπτώσεις όπου η διεπαφή χρήστη είναι εργονομικά σχεδιασμένη και λειτουργικά αποδοτική, σημαντικές καθυστερήσεις ενδέχεται να υποβαθμίσουν την εμπειρία χρήστη. Ειδικότερα, έρευνες που αφορούν εφαρμογές κινητών τηλεφώνων καταδεικνύουν ότι οι χρήστες εμφανίζουν μειωμένη ανοχή σε καθυστερήσεις, καθώς αναμένουν άμεση απόκριση από τις εφαρμογές. Η ανοχή αυτή μειώνεται σημαντικά όσο αυξάνεται ο χρόνος απόκρισης, γεγονός που αναδεικνύει τη σημασία της βελτιστοποίησης της ταχύτητας στις σύγχρονες εφαρμογές [2], [3].

- **Ρυθμαπόδοση (Throughput):**

Η ρυθμαπόδοση αποτελεί μια θεμελιώδη μετρική απόδοσης, η οποία εκφράζει τον αριθμό των αιτημάτων ή συναλλαγών που μπορεί να επεξεργαστεί ένα σύστημα εντός μιας καθορισμένης χρονικής περιόδου. Αποτελεί βασικό δείκτη της ικανότητας ενός συστήματος να διαχειρίζεται υψηλά φορτία και να εξασφαλίζει αποδοτική κλιμάκωση, ιδίως σε περιβάλλοντα με αυξημένες απαιτήσεις διαθεσιμότητας και απόκρισης.

Η ανάλυση της ρυθμαπόδοσης είναι κρίσιμη για την αξιολόγηση της ανθεκτικότητας και της απόδοσης ενός συστήματος υπό συνθήκες υψηλού φόρτου. Δοκιμές φόρτισης (load testing) και δοκιμές αντοχής (stress testing) επιτρέπουν τον εντοπισμό πιθανών σημείων συμφόρησης, όπως η εξάντληση υπολογιστικών πόρων ή η εμφάνιση διαρροών μνήμης. Επιπλέον, η αξιολόγηση της ρυθμαπόδοσης παρέχει σημαντικές πληροφορίες για την ικανότητα του συστήματος να ανακάμπτει από αιχμές στην κίνηση, όπως αυτές που παρατηρούνται κατά την κυκλοφορία νέων προϊόντων ή κατά τη διάρκεια εντατικών διαφημιστικών εκστρατειών.

Η βελτιστοποίηση της ρυθμαπόδοσης συνδέεται άμεσα με την ικανότητα του συστήματος να διαχειρίζεται παράλληλα αιτήματα χωρίς να υποβαθμίζεται η συνολική απόδοσή του. Ως εκ τούτου, η συνεχής παρακολούθηση και προσαρμογή των σχετικών παραμέτρων αποτελεί κεντρική στρατηγική για τη διατήρηση της επιχειρησιακής αποδοτικότητας και της εμπειρίας χρήστη.[4]

- **Χρήση πόρων:**

Η χρήση πόρων αναφέρεται στην κατανάλωση υπολογιστικών πόρων, όπως η κεντρική μονάδα επεξεργασίας (CPU), η μνήμη (RAM) και ο αποθηκευτικός χώρος (disk I/O), κατά την επεξεργασία των αιτημάτων των χρηστών. Η αποδοτική διαχείριση αυτών των πόρων είναι κρίσιμη για τη διατήρηση της συνολικής απόδοσης του συστήματος, καθώς επηρεάζει άμεσα τη χρονική απόκριση, την επεκτασιμότητα και τη σταθερότητα της εφαρμογής.

Συστήματα που λειτουργούν υπό υψηλό φόρτο διαχειρίζονται πολλαπλά ταυτόχρονα αιτήματα μέσω διαθέσιμων πόρων, όπως τα νήματα επεξεργασίας (threads) και οι συνδέσεις με βάσεις δεδομένων. Η αποδοτική κατανομή αυτών των πόρων αποτελεί βασικό παράγοντα για τη διασφάλιση της βέλτιστης λειτουργίας του συστήματος και την αποφυγή συμφόρησης. Για παράδειγμα, η υπέρμετρη κατανάλωση CPU (π.χ., όταν η χρήση της υπερβαίνει το 80%) ή η ανεπαρκής διαχείριση της μνήμης μπορεί να οδηγήσει σε αυξημένους χρόνους απόκρισης και αποτυχίες αιτημάτων, ιδίως σε περιόδους αιχμής.

Η εφαρμογή στρατηγικών βελτιστοποίησης, όπως η εξισορρόπηση φορτίου (load balancing) και η δυναμική κατανομή πόρων (dynamic resource allocation), συμβάλλει στη βελτίωση της αποδοτικότητας

του συστήματος. Αυτές οι τεχνικές επιτρέπουν την προσαρμογή των διαθέσιμων υπολογιστικών πόρων στις εκάστοτε απαιτήσεις, ελαχιστοποιώντας την άσκοπη κατανάλωση και διατηρώντας υψηλά επίπεδα απόδοσης. Η συνεχής παρακολούθηση και προσαρμογή των πόρων αποτελεί αναπόσπαστο μέρος της διαχείρισης της απόδοσης, διασφαλίζοντας τη σταθερή και αξιόπιστη λειτουργία της εφαρμογής υπό μεταβαλλόμενες συνθήκες φόρτου[5], [6].

- **Χρόνος αναμονής (Wait time)**

Ο χρόνος αναμονής αναφέρεται στο χρονικό διάστημα που ένα αίτημα παραμένει σε ουρά πριν ξεκινήσει η επεξεργασία του από το σύστημα. Αποτελεί ένα σημαντικό υποσύνολο του συνολικού **χρόνου απόκρισης (response time)** και έχει άμεση επίδραση στην εμπειρία του χρήστη, ειδικά σε συστήματα όπου η ταχύτητα εξυπηρέτησης είναι κρίσιμη, όπως εφαρμογές σε πραγματικό χρόνο ή σε περιβάλλοντα με υψηλό φόρτο χρηστών. Η διάρκεια του χρόνου αναμονής επιρεάζεται απο παράγοντσε οπως το φορτίο του συστήματος, δηλαδή τον αριθμό των αιτημάτων που εξηπηρετούνται ταυτόχρονα, την αποδοτικότητα του αλγορίθμου διαχείρισης των εργασιών, καθώς και τη διαθεσιμότητα των πόρων του συστήματος, όπως την υπολογιστική ισχύ και τις συνδέσεις στη βάση δεδομένων.

Όταν ο χρόνος αναμονής αυξάνεται υπερβολικά, μπορεί να οδηγήσει σε συσσώρευση αιτημάτων, δημιουργώντας συμφόρηση και υποβάθμιση της απόδοσης του συστήματος. Για να μειωθεί ο χρόνος αναμονής, χρησιμοποιούνται τεχνικές όπως η εξισορρόπηση φορτίου, η οποία κατανέμει ομοιόμορφα τα αιτήματα στους διαθέσιμους διακομιστές, η προτεραιοποίηση των αιτημάτων, ώστε τα κρίσιμα αιτήματα να εξυπηρετούνται γρηγορότερα, και η ασύγχρονη επεξεργασία, η οποία επιτρέπει την παράλληλη επεξεργασία πολλών αιτημάτων. Επίσης, η συνεχής παρακολούθηση του χρόνου αναμονής μέσω εργαλείων όπως το Little's Law ($L = \lambda W$), που συνδέει τον μέσο αριθμό αιτημάτων στο σύστημα με τον ρυθμό άφιξης τους και τον μέσο χρόνο παραμονής τους, βοηθά στη βελτίωση της συνολικής απόδοσης του συστήματος και στην αποφυγή της συμφόρησης [7].

- **Ρυθμός σφαλμάτων (Error rate)**

Ο ρυθμός σφαλμάτων αποτελεί μια από τις βασικές μετρικές απόδοσης, η οποία αποτυπώνει την αξιοπιστία του συστήματος, μετρώντας το ποσοστό των αποτυχημένων αιτημάτων ή συναλλαγών σε σχέση με το σύνολο των αιτημάτων που υποβάλλονται. Είναι ένας κρίσιμος δείκτης για την κατανόηση της λειτουργικής ακεραιότητας ενός συστήματος και της ικανότητάς του να εξυπηρετεί τους χρήστες χωρίς σφάλματα. Ο ρυθμός σφαλμάτων μπορεί να προκύπτει από διάφορους παράγοντες, όπως σφάλματα κώδικα, προβλήματα επικοινωνίας με τη βάση δεδομένων, υπερφόρτωση συστήματος ή σφάλματα δικτύου. Υψηλός ρυθμός σφαλμάτων ενδέχεται να υποδεικνύει σοβαρά προβλήματα στην υποδομή του συστήματος ή στην ποιότητα του λογισμικού, οδηγώντας σε αποτυχία της εφαρμογής ή σε κακή εμπειρία χρήστη.

Για την αποτελεσματική παρακολούθηση και βελτίωση του ρυθμού σφαλμάτων, χρησιμοποιούνται τεχνικές ανίχνευσης και καταγραφής σφαλμάτων σε πραγματικό χρόνο, προκειμένου να εντοπιστούν και να διορθωθούν άμεσα τα σφάλματα πριν επηρεάσουν την κανονική λειτουργία. Η ανάλυση του ρυθμού σφαλμάτων μπορεί να αποκαλύψει προβλήματα που σχετίζονται με την ανθεκτικότητα του συστήματος, όπως η αποτυχία αναγνώρισης αιτημάτων, ή το αδύναμο χειρισμό ακραίων καταστάσεων και σφαλμάτων. Η διατήρηση του ρυθμού σφαλμάτων σε χαμηλά επίπεδα είναι θεμελιώδης για τη διασφάλιση της σταθερότητας, της αξιοπιστίας και της γενικότερης απόδοσης του συστήματος [8].

- **Διαθεσιμότητα (Availability)**

Η διαθεσιμότητα αναφέρεται στην ικανότητα ενός συστήματος ή εφαρμογής να είναι προσβάσιμο και λειτουργικό για τους χρήστες του, χωρίς διακοπές ή ανεπιθύμητα σφάλματα, κατά τη διάρκεια μιας καθορισμένης χρονικής περιόδου. Είναι μια κρίσιμη μετρική για την αξιολόγηση της αξιοπιστίας και της συνέχειας λειτουργίας του συστήματος, καθώς η μη διαθέσιμη εφαρμογή μπορεί να προκαλέσει σοβαρές συνέπειες, όπως απώλεια χρηστών, μειωμένα έσοδα και φθορά της φήμης. Η διαθεσιμότητα υπολογίζεται συχνά ως ποσοστό του χρόνου λειτουργίας του συστήματος σε σχέση με το συνολικό χρόνο, με το ιδανικό ποσοστό να είναι 99.9% ή υψηλότερο. Παράγοντες που επηρεάζουν τη διαθεσιμότητα περιλαμβάνουν τις μηχανικές ή ηλεκτρονικές αποτυχίες των υποδομών, τη συντήρηση του συστήματος, τα σφάλματα του λογισμικού, καθώς και τις επιθέσεις ασφαλείας.

Η εξασφάλιση υψηλής διαθεσιμότητας απαιτεί τη χρήση στρατηγικών όπως η ανθεκτικότητα (resilience), η αντίγραφα ασφαλείας (redundancy) και οι υποδομές υψηλής διαθεσιμότητας (high-availability architectures), οι οποίες διασφαλίζουν ότι το σύστημα παραμένει διαθέσιμο ακόμα και σε περίπτωση αποτυχίας κάποιου από τα υποσυστήματα του. Επιπλέον, η παρακολούθηση της διαθεσιμότητας σε πραγματικό χρόνο και η ταχεία ανίχνευση και διόρθωση προβλημάτων είναι θεμελιώδη για τη διατήρηση της σταθερότητας του συστήματος και την αποφυγή χρόνων αδράνειας (downtime). Με την εφαρμογή αυτών των στρατηγικών και την προσεκτική διαχείριση των υποδομών, οι οργανισμοί μπορούν να διασφαλίσουν τη συνεχιζόμενη πρόσβαση στους χρήστες τους, ελαχιστοποιώντας τις διακοπές και βελτιώνοντας τη συνολική εμπειρία χρήστη [9] .

1.1.2 Είδη ελέγχου απόδοσης

Ο έλεγχος απόδοσης ενός συστήματος είναι κρίσιμης σημασίας για την αξιολόγηση της ικανότητάς του να ανταποκριθεί σε διάφορες συνθήκες φόρτου και χρήσης. Ανάλογα με την εφαρμογή και τους στόχους της αξιολόγησης, υπάρχουν διάφορα είδη ελέγχου που στοχεύουν να διασφαλίσουν ότι το σύστημα θα λειτουργεί αποδοτικά και αξιόπιστα κάτω από διαφορετικές συνθήκες.

Οι έλεγχοι αυτοί περιλαμβάνουν τη μέτρηση και ανάλυση των επιδόσεων του συστήματος σε σχέση με διάφορες παραμέτρους, όπως ο φόρτος, η κλιμάκωση, η αντοχή στον χρόνο, αλλά και η αντίδραση του συστήματος σε ακραίες καταστάσεις. Ο έλεγχος απόδοσης μπορεί να αποκαλύψει πιθανά σημεία συμφόρησης **(bottlenecks)**, αδυναμίες στην υποδομή ή στις εφαρμογές, και περιοχές για βελτίωση.

Οι κύριοι τύποι ελέγχου απόδοσης περιλαμβάνουν τη δοκιμή φόρτου **(Load Testing)**, τη δοκιμή υπερφόρτωσης **(Stress Testing)**, και τη δοκιμή κλιμάκωσης **(Scalability Testing)**, μεταξύ άλλων. Κάθε είδος ελέγχου αποσκοπεί στην αξιολόγηση συγκεκριμένων παραμέτρων του συστήματος, όπως η ικανότητα να διαχειριστεί αυξημένο φόρτο, η αντίδραση σε ακραίες συνθήκες, ή η ανθεκτικότητα του συστήματος με την πάροδο του χρόνου. Ακολουθεί η ανάλυση των πιο κοινών τύπων ελέγχου απόδοσης, που βοηθούν στην κατανόηση της συμπεριφοράς του συστήματος και στην αναγνώριση πιθανών σημείων βελτίωσης.

Load testing (Δοκιμή φορτίου):

Η δοκιμή φορτίου **(Load Testing)** είναι μια από τις πιο σημαντικές και ευρέως χρησιμοποιούμενες μεθόδους για τη μέτρηση της απόδοσης ενός συστήματος, υποκειμένη σε κανονικές συνθήκες χρήσης. Στόχος αυτής της διαδικασίας είναι να εκτιμηθεί η ικανότητα του συστήματος να επεξεργάζεται τον αναμενόμενο φόρτο χρηστών ή συναλλαγών χωρίς να εμφανίζει σημαντική υποβάθμιση στην απόδοσή του. Αν και η δοκιμή φορτίου συνήθως πραγματοποιείται σε ελεγχόμενα περιβάλλοντα, με συγκεκριμένους αριθμούς χρηστών ή αιτημάτων, το ζητούμενο είναι να διασφαλιστεί ότι το σύστημα λειτουργεί σύμφωνα με τις αναμενόμενες προδιαγραφές, ακόμη και υπό συνθήκες κανονικού φόρτου.

Στην πράξη, η δοκιμή φορτίου προσομοιώνει την δραστηριότητα των χρηστών ή τις συναλλαγές στην εφαρμογή κατά τη διάρκεια κανονικής χρήσης. Οι συνθήκες που προσομοιώνονται μπορεί να περιλαμβάνουν τη σταδιακή αύξηση του αριθμού των χρηστών ή την αύξηση των αιτημάτων που διαχειρίζεται το σύστημα κατά τη διάρκεια μιας συγκεκριμένης χρονικής περιόδου. Αυτό επιτρέπει στους μηχανικούς να παρατηρήσουν την αντίδραση του συστήματος σε πραγματικές συνθήκες χρήσης και να εντοπίσουν πιθανά προβλήματα ή περιορισμούς στην απόδοση, πριν από την επίσημη κυκλοφορία της εφαρμογής ή την αναβάθμιση της υποδομής.

Επιπλέον, η δοκιμή φορτίου βοηθά στην αναγνώριση του μέγιστου φόρτου που το σύστημα μπορεί να διαχειριστεί αποτελεσματικά, χωρίς να προκαλούνται καθυστερήσεις, αργοί χρόνοι απόκρισης ή υπερβολική κατανάλωση πόρων. Σε περίπτωση που το σύστημα εμφανίζει δυσλειτουργίες ή απόδοση κάτω των αναμενόμενων τιμών υπό το φορτίο, οι μηχανικοί μπορούν να αναλύσουν τα αποτελέσματα και να προχωρήσουν σε διορθωτικές ενέργειες, όπως η βελτιστοποίηση του κώδικα, η αναβάθμιση της υποδομής ή η αλλαγή του τρόπου διαχείρισης των πόρων[10].

Για παράδειγμα, ας υποθέσουμε ότι μια εφαρμογή ηλεκτρονικού εμπορίου, η οποία προσφέρει προϊόντα μέσω διαδικτύου, σχεδιάζεται να εξυπηρετεί 10.000 χρήστες την ημέρα κατά τη διάρκεια περιόδων αιχμής, όπως οι περίοδοι των προσφορών. Η δοκιμή φορτίου θα προσομοιώσει την δραστηριότητα αυτών των 10.000 χρηστών, επιτρέποντας στην ομάδα ανάπτυξης να μετρήσει την αντίδραση του συστήματος υπό κανονικές συνθήκες χρήσης. Με τη χρήση εργαλείων δοκιμής φορτίου, όπως το Apache JMeter ή το LoadRunner, το σύστημα θα εκτελέσει αιτήματα για την προβολή προϊόντων, την προσθήκη προϊόντων στο καλάθι, καθώς και την ολοκλήρωση των συναλλαγών.

Αν κατά τη διάρκεια της δοκιμής παρατηρηθεί ότι οι χρόνοι απόκρισης αυξάνονται ή ότι το σύστημα καταρρέει κάτω από τον φόρτο, οι μηχανικοί θα είναι σε θέση να εξετάσουν τα αποτελέσματα και να αναλάβουν ενέργειες για να βελτιώσουν την απόδοση πριν την κυκλοφορία της εφαρμογής στην παραγωγή.

Η δοκιμή φορτίου είναι, λοιπόν, απαραίτητη για να διασφαλιστεί ότι το σύστημα θα παραμείνει λειτουργικό και αποδοτικό κατά τη διάρκεια της καθημερινής χρήσης, εξασφαλίζοντας μια θετική εμπειρία για τους τελικούς χρήστες, ειδικά όταν το σύστημα αντιμετωπίζει μεγάλη ζήτηση.

Stress testing (Δοκιμή υπερφόρτωσης):

Η δοκιμή υπερφόρτωσης (Stress Testing) είναι μια σημαντική μέθοδος ελέγχου της απόδοσης, η οποία στοχεύει να προσδιορίσει τα όρια του συστήματος, αξιολογώντας την απόδοσή του υπό συνθήκες υπερβολικού φόρτου, οι οποίες ξεπερνούν το μέγιστο αναμενόμενο επίπεδο χρήσης. Αντίθετα με τη δοκιμή φορτίου, η οποία εξετάζει την απόδοση του συστήματος υπό κανονικές συνθήκες, η δοκιμή υπερφόρτωσης επικεντρώνεται στη θέση του συστήματος σε καταστάσεις κρίσης, όταν το φορτίο υπερβαίνει κατά πολύ τις φυσιολογικές απαιτήσεις. Η ιδέα είναι να "σπρώξουμε" το σύστημα πέρα από τα όριά του, έτσι ώστε να εντοπιστούν τα αδύνατα σημεία του, όπως η απόδοση, η αντοχή στη συντήρηση ή η σωστή ανάκαμψη από καταρρεύσεις.

Ο σκοπός της δοκιμής υπερφόρτωσης είναι να προσδιορίσει πώς το σύστημα αντιδρά σε καταστάσεις ακραίου φόρτου και ποιες είναι οι επιπτώσεις στην απόδοσή του, όπως για παράδειγμα η υπερθέρμανση της υποδομής, η αύξηση των χρόνων απόκρισης ή η κατάρρευση των υπηρεσιών. Η διαδικασία περιλαμβάνει την εφαρμογή αιτημάτων στο σύστημα με ρυθμό ταχύτερο από αυτόν που μπορεί να διαχειριστεί το σύστημα, είτε πρόκειται για αύξηση του αριθμού των χρηστών είτε για αύξηση του όγκου των δεδομένων.

Κατά τη διάρκεια της δοκιμής υπερφόρτωσης, ο στόχος είναι να αποκαλυφθούν περιοχές του συστήματος που μπορεί να αποτύχουν υπό συνθήκες ακραίου φορτίου, και να διαπιστωθεί το σημείο όπου το σύστημα δεν μπορεί πλέον να αντέξει τις απαιτήσεις, προκαλώντας πτώση της απόδοσης ή πλήρη αποτυχία της λειτουργίας του. Αυτό είναι ιδιαίτερα χρήσιμο για εφαρμογές που προορίζονται για χρήση σε περιβάλλοντα με υψηλό φόρτο, όπως οι διαδικτυακές πλατφόρμες και οι εφαρμογές που βασίζονται σε δεδομένα.

Με την ολοκλήρωση της δοκιμής, οι μηχανικοί και οι προγραμματιστές μπορούν να αξιολογήσουν πώς το σύστημα "καταρρέει" και να εξετάσουν τις συνέπειες αυτών των αποτυχιών, όπως το αν η εφαρμογή μπορεί να ανακάμψει αυτόματα ή αν απαιτεί χειροκίνητη παρέμβαση για να επανέλθει σε πλήρη λειτουργικότητα [11].

Scalability testing (Δοκιμή κλιμακωσιμότητας):

Η δοκιμή κλιμακωσιμότητας **(scalability testing)** είναι μια σημαντική διαδικασία που στοχεύει στη μέτρηση της ικανότητας ενός συστήματος να ανταποκριθεί σε αυξανόμενα επίπεδα φόρτου χωρίς να υποβαθμίζεται η απόδοσή του. Ο σκοπός αυτής της δοκιμής είναι να διαπιστωθεί αν και πώς το σύστημα μπορεί να "κλιμακωθεί", δηλαδή να αυξήσει τις δυνατότητές του, είτε οριζόντια (προσθήκη επιπλέον διακομιστών ή πόρων) είτε κάθετα (αναβάθμιση υλικού ή αυξάνοντας την υπολογιστική ισχύ) για να ανταπεξέλθει σε μεγαλύτερη ζήτηση. Η δοκιμή κλιμακωσιμότητας είναι ουσιαστική για να διασφαλιστεί ότι μια εφαρμογή ή ένα σύστημα μπορεί να συνεχίσει να λειτουργεί αποτελεσματικά και αποδοτικά, ακόμη και καθώς ο αριθμός των χρηστών ή η ποσότητα των δεδομένων αυξάνεται.

Η διαδικασία περιλαμβάνει την αύξηση της ποσότητας των αιτημάτων ή των χρηστών και την παρακολούθηση του συστήματος για να προσδιοριστεί πότε και πώς η απόδοση αρχίζει να επιδεινώνεται, αν επιδεινώνεται. Στη συνέχεια, ο στόχος είναι να διαπιστωθεί ποια είναι τα όρια του συστήματος και πώς μπορεί να βελτιωθεί η κλιμακωσιμότητα του συστήματος με την προσθήκη νέων πόρων ή την αναβάθμιση της υποδομής.

Η δοκιμή κλιμακωσιμότητας μπορεί να γίνει τόσο σε λογιστικό επίπεδο όσο και σε φυσικό επίπεδο. Σε λογιστικό επίπεδο, μπορεί να αναφέρεται στην αύξηση της ικανότητας διαχείρισης δεδομένων, στην αποδοτική κατανομή πόρων μεταξύ των εφαρμογών ή στη βελτιστοποίηση της βάσης δεδομένων. Σε φυσικό επίπεδο, αφορά την προσθήκη περισσότερων διακομιστών, τη βελτιστοποίηση των δικτυακών υποδομών και την αναβάθμιση του υλικού για να αντέχει μεγαλύτερο φόρτο.

Επιπλέον, η δοκιμή κλιμακωσιμότητας βοηθά στην αποδοτική κατανομή των πόρων και στον προγραμματισμό της υποδομής για μελλοντική ανάπτυξη. Επιτρέπει στην ομάδα ανάπτυξης να εκτιμήσει αν το σύστημα θα είναι σε θέση να διαχειριστεί την αναμενόμενη αύξηση των χρηστών ή των δεδομένων χωρίς να προκαλέσει αρνητική επίδραση στην απόδοση [12].

Endurance testing (Δοκιμή αντοχής):

Η δοκιμή αντοχής **(endurance testing)**, επίσης γνωστή ως "δοκιμή μακροχρόνιας απόδοσης", είναι μια διαδικασία που στοχεύει στην αξιολόγηση της ικανότητας ενός συστήματος να λειτουργεί αποτελεσματικά και να διατηρεί τη σταθερότητά του κάτω από συνθήκες συνεχούς φόρτου για παρατεταμένο χρονικό διάστημα. Αντίθετα με τη δοκιμή φορτίου, η οποία επικεντρώνεται στην αξιολόγηση της απόδοσης σε κανονικές συνθήκες χρήσης, η δοκιμή αντοχής εστιάζει στην απόδοση του συστήματος όταν υποβάλλεται σε σταθερή, συνεχόμενη πίεση, για να ανιχνεύσει τυχόν προβλήματα που ενδέχεται να προκύψουν με την πάροδο του χρόνου, όπως η εξάντληση πόρων, η διαρροή μνήμης ή η σταδιακή επιβράδυνση της απόδοσης.

Στην πράξη, η δοκιμή αντοχής περιλαμβάνει τη συνεχιζόμενη υποβολή του συστήματος σε αυξημένο φόρτο για πολλές ώρες ή ακόμα και ημέρες, με σκοπό να διαπιστωθεί πώς ανταποκρίνεται το σύστημα στην παρατεταμένη λειτουργία. Η διαδικασία αποκαλύπτει τη μακροχρόνια ανθεκτικότητα του συστήματος, τη σταθερότητα των πόρων και την ικανότητά του να παραμένει λειτουργικό χωρίς να εμφανίζει προβλήματα, όπως καθυστερήσεις ή απώλεια δεδομένων.

Η δοκιμή αντοχής είναι ιδιαίτερα χρήσιμη για εφαρμογές ή συστήματα που αναμένεται να είναι σε συνεχιζόμενη λειτουργία για μεγάλα χρονικά διαστήματα, όπως τα συστήματα διαχείρισης περιεχομένου, οι πλατφόρμες κοινωνικής δικτύωσης ή οι εφαρμογές cloud computing, οι οποίες απαιτούν αδιάκοπη λειτουργία και υψηλή αξιοπιστία. Σκοπός αυτής της δοκιμής είναι να διασφαλιστεί ότι το σύστημα μπορεί να υποστηρίξει μακροχρόνια φορτία χωρίς να παρουσιάζει επιβράδυνση ή σφάλματα που θα μπορούσαν να επηρεάσουν την εμπειρία του χρήστη ή την αξιοπιστία του συστήματος.

Η διαδικασία συνήθως περιλαμβάνει τη συνεχιζόμενη υποβολή του συστήματος σε σταθερό ή αυξανόμενο φόρτο για παρατεταμένο χρονικό διάστημα και τη συστηματική παρακολούθηση της χρήσης πόρων, των χρόνων απόκρισης, της σταθερότητας και της κατανάλωσης ενέργειας. Οι δοκιμές μπορούν να επαναληφθούν για να εξασφαλιστεί ότι τα αποτελέσματα είναι συνεπή και αναπαραγώγιμα [13].

Spike testing (Δοκιμή αιχμής):

Η δοκιμή αιχμής **(spike testing)** είναι μια μέθοδος ελέγχου απόδοσης που αξιολογεί την ικανότητα ενός συστήματος να ανταποκρίνεται σε απότομες και ακραίες αυξήσεις του φορτίου. Σε αντίθεση με άλλες δοκιμές που εστιάζουν στη σταθερή ή προοδευτικά αυξανόμενη πίεση στο σύστημα, η δοκιμή αιχμής εστιάζει σε ξαφνικές και δραστικές μεταβολές στον αριθμό των αιτημάτων, προσομοιώνοντας σενάρια όπως ιογενή διαδικτυακή κίνηση, flash sales ή επιθέσεις **DDoS (Distributed Denial of Service)**.

Η κύρια πρόκληση που εξετάζεται με αυτή τη δοκιμή είναι το πόσο γρήγορα και αποδοτικά μπορεί το σύστημα να ανταποκριθεί σε ένα αιφνίδιο κύμα χρηστών και αν μπορεί να επανέλθει στην κανονική του λειτουργία μόλις το φορτίο επανέλθει σε φυσιολογικά επίπεδα. Αντί να αξιολογεί μόνο τη σταθερότητα, η δοκιμή αιχμής επικεντρώνεται επίσης στη δυναμική προσαρμογή των πόρων και στη συμπεριφορά της εφαρμογής κατά τη διάρκεια και μετά από μια ξαφνική αύξηση της κίνησης.

Κατά τη διάρκεια μιας τέτοιας δοκιμής, οι βασικές μετρικές που παρακολουθούνται περιλαμβάνουν τους χρόνους απόκρισης, τη χρήση των πόρων (CPU, μνήμη, δίσκος, δίκτυο), την πιθανότητα εμφάνισης σφαλμάτων και το ποσοστό αποτυχημένων αιτημάτων. Ένα καλά σχεδιασμένο σύστημα θα πρέπει να μπορεί να διαχειριστεί τις αιχμές είτε μέσω αυτόματης κλιμάκωσης **(auto-scaling)** είτε μέσω βελτιστοποιημένων διαδικασιών διαχείρισης αιτημάτων, διατηρώντας παράλληλα αποδεκτούς χρόνους απόκρισης και σταθερότητα[14].

Volume testing (Δοκιμή όγκου):

Η δοκιμή όγκου **(volume testing)** είναι ένας τύπος ελέγχου απόδοσης που αξιολογεί τη συμπεριφορά ενός συστήματος όταν διαχειρίζεται μεγάλο όγκο δεδομένων. Ο κύριος στόχος αυτής της δοκιμής είναι να διαπιστωθεί εάν η εφαρμογή μπορεί να διαχειριστεί σωστά μεγάλες ποσότητες εισαγωγών, εγγραφών ή αιτημάτων χωρίς να επηρεαστεί η λειτουργικότητά της ή να προκληθούν σφάλματα απόδοσης.

Η δοκιμή όγκου συνήθως επικεντρώνεται σε εφαρμογές που βασίζονται σε βάσεις δεδομένων, καθώς η αυξημένη ποσότητα αποθηκευμένων δεδομένων μπορεί να επηρεάσει τους χρόνους ανάκτησης

πληροφοριών, τη χρήση μνήμης και την απόδοση των ερωτημάτων **(queries)**. Η δοκιμή αυτή αξιολογεί την ικανότητα της εφαρμογής να εκτελεί αποδοτικά λειτουργίες όπως η ανάκτηση, η εισαγωγή, η διαγραφή και η επεξεργασία δεδομένων υπό συνθήκες μεγάλου όγκου πληροφοριών.

Κατά τη διάρκεια της δοκιμής, μετρικές όπως οι χρόνοι απόκρισης των ερωτημάτων, η κατανάλωση πόρων του συστήματος (CPU, RAM, δίσκος), η διαθεσιμότητα του συστήματος και η σταθερότητα υπό μεγάλα δεδομένα καταγράφονται και αναλύονται. Εάν το σύστημα παρουσιάζει επιβράδυνση ή σφάλματα, οι μηχανικοί μπορούν να εντοπίσουν περιορισμούς αρχιτεκτονικής, να βελτιστοποιήσουν τις βάσεις δεδομένων (μέσω indexing, partitioning, caching) ή να προσαρμόσουν τις υποδομές αποθήκευσης [15].

Configuration testing (Δοκιμή διαμόρφωσης):

Η δοκιμή διαμόρφωσης (configuration testing) αποτελεί μια κρίσιμη διαδικασία ελέγχου απόδοσης, η οποία εστιάζει στην αξιολόγηση της συμπεριφοράς ενός συστήματος υπό διαφορετικές ρυθμίσεις υλικού, λογισμικού και δικτύου. Στόχος της είναι να διασφαλίσει ότι η εφαρμογή λειτουργεί ομαλά και αποδοτικά σε διάφορες συνθήκες διαμόρφωσης, αποκαλύπτοντας πιθανά προβλήματα που σχετίζονται με τη συμβατότητα, την ευστάθεια και την επεκτασιμότητα του συστήματος. Η σημασία αυτής της δοκιμής είναι ιδιαίτερα εμφανής σε περιβάλλοντα όπου η εφαρμογή πρέπει να εκτελείται σε διαφορετικές πλατφόρμες, λειτουργικά συστήματα ή εκδόσεις λογισμικού, καθώς και σε περιπτώσεις που απαιτείται υποστήριξη πολλαπλών αρχιτεκτονικών συστημάτων.

Η διαδικασία δοκιμής διαμόρφωσης περιλαμβάνει τη μεταβολή διαφόρων παραμέτρων του συστήματος και την αξιολόγηση της επίδρασής τους στην απόδοση και τη σταθερότητα της εφαρμογής. Αυτές οι παράμετροι μπορεί να αφορούν το υλικό (όπως επεξεργαστική ισχύς, διαθέσιμη μνήμη ή αποθηκευτικός χώρος), το λογισμικό (όπως διαφορετικές εκδόσεις λειτουργικών συστημάτων, βάσεων δεδομένων ή βιβλιοθηκών) και τις ρυθμίσεις δικτύου (όπως εύρος ζώνης και λανθάνουσα κατάσταση). Μέσω αυτής της δοκιμής, οι ομάδες ανάπτυξης μπορούν να εντοπίσουν κρίσιμες παραμέτρους που επηρεάζουν την απόδοση της εφαρμογής και να προσαρμόσουν ανάλογα τις ρυθμίσεις ή τις απαιτήσεις του συστήματος.

Isolation testing (Δοκιμή απομόνωσης):

Η δοκιμή απομόνωσης **(isolation testing)** αποτελεί μια εξειδικευμένη μέθοδο ελέγχου απόδοσης, η οποία στοχεύει στην εντοπισμό και ανάλυση συγκεκριμένων προβλημάτων ενός συστήματος μέσω της απομόνωσης επιμέρους συνιστωσών του. Αντί να εξετάζει τη συνολική απόδοση της εφαρμογής, η δοκιμή αυτή επικεντρώνεται σε ένα μεμονωμένο στοιχείο, όπως μια βάση δεδομένων, ένα API ή μια συγκεκριμένη διαδικασία, προκειμένου να εντοπιστούν οι ακριβείς αιτίες δυσλειτουργίας ή επιβράδυνσης. Αυτή η προσέγγιση είναι ιδιαίτερα χρήσιμη σε περιπτώσεις όπου παρουσιάζονται διαλείποντα προβλήματα απόδοσης, τα οποία είναι δύσκολο να αναπαραχθούν και να διαγνωστούν μέσω πιο γενικευμένων μεθόδων ελέγχου.

Η διαδικασία εκτέλεσης μιας δοκιμής απομόνωσης περιλαμβάνει αρχικά τον εντοπισμό του προβληματικού στοιχείου μέσα από την ανάλυση δεδομένων απόδοσης και αρχείων καταγραφής **(logs)**. Στη συνέχεια, το ύποπτο στοιχείο αποσυνδέεται από τα υπόλοιπα μέρη του συστήματος, ώστε να υποβληθεί σε ελεγχόμενες δοκιμές υπό συγκεκριμένες συνθήκες φορτίου ή χρήσης. Μέσω αυτής της διαδικασίας, οι μηχανικοί λογισμικού μπορούν να διαπιστώσουν αν το πρόβλημα οφείλεται στη συγκεκριμένη συνιστώσα ή σε κάποιον εξωτερικό παράγοντα, όπως η αλληλεπίδραση με άλλα υποσυστήματα [16].

1.2 Ορισμός και ανάλυση του προβλήματος

Η Adminkit AS, μια νορβηγική εταιρεία λογισμικού, ανέπτυξε την πλατφόρμα Adminkit, μια ολοκληρωμένη διαδικτυακή εφαρμογή σχεδιασμένη για μικρές και μεσαίες επιχειρήσεις (<100 άτομα). Η πλατφόρμα παρέχει λειτουργικότητες όπως διαχείριση προσωπικών προφίλ υπαλλήλων, ημερολόγιο αδειών, δημιουργία συμβάσεων εργασίας, διαχείριση εγγράφων, ηλεκτρονικές υπογραφές, συμμόρφωση με τον Γενικό Κανονισμό για την Προστασία Δεδομένων **(GDPR)** και πολλές άλλες δυνατότητες που διευκολύνουν την καθημερινή λειτουργία μιας επιχείρησης.

Ωστόσο, παρά τα πλεονεκτήματα που προσφέρει το λογισμικό, οι αρχικές αναφορές από τους χρήστες καθώς και τα εργαλεία παρακολούθησης της απόδοσης αποκάλυψαν σημαντικές προκλήσεις που επηρεάζουν τη συνολική εμπειρία χρήστη. Τα βασικότερα προβλήματα που εντοπίστηκαν αφορούν:

- **Αργούς χρόνους απόκρισης κατά τη διάρκεια περιόδων αιχμής:**

  Οι χρήστες παρατήρησαν ότι η εφαρμογή καθυστερεί να εκτελέσει βασικές λειτουργίες, όπως η φόρτωση προφίλ εργαζομένων ή η ενημέρωση εγγράφων, ιδιαίτερα όταν αυξάνεται ο αριθμός των ταυτόχρονων χρηστών.

- **Υψηλή κατανάλωση μνήμης υπό φορτίο:**

  Σε συνθήκες αυξημένης χρήσης, το σύστημα εμφανίζει υψηλή κατανάλωση υπολογιστικών πόρων, γεγονός που μπορεί να οδηγήσει σε μειωμένη απόδοση ή ακόμη και αστάθεια του συστήματος.

- **Επιπτώσεις στη ροή εργασίας:**

  Οι καθυστερήσεις στη λειτουργία της εφαρμογής επηρεάζουν αρνητικά την παραγωγικότητα των χρηστών, καθώς οι επιχειρήσεις βασίζονται στην Adminkit για βασικές λειτουργίες, όπως η διαχείριση αδειών και η υπογραφή συμβάσεων.

Η αντιμετώπιση αυτών των προβλημάτων είναι κρίσιμη για τη βελτίωση της συνολικής εμπειρίας χρήστη, τη διασφάλιση της σταθερότητας του συστήματος και την ικανοποίηση των απαιτήσεων μιας συνεχώς αυξανόμενης βάσης χρηστών. Για να επιτευχθεί αυτό, απαιτείται μια ολοκληρωμένη προσέγγιση στον έλεγχο απόδοσης, η οποία θα περιλαμβάνει αναλυτικές δοκιμές και την εφαρμογή βελτιώσεων τόσο σε επίπεδο λογισμικού όσο και σε επίπεδο υποδομής.

Η παρούσα μελέτη επικεντρώνεται στη διερεύνηση αυτών των ζητημάτων απόδοσης, τη σύγκριση διαφορετικών μεθοδολογιών ελέγχου, καθώς και την αξιολόγηση στρατηγικών βελτίωσης που μπορούν να οδηγήσουν σε μια πιο αποδοτική και αξιόπιστη πλατφόρμα για τους χρήστες της Adminkit.

1.3 Στόχος

Η παρούσα μελέτη αποσκοπεί στην αξιολόγηση των επιδόσεων της συγκεκριμένης διαδικτυακής εφαρμογής πριν και μετά την εφαρμογή μιας σειράς βελτιώσεων. Μέσω μιας σειράς δοκιμών επιδόσεων, οι οποίες πραγματοποιούνται σε διάφορα στάδια ανάπτυξης της εφαρμογής, στοχεύουμε στην ποσοτικοποίηση των επιπτώσεων των προσπαθειών βελτιστοποίησης σε κρίσιμες μετρικές επιδόσεων, όπως ο χρόνος απόκρισης, η ρυθμαπόδοση και η χρήση πόρων. Η κύρια επιδίωξη είναι η εκτίμηση της αποτελεσματικότητας των βελτιώσεων στην ενίσχυση της συνολικής απόδοσης της εφαρμογής, καθώς και η παροχή πληροφοριών που θα κατευθύνουν μελλοντικές στρατηγικές βελτιστοποίησης και ανάπτυξης της εφαρμογής. Η ανάλυση αυτών των μετρικών θα συμβάλει στην κατανόηση της σχέσης μεταξύ των τεχνικών βελτιώσεων και της αντίστοιχης απόδοσης, προσδιορίζοντας τα σημεία αδυναμίας και τις δυνατότητες βελτίωσης της εφαρμογής.

1.4 Πεδίο εφαρμογής

Η παρούσα μελέτη επικεντρώνεται στην αξιολόγηση της απόδοσης των αλληλεπιδράσεων μεταξύ του front-end και του back-end της εφαρμογής, καθώς και της αλληλεπίδρασης μεταξύ του back-end και της βάσης δεδομένων, σε συνθήκες τυπικής χρήσης από τους τελικούς χρήστες. Οι βελτιώσεις των επιδόσεων περιλαμβάνουν την βελτιστοποίηση των ερωτημάτων στη βάση δεδομένων, την αποδοτικότερη φόρτωση των πόρων του front-end, την ενίσχυση της επεξεργασίας από την πλευρά του διακομιστή (server-side processing) και τη βελτιστοποίηση της στρατηγικής caching. Σημειώνεται ότι η παρούσα μελέτη δεν περιλαμβάνει αξιολογήσεις στους τομείς της ασφάλειας, της χρηστικότητας ή της λειτουργικότητας, καθώς αυτοί οι τομείς βρίσκονται εκτός του πεδίου εφαρμογής της παρούσας αξιολόγησης επιδόσεων.

Κεφάλαιο 2ο: Βιβλιογραφική ανασκόπηση

2.1 Η σημασία της απόδοσης ενός συστήματος

Η απόδοση ενός συστήματος λογισμικού είναι ένας από τους πιο κρίσιμους παράγοντες που επηρεάζουν την εμπειρία των χρηστών και τη συνολική επιτυχία μιας εφαρμογής. Ένα σύστημα με υψηλή απόδοση είναι σε θέση να διαχειρίζεται αποτελεσματικά μεγάλα φορτία χρηστών, να αποκρίνεται άμεσα στα αιτήματα και να χρησιμοποιεί αποδοτικά τους διαθέσιμους πόρους. Αντίθετα, ένα σύστημα με χαμηλή απόδοση μπορεί να οδηγήσει σε καθυστερήσεις, σφάλματα και υποβάθμιση της εμπειρίας των χρηστών, γεγονός που μπορεί να έχει σοβαρές επιπτώσεις στην αποδοχή και τη βιωσιμότητα της εφαρμογής.

Η σημασία της απόδοσης ενός συστήματος γίνεται ακόμη πιο εμφανής σε σύγχρονα διαδικτυακά και επιχειρησιακά περιβάλλοντα, όπου οι απαιτήσεις για ταχύτητα, διαθεσιμότητα και αξιοπιστία είναι αυξημένες. Ειδικά σε εφαρμογές που βασίζονται σε cloud και κατανεμημένες αρχιτεκτονικές, η σωστή διαχείριση της απόδοσης είναι απαραίτητη για να διασφαλιστεί η απρόσκοπτη λειτουργία του συστήματος ακόμα και σε περιόδους υψηλού φόρτου.

Οι βασικοί παράγοντες που επηρεάζουν την απόδοση ενός συστήματος όπως αναφέραμε και στο προηγούμενο κεφάλαιο περιλαμβάνουν:

- **Χρόνος απόκρισης (Response time):**

Ο χρόνος που απαιτείται για να ανταποκριθεί το σύστημα σε ένα αίτημα χρήστη.

- **Ρυθμαπόδοση (Throughput):**

Ο αριθμός των συναλλαγών ή αιτημάτων που μπορεί να διαχειριστεί το σύστημα ανά μονάδα χρόνου.

- **Κατανάλωση πόρων (Resource utilization):**

Ο βαθμός στον οποίο το σύστημα χρησιμοποιεί επεξεργαστική ισχύ, μνήμη και άλλους πόρους.

- **Σταθερότητα και κλιμάκωση (Scalability):**

Η ικανότητα του συστήματος να προσαρμόζεται σε αυξανόμενο φορτίο χρηστών ή δεδομένων χωρίς σημαντική υποβάθμιση της απόδοσης.

- **Διαθεσιμότητα (Availability):**

Το ποσοστό του χρόνου που το σύστημα είναι διαθέσιμο και λειτουργεί σωστά.

Για να διασφαλιστεί η υψηλή απόδοση, απαιτείται συστηματικός έλεγχος, ανάλυση και βελτιστοποίηση του συστήματος. Οι μέθοδοι ελέγχου απόδοσης περιλαμβάνουν τεχνικές όπως η δοκιμή φόρτου **(load testing)**, η δοκιμή υπερφόρτωσης **(stress testing)** και η δοκιμή κλιμάκωσης **(scalability testing)**, οι οποίες βοηθούν στον εντοπισμό πιθανών σημείων συμφόρησης και βελτίωσης.

2.1.1 Ανάλυση εργαλείων ελέγχου απόδοσης

Ο έλεγχος απόδοσης ενός συστήματος απαιτεί τη χρήση εξειδικευμένων εργαλείων που επιτρέπουν την προσομοίωση πραγματικών συνθηκών λειτουργίας, τη συλλογή μετρήσεων και την ανάλυση κρίσιμων δεικτών απόδοσης. Τα εργαλεία αυτά βοηθούν στον εντοπισμό σημείων συμφόρησης, στην κατανόηση της συμπεριφοράς του συστήματος υπό διαφορετικά επίπεδα φόρτου και στη βελτιστοποίηση της συνολικής εμπειρίας του χρήστη.

Στην παρούσα ενότητα, θα αναλύσουμε τα πιο διαδεδομένα εργαλεία δοκιμών απόδοσης, όπως το Apache JMeter, το LoadRunner, το Gatling και το BlazeMeter. Κάθε εργαλείο διαθέτει διαφορετικές δυνατότητες και είναι κατάλληλο για συγκεκριμένες ανάγκες, όπως δοκιμές φόρτισης, αντοχής, κλιμάκωσης και εκρηκτικού φορτίου. Μέσα από αυτήν την ανάλυση, θα παρουσιαστούν οι βασικές λειτουργίες τους, τα πλεονεκτήματα που προσφέρουν και οι περιπτώσεις χρήσης τους, ώστε να κατανοηθεί πότε και πώς πρέπει να επιλέγεται το κατάλληλο εργαλείο για κάθε τύπο δοκιμής.

Apache JMeter

Το Apache JMeter είναι ένα εργαλείο ανοιχτού κώδικα για τη δοκιμή απόδοσης, το οποίο αναπτύχθηκε από το Apache Software Foundation. Χρησιμοποιείται κυρίως για τη μέτρηση της απόδοσης και της κλιμάκωσης web εφαρμογών, αλλά μπορεί να εφαρμοστεί και σε άλλες υπηρεσίες όπως βάσεις δεδομένων, FTP servers, API endpoints και SOAP/RESTful υπηρεσίες. Πρόκειται για ένα ισχυρό εργαλείο που επιτρέπει στους χρήστες να προσομοιώνουν μεγάλο αριθμό εικονικών χρηστών, ώστε να αναλύσουν τη συμπεριφορά του συστήματος υπό διάφορες συνθήκες φόρτου.

Λειτουργίες:

Το Apache JMeter προσφέρει πληθώρα δυνατοτήτων που το καθιστούν χρήσιμο για διαφορετικά σενάρια δοκιμών απόδοσης:

- **Δοκιμές Φόρτου (Load Testing):**

  Επιτρέπει την προσομοίωση εκατοντάδων ή χιλιάδων χρηστών για να ελέγξει πώς ανταποκρίνεται το σύστημα σε αυξανόμενο φόρτο.

- **Δοκιμές Υπερφόρτωσης (Stress Testing):**

  Αναλύει τα όρια αντοχής ενός συστήματος και πότε αρχίζει να παρουσιάζει προβλήματα.

- **Δοκιμές Κλιμάκωσης (Scalability Testing):**

  Εξετάζει αν το σύστημα μπορεί να διαχειριστεί μεγαλύτερο αριθμό χρηστών χωρίς πτώση στην απόδοση.

- **Δοκιμές Αντοχής (Endurance Testing):**

  Προσομοιώνει μεγάλο φόρτο για μεγάλο χρονικό διάστημα, για να ανιχνεύσει προβλήματα όπως διαρροές μνήμης.

- **Δοκιμές Όγκου (Volume Testing):**

  Ελέγχει την απόδοση ενός συστήματος όταν διαχειρίζεται μεγάλες ποσότητες δεδομένων.

- **Υποστήριξη Πολλαπλών Πρωτοκόλλων:**

  Το JMeter μπορεί να δοκιμάσει υπηρεσίες που χρησιμοποιούν HTTP, HTTPS, FTP, JDBC (για βάσεις δεδομένων), SOAP, REST APIs, JMS **(Java Messaging Service)**, SMTP και άλλα.

- **Δυνατότητα Αυτοματοποιημένων Δοκιμών:**

  Μπορεί να συνδυαστεί με εργαλεία όπως το Jenkins και το Selenium για αυτοματοποιημένες δοκιμές απόδοσης σε περιβάλλον Continuous Integration **(CI/CD)**.

- **Γραφική και CLI λειτουργία:**

  Παρέχει τόσο περιβάλλον χρήστη **(GUI)** για τη δημιουργία δοκιμών όσο και Command Line Interface **(CLI)** για εκτέλεση δοκιμών χωρίς γραφικό περιβάλλον, χρήσιμο σε servers και αυτοματοποιημένες ροές εργασίας.

Πλεονεκτήματα:

- **Δωρεάν και ανοιχτού κώδικα:**

  Δεν απαιτείται αγορά άδειας χρήσης, ενώ η ενεργή κοινότητα συμβάλλει συνεχώς στη βελτίωσή του.

- **Επεκτασιμότητα:**

  Υποστηρίζει plugins που μπορούν να προσθέσουν νέες λειτουργίες, βελτιώνοντας την ανάλυση δεδομένων και την καταγραφή των δοκιμών.

- **Δυνατότητα παραμετροποίησης:**

  Οι χρήστες μπορούν να προσαρμόσουν τις δοκιμές, δημιουργώντας σενάρια με μεταβλητές και δυναμικά δεδομένα.

- **Εύκολη ανάλυση δεδομένων:**

  Προσφέρει αναφορές σε μορφή γραφημάτων, πινάκων και αρχείων logs για να αξιολογηθεί η απόδοση του συστήματος.

- **Υποστήριξη κατανεμημένων δοκιμών:**

  Μπορεί να εκτελεστεί σε πολλαπλά μηχανήματα ταυτόχρονα, επιτρέποντας την προσομοίωση ακόμα μεγαλύτερου φόρτου χρηστών.

LoadRunner

Το LoadRunner της Micro Focus είναι ένα ισχυρό εργαλείο δοκιμών απόδοσης που επιτρέπει τη μέτρηση, την προσομοίωση και την ανάλυση της συμπεριφοράς εφαρμογών υπό διαφορετικές συνθήκες φόρτου. Χρησιμοποιείται από μεγάλες επιχειρήσεις και οργανισμούς για την αξιολόγηση της απόδοσης πολύπλοκων πληροφοριακών συστημάτων, όπως διαδικτυακές εφαρμογές, ERP συστήματα, τραπεζικές υπηρεσίες και cloud-based εφαρμογές.

Λειτουργίες:

Το LoadRunner προσφέρει προηγμένες δυνατότητες για τη δοκιμή απόδοσης, όπως:

- **Virtual User Generator (VuGen):**

Δημιουργεί σενάρια που προσομοιώνουν τη συμπεριφορά χιλιάδων ταυτόχρονων χρηστών.

- **Controller:**

Διαχειρίζεται και συντονίζει τις δοκιμές φορτίου σε διαφορετικές διαμορφώσεις.

- **Analysis Module:**

Συλλέγει και αναλύει δεδομένα επιδόσεων, επιτρέποντας την αναγνώριση σημείων συμφόρησης.

- **Υποστήριξη Πολλαπλών Πρωτοκόλλων:**

Συμβατό με HTTP/HTTPS, SOAP, Citrix, Oracle, SAP, και άλλα.

- **Δυνατότητα Cloud Testing:**

Επιτρέπει τη δοκιμή εφαρμογών σε υποδομές cloud για μεγαλύτερη ευελιξία.

Πλεονεκτήματα:

- **Υψηλή ακρίβεια μετρήσεων:**

Επιτρέπει τον ακριβή εντοπισμό των σημείων συμφόρησης.

- **Ευρεία υποστήριξη τεχνολογιών:**

Συμβατό με πολλαπλά πρωτόκολλα και περιβάλλοντα.

- **Ενσωμάτωση με DevOps:**

Δυνατότητα αυτοματοποίησης δοκιμών μέσα σε CI/CD pipelines.

- **Καταγραφή πραγματικών συνθηκών χρήσης:**

Προσομοίωση πραγματικών σεναρίων με υψηλό όγκο χρηστών.

Gatling

Το Gatling είναι ένα σύγχρονο εργαλείο δοκιμών απόδοσης ανοιχτού κώδικα, ειδικά σχεδιασμένο για υψηλής κλίμακας και ρεαλιστικές δοκιμές φόρτου. Χρησιμοποιείται ευρέως σε stress testing και scalability testing, καθώς

Πλεονεκτήματα για Υψηλές Επιδόσεις:

- **Υψηλή επεκτασιμότητα:**

Λόγω του ασύγχρονου μοντέλου, μπορεί να προσομοιώσει μεγάλους όγκους χρηστών χωρίς υψηλή κατανάλωση μνήμης.

- **Γρήγορη εκτέλεση και ανάλυση αποτελεσμάτων:**

Παρέχει λεπτομερείς αναφορές με οπτικοποιημένα δεδομένα για εύκολη κατανόηση της απόδοσης του συστήματος.

- **Ευελιξία και προσαρμογή:**

Δίνει τη δυνατότητα δημιουργίας πολύπλοκων σεναρίων μέσω Scala-based DSL **(Domain-Specific Language)**.

- **Αυτόματη καταγραφή και αναπαραγωγή δοκιμών:**

Μπορεί να καταγράφει αιτήσεις HTTP/S και να δημιουργεί σενάρια αυτόματα.

Χρήση σε Stress Testing και Scalability Testing:

Το Gatling είναι ιδανικό για stress testing, όπου το σύστημα υποβάλλεται σε υπερβολικό φόρτο για να εντοπιστούν τα όρια αντοχής του. Επίσης, χρησιμοποιείται σε scalability testing, όπου μετράται αν μια εφαρμογή μπορεί να διαχειριστεί σταδιακή αύξηση των χρηστών χωρίς απότομη μείωση της απόδοσής της.

BlazeMeter

Το BlazeMeter είναι ένα cloud-based εργαλείο δοκιμών απόδοσης που επιτρέπει την εκτέλεση δοκιμών φόρτου, stress testing και scalability testing χωρίς την ανάγκη για τοπική εγκατάσταση ή διαχείριση υποδομών. Είναι σχεδιασμένο για να παρέχει υψηλή ευελιξία και κλιμάκωση, επιτρέποντας στους χρήστες να προσομοιώνουν χιλιάδες ή και εκατομμύρια ταυτόχρονους χρήστες από διάφορες γεωγραφικές τοποθεσίες, προκειμένου να ελέγξουν την απόδοση ενός συστήματος υπό ρεαλιστικές συνθήκες δικτύου.

Cloud-Based Προσέγγιση & Ευκολία Κλιμάκωσης:

- **Εκτέλεση δοκιμών από το cloud:**

  Το BlazeMeter επιτρέπει την εκτέλεση δοκιμών χωρίς περιορισμούς τοπικής υποδομής, κάνοντας χρήση διακομιστών cloud για τη δημιουργία εικονικών χρηστών.

- **Συμβατότητα με υπάρχοντα εργαλεία:**

  Υποστηρίζει τη χρήση Apache JMeter, Gatling, Selenium και άλλων εργαλείων δοκιμών, καθιστώντας το ιδανικό για επιχειρήσεις που επιθυμούν ευέλικτη ενσωμάτωση.

- **Δυναμική κλιμάκωση:**

  Οι δοκιμές μπορούν να κλιμακωθούν εύκολα από εκατοντάδες σε εκατομμύρια ταυτόχρονους χρήστες ανάλογα με τις ανάγκες του συστήματος.

- **Ρεαλιστική προσομοίωση χρηστών:**

  Επιτρέπει την εκτέλεση δοκιμών από διαφορετικές γεωγραφικές περιοχές, ελέγχοντας την απόδοση του συστήματος σε διεθνές επίπεδο.

- **Προηγμένες αναφορές και ανάλυση:**

  Παρέχει διαδραστικά dashboards με λεπτομερή στατιστικά, βοηθώντας στην κατανόηση των σημείων συμφόρησης της εφαρμογής.

2.2 Προκλήσεις στον έλεγχο επιδόσεων

Ο έλεγχος επιδόσεων είναι κρίσιμος για τη διασφάλιση της αποδοτικότητας και της σταθερότητας ενός συστήματος, αλλά συνοδεύεται από σημαντικές προκλήσεις. Οι προκλήσεις αυτές μπορεί να σχετίζονται με τεχνικούς περιορισμούς, τη φύση των κατανεμημένων συστημάτων, την έλλειψη τυποποίησης ή ακόμα και την υποεκτίμηση του ρόλου της εμπειρίας χρήστη. Παρακάτω αναλύονται οι βασικότερες προκλήσεις που προκύπτουν κατά τη διαδικασία δοκιμών επιδόσεων.

2.2.1 Τεχνικές προκλήσεις

Ο έλεγχος επιδόσεων είναι μια απαιτητική διαδικασία που απαιτεί υψηλή υπολογιστική ισχύ, σταθερές δικτυακές υποδομές και τη δυνατότητα προσομοίωσης ρεαλιστικών σεναρίων φόρτου. Ωστόσο, πολλές επιχειρήσεις και ερευνητικά κέντρα αντιμετωπίζουν τεχνικούς και οικονομικούς περιορισμούς, οι οποίοι επηρεάζουν την ποιότητα και την αξιοπιστία των δοκιμών επιδόσεων. Αυτοί οι περιορισμοί επηρεάζουν τόσο το εύρος όσο και την ακρίβεια των αποτελεσμάτων, οδηγώντας σε πιθανά σφάλματα αξιολόγησης.

Υψηλό κόστος υποδομής

Οι δοκιμές μεγάλης κλίμακας απαιτούν ισχυρό hardware, λογισμικό προσομοίωσης και υποδομές cloud, κάτι που αυξάνει σημαντικά το λειτουργικό κόστος. Ιδιαίτερα στις περιπτώσεις όπου απαιτείται stress testing ή endurance testing, οι οργανισμοί χρειάζονται:

- **Ισχυρούς servers ή cloud instances** με δυνατότητα δυναμικής κλιμάκωσης.
- **Μεγάλες βάσεις δεδομένων** για τη δημιουργία ρεαλιστικών συνθηκών φόρτου.
- **Εξειδικευμένα εργαλεία και λογισμικό δοκιμών** που συχνά συνοδεύονται από υψηλά κόστη αδειοδότησης (π.χ. LoadRunner, BlazeMeter).

Ακόμα και σε περιπτώσεις χρήσης cloud-based υποδομών, οι δυναμικές χρεώσεις των παρόχων **(AWS, Google Cloud, Azure)** για μεγάλες περιόδους φόρτου μπορούν να καταστήσουν τις εκτεταμένες δοκιμές οικονομικά μη βιώσιμες.

Προσομοίωση πραγματικών συνθηκών

Για την αποτελεσματική αξιολόγηση ενός συστήματος, απαιτείται η προσομοίωση ρεαλιστικών φορτίων, κάτι που είναι τεχνικά δύσκολο λόγω:

- **Μη προβλέψιμων αιχμών ζήτησης:**

  Οι πραγματικοί χρήστες δεν δημιουργούν σταθερό φόρτο, αλλά αντίθετα ακολουθούν ακανόνιστα μοτίβα χρήσης. Για παράδειγμα, ένα e-commerce σύστημα μπορεί να δεχθεί χιλιάδες αιτήσεις σε δευτερόλεπτα κατά τη διάρκεια μιας προωθητικής καμπάνιας, κάτι που είναι δύσκολο να προσομοιωθεί με ακρίβεια.

- **Μεταβαλλόμενες δικτυακές συνθήκες:**

  Οι δοκιμές πρέπει να λαμβάνουν υπόψη latency variations, packet loss και bandwidth limitations, τα οποία είναι δύσκολο να αναπαραχθούν εργαστηριακά.

- **Ετερογένεια συσκευών και browsers:**

  Η απόδοση του συστήματος μπορεί να διαφέρει ανάλογα με την πλατφόρμα (π.χ. mobile vs desktop, διαφορετικοί browsers), κάτι που απαιτεί εκτεταμένες δοκιμές σε πολλαπλά περιβάλλοντα.

Περιορισμένη διαθεσιμότητα hardware

Ακόμη και αν υπάρχει το απαραίτητο λογισμικό, ο έλεγχος επιδόσεων συχνά απαιτεί μεγάλη υπολογιστική ισχύ. Ωστόσο, οργανισμοί που διαθέτουν περιορισμένους φυσικούς ή cloud-based πόρους δυσκολεύονται να εκτελέσουν μεγάλες δοκιμές επιδόσεων. Οι κύριοι περιορισμοί περιλαμβάνουν:

- **Ανεπάρκεια server nodes** για εκτέλεση κατανεμημένων δοκιμών.
- **Περιορισμένη μνήμη RAM και CPU resources,** που μπορεί να επηρεάσουν τη συμπεριφορά των ίδιων των δοκιμών.
- **Ανεπάρκεια εξειδικευμένων υποδομών,** όπως dedicated load balancers και traffic generators.

Συχνά, λόγω αυτών των περιορισμών, οι επιχειρήσεις καταφεύγουν σε μερικώς αντιπροσωπευτικές δοκιμές, γεγονός που μπορεί να οδηγήσει σε εσφαλμένα συμπεράσματα σχετικά με την πραγματική απόδοση του συστήματος [17].

2.2.2 Μεταβλητότητα περιβαλλόντων δοκιμής

Η απόδοση ενός συστήματος δεν είναι στατική αλλά εξαρτάται από πολλούς παράγοντες που σχετίζονται με το περιβάλλον εκτέλεσής του. Η μεταβλητότητα των περιβαλλόντων δοκιμής αποτελεί μια από τις σημαντικότερες προκλήσεις στις δοκιμές επιδόσεων, καθώς η αναπαραγωγή ακριβών και αξιόπιστων αποτελεσμάτων απαιτεί σταθερές συνθήκες που συχνά δεν είναι εφικτές. Οι διαφορές μεταξύ του δοκιμαστικού και του production περιβάλλοντος μπορούν να οδηγήσουν σε σημαντικές αποκλίσεις στα αποτελέσματα, καθιστώντας δύσκολη την πρόβλεψη της πραγματικής συμπεριφοράς της εφαρμογής κατά την κανονική της λειτουργία.

Ένα από τα κύρια προβλήματα είναι η αναντιστοιχία μεταξύ του περιβάλλοντος δοκιμών και του production περιβάλλοντος. Τα δοκιμαστικά συστήματα συχνά διαθέτουν λιγότερους πόρους, διαφορετικές ρυθμίσεις ασφαλείας και λιγότερη δικτυακή συμφόρηση, με αποτέλεσμα οι μετρήσεις απόδοσης να είναι αισθητά διαφοροποιημένες. Για παράδειγμα, μια βάση δεδομένων που λειτουργεί σε περιβάλλον δοκιμών μπορεί να παρουσιάζει εξαιρετική απόδοση λόγω του χαμηλού όγκου δεδομένων και των μειωμένων συναλλαγών, ενώ στο production περιβάλλον η υψηλή ταυτόχρονη πρόσβαση

χρηστών και τα μεγάλα φορτία μπορεί να δημιουργήσουν σημεία συμφόρησης που δεν είχαν προβλεφθεί.

Επιπλέον, η ασυνέπεια των αποτελεσμάτων μεταξύ διαφορετικών εκτελέσεων των δοκιμών αποτελεί έναν ακόμη κρίσιμο παράγοντα που μειώνει την αξιοπιστία των δοκιμών επιδόσεων. Οι παράμετροι που επηρεάζουν τη συμπεριφορά του συστήματος, όπως η χρήση της CPU, η κατανάλωση μνήμης, οι I/O διεργασίες και οι συνθήκες του δικτύου, δεν είναι σταθερές και μπορούν να μεταβάλλονται δυναμικά. Για παράδειγμα, σε ένα περιβάλλον cloud, όπου οι πόροι μοιράζονται μεταξύ πολλαπλών εικονικών μηχανών (VMs) ή κοντέινερ, η διακύμανση της απόδοσης μπορεί να είναι έντονη λόγω της συνύπαρξης διαφορετικών εφαρμογών στους ίδιους φυσικούς διακομιστές. Σε ένα τέτοιο περιβάλλον, δύο διαδοχικές δοκιμές μπορεί να εμφανίζουν διαφορετικά αποτελέσματα, γεγονός που δυσχεραίνει την εξαγωγή ασφαλών συμπερασμάτων.

Ένα ακόμη σημαντικό πρόβλημα είναι η αδυναμία προσομοίωσης της πραγματικής συμπεριφοράς των χρηστών κατά τη διάρκεια των δοκιμών επιδόσεων. Τα περισσότερα εργαλεία δοκιμών βασίζονται στη δημιουργία συνθετικών φορτίων, δηλαδή προγραμματισμένων αιτημάτων προς την εφαρμογή με συγκεκριμένα μοτίβα πρόσβασης. Ωστόσο, οι πραγματικοί χρήστες δεν ακολουθούν αυστηρά προκαθορισμένα σενάρια, αλλά αντίθετα αλληλεπιδρούν με το σύστημα με απρόβλεπτους τρόπους, επηρεάζοντας παραμέτρους όπως ο ρυθμός των αιτημάτων **(requests per second)**, ο χρόνος αδράνειας μεταξύ ενεργειών και οι παράλληλες εκτελέσεις διαφορετικών λειτουργιών. Για παράδειγμα, σε ένα σύστημα ηλεκτρονικού εμπορίου, η συμπεριφορά ενός χρήστη που περιηγείται στα προϊόντα διαφέρει σημαντικά από αυτήν ενός άλλου χρήστη που προσθέτει συνεχώς αντικείμενα στο καλάθι και προχωρά στην αγορά. Η δυσκολία προσομοίωσης αυτών των σεναρίων μπορεί να οδηγήσει σε λανθασμένες εκτιμήσεις σχετικά με την απόδοση του συστήματος σε πραγματικές συνθήκες.

Για την αντιμετώπιση αυτών των προκλήσεων, οι οργανισμοί πρέπει να υιοθετήσουν προηγμένες στρατηγικές δοκιμών επιδόσεων. Μια λύση είναι η χρήση περιβαλλόντων δοκιμών που είναι όσο το δυνατόν πιο κοντά στο production περιβάλλον, τόσο σε επίπεδο υποδομής όσο και σε ρυθμίσεις διαμόρφωσης. Η χρήση containerization τεχνολογιών, όπως το Docker και το Kubernetes, μπορεί να προσφέρει μια πιο συνεπή αναπαραγωγή των συνθηκών εκτέλεσης μεταξύ διαφορετικών περιβαλλόντων. Επιπλέον, η ενσωμάτωση A/B testing και real-user monitoring (RUM) τεχνικών επιτρέπει την ανάλυση των πραγματικών συνθηκών χρήσης και την προσαρμογή των δοκιμών ώστε να αντανακλούν ρεαλιστικά σενάρια.

Συμπερασματικά, η μεταβλητότητα των περιβαλλόντων δοκιμής αποτελεί έναν από τους μεγαλύτερους παράγοντες αβεβαιότητας στις δοκιμές επιδόσεων. Η εξασφάλιση αξιόπιστων και συγκρίσιμων αποτελεσμάτων απαιτεί τη λήψη προληπτικών μέτρων, όπως η χρήση σταθερών περιβαλλόντων δοκιμής, η υιοθέτηση τεχνολογιών απομόνωσης πόρων και η ανάλυση πραγματικών δεδομένων χρήσης. Μόνο με αυτές τις προσεγγίσεις μπορούν οι οργανισμοί να αποκτήσουν σαφή εικόνα της απόδοσης των συστημάτων τους και να λάβουν τεκμηριωμένες αποφάσεις για τη βελτιστοποίηση της εμπειρίας των χρηστών.

2.2.3 Έλλειψη τυποποιημένων μεθοδολογιών

Η απουσία ενός καθολικά αποδεκτού προτύπου για τη διενέργεια δοκιμών επιδόσεων αποτελεί μια από τις σημαντικότερες προκλήσεις στον τομέα της διασφάλισης ποιότητας λογισμικού. Οι οργανισμοί υιοθετούν διαφορετικές προσεγγίσεις, εργαλεία και μεθοδολογίες, γεγονός που οδηγεί σε μη συγκρίσιμα αποτελέσματα και σε περιορισμένη δυνατότητα αναπαραγωγής των δοκιμών. Η έλλειψη αυτής της τυποποίησης επηρεάζει τη συνολική αξιοπιστία των δοκιμών επιδόσεων, καθιστώντας

δύσκολη τη δημιουργία βέλτιστων πρακτικών που μπορούν να εφαρμοστούν σε διαφορετικά περιβάλλοντα και τεχνολογίες.

Ένα από τα βασικά προβλήματα είναι η έλλειψη ενός ενιαίου πλαισίου για τη διεξαγωγή των δοκιμών. Οι εταιρείες και οι ερευνητικές κοινότητες χρησιμοποιούν διαφορετικές στρατηγικές, με αποτέλεσμα οι ίδιες δοκιμές να παράγουν διαφορετικά αποτελέσματα ανάλογα με το περιβάλλον, τη μεθοδολογία και τα εργαλεία που χρησιμοποιούνται. Για παράδειγμα, ενώ κάποιοι οργανισμοί υιοθετούν ένα σενάριο βάσει σταθερού φορτίου **(constant load testing)**, άλλοι προτιμούν τη σταδιακή αύξηση της επιβάρυνσης **(ramp-up testing)**, με αποτέλεσμα τα αποτελέσματα να μην είναι άμεσα συγκρίσιμα. Αυτή η έλλειψη ομοιογένειας δημιουργεί δυσκολίες στη συνεργασία μεταξύ ομάδων ανάπτυξης και διασφάλισης ποιότητας, καθώς και στην αξιολόγηση της απόδοσης των συστημάτων σε διαφορετικά περιβάλλοντα.

Επιπλέον, υπάρχει σημαντική ασυνέπεια στις μετρικές απόδοσης που χρησιμοποιούνται. Διαφορετικές εταιρείες και ομάδες ανάπτυξης επιλέγουν να αξιολογήσουν την απόδοση των συστημάτων τους με βάση ξεχωριστά Key Performance Indicators (KPIs), όπως ο χρόνος απόκρισης, ο ρυθμός συναλλαγών ανά δευτερόλεπτο (TPS) ή η χρήση πόρων (CPU, RAM, δίκτυο). Ωστόσο, η απουσία ενός καθορισμένου συνόλου κοινών μετρικών οδηγεί σε προβλήματα συγκρισιμότητας μεταξύ διαφορετικών εφαρμογών και συστημάτων. Για παράδειγμα, ένας οργανισμός μπορεί να θεωρεί επιτυχημένη μια δοκιμή εάν το σύστημα διατηρεί χρόνο απόκρισης κάτω από 200ms, ενώ ένας άλλος οργανισμός μπορεί να δίνει μεγαλύτερη σημασία στον αριθμό των ταυτόχρονων χρηστών που μπορεί να υποστηρίξει η εφαρμογή. Η διαφορά στις μετρικές καθιστά δύσκολη την ανάπτυξη γενικών benchmarks που μπορούν να χρησιμοποιηθούν για τη σύγκριση της απόδοσης μεταξύ διαφορετικών συστημάτων.

Μια ακόμα σημαντική πρόκληση είναι η έλλειψη κατευθυντήριων γραμμών για τις νέες τεχνολογίες. Η ταχεία εξέλιξη του λογισμικού και των υποδομών, όπως το serverless computing, τα microservices και οι cloud-native αρχιτεκτονικές, έχει καταστήσει δυσκολότερη τη δημιουργία σταθερών και δοκιμασμένων μεθοδολογιών για τις δοκιμές επιδόσεων. Για παράδειγμα, σε ένα serverless περιβάλλον, η απόδοση μπορεί να εξαρτάται από παράγοντες όπως η δυναμική κατανομή των πόρων από τον πάροχο cloud, γεγονός που καθιστά πιο περίπλοκο τον προσδιορισμό ενός καθολικού πλαισίου δοκιμών. Σε αυτά τα περιβάλλοντα, οι παραδοσιακές τεχνικές δοκιμών επιδόσεων που εφαρμόζονταν σε μονολιθικές εφαρμογές δεν μπορούν να αποδώσουν με την ίδια ακρίβεια.

Για την αντιμετώπιση αυτών των προκλήσεων, υπάρχει ανάγκη για την ανάπτυξη διεθνών προτύπων και βέλτιστων πρακτικών στον έλεγχο επιδόσεων. Ορισμένοι οργανισμοί, όπως το IEEE και το ISO, έχουν προτείνει πρότυπα για τον έλεγχο της ποιότητας του λογισμικού, αλλά οι κατευθυντήριες γραμμές τους δεν εφαρμόζονται αυστηρά στη βιομηχανία λόγω της πολυπλοκότητας των σύγχρονων συστημάτων. Η θέσπιση τυποποιημένων frameworks και benchmarking εργαλείων, όπως το SPEC **(Standard Performance Evaluation Corporation)**, μπορεί να συμβάλει στη βελτίωση της αξιοπιστίας των δοκιμών επιδόσεων. Παράλληλα, οι εταιρείες μπορούν να συνεργαστούν στη δημιουργία κοινών best practices και guidelines που θα μπορούν να εφαρμοστούν ανεξαρτήτως τεχνολογικής πλατφόρμας.

Συμπερασματικά, η έλλειψη τυποποιημένων μεθοδολογιών στις δοκιμές επιδόσεων αποτελεί μια κρίσιμη πρόκληση που οδηγεί σε ασυνεπή αποτελέσματα και μειωμένη αξιοπιστία των δοκιμών. Η ανάπτυξη ενιαίων προσεγγίσεων, η καθιέρωση καθολικά αποδεκτών KPIs και η δημιουργία προτύπων για νέες τεχνολογίες είναι απαραίτητα βήματα για την αποτελεσματική και αξιόπιστη αξιολόγηση των συστημάτων. Μέχρι να υπάρξει ευρεία αποδοχή και εφαρμογή τέτοιων προτύπων, οι οργανισμοί πρέπει να υιοθετήσουν ευέλικτες μεθοδολογίες, να χρησιμοποιούν benchmarking εργαλεία και να διασφαλίζουν τη διαφάνεια στις διαδικασίες ελέγχου επιδόσεων

2.2.4 Περιορισμένη εστίαση στην εμπειρία χρήστη

Παρόλο που η απόδοση ενός συστήματος αποτελεί κρίσιμο παράγοντα για την επιτυχία του, οι περισσότερες δοκιμές επιδόσεων εστιάζουν αποκλειστικά σε τεχνικά μετρικά, όπως ο χρόνος απόκρισης **(response time)**, ο ρυθμός αιτημάτων ανά δευτερόλεπτο **(throughput)** και η χρήση πόρων (CPU, μνήμη, bandwidth). Ωστόσο, αυτά τα μετρικά δεν αποτυπώνουν πλήρως την εμπειρία του τελικού χρήστη, η οποία είναι καθοριστική για την αποδοχή και τη βιωσιμότητα μιας εφαρμογής. Η περιορισμένη εστίαση στην εμπειρία χρήστη **(UX performance)** μπορεί να οδηγήσει σε παραπλανητικά αποτελέσματα, όπου ένα σύστημα εμφανίζεται τεχνικά αποδοτικό, αλλά στην πράξη δεν παρέχει μια ομαλή και ευχάριστη εμπειρία στον χρήστη. Ένα από τα μεγαλύτερα ζητήματα είναι η απουσία UX-centric μετρήσεων στις παραδοσιακές δοκιμές επιδόσεων. Τα εργαλεία δοκιμών, όπως το Apache JMeter ή το LoadRunner, μετρούν κυρίως την απόκριση του διακομιστή **(server-side performance)**, παραβλέποντας κρίσιμες πτυχές της εμπειρίας του χρήστη, όπως η ταχύτητα απόδοσης του UI **(frontend performance)**, η διαδραστικότητα και η οπτική σταθερότητα της εφαρμογής. Για παράδειγμα, το Largest Contentful Paint (LCP), που μετρά τον χρόνο που χρειάζεται για να φορτωθεί το κύριο περιεχόμενο μιας σελίδας, ή το First Input Delay (FID), που καταγράφει τον χρόνο απόκρισης του συστήματος στη πρώτη ενέργεια του χρήστη, είναι βασικοί δείκτες που αποτυπώνουν καλύτερα την εμπειρία χρήστη. Παρόλα αυτά, σπάνια περιλαμβάνονται σε δοκιμές επιδόσεων, οδηγώντας σε ελλιπή αξιολόγηση της πραγματικής αίσθησης ταχύτητας που λαμβάνει ο χρήστης.

Επιπλέον, οι δοκιμές επιδόσεων συχνά πραγματοποιούνται χωρίς να λαμβάνονται υπόψη τα UI interactions και η συμπεριφορά των χρηστών, γεγονός που καθιστά δύσκολη την κατανόηση του πώς η απόδοση επηρεάζει την εμπειρία χρήστη. Σε ένα πραγματικό σενάριο, ένας χρήστης δεν εκτελεί αιτήσεις HTTP ανεξάρτητα, αλλά αλληλεπιδρά με το UI μέσω σύνθετων ακολουθιών ενεργειών, όπως το άνοιγμα μενού, η συμπλήρωση φορμών και η κύλιση σε δυναμικό περιεχόμενο. Αντίστοιχα, οι δοκιμές συχνά γίνονται σε συνθήκες ιδανικού δικτύου, χωρίς να λαμβάνουν υπόψη την επίδραση της μεταβλητότητας της σύνδεσης (π.χ. latency spikes, packet loss) στην εμπειρία του χρήστη, ειδικά σε mobile περιβάλλοντα. Η απουσία τέτοιων στοιχείων από τις δοκιμές καθιστά δύσκολο τον εντοπισμό πραγματικών προβλημάτων απόδοσης που μπορεί να αντιμετωπίσουν οι χρήστες σε καθημερινή χρήση

Μια ακόμα σημαντική παράμετρος είναι η αντίληψη της ταχύτητας από τον χρήστη, η οποία δεν εξαρτάται αποκλειστικά από τα τυπικά μετρικά απόδοσης. Για παράδειγμα, μια εφαρμογή μπορεί να έχει πολύ χαμηλό server-side latency, αλλά εάν το UI της είναι κακώς σχεδιασμένο (π.χ. αν δεν εμφανίζει loading indicators ή σταδιακά φορτωμένο περιεχόμενο), ο χρήστης μπορεί να την αντιλαμβάνεται ως αργή. Αυτό το φαινόμενο είναι ιδιαίτερα εμφανές σε web εφαρμογές με client-side rendering, όπου το backend επεξεργάζεται γρήγορα τα αιτήματα, αλλά η απόδοση στο frontend καθυστερεί λόγω βαριάς JavaScript επεξεργασίας. Στο πλαίσιο αυτό, η χρήση εργαλείων όπως το Google Lighthouse, το WebPageTest και το Chrome DevTools μπορεί να προσφέρει πολύτιμες μετρήσεις που αποτυπώνουν την εμπειρία του χρήστη, πέρα από τις παραδοσιακές μετρήσεις server performance.

Για να ξεπεραστούν αυτά τα προβλήματα, οι οργανισμοί θα πρέπει να επεκτείνουν τις δοκιμές επιδόσεων ώστε να συμπεριλάβουν end-to-end μετρήσεις που ενσωματώνουν τόσο backend όσο και frontend επιδόσεις. Οι δοκιμές θα πρέπει να προσομοιώνουν πραγματικά σενάρια χρήσης και να ενσωματώνουν στοιχεία όπως το rendering time, η απόκριση στα user interactions, και η συμπεριφορά του συστήματος σε περιβάλλοντα με υψηλό latency ή χαμηλό bandwidth. Επιπλέον, η χρήση real user monitoring (RUM) εργαλείων μπορεί να παρέχει πολύτιμα insights για το πώς οι χρήστες βιώνουν την απόδοση της εφαρμογής σε πραγματικό χρόνο.

2.3 Βελτίωση απόοσης συστήματος

Η απόδοση ενός συστήματος είναι κρίσιμη για την αποτελεσματικότητα και την εμπειρία του χρήστη, ειδικά σε εφαρμογές με αυξημένες απαιτήσεις σε κλίμακα και ταχύτητα. Ένα σύστημα που παρουσιάζει καθυστερήσεις, υψηλή κατανάλωση πόρων ή αστάθεια μπορεί να οδηγήσει σε μειωμένη παραγωγικότητα, απώλεια χρηστών και αυξημένα λειτουργικά κόστη. Για τον λόγο αυτό, είναι απαραίτητο να εφαρμόζονται στρατηγικές βελτιστοποίησης που στοχεύουν τόσο στο λογισμικό όσο και στο υλικό του συστήματος.

Στο παρόν υποκεφάλαιο, θα εξετάσουμε τις κύριες τεχνικές που συμβάλλουν στη βελτίωση της απόδοσης ενός συστήματος. Θα αναλύσουμε στρατηγικές λογισμικού που περιλαμβάνουν βελτιώσεις στον κώδικα, caching, ασύγχρονη επεξεργασία και αποδοτικότερη διαχείριση δεδομένων. Επιπλέον, θα διερευνήσουμε βέλτιστες πρακτικές βελτιστοποίησης του frontend, όπως η μείωση του φόρτου από τα scripts και η χρήση CDN, καθώς και τις τεχνικές βελτίωσης του server-side, όπως η αποδοτικότερη διαχείριση των αιτημάτων και η κλιμάκωση υποδομών. Τέλος, θα επικεντρωθούμε σε στρατηγικές που βασίζονται στο υλικό, εξετάζοντας πώς οι αναβαθμίσεις hardware μπορούν να ενισχύσουν την απόδοση του συστήματος.

2.3.1 Στρατηγικές Βασισμένες στο Λογισμικό

Η βελτίωση της απόδοσης ενός συστήματος μέσω στρατηγικών λογισμικού είναι μια πολυδιάστατη διαδικασία που περιλαμβάνει τη βελτιστοποίηση του κώδικα, τη χρήση αποδοτικότερων αλγορίθμων, τη βελτιστοποίηση της επικοινωνίας με τις βάσεις δεδομένων, καθώς και την αξιοποίηση τεχνικών παραλληλισμού και caching. Η αποδοτικότητα ενός συστήματος δεν εξαρτάται μόνο από το υλικό, αλλά και από το πόσο βέλτιστα έχει σχεδιαστεί το λογισμικό του.

Βελτιστοποίηση αλγορίθμων

Η επιλογή των σωστών αλγορίθμων παίζει καθοριστικό ρόλο στη συνολική απόδοση ενός συστήματος. Ένας αποδοτικός αλγόριθμος μπορεί να μειώσει σημαντικά τον απαιτούμενο χρόνο εκτέλεσης και τη χρήση υπολογιστικών πόρων. Η ανάλυση πολυπλοκότητας των αλγορίθμων (Big-O notation) είναι απαραίτητη για την κατανόηση του τρόπου με τον οποίο η απόδοση ενός αλγορίθμου κλιμακώνεται καθώς αυξάνονται τα δεδομένα. Για παράδειγμα, η χρήση μιας δομής δεδομένων όπως τα hash tables μπορεί να μειώσει τον χρόνο αναζήτησης από O(n) σε O(1), βελτιώνοντας δραστικά την απόκριση του συστήματος.

Βελτιστοποίηση ερωτημάτων βάσεων δεδομένων

Οι βάσεις δεδομένων συχνά αποτελούν τον κύριο περιοριστικό παράγοντα στην απόδοση των συστημάτων, ιδιαίτερα όταν διαχειρίζονται μεγάλα σύνολα δεδομένων. Η βελτιστοποίηση των ερωτημάτων SQL μέσω της χρήσης ευρετηρίων **(indexes)**, της αποφυγής πλεοναστικών ενώσεων **(joins)** και της υιοθέτησης τεχνικών caching μπορεί να μειώσει σημαντικά τον χρόνο εκτέλεσης των ερωτημάτων. Επιπλέον, η κατανομή των δεδομένων σε επιμέρους τμήματα **(sharding)** και η χρήση συστημάτων caching όπως το Redis και το Memcached συμβάλλουν στην αποφόρτιση του κεντρικού συστήματος διαχείρισης βάσεων δεδομένων.

Αναδιαμόρφωση κώδικα (Code refactoring)

Η αναδιαμόρφωση του κώδικα αποτελεί μία από τις πιο σημαντικές τεχνικές βελτιστοποίησης του λογισμικού, καθώς επιτρέπει την αποφυγή περιττών υπολογισμών και τη βελτίωση της

αναγνωσιμότητας και της συντηρησιμότητας του λογισμικού. Ένας δομημένος, καθαρός και modular κώδικας διευκολύνει την εύκολη κλιμάκωση του συστήματος και μειώνει τα σφάλματα που μπορεί να οδηγήσουν σε επιβράδυνση ή αστάθεια. Η χρήση σχεδιαστικών προτύπων **(design patterns)**, όπως το Singleton ή το Factory Pattern, επιτρέπει την καλύτερη διαχείριση των αντικειμένων και των πόρων του συστήματος.

Ταυτόχρονη εκτέλεση και παραλληλισμός

Οι σύγχρονες εφαρμογές απαιτούν την εκτέλεση πολλαπλών διαδικασιών ταυτόχρονα για να βελτιώσουν την αποδοτικότητά τους. Ο παραλληλισμός **(parallelism)** και η ταυτόχρονη εκτέλεση **(concurrency)** διαδραματίζουν βασικό ρόλο στη μεγιστοποίηση της απόδοσης των σύγχρονων πολυπύρηνων συστημάτων. Η χρήση multi-threading και asynchronous processing επιτρέπει την ταυτόχρονη διαχείριση πολλαπλών αιτημάτων, μειώνοντας τον χρόνο αναμονής και βελτιώνοντας την απόκριση του συστήματος. Οι τεχνολογίες όπως το Thread Pooling, το MapReduce, και οι asynchronous event-driven αρχιτεκτονικές σε γλώσσες όπως Node.js και Go είναι ιδανικές για την επίτευξη υψηλής ταχύτητας επεξεργασίας δεδομένων.

Συμπέρασμα

Η εφαρμογή στρατηγικών βελτιστοποίησης του λογισμικού αποτελεί καθοριστικό παράγοντα για την απόδοση ενός συστήματος. Η σωστή επιλογή αλγορίθμων, η βελτιστοποίηση των βάσεων δεδομένων, η αναδιαμόρφωση του κώδικα και η αξιοποίηση της ταυτόχρονης εκτέλεσης μπορούν να βελτιώσουν τη σταθερότητα, την ταχύτητα και τη συνολική αποδοτικότητα μιας εφαρμογής. Ο συνδυασμός αυτών των τεχνικών εξασφαλίζει ένα πιο αξιόπιστο και ανθεκτικό σύστημα που μπορεί να ανταποκριθεί στις αυξημένες απαιτήσεις των χρηστών και των επιχειρησιακών διαδικασιών.

2.3.2 Βελτιστοποίηση frontend

Η απόδοση του frontend αποτελεί έναν από τους σημαντικότερους παράγοντες που επηρεάζουν την εμπειρία του χρήστη **(User Experience - UX)** και τη συνολική απόδοση μιας διαδικτυακής εφαρμογής. Ένα αργό ή μη βελτιστοποιημένο frontend μπορεί να οδηγήσει σε υψηλό bounce rate, χαμηλή ικανοποίηση των χρηστών και χαμηλότερη απόδοση της εφαρμογής. Οι στρατηγικές βελτιστοποίησης του frontend περιλαμβάνουν τεχνικές που μειώνουν τον χρόνο φόρτωσης της σελίδας **(Page Load Time)**, τη χρήση επεξεργαστικής ισχύος του προγράμματος περιήγησης **(Browser Rendering)**, καθώς και τη μείωση της κατανάλωσης δικτυακών πόρων.

Ελαχιστοποίηση JavaScript και CSS

Ένα από τα βασικότερα εμπόδια στην απόδοση του frontend είναι η υπερβολική χρήση αρχείων JavaScript και CSS, τα οποία μπορούν να επιβαρύνουν το rendering της σελίδας. Η ελαχιστοποίηση **(minification)** αυτών των αρχείων μέσω εργαλείων όπως το UglifyJS, το Terser και το CSSNano αφαιρεί περιττούς χαρακτήρες, σχόλια και κενά, μειώνοντας το μέγεθός τους και επιταχύνοντας τον χρόνο φόρτωσης. Επιπλέον, η συνένωση **(concatenation)** πολλών αρχείων JavaScript ή CSS σε ένα ενιαίο αρχείο μειώνει τον αριθμό των HTTP requests, βελτιώνοντας την απόδοση.

Τα minified αρχεία συνήθως αποθηκεύονται με την κατάληξη .min, όπως script.min.js ή styles.min.css, ώστε να διαχωρίζονται από τις μη ελαχιστοποιημένες εκδόσεις τους. Αυτή η πρακτική επιτρέπει στους προγραμματιστές να διατηρούν ευανάγνωστο κώδικα κατά τη διάρκεια της ανάπτυξης, ενώ παράλληλα διασφαλίζει ότι η τελική έκδοση που διανέμεται στο διαδίκτυο είναι όσο το δυνατόν πιο αποδοτική. Σε

συνδυασμό με τεχνικές caching και lazy loading, η ελαχιστοποίηση αποτελεί βασικό στοιχείο της βελτιστοποίησης των επιδόσεων του frontend, μειώνοντας τη χρήση εύρους ζώνης **(bandwidth)** και επιταχύνοντας την απόδοση των ιστοσελίδων.

Αντί να φορτώνονται πολλά ξεχωριστά αρχεία CSS για διαφορετικά τμήματα της ιστοσελίδας, φορτώνονται σε ένα και με κατάληξη .min όπως φαίνεται παρακάτω.

```
1  /* Before Optimization */
2  body {
3      font-size: 16px;
4  }
5  h1 {
6      font-weight: bold;
7  }
8
9  /* After Minification */
10 body{font-size:16px}h1{font-weight:bold}
```

Εικόνα 2.1: Παράδειγμα CSS minification

Η διαφορά σε μέγεθος μπορεί να φαίνεται μικρή για ένα μικρό αρχείο, αλλά όταν εφαρμόζεται σε ολόκληρο το frontend, το συνολικό όφελος είναι σημαντικό.

Lazy loading

Το Lazy loading είναι μια τεχνική βελτιστοποίησης φόρτωσης περιεχομένου, η οποία επιτρέπει τη καθυστέρηση της φόρτωσης εικόνων και άλλων πολυμεσικών αρχείων μέχρι να χρειαστούν, δηλαδή μέχρι να εμφανιστούν στην οθόνη του χρήστη. Αυτή η προσέγγιση βοηθά στην αναβολή της φόρτωσης πόρων που δεν είναι άμεσα ορατοί στον χρήστη, με σκοπό να μειώσει το αρχικό μέγεθος της σελίδας και να επιταχύνει την αρχική φόρτωση της ιστοσελίδας. Η χρήση του lazy loading έχει σημαντικά πλεονεκτήματα στην απόδοση, καθώς επιτρέπει στην ιστοσελίδα να φορτώνει μόνο τα στοιχεία που είναι άμεσα ορατά στην περιοχή προβολής του χρήστη **(viewport)**, αποφεύγοντας έτσι την άσκοπη χρήση εύρους ζώνης **(bandwidth)** για την φόρτωση πόρων που δεν έχουν εμφανιστεί ακόμη.

Για παράδειγμα, όταν ο χρήστης φορτώνει μια σελίδα με πολλές εικόνες, τα στοιχεία που είναι ορατά στην οθόνη φορτώνονται πρώτα, ενώ οι εικόνες που βρίσκονται εκτός ορατότητας φορτώνονται αργότερα, καθώς ο χρήστης κυλάει την σελίδα. Αυτή η τεχνική μειώνει την καθυστέρηση στην αρχική φόρτωση της σελίδας, καθιστώντας την πιο γρήγορη και βελτιώνοντας την εμπειρία χρήστη.

Η υλοποίηση του lazy loading σε HTML μπορεί να γίνει με την προσθήκη του χαρακτηριστικού loading="lazy" στα <img> tags ή άλλους τύπους πολυμέσων όπως βίντεο και iframes. Η ετικέτα data-src χρησιμοποιείται για να αναφέρει την πραγματική πηγή της εικόνας ή του αρχείου, ενώ η src παραπέμπει σε μια placeholder εικόνα χαμηλής ανάλυσης ή ένα εικονικό αρχείο που χρησιμοποιείται αρχικά για να κρατά το χώρο μέχρι να φορτωθεί η πλήρης εικόνα.

Πλεονεκτήματα του Lazy Loading:

- **Μείωση του αρχικού χρόνου φόρτωσης:**

  Όταν μόνο τα στοιχεία που είναι άμεσα ορατά φορτώνονται αρχικά, η σελίδα εμφανίζεται γρηγορότερα και η απόδοση της ιστοσελίδας βελτιώνεται.

- **Μειωμένη κατανάλωση εύρους ζώνης (bandwidth):**

  Φορτώνονται μόνο τα δεδομένα που χρειάζονται τη στιγμή που χρειάζονται, μειώνοντας έτσι την υπερβολική κατανάλωση δεδομένων.

- **Καλύτερη εμπειρία χρήστη:**

  Η ταχύτερη φόρτωση της σελίδας δημιουργεί θετική εντύπωση στον χρήστη και μειώνει την πιθανότητα εγκατάλειψης της σελίδας λόγω αργής φόρτωσης.

Αυτή η τεχνική είναι ιδιαίτερα χρήσιμη για ιστοσελίδες με μεγάλο όγκο εικόνων ή βίντεο, όπως σελίδες e-commerce ή ιστοσελίδες με galleries και multimedia περιεχόμενο.

Βελτιστοποίηση των HTTP requests

Η βελτίωση των αιτημάτων HTTP **(HTTP requests)** είναι μια από τις πιο σημαντικές τεχνικές για την επιτάχυνση της φόρτωσης ιστοσελίδων. Κάθε αίτημα HTTP που αποστέλλεται στον διακομιστή προσθέτει καθυστέρηση στον χρόνο φόρτωσης της σελίδας, καθώς κάθε αίτημα απαιτεί χρόνο για την επικοινωνία με τον διακομιστή και τη λήψη των δεδομένων. Επομένως, η μείωση του αριθμού αυτών των αιτημάτων ή η βελτιστοποίηση της διαδικασίας φόρτωσης των πόρων μπορεί να βελτιώσει δραματικά την απόδοση της σελίδας.

Ορισμένες από τις βασικές τεχνικές για τη μείωση του αριθμού των αιτημάτων HTTP είναι οι εξής:

- **Χρήση sprites για εικόνες:**

Η τεχνική των sprites χρησιμοποιείται για την συνένωση πολλών εικόνων σε μία ενιαία εικόνα, η οποία στη συνέχεια χρησιμοποιείται με CSS για να εμφανίζει μόνο το τμήμα της εικόνας που απαιτείται για κάθε στοιχείο της σελίδας. Αντί να φορτώνονται πολλές μικρές εικόνες, χρησιμοποιείται μία μεγάλη εικόνα **(sprite)**, η οποία χωρίζεται σε μικρότερα μέρη μέσω του CSS. Αυτή η προσέγγιση μειώνει σημαντικά τον αριθμό των αιτημάτων HTTP, καθώς χρειάζεται μόνο ένα αίτημα για την εικόνα sprite αντί για πολλά αιτήματα για κάθε μεμονωμένη εικόνα.Για παράδειγμα, αν μια ιστοσελίδα περιέχει πολλές μικρές εικόνες, όπως εικονίδια ή διακοσμητικά στοιχεία, η χρήση ενός sprite εικόνας μειώνει τον αριθμό των αιτημάτων HTTP από, 10 σε 1.

- **Χρήση asynchronous JavaScript (async & defer):**

Η φόρτωση των αρχείων JavaScript μπορεί να προκαλέσει καθυστέρηση στη φόρτωση μιας ιστοσελίδας, επειδή το πρόγραμμα περιήγησης εκτελεί το JavaScript πριν συνεχίσει με το rendering της σελίδας. Αυτό μπορεί να εμποδίσει τη φόρτωση της σελίδας, προκαλώντας καθυστέρηση στην εμφάνιση του περιεχομένου. Για να μειωθεί η καθυστέρηση, χρησιμοποιούνται οι παράμετροι async και defer κατά την εισαγωγή του JavaScript στην ιστοσελίδα. Όταν η παράμετρος async χρησιμοποιείται σε ένα <script> tag, το πρόγραμμα περιήγησης συνεχίζει το rendering της σελίδας χωρίς να περιμένει την εκτέλεση του σεναρίου. Το script εκτελείται μόλις φορτωθεί, χωρίς να επηρεάσει τη ροή φόρτωσης της σελίδας. Η παράμετρος defer καθυστερεί την εκτέλεση του script μέχρι το HTML να έχει φορτωθεί πλήρως. Αυτό σημαίνει ότι το script εκτελείται αφού ολοκληρωθεί το rendering της σελίδας, μειώνοντας έτσι την επίδραση του JavaScript στην ταχύτητα φόρτωσης. Η διαφορά είναι ότι το async εκτελεί το script μόλις γίνει διαθέσιμο, ενώ το defer εξασφαλίζει ότι το script θα εκτελεστεί μόνο αφού φορτωθεί πλήρως το HTML.

- **Χρήση πολυάριθμων αιτημάτων για λιγότερο φορτίο:**

Η βελτίωση της απόδοσης περιλαμβάνει και την ελαχιστοποίηση της ανάγκης για νέες αιτήσεις HTTP μέσω της σωστής διαχείρισης των πόρων. Για παράδειγμα, η τεχνική της συνένωσης **(concatenation)**

πολλών αρχείων JavaScript ή CSS σε ένα ενιαίο αρχείο μειώνει τον αριθμό των αιτημάτων HTTP που απαιτούνται κατά τη φόρτωση της σελίδας, ενώ ταυτόχρονα μειώνει τη συνολική καθυστέρηση φόρτωσης.

Με την εφαρμογή αυτών των τεχνικών, η απόδοση της ιστοσελίδας βελτιώνεται, επιτρέποντας στους χρήστες να αποκτούν πρόσβαση στο περιεχόμενο γρηγορότερα και με καλύτερη εμπειρία χρήσης.

Content delivery networks (CDN)

Τα Content Delivery Networks (CDNs) είναι μια τεχνολογία που βελτιώνει την απόδοση των διαδικτυακών εφαρμογών, μειώνοντας τον χρόνο φόρτωσης περιεχομένου και αυξάνοντας τη διαθεσιμότητα των αρχείων. Τα CDNs αποτελούνται από μια κατανεμημένη ομάδα servers τοποθετημένων σε διάφορες γεωγραφικές τοποθεσίες, με σκοπό τη διανομή του περιεχομένου στους τελικούς χρήστες από τον πιο κοντινό τους server. Η βασική ιδέα πίσω από τη χρήση ενός CDN είναι η μείωση της απόστασης που πρέπει να διανύσουν τα δεδομένα μεταξύ του χρήστη και του server, ελαχιστοποιώντας τον χρόνο καθυστέρησης **(latency)** και αυξάνοντας την ταχύτητα φόρτωσης των σελίδων.

Πλεονεκτήματα της χρήσης CDN:

- **Μείωση καθυστέρησης (Latency):**

  Με τη χρήση ενός CDN, τα δεδομένα (όπως εικόνες, JavaScript, CSS και βίντεο) αποθηκεύονται και διανέμονται από servers που βρίσκονται γεωγραφικά πιο κοντά στους χρήστες. Έτσι, η διαδρομή που πρέπει να διανύσουν τα δεδομένα είναι μικρότερη, πράγμα που έχει ως αποτέλεσμα μικρότερο χρόνο καθυστέρησης. Για παράδειγμα, όταν ένας χρήστης από την Ελλάδα φορτώνει ένα αρχείο από έναν server που βρίσκεται στην Αμερική, το latency μπορεί να είναι σημαντικό. Εάν όμως το αρχείο βρίσκεται σε έναν server που βρίσκεται στην Ευρώπη, η καθυστέρηση μειώνεται σημαντικά.

- **Αύξηση απόδοσης και σταθερότητας:**

  Η κατανομή του φόρτου μεταξύ πολλών servers εξασφαλίζει ότι κανένας server δεν είναι υπερφορτωμένος, κάτι που μπορεί να προκαλέσει καθυστερήσεις ή downtime. Επιπλέον, αν ένας server αντιμετωπίσει προβλήματα ή είναι εκτός λειτουργίας, το CDN μπορεί να κατευθύνει τα αιτήματα στους επόμενους κοντινότερους servers, διασφαλίζοντας τη συνεχιζόμενη διαθεσιμότητα του περιεχομένου.

- **Αποδοτική διαχείριση εύρους ζώνης (Bandwidth):**

  Οι CDNs μπορούν να μειώσουν το φορτίο στους κύριους servers της εφαρμογής, καθώς τα στατικά αρχεία (όπως εικόνες και βιβλιοθήκες JavaScript) φορτώνονται από τους servers του CDN αντί από τον κεντρικό server. Αυτό μειώνει την κατανάλωση εύρους ζώνης του κύριου διακομιστή και ελαχιστοποιεί το κόστος και την επιβάρυνση του.

- **Ασφάλεια:**

  Η χρήση ενός CDN μπορεί επίσης να βελτιώσει την ασφάλεια των διαδικτυακών εφαρμογών, καθώς ο CDN μπορεί να παρέχει προστασία ενάντια σε επιθέσεις DDoS **(Distributed Denial-of-Service)** και άλλες μορφές κακόβουλων επιθέσεων. Επίσης, τα δεδομένα μπορούν να κρυπτογραφούνται και να προστατεύονται σε κάθε σημείο απόκτηση του περιεχομένου.

Η χρήση CDN προσφέρει σημαντικά οφέλη στην απόδοση των διαδικτυακών εφαρμογών, κυρίως λόγω της μείωσης της καθυστέρησης και της αύξησης της διαθεσιμότητας του περιεχομένου. Χρησιμοποιώντας ένα CDN, οι ιστοσελίδες γίνονται πιο γρήγορες και αξιόπιστες, μειώνοντας ταυτόχρονα την πίεση στους κεντρικούς διακομιστές και τη ζήτηση εύρους ζώνης. Ειδικότερα για τα

αρχεία που χρησιμοποιούνται ευρέως όπως βιβλιοθήκες JavaScript ή εικόνες, τα CDN είναι εξαιρετικά αποδοτικά και βελτιώνουν σημαντικά την εμπειρία χρήστη.

Εργαλεία παρακολούθησης απόδοσης frontend

Η παρακολούθηση της απόδοσης ενός frontend είναι ζωτικής σημασίας για την κατανόηση του τρόπου λειτουργίας της εφαρμογής και τη βελτιστοποίησή της. Τα εργαλεία παρακολούθησης αποδόσεων επιτρέπουν στους προγραμματιστές και τους διαχειριστές συστημάτων να συλλέγουν δεδομένα σχετικά με την ταχύτητα φόρτωσης, τα σημεία συμφόρησης, τις καθυστερήσεις και άλλες σημαντικές παραμέτρους που επηρεάζουν την εμπειρία του χρήστη. Ακολουθούν ορισμένα από τα πιο διαδεδομένα και χρήσιμα εργαλεία παρακολούθησης της απόδοσης του frontend.

- **Google Lighthouse:**

Το Google Lighthouse είναι ένα εργαλείο ανοιχτού κώδικα για τη μέτρηση της απόδοσης ιστοσελίδων, το οποίο παρέχει αναλυτικές εκθέσεις σχετικά με την απόδοση, την προσβασιμότητα, τη βελτιστοποίηση για κινητές συσκευές και τις βέλτιστες πρακτικές SEO. Το Lighthouse εκτελεί έναν πλήρη έλεγχο της ιστοσελίδας και παρέχει βαθμολογία από 0 έως 100 για κάθε τομέα, επισημαίνοντας περιοχές για βελτίωση και προτείνοντας διορθώσεις. Εκτός από την ανάλυση απόδοσης, το Lighthouse παρέχει αναφορές για τη χρήση των πόρων και τη βελτιστοποίηση των στοιχείων της σελίδας, όπως η εικόνα, το JavaScript και το CSS, ενισχύοντας έτσι την εμπειρία χρήστη.Το εργαλείο μπορεί να εκτελεστεί είτε από το Chrome DevTools είτε από τη γραμμή εντολών. Επιπλέον, το Lighthouse μπορεί να ενσωματωθεί σε διαδικασίες αυτοματισμού, όπως το Continuous Integration (CI), για τη συνεχιζόμενη παρακολούθηση των επιδόσεων των εφαρμογών.

- **WebPageTest:**

Το WebPageTest είναι ένα εργαλείο που επιτρέπει την ανάλυση της συμπεριφοράς φόρτωσης μιας ιστοσελίδας σε διάφορους περιηγητές, δίκτυα και συσκευές. Προσφέρει μια πολύ λεπτομερή αναφορά για την απόδοση της ιστοσελίδας σε επίπεδο σελίδας και προσφέρει χρήσιμες πληροφορίες όπως ο χρόνος πρώτου byte (TTFB), ο χρόνος φόρτωσης πλήρους σελίδας και ο αριθμός των αιτημάτων HTTP.Μία από τις πιο χρήσιμες δυνατότητες του WebPageTest είναι η δυνατότητα η σελίδα να δοκιμαστεί από διάφορες τοποθεσίες σε όλο τον κόσμο, αναπαριστώντας το πραγματικό περιβάλλον χρήσης και τις διαφορές στη σύνδεση του δικτύου. Επίσης, παρέχει τη δυνατότητα εκτέλεσης πολλαπλών επαναλαμβανόμενων δοκιμών για να διασφαλιστεί ότι τα αποτελέσματα δεν είναι τυχαία και παρέχουν ακριβείς πληροφορίες σχετικά με την απόδοση της σελίδας υπό διάφορες συνθήκες δικτύου.

- **Chrome DevTools:**

Τα Chrome DevTools είναι το εργαλείο ανάπτυξης που περιλαμβάνεται στον περιηγητή Google Chrome και παρέχει ισχυρές δυνατότητες debugging και παρακολούθησης της απόδοσης σε πραγματικό χρόνο. Μεταξύ των λειτουργιών του είναι η δυνατότητα ανάλυσης του χρόνου φόρτωσης της σελίδας, της απόδοσης JavaScript, της διαχείρισης μνήμης, των αιτημάτων δικτύου και άλλων στοιχείων της σελίδας.

Η βελτιστοποίηση του frontend είναι ζωτικής σημασίας για τη δημιουργία εφαρμογών με υψηλή απόδοση και άριστη εμπειρία χρήστη. Η εφαρμογή τεχνικών όπως η ελαχιστοποίηση αρχείων, το lazy loading, η βελτιστοποίηση HTTP requests και η χρήση CDN μπορεί να μειώσει δραστικά τον χρόνο φόρτωσης της σελίδας και να βελτιώσει τη διαδραστικότητα της εφαρμογής. Τα εργαλεία

παρακολούθησης επιτρέπουν τη συνεχή ανάλυση της απόδοσης, διασφαλίζοντας ότι το σύστημα παραμένει βελτιστοποιημένο ακόμα και σε αυξημένο φόρτο χρηστών.

2.3.3 Βελτιστοποίηση server-side

Η βελτιστοποίηση του server-side είναι εξίσου κρίσιμη με την βελτιστοποίηση του frontend για την επίτευξη υψηλής απόδοσης ενός συστήματος. Η αύξηση της απόδοσης στο backend επιτρέπει στο σύστημα να ανταποκρίνεται σε αυξημένα φορτία χρηστών, να μειώνει τον χρόνο απόκρισης και να εξασφαλίζει τη συνεχιζόμενη κλίμακα λειτουργίας σε πραγματικές συνθήκες. Σε αυτό το τμήμα, θα εξετάσουμε κρίσιμες στρατηγικές για τη βελτίωση της απόδοσης του backend, με έμφαση στη βελτιστοποίηση της βάσης δεδομένων, την εξισορρόπηση φορτίου **(load balancing)**, την κλιμάκωση του συστήματος, καθώς και τη χρήση containerization τεχνολογιών.

Βελτιστοποίηση βάσης δεδομένων

Η βάση δεδομένων είναι συχνά το bottleneck σε πολλές εφαρμογές, και η σωστή βελτιστοποίησή της είναι κρίσιμη για την επίτευξη υψηλών επιδόσεων. Οι στρατηγικές που χρησιμοποιούνται για τη βελτιστοποίηση της βάσης δεδομένων περιλαμβάνουν:

- **Indexing:**

  Η δημιουργία ευρετηρίων **(indexes)** στη βάση δεδομένων βοηθά στην ταχύτερη αναζήτηση των δεδομένων. Αντί να αναζητά τα δεδομένα γραμμή προς γραμμή **(sequential scan)**, το ευρετήριο επιτρέπει στην βάση να επιταχύνει την αναζήτηση, μειώνοντας τον χρόνο απόκρισης. Για παράδειγμα, η δημιουργία index σε συχνά αναζητούμενα πεδία (όπως τα ID ή τα πεδία ημερομηνίας) μπορεί να μειώσει την καθυστέρηση κατά την εκτέλεση των queries.

- **Query Optimization:**

  βελτιστοποίηση των SQL queries είναι κρίσιμη για την αποδοτικότητα της βάσης δεδομένων. Οι στρατηγικές περιλαμβάνουν τη χρήση κατάλληλων joins, την αποφυγή αχρείαστων υποερωτημάτων **(subqueries)**, και την επιλογή των σωστών SQL functions που εκτελούνται με μεγαλύτερη ταχύτητα. Επίσης, η χρήση query plan analyzers για την ανάλυση του execution plan μπορεί να βοηθήσει στη βελτίωση της απόδοσης, εντοπίζοντας περιττές λειτουργίες ή μη βέλτιστες στρατηγικές αναζήτησης.

- **Denormalization:**

  Σε ορισμένες περιπτώσεις, η αποανανοικοδόμηση **(denormalization)** της βάσης δεδομένων μπορεί να ενισχύσει την απόδοση μειώνοντας τις απαιτούμενες joins για να συλλεχθούν δεδομένα από πολλαπλούς πίνακες. Αν και μπορεί να δημιουργήσει πλεονάζοντα δεδομένα, προσφέρει σημαντικά πλεονεκτήματα σε εφαρμογές που απαιτούν υψηλή ταχύτητα ανάκτησης δεδομένων.

Load balancing

Η εξισορρόπηση φορτίου **(load balancing)** είναι μια τεχνική που χρησιμοποιείται για την κατανομή των αιτημάτων του χρήστη σε πολλαπλούς servers, προκειμένου να εξασφαλιστεί ότι κανένας server δεν υπερφορτώνεται, και όλοι οι πόροι του συστήματος αξιοποιούνται στο έπακρο. Οι στρατηγικές για την εξισορρόπηση φορτίου περιλαμβάνουν:

- **Round-robin load balancing:**

  Απλή τεχνική εξισορρόπησης όπου τα αιτήματα κατανέμονται κυκλικά στους διαθέσιμους servers. Αν και είναι αποτελεσματικό, μπορεί να μην είναι ιδανικό σε συστήματα με servers διαφορετικής ισχύος ή απόδοσης.

- **Least connections load balancing:**

  Αυτή η τεχνική ανακατευθύνει τα αιτήματα στους servers με τον λιγότερο αριθμό ενεργών συνδέσεων. Είναι πιο αποτελεσματικό για εφαρμογές όπου η φόρτωση του server ποικίλλει δραστικά.

- **IP hashing:**

  Μια στρατηγική που ανακατευθύνει αιτήματα σε servers με βάση τη διεύθυνση IP του χρήστη, εξασφαλίζοντας ότι το ίδιο αίτημα θα κατευθύνεται πάντα στον ίδιο server, βελτιώνοντας την απόδοση των συνεδριών.

Το load balancing όχι μόνο μειώνει τον χρόνο απόκρισης, αλλά και διασφαλίζει τη διαθεσιμότητα και την ανθεκτικότητα του συστήματος, καθώς τα αιτήματα μπορούν να κατευθύνονται σε εφεδρικούς servers σε περίπτωση αποτυχίας ενός ενεργού server.

Κλιμάκωση συστήματος (Scaling)

Η κλιμάκωση συστήματος **(scaling)** είναι η διαδικασία προσαρμογής των πόρων ενός συστήματος για να ανταποκριθεί στην αυξανόμενη ζήτηση ή φόρτο χρηστών. Είναι ένας κρίσιμος παράγοντας για τη διατήρηση της απόδοσης σε συστήματα που αντιμετωπίζουν αυξημένο όγκο δεδομένων ή χρήστες και διασφαλίζει ότι η εφαρμογή μπορεί να ανταποκριθεί αποτελεσματικά χωρίς να επιβραδυνθεί ή να αποτύχει. Υπάρχουν δύο βασικοί τύποι κλιμάκωσης: η οριζόντια κλιμάκωση **(horizontal scaling)** και η κατακόρυφη κλιμάκωση **(vertical scaling)**, οι οποίοι χρησιμοποιούνται ανάλογα με τις ανάγκες του συστήματος και τις προδιαγραφές του.

Οριζόντια κλιμάκωση (Horizontal scaling):

Η οριζόντια κλιμάκωση, γνωστή και ως scaling out, αναφέρεται στην προσθήκη περισσότερων servers ή κόμβων στο σύστημα για να διαχειριστεί την αυξημένη ζήτηση. Αυτή η στρατηγική επιτρέπει την κατανομή του φορτίου μεταξύ πολλών servers, οι οποίοι συνεργάζονται για να εξυπηρετήσουν περισσότερους χρήστες ή αιτήματα. Ο στόχος είναι να κατανεμηθεί το φορτίο ώστε να αποφευχθεί η υπερφόρτωση ενός μεμονωμένου server, ενώ ταυτόχρονα διατηρείται η απόδοση του συστήματος.

Η οριζόντια κλιμάκωση είναι συχνά η προτιμώμενη λύση σε κατανεμημένα συστήματα ή cloud υποδομές, καθώς επιτρέπει την ευέλικτη και δυναμική προσαρμογή στους πόρους. Αυτό σημαίνει ότι ο αριθμός των servers μπορεί να αυξάνεται ή να μειώνεται εύκολα, αναλόγως του φόρτου ή των απαιτήσεων. Ένα σημαντικό πλεονέκτημα είναι η ευχέρεια της κλιμάκωσης, καθώς μπορεί να πραγματοποιηθεί χωρίς περιορισμούς από τη φυσική υποδομή. Επίσης, πολλές πλατφόρμες containerization, όπως το Kubernetes, παρέχουν αυτοματοποιημένη κλιμάκωση, επιτρέποντας την εύκολη κλιμάκωση εφαρμογών χωρίς την ανάγκη χρονοβόρων χειροκίνητων ρυθμίσεων.

Ένα χαρακτηριστικό παράδειγμα είναι τα cloud services όπως η AWS EC2, που επιτρέπουν την οριζόντια κλιμάκωση μέσω Auto Scaling Groups, όπου οι πόροι αυξάνονται ή μειώνονται δυναμικά με βάση το φόρτο της εφαρμογής.

Κατακόρυφη κλιμάκωση (Vertical scaling):

Η κατακόρυφη κλιμάκωση, ή scaling up, αναφέρεται στην αύξηση των πόρων ενός ήδη υπάρχοντος server. Αντί να προσθέτουμε επιπλέον servers, αυξάνουμε τη δυνατότητα του υπάρχοντος server, προσθέτοντας περισσότερη μνήμη RAM, ισχυρότερους επεξεργαστές ή ταχύτερους δίσκους. Η κατακόρυφη κλιμάκωση είναι συνήθως μια πιο άμεση λύση για την ενίσχυση της απόδοσης ενός συστήματος χωρίς την ανάγκη πρόσθετων κόμβων ή servers.

Παρόλο που αυτή η στρατηγική μπορεί να φαίνεται απλή και γρήγορη, έχει περιορισμούς. Ο κύριος περιορισμός είναι η φυσική δυνατότητα του server να υποστηρίξει περισσότερους πόρους. Κάθε server έχει ένα όριο στους πόρους που μπορεί να διαχειριστεί, οπότε όταν αυτό το όριο φτάσει, δεν μπορεί να προστεθούν περαιτέρω πόροι και το σύστημα δεν μπορεί να κλιμακωθεί περαιτέρω. Επίσης, το κόστος για την αναβάθμιση ενός server μπορεί να είναι σημαντικά υψηλότερο σε σχέση με την προσθήκη περισσότερων μικρότερων servers, κάτι που καθιστά τη κατακόρυφη κλιμάκωση λιγότερο αποδοτική οικονομικά σε μεγαλύτερα περιβάλλοντα.

Ένα παράδειγμα κατακόρυφης κλιμάκωσης είναι η αναβάθμιση ενός virtual machine στον cloud, όπου οι πόροι, όπως η RAM και ο CPU, αυξάνονται χωρίς να χρειάζεται να προστίθεται άλλος server στο σύστημα.

Συγκριτικά πλεονεκτήματα και μειονεκτήματα:

Πίνακας 2.1: Σύγκριση οριζόντιας και κατακόρυφης κλιμάκωσης

ΣΤΡΑΤΗΓΙΚΗ	ΠΛΕΟΝΕΚΤΗΜΑΤΑ	ΜΕΙΟΝΕΚΤΗΜΑΤΑ
ΟΡΙΖΟΝΤΙΑ ΚΛΙΜΑΚΩΣΗ (HORIZONTAL SCALING)	Ευέλικτη και δυναμική κλιμάκωση, καλύτερη αντοχή σε αυξημένο φορτίο, χωρίς περιορισμούς από φυσική υποδομή, υποστηρίζει κατανεμημένα συστήματα και cloud περιβάλλοντα	Χρειάζεται κατανομή φόρτου, διαχείριση πολλών servers, πιθανές καθυστερήσεις στο synchronization
ΚΑΤΑΚΟΡΥΦΗ ΚΛΙΜΑΚΩΣΗ (VERTICAL SCALING)	Ευκολότερη εφαρμογή και πιο άμεση λύση σε μικρότερες κλίμακες, δεν απαιτεί νέους servers	Περιορισμένη από τους πόρους του server, αυξάνονται τα κόστη αναβάθμισης, δύσκολη κλιμάκωση σε πολύ μεγάλες κλίμακες

Η επιλογή μεταξύ οριζόντιας και κατακόρυφης κλιμάκωσης εξαρτάται από τις ανάγκες της εφαρμογής, τη φύση του φόρτου και τη δυνατότητα να διαχειριστεί το σύστημα τους απαιτούμενους πόρους. Συνήθως, η οριζόντια κλιμάκωση είναι η πιο ευέλικτη λύση για εφαρμογές σε cloud περιβάλλοντα, ενώ η κατακόρυφη κλιμάκωση μπορεί να είναι κατάλληλη για εφαρμογές με λιγότερους πόρους ή σε περιβάλλοντα με μικρότερες απαιτήσεις. Η επιλογή της κατάλληλης στρατηγικής κλιμάκωσης είναι καθοριστική για την απόδοση, τη σταθερότητα και τη βιωσιμότητα του συστήματος σε αυξανόμενα φορτία.

2.3.4 Στρατηγικές βασισμένες σε υλικό

Η βελτίωση της απόδοσης ενός συστήματος δεν περιορίζεται μόνο στη βελτιστοποίηση του λογισμικού. Συχνά, το hardware παίζει καθοριστικό ρόλο στην ενίσχυση της ταχύτητας και της απόκρισης ενός συστήματος, ειδικά όταν η εφαρμογή απαιτεί αυξημένους πόρους ή ταχύτητα επεξεργασίας. Σε αυτό το υποκεφάλαιο, θα αναλύσουμε στρατηγικές που σχετίζονται με το hardware, με σκοπό την αύξηση της απόδοσης του συστήματος σε διάφορες πτυχές του. Οι στρατηγικές αυτές περιλαμβάνουν την αναβάθμιση του αποθηκευτικού μέσου, την αύξηση της μνήμης RAM, τη βελτίωση της δικτυακής υποδομής, και την αξιοποίηση εξειδικευμένων hardware λύσεων, όπως οι GPUs για παράλληλη επεξεργασία.

Χρήση SSD αντί για HDD

Η χρήση Solid State Drives (SSD) αντί για Hard Disk Drives (HDD) αποτελεί μια κρίσιμη στρατηγική βελτίωσης της απόδοσης των συστημάτων υπολογιστών, καθώς η διαφορά στην απόδοση μεταξύ των δύο τύπων αποθηκευτικών μέσων είναι σημαντική. Ενώ οι HDDs βασίζονται σε περιστρεφόμενους δίσκους και μηχανικά εξαρτήματα για την αποθήκευση και ανάκτηση δεδομένων, οι SSDs χρησιμοποιούν flash μνήμη (NAND), η οποία δεν απαιτεί μηχανικές κινήσεις και επιτρέπει τη γρήγορη πρόσβαση στα δεδομένα. Αυτή η τεχνολογία καθιστά τους SSDs ταχύτερους σε σχέση με τους HDDs, με σημαντικά μικρότερους χρόνους ανάγνωσης και εγγραφής δεδομένων.

Η κύρια διαφορά μεταξύ HDD και SSD αφορά τη ταχύτητα ανάγνωσης/εγγραφής δεδομένων. Οι SSDs επιτυγχάνουν ταχύτητες ανάγνωσης και εγγραφής της τάξης των 500-550 MB/s για τις πιο κοινές μονάδες SATA SSD, ενώ οι HDDs έχουν ταχύτητες που συνήθως κυμαίνονται γύρω από τα 80-160 MB/s. Αυτή η διαφοροποίηση οφείλεται στην αρχιτεκτονική τους: οι SSDs, χρησιμοποιώντας ηλεκτρονικές διατάξεις μνήμης NAND, προσφέρουν τυχαία πρόσβαση στα δεδομένα, ενώ οι HDDs βασίζονται σε μηχανικά μέρη, όπως τα περιστρεφόμενα δίσκους και κεφαλές ανάγνωσης, γεγονός που καθυστερεί τη διαδικασία προσπέλασης δεδομένων.

Η ταχύτητα των SSDs εκδηλώνεται και στους χρόνους απόκρισης του συστήματος. Στους HDDs, λόγω των μηχανικών κινήσεων, οι χρόνοι αναμονής για την ανάγνωση ή εγγραφή δεδομένων είναι συνήθως αρκετά υψηλοί, κυρίως λόγω της καθυστέρησης αναζήτησης και του χρόνου περιστροφής. Αντίθετα, οι SSDs προσφέρουν πολύ χαμηλότερους χρόνους καθυστέρησης και επιτρέπουν την ταχύτερη εκκίνηση εφαρμογών, το γρηγορότερο φόρτωμα δεδομένων και τη μειωμένη καθυστέρηση κατά την εκκίνηση του λειτουργικού συστήματος. Συνεπώς, η μετάβαση σε SSD βελτιώνει τη γενική απόδοση του συστήματος και μειώνει το χρόνο αναμονής για τον χρήστη, συμβάλλοντας σε πιο άμεση και αποδοτική αλληλεπίδραση με το σύστημα.

Η μετάβαση από HDD σε SSD έχει σημαντική επίδραση στην απόδοση εφαρμογών που διαχειρίζονται μεγάλες βάσεις δεδομένων, μεγάλους όγκους δεδομένων ή απαιτούν γρήγορη πρόσβαση σε δεδομένα, όπως οι βάσεις δεδομένων και οι εφαρμογές υψηλής ταχύτητας **(real-time)**. Συστήματα που βασίζονται σε δεδομένα που συνεχώς διαβάζονται και εγγράφονται (όπως web servers, big data platforms, και cloud services) επωφελούνται άμεσα από τη χρήση SSDs, καθώς μειώνεται ο χρόνος που απαιτείται για την ανάκτηση δεδομένων από τον αποθηκευτικό χώρο.

Αύξηση μνήμης RAM

Η μνήμη RAM **(Random Access Memory)** αποτελεί έναν από τους πιο κρίσιμους παράγοντες που επηρεάζουν τη συνολική απόδοση ενός υπολογιστικού συστήματος. Είναι υπεύθυνη για την προσωρινή αποθήκευση των δεδομένων που χρησιμοποιούνται άμεσα από την Κεντρική Μονάδα Επεξεργασίας (CPU), επιτρέποντας την ταχεία πρόσβαση σε αυτά. Ουσιαστικά, η RAM λειτουργεί ως ταχεία ενδιάμεση μνήμη, επιταχύνοντας τις διαδικασίες εκτέλεσης εντολών και δεδομένων, σε αντίθεση με την αποθήκευση δεδομένων σε πιο αργούς χώρους αποθήκευσης, όπως οι σκληροί δίσκοι (HDD) ή οι SSD.

Η απόδοση ενός συστήματος επηρεάζεται άμεσα από την ικανότητα του συστήματος να διατηρεί δεδομένα στη RAM για άμεση πρόσβαση. Όταν ο διαθέσιμος χώρος μνήμης RAM είναι περιορισμένος, το σύστημα αρχίζει να χρησιμοποιεί έναν πιο αργό μηχανισμό αποθήκευσης, τον μηχανισμό paging ή swap, όπου τα δεδομένα που κανονικά θα βρίσκονταν στη RAM, αποθηκεύονται στον δίσκο (HDD ή SSD). Αυτό δημιουργεί σημαντική καθυστέρηση, καθώς η ανάγνωση και η εγγραφή στον δίσκο είναι πολύ πιο αργές από τη λειτουργία της μνήμης RAM, και μπορεί να οδηγήσει σε μείωση της ταχύτητας του συστήματος και της γενικής απόκρισης του.

Η αύξηση της μνήμης RAM επιτρέπει στο σύστημα να αποθηκεύει περισσότερα δεδομένα σε γρήγορη μνήμη, με αποτέλεσμα την εξάλειψη των καθυστερήσεων κατά την ανάκτηση δεδομένων και την ταχύτερη εκτέλεση των εργασιών. Σε περιβάλλοντα που απαιτούν εντατική επεξεργασία δεδομένων, όπως μεγάλα ερωτήματα βάσεων δεδομένων ή εφαρμογές μηχανικής μάθησης, η επαρκής μνήμη RAM είναι κρίσιμη για τη διατήρηση της απόδοσης υπό φορτίο. Η καλή εκμετάλλευση της RAM αποτρέπει τις αναγκαίες εναλλαγές μεταξύ RAM και δίσκου, επιταχύνοντας σημαντικά την απόκριση του συστήματος.

Βελτίωση δικτυακής υποδομής

Η απόδοση των δικτυακών υποδομών επηρεάζει σημαντικά την ταχύτητα των επικοινωνιών μεταξύ των συστημάτων και τις ανταλλαγές δεδομένων σε κατανεμημένα περιβάλλοντα. Ο χρόνος καθυστέρησης δικτύου **(latency)**, η χωρητικότητα του εύρους ζώνης **(bandwidth)**, και η αξιοπιστία του δικτύου παίζουν σημαντικό ρόλο στην ταχύτητα με την οποία το σύστημα μπορεί να ανταποκριθεί σε αιτήματα χρηστών ή να επεξεργαστεί δεδομένα.

Η βελτίωση της δικτυακής υποδομής περιλαμβάνει τη χρήση πιο γρήγορων και αξιόπιστων συνδέσεων δικτύου, τη μείωση της καθυστέρησης **(latency)** μέσω της στρατηγικής επιλογής τοποθεσιών server **(geographic location)** ή την ενίσχυση της ποιότητας του δικτύου μέσω προηγμένων τεχνικών όπως η χρήση δίκτυα 5G ή η μετάβαση σε υψηλής ποιότητας δικτυακά καλώδια. Επιπλέον, η ενίσχυση της υποδομής για τη διαχείριση της κυκλοφορίας **(traffic management)** και η χρήση τεχνικών όπως το Quality of Service **(QoS)** για τη βελτίωση της προτεραιότητας δεδομένων μπορεί να μειώσει τη συμφόρηση και να αυξήσει την ταχύτητα μετάδοσης των δεδομένων.

Αξιοποίηση εξειδικευμένων hardware λύσεων (π.χ. GPUs)

Σύγχρονες εφαρμογές, όπως οι μηχανισμοί μηχανικής μάθησης και η επεξεργασία μεγάλων δεδομένων, απαιτούν μια ισχυρή υπολογιστική ισχύ που συχνά ξεπερνά τις δυνατότητες των παραδοσιακών CPUs. Οι γραφικές κάρτες επεξεργασίας (GPUs) είναι εξειδικευμένα υλικά που σχεδιάστηκαν για την εκτέλεση παράλληλων υπολογισμών και είναι ιδανικές για εφαρμογές που απαιτούν υψηλή υπολογιστική ισχύ σε μεγάλες ποσότητες δεδομένων.

Η χρήση GPUs επιτρέπει τη βελτιστοποίηση των διαδικασιών που απαιτούν παράλληλη επεξεργασία, όπως η εκπαίδευση μοντέλων μηχανικής μάθησης ή η ανάλυση μεγάλων συνόλων δεδομένων. Οι GPUs έχουν συνήθως εκατοντάδες ή χιλιάδες πυρήνες, επιτρέποντας τη γρήγορη εκτέλεση εντολών σε πολλαπλές υπολογιστικές διεργασίες ταυτόχρονα. Σε αντίθεση με τις CPUs, οι οποίες είναι βελτιστοποιημένες για σειριακή εκτέλεση εντολών, οι GPUs προσφέρουν εξαιρετική απόδοση σε εφαρμογές που απαιτούν μεγάλες παράλληλες υπολογιστικές διεργασίες.

Ένας άλλος τομέας όπου οι GPUs μπορούν να προσφέρουν σημαντικά πλεονεκτήματα είναι η επεξεργασία εικόνας και βίντεο, οι οποίες συνήθως απαιτούν εκτεταμένους υπολογισμούς για την ανάλυση και την επεξεργασία. Ειδικά για εφαρμογές υψηλών επιδόσεων **(high-performance computing)**, η χρήση GPUs μπορεί να επιταχύνει τις διεργασίες και να μειώσει τον χρόνο απόκρισης, ενισχύοντας τη συνολική απόδοση του συστήματος [18].

2.4 Συγκριτικές μελέτες και κενά έρευνας

Η αποτελεσματική αξιολόγηση της απόδοσης ενός συστήματος απαιτεί τη χρήση κατάλληλων εργαλείων δοκιμών. Ωστόσο, η ποικιλία διαθέσιμων εργαλείων, οι διαφορές στις δυνατότητες τους και οι ειδικές απαιτήσεις κάθε εφαρμογής καθιστούν αναγκαία τη σύγκριση μεταξύ τους. Οι συγκριτικές

μελέτες παρέχουν πολύτιμες πληροφορίες σχετικά με τη λειτουργικότητα, την επεκτασιμότητα και την αποδοτικότητα κάθε εργαλείου, επιτρέποντας την επιλογή του καταλληλότερου για τις εκάστοτε ανάγκες.

Στο παρόν υποκεφάλαιο, θα πραγματοποιηθεί μια συγκριτική ανάλυση τεσσάρων δημοφιλών εργαλείων δοκιμών επιδόσεων: Apache JMeter, LoadRunner, Gatling και BlazeMeter. Παράλληλα, θα εντοπιστούν κενά στην έρευνα σχετικά με τις μεθοδολογίες αξιολόγησης της απόδοσης και τη βελτίωση των υφιστάμενων εργαλείων.

Η παρακάτω ανάλυση εξετάζει τα τέσσερα εργαλεία βάσει των κύριων χαρακτηριστικών τους, της καταλληλότητάς τους για συγκεκριμένα σενάρια δοκιμών και των περιορισμών τους.

Πίνακας 2.2: Σύγκριση εργαλείων δοκιμών απόδοσης

ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ	APACHE JMETER	LOADRUNNER	GATLING	BLAZEMETER
ΤΥΠΟΣ ΕΡΓΑΛΕΙΟΥ	Open-source	Εμπορικό	Open-source	Cloud-based (SaaS)
ΓΛΩΣΣΑ ΥΛΟΠΟΙΗΣΗΣ	Java	C, Java	Scala	Βασίζεται σε JMeter
ΥΠΟΣΤΗΡΙΞΗ ΠΡΩΤΟΚΟΛΛΩΝ	HTTP, HTTPS, JDBC, FTP, SOAP, REST	HTTP, HTTPS, Citrix, SAP, WebSockets, Database	HTTP, WebSockets	HTTP, HTTPS, API Testing
ΚΛΙΜΑΚΩΣΗ (SCALABILITY)	Περιορισμένη, απαιτεί κατανεμημένη αρχιτεκτονική	Πολύ υψηλή, βελτιστοποιημένη για μεγάλες δοκιμές	Σχεδιασμένο για υψηλές επιδόσεις, αλλά απαιτεί τεχνική κατάρτιση	Υψηλή, λόγω cloud υποδομής
ΔΙΑΧΕΙΡΙΣΗ ΔΟΚΙΜΩΝ	Χειροκίνητη ρύθμιση σεναρίων	Πλήρες σύστημα διαχείρισης δοκιμών	Σενάρια βασισμένα σε κώδικα	Εύκολη διαχείριση μέσω cloud
ΕΥΚΟΛΙΑ ΧΡΗΣΗΣ	Μέτρια, απαιτεί ρύθμιση	Υψηλή, διαθέτει GUI και αυτοματοποίηση	Μέτρια προς δύσκολη (χρήση Scala)	Πολύ υψηλή, βασισμένη σε UI
ΕΝΣΩΜΑΤΩΣΗ ΜΕ CI/CD PIPELINES	Ναι (με plugins)	Ναι	Ναι, ιδιαίτερα με DevOps εργαλεία	Ναι, με ενσωμάτωση JMeter

Παρά την πρόοδο στα εργαλεία δοκιμών επιδόσεων, υπάρχουν αρκετά κενά που χρήζουν περαιτέρω έρευνας:

- **Απουσία ενοποιημένων μεθοδολογιών αξιολόγησης:**

  Παρά τις τεχνολογικές εξελίξεις, δεν υπάρχει ένα ενιαίο πρότυπο για τη συγκριτική αξιολόγηση των εργαλείων, με αποτέλεσμα κάθε οργανισμός να υιοθετεί διαφορετικές πρακτικές.

- **Έλλειψη αυτοματοποιημένων λύσεων με AI/ML:**

  Οι περισσότερες δοκιμές επιδόσεων εξακολουθούν να απαιτούν σημαντική χειροκίνητη ρύθμιση. Η αξιοποίηση τεχνολογιών Machine Learning θα μπορούσε να βελτιώσει την προσαρμογή των δοκιμών σε πραγματικές συνθήκες.

- **Βελτιστοποίηση της κλιμάκωσης σε κατανεμημένα περιβάλλοντα:**

  Παρόλο που πολλά εργαλεία (όπως το BlazeMeter) αξιοποιούν cloud υποδομές, οι κατανεμημένες δοκιμές εξακολουθούν να είναι δύσκολες στη ρύθμιση και διαχείριση.

- **Συνδυασμός δοκιμών επιδόσεων με UX testing:**

  Τα περισσότερα εργαλεία επικεντρώνονται σε τεχνικά metrics **(latency, throughput)**, παραμελώντας τη συνολική εμπειρία του χρήστη. Μια μελλοντική κατεύθυνση είναι η ενσωμάτωση δεικτών που σχετίζονται με το UI responsiveness και την αντίληψη του χρήστη για την απόδοση.

- **Ασφάλεια και επιπτώσεις στις δοκιμές επιδόσεων:**

  Τα εργαλεία δοκιμών επικεντρώνονται αποκλειστικά στην απόδοση, χωρίς να ενσωματώνουν προληπτικούς μηχανισμούς για ανίχνευση ευπαθειών ασφαλείας που μπορεί να επηρεάζουν τις δοκιμές.

Η συγκριτική ανάλυση των εργαλείων δείχνει ότι δεν υπάρχει μία λύση που να καλύπτει όλες τις ανάγκες των δοκιμών επιδόσεων. Το Apache JMeter είναι μια καλή επιλογή για open-source περιβάλλοντα, το LoadRunner προσφέρει εκτεταμένες δυνατότητες αλλά έχει υψηλό κόστος, το Gatling είναι ιδανικό για DevOps σενάρια, ενώ το BlazeMeter προσφέρει την ευκολία του cloud.

Παράλληλα, η έρευνα στον τομέα των δοκιμών επιδόσεων αντιμετωπίζει προκλήσεις, όπως η έλλειψη ενοποιημένων μεθοδολογιών και η ανάγκη για μεγαλύτερη αυτοματοποίηση μέσω AI. Οι μελλοντικές εξελίξεις θα πρέπει να επικεντρωθούν στην ενσωμάτωση UX μετρήσεων, στην κλιμάκωση των δοκιμών σε κατανεμημένα περιβάλλοντα και στη βελτίωση της ασφάλειας κατά τη διαδικασία των δοκιμών [19], [20].

Κεφάλαιο 3ο: Ερευνητική προσέγγιση θέματος

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τη μεθοδολογία που ακολουθήθηκε για τη μέτρηση της απόδοσης της εφαρμογής, καθώς και τα εργαλεία που χρησιμοποιήθηκαν σε αυτήν τη διαδικασία. Αρχικά, θα περιγράψουμε αναλυτικά τα βήματα που ακολουθήθηκαν για τη συλλογή και ανάλυση δεδομένων απόδοσης, εξετάζοντας τις βασικές παραμέτρους που επηρεάζουν την ταχύτητα και τη σταθερότητα της εφαρμογής. Αυτό περιλαμβάνει τη χρήση εξειδικευμένων εργαλείων παρακολούθησης, καταγραφής και ανάλυσης, που επιτρέπουν τη λεπτομερή εξέταση της συμπεριφοράς της εφαρμογής υπό διάφορες συνθήκες φόρτου.

Στη συνέχεια, θα εξετάσουμε τις βελτιώσεις που εφαρμόστηκαν σε όλα τα επίπεδα του συστήματος, τόσο σε επίπεδο λογισμικού όσο και υποδομής. Αυτό περιλαμβάνει την αναγνώριση των σημείων συμφόρησης και των περιοχών με χαμηλή αποδοτικότητα, καθώς και τις παρεμβάσεις που έγιναν για τη βελτίωση της απόδοσης, όπως βελτιστοποίηση αλγορίθμων, αναβάθμιση υποδομών ή βελτιώσεις στη διαχείριση των βάσεων δεδομένων. Κάθε μία από αυτές τις αλλαγές θα αναλυθεί λεπτομερώς, ώστε να κατανοηθεί πώς συνέβαλαν στη συνολική βελτίωση της απόδοσης.

Τέλος, θα πραγματοποιηθεί επανέλεγχος της απόδοσης της εφαρμογής μετά τις βελτιώσεις, και θα συγκριθούν τα αποτελέσματα με τις αρχικές μετρήσεις. Η σύγκριση αυτή θα επιτρέψει την αξιολόγηση της αποτελεσματικότητας των αλλαγών και θα παράσχει σαφή στοιχεία για το πόσο οι βελτιώσεις συνέβαλαν στη μείωση των πόρων που απαιτούνται και στην αύξηση της ταχύτητας και σταθερότητας του συστήματος. Η ανάλυση των αποτελεσμάτων θα μας δώσει τη δυνατότητα να εξαγάγουμε συμπεράσματα για την επιτυχία των βελτιώσεων και να προτείνουμε περαιτέρω βήματα βελτιστοποίησης.

3.1 Χαρακτηριστικά περιβάλλοντος δοκιμών

Η εφαρμογή που αποτέλεσε το αντικείμενο της παρούσας μελέτης βασίζεται στην πλατφόρμα AdminKit, μια προηγμένη λύση Διαχείρισης Ανθρώπινου Δυναμικού **(HRM - Human Resource Management)**. Η πλατφόρμα αυτή χρησιμοποιείται από επιχειρήσεις για τη διαχείριση προσωπικού, τμημάτων, αδειών, μισθοδοσίας και άλλων σχετικών διαδικασιών.

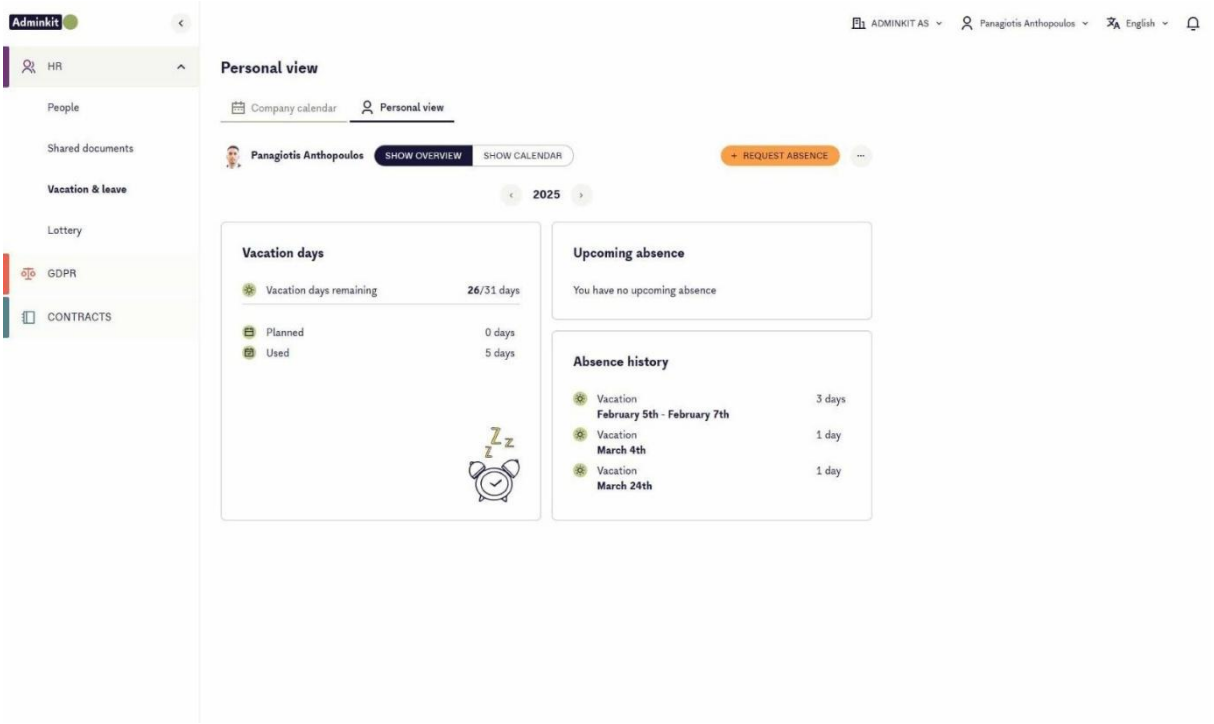
Η εφαρμογή παρέχει ένα σύγχρονο και διαδραστικό περιβάλλον χρήστη **(UI)**, το οποίο προσφέρει προσαρμόσιμους πίνακες ελέγχου **(dashboards)**, διαχείριση ρόλων χρηστών, καθώς και λειτουργίες αναφορών και στατιστικών. Σχεδιασμένη με γνώμονα τη βελτιστοποίηση της παραγωγικότητας, επιτρέπει στους υπεύθυνους ανθρώπινου δυναμικού να οργανώνουν και να παρακολουθούν τις λειτουργίες μιας εταιρείας με αποτελεσματικό και αυτοματοποιημένο τρόπο.

Η αρχιτεκτονική της εφαρμογής στηρίζεται σε cloud-based υποδομή, φιλοξενείται στο Microsoft Azure και χρησιμοποιεί σύγχρονες τεχνολογίες backend και frontend για τη βέλτιστη απόδοση. Στην παρούσα μελέτη, πραγματοποιήθηκαν δοκιμές επιδόσεων για την αξιολόγηση της ανταπόκρισης του συστήματος υπό διαφορετικά επίπεδα φόρτου, καθώς και για την ανάλυση της αποδοτικότητας των βελτιστοποιήσεων που εφαρμόστηκαν.

Ανάλυση της εφαρμογής

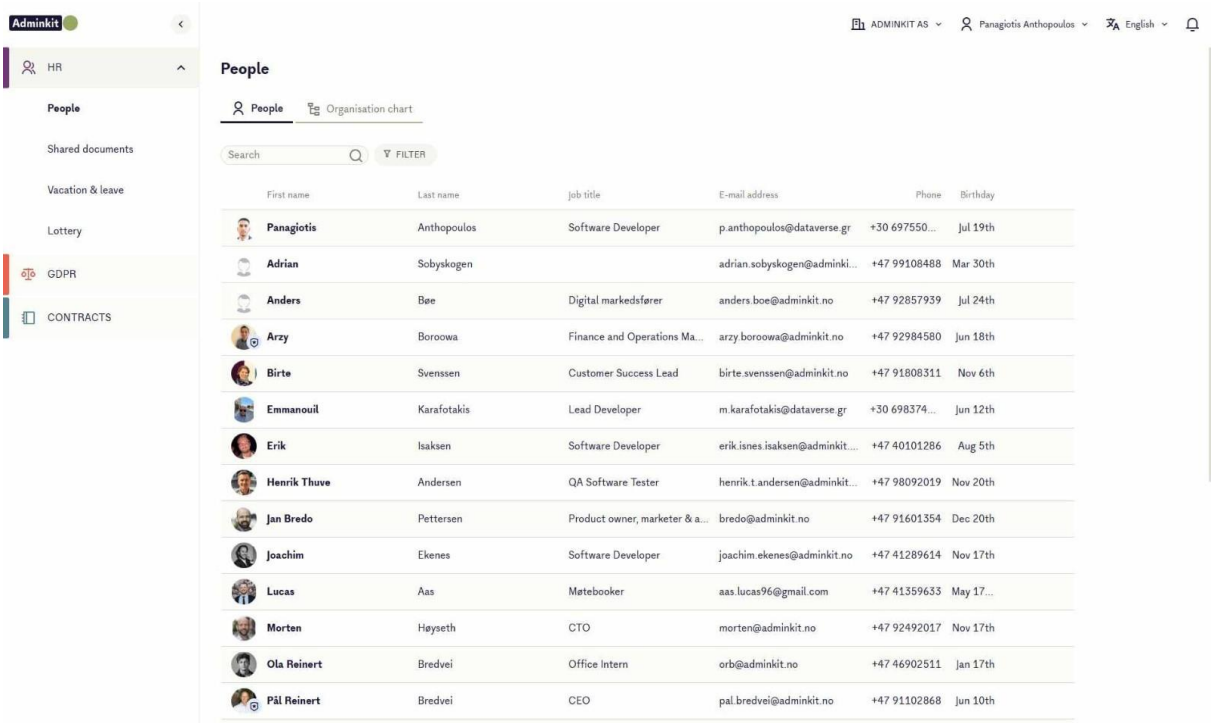
Η εφαρμογή που εξετάζεται στην παρούσα μελέτη παρέχει ένα ολοκληρωμένο σύστημα διαχείρισης ανθρώπινου δυναμικού, επιτρέποντας στις επιχειρήσεις να οργανώνουν και να παρακολουθούν τις δραστηριότητες των εργαζομένων με αυτοματοποιημένο τρόπο.

Η εφαρμογή διαθέτει ένα πλευρικό μενού πλοήγησης (Left Navigation Menu), μέσω του οποίου ο χρήστης έχει άμεση πρόσβαση σε βασικές λειτουργίες, όπως φαίνεται και στην ακόλουθη εικόνα. Αυτές που θα χρησιμοποιήσουμε για τις δοκιμές μας είναι η διαχείριση προσωπικού, την διαχείριση GDPR και τέλος την διαχείριση αδειών προσωπικού και το ημερολόγιο.



Εικόνα 3.1: Η σελίδα του προσωπικού ημερολογίου ενός υπαλλήλου

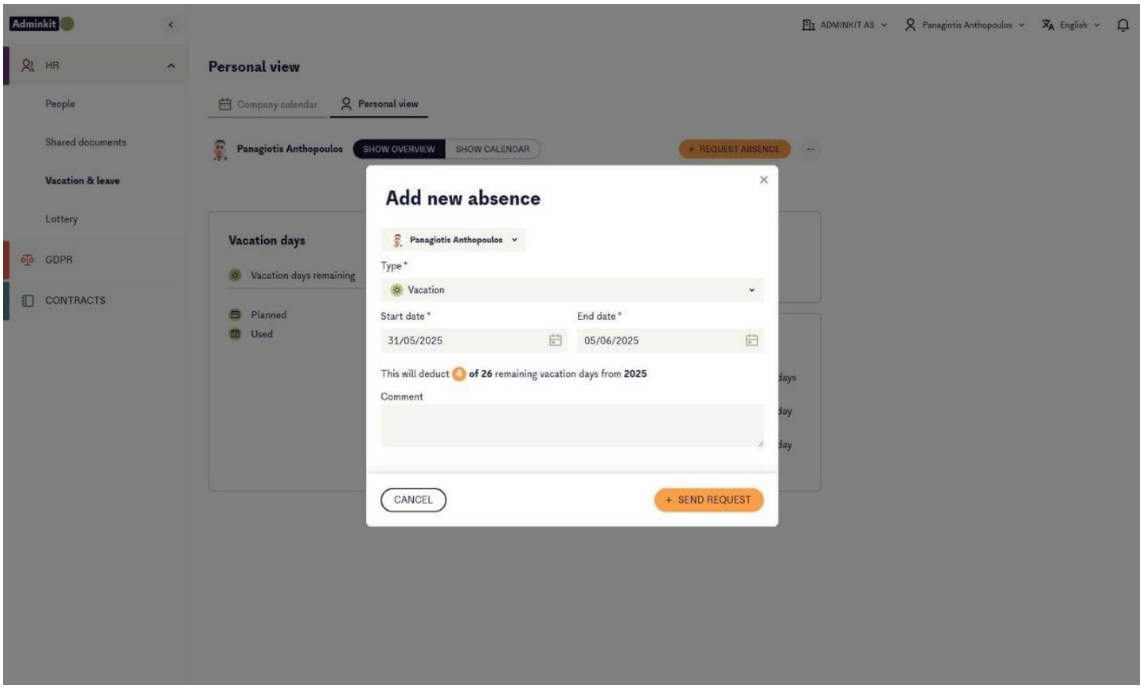
Η Διαχείριση Προσωπικού (People List - Λίστα Υπαλλήλων) επιτρέπει την προβολή όλων των εργαζομένων της επιχείρησης σε μορφή λίστας, ενώ παράλληλα δίνει τη δυνατότητα προβολής αναλυτικών προφίλ για κάθε εργαζόμενο κάνοντας κλικ στο προφίλ του αντίστοιχου, συμπεριλαμβανομένων προσωπικών στοιχείων, ρόλου στην εταιρεία, στοιχείων επικοινωνίας και ιστορικού αδειών. Στην παρακάτω εικόνα βλέπουμε πως ακριβώς είναι το UI.



Εικόνα 3.2: Η σελίδα των υπαλλήλων

Η Διαχείριση GDPR προσφέρει μηχανισμούς που επιτρέπουν στον διαχειριστή της εταιρείας να παρακολουθεί και να διαχειρίζεται τη συμμόρφωση της εταιρείας με τον Γενικό Κανονισμό Προστασίας Δεδομένων (GDPR).

Στη Διαχείριση Αδειών Προσωπικού, η εφαρμογή παρέχει ένα ημερολόγιο αδειών, μέσω του οποίου οι εργαζόμενοι μπορούν να υποβάλλουν αιτήσεις αδειών, ενώ οι διαχειριστές έχουν τη δυνατότητα έγκρισης ή απόρριψης αυτών των αιτήσεων. Παράλληλα, προσφέρει μια οπτική αναπαράσταση της διαθεσιμότητας των υπαλλήλων, διευκολύνοντας τον προγραμματισμό των εργασιακών υποχρεώσεων. Παρακάτω βλέπουμε την γραφική μορφή της αίτησης άδειας.



Εικόνα 3.3: Η φόρμα προσθήκης άδειας

Παρακάτω παρατίθενται οι τεχνικές προδιαγραφές και οι παράμετροι που υιοθετήθηκαν για την εκτέλεση των δοκιμών.

Υπολογιστική υποδομή και hardware

Η εφαρμογή αναπτύχθηκε σε εικονικές μηχανές **(VMs)** της Azure Virtual Machines, διαμορφωμένες με προεπιλεγμένες ρυθμίσεις υπολογιστικών πόρων, χωρίς προσαρμοσμένες ρυθμίσεις CPU ή μνήμης RAM. Η επιλογή της προεπιλεγμένης κατανομής πόρων βασίστηκε στην ικανότητα της πλατφόρμας να κλιμακώνει αυτόματα **(auto-scaling)**, διατηρώντας σταθερή απόδοση υπό διαφορετικά επίπεδα φορτίου.

Κατά τη διάρκεια των δοκιμών, δεν εφαρμόστηκαν μηχανισμοί εξισορρόπησης φορτίου **(load balancing)** ή clustering, διασφαλίζοντας ότι οι μετρήσεις αφορούσαν αποκλειστικά την απόδοση του συστήματος υπό μονολιθική αρχιτεκτονική. Επιπλέον, δεν πραγματοποιήθηκαν τεχνητοί περιορισμοί εύρους ζώνης ή εισαγωγή καθυστερήσεων δικτύου, ώστε να αναλυθεί η καθαρή επίδοση του backend και των εξυπηρετητών.

Λογισμικό και διαμόρφωση Συστήματος

Ο διακομιστής εφαρμογής εκτελείται σε Windows Server, χρησιμοποιώντας Internet Information Services **(IIS)** ως web server. Η βάση δεδομένων της εφαρμογής βασίζεται σε SQL, ενώ για τη βελτιστοποίηση της απόκρισης μετά τις αρχικές δοκιμές εφαρμόστηκαν μηχανισμοί προσωρινής αποθήκευσης **(caching)**.

Για την αύξηση της αποδοτικότητας, αξιοποιήθηκε in-memory caching, καθώς και Redis, μειώνοντας τη συχνότητα των κλήσεων προς τη βάση δεδομένων και επιτυγχάνοντας σημαντική μείωση των χρόνων απόκρισης.

Εργαλεία δοκιμών και διαδικασία εκτέλεσης

Οι μετρήσεις απόδοσης υλοποιήθηκαν με χρήση του Apache JMeter, ενός εξειδικευμένου εργαλείου δοκιμών επιδόσεων που επιτρέπει την προσομοίωση φορτίου υψηλής κλίμακας με τη δημιουργία πολλαπλών παράλληλων χρηστών και αιτημάτων.

Το Azure Monitoring αξιοποιήθηκε για τη συνεχή παρακολούθηση των υπολογιστικών πόρων κατά τη διάρκεια των δοκιμών, καταγράφοντας τη χρήση CPU και της μνήμης RAM.

Παράγοντες που ενδέχεται να επηρεάσουν τα αποτελέσματα

Παρόλο που το περιβάλλον δοκιμών διαμορφώθηκε ώστε να είναι όσο το δυνατόν πιο ελεγχόμενο, υπήρχαν ορισμένοι εξωτερικοί παράγοντες που ενδέχεται να επηρέασαν τις μετρήσεις:

- Η εφαρμογή φιλοξενούνταν σε υποδομή cloud, χωρίς πλήρη απομόνωση από άλλες υπηρεσίες.
- Οι δοκιμές διεξήχθησαν σε χρονικά διαστήματα όπου δεν υπήρχε σημαντική αλληλεπίδραση πραγματικών χρηστών, ώστε να ελαχιστοποιηθούν απρόβλεπτες διακυμάνσεις στην απόδοση.

Συνεπώς, τα αποτελέσματα της παρούσας μελέτης παρέχουν αξιόπιστες ενδείξεις για τη συμπεριφορά του συστήματος σε ελεγχόμενα περιβάλλοντα δοκιμών, με τη δυνατότητα περαιτέρω βελτίωσης μέσω πιο απομονωμένων ή επαναληπτικών δοκιμών σε διαφορετικές συνθήκες φόρτου.

3.2 Baseline testing

Για τη μέτρηση των αρχικών επιδόσεων της εφαρμογής πριν από οποιαδήποτε βελτιστοποίηση, πραγματοποιήθηκαν δοκιμές βασικών επιδόσεων **(baseline performance testing)**. Αυτές οι δοκιμές χρησίμευσαν ως σημείο αναφοράς για τη σύγκριση των αποτελεσμάτων πριν και μετά τις βελτιώσεις.

3.2.1 Σενάρια δοκιμών

Για την αξιολόγηση της απόδοσης, σχεδιάστηκαν και εκτελέστηκαν δύο βασικά σενάρια χρήσης, τα οποία προσομοιώνουν συνήθεις ενέργειες των χρηστών στην εφαρμογή:

Σενάριο 1: Περιήγηση στο σύστημα διαχείρισης υπαλλήλων

- Ο χρήστης συνδέεται στο σύστημα και φορτώνει την αρχική σελίδα **(dashboard)**.
- Μετά από 10 δευτερόλεπτα, μεταβαίνει στη σελίδα διαχείρισης υπαλλήλων.
- Μετά από επιπλέον 10 δευτερόλεπτα, επιλέγει έναν υπάλληλο και προβάλλει το προφίλ του.

Σενάριο 2: Διαχείριση αδειών προσωπικού

- Ο χρήστης μεταβαίνει στη σελίδα του ημερολογίου αδειών για να δει τις άδειες του προσωπικού.
- Μετά από 10 δευτερόλεπτα, μεταβαίνει στο προσωπικό ημερολόγιο.
- Μετά από άλλα 10 δευτερόλεπτα, υποβάλλει αίτημα άδειας.
- Μόλις ολοκληρωθεί η ενέργεια, το σύστημα ενημερώνει δυναμικά την προβολή του προσωπικού ημερολογίου, ώστε να αντικατοπτρίζει τη νέα άδεια.

Για την αξιολόγηση της απόδοσης, πραγματοποιήθηκαν δύο κατηγορίες δοκιμών:

Δοκιμές απόκρισης σε ιδανικές συνθήκες

Αρχικά έγινε εκτέλεση των σεναρίων με μόλις ένα χρήστη με σκοπό την εύρεση του μέσου χρόνου απόκρισης του συστήματος, σε ιδανικές συνθήκες, για αυτά τα σενάρια που αποτελούν καθημερινές ενέργειες των χρηστών. Κάθε σενάριο δοκιμάστηκε για 100 επαναλήψεις, ώστε να συγκεντρωθούν αρκετά δεδομένα για ανάλυση.

Δοκιμές φορτίου (Load testing)

Στη συνέχεια πραγματοποιήθηκες δοκιμές φορτίου. Σε αυτή τη περίπτωση πολλοί εικονικοί χρήστες πραγματοποιούσαν τις συγκεκριμένες αλληλουχίες ενεργειών ταυτόχρονα, καταπονώντας το σύστημα με σκοπό την προσομοίωση ωρών αιχμής. Προσομοίωσαν η κυκλοφορία των χρηστών αυξάνοντας σταδιακά το φορτίο σε 100 ταυτόχρονους χρήστες κατά τη διάρκεια μιας περιόδου αύξησης 5 λεπτών.

Κατά τη διάρκεια των δοκιμών, παρακολουθήθηκαν βασικές μετρήσεις όπως ο χρόνος απόκρισης, και η χρήση των πόρων (CPU, μνήμη) με τη χρήση του Azure Monitoring. Αυτές οι μετρήσεις ήταν απαραίτητες για τον εντοπισμό σημείων συμφόρησης της απόδοσης και περιορισμών του συστήματος υπό φορτίο.

3.2.2 Σχέδια δοκιμών στο Apache Jmeter

Οι δοκιμές απόδοσης πραγματοποιήθηκαν με το εργαλείο Apache JMeter, το οποίο επιτρέπει την προσομοίωση φόρτου χρηστών και την καταγραφή κρίσιμων μετρήσεων απόδοσης. Οι δοκιμές στηρίζονται σε σχέδια δοκιμών **(Test Plans)**, τα οποία καθορίζουν τις ενέργειες των εικονικών χρηστών και τις παραμέτρους των δοκιμών.

Ένα σχέδιο δοκιμής στο JMeter αποτελείται από τα εξής βασικά συστατικά:

Ομάδες νημάτων (Thread groups)

Οι ομάδες νημάτων καθορίζουν διάφορα χαρακτηριστικά όπως τον αριθμό των εικονικών χρηστών (virtual users) που συμμετέχουν στη δοκιμή.

Στη συγκεκριμένη μελέτη, διαμορφώθηκαν οι εξής ομάδες νημάτων:

- **Στατική ομάδα νημάτων:** Προσομοίωση ενός χρήστη για τη μέτρηση των χρόνων απόκρισης σε συνθήκες χαμηλού φορτίου.
- **Κλιμακούμενη ομάδα νημάτων:** Αύξηση του αριθμού των χρηστών από 1 έως 100 σε διάστημα 10 λεπτών, προσομοιώνοντας περιόδους αιχμής.

Κάθε εικονικός χρήστης εκτελούσε τα σενάρια της ενότητας 3.2.1, όπως είσοδος στο σύστημα, περιήγηση στο dashboard, προβολή της λίστας των υπαλλήλους και του ημερολογίου αδειών.

Ελεγκτές (Controllers)

Οι ελεγκτές στο JMeter χρησιμοποιούνται για την παραμετροποίηση της ροής των αιτημάτων και την προσομοίωση διαφορετικών συνθηκών χρήσης.

Οι βασικοί ελεγκτές που χρησιμοποιήθηκαν ήταν:

- **Simple Controller:** Ομαδοποίηση σχετικών αιτημάτων ώστε να μετρηθεί ο συνολικός χρόνος εκτέλεσης μιας ενέργειας (π.χ. φόρτωση dashboard).
- **Constant Timer Controller:** Προσθήκη καθυστερήσεων στις ενέργειες των εικονικών χρηστών, ώστε να προσομοιωθεί πιο ρεαλιστικά η ανθρώπινη συμπεριφορά.

Ακροατές (Listeners)

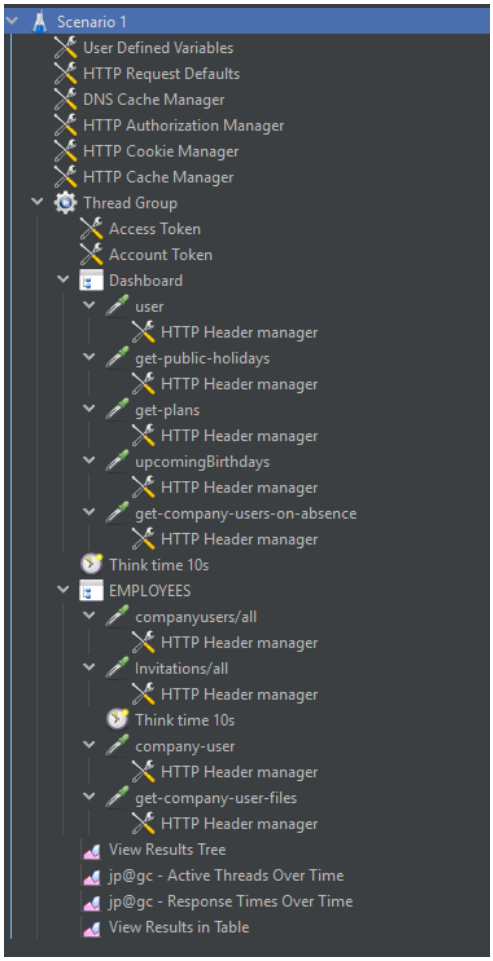
Οι ακροατές καταγράφουν και αναλύουν τα αποτελέσματα των δοκιμών.

Οι βασικοί ακροατές που χρησιμοποιήθηκαν ήταν:

- **View Results Tree:** Ανάλυση της δομής των HTTP απαντήσεων για την ανίχνευση σφαλμάτων και αποκλίσεων.
- **Response Time Graph:** Διάγραμμα των μεταβολών στον χρόνο απόκρισης καθώς αυξάνεται το φορτίο.

3.2.3 Δημιουργία σεναρίου ελέγχου

Για το πρώτο σενάριο χρησιμοποιήσαμε μια ομάδα νημάτων και δύο απλούς ελεγκτές. Ο πρώτος περιλαμβάνει τους δειγματολήπτες για τα αιτήματα που προσομοιώνουν τη φόρτωση της αρχικής σελίδας και ο δεύτερος τους δειγματολήπτες για τα αιτήματα που αφορούν τη φόρτωση της λίστας των υπαλλήλων και την προβολή του προφίλ του συνδεδεμένου χρήστη, όπως βλέπουμε και στην εικόνα παρακάτω.

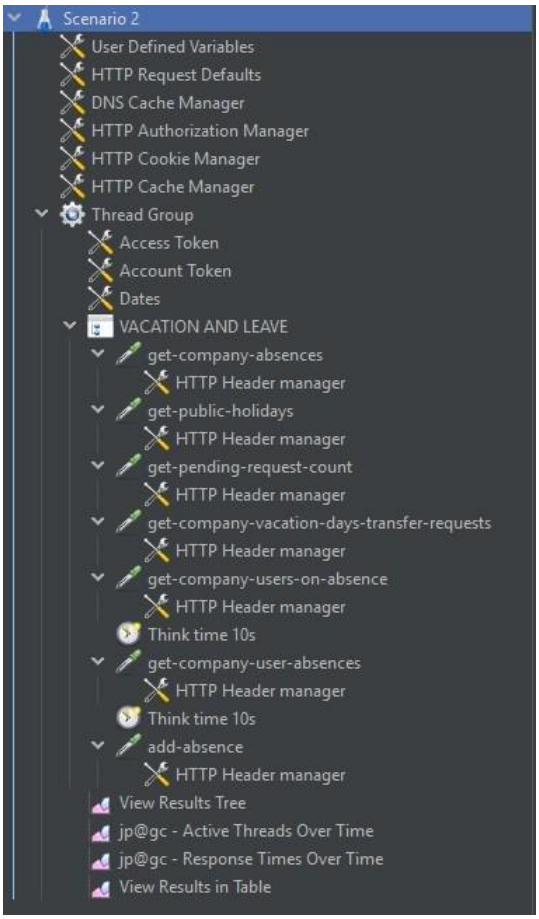


Εικόνα 3.4: Τα βήματα εκτέλεσης του πρώτου σεναρίου στο JMeter

Επιπλέον ενδιάμεσα από αυτούς τους δειγματολήπτες προσθέσαμε παύσεις για να προσομοιώσουμε τον χρόνο σκέψης του χρήστη. Για την εκτέλεση αιτημάτων HTTP στο API της εφαρμογής είναι απαραίτητη η συμπερίληψη του access token και του account token στον header του κάθε αιτήματος για την αυθεντικοποίηση του χρήστη. Τέλος προσθέσαμε τους απαραίτητους ακροατές για την καταγραφή των χρόνων απόκρισης.

Το Dashboard περιλαμβάνει διάφορες αντικείμενα που παρέχουν πληροφορίες στον χρήστη. Το endpoint /user φέρνει τα στοιχεία του χρήστη που έχει συνδεθεί, ενώ το /get-public-holidays επιστρέφει τις επίσημες αργίες. Το endpoint get-plans ανακτά τα διαθέσιμα συνδρομητικά πλάνα **(subscription plans).** Το endpoint /upcomingBirthdays εμφανίζει τους χρήστες που έχουν γενέθλια τις επόμενες 15 ημέρες, ενώ το /get-company-users-on-absence επιστρέφει τη λίστα των υπαλλήλων που βρίσκονται σε άδεια τη συγκεκριμένη ημέρα. Στη σελίδα "People", όπου εμφανίζεται η λίστα των υπαλλήλων, το endpoint /companyusers/all φέρνει όλους τους υπαλλήλους της εταιρείας, ενώ το /invitations/all επιστρέφει τις προσκλήσεις που έχει στείλει ο διαχειριστής μέσω email, ώστε οι υπάλληλοι να μπορούν να κάνουν sign-up. Στην ενότητα του προφίλ χρήστη, το endpoint /company-user ανακτά τις προσωπικές πληροφορίες του χρήστη, ενώ το /get-company-user-files φέρνει τα έγγραφα που έχει ανεβάσει ο χρήστης στο προφίλ του.

Για το δεύτερο σενάριο χρησιμοποιήσαμε μια ομάδα νημάτων **(Thread Group)** που περιλαμβάνει τα απαραίτητα διαπιστευτήρια αυθεντικοποίησης, όπως το Access Token και το Account Token, καθώς και μια μεταβλητή Dates για τον χειρισμό των ημερομηνιών. Το σενάριο επικεντρώνεται στη διαχείριση αδειών και περιλαμβάνει έναν ελεγκτή Vacation And Leave, ο οποίος περιλαμβάνει δειγματολήπτες που εκτελούν HTTP αιτήματα για την προσομοίωση των λειτουργιών του ημερολογίου αδειών της εταιρείας. Στην παρακάτω εικόνα βλέπουμε αναλυτικά το σενάριο.



Εικόνα 3.5: Τα βήματα εκτέλεσης του δεύτερου σεναρίου στο JMeter

Το Vacation and Leave περιλαμβάνει διάφορες λειτουργίες που παρέχουν πληροφορίες σχετικά με τις άδειες του προσωπικού και το ημερολόγιου του κάθε ενός. Το /get-company-absences επιστρέφει τις άδειες της εταιρείας. Το endpoint /get-public-holidays φέρνει τις επίσημες αργίες που εμφανίζονται στο ημερολόγιο. Το endpoint /get-pending-request-count επιστρέφει τον αριθμό των εκκρεμών αιτημάτων αδειών, ενώ το /get-company-vacation-days-transfer-requests φέρνει τα αιτήματα μεταφοράς ημερών άδειας σε άλλο έτος. Επίσης το /get-company-users-on-absence επιστρέφει τα άτομα που βρίσκονται σε άδεια. Ακόμη το /get-company-user-absences επιστρέφει τις άδειας ενός υπαλλήλου. Τέλος το /add-absence προσθέτει άδεια σε έναν υπάλληλο.

3.3 Εφαρμογή βελτιώσεων

Μετά την ανάλυση των αποτελεσμάτων από τις αρχικές δοκιμές, εντοπίστηκαν σημεία συμφόρησης τόσο σε επίπεδο εφαρμογής όσο και σε επίπεδο βάσης δεδομένων, όπως θα δούμε και στο επόμενο κεφάλαιο με τα αποτελέσματα. Για τη βελτίωση των επιδόσεων και της επεκτασιμότητας,

εφαρμόστηκαν τροποποιήσεις στον κώδικα του backend και του frontend, καθώς και βελτιώσεις στην υποδομή και στη διαχείριση της προσωρινής αποθήκευσης **(caching)**.

Οι κύριοι στόχοι των βελτιώσεων ήταν:

- Μείωση του χρόνου απόκρισης των αιτημάτων μέσω βελτιστοποίησης του backend και της βάσης δεδομένων.
- Ελαχιστοποίηση των συνδέσεων στη βάση δεδομένων.
- Βελτίωση της εμπειρίας χρήστη (UX) με αποδοτικότερη φόρτωση δεδομένων και δυναμική διαχείριση των περιεχομένων στο frontend.
- Αποδοτικότερη χρήση των πόρων με την αξιοποίηση caching μηχανισμών.

3.3.1 Βελτιώσεις backend

Οι αλλαγές στο backend επικεντρώθηκαν στην αποδοτικότερη διαχείριση των αιτημάτων προς τη βάση δεδομένων, μειώνοντας τον αριθμό και την πολυπλοκότητα των ερωτημάτων, ενώ ταυτόχρονα βελτιώθηκε η δομή των endpoints για την αποτελεσματικότερη ανάκτηση δεδομένων.

- **Ανασχεδιασμός των endpoints:**

  Αντί να γίνεται ανάκτηση όλων των δεδομένων και εφαρμογή φιλτραρίσματος στο επίπεδο της εφαρμογής, οι κλήσεις προς τη βάση δεδομένων προσαρμόστηκαν ώστε να επιστρέφουν μόνο τα απολύτως απαραίτητα δεδομένα ανάλογα με το εκάστοτε ερώτημα.

- **Ελαχιστοποίηση των κλήσεων στη βάση δεδομένων:**

  Εντοπίστηκε ότι ορισμένα ερωτήματα εκτελούνταν επαναλαμβανόμενα για την ίδια πληροφορία (π.χ. ανάκτηση στατικών δεδομένων όπως λίστες υπαλλήλων ή τύποι αδειών). Τα endpoints ανασχεδιάστηκαν ώστε να μειώνονται οι περιττές κλήσεις στη βάση δεδομένων και να γίνεται καλύτερη χρήση της προσωρινής αποθήκευσης.

- **Βελτίωση της διαχείρισης μεγάλων όγκων δεδομένων:**

  Σε ορισμένες περιπτώσεις, παρατηρήθηκε ότι οι κλήσεις στη βάση δεδομένων γίνονταν εκθετικά αργότερες με την πάροδο του χρόνου, καθώς πρώτα ανακτώνταν όλες οι εγγραφές και στη συνέχεια εφαρμοζόταν φιλτράρισμα στην εφαρμογή. Η βελτίωση επικεντρώθηκε στην εκτέλεση φιλτραρισμένων ερωτημάτων απευθείας στη βάση δεδομένων, ώστε να μην επιβαρύνεται το backend με την επεξεργασία μεγάλου όγκου δεδομένων.

3.3.2 Βελτιώσεις frontend

Οι αλλαγές στο backend απαίτησαν αντίστοιχες τροποποιήσεις στο frontend, τόσο για τη σωστή διαχείριση των νέων API κλήσεων όσο και για τη βελτιστοποίηση της εμπειρίας χρήστη.

Προσαρμογή του frontend στα νέα API endpoints

- **Διαχείριση φιλτραρισμένων δεδομένων:**

  Η αλλαγή στον τρόπο ανάκτησης των αδειών απαιτούσε τροποποίηση του frontend, ώστε να στέλνει σωστά τις παραμέτρους φιλτραρίσματος στα νέα API endpoints και να εμφανίζει μόνο τις σχετικές εγγραφές.

- **Βελτιστοποίηση φόρτωσης περιεχομένου (Priority loading):**

  Για τη βελτίωση της απόδοσης και της απόκρισης της εφαρμογής, εφαρμόστηκε μηχανισμός προτεραιότητας στη φόρτωση δεδομένων. Κρίσιμα στοιχεία όπως το ημερολόγιο αδειών και

λίστα αιτημάτων αδειών φορτώνονται κατά προτεραιότητα, ώστε να εμφανίζονται άμεσα στον χρήστη. Δευτερεύουσες λειτουργίες φορτώνονται στο παρασκήνιο **(lazy loading)**, βελτιώνοντας την αντιλαμβανόμενη ταχύτητα της εφαρμογής.

3.3.3 Caching

Η χρήση προσωρινής αποθήκευσης **(caching)** υιοθετήθηκε ως βασική στρατηγική για τη μείωση του φόρτου στη βάση δεδομένων και τη βελτίωση των χρόνων απόκρισης της εφαρμογής. Μέσω της προσωρινής αποθήκευσης, συχνά ζητούμενα δεδομένα αποθηκεύονται σε προσωρινή μνήμη, επιτρέποντας την ταχύτερη εξυπηρέτησή αιτημάτων χωρίς την ανάγκη επαναλαμβανόμενων ερωτημάτων στη βάση δεδομένων.

Η υλοποίηση του caching πραγματοποιήθηκε σε δύο επίπεδα:

- Caching στη μνήμη του διακομιστή **(In-Memory Caching)**, για την αποθήκευση δεδομένων που δεν μεταβάλλονται συχνά.
- Ενσωμάτωση του Redis ως κατανεμημένου μηχανισμού caching, για την επιτάχυνση της ανάκτησης δεδομένων και τη βελτίωση της επεκτασιμότητας του συστήματος.

Η προσωρινή αποθήκευση στη μνήμη του διακομιστή **(in-memory caching)** αξιοποιεί τη RAM για τη διατήρηση συχνά χρησιμοποιούμενων δεδομένων, μειώνοντας τον αριθμό των αιτημάτων προς τη βάση δεδομένων και επιταχύνοντας την απόκριση του συστήματος. Δεδομένα που δεν μεταβάλλονται συχνά, όπως λίστες επιλογών, ρυθμίσεις συστήματος και προφίλ χρηστών, αποθηκεύονται προσωρινά στη μνήμη του διακομιστή, ενώ χρησιμοποιούνται μηχανισμοί λήξης **(cache expiration)** και ανανέωσης **(cache invalidation)** για την αποφυγή χρήσης λανθασμένων δεδομένων. Τα βασικά οφέλη αυτής της προσέγγισης περιλαμβάνουν τη μείωση του φόρτου στη βάση δεδομένων, την ταχύτερη απόκριση του συστήματος και τη βελτιωμένη εμπειρία χρήστη μέσω ταχύτερης φόρτωσης σελίδων και λειτουργιών. Παράλληλα, για τη βελτίωση της απόδοσης και της επεκτασιμότητας, ενσωματώθηκε το Redis ως κατανεμημένο σύστημα προσωρινής αποθήκευσης. Το Redis, ένα υψηλής απόδοσης in-memory key-value store, επιτρέπει την ταχεία αποθήκευση και ανάκτηση δεδομένων, περιορίζοντας την ανάγκη για επαναλαμβανόμενες προσβάσεις στη βάση δεδομένων. Αποθηκεύοντας τα δεδομένα σε δομές key-value, διευκολύνει την άμεση ανάκτησή τους χωρίς περίπλοκες αναζητήσεις, ενώ ως κατανεμημένη κρυφή μνήμη **(distributed cache)** επιτρέπει τη μοιρασμένη χρήση της μνήμης σε πολλαπλούς διακομιστές της εφαρμογής. Επιπλέον, η ενσωμάτωση μηχανισμών αυτόματης ανανέωσης δεδομένων διασφαλίζει ότι η προσωρινή αποθήκευση παραμένει συγχρονισμένη με την κύρια βάση δεδομένων. Η χρήση του Redis έχει οδηγήσει σε αυξημένη επεκτασιμότητα, διατηρώντας χαμηλούς χρόνους απόκρισης ακόμα και υπό υψηλό φορτίο, ταχύτερη επεξεργασία αιτημάτων, αφού η ανάκτηση δεδομένων από το Redis είναι σημαντικά πιο γρήγορη από τις κλασικές SQL αναζητήσεις, καθώς και βελτιστοποίηση της χρήσης των πόρων του συστήματος, με μειωμένη ανάγκη για επεξεργαστική ισχύ και I/O λειτουργίες στη βάση δεδομένων.

3.4 Έλεγχος μετά τις βελτιώσεις

Μετά την εφαρμογή των βελτιώσεων στην αρχιτεκτονική, τον κώδικα και την υποδομή της εφαρμογής, πραγματοποιήθηκαν νέες δοκιμές απόδοσης για την αξιολόγηση της επίδρασης των αλλαγών. Οι δοκιμές αυτές ακολούθησαν την ίδια μεθοδολογία και τα ίδια σενάρια που χρησιμοποιήθηκαν στις αρχικές μετρήσεις, ώστε να διασφαλιστεί η συγκρισιμότητα των αποτελεσμάτων. Στο επόμενο κεφάλαιο θα αναλυθούν οι χρόνοι των δοκιμών που έχουν γίνει πριν και μετά τις βελτιώσεις.

Κεφάλαιο 4ο: Αποτελέσματα

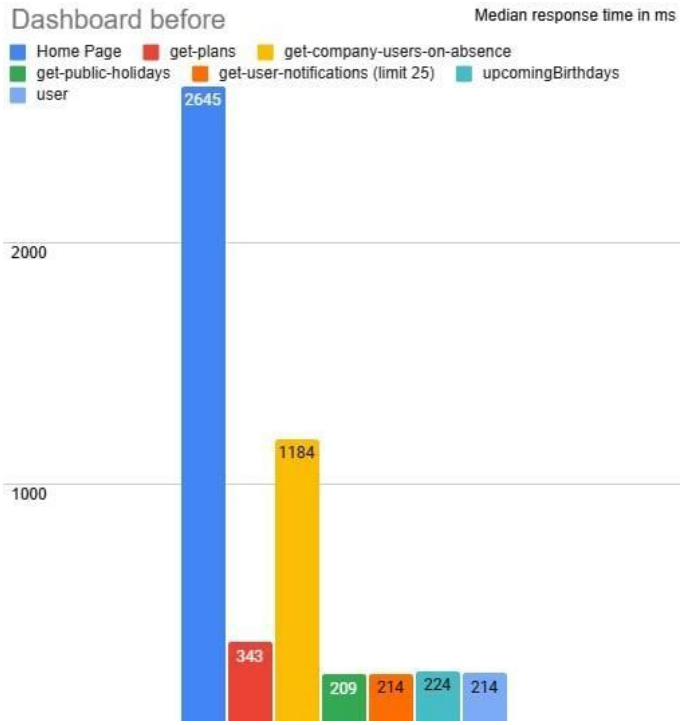
Στο παρόν κεφάλαιο παρουσιάζονται τα αποτελέσματα των δοκιμών απόδοσης που πραγματοποιήθηκαν πριν και μετά την εφαρμογή των βελτιώσεων στο σύστημα. Στόχος της ανάλυσης είναι η ποσοτικοποίηση της επίδρασης των βελτιστοποιήσεων και η αξιολόγηση της συνολικής απόδοσης της εφαρμογής.

4.1 Αρχικά αποτελέσματα

Στην ενότητα αυτή παρουσιάζονται οι μετρήσεις από τις αρχικές δοκιμές απόδοσης, πριν από την εφαρμογή οποιασδήποτε βελτίωσης στο σύστημα. Οι βασικές μετρικές που εξετάστηκαν αφορούν τον χρόνο φόρτωσης της αρχικής σελίδας, όπου αξιολογήθηκε η ταχύτητα απόκρισης του συστήματος κατά την πρώτη είσοδο του χρήστη στην εφαρμογή. Τον χρόνο απόκρισης της σελίδας υπαλλήλων, με έμφαση στον χρόνο που απαιτείται για τη φόρτωση της λίστας των εργαζομένων. Τέλος τον χρόνο απόκρισης του ημερολογίου αδειών, όπου μετρήθηκε ο χρόνος που απαιτείται για την ανάκτηση και εμφάνιση των δεδομένων αδειών και απουσιών καθώς και του χρόνου που απαιτείται για την προσθήκη μιας άδειας.

4.1.1 Ανάλυση αποτελεσμάτων από την αρχική σελίδα

Το διάγραμμα που παρέχεται είναι ένα ραβδόγραμμα **(bar chart)** που απεικονίζει τους διάμεσους χρόνους απόκρισης **(median response times)** για διάφορα αιτήματα **(requests)** σε χιλιοστά του δευτερολέπτου **(ms)**. Παρακάτω βλέπουμε το διάγραμμα πριν πραγματοποιηθούν οι βελτιώσεις.



Εικόνα 4.1: Αποτελέσματα δοκιμών στην αρχική σελίδα

Η ανάλυση του διαγράμματος απόκρισης μας παρέχει πληροφορίες σχετικά με τους διάμεσους χρόνους απόκρισης για διάφορα αιτήματα. Ο κατακόρυφος άξονας **(Y-Axis)** αντιπροσωπεύει τον διάμεσο χρόνο απόκρισης για κάθε αίτημα, με τις υψηλότερες στήλες να δείχνουν μεγαλύτερους χρόνους

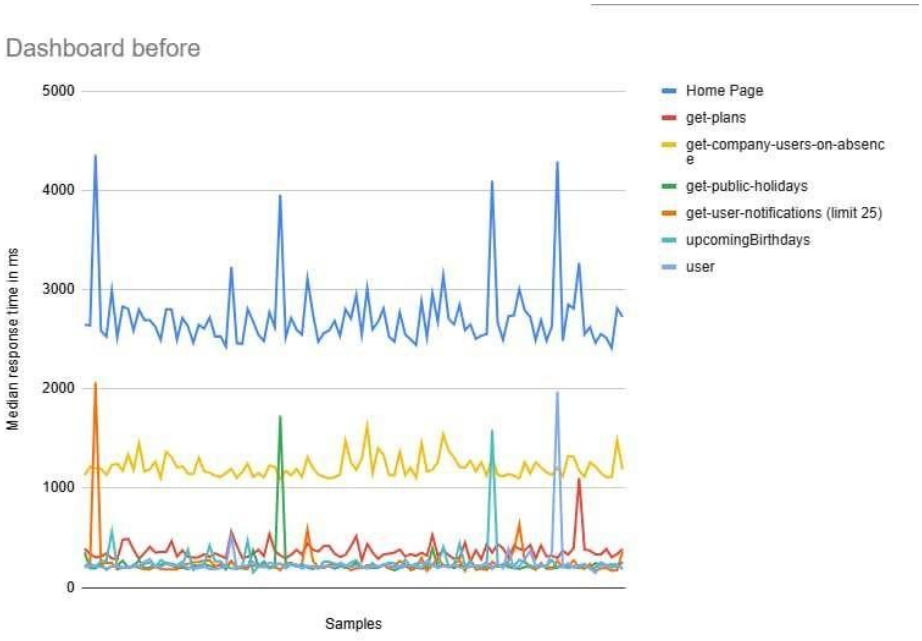
επεξεργασίας. Ο οριζόντιος άξονας **(X-Axis)** αντιστοιχεί στα διαφορετικά αιτήματα που εκτελέστηκαν κατά τη δοκιμή απόδοσης.

Τα αιτήματα που περιλαμβάνονται είναι το **get-plans** με κίτρινο, το **get-company-users-on-absence** με πορτοκαλί, το **get-public-holidays** με πράσινο, το **get-user-notifications** με γαλάζιο, το **upcomingBirthdays** με σκούρο μπλε και το **user** με κόκκινο. Με μπλε φαίνεται ο συνολικός χρόνος που απαιτείται για να φορτώσει η αρχική σελίδα. Η ανάλυση αυτών των δεδομένων μάς επιτρέπει να εντοπίσουμε πιθανά σημεία συμφόρησης και να σχεδιάσουμε στρατηγικές βελτιστοποίησης για τις πιο αργές κλήσεις.

ΑΙΤΗΜΑ	ΔΙΑΜΕΣΟΣ ΧΡΟΝΟΣ ΑΠΟΚΡΙΣΗΣ (MS)
GET-PLANS	343 ms
GET-COMPANY-USERS-ON-ABSENCE	1184 ms (υψηλότερος χρόνος)
GET-PUBLIC-HOLIDAYS	209 ms
GET-USER-NOTIFICATIONS	214 ms
UPCOMINGBIRTHDAYS	224 ms
USER	214 ms

Το αίτημα **get-company-users-on-absence** παρουσιάζει τον μεγαλύτερο διάμεσο χρόνο απόκρισης (1184ms), γεγονός που υποδηλώνει πιθανό πρόβλημα απόδοσης και ανάγκη βελτιστοποίησης. Τα υπόλοιπα αιτήματα **(get-plans, get-public-holidays, get-user-notifications, upcomingBirthdays, user)** έχουν πολύ χαμηλότερους χρόνους (209-343ms), κάτι που δείχνει ότι είναι πιο αποδοτικά.

Για την πλήρη ανάλυση των χρόνων απόκρισης θα αναλύσουμε και το γραμμικό διάγραμμα που βλέπουμε παρακάτω.



Εικόνα 4.2: Αποτελέσματα δοκιμών στην αρχική σελίδα

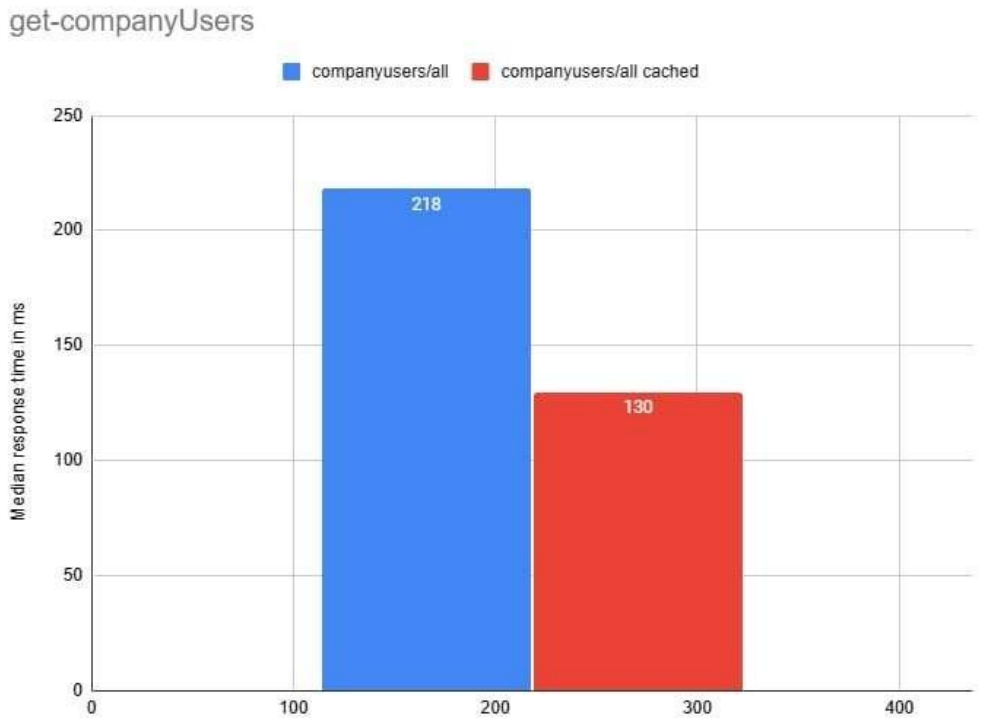
Ο Οριζόντιος άξονας **(X-Axis)** – Δείγματα **(Samples)** αντιπροσωπεύει τον αριθμό των αιτημάτων που εκτελέστηκαν κατά τη διάρκεια της δοκιμής. Κάθε σημείο στον άξονα αντιστοιχεί σε μια μεμονωμένη κλήση ή επανάληψη του τεστ.

Ο Κατακόρυφος άξονας **(Y-Axis)** – Διάμεσος Χρόνος Απόκρισης **(ms)** δείχνει τον διάμεσο χρόνο απόκρισης για κάθε αίτημα, μετρημένο σε χιλιοστά του δευτερολέπτου **(ms)**. Η διάμεση τιμή είναι χρήσιμη, καθώς δεν επηρεάζεται από ακραίες τιμές **(outliers)** και δίνει μια πιο ξεκάθαρη εικόνα της απόδοσης.

Το /get-company-users-on-absence εμφανίζει τον υψηλότερο διάμεσο χρόνο απόκρισης, γεγονός που υποδηλώνει ότι μπορεί να χρειάζεται βελτιστοποίηση για ταχύτερη φόρτωση. Αντίθετα, τα υπόλοιπα endpoints παρουσιάζουν γενικά χαμηλότερους και πιο σταθερούς χρόνους απόκρισης. Ωστόσο, παρατηρούνται αιφνίδιες αυξήσεις στους χρόνους απόκρισης για ορισμένα αιτήματα, κάτι που ενδέχεται να υποδηλώνει περιόδους υψηλού φόρτου ή συγκεκριμένα προβλήματα που απαιτούν περαιτέρω διερεύνηση.

4.1.2 Ανάλυση αποτελεσμάτων από την σελίδα των υπαλλήλων

Το διάγραμμα που παρέχεται είναι ένα ραβδόγραμμα **(bar chart)** το οποίο απεικονίζει τους διάμεσους χρόνους απόκρισης **(median response times)** για το αίτημα /get-company-user**s,** που φέρνει τις απαραίτητες πληροφορίες για την λίστα των υπαλλήλων, σε χιλιοστά του δευτερολέπτου **(ms)**. Παρακάτω βλέπουμε το διάγραμμα πριν και μετά από τις βελτιώσεις. Το μπλε χρώμα αφορά τους χρόνους απόκρισης πριν τις βελτιώσεις και το κόκκινο χρώμα μετά τις βελτιώσεις.



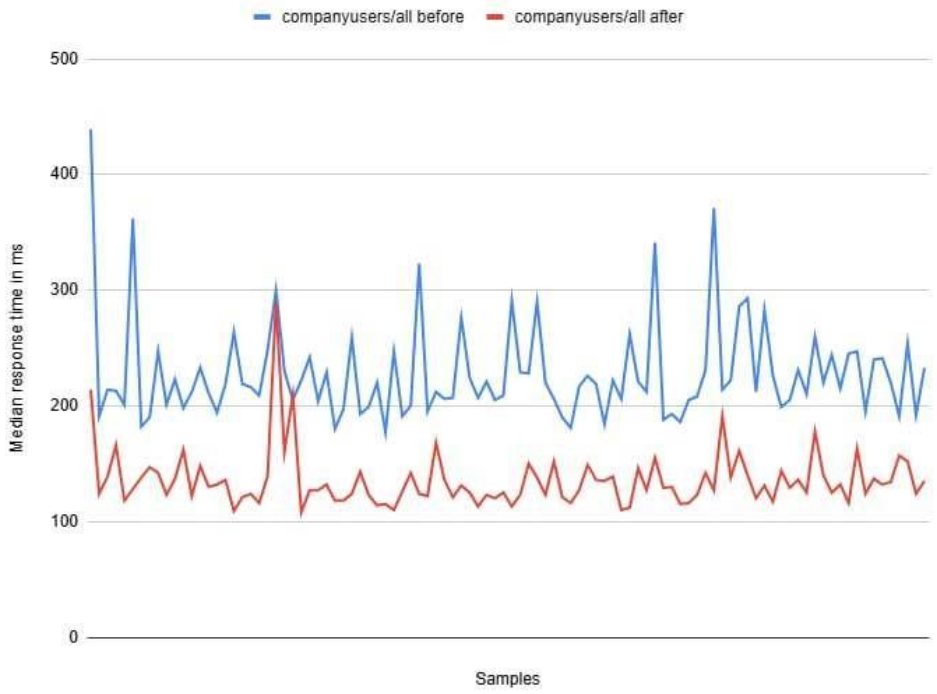
Εικόνα 4.3: Αποτελέσματα του πίνακα υπαλλήλων

Η ανάλυση του διαγράμματος απόκρισης μας παρέχει πληροφορίες σχετικά με τους διάμεσους χρόνους απόκρισης για διάφορα αιτήματα. Ο κατακόρυφος άξονας (Y-Axis) αντιπροσωπεύει τον διάμεσο χρόνο απόκρισης για κάθε αίτημα, με τις υψηλότερες στήλες να δείχνουν μεγαλύτερους χρόνους

επεξεργασίας. Ο οριζόντιος άξονας (X-Axis) αντιστοιχεί στα διαφορετικά αιτήματα που εκτελέστηκαν κατά τη δοκιμή απόδοσης.

Ο χρόνος απόκρισης για την αρχική κλήση του αιτήματος πριν τις βελτιώσεις (companyusers/all) ήταν 218 ms, ενώ μετά τις βελτιώσεις και τη χρήση προσωρινής μνήμης ο ενδιάμεσος χρόνος απόκρισης είναι 130ms.

Για την πλήρη ανάλυση των χρόνων απόκρισης θα αναλύσουμε και το γραμμικό διάγραμμα που βλέπουμε παρακάτω.



Εικόνα 4.4: Αποτελέσματα του πίνακα υπαλλήλων

Ο Οριζόντιος άξονας **(X-Axis)** – Δείγματα **(Samples)** αντιπροσωπεύει τον αριθμό των αιτημάτων που εκτελέστηκαν κατά τη διάρκεια της δοκιμής. Κάθε σημείο στον άξονα αντιστοιχεί σε μια μεμονωμένη κλήση ή επανάληψη του τεστ.

Ο Κατακόρυφος άξονας **(Y-Axis)** – Διάμεσος Χρόνος Απόκρισης **(ms)** δείχνει τον διάμεσο χρόνο απόκρισης για κάθε αίτημα, μετρημένο σε χιλιοστά του δευτερολέπτου **(ms)**. Η διάμεση τιμή είναι χρήσιμη, καθώς αγνοεί ακραίες τιμές **(outliers)** και δίνει μια πιο ξεκάθαρη εικόνα της απόδοσης.

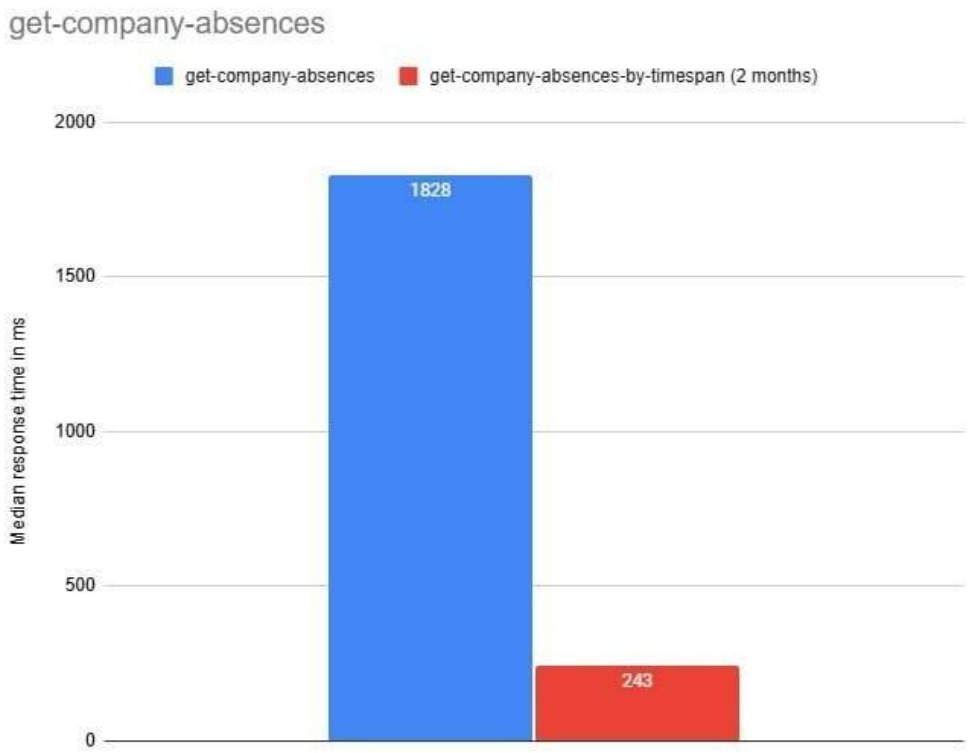
Η μπλε γραμμή (πριν τις βελτιώσεις) δείχνει ότι οι χρόνοι απόκρισης ήταν σχετικά υψηλοί και υπήρχε σημαντική διακύμανση, γεγονός που υποδεικνύει ότι το σύστημα δεν ήταν βελτιστοποιημένο για τις συγκεκριμένες εργασίες και υπήρχαν σημαντικά σημεία συμφόρησης. Η αιτία για τους υψηλούς χρόνους απόκρισης και τη σημαντική διακύμανση στην μπλε γραμμή μπορεί να προκύπτει από διάφορους παράγοντες, οι οποίοι ενδέχεται να σχετίζονται με την αρχιτεκτονική του συστήματος, ή από την διαχείριση της βάσης δεδομένων.

Η κόκκινη γραμμή (μετά τις βελτιώσεις) δείχνει πρώτον πολύ χαμηλότερους χρόνους απόκρισης και δεύτερον πιο σταθερούς χρόνους απόκρισης. Αυτό οφείλεται στη χρήση προσωρινής μνήμης, διότι τα δεδομένα υπάρχουν ήδη αποθηκευμένα οπότε δεν υπάρχει επικοινωνία με την βάση και δεν χρειάζεται

να γίνουν περεταίρω υπολογισμοί με αποτέλεσμα ο χρόνος απόκρισης να μην επηρεάζεται απο τον όγκο των δεδομένων και να παραμένει παρόμοια επίπεδα.

4.1.3 Ανάλυση αποτελεσμάτων από την σελίδα του ημερολογίου αδειών

Το διάγραμμα που παρέχεται είναι επίσης ραβδόγραμμα **(bar chart)** και απεικονίζει τους διάμεσους χρόνους απόκρισης **(median response times)** για το αίτημα get-company-absences σε χιλιοστά του δευτερολέπτου **(ms)**. Παρακάτω βλέπουμε το διάγραμμα πριν και μετά από τις βελτιώσεις. Το μπλε χρώμα αφορά τους χρόνους απόκρισης πριν τις βελτιώσεις και το κόκκινο χρώμα μετά τις βελτιώσεις



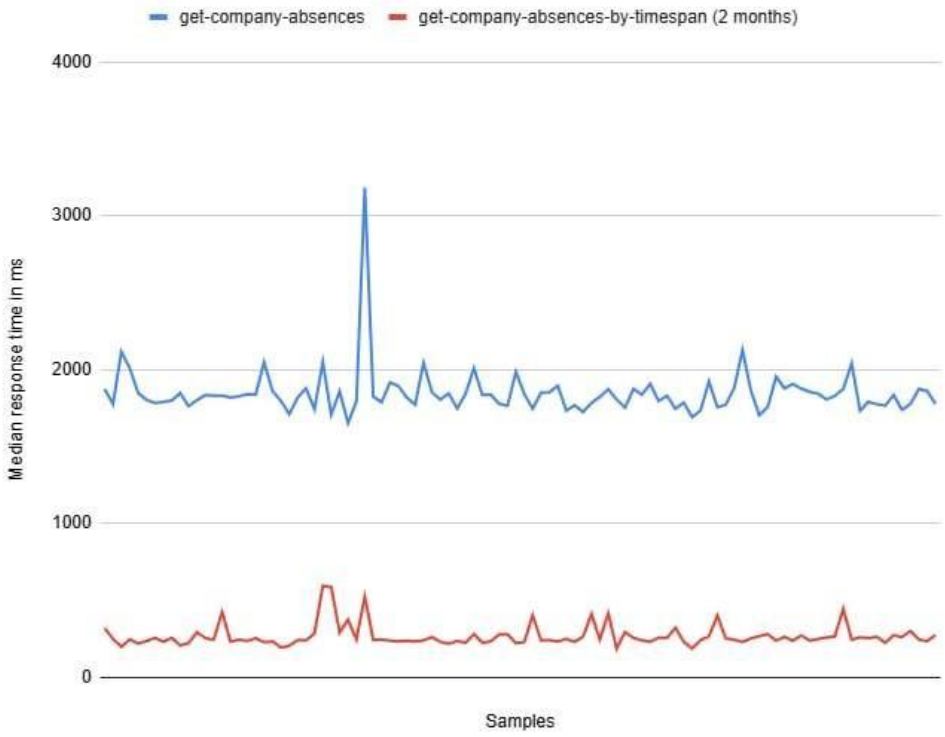
Εικόνα 4.5: Αποτελέσματα του ημερολογίου αδειών

Το μπλε κελί (πριν τις βελτιώσεις) δείχνει ότι οι χρόνοι απόκρισης ήταν σχετικά υψηλοί γεγονός που υποδεικνύει ότι το σύστημα δεν ήταν βελτιστοποιημένο για τις συγκεκριμένες εργασίες και υπήρχαν σημαντικά σημεία συμφόρησης. Η αιτία για τους υψηλούς χρόνους απόκρισης και της σημαντικής διακύμανσης του μπλε κελιού (πριν τις βελτιώσεις) μπορεί να προκύπτει από διάφορους παράγοντες, οι οποίοι ενδέχεται να σχετίζονται με την αρχιτεκτονική του συστήματος ή τη διαχείριση της βάσης δεδομένων. Κάποιες αιτίες μπορεί να είναι οι εξής:

- **Πολλά ή περίπλοκα ερωτήματα:** Πολλά ερωτήματα στη βάση δεδομένων σημαίνει ότι αυξάνεται και ο αριθμός των απαραίτητων συνδέσεων σε αυτή με αποτέλεσμα οι χρόνοι απόκρισης να αυξάνονται.
- **Αναζήτηση σε μεγάλο όγκο δεδομένων:** Αν το σύστημα φορτώνει μεγάλο όγκο δεδομένων χωρίς να περιορίζει τα αποτελέσματα (π.χ., ανακτώντας όλα τα δεδομένα αντί για συγκεκριμένα αποτελέσματα με φίλτρα), η απόδοση θα μπορούσε να είναι αργή και να προκαλεί υπερφόρτωση στη βάση δεδομένων.

- **Ανεπαρκείς υποδομές εξυπηρετητή:** Αν οι υποδομές (π.χ., CPU, μνήμη) του εξυπηρετητή δεν είναι επαρκείς, αυτό θα μπορούσε να προκαλέσει συμφόρηση και καθυστέρηση στην εκπλήρωση αιτημάτων.
- **Μή αποδοτικός κώδικας:** Αν ο κώδικας της εφαρμογής δεν έχουν σχεδιαστεί για να ανταποκρίνεταί γρήγορα σε συγκεκριμένα αιτήματα, η εφαρμογή θα μπορούσε να καθυστερεί περισσότερο από ότι χρειάζεται. Για παράδειγμα, αν γίνεται περιττή επεξεργασία πριν την αποστολή των δεδομένων στον χρήστη, αυτό θα μπορούσε να προκαλέσει καθυστερήσεις.
- **Απουσία ή ανεπαρκές caching:** Η μη υπάρξη μηχανισμού caching θα προκαλούσε συμφόρηση και καθυστέρησει, διότι θα έπρεπε να κάνει επαναλαμβανόμενα αιτήματα προς τη βάση δεδομένων για τα ίδια δεδομένα

Για την πλήρη ανάλυση των χρόνων απόκρισης θα αναλύσουμε και το γραμμικό διάγραμμα που βλέπουμε παρακάτω.



Εικόνα 4.6: Αποτελέσματα του ημερολογίου αδειών

Όπως και στα προηγούμενα διαγράμματα, ο οριζόντιος άξονας **(X-Axis)** – Δείγματα **(Samples)** αντιπροσωπεύει τον αριθμό των αιτημάτων που εκτελέστηκαν κατά τη διάρκεια της δοκιμής. Κάθε σημείο στον άξονα αντιστοιχεί σε μια μεμονωμένη κλήση ή επανάληψη του τεστ. Αντίθετα ο κατακόρυφος άξονας **(Y-Axis)** – Διάμεσος Χρόνος Απόκρισης **(ms)** δείχνει τον διάμεσο χρόνο απόκρισης για κάθε αίτημα, μετρημένο σε χιλιοστά του δευτερολέπτου **(ms)**. Η διάμεση τιμή είναι χρήσιμη, καθώς αγνοεί ακραίες τιμές **(outliers)** και δίνει μια πιο ξεκάθαρη εικόνα της απόδοσης.

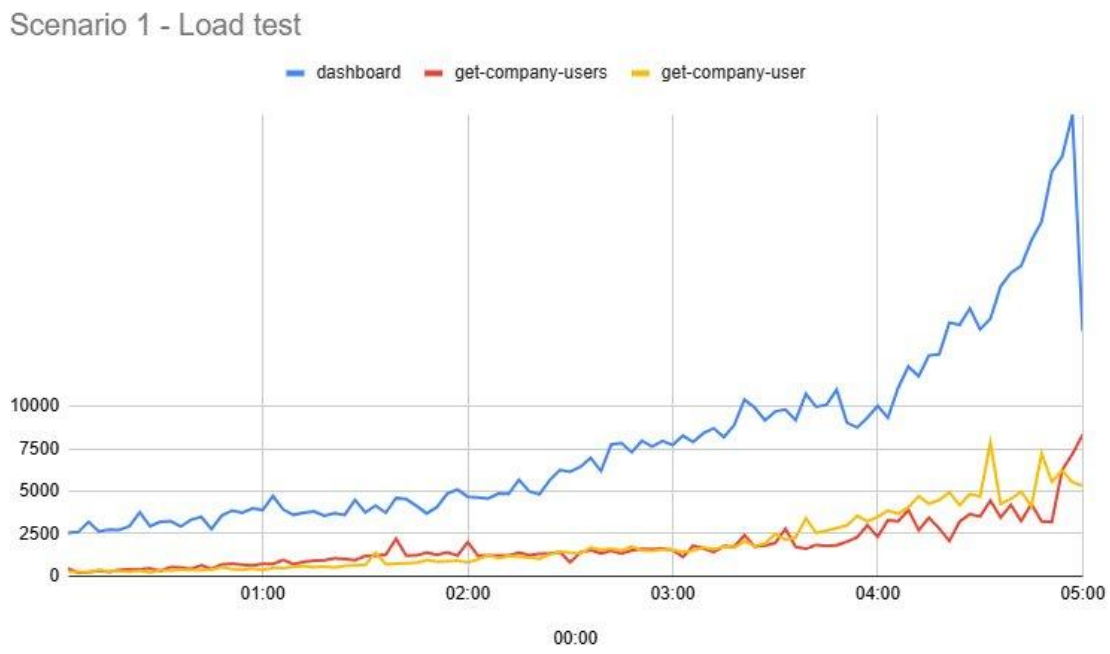
Οι χρόνοι απόκρισης παραμένουν σε σχετικά σταθερό επίπεδο, με εξαίρεση ένα μικρό διάστημα υψηλής καθυστέρησης, δείχνοντας ότι η προσέγγιση αυτή διαχειρίζεται αποδοτικά τις αιτήσεις. Ωστόσο, το γεγονός ότι ανακτά όλα τα δεδομένα αδειών χωρίς περιορισμό ενδέχεται να επιβαρύνει τη βάση δεδομένων, ειδικά σε περιπτώσεις με μεγάλο όγκο δεδομένων.

4.1.4 Ανάλυση αποτελεσμάτων δοκιμών φορτίου

Όπως περιγράψαμε νωρίτερα, πέρα από τις δοκιμές σε μεμονωμένα endpoint, εκτελέσαμε δοκιμές φόρτου σε δύο διαφορετικά σενάρια προσομοιώνοντας με τον καλύτερο δυνατό τρόπο πραγματικές συνθήκες και ρεαλιστική χρήση της εφαρμογής από χρήστες. Οι δοκιμές και για τα δύο σενάρια είχαν διάρκεια πέντε λεπτών, κατά την διάρκεια των οποίων ο αριθμός των εικονικών χρηστών αυξανόταν γραμμικά από μηδέν σε εκατό χρήστες, προσομοιώνοντας σταδιακή επιβάρυνση του συστήματος.

Σενάριο 1

Το πρώτο σενάριο αφορά την φόρτωση της αρχικής σελίδας (dashboard), της λίστας των υπαλλήλων και τέλος την περιήγηση του χρήστη στο προφίλ του. Για την ανάλυση των αποτελεσμάτων θα χρησιμοποιήσουμε την παρακάτω εικόνα.



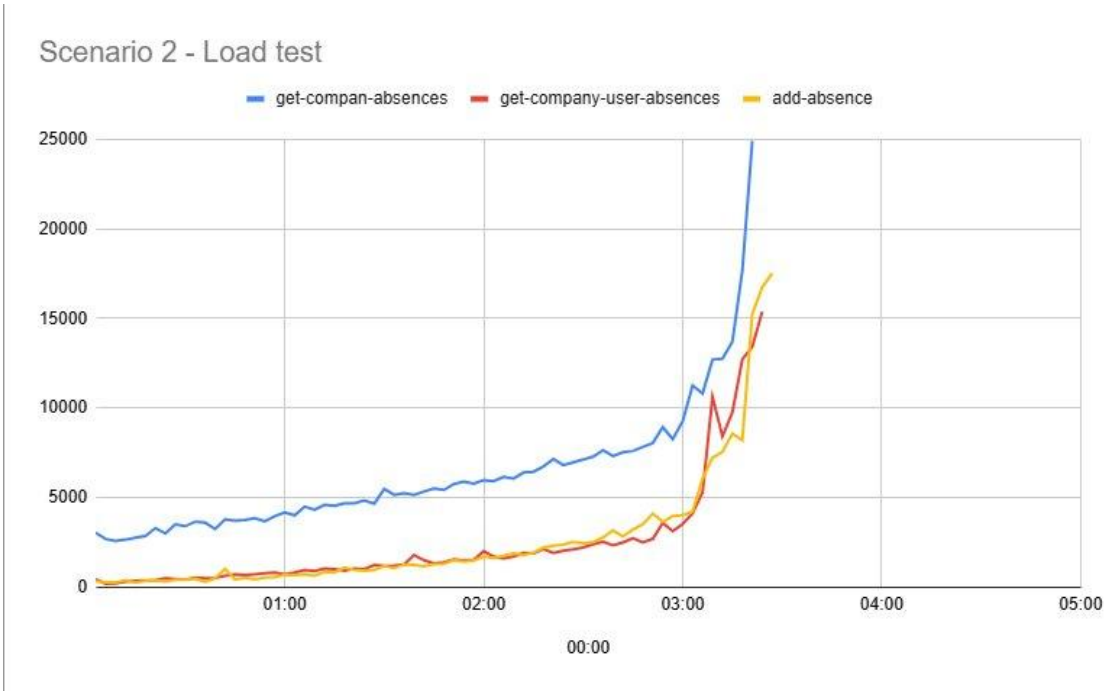
Εικόνα 4.7: Αποτελέσματα του πρώτου σεναρίου δοκιμών φορτίου

Ο οριζόντιος άξονας **(X-Axis)** αντιπροσωπεύει τον χρόνο, σε λεπτά, από την έναρξη της δοκιμής. Ο κατακόρυφος άξονας **(Y-Axis)** δείχνει τον διάμεσο χρόνο απόκρισης για κάθε αίτημα, μετρημένο σε χιλιοστά του δευτερολέπτου **(ms)**.

Παρατηρούμε ότι όσο αυξάνονται οι εικονικοί χρήστες, τόσο χειρότερη γίνεται η απόδοση του συστήματος. Πιο συγκεκριμένα, μετά την πάροδο δύο λεπτών βλέπουμε τους χρόνους απόκρισης να έχουν διπλασιαστεί και μετά την πάροδο τριών λεπτών να φτάνουν σε επίπεδα που καθιστούν την εφαρμογή μη ανταποκρίσιμη. Ιδιαίτερα, μετά από τέσσερα λεπτά δοκιμών βλέπουμε εκθετική αύξηση των χρόνων απόκρισης και ραγδαία αύξηση των σφαλμάτων.

Σενάριο 2

Το δεύτερο σενάριο εστιάζει στην απόδοση της σελίδας του ημερολογίου αδειών, του προσωπικού ημερολογίου και της προσθήκης μιας καινούριας άδειας. Για την ανάλυση των αποτελεσμάτων θα χρησιμοποιήσουμε την παρακάτω εικόνα.



Εικόνα 4.8: Αποτελέσματα του δεύτερου σεναρίου δοκιμών φορτίου

Ο οριζόντιος άξονας **(X-Axis)** αντιπροσωπεύει τον χρόνο, σε λεπτά, από την έναρξη της δοκιμής. Ο κατακόρυφος άξονας **(Y-Axis)** δείχνει τον διάμεσο χρόνο απόκρισης για κάθε αίτημα, μετρημένο σε χιλιοστά του δευτερολέπτου **(ms)**.

Όπως και στο προηγούμενο σενάριο, παράλληλα με την αύξηση των εικονικών χρηστών, παρατηρούμε και ραγδαία αύξηση των χρόνων απόκρισης του συστήματος. Συγκεκριμένα, μετά από τα πρώτα δύο λεπτά της δοκιμής βλέπουμε τους χρόνους απόκρισης να έχουν διπλασιαστεί, ενώ μετά τα πρώτα τρία λεπτά παρατηρούμε εκθετική αύξηση των χρόνων απόκρισης και των σφαλμάτων με την εφαρμογή να είναι μη ανταποκρίσιμη. Τέλος, περίπου τριάμισι λεπτά από την έναρξη της δοκιμής, το σύστημα υπέστη κατάρρευση (crash) λόγω του αυξημένου φόρτου από τον αριθμό των εικονικών χρηστών και των αιτημάτων. Η εφαρμογή σταμάτησε να ανταποκρίνεται εντελώς και δεν μπορούσε να εξυπηρετήσει τους χρήστες, οπότε ήταν αναγκαστική και η διακοπή της δοκιμής.

4.2 Αποτελέσματα μετά τις βελτιώσεις

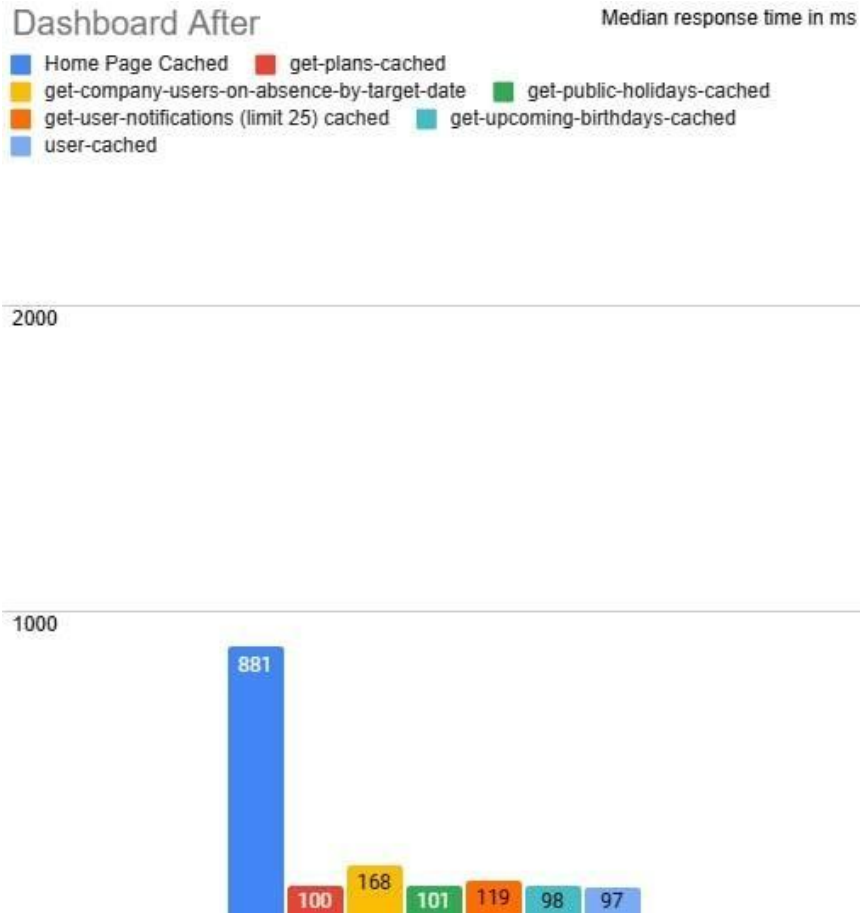
Στην ενότητα αυτή παρουσιάζονται οι μετρήσεις απόδοσης που καταγράφηκαν μετά την εφαρμογή των βελτιώσεων στο σύστημα. Οι ίδιες μετρικές που αναλύθηκαν στις αρχικές δοκιμές επανεξετάζονται, ώστε να αξιολογηθεί η επίδραση των αλλαγών στην απόδοση της εφαρμογής.

Συγκεκριμένα, εξετάζεται ο χρόνος φόρτωσης της αρχικής σελίδας, ώστε να διαπιστωθεί αν η ταχύτητα απόκρισης έχει βελτιωθεί σε σύγκριση με τις αρχικές δοκιμές. Επιπλέον, αναλύεται ο χρόνος απόκρισης της σελίδας υπαλλήλων, με στόχο την αξιολόγηση της ταχύτερης ανάκτησης και προβολής της λίστας εργαζομένων. Τέλος, μελετάται ο χρόνος απόκρισης του ημερολογίου αδειών, εστιάζοντας στη βελτίωση της διαδικασίας ανάκτησης και παρουσίασης των δεδομένων απουσιών.

Μέσα από αυτή την ανάλυση, επιδιώκεται η σύγκριση των επιδόσεων πριν και μετά τις βελτιώσεις, προκειμένου να προσδιοριστεί ο βαθμός στον οποίο οι αλλαγές συνέβαλαν στη συνολική βελτίωση της εμπειρίας χρήστη και της απόδοσης του συστήματος.

4.2.1 Ανάλυση του διαγράμματος απόκρισης από το dashboard

Παρακάτω βλέπουμε πως οι περισσότεροι χρόνοι απόκρισης έχουν μειωθεί σημαντικά μετά την εφαρμογή caching και άλλων βελτιώσεων.

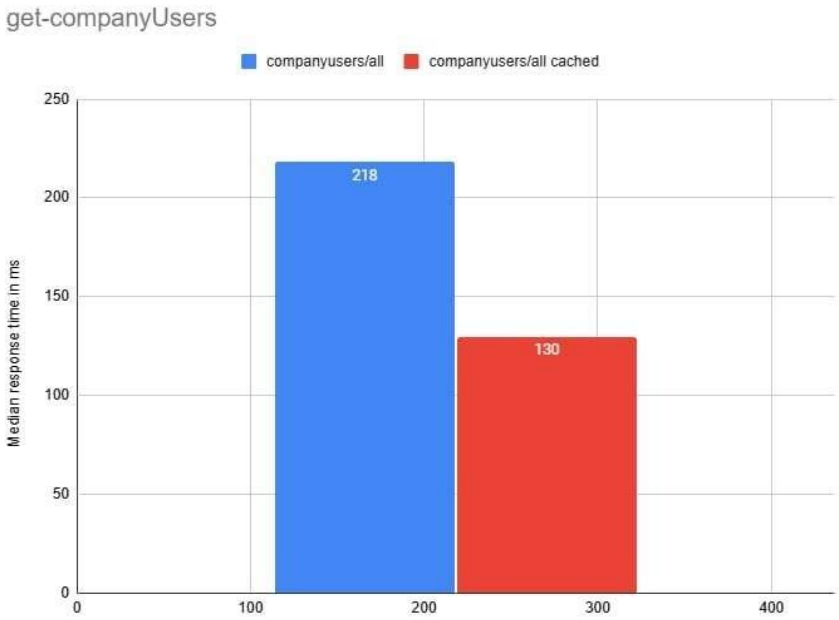


Εικόνα 4.9: Αποτελέσματα της αρχικής σελίδας μετά τις βελτιώσεις

Οι περισσότερες κλήσεις παρουσίασαν σημαντική μείωση στον χρόνο απόκρισης μετά τις βελτιώσεις. Το **get-company-users-on-absence**, που ήταν το πιο αργό αίτημα (1184 ms), φαίνεται να έχει βελτιωθεί σημαντικά χάρη στη χρήση caching, καθιστώντας τη φόρτωση δεδομένων πιο αποδοτική (168 ms). Τα υπόλοιπα αιτήματα παρότι ήταν ήδη χαμηλά παρατηρούμε εξίσου βελτίωση όχι μόνο στο χρόνο απόκρισης, αλλά και στην σταθερότητά των χρόνων.

4.2.2 Ανάλυση του διαγράμματος απόκρισης από την σελίδα των υπαλλήλων

Εξίσου στην ανάλυση του διαγράμματος απόκρισης της σελίδας των εργαζομένων παρατηρούμε βελτίωση στον χρόνο απόκρισης συγκριτικά με τα αρχικά αποτελέσματα, όπως φαίνεται και παρακάτω.

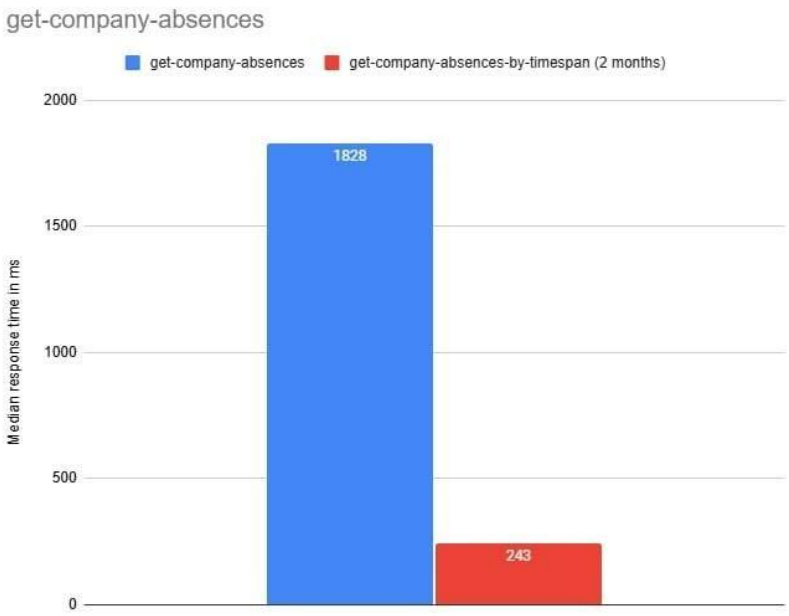


Εικόνα 4.10: Αποτελέσματα του πίνακα υπαλλήλων μετά τις βελτιώσεις

Ο χρόνος απόκρισης για την αρχική κλήση (companyusers/all) ήταν 218 ms, όταν η ανάκτηση των δεδομένων πραγματοποιούνταν απευθείας από τη βάση. Με την εφαρμογή του caching, ο χρόνος απόκρισης μειώθηκε στα 130 ms, καταγράφοντας μια βελτίωση κατά 88 ms. Η μείωση αυτή οφείλεται στο γεγονός ότι τα δεδομένα δεν χρειάζεται πλέον να ανακτηθούν από τη βάση σε κάθε αίτημα, αλλά αντλούνται απευθείας από την προσωρινή μνήμη **(cache)**.

4.2.3 Ανάλυση του διαγράμματος απόκρισης από την σελίδα του ημερολογίου αδειών

Εδώ παρατηρείται αισθητή βελτίωση χρόνων όπως θα δούμε παρακάτω το διάγραμμα, αυτό οφείλεται στην επιστροφή αδειών σε συγκεκριμένη χρονική περίοδο, αντί όλων των δεδομένων.



Εικόνα 4.11: Αποτελέσματα του ημερολογίου αδειών μετά τις βελτιώσεις

Με την εφαρμογή του φιλτραρίσματος βάσει χρονικού διαστήματος, στόχος ήταν η μείωση του όγκου των δεδομένων που επιστρέφονται και, κατά συνέπεια, η βελτίωση των χρόνων απόκρισης. Η νέα προσέγγιση διασφαλίζει ότι το σύστημα επιστρέφει δεδομένα μόνο για την καθορισμένη χρονική περίοδο, αντί για το σύνολο των διαθέσιμων εγγραφών. Με αυτό τον τρόπο καταφέραμε απο (1828 ms) να ρίξουμε τον χρόνο απόκρισης στα (243 ms).

4.2.4 Ανάλυση αποτελεσμάτων δοκιμών φορτίου μετά τις βελτιώσεις

Σενάριο 1

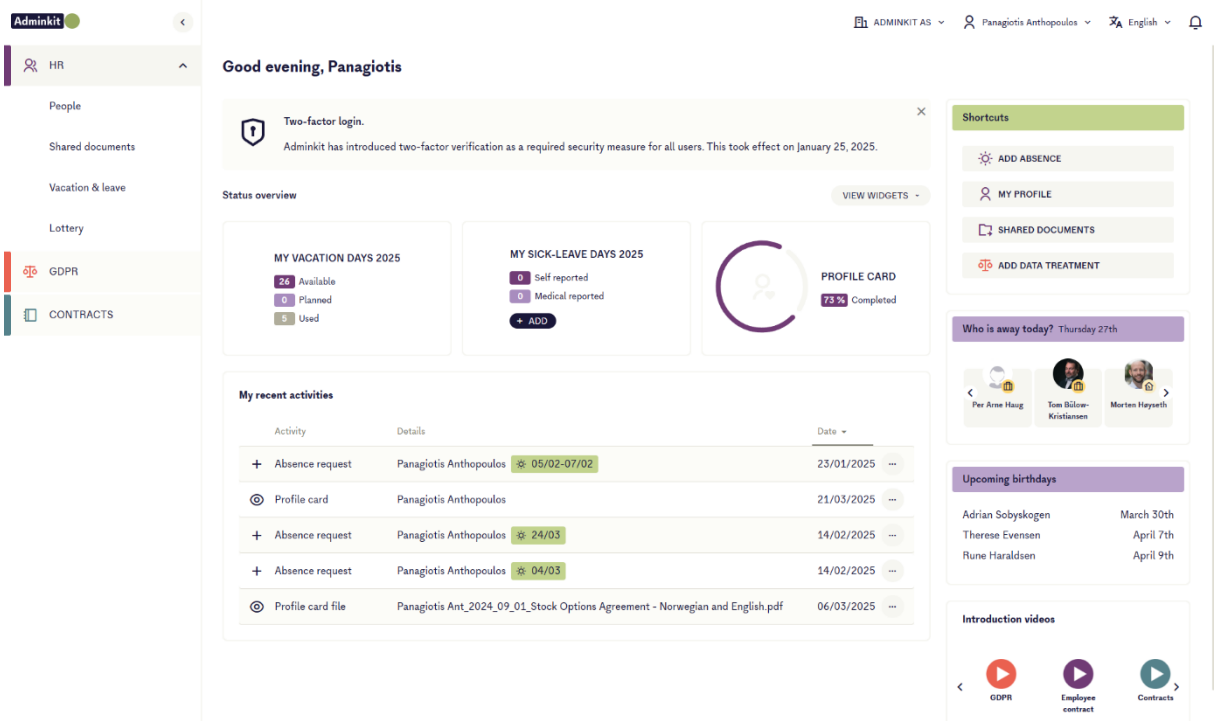
Όπως αναφέραμε και νωρίτερα, το πρώτο σενάριο αφορά την φόρτωση της αρχικής σελίδας (dashboard), της λίστας των υπαλλήλων και τέλος την περιήγηση του χρήστη στο προφίλ του.

Βελτιώσεις

Για την βελτίωση της απόδοσης στο συγκεκριμένο σενάριο ακολουθήσαμε στρατηγικές που είχαν ως σκοπό την μείωση του φόρτου της βάσης δεδομένων, την ελαχιστοποίηση των υπολογισμών και των επαναλήψεών τους, καθώς και την αποφυγή φόρτωσης μη απαραίτητων δεδομένων. Παρακάτω θα αναλύσουμε τις βελτιώσεις που έγιναν σε κάθε βήμα του σεναρίου.

Βήμα 1ο (Αρχική σελίδα-Dashboard):

Η αρχική σελίδα αποτελείται από διάφορες «περιοχές» που παρέχουν στον χρήστη πληροφορίες που αφορούν την επιχείρηση και τον ίδιο, όπως φαίνεται και στην παρακάτω εικόνα.



Εικόνα 4. 12: Η αρχική σελίδα της εφαρμογής

Οι περιοχές «Who is away today?» και «Upcoming Birthdays» μας δείχνουν τους υπαλλήλους που έχουν άδεια σήμερα και τους υπαλλήλους που έχουν γενέθλια σύντομα. Αυτές οι πληροφορίες παραμένουν ίδιες καθ' όλη τη διάρκεια της ημέρας, οπότε δεν υπάρχει λόγος να υπολογίζονται κάθε

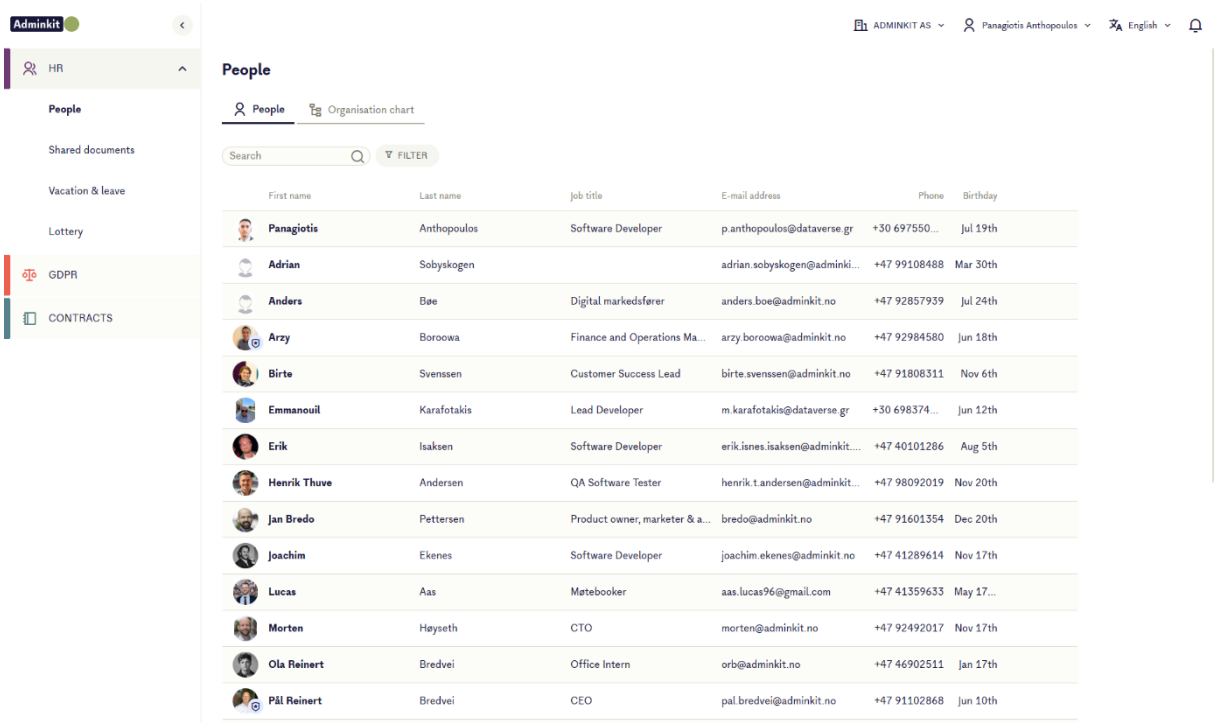
φορά που φορτώνει η αρχική σελίδα. Γι' αυτό πλέον τις αποθηκεύουμε στη προσωρινή μνήμη και έτσι υπολογίζονται μόνο μία φορά την ημέρα.

Στη περίπτωση της περιοχής «Who is away today?», πέρα από την χρήση προσωρινής μνήμης, υπήρξε και βελτίωση όσον αφορά τον υπολογισμό των υπαλλήλων με άδεια. Ο τρόπος που υπολογίζαμε αυτούς τους χρήστες ήταν να φέρουμε όλες τις άδειες και όλους τους υπαλλήλους της εταιρείας από τη βάση και στη συνέχεια να τις φιλτράρουμε με βάση την ημερομηνία, πριν τελικά επιστρέψουμε στο frontend τους χρήστες που έπρεπε. Πλέον όλη αυτή η διαδικασία γίνεται με ένα μεγάλο ερώτημα προς τη βάση που μας επιστρέφει μόνο τα στοιχεία των υπαλλήλων και της άδειάς τους που μας ενδιαφέρουν, τα οποία τα επιστρέφουμε στο frontend χωρίς περαιτέρω υπολογισμούς. Έτσι μειώσαμε δραστικά των αριθμό των συνδέσεων στη βάση δεδομένων, αλλά και την συνολική διάρκεια των υπολογισμών.

Για την περιοχή «My recent activities», η οποία δείχνει τις τελευταίες ενέργειες του χρήστη στην εφαρμογή, ακολουθήσαμε μια διαφορετική τακτική. Θεωρήσαμε ότι οι πληροφορίες που αποθηκεύονται σε αυτόν τον πίνακα δεν απαιτούν σχεσιακή βάση δεδομένων και με στόχο την ελάφρυνση του φόρτου της βάσης δεδομένων, μεταφέραμε αυτές τις πληροφορίες και την διαχείρισή τους σε μια ξεχωριστή MongoDB στην οποία αποθηκεύονται τα δεδομένα σε μορφή αρχείων JSON.

Βήμα 2ο (Σελίδα υπαλλήλων - People):

Η σελίδα των υπαλλήλων περιλαμβάνει έναν πίνακα που δείχνει κάποια βασικά στοιχεία των υπαλλήλων της εταιρείας, όπως φαίνεται και στην παρακάτω εικόνα.



Εικόνα 4.13: Η σελίδα των υπαλλήλων

Σε αυτήν τη σελίδα ακολουθήσαμε τρείς τακτικές βελτίωσης απόδοσης που αφορούν πρώτον τον γρηγορότερο υπολογισμό των δεδομένων, δεύτερον την μείωση του όγκου των δεδομένων που φορτώνονται και τρίτον την ελαχιστοποίηση των υπολογισμών.

Για την βελτίωση της ταχύτητας των υπολογισμών, κρίθηκε απαραίτητη μεγάλη αναδιαμόρφωση κώδικα με στόχο την μείωση των επαναλήψεων και την μείωση των συνδέσεων στη βάση δεδομένων.

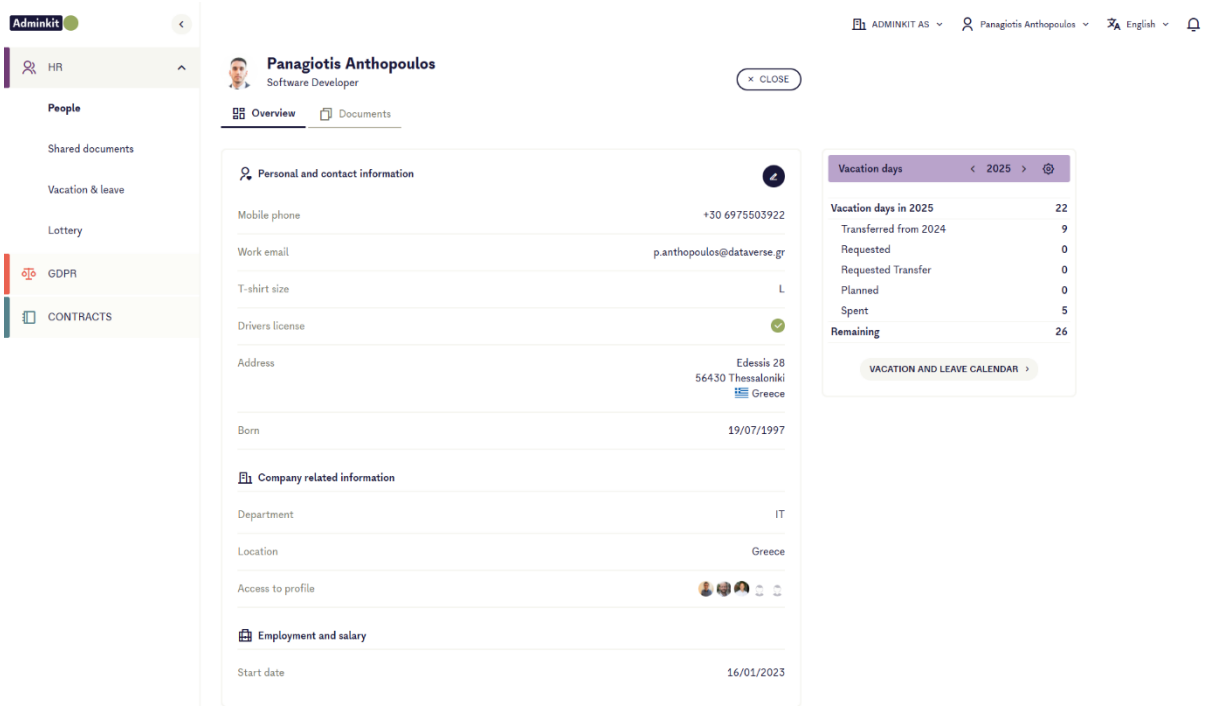
Πλέον, αντί για πολλά ερωτήματα στη βάση και συνδυασμό των απαντήσεων με διάφορους υπολογισμούς, παίρνουμε όλες τις απαραίτητες πληροφορίες κατευθείαν από τη βάση με μόλις ένα ερώτημα προς αυτήν. Εκτός από αυτό, αντί να φέρνουμε όλες τις πληροφορίες για κάθε υπάλληλο, φέρνουμε μόνο αυτές που φαίνονται στον πίνακα, όπως όνομα, τηλέφωνο, διεύθυνση ηλεκτρονικού ταχυδρομείου κλπ. Οι υπόλοιπες πληροφορίες γίνονται διαθέσιμες αν ο χρήστης περιηγηθεί στο προφίλ του εκάστοτε υπαλλήλου.

Για την μείωση του όγκου των δεδομένων που φορτώνονται ακολουθήσαμε την τακτική της σελιδοποίησης του πίνακα. Πριν τις βελτιώσεις, ο πίνακας περιείχε όλους τους υπαλλήλους της εταιρείας. Πλέον φορτώνονται μόνο δέκα υπάλληλοι κάθε φορά μειώνοντας τον φόρτο εργασίας για την εφαρμογή και καθιστώντας την απόδοσή της ανεξάρτητη από το μέγεθος της εταιρείας και συγκεκριμένα από τον αριθμό των υπαλλήλων.

Η τελευταία τακτική που ακολουθήσαμε στη συγκεκριμένη σελίδα ήταν η αποθήκευση της λίστας των υπαλλήλων στη προσωρινή μνήμη. Τα στοιχεία των υπαλλήλων δεν μεταβάλλονται συχνά, οπότε ο υπολογισμός τους κάθε φορά που φόρτωνε η σελίδα ήταν αχρείαστος. Φυσικά, σημαντική προσοχή δόθηκε στην ακύρωση της προσωρινής μνήμης σε περίπτωση που υπάρξει κάποια μεταβολή στοιχείων.

Βήμα 3ο (Προφίλ υπαλλήλου – Profile card):

Στη σελίδα του προφίλ ενός υπαλλήλου-χρήστη υπάρχουν όλες οι πληροφορίες του υπαλλήλου καθώς και αρχεία τα οποία δύναται να μεταφορτώσει ο κάθε χρήστης. Επίσης, διατίθενται και πληροφορίες για τις ημέρες άδειας του χρήστη, όπως και βλέπουμε στην παρακάτω εικόνα.



Εικόνα 4.14: Η σελίδα του προφίλ ενός υπαλλήλου

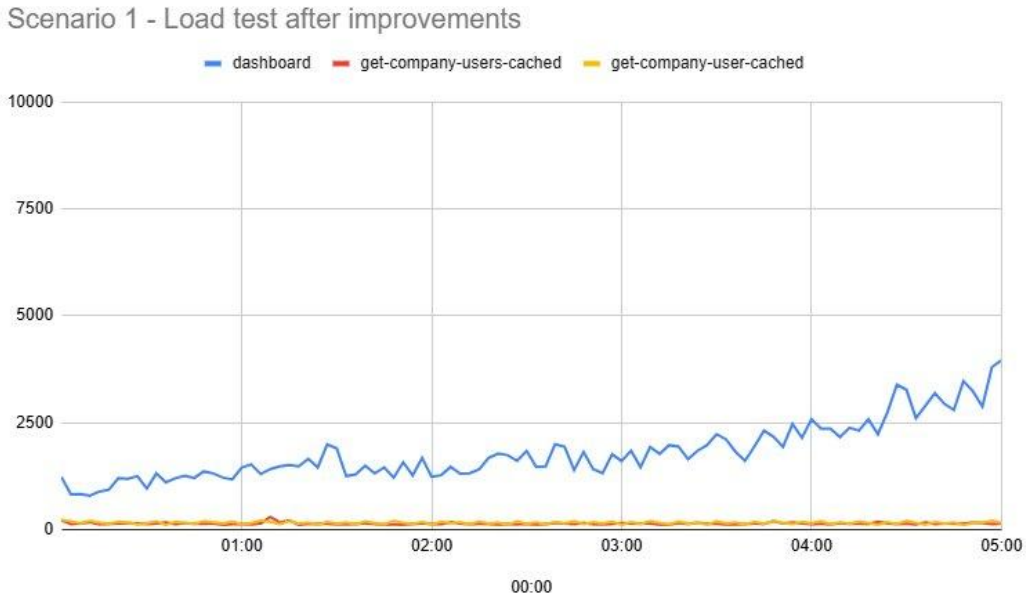
Όσον αφορά την σελίδα του προφίλ ενός υπαλλήλου-χρήστη η πιο σημαντική βελτίωση ήταν αποτέλεσμα μιας μεταβολής στην διεπαφή χρήστη της εφαρμογής (UI), η οποία ήταν η μεταφορά των αρχείων σε ξεχωριστή υποσελίδα. Πλέον σε αυτή τη σελίδα φορτώνουν μόνο τα στοιχεία του υπαλλήλου, ενώ τα αρχεία φορτώνουν μόνο αν περιηγηθεί ο χρήστης στην υποσελίδα με αυτά.

Αποτέλεσμα ήταν η μείωση του χρόνου φόρτωσης της σελίδας, ειδικά σε περιπτώσεις χρηστών που είχαν μεταφορτώσει πολλά αρχεία.

Επιπλέον χρησιμοποιήθηκε και σε αυτή την περίπτωση η προσωρινή μνήμη για την αποθήκευση των στοιχείων του υπαλλήλου, καθώς και των πληροφοριών που αφορούν της ημέρες άδειάς του.

Αποτελέσματα

Για την ανάλυση των αποτελεσμάτων μετά τις βελτιώσεις θα χρησιμοποιήσουμε την παρακάτω εικόνα.



Εικόνα 4.15: Αποτελέσματα του πρώτου σεναρίου μετά τις βελτιώσεις

Ο οριζόντιος άξονας **(X-Axis)** αντιπροσωπεύει τον χρόνο, σε λεπτά, από την έναρξη της δοκιμής. Ο κατακόρυφος άξονας **(Y-Axis)** δείχνει τον διάμεσο χρόνο απόκρισης για κάθε αίτημα, μετρημένο σε χιλιοστά του δευτερολέπτου **(ms)**.

Για τη φόρτωση της αρχικής σελίδας(dashboard) παρατηρούμε ότι όσο αυξάνονται οι εικονικοί χρήστες, τόσο χειρότερη γίνεται η απόδοση του συστήματος. Πιο συγκεκριμένα, μετά την πάροδο τριών λεπτών βλέπουμε τους χρόνους απόκρισης να έχουν διπλασιαστεί. Ωστόσο, παρά την αύξηση των χρόνων απόκρισης η εφαρμογή παραμένει πλήρως ανταποκρίσιμη, αν και εμφανώς πιο αργή, ακόμα και στο τέλος της δοκιμής. Αντίθετα, όσον αφορά τους χρόνους απόκρισης στα επόμενα βήματα της δοκιμής, δηλαδή την φόρτωση της σελίδας με την λίστα των υπαλλήλων και του προφίλ του χρήστη, βλέπουμε την χρήση προσωρινής μνήμης να είναι απολύτως αποδοτική με τους χρόνους απόκρισης να παραμένουν σχεδόν αμετάβλητοι, καθώς πλέον δεν απαιτείται ανάκληση δεδομένων από την βάση, ούτε περαιτέρω υπολογισμοί.

Σενάριο 2

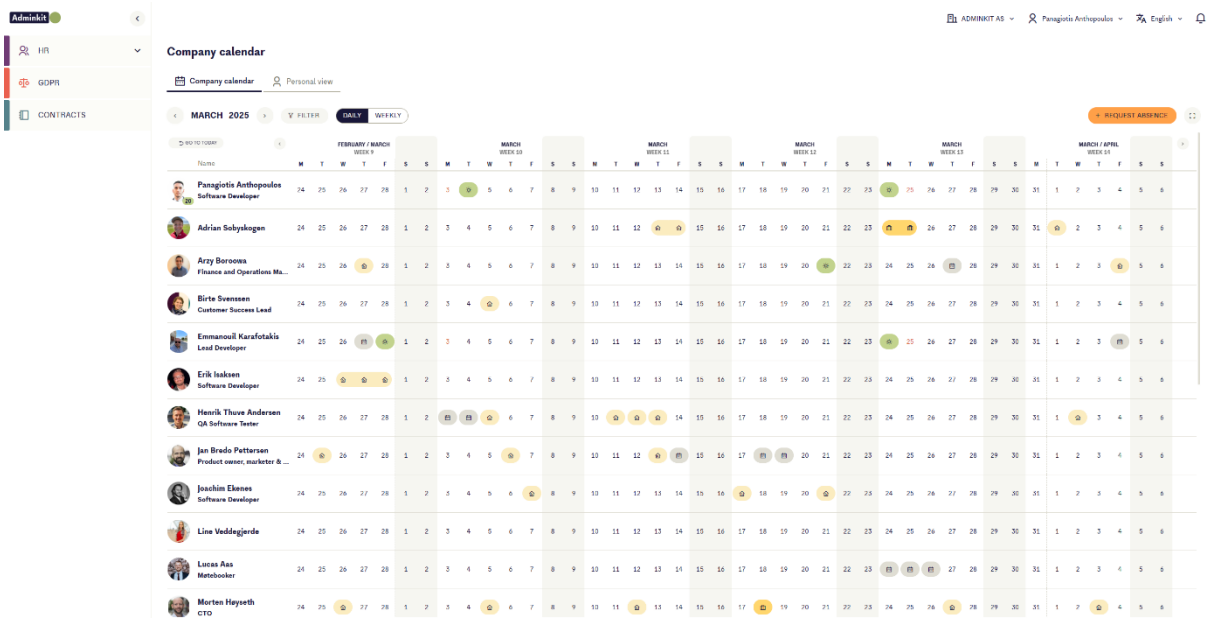
Όπως αναφέραμε και νωρίτερα, το δεύτερο σενάριο εστιάζει στην απόδοση της σελίδας του ημερολογίου αδειών, του προσωπικού ημερολογίου και της προσθήκης μιας καινούριας άδειας.

Βελτιώσεις

Για την βελτίωση της απόδοσης στο συγκεκριμένο σενάριο, όπως και στο προηγούμενο ακολουθήσαμε τεχνικές που είχαν ως σκοπό την μείωση του φόρτου της βάσης δεδομένων, την ελαχιστοποίηση των υπολογισμών και των επαναλήψεών τους, καθώς και την αποφυγή φόρτωσης μη απαραίτητων δεδομένων. Παρακάτω θα αναλύσουμε τις βελτιώσεις που έγιναν σε κάθε βήμα του σεναρίου.

Βήμα 1ο (Ημερολόγιο αδειών – Company calendar):

Η σελίδα του ημερολογίου, περιλαμβάνει το ημερολόγιο με τις άδειες όλων των υπαλλήλων της εταιρείας, όπως βλέπουμε και στην παρακάτω εικόνα.

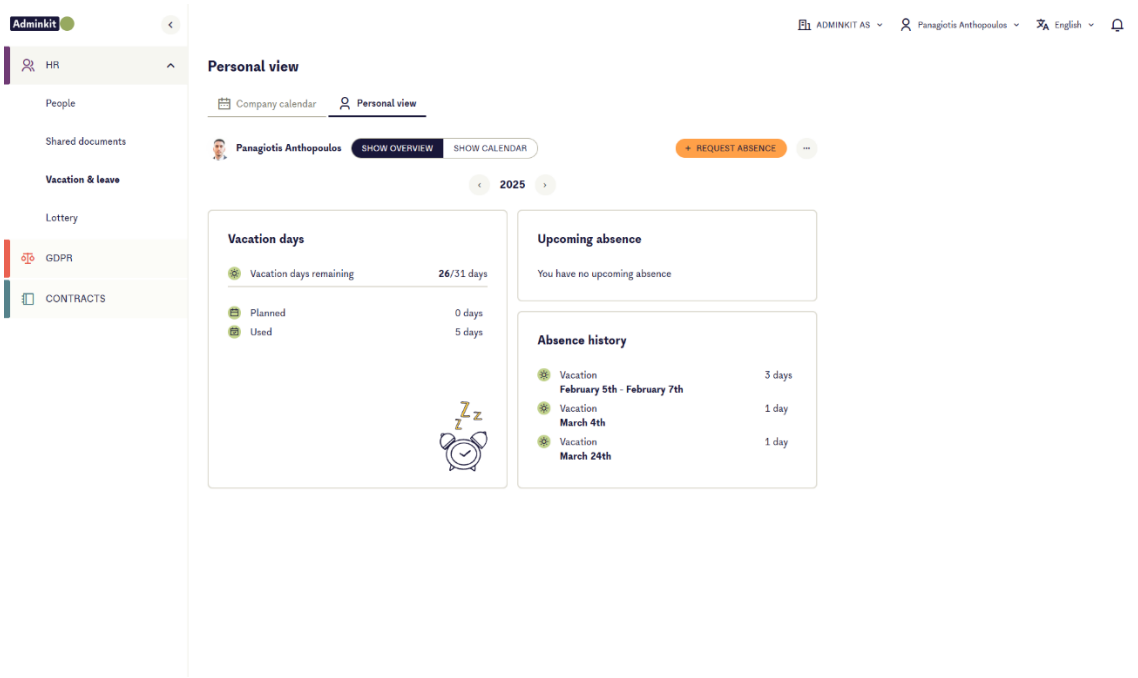


Εικόνα 4.16: Το ημερολόγιο αδειών της εφαρμογής

Η μεγαλύτερη βελτίωση σε αυτήν τη σελίδα προήλθε από την μείωση των δεδομένων που φορτώνονται. Πριν, φορτώναμε όλες τις άδειες τις εταιρείας, από την ημέρα που δημιουργήθηκε ο λογαριασμός μέχρι και σήμερα, ανεξάρτητα από το αν ήταν ορατές στο ημερολόγιο. Αυτό είχε ως επίπτωση την επιβάρυνση της εφαρμογής τόσο στο επίπεδο της βάσης δεδομένων όσο και σε επίπεδο κώδικα λόγω του μεγάλου αριθμού αχρείαστων υπολογισμών και επαναλήψεων. Το πρόβλημα αυτό γινόταν ακόμα πιο σημαντικό όσο περισσότερο καιρό χρησιμοποιούσε μια εταιρεία την εφαρμογή, καθώς ο αριθμός των αδειών αυξανόταν συνέχεια και ακολούθως και ο όγκος των δεδομένων που έπρεπε να υπολογιστούν και να φορτωθούν. Πλέον φορτώνουμε μόνο τις άδειες ένα μήνα πριν και ένα μήνα μετά το σήμερα, αυτές δηλαδή που είναι ορατές στη διεπαφή χρήστη. Όταν ο χρήστης περιηγηθεί σε κάποια άλλη χρονική περίοδο, φορτώνονται και οι σχετικές πληροφορίες αδειών. Με αυτή την αλλαγή η απόδοση της εφαρμογής δεν εξαρτάται πλέον από τον αριθμό των αδειών που έχει μια εταιρεία.

Βήμα 2ο (Προσωπικό ημερολόγιο – Personal calendar):

Σε αυτή την σελίδα φαίνονται οι πληροφορίες των αδειών ενός υπαλλήλου, όπως βλέπουμε και στην παρακάτω εικόνα.



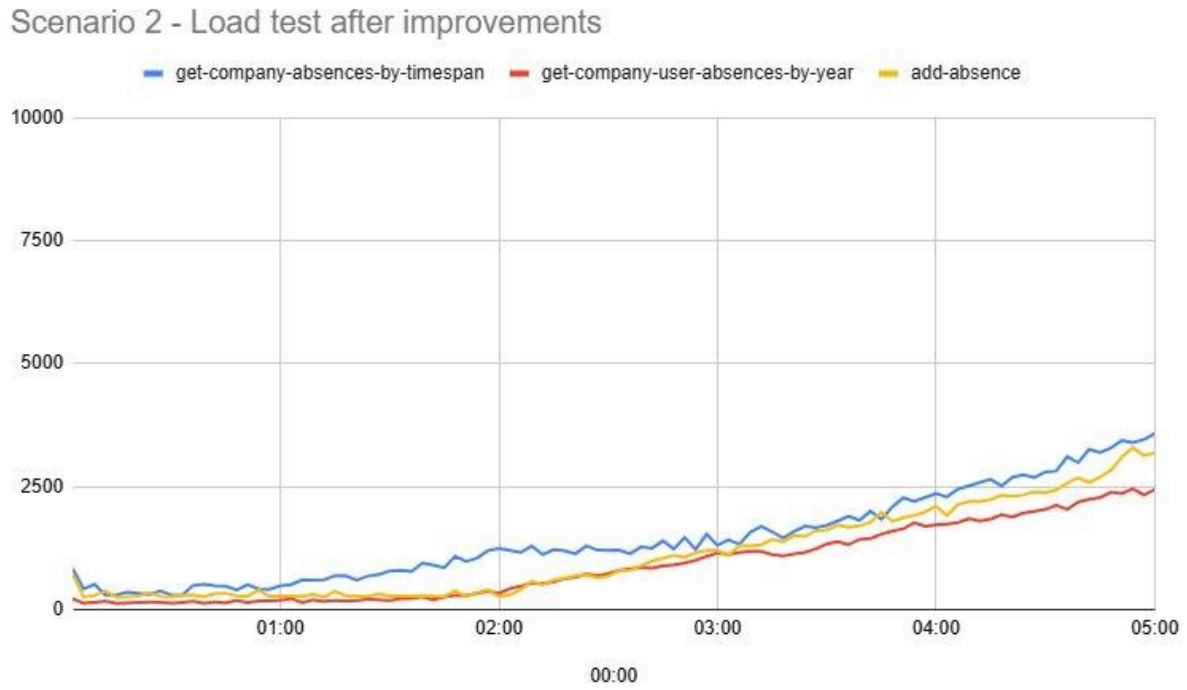
Εικόνα 4.17: Το προσωπικό ημερολόγιο ενός χρήστη

Παρομοίως με τις βελτιώσεις που υλοποιήθηκαν στο προηγούμενο βήμα, έτσι και στη σελίδα του προσωπικού ημερολογίου, αντί να φορτώνουμε όλες τις άδειες που έχει ο χρήστης από την στιγμή που άρχισε να χρησιμοποιεί την εφαρμογή, πλέον φορτώνουμε μόνο τις άδειες ενός συγκεκριμένου έτους.

Επιπλέον, οι πληροφορίες που αφορούν το υπόλοιπο ημερών άδειας του χρήστη αποθηκεύονται στη προσωρινή μνήμη για αποφυγή επανάληψης επιβαρυντικών υπολογισμών.

Αποτελέσματα

Για την ανάλυση των αποτελεσμάτων θα χρησιμοποιήσουμε την παρακάτω εικόνα.



Εικόνα 4.18: Αποτελέσματα του δεύτερου σεναρίου μετά τις βελτιώσεις

Ο οριζόντιος άξονας **(X-Axis)** αντιπροσωπεύει τον χρόνο, σε λεπτά, από την έναρξη της δοκιμής. Ο κατακόρυφος άξονας **(Y-Axis)** δείχνει τον διάμεσο χρόνο απόκρισης για κάθε αίτημα, μετρημένο σε χιλιοστά του δευτερολέπτου **(ms)**.

Όπως ήταν αναμενόμενο, παράλληλα με την αύξηση των εικονικών χρηστών, παρατηρούμε και αύξηση των χρόνων απόκρισης του συστήματος. Συγκεκριμένα, καθ'όλη τη διάρκεια της δοκιμής βλέπουμε τους χρόνους απόκρισης και στα τρία βήματα να αυξάνονται με σταθερό ρυθμό. Βλέπουμε όμως ότι παρά τους μεγαλύτερους χρόνους απόκρισης δεν εμφανίζονται αιχμές που να δείχνουν αστάθεια του συστήματος.

4.3 Σύγκριση αποτελεσμάτων και ανάλυση

Στα προηγούμενα κεφάλαια αναλύσαμε τους χρόνους απόκρισης για διάφορα αιτήματα πριν και μετά τις βελτιώσεις. Παρατηρήθηκε ότι κάποια αιτήματα (π.χ., get-company-users-on-absence) είχαν πολύ υψηλούς χρόνους απόκρισης, κάτι που υποδεικνύει σημεία συμφόρησης και ανάγκη για βελτιστοποίηση. Αφού ακολούθησαν αλλαγές (π.χ., βελτιστοποίηση του backend, χρήση caching, και φιλτράρισμα δεδομένων), οι χρόνοι απόκρισης για πολλά αιτήματα παρουσίασαν σημαντική μείωση.

Όπως φαίνεται και στα προηγούμενα διαγράμματα, οι αλλαγές στη δομή της εφαρμογής και στον τρόπο διαχείρισης των δεδομένων οδήγησαν σε σημαντική μείωση των χρόνων απόκρισης σε αρκετά αιτήματα. Η χρήση caching για δεδομένα που δεν αλλάζουν συχνά συνέβαλε στην εξοικονόμηση χρόνου, ενώ η βελτιστοποίηση των ερωτημάτων με τη χρήση ευρετηρίων στη βάση δεδομένων είχε θετική επίδραση στην απόδοση.

Κλείνοντας, οι βελτιώσεις είχαν θετική επίδραση στην απόδοση του συστήματος, με τη μείωση των χρόνων απόκρισης για αρκετά αιτήματα και τη γενικότερη βελτίωση της χρήστης εμπειρίας.

Κεφάλαιο 5ο: Συμπεράσματα

Σε αυτό το κεφάλαιο θα συνοψίσουμε τα βασικά ευρήματα της μελέτης που κάναμε, θα αναλύσουμε τους περιορισμούς που επηρέασαν τα αποτελέσματα και θα προτείνουμε πιθανές κατευθύνσεις για μελλοντική έρευνα και βελτιώσεις στο σύστημα.

5.1 Συμπεράσματα και κύρια ευρήματα

Η παρούσα μελέτη επικεντρώθηκε στην αξιολόγηση της απόδοσης ενός πληροφοριακού συστήματος, συγκεκριμένα της AdminKit, τόσο πριν όσο και μετά την εφαρμογή στοχευμένων στρατηγικών βελτιστοποίησης. Η ανάλυση των αποτελεσμάτων ανέδειξε τη σημασία της αποτελεσματικής διαχείρισης των πόρων του συστήματος καθώς και της βελτιστοποίησης του κώδικα, με στόχο τη βελτίωση των χρόνων απόκρισης και τη συνολική εμπειρία χρήστη.

Ένα από τα πλέον σημαντικά ευρήματα της μελέτης ήταν η σημαντική μείωση των χρόνων απόκρισης μετά την εφαρμογή των βελτιώσεων. Ιδιαίτερα σε αιτήματα που αφορούσαν τη διαχείριση μεγάλων όγκων δεδομένων η επίδραση των βελτιστοποιήσεων ήταν αισθητή, καθώς οι χρόνοι φόρτωσης μειώθηκαν δραστικά. Οι τεχνικές που εφαρμόστηκαν περιλάμβαναν τη χρήση caching και τη βελτιστοποίηση των ερωτημάτων βάσης δεδομένων, συμβάλλοντας στη μείωση της επιβάρυνσης του συστήματος και στην ταχύτερη επεξεργασία των αιτημάτων.

Η χρήση τεχνικών προσωρινής αποθήκευσης **(caching)**, όπως το in-memory caching και το distributed caching με την αξιοποίηση εργαλείων όπως το Redis, αποδείχθηκε ιδιαίτερα αποτελεσματική. Η υιοθέτηση αυτών των τεχνικών είχε ως αποτέλεσμα τη σημαντική μείωση του αριθμού των αιτημάτων προς τη βάση δεδομένων, οδηγώντας σε βελτιωμένη απόδοση της εφαρμογής και ταχύτερη απόκριση στις απαιτήσεις των χρηστών. Η προσωρινή αποθήκευση δεδομένων συνέβαλε επίσης στη μείωση της δικτυακής κίνησης και της υπολογιστικής ισχύος που απαιτείται για την εκτέλεση επαναλαμβανόμενων αιτημάτων.

Ωστόσο, η μελέτη ανέδειξε ότι παρά τις γενικές βελτιώσεις, ορισμένες σελίδες, όπως η φόρτωση της αρχικής σελίδας, συνέχισαν να εμφανίζουν υψηλούς χρόνους απόκρισης. Το γεγονός αυτό υποδεικνύει ότι απαιτείται περαιτέρω ανάλυση και στοχευμένη βελτιστοποίηση των συγκεκριμένων σελίδων. Επομένως, η περαιτέρω διερεύνηση της απόδοσης των συγκεκριμένων κλήσεων είναι απαραίτητη, ώστε να εντοπιστούν οι ακριβείς αιτίες της καθυστέρησης και να εφαρμοστούν πιο εξειδικευμένες τεχνικές βελτιστοποίησης.

Κλείνοντας, η εισαγωγή φιλτραρίσματος δεδομένων αποδείχθηκε ιδιαίτερα ωφέλιμη στη μείωση των χρόνων απόκρισης, καθώς επέτρεψε τη στοχευμένη επιστροφή δεδομένων βάσει χρονικού διαστήματος ή άλλων κριτηρίων. Παρόλα αυτά, η μελέτη αποκάλυψε ότι η σειρά εφαρμογής του φιλτραρίσματος παίζει καθοριστικό ρόλο στην απόδοση του συστήματος. Συγκεκριμένα, όταν το φιλτράρισμα εφαρμόζεται μετά την ανάκτηση όλων των δεδομένων από τη βάση, η συνολική απόδοση του συστήματος μπορεί να επηρεαστεί αρνητικά, καθώς αυξάνεται η υπολογιστική επιβάρυνση. Αντίθετα, η ενσωμάτωση του φιλτραρίσματος απευθείας στο επίπεδο του ερωτήματος προς τη βάση δεδομένων, μπορεί να οδηγήσει σε σημαντικά ταχύτερη επεξεργασία των αιτημάτων και στη μείωση των συνδέσεων στη βάση δεδομένων.

5.2 Περιορισμοί της μελέτης

Παρόλο που η παρούσα μελέτη απέδωσε σημαντικά ευρήματα σχετικά με την απόδοση του συστήματος, πρέπει να αναγνωριστούν ορισμένοι περιορισμοί που ενδέχεται να επηρέασαν τη γενικότητα και την εγκυρότητα των αποτελεσμάτων. Αυτοί οι περιορισμοί θα πρέπει να ληφθούν υπόψη κατά την ερμηνεία των ευρημάτων και να καθοδηγήσουν μελλοντικές έρευνες στον τομέα της βελτιστοποίησης των πληροφοριακών συστημάτων.

Οι δοκιμές που πραγματοποιήθηκαν στην παρούσα μελέτη διεξήχθησαν σε ένα ελεγχόμενο περιβάλλον, το οποίο δεν αποτυπώνει απόλυτα τις πραγματικές συνθήκες λειτουργίας ενός συστήματος σε περιβάλλον παραγωγής. Σε πραγματικές συνθήκες χρήσης, οι χρήστες ενδέχεται να αλληλοεπιδρούσαν με το σύστημα με διαφορετικούς τρόπους, όπως για παράδειγμα με πιο ασταθείς συνθήκες δικτύου.

Επίσης ένα σημαντικό μειονέκτημα της μελέτης ήταν η χρήση στατικών δεδομένων κατά τη διάρκεια των δοκιμών. Τα δεδομένα αυτά ήταν προκαθορισμένα και δεν αναπαριστούσαν τις δυναμικές συνθήκες που επικρατούν σε ένα πραγματικό σύστημα, όπου τα δεδομένα εξελίσσονται συνεχώς και η κατανομή των αιτημάτων ενδέχεται να μεταβάλλεται κατά τη διάρκεια της λειτουργίας. Στην πραγματικότητα, οι χρήστες της εφαρμογής ενδέχεται να επηρεάζουν τον όγκο των δεδομένων που αποθηκεύονται στη βάση δεδομένων, προκαλώντας μεταβολές στους χρόνους απόκρισης και στην επεξεργασία των αιτημάτων. Η χρήση δυναμικών δεδομένων και η εφαρμογή δοκιμών σε συνθήκες πιο κοντινές στην πραγματική λειτουργία του συστήματος θα μπορούσε να προσφέρει μια πιο ακριβή εκτίμηση της πραγματικής απόδοσης.

Ακόμη, η απόδοση των συστημάτων μπορεί να επηρεαστεί από εξωτερικούς παράγοντες που δεν ελήφθησαν υπόψη κατά τη διάρκεια της μελέτης, όπως η ποιότητα της δικτυακής σύνδεσης και η φόρτιση του διακομιστή κατά τη διάρκεια των δοκιμών. Οι συνθήκες δικτύου, ιδίως σε περιβάλλοντα με χαμηλές ταχύτητες ή μεταβλητότητα στην ταχύτητα σύνδεσης, μπορεί να οδηγήσουν σε μεγαλύτερους χρόνους απόκρισης, ενώ οι φόρτοι στους διακομιστές ή η συμφόρηση του συστήματος μπορεί να μειώσουν τη διαθεσιμότητα και την αξιοπιστία του συστήματος.

Συνολικά, οι περιορισμοί αυτοί επισημαίνουν την ανάγκη για περαιτέρω και πιο εκτεταμένες δοκιμές, που θα εξετάζουν τις επιδόσεις του συστήματος υπό πιο ρεαλιστικές συνθήκες χρήσης και με δυναμικά δεδομένα. Επιπλέον, η ενσωμάτωση διαφορετικών μεθόδων και εργαλείων παρακολούθησης της απόδοσης θα μπορούσε να προσφέρει μια πιο σφαιρική εικόνα για τη συμπεριφορά του συστήματος και να βοηθήσει στη βελτίωση των στρατηγικών βελτιστοποίησης για το μέλλον.

5.3 Προτάσεις για μελλοντική έρευνα

Ένα σημαντικό βήμα για τη μελλοντική βελτίωση των αποτελεσμάτων της έρευνας είναι η διεξαγωγή δοκιμών σε πραγματικές συνθήκες χρήσης με αυξημένο αριθμό ταυτόχρονων χρηστών. Στη συγκεκριμένη μελέτη, οι δοκιμές πραγματοποιήθηκαν σε ελεγχόμενο περιβάλλον, το οποίο ενδέχεται να μην αντανακλά πλήρως τη συμπεριφορά του συστήματος σε πραγματικά φορτισμένα περιβάλλοντα παραγωγής. Η προσομοίωση ενός μεγάλου αριθμού ταυτόχρονων χρηστών και η αξιολόγηση της συμπεριφοράς του συστήματος υπό συνθήκες έντονου φόρτου θα επιτρέψει την καλύτερη κατανόηση του τρόπου λειτουργίας του συστήματος σε κλίμακα και θα οδηγήσει σε πιο ακριβείς προβλέψεις για την απόδοσή του σε παραγωγικά περιβάλλοντα.

Επίσης, η χρήση τεχνικών caching, όπως το in-memory caching και το Redis, είχε σημαντικό θετικό αντίκτυπο στους χρόνους απόκρισης του συστήματος. Παρ' όλα αυτά, η στρατηγική caching μπορεί να βελτιωθεί περαιτέρω. Η έρευνα μπορεί να εστιάσει σε πιο εξελιγμένους μηχανισμούς αποθήκευσης

δεδομένων, όπως η αποθήκευση δεδομένων βάσει συχνότητας χρήσης ή η χρήση LRU **(Least Recently Used)** caching για πιο ευέλικτη διαχείριση του cache. Επίσης, η εφαρμογή cache invalidation strategies (όπως time-to-live (TTL)) μπορεί να προσφέρει μία βελτιωμένη ισορροπία μεταξύ ακρίβειας δεδομένων και απόδοσης.

Μια άλλη κατεύθυνση για τη μελλοντική έρευνα αφορά την αξιοποίηση serverless αρχιτεκτονικών, όπως αυτές που παρέχονται από πλατφόρμες cloud (π.χ., AWS Lambda, Azure Functions). Το μοντέλο serverless υπόσχεται να μειώσει το κόστος λειτουργίας του συστήματος, προσφέροντας ευέλικτη κλιμάκωση και μειώνοντας τα κόστη που σχετίζονται με την υποδομή. Η εφαρμογή αυτών των αρχιτεκτονικών μπορεί να βελτιώσει τη διαχείριση πόρων, ιδίως σε περιόδους χαμηλού φορτίου, ενώ επιτρέπει την αμεσότερη προσαρμογή του συστήματος στις αυξομειώσεις του φόρτου.

Ακόμη, από τα αποτελέσματα της μελέτης, φάνηκε ότι η εφαρμογή φιλτραρίσματος στον όγκο δεδομένων που επιστρέφονται από τη βάση δεδομένων μπορεί να μειώσει σημαντικά τους χρόνους απόκρισης. Ωστόσο, η χρήση πιο προχωρημένων τεχνικών φιλτραρίσματος μπορεί να οδηγήσει σε ακόμη καλύτερα αποτελέσματα. Για παράδειγμα, η εισαγωγή μηχανισμών φιλτραρίσματος απευθείας στο επίπεδο βάσης δεδομένων, όπως indexed filtering, partitioning και sharding μπορεί να μειώσει τον όγκο δεδομένων που επεξεργάζεται το σύστημα, βελτιώνοντας περαιτέρω την ταχύτητα απόκρισης.

Κλείνοντας, η συνεχής παρακολούθηση και ανάλυση της απόδοσης του συστήματος είναι κρίσιμη για την μακροπρόθεσμη σταθερότητα του συστήματος. Η μελλοντική έρευνα θα πρέπει να περιλαμβάνει μηχανισμούς αυτοματοποιημένων βελτιστοποιήσεων και συστήματα προειδοποίησης σε πραγματικό χρόνο, έτσι ώστε να εντοπίζονται γρήγορα τυχόν προβλήματα απόδοσης κάτω του αναμενόμενου επιπέδου.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] FasterCapital, “Performance Metrics: Load Time Statistics: Speed Matters: The Significance of Load Time Statistics,” FasterCapital.

[2] M. Yu, R. Zhou, Z. Cai, C.-W. Tan, and H. Wang, “Unravelling the relationship between response time and user experience in mobile applications,” *Internet Research*, vol. 30, no. 5, pp. 1353–1382, May 2020, doi: 10.1108/INTR-05-2019-0223.

[3] Hoxmeier JA and DiCesare C., “System Response Time and User Satisfaction: An Experimental Study of Browser-based Applications.,” *AMCIS 2000 Proceedings*, 2000.

[4] S. Machiraju and S. Gaurav, “Key Application Experiences: Latency, Scalability, and Throughput,” in *Hardening Azure Applications*, Berkeley, CA: Apress, 2019, pp. 123–139. doi: 10.1007/978-1-4842-4188-2_5.

[5] S. Bulla, C. V. R. Reddy, P. Padmavathi, and T. Padmasri, “Analytical Evaluation of Resource Estimation in Web Application Services,” *Ingénierie des systèmes d information*, vol. 25, no. 5, pp. 683–690, Nov. 2020, doi: 10.18280/isi.250516.

[6] A. Kumari, M. K. Patra, B. Sahoo, and R. K. Behera, “Resource optimization in performance modeling for serverless application,” *International Journal of Information Technology*, vol. 14, no. 6, pp. 277–288, Oct. 2022, doi: 10.1007/s41870-022-01073-x.

[7] J. D. C. Little, “A Proof for the Queuing Formula: $L = \lambda W$,” *Oper Res*, vol. 9, no. 3, pp. 383–387, Jun. 1961, doi: 10.1287/opre.9.3.383.

[8] L. Kleinrock, Queueing Systems, Volume 1: Theory. Wiley-Interscience. 1975.

[9] R. J. Wieringa, “Design Methods for Reactive Systems: Beyond Architecture.,” *Elsevier*, 2012.

[10] M. Herlihy, N. Shavit, V. Luchangco, and M. Spear, *The Art of Multiprocessor Programming*. Elsevier, 2021. doi: 10.1016/C2011-0-06993-4.

[11] E. Babb, “Implementing a relational database by means of specialzed hardware,” *ACM Transactions on Database Systems*, vol. 4, no. 1, pp. 1–29, Mar. 1979, doi: 10.1145/320064.320065.

[12] L. S. Iyer, B. Gupta, and N. Johri, “Performance, scalability and reliability issues in web applications,” *Industrial Management & Data Systems*, vol. 105, no. 5, pp. 561–576, Jun. 2005, doi: 10.1108/02635570510599959.

[13] S. Pargaonkar, “A Comprehensive Review of Performance Testing Methodologies and Best Practices: Software Quality Engineering,” *International Journal of Science and Research (IJSR)*, vol. 12, no. 8, pp. 2008–2014, Aug. 2023, doi: 10.21275/SR23822111402.

[14] C. Mărcuță, “The Importance of Scalability in Native App Development: Key to Long-Term Success.,” *MoldStud*, 2024.

[15] M. Youssef and L. Hassan, “Optimizing Database Operations for Maximum Performance: Advanced Strategies for Enhancing Efficiency, Scalability, and Reliability in High-Throughput Enterprise Systems,” 2024.

[16] F. Thillen, R. Mordinyi, and S. Biffl, “Isolated Testing of Software Components in Distributed Software Systems,” 2014, pp. 170–184. doi: 10.1007/978-3-319-03602-1_11.

[17] G. Grano, C. Laaber, A. Panichella, and S. Panichella, “Testing with Fewer Resources: An Adaptive Approach to Performance-Aware Test Case Generation,” Jul. 2019, doi: 10.1109/TSE.2019.2946773.

[18] M. Bolanowski, M. Ćmil, and A. Starzec, “New Model for Defining and Implementing Performance Tests,” *Future Internet*, vol. 16, no. 10, p. 366, Oct. 2024, doi: 10.3390/fi16100366.

[19] S. A. Khan, N. T. Oshin, M. Nizam, I. Ahmed, M. M. Musfique, and M. Hasan, “AI-Based Software Testing,” in *Lecture Notes in Networks and Systems*, Springer Science and Business Media Deutschland GmbH, 2024, pp. 323–334. doi: 10.1007/978-981-99-8346-9_28.

[20] J. R. Lewis and J. Sauro, “Usability and User Experience: Design and Evaluation,” in *Handbook of Human Factors and Ergonomics*, wiley, 2021, pp. 972–1015. doi: 10.1002/9781119636113.ch38.