ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΕΡΓΑΛΕΙΩΝ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΩΝ ΣΤΟΝ ΙΣΤΟ. ANGULAR: EXPRESSJS ΚΑΙ FLASK: POSTGRESQL ΚΑΙ MONGODB.

Διπλωματική Εργασία

του

Εμμανουήλ Κακαράκη

Θεσσαλονίκη, Ιούνιος 2023

ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΕΡΓΑΛΕΙΩΝ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΩΝ ΣΤΟΝ ΙΣΤΟ. ANGULAR: EXPRESSJS ΚΑΙ FLASK: POSTGRESQL ΚΑΙ MONGODB.

Εμμανουήλ Κακαράκης

Πτυχίο Αγρονόμων Τοπογράφων Μηχανικών, Πολυτεχνική Σχολή ΑΠΘ, 2020

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Κωνσταντίνος Μαργαρίτης

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ηη/μμ/εεεε
Ονοματεπώνυμο 1 Ονοματεπώνυμο 2 Ονοματεπώνυμο 3

....

Εμμανουήλ Κακαράκης

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Περίληψη

Η διπλωματική εργασία ασχολείται με την ανάπτυξη μιας εφαρμογής χρησιμοποιώντας ένα συνδυασμό των τεχνολογιών Flask, Expressjs, Postgres, MongoDB και Angular. Ο στόχος είναι η σύγκριση των διαφορετικών συνδυασμών των τεχνολογιών αυτών με βάση διάφορα κριτήρια, όπως η απόδοση και η ταχύτητα.

Με τη χρήση του Flask και του Expressjs στο backend της εφαρμογής, μπορούν να επιτευχθούν αποδοτικές και με ευκολια επεκτασιμοτητας λειτουργιες στο server. Οι βάσεις δεδομένων Postgres και MongoDB παρέχουν αξιόπιστες λύσεις για την αποθήκευση και διαχείριση των δεδομένων, με την κάθε μία να προσφέρει διαφορετικά πλεονεκτήματα. Η Angular χρησιμοποιείται για την ανάπτυξη του frontend της εφαρμογής, παρέχοντας δυνατότητες για τη δημιουργία αποδοτικών και δυναμικών UI.

Με αυτήν τη διπλωματική εργασία, ο στόχος είναι να αξιολογηθούν οι διάφοροι συνδυασμοί αυτών των τεχνολογιών

Για να προσδιοριστεί ο αποτελεσματικότερος συνδυασμός αυτών των τεχνολογιών για την κατασκευή μιας ισχυρής και αξιόπιστης εφαρμογής REST CRUD, μπορεί να χρησιμοποιηθεί μια προσέγγιση συγκριτικής ανάλυσης. Αυτή περιλαμβάνει την αξιολόγηση και τη σύγκριση των χαρακτηριστικών κάθε τεχνολογίας με βάση συγκεκριμένα κριτήρια, όπως η απόδοση, η επεκτασιμότητα και η ευκολία χρήσης. Με τον τρόπο αυτό, είναι δυνατόν να εντοπιστούν τα δυνατά και αδύνατα σημεία κάθε τεχνολογίας και πώς μπορούν να συνδυαστούν για να αναπτυχθεί μία εφαρμογή.

Εκτός από τα οφέλη της χρήσης αυτών των τεχνολογιών, μπορεί επίσης να υπάρξουν προκλήσεις που προκύπτουν κατά τη διάρκεια της ανάπτυξης. Ωστόσο, έχοντας επίγνωση αυτών των προκλήσεων και λαμβάνοντας μέτρα για τον μετριασμό τους, όπως η χρήση κατάλληλων εργαλείων και τεχνικών, μπορούμε να διασφαλίσουμε μια ομαλή και αποτελεσματική διαδικασία ανάπτυξης.

Keywords: CRUD, Expressjs, Flask, Angular, Postgres, Mongodb, web framework, web application, web application framework, comparative analysis, frameworks comparison, REST

Η εφαρμογή

Η εφαρμογή κατασκευάστηκε από το μηδέν χρησιμοποιώντας το Flask και Expressjs ως back-end frameworks και Angular για το front-end. Η εφαρμογή χρησιμοποιεί τη δημόσια βάση δεδομένων του IMDb και διαθέτει λειτουργίες CRUD για τους τίτλους των ταινιών. Έχει ενα φιλικό προς το χρήστη UI για την αλληλεπίδραση με τη βάση δεδομένων, επιτρέποντας στους χρήστες να περιηγούνται και να αναζητούν ταινίες, να προσθέτουν νέες ταινίες, να ενημερώνουν και να διαγράφουν υπάρχουσες ταινίες.
Η συγκριτική ανάλυση κατά τη διάρκεια της διαδικασίας ανάπτυξης της εφαρμογής βοήθησε στον εντοπισμό του πιο αποτελεσματικού συνδυασμού τεχνολογιών για αυτή την εφαρμογή, με αποτέλεσμα μια ομαλή και αποτελεσματική διαδικασία ανάπτυξης. Η ιδέα πίσω από την εφαρμογή είναι η δημιουργία μιας ολοκληρωμένης και πλήρως λειτουργικής διαδικτυακής εφαρμογής που χρησιμοποιεί τα επιλεγμένα frameworks για να αξιολογήσει και να συγκρίνει τα δυνατά και αδύνατα σημεία τους.
Όταν φορτώνεται η κύρια σελίδα, όλες οι ταινίες εμφανίζονται στην οθόνη με χρονολογική σειρά και ο χρήστης μπορεί να εκτελέσει λειτουργίες CRUD σε αυτές, συμπεριλαμβανομένης της δημιουργίας, ανάγνωσης, ενημέρωσης και διαγραφής.
Ο σύνδεσμος της εφαρμογής στο Netlify είναι https://pensive-morse-bc4497.netlify.app/home/main.

Τεχνολογίες που χρησιμοποιήθηκαν

Express.js

Το Express.js είναι ένα δημοφιλές framework για την ανάπτυξη εφαρμογών στην πλατφόρμα του Node.js. Είναι ελαφρύ, ευέλικτο και εύκολο στη χρήση, και παρέχει ένα απλό και εκφραστικό API για την δημιουργία (routes) και την υλοποίηση λειτουργικότητας στον server-side κώδικα. Το Express.js παρέχει μια απλή και ευανάγνωστη σύνταξη για τον ορισμό routes μέσω των HTTP μεθόδων (GET, POST, PUT, DELETE κ.λπ.). Υποστηρίζει τη χρήση middleware, που επιτρέπει την εκτέλεση κώδικα πριν ή μετά από την εκτέλεση ενός route. Χρησιμοποιείται για λειτουργίες όπως η authorization, η επεξεργασία δεδομένων κτλ. Μπορεί επίσης να στείλει στατικά αρχεία, όπως CSS, JavaScript και εικόνες, απευθείας από τον server, χωρίς να χρειάζεται επιπλέον client κωδικας.[1]

Flask

Το Flask είναι ένα ελαφρύ, ευέλικτο και επεκτάσιμο web framework για την ανάπτυξη εφαρμογών σε Python. Αποτελεί μία από τις πιο δημοφιλείς επιλογές για την ανάπτυξη backend λογισμικού, καθώς παρέχει έναν απλό και ευανάγνωστο τρόπο για την υλοποίηση web εφαρμογών. Απαιτεί ελάχιστο κώδικα για να ξεκινήσει μια βασική εφαρμογή. Ο προγραμματιστής έχει πλήρη έλεγχο πάνω στον τρόπο που θα οργανώσει τον κώδικα του και δεν επιβάλλει κανόνες για τον τρόπο που πρέπει να γίνει η οργάνωση του project, επιτρέποντας στον προγραμματιστή να επιλέξει την κατάλληλη δομή για την εφαρμογή του. Παρέχει ένα εύρος από extensions που μπορούν να χρησιμοποιηθούν για την προσθήκη διαφόρων λειτουργιών στην εφαρμογή. Ενσωματωμένη υποστήριξη της WSGI: Υποστηρίζει το WSGI (Web Server Gateway Interface), το οποίο είναι ένα πρότυπο διεπαφής ανάμεσα στον web server και το web application. Αυτό σημαίνει ότι το Flask μπορεί να τρέξει σε διάφορους web servers που υποστηρίζουν το WSGI, όπως ο Gunicorn ή ο uWSGI. Παρέχει επίσης μια εκτεταμένη και καλά τεκμηριωμένη ιστοσελίδα, η οποία περιέχει πληροφορίες, οδηγούς και παραδείγματα χρήσης για την ανάπτυξη εφαρμογών.[2]

Postgres

Η PostgreSQL είναι ένα ισχυρό σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων ανοικτού κώδικα που προσφέρει προηγμένα χαρακτηριστικά, όπως συναλλαγές, ακεραιότητα δεδομένων και την γλώσσα SQL. Χρησιμοποιείται ευρέως για την ανάπτυξη εφαρμογών ιστού που απαιτούν ισχυρές δυνατότητες διαχείρισης δεδομένων. Η PostgreSQL προσφέρει προηγμένη υποστήριξη ακεραιότητας δεδομένων και συναλλαγών, διασφαλίζοντας ότι τα δεδομένα αποθηκεύονται και διαχειρίζονται αξιόπιστα.[3]

MongoDB

Η MongoDB είναι μια βάση δεδομένων NoSQL με χρήση των documents που χρησιμοποιείται για την αποθήκευση και διαχείριση δεδομένων τύπου JSON. Έχει σχεδιαστεί για να είναι επεκτάσιμη, ευέλικτη και υψηλής απόδοσης και χρησιμοποιείται συχνά για εφαρμογές που απαιτούν αποθήκευση και επεξεργασία δεδομένων μεγάλης κλίμακας.[4]

Apache Benchmark

Το Apache Benchmark (ab) είναι ένα εργαλείο γραμμής εντολών που παρέχεται από τον Apache HTTP Server. Χρησιμοποιείται για τη μέτρηση της απόδοσης και της αντοχής ενός web server απέναντι σε φορτίο και αιτήματα. Με το Apache Benchmark, μπορεί να δημιουργηθεί ένα επαναλαμβανόμενο φορτίο αιτημάτων προς έναν web server και να αξιολογηθεί πόσες αιτήσεις ανταποκρίνονται επιτυχώς, πόσος χρόνος απαιτείται για κάθε αίτημα και ποια είναι η συνολική χρονική διάρκεια των αιτημάτων. Το Apache Benchmark είναι χρήσιμο για τον έλεγχο της απόδοσης του web server, τον εντοπισμό αδυναμιών και τη βελτιστοποίηση των ρυθμίσεων του. Μπορεί επίσης να χρησιμοποιηθεί για τη σύγκριση απόδοσης μεταξύ διαφορετικών web servers ή διαμόρφωσης servers. Για να χρησιμοποιηθεί το Apache Benchmark, πρέπει να καθοριστεί ο αριθμός των αιτήσεων ή ο χρόνος διάρκειας, ο αριθμός των συγχρόνων αιτημάτων και η διεύθυνση URL του web server.[5]

Nginx

Ο Nginx είναι ένας web server και reverse proxy server που είναι γνωστός για την υψηλή απόδοση, την αξιοπιστία και την ευελιξία του. Έχει σχεδιαστεί να χειρίζεται μεγάλο network traffic και να παρέχει γρήγορη και αποτελεσματική παράδοση περιεχομένου. Αρχικά, αναπτύχθηκε ως εναλλακτική λύση στον παραδοσιακό server Apache. Η κύρια διαφορά του Nginx είναι ο τρόπος με τον οποίο διαχειρίζεται τα αιτήματα. Ενώ ο Apache χρησιμοποιεί ένα νήμα ανά αίτημα (thread per request) για να εξυπηρετήσει τους πελάτες, ο Nginx χρησιμοποιεί έναν ανεξάρτητο αριθμό νημάτων για να εξυπηρετήσει πολλαπλά αιτήματα. Αυτή η αρχιτεκτονική του Nginx του επιτρέπει να αντέχει σε μεγάλο αριθμό συγχρόνων συνδέσεων και να διαχειρίζεται αποδοτικά τις ζητήσεις των πελατών/clients. Επιπλέον, διαθέτει προηγμένες λειτουργίες όπως load balancing, προστασία από DDoS επιθέσεις και caching. Επιπλέον, μπορεί να λειτουργήσει ως reverse proxy server, που σημαίνει ότι μπορεί να λάβει αιτήσεις από clients και να τις προωθήσει σε backends όπως application servers ή βάσεις δεδομένων.[6]

Docker

Το Docker είναι μια ανοικτού κώδικα πλατφόρμα που χρησιμοποιείται για την αυτοματοποίηση της δημιουργίας, της ανάπτυξης και της εκτέλεσης εφαρμογών με χρήση ελαφρών, φορητών και αυτοδύναμων περιβαλλόντων, γνωστών ως "containers". Τα containers επιτρέπουν την εκτέλεση μιας εφαρμογής και των απαραίτητων dependencies (βιβλιοθήκες, περιβάλλον εκτέλεσης) σε ένα απομονωμένο περιβάλλον. Το Docker παρέχει ένα ευέλικτο και απομονωμένο περιβάλλον για την εκτέλεση εφαρμογών, επιτρέποντας την εγκατάσταση και την εκτέλεση πολλών τεχνολογιών σε έναν κοινό υπολογιστικό πόρο, χωρίς να επηρεάζονται από τις αλλαγές στο υπόλοιπο σύστημα. Κάθε container είναι απομονωμένο και αυτοδύναμο, επιτρέποντας την εύκολη μεταφορά και αναπαραγωγή εφαρμογών σε διάφορες πλατφόρμες. Ένα από τα κύρια πλεονεκτήματα του Docker είναι η ταχύτητα και η ευκολία παραμετροποίησης και αναπαραγωγής των περιβαλλόντων εκτέλεσης. Με τη χρήση Docker images, μπορούν να δημιουργηθούν επαναλαμβανόμενα περιβάλλοντα εκτέλεσης που περιλαμβάνουν όλα τα απαραίτητα dependencies, επιτρέποντας την εκτέλεση των εφαρμογών με συνέπεια σε διάφορες μηχανές και περιβάλλοντα. Το Docker είναι ευέλικτο και ευρέως χρησιμοποιούμενο, καθώς μπορεί να ενσωματωθεί σε διάφορα περιβάλλοντα ανάπτυξης και παραγωγής, όπως σε επικείμενες εφαρμογές, frameworks και υποδομές υψηλής κλίμακας.[7]

Angular

Η Angular είναι ένα πλήρες και εκτεταμένο framework ανάπτυξης διεπαφών χρήστη (frontend) για τη δημιουργία επαγγελματικών εφαρμογών web. Αναπτύχθηκε από την ομάδα της Google και είναι βασισμένο στη γλώσσα προγραμματισμού TypeScript. Χρησιμοποιεί data binding μεταξύ του μοντέλου δεδομένων (model) και της διεπαφής (view). Παρέχει ένα ισχυρό σύστημα routing που επιτρέπει την πλοήγηση μεταξύ διαφόρων σελίδων και περιεχομένων μέσω του URL. Χρησιμοποιεί services για το business logic και λειτουργίες που μοιράζονται ανάμεσα στα components της εφαρμογής. Τα services παρέχουν την δυνατότητα να γίνεται ανεξάρτητη ανάπτυξη και διαχείριση λειτουργιών όπως η ανάκτηση δεδομένων από APIs, η αποθήκευση και ο χειρισμός δεδομένων, η επικοινωνία με τον server, και πολλά άλλα. Υιοθετεί την αρχιτεκτονική Model-View-Controller (MVC), η οποία βοηθά στον διαχωρισμό της λογικής της εφαρμογής, την εμφάνιση των δεδομένων και την αλληλεπίδραση του χρήστη. Αυτό καθιστά τον κώδικα πιο ευανάγνωστο, ευκολότερο στη συντήρηση και την ανάπτυξη

ομάδων. Τέλος, διαθέτει ένα πλούσιο οικοσύστημα επεκτάσεων και πακέτων, το οποίο περιλαμβάνει εργαλεία, βιβλιοθήκες και πρότυπα για την ευκολία της ανάπτυξης. Υπάρχουν πολλά πακέτα τρίτων που παρέχουν πρόσθετες δυνατότητες και λειτουργίες για το Angular.[13]

Σύγκριση των τεχνολογιών

Η επιλογή των τεχνολογιών

Στόχος της παρούσας διατριβής είναι να διερευνήσει και να συγκρίνει τα χαρακτηριστικά που παρέχει ένας επιλεγμένος αριθμός frameworks και βασεων δεδομενων. Επιλέχθηκαν συνολικά 3 frameworks (front end και back end) και 2 βασεις δεδομενων. Πώς όμως έγινε η επιλογή αυτων;

Ο κύριος παράγοντας που λήφθηκε υπόψη κατά την επιλογή τους ήταν η συνολική δημοτικότητά τους. Ένας τρόπος για να μετρήσουμε τη δημοτικότητα μιας τεχνολογίας είναι να μετρήσουμε πόσα αστέρια GitHub έχει το repo του.[8][9][10][11][12]

- ➔ Flask: Είναι ένα ελαφρύ και ευέλικτο πλαίσιο (framework) για την ανάπτυξη διαδικτυακών εφαρμογών με τη χρήση της γλώσσας προγραμματισμού Python. Έχει αποκτήσει μεγάλη δημοφιλία λόγω της απλότητας του, της εκτενούς τεκμηρίωσης και της ενεργής κοινότητας χρηστών. Η δημοφιλία του Flask και τα χιλιάδες αστέρια στο GitHub δείχνουν την ευρεία αποδοχή του από την κοινότητα των προγραμματιστών. [2]
- ➔ Express: Επίσης ένα ελαφρύ και ευέλικτο framework για την ανάπτυξη διαδικτυακών εφαρμογών με τη χρήση της γλώσσας προγραμματισμού JavaScript και του περιβάλλοντος εκτέλεσης Node.js. Έχει μεγάλη δημοφιλία στην κοινότητα των προγραμματιστών JavaScript και είναι αξιόπιστο και αποτελεσματικό. Οι πολλές συνεισφορές και τα GitHub stars καταδεικνύουν την ευρεία υποστήριξή του επίσης.[1]
- ➔ Postgres: Έχει εξαιρετική φήμη για την αξιοπιστία, την απόδοση και τις προηγμένες λειτουργίες που προσφέρει. Η δημοφιλία της Postgres αυξάνεται συνεχώς και τα GitHub stars του αποδεικνύουν την εκτίμηση που διακατέχει από την κοινότητα των προγραμματιστών.[3]
- ➔ MongoDB: Προσφέρει ευελιξία και επιτρέπει την αποθήκευση, ανάκτηση και επεξεργασία δεδομένων σε μορφή JSON documents. Η δημοφιλία της MongoDB αυξάνεται διαρκώς, ειδικά στον τομέα των εφαρμογών που απαιτούν ευελιξία και κλιμακωσιμότητα.[4]
- ➔ Angular: Είναι ένα πολύ δημοφιλές framework για την ανάπτυξη μοντέρνων διαδικτυακών εφαρμογών. Η διαθεσιμότητα πληθώρας πρόσθετων εργαλείων, η

ενεργή κοινότητα των προγραμματιστών και τα GitHub stars αποδεικνύουν την αναγνώριση που έχει αποκτήσει.[13]

Κριτήρια σύγκρισης των τεχνολογιών

Είναι πολύ σημαντικό να καθοριστούν τα κριτήρια που θα χρησιμοποιηθούν για την αξιολόγηση των επιλεγμένων frameworks και βάσεων δεδομένων. Παρακάτω παρουσιάζονται τα κριτήρια που επιλέχθηκαν. Να σημειωθεί ότι αυτά τα κριτήρια επιλέχθηκαν επειδή αντικατοπτρίζουν ορισμένες βασικές απαιτήσεις μιας σύγχρονης διαδικτυακής εφαρμογής. Για μια πιο σύνθετη εφαρμογή, ενδέχεται να απαιτούνται ορισμένες επιπλέον παράμετροι.

- ***Frameworks***
  - ➔ *Απόδοση και κλιμακωσιμότητα*: Πώς ανταποκρίνεται το framework σε μεγάλο αριθμό ταυτόχρονων αιτημάτων και ποια είναι η ταχύτητά του σε σχέση με την επεξεργασία δεδομένων;
  - ➔ *Authentication και ασφάλεια*: Ποιες δυνατότητες παρέχει το framework για την ασφάλεια των δεδομένων και το authorization των χρηστών;
  - ➔ *Δυνατότητες API*: Πώς υποστηρίζει το framework τη δημιουργία και τη διαχείριση των API για την αλληλεπίδραση με εξωτερικές υπηρεσίες;
  - ➔ *Ευελιξία και προσαρμοστικότητα*: Πόσο εύκολο είναι να προσαρμοστεί η ομάδα με τη χρήση του framework;
  - ➔ *Κοινότητα και υποστήριξη*: Πόσο ενεργή είναι η κοινότητα γύρω από το framework και πώς παρέχεται η υποστήριξη από τους δημιουργούς και άλλους χρήστες του;
  - ➔ *Βάσεις δεδομένων*: Ποιες βάσεις δεδομένων υποστηρίζει το framework και πώς διευκολύνει την αλληλεπίδραση με τη βάση δεδομένων;[14][18][23][26][27]

- ***Βάσεις δεδομένων***
  - ➔ *Μοντέλο δεδομένων*: Ορισμένες βάσεις δεδομένων χρησιμοποιούν σχεσιακό μοντέλο δεδομένων, ενώ άλλες χρησιμοποιούν μη σχεσιακά μοντέλα όπως το document-oriented (για παράδειγμα, MongoDB) ή το key-value (για παράδειγμα, Redis).

- *Απόδοση*: Η απόδοση μιας βάσης δεδομένων είναι σημαντική για την αποτελεσματική εκτέλεση των queries και την ανταπόκριση της εφαρμογής σας. Παράμετροι όπως ο χρόνος απόκρισης και η δυνατότητα κλιμάκωσης πρέπει να ληφθούν υπόψη κατά την επιλογή μιας βάσης δεδομένων.
- *Κλιμάκωση*: Ορισμένες βάσεις δεδομένων είναι πιο κατάλληλες για κλιμάκωση, δηλαδή την δυνατότητα να ανταποκρίνονται σε αυξημένο φορτίο εργασίας ή μεγάλους όγκους δεδομένων.
- *Ανάγνωση/Εγγραφή*: Ορισμένες βάσεις δεδομένων είναι πιο κατάλληλες για εφαρμογές που απαιτούν περισσότερες εγγραφές (write-intensive), ενώ άλλες είναι πιο κατάλληλες για εφαρμογές που απαιτούν περισσότερες αναγνώσεις (read-intensive).
- *Συνέπεια και αντοχή σε σφάλματα*: Ορισμένες βάσεις δεδομένων προσφέρουν υψηλή συνέπεια των δεδομένων, ενώ άλλες επιτρέπουν ασύγχρονες ενημερώσεις και ενδέχεται να έχουν μικρότερη συνέπεια σε περίπτωση σφάλματος.
- *Εργαλεία και Οικοσύστημα*: Η ύπαρξη εργαλείων ανάπτυξης, οι πόροι της κοινότητας και η υποστήριξη είναι επίσης σημαντικοί παράγοντες κατά την επιλογή μιας βάσης δεδομένων.[15][16][24][25]

Python vs NodeJs

- Εύκολη μάθηση και χρήση:
  - Python: Είναι μια εύκολη στην κατανόηση γλώσσα προγραμματισμού με σύνταξη και πλούσια κοινότητα που παρέχει πολλά παραδείγματα και πόρους.
  - Node.js: Απαιτεί κάποια προηγουμενη ενασχοληση με JavaScript, αλλά αν υπάρχει εξοικείωση η μετάβαση γίνεται εύκολα.
- Επεκτασιμότητα και οικοσύστημα:
  - Python: Έχει μια εκτεταμένη συλλογή frameworks (όπως Django, Flask) που καλύπτουν πολλές ανάγκες ανάπτυξης εφαρμογών.
  - Node.js: Διαθέτει ένα εκτεταμένο οικοσύστημα πακέτων (npm) που παρέχει πολλά πακέτα και εργαλεία για την ανάπτυξη εφαρμογών
- Απόδοση και κλιμακωσιμότητα:
  - Python: Είναι γενικά πιο αργή από το Node.js όταν πρόκειται για

επεξεργασία μεγάλου όγκου δεδομένων ή παράλληλες εργασίες.

- Node.js: Είναι γνωστό για την ασύγχρονη και μη μπλοκαρισμένη φύση του, κάνοντάς το κατάλληλο για εφαρμογές που απαιτούν υψηλή απόδοση και κλιμακωσιμότητα.

➔ Κοινότητα και υποστήριξη:

- Python: Έχει μια μεγάλη και ενεργή κοινότητα προγραμματιστών, προσφέροντας σταθερή υποστήριξη και συχνές ενημερώσεις.
- Node.js: Έχει επίσης μια μεγάλη και αναπτυσσόμενη κοινότητα που παρέχει πολλούς πόρους και εργαλεία.[19][20][21]

ExpressJs vs Flask

➔ Απόδοση και κλιμακωσιμότητα:

- Flask: Είναι ένα microframework που προσφέρει βασική λειτουργικότητα, καθιστώντας το πιο ελαφρύ και απλό σε σχέση με άλλα frameworks. Είναι κατάλληλο για μικρές έως μεσαίες εφαρμογές με χαμηλό traffic και απαιτήσεις. Ωστόσο, με την κατάλληλη δομή και βελτιστοποίηση, μπορεί να χειριστεί μεγαλύτερο φορτίο.
- Express.js: Είναι επίσης ένα ελαφρύ framework, αλλά παρέχει περισσότερες δυνατότητες σε σύγκριση με το Flask. Έχει ευέλικτη δομή και επιτρέπει την εύκολη προσαρμογή σε διάφορες απαιτήσεις εφαρμογών. Είναι κατάλληλο για μεσαίες έως μεγάλες εφαρμογές και μπορεί να κλιμακωθεί καλά για υψηλό traffic.

➔ Βάσεις δεδομένων:

- Flask: Δεν παρέχει ένα συγκεκριμένο ORM (Object-Relational Mapping) ή driver για βάσεις δεδομένων. Ωστόσο, μπορεί να χρησιμοποιηθεί με διάφορες βάσεις δεδομένων μέσω της εγκατάστασης αντίστοιχων εργαλείων, όπως το SQLAlchemy για συμβατότητα με πολλές βάσεις.
- Express.js: Μπορεί να χρησιμοποιηθεί με διάφορες βάσεις δεδομένων, αλλά συχνά συνδυάζεται με τη χρήση της MongoDB.

➔ Κοινότητα και υποστήριξη:

- Flask: Έχει μια ενεργή κοινότητα που παρέχει συνεχή υποστήριξη και ενημερώσεις. Υπάρχουν πληθώρα πόρων, ενημερωμένη τεκμηρίωση και

μια μεγάλη ποικιλία από παραδείγματα και επεκτάσεις που είναι διαθέσιμες στην κοινότητα.

- Express.js: Έχει επίσης μια μεγάλη και ενεργή κοινότητα που παρέχει υποστήριξη και πόρους για τους προγραμματιστές. Υπάρχουν πληθώρα πακέτων, middlewares και παραδειγμάτων που μπορούν να χρησιμοποιηθούν για να επεκτείνουν τις δυνατότητες του framework.

➔ Ευκολία χρήσης και μάθησης:

- Flask: θεωρείται ευκολότερο στην εκμάθηση και στη χρήση για αρχάριους προγραμματιστές. Έχει μια απλή και σαφή σύνταξη, μικρότερο αριθμό απαιτούμενων γραμμών κώδικα και προσφέρει μια ελαφριά προσέγγιση στην ανάπτυξη εφαρμογών.
- Express.js: μπορεί να έχει μια ελαφρώς μεγαλύτερη καμπύλη μάθησης για αρχάριους προγραμματιστές, ειδικά αν δεν έχουν εμπειρία με το Node.js. Ωστόσο, μετά την εξοικείωση με τα βασικά, το Express.js παρέχει ένα ισχυρό και ευέλικτο περιβάλλον ανάπτυξης.

➔ Δυνατότητες API: Και το Flask και το Express.js παρέχουν δυνατότητες για τη δημιουργία προηγμένων API.

- Flask υποστηρίζει προαιρετικά την επέκταση Flask-RESTful που παρέχει εργαλεία για την εύκολη δημιουργία και διαχείριση πόρων RESTful.
- Express.js διαθέτει ευέλικτα middleware και μεθόδους routing για την υλοποίηση πολύπλοκων λειτουργιών API. Μπορεί να προσαρμοστεί το authentication, το authorization και η ασφάλεια σύμφωνα με τις απαιτήσεις.

➔ Ευελιξία και προσαρμοστικότητα: Και το Flask και το Express.js είναι εξαιρετικά ευέλικτα και προσαρμόσιμα.

- Flask παρέχει μια ελαφριά βάση και επιτρέπει τη χρήση πρόσθετων για την προσθήκη λειτουργικοτητας.
- Express.js παρέχει ένα ευέλικτο μοντέλο middleware που επιτρέπει την προσθήκη λειτουργικοτητας και την προσαρμογή της ροής του προγράμματος.

➔ Authentication και ασφάλεια:

- Το Flask παρέχει ευέλικτες βιβλιοθήκες όπως το Flask-Login και το Flask-Security για authorization και ασφάλεια.

- Express.js: δεν παρέχει απευθείας ενσωματωμένη λειτουργικότητα για authorization και ασφάλεια. Ωστόσο, μπορούν να χρησιμοποιηθούν πακέτα όπως το Passport.js και το Express-Session για authorization μέσω tokens, των συνδεδεμένων λογαριασμών κοινωνικών δικτύων ή άλλων μεθόδων.[14][1][2]

Postgres vs MongoDB

➔ Μοντέλο Δεδομένων:
- PostgreSQL: Υποστηρίζει σχεσιακό μοντέλο δεδομένων, με πίνακες, σχέσεις και περιορισμούς συμμόρφωσης (constraints). Παρέχει σταθερή δομή δεδομένων και συμμόρφωση προς το σχήμα της βάσης δεδομένων.
- MongoDB: Χρησιμοποιεί document-oriented μοντέλο δεδομένων, με εύκολο σχεδιασμό και ευελιξία. Δεν απαιτεί σταθερή δομή δεδομένων και επιτρέπει την αποθήκευση δεδομένων σε μορφή JSON-εμπλουτισμένων documents.

➔ Απόδοση:
- PostgreSQL: Είναι σχεδιασμένη για να παρέχει υψηλή απόδοση και επεξεργασία μεγάλου όγκου δεδομένων. Υποστηρίζει πολλαπλούς τύπους δεικτών, βελτιστοποιητές ερωτημάτων και προηγμένη διαχείριση μνήμης.
- MongoDB: Παρέχει γρήγορη ανάγνωση και γράφημα για ευέλικτη επεξεργασία δεδομένων. Ωστόσο, η απόδοση μπορεί να επηρεαστεί από το μέγεθος της βάσης δεδομένων και την πολυπλοκότητα των ερωτημάτων.

➔ Κλιμάκωση:
- PostgreSQL: Παρέχει καλή κλιμάκωση και δυνατότητα διαχείρισης μεγάλου όγκου δεδομένων. Μπορεί να υποστηρίξει υψηλή παράλληλη επεξεργασία και αυξητικό σχεδιασμό (sharding) για αντιμετώπιση μεγάλου φόρτου.
- MongoDB: Είναι σχεδιασμένη για οριζόντια κλιμάκωση και μπορεί να αντιμετωπίσει μεγάλο όγκο δεδομένων και υψηλές απαιτήσεις. Προσφέρει ευέλικτο σχεδιασμό και αυτόματο sharding για ομαλή αύξηση της χωρητικότητας.

➔ Συνέπεια:

- PostgreSQL: Υποστηρίζει ACID (Atomicity, Consistency, Isolation, Durability) ιδιότητες, παρέχοντας ισχυρές εγγυήσεις για συνέπεια, ακεραιότητα και αντοχή σε σφάλματα.
- MongoDB: Προσφέρει ευέλικτη συνέπεια με βάση το μοντέλο συνέπειας. Υποστηρίζει τόσο συνέπεια μοναδικού κόμβου όσο και συνέπεια εύρους.

➔ Αντοχή σε σφάλματα:

- PostgreSQL: Έχει ενσωματωμένα μηχανισμούς για ανάκτηση από σφάλματα και προστασία των δεδομένων από απώλεια. Υποστηρίζει ασφαλή συναλλαγές και διαχείριση σφαλμάτων.
- MongoDB: Παρέχει αυτόματη ανάκτηση από σφάλματα, αντιγραφή αντιγράφων ασφαλείας (replication) και αποκατάσταση αντιγράφων ασφαλείας (failover). Επίσης, παρέχει επιλογές για αντιγραφή αντιγράφων ασφαλείας σε διάφορα τοπικά και απομακρυσμένα σημεία.

➔ Εργαλεία και οικοσύστημα:

- PostgreSQL: Έχει ευρεία κοινότητα και υποστηρίζεται από πολλά εργαλεία ανάπτυξης, διαχείρισης και επικοινωνίας. Παρέχει επίσης πληθώρα επιπρόσθετων πακέτων και επεκτάσεων.
- MongoDB: Διαθέτει επίσης εκτεταμένη κοινότητα και υποστηρίζεται από διάφορα εργαλεία ανάπτυξης και διαχείρισης. Παρέχει επίσης επιπλέον χαρακτηριστικά όπως ευέλικτες δυνατότητες αποθήκευσης σε cache και δυνατότητες αναζήτησης πλήρους κειμένου.[15][16][22]

Τεκμηρίωση, αποτελέσματα και σχολιασμός πειραμάτων

Στην demo εφαρμογή που αναπτύχθηκε, τα requests που χρησιμοποιούνται είναι τα GET, POST entity και PUT, οπότε οι δοκιμές επιδόσεων πραγματοποιήθηκαν για

αυτούς τους τύπους αιτημάτων. Να σημειωθεί ότι σε κάθε αίτημα GET λαμβάνονται συνολικά περίπου 8500 εγγραφές της βάσης δεδομένων. Το hardware που χρησιμοποιήθηκε είναι Macbook Pro M1 2021 32gb Ram. Ο server του κάθε framework τρέχει τοπικά στο localhost. Για το benchmarking χρησιμοποιήθηκε το ab benchmarking tool από https://httpd.apache.org/docs/2.4/programs/ab.html το οποίο εξάγει ένα αρκετά λεπτομερές report των πειραμάτων.

Express.js με Postgres

- ➔ GET:
  - Ο αριθμός των αιτήσεων που ολοκληρώθηκαν είναι 412.
  - Ο μέσος χρόνος ανά αίτημα είναι 244.764 ms, ενώ ο μέσος χρόνος ανά αίτημα για όλα τα αιτήματα είναι 24.476 ms.
  - Ο μέγιστος χρόνος αίτησης ήταν 282 ms.
- ➔ POST:
  - Ο αριθμός των αιτήσεων που ολοκληρώθηκαν είναι 16278.
  - Ο μέσος χρόνος ανά αίτημα είναι 8.285 ms, ενώ ο μέσος χρόνος ανά αίτημα για όλα τα αιτήματα είναι 0.829 ms.
  - Ο μέγιστος χρόνος αίτησης ήταν 49 ms.
- ➔ PUT:
  - Ο αριθμός των αιτήσεων που ολοκληρώθηκαν είναι 11848.
  - Ο μέσος χρόνος ανά αίτημα είναι 8.440 ms, ενώ ο μέσος χρόνος ανά αίτημα για όλα τα αιτήματα είναι 0.844 ms.
  - Ο μέγιστος χρόνος αίτησης ήταν 203 ms.

Ο αριθμός των αιτήσεων που ολοκληρώθηκαν για τις διάφορες μεθόδους (GET, POST, PUT) είναι αρκετά υψηλός, καθώς πραγματοποιήθηκαν χιλιάδες αιτήσεις σε κάθε πείραμα. Ο μέσος χρόνος απόκρισης για τις αιτήσεις GET είναι σχετικά υψηλός (244.764 ms), ενώ για τις αιτήσεις POST και PUT είναι αρκετά χαμηλός (8.285 ms και 8.440 ms αντίστοιχα). Ο μέγιστος χρόνος αίτησης για τη μέθοδο GET ήταν 282 ms, ενώ για τις μεθόδους POST και PUT ήταν 49 ms και 203 ms αντίστοιχα. Ο μέσος ρυθμός μεταφοράς για τις αιτήσεις GET είναι σχετικά υψηλός (73564.06 Kbytes/sec), ενώ για τις αιτήσεις POST και PUT είναι αρκετά χαμηλός (514.96 Kbytes/sec και 328.62 Kbytes/sec αντίστοιχα). Η μέθοδος GET έχει τον υψηλότερο μέσο χρόνο απόκρισης και μέγιστο χρόνο αίτησης, ενώ οι μέθοδοι POST και PUT έχουν χαμηλότερους μέσους χρόνους

απόκρισης και μέγιστους χρόνους αίτησης. Οι αιτήσεις POST έχουν τον υψηλότερο ρυθμό αιτήσεων ανά δευτερόλεπτο (1207.00 requests/sec), ενώ οι μέθοδοι GET και PUT έχουν χαμηλότερους ρυθμούς αιτήσεων ανά δευτερόλεπτο (40.86 requests/sec και 1184.79 requests/sec αντίστοιχα). Οι μέθοδοι POST και PUT έχουν αρκετά μικρότερο μέγεθος μεταφερόμενων δεδομένων σε σύγκριση με τη μέθοδο GET. Βάσει αυτών των αποτελεσμάτων, μπορούμε να συμπεράνουμε ότι οι μέθοδοι POST και PUT είναι γενικά πιο αποδοτικές από την μέθοδο GET, καθώς έχουν χαμηλότερους μέσους χρόνους απόκρισης και μέγιστους χρόνους αίτησης, καθώς και υψηλότερους ρυθμούς αιτήσεων ανά δευτερόλεπτο. Στις παρακάτω εικόνες μπορούμε να δούμε τα αποτελέσματα του benchmark για GET, POST, PUT όπως εξήχθησαν από το ab tool.

Εικόνα 1 ab benchmarking tool output GET

GET

Server Hostname: 127.0.0.1 Server Port: 3000

Document Path: /titles Document Length: 1820244 bytes

Concurrency Level: 10

- Time taken for tests: 10.084 seconds
- Complete requests: 412 Failed requests: 0 Total transferred: 759643108 bytes HTML transferred: 759538700 bytes
- Requests per second: 40.86 [#/sec] (mean)
- Time per request: 244.764 [ms] (mean) Time per request: 24.476 [ms] (mean, across all concurrent requests) Transfer rate: 73564.06 [Kbytes/sec] received

Percentage of the requests served within a certain time (ms) 50% 237 66% 240 75% 243 80% 246 90% 255 95% 265 98% 276 99% 279 100% 282 (longest request)

Εικόνα 2 ab benchmarking tool output POST

POST

Concurrency Level: 10

- Time taken for tests: 13.486 seconds
- Complete requests: 16278 Failed requests: 1644 (Connect: 0, Receive: 0, Length: 1644, Exceptions: 0) Total transferred: 7111649 bytes Total body sent: 4135374 HTML transferred: 3123539 bytes
- Requests per second: 1207.00 [#/sec] (mean)
- Time per request: 8.285 [ms] (mean) Time per request: 0.829 [ms] (mean, across all concurrent requests) Transfer rate: 514.96 [Kbytes/sec] received 299.45 kb/s sent 814.41 kb/s total

Connection Times (ms) min mean[+/-sd] median max Connect: 0 0 0.8 0 30 Processing: 1 4 2.5 3 49 Waiting: 0 4 2.3 3 48 Total: 1 4 2.6 3 49

Percentage of the requests served within a certain time (ms) 50% 3 66% 4 75% 4 80% 5 90% 7 95% 8 98% 10 99% 12 100% 49 (longest request)

Εικόνα 3 ab benchmarking tool output PUT

PUT

Concurrency Level: 10 Time taken for tests: 10.000 seconds

- Complete requests: 11848 Failed requests: 0 Total transferred: 3365116 bytes Total body sent: 2715253 HTML transferred: 533205 bytes
- Requests per second: 1184.79 [#/sec] (mean)
- Time per request: 8.440 [ms] (mean) Time per request: 0.844 [ms] (mean, across all concurrent requests) Transfer rate: 328.62 [Kbytes/sec] received 265.16 kb/s sent 593.78 kb/s total

Connection Times (ms) min mean[+/-sd] median max Connect: 0 0 0.3 0 12 Processing: 1 8 5.1 7 202 Waiting: 1 8 5.1 7 201 Total: 1 8 5.1 7 203

Percentage of the requests served within a certain time (ms) 50% 7 66% 9 75% 10 80% 12 90% 15 95% 17 98% 21 99% 24 100% 203 (longest request)

Express.js με MongoDB

➔ GET

- Ο αριθμός των ολοκληρωμένων αιτήσεων είναι 116.
- Ο μέσος χρόνος απόκρισης για τις αιτήσεις GET είναι 862.191 ms και ο μέγιστος χρόνος αίτησης είναι 1209 ms.

➔ POST

- Ο αριθμός των ολοκληρωμένων αιτήσεων είναι 16271.
- Ο μέσος χρόνος απόκρισης για τις αιτήσεις POST είναι 7.680 ms και ο μέγιστος χρόνος αίτησης είναι 114 ms.

➔ PUT

- Ο αριθμός των ολοκληρωμένων αιτήσεων είναι 12009.
- Ο μέσος χρόνος απόκρισης για τις αιτήσεις PUT είναι 8.329 ms και ο μέγιστος χρόνος αίτησης είναι 145 ms.

Οι μέθοδοι POST και PUT έχουν συνήθως χαμηλότερους μέσους χρόνους απόκρισης σε σύγκριση με τη μέθοδο GET. Οι μέθοδοι POST και PUT έχουν υψηλότερους μέσους ρυθμούς αιτήσεων ανά δευτερόλεπτο σε σύγκριση με τη μέθοδο GET. Επομένως, οι μέθοδοι POST και PUT εμφανίζουν βελτιωμένες επιδόσεις σε σύγκριση με τη μέθοδο GET, καθώς έχουν χαμηλότερους μέσους χρόνους απόκρισης, υψηλότερους μέσους ρυθμούς αιτήσεων ανά δευτερόλεπτο. Στη συνέχεια μπορούμε να δούμε τα αποτελέσματα του benchmark για GET, POST, PUT όπως εξήχθησαν από το ab tool.

Εικόνα 4 ab benchmarking tool output GET

GET

- Concurrency Level: 10
- Time taken for tests: 10.001 seconds
- Complete requests: 116 Failed requests: 0 Non-2xx responses: 116 Total transferred: 33756 bytes HTML transferred: 3828 bytes
- Requests per second: 11.60 [#/sec] (mean)
- Time per request: 862.191 [ms] (mean) Time per request: 86.219 [ms] (mean, across all concurrent requests) Transfer rate: 3.30 [Kbytes/sec] received

Connection Times (ms) min mean[+/-sd] median max Connect: 0 0 0.1 0 1 Processing: 214 810 142.0 797 1209 Waiting: 214 795 147.5 785 1205 Total: 214 810 142.0 797 1209

Percentage of the requests served within a certain time (ms) 50% 797 66% 860 75% 884 80% 899 90% 995 95% 1027 98% 1137 99% 1208 100% 1209 (longest request)

Εικόνα 5 ab benchmarking tool output POST

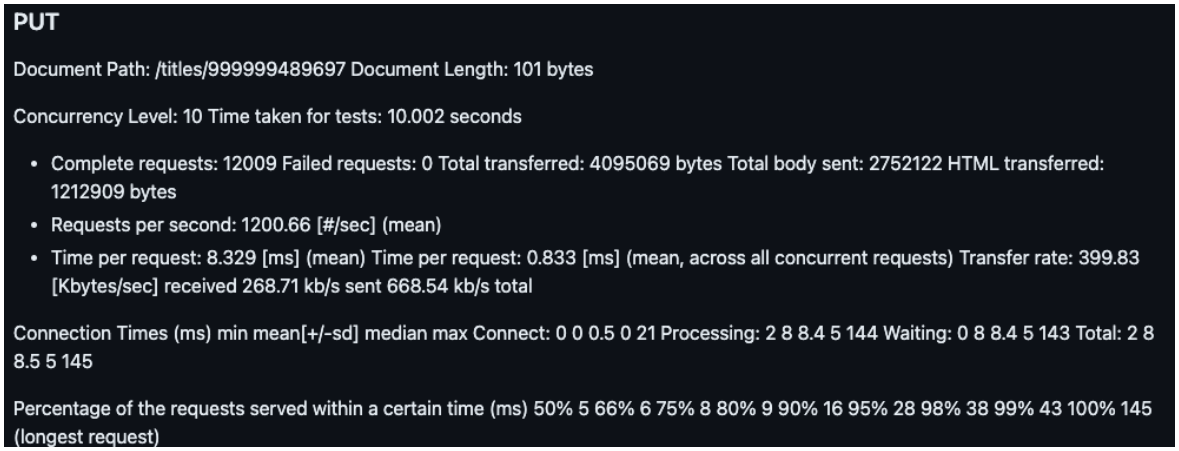
POST

- Concurrency Level: 10
- Time taken for tests: 12.495 seconds
- Complete requests: 16271 Failed requests: 0 Total transferred: 6134167 bytes Total body sent: 4133088 HTML transferred: 2229127 bytes
- Requests per second: 1302.16 [#/sec] (mean)
- Time per request: 7.680 [ms] (mean) Time per request: 0.768 [ms] (mean, across all concurrent requests) Transfer rate: 479.41 [Kbytes/sec] received 323.02 kb/s sent 802.42 kb/s total

Connection Times (ms) min mean[+/-sd] median max Connect: 0 0 0.4 0 36 Processing: 0 3 5.0 2 114 Waiting: 0 3 5.0 2 114 Total: 0 4 5.0 2 114

Percentage of the requests served within a certain time (ms) 50% 2 66% 3 75% 3 80% 3 90% 5 95% 7 98% 18 99% 26 100% 114 (longest request)

Εικόνα 6 ab benchmarking tool output PUT



Flask με Postgres

Μετά τη λήξη 10 δευτερολέπτων από αιτήσεις προς τον server και την παράλληλη εκτέλεση 10 αιτημάτων ανά φορά, παρατηρούμε τα εξής αποτελέσματα.

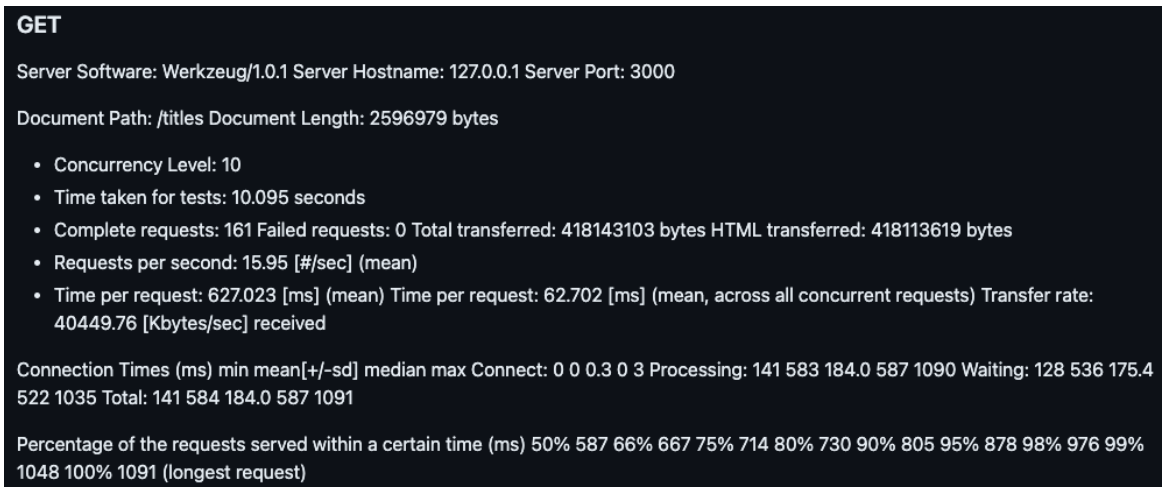
- ➔ GET
  - ◆ Συνολικός αριθμός αιτήσεων: 161
  - ◆ Ρυθμός αιτήσεων: 15.95 αιτήσεις/δευτερόλεπτο (μέσος όρος)
  - ◆ Χρόνος ανά αίτηση: 627.023 ms (μέση τιμή)
- ➔ POST
  - ◆ Συνολικός αριθμός αιτήσεων: 6451
  - ◆ Ρυθμός αιτήσεων: 645.07 αιτήσεις/δευτερόλεπτο (μέσος όρος)
  - ◆ Χρόνος ανά αίτηση: 15.502 ms (μέση τιμή)
- ➔ PUT
  - ◆ Συνολικός αριθμός αιτήσεων: 8030
  - ◆ Ρυθμός αιτήσεων: 802.93 αιτήσεις/δευτερόλεπτο (μέσος όρος)
  - ◆ Χρόνος ανά αίτηση: 12.454 ms (μέση τιμή)

Η μέθοδος GET επιτυγχάνει έναν μέτριο ρυθμό αιτήσεων με περίπου 15.95 αιτήσεις/δευτερόλεπτο, και ο μέσος χρόνος ανά αίτηση είναι σχετικά υψηλός στα 627.023 ms. Ωστόσο, ο ρυθμός μεταφοράς είναι αρκετά καλός με 40,449.76 Kbytes/δευτερόλεπτο. Η μέθοδος POST έχει έναν υψηλό ρυθμό αιτήσεων με περίπου 645.07 αιτήσεις/δευτερόλεπτο, και ο μέσος χρόνος ανά αίτηση είναι σχετικά χαμηλός στα 15.502 ms. Ο ρυθμός μεταφοράς για την αποστολή και τη λήψη είναι επίσης αρκετά καλός. Η μέθοδος PUT έχει τον υψηλότερο ρυθμό αιτήσεων με περίπου 802.93

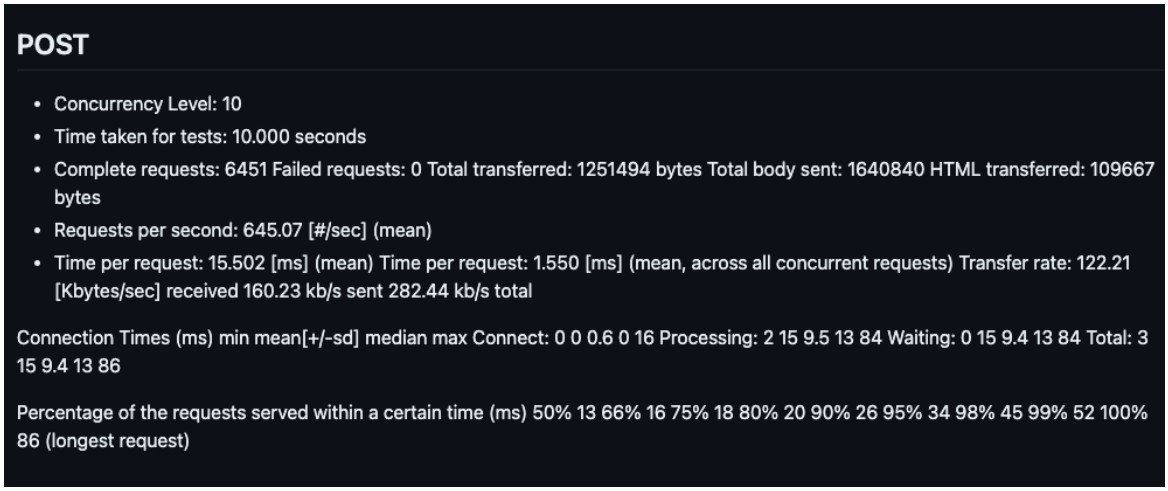
αιτήσεις/δευτερόλεπτο, και ο μέσος χρόνος ανά αίτηση είναι σχετικά χαμηλός στα 12.454 ms. Ο ρυθμός μεταφοράς για την αποστολή και τη λήψη είναι επίσης ικανοποιητικός.

Στις ακόλουθες εικόνες φαίνονται τα αποτελέσματα του benchmark για GET, POST, PUT όπως εξήχθησαν από το ab tool.

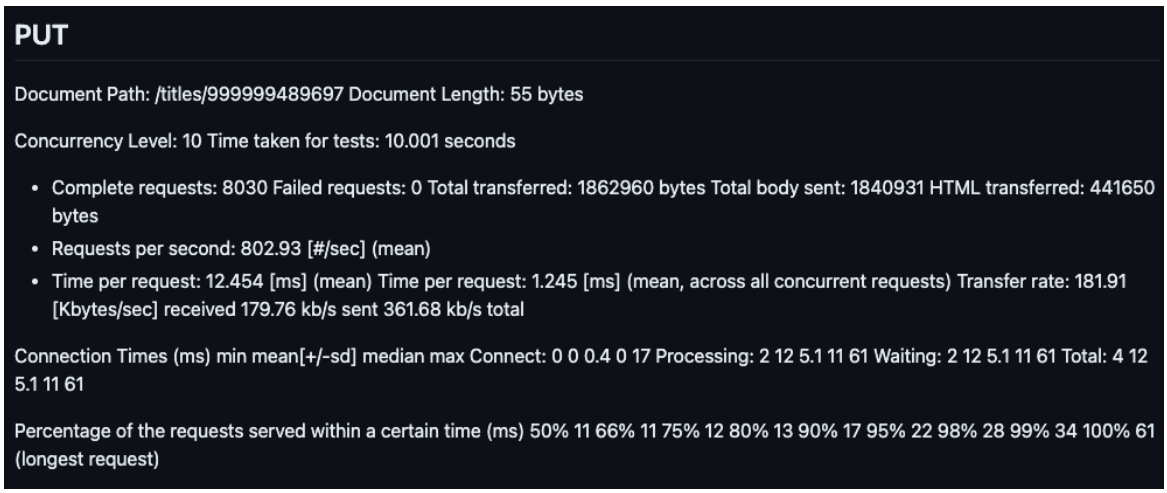
Εικόνα 7 ab benchmarking tool output GET



Εικόνα 8 ab benchmarking tool output POST



Εικόνα 9 ab benchmarking tool output PUT



Flask με MongoDB

Αφού περάσουν 10 δευτερόλεπτα αιτήσεων στον server ενώ εκτελούνται ταυτόχρονα 10 αιτήματα κάθε φορά, τα αποτελέσματα είναι τα εξής.

- ➔ GET
  - ◆ Εκτελέστηκαν 106 αιτήσεις με
  - ◆ Μέσο όρο 10.47 αιτήσεις ανά δευτερόλεπτο.
  - ◆ Ο μέσος χρόνος ανά αίτηση ήταν 95.532 ms
  - ◆ Σημαντικά ποσοστά των αιτήσεων εξυπηρετήθηκαν σε αποδεκτό χρόνο, με το 95% των αιτήσεων να ολοκληρώνονται εντός 1252 ms.
- ➔ POST
  - ◆ Εκτελέστηκαν 9561 αιτήσεις με

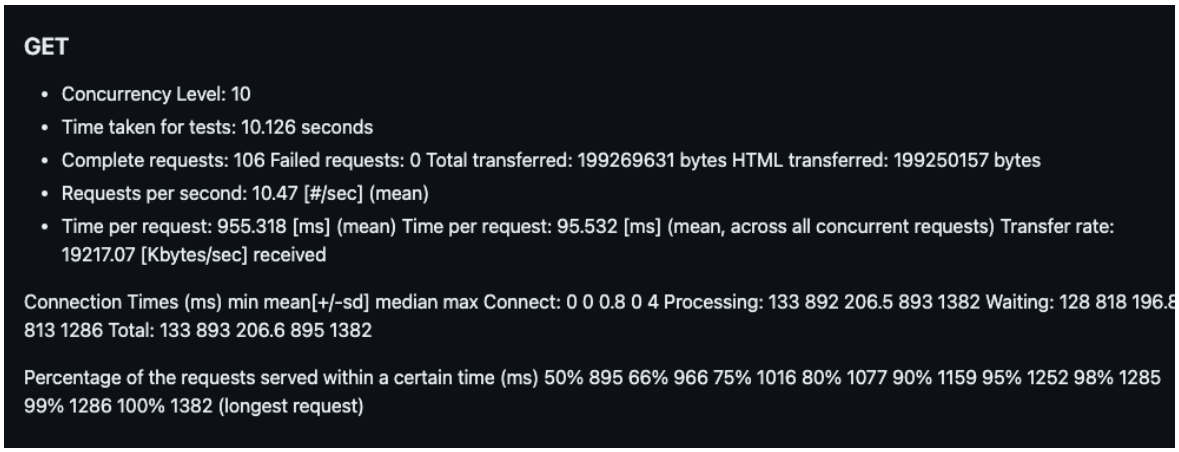
- Μέσο όρο 956.09 αιτήσεις ανά δευτερόλεπτο.
- Ο μέσος χρόνος ανά αίτηση ήταν 10.459 ms
- Οι χρόνοι απόκρισης για την πλειονότητα των αιτήσεων ήταν πολύ χαμηλοί, με το 95% των αιτήσεων να ολοκληρώνονται εντός 25 ms.

➔ PUT

- Εκτελέστηκαν 9114 αιτήσεις
- Μέσο όρο 911.23 αιτήσεις ανά δευτερόλεπτο.
- Ο μέσος χρόνος ανά αίτηση ήταν 10.974 ms
- Οι χρόνοι απόκρισης για την πλειονότητα των αιτήσεων ήταν επίσης χαμηλοί, με το 95% των αιτήσεων να ολοκληρώνονται εντός 30 ms.

Οι δοκιμές POST και PUT επίσης εμφάνισαν καλή απόδοση, με πολύ χαμηλούς χρόνους απόκρισης και υψηλό ρυθμό αιτημάτων ανά δευτερόλεπτο. Οι δοκιμές POST και PUT εμφάνισαν σχετικά χαμηλές τιμές απόκρισης για το ποσοστό των αιτήσεων που ολοκληρώθηκαν εντός των 25 ms. Συνολικά, μπορούμε να πούμε ότι οι δοκιμές έδειξαν καλή απόδοση του συστήματος, με υψηλό ρυθμό αιτημάτων και χαμηλούς χρόνους απόκρισης. Στις παρακάτω εικόνες μπορούμε να δούμε τα αποτελέσματα του benchmark για GET, POST, PUT όπως εξήχθησαν από το ab tool.

Εικόνα 10 ab benchmarking tool output GET



Εικόνα 11 ab benchmarking tool output POST

POST

Concurrency Level: 10 Time taken for tests: 10.000 seconds

- Complete requests: 9561 Failed requests: 0 Total transferred: 2744185 bytes Total body sent: 2431034 HTML transferred: 1042149 bytes
- Requests per second: 956.09 [#/sec] (mean)
- Time per request: 10.459 [ms] (mean) Time per request: 1.046 [ms] (mean, across all concurrent requests) Transfer rate: 267.98 [Kbytes/sec] received 237.40 kb/s sent 505.39 kb/s total

Connection Times (ms) min mean[+/-sd] median max Connect: 0 0 0.8 0 66 Processing: 1 10 8.3 8 110 Waiting: 1 10 8.3 7 110 Total: 1 10 8.3 8 110

Percentage of the requests served within a certain time (ms) 50% 8 66% 10 75% 11 80% 13 90% 20 95% 25 98% 37 99% 45 100% 110 (longest request)

Εικόνα 12 ab benchmarking tool output PUT

PUT

Concurrency Level: 10 Time taken for tests: 10.002 seconds

- Complete requests: 9114 Failed requests: 0 Total transferred: 2333952 bytes Total body sent: 2088938 HTML transferred: 720243 bytes
- Requests per second: 911.23 [#/sec] (mean)
- Time per request: 10.974 [ms] (mean) Time per request: 1.097 [ms] (mean, across all concurrent requests) Transfer rate: 227.88 [Kbytes/sec] received 203.96 kb/s sent 431.84 kb/s total

Connection Times (ms) min mean[+/-sd] median max Connect: 0 0 1.4 0 66 Processing: 3 11 9.2 8 100 Waiting: 2 10 9.0 7 99 Total: 3 11 9.3 8 100

Percentage of the requests served within a certain time (ms) 50% 8 66% 9 75% 10 80% 11 90% 20 95% 30 98% 39 99% 54 100% 100 (longest request)

Γραφήματα αποδόσεων

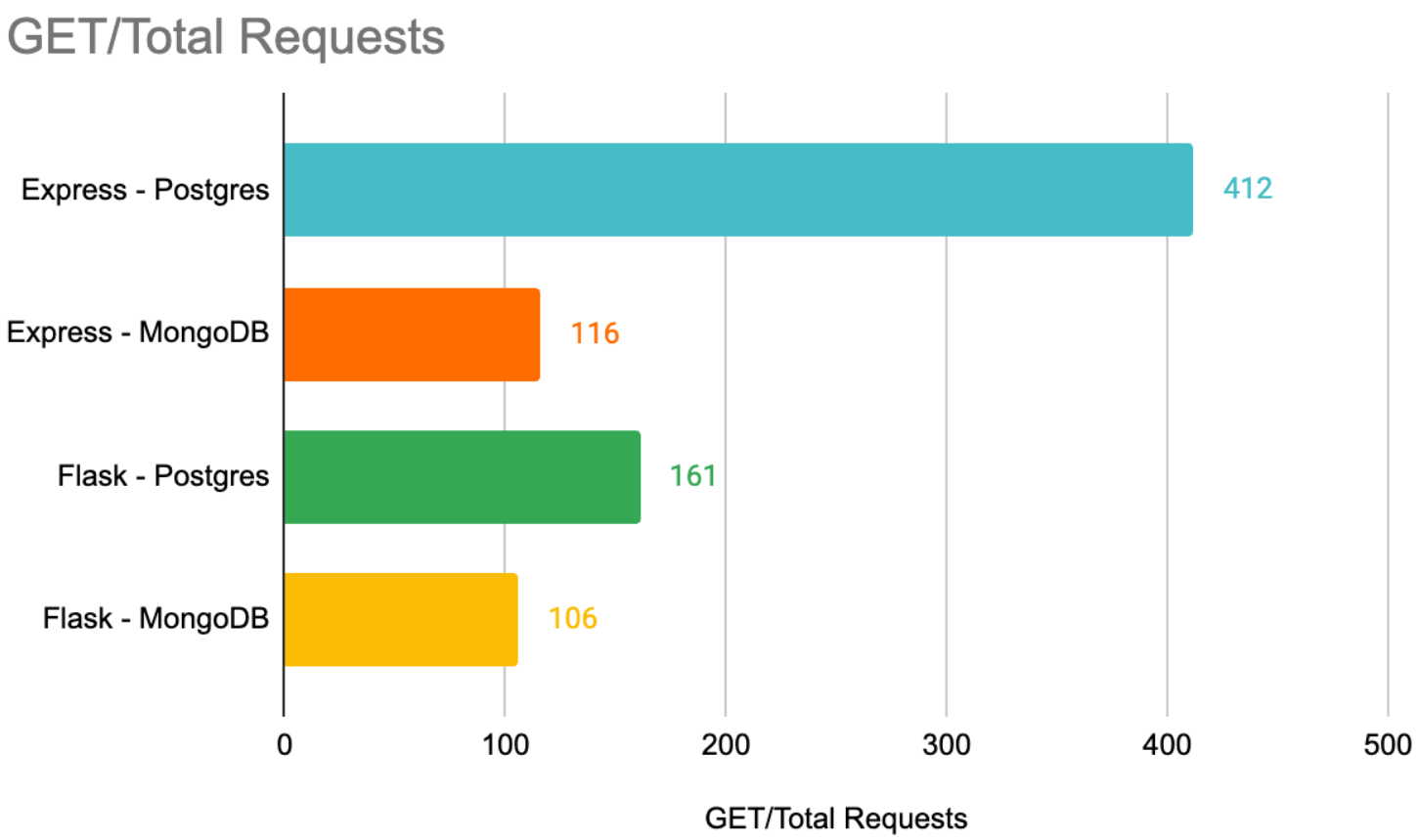
Παρακάτω ο πίνακας με τη συνολική συνοπτική παρουσίαση των benchmarks.

Πίνακας 1 Συνοπτική παρουσίαση των αποδόσεων

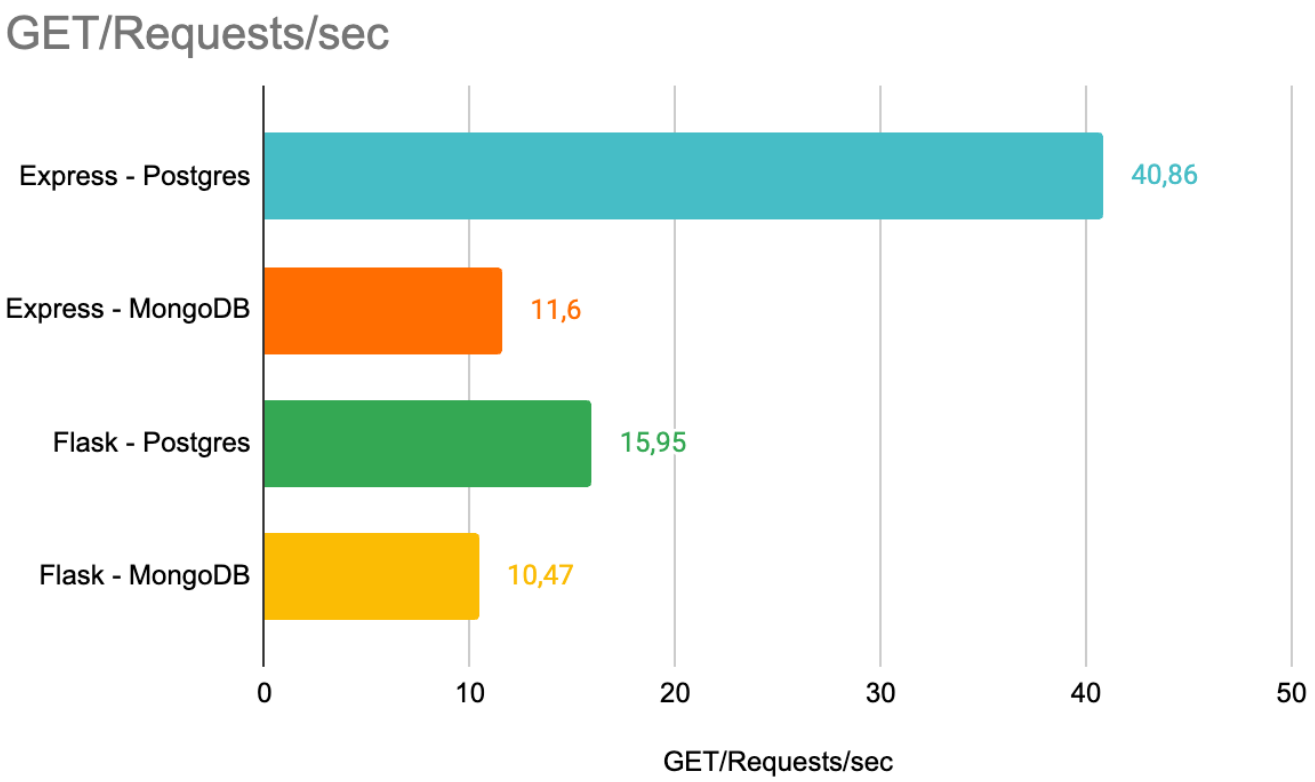
	GET			**POST**			**PUT**		
	Total Requests	**Requests/sec**	**Avg. Request Time**	**Total Requests**	**Requests/sec**	**Avg. Request Time**	**Total Requests**	**Requests/sec**	**Avg. Request Time**
Express - Postgres	412	40,86	245	16278	1207	8	11848	1184,79	8
Express - MongoDB	116	11,6	862	16271	1302,16	8	12009	1200,66	8
Flask - Postgres	161	15,95	627	6451	645,07	16	8030	802,93	12
Flask - MongoDB	106	10,47	955	9561	956,09	10	9114	911,23	11

Ακολούθως, τα αποτελέσματα σε διαγράμματα για καλύτερη κατανόηση.

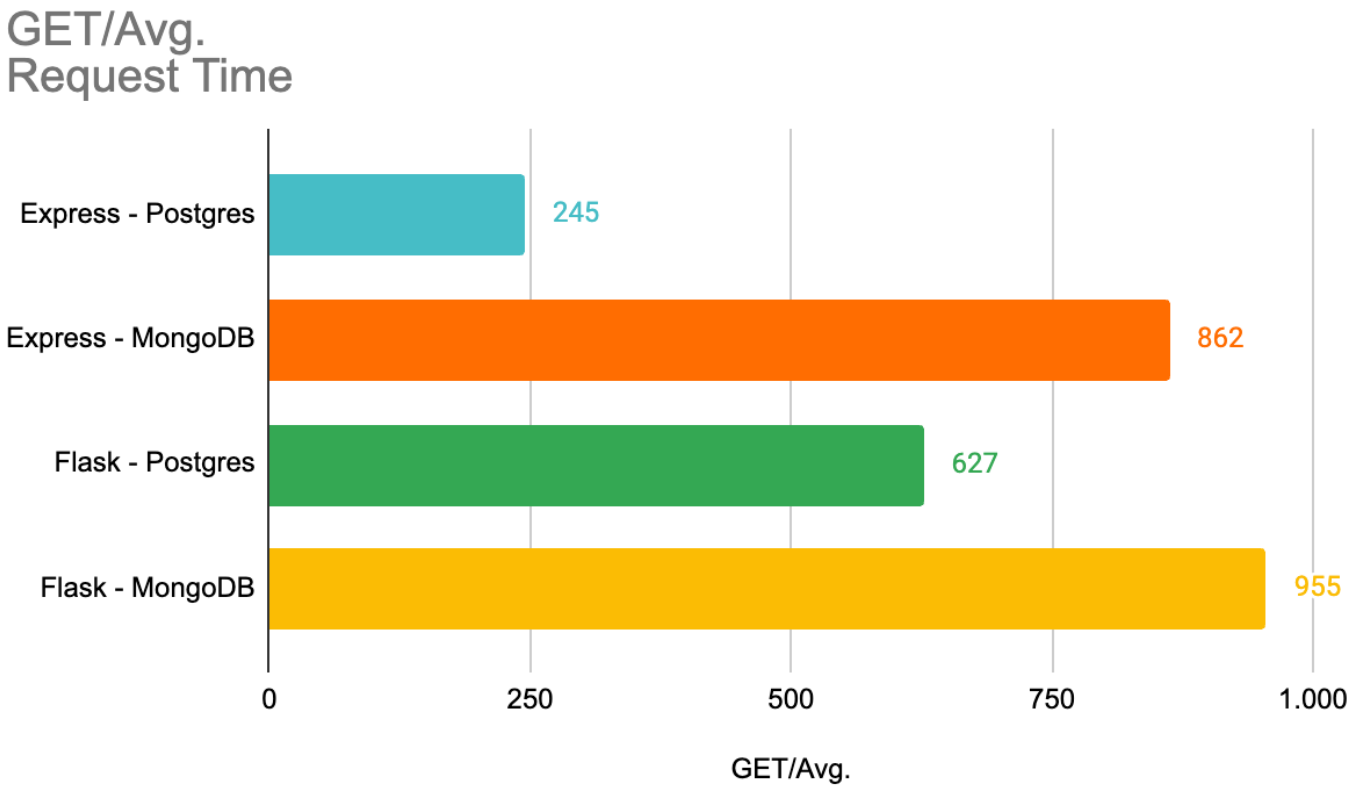
Εικόνα 13 GET total requests



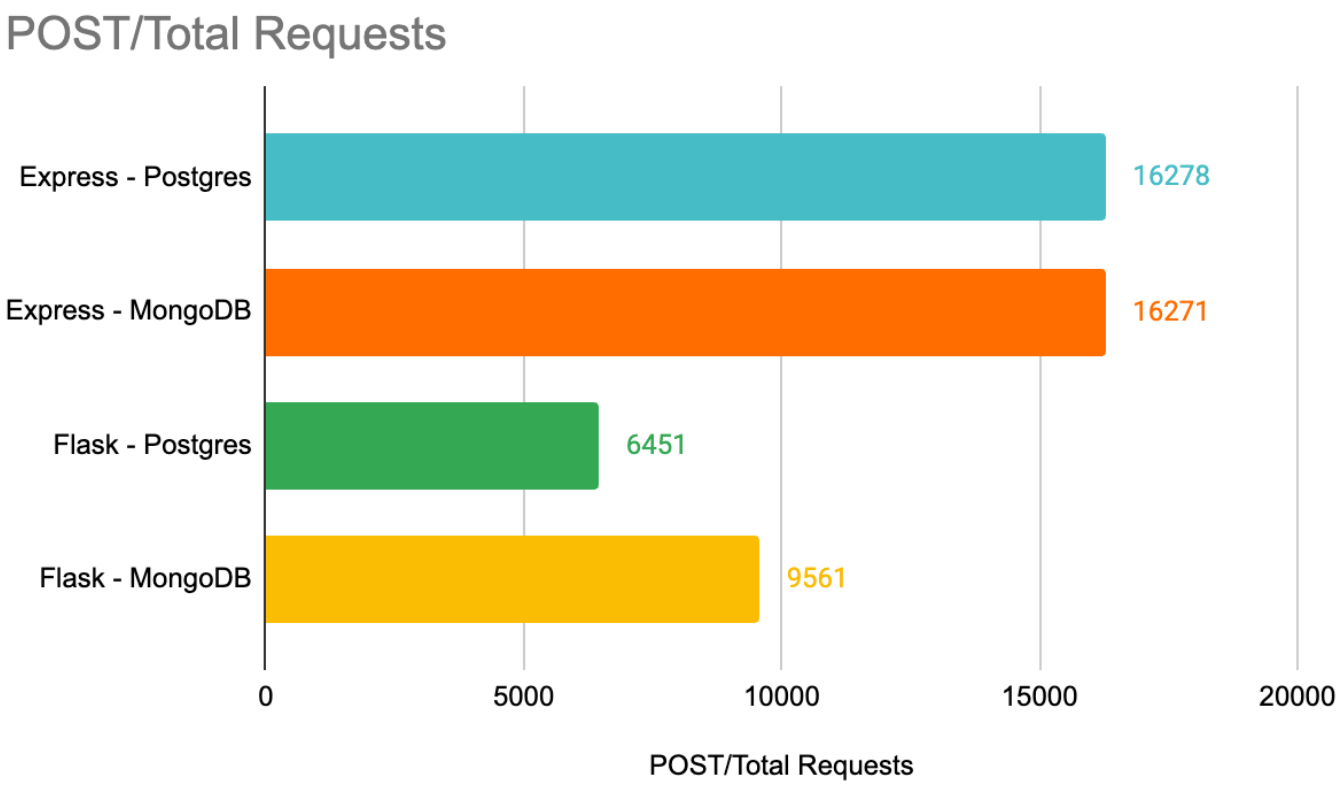
Εικόνα 14 GET requests/sec



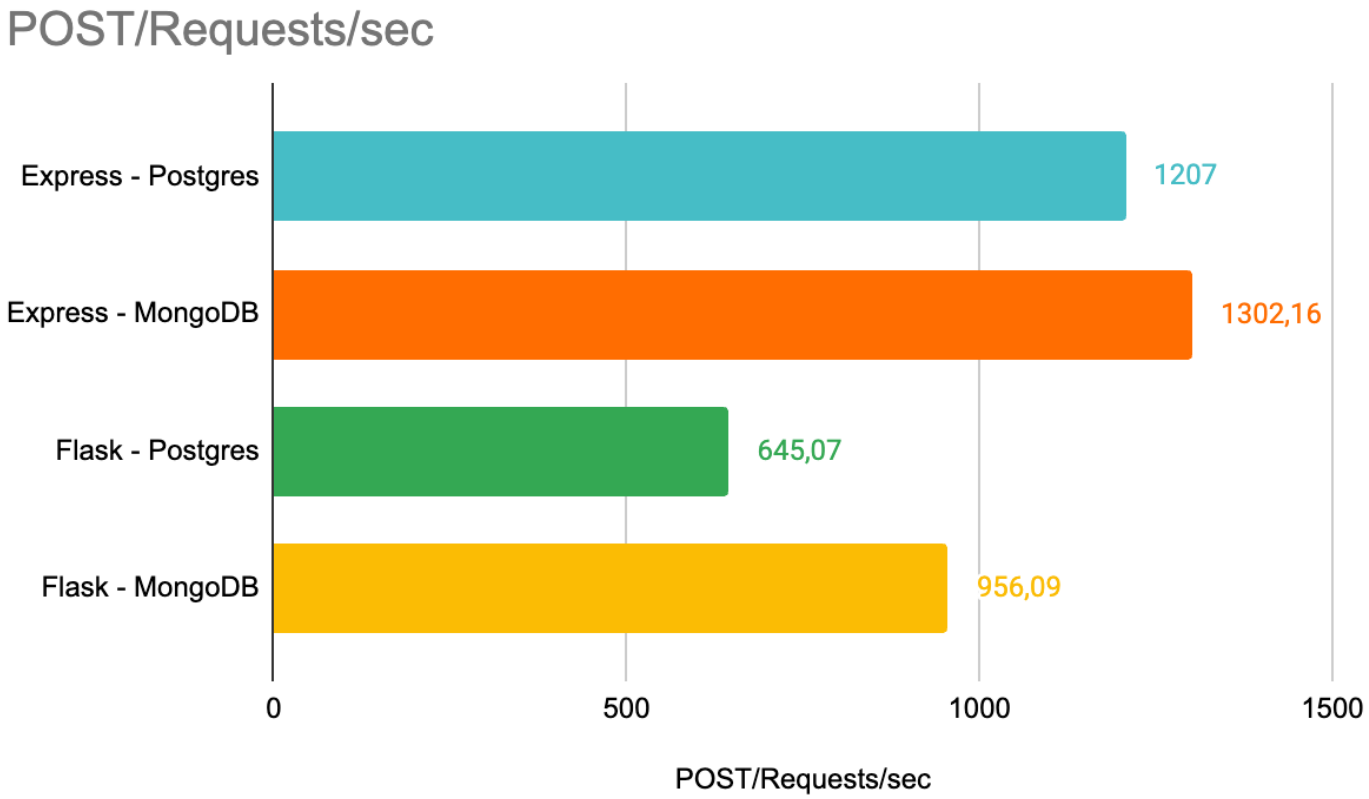
Εικόνα 15 GET Avg. Request time



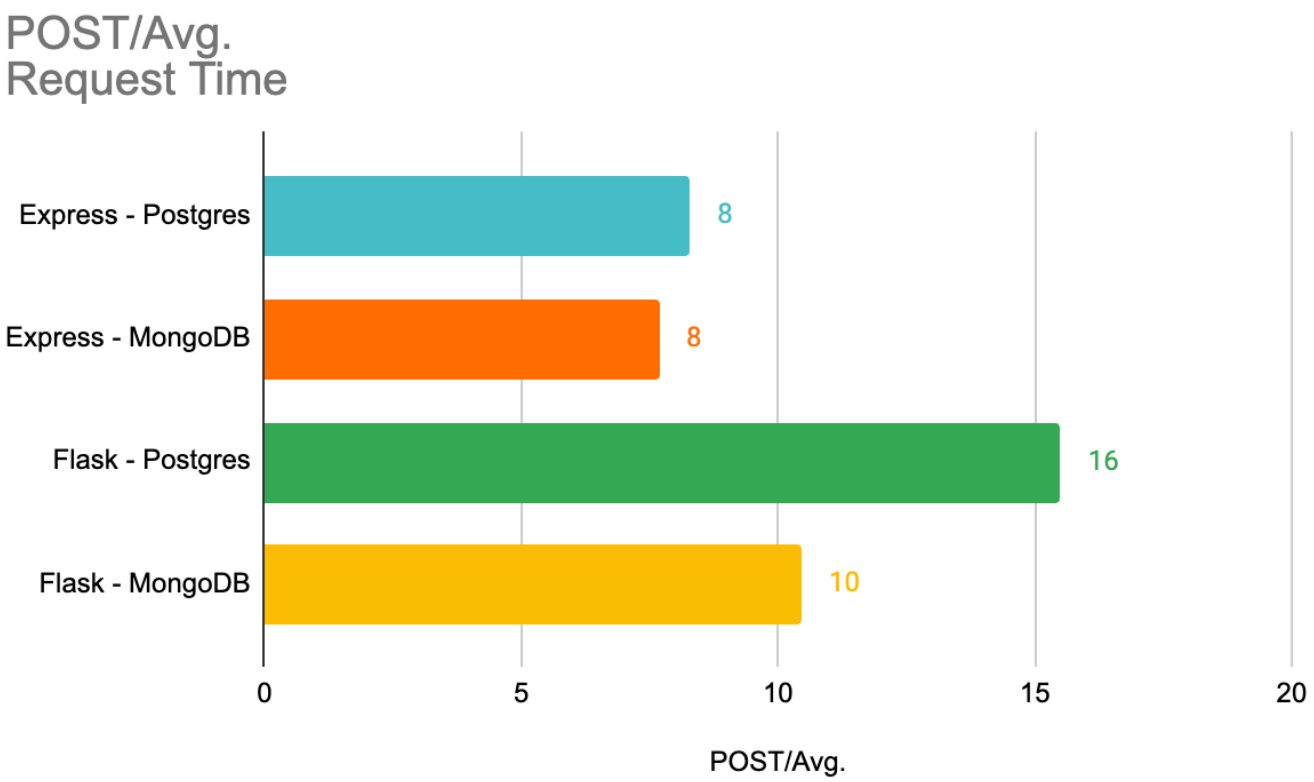
Εικόνα 16 POST total requests



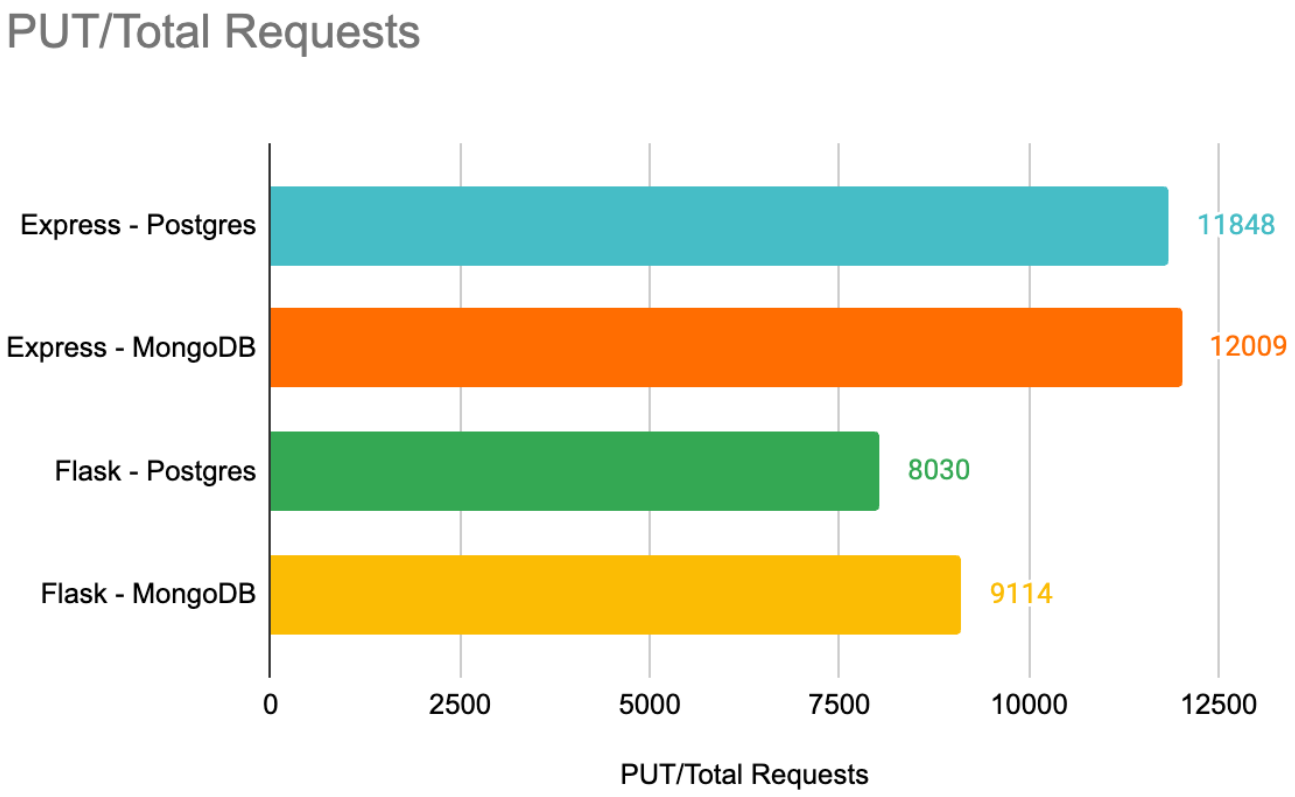
Εικόνα 17 POST requests/sec



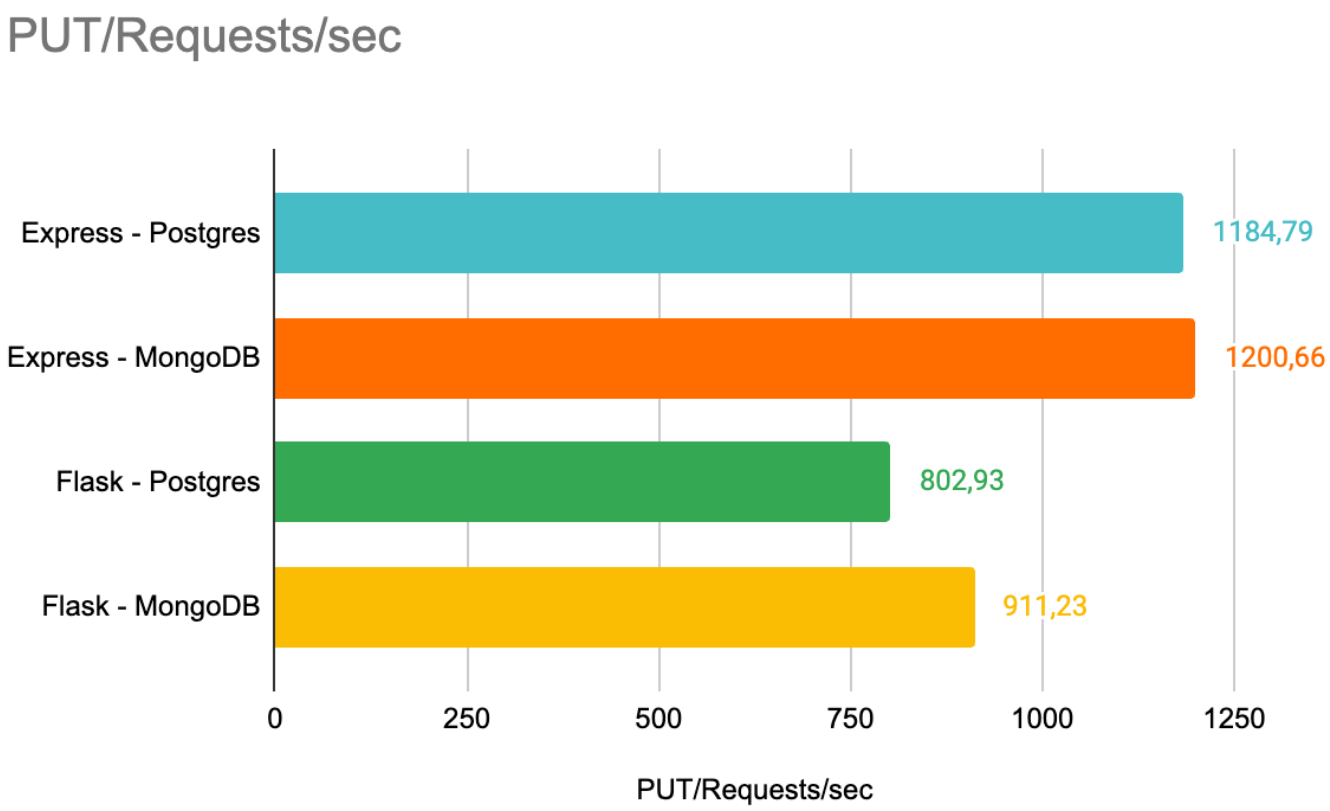
Εικόνα 18 POST Avg. request time



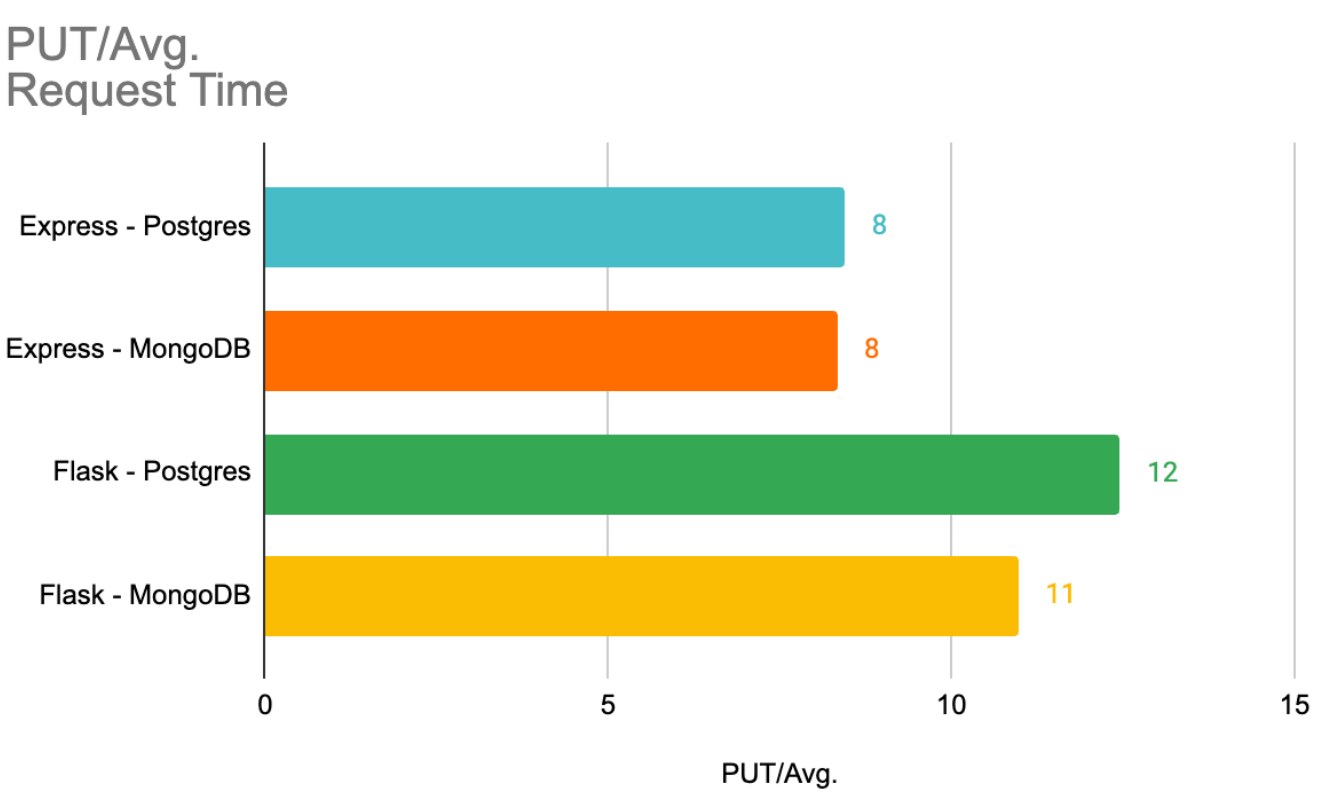
Εικόνα 19 PUT total requests



Εικόνα 20 PUT requests/sec



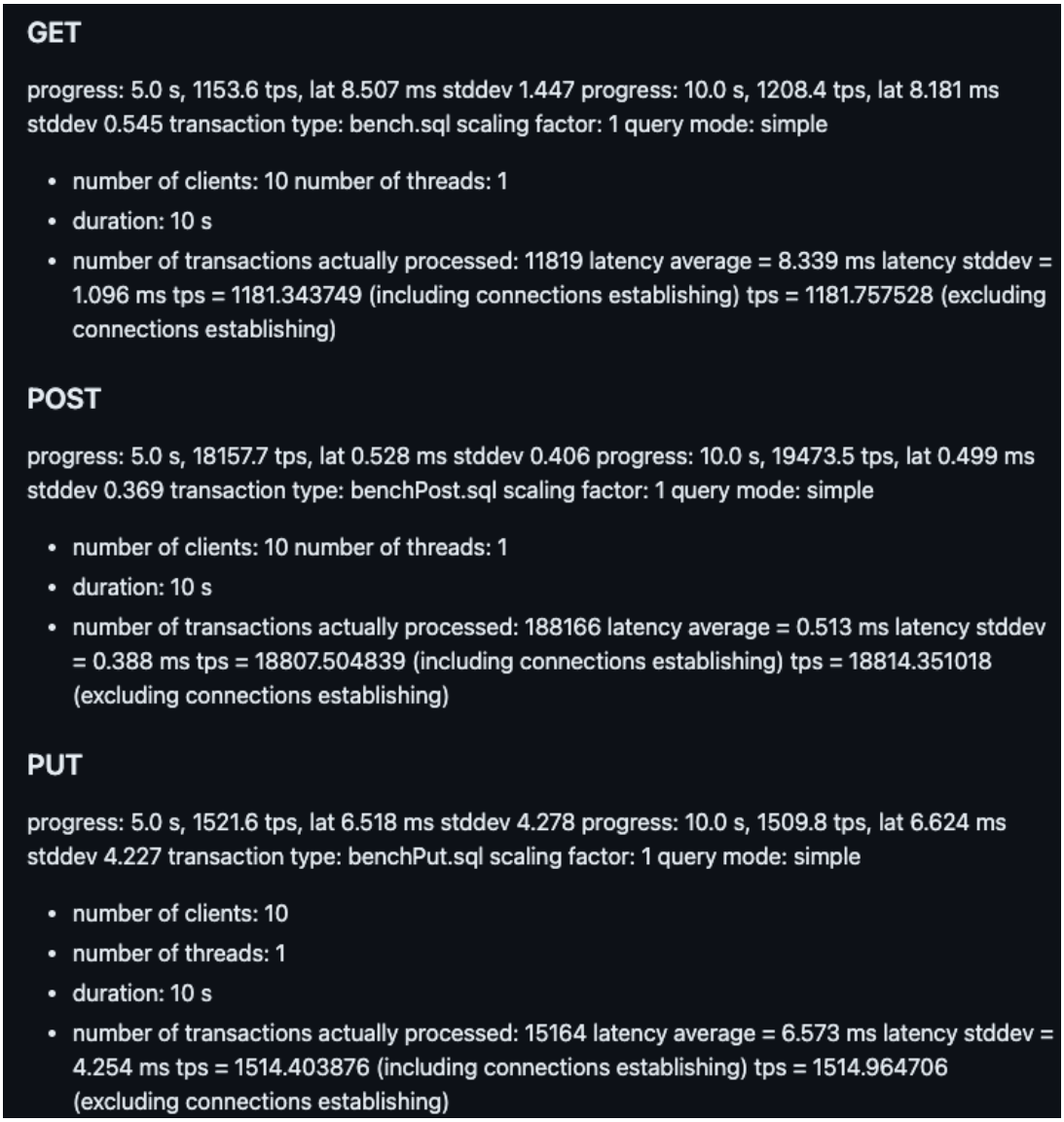
Εικόνα 21 PUT Avg. request time



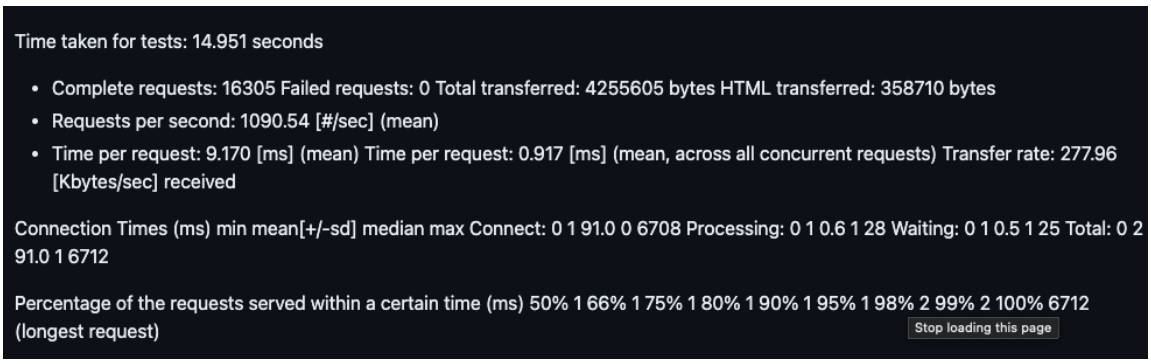
Σχολιασμός

Στην προσπάθεια να γίνει ανάλυση και τεκμηρίωση των διακυμάνσεων πραγματοποιήθηκαν δοκιμές μόνο http στους servers, benchmarking απευθείας στην postgres, και benchmarking ολων των συστημάτων με χρονομέτρηση των handlers μεσα στα routes, ολόκληρου του αιτήματος και της επικοινωνίας με τη βάση ξεχωριστά. Τα τεστ onlyHttp στα endpoints ανεξαρτήτως framework έφεραν παρόμοια αποτελέσματα οπότε οι διακυμάνσεις δεν οφείλονται σε αυτό τον παράγοντα. Κανοντας benchmark μόνο την postgres με χρήση του εργαλείου pgBench φαίνεται ότι υπάρχει μεγάλη διαφορά ανάμεσα σε GET και POST PUT. Για τη βέλτιστη ανάλυση και συνέπεια ανάμεσα στα συστήματα όλοι οι handlers των endpoints πραγματοποιούν “καθαρά” queries στη βάση χωρις περαιτέρω κώδικα. (εκτος μιας περίπτωσης στο flask που θα δούμε παρακάτω). Ακολουθούν οι εικόνες με τα αποτελέσματα των benchmarks μόνο στην postgres και μονο http στους servers.

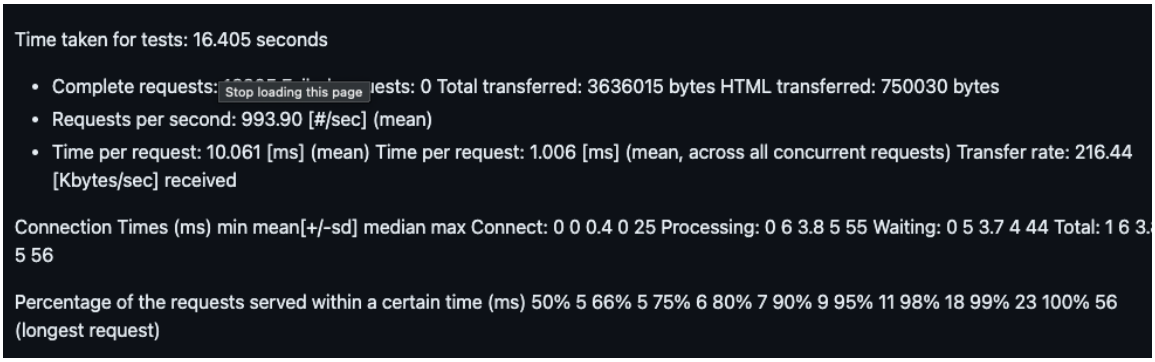
Εικόνα 22 Αποτελέσματα pgbench απευθείας στην Postgres.



Εικόνα 23 Αποτελέσματα onlyHttp benchmark στον express server



Εικόνα 24 Αποτελέσματα onlyHttp benchmark στο flask server



Η χρονομέτρηση και καταγραφή, για όλους τους servers και για καθε endpoint που διαπιστώθηκε μεγάλη διακύμανση σχετικά με το άλλο framework στην ίδια βάση δεδομένων, του χρόνου που χρειάζεται κάθε request για να επιστρέψει απάντηση και του χρόνου που χρειάζεται το καθαρο query απευθείας στη βάση με το σχετικό αντάπτορα του κάθε framework έδωσε κάποιες απαντήσεις.

Αφού έγινε εισαγωγή των αποτελεσμάτων σε excel και υπολογίστηκαν μέσοι όροι και διάμεσοι για τη σύγκριση και τη μελέτη των διαφορών και των καθυστερήσεων, έγινε φανερό ότι το flask υστερεί έναντι του express λόγω του χρόνου απόκρισης της επικοινωνίας του flask με τη βάση δεδομένων.

- Μία σημαντική διακύμανση στο GET ανάμεσα σε flask - postgres vs express - postgres οφείλεται στο ότι ο αντάπτορας του flask για την postgres επιστρέφει τα αποτελέσματα ως πίνακα πινάκων με τις τιμες κάθε στοιχείου ως τιμές στον κάθε πίνακα. Επομένως για τη σωστή λειτουργικότητα του app και τη συνέπεια μεταξύ των υπολοίπων συστημάτων, καθώς όλα τα υπόλοιπα επιστρέφουν τις εγγραφές ως objects με key values, για να δουλέψει σωστά το front end γινεται στον κώδικα μια επανάληψη και στέλνει επεξεργασμένα τα αποτελέσματα τα οποία λόγω του ότι είναι περίπου 8500 δημιουργείται αρκετή καθυστέρηση.
- Επίσης μεγάλες διαφορές ανάμεσα σε POST και PUT μεταξύ express vs flask εξηγούνται στη μεγάλη διαφορά χρόνου απόκρισης του adaptor που επικοινωνεί με την κάθε βάση και εκτελεί το query.

Στους ακόλουθους πίνακες απεικονίζονται οι διαφορές στους χρόνους εκτέλεσης των route handlers για την επικοινωνία και εκτέλεση του query στη βάση και ολόκληρου του request.

Πίνακας 2 Διαφορές χρόνων ανάμεσα σε Flask-Postgres/Express-Postgres για POST

POST	Database	Total request
Flask - Postgres		
Average	14,0910928	14,13161322
Median	10,948459	10,991583
Express - Postgres		
Average	3,88381378	3,957028259
Median	3,33147901	3,401750028

Πίνακας 3 Διαφορές χρόνων ανάμεσα σε Flask-MongoDb/Express-MongoDb για PUT

PUT	Database	Total request
Flask - MongoDb		
Average	5,348612844	5,38108304
Median	3,023333	3,051584
Express - MongoDb		
Average	3,229721704	3,28044332
Median	2,785124987	2,83302048

Πίνακας 4 Διαφορές χρόνων ανάμεσα σε Flask-Postgres/Express-Postgres για PUT

PUT	Database	Total request
Flask - Postgres		
Average	9,9263835	9,96139917
Median	9,253417	9,28375
Express - Postgres		
Average	7,83284767	7,9380983
Median	6,53575003	6,63475001

Συμπεράσματα

Η συνολική και συνοπτική παρουσίαση της σύγκρισης των τεχνολογιών απεικονίζεται στους παρακάτω πίνακες.

Πίνακας 5 Συνολικός πίνακας σύγκρισης των frameworks

Κριτήριο	**Flask**	**Express.js**
Γλώσσα προγραμματισμού	Python	JavaScript
Απόδοση	Υψηλή	Υψηλή
Κοινότητα και υποστήριξη	Μεγάλη	Μεγάλη
Αρχιτεκτονική	Microframework	Web framework
Απλότητα και ευκολία	Εύκολο στην εκμάθηση	Εύκολο στη χρήση
Ευελιξία και προσαρμοστικότητα	Υψηλή	Υψηλή
Επεκτασιμότητα	Υψηλή	Υψηλή
Δυνατότητες αυθεντικοποίησης	Με βιβλιοθήκες	Ενσωματωμένες και με βιβλιοθήκες
Οικοσύστημα εργαλείων	Ευρεία γκάμα εργαλείων	Ευρεία γκάμα εργαλείων
Δυνατότητες API	Ενσωματωμένες	Ενσωματωμένες
Βάσεις Δεδομένων	Ευέλικτο	Ευέλικτο

Πίνακας 6 Συνολικός πίνακας σύγκρισης των βάσεων δεδομένων

Χαρακτηριστικό	**PostgreSQL**	**MongoDB**
Τύπος	Σχεσιακή	NoSQL
Γλώσσα ερωτημάτων	SQL	MongoDB Query Language
Απόδοση	Υψηλή	Υψηλή
Αντοχή σε σφάλματα	Υψηλή	Υψηλή
Ευελιξία	Περιορισμένη	Υψηλή
Κλιμάκωση	Υψηλή	Υψηλή
Ευέλικτη δομή δεδομένων	Όχι	Ναι
Κοινότητα και υποστήριξη	Ενεργή και μεγάλη	Ενεργή και μεγάλη
Συνέπεια	Υψηλή	Υψηλή

➔ **GET**

- Το Express framework με τη βάση δεδομένων Postgres έχει την υψηλότερη συχνότητα αιτημάτων (40.86 αιτήματα/δευτερόλεπτο) ανάμεσα σε όλα τα σενάρια και τον μικρότερο μέσο χρόνο απάντησης (245 ms).
- Το Express framework με τη βάση δεδομένων MongoDB έχει μια μέτρια συχνότητα αιτημάτων (11.6 αιτήματα/δευτερόλεπτο) και μεγαλύτερο μέσο χρόνο απάντησης (862 ms). Αυτό υποδηλώνει ότι το σενάριο αυτό μπορεί να ανταποκριθεί σε χαμηλότερη κίνηση δεδομένων, αλλά ενδέχεται να αντιμετωπίσει προβλήματα μεγαλύτερης κίνησης.
- Οι servers με τη βάση δεδομένων Postgres εμφανίζει αποδοτική απόκριση σε όλα τα σενάρια, με μικρό μέσο χρόνο απάντησης και αξιοπρεπή συχνότητα αιτημάτων. Ειδικά στο σενάριο GET, όπου έχει το υψηλότερο αριθμό αιτημάτων (412), είναι η βάση δεδομένων που προτείνεται για υψηλές

απαιτήσεις κίνησης. Η βάση δεδομένων MongoDB έχει χαμηλότερη συχνότητα αιτημάτων και μεγαλύτερο μέσο χρόνο απάντησης σε σχέση με το Postgres.

- Το express υπερισχύει του flask

➔ **POST**

- το Express έχει πλεονεκτική θέση με αρκετά παραπάνω αιτήματα ανά δευτερόλεπτο περισσότερα από το Flask.
- Η MongoDB έχει λίγο υψηλότερη συχνότητα αιτημάτων από τη Postgres, και ο μέσος χρόνος απάντησης είναι χαμηλότερος από αυτόν της Postgres.

- Η MongoDB έχει μια μικρή πλεονεκτική θέση όσον αφορά τη συχνότητα αιτημάτων στο Flask
- Η Postgres έχει μια μικρή πλεονεκτική θέση όσον αφορά τη συχνότητα αιτημάτων στο Flask

➔ **PUT**

- Τόσο το Express όσο και το Flask εμφανίζουν αρκετά υψηλό αριθμό αιτημάτων PUT, με το Express να έχει ελαφρώς περισσότερα αιτήματα ανά δευτερόλεπτο. Ο μέσος χρόνος απάντησης για και τα δύο frameworks είναι παρόμοιος και κυμαίνεται γύρω στα 8-12 ms.
- Και το Express και το Flask μπορούν να χειριστούν αποδοτικά αιτήματα PUT, με μικρές διαφορές στον αριθμό των αιτημάτων ανά δευτερόλεπτο.
- Η mongoDb φαίνεται αποδοτικότερη σε σχέση με την postgres και το express αποδοτικότερο συγκριτικά με το flask.

Αναπτύσσοντας την εφαρμογή

Το έργο βασίζεται τόσο σε back-end όσο και σε front-end κώδικα. Υπάρχουν 4 backend φάκελοι/repos με τους αντίστοιχους servers και την επιλεγμένη βάση δεδομένων και 1 front end φάκελος/repo με την εφαρμογή Angular.

Στις επόμενες παραγράφους θα αναλυθούν λεπτομερώς τα repositories των εφαρμογών.

Express.js με Postgres

Ο φάκελος express with postgres περιέχει μια minimal και απλοϊκή προσέγγιση χωρίς πολλά abstractions, καθώς πρόκειται για μια αρκετά απλή demo εφαρμογή που έχει ως στόχο να παρουσιάσει τις λειτουργίες CRUD και να κάνει κάποιες δοκιμές συγκριτικής αξιολόγησης.

Περιέχει ένα αρχείο server.js που δημιουργεί έναν server Express.js για τη δημιουργία ενός REST API χρησιμοποιώντας Node.js και συνδέεται με μια βάση δεδομένων Postgres.
Εικόνα 25

Εικόνα 25 server.js

```
const express = require('express');

const bodyParser = require('body-parser');

const cors = require('cors');

const app = express();
app.use(cors());

const port = 3000;
const db = require('./db/queries');

app.use(bodyParser.json());
app.use(
  bodyParser.urlencoded({
    extended: true,
  })
);

app.get('/', (request, response) => {
  response.json({ info: 'Node.js, Express, and Postgres API' });
});

app.listen(port, () => {
  console.log(`App running on port ${port}.`);
});

app.get('/titles', db.getTitles);
app.get('/titles/:id', db.getTitleById);
app.post('/titles', db.createTitle);
app.put('/titles/:id', db.updateTitle);
app.delete('/titles/:id', db.deleteTitle);
```

Η μεταβλητή db εισάγει το queries.js που περιέχει τα sql queris της PostgreSQL για τις λειτουργίες CRUD.

Τέλος, οι μέθοδοι app.get(), app.post(), app.put() και app.delete() χρησιμοποιούνται για τον ορισμό των HTTP routes για τις λειτουργίες CRUD στο endpoint /titles.

Μέσα στο φάκελο db υπάρχει το αρχείο queries.js που κάνει σχεδόν αυτό που λέει ο τίτλος.

Αυτός ο κώδικας ορίζει και εξάγει ένα module που εξάγει πέντε συναρτήσεις, τις getTitles, getTitleById, createTitle, updateTitle και deleteTitle. Αυτές οι συναρτήσεις χειρίζονται αιτήματα HTTP που αντιστοιχούν σε λειτουργίες CRUD (Create, Read, Update, Delete) στη βάση δεδομένων PostgreSQL.

Η getTitles καλείται όταν γίνεται μια αίτηση GET στη διαδρομή '/titles'. Στέλνει ένα query SQL στη βάση δεδομένων που ανακτά όλες τις εγγραφές από τον πίνακα 'titles' και στέλνει τα αποτελέσματα στο response.

Η getTitleById καλείται όταν γίνεται αίτημα GET στο endpoint '/titles/:id'. Στέλνει ένα query SQL στη βάση δεδομένων που ανακτά μια συγκεκριμένη εγγραφή από τον πίνακα 'titles' με ένα δεδομένο id (tconst) και στέλνει τα αποτελέσματα στο response.

Η createTitle καλείται όταν γίνεται αίτημα POST στο endpoint '/titles'. Λαμβάνει δεδομένα από το body του αιτήματος και εισάγει μια νέα εγγραφή στον πίνακα 'titles' και αποστέλλει τη δημιουργηθείσα εγγραφή στο response.

Η updateTitle καλείται όταν γίνεται αίτημα PUT στο endpoint '/titles/:id'. Λαμβάνει δεδομένα από το body του αιτήματος και ενημερώνει μια υπάρχουσα εγγραφή στον πίνακα 'titles' με ένα δεδομένο id (tconst) και στέλνει ένα μήνυμα success στο response.

Η deleteTitle καλείται όταν γίνεται αίτημα DELETE στο endpoint '/titles/:id'. Διαγράφει μια υπάρχουσα εγγραφή από τον πίνακα 'titles' με δεδομένο id (tconst) και στέλνει μήνυμα success στο response. Εικόνα 26

Εικόνα 26 queries.js

```
const fs = require('fs');

let pool;
try {
  pool = new Pool({
    // user: 'vqlnioqfkvocxj',
    // host: 'ec2-54-229-217-195.eu-west-1.compute.amazonaws.com',
    // database: 'de9mikemijit5j',
    // password: '20de6fb3a0fd8e652444adbe7273c89429f28c11cc1376e46eb789cfb8474bef',
    // user: 'postgres',
    // host: 'localhost',
    // database: 'Movies',
    // password: '12345',
    // port: 5432,
    user: 'postgres',
    host: 'localhost',
    password: 'jijikos',
    port: 5433,
  });
} catch (error) {
  console.error(error);
}

// eslint-disable-next-line arrow-body-style
const getTitles = (request, response) => {
  pool.query('SELECT * FROM titles', (error, results) => {
    if (error) {
      throw error;
    }
    response.status(200).json(results);
  });
};

const getTitleById = (request, response) => {
  const tconst = request.params.id;
  pool.query(
    'SELECT * FROM titles WHERE tconst = $1 ORDER BY startyear ASC',
    [tconst],
    (error, results) => {
      if (error) {
        throw error;
      }
      response.status(200).json(results);
    }
  );
};

const createTitle = async (request, response) => {
  const startReq = performance.now();
  const { tconst, originalTitle, startYear, genres } = request.body;
  let endRequest;
  let endDb;
  try {
    const startDb = performance.now();
    await pool.query('BEGIN');
    const results = await pool.query(`INSERT INTO titles (tconst, originaltitle, startyear, genres) VALUES (${tconst}${Math.floor(Math.random() *
    await pool.query('COMMIT');
    endDb = performance.now();
    response.status(201).send({ res: results.rows });
    endRequest = performance.now();
    try {
      fs.appendFile('resultsPost.txt', `db: ${endDb - startDb} totRequest: ${endRequest - startReq}\n`, () => {});
    } catch (err) {
      console.error(err);
    }
  } catch (error) {
    response.status(500).send(error);
  }
};

const updateTitle = async (request, response) => {
  const startReq = performance.now();
  const tconst = request.params.id;
  const { originalTitle, startYear, genres } = request.body;
  let endRequest;
  let endDb;
  const startDb = performance.now();
  try {
    const res = await pool.query(
      `UPDATE titles SET originaltitle= '${originalTitle}', startyear= ${startYear}, genres= '${genres}' WHERE tconst = '${tconst}'`
    );
    endDb = performance.now();
    response.status(200).send({ res: `User modified with ID: ${tconst} ${JSON.stringify(res)}` });
    endRequest = performance.now();
    try {
      fs.appendFile('results.txt', `db: ${endDb - startDb} totRequest: ${endRequest - startReq}\n`, () => {});
    } catch (err) {
      console.error(err);
    }
  } catch (error) {
    console.error(error);
  }
};
```

Και στο parent directory υπάρχουν τα αντίστοιχα αρχεία των δοκιμών συγκριτικής αξιολόγησης: bench.json για τα GET και POST και benchPut.json για τα PUT. Το αρχείο benchMetrics.md έχει όλα τα αποτελέσματα σε μορφή md.

Express.js με MongoDB

Στο repo μας χωρίσαμε το project σε 3 φακέλους: controllers, routes και models και το αρχείο server.js που ζει στο parent directory και εκκινεί το server.

Στο server.js που φαίνεται στην Εικόνα 27, δημιουργείται μια σύνδεση στη βάση δεδομένων MongoDB με το όνομα movies και ορίζει routes για το χειρισμό λειτουργιών CRUD (Create, Read, Update, Delete) σε εγγραφές ταινιών. Η υλοποίηση αυτών των routes βρίσκεται σε ένα ξεχωριστό αρχείο με όνομα app.routes.js.

Εικόνα 27 server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const cors = require('cors');

mongoose.Promise = global.Promise;
mongoose
  .connect('mongodb://localhost:27017/movies', {
    useNewUrlParser: true,
  })
  .then(() => {
    console.log('Successfully connected to the database');
  })
  .catch((err) => {
    console.log('Could not connect to the database. Error...', err);
    process.exit();
  });

const app = express();

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

app.use(cors());

app.get('/', (req, res) => {
  res.json({ message: 'Server is running :D' });
});

let PORT = 3000;

require('./app/routes/app.routes.js')(app);
app.listen(PORT, () => {
  console.log(`Server is listening on port ${PORT}`);
});
```

Στην Εικόνα 28 γίνονται όλες οι απαραίτητες CRUD ενέργειες. Πιο αναλυτικά, η συνάρτηση findAll χειρίζεται ένα αίτημα GET για την ανάκτηση όλων των εγγράφων στη συλλογή App. Χρησιμοποιεί τη μέθοδο find() για να ανακτήσει όλα τα έγγραφα και τα επιστρέφει ως response σε μορφή JSON.

Η συνάρτηση create χειρίζεται ένα αίτημα POST για τη δημιουργία ενός νέου εγγράφου και το αποθηκεύει στη βάση δεδομένων χρησιμοποιώντας τη μέθοδο save() και επιστρέφει το νέο έγγραφο ως response σε μορφή JSON.

Η συνάρτηση findOne χειρίζεται ένα αίτημα GET για την ανάκτηση ενός μεμονωμένου εγγράφου στη συλλογή App. Χρησιμοποιεί τη μέθοδο find() με ένα φίλτρο του πεδίου tconst ίσο με την τιμή της παραμέτρου id στο URL της αίτησης. Εάν το έγγραφο βρεθεί, επιστρέφεται ως απάντηση σε μορφή JSON. Εάν το έγγραφο δεν βρεθεί, επιστρέφεται ένα μήνυμα σφάλματος 404.

Η συνάρτηση update χειρίζεται ένα αίτημα PUT για την ενημέρωση ενός υπάρχοντος εγγράφου στη συλλογή. Χρησιμοποιείται η μέθοδος updateOne() με ένα φίλτρο του πεδίου tconst ίσο με την τιμή της παραμέτρου id στο URL της αίτησης και ένα αντικείμενο με νέες τιμές για την ενημέρωση του εγγράφου. Εάν το έγγραφο ενημερωθεί, επιστρέφεται ως απάντηση σε μορφή JSON. Εάν το έγγραφο δεν βρεθεί, επιστρέφεται ένα μήνυμα σφάλματος 404.

Η συνάρτηση delete χειρίζεται ένα αίτημα DELETE για την αφαίρεση ενός υπάρχοντος εγγράφου από τη συλλογή App. Χρησιμοποιείται η μέθοδος findOneAndRemove() με ένα φίλτρο του πεδίου tconst ίσο με την τιμή της παραμέτρου id στη διεύθυνση URL του αιτήματος. Εάν το έγγραφο αφαιρεθεί, επιστρέφει ένα μήνυμα success ως response σε μορφή JSON. Εάν το έγγραφο δεν βρεθεί, επιστρέφει ένα μήνυμα σφάλματος 404. Στις Εικόνα 29 και Εικόνα 30 το μοντέλο των δεδομένων και τα routes αντίστοιχα.

Εικόνα 28 app.controller.js

```
const App = require('../model/app.model.js');
const fs = require('fs');

exports.findAll = (req, res) => {
  App.find()
    .then((data) => {
      res.send({ rows: data });
    })
    .catch((err) => {
      res.status(500).send({
        message: err.message || 'Some error occurred while retrieving titles.',
      });
    });
};

exports.create = (req, res) => {
  const body = new App(req.body);
  body
    .save()
    .then((data) => {
      res.send({ rows: data });
    })
    .catch((err) => {
      res.status(500).send({
        message: err.message || 'Some error occurred while creating the title.',
      });
    });
};

exports.findOne = (req, res) => {
  App.find({ tconst: req.params.id })
    .then((data) => {
      if (!data) {
        return res.status(404).send({
          message: 'Title not found with id ' + req.params.id,
        });
      }
      res.send({ rows: data });
    })
    .catch((err) => {
      if (err.kind === 'ObjectId') {
        return res.status(404).send({
          message: 'Title not found with _id ',
        });
      } else {
        return res.status(500).send({
          message: 'Error retrieving title with id ' + req.params.id,
        });
      }
    });
};

exports.update = async (req, res) => {
  try {
    const startDb = performance.now();
    const data = await App.updateOne(
      { tconst: req.params.id },
      {
        genres: req.body.genres,
        originalTitle: req.body.originalTitle,
        startYear: req.body.startYear,
      },
      { new: true }
      )
    const endDb = performance.now();
    res.send({ rows: data });
    const endRequest = performance.now();
    try {
      fs.appendFile('app/results.txt', `db: ${endDb-startDb} totRequest: ${endRequest-startDb}\n`, (err)=>{});
    } catch (err) {
      console.error(err);
    }
  } catch (error) {
    return res.status(500).send({
            message: 'Error updating title with id ' + req.params.id,
          });
  }
};
```

Εικόνα 29 Model.js

```
const mongoose = require('mongoose');

const AppSchema = mongoose.Schema(
  {
    tconst: String,
    titleType: String,
    primarytitle: String,
    originalTitle: String,
    isAdult: Number,
    startYear: Number,
    endyear: Number,
    runtimeminutes: Number,
    genres: String,
  },
  {
    collection: 'titles',
  }
);

module.exports = mongoose.model('App', AppSchema);
```

Εικόνα 30 routes.js

```
module.exports = (app) => {
  const App = require('../controllers/app.controller.js');

  app.get('/titles', App.findAll);

  app.get('/titles/:id', App.findOne);

  app.post('/titles', App.create);

  app.put('/titles/:id', App.update);

  app.delete('/titles/:id', App.delete);
};
```

Και στο parent diectory υπάρχουν τα αντίστοιχα αρχεία των δοκιμών συγκριτικής αξιολόγησης: bench.json για τα GET και POST και benchPut.json για τα PUT. Το αρχείο benchMetrics.md έχει όλα τα αποτελέσματα σε μορφή md.

Flask με Postgres

Το project χωρίζεται σε 2 αρχεία, τα python_server.py και python_db.py. Στο python_server.py όπως απεικονίζεται στην Εικόνα 32, χειριζόμαστε τις λειτουργίες CRUD στη βάση δεδομένων PostgreSQL για ένα σύνολο δεδομένων τίτλων ταινιών.

GET /titles: επιστρέφει όλους τους τίτλους ταινιών στη βάση δεδομένων.

GET /titles/<id>: επιστρέφει τον τίτλο ταινίας με το καθορισμένο id.

POST /titles: δημιουργεί έναν νέο τίτλο ταινίας με τα καθορισμένα δεδομένα.

PUT /titles/<id>: ενημερώνει τον τίτλο ταινίας με το καθορισμένο id με τα παρεχόμενα δεδομένα.

DELETE /titles/<id>: διαγράφει τον τίτλο ταινίας με το καθορισμένο id.

Ο κώδικας εισάγει τις απαραίτητες βιβλιοθήκες, όπως Flask (για τη δημιουργία του API), flask_restful (για τη δημιουργία endpoints RESTful API) και flask_cors (για τη δυνατότητα Cross-Origin Resource Sharing).

Για τα GET requests, ο κώδικας δημιουργεί μια σύνδεση με τη βάση δεδομένων, εκτελεί το κατάλληλο SQL query, ανακτά τα αποτελέσματα και αντιστοιχίζει τα αποτελέσματα σε μια λίστα λεξικών με τα ζεύγη κλειδιών-τιμών από τη λίστα κλειδιών.

Για τα POST και PUT requests, ο κώδικας εξάγει τα δεδομένα από το JSON του request, εκτελεί το κατάλληλο SQL query για τη δημιουργία ή την ενημέρωση ενός τίτλου ταινίας στη βάση δεδομένων και επιστρέφει ένα μήνυμα success.

Για DELETE requests, ο κώδικας εκτελεί το κατάλληλο ερώτημα SQL για τη διαγραφή ενός τίτλου ταινίας από τη βάση δεδομένων και επιστρέφει ένα μήνυμα success.

Στο python_db.py που απεικονίζεται στην Εικόνα 31 δημιουργείται η σύνδεση στη βάση δεδομένων PostgreSQL. Ορίζεται η συνάρτηση που ονομάζεται db() η οποία επιχειρεί να συνδεθεί στη βάση δεδομένων PostgreSQL χρησιμοποιώντας τη μέθοδο psycopg2.connect(), περνώντας τις απαραίτητες πληροφορίες σύνδεσης όπως το όνομα χρήστη, τον κωδικό πρόσβασης, τη θύρα και το όνομα της βάσης δεδομένων.

Εικόνα 31 python_db.py

```python
import psycopg2
from psycopg2 import Error


def db():
    try:
        connection = psycopg2.connect(user="postgres", password="12345",
                                      host="127.0.0.1", port="5432", database="Movies")

        cursor = connection.cursor()

        cursor.execute("SELECT version();")
        print("Connected to PostgreSQL")
    except (Exception, Error) as error:
        print("Error while connecting to PostgreSQL", error)

    finally:
        if (connection):
            cursor.close()
            connection.close()


if __name__ == '__main__':
    db()
```

Εικόνα 32 python_server.py

```python
@app.route('/titles', methods=['POST'])
def create_title():
    try:
        start = timer()
        data = getJson()
        startDb = timer()
        cursor = connection.cursor()
        cursor.execute(
            f"INSERT INTO titles (tconst, originaltitle, startyear, genres) VALUES ('{data.get('tconst')}{random.randint(1,1000000
        response = cursor.fetchall()
        connection.commit()
        endDb = timer()

        key_list = ["tconst", "titletype", "originaltitle", "primarytitle",
                    "isadult", "startyear", "endyear", "runtimeminutes", "genres"]

        res = []

        res.append({
            key_list[0]: response[0][0],
            key_list[1]: response[0][1],
            key_list[2]: response[0][2],
            key_list[3]: response[0][3],
            key_list[4]: response[0][4],
            key_list[5]: response[0][5],
            key_list[6]: response[0][6],
            key_list[7]: response[0][7],
            key_list[8]: response[0][8],
        })
        end = timer()
        f = open("./resultsPost.txt", "a")
        f.write(f"db: {(endDb - startDb)*1000} totRequest: {(end-start)*1000}\n")
        return {'rows': res}
    except psycopg2.Error as e:
            print(f"Error executing SQL query: {e}")

@app.route('/titles/<id>', methods=['PUT'])
def update_title(id):
    try:
        start = timer()
        data = getJson()
        startDb = timer()
        dbHandlingPut(id, data)
        end = timer()
        f = open("./results.txt", "a")
        f.write(f"db: {(end - startDb)*1000} totRequest: {(end-start)*1000}\n")
        return {'message': f"Updated title with id: {id}"}
    except psycopg2.Error as e:
            print(f"Error executing SQL query: {e}")

def dbHandlingPut(id, data):
    cursor = connection.cursor()
    cursor.execute(
            f"UPDATE titles SET originaltitle= '{data.get('originalTitle')}', startyear= {data.get('startYear')}, genres= '{data.g

def getJson():
    data: dict = request.get_json()
    return data


def get_db_connection():
    connection = psycopg2.connect(
        # user="vqlnioqfkvocxj",
        # password="20de6fb3a0fd8e652444adbe7273c89429f28c11cc1376e46eb789cfb8474bef",
        # host="ec2-54-229-217-195.eu-west-1.compute.amazonaws.com",
        # port="5432",
        # database="de9mikemijit5j"
        user="postgres",
        password="jijikos",
        host="localhost",
        port="5433",
        database="postgres"
    )
    return connection

if __name__ == '__main__':
    connection = get_db_connection()
    app.run(host="0.0.0.0", port=3000, debug=True, threaded=True)
```

Flask με MongoDB

Το project είναι ένα μόνο αρχείο το python_server.py χάριν στη minimal φύση των 2 τεχνολογιών. Μέσα στο python_server.py Εικόνα 33 η βιβλιοθήκη PyMongo χρησιμοποιείται για τη σύνδεση στη βάση δεδομένων MongoDB χρησιμοποιώντας ένα URI. Το λεξικό app.config ενημερώνεται με το MONGO_URI για τη σύνδεση στη βάση δεδομένων και δημιουργείται το PyMongo instance.

Το πρώτο endpoint '/titles' είναι ένα αίτημα GET που επιστρέφει όλους τους τίτλους ταινιών από τη συλλογή MongoDB. Οι τίτλοι ανακτώνται χρησιμοποιώντας τη μέθοδο find(). Στη συνέχεια, το αποτέλεσμα μεταδίδεται σε JSON μορφή χρησιμοποιώντας τη μέθοδο json_util.default από τη βιβλιοθήκη bson και επιστρέφεται ως αντικείμενο JSON.

Το δεύτερο endpoint '/titles/<id>' είναι ένα αίτημα GET που ανακτά έναν τίτλο ταινίας με το id του. Το id μεταβιβάζεται ως παράμετρος στη διεύθυνση URL και η μέθοδος find() χρησιμοποιείται για την ανάκτηση του τίτλου με το αντίστοιχο id.

Το τρίτο endpoint '/titles' είναι ένα αίτημα POST που εισάγει έναν νέο τίτλο ταινίας στη συλλογή MongoDB. Τα δεδομένα JSON εξάγονται χρησιμοποιώντας τη μέθοδο request.get_json() και στη συνέχεια εισάγονται στη συλλογή 'titles' χρησιμοποιώντας τη μέθοδο insert_one().

Το τέταρτο endpoint '/titles/<id>' είναι ένα αίτημα PUT που ενημερώνει έναν τίτλο ταινίας με το καθορισμένο id. Τα δεδομένα JSON αιτήματος εξάγονται χρησιμοποιώντας request.get_json() και η ενημέρωση εκτελείται χρησιμοποιώντας τη μέθοδο update_one().

Το πέμπτο endpoint '/titles/<id>' είναι ένα αίτημα DELETE που διαγράφει έναν τίτλο ταινίας με το καθορισμένο id. Ο τίτλος διαγράφεται χρησιμοποιώντας τη μέθοδο delete_one().

Και στο parent diectory υπάρχουν τα αντίστοιχα αρχεία των δοκιμών συγκριτικής αξιολόγησης: bench.json για τα GET και POST και benchPut.json για τα PUT. Το αρχείο benchMetrics.md έχει όλα τα αποτελέσματα σε μορφή md.

Εικόνα 33 python_server.py

```python
from bson import json_util
from flask import Flask, request
from flask_restful import Api
from flask_cors import CORS
from flask_pymongo import PyMongo
import json
from timeit import default_timer as timer


app = Flask(__name__)
api = Api(app)
CORS(app)


app.config["MONGO_URI"] = "mongodb://localhost:27017/movies"
mongo = PyMongo(app)


@app.route('/titles', methods=['GET'])
def get_titles():
    try:
        titles = [titles for titles in mongo.db.titles.find()]
        titlesStr = json.dumps(titles, default=json_util.default)
        return {'rows': json.loads(titlesStr)}
    except:
        return {'error'}



@app.route('/titles/<id>', methods=['GET'])
def get_titleById(id):
    title = [title for title in mongo.db.titles.find({'tconst': id})]
    titlesStr = json.dumps(title, default=json_util.default)
    return {'rows': json.loads(titlesStr)}


@app.route('/titles', methods=['POST'])
def create_title():
    body = getJson(request)
    mongo.db.titles.insert_one(
        {'tconst': f"{body.get('tconst')}", 'originalTitle': f"{body.get('originalTitle')}", 'startYear': f"{body.get('startYear')
    return {'tconst': f"{body.get('tconst')}", 'originaTitle': f"{body.get('originalTitle')}", 'startYear': f"{body.get('startYear

@app.route('/titles/<id>', methods=['PUT'])
def update_title(id):
    start = timer()
    body = getJson(request)
    startDb = timer()
    dbHandlingPut(id, body)
    endDb = timer()
    f = open("./results.txt", "a")
    f.write(f"db: {(endDb - startDb)*1000} totRequest: {(endDb-start)*1000}\n")
    return {'genres': body.get('genres'), 'startYear': body.get('startYear'), 'originalTitle': body.get('originalTitle')}

def dbHandlingPut(id, body):
    mongo.db.titles.update_one(
        {'tconst': id}, {"$set": {'genres': body.get('genres'), 'startYear': body.get('startYear'), 'originalTitle': body.get('ori

def getJson(request):
    body: dict = request.get_json()
    return body

@app.route('/titles/<id>', methods=['DELETE'])
def delete_titleId(id):
    mongo.db.titles.delete_one({'tconst': id})
    return {'res': f"Deleted title with id: {id}"}


if __name__ == '__main__':
    app.run(host="0.0.0.0", port=3000, debug=True, threaded=True)
```

Angular

Ο frontend κώδικας έχει μια αρκετά κοινή δομή ενός Angular project έχοντας τους φακέλους domain, home, auth και services για το χειρισμό δεδομένων και τη διεπαφή χρήστη. Το app-routing.module.ts που είναι υπεύθυνο για το routing, το data.function.ts για το χειρισμό των δεδομένων και το κύριο app.component.ts.

Ακολουθεί component based αρχιτεκτονική, όπου κάθε component αντιπροσωπεύει μια συγκεκριμένη λειτουργικότητα ή στοιχείο διεπαφής χρήστη. Το έργο χρησιμοποιεί επίσης το Observable μοτίβο για το χειρισμό ασύγχρονων λειτουργιων και ροών δεδομένων στο frontend. Έχει επίσης γίνει deploy στο Netlify ως static webapp καθώς επίσης υπάρχει και η δυνατότητα του build με docker, docker-compose και nginx σε μια προσπάθεια να διερευνηθούν όλες οι δυνατότητες και λύσεις για επεκτασιμότητα, ευκολία στη χρήση, κοινή χρήση με άλλους, που προσφέρουν αυτές οι τεχνολογίες. Στην Εικόνα 34 το βασικό component που διαχειρίζεται όλες τις CRUD λειτουργίες.

Εικόνα 34 home.component.ts CRUD operations

```
60    onSearchFilters() {
61      const req = {
62        id: this.searchForm.controls.movieId.value,
63      };
64
65      this.loading = true;
66
67      if (req.id) {
68        this.movieService.fetchMovieId(req).subscribe((res) => {
69          this.data = res.rows;
70          this.loading = false;
71        });
72        return;
73      }
74      this.movieService.fetchMovies().subscribe((res) => {
75        this.data = res.rows;
76        this.loading = false;
77      });
78    }
```

```
54    createMovie() {
55      const body = {
56        tconst: this.updateForm.controls.tconst.value,
57        originalTitle: this.updateForm.controls.originalTitle.value,
58        startYear: this.updateForm.controls.startYear.value,
59        genres: this.updateForm.controls.genres.value,
60      };
61      this.confirmationService.confirm({
62        message: 'Are you sure you want to create the movie?',
63        header: 'Confirm',
64        icon: 'pi pi-exclamation-triangle',
65        accept: () => {
66          if (!body.tconst || !body.originalTitle) {
67            this.updateForm.markAllAsTouched();
68            return;
69          }
70          this.selectedMovies = [];
71          this.movieService.createMovie(body).subscribe((res) => {
72            if (res) {
73              this.messageService.add({
74                severity: 'success',
75                summary: 'Successful',
76                detail: 'Movie Added',
77                life: 3000,
78              });
79              this.loading = true;
80              this.movieService.fetchMovies().subscribe((res) => {
81                this.data = res.rows;
82                this.loading = false;
83                console.log(res.rows);
84              });
```

```
  updateSelectedMovie(movie) {
    const body = {
      originalTitle: this.updateForm.controls.originalTitle.value,
      startYear: this.updateForm.controls.startYear.value,
      genres: this.updateForm.controls.genres.value,
    };
    this.confirmationService.confirm({
      message: 'Are you sure you want to update the selected movie?',
      header: 'Confirm',
      icon: 'pi pi-update',
      accept: () => {
        this.selectedMovies = [];
        this.movieService
          .updateMovieById(movie[0].tconst, body)
          .subscribe((res) => {
            if (res) {
              this.messageService.add({
                severity: 'success',
                summary: 'Successful',
                life: 3000,
              });
              this.loading = true;
              this.searchForm.reset();
              this.updateForm.reset();
              this.movieService.fetchMovies().subscribe((res) => {
                this.data = res.rows;
                this.loading = false;
                console.log(res.rows);
              });
            }
          });
      },
    });
  }

  deleteSelectedMovie(movie) {
    this.confirmationService.confirm({
      message:
        'Are you sure you want to delete ' + movie[0].originaltitle + '?',
      header: 'Confirm',
      icon: 'pi pi-exclamation-triangle',
      accept: () => {
        this.selectedMovies = [];
        this.movieService.deleteMovieById(movie[0].tconst).subscribe((res) => {
          if (res) {
            this.messageService.add({
              severity: 'success',
              summary: 'Successful',
              detail: 'Movie Deleted',
              life: 3000,
            });
            this.loading = true;
            this.searchForm.reset();
            this.updateForm.reset();
            this.movieService.fetchMovies().subscribe((res) => {
              this.data = res.rows;
              this.loading = false;
              console.log(res.rows);
            });
          }
        });
      },
    });
  }
```

Στην Εικόνα 35 απεικονίζεται το αρχείο με τα services που κάνουν τα http αιτήματα και στην Εικόνα 36 οι βοηθητικές συναρτήσεις.

Εικόνα 35 moviesService.ts CRUD operations services

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';
import { environment } from 'src/environments/environment';
import * as helpers from 'src/app/data.functions';
import { NewTitle, TitlesMapped } from '../domain/interfaces';

@Injectable({ providedIn: 'root' })
export class MovieService {
  url = environment.url;

  constructor(private http: HttpClient) {}

  fetchMovies(): Observable<TitlesMapped> {
    return this.http
      .get<any>(`${this.url}/titles`)
      .pipe(map((res) => helpers.mapTitles(res)));
  }
  fetchMovieId(req): Observable<TitlesMapped> {
    return this.http
      .get<any>(`${this.url}/titles/${req.id}`)
      .pipe(map((res) => helpers.mapTitles(res)));
  }
  deleteMovieById(req): Observable<any> {
    return this.http.delete<any>(`${this.url}/titles/${req}`);
  }
  updateMovieById(req, body): Observable<any> {
    return this.http
      .put<any>(`${this.url}/titles/${req}`, body)
      .pipe(map((res) => helpers.mapTitles(res)));
  }
  createMovie(body: NewTitle): Observable<any> {
    return this.http
      .post<any>(`${this.url}/titles`, body)
      .pipe(map((res) => helpers.mapTitles(res)));
  }
}
```

Εικόνα 36 data.functions.ts (Helper functions for mapping)

```ts
import { SongsMapped, TitlesMapped } from './domain/interfaces';

export function mapData(res: SongsMapped) {
  const newRes: SongsMapped = {
    data: res.data.map((item) => ({
      id: item.id,
      code: item.code,
      name: item.name,
      description: item.description,
      year: item.year,
      quantity: item.quantity,
      inventoryStatus: item.inventoryStatus,
      artist: item.artist,
      image: item.image,
      rating: item.rating,
      isLoved: item.isLoved,
    })),
  };
  return newRes;
}

export function mapTitles(res: TitlesMapped) {
  const newRes: TitlesMapped = {
    rows: res?.rows?.map((item) => ({
      tconst: item.tconst,
      titletype:
        item.titletype === (null || undefined)
          ? item.titleType
          : item.titletype,
      originaltitle:
        item.originaltitle === (null || undefined)
          ? item.originalTitle
          : item.originaltitle,
      primarytitle:
        item.primarytitle === (null || undefined)
          ? item.primarytitle
          : item.primarytitle,
      isadult:
        item.isadult === (null || undefined) ? item.isAdult : item.isadult,
      startyear:
        item.startyear === (null || undefined)
          ? item.startYear
          : item.startyear,
      endyear:
        item.endyear === (null || undefined) ? item.endYear : item.endyear,
      runtimeminutes: item.runtimeminutes,
      genres: item.genres,
    })),
  };
  return newRes;
}
```

Docker

Στην παρακάτω εικόνα φαίνονται οι εντολές που εκτελούνται μέσα στο Dockerfile για την εκκίνηση της εφαρμογής στο docker container.

Εικόνα 37 Dockerfile

```
# Stage 1
FROM node:14-alpine as builder

COPY package.json package-lock.json ./

RUN npm install --silent

RUN mkdir /ng-app && mv ./node_modules ./ng-app

WORKDIR /ng-app

COPY . .

# RUN npm install -g @angular/cli @angular-devkit/build-angular && npm install --silent

RUN npm run ng build -- --prod --output-path=dist --base-href /bo/

# Stage 2
FROM nginx:1.14.1-alpine
RUN apk add nano
COPY nginx/default.conf /etc/nginx/conf.d/
RUN rm -rf /usr/share/nginx/html/*
COPY --from=builder /ng-app/dist /usr/share/nginx/html/bo
EXPOSE 4000
CMD ["nginx", "-g","daemon off;"]
```

Nginx

Στην Εικόνα 38 το configuration για τον Nginx reverse proxy server

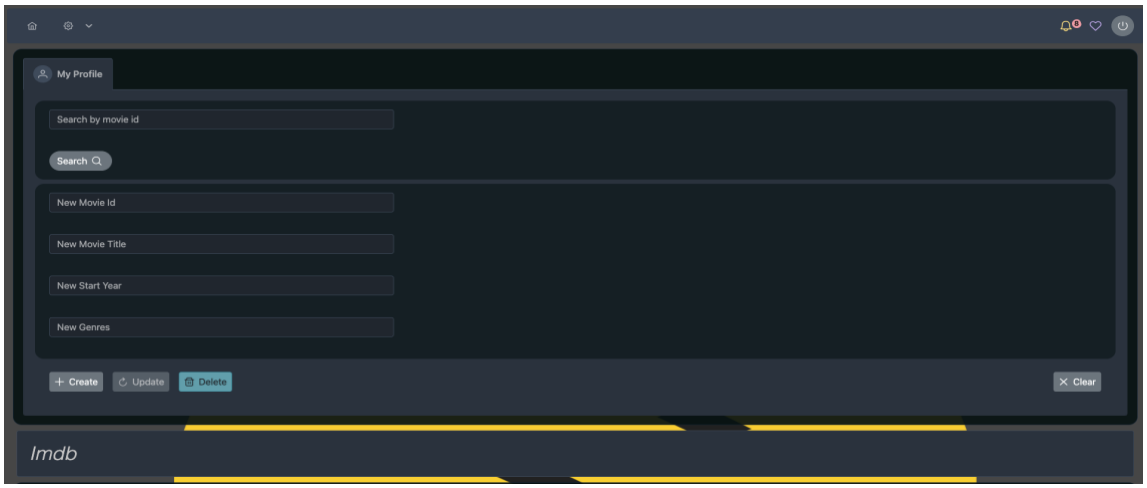
Εικόνα 38 Nginx conf

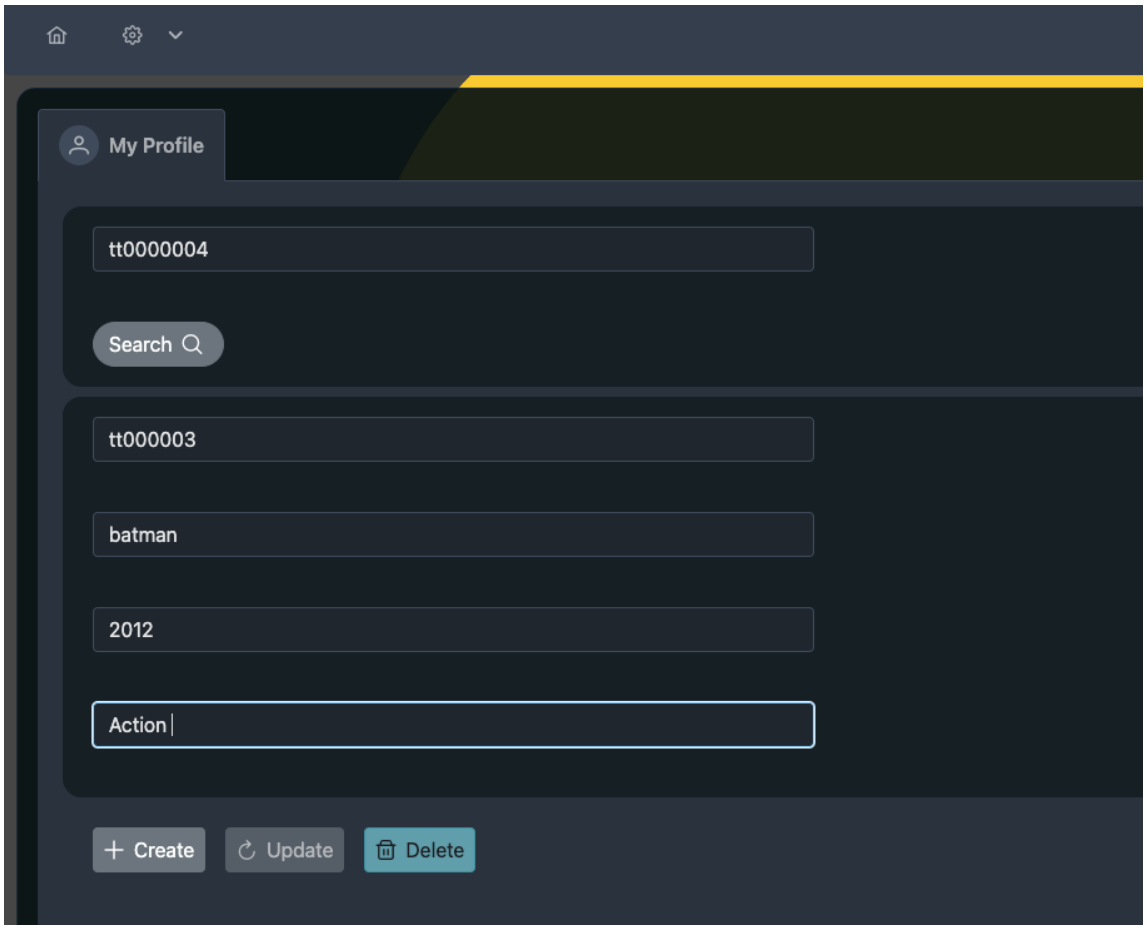
```
server {
    listen 4000;

#   server_name  localhost;

    root /usr/share/nginx/html;

    location /bo {
        try_files $uri $uri/ /bo/index.html =404;
    }

    location /staging {
        try_files $uri $uri/ /staging/index.html =404;
    }

}
```

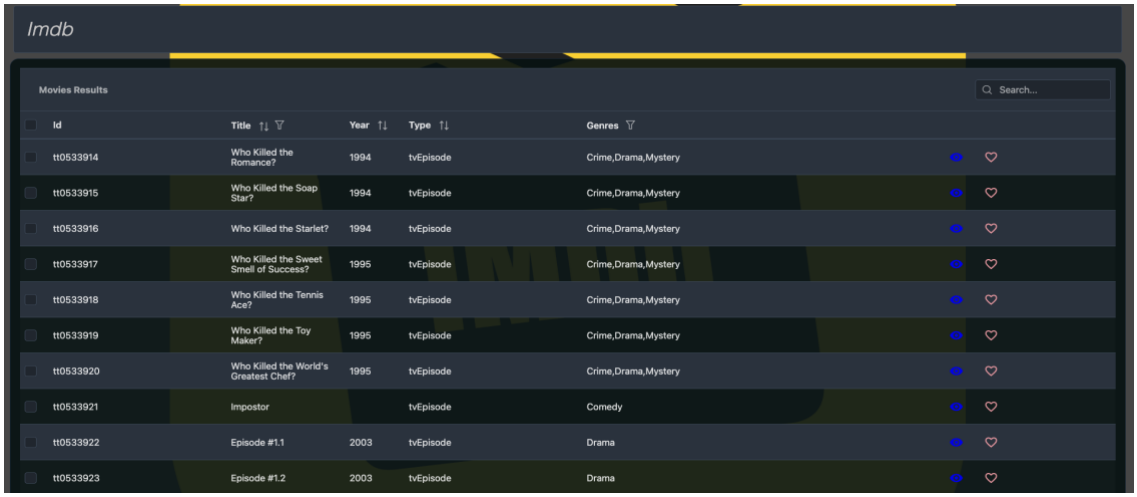
The application

Στις παρακάτω εικόνες απεικονίζεται το UI της εφαρμογής όπως δημιουργήθηκε στο front end.

Εικόνα 39 The application's UI






Imdb
Movies Results
Search...
Id
Title
Year
Type
Genres
tt0533914
Who Killed the Romance?
1994
tvEpisode
Crime,Drama,Mystery
tt0533915
Who Killed the Soap Star?
1994
tvEpisode
Crime,Drama,Mystery
tt0533916
Who Killed the Starlet?
1994
tvEpisode
Crime,Drama,Mystery
tt0533917
Who Killed the Sweet Smell of Success?
1995
tvEpisode
Crime,Drama,Mystery
tt0533918
Who Killed the Tennis Ace?
1995
tvEpisode
Crime,Drama,Mystery
tt0533919
Who Killed the Toy Maker?
1995
tvEpisode
Crime,Drama,Mystery
tt0533920
Who Killed the World's Greatest Chef?
1995
tvEpisode
Crime,Drama,Mystery
tt0533921
Impostor
tvEpisode
Comedy
tt0533922
Episode #1.1
2003
tvEpisode
Drama
tt0533923
Episode #1.2
2003
tvEpisode
Drama

Αναφορές

1. https://expressjs.com
2. https://flask.palletsprojects.com/en/2.3.x
3. https://www.postgresql.org
4. https://www.mongodb.com
5. https://httpd.apache.org/docs/2.4/programs/ab.html
6. https://www.nginx.com
7. https://docs.docker.com
8. https://github.com/expressjs/express
9. https://github.com/pallets/flask
10. https://github.com/postgres/postgres
11. https://github.com/mongodb/mongo
12. https://github.com/angular/angular
13. https://angular.io/
14. https://techtrim.tech/express-vs-flask/
15. https://medium.com/geekculture/should-i-use-mongodb-or-postgresql-ba2c1bb8b768
16. https://kinsta.com/blog/mongodb-vs-postgresql/
17. https://www.geeksforgeeks.org/difference-between-postgresql-and-mongodb/
18. Changpil Lee, "An Evaluation Model for Application Development Frameworks forWeb Applications", Master's Thesis, The Ohio State University, 2012
19. https://www.guru99.com/node-js-vs-python.html
20. https://www.python.org/
21. https://nodejs.org/en
22. https://www.geeksforgeeks.org/difference-between-postgresql-and-mongodb/
23. Dhruv Verma, "A Comparison of Web Framework Efficiency– Performance and network analysis of modern web frameworks", Bachelor's Thesis, Turku University of Applied Sciences, 2022
24. https://www.mongodb.com/compare/mongodb-postgresql
25. https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems
26. Max Jonsson & Eric Qvarnström, "A Performance Comparison on REST APIS In Express.js, Flask and ASP.NET CORE", Bachelor's Thesis, Mälardalen University, School Of Innovation Design And EngineeringVästerås Sweden, 2022
27. Ευγένιος Σωχόπουλος, "Κατανεμημένος Υπολογισμός Στην GO: Συγκριτική Ανάλυση Πλαισίων Εφαρμογών Ιστού", Τμήμα Εφαρμοσμένης Πληροφορικής, 2020