ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ, ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΟΠΤΙΚΟΠΟΙΗΣΗ ΔΕΔΟΜΕΝΩΝ ΒΙΟΜΗΧΑΝΙΑΣ ΚΙΝΗΜΑΤΟΓΡΑΦΟΥ ΜΕ ΧΡΗΣΗ ΑΡΙ ΔΗΜΟΣΙΩΝ ΑΝΟΙΚΤΩΝ ΔΕΔΟΜΕΝΩΝ

Πτυχιακή εργασία του

Νικόλαος Μαυρόπουλος (3783)

Επιβλέπων: Ν. Πεταλίδης

ΣΕΡΡΕΣ, ΔΕΚΕΜΒΡΙΟΣ 2020

Υπεύθυνη δήλωση

Υπεύθυνη Δήλωση: Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Μηχανικών Πληροφορικής, Υπολογιστών και Τηλεπικοινωνιών του Διεθνούς Πανεπιστημίου της Ελλάδος

Σύνοψη

Στην παρούσα πτυχιακή θα εξετάσουμε πως με την χρήση των Open Data μπορούν να δημιουργηθούν νέες ιδέες και τεχνολογίες με σκοπό της εξέλιξη της κοινωνίας και της τεχνολογίας. Σε αυτήν την εργασία παρουσιάζεται ένα εργαλείο που με βάση την χρήση των ανοιχτών δεδομένων (Open Data) παρουσιάζει συνδυαστικά και αναλυτικά δεδομένα, από 2 υπηρεσίες, της βιομηχανίας του κινηματογράφου με σκοπό την ευκολότερη επιτέλεση των εργασιών ατόμων που δουλεύουν στην βιομηχανία του κινηματογράφου αλλά και την ευκολότερη κατανόηση αυτών των δεδομένων. Τα δεδομένα παρουσιάζονται σε ένα φιλικό προς τον χρήση γραφικό περιβάλλον με τη χρήση γραφημάτων, χαρτών κ.α Η εφαρμογή χωρίζεται σε 2 μέρη, τον Client και τον Server και χρησιμοποιούνται δημοφιλείς τεχνολογίες για την επίτευξη των στόχων της πτυχιακής.

Περιεχόμενα

Κατάλογος πινάκων

Κατάλογος διαγραμμάτων

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους γονείς μου για την αμέτρητη υπομονή που έκαναν και τη βοήθεια που μου προσέφεραν τόσο υλική όσο και πνευματική καθ' όλη την διάρκειά της εκπόνησης της πτυχιακής μου, αλλά και καθ' όλη την διάρκεια των σπουδών μου. Επίσης θα ήθελα να ευχαριστήσω την ξαδέρφη μου Έλλη Κουριαννίδου για τη βοήθεια που μου προσέφερε με τις δικές της ιδέες, τις δικές τις διορθώσεις και γενικά το feedback για όλη την πτυχιακή, τον φίλο μου Μιχάλη Μιχαηλίδη για την βοήθεια που μου έδωσε στην αρχιτεκτονική της πτυχιακής και ιδιαίτερα του Server, και την φίλη μου Χρυσή Τσινού για το feedback που μου έδωσε, στο αισθητικό και λειτουργικό κομμάτι του Client. Επιπρόσθετα θα ήθελα να ευχαριστήσω τον θείο μου Παντελή Καπλάνογλου που είναι η αιτία που ασχολήθηκα με τον προγραμματισμό εξαρχής και ότι χρειάστηκα ήταν εκεί για να με βοηθήσει.

Τέλος ένα μεγάλο ευχαριστώ θα ήθελα να δώσω και στον καθηγητή μου, Νικόλαο Πεταλίδη, που με ώθησε να διευρύνω τις γνώσεις μου και συνετέλεσε στην προσωπική μου εξέλιξη σαν προγραμματιστή.

Ορισμοί

Parallel Programming Ο παράλληλος προγραμματισμός επιτρέπει την ταυτόχρονη εκτέλεση εντολών στον επεξεργαστή. Η χρήση του είναι καθοριστική, καθώς ορισμένες ενέργειες σε ένα πρόγραμμα ξοδεύουν παραπάνω χρόνο κάνοντας άλλες δουλειές κρατώντας τον επεξεργαστή σε αδράνεια. Εκτελώντας περισσότερες εντολές παράλληλα βοηθάει στην πιο αποδοτική χρήση του επεξεργαστή και βελτιώνει θεαματικά την απόδοση αυτών των ενεργειών.

Thread είναι ένα νήμα του επεξεργαστή που σου επιτρέπει να τρέχεις κώδικα μέσα του. Όταν τρέχουν διαφορετικά κομμάτια κώδικα σε διαφορετικά threads η εφαρμογή χαρακτηρίζεται Multi threadded, και ο κώδικας τρέχει παράλληλα.

Thread Safety Όταν ένας κώδικας τρέχει σε πολλά Threads (νήματα) ασύγχρονα, και όλα αυτά τα threads έχουν πρόσβαση στα ίδια δεδομένα θα υπάρξει ένα πρόβλημα του συγχρονισμού των δεδομένων. Για παράδειγμα, αν έχουμε 2 Threads, το Thread 1 και Thread 2, και υπάρχει στον κώδικα ένας έλεγχος σε μια μεταβλητή που είναι ίση με 0, κατά το Thread 1, ενώ στον ίδιο χρόνο το Thread 2 αλλάζει την μεταβλητή αυτήν, τότε το Thread 1 δε θα γνωρίζει για την αλλαγή και μπορεί να προκαλέσει βλάβη με αποτέλεσμα τη μη σωστή λειτουργία του κώδικα αυτού. Το Thread Safety είναι ”κανόνες” που χρησιμοποιούν τεχνικές για να αποφευχθεί αυτό το φαινόμενο.

Framework είναι ένα αφαιρετικό επίπεδο πάνω από ένα ήδη γραμμένο κώδικα το οποίο επιτρέπει στον προγραμματιστή να αλλάξει ρυθμίσεις και να γράψει περαιτέρω κώδικα που να συμπληρώνει τον ήδη υπάρχον κώδικα από το Framework. Η Διαφορά του από την βιβλιοθήκη (Library) είναι ότι η βιβλιοθήκη παρέχει συναρτήσεις και κώδικα για να τις καλέσει ο προγραμματιστής για να διευκολυνθεί,

ενώ το Framework καλεί τον κώδικα του προγραμματιστή και τον περιορίζει σε ένα συγκεκριμένο στυλ κώδικα.

Library ή αλλιώς βιβλιοθήκη, είναι μια συλλογή συναρτήσεων και κλάσεων κώδικα που επιτρέπει στον προγραμματιστή να τις χρησιμοποιήσει για να επιτύχει τους στόχους του με μεγαλύτερη ευκολία

Browser είναι ο φυλλομετρητής που επιτρέπει σε έναν χρήστη να πλοηγηθεί σε οποιαδήποτε σελίδα στο διαδύκτιο. Παραδείγματα Browsers είναι: Chrome, FireFox, Edge, Safari κ.λ.π.

Client είναι ο αποδέκτης πληροφοριών, που εμφανίζει ένα αποτέλεσμα τον τελικό χρήστη. Ο Client μπορεί να είναι και διαδραστικός έτσι ώστε σύμφωνα με τις εκάστοτε εισαγωγές του χρήστη να παίρνει δεδομένα από τον Server και να του τα εμφανίζει. Ένα πολύ διάσημο παράδειγμα client θα ήταν ο Browser. Ο Browser στην ουσία επικοινωνεί με έναν διακομιστή (Server) παίρνει ένα αρχείο HTML και το σχεδιάζει με έναν όμορφο τρόπο στο παράθυρο που βλέπει ο χρήστης.

Server ή διακομιστής είναι μια υπηρεσία που προσφέρει δεδομένα σε διάφορους clients. Ο Server δεν έχει γραφική διεπαφή, αλλά στέλνει και παίρνει πληροφορίες μέσω προγραμματιστικών διεπαφών (API).

API είναι μια διεπαφή προγραμματιστικού περιβάλλοντος που επιτρέπει διαφορετικές εφαρμογές να επικοινωνούν μεταξύ τους. Από τις πιο διάσημες διεπαφές προγραμματιστικού περιβάλλοντος είναι οι: REST, GraphQL και RPC. Η κάθε μια προσφέρει διαφορετικό τρόπο ανάκτησης και επεξεργασίας δεδομένων.

MicroService είναι μια μινιμαλιστική υπηρεσία που ο στόχος της είναι να εκτελεί μια πολύ βασική λειτουργία. Πολλές υπηρεσίες στο διαδίκτυο χρησιμοποιούν αυτό το μοντέλο χρησιμοποιώντας πολλά μικρά MicroServices καθώς αν υπερφορτωθεί το ένα μπορείς να ανοίξεις ένα ακόμα για να μοιράσεις την δουλειά. Έτσι επιτρέπεται με μεγαλύτερη ευκολία το Scaling,

Scaling είναι μια τεχνική για να αυξάνεις το μέγιστο φορτίο που μπορεί να διαχειριστεί μια υπηρεσία. Όσο παραπάνω χρήστες χρησιμοποιούν την υπηρεσία τόσο

παραπάνω Scaling χρειάζεται.

DSL είναι η Domain Specific Language. Είναι μια γλώσσα είτε προγραμματισμού είτε αναπαράστασης δεδομένων που δημιουργείται από μια εφαρμογή για να εξυπηρετήσει τους σκοπούς της.

JDL DSL είναι η Domain Specific Language του JHipster. Σε αυτήν την γλώσσα μπορείς να δηλώσεις το γενικό Domain Model της εφαρμογής σου αλλά παράλληλα και κάποιες ρυθμίσεις, όπως για παράδειγμα τον τύπο της βάσης δεδομένων που θες να χρησιμοποιηθεί, που όταν το πάρει το JHipster θα δημιουργήσει την εφαρμογή που θες με τις ρυθμίσεις που έχεις ορίσει και στον Client και τον Server.

Κεφάλαιο 1

Εισαγωγή

Η βιομηχανία της έβδομης τέχνης έχει αναπτυχθεί ραγδαία. Η τεχνολογική επανάσταση της εποχής μας, έχει οδηγήσει στη ριζική μεταβολή της παραγωγής και αναπαραγωγής ταινιών, καθώς και στην ίδρυση νέων εταιριών παραγωγής, στην αυξανόμενη δημιουργία ταινιών και νέων τεχνολογιών για την υποστήριξή τους.

Στη βιομηχανία του κινηματογράφου, όπως και σε κάθε άλλη, η βιωσιμότητα των εταιριών βασίζεται σε καίριους δείκτες απόδοσης (KPIs). Οι δείκτες αυτοί παρακολουθούνται και χρησιμοποιούνται από τους εργαζομένους στο χώρο του κινηματογράφου σε διάφορες θέσεις.

Για παράδειγμα, ένας αναλυτής πρέπει να είναι συνεχώς ενήμερος για τις εξελίξεις έτσι ώστε να είναι σε θέση να κρίνει την απόδοση των ταινιών, των σχετικών συνεργασιών με ορισμένες χώρες ή εταιρίες, των ηθοποιών. Πρέπει να μπορεί να προβλέψει τις επερχόμενες τάσεις στις προτιμήσεις του κοινού και να παρουσιάσει καίρια στοιχεία για τον ανταγωνισμό. Ένας κυνηγός ταλέντων πρέπει να παρακολουθεί συνεχώς την πορεία νέων ηθοποιών, παραγωγών ταινιών , σκηνοθετών, οπερατέρ και γενικότερα το σύνολο των ρόλων σχετιζόμενων με μια ταινία. Πρέπει επίσης να μπορεί να προβλέψει την αλληλεπίδραση που έχουν οι συντελεστές αυτοί μεταξύ τους. Πόσο καλά θα συνεργάζονται οι ηθοποιοί μεταξύ τους, πόσο καλή συνεργασία θα έχουν οι ηθοποιοί με το συνεργείο παραγωγής και ειδικότερα με τους παραγωγούς, με τους σκηνοθέτες ακόμα και με τους συγγραφείς. Να μπορεί να βρει το σωστό άτομο για την εκάστοτε θέση.

Αυτές οι εργασίες παλιότερα γινόταν πολύ πιο εύκολα καθώς δεν υπήρχε τόσος μεγάλος κορεσμός στην βιομηχανία. Τα τελευταία χρόνια με την ραγδαία αύξηση της

ζήτησης αλλά και της παραγωγής, τα διαθέσιμα δεδομένα και όγκος τους, έχουν εκτοξευθεί, δημιουργώντας την ανάγκη για εργαλεία τα οποία να μπορούν να συλλέγουν αυτόν τον όγκο δεδομένων και να τα εμφανίζουν με έναν κατανοητό τρόπο που να διευκολύνει την ανάλυση και την ανάδειξη ταλέντων. Πλέον είναι πολύ πιο δύσκολο να μπορέσει κάποιος να συλλέξει τόσα δεδομένα και να βγάλει ένα σαφές συμπέρασμα.

Όσο όμως εξελίχθηκε η βιομηχανία του κινηματογράφου, εξελίχθηκε και η τεχνολογία τα τελευταία χρόνια, παρέχοντας τα απαραίτητα εργαλεία που καθιστούν εφικτή τη διαχείριση αυτών των δεδομένων. Η εν λόγω πτυχιακή, έχει ως σκοπό να παρουσιάσει αυτά τα εργαλεία μέσω μιας πρακτικής απεικόνισης των δεδομένων, καλύπτοντας την ανάγκη για τη συλλογή, κατηγοριοποίηση και επεξεργασία ενός μεγάλου όγκου δεδομένων, και εν τέλει την παρουσίασή τους με έναν κατανοητό τρόπο για την εύκολη ανάλυση τους από τους ειδικούς.

Πιο συγκεκριμένα, η πτυχιακή έχει δύο (2) κομμάτια; τον Server και τον Client. Ο Server συλλέγει δεδομένα από διάφορες υπηρεσίες ανοικτών δεδομένων βιομηχανίας κινηματογράφου (OpenData), συσχετίζει τα δεδομένα των διάφορων υπηρεσιών μεταξύ τους, και στη συνέχεια δημιουργεί έναν πίνακα σχέσεων μεταξύ όλων των δεδομένων για περαιτέρω κατηγοριοποίηση. Αφού γίνει ο συσχετισμός των δεδομένων, τα κατηγοριοποιεί, αλλάζει τη μορφή τους και τα αποθηκεύει σε μια βάση δεδομένων. Ο Client είναι αυτός που τα παίρνει από την βάση δεδομένων και τα εμφανίζει με έναν κατανοητό τρόπο στον τελικό χρήστη.

1.1 Δομή της πτυχιακής

Έχοντας εισάγει τον αναγνώστη στο θέμα και τη χρηστικότητα της πτυχιακής στο πρώτο κεφάλαιο, εμβαθύνουμε, παρέχοντας επιπλέον πληροφορίες, ακολουθώντας την παρακάτω δομή:

- Στο κεφάλαιο δύο περιγράφουμε τα OpenData, τι είναι και πως χρησιμοποιούνται
- Στο κεφάλαιο τρία παρουσιάζουμε τις τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη της πτυχιακής
- Στο κεφάλαιο τέσσερα περιγράφουμε τις απαιτήσεις της εφαρμογής.

- Στο κεφάλαιο πέντε περιγράφουμε την αρχιτεκτονική της πτυχιακής
- Στο κεφάλαιο έξι .αναπτύσσουμε το εγχειρίδιο χρήσης, επεξηγώντας όλα τα σχετικά βήματα, κάνοντας παράλληλα χρήση εικόνων.
- Στο κεφάλαιο επτά καταλήγουμε στα συμπεράσματα και τα περιθώρια/περιπτώσεις περαιτέρω ανάπτυξης του θέματος/έρευνας.

1.2 Τεχνολογίες που χρησιμοποιήθηκαν

- **Server**
    1. Java
    2. Parallel Programming
    3. Spring Framework
    4. Liquibase
- **Client**
    1. TypeScript
    2. React Library
    3. Redux Framework
    4. CoreUI Framework
    5. HighCharts Library
- **Εξωτερικές Υπηρεσίες**
    1. PostgreSQL
    2. ElasticSearch
- **Εργαλεία για την ανάπτυξη**
    1. Εργαλείο ανάπτυξης κώδικα Intellij Idea της Jetbrains
    2. Εργαλείο διαχείρισης βάσεων δεδομένων DataGrip της Jetbrains
    3. Εργαλείο διαχείρισης της ElasticSearch, Kibana

4. Docker
5. Εργαλείο διαχείρισης εκδόσεων κώδικα GitLab
6. Εργαλείο CI, GitLab-CI
7. JHipster

Κεφάλαιο 2

OpenData

Open Data είναι η ιδέα ότι ορισμένα δεδομένα πρέπει να είναι διαθέσιμα δωρεάν σε όλον τον κόσμο για να τα χρησιμοποιεί ή να τα αναπαράγει όπως επιθυμεί χωρίς περιορισμούς από νόμους πνευματικών δικαιωμάτων, πατέντες ή μηχανισμούς ελέγχου. Οι στόχοι του κινήματος των open-source data, είναι κοινοί με των άλλων κινημάτων open-source όπως open-source software, hardware κ.ο.κ. (Wikipedia - Open Data 2020). Παραδόξως η ανάπτυξη του κινήματος open-source data είναι παράλληλη με την αύξηση των δικαιωμάτων πνευματικής ιδιοκτησίας.

Αν κάποιος αναρωτιέται γιατί είναι τόσο σημαντικό να είναι σαφές τι σημαίνει Ανοιχτά Δεδομένα και σε τι είναι χρήσιμος αυτός ο ορισμός, υπάρχει μια απλή απάντηση: η διαλειτουργικότητα. Η διαλειτουργικότητα δηλώνει τη δυνατότητα διαφορετικών συστημάτων να λειτουργούν μαζί (διαλειτουργούν). Σε αυτή τη συγκεκριμένη περίπτωση, γίνεται αναφορά στη δυνατότητα να διαλειτουργούν –ή να αναμιγνύουν- διαφορετικά σύνολα δεδομένων.

Η διαλειτουργικότητα (interoperability) είναι σημαντική επειδή επιτρέπει στις διαφορετικές συνιστώσες να λειτουργούν μαζί. Αυτή η δυνατότητα διαμοίρασης και σύνδεσης συνιστωσών έχει θεμελιώδη σημασία για τη δόμηση μεγαλύτερων και πιο πολύπλοκων συστημάτων. Χωρίς τη δυνατότητα διαλειτουργικότητας αυτό γίνεται σχεδόν αδύνατο; απόδειξη αποτελεί η διάσημη ιστορία του Πύργου της Βαβέλ, όπου η αδυναμία επικοινωνίας (διαλειτουργίας) οδήγησε στην ολοκληρωτική κατάρρευση της προσπάθειας οικοδόμησης του. (Open Knowledge Foundation 2020)

Δυστυχώς, το τοπίο των OpenData είναι πολύ θολό καθώς οι εταιρίες που προσφέρουν αυτά τα δεδομένα, πέρα από τους κυβερνητικούς οργανισμούς δεν ακολουθούν

κάποια στάνταρ, και υπάρχουν διαφορετικοί περιορισμοί στην χρήση και αναπαραγωγή αυτών των δεδομένων.

Είναι σημαντικό να υπάρξει ένα πρότυπο ή μια αρμόδια αρχή που να υπαγορεύει ακριβώς πως θα χρησιμοποιούνται τα OpenData γενικότερα αλλά επίσης είναι σημαντικό όλο και περισσότερες εταιρίες να υιοθετήσουν αυτό το μοντέλο, καθώς με το ”άνοιγμα” των δεδομένων δημιουργούνται νέες ιδέες για νέες τεχνολογίες και αναπτύσσεται γενικότερα η τεχνολογία και η κοινωνία.

Τα open source data (OpenData) δεν είναι πολύ διαφορετικά από το open source sofwtare (ή όπως χρησιμοποιείται ευρέως) OpenSource. Το κίνημα του open source software είναι ένα κίνημα για δωρεάν και ελεύθερο λογισμικό που η κοινωνία μπορεί να συμμετέχει στην εξέλιξη του και στην διασφάλιση της ποιότητας του. Το κίνημα των open source data είναι ένα κίνημα για δωρεάν και ελεύθερα δεδομένα τα οποία μπορεί να αξιοποιήσει η κοινωνία για την δημιουργία open source software.

Έχοντας παραπάνω πηγές open source data, θα δημιουργούνται περισσότερες ιδέες και παραπάνω εργαλεία που θα συμβάλλουν σημαντικά στην εξέλιξη μας. Όσες περισσότερες ιδέες και εργαλεία τόσα παραπάνω open source software θα γεννηθούν για την αξιοποίηση αυτών των δεδομένων. Η παρούσα πτυχιακή είναι ένα project το οποίο γεννήθηκε μέσα από τη δυνατότητα αξιοποίησης των open source data. Δεν θα ήταν εφικτή η υλοποίησή της χωρίς αυτά.

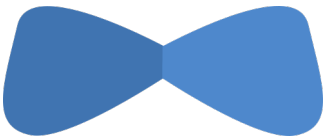
Κεφάλαιο 3

Τεχνολογίες

Με τις γλώσσες προγραμματισμού έχουμε απεριόριστες δυνατότητες, με μόνο περιορισμό τη φαντασία μας. Ο προγραμματισμός είναι μια διαδικασία που υφίσταται εδώ και πάρα πολλά χρόνια, και έχει εξελιχθεί ραγδαία. Η εξέλιξη αυτή οδήγησε στην ανάπτυξη τεχνολογιών, εργαλείων και βιβλιοθηκών ανοιχτού κώδικα που μας επιτρέπουν να υλοποιούμε με ευκολία λειτουργίες στα προγράμματα μας που διαφορετικά θα ήταν ιδιαιτέρως χρονοβόρες. Αυτά τα εργαλεία έχουν σχεδιαστεί έτσι ώστε να είναι προσαρμόσιμα σε κάθε περίπτωση και για κάθε χρήση, μέσω μερικών απλών ρυθμίσεων.

Η χρήση τέτοιων εργαλείων επιτρέπει σε έναν προγραμματιστή να επικεντρωθεί στην ουσία του προγράμματος παρά στις υλοποιήσεις της υποδομής και της ασφάλειας. Τέτοια εργαλεία έχουν χρησιμοποιηθεί για την ανάπτυξη αυτής της πτυχιακής. Παρακάτω αναφέρονται μερικά από τα πιο σημαντικά εργαλεία που χρησιμοποιήθηκαν.

3.1 JHipster



Διάγραμμα 3.1: Λογότυπο JHipster

Το JHipster είναι ένα εργαλείο αυτόματης παραγωγής κώδικα. Στόχος του είναι η αυτόματη δημιουργία αρχείων κώδικα τα οποία επαναλαμβάνονται για κάθε εφαρμογή

του είδους.

Για παράδειγμα, για την δημιουργία μιας υπηρεσίας API στην Java, χρησιμοποιώντας μια βάση δεδομένων SQL, κατά προτίμηση πρέπει αρχικά να επιλεγεί ένα Web Framework, για να διευκολύνει την διαχείριση και την ανάπτυξη του προγράμματος. Αφού επιλεγεί αυτό το Framework, πρέπει να ρυθμιστεί ανάλογα για να καλύψει τις ανάγκες της εφαρμογής. Στην συγκεκριμένη περίπτωση πρέπει να ρυθμιστεί να στέλνει δεδομένα τύπου JSON ή XML αφού ο στόχος είναι η δημιουργία μιας υπηρεσίας API. Πρέπει επίσης να ρυθμιστεί η σύνδεση με την βάση δεδομένων, πρέπει να ρυθμιστεί πως θα εμφανίζονται οι ημερομηνίες στις απαντήσεις... κ.ο.κ.

Το JHipster δημιουργεί όλα αυτά τα αρχεία και τις ρυθμίσεις δίνοντας την επιλογή περαιτέρω ρύθμισης και με αυτόν τον τρόπο εξοικονομεί χρόνο από την ρύθμιση της υποδομής και επιτρέπει άμεση εστίαση στην υλοποίηση της εφαρμογής γνωρίζοντας ότι υπάρχει μια ισχυρή υποδομή που έχει δημιουργηθεί από το JHipster με την επιλογή της επιπρόσθετων ρυθμίσεων για την βελτιστοποίηση της λειτουργίας της εφαρμογής.

Εκτός από τις ρυθμίσεις επιτρέπει τη δήλωση του Μοντέλου της εφαρμογής στην JDL DSL του, και μπορεί να δημιουργήσει τα ανάλογα Entities, Services Και Controllers και στο Frontend και στο Backend.

Η πτυχιακή αυτή αρχικά δημιουργήθηκε με το JHipster χρησιμοποιώντας το Spring Framework για το backend και την React Library για το Frontend. Επιπρόσθετα προσέθεσε ρυθμίσεις και για αρκετά άλλες τεχνολογίες που χρησιμοποιήθηκαν.

3.2 Backend

3.2.1 Spring Framework

Διάγραμμα 3.2: Λογότυπο Spring Framework

Το Spring Framework είναι ένα Framework και IoC Container που στοχεύει στην εύκολη ανάπτυξη, διαχείριση και συντήρηση μιας εφαρμογής μεγάλου μεγέθους. Έχει σχεδιαστεί για να χρησιμοποιείται σε οποιουδήποτε είδους εφαρμογής αλλά προσφέρει

επεκτάσεις και πακέτα για την ανάπτυξη Web Υπηρεσιών πάνω στην πλατφόρμα Java EE (Wikipedia - Spring Framework 2020)

Το Spring Framework είναι η καρδία του Backend της πτυχιακής καθώς πάνω σε αυτήν έχει δομηθεί και όλος ο κώδικας.

3.2.2 ElasticSearch

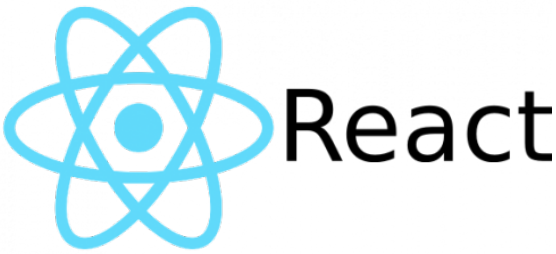
Διάγραμμα 3.3: Λογότυπο Spring Framework

Η ElasticSearch, είναι μια μηχανή αναζήτησης πραγματικού χρόνου και ανάλυσης δεδομένων. Βασίζεται στο lucene της Apache, μια υψηλών επιδόσεων με αρκετά χαρακτηριστικά βιβλιοθήκη για μηχανές αναζήτησης. (Wikipedia - ElasticSearch 2020)

Χρησιμοποιείται για την αναζήτηση Ηθοποιών, Εταιριών & Χωρών Παραγωγής και Ειδών Ταινιών, από το Frontend με μεσολαβητή το Backend.

3.3 Frontend

3.3.1 React Library



Διάγραμμα 3.4: Λογότυπο React

Η React είναι μια βιβλιοθήκη γραμμένη σε JavaScript με σκοπό την εύκολη δημιουργία SPA Σελίδων, και δημιουργήθηκε από την Facebook. Χαρακτηρίζεται βιβλιοθήκη και όχι Framework καθώς είναι μινιμαλιστική και διαθέτει πολύ απλές λειτουρ-

γίες και βασίζεται σε άλλα OpenSource πακέτα για τις πιο σύνθετες λειτουργίες όπως Routing, State Management κ.α.

Το Frontend της πτυχιακής χρησιμοποιεί σαν κύρια τεχνολογία την React σε συνδυασμό με το Redux.

3.3.2 Redux

Διάγραμμα 3.5: Λογότυπο Redux

Το Redux είναι ένα προβλέψιμο state container για εφαρμογές JavaScript. Θεωρείται προβλέψιμο διότι το State (κατάσταση της εφαρμογής) είναι αμετάβλητο και σε κάθε πρόθεση αλλαγής γίνεται αντικατάσταση. Με αυτόν τον τρόπο καθίσταται δυνατή η αλλαγή του State της εφαρμογής οπουδήποτε μέσα στο ιστορικό (undo,redo). Επίσης διευκολύνει το debugging καθώς δεν υπάρχει mutability και επομένως η χρονική στιγμή και οι παράγοντες κάθε αλλαγής είναι ορατοί ανά πάσα στιγμή/συνεχώς.

Το Redux δεν σχεδιάστηκε για την React, αλλά μεγάλο μέγεθος εφαρμογών γραμμένων σε React το χρησιμοποιούν λόγω των μεγάλων δυνατοτήτων και της απλότητας του. Οι δημιουργοί του Redux δημιούργησαν επίσης ένα επίσημο πακέτο για την εύκολη επικοινωνία της React με το Redux, με όνομα react-redux.

Το react-redux χρησιμοποιήθηκε στο Frontend της Πτυχιακής, μαζί με κάποιες ακόμα επεκτάσεις του για την λήψη των δεδομένων από το Backend καθώς και για το Caching. Χρησιμοποιήθηκε επίσης και για την δρομολόγηση των αποτελεσμάτων που θα εμφανίζονται.

Κεφάλαιο 4

Απαιτήσεις εφαρμογής

Με όλη αυτή την ανάπτυξη του κινηματογράφου και των τεχνολογιών που δημιουργήθηκαν γύρο από αυτόν, δημιουργήθηκαν εργαλεία που μας προσφέρουν αναλυτικά δεδομένα για ταινίες, σειρές κ.ο.κ. Αυτά τα εργαλεία έχουν πλούσιες βάσεις δεδομένων με πολλές εγγεγραμμένες ταινίες αλλά προσφέρουν στοιχεία μόνο για τις ταινίες αυτές. Δεν προσφέρουν συνδυαστικά και συγκεντρωτικά στοιχεία για όλες τις οντότητες που περιβάλλουν τον κινηματογράφο. Οντότητες όπως ηθοποιοί, εταιρίες παραγωγής, χώρες παραγωγής και είδη ταινιών. Υπάρχουν τόσα αναξιοποίητα δεδομένα που με τον συνδυασμό τους θα μπορούσε να δημιουργηθεί μια υπηρεσία η οποία θα μπορούσε να προσφέρει αναλυτικά στοιχεία σε βάθος χρόνου όλων αυτών των οντοτήτων. Έτσι λοιπόν γεννήθηκε η ιδέα της εφαρμογής MovieInsights.

Με την παρούσα εφαρμογή θέλω να δημιουργήσω ένα εργαλείο το οποίο θα μπορεί να χρησιμοποιηθεί από τους ειδικούς του χώρου της βιομηχανίας του κινηματογράφου αλλά παράλληλα και να δείξω πως με την αξιοποίηση αυτών των δεδομένων μπορούν να δημιουργηθούν ακόμη περισσότερες τεχνολογίες για την διευκόλυνσή των εργασιών στο χώρο. Επίσης δεν θα ήθελα να δημιουργήσω μια τεχνολογία που θα μπορούσαν μόνο οι ειδικοί να χρησιμοποιήσουν αλλά να μπορεί και ο απλός ο κόσμος να έχει πρόσβαση και να μπορεί να καταλάβει αυτόν τον μεγάλο όγκο δεδομένων σχεδιάζοντας μια φιλική προς τον χρήστη διεπαφή.

Στο υπόλοιπο αυτή της ενότητας παρουσιάζονται οι βασικές ιστορίες χρήστη της εφαρμογής

4.1 Stories Γενικού Χρήστη

1η ιστορία

Σαν Χρήστης θα ήθελα να μου προσφέρονται αναλυτικά στοιχεία ανά 4 κατηγορίες: ανά Συντελεστή, ανά Χώρα Παραγωγής, ανά Εταιρία Παραγωγής και ανά Είδος Ταινίας έτσι ώστε να έχω μια πιο μεγάλη ευελιξία στις αναζητήσεις μου και στην προβολή των στοιχείων αυτών.

2η ιστορία

Σαν Χρήστης θα ήθελα να μου προσφέρονται αυτά τα αναλυτικά στοιχεία στις 4 κατηγορίες και ανά χρόνο για να ειδικεύσω περισσότερο την αναζήτηση μου έτσι ώστε να έχω μια πιο ολοκληρωμένη εικόνα των δεδομένων που ψάχνω.

3η ιστορία

Σαν Χρήστης θα ήθελα να υπάρχει ένα πεδίο αναζήτησης για να μπορώ να ψάχνω συντελεστές , χώρες παραγωγής, εταιρίες παραγωγής και είδη ταινιών έτσι ώστε να με διευκολύνει στην εύρεση των ζητούμενων στοιχείων.

4η ιστορία

Σαν Χρήστης θα ήθελα όταν επιλέξω ένα από τα εμφανιζόμενα στοιχεία σε μια κατηγορία, να ανανεωθούν όλα τα δεδομένα βάση του επιλεχθέντος στοιχείου, ώστε να έχω μια στοχευμένη ανασκόπηση των δεδομένων σύμφωνα με την επιλογή μου.

5η ιστορία

Καθώς υπάρχουν ήδη εργαλεία τα οποία προσφέρουν αναλυτικά στοιχεία για ταινίες, ως χρήστης δε θα ήθελα να δω έναν κλώνο αυτών των υπηρεσιών. Θα ήθελα, όταν θα πατήσω σε μία από αυτές τις ταινίες, αφού μου εμφανιστεί ένα παράθυρο με τα πολύ βασικά στοιχεία, να μου προσφερθεί η επιλογή για ανακατεύθυνση σε μια από τις υπάρχουσες υπηρεσίες, σε περίπτωση που θελήσω περισσότερες πληροφορίες.

6η ιστορία

Σαν χρήστης θα ήθελα όταν θα μπω στην ιστοσελίδα, να υπάρχει ένας παγκόσμιος χάρτης στον οποίο να αναδεικνύεται ποιά χώρα έχει τις περισσότερες ταινίες. Επίσης, όταν μετακινήσω τον κένσορα του ποντικιού πάνω από μια χώρα θα ήθελα να εμφανιστεί ένα μικρο παραθυράκι (tooltip) πάνω από την χώρα, το οποίο να αναγράφει σε πόσες ταινίες έχει συμμετάσχει στην παραγωγή τους αυτή η χώρα. Αν πατήσω στην χώρα θα ήθελα να εμφανιστούν όλα τα συνδυαστικά στοιχεία για αυτήν την χώρα. Οι λειτουργίες αυτές στοχεύουν στην άμεση ανάδειξη των στατιστικών χρησιμοποιώντας γεωγραφικά κριτήρια. Η ύπαρξη του παγκόσμιου χάρτη έχει ως σκοπό τη διευκόλυνση της αναζήτησης μέσω της οπτικοποίησης των δεδομένων.

4.2 Stories Εταιρίας Παραγωγής

1η ιστορία

Σαν εταιρία παραγωγής θέλω να δω για κάθε κατηγορία τις ταινίες με τα μεγαλύτερα / μικρότερα έσοδα και έξοδα έτσι ώστε να έχω μια γενική ιδέα για το ποιές ταινίες τα πήγαν καλά για να αποφασίσω τι είδους ταινίας θα παράξω.

2η ιστορία

Σαν εταιρία παραγωγής θα ήθελα να μπορώ να δω το μέσο όρο και τα συνολικά, έσοδα, έξοδα και το καθαρό κέρδος των ταινιών ανά κατηγορία και ανά χρόνο, ώστε να έχω άμεση εικόνα πιο στοχευμένων οικονομικών στοιχείων με σκοπό μελλοντικών επενδύσεων.

3η ιστορία

Σαν εταιρία παραγωγής θα ήθελα να μπορώ να δω τη μέση βαθμολογία των ταινιών ανά χρόνο αλλά και συνολικά για κάθε κατηγορία, και την τάση αύξησης η μείωσης της βαθμολογίας (Mean) ανά χρόνο, ώστε να μπορώ να δω τα υπάρχοντα δεδομένα αλλά και τις σχετικές προβλέψεις.

4.3 Stories Παραγωγού ταινιών

1η ιστορία

Σαν Παραγωγός ταινιών σε κάθε κατηγορία θα ήθελα να εμφανίζονται στοιχεία για τον πιο δημοφιλή και λιγότερο δημοφιλή ηθοποιό έτσι να μπορώ να βρώ τον ηθοποιό που χρειάζομαι για την ταινία που θα παράξω στο μέλλον.

2η ιστορία

Σαν Παραγωγός ταινιών σε κάθε κατηγορία θα ήθελα να εμφανίζονται στοιχεία για τον πιο δημοφιλή και λιγότερο δημοφιλή σκηνοθέτη έτσι ώστε να μπορώ να βρω τον σκηνοθέτη που χρειάζομαι για την ταινία που θα παράξω στο μέλλον.

3η ιστορία

Σαν Παραγωγός ταινιών σε κάθε κατηγορία θα ήθελα να εμφανίζονται στοιχεία για τον πιο δημοφιλή και λιγότερο δημοφιλή παραγωγό έτσι ώστε να μπορώ να δω για την κάθε κατηγορία ποιος συνάδελφος τα έχει πάει καλύτερα.

4η ιστορία

Σαν Παραγωγός ταινιών θα ήθελα να μπορώ στην κατηγορία ανά συντελεστή – παραγωγό, να δω τα στοιχεία των συναδέλφων και σε περίπτωση που ο παραγωγός που έχω επιλέξει είναι ο εαυτός μου, ποιές από τις συνεργασίες μου απέδωσαν και ποιες από τις ταινίες μου είχαν επιτυχία.

4.4 Stories Cinefil

1η ιστορία

Σαν Σινεφίλ θα ήθελα να δω τις ταινίες με τη μεγαλύτερη και χαμηλότερη βαθμολογία ανά κατηγορία, ώστε να διευκολυνθεί η επιλογή μου για τις ταινίες που θα παρακολουθήσω αργότερα.

2η ιστορία

Σαν Σινεφίλ θα ήθελα να δω την κατηγορία που έχω επιλέξει το πιο δημοφιλές και λιγότερο δημοφιλές είδος ταινίας έτσι ώστε να διευρύνω τις επιλογές μου για τις ταινίες που θα παρακολουθήσω μελλοντικά.

3η ιστορία

Σαν Σινεφίλ θα ήθελα στην κατηγορία των συντελεστών, να υπάρχουν στοιχεία για τον κάθε συντελεστή σε οποιοδήποτε ρόλο έχει συμμετάσχει στην καριέρα του, και κάθε φορά να μπορώ να επιλέξω για ποιό ρόλο να δω τα στοιχεία αυτά έτσι ώστε να μπορώ να εξερευνήσω παραπάνω περιεχόμενο για το άτομο που ψάχνω.

4.5 Stories Administrator

1η ιστορία

Ως Administrator θα ήθελα να υπάρχει μια σελίδα διαχείρισης για να μπορώ να διαχειριστώ και να παρακολουθώ τα στοιχεία της εφαρμογής ανά πάσα στιγμή.

2η ιστορία

Θα ήθελα σαν Administrator μέσα στην σελίδα διαχείρισης να μπορώ να ελέγχω το traffic της εφαρμογής έτσι ώστε να μπορώ ανά πάσα στιγμή να δώσω παραπάνω resources για να μην παρατηρηθεί downtime.

3η ιστορία

Σαν Administrator θα ήθελα να μπορώ να παρακολουθώ μέσα στην σελίδα διαχείρισης τα HTTP Status Codes και πόσα είναι στο σύνολό τους, έτσι ώστε να καταλάβω αν η εφαρμογή αντιμετώπισε κάποιο πρόβλημα για να μπορέσω να το λύσω και επιπρόσθετα να παρακολουθήσω αν υπήρχε κάποια προσπάθεια για παραβίαση της εφαρμογής από κάποιον κακόβουλο χρήστη.

4η ιστορία

Σαν Administrator θα ήθελα να μπορώ να παρακολουθώ ποιές σελίδες ζητήθηκαν από τους χρήστες και πόσες φορές έτσι ώστε να έχω μια γενική ιδέα για το τι θέλουν να βλέπουν οι χρήστες έτσι ώστε αργότερα να μπορέσω να εξελίξω την εφαρμογή στην σωστή κατεύθυνση.

5η ιστορία

Σαν Administrator θα ήθελα να μπορώ να παρακολουθήσω τα στατιστικά της βάσης δεδομένων και τους χρόνους απόκρισής της έτσι ώστε να μπορώ να την κάνω Scale σε περίπτωση υπερφόρτωσης για να μειωθεί σημαντικά ο χρόνος φόρτωσης των δεδομένων.

6η ιστορία

Σαν Administrator θα ήθελα να υπάρχει μια επιλογή που να επιτρέπει την αλλαγή των επιπέδων καταγραφής δεδομένων των υπηρεσιών της εφαρμογής σε περίπτωση που εμφανιστεί κάποιο πρόβλημα, για να μπορέσω να το λύσω πιο εύκολα και όταν λυθεί το πρόβλημα να ξανά αλλάξω τα επίπεδα καταγραφής στο ελάχιστο έτσι ώστε να εξοικονομήσω αποθηκευτικό χώρο στον Server που τρέχει η εφαρμογή.

Κεφάλαιο 5

Αρχιτεκτονική Εφαρμογής

Η πτυχιακή αποτελείται από 2 μέρη. Τής επεξεργασίας και διάθεσης των δεδομένων (Server), και της κατανάλωσης και οπτικοποιήσης των δεδομένων (Client).

Κατά την εκκίνησή του Server, προτού μπει σε κατάσταση ετοιμότητας και είναι έτοιμος ώστε να διαθέσει τα δεδομένα στον Client, ξεκινάει κάποιους ελέγχους. Αυτοί οι έλεγχοι είναι σημαντικοί για να μπορεί να λειτουργήσει ορθώς όλη η εφαρμογή. Οι έλεγχοι αυτοί τρέχουν από αρμόδια συστήματα που μπορούν να φέρουν την εφαρμογή σε κατάσταση ετοιμότητας.

Αφού η εφαρμογή τεθεί σε κατάσταση ετοιμότητας, οι ανάλογες υπηρεσίες προσφέρουν δεδομένα στον Client. Μια από τις πιο σύνθετες υπηρεσίες είναι η υπηρεσία της Αναζήτησης των δεδομένων η οποία θα αναπτυχθεί παρακάτω.

5.1 Υπηρεσία εισαγωγής δεδομένων

Το σύστημα εισαγωγής δεδομένων αρχικά ελέγχει τη βάση δεδομένων για την ύπαρξη δεδομένων ταινιών, συντελεστών, εταιριών και χωρών παραγωγής και ειδών ταινιών. Είναι σημαντικό για την συνέχεια να υπάρχουν δεδομένα σε όλες τις κατηγορίες. Αν δεν υπάρχουν δεδομένα σε οποιαδήποτε από όλες τις κατηγορίες προχωράει η άμεση εισαγωγή δεδομένων από άλλες υπηρεσίες.

Το σύστημα εισαγωγής δεδομένων ανάλογα με τις ρυθμίσεις που του έχουν δοθεί θα προσπαθήσει να πάρει δεδομένα από διάφορες εξωτερικές υπηρεσίες και αφού τα συσχετίσει μεταξύ τους θα τα αποθηκεύσει στην βάση δεδομένων.

Κατά την εκπόνηση της Πτυχιακής, παρουσιάστηκαν κάποια προβλήματα. Το κυ-

ριότερο ήταν η εύρεση μιας υπηρεσίας, η οποία να μπορεί να διαθέσει όλα τα δεδομένα που χρειάζονται για να λειτουργήσει σωστά η εφαρμογή.

Μια από τις πιο διάσημες και αξιόπιστες υπηρεσίες η οποία διαθέτει API για εύκολη συλλογή δεδομένων είναι η TMDb (TheMovieDb) η οποία προσφέρει πλούσια στοιχεία για πάρα πολλές ταινίες. Δυστυχώς, αυτή η υπηρεσία είναι δημοφιλής μόνο στην προγραμματιστική κοινότητα και όχι ιδιαίτερα στο ευρύ κοινό με αποτέλεσμα να μη διαθέτει αξιόπιστες βαθμολογίες ταινιών καθώς υπάρχουν πολύ λίγες αξιολογήσεις.

Από την άλλη η πιο διάσημη υπηρεσία δεδομένων ταινιών στο ευρύ κοινό είναι η IMDb (Internet Movie Database) η οποία όμως δεν διαθέτει υπηρεσία API για πρόσβαση στα δεδομένα. Διαθέτει όμως ένα μικρό κομμάτι των δεδομένων της σε μορφή συμπιεσμένου TSV για οικιακή ή μη-εμπορική χρήση. Μέσα σε αυτό το μικρό κομμάτι των δεδομένων που διαθέτει, υπάρχουν και βαθμολογίες ταινιών.

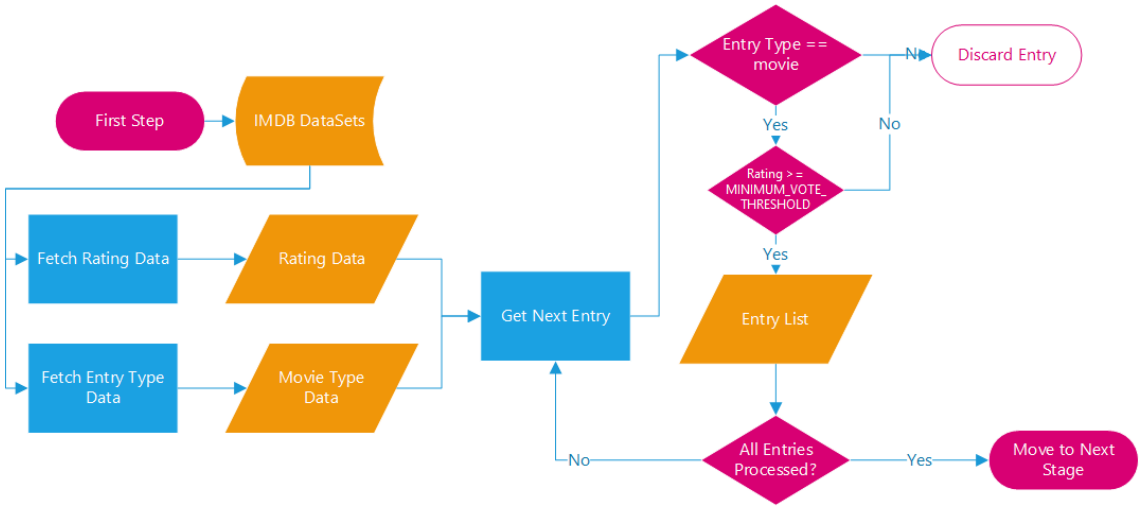
Το άλλο πρόβλημα ήταν να βρεθεί ο τρόπος για να μπορούν να συσχετισθούν τα δεδομένα των βαθμολογιών ταινιών από την υπηρεσία IMDb με τα δεδομένα ταινιών από την υπηρεσία TMDb. Αυτό αντιμετωπίστηκε επιτυχώς με την τελευταία ενημέρωση της υπηρεσίας TMDb με την οποία υπάρχει η δυνατότητα αναζήτησης ταινίας με το αναγνωριστικό κλειδί που προσφέρει η υπηρεσία IMDb.

Οι παραπάνω υπηρεσίες χρησιμοποιήθηκαν συνδυαστικά, πετυχαίνοντας την απόκτηση πλούσιων δεδομένων από την υπηρεσία TMDb και ταυτόχρονα πιο αξιόπιστων βαθμολογιών από την υπηρεσία IMDb.

5.1.1 Βήμα 1ο - Δεδομένα από IMDb

Με τις ρυθμίσεις που δόθηκαν στο σύστημα εισαγωγής δεδομένων για αυτήν την πτυχιακή αρχικά παίρνει ένα συμπιεσμένο αρχείο με δεδομένα βαθμολογιών ταινιών από την υπηρεσία IMDb το αποσυμπιέζει και αποθηκεύει τα δεδομένα στην μνήμη. Η Μορφή του αρχείου είναι TSV που στην ουσία είναι ένα αρχείο σαν πίνακας το οποίο περιέχει στήλες. Οι 3 απαραίτητες στήλες είναι το αναγνωριστικό μιας ταινίας η μιας σειράς (imdbId), η βαθμολογία της ταινίας/σειράς (rating) καθώς και το νούμερο των ψήφων με το οποίο βγήκε αυτή η βαθμολογία (voteCount). Έπειτα παίρνει ένα ακόμα συμπιεσμένο αρχείο από την υπηρεσία IMDb του οποίου η μορφή είναι επίσης TSV με στήλες το αναγνωριστικό μιας ταινίας η μιας σειράς (imdbId) και ο τύπος (titleType). Ο τύπος διευκρινίζει εάν αν είναι ταινία, σειρά ή κάτι άλλο.

Αφού φορτώσει και τα 2 αρχεία στη μνήμη ξεχωρίζει τις εγγραφές που αναφέρουν μόνο ταινίες και μετά ξεχωρίζει τις εγγραφές οι οποίες έχουν αριθμό ψήφων μεγαλύτερο ή ίσο με μια σταθερά MINIMUM_VOTES_THRESHOLD και κρατάει τα 3 αρχικά πεδία του πρώτου αρχείου. Η σταθερά αυτή έχει καθοριστεί ως 1000 ψήφοι.



Διάγραμμα 5.1: Διάγραμμα ροής δεδομένων από IMDB

5.1.2 Βήμα 2ο - Δεδομένα από TMDb

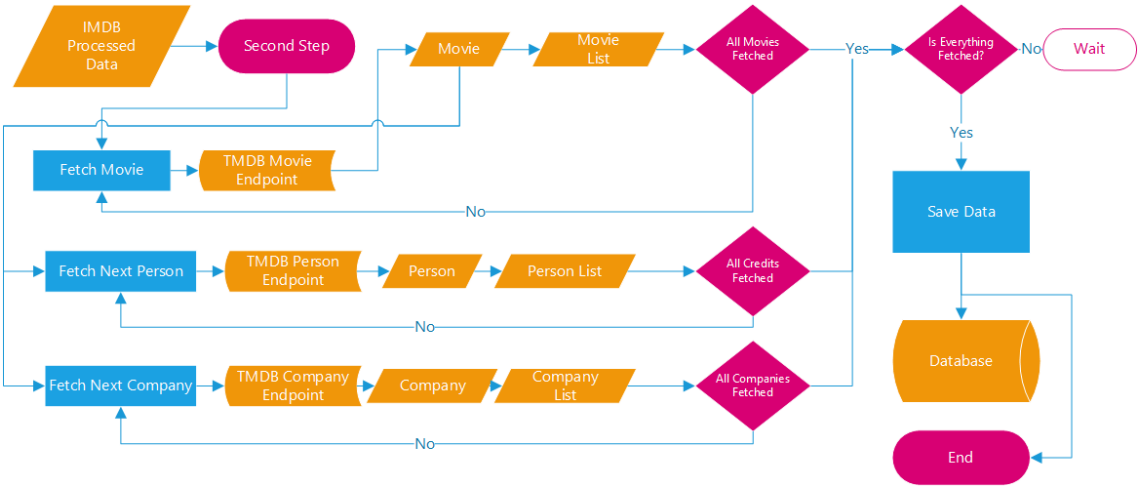
Στο 2ο βήμα αφού έχουμε τα δεδομένα που χρειάζονται από το πρώτο, δημιουργεί και προγραμματίζει για εκτέλεση ασύγχρονα ερωτήματα ζητώντας πλήρη δεδομένα ταινιών για κάθε αναγνωριστικό ταινίας που έχουμε στην υπηρεσία TMDb.

Ο στόχος δεν είναι μόνο η απόκτηση δεδομένων ταινιών αλλά και συντελεστών, εταιριών και χωρών παραγωγής, και ειδών ταινιών, συνεπώς αφού πάρει τα δεδομένα ταινιών θα χρονοδρομολογήσει και άλλα ασύγχρονα ερωτήματα στην εν λόγω υπηρεσία για να πάρει και τα υπόλοιπα δεδομένα.

Αφού τελειώσει αυτή η διαδικασία, συσχετίζει όλα τα δεδομένα μεταξύ τους και τα εισάγει στην βάση δεδομένων για περαιτέρω επεξεργασία από κάποιο άλλο σύστημα.

5.2 Σύστημα επεξεργασίας δεδομένων

Ο Ρόλος του Συστήματος επεξεργασίας δεδομένων είναι να πάρει τα ανεπεξέργαστα δεδομένα από την βάση δεδομένων και να δημιουργήσει σχέσεις μεταξύ τους

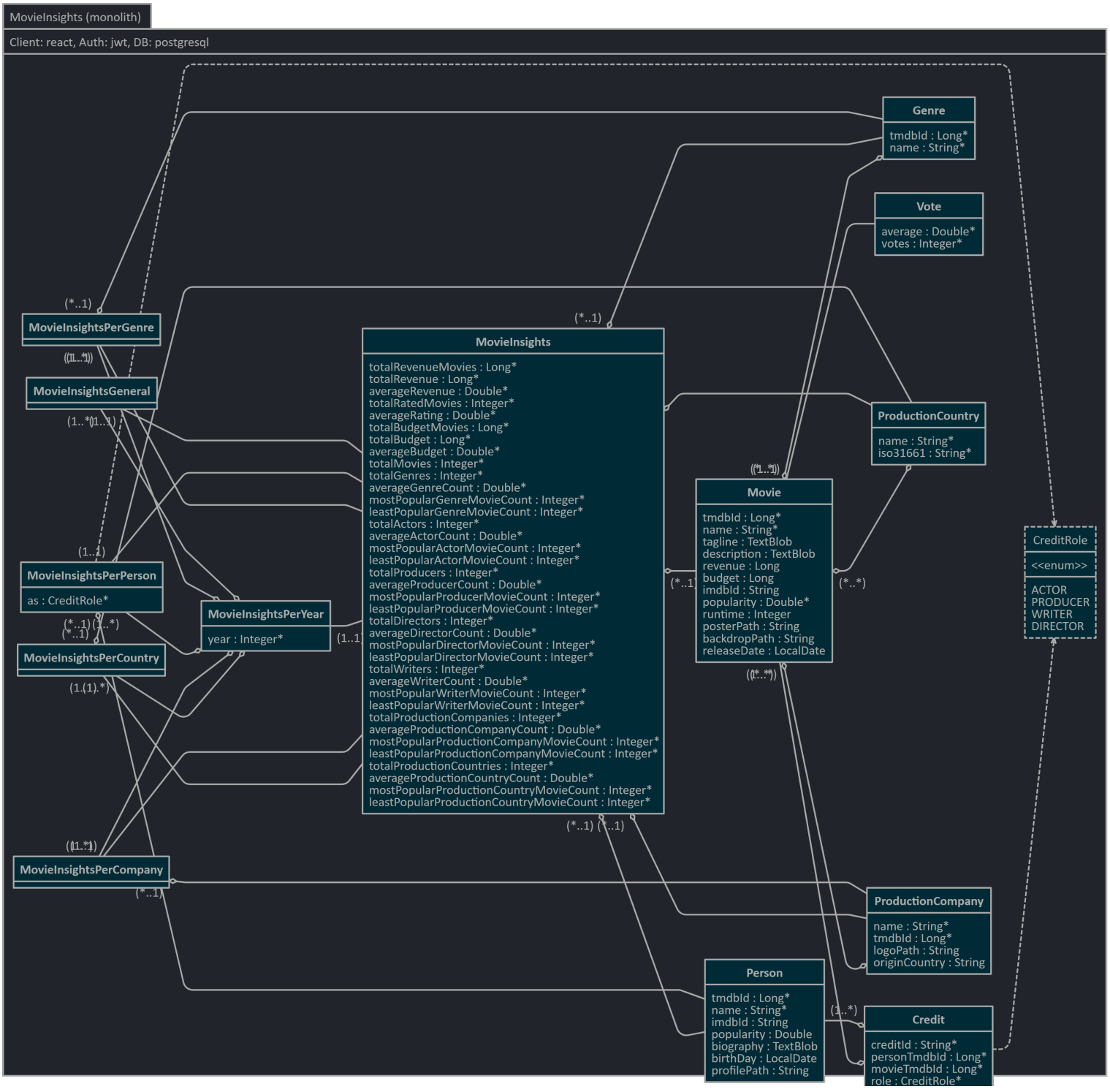


Διάγραμμα 5.2: Διάγραμμα ροής δεδομένων από TMDb

κατηγοριοποιώντας τα. Η Εφαρμογή προσφέρει δεδομένα ανά 4 κατηγορίες, και παράλληλα για κάθε κατηγορία ανά χρονολογική περίοδο. Η κατηγοριοποίηση είναι βασισμένη στα παρακάτω στοιχεία:

- Στοιχεία ανά χώρα παραγωγής
- Στοιχεία ανά εταιρία παραγωγής
- Στοιχεία ανά συντελεστή - ξεχωριστά ανά ηθοποιό, συγγραφέα, σκηνοθέτη και παραγωγό
- Στοιχεία ανά είδος ταινίας

Η κάθε κατηγορία περιέχει στοιχεία για την ταινία με την μεγαλύτερή αξιολόγηση, την ταινία με την χαμηλότερη αξιολόγηση, την μέση βαθμολογία ταινιών για αυτήν την κατηγορία, τις ταινίες με τα μεγαλύτερα και τα μικρότερα έσοδα και έξοδα, τα μέσα έξοδα / έσοδα ταινιών, τον πιο δημοφιλή και λιγότερο δημοφιλή ηθοποιό (ή συμπρωταγωνιστή αν η κατηγορία είναι ανά συντελεστή – ηθοποιό), παραγωγό (η συμπαραγωγό αν η κατηγορία είναι ανά συντελεστή – παραγωγό), σκηνοθέτη (η συν-σκηνοθέτη αν η κατηγορία είναι ανά συντελεστή – σκηνοθέτη) και συγγραφέα (ή συν-συγγραφέα αν η κατηγορία είναι ανά συντελεστή – συγγραφέα), την πιο δημοφιλή και λιγότερο δημοφιλή χώρα παραγωγής (η συμπαραγωγής αν η κατηγορία είναι ανά χώρα παραγωγής), την πιο δημοφιλή και λιγότερο δημοφιλή εταιρία παραγωγής (ή συμπαραγωγής αν η κατηγορία είναι ανά εταιρία παραγωγής) κ.α όπως φαίνονται στο σχήμα 5.3.



Διάγραμμα 5.3: Μοντελο MovieInsights

Η επεξεργασία των δεδομένων μπορούσε να γίνει είτε στο κομμάτι της βάσης δεδομένων με χρήση της γλώσσας προγραμματισμού SQL είτε στο κομμάτι του Server με χρήση της γλώσσας προγραμματισμού Java. Ο πιο αποδοτικός τρόπος ήταν να γίνει στην βάση δεδομένων με την SQL καθώς υπάρχουν διάφορα συστήματα βελτιστοποίησης τα οποία επιτρέπουν την επεξεργασία των δεδομένων αποδοτικά και γρήγορα, αλλά σε αυτήν την πτυχιακή προτιμήθηκε η Java για λόγους απλότητας.

Το σύστημα επεξεργασίας χωρίζεται σε 3 διαφορετικά κομμάτια. Το πρώτο κομμάτι συλλέγει και κατηγοριοποιεί τα δεδομένα. Το δεύτερο κομμάτι υπολογίζει τα μέγιστα τα ελάχιστα και τους μέσους των δεδομένων. Το τρίτο κομμάτι μετατρέπει τα δεδομένα σε μια φιλική προς τη βάση δεδομένων μορφή και τα αποθηκεύει.

5.2.1 Βήμα 1 - Κατηγοριοποίηση δεδομένων

```
1   categories = new HashMap<>();
2   getMovieInsights(long id, BaseWrapper<?> o) {
3       if (categories.containsKey(id)) {
4           return categoriess.get(id);
5       }
6       IMovieInsightsWrapper wrapper = createMovieInsightsWrapper(o);
7       categories.put(id,wrapper);
8       return wrapper;
9   }
10  getMovieInsightObjectsOptimized(Movie movie) {
11   // return set of getMovieInsights(...)
12  }
13  categorizeData(Movie movie) {
14      Set<IMovieInsightsWrapper> miSet = getMovieInsightObjectsOptimized(movie);
15      miSet.forEach(wrapper -> {
16          wrapper.submitMovie(movie);
17      });
18  }
19  Lists
20    .partition(dto.movies, 1000)
21    .forEach(chunk -> {
22       chunk.forEach(this::categorizeData);
23    });
```

Διάγραμμα 5.4: Σειριακή κατηγοριοποίηση με HashMap

Ο Αλγόριθμός κατηγοριοποίησης συλλέγει τα δεδομένα, τα προσθέτει στις ανάλο-

γες κατηγορίες, δημιουργεί τις κατηγορίες αν δεν υπάρχουν, και ετοιμάζει τα δεδομένα για τον τελικό υπολογισμό.

Υπήρξαν πολλές αναθεωρήσεις του αλγορίθμου μέχρι την τελική του έκδοση. Οι περισσότερες αφορούσαν βελτιστοποιήσεις απόδοσης.

Ο Αρχικός κώδικας (βλ σχήμα: 5.4) ήταν αρκετά απλός. Είχε HashMaps για την αποθήκευση των δεδομένων ανά κατηγορία, και γινόταν συνεχώς έλεγχοι για το αν υπήρχαν τα δεδομένα. Αν όχι τα δημιουργούσε αλλιώς τα μορφοποιούσε.

Αρχικά ο κώδικας έτρεχε σειριακά, καθώς όμως αυξανόταν τα δεδομένα, ανέβαινε εκθετικά και ο χρόνος κατηγοριοποίησης τους. Για παράδειγμα για 1000 ταινίες μαζί με τους συντελεστές, τις εταιρίες και χώρες παραγωγής και τα είδη των ταινιών τους, έκανε 25 λεπτά για την ολοκλήρωση της κατηγοριοποίησης. Για 5000 ταινίες έκανε πάνω από 8 ώρες.

Ο κώδικας ξαναγράφτηκε (βλ σχήμα: 5.5) για να τρέχει με παραλληλισμό. Έτσι κάποια βασικά στοιχεία άλλαξαν, όπως τα HashMaps έγιναν ConcurrentHashMaps και κάποια άλλα κομμάτια έγιναν Synchronize για να υπάρχει Thread Safety. Το αποτέλεσμα αυτών των αλλαγών ήταν η κατηγοριοποίηση 8000 ταινιών, συνυπολογίζοντας και τα λοιπά σχετικά στοιχεία, να ολοκληρώνεται εντός 25 λεπτών.

```
1    categories = new ConcurrentHashMap<>();
2    categorizeData(Movie movie) { // ...
3        miSet.parallelStream().forEach // ...
4    }
5    Lists
6      // ...
7          chunk
8             .parallelStream().forEach // ...
```

Διάγραμμα 5.5: Παράλληλης κατηγοριοποίηση με ConcurrentHashMap

Στην πορεία εμφανίστηκε ένα πρόβλημα απόδοσης, το οποίο δεν ήταν ορατό κατά την πρώτη υλοποίηση, και το οποίο αφορούσε τα HashMaps. Ενώ δίνουν με μεγάλη ευκολία, είναι στην ουσία ένα Indexer για την αποθήκευση key-value pairs, έχουν μεγάλο performance-pentalty όταν αρχίζει και μεγαλώνει ο όγκος των ανατεθειμένων δεδομένων και το πρόβλημα μεγεθύνεται ακόμα πιο πολύ όταν είναι και ConcurrentHashMaps λόγω των ελέγχων που γίνονται για το Thread Safety. Ένα δείγμα 30000 ταινιών έκανε

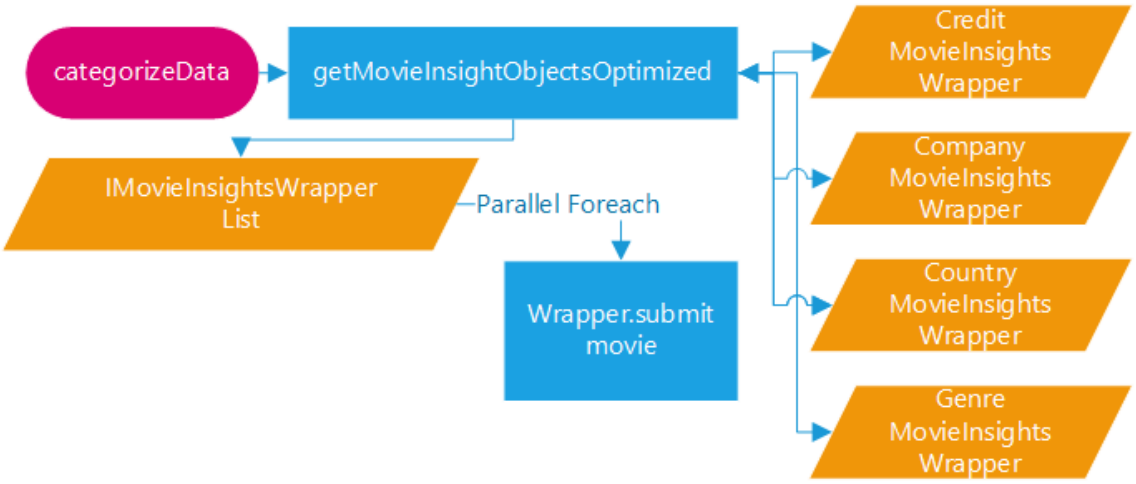
πάνω από 4.5 ώρες για να κατηγοριοποιήθει.

Κάτι έπρεπε να αντικαταστήσει τα HashMaps. Στην τελική έκδοση του αλγόριθμου (βλ σχήμα: 5.6) τα HashMaps αντικαταστάθηκαν με τα Native Arrays. Η επιλογή αυτή είχε ιδιαιτέρως θετικό αντίκτυπο στην ταχύτητα της κατηγοριοποίησης αλλά μεγάλωσε η πολυπλοκότητα διαχείρισης των δεδομένων. Συγκριτικά με τη χρήση ConcurrentHashMaps, όπου η κατηγοριοποίηση 30.000 ταινιών εκτελούνταν σε 4,5 ώρες, ο ανανεωμένος αλγόριθμος κατηγοριοποίησε 30000 ταινίες σε μόλις 8 λεπτά.

```
1   creditsArray = new
    ↪  MovieInsightsPersonWrapper[dto.maxPersonId][CreditRole.getSize()];
2   companiesArray = new MovieInsightsCompanyWrapper[dto.maxCompanyId];
3   countriesArray = new MovieInsightsCountryWrapper[dto.maxCountryId];
4   genresArray = new MovieInsightsGenreWrapper[dto.maxGenreId];
5
6   getMovieInsights(long id, BaseWrapper<?> o, IMovieInsightsWrapper[] array) {
7       synchronized (array) {
8           IMovieInsightsWrapper wrapper;
9           if ((wrapper = array[(int) id]) == null) {
10              array[(int) id] = wrapper = createMovieInsightsWrapper(o);
11          }
12          return wrapper;
13      }
14  }
```

Διάγραμμα 5.6: Παράλληλη κατηγοριοποίηση με Arrays

Ο αλγόριθμος λοιπόν κατηγοριοποιεί τα δεδομένα όπως φαίνεται στο σχήμα 5.7 και αφού τελειώσει προχωράει στο βήμα 2 του υπολογισμού των δεδομένων.



Διάγραμμα 5.7: Διάγραμμα ροής κατηγοριοποίησης δεδομένων

5.2.2 Βήμα 2 - Υπολογισμός δεδομένων

Επιλέγοντας την γλώσσα προγραμματισμού Java για τον υπολογισμό των δεδομένων και επικεντρώνοντας όλες τις βελτιστοποιήσεις στο κομμάτι τις απόδοσης, δημιουργήθηκε ένα νέο πρόβλημα. Επειδή υπάρχουν διπλότυπα δεδομένα λόγω της φύσης της δομής της πτυχιακής, η παρούσα υλοποίηση δεν είναι καθόλου βελτιστοποιημένη για την χρήση μνήμης RAM. Στο πέρας του πρώτο βήματος με δεδομένα 30000 ταινιών κατηγοριοποιημένα, η χρήση μνήμης ξεπερνάει τα 25 GB. Έχοντάς αυτό σαν δεδομένο ο αλγόριθμος υπολογισμού των δεδομένων εκτός από αποδοτικός ως προς την ταχύτητα θα έπρεπε να είναι τουλάχιστον και αποδοτικός ως προς την χρήση μνήμης.

Η αρχική προσέγγιση ήταν ο υπολογισμός των δεδομένων σε κάποια DTO με σκοπό ύστερα την δημιουργία Managed Entities του Hibernate, για να μπορέσουν να αποθηκευτούν στην βάση δεδομένων. Με αυτόν τον τρόπο όμως θα είχαμε ξανά το φαινόμενο των διπλότυπων και θα ξοδεύαμε ακόμα περισσότερη μνήμη καθώς η πλειοψηφία των δεδομένων είναι Native Ints, Longs και Doubles και όχι τόσο Reference Types. Η προσέγγιση που επιλέχτηκε ήταν η δημιουργία ενός Wrapper που κληρονομεί από το Maanaged Entity του Hibernate, κάνοντας και το Wrapper, Managed Entity. Με λίγες ρυθμίσεις πλέον το Wrapper μπορεί να σταλεί απευθείας στον Hibernate για να αποθηκευτεί στην βάση δεδομένων χωρίς να χρειαστεί η δημιουργία κάποιου άλλου Managed Entity.

Για αυτόν τον λόγο ο αλγόριθμος της επεξεργασίας δεδομένων μεταφέρθηκε μέσα σε αυτό το Wrapper έτσι ώστε να μπορεί να τρέξει μετά το πέρας της κατηγοριοποίησης αποδοτικά. Βλέποντας πίσω στο σχήμα(5.5) είναι εμφανές ότι αποστέλλεται μια ταινία στον Wrapper για περαιτέρω επεξεργασία.

Για να βελτιστοποιηθεί ακόμα περισσότερο στο βήμα της κατηγοριοποίησης ο Wrapper εκτελεί όσους υπολογισμούς μπορεί, που δεν χρειάζονται τα συνολικά δεδομένα για να υπολογιστούν, αλλά μόνο τα υπάρχοντα κάθε φορά σε κάθε submit.

Τα δεδομένα που υπολογίζονται σε κάθε submit είναι οι ταινίες με την μεγαλύτερη και χαμηλότερη βαθμολογία καθώς και η μέση βαθμολογία ταινιών για κάθε κατηγορία και οι ταινίες με τα μεγαλύτερα και χαμηλότερα έσοδα/έξοδα καθώς και τα μέσα έσοδα/έξοδα για κάθε κατηγορία. Τα υπόλοιπα δεδομένα υποβάλλονται και αποθηκεύονται σε HashMaps για τον τελικό υπολογισμό.

Ένα πρόβλημα που αντιμετωπίστηκε στον υπολογισμό της ταινίας με την μεγαλύτερη βαθμολογία ήταν η ακρίβεια της βαθμολογίας. Για παράδειγμα δεν είναι το ίδιο να υπάρχει μια ταινία με βαθμολογία 9.0 και 10.000 ψήφους και μια άλλη ταινία με βαθμολογία 9.0 και 200.000 βαθμολογίες. Όπως επίσης είναι πιο αξιόπιστη μια βαθμολογία για παράδειγμα 8.5 με 2.000.000 ψήφους από ότι μια βαθμολογία 9.0 με 100.000 ψήφους.

Για την επίλυση αυτού του προβλήματος χρησιμοποιήθηκε ένας πολύ απλός αλγόριθμος βάρους $rating * \log_2(voteCount)$, έτσι ώστε να υπολογίζονται και οι ψήφοι και να βγαίνει ενα ”σκορ” ταινίας το οποιο θα συγκρίνεται με τα άλλα ”σκορ” ταινιών έτσι ώστε να υπολογίζεται η με μεγαλύτερη ακρίβεια ποια ταινία είχε την μεγαλύτερη βαθμολογία.

```
1 double calculateScore(Vote vote, boolean high) {
2     return getScore(vote.getAverage(), vote.getVotes(), high, 2);
3 }
4 double getScore(double value, double weight, boolean high, int base) {
5     double weightLog = Math.log(weight) / Math.log(base);
6     return high ? value * weightLog : value / weightLog;
7 }
```

Διάγραμμα 5.8: Αλγόριθμος Βάρους Βαθμολογιών

Με τον παραπάνω αλγόριθμο σαν βάση, ο Αλγόριθμος υπολογισμού των ταινιών

με την μεγαλύτερη και χαμηλότερη βαθμολογία καθώς και της μέσης βαθμολογίας έχει διαμορφωθεί όπως φαίνεται στο σχήμα 5.9. Αρχικά γίνεται έλεγχος αν οι ψήφοι της βαθμολογίας της εν λόγω ταινίας είναι περισσότεροι ή ίσοι με την τιμή της σταθεράς *MINIMUM_VOTES_THRESHOLD*. Αν δεν είναι, η ταινία εξαιρείται από αυτόν τον υπολογισμό. Αλλιώς προστίθεται η βαθμολογία της ταινίας σε μια μεταβλητή που κρατάει το άθροισμά όλων των βαθμολογιών και αυξάνεται η μεταβλητή που κρατάει και πόσες ταινίες έχουν υποβάλει τις βαθμολογίες τους, για να υπολογιστεί αργότερα η μέση βαθμολογία ταινιών. Ύστερα αν δεν υπάρχει ήδη ταινία με την μεγαλύτερη η χαμηλότερη βαθμολογία τοποθετείται αυτή για την οποία γίνεται ο έλεγχος αυτήν την χρονική στιγμή, αλλιώς γίνεται έλεγχος αν αυτή η ταινία έχει μεγαλύτερη / μικρότερη βαθμολογία από την ανάλογη υπάρχουσα και ορίζεται αυτή στην θέση της σε περίπτωση που πληροί της προϋποθέσεις.

```
1  void submitRating(Movie movie) {
2      if (movie.getVote().getVotes() >= Constants.MINIMUM_VOTES_THRESHOLD) {
3          synchronized (voteLock) {
4              Vote vote = movie.getVote();
5              totalVoteAverage += vote.getAverage();
6              setTotalRatedMovies(getTotalRatedMovies() + 1);
7              if (getHighestRatedMovie() == null) {
8                  setHighestRatedMovie(movie);
9              } else {
10                 setHighestRatedMovie(calculateVote(calculateScore(
                       getHighestRatedMovie().getVote(), true),
                       getHighestRatedMovie(),calculateScore(movie.getVote(),
                       true),movie,(current, challenger) -> current < challenger));
11             }
12             if (getLowestRatedMovie() == null) {
13                 setLowestRatedMovie(movie);
14             } else {
15                 setLowestRatedMovie(calculateVote( getLowestRatedMovie().getVote().
                       getAverage(),getLowestRatedMovie(),
                       movie.getVote().getAverage(),movie,(current, challenger) ->
                       current > challenger));
16             }
17         }
18     }
19 }
```

Διάγραμμα 5.9: Αλγόριθμος Υπολογισμού Βαθμολογιών

Ο υπολογισμοί των ταινιών με τα μεγαλύτερα, μικρότερα και μέσα έσοδα / έξοδα γίνεται με έναν πολύ παρόμοιο τρόπο όπως των βαθμολογιών των ταινιών, χωρίς βέβαια να χρησιμοποιείται ο αλγόριθμος βάρους.

Ο υπολογισμός των υπόλοιπων δεδομένων γίνεται μετά το πέρας του βήματος της κατηγοριοποίησης. Για να γίνει πιο εύκολος ο υπολογισμός, αντί να υποβάλλονται τα δεδομένα ως έχουν έχουν δημιουργηθεί Wrappers γύρο από αυτά για την προσθήκη επιπρόσθετων δεδομένων και λειτουργιών που καθιστούν τον υπολογισμό των δεδομένων πιο εύκολο. Για παράδειγμα ένα αντικείμενο τύπου Genre έχει το ανάλογο GenreWrapper, όπως επίσης και ένα αντικείμενο τύπου Person έχει ανάλογα ActorWrapper, DirectorWrapper, ProducerWrapper και WritterWrapper για κάθε κατηγορία ατόμου που υποστηρίζεται από την εφαρμογή. Σε αυτά τα Wrappers υπάρχουν δεδομένα όπως ο αριθμός ταινιών που συμμετέχουν αυτά τα Wrappers για αυτήν την κατηγορία, οι ταινίες οι ίδιες, και υπάρχουν και λειτουργίες σύγκρισης (Comparators) μεταξύ ίδιων Wrappers χρησιμοποιώντας αυτά τα παραπάνω δεδομένα.

Όταν γίνεται η υποβολή δημιουργείται αρχικά ένα Wrapper προσθέτοντάς μεταδεδομένα χωρίς να ελεγχθεί αν υπάρχει ήδη Wrapper που να αναφέρεται στο ίδιο αντικείμενο, αποθηκευμένο στο ανάλογο HashMap οπως φαίνεται στο σχήμα 5.10, στην σειρά 5 για την υποβολή δεδομένων ειδών ταινιών.

```
1 void submitGenres(Movie movie) {
2     AtomicBoolean hasIncreased = new AtomicBoolean(false);
3     movie.getGenres().parallelStream().filter(Objects::nonNull).forEach(genre -> {
4         calculateTotals(genreTotals, hasIncreased);
5         submit(new GenreWrapper(genre, processor.getMovieCount(genre)), genres, movie,
          ↪  genre);
6     });
7 }
```

Διάγραμμα 5.10: Αλγόριθμος Υποβολής Δεδομένων Είδους Ταινίας

Στην γενικευμένη συνάρτηση submit πάραυτα γίνεται εκεί ο έλεγχος για την ύπαρξη του ανάλογου Wrapper, και αν υπάρχει απλά αγνοεί το νέο Wrapper και αλλάζει το υπάρχων χρησιμοποιώντας τα στοιχεία του νεου, αλλιώς τοποθετεί το δημιουργημένο Wrapper στο ανάλογο HashMap όπως φαίνεται στο σχήμα 5.11.

Επιπρόσθετα για τον υπολογισμό αυτών των δεδομένων χρησιμοποιείται μια βοη-

```
1 <T extends IdentifiedEntity, W extends BaseWrapper<T>> void submit(W wrapper,
  ↪  Map<Long, W> objMap, Movie movie, T obj) {
2     W wrapper2;
3     synchronized (objMap) {
4         if ((wrapper2 = objMap.putIfAbsent(obj.getId(), wrapper)) == null) {
5             wrapper2 = wrapper;
6         }
7     }
8     wrapper2.movies.add(movie);
9     wrapper2.count++;
10 }
```

Διάγραμμα 5.11: Αλγόριθμος Γενικευμένης υποβολής δεδομένων

θητική γενικευμένη κλάση Total, η οποία κρατάει τα στοιχεία για τα μέγιστα, ελάχιστα και μέσα αυτών των δεδομένων.

Μετά το πέρας της συλλογής και της κατηγοριοποίησης των δεδομένων, το κάθε MovieInsights Wrapper ξεχωριστά θα υπολογίσει τα υπόλοιπα δεδομένα για το ίδιο αλλά και για κάθε εξαρτώμενο Wrapper από αυτό, που στην συγκεκριμένη περίπτωση τα εξαρτώμενα Wrappers, είναι οι κατηγορίες ανά χρόνο.

Για κάθε διαφορετικό Wrapper HashMap θα υπολογιστούν τα μέγιστα και τα ελάχιστα χρησιμοποιώντας τις συναρτήσεις σύγκρισης που υπάρχουν μέσα στα Wrappers, και τα δεδομένα που περιέχουν τα μέγιστα και τα ελάχιστα θα υποβληθούν στα αντικείμενα της βοηθητικής κλάσης Total για την ανάκτηση τους αργότερα, όπως φαίνεται στα σχήματα 5.12 και 5.13.

```
1 <W extends BaseWrapper<WE>, WE> void calculateChildInsights(Map<Long, W> map,
  ↪  Comparator<? super W> comparator, Total<WE> totals) {
2     Optional<W> mostPopularEntryResult = getWrapperBasedOnComparator(map,
      ↪  comparator.reversed());
3     mostPopularEntryResult.ifPresent(totals::submitMostPopular);
4     Optional<W> leastPopularEntryResult = getWrapperBasedOnComparator(map,
      ↪  comparator);
5     leastPopularEntryResult.ifPresent(totals::submitLeastPopular);
6     totals.setTotalEntities(map.size());
7 }
```

Διάγραμμα 5.12: Γενικευμένος Αλγόριθμος υπολογισμού μέγιστων και ελάχιστων δεδομένων

```
1 private <W extends BaseWrapper<EW>, EW> Optional<W>
  ↪ getWrapperBasedOnComparator(Map<Long, W> wrapperMap, Comparator<? super W>
  ↪ comparator) {
2     return wrapperMap.values().parallelStream().sorted(comparator).filter(e -> (slave
      ↪ ? master.getCategory() : getCategory()) != e.category || e.object != (slave ?
      ↪ master.getSource().object : source.object)).findFirst();
3 }
```

Διάγραμμα 5.13: Γενικευμένος Αλγόριθμος ανάκτησης Wrapper απο ενα HashMap με βάση αλγόριθμο σύγκρισης

Τέλος όλα τα δεδομένα αποθηκεύονται στα πεδία της κληρονομουμένης κλάσης MovieInsights, ανακτώντας τα δεδομένα από τα αντικείμενα της βοηθητικής κλάσης Total.

5.2.3 Βήμα 3 - Αποθήκευση δεδομένων

Έχοντάς συλλέξει, κατηγοριοποιήσει και υπολογίσει όλα τα απαραίτητα δεδομένα, τα δεδομένα αυτά με ελάχιστες μετατροπές στέλνονται στον Hibernate για να αποθηκευτούν στην βάση δεδομένων έτσι ώστε να τεθεί η εφαρμογή σε κατάσταση ετοιμότητας.

Για την αποθήκευση δεδομένων δημιουργείται μια νέα ”συναλλαγή”. Η συναλλαγή αυτή βεβαιώνει ότι αν κάποιο από τα δεδομένα αυτά δεν μπορούσε για οποιονδήποτε λόγω να αποθηκευτεί στην βάση δεδομένων, να ακυρωθεί όλη αυτή η διαδικασία και να τερματίσει η εφαρμογή με το ανάλογο μήνυμα σφάλματος, έτσι ώστε να μπορεί να επιλυθεί απο κάποιον διαχειριστή. Η αποθήκευση μερικών δεδομένων στην βάση δεδομένων θα είχε καταστροφικές συνέπειες στην λειτουργία της εφαρμογής.

Μετά το πέρας της αποθήκευσης, γίνεται εκκαθάριση του Cache και ζητείται να γίνει και εκκαθάριση μνήμης από τον Garbage Collector το JVM, ώστε να ελευθερωθεί μνήμη για την καλύτερη λειτουργία της εφαρμογής. Πολλές φορές αυτό δεν είναι δυνατό ανάλογα με το JVM που χρησιμοποιείται και έτσι μετά την αρχικοποίηση στην περίπτωση που δεν υπάρχει διαθέσιμη μνήμη συστήνεται η επανεκκίνηση της εφαρμογής.

5.3 Μηχανή Αναζήτησης

Η μηχανή αναζήτησης είναι μια σημαντική υπηρεσία για την εφαρμογή και επιτρέπει στον χρήστη την αναζήτηση και αυτόματη συμπλήρωση της αναζήτησης συντελεστών, εταιριών και χωρών παραγωγής και ειδών ταινιών.

Η μηχανή αναζήτησης έχει την δυνατότητα να συνδεθεί σε μια εξωτερική βάση δεδομένων για την ανάκτηση δεδομένων είτε να χρησιμοποιήσει την εσωτερική της βάση δεδομένων. Για την βελτιστοποίηση της εμπειρίας του χρήστη η μηχανή αναζήτησης χρησιμοποιεί την δικιά της, εσωτερική, βάση δεδομένων, καθώς είναι πολύ πιο γρήγορη από μια συμβατική σχεσιακή SQL βάση δεδομένων, με το μειονέκτημα της διπλοτυπίας δεδομένων.

Για περαιτέρω βελτιστοποίηση δημιουργήθηκαν ξεχωριστά Indices για την μηχανή αναζήτησης και ξεχωριστά Entities για την βάση δεδομένων. Έχοντάς ως κύρια πηγή της αλήθειας τα Entities της βάσης δεδομένων SQL, τα Indicies της μηχανής αναζήτησης έχουν τα λιγότερα δυνατά δεδομένα που χρειάζονται για την αναζήτηση, μειώνοντας σημαντικά τον απαιτούμενο χώρο αποθήκευσης.

Καθώς όμως τα δεδομένα δεν βρίσκονται σε μια μόνο βάση δεδομένων, χρειάζεται ο συγχρονισμός τους καθ όλη την λειτουργία της εφαρμογής. Αυτό γίνεται εύκολα καθώς τα δεδομένα που χρησιμοποιούνται στην αναζήτηση, αλλάζουν μόνο όταν τρέχει το σύστημα εισαγωγής δεδομένων και αυτό γίνεται μόνο στην εκκίνηση της εφαρμογής.

Πάραυτα καθώς η μηχανή αναζήτησης είναι ένα εξωτερικό σύστημα, όχι άμεσα επιβλεπόμενο από την εφαρμογή, τα δεδομένα της βάσης δεδομένων της μηχανής αναζήτησης, μπορούν να αλλάξουν και να αλλοιωθούν ανά πάσα στιγμή. Για αυτόν τον λόγο έχει δημιουργηθεί μια υπηρεσία που οι αποκλειστικές αρμοδιότητες της είναι να συγχρονίζει τα δεδομένα της κύριας βάσης δεδομένων με την βάση δεδομένων της μηχανής αναζήτησης. Αυτή η υπηρεσία ενεργεί μια φορά στην εκκίνηση ακριβώς πριν τεθεί η εφαρμογή σε κατάσταση ετοιμότητας, και ενεργεί όταν ένας διαχειριστής το ζητήσει.

Η Μηχανή Αναζήτησης πέρα από τον συγχρονισμό δεδομένων επιτρέπει τον χρήστη να αναζητήσει κάτι, αλλά παράλληλα επιτρέπει και την αυτόματη συμπλήρωση. Για να λειτουργήσει η Αυτόματη συμπλήρωση όλα τα Indices έπρεπέ να ρυθμιστούν

ανάλογα.

```
1 {"analysis": {
2   "filter": {
3     "autocomplete_filter": {
4       "type": "edge_ngram",
5       "min_gram": 1,
6       "max_gram": 40
7     }
8   },
9   "analyzer": {
10    "autocomplete_search": {
11      "type": "custom",
12      "tokenizer": "standard",
13      "filter": ["lowercase"]
14    },
15    "autocomplete_index": {
16      "type": "custom",
17      "tokenizer": "standard",
18      "filter": ["lowercase", "autocomplete_filter"]
19    }
20  }
21 }}
```

Διάγραμμα 5.14: Ρυθμίσεις ενός Index της ElasticSearch

Για να λειτουργήσει η αυτόματη συμπλήρωση κάθε λέξη και πρόταση μέσα σε αυτά τα Indices έπρεπε να χωριστεί σε ngrams. Όπως φαίνεται στο σχήμα 5.14, στην σειρά 4, δηλώθηκε ενα φίλτρο το οποίο σπάει τις λέξεις και προτάσεις σε ngrams και έτσι επιτρέπει την ρύθμιση ενός πεδίου ενός Index για να μπορεί να αναζητηθεί με αυτόματη συμπλήρωση.

Το επόμενο βήμα ήταν να δηλωθούν τα πεδία στα Indices που θα έχουν τα φίλτρα του autocomplete, όπως φαίνεται στο σχήμα 5.15

Με αυτόν τον τρόπο δηλώνονται τα Indices στον κώδικα με τις σωστές ρυθμίσεις, και η βιβλιοθήκη που χρησιμοποιήθηκε για να επικοινωνήσει με την ElasticSearch, Jest, αρχικοποιεί τα Indices στην ElasticSearch και αφού προστεθούν δεδομένα από τον συγχρονισμό των 2 βάσεων, η μηχανή αναζήτησης είναι έτοιμη για χρήση.

Καθώς όπως προαναφέρθηκε η ElasticSearch είναι ένα αυτόνομο σύστημα πρέπει να του δοθεί ένα ανάλογο ερώτημα (query) που να μπορεί να το καταλάβει για να μπορέσει διαθέσει τα ανάλογα τα δεδομένα. Αυτό το κάνει η υπηρεσία SearchService.

```
@MultiField(
    mainField = @Field(
        type = Text,
        fielddata = true,
        analyzer = "autocomplete_index",
        searchAnalyzer = "autocomplete_search"),
    otherFields = {@InnerField(suffix = "verbatim", type = Keyword)})
private String name;
```

Διάγραμμα 5.15: Annotations ενός Index της ElasticSearch

Όλα τα Entities της εφαρμογής έχουν το ανάλογο Service για να επιτρέψουν την ανάκτηση, μορφοποίησή, δημιουργία αλλά και διαγραφή των δεδομένων τους, γνωστό ως CRUD. Τα Entities που έχουν ανάλογα Indicies μοιράζονται τα ίδια Services για λόγους απλότητας και συγκέντρωσης των αρμοδιοτήτων σε ένα σημείο.

Κάθε υπηρεσία ενός Entity που έχει παράλληλα και Index, κάνει εγγραφή στο SearchService κατά την εκκίνηση, όπως φαίνεται στο σχήμα 5.16, έτσι ώστε όταν φτάσει η στιγμή που θα γίνει ένα ερώτημα αναζήτησης, το SearchService να γνωρίζει από ποια Indices θα παρθούν τα δεδομένα. Το κάθε Service που γίνεται register ορίζει επίσης το όνομα του Index για το οποίο θα κατασκευάσει ένα ερώτημα το SearchService αλλά και ένα QueryConfiguration που επιτρέπει την παραμετροποίηση του ερωτήματος που θα δημιουργηθεί όπως φαίνεται στο σχήμα 5.17. Με αυτόν τον τρόπο η μηχανή αναζήτησης δεν χρειάζεται να γνωρίζει τίποτα για τα Indices πέρα από το όνομα τους, απλά παίρνει τα προκαθορισμένα ερωτήματα και τις επιπρόσθετες ρυθμίσεις τους από όλα τα Services που δηλώνουν Indices, τα συνδιάζει και τα στέλνει στην ElasticSearch. Όταν πάρει τα δεδομένα, τα αλλάζει μορφή και τα στέλνει στον Client σε μια διαφορετική μορφή για την βελτιστοποίηση του μεγέθους της απάντησης.

```
@PostConstruct
public void init() {
    searchService.register(esEntityClass, this);
}
```

Διάγραμμα 5.16: Κώδικας εγγραφής ενός Service στο SearchService

Όταν γίνεται η αυτόματη συμπλήρωση, για κάθε γράμμα που πληκτρολογείται

```
@Override
protected QueryConfiguration queryConfiguration() {
    QueryConfiguration q = QueryConfiguration.CreateDefault();
    q.setBoost(0.8f);
    return q;
}
```

Διάγραμμα 5.17: Κώδικας παραμετροποίησης ερωτήματος μηχανής αναζήτησης

στον Client, γίνεται και ένα ερώτημα στον Server και στην Μηχανή Αναζήτησης. Αντί να στέλνονται τα ερωτήματα απευθείας στην μηχανή αναζήτησης από τον Client, χρησιμοποιείται ο Server σαν διαμεσολαβητής για δύο σημαντικούς λόγους. Ο πρώτος είναι για να διασφαλιστεί η ασφάλεια της υπηρεσίας και ο δεύτερος είναι για να μπορεί ο Server να μορφοποιεί το αποτέλεσμα πριν σταλεί στον Client. Είναι πολύ σημαντικό και η μηχανή αναζήτησης να σερβίρει τα δεδομένα πολύ γρήγορα, αλλά και ο Server να απαντάει πολύ γρήγορα. Πέρα από την ταχύτητα που θα απαντάει ο Server, πρέπει επίσης η απάντηση να είναι μικρή έτσι ώστε πρώτον να εξοικονομεί δεδομένα από Clients οι οποίοι βρίσκονται σε Metered Connections (συνδέσεις οι οποίες χρεώνονται ανά MB), και να στέλνονται πιο γρήγορα τα αποτελέσματα σε Clients οι οποίοι έχουν χαμηλή ταχύτητα Internet.

Για παράδειγμα το αποτέλεσμα της αναζήτησης που επιστρέφει η μηχανή αναζήτησης όπως φαίνεται στο σχήμα 5.18 έχει μέγεθος 420 bytes για ένα αποτέλεσμα με μέγεθος αποτελέσματος 232 bytes, σε αντίθεση με το αποτέλεσμα του SearchService όπως φαίνεται στο σχήμα 5.19 με μέγεθος 166 bytes, για ένα αποτέλεσμα με μέγεθος αποτελέσματος 123 bytes. Είναι εμφανές λοιπόν ότι με την βελτιστοποίηση το μέγεθος της απάντησης είναι πολύ μικρότερο.

```
{
    "took": 23,
    "timed_out": false,
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": 37470,
        "max_score": 846.1094,
        "hits": [{
            "_index": "person",
            "_type": "person",
            "_id": "323",
            "_score": 846.1094,
            "_source": {
                "popularity": 35.808,
                "name": "Some Person",
                "profilePath": "/cckcYc2v0yh1tc9QjRelptcOBko.jpg",
                "id": 323
            }
        }]
    }
}
```

Διάγραμμα 5.18: Αποτέλεσμα αναζήτησης ElasticSearch

```
{
    "_": [{
        "e": [{
            "id": 323,
            "name": "Some Person",
            "popularity": 35.808,
            "profilePath": "/cckcYc2v0yh1tc9QjRelptcOBko.jpg"
        }],
        "i": "PERSON"
    }]
}
```

Διάγραμμα 5.19: Αποτέλεσμα Αναζήτησης SearchService

5.4 Client

Πέρα από τον Server όπως προαναφέρθηκε, το 2ο μέρος της εφαρμογής είναι ο Client. Ο Client είναι το σύστημα που θα καταναλώσει τα δεδομένα από τον Server και θα τα εμφανίσει με έναν προσιτό και κατανοητό τρόπο στον χρήστη. Ο Client δομήθηκε με την χρήση της βιβλιοθήκης React του CoreUI και του Redux Framework.

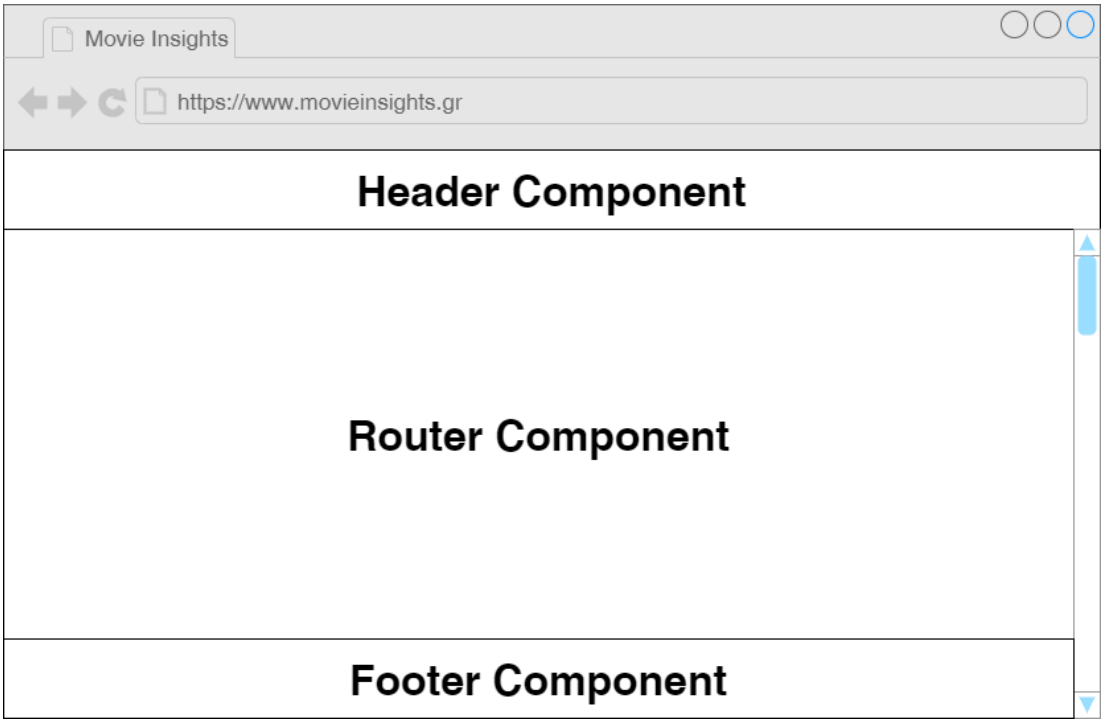
Η React είναι η κεντρική βιβλιοθήκη που χρησιμοποιήθηκε και όλα τα άλλα συστήματα αλληλεπιδρούν με αυτήν. Το Redux χρησιμοποιήθηκε για την διαχείρισή της κατάστασης της εφαρμογής και το CoreUI για τα γραφικά της. Πέρα από αυτές τις κύριες τεχνολογίες χρησιμοποιήθηκε επίσης η βιβλιοθήκη Highcharts για τα γραφήματα, τα εικονίδια της βιβλιοθήκης CoreUI και FontAwesome, η βιβλιοθήκη lodash για την ευκολότερη διαχείρισή των Javascript Objects, και κάποιες άλλες μικρότερες βιβλιοθήκες για την βελτίωση της αισθητικής του γραφικού περιβάλλοντος και την ευκολότερη διαχείριση των συστημάτων του Frontend.

Ο Client στην ουσία είναι μια Web εφαρμογή και αποτελείται από 2 κομμάτια. Το πρώτο είναι το Dashboard, το οποίο εμφανίζει όλα τα δεδομένα που προσφέρει ο Server. Το δεύτερο κομμάτι είναι η σελίδα παρακολούθησης των στατιστικών των υπηρεσιών. Η αρχιτεκτονική του Client είναι SPA, που σημαίνει ότι υπάρχει μόνο μια σελίδα και το περιεχόμενο της σελίδας αλλάζει με την React.

Η React λειτουργεί με Components. Component είναι ένα στοιχείο το οποίο εμφανίζεται κάπου μέσα στην Web εφαρμογή. Η Συνολική εφαρμογή αποτελείται από πολλά Components τα οποία συνδυάζονται και εμφανίζονται μαζί για να φανεί το τελικό αποτέλεσμα. Τα Component είναι ένας έξυπνος τρόπος δόμησης μιας Web Εφαρμογής καθώς μπορούν να ξανά χρησιμοποιηθούν σε πολλά μέρη της εφαρμογής. Με αυτόν τον τρόπο έχοντας ένα αρχείο για κάθε Component γίνεται πιο ξεκάθαρη η δομή της εφαρμογής και ο τρόπος που λειτουργεί το κάθε Component, και καθίσταται ευκολότερη η συντήρηση και αλλαγή κώδικα αλλά και η αντιμετώπιση σφαλμάτων.

Η Βασική δομή αποτελείται από το Header Component, το Router Component και το Footer Component όπως φαίνεται στο σχήμα 5.20. Το Header αποτελείται από μια εικόνα, το λογότυπο της εφαρμογής, στα αριστερά και ένα εικονίδιο με εικόνα ένα γρανάζι στα δεξιά το οποίο επιτρέπει την αλλαγή γλώσσας, την είσοδο στην εφαρμογή και σε περίπτωση που ο χρήστης είναι ήδη εξουσιοδοτημένος σαν διαχειριστής, έναν σύν-

δεσμο στο πάνελ διαχείρισης. Το Footer αναγράφει την ημερομηνία κατασκευής της εφαρμογής καθώς και το όνομα του δημιουργού στα αριστερά, και στα δεξιά αναγράφει τις κύριες τεχνολογίες που χρησιμοποιήθηκαν για να γίνει η εφαρμογή. Το Router από την άλλη είναι μια περιοχή η οποία θα αλλάζει ανάλογα με το τι χρειάζεται να εμφανιστεί κάθε φορά. Τα περιεχόμενο που αλλάζουν μέσα στο Router μπορούν να χαρακτηρησούν σαν σελίδες. Οποιαδήποτε αναφορά σε ”σελίδες” αργότερα, έκτος αν διευκρινιστεί κάτι άλλο, θα αφορά τις ”εικονικές” σελίδες που αλλάζει το Router.



Διάγραμμα 5.20: Βασική διάταξη Web εφαρμογής

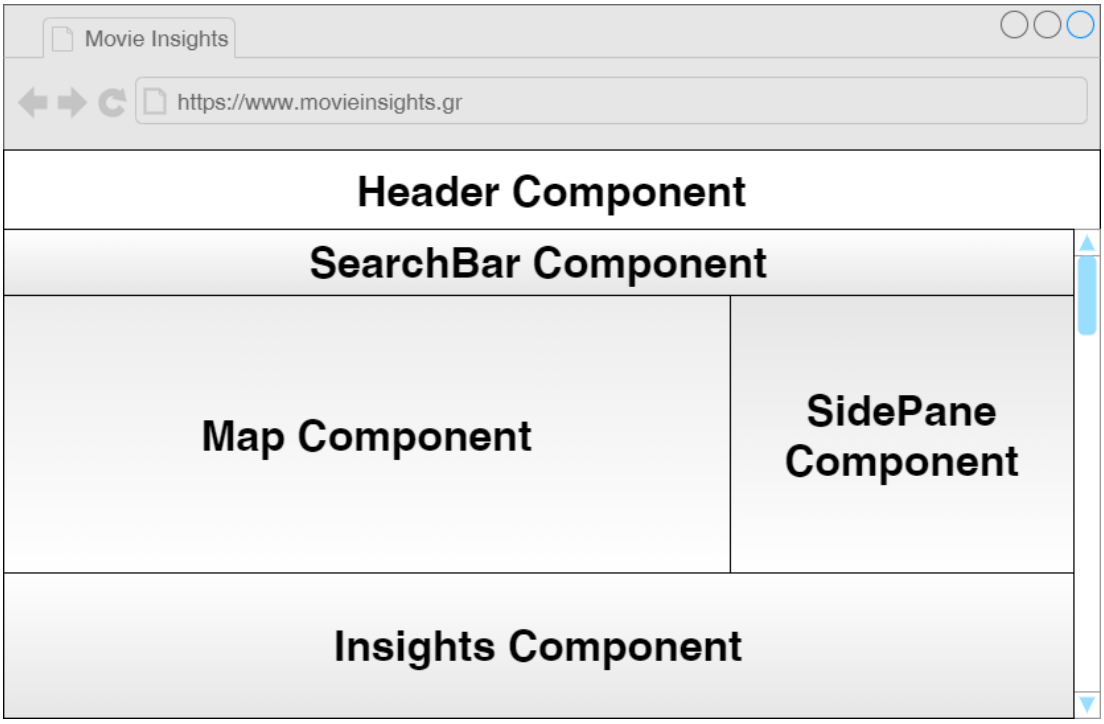
Η εφαρμογή αυτή επίσης έχει Modules. Η έννοια Module στην React δεν υπάρχει σε αντίθεση με κάποιο άλλο Web Framework όπως η Angular, αλλά για αυτήν την εφαρμογή στην ουσία ορίζει μια λειτουργία και ομαδοποιεί όλα τα Components αυτής της λειτουργίας μαζί.

Η αρχιτεκτονική του Router είναι δυναμική καθώς επιτρέπει κάθε Module να ορίζει το δικό της Router με τα δικά του Routes. Υπάρχουν 4 ειδών Routes. Το πρώτο είναι το απλό Route που όταν αλλάζει η διεύθυνση στην γραμμή εισαγωγής διεύθυνσης σελίδας αλλάζει και το περιεχόμενο που συνήθως κάθε route είναι και μια σελίδα. Το δεύτερο ονομάζεται PrivateRoute και επιτρέπει την πρόσβαση στην δηλωμένη διεύθυνση μόνο αν ο χρήστης έχει κάνει Login, και είναι εξουσιοδοτημένος για να δει την

σελίδα που ζητάει. Το τρίτο Route είναι ένα RedirectRoute που όταν πηγαίνει ο χρήστης εκεί τον ανακατευθύνει σε ένα αλλο Route, και το τέταρτο Route είναι ένα Route που ενεργοποιείται όταν η διεύθυνση που έχει δοθεί δεν ανταποκρίνεται σε κανένα δηλωμένο Route, και συνήθως σε αυτό το Route μπαίνει η σελίδα 404 Not Found.

5.4.1 Dashboard Module

Το Dashboard είναι το κεντρικό Module της εφαρμογής, που φαίνεται και στην αρχική σελίδα και εμφανίζει μέσω του MIDashboard Component 4 διαφορετικά κομμάτια. Το πρώτο κομμάτι είναι η μπάρα αναζήτησης, το δεύτερο κομμάτι είναι ένας παγκόσμιος χάρτης, το τρίτο κομμάτι είναι στα δεξιά του χάρτη και αποτελείται από γραφήματα και συγκεντρωτικά στοιχεία και το τέταρτο και τελευταίο κομμάτι είναι τα κύρια δεδομένα της εφαρμογής όπως ο πιο δημοφιλής ηθοποιός, η ή ταινία με τα μεγαλύτερα έσοδα κ.λ.π όπως φαίνεται στο σχήμα 5.21.



Διάγραμμα 5.21: Βασική διάταξη Web εφαρμογής

Το Dashboard Module ορίζει το δικό του Router που αποτελείται από ένα Route /app/* και ένα RedirectRoute από το κεντρικό Route / στο /app το οποίο είναι η κεντρική σελίδα καθώς δεν χρειάζεται κάτι σύνθετο.

Ο τρόπος που λειτουργεί είναι ο εξής. Είναι επιλεγμένη για παράδειγμα μια κατηγορία ”Στοιχεία για την χώρα Ελλάδα”. Σε αυτήν την κατηγορία στα δεδομένα υπάρχουν οι πιο δημοφιλείς ηθοποιοί, παραγωγοί, σκηνοθέτες, συγγραφείς που έχουν συμμετάσχει σε ταινίες που είτε η παρήγαγε η Ελλάδα είτε η Ελλάδα ήταν χώρα συμπαραγωγής. Αναφέρει επίσης την πιο δημοφιλή εταιρία παραγωγής που συνεργάστηκε η Ελλάδα αλλά και την πιο δημοφιλή χώρα συμπαραγωγής και είδος ταινίας. Επιπρόσθετα αναφέρει τα μέσα και συνολικά έσοδα/έξοδα καθώς και το καθαρό κέρδος. Όλα αυτά τα στοιχεία υπάρχουν επίσης και ανά χρόνο, για όσα χρόνια υπάρχουν στην βάση δεδομένων για την Ελλάδα. Όλες οι κατηγορίες λειτουργούν με τον ίδιο τρόπο με την εξαίρεση της κατηγορίας ανά άτομο. Που εκεί εκτός από την επιλογή χρόνου, υπάρχει και η επιλογή ρόλου, με διαθέσιμους ρόλους: ηθοποιός, σκηνοθέτης, συγγραφέας και παραγωγός. Για τα άτομα υπάρχουν συγκεντρωτικά στοιχεία μόνο ανά ρόλο και όχι συνολικά, καθώς είναι διαφορετικά τα δεδομένα. Πάραυτα όλα τα στοιχεία ανά ρόλο υπάρχουν και ανά χρόνο.

Το Dashboard Module περιέχει μόνο ένα Component το MIDashboard Component. Όλα τα δεδομένα που υπολογίζονται μέσα από το Dashboard Module στέλνονται απευθείας στο MIDashboard Component.

Μέσω των Component που δηλώνονται και δομείται όλη η εφαρμογή καθίσταται εφικτή η πλοήγηση σε όλο το εύρος των δεδομένων της εφαρμογής. Ενώ θα μπορούσε σε κάθε Component, που επηρεάζει τα δεδομένα που εμφανίζονται, να υλοποιηθεί η λογική της αλλαγής των δεδομένων, προτιμήθηκε όλα αυτά τα Component απλά να αλλάζουν την διεύθυνση στην γραμμή διεύθυνσης του Browser και με βάση την διεύθυνση που άλλαξε χωρίς να γίνεται ανακατεύθυνση το Dashboard Module καταλαβαίνει τι χρειάζεται να αλλάξει, αλλάζει τα δεδομένα μέσω του Redux State Manager, και ύστερα στέλνει τα δεδομένα σε όλα τα επιμέρους Components για να αλλάξουν. Ενώ ακούγεται πιο σύνθετη τεχνική, χρησιμοποιήθηκε για την ευκολία διαχείρισης και συντήρησης του κώδικα, καθώς στην άλλη περίπτωση θα έπρεπε σε περίπτωση αλλαγής του κώδικα εμφάνισης των δεδομένων, να πραγματοποιηθούν αλλαγές σε κάθε Component που άλλαζε δεδομένα. Ενώ με αυτήν την υλοποίηση τα πάντα βρίσκονται σε ένα μέρος. Ένας ακόμα σημαντικός λόγος που προτιμήθηκε η συγκεκριμένη τεχνική είναι η προσβασιμότητα στα δεδομένα αυτά. Λόγω της φύσης της αρχιτεκτονικής της εφαρμογής και όντας SPA, υπάρχει ουσιαστικά μόνο μια αληθινή σελίδα που είναι προ-

σβάσιμη από έναν Browser. Αυτή είναι η index.html που είναι ένα αρχείο HTML το οποίο φορτώνει όλον τον κώδικα της εφαρμογής. Χωρίς την αλλαγή της διεύθυνσης, ένας χρήστης θα έπρεπε να μπει στην αρχική σελίδα και να εκτελέσει κάποιες ενέργειες έτσι ώστε να φτάσει στο ίδιο σημείο που βρισκόταν πριν. Με την αλλαγή της διεύθυνσης όμως, όχι μόνο μπορεί να επανέλθει στο σημείο στο οποίο βρισκόταν αλλά μπορεί και να μοιραστεί τον σύνδεσμο με κάποιον άλλον χρήστη και να κοιτάνε τα ίδια δεδομένα.

Το Dashboard Module λοιπόν παρακολουθεί τις αλλαγές διευθύνσεων που γίνονται μετά την βασική διεύθυνση /app/ και με την χρήση κάποιον βοηθητικών συναρτήσεων αλλάζει τα δεδομένα όπως φαίνεται στον κώδικα 5.22.

```
handleViewChange = () => {
    const path = this.props.location.pathname;
    if (this.state.path !== path  !this.state.pathHandled) {
      if (!this.state.pathHandled) {
        let pathMatch: match;
        if ((pathMatch = matchPath(path,
        ↪  "/app/:entity(country|company|genre)/:id-:name/:year(\\d{4})?"))) {
          this.handleGenericChange(pathMatch.params['entity'],
          ↪  +pathMatch.params['id'], +pathMatch.params['year']);
        } else if ((pathMatch = matchPath(path,
        ↪  "/app/person/:id-:name/:role([A-z]+)?/:year(\\d{4})?"))) {
          this.handlePersonChange(+pathMatch.params['id'], +pathMatch.params['year'],
          ↪  pathMatch.params['role']);
        } else if ((pathMatch = matchPath(path,
        ↪  "/app/:general(general)?/:year(\\d{4})?"))) {
          this.handleGeneralChange(+pathMatch.params['year']);
        }
        this.scrollElement();
        if (this.state.path !== path)
          this.setState({path});
      }
      if (this.state.path !== path) {
        this.setState({pathHandled: false});
      }
    }
}
```

Διάγραμμα 5.22: Αλγόριθμος παρακολούθησης αλλαγής στην γραμμή διευθύνσεων ενός Browser.

Η αλλαγή της διεύθυνσης στην γραμμή διευθύνσεων του Browser, δημιουργείται

από μια βοηθητική συνάρτηση όπως φαίνεται στον κώδικα 5.23. Με αυτόν τον τρόπο όλα τα Components που έχουν ως σκοπό την αλλαγή των δεδομένων μπορούν να αλλάξουν εύκολα την διεύθυνση χρησιμοποιώντας αυτήν την συνάρτηση, και αν αλλάξει ποτέ η δομή της διεύθυνσης η αλλαγή θα γίνει μόνο μέσα σε αυτήν την συνάρτηση διευκολύνοντας παράλληλα την συντήρηση και την αντιμετώπιση προβλημάτων του κώδικα.

```
function generateNavigationLink(entity?: BaseNamedEntity, role?: CreditRole, year?: number, entityType?: EntityType) {
  let type = undefined;
  if (entity) {
    if (entityType) {
      type = entityType.toLowerCase();
    } else if (isMovie(entity)) {
      type = 'movie';
    } else if (isPerson(entity)) {
      type = 'person';
    } else if (isCompany(entity)) {
      type = 'company';
    } else if (isCountry(entity)) {
      type = 'country';
    } else if (isGenre(entity)) {
      type = 'genre';
    }
  } else if (year) {
    type = 'general';
  }
  return `/app${type ? `/${type}` : ''}${entity ? `/${entity.id}-${normalizeText(entity.name)}` : ''}${role && type === 'person' ? `/${role}` : ''}${year ? `/${year}` : ''}`;
}
```

Διάγραμμα 5.23: Αλγόριθμος δημιουργίας διεύθυνσης αλλαγής δεδομένων.

Το Redux State Manager χρησιμοποιήθηκε για 2 λόγους. Προσφέρει μια εύκολη διεπαφή που βασίζεται στο Immutability των δεδομένων συμβάλλοντας δραστικά στην μείωση των Side Effects, αλλά επίσης χρησιμοποιήθηκε σαν ένα Facade για την απόκτηση των δεδομένων από την υπηρεσία.

Γνωρίζοντάς ότι οποιαδήποτε ενέργεια εμπεριέχει κάποιου είδους επικοινωνία με μια εξωτερική υπηρεσία αυξάνει σημαντικά τον χρόνο της διεκπεραίωσης της, και έχοντας σαν δεδομένο ότι η JavaScript που τρέχει μέσα σε έναν Browser τρέχει σε ένα και

μοναδικό νήμα (Thread), η εμπειρία του χρήστη θα ήταν πολύ άσχημη όταν θα ζητούσε νέα δεδομένα από τον Server, καθώς η Web Εφαρμογή θα ήταν μη αποκρίσιμη για αρκετό διάστημα μέχρι την απόκτηση των δεδομένων και την εμφάνιση τους. Για αυτόν τον λόγο χρησιμοποιήθηκε ένα πρόσθετο στο Redux Fraemwork, που επιτρέπει την ασύγχρονη απόκτηση δεδομένων και την αποθήκευση τους στο κεντρικό State της εφαρμογής. Όλα τα Components που εξαρτιούνται από αυτό το Global State, όταν αλλάξει θα αλλάξουν και αυτά τα δεδομένα που εμφανίζουν.

Όπως προαναφέρθηκε το Dashboard Component περιέχει 4 σημαντικά Component που εμφανίζουν όλα τα απαραίτητα δεδομένα.

5.4.1.1 SearchBar Component

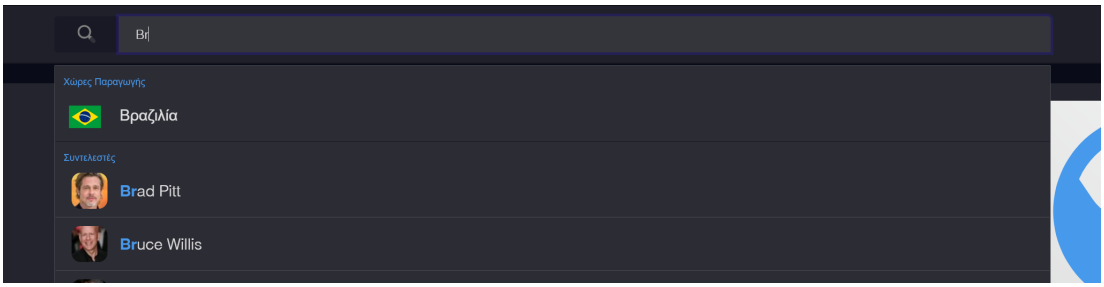
Το πρώτο Component μέσα στο MIDashboard Component είναι η γραμμή αναζήτησης, MISearchBar Component. Το MISearchBar Component δεν προσφέρει την δυνατότητα γενικής αναζήτησης αλλά μόνο την δυνατότητα επιλογής ενός αποτελέσματος από τις προτάσεις της μηχανής αναζήτησης. Οι προτάσεις δημιουργούνται από την μηχανή αναζήτησης μέσο του Server όπως προαναφέρθηκε νωρίτερα κάθε φορά που ο χρήστης γράφει οποιοδήποτε γράμμα στην μπάρα αναζήτησης όπως φαίνεται στον κώδικα 5.24.

```
async function getSuggestions(value: string): Promise<ACResult[]> {
    const results: AutoComplete = (await Service.search(value)).data;
    return results._
      .map(result => {
        result.e.forEach(e => {
          e.i = result.i
        })
        return result;
      })
      .filter(result => result.e.length > 0);
}
```

Διάγραμμα 5.24: Αλγόριθμος ανάκτησης προτάσεων αποτελεσμάτων μηχανής αναζήτησης

Οι προτάσεις της μηχανής αναζήτησης εμφανίζονται με μια μικρή εικόνα στα αριστερά και το κείμενο του αποτελέσματος ακριβώς από διπλά. Ανάλογα με το κείμενο της αναζήτησης που έγραψε ο χρήστης προσπαθεί να υπογραμμίσει με μπλε χρώμα τα

γράμματα που βρέθηκαν στα αποτελέσματα όπως φαίνεται στην εικόνα 5.25.



Διάγραμμα 5.25: MISearchBar Component

Όταν επιλεγεί ένα από τα προτεινόμενα αποτελέσματα η μπάρα αναζήτησης σβήνει και αλλάζει η διεύθυνση όπως φαίνεται στον κώδικα 5.26

```
1 private onSearch = (val: ACEntity) => {
2   this.props.history.push(AppUtils.generateNavigationLink(val, null, null, val.i));
3 }
```

Διάγραμμα 5.26: Αλγόριθμος αλλαγής διεύθυνσης απο το MISearchBar Component

5.4.1.2 MapComponent

Το δεύτερο είναι το Map Component και είναι ένα Component το οποίο προσφέρει ένα προ-ρυθμισμένο Chart από την βιβλιοθήκη Highcharts Highmaps και προσφέρει μια εύκολη διεπαφή για την αλληλεπίδραση και εμφάνιση δεδομένων σε αυτόν τον χάρτη όπως φαίνεται στο σχήμα 5.27. Κάθε φορά που ο δείκτης του ποντικιού περνάει πάνω από μια χώρα εμφανίζεται ενα tooltip που γράφει το όνομα της χώρας και σε πόσες ταινίες έχει συμμετάσχει αυτή η χώρα. Όταν πατηθεί μια χώρα θα επιλεγεί εκτός αν ήταν ήδη επιλεγμένη που τότε θα σταματήσει να είναι επιλεγμένη και θα αλλάξει η διεύθυνση όπως φαίνεται στον κώδικα στο σχήμα 5.28.

Διάγραμμα 5.27: Map Component

```
countrySelected = (data: ICountryData) => {
  this.props.history.push(AppUtils.generateNavigationLink({id: data._id, name:
  ↪  data.name, iso31661: data.iso31661, movies: []} as IProductionCountry))
}
countryUnselected = () => {
  this.props.history.push('/app')
}
```

Διάγραμμα 5.28: Αλγόριθμος αλλαγής διεύθυνσης από το Map Component.

5.4.1.3 MISidePane Component

Το MISidepane Component είναι το τρίτο Component μέσα στο MIDashboard Component και περιέχει το όνομα της κατηγορίας και μια φωτογραφία αν υπάρχει για παράδειγμα αν είναι η κατηγορία ανά ηθοποιό και ο ηθοποιός ονομαζόταν Νικόλαος Μαυρόπουλος θα εμφάνιζε αυτό το άτομο και την φωτογραφία του αν υπάρχει στην βάση δεδομένων. Περιέχει επίσης ένα Component για την δυνατότητα φιλτραρίσματος των αποτελεσμάτων ανα χρόνο, το MIYearPicker Component, και παρακάτω 4 ίδια Component τύπου MIChartCard για εμφάνιση δεδομένων και γραφημάτων συνδυαστικά. Επί το πλείστον η διάταξη του MISidePane Component παραμένει σταθερή με εξαίρεση την κατηγορία ”ανά άτομο” που προστίθεται ένα ακόμα Component το

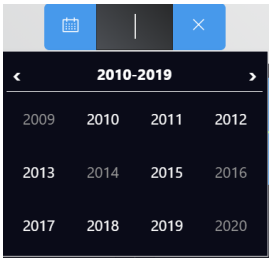
MIRolePicker που δίνει την δυνατότητα φιλτραρίσματος του ρόλου του ατόμου δηλαδή ηθοποιό, σκηνοθέτη, συγγραφέα και παραγωγό.

Το MIChartCard αποτελείται από ένα Grid το οποίο περιέχει ένα γράφημα Highcharts LineChart που στην ουσία είναι ένα γράφημα με μια η περισσότερες γραμμές, ένα εικονίδιο ακριβώς πάνω στο γράφημα περιγραφικό για τι δεδομένα παρουσιάζονται, και δεδομένα από κάτω. Τα δεδομένα του γραφήματος αναπτύσσουν ένα η παραπάνω ποσοτικό πεδίο ανά χρόνο για όσα χρόνια υπάρχουν στην συγκεκριμένη επιλεγμένη κατηγορία, ενώ τα δεδομένα από κάτω εμφανίζουν είτε τα μέγιστα, είτε τα ελάχιστα, είτε τους μέσους όρους των δεδομένων όπως φαίνεται στο σχήμα 5.29. Τα γραφήματα των MIChartCard Components όταν φιλτράρεται η κατηγορία ανά χρόνο απενεργοποιούνται, αλλά τα δεδομένα παρακάτω που αναγράφουν μέγιστα ελάχιστα και μέσους όρους παραμένουν και αναφέρονται στο επιλεγμένο έτος για αυτήν την κατηγορία. Δεν υπήρχε ουσιαστικός λόγος τα γραφήματα να έχουν δεδομένα καθώς αυτό που βλέπει ο χρήστης δεν είναι τα συνολικά δεδομένα παρά μόνο του επιλεγμένου έτους.

Διάγραμμα 5.29: MIChartCard Component

Το MIYearPicker Component αποτελείται από 2 κουμπιά και ένα πεδίο εισαγωγής όπως φαίνεται στο σχήμα 5.30. Όταν πατηθεί το κουμπί με το εικονίδιο ενός ημερολογίου η το πεδίο εισαγωγής, ανοίγει ένα μενού που επιτρέπει την επιλογή ενός έτους που υπάρχει για την επιλεγμένη κατηγορία ενώ όταν πατηθεί το κουμπί με εικονίδιο ένα ”X” καθαρίζεται η επιλογή του έτους και εμφανίζονται τα συνολικά δεδομένα. Όταν εκτελεστεί μια ενέργεια από αυτό το Component, θα αλλάξει την διεύθυνση έτσι ώστε να αναλάβει το Dashboard Module την αλλαγή των δεδομένων όπως φαίνεται στον κώδικα 5.31.

Το MIRolePicker Component εμφανίζεται μόνο όταν η επιλεγμένη κατηγορία εί-



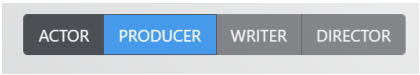
Διάγραμμα 5.30: MIYearPicker Component

```
yearSelected = (year: number) => {
  const activeView = this.props.rootState.dashboardState.activeView();
  const activeEntity = this.props.rootState.dashboardState.activeView().activeEntity;
  let role = null;
  if (activeView.entityType === EntityType.PERSON) {
    const _activeView = activeView as MovieInsightsPerPersonState;
    role = _activeView.activeRole;
  }
  this.props.history.push(AppUtils.generateNavigationLink(activeEntity.entity, role,
  ↪  year))
}
yearUnselected = () => {
  const activeEntity = this.props.rootState.dashboardState.activeView().activeEntity;
  if (activeEntity.entity) {
    this.props.history.push(AppUtils.generateNavigationLink(activeEntity.entity));
  } else {
    this.props.history.push('/app');
  }
}
```

Διάγραμμα 5.31: Αλγόριθμος αλλαγής διεύθυνσης απο το MIYearPicker Component.

ναι ”ανά άτομο”. Αποτελείται από 4 κουμπιά μονής επιλογής. Αυτό σημαίνει ότι όταν πατηθεί ένα κουμπί επιλέγεται μένει πατημένο και οποιαδήποτε άλλο κουμπί ήταν πατημένο πρωτύτερα αφαιρείται η επιλογή του. Λειτουργεί ακριβώς με τον ίδιο τρόπο με τα παραδοσιακά Radio Buttons. Τα 4 κουμπιά αντιστοιχούν στους 4 ρόλους που μπορεί να έχει ένα άτομο όπως Ηθοποιός, Σκηνοθέτης, Συγγραφέας και Παραγωγός όπως φαίνεται στο σχήμα 5.32. Όταν ένα Role είναι επιλεγμένο το ανάλογο κουμπί επιλέγεται και εμφανίζεται με χρώμα μπλε, όταν δεν υπάρχει το συγκεκριμένο Role στο άτομο είναι απενεργοποιημένο και εμφανίζεται με χρώμα ανοιχτό γκρι, και αν υπάρχει και δεν είναι επιλεγμένο εμφανίζεται με χρώμα σκούρο γκρι και ο κώδικας επιλογής φαίνεται στο σχήμα. Όταν αλλάζει η επιλογή ο κώδικας που αλλάζει την διεύθυνση φαίνεται

στο σχήμα 5.33.



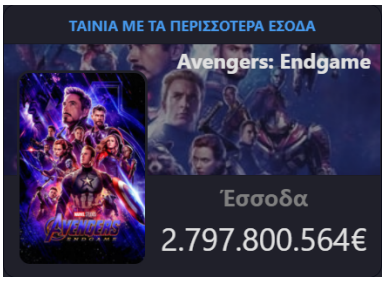
Διάγραμμα 5.32: MIRolePicker Component

```
1 private onCreditSelect = (credit: CreditRole) => {
2   const activeView = this.props.rootState.dashboardState.activeView() as
    ↪  MovieInsightsPerPersonState;
3   this.props.history.push(AppUtils.generateNavigationLink(activeView._activeEntity.
    ↪  person, credit, activeView.isPerYear ? activeView.activeYearEntity.entity :
    ↪  null));
4 }
```

Διάγραμμα 5.33: Αλγόριθμος αλλαγής διεύθυνσης από το MIRolePicker Component.

5.4.1.4 MIInsightsPanel Component

Το MIInsightsPanel Component είναι το τέταρτο και τελευταίο Component που βρίσκεται μέσα στο MIDashboard Component. Περιέχει πολλά Component ίδιου τύπου MICard Component. Το κάθε MICard Component είναι μια ”κάρτα” που εμφανίζει ένα στοιχείο για ένα άτομο, μια ταινία, μια εταιρία παραγωγής, μια χώρα παραγωγής η ένα είδος ταινίας. Για παράδειγμα αν μια κάρτα για μια ταινία με τα μεγαλύτερα έσοδα θα εμφανιζόταν όπως στο σχήμα 5.34.



Διάγραμμα 5.34: MICard Component

Το MICard Component είναι ένα Component το οποίο έχει σχεδιαστεί με ρευστή διάταξη και δυναμικό περιεχόμενο. Ανάλογα με το τι περιεχόμενο θα του δοθεί, θα αλλάξει την διάταξη του και θα εμφανίσει τα ανάλογα δεδομένα. Υποστηρίζει Placeholders για την αναμονή των απομακρυσμένων δεδομένων εμφανίζοντας

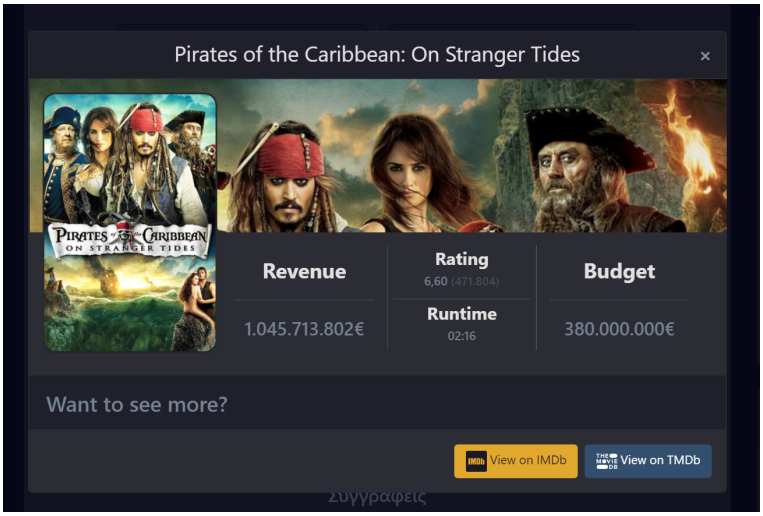
ένα Spinner, και υποστηρίζει και την εμφάνιση ενός μηνύματος λάθους σε περίπτωση που τα δεδομένα για την συγκεκριμένη κατηγορία δεν υπάρχουν. Επιπρόσθετα κάθε Component τύπου MICard μπορεί να πατηθεί σαν κουμπί και ανάλογα με το περιεχόμενο αλλάζει την διεύθυνση στην γραμμή διευθύνσεων του Browser, έτσι ώστε να εμφανιστούν τα ανάλογα δεδομένα μετέπειτα, από το Dashboard Module. Όλα τα MICard λειτουργούν με τον ίδιο τρόπο εκτός από το MICard που περιέχει δεδομένα ταινιών και το MICard που περιέχει δεδομένα χωρών. Καθώς ο ρόλος της εφαρμογής δεν είναι να εμφανίζει αναλυτικά στοιχεία για ταινίες επειδή υπάρχουν ήδη πολύ γνωστές υπηρεσίες για αυτόν τον σκοπό όπως προαναφέρθηκαν το IMDb και το TMDb, όταν πατηθεί ένα MICard με περιεχόμενο ταινιών θα εμφανιστεί ένα παράθυρο πάνω από την εφαρμογή το οποίο είναι το MIMovieInfoModal Component όπως φαίνεται στον κώδικα 5.35. Το MICard που περιέχει δεδομένα χωρών λειτουργεί όπως όλα τα άλλα πλήν του MICard με δεδομένα ταινίας, αλλά παράλληλα εστιάζει και κάνει zoom στον χάρτη την επιλεγμένη χώρα.

```
renderCardLinked() {
  const link = this.state.entity?AppUtils.generateNavigationLink(this.state.entity):''
  return (<NavLink className="mi-card-link" onClick={this.onLinkClick}
  ↪  to={link}>{this.renderCard()}</NavLink>);
}
_render() {
  return (
    <>
      {this.state.loaded && this.state.entity ?this.renderCardLinked() :
      ↪  this.renderCard()}
      {this.props.entityType === TmdbEntityType.MOVIE ?(<MIMovieInfoModal
      ↪  open={this.state.modal} onClose={() => this.setState({modal: false})}
      ↪  entity={this.props.entity as any}/>) : null}
    </>
  )
}
```

Διάγραμμα 5.35: Αλγόριθμος αλλαγής διεύθυνσης από το MIRolePicker Component.

Το MIMovieInfoModal Component περιέχει το πόστερ της ταινίας καθώς και μια εικόνα background σχετική με την ταινία και περιέχει τα πολύ βασικά στοιχεία που χρησιμοποιεί η εφαρμογή όπως, τα έσοδα, τα έξοδα, την βαθμολογία με της ψήφους της ταινίας καθώς και το πόσο διαρκεί η ταινία συνολικά. Επιπρόσθετα δίνει την επιλογή

για περισσότερα στοιχεία για αυτήν την ταινία προσφέροντας 2 συνδέσμους ανακατεύθυνσης που ο ένας πηγαίνει τον χρήστη στον ισότοπο του TMDb στην επιλεγμένη ταινία και ο άλλος στο IMDB όπως φαίνεται στο σχήμα 5.36



Διάγραμμα 5.36: MIMovieInfoModal Component

5.4.2 Admin Module

Το Admin Module είναι ένα σύνολο λειτουργιών που επιτρέπει έναν διαχειριστή να διαχειρίζεται με ευκολία την εφαρμογή. Στην παρούσα υλοποίηση το Admin Module έχει 2 διαφορετικές σελίδες - Components, το Κέντρο Ελέγχου και το Κέντρο Διαχείρισης Καταγραφών. Για να αποκτήσει πρόσβαση ένας χρήστης στις λειτουργίες του Admin Module, πρέπει να έχει κάνει Login και να είναι εξουσιοδοτημένος με τον ρόλο ADMIN, διαχειριστή.

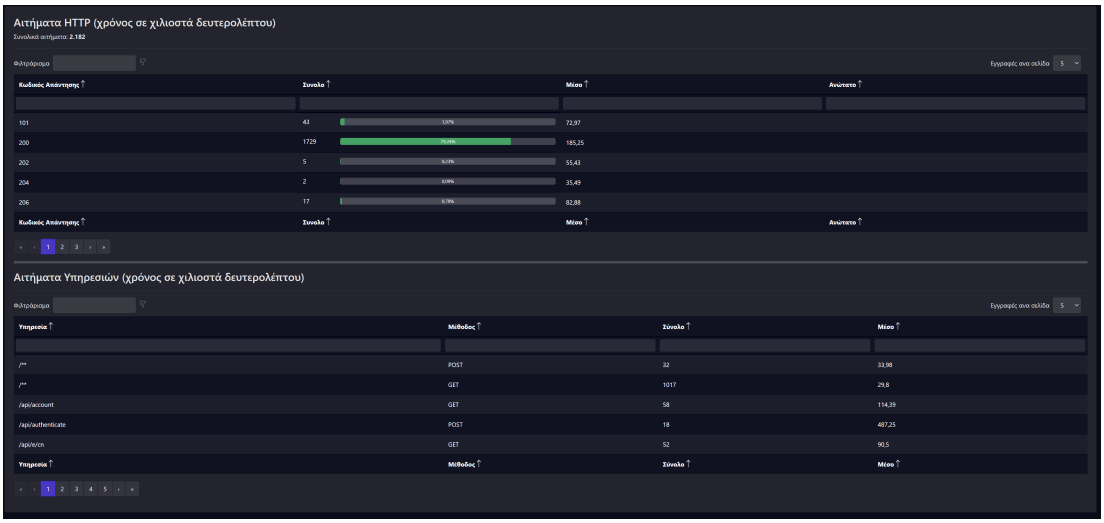
Το Κέντρο Ελέγχου είναι μια σελίδα που επιτρέπει την παρακολούθηση της κατάστασης της υγείας της εφαρμογής, προσφέρει στατιστικά των υπηρεσιών αλλά και του λειτουργικού που τρέχει η εφαρμογή αλλα παράλληλα προσφέρει και διαγνωστικά για να γνωρίζει ο διαχειριστής τι γίνεται ανά πάσα στιγμή στην εφαρμογή αυτήν όπως φαίνεται στις εικόνες 5.37 και 5.38.

Αποτελείται από πολλά διαφορετικά Components τα οποία ενημερώνονται ανάλογα με τον τύπο των δεδομένων που προσφέρουν ανά μισό, δύο και ανά πέντε δευτερόλεπτα με την χρήση WebSockets μέσο του Spring Boot Actuator έτσι ώστε η εικόνα που βλέπει ο διαχειριστής να αντικατοπτρίζει πάντα την κατάσταση της εφαρμογής εκείνη την χρονική στιγμή.

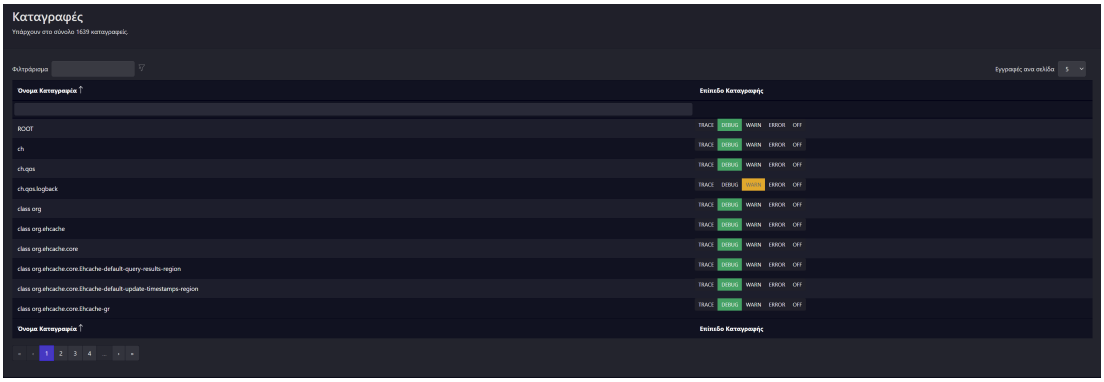
Το Κέντρο Διαχείρισής Καταγραφών είναι στην ουσία ένας διαδραστικός πίνακας που επιτρέπει στον διαχειριστή να αλλάζει τα επίπεδα καταγραφής διάφορων υπηρεσιών μέσα στον Server. Τα επίπεδα καταγραφής ορίζουν τι πληροφορίες θα δίνει η κάθε υπηρεσία. Το επίπεδο καταγραφής ERROR για παράδειγμα καταγράφει μόνο τα μηνύματα σφαλμάτων στο αρχείο καταγραφής της υπηρεσίας, ενώ το επίπεδο καταγραφής WARNING καταγράφει και τα μηνύματα σφαλμάτων αλλά και τα μηνύματα προειδοποιήσεων. Αυτό είναι πολύ χρήσιμο όταν χρησιμοποιείται σε συνδυασμό με μια υπηρεσία διαχείρισης καταγραφών όπως η Logstash για παράδειγμα. Με αυτόν τον τρόπο ο διαχειριστής δεν χρειάζεται να έχει φυσική πρόσβαση στο μηχάνημα που τρέχει την εφαρμογή για να μπορεί να δει τα αρχεία καταγραφής παρά μόνο να μπει στην υπηρεσία διαχείρισης καταγραφών και να βρει ότι χρειάζεται από εκεί.

Ο πίνακας του Κέντρου Διαχείρισής Καταγραφών φαίνεται στην εικόνα 5.39. Η συγκεκριμένη εικόνα τραβήχτηκε κατά την ανάπτυξη της εφαρμογής και δείχνει ότι το κεντρικό επίπεδο καταγραφών ROOT είναι ορισμένο στο DEBUG που σημαίνει ότι καταγράφει ακόμα και μηνύματα διαγνωστικών με κάποιες εξαιρέσεις που έχουν οριστεί σε επίπεδο WARNING για να μην υπερφορτώσουν τα αρχεία καταγραφής με περιττές πληροφορίες.

Διάγραμμα 5.37: Κέντρο Ελέγχου



Διάγραμμα 5.38: Κέντρου Ελέγχου - συνέχεια



Διάγραμμα 5.39: Κέντρο Διαχείρισής Καταγραφών

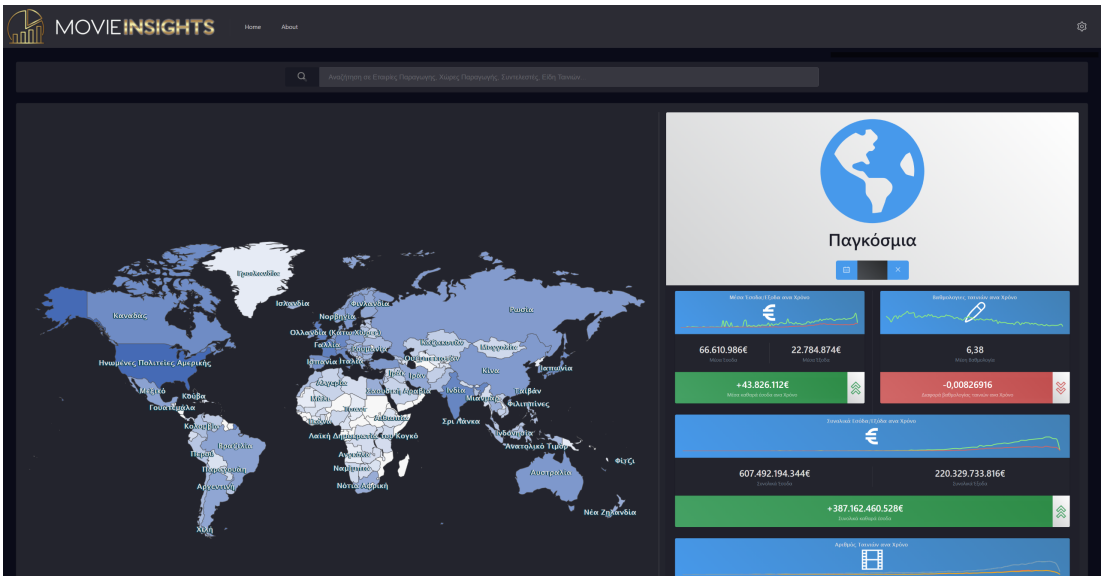
Κεφάλαιο 6

Εγχειρίδιο Χρήσης

Παρακάτω παρατίθεται ένα αναλυτικό εγχειρίδιο χρήσης για το πως μπορεί να χρησιμοποιηθεί η εφαρμογή αυτή από κάθε χρήστη.

Αρχικά ο χρήστης πρέπει να ανοίξει έναν φυλλομετρητή (Browser) και να πλοηγηθεί στην σελίδα http://www.movieinsights.gr/ . Η Αρχική σελίδα που θα μπει είναι και ουσιαστικά όλη η εφαρμογή. Η πρώτη κατηγορία που θα εμφανιστεί με το που θα μπει ο χρήστης στην εφαρμογή είναι η κατηγορία των γενικών στοιχείων.

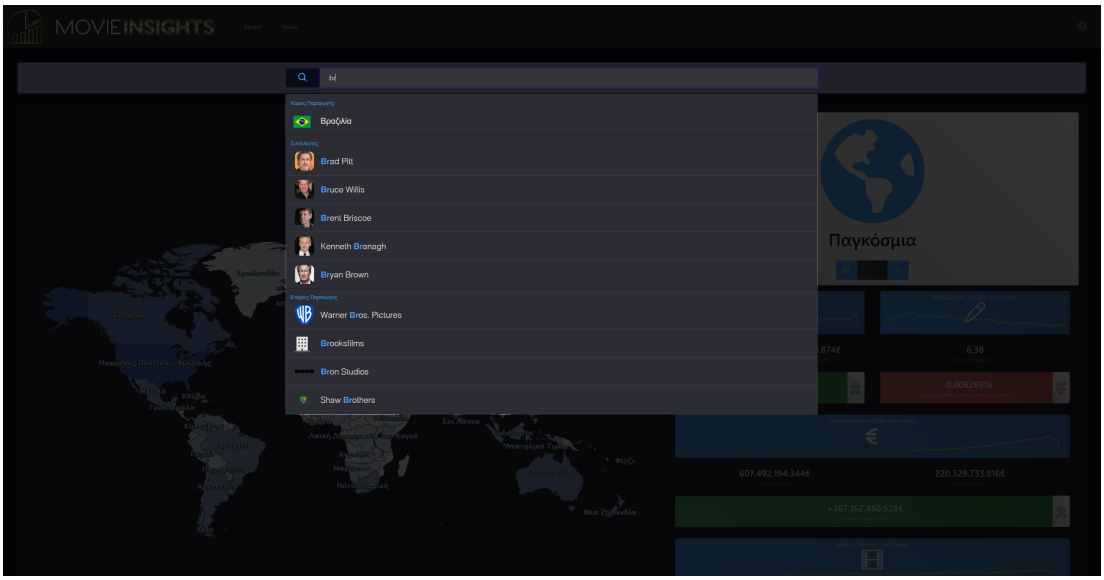
Η κατηγορία των γενικών στοιχείων περιέχει στοιχεία συγκεντρωτικά για όλες τις ταινίες, τους ηθοποιούς, τις εταιρίες παραγωγής, τις χώρες παραγωγής αλλά και τα είδη παραγωγής που υπάρχουν στην βάση δεδομένων της εφαρμογής. Αυτή η κατηγορία είναι η Παγκόσμια όπως φαίνεται στην εικόνα 6.1



Διάγραμμα 6.1: Αρχική σελίδα

6.1 Μηχανή Αναζήτησης

Ο χρήστης στην κορυφή της ιστοσελίδας θα βρει μια μπάρα αναζήτησης που του επιτρέπει να ψάξει άτομα, εταιρίες, χώρες και είδη ταινιών. Σε κάθε γράμμα που θα πληκτρολογήσει θα του εμφανίζονται ανάλογα αποτελέσματα για να επιλέξει. Ένα αποτέλεσμα αποτελείται από μια φωτογραφία και έναν τίτλο. Για παράδειγμα αν το αποτέλεσμα είναι μια χώρα η εικόνα θα είναι η σημαία της χώρας και ο τίτλος, το όνομα της. Αν το αποτέλεσμα είναι ένα άτομο η εικόνα θα είναι μια φωτογραφία του ατόμου αν υπάρχει, αλλιώς μια προκαθορισμένη φωτογραφία, και το όνομα του. Αν είναι είδος ταινίας θα υπάρχει ένα εικονίδιο για εικόνα και το όνομα του είδους. Και αν είναι εταιρία το λογότυπο της εταιρίας αν υπάρχει, αλλιώς ένα προκαθορισμένο εικονίδιο κτηρίου όπως φαίνεται στην εικόνα 6.2 στο 2ο αποτέλεσμα των εταιριών, και το όνομα της. Ο χρήστης πρέπει να επιλέξει ένα από τα προτεινόμενα αποτελέσματα καθώς η γενική αναζήτησης με το πλήκτρο ”Enter” δεν υποστηρίζεται.

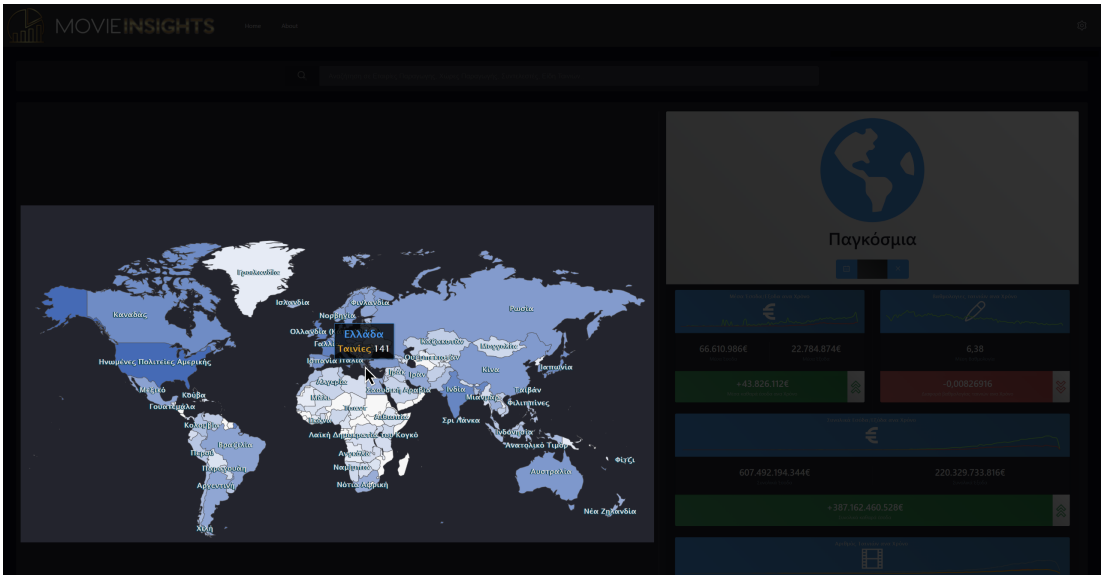


Διάγραμμα 6.2: Γραμμή Αναζήτησης

6.2 Χάρτης

Ακριβώς από κάτω από την μπάρα αναζήτησης βρίσκεται στα αριστερά ένας παγκόσμιος χάρτης ο οποίος έχει δεδομένα για όλες τις χώρες οι οποίες έχουν παράξει έστω και μια ταινία και βρίσκονται φυσικά στην βάση δεδομένων της εφαρμογής. Οι χώρες στον χάρτη χρωματίζονται από διαφορετικές αποχρώσεις του μπλε, όσο πιο

έντονο είναι το χρώμα τόσες παραπάνω ταινίες έχει αυτή η χώρα. Ο Χάρτης υποστηρίζει μεγέθυνσή και κύλιση. Αν ο χρήστης μετακινήσει τον κένσορα του ποντικιού πάνω από μια χώρα ένα βοηθητικό παραθυράκι θα εμφανιστεί από πάνω αναγράφοντας το όνομα της χώρα και το πόσες ταινίες έχει όπως φαίνεται στη εικόνα 6.3. Ο χρωματισμός και τα δεδομένα του χάρτη παραμένουν τα ίδια ανεξάρτητα από την κατηγορία ή το έτος που έχει επιλεγεί.

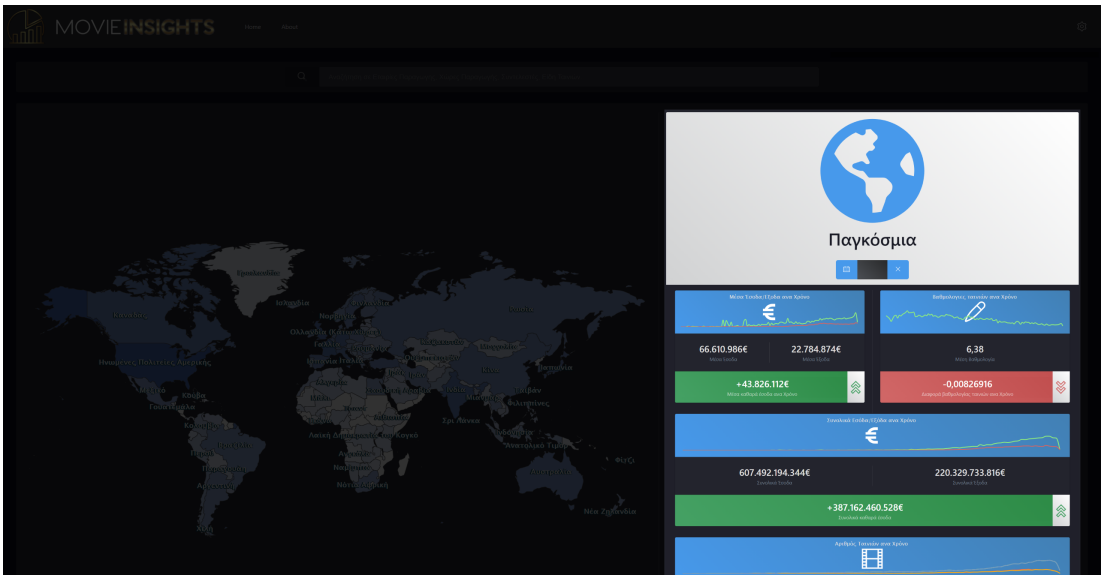


Διάγραμμα 6.3: Παγκόσμιος Χάρτης

6.3 Πλαϊνή Στήλη

Στην κατηγορία γενικών στοιχείων στα τα δεξιά του χάρτη βρίσκεται η στήλη με κάποια βασικά δεδομένα όπως φαίνεται στην εικόνα 6.4. Καθώς τα δεδομένα είναι πολλά πρέπει ο χρήστης να κυλίσει την σελίδα προς τα κάτω λίγο για να τα δει όλα. Στην αρχική σελίδα ο χρήστης βρίσκεται στην κατηγορία γενικών στοιχείων.

Στο πάνω, το άσπρο μέρος της πλαϊνής στήλης, όπως φαίνεται στην εικόνα 6.5, βρίσκονται τα στοιχεία για το ποια κατηγορία ο χρήστης αντικρίζει. Περιέχει το όνομα και την εικόνα της κατηγορίας και περιέχει ένα πεδίο που επιτρέπει στον χρήστη να επιλέξει, είτε πατώντας στο εικονίδιο του ημερολογίου, είτε πατώντας στο πεδίο εισαγωγής, ένα έτος για να δει τα στοιχεία της κατηγορίας ανά έτος. Στα δεξιά του πεδίου εισαγωγής υπάρχει ένα εικονίδιο με το σύμβολο ”X” που επιτρέπει τον χρήστη να καθαρίσει την επιλογή έτους για να δει ξανά τα συνολικά δεδομένα.

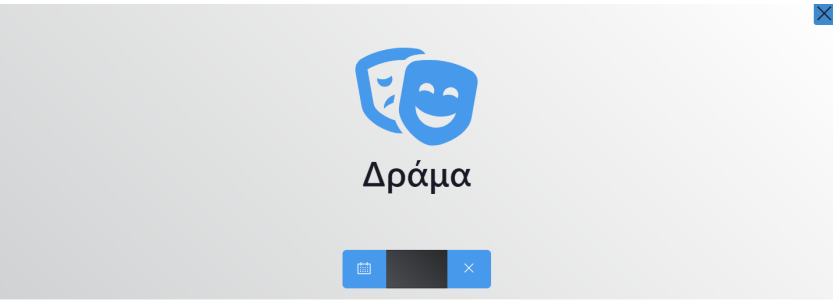


Διάγραμμα 6.4: Πλαϊνή στήλη

Διάγραμμα 6.5: Πάνω μέρος πλαϊνής στήλης

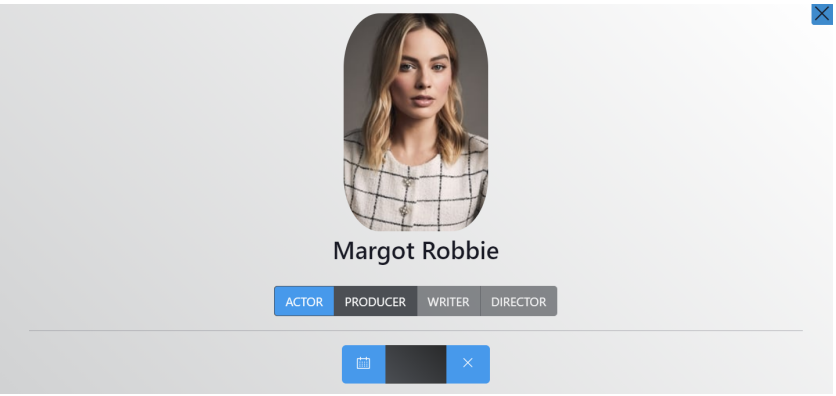
Στο πάνω μέρος της πλαϊνής στήλης όταν η κατηγορία είναι οποιαδήποτε άλλη πέρα των γενικών στοιχειών στο πάνω δεξιό μέρος εμφανίζεται ένα κουμπί με το σύμβολο ”X”, όπως φαίνεται στην εικόνα 6.6, το οποίο όταν πατηθεί επαναφέρει τον χρήστη στην κατηγορία γενικών στοιχείων.

Το πάνω μέρος της πλαϊνής στήλης παραμένει το ίδιο για όλες τις κατηγορίες εκτός της κατηγορίας ”ανά άτομο”. Όταν η κατηγορία αναφέρεται σε ένα άτομο εμφανίζεται ένα ακόμα στοιχείο πάνω από το στοιχείο επιλογής έτους. Η εφαρμογή συλλέγει δεδομένα για ένα άτομο σε κατηγορίες. Συλλέγει δεδομένα για το άτομο σαν ηθοποιό, σαν παραγωγό, σαν σκηνοθέτη και σαν συγγραφέα. Για αυτό το λόγο σε εκείνο το σημείο υπάρχουν 4 κουμπιά που σου επιτρέπουν να επιλέξεις να δεις στοιχεία για ένα άτομο σε διαφορετικό ρόλο όπως φαίνεται στην εικόνα 6.7. Από προεπιλογή επιλέγεται ο ρόλος με τον οποίο εμφανίζεται πιο συχνά το άτομο αυτό. Δεν γίνεται να μην επιλεγεί



Διάγραμμα 6.6: Κουμπί ”X” στο πάνω μέρος πλαϊνής στήλης

κανένας ρόλος καθώς τα δεδομένα είναι διαφορετικά και βγαίνουν διαφορετικά συμπεράσματα για την πορεία του συγκεκριμένου ατόμου σε κάθε ρόλο. Όταν επιλεγεί ένας ρόλος το κουμπί που κρατάει αυτόν τον ρόλο θα γίνει μπλε. Όταν ένας ρόλος δεν είναι επιλεγμένος είναι σκούρο γκρι. Βέβαια ένα άτομο μπορεί να μην έχει γίνει για παράδειγμα ποτέ συγγραφέας ταινίας η σκηνοθέτης. Όταν λοιπόν δεν υπάρχουν δεδομένα για το συγκεκριμένο άτομο για έναν συγκεκριμένο ρόλο, το κουμπί που κατέχει αυτόν τον ρόλο θα απενεργοποιηθεί και θα χρωματιστεί με χρώμα ανοιχτό γκρι. Στην εικόνα βλέπουμε την ηθοποιό Margot Robbie, με προεπιλεγμένο ρόλο ”ACTOR” για ηθοποιό και με έναν διαθέσιμο ακόμα ρόλο ”PRODUCER”, παραγωγό που σημαίνει ότι η εφαρμογή έχει στοιχεία για την Margot Robbie σαν ηθοποιό και σαν παραγωγό ταινιών αλλά δεν έχει δεδομένα σαν συγγραφέα η σκηνοθέτη.



Διάγραμμα 6.7: Επιλογή Ρόλου στο πάνω μέρος πλαϊνής στήλης

Στο κάτω μέρος της πλαϊνής στήλης βρίσκονται 4 κάρτες δεδομένων όπως φαίνεται στην εικόνα 6.8.

Η κάθε κάρτα περιέχει 3 κομμάτια και 4 καταστάσεις όπως φαίνεται στο σχήμα 6.9. Το πρώτο κομμάτι πάνω πάνω είναι ένα γράφημα το οποίο περιέχει ένα η περισσότερα

Διάγραμμα 6.8: Πάνω μέρος πλαϊνής στήλης

ποσοτικά δεδομένα ανά χρόνο, τον τίτλο του γραφήματος για να γνωρίζει ο χρήστης τι κοιτάει καθώς και ένα περιγραφικό εικονίδιο για να γνωρίζει ο χρήστης τι είναι τα ποσοτικά δεδομένα που κοιτάει. Από κάτω βρίσκεται το δεύτερο κομμάτι το οποίο περιέχει ένα η περισσότερα ποσοτικά δεδομένα που συνήθως είναι είτε τα μέγιστα, είτε τα ελάχιστα, είτε οι μέσοι όροι των δεδομένων των γραφημάτων. Το τρίτο και τελευταίο κομμάτι είναι ακριβώς από κάτω και εμφανίζει συνήθως μια διαφορά τιμών ή έναν υπολογισμό δεδομένων με βάση τα προηγούμενα.

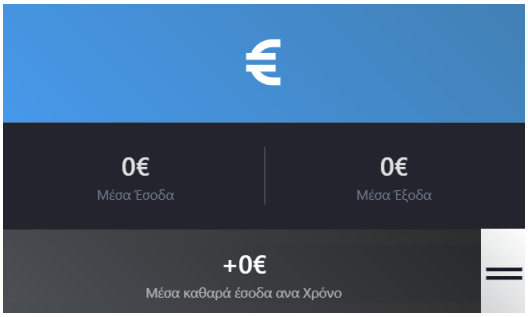
Όπως προαναφέρθηκε μια κάρτα μπορεί να έχει 4 διαφορετικές καταστάσεις. Όταν η κατηγορία που βλέπει ο χρήστης έχει φιλτραριστεί ανά έτος τότε όλα τα γραφήματα όλων των καρτών, στο πρώτο κομμάτι, που αναφέρουν στοιχεία ανά χρόνο θα σβηστούν και ανάλογα με το τι δείχνει η κάρτα, τα άλλα 2 κομμάτια θα έχουν δεδομένα είτε θα μηδενιστούν όπως φαίνεται στην Κατάσταση 4 στο σχήμα 6.9IV. Όταν δεν είναι φιλτραρισμένη η κατηγορία ανά έτος, αν το δεδομένο στο τρίτο κομμάτι της κάρτας είναι θετικό θα έχει χρώμα background πράσινο και στα δεξιά θα έχει ενα εικονίδιο με 2 βελάκια να δείχνουν προς τα πάνω όπως φαίνεται στο σχήμα της πρώτης κατάστασης 6.9I, αν είναι αρνητικό το χρώμα background θα είναι κόκκινο και στα δεξιά θα έχει ένα εικονίδιο με 2 βελάκια να δείχνουν προς τα κάτω όπως φαίνεται στο σχήμα της δεύτερης κατάστασης 6.9II, και αν είναι μηδέν τότε το χρώμα background θα είναι γκρι και το εικονίδιο στα δεξιά θα είναι 2 οριζόντιες γραμμές σαν το σύμβολο ”ίσον” των μαθηματικών, όπως φαίνεται στο σχήμα της τρίτης κατάστασης 6.9III.

Η πρώτη κάρτα στην πλαϊνή στήλη εμφανίζει στο γράφημα τα μέσα έσοδα και τα

(I) Πρώτη κατάσταση

(II) Δεύτερη κατάσταση

(III) Τρίτη κατάσταση



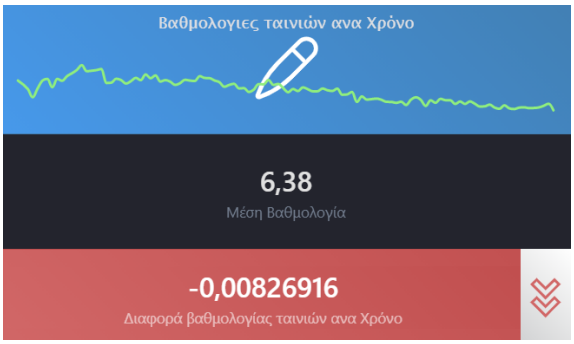
(IV) Τέταρτη κατάσταση

Διάγραμμα 6.9: Οι κάρτες της πλαϊνής στήλης και οι καταστάσεις τους

έξοδα όλων των ταινιών στην συγκεκριμένη κατηγορία, και πιο συγκεκριμένα καθώς η κατηγορία η αρχική είναι τα συνολικά δεδομένα, εμφανίζει τα μέσα έσοδα / έξοδα όλων των ταινιών της βάσης δεδομένων. Αν ο χρήστης πάει τον κένσορα πάνω από το γράφημα εμφανίζεται ένα μικρό παράθυρο πάνω από τον κένσορα που αναγράφει το έτος και τα μέσα έσοδα έξοδα του έτους για την τοποθεσία που βρίσκεται ο κένσορας όπως φαίνεται στο σχήμα 6.10. Απο κάτω αναφέρονται συνολικά κατά μέσο όρο τα μέσα έσοδα και έξοδα των ταινιών και στο τρίτο κομμάτι εμφανίζεται το μέσο καθαρό κέρδος των ταινιών συνολικά.

Διάγραμμα 6.10: Η Κάρτα με το παράθυρο απο πάνω

Στην δεύτερη κάρτα στο γράφημα εμφανίζονται οι μέσες βαθμολογίες των ταινιών ανά έτος. Στο δεύτερο κομμάτι εμφανίζεται συνολικά η μέση αξιολόγηση όλων των ταινιών της κατηγορίας και στο τρίτο κομμάτι εμφανίζεται η τάση των βαθμολογιών ανά χρόνο. Αυτό σημαίνει αν οι ταινίες μέσα στο πέρασμα των χρώνων έγιναν καλύτερες η χειρότερες. Αυτά τα δεδομένα φαίνονται στην εικόνα 6.11.

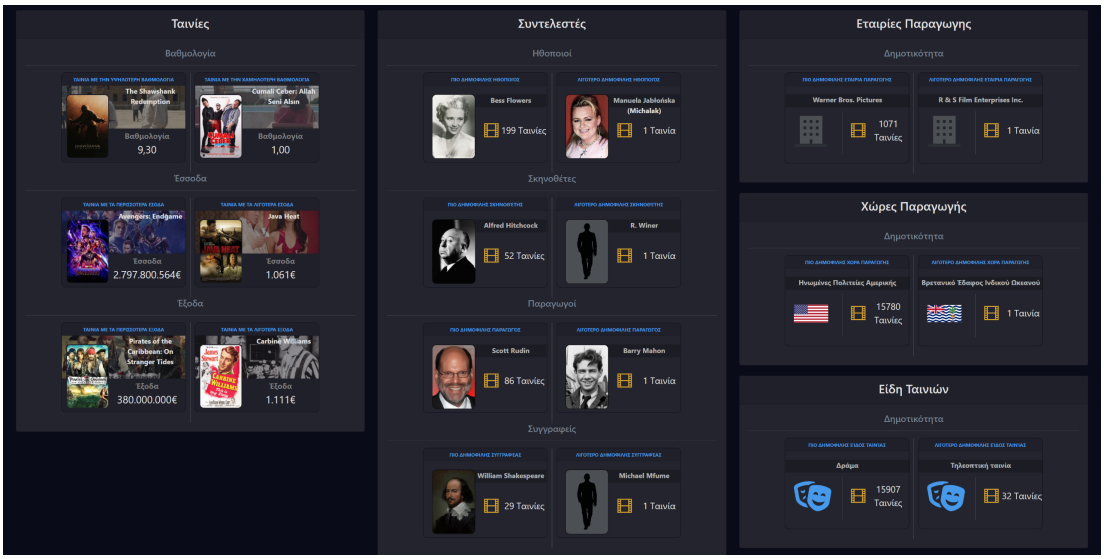


Διάγραμμα 6.11: Η Κάρτα βαθμολογιών

Η τρίτη κάρτα είναι ακριβώς το ίδιο με την πρώτη κάρτα με την διαφορά ότι αντί για τους μέσους εμφανίζει τα συνολικά δεδομένα, και η τέταρτη κάρτα αναγράφει κάποια στοιχεία για το νούμερο των ταινιών που υπάρχουν στην βάση δεδομένων και πόσες από αυτές παρέχουν δεδομένα εσόδων, εξόδων και βαθμολογιών.

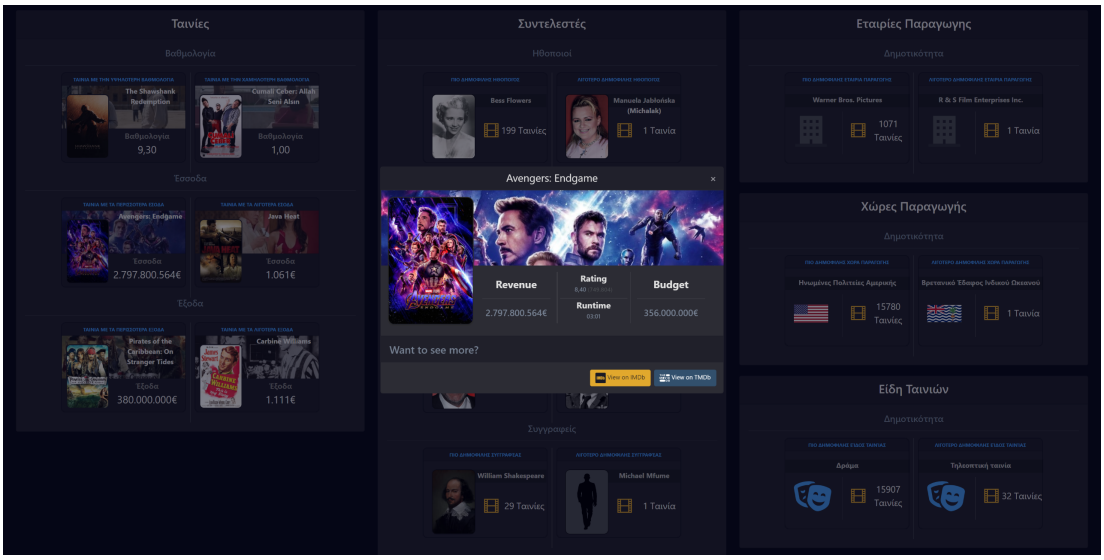
6.4 Κύρια Στοιχεία της Εφαρμογής

Ακριβώς κάτω από τον χάρτη και την πλαϊνή στήλη βρίσκονται τα κύρια στοιχεία της εφαρμογής. Χωρίζονται σε 5 κατηγορίες ”Ταινίες”, ”Συντελεστές”, ”Εταιρίες Παραγωγής”, ”Χώρες Παραγωγής” και ”Είδη Ταινιών” όπως φαίνεται στην εικόνα 6.12.



Διάγραμμα 6.12: Κύρια Στοιχεία της εφαρμογής

Στην κατηγορία των ταινιών προσφέρονται ταινίες με την μεγαλύτερη και μικρότερη βαθμολογία, και ταινίες με τα περισσότερα και λιγότερα έσοδα και έξοδα. Καθώς ο στόχος της εφαρμογής δεν είναι η προβολή αναλυτικών στοιχείων μιας ταινίας, όταν πατάει ο χρήστης πάνω σε μια από τις κάρτες που εμφανίζονται ανοίγει ένα παραθυράκι προσφέροντας κάποια πολύ βασικά δεδομένα όπως τα έσοδα, έξοδα, η βαθμολογία και ο χρόνος που διαρκεί αυτή η ταινία, και σε περίπτωση που ο χρήστης θέλει να δει παραπάνω στοιχεία για αυτήν την ταινία του δίνεται η δυνατότητα από αυτό το παράθυρο να δει τα στοιχεία που θέλει είτε στην υπηρεσία Internet Movie DataBase (IMDB) είτε στην υπηρεσία The Movie Database (TMDb) όπως φαίνεται στην εικόνα 6.13.



Διάγραμμα 6.13: Παράθυρο προβολής στοιχείων ταινίας

Στις υπόλοιπες κατηγορίες εμφανίζονται μόνο οι πιο δημοφιλής και λιγότερο δημοφιλής σε σχέση με την συμμετοχή τους σε ταινίες γενικότερα. Όταν ο χρήστης πατήσει σε μια κάρτα στις υπόλοιπες κατηγορίες η γενική κατηγορία της εφαρμογής θα αλλάξει και θα εμφανίσει στοιχεία της επιλογής του χρήστη.

Κεφάλαιο 7

Συμπεράσματα

Αν και η εφαρμογή αυτή είναι πλήρως λειτουργική κρύβει κάποια προβλήματα στην συνολική υλοποίηση της. Έχει έναν αρκετά γρήγορο τρόπο για τον υπολογισμό των δεδομένων που εξαρτάται σημαντικά από την επεξεργαστική ισχύ του μηχανήματος αλλά χρειάζεται να καταναλώσει ένα πάρα πολύ μεγάλο μέρος μνήμης RAM. Επίσης δεν υποστηρίζει ενημέρωση δεδομένων στην παρούσα κατάσταση της. Αν χρειαστεί να ενημερωθούν τα δεδομένα θα πρέπει να υπολογιστούν όλα από την αρχή, και όσα περισσότερα δεδομένα δοθούν για επεξεργασία τόση παραπάνω μνήμη θα καταναλώσει που φτάνει σε ένα σημείο θεωρητικά να μην γίνεται εφικτός ο υπολογισμός αυτών των δεδομένων αν ο όγκος των δεδομένων αυξηθεί σημαντικά.

Η παρούσα υλοποίηση δεν επιτρέπει επιπρόσθετα το Scaling. Αν αυτή η εφαρμογή γινόταν Viral δεν θα υπήρχε εφικτός τρόπος να γίνει Scale και να μπορέσει να εξυπηρετήσει όλους τους χρήστες που θα ζητούσαν δεδομένα.

Πάραυτα προσφέρει μία πληθώρα δεδομένων και ένα άτομο του χώρου της βιομηχανίας του κινηματογράφου μπορεί να τα βρει χρήσιμα για να μπορέσει να επιτελέσει το έργο του με μεγαλύτερη ευκολία.

7.1 Στόχοι

Οι στόχοι της πτυχιακής επιτεύχθηκαν στο σύνολό τους, με εξαίρεση κάποιων που επιλέχθηκε να μην υλοποιηθούν λόγω πρακτικών και τεχνικών προβλημάτων. Ένας από τους αρχικούς στόχους ήταν να υπολογιστούν με έναν συνδυασμό δεδομένων, συμπεράσματα για το πως μια ταινία έχει επηρεάσει τον τουρισμό στις χώρες και πόλεις

γυρισμάτων. Καθώς θα ήταν ένα πολύ καλό μετρικό για την ανάδειξη της δύναμης των OpenData, η ίδια η απουσία των OpenData για αυτά τα δεδομένα, συνετέλεσε στην μη υλοποίησή αυτής της λειτουργίας. Ένας ακόμα αρχικός στόχος που δεν συμπεριλήφθηκε ήταν ένας Live χάρτης ο οποίος θα έδειχνε σε πραγματικό χρόνο για μία ταινία τις χώρες από τις οποίες κατεβάζεται πειρατικά. Αυτός ο στόχος δεν υλοποιήθηκε καθώς για την παρακολούθηση τέτοιων δεδομένων, έπρεπε η εφαρμογή να συμμετέχει σε μια τέτοια παράνομη ενέργεια και προτιμήθηκε η απόρριψη της. Για την περάτωση αυτής της λειτουργίας θα έπρεπε ο Server να υλοποιήσει το πρωτόκολλο Torrent, και να συμμετέχει στον παράνομο διαμοιρασμό των ταινιών έτσι ώστε να μπορεί να αναγνωρίσει μέσω των διευθύνσεων IP από ποιές χώρες κατεβάζονται πειρατικά οι ταινίες. Πολλοί Torrent Tracker καταπολεμούν την ”παρακολούθηση” αρνούμενοι να δώσουν Peers σε Clients οι οποίοι δεν ζητάνε δεδομένα. Συνεπώς ο Client θα έπρεπε να κατεβάζει τα αρχεία αλλά και να αποθηκεύει τα δεδομένα καθώς οι Trackers ζητάνε και επαλήθευσή δεδομένων για να βεβαιωθούν ότι οι Clients τα κατεβάζουν. Πέρα από αυτό ο χειρισμός τέτοιων δεδομένων, όπως διευθύνσεις IP, ήθελαν εξαιρετικά ιδιαίτερο χειρισμό ειδικά με την νέα ευρωπαϊκή νομοθεσία GDPR. Όλοι οι άλλοι στόχοι υλοποιήθηκαν με επιτυχία και υπάρχει ένα πολύ καλό αποτέλεσμα δείχνοντας ακριβώς γιατί είναι απαραίτητο πολλές εταιρίες να υιοθετήσουν το μοντέλο των OpenData. Όσα περισσότερα δεδομένα κυκλοφορούν ελεύθερα στο διαδίκτυο τόσες περισσότερες ιδέες και εργαλεία θα δημιουργούνται καθημερινά για την αξιοποίηση τους.

7.2 Προτάσεις για εξέλιξη

Όπως προαναφέρθηκε, η εφαρμογή αυτή είναι λειτουργική αλλά δεν είναι σε καμία περίπτωση έτοιμη για ένα Production περιβάλλον. Παρακάτω παρατίθενται κάποιες προτάσεις για την εξέλιξη του εν λόγω έργου και μελλοντικών βελτιστοποιήσεων.

- Καθώς η παρούσα υλοποίηση δεν υποστηρίζει Scaling, θα μπορούσε μελλοντικά να το υποστηρίξει χρησιμοποιώντας το μοντέλο των Microservices.
- Θα μπορούσε να αλλάξει ριζικά ο αλγόριθμος υπολογισμού δεδομένων για να επιτρέπει την ενημέρωση των δεδομένων αντί για την συνολική επεξεργασία τους από την αρχή.

- Θα μπορούσε να δημιουργηθεί μια επιπρόσθετη λειτουργία στην υπηρεσία της εισαγωγής δεδομένων, έτσι ώστε να ελέγχει ανά τακτά χρονικά διαστήματα αν υπάρχουν νέα δεδομένα και να τα εισάγει αυτοματοποιημένα.

7.3 Ο κώδικας

Ο Κώδικας της εφαρμογής φιλοξενείται στο GitHub στην σελίδα:
https://github.com/ProIcons/MovieInsights

Γλωσσάρι

TCP Transmission Control Protocol

DSL Domain Specific Language

JSON JavaScript Object Notation

XML eXtensible Markup Language

IoC Inversion of Control

Java EE Java Enterprisse Edition

SPA Single Page Application

API Application Programming Interface

IoC Inversion of Control

TSV Tab-Separated Values

JVM Java Virtual Machine

CRUD **C**reate, **R**ead, **U**pdate and **D**elete

Βιβλιογραφία

Open Knowledge Foundation (2020). *Open data*. URL: https://github.com/okfn/opendatahandbook/blob/fbaa8d34ce1095741468 7f36a7fbfab8a733b6guide/el/what-is-open-data/index.md (επίσκεψη 20/10/2020).

Wikipedia - ElasticSearch (2020). *ElasticSearch — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Elasticsearch&oldid=978838583 (επίσκεψη 03/10/2020).

Wikipedia - Open Data (2020). *Open data — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Open_data&oldid=981085279 (επίσκεψη 03/10/2020).

Wikipedia - Spring Framework (2020). *Spring Framework — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Spring_Framework&oldid=977829427 (επίσκεψη 03/10/2020).

Η εργασία αυτή στοιχειοθετήθηκε με το πρόγραμμα XeLaTeX. Για τη στοιχειοθέτηση της βιβλιογραφίας χρησιμοποιήθηκε το πρόγραμμα `biber` και `biblatex`. Οι γραμματοσειρές που χρησιμοποιήθηκαν είναι οι Times New Roman και `Courier New`.