ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

"Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάση επιστημονικής παράφρασης.

Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων.

Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δε μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας."

Όνομα και Επώνυμο Συγγραφέα (Με Κεφαλαία): .....

Υπογραφή (Ολογράφως, χωρίς μονογραφή): .....

Ημερομηνία (Ημέρα – Μήνας – Έτος): .....

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1 ΠΕΡΙΛΗΨΗ

Καθημερινά δημιουργούμε, αποθηκεύουμε και εκτιθόμαστε σε ένα ασύλληπτο ποσό πληροφοριών ψηφιακής μορφής. Η πληροφοριακή υπερφόρτωση είναι ένα πρόβλημα όπου πολλοί χρήστες δεν έχουν τα μέσα να ανταπεξέλθουν. Το Φιλτράρισμα πληροφορίας είναι μια τεχνική η οποία έχει αρχίσει να εξελίσσεται τα τελευταία χρόνια με σκοπό να καταπολεμήσει την πληροφοριακή υπερφόρτωση.

Σκοπός αυτής της πτυχιακής είναι η σχεδίαση και ανάπτυξη ενός συστήματος εξατομίκευσης. Δηλαδή μια πλατφόρμα που μπορεί να προσαρμόζεται στα δεδομένα του χρήστη. Σε αυτή την περίπτωση, τα δεδομένα θα είναι ταινίες.

Αρχικά ο χρήστης βαθμολογεί ταινίες που γνωρίζει ήδη, χρησιμοποιώντας την λειτουργία αναζήτησης. Η πλατφόρμα θα επεξεργαστεί αυτά τα δεδομένα και θα παρουσιάσει στον χρήστη μια συλλογή από ταινίες που μπορεί να μην έχει δει, μαζί με την πιθανή βαθμολογία που θα τους έβαζε ο χρήστης.

Στις επόμενες σελίδες θα αναλύσουμε τα στάδια της σχεδίασης και ανάπτυξης του λογισμικού. Ξεκινώντας από την διεπαφή χρήστη, ύστερα την υποδομή του συστήματος και τέλος τον πυρήνα του συστήματος εξατομίκευσης.

Χρησιμοποιήθηκε η βάση δεδομένων της MovieLens (20Mil).

2 ΕΞΑΤΟΜΙΚΕΥΣΗ

2.1 Η ΑΝΑΓΚΗ ΓΙΑ ΕΞΑΤΟΜΙΚΕΥΣΗ

Κατά την δημιουργία μιας υπηρεσίας ή προϊόντος συναντιούνται πολλοί κύκλοι παραγωγής. Το προϊόν θα αλλάξει πολλές φορές μορφή και λειτουργικότητα μέχρι να φτάσει στα χέρια του τελικού χρήστη. Ακόμα και τότε, αφού οι δημιουργοί του προϊόντος λάβουνε σχόλια (feedback) από τους χρήστες, θα υπάρξουν αλλαγές μέχρι ένα σημαντικό ποσοστό των χρηστών είναι ικανοποιημένο. Παρατηρείται ότι είναι πολύ δύσκολο, και πολλές φορές αδύνατο, να ικανοποιηθούν όλοι οι τελικοί χρήστες από ένα προϊόν.

2.2 ΟΡΙΣΜΟΣ

Εξατομίκευση είναι η δυνατότητα προσαρμογής μιας υπηρεσίας ή ενός προϊόντος στις ανάγκες ενός ατόμου ή ομάδας ατόμων. Το προϊόν μπορεί να παρουσιάσει διαφορετικές πληροφορίες με διαφορετικό τρόπο, προσφέροντας διαφορετικές λειτουργείες ανάλογα με τα δεδομένα που έχει συλλέξει για τον χρήστη.

2.3 ΕΙΔΗ ΕΞΑΤΟΜΙΚΕΥΣΗΣ

2.3.1 ΆΜΕΣΗ

Όταν ένα προϊόν χρησιμοποιεί άμεση εξατομίκευση, παρέχει στον χρήστη επιλογές και ρυθμίσεις ώστε να το προσαρμόσει ο ίδιος στις ανάγκες του. Φυσικά το προϊόν θα πρέπει να παρέχει αρκετές επιλογές και ρυθμίσεις ώστε να καλύπτει ένα σημαντικό ποσοστό των χρηστών.

Η άμεση εξατομίκευση είναι πιο εύκολη να υλοποιηθεί και να ελεγχθεί, μιας και όλες οι πιθανές εκδοχές του προϊόντος είναι γνώστες στους σχεδιαστές, πριν αυτό παρουσιαστεί στον χρήστη.

2.3.2 ΈΜΜΕΣΗ

Όταν ένα προϊόν χρησιμοποιεί έμμεση εξατομίκευση, συγκεντρώνει στοιχεία από το περιβάλλον του. Επεξεργάζεται και αναλύει αυτά τα στοιχεία, χρησιμοποιώντας συστήματα προτάσεων και φιλτραρίσματος πληροφορίας, ώστε να προσαρμοστεί στις ανάγκες του χρήστη. Τα δεδομένα αυτά περιλαμβάνουν:

- Συμπεριφορά χρήστη
- Δυνατότητες περιβάλλοντος (π.χ. λειτουργικό σύστημα)
- Ιστορικό λειτουργίας (π.χ. αναζητήσεις χρήστη)
- Αξιολογήσεις χρήστη

2.3.3 ΥΒΡΙΔΙΑ

Προϊόντα που χρησιμοποιούν υβρίδια εξατομίκευση, χρησιμοποιούν κάποιο συνδυασμό των παραπάνω μεθόδων.

3 ΣΥΣΤΗΜΑΤΑ ΠΡΟΤΑΣΕΩΝ

3.1 Η ΑΝΑΓΚΗ ΓΙΑ ΣΥΣΤΗΜΑΤΑ ΠΡΟΤΑΣΕΩΝ

Καθημερινά δημιουργούμε, αποθηκεύουμε και εκτιθόμαστε σε ένα ασύλληπτο ποσό πληροφοριών ψηφιακής μορφής. Όταν η ποσότητα πληροφορίας σε ένα σύστημα ξεπερνάει την δυνατότητα του να την επεξεργαστεί, το σύστημα πάσχει από πληροφοριακή υπερφόρτωση. Η πληροφοριακή υπερφόρτωση είναι ένα πρόβλημα όπου οι χρήστες του διαδικτύου δεν έχουν τα μέσα να χειριστούν.

Όλες οι ψηφιακές πλατφόρμες που προσφέρουν πληθώρα περιεχομένου, υποφέρουν έμμεσα από την πληροφοριακή υπερφόρτωση. Σκοπός τους είναι να προσφέρουν στον χρήστη αυτό που χρειάζεται, την κατάλληλη στιγμή. Αν η πλατφόρμα δεν μπορεί να παρουσιάσει στον χρήστη την σωστή πληροφορία (π.χ. προϊόν, υπηρεσία, άρθρο, βίντεο), είναι πολύ πιθανό ότι ο χρήστης θα αναφερθεί σε κάποιον ανταγωνιστή για την ίδια πληροφορία.

3.2 ΟΡΙΣΜΟΣ

Συστήματα Προτάσεων είναι εργαλεία λογισμικού ή τεχνικές που προσφέρουν προτάσεις για αντικείμενα που μπορεί να χρειάζονται από τον χρήστη [3]. Αποτελούνται από την διαδικασία συλλογής, οργάνωσης και αποθήκευσης πληροφοριών σχετικά με τους χρήστες μιας πλατφόρμας, ανάλυση των πληροφοριών αυτών και παρουσίαση σε κάθε χρήστη της σωστής πληροφορίας στον σωστό χρόνο [5]. Οι προτάσεις σχετίζονται με διάφορες δραστηριότητες του χρήστη, όπως ποια αντικείμενα να αγοράσει, τι μουσική να ακούσει ή ποια νέα να διαβάσει.

3.3 ΜΕΘΟΔΟΙ ΥΛΟΠΟΙΗΣΗΣ

3.3.1 ΣΥΝΕΡΓΑΤΙΚΟ ΦΙΛΤΡΑΡΙΣΜΑ (COLLABORATIVE FILTERING)

Θα αναλυθούν περαιτέρω σε δικό τους κεφάλαιο.

3.3.2 ΒΑΣΗ ΠΕΡΙΕΧΟΜΕΝΟΥ (CONTENT-BASED)

Αυτά τα συστήματα βασίζονται στις περιγραφές, τίτλους και λέξεις κλειδιά που συσχετίζονται με κάθε αντικείμενο. Για κάθε χρήστη χτίζεται ένα προφίλ με τις λέξεις κλειδιά που τον ενδιαφέρουν καθώς χρησιμοποιεί την πλατφόρμα.

Όποτε ο χρήστης αλληλοεπιδρά με ένα αντικείμενο, το σύστημα προσθέτει τις λέξεις κλειδιά του αντικειμένου στο προφίλ του χρήστη μαζί με ένα βάρος για κάθε λέξη. Καθώς ο χρήστης επισκέπτεται και άλλα αντικείμενα, τα βάρη μεταβάλλονται ανάλογα με την συχνότητα εμφάνισης των λέξεων. Οι λέξεις με τα μεγαλύτερα βάρη χρησιμοποιούνται για να παρέχουν προτάσεις.

3.3.3 ΥΒΡΙΔΙΑ

Συστήματα που χρησιμοποιούν κάποιο συνδυασμό των παραπάνω μεθόδων.

3.4 ΑΞΙΟΛΟΓΗΣΗ ΣΥΣΤΗΜΑΤΩΝ ΠΡΟΤΑΣΕΩΝ

Υπάρχουν 3 τρόποι αξιολόγησης που μας επιτρέπουν να ελέγξουμε το πόσο αποτελεσματικό είναι κάποιο σύστημα προτάσεων, και για να συγκρίνουμε διαφορετικές μεθόδους.

3.4.1 ΜΕΛΕΤΕΣ ΧΡΗΣΤΩΝ (USER STUDIES)

Οι μελέτες χρηστών, συνήθως εκτελούνται σε μικρό αριθμό ανθρώπων [8]. Σε ένα γκρουπ 200-300 ατόμων παρουσιάζονται προτάσεις από διαφορετικά συστήματα. Οι χρήστες καλούνται να διαλέξουν πιο σύστημα παρείχε τις καλύτερες προτάσεις.

3.4.2 A/B ΤΕΣΤ (ONLINE EVALUATION)

Τα τεστ γίνονται σε χιλιάδες άτομα χρησιμοποιώντας αληθινά προϊόντα [8]. Όταν ο χρήστης επισκέπτεται την πλατφόρμα, το σύστημα διαλέγει τυχαία με ποια μέθοδο θα παρουσιάσει τα προτεινόμενα στοιχεία. Η αποτελεσματικότητα μετριέται με πιο έμμεσους δείκτες όπως το ποσοστό χρηστών που τελικά αγόρασαν το προϊόν.

3.4.3 ΑΞΙΟΛΟΓΗΣΗ ΕΚΤΟΣ ΣΥΝΔΕΣΗΣ (OFFLINE EVALUATION)

Οι αξιολογήσεις εκτός σύνδεσης χρησιμοποιούν ιστορικά στοιχεία (π.χ. βαθμολογίες που έβαλαν οι χρήστες σε ταινίες) [8]. Το σύστημα αξιολογείται με βάση το πόσο καλά μπορεί να προσεγγίσει τις πραγματικές βαθμολογίες των ιστορικών δεδομένων, έχοντας πρόσβαση μόνο σε ένα μέρος τους.

4 ΣΥΝΕΡΓΑΤΙΚΟ ΦΙΛΤΡΑΡΙΣΜΑ (COLLABORATIVE FILTERING)

4.1 ΟΡΙΣΜΟΣ

Είναι μια τεχνική που χρησιμοποιείται από τα συστήματα προτάσεων [3]. Ορίζεται ως η διαδικασία φιλτραρίσματος πληροφορίας η οποία χρησιμοποιεί πολλαπλούς παράγοντες, οπτικές γωνίες και πηγές δεδομένων [4].

4.2 ΜΕΘΟΔΟΙ ΥΛΟΠΟΙΗΣΗΣ

4.2.1 ΒΑΣΗ ΜΝΗΜΗΣ (MEMORY-BASED)

Αυτή η μέθοδος χρησιμοποιεί τις βαθμολογίες των χρηστών πάνω σε αντικείμενα. Κάθε χρήστης ανήκει σε μία ομάδα χρηστών με παρόμοιες βαθμολογίες. Χρησιμοποιώντας όλα ή μέρος αυτών των δεδομένων μπορεί να πράξει προτάσεις. Τα βήματα που ακολουθεί είναι τα εξής [7]:

- Υπολογίζει το βάρος ή την απόσταση των χρηστών.
- Παράγει μια πρόταση χρησιμοποιώντας το ζυγισμένο μέσο όλων των βαθμολογιών του χρήστη.

Μπορούμε να επιλέξουμε διάφορους μεθόδους για να χρησιμοποιήσουμε ως μέτρο απόστασης χρηστών.

4.2.1.1 ΒΑΣΗ ΣΥΣΧΕΤΗΣΗΣ (CORRELATION-BASED)

Σε αυτήν την περίπτωση η ομοιότητα μεταξύ δυο χρηστών υπολογίζεται με την συσχέτιση του pearson ή άλλα μέτρα συσχέτισης. Η συσχέτιση του pearson

μετράει το πόσο δύο μεταβλητές είναι γραμμικά όμοιες. Για έναν αλγόριθμο που συγκρίνει χρήστες, ο pearson ορίζεται ως:

$$simil(x,y) = \frac{\sum_{i \in Ixy}(r_{x,i} - \overline{r_x})(r_{y,i} - \overline{r_y})}{\sqrt{\sum_{i \in Ixy}(r_{x,i} - \overline{r_x})^2}\sqrt{\sum_{i \in Ixy}(r_{y,i} - \overline{r_y})^2}}$$

Όπου x,y είναι οι δύο χρήστες, $r_{u,I}$ η βαθμολογία του χρήστη u στο αντικείμενο i, $\overline{r_u}$ ο μέσος όρος βαθμολογιών του χρήστη u, I_{xy} είναι αντικείμενα που έχουν βαθμολογήσει και οι δύο χρήστες.

Άλλα μέτρα συσχέτισης περιλαμβάνουν:

- Spearman
- Kendal's
- Περιορισμένος Pearson

1. ΒΑΣΗ ΔΥΑΝΙΣΜΑΤΙΚΟΥ ΣΥΝΗΜΙΤΟΝΟΥ (VECTOR COSINE-BASED)

Η ομοιότητα μεταξύ δύο αντικειμένων μπορεί να υπολογιστεί αν θεωρήσουμε κάθε αντικείμενο ως ένα διάνυσμα από συχνότητες βαθμολογιών και υπολογίσουμε το συνημίτονο της γωνίας που δημιουργείται από τα διανύσματα [7].

Θεωρούμε κάθε ζευγάρι βαθμολογιών των χρηστών x, y ως διανύσματα. Στον άξονα χ τοποθετούνται οι βαθμολογίες του χρήστη χ και στον άξονα y οι βαθμολογίες του y αντίστοιχα. Κάθε ζευγάρι βαθμολογιών αντιπροσωπεύει μια ταινία με την μορφή διανύσματος. Αν πολλαπλασιάσουμε τα συνημίτονα των γωνιών που σχηματίζουν τα διανύσματα μεταξύ τους, λαμβάνουμε έναν αριθμό από το -1 μέχρι το 1, όπου -1 σημαίνει ότι οι χρήστες είναι ανόμοιοι και 1 ότι είναι όμοιοι.

Για παράδειγμα, αν δυο χρήστες x,y βαθμολόγησαν δυο αντικείμενα 1,2

$$\text{Simil}(x,y) = \frac{(r_{x,1} * r_{x,2}) + (r_{y,1} * r_{y,2})}{\sqrt{(r_{x,1})^2 + (r_{y,1})^2}\sqrt{(r_{x,2})^2 + (r_{y,2})^2}}$$

Όπου $r_{x,I}$ ο βαθμός του χρήστη χ για το αντικείμενο i.

4.2.2 ΒΑΣΗ ΜΟΝΤΕΛΟΥ (MODEL-BASED)

Αυτή η μέθοδος χρησιμοποιεί διάφορους αλγορίθμους που επιτρέπουν στο σύστημα να μαθαίνει και να αναγνωρίζει ομοιότητες χρησιμοποιώντας εκπαιδευτικά δεδομένα [7]. Περιλαμβάνονται μέθοδοι όπως:

- Νευρονικά δίκτυα
- Συσταδοποίηση
- Βάση αναδρομής

4.3 ΠΡΟΒΛΗΜΑΤΑ

4.3.1 ΈΛΛΕΙΨΗ ΔΕΔΟΜΕΝΩΝ (COLD START)

Το πρόβλημα εμφανίζεται όταν το σύστημα δεν έχει συλλέξει αρκετά στοιχεία για τον χρήστη. Έτσι η σύγκριση του με τα ιστορικά δεδομένα είναι αδύνατη [3].

Μία λύση σε αυτό το πρόβλημα είναι να προταθούν στον χρήστη τα πιο δημοφιλή αντικείμενα μέχρι να συλλεχθούν αρκετές πληροφορίες γι’ αυτόν.

4.3.2 ΕΠΙΘΕΣΕΙΣ

Είναι δυνατόν κάποιος να χρησιμοποιήσει το σύστημα για να προωθήσει ένα προϊόν [3] παρέχοντας πολλές θετικές βαθμολογίες στο προϊόν αυτό και αρνητικές στους ανταγωνιστές.

4.3.3 ΕΠΕΚΤΑΣΗ

Καθώς συλλέγονται παραπάνω βαθμολογίες και στοιχεία για μια πλατφόρμα, η επεξεργασία τους γίνεται όλο και πιο δύσκολη [3]. Όποτε συνδέεται ένας νέος χρήστης στο σύστημα, πρέπει να συγκρίνεται με όλους τους προηγούμενους.

4.3.4 ΓΚΡΙ ΠΡΟΒΑΤΑ

Γκρι πρόβατα θεωρούνται οι χρήστες όπου οι βαθμολογίες τους δεν συμβαδίζουν με καμία ομάδα χρηστών και έτσι η παροχή προτεινόμενων σε αυτούς τους χρήστες είναι σχεδόν αδύνατη.

Επίσης η ύπαρξη των αξιολογήσεων τους μέσα στα δεδομένα της πλατφόρμας έχει ως παρενέργεια την παραγωγή λάθος προτεινόμενων για τους υπόλοιπους χρήστες.

4.4 Ο ΔΙΑΓΩΝΙΣΜΟΣ ΤΟΥ NETFLIX

Τον Οκτώβριο του 2006, το NETFLIX ανακοίνωσε ένα διαγωνισμό με σκοπό να βρει καλύτερους τρόπους παραγωγής προτεινόμενων ταινιών. Θέτοντας το έπαθλο στα ένα εκατομμύριο δολάρια, χιλιάδες επιστήμονες δέχτηκαν να ανταγωνιστούν. Ως δεδομένα χρησιμοποίησαν ένα τεράστιο αρχείο από την βάση δεδομένων του NETFLIX με 480,000 χρήστες και 17,770 ταινίες.

5 ΥΛΟΠΟΙΗΣΗ

5.1 ΔΟΜΗ ΔΕΔΟΜΕΝΩΝ

5.1.1 ΑΠΛΗ

Αυτή η δομή χρησιμοποιείται για να οργανωθούν τα δεδομένα μέσα στην μνήμη με ένα τρόπο που μπορούμε να τον διαχειριστούμε με ελάχιστο κώδικα. Όλες οι δομές παρέχουν έναν bool operator<(...) για να μπορούν να ταξινομηθούν μέσα στα set.

Οι κύριες δομές είναι USER, RATING.

Η δομή USER αντιπροσωπεύει ένα χρήστη της βάσης. Μέσα του έχει:

- Την λίστα με τις βαθμολογίες του
- Τον μέσο όρο των βαθμολογιών του
- Το κωδικό του χρήστη στην βάση

```
struct user {
        unsigned id;
        mutable double avg;
        mutable set<rating> list;

        user(const unsigned &id)
                : id(id)
        {}
};
inline bool operator<(const user &left, const user &right) {
        return left.id < right.id;
}
```

Η δομή RATING αντιπροσωπεύει μια βαθμολογία ενός χρήστη της βάσης προς μια ταινία. Μέσα της έχει:

- Το κωδικό της βαθμολογίας του χρήστη
- Τον βαθμό της βαθμολογίας [1-10]

```
struct rating {
        unsigned id;
        mutable double value;
};
inline bool operator==(const rating &left, const rating &right) {
        return left.id == right.id;
}
inline bool operator<(const rating &left, const rating &right) {
        return left.id < right.id;
}
```

5.1.2 ΣΥΜΠΥΚΝΩΜΕΝΗ

Αυτή η δομή χρησιμοποιείται κατά τον υπολογισμό των προτεινομένων ταινιών και δημιουργήθηκε για να επιταχύνει αυτό τον υπολογισμό. Αποτελείται από 2 πίνακες.

Ο πρώτος είναι ένας δισδιάστατος πίνακας θετικών ακέραιων που αντιπροσωπεύουν τις βαθμολογίες των χρηστών. Η σειρά του πίνακα είναι ο εσωτερικός κωδικός του χρήστη και κάθε στήλη είναι μια βαθμολογία του. Οι βαθμολογίες φυλάσσονται ως θετικοί ακέραιοι όπου τα πρώτα bit αντιπροσωπεύουν τον εσωτερικό κωδικό της ταινίας και τα 4 τελευταία την βαθμολογία.

Ο δεύτερος πίνακας περιέχει τον μέσο όρο κάθε χρήστη.

Η δομή αναλύεται περαιτέρω στην ανάλυση του κώδικα της σελίδας compressor.h

5.2 ΔΙΕΠΑΦΗ ΧΡΗΣΤΗ (FRONT-END)

5.2.1 ΕΠΙΛΟΓΕΣ ΛΟΓΙΣΜΙΚΟΥ

5.2.1.1 BOOTSTRAP

Είναι ένα front-end framework ανοικτού κώδικα σχεδιασμένο ώστε να κάνει απλούστερη τη διαδικασία δημιουργίας του εικαστικού μιας ιστοσελίδας. Εδώ χρησιμοποιήθηκε για να εκμεταλλευτούμε τις προσχεδιασμένες του responsive css κλάσεις.

5.2.1.2 JQUERY

Είναι ένα JavaScript framework ανοικτού κώδικα σχεδιασμένο να κάνει πιο εύκολη τη διαχείριση του HTML και CSS σε μια ιστοσελίδα.

5.2.2 ΣΧΕΔΙΑΣΗ

Αρχικά επέλεξα τις δυνατότητες στις οποίες πρέπει να έχει πρόσβαση ο χρήστης:

- Αναζήτηση
- Βαθμολογία
- Προβολή βαθμολογημένων
- Καθαρισμός βαθμολογημένων
- Αλλαγή παραμέτρων
- Προβολή προτεινόμενων
- Βαθμολόγηση των προτεινόμενων

Κατά την υλοποίηση χρησιμοποιήθηκε το λογισμικό GIT, το οποίο αποθηκεύει προηγούμενες καταστάσεις των αρχείων κώδικα. Ακολουθούν μερικά αποσπάσματα του ιστορικού.

1)

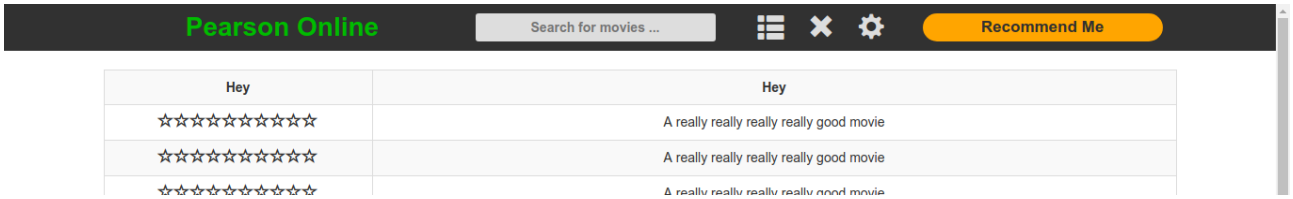
2)

3)

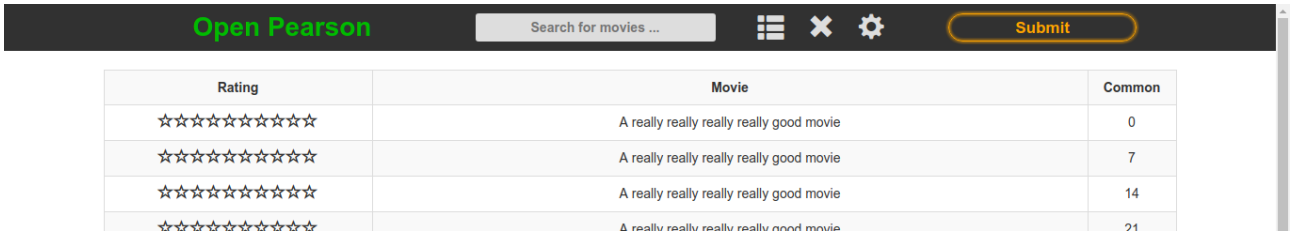
4)


This site
Upload

5)

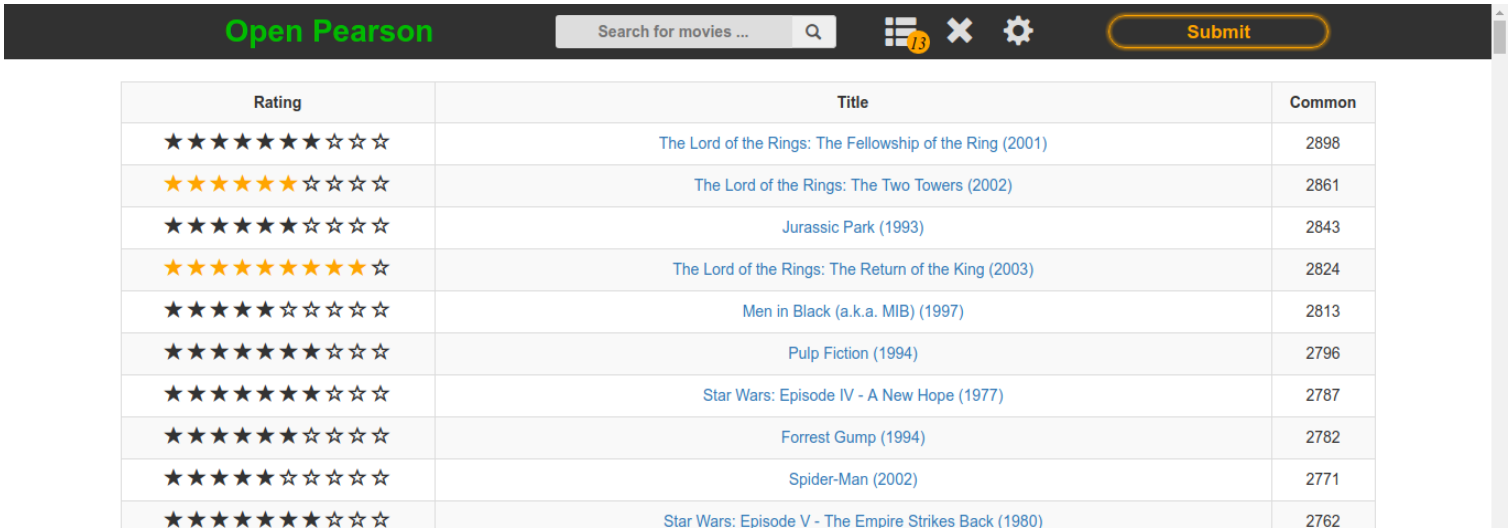

Pearson Online
Search for movies ...
Recommend Me
Hey
Hey
A really really really really good movie
A really really really really good movie

6)


Open Pearson
Search for movies ...
Submit
Rating
Movie
Common
A really really really really good movie
0
A really really really really good movie
7
A really really really really good movie
14

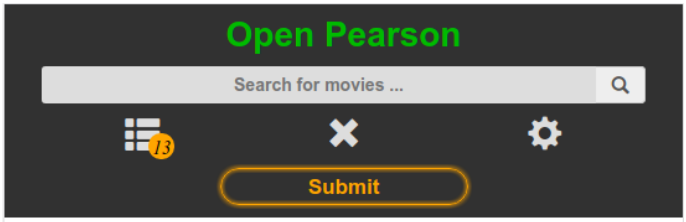
5.2.3 ΤΕΛΙΚΟ ΕΙΚΑΣΤΙΚΟ

5.2.3.1 DESKTOP



Rating	Title	Common
★★★★★★★★☆☆	The Lord of the Rings: The Fellowship of the Ring (2001)	2898
★★★★★★☆☆☆☆	The Lord of the Rings: The Two Towers (2002)	2861
★★★★★★☆☆☆☆	Jurassic Park (1993)	2843
★★★★★★★★★☆	The Lord of the Rings: The Return of the King (2003)	2824
★★★★★☆☆☆☆☆	Men in Black (a.k.a. MIB) (1997)	2813
★★★★★★★☆☆☆	Pulp Fiction (1994)	2796
★★★★★★★☆☆☆	Star Wars: Episode IV - A New Hope (1977)	2787
★★★★★★☆☆☆☆	Forrest Gump (1994)	2782
★★★★★☆☆☆☆☆	Spider-Man (2002)	2771
★★★★★★★☆☆☆	Star Wars: Episode V - The Empire Strikes Back (1980)	2762

5.2.3.2 MOBILE



Rating	Title	Common
★★★★★ ★★☆☆☆	The Lord of the Rings: The Fellowship of the Ring (2001)	2898
★★★★★ ★☆☆☆☆	The Lord of the Rings: The Two Towers (2002)	2861
★★★★★ ★☆☆☆☆	Jurassic Park (1993)	2843
★★★★★ ★★★★☆	The Lord of the Rings: The Return of the King (2003)	2824
★★★★★ ☆☆☆☆☆	Men in Black (a.k.a. MIB) (1997)	2813

5.3 ΤΟΠΙΚΟ ΣΥΣΤΗΜΑ (BACK-END)

5.3.1 ΕΠΙΛΟΓΕΣ ΛΟΓΙΣΜΙΚΟΥ

5.3.1.1 ΠΡΩΤΟΚΟΛΛΟ ΕΠΙΚΟΙΝΩΝΙΑΣ

Ως πρωτόκολλο επικοινωνίας μεταξύ του εκτελέσιμου και του εξυπηρετητή επιλέχθηκε το Fast Common Gateway Interface (FCGI). Είναι μια παραλλαγή στο Common Gateway Interface (CGI) με τη διαφορά ότι δεν δημιουργεί νέες διεργασίες για κάθε νέο συνδεδεμένο χρήστη. Έτσι χρειάζεται να φορτώνει τα δεδομένα της βάσης μόνο μια φορά.

Στην αρχή επιλέγουμε τον αριθμό των παράλληλων διεργασιών που θέλουμε και όταν μια διεργασία εξυπηρετήσει έναν χρήστη, ξεκινάει να εξυπηρετεί τον επόμενο χωρίς να ξαναφορτώσει τα δεδομένα που χρειάζεται.

5.3.1.2 ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Το πρόγραμμα έχει γραφτεί στην γλώσσα προγραμματισμού C++ λόγω του καθαρού προβαδίσματος της στον χρόνο εκτέλεσης σε σχέση με άλλες γλώσσες. Χρησιμοποιούνται οι βιβλιοθήκες: libfcgi++, nlohmann-json.

5.3.1.3 ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

Χρησιμοποιήθηκε η PostgreSQL, κυρίως γιατί θεωρείται η πιο γρήγορη και πιο ασφαλής SQL βάση δεδομένων. Εκεί βρίσκονται 2 πίνακες: ratings και movies. Χρησιμοποιήθηκαν τα δεδομένα της MovieLens (20M).

Στον πίνακα ratings υπάρχουν όλα τα ratings από το 20M Dataset της MovieLens. Η μορφή των εγγραφών είναι [χρήστης, ταινία, τιμή].

Στον πίνακα movies υπάρχουν χαρακτηριστικά που βοηθάνε στην αλληλεπίδραση του χρήστη με την πλατφόρμα, όπως το όνομα της ταινίας και ο κωδικός της στο IMDB, ώστε αν ο χρήστης θέλει να μάθει κάτι για μια ταινία που του προτείνεται μπορεί απλά να κάνει κλικ στον τίτλο της και θα ανακατευθυνθεί στη σελίδα της ταινίας στο IMDB.

5.3.1.4 ΕΞΥΠΗΡΕΤΗΤΗΣ

Χρησιμοποιήθηκε ο NGINX. Ο κύριος λόγος είναι ότι ήταν πολύ εύκολο να ρυθμιστεί και να συνδεθεί με τα υπόλοιπα κομμάτια του τοπικού συστήματος (backend). Ο NGINX έχει ρυθμιστεί ως μεσολαβητής για τα αιτήματα των χρηστών. Δηλαδή όλα τα αιτήματα δρομολογούνται στον φάκελο με τα εικαστικά στοιχεία (HTML, CSS, JS, PNG, ICO) και μόνο τα αιτήματα που χρειάζεται να πάνε στο FCGI, περνάνε σε αυτό.

5.3.2 ΕΠΕΞΗΓΗΣΗ ΤΟΥ ΚΩΔΙΚΑ

5.3.2.1MAIN.CPP

Στα προγράμματα γραμμένα σε C++, η συνάρτηση main θεωρείται σημείο εισαγωγής, κατά σύμβαση, και είναι η πρώτη συνάρτηση που εκτελείται. Εδώ η main είναι υπεύθυνη για:

- Να συνδέσει το FCGI με τη βάση δεδομένων.
- Να φέρει τα δεδομένα της SQL σε μια νέα δομή στη RAM η οποία θα χρησιμοποιηθεί για την ταχύτερη ανάλυση των δεδομένων.
- Να διαχειριστεί τυχόν σήματα του kernel (SIGINT).
- Να ξεκινήσει την επανάληψη που απαντάει στα αιτήματα του NGINX.

Αρχικά αναθέτει την signalHandler ως διαχειριστή του σήματος SIGINT για να ελευθερωθεί η κατειλημμένη μνήμη σε περίπτωση που υπάρξει αναγκαστική διακοπή του προγράμματος. Ύστερα φορτώνει όλα τα δεδομένα από την PostgreSQL στην αναλυτική δομή και δημιουργεί μερικές βοηθητικές δομές, οι οποίες θα μας χρειαστούν για την σωστή διαχείριση των δεδομένων της βάσης. Στη συνέχεια δημιουργεί την συμπιεσμένη δομή και ξεκινάει την FCGI επανάληψη που είναι υπεύθυνη να δέχεται τις αιτήσεις του NGINX.

Μόλις το FCGI δεχτεί μια αίτηση, όλα τα δεδομένα της περνάνε στη μεταβλητή request. Από εκεί χρησιμοποιούμε την κλάση ENVIRONMENT η οποία παίρνει τα δεδομένα από τη request και τα μετατρέπει σε δομές που μπορούμε να επεξεργαστούμε ευκολότερα με τη C++, όπως Map και String. Ύστερα καλείται η handle_request() η οποία θα αναλυθεί παρακάτω. Την απάντηση του FCGI την δίνουμε στο standard output, το οποίο έχουμε κάνει override με το stream της σύνδεσης του NGINX.

```
#include <iostream>
#include <memory>
#include <csignal>

#include "fcgio.h"

#include "classes.h"
#include "globals.h"

#include "WebServer/handle_request.h"
#include "WebServer/enviroment.h"
#include "Database/load_from_database.h"
#include "Utilities/id_handlers.h"
#include "Utilities/datastructure_handlers.h"
#include "Utilities/avg.h"
#include "Utilities/reverse_map.h"

void signalHandler(int signum) {

    cout << "\n\nInterrupt signal (" << signum << ") received.\n" << endl;

    users.clear();
    movies.clear();
    avg.clear();
    user_ids.clear();
    item_ids.clear();
    item_ids_reversed.clear();
    items_per_user.clear();

    exit(signum);
}

int main() {

    signal(SIGINT, signalHandler);
```

```
streambuf * cin_streambuf  = cin.rdbuf();
streambuf * cout_streambuf = cout.rdbuf();
streambuf * cerr_streambuf = cerr.rdbuf();

FCGX_Init();
FCGX_Request request;
FCGX_InitRequest(&request, 0, 0);

// Load data
users = get_users_from_db();
movies = get_movies_from_db();
cout << "Data Loaded!" << endl;

cout << "Creating helpers..." << flush;
user_ids = make_user_ids(users);
item_ids = make_item_ids(users);
item_ids_reversed = reverse_map(item_ids);
avg = get_avg(users, user_ids);
cout << "Done\n";

// Create table
cout << "Creating master table..." << flush;
items_per_user = get_items_per_user(users, user_ids, item_ids);
cout << "Done" << endl;

cout << "Structure initialized\n\n";
cout << "Ready for requests!\n\n";

while (FCGX_Accept_r(&request) == 0) {

    cout << "\n--==[ Recieved request ]==--\n";
    e = ENVIRONMENT(request);
    string response = handle_request();
    cout << "Response is ready!" << endl;
```

```
            fcgi_streambuf cin_fcgi_streambuf(request.in);
            fcgi_streambuf cout_fcgi_streambuf(request.out);
            fcgi_streambuf cerr_fcgi_streambuf(request.err);
            cin.rdbuf(&cin_fcgi_streambuf);
            cout.rdbuf(&cout_fcgi_streambuf);
            cerr.rdbuf(&cerr_fcgi_streambuf);

            cout << "Content-type: text/html\r\n\r\n"
                 << response;

            cin.rdbuf(cin_streambuf);
            cout.rdbuf(cout_streambuf);
            cerr.rdbuf(cerr_streambuf);

            cout << "Sending Response..." << endl;
        }

        return 0;
}
```

5.3.2.2 GLOBALS.H

Αυτό το αρχείο περιέχει σημαντικούς ορισμούς οι οποίοι χρησιμοποιούνται σε πολλά σημεία στο πρόγραμμα. Κάποιοι από αυτούς είναι:

- Η σύνδεση στη βάση δεδομένων
- Μακροεντολές
- Η απλή δομή
- Η συμπιεσμένη δομή
- Οι ρυθμίσεις που επέλεξε ο χρήστης

Πιο αναλυτικά:

bitmask - bitshift

Χρησιμοποιούνται κατά τη συμπίεση για τη συμπιεσμένη δομή δεδομένων.

min rating - max rating

Χρησιμοποιούνται κατά τον υπολογισμό της προτεινόμενης βαθμολογίας μιας ταινίας από τον χρήστη.

max response

Είναι ο μέγιστος αριθμός ταινιών που θα προταθούν στον χρήστη. Το όριο αυτό υπάρχει λόγω τεχνικών περιορισμών του υλικού που χρησιμοποιήθηκε για την υλοποίηση αυτής της εργασίας.

db

Είναι μια κλάση η οποία περιέχει τη σύνδεση στη βάση μαζί με χρήσιμες συναρτήσεις. Κατά τη δημιουργία της περνάμε το όνομα χρήστη και τον κωδικό της βάσης.

ENVIRONMENT e

Είναι τα δεδομένα της τρέχουσας αίτησης μετασχηματισμένα σε πιο ευκολόχρηστες δομές στη c++, όπως Map και String.

AVG

Ο πίνακας που κρατάει το μέσο όρο των βαθμολογιών κάθε χρήστη.

Pearson – movies – recommended

Είναι οι ρυθμίσεις που περνάει ο χρήστης από το site.

Οι δομές δεδομένων.

```
#ifndef GLOBALS_H
#define GLOBALS_H

#define BIT_MASK 0b1111
#define BITSHIFT 4
#define MIN_RATING 1
#define MAX_RATING 10

#define MAX_RESPONSE 1000

#include "classes.h"
#include "WebServer/environment.h"

my_db db("curious_scavenger", "badpa55.64");

ENVIRONMENT e;

set<user> users;
set<movie> movies;

vector<double> avg;

map<unsigned, unsigned> user_ids;
map<unsigned, unsigned> item_ids;
map<unsigned, unsigned> item_ids_reversed;

vector<vector<unsigned>> items_per_user;

double MIN_PEARSON;
unsigned MIN_COMMON_MOVIES;
unsigned MIN_RECOMMENDED;

#endif // GLOBALS_H
```

5.3.2.3 CLASSES.H

Εδώ δηλώνονται όλες οι κλάσεις του προγράμματος. Τοποθετήθηκαν όλες μαζί γιατί χρησιμοποιούνται σχεδόν πάντα όλες μαζί σε όλα τα άλλα αρχεία, έτσι μπορούμε να έχουμε μόνο μια #include σε κάθε αρχείο.

my_db

Χρησιμοποιείται με το μοτίβο singleton και αντιπροσωπεύει τη σύνδεση στη βάση δεδομένων PostgreSQL. Οπότε το πρόγραμμα χρειάζεται κάτι από τη βάση, το ζητάει μέσω αυτής της κλάσης.

Rating - Movie

Αντιπροσωπεύει μια αξιολόγηση ενός χρήστη για μια ταινία και μια ταινία αντίστοιχα. Χρησιμοποιείται μόνο για να μαζέψουμε σχετικές πληροφορίες σε ένα σημείο και γι’ αυτό δεν περιέχει μεθόδους.

User

Αυτή είναι η δεύτερη πιο σημαντική δομή στο πρόγραμμα. Περιέχει στοιχεία ενός χρήστη. Συμπεριλαμβάνει τον κωδικό του, τον μέσο όρο βαθμολογιών και μια λίστα με τις ταινίες που έχει βαθμολογήσει.

returning

Περιέχει τα αποτελέσματα του υπολογισμού προτεινόμενων ταινιών. Αυτά τα δεδομένα θα σταλούν στον χρήστη.

```
#ifndef CLASSES_H
#define CLASSES_H

#include <pqxx/pqxx>
#include <map>
#include <set>

using namespace std;
using namespace pqxx;

class my_db {

    connection *conn;

public:

    my_db(const string &username, const string &password) {

        conn = new connection(
            " user="     + username +
            " password=" + password +
            " host="     + "localhost" +
            " dbname="   + username
        );

        conn->prepare(
            "search",
            R"(
                select *
                from movies
                where
                    lower(name) like lower(
                        '%' || $1 || '%')
                limit 100
            )");
```

```
        }
        ~my_db() {
                delete conn;
        }

        connection *get_conn() { return conn; }
};

struct rating {

        unsigned id;
        mutable double value;
};
inline bool operator==(const rating &left, const rating &right) {
        return left.id == right.id;
}
inline bool operator<(const rating &left, const rating &right) {
        return left.id < right.id;
}

struct movie {

        unsigned id;
        string name;
        string imdb;
        string tmdb;

        movie(
                const unsigned &id,
                const string &name,
                const string &imdb,
                const string &tmdb)
                : id(id)
                , name(name)
```

```
            , imdb(imdb)
            , tmdb(tmdb)
        {}
};
inline bool operator<(const movie &left, const movie &right) {
        return left.id < right.id;
}

struct user {

        unsigned id;
        mutable double avg;
        mutable set<rating>::iterator hide_me;
        mutable set<rating> list;

        user(const unsigned &id)
            : id(id)
        {}
};
inline bool operator<(const user &left, const user &right) {
        return left.id < right.id;
}

struct returning {

        unsigned id;
        double value;
        size_t common_users;

        returning(
            const unsigned &id,
            const double &value,
            const size_t &common_users)
            : id(id)
```

```
            , value(value)
            , common_users(common_users)
        {}
};

#endif // CLASSES_H
```

5.3.2.4AVG.H

Εδώ περιέχονται οι συναρτήσεις που χρησιμοποιούνται για τον υπολογισμό του μέσου όρου των χρηστών.

find_avg_from_set

Δέχεται μια λίστα από βαθμολογίες και επιστρέφει τον μέσο όρο τους.

get_avg

Χρησιμοποιεί την **find_avg_from_set** και την εφαρμόζει πάνω στην λίστα κάθε χρήστη. Επιστρέφει ένα πίνακα με τον μέσο όρο όλων των χρηστών.

Η λειτουργία εύρεσης του μέσου όρου των χρηστών έχει χωριστεί σε 2 κομμάτια γιατί ένα μέρος της χρησιμοποιείται σε διάφορα μέρη του συστήματος, όπως κατά τον υπολογισμό του μέσου όρου του συνδεδεμένου χρήστη.

```
#ifndef AVG_H
#define AVG_H

#include "classes.h"
#include <set>

using namespace std;

double find_avg_from_set(const set<rating> &list) {

	double avg = 0;
	for (const rating &item : list) { avg += item.value; }
	return avg / (double) list.size();
}

vector<double> get_avg(
		const set<user> &users,
		const map<unsigned, unsigned> &user_ids)
{
	vector<double> avg(user_ids.size());

	for (const user &u : users) {
		u.avg = find_avg_from_set(u.list);
		avg[user_ids.find(u.id)->second] = u.avg;
	}

	return avg;
}

#endif // AVG_H
```

5.3.2.5 COMPRESSOR.H

Εδώ βρίσκουμε τη συνάρτηση που μας βοηθάει να συμπιέσουμε τα δεδομένα της βάσης σε ένα δισδιάστατο πίνακα από θετικούς ακέραιους.

compress

Χρησιμοποιείται για να συμπιέσουμε τον κωδικό μιας ταινίας και τη βαθμολογία που της έβαλε ο χρήστης μέσα σε έναν αριθμό. Ξεκινάμε με ένα θετικό ακέραιο που έχει μήκος 32bits.

Τοποθετούμε τον κωδικό της ταινίας μέσα στη μεταβλητή και την κάνουμε BITSHIFT κατά MACROHERE αριστερά. Στα MACROHERE κενά bit στα δεξιά, τοποθετούμε τη βαθμολογία του χρήστη.

get_id , get_value

Χρησιμοποιούνται για να εξάγουμε τις ανάλογες πληροφορίες από ένα συμπιεσμένο θετικό ακέραιο.

Παράδειγμα:

Συμπίεση:

0000000000000000

ID: 49

0000000000110001

Left Shift 4:

0000001100010000

Add rating 9:

0000001100011001

Αποσυμπίεση ID:

Right Shift 4:

0000000000110001

Αποσυμπίεση Value:

Mask 1111:

0000000000001001

Σε όλες τις συναρτήσεις υπάρχουν έλεγχοι (assert) ώστε αν ένας κωδικός ταινίας δεν χωράει μέσα στην συμπιεσμένη δομή, το πρόγραμμα σταματάει και ενημερώνει τον διαχειριστή για το λάθος.

```
#ifndef COMPRESSOR_H
#define COMPRESSOR_H

#include <cassert>
#include "globals.h"

inline unsigned compress(const unsigned &id, const double &value) {

    short integer = static_cast<short>(value);

    // has no decimal
    assert(value == integer);

    // can fit in bitmask
    assert(integer == (integer & BIT_MASK));

    // id can fit without bitmask
    assert(id == ((id << BITSHIFT) >> BITSHIFT));

    return (id << BITSHIFT) + (static_cast<short>(value) & BIT_MASK);
}

inline unsigned get_id(const unsigned &compressed) {

    return (compressed >> BITSHIFT);
}

inline double get_value(const unsigned &compressed) {

    assert((compressed & BIT_MASK) <= MAX_RATING);
    assert((compressed & BIT_MASK) >= MIN_RATING);

    return (compressed & BIT_MASK);
}

#endif // COMPRESSOR_H
```

5.3.2.6 DATASTRUCTURE_HANDLERS.H

Αυτές οι συναρτήσεις χρησιμοποιούνται για να μετατρέψουμε την αναλυτική δομή στη συμπιεσμένη δομή.

get_items_per_user

Δέχεται την απλή δομή και επιστρέφει ένα δισδιάστατο πίνακα όπου κάθε σειρά αντιπροσωπεύει έναν χρήστη και κάθε στήλη μια ταινία αυτού του χρήστη.

get_users_per_item

Δέχεται την απλή δομή και επιστρέφει ένα δισδιάστατο πίνακα όπου κάθε σειρά αντιπροσωπεύει μια ταινία και κάθε στήλη τους χρήστες που έχουν βαθμολογήσει αυτήν την ταινία.

```
#ifndef DATASTRUCTURE_HANDLERS_H
#define DATASTRUCTURE_HANDLERS_H

#include <vector>
#include <map>

#include "Utilities/compressor.h"

using namespace std;

vector<vector<unsigned>> get_items_per_user(
	const set<user> &users
	, map<unsigned, unsigned> &user_ids
	, map<unsigned, unsigned> &item_ids)
{

	vector<vector<unsigned>> items_per_user(user_ids.size());

	for (const user &u : users) {

		unsigned user_id = user_ids[u.id];

		for (const rating &r : u.list) {
			items_per_user[user_id]
			.push_back(compress(item_ids[r.id], r.value));
		}
	}

	return items_per_user;
}

vector<vector<unsigned>> get_users_per_item(
	const set<user> &users
	, const map<unsigned, unsigned> &user_ids
	, const map<unsigned, unsigned> &item_ids)
```

```
{

    vector<vector<unsigned>> users_per_item(item_ids.size());

    for (const user &u : users) {
        for (const rating &r : u.list) {
            users_per_item[item_ids.find(r.id)->second]
            .push_back(
              compress(
                user_ids.find(u.id)->second,
                r.value
              )
            );
        }
    }

    return users_per_item;
}

#endif // DATASTRUCTURE_HANDLERS_H
```

5.3.2.7ID_HANDLERS.H

Αυτές οι συναρτήσεις βοηθάνε στην επικοινωνία της βάσης με το πρόγραμμα. Τα δεδομένα στη βάση έχουν άγνωστη μορφή. Τα ID κάθε ταινίας μπορεί να είναι τυχαία.

Για να λύσουμε αυτό το πρόβλημα, αντιστοιχούμε κάθε ID της βάσης με ένα εσωτερικό ID το οποίο είμαστε σίγουροι ότι:

- ξεκινάει από το 0
- είναι συνεχές σε ένα πεδίο π.χ. [0, 38]

make_user_ids , make_item_ids

Δημιουργούν τα εσωτερικά ID για τους χρήστες και τις ταινίες αντίστοιχα.

Εισάγουν όλα τα αντικείμενα σε ένα map<unsigned, unsigned> και ύστερα, χρησιμοποιώντας έναν θετικό ακέραιο, ξεκινάει από το 0 και αναθέτει ένα νέο ID σε κάθε αντικείμενο στην λίστα.

```
#ifndef ID_HANDLERS_H
#define ID_HANDLERS_H

#include <map>
#include <set>
#include "classes.h"

using namespace std;

map<unsigned, unsigned> make_user_ids(const set<user> &users) {

	map<unsigned, unsigned> user_ids;

	for (const auto &u : users) {
		user_ids.insert(make_pair(u.id, 0));
	}

	unsigned uid = 0;
	for (auto &u : user_ids) {
		u.second = uid++;
	}

	return user_ids;
}

map<unsigned, unsigned> make_item_ids(const set<user> &users) {

	map<unsigned, unsigned> item_ids;

	for (const auto &u : users) {
		for (const auto & r : u.list) {
			item_ids.insert(make_pair(r.id, 0));
		}
	}

	unsigned iid = 0;
```

```
        for (auto &i : item_ids) {
                i.second = iid++;
        }

        return item_ids;
}

#endif // ID_HANDLERS_H
```

5.3.2.8JSON_TO_MAP.H

Σε αυτό το Project χρησιμοποιήθηκε η βιβλιοθήκη json του nlohmann. Μας επιτρέπει να διαβάζουμε και να εξάγουμε δεδομένα σε μορφή json. Json σημαίνει JavaScript Object Notation και είναι ο τρόπος που χρησιμοποιεί η JavaScript για να διαχειριστεί δεδομένα και κλάσεις.

json_to_map

Χρησιμοποιείται για να μετατρέψουμε ένα json σε μια δομή που μπορούμε να χρησιμοποιήσουμε ευκολότερα στη c++. Σε αυτήν την περίπτωση ένα MAP.

```cpp
#ifndef JSON_TO_MAP_H
#define JSON_TO_MAP_H

#include "Utilities/json.h"
#include <map>

using namespace std;

template<typename KEY_T, typename VALUE_T>
map<KEY_T, VALUE_T> json_to_map(
        const string &json_string,
        const string &key_field,
        const string &value_field)
{
    map<KEY_T, VALUE_T> returning;

    KEY_T key;
    VALUE_T value;

    for (const auto &r : nlohmann::json::parse(json_string)) {
        key = static_cast<KEY_T>(r[key_field]);
        value = static_cast<VALUE_T>(r[value_field]);

        returning.insert({key, value});
    }

    return returning;
}

#endif // JSON_TO_MAP_H
```

5.3.2.9 REVERSE_MAP.H

reverse_map

Αντιστρέφει τα κλειδιά και τις τιμές του Map που της περνάμε.

```
#ifndef REVERSE_MAP_H
#define REVERSE_MAP_H

#include <map>

using namespace std;

map<unsigned, unsigned> reverse_map(const map<unsigned, unsigned> &m) {

	map<unsigned, unsigned> r;

	for (auto x : m) {
		r.insert({x.second, x.first});
	}
	return r;
}

#endif // REVERSE_MAP_H
```

5.3.2.10 LOAD_FROM_DATABASE.H

Εδώ φορτώνονται όλα τα στοιχεία από τη βάση δεδομένων PostgreSQL.

get_users_from_db

Κάνει αίτηση στη βάση για τον πίνακα των ratings και τα περνάει στην αναλυτική δομή.

get_movies_from_db

Κάνει αίτηση στη βάση για τον πίνακα των ταινιών και τα περνάει σε μια λίστα με ταινίες.

```
#ifndef LOAD_FROM_DATABASE_H
#define LOAD_FROM_DATABASE_H

#include <pqxx/pqxx>
#include "classes.h"

#include <iomanip>

using namespace pqxx;

set<user> get_users_from_db() {

	set<user> users;

	cout << "Running ratings query... " << flush;
	work w(*db.get_conn());
	auto result = w.exec("select uid, mid, value from ratings");
	cout << "Done" << endl;

	unsigned total = result.size();
	unsigned current = 0;
	cout << "Loading to RAM..." << endl;
	cout << fixed << setprecision(2);
	for (const auto r : result) {
		users.insert(
			user(r.at("uid").as<unsigned>())
		)
		.first->list.insert(
			{
				r.at("mid").as<unsigned>(),
				r.at("value").as<double>()
			}
		);
		if (++current % 100000 == 0) {
```

```
                cout << current / (double) total * 100 << "\r" << flush;
            }
        }
        cout << "Done          " << "\r" << endl;

        return users;
}

set<movie> get_movies_from_db() {

        work w(*db.get_conn());
        set<movie> movies;

        cout << "Running movies query..." << flush;
        auto result = w.exec("select id, name, imdb, tmdb from movies");
        cout << "Done" << endl;

        cout << "Loading to RAM..." << endl;
        for (const auto row : result) {
            movies.insert(movie(
                row.at("id").as<unsigned>(),
                row.at("name").as<string>(),
                row.at("imdb").as<string>(),
                row.at("tmdb").as<string>()
            ));
        }

        return movies;
}

#endif // LOAD_FROM_DATABASE_H
```

5.3.2.11 ENVIRONMENT.H

Η κλάση **ENVIRONMENT** είναι αυτή που κρατάει όλα τα δεδομένα για την τρέχουσα αίτηση του συνδεδεμένου χρήστη. Στο map<string, string> get υπάρχουν τα στοιχεία που πέρασε ο χρήστης μέσω της μεθόδου **HTTP: GET**.

Η αίτηση:

url/**?my_ratings**={“id”:300, “value”: 5}**&my_settings**={“pearson”:67}

Μετατρέπεται σε:

my_ratings

ID	300
Value	5

my_settings

Pearson	67

Οι υπόλοιπες συναρτήσεις της κλάσης βοηθάνε στο να γεμίσουμε αυτήν τη δομή.

fill_get

Γεμίζει τα περιεχόμενα του Map get. Αρχικά ελέγχει αν υπάρχει στο url το σύμβολο **?**. Αν δεν υπάρχει, τότε δεν υπάρχουν δεδομένα GET.

Αν υπάρχει, κρατάμε το περιεχόμενο μετά το σύμβολο **?** σε ένα String και το σπάμε σε κάθε ζευγάρι key-value που διαχωρίζονται από το σύμβολο **&**.

Ύστερα κάνει split κάθε ζευγάρι key-value στο σύμβολο **=** που διαχωρίσει το key από το value.

Τα 2 αυτά κομμάτια περνάνε στο map get.

```
#ifndef ENVIRONMENT_H
#define ENVIRONMENT_H

#include <pqxx/pqxx>
#include "WebServer/utilities.h"

using namespace pqxx;
using namespace std;

struct ENVIRONMENT {

    mutable map<string, string> get;
    string full_request;
    string uri;

    ENVIRONMENT() {}

    ENVIRONMENT(const FCGX_Request &request) {

        full_request = FCGX_GetParam(
            "REQUEST_URI",
            request.envp
        );

        size_t q_pos = full_request.find("?");

        uri = full_request.substr(0, q_pos);
        // + 1 to remove '?'
        if (q_pos != string::npos) {
            fill_get(full_request.substr(q_pos + 1));
        }
    }

    void fill_get(const string &get_request) {

        string key, value;
```

```
        vector<string> pairs = split(get_request, '&');

        for (string &x : pairs) {

            string p = unescape_post_data(x);

            size_t eq_pos = p.find("=");

            // asdasd/?debug
            if (eq_pos == string::npos) {
                get.insert({p, ""});
                continue;
            }

            key = p.substr(0 ,eq_pos);
            // + 1 to remove '='
            value = p.substr(eq_pos + 1);

            // asdasd/?=wrong
            if (key.size() == 0) {
                continue;
            }

            // asdasd/?asd=
            if (value.size() == 0) {
                get.insert({key, ""});
                continue;
            }

            // asdasd/?saladas=true
            get.insert({key, value});
        }
}

bool contains(const string &path) const {
```

```
        return full_request.find(path) != string::npos;
    }

    vector<string> split(const string &s, const char &delimiter) {

        vector<string> res;
        stringstream ss(s);

        string item;
        while (std::getline(ss, item, delimiter)) {
            res.push_back(item);
        }
        return std::move(res);
    }
};

#endif // ENVIROMENT_H
```

5.3.2.12 HANDLE_REQUEST.H

handle_request

Καλείται σε κάθε αίτημα και χρησιμοποιείται ως διακομιστής μεσολάβησης για άλλες συναρτήσεις.

Δηλαδή, αν η αίτηση περιέχει τη λέξη calculate, τότε καλείται η συνάρτηση calculate και επιστρέφεται το αποτέλεσμα που φέρνει.

Το ίδιο συμβαίνει και για τις συναρτήσεις show_selected και search.

```
#ifndef HANDLE_REQUEST_H
#define HANDLE_REQUEST_H

#include "classes.h"
#include "globals.h"
#include "WebServer/Actions/calculate.h"
#include "WebServer/Actions/search.h"
#include "WebServer/Actions/show_selected.h"

string handle_request() {

	cout << "UnEncoded:"
		<< unescape_post_data(e.full_request)
		<< endl;

	if (e.contains("calculate")) {
		return calculate();
	}

	if (e.contains("show_selected")) {
		return show_selected();
	}

	if (e.contains("search")) {
		return search();
	}

	return "";
}

#endif // HANDLE_REQUEST_H
```

5.3.2.13 TEMPLATE.H

Η κλάση Template δέχεται μια θέση ενός αρχείου στον δίσκο και το φορτώνει στην μνήμη. Αυτό το αρχείο περιέχει λέξεις και φράσεις που περιφράσσονται με τα σύμβολα **{{** και **}}** τις οποίες ονομάζουμε tags. Βρίσκει αυτά τα tags στο αρχείο και δημιουργεί μια δομή δεδομένων που μπορεί να αντιστοιχήσει μια συμβολοσειρά σε κάθε tag.

Όταν έχουμε αντιστοιχήσει όλα τα tags που χρειαζόμαστε, καλούμε την συνάρτηση **render**(). Έτσι όλα τα tag που θέσαμε, αντικαθίστανται με την συμβολοσειρά που τους ανατέθηκε στην δομή δεδομένων.

Παράδειγμα:

Περιεχόμενο αρχείου:

Γειά σου, {{name}}, θες να βρεθούμε {{day}} στις {{time}};

Γεμίζουμε τα tags:

tag["name"] = Νίκο

tag["day"] = το Σάββατο 2017/11/4

tag["time"] = 5:30

Καλούμε την **render**():

Γειά σου, Νίκο, θες να βρεθούμε το Σάββατο 2017/11/4 στις 5:30;

```cpp
#pragma once
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <cassert>
#include <map>
using namespace std;
namespace ATC {
enum class TType {RAW, FILE};

class Template {

   string html_content;
   map<string, stringstream> rep;

public:

   Template(const string &str, TType type = TType::FILE) {

      if (type == TType::RAW) {
         html_content = str;
         return;
      }

      ifstream template_file(str);
      assert(template_file.is_open());
      stringstream buffer;
      buffer << template_file.rdbuf();
      html_content = buffer.str();
   }

   stringstream & operator[](const string &key) {
      return rep[key];
```

```
    }

    void clear() { rep.clear(); }


    string render() {

        string final = html_content;

        for (const auto &p : rep) {

            size_t pos = final.find("{{"+p.first+"}}");
            while (pos != string::npos) {
                // keep the pos so we can skip the allready scanned file
                final.replace(pos, p.first.length()+4, p.second.str());
                pos = final.find("{{"+p.first+"}}", pos);
            }
        }
        return final;
    }
};
}
```

5.3.2.14 UTILITIES.H

Η αίτηση του χρήστη φτάνει στον server ως μια URL-Encoded συμβολοσειρά. Αυτό σημαίνει ότι τα σύμβολα που δεν είναι ασφαλή να μεταφέρονται ως κείμενο (π.χ. {, }, “), έχουν αντικατασταθεί με το σύμβολο **%** ακολουθούμενο με τον **δεκαεξαδικό κωδικό ASCII** του αρχικού συμβόλου.

Χαρακτήρας	**ASCII**	**URL κωδικός**
space	32	%20
!	33	%21
“	34	%22
#	35	%23
$	36	%24
%	37	%25
&	38	%26
…	…	…

unescape_post_data

Δέχεται την url-encoded συμβολοσειρά και επιστρέφει ένα String με τα κανονικά σύμβολα.

```
#ifndef UTILITIES_H
#define UTILITIES_H

#include "classes.h"

string unescape_post_data(string &data) {

   string decoded = data;
   size_t pos;
   while ((pos = decoded.find("+")) != string::npos) {
      decoded.replace(pos, 1, " ");
   }

   // From stackoverflow:
   while ((pos = decoded.find("%")) != string::npos) {
      if (pos <= decoded.length() - 3) {
         char replace[2] = {
            (char)(stoi("0x" + decoded.substr(pos+1,2), NULL, 16)),
            '\0'
         };
         decoded.replace(pos, 3, replace);
      }
   }

   return decoded;
}

#endif // UTILITIES_H
```

5.3.2.15 CALCULATE.H

calculate

Καλείται όταν ο χρήστης πατάει submit. Η λειτουργία της είναι να ετοιμάσει τα στοιχεία που έστειλε ο χρήστης ώστε να μπορούν να χρησιμοποιηθούν από τη συνάρτηση calculator(). Επίσης είναι υπεύθυνη να δημιουργήσει την απάντηση που θα λάβει ο χρήστης.

Αρχικά παίρνει τα rating του χρήστη και τα μετατρέπει στη συμπυκνωμένη δομή. Υπολογίζει το μέσο όρο του χρήστη και περνάει αυτά τα δεδομένα στην calculator().

Η calculator θα επιστρέψει έναν πίνακα με τις προτεινόμενες ταινίες. Χρησιμοποιώντας την κλάση template για κάθε προτεινόμενη ταινία, γεμίζει μια template-σειράς και την προσθέτει στο template-απαντήσεις.

```
#ifndef CALCULATE_H
#define CALCULATE_H

#include "classes.h"
#include "WebServer/environment.h"
#include "WebServer/template.h"
#include "Utilities/json_to_map.h"
#include "Utilities/avg.h"
#include "globals.h"
#include "WebServer/Actions/set_user_settings.h"
#include "Calculator/calculator.h"

string calculate() {

    ATC::Template t("front_end/html_parts/show_results.html");

    ATC::Template line("front_end/html_parts/show_results_row.html");

    map<unsigned, double> rating_map =
        json_to_map<unsigned, double>(
            e.get["my_ratings"],
            "id",
            "value"
        );
    set<rating> user_ratings;
    for (const auto &r : rating_map) {
        user_ratings.insert({r.first, r.second});
    }

    double user_avg = find_avg_from_set(user_ratings);
    cout << "user's avg: " << user_avg << endl;

    set_user_settings();

    // Print user's ratings
```

```
    for (auto &r : user_ratings) {
        cout << "DB.id:" r.id  << "\t" << r.value
            << "\t" << movies.find({r.id, "", "", ""})->name << endl;
    }

    vector<unsigned> x;
    for (const auto &r : user_ratings) {
        x.push_back(compress(item_ids[r.id], r.value));
    }

    vector<returning> results =
        calculator(items_per_user, avg, x, user_avg);

    t["content"] << "";
    for (auto m : results) {

        unsigned db_id = item_ids_reversed[m.id];

        line.clear();

        line["id"] << db_id;
        line["name"] << movies.find({db_id, "", "", ""})->name;
        line["value"] << round(m.value);
        line["common"] << m.common_users;
        line["imdb_link"] << movies.find( db_id, "", "", ""}  )->imdb;

        t["content"] << line.render();
    }
    cout<< "Template filled" << endl;
    return t.render();
}

#endif // CALCULATE_H
```

5.3.2.16 SEARCH.H

search

Καλείται όταν ο χρήστης κάνει αναζήτηση. Παίρνει το κείμενο που αναζήτησε ο χρήστης (token) και το ψάχνει μέσα στα ονόματα όλων των ταινιών της βάσης. Επιστρέφει τις ταινίες που περιέχουν το token στον τίτλο τους.

```
#ifndef SEARCH_H
#define SEARCH_H

#include <classes.h>
#include "WebServer/environment.h"
#include "WebServer/template.h"

string search() {

    ATC::Template t("front_end/html_parts/search.html");

    ATC::Template line("front_end/html_parts/search_row.html");

    work w(*db.get_conn());
    result res = w.prepared("search")(e.get["token"]).exec();
    cout << "Found " << res.size() << " matches.\n";

    t["content"] << "";

    for (const auto &r : res) {

        line.clear();

        line["id"] << r.at("id").as<unsigned>();
        line["name"] << r.at("name").as<string>();
        line["imdb_link"] << r.at("imdb").as<string>();

        t["content"] << line.render();
    }

    return t.render();
}

#endif // SEARCH_H
```

5.3.2.17 SET_USER_SETTINGS.H

set_user_settings

Εκτελείται κάθε φορά που ο χρήστης πατάει submit. Χρησιμοποιείται για να διαβάσει τις ειδικές ρυθμίσεις που έδωσε ο χρήστης από το παράθυρο ρυθμίσεων και να τις σώσει στις ανάλογες μεταβλητές, ώστε να συμπεριληφθούν στον υπολογισμό γειτόνων και προτεινόμενων βαθμολογιών.

```
#ifndef SET_USER_SETTINGS_H
#define SET_USER_SETTINGS_H

#include "classes.h"

void set_user_settings() {

	bool p, c, r;
	p = c = r = false;

	// Set user's settings
	auto map_settings = json_to_map<unsigned, double>(
		e.get["my_settings"],
		"id",
		"value"
	);
	for (const auto &s : map_settings) {

		if (s.first == 1) {
			MIN_PEARSON =
```

```
                    (s.second < 0.0 || s.second > 1.0)
                    ? 0.0 : s.second;
            p = true;
        }
        if (s.first == 2) {
            MIN_COMMON_MOVIES =
                (s.second < 1) ? 1 : s.second;
            c = true;
        }
        if (s.first == 3) {
            if (s.second < MIN_RATING) {
                MIN_RECOMMENDED = MIN_RATING;
            } else if (s.second > MAX_RATING) {
                MIN_RECOMMENDED = MAX_RATING;
            } else {
                MIN_RECOMMENDED = s.second;
            }
            r = true;
        }
    }

    // Fallback
    if (!p) { MIN_PEARSON = 0.2; }
    if (!c) { MIN_COMMON_MOVIES = 5; }
    if (!r) { MIN_RECOMMENDED = 5; }

    cout << "MIN_PEARSON:      " << MIN_PEARSON << endl;
    cout << "MIN_COMMON_MOVIES: " << MIN_COMMON_MOVIES << endl;
    cout << "MIN_RECOMMENDED:   " << MIN_RECOMMENDED << endl;
}

#endif // SET_USER_SETTINGS_H
```

5.3.2.18 SHOW_SELECTED.H

Show_selected

Εκτελείται όταν ο χρήστης επιλέξει να δει όλες τις ταινίες που έχει βαθμολογήσει.

Χρησιμοποιώντας 2 template, σελίδας και γραμμής, δημιουργεί έναν πίνακα με την βαθμολογία και το όνομα των ταινιών που βαθμολόγησε ο χρήστης.

```
#ifndef SHOW_SELECTED_H

#define SHOW_SELECTED_H

#include <classes.h>
#include "WebServer/environment.h"
#include "WebServer/template.h"

string show_selected() {

    ATC::Template t("front_end/html_parts/show_selected.html");

    ATC::Template line(
        "front_end/html_parts/show_selected_row.html"
    );

    map<unsigned, double> user_ratings =
```

```
                json_to_map<unsigned, double> (
                        e.get["my_ratings"], "id", "value"
                );

        multimap<double, unsigned> ordered_user_ratings;
        for (const auto &r : user_ratings) {
                ordered_user_ratings.insert({r.second, r.first});
        }

        t["content"] << "";

        for (const auto &r : ordered_user_ratings) {

                line.clear();

                line["id"] << r.second;
                line["name"] << movies.find({r.second, "", "", ""})
                        ->name;
                line["value"] << r.first;
                line["imdb_link"] << movies.find({r.second, "", "", ""})
                        ->imdb;

                t["content"] << line.render();
        }

        return t.render();

}

#endif // SHOW_SELECTED_H
```

5.3.2.19 CALCULATOR.H

calculator

Κάνει τους υπολογισμούς για να βρεθούν οι προτεινόμενες ταινίες για τον συνδεδεμένο χρήστη.

Αρχικά βρίσκει όλους τους γείτονες του χρήστη χρησιμοποιώντας 8 υποδιεργασίες οι οποίες θα αναλυθούν παρακάτω.

Ύστερα μαζεύει όλες τις ταινίες που έχουν δει όλοι οι γείτονες, αφαιρεί αυτές που έχει δει ο χρήστης, τις ταξινομεί ανάλογα με το πόσοι γείτονες έχουν δει την κάθε ταινία και κρατάει τις 1000 πρώτες με τα περισσότερα views.

Τέλος υπολογίζει την προτεινόμενη βαθμολογία για κάθε ταινία από τις 1000 και επιστρέφει έναν πίνακα της κλάσης returning όπου περιέχει όλα τα στοιχεία που θα παρουσιαστούν στον χρήστη.

```
#ifndef CALCULATOR_H
#define CALCULATOR_H

#include "classes.h"
#include "Calculator/multithread.h"
#include "Calculator/pearson.h"
#include "Utilities/compressor.h"

vector<returning> calculator(
        const vector<vector<unsigned>> &items_per_user,
        const vector<double> &avg,
        const vector<unsigned> &x,
        const double &x_avg)
{
        map<unsigned, double> neighbors =
                        multi_thread_me(
                                items_per_user, x, avg, x_avg, 8);

        cout << "Neighbors: " << neighbors.size() << endl;

        // Count common movies
        map<unsigned, size_t> common_movies;
        for (const auto &n : neighbors) {
                for (const unsigned &i : items_per_user[n.first]) {
                        common_movies.insert(
                                {get_id(i), 0}
                        ).first->second++;
                }
        }

        cout << "Common movies: " << common_movies.size() << endl;

        // Remove viewed
        for (const auto &r : x) {
```

```
        auto it = common_movies.find(get_id(r));
        if (it != common_movies.end()) {
            common_movies.erase(it);
        }
}
cout << "Removed seen: " << common_movies.size() << endl;

vector<pair<unsigned, unsigned>> common;
common.reserve(common_movies.size());
for (auto &c : common_movies) {
        common.push_back({c.first, c.second});
}

// Sort descending views
sort(
        begin(common),
        end(common),
        [](
            const pair<unsigned, unsigned> &lhs,
            const pair<unsigned, unsigned> &rhs)
        {
            return lhs.second > rhs.second;
        }
);

if (common.size() > MAX_RESPONSE) {
        common.resize(MAX_RESPONSE);
}
cout << "Keeping  " << common.size() << " movies" << endl;

// todo multi thread
double score, weight, recommend, value;
vector<returning> results;
for (const auto &m : common) {
```

```
            score = weight = recommend = 0;
            for (auto y : neighbors) {

                // Has this user seen this movie?
                value = -10;
                for (auto r : items_per_user[y.first]) {

                    // Because it's sorted
                    if (m.first < get_id(r)) { break; }

                    if (get_id(r) == m.first) {
                        value = get_value(r);
                        break;
                    }
                }
                if (value < -1) { continue; }

                score  += y.second * (value - avg[y.first]);
                weight += y.second;
            }

            recommend = x_avg  + (score / weight);

            if (recommend <= MIN_RECOMMENDED) { continue; }

            results.push_back({m.first, recommend, m.second});
        }

        cout << "Done! Returning " << results.size() << endl;

        return results;
}

#endif // CALCULATOR_H
```

5.3.2.20 GET_NEIGHBORS.H

get_neighbors

Χρησιμοποιείται σε συνδυασμό με την multithread_me για να βρει τους γείτονες του συνδεδεμένου χρήστη. Δέχεται τον πίνακα των χρηστών της βάσης μαζί με ορίσματα για τα όρια του πεδίου που θα πρέπει να καλύψει αυτό το thread.

```
#ifndef GET_NEIGHBORS_H
#define GET_NEIGHBORS_H

#include "classes.h"
#include "globals.h"
#include "Calculator/pearson.h"
#include <future>

void get_neighbors(
        const unsigned &from,
        const unsigned &to,
        const vector<vector<unsigned>> &items_per_user,
        const vector<unsigned> &x,
        const vector<double> &avg,
        const double &x_avg,
        promise<map<unsigned, double>> promm)
{
        map<unsigned, double> neighbors;
        double pearson;

        for (size_t i = from; i < to; ++i) {

                pearson = pearson_calc(
                        items_per_user[i], x, avg, x_avg, i
);
                if (pearson <= MIN_PEARSON) { continue; }
                neighbors.insert({i, pearson});
        }

        promm.set_value(neighbors);
}

#endif // GET_NEIGHBORS_H
```

5.3.2.21 MULTITHREAD.H

multi_thread_me

Χρησιμοποιείται ως βοηθητική συνάρτηση κατά την εύρεση των γειτόνων του χρήστη. Επειδή αυτή είναι η πιο δύσκολη διαδικασία, αυτή η συνάρτηση βοηθάει στο να εκτελεστεί πιο γρήγορα. Πιο συγκεκριμένα, δημιουργεί 8 νέες διεργασίες οι οποίες δέχονται το 1/8 των χρηστών της βάσης και τα συγκρίνουν με το συνδεδεμένο χρήστη. Τα vector που επιστρέφει κάθε διεργασία συνδέεται με τα vector των άλλων διεργασιών ώστε η συνάρτηση να επιστρέψει όλους τους γείτονες του συνδεδεμένου χρήστη.

```
#ifndef MULTITHREAD_H
#define MULTITHREAD_H

#include "classes.h"
#include "Calculator/get_neighbors.h"
#include <memory>
#include <future>

map<unsigned, double> multi_thread_me(
        const vector<vector<unsigned>> &items_per_user,
        const vector<unsigned> &x,
        const vector<double> &avg,
        const double &x_avg,
        const unsigned &ithreads)
{

    vector<unsigned> seperators;
    for (unsigned i = 0; i < ithreads; ++i) {
        seperators.push_back((items_per_user.size() / ithreads) * i);
    }
    seperators.push_back(items_per_user.size());

    vector<promise<map<unsigned, double>>> prom(ithreads);
    vector<future<map<unsigned, double>>>  futu(ithreads);
    vector<thread> threads(ithreads);

    for (unsigned i = 0; i < ithreads; ++i) {
        futu[i] = prom[i].get_future();
    }
    for (unsigned i = 0; i < ithreads; ++i) {
        //cout << "Creating thread" << endl;
        threads[i] = thread(
            get_neighbors
            , seperators[i]
```

```
                , seperators[i + 1]
                , items_per_user
                , x
                , avg
                , x_avg
                , move(prom[i])
            );
        }
        // --> wait here <--
        map<unsigned, double> results;
        for (unsigned i = 0; i < ithreads; ++i) { threads[i].join(); }
        for (unsigned i = 0; i < ithreads; ++i) {
            map<unsigned, double> returned = futu[i].get();
            results.insert(returned.begin(), returned.end());
        }

        return results;
}

#endif // MULTITHREAD_H
```

5.3.2.22 PEARSON.H

pearson_calc

Η pearson_calc υλοποιεί τον αλγόριθμο του pearson. Δέχεται 2 πίνακες με τα συμπιεσμένα rating του κάθε χρήστη μαζί με το μέσο όρο του κάθε χρήστη και επιστρέφει το μέτρο ομοιότητας pearson από -1 έως 1.

```
#ifndef PEARSON_H

#define PEARSON_H

#include "classes.h"
#include "Utilities/compressor.h"

double pearson_calc(
            const vector<unsigned> &x,
            const vector<unsigned> &y,
            const vector<double> &avg,
            const double &x_avg,
            const unsigned &y_id)
{
      size_t x_r = 0, y_r = 0, common = 0;
      double calc_x, calc_y, up = 0, down_l = 0, down_r = 0;
      x_r = y_r = common = up = down_l = down_r = 0;

      while (x_r < x.size() && y_r < y.size()) {

            if (get_id(x[x_r]) < get_id(y[y_r])) { x_r++; continue; }
```

```
            if (get_id(x[x_r]) > get_id(y[y_r])) { y_r++; continue; }
            ++common;

            calc_x = get_value(x[x_r]) - x_avg;
            calc_y = get_value(y[y_r]) - avg[y_id];

            up     += calc_x * calc_y;
            down_l += calc_x * calc_x;
            down_r += calc_y * calc_y;

            ++x_r;
        }

        if (
            common <= MIN_COMMON_MOVIES
            || up == 0
            || down_l == 0
            || down_r == 0
        ) {
            return -10;
        } else {
            return up / (sqrt(down_l * down_r));
        }
}

#endif // PEARSON_H
```

5.4 ΕΝΗΜΕΡΩΣΕΙΣ ΔΙΑΧΕΙΡΙΣΤΗ

5.4.1 ΕΚΚΙΝΗΣΗ

```
Running ratings query... Done

Loading to RAM...

Done

Running movies query...Done

Loading to RAM...

Data Loaded!
```

5.4.2 ΑΝΑΖΗΤΗΣΗ

```
--==[ Recieved request ]==--

UnEncoded:/search?token=titanic

Found 7 matches.

Response is ready!

Sending Response...
```

5.4.3 ΥΠΟΛΟΓΙΣΜΟΣ

```
--==[ Recieved request ]==--
Decoded:/calculate?my_ratings=[{"id":118916,"value":10},{"id":1721,"value":1
0},{"id":74754,"value":1},{"id":72998,"value":10},{"id":53996,"value":9},{"id":6
9526,"value":4},{"id":87520,"value":3},{"id":112370,"value":1},{"id":2571,"valu
e":10},{"id":6365,"value":6},{"id":6934,"value":4}]&my_settings=[]
user's avg: 6.18
MIN_PEARSON:      0.20
MIN_COMMON_MOVIES: 5
MIN_RECOMMENDED:   5
DB.id:1721        10.00     Titanic (1997)
DB.id:2571        10.00     The Matrix (1999)
DB.id:6365        6.00      The Matrix Reloaded (2003)
DB.id:6934        4.00      The Matrix Revolutions (2003)
DB.id:53996       9.00      Transformers (2007)
DB.id:69526       4.00      Transformers: Revenge of the Fallen (2009)
DB.id:72998       10.00     Avatar (2009)
DB.id:74754       1.00      The Room (2003)
DB.id:87520       3.00      Transformers: Dark of the Moon (2011)
DB.id:112370      1.00      Transformers: Age of Extinction (2014)
DB.id:118916      10.00     Titanic (1996)
Neighbors: 1307
Common movies: 21554
Removed seen: 21543
Keeping  1000 movies
Done! Returning 930
Template filled
Response is ready!
Sending Response...
```

6 ΣΥΜΠΕΡΑΣΜΑΤΑ

Σε αυτή την πτυχιακή αναλύθηκαν τα βήματα σχεδίασης και ανάπτυξης ενός συστήματος εξατομίκευσης. Οι επιλογές των λογισμικών έγιναν με στόχο την σταθερότητα του συστήματος.

Κατά τον σχεδιασμό του συστήματος σημαντικό ρόλο έπαιξε η αναγνώριση των λειτουργιών που θα έπρεπε να παρέχει. Όταν συγκεντρώθηκε η λίστα των προδιαγραφών, η σχεδίαση της διεπαφής χρήστη συγκροτήθηκε κυρίως από την τοποθέτηση αυτών των στοιχείων στην Desktop και Mobile προβολή.

Η διαδικασία υλοποίησης του τοπικού συστήματος επισήμανε την σημαντικότητα της σωστής οργάνωσης και του διαχωρισμού καθηκόντων ανάμεσα στα αρχεία του πηγαίου κώδικα της εφαρμογής. Επίσης, παρατηρήθηκε ότι η βιβλιοθήκη libfcgi++ δεν παρέχει ξεκάθαρο τρόπο εξαγωγής των δεδομένων POST από ένα HTTP αίτημα.

Πιθανές μελλοντικές υλοποιήσεις ενός παρόμοιου συστήματος θα πρέπει να λάβουν υπόψη τον διαχωρισμό των λειτουργιών του (όπως η σύνδεση στην βάση, εξαγωγή πληροφοριών από αιτήματα και υπολογισμός προτεινόμενων αντικειμένων).

Περισσότερη έρευνα χρειάζεται στον τρόπο υπολογισμού των προτεινομένων αντικειμένων χρησιμοποιώντας συνεργατικό φιλτράρισμα. Με τους τρέχοντες αλγόριθμους, για την σωστή λειτουργεία του συστήματος, ο χρήστης πρέπει να βαθμολογήσει και αντικείμενα που δεν του αρέσουν, δράση που δεν είναι πάντα εφικτή.

Καθημερινά δημιουργούνται και δημοσιεύονται διάφορες υλοποιήσεις συστημάτων εξατομίκευσης ανοιχτού κώδικα. Στο μέλλον όλο και περισσότερες πλατφόρμες θα χρησιμοποιούνε συστήματα εξατομίκευσης, διευκολύνοντας την εύρεση πληροφοριών στο διαδίκτυο.

7 ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Federal Standard 1037C data stream
2. Hanani, U., Shapira, B. & Shoval, P. User Modeling and User-Adapted Interaction (2001) 11: 203.
3. Francesco Ricci and Lior Rokach and Bracha Shapira, Introduction to Recommender Systems Handbook, Recommender Systems Handbook, Springer, 2011
4. Terveen, Loren; Hill, Will (2001). "Beyond Recommender Systems: Helping People Help Each Other" (PDF). Addison-Wesley. p. 6. Retrieved 16 January 2012
5. Personalization Techniques and Their Application - Juergen Anke (Dresden University of Technology, Germany) and David Sundaram (University of Auckland, New Zealand)
6. Gross, Bertram M. (1964). The Managing of Organizations: The Administrative Struggle, vol 2
7. Xiaoyuan Su and Taghi M. Khoshgoftaar, “A Survey of Collaborative Filtering Techniques,” Advances in Artificial Intelligence, vol. 2009, Article ID 421425, 19 pages, 2009. doi:10.1155/2009/421425A
8. Comparative Analysis of Offline and Online Evaluations and Discussion of Research Paper Recommender System Evaluation