



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών « ΠΜΣ ΠΛΗΡΟΦΙΚΗΣ »

Μεταπτυχιακή Διατριβή

| | |
|-----------------------|---|
| Τίτλος Διατριβής | Ανάπτυξη εφαρμογής για την ηλεκτρονική διαχείριση πολυχώρου κινηματογραφικών αιθουσών Cinema Complex Application development for managing multiplex cinema halls at Cinema Complex |
| Ονοματεπώνυμο Φοιτητή | Μαρία Κατρινάκη |
| Πατρώνυμο | Δημήτριος |
| Αριθμός Μητρώου | ΜΠΠΛ18033 |
| Επιβλέπων | Ευθύμιος Αλέπης, Αναπληρωτής Καθηγητής |

Ημερομηνία Παράδοσης

Δεκέμβριος 2023

Τριμελής Εξεταστική Επιτροπή

Ευθύμιος Αλέπης
Αναπληρωτής Καθηγητής

Μαρία Βίρβου
Καθηγήτρια

Σακκόπουλος Ευάγγελος
Αναπληρωτής Καθηγητής

Περιεχόμενα

| | |
|--|----|
| Περίληψη..... | 3 |
| Abstract | 4 |
| 1.Εισαγωγή..... | 5 |
| 2.Είσοδος στην εφαρμογή Cinema Complex | 5 |
| 3. Μενού Ταινιών | 6 |
| 3.1 Η Ταινία με βάση τον Τίτλο | 6 |
| 3.2 Οι Ταινίες με βάση την Κατηγορία | 7 |
| 3.3 Πληροφορίες Ταινίας | 8 |
| 4. Μενού Κράτηση Θέσεων | 9 |
| 4.1 Μενού Ταινιών του Χρήστη..... | 10 |
| 5. Μενού Προϊόντων | 11 |
| 5.1 Μενού Καλάθι Αγορών | 12 |
| 6. Μενού Παραγγελίας..... | 13 |
| 7. Μενού Επιβεβαίωσης Παραγγελιών από τον υπάλληλο | 14 |
| 7.1 Μενού Επιβεβαίωσης Εισιτηρίων από τον υπάλληλο | 15 |
| 8. Αρχιτεκτονική Συστήματος..... | 16 |
| 8.1 Spring Framework | 16 |
| 8.1.1 Spring Framework-Architecture | 17 |
| 8.1.2 Data Access/Integration..... | 18 |
| 8.1.3 WEB | 18 |
| 8.1.4 Aspect Oriented Programming | 18 |
| 8.1. 5 Messaging..... | 18 |
| 8.1.6 Test..... | 19 |
| 8.2 Spring Boot Framework | 19 |
| 8.2.1 Spring Boot Architecture..... | 20 |
| 8.2.2 Dependency Injection | 20 |
| 8.2.3 Spring Boot Annotation..... | 22 |
| 8.2.4 Hibernate | 23 |
| 8.2.5 JPA Cascade Type..... | 24 |
| 8.3 Maven..... | 25 |
| 8.4 Thymeleaf..... | 28 |
| 9. Αρχιτεκτονική της εφαρμογής..... | 30 |
| 9.1 Class @Entity | 32 |
| 9.2 Interface @Repository | 33 |
| 9.3 Class @Service..... | 33 |

| | |
|-----------------------------|----|
| 9.4 Class @Controller | 34 |
| 9.5 MySql Database..... | 34 |
| 10. Συμπεράσματα..... | 39 |
| 11. Βιβλιογραφία..... | 39 |

Περίληψη

Η παρούσα διπλωματική διατριβή έχει ως στόχο να παρέχει μια βελτιωμένη διαχείριση των διάφορων λειτουργιών ενός σύγχρονου κινηματογράφου. Η εφαρμογή αυτή προσφέρει μια πληθώρα λειτουργιών τόσο για τον απλό χρήστη που θέλει να πραγματοποιήσει μια κράτηση για ταινία όσο και για τον υπάλληλο του σινεμά που μπορεί να επιβεβαιώσει τις κρατήσεις του σινεμά.

Η διαδικασία σχεδιασμού της εφαρμογής πραγματοποιήθηκε μέσα από την ανάλυση των απαιτήσεων που θα χρειαστεί να υλοποιηθούν καθώς και μέσα από την χρήση των διάφορων τεχνολογιών.

Πιο συγκεκριμένα για το backend της εφαρμογής χρησιμοποιήθηκαν οι τεχνολογίες του Spring Boot όπου είναι ένα framework της java κατάλληλο για την ανάπτυξη web εφαρμογών. Η διαδικασία αποθήκευσης των δεδομένων υλοποιήθηκε στην σχεσιακή βάση δεδομένων MySQL .

Όσο αφορά αλληλεπίδραση με την βάση δεδομένων χρησιμοποιήθηκε η αρχιτεκτονική REST API καθώς και η πλατφόρμα για API Postman και η τεχνική ORM (Object-Relational-Mapping).

Τέλος, για το frontend της εφαρμογής χρησιμοποιήθηκαν οι τεχνολογίες Thymeleaf (μια σύγχρονη μηχανή προτύπων Java από την πλευρά του server που χρησιμοποιείται για web περιβάλλοντα) , JavaScript CSS ,HTML δημιουργώντας ένα εύχρηστο και φιλικό περιβάλλον στον χρήστη.

Abstract

This diploma thesis aims to provide an improved management of the various functions of a modern cinema. This application offers a wealth of features for both the casual user who wants to make a movie reservation and the cinema employee who can confirm cinema reservations.

The design process of the application was carried out through the analysis of the requirements that will need to be implemented as well as using various technologies.

More specifically, Spring Boot technologies were used for the backend of the application, which is a java framework suitable for the development of web applications. The process of storing the data was implemented with MySQL which is a relational database.

Regarding interaction with the database, the REST API architecture was used as well as the Postman API platform and the ORM (Object-Relational-Mapping) technique.

Finally, for the frontend development, Thymeleaf technologies were used (a modern server-side Java template engine used for web environments), JavaScript, CSS, HTML, creating an easy-to-use and user-friendly environment.

1.Εισαγωγή

Η ραγδαία ανάπτυξη της τεχνολογίας στην σύγχρονη εποχή έχει δημιουργήσει μια μεγάλη μερίδα πληθυσμού η οποία είναι αρκετά εξοικειωμένη με την χρήση της τεχνολογίας σε όλο το φάσμα της κοινωνικής ζωής. Έχει δημιουργηθεί λοιπόν η ανάγκη να αναπτυχθούν πλατφόρμες και λογισμικά τα οποία θα εξυπηρετούν πλέον τις ανάγκες ενός απλού καθημερινού ανθρώπου οι οποίες μπορεί να περιλαμβάνουν από την ανάγκη για διασκέδαση μέχρι μια εργασία σε κάποιο δημόσιο φορέα που μπορεί να πραγματοποιηθεί ηλεκτρονικά προσφέροντας του μεγαλύτερη ευκολία και αποφεύγοντας τις ώρες αναμονής σε κάποια ουρά με κόσμο.

Στο πλαίσιο αυτό λοιπόν η εφαρμογή Cinema Complex έχει δημιουργηθεί με στόχο να προσφέρει μια πληθώρα λειτουργιών για την καλύτερη εξυπηρέτηση τόσο του πελάτη του Cinema Complex όσο και να διευκολύνει τις εργασίες που εκτελεί ο υπάλληλος του σινεμά.

Η εφαρμογή αυτή προσφέρει την δυνατότητα για τον απλό χρήστη ο οποίος συνδέεται με την ιδιότητα του επισκέπτη να επιλέξει από το μενού να ενημερωθεί ποιες ταινίες παίζονται στο σινεμά, να δει πληροφορίες για τις ταινίες, να επιλέξει την ημέρα προβολής που επιθυμεί και να πραγματοποιήσει κράτηση εισιτηρίου το οποίο είτε το πληρώνει online την ώρα που πραγματοποιεί την κράτηση είτε αν δεν επιθυμεί πληρωμή αποθηκεύεται στην βάση σαν οφειλή και έχει την δυνατότητα να το πληρώσει όποτε εκείνος επιθυμεί. Επιπλέον μπορεί να ενημερωθεί για όλες τις παραγγελίες που έχει πραγματοποιήσει πληρωμένες και μη.

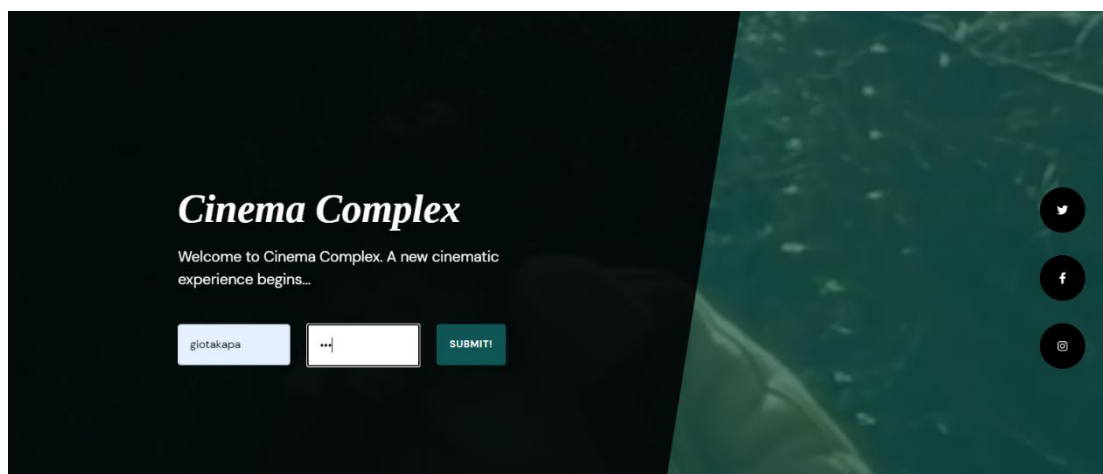
Μια άλλη δυνατότητα του επισκέπτη είναι ότι μπορεί να παραγγείλει προϊόντα από το μπαρ του σινεμά μέσω της εφαρμογής.

Από την άλλη μεριά ο χρήστης που συνδέεται ως υπάλληλος στην εφαρμογή έχει την δυνατότητα να ενημερωθεί για όλες τις παραγγελίες και κρατήσεις εισιτηρίων που έχουν πραγματοποιήσει οι πελάτες στην εφαρμογή καθώς και να τις επιβεβαιώσει αν έχουν εξοφληθεί αλλιώς να ειδοποιήσει το χρήστη ότι η παραγγελία του δεν έχει εξοφληθεί.

2.Είσοδος στην εφαρμογή Cinema Complex

Στην εφαρμογή μπορούν να συνδεθούν μόνο εγγεγραμμένοι χρήστες δηλαδή χρήστες οι οποίοι είναι αποθηκευμένοι στην βάση δεδομένων και οι οποίοι διακρίνονται σε επισκέπτες(visitor) και υπαλλήλους(employee). Στην βάση δεδομένων αυτή την στιγμή υπάρχουν τρεις χρήστες ένας με την ιδιότητα του υπαλλήλου με username mariakapa και δυο με την ιδιότητα του επισκέπτη : katerinakapa, giotakapa

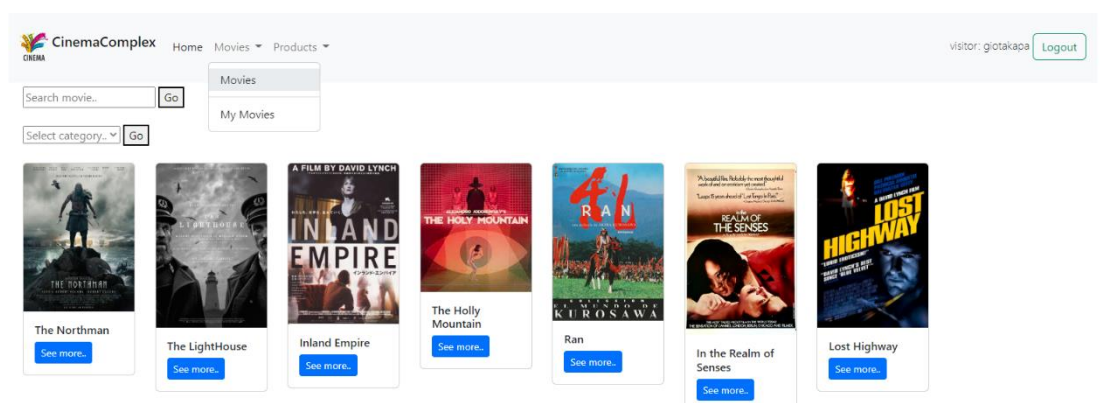
Η παρακάτω εικόνα παρουσιάζει την αρχική σελίδα της εφαρμογής όπου ο εγγεγραμμένος χρήστης μπορεί να εισέλθει πληκτρολογώντας το username και το password που βρίσκονται εγγεγραμμένα στην βάση δεδομένων.



Είσοδος χρήστη στην εφαρμογή

3. Μενού Ταινιών

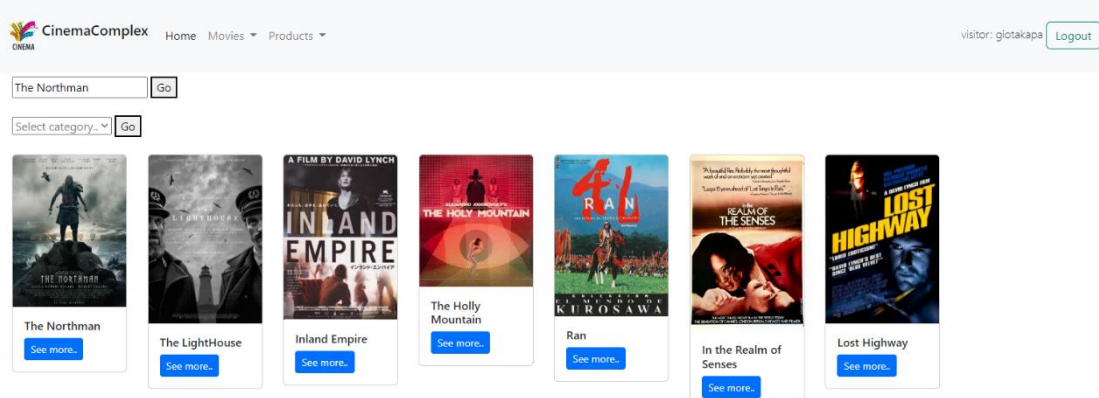
Ο χρήστης εφόσον εισέλθει στην εφαρμογή με την ιδιότητα του επισκέπτη όταν επιλέξει από το μενού πλοήγησης την επιλογή Movies του παρουσιάζονται οι ταινίες που παίζονται στο σινεμά και έχει την δυνατότητα να δει πληροφορίες για την ταινία και να πραγματοποιήσει κράτηση θέσης.



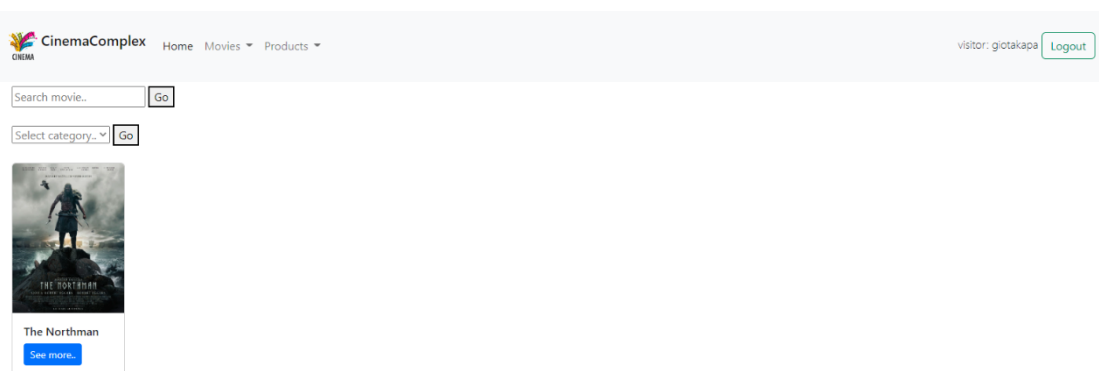
Μενού ταινιών

3.1 Η Ταινία με βάση τον Τίτλο

Στις παρακάτω εικόνες παρουσιάζονται δυο πεδία αναζήτησης στο πρώτο ο χρήστης με την ιδιότητα του επισκέπτη μπορεί να κάνει αναζήτηση με βάση τον τίτλο της ταινίας και στο δεύτερο πεδίο ο χρήστης μπορεί να κάνει αναζήτηση με βάση την κατηγορία στην οποία ανήκει η ταινία. Όπως παρατηρούμε ο χρήστης πραγματοποίησε αναζήτηση με βάση τον τίτλο της ταινίας The Northman συνεπώς εμφανίζεται μόνο η συγκεκριμένη ταινία ενώ οι υπόλοιπες αποκρύπτονται.



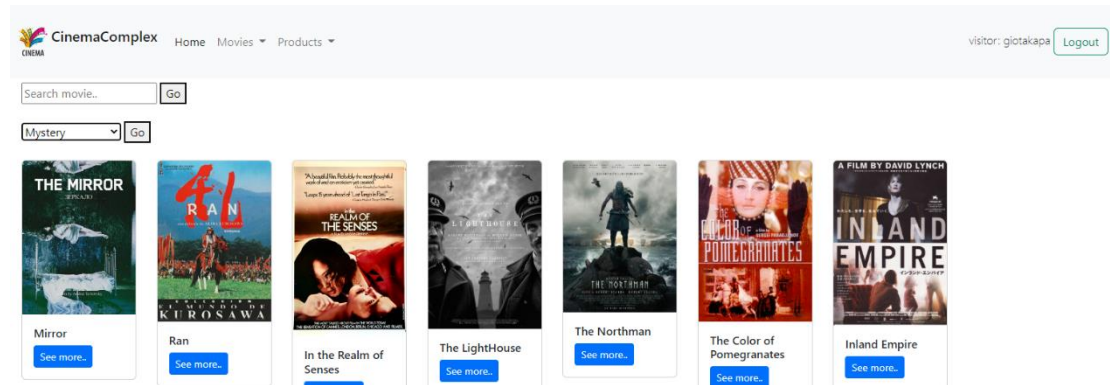
Αναζήτηση με βάση τον τίτλο της ταινίας



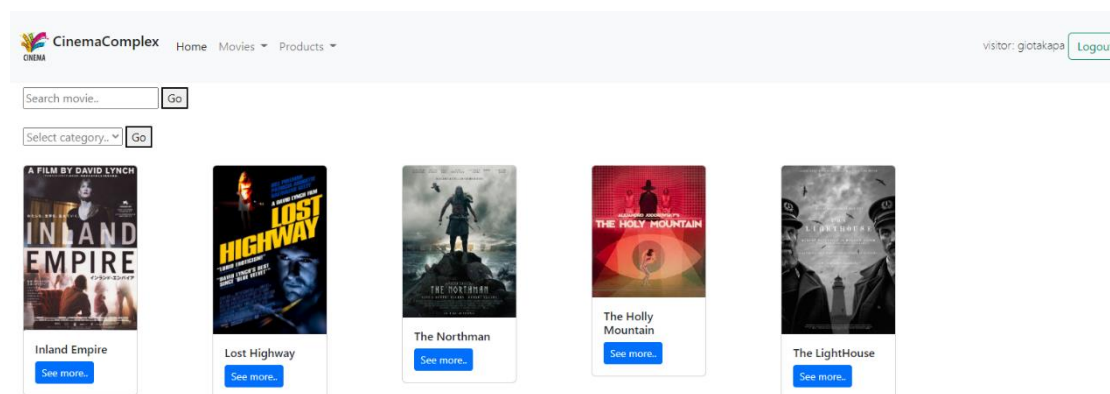
The Northman

3.2 Οι Ταινίες με βάση την Κατηγορία

Ο χρήστης με την ιδιότητα επισκέπτη έχει επίσης την δυνατότητα να ψάξει ταινίες με βάσεις τις κατηγορίες που υπάρχουν οι οποίες είναι :Mystery, Drama, Adventure. Όπως παρατηρούμε και στην παρακάτω εικόνα ο χρήστης έχει πραγματοποιήσει αναζήτηση με βάση την κατηγορία μυστηρίου όποτε εμφανίζονται μόνο οι ταινίες που ανήκουν στην συγκεκριμένη κατηγορία.



Αναζήτηση ταινίας με βάση την κατηγορία της



Ταινίες που ανήκουν στην κατηγορία μυστηρίου

3.3 Πληροφορίες Ταινίας

Στην παρακάτω εικόνα ο χρήστης επέλεξε να δει πληροφορίες για την ταινία Inland Empire και όπως βλέπουμε του εμφανίζονται όλες οι πληροφορίες που αφορούν την ταινία όπως : υπόθεση, σκηνοθετής, ηθοποιοί, βαθμολογία ταινίας, διάρκεια ταινίας, τιμή εισιτηρίου και ημέρα και ώρα προβολής που η ταινία προβάλλεται. Επιπλέον έχει επιλέξει ημέρα ώρα προβολής για να πραγματοποιήσει κράτηση εισιτηρίου ηλεκτρονικά.



Inland Empire

As an actress begins to adopt the persona of her character in a film, her world becomes nightmarish and surreal.

Actors
Laura Dern, Jeremy Irons, Justin Theroux, Harry Dean Stanton

Director
David Lynch

Rating
9.0

Ticket Price
10.00 Euros

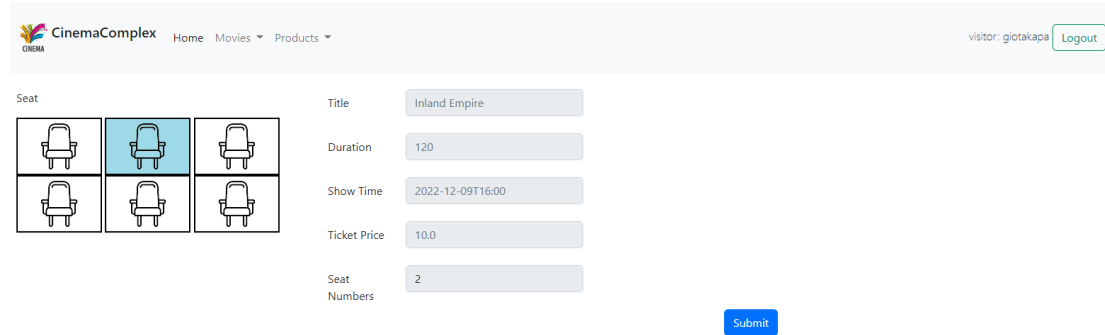
Duration
120'

Select Show dates
☒ 2022-12-09T16:00
☐ 2022-12-09T19:00
[Book Movie](#)

Πληροφορίες ταινίας

4. Μενού Κράτηση Θέσεων

Ο χρήστης επισκέπτης αφού έχει επιλέξει την ημερομηνία και ώρα προβολής έχει επιλέξει και τον αριθμό θέσης όπως παρατηρούμε στην παρακάτω εικόνα. Όπως παρατηρούμε δεξιά παρουσιάζονται ο τίτλος της ταινίας, η διάρκεια της ταινία, η τιμή εισιτηρίου και η θέση που έχει επιλέξει για την κράτηση.

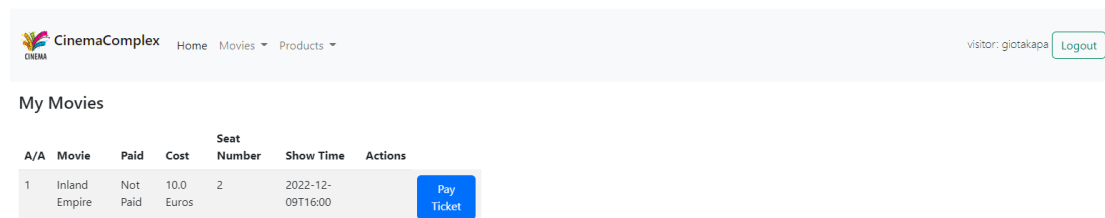


The screenshot shows the CinemaComplex website with the user logged in as 'giotakapa'. The 'My Movies' section displays a table with one entry for 'Inland Empire'. The 'Seat' section shows a 2x3 grid of seats, with the middle seat in the first row highlighted in blue. The 'Title' field is set to 'Inland Empire', 'Duration' is 120, 'Show Time' is 2022-12-09T16:00, 'Ticket Price' is 10.0, and 'Seat Numbers' is 2. A 'Submit' button is visible at the bottom right.

Κράτηση θέσης

4.1 Μενού Ταινιών του Χρήστη

Ο χρήστης επισκέπτης στην παρακάτω εικόνα έκανε κράτηση ένα εισιτήριο στην ταινία Inland Empire και εκκρεμεί η πληρωμή όπως φαίνεται στην παρακάτω εικόνα. Όπως παρατηρούμε υπάρχει ένα κουμπί actions όπου πατώντας ο χρήστης μπορεί να πραγματοποιήσει την online πληρωμή.

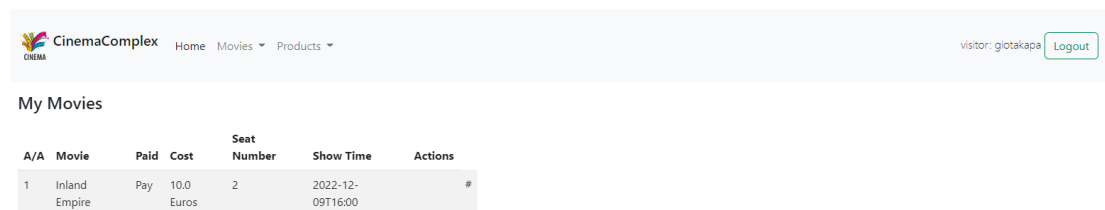


The screenshot shows the CinemaComplex website with the user logged in as 'giotakapa'. The 'My Movies' section displays a table with one entry for 'Inland Empire'. The 'Seat' section shows a 2x3 grid of seats, with the middle seat in the first row highlighted in blue. The 'Title' field is set to 'Inland Empire', 'Duration' is 120, 'Show Time' is 2022-12-09T16:00, 'Ticket Price' is 10.0, and 'Seat Numbers' is 2. A 'Submit' button is visible at the bottom right.

| A/A | Movie | Paid | Cost | Seat Number | Show Time | Actions |
|-----|---------------|----------|------------|-------------|------------------|------------|
| 1 | Inland Empire | Not Paid | 10.0 Euros | 2 | 2022-12-09T16:00 | Pay Ticket |

Οι ταινίες μου

Ο χρήστης επισκέπτης στην παρακάτω εικόνα πραγματοποίησε την πληρωμή συνεπώς το κουμπί Pay Ticket εξαφανίστηκε και δεν εμφανίζονται πλέον άλλες ενέργειες.

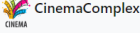


The screenshot shows the CinemaComplex website with the user logged in as 'giotakapa'. The 'My Movies' section displays a table with one entry for 'Inland Empire'. The 'Seat' section shows a 2x3 grid of seats, with the middle seat in the first row highlighted in blue. The 'Title' field is set to 'Inland Empire', 'Duration' is 120, 'Show Time' is 2022-12-09T16:00, 'Ticket Price' is 10.0, and 'Seat Numbers' is 2. A 'Submit' button is visible at the bottom right.

| A/A | Movie | Paid | Cost | Seat Number | Show Time | Actions |
|-----|---------------|------|------------|-------------|------------------|---------|
| 1 | Inland Empire | Pay | 10.0 Euros | 2 | 2022-12-09T16:00 | # |

Πληρωμή εισιτηρίων

Στην περίπτωση που ο ίδιος χρήστης ή κάποιος άλλος χρήστης επιλέξει την ίδια ταινία με ίδια ημερομηνία προβολής του εμφανίζει τις θέσεις στις οποίες έχει γίνει ήδη κράτηση άρα είναι κατειλημμένες και δεν μπορεί να τις επιλέξει όπως παρουσιάζεται στην παρακάτω εικόνα. Ο χρήστης giotakapa σε προηγούμενο βήμα επέλεξε την θέση 2 οπότε η θέση αυτή έχει πλέον μπλοκαριστεί και δεν μπορεί να την ξανά επιλέξει ούτε ο ίδιος πάλι ούτε άλλος χρήστης.


[Home](#)
[Movies](#)
[Products](#)

visitor: giotakapa
 Logout

Seat









Title

Duration

Show Time

Ticket Price

Seat Numbers

Η θέση είναι κατειλημμένη








5. Μενού Προϊόντων

Ο χρήστης επισκέπτης στην εφαρμογή έχει την δυνατότητα να πραγματοποιήσει ηλεκτρονική πληρωμή προϊόντων που υπάρχουν στο μπαρ του σινεμά.








Στην παραπάνω εικόνα παρουσιάζονται τα προϊόντα όπου ο χρήστης μπορεί να επιλέξει. Όπως παρατηρούμε ο χρήστης έχει επιλέξει ένα πατατάκι ,δύο ποπκορν και δύο λεμονάδες .

Products List

Foods

| Id | Name | Photo | Quantity | Price | Add to Cart |
|----|----------|---|--------------------------------|-------|-------------------------------------|
| 1 | chips |  | <input type="text" value="1"/> | 2.0 | <input checked="" type="checkbox"/> |
| 2 | popcorn |  | <input type="text" value="2"/> | 3.0 | <input checked="" type="checkbox"/> |
| 3 | candy |  | <input type="text"/> | 2.0 | <input type="checkbox"/> |
| 4 | cookies |  | <input type="text"/> | 1.0 | <input type="checkbox"/> |
| 5 | iceCream |  | <input type="text"/> | 3.0 | <input type="checkbox"/> |
| 6 | donats |  | <input type="text"/> | 3.0 | <input type="checkbox"/> |
| 7 | toast |  | <input type="text"/> | 2.0 | <input type="checkbox"/> |

Drinks

| Id | Name | Photo | Quantity | Price | Add to Cart |
|----|--------------|---|--------------------------------|-------|-------------------------------------|
| 8 | lemonade |  | <input type="text" value="2"/> | 1.0 | <input checked="" type="checkbox"/> |
| 9 | sprite |  | <input type="text"/> | 1.0 | <input type="checkbox"/> |
| 10 | cocaCola |  | <input type="text"/> | 1.0 | <input type="checkbox"/> |
| 11 | Orange Juice |  | <input type="text"/> | 2.0 | <input type="checkbox"/> |
| 12 | Cherry Juice |  | <input type="text"/> | 1.0 | <input type="checkbox"/> |
| 13 | water 50ml |  | <input type="text"/> | 0.5 | <input type="checkbox"/> |
| 14 | water 100ml |  | <input type="text"/> | 1.0 | <input type="checkbox"/> |

Μενού προϊόντων




| | | | | | |
|----|-------------|---|----------------------|-----|--------------------------|
| 6 | donats |  | <input type="text"/> | 3.0 | <input type="checkbox"/> |
| 7 | toast |  | <input type="text"/> | 2.0 | <input type="checkbox"/> |
| 13 | water 50ml |  | <input type="text"/> | 0.5 | <input type="checkbox"/> |
| 14 | water 100ml |  | <input type="text"/> | 1.0 | <input type="checkbox"/> |

Προσθήκη στο καλάθι αγορών

5.1 Μενού Καλάθι Αγορών

Ο χρήστης giotakara αφού μεταφέρεται στο καλάθι αγορών του βλέπει τα προϊόντα που έχει επιλέξει να του εμφανίζονται με τις αντίστοιχες ποσότητες, την συνολική τιμή για το κάθε προϊόν καθώς και το συνολικό κόστος που θα πρέπει να πληρώσει. Επιπλέον έχει την δυνατότητα να επεξεργαστεί τα προϊόντα του είτε διαγράφοντας ένα από αυτά, είτε μειώνοντας την ποσότητα που έχει επιλέξει. Τέλος, έχει την δυνατότητα να υποβάλει την παραγγελία του.

My Shopping Cart



| id | Name | Image | Quantity | Price | Delete Check |
|----|----------|---|--------------------------------|----------------------------------|--------------------------|
| 2 | popcorn |  | <input type="text" value="2"/> | <input type="text" value="6,0"/> | <input type="checkbox"/> |
| 8 | lemonade |  | <input type="text" value="2"/> | <input type="text" value="2,0"/> | <input type="checkbox"/> |
| 1 | chips |  | <input type="text" value="1"/> | <input type="text" value="2,0"/> | <input type="checkbox"/> |

TotalCost:

Καλάθι αγορών του χρήστη

The shopping cart of the giotakara

My Shopping Cart

| id | Name | Image | Quantity | Price | Delete Check |
|----|----------|---|--------------------------------|----------------------------------|--------------------------|
| 2 | popcorn |  | <input type="text" value="1"/> | <input type="text" value="3,0"/> | <input type="checkbox"/> |
| 8 | lemonade |  | <input type="text" value="2"/> | <input type="text" value="2,0"/> | <input type="checkbox"/> |

TotalCost:

Λειτουργίες στο καλάθι αγορών του χρήστη

Στην παραπάνω εικόνα παρατηρούμε ότι ο χρήστης διέγραψε το προϊόν πατατάκια και μείωσε κατά ένα την ποσότητα του ποπκορν. Συνεπώς η τιμή του συνολικού κόστους έχει αλλάξει κατά πολύ.

The screenshot shows the CinemaComplex website interface. On the left, the 'My Shopping Cart' is displayed with two items: popcorn (quantity 1, price 3.0) and lemonade (quantity 2, price 2.0). The total cost is 5.0. On the right, a 'CREDIT/DEBIT CARD PAYMENT' modal is open, showing fields for CARD NUMBER (12325364764), CARD EXPIRY (25/45), and CARD CVC (975). The total cost in the modal is 5.0€uros. Buttons for 'Pay Order' and 'Register Order' are visible at the bottom of the modal.

Υποβολή παραγγελίας

Παραπάνω ο χρήστης giotakapa υπέβαλε την παραγγελία του όπως παρατηρούμε αναδύθηκε ένα καινούριο παράθυρο το οποίο προτρέπει τον χρήστη να συμπληρώσει τα στοιχεία της κάρτας του για να πληρώσει την παραγγελία του. Στην εικόνα διαφαίνονται δύο κουμπιά το ένα είναι για να πληρώσει την παραγγελία του και το άλλο για να καταγραφεί στην βάση δεδομένων για μελλοντική πληρωμή.

The screenshot shows the CinemaComplex website interface with the message 'My Shopping cart is empty' displayed in the center. The top navigation bar includes the CinemaComplex logo, Home, Movies, and Products links. The user is identified as 'visitor: giotakapa' with a 'Logout' button.

Το καλάθι αγορών του χρήστη είναι άδειο

Όταν ο χρήστης επιλέξει είτε να πληρώσει την παραγγελία του είτε να εγγραφεί στην βάση για μελλοντικές πληρωμές το καλάθι αγορών του μένει άδειο και μπορεί να ξανακάνει από την αρχή την ίδια διαδικασία.

6. Μενού Παραγγελίας

Ο χρήστης έχει την δυνατότητα να επιλέξει να προηγηθεί είτε στην σελίδα που παρουσιάζονται όλα τα προϊόντα, είτε στο καλάθι αγορών του είτε στην παραγγελίες του. Παραπάνω ο χρήστης επιλέγει την κατηγορία παραγγελίες μου.

My Orders

| A/A | Paid | Cost | Ticket | Actions |
|-----|------|-----------|-------------------------------|---------|
| 1 | Paid | 5.0 Euros | Display Order | # |

Μενού παραγγελίας

Στην κατηγορία οι παραγγελίες μου ο χρήστης έχει την δυνατότητα να επεξεργαστεί τις πληροφορίες τις παραγγελίας του. Όπως παρατηρούμε στο παραπάνω παράδειγμα βλέπουμε πληροφορίες αν η παραγγελία είναι πληρωμένη, το κόστος της παραγγελίας. Σε περίπτωση που η παραγγελία δεν ήταν πληρωμένη θα έβγαζε στο πεδίο actions και την δυνατότητα στο χρήστη να πληρώσει συμπληρώνοντας ξανά τα στοιχεία της κάρτας του όπως βλέπουμε στην περίπτωση της παρακάτω εικόνας.

My Orders

| A/A | Paid | Cost | Ticket | Actions |
|-----|----------|-----------|-------------------------------|---------------------------|
| 1 | Paid | 5.0 Euros | Display Order | # |
| 2 | Not Paid | 3.0 Euros | Display Order | Pay Order |

Οι ενέργειες του χρήστη στις παραγγελίες του

Ο χρήστης επιπλέον πατώντας το κουμπί Display Order έχει την δυνατότητα να δει πιο αναλυτικά την παραγγελία του με τα προϊόντα που επέλεξε, την ποσότητα των προϊόντων του και το κόστος για το κάθε προϊόν ξεχωριστά.

My Orders

| A/A | Paid | Cost | Ticket | Actions |
|-----|----------|-----------|-------------------------------|---------------------------|
| 1 | Paid | 5.0 Euros | Display Order | # |
| 2 | Not Paid | 3.0 Euros | Display Order | Pay Order |

Order Details

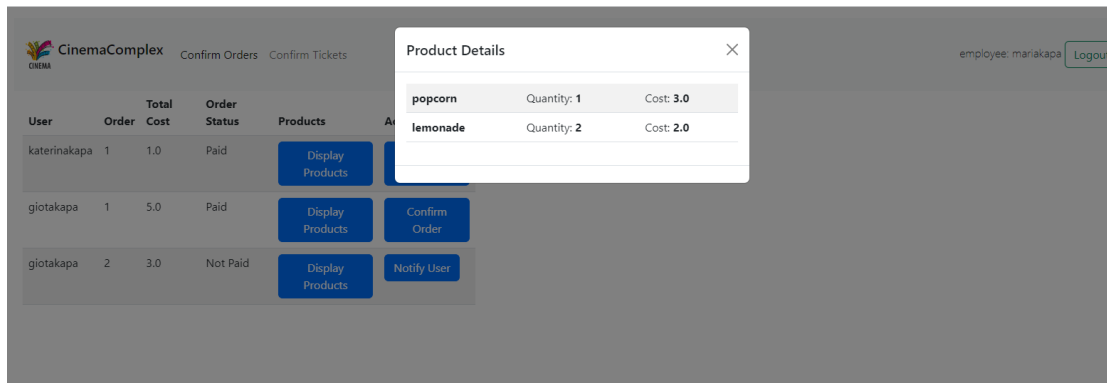
| | | |
|----------|-------------|-----------|
| popcorn | Quantity: 1 | Cost: 3.0 |
| lemonade | Quantity: 2 | Cost: 2.0 |

Λεπτομέρειες προϊόντων

7. Μενού Επιβεβαίωσης Παραγγελιών από τον υπάλληλο

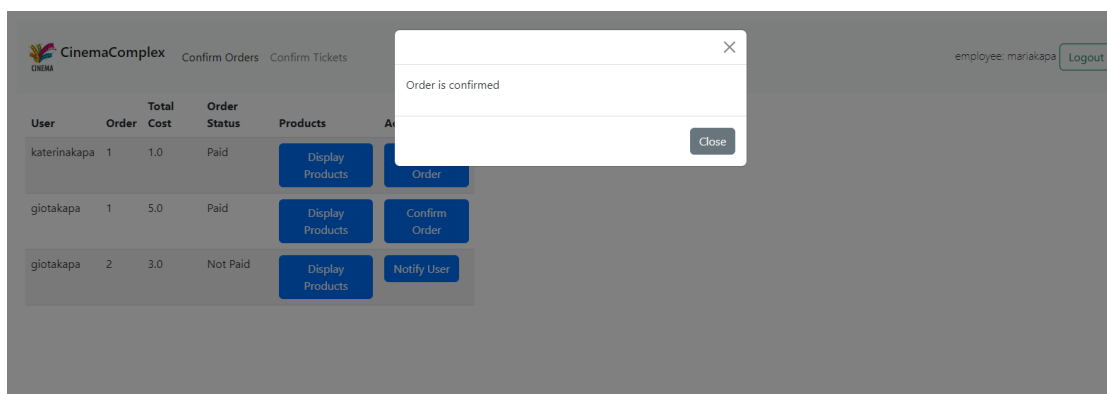
Ο χρήστης mariakapa που έχει συνδεθεί στο σύστημα με την ιδιότητα του υπαλλήλου όπως βλέπουμε στην παραπάνω εικόνα έχει την δυνατότητα να πληροφορηθεί για το ποιοι έχουν πραγματοποιήσει την παραγγελία, αν είναι πληρωμένη ή όχι και αντίστοιχα να επιβεβαιώσει την παραγγελία ή στην περίπτωση που

δεν είναι πληρωμένη να τον ειδοποιήσει ότι πρέπει να πληρωθεί και τέλος το συνολικό κόστος της παραγγελίας. Επιπλέον μπορεί να πληροφορηθεί για το ποια προϊόντα υπάρχουν στην παραγγελία και τις ποσότητες αυτών.



Λεπτομέρειες προϊόντων από το μενού επιβεβαίωση παραγγελία

Στην παραπάνω εικόνα ο χρήστης πατώντας το κουμπί Display Products αναδύεται ένα παράθυρο που πληροφορεί τον υπάλληλο mariakapa για τα προϊόντα που έχει επιλέξει ο χρήστης giotakapa στην πρώτη παραγγελία του.

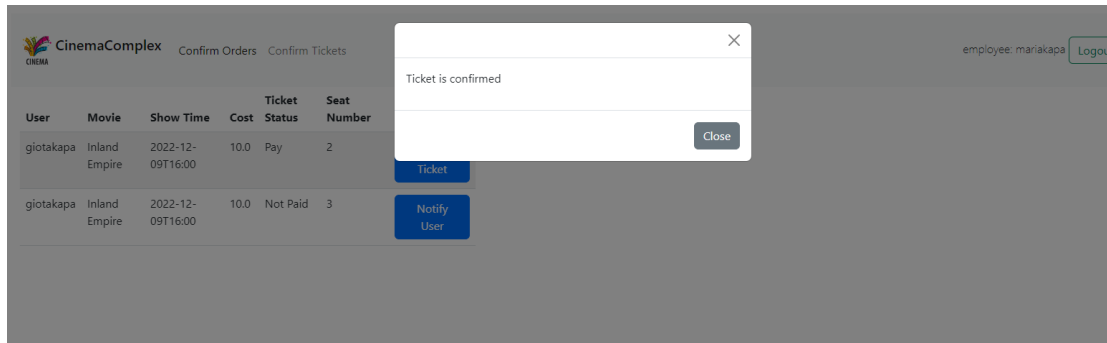


Η παραγγελία επιβεβαιώθηκε

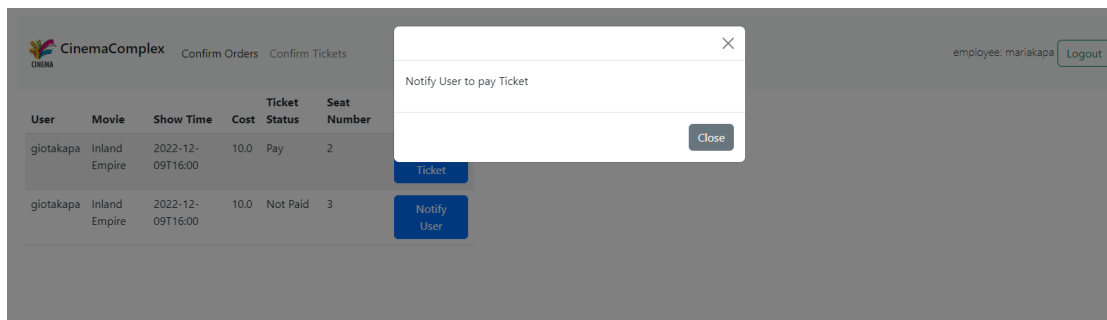
Στην παραπάνω εικόνα ο υπάλληλος mariakapa πατώντας το κουμπί Confirmed Order επιβεβαιώνει την παραγγελία και αναδύεται ένα παράθυρο που τον ενημερώνει ότι η παραγγελία επιβεβαιώθηκε.

7.1 Μενού Επιβεβαίωσης Εισιτηρίων από τον υπάλληλο

Όταν ο υπάλληλος maria kapa συνδεθεί στην εφαρμογή με την ιδιότητα του υπαλλήλου και πλοηγηθεί στο μενού Confirm Tickets του εμφανίζονται πληροφορίες σχετικά με τους χρήστες που έχουν πραγματοποιήσει είτε αγορά εισιτηρίου είτε δεν έχουν πληρώσει το εισιτήριό τους. Στην παρακάτω εικόνα ο υπάλληλος mariakapa πατώντας το κουμπί Confirm Ticket επιβεβαιώνει την αγορά εισιτηρίου του χρήστη giotakapa.



Το εισιτήριο επιβεβαιώθηκε



Ενημέρωσε το χρήστη να πληρώσει το εισιτήριο

Στην παραπάνω εικόνα ο υπάλληλος mariakapa πατώντας το κουμπί Notify User αναδύεται ένα παράθυρο που τον ενημερώνει να ειδοποιήσει τον χρήστη giotakapa.

8. Αρχιτεκτονική Συστήματος

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκαν οι παρακάτω τεχνολογίες

Γλώσσα προγραμματισμού : Java Version 11

Περιβάλλον ανάπτυξης λογισμικού : IntelliJ IDEA Ultimate edition 2022.1

Java Framework : Spring Boot 2.6.7

Hibernate ORM(Object Relational Mapping) για την επικοινωνία μεταξύ σχεσιακής βάσης δεδομένων και αντικειμένων τα οποία δημιουργούνται από κλάσεις,

Maven build automation tool για τον έλεγχο των dependencies της εφαρμογής,

Thymeleaf template engine για την δημιουργία HTML views.

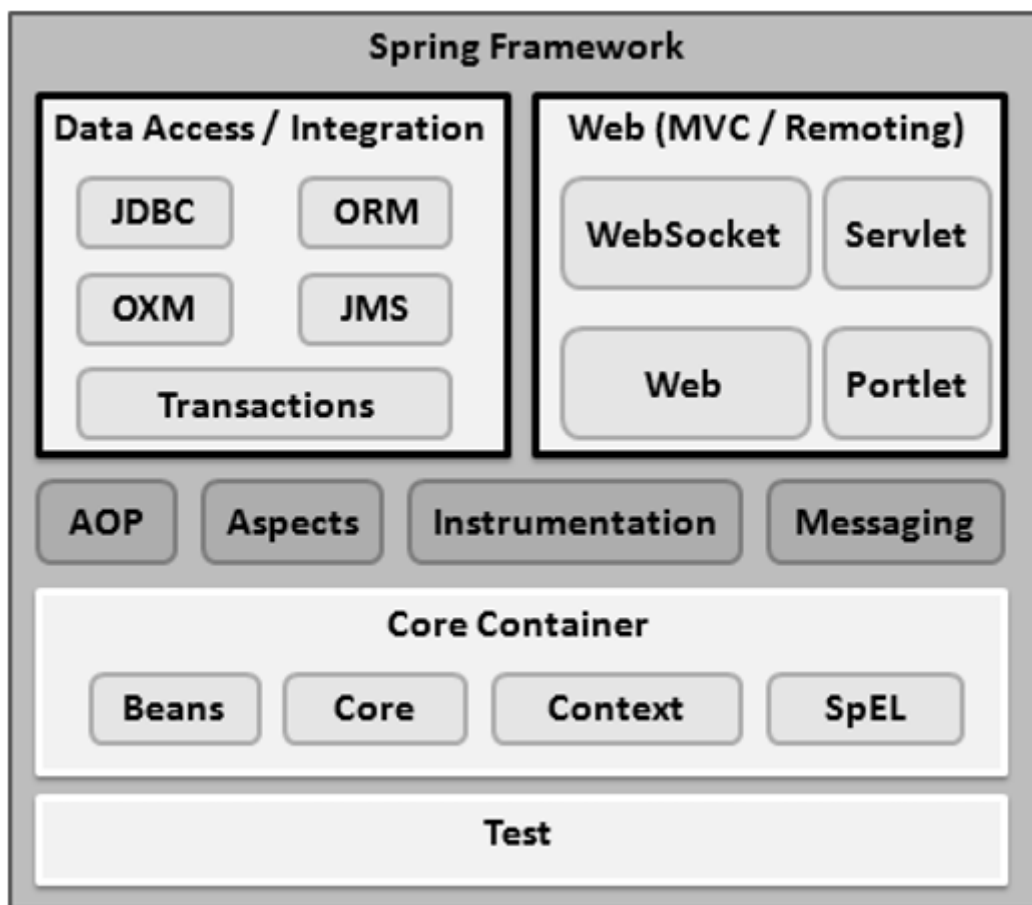
Βάση Δεδομένων : MySQL

8.1 Spring Framework

Το Spring Framework είναι μια πλατφόρμα για την ανάπτυξη Java εφαρμογών. Χρησιμοποιείται επίσης για την ανάπτυξη web εφαρμογών και είναι ανοιχτού κώδικα. Το Spring Framework είναι αρκετά δημοφιλές ως περιβάλλον ανάπτυξης εφαρμογών καθώς προσφέρει ταχύτητα στην επεξεργασία δεδομένων ,επαναχρησιμοποιήσιμο κώδικα ,ευκολία στην εξέταση μονάδας κώδικα (testing) , είναι αρκετά ελαφρύ και χρησιμοποιεί καλές πρακτικές προγραμματισμού όπως το POJO(Plain Old Java Object)-based model.

Πλεονεκτήματα που προσφέρει στους προγραμματιστές:

- Οι προγραμματιστές έχουν την δυνατότητα να αναπτύσσουν εφαρμογές μεγάλης εμβέλειας χρησιμοποιώντας POJO κλάσεις όπου δεν χρειάζονται κάποιον server εφαρμογών αλλά έναν servlet container όπως ο Tomcat.
- Είναι πιο εύκολο να πραγματοποιήσεις testing στον κώδικα σου καθώς όλα είναι διαμορφωμένα από το ίδιο το Framework.
- Επιπλέον το Spring Web Framework είναι βασισμένο πάνω στο MVC(Model-View-Controller) όπου είναι ένα μοντέλο αρχιτεκτονικής λογισμικού και οργάνωσης κώδικα που προσφέρει στον προγραμματιστή την ευκολία να έχει καλύτερο έλεγχο των συστημάτων που αλληλοεπιδρούν μεταξύ τους .
- Είναι ελαφρύ και προσφέρει στους προγραμματιστές λιγότερη κατανάλωση στα αποθέματα του υπολογιστή σε μνήμη και επεξεργαστές.



Spring Framework Architecture

8.1.1 Spring Framework-Architecture

Αποτελείται από το Spring Core ,Spring Beans, Spring Context, Expression Language.

- Το Core τμήμα προσφέρει τα στοιχειώδεις κομμάτια του framework όπως IoC containers και το κονσепт του Dependency Injection το οποίο είναι υπεύθυνο για την δημιουργία αυτόματων αντικειμένων μέσω των κατασκευαστών καθώς και για τον κύκλο ζωής τους στην εφαρμογή.

- Το Bean τμήμα προσφέρει το κόνσепт του Bean Factory όπου αντιπροσωπεύει έναν Spring Container οποίος είναι ένα parent interface του Application Context.
- Το Context τμήμα όπου είναι ένα μέσo για την πρόσβαση σε οποιοδήποτε αντικείμενο είναι ορισμένο και διαμορφωμένο.

8.1.2 Data Access/Integration

Η πρόσβαση στα δεδομένα αποτελείται από διάφορα επίπεδα όπως το JDBC,ORM,OXM,JMS και στα τμήματα συναλλαγής.

- Το **JDBC** module προσφέρει ένα αφαιρετικό επίπεδο που αφαιρεί την ανάγκη για κώδικα σχετιζόμενο με το JDBC
- Το **ORM** module προσφέρει ενσωματωμένα επίπεδα για δημοφιλείς relational mapping APIs, συμπεριλαμβάνοντας JPA, JDO, Hibernate, and iBatis.
- Το **OXM** module προσφέρει ένα αφαιρετικό επίπεδο που υποστηρίζει Object/XML mapping εφαρμογής για JAXB, Castor, XMLBeans, JiBX and XStream.
- Το Java Messaging Service **JMS** module περιλαμβάνει χαρακτηριστικά για την παραγωγή και κατανάλωση μηνυμάτων.
- Το Transaction module υποστηρίζει προγραμματιστικές και δηλωτικές συναλλαγές για τις κλάσεις που εφαρμόζουν ειδικά interface αλλά και για όλες τις POJO κλάσεις.

Όλα αυτά τα modules προσφέρουν στους προγραμματιστές να μπορούν με ευκολία να αλληλοεπιδράσουν με την βάση καθώς και να επιλύουν και διάφορα άλλα προβλήματα όπως το exception handling.

8.1.3 WEB

- Το Web επίπεδο αποτελείται από το Web, Web-MVC, Web-Socket,Web-modules.
- Το **Web module** προσφέρει web-oriented integration features όπως το multipart file-upload functionality καθώς και την αρχικοποίηση του IoC Container χρησιμοποιώντας servlet listeners και web-oriented application context.
- Το **Web-MVC** module περιλαμβάνει Spring's Model-View-Controller (MVC) για τις web εφαρμογές .
- Το **Web-Socket** module προσφέρει υποστήριξη για WebSocket-based, two-way επικοινωνία μεταξύ client και server στις web εφαρμογές.

8.1.4 Aspect Oriented Programming

Το κόνσепт του Aspect Oriented Programming είναι ένα πολύ σημαντικό χαρακτηριστικό του Spring Framework το οποίο επιτρέπει να ορίσεις μεθόδους παρεμπόδισης και σημείων που επιτρέπουν την αποσύνδεση κώδικα που υλοποιεί λειτουργίες που πρέπει να διαχωριστούν. Συνεπώς μπορούμε να δημιουργήσουμε ανεξάρτητα τμήματα κώδικα ανάλογα με την λειτουργικότητα που εξυπηρετούν .

Το Aspects module προσφέρει ενσωμάτωση με το AspectJ το οποίο είναι ένα πολύ δυνατό AOP framework .

8.1. 5 Messaging

Το Messaging module περιλαμβάνει σημαντικές αφηρημένες έννοιες οι οποίες προέρχονται από το Spring Integration project όπως το Message, Message Channel, Message Handler και άλλες και αποτελούν την

βάση για messaging-based εφαρμογές. Το Messaging module περιλαμβάνει μια ομάδα από σχόλια(annotation) για την αντιστοίχιση μηνυμάτων με μεθόδους παρόμοια με το Spring MVC model το οποίο επίσης βασίζεται σε annotations.

8.1.6 Test

Το Spring Test module υποστηρίζει την εξέταση κώδικα των Spring components με το Junit ή με το TestNG framework. Παρέχει συνεπή φόρτωση των Spring Application Contexts και προσωρινή αποθήκευση(caching) αυτών των contexts. Επίσης παρέχει ψεύτικα αντικείμενα(mock objects) τα οποία μπορούν να χρησιμοποιηθούν για την εξέταση του κώδικα σε περιβάλλον απομόνωσης

8.2 Spring Boot Framework

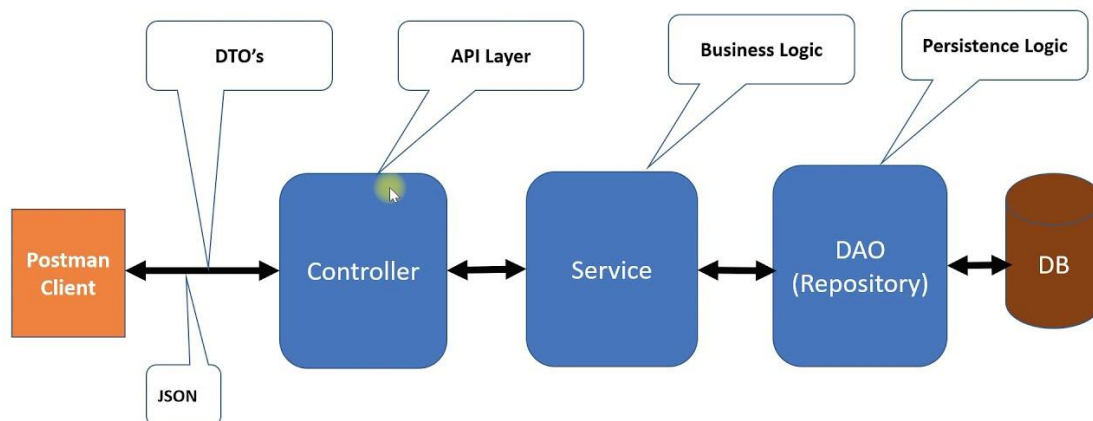
Το Spring Boot Framework αποτελεί μια επέκταση του Spring Framework παρέχοντας όμως την δυνατότητα στον προγραμματιστή να αναπτύσσει εφαρμογές πιο γρήγορα με ελάχιστες διαμορφώσεις. Το Spring Boot Framework σχεδιάστηκε ώστε:

- Να μην χρειάζεται η διαμόρφωση σύνθετων XML αρχείων
- Η ανάπτυξη Spring εφαρμογών με ένα πιο εύκολο τρόπο,
- για να μειώσει τον χρόνο του development και η εφαρμογή να τρέχει πιο γρήγορα
- Προσφέρει έναν πιο εύκολο τρόπο για να ξεκινήσεις μια εφαρμογή

Επιπλέον το Spring Boot προσφέρει

- έναν πιο ευέλικτο τρόπο διαμόρφωσης των Java Beans , XML configurations και των συναλλαγών με την βάση.
- Ένα δυναμικό επεξεργασίας και διαχείρισης των Rest end points.
- Την ευκολία τα πάντα να είναι αυτοματοποιημένα χωρίς χειροκίνητες διαμορφώσεις.
- Μηνύματα(annotation) τα οποία είναι της spring εφαρμογής
- Διευκολύνει το κόνσεπτ του dependency management
- Περιλαμβάνει ενσωματωμένους servlet container.

Spring Boot Application Architecture



By Ramesh Fadatore (Java Guides)

Spring Boot Architecture

8.2.1 Spring Boot Architecture

Στην παραπάνω εικόνα βλέπουμε πως αλληλοεπιδρούν μεταξύ τους τα διάφορα επίπεδα μια εφαρμογής που έχει υλοποιηθεί σε Spring Boot Framework. Αρχικά έχουμε το client side κομμάτι που δέχεται τα https αιτήματα τα οποία μπορεί να είναι get που είναι για να πάρουμε δεδομένα από την βάση, post για να δημιουργήσουμε δεδομένα στην βάση, put ή update για να αλλάζουμε τα ήδη υπάρχον δεδομένα και delete για την διαγραφή των δεδομένων. Έπειτα έχουμε τους controllers οι οποίοι αλληλοεπιδρούν με το service επίπεδο όπου εκεί είναι σχεδιασμένη όλη η λειτουργικότητα της εφαρμογής και οι οποίοι παραλαμβάνουν τα αιτήματα και τα στέλνουν πίσω. Το service επίπεδο αλληλοεπιδρά με το DAO repository το οποίο επικοινωνεί με την βάση δεδομένων και διαχειρίζεται τα δεδομένα.

Η παρούσα εφαρμογή έχει δημιουργηθεί με τον τρόπο που απεικονίζεται στην παραπάνω εικόνα. Πιο συγκεκριμένα στην εφαρμογή μας έχουμε τρία επίπεδα : Controller, Service, DAO repository τα οποία αλληλοεπιδρούν μεταξύ τους δημιουργώντας αντικείμενα για το κάθε επίπεδο μέσω της τεχνικής Dependency Injection επιπλέον γίνεται χρήση και των annotations του Spring Boot τα οποία θα παρουσιάσω παρακάτω.

8.2.2 Dependency Injection

Με την τεχνική του dependency injection ο προγραμματιστής δεν χρειάζεται να δημιουργήσει σε κάθε επίπεδο αντικείμενα του προηγούμενου επιπέδου αλλά τα αντικείμενα αυτά περνάνε από το ένα επίπεδο στο άλλο αυτοματοποιημένα χρησιμοποιώντας το annotation @Autowired. Με αυτό τον τρόπο μειώνεται η εξάρτηση των κλάσεων από άλλες κλάσεις έχουμε δηλαδή χαλαρή σχέση μεταξύ των κλάσεων το οποίο ονομάζεται decoupling.

```
1 package gr.mariakapa.cinema.Entity;
2
3 import ...
4
12
13
14 @Entity
15 @Table
16 public class Movie {
17
18
19     @Id
20     @GeneratedValue(strategy = GenerationType.IDENTITY)// AUTOINCREMENT 1,2,3 NOT SEQUENCE IN TABLE
21     @Column(name = "id_movie")
22     private int idMovie;
23
```

Entity movie table

Στην παραπάνω εικόνα παρατηρούμε ότι έχουμε μια οντότητα ταινία η οποία θα δημιουργήσει ένα πίνακα στην βάση μας

```
1 package gr.mariakapa.cinema.Repository;
2
3 import ...
4
11
12 4 usages
13 @Repository
14 public interface MovieRepository extends JpaRepository<Movie,Integer> {
15     public Movie findMovieByTitle(String title);
16
17     1 usage
18     Movie findByIdMovie(int idMovie);
19
```

Movie Repository

Έπειτα θα δημιουργήσουμε ένα repository το οποίο θα αλληλοεπιδρά με την βάση μας , όπως βλέπουμε έχουμε και μια μέθοδο findByIdMovie(int idMovies) η οποία θα λάβει ως παράμετρο ένα id και θα επιστρέφει μια ταινία η οποία έχει το id που στείλαμε ως παράμετρο στην μέθοδο.

```
1 package gr.mariakapa.cinema.Service;
2
3
4 import ...
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 @Service
24 public class MovieService {
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Movie Service

Όπως παρατηρούμε παραπάνω δημιουργούμε με την τεχνική του dependency injection ένα αντικείμενο του repository το οποίο μας συσχετίζει το service layer με το repository layer το οποίο αφορά την λειτουργικότητα με την βάση μας.

```
2
3 import ...
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18 @Controller
19 @RequestMapping("/")
20
21 public class MoviesController {
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Movie Controller

Τέλος θα δημιουργήσουμε τον Movie Controller ο οποίος θα αλληλοεπιδράσει με το service layer μέσω του αντικειμένου movieService στο οποίο όπως παρατηρούμε έχει το annotation @Autowire έχουμε δηλαδή ξανά την τεχνική του dependency injection.

8.2.3 Spring Boot Annotation

@Controller : Χρησιμοποιείται στην αρχιτεκτονική MVC(Model View Controller) για κλάσεις οι οποίες έχουν τον ρόλο του Controller σε μια εφαρμογή, δηλαδή διαθέτουν μεθόδους οι οποίες επιστρέφουν HTML Views.

@RequestMapping : Χρησιμοποιείται στην μεθόδου και πραγματοποιεί την αντιστοίχιση(mapping) Web Requests και μεθόδων. Δημιουργεί ένα URL για έναν συγκεκριμένο controller ο οποίος θα χρησιμοποιήσει για μια λειτουργικότητα.

@Autowired : Είναι το annotation που χρησιμοποιεί την τεχνική του dependency injection. Δηλώνουμε μια μεταβλητή μιας κλάσης και χρησιμοποιώντας το συγκεκριμένο annotation πάνω στην μεταβλητή παράγεται το αντικείμενο της κλάσης.

@RestController : Παρέχει την ίδια περίπου λειτουργικότητα όπως και ο Controller μόνο που εδώ δεν επιστρέφει html σελίδα αλλά αντικείμενα με την μορφή Json . Χρησιμοποιείται η αρχιτεκτονική REST(Representational State Transfer), δηλαδή η επικοινωνία μεταξύ μηχανημάτων(client, server) γίνεται μέσω πρωτοκόλλου HTTP.

@GetMapping : Χρησιμοποιείται σε επίπεδο μεθόδου για την αντιστοίχιση(mapping) HTTP GET Requests και μεθόδων οι οποίες χειρίζονται αυτά τα αιτήματα. Αυτό το annotation αποτελεί την συντομογραφία του **@RequestMapping(method = RequestMethod.GET)** και χρησιμοποιείται για να φέρνει δεδομένα.

@DeleteMapping: Χρησιμοποιείται επίπεδο μεθόδου για την αντιστοίχιση(mapping) HTTP DELETE Requests και μεθόδων οι οποίες χειρίζονται αυτά τα αιτήματα. Αυτό το annotation αποτελεί την συντομογραφία του **@RequestMapping(method = RequestMethod.DELETE)** και χρησιμοποιείται για να διαγράφει δεδομένα.

@PutMapping : Χρησιμοποιείται σε επίπεδο μεθόδου για την αντιστοίχιση(mapping) HTTP PUT Requests και μεθόδων οι οποίες χειρίζονται αυτά τα αιτήματα. Αυτό το annotation αποτελεί την συντομογραφία του **@RequestMapping(method = RequestMethod.PUT)**

@PostMapping: Χρησιμοποιείται σε επίπεδο μεθόδου για την αντιστοίχιση(mapping) HTTP POST Requests και μεθόδων οι οποίες χειρίζονται αυτά τα αιτήματα. Αυτό το annotation αποτελεί την συντομογραφία του **@RequestMapping(method = RequestMethod.POST)** και χρησιμοποιείται για να δημιουργεί δεδομένα στην βάση .

@Service : Χρησιμοποιείται σε επίπεδο κλάσης. Χαρακτηρίζει την κλάση σαν Service και περιλαμβάνει όλο το business κομμάτι της εφαρμογής. Επίσης το **@Service** annotation δηλώνει την κλάση σαν Bean στο Spring Container και έτσι αντικείμενα(objects) της κλάσης αυτής μπορούν να χρησιμοποιηθούν μέσω Dependency Injection σε άλλα μέρη του προγράμματος.

@Entity: Χρησιμοποιείται για να δηλώσουμε ότι αυτό θα είναι μια οντότητα. Χρησιμοποιείται στο απλό επίπεδο της κλάσης που θα δημιουργήσει οντότητες στην βάση.

@Table: Χρησιμοποιείται στην κλάση για να δημιουργηθεί ο πίνακας στην βάση δεδομένων μας.

@Repository : Χρησιμοποιείται στην κλάση για να δηλώσουμε ότι θα αλληλοεπιδρά με την βάση μας. Επίσης αυτό το annotation δηλώνει την κλάση σαν Bean στο Spring Container και έτσι αντικείμενα(objects) της κλάσης αυτής μπορούν να χρησιμοποιηθούν μέσω Dependency Injection σε άλλα μέρη του προγράμματος.

@SpringBootApplication : Χρησιμοποιείται στην αρχική κλάση της Spring Boot εφαρμογής. Αυτό που κάνει είναι εξέταση της εφαρμογής για αναζήτηση κλάσεων οι οποίες θα προστεθούν στο Spring Container , έτσι ώστε να μπορούν να δημιουργηθούν Beans από αυτές.

8.2.4 Hibernate

Είναι ένα ORM(Object Relational Mapping) εργαλείο για την γλώσσα προγραμματισμού Java και αποτελεί υλοποίηση του Java Persistence API(JPA), ακολουθώντας τα κοινά πρότυπα τα οποία παρέχονται από αυτό. Παρέχει ένα framework για την αντιστοίχιση-απεικόνιση ενός μοντέλου αντικειμενοστραφούς τομέα σε μια σχεσιακή βάση δεδομένων. Το Hibernate χειρίζεται τα προβλήματα αντιστοίχισης-απεικόνισης μεταξύ αντικειμένων και βάσης δεδομένων αντικαθιστώντας τις απευθείας προσβάσεις στην βάση δ. οι οποίες αφορούν την αποθήκευση(persistence), με μεθόδους οι οποίες χειρίζονται αντικείμενα. Εσωτερικά

χρησιμοποιείται το JDBC(Java Database Connectivity) API για την επικοινωνία με την βάση δεδομένων. Ουσιαστικά αποτελεί ένα επιπλέον επίπεδο αφηρημένης έννοιας(abstraction) πάνω από το JDBC, επιτρέποντας στον προγραμματιστή να μην χρησιμοποιεί χαμηλού επιπέδου(low level) JDBC κλήσεις.

Το κύριο χαρακτηριστικό του Hibernate είναι η αντιστοίχιση-απεικόνιση Java κλάσεων σε πίνακες σχεσιακών βάσεων δεδομένων και η αντιστοίχιση-απεικόνιση τύπων δεδομένων της Java σε τύπους δεδομένων SQL(Structured Query Language). Επίσης παρέχει διευκολύνσεις τόσο για ερωτήσεις(queries) προς την βάση δ., όσο και για την ανάκτηση δεδομένων τα οποία προέρχονται από αυτά τα ερωτήματα. Παράγει κλήσεις SQL και απαλλάσσει τον προγραμματιστή από την ανάγκη χειροκίνητου χειρισμού και μετατροπής του εκάστοτε αποτελέσματος από την βάση δ. σε Java αντικείμενο.

Η αντιστοίχιση-απεικόνιση Java κλάσεων σε πίνακες βάσης δεδομένων υλοποιείται με την παραμετροποίηση ενός XML αρχείου ή χρησιμοποιώντας Java annotations. Όταν χρησιμοποιείται αρχείο XML, το Hibernate μπορεί να παράγει έναν σκελετό πηγαίου κώδικα για τις κλάσεις που αφορούν την αποθήκευση δεδομένων, κάτι το οποίο είναι βοηθητικό όταν χρησιμοποιούνται annotations. Επίσης το Hibernate μπορεί να χρησιμοποιήσει το XML αρχείο ή τα Java annotations για την συντήρηση του σχήματος(schema) της βάσης .

Τέλος υποστηρίζει λειτουργίες για την διευθέτηση σχέσεων one-to-many(1:N) και many-to-many(M:N) μεταξύ κλάσεων. Το Hibernate παρέχει μια γλώσσα εμπνευσμένη από την SQL, η οποία καλείται Hibernate Query Language(HQL), για την συγγραφή ερωτήσεων τύπου SQL προς τα αντικείμενα δεδομένων(data objects) του Hibernate. Η HQL αποτελεί την αντικειμενοστραφή έκδοση της γλώσσας SQL. Παράγει ερωτήσεις προς την βάση . ανεξάρτητες από την τεχνολογία-έκδοση της βάσης δ.

Χωρίς αυτήν την δυνατότητα, οποιαδήποτε αλλαγή στην βάση δ. θα απαιτούσε και την αλλαγή των μεμονωμένων ερωτήσεων SQL, οδηγώντας σε προβλήματα συντήρησης. Μια αντικειμενοστραφής εναλλακτική σε σχέση με την γλώσσα HQL αποτελεί το ερώτημα βάσει κριτηρίων(Criteria Query). Το ερώτημα βάσει κριτηρίων χρησιμοποιείται για την τροποποίηση και τον παροχή περιορισμών σε αντικείμενα. Το Hibernate παρέχει την δυνατότητα αποθήκευσης σε POJOs. Η μόνη προϋπόθεση για να μπορεί μία κλάση να χρησιμοποιηθεί προς αποθήκευση είναι η ύπαρξη κατασκευαστή χωρίς παραμέτρους, ο οποίος δεν είναι απαραίτητο να είναι public. Οι συλλογές(Collections) οι οποίες αποτελούνται από αντικείμενα δεδομένων αποθηκεύονται σε Java κλάσεις συλλογών(Collection Classes), όπως οι υλοποιήσεις των Set και List interfaces. Επίσης η τεχνολογία Java generics, με την οποία δίνεται η δυνατότητα να δημιουργήσουμε κλάσεις, διεπαφές(interfaces) και μεθόδους οι οποίες θα λειτουργούν με έναν τρόπο ανεξάρτητο από τον τύπο των δεδομένων τους(type-safe) όπως και με διάφορα είδη δεδομένων.

8.2.5 JPA Cascade Type

Στο Hibernate, οι σχέσεις μεταξύ οντοτήτων(entities) συχνά εξαρτώνται από την ύπαρξη κάποιας άλλης οντότητας. Για παράδειγμα στην περίπτωση της σχέσης ανθρώπου-διεύθυνσης κατοικίας, όπου χωρίς την οντότητα του ανθρώπου η οντότητα της διεύθυνσης κατοικίας δεν έχει κανένα νόημα. Για αυτόν ακριβώς τον λόγο χρησιμοποιούμε την διαδικασία της αλληλουχίας με μορφή καταρράκτη(Cascading). Έτσι όταν εκτελέσουμε κάποια ενέργεια σε μια συγκεκριμένη οντότητα, η ίδια ενέργεια θα εφαρμοστεί και στην οντότητα με την οποία συσχετίζεται η οντότητα αυτή.

Οι τύποι Cascade οι οποίοι υποστηρίζονται από το Hibernate ORM είναι οι εξής :

- ALL(CascadeType.ALL) : Μεταδίδει όλες τις ενέργειες από μία οντότητα γονέα σε μία οντότητα παιδί.
- PERSIST(CascadeType.PERSIST):Μεταδίδει την ενέργεια αποθήκευσης(persist) από μία οντότητα γονέα σε μία οντότητα παιδί.

- MERGE(CascadeType.MERGE) : Μεταδίδει την ενέργεια συγχώνευσης(merge) από μία οντότητα γονέα σε μία οντότητα παιδί. Με την ενέργεια αυτή γίνεται ενημέρωση κάποιου αντικείμενου με νέες τιμές και στην συνέχεια ενημερώνεται η βάση δεδομένων με βάση αυτό το αντικείμενο.
- REMOVE(CascadeType.REMOVE) : Μεταδίδει την ενέργεια διαγραφής(remove) από μία οντότητα γονέα σε μία οντότητα παιδί. Η ενέργεια αυτή διαγράφει την εγγραφή η οποία αντιστοιχεί στην συγκεκριμένη οντότητα, από την βάση δεδομένων όπως επίσης και από το Persistence Context.
- DETACH(CascadeType.DETACH) : Μεταδίδει την ενέργεια απόσπασης(detach) από μία οντότητα γονέα σε μία οντότητα παιδί. Αυτή η ενέργεια αφαιρεί την οντότητα από το Persistent Context.
- LOCK(CascadeType.LOCK) : Μεταδίδει την ενέργεια κλειδώματος(lock) από μία οντότητα γονέα σε μία οντότητα παιδί. Με την ενέργεια αυτή η οντότητα γονέας και η οντότητα παιδί επανέρχονται ξανά στο Persistence Context.
- REFRESH(CascadeType.REFRESH): Μεταδίδει την ενέργεια ανανέωσης(refresh) από μία οντότητα γονέα σε μία οντότητα παιδί. Με την ενέργεια αυτή γίνεται ξανά ανάγνωση των τιμών μιας οντότητας από την βάση δεδομένων.
- REPLICATE(CascadeType.REPLICATE): Μεταδίδει την ενέργεια αντιγραφής(replicate) από μία οντότητα γονέα σε μία οντότητα παιδί. Η ενέργεια αυτή χρησιμοποιείται όταν υπάρχουν παραπάνω από μία πηγές δεδομένων και χρειαζόμαστε συγχρονισμό των δεδομένων αυτών.
- SAVE_UPDATE(CascadeType.SAVE_UPDATE) : Μεταδίδει τις ενέργειες save, update και saveOrUpdate από μία οντότητα γονέα σε μία οντότητα παιδί. Οι παραπάνω ενέργειες δεν αποτελούν μέρος του JPA και χρησιμοποιούνται αποκλειστικά στο Hibernate

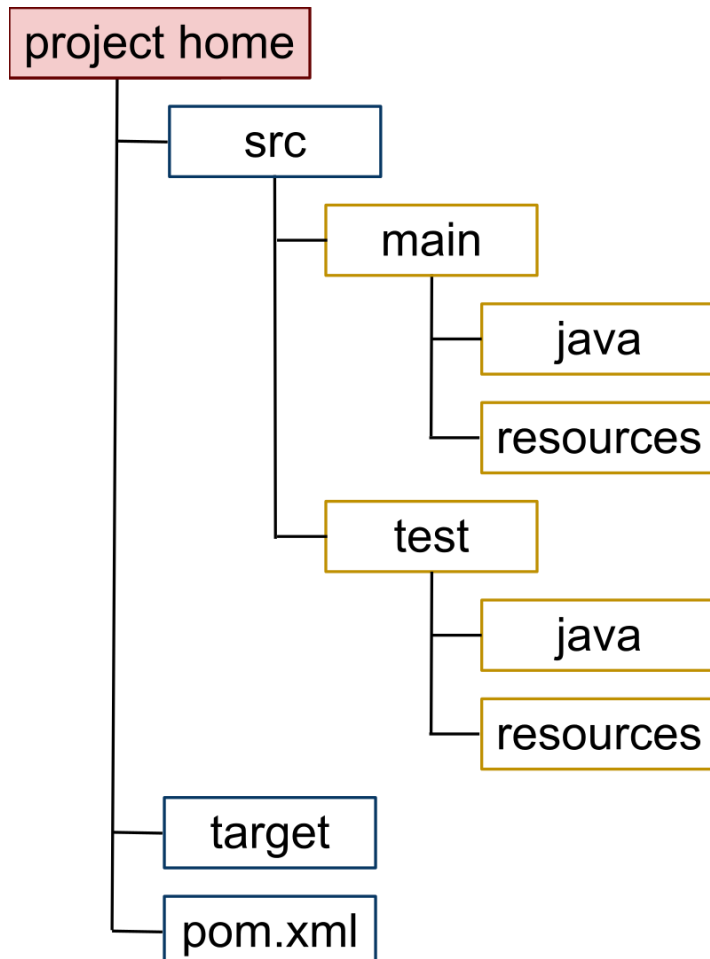
8.3 Maven

Η κατασκευή ενός έργου(project) λογισμικού συνήθως αποτελείται από εργασίες όπως το κατέβασμα(downloading) διάφορων εξαρτήσεων(dependencies), την προσθήκη JAR(Java ARchives) αρχεία στην διαδρομή του project(classpath), την μεταγλώττιση(compiling) πηγαίου(source) κώδικα σε δυαδικό(binary) κώδικα, την εκτέλεση δοκιμών(testing), το πακετάρισμα(packaging) μεταγλωττισμένου κώδικα σε αρχεία τύπου JAR, WAR ή ZIP και την ανάπτυξη αυτών των αρχείων σε διακομιστές εφαρμογών(application server) ή αποθήκες δεδομένων(repository). Το Apache Maven αυτοματοποιεί τέτοιου είδους εργασίες, ελαχιστοποιώντας τον κίνδυνο ανθρώπινων λαθών κατά την διαδικασία κατασκευής λογισμικού χειροκίνητα και διαχωρίζει την διαδικασία μεταγλώττισης και πακεταρίσματος του κώδικα από αυτήν της κατασκευής κώδικα. Η διαχείριση των dependencies όπως αναφέρθηκε, αποτελεί ένα σημαντικό χαρακτηριστικό του Maven.

Τα dependencies είναι αρχεία όπως JAR, ZIP κ.τ.λ. τα οποία είναι απαραίτητα στο project έτσι ώστε να μπορεί να μεταγλωττιστεί, να γίνει build, να τρέξει τεστ και να εκτελεστεί. Με λίγα λόγια τα dependencies είναι εξωτερικές βιβλιοθήκες(libraries) τις οποίες χρησιμοποιεί το project. Όταν εκτελούμε ένα build για παράδειγμα, τα dependencies αναλύονται(resolved) και στην συνέχεια φορτώνονται(loaded) από κάποιο repository, τοπικό ή απομακρυσμένο. Τα κύρια χαρακτηριστικά του Maven είναι :

- Απλή διαδικασία εγκατάστασης του project η οποία ακολουθεί δοκιμασμένες πρακτικές(best practices) : Το Maven αποφεύγει όσο το δυνατόν περισσότερες παραμετροποιήσεις, παρέχοντας έτοιμα project templates τα οποία ονομάζονται archetypes.
- Διαχείριση dependencies : Περιλαμβάνει αυτόματη ενημέρωση, κατέβασμα και έλεγχο συμβατότητας των dependencies όπως επίσης και την αναφορά του κλεισίματος του dependency.
- Απομόνωση των dependencies και των plugins ενός project : Με το Maven τα project dependencies ανακτώνται από τα dependency repositories ενώ τα dependencies των plugins ανακτώνται από τα plugin repositories. Αυτό έχει σαν αποτέλεσμα λιγότερες συγκρούσεις όταν τα plugins κατεβάζουν επιπρόσθετα dependencies.

• Κεντρικό σύστημα repository : Τα dependencies του project μπορούν να φορτωθούν από το τοπικό σύστημα αρχείων(file system) από δημόσια repositories, όπως το Maven central. Η παραμετροποίηση ενός Maven project πραγματοποιείται μέσω ενός Project Object Model(POM), το οποίο αναπαρίσταται από το αρχείο pom.xml. Το POM περιγράφει το project, διαχειρίζεται τα dependencies και παραμετροποιεί τα plugins για την διαδικασία build του λογισμικού. Το POM επίσης ορίζει τις σχέσεις μεταξύ των modules σε project τα οποία περιλαμβάνουν πολλαπλά modules.



A directory structure for a java project auto-generated by Maven

Το Maven χρησιμοποιεί μια ομάδα από αναγνωριστικά(identifiers) τα οποία καλούνται και συντεταγμένες(coordinates), για αναγνωρίσει μοναδικά ένα project και να προσδιορίσει πως αυτό το project θα μετατραπεί σε πακέτο(package). Ακολουθεί η επεξήγηση των identifiers : • groupId : Ένα μοναδικό όνομα της εταιρίας ή ομάδας η οποία δημιούργησε το project

- artifactId : Ένα μοναδικό όνομα για το συγκεκριμένο project
- version : Μία έκδοση του project
- packaging : Μία μέθοδος δημιουργίας package, όπως WAR, JAR, ZIP κ.τ.λ. Τα πρώτα τρία αναγνωριστικά(groupId, artifactId, version) συνδυάζονται για να δημιουργηθεί το μοναδικό identifier και είναι ο

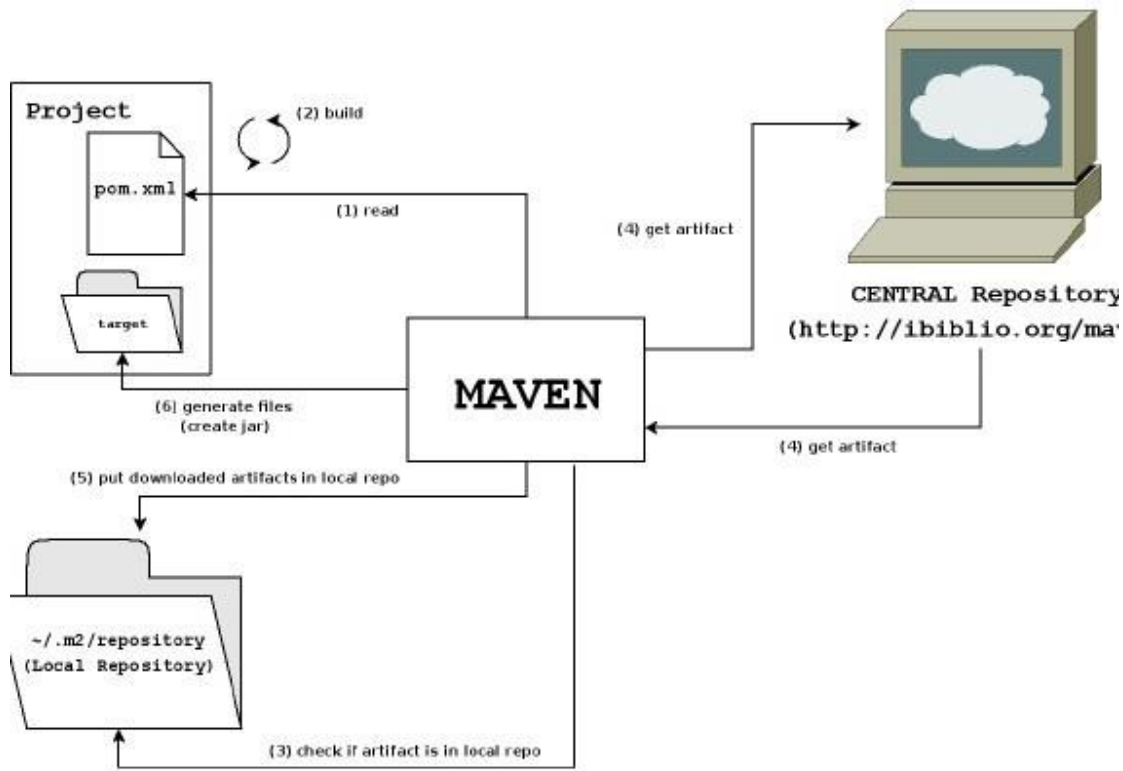
μηχανισμός με τον οποίο προσδιορίζουμε ποιες εκδόσεις εξωτερικών βιβλιοθηκών θα χρησιμοποιήσει κάποιο project. Για την δήλωση των dependencies εξωτερικών βιβλιοθηκών χρειάζεται να παρέχουμε το groupId, το artifactId και το version της βιβλιοθήκης, όπως φαίνεται και στο παραπάνω παράδειγμα. Κατά την διάρκεια της επεξεργασίας των dependencies, το Maven θα κατεβάσει την Spring βιβλιοθήκη στο τοπικό Maven repository. Το τμήμα build είναι επίσης πολύ σημαντικό τμήμα στο Maven POM. Παρέχει πληροφορίες σχετικά με το προεπιλεγμένο σκοπό(goal) του Maven, τον φάκελο για το μεταγλωττισμένο project και το τελικό όνομα της εφαρμογής. Κάθε Maven build ακολουθεί έναν συγκεκριμένο κύκλο ζωής(lifecycle). Μπορούν να εκτελεστούν πολλαπλά build lifecycle goals όπως αυτό της μεταγλώττισης του κώδικα του project, της δημιουργίας ενός package και της εγκατάστασης κάποιου αρχείου στο τοπικό dependency repository.

Η ακόλουθη λίστα παρουσιάζει τις πιο σημαντικές φάσεις του Maven κύκλου ζωής :

- validate : Ελέγχει την ορθότητα του project
- compile : Μεταγλωττίζει τον πηγαίο κώδικα σε δυαδικά αντικείμενα(artifacts)
- test : Εκτελεί τα Unit tests
- package : Συσκευάζει(packages) τον μεταγλωττισμένο κώδικα σε αρχείο τύπου archive
- integration test: Εκτελεί επιπλέον τεστ, τα οποία απαιτούν την συσκευασία(packaging)
- verify : Ελέγχει εάν το πακέτο(package) είναι έγκυρο(valid)
- install : Εγκαθιστά το πακέτο στο τοπικό Maven repository
- deploy : Αναπτύσσει(deploy) το πακέτο σε έναν απομακρυσμένο server ή repository

Όσον αφορά τα Maven plugins τα οποία αναφέρονται στο POM, αυτά αποτελούν μια συλλογή από ένα ή περισσότερους στόχους(goals). Τα goals εκτελούνται σε φάσεις, κάτι το οποίο βοηθά να στο να καθοριστεί η σειρά με την οποία αυτά τα goals θα εκτελεστούν. Τέλος παρουσιάζεται η λειτουργία των repositories στο Maven. Η διαδικασία έχει ως εξής :

- Τα dependencies του project, τα οποία αναφέρονται στο αρχείο παραμετροποίησης pom.xml, διαβάζονται από το Maven
- Τα dependencies τα οποία περιέχονται στο POM αρχείο αναζητούνται στο τοπικό repository
- Εάν τα dependencies δεν βρεθούν στο τοπικό repository, τότε αναζητούνται στο απομακρυσμένο Maven central repository
- Τα dependencies τα οποία βρέθηκαν στο Maven central repository αποθηκεύονται τοπικά στο τοπικό repository
- Τα dependencies τα οποία βρίσκονται τοπικά πλέον, χρησιμοποιούνται για να γίνει build και να εκτελεστεί η εφαρμογή



Local and Central Repository of maven

Υπάρχουν δύο ειδών repositories, 1) το τοπικό(local) repository και 2) το κεντρικό(central) repository. Το local repository εγκαθίσταται στον Η/Υ στον οποίο γίνεται η ανάπτυξη της εφαρμογής. Σε λειτουργικά συστήματα MS Windows το local repository βρίσκεται στην διαδρομή C:\users\...\m2\repository, ενώ σε λειτουργικά συστήματα Mac και Linux βρίσκεται στην διαδρομή ~/.m2/repository (όπου ~ είναι το home directory). Το Maven θα αναζητήσει τα dependencies πρώτα στο local repository πριν στραφεί στο central repository. Με λίγα λόγια το local repository παίζει τον ρόλο της μνήμης cache όσον αφορά τα dependencies μιας εφαρμογής. Το Maven θα αναζητήσει το απομακρυσμένο central repository στην προεπιλεγμένη διεύθυνση <https://repo.maven.apache.org/maven2/>. Όταν τα αρχεία τα οποία αποτελούν τα dependencies κατέβουν από το διαδίκτυο αποθηκεύονται στο local repository. Κατ' αυτόν τον τρόπο όταν το Maven χρειαστεί κάποιο από αυτά τα dependencies ξανά, θα το εντοπίσει αποθηκευμένο στο local repository και έτσι δεν θα χρειαστεί να γίνει ξανά αναζήτηση στο central repository.

8.4 Thymeleaf

Μία μηχανή υποδειγμάτων(template engine) επαναχρησιμοποιεί στατικά στοιχεία ιστοσελίδων ενώ ορίζει δυναμικά στοιχεία τα οποία βασίζονται σε παραμέτρους οι οποίες προέρχονται από web requests. Οι προγραμματιστές μπορούν να υλοποιήσουν templates με την βοήθεια Content management Systems(CMS), Web application frameworks και HTML editors.

Το Thymeleaf είναι μία Java XML/XHTML/HTML5 template engine, η οποία μπορεί να χρησιμοποιηθεί τόσο σε web(βασισμένα σε servlets) όσο και σε μη web περιβάλλοντα. Είναι κατάλληλο για να εξυπηρετεί XHTML/HTML5 σελίδες στο επίπεδο View μιας MVC(Model-View Controller) web εφαρμογής. Επίσης μπορεί να επεξεργαστεί οποιοδήποτε XML αρχείο ακόμα και σε περιβάλλοντα εκτός διαδικτύου. Σε web

εφαρμογές το Thymeleaf στοχεύει να αποτελέσει τον αντικαταστάτη των Java Server Pages(JSP) και υλοποιεί την έννοια των Natural Templates. Template αρχεία τα οποία μπορεί να ανοίξει κανείς απευθείας σε φυλλομετρητές(browsers) και τα οποία λειτουργούν σωστά σαν ιστοσελίδες. Το Thymeleaf αποτελεί λογισμικό ανοιχτού κώδικα, με άδεια χρήσης σύμφωνη με την άδεια Apache License 2.0 . Τα κύρια χαρακτηριστικά του Thymeleaf αναφέρονται παρακάτω :

- Java template engine για XML, XHTML και HTML5
- Χρησιμοποιείται σε web και μη web(offline) περιβάλλοντα. Δεν υπάρχει μεγάλη εξάρτηση με το Servlet Application Programming Interface(API)
- Βασίζεται σε αρθρωτά(modular) σύνολα χαρακτηριστικών τα οποία ονομάζονται διάλεκτοι(dialects)
- Τα χαρακτηριστικά ενός dialect, όπως είναι η αξιολόγηση, η επανάληψη κ.α., εφαρμόζονται συνδέοντάς τα με tags ή attributes ενός template
- Δύο dialects είναι άμεσα διαθέσιμοι : Standard και Spring Standard(για Spring MVC εφαρμογές)
- Οι προγραμματιστές έχουν την δυνατότητα να επεκτείνουν και να δημιουργήσουν δικές τους διαλέκτους(dialects)
- Πολλαπλές template λειτουργίες
- XML
- XHTML
- HTML5 : XML κώδικας και παλαιότερος κώδικας HTML5
- Πλήρης υποστήριξη για διεθνοποίηση(internationalization)
- Παραμετροποιήσιμη, υψηλής απόδοσης template cache, η οποία μειώνει την είσοδο(input)/έξοδο(output) στο ελάχιστο
- Αυτόματες μεταφράσεις DOCTYPE για επαλήθευση τόσο του template όσο και του εξαγόμενου κώδικα
- Πάρα πολύ επεκτάσιμο : μπορεί να χρησιμοποιηθεί σαν template engine framework εάν χρειαστεί
- Πλήρης τεκμηρίωση(documentation) συμπεριλαμβανομένων πολλών εφαρμογών παραδειγμάτων

```

47 <thead class = "table-dark">
48 <th scope = "row">Id</th>
49 <th scope = "row">Name</th>
50 <th scope = "row">Photo</th>
51 <th scope = "row">Quantity</th>
52 <th scope = "row">Price</th>
53 <th scope = "row">Add to Cart</th>
54 </thead>
55 <tbody>
56 <tr th:each="item,iterStart : ${productList}" th:if="${iterStart.count<8}">
57
58 <td th:text = "${item.idProduct}"></td>
59 <td th:text = "${item.productName}"></td>
60 <td  </td>
61
62 <td>
63
64 <input type = "number" name=quantities min="0" max="999" >
65 </td>
66 <td th:text="${item.price}"></td>

```

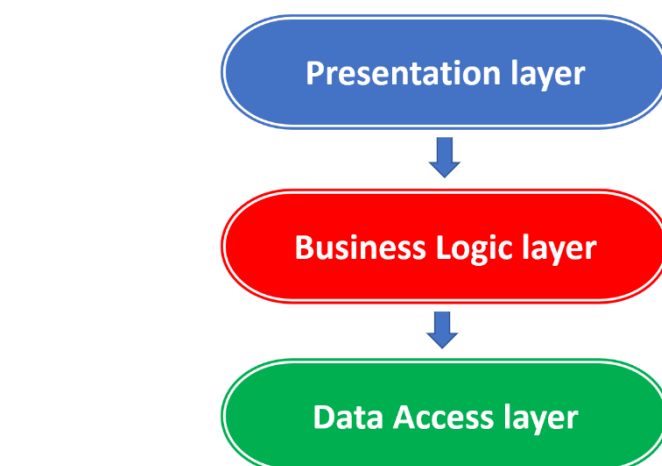
Thymeleaf example

Στην παραπάνω εικόνα η οποία αποτελεί ένα παράδειγμα από την εφαρμογή παρατηρούμε ένα παράδειγμα χρησιμοποιώντας την τεχνολογία του Thymeleaf όπου διατρέχουμε μια λίστα από προϊόντα και εμφανίζουμε σε ένα πίνακα το id του προϊόντος, το όνομα του, την φωτογραφία του και υπάρχει και ένα input πεδίο όπου ο χρήστης εισάγει την ποσότητα του προϊόντος που εκείνος επιθυμεί.

9. Αρχιτεκτονική της εφαρμογής

Το Spring Boot ακολουθεί μια αρχιτεκτονική επιπέδων όπου το κάθε επίπεδο επικοινωνεί άμεσα με το άλλο. Στην παρακάτω εικόνα παρουσιάζεται το 3-επίπεδων (Three-Tier Architecture).

Three-Tier (or Three-Layer) Architecture



Three-Tier Architecture

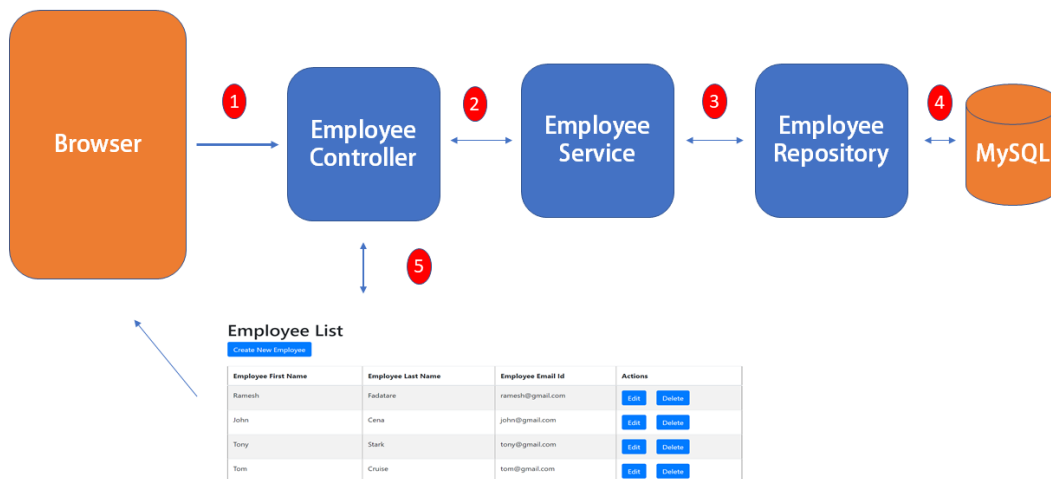
Presentation layer: Αυτό το επίπεδο αποτελεί την αλληλεπίδραση του χρήστη με την εφαρμογή όπου παρουσιάζονται τα δεδομένα και οι λειτουργικότητες που μπορεί να πραγματοποιήσει ο χρήστης στην εφαρμογή.

Business Logic: Σε αυτό το επίπεδο περιέχεται όλη η λειτουργικότητα της εφαρμογής και περιέχει μεθόδους που επεξεργάζονται δεδομένα.

Data Access layer: Το επίπεδο αυτό είναι υπεύθυνο για την αλληλεπίδραση με την βάση δεδομένων και παρέχει λειτουργικότητες όπως διαγραφή, αποθήκευση και ανάκτηση δεδομένων.

Μια web- εφαρμογή στο Spring Boot εφαρμόζει το MVC μοντέλο. Στην παρακάτω εικόνα παρουσιάζεται ένα διάγραμμα για το πως λειτουργεί μια web εφαρμογή με χρήση της τεχνολογίας Thymeleaf στο Spring Boot.

Spring Boot Thymeleaf CRUD Database Real-Time Project



Spring Boot Thymeleaf Hibernate MySQL CRUD Example

Όπως παρατηρούμε στο παραπάνω παράδειγμα υπάρχουν οι λειτουργίες που συμβαίνουν είναι οι εξής:

1. Ο Controller λαμβάνει ένα http αίτημα από τον client
2. Ο Controller επεξεργάζεται το αίτημα και το στέλνει στο Service Layer.
3. Το Service layer είναι υπεύθυνο για όλη την λειτουργικότητα που εξυπηρετεί την εφαρμογή.
4. Το Repository layer είναι υπεύθυνο για την αλληλεπίδραση με την βάση δεδομένων.
5. Ο Controller επιστρέφει την παρουσίαση της εφαρμογής σε Thymeleaf για την απεικόνιση στον περιηγητή.

Η εφαρμογή του Cinema είναι σχεδιασμένη με αυτά τα βήματα που περιέγραψα παραπάνω. Πιο συγκεκριμένα στην αρχή σχεδιάζουμε τις κλάσεις οι οποίες με τα κατάλληλα annotation δημιουργούν τους αντίστοιχους πίνακες στην βάση.

Την διαδικασία της αντιστοίχισης των διάφορων πινάκων και πεδίων στην βάση δεδομένων την αναλαμβάνει το εργαλείο Hibernate όπου ανάλογα με τα σχόλια(annotation) που υπάρχουν στην κάθε κλάση κάνει και την αντίστοιχη λειτουργία στην βάση δεδομένων. Παρακάτω παρουσιάζουμε τα annotation που χρησιμοποιήθηκαν για τον σχεδιασμό της βάση από το Spring Boot:

@Entity: Χαρακτηρίζει μία κλάση σαν οντότητα η οποία αντιστοιχίζεται-απεικονίζεται σε έναν πίνακα της σχεσιακής βάσης δεδομένων.

@Table: Χαρακτηρίζει ένα πίνακα στην σχεσιακή βάση δεδομένων.

@Id: Χαρακτηρίζει μια μεταβλητή μέλος μιας κλάσης σαν κύριο κλειδί(primary key). Έτσι η μεταβλητή αυτή θα αντιστοιχισθεί σε ένα πεδίο ενός πίνακα, το οποίο θα αποτελέσει το κύριο κλειδί του πίνακα αυτού.

@GeneratedValue: (strategy = GenerationType.IDENTITY) : Η συγκεκριμένη στρατηγική η οποία επιλέχθηκε για το κύριο κλειδί είναι αυτή της αυτόματης αύξησης(auto increment). Κάθε φορά που δημιουργείται καινούρια εγγραφή στην βάση δ., η τιμή του πεδίου αυτού αυξάνεται κατά ένα σε σχέση με την προηγούμενη εγγραφή.

@Column: Ορίζει ιδιότητες για κάποια μεταβλητή μέλος μιας κλάσης τις οποίες θα έχει το αντίστοιχο πεδίο πίνακα στην βάση δεδομένων.

@OneToMany : Ορίζει την σχέση ένα προς πολλά που θα δημιουργηθεί στην βάση δεδομένων.

@ManyToMany: Ορίζει την σχέση πολλά προς πολλά που θα δημιουργηθεί στην βάση δεδομένων.

@JoinTable: Ορίζει την συσχέτιση σε μια σχεσιακή βάση δεδομένων χρησιμοποιείται στις σχέσεις ένα προς πολλά και πολλά προς πολλά για να ορίσει ποιος από τους δύο πίνακες είναι ο κυρίαρχος.

@ManyToOne: Ορίζει την σχέση πολλά προς ένα που θα δημιουργηθεί στην βάση δεδομένων

@OneToOne: Ορίζει την σχέση ένα προς ένα που θα δημιουργηθεί στην βάση δεδομένων

9.1 Class @Entity

Οι κλάσεις της εφαρμογής οι οποίες ξεκινούν με το annotation *@Entity* αποτελούν της βασικές κλάσεις της εφαρμογής μας και οι οποίες σχηματίζουν και τους αντίστοιχους πίνακες στην εφαρμογή.

Οι κλάσεις δηλωμένες με το *@Entity* είναι οι εξής:

- CartItem
- Genre
- Movie
- Order
- OrderProduct
- Product
- Seat
- ShoppingCart
- ShowTime
- Ticket
- User
- UserType

9.2 Interface @Repository

Για την αλληλεπίδραση μας με τον κάθε πίνακα που έχει δημιουργηθεί στην βάση δεδομένων χρειαζόμαστε την δημιουργία interface τα οποία και ξεκινούν με το annotation `@Repository` και κληρονομούν από το `JpaRepository` το οποίο διαθέτει λειτουργικότητες δημιουργίας, αναβάθμισης και διαγραφής δεδομένων από την βάση.

Οι κλάσεις δηλωμένες με το `@Repository` είναι οι εξής:

- `CartItemRepository`
- `GenreRepository`
- `MovieRepository`
- `OrderRepository`
- `OrderProductRepository`
- `ProductRepository`
- `SeatRepository`
- `ShoppingCartRepository`
- `ShowTimeRepository`
- `TicketRepository`
- `UserRepository`
- `UserTypeRepository`

9.3 Class @Service

Σε αυτό το επίπεδο στο οποίο είναι δομημένη όλη η business λογική της εφαρμογής έχουν δημιουργηθεί οι αντίστοιχες κλάσεις για κάθε οντότητα που σχεδιάσαμε για να δημιουργηθούν οι πίνακες στην βάση δεδομένων. Συνεπώς έχουμε κλάσεις οι οποίες ξεκινάνε με το annotation `@Service` και περιέχουν μεθόδους οι οποίες λειτουργούν σαν διαμεσολαβητές ανάμεσα στα επίπεδα της πρόσβασης δεδομένων στην βάση και στο επίπεδο παρουσίασης της εφαρμογής. Σε αυτό το σημείο θέλω να επισημάνω ότι χρησιμοποιείται η τεχνική του Dependency Injection με το annotation `@Autowired` πάνω από την μεταβλητή του `Repository`. Με αυτόν τον τρόπο δημιουργούμε ένα αντικείμενο του `Repository` και έτσι έχουν πρόσβαση στις μεθόδους του.

Οι κλάσεις δηλωμένες με το `@Service` είναι οι εξής:

- `CartItemService`
- `GenreService`
- `MovieService`
- `OrderService`
- `OrderProductService`
- `ProductService`
- `SeatService`
- `ShoppingCartService`
- `ShowTimeService`
- `TicketService`
- `UserService`
- `UserTypeService`

9.4 Class @Controller

Το επίπεδο αυτό είναι υπεύθυνο για την παρουσίαση της διαδικτυακής εφαρμογής και αναλαμβάνει να δημιουργεί σελίδες html στις οποίες περιέχεται οι τεχνολογία *Thymeleaf*. Οι κλάσεις σε αυτό το επίπεδο ξεκινάνε με το annotation *@Controller*. Σε αυτές τις κλάσεις δηλώνονται μεταβλητές τύπου κλάσεων *Service* η οποίες δηλώνονται ως *@Autowired* και έτσι παράγονται αντικείμενα τύπου *Service*. Με αυτό τον τρόπο έχουμε πρόσβαση στις μεθόδους του επιπέδου *Service* και μπορούμε να πραγματοποιήσουμε διάφορες λειτουργικότητες.

Οι κλάσεις δηλωμένες με το *@Controller* είναι οι εξής:

- ConfirmOrdersController
- ConfirmTicketsController
- HomeController
- LoginController
- MoviesController
- ProductsController

9.5 MySql Database

Η σχεσιακή βάση δεδομένων που χρησιμοποιήθηκε είναι η MySql η οποία συμπεριλαμβάνει τους εξής πίνακες:

User

| | |
|------------------|-------------|
| Int(Primary Key) | Id_user |
| Varchar | Name |
| Varchar | Password |
| Varchar | Username |
| Int(Foreign Key) | Id_usertype |

User-Type

| | |
|------------------|-------------|
| Int(Primary Key) | Id_usertype |
| Varchar | Type |

Η σχέση μεταξύ του πίνακα User και User-Type είναι 1:1.

Movie

| | |
|------------------|----------------|
| Int(Primary Key) | Id_movie |
| Long Text | Actors |
| Long Text | Description |
| Long Text | Director |
| Varchar | Image |
| Int | Movie_duration |
| Double | Price |
| Varchar | Rating |
| Varchar | Title |

Genre-Movie

| | |
|-------------|----------|
| Foreign Key | Id_genre |
| Foreign Key | Id_movie |

Genre

| | |
|------------------|----------|
| Int(Primary Key) | Id_genre |
| Varchar | name |

Η σχέση μεταξύ του *Movie* και του *Genre* είναι $N:N$, και δημιουργείται ένας ενδιάμεσος πίνακας που έχει δύο *Foreign Key* που αντιστοιχούν στα *Primary Key* του πίνακα *Movie* και του πίνακα *Genre*.

Show-Time

| | |
|------------------|-------------|
| Int(Primary Key) | Id_showtime |
| DateTime | start_movie |
| Int(Foreign Key) | Id_movie |

Η σχέση μεταξύ του *Movie* και του *Show_Time* είναι $1:N$, δηλαδή μία ταινία μπορεί να έχει πολλές προβολές αλλά είμαι προβολή σχετίζεται με μια ταινία.

Ticket

| | |
|------------------|-------------|
| Int(Primary Key) | Id_ticket |
| Varchar | pay |
| Double | ticket_cost |
| Int(Foreign Key) | Id_seat |
| Int(Foreign Key) | Id_showtime |
| Int(Foreign Key) | Id_user |

Seat

| | |
|------------------|-------------|
| Int(Primary Key) | Id_seat |
| Int | number_seat |

Η σχέση μεταξύ του πίνακα *Show_Time* και του πίνακα *Ticket* είναι $1:N$, δηλαδή μια συγκεκριμένη προβολή μπορεί να έχει πολλά εισιτήρια αλλά ένα συγκεκριμένο εισιτήριο σχετίζεται με μια προβολή. Επιπλέον ένα εισιτήριο αντιστοιχεί για την κάθε θέση. Στο σημείο αυτό να διευκρινίσουμε ότι κάθε εισιτήριο σχετίζεται μοναδικά με την θέση και την ώρα προβολής όπως βλέπουμε και στην παρακάτω εικόνα. Τέλος, η σχέση μεταξύ του πίνακα *User* και του πίνακα *Ticket* είναι $1:N$ δηλαδή κάθε χρήστης μπορεί να αγοράσει πολλά εισιτήρια αλλά κάθε εισιτήριο αντιστοιχεί σε έναν χρήστη.

```

1 package gr.mariakapa.cinema.Entity;
2
3 import ...
4
5
6
7 @Entity
8 @Table(name = "ticket",
9         uniqueConstraints = { @UniqueConstraint(columnNames =
10             { "id_showtime", "id_seat" }) })
11 public class Ticket {
12
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     @Column(name = "id_ticket", nullable = false)
16     private int idTickets;
17

```

Unique Constrains (Id_showtime, Id_seat)

Order

| | |
|------------------|--------------|
| Int(Primary Key) | Id_order |
| Varchar | Order_status |
| Double | total_cost |
| Int(Foreign Key) | Id_user |

Order_Product

| | |
|------------------|------------------|
| Int(Primary Key) | Id_order_product |
| Double | Cost |
| Int | Quantity |
| Int(Foreign Key) | Id_order |
| Int(Foreign Key) | Id_product |

Product

| | |
|------------------|--------------|
| Int(Primary Key) | Id_product |
| Varchar | Image |
| Double | Price |
| Varchar | Product_name |

Η σχέση μεταξύ του πίνακα *Order* και του πίνακα *User* είναι *N:1*, δηλαδή μια παραγγελία συνδέεται με έναν χρήστη αλλά ένας χρήστης μπορεί να πραγματοποιήσει πολλές παραγγελίες. Επίσης, η σχέση του πίνακα *Order* με τον πίνακα *Product* είναι *N:N* από αυτή την σχέση προκύπτει και ο ενδιάμεσος πίνακας.

Cart_Item

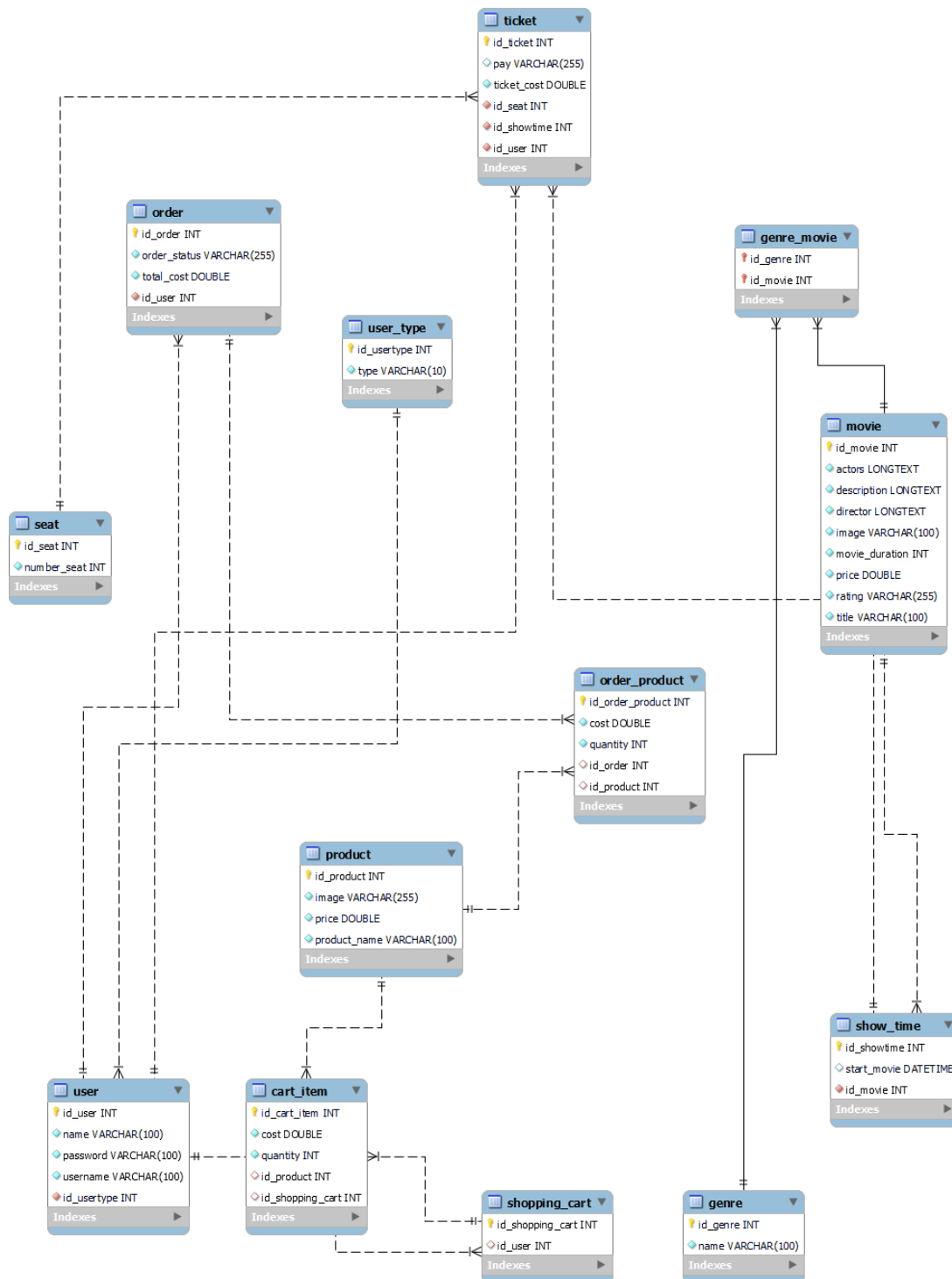
| | |
|------------------|------------------|
| Int(Primary Key) | Id_cart_Item |
| Double | Cost |
| Int | Quantity |
| Int(Foreign Key) | Id_shopping_cart |

| | |
|------------------|------------|
| Int(Foreign Key) | Id_product |
|------------------|------------|

Shopping_Cart

| | |
|------------------|------------------|
| Int(Primary Key) | Id_shopping_cart |
| Int(Foreign Key) | Id_user |

Οι παραπάνω πίνακες έχουν σχεδιαστεί για την δημιουργία ενός καλαθιού αγορών του χρήστη όπου κάθε χρήστης έχει ένα καλάθι αγορών το οποίο αδειάζει και γεμίζει προϊόντα. Ο πίνακας *Cart_item* έχει σχεδιαστεί για να αποτυπώσει την σχέση *N:N* που έχουν οι πίνακες *Shopping_Cart* και *Product* αλλά και ότι ένα προϊόν μέσα στο καλάθι αγορών έχει μοναδική ποσότητα και κόστος.



Erd_Diagram_Complex_Cinema

Τέλος όλες οι απαραίτητες ιδιότητες για την σύνδεση της εφαρμογής μας με την βάση δεδομένων MySql ορίζονται στο αρχείο application.properties του project. Αυτές είναι το URL(Uniform Resource Locator)

για την σύνδεση με την βάση δεδομένων., το όνομα χρήστη(username) και ο κωδικός(password), ο οδηγός(driver) ο οποίος χρησιμοποιείται για την σύνδεση με την συγκεκριμένη βάση δεδομένων. Όλες αυτές οι ρυθμίσεις δίνονται παρακάτω :

```
spring.datasource.url=jdbc:mysql://$MYSQ_HOST:localhost:3306/cinema?useSSL=false&serverTimezone=Europe/At
spring.datasource.username= root
spring.datasource.password= root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

application.properties

Επίσης στο αρχείο application.properties ορίζεται και η πόρτα την οποία χρησιμοποιεί ο τοπικός διακομιστής διαδικτύου(Localhost Web Server) , στην συγκεκριμένη εφαρμογή χρησιμοποιείται ο ήδη υπάρχων τοπικός διακομιστής διαδικτύου που έχει ρυθμίσει το Spring Boot που είναι προσβάσιμος <http://localhost:8080 /login> , προσθέτοντας το path /login.

```
server.tomcat.context-path=/login
```

application.properties

10. Συμπεράσματα

Η ιδέα για την ανάπτυξη της εφαρμογής προήλθε έπειτα από μια σχετική έρευνα στο αν υπάρχουν αντίστοιχες πλατφόρμες στους κινηματογράφους της Ελλάδας και παρατηρήθηκε ότι οι κινηματογράφοι που διαθέτουν αντίστοιχες πλατφόρμες δεν διαθέτουν μια ολοκληρωμένη ψηφιακή πρόταση για την ηλεκτρονική διαχείριση του πολυχώρου κινηματογραφικών αιθουσών ενός σινεμά.

Η παρούσα διατριβή λοιπόν αναπτύχθηκε σε αυτό το πλαίσιο με στόχο να δώσει μια πιο ολοκληρωμένη πλατφόρμα που να καλύπτει την ηλεκτρονική διαχείριση διάφορων ενεργειών και λειτουργιών ενός κινηματογράφου τόσο από την πλευρά του θεατή όσο και από την πλευρά του εργαζόμενου του σινεμά.

Η εφαρμογή αυτή σχεδιάστηκε με στόχο να εξυπηρετήσει αυτή την ανάγκη και μέσα από την χρήση σύγχρονων τεχνολογιών είναι εύκολα συντηρήσιμη και προσφέρει ένα φιλικό και εύχρηστο περιβάλλον στο χρήστη .

11. Βιβλιογραφία

1. Java The Complete Reference Eleventh Edition, Herbert Schildt
2. <https://www.baeldung.com/>
3. <https://www.w3schools.com/java/>
4. <https://stackoverflow.com/>
5. <https://www.thymeleaf.org/>
6. https://www.youtube.com/watch?v=9SGDpanrc8U&ab_channel=Amigoscode