ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΣΧΟΛΗ: ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ: ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΛΟΓΙΣΜΙΚΟΥ ΜΕ ΤΕΧΝΟΛΟΓΙΕΣ JAVA EE

Δημήτριος Κυριάκος Γεράσιμος Τσιμάρας

Επιβλέπων: Μάριος Μάντακας
Αναπληρωτής Καθηγητής

Άρτα, Ιούνιος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΣΧΟΛΗ: ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ: ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΛΟΓΙΣΜΙΚΟΥ ΜΕ ΤΕΧΝΟΛΟΓΙΕΣ JAVA EE

Δημήτριος Κυριάκος Γεράσιμος Τσιμάρας

Επιβλέπων: Μάριος Μάντακας
Αναπληρωτής Καθηγητής
Άρτα, Ιούνιος 2022

SOFTWARE APPLICATION DEVELOPMENT WITH JAVA EE TECHNOLOGIES

Εγκρίθηκε από τριμελή εξεταστική επιτροπή
Άρτα, 30/6/2022

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Μάριος Μάντακας
   αναπληρωτής καθηγητής

2. Μέλος επιτροπής

   Γιαννακέας Νικόλαος
   επίκουρος καθηγητής

3. Μέλος επιτροπής

   Γκόγκος Χρήστος
   αναπληρωτής καθηγητής

Ο Προϊστάμενος του Τμήματος

Ευριπίδης Γλαβάς
καθηγητής

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Τσιμάρας, Δημήτριος

Υπογραφή

Περίληψη

Η εργασία αφορά την ανάπτυξη μιας εφαρμογής ιστού με βάση δεδομένων, με στόχο την μελέτη τεχνολογιών Java EE και ειδικότερα JSF/PrimeFaces, CDI, EJB, και JPA, αυθεντικοποίηση και εξουσιοδότηση. Η εφαρμογή είναι τμήμα ενός συστήματος διαχείρισης βιβλιοθήκης που επιλέχτηκε με σκοπό την επίδειξη των εννοιών, χωρίς απαίτηση πληρότητας.

Η εργασία περιλαμβάνει το παρόν κείμενο και τον κώδικα της εφαρμογής, διαθέσιμο στο URL https://github.com/dtsimaras/MyLMS

Η εργασία στηρίζεται στο μάθημα Σχεδιασμός Πληροφοριακών Συστημάτων και προεκτείνει την μελέτη σε θέματα που δεν καλύφθηκαν στο Μάθημα αυτό.

Το Μέρος 1 του παρόντος κειμένου συνοψίζει την μελέτη των τεχνολογιών Java EE που δεν καλύφθηκαν στο προαναφερθέν Μάθημα: PrimeFaces και Ajax, JSF localization (διεθνοποίηση – δυνατότητα πολυγλωσσίας) και JSF templating (πρότυπα σελίδων), αυθεντικοποίηση και εξουσιοδότηση με GlassFish Server.

Το Μέρος 2 παρουσιάζει την ανάπτυξη της εφαρμογής και πιο συγκεκριμένα απαιτήσεις λογισμικού, ανάλυση και σχεδιασμό, και υλοποίηση για την βάση δεδομένων σε Oracle XE και την κυρίως εφαρμογή σε Java.

Λέξεις-κλειδιά: Java EE, JSF, PrimeFaces, Ajax, αυθεντικοποίηση, διαχείριση βιβλιοθήκης, εφαρμογή ιστού και βάσης δεδομένων

Abstract

The thesis is about the development of a web application with a database, which aims at the study of Java EE technologies and more specifically JSF/PrimeFaces, CDI, EJB and JPA authentication and authorization. The application is part of a library management system which has been selected in order to display the meanings without having to be complete.

The thesis includes this text as well as the application code, available at

URL: https://github.com/dtsimaras/MyLMS

The thesis is based on the course on Designs of Enterprise Systems and has extended its study on issues which had not been implemented during the course.

Part 1 of the current text consists of the summary of the study of Java EE technologies which had not been included in the above course: PrimeFaces and Ajax, JSF localization (internationalization - multilingual ability) and JSF templating (page templates), authentication and authorization through GlassFish Server.

Part 2 presents the development of the application and more specifically software requirements, designing and analysis as well as the implementation of the database on Oracle XE and its main application on Java.

Key words : Java EE, JSF, PrimeFaces, Ajax, authentication, library management, web application and data base.

Περιεχόμενα

Περιεχόμενα Εικόνων

Πρόλογος

Η πτυχιακή εργασία αφορά την ανάπτυξη μιας εφαρμογής ιστού με βάση δεδομένων, με στόχο την μελέτη τεχνολογιών Java EE και ειδικότερα JSF/PrimeFaces, CDI, EJB, και JPA, αυθεντικοποίηση και εξουσιοδότηση. Η εφαρμογή είναι τμήμα συστήματος διαχείρισης βιβλιοθήκης που επιλέχτηκε με σκοπό την επίδειξη των εννοιών, χωρίς απαίτηση πληρότητας.

Η εργασία περιλαμβάνει το παρόν κείμενο και τον κώδικα της εφαρμογής, διαθέσιμο στο URL https://github.com/dtsimaras/MyLMS

Η εργασία στηρίζεται στο μ άθημα Σχεδιασμός Πληροφοριακών Συστημάτων και προεκτείνει την μελέτη σε θέματα που δεν καλύφθηκαν στο Μάθημα αυτό.

Το Μέρος 1 του παρόντος κειμένου συνοψίζει την μελέτη των τεχνολογιών Java EE που δεν καλύφθηκαν στο προαναφερθέν Μάθημα: PrimeFaces και Ajax, JSF localization (διεθνοποίηση – δυνατότητα πολυγλωσσίας) και JSF templating (πρότυπα σελίδων), αυθεντικοποίηση και εξουσιοδότηση με GlassFish Server. Προαπαιτούμενες είναι τεχνολογίες Java EE, και ειδικότερα, JSF, CDI, EJB, JPA, που δεν παρουσιάζονται αναλυτικά.

Το Μέρος 2 παρουσιάζει την ανάπτυξη της εφαρμογής και πιο συγκεκριμένα απαιτήσεις λογισμικού, ανάλυση και σχεδιασμό, και υλοποίηση για την βάση δεδομένων σε Oracle XE και την κυρίως εφαρμογή σε Java.

Μέρος 1 - Έννοιες, τεχνολογίες, τεχνικές

Το Μέρος 1 του παρόντος κειμένου συνοψίζει την μελέτη των τεχνολογιών Java EE που δεν καλύφθηκαν στο προαναφερθέν Μάθημα: PrimeFaces και Ajax, JSF localization (διεθνοποίηση – δυνατότητα πολυγλωσσίας) και templating, αυθεντικοποίηση και εξουσιοδότηση με GlassFish Server.

Java Server Faces

Η τεχνολογία Java Server Faces (JSF) είναι ένα πλαίσιο διεπαφής χρήστη (user interface framework) για δημιουργία εφαρμογών ιστού (Oracle 2014, Τμήμα 1.7.3). Η JSF περιλαμβάνει:

- Ένα πλαίσιο συνιστωσών (components) γραφικής διεπαφής χρήστη (GUI)
- Ένα μοντέλο για την απόδοση των συνιστωσών σε HTML ή άλλες τεχνολογίες. Ένας renderer μετατρέπει τα δεδομένα του μοντέλου σε τύπους διεπαφών και δημιουργεί το markup των διεπαφών HTML.

Τα παρακάτω χαρακτηριστικά υποστηρίζουν την γραφική διεπαφή χρήστη:

- Επικύρωση εισόδου
- Χειρισμός γεγονότων
- Μετατροπές δεδομένων μεταξύ των αντικειμένων του μοντέλου και των γραφικών συνιστωσών
- Παραμετροποίηση της πλοήγησης σε σελίδες
- Διαχειριζόμενη δημιουργία αντικειμένων του μοντέλου
- Expression Language (EL)

JavaServer Faces & MVC

Στην JSF (JavaServer Faces) χρησιμοποιείται το πρότυπο σχεδιασμού MVC (Model-View-Controller). Το MVC είναι ένα πρότυπο αρχιτεκτονικής λογισμικού που διαχωρίζει τα μέρη της εφαρμογής σε:

- Model: το οποίο μέρος διαχειρίζεται τη λογική των δεδομένων.
- View: είναι το κομμάτι της προβολής, και παρουσιάζει τα δεδομένα στους χρήστες
- Controller: ελέγχει τη ροή των δεδομένων και τη λογική της εφαρμογής και ενημερώνει το View.

Είναι δύσκολο να διευκρινίσουμε ποιο είναι ποιο κομμάτι όμως στη JSF, διότι εξαρτάται από την οπτική (Scholtz 2018, σελ. 275).

Από την οπτική του JSF framework

- Το model είναι το backing bean
- Το view είναι το δέντρο συνιστωσών (αρχεία Facelets)
- Και ο controller είναι το FacesServlet

Από την οπτική του διακομιστή εφαρμογής της Java EE

- Το model αναπαρίσταται από το επίπεδο υπηρεσιών (service layer), συνήθως κλάσεις EJB και JPA entities
- Το view αναπαρίσταται από τον JSF κώδικα

- Και ο controller είναι το FacesServlet

Από την οπτική του JSF προγραμματιστή

- Το model αναπαρίσταται από το επίπεδο υπηρεσιών
- Το view από τα αρχεία Facelets
- Και ο controller αναπαρίσταται από τα backing bean.

Το backing bean δηλαδή μπορεί να είναι είτε model, είτε view, είτε controller ανάλογα την οπτική, ενώ το επίπεδο υπηρεσιών είναι πάντα το model και οι σελίδες Facelets είναι πάντα το view.

Σε αυτό το κείμενο, ο *προγραμματιστής* είμαι εγώ, που αναπτύσσω μια εφαρμογή ιστού χρησιμοποιώντας το *JSF framework* για ένα *διακομιστή εφαρμογής Java EE*.

Ajax

Ajax (Asynchronous JavaScript and XML) είναι μία τεχνολογία πελάτη-περιηγητή που επιτρέπει σε μια σελίδα HTML περιηγητή, μετά τη φόρτωση της από το διακομιστή

- την αποστολή δεδομένων στο διακομιστή και την εκτέλεση κώδικα στο διακομιστή
- τη λήψη δεδομένων από το διακομιστή
- την (ενημέρωση της σελίδας (δυνατότητα μερικής ενημέρωσης επιλεγμένων γραφικών συνιστωσών) χωρίς επαναφόρτωση της σελίδας.

Στηρίζεται σε εκτέλεση κώδικα JavaScript της σελίδας ο οποίος επικοινωνεί με τον server (W3 schools, JS Ajax). Η ενημέρωση της σελίδας γίνεται χωρίς το οπτικό αποτέλεσμα της επαναφόρτωσης (Geary, Horstman, 2010, σελ. 21).

Τυπικές περιπτώσεις χρήσης Ajax είναι η επικύρωση τιμών (validation) πεδίων κειμένου, και η εμφάνιση ενδείξεων προόδου (progress indicators) (Geary, Horstman, 2010, Κεφάλαιο 10).

Ajax στην JSF

Η JSF υποστηρίζει τη χρήση Ajax, χρησιμοποιώντας μία πρότυπη βιβλιοθήκη JavaScript. Επίσης, οι επεκτάσεις της JSF, όπως PrimeFaces χρησιμοποιούν Ajax.

Λειτουργικότητα Ajax μπορεί να προστεθεί στην JSF με τους εξής τρόπους (Oracle, 2014, Chapter 13):

- Με προσθήκη κώδικα JavaScript στην δήλωση των JSF views.
- Με χρήση της βιβλιοθήκης πόρων Ajax της JSF.
    - Οι συνιστώσες γραφικής διεπαφής UI της JSF, όπως πεδία κειμένου, κουμπιά, labels, μπορούν να υποστηρίξουν υπηρεσίες Ajax βάσει δηλώσεων στα αρχεία xml των JSF views. Αυτός είναι ο συνηθέστερος τρόπος χρήσης Ajax στην JSF.
    - Επίσης, ο κώδικας Java των managed beans υποστήριξης των JSF views μπορεί να καλεί τις μεθόδους της βιβλιοθήκης.

Για να χρησιμοποιήσουμε Ajax σε δήλωση JSF view:

1. Συσχετίζουμε μια συνιστώσα και ένα γεγονός με ένα αίτημα Ajax
2. Επιλέγουμε συνιστώσες που θα εκτελεστούν στον διακομιστή μετά το αίτημα Ajax
3. Επιλέγουμε συνιστώσες που θα αποδοθούν μετά το αίτημα Ajax,

όπως στο παρακάτω παράδειγμα.

Μια συνιστώσα αποκτά δυνατότητα υποστήριξης Ajax αν έχει παιδί ή γίνει παιδί ενός tag `f:ajax`.

JSF Ajax tag attributes	
event	Το γεγονός που ενεργοποιεί το αίτημα Ajax. Μπορεί να είναι • γεγονός JavaScript χωρίς το πρόθεμα on (π.χ. onblur/blur, onkeyup/keyup) • γεγονότα action (για commands) και valueChange (για inputs).
execute	Λίστα συνιστωσών που ο διακομιστής εκτελεί κατά την κλήση Ajax. Ειδικές τιμές: @all @form @none. Default τιμή @this (παρούσα συνιστώσα).
render	Λίστα συνιστωσών που ο διακομιστής αποδίδει στην τελευταία φάση της κλήσης Ajax. Ίδιες ειδικές τιμές όπως για το execute. Default τιμή @none.

Πίνακας - Μερικά attributes του JSF tag (προσαρμοσμένα από το Geary, Horstman, 2010, Table 10-1).

Παράδειγμα 1

```
<h:inputText id="name" value="#{user.name}">
      <f:ajax event="keyup" execute="@this" render="echo"/>
</h:inputText>
...
<h:outputText id="echo" value="#{user.name}"/>
```

Κώδικας από Geary, Horstman, 2010, σελ. 389.

Στο παραπάνω κομμάτι κώδικα προστίθεται μια συμπεριφορά Ajax στη συνιστώσα `inputText`.

Κάθε φορά που ο χρήστης αφήνει ένα πλήκτρο (`keyup event`) στη συνιστώσα `inputText`, η JSF στέλνει ένα αίτημα Ajax στον διακομιστή. Στο διακομιστή η Ajax εκτελεί (`execute`) τη συνιστώσα `name` και όταν επιστραφεί το αίτημα Ajax η JSF αποδίδει (`render`) μόνο τη συνιστώσα `echo` στον πελάτη (Εικόνα 1).

Στο χαρακτηριστικό `execute` το `@this` αναφέρεται στη συνιστώσα που περικλείει την `f:ajax` σε αυτή την περίπτωση στο `inputText` με `id name`. Η default τιμή του `execute` είναι `@this` οπότε μπορούμε να το παραλείψουμε και εντελώς.

Εικόνα 1. Παράδειγμα Ajax keyup event (Geary, Horstman, 2010, Figure 10-4)

Παράδειγμα 2

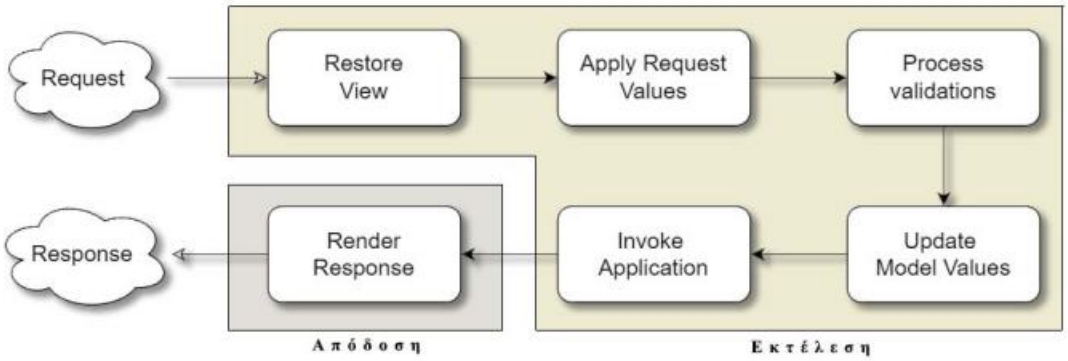
```
...
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:f="http://xmlns.jcp.org/jsf/core">
      ...
<h:form>
      <h:inputText id="nameInput" value="#{user.name}">
            <f:ajax event="blur"
                    execute="@this passwordInput"
                    render="nameError passwordError refreshThisToo"/>
      </h:inputText>
      <h:outputText id="nameError"/>
      ...
      <h:inputText id="passwordInput"/>
      <h:outputText id="passwordError" value="#{user.passwordError}"/>
<h:form>
<h:panelGroup id="refreshThisToo">
       ...
 </h:panelGroup>
```

Παράδειγμα χρήσης Ajax σε JSF view (προσαρμοσμένο από Geary, Horstman, 2010, σελ.389 και How to refresh an element outside form with h:command ajax in jsf2?).

Κύκλος Ζωής JSF και Ajax

Η JSF χωρίζει τον κύκλο ζωής σε 2 τμήματα, την εκτέλεση και την απόδοση (Εικόνα 2, προσαρμοσμένο από Geary & Horstman, Figure 10-2 και Figure 10-3, σελ 387).

Σε κάθε αίτημα Ajax ορίζονται οι συνιστώσες που ο διακομιστής θα εκτελέσει και που θα αποδώσει.



Εικόνα 2. Κύκλος Ζωής JSF

Όταν η JSF εκτελεί μια συνιστώσα στο διακομιστή:

- Μετατρέπει και επικυρώνει την τιμή της συνιστώσας (αν είναι είσοδος)
- Μεταφέρει τις επικυρωμένες τιμές στο μοντέλο (αν είναι συνδεδεμένο με κάποια ιδιότητα)
- Εκτελεί ενέργειες και action listeners

Άρα η JSF ουσιαστικά έχει δύο κύκλους ζωής: έναν που εκτελεί συνιστώσες, και έναν που αποδίδει συνιστώσες. Η JSF πάντα, πρώτα εκτελεί και μετά αποδίδει συνιστώσες.

Σε κανονικά HTTP αιτήματα, όλες οι συνιστώσες σε μια φόρμα και εκτελούνται και αποδίδονται, ενώ σε αιτήματα Ajax, η JSF εκτελεί μία ή περισσότερες συνιστώσες, και αποδίδει καμία ή περισσότερες συνιστώσες.

PrimeFaces

Η PrimeFaces προσφέρει τις ακόλουθες λειτουργίες που την καθιστούν μια ισχυρή βιβλιοθήκη γραφικών συνιστωσών, η οποία παραμένει και εύκολη στη χρήση (K. Siva, P. Reddy 2013, Κεφ. 1):

- Πάνω από 100 πλούσιες συνιστώσες γραφικών
- Ενσωματωμένη υποστήριξη Ajax
- Δε χρειάζεται καθόλου ρυθμίσεις
- Δεν εξαρτάται από βιβλιοθήκες τρίτων για τις περισσότερες συνιστώσες
- Ενσωματωμένο με το ThemeRoller
- 30+ έτοιμα διαθέσιμα θέματα
- Υποστηρίζει τους περιηγητές IE8+, Chrome, Firefox, Safari και Opera

Έγινε η επιλογή της PrimeFaces σε αυτό το έργο κυρίως για τις επιπλέον λειτουργίες αναζήτησης, ταξινόμησης και lazy loading που προσφέρει η συνιστώσα p:dataTable. Ύστερα από εξερεύνηση των συνιστωσών, χρησιμοποιήθηκαν και συνιστώσες που δεν υπάρχουν στη JSF καθώς και κάποιες οι οποίες χρησιμοποιήθηκαν απλά για τον έτοιμο μοντέρνο σχεδιασμό τους όπως τα κουμπιά.

Για να προστεθεί η PrimeFaces (έκδοση 10.0.0) σε ένα maven project πρέπει να προστεθεί η παρακάτω εξάρτηση στο αρχείο pom του project:

```
<dependency>
      <groupId>org.primefaces</groupId>
      <artifactId>primefaces</artifactId>
      <version>10.0.0</version>
</dependency>
```

PrimeFaces Ajax

Η υποστήριξη Ajax έχει γίνει από τις σημαντικότερες τεχνολογίες για κάθε framework ή βιβλιοθήκη που χρησιμοποιείται για την δημιουργία πλούσιων και διαδραστικών εφαρμογών ιστού. Η PrimeFaces έχει ενσωματωμένη υποστήριξη Ajax και βασίζεται στα API's της JSF από τη μεριά του διακομιστή, και από τη μεριά του πελάτη χρησιμοποιεί την JavaScript βιβλιοθήκη jQuery (K. Siva, P. Reddy 2013, σελ. 25).

Στην PrimeFaces χρησιμοποιείται η συνιστώσα `p:ajax` αντί `f:ajax` και τα χαρακτηριστικά `execute` και `render`, ονομάζονται `process` και `update`.

Επίσης η PrimeFaces έχει προσθέσει υποστήριξη Ajax σε πολλές συνιστώσες τις, οπότε μπορεί να παραληφθεί και η χρήση της συνιστώσας `p:ajax` όπως στο παρακάτω παράδειγμα.

Παράδειγμα 1

```
...
  <h:inputText value="#{userBean.user}/>
  <p:commandButton value="Submit" update="label"/>
  <h:outputLabel id="label" value="#{userBean.user}"/>
...
```

Σε αυτήν την περίπτωση η συνιστώσα label θα ενημερωθεί μέσω Ajax χωρίς να γίνει απόδοση ολόκληρης της σελίδας. Στη συνιστώσα `commandButton` της PrimeFaces αντί να χρησιμοποιηθεί μια εμφωλευμένη συνιστώσα `p:ajax` χρησιμοποιείται το χαρακτηριστικό `update`. Η δυνατότητα Ajax του κουμπιού ορίζεται από το χαρακτηριστικό `ajax` το οποίο έχει default τιμή true οπότε μπορεί να παραληφθεί (PrimeFaces 10.0.0, CommandButton).

Επικύρωση δεδομένων - Validation

Η επικύρωση δεδομένων (validation) που εισάγει ο χρήστης είναι ένα κοινό και κρίσιμο κομμάτι κάθε εφαρμογής ιστού. Η JSF υποστηρίζει την επικύρωση γραφικών συνιστωσών και η PrimeFaces την ενισχύει με επιπρόσθετες λειτουργίες.

Ένα παράδειγμα επικύρωσης δεδομένων με JSF είναι το παρακάτω:

Παράδειγμα 1

```
<h:inputText id="name" value="#{userBean.name}"
      required=true
      requiredMessage=”Name is Required”/>
<h:message for=”name” styleClass=”errorMessage”>
```

Όταν υποβληθεί η φόρμα και τρέξει ο παραπάνω κώδικας θα ελεγχθεί αν είναι κενό το `inputText`, και αν είναι, θα αποδοθεί η ίδια σελίδα με το αντίστοιχο μήνυμα “Name is Required”.

Το ίδιο παράδειγμα με PrimeFaces και χρήση Ajax ώστε να εμφανίζεται το μήνυμα χωρίς να χρειάζεται να υποβάλλουμε τη φόρμα φαίνεται παρακάτω:

Παράδειγμα 2

```
<p:inputText id="name" value="#{userBean.name}"
      required="true"
      requiredMessage="Name is required">
      <p:ajax event="keyup" update="nameMsg"/>
</p:inputText>
<p:message id="nameMsg" for="name"/>
```

Όταν ο χρήστης πληκτρολογεί στο `inputText` αν σβήσει τελείως το πεδίο θα εμφανιστεί το μήνυμα “Name is required” μέσω Ajax.

Ajax Listeners

Η PrimeFaces προσφέρει επίσης υποστήριξη για Ajax listeners ώστε να καλεί μεθόδους από κάποιο managed bean. Μπορεί να χρησιμοποιηθεί αυτή δυνατότητα για να γίνει ένας ασύγχρονος έλεγχος ενός πεδίου μέσω μιας μεθόδου στο managed bean και να ενημερωθεί το μήνυμα στην σελίδα μέσω Ajax, χωρίς να χρειάζεται η υποβολή της φόρμας (K. Siva, P. Reddy 2013, σελ. 27).

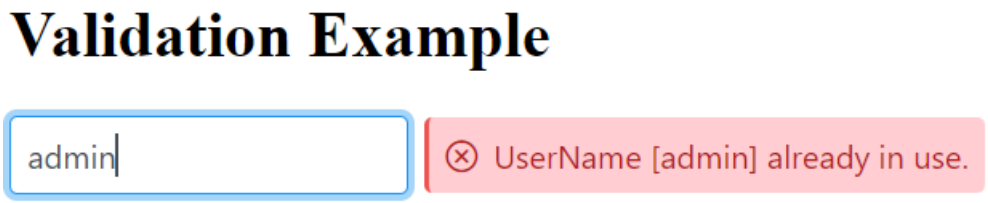
Αρχείο Facelets

```
...
  <p:inputText id="nameText" value="#{userBean.name}">
      <p:ajax event="keyup"
        listener="#{userBean.checkNameExists()}"
        update="nameMsg"/>
  </p:inputText>
  <p:message id="nameMsg" for="nameText"/>
...
```

Μέθοδος στο Managed Bean

```
public void checkNameExists()
{

  if("admin".equals(name))
  {
      String msg = "UserName ["+name+"] already in use.";
      FacesContext.getCurrentInstance()
            .addMessage("nameText",
            new FacesMessage(FacesMessage.SEVERITY_ERROR, msg,
            msg));
  }
}
```

Στον παραπάνω κώδικα υπάρχει ένα listener με ajax συμπεριφορά στο `nameText`. Για κάθε συμβάν `keyup` θα καλείται η μέθοδος `checkNameExists()` για να ελέγξει αν το όνομα ισούται με “admin”, και αν ισούται, προσθέτει ένα μήνυμα σφάλματος. Επίσης με κάθε συμβάν `keyup` ενημερώνεται και η συνιστώσα με id `nameMsg` μέσω του χαρακτηριστικού `update` και εμφανίζεται αμέσως το μήνυμα αν υπάρχει (Εικόνα 3).



Εικόνα 3. Παράδειγμα validation με Ajax listener

Σημείωση: Σε μια εφαρμογή πιθανώς να γινόταν σύγκριση του name ή username με μια λίστα ονομάτων.

PrimeFaces dataTable

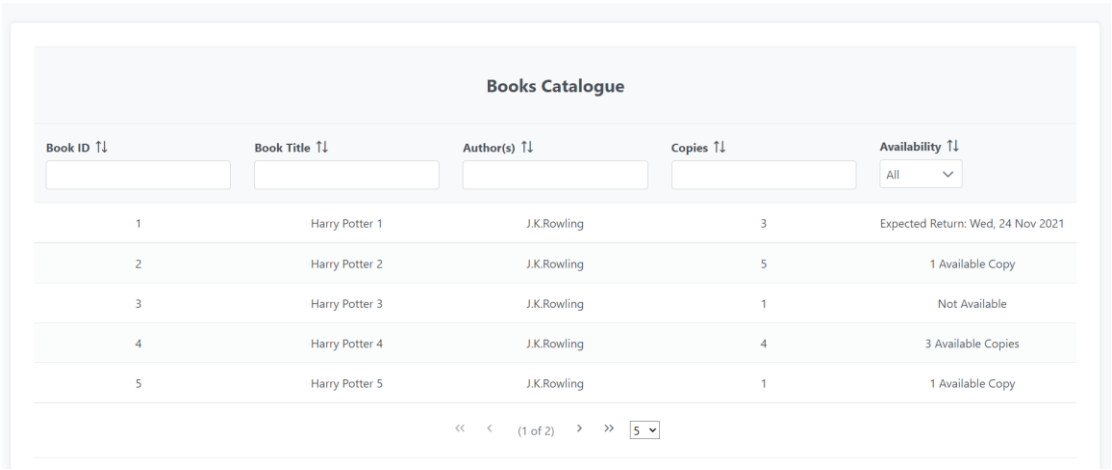
Το dataTable της PrimeFaces έχει διάφορες πρόσθετες δυνατότητες (K. Siva, P. Reddy 2013, Σελ.201) που επεκτείνουν την κλασική λειτουργία του dataTable κυρίως μέσω νέων χαρακτηριστικών (attributes) και Ajax δυνατοτήτων όπως:

- Σελιδοποίηση (pagination)
- «Τεμπέλικη φόρτωση» (lazy loading)
- Ταξινόμηση (sorting)
- Φιλτράρισμα (filtering)
- Επιλογή γραμμής (row selection)
- Επεξεργασία γραμμής/κελιού
- Επεκτάσιμες γραμμές
- Ομαδοποίηση κ.ά.

Στην παρακάτω εικόνα φαίνεται ο κώδικας για ένα p:dataTable ο οποίος χρησιμοποιεί κάποιες από τις παραπάνω δυνατότητες (Εικόνα 4). Το παράδειγμά αυτό είναι από την εφαρμογή στο μέρος 2 και αποδίδει έναν κατάλογο βιβλίων ο οποίος φαίνεται στην επόμενη εικόνα (Εικόνα 5).

```
<p:dataTable var="book" value="#{bookView.lazyModel}"  id="bookTable"
             widgetVar="dtbooks"
             paginator="true" rows="5"
             paginatorTemplate="{FirstPageLink} {PreviousPageLink}
             {CurrentPageReport} {NextPageLink} {LastPageLink} {RowsPerPageDropdown}"
             paginatorPosition="bottom"
             rowsPerPageTemplate="5,10,20"
             stripedRows="true"
             reflow="true"
             lazy="true"
             selectionMode="single"
             selection="#{bookView.selectedBook}" rowKey="#{book.id}">
    <p:ajax event="rowSelect" update="form:view-book-content" oncomplete="PF('viewBookDialog').show()"/>
    <f:facet ...5 lines />

    <p:column field="id" headerText="#{msgs.BookID}"/>

    <p:column field="title" headerText="#{msgs.BookTitle}"/>

    <p:column field="bookAuthorListToString" headerText="#{msgs.Authors}"/>

    <p:column field="copies" headerText="#{msgs.Copies}"/>

    <p:column ...9 lines />

    <p:column ...13 lines />

</p:dataTable>
```

Εικόνα 4. Πίνακας dataTable από τον κατάλογο βιβλίων



Εικόνα 5. Κατάλογος Βιβλίων

Φαίνεται ότι χρησιμοποιούνται διάφορα χαρακτηριστικά στο παραπάνω παράδειγμα, για διευκόλυνση δίνεται ο παρακάτω πίνακας με τα χαρακτηριστικά που χρησιμοποιούνται στο dataTable μαζί με μια σύντομη εξήγηση τους.

Πίνακας χαρακτηριστικών dataTable από τον κατάλογο βιβλίων

Name	Default	Type	Description
var	null	String	Όνομα της μεταβλητής που χρησιμοποιείται για να αναφερθούμε σε κάθε δεδομένο.
value	null	Object	Τα δεδομένα που θα προβληθούν.
id	null	String	Μοναδικό αναγνωριστικό για την συνιστώσα.

widgetVar	null	String	Όνομα του widget (dataTable) από την μεριά του πελάτη. Υπάρχει ένα ενδιαφέρον άρθρο για όποιον θέλει να μάθει περισσότερα: http://blog.hatemalimam.com/intro-to-primefaces-widgetvar/
paginator	false	Boolean	Ενεργοποιεί την σελιδοποίηση.
rows	null	Integer	Αριθμός γραμμών ανά σελίδα.
paginatorTemplate	null	String	Πρότυπο της σελιδοποίησης.
paginatorPosition	both	String	Θέση της σελιδοποίησης.
rowsPerPageTemplate	null	String	Πρότυπο της ρύθμισης για το dropdown γραμμών ανά σελίδα. Μπορούμε να επιλέξουμε πόσες γραμμές να φαίνονται μέσω ενός dropdown.
stripedRows	false	Boolean	Ορίζει αν θα είναι ριγέ οι γραμμές για να ξεχωρίζουν τα δεδομένα.
reflow	false	Boolean	Ένας responsive τρόπος να εμφανίζει της στήλες στοιβαγμένες, ανάλογα το μέγεθος της οθόνης.
lazy	false	Boolean	Ελέγχει το lazy loading. Μπορεί να παραληφθεί καθώς εντοπίζεται αυτόματα αν το value συνδέεται με κάποιο LazyDataModel.
selectionMode	null	String	Ενεργοποιεί την δυνατότητα επιλογής σειράς, δεκτές τιμές “single” και “multiple”.
selection	null	Object	Αναφορά στο επιλεγμένο/α δεδομένο/α.
rowKey	null	String	Μοναδικό αναγνωριστικό μιας γραμμής. Πρέπει να ορίζεται όταν χρησιμοποιούμε το selection και μη-lazy πηγή πληροφοριών.

Σελιδοποίηση

Έχει προστεθεί σελιδοποίηση με χρήση του χαρακτηριστικού `paginator` και μερικών βοηθητικών χαρακτηριστικών (`rows, paginatorTemplate, paginatorPosition, rowsPerPageTemplate`) η οποία φαίνεται στην απόδοση του πίνακα στην Εικόνα 5.

Ταξινόμηση

Ορίζοντας το χαρακτηριστικό `sortBy` ή το χαρακτηριστικό `field` ενεργοποιείται η Ajax ταξινόμηση στη στήλη του πίνακα (Παράδειγμα 1). Κάθε στήλη έχει ένα χαρακτηριστικό `sortable` με default τιμή true άρα η ταξινόμηση είναι αυτόματα ενεργοποιημένη, αλλά χρειάζεται να οριστεί το πεδίο στο οποίο θα γίνει η ταξινόμηση. Θέτοντας το `sortable` ίσο με false η ταξινόμηση απενεργοποιείται.

```
//Τρόπος 1:
  <p:column field="id" headerText="#{msgs.BookID}"/>
//Τρόπος 2:
  <p:column headerText="#{msgsBookID}" sortBy="#{book.id}"/>
```

Το χαρακτηριστικό `field` είναι στενογραφία για να οριστεί η ταξινόμηση, το φιλτράρισμα καθώς η προβολή ιδιοτήτων μιας στήλης. Στην PrimeFaces στο `field, sortBy, filterBy`, μπορεί να παραληφθεί το όνομα του bean και να γραφτεί κατευθείαν το όνομα της ιδιότητας π.χ. `field="id"` αντί για `field="#{book.id}"`.

Ένα ακόμα ενδιαφέρον χαρακτηριστικό που δεν υπάρχει στον κώδικα μου είναι το `sortFunction`, με χρήση του μπορεί να οριστεί κάποιος αλγόριθμος ταξινόμησης. Αν δε οριστεί η p:dataTable χρησιμοποιεί ένα Java Comparator (PrimeFaces 10.0.0, dataTable).

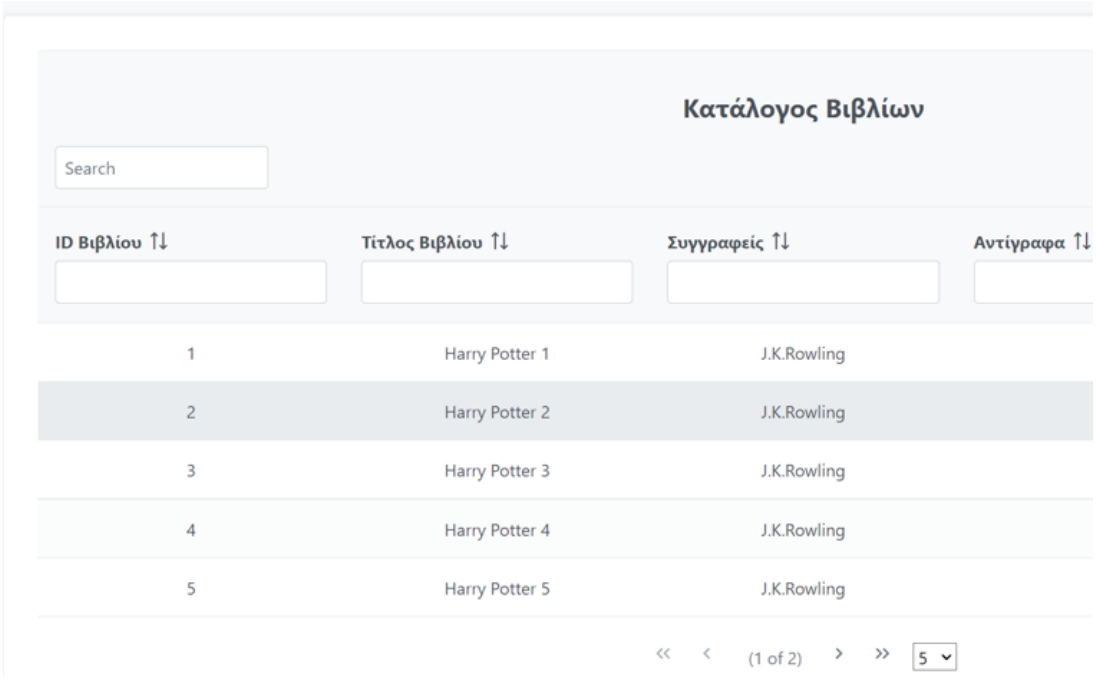
Φιλτράρισμα – Αναζήτηση

Στα φίλτρα υπάρχει το χαρακτηριστικό `filterable` το οποίο είναι Boolean και ορίζει την δυνατότητα φιλτραρίσματος. Το χαρακτηριστικό μπορεί να παραληφθεί στην οποία περίπτωση θα είναι ενεργοποιημένο το φιλτράρισμα καθώς η default τιμή του είναι true.

Το φιλτράρισμα δεν λειτουργεί μέχρι να οριστεί το χαρακτηριστικό `field` ή το `filterBy` σε κάποια στήλη, το οποίο δηλώνει το κριτήριο φιλτραρίσματος όπως στο παράδειγμα στην Εικόνα 4.

Στον κώδικα έχει προστεθεί φιλτράρισμα σε κάθε στήλη ξεχωριστά μέσω του χαρακτηριστικού `field`, αλλά υπάρχει και η δυνατότητα χρήσης ενός πεδίου αναζήτησης για όλο τον πίνακα. Για να γίνει αυτό πρέπει να γραφτεί στο header του dataTable ο παρακάτω κώδικας:

```
...
  <f:facet name="header">
      <div style="text-align: center">
            <h2>Κατάλογος Βιβλίων</h2>
      </div>
      <p:inputText id="globalFilter"
      onkeyup="PF('bookTable').filter()"
      placeholder="Search"/>
  </f:facet>
...
```



Εικόνα 6. Προσθήκη Global Αναζήτησης

Φαίνεται ότι προστέθηκε ένα πεδίο αναζήτησης στην κεφαλίδα του πίνακα (Εικόνα 6). Είναι σημαντικό το id να είναι `globalFilter`, το οποίο είναι κατοχυρωμένο id για το DataTable. Το φίλτρο είναι global («παγκόσμιο», καλύπτει όλο τον πίνακα) και καλύπτει όλα τα fields, αυτό υλοποιείται καλώντας τη μέθοδο `filter()` από ένα API στη μεριά του χρήστη.

Αξίζει να αναφερθεί τέλος το χαρακτηριστικό `filterMatchMode` το οποίο ορίζει τον τρόπο που θα φιλτράρονται τα δεδομένα, default τιμή είναι το `startsWith`. Άλλες τιμές που μπορεί να πάρει είναι `endsWith`, `contains`, `exact`, `lt` (less than), `lte` (less than or equal), `gt` (greater than), `gte` (greater than or equal), `equals`, `in`, `range`. Αν δεν αρκούν οι μέθοδοι που ήδη υπάρχουν, υπάρχει η δυνατότητα επιλογής μεθόδου φιλτραρίσματος μέσω του χαρακτηριστικού `filterFunction`.

Προτείνεται να χρησιμοποιείται ένα scope μεγαλύτερο από το request scope, όπως το view scope ώστε να διατηρούνται τα δεδομένα των φίλτρων (PrimeFaces 10.0.0, dataTable, Filtering).

Επιλογή σειράς (γραμμής)

Για την επιλογή γραμμής ανέφερα κάποια χαρακτηριστικά στον παραπάνω πίνακα, όπως το `selection`, `selectionMode`, `rowKey`. Υπάρχουν επίσης τα συμβάντα (`event`) `rowSelect` και `rowUnselect` τα οποία ενεργοποιούνται όταν επιλεχθεί μία γραμμή (K. Siva, P. Reddy 2013, σελ. 208).

Στον κώδικα της εικόνας 4 ορίζονται τα χαρακτηριστικά που χρειάζονται για την επιλογή γραμμής. Το χαρακτηριστικό `selectionMode=single` κάνει δυνατή την επιλογή μόνο μίας γραμμής και το χαρακτηριστικό `selection` αντιστοιχεί την επιλεγμένη γραμμή στην ιδιότητα `selectedBook`.

Υπάρχει και μια συνιστώσα `ajax` με event `rowSelect`, άρα δράση που θα γίνεται κάθε φορά που επιλέγεται μια γραμμή. Χρησιμοποιείται το χαρακτηριστικό `update` για να ενημερωθούν κάποια δεδομένα στη συνιστώσα με id `view-book-content`. Η συγκεκριμένη συνιστώσα η οποία φαίνεται παρακάτω περιλαμβάνει πολλές τιμές της ιδιότητας `selectedBook` (Εικόνα 7).

```
<p:dialog header="#{msgs.BookDetailsDialogHeader}" showEffect="fade" modal="true"
          widgetVar="viewBookDialog" responsive="true" fitViewport="true">
    <p:outputPanel id="view-book-content" class="ui-fluid" style="width: 400px;">
        <p:graphicImage id="view-image" value="#{bookView.selectedBook.imagepath}" width="130px" height="200px"
                        alt="#{msgs.NoPictureAvailable}"/>
        <p:panelGrid columns="2">
            <p:outputLabel for="view-id" value="#{msgs.IDLabel}"/>
            <p:outputLabel id="view-id" value="#{bookView.selectedBook.id}"/>

            <p:outputLabel for="view-title" value="#{msgs.TitleLabel}"/>
            <p:outputLabel id="view-title" value="#{bookView.selectedBook.title}"/>

            <p:outputLabel for="view-copies" value="#{msgs.CopiesLabel}"/>
            <p:outputLabel id="view-copies" value="#{bookView.selectedBook.copies}"/>

            <p:outputLabel for="view-authors" value="#{msgs.AuthorsLabel}"/>
            <p:outputLabel id="view-authors" value="#{bookView.selectedBook.bookAuthorListToString}"/>
        </p:panelGrid>
    </p:outputPanel>
</p:dialog>
```

Εικόνα 7. Dialog για την προβολή βιβλίου

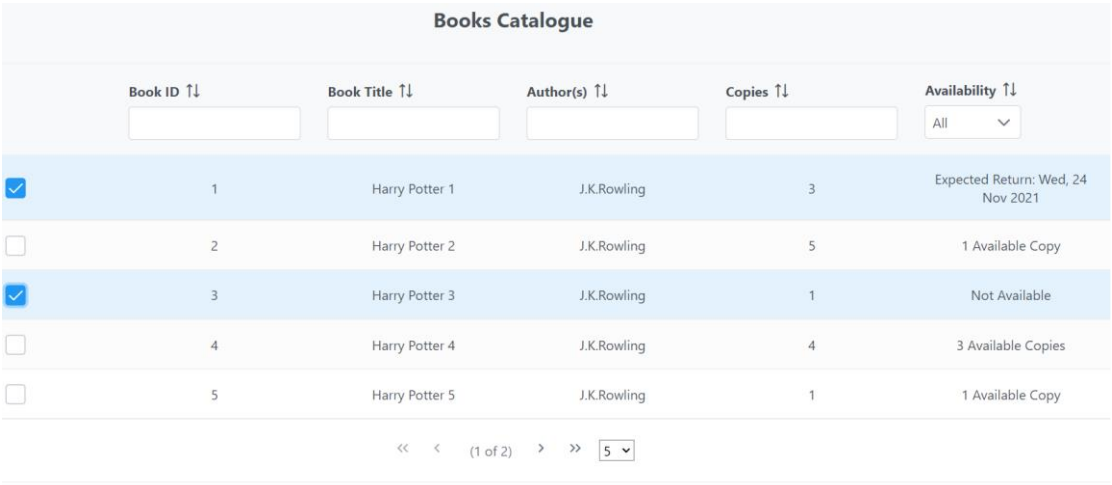
Όταν επιλεχθεί μια γραμμή του πίνακα:

- το selectedBook ενημερώνεται στο bean μέσω του χαρακτηριστικού `selection`, ώστε να ισούται με το αντικείμενο της επιλεγμένης γραμμής,
- Ενημερώνεται μέσω ajax η συνιστώσα με id `view-book-content` μέσω του χαρακτηριστικού `update`,
- Τέλος, μέσω του χαρακτηριστικού `oncomplete` και τη μέθοδο `show()` γίνεται ορατή η συνιστώσα με widgetVar "`viewBookDialog`"s το οποίο είναι ένα `dialog` με πληροφορίες για το επιλεγμένο βιβλίο.

Είναι δυνατή η επιλογή πολλαπλών γραμμών του πίνακα θέτοντας το χαρακτηριστικό του πίνακατα dataTable `selectionMode="multiple"`. Είναι χρήσιμη δυνατότητα για ένα σενάριο στο οποίο χρειάζεται η μαζική διαγραφή, η μαζική επεξεργασία ή και άλλες ενέργειες.

Για την εμφάνιση checkbox για πολλαπλή επιλογή γραμμών, το χαρακτηριστικό `selectionMode` πρέπει να προστεθεί σε μία στήλη (Εικόνα 8) (PrimeFaces 10.0.0, dataTable, Row Selection).

```
<p:column selectionMode="multiple"/>
```



Εικόνα 8. Επιλογή πολλαπλών γραμμών

Lazy Loading

Lazy loading μεταφράζεται σαν «τεμπέλικη φόρτωση» και είναι μια τεχνική που χρησιμοποιείται για να φορτώνονται μερικώς τα δεδομένα του πίνακα, δηλαδή μόνο τις γραμμές που θα φαίνονται.

Τα δεδομένα που θα εμφανιστούν στον πίνακα υπολογίζονται από την τωρινή σελίδα, τον αριθμό των γραμμών ανά σελίδα, την ταξινόμηση αν υπάρχει καθώς και τα κριτήρια αναζήτησης (φιλτραρίσματος).

Το lazy loading φορτώνει τα δεδομένα μέσω Ajax επιτυγχάνοντας γρηγορότερη φόρτωση της σελίδας καθώς δε χρειάζεται να φορτωθεί όλη η λίστα των βιβλίων, και ούτε χρειάζεται να ξανά-αποδοθεί η σελίδα.

Το dataTable υποστηρίζει lazy loading μέσω του Boolean χαρακτηριστικού "`lazy`". Χχρησιμοποιεί την μέθοδο load της κλάσης `org.primefaces.model.LazyDataModel` για να λάβει τα δεδομένα από τον server φιλτραρισμένα. Υπάρχει η δυνατότητα να δημιουργηθεί κλάση η οποία να επεκτείνει την LazyDataModel σε περίπτωση που χρειάζεται επιπλέον παραμετροποίηση. Η λίστα που εμφανίζεται στον πίνακα dataTable είναι έξοδος της μεθόδου load() του LazyDataModel και μπορεί να παραμετροποιηθεί (PrimeFaces Documentation v10.0.0, DataTable).

```
public List<Book> load(int first, int pageSize, Map<String, SortMeta>
sortBy, Map<String, FilterMeta> filters) {

// code to fetch data based on applied sorting/filters

}
```

Οι παράμετροι της μεθόδου load είναι οι εξής:

- `first`: Ορίζει το δείκτη της πρώτης γραμμής που θα προβληθεί
- `pageSize`: Ορίζει τον αριθμό των γραμμών που θα φορτώσει ανά σελίδα
- `sortBy`: ένα map με όνομα πεδίων (sortField) και σειρά ταξινόμησης (sortOrder enum της PrimeFaces με τιμές ASCENDING ή DESCENDING)
- `field`: ένα map με όνομα πεδίων ως κλειδιά (π.χ. "id" από το field="id") και τις αντίστοιχες τιμές των φίλτρων

Στο παράδειγμα από την υλοποίηση μέρος 2 (Εικόνα 4) βλέπουμε ότι το value του dataTable είναι ίσο με την ιδιότητα lazyModel. Η ιδιότητα αυτή είναι η υλοποίηση του δικού μου μοντέλου που επεκτείνει το LazyDataModel για τον κατάλογο των βιβλίων.

Υπάρχουν και άλλα χαρακτηριστικά και δυνατότητες του p:dataTable που δεν αναφέρθηκαν εδώ όπως επεξεργασία κελιού, επέκταση γραμμής, ομαδοποίηση, εξαγωγή δεδομένων κ.ά.. Για όποιον ενδιαφέρεται να μάθει παραπάνω μπορεί να ανατρέξει στις παρακάτω αναφορές, καθώς και να κοιτάξει τις τελευταίες εκδόσεις των τεχνολογιών στις οποίες σίγουρα θα έχουν προστεθεί καινούργιες και ενδιαφέρον δυνατότητες.

Αναφορές

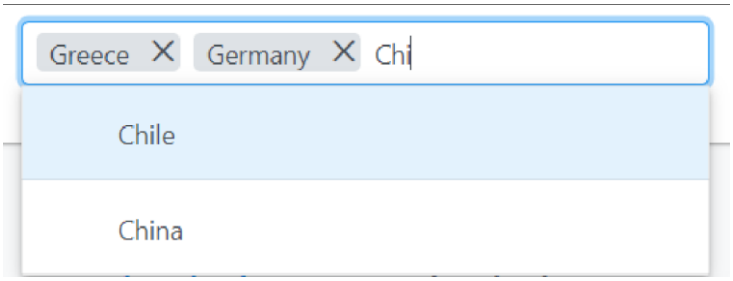
- Column documentation: https://primefaces.github.io/primefaces/10_0_0/#/components/column?id=column
- DataTable documentation: https://primefaces.github.io/primefaces/10_0_0/#/components/datatable
- DataTable showcase: http://www.primefaces.org:8080/showcase/ui/data/datatable/basic.xhtml?jfwid=02b0c
- K.Siva P. Reddy (2013), PrimeFaces Beginner's Guide, Κεφάλαιο 8

PrimeFaces autoComplete

Η `autoComplete` είναι μία συνιστώσα εισαγωγής δεδομένων και «αυτόματης συμπλήρωσης», δηλαδή προσφέρει ζωντανές υποδείξεις ενώ πληκτρολογείτε η είσοδος.

Οι υποδείξεις ορίζονται σε μία λίστα μέσω του χαρακτηριστικού `completeMethod`. Θέτοντας το χαρακτηριστικό `multiple` ίσο με `true` μπορούμε να εισάγουμε πολλαπλές τιμές στο πεδίο (PrimeFaces Documentation v10.0.0, autoComplete).

```
<p:autoComplete id="countries" multiple="true"
      value="#{countriesManagedBean.names}"
      completeMethod="#{countriesManagedBean.completeText}"/>
```



Εικόνα 9. Παράδειγμα αυτόματης συμπλήρωσης με πολλαπλές τιμές

PrimeFaces dialog & confirmDialog

Η συνιστώσα `dialog` είναι ένας «περιέχοντας», ένα δοχείο (container) για προβολή περιεχομένου σε ένα πλαίσιο που επικαλύπτει τη σελίδα (PrimeFaces Showcase v10.0.0, dialog). Θέτοντας το χαρακτηριστικό `modal=true` ο χρήστης δεν μπορεί να αλληλοεπιδράσει με την υπόλοιπη σελίδα μέχρι να εξαφανιστεί το πλαίσιο `dialog`. Μπορεί να έχει πολλές χρήσεις, εγώ το χρησιμοποιώ για να προβάλλω τα δεδομένα της λίστας μου καθώς και για να προβάλλω μια φόρμα για την επεξεργασία δεδομένων.

Εικόνα 10. Confirm Dialog

Το `confirmDialog` είναι από ένα πλαίσιο σαν το `dialog` το οποίο χρησιμοποιείται για επιβεβαίωση και έχει την ιδιότητα modal από μόνο του. Ένα `confirmDialog` χρησιμοποιείται για να εμφανίζει μήνυμα επιβεβαίωσης για να προχωρήσει μία κίνηση π.χ. διαγραφή βιβλίου. Αφού το `confirmDialog` έχει τόσο συγκεκριμένη λειτουργία έχει προστεθεί η δυνατότητα θέτοντας το χαρακτηριστικό `global=true` να έχουμε ένα `confirmDialog` για πολλές διαφορετικές επιβεβαιώσεις στην σελίδα, έτσι ώστε να μη χρειάζεται να γράφουμε πολλές φορές τον ίδιο κώδικα (PrimeFaces Documentation v10.0.0, ConfirmDialog).

PrimeFaces idleMonitor

Η συνιστώσα `idleMonitor` παρακολουθεί τις ενέργειες του χρήστη σε μια σελίδα και ειδοποιεί όταν ο χρήστης είναι αδρανής και όταν είναι ενεργός.

Στην JSF κάθε συνεδρία λήγει κάποια στιγμή και μαζί της λήγουν και τα session και view scoped beans οπότε η σύνδεση του χρήστη χάνεται καθώς και αρκετές λειτουργίες της σελίδας. Για να αντιμετωπίσω αυτό το πρόβλημα χρησιμοποίησα το `idleMonitor` έτσι ώστε όταν ο χρήστης είναι αδρανής μέχρι το τέλος μια συνεδρίας να γίνεται με κάποιο τρόπο ανανέωση της σελίδας.

```
<p:idleMonitor
      timeout="#{session.maxInactiveInterval*1000}"
      onidle="PF('idleDialog').show();"/>
```

Στο χαρακτηριστικό `timeout` θέτουμε τον χρόνο τον οποίο πρέπει να μείνει αδρανής ο χρήσης ώστε να γίνει κάποια ενέργεια. Ο χρόνος που έχουμε ορίσει είναι ο χρόνος που κάνει η συνεδρία να λήξει. Όταν ο χρήστης μείνει αδρανής για το χρόνο αυτόν, εμφανίζεται η συνιστώσα με widgetVar “idleDialog” όπως έχουμε ορίσει στο χαρακτηριστικό `onidle`. Η συνιστώσα αυτή είναι ένα απλό dialog με ένα κουμπί ανακατεύθυνσης στην αρχική σελίδα, όταν πατηθεί και φορτωθεί η σελίδα, ξαναδημιουργούνται τα beans που χρειάζονται και έχουν λήξει, οπότε λύνεται το πρόβλημα μας. Ότι δεδομένα υπήρχαν στο προηγούμενο session όμως χάνονται.

Συνιστώσες που χρησιμοποίησα

Παρακάτω αναφέρονται οι συνιστώσες που χρησιμοποιήθηκαν στην εφαρμογή βιβλιοθήκης. Αναφέρονται πρώτα οι συνιστώσες της JSF και ύστερα της PrimeFaces. Οι συνιστώσες περιγράφονται στις εξής αναφορές:

JSF 2.2 docs: https://docs.oracle.com/javaee/7/javaserver-faces-2-2/vdldocs-facelets
JSF 2.3 docs: https://javaee.github.io/glassfish/doc/5.0/vdldoc/index.html
JSF tutorials: https://www.oracle.com/java/technologies/javaserverfaces.html
JSF 2.3 Specs: https://download.oracle.com/otn-pub/jcp/jsf-2_3-final-eval-spec/JSF_2.3.pdf

PrimeFaces παραδείγματα (Showcase): https://www.primefaces.org/showcase
PrimeFaces documentation: https://primefaces.github.io/primefaces/10_0_0/#/

Συνιστώσες JSF που χρησιμοποίησα

`<ui:composition>` προσδιορίζει μία σύνθεση η οποία προαιρετικά χρησιμοποιεί κάποιο πρότυπο (template).

`<ui:define>` προσδιορίζει το περιεχόμενο που θα εισαχθεί σε μια σελίδα από το πρότυπο.

`<ui:insert>` εισάγει περιεχόμενο σε ένα πρότυπο.

`<ui:include>` χρησιμοποιείται για να ενθυλακώσουμε και επαναχρησιμοποιήσουμε περιεχόμενο σε πολλές σελίδες.

`<h:form>` αποδίδει ένα στοιχείο φόρμας HTML.

`<h:outputLabel>` αποδίδει ένα στοιχείο label της HTML.

`<h:link>` αποδίδει έναν σύνδεσμο HTML.

`<h:outputLink>` αποδίδει έναν σύνδεσμο HTML. Η διαφορά με το h:link είναι ότι δε χρησιμοποιεί τους κανόνες πλοήγησης της JSF και γι' αυτό χρησιμοποιείται κυρίως για εξωτερικούς συνδέσμους.

`<h:panelGroup>` αποδίδει ένα στοιχείο HTML div ή span ανάλογα το χαρακτηριστικό layout.

`<h:outputStylesheet>` αποδίδει ένα σύνδεσμο HTML για ένα αρχείο css.

`<f:view>` περιέχοντας για όλες τις ενέργειες JSF συνιστωσών σε μία σελίδα.

`<f:selectItem>` προσθέτει ένα παιδί UISelectItem στο σχετικό UIComponent (π.χ. SelectOneMenu) με την πλησιέστερη "γονική" UIComponent ενέργεια.

`<f:selectItems>` προσθέτει ένα παιδί UISelectItems στο UIComponent (π.χ. SelectOneMenu) που σχετίζεται με την πλησιέστερη ενέργεια γονικού UIComponent.

`<f;facet>` Καταχωρεί ένα επώνυμο facet στο UIComponent που σχετίζεται με την πλησιέστερη ενέργεια γονικού UIComponent.

`<f:convertDateTime>` χρησιμοποιείται για να επεξεργαστούμε τη μορφή με την οποία εμφανίζεται η ώρα.

`<f:setPropertyActionListener>` χρησιμοποιείται για να θέσουμε μια τιμή στην έκφραση που ορίζουμε στο χαρακτηριστικό "target" αφότου γίνει κάποια ενέργεια.

Συνιστώσες PrimeFaces που χρησιμοποίησα

<p:panelGrid> επέκταση του κλασικού h:panelGrid με δυνατότητα responsiveness**,** ομαδοποίησης και θεματοποίησης.

<p:outputLabel> επέκταση της κλασικής συνιστώσας h:outputLabel.

<p:inputNumber> διαμορφώνει το πεδίο με ένα αριθμητικό String. Υποστηρίζει σύμβολα νομίσματος, ελάχιστη και μέγιστη τιμή, αρνητικούς αριθμούς κ.ά..

<p:inputText> επέκταση της κλασικής συνιστώσας h:inputText με δυνατότητες επεξεργασίας εμφάνισης.

<p:commandButton> επέκταση του κλασικού h:commandButton με δυνατότητες ajax και θεματοποίησης.

<p:menu> μία συνιστώσα πλοήγησης με ένα επίπεδο υπό-μενού.

<p:submenu> μία συνιστώσα εμφωλευμένη σε συνιστώσες μενού και αντιπροσωπεύει ένα υπό-μενού.

<p:menuitem> χρησιμοποιείται σε διάφορες συνιστώσες μενού και αντιπροσωπεύει ένα αντικείμενο του.

<p:divider> διαχωριστικό που χρησιμοποιείται για να διαχωρίζει περιεχόμενα.

<p:selectOneMenu> επέκταση της κλασικής συνιστώσας h:selectOneMenu.

<p:idleMonitor> παρακολουθεί τις ενέργειες του χρήστη σε μια σελίδα και ειδοποιεί όταν ο χρήστης είναι αδρανής και όταν είναι ενεργός.

<p:dialog> μία συνιστώσα πλαισίου η οποία μπορεί να επικαλύψει άλλα στοιχεία σε μια σελίδα.

<p:outputPanel> μία συνιστώσα πλαισίου με τη δυνατότητα καθυστερημένης φόρτωσης.

<p:fileUpload> είναι κάτι παραπάνω από ένα απλό input type=”file” χρησιμοποιώντας χαρακτηριστικά της HTML5 για ένα πλούσιο αποτέλεσμα καθώς και υποβάθμιση για παλαιότερα προγράμματα περιήγησης.

<p:button> επέκταση της κλασικής συνιστώσας h:button με δυνατότητες επεξεργασίας εμφάνισης.

<p:autoComplete> προσφέρει ζωντανές υποδείξεις ενώ πληκτρολογείτε η είσοδος και έχει τη δυνατότητα εισαγωγής πολλών δεδομένων.

<p:dataTable> εμφανίζει τα δεδομένα σε μορφή πίνακα.

<p:toolbar> μία συνιστώσα οριζόντιας ομαδοποίησης.

<p:toolbarGroup> μία βοηθητική συνιστώσα της p:toolbar για να ορίζει placeholders.

<p:linkButton> ένας απλός σύνδεσμος με εμφάνιση κουμπιού.

<p:ajax> επέκταση της κλασικής συνιστώσας f:ajax.

<p:column> επέκταση της κλασικής συνιστώσας h:column που χρησιμοποιείται σε συνιστώσες όπως dataTable, treeTable κ.ά..

<p:resetInput> οι συνιστώσες εισόδου διατηρούν τις τιμές τους όταν οι έλεγχοι αποτυγχάνουν. Το resetInput χρησιμοποιείται για να καθαρίσει αυτές τις τιμές ώστε οι συνιστώσες να παίρνουν τις τιμές τους από το backing bean.

<p:scrollPanel> χρησιμοποιείται για να προβάλει περιεχόμενο σε ένα πλαίσιο με δυνατότητα scrolling.

<p:graphicImage> επεκτείνει την κλασική συνιστώσα h:graphicImage.

<p:confirmDialog> είναι ένα αντικατάστατο για το legacy javascript παράθυρο επιβεβαίωσης. Σημαντικά πλεονεκτήματα είναι η εμφάνιση, παραμετροποίηση και αποφυγή αποκλεισμού αναδυόμενων παραθύρων.

<p:password> επεκτείνει τη συνιστώσα h:inputSecret με δυνατότητα θεματοποίησης και δείκτη ισχύς κωδικού.

Μορφοποίηση σελίδων με Templates

Η Τεχνολογία JavaServer Faces προμηθεύει εργαλεία για την υλοποίηση διεπαφών χρήστη οι οποίες μπορούν εύκολα να επεκταθούν και επαναχρησιμοποιηθούν. Τα Templates (πρότυπα) είναι ένα χρήσιμο εργαλείο των Facelets το οποίο επιτρέπει την δημιουργία μιας σελίδα η οποία θα δρα ως βάση, ή αλλιώς πρότυπο, για τις υπόλοιπες σελίδες σε μια εφαρμογή. Έτσι επίσης είναι εφικτή η μορφοποίηση της σελίδας αλλάζοντας μόνο το πρότυπο και όχι κάθε σελίδα.

Χρησιμοποιώντας πρότυπα μπορεί να ξαναχρησιμοποιηθεί το ίδιο κομμάτι κώδικα και αποφεύγεται η δημιουργία πανομοιότυπων σελίδων. Η χρήση προτύπων επίσης βοηθάει στην διατήρηση μιας ενιαίας εμφάνισης σε μια εφαρμογή με πολλαπλές σελίδες. Υπάρχει παράδειγμα στο μέρος 2 από την υλοποίηση της εφαρμογής (Oracle 2014, κεφ. 8.4).

Localization

[Oracle 2014, Κεφ. 20.2]
Τα μηνύματα και οι ετικέτες (labels) θα πρέπει να προσαρμόζονται ανάλογα με τη γλώσσα και περιοχή του χρήστη. Υπάρχουν δύο τρόποι για να γίνει αυτό σε μια εφαρμογή ιστού.

- Δημιουργία αντίγραφων των σελίδων σε κάθε μία από τις γλώσσες που χρειάζεται και ύστερα ανάλογα με την τοποθεσία να στέλνεται η ανάλογη σελίδα στον client.
- Η απομόνωση των μηνυμάτων και πληροφοριών σε «πακέτα πόρων» (resource bundles), και ανάλογα τη γλώσσα θα επιλέγεται το πακέτο για την προβολή των μηνυμάτων.

Στην εφαρμογή μου διάλεξα το δεύτερο τρόπο καθώς πιστεύω πως είναι πιο εύκολο, λειτουργικό και οργανωμένο να ενημερώνεις πακέτα μηνυμάτων παρά ολόκληρες σελίδες. Υπάρχει παράδειγμα στο μέρος 2 από την υλοποίηση της εφαρμογής.

ORM

[Keith, Schincariol, Nardone 2018, Κεφ. 1, σελ 3]

Στο αντικειμενοστραφές μοντέλο έχουμε κλάσεις. Στην σχεσιακή βάση δεδομένων έχουμε πίνακες. Αυτά τα δύο μοντέλα μοιάζουν κάπως μεταξύ τους και συχνά θέλουμε να τα συσχετίζουμε. Η τεχνική που χρησιμοποιείται για να γεφυρώσουμε το χάσμα ανάμεσα από το αντικειμενοστραφές μοντέλο και το σχεσιακό μοντέλο ονομάζεται object-relationship mapping ή ORM. Η τεχνική παίρνει το όνομα μας της από την ιδέα ότι κατά κάποιο τρόπο χαρτογραφούμε (mapping) από το ένα μοντέλο στο άλλο, με στόχο την εισαγωγή ενός διαμεσολαβητή ο οποίος χειρίζεται αυτόματα τη μετατροπή μεταξύ των μοντέλων.

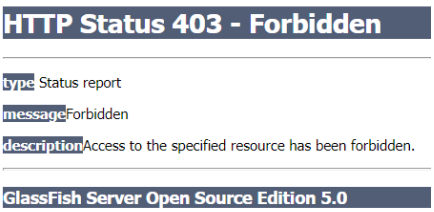
Ο παρακάτω Πίνακας δίνει μια αντιστοίχιση μερικών εννοιών των 2 μοντέλων.

Σχεσιακό μοντέλο και SQL	**Αντικειμενοστραφές μοντέλο και Java**
πίνακας	κλάση
γραμμή πίνακα	αντικείμενο
στήλη πίνακα	πεδίο (μη στατικό)
πρωτεύον κλειδί	πρωτεύον κλειδί
ξένο κλειδί	αναφορά σε αντικείμενο

Mapping Errors to Error Pages

[Geary, 2010 σελ.582, Oracle 2014 κεφ.6.5.3.1]

Όταν τρέχει μια εφαρμογή και συναντήσει σφάλμα, εμφανίζεται ένα μήνυμα σφάλματος σαν το παρακάτω παράδειγμα:



Εικόνα 11. Μήνυμα σφάλματος 403

Για να αντικατασταθεί η σελίδα σφάλματος με μία καλύτερη-προσαρμοσμένη χρησιμοποιείται το tag `error-page` στο αρχείο web.xml. Για να οριστεί η σελίδα που θα εμφανίζεται όταν υπάρξει σφάλμα χρησιμοποιείται το tag `location`. Για να επιλέξουμε για ποιο σφάλμα θα εμφανίζεται αυτή η σελίδα υπάρχουν δύο τρόποι:

- Επιλέγοντας έναν κωδικό σφάλματος HTTP με το tag `error-code`.
- Επιλέγοντας μία κλάση εξαίρεσης Java με το tag `exception-type`.

Στον παρακάτω κώδικα ορίζεται μία σελίδα για την εξαίρεση ViewExpiredException, τα οποίο εμφανίσει τη σελίδα index.xhtml.

Ορίζεται και μια σελίδα για τον κωδικό σφάλματος 403 ο οποίος θα εμφανίσει τη σελίδα error403.xhtml.

```
<error-page>
    <exception-type>javax.faces.application.ViewExpiredException</exception-type>
    <location>/index.xhtml</location>
</error-page>
<error-page>
    <error-code>403</error-code>
    <location>/error/error403.xhtml</location>
</error-page>
```

Εικόνα 12. Ορισμός σελίδας σφάλματος

Μπορεί να οριστεί επίσης μία κοινή σελίδα για όλα τα σφάλματα προσθέτοντας την κλάση java.lang.Throwable η οποία είναι υπερκλάση όλων των σφαλμάτων και εξαιρέσεων της Java.

```
...
<error-page>
      <exception-type>java.lang.Throwable</exception-type>
      <location>/error_page.xhtml</location>
</error-page>
...
```

Ασφάλεια δεδομένων, αυθεντικοποίηση και εξουσιοδότηση

[Oracle 2014, Κεφ. 47.1]
Υπάρχουν 2 βασικά είδη ασφαλείας σε πολύ-επίπεδες εφαρμογές ιστού Java EE:

- Δηλωτική ασφάλεια (Declarative security) η οποία εκφράζει τις απαιτήσεις ασφάλειας ενός «τμήματος» (component) χρησιμοποιώντας deployment descriptors ή annotations.

  Ένα deployment descriptor είναι ένα αρχείο XML που δεν ανήκει στην κύρια εφαρμογή και περιγράφει τη δομή ασφάλειας της σελίδας, τους ρόλους ασφάλειας, τον έλεγχο πρόσβασης και τις απαιτήσεις αυθεντικοποίησης.

  Τα annotations, ή αλλιώς metadata, περιγράφουν πληροφορίες για την ασφάλεια μέσα σε ένα αρχείο κλάσεων.

- Προγραμματιστική ασφάλεια (Programmatic security) η οποία είναι κομμάτι της εφαρμογής και παίρνει αποφάσεις για την ασφάλεια. Είναι χρήσιμη όταν η δηλωτική ασφάλεια δεν αρκεί για να εκφραστεί το μοντέλο ασφάλειας της εφαρμογής.

Αυθεντικοποίηση και εξουσιοδότηση με GlassFish Server

Realms, Users, Groups, Roles

[Oracle 2014, Κεφ. 17.5.1]
Ένα «βασίλειο» (**realm**) είναι μια πολιτική τομέα ασφαλείας για εφαρμογές ιστού ή διακομιστές εφαρμογών. Ένα realm περιλαμβάνει μια συλλογή από χρήστες (users), οι οποίοι μπορεί να ανήκουν σε κάποια ομάδα (group).

Η υπηρεσία αυθεντικοποίησης διακομιστή Java EE μπορεί να διαχειρίζεται χρήστες σε πολλαπλά βασίλεια. Τα βασίλεια file, admin-realm, και certificate έρχονται ήδη έτοιμα για τον GlassFish Server.

Ένας χρήστης (**user**) είναι ένα άτομο, μία ταυτότητα η οποία έχει οριστεί στον GlassFish Server. Ένας χρήστης μπορεί να συνδέεται με κάποιους ρόλους που του δίνουν πρόσβαση σε πόρους οι οποίοι είναι προστατευμένοι από αυτούς τους ρόλους. Ένας χρήστης μπορεί να ανήκει και σε κάποια ομάδα.

Μία ομάδα (**group**) είναι ένα σύνολο από αυθεντικοποιημένους χρήστες με κοινά χαρακτηριστικά. Η ομαδοποίηση των χρηστών κάνει πιο εύκολη την διαχείριση της ασφάλειας για μεγάλο αριθμό χρηστών.

Ένας ρόλος (**role**) είναι μια αφηρημένη έννοια που αντιπροσωπεύει την άδεια πρόσβασης σε συγκεκριμένους πόρους σε μια εφαρμογή.

Μία ομάδα και ένας ρόλος έχουν διαφορετικό πεδίο εφαρμογής. Η ομάδα ορίζεται για όλο το εύρος του GlassFish Server, ενώ ένας ρόλος συσχετίζεται μόνο με μια συγκεκριμένη εφαρμογή του GlassFish Server.

Security Constraint

[Oracle 2014, Κεφ. 48.2.1]
Ένας περιορισμός ασφάλειας (security constraint) χρησιμοποιείται για να οριστεί η πρόσβαση σε κάποιους πόρους με τη χρήση χαρτογράφησης URL.

Αν γίνεται χρήση servlet στην εφαρμογή οι περιορισμοί ασφάλειας μπορούν να οριστούν με τη χρήση annotations. Διαφορετικά πρέπει να οριστεί ένα security-constraint στο αρχείο deployment descriptor της εφαρμογής.

Ένα security-constraint μπορεί να περιλαμβάνει:

- **Συλλογή πόρων ιστού (web resource collection)**: Μία λίστα από μοτίβα URL και μεθόδους HTTP που περιγράφουν τους πόρους που προστατεύονται.
- **Περιορισμό εξουσιοδότησης (authorization constraint)**: Ορίζει αν θα χρησιμοποιηθεί αυθεντικοποίηση και τους ρόλους για τους οποίους επιτρέπεται η πρόσβαση στους πόρους.
- **Περιορισμό δεδομένων χρήστη (user data constraint)**: Ορίζει πως προστατεύονται τα δεδομένα όταν μεταφέρονται μεταξύ του client και του server.

Authentication Mechanisms

[Oracle 2014, Κεφ. 48.2.2]
Όταν ορίζεται μηχανισμός αυθεντικοποίησης, ο χρήστης πρέπει να αυθεντικοποιηθεί πριν του δοθεί πρόσβαση σε περιορισμένους πόρους. Ένας μηχανισμός αυθεντικοποίησης ορίζει:

- Τον τρόπο με τον οποίο ο χρήστης αποκτά πρόσβαση στα περιεχόμενα της σελίδας
- το βασίλειο στο οποίο ο χρήστης θα αυθεντικοποιείται
- Με αυθεντικοποίηση φόρμας, κάποια επιπλέον χαρακτηριστικά

Στη Java EE 7 υπάρχουν οι παρακάτω τρόποι αυθεντικοποίησης:

  - Basic authentication
  - Form-based authentication
  - Digest authentication
  - Client authentication
  - Mutual authentication

- Στην βασική αυθεντικοποίηση (basic authentication) γίνονται τα εξής βήματα:
  1. Ο client ζητάει πρόσβαση σε προστατευμένους πόρους.
  2. Ο server επιστρέφει ένα παράθυρο (dialog box) το οποίο ζητάει όνομα χρήστη και κωδικό.
  3. Ο client υποβάλει όνομα χρήστη και κωδικό.
  4. Ο server αυθεντικοποιεί τον χρήστη στο βασίλειο που έχει οριστεί, και αν είναι επιτυχής εμφανίζει τους πόρους (π.χ. κάποια σελίδα)

- Στην αυθεντικοποίηση φόρμας (form-based authentication)
  1. Ο client ζητάει πρόσβαση σε προστατευμένους πόρους.
  2. Αν ο χρήστης είναι ασύνδετος, ο server ανακατευθύνει τον client σε μια σελίδα σύνδεσης.
  3. Ο client υποβάλει τη φόρμα σύνδεσης.

4. Ο server αυθεντικοποιεί τον χρήστη στο βασίλειο που έχει οριστεί, και αν είναι επιτυχής και ο χρήστης έχει το δικαίωμα πρόσβασης στους πόρους, εμφανίζει τους πόρους (π.χ. κάποια σελίδα). Αν η αυθεντικοποίηση αποτύχει ο client ανακατευθύνεται σε μια σελίδα σφάλματος σύνδεσης.

Στην αυθεντικοποίηση φόρμας ο προγραμματιστής έχει τον έλεγχο της εμφάνισης της σελίδας σύνδεσης καθώς και της σελίδας σφάλματος.

Μέρος 2 - Εφαρμογή επίδειξης

Το Μέρος 2 παρουσιάζει την ανάπτυξη εφαρμογής ιστού και βάσης δεδομένων, και συγκεκριμένα απαιτήσεις λογισμικού, ανάλυση και σχεδιασμό, και υλοποίηση για την βάση δεδομένων σε Oracle XE και την κυρίως εφαρμογή σε Java.
Ο κώδικας της εφαρμογής είναι διαθέσιμος στον ιστότοπο
https://github.com/dtsimaras/MyLMS

Στόχοι της Εφαρμογής

Η εφαρμογή είναι λογισμικό ιστού με βάση δεδομένων. Αποσκοπεί στην επίδειξη της χρήσης τεχνολογιών Java/Jakarta EE που παρουσιάστηκαν στο Μέρος 1, και συγκεκριμένα των τεχνολογιών JSF/PrimeFaces, CDI, EJB, και JPA, αυθεντικοποίησης και εξουσιοδότησης. Η λειτουργικότητα της εφαρμογής είναι επιλεγμένη για τον σκοπό αυτό. Η εφαρμογή δεν αποσκοπεί σε λειτουργική πληρότητα.

Πεδίο της εφαρμογής

Επιλέξαμε το πεδίο διαχείρισης βιβλιοθήκης (library management).

Πραγματοποιήθηκε μια αναζήτηση και μελέτη των έτοιμων εφαρμογών που αναφέρονται στο Παράρτημα.

Η μελέτη αυτή μας κατεύθυνε την επιλογή μιας περιορισμένης λειτουργικότητας, που προδιαγράφεται στις παρακάτω απαιτήσεις λογισμικού.

Απαιτήσεις Λογισμικού

Χαρακτηριστικά της εφαρμογής

Η εφαρμογή:

- αφορά τμήμα συστήματος διαχείρισης βιβλιοθήκης
- έχει τρεις βασικές έννοιες-δεδομένα: βιβλίο, δάνειο, χρήστης
- κάνει CRUD πάνω στα δεδομένα αυτά

Από άποψη δομής/τεχνολογίας:

- Είναι εφαρμογή ιστού με βάση δεδομένων
- Χρησιμοποιεί τις εξής βασικές τεχνολογίες Java EE: JSF, CDI, EJB, JPA, authentication kai authorization

Επιδεικνύει την χρήση των εξής ειδικών τεχνολογιών και τεχνικών:

- PrimeFaces
- JSF Localization
- JSF Templates
- GlassFish Server authentication και authorization

Κατηγορίες Χρηστών

Οι χρήστες (δράστες) της εφαρμογής θα είναι οι εξής ανθρώπινοι ρόλοι:

1. Επισκέπτης – μη συνδεδεμένος χρήστης

2. Μέλος (member) – εγγεγραμμένος χρήστης

3. Βιβλιοθηκάριος (librarian) – υπάλληλος της βιβλιοθήκης

4. Διαχειριστής (admin) – υπάλληλος της βιβλιοθήκης με επιπλέον δικαιώματα

Οι χρήστες ορίζονται με βάση τα δικαιώματα εκτέλεσης περιπτώσεων χρήσης, όπως αναφέρονται στον παρακάτω Πίνακα. Τα δικαιώματα μέλους, βιβλιοθηκάριου και διαχειριστή αφορούν συνδεδεμένο χρήστη (προϋποθέτουν σύνδεση).

Το μέλος έχει υπερσύνολο των δικαιωμάτων επισκέπτη.

Ο βιβλιοθηκάριος έχει υπερσύνολο των δικαιωμάτων μέλους.

Ο διαχειριστής έχει υπερσύνολο των δικαιωμάτων βιβλιοθηκάριου.

Πίνακας Δικαιωμάτων Χρηστών

ΔΙΚΑΙΩΜΑΤΑ ΧΡΗΣΤΩΝ		**ΡΟΛΟΙ**			
		Επισκέπτης	**Μέλος**	**Βιβλιοθηκάριος**	**Διαχειριστής**
ΛΕΙΤΟΥΡΓΙΕ	Επιλογή Γλώσσας	✓	✓	✓	✓
	Προβολή Καταλόγου Βιβλίων	✓	✓	✓	✓
	Προβολή Πληροφοριών Βιβλίου	✓	✓	✓	✓
	Αναζήτηση Βιβλίου	✓	✓	✓	✓

	Προβολή Δανεισμένων Βιβλίων και Ιστορικό Δανεισμού	×	✓	✓	✓
	Δημιουργία Βιβλίου	×	×	✓	✓
	Επεξεργασία Βιβλίου	×	×	✓	✓
	Διαγραφή Βιβλίου	×	×	×	✓
	Προβολή Δανείων	×	×	✓	✓
	Δημιουργία Δανείου	×	×	✓	✓
	Επιστροφή Βιβλίου	×	×	✓	✓
	Διαγραφή Δανείου	×	×	×	✓
	Προβολή Χρηστών	×	×	✓	✓
	Δημιουργία Χρήστη	×	×	✓	✓
	Επεξεργασία Χρήστη	×	×	✓	✓
	Επεξεργασία Ρόλου Χρήστη	×	×	×	✓
	Διαγραφή Χρήστη	×	×	×	✓
	Επεξεργασία Ρυθμίσεων Συστήματος	×	×	×	✓
	Σύνδεση	✓	×	×	×
	Αποσύνδεση	×	✓	✓	✓

Μη Λειτουργικές Απαιτήσεις

Ταυτότητα		**Απαίτηση**
μλα.χρήση		Το λογισμικό θα είναι μια εφαρμογή ιστού.
μλα.δεδομένα.μονιμότητα		Η αποθήκευση των μόνιμων δεδομένων θα γίνεται σε σχεσιακή βάση δεδομένων Oracle.
μλα.γλώσσα		Το λογισμικό χρήστη θα είναι στην Ελληνική και Αγγλική γλώσσα.
μλα.πλατφόρμαΕκτέλεσης	.ΒάσηΔεδομένων	Oracle Database 11g Express Edition
	.Java	• JDK • GlassFish Server 5.0 • Java EE 7 • JSF 2.3 • PrimeFaces 10.0.0
Μλα.πλατφόρμαΑνάπτυξης	.ΒάσηΔεδομένων	Oracle SQL Developer v.20.4.1
	.Java	Apache Netbeans IDE 12.1

Περιπτώσεις Χρήσης

Οι περιπτώσεις χρήσης περιγράφουν τι θα μπορεί να κάνει ο κάθε χρήστης μέσω της εφαρμογής. Οι παρακάτω περιπτώσεις χρήσης δημιουργήθηκαν με βάση τους στόχους της εφαρμογής για τη λειτουργικότητα που θα υπάρχει στην πρώτη έκδοση.

Δράστες	**Ταυτότητα**	**Όνομα**
Επισκέπτης Μέλος Βιβλιοθηκάριος Διαχειριστής	πχ.βιβλίο.λίστα.προβολή	Προβολή Καταλόγου Βιβλίων
	πχ.βιβλίο.λίστα.αναζήτηση	Αναζήτηση Βιβλίου
	πχ.βιβλίο.πληροφορίες	Προβολή Πληροφοριών Βιβλίου
	πχ.επιλογήΓλώσσας	Επιλογή Γλώσσας
	πχ.λήξη_συνεδρίας	Λήξη Συνεδρίας
Μέλος Βιβλιοθηκάριος Διαχειριστής	πχ.αποσύνδεση	Αποσύνδεση Χρήστη
	πχ.χρήστης.δάνεια	Προβολή Δανείων Χρήστη
Βιβλιοθηκάριος Διαχειριστής	πχ.βιβλίο.δημιουργία	Εισαγωγή Νέου Βιβλίου
	πχ.βιβλίο.επεξεργασία	Επεξεργασία Βιβλίου
	πχ.δάνειο.προβολή_λίστας	Προβολή Λίστας Δανείων
	πχ.δάνειο.δημιουργία	Δημιουργία Δανείου
	πχ.δάνειο.επιστροφή	Επιστροφή Βιβλίου
	πχ.χρήστης.προβολή_λίστας	Προβολή Χρηστών
	πχ.χρήστης.δημιουργία	Δημιουργία Χρήστη
	πχ.χρήστης.επεξεργασία	Επεξεργασία Χρήστη
Διαχειριστής	πχ.δάνειο.διαγραφή	Διαγραφή Δανείου
	πχ.χρήστης.ρόλος	Επεξεργασία Ρόλου Χρήστη
	πχ.βιβλίο.διαγραφή	Διαγραφή Βιβλίου
	πχ.χρήστης.διαγραφή	Διαγραφή Χρήστη
	πχ.ρυθμίσεις	Ρυθμίσεις Συστήματος
Εγγεγραμμένος Επισκέπτης	πχ.σύνδεση	Σύνδεση Χρήστη

Λεξικό δεδομένων

Καταγράφουμε εδώ τα βασικά δεδομένα των περιπτώσεων χρήσης. Το λεξικό δεδομένων θα υποστηρίξει

- τον λογικό σχεδιασμό της βάσης δεδομένων
- και την δημιουργία του μοντέλου κλάσεων οντοτήτων της κυρίως εφαρμογής.

Υπάρχουν τα εξής 3 σύνθετα (πολυδιάστατα) δεδομένα:

Χρήστης = όνομα

+ επώνυμο

+ ρόλος (τιμή από το σύνολο: επισκέπτης, μέλος, βιβλιοθηκάριος, διαχειριστής)

+ email (username)

+ password

Βιβλίο =

τίτλος

+ φωτογραφία εξώφυλλου

+ συγγραφείς (από ένας έως πολλοί, διατεταγμένοι: 1ος, 2ος, 3ος κλπ)

+ αντίγραφα (αριθμός αντιγράφων, απόθεμα)

Δάνειο =

βιβλίο

+ χρήστης

+ ημερομηνία δανεισμού

+ ημερομηνία επιστροφής

Οι διαστάσεις των δεδομένων αυτών περιγράφονται στην συνέχεια:

Συγγραφέας

= ονοματεπώνυμο

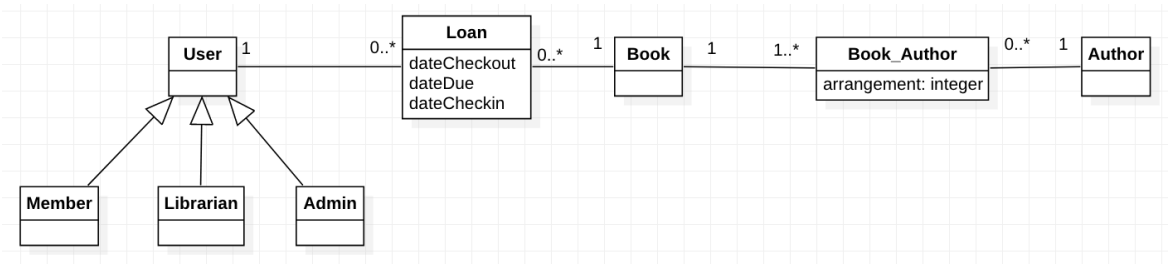
Ανάπτυξη της βάσης δεδομένων

Ανάλυση και λογικός σχεδιασμός της βάσης δεδομένων

Τα τρία σύνθετα δεδομένα στο λεξικό δεδομένων μοντελοποιούνται με οντότητες γιατί έχουν διαστάσεις χαρακτηριστικά, βλ. παρακάτω διάγραμμα οντοτήτων-συσχετίσεων.

Οι συγγραφείς έχουν μια δυαδική συσχέτιση πολλά προς πολλά με τα βιβλία που κανονικοποιείται με χρήση ενδιάμεσης οντότητας. Στην ενδιάμεση οντότητα, κωδικοποιείται και η τάξη (σειρά) εμφάνισης κάθε συγγραφέα (1ος, 2ος, 3ος κλπ) για βιβλία με πολλούς συγγραφείς.

Οι τρεις υποκατηγορίες χρηστών (τρεις γενικεύσεις) που παριστάνονται στην βάση δεδομένων είναι τα μέλη, οι βιβλιοθηκάριοι, και οι διαχειριστές. Με βάση τις απαιτήσεις λογισμικού, θα μπορούσε να υπάρχει μια ιεραρχία γενίκευσης με επίπεδα (από τον χρήστη-μέλος στον βιβλιοθηκάριο και από τον βιβλιοθηκάριο στον διαχειριστή) αλλά προτιμήσαμε εδώ την απλούστερη παράσταση που είναι πλησιέστερη και στην υλοποίηση, βλ. συνέχεια.



Εικόνα 13. Διάγραμμα E/R δεδομένων

Υλοποίηση της βάσης δεδομένων

Μετατρέπουμε το μοντέλο E/R σε SQL-DDL και συζητάμε τις τεχνικές προσθήκες.

Σχετικά με τις γενικεύσεις για τους χρήστες: Υπάρχουν οι γενικές τρεις δυνατότητες μετατροπής σε SQL: με έναν πίνακα για κάθε έννοια, με έναν πίνακα-υπερτύπο και έναν πίνακα για κάθε έννοια-υποτύπο, ή με έναν πίνακα με στήλες με τον σύνολο των χαρακτηριστικών για τους τρείς υποτύπους και ειδική στήλη για την διάκριση των υποτύπων, βλ. Teorey et al. (2011), Chapter 5, Generalization and Aggregation. Ακολουθούμε την τελευταία δυνατότητα γιατί δίνει ευκολότερα διαχειρίσιμη βάση δεδομένων και αντίστοιχη κλάση οντότητας στην Java.

Κάθε οντότητα δίνει ένα ομώνυμο πίνακα. Για τον πίνακα χρηστών χρησιμοποιούμε το όνομα users στον πληθυντικό γιατί ο ενικός είναι δεσμευμένη λέξη της Oracle. Η στήλη users.role διαχωρίζει τους τρεις υποτύπους της γενίκευσης και έχει πεδίο ορισμού {'member', 'librarian', 'admin'} με περιορισμό τιμών (check constraint).

Οι συσχετίσεις είναι ένα προς πολλά και δίνουν ξένα κλειδιά στην μεριά του πολλά. Η βάση που προκύπτει είναι τουλάχιστον σε 3η κανονική μορφή.

Για πρωτεύον κλειδί, χρησιμοποιούμε μια τεχτητή στήλη id σε όλους τους πίνακες, εκτός από τον ενδιάμεσο πίνακα book_author όπου χρησιμοποιούμε τον συνδυασμό των δύο ξένων κλειδιών.

Η τεχνητή στήλη book.availableCopies είναι υπολογιζόμενη, δίνει το διαθέσιμα απόθεμα βιβλίων (αριθμός διαθέσιμων προς δανεισμό αντιγράφων) και αποσκοπεί στην απλοποίηση της υλοποίησης της περίπτωση χρήσης δανεισμού? Ο υπολογισμός της τιμής γίνεται σε κάθε δανεισμό και επιστροφή βιβλίου με αντίστοιχη μείωση ή αύξηση της τιμής.

Για την αυτόματη δημιουργία τιμής πρωτεύοντος κλειδιού κατά την εισαγωγή (insert), για κάθε τεχνητό πρωτεύον κλειδί δημιουργούμε μια ακολουθία (sequence) και ένα trigger (before insert for each row).

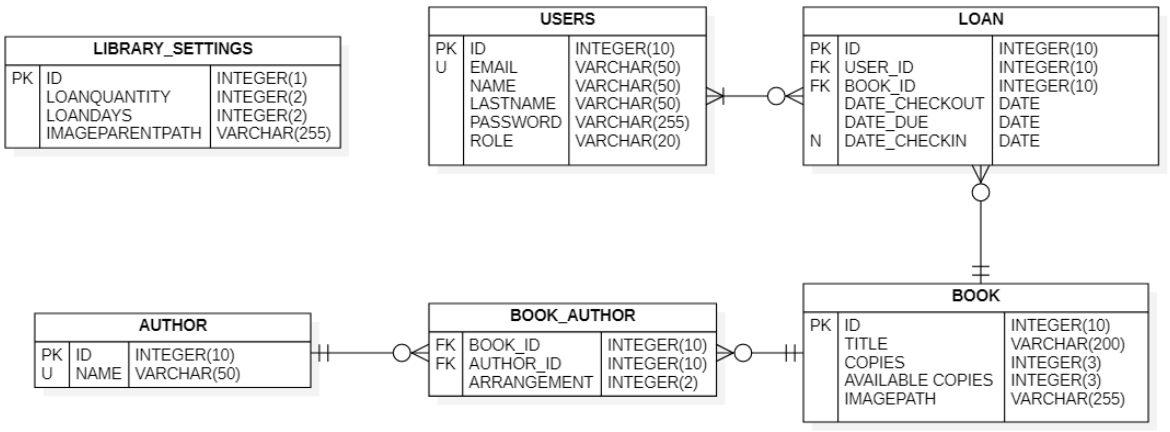
Επιλέξαμε να αποθηκεύουμε τις εικόνες των εξωφύλλων των βιβλίων εκτός βάσης δεδομένων στο σύστημα αρχείων. Η στήλη book.imagepath δίνει την διαδρομή του αρχείου.

Οι στήλες όλων των πινάκων είναι μη κενές (not null) εκτός την loan#date_checkin.

Η υλοποίηση συνοψίζεται στο παρακάτω διάγραμμα πινάκων βάσης δεδομένων. Ο αντίστοιχος κώδικας SQL-DDL δίνεται στο https://github.com/dtsimaras/MyLMS. Στο διάγραμμα, οι τύποι των δεδομένων είναι εδώ γενικοί τύποι SQL. Στον κώδικα, οι τύποι των δεδομένων είναι γενικοί ή στην διάλεκτο της Oracle.

Ο πρόσθετος πίνακας library_settings έχει μια εγγραφή και αποθηκεύει γενικές παραμέτρους της εφαρμογής στις εξής στήλες:

- loandays: μέγιστος αριθμός ημερών δανεισμού
- loanquantity: μέγιστος αριθμός βιβλίων που μπορεί να δανειστεί ένας χρήστης.
- Imageparentpath: διαδρομή εικόνων στο file system που δίνει ευχέρεια μετακίνησης αν χρειαστεί.



Εικόνα 14. Πίνακες Βάσης Δεδομένων

Ανάπτυξη της κυρίως εφαρμογής σε Java EE

Γενική Αρχιτεκτονική και Σχεδιασμός

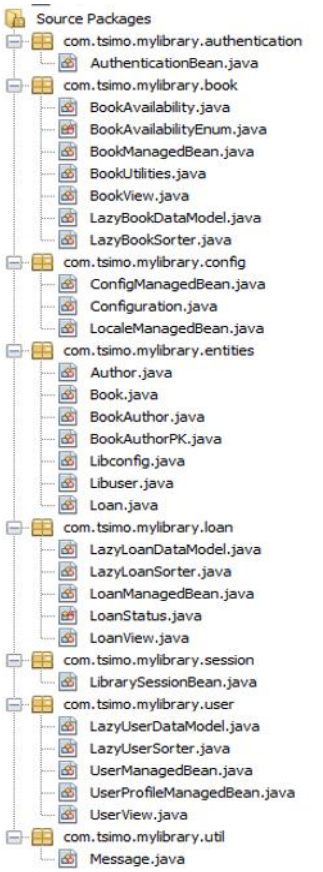
Η εφαρμογή ακολουθεί την εξής γενική αρχιτεκτονική με τις αντίστοιχες τεχνολογίες Java EE:

- Επίπεδο μόνιμων δεδομένων: αντικειμενοστραφές μοντέλο των μόνιμων δεδομένων με κλάσεις οντοτήτων (entity classes), με τεχνολογία JPA (Java Persistence). Οι κλάσεις αντιστοιχούν στους πίνακες της βάσης δεδομένων (σχεσιακό μοντέλο). Η JPA υλοποιεί το ORM (object-relational mapping). Χρησιμοποήθηκε η υλοποίηση EclipseLink της JPA.
- Επίπεδο βαθιάς επιχειρησιακής λογικής: κλάσεις με τεχνολογία EJB SSB (Enterprise JavaBeans, stateless session beans), που συνεργάζονται με την JPA, και διαχειρίζονται την επικοινωνία με την βάση δεδομένων (CRUD δεδομένων).
- Επίπεδο ιστού: JSF views (JavaServer Faces views) για τον ορισμό των σελίδων (views) στην πλευρά του διακομιστή με αρχεία xml/Facelets, και κλάσεις Java με τεχνολογία CDI που υποστηρίζουν τα views και γίνονται πελάτες της βαθειάς επιχειρησιακής λογικής, χρησιμοποιώντας και τις JPA entity classes.

Η πρόσβαση στο επίπεδο του ιστού (σελίδες) ελέγχεται με αυθεντικοποίηση και εξουσιοδότηση μέσω GlassFish Server.

Οργάνωση της εφαρμογής σε Packages

Σύμφωνα με την σύμβαση για τα ονόματα των πακέτων, τα πακέτα αρχίζουν με το όνομα της σελίδας (domain) σε αντίστροφη σειρά. Αυτό βοηθάει να έχουν μοναδικά ονόματα τα πακέτα και έτσι αποτρέπεται η σύγκρουση στα ονόματα τους. Εγώ με βάση το υποθετικό όνομα του domain μου mylibary.tsimo.com δημιούργησα πακέτα που με όνομα com.tsimo.mylibrary, έτσι το όνομα του πακέτου που έχω για τις κλάσεις entity ονομάζεται com.tsimo.mylibary.entities. Για συντομία παρακάτω θα παραλείπω το πρώτο κομμάτι και θα αναφέρομαι στο πακέτο ως απλά πακέτο entities και αντίστοιχα και για τα υπόλοιπα.Για την εφαρμογή αυτή δημιούργησα τα παρακάτω πακέτα:



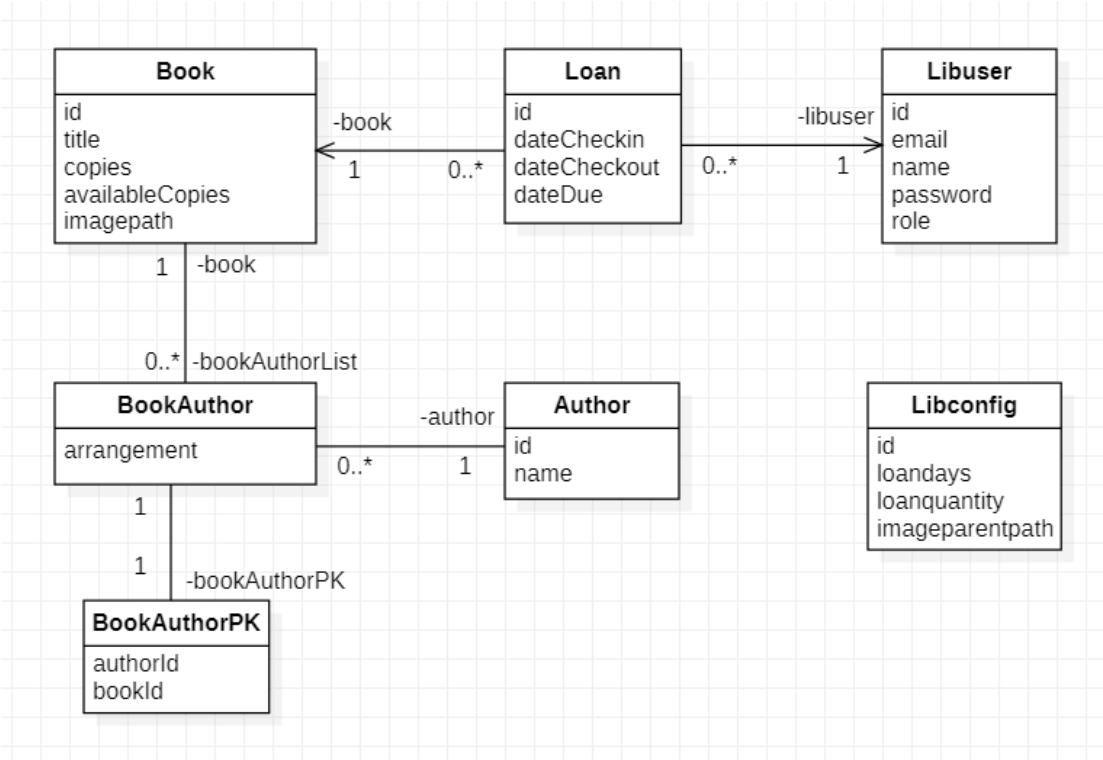
Εικόνα 15. Πακέτα και Κλάσεις της εφαρμογής

- **entities:** περιλαμβάνει όλες τις κλάσεις Entity της εφαρμογής.
- **session:** περιλαμβάνει την κλάση LibrarySessionBean η οποία κλάση χρησιμοποιείται από πολλές κλάσεις για να επικοινωνούν με την βάση δεδομένων.
- **authentication:** περιλαμβάνει την κλάση AuthenticationBean για την αυθεντικοποίηση του εξουσιοδότηση του χρήστη.
- **book:** περιλαμβάνει τις κύριες κλάσεις που χρειάζονται για το CRUD των βιβλίων.
- **loan:** περιλαμβάνει τις κύριες κλάσεις που χρειάζονται για το CRUD των δανείων.
- **user:** περιλαμβάνει τις κύριες κλάσεις που χρειάζονται για το CRUD των χρηστών. Επίσης περιλαμβάνει την κλάση UserProfileManagedBean η οποία χρησιμοποιείται ως managed bean για την σελίδα στην οποία βλέπει ο χρήστης τα βιβλία που έχει δανειστεί.
- **config:** περιλαμβάνει κλάσεις και managed bean για την σελίδα στην οποία ρυθμίζονται κάποιες παράμετροι τις βιβλιοθήκης, του πίνακα library_settings.
- **util:** Περιλαμβάνει την κλάση Message η οποία είναι μια βοηθητική κλάση η οποία χρησιμοποιείται σε όλη την εφαρμογή για να εμφανίζονται μηνύματα.

Δημιουργία Κλάσεων Οντοτήτων JPA

Στην κυρίως εφαρμογή Java, τα δεδομένα που αντιστοιχούν σε πίνακες της βάσης δεδομένων μοντελοποιούνται με κλάσεις οντοτήτων (Entity classes). Θα χρησιμοποιήσουμε την τεχνολογία JPA (Java Persistence API) που προσφέρει ένα ψηλού επιπέδου χειρισμό του ORM (object-relationship mapping).

Οι κλάσεις οντοτήτων έχουν αντιστοιχία 1:1 με τους πίνακες εκτός από την κλάση BookAuthorPK η οποία μοντελοποιεί το σύνθετο πρωτεύον κλειδί του πίνακα BookAuthor.



Εικόνα 16. Διάγραμμα κλάσεων UML με κλάσεις οντοτήτων

Κατά την ημιαυτόματη δημιουργία των κλάσεων οντοτήτων με EclipseLink, δημιουργούνται σε κάθε κλάση οντότητας:

- getter/setter για κάθε πεδίο που ορίζουν μια αντίστοιχη ιδιότητα (Java property) για ανάγνωση και εγγραφή. Μερικές από τις ιδιότητες αυτές εκτίθενται σε JPA views.
- JPQL named queries.

Σύνοψη του σχεδιασμού ανά περίπτωση χρήσης

Κώδικας εφαρμογής διαθέσιμος στο https://github.com/dtsimaras/MyLMS

Ρόλος χρήστη	Περίπτωση χρήσης	Σελίδες και βασικά tags	Βασικές κλάσεις JSF backing beans και μέθοδοι	Συμπληρωματικές κλάσεις JSF backing beans (ίσως και μέθοδοι)	Κλάσεις session beans και μέθοδοι	Κλάσεις JPA
Επισκέπτης Μέλος Βιβλιοθηκάριος Διαχειριστής	πχ.βιβλίο.λίστα.προβολή	Index • p:button header • p:menuitem books • p:dataTable	BookView • init() • getLazyModel() • getAvailabilityValues()	BookAvailability BookAvailabilityEnum LazyBookDataModel LazyBookSorter AuthenticationBean	LibrarySessionBean • findAllBooks()	Book BookAuthor BookAuthorPK Author
	πχ.βιβλίο.λίστα.αναζήτηση	books • p:dataTable	BookView • getLazyModel() • getAvailabilityValues() LazyBookDataModel • load()	BookAvailability BookAvailabilityEnum LazyBookSorter	-	Book BookAuthor BookAuthorPK Author
	πχ.βιβλίο.πληροφορίες	books • p:dataTable • p:ajax • p:dialog	BookView • getSelectedBook()	-	-	Book
	πχ.γλώσσα	footer locale • p:selectOneMenu	LocaleManagedBean • init() • getLanguage() • setLanguage()	-	-	-
	πχ.λήξη_συνεδρίας	sessionTimeout • p:idleMonitor • p:dialog	-	-	-	-
Μέλος Βιβλιοθηκάριος Διαχειριστής	πχ.αποσύνδεση	header • p:menuitem	AuthenticationBean • logout() • getRequest()	Message	-	-
	πχ.χρήστης.δάνεια	header • p:menuitem profile • p:dataTable	UserProfileManagedBean • init() • getUsername() • getOpenLoans() • getClosedLoans()	Message	LibrarySessionBean • findUseByEmail() • findUserOpenLoans() • findUserClosedLoans()	Libuser Loan
Βιβλιοθηκάριος Διαχειριστής	πχ.βιβλίο.δημιουργία	header • p:menuitem books • p:toolbar • p:linkButton createBook • h:form • p:fileUpload • p:autoComplete	BookManagedBean • init() • getBook() • setBook() • getAuthorNames() • setAuthorNames() • getFile() • setFile() • completeText() • createBook() BookUtilities • createBookAuthors()	Configuration Message AuthenticationBean	LibrarySessionBean • createAndGetBook() • findAllAuthors() • deleteBookAuthors() • create() • update()	Author Book BookAuthor BookAuthorPK
	πχ.βιβλίο.επεξεργασία	books • p:dataTable • p:commandButton • f:setPropertyActionListener • p:dialog	BookView • getSelectedBook() • setSelectedBook() • getAuthorNames() • setAuthorNames() • getFile() • setFile() • updateBook() BookUtilities • createBookAuthors()	Configuration Message AuthenticationBean	LibrarySessionBean • findAllAuthors() • deleteBookAuthors() • create() • update()	Author Book BookAuthor BookAuthorPK
	πχ.δάνειο.προβολή_λίστας	header • p:menuitem loans • p:dataTable	LoanView • init() • getLazyModel() • getStatusValues()	LoanStatus LazyLoanDataModel LazyLoanSorter AuthenticationBean	LibrarySessionBean • findAllLoans()	Loan Book
	πχ.δάνειο.δημιουργία	header • p:menuitem loans • p:linkButton createLoan • h:form	LoanManagedBean • init() • getEmail() • setEmail() • getBookId() • setBookId()	Configuration Message AuthenticationBean	LibrarySessionBean • findUserByEmail() • findBookById() • findUserOpenLoans() • create() • update()	Loan Libuser Book

			• createLoan()			
	πχ.δάνειο.επιστροφή	loans • p:dataTable • p:ajax • p:dialog	LoanView • getSelectedLoan() • setSelectedLoan() • updateLoanStatus()	Message AuthenticationBean	LibrarySessionBean() • update()	Loan Book
	πχ.χρήστης.προβολή_λίστας	header • p:menuitem users • p:dataTable	UserView • init() • getLazyDataModel()	LazyUserDataModel LazyUserSorter AuthenticationBean	LibrarySessionBean • findAllUsers()	Libuser
	πχ.χρήστης.δημιουργία	header • p:menuitem users • p:linkButton createUser • h:form • p:password	UserManagedBean • init() • getUser() • setUser() • getVerifyPassword() • setVerifyPassword() • hashPassword() • createUser()	Message	LibrarySessionBean • create()	Libuser
	πχ.χρήστης.επεξεργασία	users • p:dataTable • p:ajax • p:dialog	UserView • getSelectedUser() • setSelectedUser() • hashPassword() • updateUser()	AuthenticationBean Message	LibrarySessionBean • update()	Libuser
Διαχειριστής	πχ.χρήστης.ρόλος	users • p:dataTable • p:ajax • p:dialog	UserView • getSelectedUser() • setSelectedUser() • hashPassword() • updateUser()	AuthenticationBean Message	LibrarySessionBean • update()	Libuser
	πχ.βιβλίο.διαγραφή	books • p:dataTable • p:commandButton • f:setPropertyActionListener • p:confirmDialog	BookView • getSelectedBook() • deleteBook()	AuthenticationBean Message	LibrarySessionBean • openLoanExists() • delete()	Book
	πχ.δάνειο.διαγραφή	loans • p:dataTable • p:commandButton • f:setPropertyActionListener • p:confirmDialog	LoanView • getSelectedLoan() • deleteLoan()	AuthenticationBean Message	LibrarySessionBean • delete()	Loan
	πχ.χρήστης.διαγραφή	users • p:dataTable • p:commandButton • f:setPropertyActionListener • p:confirmDialog	UserView • getSelectedUser() • deleteUser()	AuthenticationBean Message	LibrarySessionBean • findUserOpenLoans() delete()	Libuser
	πχ.ρυθμίσεις	libConfiguration • h:form	ConfigManagedBean • init() • getLibConfig() • setLibConfig() • updateLibConfig()	Configuration	LibrarySessionBean • getLibConfig() • update()	Libconfig
Επισκέπτης	πχ.σύνδεση	login • h:form	AuthenticationBean • getUsername() • setUsername() • getPassword() • setPassword() • getRequest() • login()	Message	-	-

1. Προβολή καταλόγου βιβλίων

Δίνουμε ένα αναλυτικό παράδειγμα ανάλυσης/σχεδιασμού και υλοποίησης της προβολής του καταλόγου των βιβλίων που αντιστοιχεί στις παρακάτω απαιτήσεις λογισμικού. Η περίπτωση χρήσης είναι τμήμα του CRUD βιβλίων. Παρόμοιος σχεδιασμός χρησιμοποιείται και για την προβολή των χρηστών και των δανείων.

Υπενθυμίζουμε ότι υπάρχουν οι εξής 6 περιπτώσεις χρήσεις σχετικές με τα βιβλία:

Η προβολή του καταλόγου είναι ξεχωριστή περίπτωση χρήσης, αλλά συμπεριλαμβάνεται επίσης στις περιπτώσεις χρήσης αναζήτηση, ταξινόμηση.

Ταυτότητα Περίπτωσης Χρήσης			**Όνομα**
πχ.βιβλίο	.λίστα	.προβολή	Προβολή Καταλόγου Βιβλίων
		.αναζήτηση	Αναζήτηση Βιβλίων
		.ταξινόμηση	Ταξινόμηση βιβλίων
	.πληροφορίες		Προβολή Πληροφοριών Βιβλίου
	.δημιουργία		Εισαγωγή Νέου Βιβλίου
	.επεξεργασία		Επεξεργασία Βιβλίου
	.διαγραφή		Διαγραφή Βιβλίου

Υπενθύμιση απαιτήσεων λογισμικού

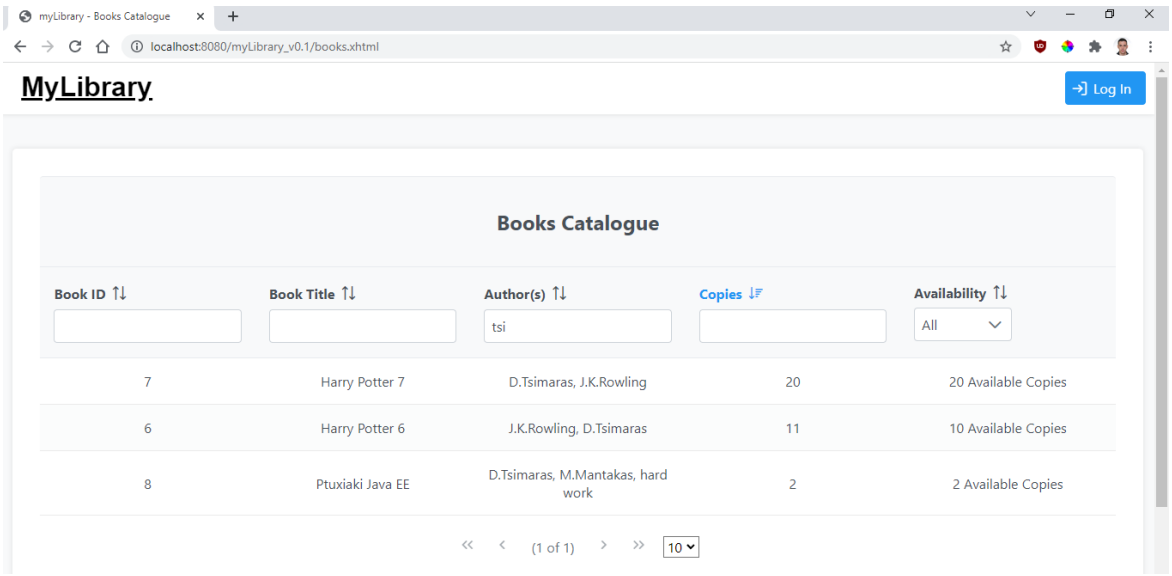
Περίπτωση χρήσης πχ.βιβλίο.λίστα.προβολή - Προβολή του καταλόγου των βιβλίων*. Η περίπτωση χρήσης αφορά όλους τους δράστες.

*Κατάλογος βιβλίων: Για κάθε βιβλίο, κωδικός βιβλίου, τίτλος, συγγραφείς, αριθμός διαθέσιμων αντιγράφων, και δεδομένα διαθεσιμότητας**. Θα υποστηρίζει σελιδοποίηση.

** Στην προβολή της διαθεσιμότητας βιβλίου, θα εμφανίζεται ο τρέχων διαθέσιμος αριθμός αντιγράφων, εκτός κι αν όλα τα αντίγραφα είναι δανεισμένα, οπότε θα εμφανίζεται η πρώτη ημερομηνία αναμενόμενης επιστροφής αντιγράφου. Ειδικότερα, τα μηνύματα θα είναι τα εξής:

- Αν υπάρχει μόνο ένα διαθέσιμο αντίγραφο: «1 Available Copy»
- Αν υπάρχουν παραπάνω εμφανίζεται ο αριθμός των βιβλίων ακολουθημένο από το String: « Available Copies»
- Αν δεν υπάρχει διαθέσιμο αντίγραφο εμφανίζεται το μήνυμα «Expected Return: » ακολουθημένο από την πιο κοντινή ημερομηνία που περιμένουμε να επιστραφεί κάποιο αντίγραφο

Σε άλλη περίπτωση, όπως αν δεν υπάρχουν καθόλου αντίγραφα ή αν η ημερομηνία που περιμένουμε να επιστραφούν τα αντίγραφα ενός βιβλίου έχει περάσει εμφανίζεται το μήνυμα «Not Available»



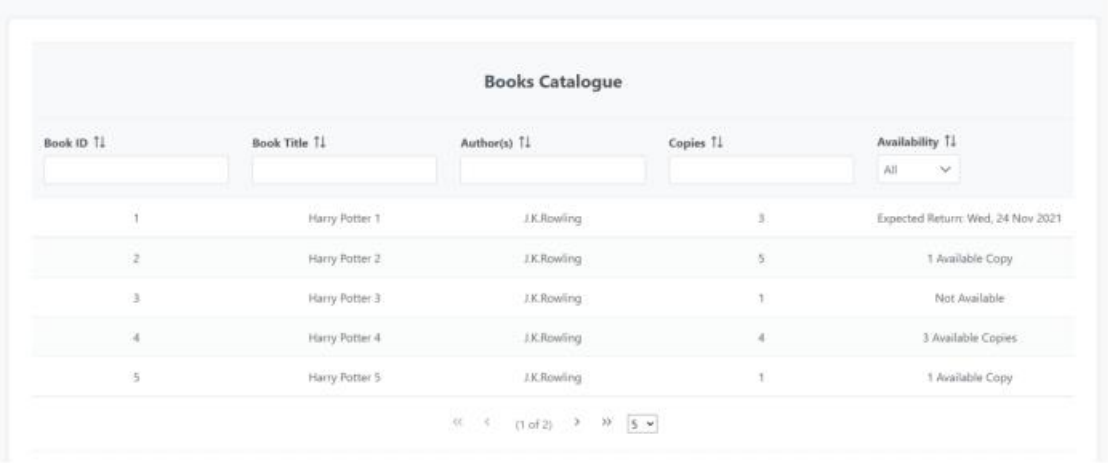
Εικόνα 17. Κατάλογος Βιβλίων με κριτήριο αναζήτησης και ταξινομημένα ανά αριθμό αντιγράφων.

Ανάλυση και Σχεδιασμός

Διεπαφή χρήστη

JSF view books.xhtml με βασική συνιστώσα PrimeFaces dataTable. Η συνιστώσα αυτή επελέγη γιατί προσφέρει δυνατότητες σελιδοποίησης, ταξινόμησης, φιλτραρίσματος (αναζήτησης) και τεμπέλικης φόρτωσης.

Αντί της περιγραφής του JSF view, παραθέτουμε ένα παράδειγμα απόδοσης της διεπαφής στον browser. Το παράδειγμα απεικονίζει και την δυνατότητα ταξινόμησης και αναζήτησης σε κάθε στήλη.



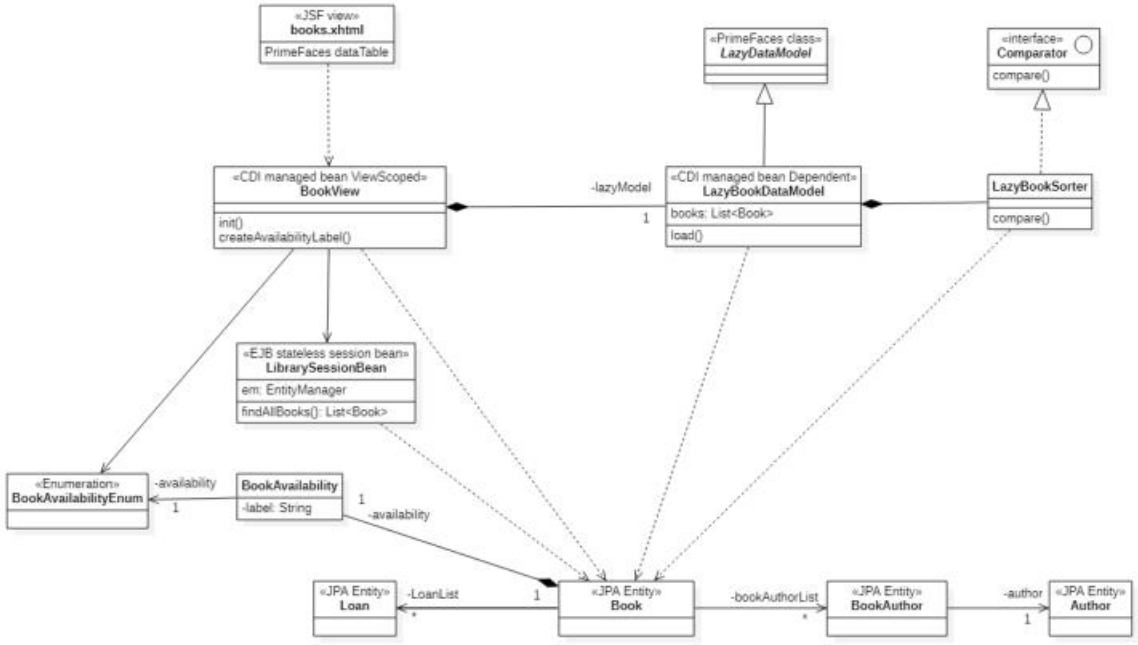
Εικόνα 18. Παράδειγμα απόδοσης ιστοσελίδας καταλόγου βιβλίων.

Αρχιτεκτονική
Η JSF view και οι τρεις πρώτες κλάσεις ανήκουν στο επίπεδο του ιστού.

- **books.xhtml:** JSF view με συνιστώσα PrimeFaces dataTable για την προβολή του καταλόγου βιβλίων, με σελιδοποίηση και δυνατότητα ταξινόμησης (sorting).
- **BookView:** CDI-managed bean ViewScoped* για την υποστήριξη του JSF view.
- **LazyBookDataModel** CDI-managed bean με scope Dependent για την υλοποίηση της σελιδοποίησης και της τεμπέλικης φόρτωσης (lazy loading) μιας σελίδας του καταλόγου βιβλίων - φιλτράρει την λίστα των βιβλίων επιλέγοντας τα βιβλία για επιλεγμένη σελίδα. Υποκλάση της αφηρημένης κλάσης **LazyDataModel** των PrimeFaces.
- **LazyBookSorter:** CDI-managed bean με scope ? για τον διαχωρισμό της υλοποίησης της ταξινόμησης των βιβλίων. Προμηθευτής της LazyBookDataModel.
- **LibrarySessionBean:** EJB stateless session bean για την βαθιά επιχειρησιακή λογική της επικοινωνίας με την βάση δεδομένων - δημιουργία της πλήρους λίστας βιβλίων με χρήση μεθόδων του JPA entity manager.
- **Book:** κλάση τύπου JPA entity για την μοντελοποίηση των βιβλίων, αντίστοιχη του πίνακα Book της βάσης δεδομένων.

* JSF Scope View: Το backing bean BookView έχει scope view (ViewScoped). Ένα ViewScoped CDI-managed bean ζει όσο το JSF view που υποστηρίζει. Τα JSF views περιέχουν Ajax που δημιουργεί αιτήματα Ajax προς τον διακομιστή, π.χ. κατά την ταξινόμηση των στηλών ενός PrimeFaces dataTable. Το ViewScoped bean συνεχίζει να ζει μετά την απόκριση αιτημάτων Ajax γιατί δεν αλλάζει το view, σε αντίθεση με ένα RequestScoped bean που θα πέθαινε και ένα νέο θα το αντικαθιστούσε κατά την απόκριση. Αυτή η διατήρηση του backing bean μετά την απόκριση αιτημάτων Ajax είναι απαραίτητη για την λειτουργία της εφαρμογής.

Πηγή: https://stackoverflow.com/questions/6025998/difference-between-view-and-request-scope-in-managed-beans



Εικόνα 19. Σύνοψη της ανάλυσης για την περίπτωση χρήσης Προβολή καταλόγου βιβλίων. Διάγραμμα κλάσεων UML και JSF view

Διαθεσιμότητα βιβλίου

Η διαθεσιμότητα του βιβλίου, όπως εξηγήθηκε, έχει δύο υπολογιζόμενες συνιστώσες, την κατάσταση διαθεσιμότητας βιβλίου και μια ετικέτα για την εμφάνιση. Παριστάνεται με αντικείμενο της κλάσης BookAvailability. Με το αντικείμενο αυτό συνδέεται ο αντικείμενο βιβλίου με ένα πεδίο με annotation @ javax.persistence.Transient γιατί το πεδίο δεν παριστάνεται στην βάση δεδομένων. Η κατάσταση διαθεσιμότητας παίρνει τιμές από ένα σύνολο τριών τιμών που παριστάνεται με το enumeration BookAvailabilityEnum.

Ο υπολογισμός της διαθεσιμότητας (των δύο συνιστωσών) μπορεί να γίνεται για κάθε βιβλίο ή για τα βιβλία του τεμπέλικου μοντέλου, με βάση το απόθεμα (αριθμός αντιγράφων της Book) και τα δάνεια (Loan).

Κατά την αναζήτηση βιβλίου, ο χρήστης μπορεί να επιλέξει την κατάσταση διαθεσιμότητας βιβλίου από την διεπαφή χρήστη με ένα selectOneMenu.

Υλοποίηση

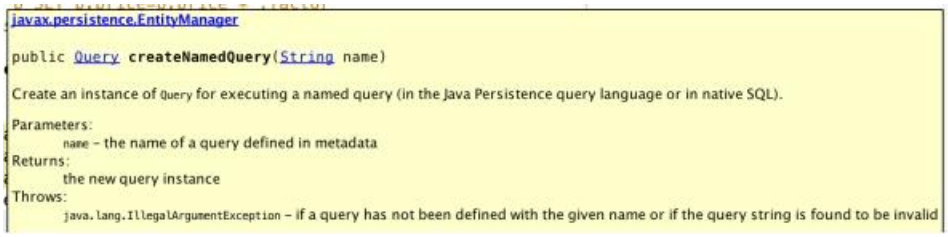
LibrarySessionBean

Η δομή δεδομένων των προϊόντων δημιουργείται με την μέθοδο findAllBooks σε ένα αντικείμενο της κλάσης stateless session bean LibrarySessionBean. Η δημιουργία στηρίζεται στην JPA, και την μέθοδο createNamedQuery() του entity manager.

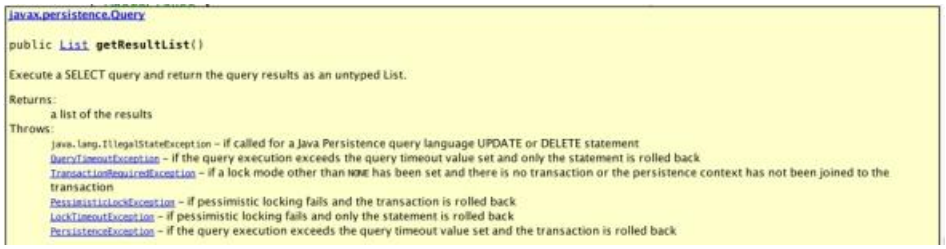
```
18  @Stateless
19  public class LibrarySessionBean {
20
21      @PersistenceContext(unitName = "com.tsimo_myLibrary_v0.1_war_0.1PU")
22      private EntityManager em;
23
24      public List<Book> findAllBooks() {
25          return em.createNamedQuery("Book.findAll").getResultList();
26      }
```

Εικόνα 20. LibrarySessionBean#findAllBooks()

Οι μέθοδοι EntityManager#createNamedQuery() και Query#getResultList() αίρουν εξαιρέσεις, όπως οι παρακάτω της getResultList(). Οι εξαιρέσεις αυτές θα πιαστούν στο αντικείμενο του πελάτη BookView, όπου προωθούνται ως εξαίρεση javax.ejb.EJBException.



Εικόνα 21. Εξαίρεση κατά την δημιουργία του Query



Εικόνα 22. Εξαίρεση κατά την επιστροφή της λίστας

BookView

Η LibrarySessionBean#findAllBooks() καλείται από την BookView#init(). Η init() εκτελείται μετά την δημιουργία του αντικειμένου της BookView και το injection της LibrarySessionBean. Η BookView είναι CDI-managed ViewScoped.

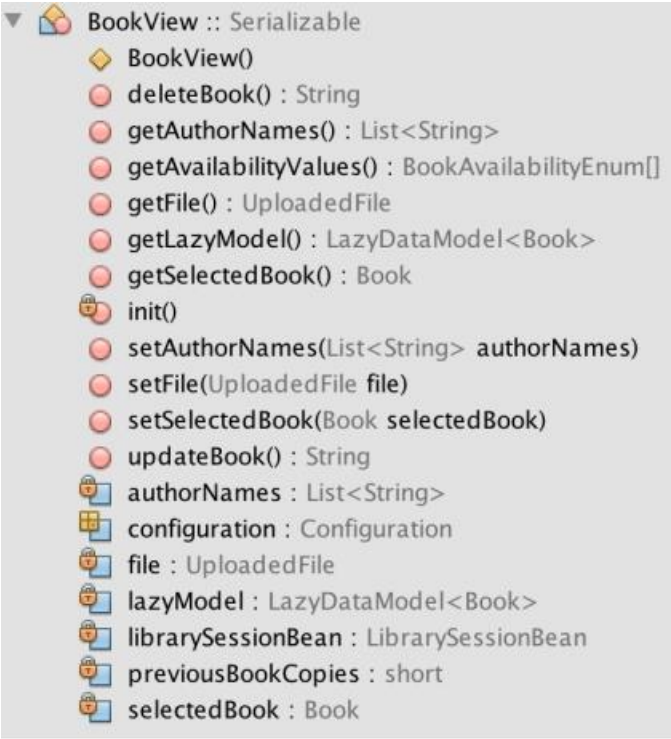
Το αντικείμενο της δημιουργείται από την JSF όταν χρειάζεται για το rendering του JSF view books.xhtml και ζει όσο ζει το JSF view.

Η init() δημιουργεί το αντικείμενο της LazyBookDataModel, περνώντας την αναφορά της λίστας των βιβλίων. Πιάνει και χειρίζεται πιθανές εξαιρέσεις από τον προμηθευτή session bean που φτάνουν ως EJBException.

```
21  import javax.inject.Named;
22  import javax.ejb.EJB;
23  import javax.ejb.EJBException;
24
25  @Named(value = "bookView")
26  @ViewScoped
27  public class BookView implements Serializable {
28
29      @EJB
30      private LibrarySessionBean librarySessionBean;
31
32      @PostConstruct
33      private void init()
34          {
35          try
36              {
37              lazyModel = new LazyBookDataModel(librarySessionBean.findAllBooks());
38              }
39          catch (EJBException e)
40              {
41              // χειρισμός σφάλματος
42              }
43          }
```

Εικόνα 23. CDI ManagedBean BookView

Επίσης, η BookView ορίζει και εκθέτει την ιδιότητα για ανάγνωση lazyModel στην JSF view books.xhtml:



Εικόνα 24. Μέθοδοι και ιδιότητες της BookView

LazyBookDataModel

Η LazyBookDataModel

- είναι υποκλάση της αφηρημένης LazyDataModel με παράμετρο Book
- υλοποιεί τις παρακάτω μεθόδους της αφηρημένης υπερκλάσης

Η μέθοδος load()

- δημιουργεί και επιστρέφει το τεμπέλικο μοντέλο, δηλαδή την λίστα με το υποσύνολο των βιβλίων που ικανοποιούν την σελιδοποίηση, αναζήτηση, και ταξινόμηση, με βάση την (πλήρη) λίστα των βιβλίων
- καλείται από τον κώδικα της PrimeFaces για την φόρτωση του καταλόγου των βιβλίων σε JSF view.
-

Εικόνα 25. Μέθοδοι και πεδία της κλάσης LazyBookDataModel

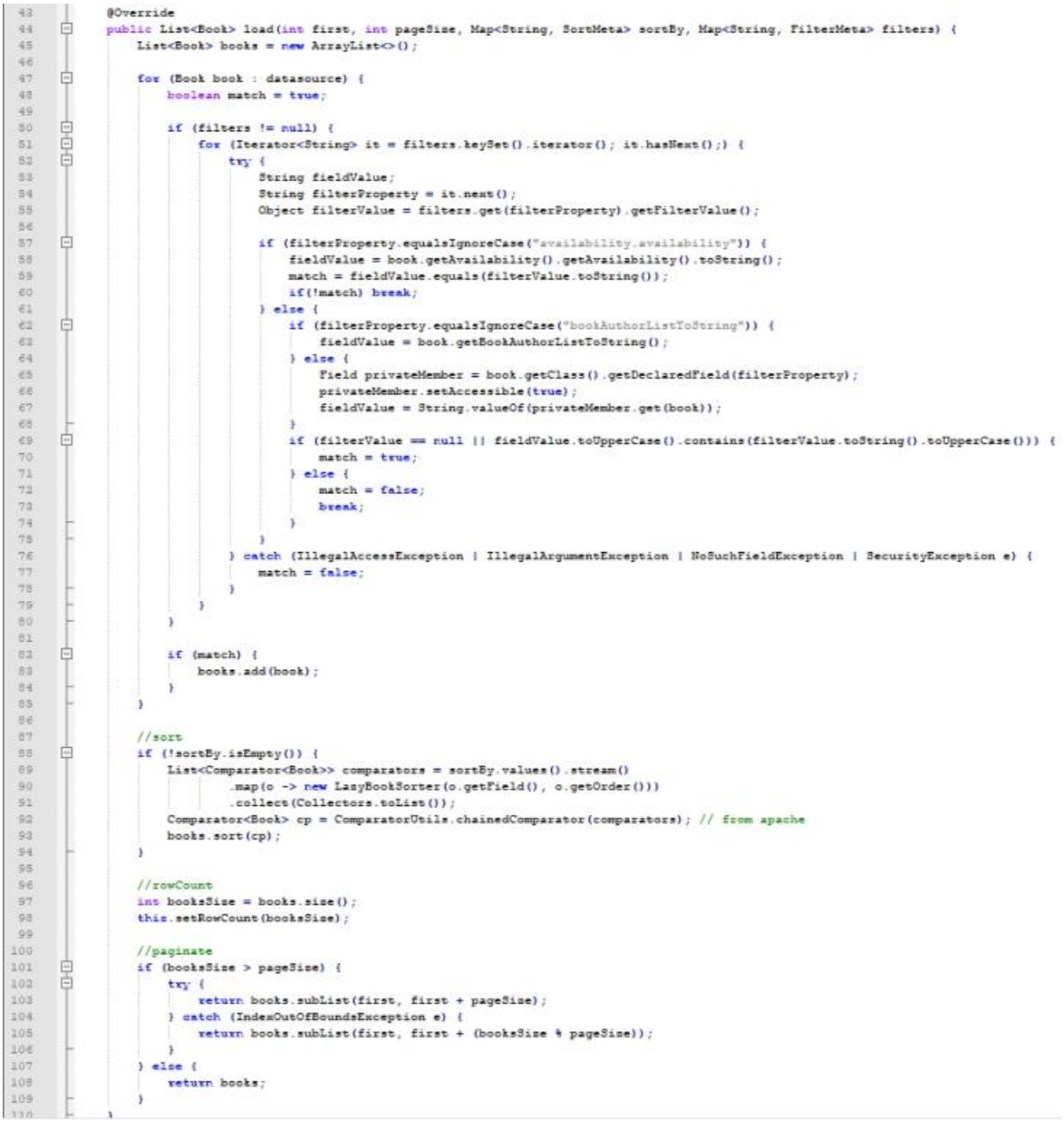
Σε κάθε ενέργεια του χρήστη για αναζήτηση, ταξινόμηση, αλλαγή σελίδας, το αίτημα υποβάλλεται ασύγχρονα (με Ajax) και καλείται η LazyBookDataModel#load(). Η μέθοδος υπολογίζει (ξανά) το lazy model, και επιστρέφει την λίστα βιβλίων.

Η υλοποίηση των μεθόδων getRowData(), getRowKey(), και load() δίνονται παρακάτω:

```
    @Override
    public Book getRowData(String rowKey) {
        for (Book book : datasource) {
            if (book.getId() == Integer.parseInt(rowKey)) {
                return book;
            }
        }
        return null;
    }

    @Override
    public String getRowKey(Book book) {
        return String.valueOf(book.getId());
    }
```

Η μέθοδος **load()** επιστρέφει τη λίστα με τα βιβλία του τεμπέλικου μοντέλου, που θα προβληθούν στην τρέχουσα σελίδα. Πιο συγκεκριμένα:

- **Σειρά 45:** Δημιουργείται μια κενή λίστα βιβλίων ώστε να προστεθούν τα βιβλία που θα προβληθούν.
- **Σειρά 47:** υπάρχει μία επανάληψη η οποία προσπελάζει όλη τη λίστα των βιβλίων που πήραμε από την βάση δεδομένων στην μέθοδο δημιουργίας της κλάσης. **Ελέγχει** αν υπάρχουν ενεργά **φίλτρα** (κριτήρια αναζήτησης), και αν ναι, ελέγχει αν το βιβλίο πληροί τα κριτήρια. Αν δεν τα πληροί θέτει την μεταβλητή match = false. Στο τέλος αυτού του ελέγχου αν το match = true, δηλαδή το βιβλίο πληροί τα κριτήρια αναζήτησης, το εισάγει στην λίστα των βιβλίων (σειρά 82).
- **Σειρά 88: Ελέγχει** αν έχει επιλεχθεί κάποια **ταξινόμηση**, και αν ναι, ταξινομεί την λίστα των βιβλίων ανάλογα, χρησιμοποιώντας την βοηθητική κλάση LazyBookSorter.
- **Σειρά 97-98:** Βρίσκει το μέγεθος της λίστας των βιβλίων που πληρούν τα κριτήρια αναζήτησης, και θέτει τον αριθμό των γραμμών του πίνακα ίσο με τόσο.
- **Σειρά 101:** Τέλος, σύμφωνα με το πόσα βιβλία έχουν επιλέξει να εμφανίζονται στη σελίδα (pageSize), και σε ποια σελίδα του πίνακα βρίσκεται ο χρήστης, επιστρέφεται το υποσύνολο των βιβλίων το οποίο πρέπει να εμφανιστεί. Αυτό εδώ είναι και το κομμάτι του κώδικά που ουσιαστικά κάνει το lazy loading. Αντί να επιστρέψει όλη τη λίστα με τα βιβλία επιστρέφει ένα υποσύνολο της λίστας σύμφωνα με τον αριθμό των βιβλίων που θα φαίνονται στην τωρινή σελίδα.



```
@Override
public List<Book> load(int first, int pageSize, Map<String, SortMeta> sortBy, Map<String, FilterMeta> filters) {
    List<Book> books = new ArrayList<>();

    for (Book book : datasource) {
        boolean match = true;

        if (filters != null) {
            for (Iterator<String> it = filters.keySet().iterator(); it.hasNext();) {
                try {
                    String fieldValue;
                    String filterProperty = it.next();
                    Object filterValue = filters.get(filterProperty).getFilterValue();

                    if (filterProperty.equalsIgnoreCase("availability.availability")) {
                        fieldValue = book.getAvailability().getAvailability().toString();
                        match = fieldValue.equals(filterValue.toString());
                        if(!match) break;
                    } else {
                        if (filterProperty.equalsIgnoreCase("bookAuthorListToString")) {
                            fieldValue = book.getBookAuthorListToString();
                        } else {
                            Field privateMember = book.getClass().getDeclaredField(filterProperty);
                            privateMember.setAccessible(true);
                            fieldValue = String.valueOf(privateMember.get(book));
                        }
                        if (filterValue == null || fieldValue.toUpperCase().contains(filterValue.toString().toUpperCase())) {
                            match = true;
                        } else {
                            match = false;
                            break;
                        }
                    }
                } catch (IllegalAccessException | IllegalArgumentException | NoSuchFieldException | SecurityException e) {
                    match = false;
                }
            }
        }

        if (match) {
            books.add(book);
        }
    }

    //sort
    if (!sortBy.isEmpty()) {
        List<Comparator<Book>> comparators = sortBy.values().stream()
                .map(o -> new LazyBookSorter(o.getField(), o.getOrder()))
                .collect(Collectors.toList());
        Comparator<Book> cp = ComparatorUtils.chainedComparator(comparators); // from apache
        books.sort(cp);
    }

    //rowCount
    int booksSize = books.size();
    this.setRowCount(booksSize);

    //paginate
    if (booksSize > pageSize) {
        try {
            return books.subList(first, first + pageSize);
        } catch (IndexOutOfBoundsException e) {
            return books.subList(first, first + (booksSize % pageSize));
        }
    } else {
        return books;
    }
}
```

Εικόνα 26. Η μέθοδος LazyBookDataModel#load()

books.xhtml JSF view

Η JSF view είναι η δήλωση στην πλευρά του διακομιστή της διεπαφής καταλόγου βιβλίων. Στην δήλωση της JSF view, η βασική συνιστώσα PrimeFaces dataTable χρησιμοποιεί την ιδιότητα lazyModel που ορίζεται στο αντικείμενο της BookView.

Τα χαρακτηριστικά του dataTable εξηγήθηκαν αναλυτικά στο Μέρος 1, Κεφάλαιο PrimeFaces.

```
<p:dataTable var="book" value="#{bookView.lazyModel}"  id="bookTable"
             widgetVar="dtbooks"
             paginator="true" rows="5"
             paginatorTemplate="{FirstPageLink} {PreviousPageLink}
             {CurrentPageReport} {NextPageLink} {LastPageLink} {RowsPerPageDropdown}"
             paginatorPosition="bottom"
             rowsPerPageTemplate="5,10,20"
             stripedRows="true"
             reflow="true"
             lazy="true"
             selectionMode="single"
             selection="#{bookView.selectedBook}" rowKey="#{book.id}">
    <p:ajax event="rowSelect" update="form:view-book-content" oncomplete="PF('viewBookDialog').show()"/>
    <f:facet ...5 lines />

    <p:column field="id" headerText="#{msgs.BookID}"/>

    <p:column field="title" headerText="#{msgs.BookTitle}"/>

    <p:column field="bookAuthorListToString" headerText="#{msgs.Authors}"/>

    <p:column field="copies" headerText="#{msgs.Copies}"/>

    <p:column ...9 lines />

    <p:column ...13 lines />

</p:dataTable>
```

Εικόνα 27. Κώδικας τμήματος JSF view books.xhtml με PrimeFaces dataTable για την προβολή του καταλόγου βιβλίων

Αυτές ήταν οι κύριες κλάσεις για την προβολή του καταλόγου, υπάρχουν και μερικές βοηθητικές κλάσεις (Εικόνα 37).

Για την αναζήτηση βιβλίου με κριτήριο την κατάσταση χρησιμοποιείται η συνιστώσα SelectOneMenu, και τις διαθέσιμες επιλογές της παίρνει από τη μέθοδο getAvailabilityValues της BookView η οποία μέθοδο επιστρέφει όλες της διαθέσιμες τιμές του BookAvailabilityEnum (AVAILABLE, UNAVAILABLE, EXPECTED).

Κλάση LazyBookSorter και ταξινόμηση

Η ταξινόμηση των στηλών του τεμπέλικου μοντέλου πραγματοποιείται στην LazyBookDataModel#load(), με την βοήθεια και της LazyBookSorter#compare(). Η μέθοδος compare() υλοποιεί το interface Comparator και συγκρίνει 2 τιμές. Χρησιμοποιείται για την ταξινόμηση του των βιβλίων βάσει του κωδικού βιβλίου (id). Στην διεπαφή χρήστη, οι τιμές του id είναι τύπου String. Η compare() συγκρίνει τις αντίστοιχες ακέραιες τιμές για ταξινόμηση με ακέραια σειρά.

Ο κώδικας φαίνεται στην παρακάτω εικόνα:

```
class LazyBookSorter implements Comparator<Book> {

    private final String sortField;
    private final SortOrder sortOrder;

    LazyBookSorter(String sortField, SortOrder sortOrder) {
        this.sortField = sortField;
        this.sortOrder = sortOrder;
    }

    @Override
    public int compare(Book book1, Book book2) {
        try {
            int value;
            if (sortField.equalsIgnoreCase("availability.availability")) {
                String value1 = book1.getAvailability().getAvailability().toString();
                String value2 = book2.getAvailability().getAvailability().toString();
                value = ((Comparable) value1).compareTo(value2);
            } else if (sortField.equalsIgnoreCase("bookAuthorListToString")) {
                String value1 = book1.getBookAuthorListToString();
                String value2 = book2.getBookAuthorListToString();
                value = ((Comparable) value1).compareTo(value2);
            } else {
                Field privateMember1 = book1.getClass().getDeclaredField(sortField);
                privateMember1.setAccessible(true);
                String value1 = String.valueOf(privateMember1.get(book1));

                Field privateMember2 = book2.getClass().getDeclaredField(sortField);
                privateMember2.setAccessible(true);
                String value2 = String.valueOf(privateMember2.get(book2));

                if (privateMember1.getType() == Long.class
                        || privateMember1.getType() == short.class) {
                    int int1 = Integer.parseInt(value1);
                    int int2 = Integer.parseInt(value2);
                    value = ((Comparable) int1).compareTo(int2);
                } else {
                    value = ((Comparable) value1).compareTo(value2);
                }
            }

            return SortOrder.ASCENDING.equals(sortOrder) ? value : -1 * value;
        } catch (IllegalAccessException | IllegalArgumentException | NoSuchFieldException | SecurityException e) {
            throw new RuntimeException(e);
        }
    }

}
```

Εικόνα 28. Κλάση LazyBookSorter

2. Προβολή Πληροφοριών Βιβλίου

Για την επιλογή γραμμής και προβολή των πληροφοριών του βιβλίου ο κώδικας που το κάνει δυνατό από τον πίνακα books.xhtml είναι ο παρακάτω:

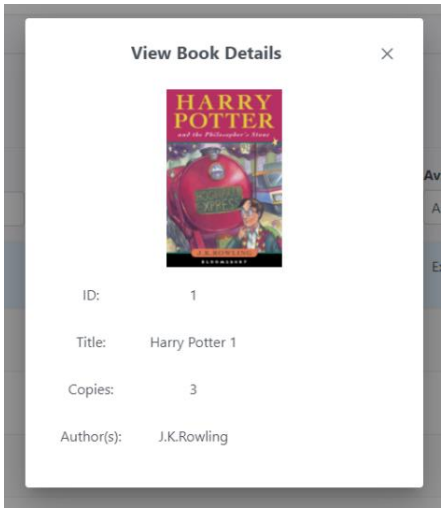
```
…
   selectionMode="single"
   selection="#{bookView.selectedBook}" rowKey="#{book.id}">
<p:ajax event="rowSelect" update="form:view-book-content"
oncomplete="PF('viewBookDialog').show()"/>
…
```

Όταν επιλεχτεί μια γραμμή (αριστερό κλικ) του πίνακα γίνονται 3 πράγματα:

- Ενημερώνεται το πεδίο selectedBook της BookView επειδή έχει τεθεί το χαρακτηριστικό selection της dataTable =#{bookView.selectedBook}.
- Ενημερώνεται η συνιστώσα με id view-book-content μέσω Ajax, και η ιδιότητα selectedBook παίρνει τις ανανεωμένες τιμές.
- Όταν ενημερωθεί η συνιστώσα μέσω του χαρακτηριστικού oncomplete εμφανίζεται η συνιστώσα dialog με widgetVar viewBookDialog.

```
<p:dialog header="#{msgs.BookDetailsDialogHeader}" showEffect="fade" modal="true"
          widgetVar="viewBookDialog" responsive="true" fitViewport="true">
    <p:outputPanel id="view-book-content" class="ui-fluid" style="width: 400px;">
        <p:graphicImage id="view-image" value="#{bookView.selectedBook.imagepath}" width="130px" height="200px"
                        alt="#{msgs.NoPictureAvailable}"/>
        <p:panelGrid columns="2">
            <p:outputLabel for="view-id" value="#{msgs.IDLabel}"/>
            <p:outputLabel id="view-id" value="#{bookView.selectedBook.id}"/>

            <p:outputLabel for="view-title" value="#{msgs.TitleLabel}"/>
            <p:outputLabel id="view-title" value="#{bookView.selectedBook.title}"/>

            <p:outputLabel for="view-copies" value="#{msgs.CopiesLabel}"/>
            <p:outputLabel id="view-copies" value="#{bookView.selectedBook.copies}"/>

            <p:outputLabel for="view-authors" value="#{msgs.AuthorsLabel}"/>
            <p:outputLabel id="view-authors" value="#{bookView.selectedBook.bookAuthorListToString}"/>
        </p:panelGrid>
    </p:outputPanel>
</p:dialog>
```

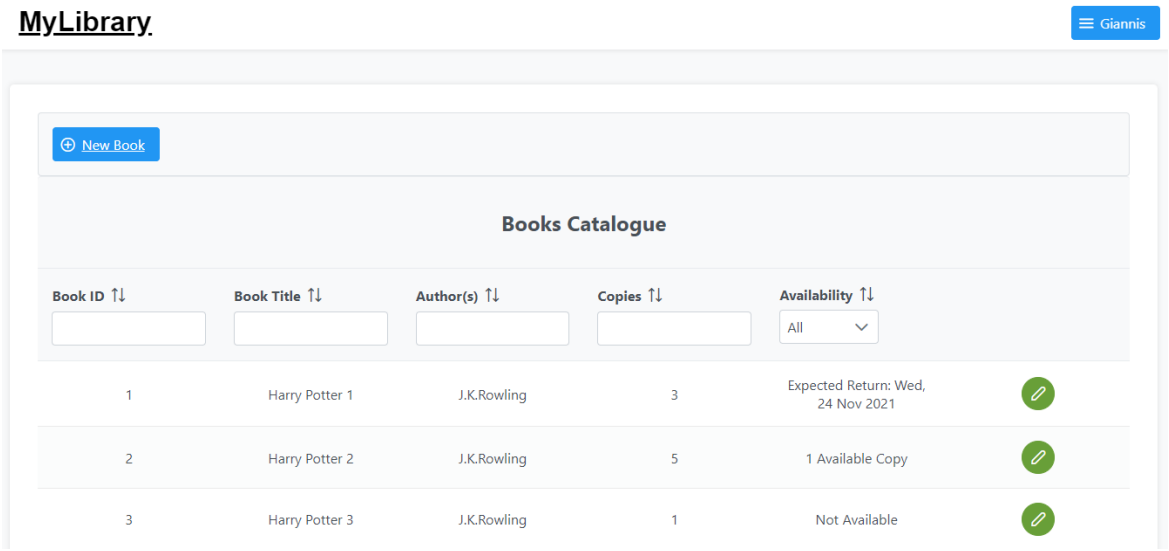
Εικόνα 29. Κώδικας Dialog για προβολή πληροφοριών βιβλίου



Εικόνα 30. Dialog για προβολή πληροφοριών βιβλίου

3. Επεξεργασία Βιβλίου

Για την επεξεργασία του βιβλίου έχει προστεθεί μία στήλη με ένα κουμπί στο dataTable το οποίο εμφανίζεται μόνο αν ο χρήστης έχει δικαιώματα βιβλιοθηκάριου ή διαχειριστή.



Εικόνα 31. Κατάλογος βιβλίων, είσοδος ως βιβλιοθηκάριος

Το πότε εμφανίζεται το κουμπί της επεξεργασίας ελέγχεται μέσω της μεθόδου hasLibrarianPrivileges() της κλάσης AuthenticationBean.

```
public boolean hasLibrarianPrivileges() {
        return getRequest().isUserInRole("librarian") ||
getRequest().isUserInRole("admin");
    }

private HttpServletRequest getRequest() {
   FacesContext context = FacesContext.getCurrentInstance();
   return (HttpServletRequest)context.getExternalContext().getRequest();
}
```

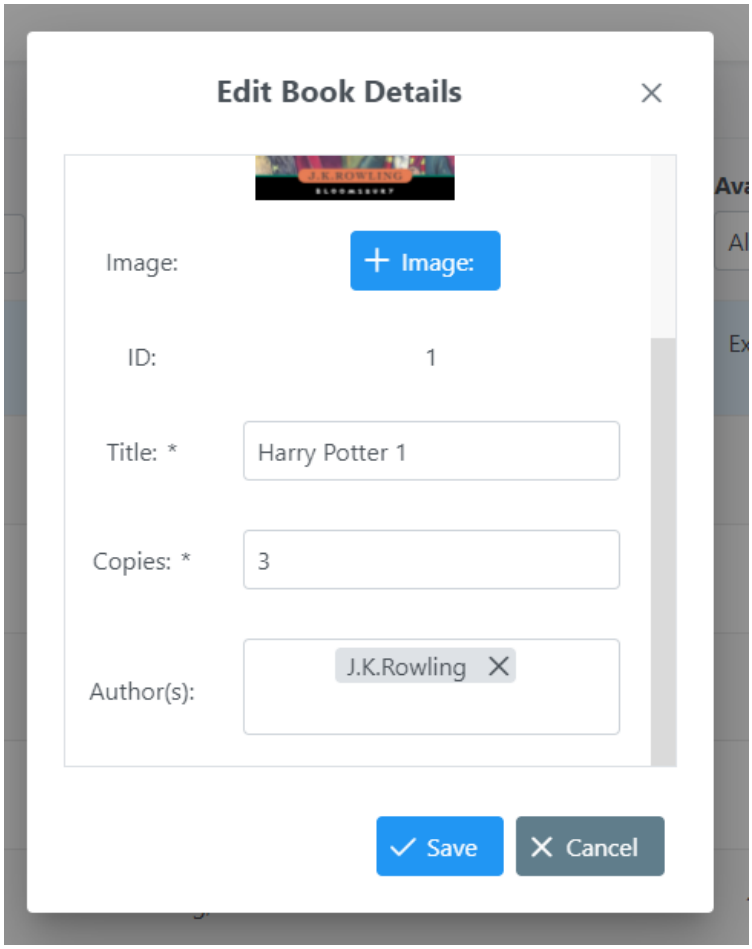
Η μέθοδος getRequest δίνει πρόσβαση στο τωρινό αίτημα και χρησιμοποιώντας τη μέθοδο isUserInRole() γίνεται να ελεγχθεί ο ρόλος του συνδεδεμένου χρήστη.

```
<p:column rendered="#{authenticationBean.hasLibrarianPrivileges()}">
    <p:commandButton icon="pi pi-pencil" update=":form:manage-book-content"
                     oncomplete="PF('manageBookDialog').show()"
                     styleClass="edit-button rounded-button ui-button-success" process="@this">
        <f:setPropertyActionListener value="#{book}" target="#{bookView.selectedBook}"/>
        <p:resetInput target=":form:manage-book-content"/>
    </p:commandButton>
```

Εικόνα 32. Κώδικας στήλης επεξεργασίας βιβλίου

Αν πατηθεί το κουμπί επεξεργασίας το επιλεγμένο βιβλίο, όπως και στην περίπτωση της προβολής βιβλίου, θέτεται σαν selectedBook, με τη διαφορά ότι εδώ γίνεται μέσω της συνιστώσας setPropertyActionListener (Εικόνα 37).

Στη συνέχεια ενημερώνεται ένα dialog με id manage-book-content και εμφανίζεται, μέσω των χαρακτηριστικών update και oncomplete αντίστοιχα (Εικόνα 37).



Εικόνα 33. Dialog Επεξεργασίας Βιβλίου

Το dialog για την επεξεργασία βιβλίου περιέχει μία φόρμα με τα εξής:

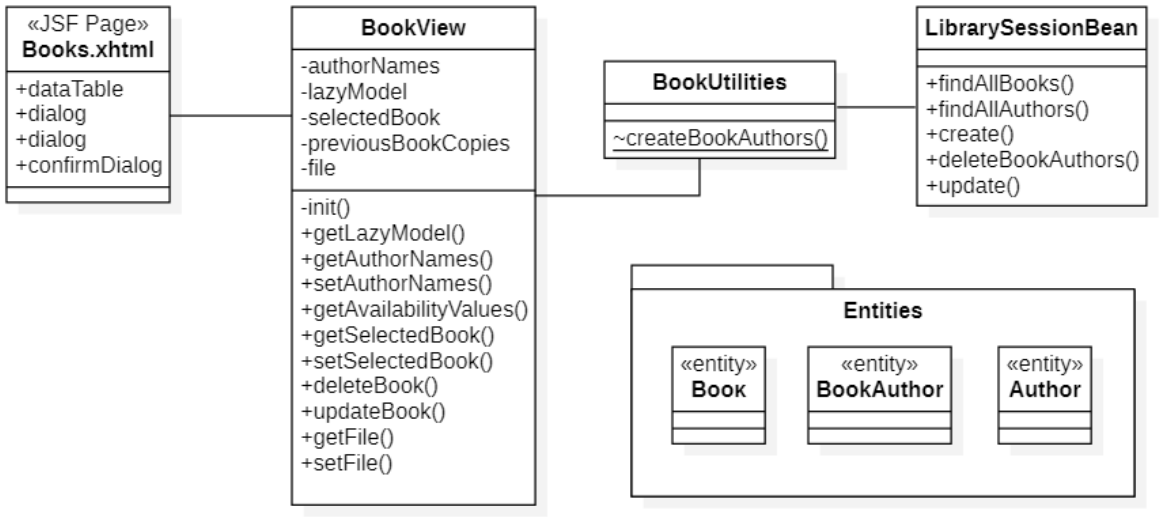
- Μια συνιστώσα graphicImage για να εμφανιστεί η εικόνα του βιβλίου
- Μια συνιστώσα fileUpload για ανέβασμα καινούργιας εικόνας η οποία συνδέεται με τις μεθόδους getFile(), setFile() του BookView
- Μια συνιστώσα outputLabel για την προβολή του id
- Μια συνιστώσα inputText για το όνομα του βιβλίου
- Μια συνιστώσα inputNumber με περιορισμό να δέχεται μόνο ακέραιους για τον αριθμό των αντιγράφων.
- Μία συνιστώσα autoComplete για τους συγγραφείς η οποία χρησιμοποιεί τις μεθόδους getAuthorNames(), setAuthorNames() του BookView καθώς και τη μέθοδο completeText() του BookManagedBean (Δεν είναι το ιδανικό μέρος για να αποθηκευτεί αυτή η μέθοδος, πιθανώς στην κλάση BookUtilities, μελλοντικά όταν κοιτάξω να βελτιστοποιήσω την εφαρμογή αυτό θα αλλάξει καθώς και πολλά ακόμα).

Όταν ολοκληρωθεί η επεξεργασία και πατηθεί Save καλείται η μέθοδος updateBook() της BookView η οποία ενημερώνει το βιβλίο. Η μέθοδος αυτή καλεί με την σειρά της τη στατική μέθοδο της BookUtilites createBookAuthors() η οποία δημιουργεί/ενημερώνει τους συγγραφείς του Βιβλίου και ενημερώνει το βιβλίο στη βάση.

Η createBookAuthors() επικοινωνεί με την βάση μέσω του LibrarySessionBean και κάνει τα εξής βήματα:

- Βρίσκει μια λίστα με όλους τους συγγραφείς στη βάση (findAllAuthors()),

- Ελέγχει τη λίστα και δημιουργεί τους συγγραφείς του βιβλίου εάν δεν υπάρχουν,
- Ξαναφορτώνει τη λίστα συγγραφέων και βρίσκει τους συγγραφείς του επιλεγμένου βιβλίου (Αυτό γίνεται ώστε ώστε να πάρουμε από την βάση το id του συγγραφέα, ο οποίος έπρεπε να δημιουργηθεί πρώτα για να αποκτήσει id),
- Διαγράφει από τον πίνακα BookAuthor τους παλιούς συγγραφείς του βιβλίου (deleteBookAuthors()),
- Προσθέτει τους καινούργιους συγγραφείς στον πίνακα BookAuthor (create())
- Ενημερώνει το βιβλίο με την μέθοδο update()



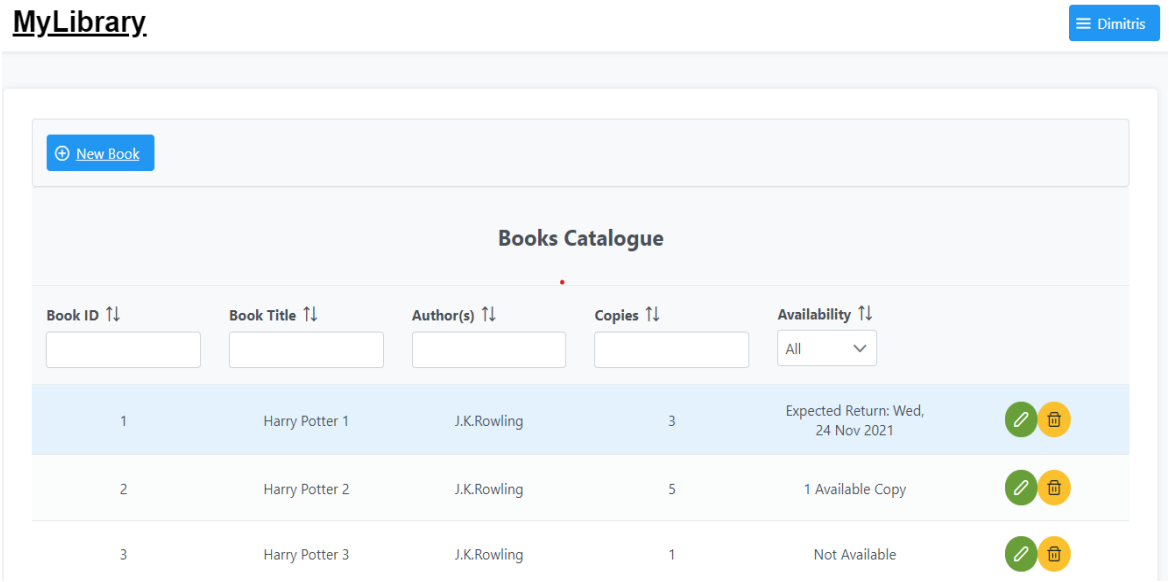
Εικόνα 34. Απλοποιημένο Διάγραμμα UML για την επεξεργασία βιβλίου

4. Διαγραφή Βιβλίου

Στη τελευταία στήλη του καταλόγου βιβλίων, η οποία εμφανίζεται μόνο αν ο χρήστης έχει δικαιώματα βιβλιοθηκάριου, έχει προστεθεί και ένα επιπλέον κουμπί για τη διαγραφή του βιβλίου, το οποίο όμως εμφανίζεται μόνο αν ο χρήστης έχει δικαιώματα διαχειριστή, το οποίο ελέγχεται και αυτό με τον ίδιο τρόπο μέσω της μεθόδου hasAdminPrivileges().

```
<p:column rendered="#{authenticationBean.hasLibrarianPrivileges()}">
    <p:commandButton icon="pi pi-pencil" update=":form:manage-book-content"
                     oncomplete="PF('manageBookDialog').show()"
                     styleClass="edit-button rounded-button ui-button-success" process="@this">
        <f:setPropertyActionListener value="#{book}" target="#{bookView.selectedBook}"/>
        <p:resetInput target=":form:manage-book-content"/>
    </p:commandButton>
    <p:commandButton class="ui-button-warning rounded-button" icon="pi pi-trash"
                     oncomplete="PF('deleteBookDialog').show()" process="@this"
                     rendered="#{authenticationBean.hasAdminPrivileges()}">
        <f:setPropertyActionListener value="#{book}" target="#{bookView.selectedBook}"/>
    </p:commandButton>
</p:column>
```

Εικόνα 35. Κώδικας για κουμπιά επεξεργασία και διαγραφής βιβλίου



Εικόνα 36. Κατάλογος Βιβλίων, είσοδος ως διαχειριστής

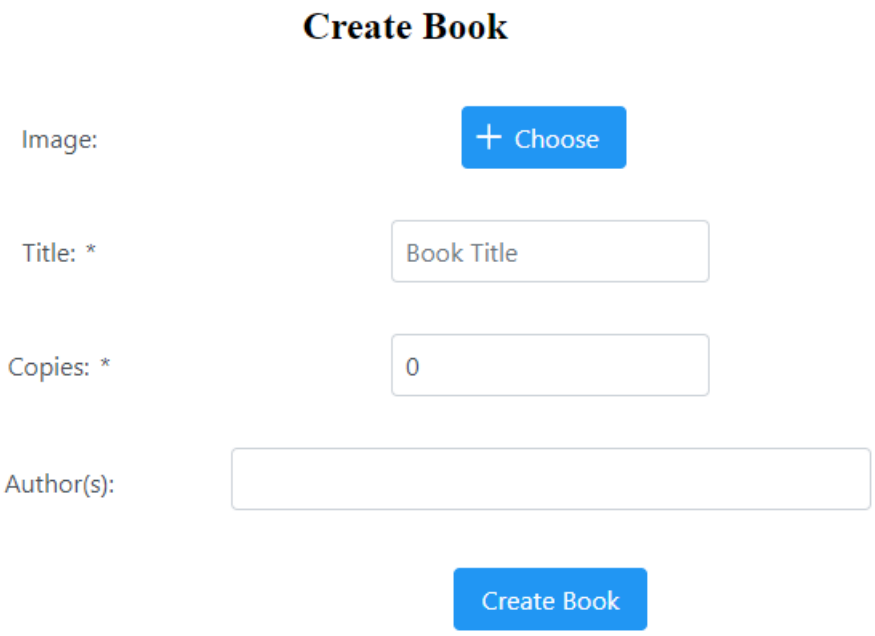
Στη διαγραφή όπως και στην επεξεργασία θέτεται σαν selectedBook το βιβλίο της γραμμής μέσω της συνιστώσας setPropertyActionListener και εμφανίζεται ένα confirmDialog, στο οποίο, αν επιλεχθεί το "ναι", καλείται η μέθοδος deleteBook() της BookView η οποία διαγράφει το συγκεκριμένο βιβλίο καλώντας τη μέθοδο delete της LibrarySessionBean, με την προϋπόθεση ότι το βιβλίο δεν ανήκει σε κάποιο ανοιχτό δάνειο. Η μέθοδος openLoanExists(), ελέγχει αν υπάρχει στο πίνακα loan κάποια δάνειο με το συγκεκριμένο βιβλίο, στο οποίο η ημερομηνία επιστροφής είναι null, δηλαδή δεν έχει επιστραφεί.

```
public String deleteBook() {
    try {
        if (librarySessionBean.openLoanExists(selectedBook.getId())) {
            Message.addErrorMessage("Book Deletion Failed", "Open Loan Exists");
            return "/books.xhtml?faces-redirect=true";
        }
        librarySessionBean.delete(selectedBook);
        selectedBook = null;
        Message.addSuccessMessage("Book Deleted");
    } catch (Exception e) {
        Message.addErrorMessage("Book Deletion Failed");
    }
    return "/books.xhtml?faces-redirect=true";
}
```

Εικόνα 37. Μέθοδος deleteBook()

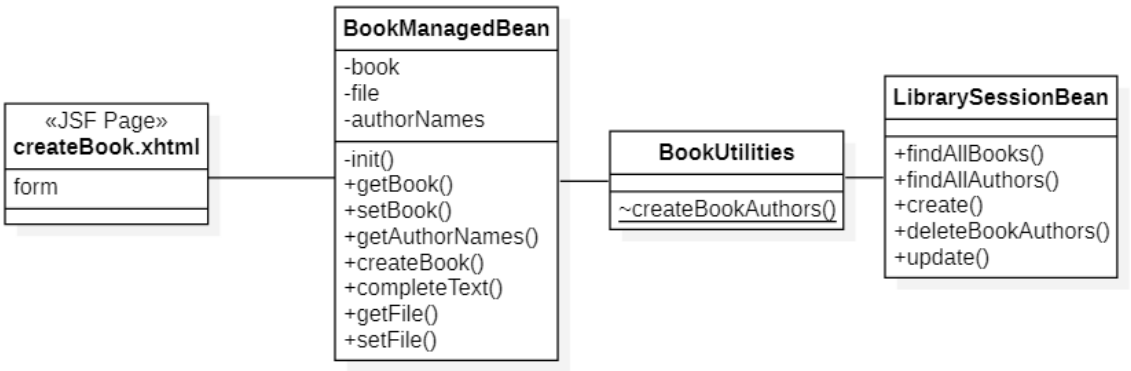
5. Εισαγωγή Νέου Βιβλίου

Αν ο χρήστης που έχει συνδεθεί στην εφαρμογή έχει τον ρόλο του βιβλιοθηκάριου ή του διαχειριστή, στην σελίδα του καταλόγου βιβλίων καθώς και στο κεντρικό μενού της εφαρμογής, εμφανίζεται η επιλογή New Book. Επιλέγοντας το, εμφανίζεται μια νέα σελίδα JSF η οποία περιέχει μια απλή φόρμα για την δημιουργία ενός νέου βιβλίου (Εικόνα 43).



Εικόνα 38. Φόρμα δημιουργίας βιβλίου

Η φόρμα αυτή χρησιμοποιεί της ίδιες συνιστώσες με τη φόρμα για την επεξεργασία του βιβλίου που προανέφερα, με τη διαφορά ότι οι πληροφορίες του βιβλίου αποθηκεύονται στο BookManagedBean, στο πεδίο book. Όταν υποβληθεί η φόρμα, καλείται η μέθοδος createBook() της BookManagedBean, η οποία με τη σειρά της καλεί τη στατική μέθοδο της BookUtilites createBookAuthors() η οποία δημιουργεί/ενημερώνει τους συγγραφείς του βιβλίου και ύστερα δημιουργεί το βιβλίο.



Εικόνα 39. Απλοποιημένο Διάγραμμα UML για την δημιουργία βιβλίου

JSF view template

Η κοινή δομή των JSF views ορίζεται με Facelets templating. Ειδικότερα, οι κοινοί header και footer δηλώνονται στο έγγραφο template mainLayout.xhtml.

Χρησιμοποιώ το tag `<ui:include src="/components/header.xhtml"/>` για να εισάγω το αρχείο header.xhtml στο template, και αντίστοιχα και το footer. Το tag <ui:include> εισάγει σε εκείνη τη θέση του template το αρχείο που αναφέρεται στο χαρακτηριστικό src.

Το tag `<ui:insert name="content"/>` το οποίο διευκρινίζει ότι εδώ θα εισαχθεί το content στην σελίδα template. Ύστερα χρησιμοποιώντας στις σελίδες το tag `<ui:define name="content">` ορίζω το περιεχόμενο που θέλω να εμφανίζεται στην θέση που υπάρχει στο <ui:insert name="content"> στο template. Αν δεν έχει οριστεί content μπορούμε να επιλέξουμε να εμφανίζεται κάτι άλλο γράφοντας μέσα στο tag, π.χ.:

```
<ui:insert name="content">
      Default Content Here
</ui:insert>
```

Για να ορίσω το content στις σελίδες μου π.χ. στην σελίδα books:

```
<ui:composition template="/templates/mainLayout.xhtml">
      <ui:define name="title">
            myLibrary - Books Catalogue
      </ui:define>

      <ui:define name="content">
            Κυρίως περιεχόμενο - Πίνακας με Βιβλία
      </ui:define>
</ui:composition>
```

Το παραπάνω θα εισάγει στο mainLayout εκεί που έχω insert με name=title το περιεχόμενο του define με name=title και αντίστοιχα και για το content.

Τέλος, στο mainLayout ορίζω και meta tags καθώς και μια σύνδεση με ένα απλό css αρχείο που δημιούργησα για την εμφάνιση της σελίδας. Τα αρχεία css σε τέτοια project τοποθετούνται συνήθως στο φάκελο webapp/resources/css. Σύμφωνα με αυτό το template και την css η μορφή της σελίδας μου είναι όπως παρακάτω:

Εικόνα 40. Κύριο πρότυπο σελίδων της εφαρμογής

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html>
<html     xmlns="http://www.w3.org/1999/xhtml"
          xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
          xmlns:h="http://xmlns.jcp.org/jsf/html"
          xmlns:p="http://primefaces.org/ui"
          xmlns:f="http://xmlns.jcp.org/jsf/core">

    <f:view locale="#{localeManagedBean.locale}">
        <h:head>
            <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
            <meta name="viewport" content="width=device-width, initial-scale=1.0" />
            <ui:insert name="metadata"/>
            <title><ui:insert name="title">myLibrary</ui:insert></title>
            <h:outputStylesheet library="css" name="style.css"/>
        </h:head>

        <h:body>
            <p:growl id="messages" showDetail="true"/>
            <ui:include src="/components/sessionTimeout.xhtml"/>

            <div id="main-header">
                <ui:include src="/components/header.xhtml"/>
            </div>

            <div class="wrap">
                <div id="content" class="box">
                    <ui:insert name="content"/>
                </div>
            </div>

            <div id="footer" style="margin-top: 0">
                <ui:include src="/components/footer.xhtml"/>
            </div>
        </h:body>
    </f:view>
</html>
```

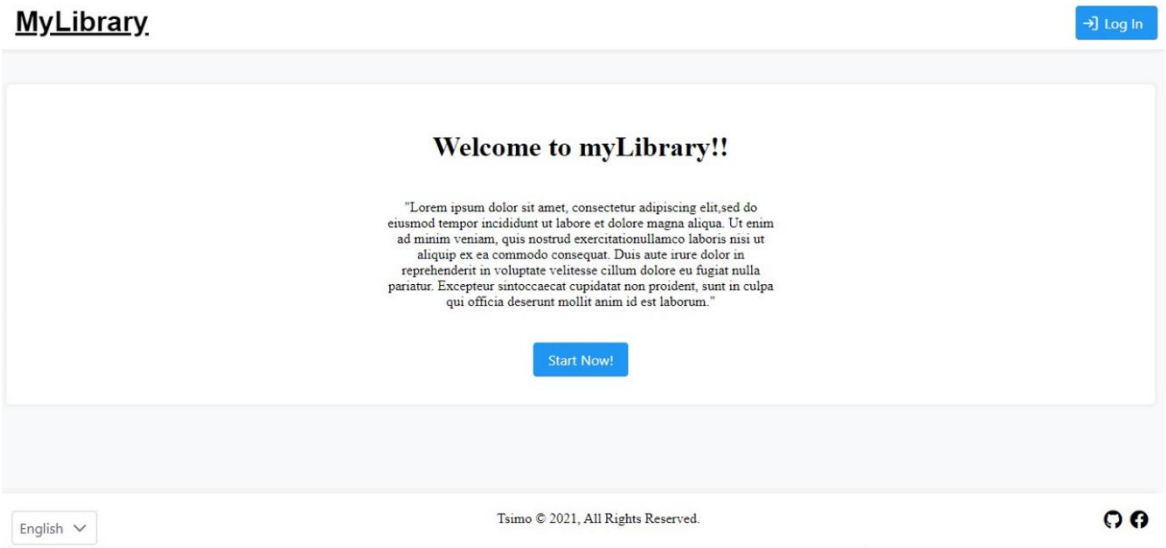
Εικόνα 41. Κώδικας πρότυπου της εφαρμογής, αρχείο mainLayout.xhtml

Δημιουργία σελίδων Index, Header & Footer

Η αρχική σελίδα (index.xhtml) περιέχει ένα απλό μήνυμα καλωσορίσματος, ένα δειγματικό κείμενο και ένα κουμπί που κατευθύνει το χρήστη στον κατάλογο με τα βιβλία. Το κουμπί είναι `<p:button>` διότι υιοθέτησα το design της PrimeFaces σε όλη την εφαρμογή και τα κουμπιά της έχουν αυτόματα μια πιο ωραία μοντέρνα μορφή από της JSF (`<h:button>`).

Το header της εφαρμογής περιέχει ένα link για την αρχική σελίδα index.xhtml το οποίο το χρησιμοποιώ και σαν τίτλο της εφαρμογής, θα μπορούσε να αντικατασταθεί με ένα λογότυπο. Στη δεξιά μεριά του header θα προσθέσω ένα κουμπί Log In, και όταν ο χρήστης κάνει Log In το κουμπί θα κρύβεται και θα εμφανίζεται ένα ανάλογο menu, χρησιμοποιώντας το `<p:menu>` tag. Αυτό το πετυχαίνω με το χαρακτηριστικό rendered ελέγχοντας αν υπάρχει συνδεδεμένος χρήστης, και τι δικαιώματα έχει ο συγκεκριμένος χρήστης μέσω του AuthenticationManagedBean. Η μορφή του Header και του menu φαίνεται στις παρακάτω εικόνες.

Το footer μου είναι λυτό. Στα αριστερά έχω προσθέσει ένα dropdown από το οποίο μπορούμε να επιλέξουμε την γλώσσα της εφαρμογής (περισσότερα στο localization), στην μέση ένα κείμενο για τα copyrights και δεξιά 2 εικονίδια-συνδέσμους για social προφίλ. Σε μια αληθινή χρήση της εφαρμογή.



Εικόνα 42. Header, Index και Footer της εφαρμογής

JSF Localization - Resources Bundle

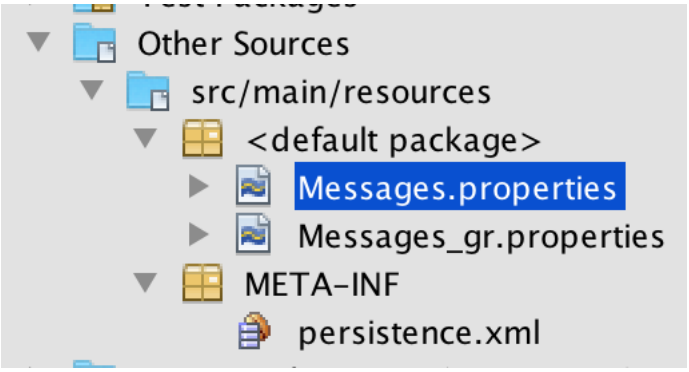
Μη λειτουργική απαίτηση: μλα.γλώσσα: Η διεπαφή χρήστη θα μπορεί να εμφανίζεται σε δύο γλώσσες, ελληνικά και αγγλικά.

Περίπτωση χρήσης: πχ.επιλογήΓλώσσας

Τα κείμενα των JSF views μπορούν να αποθηκευτούν σε αρχείο με ζεύγη κωδικών και κειμένων. Στα views δηλώνονται οι κωδικοί και εμφανίζονται τα κείμενα. Τα ζεύγη αυτά αποθηκεύονται σε αρχεία ASCII (με κατάληξη .properties). Με αυτόν τον τρόπο έχουμε

όλα τα μηνύματα μαζεμένα το οποίο κάνει πιο εύκολη την διαχείριση τους. Επίσης είναι δυνατό να έχουμε διαφορετικά αρχεία για διαφορετικές γλώσσες.

Σε εφαρμογές που χρησιμοποιούν Maven, τα αρχεία .properties πρέπει να αποθηκευτούν στο φάκελο resources (src/main/resources).



Εικόνα 43. Πακέτα Πόρων για αποθήκευση μηνυμάτων

Η χρήση αρχείων μηνυμάτων για πολλαπλές γλώσσες δηλώνεται μέσω της παρακάτω παραμετροποίησης του αρχείου faces-config.xml:

```
<application>
      <resource-bundle>
            <base-name>
                Messages
            </base-name>
            <var>msgs</var>
      </resource-bundle>
      <locale-config>
            <default-locale>en</default-locale>
            <supported-locale>gr</supported-locale>
      </locale-config>
</application>
```

Για παράδειγμα για να εμφανιστεί ο τίτλος του καταλόγου βιβλίων που δείξαμε στο κεφάλαιο Δημιουργία Βασικού Template, μπορεί να προστεθεί στο αρχείο Messages.properties η γραμμή `BooksTitle=myLibrary - Books Catalogue` και να προστεθεί στη σελίδα ο κωδικός του κειμένου EL όπως παρακάτω:

```
<ui:define name="title">
      #{msgs.BooksTitle}
</ui:define>
```

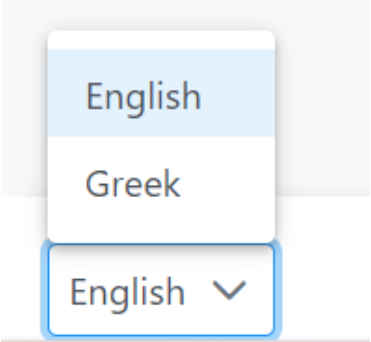
Με αυτόν τον τρόπο θα εμφανίζονται και τα υπόλοιπα μηνύματα της εφαρμογής ώστε να γίνει εφικτή η αλλαγή γλώσσας.

```
HomePageWelcome=Welcome to myLibrary!!
HomePageText="Lorem ipsum dolor sit amet, consectetur adipiscing elit,sed do eiusmod tempor inci
HomePageActionButton=Start Now!

FooterCopyrightText=Tsimo © 2021, All Rights Reserved.

LibraryConfigurationTitle=System Configuration
LibraryConfigurationLoanDays=Max Loan Days:
LibraryConfigurationLoanDaysRequired=Please enter loan days
LibraryConfigurationLoanQuantity=Max Loan Books:
LibraryConfigurationLoanQuantityRequired=Please enter book quantity
LibraryConfigurationImagePath=Images Directory:
LibraryConfigurationImagePathRequired=Please enter images directory
```

Εικόνα 44. Αρχείο Messages.properties

```
HomePageWelcome=Καλωσήρθατε στο myLibrary!!
HomePageText=Λορεμ ιπσθμ δολορ σιτ αμετ, ιδ ιθσ αντιοπαμ φορενσιβθσ δεφινιτιονεμ, μελ αδ
HomePageActionButton=Έναρξη!

FooterCopyrightText=Tsimo © 2021, Με Επιφύλαξη Παντός Δικαιώματος.

LibraryConfigurationTitle=Ρυθμίσεις Συστήματος
LibraryConfigurationLoanDays=Μέγιστος Αριθμός Ημερών Δανεισμού:
LibraryConfigurationLoanDaysRequired=Παρακαλώ εισάγετε ημέρες δανεισμού
LibraryConfigurationLoanQuantity=Μέγιστος Αριθμός Βιβλίων Δανεισμού:
LibraryConfigurationLoanQuantityRequired=Παρακαλώ εισάγετε αριθμό βιβλίων
LibraryConfigurationImagePath=Τοποθεσία Εικόνων:
LibraryConfigurationImagePathRequired=Παρακαλώ εισάγετε τοποθεσία εικόνων
```

Εικόνα 45. Αρχείο Messages_gr.properties



Εικόνα 46. SelectOneMenu για επιλογή γλώσσας

Για να αποθηκεύεται η γλώσσα για την τωρινή συνεδρία και όχι απλά για την σελίδα, δημιουργώ ένα απλό LocaleManagedBean το οποίο θα είναι SessionScoped και θα αποθηκεύει την επιλογή γλώσσας του χρήστη. Η επιλογή γλώσσας γίνεται μέσω ενός dropdown (στην JSF SelectOneMenu) στο footer της σελίδας, με επιλογές Ελληνικά και Αγγλικά και default τιμή τα Αγγλικά.

Τέλος πρέπει να περιτυλίξουμε κάθε σελίδα στην οποία θα είναι δυνατή η μετάφραση με το tag `<f:view locale="#{localeManagedBean.locale}">` το οποίο μπαίνει αμέσως μετά το html tag και κλείνει πριν κλείσει το html tag αντίστοιχα. Στην εφαρμογή αυτή αφού γίνεται η χρήση template αρκεί να προστεθεί το tag στο template mainLayout.xhtml.

Μηνύματα JSF

Στις σελίδες της εφαρμογής θα εμφανίζονται κάποια μηνύματα, για παράδειγμα αν ο χρήστης εισάγει λάθος κωδικό, ή αν δεν πετύχει η εισαγωγή ενός νέου βιβλίου στο σύστημα. Στην Java EE υπάρχουν μηνύματα πολλών επιπέδων. Υπάρχουν τα μηνύματα τα οποία ορίζονται στις σελίδες JSF αν κάποιο πεδίο είναι required, υπάρχουν και

μηνύματα τα οποία εμφανίζονται μέσα από κάποια μέθοδο ανάλογα με το αποτέλεσμα των ελέγχων της, επίσης εμφανίζονται και μηνύματα σφάλματος (Exceptions) από περιορισμούς που έχουν τεθεί στα entity classes.

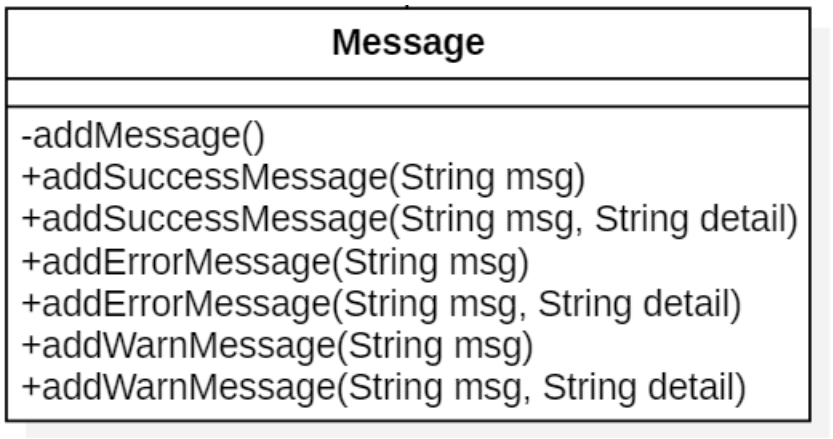
Αναφέρομαι στα 2 πρώτα είδη μηνυμάτων. Τα μηνύματα του χαρακτηριστικού requiredMessage των input πεδίων των σελίδων JSF της εφαρμογής αποθηκεύονται στο αρχείο Messages.properties ώστε να τα προσφέρονται και σε άλλη γλώσσα.

Για να προστεθεί μήνυμα στη σελίδα μέσω ενός CDI Bean πρέπει να προστεθεί σε κάθε σελίδα το tag `<h:message>` ή `<h:messages>`. Η PrimeFaces προσφέρει ένα πιο έτοιμο tag για να εμφανίζει μηνύματα το `<p:growl>`, το οποίο αντί να το προσθέσω σε κάθε σελίδα, θα το προσθέσω στο template mainLayout.xhtml.

Επίσης για να εμφανιστεί ένα μήνυμα στις σελίδες πρέπει να προσθέτω κάθε φορά τις παρακάτω σειρές κώδικα.

```
String msg = "….";
String detail= "….";
FacesMessage message = new
      FacesMessage(FacesMessage.SEVERITY_INFO, msg, detail);
FacesContext.getCurrentInstance().addMessage(null, message);
FacesContext.getCurrentInstance().getExternalContext().getFlash().
setKeepMessages(true);
```

Για να μην επαναλαμβάνεται ο κώδικας δημιούργησα μία κλάση Message με μερικές στατικές μεθόδους ώστε και να εμφανίζονται τα μηνύματα μέσα από τις μεθόδους αυτές (Εικόνα 24). Η κλάση αυτή χρησιμοποιείται από πολλαπλές κλάσεις τις εφαρμογής για εμφάνιση μηνυμάτων.



Εικόνα 47. Κλάση Message

Αυθεντικοποίηση και Εξουσιοδότηση με GlassFish Server

Αυτό το κεφάλαιο αφορά την σύνδεση του χρήστη καθώς και την αποσύνδεση. Επίσης αναφέρεται ο περιορισμός πρόσβασης σε σελίδες με βάση τα δικαιώματα του χρήστη.

Για να προστεθούν οι δυνατότητες αυθεντικοποίησης και εξουσιοδότησης στην εφαρμογή πρέπει να γίνουν τα εξής:

- Δημιουργία βασιλείου (Realm)
    - Δημιουργία JDBC Connection Pool
    - Δημιουργία JDBC Resource
    - Δημιουργία του Realm
- Δημιουργία μηχανισμού αυθεντικοποίησης (Authentication)
- Δημιουργία περιορισμών ασφάλειας (Authorization – Security Constraint)

Create Realm

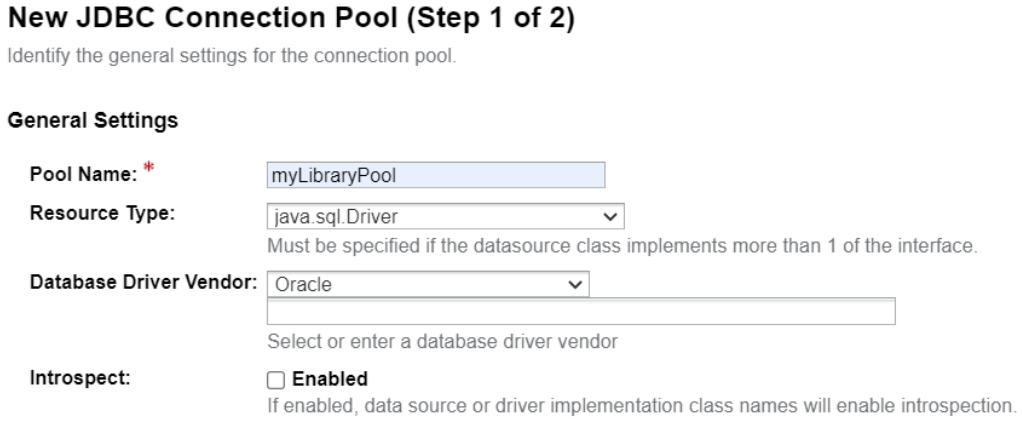
Το βασίλειο είναι μια πολιτική τομέα ασφαλείας που περιλαμβάνει χρήστες, οι οποίοι μπορεί να ανήκουν σε κάποια ομάδα. Για να γίνεται αυθεντικοποίηση πρέπει να ορίσουμε το βασίλειο στο οποίο θα υπάρχουν οι χρήστες.

Πριν δημιουργηθεί το βασίλειο χρειάζεται να δημιουργηθεί μία σύνδεση με τη βάση δεδομένων μέσω ενός JDBC Connection Pool διότι στην παρούσα εφαρμογή οι χρήστες είναι αποθηκευμένοι στον πίνακα USERS της βάσης δεδομένων. Ύστερα πρέπει να δημιουργηθεί και ένα JDBC Resource το οποίο θα επιτρέπει στο βασίλειο να έχει πρόσβαση στο JDBC Connection Pool.

Η πρόσβαση στη διεπαφή του GlassFish Server είναι εφικτή μέσω του browser εισάγοντας την διεύθυνση “localhost:4848” (default). Μέσω της διεπαφής θα δημιουργηθούν τα παραπάνω.

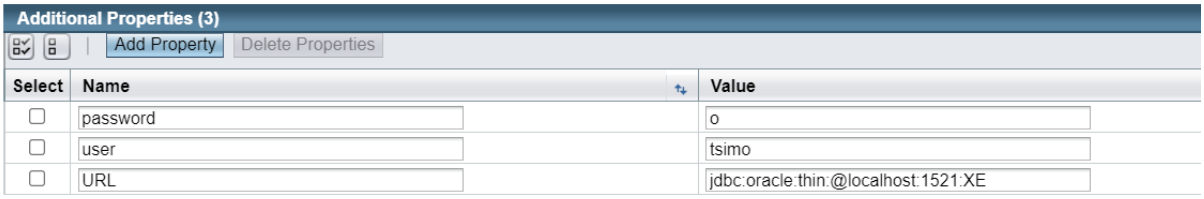
Δημιουργία JDBC Connection Pool

Για την δημιουργία JDBC Connection Pool, από τη διεπαφή επιλέγουμε Resources -> JDBC -> JDBC Connection Pools -> New. Δημιουργούμε μια σύνδεση με τη βάση δεδομένων μας με ένα όνομα της επιλογής μας (Pool Name) και συμπληρώνουμε σωστά τα υπόλοιπα πεδία (Εικόνα 47).



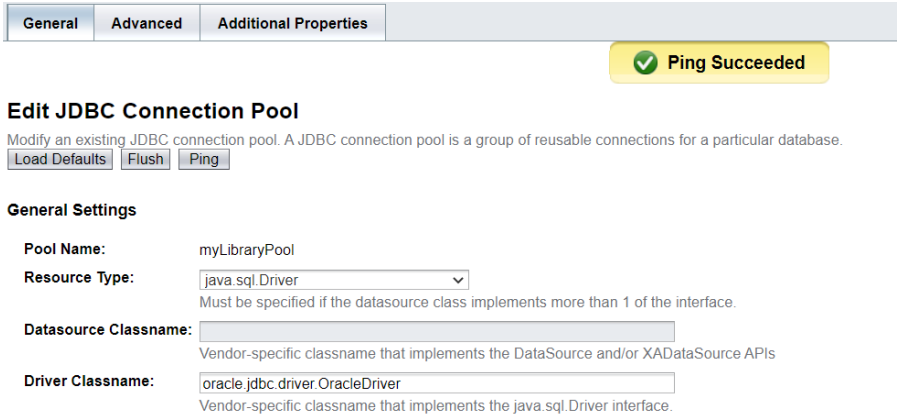
Εικόνα 48. Glassfish New JDBC Connection Pool

Πατάμε next και συμπληρώνουμε τις ιδιότητες που χρειάζεται να γνωρίζει ο GlassFish ώστε να μπορεί να συνδεθεί με την βάση δεδομένων και πατάμε Finish (Εικόνα 48).



Εικόνα 49. JDBC Pool, Additional Properties

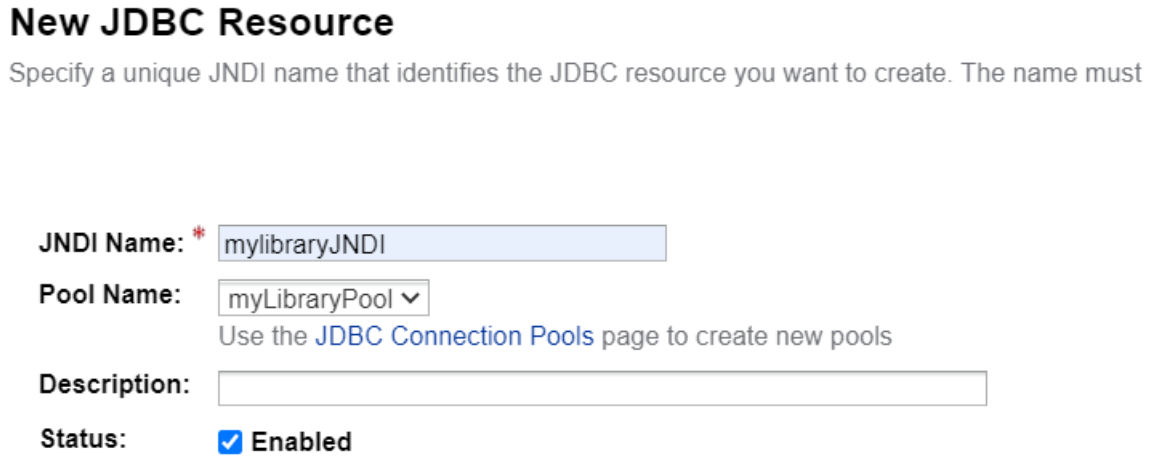
Αν επιλέξουμε τώρα το connection pool που δημιουργήσαμε, μπορούμε να στείλουμε ένα ping πατώντας το αντίστοιχο κουμπί για να ελέγξουμε αν πραγματοποιείται η σύνδεση (Εικόνα 49).



Εικόνα 50. Ping JDBC connection pool

Δημιουργία JDBC Resource

Για την δημιουργία του JDBC Resource από τη διεπαφή επιλέγουμε Resources -> JDBC -> JDBC Resources -> New. Δημιουργούμε ένα JDBC Resource, ώστε να αντιστοιχίσουμε ένα όνομα με το JDBC Connection Pool που δημιουργήσαμε νωρίτερα και να μπορούμε να το συνδέσουμε την εφαρμογή μας, πιο συγκεκριμένα να το χρησιμοποιήσουμε στο realm που θα δημιουργήσουμε.

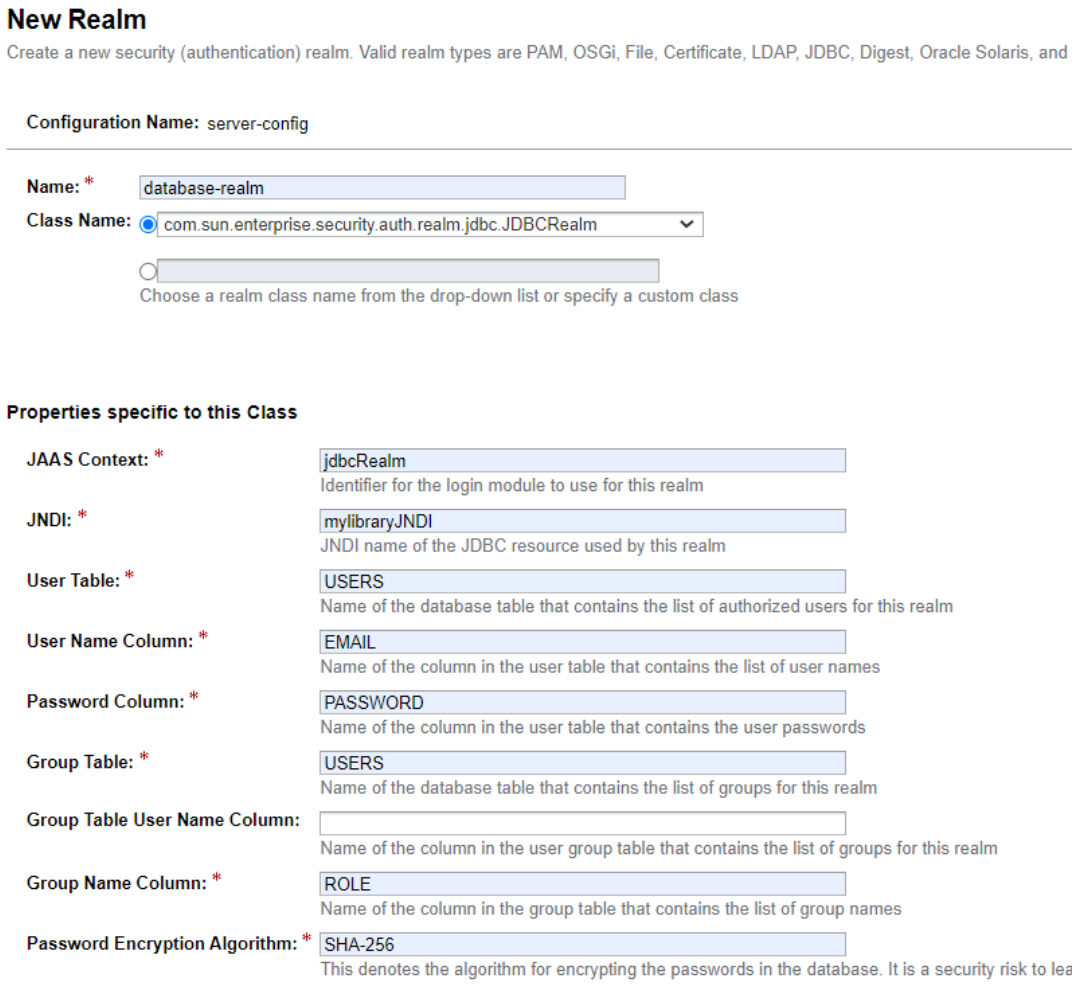


Εικόνα 51. GlassFish JDBC Resource

Δημιουργία του Realm

Στη διεπαφή του Glassfish επιλέγουμε Configurations -> server-config -> Security -> Realms -> New. Συμπληρώνουμε τα υποχρεωτικά πεδία όπως φαίνεται στην παρακάτω εικόνα (Εικόνα 51). Τα πεδία που θα συμπληρωθούν είναι τα εξής:

- **Name:** Όνομα βασιλείου
- **Class Name:** Επιλογή class name (ποιον τρόπο αποθήκευσης δεδομένων θα χρησιμοποιήσουμε file, ldap, certificate, jdbc)
- **JAAS Context:** Το module που θα χρησιμοποιηθεί για το login. *Πιο αναλυτικά πληροφορίες στο φάκελο του GlassFish Server, στην τοποθεσία: GlassFish5.0\glassfish\domains\domain1\config\login.config*
- **JNDI:** Το όνομα του JDBC Resource.
- **User Table:** Ο πίνακας στον οποίο είναι αποθηκευμένα τα στοιχεία σύνδεσης των χρηστών της εφαρμογής.
- **User Name Column:** Η στήλη η οποία αντιστοιχεί στο username.
- **Password Column:** Η στήλη η οποία αντιστοιχεί στο password.
- **Group Table:** ο πίνακας στον οποίο υπάρχει η στήλη με τους ρόλους.
- **Group Name Column:** το όνομα της στήλης στην οποία αποθηκεύονται τα ονόματα των ρόλων.
- **Password Encryption Algorithm:** Ο αλγόριθμος με τον οποίο θα αποθηκεύονται οι κωδικοί στην βάση δεδομένων, το default είναι SHA-256.



Εικόνα 52. GlassFish Realm

Authentication

Τώρα έχει δημιουργηθεί βασίλειο και υπάρχει σύνδεση με την βάση ακολουθεί η δημιουργία της αυθεντικοποίησης.

Η αυθεντικοποίηση θα γίνει με form-based authentication ώστε να υπάρχει η ευελιξία τροποποίησης της σελίδας σύνδεσης χρήστη. Με αυτόν τον τρόπο είναι δυνατό να εφαρμοστεί το πρότυπο (template) και στην σελίδα σύνδεσης χρήστη.

Με form-based authentication πρέπει να δημιουργηθεί ένα JSF view για την σύνδεση και ένα για περίπτωση σφάλματος, στην παρούσα εφαρμογή είναι αντίστοιχα το login.xhtml και loginError.xhtml.

Αρχικά για την αυθεντικοποίηση πρέπει να οριστεί ο τρόπος αυθεντικοποίησης, το βασίλειο με τους χρήστες, και οι σελίδες σύνδεσης και σφάλματος σύνδεσης. Αυτό γίνεται με τον παρακάτω κώδικα στο αρχείο web.xml.

```
<login-config>
   <auth-method>FORM</auth-method>
   <realm-name>database-realm</realm-name>
   <form-login-config>
      <form-login-page>/login.xhtml</form-login-page>
      <form-error-page>/loginError.xhtml</form-error-page>
```

```
    </form-login-config>
</login-config>
```

Στη συνέχεια υπάρχουν 2 τρόποι να υλοποιηθεί το JSF view login.xhtml για την σύνδεση του χρήστη. Ο πρώτος τρόπος είναι να διαχειρίζεται πλήρως την σύνδεση ο GlassFish Server, αυτό γίνεται χρησιμοποιώντας την παρακάτω φόρμα (Εικόνα 46). Το αίτημα της φόρμας πρέπει να είναι προς j_security_check με παραμέτρους j_username και j_password. Η σελίδα σφάλματος μπορεί να είναι οτιδήποτε.

```
<form method="POST" action="j_security_check">
    <div>
        User name: <input type="text" name="j_username"/>
        Password: <input type="password" name="j_password"/>
        <input type="submit" value="Login"/>
    </div>
</form>
```

Εικόνα 53. Glassfish authentication login form

Ο δεύτερος τρόπος, ο οποίος χρησιμοποιείται στην παρούσα εφαρμογή είναι να δημιουργηθεί ένα CDI managed bean το οποίο θα επεξεργαστεί το αίτημα της φόρμας και θα στείλει η κλάση αυτή το αίτημα για σύνδεση. Με αυτόν τον τρόπο υπάρχει μεγαλύτερη ευελιξία.

Επέλεξα τον δεύτερο τρόπο ώστε να ορίσω ότι αν δεν πετύχει η σύνδεση (λάθος δεδομένα) να ξαναεμφανίζεται η ίδια σελίδα σύνδεσης με ένα μήνυμα σφάλματος αντί για μία σελίδα σφάλματος.

Σε αυτήν την περίπτωση η φόρμα δε χρειάζεται τις δεσμευμένες λέξεις απλά σύνδεση με ιδιότητες για το όνομα χρήστη και τον κωδικό καθώς και με μία μέθοδο για την υποβολή της φόρμας. Ακολουθεί ο κώδικας της φόρμας στην εικόνα με σημειωμένα τα σημαντικά στοιχεία.

```
<h:form >
    <div class="form-component">
        <p:outputLabel for="username" value="#{msgs.UsernameLabel}" class="form-label"/>
        <p:inputText id="username" value="#{authenticationBean.username}" placeholder="#{msgs.Username}"
                     required="true"
                     requiredMessage="#{msgs.UsernameRequired}"/>
    </div>
    <div class="form-component">
        <p:outputLabel for="password" value="#{msgs.PasswordLabel}" class="form-label"/>
        <p:inputText id="password" value="#{authenticationBean.password}" placeholder="#{msgs.Password}"
                     required="true" type="password"
                     requiredMessage="#{msgs.PasswordRequired}"/>
    </div>
    <p:commandButton type="Submit" value="#{msgs.SignIn}" action="#{authenticationBean.login()}" ajax="false"/>
</h:form>
```

Εικόνα 54. Κώδικας της φόρμας σύνδεσης

Η κλάση που διαχειρίζεται την σύνδεση του χρήστη ονομάζεται AuthenticationBean το οποίο είναι SessionScoped και η μέθοδος για την σύνδεση είναι η παρακάτω:

```
public String login() {
    try {
            getRequest().login(getUsername(), getPassword());
    } catch (ServletException e) {
            Message.addWarnMessage("Login Failed", "Wrong Credentials");
            username = null;
```

```
            password = null;
            return "/login.xhtml?faces-redirect=true";
        }
        return "/member/profile.xhtml?faces-redirect=true";
    }

    private HttpServletRequest getRequest() {
        FacesContext context = FacesContext.getCurrentInstance();
        return (HttpServletRequest) context.getExternalContext().getRequest();
    }
```

Σε αυτό το σημείο ο μηχανισμός αυθεντικοποίησης έχει δημιουργηθεί. Στο βασικό μενού της εφαρμογής ύστερα προστίθεται και ένα κουμπί για την αποσύνδεση του χρήστη η οποία γίνεται μέσω της παρακάτω μεθόδου AuthenticationBean#logout():

```
    public String logout() {
        try {
            getRequest().logout();
            username = null;
            password = null;
        } catch (ServletException e) {
            Message.addErrorMessage("Logout Failed", "Please contact us if you
see this message");
        }
        return "/index.xtml?faces-redirect=true";
    }

    private HttpServletRequest getRequest() {
        FacesContext context = FacesContext.getCurrentInstance();
        return (HttpServletRequest) context.getExternalContext().getRequest();
    }
```

Security Constraint

Στην Java EE είναι εφικτό να περιοριστεί η πρόσβαση σε συγκεκριμένες σελίδες και πόρους ανάλογα τον ρόλο που έχει κάποιος χρήστης χρησιμοποιώντας περιορισμούς ασφάλειας (security constraint). Αυτό το κομμάτι αφορά την εξουσιοδότηση που έχει ο χρήστης για συγκεκριμένα JSF views. Ο περιορισμός επιτυγχάνετε προσθέτοντας xml tags στο web.xml αρχείο.

Τα επόμενα στοιχεία μπορούν να περιέχονται σε ένα security constraint.

- **Web resource collection**: Ορίζοντας ένα URL pattern για τον περιορισμό, π.χ. "/admin/*", σημαίνει πως θα εμποδίζεται η πρόσβαση σε όσα URL έχουν αυτό το μοτίβο.
- **Authorization constraint**: Ορίζει ποιοι ρόλοι χρηστών θα έχουν πρόσβαση στους πόρους στους περιορισμένους πόρους.
- **User data constraint:** Μπορεί να επιλεχθεί πως θα προστατεύονται τα δεδομένα όταν μεταφέρονται μεταξύ client και server. Οι επιλογές confidential και integral απαιτούν την ύπαρξη SSL ώστε να επιτρέπεται η μεταφορά δεδομένων, ενώ η επιλογή none δεν έχει κάποια απαίτηση. Στην υλοποίηση της εφαρμογής αυτής θα παραλειφθεί αυτού του τύπου ο περιορισμός.

Ακολουθεί ένα παράδειγμά από την εφαρμογή για περιορισμό σε κάποια JSF view, ώστε να έχει πρόσβαση μόνο ο διαχειριστής. Για διευκόλυνση και καλύτερη οργάνωση οι σελίδες στις οποίες πρέπει να έχει πρόσβαση μόνο ο διαχειριστής είναι στον ίδιο φάκελο

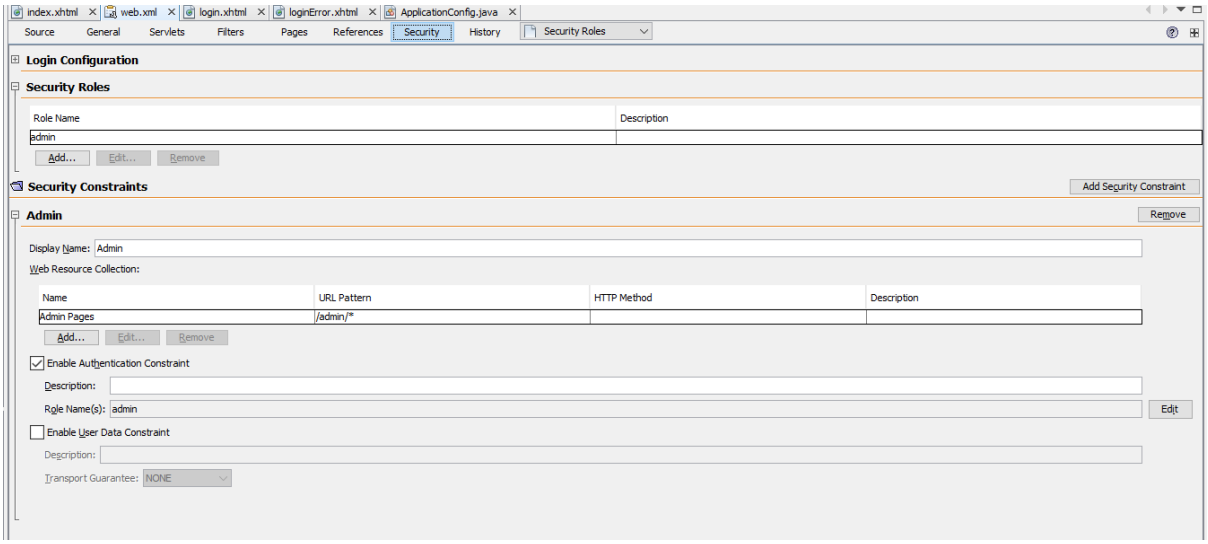
με όνομα admin. Τα xml tags που χρησιμοποιούνται είναι απαραίτητα για να λειτουργήσει σωστά ο περιορισμός αλλιώς εμφανίζεται μήνυμα λάθους.

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Admin Pages</web-resource-name>
        <url-pattern>/admin/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>
```

Με τον ίδιο τρόπο δημιουργείται και ο περιορισμός για τις σελίδες των βιβλιοθηκάριων, στις οποίες θα έχουν πρόσβαση όσοι έχουν το ρόλο librarian ή το ρόλο admin. Αυτό γίνεται προσθέτοντας ένα ακόμα xml tag <auth-constraint> για το δεύτερο ρόλο χρήστη που θα έχει πρόσβαση στις σελίδες.

Αντίστοιχα στις σελίδες των μελών θα έχουν πρόσβαση και οι 3 ρόλοι.

Στο Netbeans (Έκδοση 12.1) υπάρχει επιλογή από τα tabs “security” στο αρχείο web.xml, ώστε να τεθούν από εκεί περιορισμοί για διευκόλυνση, όπως επίσης υπάρχουν και ρυθμίσεις για τον τρόπο αυθεντικοποίησης (Εικόνα 45).



Εικόνα 55. Netbeans web.xml security tab

Συμπεράσματα

Με την πτυχιακή εργασία αυτή

- μελέτησα εκτενέστερα τα θέματα απαιτήσεων λογισμικού, ανάλυσης και σχεδιασμού, και τεχνολογιών Java EE, που καλύφθηκαν στα μαθήματα προγραμματισμού του προγράμματος σπουδών του Τμήματος, ιδιαίτερα των Σχεδιασμός Πληροφοριακών Συστημάτων και Τεχνολογία Λογισμικού.
- μελέτησα επί πλέον θέματα τεχνολογιών Java EE, που συνοψίζονται στο Μέρος 1. Ειδικότερα, μελέτησα τεχνικά βιβλία προγραμματισμού Java EE και αφιέρωσα χρόνο στην αναζήτηση λύσεων σε φόρουμ όπως το StackOverflow.
- εφάρμοσα τις έννοιες και τεχνικές στην ανάπτυξη μιας εφαρμογής διαχείρισης βιβλιοθήκης, όπως συνοψίζεται στο Μέρος 2.

Παράρτημα

Ενημέρωση για Συστήματα Διαχείρισης Βιβλιοθήκης

$$TODO

Μελετήθηκαν οι παρακάτω εφαρμογές.
Πριν αρχίσω την υλοποίηση της εφαρμογής είναι σημαντικό να κάνω έρευνα σε παρόμοιες εφαρμογές και πτυχιακές ώστε να έχω ένα καλύτερο αποτέλεσμα. Οι πιο αξιοσημείωτες εφαρμογές που κοίταξα είναι οι παρακάτω.

Educative.io
https://www.educative.io/courses/grokking-the-object-oriented-design-interview/RMlM3NgjAyR

Στην ιστοσελίδα αυτή υπάρχει ένα παράδειγμα αντικειμενοστραφούς σχεδιασμού ενός πληροφοριακού συστήματος βιβλιοθήκης. Το σύστημα που υλοποιείται εδώ έχει ένα πολύ ωραίο και λεπτομερές διάγραμμα κλάσεων για τις περιπτώσεις χρήσης του.

Από εδώ πήρα την ιδέα για χρήση barcodes ώστε να ξεχωρίζουμε τα αντίτυπα του ίδιου βιβλίου. Η δεύτερη ιδέα που μου άρεσε ήταν ότι και το σύστημα είναι ένας δράστης της εφαρμογής, ο οποίος θα μπορούσε να εμφανίζει μηνύματα στο λογαριασμό του κάθε χρήστη εάν κάποιο βιβλίο δεν έχει επιστραφεί μέχρι την ημερομηνία επιστροφής. Θα μπορούσε να χρησιμοποιηθεί και μία στήλη Status ώστε αν κάποιος δεν έχει επιστρέψει κάποιο βιβλίο εντός προθεσμίας να κλειδώνει ο λογαριασμός και να μην μπορεί να δανειστεί άλλα βιβλία μέχρι να επιστραφεί το προηγούμενο.

Η δυνατότητα κράτησης βιβλίου, καθώς και ειδοποιήσεων μέσω email είναι και αυτές αξιοσημείωτες και πιστεύω πρέπει να υπάρχουν στην εφαρμογή.

Amazon.com
www.amazon.com/b?node=283155

Κοιτώντας το χώρο του amazon για τα βιβλία το πρώτο πράγμα που παρατήρησα και μου φάνηκε ενδιαφέρον να προσθέσω στην εφαρμογή μου είναι μια σελίδα ή κριτήριο αναζήτησης με τους πιο περιζήτητους τίτλους. Θα μπορούσα να εμφανίζω κάπου τα βιβλία που δανείζεται περισσότερο ο κόσμος στη βιβλιοθήκη, αν το πάω και ένα βήμα παραπέρα, να το κάνω και για κάθε κατηγορία βιβλίων ξεχωριστά.

Στο amazon επίσης παρατήρησα και μερικά επιπλέον χαρακτηριστικά που θα μπορούσα να προσθέσω για κάθε βιβλίο όπως την γλώσσα. Τέλος παρατήρησα πως καταγράφεται ISBN-10 και ISBN-13 και με μια γρήγορη αναζήτηση είδα πως το ISNB-10 ήταν το στάνταρ μέχρι το 2007 που καθιερώθηκε το ISBN-13 (Πηγή: www.isbn.org/about_ISBN_standard).

Other Java LMS

Οι παρακάτω είναι μερικές από τις εφαρμογές σε Java που κοίταξα ψάχνοντας παρόμοιες υλοποιήσεις συστημάτων βιβλιοθήκης καθώς και πτυχιακές συναδέλφων. Παρατηρώντας

αυτές τις εφαρμογές είδα τις κοινές περιπτώσεις χρήσεις που χρησιμοποιούν καθώς και τον τρόπο με τον οποίο μοντελοποιούν τα δεδομένα τους στις κλάσεις και στις βάσεις. Μου έδωσαν μια πηγή έμπνευσης για τον σχεδιασμό της εφαρμογής μου αλλά προφανώς ο τελικός σχεδιασμός που υλοποιώ είναι βάσει της δικής μου ανάλυσης.

Υλοποιήσεις

1. https://www.edureka.co/blog/library-management-system-project-in-java
2. https://code-projects.org/simple-library-management-system-in-java-with-source-code-2/
3. https://github.com/OSSpk/Library-Management-System-JAVA
4. https://codebun.com/library-management-system-project-in-java-with-source-code/
5. https://itsourcecode.com/free-projects/java-projects/library-management-system-java-project-with-source-code/
6. https://www.javatpoint.com/library-management-system-in-java-swing

Πτυχιακές Εργασίες:

1. Παπαδανέλλης Γ. (2007). Διαδικτυακή Εφαρμογή Ηλεκτρονική Δανειστική Βιβλιοθήκη. Πτυχιακή Εργασία. Τμήμα Εφαρμοσμένης Πληροφορικής και Πολυμέσων. ΤΕΙ Κρήτης.
2. Μπριασούλη Α. (2014). Σύστημα Δανειστικής Βιβλιοθήκης. Μεταπτυχιακή Διατριβή. Τμήμα Πληροφορικής. Πανεπιστήμιο Πειραιώς.
3. Κυπριάδου Α., Γκούμας Σ, Συμεωνίδης Σ. (2018) Σχεδίαση και υλοποίηση ενός Πληροφοριακού Συστήματος για μια Ακαδημαϊκή Δανειστική Βιβλιοθήκη. Ηλεκτρονικό Περιοδικό Επιστήμης και Τεχνολογίας. Πανεπιστήμιο Δυτικής Αττικής.

Αναφορές

- Geary D., Horstmann, C.S. (2010) "Core JavaServer Faces", 3rd edition, Pearson Education.
    - σημαντική διόρθωση σφαλμάτων του βιβλίου
- Keith M., Schincariol M., Nardone M. (2018) "Pro JPA 2 in Java EE 8: An In-Depth Guide to Java Persistence API", 3rd edition, Apress.
    - κώδικας του βιβλίου - δεν έχει κώδικα για το Κεφάλαιο 13, αλλά υπάρχει το ολοκληρωμένο παράδειγμα του Sylvain Autran, βλ. παρακάτω.
- Oracle (2014) “Java Platform, Enterprise Edition, The Java EE Tutorial
    - κώδικας του παραπάνω tutorial περιλαμβάνεται στο κατέβασμα του NetBeans/Glassfish Server και είναι ένας φάκελος all_javaee-tutorial.
- Scholtz B., Tijms A. (2018) "The Definitive Guide to JSF in Java EE 8", Apress.
- Toby J. Teorey, Sam S. Lightstone, Tom Nadeau, H.V. Jagadish (2011) "Database Modeling and Design, Fifth Edition Logical Design", Morgan Kaufmann
- Siva K., Reddy P. (2013) "PrimeFaces Beginners Guide", Packt Publishing
- PrimeFaces 10.0.0 Showcase: https://www.primefaces.org/showcase
- PrimeFaces 10.0.0 Documentation: https://primefaces.github.io/primefaces/10_0_0/#/

Αναφορές σε άρθρα από το Stack Overflow που με βοήθησαν:

- Database Design for users and profile: https://stackoverflow.com/questions/19755056/mysql-database-design-separate-tables-for-users-and-profile
- Database Design for users and profile: https://stackoverflow.com/questions/3889797/mysql-user-profile-details-table-setup-best-practice

- JSF page reflect date incorrectly: https://stackoverflow.com/questions/6244005/jsf-page-reflects-date-incorrectly-1-day-shifted
- How to use enum values in f:selectItems: https://stackoverflow.com/questions/8229638/how-to-use-enum-values-in-fselectitems
- PrimeFaces process/update JSF execute/render: https://stackoverflow.com/questions/25339056/understanding-primefaces-process-update-and-jsf-fajax-execute-render-attributes