Σύγχρονη παρουσιάση της πλατφόρμας ανάπτυξης λογισμικού **Java** και των δημοφιλέστερων σχετικών βιβλιοθηκών και **framework**

Μπακάλης Δημήτριος - 40559

Εισηγητής : Δρ. Γεώργιος Πρεζεράκος - Καθηγητής

Μηχανικών Πληροφορικής και Υπολογιστών
Πανεπιστήμιο Δυτικής Αττικής
Φεβρουάριος 2019
Αθήνα

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σύγχρονη παρουσιάση της πλατφόρμας ανάπτυξης λογισμικού **Java** και των δημοφιλέστερων σχετικών βιβλιοθηκών και **framework**

Δημήτριος Ι. Μπακάλης
Α.Μ. 40559

Εισηγητής: Δρ. Γεώργιος Πρεζεράκος, Καθηγητής

Εξεταστική Επιτροπή: ————————

Ημερομηνία Εξέτασης: ————————

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Μπακάλης Δημήτριος, του Ιωάννη, με αριθμό μητρώου 40559. φοιτητής του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών του Πανεπιστημίου Δυτικής Αττικής, πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω: «Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο. Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης. Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.

Περίληψη

Σκοπός της εργασίας αυτής είναι η μελέτη και παρουσίαση των κυριότερων τεχνολογιών που σχετίζονται με την πλατφόρμα ανάπτυξης λογισμικού Java, μιας εκ των σημαντικότερων, δημοφιλέστερων και πιο ολοκληρωμένων πλατφόρμων ανάπτυξης λογισμικού που υπάρχουν. Στόχος είναι η παρουσίαση των σημαντικότερων δυνατοτήτων που προσφέρει η κάθε τεχνολογία. Στα πλαίσια αυτής της εργασίας θα δημιουργηθεί λογισμικό - παράδειγμα, το οποίο θα επιδεικνύει πολλές από τις λειτουργίες διευκόλυνσης της ανάπτυξης που προσφέρουν αυτές οι τεχνολογίες.

Abstract

The purpose of this thesis is to study and present the most important technology tools affiliated to the Java software development platform, one of the most significant, complete and popular software development platforms. Its target is to show the most notable features of each technology mentioned. Furthermore, a piece of software will be developed, in which most of the features these technology tools offer to make software development easier will be presented.

Περιεχόμενα

Κεφάλαιο 1

Η γλώσσα προγραμματισμού Java

1.1 Ιστορία της γλώσσας

Η γλώσσα προγραμματισμού Java δημιουργήθηκε το 1991 από 5 εργαζόμενους της εταιρίας Sun Microsystems, τους James Gosling, Patrick Naughton, Chris Warth, Mike Sheridan και Ed Frank. Το αρχικό όνομα της γλώσσας ήταν Oak, αλλά το 1995 αναγκάστηκαν να το αλλάξουν λόγω έλλειψης δικαιωμάτων χρήσης.

Την εποχή που δημιουργήθηκε η Java, στον χώρο της ανάπτυξης λογισμικού μεσουρανούσαν οι γλώσσες C και C++, οι οποίες είχαν φέρει μια επανάσταση στον τρόπο με τον οποίο αυτή γινόταν και θεωρούνταν σαν ικανές να δώσουν λύση στο μεγαλύτερο ποσοστό των προβλημάτων της εποχής.
Η διάδοση όμως της χρήσης του ίντερνετ έφερε την ανάγκη μιας γλώσσας, τα προγράμματα τα οποία θα ήταν γραμμένα σε αυτήν να μπορούν να τρέχουν σε πολλά διαφορετικά υπολογιστικά συστήματα, τα οποία πιθανώς θα είχαν διαφορετικά λειτουργικά συστήματα. Τα προγράμματα που ήταν γραμμένα στις προηγούμενες διαδεδομένες γλώσσες έπρεπε να μεταγλωττιστούν με την χρήση ενός μεταγλωττιστή ο οποίος δημιουργούσε αντικείμενο πρόγραμμα προορισμένο για συγκεκριμένο τύπο λειτουργικού συστήματος.

Η γλώσσα προγραμματισμού Java δημιουργήθηκε έχοντας συγκεκριμένους στόχους:

- Αντικειμενοστραφής - Όλες οι εντολές της Java βρίσκονται μέσα σε κλάσεις.
- Γνώριμη - Η γλώσσα Java χρησιμοποιεί το συντακτικό της C/C++.
- Ασφαλής - Οι μηχανισμοί της πλατφόρμας προστατεύουν τις εφαρμογές από εξωτερική παρέμβαση.
- Φορητή - Ο μεταγλωττισμένος κώδικας Java τρέχει ώς έχει σε όλα τα λειτουργικά συστήματα.

- Αποδοτική - Τα εργαλεία της πλατφόρμας έχουν σχεδιαστεί έτσι ώστε να μεγιστοποιούν την ταχύτητα εκτέλεσης.
- Πολυνηματική - Τα εργαλεία της γλώσσας είναι σχεδιασμένα να υποστηρίζουν την εκτέλεση πολλών νημάτων.

Η πλατφόρμα Java κυκλοφορεί σε 3 διαφορετικές εκδόσεις, καθεμία με το σκοπό της. Αυτές οι εκδόσεις είναι:

- Java Standard Edition - Η βασική έκδοση της γλώσσας. Περιέχει τα εργαλεία που χρειαζόμαστε για να αναπτύξουμε εφαρμογές γραμμένες σε Java.
- Java Enterprise Edition - Ένα σύνολο από εργαλεία που βοηθούν στην ανάπτυξη εφαρμογών μεγάλης κλίμακας.
- Java Micro Edition - Ένα σύνολο από εργαλεία που βοηθούν στην χρήση της γλώσσας σε μικροελεγκτές και φορητές συσκευές.

Το 2008 η Google επέλεξε την Java σαν γλώσσα προγραμματισμού των εφαρμογών στο νέο της λειτουργικό σύστημα για κινητά τηλέφωνα, Android, δημιουργώντας το πρώτο Android SDK. Σήμερα, έχουν δημιουργηθεί εργαλεία για την δημιουργία εφαρμογών για το Android κάνοντας χρήση άλλων τεχνολογιών, αλλά ακόμα οι περισσότερες εφαρμογές γράφονται σε Java.

Σήμερα, προγράμματα γραμμένα στη γλώσσα Java τρέχουν σε παραπάνω από ένα δισεκατομμύριο συσκευές. Αυτό φέρνει την Java ψηλά στη λίστα με τις σημαντικές και δημοφιλείς γλώσσες προγραμματισμού.

1.2 Η έκδοση Java Platform, Standard Edition

Η έκδοση Java Standard Edition (SE) περιλαμβάνει τα εργαλεία, τα οποία χρησιμοποιούμε για να αναπτύξουμε και εκτελέσουμε προγράμματα γραμμένα στη γλώσσα Java.
Υπεύθυνος για την ανάπτυξη και την διανομή αυτών των εργαλείων είναι η Oracle, η εταιρία στην οποία ανήκει η Java, η οποία εκδίδει πρότυπα για υλοποίηση αυτών των εργαλείων από τρίτους, αλλά και τις δικές τις υλοποιήσεις. Αυτή την στιγμή η πλατφόρμα Java SE βρίσκεται στην έκδοση 11.

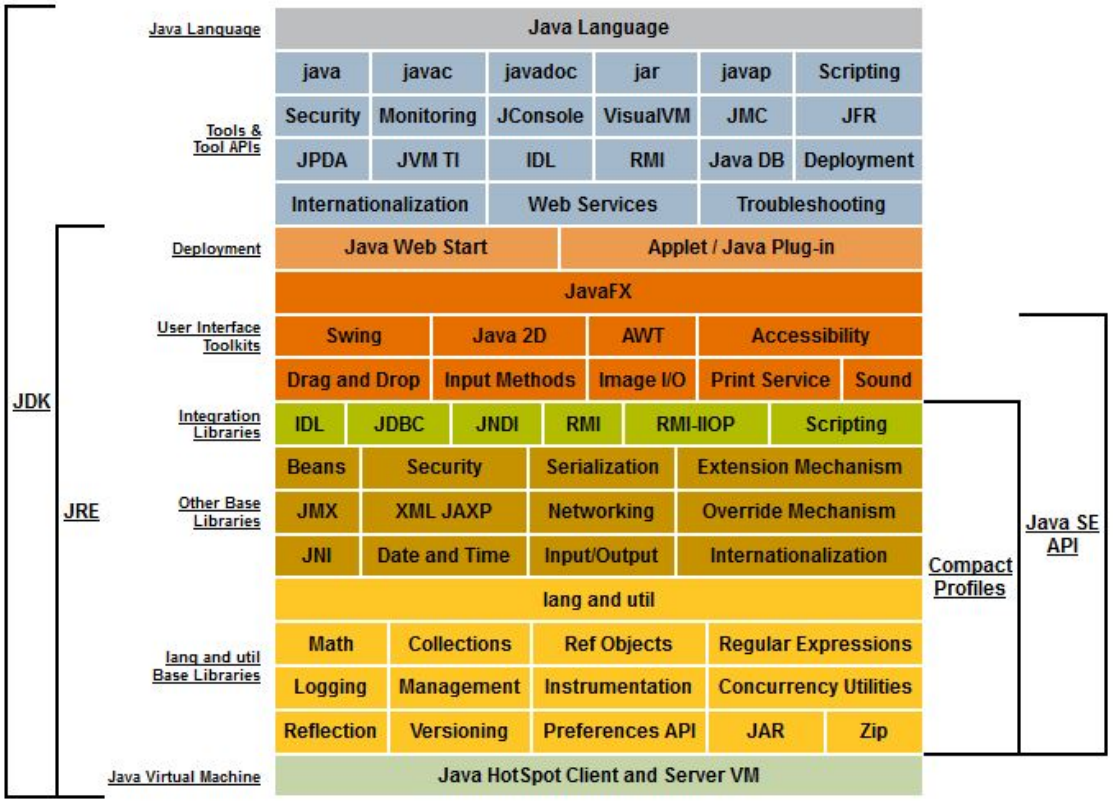

Figure 1.1: Χάρτης της πλατφόρμας Java SE

Το εργαλείο με το οποίο εκτελούμε προγράμματα γραμμένα σε Java ονομάζεται Java Runtime Environment (JRE) και αποτελείται από δύο βασικά μέρη. Το Java Virtual Machine (JVM) και τις βιβλιοθήκες της γλώσσας (Java SE API).

Το Java Virtual Machine είναι ίσως το σημαντικότερο εργαλείο της πλατφόρμας. Πρόκειται για ένα πρόγραμμα που εξομοιώνει την λειτουργία ενός υπολογιστή. Έχει το δικό του ρεπερτόριο εντολών, διαχειρίζεται την μνήμη. Η μεταγλώττιση ενός προγράμματος Java δημιουργεί αυτό που λέμε bytecode. Το JVM εκτελεί αυτόν τον κώδικα μετατρέποντάς τον στο αντικείμενο πρόγραμμα για το συγκεκριμένο σύστημα στο

οποίο είναι εγκατεστημένο. Για κάθε διαφορετικό σύστημα, υπάρχει μια διαφορετική υλοποίηση του JVM, όλες όμως οι υλοποιήσεις εκτελούν τον κώδικα που προκύπτει από την μεταγλώττιση ενός προγράμματος σε Java.
Το Java Virtual Machine διαθέτει επίσης ένα μηχανισμό που ονομάζεται Garbage Collector, ο οποίος είναι υπεύθυνος για την διαχείριση της μνήμης. Η διαχείριση της μνήμης στις γλώσσες C/C++ γινόταν απευθείας από τον προγραμματιστή, με εντολές. Ο Garbage Collector αναγνωρίζει κομμάτια της μνήμης που δεν χρειάζεται πια και τα απελευθερώνει.
Επίσης το Java Virtual Machine παίζει τον ρόλο του Class Loader και του Linker στην εκτέλεση ενός προγράμματος, ενώνει δηλαδή τον κώδικα του προγράμματος με τις κλήσεις σε εξωτερικές βιβλιοθήκες και τον ετοιμάζει για εκτέλεση.

Το άλλο πολύ σημαντικό κομμάτι του Java Runtime Environment είναι το Java SE API. Πρόκειται για το σύνολο των βιβλιοθηκών που έχει δημιουργήσει η εταιρία και αντιμετωπίζουν αρκετά κύρια προβλήματα που σχετίζονται με την ανάπτυξη εφαρμογών στη γλώσσα. Κάποια πακέτα από αυτές τις βιβλιοθήκες περιέχουν κλάσεις απαραίτητες για την λειτουργία της γλώσσας όπως π.χ. τύπους δεδομένων. Άλλα πακέτα περιέχουν κλάσεις οι οποίες υλοποιούν κλήσεις συστήματος για χρήση του λειτουργικού συστήματος, άλλα πακέτα διαχειρίζονται αρχεία, άλλα περιέχουν βιβλιοθήκες γραφικών. Η έκδοση περιέχει μια ευρεία γκάμα από έτοιμο κώδικα για διάφορες εργασίες, που επεκτείνεται συνεχώς.

Το εργαλείο με το οποίο αναπτύσσουμε προγράμματα σε Java ονομάζεται Java Development Kit (JDK) και είναι ένα υπερσύνολο του JRE, όπως φαίνεται και στον παραπάνω χάρτη. Περιλαμβάνει δηλαδή όλα τα εργαλεία που είδαμε παραπάνω, καθώς και εργαλεία ανάπτυξης όπως μεταγλωττιστές, εργαλεία εκσφαλμάτωσης και εργαλεία που βοηθούν στην έκδοση και υπογραφή των εφαρμογών (building - packaging).

1.3 Η έκδοση Java Platform, Enterprise Edition

Η έκδοση Java Enterprise Edition (EE) είναι ένα σύνολο από πρότυπα επίλυσης πολλών προβλημάτων που σχετίζονται με τις εφαρμογές εταιρειών μεγάλης κλίμακας (Enterprise Applications).Υπεύθυνη για την έκδοση αυτών των προτύπων είναι μια οργάνωση που ονομάζεται Java Community Process (JCP), η οποία αποτελείται από ειδικούς εταιριών ανάπτυξης λογισμικού, από οργανώσεις ανοιχτού κώδικα, αλλά και μεμονωμένους προγραμματιστές, οι οποίοι θέλουν να συμβάλλουν στην ανάπτυξη και την προτυποποίηση της πλατφόρμας.

Για κάθε διαφορετικό πρόβλημα που καλείται να αντιμετωπιστεί στην ανάπτυξη Enterprise εφαρμογών, η οργάνωση JCP έχει δημιουργήσει πρότυπα επίλυσης. Αυτά τα πρότυπα ονομάζονται JSR (Java Specification Requests). Σκοπός είναι να δημιουργηθούν βιβλιοθήκες / Runtime Environments που θα επιλύουν τα προβλήματα με κάποιον κοινό / ομοιόμορφο τρόπο.

Η πρώτη έκδοση της Java EE ήταν η Java 2 Enterprise Edition 1.2, η οποία εκδόθηκε τον Δεκέμβριο του 1999 και περιείχε τα εξής πρότυπα:

- Java 2 Platform, Enterprise Edition Specification 1.2
- J2EE Connector Specification 1.0
- Enterprise JavaBeans Specification 1.1
- Enterprise JavaBeans to CORBA Mapping 1.1
- JavaServer Pages Specification 1.1
- Java Servlet Specifications 2.2
- Java Naming and Directory Interface Specification, 1.2.1
- JDBC Specifications, 3.0, 2.1, and Optional Package API (2.0)
- JavaMail API Specification 1.1
- JavaBeans Activation Framework Specification 1.0.1
- Java Transaction API Specification 1.0.1
- Java Transaction Service Specification 1.1

Η τελευταία έκδοση που έχει κυκλοφορήσει είναι η Java EE 8, η οποία εκδόθηκε τον Αύγουστο του 2017. Νέα πρότυπα αυτής της έκδοσης είναι τα παρακάτω:

- JSR 366 – Java EE 8 Platform
- JSR 365 – Contexts and Dependency Injection (CDI) 2.0
- JSR 367 – The Java API for JSON Binding (JSON-B) 1.0
- JSR 369 – Java Servlet 4.0
- JSR 370 – Java API for RESTful Web Services (JAX-RS) 2.1
- JSR 372 – JavaServer Faces (JSF) 2.3
- JSR 374 – Java API for JSON Processing (JSON-P)1.1
- JSR 375 – Java EE Security API 1.0
- JSR 380 – Bean Validation 2.0
- JSR 250 – Common Annotations 1.3
- JSR 338 – Java Persistence 2.2
- JSR 356 – Java API for WebSocket 1.1
- JSR 919 – JavaMail 1.6

Πολλά από αυτά τα πρότυπα περιγράφουν εργαλεία η χρήση των οποίων γίνεται σε συνεργασία με λογισμικό το οποίο ονομάζεται Application Server, ενώ άλλα περιγράφουν λειτουργίες που πρέπει να υλοποιεί ο Application Server. Υπάρχουν 3 διαφορετικές υλοποιήσεις Application Server που είναι συμβατές και υλοποιούν τα πρότυπα της έκδοσης 8 της Java EE. Αυτές είναι οι:

α) GlassFish Server Open Source Edition 5.0 της Oracle
β) WebSphere Application Server Liberty Version 18.0.0.2 της IBM και
γ) Wildfly 14.x της Red Hat.

Κεφάλαιο 2

Η εξέλιξη της χρήσης της Java στο web

2.1 Η αρχή της ανάπτυξης δυναμικών ιστοσελίδων

Στην αρχή της χρήσης του Web, οι πρώτοι Web Servers είχαν την εξής λειτουργικότητα: Δέχονταν αιτήσεις HTTP, οι οποίες ζητούσαν κάποιο συγκεκριμένο πόρο / αρχείο. Ο Server έψαχνε το σύστημα αρχείων του για το συγκεκριμένο πόρο και, αν τον έβρισκε,έστελνε την κατάλληλη HTTP απάντηση η οποία περιείχε τα περιεχόμενα του αρχείου. Αν όχι, έστελνε την αντίστοιχη απάντηση πως ο ζητούμενος πόρος δεν βρέθηκε. Το περιεχόμενο αυτού του πόρου ήταν στατικό, δηλαδή τα περιεχόμενα που ο Web Server έστελνε στον χρήστη υπήρχαν στο αρχείο πρίν από την επεξεργασία της αίτησης και δεν επηρεάζονταν από αυτήν.

Το επόμενο βήμα στην ανάπτυξη των ιστοσελίδων ήταν η δημιουργία αυτού που ονομάζουμε δυναμικές ιστοδελίδες. Ιστοσελίδες, το περιεχόμενο των οποίων δημιουργείται κατά την εκτέλεση ενός προγράμματος στον Server, κατά την επεξεργασία της αίτησης HTTP.

Το πρωτόκολλο HTTP είναι ένα πρωτόκολλο επικοινωνίας που βασίζεται σε κείμενο. Τα προγράμματα που συντάσσουν και αποστέλλουν τις αιτήσεις, καθώς και απεικονίζουν κατάλληλα τα περιεχόμενα των αρχείων που εμπεριέχονται στις απαντήσεις των αιτήσεων ονομάζονται φυλλομετρητές (Web Browsers). Τα αντίστοιχα προγράμματα που δέχονται τις αιτήσεις, συντάσσουν και αποστέλλουν τις απαντήσεις ονομάζονται Web Servers. Στην περίπτωση των στατικών ιστοσελίδων όπως είπαμε, αυτά τα προγράμματα απλά αντέγραφαν στο κείμενο της απάντησης τα περιεχόμενα του αρχείου που είχε ζητηθεί. Στην περίπτωση όμως των δυναμικών ιστοσελίδων τα προγράμματα αυτά αναλαμβάνουν να καλέσουν κώδικα ο οποίος εκτελούμενος δημιουργεί το περιεχόμενο που θα αποσταλεί στην απάντηση.

2.2 Η ανάπτυξη δυναμικών ιστοσελίδων με χρήση της Java - Servlets

Η έκδοση Java Enterprise Edition, από την αρχική της κυκλοφορία περιείχε πρότυπο για την δημιουργία των προγραμμάτων που περιγράψαμε στην εισαγωγή. Τα προγράμματα αυτά ονομάζονται Servlet Containers και είναι βασικό κομμάτι των Web Server που εκδίδουν διαδικτυακές εφαρμογές γραμμένες στην γλώσσα Java. Η βασική λειτουργία αυτών των προγραμμάτων είναι, δεχόμενα μια αίτηση HTTP, να καλούν τον κατάλληλο κώδικα της διαδικτυακής εφαρμογής, ο οποίος αναλαμβάνει να δημιουργήσει το περιεχόμενο της απάντησης HTTP. Αυτός ο κώδικας της διαδικτυακής εφαρμογής ονομάζεται Servlet.

Κάθε διαδικτυακή εφαρμογή διαθέτει ένα αρχείο, το web.xml το οποίο υποδεικνύει στον Servlet Container ποιό Servlet θα καλέσει για κάθε διαφορετική διεύθυνση αίτησης.

Για να δημιουργήσουμε ένα Servlet στην διαδικτυακή μας εφαρμογή, αρκεί να δημιουργήσουμε μια κλάση η οποία θα κάνει extend την κλάση javax.servlet.http.HttpServlet, η οποία περιέχεται στο SDK της πλατφόρμας Java EE. Η συγκεκριμένη κλάση διαθέτει τις παρακάτω μεθόδους:

- init() - Αυτή η μέθοδος καλείται όταν αρχικοποιείται το Servlet.
- destroy() - Αυτή η μέθοδος καλείται όταν καταστρέφεται το Servlet.
- doGet() - Αυτή η μέθοδος καλείται όταν ο Web Server δέχεται μια αίτηση με την μέθοδο GET του πρωτοκόλλου HTTP με αποδέκτη το συγκεκριμένο Servlet.
- doPost() - Αυτή η μέθοδος καλείται όταν ο Web Server δέχεται μια αίτηση με την μέθοδο POST του πρωτοκόλλου HTTP με αποδέκτη το συγκεκριμένο Servlet.
- doPut() - Αυτή η μέθοδος καλείται όταν ο Web Server δέχεται μια αίτηση με την μέθοδο PUT του πρωτοκόλλου HTTP με αποδέκτη το συγκεκριμένο Servlet.
- doDelete() - Αυτή η μέθοδος καλείται όταν ο Web Server δέχεται μια αίτηση με την μέθοδο DELETE του πρωτοκόλλου HTTP με αποδέκτη το συγκεκριμένο Servlet.
- service() - Αυτή η μέθοδος καλείται όταν φτάσει μια αίτηση με αποδέκτη το συγκεκριμένο Servlet, πριν από την κλήση της κατάλληλης μεθόδου doXXX().
- getServletInfo() - Αυτή η μέθοδος δίνει πληροφορίες για το υπάρχον Servlet.

Την εκτέλεση αυτών των μεθόδων την κατάλληλη στιγμή αναλαμβάνει ο Servlet Container, ανάλογα με την αίτηση που δέχεται. Επίσης παρέχει σε αυτές της μεθόδους, ώς παραμέτρους, αντικείμενα που διαχειρίζονται την αίτηση αλλά και την απάντηση. Ο

προγραμματιστής υλοποιεί αυτές τις μεθόδους, χρησιμοποιώντας αυτά τα αντικείμενα, για να δημιουργήσει την απάντηση. Πιο συγκεκριμένα οι βασικότερες μέθοδοι που διαθέτει η κλάση javax.servlet.http.HttpServletResponse, αντικείμενο της οποίας παρέχει ώς παράμετρο στις παραπάνω μεθόδους ο Servlet Container κατά την κλήση τους, με την βοήθεια των οποίων δημιουργούμε την απάντηση, είναι οι παρακάτω:

- addCookie(Cookie cookie) - Αυτή η μέθοδος προσθέτει στην απάντηση ένα HTTP Cookie.
- addHeader(String name, String value) - Αυτή η μέθοδος προσθέτει στην απάντηση ένα HTTP Header.
- setStatus(int sc) - Αυτή η μέθοδος ορίζει τον κωδικό κατάστασης HTTP της απάντησης.
- setContentType(String type) - Αυτή η μέθοδος ορίζει τον τύπο των δεδομένων που περιέχει η απάντηση.
- getWriter() - Αυτή η μέθοδος επιστρέφει ένα αντικείμενο της κλάσης java.io.PrintWriter με την χρήση του οποίου γράφουμε το σώμα της απάντησης.

Επίσης στις μεθόδους αυτές παρέχεται και ένα αντικείμενο τις κλάσης javax.servlet.http.HttpServletRequest, το οποίο περιέχει πληροφορίες για την αίτηση. Οι βασικότερες μέθοδοι που διαθέτει αυτή η κλάση είναι οι παρακάτω:

- getCookies() - Αυτή η μέθοδος επιστρέφει τα Cookies που περιείχε η αίτηση.
- getHeaderNames() - Αυτή η μέθοδος επιστρέφει τα ονόματα των HTTP Headers που περιείχε η αίτηση.
- getHeaders(String name) - Αυτή η μέθοδος επιστρέφει τις τιμές του HTTP Header με όνομα την τιμή της παραμέτρου.
- getMethod() - Αυτή η μέθοδος επιστρέφει την μέθοδο με την οποία έγινε η αίτηση.
- getRequestURI() / getRequestURL() - Αυτή η μέθοδος επιστρέφει το URI/URL της αίτησης.
- getContentType() - Αυτή η μέθοδος επιστρέφει τον τύπο των δεδομένων που περιέχονται στην αίτηση.
- getReader() - Αυτή η μέθοδος επιστρέφει ένα αντικείμενο της κλάσης java.io.BufferedReader το οποίο χρησιμοποιείται για να διαβάσει το σώμα της αίτησης.

Η τεχνολογία των Servlets ήταν πολύ χρήσιμη στην αρχή της ανάπτυξης δυναμικών διαδικτυακών εφαρμογών με την χρήση της γλώσσας Java, αλλά η συγγραφή των σελίδων με αυτόν τον τρόπο ήταν ιδιαίτερα επίπονη, απαιτούσε την γνώση της γλώσσας Java, ενώ σκοπός ήταν κυρίως η συγγραφή αρχείων HTML, και έδινε στον προγραμματιστή την πλήρη ευθύνη της συγγραφής της απάντησης για κάθε αίτηση HTTP. Ενώ η λογική των Servlets παραμένει και σήμερα κομβική για την ανάπτυξη διαδικτυακών εφαρμογών, έχουν αναπτυχθεί διάφορα εργαλεία, τα οποία απομακρύνουν την διαδικασία της ανάπτυξης από την συγγραφή των απαντήσεων HTTP, και την κατευθύνουν προς την συγγραφή αρχείων τα οποία τα ίδια τα εργαλεία θα χρησιμοποιήσουν για να συντάξουν την αίτηση.

2.3 Java Server Pages

Το μεγαλύτερο πρόβλημα με την χρήση των Servlets για την δημιουργία δυναμικών ιστοσελίδων ήταν το γεγονός πως ενώ σκοπός ήταν να σταλεί στον browser μια απάντηση που θα περιείχε ένα αρχείο HTML, το αρχείο που έκανε αυτή την δουλειά δεν θύμιζε σε τίποτα κάποιο αρχείο HTML. Όλα τα HTML tags ήταν κλεισμένα σε πολλαπλές εντολές print. Επίσης κομμάτια του παραγόμενου αρχείου HTML που ήταν στατικά, δεν επηρεάζονταν δηλαδή άμεσα από την εκτέλεση του προγράμματος, ήταν υποχρεωτικό να εκτυπωθούν στην αίτηση με την χρήση εντολών Java, αυξάνοντας πολλαπλά τον όγκο των εντολών, ακόμα και αν οι απαραίτητες για την μετατροπή της σελίδας σε δυναμική εντολές ήταν ελάχιστες. Μια λύση σε αυτό το πρόβλημα κλήθηκε να δώσει η ανάπτυξη της τεχνολογίας JSP (Java Server Pages), το πρότυπο της οποίας εκδόθηκε με την πρώτη έκδοση της Java Enterprise Edition.

Η ιδέα πίσω από την δημιουργία των JSPs ήταν να αντιστρέψουμε την λογική της συγγραφής των Servlets, δηλαδή αντί να προσθέτουμε τον κώδικα HTML μέσα σε αρχεία Java, να προσθέσουμε εντολές της Java μέσα σε αρχεία HTML. Οι εντολές αυτές θα τοποθετούνται όπου είναι απαραίτητο για να δημιουργήσουν δυναμικό περιεχόμενο, ενώ το στατικό περιεχόμενο θα μένει ώς έχει.

Οι JSPs βασίζονται στην τεχνολογία των Servlets για να δουλέψουν. Ουσιαστικά όταν καλείται ένα JSP αρχείο, τα εργαλεία της πλατφόρμας το μετατρέπουν στο αντίστοιχο Servlet, το οποίο εκτελείται.

Στις JSPs, οι εντολές τοποθετούνται ανάμεσα στα στοιχεία του αρχείου HTML. Για να ξεχωρίζουν όμως από τα απλά στοιχεία HTML, οι εντολές μας περικλείονται σε ειδικά tags, τα <% ... %>Υπάρχουν διάφορες κατηγορίες εντολών που μπορούν να συμπεριληφθούν σε ένα αρχείο JSP:

α) Οδηγίες: Αυτές οι εντολές παρέχουν πληροφορίες στον JSP Container, τις οποίες χρησιμοποιεί κατά την μετατροπή του JSP σε Servlet, ή κατά την εκτέλεση του αντίστοιχου Servlet. Η σύνταξη μιας εντολής - οδηγίας είναι η παρακάτω:
<%@ τύπος_οδηγίας attribute="value" ..... %>.
Οι διαφορετικοί τύποι των οδηγιών που υπάρχουν είναι οι εξής:

- page - Αυτή η οδηγία ορίζει διάφορες ρυθμίσεις οι οποίες αφορούν την σελίδα, όπως Content-Type, γλώσσα που χρησιμοποιείται, σελίδα σφαλμάτων.
- include - Αυτή η οδηγία χρησιμοποιείται για να προσθέσει στον κώδικά μας κώδικα από κάποιο άλλο αρχείο.
- taglib - Αυτή η οδηγία χρησιμοποιείται για να δώσει στην σελίδα πρόσβαση σε αρχείο βιβλιοθήκης στοιχείων JSP. Αυτά τα στοιχεία μπορούν να χρησιμοποιηθούν μετέπειτα στον κώδικα της σελίδας.

β) Εντολές της γλώσσας Java. Υπάρχουν κι εδώ διάφορες κατηγορίες εντολών που μπορούν να χρησιμοποιηθούν:

- δηλώσεις - Με αυτόν τον τύπο εντολής μπορούμε να δηλώσουμε μεταβλητές ορατές από όλη τη σελίδα. Η σύνταξη μιας δήλωσης είναι <%! int i=0; %>.
- εκφράσεις - Με αυτόν τον τύπο εντολής μπορούμε να γράψουμε εκφράσεις της γλώσσας Java, οι οποίες θα υπολογιστούν και το αποτέλεσμά τους θα εκτυπωθεί στην έξοδο. Η σύνταξη μιας έκφρασης είναι: <%= 3+i %>.
- scriptlets - Ανάμεσα στα tags <% %>μπορούμε να γράψουμε κώδικα Java. Αυτή η σύνταξη ονομάζεται scriptlet (μικρό σενάριο).
- σχόλια - Με αυτόν τον τύπο εντολής μπορούμε να γράψουμε σχόλια τα οποία δεν θα ληφθούν υπ΄ όψιν κατά την μετατροπή του JSP σε Servlet. Η σύνταξη ενός σχολίου είναι: <%– This is a JSP comment %>.

γ) δράσεις. Η εκτέλεση αυτών των εντολών JSP έχει ώς αποτέλεσμα κάποια δράση, π.χ. δημιουργία κάποιου αντικειμένου, μεταφορά σε άλλη διεύθυνση, προσθήκη των περιεχομένων κάποιου αρχείου. Η σύνταξη των εντολών δράσης είναι η εξής:
<jsp:tag_name attribute=”value”>
.....
</jsp:tag_name>
Το πρότυπο JSP περιέχει τα εξής tags δράσης:

- jsp:useBean - Με αυτή την εντολή δημιουργούμε ένα αντικείμενο μιας κλάσης Java. Η σύνταξη μιας τέτοιας εντολής είναι η εξής:
  <jsp:useBean id=”objName” class=”path.to.the.class.ClassName” scope=”page”/>.
- jsp:setProperty - Με αυτή την εντολή δίνουμε τιμή σε μια ιδιότητα ενός αντικειμένου που έχει οριστεί με την χρήση της εντολής jsp:useBean. Η σύνταξη μιας τέτοιας εντολής είναι η εξής:
  <jsp:setProperty id=”objName” property=”name” value=”John Doe”/>.
- jsp:getProperty - Με αυτή την εντολή εκτυπώνουμε την τιμή μιας ιδιότητα ενός αντικειμένου που έχει οριστεί με την χρήση της εντολής jsp:useBean. Η σύνταξη μιας τέτοιας εντολής είναι η εξής:
  <jsp:getProperty id=”objName” property=”name”/>.
- jsp:include - Αυτή η εντολή μοιάζει με την εντολή - οδηγία include , προσθέτει δηλαδή κώδικα από άλλο αρχείο. Η διαφορά είναι πως ενώ η εντολή οδηγία το κάνει κατά την μετατροπή του JSP σε Servlet, η εντολή - δράση το κάνει κατά την επεξεργασία της αίτησης. Η σύνταξη μιας τέτοιας εντολής είναι η εξής:
  <jsp:include page=”/pathTo/pageFile.jsp” flush=”true” />.

- jsp:forward - Με αυτή την εντολή τερματίζουμε την εκτέλεση του τρέχοντος JSP και αρχίζουμε την εκτέλεση ενός νέου. Η σύνταξη μιας τέτοιας εντολής είναι η εξής:
  <jsp:forward page="/pathTo/pageFile.jsp" />.

- jsp:param - Με αυτή την εντολή προσθέτουμε παραμέτρους στην αίτηση που δημιουργείται από την κλήση των εντολών include και forward. Η σύνταξη μιας τέτοιας εντολής είναι η εξής:
  <jsp:param name="name" value="John Doe"/>.

Σε μια JSP το μεγαλύτερο μέρος της λειτουργικότητας υλοποιείται σε κλάσεις της Java, και η κλήση αυτών των λειτουργιών γίνεται μέσω αντικειμένων που είναι προσβάσιμα από την JSP με την χρήση της εντολής jsp:useBean. Υπάρχουν όμως και αντικείμενα τα οποία είναι διαθέσιμα by default σε όλες τις JSPs. Αυτά είναι τα παρακάτω:

- page - Αυτό είναι ένα αντικείμενο μιας υλοποίησης του Interface javax.servlet.jsp.HttpJspPage. Διαθέτει ιδιότητες και μεθόδους αντιπροσωπευτικές για την σελίδα. Μπορεί να προσπελαστεί και με το keyword this.

- request - Αυτό είναι ένα αντικείμενο μιας υλοποίησης του Interface javax.servlet.http.HttpServletRequest . Διαθέτει ιδιότητες και μεθόδους αντιπροσωπευτικές για την αίτηση HTTP.

- response - Αυτό είναι ένα αντικείμενο μιας υλοποίησης του Interface javax.servlet.http.HttpServletResponse . Διαθέτει ιδιότητες και μεθόδους αντιπροσωπευτικές για την απάντηση HTTP.

- pageContext - Αυτό είναι ένα αντικείμενο της κλάσης javax.servlet.jsp.PageContext. Διαθέτει ιδιότητες και μεθόδους αντιπροσωπευτικές για το σύνολο των αντικειμένων και των μεταβλητών που είναι προσπελάσιμα από την σελίδα.

- session - Αυτό είναι ένα αντικείμενο της κλάσης javax.servlet.http.HttpSession. Διαθέτει ιδιότητες και μεθόδους αντιπροσωπευτικές για την HTTP Session.

- application - Αυτό είναι ένα αντικείμενο της κλάσης javax.servlet.ServletContext. Διαθέτει ιδιότητες και μεθόδους αντιπροσωπευτικές για το σύνολο των αντικειμένων και των μεταβλητών που είναι προσπελάσιμα από την διαδικτυακή εφαρμογή.

- out - Αυτό είναι ένα αντικείμενο της κλάσης javax.servlet.jsp.JspWriter. Διαθέτει μεθόδους για να εκτυπώνει περιεχόμενο στην σελίδα.

- config - Αυτό είναι ένα αντικείμενο μιας υλοποίησης του Interface javax.servlet.ServletConfig. Διαθέτει ιδιότητες και μεθόδους που αντιπροσωπεύουν τις ρυθμίσεις που είναι απαραίτητες για την αρχικοποίηση του παραγόμενου Servlet.

- exception - Αυτό το αντικείμενο είναι διαθέσιμο μόνο σε JSPs που καλούνται για να διαχειριστούν σφάλματα. Είναι το αντικείμενο που περιγράφει το αντίστοιχο σφάλμα.

Στην δήλωση ενός αντικειμένου στην JSP, ορίζουμε μια παράμετρο που ονομάζεται scope. Αυτή η παράμετρος ορίζει σε ποιό εύρος θα είναι προσβάσιμο αυτό το αντικείμενο. Default τιμή είναι το page, δηλαδή το αντικείμενο είναι διαθέσιμο για κλήση στο πλαίσιο του αρχείου JSP. Άλλες τιμές είναι οι:

- request: Το αντικείμενο είναι προσβάσιμο σε όλα τα αρχεία που εξυπηρετούν την ίδια αίτηση.
- session: Το αντικείμενο είναι προσβάσιμο σε όλα τα αρχεία κατά το ίδιο HTTP Session.
- application: Το αντικείμενο είναι προσβάσιμο σε όλα τα αρχεία της εφαρμογής.

Με το πέρασμα του χρόνου, αφού η τεχνολογία των JSPs έγινε πιο δημοφιλής, δημιουργήθηκαν βιβλιοθήκες οι οποίες αυτοματοποιούν μεγάλο μέρος του κώδικα Java που χρειαζόμαστε για να αναπτύξουμε διάφορες λειτουργίες. Για να αποκτήσουμε πρόσβαση σε αυτές τις βιβλιοθήκες χρησιμοποιούμε την οδηγία taglib, δίνοντας ώς παραμέτρους τη διεύθυνση στην οποία βρίσκεται το αρχείο που περιγράφει τη βιβλιοθήκη και το πρόθεμα που χρησιμοποιεί η καθεμία για να ξεχωρίσει τα JSP tags της από τα κανονικά HTML tags. Μια πολύ σημαντική βιβλιοθήκη είναι η JSTL (Java Standard Tag Library), υπάρχει όμως μεγάλος αριθμός από βιβλιοθήκες διαθέσιμες για χρήση.

2.4 Σύγχρονες μεθοδολογίες ανάπτυξης - Κατηγορίες Εργαλείων

Όπως είδαμε, το Web, στην πρωταρχική μορφή του αποτελούσε έναν τρόπο μεταφοράς και απεικόνισης στατικής πληροφορίας μέσω διαδικτύου. Η τεχνολογία των Servlets αρχικά, και ένα σύνολο από άλλες τεχνολογίες, προσπάθησαν και κατάφεραν να δώσουν μια πληθώρα από δυνατότητες στις διαδικτυακές εφαρμογές, όπως δημιουργία δυναμικού περιεχομένου, αλληλεπίδραση με το χρήστη και, ολοένα και περισσότερο όσο αναπτύσσεται η τεχνολογία, διαχείριση και ανάλυση μεγάλου όγκου δεδομένων.

Βέβαια, όσο οι δυνατότητες του λογισμικού αυξάνονται, τόσο αυξάνεται και η πολυπλοκότητά του. Αυτό δημιουργεί την ανάγκη εύρεσης τεχνικών, οι οποίες θα διευκολύνουν όχι μόνο την ανάπτυξη του λογισμικού, αλλά και την διασφάλιση της ποιότητάς του, της ασφάλειας κατά την χρήση του, και την συντήρηση και μετέπειτα εξέλιξή του. Κατά το πέρασμα των χρόνων, έχουν δημιουργηθεί αρκετά εργαλεία, βιβλιοθήκες, προγράμματα και Frameworks, τα οποία διευκολύνουν την δουλειά των προγραμματιστών προς αυτές τις κατευθύνσεις.

Πιο συγκεκριμένα, για την διευκόλυνση της ανάπτυξης και της συντήρησης διαδικτυακών εφαρμογών, εφαρμόζεται σε ένα τεράστιο ποσοστό η αρχιτεκτονική MVC (Model - View - Controller). Σκοπός της είναι να διαχωρίσει τον κώδικα που απαιτείται για την δημιουργία των εφαρμογών σε τρία ξεχωριστά μέρη, το καθ΄ ένα με το δικό του σκοπό, ώστε η τροποποίηση κάποιου μέρους και η συντήρηση να είναι ευκολότερη.

Μια από τις σημαντικότερες δυνατότητες των διαδικτυακών εφαρμογών, είναι η διαχείριση και χρήση δεδομένων, μόνιμα αποθηκευμένων σε κάποιο σύστημα βάσης δεδομένων. Έχουν δημιουργηθεί τεχνικές και εργαλεία, τα οποία μας βοηθούν να διαχειριστούμε ευκολότερα, γρηγορότερα και αποδοτικότερα αυτά τα δεδομένα.

Η βάση στην ανάπτυξη οποιασδήποτε εφαρμογής είναι αυτό που λέμε Business Logic. Το κομμάτι δηλαδή της εφαρμογής στο οποίο αναπτύσσεται η λογική και η λειτουργικότητα της εφαρμογής. Καθώς οι απαιτήσεις και η πολυπλοκότητα των εφαρμογών αυξάνονται συνεχώς, η ανάγκη οργάνωσης και διαχωρισμού του κώδικα που υλοποιεί το Business Logic (Modularity), καθώς και η επαναχρησιμοποίηση κώδικα που υλοποιεί λειτουργίες που είναι απαραίτητες σε περισσότερα από ένα σημεία, ή σε διαφορετικές εφαρμογές είναι επιτακτική.

Στις διαδικτυακές εφαρμογές, η απεικόνιση των δεδομένων και ο γενικός οπτικός σχεδιασμός τους, δημιουργείται με γλώσσες οι οποίες δεν ανήκουν στην οικογένεια της πλατφόρμας Java. Η ανάπτυξη των εφαρμογών αυτών με την χρήση των εργαλείων της πλατφόρμας, απαιτεί την χρήση των γλωσσών που δημιουργούν τον οπτικό σχεδιασμό. Έχουν δημιουργηθεί εργαλεία τα οποία επιτρέπουν και διευκολύνουν τη συνεργασία των δύο.

Καθώς η πολυπλοκότητα των εφαρμογών αυξάνεται, ο έλεγχος της ποιότητάς τους και ο όσο το δυνατόν περιορισμός των σφαλμάτων τους γίνεται επίσης πιο πολύπλοκος. Γι΄ αυτό δημιουργήθηκε η ανάγκη ύπαρξης εργαλείων που θα διευκολύνουν αυτή τη διαδικασία.

Καθώς πολλές εφαρμογές, όσο και να διαφέρει ο σκοπός για τον οποίο δημιουργήθηκαν, σε πολλά σημεία αντιμετωπίζουν τις ίδιες ανάγκες, η επαναχρησιμοποίηση ήδη υπάρχοντος κώδικα ο οποίος τις αντιμετωπίζει είναι απαραίτητη. Αυτός είναι και ο λόγος που δημιουργούνται όλα τα εργαλεία, τα οποία εξετάζουμε και μελετούμε. Αυτός ο κώδικας μας δίνεται για να τον προσθέσουμε στην εφαρμογή μας, με την μορφή εξωτερικών αρχείων βιβλιοθηκών. Καθώς οι εφαρμογές γίνονται ολοένα και πιο πολύπλοκες, και η ανάγκη προσθήκης πολλαπλών βιβλιοθηκών γίνεται επιτακτική, η διαδικασία της προσθήκης των καταλλήλων βιβλιοθηκών, καθώς και της τελικής δημιουργίας εκτελέσιμων αρχείων της εφαρμογής μας, γίνεται ολοένα και πιο πολύπλοκη και χρονοβόρα. Γι΄ αυτό, έχουν δημιουργηθεί εργαλεία, τα οποία μας βοηθούν στην αντιμετώπιση πολλών από αυτά τα προβλήματα.

Μιλώντας για επαναχρησιμοποίηση κώδικα, μια πολύ χρήσιμη κατηγορία εφαρμογών είναι οι διαδικτυακές υπηρεσίες. Πρόκειται για μια κατηγορία εφαρμογών, οι οποίες θέτουν τον εαυτό τους διαθέσιμο στο διαδίκτυο, και άλλες εφαρμογές μπορούν να τις χρησιμοποιήσουν, χωρίς να απαιτείται κάποια αλλαγή στον κώδικά τους, για την προσθήκη αλληλεπίδρασης με κάποια νέα εφαρμογή.

Στα επόμενα κεφάλαια, θα παρουσιάσουμε τεχνικές και εργαλεία που ασχολούνται με όλα αυτά τα ζητήματα που αναφέρουμε, όπως αντιμετωπίζονται με την χρήση της πλατφόρμας Java.

Κεφάλαιο 3

Web MVC Frameworks - Εργαλεία δημιουργίας ιστοσελίδων

3.1 Η αρχιτεκτονική MVC (Model-View-Controller)

Σε μια εφαρμογή ηλεκτρονικών υπολογιστών, όλες οι πληροφορίες τις οποίες χρειάζεται ο υπολογιστής για να παράξει το απαιτούμενο αποτέλεσμα βρίσκονται σε αρχεία, τα οποία έχουν παραχθεί από την μεταγλώτισση αρχείων γραμμένων σε κάποια γλώσσα προγραμματισμού. Οι πληροφορίες αυτές περιλαμβάνουν τις εντολές για την διαχείριση των δεδομένων της εφαρμογής, για την δημιουργία της διεπαφής χρήστη, καθώς και για την εκτέλεση των απαιτούμενων ενεργειών για την εκτέλεση της εκάστοτε εργασίας για την οποία δημιουργήθηκε η εφαρμογή. Όλες αυτές οι πληροφορίες είναι αποθηκευμένες σε αρχεία κώδικα κάποιας γλώσσας προγραμματισμού.

Η αρχιτεκτονική MVC (Model - View - Controller) έχει ώς σκοπό να θέσει σαφή όρια ανάμεσα στα αρχεία που περιέχουν τον απαραίτητο κώδικα για καθεμία από αυτές τις κατηγορίες. Πιο συγκεκριμένα χωρίζει την εφαρμογή μας σε τρία μέρη:

- Model - Σε αυτό το σημείο της εφαρμογής περιγράφεται η δομή των δεδομένων που χρησιμοποιεί η εφαρμογή μας.
- View - Σε αυτό το σημείο της εφαρμογής ορίζεται ο κώδικας που είναι υπεύθυνος για την δημιουργία της διεπαφής χρήστη (User Interface).
- Controller - Σε αυτό το σημείο της εφαρμογής ορίζεται ο κώδικας που είναι υπεύθυνος για όλες τις απαραίτητες ενέργειες για την υλοποίηση της λειτουργικότητας της εφαρμογής.

Η αρχιτεκτονική αυτή μπορεί να χρησιμοποιηθεί σε όλες τις εφαρμογές της ανάπτυξης λογισμικού. Στον τομέα της ανάπτυξης διαδικτυακών εφαρμογών ειδικά, στην σύγχρονη εποχή, έχει κυριαρχήσει και χρησιμοποιείται στο σύνολο σχεδόν των εφαρμογών.

Στην πλατφόρμα Java υπάρχουν διάφορα Frameworks τα οποία υλοποιούν την αρχιτεκτονική MVC για την δημιουργία διαδικτυακών εφαρμογών. Στα πλαίσια αυτής της εργασίας θα ασχοληθούμε με τρία δημοφιλή Web MVC Frameworks, θα παρουσιάσουμε πώς το καθένα από αυτά υλοποιεί τα κύρια σημεία της αρχιτεκτονικής MVC, καθώς και το πώς μας βοηθούν να δημιουργήσουμε διαδικτυακές εφαρμογές. Αυτά θα είναι και τα Frameworks με την χρήση των οποίων θα δημιουργήσουμε την εφαρμογή - παράδειγμα της εργασίας.

3.2 Το Spring Web MVC Framework

Το Spring Framework διαθέτει ένα Project το οποίο χρησιμοποιεί τις δυνατότητές του Framework για την δημιουργία διαδικτυακών εφαρμογών, κάνοντας χρήση της αρχιτεκτονικής MVC. Αυτό το Project ονομάζεται Spring Web MVC.

Το Spring Web MVC χρησιμοποιεί την τεχνολογία των Servlets για την δημιουργία διαδικτυακών εφαρμογών. Πιο συγκεκριμένα, βάση του Framework, είναι μια κλάση Servlet η οποία ονομάζεται DispatcherServlet. Το DispatcherServlet είναι υπεύθυνο για την εκτέλεση όλων των απαραιτήτων ενεργειών, από την στιγμή που δέχεται την αίτηση HTTP, μέχρι την στιγμή που αποστέλλει την απάντηση.

Για την ακρίβεια, το Spring Web MVC διαθέτει κάποια Beans στα οποία το DispatcherServlet αναθέτει τις απαιτούμενες εργασίες. Τα Beans είναι αντικείμενα την διαχείριση των οποίων αναλαμβάνει το Spring Framework (περισσότερα γι΄ αυτό στο αντίστοιχο κεφάλαιο). Οι σημαντικότερες κατηγορίες αυτών των Beans είναι οι παρακάτω:

- HandlerMapping - Αυτή η κατηγορία Bean αναλαμβάνει να αντιστοιχίσει τις αιτήσεις HTTP με τις κατάλληλες μεθόδους του αντίστοιχου Controller.
- ViewResolver - Αυτή η κατηγορία Bean αναλαμβάνει να αντιστοιχίσει τις τιμές που επιστρέφει ο Controller (συνήθως τύπου String), με τα κατάλληλα Views.
- LocaleResolver - Αυτή η κατηγορία Bean αναλαμβάνει να ανακαλύψει πληροφορίες για την τοποθεσία του χρήστη. π.χ. ζώνη ώρας, γλώσσα κ.λπ.
- MultipartResolver - Αυτή η κατηγορία Bean αναλαμβάνει την ανάκτηση αρχείων τα οποία μεταφέρονται στον Server με την χρήση του πρωτοκόλλου HTTP.
- HandlerExceptionResolver - Αυτή η κατηγορία Bean αναλαμβάνει να επιστρέψει την κατάλληλη απάντηση σε περίπτωση σφάλματος.

Για να αρχικοποιηθεί το DispatcherServlet απαιτείται ένα πρόσθετο Configuration στο οποίο θα ορίσουμε και θα κάνουμε Configure συγκεκριμένα αντικείμενα για την καθεμία από αυτές τις κατηγορίες που θέλουμε να χρησιμοποιήσουμε. Αυτό το Configuration μπορεί να γίνει, όπως και σε όλες τις περιπτώσεις Spring Configuration είτε με αρχείο XML, το οποίο θα πρέπει να ονομάζεται servletName-servlet.xml, όπου servletName το όνομα του DispatcherServlet όπως το ορίσαμε στο Configuration του Servlet Container, είτε με κώδικα Java.

Όπως άλλωστε λέει και το όνομά του, το Spring Web MVC είναι ένα MVC Framework. Αυτό σημαίνει πως υπάρχει πλήρης διαχωρισμός ανάμεσα στον κώδικα που περιέχει τα διάφορα μέρη, δηλαδή τα Models, τα Views και τους Controllers.

Στο Spring Web MVC, το μοντέλο μας βρίσκεται σε διάφορες κλάσεις Java. Αυτές οι κλάσεις δεν είναι απαραίτητο να κληρονομούν από κάποια κλάση του Framework, μπορούν δηλαδή να είναι POJO (Plain Old Java Objects). Το Framework μας προσφέρει μόνο κλάσεις με τις οποίες μεταφέρονται τα δεδομένα του Μοντέλου από τους Controllers στα Views.

Το Spring Web MVC δεν απαιτεί την χρήση κάποιας συγκεκριμένης τεχνολογίας για την δημιουργία των Views. Αντιθέτως, η κάθε τεχνολογία προσφέρει την αντίστοιχη υλοποίηση του ViewResolver, η οποία είναι απαραίτητη για την λειτουργία της, και αυτή αναλαμβάνει την δημιουργία των View και την επιστροφή του κατάλληλου κώδικα στον DispatcherServlet για να δημιουργήσει την κατάλληλη απάντηση. Οπότε το View μέρος της εφαρμογής βρίσκεται στα αντίστοιχα αρχεία, είτε αυτά είναι αρχεία JSP, είναι είναι αρχεία HTML, τα οποία περιέχουν εντολές της Thymeleaf ή του Apache Freemarker, είτε είναι αρχεία κάποιας άλλης τεχνολογίας View.

Οι Controllers στο Spring Web MVC, είναι κλάσεις οι οποίες έχουν οριστεί με την χρήση του Annotation @Controller. Αυτό, όπως θα δούμε και σε επόμενο κεφάλαιο, είναι ένα στερεοτυπικό Annotation το οποίο δηλώνει πως η κλάση αυτή είναι ένας Controller, και πως ένα αντικείμενο αυτής της κλάσης πρέπει να δημιουργηθεί και να μπεί ύπο την διαχείριση του Spring Framework κατά την αρχικοποίηση της εφαρμογής.

Η κλάση Controller περιέχει μεθόδους οι οποίες διαχειρίζονται τις διάφορες αιτήσεις HTTP. Η αντιστοίχιση ανάμεσα στις διάφορες αιτήσεις και τις κατάλληλες μεθόδους γίνεται με την χρήση της οικογένειας Annotation @RequestMapping. Το συγκεκριμένο Annotation δέχεται κάποιες παραμέτρους, οι οποίες ορίζουν τις προϋποθέσεις υπό τις οποίες η μέθοδος θα διαχειριστεί την αντίστοιχη αίτηση. Οι πιο σημαντικές από αυτές τις παραμέτρους είναι οι ακόλουθες:

- path - Αυτή η παράμετρος ορίζει ένα μοτίβο για τις ηλεκτρονικές διευθύνσεις των αιτήσεων, για τις οποίες είναι υπεύθυνη η μέθοδος.
- method - Αυτή η παράμετρος ορίζει την μέθοδο HTTP των αιτήσεων, για τις οποίες είναι υπεύθυνη η μέθοδος.
- consumes - Αυτή η παράμετρος ορίζει το Content-type των αιτήσεων, για τις οποίες είναι υπεύθυνη η μέθοδος.
- produces - Αυτή η παράμετρος ορίζει το περιεχόμενο της κεφαλίδας HTTP Accept των αιτήσεων, για τις οποίες είναι υπεύθυνη η μέθοδος.
- headers - Αυτή η παράμετρος ορίζει το περιεχόμενο των κεφαλίδων των αιτήσεων, για τις οποίες είναι υπεύθυνη η μέθοδος.

Οι μέθοδοι του Controller που διαχειρίζονται τις αιτήσεις HTTP επιστρέφουν κάποια αντικείμενα ή μεταβλητές, οι τιμές των οποίων σχετίζονται με την απάντηση HTTP που πρέπει να δοθεί. Είτε δίνουν τις απαραίτητες πληροφορίες στον ViewResolver για την επιλογή του κατάλληλου View, είτε επιστρέφουν κάποιο σφάλμα, οι πληροφορίες για το επόμενο βήμα μετά το τέλος της εκτέλεσης της κατάλληλης μεθόδου βρίσκονται στην τιμή που επιστρέφει αυτή η μέθοδος. Οι σημαντικότεροι τύποι των διαφόρων τιμών που επιστρέφουν οι μέθοδοι των κλάσεων Controller είναι οι παρακάτω:

- String - Όταν η μέθοδος του Controller επιστρέφει ένα String, ο ViewResolver αναλαμβάνει να χρησιμοποιήσει αυτό το String για να επιλέξει το κατάλληλο View.
- ModelAndView - Όταν η μέθοδος του Controller επιστρέφει ένα αντικείμενο ModelAndView ο ViewResolver αναλαμβάνει να επιλέξει το κατάλληλο View με βάση αυτό το αντικείμενο.
- void - Όταν η μέθοδος του Controller επιστρέφει void, αυτό σημαίνει πως η επεξεργασία της αίτησης έχει ολοκληρωθεί.
- @ResponseBody Object - Όταν η μέθοδος ή η κλάση έχει οριστεί με το Annotation @ResponseBody και ο τύπος που επιστρέφει είναι αντικείμενο, τότε καλείται ένα Bean το οποίο είναι υπεύθυνο να γράψει τα περιεχόμενα του αντικειμένου στην απάντηση, με την κατάλληλη μορφή π.χ. JSON σε μια RESTful Web Service.
- HttpHeaders - Όταν η μέθοδος του Controller επιστρέφει ένα αντικείμενο αυτού του τύπου, τότε δημιουργείται μια HTTP απάντηση χωρίς Body, η οποία περιέχει αυτές τις κεφαλίδες HTTP.

Οι μέθοδοι του Controller μπορούν να δέχονται ένα σύνολο από παραμέτρους, οι οποίες δηλώνουν τιμές και πληροφορίες οι οποίες υπάρχουν στις HTTP αιτήσεις. Το Spring Framework αναλαμβάνει να δώσει τις κατάλληλες τιμές σε αυτές τις παραμέτρους κατά την εκτέλεση της μεθόδου. Οι πιο σημαντικές από αυτές τις παραμέτρους είναι οι παρακάτω:

- javax.servlet.ServletRequest / javax.servlet.ServletResponse - Αν ορίσουμε παραμέτρους αυτού του τύπου στη δήλωση μιας μεθόδου, το Spring Web MVC μας κάνει Inject τα αντίστοιχα αντικείμενα της αίτησης και της απάντησης HTTP για την συγκεκριμένη αίτηση.
- javax.servlet.http.HttpSession Αν ορίσουμε παραμέτρο αυτού του τύπου στη δήλωση μιας μεθόδου, το Spring Web MVC μας κάνει Inject το αντίστοιχο αντικείμενο της HTTP Session που σχετίζεται με την συγκεκριμένη αίτηση.
- @RequestParam - Αυτό το Annotation, χρησιμοποιούμενο στον ορισμό μιας παραμέτρου της μεθόδου του Controller ορίζει πως το Spring Web MVC θα

κάνει Inject σε αυτήν την παράμετρο την τιμή της μεταβλητής που περιέχεται στο σώμα της HTTP αίτησης, που έχει το ίδιο όνομα με το όνομα της παραμέτρου. Εναλλακτικά, το Annotation δέχεται μια παράμετρο που ονομάζεται name, η οποία περιέχει το όνομα της μεταβλητής που υπάρχει στο σώμα της αίτησης. Ισχύει για αιτήσεις της μεθόδου POST, οι οποίες περιέχουν μεταβλητές στο σώμα τους.

- @PathVariable - Αυτό το Annotation, χρησιμοποιούμενο στον ορισμό μιας παραμέτρου της μεθόδου του Controller ορίζει πως το Spring Web MVC θα κάνει Inject σε αυτήν την παράμετρο την τιμή της μεταβλητής που περιέχεται στο URL της αίτησης HTTP. Το μοτίβο URL, στο οποίο ορίζεται το όνομα της μεταβλητής κλεισμένο σε Curly Bracers, ορίζεται στο αντίστοιχο Mapping Annotation. Το όνομα της παραμέτρου πρέπει να είναι ίδιο με την τιμή της παραμέτρου του Mapping Annotation.

- @MatrixVariable - Αυτό το Annotation, χρησιμοποιούμενο στον ορισμό μιας παραμέτρου της μεθόδου του Controller ορίζει πως το Spring Web MVC θα κάνει Inject σε αυτήν την παράμετρο την τιμή της μεταβλητής που περιέχεται σε ένα σύνολο από μεταβλητές χωριζόμενες με το Delimeter ;, οι οποίες περιγράφονται στο μοτίβο URL του αντιστοίχου Mapping Annotation. Το όνομα της παραμέτρου πρέπει να είναι ίδιο με την τιμή της παραμέτρου του Mapping Annotation.

- @CookieValue - Αυτό το Annotation, χρησιμοποιούμενο στον ορισμό μιας παραμέτρου της μεθόδου του Controller ορίζει πως το Spring Web MVC θα κάνει Inject σε αυτήν την παράμετρο την τιμή του Cookie που δηλώνεται σαν παράμετρος στο Annotation.

- HttpMethod - Αν ορίσουμε παραμέτρους αυτού του τύπου στη δήλωση μιας μεθόδου, το Spring Web MVC μας κάνει Inject το αντίστοιχο αντικείμενο της μεθόδου HTTP της αίτησης.

- @RequestHeader - Αυτό το Annotation, χρησιμοποιούμενο στον ορισμό μιας παραμέτρου της μεθόδου του Controller ορίζει πως το Spring Web MVC θα κάνει Inject σε αυτήν την παράμετρο την τιμή της κεφαλίδας HTTP που δηλώνεται σαν παράμετρος στο Annotation.

- @ModelAttribute - Αυτό το Annotation, χρησιμοποιούμενο στον ορισμό μιας παραμέτρου της μεθόδου του Controller ορίζει πως το Spring Web MVC θα κάνει Inject σε αυτήν την παράμετρο το αντικείμενο που παράγεται από τις τιμές της αντίστοιχης μεταβλητής που ορίζεται στο σώμα της αίτησης. Το όνομα της παραμέτρου πρέπει είτε να είναι ίδιο με το όνομα της αντίστοιχης μεταβλητής του σώματος της αίτησης, είτε το όνομα της επιθυμητής μεταβλητής να ορίζεται σαν παράμετρος στο Annotation.

- java.util.Locale Αν ορίσουμε παραμέτρο αυτού του τύπου στη δήλωση μιας μεθόδου, το Spring Web MVC μας κάνει Inject το αντίστοιχο αντικείμενο που περιέχει τις πληροφορίες θέσης του χρήστη.

- org.springframework.ui.Model Αν ορίσουμε παραμέτρο αυτού του τύπου στη δήλωση μιας μεθόδου, το Spring Web MVC μας κάνει Inject το αντίστοιχο

αντικείμενο στο οποίο θα τοποθετήσουμε τις πληροφορίες που θα παραδοθούν στο View.

Ένα παράδειγμα για να περιγράψουμε την τυπική λειτουργία του Spring Web MVC κατά την λήψη και επεξεργασία μιας αίτησης HTTP είναι το παρακάτω:

α) Η αίτηση παραδίδεται στο DispatcherServlet.
β) Το αντικείμενο HandlerMapping καλεί τον κατάλληλο Controller για αυτήν την αίτηση
γ) Η μέθοδος εκτελείται και επιστρέφει το κατάλληλο αποτέλεσμα. Ο ViewResolver χρησιμοποιεί το αποτέλεσμα για να εντοπίσει το κατάλληλο View και του παραδίδει τις απαραίτητες πληροφορίες.
δ) Η αντίστοιχη View Technology που χρησιμοποιείται δημιουργεί τον τελικό κώδικα του View και τον παραδίδει στο DispatcherServlet.
ε) Ο DispatcherServlet δημιουργεί την τελική απάντηση HTTP και την αποστέλλει στον Client.

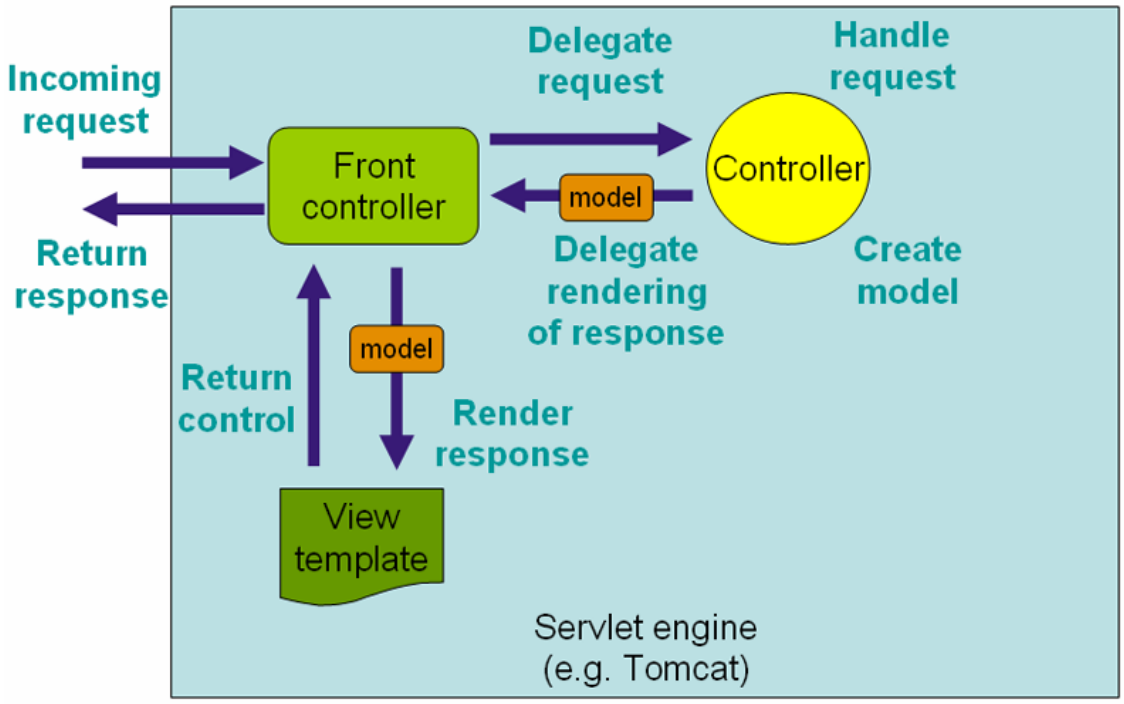

Figure 3.1: Επισκόπηση χρήσης του Spring Web MVC DispatcherServlet (Front Controller)

3.3 Το MVC Framework Java Server Faces

Στην πλατφόρμα Java Enterprise Edition, για την δημιουργία ενός Framework που θα έχει ως αντικείμενο την ανάπτυξη διαδικτυακών εφαρμογών, δημιουργήθηκε ένα πρότυπο. Το πρότυπο αυτό ονομάζεται Java Server Faces και την υλοποίησή του ανέλαβε η Oracle.

Το Java Server Faces, όπως και το Spring Web MVC, χρησιμοποιεί την τεχνολογία των Servlets για την δημιουργία διαδικτυακών εφαρμογών. Πιο συγκεκριμένα, το Framework διαθέτει μια κλάση Servlet η οποία είναι υπεύθυνη για την διαχείριση της αίτησης και την δημιουργία της απάντησης HTTP. Αυτή η κλάση ονομάζεται FacesServlet.

Το Configuration ενός Project του Java Server Faces μπορεί να γίνει με δύο τρόπους. Είτε με μορφή XML, σε ένα αρχείο που ονομάζεται faces-config.xml, είτε με την χρήση κώδικα Java. Σε αυτό το Configuration περιέχονται πληροφορίες για τον ορισμό διαφόρων αντικειμένων, τα οποία εκτελούν διάφορες λειτουργίες σχετικές με την ανάπτυξη της λειτουργικότητας. Οι πιο σημαντικές κατηγορίες τέτοιων αντικειμένων είναι οι παρακάτω:

- ManagedBeans - Αυτά τα αντικείμενα περιέχουν τα δεδομένα και τις μεθόδους που είναι προσβάσιμες από τα Views.
- Validators - Αυτά τα αντικείμενα εκτελούν ελέγχους εγκυρότητας στα δεδομένα.
- Converters - Αυτά τα αντικείμενα εκτελούν διάφορες τροποποιήσεις στα δεδομένα. π.χ. αλλαγή ενός πεδίου ανάλογα με την τοποθεσία του Client. κ.λπ.
- Listeners - Αυτά τα αντικείμενα ελέγχουν την εκτέλεση της εφαρμογής για γεγονότα και καλούν τον κατάλληλο κώδικα κατά την πυροδότηση τους.

Το Framework Java Server Faces ακολουθεί την αρχιτεκτονική MVC. Υπάρχει δηλαδή διαχωρισμός ανάμεσα στον κώδικα που περιέχει τα Models, τα Views και τον Controller.

Στο Java Server Faces, το μοντέλο μας βρίσκεται σε αντικείμενα τα οποία ονομάζονται ManagedBeans. Πρόκειται για αντικείμενα τα οποία τα διαχειρίζεται το Framework, και τα οποία είναι προσβάσιμα από τα Views. Τα αντικείμενα αυτά μπορεί να περιέχουν διάφορα πεδία, αλλά και μεθόδους, τις οποίες μπορούμε να καλέσουμε από τα Views. Για να δημιουργήσουμε ένα ManagedBean απλά δημιουργούμε μια κλάση, την οποία ορίζουμε με το Annotation @ManagedBean στο οποίο ορίζουμε την παράμετρο name. Η τιμή αυτής της παραμέτρου είναι το όνομα που θα χρησιμοποιήσουμε στα View για να αποκτήσουμε πρόσβαση σε αυτό το αντικείμενο. Επίσης σε αυτήν την κλάση θα πρέπει να τοποθετήσουμε ένα Annotation το οποίο θα ορίζει τον κύκλο ζωής αυτού του αντικειμένου. Τα πιο σημαντικά πιθανά τέτοια Annotations είναι τα παρακάτω:

- @RequestScoped - Το αντικείμενο αυτό διατηρείται για όσο διαρκεί η επεξεργασία της αίτησης. Για επόμενη αίτηση δημιουργείται ένα νέο Instance.
- @ViewScoped - Το αντικείμενο αυτό διατηρείται όσο η εφαρμογή βρίσκεται στο View το οποίο το κάλεσε. Αν καλεστεί κάποιο άλλο View, δημιουργείται ένα νέο Instance.
- @SessionScoped - Το αντικείμενο αυτό διατηρείται όσο διαρκεί η HTTP Session η οποία το κάλεσε.
- @ApplicationScoped - Το αντικείμενο αυτό διατηρείται για όσο εκτελείται η εφαρμογή μας.

Στο Java Server Faces δεν χρειάζεται να δημιουργήσουμε τον δικό μας Controller. Τον ρόλο αυτό αναλαμβάνει το FacesServlet. Αυτό σημαίνει πως η επιλογή του κατάλληλου View για κάθε αίτηση γίνεται από αυτό και πως αυτό είναι υπεύθυνο να τοποθετεί τα δεδομένα που παρέχει ο χρήστης στα ManagedBeans. Αυτό σημαίνει πως στην ανάπτυξη εφαρμογών με την χρήση του Java Server Faces ο ρόλος του προγραμματιστή είναι να δημιουργεί Models και Views.

Στο Java Server Faces τα Views δημιουργούνται χρησιμοποιώντας μια τεχνολογία η οποία ονομάζεται Facelets. Πρόκειται για μία βιβλιοθήκη, η οποία εκμεταλλεύεται τις δυνατότητες της επεκτάσιμης γλώσσας υπερκειμένου (eXtensible HyperText Markup Language - XHTML) για την ενσωμάτωση βιβλιοθηκών της τεχνολογίας JSP και την προσθήκη νέων, με σκοπό την δημιουργία επαναχρησιμοποιήσιμων Components. Οι taglibs που διαθέτει αυτή η τεχνολογία είναι οι παρακάτω:

- h - Αυτή η βιβλιοθήκη περιέχει Tags τα οποία χρησιμοποιούνται για την παραγωγή HTML Components.
- f - Αυτή η βιβλιοθήκη περιέχει Tags τα οποία χρησιμοποιούνται για να συνδέσουν αντικείμενα όπως Validators, Listeners και Converters στα HTML Components.
- ui - Αυτή η βιβλιοθήκη περιέχει Tags τα οποία χρησιμοποιούνται για την προσθήκη δομών όπως δομή επιλογής, δομή επανάληψης και προσθήκη κώδικα από εξωτερικά αρχεία.
- JSTL Core / JSTL Functions - Οι βιβλιοθήκες JSTL (Java Standard Tag Library) της JSP

Τα διάφορα Tags αυτών των βιβλιοθηκών διαθέτουν Attributes στα οποία ενσωματώνουμε τα δεδομένα που υπάρχουν στα Managed Bean. Με αυτόν τον τρόπο γίνεται η μεταφορά των δεδομένων από τα Managed Beans στα Views και αντίστροφα. Για αυτόν τον σκοπό χρησιμοποιείται μια δομή η οποία ονομάζεται Expression Language του Java Server Faces Framework. Οτιδήποτε υπάρχει ανάμεσα στα Curly Bracers στην δομή #{....} είναι μια έκφραση. Χρησιμοποιώντας αυτές τις εκφράσεις αποκτούμε πρόσβαση στα Managed Beans μας, στα πεδία και τις μεθόδους τους. π.χ.

```
1 <h:inputText id="name"
2                value="#{myBean.myProperty}">
3 </h:inputText>
```

Τα Managed Beans μας εκτός από μεταβλητές, μπορούν να περιέχουν και μεθόδους. Αυτές οι μέθοδοι ονομάζονται Action Methods, και μπορούμε να τις καλέσουμε από τα View κάνοντας χρήση των ειδικών Tags h:commandButton ή h:commandLink και ορίζοντας την μέθοδο που θα καλεστεί όταν πατηθεί το αντίστοιχο κουμπί ή Link στο Attribute action. π.χ.

```
1 <h:commandButton id="name"
2                  action="#{myBean.myMethod()}">
3 </h:commandButton>
```

Μια Action Method θα πρέπει να επιστρέφει μια τιμή τύπου String. Αυτή η τιμή ορίζει σε ποιό View θα επιστρέψει η εφαρμογή μετά την εκτέλεση της μεθόδου.

Ένα παράδειγμα για να περιγράψουμε την τυπική λειτουργία του Java Server Faces Framework κατά την λήψη και επεξεργασία μιας αίτησης HTTP είναι το παρακάτω:

α) Η αίτηση παραδίδεται στο FacesServlet.
β) Το κατάλληλο View δημιουργείται και επιστρέφεται.
γ) Όταν υπάρξει κάποιο Event το οποίο είτε αλλάζει τα δεδομένα του Managed Bean, καλεστεί δηλαδή κάποια μέθοδος Setter, είτε καλέσει κάποιο Action Method, η αντίστοιχη λειτουργικότητα εκτελείται.
δ) Η μέθοδος που εκτελείται επιστρέφει το ποιό View πρέπει να επιστραφεί και αυτό δημιουργείται και αποστέλλεται στον Client.

3.4 Το MVC Framework Struts 2

Το τρίτο Web MVC Framework που θα εξετάσουμε είναι ένα Project του Apache Foundation που ονομάζεται Struts 2.

Το Struts 2 χρησιμοποιεί την τεχνολογία των φίλτρων των Servlet Containers για να αντιστοιχίσει τις διάφορες αιτήσεις HTTP που δέχεται η εφαρμογή με τον κατάλληλο κώδικα της εφαρμογής που τις διαχειρίζεται. Τον ρόλο αυτό παίζει ένα αντικείμενο της κλάσης StrutsPrepareAndExecuteFilter.

Η αντιστοίχιση αυτή γίνεται χρησιμοποιώντας ένα αρχείο Configuration στο οποίο περιέχονται οι απαραίτητες πληροφορίες. Το αρχείο αυτό ονομάζεται struts.xml και σε αυτό ορίζουμε διαφορετικές κατηγορίες πληροφοριών. Οι σημαντικότερες από αυτές είναι:

- Beans - Εδώ ορίζουμε πληροφορίες για τα αντικείμενα τα οποία διαχειρίζεται το Framework.
- Interceptors - Εδώ ορίζουμε πληροφορίες για μεθόδους που καλούνται πριν την εκτέλεση των Action μεθόδων.
- Actions - Εδώ ορίζουμε πληροφορίες για την αντιστοίχιση των αιτήσεων με τις κατάλληλες Action μεθόδους.
- Results - Εδώ ορίζουμε πληροφορίες για την αντιστοίχιση των αποτελεσμάτων των Action μεθόδων με τα κατάλληλα Views.

Κατά το πρότυπο MVC, το Μοντέλο στο Struts 2 Framework βρίσκεται σε κλάσεις Java. (POJO). Ο τρόπος με τον οποίο τα View αποκτούν πρόσβαση στα δεδομένα του Μοντέλου είναι μέσω πεδίων στις Action κλάσεις.

Στο Struts 2, τον ρόλο των Controllers παίζουν οι Action κλάσεις. Οι κλάσεις αυτές περιέχουν μια μέθοδο που ονομάζεται execute. Η μέθοδος αυτή εκτελείται όταν η εφαρμογή λάβει την αντίστοιχη αίτηση HTTP, σύμφωνα με τις πληροφορίες που περιέχονται στο αρχείο struts.xml. Αυτή η μέθοδος, επιστρέφει ένα String. Οι πληροφορίες που περιέχονται στο αρχείο struts.xml χρησιμοποιούνται για να αντιστοιχίσουν αυτό το String με το κατάλληλο View, το οποίο θα καλεστεί.

Τα Views στο Struts 2 μπορούν να δημιουργηθούν σε οποιαδήποτε View τεχνολογία. Το αρχείο struts.xml περιέχει τις πληροφορίες που χρειάζεται το Framework ώστε να καλέσει το κατάλληλο View έχοντας ώς δεδομένο το Result String, το οποίο επιστρέφει η μέθοδος execute. Default View τεχνολογία όμως είναι οι JSPs. Το Struts 2 διαθέτει ειδικά Tag Libraries τα οποία είναι σχεδιασμένα για την ανάπτυξη των JSPs των εφαρμογών που είναι γραμμένες σε αυτό.

Ένα παράδειγμα για να περιγράψουμε την τυπική λειτουργία του Framework Struts 2 κατά την λήψη και επεξεργασία μιας αίτησης HTTP είναι το παρακάτω:

α) Η αίτηση παραδίδεται στο Filter.
β) Εκτελούνται τα κατάλληλα Interceptors.
γ) Η κατάλληλη Action μέθοδος καλείται.
δ) Η μέθοδος επιστρέφει το String, το οποίο χρησιμοποιείται για να καλεστεί το κατάλληλο View.
ε) Το View δημιουργείται και επιστρέφεται.

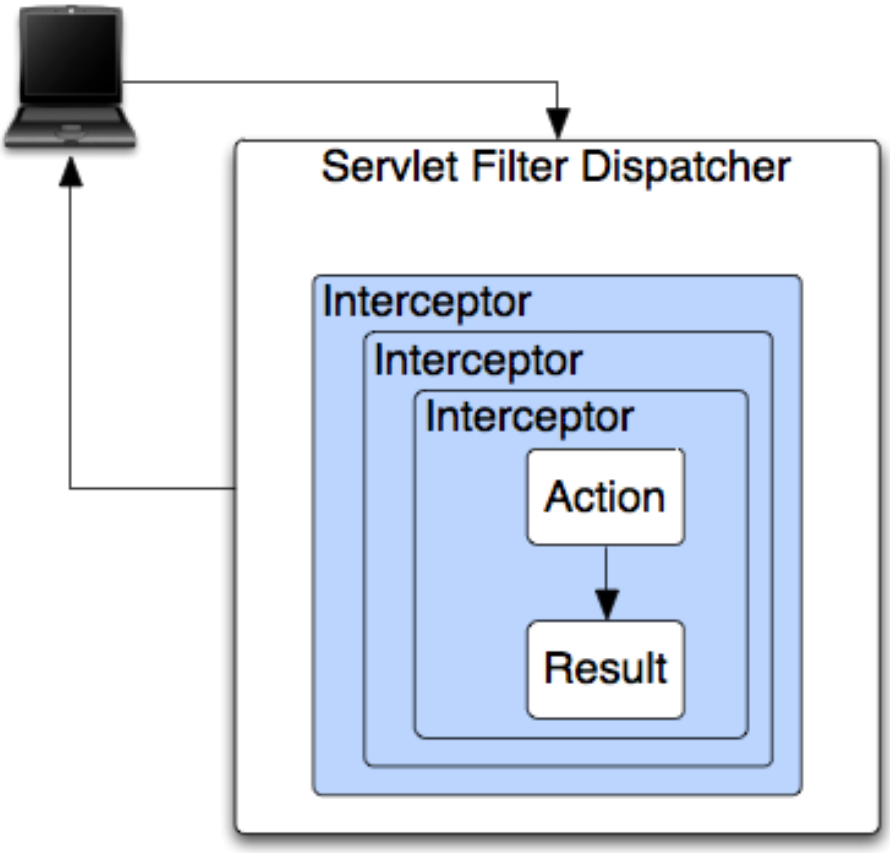

Figure 3.2: Τυπική λειτουργία του Struts 2 Framework

3.5 Συμπεράσματα από την χρήση των διαφόρων frameworks

Στο κεφάλαιο αυτό είδαμε συνοπτικά πώς τρία διαφορετικά Frameworks υλοποιούν την αρχιτεκτονική MVC με σκοπό την δημιουργία διαδικτυακών εφαρμογών. Καθώς αυτά τα Frameworks πραγματεύονται το ίδιο θέμα, έχουν αρκετές ομοιότητες. Πολλά πράγματα όμως τα κάνουν διαφορετικά. Σε αυτό το σημείο θα εξετάσουμε επιγραμματικά τα πλεονεκτήματα αλλά και τις αδυναμίες καθενός από αυτά.

Το Spring Web MVC Framework είναι ένα αρκετά ισχυρό εργαλείο. Οι δυνατότητες παραμετροποίησης και επιλογής του τρόπου και των εργαλείων ανάπτυξης σε κάθε επίπεδο της εφαρμογής μας είναι πολύ περισσότερες από ότι στα άλλα δύο εργαλεία που εξετάσαμε. Επίσης η λειτουργία του στηρίζεται στο βασικό κομμάτι του Spring Framework, το οποίο μας προσφέρει αρκετές δυνατότητες (Dependency Injection, κεντρική διαχείριση αντικειμένων κ.λπ.). Υπάρχει βέβαια η δυνατότητα να χρησιμοποιήσουμε το Spring Framework στις εφαρμογές που δημιουργούμε με την βοήθεια των άλλων MVC Frameworks, αλλά αυτό απαιτεί πρόσθετο Configuration.

Από την άλλη το Spring Web MVC παρουσιάζει τα μειονεκτήματα του Spring Framework, τα οποία δεν είναι άλλα από την πολυπλοκότητα, το γεγονός πως δεν είναι φιλικό προς τον αρχάριο χρήστη, καθώς και το γεγονός πως το αρχικό Configuration είναι ιδιαίτερα απαιτητικό ακόμα και για τις εφαρμογές με τις μικρότερες δυνατές ανάγκες.

Το Framework Java Server Faces αποτελεί ένα πρότυπο της πλατφόρμας Java Enterprise Edition. Αυτό σημαίνει πως υποστηρίζεται από ένα μεγάλο κομμάτι της κοινότητας, οπότε υπάρχει μια μεγάλη πηγή πληροφόρησης. Επίσης σημαίνει πως πολλά Projects το λαμβάνουν υπ΄ όψιν και φροντίζουν ο κώδικας τους να γίνεται εύκολα Integrate με αυτό. Το Framework είναι αρκετά εύκολο στη χρήση, προσφέρει ένα αρκετά εκτενές API για την δημιουργία οπτικού περιεχομένου, και η ανάπτυξη της λειτουργικότητας είναι αρκετά απλή.

Από την άλλη όμως, η χρήση του επιβάλλει συγκεκριμένο τρόπο ανάπτυξης της λειτουργικότητας, καθώς και συγκεκριμένα εργαλεία για την δημιουργία του User Interface.

Το Framework Struts 2 είναι ένα εύκολο στη χρήση, φιλικό προς τον αρχάριο χρήστη Web MVC Framework, το οποίο απαιτεί ένα αρκετά σύντομο Configuration. Διαθέτει ένα Straightforward τρόπο δημιουργίας οπτικού περιεχομένου.

Στην αντίπερα όχθη, παρ΄ όλο που το Struts 2 είναι σχετικά δημοφιλές (αν και όχι όσο τα άλλα δύο), η βάση των πληροφοριών που μπορούμε να βρούμε γι΄ αυτό είναι συγκριτικά περιορισμένη.

Κεφάλαιο 4

Web Services - Διαδικτυακές Υπηρεσίες

4.1 Εισαγωγή

Διαδικτυακές υπηρεσίες (Web Services) ονομάζονται οι εφαρμογές, οι λειτουργίες των οποίων μπορούν να κληθούν από κώδικα άλλων εφαρμογών με την χρήση του διαδικτύου (Worldwide Web).

Οι εφαρμογές που τις καλούν δεν χρειάζεται να είναι γραμμένες στην ίδια γλώσσα. Χρειάζονται μόνο μηχανισμούς για να επικοινωνούν μέσω δικτύου, για να χρησιμοποιούν το πρωτόκολλο HTTP και χρειάζονται επίσης μηχανισμούς για την διαχείριση ενός ή περισσοτέρων μορφών απεικόνισης των μηνυμάτων που περιέχονται στις αιτήσεις προς τον εξυπηρετητή της υπηρεσίας και τις αντίστοιχες απαντήσεις. Αυτές οι μορφές απεικόνισης είναι συνήθως XML (eXtensible Markup Language) ή JSON (JavaScript Object Notation).

Υπάρχουν δύο είδη διαδικτυακών υπηρεσιών: Οι SOAP Web Services και οι RESTful Web Services.

Οι SOAP Web Services είναι το πρώτο χρονικά πρότυπο για τον σχεδιασμό και την υλοποίηση διαδικτυακών υπηρεσιών. Βασίζονται στο πρωτόκολλο SOAP (Simple Object Access Protocol) το οποίο χρησιμοποιεί οποιοδήποτε πρωτόκολλο επικοινωνίας του διαδικτύου (HTTP, SMTP) κ.λπ. για να μεταφέρει τα μηνύματα είτε της αίτησης είτε της απάντησης, τα οποία είναι γραμμένα αποκλειστικά στην μορφή XML.

Σε κάθε μήνυμα (αίτησης ή απάντησης) του πρωτοκόλλου SOAP το κύριο στοιχείο που περιέχει όλο το μήνυμα είναι ο φάκελος (SOAP Envelope).
Έπειτα ένα μήνυμα μπορεί να περιέχει προαιρετικά μια κεφαλίδα (SOAP Header), και πρέπει να περιέχει υποχρεωτικά το κυρίως μήνυμα (SOAP Body). Στο κυρίως μήνυμα αναφέρεται ποιά λειτουργία της υπηρεσίας θέλουμε να χρησιμοποιήσουμε καθώς και δίδονται οι παράμετροι της κλήσης.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DIS</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 4.1: Παράδειγμα αίτησης HTTP που περιέχει μήνυμα του πρωτοκόλλου SOAP

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse xmlns:m="Some-URI">
           <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 4.2: Παράδειγμα απάντησης HTTP που περιέχει μήνυμα του πρωτοκόλλου SOAP

Κάθε SOAP Web Service που εκδίδεται στο διαδίκτυο, παραθέτει ένα αρχείο που ονομάζεται WSDL (Web Service Description Language) και το οποίο είναι ένα αρχείο XML όπου εξηγούνται τα διάφορα χαρακτηριστικά για την συγκεκριμένη υπηρεσία. Πιο συγκεκριμένα:

α) Επιδεικνύει τις διάφορες λειτουργίες (συναρτήσεις) τις οποίες διαθέτει η υπηρεσία, καθώς και τις παραμέτρους για την κλήση καθεμίας από αυτές
β) Εξηγεί το πώς πρέπει να καλεστεί η υπηρεσία. Σε ποια διεύθυνση (URL) θα σταλεί η αίτηση (αυτή η διεύθυνση ονομάζεται Endpoint)
γ) Καταδεικνύει την δομή του κειμένου XML και της αίτησης και της απάντησης
δ) Καθορίζει τους τύπους δεδομένων των παραμέτρων και των απαντήσεων

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description
    targetNamespace="http://example.org/TicketAgent.wsdl20"
    xmlns:xsTicketAgent="http://example.org/TicketAgent.xsd"
    xmlns:wsdl="http://www.w3.org/ns/wsdl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/ns/wsdl http://www.w3.org/2007/06/wsdl/wsdl20.xsd">

    <wsdl:types>
        <xs:import schemaLocation="TicketAgent.xsd"
                   namespace="http://example.org/TicketAgent.xsd" />
    </wsdl:types>

    <wsdl:interface name="TicketAgent">
        <wsdl:operation name="listFlights"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:listFlightsRequest"/>
            <wsdl:output element="xsTicketAgent:listFlightsResponse"/>
        </wsdl:operation>

        <wsdl:operation name="reserveFlight"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:reserveFlightRequest"/>
            <wsdl:output element="xsTicketAgent:reserveFlightResponse"/>
        </wsdl:operation>
    </wsdl:interface>
</wsdl:description>
```

Figure 4.3: Παράδειγμα αρχείου περιγραφής Web Service WSDL

Κάθε διαδικτυακή υπηρεσία που εκδίδεται στο δίκτυο μπορεί να εγγραφεί σε ένα ηλεκτρονικό μητρώο που ονομάζεται UDDI (Universal Description, Discovery and Integration). Σε αυτό το μητρώο οι προγραμματιστές μπορούν να κάνουν αναζήτηση στις υπάρχουσες υπηρεσίες για να βρούν αυτή που ταιριάζει στις απαιτήσεις τις εφαρμογής τους. Σε αυτό το μητρώο περιέχεται και το αρχείο WSDL, από το οποίο οι προγραμματιστές μπορούν να αντλήσουν όλες τις πληροφορίες για την κλήση της υπηρεσίας.

Οι RESTful Web Services (REpresentational State Transfer) δεν βασίζονται σε κάποιο πρωτόκολλο για να καθορίσουν τα μηνύματα που ανταλλάσονται κατά την επικοινωνία, αλλά εκμεταλλεύονται τις δυνατότητες του πρωτοκόλλου επικοινωνίας HTTP, εκτελώντας διαφορετικές λειτουργίες ανάλογα με την αίτηση HTTP που δέχονται. Για την ακρίβεια η αρχιτεκτονική REST δεν καθορίζει την μορφή των μηνυμάτων. Απλά βασίζεται στο γεγονός πως οι δύο εμπλεκόμενες εφαρμογές πρέπει να καταλαβαίνουν η μία την άλλη. Για την επικοινωνία με υπηρεσίες RESTful, κατα κύριο λόγο χρησιμοποιείται η μορφή JSON, αλλά μπορεί κάλλιστα να χρησιμοποιηθεί και η μορφή XML.

Βασικό χαρακτηριστικό της αρχιτεκτονικής είναι πως χρησιμοποεί τις διαφορετικές μεθόδους του πρωτοκόλλου HTTP για να εκτελέσει διαφορετικές λειτουργίες. Έτσι μια αίτηση με την μέθοδο GET χρησιμοποιείται για να ανακτήσει δεδομένα, μια αίτηση με την μέθοδο POST χρησιμοποιείται για να αποθηκεύσει καινούρια δεδομένα, μια αίτηση τύπου PUT χρησιμοποιείται για να αλλάξει την κατάσταση υπαρχόντων δεδομένων και μια αίτηση τύπου DELETE χρησιμοποιείται για να διαγράψει δεδομένα. Επίσης γίνεται χρήση των κωδικών απάντησης του πρωτοκόλλου HTTP με σκοπό την ενημέρωση της εφαρμογής - πελάτη για την επιτυχία ή την αποτυχία της εκτέλεσης της λειτουργίας, καθώς και για ποιό λόγο υπήρξε αποτυχία.

Ένα άλλο χαρακτηριστικό αυτών των υπηρεσιών είναι πως κάθε αίτηση είναι ανεξάρτητη, ο εξυπηρετητής δεν κρατά καμία πληροφορία για την εφαρμογή πελάτη που του έκανε την αίτηση.

Βασική διαφορά με τις SOAP Web Services είναι το γεγονός πως οι πρώτες δέχονταν όλες τις αιτήσεις σε μια ηλεκτρονική διεύθυνση και η πληροφορία του ποιά λειτουργία θα κληθεί / ποια δεδομένα θα επεξεργαστούν εμπεριέχονταν στο μήνυμα SOAP. Στις RESTful Web Services όμως, η πληροφορία αυτή είναι ένας συνδυασμός του με ποιά μέθοδο HTTP έγινε η αίτηση, καθώς και σε ποιά ηλεκτρονική διεύθυνση έγινε. Το μήνυμα αυτό καθ΄ αυτό περιέχει παραμέτρους για την κλήση της λειτουργίας.

Καθώς η RESTful είναι μια αρχιτεκτονική που χρησιμοποιεί απλά τα βασικά πρωτόκολλα επικοινωνίας του διαδικτύου, δίνοντας κατευθύνσεις για το πώς θα πρέπει να δημιουργηθεί μια ηλεκτρονική υπηρεσία, χωρίς να περιορίζει τον προγραμματιστή με την χρήση κάποιου αυστηρού πρωτοκόλλου, όπως γίνεται στην περίπτωση των υπηρεσιών SOAP, παρατηρείται το φαινόμενο αρκετές υπηρεσίες να ακολουθούν λιγότερο ή περισσότερο αυτές τις κατευθύνσεις. Γι΄ αυτό το λόγο υπάρχει ένα μοντέλο, το Richardson Maturity Model, το οποίο κατηγοριοποιεί τις υπηρεσίες με βάση το κατα πόσο κάνουν χρήση των κατευθύνσεων της αρχιτεκτονικής. Αυτή η κατηγοριοποίηση γίνεται ώς εξής:

α) Επίπεδο μηδέν: Σε αυτό το επίπεδο η εφαρμογή δεν κάνει χρήση καμίας εκ των κατευθύνσεων τις αρχιτεκτονικής, δηλαδή δεν χρησιμοποιεί διαφορετικά URLs για να δείξει ποιά λειτουργία θα εκτελεστεί, δεν χρησιμοποιεί διαφορετικές μεθόδους του πρωτοκόλλου HTTP για τις διαφορετικές κατηγορίες επίδρασης στα δεδομένα και δεν κάνει χρήση του HATEOAS, μιας κατεύθυνσης της αρχιτεκτονικής που θα εξηγήσουμε παρακάτω. Αυτού του τύπου οι εφαρμογές δεν μπορούν να χαρακτηριστούν RESTful.

α) Επίπεδο ένα: Σε αυτό το επίπεδο η εφαρμογή χρησιμοποιεί διαφορετικά URLs για να δείξει ποιά λειτουργία θα εκτελεστεί, αλλά δεν χρησιμοποιεί διαφορετικές μεθόδους του πρωτοκόλλου HTTP για τις διαφορετικές κατηγορίες επίδρασης στα δεδομένα και δεν κάνει χρήση του HATEOAS.

α) Επίπεδο δύο: Σε αυτό το επίπεδο η εφαρμογή χρησιμοποιεί διαφορετικά URLs για να δείξει ποιά λειτουργία θα εκτελεστεί και διαφορετικές μεθόδους του πρωτοκόλλου HTTP για τις διαφορετικές κατηγορίες επίδρασης στα δεδομένα, αλλά δεν κάνει χρήση του HATEOAS.

α) Επίπεδο τρία: Σε αυτό το επίπεδο η εφαρμογή πλέον χαρακτηρίζεται full REST, ακολουθεί δηλαδή όλες τις κατεθύνσεις που ορίζει η αρχιτεκτονική. Βασική διαφορά με μια υπηρεσία που χαρακτηρίζεται σαν επιπέδου δύο είναι πως υλοποιεί το HATEOAS (Hypermedia As The Engine Of Application State). Αυτή η κατεύθυνση ορίζει πως εκτός από το αποτέλεσμα της ζητούμενης διαδικασίας, σαν απάντηση η υπηρεσία θα πρέπει να επιστρέφει στην εφαρμογή - πελάτη μια λίστα από ηλεκτρονικές διευθύνσεις στις οποίες βρίσκονται λειτουργίες τις οποίες είναι πιθανόν να θέλει η εφαρμογή - πελάτης να καλέσει στο μέλλον.

4.2 SOAP Web Services με την χρήση της Java

Η πλατφόρμα Java Enterprise Edition διαθέτει τα εργαλεία που χρειαζόμαστε για να δημιουργήσουμε SOAP Web Services, αλλά και εργαλεία που χρειαζόμαστε για να δημιουργήσουμε εφαρμογές που θα καλούν SOAP Web Services. Πιο συγκεκριμένα υπάρχει το JSR 224: JAX-WS (Java API for XML Web Services).

Το JAX-WS API είναι ένα σύνολο από κλάσεις τις οποίες μπορούμε να χρησιμοποιήσουμε για να μετατρέψουμε μια οποιαδήποτε κλάση της γλώσσας Java σε μια διαδικτυακή υπηρεσία που χρησιμοποιεί το πρωτόκολλο SOAP. Για να δημιουργήσουμε δηλαδή μια διαδικτυακή υπηρεσία, δημιουργούμε μια κλάση Java στην οποία υλοποιούμε όλες τις λειτουργίες οι οποίες θέλουμε να είναι διαθέσιμες για κλήση στην διαδικτυακή μας υπηρεσία. Αυτή η κλάση ονομάζεται Service Endpoint Implementation (SEI).

Μια τέτοια κλάση πρέπει να πληροί κάποιες προϋποθέσεις:

α) Πρέπει να δηλωθεί χρησιμοποιώντας το annotation javax.jws.WebService.
β) Οι μέθοδοι που θα είναι διαθέσιμες προς κλήση από τις εφαρμογές - πελάτες της υπηρεσίας πρέπει να μην είναι δηλωμένες ώς static, final ή abstract.
γ) Οι μέθοδοι που θα είναι διαθέσιμες προς κλήση από τις εφαρμογές - πελάτες της υπηρεσίας πρέπει να δηλωθούν χρησιμοποιώντας το annotation javax.jws.WebMethod.
δ) Θα πρέπει να περιέχει έναν default κατασκευαστή.
ε) Δεν θα πρέπει να περιέχει υλοποίηση της μεθόδου finalize.
στ) Οι τύποι δεδομένων των παραμέτρων αλλά και ο τύπος του αποτελέσματος της εκτέλεσης κάθε μεθόδου που θα είναι διαθέσιμη προς κλήση από τις εφαρμογές - πελάτες της υπηρεσίας πρέπει να είναι συμβατοί με τους τύπους του JAXB (βλέπε πίνακα).

Οι διαδικτυακές υπηρεσίες όπως είπαμε δεν είναι σίγουρο ότι θα είναι υλοποιημένες στην ίδια γλώσσα με τις εφαρμογές που τις καλούν. Αυτό σημαίνει πως οι τύποι δεδομένων της μίας τεχνολογίας θα διαφέρουν, τουλάχιστον ως προς την υλοποίηση από τους τύπους της άλλης. Για να λυθεί αυτό το θέμα στο WSDL περιγράφονται όλοι οι τύποι δεδομένων που χρησιμοποιούνται στις μεθόδους τις υπηρεσίας με όρους XML. Η κάθε γλώσσα στην οποία γράφονται είτε οι υπηρεσίες, είτε οι εφαρμογές - πελάτες διαθέτει μηχανισμούς για να μετατρέπει αυτούς τους όρους στους αντίστοιχους τύπους δεδομένων και αντίστροφα. Στην περίπτωση της Java αυτός ο μηχανισμός ονομάζεται JAXB (Java Architecture for XML Binding).

Όρος στο έγγραφο XML	Τύπος δεδομένων στην γλώσσα Java
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd.long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	javax.xml.datatype.XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:time	javax.xml.datatype.XMLGregorianCalendar
xsd:date	javax.xml.datatype.XMLGregorianCalendar
xsd:g	javax.xml.datatype.XMLGregorianCalendar
xsd:anySimpleType	java.lang.Object
xsd:anySimpleType	java.lang.String
xsd:duration	javax.xml.datatype.Duration
xsd:NOTATION	javax.xml.namespace.QName

Table 4.1: Αντιστοιχίες όρων XML με τύπους δεδομένων στην Java

Είπαμε πως ο τρόπος με τον οποίο ο προγραμματιστής μιας εφαρμογής που καλεί την διαδικτυακή μας υπηρεσία μαθαίνει τις λεπτομέρειες που χρειάζεται για να μπορεί να το κάνει είναι διαβάζοντας το αρχείο WSDL. Αυτό το αρχείο παράγεται αυτόματα κατά την εγκατάσταση και εκτέλεση της υπηρεσίας μας χάρη στη χρήση των annotations του JAX-WS API που χρησιμοποιήσαμε στην υλοποίηση της κλάσης SEI. Οι πληροφορίες που υπάρχουν στην κλάση ή συμπεριλαμβάνονται σαν παράμετροι στα εν λόγω annotations χρησιμοποιούνται για την παραγωγή του WSDL. Πιο συγκεκριμένα:

Το annotation javax.ws.WebService δέχεται τις εξής παραμέτρους:

α) name: Το όνομα της υπηρεσίας, όπως αναφέρεται στο στοιχείο <wsdl:portType > του αρχείου WSDL. Σε περίπτωση που δεν ορίζεται τιμή της παραμέτρου, ώς default τιμή ορίζεται το όνομα της κλάσης.
β) targetNamespace: Το namespace που χρησιμοποιείται για να οριστούν τα στοιχεία XML που παράγονται από την υπηρεσία.
γ) serviceName: Το όνομα της υπηρεσίας, όπως αναφέρεται στο στοιχείο <wsdl:service > του αρχείου WSDL. Σε περίπτωση που δεν ορίζεται τιμή της παραμέτρου, ώς default

τιμή ορίζεται το όνομα της κλάσης ακολουθούμενο από τη λέξη Service.
δ) wsdlLocation: Η ηλεκτρονική διεύθυνση στην οποία βρίσκεται το αρχείο WSDL, σε περίπτωση που υπάρχει έτοιμο και δεν το δημιουργεί αυτόματα η εφαρμογή. Επιστρέφει λάθος άν το αρχείο δεν έχει τις σωστές τιμές που αναφέρονται ώς τιμές στις παραμέτρους του annotation και στα διάφορα ονόματα κλάσης και μεθόδων. Στις περισσότερες περιπτώσεις δεν τοποθετούμε κάποια τιμή.
ε) endpointInterface: Το όνομα του αρχείου στο οποίο βρίσκεται το endpoint Interface. Θεωρεί ότι έχουμε δημιουργήσει και τοποθετήσει το εν λόγω αρχείο στο classpath. Συνήθως αυτό το αρχείο το δημιουργεί αυτόματα η εφαρμογή, οπότε δεν τοποθετούμε κάποια τιμή.

Το annotation javax.ws.WebMethod δέχεται τις εξής παραμέτρους:

α) operationName: Το όνομα της συγκεκριμένης μεθόδου, όπως αναφέρεται στο στοιχείο <wsdl:operation > του αρχείου WSDL. Σε περίπτωση που δεν ορίζεται τιμή της παραμέτρου, ώς default τιμή ορίζεται το όνομα της μεθόδου.
β) action: Το όνομα της διεύθυνσης που στέλνεται το μήνυμα για την κλήση της μεθόδου. όπως ορίζεται στην τιμή της παραμέτρου SOAPAction του SOAPHeader.

Για να δημιουργήσει τις πληροφορίες για τις παραμέτρους μιας μεθόδου στο αρχείο WSDL, χρησιμοποιείται το annotation javax.ws.WebParam, το οποίο δέχεται τις εξής παραμέτρους:

α) name: Το όνομα της συγκεκριμένης παραμέτρου, όπως αναφέρεται στο στοιχείο <wsdl:part >του αρχείου WSDL, αν το αρχείο αναφέρεται στις παραμέτρους με το SOAPBinding style RPC, ή όπως αναφέρεται στο αντίστοιχο στοιχείο XML του αρχείου παραμέτρων, αν το WSDL αναφέρεται στις παραμέτρους με το SOAPBinding style Document. Η default τιμή της είναι το όνομα της παραμέτρου
β) targetNamespace: Το XML Namespace στο οποίο ανήκει το στοιχείο XML που περιγράφει την παράμετρο.
γ) header: Boolean τιμή η οποία περιγράφει αν η τιμή της παραμέτρου υπάρχει στο SOAPHeader. Η default τιμή είναι false.

Για να δημιουργήσει τις πληροφορίες για τις τιμές που επιστρέφονται από την κλήση μιας μεθόδου τις υπηρεσίας στο αρχείο WSDL, χρησιμοποιείται το annotation javax.ws.WebResult, το οποίο δέχεται τις εξής παραμέτρους:

α) name: Το όνομα της συγκεκριμένης τιμής, όπως αναφέρεται στο στοιχείο <wsdl:part >του αρχείου WSDL, αν το αρχείο αναφέρεται στις τιμές με το SOAPBinding style RPC, ή όπως αναφέρεται στο αντίστοιχο στοιχείο XML του αρχείου παραμέτρων, αν το WSDL αναφέρεται στις τιμές με το SOAPBinding style Document. Η default τιμή της είναι η λέξη result.
β) targetNamespace: Το XML Namespace στο οποίο ανήκει το στοιχείο XML που περιγράφει την τιμή.

Δημιουργώντας την κλάση, τοποθετώντας τα κατάλληλα annotations και γράφοντας τις μεθόδους που επιθυμούμε να εκδώσουμε σαν μεθόδους, έχουμε δημιουργήσει μια διαδικτυακή υπηρεσία SOAP. Το μόνο που μένει είναι να την μεταγλωττίσουμε και να την τρέξουμε σε έναν Application Server, όπως ο Glashfish της Oracle. Αν ο εν λόγω Application Server είναι συνδεδεμένος στο διαδίκτυο, τότε η διαδικτυακή μας υπηρεσία είναι διαθέσιμη και μπορεί ο καθένας να την καλέσει.

Όπως είπαμε, για να καλέσουμε τις μεθόδους μιας διαδικτυακής υπηρεσίας μέσα στην εφαρμογή μας, θα πρέπει η εφαρμογή μας να μπορεί να δημιουργήσει το κατάλληλο μήνυμα SOAP, να το στείλει στην κατάλληλη διεύθυνση και να επεξεργαστεί την απάντηση. Τα εργαλεία της πλατφόρμας Java αυτοματοποιούν το μεγαλύτερο μέρος αυτής της διαδικασίας. Οι πληροφορίες που χρειαζόμαστε για να συντάξουμε το κατάλληλο μήνυμα SOAP για την κλήση των μεθόδων της υπηρεσίας βρίσκονται, όπως είπαμε στο αρχείο WSDL. Τα εργαλεία της πλατφόρμας διαβάζουν αυτό το αρχείο και δημιουργούν κλάσεις με μεθόδους οι οποίες δημιουργούν τα κατάλληλα μηνύματα, τα στέλνουν στην σωστή διεύθυνση, λαμβάνουν την απάντηση και την τοποθετούν σε αντικείμενα έτοιμα για χρήση.

Για να δημιουργήσουμε αυτές τις κλάσεις αρκεί να καλέσουμε το εργαλείο wsimport , που είναι διαθέσιμο στην πλατφόρμα Java SE με παραμέτρους -keep -s directory_name my.server.com/myWebService?wsdl, όπου directory_name το σχετικό path του φακέλου στον οποίο θέλουμε να αποθηκεύσουμε τα αρχεία των κλάσεων και my.server.com/myWebService?wsdl το URL στο οποίο βρίσκεται το αρχείο WSDL της υπηρεσίας του παραδείγματος.

Στις κλάσεις που δημιουργεί το εργαλείο υπάρχει σίγουρα μία κλάση το αντικείμενο της οποίας αντιπροσωπεύει ένα instance της διαδικτυακής υπηρεσίας, καθώς και μια κλάση που αντιπροσωπεύει ένα proxy αντικείμενο μέσω του οποίου μπορούμε να καλέσουμε τις μεθόδους της υπηρεσίας. Αντικέιμενο αυτής της κλάσης μπορούμε να δημιουργήσουμε καλώντας μια μέθοδο του αντικειμένου της διαδικτυακής υπηρεσίας, η οποία το δημιουργεί. Επίσης δημιουργούνται κλάσεις για όλους τους σύνθετους τύπους δεδομένων που χρησιμοποιούνται στις διάφορες μεθόδους της υπηρεσίας. Κάνοντας χρήση αυτών των κλάσεων μπορούμε να καλέσουμε τις μεθόδους της υπηρεσίας.

4.3 RESTful Web Services με την χρήση της Java

Σε αντίθεση με τις SOAP Web Services, για να δημιουργήσουμε RESTful Web Services δεν χρειάζεται να κατασκευάσουμε συγκεκριμένα μηνύματα κάνοντας χρήση ενός πρωτοκόλλου. Το μόνο που χρειάζεται είναι να μπορούμε να επικοινωνήσουμε κάνοντας χρήση του πρωτοκόλλου HTTP, να διαχειριστούμε τις αιτήσεις του, καλώντας διαφορετικές μεθόδους της εφαρμογής μας για αιτήσεις σε διαφορετική ηλεκτρονική διεύθυνση και με διαφορετική μέθοδο του πρωτοκόλλου και να μπορούμε να δημιουργήσουμε μηνύματα στην μορφή / στις μορφές απεικόνισης των δεδομένων (XML/JSON).

Επίσης για να καλέσεις κάποια RESTful Web Service από τον κώδικα της εφαρμογής - πελάτη, αρκεί ομοίως να έχεις την δυνατότητα επικοινωνίας χρησιμοποιώντας το πρωτόκολλο HTTP, να μπορείς να δημιουργήσεις μηνύματα στη μορφή / στις μορφές απεικόνισης των δεδομένων (XML/JSON), καθώς και να γνωρίζεις τις ηλεκτρονικές διευθύνσεις στις οποίες βρίσκονται οι διάφορες λειτουργίες της υπηρεσίας (διαχείριση πόρων).

Η πλατφόρμα Java προσφέρει αρκετές λύσεις για την υλοποίηση RESTful διαδικτυακών υπηρεσιών. Το standard πρότυπο, όπως περιγράφεται από την έκδοση της Java Enterprise Edition 6 και έπειτα, είναι το JSR 339: JAX-RS (Java API for RESTful Web Services). Μια εναλλακτική λύση είναι η χρήση των RestControllers του Spring Web MVC Framework.

Το JAX-RS είναι ένα σύνολο από κλάσεις της Java, σχεδιασμένες για να μας διευκολύνουν να γράψουμε διαδικτυακές υπηρεσίες που ακολουθούν την αρχιτεκτονική REST.

Το πρότυπο αυτό διαθέτει annotations τα οποία διαχειρίζονται τις αιτήσεις HTTP και τις προωθούν στις κατάλληλες μεθόδους. Στις διαδικτυακές εφαρμογές γραμμένες σε γλώσσα Java αυτή τη αρμοδιότητα την έχουν τα Servlets. Για να γράψουμε λοιπόν μια εφαρμογή που θα χρησιμοποιεί το πρότυπο JAX-RS, θα πρέπει την διαχείριση των ηλεκτρονικών διευθύνσεων με τις οποίες θα καλούμε τις μεθόδους της υπηρεσίας να την αναλάβει ένα Servlet που θα υποστηρίζει το πρότυπο. Υπάρχουν αρκετές υλοποιήσεις του προτύπου και των εργαλείων που χρειάζεται μια εφαρμογή για να το χρησιμοποιήσει. Η Oracle έχει δημιουργήσει μια υλοποίηση που ονομάζεται Jersey.

Μια διαδικτυακή υπηρεσία με την χρήση του JAX-RS είναι κατ΄ αρχάς μια διαδικτυακή εφαρμογή Java. Αυτό σημαίνει πως διαθέτει το αρχείο web.xml, στο οποίο καταγράφονται οι ρυθμίσεις της εφαρμογής. Σε αυτό το αρχείο βρίσκεται η ρύθμιση στην οποία ορίζεται ποιό Servlet θα αναλάβει την διαχείριση των αιτήσεων για ένα συγκεκριμένο URL Pattern. Εκεί τοποθετούμε το αντίστοιχο Servlet της υλοποίησης που χρησιμοποιούμε. Θα πρέπει επίσης να ορίσουμε και την ρύθμιση η οποία θα ενημερώνει το εν λόγω Servlet σε ποιό πακέτο Java της εφαρμογής μας υπάρχουν οι κλάσεις που χρησιμοποιούν τα annotations του προτύπου.

Ειπαμε πως για να υλοποιήσουμε μια διαδικτυακή υπηρεσία που ακολουθεί την αρ-

χιτεκτονική REST, χρειαζόμαστε δύο μηχανισμούς. Ο πρώτος είναι η διαχείριση των αιτήσεων και η κλήση της κατάλληλης μεθόδου ανάλογα με την αίτηση και ο δεύτερος η δημιουργία της απάντησης στην μορφή απεικόνισης των δεδομένων (XML/JSON) . Το Servlet της υλοποίησης του προτύπου στο οποίο αναθέσαμε την διαχείριση των αιτήσεων χάρη στις παραπάνω ρυθμίσεις διαθέτει και τους δύο μηχανισμούς, τους οποίους εκμεταλλευόμαστε κάνοντας χρήση των κατάλληλων annotations.

Τα annotations του JAX-RS με τα οποία δημιουργούμε RESTful διαδικτυακές υπηρεσίες είναι τα ακόλουθα:

- @Path - Αυτό το annotation είναι υπεύθυνο για την κλήση της κατάλληλης μεθόδου ανάλογα με την ηλεκτρονική διεύθυνση της αίτησης. Τοποθετείται στον ορισμό μιας κλάσης ή μιας μεθόδου και έχει ώς παράμετρο την ηλεκτρονική διεύθυνση που αφορά τις μεθόδους της κλάσεις ή καλεί την εν λόγω μέθοδο αντίστοιχα. Η ηλεκτρονική διεύθυνση έχει την δυνατότητα να περιέχει μεταβλητές.
- @GET - Αυτό το annotation τοποθετείται στον ορισμό μιας μεθόδου και ορίζει πως η μέθοδος θα καλείται μόνο όταν οι αιτήσεις προς την ηλεκτρονική της διεύθυνση έχουν γίνει με την μέθοδο GET του πρωτοκόλλου HTTP.
- @POST - Αυτό το annotation τοποθετείται στον ορισμό μιας μεθόδου και ορίζει πως η μέθοδος θα καλείται μόνο όταν οι αιτήσεις προς την ηλεκτρονική της διεύθυνση έχουν γίνει με την μέθοδο POST του πρωτοκόλλου HTTP.
- @PUT - Αυτό το annotation τοποθετείται στον ορισμό μιας μεθόδου και ορίζει πως η μέθοδος θα καλείται μόνο όταν οι αιτήσεις προς την ηλεκτρονική της διεύθυνση έχουν γίνει με την μέθοδο PUT του πρωτοκόλλου HTTP.
- @DELETE - Αυτό το annotation τοποθετείται στον ορισμό μιας μεθόδου και ορίζει πως η μέθοδος θα καλείται μόνο όταν οι αιτήσεις προς την ηλεκτρονική της διεύθυνση έχουν γίνει με την μέθοδο DELETE του πρωτοκόλλου HTTP.
- @HEAD - Αυτό το annotation τοποθετείται στον ορισμό μιας μεθόδου και ορίζει πως η μέθοδος θα καλείται μόνο όταν οι αιτήσεις προς την ηλεκτρονική της διεύθυνση έχουν γίνει με την μέθοδο HEAD του πρωτοκόλλου HTTP.
- @PathParam - Αυτό το annotation τοποθετείται στον ορισμό μιας παραμέτρου μιας μεθόδου της οποίας η ηλεκτρονική διεύθυνση που την καλεί περιέχει μεταβλητές που ορίζονται στο annotation @Path. π.χ. @Path("/example/{username}"). Παίρνει σαν παράμετρο το όνομα της μεταβλητής που ορίζεται στο @Path. Σε αυτό το παράδειγμα ο ορισμός της μεθόδου θα ήταν ο εξής:

```
1  @Path("/example/{username}}")
2  public ExampleClass returnExample(
3  @PathParam("username") String username){
4          .........
5  }
```

- @QueryParam - Αυτό το annotation τοποθετείται στον ορισμό μιας παραμέτρου μιας μεθόδου της οποίας η ηλεκτρονική διεύθυνση που την καλεί περιέχει μεταβλητές οι οποίες ορίζονται με τον τρόπο που ορίζονται οι μεταβλητές σε μια αίτηση τύπου GET. (example.com/example?x=1&y=2). Σε αυτό το παράδειγμα ο ορισμός της μεθόδου θα ήταν ο εξής:

```
1   @Path("/example")
2   public ExampleClass returnExample(
3   @QueryParam("x") String x,
4   @QueryParam("y") String y){
5           ........
6   }
```

https://v2.overleaf.com/project/5b9cb669a4eb99158ac0d32b

- @Consumes - Αυτό το annotation τοποθετείται στον ορισμό μιας μεθόδου και ορίζει τον τύπο δεδομένων που δέχεται η μέθοδος. Δέχεται σαν παράμετρο το αποδεκτό / τα αποδεκτά MIME Types. Επίσης είναι το annotation με το οποίο η υλοποίηση του JAX-RS ενημερώνεται για την μορφή των δεδομένων που στέλνει η αίτηση και αναλαμβάνει την μετατροπή τους στους τύπους δεδομένων των παραμέτρων της μεθόδου.

- @Produces - Αυτό το annotation τοποθετείται στον ορισμό μιας μεθόδου και ορίζει τον τύπο δεδομένων που επιστρέφει η μέθοδος. Δέχεται σαν παράμετρο το αποδεκτό / τα αποδεκτά MIME Types Επίσης είναι το annotation με το οποίο η υλοποίηση του JAX-RS ενημερώνεται για την μορφή των δεδομένων στην οποία πρέπει να μετατρέψει το αποτέλεσμα της εκτέλεσης της μεθόδου ώστε να δημιουργήσει την απάντηση.

- @Provider - Με αυτό το annotation δίνεται η δυνατότητα στον προγραμματιστή της εφαρμογής να υλοποιήσει δικά του εργαλεία τα οποία αλληλεπιδρούν με την υλοποίηση του προτύπου.

Δημιουργώντας μια διαδικτυακή εφαρμογή με αυτές τις ρυθμίσεις και υλοποιώντας τις κλάσεις και τις μεθόδους με τις οποίες αυτή η εφαρμογή θα διαχειρίζεται τις κατάλληλες αιτήσεις και θα επιστρέφει τις κατάλληλες απαντήσεις έχουμε υλοποιήσει μια RESTful διαδικτυακή υπηρεσία. Το μόνο που μένει είναι να την μεταγλωττίσουμε και να την τρέξουμε σε ένα Servlet Container / Server, όπως ο Apache Tomcat. Αν ο εν λόγω Server είναι συνδεδεμένος στο διαδίκτυο, τότε η διαδικτυακή μας υπηρεσία είναι διαθέσιμη και μπορεί ο καθένας να την καλέσει.

Για να καλέσουμε μια RESTful διαδικτυακή υπηρεσία, η εφαρμογή μας απλά θα πρέπει να έχει την δυνατότητα να επικοινωνεί με το πρωτόκολλο HTTP, καθώς και να μπορεί να διαχειριστεί δεδομένα στις κατάλληλες μορφές απεικόνισης. Σε αντίθεση με τις διαδικτυακές υπηρεσίες SOAP, δεν χρειαζόμαστε κάποιες έτοιμες κλάσεις των οποίων τις μεθόδους θα καλέσουμε. Ο κώδικας μας απλά θα πρέπει να στέλνει τις κατάλληλες αιτήσεις HTTP και να επεξεργάζεται τις απαντήσεις. Αυτό συμβαίνει

διότι η σύνταξη των αντίστοιχων αιτήσεων, όπως και η εξαγωγή των δεδομένων από τις αντίστοιχες απαντήσεις είναι διαδικασίες πολύ πιο απλές από την περίπτωση των SOAP διαδικτυακών υπηρεσιών, στις οποίες ένα αρκετά πολύπλοκο μήνυμα του πρωτοκόλλου SOAP πρέπει να συνταχθεί και να αποσταλλεί / πρέπει να ληφθεί και να γίνει εξαγωγή των δεδομένων απο την απάντηση.

Όπως είδαμε στο προηγούμενο κεφάλαιο, η αρχιτεκτονική MVC στις διαδικτυακές εφαρμογές προστάζει κάθε αίτηση που δέχεται η εφαρμογή να διοχετεύεται για διαχείριση σε κάποιον κατάλληλο Controller. Επίσης είδαμε πως μια διαδικτυακή υπηρεσία που έχει δημιουργηθεί με την χρήση κάποιας υλοποίησης του JAX-RS, είναι στην ουσία μια διαδικτυακή εφαρμογή, επικοινωνεί δηλαδή κάνοντας χρήση του πρωτοκόλλου HTTP. Το Spring Web MVC, όπως είδαμε, είναι, όπως λέει και το όνομα του, ένα framework το οποίο υλοποιεί την αρχιτεκτονική MVC, για να δημιουργήσει Web εφαρμογές. Δηλαδή μια από τις δύο απαραίτητες προϋποθέσεις για την ανάπτυξη μιας RESTful διαδικτυακής υπηρεσίας ήδη ικανοποιείται από το εν λόγω framework. Η δεύτερη προϋπόθεση, όπως είπαμε είναι να υπάρχει η δυνατότητα δημιουργίας της απάντησης χρησιμοποιώντας τις κατάλληλες μορφές απεικόνισης των δεδομένων. Αυτήν την προϋπόθεση, το Spring Web MVC Framework την ικανοποιεί, με την χρήση του annotation @RestController.

Η λογική ανάπτυξης της διαδικτυακής υπηρεσίας με την χρήση του Spring Web MVC Framework είναι η ίδια, απλά η κλήση των μεθόδων του RestController, καθώς και η μετατροπή των παραμέτρων τις αίτησης στις μεταβλητές των παραμέτρων της μεθόδου γίνεται με τον τρόπο που αναλύσαμε στο προηγούμενο κεφάλαιο και αν στο αντίστοιχο mapping annotation κάθε μεθόδου ορίσεις τις παραμέτρους consumes και produces, έχεις το ίδιο αποτέλεσμα που θα είχες χρησιμοποιώντας τα αντίστοιχα annotations της υλοποίησης του JAX-RS.

Η έκδοση της διαδικτυακής υπηρεσίας που δημιουργήθηκε με την χρήση του Spring Web MVC γίνεται όπως κάθε διαδικτυακή εφαρμογή που δημιουργείται με την χρήση του εν λόγω framework.

4.4 Σύγκριση των δύο τεχνολογιών

Σε αυτό το κεφάλαιο είδαμε τις δύο διαφορετικές κατηγορίες διαδικτυακών υπηρεσιών, SOAP/ RESTful. Αυτές οι δύο διαφορετικές προσεγγίσεις έχουν τον ίδιο στόχο. Να δημιουργήσουν ένα API, το οποίο μπορεί να καλεστεί από εφαρμογές που είναι γραμμένες σε διαφορετικές γλώσσες, που τρέχουν σε διαφορετικές πλατφόρμες, διαφορετικά λειτουργικά συστήματα. Να κάνουν ακόμα πιο εύκολη την επαναχρησιμοποίηση της λειτουργικότητάς τους, αφού είναι διαθέσιμες στο δίκτυο και μπορούν να χρησιμοποιηθούν ανα πάσα στιγμή από πολλαπλές εφαρμογές. Μια διαδικτυακή υπηρεσία η οποία έχει πρόσβαση π.χ. σε μια βάση δεδομένων μπορεί να επαναχρησιμοποιηθεί από πολλαπλές εφαρμογές γραμμένες σε διαφορετικές γλώσσες η καθεμία χωρίς να χρειάζονται είτε πρόσβαση στα ίδια δεδομένα, είτε αντίγραφο αυτών των δεδομένων, καθώς και χωρίς να χρειάζονται την ύπαρξη κώδικα υλοποίησης της λειτουργικότητας της υπηρεσίας, για κάθε μια διαφορετική τεχνολογία υλοποίησης. Αυτό είναι το μεγάλο πλεονέκτημα της χρήσης διαδικτυακών υπηρεσιών και ο λόγος που η ανάπτυξή τους είναι τόσο δημοφιλής. Η κάθε κατηγορία όμως εστιάζει σε διαφορετικά χαρακτηριστικά και έχει διαφορετικά πλεονεκτήματα.

Οι SOAP Web Services, όπως είπαμε περιγράφονται από ένα αρχείο XML, που ονομάζεται WSDL. Ο κάθε προγραμματιστής που θέλει να χρησιμοποιήσει την λειτουργικότητα της υπηρεσίας στην εφαρμογή του χρησιμοποιεί τα εργαλεία της αντίστοιχεις πλατφόρμας που χρησιμοποιεί για να πάρει από αυτό το αρχείο όλες τις πληροφορίες που χρειάζεται για να καλέσει την υπηρεσία. Αυτό σημαίνει πως κάθε SOAP Web Service είναι πλήρως ορισμένο κατά την έκδοση. Ο προγραμματιστής που διαβάζει το αρχείο ξέρει από την πρώτη μέχρι και την τελευταία λειτουργία της υπηρεσίας. Αυτό είναι ένα σημαντικό πλεονέκτημα έναντι των RESTful Web Services, στις οποίες δεν αναφέραμε κάποια υποχρέωση του δημιουργού της υπηρεσίας να ενημερώσει τους χρήστες τις για τις δυνατότητές τις. Αναφέραμε μόνο, σαν καλή πρακτική, το HATEOAS, μια κατεύθυνση της αρχιτεκτονικής που έχει ώς σκοπό να βοηθήσει τις εφαρμογές - πελάτες της υπηρεσίας και τους προγραμματιστές τους να μάθουν περισσότερες πληροφορίες για την λειτουργικότητά της. Η μεγάλη διαφορά εδώ είναι πως στην περίπτωση των SOAP Web Services, ΕΙΝΑΙ ΑΔΥΝΑΤΟΝ να εκδώσεις μια υπηρεσία χωρίς αρχείο WSDL. Θεωρητικά αν στείλεις μια αίτηση στην σωστή διεύθυνση, με το σωστό μήνυμα, τότε η υπηρεσία θα σου στείλει την σωστή απάντηση. Αλλά όπως είπαμε το μήνυμα που πρέπει να στείλεις είναι αρκετά πιο πολύπλοκο από την απλή HTTP αίτηση που απαιτείται στην περίπτωση μιας υπηρεσίας REST, τόσο, που είναι αδύνατο να συντάξεις μια σωστή αίτηση και να εξάγεις τα δεδομένα από μια απάντηση μιας υπηρεσίας SOAP, χωρίς να κάνεις χρήση των πληροφοριών που περιέχονται στο αρχείο WSDL. Το μειονέκτημα όμως σε αυτήν την περίπτωση είναι αρκετά σημαντικό. Σε περίπτωση που ο πάροχος θέλει να κάνει οποιαδήποτε αλλαγή, θα πρέπει να ενημερώσει το αρχείο WSDL της υπηρεσίας, κάνοντας το προηγούμενο αρχείο περιγραφής και όλες τις εφαρμογές που έχουν δημιουργηθεί, χρησιμοποιώντας τις πληροφορίες του outdated. Θα πρέπει ο προγραμματιστής της εφαρμογής πελάτη να πάρει το καινούριο αρχείο και να κάνει τις απαραίτητες αλλαγές για να μπορεί να χρησιμοποιήσει την νέα υπηρεσία.

Στην αντίστοιχη περίπτωση ο πάροχος μιας υπηρεσίας REST μπορεί να κάνει όσες αλλαγές θέλει χωρίς να είναι απαραίτητο να ενημερώσει κανέναν και οι ήδη υπάρχουσες εφαρμογές που χρησιμοποιούν την υπηρεσία δεν θα επηρεαστούν, εφόσον βέβαια οι αλλαγές δεν αφορούν τον τρόπο με τον οποίο επικοινωνούν (αλλαγή κάποιας διεύθυνσης). Το μεγάλο μειονέκτημα είναι πως η υπηρεσίες αυτού του τύπου δεν είναι πλήρως ορισμένες από την αρχή. Πρέπει ο πάροχός τους να δημιουργήσει π.χ. κάποιο documentation στο οποίο ο προγραμματιστής των εφαρμογών - πελατών θα ανατρέξει για να αντλήσει πληροφορίες για αυτές.

Κεφάλαιο 5

Spring Framework - Το δημοφιλέστερο IoC framework

5.1 Εισαγωγή

Στις αρχικές εκδόσεις της, η πλατφόρμα Java Enterprise Edition, παρά το γεγονός ό,τι έδινε λύσεις σε πολλά προβλήματα που σχετίζονταν με την ανάπτυξη εφαρμογών μεγάλης κλίμακας, για τις ανάγκες εταιριών, ήταν κατά βάση πολύπλοκη στην χρήση της. Έτσι, το 2003 δημιουργήθηκε το Spring Framework με σκοπό να δώσει πιο εύχρηστες εναλλακτικές στα διάφορα προβλήματα των Enterprise Applications.
Το Spring Framework βασίζεται σε δικές του σχεδιαστικές ιδέες, μερικές από οποίες απομακρύνονται από τα πρότυπα της JEE. Ακολουθεί όμως πολλά από αυτά τα πρότυπα. Πιο συγκεκριμένα ακολουθεί τα:

- Servlet API - JSR 340
- WebSocket API - JSR 356
- Concurrency Utilities - JSR 236
- JSON Binding API - JSR 367
- Bean Validation - JSR 303
- Java Persistence API - JSR 338
- Java Messaging Service - JSR 914
- Java Transaction API - JSR 907

Οι χρήστες μπορούν επίσης να επιλέξουν να ακολουθήσουν τα πρότυπα Dependency Injection - JSR 330 / Common Annotations - JSR 250 , αντί για τους μηχανισμούς του framework, αφού υποστηρίζονται και τα δύο.

Το Spring Framework είναι σχεδιασμένο βασισμένο σε κάποιες αρχές:

- Να υπάρχει η δυνατότητα επιλογής εργαλείων σε κάθε επίπεδο: Το Spring Framework δίνει την δυνατότητα αλλαγής στην επιλογή διαφόρων εξωτερικών βιβλιοθηκών μόνο μέσα από το configuration, χωρίς περαιτέρω αλλαγή στον κώδικα.
- Ευελιξία: Να μην περιορίζει τον χρήστη σε συγκεκριμένο τρόπο επίλυσης κάθε θέματος.
- Να μην υπάρχουν υπερβολικά μεγάλες αλλαγές που θα κάνουν τις παλιές εκδόσεις απαρχαιωμένες, αλλά να εξελίσσεται σταδιακά με έμφαση στην συντήρηση παλαιοτέρων εφαρμογών.
- Να δίνει έμφαση στην ποιότητα του κώδικα και να ενθαρρύνει τις βέλτιστες πρακτικές.

Σήμερα, με τον όρο Spring, είτε εννοούμε το Spring Core, το βασικό και αρχικό κομμάτι του Spring Framework, είτε ονομάζουμε το σύνολο των projects που έχουν δημιουργηθεί στο πέρασμα των χρόνων από το ίδρυμα Spring, όπως τα Spring Boot, Spring Cloud, Spring Data, Spring Security, Spring for Android, Spring Integration, Spring Batch, Spring LDAP κ.α.

5.2 Spring Core - Η καρδιά του Spring Framework

Το Spring Framework είναι χτισμένο πάνω σε μια αρχή που ονομάζεται αντιστροφή του ελέγχου - Inversion of Control (IoC) ή Dependency Injection (DI). Σύμφωνα με αυτήν την αρχή, τα αντικείμενα της εφαρμογής μας δηλώνουν τις εξαρτήσεις τους από άλλα αντικείμενα (dependencies) μόνο σαν παραμέτρους στους κατασκευαστές τους ή σε κάποιες setter μεθόδους, αντί να τις δημιουργούν σαν νέα αντικείμενα κάνοντας χρήση της εντολής new στον κώδικά τους. Το εκάστοτε DI Framework, στην περίπτωσή μας το Spring Framework αναλαμβάνει να ανακαλύψει τα αντικείμενα - dependencies - κατά την κλήση του κατασκευαστή ή των αντίστοιχων setters, αν υπάρχουν, και να τα τοποθετήσει στις παραμέτρους. Αν τα κατάλληλα αντικείμενα δεν υπάρχουν, αναλαμβάνει να τα δημιουργήσει.

Στην προηγούμενη παράγραφο αναφέρουμε πώς το Framework αναλαμβάνει να ανακαλύψει αντικείμενα που ήδη έχουν δημιουργηθεί και πληρούν τις προυποθέσεις για να δοθούν ώς παράμετροι κατά την δημιουργία νέων αντικειμένων. Αυτό συμβαίνει διότι το Spring Framework διαθέτει ένα μηχανισμό που ονομάζεται Spring Container. Αυτός ο μηχανισμός είναι υπεύθυνος για την διαχείριση των αντικειμένων μιας εφαρμογής δημιουργημένης με την χρήση του Spring Framework για όλον τον κύκλο ζωής τους, από την δημιουργία τους μέχρι την καταστροφή τους, την απελευθέρωση δηλαδή της μνήμης που καταλαμβάνουν. Τα αντικείμενα αυτά ονομάζονται Beans. Παρακάτω παρουσιάζεται ένα σχήμα το οποίο δείχνει τον ρόλο του Spring IoC Container στην δημιουργία εφαρμογών στο περιβάλλον του Spring Framework:

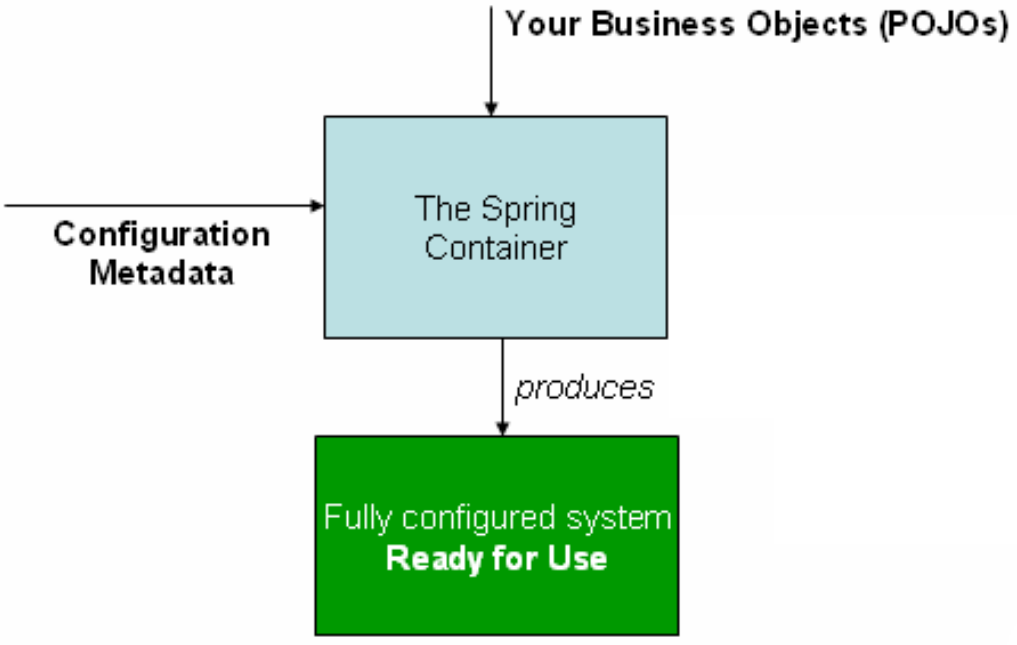

Figure 5.1: Επισκόπηση χρήσης του Spring IoC Container

Όπως φαίνεται και στο σχήμα, εκτός από τα αντικείμενά της εφαρμογής μας, ο Spring Container χρειάζεται και κώδικα ο οποίος ορίζει διάφορες ρυθμίσεις του και ανάλογα με τα περιεχόμενα αυτού του κώδικα ορίζεται η συμπεριφορά του κατά την διαχείριση των αντικειμένων μας. Στις πρώτες εκδόσεις του Spring Framework αυτές οι ρυθμίσεις περιέχονταν αυστηρά σε αρχεία XML. Σήμερα όμως, υπάρχει η δυνατότητα να οριστούν οι ρυθμίσεις αυτές με τρείς διαφορετικούς τρόπους:

- αρχεία XML - Η ρύθμιση του Spring IoC Container μπορεί ακόμα να περίεχεται - παραδοσιακά - σε αρχεία XML.
- Java Annotations - Οι μετέπειτα εκδόσεις του Spring Framework περιέχουν μια σειρά απο Annotations με τα οποία μπορούμε να ορίσουμε τις κατάλληλες ρυθμίσεις.
- Java Code - Οι μετέπειτα εκδόσεις του Spring Framework περιέχουν μια σειρά από κλάσεις και μεθόδους με τις οποίες μπορούμε να ορίσουμε τις κατάλληλες ρυθμίσεις.

Στην εισαγωγή είπαμε πως το Spring Framework μας δίνει την δυνατότητα η επιλογή των εργαλείων που χρησιμοποιούμε στην εφαρμογή μας να γίνεται μέσα από το Configuration. Αυτό συμβαίνει διότι μέσα από αυτήν την διαδικασία δημιουργείται και ρυθμίζεται ένα σύνολο από Beans, δηλαδή αντικείμενα τα οποία διαχειρίζεται ο Spring Container, κάθε ένα από τα οποία είναι υπεύθυνο για κάποια/ες εργασία/ες του αντίστοιχου εργαλείου.

Στον κώδικα του Configuration είτε αυτός βρίσκεται σε αρχεία XML, είτε σε κλάσεις Java, υπάρχουν ορισμοί των διαφόρων Beans (Bean Definitions). Αυτό σημαίνει πως ορίζονται πλήρως τα αντικείμενα που θα δημιουργηθούν κατά την επεξεργασία του Configuration. Μια πολύ σημαντική παράμετρος στο Configuration των Beans της εφαρμογής μας είναι αυτό που ονομάζουμε Bean Scopes, δηλαδή πόσα διαφορετικά instances αντικειμένων με το ίδιο Bean Definition επιτρέπεται να υπάρχουν και να διαχειρίζονται από τον ίδιο Spring Container. Υπάρχουν 6 διαφορετικοί τύποι Bean Scope. Οι 4 από αυτούς έχουν νόημα μόνο σε διαδικτυακές εφαρμογές. Αυτοί είναι οι παρακάτω:

- singleton - Αυτή είναι η Default τιμή του Bean Scope ενός Bean Definition. Επιτρέπει την ύπαρξη μόνο ενός instance αυτού του αντικειμένου σε ένα Spring Container. Όλα τα αντικείμενα που έχουν σαν Dependency αυτό το Bean μοιράζονται το ίδιο instance.
- prototype - Για κάθε αντικείμενο που έχει σαν Dependency αυτό το Bean δημιουργείται ένα νέο instance.

- request - Δημιουργείται νέο instance του Bean Definition για κάθε διαφορετική αίτηση HTTP. Έχει νόημα μόνο σε διαδικτυακές εφαρμογές.
- session - Δημιουργείται νέο instance του Bean Definition για κάθε διαφορετική HTTP Session. Έχει νόημα μόνο σε διαδικτυακές εφαρμογές.
- application - Δημιουργείται νέο instance του Bean Definition για κάθε διαφορετικό ServletContext. Έχει νόημα μόνο σε διαδικτυακές εφαρμογές.
- websocket - Δημιουργείται νέο instance του Bean Definition για κάθε διαφορετικό WebSocket. Έχει νόημα μόνο σε διαδικτυακές εφαρμογές.

Υπάρχει η δυνατότητα να καλέσουμε Callback μεθόδους σε σημεία του κύκλου ζωής ενός Bean, όπως μετά την κατασκευή του και πριν την καταστροφή του.

Το Spring Framework μας προσφέρει ένα σύνολο από Interfaces, τα οποία μπορούμε να κάνουμε Implement στις κλάσεις μας, και να αποκτήσουμε πρόσβαση σε αντικείμενα που περιέχουν σημαντικές πληροφορίες για τα αντικείμενα που διαχειρίζεται ο Spring Container, αλλά και για τον ίδιο. Τα Interfaces αυτά είναι γνωστά ώς Aware Interfaces, διότι καθιστούν το Bean το οποίο τα κάνει Implement ενήμερο για διάφορες πληροφορίες. Σημαντικά Interfaces αυτού του τύπου είναι τα ApplicationContextAware, BeanNameAware, ServletConfigAware, ServletContextAware.

Το Spring Framework μας δίνει την δυνατότητα χρησιμοποιώντας τα κατάλληλα Annotations, να ορίσουμε τον Spring Container υπεύθυνο για το Dependency Injection των καταλλήλων Beans στα Annotated Fields μας. Το Annotation που χρησιμοποιεί το Spring Framework γι΄ αυτή τη δουλειά είναι το @Autowired. Αν κάποιο πεδίο κάποιας από τις κλάσεις μας έχει το Annotation @Autowired, τότε ο Spring Container αναλαμβάνει να ψάξει το Application Context για το κατάλληλο Bean και να το κάνει Inject σε αυτό το πεδίο. Υπάρχουν δύο διαφορετικοί τύποι Autowiring:

α) Autowiring με βάση το όνομα: Σε αυτήν την περίπτωση ο Spring Container ψάχνει για Bean με το ίδιο όνομα με αυτό του field.
β) Autowiring με βάση τον τύπο: Σε αυτήν την περίπτωση ο Spring Container ψάχνει για Beans της ίδιας κλάσης με το αντίστοιχο field.

Στην δεύτερη περίπτωση επειδή υπάρχει η πιθανότητα να υπάρχουν δύο Beans του ίδιου τύπου στο Application Context, υπάρχει ένα Annotation με το οποίο μπορούμε να καταδείξουμε ένα Bean το οποίο θα γίνεται Inject στα Autowired fields αυτού του τύπου. Αυτό το Annotation είναι το @Primary.

Το Spring Framework μας δίνει την δυνατότητα να δημιουργήσουμε Beans της εφαρμογής απλά προσθέτοντας ένα Annotation στην κλάση του. Επίσης προσφέρει μερικά Annotations, τα οποία δίνουν στον Spring Container περισσότερες πληροφορίες για αυτό το Bean. Αυτά τα Annotations ονομάζονται στερεότυπα (Stereotypes) και είναι τα παρακάτω:

- @Component - Αυτό είναι το γενικό Annotation. Δηλώνει πώς η συγκεκριμένη κλάση ορίζει ένα Bean, το οποίο πρέπει να δημιουργηθεί κατά την επεξεργασία του Configuration.
- @Service - Αυτό το Annotation λειτουργεί όπως και το @Component με την διαφορά ότι δίνει την πληροφορία στο Spring Framework πως η συγκεκριμένη κλάση ορίζει μια υπηρεσία (Service)
- @Repository - Αυτό το Annotation λειτουργεί όπως και το @Component με την διαφορά ότι δίνει την πληροφορία στο Spring Framework πως η συγκεκριμένη κλάση ορίζει ένα Data Access Object (Repository).
- @Controller - Αυτό το Annotation λειτουργεί όπως και το @Component με την διαφορά ότι δίνει την πληροφορία στο Spring Framework πως η συγκεκριμένη κλάση ορίζει έναν MVC Controller.

By Default το Spring Framework δεν αναζητά αυτά τα Annotations σε όλες τις κλάσεις του Project μας. Θα πρέπει στο αρχείο στο οποίο είναι γραμμένο το Configuration, είτε αυτό είναι XML, είτε Java, να ορίσουμε εμείς τα πακέτα Java στα οποία υπάρχουν οι κλάσεις - Components μας.

Σε αυτό το μέρος του κεφαλαίου είδαμε το βασικό κομμάτι του Spring Framework, που είναι η κεντρική διαχείριση των αντικειμένων της εφαρμογής από τον Spring IoC Container, το Dependency Injection, και τους τρόπους με τους οποίους δημιουργούμε δικά μας Beans. Είδαμε επιγραμματικά ένα μικρό μέρος από τις δυνατότητες που μας δίνει το Configuration. Καθώς όπως είπαμε στην αρχή του κεφαλαίου, το Spring Framework είναι χτισμένο βασισμένο πάνω στην ευελιξία, οι πλήρεις δυνατότητες ρύθμισης του είναι πάρα πολλές σε αριθμό και η ανάλυσή τους ξεφεύγει από τα πλαίσια της εργασίας.

5.3 Spring Boot - Δημιουργώντας Spring Applications εύκολα

Στο προηγούμενο κεφάλαιο είπαμε πως το Spring Framework έχει πάρα πολλές δυνατότητες παραμετροποίησης μέσω του Configuration. Δυστυχώς αυτό σημαίνει πώς στο ξεκίνημα της ανάπτυξης οποιασδήποτε εφαρμογής με χρήση του, χρειάζεται να διαθέσουμε ένα εύλογο διάστημα μόνο για το αρχικό Configuration, προσθέτοντας δηλαδή και ρυθμίζοντας τα κατάλληλα Beans για τις ανάγκες μας. Σε αυτήν την διαδικασία, όπως είπαμε και προηγουμένως δημιουργούμε αρχεία στα οποία περιέχεται ο κώδικας του Configuration. Ένα απλό Starter Spring Project λοιπόν, έχει ήδη ξεκινώντας αρκετό κώδικα μόνο και μόνο για να μπορέσει να δουλέψει. Αυτό στην αρχή δημιουργίας μεγάλων Enterprise εφαρμογών βέβαια δεν είναι πρόβλημα, αλλά σε απλές εφαρμογές ειδικά εκπαιδευτικού σκοπού προσθέτει μια αχρείαστη πολυπλοκότητα.

Για να δώσει μια απάντηση σε αυτά τα θέματα, το Spring Foundation δημιούργησε ένα Project το οποίο ονομάζεται Spring Boot. Το Spring Boot δημιουργεί Spring Projects στα οποία υπάρχει ήδη ένα Configuration το οποίο είναι ικανοποιητικό στις περισσότερες περιπτώσεις εφαρμογών. Ο προγραμματιστής έπειτα μπορεί να δημιουργήσει τα δικά του αρχεία ρυθμίσεων και να κάνει τα κατάλληλα Override. Επίσης εκτός από τις διάφορες ρυθμίσεις των Modules του Spring Framework, το Spring Boot αναλαμβάνει να υλοποιήσει τις απαραίτητες ρυθμίσεις για έναν μεγάλο αριθμό δημοφιλών εξωτερικών βιβλιοθηκών, ώστε να είναι έτοιμες για χρήση το συντομότερο δυνατόν.

Το Spring Foundation έχει δημιουργήσει μια εφαρμογή, στην οποία μπορούμε να δημιουργήσουμε έτοιμα Maven ή Gradle Spring Boot Projects και να τους προσθέσουμε Dependencies από ένα σύνολο εξωτερικών βιβλιοθηκών που υποστηρίζονται. Η εφαρμογή έπειτα μας δημιουργεί ένα zip αρχείο, στο οποίο περιέχεται το Project μας, με το Configuration των Dependencies που επιλέξαμε ρυθμισμένο σύμφωνα με τις Default επιλογές του Spring Boot και έτοιμο για χρήση. Η εφαρμογή αυτή ονομάζεται Spring Initializr και, την στιγμή που γράφεται η εργασία, είναι προσβάσιμη στην ηλεκτρονική διεύθυνση https://start.spring.io

Επίσης, μια τυπική εφαρμογή γραμμένη με την βοήθεια του Spring Boot, ακόμα κι αν είναι μια διαδικτυακή εφαρμογή, είναι δομημένη σαν μια απλή εφαρμογή Java. Διαθέτει δηλαδή την μέθοδο main από την οποία ξεκινάει η εκτέλεσή της και γίνεται deploy σαν αρχείο JAR. Καθώς όμως, όπως είπαμε σε προηγούμενο κεφάλαιο, οι διαδικτυακές εφαρμογές γραμμένες σε Java χρειάζονται έναν Servlet Container για να τρέξουν, το Spring Boot διαθέτει ένα σύνολο από διαθέσιμους Embedded Servlet Containers, οι οποίοι αρχικοποιούνται, ρυθμίζονται και τρέχουν ώς μέρος της εφαρμογής.

Το Spring Boot διατηρεί ένα σύνολο από παραμέτρους τις οποίες μπορούμε να δώσουμε σαν ορίσματα κατά την εκτέλεση του JAR αρχείου της εφαρμογής μας σαν μεταβλητές περιβάλλοντος (Environment Variables). Μπορούμε να ορίσουμε τιμές σε αυτές τις μεταβλητές χρησιμοποιώντας το αρχείο application.properties. Οι μεταβλητές που ορίζουμε σε αυτό το αρχείο φορτώνονται στην εφαρμογή μας κατά την εκκίνηση της εκτέλεσής της.

Το Spring Boot επίσης προσφέρει ένα module το οποίο ονομάζεται Actuator. Πρόκειται για λογισμικό το οποίο προσθέτουμε στην εφαρμογή μας για να εκτελεί διάφορες μετρήσεις και να μας παρουσιάζει ένα σύνολο από δεδομένα σχετικά με την εφαρμογή μας. Στην πιο συνηθισμένη μορφή του είναι προσβάσιμο μέσω του πρωτοκόλλου HTTP, στην σχετική διεύθυνση /actuator. Κάθε διαφορετικός τύπος δεδομένων - ονομάζεται Actuator Endpoint - έχει το δικό του όνομα και είναι προσβάσιμος στην σχετική διεύθυνση /actuator/EndpointName. Μερικά σημαντικά Endpoints είναι τα παρακάτω:

- beans - Δείχνει μια λίστα με όλα τα Beans που διαχειρίζεται ο Spring IoC Container.
- health - Παρουσιάζει δεδομένα σχετικά με την υγεία της εφαρμογής.
- loggers - Διαχειρίζεται τα διάφορα αντικείμενα της εφαρμογής που είναι υπεύθυνα για το Logging.
- mappings - Δείχνει μια λίστα με όλους τους συσχετισμούς διευθύνσεων - Controllers.
- info - Παρουσιάζει ένα σύνολο από πληροφορίες σχετικές με την εφαρμογή.

Σε αυτό το μέρος του κεφαλαίου είδαμε τις βασικές διαφορές που έχει μια εφαρμογή γραμμένη με την βοήθεια του Spring Boot σε σχέση με μια απλή εφαρμογή του Spring Framework. Είδαμε επίσης τα σημαντικότερα εργαλεία που προσφέρει το Spring Boot για να κάνει την ανάπτυξη εφαρμογών Spring όσο ευκολότερη και γρηγορότερη μπορεί, καθώς και το εργαλείο που διαθέτει για να μας παρουσιάζει χρήσιμα δεδομένα για την εφαρμογή μας.

5.4 Spring Data - Εργαλείο διαχείρισης και αποθήκευσης δεδομένων

Το Spring Data είναι μια σειρά από Projects τα οποία έχουν ώς σκοπό να δημιουργήσουν ένα πρότυπο για την αναζήτηση και την οργάνωση των δεδομένων σε ένα σύνολο από διαφορετικές πλατφόρμες αποθήκευσης. Για κάθε πλατφόρμα αποθήκευσης υπάρχει και ένα διαφορετικό Spring Data Project. Τα πιο σημαντικά από αυτά είναι τα:

- Spring Data JDBC - Το συγκεκριμένο Project είναι υπεύθυνο για την δημιουργία Repositories για βάσεις δεδομένων προσβάσιμες μέσω του JDBC (Java DataBase Connector).
- Spring Data JPA - Το συγκεκριμένο Project είναι υπεύθυνο για την δημιουργία Repositories για βάσεις δεδομένων προσβάσιμες μέσω υλοποιήσεων του προτύπου JPA (Java Persistence API).
- Spring Data for Apache Solr - Το συγκεκριμένο Project είναι υπεύθυνο για την δημιουργία Repositories για την πρόσβαση στα δεδομένα της μηχανής αναζήτησης Apache Solr.
- Spring Data MongoDB - Το συγκεκριμένο Project είναι υπεύθυνο για την δημιουργία Repositories για βάσεις δεδομένων NoSQL MongoDB.

Στα πλαίσια της εργασίας θα ασχοληθούμε με Spring Data JPA Repositories.

Το βασικό Interface ενός Spring Data Project είναι το Repository. Όμως ανάλογα με την εργασία την οποία θέλουμε να εκτελέσουμε στα δεδομένα μας, υπάρχουν διάφορα Interfaces που κληρονομούν από το Repository και διαθέτουν τις κατάλληλες μεθόδους για την αντίστοιχη εργασία. Τα σημαντικότερα από αυτά τα Repositories είναι:

- JpaRepository - Το γενικότερο JPA Repository Interface. Προσφέρει μια σειρά από μεθόδους ανάκτησης δεδομένων.
- CrudRepository - Αυτός ο τύπος Repository προσφέρει μια σειρά από μεθόδους για CRUD (Create, Read, Update, Delete) λειτουργίες στα δεδομένα.
- PagingAndSortingRepository - Αυτός ο τύπος Repository προσφέρει μεθόδους σελιδοποίησης και ταξινόμησης για τα δεδομένα μας.

Για να δημιουργήσουμε ένα Repository αρκεί να δημιουργήσουμε ένα νέο Interface το οποίο θα κάνει extend το κατάλληλο Repository για τις ανάγκες μας, και στο οποίο θα προσθέσουμε σαν παραμέτρους τον τύπο δεδομένων τον οποίο θα διαχειρίζεται, καθώς και τον τύπο δεδομένων του πρωτεύοντος κλειδιού. Σε αυτό το Interface θα προσθέσουμε το στερεοτυπικό Annotation @Repository. Αυτό, όπως είδαμε θα δημιουργήσει ένα Bean μιας υλοποίησης του Interface μας την οποία αναλαμβάνει να δημιουργήσει το ίδιο το Spring Framework. Μπορούμε έπειτα να χρησιμοποιήσουμε αυτό το Bean για να καλέσουμε τις μεθόδους του Repository από οποιαδήποτε κλάση της εφαρμογής μας.

Για να μπορέσει το Spring Framework να ανακαλύψει τα Repositories μας και να δημιουργήσει της υλοποιήσεις τους, θα πρέπει σε μια κλάση Configuration να προσθέσουμε το Annotation @EnableJpaRepositories(basePackage= "my.repositories.package")

Μια πολύ σημαντική δυνατότητα που μας δίνουν τα JPA Repositories είναι το γεγονός πως μπορούμε να ορίσουμε διάφορες μεθόδους με τις οποίες μπορούμε να ανακτήσουμε κάποιο αντικείμενο από τα δεδομένα μας με βάση ένα από τα πεδία του, δίνοντας στην μέθοδο όνομα σύμφωνα με ένα συγκεκριμένο Convention, και το Spring Framework αναλαμβάνει να δημιουργήσει και αυτές τις μεθόδους καθώς δημιουργεί την υλοποίηση του Interface. π.χ.

Έχουμε ένα CrudRepository το οποίο διαχειρίζεται δεδομένα της κλάσης Car με τύπο δεδομένων του πρωτεύοντος κλειδιού Integer. Η κλάση Car διαθέτει ένα πεδίο τύπου String που ονομάζεται brand. Άν στο CrudRepository μας ορίσουμε την μέθοδο findByBrand(String brand), χωρίς να δημιουργήσουμε μια κλάση που θα την υλοποιεί, το Bean που θα δημιουργηθεί θα την περιέχει κανονικά υλοποιημένη.

Σε αυτό το μέρος του κεφαλαίου είδαμε επιγραμματικά τη δυνατότητα διαχείρισης δεδομένων που μας δίνουν τα JPA Repositories. Περισσότερες πληροφορίες για το πρότυπο JPA, την αντιστοίχιση Αντικειμένων με Σχεσιακές Βάσεις - ORM (Object Relational Mapping) και για την μόνιμη αποθήκευση δεδομένων θα δούμε στο επόμενο κεφάλαιο.

5.5 Spring Security - Προσθέτοντας ασφάλεια σε ένα Spring Application

Από την αρχή της χρήσης του διαδικτύου, όπου διάφορες εφαρμογές άρχισαν να μοιράζονται δεδομένα μέσω ενός κοινού δικτύου, η ανάγκη ύπαρξης ασφαλών εφαρμογών, όπου τα εν λόγω δεδομένα θα είναι προσβάσιμα μόνο από τους εξουσιοδοτημένους χρήστες ολοένα και αυξάνεται. Αυτό έχει ώς αποτέλεσμα η προσοχή ενός μεγάλου μέρους προγραμματιστών να στραφεί στην αναζήτηση τρόπων ανάπτυξης όλο και πιο ασφαλών εφαρμογών. Σήμερα ο κλάδος της ασφάλειας διαδικτυακών εφαρμογών έχει αναπτυχθεί τόσο, που η πλειοψηφία των εφαρμογών του ξεφεύγει από τα πλαίσια της εργασίας. Σε αυτό το κεφάλαιο θα ασχοληθούμε με δύο μόνο λειτουργίες ασφαλείας, την ταυτοποίηση (Authentication) και την Εξουσιοδότηση (Authorization), και πώς μπορούμε να τις υλοποιήσουμε με την χρήση του Spring Security Module.

Το Spring Security είναι ένα από τα πολλά Projects της οικογένειας Spring. Σκοπός του είναι να προσφέρει τα κατάλληλα εργαλεία για την προσθήκη ασφάλειας σε εφαρμογές βασισμένες στο Spring Framework.

Ο τρόπος με τον οποίο το Spring Security προστατεύει τις διαδικτυακές μας εφαρμογές είναι φιλτράροντας τις διάφορες αιτήσεις HTTP, πριν αυτές φτάσουν στο αντίστοιχο Servlet για το οποίο προορίζονται. Μόλις λάβει μια αίτηση, βεβαιώνεται άν η ταυτότητα του χρήστη έχει επαληθευτεί, σε διαφορετική περίπτωση την επαληθεύει, έπειτα βεβαιώνεται πως ο συγκεκριμένος χρήστης είναι εξουσιοδοτημένος για να έχει πρόσβαση στην ηλεκτρονική διεύθυνση στην οποία εστάλη η αίτηση και αν είναι, τότε την προωθεί στο Servlet. Σε αντίθετη περίπτωση στέλνει HTTP απάντηση σφάλματος.

Όπως σε όλες τις περιπτώσεις των Module του Spring Framework, έτσι και το Configuration του Spring Security μπορεί να γίνει και με τους δύο τρόπους (XML, Java). Για λόγους ευκολίας, σε αυτό το μέρος του κεφαλαίου οι αναφορές στα διάφορα μέρη του Configuration θα γίνουν χρησιμοποιώντας κώδικα Java.

Η βάση για το Configuration των φίλτρων του Spring Security βρίσκεται σε μια κλάση που μας προσφέρει το Module, την WebSecurityConfigurerAdapter. Στην εφαρμογή μας πρέπει να δημιουργήσουμε μια κλάση - Configuration που θα την κάνει extend. Σε αυτήν την κλάση θα πρέπει να υλοποιήσουμε τις μεθόδους που κληρονομούμε από την WebSecurityConfigurerAdapter, στις οποίες θα ορίσουμε πού μπορεί το Spring Security να βρεί τους εξουσιοδοτημένους χρήστες, αν υπάρχει κάποια κωδικοποίηση (Encoding) του κωδικού πρόσβασης, ποιές ηλεκτρονικές διευθύνσεις θέλουμε να προστατεύσουμε καθώς και πολλές άλλες πληροφορίες για την διαδικασία με την οποία το Spring Security θα εκτελέσει την ταυτοποίηση. (LDAP, OAuth2, JWT κ.λπ.)

```java
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/", "/home").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
            .logout()
                .permitAll();
    }
```

Figure 5.2: Παράδειγμα κλάσης WebSecurityConfigurerAdapter

Η κλάση WebSecurityConfigurerAdapter διαθέτει δύο μεθόδους που ονομάζονται configure, καθεμία από τις οποίες όμως ρυθμίζει ένα διαφορετικό Bean. Η μέθοδος configure που φαίνεται στο παραπάνω παράδειγμα ρυθμίζει το αντικείμενο HttpSecurity. Σε αυτό το αντικείμενο περιέχονται οι ρυθμίσεις που αφορούν το ποιές ηλεκτρονικές διευθύνσεις της εφαρμογής μας θα προστατεύσουμε, καθώς και που βρίσκεται η σελίδα login, όπου ο χρήστης θα κληθεί να δώσει τα στοιχεία ταυτοποίησής του. Σημαντικές μέθοδοι που διαθέτει αυτό το αντικείμενο είναι οι παρακάτω:

- anyRequest() - Ορίζει πως οι ρυθμίσεις που ακολουθούν θα ισχύουν για όλες τις αιτήσεις.
- antMatchers(String pattern) - Ορίζει ένα μοτίβο ηλεκτρονικών διευθύνσεων για το οποίο θα ισχύουν οι ρυθμίσεις που ακολουθούν.
- loginPage(String url) - Ορίζει την ηλεκτρονική διεύθυνση στην οποία βρίσκεται η σελίδα Login.
- and() - Η μέθοδος αυτή ορίζει ένα σημείο κατάληξης των ρυθμίσεων για ένα μοτίβο ηλεκτρονικών διευθύνσεων ώστε να ξεκινήσουν ρυθμίσεις για κάποιο άλλο.
- authenticated() - Η μέθοδος αυτή ορίζει πως οποιαδήποτε διεύθυνση που ικανοποιεί το μοτίβο που περιγράφηκε πριν θα προστατεύεται.
- permitAll() - Η μέθοδος αυτή ορίζει πως οποιαδήποτε διεύθυνση που ικανοποιεί το μοτίβο που περιγράφηκε πριν θα είναι προσβάσιμη από όλους τους χρήστες.

- hasRole(String role) - Η μέθοδος αυτή ορίζει πως οποιαδήποτε διεύθυνση που ικανοποιεί το μοτίβο που περιγράφηκε πριν θα είναι προσβάσιμη από τους χρήστες που διαθέτουν τον αντίστοιχο ρόλο που ορίζει η παράμετρος.
- csrf() - Αυτή η μέθοδος επιστρέφει ένα αντικείμενο με το οποίο ρυθμίζουμε την αντιμετώπιση των Cross-site Request Forgery περιπτώσεων.
- logout() - Αυτή η μέθοδος επιστρέφει ένα αντικείμενο με το οποίο ρυθμίζουμε το σημείο στο οποίο ο χρήστης μπορεί να αποσυνδεθεί από την εφαρμογή.

Η δεύτερη μέθοδος configure ρυθμίζει το αντικείμενο AuthenticationManagerBuilder. Αυτό το αντικείμενο είναι υπεύθυνο για την δημιουργία ενός αντικειμένου AuthenticationManager, το οποίο ορίζει πώς θα γίνει η διαδικασία της ταυτοποίησης. Σημαντικές μέθοδοι αυτού του αντικειμένου είναι οι παρακάτω:

- inMemoryAuthentication() - Αυτή η μέθοδος ορίζει πως τα δεδομένα των εξουσιοδοτημένων χρηστών θα βρίσκονται στην μνήμη. Ιδανική για δοκιμές και μικρές εφαρμογές με ελάχιστο πλήθος χρηστών, αλλά πρέπει να αποφεύγεται σε production-ready εφαρμογές.
- jdbcAuthentication() - Αυτή η μέθοδος ορίζει πως τα δεδομένα των εξουσιοδοτημένων χρηστών θα βρίσκονται σε κάποια βάση δεδομένων.
- userDetailsService(T userDetailsService) - Αυτή η μέθοδος ορίζει πως τα δεδομένα των εξουσιοδοτημένων χρηστών θα παρέχονται στον AuthenticationManager μέσω του αντικειμένου της υπηρεσίας που δέχεται ώς παράμετρο.
- ldapAuthentication() - Αυτή η μέθοδος ορίζει πως τα δεδομένα των εξουσιοδοτημένων χρηστών θα βρίσκονται σε κάποιον LDAP Server.

Το Spring Security διαθέτει ένα Interface το οποίο ορίζει ένα μοτίβο για τα δεδομένα που κρατάμε για τον εξουσιοδοτημένο χρήστη της εφαρμογής. Αυτό το Interface ονομάζεται UserDetails. Με την χρήση αυτού του Interface μπορούμε να δημιουργήσουμε την δική μας Custom υπηρεσία ταυτοποίησης, που θα κάνει extend την αντίστοιχη UserDetailsService που προσφέρει το Framework.

Σε αυτό το μέρος του κεφαλαίου είδαμε μια γενική εικόνα των δυνατοτήτων προστασίας των εφαρμογών που μας προσφέρει το Spring Security Module, καθώς και μια εικόνα του τρόπου με τον οποίο γίνεται η ταυτοποίηση των χρηστών σε μια διαδικτυακή εφαρμογή.

Κεφάλαιο 6

Persistence - Η μόνιμη αποθήκευση δεδομένων

6.1 Εισαγωγή

Καθώς η διαχείριση δεδομένων είναι ένα αναπόσπαστο και βασικό στοιχείο της ανάπτυξης εφαρμογών, μεγάλη έμφαση έχει δοθεί στην ανάπτυξη συστημάτων διαχείρισης τα οποία προσφέρουν αξιοπιστία και ασφάλεια κατά την αποθήκευσή τους, καθώς και ευκολία και αποδοτικότητα κατά την εξόρυξη πληροφορίας από αυτά. Τα συστήματα αυτά ονομάζονται Συστήματα Διαχείρισης Βάσεων Δεδομένων (DataBase Management Systems). Οι βάσεις αυτές μπορεί είτε να ακολουθούν το σχεσιακό μοντέλο (Relational Databases) και να αποθηκεύουν τα δεδομένα τους σε πίνακες, είτε να μην το ακολουθούν, και να διαθέτουν δικό τους τρόπο αποθήκευσης και ανάκτησης των δεδομένων (NoSQL Databases). Στα πλαίσια της εργασίας θα ασχοληθούμε με σχεσιακές βάσεις δεδομένων.

Τα συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων διαθέτουν την δική τους γλώσσα, την οποία ο χρήστης χρησιμοποιεί για να επικοινωνήσει με το σύστημα, ώστε να εκτελέσει οποιαδήποτε λειτουργία στις βάσεις δεδομένων του. Αυτές οι λειτουργίες μπορεί να είναι είτε διαχείριση - τροποποίηση της δομής της βάσης δεδομένων (προσθήκη - κατάργηση - αλλαγή της μορφής πινάκων), είτε προσθήκη - διαγραφή - τροποποίηση δεδομένων, είτε ανάκτηση δεδομένων με βάση κάποια κριτήρια. Αυτή η γλώσσα ονομάζεται SQL (Structured Query Language).

Οι εφαρμογές μας, οι οποίες είναι γραμμένες στην γλώσσα Java, η οποία είναι αντικειμενοστραφής γλώσσα, αποθηκεύουν τα δεδομένα μας σε αντικείμενα. Το μοντέλο αποθήκευσης του αντικειμενοστραφούς μοντέλου είναι πολύ διαφορετικό από το μοντέλο αποθήκευσης μιας σχεσιακής βάσης δεδομένων. Για να χρησιμοποιήσουμε λοιπόν ένα σύστημα διαχείρισης σχεσιακής βάσης δεδομένων θα πρέπει να αντιστοιχίσουμε τα δεδομένα που βρίσκονται στα αντικείμενα μας, για να δημιουργήσουμε τις κατάλληλες εντολές της γλώσσας SQL, οι οποίες θα εκτελέσουν την επιθυμητή λειτουργία στα δεδομένα που βρίσκονται αποθηκευμένα στην βάση δεδομένων.

Η πλατφόρμα Java διαθέτει δύο τρόπους τους οποίους μπορούμε να χρησιμοποιήσουμε για να συνδέσουμε τις εφαρμογές μας με σχεσιακές βάσεις δεδομένων:

Ο πρώτος τρόπος ονομάζεται JDBC, το οποίο σημαίνει Java DataBase Connector. Πρόκειται για ένα API το οποίο μας προσφέρει την δυνατότητα να συνδέσουμε την εφαρμογή μας σε μια βάση δεδομένων είτε στο ίδιο μηχάνημα είτε απομακρυσμένη, μέσω του διαδικτύου, και να συντάσσουμε και να εκτελούμε εντολές SQL σε αυτήν. Στην πλατφόρμα Java EE υπάρχει το πρότυπο για αυτήν την τεχνολογία και η εταιρία η οποία εκδίδει την Java η Oracle έχει δημιουργήσει και διαθέτει μια υλοποίησή της.

Καθώς είπαμε πως το μοντέλο αποθήκευσης δεδομένων σε μια σχεσιακή βάση δεδομένων άρα και στις εντολές SQL με τις οποίες την διαχειριζόμαστε είναι διαφορετικό από το μοντέλο αποθήκευσης των δεδομένων στο αντικειμενοστραφές μοντέλο με το οποίο είναι σχεδιασμένη η εφαρμογή μας, η απαραίτητη αντιστοίχιση των δύο μοντέλων και μετατροπή των δεδομένων από το ένα στο άλλο κατά την δημιουργία των εντολών SQL από τα δεδομένα που είναι αποθηκευμένα στα αντικείμενα και κατά την αποθήκευση των δεδομένων που μας αποστέλλει η επεξεργασία αυτών των εντολών σε αντικείμενα, είναι ευθύνη του προγραμματιστή της εφαρμογής.

Για να ελευθερωθεί ο προγραμματιστής μιας εφαρμογής από αυτήν την ευθύνη, δημιουργήθηκε ένα πρότυπο της πλατφόρμας Java EE, το οποίο ονομάζεται JPA (Java Persistence API). Αυτό το πρότυπο περιγράφει τις προδιαγραφές για δημιουργία εργαλείων τα οποία αναλαμβάνουν μια διαδικασία που ονομάζεται ORM (Object - Relational Mapping). Χρησιμοποιώντας αυτά τα εργαλεία, ο προγραμματιστής της εφαρμογής ορίζει στις κλάσεις της εφαρμογής όπου θα αποθηκευτούν τα δεδομένα διάφορες πληροφορίες σχετικά με το τι θέλει να αποθηκευθεί στην βάση δεδομένων, καθώς και διάφορες πληροφορίες που αφορούν την δομή της βάσης δεδομένων και των σχέσεων μεταξύ διαφόρων πεδίων των πινάκων. Το εργαλείο έπειτα αναλαμβάνει να δημιουργήσει τους κατάλληλους πίνακες στη βάση δεδομένων.

Επίσης αυτά τα εργαλεία προσφέρουν μεθόδους με τις οποίες μπορούμε να διαχειριστούμε την βάση δεδομένων μας και είτε να ανακτήσουμε, είτε να τροποποιήσουμε τα δεδομένα της, χωρίς να χρειαστεί να γράψουμε εντολές SQL. Αυτές οι μέθοδοι δέχονται ώς παραμέτρους αντικείμενα, και αναλαμβάνουν αυτές την αντιστοίχιση. Επίσης επιστρέφοντας αποτελέσματα, παράγουν αυτόματα τα κατάλληλα αντικείμενα για να αποθηκεύσουν τα δεδομένα.

Στα πλαίσια της εργασίας θα ασχοληθούμε με δύο διαφορετικά εργαλεία ORM. Μέσα από αυτά θα δούμε πώς το πρότυπο JPA χρησιμοποιείται για να περιγράψει τις διάφορες πληροφορίες που χρειάζεται το εργαλείο για να δημιουργήσει την βάση δεδομένων στις κλάσεις μας, καθώς και το πώς έπειτα εμείς μπορούμε να χρησιμοποιήσουμε αυτά τα εργαλεία για να εκτελέσουμε διάφορες λειτουργίες στην βάση δεδομένων μας.

6.2 Hibernate - Το δημοφιλέστερο ORM Tool

Στον κόσμο της Java, η έννοια ORM Tool είναι σχεδόν ταυτόσημη με το Hibernate. Το ποσοστό χρήσης του έναντι άλλων αντιστοίχων ORM Tools το καθιστά de facto Standard εργαλείο για αυτήν την εργασία. Το Hibernate υλοποιεί το πρότυπο JPA για την αντιστοίχιση των δεδομένων από το αντικειμενοστραφές μοντέλο και αντίστροφα. Από την αρχή της ανάπτυξής του όμως, δίνει μια έμφαση στην προσθήκη δικών του λειτουργιών, όπου πολλές από αυτές με τον καιρό ενσωματώθηκαν και υιοθετήθηκαν από το JCP και προστέθηκαν στο πρότυπο JPA.

Για να λειτουργήσει το Hibernate χρειάζεται ένα Configuration, το οποίο ενώ αρχικά ήταν γραμμένο αποκλειστικά σε μορφή XML, σε ένα αρχείο που ονομάζεται hibernate.cfg.xml, σε επόμενες εκδόσεις δόθηκε η δυνατότητα να γίνεται και μέσω κώδικα Java. Χρησιμοποιώντας τις πληροφορίες που βρίσκονται σε αυτό το Configuration δημιουργούμε το αντικείμενο της κλάσης SessionFactory, το οποίο παράγει τις Sessions οι οποίες διαθέτουν τις μεθόδους για να εκτελέσουμε τις διάφορες λειτουργίες στη βάση μας. Σε αυτό το Configuration ορίζουμε την τιμή σε κάποιες ιδιότητες. Οι πιο σημαντικές από αυτές είναι οι παρακάτω:

- hibernate.connection.driver_class / javax.persistence.jdbc.driver - Σε αυτήν την ιδιότητα ορίζουμε την κλάση του JDBC Driver την οποία χρησιμοποιεί το Hibernate.
- hibernate.connection.url / javax.persistence.jdbc.url - Σε αυτήν την ιδιότητα ορίζουμε την ηλεκτρονική διεύθυνση στην οποία βρίσκεται η βάση δεδομένων μας.
- hibernate.connection.username / javax.persistence.jdbc.user Σε αυτήν την ιδιότητα ορίζουμε το όνομα χρήστη της βάσης δεδομένων.
- hibernate.connection.password / javax.persistence.jdbc.password Σε αυτήν την ιδιότητα ορίζουμε τον κωδικό χρήστη της βάσης δεδομένων.
- hibernate.show_sql - Σε αυτήν την ιδιότητα ορίζουμε αν το Hibernate θα εκτυπώνει στην κονσόλα τα SQL Queries που εκτελεί. Χρησιμοποιείται κυρίως για λόγους Debugging και Testing.
- hibernate.hbm2ddl.auto - Σε αυτήν την ιδιότητα ορίζουμε άν το Hibernate θα δημιουργεί από την αρχή την βάση δεδομένων κατά την αρχικοποίηση ή αν απλά θα προσθέτει τα νέα δεδομένα στα ήδη υπάρχοντα.

Η βασική λειτουργία του Hibernate όπως είπαμε είναι το ORM, η αντιστοίχιση δηλαδή των δεδομένων ανάμεσα στο αντικειμενοστραφές και το σχεσιακό μοντέλο. Αυτό κάθε εργαλείο που υλοποιεί το πρότυπο JPA το επιτυγχάνει ενσωματώνοντας στις κλάσεις που θα περιέχουν αυτά τα δεδομένα τις κατάλληλες πληροφορίες, είτε

μέσω αρχείων XML, είτε, όπως θα δούμε σε αυτήν την εργασία, χρησιμοποιώντας τα κατάλληλα Annotations.

Η κύρια έννοια του προτύπου JPA, χάρη στην οποία το εκάστοτε εργαλείο πραγματοποιεί την αντιστοίχιση είναι η οντότητα (Entity). Το Entity είναι μια κλάση η οποία αντιστοιχίζεται με έναν πίνακα. Κάποια από ή όλα τα πεδία αυτής της κλάσης αντιστοιχίζονται με τις στήλες του πίνακα, και σε αυτού του είδους τις κλάσεις περιέχονται όλες οι απαραίτητες πληροφορίες για τις σχέσεις μεταξύ στηλών των διαφόρων πινάκων.

Σε μια τυπική εφαρμογή στην οποία γίνεται χρήση του Hibernate, στις κλάσεις Entity συμπεριλαμβάνονται διάφορα Annotations του προτύπου JPA, τα οποία ορίζουν τις περισσότερες απαραίτητες πληροφορίες για το ORM. Πρόκειται για μια μακρά λίστα από Annotations, τα σημαντικότερα εκ των οποίων είναι τα παρακάτω:

- @Entity - Προσθέτοντας αυτό το Annotation στον ορισμό μιας κλάσης ορίζουμε πως η κλάση αυτή θα περιγράφει μια οντότητα.
- @Column - Προσθέτοντας αυτό το Annotation στον ορισμό ενός πεδίου μιας κλάσης Entity, ορίζουμε πως το πεδίο αυτό αντιστοιχίζεται σε μια στήλη του αντίστοιχου πίνακα.
- @Id - Προσθέτοντας αυτό το Annotation στον ορισμό ενός πεδίου μιας κλάσης Entity, ορίζουμε πως το πεδίο αυτό αποτελεί το πρωτεύον κλειδί του αντίστοιχου πίνακα.
- @GeneratedValue - Προσθέτοντας αυτό το Annotation στον ορισμό ενός πεδίου μιας κλάσης Entity, ορίζουμε πως οι τιμές αυτού του πεδίου θα παράγονται αυτόνομα, σύμφωνα με μια στρατηγική. Δέχεται σαν παράμετρο μια τιμή που ορίζει ποια στρατηγική θα είναι αυτή. Χαρακτηριστικό παράδειγμα είναι το Auto_increment.
- @OneToOne / @OneToMany / @ManyToOne / @ManyToMany - Προσθέτοντας αυτό το Annotation σε ένα πεδίο, η κλάση του οποίου αποτελεί Entity, ορίζει την αντίστοιχη σχέση μεταξύ των δύο πινάκων με βάση το σχεσιακό μοντέλο.
- @JoinColumn - Όμοίως με την προηγούμενη περίπτωση, χρησιμοποιώντας αυτό το Annotation ορίζεις το όνομα της στήλης στην οποία σχετίζονται οι αντίστοιχοι πίνακες, καθώς και ποίο είναι το ξένο κλειδί στον άλλο πίνακα.
- @Table - Προσθέτοντας αυτό το Annotation στον ορισμό μιας κλάσης Entity, μπορούμε να δώσουμε όνομα στον παραγόμενο πίνακα (Το default όνομα του παραγομένου πίνακα είναι το όνομα της κλάσης.)
- @Embeddable - Προσθέτοντας αυτό το Annotation στον ορισμό μιας κλάσης Entity, ορίζουμε πως για αυτήν την κλάση δεν θα δημιουργηθεί πίνακας, αλλά αυτή η κλάση μπορεί να συμπεριληφθεί σαν πεδίο μιας άλλης οντότητας, και τα αντίστοιχα πεδία της θα προστεθούν σαν στήλες εκείνου του πίνακα.

- @Embedded - Προσθέτοντας αυτό το Annotation στον ορισμό ενός πεδίου, η κλάση του οποίου έχει οριστεί ώς Embeddable, ορίζεις πως οι στήλες της κλάσης αυτού του πεδίου θα προστεθούν σαν στήλες του πίνακα.

- @Transient - Προσθέτοντας αυτό το Annotation στον ορισμό ενός πεδίου μιας κλάσης Entity, ορίζουμε πως αυτό το πεδίο δεν θα λαμβάνεται υπ΄ όψιν κατά την διαδικασία της αντιστοίχισης και δεν θα αποθηκεύεται στην βάση.

Εκτός από την διαδικασία της αντιστοίχισης του αντικειμενοστραφούς μοντέλου αποθήκευσης δεδομένων με το σχεσιακό μοντέλο, το Hibernate μας προσφέρει ένα σύνολο απο μεθόδους τις οποίες μπορούμε να χρησιμοποιήσουμε για να εκτελέσουμε διάφορες λειτουργίες πάνω στην βάση δεδομένων μας, χωρίς να χρησιμοποιήσουμε το JDBC API ή εντολές SQL.

Για να αποκτήσουμε πρόσβαση στις μεθόδους με τις οποίες μπορούμε να διαχειριστούμε δεδομένα π.χ. να αποθηκεύσουμε κάποιο αντικείμενο στην βάση μας, να διαγράψουμε κάποια εγγραφή, να ανακτήσουμε κάποια πληροφορία, θα πρέπει να δημιουργήσουμε κάποια αντικείμενα. Προηγουμένως είπαμε πως το Configuration του Hibernate χρησιμοποιείται για να δημιουργήσει ένα Singleton αντικείμενο, η κλάση του οποίου ονομάζεται SessionFactory. Αυτό το αντικείμενο διαθέτει μια μέθοδο που ονομάζεται createSession(). Αυτή η μέθοδος μας επιστρέφει ένα αντικείμενο της κλάσης Session, το οποίο είναι βασικό για την εκτέλεση των απαραίτητων λειτουργιών.

Εκτός από το αντικείμενο Session θα χρειαστούμε και ένα αντικείμενο της κλάσης Transaction. Αυτό το αντικείμενο, όπως λέει και το όνομά του, αντιπροσωπεύει μια συναλλαγή (Transaction). Οι Transactions είναι διαδικασίες οι οποίες περιέχουν έναν αριθμό από βήματα. Για να εκτελεστούν αυτά τα βήματα, η υλοποίηση αυτής της διαδικασίας απαιτεί να ολοκληρωθούν επιτυχώς όλα. Αν κάποιο από τα βήματα αποτύχει, τότε αποτυγχάνει ολόκληρο το Transaction και όποια βήματα είχαν ήδη εκτελεστεί ακυρώνονται μέσα από μια διαδικασία που ονομάζεται Rollback, φέρνοντας την διαδικασία στην κατάσταση που ήταν πρίν αρχίσει η εκτέλεση. Η τελική εκτέλεση όλων των βημάτων ονομάζεται Commit. Πρόσβαση στο αντικείμενο Transaction του Hibernate αποκτούμε μέσω της μεθόδου beginTransaction() του αντικείμενου Session.

Από την κλήση της μεθόδου beginTransaction() του αντικειμένου Session μέχρι την κλήση της μεθόδου commit() του αντικειμένου Transaction μπορούμε να προσθέσουμε τις διάφορες εντολές με τις οποίες διαχειριζόμαστε τα δεδομένα μας. Κάθε τέτοια εντολή αποτελεί και ένα βήμα στο εν λόγω Transaction. Οι κυριότερες από αυτές τις εντολές παρουσιάζονται παρακάτω:

- save(Object obj) - Δέχεται σαν παράμετρο ένα αντικείμενο μιας Entity κλάσης και το αποθηκεύει στον αντίστοιχο πίνακα.

- update(Object obj) - Δέχεται σαν παράμετρο ένα αντικείμενο μιας Entity κλάσης το οποίο είναι ήδη αποθηκευμένο στην βάση (έχει το ίδιο πρωτεύον κλειδί με

κάποια εγγραφή). Το αποθηκεύει στον αντίστοιχο πίνακα, αντικαθιστώντας τις τιμές που διαφέρουν

- saveOrUpdate(Object obj) - Δέχεται σαν παράμετρο ένα αντικείμενο μιας Entity κλάσης και είτε το αποθηκεύει σαν νέα εγγραφή, αν δέν υπάρχει ήδη, είτε αντικαθιστά τις τιμές που διαφέρουν, αν υπάρχει.

- delete(Object object) - Δέχεται σαν παράμετρο ένα αντικείμενο μιας Entity κλάσης και άν υπάρχει σαν εγγραφή στον αντίστοιχο πίνακα, την διαγράφει.

- get(Class<T>type, Serializable id) - Δέχεται σαν παραμέτρους μιά Entity κλάση και ένα πρωτεύον κλειδί και επιστρέφει σαν αντικείμενο την αντίστοιχη εγγραφή, αν αυτή υπάρχει.

Αυτές είναι οι βασικότερες μέθοδοι με τις οποίες εκτελούμε CRUD λειτουργίες στα δεδομένα της βάσης μας με την βοήθεια του Hibernate. Ενώ όμως στην περίπτωση της αποθήκευσης, τροποποίησης ή αλλαγής των δεδομένων της βάσης δεδομένων αυτές οι μέθοδοι φαίνονται αρκετές στις περισσότερες περιπτώσεις, η ανάκτηση των δεδομένων είναι μια πιο πολύπλοκη διαδικασία. Γι΄ αυτό το Hibernate μας προσφέρει διάφορους τρόπους με τους οποίους μπορούμε να συγκεκριμενοποιήσουμε τι είδους δεδομένα θέλουμε να ανακτήσουμε.

Ένας βασικός λόγος που οι σχεσιακές βάσεις δεδομένων έγιναν ένας τόσο δημοφιλής τρόπος μόνιμης αποθήκευσης δεδομένων είναι το γεγονός πως η γλώσσα SQL είναι μια ισχυρή γλώσσα όσον αφορά την ανάκτηση των δεδομένων από την βάση. Στην συζήτηση για το Hibernate και τα διάφορα ORM Tools παρουσιάσαμε σαν θετικό επιχείρημα το γεγονός πως αυτά τα εργαλεία μας δίνουν την δυνατότητα να εκτελέσουμε τις διάφορες λειτουργίες χωρίς την χρήση εντολών SQL. Αυτό δεν σημαίνει πως η χρήση των εντολών SQL δεν είναι δυνατή, ή σε κάποιες περιπτώσεις αναγκαία. Το Hibernate μας προσφέρει την δυνατότητα να εκτελέσουμε SQL Queries κατ΄ ευθείαν, χωρίς την ανάγκη της χρήσης του JDBC API κάνοντας χρήση της μεθόδου createSQLQuery(String query). Η μέθοδος αυτή δέχεται σαν παράμετρο ένα SQL Query και επιστρέφει ένα αντικείμενο της κλάσης NativeQuery, το οποίο διαθέτει μεθόδους για την εκτέλεσή του και την επεξεργασία των αποτελεσμάτων του.

Ένας διαφορετικός και πολύ δημοφιλής τρόπος ανάκτησης δεδομένων με την χρήση του Hibernate είναι η γλώσσα HQL (Hibernate Query Language). Σαν βασικό επιχείρημα κατά της χρήσης της γλώσσας SQL είναι το γεγονός πως τα SQL Queries είναι γραμμένα με βάση το σχεσιακό μοντέλο αποθήκευσης των δεδομένων, και η μετατροπή των δεδομένων από τα αντικείμενα σε αυτό το μοντέλο γίνεται ευθύνη του προγραμματιστή. Η γλώσσα HQL συνδυάζει τα πλεονεκτήματα των SQL Queries σαν ισχυρός τρόπος ανάκτησης δεδομένων και της γλώσσας προγραμματισμού της εφαρμογής, ενσωματώνοντας το αντικειμενοστραφές μοντέλο αποθήκευσης των δεδομένων σε Queries. Πιο συγκεκριμένα η γλώσσα HQL διαθέτει τις ίδιες εντολές με την γλώσσα SQL, αλλά στην θέση των πινάκων της βάσης, χρησιμοποιεί ονόματα αντικειμένων, και στην θέση των ονομάτων των στηλών του πίνακα χρησιμοποιεί τα ονόματα των αντιστοίχων πεδίων.

Τέλος, ο τρίτος τρόπος που μας προσφέρει το Hibernate για την ανάκτηση δεδομένων ονομάζεται Criteria Query. Σε αυτήν την περίπτωση, δημιουργούμε ένα αντικείμενο, στο οποίο καλούμε μια σειρά από μεθόδους με τις οποίες δίνουμε πληροφορίες για τα δεδομένα που θέλουμε να ανακτήσουμε. Στο τέλος χρησιμοποιώντας την μέθοδο του αντικειμένου Session createQuery() και δίνοντας σαν παράμετρο αυτό το αντικείμενο CriteriaQuery δημιουργείται αυτόματα το κατάλληλο Query. Έπειτα η μέθοδος getResultList() του παραγόμενου Query περιέχει τα αποτελέσματα της εκτέλεσής του.

Τέλος, όπως είδαμε το Hibernate υποστηρίζει τις σχέσεις μεταξύ πεδίων ενός πίνακα και αντίστοιχων πινάκων, όπως περιγράφονται στις σχέσεις 1 προς 1, 1 προς πολλά, πολλά προς 1 και πολλά προς πολλά του σχεσιακού μοντέλου. Όμως, εφ΄ όσον το αποτέλεσμα μιας διαδικασίας ανάκτησης δεδομένων αποθηκεύεται σε ένα αντικείμενο, και το αντικείμενο στις περιπτώσεις 1 προς πολλά και πολλά προς πολλά περιέχει μια λίστα με όλες τις εγγραφές του αντίστοιχου πεδίου, όταν εκτελέσουμε κάποια διαδικασία η οποία θα μας επιστρέψει αυτό το αντικείμενο, ή πολλά τέτοια αντικείμενα, θα πρέπει να εκτελεστούν και οι διαδικασίες για να επιστραφούν οι αντίστοιχες λίστες με τα αντικείμενα του αντίστοιχου πεδίου. Αυτό συμβαίνει άσχετα με το αν ο προγραμματιστής θέλει πρόσβαση σε αυτή τη λίστα ή αν απλά έκανε μια αναζήτηση με βάση ένα id με σκοπό π.χ. να μάθει ένα πεδίο τύπου String. Σε αυτήν την περίπτωση το Hibernate θα εκτελέσει έναν αριθμό από Queries χωρίς λόγο, και τα παραγόμενα αποτελέσματα θα μεταφερθούν στην μνήμη χωρίς λόγο. Για να αποφύγει αυτό το πρόβλημα το Hibernate διαθέτει ένα feature το οποίο ονομάζεται Fetch Strategy. Μπορούμε να ορίσουμε το Fetch Strategy το οποίο επιθυμούμε σαν παράμετρο στα διάφορα Annotations με τα οποία περιγράφουμε τις αντίστοιχες σχέσεις. Υπάρχουν δύο τύποι Fetch Strategy. Η μία είναι η Eager, κατά την οποία όταν δημιουργείται το εκάστοτε αντικείμενο, όλα τα αντικείμενα που περιέχονται ανακτώνται, και η δεύτερη είναι η Lazy, όπου τα αντίστοιχα αντικείμενα που περιέχονται ανακτώνται μόνο εάν υπάρξει κάποια αναφορά σε αυτά σε μεταγενέστερο σημείο της εφαρμογής.

6.3 EclipseLink - Μία δημοφιλής εναλλακτική

Το Hibernate, ενώ στην γενική περίπτωση αποτελεί μια υλοποίηση του προτύπου JPA, αν μελετήσει κανείς το πρότυπο καλύτερα, θα ανακαλύψει πως υπάρχουν αρκετά πράγματα τα οποία είναι υλοποιημένα με διαφορετικό τρόπο. Αυτό συμβαίνει κατά κύριο λόγο επειδή οι ανάγκες της αγοράς της ανάπτυξης λογισμικού ορισμένες φορές δημιουργούνται με πιο γρήγορο ρυθμό από ότι το JCP μπορεί να ανταποκριθεί. Η οργάνωση που αναπτύσσει το Hibernate σε αρκετές περιπτώσεις προσθέτει features ανάλογα με τις ανάγκες που υπάρχουν και τις νέες δυνατότητες που προσφέρει η πλατφόρμα Java. Σε αρκετές περιπτώσεις αυτά τα features είναι τόσο καινοτόμα που προστίθενται στο πρότυπο JPA στις επόμενες εκδόσεις του.

Το επόμενο ORM Tool που θα παρουσιάσουμε, αντίθετα με το Hibernate, αποτελεί μια σχετικά πιστή υλοποίηση του προτύπου JPA. Ονομάζεται EclipseLink και υπεύθυνος για την ανάπτυξή του είναι το Eclipse Foundation, η οργάνωση που δημιούργησε το Eclipse IDE.

Στο πρότυπο JPA, το Configuration ορίζεται σε ένα XML αρχείο, το οποίο ονομάζεται persistence.xml. Σε αντίθεση με το Hibernate, όπου είδαμε πως το Configuration γίνεται διαφορετικά, στο EclipseLink η ύπαρξη αυτού του αρχείου είναι απαραίτητη. Οι πληροφορίες που υπάρχουν σε αυτό το αρχείο χρησιμοποιούνται για να δημιουργήσουμε ένα Singleton αντικείμενο της κλάσης EntityManagerFactory, το οποίο είναι αντίστοιχο με το αντικείμενο SessionFactory που δημιουργήσαμε στο Hibernate. Κατά την παρουσίαση του Hibernate είδαμε κάποιες από τις σημαντικότερες ιδιότητες που ορίζουμε στο Configuration. Οι ιδιότητες που έχουν σαν όνομα hibernate.* είναι ιδιότητες της υλοποίησης του Hibernate, δεν υπάρχουν στο πρότυπο JPA, και δεν υπάρχουν στο EclipseLink. Οι ιδιότητες όμως που είχαν σαν όνομα javax.persistence.jdbc.* ανήκουν στο πρότυπο και υπάρχουν κανονικά. Πρόκειται για τις javax.persistence.jdbc.driver, javax.persistence.jdbc.url, javax.persistence.jdbc.user, javax.persistence.jdbc.password.

Όπως το Hibernate, έτσι και το EclipseLink ακολουθεί το πρότυπο JPA πιστά για να περιγράψει τις κλάσεις Entity, ώστε να είναι δυνατή η διαδικασία του ORM. Έτσι τα Annotations που περιγράψαμε στο μέρος του κεφαλαίου για το Hibernate είναι τα ίδια με αυτά που χρησιμοποιούνται σε μια εφαρμογή που χρησιμοποιεί το EclipseLink.

Για να εκτελέσουμε μεθόδους για την διαχείριση των δεδομένων μας, στο EclipseLink θα χρειαστεί να δημιουργήσουμε τρία αντικείμενα, ομοίως με το Hibernate. Η διαφορά είναι πως το Hibernate διαθέτει υλοποιήσεις αυτών των αντικειμένων, οι οποίες διαφέρουν από το πρότυπο JPA, το οποίο το EclipseLink ακολουθεί. Οπότε τα αντίστοιχα αντικείμενα στο EclipseLink ονομάζονται EntityManagerFactory - (αντίστοιχο με το SessionFactory), EntityManager - (αντίστοιχο με το Session) και EntityTransaction - (αντίστοιχο με το Transaction).

Οι μέθοδοι διαχείρισης των δεδομένων έχουν υλοποιηθεί διαφορετικά στο Hibernate. Οι αντίστοιχες μέθοδοι του προτύπου, είναι οι παρακάτω:

- persist(Object obj) - Δέχεται σαν παράμετρο ένα αντικείμενο μιας Entity κλάσης και το αποθηκεύει στον αντίστοιχο πίνακα. Είναι αντίστοιχη με την μέθοδο save του Hibernate.
- refresh(Object obj) - Δέχεται σαν παράμετρο ένα αντικείμενο μιας Entity κλάσης το οποίο είναι ήδη αποθηκευμένο στην βάση (έχει το ίδιο πρωτεύον κλειδί με κάποια εγγραφή). Το αποθηκεύει στον αντίστοιχο πίνακα, αντικαθιστώντας τις τιμές που διαφέρουν. Είναι αντίστοιχη με την μέθοδο update του Hibernate.
- remove(Object object) - Δέχεται σαν παράμετρο ένα αντικείμενο μιας Entity κλάσης και άν υπάρχει σαν εγγραφή στον αντίστοιχο πίνακα, την διαγράφει. Είναι αντίστοιχη με την μέθοδο delete του Hibernate.
- find(Class<T>type, Object primaryKey) - Δέχεται σαν παραμέτρους μιά Entity κλάση και ένα πρωτεύον κλειδί και επιστρέφει σαν αντικείμενο την αντίστοιχη εγγραφή, αν αυτή υπάρχει. Είναι αντίστοιχη με την μέθοδο get του Hibernate.

Χρησιμοποιώντας το EclipseLink μπορούμε να χρησιμοποιήσουμε και τους τρείς τρόπους ανάκτησης δεδομένων, τους οποίους αναφέραμε στο προηγούμενο μέρος. Πιο συγκεκριμένα, η κλάση EntityManager διαθέτει την μέθοδο createNativeQuery(String query) με την οποία μπορούμε να εκτελέσουμε SQL Queries. Επίσης τα Criteria Queries είναι υλοποιημένα και στα δύο εργαλεία με τον ίδιο τρόπο, σύμφωνα με το πρότυπο, οπότε ισχύουν όσα είπαμε στο προηγούμενο μέρος. Τέλος, η γλώσσα HQL, που αναφέραμε στο προηγούμενο μέρος, είναι ένα υπερσύνολο μιας Querying γλώσσας που ορίζεται στο πρότυπο JPA και ονομάζεται JPQL (Java Persistence Query Language). Κάνοντας χρήση του EclipseLink μπορούμε να εκτελέσουμε Queries γραμμένα σε αυτή τη γλώσσα.

Τέλος, η ίδια υλοποίηση ακολουθείται στα Fetch Strategies και στα δύο εργαλεία, όπως περιγράφεται στο πρότυπο JPA.

6.4 Συμπεράσματα από την χρήση των διαφόρων εργαλείων

Τα δύο αυτά ORM Tools αποτελούν υλοποιήσεις του ίδιου προτύπου, οπότε σε γενικές γραμμές ακολουθούν την ίδια λογική και έχουν παρόμοια features. Ο λόγος που το Hibernate υπερτερεί σε δημοφιλία με συντριπτική διαφορά είναι πως, σε αντίθεση με το EclipseLink, δίνει έμφαση στην ανάπτυξη και εξέλιξη του ORM Tool μέρους του, ενώ το EclipseLink απλά ακολουθεί τις νέες εκδόσεις του προτύπου JPA. Για τον λόγο αυτό διαθέτει και ένα πιο πλούσιο σύνολο από features. Και οι δύο οργανώσεις οι οποίες τα αναπτύσσουν, αναπτύσσουν παράλληλα και άλλα Projects σχετικά με διαχείριση δεδομένων.

Το Hibernate, όπως είπαμε σε αρκετές περιπτώσεις ξεφεύγει αρκετά από το πρότυπο JPA. Στην περίπτωση που επιθυμούμε η εφαρμογή μας να συμμορφώνεται με τα πρότυπα της πλατφόρμας Java Enterprise Edition, η χρήση του δεν ενδείκνυται.

Τέλος, λόγω της δημοφιλίας του Hibernate σαν ORM Tool, υπάρχει μια πολύ μεγαλύτερη κοινότητα η οποία ασχολείται με αυτό, οπότε ο προγραμματιστής που θέλει να το χρησιμοποιήσει στην ανάπτυξη της εφαρμογής του έχει την δυνατότητα να αντλήσει πληροφορίες από μια μεγαλύτερη βάση.

Κεφάλαιο 7

Οι σημαντικότερες Templating Libraries

7.1 Εισαγωγή

Σε προηγούμενο κεφάλαιο είδαμε την αρχική προσέγγιση της συγγραφής του Front-end μέρους μιας διαδικτυακής εφαρμογής, την τεχνολογία JSP. Η χρήση αυτής της τεχνολογίας βοήθησε πολύ στον διαχωρισμό του οπτικού σχεδιασμού και της υλοποίησης της λειτουργικότητας μιας διαδικτυακής εφαρμογής γραμμένης στην γλώσσα Java, και είναι δημοφιλής ακόμη και σήμερα. Έχει όμως κάποια μειονεκτήματα. Πιο συγκεκριμένα:

- Ένα αρχείο JSP, ενώ κατα βάση περιέχει κώδικα HTML, δεν μπορεί να παρουσιαστεί από έναν φυλλομετρητή ώς έχει. Θα πρέπει να εκτελεστεί από έναν JSP Container, αφού αυτός λάβει την κατάλληλη αίτηση HTTP. Ο προγραμματιστής που είναι υπεύθυνος για τον οπτικό σχεδιασμό, θα πρέπει να έχει πρόσβαση σε αυτό το εργαλείο.

- Ενώ οι βιβλιοθήκες της JSP αυτοματοποιούν μεγάλο μέρος της λειτουργικότητας, οι εντολές τους αποτελούνται από custom JSP tags. Αυτό σημαίνει πως για να μετατρέψεις ένα υπάρχον αρχείο HTML, το οποίο έχει δημιουργήσει ένας Web Designer, σε λειτουργικό αρχείο JSP απαιτεί την αντικατάσταση των συμβατικών HTML tags, με τα κατάλληλα JSP tags. Αυτό απαιτεί περισσότερη εργασία.

- Η λειτουργία των JSPs είναι άρρηκτα συνδεδεμένη με τα Servlets. Αυτό σημαίνει πως η τεχνολογία μπορεί να χρησιμοποιηθεί μόνο για την επεξεργασία αιτήσεων HTTP, και όχι για άλλες εργασίες παραγωγής κειμένου (δημιουργία email κ.λπ.)

Για να αντιμετωπιστούν μερικά από αυτά τα μειονεκτήματα, δημιουργήθηκαν οι βιβλιοθήκες προτύπων (Template Libraries). Οι βιβλιοθήκες αυτές κυρίως χρησιμοποιούνται για την ανάπτυξη του View μέρους μιας MVC εφαρμογής, αλλά μπορούν να χρησιμοποιηθούν για οποιαδήποτε λειτουργικότητα της εφαρμογής απαιτεί την παραγωγή κειμένου.

7.2 Η βιβλιοθήκη προτύπων Thymeleaf

Μία πολύ δημοφιλής και χρήσιμη βιβλιοθήκη προτύπων είναι η Thymeleaf. Η βιβλιοθήκη αυτή διαθέτει όλες τις εντολές που χρειαζόμαστε για να δημιουργήσουμε δυναμικές ιστοσελίδες, τις οποίες ενσωματώνουμε στα στοιχεία HTML του αρχείου μας σαν attributes. Κάθε τέτοιο attribute έχει την μορφή: th:command="value".

Σε ένα παραδοσιακό Web Application, χωρίς την χρήση του Spring Framework, το κάθε Servlet θα πρέπει να καλεί ένα αντικείμενο της κλάσης org.thymeleaf.TemplateEngine, το οποίο θα επεξεργάζεται το αντίστοιχο αρχείο που περιέχει τις εντολές της Thymeleaf. Το Servlet έπειτα θα εκτυπώνει το αποτέλεσμα, για κάθε αίτηση HTTP.

Το Spring Web MVC Framework ωστόσο, όπως είπαμε. διαθέτει έναν μηχανισμό που ονομάζεται View Resolver, με την χρήση του οποίου το κατάλληλο View εκτελείται και τα περιεχόμενα του επιστρέφονται στην απάντηση HTTP. Η βιβλιοθήκη Thymeleaf κάνει χρήση μιας δικιάς της υλοποίησης του View Resolver την κλάση org.thymeleaf.spring4.view.ThymeleafViewResolver. Η αυτοματοποίηση της διαδικασίας που επιτυγχάνεται με αυτόν τον τρόπο καθιστά την βιβλιοθήκη μια εξαιρετική επιλογή για την ανάπτυξη του View tier μιας εφαρμογής Spring.

Το dispatcher-servlet του Spring Web MVC αναλαμβάνει να παραδώσει τις απαραίτητες μεταβλητές με την χρήση αντικειμένων της κλάσης Model, στα οποία είναι αποθηκευμένα τα δεδομένα που χρειαζόμαστε για να δημιουργήσουμε το δυναμικό περιεχόμενο, στο View Resolver της Thymeleaf. Με αυτόν τον τρόπο οι μεταβλητές αυτές είναι διαθέσιμες για χρήση στο εκάστοτε αρχείο.

Στην Thymeleaf για να αποκτήσουμε πρόσβαση στις μεταβλητές του μοντέλου που παραλάβαμε χρησιμοποιούμε την έκφραση ${...}. Στην έκφραση αυτή μπορούμε να χρησιμοποιήσουμε τα ονόματα των μεταβλητών που περιέχονται στο μοντέλο το οποίο έχει παραδοθεί στο View. π.χ.

```
1 <p th:text="\${myVariable}"> </p>
```

Εκτός των μεταβλητών του μοντέλου μας, στην έκφραση μεταβλητής μπορούν να χρησιμοποιηθούν αντικείμενα που μας παρέχει η Thymeleaf. Για να έχουμε πρόσβαση σε αυτά τα αντικείμενα, αρκεί να χρησιμοποιήσουμε το σύμβολο της δίεσης πρίν το όνομα του κάθε αντικειμένου. Αυτά τα αντικείμενα είναι τα:

- #ctx - Το αντικείμενο του Context
- #var - Οι μεταβλητές του Context
- #locale - Το αντικείμενο Locale του Context

- #httpServletRequest - Το αντικείμενο HttpServletRequest
- #httpSession - Το αντικείμενο HttpSession

Η Thymeleaf μας προσφέρει επίσης αντικείμενα με χρήσιμες μεθόδους, όπως τα:

- #dates - Αντικείμενο με μεθόδους επεξεργασίας ημερομηνιών.
- #numbers - Αντικείμενο με μεθόδους επεξεργασίας αριθμών.
- #strings - Αντικείμενο με μεθόδους επεξεργασίας συμβολοσειρών.
- #objects - Αντικείμενο με μεθόδους διαχείρισης αντικειμένων.
- #bools - Αντικείμενο με μεθόδους διαχείρισης λογικών εκφράσεων.
- #arrays - Αντικείμενο με μεθόδους διαχείρισης πινάκων.
- #lists - Αντικείμενο με μεθόδους διαχείρισης λιστών.
- #maps - Αντικείμενο με μεθόδους διαχείρισης χαρτών.
- #messages - Αντικείμενο με μεθόδους διαχείρισης των μηνυμάτων κειμένου.

Υπάρχει η δυνατότητα να επιλέξουμε ένα αντικείμενο σε ένα γονικό στοιχείο HTML, και έπειτα να έχουμε πρόσβαση στις ιδιότητες και στις μεθόδους του με την έκφραση *{...}. Αυτή η έκφραση αντί να ψάχνει για την μεταβλητή σε όλα τα αντικείμενα που έχουμε στη διάθεσή μας την ψάχνει στο αντικείμενο που είναι επιλεγμένο σε αυτήν την ομάδα στοιχείων. π.χ.

```
1  <div data-th-object="${session.user}">
2    <p>Name: <span data-th-text="*{firstName}">
3    Sebastian</span>.</p>
4    <p>Surname: <span data-th-text="*{lastName}">
5    Pepper</span>.</p>
6    <p>Nationality: <span data-th-text="*{nationality}">
7    Saturn</span>.</p>
8  </div>
```

Σε αυτό το παράδειγμα, στην ομάδα στοιχείων που περιέχεται στο div μας στο οποίο έχουμε επιλέξει ώς αντικείμενο μας το session.user η έκφραση *{firstName} ισοδυναμεί με την έκφραση ${session.user.firstName}. Σε περίπτωση που δεν έχει επιλεχθεί αντικείμενο οι εκφράσεις ${session.user.firstName} και *{session.user.firstName} είναι ισοδύναμες.

Η Thymeleaf έχει την δυνατότητα αναπαράστασης URL. Αυτό το κάνει με την χρήση της έκφρασης @{...}.

Υπάρχουν δυο διαφορετικοί τύποι URL. Το απόλυτο URL (http://www.myserver.com), και το σχετικό URL (/anotherFolder/myFile.html). π.χ.

Απόλυτο URL:

```
1 <a data-th-href="@{http://www.myserver.com}"> </a>
```

Σχετικό URL:

```
1 <a data-th-href="@{/anotherFolder/myFile.html}"> </a>
```

Στις εκφράσεις της Thymeleaf μπορούμε να εισάγουμε κείμενο, αριθμούς, μπορούμε να κάνουμε πράξεις με αριθμούς. π.χ.

```
1 <p>In two years, the year will be <span
2 data-th-text="2018 + 2">1494</span>.</p>
```

Μπορούμε επίσης να εισάγουμε τις λογικές εκφράσεις true και false. Καθώς και το null.
Ο τελεστής ένωσης (Concatenation) είναι όμοιος με αυτόν της γλώσσας Java και είναι ο +.
Υπάρχει η δυνατότητα να γράψουμε μια έκφραση η οποία συνδυάζει μεταβλητές και κείμενο. Αυτή η έκφραση πρέπει να ξεκινάει και να τελειώνει με κάθετη μπάρα π.χ.

```
1 <span data-th-text="|Welcome to our application,
2 ${user.name}!|">
```

Οι τελεστές αριθμητικών πράξεων είναι όμοιοι με αυτούς της γλώσσας Java και είναι οι +, -, *, / και %.
Οι τελεστές σύγκρισης είναι διαφορετικοί επειδή τα σύμβολα < και > είναι σύμβολα της γλώσσας HTML. Στη θέση τους χρησιμοποιούνται τα αντίστοιχα HTML Entities (> < ≥ ≤).
Οι τελεστές ισότητας και ανισότητας καθώς και ο τελεστής not είναι οι ίδιοι όπως και στην γλώσσα Java. (== , != , !).
Αντίστοιχα με την γλώσσα Java λειτουργεί και ο τελεστής ? : π.χ.

```
1 <tr data-th-class="${row.even}? 'even' : 'odd'">
2   ...
3 </tr>
```

Σε αυτό το παράδειγμα αν η σειρά είναι περιττή η μέθοδος εεν θα επιστρέψει true και η τιμή του class θα γίνει even. Σε αντίθετη περίπτωση η μέθοδος θα επιστρέψει false και η τιμή του class θα γίνει odd.

Η Thymeleaf χρησιμοποιεί τα ειδικά HTML attributes data-th-* για να αλλάξει την τιμή στα αντίστοιχα attributes των στοιχείων HTML του View.
Υπάρχουν πολλά διαφορετικά attributes τα οποία μπορούν να αλλάξουν με αυτόν τον τρόπο. Αυτά είναι τα:

abbr, accesskey, alt, autocomplete, accept, action, archive, axis, accept-charset, align, audio, background, bgcolor, border, cellpadding, cellspacing, challenge, charset, cite, class, classid, codebase, codetype, cols, colspan, compact, content, contenteditable, contextmenu, data, datetime, dir, draggable, dropzone, enctype, for, form, formaction, formenctype, formmethod, formtarget, frame, frameborder, headers, height, high, href, hreflang, hspace, http-equiv, icon, id, keytype, kind, label, lang, list, longdesc, low, manifest, marginheight, marginwidth, max, maxlength, media, method, min, name, optimum, pattern, placeholder, poster, preload, radiogroup, rel, rev, rows, rowspan, rules, sandbox, scheme, scope, scrolling, size, sizes, span, spellcheck, src, srclang, standby, start, step, style, summary, tabindex, target, title, type, usemap, value, valuetype, vspace, width, wrap, xmlbase, xmllang, xmlspace

Μια πάρα πολύ χρήσιμη δυνατότητα είναι αυτή της επανάληψης διαφόρων στοιχείων. Αν έχουμε δηλαδή π.χ. έναν πίνακα από δεδομένα και θέλουμε να τον εκτυπώσουμε σε έναν πίνακα HTML. Αυτό στην Thymeleaf γίνεται με την εντολή data-th-each. π.χ.

```
1    <table>
2      <tr>
3        <th>NAME</th>
4        <th>PRICE</th>
5        <th>IN STOCK</th>
6      </tr>
7      <tr data-th-each="prod : ${prods}">
8        <td data-th-text="${prod.name}">Onions</td>
9        <td data-th-text="${prod.price}">2.41</td>
10     </tr>
11   </table>
```

Σε αυτό το παράδειγμα η εντολή μας λέει πως για κάθε εγγραφή του πίνακα prods θα επαναλαμβάνεται η εκτύπωση του στοιχείου tr και των υποστοιχείων του. Επίσης σημαίνει πως το κάθε ένα στοιχείο του τρέχοντος στιγμιοτύπου της επανάληψης θα ορίζεται ως prod.
Σε αυτήν την εντολή μπορούν να χρησιμοποιηθούν όλα τα αντικείμενα που κάνουν Implement τα Interfaces Iterable ή Map της Java, καθώς και όλοι οι πίνακες.

Επίσης με την χρήση ενός αντικειμένου που ορίζεται μετά το στοιχείο του τρέχοντος στιγμιοτύπου στην εντολή μας, χωριζόμενο με κόμμα, μπορούμε να κρατήσουμε τα

εξής στοιχεία σχετικά με την επανάληψη:

Τον αριθμό του τρέχοντος στιγμιοτύπου της επανάληψης, ξεκινώντας από την τιμή 0, με το όνομα index.
Τον αριθμό του τρέχοντος στιγμιοτύπου της επανάληψης, ξεκινώντας από την τιμή 1, με το όνομα count.
Τον συνολικό αριθμό των επαναλήψεων, με το όνομα size.
Το στοιχείο του πίνακα/λίστας του τρέχοντος στιγμιοτύπου της επανάληψης, με το όνομα current.
Αν ο αριθμός της τρέχουσας επανάληψης είναι άρτιος ή περιττός αριθμός, με τις λογικές μεταβλητές με όνομα even και odd.
Αν βρισκόμαστε στην πρώτη ή στην τελευταία επανάληψη με τις λογικές μεταβλητές με όνομα first/last.

Μια επίσης απαραίτητη δυνατότητα είναι η εκτύπωση στοιχείων ή όχι με βάση κάποια συνθήκη. Αυτό γίνεται με την εντολή data-th-if, στην οποία δίνουμε σαν τιμή μια λογική συνθήκη. π.χ.

```
1     <div data-th-if="${array!=null}">
2         <tr data-th-each="elem : ${array}">
3             <td data-th-text="${elem.value}"></td>
4         </tr>
5     </div>
```

Σε αυτό το παράδειγμα θέλουμε να δημιουργηθεί το div στο οποίο θα εκτυπωθεί ο πίνακάς μας μόνο αν αυτός υπάρχει. Σε περίπτωση που δεν υπάρχει, το div δεν θα δημιουργηθεί.
Υπάρχει και η εντολή data-th-unless η οποία, όπως λέει και το όνομά της, λειτουργεί αντίστροφα.

Υπάρχει επίσης η εντολή data-th-switch η οποία έχει όμοια λειτουργία με την δομή switch της Java. π.χ.

```
1 <div data-th-switch="${user.role}">
2     <p data-th-case="'admin'">User is an administrator</p>
3     <p data-th-case="'manager'">User is a manager</p>
4 </div>
```

Σε πολλές περιπτώσεις, ιδίως όταν γίνεται επαναχρησιμοποίηση του ίδιου κώδικα, θέλουμε έναν τρόπο να αποθηκεύουμε τον κοινό κώδικα σε ένα σημείο και να τον καλούμε στα διάφορα μέρη, όπου χρειάζεται. Αυτό στην Thymeleaf γίνεται με την βοήθεια των fragments.
Σε ένα οποιοδήποτε αρχείο μπορούμε να ορίσουμε ένα fragment δημιουργώντας ένα

γονικό tag και προσθέτοντας ώς attribute το data-th-fragment=”fragmentName”.

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

  <body>

    <div data-th-fragment="copy">
      &copy; MyFooterText
    </div>

  </body>

</html>
```

Έπειτα, το fragment μας μπορεί να καλεστεί με την βοήθεια μιας εκ τριών διαφορετικών εντολών. Αυτές είναι οι:

α) data-th-insert=” templateName :: fragmentName” όπου templateName το όνομα του αρχείου στο οποίο βρίσκεται το fragment και fragmentName το όνομα του.
Αν αυτή η εντολή εφαρμοστεί π.χ. σε ένα div tag, θα προσθέσει τον κώδικα του fragment μέσα σε αυτό το div tag.
β) data-th-replace=” templateName :: fragmentName” Αν αυτή η εντολή εφαρμοστεί π.χ. σε ένα div tag, θα αντικαταστήσει αυτό το tag με τον κώδικα του φραγμεντ.
γ) data-th-include=” templateName :: fragmentName” Αν αυτή η εντολή εφαρμοστεί π.χ. σε ένα div tag, θα προσθέσει μόνο τα περιεχόμενα του γονικού tag του fragment μέσα σε αυτό το div tag.

Αν θεωρήσουμε πως το fragment μας είναι αυτό:

```
<footer data-th-fragment="copy">
  &copy; MyFooterText
</footer>
```

Αυτός ο κώδικας:

```
<body>

  <div th:insert="footer :: copy"></div>

  <div th:replace="footer :: copy"></div>
```

```
6
7   <div th:include="footer :: copy"></div>
8
9 </body>
```

θα παράξει αυτό το αποτέλεσμα:

```
1 <body>
2
3   <div>
4     <footer>
5       &copy; MyFooterText
6     </footer>
7   </div>
8
9   <footer>
10    &copy; MyFooterText
11  </footer>
12
13  <div>
14    &copy; MyFooterText
15  </div>
16
17 </body>
```

Υπάρχει η δυνατότητα αποστολής παραμέτρων στην κλήση ενός fragment, σαν να καλούσαμε μια συνάρτηση π.χ.

```
1 <div data-th-fragment="frag (onevar,twovar)">
2     <p data-th-text="${onevar} + ' - ' + ${twovar}">...</p>
3 </div>
```

Η κλήση του fragment θα πρέπει να είναι κάπως έτσι:

```
1 <div th:replace="::frag (${value1},${value2})">...</div>
2 <div th:replace="::frag (onevar=${value1},
3 twovar=${value2})">...</div>
```

7.3 Η βιβλιοθήκη προτύπων Apache Freemarker

Μια εναλλακτική βιβλιοθήκη προτύπων με την οποία μπορούμε να δημιουργήσουμε το View κομμάτι μιας Web εφαρμογής είναι η Apache Freemarker. Η βιβλιοθήκη διαθέτει τις αντίστοιχες εντολές για την δημιουργία δυναμικών ιστοσελίδων, τις οποίες ενσωματώνει στον κώδικα HTML με την μορφή tags.

Για να χρησιμοποιήσουμε την βιβλιοθήκη σε μια παραδοσιακή διαδικτυακή εφαρμογή Java, χρειαζόμαστε 3 αντικείμενα. Ένα αντικείμενο της κλάσης freemarker.template.Configuration, στο οποίο θα ορίζονται οι απαραίτητες ρυθμίσεις, ένα αντικείμενο το οποίο θα περιέχει το μοντέλο μας, δηλαδή τις μεταβλητές που χρησιμοποιούμε κατά την δημιουργία του περιεχομένου του προτύπου, και ένα αντικείμενο της κλάσης freemarker.template.Template, το οποίο δημιουργούμε καλώντας την μέθοδο getTemplate, με παράμετρο το όνομα του αρχείου του προτύπου. Το αντικείμενο της κλάσης Template έχει μια μέθοδο, την process, η οποία παίρνει ώς παραμέτρους το αντικείμενο που περιέχει το μοντέλο και ένα αντικείμενο Writer, το οποίο στην περίπτωση μας είναι ο Writer της απάντησης HTTP.

Το Spring Web MVC Framework, όπως και στην περίπτωση της Thymeleaf, διαθέτει την δική του υλοποίηση ενός View Resolver για την βιβλιοθήκη, καθώς και μια κλάση η οποία είναι υπεύθυνη για τον ορισμό των καταλλήλων ρυθμίσεων.

Αντίστοιχα με την βιβλιοθήκη Thymeleaf, η Apache Freemarker διαθέτει την έκφραση ${...}, στην οποία μπορούμε να τοποθετήσουμε οποιαδήποτε έκφραση, η οποία θα υπολογιστεί και θα εκτυπωθεί.

Αντί για αντικείμενα τα οποία διαθέτουν μεθόδους με τις οποίες επεξεργαζόμαστε διάφορους τύπους μεταβλητών, η βιβλιοθήκη μας διαθέτει μεθόδους - φίλτρα τις οποίες καλούμε με τον τελεστή ? ακολουθούμενο από το όνομα του φίλτρου. Τα διάφορα φίλτρα και η λειτουργικότητά τους αναλύονται στο documentation της βιβλιοθήκης (https://freemarker.apache.org/docs/ref_builtins_alphaidx.html)

Ομοίως με την βιβλιοθήκη Thymeleaf, οι τελεστές αριθμητικών πράξεων είναι όμοιοι με αυτούς της γλώσσας Java και είναι οι +, -, *, / και %.
Οι τελεστές σύγκρισης είναι διαφορετικοί επειδή τα σύμβολα < και > είναι σύμβολα της γλώσσας HTML. Στη θέση τους χρησιμοποιούνται οι εκφράσεις gt, lt, gte, lte.
Οι τελεστές ισότητας και ανισότητας καθώς και ο τελεστής not είναι οι ίδιοι όπως και στην γλώσσα Java. (== , != , !).

Στην Apache Freemarker, οι εντολή με την οποία γίνεται η επανάληψη είναι η <#list> π.χ.

```
<ul>
<#list users as user>
    <li>${user}</li>
</#list>
</ul>
```

Η αντίστοιχη εντολή με την οποία μπορούμε να εκτυπώσουμε στοιχεία με βάση κάποια συνθήκη είναι η <#if> και οι αντίστοιχες <#elseif>, <#else>, </#if> π.χ.

```
<#if x==5>
    <p>The number is 5</p>
<#elseif x==3>
    <p>The number is 3</p>
<#else>
    <p> The number is neither 5 nor 3</p>
</#if>
```

Υπάρχει και η δομή switch με τις αντίστοιχες εντολές <#switch>, <#case>, <#break>, <#default>, </#switch> π.χ.

```
<#switch x>
    <#case 5>
        <p>The number is 5</p>
        <#break>
    <#case 3>
        <p>The number is 3</p>
        <#break>
    <#default>
    <p> The number is neither 5 nor 3</p>
    <#break>
</#switch>
```

Η αντίστοιχη λειτουργικότητα προσθήκης περιεχόμενου από διαφορετικό αρχείο γίνεται με την εντολή <#include> π.χ.

```
<div>
    <p>Content goes here!</p>
</div>
<#include "/Directory/fileName.ftl">
```

Σε αυτό το παράδειγμα στην θέση της εντολής include θα εκτυπωθούν τα περιεχόμενα του αρχείου.

Η βιβλιοθήκη προσφέρει τις παρακάτω εντολές:
<#attempt >, <#recover >, <#attempt>. Αν ο κώδικας ανάμεσα στην εντολή attempt και στην εντολή recover παρουσιάσει κάποιο σφάλμα, ακυρώνεται η εκτέλεσή του και εκτελείται το block που είναι ανάμεσα στην εντολή recover και στην εντολή /attempt. π.χ.

```
1 <#attempt>
2     <p>$\{transationalVariable}</p>
3 <#recover>
4     <p>Error during transaction</p>
5 </#attempt>
```

Δίνεται επίσης η δυνατότητα δημιουργίας μεταβλητής στο αρχείο του template με την χρήση της εντολής <#assign>π.χ.

```
1 <#assign x=5 y=6 z=4>
2 <p>${x}</p>
```

Μια πολύ σημαντική δυνατότητα που δίνει αυτή η βιβλιοθήκη είναι η δυνατότητα να δημιουργήσει ο χρήστης δικές του εντολές.

7.4 Συμπεράσματα από την χρήση των διαφόρων βιβλιοθηκών

Οι βιβλιοθήκες προτύπων είναι μια πολύ καλή εναλλακτική στην χρήση της τεχνολογίας JSP για την δημιουργία δυναμικού περιεχομένου προερχόμενου από τον Web Server. Και οι δύο βιβλιοθήκες προσφέρουν όμοια λειτουργικότητα, όμως η βασική διαφορά της Thymeleaf είναι πως δίνει έμφαση στην δημιουργία αρχείων τα οποία μπορούν να ανοιχθούν από έναν φυλλομετρητή και να παρουσιαστούν ώς έχουν, δείχνοντας τον βασικό σχεδιασμό, χωρίς να χρειάζεται να εκτελεστούν από τον Web Server. Η βιβλιοθήκη Apache Freemarker δεν παρέχει αυτήν την δυνατότητα. Όμως προσφέρει πολλές διαφορετικές πρόσθετες λειτουργίες, καθώς και την δυνατότητα στον χρήστη να δημιουργήσει δικές του εντολές, ακόμα και βιβλιοθήκες με πρόσθετες custom εντολές. Η επιλογή βιβλιοθήκης για την ανάπτυξη της εφαρμογής έγκειται καθαρά στις προτιμήσεις του προγραμματιστή και τις ανάγκες της εκάστοτε εφαρμογής.

Κεφάλαιο 8

Τα σημαντικότερα Building Tools

8.1 Εισαγωγή

Όπως είπαμε, η πλατφόρμα Java προσφέρει εργαλεία με τα οποία μπορούμε να μεταγλωττίσουμε μια εφαρμογή γραμμένη στην γλώσσα και να την πακετάρουμε σε αρχεία JAR, αν πρόκειται για απλή εφαρμογή και WAR, αν πρόκειται για διαδικτυακή εφαρμογή. Σε περίπτωση που επιθυμούμε να προσθέσουμε κώδικα μιας βιβλιοθήκης ή ενός framework στον κώδικά μας, θα πρέπει να κατεβάσουμε τα αντίστοιχα αρχεία JAR, και να τα τοποθετήσουμε στις βιβλιοθήκες που μπορεί να χρησιμοποιήσει η εφαρμογή μας.

Για την αυτοματοποίηση αυτών των διαδικασιών δημιουργήθηκαν εργαλεία τα οποία ονομάζονται Build tools, ακριβώς επειδή κύρια ασχολία τους είναι το ¨κτίσιμο' των εφαρμογών, δηλαδή η μεταγλώτισση και το πακετάρισμα στα κατάλληλα αρχεία (JAR,WAR). Το πρώτο εργαλείο που δημιουργήθηκε με αυτό το σκοπό ήταν το Apache Ant. Δημιουργούσαμε αρχεία τα οποία περιέγραφαν την διαδικασία του building, και αυτό έπειτα τα εκτελούσε. Η λειτουργικότητά του όμως περιοριζόταν στο απλό ¨κτίσιμο' ήδη υπάρχοντος κώδικα, σε αντίθεση με τα εργαλεία που θα αναλύσουμε σε αυτό το κεφάλαιο, τα οποία διαθέτουν αρκετές ακόμα λειτουργίες.

8.2 Το εργαλείο build Maven

Το Apache Maven δημιουργήθηκε το 2004 για να δώσει λύση στο γεγονός πως τα διάφορα projects του Apache Foundation είχαν διαφορετική δομή και δεν είχαν έναν τυποποιημένο τρόπο να κάνουν build, αλλά για το κάθε ένα έπρεπε να δημιουργηθεί ένα διαφορετικό αρχείο για το εργαλείο Apache Ant. Σκοπός του Maven ήταν να διευκολύνει την διαδικασία του building, να δημιουργήσει ένα πρότυπο δομής για κάθε project γραμμένο σε Java και να βρεί έναν κοινό τρόπο να δίδονται σημαντικές πληροφορίες για κάθε project.

Κάθε project που δημιουργείται με την χρήση του Maven διαθέτει ένα αρχείο που ονομάζεται pom.xml. (Project Object Model). Σε αυτό το αρχείο περιέχονται διάφορες πληροφορίες για το project, καθώς και οι απαραίτητες ρυθμίσεις τις οποίες χρειάζεται το Maven για να εκτελέσει τις διάφορες λειτουργίες του πάνω στο project. Κάθε pom.xml κληρονομεί τις βασικές ρυθμίσεις από ένα αρχείο που ονομάζεται Super POM. Ο χρήστης μπορεί να αλλάξει οποιαδήποτε ρύθμιση δηλώνοντάς την στο pom.xml του project του. Κάθε pom.xml πρέπει να διαθέτει κάποιες συγκεκριμένες ρυθμίσεις.

α) root element του αρχείου πρέπει να είναι το <project>.
β) Πρέπει να διαθέτει την παρακάτω ρύθμιση: <modelVersion>4.0.0</modelVersion>
γ) groupId - Σε αυτή τη ρύθμιση ορίζεται ένα αναγνωριστικό για την εταιρία - τον προγραμματιστή στον οποίο ανήκει το project.
δ) artifactId - Σε αυτή τη ρύθμιση ορίζεται ένα αναγνωριστικό για το συγκεκριμένο project.
ε) version - Σε αυτή τη ρύθμιση ορίζεται ο αριθμός της έκδοσης του συγκεκριμένου project.

Ορίζοντας ένα στοιχείο parent, μπορούμε να κληρονομήσουμε τις ρυθμίσεις από ένα διαφορετικό αρχείο pom.xml π.χ.

```
1  <project>
2    <parent>
3      <groupId>com.mycompany.app</groupId>
4      <artifactId>my-app</artifactId>
5      <version>1</version>
6      <relativePath>../parent/pom.xml</relativePath>
7    </parent>
8    <modelVersion>4.0.0</modelVersion>
9    <artifactId>my-module</artifactId>
10 </project>
```

Το Maven ορίζει μια συγκεκριμένη δομή στα project του:

Οι βασικές κλάσεις της εφαρμογής πρέπει να βρίσκονται στον φάκελο src/main/java.
Οι πόροι της εφαρμογής πρέπει να βρίσκονται στον φάκελο src/main/resources.
Τα αρχεία φίλτρων της εφαρμογής πρέπει να βρίσκονται στον φάκελο src/main/filter
Τα αρχεία που περιέχουν πληροφορίες για το documentation πρέπει να βρίσκονται στον φάκελο src/site
Τα αρχεία διαδικτυακών εφαρμογών πρέπει να βρίσκονται στον φάκελο src/main/webapp
Οι κλάσεις των δοκιμών (tests) πρέπει να βρίσκονται στον φάκελο src/test/java
Οι πόροι των δοκιμών πρέπει να βρίσκονται στον φάκελο src/test/resources.
Τα αρχεία φίλτρων των δοκιμών πρέπει να βρίσκονται στον φάκελο src/test/filter
Το αρχείο της άδειας χρήσης της εφαρμογής πρέπει να βρίσκεται στον root φάκελο και να ονομάζεται LICENCE.txt
Το αρχείο readme της εφαρμογής πρέπει να βρίσκεται στον root φάκελο και να ονομάζεται README.txt

Η ύπαρξη αυτής της κοινής δομής απλοποίησε πολύ την διαδικασία ανάπτυξης των εφαρμογών καθώς ένας προγραμματιστής που έχει ασχοληθεί με την ανάπτυξη ενός έργου γραμμένου με την βοήθεια του Maven, αναγνωρίζει αυτή τη δομή σε όλες τις εφαρμογές που είναι γραμμένες με την βοήθεια του Maven.

Μια από τις χρησιμότερες δυνατότητες του Maven είναι αυτό που ονομάζουμε Dependency Mechanism. Αντί ο προγραμματιστής να πρέπει να βρεί τα αρχεία JAR που περιέχουν τον κώδικα που χρειάζεται για την εφαρμογή του, να τα κατεβάσει και να τα τοποθετήσει στο κατάλληλο μέρος, απλά καταγράφει στο αρχείο pom.xml τι χρειάζεται και το Maven αναλαμβάνει όλη τη διαδικασία. π.χ.

```
1  <project>
2      ....
3      <dependencies>
4          <dependency>
5            <groupId>group-a</groupId>
6            <artifactId>artifact-a</artifactId>
7            <version>1.0</version>
8          </dependency>
9          <dependency>
10           <groupId>junit</groupId>
11           <artifactId>junit</artifactId>
12           <version>4.11</version>
13           <scope>test</scope>
14         </dependency>
15     </dependencies>
16 </project>
```

Το Maven ουσιαστικά είναι ένα πρόγραμμα, το οποίο μπορούμε να καλέσουμε είτε μέσω εντολών στο command line, είτε μέσω plugin του IDE στο οποίο εργαζόμαστε, για να εκτελέσουμε διάφορες εργασίες στο project μας. Αυτές οι εργασίες είναι οι:

- mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DgroupId=com.mycompany.app -DartifactId=my-app
  Με αυτήν την εντολή δημιουργούμε ένα project με βάση το πρότυπο Archetype του Maven. Στις παραμέτρους -DgroupId, -DartifactId τοποθετούμε τις τιμές του groupId και του artifactId του project μας αντίστοιχα.
- mvn validate - Με αυτήν την εντολή ελέγχουμε αν το project μας περιέχει λάθη.
- mvn compile - Με αυτήν την εντολή το Maven μεταγλωττίζει το project μας.
- mvn test - Με αυτήν την εντολή το Maven τρέχει όλα τα tests που υπάρχουν στο project μας, και μας ενημερώνει αν ήταν όλα επιτυχή ή ποια απέτυχαν.
- mvn package - Με αυτήν την εντολή το Maven δημιουργεί το αρχείο JAR/WAR για το project μας.
- mvn install - Με αυτήν την εντολή το Maven εγκαθιστά το project μας στο μηχάνημά μας έτσι ώστε να μπορεί να χρησιμοποιηθεί ώς dependency σε κάποιο άλλο project.
- mvn deploy - Με αυτήν την εντολή το Maven καθιστά το project μας διαθέσιμο για χρήση από όλη την κοινότητά του.
- mvn clean - Με αυτήν την εντολή το Maven διαγράφει τα JAR/WAR αρχεία που έχουν ήδη δημιουργηθεί για το project μας.
- mvn site - Με αυτήν την εντολή το Maven δημιουργεί το documentation του project μας, σύμφωνα με τις πληροφορίες που έχουμε δώσει στο αρχείο pom.xml, και τα περιεχόμενα του φακέλου src/site

Ανάλογα με την εργασία που εκτελούμε στο project, έχουμε τη δυνατότητα να επιλέξουμε ποιά πακέτα από τα dependencies θα ενσωματωθούν στον κώδικά μας, με μια ρύθμιση των dependencies που ονομάζεται scope. Υπάρχουν 6 διαφορετικές τιμές που μπορεί να δοθούν σε αυτή τη ρύθμιση:

- compile - Αυτή είναι η default τιμή της ρύθμισης. Σημαίνει πως η dependency ενσωματώνεται στον κώδικά μας κατά την μεταγλώττιση.
- provided - Ίδιο με το compile, με την διαφορά πως την ενσωμάτωση της dependency αναλαμβάνει το JDK, ή κάποιος Servlet Container.
- system - Ίδιο με το provided, με την διαφορά πως το αρχείο JAR στο οποίο βρίσκεται η dependency το παρέχουμε εμείς, οπότε το Maven δεν το αναζητά στα μητρώα.
- runtime - Σημαίνει πως πως η dependency ενσωματώνεται στον κώδικά μας κατά την εκτέλεση.

- test - Σημαίνει πως πως η dependency ενσωματώνεται στον κώδικά μας κατά την εκτέλεση των δοκιμών.
- import - Αυτή η επιλογή είναι διαθέσιμη μόνο για pom dependencies. Προσθέτει τα dependencies του εν λόγω pom στο τρέχον.

8.3 Το εργαλείο build Gradle

Το Gradle είναι ένα εργαλείο αυτοματοποίησης της διαδικασίας του building.

Κάθε Gradle project διαθέτει ένα αρχείο που ονομάζεται build.gradle. Αυτό το αρχείο αποτελείται από εντολές μιας γλώσσας που ονομάζεται Groovy, με τις οποίες ορίζουμε διάφορες πληροφορίες τις οποίες χρειάζεται το εργαλείο για να εκτελέσει τις διάφορες λειτουργίες του στο project μας. Χρησιμοποιώντας εντολές σε αυτό το αρχείο, μπορούμε να ορίσουμε τις δικές μας λειτουργίες, τις οποίες μπορούμε να εκτελέσουμε έπειτα χρησιμοποιώντας το πρόγραμμα gradle. Κάθε λειτουργία, είτε κάποια από αυτές που είναι ήδη υλοποιημένες, είτε κάποια που ορίζουμε εμείς, ονομάζεται εργασία (task).

Το Gradle δημιουργήθηκε με σκοπό να αποτελέσει εργαλείο για το building έργων σε διάφορες γλώσσες προγραμματισμού, και όχι μόνο της Java. Για κάθε διαφορετική γλώσσα υπάρχει ένα αντίστοιχο plugin το οποίο ορίζουμε στο αρχείο build.gradle. Για την Java, υπάρχει το αντίστοιχο plugin java. Το Java plugin διαθέτει ένα σύνολο από υλοποιημένες tasks. Οι σημαντικότερες είναι οι:

- compileJava - Καλώντας αυτήν την εργασία μεταγλωττίζουμε τον κώδικά μας.
- compileTestJava - Καλώντας αυτήν την εργασία μεταγλωττίζουμε τα αρχεία των δοκιμών.
- test - Καλώντας αυτήν την εργασία τρέχουμε τις δοκιμές.
- javadoc - Καλώντας αυτήν την εργασία δημιουργούμε documentation κάνοντας χρήση του Javadoc.
- clean - Καλώντας αυτήν την εργασία διαγράφουμε τα αποτελέσματα προηγουμένων builds.
- jar - Καλώντας αυτήν την εργασία δημιουργούμε το αρχείο JAR για το τρέχον project.

Το Gradle γενικά δεν καθορίζει μια αυστηρή δομή για τα αρχεία και τους φακέλους των projects, σε αντίθεση με το Maven, αντίθετα μπορούμε να ορίσουμε ένα όνομα για ένα σύνολο φακέλων και να εκτελούμε tasks σε αυτό δίνοντας ώς παράμετρο το όνομα. Το Java plugin όμως, λόγω της υιοθέτησης της δομής των Java projects που όρισε το Maven στην πλειοψηφία των έργων, χρησιμοποιεί την εν λόγω δομή by default στην εκτέλεση των tasks της.

Το Gradle, όπως και το Maven, προσφέρει την δυνατότητα της αυτόματης προσθήκης dependencies στο project. Για να το κάνει αυτό χρειάζεται τρείς πληροφορίες:

α) Ποιο είναι αυτό το dependency (groupId, artifactId, version)
β) Σε ποιά από τις εργασίες θα ενσωματωθεί
γ) Που βρίσκεται

Για να ορίσουμε μια dependency την οποία το Gradle θα αναζητήσει στα κεντρικά Repositories του Maven θα πρέπει να γράψουμε στο αρχείο build.gradle τις παρακάτω εντολές:

```
1 ........
2
3 repositories {
4     mavenCentral()
5 }
6
7 dependencies {
8     implementation 'groupId:artifactId:Version'
9 }
```

Η εντολή mavenCentral() ορίζει πως το Gradle θα αναζητήσει την dependency στο κεντρικό Repository του Maven. Η λέξη implementation αντιπροσωπεύει μια dependency configuration και το όρισμα είναι τα στοιχεία της εκάστοτε dependency. Οι dependency configurations είναι η αντίστοιχη επιλογή scope του Maven. Ορίζει σε ποιές εργασίες θα ενσωματώνεται η dependency στον κώδικα του project. Το Gradle μας δίνει την δυνατότητα να δημιουργήσουμε τις δικές μας dependency configurations, καθώς και προσφέρει τις παρακάτω:

- compileOnly - Η dependency θα ενσωματώνεται στον κώδικα κατά την μεταγλώττιση.
- implementation - Η dependency θα ενσωματώνεται στον κώδικα κατά την μεταγλώττιση και την εκτέλεση.
- runtimeOnly - Η dependency θα ενσωματώνεται στον κώδικα μόνο κατά την εκτέλεση.
- testCompileOnly - Η dependency θα ενσωματώνεται στον κώδικα κατά την μεταγλώττιση των δοκιμών.
- testImplementation - Η dependency θα ενσωματώνεται στον κώδικα κατά την μεταγλώττιση και την εκτέλεση των δοκιμών.
- testRuntimeOnly - Η dependency θα ενσωματώνεται στον κώδικα μόνο κατά την εκτέλεση των δοκιμών.

Για να εκτελέσουμε ένα task χρησιμοποιούμε το πρόγραμμα gradle και το όνομα του task π.χ. gradle compileJava.

8.4 Συμπεράσματα από την χρήση των διαφόρων tools

Το Maven δημιουργήθηκε με σκοπό να δημιουργήσει ομοιογένεια στην δομή των διαφόρων Java Projects, όταν κάθε προγραμματιστής δημιουργούσε την δική του. Αυτή είναι η αιτία που υπάρχουν αυστηροί ορισμοί στην δομή των φακέλων του κάθε project και στην μορφή του αρχείου περιγραφής. Αντιθέτως το Gradle δημιουργήθηκε την εποχή που ο σκοπός της ύπαρξης του Maven είχε επιτευχθεί, ώς ένα εργαλείο που θα κάνει ευκολότερη την παραμετροποίηση και την διαφοροποίηση από τα στεγανά του Maven.

Από την άποψη της απόδοσης, το Gradle είναι ταχύτερο από το Maven, διότι κάνει χρήση της build cache, διαφοροποιεί δηλαδή κάθε νέο build σε σχέση μόνο με τα αρχεία που άλλαξαν από τα προηγούμενα, την κατάσταση των οποίων κρατά στην μνήμη.

Από την άλλη το Maven, σαν πιο ώριμη τεχνολογία, διαθέτει περισσότερα εργαλεία που βοηθούν στην ανάπτυξη των project του, καθώς και εκτενέστερο όγκο πληροφοριών σχετικά με αυτό.

Κεφάλαιο 9

Οι σημαντικότερες Testing Libraries

9.1 Εισαγωγή

Η διαδικασία με την οποία διασφαλίζουμε πως οι εφαρμογές που αναπτύσσουμε δουλεύουν σύμφωνα με τις απαιτούμενες προδιαγραφές ονομάζεται Testing. Μπορούμε να εκτελέσουμε αυτή τη διαδικασία είτε μόνοι μας, χρησιμοποιώντας την εφαρμογή και δοκιμάζοντας τις λειτουργίες της είτε κάνοντας χρήση κάποιων Testing Libraries, γράφοντας δηλαδή κώδικα ο οποίος θα ελέγχει την εφαρμογή μας.

Υπάρχουν διαφορετικά επίπεδα ελέγχου της λειτουργικότητας της εφαρμογής:

- Unit Testing - Σε αυτό το επίπεδο διενεργούμε ελέγχους στο μικρότερο δυνατό σύνολο κώδικα που υπάρχει, δηλαδή τις μεθόδους / τις κλάσεις της εφαρμογής ώστε να σιγουρευτούμε πως παράγουν το αναμενόμενο αποτέλεσμα.
- Integration Testing - Σε αυτό το επίπεδο διενεργούμε ελέγχους στον κώδικα που ενώνει τις μεθόδους που περιέχουν κομμάτια τις λειτουργικότητας, τα οποία έχουμε ελέγξει πως λειτουργούν σωστά, για να σιγουρευτούμε πως όλα δουλεύουν σύμφωνα με τις προδιαγραφές.
- System Testing - Σε αυτό το επίπεδο ελέγχουμε την εφαρμογή σαν σύνολο σε κατηγορίες όπως έλεγχος της λειτουργικότητας, έλεγχος ασφαλείας, έλεγχος απόδοσης.
- Beta Testing - Σε αυτό το επίπεδο διαθέτουμε την εφαρμογή προς χρήση σε ένα πλήθος χρηστών, ώστε να ελέγξουμε την εφαρμογή σε ένα σενάριο πολύ κοντά στην πραγματική χρήση.

Στα πλαίσια τις εργασίας θα ασχοληθούμε με βιβλιοθήκες οι οποίες μας βοηθούν να δημιουργήσουμε Unit tests.

9.2 JUnit -Το δημοφιλέστερο Unit Testing Framework

Το JUnit είναι η de facto βιβλιοθήκη η οποία χρησιμοποιείται για να δημιουργηθούν Unit Tests για εφαρμογές γραμμένες στην γλώσσα Java. Σκοπός της χρήσης του JUnit είναι να δημιουργηθεί κώδικας ο οποίος θα μπορεί να διενεργεί ελέγχους στον κώδικα της εφαρμογής μας, θα μπορεί να εκτελεστεί αυτούσιος, χωρίς να εκτελείται η εφαρμογή μας, και το αποτέλεσμα της εκτέλεσης του θα μας απαντά άν όλοι οι έλεγχοι μας ήταν επιτυχείς ή ποιοί από τους ελέγχους μας απέτυχαν και γιατί. Στο προηγούμενο κεφάλαιο είδαμε πως τα εργαλεία με τα οποία κάνουμε Build την εφαρμογή μας διαθέτουν την δυνατότητα να εκτελούν ελέγχους. Οι έλεγχοι αυτοί γίνονται με την χρήση κώδικα που χρησιμοποιεί το JUnit.

Το Maven Central Repository διαθέτει όλα τα πακέτα που χρειαζόμαστε για να χρησιμοποιήσουμε το JUnit, τα οποία μπορούμε να προσθέσουμε στην εφαρμογή μας κάνοντας χρήση οποιουδήποτε Building tool. Σε περίπτωση που δεν κάνουμε χρήση ενός τέτοιου εργαλείου, μπορούμε να κατεβάσουμε τα απαραίτητα αρχεία JAR και να τα προσθέσουμε στις βιβλιοθήκες της εφαρμογής μας. Η τρέχουσα έκδοση του JUnit σήμερα είναι η έκδοση 5.

Για να δημιουργήσουμε κώδικα που εκτελεί Unit tests στον κώδικα της εφαρμογής μας χρησιμοποιούμε τα κατάλληλα Annotations που μας παρέχει το JUnit. Τα πιο σημαντικά από αυτά είναι τα ακόλουθα:

- @Test, @ParameterizedTest, @RepeatedTest - Προσθέτοντας ένα από αυτά τα annotations στον ορισμό μιας μεθόδου, δημιουργούμε ένα Unit Test. Το @ParameterizedTest ορίζει ένα test στο οποίο μπορούμε να προσθέσουμε παραμέτρους, ενώ το @RepeatedTest ορίζει ένα test το οποίο επαναλαμβάνεται και παρέχει μεταβλητές που αντιπροσωπεύουν τις επαναλήψεις.
- @DisplayName - Ορίζει το όνομα με το οποίο θα εμφανίζεται η κλάση ή η μέθοδος που ορίζεται με την χρήση αυτού του Annotation.
- @BeforeEach, @AfterEach - Η μέθοδος που ορίζεται με την χρήση ενός από αυτά τα Annotations εκτελέται πρίν / μετά από κάθε μέθοδο Test.
- @BeforeAll, @AfterAll - Η μέθοδος που ορίζεται με την χρήση ενός από αυτά τα Annotations εκτελέται πρίν εκτελεστούν οι μέθοδοι Test της τρέχουσας κλάσης / αφού εκτελεστούν όλες οι μέθοδοι Test της τρέχουσας κλάσης.
- @Nested - Μια κλάση στον ορισμό της οποίας βρίσκεται το συγκεκριμένο Annotation δεν μπορεί να περιέχει μεθόδους που στον ορισμό τους βρίσκονται @BeforeAll / @AfterAll Annotations.
- @Disabled - Η κλάση ή η μέθοδος στον ορισμό της οποίας βρίσκεται το συγκεκριμένο Annotation θα αγνοηθεί και δεν θα εκτελεστεί κατά την εκτέλεση των Test.

Με την χρήση αυτών των Annotations μπορούμε να δημιουργήσουμε κλάσεις και μεθόδους που θα εκτελούνται με την χρήση των εργαλείων του JUnit. Σε αυτές τις μεθόδους θα πρέπει να γράψουμε τον κώδικα που θα ελέγχει την λειτουργικότητα του κώδικα της εφαρμογής μας.

Στο JUnit κάθε μέθοδος Test είτε μπορεί να είναι επιτυχής, είτε να αποτύχει. Αν το σύνολο του κώδικα μιας τέτοιας μεθόδου εκτελεστεί επιτυχώς, χωρίς να επιστρέψει κάποιο σφάλμα, τότε ο έλεγχος θεωρείται επιτυχής. Το JUnit μας προσφέρει ένα πλήθος από μεθόδους με τις οποίες μπορούμε να κάνουμε διάφορους ελέγχους, και σε περίπτωση που κάποιος έλεγχος αποτύχει, το Test μας να αποτύχει. Αυτές οι μέθοδοι ονομάζονται Assertions. Οι σημαντικότερες από αυτές είναι οι εξής:

- assertEquals(int expected, int actual, String message) - Η μέθοδος δέχεται σαν παραμέτρους την τιμή που περιμένουμε να πάρουμε από την εκτέλεση μιας εργασίας, την τιμή που παίρνουμε από την εκτέλεση μιας εργασίας και ένα μήνυμα που θα επιστραφεί σε περίπτωση αυτές οι δύο τιμές δεν συμπίπτουν. Επίσης σε αυτήν την περίπτωση επιστρέφει σφάλμα οδηγώντας το Test που την περιέχει σε αποτυχία. Αντίστοιχες μέθοδοι υπάρχουν για όλους τους γνωστούς τύπους δεδομένων.
- assertNotEquals(....) - Η αντίστροφη μέθοδος της προηγούμενης. Επιστρέφει σφάλμα μόνο όταν οι δύο τιμές συμπίπτουν.
- assertNotNull(int actual) / assertNull(int actual) - Αυτή η μέθοδος επιστρέφει σφάλμα όταν η τιμή της παραμέτρου είναι Null. / δεν είναι Null.
- assertSame(Object expected, Object actual) - Αυτή η μέθοδος επιστρέφει σφάλμα όταν οι δύο αυτές μεταβλητές δεν αναφέρονται στο ίδιο αντικείμενο.
- assertTrue(boolean actual) / assertFalse(boolean actual) - Αυτή η μέθοδος επιστρέφει σφάλμα όταν η τιμή της παραμέτρου είναι false / είναι true.
- assertTimeout(Duration timeout, Executable executable) - Αυτή η μέθοδος επιστρέφει σφάλμα αν η διαδικασία που δώσαμε σαν παράμετρο δεν ολοκληρωθεί μέσα στον χρόνο που δώσαμε σαν παράμετρο.
- assertThrows(Class<T>expectedType, Executable executable) - Αυτή η μέθοδος επιστρέφει σφάλμα αν η διαδικασία που δώσαμε σαν παράμετρο δεν επιστρέψει κάποιο exception της κλάσης που δίνουμε σαν παράμετρο.
- fail(String message) - Αυτή η μέθοδος επιστρέφει πάντοτε σφάλμα. Δέχεται σαν παράμετρο ένα μήνυμα που θα επιστραφεί μαζί με το σφάλμα.

Υπάρχει μια σειρά από μεθόδους με τις οποίες μπορούμε να εκτελέσουμε κάποιους ελέγχους και αν αυτοί αποτύχουν, το Test το οποίο τις περιέχει δεν αποτυγχάνει, αλλά σταματά να εκτελείται και το αποτέλεσμά του αγνοείται. Αυτές οι μέθοδοι ονομάζονται Assumptions.

Είδαμε νωρίτερα πως μπορούμε να αγνοήσουμε συγκεκριμένες κλάσεις ή μεθόδους που περιέχουν Test κάνοντας χρήση του @Disabled Annotation. Το JUnit όμως μας προσφέρει μια σειρά από Annotations με τα οποία η μέθοδος στον ορισμό της οποίας χρησιμοποιούνται εκτελείται ή αγνοείται με βάση κάποια συνθήκη. Τα πιο σημαντικά από αυτά είναι:

- @EnabledOnOs(OS) / @DisabledOnOs - Η μέθοδος που ορίζεται με την χρήση αυτού του Annotation εκτελείται μόνο στο συγκεκριμένο λειτουργικό σύστημα / αγνοείται σε όλα τα υπόλοιπα λειτουργικά συστήματα
- @EnabledOnJre(Version) / @DisabledOnJre- Η μέθοδος που ορίζεται με την χρήση αυτού του Annotation εκτελείται μόνο στην συγκεκριμένη έκδοση του Java Runtime Environment / αγνοείται σε όλες τις υπόλοιπες.
- @EnabledIf(expression) @DisabledIf - Η μέθοδος που ορίζεται με την χρήση αυτού του Annotation εκτελείται μόνο αν η αναγραφόμενη συνθήκη είναι αληθής / αγνοείται άν η αναγραφόμενη συνθήκη είναι αληθής.

Είδαμε τα βασικά εργαλεία που μας παρέχει το JUnit έτσι ώστε να δημιουργήσουμε τα δικά μας Unit Tests. Στο προηγούμενο κεφάλαιο είδαμε πώς τα διάφορα Building Tools μας βοηθούν να τρέξουμε αυτά τα Test. Ένας εναλλακτικός τρόπος για να τρέξουμε τα Tests μας είναι μέσω του IDE στο οποίο γράφουμε τον κώδικά μας. Όλα τα δημοφιλή IDE για την γλώσσα Java διαθέτουν μηχανισμούς για να τρέξουν JUnit Tests. Ένας τρίτος τρόπος είναι να χρησιμοποιήσουμε το Command Line tool του JUnit για να τρέξουμε τα Tests μας από την γραμμή εντολών.

9.3 Mockito - Ένα εύχρηστο Mocking Library

Κατά την διαδικασία του Unit Testing, σκοπός μας είναι σε κάθε Test να εξετάσουμε κάτι συγκεκριμένο, ώστε άν το τεστ αποτύχει να γνωρίζουμε όσο το δυνατόν περισσότερο τι πήγε στραβά. Ένα τυπικό κομμάτι κώδικα της εφαρμογής για το οποίο γράφουμε Unit tests όμως, συνήθως διαχειρίζεται αντικείμενα από άλλες κλάσεις που, είτε δεν μπορούμε να χρησιμοποιήσουμε κατά την διαδικασία του Testing (κάποια διαδικτυακή υπηρεσία στην οποία δεν έχουμε πρόσβαση εκείνη την στιγμή), είτε η χρήση τους ξεφεύγει από το σκοπό του συγκεκριμένου Test. Μια λύση σε αυτό το πρόβλημα είναι η δημιουργία ψεύτικων αντικειμένων (Mock Objects), τα οποία δημιουργούμε, προγραμματίζουμε την συμπεριφορά τους και τα χρησιμοποιούμε στα Unit Tests μας, καλώντας τις μεθόδους της εφαρμογής που τα χρειάζονται. Στα πλαίσια της εργασίας θα μελετήσουμε μια δημοφιλή βιβλιοθήκη με την οποία μπορούμε να δημιουργήσουμε και να διαχειριστούμε ψεύτικα αντικείμενα, το Mockito.

Η βιβλιοθήκη Mockito διαθέτει ένα σύνολο από στατικές μεθόδους με τις οποίες μπορούμε να δημιουργήσουμε ψεύτικα αντικείμενα, να καθορίσουμε πλήρως την συμπεριφορά τους και να επεξεργαστούμε πληροφορίες για την εκτέλεση μεθόδων αυτών των ψεύτικων αντικειμένων στα Tests μας.

Υπάρχουν δύο τρόποι να δημιουργήσουμε ένα ψεύτικο αντικείμενο. Ο πρώτος είναι κάνοντας χρήση της στατικής μεθόδου org.mockito.Mockito.mock(), η οποία δέχεται ώς παράμετρο μια κλάση και επιστρέφει ένα ψεύτικο αντικείμενο αυτής της κλάσης. Ο δεύτερος τρόπος είναι στην δήλωση ενός αντικειμένου να τοποθετήσουμε το Annotation @Mock.

Το βασικό πλεονέκτημα της χρήσης ψεύτικων αντικειμένων κατά την διαδικασία του Testing είναι το γεγονός πως μπορούμε να έχουμε τον πλήρη έλεγχο της συμπεριφοράς των αντικειμένων αυτών, ακόμα και άν δεν έχουμε γράψει εμείς τον κώδικά τους. Αυτό συμβαίνει διότι η βιβλιοθήκη μας προσφέρει μεθόδους οι οποίες αναγνωρίζουν πότε καλείται κάποια λειτουργία του ψεύτικου αντικειμένου και μεθόδους οι οποίες αντικαθιστούν το αποτέλεσμα αυτής της κλήσης, προσφέροντας την δική τους λειτουργικότητα. Πρόκειται για τις παρακάτω μεθόδους:

- when(Object obj) - Η μέθοδος παίρνει ώς παράμετρο μια μη void μέθοδο ενός ψεύτικου αντικειμένου, και επιστρέφει ένα αντικείμενο το οποίο διαθέτει μεθόδους οι οποίες καλούνται όταν καλεστεί η μέθοδος - παράμετρος.

- thenReturn(Object obj) - Αυτή είναι μια από τις μεθόδους του αντικειμένου που επιστρέφει η when. Δέχεται ώς παράμετρο ένα αντικείμενο και αντικαθιστά το αποτέλεσμα της μεθόδου - παραμέτρου της when με αυτό το αντικείμενο.

- thenThrow(Exception ex) - Αυτή είναι μια από τις μεθόδους του αντικειμένου που επιστρέφει η when. Δέχεται ώς παράμετρο ένα σφάλμα και το ενεργοποιεί όταν καλεστεί η μέθοδος παράμετρος της when.

- doReturn(Object obj) - Αυτή η μέθοδος παίρνει ώς παράμετρο το αντικείμενο που θα επιστραφεί κατα την κλήση μιας μεθόδου του ψεύτικου αντικειμένου και επιστρέφει ένα αντικείμενο το οποίο διαθέτει την μέθοδο when με την χρήση της οποίας μπορούμε να ορίσουμε ποιά είναι αυτή η μέθοδος. Αυτή η σύνταξη χρησιμοποιείται για να ελέγξουμε την συμπεριφορά των void μεθόδων του ψεύτικου αντικειμένου.
- doThrow(Exception ex) - Ομοίως με την doReturn, αυτή η μέθοδος χρησιμοποιείται για να ενεργοποιήσει κάποιο σφάλμα όταν κληθεί κάποια void μέθοδος του ψεύτικου αντικειμένου.

```
1 ArrayList<String> list;
2 list = mock(ArrayList.class); //Creating mock obj
3 when(list.get(3)).thenReturn("third item");
4 //list.get(3) returns the String "third item"
5
6 Exception ex = new Exception();
7
8 doThrow(Exception ex).when(list).clear();
9 // throws an exception when clear is called.
```

Η μέθοδος when δέχεται όπως είπαμε ώς παράμετρο την κλήση μιας μεθόδου του ψεύτικου αντικειμένου. Αυτή η μέθοδος μπορεί να διαθέτει τις δικές τις παραμέτρους. Για να ελέγξουμε την συμπεριφορά της κλήσης της μεθόδου του ψεύτικου αντικειμένου για κάθε διαφορετική τιμή των παραμέτρων, θα έπρεπε να δημιουργήσουμε διαφορετικές κλήσης της μεθόδου when. Για την περίπτωση όμως που θέλουμε διαφορετικές κλήσεις της when να παράγουν το ίδιο αποτέλεσμα η βιβλιοθήκη μας προσφέρει μεθόδους τις οποίες τοποθετούμε σαν παραμέτρους στην κλήση της μεθόδου του αντικειμένου εντός της when. Αυτές οι μέθοδοι Argument Matchers. Οι πιο σημαντικές ειναι οι παρακάτω:

- any() - Η when θα ενεργοποιηθεί στην κλήση της μεθόδου του ψεύτικου αντικειμένου με οποιαδήποτε τιμή αυτής της παραμέτρου, ακόμα και null.
- any(Class cls) - Η when θα ενεργοποιηθεί στην κλήση της μεθόδου του ψεύτικου αντικειμένου όταν η εν λόγω παράμετρος είναι αντικείμενο της κλάσης cls. Δεν δέχεται την τιμή null.
- anyInt() / anyString() / anyBoolean() - Η when θα ενεργοποιηθεί στην κλήση της μεθόδου του ψεύτικου αντικειμένου όταν η εν λόγω παράμετρος είναι int/String/boolean. Δεν δέχεται την τιμή null.
- anyIterable() - Η when θα ενεργοποιηθεί στην κλήση της μεθόδου του ψεύτικου αντικειμένου όταν η εν λόγω παράμετρος είναι αντικείμενο μιας κλάσης η οποία κάνει implement το Iterable Interface. Δεν δέχεται την τιμή null.
- anyList() - Η when θα ενεργοποιηθεί στην κλήση της μεθόδου του ψεύτικου αντικειμένου όταν η εν λόγω παράμετρος είναι αντικείμενο μιας κλάσης η οποία κάνει implement το List Interface. Δεν δέχεται την τιμή null.

- notNull() - Η when θα ενεργοποιηθεί στην κλήση της μεθόδου του ψεύτικου αντικειμένου με οποιαδήποτε τιμή αυτής της παραμέτρου, εκτός από null.

```
1 ArrayList<String> list;
2 list = mock(ArrayList.class); //Creating mock obj
3 when(list.get(anyInt())).thenReturn("an Integer");
4 //any list.get call with an int argument
5 //returns the String "an Integer"
```

Η βιβλιοθήκη μας δίνει την δυνατότητα να ελέγξουμε αν οι μέθοδοι του ψεύτικου αντικειμένου καλέστηκαν με την χρήση της μεθόδου verify. Η μέθοδος δέχεται ώς παράμετρο ένα ψεύτικο αντικείμενο και επιστρέφει ένα αντικείμενο Stubber, το οποίο χρησιμοποιούμε για να καταδείξουμε ποιές συναρτήσεις θέλουμε να δούμε αν καλέστηκαν. Σε περίπτωση που το Test μας ολοκληρωθεί χωρίς να καλεστούν αυτές οι συναρτήσεις, τότε αποτυγχάνει.

Υπάρχει και η δυνατότητα να ελέγξουμε τον αριθμό τον φορών που καλέστηκαν οι συναρτήσεις των ψεύτικων αντικειμένων, δίνοντας ώς παράμετρο στην μέθοδο verify κάποιες μεθόδους που μας προσφέρει η βιβλιοθήκη. Αυτές οι μέθοδοι είναι οι:

- times(int times) - Η μέθοδος αυτή ορίζει τον αριθμό των φορών που πρέπει να καλεστεί η αντίστοιχη μέθοδος του ψεύτικου αντικειμένου. Αν καλεστεί λιγότερες ή περισσότερες, τότε το Test μας αποτυγχάνει.

- never() - Η μέθοδος αυτή ορίζει πως η αντίστοιχη μέθοδος του ψεύτικου αντικειμένου δεν πρέπει να καλεστεί ποτέ στα πλαίσια αυτού του Test. Αν η μέθοδος καλεστεί, τότε το Test μας αποτυγχάνει.

- atLeastOnce() / atLeast(int times) - Η μέθοδος αυτή ορίζει πως η αντίστοιχη μέθοδος του ψεύτικου αντικειμένου πρέπει να καλεστεί τουλάχιστον μια φορά / τουλάχιστον times φορές. Αν καλεστεί λιγότερες, τότε το Test μας αποτυγχάνει.

- atMost(int times) - Η μέθοδος αυτή ορίζει πως η αντίστοιχη μέθοδος του ψεύτικου αντικειμένου πρέπει να καλεστεί το πολύ times φορές. Αν καλεστεί περισσότερες, τότε το Test μας αποτυγχάνει.

- timeout(int milliseconds) - Αυτή η μέθοδος ορίζει χρόνο μέσα στον οποίο πρέπει η αντίστοιχη μέθοδος του ψεύτικου αντικειμένου να καλεστεί. Μπορεί να συνδυαστεί με τις παραπάνω μεθόδους. Αν ο χρόνος περάσει χωρίς να έχουν γίνει οι απαραίτητες κλήσεις, τότε το Test μας αποτυγχάνει.

Παρακάτω παραθέτονται μερικά παραδείγματα χρήσης των ανωτέρω:

```
1  ArrayList<String> list;
2  list = mock(ArrayList.class); //Creating mock obj
3  verify(list).get(anyInt());
4  //if no list.get is called then test fails
5
6  verify(list, times(2)).get(anyInt());
7  //list.get must get called 2 times or test fails
8
9  verify(list, never()).get(anyInt());
10 //if list.get gets called test fails
11
12 verify(list, atLeast(3)).get(anyInt());
13 //if list.get gets called less than 3 times test fails
14
15 verify(list, atMost(3)).get(anyInt());
16 //if list.get gets called more than 3 times test fails
17
18 verify(list, timeout(1000).atLeast(1)).get(anyInt());
19 //if list.get doesn't get called at least once
20 //in 1 second test fails
```

Τέλος υπάρχει η δυνατότητα όταν καλούμε μια μέθοδο ενός ψεύτικου αντικειμένου, να εκτελεστεί η πραγματική μέθοδος. Αυτό γίνεται με την χρήση της μεθόδου thenCallRealMethod() μετά από την μέθοδο when π.χ.

```
1  ArrayList<String> list;
2  list = mock(ArrayList.class); //Creating mock obj
3
4  when(list.get(anyInt())).thenCallRealMethod();
5  //list.get calls the real get method
6  //instead of a potential stub
```

Κεφάλαιο 10

Η εφαρμογή

10.1 Περιγραφή της εφαρμογής

Όπως αναφέρθηκε προηγουμένως, στα πλαίσια αυτής της εργασίας αναπτύχθηκε λογισμικό με σκοπό την εφαρμογή στην πράξη των εργαλείων και των τεχνικών ανάπτυξης που παρουσιάζονται σε αυτήν. Πιο συγκεκριμένα, δημιουργήθηκαν τρείς εκδοχές της ίδιας διαδικτυακής εφαρμογής, κάνοντας χρήση των τριών Web MVC Frameworks που είδαμε στο κεφάλαιο 3. Και στις τρείς εκδοχές γίνεται επίσης χρήση των παρακάτω τεχνολογιών:

α) MySQL Database
β) Hibernate
γ) Bootstrap
δ) Spring Security Module για την προσθήκη της λειτουργίας σύνδεσης χρήστη

Η εν λόγω εφαρμογή είναι λογισμικό για την διαχείριση μιας αποθήκης. Διαχειρίζεται διάφορα δεδομένα για τα περιεχόμενα της αποθήκης και για τα άτομα που είτε προμηθεύουν είτε προμηθεύονται προϊόντα από αυτήν. Πιο συγκεκριμένα, κρατά τα παρακάτω δεδομένα:

α) Κατηγορίες προϊόντων - Id και όνομα κατηγορίας.
β) Εγγεγραμμένοι πελάτες - Id και όνομα πελάτη.
γ) Προϊόντα - Κωδικό προϊόντος, όνομα προϊόντος, διαθέσιμη ποσότητα, κωδικό αποθήκευσης (Πληροφορία για το σημείο της αποθήκης όπου βρίσκεται το εν λόγω προϊόν).
δ) Ιστορικό συναλλαγών - Κωδικό προϊόντος, όνομα προϊόντος, όνομα πελάτη με τον οποίο έγινε η συναλλαγή, ποσότητα που μεταφέρθηκε (θετική άν τοποθετήσαμε προϊόν στην αποθήκη, αρνητική άν αφαιρέσαμε) και η στιγμή που αποθηκεύθηκε η συναλλαγή.

Παρέχονται οι παρακάτω λειτουργίες:

- Εμφάνιση των προϊόντων που βρίσκονται στην αποθήκη. Αναζήτηση με βάση οποιοδήποτε από τα πεδία τους.
- Εμφάνιση των συναλλαγών που έχουν γίνει ώς εκείνη τη στιγμή.
- Διαχείριση των κατηγοριών προϊόντων.
- Διαχείριση των εγγεγραμμένων πελατών.
- Εισαγωγή και εξαγωγή προϊόντων.

Όλες οι λειτουργίες εκτός από την εμφάνιση των προϊόντων απαιτούν την σύνδεση ενός εξουσιοδοτημένου χρήστη για να είναι προσβάσιμες.

Παρακάτω παραθέτουμε κάποια χαρακτηριστικά Screenshots από την εφαρμογή:

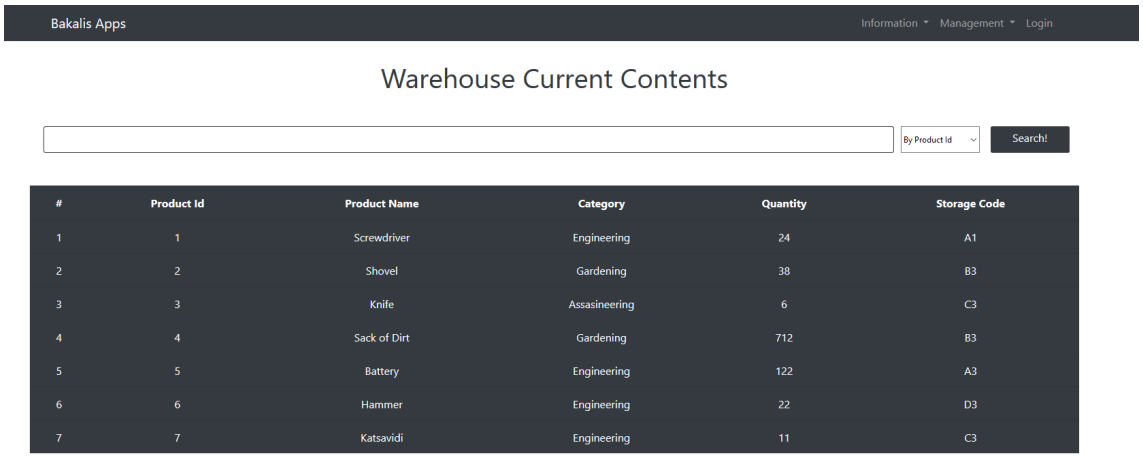

Figure 10.1: Η σελίδα εμφάνισης των περιεχομένων της αποθήκης

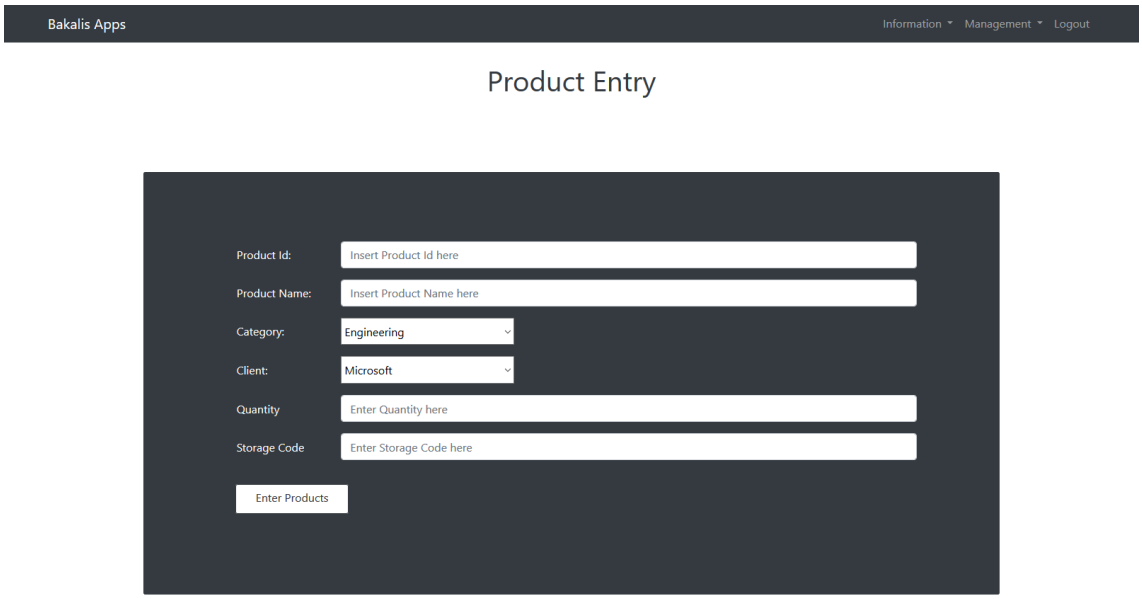

Figure 10.2: Η σελίδα εισαγωγής προϊόντων

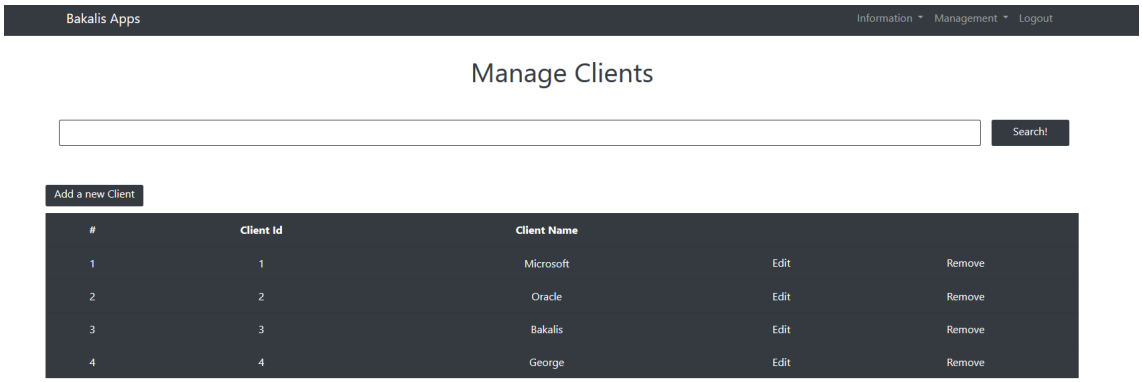

Figure 10.3: Η σελίδα διαχείρισης εγγεγραμμένων πελατών

10.2 Ανάλυση του κώδικα σε κάθε επίπεδο αρχιτεκτονικής

Η εφαρμογή μας έχει δημιουργηθεί σε τρείς εκδοχές, καθεμία από αυτές με την χρήση ενός διαφορετικού Web MVC Framework. Σε αυτό το μέρος θα παρουσιάσουμε μερικά παραδείγματα του πώς το κάθε εργαλείο υλοποιεί τις λειτουργίες σε κάθε επίπεδο αρχιτεκτονικής.

Σε όλες τις εφαρμογές, στο επίπεδο του Business Logic, της υλοποίησης δηλαδή της λογικής της εφαρμογής, έχουμε δημιουργήσει δύο Business Service κλάσεις, στις οποίες έχουμε υλοποιήσει όλες τις απαραίτητες μεθόδους τις οποίες καλούμε από τις κατάλληλες δομές των εφαρμογών, ώστε να εκτελέσουμε την απαραίτητη λειτουργικότητα. Οι Business Services αυτές είναι η κλάση ContentsService, στην οποία υπάρχουν μέθοδοι οι οποίες ανακτούν και επιστρέφουν πληροφορίες, και η κλάση TransactionsService, στην οποία υπάρχουν μέθοδοι οι οποίες επηρεάζουν την κατάσταση αυτών των πληροφοριών, είτε προσθέτουν καινούρια δεδομένα, είτε τροποποιούν τα υπάρχοντα. Έχουμε δημιουργήσει επίσης δύο βοηθητικές Business Services, τις ErrorLoggingService, η οποία κρατά ένα στατικό String, το οποίο χρησιμοποιείται για την αποθήκευση κάποιου πιθανού μηνύματος σφάλματος, και διαθέτει μεθόδους για τον έλεγχο της ύπαρξής του και την τροποποίησή του, καθώς και την ValidationService, η οποία περιέχει μεθόδους που εκτελούν ελέγχους εγκυρότητας στα δεδομένα μας. Η αλληλεπίδραση της εφαρμογής με την βάση δεδομένων λαμβάνει χώρα στις Business Services ContentsService και TransactionsService, κάνοντας χρήση του Criteria API του Hibernate.

Στην εφαρμογή που δημιουργήσαμε με την βοήθεια του Spring Web MVC, ο Controller βρίσκεται σε μια κλάση η οποία έχει οριστεί με το στερεοτυπικό Annotation @Controller, η οποία ονομάζεται MappingController. Οι μέθοδοι αυτής της κλάσης διαχειρίζονται τις διάφορες αιτήσεις HTTP που δέχεται η εφαρμογή και επιστρέφουν μια τιμή String, η οποία χρησιμοποιείται για να καθορίσει το View το οποίο θα δημιουργηθεί για να επιστραφεί ώς απάντηση. Τα δεδομένα τα οποία θα είναι προσβάσιμα από αυτό το View τοποθετούνται σε ένα αντικείμενο της κλάσης Model, το οποίο μας παρέχει το Spring Framework σαν παράμετρο αυτής της μεθόδου. π.χ.

```
1      @GetMapping("/index")
2      public String index(Model model){
3          ArrayList<Product> contents =
4          contentsService.getAllContents();
5          model.addAttribute("contents", contents);
6          errorService.passErrorMessageToModel(model);
7          contentsService.passAuthObjectToModel(model);
8          return "index";
9  }
```

Η τεχνολογία View που χρησιμοποιούμε στην εφαρμογή με την χρήση του Spring Web MVC είναι η Thymeleaf. Για την χρήση αυτής της τεχνολογίας υπάρχει εκτενές περιεχόμενο στο κεφάλαιο 7.

Στην εφαρμογή που δημιουργήθηκε με την χρήση του Java Server Faces, ο Controller παρέχεται από το Framework. Αυτό σημαίνει πως η κλήση του κατάλληλου View για κάθε αίτηση γίνεται αυτόματα από το FacesServlet. Όπως αναφέραμε και στο κεφάλαιο 3, τα Views έχουν πρόσβαση στα απαραίτητα δεδομένα μέσω αντικειμένων που διαχειρίζεται το Framework τα οποία ονομάζονται ManagedBeans. Η ροή των δεδομένων από τα Views στα ManagedBeans και αντίστροφα είναι ευθύνη του Framework. Υπάρχει επίσης η δυνατότητα σύνδεσμοι ή κουμπιά των Views να καλούν απ' ευθείας μεθόδους των ManagedBeans. Η λειτουργικότητα της εφαρμογής που δημιουργήθηκε με την χρήση αυτού του Framework είναι βασισμένη σε μεγάλο βαθμό σε αυτήν την δυνατότητα. Ακολουθεί παράδειγμα της χρήσης:

```
1     <h:commandButton value="Create Client"
2      class="btn btn-inverse"
3      action="#{clients.addClient()}" />
```

Αυτό το κομμάτι κώδικα δημιουργεί ένα Submit Button μιας φόρμας, το οποίο όταν πατηθεί καλεί την μέθοδο addClient του ManagedBean με το όνομα clients. Η εν λόγω μέθοδος παρατίθεται παρακάτω:

```
1  public String addClient(){
2          transactionsService.addClient(this.newClientId,
3          this.newClientName);
4          this.setNewClientId(null);
5          this.setNewClientName(null);
6          return "clients.xhtml?faces-redirect=true";
7  }
```

Οι απαραίτητες τιμές που χρησιμοποιούνται στην μέθοδο, όπως βλέπουμε, είναι τιμές των πεδίων του ίδιου του ManagedBean. Αυτό συμβαίνει διότι κάνουμε Bind το περιεχόμενο των στοιχείων input της φόρμας στα πεδία του αντίστοιχου ManagedBean. π.χ.

```
1     <h:inputText value="#{clients.editedClient.id}"
2      class="form-control" readonly="true"></h:inputText>
```

Αυτό το κομμάτι κώδικα δημιουργεί ένα Input τύπου text, η τιμή του οποίου αντιστοιχίζεται με το πεδίο id του αντικειμένου editedClient του ManagedBean με το όνομα clients.

Η τεχνολογία View που χρησιμοποιείται σε μια εφαρμογή δημιουργημένη με το Java Server Faces, όπως είπαμε ονομάζεται Facelets και βασίζεται σε βιβλιοθήκες που επεκτείνουν την γλώσσα XHTML. Οι βασικότερες από αυτές τις βιβλιοθήκες είναι:

α)Η βιβλιοθήκη JSTL, που ορίζεται με το πρόθεμα c. Η βασικότερη εντολή αυτής της βιβλιοθήκης, που χρησιμοποιούμε στην εφαρμογή μας είναι η c:if, η οποία χρησιμοποιείται για να ορίσει περιεχόμενο, το οποίο θα εκτυπωθεί με βάση το αν ισχύει μια συνθήκη.
β)Η βιβλιοθήκη UI, που ορίζεται με το πρόθεμα ui. Η βασικότερη εντολή, που χρησιμοποιούμε στην εφαρμογή μας είναι η ui:repeat, η οποία χρησιμοποιείται για να επαναλάβει περιεχόμενο με βάση τα περιεχόμενα μιας συλλογής αντικειμένων.
γ)Η βιβλιοθήκη HTML, που ορίζεται με το πρόθεμα h. Αυτή είναι η βασικότερη βιβλιοθήκη, και στην εφαρμογή χρησιμοποιούνται πολλές εντολές της, όπως η h:head η οποία δημιουργεί το head tag, η h:body, που δημιουργεί το body tag, η h:form, που δημιουργεί μια φόρμα, η h:inputText, που δημιουργεί ένα input τύπου text και αρκετές άλλες. π.χ.

```
1  <c:if test="#{error.error!=null}">
2      <div class="alert alert-danger text-center">
3      #{error.error}
4      </div>
5  </c:if>
6  <h:form method="post">
7          <div class="form-row searchRow">
8              <div class="col-lg-11">
9                  <h:inputText
10                 value="#{categories.searchBar}"
11                 class="searchBar"></h:inputText>
12             </div>
13             <div class="col-lg-1">
14                 <h:commandButton value="Search!"
15                 class="btn searchBtn"
16                 action="#{categories
17                 .getSearchedCategories()}" />
18             </div>
19         </div>
20 </h:form>
```

Στην εφαρμογή που δημιουργήθηκε με την χρήση του Struts 2 Web MVC Framework, οι Controllers είναι ένας συνδυασμός του Struts Filter, των μεθόδων execute των κλάσεων Action και του αρχείου struts.xml. Μέσω αυτών ορίζεται ο κώδικας που θα εκτελεστεί για κάθε αίτηση HTTP. Τα απαραίτητα δεδομένα τα οποία είναι προσβάσιμα από τα Views είναι αποθηκευμένα ώς πεδία στο αντίστοιχο Action το οποίο κάλεσε το εκάστοτε View. Ομοίως με την περίπτωση του Java Server Faces, την μεταφορά των δεδομένων από τα Views στα πεδία των Action Classes και αντιστρόφως αναλαμβάνει το framework. Όταν η εφαρμογή κάνει χρήση μιας φόρμας για την αποστολή δεδομένων στον Server, τα δεδομένα αποθηκεύονται στα αντίστοιχα πεδία της Action Class η οποία διαχειρίζεται αυτό το αίτημα. Επίσης, ομοίως με την περίπτωση του Java Server Faces, υπάρχει η δυνατότητα να κάνουμε Bind το περιεχόμενο των στοιχείων input της φόρμας στα πεδία του αντίστοιχου Action Class. π.χ.

```
1  <s:textfield cssClass="form-control" name="productId"
2  placeholder="Insert Product Id here"/>
```

Αυτό το κομμάτι κώδικα δημιουργεί ένα Input τύπου text, η τιμή του οποίου αντιστοιχίζεται με το πεδίο productId της κλάσης Action που διαχειρίζεται το Submission της φόρμας.

Η τεχνολογία View που χρησιμοποιούμε στην εφαρμογή που είναι δημιουργημένη με το Struts 2 είναι η τεχνολογία JSP, επεκτεινόμενη με την χρήση της βιβλιοθήκης tags που προσφέρει το Framework. Αυτή η βιβλιοθήκη χρησιμοποιεί το πρόθεμα s, και περιέχει ένα σύνολο απο εντολές οι οποίες δημιουργούν τα διάφορα απαραίτητα στοιχεία HTML, που χρειαζόμαστε για να δημιουργήσουμε την εφαρμογή μας, καθώς και Attributes τα οποία επιτρέπουν σε αυτά τα στοιχεία να έχουν πρόσβαση στα δεδομένα. π.χ.

```
1       <div class="form-group row">
2          <label for="categoryFilter"
3          class="col-sm-2 col-form-label">
4          Category:</label>
5          <div class="col-lg-10">
6             <s:select name="category" list="categories"
7             cssClass="categoryFilter"/>
8          </div>
9       </div>
10      <div class="form-group row">
11         <label for="code" class="col-sm-2 col-form-label">
12         Storage Code</label>
13         <div class="col-lg-10">
14            <s:textfield cssClass="form-control" name="code"
15            placeholder="Enter Storage Code here"/>
16         </div>
17      </div>
```

10.3 Διαφορές και συμπεράσματα από την χρήση των διαφορετικών frameworks

Κατ΄ αρχάς, και τα τρία αυτά Frameworks είναι σχεδιασμένα για να υλοποιούν την αρχιτεκτονική MVC, με σκοπό την δημιουργία διαδικτυακών εφαρμογών. Αυτό σημαίνει πως, επειδή υλοποιούν την ίδια λογική, έχουν πάρα πολλές ομοιότητες. Έχουν όμως και αρκετές διαφορές, οι οποίες κυρίως οφείλονται στην διαφορετική οπτική με την οποία αντιμετωπίζουν αρκετά ζητήματα.

Το Spring Web MVC είναι χτισμένο σαφώς με γνώμονα τη νοοτροπία με την οποία έχει δημιουργηθεί το Spring Framework, όπως την περιγράψαμε στο αντίστοιχο κεφάλαιο. Προσπαθεί δηλαδή να δώσει στον προγραμματιστή το μεγαλύτερο δυνατό ποσοστό ελευθερίας στο πώς θα δημιουργήσει την εφαρμογή του, καθώς και το τι εργαλεία θα χρησιμοποιήσει. Γι΄ αυτό το λόγο απαιτεί και τη μεγαλύτερη ποσότητα αρχικών ρυθμίσεων από τα τρία. Ο κύριος τρόπος με τον οποίο το επιτυγχάνει είναι ο διαχωρισμός των διαφόρων ευθυνών για κάθε ζήτημα σε διαφορετικά αντικείμενα (Beans), και η ύπαρξη διαφορετικών διαθέσιμων αντικειμένων για κάθε εργαλείο. Επίσης, καθώς βασίζεται στο Spring Core, η ύπαρξη του IoC Container, καθιστά εύκολη την κεντρική διαχείριση των αντικειμένων που είτε έχει δημιουργήσει ο χρήστης, είτε έχουν δημιουργηθεί και αρχικοποιηθεί από το ίδιο το Framework. Αυτό μας δίνει την δυνατότητα να έχουμε εύκολη πρόσβαση σε Singleton αντικείμενα από κάθε σημείο της εφαρμογής μας.

Μια άλλη πολύ σημαντική διαφορά του Spring Web MVC με τα άλλα δύο, είναι ο τρόπος με τον οποίο γίνεται η αντιστοίχιση των τιμών που είναι προσβάσιμες από τα Views, και των αντικειμένων του κώδικα Java. Για κάθε διαφορετικό αίτημα, ο Controller αναλαμβάνει να συλλέξει τα κατάλληλα δεδομένα και να τα τοποθετήσει σε ένα αντικείμενο, το οποίο θα είναι προσβάσιμο από το View. Στην αντίστροφη περίπτωση, όπου θέλουμε να μεταφέρουμε δεδομένα από τα Views προς τον κατάλληλο Controller, το ίδιο το Framework αναλαμβάνει να κάνει Inject τις κατάλληλες τιμές, σαν παραμέτρους στην μέθοδο που διαχειρίζεται το αίτημα, κάνοντας χρήση των φορμών HTML και της κατάλληλης μεθόδου. Αυτό σημαίνει πως κάθε ανταλλαγή δεδομένων από τον Controller στα Views και αντίστροφα είναι προσωρινή και αφορά μόνο το συγκεκριμένο αίτημα. Όπως θα δούμε έπειτα, τα άλλα Frameworks αντιμετωπίζουν διαφορετικά αυτό το ζήτημα.

Το Java Server Faces, αντιθέτως, έχει την νοοτροπία της Java Enterprise Edition. Υπάρχει δηλαδή η προσπάθεια όσο το δυνατόν μεγαλύτερης προτυποποίησης στον τρόπο ανάπτυξης μιας διαδικτυακής εφαρμογής με την χρήση του. Αυτό, ενώ διευκολύνει την ανάπτυξη, ελαχιστοποιεί τον όγκο των αρχικών ρυθμίσεων και μετατρέπει ένα μεγάλο μέρος της εφαρμογής σε κάτι γνώριμο, που είναι παρόμοιο σε κάθε εφαρμογή γραμμένη με την χρήση του, σε πολλές περιπτώσεις περιορίζει τον προγραμματιστή στην χρήση συγκεκριμένων πρακτικών. Χαρακτηριστικό παράδειγμα είναι η χρήση συγκεκριμένων διευθύνσεων για την ανάκτηση των κατάλληλων Views. Ενώ στο Spring MVC ορίζουμε ένα μοτίβο διευθύνσεων και αναθέτουμε σε συγκεκριμένη

μέθοδο του Controller την διαχείρισή της, στο Java Server Faces χρειάζεται να χρησιμοποιούμε αποκλειστικά διευθύνσεις που αντιστοιχούν σε συγκεκριμένο αρχείο xhtml. Αυτό συμβαίνει διότι ο προγραμματιστής δεν δημιουργεί, ούτε παραμετροποιεί τον Controller της εφαρμογής, αλλά χρησιμοποιεί τον ήδη υπάρχοντα, ο οποίος λειτουργεί με τον δικό του τρόπο.

Επίσης, στο Java Server Faces, οι επιλογές που δίνονται για την χρήση των τεχνολογιών View είναι δύο: JSP και Facelets, και η χρήση της JSP δεν ενδύκνειται. Ουσιαστικά το Framework μας περιορίζει στην χρήση των Facelets, τα οποία ωστόσο αποτελούν μια αρκετά εύχρηστη τεχνολογία.

Όπως είπαμε, σε αντίθεση με το Spring Web MVC, οι μεταβλητές που είναι προσβάσιμες από τα διάφορα Views μιας εφαρμογής δημιουργημένης με την χρήση του Java Server Faces αποτελούν πεδία κλάσεων οι οποίες ονομάζονται Managed Beans. Την μεταφορά των δεδομένων από το ManagedBean προς το View και αντίστροφα την αναλαμβάνει το ίδιο το Framework και, αντίθετα από την περίπτωση του Spring Web MVC, και στις δύο περιπτώσεις χρησιμοποιείται το ίδιο αντικείμενο. Επίσης, υπάρχει η δυνατότητα να παρατείνουμε τον κύκλο ζωής αυτών των αντικειμένων, από όσο διαρκεί μια αίτηση, έως όσο τρέχει η εφαρμογή. Πρέπει να δώσουμε λοιπόν ιδιαίτερη προσοχή στο τι τιμή έχουν αυτές οι μεταβλητές ανα πάσα στιγμή.

Το τρίτο Web MVC Framework που εξετάζουμε, το Struts 2, έχει περισσότερες ομοιότητες με το Java Server Faces, συγκριτικά με το Spring Web MVC. Πιο συγκεκριμένα, οι μεταβλητές που είναι προσβάσιμες από τα διάφορα Views είναι πεδία των κλάσεων Action. Η διάρκεια ζωής όμως των αντικειμένων είναι αποκλειστικά όσο διαρκεί το αίτημα.

Η βασική διαφορά του Struts 2 με το Java Server Faces είναι το γεγονός πως για μια εφαρμογή δημιουργημένη με την χρήση του, θα πρέπει να δημιουργήσουμε τους δικούς μας Controllers. Αυτοί οι Controllers είναι ένας συνδυασμός των μεθόδων execute που υπάρχουν στις διάφορες Action Classes, καθώς και του αρχείου Configuration struts.xml. Αυτό όμως μας δίνει μια ευελιξία που δεν συναντάμε στην αντίστοιχη προηγούμενη περίπτωση.

Στο Struts 2, προτεινόμενη τεχνολογία View είναι η JSP, καθώς το Framework μας προσφέρει τις βιβλιοθήκες tags για να δημιουργήσουμε το απαραίτητο User Interface. Σε αντίθεση με το Java Server Faces όμως, δεν μας περιορίζει στην χρήση της, και η δυνατότητα επιλογής εναλλακτικών τεχνολογιών είναι δυνατή. Γενικότερα, η δυνατότητα παραμετροποίησης και ευελιξίας μιας εφαρμογής δημιουργημένης με το Struts 2 είναι πολύ μεγαλύτερη.

Γενικότερα, το Struts 2 προσπαθεί να δώσει μια απάντηση στις δυσκαμψίες που μπορούν να παρουσιαστούν κατά την χρήση του Java Server Faces, ενώ κατα τα άλλα προσπαθεί να εφαρμόσει όμοια λογική κατά την ανάπτυξη των εφαρμογών.

Τέλος, και τα τρία Web MVC Frameworks μας προσφέρουν την δυνατότητα να δημιουργήσουμε διαδικτυακές εφαρμογές. Την κύρια διαφορά την κάνει το Spring Web MVC, καθώς οι δυνατότητες επιλογής των επιθυμητών εργαλείων και τεχνολογιών και η ευκολία στο Integration τους με την εφαρμογή είναι σαφώς ανώτερες από τα άλλα δύο. Το Java Server Faces πάλι, και σε ένα βαθμό και το Struts 2, είναι ένα Straightforward Framework, προσφέρει δηλαδή έναν συγκεκριμένο τρόπο και συγκεκριμένα εργαλεία για την δημιουργία διαδικτυακών εφαρμογών.