ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη Web Tutoring Service με Java EE 7

Του φοιτητή

Σουβατζή Βασίλη

Αρ. Μητρώου: 05/2781

Επιβλέπων καθηγητής

Σφέτσος Παναγιώτης

Θεσσαλονίκη 2014

ΠΡΟΛΟΓΟΣ

Ο παγκόσμιος ιστός έχει βοηθήσει σε μεγάλο βαθμό τη διακίνηση πληροφοριών και την επικοινωνία των ανθρώπων. Εταιρείες και οργανισμοί με τα δικά τους τοπικά υπολογιστικά συστήματα, έχουν συνδεθεί μεταξύ τους σε μικρότερα δίκτυα και μεταδίδουν πληροφορίες γρηγορότερα από ποτέ.

Οι θυγατρικές μιας εταιρείας που βρίσκονται διασκορπισμένες σε διάφορες χώρες, επικοινωνούν μέσω ενός κεντρικού συστήματος κι έχουν πρόσβαση στα ίδια δεδομένα ανά πάσα στιγμή. Οι μεγαλύτεροι εμπορικοί κολοσσοί κι όχι μόνο ενημερώνουν αυτόματα τις προμήθειές σ' όλες τις αποθήκες τους, τραπεζικοί οργανισμοί μεταφέρουν χρήματα χωρίς ανοχή σε σφάλματα και με τη μέγιστη ασφάλεια, οι χρήστες μιας υπηρεσίας απαιτούν γρήγορη εξυπηρέτηση. Αυτό το λογισμικό, που εξυπηρετεί το στόχο μιας οντότητας και προσφέρει μια υπηρεσία, είναι οι enterprise εφαρμογές.

Η JavaEE είναι μια από τις πλατφόρμες που χρησιμοποιείται για την ανάπτυξη enterprise εφαρμογών. Αποτελείται από πολλές τεχνολογίες οι οποίες συνθέτουν το stack της JavaEE. Από την πρώτη έκδοση της πλατφόρμας μέχρι και σήμερα, έχει αλλάξει και απλοποιηθεί σε μεγάλο βαθμό με αποτέλεσμα η ανάπτυξη εφαρμογών να είναι ευκολότερη από ποτέ.

Λόγω του εύρους και της πληθώρας των τεχνολογιών, δεν είναι εύκολο να αναλύσουμε εις βάθος τα API και πώς αυτά χρησιμοποιούνται. Γι' αυτό το λόγο, η παρούσα εργασία ασχολείται κυρίως με τις πιο συχνά χρησιμοποιούμενες τεχνολογίες παραδίδοντας και την αντίστοιχη ολοκληρωμένη εφαρμογή στο τέλος.

ΠΕΡΙΛΗΨΗ

Το θέμα αυτής της εργασίας είναι η ανάπτυξη μιας web εφαρμογής με χρήση της JavaEE 7.

Στο πρώτο κεφάλαιο γίνεται μια μικρή αναφορά στην ιστορία της Java ως πλατφόρμα προγραμματισμού αλλά και στις εκδόσεις της.

Στο δεύτερο κεφάλαιο μπαίνουμε σταδιακά στον JavaEE κόσμο με εξήγηση του τί είναι οι enterprise εφαρμογές, από τι αποτελούνται και πώς λειτουργούν.

Το τρίτο είναι το τελευταίο από τα εισαγωγικά κεφάλαια καθώς αναφέρει τα specifications της JavaEE 7 και το Java Community Process, το οποίο παρακολουθεί και αναπτύσσει τα specifications.

Με την εισαγωγή του τέταρτου κεφαλαίου, γίνεται η διαφοροποίηση των web applications σε presentation-oriented και service-oriented εφαρμογές και πού χρησιμοποιούνται.

Το πέμπτο κεφάλαιο είναι η εισαγωγή στο πρακτικό μέρος της εργασίας καθώς αναφέρει το JavaServer Faces, το API της JavaEE για δημιουργία του UI της εφαρμογής.

Στο έκτο κεφάλαιο έχουμε μια σύντομη αναφορά στο CDI, το API που βοηθά εξαιρετικά την ελαχιστοποίηση του χρόνου ανάπτυξης αλλά και εισαγωγής αντικειμένων στον κύκλο ζωής άλλων αντικειμένων.

Οι βάσεις δεδομένων είναι αναπόσπαστο μέρος των enterprise εφαρμογών και γι' αυτό το λόγο στο έβδομο κεφάλαιο γίνεται λόγος για το JPA, πώς η JavaEE επικοινωνεί με βάσεις δεδομένων κι εκτελεί CRUD λειτουργίες.

Το business logic μιας εφαρμογής κρύβεται στο όγδοο κεφάλαιο, όπου μέσω των enterprise javabeans βλέπουμε πώς σ' αυτά αναπτύσσεται ο "σημαντικός" κώδικας.

Το δέκατο και τελευταίο κεφάλαιο κάνει μια αναδρομή στο παρελθόν με τα Servlets, μπορεί να ήταν η πρώτη τεχνολογία της JavaEE για δυναμικό περιεχόμενο, τα Servlets όμως χρησιμοποιούνται πολύ ακόμα και τώρα.

ABSTRACT

The topic of this paper is to develop a web application using JavaEE 7.

In the first chapter a small report is made around the Java platform and its versions.

In the second chapter, we're slowly entering the JavaEE domain, what enterprise applications are, what they consist of and how they operate.

The third chapter is the last of the introductory ones since it refers to the platform specifications and the Java Community Process, the people governing and developing the specifications.

With the introduction to the fourth chapter, a differentiation between presentation-oriented and service-oriented applications is made, along with how they're used.

The fifth chapter is the grand entrance to the programming part of the paper, since it talks about JavaServer Faces, the JavaEE API to build the UI of the application.

In the sixth chapter, there's a small reference to the CDI, the API that drastically reduces the development time and injects objects in the lifecycle of others.

Databases are a crucial part of any enterprise application so in the seventh chapter we talk about the JPA API, how JavaEE operates with databases and runs CRUD methods.

The business logic of an application is encapsulated in the eighth chapter, where enterprise javabeans hide the "serious" code.

The tenth chapter makes a flashback to the past via the Servlets, they may have been the first JavaEE technology to build dynamic content, they're still strongly used.

ΕΥΧΑΡΙΣΤΙΕΣ (προαιρετικά)

ΠΕΡΙΕΧΟΜΕΝΑ

Ευρετήριο σχημάτων

Ευρετήριο πινάκων

ΕΙΣΑΓΩΓΗ

Περιλαμβάνει τους στόχους και σκοπούς της Π/Ε καθώς και περιγραφή των κεφαλαίων που ακολουθούν.

ΚΕΦΑΛΑΙΟ 1

ΠΕΡΙ JAVA

ΕΙΣΑΓΩΓΗ

Ξεκινώντας μια εργασία για την ανάπτυξη εφαρμογής με τη Java EE 7, είναι καλό να γίνει μια μικρή εισαγωγή για τη Java γενικότερα. Θα γίνει μια μικρή αναφορά στην ιστορία της, το λόγο που δημιουργήθηκε αλλά και το ότι δεν πρόκειται απλά για μια γλώσσα προγραμματισμού.

ΑΝΑΦΟΡΙΚΑ ΓΙΑ ΤΗ JAVA

Η Java δημιουργήθηκε από τον James Gosling της Sun Microsystems, με σκοπό να γράφεται μια φορά ο πηγαίος κώδικας και να εκτελείται παντού, χωρίς να χρειάζεται εκ νέου μεταγλώττιση. Ενσωματώθηκε το 1995 στη Java Platform της Sun Microsystems και μέσω της Java Virtual Machine, η τυχόν διαφορετική αρχιτεκτονική του συστήματος που εκτελείται το εκάστοτε πρόγραμμα, δεν είναι ανασταλτικός παράγοντας για την εκτέλεσή του.

Είναι γλώσσα υψηλού επιπέδου (high-level), αντικειμενοστραφής (object-oriented), ασφαλής, δυναμική (dynamic) και με αυστηρή σύνταξη. Ο πηγαίος κώδικας γράφεται σε απλά αρχεία κειμένου με κατάληψη .java και μεταγλωττίζεται σε αρχεία .class που περιέχουν το bytecode κώδικα της Java Virtual Machine. Αυτά τα .class αρχεία είναι που δίνουν στη Java το "write once, run anywhere" παράγοντα.

Στην εισαγωγή του κεφαλαίου, έγινε αναφορά ότι η Java δεν είναι απλά μια γλώσσα προγραμματισμού. Πραγματικά, ο συνδυασμός της Java Virtual Machine αλλά και των API που απαρτίζουν τη γλώσσα προγραμματισμού, κάνουν τη Java μια πλατφόρμα προγραμματισμού κι όχι απλά γλώσσα. Η Java λοιπόν, αποτελείται από τέσσερις εκδόσεις της πλατφόρμας της οι οποίες έχουν διαφορετικό πεδίο χρήσης.

JAVA PLATFORMS

Java SE (Standard Edition)

Η Java SE περιέχει τον πυρήνα των API που προσφέρουν τη λειτουργικότητα της γλώσσας. Περιέχει όλους τους προσδιορισμούς των βασικών τύπων και αντικειμένων. Προσδιορίζει επίσης όλες τις υπόλοιπες κλάσεις που χρησιμοποιούνται για πρόσβαση σε βάσεις δεδομένων, χειρισμό αρχείων, δικτύωση, δημιουργία γραφικού περιβάλλοντος (GUI) κ.α. Στόχος της είναι τα desktop συστήματα.

Java EE (Enterprise Edition)

Η Java EE έχει κατασκευαστεί πάνω στην SE, ορίζοντας τα δικά της ανεξάρτητα API. Στόχος της EE είναι το enterprise λογισμικό, εφαρμογές δηλαδή στον εταιρικό τομέα που απαιτείται διαδικτύωση, ασφάλεια κι εύκολο scaling όταν χρειάζεται.

Java ME (Micro Edition)

Ο σκοπός της Java ME είναι οι συσκευές περιορισμένων δυνατοτήτων όπως τα κινητά τηλέφωνα ή γενικότερα φορητές συσκευές. Σ' αυτές τις συσκευές υπάρχει περιορισμός στο υλικό (επεξεργαστής, μνήμη κλπ) επομένως απαιτείται μικρότερο footprint στην πλατφόρμα. Αυτό κάνει η Java ME παρέχοντας ένα υποσύνολο των API της SE, καθώς και κλάσεις που βοηθούν στην ανάπτυξη εφαρμογών για τέτοιες συσκευές.

JavaFX

Η JavaFX δημιουργήθηκε με σκοπό την κατασκευή Rich Internet Applications (RIAs), προσφέροντας μοντέρνο Look-And-Feel, δυνατότητα χρήσης CSS αλλά και API για Drag and Drop ενέργειες, κατασκευή γραφημάτων αλλά και animations. Έχει ενσωματωθεί τώρα πια στη Java SE μιας και προορίζεται για αντικαταστάτης της Swing.

ΕΠΙΛΟΓΟΣ

Σ' αυτό το κεφάλαιο έγινε μια σύντομη εισαγωγή στη Java και είδαμε ότι πρόκειται για πλατφόρμα ανάπτυξης κι όχι απλά για μια γλώσσα προγραμματισμού. Λίγα λόγια για το πως δουλεύει αλλά και το σκοπό για τον οποίο δημιουργήθηκε. Επίσης, έγινε αναφορά στις τέσσερις διαφορετικές πλατφόρμες της Java καθώς και στο πεδίο το οποίο ειδικεύονται. Μια από αυτές τις τέσσερις πλατφόρμες είναι η Java EE, το αντικείμενο που πραγματεύεται αυτή η εργασία.

ΚΕΦΑΛΑΙΟ 2

ΕΙΣΑΓΩΓΗ ΣΤΗ JAVA EE

ΕΙΣΑΓΩΓΗ

Η έλευση της JavaEE έγινε το 1999 για να καλύψει συγκεκριμένες ανάγκες της εποχής. Πολλά έχουν αλλάξει από τότε αλλά ο σκοπός της μένει ο ίδιος. Σ' αυτό το κεφάλαιο θα δούμε για ποιό λόγο υλοποιήθηκε η JavaEE, από τι αποτελούνται οι συγκεκριμένες εφαρμογές καθώς και τι υπηρεσίες προσφέρει η JavaEE στους developers.

ΤΙ ΕΙΝΑΙ ΟΙ ENTERPRISE ΕΦΑΡΜΟΓΕΣ

Τις τελευταίες δύο δεκαετίες, λόγω της παγκοσμιοποίησης και ανάπτυξης του Διαδικτύου, οι εταιρείες μπόρεσαν να επεκταθούν πολύ πιο εύκολα από πριν. Δημιούργησαν παραρτήματα σε άλλα μέρη, μετέφεραν το επιχειρηματικό τους πλάνο στο Διαδίκτυο και κατάφεραν να προσεγγίσουν περισσότερους πελάτες μέσω όλης αυτής της κατάστασης.

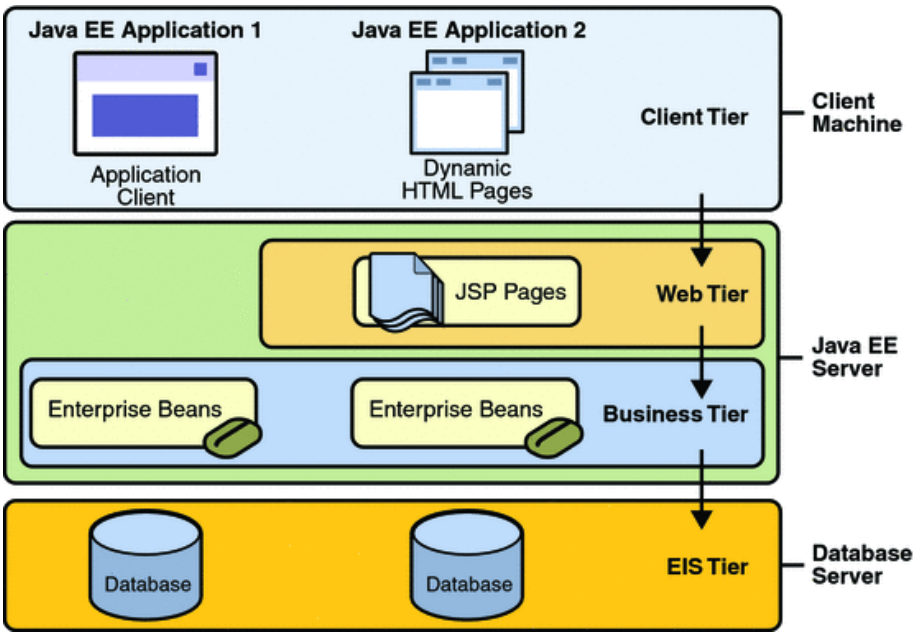
Απαιτείται όμως η δημιουργία εφαρμογών οι οποίες θα διατηρήσουν την επικοινωνία μεταξύ των παραρτημάτων της εταιρείας, των πελατών τους, και θα μπορούν ν' ανταποκριθούν σ' ένα συνεχώς διογκούμενο επιχειρηματικό πλάνο που προσπαθεί να καλύψει περισσότερες ανάγκες και ν' αυξήσει τα κέρδη της εταιρείας. Όλα αυτά ενώ προσπαθεί να κρατήσει τα δεδομένα ασφαλή και την εφαρμογή όσο το δυνατόν περισσότερο επεκτάσιμη γίνεται.

Αυτές είναι οι enterprise εφαρμογές, χωρισμένες σε διάφορα σημεία του κόσμου, ασφαλείς, μεγάλης εμβέλειας και με δυνατότητα να ικανοποιήσουν ένα μεγάλο αριθμό αιτημάτων.

MULTI-TIERED ΕΦΑΡΜΟΓΕΣ

Η λειτουργικότητα μιας enterprise εφαρμογής χωρίζεται σε διαφορετικά επίπεδα (tiers) τα οποία, όπως αναφέρθηκε παραπάνω, μπορεί να είναι εγκατεστημένα σε διαφορετικά φυσικά μηχανήματα αλλά και οπουδήποτε στον κόσμο.

Μια τυπική JavaEE εφαρμογή είναι 3-tier, αποτελείται από το client-tier, το server-tier (ή αλλιώς web-tier) και το enterprise information system(database)-tier.



Σχήμα 1 Java EE 3-tier application

Client tier

Η εφαρμογή-πελάτης (client) είναι αυτή που θα συνδεθεί στο JavaEE server και θα κάνει αιτήσεις, λαμβάνοντας αργότερα τις απαντήσεις από το server. Συνήθως βρίσκονται σε διαφορετικό μηχάνημα από το server και δεν είναι αναγκαίο να είναι Java εφαρμογές. Οι clients μπορούν να είναι προγραμματισμένοι σε οποιαδήποτε γλώσσα μπορεί να συνδεθεί στο server, από applets, JavaFX και Swing εφαρμογές, έως εφαρμογές σε Ruby, Python κλπ. Ακόμα και μια command line εφαρμογή μπορεί να είναι πελάτης, ή ένας browser, όπως κι ένας άλλος server. Οι web browsers συνήθως επικοινωνούν με το web tier (επόμενη ενότητα), ενώ οι GUI standalone εφαρμογές απευθείας με το business tier (επόμενη ενότητα).

Server tier

Το server tier αποτελείται από το web tier και το business tier. Το web tier αναλαμβάνει την επικοινωνία μεταξύ του client και του business tier, λαμβάνοντας δεδομένα από το χρήστη μέσω του client και εμφανίζοντας δυναμικά τα ανάλογα αποτελέσματα. Είναι ανάγκη βέβαια να διατηρεί την κατάσταση των δεδομένων για το εκάστοτε session και προσωρινά ν' αποθηκεύει δεδομένα σε JavaBeans.

Εν συντομία, κάποιες από τις τεχνολογίες που χρησιμοποιούνται στο web tier είναι τα Servlets, τα οποία επεξεργάζονται δυναμικά τις αιτήσεις και δομούν τ' αποτελέσματα, τα JavaServer Pages που εισάγουν δυναμικό περιεχόμενο σε στατικές σελίδες, το JavaServer Faces framework που προσφέρει γραφικό περιβάλλον στη web εφαρμογή, ενώ τα JavaBeans είναι αντικείμενα που αποθηκεύουν προσωρινά δεδομένα στις σελίδες της εφαρμογής. Το business tier αποτελείται από τα συστατικά που επιλύουν το business logic του τομέα της εφαρμογής, είτε πρόκειται για τραπεζικό σύστημα ή για ηλεκτρονικές πωλήσεις

κλπ. Στο business tier υπάρχουν τα enterprise beans που λειτουργούν ως προσωρινή αποθήκη δεδομένων ανάμεσα στον client και τη βάση δεδομένων.

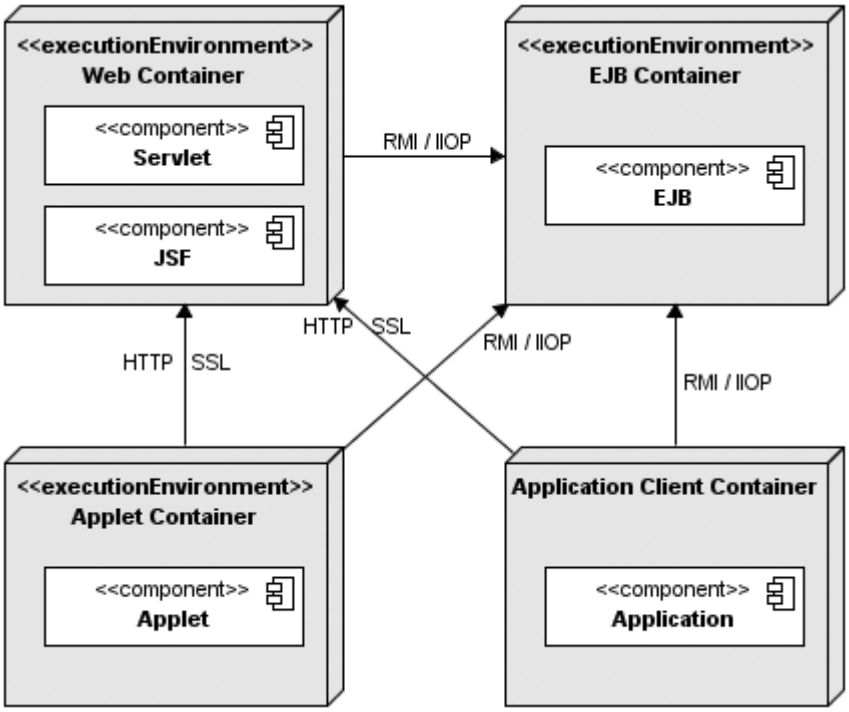
Κάποιες από τις τεχνολογίες που χρησιμοποιούνται στο business tier είναι τα enterprise beans όπως αναφέρθηκε και οι RESTful web services για επικοινωνία μεταξύ client και server.

Enterprise Information System tier

Σ' αυτό το επίπεδο πρακτικά έχουμε τις αποθήκες δεδομένων. Σ' αυτές συγκαταλέγονται οι database servers, τα ERP συστήματα, mainframes ή άλλα legacy συστήματα. Ακόμα κι ένα απλό αρχείο κειμένου μπορεί να θεωρηθεί βάση δεδομένων, αν και βέβαια συγκαταλέγεται στα legacy συστήματα.

ΑΡΧΙΤΕΚΤΟΝΙΚΗ

Η αρχιτεκτονική μιας Java EE εφαρμογής, αποτελείται από μικρότερες οντότητες λογισμικού, οι οποίες αλληλεπιδρούν μεταξύ τους. Αυτές οι οντότητες λογισμικού (components) είναι υπεύθυνες για συγκεκριμένες λειτουργίες κι εκτελούνται στ' αντίστοιχα περιβάλλοντά τους τα οποία ονομάζονται containers. Τα containers αναλαμβάνουν να προσφέρουν έτοιμες υπηρεσίες (services) στα components για τη μεταξύ τους αλληλεπίδραση. Βέβαια τα components (κι άρα τα containers) αντιστοιχούν στο εκάστοτε tier της εφαρμογής. Το παρακάτω σχήμα οπτικοποιεί την αρχιτεκτονική μιας enterprise εφαρμογής.



Σχήμα 2 Αρχιτεκτονική: Containers & Components

COMPONENTS & CONTAINERS

Όπως αναφέρθηκε και παραπάνω, τα containers είναι τα περιβάλλοντα εκτέλεσης του εκάστοτε component, παρέχοντάς τους τις υπηρεσίες που χρειάζονται.

Τέσσερα είναι τα containers που πρέπει να υποστηρίζονται, το applet container εκτελείται στον client μαζί με το αντίστοιχο Java plugin, ενώ παρόμοια το application container εκτελείται κι αυτό στον client, αποτελούμενο από application components με GUI σε Swing ή JavaFX (για Java εφαρμογές), ή οποιοδήποτε άλλο application που έχει πρόσβαση στις υπηρεσίες του Java EE server.

Το web container είναι υπεύθυνο για την εκτέλεση των web components, περιέχει Servlets, JavaServer Faces pages, JavaServer Pages κ.ά. ενώ τους δίνει τη δυνατότητα να επικοινωνήσουν με το web client μέσω HTTP(S) αιτήσεων. Αυτό είναι που προωθεί τις web pages στον web client.

Το ejb container, περιέχει τα enterprise java beans και ρυθμίζει το lifecycle τους, καθώς και υπηρεσίες για ασφάλεια, συγχρονισμό και συναλλαγές.

SERVICES

Στις τελευταίες δύο ενότητες, γίνεται συνεχώς αναφορά στις by default υπηρεσίες που παρέχουν τα containers προς διευκόλυνση του developer. Με αυτόν τον τρόπο θα επικεντρωθεί στην επίλυση του προβλήματος και θ' αγνοήσει προβλήματα όπως η αποστολή email. Ακολουθεί μια σύντομη λίστα με υπηρεσίες που παρέχονται από τη Java EE.

- Persistence: Μέθοδος σύνδεσης ενός αντικειμενοστραφούς μοντέλου και μιας σχεσιακής βάσης δεδομένων.
- Transaction: Μετά από κάθε συναλλαγή με τη βάση δεδομένων, όλες οι εφαρμογές που βλέπουν τα ίδια δεδομένα, βλέπουν τις καινούργιες τιμές των δεδομένων.
- Mail: Αποστολή email μέσω της εκάστοτε εφαρμογής.
- XML & JSON processing: Για όσες εφαρμογές χρειάζεται να επεξεργαστούν XML ή JSON αρχεία.
- Security: Παρέχονται μηχανισμοί ασφάλειας τόσο για την αυθεντικοποίηση των χρηστών, όσο και γι' άλλες υπηρεσίες.

NETWORK PROTOCOLS

Τα πρωτόκολλα που υποστηρίζει η Java EE για κλήση των εκάστοτε components είναι τα HTTP, HTTPS και RMI-IIOP. Το HTTP είναι το πρωτόκολλο λειτουργίας του Web και χρησιμοποιείται σ' όλες τις μοντέρνες εφαρμογές. Το HTTPS είναι συνδυασμός του HTTP με το SSL, ενώ το RMI-IIOP είναι τρόπος επίκλησης απομακρυσμένων αντικειμένων ανεξάρτητα από το πρωτόκολλο επικοινωνίας.

PACKAGING

Όλα αυτά τα components για ν' αναπτυχθούν στο αντίστοιχό τους container, πρέπει να συσκευαστούν σε αρχεία. Στη Java SE για παράδειγμα, όταν γίνεται build μια εφαρμογή, δημιουργείται ένα jar αρχείο που περιέχει τα class αρχεία της εφαρμογής, επιπλέον resources αρχεία (εικόνες, αρχεία κειμένου κλπ) αν υπάρχουν, εξωτερικές βιβλιοθήκες και το deployment descriptor της εφαρμογής. Αντίστοιχα συμβαίνει και στις Java EE εφαρμογές. Χωρίς να γίνει λεπτομερής ανάλυση του packaging των components, έχουμε τα παρακάτω.

Το ejb μέρος της εφαρμογής, συσκευάζεται σ' ένα jar αρχείο με τον προαιρετικό descriptor του ο οποίος έχει όνομα ejb-jar.xml.

Το web μέρος της εφαρμογής, είπαμε ότι περιέχει JSF, JSP, (X)HTML, CSS αρχεία, αυτά δηλαδή που αλληλεπιδρούν με τον client (συνήθως web client). Αυτά τα συστατικά συσκευάζονται σ' ένα war αρχείο, που εκτός από τα παραπάνω στοιχεία περιέχει τον επίσης προαιρετικό deployment descriptor, ο οποίος ονομάζεται απλά web.xml.

Το enterprise μέρος της εφαρμογής (δηλαδή όλα τα παραπάνω), συσκευάζεται σ' ένα ear αρχείο που περιέχει όλα τα συσκευασμένα μέρη (war, ejb jars κλπ). Με αυτόν τον τρόπο, όταν γίνει το deployment της εφαρμογής στον application server, όλα τα components θα γίνουν deployed ταυτόχρονα. Στο ear αρχείο, όπως και στα επιμέρους packages, ο deployment descriptor είναι προαιρετικός κι έχει όνομα application.xml.

ANNOTATIONS AND DEPLOYMENT DESCRIPTORS

Στην προηγούμενη ενότητα για το packaging των διαφόρων components, έγινε αναφορά στους deployment descriptors και τ' ότι είναι προαιρετικοί.

Οι deployment descriptors και τα annotations, είναι metadata που ορίζουν, ρυθμίζουν και συνδέουν τα services κάθε container, με fields, interfaces, methods κλπ των Java κλάσεων. Είναι προαιρετικοί λόγω των επίσης προαιρετικών annotations.

Τα annotations δε βρίσκονται σε εξωτερικό αρχείο (όπως *.xml για τους descriptors) αλλά μέσα στη Java κλάση και συνδέουν το εκάστοτε μέλος της τάξης με την αντίστοιχη υπηρεσία.

Τόσο οι deployment descriptors όσο και τα annotations, είναι προαιρετικά. Φυσικά πρέπει οπωσδήποτε να υπάρχει τουλάχιστον ένα από τα δύο είδη metadata, αλλά μπορούν να χρησιμοποιηθούν και τα δύο μαζί. Ανάλογα το είδος της εφαρμογής, μπορεί να χρησιμοποιηθεί μόνο deployment descriptor ή μόνο annotations. Στην περίπτωση που χρησιμοποιούνται και τα δύο, οι ορισμοί του deployment descriptor έχουν μεγαλύτερη βαρύτητα από τα annotations κι έτσι χρησιμοποιούνται αυτά.

Στο σύγχρονο development προτιμώνται τα annotations. Είναι λογικό καθώς είναι πολύ ευκολότερο να χρησιμοποιήσουμε μόνο Java, αντί Java και XML. Επιπλέον, τα metadata βρίσκονται στο ίδιο αρχείο με τον κώδικα, αυτή η κατάσταση διευκολύνει την ανάλυση, το debugging και γενικά όλη τη διαδικασία ανάπτυξης.

ΕΠΙΛΟΓΟΣ

Στο δεύτερο κεφάλαιο, έγινε γνωστό ότι οι enterprise εφαρμογές στοχεύουν στον επιχειρηματικό τομέα και την επίλυση ενός συγκεκριμένου προβλήματος, για παράδειγμα η μεταφορά χρημάτων σ' ένα τραπεζικό σύστημα, οι online πωλήσεις μιας επιχείρησης ή ακόμα κι η κράτηση θέσεων σ' ένα αεροδρόμιο.

Είδαμε ότι μια enterprise εφαρμογή χωρίζεται στα τρία (συνήθως) επίπεδα του πελάτη, διακομιστή και πληροφοριακού συστήματος. Η JavaEE επικεντρώνεται στο επίπεδο του διακομιστή, απ' τη στιγμή που ο πελάτης μπορεί να έχει υλοποιηθεί με μη-Java τεχνολογίες και το πληροφοριακό σύστημα είναι ξεχωριστός τομέας από μόνο του.

Έγινε μια γρήγορη αναφορά στην αρχιτεκτονική των enterprise εφαρμογών και στα στοιχεία απ' τα οποία αυτή αποτελείται. Τα στοιχεία αυτά εκτελούνται μέσα στα δικά τους περιβάλλοντα, με τα δεύτερα να παρέχουν έτοιμες υπηρεσίες ώστε να μειωθεί ο χρόνος ανάπτυξης της εφαρμογής.

Είδαμε επίσης πώς αυτά τα επιμέρους στοιχεία συσκευάζονται για ν' αναπτυχθούν στα εκάστοτε περιβάλλοντά τους. Ακολουθείται μια συγκεκριμένη διαδικασία συσκευασίας των στοιχείων, τα οποία σε συνεργασία με τα μεταδεδομένα τους κάνουν χρήση των διαφόρων υπηρεσιών.

Στο επόμενο κεφάλαιο, θα δούμε με ποιον τρόπο όλες αυτές οι υπηρεσίες, περιβάλλοντα, ενημερώσεις, διαδικασίες κλπ εισάγονται στη JavaEE και ποιοί είναι υπεύθυνοι γι' αυτό.

ΚΕΦΑΛΑΙΟ 3

SPECIFICATIONS OVERVIEW

ΕΙΣΑΓΩΓΗ

Η JavaEE, όπως άλλωστε και κάθε άλλη έκδοση της Java, αποτελείται από specifications. Αυτά είναι μεμονωμένες τεχνολογίες οι οποίες επιλύουν ένα συγκεκριμένο πρόβλημα, για παράδειγμα επεξεργασία JSON αρχείων ή ένας νέος τρόπος για το deployment της εφαρμογής στο server.

Παρακάτω θα γίνει μια σύντομη αναφορά στην ιστορία της JavaEE και τα specifications που περιλάμβανε, θα δούμε με ποιόν τρόπο εισάγονται νέες τεχνολογίες καθώς και μια επιγραμματική αναφορά σ' αυτές.

BRIEF HISTORY

Η επίσημη ανακοίνωση της τότε JavaEE, έγινε το 1998 και ονομαζόταν Java Professional Edition (JPE). Η πρώτη κοινοποίηση της πλατφόρμας από τη Sun, έγινε στα τέλη του 1999 με ονομασία J2EE 1.2 και περιείχε 10 specifications για να καλύψει τις τότε enterprise ανάγκες.

Η J2EE 1.3 που κοινοποιήθηκε το 2001, αναπτύχθηκε από τη Java Community Process κι έτσι συμβαίνει έκτοτε. Για τη J2EE 1.4 θεωρούσαν πως θα είχε μεγάλη εμπλοκή στην αγορά, αλλά όπως φάνηκε ήταν αρκετά πολύπλοκη και δύσχρηστη.

Η Java EE 5 όμως ήταν ορόσημο για την πλατφόρμα αφού επανέφερε το POJO προγραμματιστικό μοντέλο, εισήγαγε τα annotations και οι deployments descriptors έγιναν προαιρετικοί.

Στο τέλος του 2009, η Java EE 6 εισήλθε δυναμικά με πολλές βελτιώσεις και καινούργιες τεχνολογίες. Έφερε ολική υποστήριξη στο development με annotations και specification για RESTful υποστήριξη, αλλά και μεγάλου μεγέθους revision στα ήδη υπάρχοντα specifications. Πολλά από αυτά χαρακτηρίστηκαν για διαγραφή λόγω παλαιότητας.

JAVA COMMUNITY PROCESS (JCP)

Ένας ανοιχτός οργανισμός, ιδρυμένος το 1998 από τη Sun, η JCP έχει μέλη απ' όλο τον κόσμο. Εταιρείες, οργανισμοί, πανεπιστήμια αλλά και ιδιώτες παίρνουν μέρος στη διαδικασία εξέλιξης της Java ως πλατφόρμα.

Όταν κάποιος θέλει να εισάγει ή ν' αλλάξει ένα feature της πλατφόρμας, υποβάλλει ένα Java Specification Request (JSR). Το συγκεκριμένο άτομο αυτόματα χαρακτηρίζεται ως Specification Lead του συγκεκριμένου JSR και μαζί με το JSR υποβάλλει και θεμελιώδη συστατικά του.

Αν το JSR εγκριθεί από την ορισμένη επιτροπή (Executive Commitee - EC), δίνεται στην κοινότητα για υλοποίηση.

EE 7 SPECIFICATIONS

Κάθε έκδοση της Java, είναι από μόνη της ένα JSR. Η Java EE 7 δεν αποτελεί εξαίρεση, με τη διαφορά ότι είναι ένα JSR που περιέχει άλλα JSR. Τεχνολογία δηλαδή που απαρτίζεται από άλλες τεχνολογίες. Ακολουθούν πίνακες[1] με τα specifications της Java EE 7.

Πίνακας 1Web Services Specifications

Specification	Version	JSR
JAX-WS	2.2a	224
JAXB	2.2	222
Web Services	1.3	109
Web Services Metadata	2.1	181
JAX-RS	2.0	339
JSON-P		

Πίνακας 2: Web Specifications

Specification	Version	JSR
JSF	2.2	344
JSP	2.3	245
Debugging Support for Other Languages	1.0	45
JSTL	1.2	52
Servlet	3.1	340
WebSocket	1.0	356
Expression Language	3.0	341

[1] Οι πίνακες αυτούσιοι από τις σελίδες 18 και 19 του βιβλίου Beginning Java EE 7, της βιβλιογραφίας

Πίνακας 3: Enterprise Specifications

Specification	Version	JSR
EJB	3.2	345
Interceptors	1.2	318
JavaMail	1.5	919
JCA	1.7	322
JMS	2.0	343
JPA	2.1	338
JTA	1.2	907

Πίνακας 4: Management, Security and Other Specifications

Specification	Version	JSR
JACC	1.4	115
Bean Validation	1.1	349
Contexts and Dependency Injection	1.1	346
Dependency Injection for Java	1.0	330
Batch	1.0	352
Concurrency Utilities for Java EE	1.0	236
Java EE Management	1.1	77
Java Authentication Service Provider Interface for Containers	1.0	196

Ο παρακάτω πίνακας περιέχει τα specifications της Java SE 7, τα οποία υποστηρίζει φυσικά κι η EE καθώς είδαμε ότι η EE είναι κατασκευασμένη πάνω στην SE.

Πίνακας 5: Related Technologies in Java SE 7

Specification	Version	JSR
Common Annotations	1.2	250
JDBC	4.0	221
JNDI	1.2	
JAXP	1.3	206
StAX	1.0	173
JAAS	1.0	
JMX	1.2	3
JAXB	2.2	222
JAF	1.1	925
SAAJ	1.3	

WEB PROFILE 7 SPECIFICATIONS

Τα profiles δημιουργήθηκαν για να περιορίσουν το μέγεθος της πλατφόρμας ανάλογα με τις ανάγκες της κάθε εφαρμογής. Η Java EE 7 με τα 31 specifications της, ίσως είναι υπερβολικά "πολλή" για κάποιες εφαρμογές.

Το Web Profile 7 είναι το μοναδικό profile που προσφέρει η Java EE 7 για την ανάπτυξη web εφαρμογών. Φυσικά αργότερα ίσως δημιουργηθούν άλλα profiles για να καλύψουν διαφορετικές ανάγκες. Ακολουθεί πίνακας[2] με τα specifications του Web Profile 7 Specification.

Πίνακας 6: Web Profile 7 Specifications

JSF	JSP	JSTL	Servlet
WebSocket	Expression Language	EJB Lite	JPA
JTA	Bean Validation	Managed Beans	Interceptors
Contexts and Dependency Injection	Dependency Injection for Java	Debugging Support for Other Languages	JAX-RS
JSON-P			

[2] Ο πίνακας είναι αυτούσιος των σελίδων 20 και 21 του βιβλίου Beginning Java EE 7 της βιβλιογραφίας

ΕΠΙΛΟΓΟΣ

Η JavaEE άλλαξε πολύ από την πρώτη της έκδοση τη δεκαετία του '90. Από μία δύσχρηστη και απαιτητική πλατφόρμα, κατάφερε να βρεθεί στην πρώτη γραμμή του μετώπου για την ανάπτυξη εφαρμογών στον enterprise κόσμο.

Η Java Community Process, έχει εισάγει πάνω από 30 τεχνολογίες κατάλληλες για τις ανάγκες των enterprise εφαρμογών και συνεχώς προσθέτει νέες ή αναβαθμίζει παλαιότερες.

Σ' αυτό το σημείο τα specifications της JavaEE 7 αναφέρθηκαν επιγραμματικά, στα επόμενα κεφάλαια θα γίνει καλύτερη ανάλυση στα βασικότερα από αυτά, αφού όμως δούμε πρώτα τους τύπους των Web εφαρμογών..

ΚΕΦΑΛΑΙΟ 4

WEB APPLICATIONS

ΕΙΣΑΓΩΓΗ

Οι Web εφαρμογές της JavaEE δεν έχουν καμία διαφορά με τις Web εφαρμογές που υλοποιούνται σε άλλες γλώσσες προγραμματισμού.

Η εφαρμογή εκτελείται σ' έναν application server κι έχει πρόσβαση σε κάποια βάση δεδομένων, παρέχοντας δεδομένα στο χρήστη. Ο χρήστης μπορεί να είναι είτε κάποιος άνθρωπος είτε κάποιο μηχάνημα/υπηρεσία, η διακίνηση των δεδομένων άλλωστε δεν είναι μόνο μεταξύ μηχανής-ανθρώπου αλλά και μεταξύ μηχανής-μηχανής.

Παρ' ότι σ' αυτό το κεφάλαιο δε θα γίνει εκτενής αναφορά στους τύπους εφαρμογών παρά μια μικρή εισαγωγή, ανάλογα τους εμπλεκόμενους στη διακίνηση των δεδομένων, γίνεται η διάκριση των Web εφαρμογών σε Presentation-oriented και Service-oriented εφαρμογές.

PRESENTATION-ORIENTED WEB APPLICATIONS

Οι Presentation-oriented web εφαρμογές είναι για την περίπτωση της διακίνησης δεδομένων μεταξύ ανθρώπων και μηχανής. Δημιουργούν δυναμικό περιεχόμενο και το παρουσιάζουν στο χρήστη μέσω μιας markup language (HTML, XHTML ή XML), συνήθως μέσω ενός web browser.

Οι τεχνολογίες που χρησιμοποιούνται στις Presentation-oriented εφαρμογές για να παρουσιάσουν το περιεχόμενο στο χρήστη είναι οι: JavaServer Faces, JavaServer Pages, Facelets, Expression Language και Servlets.

SERVICE-ORIENTED WEB APPLICATIONS

Αντίστοιχα, οι Service-oriented web εφαρμογές εμπλέκονται στη διακίνηση δεδομένων μεταξύ μηχανών. Μια Presentation-oriented εφαρμογή X, θα χρησιμοποιήσει ένα service της Service-oriented εφαρμογής Y για να λάβει δεδομένα και να τα παρουσιάσει στο χρήστη ή προωθήσει αλλού.

Οι τεχνολογίες για την ανάπτυξη Service-oriented web εφαρμογών είναι οι JAX-WS και JAX-RS, η πρώτη χρησιμοποιεί XML μηνύματα ενώ η δεύτερη ακολουθεί τη REST μεθοδολογία.

Αξίζει ν' αναφερθεί ότι ενώ η Servlet τεχνολογία μπορεί να χρησιμοποιηθεί για να παρουσιάσει δεδομένα στο χρήστη, είναι καταλληλότερη για Service-oriented εφαρμογές αφού ένα service μπορεί να υλοποιηθεί ως Servlet.

ΕΠΙΛΟΓΟΣ

Το κεφάλαιο αν και μικρό σε έκταση, έκανε τη διάκριση ανάμεσα στους δύο τύπους web εφαρμογών, Presentation-oriented και Service-oriented εφαρμογές.

Έγινε γνωστό ότι οι πρώτες είναι κατάλληλες για να παρουσιάζουν περιεχόμενο στον άνθρωπο μέσω ενός web browser, ενώ οι δεύτερες ως πύλη διακίνησης δεδομένων μεταξύ εφαρμογών.

Η επιγραμματική αναφορά στις εκάστοτε τεχνολογίες που αποτελούν τους τύπους των web εφαρμογών, ήταν αναγκαία για να δούμε τι θ' ακολουθήσει στα επόμενα κεφάλαια.

Βαθύτερη ανάλυση με κώδικα και αναφορά στην εφαρμογή εκπαίδευσης on-demand που αναπτύσσω (ως το πρακτικό μέρος αυτής της πτυχιακής), ξεκινάει από το επόμενο κεφάλαιο και τη JavaServer Faces τεχνολογία.

ΚΕΦΑΛΑΙΟ 5

JAVASERVER FACES

ΕΙΣΑΓΩΓΗ

Όπως είναι λογικό, σε μια web εφαρμογή θέλουμε να παρουσιάσουμε στο χρήστη δεδομένα που λαμβάνονται από το back-end. Η χρήση ενός user interface είναι δεδομένη, ο τύπος του user interface (επομένως του client) όμως ποικίλει.

Είναι δυνατόν να έχουμε μια desktop εφαρμογή με γραφικό περιβάλλον, μια web εφαρμογή που εκτελείται στον browser, μια mobile εφαρμογή σε αντίστοιχη mobile συσκευή, ακόμα και μια εφαρμογή σε command line που δίνει στο χρήστη ένα υποτυπώδες user interface.

Το JavaServer Faces (JSF) είναι server-side framework της JavaEE και εντάχθηκε σ' αυτήν το 2004. Παρέχει API για την αντιστοίχιση components σε tags, χειρισμό γεγονότων και μετατροπή δεδομένων σε άλλες μορφές, ορισμό του navigation μεταξύ των σελίδων της εφαρμογής κ.ά. Τα tags που αναφέρθηκαν νωρίτερα, χρησιμοποιούνται για να τοποθετήσουμε τα εκάστοτε components μες στις σελίδες και να τα συνδέσουμε με server-side αντικείμενα (δυναμικό περιεχόμενο).

ΔΟΜΗ ΜΙΑΣ JSF ΕΦΑΡΜΟΓΗΣ

Τυπικά μια JSF εφαρμογή αποτελείται από τις ιστοσελίδες της εφαρμογής με τα tags των αντικειμένων που θα συνθέσουν την εκάστοτε σελίδα, τα Managed Beans που χρησιμοποιούνται για να δημιουργήσουν δυναμικά τα components της σελίδας και να ορίσουν τις ιδιότητες της σελίδας, καθώς κι έναν web deployment descriptor (web.xml).

XHTML

Για την κατασκευή των ιστοσελίδων η HTML είναι η κατεξοχήν markup γλώσσα του web, το JSF όμως προτιμά την XHTML. Η XHTML ακολουθεί αυστηρή δόμηση μιας και βασίζεται στην XML, παρέχοντας παράλληλα επικύρωση της δομής του εγγράφου μέσω σχετικών XML εργαλείων (XSL, XSLT κλπ).

Facelets

Αρχικά, ο τρόπος δήλωσης της σελίδας στο JSF ήταν με τη γλώσσα JavaServer Pages (JSP). Οι ανάγκες των web εφαρμογών όμως άλλαξαν και το JSP έγινε δυσκίνητο. Αναπτύχθηκε λοιπόν η γλώσσα Facelets και τώρα πια είναι η προτιμώμενη page declaration language στο JSF. Εισάγει επιπλέον βιβλιοθήκες με tags εμπλουτίζοντας το περιεχόμενο των σελίδων.

Backing Beans

Αν και αναφέρθηκαν νωρίτερα ως managed beans, σωστότερος όρος είναι backing beans. Τα backing beans είναι ο συνδετικός κρίκος μεταξύ της σελίδας και

του back-end. Αν θέλουμε να δημιουργήσουμε δυναμικά ένα component, ή να φέρουμε δεδομένα για εμφάνιση ή να μετακινηθούμε μεταξύ ιστοσελίδων, θα χρησιμοποιήσουμε ένα backing bean με την αντίστοιχη λειτουργικότητα.

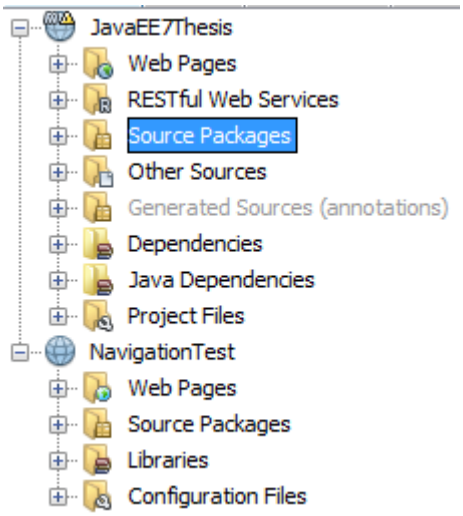
Expression Language

Η Expression Language (EL) είναι η γλώσσα που συνδέει τη JSF σελίδα με τα properties ενός backing bean. Μέσω της EL μπορούμε να εμφανίσουμε τα properties ενός backing bean στη σελίδα, να εκτελέσουμε μεθόδους αυτού όπως για παράδειγμα μετακίνηση σε άλλη σελίδα, εύρεση κι επιστροφή της συνολικής χρηματικής τιμής κάποιων αντικειμένων αλλά και κλήσεις σε βάσεις δεδομένων μέσω των Enterprise JavaBeans (για τα EJB σε επόμενο κεφάλαιο).

Ακολουθεί παράδειγμα με κώδικα κι εξήγηση των παραπάνω τεχνολογιών.

ΠΑΡΑΔΕΙΓΜΑ ΚΙ ΕΞΗΓΗΣΗ ΤΩΝ ΤΕΧΝΟΛΟΓΙΩΝ

Όπως είπαμε, τα βασικά στοιχεία μιας JSF εφαρμογής είναι οι ιστοσελίδες, ο Java κώδικας και ο web deployment descriptor. Στο Παράρτημα 1 παρατίθεται ο συνολικός κώδικας των αρχείων που θ' αναφερθούν παρακάτω, εδώ γίνεται αναφορά σε συγκεκριμένα σημεία για καλύτερη κατανόηση.



Σχήμα 3 - JSF project structure

Στο Σχήμα 3 βλέπουμε δύο JSF project στο Netbeans IDE. Το JavaEE7Thesis είναι η συμπληρωματική εφαρμογή της πτυχιακής ενώ το NavigationTest ένα δοκιμαστικό πρόγραμμα. Το πρώτο είναι κατασκευασμένο με Maven (Παράρτημα 3) ενώ το δεύτερο με Ant και Ivy. Τα build tools όπως και τα διάφορα IDE αλλάζουν σε κάποιο βαθμό τη δομή του project, παρατηρούμε όμως τις βασικές ομοιότητες που μας ενδιαφέρουν.

Παρατηρούμε ότι και στα δύο project υπάρχει κόμβος Web Pages και Source Packages. Στον πρώτο όπως είναι λογικό βρίσκονται οι σελίδες της εφαρμογής και στο δεύτερο όλος ο Java κώδικας.

Login.xhtml

Το login.xhtml ξεκινά με τα namespaces που υποστηρίζει το JSF και Facelets.

```
xmlns:h="http://xmlns.jcp.org/jsf/html
```

Η h σήμανση μας δίνει πρόσβαση σ' όλα τα βασικά components του JSF τα οποία θα μετατραπούν σε HTML αντικείμενα κατά το rendering της σελίδας.

```
<h:inputText id="j_username" value="#{authenticationJSFBean.username}"/>
```

Το παραπάνω snippet κώδικα έχει τη βασική λειτουργικότητα των JSF τεχνολογιών. Το tag h:inputText θα μετατραπεί σε HTML στοιχείο με id "j_username" και θα εμφανίσει ένα πλαίσιο για να εισάγει ο χρήστης κείμενο. Το #{authenticationJSFBean.username} συνδέει την τιμή αυτού του inputText μ' ένα String property με όνομα username στο backing bean.

```
<h:commandButton id="login" action="#{authenticationJSFBean.login()}"
value="Login"/>
```

Αυτό το tag εισάγει ένα HTML button τύπου Submit για εκτέλεση της φόρμας. Στο action attribute βλέπουμε πως κάνει κλήση της μεθόδου login() από το backing bean ενώ το value attribute είναι απλά ο τίτλος που εμφανίζει το κουμπί. Αν στο action έχουμε κάποιο String, το commandButton κάνει navigate στην σελίδα που ορίζει το String.

Το namespace ui εισάγει τα templates που υποστηρίζει το Facelets, δίνοντάς μας τη δυνατότητα να έχουμε μία ομοιογενή εφαρμογή οι ιστοσελίδες της οποίας βασίζονται σ' ένα αρχικό πρότυπο.

```
<div id="top">
            <ui:insert name="top">Top</ui:insert>
        </div>
        <div>
            <div id="left">
                <ui:insert name="left">Left</ui:insert>
            </div>

         </div>

</div>
```

Αν δούμε το template basicTemplate.xhtml στο οποίο βασίζεται η login σελίδα, βρίσκουμε σημάνσεις div οι οποίες χρησιμοποιούνται ως placeholders για να δημιουργήσουν τα πλαίσια header, left, right και content.

Μιας και το login.xhtml κάνει χρήση του συγκεκριμένου template, με το ui namespace μπορούμε να ορίσουμε αυτά τα placeholders και να εισάγουμε το δικό μας περιεχόμενο όπως φαίνεται παρακάτω

```
<ui:define name="top">
    <h1>Please enter your credentials</h1>
```

```
</ui:define>
```

AuthenticationController.java

Όπως αναμέναμε, το backing bean έχει ένα property username και μια μέθοδο login().

```
private String username;

public String login() {
    //Processing
    if (…) {
        …
        return "users/main.xhtml";
    } else {
        …
        return "loginerror";
    }
}
```

Τα properties για να χρησιμοποιηθούν με την EL πρέπει οπωσδήποτε να έχουν μια getProperty() (πχ. getUsername()) μέθοδο, αλλιώς δεν είναι ορατά. Αυτό αυτομάτως σημαίνει ότι δε δίνουμε πρόσβαση μέσω της EL σ' όλα τα αντικείμενα του backing bean, μόνο σ' αυτά που μας ενδιαφέρουν.

Η μέθοδος login() τώρα, βλέπουμε πως επιστρέφει ένα String. Αν ο χρήστης έχει πιστοποιηθεί με τα στοιχεία του επιστρέφεται το String main.xhtml, αλλιώς επιστρέφεται το loginerror. Και τα δύο String είναι ιστοσελίδες στις οποίες θα γίνει navigate μέσω του commandButton που καλεί τη μέθοδο.

Μέχρι στιγμής δεν έχουμε κάτι που να ορίζει το backing bean ή να εξηγεί τι είναι αυτό. Πραγματικά, από τον κώδικα της κλάσης καταλαβαίνουμε πως είναι ένα POJO (Plain Old Java Object) με τα properties και τις μεθόδους του. Αυτό που κάνει τη διαφορά και επιτρέπει το JSF να χρησιμοποιήσει ένα POJO ως backing bean, είναι το annotation @Named.

Με τη χρήση του @Named annotation, η διαχείριση και ο κύκλος ζωής του αντικειμένου περνούν στο container. Το container θα δημιουργήσει το backing bean όταν μεταβούμε στην εκάστοτε ιστοσελίδα και θα το κρατήσει εν ζωή αναλόγως του scope του (request, session κλπ).

ΕΠΙΛΟΓΟΣ

Αν και δεν αναπτύχθηκε πολύ αυτό το κεφάλαιο, δόθηκε μια βασική ιδέα για το JSF και τις τεχνολογίες που το συμπληρώνουν. Είδαμε πως οργανώνονται τα JSF αντικείμενα σε μια ιστοσελίδα και πως γίνεται η σύνδεση μιας σελίδας μ' ένα backing bean δια μέσου της EL.

Στο κεφάλαιο που ακολουθεί θα δούμε τι είναι το Context and Dependency Injection και πώς τα beans χρησιμοποιούνται στο life cycle πολλών σελίδων.

ΚΕΦΑΛΑΙΟ 6

CONTEXT AND DEPENDENCY INJECTION

ΕΙΣΑΓΩΓΗ

Το Context and Dependency Injection (CDI) είναι ένα από τα API που αναβάθμισε και απλούστευσε σε μεγάλο βαθμό την ανάπτυξη των enterprise εφαρμογών. Βοηθά στο να συνδέσει σωστά κι αποτελεσματικά το web tier με το business tier, προσφέροντας στον προγραμματιστή έναν εύκολο τρόπο να χρησιμοποιήσει πρακτικά οποιαδήποτε Java κλάση στον κύκλο ζωής μιας σελίδας.

ΑΝΑΦΟΡΙΚΑ ΓΙΑ ΤΟ CDI

Η ιδέα του CDI είναι να συνδέσει εύκολα κι απλά το JSF μέρος της εφαρμογής με τα διάφορα enterprise beans που υλοποιούν το business logic. Αυτά τα "beans" δεν είναι τίποτα παραπάνω από αντικείμενα (Java κλάσεις) τα οποία ακολουθούν συγκεκριμένους κανόνες όσον αφορά τα properties, μεθόδους και interfaces που υλοποιούν.

Αν και υπάρχουν διάφορες παραλλαγές ενός bean (enterprise, pojo, managed), όλα ακολουθούν συγκεκριμένους κανόνες ανάλογα με τον τομέα στον οποίο χρησιμοποιούνται.

Ο κύκλος ζωής μιας σελίδας ορίζεται από το χρήστη και πώς αυτός αλληλεπιδρά με την εφαρμογή. Απ' τη στιγμή που θα γίνει κλήση στο server για να φορτώσει μια σελίδα, έχουμε ένα request. Το CDI bean που θα συνδεθεί με τη συγκεκριμένη λειτουργία της σελίδας, χρειάζεται μόνο request scope (θέαση) αφού έχουμε μία κλήση.

Για παράδειγμα, όταν ο χρήστης επιλέξει την αποσύνδεση του λογαριασμού του (logout) από την εφαρμογή, γίνεται μία κλήση στο server για τερματισμό της σύνδεσης. Άρα το CDI bean που θα μεταφέρει τα στοιχεία του χρήστη γι' αποσύνδεση, θα έχει διάρκεια ζωής για ένα request.

Στην περίπτωση των πολλαπλών κλήσεων, επειδή ο χρήστης συμπληρώνει μια φόρμα που εκτείνεται σε δύο ή και παραπάνω σελίδες, το scope του bean που αποθηκεύει τα στοιχεία θα πρέπει να είναι ευρύτερο, view, session ή application.

ΤΡΟΠΟΣ ΧΡΗΣΗΣ ΤΟΥ CDI

Είναι εξαιρετικά απλός ο τρόπος που χρησιμοποιείται ένα CDI bean και συνήθως δεν απαρτίζεται από τίποτα παραπάνω από δύο annotations (δεδομένου όμως ότι ακολουθεί τους εκάστοτε κανόνες των properties, methods και interfaces).

Προτού όμως παρατεθούν αποσπάσματα κώδικα από την εφαρμογή που υλοποίησα, πρέπει να γίνει γνωστό το business logic της συγκεκριμένης περίπτωσης που είναι η προσθήκη προϊόντων σε καλάθι αγορών από το χρήστη.

Ο χρήστης για παράδειγμα βλέπει τα διαθέσιμα προϊόντα (μαθήματα στη συγκεκριμένη εφαρμογή), διαβάζει ίσως τα περιεχόμενα του καθενός κι αν κάποιο τον ενδιαφέρει το εισάγει στο καλάθι με το ανάλογο κουμπί.

Η όλη διαδικασία περιλαμβάνει πολλές μεταβάσεις από σελίδα σε σελίδα κι άρα πολλές κλήσεις προς το server. Το CDI bean που είναι το καλάθι αγορών, θα πρέπει να διατηρεί τα περιεχόμενα του χρήστη για πολλές αλληλεπιδράσεις της εφαρμογής. Επομένως κάθε φορά που ο χρήστης επιλέγει "Add" για να εισάγει ένα μάθημα στο καλάθι αγορών, δε μπορούμε να χρησιμοποιούμε το keyword "new" για να δημιουργήσουμε ένα καλάθι αγορών, σε κάθε σελίδα θα έχουμε διαφορετικό αντικείμενο που ναι μεν θα εισάγει μαθήματα στο συγκεκριμένο καλάθι, θα καταστρέφεται δε κάθε φορά που θ' αλλάζει ο χρήστης σελίδα. Η χρήση λοιπόν ενός μόνο "καθολικού" αντικειμένου που θα καλείται όποτε χρειάζεται, είναι απαραίτητη.

Ας δούμε λοιπόν συγκεκριμένα σημεία της κλάσης Shopping Cart (ολόκληρος ο κώδικας της κλάσης περιλαμβάνεται στο Παράρτημα 2).

```
@Named("shoppingCart")
@SessionScoped
public class ShoppingCart implements Serializable {...
```

Το @Named ορίζει ότι αυτή η κλάση είναι CDI bean κι οπουδήποτε στον κώδικα της εφαρμογής μπορούμε ν' αναφερόμαστε σ' αυτήν ως το String που ορίζουμε στην παρένθεση. Το String στην παρένθεση δεν είναι απαραίτητο και μάλιστα θα μπορούσαμε να τ' ονομάσουμε όπως θέλουμε, για παράδειγμα "kala8iAgorwn" αντί για "shoppingCart". Απλώς αν παραλείπεται, το CDI bean αναφέρεται ως το όνομα της κλάσης με πεζό το πρώτο γράμμα.

Το @SessionScoped ορίζει ότι η διάρκεια ζωής του CDI bean είναι γι' αρκετές κλήσεις προς το server κατά τη διάρκεια της σύνδεσης του κάθε χρήστη με την εφαρμογή. Αυτό φυσικά σημαίνει ότι ο κάθε χρήστης έχει το δικό του instance του αντικειμένου, δε μοιράζονται ένα μοναδικό.

Τα properties της κλάσης είναι private κι έχουν τις ανάλογες getter/setter μεθόδους, ο δομητής της κλάσης είναι ο default κι η κλάση υλοποιεί το Serializable interface, άρα πληροί τις προϋποθέσεις ως CDI bean.

Η συγκεκριμένη κλάση μπορεί να χρησιμοποιηθεί σε πάνω από μία περιπτώσεις, στη δική μου εφαρμογή χρησιμοποιείται στη σελίδα που εμφανίζονται όλα τα μαθήματα κι ο χρήστης εισάγει όσα θέλει στο καλάθι του, αλλά και στην σελίδα πληρωμής που ο χρήστης εγγράφεται στα μαθήματα και πληρώνει γι' αυτά.

Τα JSF backing beans των συγκεκριμένων δύο σελίδων, όπως είπαμε δε θα χρησιμοποιήσουν το keyword "new" στο δομητή τους για να δημιουργήσουν το shoppingCart. Αντ' αυτού θα κάνουν

```
@Inject
```

```
private ShoppingCart shoppingCart;
```

ως δικό τους property και ανάλογα θα χρησιμοποιήσουν τις μεθόδους του. Για παράδειγμα, αν θέλουμε να διαγράψουμε τα περιεχόμενα του καλαθιού από την checkout σελίδα, το checkoutBean εκτελεί την παρακάτω μέθοδο:

```
private void clearShoppingCart() {
    shoppingCart.clearCart();
}
```

ΕΠΙΛΟΓΟΣ

Με την εισαγωγή του CDI στη JavaEE 6, απλουστεύθηκε σε μεγάλο βαθμό η σύνδεση των JSF views και του business logic της εφαρμογής. Είναι όντως δυνατό να μη χρησιμοποιήσουμε τη CDI μεθοδολογία για την ανάπτυξη μιας εφαρμογής, δυσχεραίνουμε κατά πολύ όμως τη θέση μας αφού απλά και μόνο χρησιμοποιώντας μερικά annotations, έχουμε απλούστατη σύνδεση αντικειμένων χωρίς ν' ανησυχούμε για το lifecycle τους.

Στο επόμενο κεφάλαιο, θα γίνει αναφορά σε οντότητες (entities) και πως η JavaEE διαχειρίζεται τους πίνακες μιας βάσης δεδομένων αλλά και τις σχέσεις μεταξύ αυτών. Το JPA είναι ένα ακόμα μεγάλο API που βοήθησε σε μεγάλο βαθμό τη γρήγορη ανάπτυξη εφαρμογών με τη JavaEE.

ΚΕΦΑΛΑΙΟ 7

JAVA PERSISTENCE API

ΕΙΣΑΓΩΓΗ

Στο δεύτερο κεφάλαιο της εργασίας, είδαμε τα μέρη από τα οποία αποτελείται μια enterprise εφαρμογή και δη μια JavaEE web εφαρμογή. Είδαμε επίσης ότι η JavaEE επικεντρώνεται στο web tier, αναλαμβάνοντας ουσιαστικά να επεξεργαστεί δεδομένα από τις βάσεις δεδομένων και να τα παρουσιάσει στον client. Για να εκτελέσει τις CRUD λειτουργίες όμως με τις βάσεις δεδομένων (που στην πλειονότητά τους είναι RDBMS), πρέπει να υπάρχει ο κατάλληλος μηχανισμός μετάφρασης των σχέσεων μεταξύ των πινάκων μιας αντικειμενοσχεσιακής βάσης δεδομένων και της αντικειμενοστρέφειας στην οποία βασίζεται η Java.

Η ΕΝΝΟΙΑ ΤΗΣ ΟΝΤΟΤΗΤΑΣ

Όπως γνωρίζουμε, σ' ένα RDBMS τα δεδομένα οργανώνονται σε πίνακες (Υπάλληλοι, Τμήμα κλπ) και κάθε πίνακας περιέχει πολλές εγγραφές της εκάστοτε οντότητας (εδώ ο όρος "οντότητα" αναφέρεται στις οντότητες των RDBMS). Οι σχέσεις μεταξύ των οντοτήτων ορίζονται ως ξένα πεδία στον πίνακα μιας άλλης, ή ως ξένα πεδία σε ενδιάμεσους πίνακες χτίζοντας τις γνωστές ένα-προς-πολλά κλπ σχέσεις των οντοτήτων, ενώ η γλώσσα ερωτημάτων είναι η SQL.

Η Java όμως είναι αντικειμενοστρεφής γλώσσα προγραμματισμού και χειρίζεται ευκολότερα αντικείμενα παρά πίνακες βάσεων δεδομένων. Γι' αυτό το λόγο ορίστηκε η έννοια της οντότητας στο JPA. Πρόκειται για οποιοδήποτε POJO και ορισμένα annotations τα οποία αντικατοπτρίζουν τον πίνακα της βάσης σε JPA οντότητες.

Ένα απλό παράδειγμα οντότητας είναι το παρακάτω:

```
@Entity
public class User {
	@Id
	@GeneratedValue
	private Long id;
	private String username;
	private String password;

	public User(){}

	//Getters, setters
}
```

Το annotation Entity είναι αυτό που μετατρέπει το POJO σε οντότητα. Αυτό βέβαια δε σημαίνει ότι αυτόματα μόνο με το @Entity έχουμε και σωστή χαρτογράφηση

του αντίστοιχου πίνακα, χρειάζονται μερικά επιπλέον annotations τα οποία θα εξηγηθούν μετά το αναλυτικότερο απ' το προηγούμενο κομμάτι κώδικα που ακολουθεί.

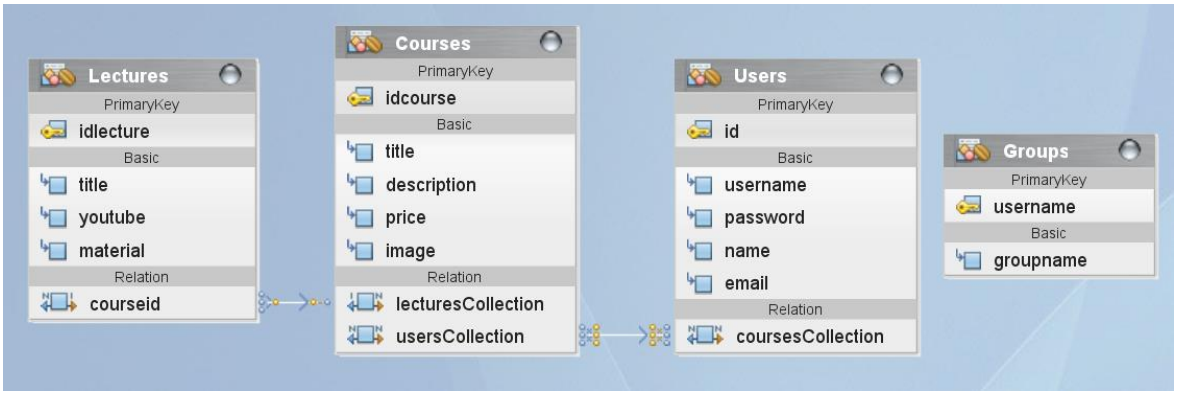
```
@Entity
@Table(name = "USERS")
public class User implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "ID")
    private Integer id;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 255)
    @Column(name = "USERNAME")
    private String username;

    @JoinTable(name = "USERCOURSES", joinColumns = {
        @JoinColumn(name = "USERID", referencedColumnName = "ID")},
inverseJoinColumns = {
        @JoinColumn(name = "COURSEID", referencedColumnName =
"IDCOURSE")})
    @ManyToMany
    private List<Course> courses;
}
```

Το @Entity είναι γνωστό από προηγουμένως. Το @Table αναφέρεται στον πίνακα της βάσης δεδομένων που θα αντιστοιχιστεί η οντότητα, λόγω ονόματος βλέπουμε ότι είναι ο πίνακας "USERS". Τα πεδία της κλάσης ορίζονται ως βασικοί τύποι, not null, τα String με μέγεθος 1 έως 255 χαρακτήρες, ενώ αντιστοιχίζονται στις στήλες του πίνακα με το @Column.

Το πεδίο List<...> με τις @JoinTable και @ManyToMany αναφορές είναι που χαρτογραφεί την Πολλά-Προς-Πολλά σχέση της User οντότητας με την οντότητα Course. Δηλαδή μία οντότητα User μπορεί να έχει σχέση με πολλές Course και το ανάποδο, μία Course να σχετίζεται με πολλές οντότητες User. Ο ενδιάμεσος πίνακας "USERCOURSES" χρησιμοποιείται για να κρατήσει την Πολλά-Προς-Πολλά σχέση και οι @JoinColumn αναφορές δείχνουν ποιά στήλη αυτής της οντότητας αντιστοιχίζεται με ποιά του ενδιάμεσου πίνακα.

Με καλύτερη παρατήρηση στο όνομα της κλάσης, βλέπουμε ότι δεν είναι το ίδιο με το όνομα του πίνακα που αντιστοιχίζεται. Ο πίνακας λέγεται USERS ενώ η κλάση User. Αυτό έχει απόλυτο νόημα καθώς ο πίνακας περιέχει πολλές εγγραφές της ίδιας οντότητας, ενώ η κλάση User αντιστοιχεί σε μία μόνο οντότητα. Το αντίστοιχο δηλαδή του πίνακα θα ήταν μια λίστα από User οντότητες.

Παρακάτω παρατίθεται εικόνα που δείχνει τις σχέσεις και τα πεδία των οντοτήτων που χρησιμοποιώ στη δική μου εφαρμογή.



ΔΙΑΧΕΙΡΙΣΗ ΟΝΤΟΤΗΤΩΝ

Υπεύθυνος για τη διαχείριση των οντοτήτων σε μια εφαρμογή είναι ο Διαχειριστής Οντοτήτων (Entity Manager). Ο em αντιστοιχίζεται κάθε φορά μ' ένα συγκεκριμένο πλαίσιο λειτουργίας, δηλαδή τις οντότητες τις οποίες θα διαχειρίζεται καθώς και τη βάση δεδομένων πάνω στην οποία θα λειτουργεί.

Ο em συνήθως χρησιμοποιείται σε enterprise javabeans (επόμενο κεφάλαιο) που εκτελούν το business logic της εφαρμογής και θέλουν πρόσβαση στη βάση δεδομένων, για να αποθηκεύσουν για παράδειγμα ένα νέο χρήστη ή να επιστρέψουν τα συνολικά κέρδη του προηγούμενου μήνα.

ΕΡΩΤΗΜΑΤΑ ΣΕ ΟΝΤΟΤΗΤΕΣ

Το JPA προσφέρει δύο τρόπους για ερωτήματα σε οντότητες, τη Java Persistence Query Language (JPQL) και το Criteria API.

Η JPQL είναι μια απλή γλώσσα ερωτημάτων οντοτήτων παραπλήσια με την SQL με τα ερωτήματα να είναι τύπου String. Το Criteria API δημιουργεί αντικείμενα και τα σχετίζει μεταξύ τους προγραμματιστικά στον κώδικα. Αυτό έχει ως αποτέλεσμα να γράφεται περισσότερος κώδικας και να δημιουργούνται περισσότερα αντικείμενα πριν την εκτέλεση του ερωτήματος, η String-based JPQL όμως χαρακτηρίζεται από τα οποιαδήποτε λάθη μπορούν να γίνουν στην κατασκευή του String (πχ ορθογραφικά λάθη), ενώ είναι απαραίτητο το casting.

Ας δούμε κάποια παραδείγματα. Για να επιστρέψουμε όλες τις εγγραφές ενός πίνακα με SQL θα γράφαμε

```
select * from Users
```

Το αντίστοιχο με JPQL θα ήταν

```
select u from User u
```

Χρησιμοποιώντας το Criteria API (αλλά και διαφορετικό παράδειγμα στον κώδικα) θα έπρεπε να γράψουμε

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<ConfigurationEntry> cq =
cb.createQuery(ConfigurationEntry.class);
```

```
Root<ConfigurationEntry> rootEntry = cq.from(ConfigurationEntry.class);
CriteriaQuery<ConfigurationEntry> all = cq.select(rootEntry);
TypedQuery<ConfigurationEntry> allQuery = em.createQuery(all);
return allQuery.getResultList();
```

(για πηγή, βλέπε Αναφορές: Κώδικας 1)

ΠΕΡΙΣΣΟΤΕΡΟΙ ΤΥΠΟΙ ΕΡΩΤΗΜΑΤΩΝ

Το JPA προσφέρει κι άλλους τρόπους για ερωτήματα σε οντότητες. Μας δίνει τη δυνατότητα να κατασκευάζουμε δυναμικά ερωτήματα, ονομαστικά ερωτήματα, εγγενή σε SQL αλλά και φυσικά stored procedures.

Τα δυναμικά ερωτήματα είναι ένα αρχικό String σε μορφή JPQL, με String concatenations όταν υπάρχει συνθήκη. Έτσι για παράδειγμα θα φτιάχναμε δυναμικά ένα ερώτημα

```
String jpqlQ = "SELECT u FROM User u";
if(sth-happens)
      jpqlQ += " WHERE u.name = 'Vasilis'";
Query query = em.createQuery(jpqlQ);
List<User> users = query.getResultList();
```

Τα ονομαστικά ερωτήματα (named queries) είναι static και δεν αλλάζουν. Είναι ταχύτερα στην εκτέλεση αφού η σύνδεση με τη βάση δεδομένων τα μεταφράζει αμέσως σε SQL ερωτήματα κι όχι κάθε φορά που εκτελείται το ερώτημα. Γράφονται ως annotations στις κλάσεις των οντοτήτων κι είναι της μορφής

```
@Entity
@Table(name = "USERS")
@NamedQueries({
    @NamedQuery(name = "Users.findById", query = "SELECT u FROM Users u
WHERE u.id = :id"),
    @NamedQuery(name = "Users.findByUsername", query = "SELECT u FROM
Users u WHERE u.username = :username"),
    @NamedQuery(name = "Users.findByPassword", query = "SELECT u FROM
Users u WHERE u.password = :password"),
    @NamedQuery(name = "Users.findByName", query = "SELECT u FROM Users u
WHERE u.name = :name"),
    @NamedQuery(name = "Users.findByEmail", query = "SELECT u FROM Users
u WHERE u.email = :email")})
    @NamedQuery(name = "Users.findAllUsers", query = "SELECT u FROM USERS
u")})
public class User implements Serializable {
```

Μ' αυτόν τον τρόπο, απλά γίνεται κλήση της μεθόδου createNamedQuery("...") με το όνομα του named query που επιθυμούμε. Η εκάστοτε παράμετρος (όπως στα ερωτήματα με WHERE) μπορεί να περάσει με κλήση της μεθόδου setParameter(...).

PERSISTENCE UNIT

Για να γίνουν όλα αυτά τα ερωτήματα προς τη βάση δεδομένων, πρέπει ν' αποκτήσουμε πρόσβαση σ' αυτή. Πρέπει να οριστεί ο JDBC driver αναλόγως της βάσης δεδομένων, πχ Derby, Oracle, MySQL κλπ, το όνομα αυτής κ.ά.

Απαιτείται προφανώς ένα αρχείο ρυθμίσεων το οποίο περιέχει όλους αυτούς τους ορισμούς. Το αρχείο αυτό ονομάζεται persistence.xml και "συνδέει" τους εκάστοτε entity managers με τις βάσεις δεδομένων που θα χρησιμοποιήσουν καθώς κι όλες τις παραμέτρους για να συνδεθούν.

Στο παρακάτω παράδειγμα (Αναφορές, Persistence 1 για πηγή) βλέπουμε ένα ολοκληρωμένο αρχείο ρυθμίσεων persistence.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
             version="2.1">

  <persistence-unit name="chapter04PU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>org.agoncal.book.javaee7.chapter04.Book</class>
    <properties>
      <property name="javax.persistence.schema-
generation.database.action" value="drop-and-create"/>
      <property name="javax.persistence.jdbc.driver"
value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="javax.persistence.jdbc.url"
value="jdbc:derby://localhost:1527/chapter04DB;create=true"/>
      <property name="javax.persistence.jdbc.user" value="APP"/>
      <property name="javax.persistence.jdbc.password" value="APP"/>
      <property name="javax.persistence.sql-load-script-source"
value="insert.sql"/>
      <property name="eclipselink.logging.level" value="INFO"/>
    </properties>
  </persistence-unit>
</persistence>
```

Στις απαραίτητες ρυθμίσεις, ορίζεται το όνομα του persistence unit που σ' αυτή την περίπτωση είναι "chapter04PU". Με το "ClientDriver" ορίζεται ο JDBC driver για τη Derby βάση δεδομένων, ενώ με το property "url" ορίζεται η τοποθεσία και το όνομα αυτής, εδώ έχουμε τη βάση με όνομα "chapter04DB" στο localhost:1527. Τα credentials για σύνδεση στη βάση είναι "APP" τόσο για το username όσο και για τον κωδικό πρόσβασης. Τέλος, το class tag ορίζει ότι η κλάση Book θα έχει persistent ιδιότητες μέσω του entity manager στη βάση δεδομένων.

ΕΠΙΛΟΓΟΣ

Κλείνοντας αυτό το κεφάλαιο, γίνεται κατανοητό ότι απλά αγγίχθηκε η επιφάνεια του JPA. Περιέχει πολλά περισσότερα απ' όσα αναφέρθηκαν εδώ αλλά θα ξεπερνούσαν το scope της εργασίας. Είδαμε με ποιόν τρόπο πίνακες μιας βάσης δεδομένων αντιστοιχίζονται σε κλάσεις οντοτήτων μαζί με τις συσχετίσεις τους, καθώς και τον entity manager που αποκτά πρόσβαση σε βάσεις δεδομένων που ορίζονται μέσω του JDBC API κι εκτελεί ερωτήματα σε οντότητες. Επιπλέον,

είδαμε αρκετούς τρόπους δημιουργίας ερωτημάτων σε οντότητες για να ξέρουμε ότι δεδομένης οποιασδήποτε κατάστασης, σίγουρα θα βρούμε τρόπο να αλληλεπιδράσουμε με τη βάση δεδομένων.

Στο επόμενο κεφάλαιο θα δούμε πως οι entity managers χρησιμοποιούνται στα λεγόμενα enterprise javabeans, που αναφέρθηκαν τόσες φορές ως τώρα.

ΚΕΦΑΛΑΙΟ 8

ENTERPRISE JAVABEANS

ΕΙΣΑΓΩΓΗ

Μέχρι τώρα είδαμε δύο πολύ σημαντικούς μηχανισμούς για την κατασκευή enterprise εφαρμογών. Το JSF είναι υπεύθυνο για το UI της εφαρμογής, ενώ το JPA αναλαμβάνει να ολοκληρώσει τις συναλλαγές με τη βάση δεδομένων. Το business logic όμως μιας εφαρμογής κάπου πρέπει να εφαρμόζεται. Δεν είναι λογικό να εφαρμόζεται στο UI μιας και ο client ίσως να μην είναι μοναδικός, όπως είπαμε μπορεί να υπάρχει ο browser αλλά και μια Android εφαρμογή ή κάποια άλλη κατασκευασμένη σε Swing. Για να υπάρχει λοιπόν ο "διαχωρισμός

καθηκόντων" σε μια εφαρμογή (το κάθε component να εκτελεί μια συγκεκριμένη λειτουργία), το business logic εφαρμόζεται στα enterprise javabeans.

ΤΙ ΕΙΝΑΙ ΤΑ ENTERPRISE JAVABEANS

Τα EJB είναι components που εφαρμόζουν το business logic της εκάστοτε εφαρμογής. Είναι αυτά που θα καλέσουν για παράδειγμα τη checkIfApplicableForLoan() μέθοδο της εφαρμογής, ή θα κάνουν έλεγχο για το ποιά προϊόντα είναι σε μικρή διαθεσιμότητα και χρειάζεται παραγγελία τους. Ακόμη, κάποιο EJB θα ολοκληρώσει την αγορά προϊόντων του χρήστη σ' ένα online κατάστημα ενώ ίσως κάποιο άλλο θ' αναλάβει να χρεώσει τον τραπεζικό λογαριασμό του. Κάποιο EJB θ' αναλάβει στο τέλος κάθε μήνα να διαβάσει τη βάση δεδομένων και να στείλει εντολή στο mail server για αποστολή μηνύματος στους χρήστες, ότι ξεπέρασαν το δωρεάν όριο ομιλίας στο συμβόλαιο κινητής τους.

Μεγάλο πλεονέκτημα της χρήσης των EJB είναι η ελάττωση του χρόνου ανάπτυξης της εφαρμογής και η επαναχρησιμοποίηση του κώδικα. Οι προγραμματιστές δε χρειάζεται ν' αναμιχθούν στο πώς θα επικοινωνήσει το EJB με τις υπηρεσίες του application server, ο server μέσω του EJB container αναλαμβάνει από μόνος του να εκτελέσει κάποιες λειτουργίες όπως η επικοινωνία με τη βάση δεδομένων.

Επιπλέον, τα συγκεκριμένα EJB που εκτελούν λειτουργίες σε μια εφαρμογή μπορούν να μεταφερθούν σε μια άλλη για να κατασκευάσουν μια διαφορετική εφαρμογή. Αναλόγως του packaging των EJB, μπορούμε να 'χουμε για παράδειγμα έναν application server για web clients κι έναν για command line clients. Και οι δύο μπορούν να μοιράζονται την ίδια βάση δεδομένων αλλά και τα ίδια EJB. Απλά ο web client επικοινωνεί με τα EJB μέσω άλλων bean, ενώ ο command line client απευθείας με τα EJB. Μ' αυτόν τον τρόπο φαίνεται η επαναχρησιμοποίηση του κώδικα αλλά κι η φορητότητα του συγκεκριμένου API.

ΤΥΠΟΙ ΤΩΝ ENTERPRISE JAVABEANS

Τα EJB χωρίζονται σε δύο τύπους, τα session beans και τα message-driven beans. Τα message-driven beans χρησιμοποιούνται ως listeners για μετάδοση μηνυμάτων μέσω του Java Message Service API και δε θα μας απασχολήσουν σ' αυτή την εργασία.

Τα session beans όμως είναι αυτά που ενθυλακώνουν το business logic κι εκτελούν λειτουργίες για τον client. Αυτά, χωρίζονται εκ νέου σε τρεις τύπους, τα stateful, stateless και singleton.

Τα stateful session beans συγκρατούν την κατάστασή τους για πολλές κλήσεις μεταξύ client και server. Για παράδειγμα, στην ολοκλήρωση αγοράς προϊόντων που απαιτούνται κάποια βήματα επιβεβαίωσης (έλεγχος διαθέσιμου υπολοίπου στο λογαριασμό, ερώτηση για χρέωση σε άλλο λογαριασμό και τελική επιβεβαίωση), το EJB θα πρέπει να κρατά αποθηκευμένα τα στοιχεία και στα τρία

αυτά βήματα μέχρι και την αποθήκευση στη βάση δεδομένων. Αν όμως η συνδιαλλαγή μεταξύ client και server αποτύχει, το state (η κατάσταση) του session bean χάνεται.

Αντίθετα, τα stateless session beans δεν κρατούν το state τους για πάνω από μία κλήση. Στο τελικό στάδιο για παράδειγμα της αγοράς προϊόντων, το session bean που θ' αναλάβει να καλέσει τον entity manager και να αποθηκεύσει τα προϊόντα στη βάση δεδομένων, θα είναι τύπου stateless αφού καλεί τη μέθοδο persist() του entity manager μια φορά. Ακόμα, ένα stateless bean θα χρησιμοποιηθεί για στείλει ένα email στο χρήστη μετά την ολοκλήρωση εγγραφής σε μια υπηρεσία.

Τα singleton beans είναι διαθέσιμα για όλους τους χρήστες της εφαρμογής και υπάρχει μόνο ένα για όλη την εφαρμογή, σε αντίθεση με τα stateless και stateful beans όπου κάθε χρήστης μπορεί να έχει πολλά απ' το καθένα. Επιπλέον, υποστηρίζουν το concurrency (συγχρονισμός) αφού πολλοί χρήστες έχουν ταυτόχρονη πρόσβαση σ' αυτό. Ως παράδειγμα θα μπορούσε ν' αναφερθεί ένα online collaboration εργαλείο όπου πολλοί χρήστες έχουν πρόσβαση στο ίδιο έγγραφο και το επεξεργάζονται ταυτόχρονα.

ΔΟΜΗ ΚΑΙ ΧΡΗΣΗ ΕΝΟΣ ENTERPRISE JAVABEAN

Θα περίμενε κανείς ότι η κατασκευή ενός session bean θα ήταν περίπλοκη, μιας και διαθέτει τόσες πολλές δυνατότητες. Κι όμως, το μόνο που χρειάζεται για να δημιουργηθεί ένα session bean είναι μια Java κλάση κι ένα annotation. Παρατίθενται δύο απλά παραδείγματα.

```
/**
 * @author Arun Gupta
 */
@Stateless
public class AccountSessionBean {

    private float amount = 0;

    public String withdraw(float amount) {
        this.amount -= amount;
        return "Withdrawn: " + amount;
    }

    public String deposit(float amount) {
        this.amount += amount;
        return "Deposited: " + amount;
    }

    public float getAmount() {
        return this.amount;
    }
}
```

Στο παράδειγμα αυτό (βλέπε Αναφορές, Κώδικας 2 για πηγή) βλέπουμε ότι η κλάση AccountSessionBean εκτελεί μια συγκεκριμένη λειτουργία ενθυλακώνοντας το business logic της συγκεκριμένης περίπτωσης, ερώτηση υπολοίπου, ανάληψη και κατάθεση. Βέβαια αν το παράδειγμα ήταν πιο ολοκληρωμένο και υπήρχε το

υπόλοιπο context, ο κώδικας θα είχε περισσότερο νόημα. Παρ' όλ' αυτά, ο παρόν κώδικας έχει νόημα και παρουσιάζει την ιδέα του EJB.

Το επόμενο κομμάτι κώδικα (πλήρης στο Παράρτημα 3), είναι από τη δική μου εργασία και παρουσιάζει πώς το session bean συνεργάζεται με το JPA και τον entity manager για να συσχετίσει το χρήστη με τα μαθήματα που έχει αγοράσει.

```
@Stateless
public class CourseFacade extends AbstractFacade<Course> {
    @PersistenceContext(unitName = "vasouvPU")
    private EntityManager em;

    public void setCourseUser(User u, int courseID) {
        Course c =
(Course)em.createNamedQuery("Courses.findByIdcourse").setParameter("idcou
rse", courseID).getSingleResult();
        c.getUsers().add(u);
        em.merge(c);
    }
}
```

Αυτό που πρέπει να παρατηρηθεί είναι η χρήση του @PersistenceContext(...) όπου "συνδέει" τον entity manager με τη βάση δεδομένων στην οποία θα λειτουργήσει.

Η χρήση των EJB στον κώδικα της εφαρμογής γίνεται με τον Injection μηχανισμό που είδαμε στο CDI κεφάλαιο, με τη διαφορά αντί για το @Inject χρησιμοποιείται το @EJB. Τα παρακάτω snippets κώδικα προέρχονται από την κλάση CheckoutBean της εργασίας μου, ο πλήρης κώδικας της οποίας βρίσκεται στο Παράρτημα 4.

```
@EJB
private AuthenticationEJB auth;
```

Τίποτα παραπάνω δε χρειάζεται για να χρησιμοποιηθεί ένα session bean στον κώδικα μιας άλλης κλάσης. Στη συγκεκριμένη περίπτωση το χρειαζόμαστε για να λάβουμε το όνομα του χρήστη ανά πάσα στιγμή γράφοντας

```
username = auth.getUser().getUsername();
```

Ακριβώς όπως και στα CDI beans, υπάρχει άμεση πρόσβαση στα πεδία και μεθόδους του αντικειμένου που εισάγεται.

ΕΠΙΛΟΓΟΣ

Ολοκληρώνοντας το κεφάλαιο, γίνεται κατανοητό ότι η συμβολή του EJB API στο stack της JavaEE έφερε μεγάλες αλλαγές στην ανάπτυξη enterprise εφαρμογών. Το layering κι ο "διαχωρισμός καθηκόντων" σε μια εφαρμογή όχι μόνο μειώνει το χρόνο ανάπτυξης, αλλά διευκολύνει και το scalability αυτής.

Από ένα "απλό" EJB που αναλαμβάνει να προσθέσει δύο αριθμούς, έως ένα πολυπλοκότερο που αλληλεπιδρά με τη βάση δεδομένων μέσω του JPA, η χρήση τους γίνεται μέσω του Injection μηχανισμού του CDI. Τόσο για τη δημιουργία όσο και για τη χρήση ενός EJB, το μόνο που χρειάζεται κάποιος είναι ένα annotation.

Αφού τα EJB προσφέρουν τέτοια πλεονεκτήματα, η χρήση τους δεν περιορίζεται σε web εφαρμογές μ' έναν browser για client. Χρησιμοποιούνται και σε web services με τη SOAP ή REST μεθοδολογία, το καθένα με τις ιδιαιτερότητές του. Αυτά όμως στο επόμενο κεφάλαιο.

ΚΕΦΑΛΑΙΟ 9

WEB SERVICES

ΕΙΣΑΓΩΓΗ

Ο όρος "web service" σημαίνει ότι "κάτι" είναι προσβάσιμο στο "web" προσφέροντας μια "υπηρεσία"3. Πρόκειται για εφαρμογές client και server που επικοινωνούν μέσω HTTP, οι servers προσφέρουν την υπηρεσία και οι clients την καταναλώνουν (consume). Δεδομένου ότι τα web services μεταδίδουν τα μηνύματά τους μέσω XML, η διαλειτουργικότητα (interoperability) που επιτυγχάνεται είναι πολύ μεγάλη. Ο client μπορεί να είναι κατασκευασμένος σε οποιαδήποτε πλατφόρμα, Python, Ruby, C++ κλπ, αρκεί να μπορεί να διαβάσει XML μηνύματα. Στη JavaEE, οι τύποι ανάπτυξης των web service είναι οι SOAP και RESTful μεθοδολογίες.

SOAP ΚΑΙ RESTFUL WEB SERVICES

Η SOAP μεθοδολογία χρησιμοποιεί το ομώνυμο πρωτόκολλο κωδικοποίησης για μετάδοση μηνυμάτων (Simple Object Access Protocol - SOAP), τη WSDL γλώσσα (Web Services Definition Language, βασισμένη στην XML), ενώ για τη μεταφορά του μηνύματος δεν περιορίζεται στο σύνηθες HTTP πρωτόκολλο αλλά χρησιμοποιεί και άλλες τεχνολογίες όπως SMTP ή JMS. Η WSDL χρησιμοποιείται για να δημιουργήσει το επίσημο συμβόλαιο της λειτουργίας του service. Αναφέρει τις λειτουργίες, τύπους δεδομένων, μηνύματα αλλά και την τοποθεσία του service. Ακόμα, περιγράφει συγκεκριμένες απαιτήσεις που χρειάζονται αναλόγως της περίστασης, για παράδειγμα απαιτήσεις ασφάλειας και συναλλαγών με μια βάση δεδομένων.

Λόγω της αρχιτεκτονικής του SOAP, είναι ενδεδειγμένη λύση για "big" web services που χρειάζεται μεγάλο Quality of Service, εκεί δηλαδή που απαιτείται ασφάλεια και διακριτικότητα όπως είναι οι enterprise εφαρμογές τραπεζικών οργανισμών.

Για άλλα απλούστερα σενάρια, χρησιμοποιείται η REST μεθοδολογία. Βασισμένη στο HTTP πρωτόκολλο και χρησιμοποιώντας το στο έπακρο, είναι πιο απλή στη χρήση και οι εφαρμογές της πιο διαδεδομένες. Κάθε τύπος πληροφορίας που κάνει expose η REST, είναι και διαφορετικό resource με δική του διεύθυνση. Μ' αυτόν τον τρόπο χρησιμοποιεί τη δομή του web και κάθε resource έχει το δικό του URI. Δεν απαιτεί τη χρήση XML μηνυμάτων ούτε WSDL συμβολαίων, αν και συνήθως τα μηνύματα είναι τύπου XML ή JSON. Η απουσία συμβολαίων, έχει ως αποτέλεσμα την απαίτηση για πλήρη κατανόηση της δομής του μηνύματος και του περιεχομένου του.

[3] Antonio Goncalves, Beginning JavaEE 7, p.455

Δεδομένης της μεγάλης υιοθέτησης της RESTful μεθοδολογίας, η ενδεδειγμένη χρήση της είναι για web εφαρμογές καθώς πολλοί clients έχουν τη δυνατότητα να καταναλώσουν το REST περιεχόμενο. Ακόμα, το εύκολο scalability της εφαρμογής αλλά και η απλότητα της αρχιτεκτονικής τους, καθιστούν τα RESTful web services κατάλληλα για γενική χρήση.

ΠΑΡΑΔΕΙΓΜΑ ΧΡΗΣΗΣ ΤΟΥ JAX-WS

Για καλύτερη κατανόηση του JAX-WS API, το παρακάτω απλό παράδειγμα δείχνει πόσο εύκολο είναι να κατασκευαστεί ένα web service. Ο κώδικας προέρχεται από το 10 κεφάλαιο του Java EE 7 with Glassfish 4 Application Server βιβλίου.

```
@WebService
public class Calculator {

    @WebMethod
    public int add(int first, int second) {
        return first + second;
    }

    @WebMethod
    public int sub(int first, int second) {
        return first - second;
    }

}
```

Η κλάση Calculator γίνεται expose ως web service με το annotation @WebService. Οι public μέθοδοι της κλάσης γίνονται αυτόματα expose ως web services αλλά το @WebMethod annotation είναι προαιρετικό.

Μετά το compile, package και deploy της εφαρμογής στον Glassfish, ανοίγοντας το admin console αυτού και πηγαίνοντας στο node της εφαρμογής, αφού πατήσουμε στο View Waypoint έχουμε την παρακάτω εικόνα

Web Service Endpoint Information

View details about a web service endpoint.

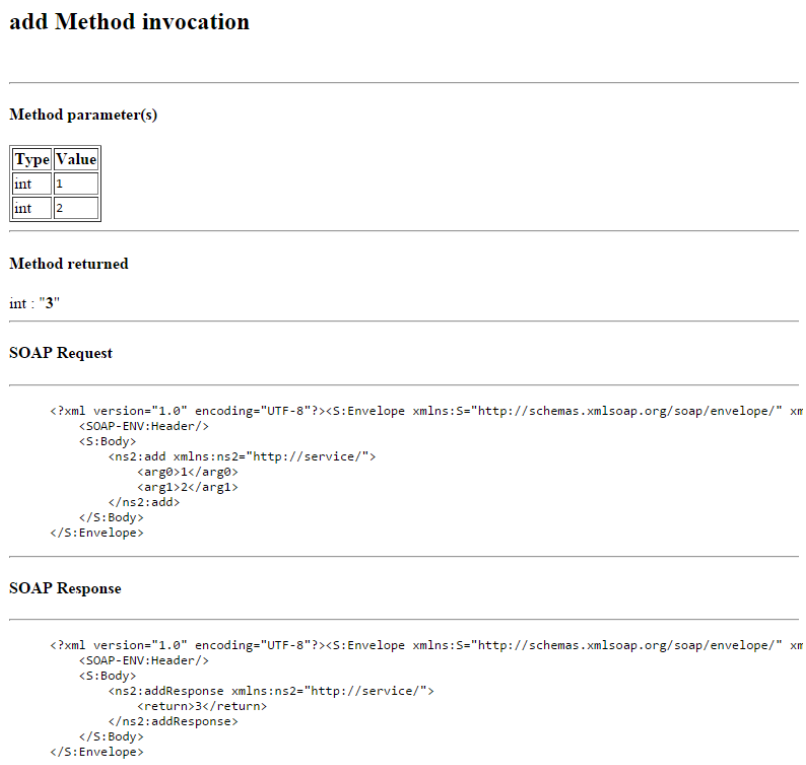
Application Name:	WebApplication3
Tester:	/WebApplication3/CalculatorService?Tester
WSDL:	/WebApplication3/CalculatorService?wsdl
Endpoint Name:	Calculator
Service Name:	CalculatorService
Port Name:	CalculatorPort
Deployment Type:	109
Implementation Type:	SERVLET
Implementation Class Name:	service.Calculator
Endpoint Address URI:	/WebApplication3/CalculatorService
Namespace:	http://service/

Σχήμα 4: Web Service Viewpoint

Το Tester link μας επιτρέπει να δοκιμάσουμε το service και τις μεθόδους του, επομένως θα δούμε την παρακάτω εικόνα

Σχήμα 5: Web Service Tester

Με την εισαγωγή αριθμών στις ανάλογες μεθόδους, έχουμε κλήση της μεθόδου και η απάντηση του service φαίνεται παρακάτω



Σχήμα 6: Web Service Response

Μέρος του WSDL συμβολαίου που δημιουργήθηκε και ορίζει τις λειτουργίες του web service, φαίνεται παρακάτω

```
▼<types>
  ▼<xsd:schema>
     <xsd:import namespace="http://service/" schemaLocation="http://vasouv-
     desktop:8080/WebApplication3/CalculatorService?xsd=1"/>
   </xsd:schema>
  </types>
▶<message name="add">...</message>
▶<message name="addResponse">...</message>
▶<message name="sub">...</message>
▶<message name="subResponse">...</message>
▼<portType name="Calculator">
  ▶<operation name="add">...</operation>
  ▶<operation name="sub">...</operation>
  </portType>
▼<binding name="CalculatorPortBinding" type="tns:Calculator">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  ▶<operation name="add">...</operation>
  ▶<operation name="sub">...</operation>
  </binding>
▼<service name="CalculatorService">
  ▼<port name="CalculatorPort" binding="tns:CalculatorPortBinding">
     <soap:address location="http://vasouv-desktop:8080/WebApplication3/CalculatorService"/>
   </port>
  </service>
</definitions>
```

Σχήμα 7: Web Service WSDL

Στην αρχή του κεφαλαίου αναφέρθηκε ότι τα web services είναι client και server εφαρμογές. Στην περίπτωση της παραπάνω εφαρμογής, client είναι ο browser και το response του service έρχεται μετά από επίσκεψη στο Tester link.

ΠΑΡΑΔΕΙΓΜΑ ΧΡΗΣΗΣ ΤΟΥ JAX-RS

Η REST μεθοδολογία είναι στενά συνδεδεμένη με το HTTP πρωτόκολλο και γι' αυτό το λόγο τα RESTful web services πρέπει να υποστηρίζουν τουλάχιστον μία από τις HTTP μεθόδους GET, POST, PUT και DELETE. Και στο JAX-RS, χρησιμοποιούνται annotations για να δείξουν ποιά μέθοδος της Java κλάσης θα κληθεί στην ανάλογη HTTP μέθοδο.

Τα κομμάτια κώδικα που ακολουθούν είναι αποσπάσματα από το Παράρτημα 5, ενώ ο συνολικός κώδικας δημιουργήθηκε αυτόματα με το NetBeans.

```
@Stateless
@Path("vasouv.javaee7thesis.courses.course")
public class CourseFacadeREST extends AbstractFacade<Course> {
    @PersistenceContext(unitName = "vasouvPU")
    private EntityManager em;

    @POST
    @Override
    @Consumes({"application/xml", "application/json"})
    public void create(Course entity) {
        super.create(entity);
    }

    @PUT
    @Path("{id}")
    @Consumes({"application/xml", "application/json"})
    public void edit(@PathParam("id") Integer id, Course entity) {
        super.edit(entity);
    }
```

```
@DELETE
@Path("{id}")
public void remove(@PathParam("id") Integer id) {
    super.remove(super.find(id));
}

@GET
@Override
@Produces({"application/xml", "application/json"})
public List<Course> findAll() {
    return super.findAll();
}
```

Στην κλάση CourseFacadeREST παρατηρούμε το εξής, πρόκειται για stateless EJB με όλες τις μεθόδους της ν' απαντούν σε κάποια HTTP μέθοδο. Το @Path αναφέρει το URI της κλάσης το οποίο είναι μοναδικό. Η μέθοδος findAll() απαντάει στη GET μέθοδο και καλείται αυτόματα όταν επισκεπτόμαστε για πρώτη φορά το URI. Η create() μέθοδος χρησιμοποιείται για να εισάγει ένα νέο resource μέσω της POST μεθόδου. Αντίστοιχα οι edit() και remove() μέθοδοι ενημερώνουν και διαγράφουν resources από τη βάση δεδομένων με τις αντίστοιχες HTTP κλήσεις.

Απαιτείται όμως και ρύθμιση του path των resources, από ποιό αρχικό σημείο είναι δηλαδή προσβάσιμα. Μια κλάση λοιπόν η οποία κάνει extend το javax.ws.rs.core.Application και με annotation @javax.ws.rs.ApplicationPath("webresources") αρκεί για να ονομάσει το path.

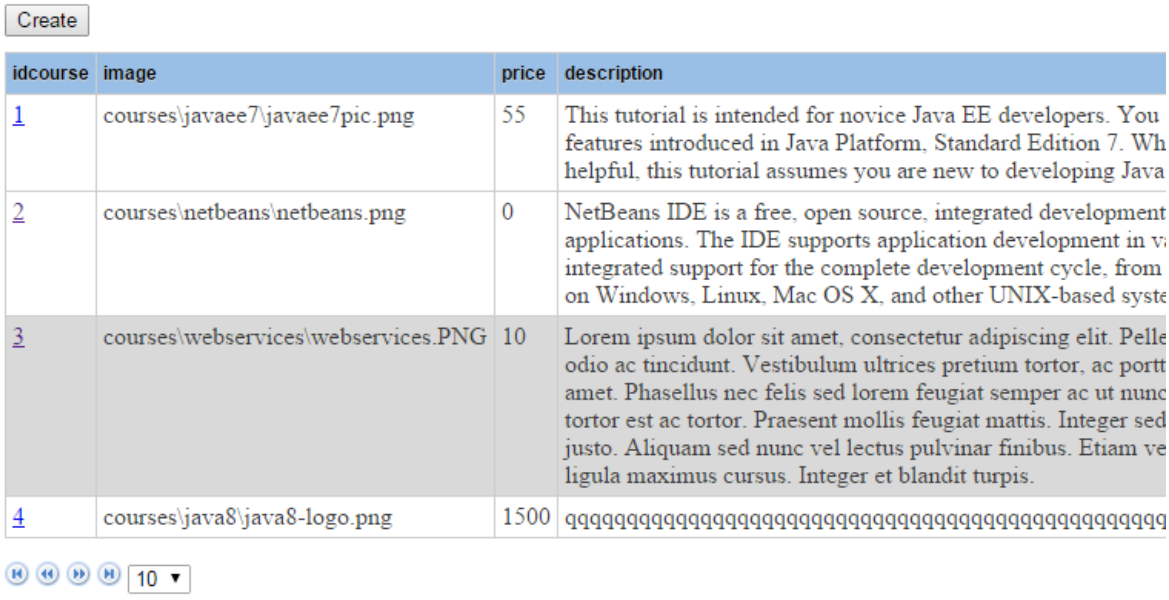
Με επίσκεψη λοιπόν στο link <...>/webresources/vasouv.javaee7thesis.courses.course/ έχουμε την παρακάτω εικόνα

```
▼<courses>
  ▼<course>
    ▶<description>...</description>
     <idcourse>1</idcourse>
     <image>courses\javaee7\javaee7pic.png</image>
     <price>55</price>
     <title>Learning Java EE 7</title>
   </course>
  ▼<course>
    ▶<description>...</description>
     <idcourse>2</idcourse>
     <image>courses\netbeans\netbeans.png</image>
     <price>0</price>
     <title>Developing Applications with NetBeans 8</title>
   </course>
  ▼<course>
    ▶<description>...</description>
     <idcourse>3</idcourse>
     <image>courses\webservices\webservices.PNG</image>
     <price>10</price>
     <title>Web Services</title>
   </course>
  ▼<course>
    ▶<description>...</description>
     <idcourse>4</idcourse>
     <image>courses\java8\java8-logo.png</image>
     <price>1500</price>
     <title>Java8</title>
   </course>
</courses>
```

Σχήμα 8: REST GET

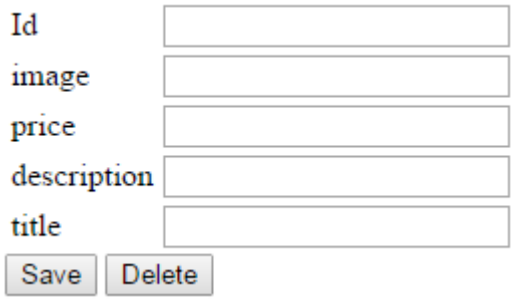
όπου σε XML μορφή φαίνεται το περιεχόμενο που επιστρέφει η μέθοδος findAll(). Οι υπόλοιπες μέθοδοι είναι ευκολότερο να οπτικοποιηθούν αν χρησιμοποιήσουμε το link που δημιούργησε αυτόματα το Netbeans για το συγκεκριμένο σκοπό.

Βρισκόμαστε αντιμέτωποι με την παρακάτω εικόνα



Σχήμα 9: Netbeans REST

Πρόκειται για το αρχικό GET request κι εμφανίζει τα περιεχόμενα της μεθόδου. Με κλικ στο "Create" εμφανίζεται το πλαίσιο



Σχήμα 10: Netbeans POST

όπου μπορούμε να εισάγουμε τιμές για τις μεταβλητές. Μετά το "Save" εκτελείται η POST μέθοδος κι αποθηκεύει τα δεδομένα.

ΕΠΙΛΟΓΟΣ

Το stack της JavaEE με την εισαγωγή των δύο web services API, διευκόλυνε σε πολύ μεγάλο βαθμό την ανάπτυξη web services αλλά και το integration αυτών σε ήδη υπάρχουσες enterprise εφαρμογές. Είναι ξεκάθαρο ότι με λίγες μόνο γραμμές κώδικα μπορούμε να έχουμε έτοιμα web services που ακολουθούν την εκάστοτε μεθοδολογία. Είτε πρόκειται για SOAP ή για RESTful εφαρμογές, η ανάπτυξη αυτών είναι απλούστατη αφήνοντας τον προγραμματιστή ν' ασχοληθεί με το business logic αντί την ενσωμάτωση αυτών των τεχνολογιών σε μια εφαρμογή.

Μετά από αυτή την αναφορά στο HTTP πρωτόκολλο και τις τεχνολογίες που βασίζονται σ' αυτό, θα γίνει μια αναδρομή στο παρελθόν και το Servlet API καθώς ήταν ο πρώτος τρόπος της JavaEE να παρουσιάσει δυναμικό περιεχόμενο στις enterprise εφαρμογές.

ΚΕΦΑΛΑΙΟ 10

SERVLETS

ΕΙΣΑΓΩΓΗ

Το Servlet API είναι ο αρχικός μηχανισμός που εισήγαγε η Sun στη JavaEE για την ανάπτυξη δυναμικών web εφαρμογών. Frameworks όπως τα JSF και JSP μετατρέπουν τον κώδικά τους σε servlets, είναι χτισμένα πάνω σ' αυτό το API και λειτουργούν σύμφωνα μ' αυτό παρ' όλο που η όλη διεργασία γίνεται χωρίς να το γνωρίζει ο προγραμματιστής.

ΑΝΑΦΟΡΙΚΑ ΓΙΑ ΤΑ SERVLETS

Τα servlets είναι Java κλάσεις που γίνονται deploy στον application server κι εκτελούν διάφορες διεργασίες απαντώντας σε request-response HTTP μεθόδους. Όταν γίνει το request από τον client, ο servlet container βρίσκει το servlet που πρέπει να εκτελεστεί κι αυτό καλεί αναλόγως την doGet(), doPost(), doPut(), doDelete() μέθοδο. Αυτές είναι οι πιο συνηθισμένες αν και μπορεί να καλέσει πολλές ακόμα. Δεδομένου ότι πρόκειται για Java κλάσεις, τα servlets έχουν όλα τα χαρακτηριστικά των Java κλάσεων και μπορούν να εκτελέσουν οποιοδήποτε έγκυρο Java κώδικα. Αυτό σημαίνει ότι μπορούν να αλληλεπιδράσουν με άλλα αντικείμενα για να διαχειριστούν πληροφορίες, να έχουν πρόσβαση σε μια βάση δεδομένων αλλά και να χρησιμοποιήσουν το default μηχανισμό ασφάλειας του application server.

ΠΑΡΑΔΕΙΓΜΑ ΧΡΗΣΗΣ SERVLET

Το παρακάτω κομμάτι κώδικα προέρχεται από τα JavaEE 7 Tutorial Examples, στην ολόκληρη μορφή του παρατίθεται στο Παράρτημα 6.

```
@WebServlet("/report")
public class MoodServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            //code
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        processRequest(request, response);
    }

    @Override
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

Σ' αυτήν την κλάση παρατηρούμε τα εξής, το annotation @WebServlet χαρακτηρίζει το servlet έτσι ώστε να μη χρειάζεται ν' αναφερθεί στον configuration descriptor (web.xml) και θέτει το path του servlet. Η κλάση υλοποιεί τις μεθόδους doGet() και doPost() οι οποίες απαντούν στις αντίστοιχες HTTP αιτήσεις. Η μέθοδος processRequest() εκτελείται στη συγκεκριμένη περίπτωση τόσο για τη GET όσο και για την POST αίτηση, φυσικά και θα μπορούσε να εκτελείται διαφορετική μέθοδος κάθε φορά.

Μέσω της παραμέτρου request μπορούμε να λάβουμε δεδομένα από την αίτηση. Στην περίπτωση για παράδειγμα που υπάρχει μια HTML φόρμα κι ο χρήστης προσπαθεί να συνδεθεί στην εφαρμογή, η request περιλαμβάνει τα στοιχεία που δόθηκαν σύμφωνα με το HTTP πρωτόκολλο. Με έναν υποτιθέμενο έλεγχο στη βάση δεδομένων, η response παράμετρος θα μετέφερε το χρήστη στο λογαριασμό του αν τα στοιχεία του ήταν σωστά, ή σε μια σελίδα λάθους αν τα στοιχεία ήταν λανθασμένα. Αυτό θα συνέβαινε στην processRequest() μες στο try{} όπου ο PrintWriter κατασκευάζει τη σελίδα.

ΕΠΙΛΟΓΟΣ

Μπορεί το παράδειγμα που προηγήθηκε να είναι εξαιρετικά απλό στη δομή του, εξηγεί τη βασική λειτουργία των servlets: να απαντούν σε HTTP μεθόδους. Αρκετά διαφορετικό προγραμματιστικό μοντέλο σε σχέση με το JSF, ήταν το πρώτο API που δημιούργησε δυναμικό περιεχόμενο στη JavaEE stack. Μετά από πολλές αλλαγές κι ενημερώσεις, χρησιμοποιείται ακόμα και σήμερα μιας και είναι το underlying μοντέλο στο οποίο βασίζεται το JSF.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Σ' αυτό το σημείο θα ήθελα ν' αναφέρω κάποια πράγματα που πιθανότατα είναι ήδη εμφανή, τόσο σ' αυτή την εργασία αλλά και στο project που τη συνοδεύει.

Η JavaEE 7 αποτελείται από πολλές τεχνολογίες με πολύ μεγάλο φάσμα περιπτώσεων χρήσης. Δεν ήταν δυνατό ν' αναφέρω περισσότερα API καθώς το μέγεθος του εγγράφου θα μεγάλωνε αρκετά. Η αναφορά άλλων τεχνολογιών θα έκανε ακόμα πιο επιφανειακή την ενασχόληση με τα ήδη αναφερθέντα API και δε θα υπήρχε σωστή ενσωμάτωση στην εργασία.

Θεωρώ πως υπάρχει ισορροπία στις τεχνολογίες που χρησιμοποίησα στο project και σ' αυτές που δε χρησιμοποίησα αλλά αναφέρονταν στη δήλωση της πτυχιακής, με ανάλογη έκταση στις παραπάνω σελίδες. Προτίμησα την ενασχόληση με τα πλέον σημαντικά API της JavaEE, παρά την εμβάθυνση σε ορισμένα από αυτά και παραμέληση άλλων.

Το project που συνοδεύει αυτό το έγγραφο, χρήζει κάποιων αναφορών επίσης. Παρ' όλο που πρόκειται για ένα σύστημα που λειτουργεί σωστά, υπάρχουν κάποια σημεία που εντοπίζεται μερική ή καθόλου υλοποίηση. Αυτές ήταν αυθαίρετες αποφάσεις που λήφθηκαν είτε γιατί δεν ήταν ιδιαίτερα σχετικές με το θέμα ή γιατί δεν ήμουν σίγουρος για την υλοποίηση.

Στο σημείο για παράδειγμα που ο χρήστης καλείται να πληρώσει για τις διαλέξεις που επιθυμεί, κανονικά πρέπει να υπάρχει ενσωμάτωση με το API του PayPal ή παρόμοιας e-banking υπηρεσίας. Δεν υπάρχει τέτοια υλοποίηση στο project για ευνόητους λόγους. Η Contact σελίδα είναι χωρίς υλοποίηση καθώς δεν ήξερα πώς να κατασκευάσω την εν λόγω υπηρεσία, ίσως με χρήση του JMS API ή JSON.

Επίσης, στην σελίδα δημιουργίας των μαθημάτων αυτή τη στιγμή μπορούν να εισαχθούν μόνο τρία μέρη για το κάθε μάθημα, τα tabs είναι στατικά. Θα ήθελα να υπάρχει δυναμική δημιουργία των tabs μ' ένα αντίστοιχο κουμπί αλλά δεν κατάφερα να το υλοποιήσω.

Υπάρχει μεγάλος χώρος για βελτίωση του project αλλά την παρούσα στιγμή είμαι ευχαριστημένος από το αποτέλεσμα.

ΑΝΑΦΟΡΕΣ

Σχήμα 1 Java EE 3-tier application - www.javacodegeeks.com/2014/03/java-ee-the-basics.html

Σχήμα 2 Αρχιτεκτονική: Containers & Components - www.javacodegeeks.com/2014/03/java-ee-the-basics.html

Σχήμα 3 - JSF project structure

Κώδικας 1: Adam Bien's Weblog - http://www.adam-bien.com/roller/abien/entry/selecting_all_jpa_entities_as

Persistence 1: Antonio Goncalves - Beginning with JavaEE 7, p.110 - 111

Κώδικας 2: Arun Gupta - https://github.com/javaee-samples/javaee7-samples/blob/master/ejb/stateless/src/main/java/org/javaee7/ejb/stateless/AccountSessionBean.java

http://www.itcuties.com/j2ee/jsf-2-download-file/

http://stackoverflow.com/questions/9391838/how-to-stream-a-file-download-in-a-jsf-backing-bean

http://alextretyakov.blogspot.gr/2013/07/jpa-many-to-many-mappings.html

http://www.java2s.com/Code/Java/JPA/ManyToManyJoinedTable.htm

http://stackoverflow.com/questions/2439594/how-do-i-set-the-value-of-htmloutputtag-in-jsf

http://stackoverflow.com/questions/15345349/create-inputtext-dynamically

http://stackoverflow.com/questions/15787053/how-to-bind-value-of-dynamically-created-htmlinputtext-component-to-bean-propert

http://stackoverflow.com/questions/11558379/jsf-binding-with-setvalueexpression-read-only

https://rahulganesh.wordpress.com/2011/07/06/jsfdynamicadditionofcomponents/

http://www.javadomain.in/dynamically-addremove-jsf-components-example/

http://stackoverflow.com/questions/3409053/jsf2-can-i-add-jsf-components-dynamically

http://java.dzone.com/articles/creating-jsf-pages-pure-java

http://www.codejava.net/java-ee/servlet/java-file-upload-example-with-servlet-30-api

http://www.coderanch.com/t/523364/JSF/java/Dynamic-JSF-programatically-adding-components

http://stackoverflow.com/questions/16022319/how-to-add-create-a-primefaces-progressbar-component-from-a-managed-bean

http://blog.c2b2.co.uk/2014/03/writing-and-deploying-simple-web.html

http://forum.primefaces.org/viewtopic.php?f=3&t=38525&p=121826&hilit=component+binding#p121826

http://forum.primefaces.org/viewtopic.php?f=3&t=35259&p=112698&hilit=component+binding#p112698

http://stackoverflow.com/questions/22502255/on-primeface-tab-view

http://www.coreservlets.com/JSF-Tutorial/primefaces/

http://www.coreservlets.com/JSF-Tutorial/jsf2/

http://forum.primefaces.org/viewtopic.php?f=3&t=39981

http://stackoverflow.com/questions/19064274/how-to-insert-a-primefaces-input-text-dynamically

http://stackoverflow.com/questions/7308535/how-to-dynamically-add-and-remove-a-tab-in-ptabview-component

http://stackoverflow.com/questions/15819137/dynamically-add-and-remove-panels-primeface

http://www.beyondjava.net/blog/how-to-write-a-dynamic-jsf-2-x-component/

http://www.oio.de/public/java/jsf-best-practices-javaserver-faces-session-tips.htm

http://en.wikibooks.org/wiki/Java_Persistence/JPQL

http://blog.jbaysolutions.com/2014/10/16/jpa-2-tutorial-queries-on-the-model/

http://blog.jbaysolutions.com/2012/12/17/jpa-2-relationships-many-to-many/

http://blog.jbaysolutions.com/2011/09/19/jpa-2-relationships-onetomany/

http://stackoverflow.com/questions/13485752/jpql-and-join-table

http://www.thejavageek.com/jpa-tutorials/

http://www.oracle.com/technetwork/articles/vasiliev-jpql-087123.html

http://stackoverflow.com/questions/7979382/how-to-create-join-table-with-jpa-annotations

http://javarevisited.blogspot.gr/2014/08/how-to-send-email-from-java-program-example.html

http://stackoverflow.com/questions/2422468/how-to-upload-files-to-server-using-jsp-servlet

https://netbeans.org/kb/docs/javaee/ecommerce/security.html#formBased

http://blog.eisele.net/2013/01/jdbc-realm-glassfish312-primefaces342.html

http://stackoverflow.com/questions/14758429/accessing-user-details-after-logging-in-with-java-ee-form-authentication

http://stackoverflow.com/questions/15022473/how-to-trigger-bean-after-j-security-check-complete-authentication-in-java

http://stackoverflow.com/questions/2206911/performing-user-authentication-in-java-ee-jsf-using-j-security-check

http://jj-blogger.blogspot.gr/2014/03/resource-library-contracts-with.html

http://stackoverflow.com/questions/22293708/how-to-add-tabs-in-tabview-in-primefaces-dynamically-on-click-of-a-command-but

http://stackoverflow.com/questions/7146625/how-to-group-radio-buttons-in-hdatatable-jsf2-0

https://github.com/jirkapinkas/example-contact-form

http://threadbarecanvas.azurewebsites.net/java-web/creating-a-javaweb-email-contact-form/

http://www.codejava.net/java-ee/jsp/sending-e-mail-with-jsp-servlet-and-javamail

http://www.javacodegeeks.com/2011/10/sending-emails-with-java.html

https://www.youtube.com/watch?v=GLvVnhFAvSY&ab_channel=JosephBernabeBagnes

https://www.youtube.com/watch?v=_zW27Y2boCo&ab_channel=JosephBernabeBagnes

http://www.javacodegeeks.com/2014/08/ejb-3-x-lifecycle-and-concurrency-models-part-1.html

https://netbeans.org/kb/docs/javaee/ecommerce/entity-session.html

http://www.adam-bien.com/roller/abien/entry/the_return_of_jsps_in

http://ramanathbhongalejsf.blogspot.gr/2013/04/create-login-page-with-jsf.html

https://www.youtube.com/watch?v=1xsU6juUZd0&ab_channel=JimLombardo

http://jugojava.blogspot.gr/2011/02/jdbc-security-realm-with-glassfish-and.html

http://java.dzone.com/articles/java-web-application-security

https://www.nabisoft.com/tutorials/glassfish/securing-java-ee-6-web-applications-on-glassfish-using-jaas

https://www.youtube.com/playlist?list=PLS1QulWo1RIbAiu7-bY4FpgYe3kzpC9tx

http://stackoverflow.com/questions/15055968/java-ee-simple-login

http://blog.mueller-bruehl.de/tutorial-web-development/

http://vladmihalcea.com/2014/07/30/a-beginners-guide-to-jpahibernate-entity-state-transitions/

http://ensode.net/roller/dheffelfinger/entry/javaserver_faces_jsf_in_a

https://www.youtube.com/user/koushks?&ab_channel=JavaBrains

http://javaknowledge.info/authentication-based-secure-login-logout-using-jsf-2-0-and-primefaces-3-4-1/

https://www.youtube.com/watch?v=vuwXxuCjOm0&ab_channel=SparkyGlassFish

http://opendevelopmentnotes.blogspot.gr/2014/07/javaee-from-zero-to-app-in-minutes.html

https://speakerdeck.com/ppapapetrou76/introduction-to-jee

http://www.javacodegeeks.com/2013/02/jdbc-realm-and-form-based-authentication-with-glassfish-3-1-2-2-and-primefaces-3-4.html

http://www.oracle.com/technetwork/articles/java/java-primefaces-2191907.html

http://zeroturnaround.com/rebellabs/how-to-use-jpa-correctly-to-avoid-complaints-of-a-slow-application/

https://code.google.com/p/javaee6-crud-example/

https://javaserverfaces.java.net/

http://java.dzone.com/articles/java-ee-7-whats-new

http://www.javacodegeeks.com/2013/05/java-ee-cdi-dependency-injection-inject-tutorial.html

http://www.reddit.com/r/java/comments/1ufagh/what_framework_do_you_recommend_to_build_web/

http://www.saltwebsites.com/2014/java-ee-notes

http://www.coreservlets.com/

http://en.wikipedia.org/wiki/JavaServer_Pages

http://en.wikipedia.org/wiki/JavaServer_Faces

http://en.wikipedia.org/wiki/Facelets

http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

http://en.wikipedia.org/wiki/Comparison_of_application_servers

https://netbeans.org/features/java-on-server/java-ee.html

http://en.wikibooks.org/wiki/Java_EE_Programming

http://www.oracle.com/technetwork/java/javaee/tech/index.html

http://www.javacodegeeks.com/2014/02/wildfly-8-vs-glassfish-4-which-application-server-to-choose.html

http://www.javacodegeeks.com/2014/03/java-enterprise-software-versus-what-it-should-be.html

http://www.javacodegeeks.com/2014/03/java-ee-7-whats-new.html

http://www.javacodegeeks.com/2014/03/java-ee-the-basics.html

http://www.javacodegeeks.com/2013/07/java-ee-ejb-interceptors-tutorial-and-example.html

http://www.javacodegeeks.com/2013/08/file-upload-example-in-servlet-and-jsp.html

http://www.javacodegeeks.com/2013/11/what-is-your-structure-of-jee-based-web-projects.html

http://www.slideshare.net/arungupta1/java-ee7-webinar

http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition

http://programming-simplified.com/java_ee_6_video_training.html

http://www.javacodegeeks.com/2013/02/introduction-to-javaee-concepts.html

ΒΙΒΛΙΟΓΡΑΦΙΑ

Civici, C. (2014), Primefaces 5.1 User Guide, http://www.primefaces.org

Evans, I. (2013), Your First Cup: An Introduction to the Java EE Platform, Volume 1, Release 7 for Java Platform, Enterprise Edition, Oracle.

Goncalves, A. (2013), Beginning Java EE 7, Apress, New York, USA.

Heffelfinger, D. (2014), Java EE 7 with Glassfish 4 Application Server, Packt Publishing, Birmingham, UK.

Jendrock, E., Cervera-Navarro, R., Evans, I., Haase, K., Markito, W. (2014), The Java EE 7 Tutorial, Oracle.

Juneau, J. (2013), Java EE 7 Recipes, Apress, New York, USA.

Keith, M., Schincariol, M., (2013) Pro JPA 2, Apress, New York, USA.

Leonard, A. (2014), Mastering JavaServer Faces 2.2, Packt Publishing, Birmingham, UK.

Wetherbee, J., Rathod, C., Kodali, R., Zadrozny, P. (2013), Beginning EJB 3, Apress, New York, USA.

ΠΑΡΑΡΤΗΜΑΤΑ

Παράρτημα 1: JSF κώδικας κεφαλαίου 5

Login.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:p="http://primefaces.org/ui">
    <h:head>
        <title>Login Page</title>
    </h:head>
    <h:body>
        <ui:composition template="./resources/basicTemplate.xhtml">

            <ui:define name="top">
                <h1>Please enter your credentials</h1>
            </ui:define>

            <ui:define name="left">
                <p:link outcome="/index" value="Home Page" />
                <br/>
                <br/>
                <p:link outcome="register.xhtml" value="New Account" />
                <br/>
                <br/>
            </ui:define>

            <ui:define name="right">
                <p:link outcome="admin/admin-main" value="Admin section"
/>
            </ui:define>

            <ui:define name="content">
                <h2>---Admin Sample Accounts---</h2>
                <ul>
                    <li>Username, password: vasouv, vasouv</li>
                    <li>Username, password: geo, geo</li>
                </ul>
                <h2>---User Sample Account---</h2>
                <ul>
                    <li>Username, password: maik, maik</li>
                </ul>

                <h:form id="login">
                    <h:panelGrid columns="2">
                        <h:outputLabel value="Username"/>
                        <h:inputText id="j_username"
value="#{authenticationJSFBean.username}"/>
                        <h:outputLabel value="Password"/>
                        <h:inputSecret id="j_password"
value="#{authenticationJSFBean.password}"/>
                        <h:commandButton id="login"
action="#{authenticationJSFBean.login()}" value="Login"/>
                    </h:panelGrid>
                </h:form>
            </ui:define>
```

```
        </ui:composition>
    </h:body>
</html>
```

AuthenticationController.java

```
/**
 * @author Josh Juneau - JavaEE 7 Recipes - Ch14.3 - Apress
 */
@Named(value = "authenticationJSFBean")
@SessionScoped
public class AuthenticationController implements Serializable {

    @EJB
    private AuthenticationEJB authenticationFacade;
    private String username;
    private User user;
    private boolean authenticated = false;
    private HttpSession session = null;

    public AuthenticationController() {}

    public HttpSession getSession() {
        FacesContext context = FacesContext.getCurrentInstance();
        HttpServletRequest request = (HttpServletRequest)
context.getExternalContext().getRequest();
        session = request.getSession();
        return session;
    }

    public String login() {
        authenticationFacade.setUser(getUser());
        boolean authResult = authenticationFacade.login();
        if (authResult) {
            this.authenticated = true;
            setUser(authenticationFacade.getUser());
            return "users/main.xhtml";
        } else {
            this.authenticated = false;
            setUser(null);
            return "loginerror";
        }
    }

    public String logout() {
        user = null;
        this.authenticated = false;
        FacesContext facesContext = FacesContext.getCurrentInstance();
        ExternalContext externalContext =
facesContext.getExternalContext();
        externalContext.invalidateSession();
        return "logout";
    }

    //GETTERS & SETTERS

    public String getUsername() {
        this.username = getUser().getUsername();
        return this.username;
```

```
    }

    public void setUsername(String username) {
        this.username = username;
        getUser().setUsername(username);
    }

    public String getPassword() {
        return authenticationFacade.getPassword();
    }

    public void setPassword(String password) {
        authenticationFacade.setPassword(password);
    }

    public User getUser() {
        if (this.user == null) {
            user = new User();
            setUser(authenticationFacade.getUser());
        }
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public void setAuthenticated(boolean authenticated) {
        this.authenticated = authenticated;
    }
}
```

Παράρτημα 2: ShoppingCart.java

```
package vasouv.javaee7thesis.shoppingcart;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;
import vasouv.javaee7thesis.courses.Course;

/**
 *
 * @author vasouv
 */
@Named("shoppingCart")
@SessionScoped
public class ShoppingCart implements Serializable {

    //Courses added to the shopping cart
    private List<Course> coursesToBuy;

    //Course being removed from the cart
    private Course selectedToRemove;

    public ShoppingCart() {
        coursesToBuy = new ArrayList();
    }

    public void addCourse(Course c) {
```

```
        coursesToBuy.add(c);
    }

    public void removeCourse() {
        coursesToBuy.remove(selectedToRemove);
    }

    public int getTotalPrice() {
        int total = 0;
        total = coursesToBuy.stream().map((c) ->
c.getPrice()).reduce(total, Integer::sum);
        return total;
    }

    public String buyCourses() {
        return "checkout";
    }

    public void clearCart() {
        coursesToBuy.clear();
    }

    // ----- GETTERS & SETTERS -----

    public List<Course> getCoursesToBuy() {
        return coursesToBuy;
    }

    public void setCoursesToBuy(List<Course> coursesToBuy) {
        this.coursesToBuy = coursesToBuy;
    }

    public Course getSelectedToRemove() {
        return selectedToRemove;
    }

    public void setSelectedToRemove(Course selectedToRemove) {
        this.selectedToRemove = selectedToRemove;
    }

}
```

Παράρτημα 3: CourseFacade.java

```
package vasouv.javaee7thesis.courses.sessionbeans;

import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import vasouv.javaee7thesis.courses.Course;
import vasouv.javaee7thesis.register.User;

/**
 *
 * @author vasouv
 */
@Stateless
public class CourseFacade extends AbstractFacade<Course> {
    @PersistenceContext(unitName = "vasouvPU")
    private EntityManager em;
```

```
    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public CourseFacade() {
        super(Course.class);
    }

    /**
     * Selects all courses.
     *
     * This method queries the DB and fetches a list of all available
courses.
     *
     * @return List containing all Course entities.
     */
    @Override
    public List<Course> findAll() {
        return em.createQuery("SELECT c FROM Course c").getResultList();
    }

    /**
     * Finds the Course MAX ID.
     *
     * @return Integer max id
     */
    public Integer findCourseMaxID() {
        return (Integer)em.createQuery("select max(c.idcourse) from
Course c").getSingleResult();
    }

    /**
     * Sets the relationship between Courses and Users.
     *
     * A course is retrieved from the DB with the specified courseID. The
user param gets added
     * to the Course and the EntityManager merges the entities. The
actual result is seen in the
     * USERCOURSES table.
     *
     * @param u User to be set to the Courses
     * @param courseID Integer to find the Course by ID
     */
    public void setCourseUser(User u, int courseID) {
        Course c =
(Course)em.createNamedQuery("Courses.findByIdcourse").setParameter("idcou
rse", courseID).getSingleResult();
        c.getUsers().add(u);
        em.merge(c);
    }

    /**
     * Finds the Courses for a specific User.
     *
     * @param usro The User's username
     * @return List of Courses
     */
    public List<Course> findCourseNamesByUser(String usro) {
```

```
        return em.createQuery("select c from Course c inner join c.users
us where us.username=:usr").setParameter("usr", usro).getResultList();
    }

}
```

Παράρτημα 4: CheckoutBean.java

```
package vasouv.javaee7thesis.checkout;

import java.io.Serializable;
import javax.annotation.PostConstruct;
import javax.ejb.EJB;
import javax.inject.Named;
import javax.enterprise.context.RequestScoped;
import javax.inject.Inject;
import vasouv.javaee7thesis.courses.sessionbeans.CourseFacade;
import vasouv.javaee7thesis.login.AuthenticationEJB;
import vasouv.javaee7thesis.register.User;
import vasouv.javaee7thesis.register.sessionbeans.UserFacade;
import vasouv.javaee7thesis.shoppingcart.ShoppingCart;

/**
 *
 * @author vasouv
 */
@Named("checkoutBean")
@RequestScoped
public class CheckoutBean implements Serializable {

    @Inject
    private ShoppingCart shoppingCart;

    @EJB
    private CourseFacade courseFacade;

    @EJB
    private UserFacade userFacade;

    //Retrieves the logged in User's username
    @EJB
    private AuthenticationEJB auth;

    //Shows the username in the View
    private String username;

    //Credit card number - simply a String
    private String creditCard;

    @PostConstruct
    public void init() {
        //Sets the username so it shows upon View time
        username = auth.getUser().getUsername();
    }

    public CheckoutBean() {
    }

    /**
     * Finalizes the purchase.
```

```
     *
     * @return Navigates to the Checkout Complete page.
     */
    public String finalizePurchase(){
        persistUserCourses();
        clearShoppingCart();
        return "checkoutcomplete";
    }

    /**
     * Sets the User - Courses relationship to the DB.
     *
     * A temp User is retrieved from the DB. For each of the Courses in
the shopping
     * cart, the relationship is persisted to the DB.
     */
    private void persistUserCourses() {

        //Retrieves the User entity from the DB
        User use = userFacade.findByUsernameSingle(username);

        //For every course in the cart, calls the method to persist the
users
        //Uses Java8 streams and Lambdas
        shoppingCart.getCoursesToBuy().stream().forEach((co1) -> {
            courseFacade.setCourseUser(use, co1.getIdcourse());
        });

    }

    /**
     * Clears the shopping cart.
     */
    private void clearShoppingCart() {
        shoppingCart.clearCart();
    }

    // --- GETTERS & SETTERS ---

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getCreditCard() {
        return creditCard;
    }

    public void setCreditCard(String creditCard) {
        this.creditCard = creditCard;
    }



}
```

Παράρτημα 5: CourseFacadeREST.java

```
@Stateless
@Path("vasouv.javaee7thesis.courses.course")
public class CourseFacadeREST extends AbstractFacade<Course> {
    @PersistenceContext(unitName = "vasouvPU")
    private EntityManager em;

    public CourseFacadeREST() {
        super(Course.class);
    }

    @POST
    @Override
    @Consumes({"application/xml", "application/json"})
    public void create(Course entity) {
        super.create(entity);
    }

    @PUT
    @Path("{id}")
    @Consumes({"application/xml", "application/json"})
    public void edit(@PathParam("id") Integer id, Course entity) {
        super.edit(entity);
    }

    @DELETE
    @Path("{id}")
    public void remove(@PathParam("id") Integer id) {
        super.remove(super.find(id));
    }

    @GET
    @Path("{id}")
    @Produces({"application/xml", "application/json"})
    public Course find(@PathParam("id") Integer id) {
        return super.find(id);
    }

    @GET
    @Override
    @Produces({"application/xml", "application/json"})
    public List<Course> findAll() {
        return super.findAll();
    }

    @GET
    @Path("{from}/{to}")
    @Produces({"application/xml", "application/json"})
    public List<Course> findRange(@PathParam("from") Integer from,
@PathParam("to") Integer to) {
        return super.findRange(new int[]{from, to});
    }

    @GET
    @Path("count")
    @Produces("text/plain")
    public String countREST() {
        return String.valueOf(super.count());
    }

    @Override
```

```
    protected EntityManager getEntityManager() {
        return em;
    }

}
```

Παράρτημα 6: MoodServlet.java

```
package javaeetutorial.mood;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/report")
public class MoodServlet extends HttpServlet {
    private static final long serialVersionUID = 18925377774889413L;

    /**
     * Processes requests for both HTTP <code>GET</code> and
<code>POST</code>
     *  methods.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<html lang=\"en\">");
            out.println("<head>");
            out.println("<title>Servlet MoodServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet MoodServlet at "
                    + request.getContextPath() + "</h1>");
            String mood = (String) request.getAttribute("mood");
            out.println("<p>Duke's mood is: " + mood + "</p>");
            switch (mood) {
                case "sleepy":
                    out.println("<img
src=\"resources/images/duke.snooze.gif\" alt=\"Duke sleeping\"/><br/>");
                    break;
                case "alert":
                    out.println("<img
src=\"resources/images/duke.waving.gif\" alt=\"Duke waving\"/><br/>");
                    break;
                case "hungry":
                    out.println("<img
src=\"resources/images/duke.cookies.gif\" alt=\"Duke with
cookies\"/><br/>");
                    break;
```

```
                case "lethargic":
                    out.println("<img
src=\"resources/images/duke.handsOnHips.gif\" alt=\"Duke with hands on
hips\"/><br/>");
                    break;
                case "thoughtful":
                    out.println("<img
src=\"resources/images/duke.pensive.gif\" alt=\"Duke thinking\"/><br/>");
                    break;
                default:
                    out.println("<img
src=\"resources/images/duke.thumbsup.gif\" alt=\"Duke with thumbs-up
gesture\"/><br/>");
                    break;
            }
            out.println("</body>");
            out.println("</html>");
        }
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods.
Click on the + sign on the left to edit the code.">
    /**
     * Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        processRequest(request, response);
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ

Για την εγκατάσταση και χρήση του project απαιτείται το παρακάτω λογισμικό:

- Java 8 JDK
- JavaDB 10.10.2.0
- JavaEE 7
- GlassFish 4.1 Full
- NetBeans 8.0.2

Η JavaDB περιέχεται στην εγκατάσταση του JDK ενώ η JavaEE 7 περιέχεται στον GlassFish. Default επιλογές κατά την εγκατάσταση του JDK και NetBeans (EE).

Λήψη του προ-ρυθμισμένου GlassFish μέσω του Dropbox μου και της βάσης δεδομένων. Λήψη όλου του project από το GitHub repository της εργασίας με επιλογή του "Download ZIP" στα δεξία της σελίδας.

Αποσυμπίεση του GlassFish στο C: και εγκατάσταση του NetBeans χωρίς την επιλογή για εκ νέου εγκατάσταση του GlassFish.

Σύνδεση του GlassFish στο NetBeans με επιλογή του Services window, δεξί κλικ στο Servers και GlassFish. Δείχνουμε το "C:/glassfish" χωρίς άλλες αλλαγές στην επόμενη σελίδα.

Τοποθέτηση του project στο "Τα έγγραφά μου" μες στον κατάλογο "NetBeansProjects" και της βάσης δεδομένων στο "C:/<user>/.netbeans-derby".

Με Build και Deploy του project, γίνεται αυτόματα εκκίνηση του browser κι εμφανίζεται η αρχική σελίδα του συστήματος.