ΘΕΜΑ:

ΑΝΑΛΥΣΗ, ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΔΗΜΙΟΥΡΓΙΑ ΗΛΕΚΤΡΟΝΙΚΟΥ ΚΑΤΑΣΤΗΜΑΤΟΣ

ΝΑΘΑΝΑΗΛ ΗΛΙΑΣ Α.Μ. : ΕΜ/07068

Πανεπιστήμιο Πειραιώς
Τμήμα Ψηφιακών Συστημάτων
Πρόγραμμα Μεταπτυχιακών Σπουδών
Κατεύθυνση :Ψηφιακών Επικοινωνιών & Δικτύων

Επιβλέπων καθηγητής :

Μαρίνος Θεμιστοκλέους

(Επίκουρος Καθηγητής)

ΠΕΙΡΑΙΑΣ 2010

ΕΥΧΑΡΙΣΤΙΕΣ

Επιθυμώ να ευχαριστήσω θερμά όλους εκείνους που συνέβαλαν με οποιονδήποτε τρόπο στην αποπεράτωση της εργασίας μου και ιδιαίτερες ευχαριστίες εκφράζω:

Στον Επίκουρο καθηγητή Μαρίνο Θεμιστοκλέους για τη επίβλεψη και την αμέριστη και αδιάκοπη βοήθεια που μου παρείχε ώστε να τελειώσει η παρούσα Διπλωματική. Επίσης θέλω να ευχαριστήσω την οικογένεια μου και την κοπέλα μου Κωνσταντίνα Παπαδοπούλου για την ψυχολογική υποστήριξη που μου παρείχαν κατά την διάρκεια της συγγραφής της.

Πίνακας περιεχομένων

1. Εισαγωγή

Η ανάγκη για το ηλεκτρονικό εμπόριο προέκυψε από την απαίτηση των επιχειρήσεων (για λόγους απόκτησης ανταγωνιστικού πλεονεκτήματος) και των κυβερνήσεων (για λόγους επικράτησης ανταγωνιστικότερων όρων λειτουργίας της αγοράς) για καλύτερη χρήση της τεχνολογίας των υπολογιστών και των τηλεπικοινωνιών ώστε να βελτιωθούν οι σχέσεις αμφίδρομης επικοινωνίας με τους πολίτες/καταναλωτές/πελάτες, οι επιχειρηματικές διεργασίες και η ανταλλαγή πληροφοριών ενδο-επιχειρησιακά, αλλά και κυρίως μεταξύ των επιχειρήσεων. Η τεχνολογία και ειδικότερα το ηλεκτρονικό εμπόριο παρέχει ευέλικτες και ολοκληρωμένες λύσεις προώθησης των επιχειρήσεων στις επιθυμητές αγορές (target markets) παρεμβαίνοντας ευεργετικά σε κάθε στάδιο της αλυσίδας της αξίας τους (value chain).

Το ηλεκτρονικό εμπόριο μπορεί να οριστεί ως ένα σύνολο επιχειρηματικών και διεπιχειρησιακών στρατηγικών καθώς και μοντέλων ολοκλήρωσης που μπορούν να υποστηρίξουν όλους τους τομείς επιχειρηματικής δραστηριότητας, μέσω της χρήσης νέων τεχνολογιών και κατάλληλων τηλεπικοινωνιακών υποδομών

Το ηλεκτρονικό εμπόριο έχει αναπτυχθεί ραγδαία τα τελευταία χρόνια. Στα πλαίσια της εργασίας αυτής εξετάζεται η ανάπτυξη και οι απαιτήσεις του Ηλεκτρονικού Εμπορίου. Γίνεται ιδιαίτερη αναφορά και επικέντρωση σε ένα ιδιαίτερο κομμάτι του Ηλεκτρονικού Εμπορίου, τις συναλλαγές μεταξύ διαφορετικών τμημάτων μιας επιχείρησης και τις συναλλαγές μεταξύ επιχειρήσεων (Business-to-Business – B2B).

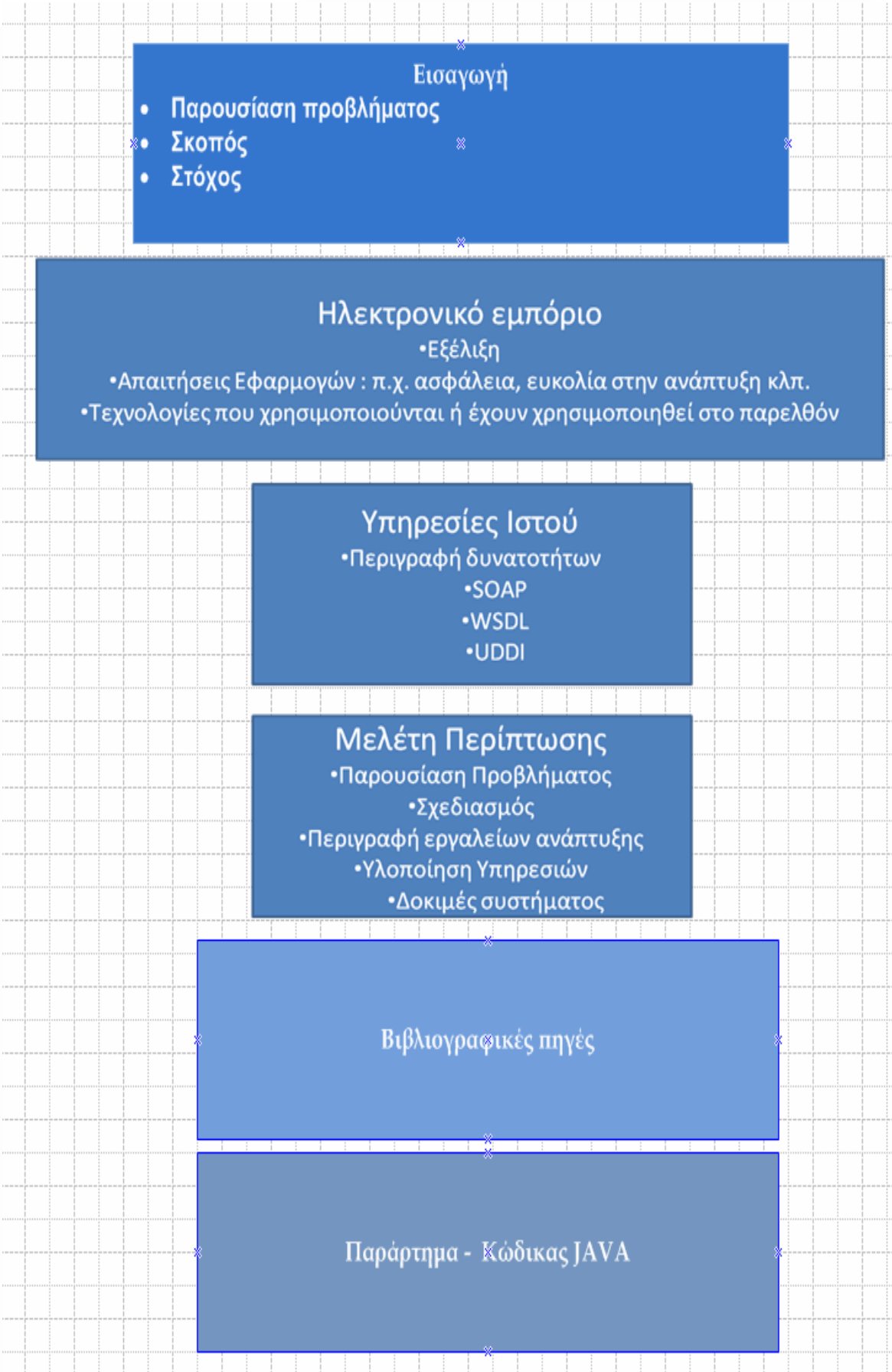
Θα γίνει βιβλιογραφική αναφορά για το Ηλεκτρονικό εμπόριο, τις κατηγορίες του και τις τεχνολογικές του απαιτήσεις. Στη συνέχεια, επικεντρώνοντας στο B2B, θα παρουσιαστούν διάφορες τεχνολογικές λύσεις και ειδικότερα η λύση των Υπηρεσιών Ιστού (Web Services) που φαίνεται να είναι η ενδεδειγμένη.

Θα γίνει ιδιαίτερα εκτενής παρουσίαση στις τεχνολογίες των Υπηρεσιών Ιστού, επιδεικνύοντας όλα τα πλεονεκτήματα που εμφανίζουν και καθιστούν τη χρήση συστημάτων B2B εύκολα στην ανάπτυξη, συντήρηση και χρήση.

Στα πλαίσια της εργασίας αυτής σχεδιάζεται και αναπτύσσεται το backend τμήμα ενός ηλεκτρονικού καταστήματος βασισμένο σε Web Services. Ο σχεδιασμός και ανάπτυξη του υποσυστήματος αυτού έχει στόχο να επιδείξει και να επαληθεύσει τα πλεονεκτήματα της χρήσης Υπηρεσιών Ιστού. Πιο συγκεκριμένα, αναπτύσεται ένα λογισμικό middleware το οποίο λειτουργεί με την αρχή του πελάτη-εξυπηρετητή, όπου τα διάφορα καταστήματα (front-end) μιλούν με τη χρήση του Web Service Client στο κεντρικό κατάστημα όπου βρίσκεται το Web Service και η βάση δεδομένων (back-end). Το Web Service αποθηκεύει και ανακτά πληροφορίες από τη βάση δεδομένων σύμφωνα με τις κλήσεις από τους Web Service Clients.

Το πρόβλημα που καλείται να λύσει το λογισμικό που θα αναπτυχθεί είναι η ασφαλής, αξιόπιστη και ταχύτατη επικοινωνία απομακρυσμένων συστημάτων μιας επιχείρησης. Επίσης, η εταιρία θέλει να αναπτύξει διεπαφές για την ανάπτυξη B2B δραστηριότητας.

Η δομή της εργασίας συνοψίζεται στο παρακάτω διάγραμμα:


Εισαγωγή
• Παρουσίαση προβλήματος
• Σκοπός
• Στόχος
Ηλεκτρονικό εμπόριο
•Εξέλιξη
•Απαιτήσεις Εφαρμογών : π.χ. ασφάλεια, ευκολία στην ανάπτυξη κλπ.
•Τεχνολογίες που χρησιμοποιούνται ή έχουν χρησιμοποιηθεί στο παρελθόν
Υπηρεσίες Ιστού
•Περιγραφή δυνατοτήτων
•SOAP
•WSDL
•UDDI
Μελέτη Περίπτωσης
•Παρουσίαση Προβλήματος
•Σχεδιασμός
•Περιγραφή εργαλείων ανάπτυξης
•Υλοποίηση Υπηρεσιών
•Δοκιμές συστήματος
Βιβλιογραφικές πηγές
Παράρτημα - Κώδικας JAVA

2. Ηλεκτρονικό Εμπόριο

Το ηλεκτρονικό εμπόριο έχει αναπτυχθεί ραγδαία τα τελευταία χρόνια. Ως ηλεκτρονικό εμπόριο ορίζεται κάθε είδος εμπορικής συναλλαγής μεταξύ προσώπων (φυσικών και μη) που πραγματοποιείται με ηλεκτρονικά μέσα. Πρόκειται για την αγοραπωλησία αγαθών, πληροφοριών και υπηρεσιών μέσα από δίκτυα ηλεκτρονικών υπολογιστών, χωρίς την ανάγκη για φυσική παρουσία και χωρίς τη χρήση χαρτιού.

Το ηλεκτρονικό εμπόριο μπορεί επίσης να περιγραφεί κάτω από τέσσερις διαφορετικές οπτικές, δίνοντας βάση κάθε φορά στους εξής παράγοντες: επιχειρήσεις, υπηρεσίες, απόσταση, επικοινωνία. Πιο συγκεκριμένα, επηρεάζει δραματικά τον τρόπο λειτουργίας των επιχειρήσεων, καθώς η εφαρμογή του αυτοματοποιεί τις συναλλαγές ανάμεσα σε εταιρίες και πελάτες, αλλά και ανάμεσα σε ξεχωριστές εταιρίες που συμμετέχουν στην αλυσίδα παραγωγής ενός προϊόντος ή υπηρεσίας. Παράλληλα, αυτοματοποιεί σε μεγάλο βαθμό τη ροή εργασιών και μέσα στην ίδια την εταιρία. Το ηλεκτρονικό εμπόριο μπορεί να περιγραφεί και ως μηχανισμός παροχής καλύτερης ποιότητας υπηρεσιών καθώς προσφέρει μεγαλύτερη ταχύτητα εκτέλεσης συναλλαγών και (συνήθως) μικρότερο κόστος. Παράλληλα δίνει τη δυνατότητα αγοραπωλησίας προϊόντων και υπηρεσιών μέσω του Διαδικτύου που θα ήταν αδύνατο να γίνουν διαφορετικά λόγω της γεωγραφικής απόστασης.

2.1. Πρώιμη εξέλιξη

Η έννοια του ηλεκτρονικού εμπορίου έχει αλλάξει τα τελευταία 30 χρόνια. Αρχικά, το ηλεκτρονικό εμπόριο σήμαινε τη διευκόλυνση των εμπορικών συναλλαγών ηλεκτρονικά, μέσω τεχνολογίας όπως το Electronic Data Interchange (EDI) και την ηλεκτρονική μεταφορά χρημάτων (Electronic Funds Transfer - EFT). Αυτές οι τεχνολογίες εισήχθησαν στα τέλη της δεκαετίας του 1970, επιτρέποντας στις επιχειρήσεις να αποστέλλουν έγγραφα όπως εντολές αγοράς ή τιμολογίων με ηλεκτρονικά μέσα. Η ανάπτυξη και η αποδοχή των πιστωτικών καρτών, των ATM και των τραπεζικών συναλλαγών μέσω τηλεφώνου (telephone banking) κατά τη δεκαετία του 1980 ήταν επίσης μορφές ηλεκτρονικού εμπορίου. Μια άλλη μορφή του ηλεκτρονικού εμπορίου ήταν το σύστημα κράτησης θέσεων αεροπορικών εταιρειών.

Το Online shopping , μια μορφή ηλεκτρονικού εμπορίου, είναι προγενέστερη του Παγκόσμιου Ιστού (WWW). Το 1979 ο Michael Aldrich , ένας Άγγλος εφευρέτης, σύνδεσε με τροποποιημένη έγχρωμη τηλεόραση 26 " με έναν υπολογιστή επεξεργασίας συναλλαγών σε πραγματικό χρόνο, μέσω μιας εσωτερικής τηλεφωνικής γραμμής και εφηύρε τις ηλεκτρονικές αγορές [1]. Ο Aldrich εφηύρε τόσο το online σύστημα αγορών όσο και την επιχειρηματική λογική για τη χρήση του. Το σύστημα του και οι ιδέες του αντιγράφηκαν από συστήματα του 1980 ήταν τόσο γρήγορα όσο σύγχρονα συστήματα αγορών στο internet. Χρησιμοποιούσαν dial-up και μισθωμένες τηλεφωνικές γραμμές καθώς τεχνολογία για ευρυζωνικές συνδέσεις δεν ήταν διαθέσιμη. Δεν πατεντάρισε ποτέ το σύστημά του και οι ιδέες του αποτελούν τη βάση των αγορών στο internet.

Από τη δεκαετία του 1990 και μετά, το ηλεκτρονικό εμπόριο περιλαμβάνει επιπλέον Enterprise Resource Planning συστήματα (ERP), καθώς εξόρυξη δεδομένων (data mining και data warehousing). Το 1990, ο

Tim Berners-Lee εφεύρε τον περιηγητή ιστού (web browser) και οδήγησε στη δημιουργία του Παγκόσμιου Ιστού και της διάδοσης του Διαδικτύου [2]. Η δημιουργία εμπορικών επιχειρήσεων στο Διαδίκτυο ήταν αυστηρά απαγορευμένη μέχρι το 1991 [3]. Παρόλο που το Διαδίκτυο έγινε δημοφιλές σε όλο τον κόσμο γύρω στο 1994, χρειάστηκαν αρκετά χρόνια ώστε να δημιουργηθούν πρωτόκολλα ασφάλειας και τη χρήση του DSL που επιτρέπει τη συνεχή σύνδεση στο Internet για την ανάπτυξη του ηλεκτρονικού εμπορίου. Τη τελευταία δεκαετία, το ηλεκτρονικό εμπόριο έχει αναπτυχθεί ιδιαίτερα, καθώς άρχισε να σχετίζεται με την ικανότητα της αγοράς διαφόρων αγαθών μέσω του Διαδικτύου χρησιμοποιώντας ασφαλή πρωτόκολλα και υπηρεσίες ηλεκτρονικών πληρωμών.

2.2. Μορφές Ηλεκτρονικού Εμπορίου

Επιγραμματικά οι μορφές Ηλεκτρονικού εμπορίου είναι οι Business to Consumer και Business to Business. Η μορφή Business to Consumer - B2C περιγράφει τις δραστηριότητες των επιχειρήσεων που εξυπηρετούν τους τελικούς καταναλωτές με προϊόντα ή / και υπηρεσίες. Η Business to Business - B2B περιγράφει εμπορικές συναλλαγές μεταξύ επιχειρήσεων, όπως είναι μεταξύ ενός παραγωγού και ενός χονδρεμπόρου , ή μεταξύ ενός πωλητή χονδρικής και ενός λιανοπωλητή.

Ένα παράδειγμα μιας συναλλαγής B2C θα ήταν ένα άτομο που αγοράζει ένα ζευγάρι παπούτσια από τον λιανοπωλητή. Οι συναλλαγές που οδήγησαν στην παπούτσια είναι διαθέσιμα για την αγορά, δηλαδή η αγορά των δερμάτινων ειδών, τα κορδόνια, καουτσούκ, κλπ. καθώς και την πώληση του παπουτσιού από το τσαγκάρη στον λιανοπωλητή θα μπορούσε να θεωρηθεί συναλλαγή Business to Business (B2B).

2.3. Απαιτήσεις Εφαρμογών Ηλεκτρονικού Εμπορίου

Με την ανάπτυξη του ηλεκτρονικού εμπορίου δημιουργούνται αντίστοιχα διάφορες απαιτήσεις/προβλήματα που καλούνται να δώσουν οι διαθέσιμες τεχνολογικές λύσεις. Επιγραμματικά οι τεχνολογικές απαιτήσεις που έχει το Ηλεκτρονικό Εμπόριο είναι οι παρακάτω.

2.3.1. Λειτουργικότητα και αμεσότητα εφαρμογών

Η πρώτη απαίτηση είναι η ευκολία στη χρήση. Ο χρήστης για να προτιμήσει τις ηλεκτρονικές συναλλαγές από τη φυσική παρουσία πρέπει να έχει συγκεκριμένα οφέλη από τη χρήση. Οι εφαρμογές πρέπει να είναι απλές στη χρήση, να χρησιμοποιούν όσο το δυνατόν λιγότερο εξειδικευμένο λογισμικό. Στην τεχνολογία λογισμικού, μια εφαρμογή Ιστού είναι μια εφαρμογή που προσεγγίζεται μέσω ενός φυλλομετρητή ιστοσελίδων (web browser) πάνω από το δίκτυο. Ένας φυλλομετρητής ιστοσελίδων είναι ένα λογισμικό που επιτρέπει στον χρήστη του να προβάλλει, και να αλληλεπιδρά με, κείμενα, εικόνες, βίντεο, μουσική, παιχνίδια και άλλες πληροφορίες συνήθως αναρτημένες σε μια ιστοσελίδα ενός ιστότοπου στον Παγκόσμιο Ιστό ή σε ένα τοπικό δίκτυο. Το κείμενο και οι εικόνες σε μια ιστοσελίδα μπορεί να περιέχουν συνδέσμους προς άλλες ιστοσελίδες του ίδιου ή διαφορετικού ιστότοπου. Ο φυλλομετρητής επιτρέπει στον χρήστη την γρήγορη και εύκολη πρόσβαση σε πληροφορίες που βρίσκονται σε διάφορες ιστοσελίδες και ιστότοπους εναλλάσσοντας τις ιστοσελίδες μέσω συνδέσμων (links). Η κύρια γλώσσα που χρησιμοποιείται από τις εφαρμογές ιστού και τους φυλλομετρητές είναι η γλώσσα μορφοποίησης HTML για την προβολή των ιστοσελίδων. Για την ανάπτυξη εφαρμογών που χρειάζονται περισσότερες διαδραστικά χαρακτηριστικά και δυνατότητες χρησιμοποιούνται και άλλες

γλώσσες προγραμματισμού παράλληλα με την HTML όπως η JavaScript η Java και η PHP.

Οι εφαρμογές Ιστού είναι δημοφιλείς επειδή δεν υπάρχει σύγχρονο λειτουργικό σύστημα που προορίζεται για υπολογιστές γραφείου που να μην έχει προ-εγκατεστημένο έναν ή περισσότερους φυλλομετρητές. Οι φυλλλομετρητές ουσιαστικά αποτελούν λογισμικό πελάτη του δικτυακού πρωτοκόλλου επιπέδου εφαρμογών HTTP. Για κάθε web browser διατίθενται, επίσης, και αρκετά πρόσθετα στοιχεία (add-ons), με στόχο την επαύξηση των δυνατοτήτων τους, τη βελτίωση της χρηστικότητας τους και την προστασία του χρήστη σε θέματα ασφάλειας. και η ευκολία της χρησιμοποίησης μιας μηχανής αναζήτησης Ιστού ως πελάτη, αποκαλούμενη μερικές φορές λεπτό πελάτη. Η δυνατότητα να ενημερωθούν και να διατηρηθούν οι εφαρμογές Ιστού χωρίς τη διανομή και εγκατάσταση του λογισμικού ενδεχομένως σε χιλιάδες υπολογιστές πελατών είναι ένας βασικός λόγος για τη δημοτικότητά τους.

2.3.2. Ασφάλεια

Υπάρχουν ιδιαίτερα αυστηρές απαιτήσεις στην ασφάλεια. Σε γενικές γραμμές υπάρχουν τρία διαφορετικά θέματα που αφορούν την ασφάλεια στην επικοινωνία ανάμεσα σε δυο απομακρυσμένα άκρα.

Εμπιστευτικότητα

Εάν ένας πελάτης στέλνει ένα XML αίτημα σε έναν εξυπηρετητή, το ερώτημα είναι πως μπορεί να διασφαλιστεί ότι η επικοινωνία παραμένει εμπιστευτική. Μια συνιθισμένη λύση στο πρόβλημα αυτό, στο χώρο των τεχνολογιών Ιστού, είναι το SSL (Secure Sockets Layer). Το SSL είναι ευρέως διαδεδομένο, χρησιμοποιείται πάνω από το HTTP, για τη δημιουργία συμμετρικής κρυπτογράφησης των πακέτων HTTP, αλλά όπως θα φανεί και στη συνέχεια δεν είναι πάντα η πιο αποδοτική λύση.

Πιστοποίηση

Εάν ένας πελάτης συνδέεται σε μια Υπηρεσία Ιστού, το ερώτημα είναι πώς μπορεί αρχικά να γίνει ταυτοποίηση του χρήστη και στη συνέχεια να ελεγχθεί αν ο συγκεκριμένος χρηστης δικαιούται πρόσβαση στη συγκεκριμένη υπηρεσία.

Μια μερική λύση είναι η αναγνώριση μέσω HTTP. Το HTTP περιλαμβάνει ενσωματωμένη υποστήριξη για την ανανγώριση της ταυτότητας ενός χρήστη.

Δικτυακή ασφάλεια

Η παροχή υπηρεσιών πάνω από το Διαδίκτυο δίνει τη δυνατότητα σε απομακρυσμένους πελάτες να καλέσουν (έστω και ελεγχόμενα) εντολές και διαδικασίες σε συστήματα, παρακάμπτοντας firewalls και συστήματα ασφαλείας. Στο πρόβλημα αυτό δεν υπάρχει απλή λύση, καθώς η ανάγκη για παροχή εξελιγμένων και πολύπλοκων υπηρεσιών και η ανάγκη για ασφάλεια πολλές φορές είναι αντίρροπες δυνάμεις στο σχεδιασμό ενός συστήματος.

2.3.3. Ευκολία στην ανάπτυξη και Επαναχρησιμοποίηση κώδικα

Ειδικά για εφαρμογές τύπου B2B υπάρχει συχνά η απαίτηση για την επαναχρησιμοποίηση κώδικα, και την ανταλλαγή δεδομένων που βασίζονται σε Legacy εφαρμογές. Legacy εφαρμογές είναι αυτές οι εφαρμογές οι οποίες βασίζονται σε ξεπερασμένες τεχνολογίες και συστήματα, είναι δύσκολο ή αδύνατο να αναβαθμιστούν ή να προστεθεί σε αυτές νέα λειτουργικότητα, αλλά είναι απαραίτητο να λειτουργούν ακόμα. Συχνές περιπτώσεις τέτοιων εφαρμογών εμφανίζονται σε μεγάλα κεντρικά συστήματα όπως σε τράπεζες. Για τις εφαρμογές αυτές έχει δαπανηθεί μεγάλο ποσό στο παρελθόν και λόγω του μεγέθους και της κρισιμότητας είναι δύσκολο να αναβαθμιστούν. Συνεπώς, προκειμένου να συνεχίσουν να χρησιμοποιούνται οι Legacy

εφαρμογές αποδοτικά και σε καινούρια πλαίσια, είναι βολικό πολλές φορές να καλούνται απομακρυσμένα και ελεγχόμενα από σύγχρονες τεχνολογίες μεταφοράς δεδομένων για λόγους ασφάλειας και ευχρηστίας.

2.4. Τεχνολογίες για Service Oriented Architecture

Η προσανατολισμένη στις υπηρεσίες αρχιτεκτονική (Service Oriented Architecture - SOA) είναι μια αρχιτεκτονική προσέγγιση με στόχευση στις επιχειρήσεις που υποστηρίζει την ενσωμάτωση επιχειρησιακού λογισμικού σε συνδεμένες υπηρεσίες [4]. Η αρχιτεκτονική SOA διαχωρίζει τις λειτουργίες σε ευδιάκριτες μονάδες, ή υπηρεσίες (υπηρεσίες Ιστού), οι οποίες είναι προσιτές πάνω από το δίκτυο έτσι ώστε μπορούν να συνδυαστούν και να επαναχρησιμοποιηθούν στην παραγωγή επιχειρηματικών εφαρμογών. Οι υπηρεσίες SOA και Ιστού είναι δύο διαφορετικές έννοιες, αλλά οι υπηρεσίες Ιστού είναι ο προτιμότερος και εξαιρετικά διαδεδομένος τρόπος εφαρμογής SOA. Η αρχιτεκτονική SOA είναι ένα καυτό θέμα στο επιχειρησιακό λογισμικό επειδή η χρήση της μπορεί να οδηγήσει στην οικοδόμηση (ή την επαναχρησιμοποίηση) εφαρμογών και συστημάτων καθιστώντας τα με τον τρόπο αυτό οικονομικά αποδοτικά. Τα σημαντικότερα πλεονεκτήματα της χρησιμοποίησης SOA απεικονίζονται [4][5] και συνοψίζονται στη συνέχεια:

Δυνατότητα επαναχρησιμοποίησης

Ίσως το σημαντικότερο προτέρημα της αρχιτεκτονικής SOA είναι η δυνατότητα να επαναχρησιμοποιηθούν υπάρχουσες υπηρεσίες. Οι υπεύθυνοι για την ανάπτυξη μέσα σε μια επιχείρηση και ανάμεσα σε επιχειρήσεις (ιδιαίτερα, στις επιχειρησιακές συνεργασίες) μπορούν να επαναχρησιμοποιήσουν τον υπάρχοντα κώδικα που αναπτύσσεται για τις επιχειρηματικές εφαρμογές, προσαρμόζοντάς τον ως υπηρεσίες Ιστού ώστε να καλυφθούν οι νέες επιχειρησιακές απαιτήσεις. Αυτή η διαδικασία μπορεί να

οδηγήσει σε σημαντική αποταμίευση στη ανάπτυξη εφαρμογών από την άποψη του χρόνου και του κόστους. Το όφελος της δυνατότητας επαναχρησιμοποίησης κώδικα γίνεται σημαντικότερο δεδομένου ότι όλο και περισσότερες εφαρμογές πρέπει να δημοσιευθούν ως υπηρεσίες Ιστού στο Διαδίκτυο. Ένα σοβαρό πρόβλημα σε αυτήν την περίπτωση είναι ότι αυτές οι υπάρχουσες εφαρμογές μπορούν να χτιστούν στα διάφορα συστήματα και να αναπτυχθούν κάτω από τις ετερογενείς πλατφόρμες λογισμικού.

Μερικές φορές, ακόμη και διαφορετικά τμήματα μέσα στην ίδια επιχείρηση, μπορούν να έχουν εφαρμογές με τα μοναδικά χαρακτηριστικά που τρέχουν σε διαφορετικά λειτουργικά περιβάλλοντα, να έχουν αναπτυχθεί σε διαφορετικές γλώσσες, χρησιμοποιούν διαφορετικές προγραμματιστικές διεπαφές και ενδεχομένως δυαδικά πρωτόκολλα που είναι κλειστά (proprietary, platform specific). Στην αρχιτεκτονική SOA και (στις υπηρεσίες Ιστού συγκεκριμένα), το μόνο χαρακτηριστικό που απαιτείται είναι μια δημόσια διεπαφή (interface) της υπηρεσίας, έτσι η παροχή της πρόσβασης στις λειτουργίες ενός συστήματος, -ακόμη και σε ένα απαρχαιωμένο (legacy system) - γίνεται ανώδυνη.

Διαλειτουργικότητα

Ένα άλλο σημαντικό πλεονέκτημα της αρχιτεκτονικής SOA είναι η διαλειτουργικότητα. Με άλλα λόγια, ο στόχος είναι να υπάρχει αδιαφανής και απροβλημάτιστη επικοινωνία ανάμεσα σε πελάτες ανεξάρτητα από το πόσο ετερογενείς είναι. Αυτός ο στόχος μπορεί να επιτευχθεί μόνο εάν οι πελάτες και οι υπηρεσίες έχουν έναν τυποποιημένο τρόπο επικοινωνίας ο ένας με τον άλλον. Στην πραγματικότητα, οι υπηρεσίες Ιστού προσφέρουν ένα σύνολο πρωτοκόλλων και τεχνολογιών που γίνονται αποδεκτά ευρέως και που είναι ανεξάρτητα πλατφόρμας, συστήματος, και γλώσσας προγραμματισμού.

Ευελιξία

Η αρχιτεκτονική SOA παρέχει ευελιξία επειδή οι υπηρεσίες Ιστού που απαιτούνται από μια συγκεκριμένη εφαρμογή συνδέονται με χαλαρούς δεσμούς. Σε μια αρχιτεκτονική όπου οι διαθέσιμες λειτουργίες συνδέονται στενά, διαφορετικά τμήματα (components) μιας εφαρμογής είναι έντονα συνδεδεμένα το ένα στο άλλο δεδομένου ότι μοιράζονται βιβλιοθήκες, ενότητες κώδικα και συχνά ακόμη και την κατάσταση. Αυτές οι εξαρτήσεις αποτρέπουν τις εφαρμογές από να εξελιχθούν ανάλογα με τις μεταβαλλόμενες επιχειρησιακές απαιτήσεις. Εντούτοις, φύση των υπηρεσιών Ιστού, η οποία είναι βασισμένη στη γλώσσα XML, έχει χαλαρούς δεσμούς ανάμεσα σε διαφορετικές λειτουργίες και στερείται κατάστασης, προσφέρει ευελιξία στις εφαρμογές, και τη δυνατότητα να εξελιχθούν με τις τρέχουσες απαιτήσεις.

2.4.1. Ηλεκτρονική Ανταλλαγή Δεδομένων (EDI - Electronic Data Interchange)

Το EDI δημιουργήθηκε στις αρχές της δεκαετίας του '70 και είναι μια κοινή δομή αρχείων που σχεδιάστηκε ώστε να επιτρέψει σε μεγάλους οργανισμούς να μεταδίδουν πληροφορίες μέσα από μεγάλα ιδιωτικά δίκτυα [6]. Πρόκειται για την ηλεκτρονική ανταλλαγή εμπορικών και διοικητικών δεδομένων από υπολογιστή σε υπολογιστή, με την ελάχιστη παρέμβαση χειρόγραφων διαδικασιών. Τα δεδομένα αυτά είναι οργανωμένα σε αυτοτελή μηνύματα (τιμολόγια, παραγγελίες, τιμοκατάλογοι, φορτωτικές κλπ.), το περιεχόμενο και η δομή των οποίων καθορίζονται από κάποιο κοινώς αποδεκτό πρότυπο. Τα πρότυπα που χρησιμοποιούνται σε παγκόσμιο επίπεδο προέρχονται από τον Οργανισμό Ηνωμένων Εθνών και καλύπτουν ένα ευρύ φάσμα επικοινωνιακών αναγκών των εμπορικών εταιριών.

Το πρότυπο αυτό είναι το EDIFACT (EDI For Administration, Commerce and Transportation).

2.4.2. Corba

Η Corba είναι ένα σύνολο από πρότυπα και πρωτόκολλα για σχεδιασμένη για την παροχή επικοινωνία μεταξύ διεργασιών σε ετερογενή περιβάλλοντα [7]. Εμφανίστηκε το 1994.

Στόχος ήταν με την CORBA οι προγραμματιστές να μπορούν εύκολα να "συνδέσουν" εφαρμογές που τρέχουν σε διαφορετικές μηχανές, με διαφορετικά λειτουργικά συστήματα και είναι γραμμένες σε διαφορετικές γλώσσες προγραμματισμού. Πιο συγκεκριμένα η Corba ορίζει προδιαγραφές για τον καθορισμό των συμβάσεων και τα πρωτόκολλων που πρέπει να ακολουθούνται από τις υλοποιήσεις της. Υποστηρίζονται μια πληθώρα από λειτουργικά συστήματα (UNIX, Windows,...) και γλώσσες προγραμματισμού (JAVA, C++, C, Ada, Lisp ...). Η ιδέα πίσω από τη χρήση της είναι ότι ο συγκεκριμένος τρόπος που θα υλοποιηθούν οι προδιαγραφές επαφίεται στους κατασκευαστές.

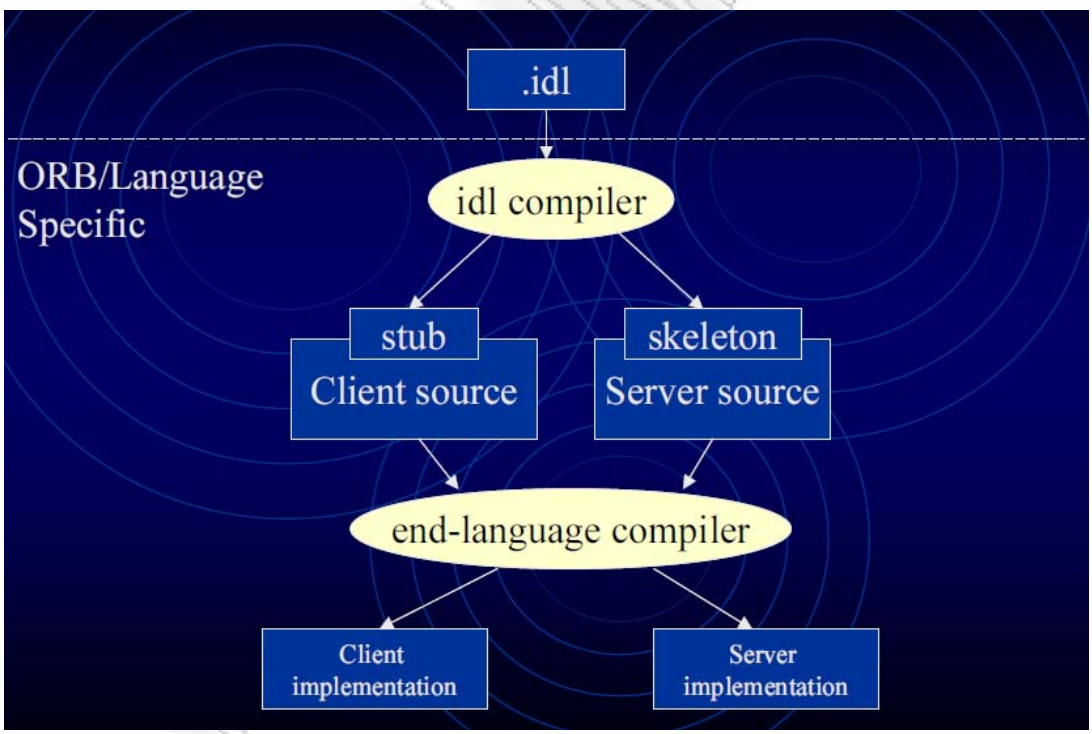
Η αρχιτεκτονική της Corba βασίζεται σε αντικείμενα. Πιο συγκεκριμένα, μια εφαρμογή CORBA χτίζεται από αντικείμενα (Objects). Τα αντικείμενα μπορούν να βρίσκονται διεσπαρμένα σε διάφορες μηχανές με κάθε ένα να υλοποιεί μια συγκεκριμένη σαφώς καθορισμένη λειτουργία. Η διαχείριση των αντικειμένων γίνεται με το ORB (Object Request Broker) το οποίο είναι η καρδιά ενός συστήματος CORBA και αποτελεί τον μεσάζοντα μεταξύ των αντικειμένων. Μάλιστα, προκειμένου να υπάρχει αποδοτικότερη κατανομή των πόρων, προβλέπεται πολλαπλά ORB να επικοινωνούν σε ένα κατανεμημένο περιβάλλον.

Οι υπηρεσίες που προσφέρονται μέσω της Corba προσφέρουν στα αντικείμενα έτοιμη λειτουργικότητα ενώ η υλοποίησή τους από ένα ORB είναι προαιρετική. Οι υπηρεσίες που παρέχονται από τη Corba είναι ονομαστικά οι παρακάτω: Naming, Event, Life Cycle, Trader, Transaction, Concurrency, Security, Persistence, Externalization, Query, Collection, Relationship, Time,

Properties, Licensing. Οι βασικές υπηρεσίες που συνήθως παρέχονται είναι οι Naming, Event και Trader.

Η Επικοινωνία μεταξύ ORB γίνεται με το πρωτόκολλο General Inter-Orb Protocol (GIOP). Οι παράμετροι και τα αποτελέσματα μεταφέρονται μεταξύ των αντικειμένων, σε μηνύματα GIOP χρησιμοποιώντας το πρωτόκολλο Common Data Representation (CDR). Η μετάδοση μηνυμάτων GIOP γίνεται μέσω TCP/IP.

Η γλώσσα που χρησιμοποιείται για τον καθορισμό των μεθόδων και των παραμέτρων που περικλείει ένα αντικείμενο είναι η Interface Definition Language (IDL). Είναι ανεξάρτητη της γλώσσα προγραμματισμού που θα χρησιμοποιηθεί τελικά για την υλοποίηση και έχει παρόμοια δομή με C++, Java. Η εφαρμογή γράφεται στη γλώσσα IDL και στη συνέχεια γίνεται compile με ειδικό compiler. Για κάθε γλώσσα προγραμματισμού και για κάθε ORB υπάρχει ένας συγκεκριμένος IDL compiler. Η διαδικασία αυτή περιγράφεται στο παρακάτω σχήμα:



Σχήμα 1 Χρήση της IDL

ΠΗΓΗ: The official CORBA standard from the OMG group"

http://www.itl.nist.gov/fipspubs/fip161-2.htm

2.4.3. Java RMI

Η τεχνολογία Απομακρυσμένης Επίκληση Μεθόδων (Java RMI) [8], επιτρέπει στον προγραμματιστή να δημιουργήσει κατανεμημένες Java εφαρμογές, στις οποίες οι μέθοδοι των απομακρυσμένων αντικειμένων Java μπορεί να καλεστούν από διαφορετικά Java virtual machines (JVM), που μπορεί να φιλοξενούνται σε διαφορετικά συστήματα. Το RMI χρησιμοποιεί object serialization για την αποστολή και λήψη παραμέτρων προσφέροντας ευκολία στην ανταλλαγή δεδομένων ανάμεσα σε απομακρυσμένα συστήματα.

Το Java RMI επιτρέπει σε εφαρμογές την κλήση μεθόδων από απομακρυσμένα αντικείμενα. Για τη χρήση του RMI Απαιτείται η ύπαρξη ενός RMI client και ενός RMI server, ενώ παρέχεται μηχανισμός επικοινωνίας ανάμεσα στον Client και στον Server. Με τον τρόπο αυτό, δίνεται η δυνατότητα σε οποιοδήποτε αντικείμενο της Java να χρησιμοποιηθεί, ακόμα και αν ο RMI server δεν γνωρίζει εκ των προτέρων για την ύπαρξή του.

Πιο συγκεκριμένα, μια τυπική εφαρμογή εξυπηρέτη δημιουργεί απομακρυσμένα αντικείμενα, αναφορές σε αυτά ώστε να είναι προσβάσιμα και περιμένει τους πελάτες να καλέσουν μεθόδους πάνω σε αυτά. Παράλληλα, μια τυπική εφαρμογή πελάτη καλεί μεθόδους πάνω σε απομακρυσμένα αντικείμενα μέσω μιας απομακρυσμένης αναφοράς.

Για την εύρεση απομακρυσμένων αντικειμένων γίνεται χρήση του ενός κεντρικού καταλόγου (Registry). Πιο συγκεκριμένα, το RMI προσφέρει ένα απλό σχήμα ονοματοδοσίας, στο οποίο ένα απομακρυσμένο αντικείμενο δίνει στον εαυτό του ένα όνομα όταν τρέχει για πρώτη φορά. Στη συνέχεια καταχωρείται στο RMI registry με μια διαδικασία εγγραφής. Το registry συνδέει το όνομα του αντικειμένου (όχι το όνομα της κλάσης) και το ίδιο το αντικείμενο. Στη Java, όταν ένα απομακρυσμένο αντικείμενο εγγράφεται στο registry μίας συγκεκριμένης μηχανής, συνδέται με ένα αντικείμενο ονοματοδοσίας. Αν ένα πελάτης θέλει να χρησιμοποιήσει ένα αντικείμενο, το

οποίο βρίσκεται σε έναν απομακρυσμένο κόμβο (έστω κόμβος Α), κάνει μία αναζήτηση στο Registry του κόμβου Α. Στη συνέχεια, χρησιμοποιεί το αποτέλεσμα της αναζήτησης για να συνδεθεί με το απομακρυσμένο αντικείμενο, και να παρεμβάλλει τις μεθόδους του.

Η πραγματική ισχύς του RMI είναι η δυνατότητα δυναμικής φόρτωσης κλάσεων στον εξυπηρετητή. Πιο συγκεκριμένα, παρέχεται η δυνατότητα, ο πελάτης να στείλει στον εξυπηρετητή τη μέθοδο που επιθυμεί να εκτελεστεί. Για να γίνει αυτό θα πρέπει να υπάρχει και στον πελάτη εν λειτουργία εξυπηρετητής Web. Αν δεν υπάρχει εξυπηρετητής Web στο σύστημα, τότε το RMI εκτελεί αυτόματα ένα απλό εξυπηρετητή Web, για την ανταλλαγή μηνυμάτων και τη μεταφορά των κλάσεων.

2.4.4. Υπηρεσίες Ιστού

Οι Υπηρεσίες Ιστού (Web Services) είναι μια τεχνολογία που επιτρέπει στις εφαρμογές να επικοινωνούν μεταξύ τους ανεξαρτήτως πλατφόρμας και γλώσσας προγραμματισμού. Ένα web service είναι μια διεπαφή λογισμικού (software interface) που περιγράφει μια συλλογή από λειτουργίες οι οποίες μπορούν να προσεγγιστούν από το δίκτυο μέσω πρότυπων μηνυμάτων XML. Χρησιμοποιεί πρότυπα βασισμένα στη γλώσσα XML για να περιγράψει μία λειτουργία (operation) προς εκτέλεση και τα δεδομένα προς ανταλλαγή με κάποια άλλη εφαρμογή. Οι υπηρεσίες Ιστού θα αναλυθούν διεξοδικά στη συνέχεια.

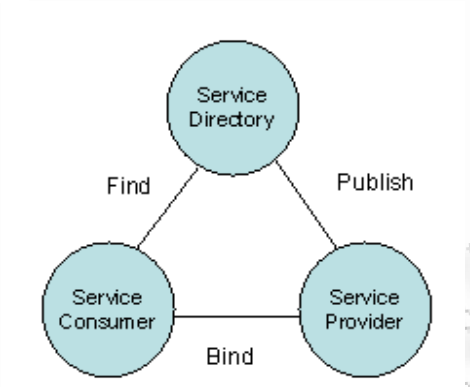
3. Υπηρεσίες Ιστού

3.1. Εισαγωγή στις Υπηρεσίες Ιστού

Μια υπηρεσία Ιστού είναι ένα σύστημα λογισμικού με σκοπό την υποστήριξη τη διαλειτουργικής αλληλεπίδραση υπολογιστή με υπολογιστή πάνω από το δίκτυο. Είναι διαθέσιμη μέσω του Διαδικτύου, χρησιμοποιεί ένα τυποποιημένο σύστημα ανταλλαγής μηνυμάτων βασισμένο στην XML, και δεν είναι δεμένο σε οποιοδήποτε λειτουργικό σύστημα ή γλώσσα προγραμματισμού [1].

Αν και δεν απαιτούνται, μια υπηρεσία Ιστού πρέπει επίσης να έχει δύο άλλες σημαντικές ιδιότητες:

- Μια υπηρεσία Ιστού πρέπει να μπορεί να περιγράφει τον εαυτό της. Αυτό είναι απαραίτητο προκειμένου να μπορεί να βρεθεί η πληροφορία (το πρωτόκολλο) που απαιτείται για την επιτυχή επικοινωνία με την υπηρεσία.
- Για κάθε υπηρεσία Ιστού, κάποια πρέπει επίσης να μπορεί κανείς να δημιουργήσει μια δημόσια διεπαφή στην υπηρεσία.
- Επιπλέον, μια υπηρεσία Ιστού πρέπει επίσης να μπορεί να βρεθεί. Για να είναι αυτό δυνατό, πρέπει να υπάρξει κάποιος απλός μηχανισμός που παρέχεται στους ενδιαφερόμενους χρήστες, επιτρέποντας τους να ψάξουν για την υπηρεσία και να εντοπίσουν τη δημόσια διεπαφή της. Ο ακριβής μηχανισμός θα μπορούσε να είναι μέσω ενός εντελώς αποκεντρωμένου συστήματος ή ενός κεντρικού συστήματος καταλόγου όπως το UDDI. Η σχέση μεταξύ των προαναφερθεισών εννοιών των υπηρεσιών Ιστού απεικονίζεται στο επόμενο σχήμα.



Σχήμα 1 : Υπηρεσίες Ιστού Σχέση "Publish-Find-Bind"

ΠΗΓΗ: Aldrich.M The Inventor's Story Aldrich Archive, University of Brighton http://www.aldricharchive.com/inventors_story.html

3.2. Κύριες τεχνολογίες των Υπηρεσιών Ιστού

Η διείσδυση υπηρεσιών Ιστού στο Διαδίκτυο έχει εξελιχθεί τόσο γρήγορα τα τελευταία έτη που διάφορες ανταγωνιστικές τεχνολογίες προσπαθούν να παρέχουν την ίδια λειτουργικότητα. Εντούτοις, προκειμένου να επιτευχθεί η παγκόσμια επιχειρησιακή ολοκλήρωση με το χρήστη των υπηρεσιών Ιστού, οι βασικές τεχνολογίες πρέπει να υποστηριχθούν από κάθε σημαντική εταιρεία λογισμικού στον κόσμο.

Η επικοινωνία ανάμεσα σε πελάτη και εξυπηρετητή στον κόσμο των Υπηρεσιών Ιστού βασίζεται στην XML. Η XML έχει εξελιχθεί και γνωρίσει τεράστια διάδοση τα τελευταία χρόνια. Έχει κερδίσει μεγάλη αποδοχή επειδή επιτρέπει σε διαφορετικά συστήματα ηλεκτρονικών υπολογιστών να μοιραστούν δεδομένα ευκολότερα, ανεξάρτητα από το λειτουργικό σύστημα ή τη γλώσσα προγραμματισμού που χρησιμοποιούν. Υπάρχουν δεκάδες εργαλεία XML, συμπεριλαμβανομένων parser και συντακτών που είναι διαθέσιμοι για σχεδόν κάθε λειτουργικό σύστημα και κάθε γλώσσα προγραμματισμούQ: Java, Perl, Python, C#, C, C++, και λοιπά. Συνεπώς για την ανάπτυξη των υπηρεσιών Ιστού, η XML ήταν μια φυσική επιλογή. Υπάρχουν δύο κύριοι υποψήφια πρωτόκολλα για τη σύνταξη ενός μηνύματος XML: το XML-RPC και το SOAP.

3.2.1. XML-RPC

Το XML-RPC είναι ένα απλό πρωτόκολλο που χρησιμοποιεί μηνύματα XML για να εκτελέσει RPCs. Ως RPC (Remote Procedure Call) καλείται η τεχνολογία η οποία επιτρέπει σε ένα υπολογιστικό σύστημα να καλέσει μια ρουτίνα σε ένα απομακρυσμένο σύστημα, χωρίς ο κώδικας της κλήσης να εξαρτάται από τη φύση του απομακρυσμένου συστήματος. Το XML-RPC χρησιμοποιεί το πρωτόκολλο HTTP, το οποίο χρησιμοποιείται παραδοσιακά στο παγκόσμιο ιστό. Τα αιτήματα κωδικοποιούνται σε XML και στέλνονται μέσω HTTP POST. Οι απαντήσεις XML ενσωματώνονται στο σώμα της απάντησης HTTP. Επειδή το XML-RPC είναι ανεξάρτητο πλατφόρμας, επιτρέπει σε διαφορετικές εφαρμογές να επικοινωνήσουν μεταξύ τους. Παραδείγματος χάριν, ένας πελάτης της Java μπορεί να μιλήσει XML-RPC σε έναν εξυπηρετητή Perl.

Ας δούμε το παρακάτω παράδειγμα που αφορά μια απλή υπηρεσία ενημέρωσης για τον καιρό. Η υπηρεσία αναμένει έναν ταχυδρομικό κώδικα και επιστρέφει την τρέχουσα θερμοκρασία για τη συγκεκριμένη περιοχή. Στη συνέχεια παρατίθεται ένα δείγμα αιτήματος σε XML-RPC στην υπηρεσία (επικεφαλίδες του HTTP παραλείπονται) [10]:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
<methodName>weather.getWeather</methodName>
<params>
<param><value>10016</value></param>
</params>
</methodCall>
```

ΠΗΓΗ:Ethan Cerami, Web Services Essentials, O'Reilly, First Edition

February 2002

Το αίτημα αποτελείται από ένα από αντικείμενο methodCall που ορίζει το όνομα της υπηρεσίας και τις παραμέτρους της.

Ακολουθεί το παράδειγμα απάντησης σε XML-RPC [10].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodResponse>
<params>
<param>
<value><int>65</int></value>
</param>
</params>
</methodResponse>
```

Η απάντηση αποτελείται από ένα από αντικείμενο methodResponce που ορίζει τη τιμή που επιστρέφεται. Στην περίπτωση αυτή η τιμή είναι ένας ακέραιος.

Το XML-RPC είναι ο ευκολότερος τρόπος να χρησιμοποιήσει κανείς Υπηρεσίες Ιστού. Από πολλές απόψεις, είναι απλούστερο από το SOAP το οποίο θα παρουσιαστεί στη συνέχεια, και ευκολότερο να υιοθετήσει. Εντούτοις, αντίθετα από το SOAP, το XML-RPC δεν έχει καμία αντίστοιχη γραμματική περιγραφής υπηρεσιών. Αυτό αποτρέπει την αυτόματη επίκληση υπηρεσιών XML-RPC, η οποία είναι ένα βασικό συστατικό για τη διευκόλυνση ανάπτυξης εφαρμογών που βασίζονται στις Υπηρεσίες Ιστού.

3.2.2. Βασικές τεχνολογίες Web Services

Τρεις αρχικές τεχνολογίες έχουν προκύψει ως παγκόσμια πρότυπα που αποτελούν τον πυρήνα των σύγχρονων τεχνολογιών Υπηρεσιών Ιστού. Αυτές οι τεχνολογίες είναι οι SOAP, WSDL, και UDDI [5].

Το SOAP είναι ένα ελαφρύ πρωτόκολλο προοριζόμενο για την ανταλλαγή δομημένων πληροφοριών σε ένα αποκεντρωμένο, κατανεμημένο περιβάλλον [3]. Παρέχει μια τυποποιημένη δομή για τη μεταφορά εγγράφων XML πάνω από ποικίλες τυποποιημένες τεχνολογίες Διαδικτύου, συμπεριλαμβανομένου του SMTP, του HTTP, και του FTP. Παρέχοντας ένα τυποποιημένο μηχανισμό μεταφορών, οι ετερογενείς πελάτες και εξυπηρετητές, που αναπτύσσονται σε διαφορετικές πλατφόρμες (.NET, Java, PHP κ.λπ.) μπορούν ξαφνικά να γίνουν διαλειτουργικοί.

Στο παρακάτω απόσπασμα φαίνεται ένα απλό SOAP request, με τις επικεφαλίδες του HTTP να έχουν παραληφθεί [10]:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <SOAP-ENV:Body>
        <ns1:getWeather
           xmlns:ns1="urn:examples:weatherservice"
           SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/soap-
             encoding/">
           <zipcode xsi:type="xsd:string">10016</zipcode>
         </ns1:getWeather>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

ΠΗΓΗ:Ethan Cerami, Web Services Essentials, O'Reilly, First Edition

February 2002

Όπως μπορεί να δει κανείς η αίτηση SOAP είναι κάπως πιο πολύπλοκη από το την αίτηση XML-RPC. Κάνει χρήση τόσο XML namespaces όσο και XML σχημάτων. Όπως σε μια κλήση XML-RPC, το σώμα του αιτήματος SOAP

περιγράφει τόσο το όνομα της μεθόδου, όσο και τη λίστα των παραμέτρων που δέχεται.

Ακολουθεί το παράδειγμα απάντηση στην υπηρεσία καιρού [10]:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
     <ns1:getWeatherResponse
       xmlns:ns1="urn:examples:weatherservice"
       SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/soap-
encoding/">
       <return xsi:type="xsd:int">65</return>
     </ns1:getWeatherResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

ΠΗΓΗ:Ethan Cerami, Web Services Essentials, O'Reilly, First Edition

February 2002

Το WSDL είναι μια τεχνολογία XML που περιγράφει τη διεπαφή μιας υπηρεσίας Ιστού με έναν τυποποιημένο τρόπο. Μέσω του WSDL, η σύνδεση πρωτοκόλλου της υπηρεσίας, οι παράμετροι εισαγωγής και παραγωγής της υπηρεσίας περιγράφονται, και ο τύπος της κλήσης (π.χ. μόνο είσοδος, είσοδος/έξοδος, κ.λπ.) είναι δημόσια και διαθέσιμα. Το WSDL επιτρέπει την αυτόματη δημιουργία ενός πελάτη υπηρεσιών Ιστού.

Ακολουθεί ένα παράδειγμα κειμένου WSDL.Το κείμενο αυτό περιγράφει τη δημόσια διεπαφή για την υπηρεσία καιρού που παρουσιάστηκε προηγουμένως.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="WeatherService"
  targetNamespace="http://www.ecerami.com/wsdl/WeatherService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ecerami.com/wsdl/WeatherService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <message name="getWeatherRequest">
    <part name="zipcode" type="xsd:string"/>
  </message>
  <message name="getWeatherResponse">
    <part name="temperature" type="xsd:int"/>
  </message>

  <portType name="Weather_PortType">
    <operation name="getWeather">
      <input message="tns:getWeatherRequest"/>
      <output message="tns:getWeatherResponse"/>
     </operation>
  </portType>

  <binding name="Weather_Binding" type="tns:Weather_PortType">
    <soap:binding style="rpc"
       transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getWeather">
      <soap:operation soapAction=""/>
      <input>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:weatherservice"
          use="encoded"/>
       </input>
       <output>
         <soap:body
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
           namespace="urn:examples:weatherservice"
           use="encoded"/>
       </output>
     </operation>
  </binding>

  <service name="Weather_Service">
    <documentation>WSDL File for Weather Service</documentation>
    <port binding="tns:Weather_Binding" name="Weather_Port">
    <soap:address
       location="http://localhost:8080/soap/servlet/rpcrouter"/>
   </port>
  </service>
</definitions>
```

ΠΗΓΗ:Ethan Cerami, Web Services Essentials, O'Reilly, First Edition

February 2002

Με τη χρήση του WSDL, ένας πελάτης μπορεί να εντοπίσει μια Υπηρεσία Ιστού και να καλέσει οποιοδήποτε από τις δημόσιες μεθόδους της. Επίσης, με εργαλεία που έχουν προγραμματιστεί κατάλληλα ώστε να γνωρίζουν το WSDL, η παραπάνω διαδικασία μπορεί να είναι πλήρως αυτοματοποιημένη, δίνοντας τη δυνατότητα σε προγραμματιστές να προσθέτουν νέα λειτουργικότητα σε εφαρμογές με συγγραφή ελάχιστου ή και καθόλου κώδικα.

Το UDDI παρέχει ένα παγκόσμιο κατάλογο για λόγους διαφήμισης, ανακάλυψης και ολοκλήρωσης υπηρεσιών Ιστού. Οι επιχειρησιακοί αναλυτές και οι τεχνικοί χρησιμοποιούν UDDI για να ανακαλύψουν διαθέσιμες υπηρεσίες Ιστού αναζητώντας το όνομα, κάποιο προσδιοριστικό, κατηγορίες, ή τις προδιαγραφές που εφαρμόζονται από μια υπηρεσία Ιστού. Το UDDI παρέχει μια δομή για την αναπαράσταση επιχειρήσεων, επιχειρησιακών σχέσεων, Υπηρεσιών Ιστού, των μεταδεδομένων των προδιαγραφών, και των σημείων πρόσβασης σε Υπηρεσίες Ιστού.

Καθεμιά από τις τεχνολογίες που αναφέρθηκαν παραπάνω παρέχει πρότυπα για το επόμενο βήμα στην πρόοδο των υπηρεσιών Ιστού, της περιγραφής τους, ή της ανακάλυψής τους. Εντούτοις, η δυναμική ολοκλήρωση των συστημάτων που ανακαλύπτουν το ένα το άλλο αυτόματα και επικοινωνούν αδιαφορώντας για την πλατφόρμα στην οποία στηρίζονται, απαιτεί τη συνδυασμένη χρήση του SOAP, WSDL, και UDDI έτσι ώστε να παράσχει μια δυναμική, τυποποιημένη υποδομή.

Ο συνδυασμός αυτών των τεχνολογιών παρέχει έναν επαναστατικό τρόπο ολοκλήρωσης επιχειρήσεων. Αν και έχουν υπάρξει τεχνολογίες

ισοδύναμες με το SOAP, WSDL, και UDDI στο παρελθόν, όπως η CORBA και h Java RMI, δεν υποστηρίχθηκαν από κάθε σημαντική εταιρία και δεν διέθεταν την ευελιξία της XML.

Στη συνέχεια θα γίνει αναλυτική περιγραφή των τεχνολογιών SOAP, WSDL, και UDDI.

3.3. Simple Object Access Protocol (SOAP)

Το SOAP είναι ένα ελαφρύ πρωτόκολλο προοριζόμενο για την ανταλλαγή δομημένων πληροφοριών σε ένα αποκεντρωμένο, κατανεμημένο περιβάλλον. Η προδιαγραφή SOAP ορίζει τρία βασικά μέρη:

Προδιαγραφή φακέλου SOAP

Το SOAP XML Envelope καθορίζει συγκεκριμένους κανόνες εγκλεισμού δεδομένων που μεταφέρονται μεταξύ υπολογιστών. Αυτό περιλαμβάνει δεδομένα που αφορούν την εκάστοτε εφαρμογή, όπως είναι το όνομα μεθόδου που καλείται, οι παράμετροι της μεθόδου, ή οι τιμές που επιστρέφονται. Μπορεί επίσης να περιλαμβάνει πληροφορίες σχετικά με το ποιος θα πρέπει να επεξεργαστεί τα περιεχόμενα φακέλου και, σε περίπτωση λάθους, πώς κωδικοποιούνται τα μηνύματα λάθους.

Για την ανταλλαγή δεδομένων, οι υπολογιστές πρέπει να συμφωνήσουν σχετικά με τους κανόνες για την κωδικοποίηση συγκεκριμένων τύπους δεδομένων όπως αλφαριθμητικά, ακέραιους αριθμούς, πίνακες και λοιπά. Το SOAP περιλαμβάνει λοιπόν το δικό του σύνολο συμβάσεων για την κωδικοποίηση των δεδομένων σε τύπους. Οι περισσότερες από αυτές τις συμβάσεις βασίζονται στο W3C XML Schema.

Συμβάσεις RPC

Το SOAP μπορεί να χρησιμοποιηθεί σε διάφορα συστήματα ανταλλαγής μηνυμάτων, συμπεριλαμβανομένης μονόδρομης και αμφίδρομης επικοινωνίας. Για την αμφίδρομη αποστολή μηνυμάτων, το SOAP καθορίζει μια απλή σύμβαση για την αναπαράσταση απομακρυσμένων κλήσεων και απαντήσεων. Αυτό επιτρέπει σε μια εφαρμογή-πελάτη να ορίσει το όνομα μιας απομακρυσμένης μεθόδου, να περιλάβει οποιοδήποτε αριθμό παραμέτρων και να λάβει απόκριση από τον εξυπηρετητή.

3.3.1. Η αίτηση SOAP

Η αίτηση από το πελάτη πρέπει να περιλαμβάνει το όνομα της μεθόδου που θα κληθεί και τις παραμέτρους που απαιτούνται. Στη συνέχεια παραθέτουμε ένα δείγμα αίτησης πελάτη.

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:getTemp
xmlns:ns1="urn:xmethods-Temperature"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<zipcode xsi:type="xsd:string">10016</zipcode>
</ns1:getTemp>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Στο παραπάνω δείγμα πρέπει να σημειωθούν καποια σημαντικά στοιχεία. Πρώτον, η αίτηση περιλαμβάνει ένα μοναδικό υποχρεωτικό στοιχείο (ετικέτα – element) με όνομα Envelope, το οποία με τη σειρά του περιλαμβάνει ένα υποχρεωτικό στοιχείο Body.

Δεύτερον, συνολικά ορίζονται τέσσερα XML namespaces. Namespaces χρησιμοποιούνται για να ξεχωρίζουν ετικέτες XML και χαρακτηριστικά, και συχνά χρησιμοποιούνται για την αναφορά εξωτερικών σχημάτων. Στο δείγμα αίτησης SOAP που παρατίθεται παραπάνω, χρησιμοποιούνται namespaces για να διαχωρίσουν αναγνωριστικάπου συνδέονται με το SOAP Envelope (http://schemas.xmlsoap.org/soap/envelope/), τη κωδικοποίηση δεδομένων μέσω του σχήματος XML (http://www.w3.org/2001/XMLSchema-instance και http://www.w3.org/2001/XMLSchema), και αναγνωριστικά που σχετίζονται με την εφαρμογή. Αυτό παρέχει τη μέγιστη δυνατή ευελιξία για μελλοντικές αλλαγές σε προδιαγραφές XML.

Το στοιχείο Body περιέχει το κύριο «ωφέλιμο φορτίο» του μηνύματος SOAP. Το μόνο στοιχείο είναι το getTemp, το οποίο είναι συνδεδεμένο με το χώρο ονομάτων XMethods και αντιστοιχεί στο απομακρυσμένο όνομα μεθόδου. Κάθε παράμετρος στη μέθοδο εμφανίζεται ως υπο-στοιχείο. Στην περίπτωσή μας, έχουμε ένα μόνο στοιχείο, το ταχυδρομικό κώδικα, το οποίο αποδίδεται στο XML Schema xsd: string και έχει οριστεί στη τιμή 10016. Αν απαιτούνται πρόσθετοι παράμετροι, καθεμία μπορεί να έχει τα δικά της δεδομένα και τύπο.

3.3.2. Η απάντηση SOAP

Ακριβώς όπως και η αίτηση, η απάντηση περιλαμβάνει Envelope και το Body, και τα ίδια XML namespaces. Αυτή τη φορά, ωστόσο, το στοιχείο Body περιλαμβάνει ένα μοναδικό getTempResponse στοιχείο, που αντιστοιχεί στην πρώτη αίτησή. Το στοιχείο της απόκρισης περιλαμβάνει ένα μόνο στοιχείο επιστροφή, με xsd: float τύπο δεδομένων.

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<SOAP-ENV:Body>
<ns1:getTempResponse
xmlns:ns1="urn:xmethods-Temperature"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:float">71.0</return>
</ns1:getTempResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

3.3.3. Στοιχεία του μηνύματος SOAP

Κάθε μήνυμα SOAP έχει ως αρχικό στοιχείο το Envelope. Το Envelope μπορεί να περιέχει προαιρετικά το στοιχείο header και υποχρεωτικά το body. Στο header ορίζονται συνήθως επιπρόσθετα στοιχεία σχετικά με την εφαρμογή, για παράδειγμα ψηφιακές υπογραφές, ή λοιπά στοιχεία που αφορούν κάποιο λογαριασμό. Ακολουθεί ένα παράδειγμα:

ΠΗΓΗ: M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. Nielsen, "W3C Recommendation SOAP Version 1.2 Part 2: Adjuncts", June 2003

```
<SOAP-ENV:Header>
<ns1:PaymentAccount          xmlns:ns1="urn:ecerami"          SOAP-ENV:
mustUnderstand="true">
orsenigo473
</ns1:PaymentAccount >
</SOAP-ENV:Header>
```

Ένα ακόμα στοιχείο του SOAP, που εμφανίζεται μέσα στο Body είναι το στοιχείο Fault. Χρησιμοποιείται για την αποτύπωση λαθών που προκύπτουν κατά την εκτέλεση κάποιας μεθόδου. Στη συνέχεια παρατίθεται

ένα λάθος που προέκυψε κατά την αίτηση SOAP και ο εξυπηρετητής επιστρέφει την απάντηση SOAP:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>
      <faultstring xsi:type="xsd:string">
           Failed to locate method (ValidateCreditCard) in class
           (examplesCreditCard) at /usr/local/ActivePerl-5.6/lib/
             site_perl/5.6.0/SOAP/Lite.pm line 1555.
     </faultstring>
   </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Το στοιχεί Fault μπορεί να περιέχει ειδικά υπο-στοιχεία τα οποία παρατίθεται στο παρακάτω πίνακα:

Όνομα στοιχείου	Περιγραφή
faultCode	Κωδικός που περιγράφει το είδος του λάθους.
faultString	Επεξήγηση του λάθους που διαβάζεται από άνθρωπο
faultString	Ενα κείμενο που δείχνει ποιος προκάλεσε το λάθος. Αυτό είναι χρήσιμο στη περίπτωση που το μήνυμα SOAP περνά από διάφορους κόμβους και ο πελάτης πρέπει να ξέρει ποιος το προκάλεσε
detail	Ένα στοιχείο που χρησιμοποιείται για την αποθήκευση μηνυμάτων λάθους που σχετίζονται με την εφαρμογή.

3.3.4. Τύποι δεδομένων του SOAP

Το SOAP ιοθετεί όλους τους τύπους που ορίζονται από τη προδιαγραφή του XML Schema όπως για παράδειγμα, string, float, double, integer. Ακολουθεί ένα παράδειγμα απάντησης SOAP που περιέχει ένα τύπο double.

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:getPriceResponse
xmlns:ns1="urn:examples:priceservice"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/soap-encoding">
<return xsi:type="xsd:double">54.99</return>
</ns1:getPriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Εκτός από απλούς τύπους, το SOAP υποστηρίζει και πολύπλοκους τύπους(compound) που μπορεί να περιέχουν arrays ή αντικείμενα. Στη συνέχεια παρατίθεται ο ορισμός ενός αντικειμένου Forecast που περιέχει μια ακολουθία από πεδία.

```
<schema                          xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://weather">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="Forecast">
    <sequence>
     <element name="zip" nillable="true" type="xsd:string"/>
     <element name="city" nillable="true" type="xsd:string"/>
     <element name="state" nillable="true" type="xsd:string"/>
     <element name="date" nillable="true" type="xsd:string"/>
     <element name="forecast" nillable="true" type="xsd:string"/>
     <element name="hi" type="xsd:byte"/>
```

```
      <element name="low" type="xsd:byte"/>
      <element name="precip" type="xsd:byte"/>
     </sequence>
    </complexType>
   </schema>
```

3.4. Web Service Description Language (WSDL)

Το WSDL είναι μια τεχνολογία XML που περιγράφει τη διεπαφή μιας υπηρεσίας Ιστού με έναν τυποποιημένο τρόπο. Στο WSDL ορίζονται τέσσερα σημαντικά σημεία για τη λειτουργία ενός Web Service:

- Interface: περιγραφή όλων των διαθέσιμων μεθόδων
- Data type: περιγραφή των τύπων δεδομένων και των αιτήσεων και απαντήσεων
- Binding: περιγραφή για το πρωτόκολλο επικοινωνίας στο οποίο μεταφέρεται το μήνημα SOAP
- Address: πληροφορίες για την εύρεση της Υπηρεσίας Ιστού.

3.4.1. Τύποι δεδομένων του SOAP

Το WSDL είναι μια γραμματική XML για τη περιγραφή υπηρεσιών Ιστού. Η προδιαγραφή αυτή χωρίζεται σε έξι κύρια στοιχεία:

Definitions

Το στοιχείο ορισμοί (definitions) πρέπει να είναι το στοιχείο "ρίζα" όλων των εγγράφων WSDL. Αυτό καθορίζει το όνομα της υπηρεσίας Ιστού, δηλώνει πολλαπλά namespaces που χρησιμοποιούνται σε όλο το υπόλοιπο

του εγγράφου, και περιέχει όλα τα στοιχεία της υπηρεσίας που περιγράφεται εδώ.

Types

Το στοιχείο τύποι (types) περιγράφει όλα τα είδη δεδομένων που χρησιμοποούνται μεταξύ του πελάτη και εξυπηρετητή. Το WSDL δεν συνδέεται αποκλειστικά με ένα συγκεκριμένο σύστημα τύπων, αλλά χρησιμοποιεί το W3C XML Schema, ως προεπιλογή. Αν η υπηρεσία χρησιμοποιεί μόνο απλούς τύπους του XML Schema, όπως string και integer, το στοιχείο types δεν απαιτείται.

Message

Το στοιχείο μήνυμα περιγράφει ένα μονόδρομο μήνυμα, είτε πρόκειται για μια αίτηση ή μια απάντηση SOAP. Ορίζει το όνομα του μηνύματος και περιέχει μηδέν ή περισσότερα υπο-στοιχεία με όνομα «part», τα οποία μπορεί να αναφέρονται είτε σε παραμέτρους ή τις τιμές που επιστρέφονται.

portType

Το portType στοιχείο συνδυάζει πολλά στοιχεία "message" που περιγράφουν μια ολοκληρωμένη λειτουργία (operation) (μονόδρομη ή αμφίδρομη). . Για παράδειγμα, ένα portType μπορεί να συνδυάσει ένα μήνυμα αίτηση και ένα μήνυμα απάντηση σε μια ενιαία λειτουργία. Ας σημειωθεί ότι ένα portType μπορεί (και συχνά ισχύει αυτό) να ορίσει πολλαπλές λειτουργίες.

binding

Το binding περιγράφει τις συγκεκριμένες λεπτομέρειες για το πώς η υπηρεσία θα εφαρμοστεί στο δίκτυο. Το WSDL περιλαμβάνει ενσωματωμένες επεκτάσεις για τον καθορισμό των υπηρεσιών SOAP.

Service

Το στοιχείο service (υπηρεσία) καθορίζει τη διεύθυνση για την κλήση της συγκεκριμένης υπηρεσίας. Συνήθως, αυτό περιλαμβάνει ένα URL στο οποίο απαντά ο εξυπηρετητής που προσφέρει την υπηρεσία.

Ακολουθεί ένα βασικό παράδειγμα WSDL κειμένου. Στο κείμενο αυτό ορίζεται μια διαθέσιμη μέθοδος η sayHello η οποία έχει ως παράμετρο ένα string και επιστρέφει επίσης ένα string.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<message name="SayHelloRequest">
       <part name="firstName" type="xsd:string"/>
</message>

<message name="SayHelloResponse">
       <part name="greeting" type="xsd:string"/>
</message>

<portType name="Hello_PortType">
       <operation name="sayHello">
       <input message="tns:SayHelloRequest"/>
       <output message="tns:SayHelloResponse"/>
       </operation>
</portType>


<binding name="Hello_Binding" type="tns:Hello_PortType">
<soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="sayHello">
    <soap:operation soapAction="sayHello"/>
     <input>
    <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:examples:helloservice"
```

```
use="encoded"/>
</input>

 <output>
  <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:examples:helloservice"
use="encoded"/>
  </output>
</operation>

</binding>


<service name="Hello_Service">
<documentation>WSDL File for HelloService</documentation>
<port binding="tns:Hello_Binding" name="Hello_Port">
<soap:address
location="http://localhost:8080/soap/servlet/rpcrouter"/>
</port>
</service>
</definitions>
```

3.5. Universal Description, Discovery, Integration (UDDI)

Το UDDI παρέχει ένα παγκόσμιο κατάλογο για λόγους διαφήμισης, ανακάλυψης και ολοκλήρωσης υπηρεσιών Ιστού. Οι επιχειρησιακοί αναλυτές και οι τεχνικοί χρησιμοποιούν UDDI για να ανακαλύψουν διαθέσιμες υπηρεσίες Ιστού. Ένα από τα σημαντικά πλεονεκτήματα των Υπηρεσιών Ιστού έχουν νόημα μόνο όταν δυνητικοί χρήστες μπορούν να βρουν πληροφορίες για αυτά έυκολα ώστε να επιτρέψουν την εκτέλεσή τους. Το Universal Description Discovery & Integration (UDDI) εστιάζει στον καθορισμό ενός συνόλου από υπηρεσίες που θα υποστηρίζουν την περιγραφή και την ανακάλυψη:

1. Των εταιριών, των οργανισμών και άλλων παρόχων web services
2. Των Υπηρεσιών Ιστού που είναι διαθέσιμες
3. Των διεπαφών (Interfaces) οι οποίες μπορούν να χρησιμοποιηθούν ώστε να έχει κάποιος χρήστης πρόσβαση σε αυτές τις υπηρεσίες.

Βασισμένο σε ένα κοινό σύνολο από βιομηχανικά πρότυπα, συμπεριλαμβανομένων των HTTP, XML, XML Schema και SOAP το UDDI παρέχει μία διαλειτουργική, θεμελιώδη υποδομή για ένα περιβάλλον λογισμικού προσανατολισμένο στις υπηρεσίες τόσο για δημόσια διαθέσιμες υπηρεσιές όσο και για υπηρεσίες που εκτίθενται μόνο εσωτερικά ενός οργανισμού."

Με πιο απλά λόγια το UDDI είναι ο «χρυσός οδηγός» των web services. Όπως σε ένα χρυσό οδηγό, μπορούμε να αναζητήσουμε μια εταιρία που προσφέρει τις υπηρεσίες που χρειαζόμαστε, να διαβάσουμε για μια προσφερόμενη υπηρεσία και να επικοινωνήσουμε με κάποιον για περισσότερες λεπτομέρειες. Φυσικά μπορούμε να προσφέρουμε ένα web service χωρίς να το καταχωρίσουμε στο UDDI, όπως αν ανοίγαμε μία επιχείρηση στο υπόγειο του σπιτιού μας και βασιζόμασταν στη διαφήμιση από στόμα σε στόμα. Αλλά αν θέλουμε να αγγίξουμε το ευρύ κοινό, θα χρειαστούμε το UDDI ώστε οι δυνητικοί μας πελάτες να μπορέσουν να μας βρούν.

Το UDDI παρέχει ένα μηχανισμό στους πελάτες να βρίσκουν δυναμικά άλλα web services. Χρησιμοποιώντας μία διεπαφή UDDI, οι επιχειρήσεις μπορούν να συνδεθούν δυναμικά με υπηρεσίες που παρέχονται από εξωτερικούς συνεργάτες. Ένας κατάλογος UDDI (UDDI registry) είναι όμοιος μέ ένα CORBA trader, ή θα μπορούσαμε να τον παρομοιάσουμε με μία υπηρεσία DNS αλλά για εφαρμογές.

Ένας κατάλογος UDDI έχει δύο ειδών πελάτες : επιχειρήσεις που θέλουν να δημοσιεύσουν μια υπηρεσία (και τις διεπαφές της), και πελάτες που θέλουν να χρησιμοποιήσουν συγκεκριμένες υπηρεσίες και συνδέονται προγραμματιστικά με αυτές.

Στη συνέχεια παρουσιάζονται οι προσφερόμενες λειτουργίες του UDDI.

Πληροφορία

White pages: Πληροφορίες όπως το όνομα, η διεύθυνση, το τηλέφωνο και άλλες πληροφορίες επικοινωνίας για μία επιχείρηση.

Yellow Pages: Πληροφορίες που κατηγοριοποιούν επιχειρήσεις. Βασίζονται σε υπάρχοντα πρότυπα κατηγοριοποίησης (μη ηλεκτρονικά).

Green Pages: Τεχνικές πληροφορίες για τα web services που παρέχονται από μία επιχείρηση.

Λειτουργίες

Publish: Πώς ο προμηθευτής ενός web service καταχωρεί των εαυτό του.

Find: Πώς μία εφαρμογή βρίσκει ένα συγκεκριμένο web service.

Bind: Πώς μία εφαρμογή συνδέεται, και αλληλεπιδρά με ένα web service αφού αυτό βρεθεί.

Λεπτομέρειες (που υποστηρίζονται από το API)

Business Information: Περιλαμβάνεται σε ένα αντικείμενο ***BusinessEntity***, το οποίο με τη σειρά του περιλαμβάνει πληροφορίες για υπηρεσίες, κατηγορίες, επαφές, URLs, και άλλα αναγκαία στοιχεία για να αλληλεπιδράσουμε με μία επιχείρηση.

Service Information: Περιγράφει μία ομάδα από web services. Αυτές περιλαμβάνονται σε ένα αντικείμενο ***BusinessService***.

Binding Information: Οι απαραίτητες τεχνικές λεπτομέρειες για την κλήση μιας Υπηρεσίας Ιστού. Περιλαμβάνουν τα URLs, πληροφορίες για

ονόματα μεθόδων, τύπους ορισμάτων και ούτω καθεξής. Το αντικείμενο ***BindingTemplate*** αναπαριστά αυτά τα δεδομένα.

Στο παρακάτω παράδειγμα φαίνεται ένα ερώτημα σε SOAP και η απάντηση από ένα κατάλογο UDDI

```
<find_business generic="1.0" xmlns="urn:uddi-org:api">
<name>Microsoft</name>
</find_business>
```

```
<serviceList generic="1.0" operator="Microsoft Corporation"
truncated="false" xmlns="urn:uddi-org:api">
<serviceInfos>
<serviceInfo
serviceKey="618167a0-3e64-11d5-98bf-002035229c64"
businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64">
<name>XMethods Currency Exchange Rates</name>
</serviceInfo>
<serviceInfo
serviceKey="d5921160-3e16-11d5-98bf-002035229c64"
businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64">
<name>XMethods Delayed Stock Quotes</name>
</serviceInfo>
<serviceInfo
serviceKey="ed85f000-4345-11d5-bd6c-002035229c64"
businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64">
<name>XMethods Pacific Bell SMS Service</name>
</serviceInfo>
<serviceInfo
serviceKey="d5b180a0-4342-11d5-bd6c-002035229c64"
businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64">
<name>XMethods Barnes and Noble Quote</name>
</serviceInfo>
</serviceInfos>
</serviceList>
```

Στο παρακάτω παράδειγμα φαίνεται ένα ακόμα ερώτημα σε SOAP και η απάντηση από ένα κατάλογο UDDI

```
<get_bindingDetail generic="1.0" xmlns="urn:uddi-org:api">
<bindingKey>d594a970-3e16-11d5-98bf-002035229c64</bindingKey>
</get_bindingDetail>
```

```
<bindingDetail generic="1.0" operator="Microsoft Corporation"
truncated="false" xmlns="urn:uddi-org:api">
<bindingTemplate
serviceKey="d5921160-3e16-11d5-98bf-002035229c64"
bindingKey="d594a970-3e16-11d5-98bf-002035229c64">
<description xml:lang="en">
SOAP binding for delayed stock quotes service
</description>
<accessPoint URLType="http">
http://services.xmethods.net:80/soap
</accessPoint>
<tModelInstanceDetails>
<tModelInstanceInfo
tModelKey="uuid:0e727db0-3e14-11d5-98bf-002035229c64" />
</tModelInstanceDetails>
</bindingTemplate>
</bindingDetail>
```

3.6. Ασφάλεια στις Υπηρεσίες Ιστού

Μια νέα τάξη τεχνικών ασφάλειας έχει αναπτυχθεί για την αντιμετώπιση των θεμάτων ασφαλείας στα Web Services. Η αντιμετώπιση στα ζητήματα που έχουν προκύψει, εξακολουθούν να είναι ακόμη στα πρώιμα στάδιά τους.

Τα ζητήματα ασφαλείας που προκύπτουν με τη χρήση Υπηρεσιών Ιστού είναι τα ακόλουθα:

- Εταιρικές εφαρμογές και οι διεπαφές τους διατίθενται στο κοινό. Οι υπηρεσίες αυτές είναι διαθέσιμες μέσω της θύρας 80 (κίνηση HTTP), ανοίγοντας μια τρύπα στο τείχος προστασίας της κάθε επιχείρησης. Εφαρμογές που παρέχουν frontends για την (έστω και ελεγχόμενη) πρόσβαση σε κρίσιμα δεδομένα μέσω HTTP και που συχνά έχουν δημοσιευθεί σε δημόσιους καταλόγους (UDDI) είναι ουσιαστικά τρύπες ασφαλείας.
- Η αποστολή και λήψη δεδομένων δεν πρέπει να υλοποιηθεί με τη χρήση της ίδιας πλατφόρμες λογισμικού, δηλαδή, δεν πρέπει να έχει τις ίδιες βιβλιοθήκες ασφαλείας σε όλες τις υλοποιήσεις. Για το λόγο αυτό, απαιτείται ένα σύνολο τυποποιημένων, ανεξάρτητα από την πλατφόρμα λύσεων ασφάλειας.
- Η XML είναι πολύ φλύαρη γλώσσα καθώς μεταφέρεται ως κείμενο και είναι ιδιαίτερα περιγραφική. Για το λόγο αυτό, η κρυπτογράφηση δεδομένων σε XML κοστίζει πολύ, αυξάνει το μέγεθος των δεδομένων που αποστέλλονται και απαιτεί υπολογιστικούς πόρους.
- Το όραμα των Υπηρεσιών Ιστού περιλαμβάνει την ολοκλήρωση υπηρεσιών για την παροχή υπηρεσιών από πολλαπλές οντότητες που δεν έχουν απαραίτητα ιδιαίτερη σχέση μεταξύ τους. Κλασικό παράδειγμα αποτελεί η αγορά ενός πακέτου διακοπών, με την αυτόματη επιλογή επιμέρους υπηρεσιών (αεροπορικά εισιτήρια, ξενοδοχεία, ενοικίαση αυτοκινήτου και λοιπά) από διαφορετικές επιχειρήσεις. Αυτό το όραμα απαιτεί σύνθετες αλληλεπιδράσεις, στις οποίες ένα μήνυμα SOAP διατρέχει πολλούς μεσάζοντες. Αν δεν προϋπάρχουν συμφωνίες ανάμεσα στους μεσάζοντες δεν υπάρχει και κοινή υποδομή. Συνεπώς, το ερώτημα που προκύπτει είναι πως θα γίνει η διαχείριση των κλειδιών κρυπτογράφησης σε ένα τέτοιο περιβάλλον.

Ως επί το πλείστον, η αντιμετώπιση στα θέματα ασφαλείας γίνεται με ήδη υπάρχουσες τεχνολογίες όπως οι όπως η Υποδομές Δημοσίου Κλειδού (PKI), το πρωτόκολλο Secure HTTP (HTTPS).

Το ηλεκτρονικό εμπόριο παραδοσιακά βασίζεται στο HTTP και η ασφάλεια επιτυγχάνεται με τη χρήση του SSL, η οποία θέτει σε τελική ανάλυση την ευθύνη για την ασφάλεια στο επίπεδο του πρωτοκόλλου μεταφοράς. Αν και αυτή η προσέγγιση είναι επαρκής μέχρι στιγμής, για τις Υπηρεσίες Ιστού απαιτείται ένα νέο σύνολο δυνατοτήτων που επιτρέπουν τη χρήση των ψηφιακών πιστοποιητικών, ψηφιακές υπογραφές, και πιστοποίησης της γνησιότητας σε XML έγγραφα.

Δεν πρέπει να στηριχθεί κανείς στο υποκείμενο πρωτόκολλο μεταφορών για την ασφάλεια, καθώς το SOAP και οι υπηρεσίες Ιστού αφενός υποτίθεται ότι είναι ανεξάρτητα πρωτόκολλου, και αφετέρου μπορεί να απαιτείται κρυπτογράφηση μέρους της πληροφορίας που μεταφέρεται στο μήνυμα SOAP και όχι ολόκληρος ο φάκελος. Παράλληλα, απαιτούνται συχνά χαρακτηριστικά ασφαλείας (όπως οι ψηφιακές υπογραφές) από τις ίδιες τις υπηρεσίες Ιστού για λόγους της εκάστοτε εφαρμογής, (π.χ πιστοποίηση και εξουσιοδότηση χρηστών).

Πριν περιγράψουμε τις λύσεις ασφαλείας που προσφέρονται στις υπηρεσίες Ιστού θα παραθέσουμε μερικές βασικές έννοιες κρυπτογράφηση και ασφάλειας.

3.6.1. Χαρακτηριστικά Ασφάλειας και Κρυπτογράφησης

Η κρυπτογραφία αποτελεί τη διαδικασία μετατροπής της πληροφορίας σε μη αναγνωρίσιμη μορφή με τη χρήση κάποιου αλγορίθμου κρυπτογραφίας, με στόχο την αποστολή της μέσω επισφαλών καναλιών επικοινωνίας. Με τη μέθοδο αυτή συνεπώς το αρχικό κείμενο (plaintext) μετατρέπεται σε ένα ακατάληπτο μήνυμα (ciphertext) το οποίο και μεταδίδεται. Η αντίστροφη διαδικασία ανάκτησης του αρχικού μηνύματος ονομάζεται αποκρυπτογράφηση. Η κρυπτογραφία λοιπόν, ως εργαλείο ασφάλειας των δικτύων και της πληροφορίας έχει τους εξής τέσσερις στόχους:

- Εμπιστευτικότητα (Confidentiality), που εξασφαλίζει ότι το περιεχόμενο της είναι διαθέσιμο μόνο στο νόμιμο κάτοχο αυτής και μυστικό σε οποιονδήποτε άλλο χρήστη.

- Την Ακεραιότητα της Πληροφορίας (Integrity), που αφορά την αποφυγή της μη εξουσιοδοτημένης τροποποίησης των πληροφοριών που ανταλλάσσονται και παρέχονται μέσω ψηφιακής υπογραφής.

- Την Πιστοποίηση (Authentication), που αναφέρεται αφενός στο ότι οι δύο οντότητες που επικοινωνούν θα πρέπει να αναγνωρίζει η μία την άλλη (entity authentication) και αφετέρου στο ότι πρέπει να αναγνωρίζεται η προέλευση της πληροφορίας (data origin authentication).

- Τη Μη Αποκήρυξη Αποστολής της Πληροφορίας (Non-Repudiation), που αφορά στο να διασφαλίσει ότι ο αποστολέας του μηνύματος δε θα αρνηθεί ότι πράγματι έστειλε την πληροφορία

Για την κρυπτογράφηση / αποκρυπτογράφηση δεδομένων απαιτείται η χρήση κατάλληλου κλειδιού, εάν πρόκειται για συμμετρικό αλγόριθμο

κρυπτογραφίας, ή ζεύγους κλειδιών εάν πρόκειται για ασύμμετρο αλγόριθμο κρυπτογραφίας.

Οι αλγόριθμοι συμμετρικού κλειδιού προϋποθέτουν ότι χρησιμοποιείται το ίδιο μυστικό κλειδί για την κρυπτογράφηση και αποκρυπτογράφηση των δεδομένων. Στην κρυπτογραφία ασύμμετρου κλειδιού χρησιμοποιούνται δύο διαφορετικά κλειδιά για την κρυπτογράφηση και αποκρυπτογράφηση του μηνύματος. Το ιδιωτικό κλειδί (private key) το οποίο χρησιμοποιείται μόνο από τον ιδιοκτήτη του και δεν μεταδίδεται ποτέ και το δημόσιο κλειδί (public key) το οποίο δημοσιοποιείται και χρησιμοποιείται από οποιονδήποτε ενδιαφερόμενο.

Το πρωτόκολλο SSL, το οποίο χρησιμοποιείται για τη κρυπτογράφηση του HTTP και συνεπώς μπορεί να χρησιμοποιηθεί και για τη κρυπτογράφηση της επικοινωνίας με Υπηρεσίες Ιστού TLS λειτουργεί σε τρεις βασικές φάσεις:

- Αμοιβαία διαπραγμάτευση για υποστήριξη κρυπτογραφικών αλγορίθμων
- Ανταλλαγή κλειδιών και πιστοποίηση των δυο άκρων που συμμετέχουν στην ανταλλαγή δεδομένων
- Συμμετρική κρυπτογράφηση και πιστοποίηση μηνυμάτων

Όταν λοιπόν δεν χρησιμοποιείται το SSL, αλλά απαιτείται η επιμέρους κρυπτογράφηση κομματιών του εγγράφου XML, πρέπει να βρεθούν λύσεις για την εμπιστευτικότητα, την ακεραιότητα της πληροφορίας και την πιστοποίηση και τη μη αποκήρυξη αποστολής της πληροφορίας από τα δυο άκρα.

Το πρόβλημα της πιστοποίησης και της μη αποκήρυξης αποστολής δεδομένων, καθώς και της ακεραιότητας της πληροφορίας λύνεται με τις ψηφιακές υπογραφές. ι αλγόριθμοι κρυπτογραφίας δημοσίου κλειδιού μπορούν να χρησιμοποιηθούν για τη δημιουργία ψηφιακών υπογραφών. Η ψηφιακή υπογραφή αποτελεί μία ομάδα χαρακτήρων η οποία κρυπτογραφήθηκε με τη χρήση του ιδιωτικού κλειδιού του αποστολέα - χρήστη και σχετίζεται με το περιεχόμενο του μηνύματος, την ταυτότητα του

και προαιρετικά την ημερομηνία δημιουργίας του μηνύματος. Χρησιμοποιείται από τον παραλήπτη για την επαλήθευση της ταυτότητας του αποστολέα και την εγκυρότητα των δεδομένων του μηνύματος, με τη χρήση του δημόσιου κλειδιού του αποστολέα.

Ακολουθεί ένα παράδειγμα χρήσης της ψηφιακής υπογραφής, με τη ψηφιακή υπογραφή να βρίσκεται στο στοιχείο <Signature>.

```
<PurchaseOrder xmlns=" urn:simple-example ">
<shipTo country="US">
<name>Joe Smith</name>
<street>14 Oak Park</street>
<city>Bedford</city>
<state>MA</state>
<zip>01730</zip>
</shipTo>
<items>
<item partNum="872-AA">
<productName>Candy Canes</productName>
<quantity>444</quantity>
<price>1.68</price>
<comment>I want candy!</comment>
</item>
</items>

<Signature Id="EnvelopedSig"
xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo Id="EnvelopedSig.SigInfo">
<CanonicalizationMethod Algorithm=
"http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
<SignatureMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference Id="EnvelopedSig.Ref" URI="">
<Transforms>
<Transform Algorithm=
"http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</Transforms>

<DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>
yHIsORnxE3nAObbjMKVo1qEbToQ=
</DigestValue>
</Reference>
</SignedInfo>
```

```
<SignatureValue Id="EnvelopedSig.SigValue">
GqWAmNzBCXrogn0BlC2VJYA8CS7gu9xH/XVWFa08eY9HqVnrfU6Eh5Ig6wlcvj4RrpxnN
klBnOu
vv
JCKqllQy4e76Tduvq/N8kVd0SkYf2QZAC+j1IqUPFQe8CNA0CfUrHZdiS4TDDVv4sf0V1
c6UBj7
zT
7leCQxAdgpOg/2Cxc=
</SignatureValue>
<KeyInfo Id="EnvelopedSig.KeyInfo">
<KeyValue>
<RSAKeyValue>
<Modulus>
AIvPY8i2eRs9C5FRc61PAOtQ5fM+g3R1Yr6mJVd5zFrRRrJzB/awFLXb73kSlWqHao+3n
xuF38r
RkqiQ0HmqgsoKgWChXmLuQ5RqKJi1qxOG+WoTvdYY/KB2q9mTDj0X8+OGlkSCZPRTkGIK
jD7rw4
Vvml7nKlqWg/NhCLWCQFWZ
</Modulus>
<Exponent>AQAB</Exponent>
</RSAKeyValue>
</KeyValue>
</KeyInfo>
</Signature>

</PurchaseOrder>
```

Το πρώτο βήμα προς τη δημιουργία μιας ψηφιακής υπογραφής είναι η διασφάλιση ότι η ίδια και τα δεδομένα που υπογράφηκαν δεν μπορεί να παραποιηθούν. Αυτό το βήμα γίνεται με την εφαρμογή ενός μαθηματικού αλγόριθμου που ονομάζεται αλγόριθμος κατακερματισμού (hash algorithm) και εφαρμόζεται σε κάποιο μέρος των στοιχείων του μηνύματος. Το αποτέλεσμα ονομάζεται "digest" του μηνύματος. Το επόμενο βήμα είναι το digest μαζί με την επιπρόσθετη πληροφορία για τον αλγόριθμο που βρίσκεται στο SignedInfo να περάσει πάλι από hash, να κρυπτογραφηθεί και να προστεθεί στο έγγραφο XML ως ψηφιακή υπογραφή. Στο προηγούμενο παράδειγμα, το digest παρατίθεται στο στοιχείο Digestvalue, ο αλγόριθμος κατακερματισμού (SHA-1) ορίζεται στο DigestMethod. Το στοιχείο Reference παρέχει πληροφορία για την παραγωγή του digest. Ο αλγόριθμος

που χρησιμοποιείται για τη ψηφιακή υπογραφή ορίζεται στο SignatureMethod.

Το κλειδί αποκρυπτογράφησης αποθηκεύεται στο <KeyInfo> στοιχείο. Ο παραλήπτης μπορεί να καθοριστεί αν η υπογραφή είναι έγκυρη από την αποκωδικοποίηση του digest και να τη δημιουργία εκ νέου της όλης διαδικασία που πραγματοποιήθηκε από τον αποστολέα για να δημιουργήσει την υπογραφή. Αν το digest που προκύπτει είναι το ίδιο με αυτό στο έγγραφο, το περιεχόμενο του εγγράφου δεν έχει αλλοιωθεί.

Το επόμενο βήμα μετά τη προσθήκη της ψηφιακής υπογραφής σε ένα έγγραφο XML είναι η κρυπτογράφηση του κειμένου (ολόκληρου ή μέρους του). Υπάρχει προδιαγραφή για τη κρυπτογράφηση κάθε είδους ψηφιακού περιεχομένου, η οποία φαίνεται παρακάτω, όπου τα περιεχόμενα της ετικέτας <Items> κρυπτογραφούνται.

```
<PurchaseOrder xmlns="urn:simple-example">
<shipTo country="US">
<name>Joe Smith</name>
<street>14 Oak Park</street>
<city>Bedford</city>
<state>MA</state>
<zip>01730</zip>
</shipTo>
<items>
        <EncryptedData Id="ED" Nonce="16"
        Type=http://www.w3.org/2001/04/xmlenc#Content
        xmlns="http://www.w3.org/2001/04/xmlenc#"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        <EncryptionMethod Algorithm
        ="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
        <ds:KeyInfo>
        <ds:KeyName>jaws</ds:KeyName>
        </ds:KeyInfo>
        <CipherData>
        <CipherValue>
        dRDdYjYs11jW5EDy0lucPkWsBB3NmK0AFNxvFjfeUKxP75cx7KP0PB3BjXPg1
4kJv74i7F00XZ5WhqOISswIkdN/pIVeqRZWqOVjFA8izR6wqOb7UCpH+weoGt0UFOEkID
Gbemm23eu812Ob5eYVL8n/DtO81OhYeCXksSMGUZiUNj/tfBCAjvqG2jlslQM6n4jJ3QN
aR4+B2RisOD6Ln+x2UtNu2J7wIYmlUe7mSg
ZiJ5eHym8EpkE4vjmr2oCWwTUu91xcayZtbEpOFVFs6A==
```

```
        </CipherValue>
        </CipherData>
        </EncryptedData>
</items>

<Signature Id="EnvelopedSig"
xmlns="http://www.w3.org/2000/09/xmldsig#">
...
</Signature>
</PurchaseOrder>
```

Το στοιχείο <EncryptedData> στοιχείο καθορίζει το σύστημα κρυπτογράφησης που εφαρμόζεται. Το στοιχείο <CipherData> έχει δημιουργηθεί για να περιέχει το κρυπτογραφημένο serialization του στοιχείου <Items>. Σε αυτό το παράδειγμα, το αποτέλεσμα περιέχεται στο <CipherValue> στοιχείο, μολονότι, εναλλακτικά, μπορεί κανείς να χρησιμοποιήσει ένα URI για να υποδείξει μια άλλη τοποθεσία, όπου βρίσκεται ο αλγόριθμος κρυπτογράφησης χρησιμοποιώντας το <CipherReference> στοιχείο. Η χρήση των <EncryptionMethod> και <KeyInfo> είναι προαιρετική.

4. Μελέτη Περίπτωσης: Κατασκευή Υποσυστήματος Ηλεκτρονικού Καταστήματος

Στην ενότητα αυτή σχεδιάζεται και αναπτύσσεται το backend τμήμα ενός ηλεκτρονικού καταστήματος βασισμένο σε Web Services. Ο σχεδιασμός και ανάπτυξη του υποσυστήματος αυτού έχει στόχο να επιδείξει και να επαληθεύσει τα πλεονεκτήματα της χρήσης Υπηρεσιών Ιστού. Πιο συγκεκριμένα, αναπτύσσεται ένα λογισμικό middleware το οποίο λειτουργεί με την αρχή του πελάτη-εξυπηρετητή, όπου τα διάφορα καταστήματα (front-end) μιλούν με τη χρήση του Web Service Client στο κεντρικό κατάστημα

όπου βρίσκεται το Web Service και η βάση δεδομένων (back-end). Το Web Service αποθηκεύει και ανακτά πληροφορίες από τη βάση δεδομένων σύμφωνα με τις κλήσεις από τους Web Service Clients.

4.1. Παρουσίαση Προβλήματος

Το πρόβλημα που καλείται να λύσει το λογισμικό που θα αναπτυχθεί είναι η ασφαλής, αξιόπιστη και ταχύτατη επικοινωνία απομακρυσμένων συστημάτων μιας επιχείρησης. Επίσης, η εταιρία θέλει να αναπτύξει διεπαφές για την ανάπτυξη B2B δραστηριότητας.

Πιο συγκεκριμένα η επιχείρηση έχει τις παρακάτω λειτουργικές απαιτήσεις:

Στόχος είναι η παροχή προϊόντων στο Διαδίκτυο και η αλλαγή λειτουργίας και τρόπου συναλλαγών τόσο στις αγοραπωλησίες με τους πελάτες όσο και με άλλες εταιρίες. Πιο συγκεκριμένα, ζητείται να αλλάξει ο τρόπος λειτουργίας της επιχείρησης , με την αυτοματοποίηση συναλλαγών ανάμεσα στην εταιρία και τους πελάτες, αλλά και ανάμεσα σε ξεχωριστά τμήματα της εταιρίας και κατά επέκταση ανάμεσα στην εταιρία και άλλες εταιρίες που σχετίζονται στην αλυσίδα παραγωγής προϊόντων .

Παράλληλα, ζητείται η αυτοματοποίηση σε μεγάλο βαθμό της ροής εργασιών μέσα στην ίδια την εταιρία. Παραγγελίες μέσω του Διαδικτύου είτε με τη φυσική παρουσία σε κάποιο υποκατάστημα της εταιρίας ζητείται να ελέγχονται και να μεταφέρονται αυτόματα στο κεντρικό κατάστημα, όπου διατηρείται η βάση δεδομένων και γίνεται έλεγχος για την διαθεσιμότητα του κάθε προϊόντος. Τέλος ζητείται το άνοιγμα των αγορών και η δυνατότητα συνεργασίας με εταιρίες και πελάτες μέσω του Διαδικτύου που θα ήταν αδύνατο να γίνουν διαφορετικά λόγω γεωγραφικής απόστασης.

Εξαιτίας των παραπάνω προδιαγραφών, δημιουργείται ένα ηλεκτρονικό κατάστημα, και η δημιουργία Υπηρεσιών Ιστού για τη πρόσβαση στη βάση δεδομένων του συστήματος. Η χρήση Υπηρεσιών Ιστού προσφέρει

ευκολία στην ανάπτυξη και συντήρηση του κώδικα, καθώς και τη δυνατότητα σε άλλες εταιρίες που έρχονται σε συνεργασία να υλοποιήσουν εύκολα λογισμικό για την επικοινωνία με το κεντρικό σύστημα.

Στη συνέχεια θα παρουσιαστεί ο σχεδιασμός και η υλοποίηση της βάσης δεδομένων και των Υπηρεσιών Ιστού. Πέρα από τη βάση δεδομένων και το Web Service, θα κατασκευαστεί για λόγους δοκιμών Web Service Client ο οποίος θα είναι προσβάσιμος μέσω web browser.

4.2. Σχεδιασμός Υποσυστήματος

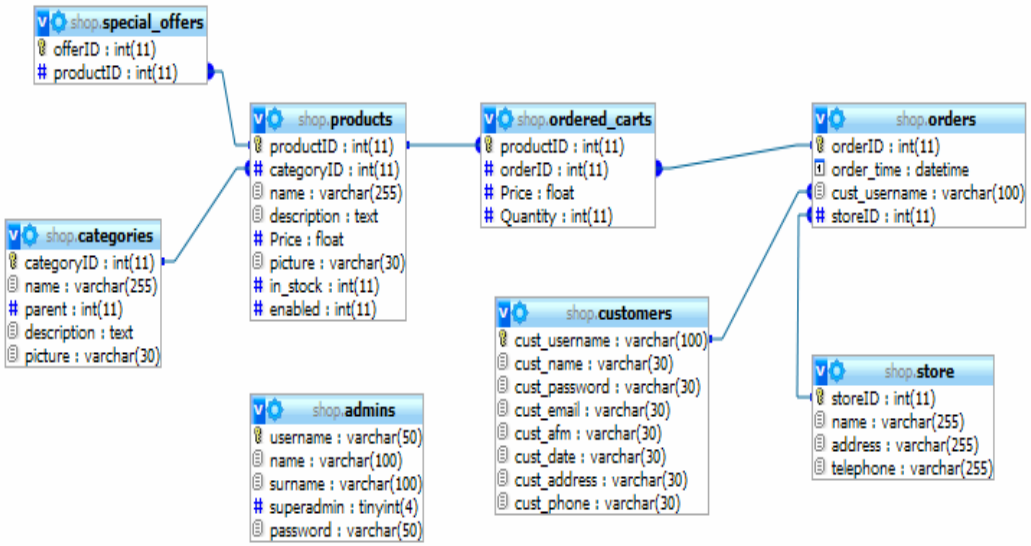
Το υποσύστημα συνοπτικά αποτελείται από την βάση δεδομένων στην οποία αποθηκεύονται πληροφορίες για τα προϊόντα και τις συναλλαγές της επιχείρησης, και από το σύστημα ανταλλαγής δεδομένων.

4.2.1. Σχεδιασμός Βάσης Δεδομένων

Στη συνέχεια παρουσιάζεται η δομή της βάσης δεδομένων που χρησιμοποιείται για την εφαρμογή. Όταν ο χρήστης όταν κάνει μια παραγγελία, τότε δημιουργείται αυτόματα ένα καινούριο tuple στον πίνακα orders. Κάθε εγγραφή στον πίνακα αυτό αντιστοιχεί σε μια παραγγελία, δηλαδή την ώρα, το κατάστημα στο οποίο έγινε (storied) και τον πελάτη (cust_username). Για κάθε ένα προϊόν με κωδικό productID (ή το σύνολο των ίδιων προϊόντων, το πλήθος των οποίων ορίζεται στο Quantity) που έχει στο καλάθι του ο πελάτης, δημιουργείται ένα tuple στο ordered_carts με το αναγνωριστικό της παραγγελίας, την ποσότητα και τη συνολική τιμή για το συγκεκριμένο προϊόν. (Μπορεί να υπάρχουν ένα ή περισσότερα, ίδια ή διαφορετικά προϊόντα στο καλάθι).

Ο πίνακας special_offers αντιστοιχεί στις προσφορές που εμφανίζονται στην αρχική σελίδα. Ο πίνακας categories έχει τις κεντρικές κατηγορίες και τις υποκατηγορίες τους, με τις κεντρικές να έχουν parent=0, ενώ τις υποκατηγορίες να έχουν ως parent το categoryID της κεντρικής κατηγορίας στην οποία ανήκει η κάθε μια. Ο πίνακας stores περιέχει πληροφορίες για τα διάφορα υποκαταστήματα.

Τέλος, οι πίνακες customers και admins αντιστοιχούν στους εγγεγραμμένους πελάτες και τους διαχειριστής της εφαρμογής αντίστοιχα.



Σχήμα 2 Η βάση δεδομένων του συστήματος

Ακολουθεί ο κώδικας σε SQL που απαιτείται για τη δημιουργία της βάσης δεδομένων.

```
-- Βάση: `shop`
--

-- --------------------------------------------------------

--
-- Δομή Πίνακα για τον Πίνακα `admins`
--
```

```
DROP TABLE IF EXISTS `admins`;
CREATE TABLE IF NOT EXISTS `admins` (
  `username` varchar(50) NOT NULL,
  `name` varchar(100) NOT NULL,
  `surname` varchar(100) NOT NULL,
  `superadmin` tinyint(4) NOT NULL DEFAULT '0',
  `password` varchar(50) NOT NULL,
  PRIMARY KEY (`username`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Δομή Πίνακα για τον Πίνακα `categories`
--

DROP TABLE IF EXISTS `categories`;
CREATE TABLE IF NOT EXISTS `categories` (
  `categoryID` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `parent` int(11) DEFAULT NULL,
  `description` text,
  `picture` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`categoryID`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=81 ;

-- --------------------------------------------------------

--
-- Δομή Πίνακα για τον Πίνακα `customers`
--

DROP TABLE IF EXISTS `customers`;
CREATE TABLE IF NOT EXISTS `customers` (
  `cust_username` varchar(100) NOT NULL,
  `cust_name` varchar(30) DEFAULT NULL,
  `cust_password` varchar(30) DEFAULT NULL,
  `cust_email` varchar(30) DEFAULT NULL,
  `cust_afm` varchar(30) DEFAULT NULL,
  `cust_date` varchar(30) DEFAULT NULL,
  `cust_address` varchar(30) DEFAULT NULL,
  `cust_phone` varchar(30) DEFAULT NULL,
  PRIMARY KEY (`cust_username`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Δομή Πίνακα για τον Πίνακα `ordered_carts`
--
```

```
DROP TABLE IF EXISTS `ordered_carts`;
CREATE TABLE IF NOT EXISTS `ordered_carts` (
  `productID` int(11) NOT NULL,
  `orderID` int(11) NOT NULL,
  `Price` float DEFAULT NULL,
  `Quantity` int(11) DEFAULT NULL,
  PRIMARY KEY (`productID`,`orderID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Δομή Πίνακα για τον Πίνακα `orders`
--

DROP TABLE IF EXISTS `orders`;
CREATE TABLE IF NOT EXISTS `orders` (
  `orderID` int(11) NOT NULL AUTO_INCREMENT,
  `order_time` datetime DEFAULT NULL,
  `cust_username` varchar(100) DEFAULT NULL,
  `storeID` int(11) NOT NULL,
  PRIMARY KEY (`orderID`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=24 ;

-- --------------------------------------------------------

--
-- Δομή Πίνακα για τον Πίνακα `products`
--

DROP TABLE IF EXISTS `products`;
CREATE TABLE IF NOT EXISTS `products` (
  `productID` int(11) NOT NULL AUTO_INCREMENT,
  `categoryID` int(11) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `description` text,
  `Price` float DEFAULT NULL,
  `picture` varchar(30) DEFAULT NULL,
  `in_stock` int(11) DEFAULT NULL,
  `enabled` int(11) NOT NULL DEFAULT '1',
  PRIMARY KEY (`productID`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=119 ;

-- --------------------------------------------------------

--
-- Δομή Πίνακα για τον Πίνακα `special_offers`
--

DROP TABLE IF EXISTS `special_offers`;
CREATE TABLE IF NOT EXISTS `special_offers` (
```

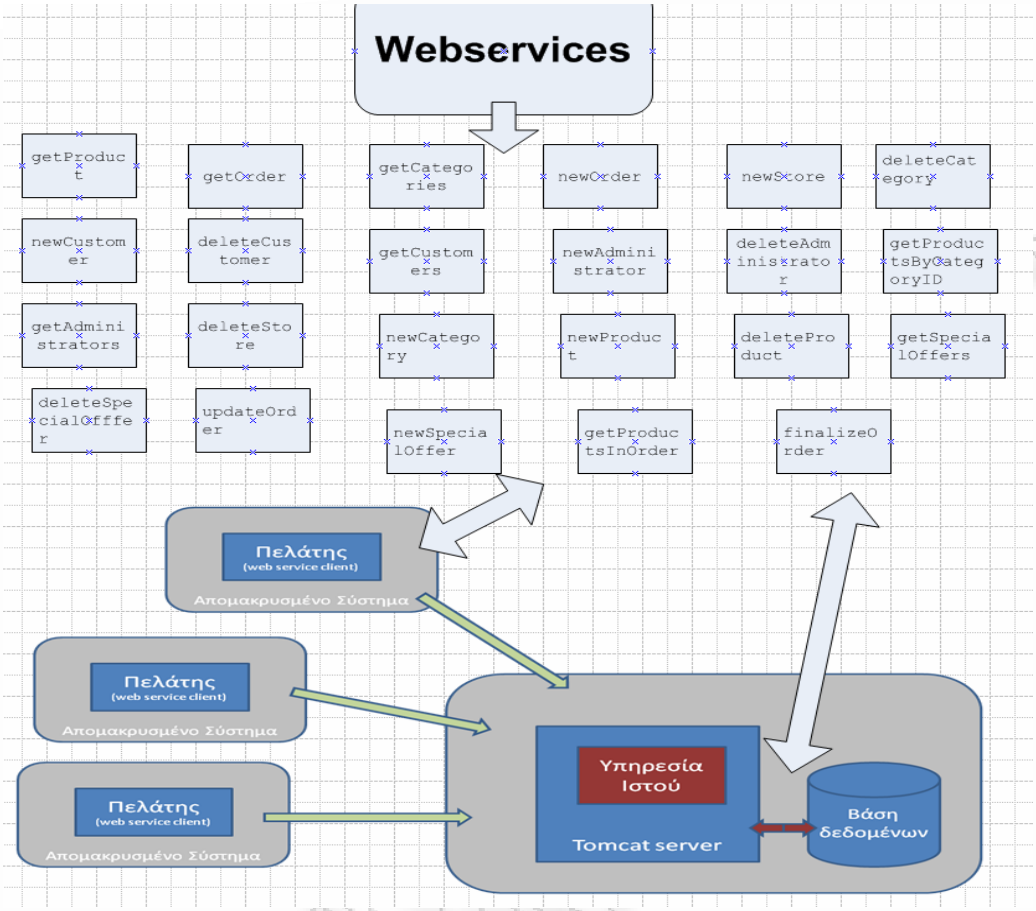
```
  `offerID` int(11) NOT NULL AUTO_INCREMENT,
  `productID` int(11) DEFAULT NULL,
  PRIMARY KEY (`offerID`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=10 ;

-- --------------------------------------------------------

--
-- Δομή Πίνακα για τον Πίνακα `store`
--

DROP TABLE IF EXISTS `store`;
CREATE TABLE IF NOT EXISTS `store` (
  `storeID` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `address` varchar(255) NOT NULL,
  `telephone` varchar(255) NOT NULL,
  PRIMARY KEY (`storeID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

4.2.2. Σχεδιασμός Υπηρεσιών Ιστού

Στη συνέχεια θα παρουσιαστεί η λειτουργικότητα που πρέπει να καλύπτουν οι Υπηρεσίες Ιστού στο Ηλεκτρονικό Κατάστημα. Η λίστα με τις λειτουργίες παρατίθεται στη συνέχεια:

```
getProduct
getOrder
getCategories
newOrder
newStore
deleteCategory
 newCustomer
deleteCustomer
getCustomers
newAdministrator
deleteAdministrator
getAdministrators
deleteStore
getStores
newCategory
newProduct
deleteProduct
getProductsByCategoryID
deleteSpecialOfffer
newSpecialOffer
getSpecialOffers
updateOrder
getProductsInOrder
finalizeOrder
```

Το παρακάτω διάγραμμα παρουσιάζει την αρχιτεκτονική του συστήματος:



4.3. Υλοποίηση των Υπηρεσιών Ιστού

Η υλοποίηση του Web Service έγινε σε περιβάλλον eclipse. Πιο συγκεκριμένα, αρχικά κατασκευάστηκαν κλάσεις οι οποίες αντιστοιχούν στις βασικές οντότητες που περιγράφονται στη βάση δεδομένων.

4.4. Εργαλεία και Τεχνολογίες για την ανάπτυξη των Υπηρεσιών Ιστού.

Στην ενότητα αυτή παρουσιάζονται τα εργαλεία και οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη του λογισμικού. Για την

εγκατάσταση της εφαρμογής απαιτείται Apache Tomcat Server ≥5.5 (apache tomcat 6.0.18) και εγκατεστημένη Java ≥1.5.(jdk-6u11-windows-i586-p , jre-6u11-windows-i586-p) Επίσης, για τη σύνδεση με τη βάση, ο Tomcat χρειάζεται τη βιβλιοθήκη σύνδεσης με MySQL.

Τέλος, για την ανάπτυξη του κώδικα του υποσυστήματος χρησιμοποιήθηκε το περιβάλλον ανάπτυξης λογισμικού που ονομάζεται Eclipse.

4.4.1. Eclipse

Για την ανάπτυξη του κώδικα χρησιμοποιήθηκε το Eclipse. Το eclipse αποτελεί ένα SDK (Software Development Kit), ένα ολοκληρωμένο δηλαδή περιβάλλον μέσα από το οποίο μπορούμε να γράψουμε και να εκτελέσουμε κώδικα. Το περιβάλλον αυτό καθώς και όλα τα υπόλοιπα προγράμματα που χρειάζονται για να εκτελέσουμε κώδικα σε Java ή/και C/C++ είναι ελεύθερης διανομής (freeware) και ανοικτού κώδικα (open source). Επιπλέον αποτελεί το ανερχόμενο περιβάλλον ανάπτυξης κώδικα, καθώς χρησιμοποιείται από ολοένα και περισσότερους χρήστες αλλά και εταιρείες.

4.4.2. MySQL Server

Για την χρήση τη βάσης δεδομένων που περιγράφηκε σε προηγούμενη ενότητα, χρησιμοποιήθηκε MySQL Server. Η MySQL ανήκει στην γενική κατηγορία των DBMS (Database Management Systems) που έχουν σαν πρωταρχικό τους ρόλο την αποθήκευση των δεδομένων, την ελεγχόμενη και ασφαλή πρόσβαση στις πληροφορίες, και την διαχείριση των δικαιωμάτων με τα οποία οι χρήστες μπορούν να επέμβουν και να αλλάξουν στοιχεία πληροφοριών. Κάθε φορά που ένας χρήστης προσπαθεί να διαχειριστεί ένα όγκο δεδομένων προσθέτοντας νέα στοιχεία ή αφαιρώντας κάποια άλλα που δεν ισχύουν πια, το DBMS είναι υπεύθυνο να ακολουθήσει αυτή την

διαδικασία από την αρχή της ενεργοποίησης της μέχρι το τέλος της. Παίζει το ρόλο του "μεσάζων" γιατί απλά το DBMS δεν αφήνει το χρήστη να έχει άμεση πρόσβαση στα δεδομένα. Οι εντολές απευθύνονται αποκλειστικά στο DBMS και αφού ελεγχθούν για την εγκυρότητα τους τότε το σύστημα αναλαμβάνει την μεταφορά της πληροφορίας στο χρήστη που την αναζήτησε. Η MySQL ανήκοντας στην κατηγορία των DBMS μπορεί να αναλάβει την διαχείριση πολλών βάσεων μαζί. Κατά την εγκατάσταση έχει ήδη δημιουργηθεί μια βάση με το όνομα mysql η οποία χρησιμοποιείται αποκλειστικά για την καταγραφή πληροφοριών του συστήματος.

4.4.3. Υλοποίηση των Web Services

Η υλοποίηση του Web Service έγινε σε περιβάλλον eclipse. Πιο συγκεκριμένα, αρχικά κατασκευάστηκαν κλάσεις οι οποίες αντιστοιχούν στις βασικές οντότητες που περιγράφονται στη βάση δεδομένων. Ακολουθούν οι κλάσεις αυτές με τα πεδία τους….

```
public class Administrator {

                String username;
                String name;
                String surname;
                String password;
                boolean superadmin;
}

public class Category {
        long categoryID;
        String name;
        long parent;
        String description;
        String picture;
```

```
}

public class Customer  {
    private java.lang.String username;
    private java.lang.String password;
    private java.lang.String name;
    private java.lang.String AFM;
    private java.lang.String email;
    private java.lang.String telephone;
    private java.lang.String address;
}

public class Order  {
    private java.lang.String username;
    private int productId;
    private int storeId;
    private int quantity;
    private long orderid;
}

public class Product {
        long productID;
        long categoryID;
        String name;
        String manufacturer;
        String description;
        float price;
        String picture;
        int availability;
        boolean enabled;
}
public class Store {
        long storeID;
        String name;
        String address;
        String telephone;
}
```

Στη συνέχεια κατασκευάζεται η κλάση WebStore η οποία αντιστοιχεί στο Web Service. Στη κλάση αυτή ορίζονται όλες οι κλήσεις του Web Service που θέλουμε να κατασκευάσουμε.

Για παράδειγμα, η μέθοδος newCustomer δέχεται ως όρισμα ένα αντικείμενο τύπου Customer προκειμένου να το εισάγει στη βάση:

```
public Response newCustomer(Customer customer){
                Response resp = db.newCustomer(customer);
```

```
                db.close();
                return resp;
        }
```

Κάθε τέτοια μέθοδος χρησιμοποιεί την αντίστοιχη μέθοδο από τη κλάση Database. Στην κλάση αυτή γίνεται η επικοινωνία με τη βάση δεδομένων. Ο κατασκευαστής της κλάσης δημιουργεί τη σύνδεση με τον My SQL Server που θεωρείται ότι βρίσκεται στο τοπικό μηχάνημα:

```
public Database(){

              //Define URL of database server for
              // database named shop on  localhost
              // with the default port number 3306.
              String url ="jdbc:mysql://localhost:3306/shop";

              try {
                try {

        Class.forName("com.mysql.jdbc.Driver");
                        } catch (ClassNotFoundException e) {
                                e.printStackTrace();
                        }
                        con  =  DriverManager.getConnection(url,"root",
"sadm");
                } catch (SQLException e) {
                        e.printStackTrace();
                }

        }
```

Αφού καλεστεί ο κατασκευαστής και αρχικοποιήσει τη σύνδεση με τη βάση, μπορεί να γίνει χρήση κάποιας από τις υπόλοιπες μεθόδους της κλάσης Database.java. Για παράδειγμα, στη μέθοδο newCustomer της WebStore καλείται η ομώνυμη μέθοδος από τη κλάση Database:

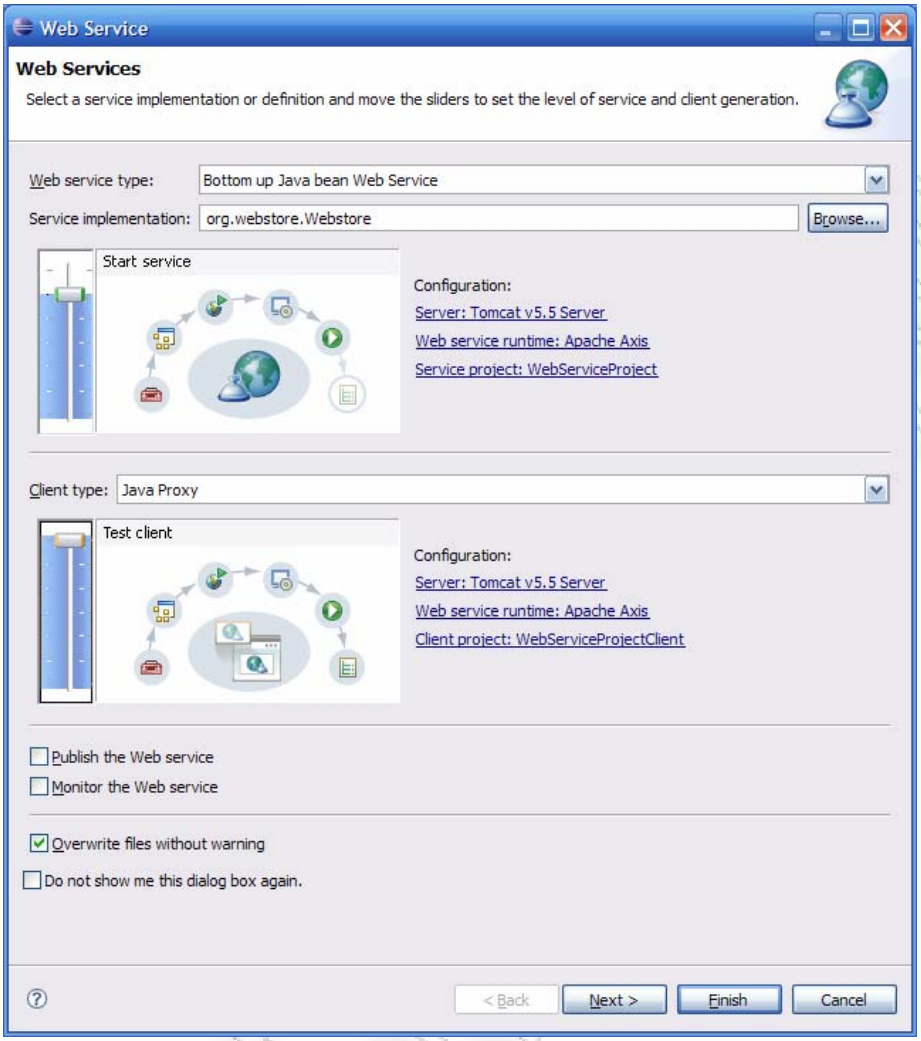
```
public Response newCustomer(Customer customer){
        Response resp = new Response(false,"");
        Statement stmt;
```

```
        try {
                stmt = con.createStatement();
                String insertString = "INSERT INTO customers
(cust_username ,cust_password ,cust_name ,cust_email ,cust_address
,cust_phone , cust_afm, cust_date) VALUES('" +customer.getUsername()
+ "','" + customer.getPassword() + "','" + customer.getName() +
"','" + customer.getEmail()+ "','" + customer.getAddress()  + "','"
+ customer.getTelephone() + "','"+ customer.getAFM() +"' , CURDATE()
)";

stmt.executeUpdate(insertString);
        //if statement executes without exception, set status of resp
to true
        resp.setState(true);
        } catch (SQLException e) {
                resp.setInfo(e.getMessage());
}

        return resp;
}
```

Ο παραπάνω κώδικας εκτελεί ένα insert query στη βάση εισάγοντας ένα νέο πελάτη (Customer) με τα χαρακτηριστικά του αντικειμένου Customer που δίνεται ως όρισμα στη μέθοδο.

Αφού κατασκευαστούν όλες οι μέθοδοι στη κλάση WebStore μπορεί να παραχθεί το Web Service. Αυτό πραγματοποιείται μέσω του eclipse:

Web Service:



Στο παραπάνω σχήμα, φαίνεται πως μέσω Eclipse, δημιουργείται το Web Service και παράγονται αυτόματα οι κλήσεις του χρησιμοποιώντας το κώδικα από τις μεθόδους της κλάσης WebStore. Παράλληλα, εκτός από το Web Service παράγεται και Web Service Client ο οποίος με τη χρήση της τεχνολογίας JSP είναι προσβάσιμος μέσω Web για τη δοκιμή του Web Service.

Με τη διαδικασία που φαίνεται στο προηγούμενο σχήμα λοιπόν, Web Service και Web Service Client φορτώνονται σε τοπικό Apache Tomcat για τις δοκιμές. Το Web Service είναι στη συνέχεια προσβάσιμο στο URL:

http://localhost:8080/WebServiceProject/services

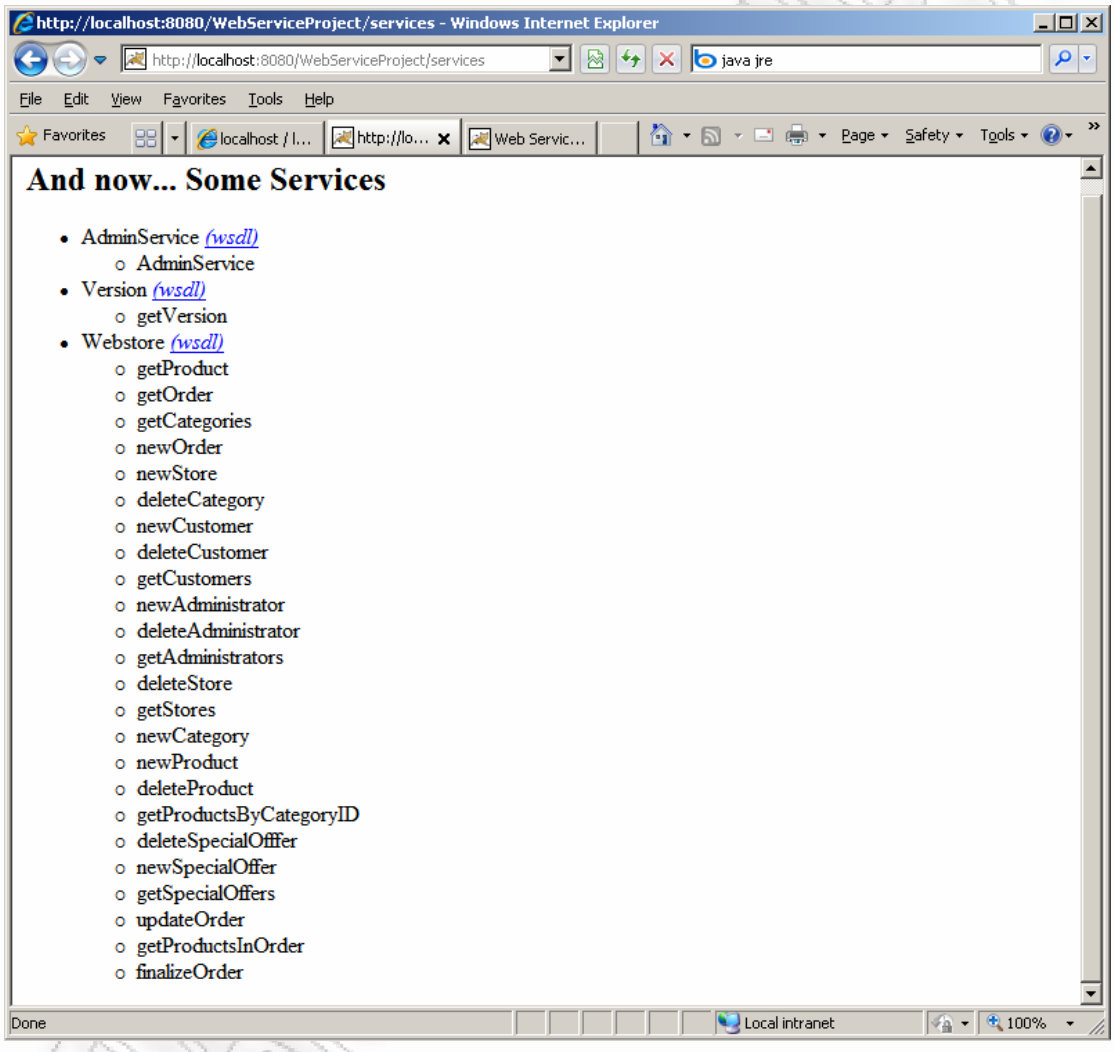
http://localhost:8080/WebServiceProjectClient/sampleWebstoreProxy/TestClient.jsp

Όλες οι κλάσεις του Web Service client παράχθηκαν αυτόματα από τη χρήση του εργαλείου wsdl2java με βάση τη WebStore.

Η κλάση XMLOrder έχει μια μόνο μέθοδο τη void serialize(), η οποία παράγει το έγγραφο XML στο αρχείο test.xml. Το όνομα του αρχείου θα μπορούσε να είναι μοναδικό, αλλά για λόγους οικονομίας (να μη δημιουργούνται σκουπίδια στο filesystem) έχει πάντα το ίδιο όνομα και αναδημιουργείται για κάθε νέα παραγγελία ή προσθήκη προϊόντος σε υπάρχουσα παραγγελία. Η serialize() χρησιμοποιεί τον SAX Parser προκειμένου να δημιουργεί συντακτικά σωστά κείμενα XML.

5. Δοκιμές συστήματος και στιγμιότυπα λειτουργίας

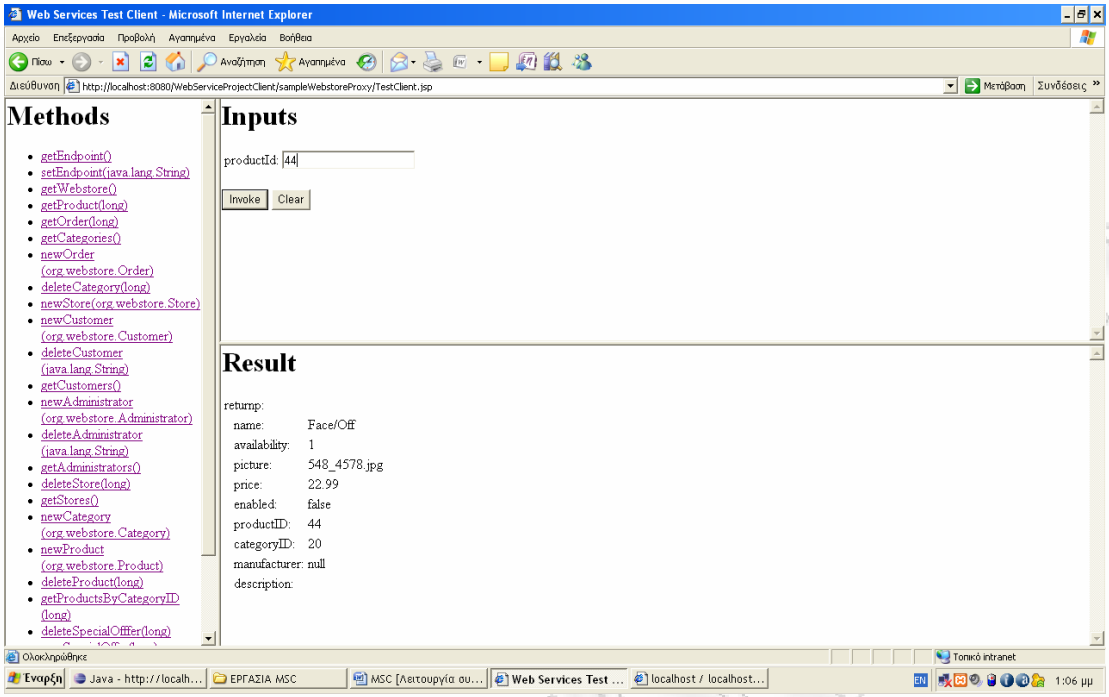
Στο παρακάτω στιγμιότυπο φαίνεται η λίστα με τα operations που δημοσιεύει το Web Service καθώς και οι σύνδεσμοι με τα αρχεία WSDL που μπορούν να χρησιμοποιηθούν για τη δημιουργία του client.



Σχήμα 3 Λίστα Υπηρεσιών Ιστού

Στιγμιότυπο getProduct( )

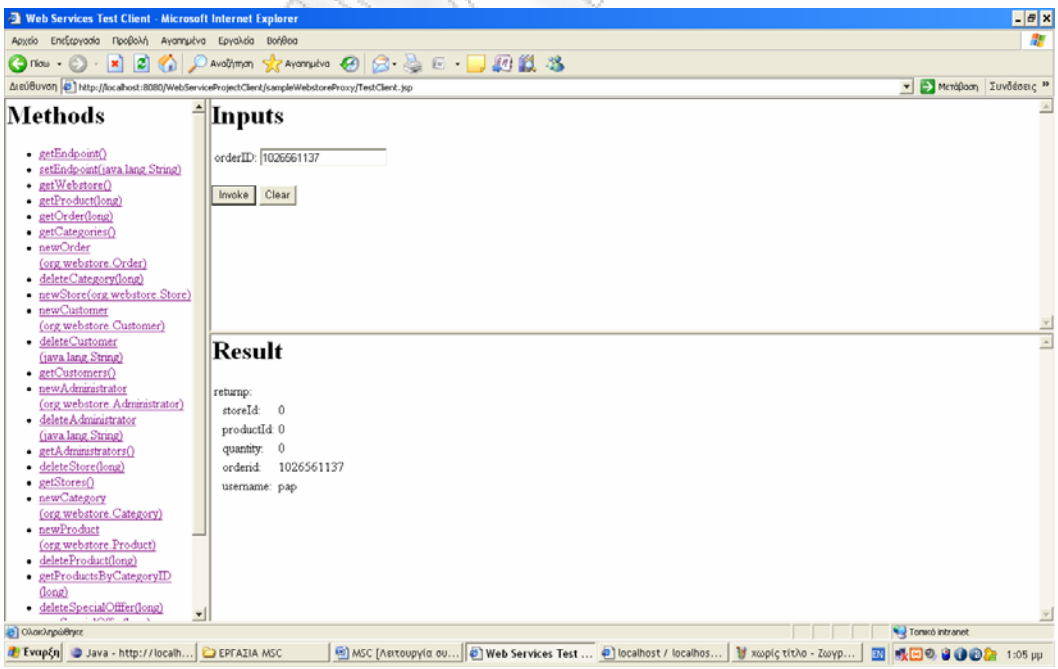
Στο παρακάτω στιγμιότυπο φαίνεται η δοκιμή της κλήσης getProduct( ) από τη σελίδα του ProxyClient. Φαίνεται δίνοντας το productID 44 να επιστρέφεται από τη βάση δεδομένων το αντίστοιχο προϊόν.



Σχήμα 4

Στιγμιότυπο getOrder( )

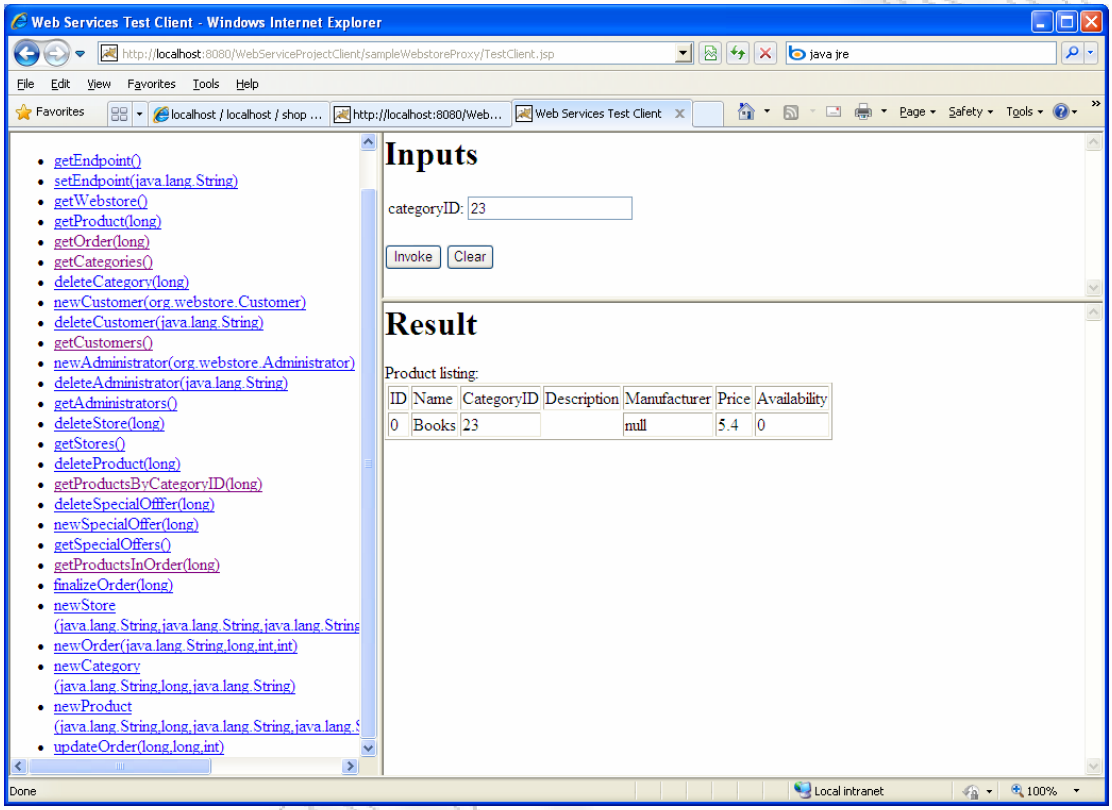
Στο παρακάτω στιγμιότυπο φαίνεται η δοκιμή της κλήσης getOrder( ) από τη σελίδα του ProxyClient. Φαίνεται δίνοντας το orderID : 1026561137 να επιστρέφεται από τη βάση δεδομένων η αντίστοιχη παραγγελία.



Σχήμα 5

Στιγμιότυπο getProductsByCategoryID(long)

Ομοίως και για την κλήση της μεθόδου getProductsByCategoryID(long) όταν δώσουμε το categoryID.

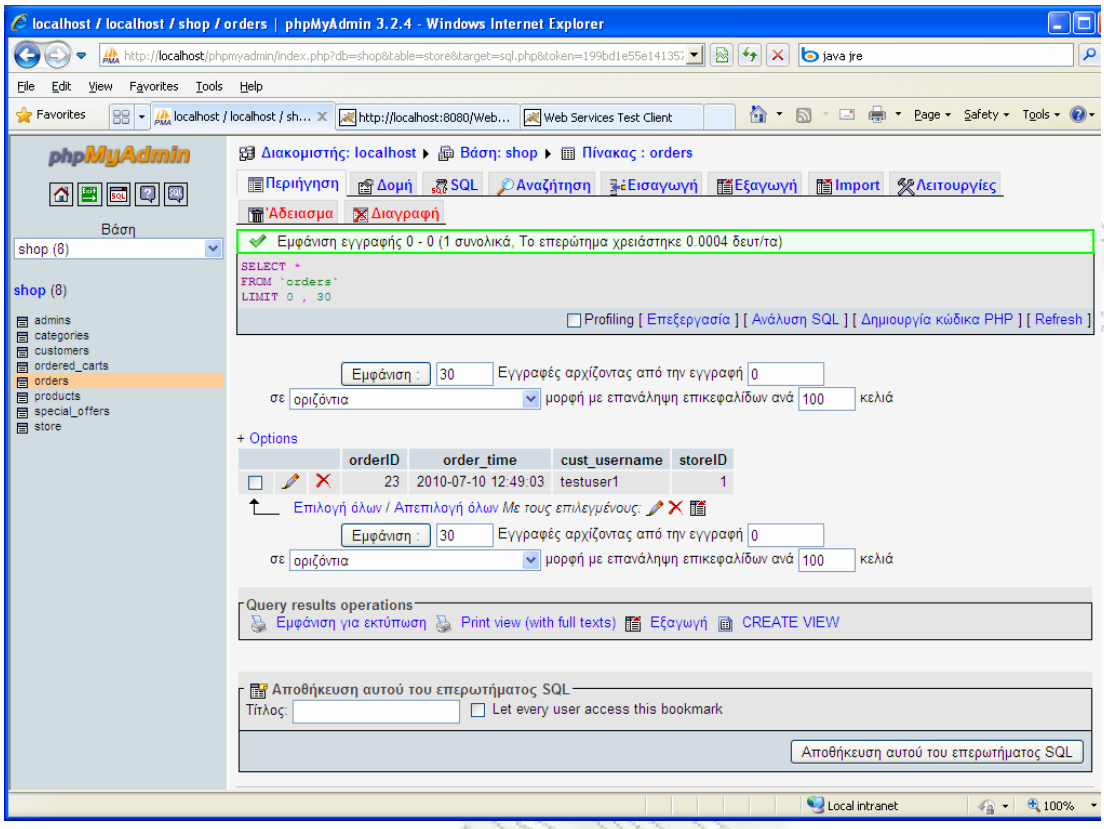


Σχήμα 6

Στιγμιότυπο newOrder(java.lang.String username,long productID, long storeID, int quantity)

Στο παρακάτω στιγμιότυπο φαίνεται η δοκιμή της κλήσης της μεθόδου newOrder() από τη σελίδα του ProxyClient. Η κλήση αυτής της μεθόδου δημιουργεί μία νέα παραγγελία στη βάση δεδομένων και δίνει αυτόματα ένα νέο ID στην νέα παραγγελία που είναι μοναδικό.

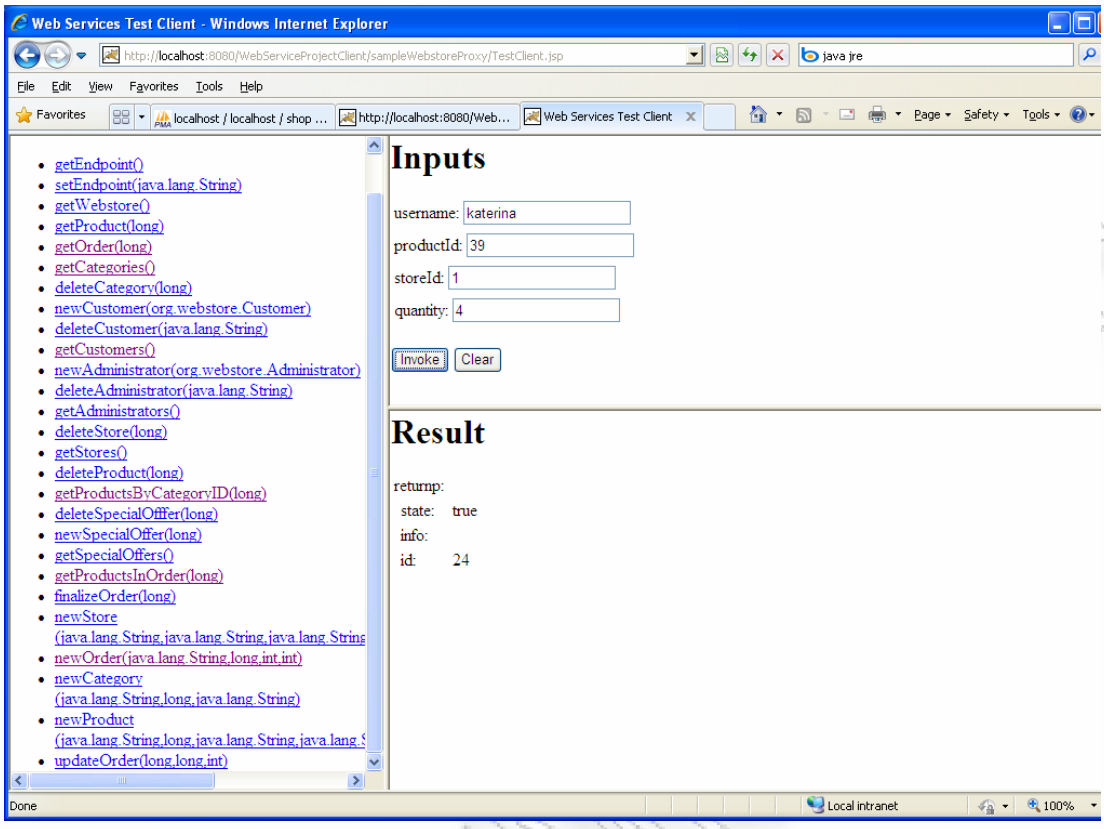
Στην Β.Δ υπάρχουν οι παρακάτω παραγγελίες:



Σχήμα 7

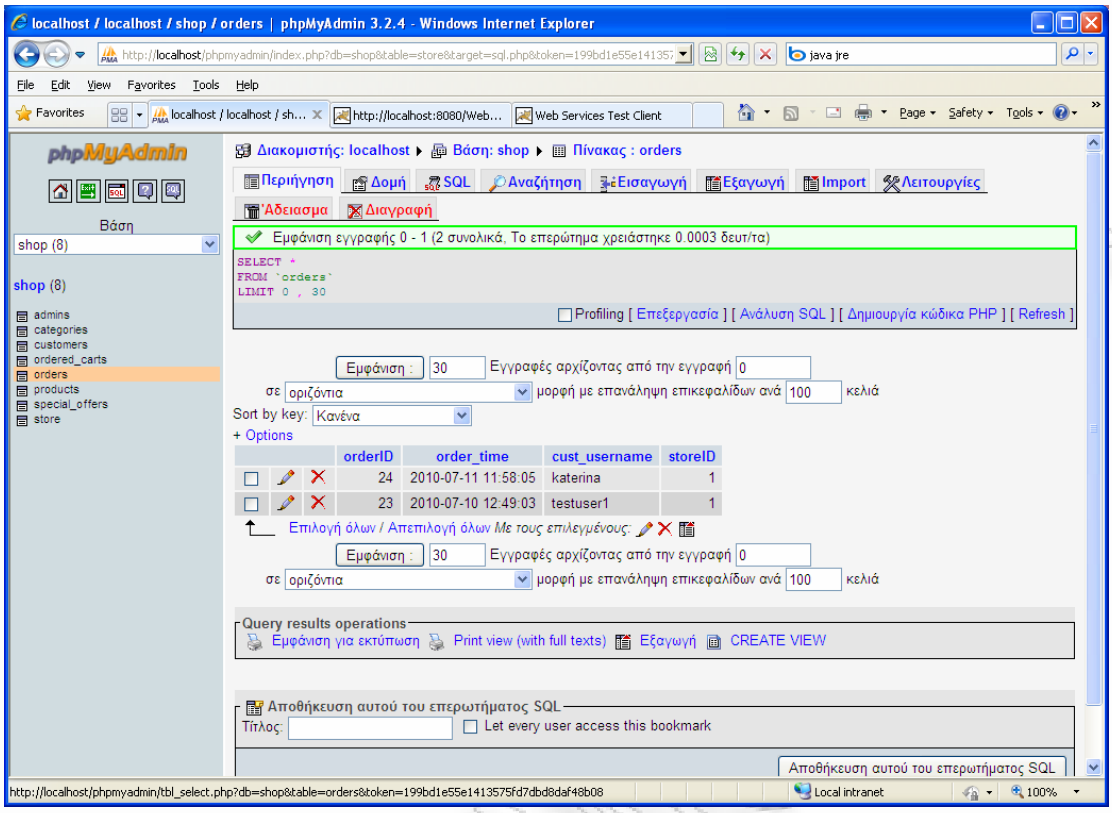
Στιγμιότυπο newOrder(org.webstore.Order)

Εκτελώντας την κλήση της μεθόδου newOrder() παίρνουμε το νέο ID της νέας παραγγελίας:



Σχήμα 8

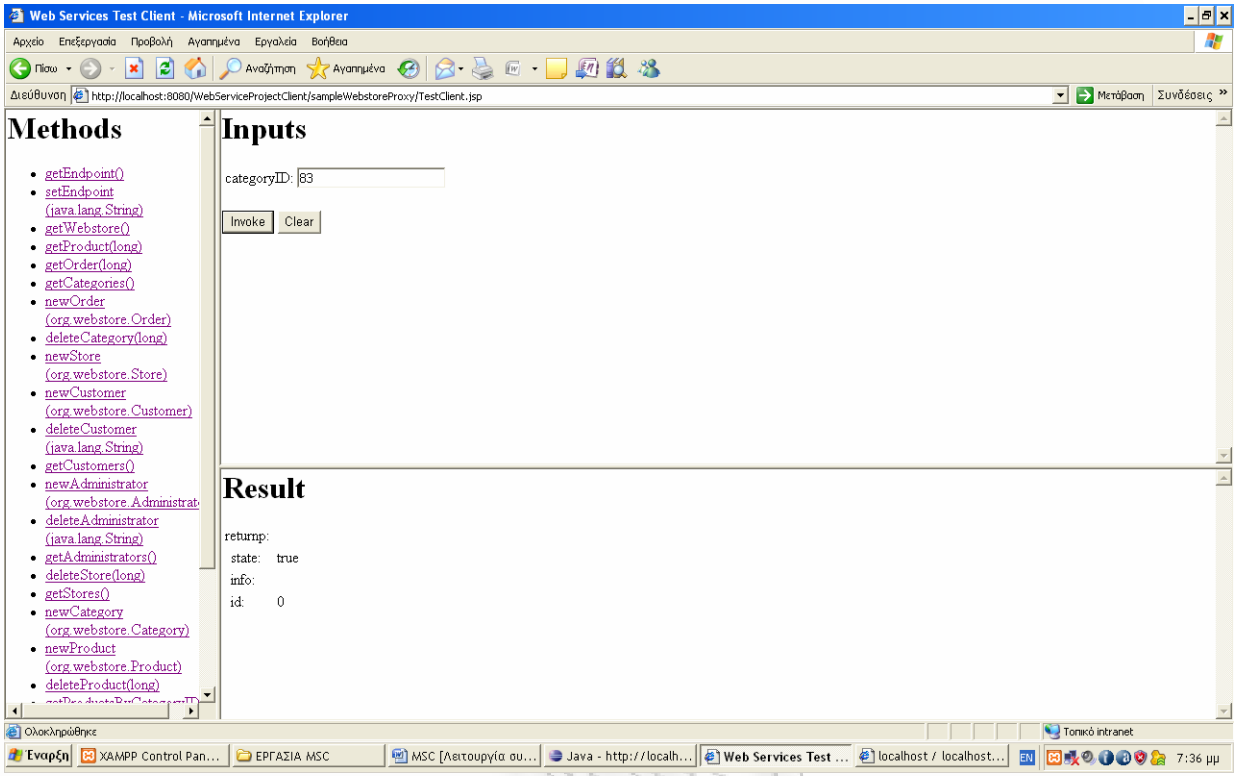
Το Web Service επιστρέφει μετά τη καταχώρηση της νέας παραγγελίας με ID=24. Και στην Β.Δ. φαίνεται η νέα παραγγελία με το ID: 24



Σχήμα 9

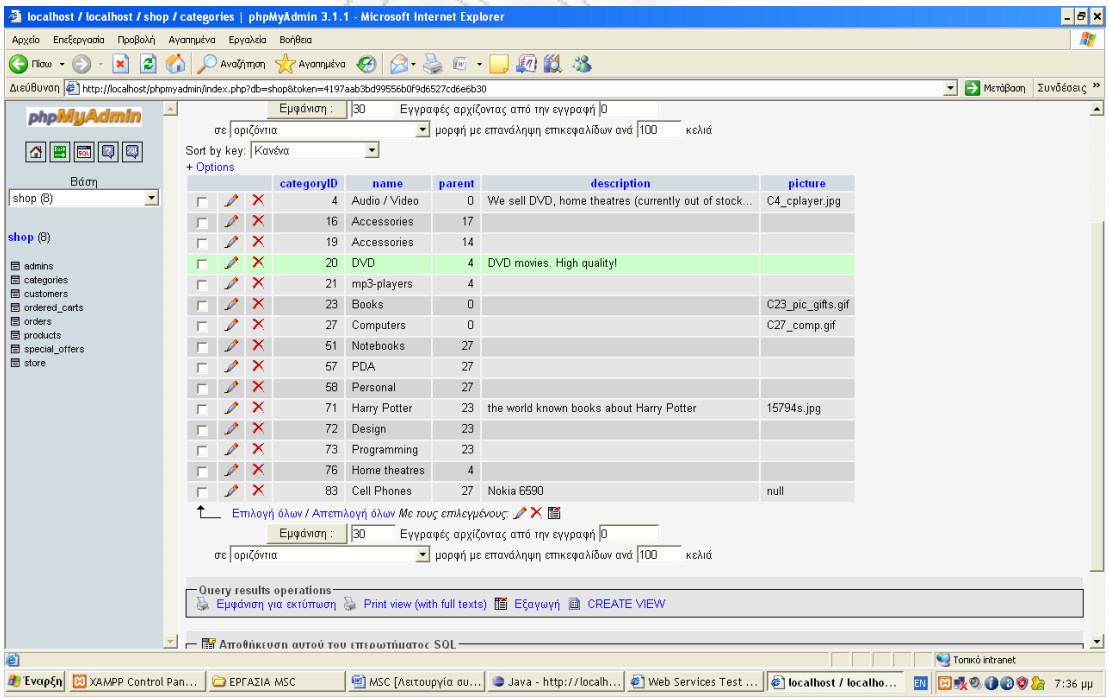
Στιγμιότυπο deleteCategory(long)

Στο επόμενο στιγμιότυπο φαίνεται η κλήση της μεθόδου deleteCategory(long) η οποία αφαιρεί μία κατηγορία στη βάση δεδομένων. Δίνουμε το ID και αφαιρεί την κατηγορία από την Β.Δ.



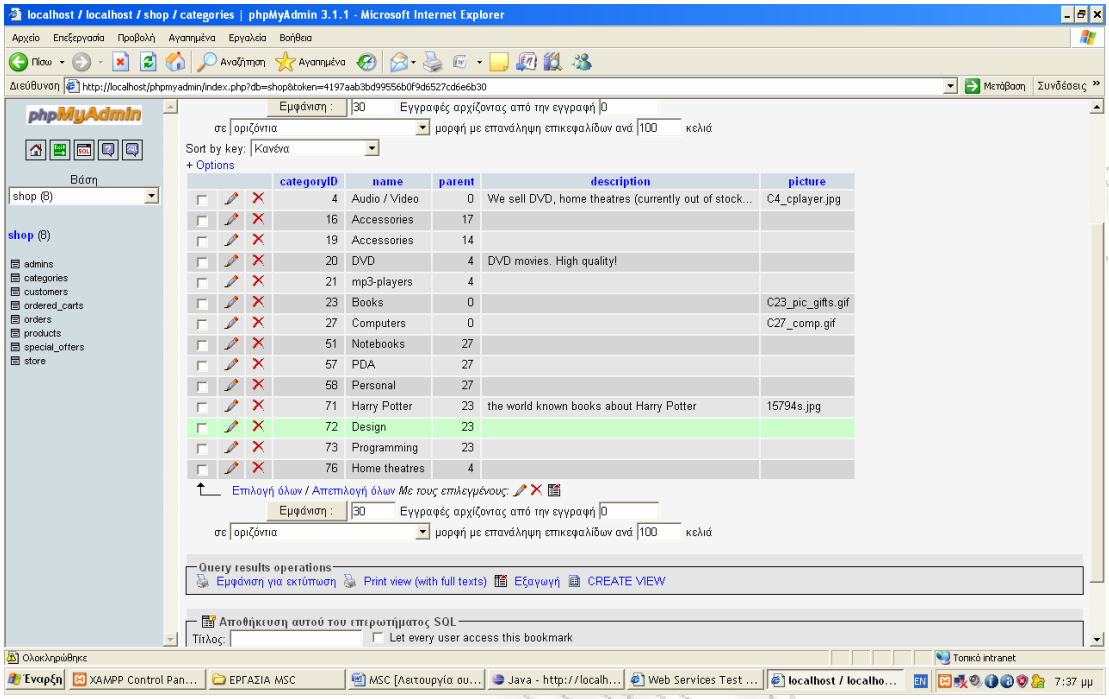
Σχήμα 10

Η Β.Δ. Πριν:



Σχήμα 11

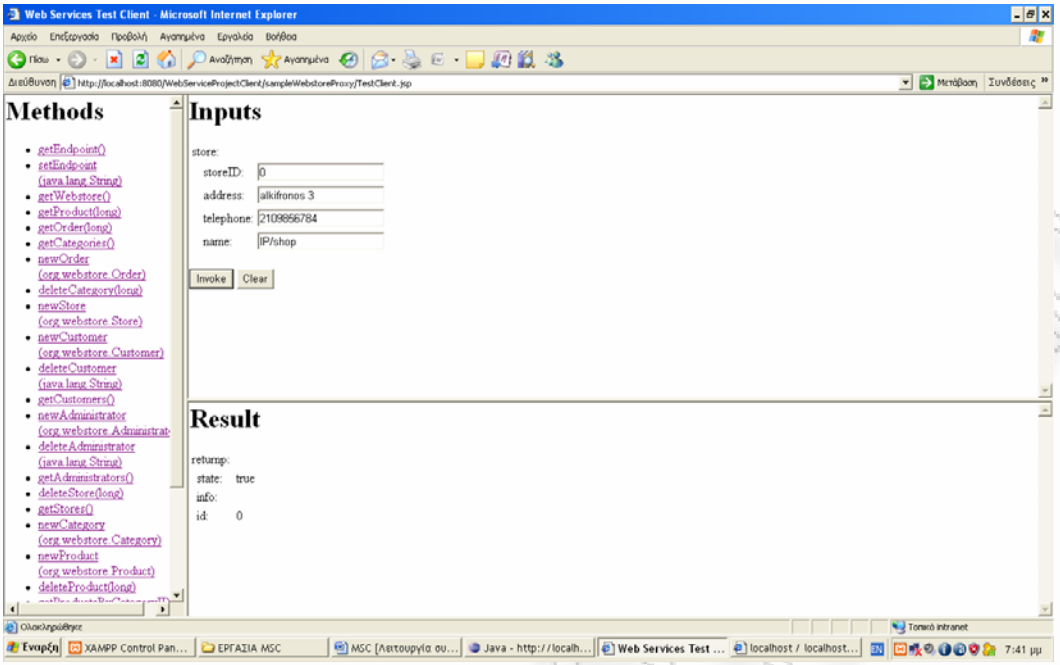
Η Β.Δ. μετά:



Σχήμα 12

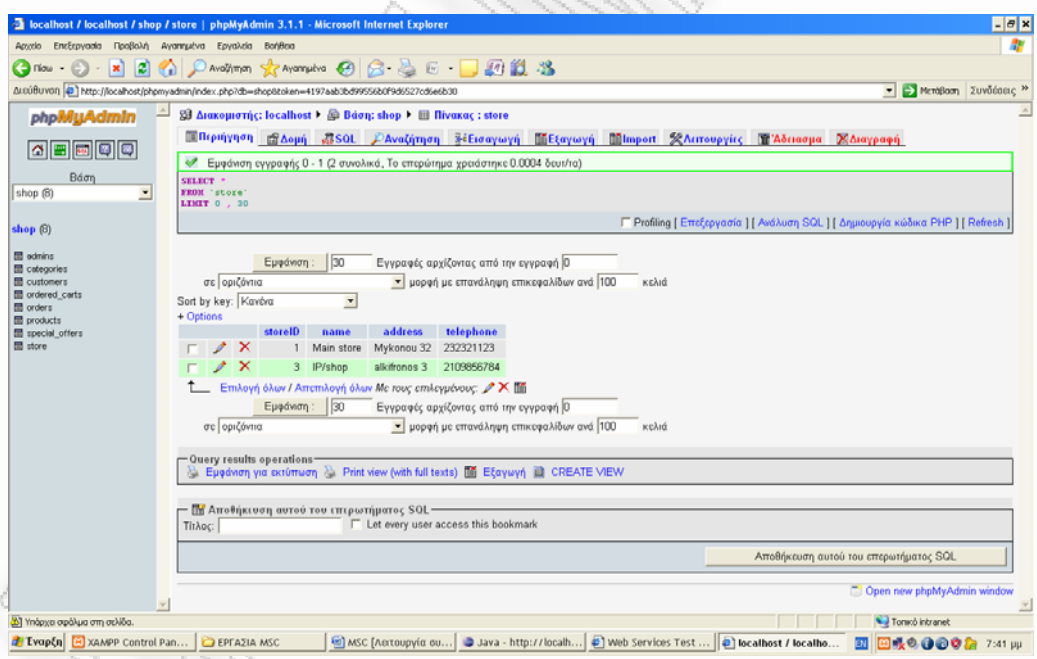
Στιγμιότυπο newStore(org.webstore.Store)

Στο επόμενο στιγμιότυπο φαίνεται η κλήση της μεθόδου newStore(org.webstore.Store) η οποία προσθέτει ένα καινούρια κατάστημα στη βάση δεδομένων.



Σχήμα 13

Και στην Β.Δ. φαίνεται παρακάτω:



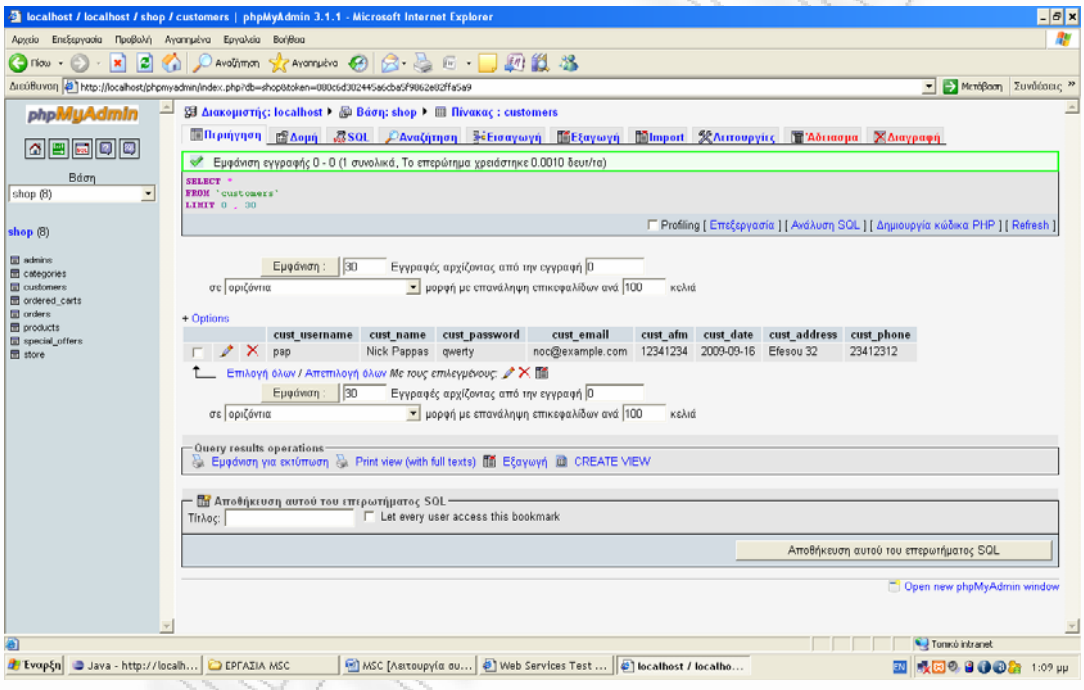
Σχήμα 14

Στιγμιότυπο newCustomer

Στο επόμενο στιγμιότυπο φαίνεται η κλήση της μεθόδου newCustomer η οποία προσθέτει ένα καινούριο πελάτη στη βάση δεδομένων. Εδώ θα διακρίνουμε δύο περιπτώσεις

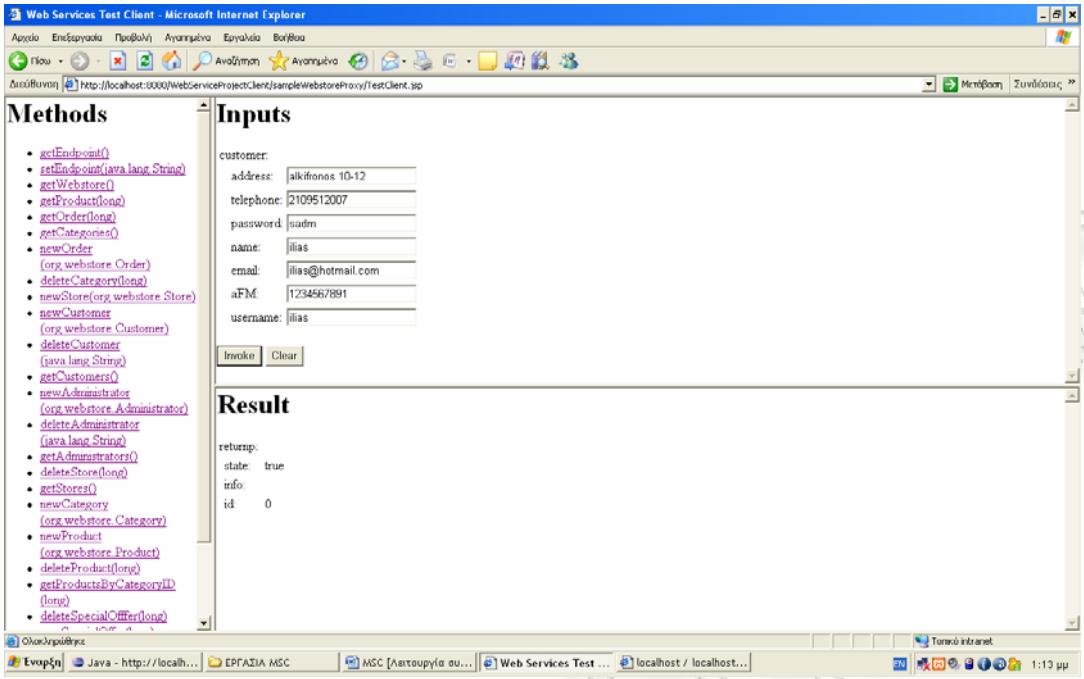
- Εισαγωγή νέου χρήστη στην Β.Δ
- Η βάση δεν επιτρέπει την εισαγωγή νέου χρήστη με το ίδιο username, που είναι μοναδικό κλειδί στον αντίστοιχο πίνακα customers.

Η βάση μας όπως φαίνεται παρακάτω έχει μόνο έναν πελάτη με username :pap



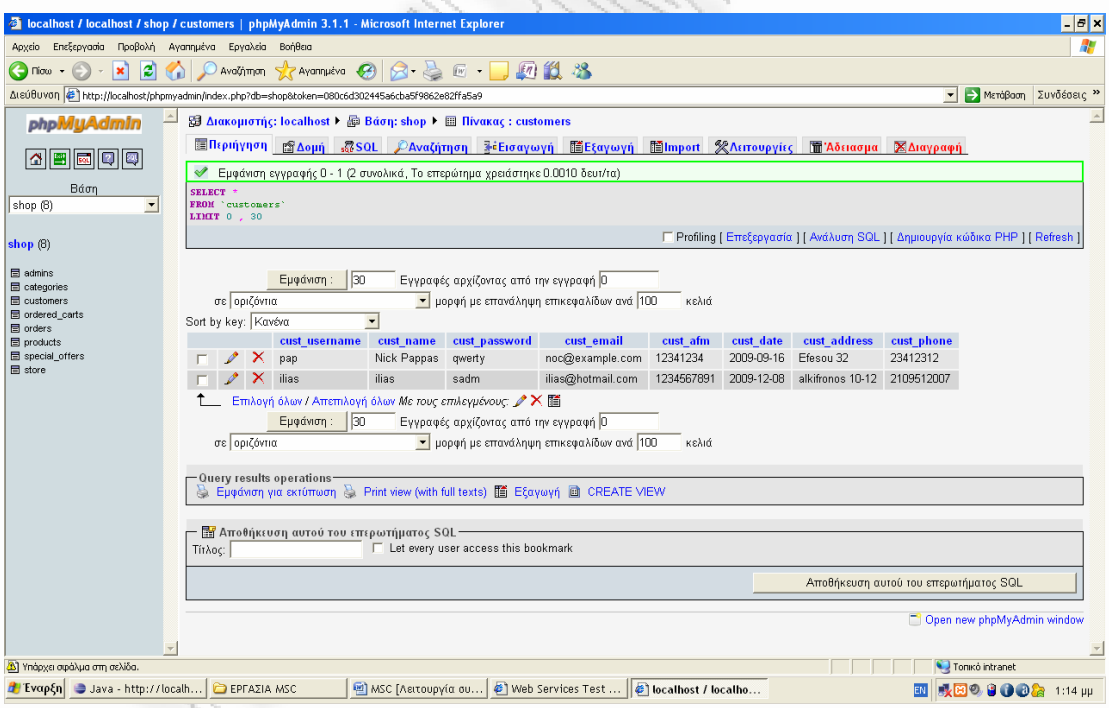
Σχήμα 15

Αν καλέσουμε την κλήση της μεθόδου newCustomer τότε θα εισάγουμε έναν νέο πελάτη :



Σχήμα 16

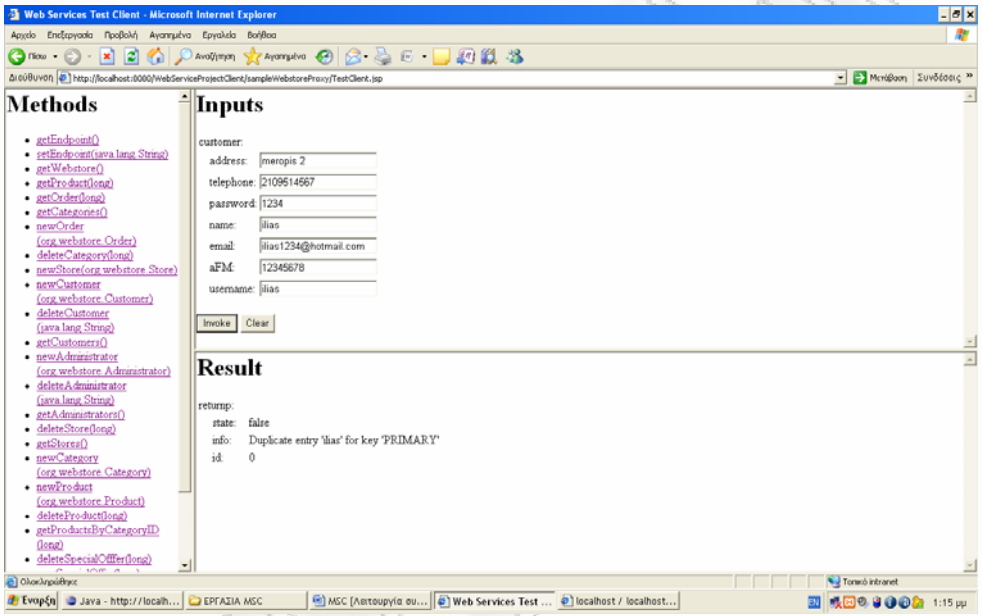
Και στην Β.Δ βλέπουμε ότι έχει προστεθεί ο πελάτης με username:ilias



Σχήμα 17

Στιγμιότυπο newCustomer

Στο επόμενο στιγμιότυπο φαίνεται η κλήση της μεθόδου newCustomer η οποία προσθέτει ένα καινούριο πελάτη στη βάση δεδομένων. Στη περίπτωση αυτή, η εισαγωγή είναι ανεπιτυχής, και η κλήση επιστρέφει το μήνυμα λάθους. Η βάση δεν επιτρέπει την εισαγωγή νέου χρήστη με το ίδιο username, που είναι μοναδικό κλειδί στον αντίστοιχο πίνακα customers



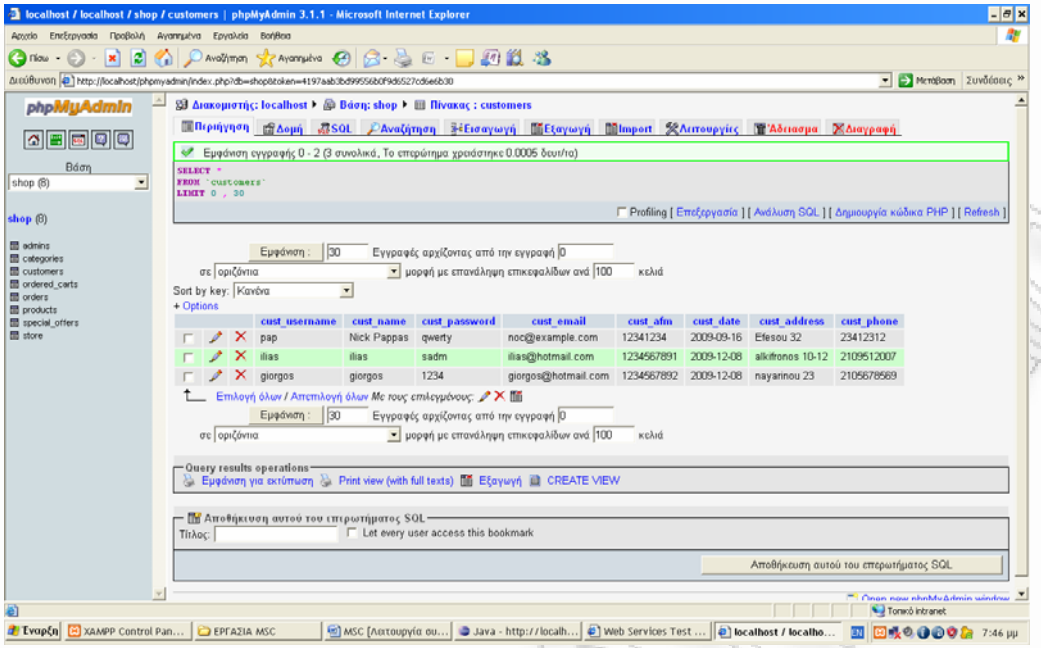
Σχήμα 18

Στιγμιότυπο newAdministrator

Ομοίως γίνεται και με την κλήση της μεθόδου newAdministrator ,η οποία προσθέτει ένα νέο διαχειριστή στη Β.Δ.

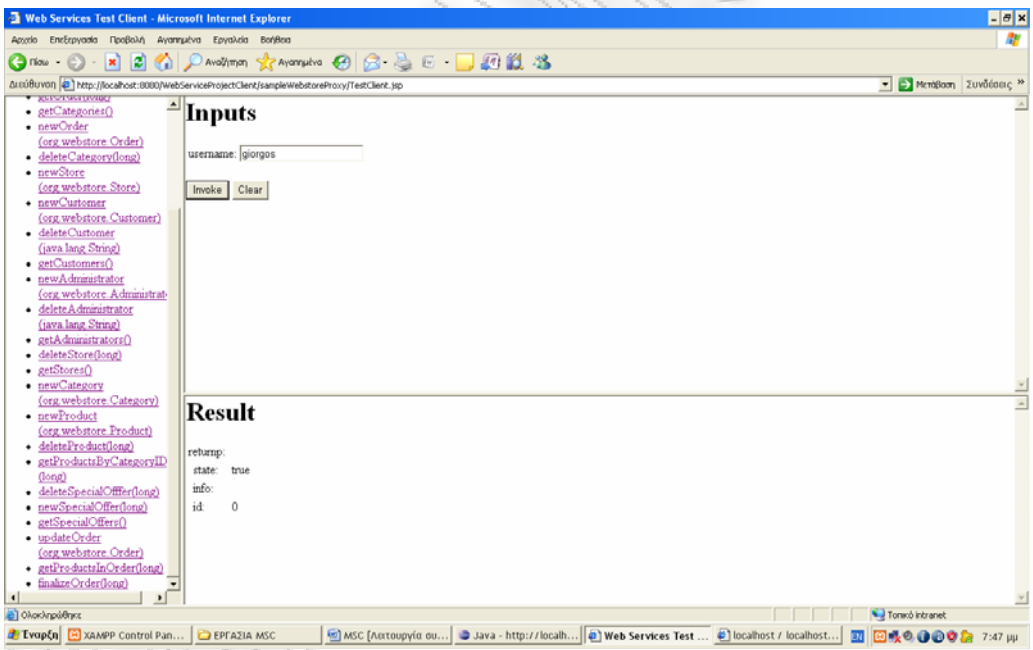
Στιγμιότυπο deleteCustomer

Στο επόμενο στιγμιότυπο φαίνεται η κλήση της μεθόδου deleteCustomer η οποία αφαιρεί ένα πελάτη στη βάση δεδομένων. Η Β.Δ. πριν



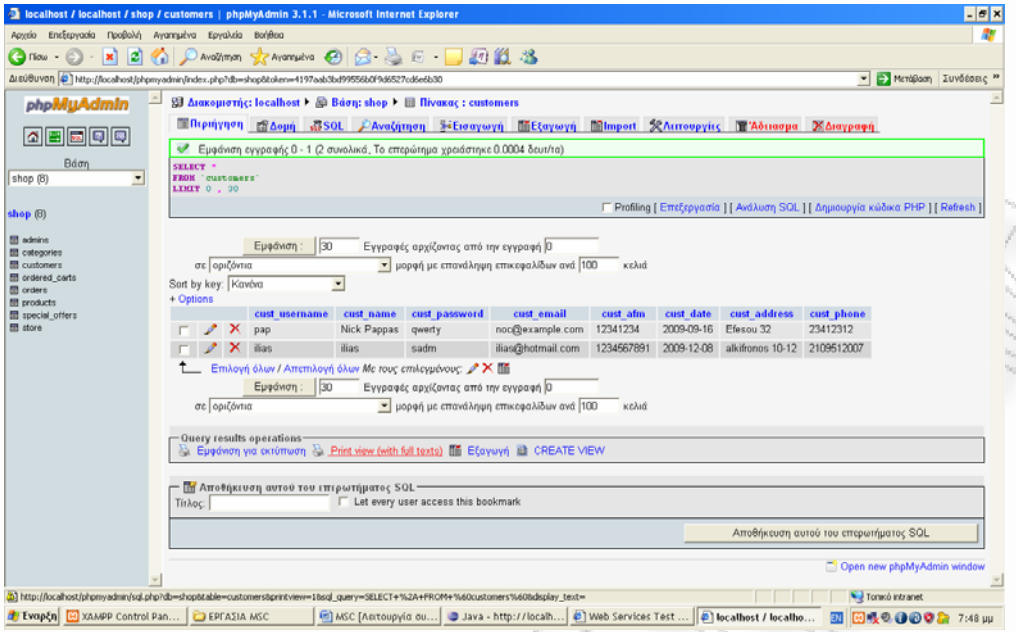
Σχήμα 19

Εκτελώντας την κλάση της μεθόδου deleteCustomer :



Σχήμα 20

Βλέπουμε την Β.Δ. μετά:



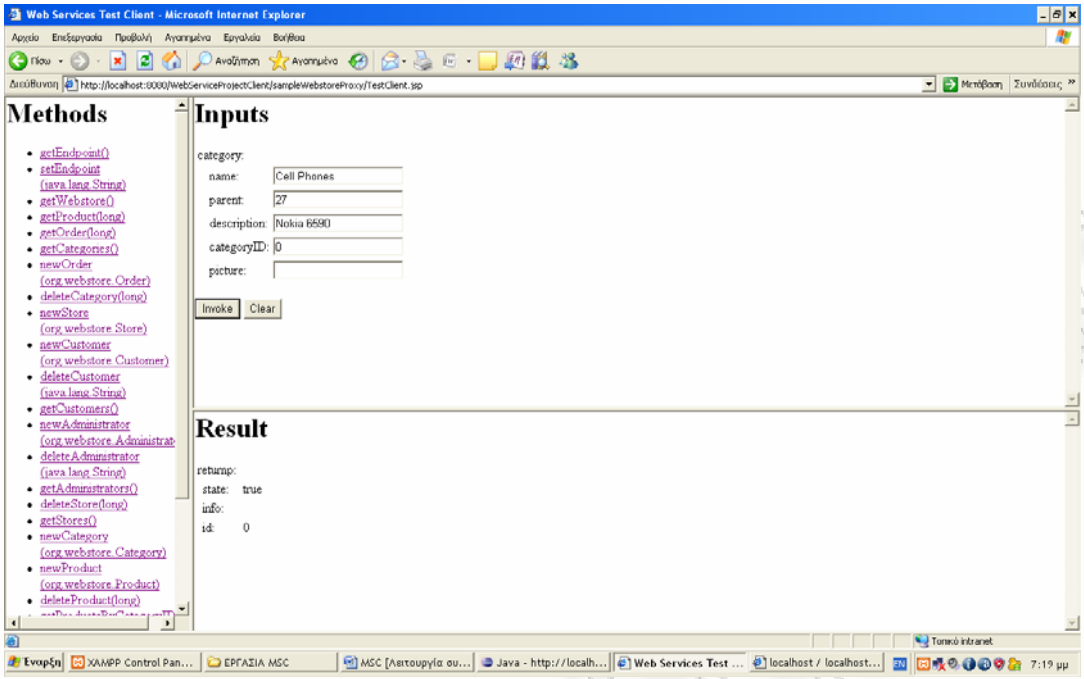
Σχήμα 21

Στιγμιότυπο deleteStore(long), deleteAdministrator, deleteProduct(long)

Ομοίως γίνεται και με την κλήση των μεθόδων deleteStore(long) , deleteAdministrator, deleteProduct(long) οι οποίες αφαιρούν από την Β.Δ. αντίστοιχα ένα κατάστημα έναν διαχειριστή και ένα προιόν.

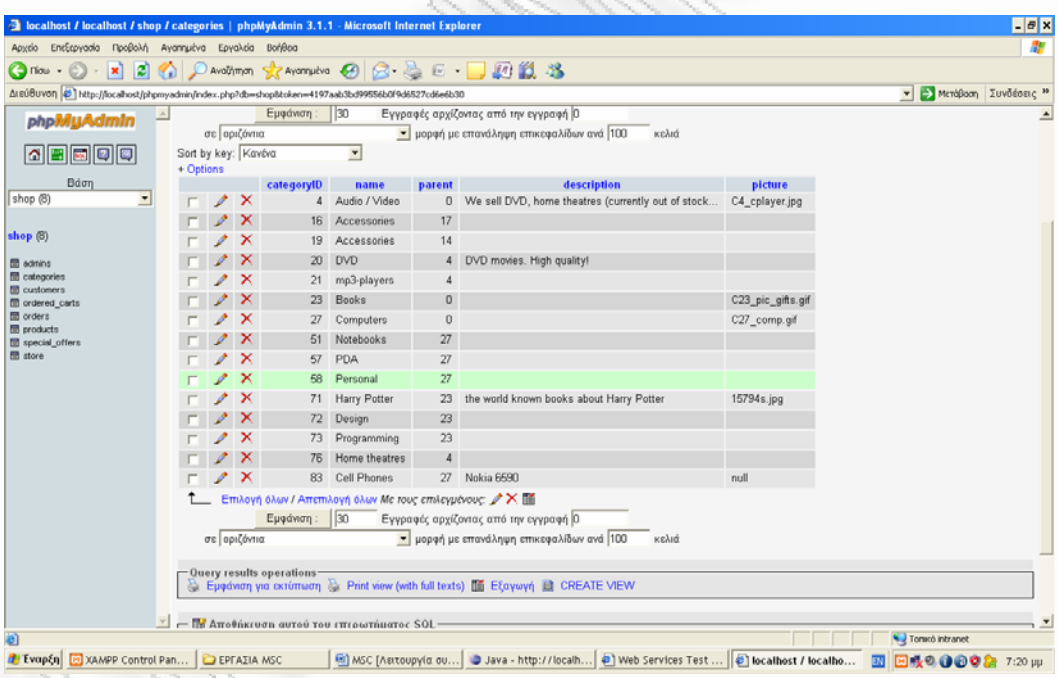
Στιγμιότυπο newCategory

Στο επόμενο στιγμιότυπο φαίνεται η κλήση της μεθόδου newCategory η οποία προσθέτει μια καινούρια κατηγορία στη βάση δεδομένων. Το category ID δίνεται αυτόματα.



Σχήμα 22

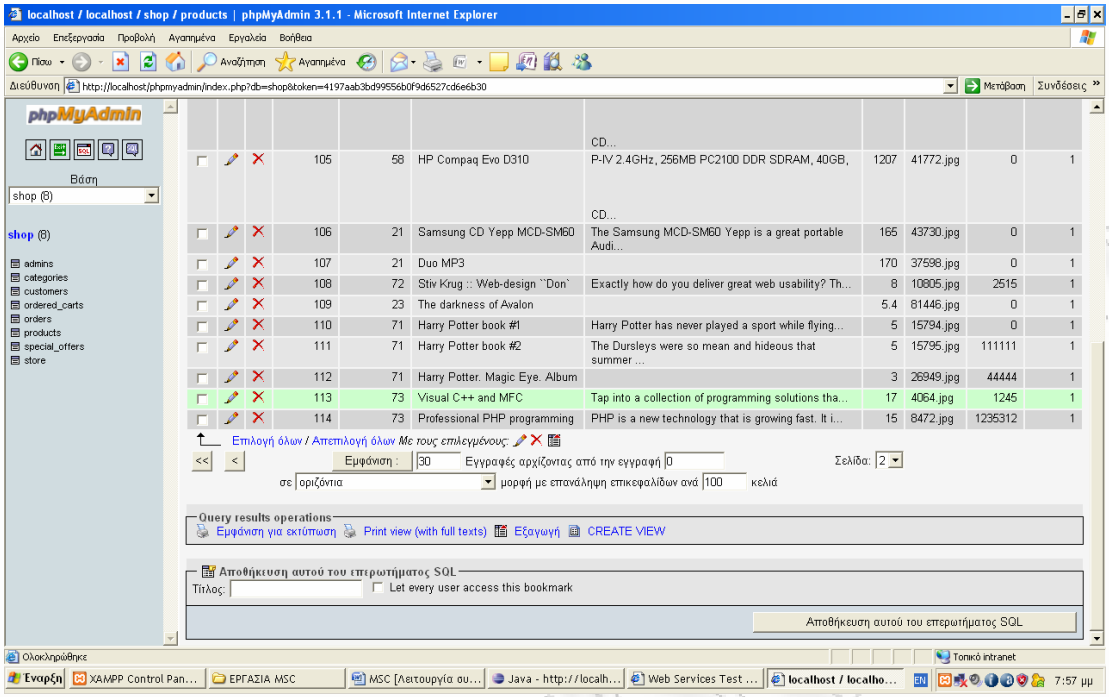
και την Β.Δ. φαίνεται παρακάτω:



Σχήμα 23

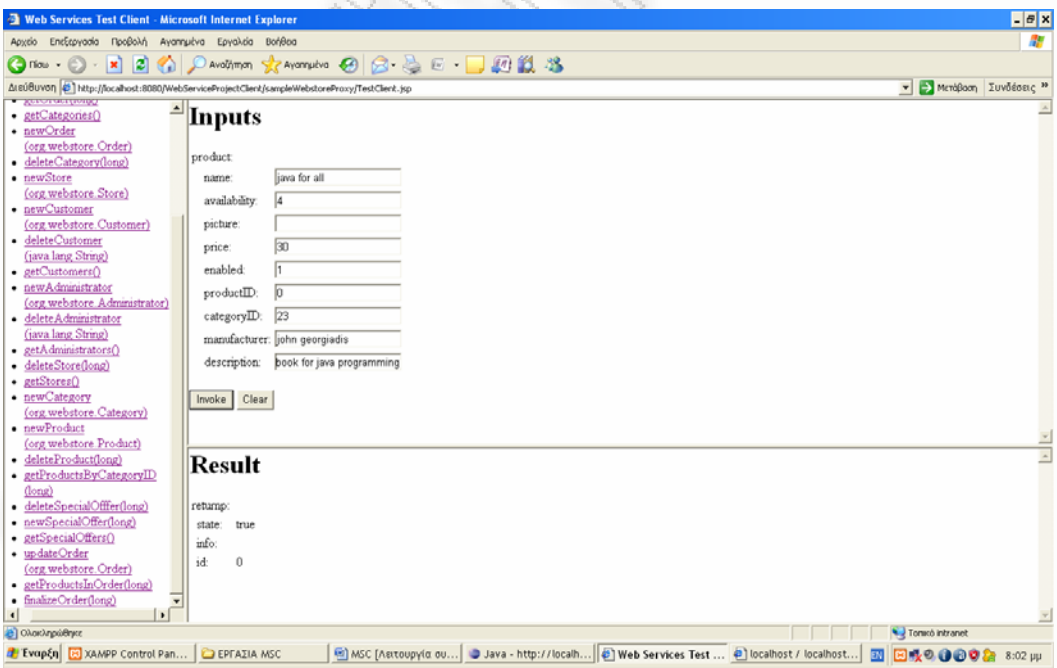
Στιγμιότυπο newProduct

Στο επόμενο στιγμιότυπο φαίνεται η κλήση της μεθόδου newProduct η οποία προσθέτει ένα καινούριο προϊόν στη βάση δεδομένων. Η Β.Δ. πριν :



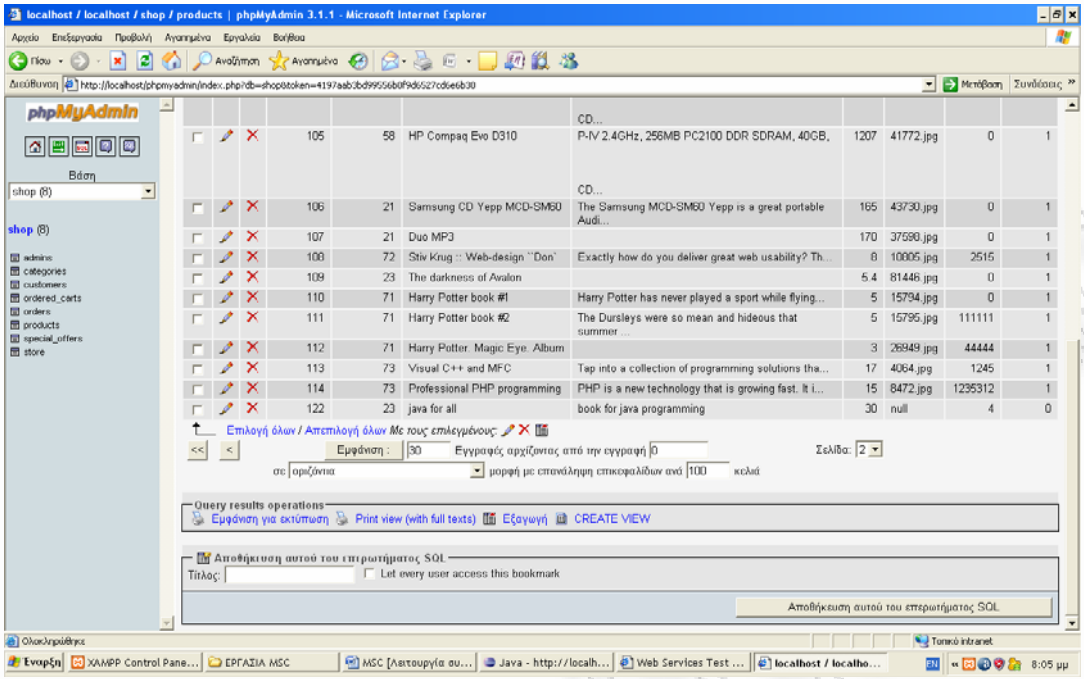
Σχήμα 24

Εκτελώντας την κλήση της μεθόδου newProduct:προσθέτουμε ένα νέο βιβλίο στην κατηγορία 23 η οποία είναι η κατηγορία βιβλίων (books) και βλέπουμε ότι πήραμε και αυτόματα ένα νέο ID για το προϊόν (product)



Σχήμα 25

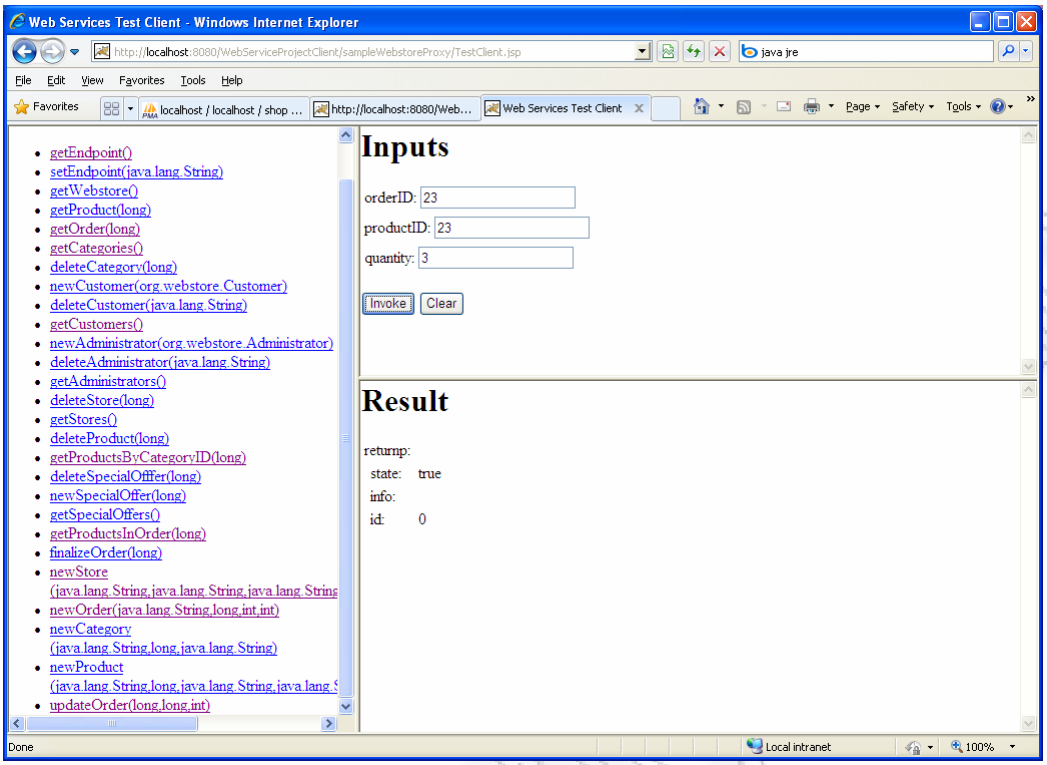
Η Β.Δ. μετά:



Σχήμα 26

Στιγμιότυπο updateOrder(long orderID, long productID, int quantity)

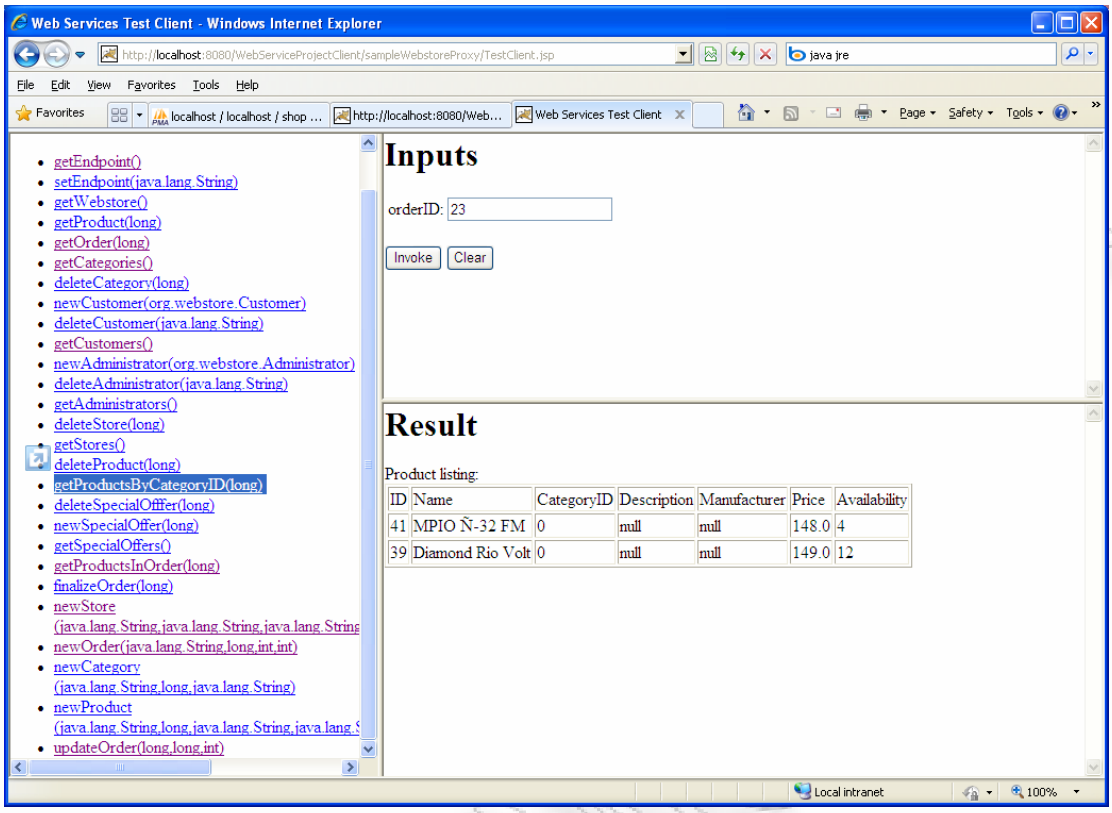
Εκτελώντας την κλήση της μεθόδου updateOrder(long orderID, long productID, int quantity):προσθέτουμε ένα νέο προϊόν στην παραγγελία που έχουμε κάνει με το συγκεκριμένο ID της.



Σχήμα 27

Στιγμιότυπο getProductsInOrder(long orderID)

Τα προιόντα στη παραγελλία φαίνονται καλώντας την getProductsByCategoryID(long).



Σχήμα 28

Στιγμιότυπο finalizeOrder(long orderID)

Η μέθοδος αυτή χρησιμοποιείται για την παραγωγή ενός αρχείου XML που περιγράφει τα περιεχόμενα μιας παραγγελίας. Το έγγραφο XML που παράγεται έχει την παρακάτω μορφή:

```
<?xml version="1.0" encoding="ISO-8859-7"?>
<!DOCTYPE ORDER SYSTEM "order.dtd">
<ORDER ID="23">
    <Product ID="41" NAME="MPIO  23 FM" QUANTITY="4"/>
    <Product ID="39" NAME="Diamond Rio Volt" QUANTITY="12"/>
    <Store ID="0"/>
    <Customer ID="testuser1"/>
</ORDER>
```

Σχήμα 29

6. Επίλογος

Το ηλεκτρονικό εμπόριο έχει αναπτυχθεί ραγδαία τα τελευταία χρόνια. Στα πλαίσια της εργασίας αυτής εξετάζεται η ανάπτυξη και οι απαιτήσεις του Ηλεκτρονικού Εμπορίου. Γίνεται ιδιαίτερη αναφορά και επικέντρωση σε ένα ιδιαίτερο κομμάτι του Ηλεκτρονικού Εμπορίου, τις συναλλαγές μεταξύ διαφορετικών τμημάτων μιας επιχείρησης και τις συναλλαγές μεταξύ επιχειρήσεων (Business-to-Business – B2B).

Στόχος της υποτιθέμενης εταιρίας ηταν η παροχή προϊόντων στο Διαδίκτυο και η αλλαγή λειτουργίας και τρόπου συναλλαγών τόσο στις αγοραπωλησίες με τους πελάτες όσο και με άλλες εταιρίες. Πιο συγκεκριμένα, ζητείται να αλλάξει ο τρόπος λειτουργίας της επιχείρησης, με την αυτοματοποίηση συναλλαγών ανάμεσα στην εταιρία και τους πελάτες, αλλά και ανάμεσα σε εχωριστά τμήματα της εταιρίας και κατά επέκταση ανάμεσα στην εταιρία και άλλες εταιρίες που σχετίζονται στην αλυσίδα παραγωγής προϊόντων .

Παράλληλα, ζητείται η αυτοματοποίηση σε μεγάλο βαθμό της ροής εργασιών μέσα στην ίδια την εταιρία. Παραγγελίες μέσω του Διαδικτύου είτε με τη φυσική παρουσία σε κάποιο υποκατάστημα της εταιρίας ζητείται να ελέγχονται και να μεταφέρονται αυτόματα στο κεντρικό κατάστημα, όπου διατηρείται η βάση δεδομένων και γίνεται έλεγχος για την διαθεσιμότητα του κάθε προϊόντος. Τέλος ζητείται το άνοιγμα των αγορών και η δυνατότητα συνεργασίας με εταιρίες και πελάτες μέσω του Διαδικτύου που θα ήταν αδύνατο να γίνουν διαφορετικά λόγω γεωγραφικής απόστασης.

Στα πλαίσια της εργασίας αυτής, με βάση τις παραπάνω προδιαγραφές σχεδιάστηκε και αναπτύχθηκε το backend τμήμα ενός ηλεκτρονικού καταστήματος βασισμένο σε Web Services. Ο σχεδιασμός και ανάπτυξη του υποσυστήματος αυτού έχει στόχο να επιδείξει και να επαληθεύσει τα πλεονεκτήματα της χρήσης Υπηρεσιών Ιστού. Πιο συγκεκριμένα, αναπτύχθηκε ένα λογισμικό middleware το οποίο λειτουργεί

με την αρχή του πελάτη-εξυπηρετητή, όπου τα διάφορα καταστήματα (front-end) μιλούν με τη χρήση του Web Service Client στο κεντρικό κατάστημα όπου βρίσκεται το Web Service και η βάση δεδομένων (back-end). Το Web Service αποθηκεύει και ανακτά πληροφορίες από τη βάση δεδομένων σύμφωνα με τις κλήσεις από τους Web Service Clients.

Βιβλιογραφικές πηγές

[1] Aldrich.M The Inventor's Story Aldrich Archive, University of Brighton http://www.aldricharchive.com/inventors_story.html

[2] http://en.wikipedia.org/wiki/Tim_Berners-Lee

[3] Kevin Kelly: We Are the Web,Wired magazine, Issue 13.08, August 2005

[4] M. Bell, Service-Oriented Modeling: Service Analysis, Design, and Architecture, John Wiley & Sons, 2008.

[5] T. Erl, Service-oriented Architecture: Concepts, Technology, and Design. Upper Saddle River: Prentice Hall, 2005.

[6] Kantor, Michael; James H. Burrows (1996-04-29). "ELECTRONIC DATA INTERCHANGE (EDI)". National Institute of Standards and Technology. http://www.itl.nist.gov/fipspubs/fip161-2.htm

[7] "The official CORBA standard from the OMG group" http://www.itl.nist.gov/fipspubs/fip161-2.htm

[8] JAVA RMI Tutorial: http://download.oracle.com/docs/cd/E17409_01/javase/tutorial/rmi/index.html

[9] David Chappell, Tyler Jewell, Java Web Services, O'Reilly, First Edition March 2002

[10] Ethan Cerami, Web Services Essentials, O'Reilly, First Edition February 2002

[11] M. Paolucci, N. Srinivasan, K. Sycara, “OWL Ontology of Web Service Architecture Concepts”, http://www.w3.org/2004/02/wsa/.

[12] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. Nielsen, “W3C Recommendation SOAP Version 1.2 Part 2: Adjuncts”, June 2003, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.

[13] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler., “W3C Recommendation Extensible Markup Language (XML) 1.0 (Fourth Edition)”, August 2006, http://www.w3.org/TR/REC-xml/.

[14] IBM, “IBM Service Oriented Architecture (SOA)”, October 2008, http://www-01.ibm.com/software/solutions/soa/.

[15] Wikipedia, “Web Service”, October 2008, http://en.wikipedia.org/wiki/Web_Services.

[16] Wikipedia, “Service Oriented Modeling”, October 2008, http://en.wikipedia.org/wiki/Service-oriented_modeling.

[17] A. Yee, Testing Web Services: What You Need to Know, November 2003, http://www.ebizq.net/topics/web_services/features/3307.html.

ΠΑΡΑΡΤΗΜΑ – ΚΩΔΙΚΑΣ

Κλάση: Administrator

```
package org.webstore;

public class Administrator {

		String username;
		String name;
		String surname;
		String password;
		boolean superadmin;


		public Administrator(){

		}

		public  Administrator(String  username,  String  name,
String surname,
				String password, boolean superadmin) {
			super();
			this.username = username;
			this.name = name;
			this.surname = surname;
			this.password = password;
			this.superadmin = superadmin;
		}

		public String getUsername() {
			return username;
		}
		public void setUsername(String username) {
			this.username = username;
		}
		public String getName() {
			return name;
		}
		public void setName(String name) {
			this.name = name;
		}
		public String getSurname() {
			return surname;
		}
		public void setSurname(String surname) {
```

```
                        this.surname = surname;
                }
                public String getPassword() {
                        return password;
                }
                public void setPassword(String password) {
                        this.password = password;
                }
                public boolean isSuperadmin() {
                        return superadmin;
                }
                public void setSuperadmin(boolean superadmin) {
                        this.superadmin = superadmin;
                }



}
```

Κλάση: Category

```
        package org.webstore;

public class Category {
        long categoryID;
        String name;
        long parent;
        String description;
        String picture;

        public Category(){

        }

        public Category(long categoryID, String name, long parent,
                        String description, String picture) {
                super();
                this.categoryID = categoryID;
                this.name = name;
                this.parent = parent;
                this.description = description;
                this.picture = picture;
        }

        public long getCategoryID() {
                return categoryID;
```

```
	}

	public void setCategoryID(long categoryID) {
		this.categoryID = categoryID;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public long getParent() {
		return parent;
	}

	public void setParent(long parent) {
		this.parent = parent;
	}

	public String getDescription() {
		return description;
	}

	public void setDescription(String description) {
		this.description = description;
	}

	public String getPicture() {
		return picture;
	}

	public void setPicture(String picture) {
		this.picture = picture;
	}


}
```

Κλάση: Configuration

```
	package org.webstore;

import java.util.MissingResourceException;
```

```
import java.util.ResourceBundle;

public class Configuration {
	private static final String BUNDLE_NAME = "database"; //$NON-
NLS-1$

	private static final ResourceBundle RESOURCE_BUNDLE =
ResourceBundle
			.getBundle(BUNDLE_NAME);

	private Configuration() {
	}

	public static String getString(String key) {
		try {
			return RESOURCE_BUNDLE.getString(key);
		} catch (MissingResourceException e) {
			return '!' + key + '!';
		}
	}
}
```

Κλάση: Customer

```
package org.webstore;

public class Customer  {
    private java.lang.String username;  // attribute

    private java.lang.String password;  // attribute

    private java.lang.String name;  // attribute

    private java.lang.String AFM;  // attribute

    private java.lang.String email;  // attribute

    private java.lang.String telephone;  // attribute

    private java.lang.String address;  // attribute

    public Customer() {
    }

    public Customer(
           java.lang.String username,
           java.lang.String password,
```

```
            java.lang.String name,
            java.lang.String AFM,
            java.lang.String email,
            java.lang.String telephone,
            java.lang.String address) {
            this.username = username;
            this.password = password;
            this.name = name;
            this.AFM = AFM;
            this.email = email;
            this.telephone = telephone;
            this.address = address;
    }


    /**
     * Gets the username value for this Customer.
     *
     * @return username
     */
    public java.lang.String getUsername() {
        return username;
    }


    /**
     * Sets the username value for this Customer.
     *
     * @param username
     */
    public void setUsername(java.lang.String username) {
        this.username = username;
    }


    /**
     * Gets the password value for this Customer.
     *
     * @return password
     */
    public java.lang.String getPassword() {
        return password;
    }


    /**
     * Sets the password value for this Customer.
     *
     * @param password
     */
    public void setPassword(java.lang.String password) {
```

```
        this.password = password;
    }


    /**
     * Gets the name value for this Customer.
     *
     * @return name
     */
    public java.lang.String getName() {
        return name;
    }


    /**
     * Sets the name value for this Customer.
     *
     * @param name
     */
    public void setName(java.lang.String name) {
        this.name = name;
    }


    /**
     * Gets the surname value for this Customer.
     *
     * @return surname
     */
    public java.lang.String getAFM() {
        return AFM;
    }


    /**
     * Sets the surname value for this Customer.
     *
     * @param surname
     */
    public void setAFM(java.lang.String AFM) {
        this.AFM = AFM;
    }


    /**
     * Gets the email value for this Customer.
     *
     * @return email
     */
    public java.lang.String getEmail() {
        return email;
```

```
    }


    /**
     * Sets the email value for this Customer.
     *
     * @param email
     */
    public void setEmail(java.lang.String email) {
        this.email = email;
    }


    /**
     * Gets the telephone value for this Customer.
     *
     * @return telephone
     */
    public java.lang.String getTelephone() {
        return telephone;
    }


    /**
     * Sets the telephone value for this Customer.
     *
     * @param telephone
     */
    public void setTelephone(java.lang.String telephone) {
        this.telephone = telephone;
    }

    /**
     * Gets the address value for this Customer.
     *
     * @return address
     */
    public java.lang.String getAddress() {
        return address;
    }


    /**
     * Sets the address value for this Customer.
     *
     * @param address
     */
    public void setAddress(java.lang.String address) {
        this.address = address;
    }
```

```
}
```

Κλάση: Database

```
package org.webstore;

import    java.sql.Connection;import    java.sql.DriverManager;import
java.sql.ResultSet;import                 java.sql.SQLException;import
java.sql.Statement;import java.util.Vector;
public class Database {

        Connection con;        String
user=Configuration.getString("Database.user"); //$NON-NLS-1$ String
password=Configuration.getString("Database.password");  //$NON-NLS-1$
        String  db=Configuration.getString("Database.name");  //$NON-
NLS-1$
        public Database(){

              //Define URL of database server for
              // database named shop on  localhost
              // with the default port number 3306.
              String   url   ="jdbc:mysql://localhost:3306/"   +   db;
//$NON-NLS-1$

              try {
                try {

        Class.forName("com.mysql.jdbc.Driver"); //$NON-NLS-1$
                        } catch (ClassNotFoundException e) {
                                e.printStackTrace();
                        }
                        con   =   DriverManager.getConnection(url,user,
password);
                } catch (SQLException e) {
                        e.printStackTrace();
                }

        }
        /**       * Creates a new Customer.       *        * @return
Response     */public    Response    newCustomer(Customer    customer){
        Response resp = new Response(false,""); //$NON-NLS-1$
        Statement stmt;        try {                  stmt          =
con.createStatement();                String insertString = "INSERT
INTO customers (cust_username ,cust_password ,cust_name ,cust_email
,cust_address   ,cust_phone   ,   cust_afm,   cust_date)   VALUES('"
+customer.getUsername()  + "','" +  customer.getPassword() + "','" +
customer.getName()   +   "','"   +   customer.getEmail()+   "','"   +
```

```
customer.getAddress()    + "','" + customer.getTelephone() + "','"+
customer.getAFM()  +"'  ,  CURDATE()  )"; //$NON-NLS-1$  //$NON-NLS-2$
//$NON-NLS-3$ //$NON-NLS-4$ //$NON-NLS-5$ //$NON-NLS-6$ //$NON-NLS-7$
//$NON-NLS-8$                  stmt.executeUpdate(insertString);
{                      resp.setInfo(e.getMessage());          }
username){             Response   resp   =   new   Response(false,"");
//$NON-NLS-1$          Statement stmt;        try {
       stmt = con.createStatement();                  String
insertString  =  "Delete  from  customers  where  cust_username='"  +
username  + "'"; //$NON-NLS-1$ //$NON-NLS-2$
       stmt.executeUpdate(insertString);                    //if
statement executes without exception, set status of resp to true
Returns the   Customer list.      *       * @return Vector<Customer>
*/      public Vector <Customer> getCustomers(){
       Statement stmt;        Vector<Customer>      c      =      new
Vector<Customer>();           Customer customer;             try{
the result                    // in an object of type ResultSet
Customer();
       customer.setUsername(rs.getString("cust_username"));   //$NON-
NLS-1$
       customer.setName(rs.getString("cust_name")); //$NON-NLS-1$
1$
       customer.setTelephone(rs.getString("cust_phone"));     //$NON-
NLS-1$
       customer.setEmail(rs.getString("cust_email")); //$NON-NLS-1$
c;             }                                /**     * Creates a new
Administrator.     *      * @return Response     */  public  Response
newAdministrator(Administrator admin){        Response   resp   =   new
Response(false,""); //$NON-NLS-1$             Statement stmt;
       try {                  stmt = con.createStatement();
       int superadmin;                if(admin.isSuperadmin())
,name ,surname , superadmin) VALUES('" +admin.getUsername()   + "','"
+     admin.getPassword()   +   "','"   +   admin.getName()   +   "','"   +
admin.getSurname()+  "','"  +   superadmin      +"'  )"; //$NON-NLS-1$
//$NON-NLS-2$ //$NON-NLS-3$ //$NON-NLS-4$ //$NON-NLS-5$ //$NON-NLS-6$
(SQLException e) {                    resp.setInfo(e.getMessage());
       }                              return resp;           }
       /**      * Deletes an Administrator.      *       * @return
Response     */public Response deleteAdministrator(String username){
       Response resp = new Response(false,""); //$NON-NLS-1$
       Statement stmt;        try {                   stmt          =
con.createStatement();                String  insertString  =  "Delete
from  admins  where  username='"  +  username    +  "'";  //$NON-NLS-1$
//$NON-NLS-2$                  stmt.executeUpdate(insertString);
{                      resp.setInfo(e.getMessage());          }
getAdministrators(){          Statement stmt;
       Vector<Administrator>  admins  =  new  Vector<Administrator>();
       Administrator admin;          try{                    stmt   =
con.createStatement(
ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);                  //Query  the  database,
storing the result                    //   in   an   object   of   type
```

```
ResultSet              ResultSet  rs  =  stmt.executeQuery("SELECT  *
from admins"); //$NON-NLS-1$                  while(rs.next()){
NLS-1$
       admin.setSurname(rs.getString("surname")); //$NON-NLS-1$
1$                                  admin.setSuperadmin(true);
catch (SQLException e) {                      e.printStackTrace();
       }              return admins;         }
       /**     * Creates a new Store.     *      * @return Response
*/     public  Response  newStore(String  name,  String  address,String
telephone){            Response   resp   =   new   Response(false,"");
//$NON-NLS-1$          Statement stmt;        long storeID=0;
       try {                  stmt = con.createStatement();
//$NON-NLS-3$ //$NON-NLS-4$
       stmt.executeUpdate(insertString);
       ResultSet rs = stmt.getGeneratedKeys();
       if(rs.next()){                         storeID               =
rs.getLong(1);                }                    //if   statement
executes without exception, set status of resp to true
       resp.setState(true);                   resp.setId(storeID);
       } catch (SQLException e) {
       resp.setInfo(e.getMessage());          }
       return resp;           }              /**        * Deletes a
Store.     *      * @return Response     */   public          Response
deleteStore(long storeID){            Response     resp     =     new
Response(false,""); //$NON-NLS-1$             Statement stmt;
       try {                  stmt = con.createStatement();
       String insertString = "Delete from store where storeID='" +
storeID  + "'"; //$NON-NLS-1$ //$NON-NLS-2$
       stmt.executeUpdate(insertString);                    //if
statement executes without exception, set status of resp to true
Returns the   Stores list.     *      * @return Vector<Store>     */
       public Vector <Store> getStores(){            Statement  stmt;
       Vector<Store> stores = new Vector<Store>();          Store
store;         try{                   stmt = con.createStatement(
ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);                  //Query  the  database,
storing the result                    //   in   an   object   of   type
ResultSet                      ResultSet             rs             =
stmt.executeQuery("SELECT * from store"); //$NON-NLS-1$
       while(rs.next()){                              store    =    new
Store();
       store.setStoreID(rs.getLong("storeID")); //$NON-NLS-1$
1$
       store.setTelephone(rs.getString("telephone")); //$NON-NLS-1$
stores;        }                      /**          * Creates a new
Category.     *      * @return Response     */       public Response
newCategory(Category category){               Response   resp   =   new
Response(false,""); //$NON-NLS-1$             Statement stmt;
       long categoryID=0;             try {                  stmt    =
con.createStatement();                                       String
insertString = "INSERT INTO categories (name , parent, description ,
picture)    VALUES('"    +    category.getName()    +    "','"    +
```

```
category.getParent() + "','" +  category.getDescription()    +"','" +
category.getPicture()  +  "'  )"; //$NON-NLS-1$  //$NON-NLS-2$  //$NON-
NLS-3$ //$NON-NLS-4$ //$NON-NLS-5$
        stmt.executeUpdate(insertString);                    //if
statement executes without exception, set status of resp to true
rs.getLong(1);                }
        resp.setState(true);                  resp.setId(categoryID);
a Category.     *      * @return Response     */     public  Response
deleteCategory(long categoryID){              Response  resp  =  new
Response(false,""); //$NON-NLS-1$            Statement stmt;
        try {                 stmt = con.createStatement();
        String  insertString  =  "Delete  from  categories  where
categoryID='" + categoryID  + "'"; //$NON-NLS-1$ //$NON-NLS-2$
(SQLException e) {                    resp.setInfo(e.getMessage());
        }                             return resp;          }
        /**     * Returns the   Categories list.     *      * @return
Vector<Store>     */   public   Vector   <Category>   getCategories(){
        Statement stmt;        Vector<Category>   categories   =   new
Vector<Category>();           Category category;            try{
the result                    // in an object of type ResultSet
rs.getString("name"),  rs.getLong("parent")  ,  //$NON-NLS-1$  //$NON-
NLS-2$ //$NON-NLS-3$
        rs.getString("description")   ,   rs.getString("picture")   );
//$NON-NLS-1$ //$NON-NLS-2$
        categories.add(category);
Returns the   Product list in the specified category.      *       *
@return Vector<Product>     */ public         Vector         <Product>
getProductsByCategoryID(long categoryID){
        Product product = null;               Vector<Product> v = new
Vector<Product>();            Statement stmt;        try{
        stmt = con.createStatement(
ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);                  //Query  the  database,
storing the result                    //   in   an   object   of   type
ResultSet                     ResultSet             rs             =
stmt.executeQuery("SELECT  *  FROM  `categories`  ,  `products`   WHERE
`categories`.`categoryID`      =      `products`.`categoryID`      AND
`categories`.`categoryID` ="+ categoryID); //$NON-NLS-1$
        while(rs.next()){                             product   =   new
Product();
        product.setAvailability(rs.getInt("productID"));  //$NON-NLS-
1$                            product.setName(rs.getString("name"));
//$NON-NLS-1$
        product.setDescription(rs.getString("description"));  //$NON-
NLS-1$
        product.setAvailability(rs.getInt("in_stock")); //$NON-NLS-1$
1$
        product.setCategoryID(rs.getLong("categoryID"));  //$NON-NLS-
1$                            v.add(product);               }
Response     */
        public Product getProduct(long productId){
```

```
                Product product = null;

                 Statement stmt;
                try {
                        stmt = con.createStatement();
                        //Get another statement object initialized
                       // as shown.
                        stmt = con.createStatement(
                                ResultSet.TYPE_SCROLL_INSENSITIVE,
                                      ResultSet.CONCUR_READ_ONLY);

                     //Query the database, storing the result
                     // in an object of type ResultSet
                        ResultSet  rs  =  stmt.executeQuery("SELECT  *
from products where productID="+productId); //$NON-NLS-1$
                        while(rs.next()){
                                product = new Product();
                                product.setProductID(productId);
                                product.setName(rs.getString("name"));
//$NON-NLS-1$
        product.setDescription(rs.getString("description"));   //$NON-
NLS-1$

        product.setAvailability(rs.getInt("in_stock")); //$NON-NLS-1$

        product.setPicture(rs.getString("picture")); //$NON-NLS-1$

        product.setCategoryID(rs.getLong("categoryID"));   //$NON-NLS-
1$
                        }

                } catch (SQLException e) {
                        e.printStackTrace();
                }

                return product;
        }

                /**     * Creates a new Product.     *      * @return
Response     */public Response newProduct(Product product){
        Response resp = new Response(false,""); //$NON-NLS-1$
        Statement stmt;         long productID=0;              try {
enabled;                        if(product.isEnabled())
(categoryID,name,description,Price,picture,in_stock,enabled    )  "  +
//$NON-NLS-1$                                     "VALUES('"
+product.getCategoryID()   +  "','"  +   product.getName()  +  "','"  +
product.getDescription() + "','"  //$NON-NLS-1$ //$NON-NLS-2$ //$NON-
NLS-3$ //$NON-NLS-4$                                        +
product.getPrice()  +  "','"+    product.getPicture()    +  "','"    +
product.getAvailability()    + "','" + enabled + "' )"; //$NON-NLS-1$
//$NON-NLS-2$ //$NON-NLS-3$ //$NON-NLS-4$
```

```
        stmt.executeUpdate(insertString);                    //get
productID                    ResultSet            rs            =
stmt.getGeneratedKeys();                    if(rs.next()){
true                  resp.setState(true);
        resp.setId(productID);        } catch (SQLException e) {
Response deleteProduct(long productID){             Response resp =
new Response(false,""); //$NON-NLS-1$        Statement stmt;
        try {                 stmt = con.createStatement();
        String   insertString   =   "Delete   from   products   where
productID='" + productID  + "'"; //$NON-NLS-1$ //$NON-NLS-2$
        stmt.executeUpdate(insertString);                    //if
statement executes without exception, set status of resp to true
Creates a new special offer using an existing product.     *      *
@return Response     */       public  Response  newSpecialOffer(long
productID){           Response   resp   =   new   Response(false,"");
//$NON-NLS-1$          Statement stmt;        try {
        stmt = con.createStatement();
        String insertString = "INSERT INTO special_offers( productID
)  " + //$NON-NLS-1$                                  "VALUES('"
+productID + "' )"; //$NON-NLS-1$ //$NON-NLS-2$
        stmt.executeUpdate(insertString);                    //if
statement executes without exception, set status of resp to true
Deletes a special offer.     *      * @return Response     */public
Response deleteSpecialOffer(long productID){         Response resp =
new Response(false,""); //$NON-NLS-1$        Statement stmt;
        try {                 stmt = con.createStatement();
statement executes without exception, set status of resp to true
Returns the  Product list tagged as special offers.      *       *
@return Vector<Product>     */ public         Vector         <Product>
getSpecial_offers(){                          Product product = null;
ResultSet.CONCUR_READ_ONLY);                  //Query  the  database,
storing the result                   //   in   an   object   of   type
ResultSet                    ResultSet            rs            =
stmt.executeQuery("SELECT   *   FROM   special_offers,products   where
products.productID=special_offers.productID"); //$NON-NLS-1$
        while(rs.next()){                             product   =   new
Product();
        product.setProductID(rs.getInt("productID")); //$NON-NLS-1$
1$
        product.setAvailability(rs.getInt("in_stock")); //$NON-NLS-1$
1$
        product.setCategoryID(rs.getLong("categoryID"));   //$NON-NLS-
1$                           v.add(product);               }
orderID){             Statement stmt;        Order       order=null;
        try{                  stmt = con.createStatement(
ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);                  //Query  the  database,
storing the result                   //   in   an   object   of   type
ResultSet                    ResultSet            rs            =
stmt.executeQuery("SELECT  *  from  orders  where  orderID="+orderID);
//$NON-NLS-1$                 while(rs.next()){
        order = new Order();
```

```
        order.setOrderid(rs.getLong("orderID")); //$NON-NLS-1$
{                       e.printStackTrace();            }
        return order;           }               /*      *   Created   a
new order       */      public Response newOrder(Order order){
        Response resp = new Response(false,""); //$NON-NLS-1$
        Statement stmt;         long orderID=0;         try {
        stmt = con.createStatement();                   //add new order
in table order                  String  insertString  =  "INSERT  INTO
`shop`.`orders`  (`cust_username`  ,`storeid`  ,`order_time`  )VALUES
('"+ order.getUsername()  +"', '"+ order.getStoreId() +"', NOW( ) )";
//$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$
        stmt.executeUpdate(insertString);
rs.getLong(1);                  }
,`productID`)VALUES  ('"+  orderID  +"',  '"+  order.getQuantity()  +"',
'"+  order.getProductId()  +"')";  //$NON-NLS-1$  //$NON-NLS-2$  //$NON-
NLS-3$ //$NON-NLS-4$                              stmt                  =
con.createStatement();
        stmt.executeUpdate(insertString);
        //if  statement(s)  execute  without  exception,  set  status  of
resp to true                            resp.setState(true);
{                       resp.setInfo(e.getMessage());           }
//$NON-NLS-1$           Statement stmt;         try {
        stmt = con.createStatement();                   //add         new
product in specific order                       String  insertString  =
"INSERT    INTO    `shop`.`ordered_carts`    (`orderID`    ,`Quantity`
,`productID`)  VALUES  ('"+  orderID  +"',  '"+  quantity  +"',  '"+
productID  +"')";  //$NON-NLS-1$  //$NON-NLS-2$  //$NON-NLS-3$  //$NON-
NLS-4$                  stmt.executeUpdate(insertString);
        //if  statement(s)  execute  without  exception,  set  status  of
resp to true                    resp.setState(true);            }  catch
(SQLException e) {                      resp.setInfo(e.getMessage());
        }                               return resp;            }

        public Vector <Product> getProductsInOrder(long orderID){

                Product product = null;
                Vector<Product> v = new Vector<Product>();
                Statement stmt;
                try{
                        stmt = con.createStatement(
                        ResultSet.TYPE_SCROLL_INSENSITIVE,
                              ResultSet.CONCUR_READ_ONLY);

                        //Query the database, storing the result
                        // in an object of type ResultSet
                        ResultSet   rs   =   stmt.executeQuery("SELECT
`products`.`productid`,   `products`.`name`,      `products`.`Price`,
`Quantity`  FROM  `ordered_carts`  ,  `products`  ,  `orders`  WHERE
`ordered_carts`.`productID`     =     `products`.`productID`     AND
`ordered_carts`.`orderID` = `orders`.`orderID` AND `orders`.`orderID`
="+orderID); //$NON-NLS-1$
                        while(rs.next()){
```

```
                                product = new Product();

        product.setProductID(rs.getInt("productid")); //$NON-NLS-1$
                                product.setName(rs.getString("name"));
//$NON-NLS-1$

        product.setAvailability(rs.getInt("Quantity")); //$NON-NLS-1$

        product.setPrice(rs.getFloat("Price")); //$NON-NLS-1$
                                v.add(product);
                        }

                } catch (SQLException e) {
                        e.printStackTrace();
                }

                return v;
        }

        public void close(){
                try {
                        con.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }



}
```

Κλάση: Order

```
package org.webstore;

public class Order  {
    private java.lang.String username;  // attribute

    private long productId;  // attribute

    private long storeId;  // attribute

    private int quantity;  // attribute
```

```
    private long orderid;  // attribute

    public Order() {
    }

    public Order(
           java.lang.String username,
           long productId,
           int storeId,
           int quantity,
           long orderid) {
           this.username = username;
           this.productId = productId;
           this.storeId = storeId;
           this.quantity = quantity;
           this.orderid = orderid;
    }


    /**
     * Gets the username value for this Order.
     *
     * @return username
     */
    public java.lang.String getUsername() {
        return username;
    }


    /**
     * Sets the username value for this Order.
     *
     * @param username
     */
    public void setUsername(java.lang.String username) {
        this.username = username;
    }


    /**
     * Gets the productId value for this Order.
     *
     * @return productId
     */
    public long getProductId() {
        return productId;
    }


    /**
```

```
     * Sets the productId value for this Order.
     *
     * @param productId
     */
    public void setProductId(long productId) {
        this.productId = productId;
    }


    /**
     * Gets the storeId value for this Order.
     *
     * @return storeId
     */
    public long getStoreId() {
        return storeId;
    }


    /**
     * Sets the storeId value for this Order.
     *
     * @param storeId
     */
    public void setStoreId(long storeId) {
        this.storeId = storeId;
    }


    /**
     * Gets the quantity value for this Order.
     *
     * @return quantity
     */
    public int getQuantity() {
        return quantity;
    }


    /**
     * Sets the quantity value for this Order.
     *
     * @param quantity
     */
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }


    /**
     * Gets the orderid value for this Order.
```

```
     *
     * @return orderid
     */
    public long getOrderid() {
        return orderid;
    }


    /**
     * Sets the orderid value for this Order.
     *
     * @param orderid
     */
    public void setOrderid(long orderid) {
        this.orderid = orderid;
    }


}
```

Κλάση: Product

```
package org.webstore;

public class Product {
        long productID;
        long categoryID;
        String name;
        String manufacturer;
        String description;
        float price;
        String picture;
        int availability;
        boolean enabled;



        public Product(){

        }


        public Product(long productID, long categoryID, String name,
```

```
                        String manufacturer, String description, float
price,
                        String  picture,  int  availability,  boolean
enabled) {
                super();
                this.productID = productID;
                this.categoryID = categoryID;
                this.name = name;
                this.manufacturer = manufacturer;
                this.description = description;
                this.price = price;
                this.picture = picture;
                this.availability = availability;
                this.enabled = enabled;
        }

        public String getManufacturer() {
                return manufacturer;
        }

        public void setManufacturer(String manufacturer) {
                this.manufacturer = manufacturer;
        }

        public long getProductID() {
                return productID;
        }
        public void setProductID(long productID) {
                this.productID = productID;
        }
        public long getCategoryID() {
                return categoryID;
        }
        public void setCategoryID(long categoryID) {
                this.categoryID = categoryID;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getDescription() {
                return description;
        }
        public void setDescription(String description) {
                this.description = description;
        }
        public float getPrice() {
                return price;
        }
```

```
        public void setPrice(float price) {
                this.price = price;
        }
        public String getPicture() {
                return picture;
        }
        public void setPicture(String picture) {
                this.picture = picture;
        }
        public int getAvailability() {
                return availability;
        }
        public void setAvailability(int availability) {
                this.availability = availability;
        }
        public boolean isEnabled() {
                return enabled;
        }
        public void setEnabled(boolean enabled) {
                this.enabled = enabled;
        }


}
```

Κλάση: Response

```
        package org.webstore;

public class Response {
        String info;    //message in case of failure
        boolean state;  //true if transaction succeeds
        long id;                //  contains  the  id  of  the  newly
created object, if any


        public Response(){

        }

        public Response(boolean state,String info) {
                super();
                this.info = info;
                this.state = state;
```

```
                this.id=0;
        }

        public String getInfo() {
                return info;
        }
        public void setInfo(String info) {
                this.info = info;
        }
        public boolean isState() {
                return state;
        }
        public void setState(boolean state) {
                this.state = state;
        }

        public long getId() {
                return id;
        }

        public void setId(long id) {
                this.id = id;
        }


}
```

Κλάση: Store

```
        package org.webstore;

public class Store {
        long storeID;
        String name;
        String address;
        String telephone;

        public Store(){

        }

        public  Store(long  storeID,  String  name,  String  address,
String telephone) {
                super();
                this.storeID = storeID;
                this.name = name;
                this.address = address;
```

```
                this.telephone = telephone;
        }

        public long getStoreID() {
                return storeID;
        }

        public void setStoreID(long storeID) {
                this.storeID = storeID;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public String getAddress() {
                return address;
        }

        public void setAddress(String address) {
                this.address = address;
        }

        public String getTelephone() {
                return telephone;
        }

        public void setTelephone(String telephone) {
                this.telephone = telephone;
        }



}
```

Κλάση: XMLOrder

```
package org.webstore;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
import java.util.Vector;

import org.xml.sax.ContentHandler;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.AttributesImpl;

import com.sun.org.apache.xml.internal.serialize.OutputFormat;
import com.sun.org.apache.xml.internal.serialize.XMLSerializer;

public class XMLOrder {

	Order order;
	String filename;

	public XMLOrder(Order order){
		this.order=order;
		filename =  "test.xml";
	}


	public Response serialize(){

		boolean status=false;
		FileOutputStream fos;
		String message="";
		Response resp= new Response(status,message);
		try {
			fos = new FileOutputStream(filename);
			// set encoding

	com.sun.org.apache.xml.internal.serialize.OutputFormat  of  =
new OutputFormat("XML","ISO-8859-7",true);
			of.setIndent(1);
			of.setIndenting(true);
			of.setDoctype(null,"order.dtd");
			XMLSerializer      serializer      =      new
XMLSerializer(fos,of);
			// SAX2.0 ContentHandler.
			ContentHandler              hd              =
serializer.asContentHandler();
			hd.startDocument();

			AttributesImpl atts = new AttributesImpl();
			atts.addAttribute("","","ID","CDATA",new
Long(order.getOrderid()).toString());
			// Order tag.
			hd.startElement("","","ORDER",atts);

			Database d = new Database();
			Vector<Product>              v              =
d.getProductsInOrder(order.getOrderid());
```

```
                        d.close();
                        for(int i=0;i<v.size();i++){
                                // add product
                                atts.clear();

        atts.addAttribute("","","ID","CDATA",new
Long(v.elementAt(i).getProductID()).toString());

        atts.addAttribute("","","NAME","CDATA",
v.elementAt(i).getName() );

        atts.addAttribute("","","QUANTITY","CDATA",new
Integer(v.elementAt(i).getAvailability()).toString() );
                                hd.startElement("","","Product",atts);
                                hd.endElement("","","Product");
                        }
                        //add store
                        atts.clear();
                        atts.addAttribute("","","ID","CDATA",new
Long(order.getStoreId()).toString());
                        hd.startElement("","","Store",atts);
                        hd.endElement("","","Store");

                        //add customer
                        atts.clear();

        atts.addAttribute("","","ID","CDATA",order.getUsername());
                        hd.startElement("","","Customer",atts);
                        hd.endElement("","","Customer");

                        hd.endElement("","","ORDER");
                        hd.endDocument();
                        fos.close();
                        status=true;

                } catch (FileNotFoundException e) {
                        message = message + "\n" +  e.getMessage();
                } catch (IOException e) {
                        message = message + "\n" +  e.getMessage();
                } catch (SAXException e) {
                        message = message + "\n" +  e.getMessage();
                }

                resp.setInfo(message);
                resp.setState(status);
                return resp;
        }
}
```

Κλάση: Webstore

```
package org.webstore;

import java.util.Vector;

public class Webstore {

	Database db;

	public Webstore(){
		db = new Database();
	}

	public Response newCustomer(Customer customer){
		Response resp = db.newCustomer(customer);
		db.close();
		return resp;
	}

	public Response deleteCustomer(String username){
		Response resp = db.deleteCustomer(username);
		db.close();
		return resp;
	}

	public Customer[] getCustomers(){
		Vector<Customer> customerVector = db.getCustomers();
		Customer customers[] = new
Customer[customerVector.size()];
		for(int i=0;i<customerVector.size();i++){
			customers[i] = customerVector.elementAt(i);
		}
		db.close();
		return customers;
	}
```

```
        public Response newAdministrator(Administrator admin){
                Response resp = db.newAdministrator(admin);
                db.close();
                return resp;
        }

        public Response deleteAdministrator(String username){
                Response resp = db.deleteAdministrator(username);
                db.close();
                return resp;
        }

        public Administrator[] getAdministrators(){
                Vector<Administrator>          adminVector          =
db.getAdministrators();
                Administrator          admins[]          =          new
Administrator[adminVector.size()];
                for(int i=0;i<adminVector.size();i++){
                        admins[i] = adminVector.elementAt(i);
                }
                db.close();
                return admins;
        }


        public  Response  newStore(String  name,String  address,String
telephone){
                Response resp = db.newStore(name,address,telephone);
                db.close();
                return resp;
        }

        public Response deleteStore(long storeID){
                Response resp = db.deleteStore(storeID);
                db.close();
                return resp;
        }

        public Store[] getStores(){
                Vector<Store> storeVector = db.getStores();
                Store stores[] = new  Store[storeVector.size()];
                for(int i=0;i<storeVector.size();i++){
                        stores[i] = storeVector.elementAt(i);
                }
                db.close();
                return stores;
        }
```

```
	public Response newCategory(String name, long parent, String
description){
		Category category = new Category(0, name, parent,
description, "");
		Response resp = db.newCategory(category);
		db.close();
		return resp;
	}

	public Response deleteCategory(long categoryID){
		Response resp = db.deleteCategory(categoryID);
		db.close();
		return resp;
	}

	public Category[] getCategories(){
		Vector<Category> categoryVector = db.getCategories();
		Category categories[] = new
Category[categoryVector.size()];
		for(int i=0;i<categoryVector.size();i++){
			categories[i] = categoryVector.elementAt(i);
		}
		db.close();
		return categories;
	}

	public Response newProduct(String name, long categoryID,
String manufacturer, String description, float price,
			String picture, int availability, boolean
enabled)
	{
		Product product = new Product(0, categoryID,
name,manufacturer, description, price,picture, availability,
enabled);
		Response resp = db.newProduct(product);
		db.close();
		return resp;
	}

	public Response deleteProduct(long productID){
		Response resp = db.deleteProduct(productID);
		db.close();
		return resp;
	}

	public Product getProduct(long productId){
		Product p = db.getProduct(productId);
		return p;
	}

	public Product[] getProductsByCategoryID(long categoryID){
```

```
                Vector<Product>            productVector            =
db.getProductsByCategoryID(categoryID);
                Product          products[]          =          new
Product[productVector.size()];
                for(int i=0;i<productVector.size();i++){
                        products[i] = productVector.elementAt(i);
                }
                db.close();
                return products;
        }

        public Response deleteSpecialOfffer(long productID){
                Response resp = db.deleteSpecialOffer(productID);
                db.close();
                return resp;
        }

        public Response newSpecialOffer(long productID){
                Response resp = db.newSpecialOffer(productID);
                db.close();
                return resp;
        }

        public Product[] getSpecialOffers(){
                Vector<Product>            productVector            =
db.getSpecial_offers();
                Product          products[]          =          new
Product[productVector.size()];
                for(int i=0;i<productVector.size();i++){
                        products[i] = productVector.elementAt(i);
                }
                db.close();
                return products;
        }


        public Response newOrder(String username,long productId, int
storeId,int quantity){
                 Order  order  =  new    Order(  username,  productId,
storeId,quantity,0);
                Response resp = db.newOrder(order);
                db.close();
                return resp;
        }

        public Response updateOrder(long orderID, long productID, int
quantity){

                Response  resp  =  db.updateOrder(orderID,  productID,
quantity);
                db.close();
```

```
                return resp;
        }

        public Order getOrder(long orderID){
                Order order= db.getOrder(orderID);
                db.close();
                return order;
        }

        public Product[] getProductsInOrder(long orderID){
                Vector<Product>            productVector            =
db.getProductsInOrder(orderID);
                db.close();
                Product          products[]          =          new
Product[productVector.size()];
                for(int i=0;i<productVector.size();i++){
                        products[i] = productVector.elementAt(i);
                }
                return products;
        }

        public Response finalizeOrder(long orderID){
                Order order= db.getOrder(orderID);
                db.close();
                XMLOrder xml = new XMLOrder(order);
                Response resp = xml.serialize();
                return resp;
        }

}
```