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

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

<< Κατασκευή Website Παρουσίασης Κατηγοριοποιημένων Αντικειμένων Με Custom Web Service>>

Των φοιτητών

Ανθιμίδης Νίκος

Αρ. Μητρώου: 093480

Κουπτσίδης Αβραάμ

Αρ. Μητρώου: 093445

Επιβλέπων καθηγητής

Δρ. Σφέτσος Παναγιώτης

ΠΡΟΛΟΓΟΣ

Στην παρούσα πτυχιακή εργασία η οποία πραγματοποιήθηκε στο Αλεξάνδρειο Τεχνολογικό Ίδρυμα Θεσσαλονίκης, στο τμήμα Μηχανικών Πληροφορικής της Σχολής Τεχνολογικών Εφαρμογών αναλύθηκαν και περιγράφονται τεχνικές ανάπτυξης τόσο διαδικτυακών όσο και εφαρμογών για κινητά ως ένα ολοκληρωμένο σύστημα. Κατόπιν αναλύσεως και προσεκτικής και ενδελεχούς έρευνας καταλήξαμε στα συστατικά που θα αποτελούν το σύστημά μας, δίνοντάς του ευελιξία, επεκτασιμότητα και ευκολία στην διαχείριση του περιεχομένου. Έτσι λοιπόν έγινε χρήση της πλατφόρμας Wordpress για την κεντρική παρουσίαση της εφαρμογής μας και συγχρόνως κατασκευάστηκε υβριδική εφαρμογή για smartphones (για Android και iOS με την χρήση του Ionic 1 Framework και του Adobe Phonegap) η οποία συγχρονίζεται με τα δεδομένα της ιστοσελίδας αξιοποιώντας ένα Web Service API. Με το API αυτό καταφέραμε τη διασύνδεση και επικοινωνία της εφαρμογής με την ιστοσελίδα και ταυτόχρονα μπορέσαμε να παρέχουμε μια υποδομή για μελλοντική επέκταση του συστήματος.

Η εφαρμογή είναι ενημερωτική και έχει ως θέμα την υποβολή παραπόνων με χρήση μιας φόρμας ηλεκτρονικής μορφής, η οποία ακολουθεί τους κανόνες τον αντίστοιχων φορμών που βρίσκονται στα κιτία παραπόνων στα φυσικά καταστήματα. Ταυτόχρονα παρέχονται δυνατότητες παρουσίασης, αναζήτησης και κατηγοριοποίησης της εκάστοτε επιχείρησης.

Οι λόγοι οι οποίοι μας οδήγησαν στην επιλογή αυτού του θέματος είναι η ανάγκη για ενημέρωση πάνω στα υγειονομικά προβλήματα των επιχειρήσεων παρέχοντας την δυνατότητα στους χρήστες να ενημερωθούν άμεσα για τυχόν προβλήματα της επιχείρηση μέσα από τα σχόλια άλλων πελατών. Στην συνέχεια οι φόρμες αυτές, όπως και οι αναλογικές, με την χρήση του Web Service που αναπτύχθηκε θα μπορούσαν να κατατίθενται ηλεκτρονικά στην αντίστοιχη υπεύθυνη αρχή για τα υγειονομικά παράπονα, επιτρέποντας έτσι την άμεση αξιολόγηση του καταστήματος. Ενώ τέλος επιλέχθηκε για ανάπτυξη μια υβριδική εφαρμογή με σκοπό την παροχή αυτών των υπηρεσιών στις πλατφόρμες που καταλαμβάνουν το μεγαλύτερο μερίδιο στην αγορά κινητών συσκευών.

ΠΕΡΙΛΗΨΗ

Σε αυτή την εργασία έγινε προσπάθεια να παρουσιαστούν κάποιες τεχνολογίες που έχουν ιδιαίτερη ανάπτυξη στις μέρες μας, σε σχέση με το διαδίκτυο και τα smartphones. Επίσης έγινε υλοποίηση ενός συστήματος χρησιμοποιώντας τεχνολογίες που παρουσιάστηκαν στην εργασία. Το σύστημα που υλοποιήθηκε περιλαμβάνει μία ιστοσελίδα, την deltio-paraponon.gr, που είναι υλοποιημένη με το πλέον δημοφιλέστερο CMS το Wordpress, μία mobile application (Kitio Paraponon App) και ένα Rest Web Service API το οποίο χρησιμεύει στο συγχρονισμό της ιστοσελίδας με mobile app.

Αρχικά βλέπουμε γενικότερα τι ορίζουμε σαν web application και ποιες οι λειτουργίες του. Web application μπορούμε να πούμε ότι είναι κάθε εφαρμογή στην οποία ο χρήστης έχει πρόσβαση από κάποιο browser και του δίνονται δυνατότητες να προσπελάσει ή να αποθηκεύσει δεδομένα. Οι εφαρμογές αυτές έχουν τον εξής τρόπο λειτουργίας, ο χρήστης δημιουργεί ένα αίτημα προς την εφαρμογή από browser (πχ ζητά να του ανοίξει η ιστοσελίδα google.gr), ο server προωθεί το αίτημα στην εφαρμογή, η εφαρμογή ετοιμάζει την απάντηση και τέλος ο server δίνει την απάντηση στον browser του χρήστη. Η απάντηση αυτή αποτελείται από γλώσσες τις οποίες αναγνωρίζουν οι browser(HTML, css, javascript).

Τα πλεονεκτήματα των διαδικτυακών εφαρμογών έναντι των τοπικών εφαρμογών είναι ότι είναι προσβάσιμες από όπου υπάρχει σύνδεση στο internet, είναι προσπελάσιμες από κάθε λειτουργικό σύστημα και δεν καταλαμβάνουν πόρους του συστήματος του χρήστη.

Μια διαδικτυακή εφαρμογή είναι και το Wordpress CMS (Content Management System). Με τον όρο CMS ορίζουμε τις εφαρμογές που διαχειρίζονται μαζικά ψηφιακό περιεχόμενο και μπορούν να χρησιμοποιηθούν ακόμα και από αρχάριους χρήστες χωρίς προγραμματιστικές γνώσεις. Το Wordpress είναι λογισμικό ανοιχτού κώδικα και είναι υλοποιημένο σε PHP.

Το Wordpress αν και ξεκίνησε σαν λογισμικό αποκλειστικά για την δημιουργία blog, στις ημέρες μας έχει εξελιχθεί σε μία πλατφόρμα που μπορούμε να δημιουργήσουμε από απλές ιστοσελίδες παρουσίασης μέχρι ηλεκτρονικά καταστήματα και πολύπλοκα μέσα κοινωνικής δικτύωσης. Αυτό

έχει επιτευχθεί χάριν της τεράστιας κοινότητας που υποστηρίζει το συγκεκριμένο λογισμικό. Υπάρχει ένας τεράστιος αριθμός πρόσθετων που μπορούν να εγκατασταθούν στη βασική έκδοση του Wordpress και να προσθέσουν πληθώρα λειτουργιών σε αυτό.

Ως μία διαδικτυακή εφαρμογή μπορούμε να ορίσουμε και τα Web Services. Τα web services είναι σχεδιασμένα ώστε να επιτρέπουν την επικοινωνία μεταξύ δύο ή και περισσότερων εφαρμογών. Γίνονται κλήσεις σε αυτά και οι απαντήσεις έχουν συνήθως τη μορφή XML(Soap επικοινωνία) ή JSON(Rest Επικοινωνία). Κάθε ένας από αυτούς του τρόπους επικοινωνίας έχει δικούς του κανόνες καθώς και διαφορετικά πλεονεκτήματα.

Τα web services συνήθως χρησιμοποιούνται για τον συγχρονισμό εφαρμογών, για την ανάκτηση δεδομένων και την ανταλλαγή δεδομένων μεταξύ συστημάτων. Στις μέρες μας γίνονται ιδιαίτερα δημοφιλή καθώς μπορούν να πετύχουν την επικοινωνία μεταξύ αυτόνομων συστημάτων που είναι υλοποιημένα σε διαφορετικές γλώσσες προγραμματισμού.

Ένας ακόμα τομέας που βλέπει ιδιαίτερη άνθιση στις ημέρες μας είναι οι εφαρμογές για κινητά τηλέφωνα (smartphones). Καθώς οι χρήστες των smartphones αυξάνονται ραγδαία, προκύπτει η ανάγκη δημιουργίας εφαρμογών για διάφορους τομείς και ζητήματα. Επειδή υπάρχουν διάφορες πλατφόρμες smartphones με τις πιο διαδεδομένες να είναι το Android της Google, το IOS της Apple και το Windows Phone από την Microsoft προκύπτει το πρόβλημα ότι για τις πλατφόρμες αυτές η ανάπτυξη θα πρέπει να γίνεται με διαφορετικό τρόπο.

Οι δύο βασική τρόποι υλοποίησης αυτόνομων εφαρμογών για smartphones είναι η Native και η υβριδική (Hybrid). Με την πρώτη υλοποιούμε την κάθε εφαρμογή ξεχωριστά για κάθε πλατφόρμα ενώ με τη δεύτερη μεθοδολογία η υλοποίηση γίνεται μία φορά και έπειτα ο κώδικας μεταφράζεται με μικρές αλλαγές για κάθε πλατφόρμα. Η κάθε μεθοδολογία έχει τα δικά της πλεονεκτήματα τα οποία θα αναλυθούν εκτενέστερα στην εργασία.

Μια βασική ανάγκη που υπάρχει στις διαδικτυακές εφαρμογές είναι η διαχείριση μεγάλου όγκου δεδομένων. Αυτό το πετυχαίνουμε με τη χρήση

Βάσεων Δεδομένων. Με τις βάσεις δεδομένων μπορούμε να οργανώσουμε, να αποθηκεύσουμε και να προσπελάσουμε μεγάλο όγκο δεδομένων σε μικρό χρόνο.

Οι πιο διαδεδομένες βάσεις δεδομένων είναι οι σχεσιακές βάσεις δεδομένων, οι οποίες αποθηκεύουν τα δεδομένα σε μορφή πινάκων και έπειτα μπορούμε να τις προσπελάσουμε με μία γλώσσα προγραμματισμού σχεδιασμένη γι' αυτό το σκοπό, την SQL. Η SQL είναι μία μη διαδικαστική γλώσσα προγραμματισμού που αρχικά η σχεδίασή της βασίστηκε στην σχεσιακή αλέγρα. Μια από τις πιο διαδεδομένες open source βάσεις δεδομένων είναι η MySQL η οποία είναι και η βάση δεδομένων που υποστηρίζει το Wordpress και την χρησιμοποιούμε και στη δική μας εφαρμογή.

Για μικρότερες ανάγκες αποθήκευσης δεδομένων, όπως για παράδειγμα σε εφαρμογές κινητών τηλεφώνων, μπορούμε να χρησιμοποιήσουμε την SQLite η οποία είναι μία σχεσιακή βάση δεδομένων που λειτουργεί σε ένα αρχείο και δεν χρειάζεται ολόκληρο εξυπηρετητή για τη λειτουργία της.

Με σκοπό να συνδυάσουμε τις παραπάνω τεχνολογίες σε ένα σύστημα αποφασίσαμε να δημιουργήσουμε μία διαδικτυακή εφαρμογή (detio-paraponon.gr) στην οποία θα βάλουμε κατηγοριοποιημένα καταστήματα και οι χρήστες θα έχουν τη δυνατότητα να αφήνουν σχόλια για αρνητικές εμπειρίες από τα καταστήματα αυτά. Οι υπόλοιποι χρήστες θα έχουν τη δυνατότητα να βλέπουν αυτές τις αξιολογήσεις. Τα καταστήματα επίσης θα παρουσιάζονται και σε μορφή χάρτη. Στη συνέχεια υλοποιήσαμε μία mobile app η οποία εκμεταλλεύεται τα δεδομένα της διαδικτυακής εφαρμογής μέσω ενός API που δημιουργήσαμε. Έτσι οι χρήστες της εφαρμογής θα έχουν πρόσβαση σε αυτά τα δεδομένα και από το κινητό τους τηλέφωνο.

Ολόκληρο το σύστημά μας αποτελείται από τρεις οντότητες, τα καταστήματα, τους χρήστες και τις αξιολογήσεις.

Η διαδικτυακή εφαρμογή υλοποιήθηκε σε Wordpress. Αγοράσαμε ένα Theme το οποίο βασιζόταν σε ένα πρόσθετο, κατάλληλο για την κατηγοριοποίηση και παρουσίαση οντοτήτων. Στην δική μας περίπτωση αυτές οι οντότητες είναι τα καταστήματα.

Όπως έχουμε αναφέρει, με το Wordpress μπορούμε να υλοποιήσουμε ιστοσελίδες για πολλούς σκοπούς. Η υλοποίηση συνήθως δεν γίνεται από την αρχή καθώς υπάρχουν εταιρίες ή και μεμονωμένοι προγραμματιστές που υλοποιούν Themes για διάφορα σενάρια χρήσης. Αυτά τα themes διαθέτουν το αισθητικό κομμάτι που είναι κατάλληλο ανάλογα με το θέμα που πραγματεύονται, καθώς και την υποδομή για να αποθηκεύουν σχετικά δεδομένα από τον χρήστη. Έτσι με σχετικά χαμηλή τιμή από δωρεάν έως 90€ μπορούμε να αποκτήσουμε κάποιο Theme σχετικό με αυτό που θέλουμε να υλοποιήσουμε και να έχουμε σε μεγάλο βαθμό την υποδομή για να βασιστούμε.

Η εγκατάσταση του Wordpress γίνεται με εύκολο τρόπο, κατεβάζοντας την τελευταία έκδοση από την επίσημη ιστοσελίδα και στη συνέχεια ανεβάζοντας την στο server που την φιλοξενήσει. Αρχικά γίνονται οι ρυθμίσεις σχετικά με την ονομασία της ιστοσελίδας, την δημιουργία χρήστη και τις ρυθμίσεις σύνδεσης με τη βάση δεδομένων και έπειτα εκτελείτε η εγκατάσταση. Με το που ολοκληρωθεί η εγκατάσταση η ιστοσελίδα είναι έτοιμη για τον χρήστη.

Μετά την εγκατάσταση η συνήθης διαδικασία είναι να γίνεται προσθήκη κάποιο theme και των σχετικών plugins για το σκοπό που θέλουμε να δημιουργήσουμε την ιστοσελίδα. Σε περίπτωση που θέλουμε να κάνουμε προγραμματιστικές αλλαγές, η πιο σωστή πρακτική είναι να χρησιμοποιούμε child-theme και το hook σύστημα του Wordpress (θα γίνει αναλυτική αναφορά στην σχετική ενότητα της εργασίας).

Το Wordpress διαθέτει διασύνδεση με τη βάση δεδομένων MySQL και ο τρόπος με τον οποίο αποθηκεύει τα δεδομένα σε αυτήν είναι ο εξής: Τα βασικά δεδομένα αποθηκεύονται στον πίνακα wp_posts και τη χρήση μίας στήλης που σχετίζεται με το τι τύπος δεδομένων είναι η εγγραφή έτσι γίνεται ο σχετικός διαχωρισμός στο είδος των δεδομένων. Επίσης υπάρχει ο πίνακας wp_postmeta που αποθηκεύει δεδομένα που σχετίζονται με τα δεδομένα του wp_posts και δεν υπάρχει σχετική στήλη για αυτά. Άλλοι βασικοί πίνακες είναι οι wp_users, wp_usermeta και wp_comments που σχετίζονται με τους χρήστες και τα σχόλια που δημιουργούν στην ιστοσελίδα.

Τα διάφορα plugins, για να αποθηκεύουν δεδομένα, έχουν τη δυνατότητα είτε να χρησιμοποιούν τους πίνακες που υπάρχουν ήδη (με αυτή τη λογική

γίνεται η αποθήκευση των καταστημάτων στο plugin που χρησιμοποιούμε εμείς), δημιουργώντας καινούριους τύπους δεδομένων, είτε να δημιουργούν καινούριους πίνακες που να είναι κατάλληλοι για τα δεδομένα τους.

Σε σχετική ενότητα στην εργασία μας θα παρουσιάσουμε αναλυτικά τον τρόπο που αποθηκεύει τα δεδομένα το Wordpress καθώς και η εφαρμογή μας συγκεκριμένα. Επίσης θα παρουσιάσουμε και κομμάτια κώδικα του λογισμικού.

Για τη δημιουργία Web Service API για ανάκτηση δεδομένων μίας ιστοσελίδας Wordpress, υπάρχουν έτοιμα plugins. Εμείς δοκιμάσαμε να κάνουμε χρήση κάποιου τέτοιου plugin αλλά υπήρξε πρόβλημα στην ανάκτηση των δεδομένων του πρόσθετου για τα listings που χρησιμοποιεί η ιστοσελίδα μας. Για αυτό το λόγο αποφασίσαμε να δημιουργήσουμε ένα Web Service από την αρχή, το οποίο θα κάνει απευθείας ερωτήματα στη βάση δεδομένων, θα ανακτά και θα παρουσιάζει τα δεδομένα που χρειάζεται σε μορφή JSON.

Αυτό το Web Service το υλοποιήσαμε σε Object Oriented PHP και διαθέτει έξι αρχεία, εκ των οποίων τα πέντε έχουν από μία κλάση που είναι υπεύθυνη να διαχειρίζεται κάποια λειτουργία της εφαρμογή (πχ τη σύνδεση στη βάση δεδομένων) ή να διαχειρίζεται κάποια οντότητα της εφαρμογής (πχ τα listings ή τους χρήστες). Το ένα αρχείο που περισσεύει είναι το αρχείο που γίνεται η εισαγωγή στο API και ρυθμίζει το routing με βάση τις μεταβλητές που καλείτε το API. Εντός των κλάσεων έχουμε ιδιωτικές μεθόδους, οι οποίες εκτελούν - ετοιμάζουν τα δεδομένα που ανακτούμε και δημόσιες μέθοδοι που παρουσιάζουν αυτά τα δεδομένα.

Τα πλεονεκτήματα της υλοποίησης custom Web Service API είναι ότι το προσαρμόζουμε ακριβώς στα ανάγκες μας, ανακτούμε δηλαδή τα δεδομένα με τη δομή που θέλουμε. Επίσης το API είναι αυτόνομο και ανεξάρτητο από το Wordpress και τέλος είναι πιο γρήγορο καθώς εκτελεί αποκλειστικά τις λειτουργίες που θέλουμε και τίποτα παραπάνω.

Ο λόγος ύπαρξης του Web Service είναι να μπορεί να τροφοδοτεί την mobile application με δεδομένα. Η mobile app υλοποιήθηκε με την υβριδική μεθοδολογία ανάπτυξης εφαρμογών για κινητά τηλέφωνα. Ποιο συγκεκριμένα χρησιμοποιήθηκαν το Ionic 1 Framework για την ανάπτυξη και η υπηρεσία

Adobe Phonegap Build για να γίνει η παραγωγή των τελικών app από των πηγαίο κώδικά.

Με το Ionic 1 η υλοποίηση πραγματοποιείται σε ένα κανονικό browser με τη χρήση των γλωσσών HTML, CSS και JavaScript. Έπειτα από αυτόν τον κώδικα παράγονται οι τελικές εφαρμογές που διατίθενται στους χρήστες smartphone.

Η εφαρμογή ανακτά τα δεδομένα της ιστοσελίδας με τη χρήση του Web Service. Επειδή θέλαμε η εφαρμογή μας να αποθηκεύει τα δεδομένα τοπικά στη συσκευή ώστε να μπορεί να εκτελείται και χωρίς internet, έχουμε χρησιμοποίηση την SQLite για την αποθήκευση των δεδομένων στο κινητό.

Τα αρχικά δεδομένα τα κατεβάζει ο χρήστης την πρώτη φορά που θα κατεβάσει την εφαρμογή. Έπειτα γίνεται η ανανέωση των δεδομένων με τον εξής τρόπο: όταν ανοίξει η εφαρμογή γίνεται έλεγχος εάν το κινητό έχει πρόσβαση στο internet, σε περίπτωση που έχει γίνεται ερώτηση στο Web Service για το αν τα δεδομένα της ιστοσελίδας είναι πιο πρόσφατα από αυτά του κινητού. Σε περίπτωση που τα δεδομένα της ιστοσελίδας είναι πιο πρόσφατα γίνεται drop όλων των τοπικών δεδομένων και στη συνέχεια γίνεται εισαγωγή όλων των σχετικών δεδομένων της ιστοσελίδας. Με το που ολοκληρωθεί όλη η διαδικασία η εφαρμογή είναι έτοιμη για χρήση.

Η εφαρμογή ακολουθεί την MVC (Model View Controller) σχεδίαση. Το Ionic 1 διαθέτει MVC σχεδιασμό με μία διαφοροποίηση του model. Δεν υπάρχει η έννοια του Model αλλά εισάγεται η έννοια του Service. Services είναι αυτόνομες λειτουργίες που ενεργούν πάνω στα δεδομένα της εφαρμογής και μπορούν να χρησιμοποιηθούν από όλους τους controllers της εφαρμογής αρκεί να δηλωθούν σαν dependencies. Στα services της εφαρμογής μας έχουμε υλοποιήσεις τις κλήσεις στο API και την ανάκτηση των δεδομένων από την τοπική SQLite.

Στην ανάπτυξη των υβριδικών εφαρμογών με το Ionic ή με το Phonegap / Cordova μπορούμε να προσθέσουμε plugins τα οποία μας βοηθούν να υλοποιήσουμε λειτουργίες όπως τη σύνδεση με την SQLite, τον έλεγχο αν η συσκευή έχει internet κα.

Με το που ολοκληρώσαμε την υλοποίηση οι επόμενες δύο ενέργειές μας ήταν να κάνουμε upload των κώδικα στο phonegap build έτσι ώστε να παράγουμε τα τελικά apps. Τελευταίο στάδιο είναι να διαθέσουμε αυτά τα app στα καταστήματα εφαρμογών του κάθε λειτουργικού συστήματος.

Βλέποντας το τελικό αποτέλεσμα, συμπεράνουμε ότι οι τεχνολογίες που συνδυάσαμε (Wordpress CMS, Custom PHP Web Service & Hybrid mobile app με τη χρήση του Ionic 1 Framework) μπορούν να συνεργαστούν άψογα μεταξύ τους.

Αν αναλογιστεί κάποιος την πληθώρα των τεχνολογιών που υπάρχουν για την ανάπτυξη τέτοιου τύπου εφαρμογών, μπορεί να συμπεράνει ότι το συγκεκριμένο αποτέλεσμα θα μπορούσαμε να το υλοποιήσουμε και με τη χρήση κάποιων άλλων τεχνολογιών.

Οπότε αυτό που έχει σημασία είναι η επιλογή των τεχνολογιών που θα χρησιμοποιηθούν για να επιλύσουν μία σύνθετη υλοποίηση, να μπορούν να συνδυαστούν αρμονικά μεταξύ τους. Επίσης, επειδή μεγάλη σημασία έχει και η ταχύτητα στην υλοποίηση κάποιας εφαρμογής, είναι σημαντικό να επιλέγονται τεχνολογίες με τις οποίες υπάρχει εξοικείωση από την ομάδα προγραμματιστών που θα αναλάβει την υλοποίηση.

ABSTRACT

In this thesis, which took place at the Alexandrio Technological Institute of Thessaloniki, in the Department of Informatics Engineering, we have created a system which includes several technologies such as Wordpress CMS, Hybrid Mobile app and Web Service Api.

Initially we define what is a Web application, how does it work and what benefits does it provide. We are referring to such matter because it is important to understand the applicable advantages such an application, when deployed in the Web, has.

Afterwards, an extensive report is made on the concept of Content Management Systems (CMS) where the definition of a CMS is given, alongside with the basic functionality and architecture of such systems. Moreover, we present Wordpress as our CMS where references are given to its main features and also some historical data on its evolution over time and some features of it's versions.

Focus is given on an attempt to give a clear definition of what a Web service is, how does it work and what are the main advantages and disadvantages using them. In addition, a reference is made in the way we can access a Web service presented the two main access methods, which are SOAP and ReST.

In the following chapter, we discuss about ways and technologies for developing mobile applications. We give a thorough examination on each technology providing a definition and comparing each development technology.

Trailing the path of a Web based system, we presented a crucial component of every web application, the database. Here we try to approach the way each database achieves its goal by analyzing what a database is, the system behind it in order to process, create, manage the stored data and of course the programming language to interact with it. The analysis includes also, the ways mobile application manages databases and their data.

Finally, we present the system which was implemented by describing the functions used and its development process.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΠΙΝΑΚΑΣ ΠΙΝΑΚΩΝ

ΚΕΦΑΛΑΙΟ 1

Web Applications

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο θα παρουσιαστεί η έννοια του web application, ο τρόπος λειτουργίας του και θα αποδοθούν κάποια θετικά και αρνητικά χαρακτηριστικά της χρήσης τέτοιων.

1.1 Τι είναι ένα Web application

Μια διαδικτυακή εφαρμογή αποτελείται από ένα σύνολο λειτουργιών οι οποίες δίνουν στην εφαρμογή δυνατότητες αλληλεπίδρασης με τον χρήστη τέτοιες ώστε να μπορεί να αποθηκεύει, ανακτά και να διαβάζει δεδομένα είτε προσωπικά είτε γενικότερης σημασίας. Η πρόσβαση στις δυνατότητες της εφαρμογής επιτυγχάνεται με την διαθεσιμότητα της μέσω Διαδικτύου ή ενδοδικτύου και πάντα με την χρήση ενός περιηγητή. Οι εφαρμογές αυτές συνήθως εκτελούνται σε ισχυρές υπολογιστικές μηχανές οι οποίες έχουν τον

ρόλο του σταθμού εξυπηρέτησης και παρέχουν τις υπηρεσίες τους σε περισσότερους του ενός χρήστη.

1.2 Τρόπος λειτουργίας ενός Web Application

Μια διαδικτυακή εφαρμογή είναι συνήθως κωδικοποιημένη σε γλώσσες οι οποίες υποστηρίζονται από την πλειοψηφία των περιηγητών, όπως η HTML, και η Javascript. Αυτό συμβαίνει γιατί δίνεται η δυνατότητα η εκτέλεση τέτοιων εφαρμογών να γίνεται από τους περιηγητές. Βέβαια υπάρχουν και εφαρμογές δυναμικού περιεχομένου, οι οποίες εκτελούνται σε έναν εξυπηρετητή επιστρέφοντας κάποιο αποτέλεσμα το οποίο με την σειρά του ερμηνεύεται από κάποιον περιηγητή. Αυτή είναι μια πρακτική η οποία χρησιμοποιείται όταν χρειάζεται για παράδειγμα να αποθηκευτούν δεδομένα για πολλούς χρήστες σε μια βάση.

Παρακάτω παρουσιάζεται μια βασική ροή του τρόπου λειτουργίας μιας τέτοιας εφαρμογής:

- Ο χρήστης κάνει ένα αίτημα για μια πληροφορία στον εξυπηρετητή μέσω του Διαδικτύου με την βοήθεια ενός περιηγητή.
- Ο εξυπηρετητής προωθεί το αίτημα στην εφαρμογή.
- Η εφαρμογή πραγματοποιεί την απαραίτητη διεργασία, όπως το να βρει δεδομένα από μια βάση, και παράγει το σχετικό αποτέλεσμα.
- Το αποτέλεσμα αυτό στέλνεται στον εξυπηρετητή.
- Ο εξυπηρετητής «απαντάει» στον χρήστη με την πληροφορία η οποία ζητήθηκε.

Εικόνα 1 – Βασικός κύκλος ροής διαδικτυακών εφαρμογών

1.3 Πλεονεκτήματα και μειονεκτήματα διαδικτυακών εφαρμογών

Από την πρώτη εμφάνιση των ηλεκτρονικών υπολογιστών οι τοπικές εφαρμογές είναι αυτές που κατ' εξοχήν είναι συνδεδεμένες, στο μυαλό των

χρηστών, με τις δυνατότητες των υπολογιστών. Βέβαια με την εξέλιξη του Διαδικτύου τα πράγματα έχουν διαφοροποιηθεί και έτσι οι χρήστες καλούνται να αποφασίσουν τι είναι αυτό που αναζητούν και να αποφασίσουν ποιος τύπος εφαρμογών τους ταιριάζει. Παρακάτω παρατίθενται κάποια θετικά και αρνητικά της χρήσης διαδικτυακών εφαρμογών.

Πλεονεκτήματα

- **Συμβατές με όλα τα λειτουργικά συστήματα και από οποιαδήποτε συσκευή:** Οι χρήστες έχουν την δυνατότητα άμεσης πρόσβασης στις εφαρμογές που θέλουν να χρησιμοποιήσουν ανεξαρτήτως της συσκευής ή του λειτουργικού συστήματος αρκεί η συσκευή να μπορεί να συνδεθεί στο Διαδίκτυο και να υπάρχει κάποιο πρόγραμμα περιηγητή. Και οι δυο προηγούμενες προϋποθέσεις απαντώνται σε όλες τις σύγχρονες συσκευές.
- **Δυνατότητα χρήσης ανεξαρτήτου τοποθεσίας:** από την στιγμή που μια συσκευή είναι συνδεδεμένη στο Διαδίκτυο τότε υπάρχει πρόσβαση στις εφαρμογές που θέλει ο χρήστης ανεξαρτήτως του χώρου στον οποίο βρίσκεται. Έτσι δίνεται η δυνατότητα στους χρήστες να εργάζονται από απομακρυσμένες περιοχές.
- **Δεν καταλαμβάνουν χώρο και πόρους:** εξαιτίας του ότι οι Διαδικτυακές εφαρμογές δεν εκτελούνται στην συσκευή αυτή καθαυτή δεν απορροφώνται πόροι από την συσκευή. Ταυτόχρονα η αποθήκευση δεδομένων γίνεται στον εξυπηρετητή και όχι στην συσκευή. Έτσι δεν καταναλώνεται και χώρος στην συσκευή.

Μειονεκτήματα

- **Συμβατότητα των περιηγητών:** αρκετοί περιηγητές δεν είναι ακόμα πλήρως συμβατοί με την τελευταία έκδοση της HTML, έτσι δεν γίνεται πλήρη χρήση των δυνατοτήτων αυτών. Συνεπώς αν δεν έχει γίνει πρόβλεψη για την μη λειτουργία κάποιου χαρακτηριστικού της εφαρμογής σε κάποιον περιηγητή αυτό μπορεί να έχει ως αποτέλεσμα να μην λειτουργεί σωστά ή να μην λειτουργεί καθόλου η εφαρμογή.

- **Πλήρη χρήση χωρίς σύνδεση στο Διαδίκτυο:** χρησιμοποιώντας την τελευταία έκδοση της HTML μπορούμε να δώσουμε στην εφαρμογή μας δυνατότητες χρήσης εκτός σύνδεσης.

ΕΠΙΛΟΓΟΣ

Καταλήγοντας, σε αυτό το κεφάλαιο παρουσιάστηκε η έννοια της διαδικτυακής εφαρμογής. Έγινε μια προσπάθεια ανάλυσης και παρουσίασης ενός ορισμού για τέτοιου είδους εφαρμογές. Ταυτόχρονα δόθηκαν σαφής αναλύσεις και διαχωριστικές γραμμές του τρόπου λειτουργίας και ανάπτυξης εφαρμογών επιπέδου διαδικτύου όπως επίσης παρατέθηκαν πλεονεκτήματα και μειονεκτήματα της κατά κόρου χρήσης τους σε σχέση με ποιο συμβατικού τύπου εφαρμογών όπως για παράδειγμα οι τοπικές εφαρμογές. Όλα αυτά θα βοηθήσουν να καταλάβουμε καλύτερα την αιτία επιλογής ενός CMS(Σύστημα διαχείρισης περιεχομένου) και δη του Wordpress για την ανάπτυξη της εφαρμογής μας.

ΚΕΦΑΛΑΙΟ 2

CMS - Wordpress

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφαλαίο θα αναλύσουμε τις βασικές ιδιότητες ενός CMS και ειδικότερα του Wordpress με σκοπό να αντιληφθούμε το τρόπο λειτουργίας του. Επίσης θα παρουσιαστεί η εξέλιξη του και τα βασικά χαρακτηριστικά του.

2.1 Ορισμός CMS

Ένα CMS αποτελεί μια εφαρμογή διαχείρισης ψηφιακού περιεχομένου. Για να χαρακτηριστεί μια εφαρμογή ως CMS πρέπει να πληροί κάποιες προδιαγραφές και να υλοποιεί συγκεκριμένους μηχανισμούς οι οποίοι βοηθούν ακόμα και τους πιο αρχάριους και μη εξοικειωμένους χρήστες στην διαχείριση περιεχομένου.

2.2 Βασικές λειτουργίες ενός CMS

Για να μπορέσει ένα τέτοιο σύστημα να λειτουργήσει αποδοτικά, εξαιτίας του ότι βρίσκεται στο Διαδίκτυο και η πρόσβασή του γίνεται από πολλούς

χρήστες ταυτόχρονα, πρέπει να πληρούνται κάποιες προδιαγραφές. Μερικές από αυτές αναφέρονται παρακάτω:

- Πρέπει να υπάρχει ένα σύστημα αρχείων που αποθηκεύει προσωρινά σε μια κρυφή μνήμη τις ιστοσελίδες. Ο εξυπηρετητής επικοινωνεί με το σύστημα αρχείων για να βρει τις ιστοσελίδες στην κρυφή μνήμη.
- Πρέπει να υπάρχει μια βάση δεδομένων που προσομοιώνει την δομή των φακέλων έτσι ώστε να συσχετίζονται οι ιστοσελίδες με διάφορα χαρακτηριστικά.
- Ένα κεντρικό σύστημα διαχείρισης χρηστών το οποίο είναι υπεύθυνο για την παραχώρηση ιεραρχικών δικαιωμάτων σε τουλάχιστον ένα σύνολο χρηστών με σκοπό την επεξεργασία των ιστοσελίδων.
- Ένα κεντρικό σύστημα διαχείρισης σελίδων με σκοπό την επεξεργασία, την δημιουργία και την σύνδεση διαφόρων χαρακτηριστικών των σελίδων καθώς επίσης και την σύζευξη των σελίδων αυτών με την ιεραρχική δομή και πλοήγηση της εφαρμογής.

2.3 Παρουσίαση Wordpress

Το Wordpress αποτελεί ένα OnLine, ανοιχτού κώδικα σύστημα δημιουργίας και διαχείρισης περιεχομένου και ιστοσελίδων. Δίνεται έτσι η δυνατότητα δημοσίευσης κάθε είδους περιεχόμενου όπως άρθρα, εικόνες, βίντεο και γενικότερα πληροφοριών για προϊόντα και υπηρεσίες.

Για την σωστή λειτουργία του πρέπει να πληρούνται οι παρακάτω προδιαγραφές:

- Εγκατάσταση PHP έκδοσης 7 ή μεγαλύτερη
- Εγκατάσταση MySQL έκδοσης 5.6 ή μεγαλύτερη / MariaDB έκδοσης 10.0 ή μεγαλύτερη
- Υποστήριξη HTTPS πρωτοκόλλου.

2.4 Ιστορικά στοιχεία Wordpress

Η αρχική ημερομηνία κυκλοφορίας του Wordpress, το όνομα του οποίου σκέφτηκε μια φίλη του δημιουργού ονόματι Christine Selleck, ήταν στις 27 Μαΐου 2003 από τον Matt Mullenweg ως παραλλαγή του b2/cafelog. Αργότερα μέσα στον χρόνο η ανταγωνίστρια εταιρία άλλαξε κάποιους από τους όρους

χρήσης της οδηγώντας έτσι πολλούς χρήστες στο Wordpress. Από την στιγμή αυτή και έπειτα βραβεύεται ανά σειρά ετών(2007, 2009, 2010) ως ένα από τα καλύτερα εργαλεία διαχείρισης περιεχομένου.

Παρακάτω ακολουθούν οι ημερομηνίες κλειδί καθώς και οι εκδόσεις για της εξέλιξη του:

Version	Code name	Release date	Notes
0.7	none	Μάιος 27, 2003	Ουσιαστικά αποτελεί συνέχεια του προκατόχου του, *b2/cafelog*, γι' αυτό και συνεχίστηκε η αρίθμηση από την τελευταία του έκδοση.
1.0	*Davis*	Ιανουάριος 3, 2004	Προστέθηκε η δυνατότητα χρησιμοποίησης φιλικών προς τις μηχανές αναζήτησης μόνιμων συνδέσμων καθώς και η υποστήριξη πολλαπλών κατηγοριών.
1.2	*Mingus*	Μάιος 22, 2004	Προστέθηκε η δυνατότητα χρησιμοποίησης Plugins.
1.5	*Strayhorn*	Φεβρουάριος 17, 2005	Προστέθηκε η δυνατότητα διαχείρισης στατικών σελίδων και το σύστημα οπτικών θεμάτων. Επίσης συνοδευόταν από ένα καινούριο προεπιλεγμένο οπτικό θέμα, με το όνομα Kubrick.
2.0	*Duke*	Δεκέμβριος 31, 2005	Προστέθηκε η δυνατότητα επεξεργασίας του κειμένου, ανέβασμα εικόνων κτλ. Με την έκδοση αυτή βελτιώθηκαν οι δυνατότητες του προγραμματισμού πρόσθετων λειτουργιών.
2.1	*Ella*	Ιανουάριος 22, 2007	Διορθώθηκαν σημαντικά θέματα ασφάλειας καθώς επίσης έγινε και ανασχεδίαση της βασικής διεπαφής.

2.2	*Getz*	Μάιος 16, 2007	Προστέθηκε η δυνατότητα υποστήριξης widget σε οπτικά θέματα.
2.3	*Dexter*	Σεπτέμβριος 24, 2007	Προστέθηκε η υποστήριξη ετικετών, δημιουργήθηκε νέο σύστημα ιεραρχίας των κατηγοριών και βελτιώθηκε η διαδικασία των ειδοποιήσεων.
2.5	*Brecker*	Μάρτιος 29, 2008	Τροποποιήθηκε ο πίνακας ελέγχου, υποστήριξη για πολλαπλό ανέβασμα αρχείων.
2.6	*Tyner*	Ιούλιος 15, 2008	Προστέθηκε η δυνατότητα εντοπισμού αλλαγών σε κάθε άρθρο και σελίδα.
2.7	*Coltrane*	Δεκέμβριος 11, 2008	Ανασχεδιασμός του διαχειριστικού πίνακα ελέγχου.
2.8	*Baker*	Ιούνιος 10, 2009	Βελτιώθηκε η ταχύτητα, και εισάχθηκε η αυτόματη εγκατάσταση οπτικών θεμάτων μέσα από το περιβάλλον διαχείρισης.
2.9	*Carmen*	Δεκέμβριος 19, 2009	Προστέθηκε η δυνατότητα αναίρεσης κινήσεων σε πολλά χαρακτηριστικά του προγράμματος, ένας ενσωματωμένος επεξεργαστής εικόνων, μαζική αναβάθμιση πρόσθετων λειτουργιών,.
3.0	*Thelonious*	Ιούνιος 17, 2010	Προστέθηκαν καινούργια APIs για τα οπτικά θέματα, δημιουργήθηκε η λειτουργία πολλαπλών ιστοτόπων, δημιουργήθηκε ένα προ εγκατεστημένο οπτικό θέμα, το "Twenty Ten".
3.1	*Reinhardt*	Φεβρουάριος 23, 2011	Προστέθηκε μια μπάρα διαχείρισης, η οποία προβάλλεται σε όλες τις σελίδες του ιστολογίου όταν ο διαχειριστής είναι συνδεδεμένος.
3.2	*Gershwin*	Ιούλιος 4, 2011	Δόθηκε βάση στην βελτιστοποίηση της ταχύτητας και του μεγέθους του Wordpress.

3.3	*Sonny*	Δεκέμβριος 12, 2011	Δόθηκε βάση στο να γίνει φιλικότερο προς τους αρχάριους χρήστες.
3.4	*Green*	Ιούνιος 13, 2012	Δόθηκε βάση σε βελτιστοποιήσεις που αφορούν την παραμετροποίηση των οπτικών θεμάτων.
3.5	*Elvin*	Δεκέμβριος 11, 2012	Προστέθηκε η υποστήριξη της Retina Display των υπολογιστών Macintosh, η δυνατότητα επιλογής χρώματος και το νέο προεπιλεγμένο οπτικό θέμα "Twenty Twelve".
3.6	*Oscar*	Αύγουστος 1, 2013	Νέο προεπιλεγμένο οπτικό θέμα "Twenty Thirteen", νέο σύστημα αναθεώρησης άρθρων και σελίδων, αυτόματη αποθήκευση και κλείδωμα άρθρου.
3.7	*Basie*	Οκτώβριος 24, 2013	Προστέθηκε η δυνατότητα συντήρησης και ενημέρωσης στο παρασκήνιο, επιλογής και διατήρησης των σωστών αρχείων γλώσσας.
3.8	*Parker*	Δεκέμβριος 12, 2013	Βελτιστοποίηση της διαχείρισης, δυνατότητα χρήσης γραμματοσειράς Open Sans, νέο προεπιλεγμένο οπτικό θέμα "Twenty Fourteen".
3.9	*Smith*	Απρίλιος 16, 2014	Βελτιστοποίηση του επεξεργαστή κειμένου για την υποστήριξη οπτικών μέσων, προστέθηκε η δυνατότητα επισκόπηση των widget και των κεφαλίδων.
4.0	*Benny*	Σεπτέμβριος 4, 2014	Βελτίωση διαχείρισης οπτικοακουστικών μέσων, ευκολία αλλαγής γλώσσας, παραμετροποίηση οπτικών ενθεμάτων.
4.1	*Dinah*	Δεκέμβριος 18, 2014	Νέο προεπιλεγμένο οπτικό θέμα "Twenty Fifteen".
4.2	*Powell*	Απρίλιος 23, 2015	Προστέθηκε η λειτουργία "Press This", βελτίωση υποστήριξης χαρακτήρων,

			υποστήριξη emoji, βελτίωση του συστήματος των προσθέτων.
4.3	*Billie*	Αύγουστος 18, 2015	Βελτίωση της χρήσης από κινητές συσκευές.
4.4	*Clifford*	Δεκέμβριος 8, 2015	Νέο προεπιλεγμένο οπτικό "Twenty Sixteen", και βελτίωση του τρόπου εμφάνισης των εικόνων σε διαφορετικές αναλύσεις.
4.5	*Coleman*	Απρίλιος 12, 2016	Προστέθηκε η δυνατότητα "επι τόπου αναθεώρησης" των εικόνων.
4.6	*Pepper*	Αύγουστος 16, 2016	Βελτιώθηκε το σύστημα ενημερώσεων καθώς και ο επεξεργαστής κειμένου, προστέθηκαν επιπλέον γραμματοσειρές.
4.7	*Vaughan*	Δεκέμβριος 6, 2016	Νέο προεπιλεγμένο οπτικό "Twenty Seventeen", προστέθηκε υποστήριξη κεφαλίδων για βίντεο, προεπισκόπηση PDF, προεπισκόπηση CSS, βελτιώσεις στον επεξεργαστή κειμένου.

Πίνακας 1 – Ιστορικά στοιχεία Wordpress

ΕΠΙΛΟΓΟΣ

Συνοψίζοντας, σε αυτό το κεφάλαιο γίνεται μια αναφορά στο τι είναι ένα σύστημα διαχείρισης περιεχομένου και ποιες είναι οι λειτουργίες αυτές που του δίνουν την δομή του. Εκτενέστερα παρουσιάστηκε η πλατφόρμα Wordpress ως ένα χαρακτηριστικό παράδειγμα ενός τέτοιου συστήματος και συγχρόνως δόθηκαν οι απαραίτητες προδιαγραφές για την σωστή χρήση της. Τέλος, έγινε ανάλυση της προόδου της πλατφόρμας με την πάροδο του χρόνου και την εξέλιξη της τεχνολογίας με σκοπό να γίνουν αντιληπτοί οι αυτοματισμοί και οι βελτιώσεις που προστέθηκαν. Έτσι τα παραπάνω μας βοηθούν να συνθέσουμε την εφαρμογή μας δίνοντάς της δυνατότητες αλληλεπίδρασης με διαφορετικές τεχνολογίες και πλατφόρμες προσθέτοντάς της υλοποιήσεις που αξιοποιούν διάφορες υπηρεσίες Διαδικτύου (Web services).

ΚΕΦΑΛΑΙΟ 3

Web Services

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφαλαίο θα δώσουμε έναν ορισμό για τα Web services καθώς επίσης θα επιχειρήσουμε να αναλύσουμε τόσο τον τρόπο λειτουργίας τους και τις δυνατότητες τους όσο και τις μεθόδους πρόσβασης σε αυτά και ταυτόχρονα θα γίνει προσπάθεια εμπειρικής προσέγγισης σε αυτά με σκοπό την αιτιολόγηση χρήσης και την σαφέστερη κατανόησή τους. Τέλος θα απαριθμηθούν τα θετικά και τα αρνητικά της χρήσης αυτής της τεχνολογίας.

3.1 Ορισμός Web service

Ένα Web service χαρακτηρίζεται ως μια εφαρμογή λογισμικού που αναγνωρίζεται από μια διεύθυνση URI, της οποίας οι διεπαφές και οι διασυνδέσεις μπορούν να προσδιοριστούν, να περιγραφούν και να ανακαλυφθούν ως κάποιου τύπου αντικείμενα όπως για παράδειγμα XML, JSON κτλ. Ένα web service υποστηρίζει την απευθείας ενσωμάτωση με άλλα προγράμματα λογισμικού με την ανταλλαγή μηνυμάτων μέσω πρωτοκόλλων του Διαδικτύου. Ένα από τα σημαντικότερα κομμάτια μιας υπηρεσίας Διαδικτύου είναι η ανεξαρτησία που παρέχει σε τρίτες εφαρμογές λογισμικού να επικοινωνούν μεταξύ τους ανεξαρτήτως πλατφόρμας ή γλώσσας προγραμματισμού.

3.2 Μέθοδοι πρόσβασης

Για να μπορέσει ένα Web Service να καταστεί λειτουργικό ως προς τον σκοπό του πρέπει να τηρούνται κάποιοι κανόνες. Με αυτή την λογική δημιουργήθηκε ένα πρωτόκολλο το οποίο οριοθέτησε τον τρόπο πρόσβασης από και προς το Web Service επιτρέποντας έτσι την χωρίς λάθος ανταλλαγή μηνυμάτων από και προς το Web Service διατηρώντας έτσι έναν ομαλό κύκλο λειτουργίας. Το πρωτόκολλο αυτό ονομάστηκε *Simple Object Access Protocol (SOAP)* και έχει ως μέσο πρόσβασης την ανταλλαγή XML αντικειμένων. Με την πάροδο των χρόνων έγινε σαφές ότι αυτή η μέθοδος είναι αρκετά απαιτητική τόσο από άποψη πόρων όσο και από άποψη πολυπλοκότητας. Για τον λόγο

αυτό χρησιμοποιήθηκε ένας απλουστευμένος τρόπος πρόσβασης ο οποίος ονομάστηκε *Representational State Transfer (REST).*

3.2.1 Simple Object Access Protocol

Ο ορισμός για αυτό το πρωτόκολλο που δίνεται από το W3C καλύπτει πλήρως την έννοια του. "Το SOAP είναι ένα ελαφρύ πρωτόκολλο προορισμένο για την ανταλλαγή δομημένων πληροφοριών σε ένα αποκεντρωμένο, διανεμημένο περιβάλλον. Χρησιμοποιεί τεχνολογίες XML για να καθορίσει ένα επεκτάσιμο πλαίσιο παρέχοντας μια δομή μηνυμάτων η οποία μπορεί να ανταλλαχτεί πάνω από ποικίλα δικτυακά πρωτόκολλα. Το πλαίσιο έχει σχεδιαστεί να είναι ανεξάρτητο από οποιοδήποτε προγραμματιστικό μοντέλο και σημασιολογία υλοποίησης". Σκοπός για την δημιουργία του ήταν αν και κατά πόσο είναι εφικτή η επικοινωνία μεταξύ των εφαρμογών μέσω κλήσεων απομακρυσμένων διαδικασιών (Remote Procedure Calls-RPCs), χρησιμοποιώντας απλά πρωτόκολλα δικτύου, όπως το HTTP. Αποτελεί λοιπόν ένα πρωτόκολλο, που ορίζει με σαφήνεια την ανταλλαγή δομημένων πληροφοριών αλλά ταυτόχρονα αποτελεί ένα σύνολο συμβάσεων που ορίζουν και τις αλληλεπιδράσεις των κόμβων SOAP , οι οποίοι επεξεργάζονται τα μηνύματα κατά μήκος της διαδρομής τους και τα μέρη που το αποτελούν είναι:

- ο φάκελος, που χρησιμοποιείται για την περιγραφή του περιεχομένου ενός μηνύματος και μερικών ενδείξεων σχετικά με το πώς να το επεξεργαστεί
- οι κανόνες κωδικοποίησης των δομημένων δεδομένων
- μια σύμβαση για την σωστή αναπαράσταση κλήσεων διαδικασίας και απαντήσεων σε αυτές

3.2.1.1 Μήνυμα SOAP

Όπως αναφέρθηκε και παραπάνω τα μηνύματα μεταξύ των SOAP υπηρεσιών αποστέλλονται ως XML έγγραφα. Η δομή αυτών των εγγράφων θα μπορούσε να παρομοιαστεί με την απλή δομή των γραμμάτων που ταχυδρομούνται στην πραγματική ζωή. Έτσι λοιπόν τα στοιχεία που το αποτελούν είναι:

- Φάκελος (Envelope)

Το συγκεκριμένο στοιχείο είναι υποχρεωτικό στοιχείο του μηνύματος καθώς είναι αυτό που προσδιορίζει ότι το συγκεκριμένο XML έγγραφο είναι μήνυμα SOAP. Αποτελεί λοιπόν στοιχείο-ρίζα του μηνύματος καθώς όλα τα υπόλοιπα στοιχεία περικλείονται σε αυτό. Απαραίτητα χαρακτηριστικά αυτού του στοιχείου είναι τα εξής:

1. Το όνομα του στοιχείου πρέπει να είναι Envelope
2. Το όνομα του namespace του να είναι: http://www.w3.org/2003/05/soap-envelope
3. Να περιέχει μηδέν ή περισσότερες ιδιότητες ορισμένες με namespace.
4. Να περιέχει ένα ή δυο παιδιά-στοιχεία με την εξής σειρά:
    a. Προαιρετικό στοιχείο Header,
    b. Υποχρεωτικό στοιχείο Body

- Επικεφαλίδα (Header)

  Με την χρήση αυτής της επικεφαλίδας παρέχεται ένας μηχανισμός επέκτασης ενός μηνύματος γι' αυτό αποτελεί προαιρετικό στοιχείο. Κάθε στοιχείο Envelope μπορεί να περιέχει περισσότερα από ένα στοιχεία Headers, τα οποία είναι συνδεδεμένα μαζί με το μήνυμα και μπορούν να γίνουν στόχος για επέκταση σε συγκεκριμένους κόμβους του δικτύου.

- Σώμα (Body)

  Αποτελεί υποχρεωτικό στοιχείο του μηνύματος και αυτό είναι που περιέχει όλη την πληροφορία που θα σταλθεί από τον παραλήπτη στον αποστολέα. Απαραίτητα χαρακτηριστικά αυτού του είναι τα εξής:

  1. Το όνομα του στοιχείου πρέπει να είναι Body
  2. Το όνομα του namespace του να είναι: http://www.w3.org/2003/05/soap-envelope
  3. Να περιέχει μηδέν ή περισσότερες ιδιότητες ορισμένες με namespace.

4. Μπορεί να περιέχει μηδέν ή περισσότερα στοιχεία-παιδιά.
5. Μπορεί να περιέχει μηδέν ή περισσότερους κόμβους-παιδιά χαρακτήρων.

Τέλος, ένα από τα σημαντικότερα χαρακτηριστικά του είναι ο εσωτερικός μηχανισμός για την διαχείριση σφαλμάτων. Αυτό είναι ιδιαίτερα σημαντικό αν αναλογιστεί κανείς ότι η πλειοψηφία των Web Service που χρησιμοποιούνται είναι κατασκευασμένα από τρίτους. Έτσι εάν υπάρξει πρόβλημα με το αίτημα που στάλθηκε, η απάντηση θα περιέχει πληροφορίες για το τι προξένησε το πρόβλημα.

3.2.2 Representational State Transfer

Για να αποφευχθεί η πολυπλοκότητα που εμφανίζεται με την χρήση SOAP μεθόδων δημιουργήθηκε ένας άλλος τρόπος επικοινωνίας από και προς το Web Service. Έτσι λοιπόν, αν και ξεφεύγει από την τυποποίηση ενός πρωτοκόλλου και πλησιάζει περισσότερο σε μια διαφορετικού τύπου αρχιτεκτονική, έχει ξεκινήσει να αποτελεί έναν από τους σημαντικότερους τρόπους πρόσβασης σε Web Services. Όλα τα δεδομένα και η λειτουργικότητα μεταξύ πελάτη και service θεωρούνται πόροι και είναι προσβάσιμα μέσω απλών Uniform Resource Identifiers (URIs) δηλαδή απλών συνδέσμων. Οι αιτήσεις που γίνονται σε αυτά τα URI συνήθως επιστρέφουν ως απάντηση ένα XML, HTML, JSON περιεχόμενο. Τέλος κάνοντας χρήση ενός stateless πρωτοκόλλου και χρησιμοποιώντας τυποποιημένες λειτουργίες, όπως θα δούμε παρακάτω, το Web Service στοχεύει σε υψηλές αποδόσεις, και την δυνατότητα επέκτασης με την επαναχρησιμοποίηση συστατικών τα οποία είναι διαχειρίσημα χωρίς να επηρεάζουν ολόκληρο το σύστημα.

Για να θεωρηθεί μια υπηρεσία RESTful πρέπει να ακολουθούνται τα εξής χαρακτηριστικά:

- Η κατάσταση και η λειτουργικότητα, χωρίζονται σε διανεμημένους πόρους.

- Κάθε πόρος είναι μοναδικά προσβάσιμος χρησιμοποιώντας μια τυποποίηση και ένα σύνολο εντολών. Όπως αναφέρθηκε παραπάνω αυτές είναι οι HTTP εντολές GET, PUT, PUSH, DELETE.
- Η αρχιτεκτονική αυτή συμβαίνει μεταξύ διακομιστή/πελάτη, είναι stateless, είναι πολυεπίπεδη και υποστηρίζει αποθήκευση στην κρυφή μνήμη.

3.2.2.1 Βασικά στοιχεία αρχιτεκτονικής

Μια ReST υλοποίηση διακρίνεται σε τρία βασικά αρχιτεκτονικά στοιχεία:

- Στοιχεία δεδομένων (Data elements)
  το κυριότερο στοιχείο μιας τέτοιας αρχιτεκτονικής είναι η κατάσταση των δεδομένων. Τα επιμέρους μέρη επικοινωνούν μεταφέροντας αναπαραστάσεις από το τωρινό ή το επιθυμητό στοιχείο δεδομένων.
- Συνδέσεις (Connectors)
  είναι ένας αφηρημένος μηχανισμός που μεσολαβεί στην επικοινωνία, το συντονισμό και τη συνεργασία μεταξύ των επιμέρους συστατικών ενός service.
- Στοιχεία (Components)
  τα ReST στοιχεία αναγνωρίζονται από τον ρόλο τους μέσα στην εφαρμογή. Αποτελούν μια αφηρημένη μονάδα οδηγιών λογισμικού.

3.2.2.2 Αρχιτεκτονικοί περιορισμοί

Υπάρχουν έξι περιορισμοί / οδηγίες οι οποίες καθορίζουν ένα ReSTful σύστημα. Αυτοί οι περιορισμοί περιορίζουν τον τρόπο με τον οποίο ένας διακομιστής θα επεξεργαστεί και θα απαντήσει σε ένα αίτημα πελάτη δίνοντάς του έτσι τις επιθυμητές μη – λειτουργικές ιδιότητες όπως, η απόδοση, η επεκτασιμότητα, η δυνατότητα τροποποίησης, η μεταφερσιμότητα και η αξιοπιστία. Σε περίπτωση που υπάρχει παραβίαση μιας ιδιότητας τότε το service δεν μπορεί να θεωρηθεί ReSTful. Έτσι λοιπόν έχουμε τους εξής περιορισμούς:

- Client-server

  Εδώ το κλειδί πίσω από αυτό τον περιορισμό είναι ο διαχωρισμός τελικού χρήστη και διακομιστή. Έτσι λοιπόν, διαχωρίζοντας την διεπαφή χρήστη από την αποθήκευση δεδομένων βελτιώνεται η μεταφερσιμότητα της διεπαφής χρήστη σε πολλαπλές και διαφορετικές πλατφόρμες καθώς επίσης βελτιώνεται και η επεκτασιμότητα εξαιτίας του ότι απλοποιούνται αρκετά συστατικά του διακομιστή. Καταλήγουμε στο ότι διακομιστές και πελάτες μπορούν να αναπτυχθούν ανεξάρτητα, εφ' όσον η διασύνδεση μεταξύ τους δεν μεταβάλλεται.
- Stateless

  η επικοινωνία μεταξύ πελάτη / διακομιστή περιορίζεται εξαιτίας του ότι δεν αποθηκεύεται καμία πληροφορία για τον πελάτη μεταξύ των αιτημάτων στον διακομιστή. Κάθε αίτημα από οποιονδήποτε πελάτη περιέχει όλες τις απαραίτητες πληροφορίες για να παραδοθεί το αίτημα και ταυτόχρονα η κατάσταση του session αποθηκεύεται στον πελάτη. Υπάρχει η δυνατότητα αποθήκευσης του session μέσω ενός άλλου service σε μια βάση δεδομένων με σκοπό την μακροβιότερη πιστοποίηση του πελάτη. Ο πελάτης αρχίζει να στέλνει αιτήματα όταν είναι έτοιμος να μεταβεί σε μια άλλη κατάσταση. Τέλος λοιπόν, η αναπαράσταση της κατάστασης κάθε εφαρμογής περιέχει συνδέσμους οι οποίοι μπορούν να χρησιμοποιηθούν την επόμενη φορά που ο πελάτης θα επιλέξει να ξεκινήσει μια καινούργια μεταβατική κατάσταση.
- Cacheable

  Η αποθήκευση στην κρυφή μνήμη δεν αποτελεί καινούργια τεχνολογία, εφαρμόζεται στο World Wide Web εδώ και αρκετό καιρό δίνοντας την δυνατότητα, μέσω σωστής διαχείρισης της αποθήκευσης στην κρυφή μνήμη, να εκμηδενιστούν ή να εξαφανιστούν τελείως οποιεσδήποτε αλληλεπιδράσεις μεταξύ πελάτη/διακομιστή. Όπως και εκεί λοιπόν έτσι και εδώ οι πελάτες όπως και άλλοι μεσάζοντες

έχουν την δυνατότητα να αποθηκεύσουν απαντήσεις στην κρυφή μνήμη βελτιώνοντας έτσι την επεκτασιμότητα και την απόδοση του συστήματος. Από τα παραπάνω λοιπόν προκύπτει η ανάγκη προσδιορισμού των απαντήσεων ως cachable ή όχι. Έτσι θα αποτραπούν οι πελάτες από την επαναχρησιμοποίηση παλιών ή ακατάλληλων δεδομένων σε περεταίρω αιτήματα.

- Layered system

  Υπό κανονικές συνθήκες ένας πελάτης δεν μπορεί να ξεχωρίσει εάν είναι συνδεδεμένος απευθείας σε ένα διακομιστή ή σε κάποιον μεσάζοντα κατά την διαδρομή προς τον διακομιστή. Προκύπτει λοιπόν ότι χρησιμοποιώντας μεσάζοντες (layers) θα βελτιωθεί η επεκτασιμότητα με την χρήση εργαλείων ελέγχου φόρτου καθώς επίσης θα μπορεί να χρησιμοποιηθεί κοινή δημόσια κρυφή μνήμη.

- Code on demand

  αποτελεί τον μοναδικό μη υποχρεωτικό περιορισμό αυτής της αρχιτεκτονικής. Οι διακομιστές έχουν την δυνατότητα, προσωρινά, να επεκτείνουν ή να παραμετροποιήσουν την λειτουργικότητα ενός πελάτη μεταφέροντας κομμάτια εκτελέσιμου κώδικα. Κλασσικό παράδειγμα αποτελούν οι δέσμες ενεργειών πελάτη σε Javascript όπως επίσης και συστατικά σαν τα Java applets.

- Uniform interface

  αποτελεί έναν από τους σημαντικότερους περιορισμούς/οδηγίες για μια ReST υλοποίηση. Αυτό συμβαίνει γιατί απλοποιεί και αποσυνδέει την αρχιτεκτονική δίνοντας την δυνατότητα σε κάθε μέρος να εξελιχθεί ανεξάρτητα. Αυτή η οδηγία αποτελείται από τέσσερις υπό περιορισμούς τους οποίους αναλύοντας τους θα αντιληφθούμε την σημαντικότητα αυτής της οδηγίας. Έτσι λοιπόν έχουμε:

    1. Προσδιορισμός των πόρων

επιμέρους πόροι προσδιορίζονται στα αιτήματα όπως πχ τα επιμέρους URI μιας ReST υπηρεσίας. Οι συγκεκριμένοι πόροι είναι εννοιολογικά ξεχωριστοί από τις αναπαραστάσεις που επιστρέφονται στον πελάτη. Συνεπώς για παράδειγμα, ο διακομιστής μπορεί να αποστείλει δεδομένα από μια βάση ως HTML, XML, JSON, αλλά καμία από αυτές δεν θα αποτελεί την εσωτερική αναπαράσταση του διακομιστή.

2. Χειρισμός πόρων μέσω αναπαραστάσεων
   όταν ο πελάτης έχει στην κατοχή του μια αναπαράσταση ενός πόρου, και όποια μετα-δεδομένα επισυνάπτονται σε αυτόν, τότε έχει αρκετές πληροφορίες για να τροποποιήσει ή να διαγράψει αυτόν τον πόρο.
3. Αυτοπεριγραφικά μηνύματα
   κάθε μήνυμα εμπεριέχει αρκετές πληροφορίες για να αυτοπροσδιορίσει πως να περιγράφει. Για παράδειγμα, ποιος αναλυτής να οριστεί μπορεί να προσδιοριστεί από έναν τύπο μέσων διαδικτύου.
4. Υπερμέσα ως η κινητήρια δύναμη της κατάστασης της εφαρμογής
   με το που υπάρξει επίσκεψη σε ένα URI μιας ReST υπηρεσίας θα πρέπει ο πελάτης που την επισκέπτεται να μπορεί μετέπειτα να χρησιμοποιήσει συνδέσμους που παρέχονται από τον διακομιστή για να ανακαλύψει όλες τις διαθέσιμες ενέργειες και πόρους που χρειάζεται. Όσο η πρόσβαση συνεχίζεται, ο διακομιστής απαντάει με κείμενο το οποίο περιλαμβάνει υπερ-συνδέσμους προς άλλες ενέργειες που παρέχονται. Από τα παραπάνω προκύπτει εύλογα το συμπέρασμα ότι δεν υπάρχει η ανάγκη ο πελάτης να έχει ενσωμάτωση στον βασικό

του κώδικα στοιχεία, κατασκευαστικά ή λειτουργικά, του web service.

3.2.2.3 Επικοινωνία με την ReST υπηρεσία

Για να μπορέσει λοιπόν μια υπηρεσία τέτοιου τύπου να χαρακτηριστεί ως RESTful API και ως συνέχεια να μπορούν να μεταφερθούν επιτυχώς μηνύματα από και προς την ReST υπηρεσία θα πρέπει αρχικά να υπάρξει το αντίστοιχο αίτημα στο HTTP πρωτόκολλο το οποίο είναι αυτό που χρησιμοποιείται και πάνω σε αυτό είναι βασισμένη αυτή η αρχιτεκτονική. Ανάλογα με τον τύπο του αιτήματος έχουμε τις εξής μεθόδους που περιγράφουν την κατάλληλη ενέργεια που πρέπει να γίνει σε ένα πόρο:

1. GET

   με την χρήση αυτής της μεθόδου ζητείται μια αναπαράσταση ενός συγκεκριμένου πόρου. Αιτήματα GET πρέπει αποκλειστικά και μόνο να ανακτούν δεδομένα.
2. POST

   με την χρήση αυτής της μεθόδου ζητείται από τον διακομιστή να δεχτεί την οντότητα που βρίσκεται εγκλεισμένη στο αίτημα ως υπό πόρο του πόρου που ζητήθηκε αρχικά μέσω του URI.
3. PUT

   με την χρήση αυτής της μεθόδου ζητείται η εγκλεισμένη στο αίτημα οντότητα να αποθηκευτεί κάτω από το ζητούμενο URI. Αν το URI αναφέρεται σε υπάρχων πόρο τότε αυτός τροποποιείται ενώ αν δεν υπάρχει τότε δημιουργείται από το διακομιστή με το URI που ζητήθηκε.
4. DELETE

   με την χρήση αυτής της μεθόδου ζητείται η διαγραφή του πόρου.

Παρακάτω λοιπόν δίνεται μια αντιστοιχία μεταξύ των μεθόδων αυτών και πως χρησιμοποιούνται έμπρακτα σε ένα ReST API

Uniform Resource Locator (URL)	GET	PUT	POST	DELETE
Collection, such as `http://api.example.com/resources/`	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation. [17]	**Delete** the entire collection.
Element, such as `http://api.example.com/resources/item17`	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it does not exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry within it.[17]	**Delete** the addressed member of the collection.

Πίνακας 2 – Μέθοδοι πρόσβασης μέσω URI

Υπάρχουν αρκετές μορφοποιήσεις για την αποστολή μηνυμάτων μεταξύ ReST υπηρεσιών. Αυτό αποτελεί και το μεγάλο πλεονέκτημα της χρήσης αυτής της αρχιτεκτονικής. Η ελαστικότητα στην επιλογή της

μορφής του μηνύματος βοηθάει τους προγραμματιστές τέτοιων υπηρεσιών δίνοντάς τους τα εργαλεία για ένα ανοιχτού τύπου σύστημα το οποίο ταυτόχρονα μπορεί να χαρακτηριστεί αποδοτικό και ευέλικτο μιας και δεν εμφανίζεται πολυπλοκότητα για την αποστολή και λήψη απλών δεδομένων. Παρακάτω θα αναλύσουμε έναν μια από τις πιο διαδεδομένες μορφοποιήσεις

1. JavaScript Object Notation (JSON)

   όπως αναφέρθηκε παραπάνω όταν αποστέλλονται πληροφορίες μεταξύ πελάτη και διακομιστή αυτές μπορεί να είναι μόνο κείμενο. Έτσι λοιπόν χρησιμοποιείται το JSON το οποίο αποτελεί ένα αρχείο ανοιχτού προτύπου το οποίο χρησιμοποιεί κείμενο κατανοητό από τον άνθρωπο για την μεταφορά δεδομένων σε αντικείμενα τα οποία αποτελούνται από ζεύγη χαρακτηριστικών-τιμών και τύπους δεδομένων σε συστοιχίες. Από την στιγμή που ο τύπος αρχείου αυτός χρησιμοποιεί την Javascript τότε είναι εύκολα κατανοητό ότι μπορούμε να μετατρέψουμε οποιοδήποτε αντικείμενο σε Javascript σε JSON μορφή και να το αποστείλουμε στον διακομιστή και το αντίστροφο. Με αυτό τον τρόπο μειώνεται ο φόρτος στον διακομιστή και συγχρόνως κάνει τον προγραμματισμό πιο εύκολο εξαιτίας του ότι δεν χρειάζεται κάποια εξεζητημένη ανάλυση ή μετάφραση του κώδικα. Για να μπορέσουμε να αντιληφθούμε καλύτερα την ευχρηστία και την απλότητα αυτής της μορφοποίησης θα παραθέσουμε μια μικρή ανάλυση των δομών δεδομένων που την αποτελούν καθώς και ένα μικρό κομμάτι κώδικα:

   a. Number

      αποτελεί ένα δεκαδικό αριθμό που μπορεί να περιέχει ένα κλασματικό μέρος και μπορεί να

χρησιμοποιεί εκθετική σημείωση e, αλλά δεν μπορεί να περιλαμβάνει μη αριθμούς όπως το NaN. Δεν γίνεται διάκριση μεταξύ ακέραιου και κυμαινόμενου σημείου. Η JavaScript χρησιμοποιεί μια μορφή κυμαινόμενης θέσης διπλής ακρίβειας για όλες τις αριθμητικές τιμές της.

b. String:
   μια ακολουθία μηδενικών ή περισσότερων χαρακτήρων Unicode.
c. Boolean:
   δέχεται μία από τις τιμές true ή false.
d. Array
   ένας ταξινομημένος κατάλογος μηδενικών ή περισσοτέρων τιμών, καθεμία από τις οποίες μπορεί να είναι οποιουδήποτε τύπου.
e. Object
   μια μη ταξινομημένη συλλογή ζευγών ονόματος / τιμής όπου τα ονόματα (που ονομάζονται επίσης κλειδιά) είναι συμβολοσειρές.
f. Null
   μια κενή τιμή, χρησιμοποιώντας τη λέξη null.

Το παρακάτω παράδειγμα δείχνει μια πιθανή αναπαράσταση JSON που περιγράφει ένα άτομο.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
```

```
      "postalCode": "10021-3100"
    },
    "phoneNumbers": [
      {
        "type": "home",
        "number": "212 555-1234"
      },
      {
        "type": "office",
        "number": "646 555-4567"
      },
      {
        "type": "mobile",
        "number": "123 456-7890"
      }
    ],
    "children": [],
    "spouse": null
}
```


3.2.3 Μειονεκτήματα / Πλεονεκτήματα SOAP και ReST

Με την βοήθεια μιας μικρής σύγκρισης μεταξύ των δυο θα γίνει ευκολότερα αντιληπτή η χρησιμότητα τους αλλά ταυτόχρονα θα μας βοηθήσει να αποφασίσουμε και με ποιο από τα δυο θα ασχοληθούμε και για ποιον λόγο.

- Το SOAP είναι ένα πρωτόκολλο. Το REST είναι αρχιτεκτονικό στυλ. Ένα API έχει σχεδιαστεί για να εκθέτει ορισμένες πτυχές της επιχειρησιακής λογικής μιας εφαρμογής σε ένα διακομιστή. Το SOAP χρησιμοποιεί μια διεπαφή υπηρεσίας για να το κάνει αυτό ενώ το REST χρησιμοποιεί URIs.
- Τα REST API έχουν πρόσβαση σε έναν πόρο για δεδομένα (URI). Τα SOAP API εκτελούν μια ενέργεια.
- Η διαχείριση της ασφάλειας είναι διαφορετική. Το SOAP υποστηρίζει το WS-Security, το οποίο είναι εξαιρετικό στο επίπεδο των μεταφορών και είναι λίγο πιο ολοκληρωμένο από το SSL και πιο ιδανικό για την ενσωμάτωση σε εργαλεία ασφάλειας σε επιχειρήσεις. Και οι δύο υποστηρίζουν το SSL για ασφάλεια από άκρο σε άκρο και το REST μπορεί να

χρησιμοποιήσει την ασφαλή έκδοση του πρωτοκόλλου HTTP το HTTPS.

- Οι κλήσεις REST μπορούν να αποθηκευτούν προσωρινά, οι κλήσεις με βάση το SOAP δεν μπορούν να αποθηκευτούν προσωρινά. Τα δεδομένα μπορούν να επισημανθούν ως αποθηκευμένα στο αρχείο, πράγμα που σημαίνει ότι μπορεί να ξαναχρησιμοποιηθούν από το πρόγραμμα περιήγησης αργότερα χωρίς να χρειάζεται να ξεκινήσει ένα άλλο αίτημα πίσω στο διακομιστή.

3.3 Τρόπος λειτουργίας και δυνατότητες

Με την χρήση των Web services δίνεται η δυνατότητα σε χρήστες και επιχειρήσεις να έχουν πρόσβαση σε ένα πλήθος επιλογών και λειτουργιών η αξιοποίηση των οποίων θα βελτιστοποιήσει τις υπηρεσίες που παρέχονται. Μπορούμε λοιπόν να σκεφτούμε τα Web services σαν ένα κομμάτι λογισμικού που πραγματοποιούν μια συγκεκριμένη λειτουργία η οποία γίνεται διαθέσιμη για αξιοποίηση μέσω ενός συνόλου μεθόδων που μπορούν να πραγματοποιηθούν. Επιπλέον, κάθε μέθοδος διαθέτει ένα σύνολο μεταβλητών / παραμέτρων που μπορούν να δεχθούν κάποια δεδομένα που ζητούνται από τις μεθόδους.

Εξαιτίας του ότι διαφορετικές εφαρμογές είναι γραμμένες σε διαφορετικές γλώσσες προγραμματισμού είναι συχνό φαινόμενο να μην μπορούν να επικοινωνήσουν μεταξύ τους. Έτσι λοιπόν με την χρήση των Web services αυτή η επικοινωνία μπορεί να επιτευχθεί με την χρήση ανοιχτών τυποποιημένων πρωτοκόλλων όπως XML, SOAP και WSDL. Η XML χρησιμοποιείται για την σήμανση των δεδομένων, το SOAP για την μεταφορά μηνυμάτων και η WSDL για να περιγραφεί η διαθεσιμότητα της υπηρεσίας.

Έτσι, για παράδειγμα, μια επιχείρηση Α δημιουργεί ένα Web service που παρέχει ως βασική λειτουργία την εύρεση της ισοτιμίας νομίσματος χρησιμοποιώντας την μέθοδο με όνομα GetRate. Μια επιχείρηση Β είναι σε θέση να στείλει μια παράμετρο με όνομα CountryCode, η οποία περιέχει τον κωδικό της χώρας, στην μέθοδο GetRate. Η μέθοδος με την σειρά της δέχεται την παράμετρο και αφού αναζητήσει την ισοτιμία νομίσματος από την βάση δεδομένων την επιστρέφει στο πρόγραμμα το οποίο την ζήτησε.

3.4 Πλεονεκτήματα και μειονεκτήματα των Web services

Πλεονεκτήματα

- **Διαλειτουργικότητα:** συνήθως εκτελούνται εκτός ιδιωτικών δικτύων δίνοντας έτσι στους προγραμματιστές κοινόχρηστες δομές προς χρήση. Έτσι είναι πιθανόν να έχουν μεγαλύτερο κύκλο ζωής προσφέροντας έτσι καλύτερη απόδοση για την επένδυση της αναπτυγμένης υπηρεσίας. Τέλος δίνεται λοιπόν η δυνατότητα χρήσης πολλών διαφορετικών γλωσσών προγραμματισμού καθώς επίσης προσφέρεται και ανεξαρτησία από οποιαδήποτε πλατφόρμα εξαιτίας της χρήσης ανοιχτών πρωτοκόλλων επικοινωνίας.
- **Ευχρηστία / Επαναχρησιμοποίηση:** επιτρέπουν την χρήση επιχειρησιακής λογικής πολλών διαφορετικών συστημάτων δίνοντας έτσι την δυνατότητα αξιοποίησης, για την υπό ανάπτυξη εφαρμογή, της υπηρεσίας Web που έχουν ανάγκη. Συνεπώς αφού δεν υπάρχει δέσμευση για το τι θα χρησιμοποιηθεί καθίσταται εύκολη η επαναχρησιμοποίηση συστατικών Web service ανάλογα με την περίπτωση και σε άλλες υπηρεσίες.
- **Δυνατότητα ανάπτυξης:** τα Web services αναπτύσσονται πάνω σε τυποποιημένες τεχνολογίες Διαδικτύου. Έτσι δίνεται η δυνατότητα να αναπτυχθούν Web services ακόμα και πάνω από κάποιο firewall ενός εξυπηρετητή. Τέλος επειδή χρησιμοποιούνται συγκεκριμένες κοινοτικές τυποποιήσεις η ασφάλεια επιπέδου SSL είναι ήδη ενσωματωμένη.

Μειονεκτήματα

- **Απλότητα:** παρόλο που η απλότητα πολλές φορές είναι κάτι θετικό αρκετές μπορεί να αποτελεί εμπόδιο. Στην συγκεκριμένη περίπτωση αυτό συμβαίνει γιατί τα Web services χρησιμοποιούν πρωτόκολλα απλού κειμένου που χρησιμοποιούν μια αρκετά λεπτομερή μέθοδο για τον εντοπισμό των δεδομένων. Με αυτό τον τρόπο όμως τα αιτήματα που γίνονται σε ένα Web service είναι αισθητά μεγαλύτερα από αυτά που π.χ. έχουν κωδικοποιηθεί δυαδικά.
- **HTTP / HTTPS πρωτόκολλα:** ενώ είναι αρκετά απλά στην υλοποίηση τους δεν είναι σχεδιασμένα για συνεδρίες μεγάλης διάρκειας εξαιτίας του

ότι γίνεται συνήθως αποσύνδεση του πελάτη σε μικρό χρονικό διάστημα. Επίσης ένα πρόβλημα είναι ότι τα πρωτόκολλα αυτά δεν μπορούν να στείλουν σήματα κατάστασης, έτσι όταν δεν υπάρχουν δεδομένα προς αποστολή ο εξυπηρετητής και ο πελάτης δεν έχουν γνώση ο ένας για τον άλλο.

ΕΠΙΛΟΓΟΣ

Συνοψίζοντας, σε αυτό το κεφάλαιο γίνεται προσπάθεια κάλυψης του θέματος των υπηρεσιών Διαδικτύου (Web services) δίνοντας έναν ορισμό και άρα κάποια σαφή όρια που το χαρακτηρίζουν. Έγινε μια εκτενής ανάλυση στις μεθόδους και τους τρόπους πρόσβασης σε τέτοιες υπηρεσίες ενώ συγχρόνως έγινε σύγκριση κάποιων βασικών μεθόδων με σκοπό την αιτιολόγηση της χρήσης της ReST υπηρεσίας από αυτή μας την εργασία. Περαιτέρω παρουσιάστηκαν οι δυνατότητές τους σε βάθος χρόνου με τις εξελίξεις της τεχνολογίας και τις βελτιώσεις που επήλθαν και επηρέασαν αυτές τις υπηρεσίες. Ταυτόχρονα δόθηκε χαρακτηριστικό παράδειγμα του τρόπου λειτουργίας ενός τέτοιου συνόλου άρα πως αυτό μας βοηθά στον συνδυασμό διαφορετικών τεχνολογιών και μέσων.

ΚΕΦΑΛΑΙΟ 4

Hybrid application development

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο αρχικά θα αναφερθούμε στους τρόπους και στις τεχνολογίες ανάπτυξης εφαρμογών για κινητά τηλέφωνα (mobile apps). Επίσης θα κάνουμε κάποιες συγκρίσεις πάνω σε αυτές τις τεχνολογίες. Τέλος θα αναλύσουμε εκτενέστερα κάποιες από τις παραπάνω τεχνολογίες οι οποίες και χρησιμοποιήθηκαν για να υλοποιηθεί η εφαρμογή μας.

4.1 Τρόποι ανάπτυξης mobile apps

Σε γενικές γραμμές υπάρχουν τρεις προσεγγίσεις για ανάπτυξη λογισμικού που στοχεύουν στα κινητά. Η μία από αυτές είναι ο σχεδιασμός εφαρμογών που θα εκτελούνται σε browser κινητών(Mobile web-apps) ενώ οι άλλες δύο προσεγγίσεις αφορούν την δημιουργία αυτόνομων εφαρμογών(Native apps, Hybrid apps).

4.1.1 Mobile web apps

Εύκολα καταλαβαίνουμε ότι οι ανάγκες των χρηστών κινητών τηλεφώνων είναι διαφορετικές από αυτών που χρησιμοποιούν του ηλεκτρονικούς υπολογιστές. Μια ιστοσελίδα που είναι σχεδιασμένη για να εμφανίζεται αποκλειστικά σε οθόνη υπολογιστή, όταν χρειάζεται να εμφανιστή στο browser ενός smartphone πολλές φορές μπορεί να είναι αρκετά δύσχρηστη.

Γι’ τον παραπάνω λόγο υπάρχει η προσέγγιση οι διαδικτυακές εφαρμογές να έχουν και κάποιο ειδικό σχεδιασμό για τους browser των smartphone και tablet. Αυτό παλαιότερα το πετυχαίναμε με δημιουργία mobile έκδοση της διαδικτυακής εφαρμογής, όπου όταν ο χρήστης έμπαινε από κινητό, η εφαρμογή του εμφάνιζε την ιδική αυτή σχεδίαση. Πλέον που έχει αναπτυχθεί ιδιαίτερα αυτός ο τομέας το οι διαδικτυακές εφαρμογές σχεδιάζονται με responsive design ώστε η ίδια εφαρμογή να είναι εύκολα προσαρμόσιμή σε όλα τα μεγέθη οθονών.

Οι παραπάνω εφαρμογές σχεδιάζονται με σκοπό να εκτελούνται αποκλειστικά σε browser, όπου πλέον οι browser των smartphone έχουν

πολλές δυνατότητες. Επίσης οι συγκεκριμένες εφαρμογές αναπτύσσονται αποκλειστικά με γλώσσες προγραμματισμού client-side(HTML, CSS, JS).

4.1.2 Native apps

Σε αυτή την κατηγορία ανήκουν οι εφαρμογές που αναπτύσσονται για να εκτελούνται τελείως αυτόνομα στα smartphones, επικοινωνώντας κατευθείαν με το λειτουργικό του σύστημα. Οι χρήστες κατεβάζουν τα εκτελέσιμα αρχεία από το store και τα αποθηκεύουν τοπικά. Η εφαρμογή εγκαθίσταστε στο κινητό και έχει τη δυνατότητα εκτελείτε σαν service του κινητού αλλά και να επικοινωνεί με τα υπόλοιπα services που παρέχει το λειτουργικό σύστημα.

Σε αυτή την προσέγγιση καθώς υπάρχουν διάφορα λειτουργικά συστήματα κινητών, (αναφέρουμε ενδεικτικά τα επικρατέστερα: Android, Apple iOS, Blackberry OS, Windows phones) η εφαρμογές θα πρέπει να αναπτύσσονται ξεχωριστά για την κάθε πλατφόρμα. Αυτό είναι αναγκαίο γιατί κάθε πλατφόρμα έχει διαφορετικό τρόπο ανάπτυξης καθώς και οι εφαρμογές αναπτύσσονται με διαφορετικές γλώσσες προγραμματισμού.

	Apple iOS	**Android**	**Blackberry OS**	**Windows Phone**
Γλώσσες	Objective-C, C, C++	Java (some C, C++)	Java	C#, VB.NET and more
Εργαλεία Ανάπτυξης	Xcode	Android SDK	BB Java Eclipse Plug-in	Visual Studio, Windows Phone development tools
Εκτελέσιμο	.app	.apk	.cod	.xap
Store	Apple App Store	Google Play	Blackberry App World	Windows Phone Marketplace

Πίνακας 3 – Γλώσσες προγραμματισμού και πλατφόρμες ανάπτυξης

Όταν αναπτύσσονται native εφαρμογές για smartphones οι προγραμματιστές έχουν τη δυνατότητα να επικοινωνούν κατευθείαν με το λειτουργικό σύστημα του κινητού μέσω τον API's που προσφέρονται από αυτό. Λέγοντας API αναφερόμαστε στους τρόπους διασύνδεσης που δίνει το λειτουργικό σύστημα για την επικοινωνία με τα υποσυστήματα του smartphone(οθόνη, πληκτρολόγιο, κάμερα, GPS, μικρόφωνο ηχεία).

Ένα άλλο σημαντικό κομμάτι στην ανάπτυξη native εφαρμογών είναι το διαφορετικό γραφικό περιβάλλον(GUI) που παρέχει το κάθε λειτουργικό σύστημα. Βλέπουμε ότι κάθε λειτουργικό σύστημα έχει δικό του σετ από κουμπιά, μπάρες, εικονίδια, sliders κ.α. Οπότε κάθε εφαρμογή που πρέπει να εκτελείται σε διαφορετικά λειτουργικά συστήματα θα πρέπει να αναπτύσσεται ξεχωριστά σε αυτά,(στα API που προσφέρει το κάθε λειτουργικό σύστημα) καθώς και να σχεδιάζετε για το κάθε GUI.

Λόγο των παραπάνω καταλαβαίνουμε ότι υπάρχει μεγάλη ευελιξία και πληθώρα δυνατοτήτων στις native εφαρμογές, αλλά θα πρέπει να γίνεται ξεχωριστή αυτόνομη ανάπτυξη στο κάθε λειτουργικό σύστημα.

4.1.3 Hybrid apps

Η υβριδική ανάπτυξη εφαρμογών για smartphones πετυχαίνει τον συνδυασμό των δύο παραπάνω τρόπων ανάπτυξης. Το αποτέλεσμα είναι ένα αυτόνομο app το οποίο ο χρήστης το κατεβάζει από το store και εκτελείτε όπως η εφαρμογή στην Native προσέγγιση, χωρίς όμως αυτό να σημαίνει ότι αυτή η εφαρμογή πρέπει να υλοποιηθεί για την κάθε πλατφόρμα ξεχωριστά.

Στην υβριδική προσέγγιση η υλοποίηση επιτυγχάνεται χρησιμοποιώντας το web view των smartphones. Web view είναι ένα service των smartphones που μπορεί και προβάλει html σε ολόκληρη την οθόνη του κινητού. Μπορούμε να το σκεφτούμε σαν ένα browser ο οποίος είναι τελείως άδειος και απλά προβάλει το html αποτέλεσμα. Έτσι για τη δημιουργία μιας hybrid app η υλοποίηση της βασικής εφαρμογής γίνεται με τεχνολογίες client side προγραμματισμού HTML, JS, CSS και η επικοινωνία της εφαρμογής με τα υπόλοιπα services του κινητού (πχ κάμερα) γίνεται με γεφυρώσεις που υλοποιούνται με native τρόπο.

Η υλοποίηση μπορεί να γίνει σε έναν κανονικό desktop browser, αλλά για να παραχθεί η τελική εφαρμογή χρησιμοποιούμε κάποιες τεχνολογίες οι οποίες μεταφράζουν τον HTM, CSS & JS κώδικα στο τελικό εκτελέσιμο αρχείο της εκάστοτε πλατφόρμας (.apk, .app, .xap κα) το οποίο και διατίθεται στα store από τα οποία ο τελικός χρήστης μπορεί να κατεβάσει την εφαρμογή.

Δύο βασικές ανταγωνιστικές πλατφόρμες που χρησιμοποιούνται για την ανάπτυξη hybrid mobile apps είναι το Cordova (Adobe Phonegap) και το

Appcelerator Titanium. Με το Cordova η υλοποίηση γίνεται, ακολουθώντας την παραπάνω μεθοδολογία και έπειτα το χρησιμοποιούμε για να παράγουμε το τελικό app. Σε παρακάτω κεφάλαιο θα αναλύσουμε λίγο πιο αναλυτικά το Phonegap – διάδοχος του Cordova, καθώς είναι η τεχνολογία που χρησιμοποιήθηκε για να υλοποιηθεί η mobile εφαρμογή μας. Το Appcelerator Titanium λειτουργεί με κάπως διαφορετικό τρόπο για την υλοποίηση των εφαρμογών, χρησιμοποιώντας μια πιο ολοκληρωμένη πλατφόρμα για την υλοποίηση και το τελικό αποτέλεσμα εκτελείτε χρησιμοποιώντας κάποια μηχανή javascript.

Adobe Phonegap / Apache Cordova

Το Apache Cordova και ο διάδοχός του το Adobe Phonegap είναι frameworks με τα οποία μπορούν να αναπτυχθούν υβριδικές mobile εφαρμογές. Το Cordova αναπτύχθηκε από την εταιρία Nitobi αλλά όταν αυτή η εταιρία εξαγοράστηκε από την Adobe, παρέμεινε το Cordova αλλά αναπτύχθηκε παράλληλα και το Phonegap.

Το Adobe Phonegap διαθέτει έναν online compiler στον οποίο οι προγραμματιστές ανεβάζουν τον κώδικα που έχουν αναπτύξει, τα απαραίτητα πιστοποιητικά που χρειάζονται για την κάθε πλατφόρμα και δημιουργούνται από εκεί τα τελικά app από τον compiler. Αυτή η υπηρεσία διατίθεται στη διεύθυνση https://build.phonegap.com/ και υπάρχει δωρεάν πλάνο με μία εφαρμογή και πλάνα επί πληρωμή με περισσότερες εφαρμογές.

Ionic Framework

Το Ionic Framework είναι ένα από τα πλέον διαδεδομένα framework για την ανάπτυξη υβριδικών mobile apps. Είναι λογισμικό ανοιχτού κώδικα (open source) το οποίο βασίζεται στο AngularJs, ένα από τα πιο διάσημα javascript frameworks και στο Cordova/Phonegap. Με το Ionic Framework η ανάπτυξη των εφαρμογών γίνεται με τη χρήση HTML, CSS & Javascript και το τελικό app παράγεται από τους compailer το Apache Cordova ή το Adobe Phonegap.

Ένα από τα βασικά του προτερήματα είναι ότι η αίσθηση της τελικής παραγόμενης εφαρμογής είναι παρόμοια με native.

Το Ionic 1 πρωτοεμφανίστηκε το 2014 με beta έκδοση ενώ η σταθερή έκδοση διατέθηκε το 2015. Το 2016 δόθηκε το Ionic 2 το οποίο διαφέρει αρκετά από το Ionic 1 καθώς βασίστηκε στην καινούρια έκδοση του AngularJS.

Η εγκατάσταση του Ionic γίνεται με κάποια απλά βήματα. Αρχικά κάνουμε εγκατάσταση στο σύστημά μας το node.js, έπειτα με τη χρήση του npm (node.js) εγκαθιστούμε πρώτα το cordova και έπειτα το ionic με τις παρακάτω εντολές:

```
npm install -g cordova
npm install -g ionic
```

Τέλος ξεκινάμε ένα καινούριο project με την παρακάτω εντολή

```
ionic start todo blank --type ionic1
```

4.2 Σύγκριση μεθόδων ανάπτυξης

Για να καταφέρουμε να κάνουμε μία αξιόπιστη σύγκριση μεταξύ των παραπάνω τρόπων ανάπτυξης, θα πρέπει να εστιάσουμε σε κάποιους συγκεκριμένους τομείς για να συγκρίνουμε. Αυτοί οι τομείς θα είναι η απόδοση - δυνατότητες των εφαρμογών, η ταχύτητα υλοποίησης, τα γραφικά της εφαρμογής, δυνατότητα αναβάθμισης & επέκταση εφαρμογής και οι τεχνολογίες – εμπειρία που χρειάζεται για την ανάπτυξη εφαρμογών.

Η σύγκριση θα γίνει μεταξύ της Native & της Hybrid προσέγγισης αφήνοντας απ’ έξω την mobile web app προσέγγιση, γιατί στη συγκεκριμένη στο συγκεκριμένο μέρος της εργασίας μας απασχολούν οι εφαρμογές που εκτελούνται αυτόνομα στη συσκευή.

4.2.1 Απόδοση - Δυνατότητες Εφαρμογής

Μπορούμε εύκολα να βγάλουμε το συμπέρασμα ότι ο native τρόπος υλοποίησης εφαρμογών μπορεί να πετύχει την μέγιστη απόδοση. Αυτό γίνεται γιατί η εφαρμογή επικοινωνία κατευθείαν με το λειτουργικό σύστημα χωρίς κάποια ενδιάμεση γεφύρωση. Επίσης με αυτό τον τρόπο ανάπτυξης οι εφαρμογές έχουν πρόσβαση σε όλα τα service του smartphone (κάμερα, επαφές, GPS κλπ) κατευθείαν.

Με την hybrid υλοποίηση η απόδοση των εφαρμογών είναι κάπως χαμηλότερη από τις native, χωρίς όμως αυτό να είναι ιδιαίτερα εμφανές στον

τελικό χρήστη και ειδικότερα σε εφαρμογές που δεν έχουν μεγάλες απαιτήσεις. Επίσης στις hybrid εφαρμογές υπάρχει η δυνατότητα χρήσης των services του smartphone χρησιμοποιώντας είτε τις δυνατότητες του web view είτε native γέφυρες με τα services.

4.2.2 Ταχύτητα υλοποίησης

Με τη Native προσέγγιση η υλοποίηση θα πρέπει να γίνει για κάθε λειτουργικό σύστημα ξεχωριστά. Ενώ στη hybrid προσέγγιση η βασική υλοποίηση γίνεται μία φορά για όλα τα λειτουργικά συστήματα.

Έτσι βλέπουμε ότι για κάποια εφαρμογή που θα χρειαστούμε να εκτελείτε σε παραπάνω από ένα λειτουργικά συστήματα η hybrid προσέγγιση είναι αρκετά πιο γρήγορη στην υλοποίηση από την native.

4.2.3 Δυνατότητα αναβάθμισης & επέκταση εφαρμογής

Ομοίως με την παραπάνω σύγκριση βλέπουμε ότι για να βγει κάποια ενημέρωση μίας εφαρμογής ή κάποια επέκταση στις λειτουργίες τις, είναι πιο εύκολο να γίνει αυτό στην hybrid προσέγγιση, γιατί στην native θα πρέπει να γίνει υλοποίηση σε κάθε λειτουργικό σύστημα ξεχωριστά.

4.2.4 Γραφικά εφαρμογής

Στη native προσέγγιση χρησιμοποιούμε τις τεχνολογίες και τα διάφορα σχεδιαστικά πρότυπα που δίνει η κάθε πλατφόρμα. Έτσι είναι σχετικά εύκολο να υλοποιηθούν οι εμφανίσεις των εφαρμογών. Επίσης επειδή οι εφαρμογές χρησιμοποιούν τα γραφικά των λειτουργικών συστημάτων έχουν τη μέγιστη απόδοση.

Στην υβριδική προσέγγιση τα γραφικά δημιουργούνται αποκλειστικά με τεχνολογίες web, HTML, CSS, JS που γίνονται rendered στο web view. Για αυτό το λόγο το γραφικό περιβάλλον σε αυτές τις εφαρμογές, έχει χαμηλότερη απόδοση χωρίς αυτό να σημαίνει ότι υπολείπονται κατά πολύ των native εφαρμογών.

4.2.5 Τεχνολογίες – εμπειρία

Στην native προσέγγιση, για κάθε λειτουργικό σύστημα χρειάζεται οι προγραμματιστές να γνωρίζουν την τεχνολογία που χρησιμοποιεί. Οπότε για την υλοποίηση μίας εφαρμογής για 3-4 βασικές πλατφόρμες χρειάζεται οι

προγραμματιστές να γνωρίζουν 4 διαφορετικές γλώσσες προγραμματισμού και αντίστοιχους τρόπους ανάπτυξης για αυτές τις γλώσσες.

Για να ξεκινήσει κάποιος να αναπτύσσει native εφαρμογές, δεν χρειάζεται κάποια εμπειρία σε κάποια άλλη τεχνολογία.

Στην hybrid προσέγγιση χρησιμοποιούνται οι βασικές client side γλώσσες προγραμματισμού (HTML, JS, CSS) για όλες τις πλατφόρμες. Επίσης καθώς έχει εξελιχθεί ο συγκεκριμένος τρόπος ανάπτυξης στις μέρες μας υπάρχουν αρκετά frameworks και βιβλιοθήκες που βοηθούν στην υλοποίηση και βασίζονται σε αυτές τις βασικές τεχνολογίες. Ενδεικτικά αναφέρουμε κάποια από αυτά τα framework Ionic Framework, Sencha Touch, Kendo UI, jQuery mobile.

Σύμφωνα με τα παραπάνω καταλαβαίνουμε ότι για να ξεκινήσει κάποιος με την hybrid ανάπτυξη εφαρμογών θα ήταν καλό να έχει κάποια εμπειρία στην ανάπτυξη client side διαδικτυακών εφαρμογών.

4.2.6 Συγκριτικός πίνακας

	Native	**Hybrid**
Απόδοση	Μέγιστη	Μέτρια
Δυνατότητες	Πλήρης	Πλήρης
Ταχύτητα υλοποίησης	Αργή	Γρήγορη
Αναβαθμίσεις	Δύσκολα	Εύκολα
Γραφικά	Μέγιστα	Μέτρια
Τεχνολογίες	Πολλές	Λίγες
Εμπειρία	Δεν χρειάζεται	Επιθυμητή

Πίνακας 4 – Συγκριτικός πίνακας μεθόδων ανάπτυξης

ΕΠΙΛΟΓΟΣ

Σε αυτό το κεφάλαιο αναλύσαμε τους τρεις τρόπους ανάπτυξης εφαρμογών για κινητά τηλέφωνα. Πιο συγκεκριμένα είδαμε πως γίνεται η ανάπτυξη εφαρμογών με καθεμία από αυτές τις μεθοδολογίες. Αναλύσαμε λίγο περισσότερο το Ionic Framework & το Adobe Phonegap που είναι και οι τεχνολογίες που αναπτύξαμε την εφαρμογή μας. Στο τέλος έγινε σύγκριση σε κάποιους τομής μεταξύ της υβριδικής ανάπτυξης με την native ανάπτυξη. Η σύγκριση έγινε μεταξύ των δύο, καθώς μόνο με αυτές τις δύο μεθοδολογίες μπορούμε να υλοποιήσουμε αυτόνομες εφαρμογές που δεν εξαρτώνται από κάποιο browser.

ΚΕΦΑΛΑΙΟ 5

Βάσεις Δεδομένων – SQL

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο θα κάνουμε μία μικρή αναφορά στις βάσεις δεδομένων μιας και αυτές πλέον είναι αναπόσπαστο κομμάτι των διαδικτυακών εφαρμογών για την αποθήκευση δεδομένων. Επίσης θα γίνει αναφορά και στην MySQL, η βάση δεδομένων που χρησιμοποιεί η δική μας εφαρμογή. Τέλος θα κάνουμε και μία αναφορά και στις βάσεις δεδομένων για τις mobile εφαρμογές.

5.1 Βάσεις δεδομένων

Οι σύγχρονες εφαρμογές χρησιμοποιούν και επεξεργάζονται πληθώρα πληροφοριών σε διάφορες μορφές. Οπότε προκύπτει η ανάγκη για την μετατροπή των πληροφοριών αυτών σε δεδομένα με σκοπό την ομαδοποίηση, για λόγους ταξινόμησης, και την αποθήκευση, για λόγους επανάχρησης, αυτών των δεδομένων. Σε αυτό δίνουν τη λύση οι βάσεις δεδομένων. Με την χρήση των βάσεων δεδομένων λοιπόν, καταφέρνουμε να οργανώσουμε τα δεδομένα σε σειρές, στήλες και πίνακες καθώς επίσης να δημιουργήσουμε και ευρετήρια για τον πιο εύκολο εντοπισμό των δεδομένων προς αναζήτηση.

Εξαιτίας του όγκου των δεδομένων και των μεταξύ τους συσχετίσεων πρέπει να ακολουθηθεί κάποιο μοντέλο, βάση του οποίου τα δεδομένα αυτά θα οργανώνονται και θα κατηγοριοποιούνται με ενιαίο και γενικό τρόπο. Με την πάροδο των χρόνων έχουν αναπτυχθεί αρκετά μοντέλα δεδομένων τα οποία αξιοποιώντας τα μπορούμε να αποδώσουμε μια λογική δομή στις βάσεις δεδομένων καθιστώντας τες έτσι κατανοητές, οργανωμένες και διαχειρίσιμες.

Έτσι λοιπόν το πιο διαδεδομένο, και αυτό στο οποίο στηρίχθηκε αυτή η πτυχιακή, μοντέλο βάσεων δεδομένων είναι οι σχεσιακές βάσεις δεδομένων, με τις αντικειμενοστραφείς βάσεις δεδομένων να είναι το επόμενο μοντέλο που χρησιμοποιείτε περισσότερο.

5.1.1 Σχεσιακές βάσεις δεδομένων & SQL

Οι σχεσιακές βάσεις δεδομένων είναι αυτές οι οποίες στηρίζουν την οργάνωσή τους στο σχεσιακό μοντέλο δεδομένων, όπως αυτό προτάθηκε από

τον E. F. Codd το 1970. Κατά το οποίο τα δεδομένα οργανώνονται σε ένα σύνολο πινακοειδών μορφών όπου κάθε πίνακας αποτελεί ξεχωριστή οντότητα. Οι πίνακες αυτοί περιέχουν δεδομένα που ανήκουν σε συγκεκριμένες κατηγορίες. Κάθε πίνακας έχει τουλάχιστον μια κατηγορία δεδομένων ως στήλη ενώ ταυτόχρονα κάθε σειρά έχει ένα συγκεκριμένο στιγμιότυπο των κατηγοριών της εκάστοτε οντότητας ενώ συγχρόνως ένα ξεχωριστό κλειδί την χαρακτηρίζει. Τέλος, αυτό που δίνει ιδιαίτερο βάρος στην χρήση αυτής της μεθοδολογίας είναι το πλεονέκτημα ότι αποτελεί μια σχετικά εύκολη προσέγγιση ως αναφορά την επεκτασιμότητα της βάσης. Μετά την αρχική δημιουργία της βάσης δεδομένων, μια νέα κατηγορία μπορεί να προστεθεί σε κάποιον πίνακα χωρίς να απαιτείται να τροποποιηθεί ολόκληρη η βάση.

Από την πρώτη στιγμή που ανακαλύφθηκε το μοντέλο αυτό προτάθηκαν κάποιες οδηγίες / κανόνες που πρέπει να ακολουθούνται έτσι ώστε η βάση να ακολουθεί το σχεσιακό μοντέλο. Αυτές οι οδηγίες θεσπίστηκαν από τον δημιουργό του μοντέλου αυτού και είναι οι εξής:

- Κανόνας 0 - Ο κανόνας θεμελίωσης:
  Για οποιοδήποτε σύστημα που αποκαλείται ως σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων, ή που ισχυρίζεται ότι είναι, το σύστημα αυτό πρέπει να είναι σε θέση να διαχειρίζεται βάσεις δεδομένων εξ ολοκλήρου μέσω των σχεσιακών δυνατοτήτων του.
- Κανόνας 1 - Ο κανόνας πληροφοριών:
  Όλες οι πληροφορίες σε μια σχεσιακή βάση δεδομένων αναπαρίστανται στο λογικό επίπεδο και ακριβώς με έναν τρόπο - από τις τιμές στους πίνακες.
- Κανόνας 2 - Ο κανόνας εγγυημένης πρόσβασης:
  Κάθε ατομική τιμή σε μια σχεσιακή βάση δεδομένων είναι εγγυημένο ότι είναι λογικά προσπελάσιμη χρησιμοποιώντας έναν συνδυασμό ονόματος πίνακα, τιμής πρωτεύοντος κλειδιού και ονόματος στήλης.
- Κανόνας 3 - Συστηματική αντιμετώπιση των μηδενικών τιμών:
  Οι μηδενικές τιμές υποστηρίζονται εξαιτίας του ότι αναπαριστούν πληροφορίες που λείπουν ή/και μη εφαρμόσιμες πληροφορίες με έναν συστηματικό, αξιόπιστο και ανεξάρτητο τρόπο.

- Κανόνας 4 - Δυναμικός ηλεκτρονικός κατάλογος βασισμένος στο σχεσιακό μοντέλο:
  Η περιγραφή της βάσης δεδομένων αντιπροσωπεύεται στο λογικό επίπεδο με τον ίδιο τρόπο όπως τα συνηθισμένα δεδομένα.
- Κανόνας 5 - Ο περιεκτικός κανόνας της υπογλώσσας δεδομένων
- Κανόνας 6 - Ο κανόνας ενημέρωσης προβολής
- Κανόνας 7 - Εισαγωγή, ενημέρωση και διαγραφή υψηλού επιπέδου

- Κανόνας 8 - Η ανεξαρτησία των φυσικών δεδομένων:
  Τα προγράμματα εφαρμογών και οι δραστηριότητες των τερματικών παραμένουν λογικά άθικτα όποτε γίνονται αλλαγές σε παραστάσεις αποθήκευσης ή μεθόδους πρόσβασης.
- Κανόνας 9 - Ανεξαρτησία λογικών δεδομένων
- Κανόνας 10 – Ανεξαρτησία της ακεραιότητας
- Κανόνας 11 - Ανεξαρτησία διανομής
- Κανόνας 12 - Κανόνας μη αντικατάστασης

5.1.2 Structured Query Language (SQL)

Αποτελεί μια, μη-διαδικαστική, γλώσσα προγραμματισμού με την βοήθεια και την χρήση της οποίας επιτυγχάνεται η διαχείριση των δεδομένων μιας σχεσιακής βάσης. Είναι αυτή δηλαδή που μας δίνει την δυνατότητα να επικοινωνήσουμε, να εκτελέσουμε λειτουργίες και να ανακτήσουμε πληροφορίες από μια βάση δεδομένων χωρίς να χρειάζεται να προσδιορίσουμε τον τρόπο με τον οποίο τα δεδομένα αυτά θα ανακτηθούν. Αρχικά, για την κατασκευή της, είχε βασιστεί στην σχεσιακή άλγεβρα και στον σχεσιακό λογισμό πλειάδων. Τα βασικά στοιχεία που την αποτελούν και της δίνουν ιδιότητες, οι εντολές της δηλαδή, είναι:

- Data Definition Language commands (DDL)
  Οι εντολές DDL χρησιμοποιούνται για τον ορισμό μιας βάσης δεδομένων, συμπεριλαμβανομένης της δημιουργίας, της τροποποίησης, της διαγραφής και της εισαγωγής περιορισμών στους πίνακες.

- Data Manipulation Language commands (DML)
  Οι εντολές DML χρησιμοποιούνται για τη συντήρηση και την αναζήτηση μιας βάσης δεδομένων, συμπεριλαμβανομένης της ενημέρωσης, της εισαγωγής, της τροποποίησης και της εύρεσης δεδομένων.
- Data Control Language commands (DCL)
  Οι εντολές DCL χρησιμοποιούνται για τον έλεγχο μιας βάσης δεδομένων που περιλαμβάνει τη διαχείριση προνομίων και την αποθήκευση δεδομένων. Οι εντολές αυτές χρησιμοποιούνται για να προσδιορίσουν εάν ο χρήστης μπορεί να πραγματοποιήσει μια συγκεκριμένη ενέργεια ή όχι.

5.1.2.1 SQL σύνταξη

Η SQL ακολουθείται από ένα μοναδικό σύνολο κανόνων και οδηγιών που ονομάζεται Σύνταξη. Όλες οι εντολές SQL ξεκινούν με οποιαδήποτε από τις λέξεις-κλειδιά, στο σύνολο των οποίων δεν γίνεται διαχωρισμός μεταξύ πεζών και κεφαλαίων, όπως SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW και όλες οι δηλώσεις τελειώνουν με ερωτηματικό (;). Έτσι λοιπόν, παρακάτω, θα παρατεθούν τα βασικά στοιχεία της γλώσσας για να δημιουργηθεί μια ολοκληρωμένη SQL δήλωση.

- Ρήτρες
  Οι δηλώσεις υποδιαιρούνται σε ρήτρες με την πιο δημοφιλή να είναι η ρήτρα WHERE.
- Προβλέψεις
  Περιγράφονται οι συνθήκες εκείνες που μπορούν να προσδιοριστούν από μια δυαδική τιμή. Π.χ. BETWEEN, LIKE, IS NULL, IN, SOME/ANY, ALL, EXISTS.
- Εκφράσεις
  Οι εκφράσεις είναι αριθμητικές ή συμβολοσειρές από μόνες τους ή το αποτέλεσμα αριθμητικών χειριστών ή το αποτέλεσμα λειτουργιών.
- Ονόματα αντικειμένων
  Ονόματα αντικειμένων βάσης δεδομένων όπως πίνακες, προβολές, στήλες, λειτουργίες.

- Τιμές
  Αριθμητικές ή συμβολοσειρές.
- Ερωτήματα
  Τα οποία ανακτούν τα δεδομένα με βάση συγκεκριμένα κριτήρια.

Παρακάτω θα παρατεθούν οι βασικοί χειριστές (operators), όπως αριθμητικοί, σύγκρισης, υπολογισμού, λογικοί κοκ.

`=`	Ίσο με
`<>`	Άνισο (αρκετές ΒΔ δέχονται και το `!=` ταυτόχρονα με το `<>`)
`>`	Μεγαλύτερο από
`<`	Μικρότερο από
`>=`	Μεγαλύτερο ή ίσο
`<=`	Μικρότερο ή ίσο
`BETWEEN`	Ανάμεσα σε ένα εύρος
`LIKE`	Ταίριασμα με ένα μοτίβο χαρακτήρων
`IN`	Ίσο με μια από ή περισσότερες πιθανές τιμές
`IS` *or* `IS NOT`	Σύγκριση με το NULL(κενό)
`IS [NOT] TRUE` *or* `IS [NOT] FALSE`	Δοκιμή Boolean τιμή αλήθειας

`IS NOT DISTINCT FROM`	Είναι ίσο με την τιμή ή και τα δύο είναι μηδενικά
`AS`	Χρησιμοποιείται για την αλλαγή ονόματος στήλης κατά την προβολή των αποτελεσμάτων

Πίνακας 5 – Βασικοί SQL operators

5.2 Σύστημα διαχείρισης Βάσεων Δεδομένων – MySQL

Εξαιτίας του μεγάλου όγκου δεδομένων αλλά και ενδεχομένως την πολυπλοκότητα στις συσχετίσεις που εμφανίζονται σε μια βάση δεδομένων έπρεπε να βρεθεί ένα αποδοτικός τρόπος συντήρησης και διαχείρισης μιας βάσης. Το αυτό έγινε εφικτό μέσα από ένα Σύστημα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) ή στα αγγλικά Database Management System (DBMS).

Ένα Σύστημα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) αποτελεί ένα πρόγραμμα λογισμικού το οποίο αλληλοεπιδρά με τους χρήστες, με άλλες εφαρμογές λογισμικού αλλά και με την βάση αυτή καθαυτή με σκοπό την δημιουργία ή/και την διαχείρισή της. Έτσι λοιπόν, κατά γενίκευση, με ένα ΣΔΒΔ μπορούμε να ορίσουμε, να δημιουργήσουμε, να ρωτήσουμε, να ανανεώσουμε και να διαχειριστούμε μια βάση δεδομένων. Για τους προαναφερθείς λόγους λοιπόν, μπορούμε να χωρίσουμε την λειτουργικότητα ενός τέτοιου συστήματος σε τέσσερις κατηγορίες:

- Ορισμός δεδομένων
- Ανανέωση
- Ανάκτηση
- Διαχείριση

Αφού λοιπόν έγινε μια εκτενής αναφορά με ποιόν τρόπο και για ποιόν λόγο είναι απαραίτητο ένα Σύστημα Διαχείρισης Βάσεων Δεδομένων, παρακάτω θα αναφερθούμε στις υποκατηγορίες του και θα δώσουμε ιδιαίτερη σημασία στο Σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων (RDBMS) μιας και έχουμε επιλέξει η βάση δεδομένων μας να ακολουθεί το σχεσιακό μοντέλο. Έτσι λοιπόν έχουμε τα εξής μοντέλα:

- Μοντέλο οντοτήτων-συσχετίσεων
- Μοντέλο αντικειμένου

- Σχεσιακό μοντέλο
- Ιεραρχικό μοντέλο
- Αποθετικό μοντέλο
- Μοντέλο δικτύου

5.2.1 Σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων (RDBMS)

Όπως έχουμε ήδη αναφερθεί αποτελεί ένα σύστημα διαχείρισης βάσεων δεδομένων το οποίο ακολουθεί το μοντέλο συσχετίσεων και αποτελεί το πιο διαδομένο μοντέλο ανάμεσα στις βάσεις δεδομένων.

Στην προσπάθεια να δώσουμε έναν πιο σαφή ορισμό μπορούμε να πούμε ότι αποτελεί ένα σύνολο που παρουσιάζει μια εικόνα των δεδομένων ως μια συλλογή γραμμών και στηλών. Εδώ πρέπει να αναφερθεί ότι εφαρμόζουν συνήθως ορισμένους, αλλά όχι όλους τους κανόνες του σχεσιακού μοντέλου. Κατά συνέπεια ο όρος σταδιακά ήρθε να περιγράψει μια ευρύτερη κατηγορία συστημάτων βάσεων δεδομένων που ακολουθεί, το ελάχιστο δυο κανόνες:

- Παρουσίαση των δεδομένων στον χρήστη ως σχέσεις
- Παροχή σχεσιακών τελεστών για να χειρισμό των δεδομένων σε μορφή πίνακα

Τέλος, θα παραθέσουμε επιγραμματικά τα πιο διαδομένα συστήματα που ακολουθούν αυτό το μοντέλο όπως αυτά παρουσιάστηκαν στο DB-Engines:

- Oracle Database – 70%
- Microsoft SQL Server – 68%
- MySQL (Oracle Corporation) – 50%
- IBM DB2 – 39%
- IBM Informix – 18%
- SAP Sybase Adaptive Server Enterprise – 15%
- SAP Sybase IQ – 14%
- Teradata – 11%

5.2.1 MySQL

Η MySQL είναι ένα σύστημα, ανοιχτού κώδικα και ανεξάρτητου λειτουργικού, διαχείρισης σχεσιακών βάσεων δεδομένων και η γλώσσα

προγραμματισμού για την υλοποίησή του είναι η C και η C++, ενώ συγχρόνως ο αναλυτής SQL είναι γραμμένος σε yacc αλλά χρησιμοποιεί ένα αναλυτή-λεξικό γραμμένο μόνο γι' αυτό το σύστημα.

Παρακάτω θα αναφερθούμε επιγραμματικά στα βασικά χαρακτηριστικά του, που το κάνουν ένα τόσο διαδομένο και ταυτόχρονα εύκολο στην χρήση και αποδοτικό σύστημα:

- Υποστήριξη μεταξύ πλατφορμών
- Αποθηκευμένες διαδικασίες
- Triggers
- Υποστήριξη Unicode
- Πολλαπλοί μηχανισμοί αποθήκευσης, που επιτρέπουν σε κάποιον να επιλέξει αυτόν που είναι πιο αποτελεσματικός για κάθε πίνακα

Εξαιτίας των παραπάνω λοιπόν αποτελεί ένα σύστημα το οποίο συνεργάζεται με τρίτα προϊόντα λογισμικού αλλά και εταιρίες παροχής λογισμικού, όταν αυτά απαιτούν αποθήκευση δεδομένων, όπως Wordpress, Joomla, Drupal, Facebook, Google, Twitter κτλ.

5.3 Βάσεις δεδομένων και mobile apps

Πολλές φορές στην ανάπτυξη mobile apps δημιουργείτε η ανάγκη αποθήκευσης δεδομένων. Παρόλο που αυτό μπορεί να είναι απλό για λίγα δεδομένα, όπως για παράδειγμα κάποιες προτιμήσεις του χρήστη, υπάρχει η περίπτωση απαιτείται η υποστήριξη πιο περίπλοκων δεδομένων σε κάποιο σενάριο. Σε μια κατάσταση όπως αυτή, η SQLite μπορεί να είναι μια καλή επιλογή για την αποθήκευση, την ασφάλεια και την απόδοση των δεδομένων μίας mobile app.

Το πρώτο βήμα στη χρήση της SQLite σε μία εφαρμογή είναι να κατανοήσουμε τους περιορισμούς που υπάρχουν. Όταν σκεφτόμαστε μια βάση δεδομένων, συχνά σκεφτόμαστε τα μεγάλα συστήματα υψηλής χωρητικότητας. Αυτό όμως εύκολα καταλαβαίνουμε ότι δεν μπορεί να ισχύσει στα κινητά τηλέφωνα, καθώς δεν έχουν τη δυνατότητα να τρέχουν ολόκληρο εξυπηρετητή

για τη βάση δεδομένων. Αν και η SQLite σε μια mobile app είναι ένα πολύ ικανό εργαλείο, δεν είναι διακομιστής βάσεων δεδομένων και γι’ αυτό η χρήση της σε περιβάλλον κινητού τηλεφώνου απαιτεί προσεκτικό προγραμματισμό και δοκιμές για την ομαλή λειτουργία της.

Οι συνήθεις πρακτικές για τη χρήση της SQLite se mobile apps είναι είτε να δημιουργούνται με την εγκατάσταση της εφαρμογής είτε να παρέχονται έτοιμες μαζί με την εφαρμογή.

ΕΠΙΛΟΓΟΣ

Σε αυτό το κεφάλαιο προσπαθήσαμε να αναλύσουμε και να αποσαφηνίσουμε τις βασικές αρχές που διέπουν τις βάσεις δεδομένων. Αναλυτικότερα παρουσιάστηκαν, περιγραφικά, όλα τα μοντέλα που μπορεί να υλοποιήσει μια βάση και ειδικότερα δόθηκε ιδιαίτερη βαρύτητα στις σχεσιακές βάσεις καθώς επίσης και στο σχεσιακό σύστημα διαχείρισης δεδομένων με την βοήθεια του οποίου είδαμε ότι μπορούμε να επεξεργαστούμε την βάση. Στην συνέχεια μιλήσαμε για τον τρόπο επικοινωνίας του χρήστη με την βάση με την βοήθεια της γλώσσας SQL, ενώ τέλος αναπτύξαμε τον τρόπο εκείνο που μπορεί να επιτραπεί η επικοινωνία της βάσης δεδομένων τόσο με διαδικτυακές εφαρμογές όσο και με εφαρμογές για κινητές συσκευές.

ΚΕΦΑΛΑΙΟ 6

Αναλυτική λειτουργία και χρήση της εφαρμογής

Εισαγωγή

Σε αυτή την ενότητα θα αναλύσουμε το σύνολο της εφαρμογής που υλοποιήθηκε. Αρχικά θα μιλήσουμε για το σκοπό της εφαρμογής, έπειτα θα αναλύσουμε κάθε κομμάτι της ξεχωριστά. Θα εξετάσουμε τις τεχνολογίες που χρησιμοποιήθηκαν και επίσης θα αναλύσουμε τεχνικά θέματα καθώς και κομμάτια κώδικά της εφαρμογής.

6.1 Σκοπός της εφαρμογής

Ο βασικός σκοπός της εφαρμογής είναι αρχικά η ενημέρωση των καταναλωτών για την ποιότητα των προϊόντων και των υπηρεσιών που προσφέρουν διάφορα καταστήματα, καθώς και η βελτιστοποίηση αυτών μέσω της αξιολόγησης των καταναλωτών. Η λογική πίσω από την εφαρμογή είναι εμπνευσμένη από το κιτίο παραπόνων που διαθέτουν τα καταστήματα.

Στην ουσία αυτό που θέλει να υλοποιήσει η εφαρμογή είναι μία ηλεκτρονική μορφή του κιτίου παραπόνων, στο οποίο χρήστες θα μπορούν να αφήνουν σχόλια για καταστήματα. Με αυτόν το τρόπο οι ιδιοκτήτες θα παίρνουν αξιολογήσεις για τα προϊόντα ή τις υπηρεσίες που προσφέρουν καθώς και οι υπόλοιποι χρήστες τις εφαρμογής θα μπορούν να ενημερώνονται για διάφορα καταστήματα που τους ενδιαφέρουν.

6.2 Οντότητες της εφαρμογής

Το σύνολο της εφαρμογής βασίζεται σε 3 οντότητες, τους χρήστες, τα καταστήματά και τις αξιολογήσεις.

6.2.1 Καταστήματα

Τα καταστήματα κατηγοριοποιούνται σε κάποιες βασικές κατηγορίες με τις οποίες μπορούν να τα αναζητήσουν οι χρήστες. Στην πρώτη έκδοση της εφαρμογής οι κατηγορίες που έχουμε είναι:

- Καταστήματα Εστίασης
- Καφετέριες

- Μπαρ
- Σούπερ-μάρκετ
- Χώροι διασκέδασης
- Καταστήματα γενικού ενδιαφέροντος

Σε επόμενες εκδόσεις υπάρχει σχεδιασμός για την εισαγωγή και άλλων κατηγοριών όπως δημόσιες υπηρεσίες, τράπεζες ταχυδρομεία κλπ.

6.2.2 Χρήστες

Οι χρήστες που εντοπίζουμε στην πρώτη έκδοση της εφαρμογής, είναι οι διαχειριστές και οι πελάτες.

Διαχειριστές: έχουν την πλήρη εποπτεία των δεδομένων τις εφαρμογής. Οι βασικές τους λειτουργίες είναι να εισάγουν τα καταστήματα στην εφαρμογή καθώς και να κάνουν έλεγχο των σχολίων τον πελατών πριν δημοσιευτούν στην εφαρμογή.

Πελάτες: Οι πελάτες είναι αυτοί που αφήνουν τις αξιολογήσεις για τα καταστήματα. Μία ακόμα λειτουργία που υποστηρίζεται για τους πελάτες είναι ότι μπορούν να κάνουν αιτήματα για την εισαγωγή νέου καταστήματος στην εφαρμογή.

Σε επόμενη έκδοση της εφαρμογής υπάρχει σχεδιασμός για να εισαγωγή και νέα ομάδα χρηστών. Η ομάδα αυτοί θα είναι οι καταστηματάρχες, οι οποίοι θα συνδέονται με κάποιο κατάστημα, έπειτα από έτοιμά τους προς τους διαχειριστές. Οι λειτουργίες που θα υποστηρίζονται γι’ αυτούς θα είναι να απαντούν στους χρήστες για σχόλιο στο κατάστημά τους καθώς και να λαμβάνουν ειδοποιήσεις για καινούρια σχόλια που τους αφορούν.

6.3 Γενικό σχέδιο της εφαρμογής

Η εφαρμογή αποτελείτε από μία ιστοσελίδα και από μία εφαρμογή για smartphones.

Οι χρήστες έχουν τη δυνατότητα να επισκέπτονται την εφαρμογή και από τον υπολογιστή τους μέσο της ιστοσελίδας ή μέσο του κινητού τους από την σχετική εφαρμογή. Από την ιστοσελίδα οι διαχειριστές μπορούν να επεξεργάζονται τα δεδομένα της εφαρμογής.

Τα δεδομένα είναι αποθηκευμένα στην ιστοσελίδα και η mobile app συγχρονίζετε με αυτήν.

6.4 Τεχνικός σχεδιασμός & τεχνολογίες που επιλέχθηκαν

Για να μπορέσουμε να πετύχουμε το παραπάνω σχήμα, χρειάστηκε να χρησιμοποιήσουμε ένα αριθμό τεχνολογιών που συνδυάζονται μεταξύ τους για να παραχθεί το τελικό αποτέλεσμα. Οι επιλογή τον τεχνολογιών έγινε με βάση την καταλληλόλητά τους για το σκοπό της εφαρμογής, την ταχύτητα υλοποίησης καθώς και το αν μπορούν να συνδυαστούν μεταξύ τους.

6.4.1 Βασικό σύστημα της εφαρμογής

Το βασικό σύστημα της εφαρμογής είναι το Wordpress. Το Wordpress με τη χρήση κάποιο plugin που διαχειρίζεται linstings ήταν η επιλογή μας για την ιστοσελίδα και το τη διαχείριση των δεδομένων και των χρηστών. Επίσης το αισθητικό κομμάτι της εφαρμογής υλοποιήθηκε με το θέμα για Wordpress, Javo Spot, το οποίο είχε ενσωματωμένο και το plugin που διαχειρίζεται τα καταστήματα (listings).

Με την βασική εγκατάσταση του Wordpress έχουμε τη λειτουργικότητα που χρειαζόμαστε για τους χρήστες. Οι λειτουργίες αυτές είναι η ομάδες χρηστών (Διαχειριστές και πελάτες), η πιστοποίηση του χρήστη για την εισαγωγή στην εφαρμογή (authentication) και οι αποθήκευση των δεδομένων του χρήστη στην βάση δεδομένων του Wordpress.

Η κατηγοριοποίηση και η αποθήκευση των καταστημάτων υλοποιείται με τη χρήση ενός plugin που είναι ενσωματωμένο στο theme που χρησιμοποιήσαμε και διαχειρίζεται τα listings. Με τη χρήση αυτού του plugin μπορούμε να δημιουργήσουμε και να επεξεργαστούμε καταστήματα και κατηγορίες καταστημάτων. Έπειτα αυτά τα αποθηκεύουμε στη βάση δεδομένων της εφαρμογής. Επίσης με το συγκεκριμένο Plugin μπορούμε να παρουσιάζουμε τα καταστήματα στην ιστοσελίδα είτε με τη μορφή λίστας είτε σε χάρτη. Τέλος έχουμε και λεπτομερής απεικόνιση του κάθε καταστήματος με στοιχεία όπως ονομασία, περιγραφή, στοιχεία επικοινωνίας, απεικόνιση στο χάρτη.

6.4.2 Βάση δεδομένων της εφαρμογής

Η βάση δεδομένων που χρησιμοποιήθηκε είναι η MySQL. Η MySQL είναι μία από της βάσεις δεδομένων που υποστηρίζει το Wordpress με έτοιμους τρόπους σύνδεσης με το σύστημα. Επίσης αυτή η βάση δεδομένων υπάρχει στα περισσότερα πακέτα φιλοξενίας που διατίθενται για τη φιλοξενία ιστοσελίδων.

Σε παρακάτω κεφάλαια θα αναλύσουμε το σχήμα της βάσης δεδομένων καθώς και θα παρουσιάσουμε σχετικό κώδικα για την σύνδεση με τη βάση και τις κλήσεις σε αυτή.

6.4.3 Application Κινητού

Η εφαρμογή του για τα smartphone υλοποιήθηκε με την υβριδική μέθοδο ανάπτυξης και πιο συγκεκριμένα με τη χρήση του Adobe Phonegap και του Ionic Framework. Η επιλογή του τρόπου ανάπτυξης έγινε με βάση την ταχύτητα υλοποίηση. Με τη χρήση των δύο τεχνολογιών που αναφέρουμε παραπάνω, καταφέραμε να υλοποιήσουμε την εφαρμογή και έπειτα να είμαστε σε θέση να την διαθέσουμε και στα δύο βασικά λειτουργικά συστήματα για Smartphones μόνο με κάποιες μικροαλλαγές στην βασική υλοποίηση.

Το mobile app εκμεταλλεύεται τα δεδομένα που υπάρχουν στην ιστοσελίδα με τη χρήση μίας ενδιάμεσης γεφύρωσης μεταξύ της βάσης δεδομένων της κεντρικής εφαρμογής και του κινητού.

6.4.4 Γεφύρωση – Web Service

Για να έχει τη δυνατότητα το mobile app να τραβάει τα δεδομένα που χειρίζεται η κεντρική εφαρμογή μέσω του Wordpress χρειάζεται κάποια ενδιάμεση γέφυρα.

Το Wordpress διαθέτει έτοιμα plugins τα οποία έχουν τη δυνατότητα να δημιουργούν web service χρησιμοποιώντας τα δεδομένα που είναι αποθηκευμένα στη βάση δεδομένων του Wordpress. Έγινε απόπειρα για χρήση τέτοιου plugin αλλά επειδή το plugin που χρησιμοποιεί η ιστοσελίδα για την διαχείριση των listings έχει δικό του τρόπο για την αποθήκευση των δεδομένων, που ξεφεύγει από την συνηθισμένη μέθοδο του Wordpress, δεν υπήρχε η δυνατότητα να χρησιμοποιηθεί κάποια τέτοια λύση.

Η λύση που δώσαμε ήταν να αναπτύξουμε ένα web service από την αρχή, υλοποιημένο σε PHP. Το web service αυτό ανάλογα με της κλήσεις που του γίνονται, επιστρέφει τα δεδομένα που ανακτά με ερωτήματα SQL απευθείας από τη βάση δεδομένων σε μορφή JSON.

Σε παρακάτω ενότητες θα παρουσιάσουμε τη δομή του web service τον τρόπο που γίνονται οι κλήσεις καθώς και τα επιστρεφόμενα αποτελέσματα.

6.5 Ανάλυση κεντρικής εφαρμογής (Wordpress)

Το Wordpress αν και ξεκίνησε σαν λογισμικό αποκλειστικά για την δημιουργία Blogs, στην πορεία μεταλλάχθηκε σε ένα λογισμικό κατάλληλο για την δημιουργία κάθε ίδιους ιστοσελίδας. Υπάρχει μεγάλη ποικιλία από plugins τα οποία μπορούν να προστεθούν στη βασική εγκατάσταση και προσφέρουν επιπλέον λειτουργικότητα την ιστοσελίδα. Έτσι ο χρήστης έχει τη δυνατότητα να δημιουργήσει από απλή ιστοσελίδα παρουσίασης μέχρι ηλεκτρονικό κατάστημα ή κοινωνικό δίκτυο.

Έτσι και εμείς αφού εγκαταστήσαμε τη βασική έκδοση του Wordpress εγκαταστήσαμε το Javo Spot theme το οποίο το προμηθευτήκαμε από το themeforest.net Το συγκεκριμένο Theme είναι κατάλληλο για την κατηγοριοποίηση και προβολή οντοτήτων.

6.5.1 Εγκατάσταση Wordpress

Για να λειτουργήσει το Wordpress χρειάζεται να εγκατασταθεί σε κάποιον Web Server που μπορεί να εκτελεί PHP. Επίσης χρειάζεται να υπάρχει κάποια βάση δεομένων MySQL ή Maria DB. Οι βασικές απαιτήσεις είναι PHP 5.6 και μεγαλύτερη καθώς και MySQL 5.6 και μεγαλύτερη ή MariaDB 10.0 και μεγαλύτερη.

Η δική μας εγκατάσταση έγινε σε Linux Server, χρησιμοποιεί PHP 5.6 και MySQL με Maria DB Server 10.0.31.

Η εγκατάσταση του Wordpress είναι ιδιαίτερα εύκολη. Αρχικά δημιουργεί ο χρήστης την βάση δεδομένων που θα χρησιμοποίηση. Έπειτα αφού κατεβάσει την τελευταία έκδοση του Wordpress από την επίσημη ιστοσελίδα, ανεβάζει το .zip αρχείο στον server και κάνει αποσυμπίεση τα αρχεία στο φάκελο που επιθυμεί.

Για να ολοκληρωθεί η εγκατάσταση ο χρήστης πρέπει από κάποιο browser να ανοίξει τη διεύθυνση που έκανε αποσυμπίεση τα αρχεία του Wordpress. Εκεί θα του εμφανιστεί ένας οδηγός εγκατάστασης όπου θα εισάγει κάποια στοιχεία, όπως το όνομα της ιστοσελίδας, το όνομα χρήστη, ο κωδικός και το email του διαχειριστή, καθώς και τα στοιχεία της βάσης δεδομένων. Με το που συμπληρωθούν τα απαραίτητα στοιχεία ολοκληρώνεται η εγκατάσταση και ο χρήστης έχει πρόσβαση στο διαχειριστικό περιβάλλον της ιστοσελίδας.

6.5.2 Δομή Αρχείων του Wordpress

Οι φάκελοι και τα αρχεία του Wordpress έχουν την παρακάτω δομή:

Ο κεντρικός φάκελος περιέχει:

- Τον φάκελο wp-admin ο οποίος περιέχει τις λειτουργίες του διαχειριστικού περιβάλλοντος
- Τον φάκελο wp-content όπου περιέχει τα themes, τα plugins και τα δεδομένα που φορτώνουν οι χρήστες, πχ φωτογραφίες.
- Τον φάκελο wp-includes όπου περιέχει διάφορες βιβλιοθήκες που χρειάζεται το Wordpress για να εκτελεστεί.
- Επίσης ο κεντρικός φάκελος εμπεριέχει και τα αρχεία index.php όπου από εκεί ξεκινά η εκτέλεση του Wordpress, wp-config.php όπου υπάρχουν οι βασικές ρυθμίσεις πχ τα στοιχεία σύνδεσης στη βάση δεδομένων, wp-login.php που είναι υπεύθυνο για τη σύνδεση των χρηστών. Υπάρχουν επίσης και άλλα αρχεία τα οποία δεν χρειάζεται να αναλυθούν στην παρούσα φάση.

6.5.3 Εγκατάσταση του Theme

Ένας από τους βασικούς λόγους που χρησιμοποιούμε το Wordpress είναι η ταχύτητα με την οποία μπορούμε να υλοποιήσουμε εφαρμογές. Για αυτό το λόγο όταν ξεκινάμε την υλοποίηση κάποιας διαδικτυακής εφαρμογής με Wordpress, συνήθως επιλέγουμε κάποιο έτοιμο Theme το οποίο περιέχει το γραφικό περιβάλλον που βλέπουν οι επισκέπτες καθώς και τις λειτουργίες που θέλουμε να έχει η ιστοσελίδα μας. Αυτά τα Theme υλοποιούνται από εταιρίες οι μεμονωμένους προγραμματιστές και διατίθενται για αγορά προς πώληση ή και δωρεάν. Οι τιμές συνήθως είναι σχετικά χαμηλές και κυμαίνονται από 30€ έως 100€.

Η εγκατάσταση του Theme γίνεται με δύο τρόπους. Ο ένας είναι να γίνει εγκατάσταση μέσω του διαχειριστικού περιβάλλοντάς και ο άλλος είναι να ανεβάσει ο χρήστης χρησιμοποιώντας FTP σύνδεση το theme στον φάκελο όπου τοποθετεί το Wordpress τα themes που είναι εγκατεστημένα. Τα theme είναι ένα zip αρχείο που περιλαμβάνει ένα φάκελο με όλα τα αρχεία που χρειάζεται το theme για να λειτουργήσει.

Μετά την εγκατάσταση του theme συνήθως ακολουθούν δύο άλλες ενέργειες:

1. Εγκατάσταση plugins του theme Τα περισσότερα theme χρησιμοποιούν κάποια Wordpress plugins. Οπότε για να μπορέσουν να πλήρως λειτουργικά χρειάζεται αμέσως μετά την εγκατάσταση του theme να εγκατασταθούν και αυτά τα πρόσθετα. Πλέον αυτό γίνεται αυτοματοποιημένα μέσα από το theme ενώ παλιότερα χρειαζόταν οι χρήστες να ανεβάσουν και να εγκαταστήσουν τα plugin αυτόνομα.
2. Εισαγωγή Demo Δεδομένων Τα theme στην ουσία σχηματίζουν το πώς θα παρουσιάζονται τα δεδομένα της ιστοσελίδας, του χώρους που θα υπάρχουν κάποια πράγματα όπως τα μενού, τα sidebar, τα κείμενα, οι φωτογραφίες κλπ. Έπειτα πρέπει να εισαχθούν και κάποια demo δεδομένα όπως εικόνες κείμενα φωτογραφίες για να μπορεί το theme να εμφανίζεται σαν κανονική ιστοσελίδα και έπειτα το προγραμματιστής να ξεκινήσει την υλοποίηση πατώντας πάνω σε αυτά τα δεδομένα. Τα δεδομένα εγκαθιστούν τε είτε αυτόματα με κάποιες ενέργειες στο διαχειριστικό του Wordpress, είτε ανεβάζοντας τα δεδομένα κατευθείαν στο server και στη βάση δεδομένων.

Το theme που εγκαταστήσαμε στη δική μας εφαρμογή είναι το Javo Spot theme. Το προμηθευτήκαμε από το Themeforest, το μεγαλύτερο marketplace για λογισμικό διαδικτυακών εφαρμογών, στην τιμή των 35€

6.5.4 Επέκταση Wordpress με την χρήση child theme

Σε γενικές γραμμές το Wordpress εξελίσσεται συνεχώς και σε μικρά χρονικά διαστήματα βγαίνουν διάφορες αναβαθμίσεις στο Wordpress ή στα themes ή στα plugins. Για αυτό το λόγο εάν χρειαστεί να γίνει επέκταση από προγραμματιστή δεν προτείνεται να γίνονται αλλαγές κατευθείαν στα αρχεία του

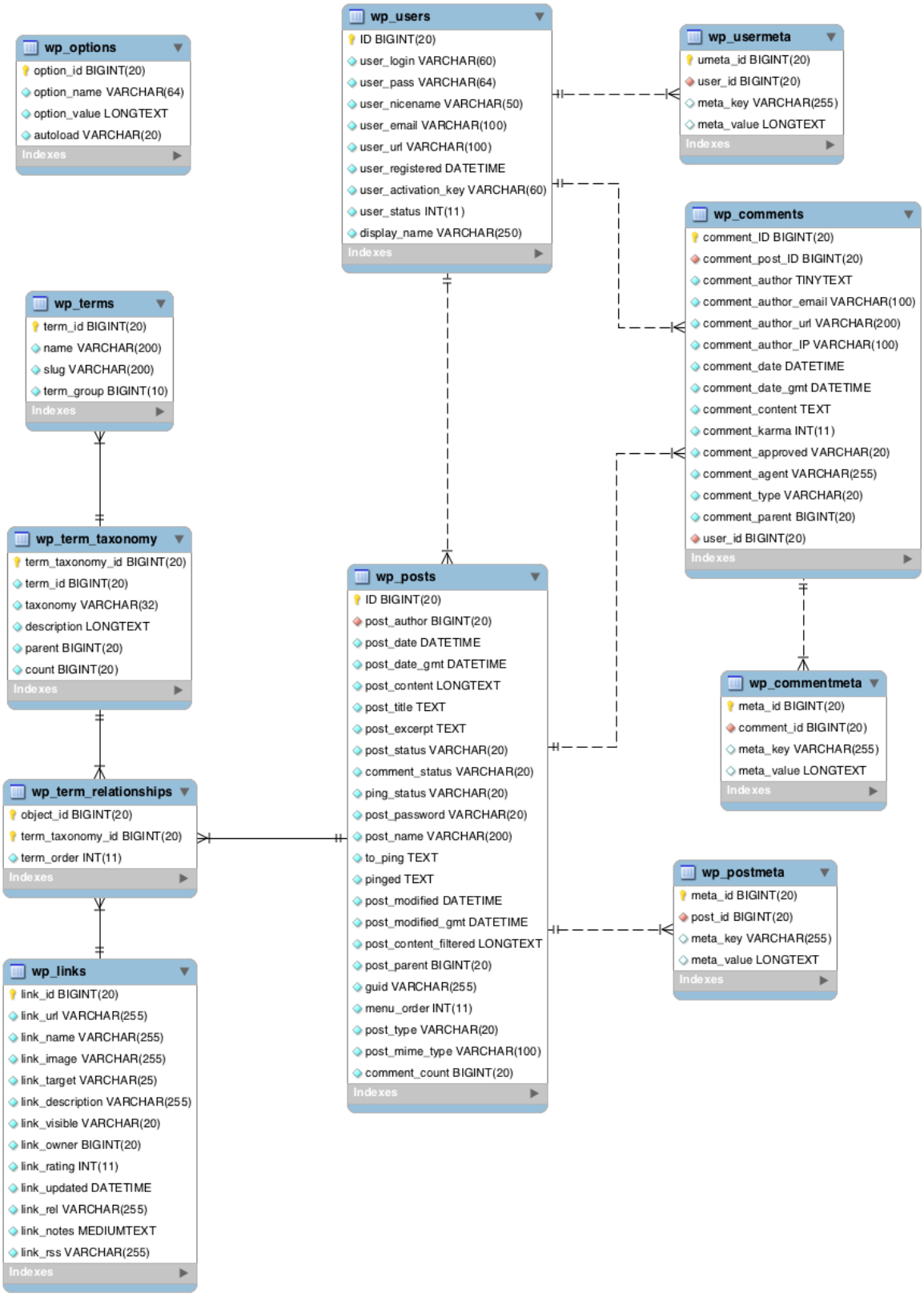
Wordpress καθώς αυτές οι αλλαγές είναι πιθανόν να χαθούν σε κάποια αναβάθμιση.

Γι’ αυτό το λόγο προτείνεται η λειτουργικότητα του child theme, το οποίο είναι στην ουσία μία επέκταση του theme που χρησιμοποιεί η ιστοσελίδα και όλες οι αλλαγές που γίνονται εκεί δεν τροποποιούνται από τυχόν αναβαθμίσεις. Ο προγραμματισμός στο Wordpress γίνεται χρησιμοποιώντας το hooking σύστημα του Wordpress. Οι προγραμματιστές χρησιμοποιώντας τα δίφορα hooks που έχει το βασικό Wordpress καθώς και τα plugins του μπορούν να ενσωματώσουν διάφορες λειτουργικότητες σε κομμάτια της ιστοσελίδας.

Ο προγραμματιστής χρησιμοποιώντας το αρχείο functions.php του child theme μπορεί να γράψει μεθόδους χρησιμοποιώντας hooks και έπειτα κατά την εκτέλεση του Wordpress να ενσωματώνονται οι αλλαγές από αυτές τις μεθόδους στην ιστοσελίδα.

6.5.5 Βάση δεδομένων του Wordpress

Το σχήμα της βάσης δεδομένων του Wordpress είναι το παρακάτω.



Εικόνα 2 – Σχήμα βάσης δεδομένων Wordpress

Οι βασικοί πίνακες

Ο wp-posts, στον οποίο αποθηκεύονται τα περισσότερα δεδομένα που υπάρχουν στο Wordpress. Σε αυτόν το πίνακα αποθηκεύονται οι σελίδες οι φωτογραφίες τα posts και οι διάφορες άλλες οντότητες. Με το πεδίο post_type δηλώνεται τι είδος είναι η κάθε εγγραφή.

Ο wp_postmeta, σε αυτόν αποθηκεύονται επιπλέον χαρακτηριστικά που μπορεί να έχει κάποια εγγραφή – οντότητα που αποθηκεύεται στον πίνακα wp_posts.

Ο wp_users & wp_usermeta, εδώ αποθηκεύονται οι χρήστες και τα δεδομένα τους

Ο wp_comments, που αποθηκεύονται τα σχόλια των χρηστών

Παράδειγμα αποθήκευσης δεδομένων της εφαρμογής μας

Όπως αναφέραμε και παραπάνω η εφαρμογή μας χρησιμοποιεί ένα plugin για να διαχειρίζεται listings, δηλαδή τα καταστήματα. Τα καταστήματα είναι αποθηκευμένα στον πίνακα wp_posts με post_type = lv_listing. Σε αυτόν τον πίνακα υπάρχει το όνομα του καταστήματος, η περιγραφή του και πληροφορίες για το πότε δημιουργήθηκε/τροποποιήθηκε η εγγραφή.

Σε αυτόν τον πίνακα αλλά με σε διαφορετική εγγραφή αποθηκεύονται και οι εικόνες που σχετίζονται με τα καταστήματα. Αυτό γίνεται με το να αποθηκεύονται αυτές οι εγγραφές με post_type = attachment και χρησιμοποιώντας το post_parent για να δείξουν το id της εγγραφής που είναι το κατάστημα.

Για να αποθηκευτούν τα επιπλέον δεδομένα του καταστήματος όπως πχ η διεύθυνση του, χρησιμοποιούμε τον πίνακα wp_postmeta. Σε αυτό τον πίνακα υπάρχει η εγγραφή post_id που δείχνει με πιο κατάστημα που είναι αποθηκευμένο στον wp_posts συνδέετε αυτό το επιπλέον χαρακτηριστικό. Έπειτα υπάρχει η στήλη meta_key στην οποία λέμε ποιο είναι το χαρακτηριστικό πχ για να δηλώσουμε τη διεύθυνση του καταστήματος βάζουμε _address. Τέλος υπάρχει το meta_value που δηλώνουμε την τιμή του χαρακτηριστικού, πχ 'Πλατεία Ιπποδρομίου 10' για τη διεύθυνση του καταστήματος.

6.5.6 Παρουσίαση ενδεικτικών κομματιών κώδικα

Εδώ θα παρουσιάσουμε τρία ενδεικτικά παραδείγματα κώδικα. Θα παρουσιάσουμε τη σύνδεση με τη βάση δεδομένων, τη σύνδεση των χρηστών καθώς και το πώς ορίζουμε το child theme.

Σύνδεση στη βάση δεδομένων

```
public function __construct( $dbuser, $dbpassword, $dbname, $dbhost ) {
	register_shutdown_function( array( $this, '__destruct' ) );
		if ( WP_DEBUG && WP_DEBUG_DISPLAY )
		$this->show_errors();
		/* Use ext/mysqli if it exists and:
		 *  - WP_USE_EXT_MYSQL is defined as false, or
		 *  - We are a development version of WordPress, or
		 *  - We are running PHP 5.5 or greater, or
		 *  - ext/mysql is not loaded.
		 */
		if ( function_exists( 'mysqli_connect' ) ) {
			if ( defined( 'WP_USE_EXT_MYSQL' ) ) {
				$this->use_mysqli = ! WP_USE_EXT_MYSQL;
			} elseif ( version_compare( phpversion(), '5.5', '>=' ) || ! function_exists(
'mysql_connect' ) ) {
				$this->use_mysqli = true;
			} elseif ( false !== strpos( $GLOBALS['wp_version'], '-' ) ) {
				$this->use_mysqli = true;
			}
		}

		$this->dbuser = $dbuser;
		$this->dbpassword = $dbpassword;
		$this->dbname = $dbname;
		$this->dbhost = $dbhost;

		// wp-config.php creation will manually connect when ready.
		if ( defined( 'WP_SETUP_CONFIG' ) ) {
			return;
		}

		$this->db_connect();
	}
```

Στο παραπάνω κομμάτι κώδικα, βλέπουμε τον τρόπο με τον οποίο γίνεται η σύνδεση στη βάση δεδομένων του Wordpress. Η σύνδεση πραγματοποιείται στον constructor της κλάσης **wpdb** που είναι υπεύθυνη για την επικοινωνία του Wordpress με τη βάση δεδομένων.

Όπως βλέπουμε στην παραπάνω μέθοδο, παίρνει σαν όρισμα τα στοιχεία σύνδεσης της βάσης δεδομένων. Έπειτα γίνεται έλεγχος εάν είναι ορισμένος ο mysqli connector (Ορίζεται στο αρχείο wp_config.php αυτόματα κατά την εγκατάσταση). Στη συνέχεια γίνεται έλεγχος σχετικά με την έκδοση της php και

της mysql που είναι εγκατεστημένες στον server. Τέλος στην τελευταία γραμμή πραγματοποιείτε η σύνδεση.

Σύνδεση χρήστη

```
/**
 * Authenticate a user, confirming the username and password are valid.
 *
 * @since 2.8.0
 *
 * @param WP_User|WP_Error|null $user     WP_User or WP_Error object
from a previous callback. Default null.
 * @param string               $username Username for authentication.
 * @param string               $password Password for authentication.
 * @return WP_User|WP_Error WP_User on success, WP_Error on failure.
 */
function wp_authenticate_username_password($user, $username,
$password) {
    if ( $user instanceof WP_User ) {
            return $user;
    }
    if ( empty($username) || empty($password) ) {
            if ( is_wp_error( $user ) )
                    return $user;
            $error = new WP_Error();
            if ( empty($username) )
                    $error->add('empty_username',
__('<strong>ERROR</strong>: The username field is empty.'));
            if ( empty($password) )
                    $error->add('empty_password',
__('<strong>ERROR</strong>: The password field is empty.'));
            return $error;
    }
    $user = get_user_by('login', $username);
    if ( !$user ) {
            return new WP_Error( 'invalid_username',
                    __( '<strong>ERROR</strong>: Invalid username.' ) .
                    ' <a href="' . wp_lostpassword_url() . '">' .
                    __( 'Lost your password?' ) .
                    '</a>'
            );
    }
    /**
     * Filter whether the given user can be authenticated with the provided
$password.
     *
     * @since 2.5.0
     *
     * @param WP_User|WP_Error $user     WP_User or WP_Error object if a
previous
```

```
         *                          callback failed authentication.
         * @param string           $password Password to check against the user.
         */
        $user = apply_filters( 'wp_authenticate_user', $user, $password );
        if ( is_wp_error($user) )
                return $user;
        if ( ! wp_check_password( $password, $user->user_pass, $user->ID ) ) {
                return new WP_Error( 'incorrect_password',
                        sprintf(
                                /* translators: %s: user name */
                                __( '<strong>ERROR</strong>: The password you
entered for the username %s is incorrect.' ),
                                '<strong>' . $username . '</strong>'
                        ) .
                        ' <a href="' . wp_lostpassword_url() . '">' .
                        __( 'Lost your password?' ) .
                        '</a>'
                );
        }
        return $user;
}
```

Η παραπάνω μέθοδος είναι υπεύθυνη για την σύνδεση του χρήστη. Δέχεται σαν ορίσματα τα στοιχεία του χρήστη, και ελέγχει εάν τα στοιχεία που δόθηκαν δεν είναι κενά, έπειτα ελέγχει εάν υπάρχει ο χρήστης και μετά εάν ο κωδικός του χρήστη είναι σωστός. Τέλος εάν όλοι οι έλεγχοι είναι σωστοί, επιστρέφει το αντικείμενο user με όλα τα χαρακτηριστικά του.

Ορισμός του child theme

```
/*
Theme Name: Javo Spot
Theme URI: http://javothemes.com/spot
Description: Child theme for the Javo Spot
Author: Javo Themes
Author URI: http://javothemes.com
Template: javo-spot
Version: 1.0
*/
@import url ("../javo-spot/style.css");
/*Add custom Styles From Here*/
```

Με τον παραπάνω κώδικα ορίζουμε το child theme. Πρώτα δημιουργούμε έναν φάκελο με το όνομα του theme –child. Έπειτα δημιουργούμε εκεί το αρχείο style.css και εκεί γράφουμε τον παραπάνω κώδικα για να δηλώσουμε σε πιο theme είναι child το παρόν.

6.6 Web Service

Όπως αναφέραμε και σε παραπάνω κεφάλαιο, το Wordpress έχει έτοιμες λύσεις για να παρέχει τα δεδομένα του σε κάποιο Api.Ποιο συγκεκριμένα χρησιμοποιήσαμε το WP REST API plugin. Παρατηρήσαμε ότι η λειτουργία του είναι πολύ καλή όσον αφορά τα posts του Wordpress, αλλά επειδή η δική μας ιστοσελίδα χρησιμοποιούσε το plugin για τα listings δεν είχαμε σωστό αποτέλεσμα με αυτόν τον τύπο δεδομένων. Έτσι αποφασίσαμε να δημιουργήσουμε ένα δικό μας Web Service Api, το οποίο θα κάνει απευθείας ερωτήματα SQL στη βάση δεδομένων και θα επιστρέφει τα αποτελέσματα σε μορφή JSON.

Το Web Service είναι υλοποιημένο σε Object Oriented PHP. Δέχεται κλήσεις χρησιμοποιώντας κάποιες παραμέτρους στο URL και επιστρέφει τα σχετικά δεδομένα.

6.6.1 Δομή του Web Service

Το Web Service αποτελείτε από 6 αρχεία PHP. Τα πέντε από αυτά αποτελούν από μία κλάση που είναι υπεύθυνη για κάθε μία από τις οντότητες που πραγματεύεται η εφαρμογή. Το ένα αρχείο που περισσεύει είναι το αρχείο εισαγωγής, το οποίο δέχεται τις κλήσεις και εκτελεί ολοκληρώνει τις ενέργειες.

Αναλυτικά για το κάθε αρχείο:

Index.php

Αυτό το αρχείο είναι υπεύθυνο για το routing του service. Σε αυτό το αρχείο γίνεται η εισαγωγή στο σύστημα. Εξετάζονται οι παράμετροι με τις οποίες έγινε η κλήση και έπειτα εκτελούνται οι απαραίτητες ενέργειες. Σε αυτό το αρχείο επίσης, γίνονται τα απαραίτητα require για τα υπόλοιπα αρχεία ώστε να μπορεί να έχει πρόσβαση η εφαρμογή στις κλάσεις που αναφέραμε παραπάνω.

db_connect.php

Σε αυτό το αρχείο υπάρχει η κλάση ConnectToDB που είναι υπεύθυνη για τη σύνδεση με τη βάση δεδομένων. Επίσης σε αυτό το αρχείο υπάρχουν τα στοιχεία σύνδεσης με τη βάση δεδομένων.

Στο δομητή αυτής της κλάσης δημιουργείτε η σύνδεση με τη βάση δεδομένων. Αρχικοποιείται το αντικείμενο της σύνδεσης το οποίο μπορούμε να

το ανακτήσουμε με τη μέθοδο getConection() και να εκτελούμε ερωτήματα SQL κατευθείαν στη βάση δεδομένων.

listings.php

Εδώ έχουμε την κλάση Listings η οποία διαχειρίζεται τα listings, δηλαδή τα καταστήματα, που αποθηκεύει στη βάση δεδομένων το Wordpress.

Στον δομητή αυτής της κλάσης αρχικοποείται το αντικείμενο που είναι υπεύθυνο για τη σύνδεση με τη βάση δεδομένων, χρησιμοποιώντας την μέθοδο που αναφέραμε παραπάνω (getConection() από την κλάση ConnectToDB).

Οι μέθοδοι που υπάρχουν σε αυτή την κλάση χωρίζονται στις ιδιωτικές (private), οι οποίες επεξεργάζονται κάποια δεδομένα εντός της κλάσης και οι δημόσιες (public), οι οποίες καλούνται και επιστρέφουν τα δεδομένα που ανακτώνται από τη βάση δεδομένων.

Αρχικά θα δούμε τις ιδιωτικές μεθόδους και τι εξυπηρετούν και έπειτα τις δημόσιες.

Ιδιωτικές μέθοδοι:

- *_removeHtmlFromContent()*

  Το Wordpress όταν αποθηκεύει δεδομένα στη βάση δεδομένων τα οποία προέρχονται από τον κειμενογράφο του, συνήθως αποθηκεύει μαζί με το κείμενο και κάποια html tags. Όταν ανακτούμε αυτά τα δεδομένα για να τα χρησιμοποιήσουμε στην mobile app, τα χρειαζόμαστε χωρίς html tags γιατί είναι πιθανόν παραμορφώνουν το αισθητικό αποτέλεσμα της εφαρμογής.

  Αυτή η μέθοδος δέχεται σαν όρισμα κάποια string, διαγράφει τα HTML tags από αυτό και το επιστρέφει με καθαρό κείμενο.

- *_getIdFromCategory()*

  Οι κλήσεις στο service που σχετίζονται με κατηγορίες καταστημάτων γίνονται αναφέροντας το slug του ονόματος της κατηγορίας. Για να μπορέσουν όμως να διεκπεραιωθούν οι κατηγορίες εσωτερικά το service χρειάζεται να γνωρίζουμε το id τους.

Σε αυτή τη μέθοδο έχουμε ορίσει ποιο είναι το id της κάθε κατηγορίας και έτσι γίνεται η αντιστοίχιση. Καλούμε τη μέθοδο με όρισμα το slug της κατηγορίας και παίρνουμε το id.

- *_sqlListingResultToJson() & _sqlListingResultToJsonOneListing()*

Αυτές οι δύο μέθοδοι, δέχονται σαν όρισμα ένα ερώτημα SQL σε μορφή string, έπειτα εκτελούν αυτό το ερώτημα, χρησιμοποιώντας το αντικείμενο της σύνδεσης με τη βάση δεδομένων. Στη συνέχεια μετατρέπουν το αποτέλεσμα της του SQL ερωτήματος σε πίνακα. Τέλος κάνουν print τον πίνακα, με τη χρήση της μεθόδου της PHP json_encode() για να εκτυπώσει τα δεδομένα σε μορφή JSON.

Η διαφορά τους είναι ότι η *_sqlListingResultToJson()* επιστρέφει δεδομένα για ένα αποτέλεσμα, ενώ η *_sqlListingResultToJsonOneListing()* επιστρέφει δεδομένα για πολλές εγγραφές.

Δημόσιες μέθοδοι:

Οι δημόσιες μέθοδοι που υποστηρίζει αυτή η κλάση είναι οι εξής: *getAllListings()* η οποία εκτυπώνει όλα τα καταστήματα, *getListingsByCategory()* η οποία δέχεται σαν όρισμα κάποια κατηγορία και εκτυπώνει τα καταστήματα αυτής της κατηγορίας, *getListing()* η οποία δέχεται το id ενός καταστήματος και εκτυπώνει τα δεδομένα του, *getLasListingsUpdate()* η οποία εκτυπώνει την ημερομηνία πότε έγινε η τελευταία ανανέωση καταστήματος στη βάση δεδομένων.

Οι παραπάνω μέθοδοι έχουν τον ίδιο τρόπο λειτουργίας. Σε κάθε μέθοδο από αυτές έχουμε ορίσει το SQL query που χρειάζεται για να ανακτηθούν τα δεδομένα. Έπειτα η μέθοδος καλεί μία από τις *_sqlListingResultToJson() & _sqlListingResultToJsonOneListing()* για να εκτυπώσει τα δεδομένα σε μορφή JSON.

users.php

Αυτό το αρχείο περιλαμβάνει την κλάση Users. Η συγκεκριμένη κλάση προς το παρών έχει ακριβώς την ίδια λειτουργία με την Listings, μόνο που αντί να επιστρέφει δεδομένα σχετικά με τα καταστήματα, επιστρέφει δεδομένα σχετικά με τους χρήστες.

news.php

Σε αυτό το αρχείο έχουμε την κλάση News οι οποία διαχειρίζεται την ενότητα των νέων / ανακοινώσεων της ιστοσελίδας. Οι μέθοδοι που υπάρχουν σε αυτή την κλάση είναι πανομοιότυπες με αυτές των παραπάνω κλάσεων και λειτουργούν με την ίδια ακριβώς λογική.

contact.php

Εδώ θα βρούμε την κλάση Contact η οποία είναι υπεύθυνη για την αποθήκευση των μηνυμάτων(παραπόνων) στη βάση δεδομένων.

Μία ακόμα λειτουργία που εκτελείται από αυτή την κλάση αυτή είναι η αποστολή email. Γνωρίζουμε ότι από την mobile app δεν είναι τεχνικά δυνατόν να γίνει αποστολή email χωρίς την χρήση server. Γι' αυτό το λόγο όταν κάποιος χρήστης θέλει να επικοινωνήσει από τη φόρμα επικοινωνίας της εφαρμογής, κατευθείαν στους διαχειριστές της ιστοσελίδας, καλείτε από την mobile app αυτή η μέθοδος για να αποσταλεί το μήνυμα της φόρμας με email.

Οι δύο δημόσιες μέθοδοι που διεκπεραιώνουν τις παραπάνω λειτουργίες είναι η *store_message()* η οποία δέχεται σαν όρισμα το παράπονο / αξιολόγηση και το id του καταστήματος και έπειτα τα αποθηκεύει στη βάση δεδομένων. Η *sendEmail()* η οποία δέχεται σαν όρισμα τα στοιχεία του email και έπειτα χρησιμοποιώντας τη μέθοδο mail() της PHP κάνει αποστολή του email.

6.6.2 Κλήσεις στο Api & δεδομένα που επιστρέφει

Η κλήση για να ανακτήσουμε όλα τα καταστήματα από το Web Service γίνεται με το URL:

> http://www.deltio-paraponon.gr/iserv_api/index.php?method=all

Το αποτέλεσμα την κλήσης του παραπάνω αποτελέσματος είναι ένας πίνακας που αποτελείτε από αντικείμενα με τα στοιχεία των καταστημάτων. Πιο συγκεκριμένα τα στοιχεία που ανακτάμε για κάθε κατάστημα είναι το όνομά του

(title), το email του, το website του, η διεύθυνσή του (address), το σημείο του στο χάρτη (lv_listing_lat, lv_listing_lon), η εικόνα του (image) και η περιγραφή του (description).


```
[
{
        "_website":"",
        "_email":"info@dentrospito.gr",
        "_address":"Δαναϊδών 9, Θεσσαλονίκη",
        "_phone1":"2310 557270",
        "lv_listing_lat":"40.6400773",
        "lv_listing_lng":"22.938924",
        “image”:”http://deltio-paraponon.gr/wp-content/uploads/2016/11 /Δεντρόσπιτο.jpg”,
        "description":"Στο κέντρο της πόλης και ταυτόχρονα τόσο μακριά από αυτήν.
Μέσα στο πάρκο της ΧΑΝΘ θα συναντήσετε ένα χώρο, προσαρμοσμένο απόλυτα στο φυσικό περιβάλλον, με σεβασμό στις ελληνικές πρώτες ύλες. Λαδοτύρι Μυτιλήνης, ρακή Κρήτης, ελαιόλαδο Χαλκιδικής, κρασιά Φλώρινας, τσίπουρο Παγγαίου",
"title":"Δεντρόσπιτο",
},
{
        "_website" : "",
        "_email" : "",
        "_address" : " Δημητρίου Γούναρη 23  ",
        "_phone1" : "2310 557270",
        "lv_listing_lat" : "40.6340334",
        "lv_listing_lng" : "22.088412",
        “image”:”http://deltio-paraponon.gr/wp-content/uploads/2016/11 /MrJones.jpg”
        "description":" Ένας ζεστός και ιδιαίτερος χώρος με δική του προσωπικότητα, φτιαγμένος με πολύ μεράκι. Ανοιχτό από νωρίς το πρωί, έχει φιλοξενήσει από εκθέσεις ζωγραφικής μέχρι και jazz βραδιές. Αν και έχει φανατικούς θαμώνες, διαρκώς προστίθεται καινούργιος κόσμος στη συντροφιά του Mr Jones. Επισκεφτείτε το για τον πρωινό σας καλοψημένο καφέ, σερβιρισμένο σε ατομικό μπρούτζινο σερβίτσιο, με μπισκοτάκι. Τι καλύτερο από ένα αρωματικό ξεκίνημα της ημέρας. ",
"title":"Mr. Jones",
}, …
]
```


Οι υπόλοιπες κλήσεις που υποστηρίζει το API μέχρι στιγμής είναι

Η ανάκτηση όλων των χρηστών του συστήματος,

```
http://www.deltio-paraponon.gr/iserv_api/index.php?method=get_all_users
```

Η ανάκτηση των ειδήσεων της ιστοσελίδας, με παράμετρο την κατηγορία τους,

```
http://www.deltio-paraponon.gr/iserv_api/index.php?method=get_news_by_category&category=' + categoty
```

Η ανάκτηση της ημερομηνία που έγινε η τελευταία ανανέωση στην ιστοσελίδα,

```
http://www. deltio-paraponon.gr/iserv_api/index.php?method=last_update
```

Η τελευταία κλήση στο API αφορά την αποστολή μηνυμάτων στην ιστοσελίδα,

```
http://www.deltio-paraponon.gr/iserv_api/index.php?method=send_message&name=' + name + '&email=' + email + '&subject=' + subject + '&message=' +message
```

Οι απαντήσεις στα παραπάνω ερωτήματα γίνονται με ανάλογο τρόπο όπως το πρώτο response που παρουσιάσαμε.

6.6.3 Παρουσίαση κομματιών κώδικα

Σύνδεση στη βάση δεδομένων

```
    protected $servername = 'localhost';
    protected $username = 'deltio_hlpdbus';
    protected $password = 'vr3sdbp@n3lhlpd1$@dm';
    protected $dbname = 'deltio_hlpdbis';
    protected $conn;

    // Create connection
    public function __construct () {
        $this->conn = mysqli_connect($this->servername, $this->username , $this->password, $this->dbname);

        if ($this->conn->connect_error) {
            die("Connection failed: " . $this->conn->connect_error);
        }
        mysqli_set_charset($this->conn,"utf8");
    }
```

Η σύνδεση στη βάση δεδομένων γίνεται χρησιμοποιώντας την μέθοδο mysqli_connect και χρησιμοποιώντας τα στοιχεία σύνδεσης που ορίσαμε πάνω από τη μέθοδο. Η σύνδεση γίνεται στο δομητή της κλάσης ConnectToDB. Επίσης με την mysqli_set_charset μετατρέπουμε το επιστρεφόμενα δεδομένα από τη βάση δεδομένων σε Utf-8 ώστε να μην έχουμε πρόβλημα στην κωδικοποίηση των ελληνικών χαρακτήρων.

Routing σύστημα

```
if (isset($_GET['method'])) {
    $method = $_GET['method'];

}

$testCon = new Listings();
$users = new Users();
$news = new News();
$contact = new Contact();

switch ($method) {
    case "all":
        $testCon->getAllListings();
        break;
    case "last_update":
        $testCon->getLasListingsUpdate();
        break;
    case "get_listing_by_category":
        if (isset($_GET['category'])) {
            $category = $_GET['category'];
        }
        $testCon->getListingsByCategory($category);
        break;
    case "get_listing":
        if (isset($_GET['id'])) {
            $id = $_GET['id'];
        }
        $testCon->getListing($id);
        break;
    case "get_all_users":
        $users->getAllUsers();
        break;
    case "get_news_by_category":
        if (isset($_GET['category'])) {
            $category = $_GET['category'];
        }
        $news->getNewsByCategory($category);
        break;
    case "send_message":
```

```
$name = $email =$subject = $message = null;
if (isset($_GET['name'])) {
   $name = $_GET['name'];
}
if(isset($_GET['email'])){
   $email = $_GET['email'];
}
if(isset($_GET['subject'])){
   $subject = $_GET['subject'];
}
if(isset($_GET['message'])){
   $message = $_GET['message'];
}
$contact->sendEmail($name,$email,$subject,$message);
break;
```

Με τον παραπάνω κώδικα δείχνουμε πως γίνονται οι ανακατευθύνεις εντός του συστήματος. Αρχικά αποθηκεύουμε σε μία μεταβλητή τη μέθοδο με την οποία καλεί ο χρήστης το Web Service και έπειτα ανάλογα με τη μέθοδο καλούμε το αντικείμενο που είναι υπεύθυνο για να ολοκληρώσει το κάθε αίτημα.

Query για την ανάκτηση των καταστημάτων

```
SELECT wp_posts.ID, wp_posts.post_title, wp_posts.post_content, wp_postmeta.meta_key, wp_postmeta.meta_value, wp_terms.name, wp_terms.term_id
            FROM wp_posts
               INNER JOIN wp_postmeta ON wp_posts.ID = wp_postmeta.post_id INNER JOIN wp_term_relationships ON wp_posts.ID = wp_term_relationships.object_id INNER JOIN wp_terms ON wp_term_relationships.term_taxonomy_id = wp_terms.term_id
                  WHERE wp_posts.post_type =  "lv_listing"
                     AND (
                     wp_postmeta.meta_key =  "_email"
                     OR wp_postmeta.meta_key =  "_address"
                     OR wp_postmeta.meta_key =  "_phone1"
                     OR wp_postmeta.meta_key =  "lv_listing_lat"
                     OR wp_postmeta.meta_key =  "lv_listing_lng"
                     OR wp_postmeta.meta_key =  "_website"
                     OR wp_postmeta.meta_key =  "_thumbnail_id"
                     )

                     AND(
                           wp_term_relationships.term_taxonomy_id = 98
                           OR  wp_term_relationships.term_taxonomy_id  = 297
```

```
                    OR  wp_term_relationships.term_taxonomy_id  =
141
                    OR  wp_term_relationships.term_taxonomy_id  =
117
                    OR  wp_term_relationships.term_taxonomy_id  =
142
                    OR  wp_term_relationships.term_taxonomy_id  =
144
                )

            ORDER                                   BY
wp_term_relationships.term_taxonomy_id,wp_posts.post_title
```

Το παραπάνω ερώτημα το εκτελούμε στη βάση δεδομένων όταν θέλουμε να ανακτήσουμε τα καταστήματα. Όπως βλέπουμε είναι ένα περίπλοκο query και αυτό οφείλετε στον τρόπο με τον οποίο αποθηκεύονται τα δεδομένα στη βάση του Wordpress.

6.6.4 Πλεονεκτήματα υλοποίησης Custom Web Service

Ένα βασικό πλεονεκτήματα της χρήσης δικού μας web service είναι ότι δε δεσμευόμαστε από το Wordpress. Πιο συγκεκριμένα, το api μπορεί να λειτουργεί ανεξάρτητα από τις ανανεώσεις που γίνονται στο λογισμικό. Ο μόνος λόγος που μπορεί να «σπάσει» η λειτουργία του service είναι με κάποια ανανέωση, να γίνει αλλαγή στη δομή της βάσης δεδομένων του λογισμικού. Αυτό όμως είναι σπάνιο να γίνει οπότε διαφυλάσσετε η ομαλή λειτουργεία του service.

Επίσης βασικό πλεονέκτημα είναι ότι εφόσον έχει σχεδιαστεί από εμάς, δεν περιοριζόμαστε στην ανάπτυξη όποιον επιπρόσθετων λειτουργιών θέλουμε να προσθέσουμε σε αυτό.

Τέλος μπορούμε να πούμε ότι ένα custom web service θα ήταν πιο γρήγορο από κάποιο api που θα βασιζόταν στο Wordpress. Αυτό γίνεται γιατί το custom web service κάνει αποκλειστικά την λειτουργία που σχεδιάστηκε να κάνει και δεν χρειάζεται να φορτώσει όλα τα στοιχεία του Wordpress.

6.7 Mobile Application

Η ανάπτυξη της mobile application έγινε με τον υβριδικό τρόπο ανάπτυξης, χρησιμοποιώντας το Ionic Framework για την υλοποίηση και το Adobe

Phonegap για να παραχθεί το τελικό apk. Στη συνέχεια θα δείξουμε τον τρόπο που δομήθηκε ο κώδικας, τις λειτουργίες που υποστηρίζει η εφαρμογή και το πώς μπορεί κάποιος να το μεταφορτώσει στα Application Store για να το κατεβάζουν οι χρήστες.

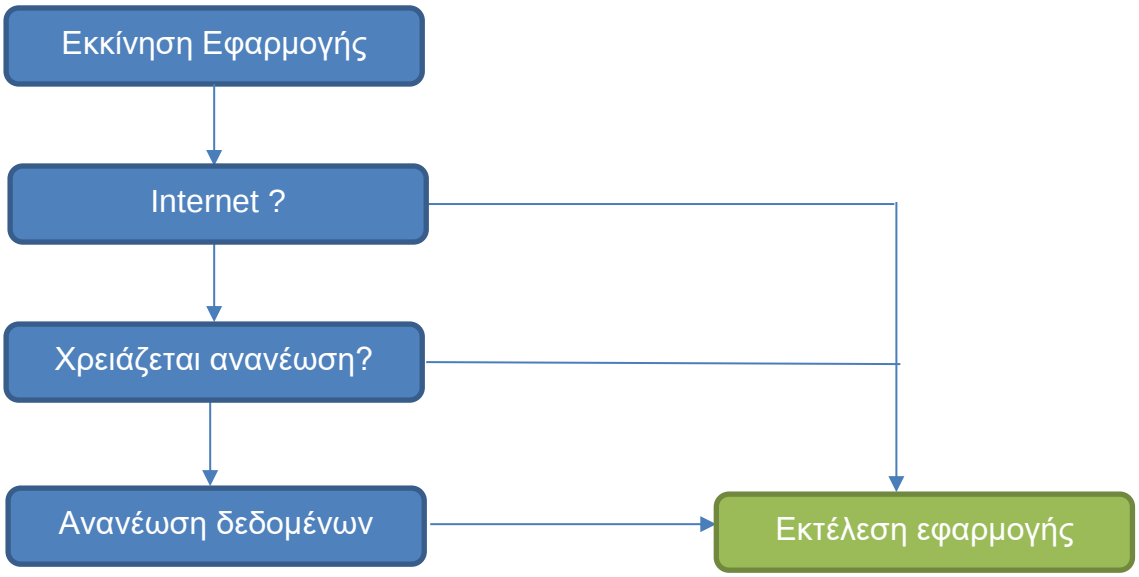
6.7.1 Ανάλυση Mobile App

Η mobile app είναι πλήρως εξαρτημένη από την ιστοσελίδα. Από αυτήν ανακτά όλα τα δεδομένα που χρειάζεται για να λειτουργήσει, μέσω του web service. Για αυτό το λόγο για να μπορέσει να λειτουργήσει χρειάζεται internet ώστε να ανακτά τα δεδομένα τις ιστοσελίδας. Έχουμε αναπτύξει ένα μηχανισμό εντός της εφαρμογής ώστε να μπορεί να λειτουργήσει, έστω και με τα τελευταία δεδομένα που είχε ανακτήσει. Η mobile app διαθέτει τοπική βάση δεδομένων, στην οποία αποθηκεύονται τα δεδομένα ώστε να έχει τη δυνατότητα να λειτουργεί και χωρίς internet.

Η λογική με την οποία γίνεται η ανανέωση των δεδομένων είναι η εξής:

1. Με το που ανοίγει η εφαρμογή, γίνεται έλεγχος εάν η συσκευή είναι συνδεδεμένη στο internet.
2. Εάν η συσκευή δεν είναι συνδεδεμένη στο internet, εκτελείτε η εφαρμογή με τα δεδομένα που υπάρχουν στην τοπική sqlite βάση της εφαρμογής.
3. Εάν η συσκευή έχει internet, γίνεται ερώτηση στο web service πότε ήταν η τελευταία αλλαγή στα δεδομένα της ιστοσελίδας
4. Γίνεται σύγκριση με τα δεδομένα τις τοπικής βάσης της εφαρμογής.
5. Αν, τα δεδομένα της ιστοσελίδας είναι νεότερα από αυτά της εφαρμογής, γίνεται διαγραφή όλων των δεδομένων της τοπικής βάσης και έπειτα εισάγονται ξανά τα νεότερα δεδομένα. Έπειτα η εφαρμογή είναι έτοιμη για τον χρήστη.
6. Σε περίπτωση που στην βάση δεδομένων της ιστοσελίδας δεν υπάρχουν νεότερες αλλαγές, γίνεται μετάβαση στο αρχικό view της εφαρμογής.

Σχηματική απεικόνιση της παραπάνω λογικής:



Εικόνα 3 – Κύκλος ροής ανανέωσης δεδομένων εφαρμογής

6.7.2 Δομή φακέλων και αρχείων κώδικα

Η δομή των αρχείων και των φακέλων καθορίζεται από το τη λογική που ορίζει το Ionic Framework. Η εφαρμογή μας έχει την παρακάτω δομή φακέλων και αρχείων εντός του αρχικού φάκελου, η οποία ορίζεται από το Ionic Framework. Αναλυτικότερα έχουμε:

- plugins: σε περίπτωση που η εφαρμογή χρησιμοποιεί κάποια phonegap plugins, αυτά κατά την εγκατάστασή του τοποθετούνται σε αυτόν το φάκελο.
- scss: είναι μία εξέλιξη του css και σε περίπτωση που η εφαρμογή χρησιμοποιεί scss οι σχετικές βιβλιοθήκες τοποθετούνται εκεί.
- www: εδώ γίνεται η βασική ανάπτυξη της εφαρμογής. Σε αυτόν τον φάκελο έχουμε όλο το js κώδικα τους controllers της εφαρμογής και τα services που χρησιμοποιεί η εφαρμογή.

Αναλυτικότερα για τον φάκελο www στον οποίο υπάρχει η βασική δομή της εφαρμογής.

Στον φάκελο αυτό υπάρχουν οι φάκελοι css, στον οποίο υπάρχουν τα stylesheet της εφαρμογής, fonts σε περίπτωση που η εφαρμογή χρησιμοποιεί extra fonts, img εικονίδια – εικόνες της εφαρμογής, js εδώ υπάρχουν όλα τα controllers και τα services της εφαρμογής, lib εδώ έχουμε όλες τις βιβλιοθήκες

του framework, res σε αυτό το φάκελο έχουμε κάποιες εικόνες που τις χρησιμοποιεί η εφαρμογή, ποιο συγκεκριμένα εδώ έχουμε τις εικόνες που χρησιμοποιεί το splashscreen της εφαρμογής, templates εδώ έχουμε τον html κώδικα της εφαρμογής, δηλαδή την εμφάνιση της εφαρμογή.

Τα αρχεία που υπάρχουν στο βασικό φάκελο είναι τα icon.png: το εικονίδιο της εφαρμογής, αυτό που θα εμφανίζεται και στο μενού της συσκευής, index.html το αρχείο στο οποίο γίνεται η εισαγωγή στην εφαρμογή και γίνονται link όλα τα αρχεία που χρειάζεται η εφαρμογή, screen.jpg: η βασική εικόνα του splashscreen της εφαρμογής, deltio.db: η τοπική sqlite βάση δεδομένων της εφαρμογής.

6.7.3 MVC σχεδίαση της εφαρμογής

Η εφαρμογή χρησιμοποιεί MVC σχεδίαση και για κάθε ενότητα της εφαρμογής (αρχικό view, λίστα καταστημάτων, λεπτομέρειες καταστήματος κ.α.) έχουμε ένα view και ένα controller.

Στην εφαρμογή έχουμε κάνει τον εξής διαχωρισμό. Εντός στο φάκελο js έχουμε τον φάκελο controllers στον οποίο σε κάθε αρχείο έχουμε έναν controller.

Τα views της εφαρμογής είναι τοποθετημένα στο φάκελο templates και το καθένα συνδέεται με κάποιο controller μέσω του routing της εφαρμογής.

Στην MVC σχεδίαση του Ionic Framework αλλάζει ο ρόλος του model και αντικαθιστάτε με τα services. Τα services είναι αυτόνομες λειτουργίες οι οποίες μπορούν να εκτελεστούν από τος controllers εάν οριστεί το service στα dependencies του controller. Η λειτουργία των services διαφέρει από αυτήν των models στην MVC σχεδίαση, αλλά είναι πλέον κατάλληλη για την ανάπτυξη εφαρμογών με τη χρήση του Ionic Framework.

6.7.4 Ανάλυση κώδικά και λογική κώδικα εφαρμογής

Βασικά αρχεία εφαρμογής

www/index.php

Σε αυτό το αρχείο γίνεται η εισαγωγή στην εφαρμογή. Εδώ κάνουμε link όλα τα αρχεία js και css που χρειάζεται η εφαρμογή για να λειτουργήσει. Επίσης εδώ έχουμε και τον κώδικα στον οποίο εσωτερικά εκτελείτε όλη η εφαρμογή:

```
<body ng-app="starter">
    <ion-nav-view></ion-nav-view>
</body>
```

www/js/app.js

Σε αυτό το αρχείο γίνεται ο βασικός ορισμός της εφαρμογής, ορίζουμε plugins και dependences που χρειάζεται η εφαρμογή για να εκτελεστεί. Επίσης σε αυτό το αρχείο ορίζουμε και το routing της εφαρμογής.

Ορισμός της εφαρμογής και dependences

```
var deltio = angular.module('starter', ['ionic', 'slugifier', 'underscore', 'ngMap',
'ngCordova', 'uiGmapgoogle-maps'])
```

Με τον παραπάνω κώδικα ορίζουμε ότι τη μεταβλητή deltio θα αναφερόμαστε στην εφαρμογή. Με το angular.module δημιουργούμε την εφαρμογή με όνομα starter και σε πίνακα ορίζουμε τα dependences της εφαρμογής.

Ορισμός global ενεργειών με το που ανοίξει η εφαρμογή

```
deltio.run(function($ionicPlatform,$rootScope,$ionicConfig,$state) {
        $ionicPlatform.registerBackButtonAction(function (event) {
                if($state.current.name=="app.dash"){
                        navigator.app.exitApp(); //<-- remove this line to disable the exit
                }
                else {
                        navigator.app.backHistory();
                }
        }, 100);

        $rootScope.$on("$stateChangeSuccess", function(event, toState, toParams,
fromState, fromParams){
```

```
                if(toState.name.indexOf('app.orgcategories') > -1) {
                        // Restore platform default transition. We are just hardcoding
android transitions to auth views.
                        $ionicConfig.views.transition('platform');
                        // If it's ios, then enable swipe back again
                        if(ionic.Platform.isIOS()){
                                $ionicConfig.views.swipeBackEnabled(true);
                        }
                        console.log("enabling swipe back and restoring transition to
platform default", $ionicConfig.views.transition());
}});});
```

Σε αυτή τη μέθοδο ορίζουμε τις ενέργειες που θέλουμε να υπάρχουν σε όλη την εφαρμογή. Για παράδειγμα η πρώτη μέθοδος ορίζει τη λειτουργία του backbutton του κινητού και πιο συγκεκριμένα ορίζει ότι αν βρισκόμαστε στο αρχικό view και πατηθεί το backbutton τότε κλείνει η εφαρμογή, εάν δεν είμαστε στο αρχικό view ανοίγει το προηγούμενο view που ήταν ο χρήστης.

Routing εφαρμογής

```
deltio.config(function($stateProvider, $urlRouterProvider, $sceDelegateProvider) {

  $sceDelegateProvider.resourceUrlWhitelist ( [ 'self' , 'http://www.deltio-
paraponon.gr/*', 'https://maps.google.com/maps/api/*']);

  $stateProvider

  .state('app', {
    url: '/app',
    abstract: true,
    templateUrl: 'templates/menu.html',
  })

  .state('app.dash', {
    url: '/dash',
    views: {
      'menuContent': {
        templateUrl: 'templates/dash.html',
        controller: 'DashController'
      }
    }
  })

  .state('app.listingcategories', {
    url: '/listingcategories',
    views: {
      'menuContent': {
        templateUrl: 'templates/listing-categories.html',
        controller: 'ListingCategoriesController'
```

```
        }
      }
   })

   .state('app.listings', {
      url: "/ listings/:categoryId/:categoryName",
      views: {
         'menuContent': {
            templateUrl: "templates/ listings.html",
            controller: ' ListingsController'
         }
      }
   })
…

   $urlRouterProvider.otherwise('/app/dash');

})
```

Με τον παραπάνω κώδικα γίνεται το routing της εφαρμογής. Με την πρώτη γραμμή κώδικα $sceDelegateProvider.resourceUrlWhitelist (['self' , 'http://www.deltio-paraponon.gr/*', 'https://maps.google.com/maps/api/*']); ορίζουμε τις διευθύνσεις τις οποίες επιτρέπει η εφαρμογή να γίνονται κλήσεις χωρίς να τις μπλοκάρει. Οι διευθύνσεις που ορίζουμε στη δική μας εφαρμογή είναι το deltio-paraponon.gr στο οποίο βρίσκεται το web service μας και η άλλη διεύθυνση είναι των maps της google καθώς η εφαρμογή μας χρησιμοποιεί τη συγκεκριμένη υπηρεσία για να εμφανίζει σε χάρτη τα καταστήματα.

Στη συνέχεια ορίζουμε τα views της εφαρμογής. Αρχικά στο state ‘app’ ορίζουμε το μενού που θα εμφανίζεται σε ολόκληρη την εφαρμογή. Έπειτα στο state 'app.dash' ορίζουμε το αρχικό view της εφαρμογής, με τις παραμέτρους templateUrl ορίζουμε το view template που χρησιμοποιεί το συγκεκριμένο state και με τη παράμετρο controller ορίζουμε το js controller αυτού του state.

Αντίστοιχα με τον παρακάτω κώδικα ορίζονται και τα υπόλοιπα states και τα template views και controllers τους.

Controllers εφαρμογής

www/js/controllers/dash.js

```
deltio.controller(
        'DashController',
```

```
        ['$scope', '$ionicPlatform', 'ListingsService', '$ionicLoading' , 'UpdateService', function ($scope, $ionicPlatform, ListingsService, $ionicLoading, UpdateService) {

    $ionicLoading.show({
        content: 'Loading',
        animation: 'fade-in',
        showBackdrop: true,
        maxWidth: 200,
        showDelay: 0
    });

    $ionicPlatform.ready(function() {

        //UpdateService.updateLastUpdateDay();
        //check if database is created
        ListingsService.checkIfDBHasValues().then(function(resp){
            //if there is a result, db exists
            if(resp){
                //if db is created, check if db need to update
                UpdateService.checkIfNeedToUpdate().then(function(resp){
                    //if update needed - update db
                    if(resp){
                        UpdateService.updateDB().then(function(){
                            $ionicLoading.hide();
                        });
                    }
                    //if update not need - continue
                    else{
                        $ionicLoading.hide();
                    }
                });
            }
            //if there db is not created  create db
            else{
                UpdateService.updateDB().then(function(){
                    $ionicLoading.hide();
                });
            }
        });
    });

}]);
```

Ο παραπάνω κώδικας είναι ο controller του αρχικού view της εφαρμογής. Καλώντας τη μέθοδο controller της μεταβλητής deltio δημιουργούμε τον controller με όνομα 'DashController' και στον διπλανό πίνακα ορίζουμε τα dependences θέλουμε να έχουμε πρόσβαση στον controller.Οι δύο λειτουργίες

που εκτελούνται από τον controller είναι στο πρώτο block κώδικα εμφανίζουμε ένα loader ώστε να εμφανίζεται μέχρι η εφαρμογή να είναι έτοιμη για τον χρήστη. Στο δεύτερο block κώδικα γίνεται η λειτουργία ανανέωσης των δεδομένων, χρησιμοποιώντας ένα από τα services που έχουμε δημιουργήσει για την εφαρμογή. Τα services θα τα αναλύσουμε παρακάτω.

www/js/controllers/listings.js

```
deltio.controller('ListingsController',
['$scope', '$stateParams' , 'ListingsService', function ($scope,
$stateParams,  ListingsService) {

   $scope.listings = [];
   $scope.categoryName = $stateParams.categoryName;
   $scope.categoryId = $stateParams.categoryId;


ListingsService.getListingsByCategory($stateParams.categoryId).then(funct
ion(resp){
      $scope.listings=resp;
   });
}]);
```

Εδώ βλέπουμε τον controller της λίστας των καταστημάτων ανά κατηγορία. Το όνομα του controller είναι 'ListingsController' και οι λειτουργίες που εκτελούνται είναι: αρχικά ορίζουμε ένα πίνακα listings χρησιμοποιώντας την παράμετρο $scope ώστε να έχει πρόσβαση το view σε αυτά τα δεδομένα, έπειτα ανακτούμε το όνομα και το id της κατηγορίας που επέλεξε ο χρήστης και τέλος χρησιμοποιώντας το ListingsService γεμίζουμε με δεδομένα τον πίνακα ώστε να τα εμφανίζουμε στο view.

www/js/controllers/listings-map.js

```
deltio.controller('ListingsMapCtrl',  function($scope,  uiGmapGoogleMapApi,
$stateParams, $ionicHistory) {

   var lat = $stateParams.lat;
   var lon = $stateParams.lon;
   var title = $stateParams.title;

   // Define variables for our Map object
   $scope.markers = [];

   uiGmapGoogleMapApi.then(function(maps) {
      $scope.map     = { center: { latitude: lat, longitude: lon }, zoom: 15 };
      $scope.options = { scrollwheel: false };
```

```
        $scope.markers.push({
            latitude: lat,
            longitude: lon,
            title: title,
            id: 1,
            icon: "http://vresvoitheia.gr/iserv_api/marker.png"
        });

    });

    $scope.myGoBack = function() {
        $ionicHistory.goBack();
    };
});
```

Το 'ListingsMapCtrl' είναι υπεύθυνο για τη παρουσίαση των καταστημάτων σε χάρτη. Σε αυτό το controller ορίζουμε το χάρτη χρησιμοποιώντας το uiGmapGoogleApi plugin και έπειτα δημιουργούμε markers για το κάθε κατάστημα ξεχωριστά. Έπειτα τα δεδομένα αυτά εμφανίζονται στο view που είναι συνδεδεμένο με αυτό το controller.

Υπόλοιπα controllers

Με τον ίδιο τρόπο ορίζουμε και τα υπόλοιπα controllers της εφαρμογής. Δεν υπάρχει λόγος να τα αναλύσουμε όλα ένα προς ένα, καθώς και τα υπόλοιπα controllers εκτελούν παρόμοιες λειτουργίες.

Views εφαρμογής

www/templates/dash.html

```
<ion-view view-title="" class="feeds-categories-view"s>
    <ion-nav-bar class="bar-header bar-dark">
        <ion-nav-buttons side="left">
            <!-- Toggle left side menu -->
            <button menu-toggle="left" class="button button-icon icon ion-navicon"></button>
        </ion-nav-buttons>
        <ion-nav-title>
            Δελτίο Παραπόνων App
        </ion-nav-title>
        <ion-nav-buttons side="right">
            <a nav-clear menu-close ui-sref="app.dash">
                <img src="img/logo-small.png"/>
            </a>
```

```
      </ion-nav-buttons>
   </ion-nav-bar>
   <ion-content class="body_custom">
      <div class="item" style="background-color: inherit; border: 0px none;">
         <img style="background-color: rgba(255, 242, 225, 0.68); padding: 16px; border-radius: 61px; margin-top: 60px;" class="full_container_image" src="img/logo.png"/>
      </div>
      <div class="item" style="z-index: 5;white-space: nowrap; font-size: 18px; line-height: 28px; font-weight: 700; color: rgb(255, 255, 255); font-family: Roboto; background-color: rgb(79, 149, 141); padding: 5px 18px 9px 0px; border-radius: 0px 0px 61px; visibility: inherit; transition: none 0s ease 0s; margin: 0px; letter-spacing: 0px; min-height: 0px; min-width: 0px; max-height: none; max-width: 341px;"> Αξιολογήσεις καταστημάτων </div>

      <div class="item" style="z-index: 5; white-space: nowrap;
   font-size: 18px; line-height: 22px; font-weight: 700; color: rgb(255, 255, 255); font-family: Roboto; background-color: rgb(234, 192, 134); padding: 3px 5px 9px 30px; visibility: inherit; transition: none 0s ease 0s; margin: 37px 0px 0px; letter-spacing: 0px; min-height: 0px; min-width: 0px; max-height: none; max-width: 355px; float: right; border-radius: 0px 0px 0px 76px;"> Ηλεκτρονικό κιτίο παραπόνων!?</div>
   </ion-content>
</ion-view>
```

Σε αυτό το αρχείο είναι το αρχικό view της εφαρμογής. Με το που εκτελείτε το state που συμπεριλαμβάνει αυτό το view ο κώδικας του φορτώνεται εντός του <ion-nav-view></ion-nav-view> που ορίζεται στο index.html. Σε αυτό το view έχουμε ένα στατικό περιεχόμενο με μία εικόνα και κάποιους τίτλους. Επίσης έχουμε και ένα loader το οποίο εμφανίζεται μέχρι να ετοιμαστούν τα δεδομένα της εφαρμογής για εκτέλεση. Έπειτα το Loader αποκρύπτεται από τον controller του συγκεκριμένου view όπως αναφέραμε παραπάνω.

www/templates/listings-categories.html

```
<ion-view class="category-feeds-view">
   <ion-nav-buttons side="right">
      <a href="#/app/listings " class="return-button">
         <i class="ion-ios-undo"></i>
      </a>
      <a nav-clear menu-close ui-sref="app.dash">
         <img src="img/logo-small.png"/>
      </a>
   </ion-nav-buttons>
   <ion-nav-title>
      <span>{{categoryName}}</span>
   </ion-nav-title>
   <ion-content>
```

```
      <div class="bar bar-header item-input-inset">
         <input type="text" placeholder="Αναζήτηση..." ng-model="search"
            style="width:100%; padding-left:11px;" />
      </div>
      <div class="list category-feeds">
         <a ng-repeat="listing in listings | filter:search" class="item item-icon-
right"
            ui-sref="app.listing({listing_id:                    listing.id,category_id:
categoryId,category_name: categoryName})">
            <div class="thumbnail-outer">
               <pre-img ratio="_1_1" helper-class="">
                  <!--img ng-if="listing.image != 'undefined' " class="thumbnail"
ng-src="{{/*listing.image*/}}" spinner-on-load-->
                  <img class="thumbnail" ng-src="img/default.jpg" spinner-on-
load>
               </pre-img>
            </div>
            <div>
               <span class="title">{{listing.title | cut:true:35:' ...'}}</span>
               <!--p class="description">{{/*source.description*/}}</p-->
            </div>
            <i class="icon ion-arrow-right-c"></i>
         </a>
      </div>
   </ion-content>
</ion-view>
```

Εδώ έχουμε το view που εμφανίζουμε την λίστα με τα καταστήματα. Αρχικά εμφανίζουμε τα navigation buttons που έχουμε στο συγκεκριμένο view(backbutton και menu button) έπειτα έχουμε τον τίτλο του view και τέλος δημιουργούμε τη λίστα χρησιμοποιώντας το ng-repeat directive του AngularJs. Στο ng-repeat δίνουμε τη λίστα των καταστημάτων που δημιουργήσαμε στο controller κα με αυτό τον τρόπο δημιουργείτε ένα item της λίστα για κάθε κατάστημα.

www/templates/map-listings.html

```
<ion-view class="category-feeds-view">
   <ion-nav-buttons side="right">
      <a href="#/app/listings" class="return-button">
         <i class="ion-ios-undo"></i>
      </a>
      <a nav-clear menu-close ui-sref="app.dash">
         <img src="img/logo-small.png"/>
      </a>
   </ion-nav-buttons>
```

```
    <ion-nav-title>
      <span>{{categoryName}}</span>
    </ion-nav-title>
    <ion-content>
      <div class="bar bar-header item-input-inset">
        <input type="text" placeholder="Αναζήτηση..." ng-model="search"
          style="width:100%; padding-left:11px;" />
      </div>
      <div class="list category-feeds">
        <a ng-repeat="listing in listings | filter:search" class="item item-icon-
right"
          ui-sref="app.organization({listing_id:          listing.id,category_id:
categoryId,category_name: categoryName})">
          <div class="thumbnail-outer">
            <pre-img ratio="_1_1" helper-class="">
              <!--img ng-if="listing.image != 'undefined' " class="thumbnail"
ng-src="{{/*listing.image*/}}" spinner-on-load-->
              <img class="thumbnail" ng-src="img/default.jpg" spinner-on-
load>
            </pre-img>
          </div>
          <div>
            <span class="title">{{listing.title | cut:true:35:' ...'}}</span>
            <!--p class="description">{{/*source.description*/}}</p-->
          </div>
          <i class="icon ion-arrow-right-c"></i>
        </a>
      </div>
    </ion-content>
</ion-view>
```

Σε αυτό το view εμφανίζουμε τα καταστήματα κάποιας κατηγορίας που επέλεξε ο χρήστης σε χάρτη με κάποια markers τα οποία δημιουργήθηκαν στον controller. Επίσης σε αυτό το view έχουμε και μία λίστα με αυτά τα καταστήματα ώστε ο χρήστης να μπορεί να κάνει και αναζήτηση σε αυτά.

Υπόλοιπα views της εφαρμογής

Τα υπόλοιπα views έχουν παρόμοιες λειτουργείες και ως εκ τούτου δεν θα συνεχίσουμε την ανάλυση καθενός ξεχωριστά.

Servicies εφαρμογής

Τα services που έχουμε αναπτύξει για την εφαρμογή είναι τα εξής, DB το οποίο είναι υπεύθυνο για τη σύνδεση και τις κλήσεις με την τοπική βάση δεδομένων, ListingsService το οποίο σχετίζεται με τις λειτουργίες που

υποστηρίζει η εφαρμογή σε σχέση με την τοπική βάση δεδομένων, UpdateService εδώ εκτελούνται οι έλεγχοι για το αν η εφαρμογή χρειάζεται ανανέωση στα δεδομένα, UsersService το συγκεκριμένο service χειρίζεται τους χρήστες της εφαρμογής, DeltioParapononService αυτό το service κάνει τις κλήσεις στο web service της ιστοσελίδας.

Θα αναλύσουμε ενδεικτικά δύο από τα services που αναφέραμε παραπάνω.

www/js/services/DB.js

Επειδή οι λειτουργίες του service είναι αρκετές, θα της αναλύσουμε σταδιακά.

```
deltio.factory('DB', function($q,$cordovaSQLite) { });
```

Ορισμός του service γίνεται με τη μέθοδο factory, η ονομασία του service είναι ‘DB’ και έπειτα ακολουθούν τα dependences του service που σε αυτή τη περίπτωση είναι το $q το οποίο το χρησιμοποιούμε για να κάνουμε κάποιου είδους ασύγχρονες εργασίες και το $cordovaSQLite το οποίο είναι το plugin το οποίο είναι ο driver για να συνδεόμαστε στην τοπική sqlite της εφαρμογής.

```
  var self = this;
  var _db;
```

Ορίζουμε δύο μεταβλητές για να είναι πιο ξεκάθαρο στον κώδικα ότι αναφερόμαστε στην συγκεκριμένη κλάση και στην βάση δεδομένων.

```
  self.db = function () {
      if (!_db) {
          if (window.sqlitePlugin !== undefined) {
              _db = window.sqlitePlugin.openDatabase({ name: "vres.db",
location: 2, createFromLocation: 1 });
          } else {
              // For debugging in the browser
              _db = window.openDatabase("vres.db", "1.0", "Database",
200000);
          }
      }
      return _db;
  };
```

Με τη μέθοδο db δημιουργούμε τη σύνδεση με τη τοπική βάση δεδομένων. Όπως αναφέραμε επειδή η ανάπτυξη της εφαρμογής γίνεται σε browser, κάνουμε τον έλεγχο εάν η εφαρμογή εκτελείτε σε browser ή σε συσκευή smartphone ώστε να γίνει η σύνδεση στην sqlite με τον ανάλογο τρόπο. Σε browser η βάση ανοίγει κατευθείαν με τη μέθοδο openDatabse της javascript ενώ στη συσκευή η σύνδεση γίνεται μέσω του plugin που αναφέραμε.

```
self.executeSql = function (query, parameters) {
    return $cordovaSQLite.execute(self.db(), query, parameters);
};
```

Με τη μέθοδο executeSql εκτελούμε κάποιο query(query) που το περνάμε ως παράμετρο στη μέθοδο μαζί με τις παραμέτρους parameters(parameters) του query. Η εκτέλεση γίνεται με τη χρήση του cordovaSQLite plugin απευθείας στην sqlite database.

```
self.query = function(query, bindings) {
    bindings = typeof bindings !== 'undefined' ? bindings : [];
    var deferred = $q.defer();

    self.executeSql(query, bindings).then(function (res) {
        return deferred.resolve(res);
    }, function (err) {
        return deferred.resolve(false);
    });

    return deferred.promise;
};
```

Την παραπάνω μέθοδο την χρησιμοποιούμε για να καλέσουμε την executeSql. Χρησιμοποιούμε αυτή τη μέθοδο ώστε οι κλήσεις στη βάση δεδομένων να γίνεται με ασύγχρονο τρόπο. Ποιο συγκεκριμένα χρησιμοποιούμε τη μεταβλητή $q και δημιουργούμε μία «υπόσχεση», έπειτα καλούμε τη μέθοδο executeSql χρησιμοποιώντας την υπόσχεση, όταν εκπληρωθεί αυτή η υπόσχεση παίρνουμε τα στοιχεία που μας στέλνει η κλήση στη βάση δεδομένων.

```
self.insertAll = function(listings){

    for (var i=0; i<listings.length;i++) {
        var data = listings[i];
```

```
        var query = "INSERT INTO Listings (id, email, phone1, website, description, image, lv_listing_lat, lv_listing_lng, title, category, category_id, address) VALUES (?,?,?,?,?,?,?,?,?,?,?,?)";
        self.query(query,
            [data['id'], data['_email'], data['_phone1'], data['_website'], data['description'], data['image'], data['lv_listing_lat'], data['lv_listing_lng'], data['title'], data['category'], data['category_id'], data['_address']
            ]
        );
    };
};
```

Σε αυτή τη μέθοδο αρχικά δημιουργούμε το query για κάθε ένα από τα listings έπειτα εκτελούμε ένα - ένα χρησιμοποιώντας τη μέθοδο query του service. Όπως βλέπουμε στον κώδικα "INSERT INTO Listings (id, email, phone1, website, description, image, lv_listing_lat, lv_listing_lng, title, category, category_id, address) VALUES (?,?,?,?,?,?,?,?,?,?,?,?)" έχουμε το SQL query και στις τιμές που είναι να περάσουμε βάζουμε το σύμβολο ?. Όταν είναι να εκτελεστεί το query με το $cordovaSQLite plugin, περνάμε όσες τιμές χρειάζεται το query(δηλαδή όσα σύμβολα ? χρησιμοποιήσαμε) σε πίνακα.

```
self.getItems = function (query, parameters) {
    var deferred = $q.defer();
    self.query(query, parameters).then(function (res) {
        var items = [];
        for (var i = 0; i < res.rows.length; i++) {
            items.push(res.rows.item(i));
        }
        return deferred.resolve(items);
    }, function (err) {
        return deferred.reject(err);
    });

    return deferred.promise;
};
```

Με τη μέθοδο getItems εκτελούμε query το οποίο έχει σαν στόχο να ανακτήσει δεδομένα. Στέλνουμε το σχετικό query και έπειτα τα δεδομένα που μας επιστρέφει η τοπική βάση δεδομένων τα μετατρέπει σε πίνακα.

www/js/services/DeltioParapononService.js

```
deltio.factory('DeltioParapononService', function ($http) {

    // The object that this factory returns
```

```
    return {
        //return feed, parsed as xml
        getAllListings: function(){
            return
$http.get("http://www.vresvoitheia.gr/iserv_api/index.php?method=all").then
(function (response){
                return response.data;
            });
        },
        getAllUsers: function(){
            return
$http.get("http://www.vresvoitheia.gr/iserv_api/index.php?method=get_all_u
sers").then(function (response){
                return response.data;
            });
        },
        getNewsByCategory: function(categoty){
            return
$http.get('http://www.vresvoitheia.gr/iserv_api/index.php?method=get_news
_by_category&category='+categoty).then(function (response){
                return response.data;
            });
        },
        getLastUpdateDay: function(){
            return
$http.get('http://www.vresvoitheia.gr/iserv_api/index.php?method=last_upda
te').then(function (response){
                return response.data;
            });
        },
        sendMessage: function(name,email,subject,message){
            return
$http.get('http://www.vresvoitheia.gr/iserv_api/index.php?method=send_me
ssage&name='+name+'&email='+email+'&subject='+subject+'&message='+
message).then(function (response){
                return response.data;
            });
        }
    };
});
```

Σε αυτό το service κάνουμε τις κλήσεις στο web service της ιστοσελίδας. Καλούμε τις μεθόδους του service χρησιμοποιώντας τη μέθοδο get της μεταβλητής $http (μεταβλητή του ionic framework που χειρίζεται http κλήσεις). Έπειτα επιστρέφομε τα δεδομένα που μας έρχονται από το api.

6.7.5 Compile κώδικα με τη χρήση του Phonegap build

Με το που ολοκληρωθεί η ανάπτυξη του κώδικα, θα πρέπει να παράγουμε το τελικό app το οποίο θα το ανεβάσουμε στα store ώστε να μπορούν να κατεβάσουν την εφαρμογή οι χρήστες smartphone. Όπως έχουμε αναφέρει και παραπάνω το Ionic Framework είναι πλήρως συμβατό το Adobe Phonegap. Γι' αυτό το λόγο εκμεταλλευτήκαμε αυτή τη δυνατότητα και χρησιμοποιήσαμε το 'Phonegap Build για να παράγουμε το τελικό app.

Για να μπορέσει κάποιος να χρησιμοποιήσει τον online compiler, πρέπει να συνδεθεί στην σχετική υπηρεσία. Αυτό γίνεται είτε δημιουργώντας λογαριασμό είτε κάνοντας σύνδεση με το Adobe ID. Αφού συνδεθεί ο χρήστης πρέπει να ετοιμάσει τον κώδικα τον κώδικα για να τον ανεβάσει στην υπηρεσία. Αυτό γίνεται συμπιέζοντας σε μορφή .zip τον κώδικα του φακέλου www μαζί με το αρχείο config.xml στο οποίο ορίζονται οι ρυθμίσεις της εφαρμογής. Επίσης με αυτό το αρχείο δείχνουμε στον online compiler εάν η εφαρμογή χρησιμοποιεί plugins για τα συμπεριληφθούν και αυτά στο τελικό προϊόν. Με το που ολοκληρωθεί η μεταφόρτωση του αρχείου ξεκινάει η διαδικασία του compile του κώδικα και μετά από λίγα δευτερόλεπτα ο χρήστης μπορεί να κατεβάσει το τελικό app. Οι πλατφόρμες που υποστηρίζονται είναι android, ios, windows.

6.7.6 Διάθεσή της mobile app

Για να έχουν οι χρήστες τη δυνατότητα να προμηθευτούν την εφαρμογή θα πρέπει να τη διαθέσουμε μέσω των store του κάθε λειτουργικού συστήματος. Η διαδικασία που χρειάζεται για να διατεθούν οι εφαρμογές είναι περίπου η ίδια για όλα τα stores.

Για να μπορέσει κάποιος χρήστης να ανεβάσει μία εφαρμογή στο Play Store της Google, θα χρειαστεί αρχικά να κάνει λογαριασμό στην ενότητα Developers της Google. Υπάρχει συνδρομή 50$ για αυτή την υπηρεσία και ο χρήστης μπορεί να ανεβάζει όσες εφαρμογές θέλει για πάντα. Αφού ολοκληρωθεί η εγγραφή, ο χρήστης μπορεί να μεταφορτώσει εφαρμογές στον λογαριασμό του και έπειτα να τις διαθέσει στους χρήστες.

Ο χρήστης αρχικά ανεβάζει το .apk αρχείο. Έπειτα δίνει την περιγραφή της εφαρμογής μαζί με κάποιες αντιπροσωπευτικές εικόνες. Έπειτα γίνεται έλεγχος

της εφαρμογής καθώς και των περιγραφών και αν όλοι οι έλεγχοι εκτελεστούν με επιτυχία, η εφαρμογή εμφανίζεται στο Play Store.

Η παραπάνω διαδικασία είναι ανάλογη και στα store των υπόλοιπων λειτουργικών συστημάτων.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Στις μέρες μας όπου οι διαδικτυακές εφαρμογές καθώς και οι εφαρμογές των κινητών τηλεφώνων έχουν τεράστια ανάπτυξη, βλέπουμε ότι υπάρχει η ανάγκη για συνδυασμό τεχνολογιών ώστε να παράγεται κάποιο τελικό προϊόν. Επίσης βλέπουμε να υπάρχουν διάφοροι τρόποι ανάπτυξης για το ίδιο τελικό αποτέλεσμα.

Ο στόχος της δικής μας δουλειάς ήταν να παραδώσουμε μία εφαρμογή στην οποία ο χρήστης θα έχει πρόσβαση και από τον υπολογιστή του και από το κινητό του τηλέφωνο με τη χρήση ίδιων δεδομένων. Για να μπορέσουμε να το πετύχουμε αυτό χρειάστηκε να κάνουμε συνδυασμό 3 αυτόνομων συστημάτων, του Wordpress CMS, ενός custom Web Service API και μίας mobile application για κάθε λειτουργικό σύστημα. Αυτό το αποτέλεσμα θα μπορούσαμε να το πετύχουμε με το συνδυασμό κάποιων διαφορετικών τεχνολογιών από αυτές που χρησιμοποιήσαμε. Επιλέξαμε αυτές τις τεχνολογίες με κριτήριο ότι έχουν τη δυνατότητα να συνδυαστούν μεταξύ τους και λόγο της εμπειρίας που είχαμε πάνω σε αυτές τις τεχνολογίες. Πιθανότατα κάποια άλλη ομάδα προγραμματιστών θα μπορούσε να επιλέξει άλλες τεχνολογίες για να πετύχει τον ίδιο στόχο.

Αυτό που φαίνεται να έχει σημασία στην επιλογή των τεχνολογιών από τους προγραμματιστές, είναι σε πρώτη φάση να ορίζονται οι στόχοι που θέλουμε να επιτύχουμε. Μετά να βλέπουμε ποιες τεχνολογίες μπορούν να μας δώσουν αυτό το αποτέλεσμα. Έπειτα η τελική επιλογή να γίνεται με βάση την εμπειρία που έχουμε σε τεχνολογίες και μεθοδολογίες ανάπτυξης που επιλέχθηκαν ότι μπορούν να εκπληρώσουν τους στόχους που θέσαμε. Με αυτό τον τρόπο θα μπορούμε να πετύχουμε το καλύτερο αποτέλεσμα στο μικρότερο χρονικό διάστημα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ricca, F., & Tonella, P. (2001, July). Analysis and testing of web applications. In *Proceedings of the 23rd international conference on Software engineering* (pp. 25-34). IEEE Computer Society.

Offutt, J. (2002). Quality attributes of web software applications. *IEEE software*, *19*(2), 25-32.

Rossi, G., Pastor, Ó., Schwabe, D., & Olsina, L. (Eds.). (2007). *Web engineering: modelling and implementing web applications*. Springer Science & Business Media.

Baxter, S., & Vogt, L. C. (2002). *U.S. Patent No. 6,356,903*. Washington, DC: U.S. Patent and Trademark Office.

Dan, N., & Brown, A. C. (2003). *U.S. Patent No. 6,560,639*. Washington, DC: U.S. Patent and Trademark Office.

Patel, S. K., Rathod, V. R., & Parikh, S. (2011, December). Joomla, Drupal and WordPress-a statistical comparison of open source CMS. In *Trendz in Information Sciences and Computing (TISC), 2011 3rd International Conference on* (pp. 182-187). IEEE.

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). Web services. In *Web Services* (pp. 123-149). Springer Berlin Heidelberg.

Sumathi, S., & Esakkirajan, S. (2007). Structured Query Language. *Fundamentals of Relational Database Management Systems*, 111-212.

Bowman, J. S., Emerson, S. L., & Darnovsky, M. (1996). *The practical SQL handbook: using structured query language*. Addison-Wesley Longman Publishing Co., Inc.

Spertus, E., & Stein, L. A. (2000). Squeal: a structured query language for the Web. *Computer Networks*, *33*(1), 95-103.

Charland, Andre, and Brian Leroux. "Mobile application development: web vs. native." Communications of the ACM 54.5 (2011): 49-53.

Srini, S., & Venkatraman, S. (2012). Hybrid Mobile Application Development Approaches. Tata Consultancy Services.

Ghatol, R., & Patel, Y. (2012). Beginning phonegap. New York: Apress Media.

Wargo, J. M. (2012). PhoneGap essentials: Building cross-platform mobile apps. Addison-Wesley.

Zibula, A., & Majchrzak, T. A. (2012, April). Cross-platform development using HTML5, jQuery Mobile, and PhoneGap: Realizing a smart meter application. In International Conference on Web Information Systems and Technologies (pp. 16-33). Springer Berlin Heidelberg.

Mahesh, B. R., Kumar, M. B., Manoharan, R., Somasundaram, M., & Karthikeyan, S. P. (2012, December). Portability of mobile applications using phonegap: A case study. In Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012), International Conference on (pp. 1-6). IET.

Camden, R. K. (2015). Apache Cordova in action. Manning Publications Co..

Elmarsi, R., Navathe, S. B. (2007), Θεμελιώδεις αρχές συστημάτων βάσεων δεδομένων

Ramakrishnan, R., Gehrke, J. (2000), Συστήματα διαχείρισης βάσεων δεδομένων

Welling, L., Thomson, L. (2009), Ανάπτυξη web εφαρμογών με PHP και MySQL

P. MacIntyre (2016), Building exceptional sites with Wordpress & Thesis: A php[architect] Guide

Malavolta, I., Ruberto, S., Soro, T. (2015), End Users' Perception of Hybrid Mobile Apps in the Google Play Store

Seung-Ho Lim. (2015), Experimental Comparison of Hybrid and Native Applications for Mobile Systems. International Journal of Multimedia and Ubiquitous Engineering

TD Hedengren. (2012), Smashing Wordpress: Beyond the Blog

Williams B., Damstra D., Stern H. (2015), Professional WordPress: Design and Development

Xanthopoulos, S., Xinogalos, S. (2013), A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications.

Codd, E. F. (1982). Relational database: a practical foundation for productivity. Communications of the ACM, 25(2), 109-117.

Buckles, B. P., & Petry, F. E. (1982). A fuzzy representation of data for relational databases. Fuzzy sets and systems, 7(3), 213-226.

Abiteboul, S., Hull, R., & Vianu, V. (1995). Foundations of databases: the logical level. Addison-Wesley Longman Publishing Co., Inc..

Maier, D. (1983). Theory of relational databases. Computer Science Pr.

Date, C. J., & Darwen, H. (1987). A Guide to the SQL Standard (Vol. 3). New York: Addison-Wesley.

Sumathi, S., & Esakkirajan, S. (2007). Structured Query Language. Fundamentals of Relational Database Management Systems, 111-212.

Harkins, S. S., & Reid, M. W. (2002). Structured query language. In SQL: Access to SQL Server (pp. 1-5). Apress.

Alashqur, A. M., Su, S. Y., & Lam, H. (1989, July). OQL: a query language for manipulating object-oriented databases. In Proceedings of the 15th international conference on Very large data bases (pp. 433-442). Morgan Kaufmann Publishers Inc..

Kim, W. (1990). Introduction to object-oriented databases (Vol. 90). Cambridge, MA: MIT press.

Juneau, J. (2013). The Query API and JPQL. In Java EE 7 Recipes (pp. 447-470). Apress.

Sharma, A., & Barwal, P. N. JOOQ-JAVA OBJECT ORIENTED QUERYING.