ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και Ανάπτυξη Διαδικτυακών Υπηρεσιών Αρχιτεκτονικής REST

Κωνσταντίνος Κοροβέσης, Α.Μ.: 35959
Χρυσόστομος Μανώλης, Α.Μ.: 32637

Εισηγητής:

Εξεταστική Επιτροπή:

Ημερομηνία Εξέτασης:

Δήλωση Συγγραφέα Πτυχιακής Εργασίας

Ο/Η κάτωθι υπογεγραμμένος/η Κωνσταντίνος Κοροβέσης του Ιωάννη, με αριθμό μητρώου 35959 και Χρυσόστομος Μανώλης του Ευαγγέλου με αριθμό μητρώου 32637, φοιτητής/τρια του Τμήματος Μηχανικών Η/Υ Συστημάτων Τ.Ε. του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

Περίληψη

Ο διεθνής οργανισμός W3C ορίζει ως *διαδικτυακή υπηρεσία* ένα σύστημα λογισμικού σχεδιασμένο να υποστηρίζει αλληλεπίδραση μηχανής-προς-μηχανή μέσω ενός δικτύου, μέσω μιας μηχανικά επεξεργάσιμης διεπαφής. Στην παρούσα πτυχιακή εργασία παρουσιάζεται ο σχεδιασμός και η υλοποίηση μιας υπηρεσίας ιστού αρχιτεκτονικής *Representational State Transfer* καθώς και του συστήματος διεπαφής χρήστη που αλληλοεπιδρά μαζί της.

Υλοποιούμε μια υπηρεσία ιστού που επιτρέπει τη δημιουργία, αποθήκευση και ανάκτηση προσωπικών σημειώσεων με χρήση της γλώσσας προγραμματισμού Python και του πλαίσιο ανάπτυξης Django, ένα διαδεδομένο εργαλείο σχεδιασμένο για την ανάπτυξη σύγχρονων υπηρεσιών ιστού. Με τη χρήση προγραμματιζόμενης διεπαφής αρχιτεκτονικής *REST* επιτυγχάνουμε επικοινωνία με την ιστοσελίδα διεπαφής χρήστη, αναπτυγμένη με τη γλώσσα προγραμματισμού Javascript και το πλαίσιο ανάπτυξης AngularJS.

Επιπλέον, με τη χρήση των παραπάνω εργαλείων ανοιχτού πηγαίου κώδικα και προτύπων ταχείας ανάπτυξης εφαρμογών, αναδεικνύουμε τον ρόλο τους στην γρήγορη και ανέξοδη ανάπτυξη σύγχρονων υπηρεσιών ιστού.

Abstract

The international organization W3C defines a *web service* as a software system designed to support machine-to-machine interaction over a network via a machine-processable interface. In the present thesis we present the design and implementation of a *Representational State Transfer* web service as well as a user interface system that interact with it.

We implement a web service that allows the creation, storage and retrieval of personal notes using the Python programming language and the Django development framework, a popular tool designed for the development of modern web services. By using an application programming interface(API) of RESTful architecture we achieve communication with the user interface web client, developed using the programming language Javascript and the AngularJS development framework.

Moreover, by using the aforementioned open source tools and rapid development standards, we highlight their role in fast and inexpensive development of modern web services.

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Εισαγωγή

Παρατηρείται ότι οι περισσότερες διαδικτυακές εφαρμογές λογισμικού τα τελευταία χρόνια εξελίσσονται και τείνουν να υιοθετήσουν χαρακτήρα υπηρεσίας. Ο λόγος είναι ότι μία εφαρμογή παγκόσμιου ιστού η οποία προσανατολίζεται στην υπηρεσία, ή υπηρεσιοστραφής διαδικτυακή εφαρμογή καλύπτει τις απατήσεις που θέτουν τα σύγχρονα συστήματα. Μια διαδικτυακή υπηρεσία εφαρμόζει τεχνολογίες που προσφέρουν ταχύτητα, απλότητα, χαμηλό κόστος και ανεξαρτησία από λειτουργικά συστήματα και πλατφόρμες. Εξυπηρετεί ταυτόχρονα και τους σκοπούς του παρόχου αλλά και του πελάτη που θα καταναλώσει το προϊόν της υπηρεσίας.

Κατά την κατασκευή μιας διαδικτυακής εφαρμογής ο προγραμματιστής πρέπει να ακολουθήσει συγκεκριμένες αρχές, πρότυπα και αρχιτεκτονικές, ώστε το τελικό προϊόν να μπορεί να χαρακτηριστεί ως SOA (Service-Oriented Application) παγκόσμιου ιστού ή αλλιώς Διαδικτυακή Υπηρεσία. Δύο είναι οι βασικότερες αρχές, εκτός των άλλων, που θα πρέπει να ακολουθήσει ο προγραμματιστής κατά τον σχεδιασμό και την υλοποίηση. Η υπηρεσία πρέπει να είναι εύκολα προσβάσιμη μέσω του διαδικτύου ακολουθώντας κλασσικά πρωτόκολλα επικοινωνίας και να επιτευχθεί ο πλήρης διαχωρισμός της εφαρμογής με την διεπαφή, του server με τον client, δηλαδή της ίδιας της υπηρεσίας με την πλατφόρμα που θα καταναλώνει προϊόν. Το έργο ουσιαστικά θα αποτελείται από δύο ανεξάρτητα μέρη που δεν παρουσιάζουν περιορισμούς μεταξύ τους και μπορούν ακόμα και να κατασκευαστούν ξεχωριστά. Η τεχνολογία όμως που απαιτείται για την ολοκλήρωση ενός τέτοιου έργου προβληματίζει τον προγραμματιστή.

Σκοπός της εργασίας αυτής είναι να αναδείξει εργαλεία κατασκευής μιας διαδικτυακής υπηρεσίας τα οποία καλύπτουν τις ανάγκες υλοποίησης και εξασφαλίζουν τις αρχές σχεδιασμού της εφαρμογής. Χρησιμοποιώντας συγκεκριμένα εργαλεία γνωστά ως frameworks καταφέρνουμε να δημιουργήσουμε δύο αυτόνομα επιμέρους κομμάτια της εφαρμογής. Η εφαρμογή θα προσφέρει την δυνατότητα στον χρήστη της να αποθηκεύει κείμενο και ηλεκτρονικές σημειώσεις και να έχει πρόσβαση σε αυτές άμεσα μέσω διαδικτύου. Η εφαρμογή αποτελείται από ένα backend κομμάτι, σχεδιασμένο σε γλώσσα Python και με την χρήση του Django Rest framework, και το frontend κομμάτι σε JavaScript με το AngularJS. Ακολουθώντας σύγχρονες τεχνικές και κλασικά πρότυπα και πρωτόκολλα επικοινωνίας εξασφαλίζουμε την ανεξαρτησία των δύο επιμέρους στοιχείων της εφαρμογής και διασφαλίσαμε της σωστή επικοινωνία μεταξύ τους.

Η παρακάτω εργασία αποτελείται από δύο μέρη και ο στόχος της είναι να αναλύσει και να παρουσιάσει την τεχνολογία πίσω από την υπηρεσία αυτή και να επεξηγήσει την χρήση των εργαλείων που χρησιμοποιήθηκαν. Στο πρώτο μέρος γίνεται μια θεωρητική ανάλυση της τεχνολογίας, των αρχιτεκτονικών, των προτύπων και των αρχων των

Διαδικτυακών υπηρεσιών και μία αναλυτική παρουσίαση των εργαλείων ή frameworks που χρησιμοποιήθηκαν. Στο δεύτερο μέρος παρουσιάζουμε βασικά στάδια κατασκευής της συγκεκριμένης εφαρμογής, τα βήματα που ακολουθήθηκαν και την ανάλυση του κυρίως μέρους του κώδικα της εφαρμογής.

Κεφάλαιο 2

Διαδικτυακές Υπηρεσίες

2.1 Διαδικτυακές Υπηρεσίες

Η διαδικτυακή υπηρεσία είναι μια εφαρμογή με χαρακτήρα υπηρεσίας η οποία παρέχει μια διεπαφή προσβάσιμη μέσω διαδικτύου για τη λειτουργία της. Η εφαρμογή έχει χτιστεί χρησιμοποιώντας καθορισμένες τεχνολογίες του διαδικτύου. Με άλλα λόγια αν μία εφαρμογή μπορεί να προσεγγιστεί μέσω δικτύου χρησιμοποιώντας πρωτόκολλα όπως HTTP, SMTP τότε αποτελεί διαδικτυακή υπηρεσία. Ουσιαστικά, οι διαδικτυακές υπηρεσίες υφίστανται πολλά χρόνια στο διαδίκτυο αποτελώντας βασική αρχή στην εξέλιξη του.

Όπως κάθε άλλη υπηρεσία έτσι και η διαδικτυακή υπηρεσία βασίζεται στο μοντέλο πελάτη – εξυπηρετητή. Συγκεκριμένα βασίζεται στην αρχιτεκτονική η οποία ακολουθεί αυτό το μοντέλο. Ο αιτών (πελάτης) κάνει μια αίτηση για μια συγκεκριμένη δράση στο πλαίσιο μιας συγκεκριμένης υπηρεσίας. Ανάλογα με τον πελάτη και την αίτηση ο πάροχος υπηρεσιών (εξυπηρετητής) ανταποκρίνεται κατάλληλα.

Σχήμα 2.1: Επικοινωνία Πελάτη-Διακομιστή

Η αρχιτεκτονική αυτή στην οποία βασίστηκαν οι διαδικτυακές υπηρεσίες είναι η SOA (Service Oriented Architecture). Η αρχιτεκτονική διαδικτυακών υπηρεσιών (SOA) ακολουθεί την απλή έννοια ενός εντελώς ανοιχτού περιβάλλοντος στο οποίο χρήστες υπηρεσιών ή πελάτες επικοινωνούν με εξυπηρετητές ή παρόχους υπηρεσιών, προκειμένου να έχουν την πλήρη διαχείριση των πόρων που διαθέτει το σύστημα. Μπορούν εύκολα να εντοπίστουν σφάλματα που προκύπτουν και να διορθώθούν ενώ παράλληλα είναι ευκολα διαχειρίσημη η προσθήκη και η αφαίρεση στοιχείων που περιλαμβάνει το σύστημα. Έτσι, σε ένα σύστημα που βασίζει τη λειτουργία του

σε υπηρεσίες, είναι σημαντική η χρήση αυτής της αρχιτεκτονικής προκειμένου να οργανωθεί και να λειτουργήσει με το βέλτιστο δυνατό τρόπο.

Η αρχή που πραγματώνει η SOA εστιάζει στην οργάνωση και τη διαχείριση της επικοινωνίας και της αλληλεπίδρασης μεταξύ διαφορετικών υπηρεσιών καθώς και τη δημιουργία συστημάτων βασισμένα σε υπηρεσίες. Αυτή η αρχή έρχεται σε αντίθεση με την αρχή της αντικειμενοστραφούς αρχιτεκτονικής βάσει της οποίας αναπτύσσονταν αρχικά οι υπηρεσίες αλλά πολύ γρήγορα προέκυψε το πρόβλημα της διαχείρισης και αλληλεπίδρασης των υπηρεσιών.

Ένα σύστημα SOA αποτελείται από διάφορες οντότητες οι οποίες συνεργάζονται μεταξύ τους, και είναι οι παρακάτω:

2.1.1 Πάροχος Υπηρεσίας (Service Consumer)

Είναι η οντότητα η οποία είναι εξουσιοδοτημένη από το δίκτυο να εκτελεί τα αιτήματα των καταναλωτών υπηρεσίας. Αυτός μπορεί να είναι ένας ισχυρός υπολογιστής, μία απλή συσκευή ή οποιοσδήποτε τύπος αιτήματος. Ο φορέας υπηρεσίας μπορεί να δημοσιεύσει τις παρεχόμενες υπηρεσίες του μητρώο υπηρεσίας (Service Registry) προκειμένου αυτές να είναι ορατές στους καταναλωτές υπηρεσία.

2.1.2 Καταναλωτής Υπηρεσίας (Service Provider)

Ο καταναλωτής υπηρεσίας μπορεί να είναι μία εφαρμογή, μία υπηρεσία ή οποιαδήποτε μορφή λογισμικού που χρειάζεται μία υπηρεσία. Είναι η οντότητα που εκκινεί μία υπηρεσία από το σημείο που είναι καταχωρημένη, τη δεσμεύει εγκαθιστώντας μία σύνδεση μεταξύ του καταναλωτή και της υπηρεσίας και εκτελεί τις λειτουργίες της. Ο καταναλωτής της υπηρεσίας μπορεί να την εκτελέσει μέσω της αποστολής ενός αιτήματος το οποίο διαμορφώνεται σύμφωνα με τα πρότυπα που ισχύουν στο σύστημα.

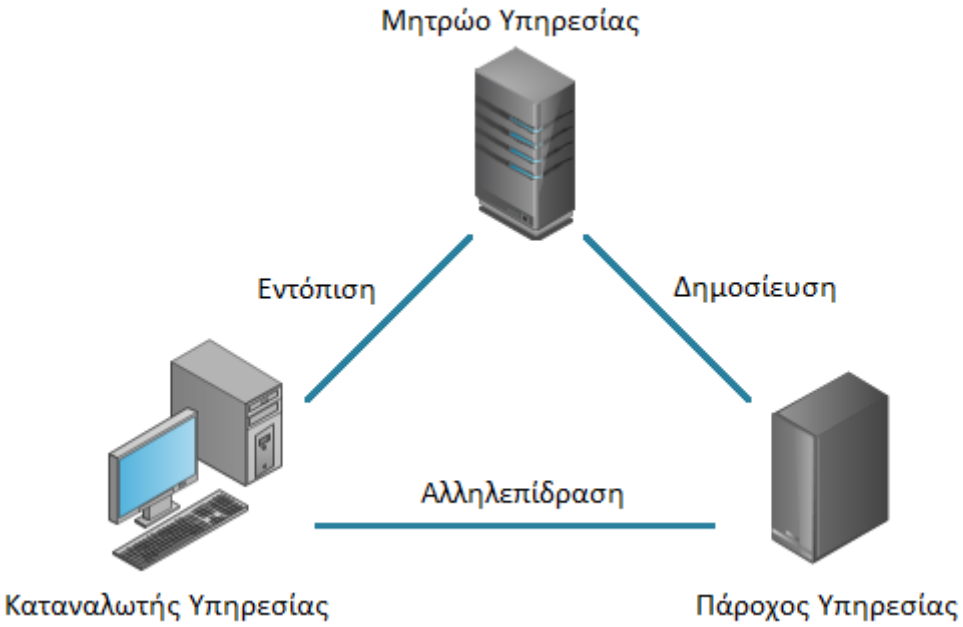
2.1.3 Μητρώο Υπηρεσίας (Service Registry)

Είναι ένα μητρώο δικτύου το οποίο περιέχει τις διαθέσιμες υπηρεσίες από τους παρόχους. Αποθηκεύει στοιχεία που αφορούν τις πληροφορίες των παρεχόμενων υπηρεσιών από τους φορείς, ενώ παράλληλα διαθέτει τις πληροφορίες αυτές στους καταναλωτές υπηρεσιών που πραγματοποιούν αναζήτηση.

Γίνεται ξεκάθαρο τώρα ότι το μοντέλο πελάτη-εξυπηρετητή οδήγησε στην αρχιτεκτονική SOA στην οποία βασίζονται όλες οι διαδικτυακές υπηρεσίες. Το μοντέλο πελάτη-εξυπηρετητή στα συστήματα ν-επιπέδων, στα κατανεμημένα συστήματα, στις εφαρμογές διαδικτυακών υπηρεσιών αντιπροσωπεύει το αποκορύφωμα καθεμίας από αυτές τις αρχιτεκτονικές σε συνδυασμό με το διαδίκτυο.

Οι διαδικτυακές υπηρεσίες είναι μια τεχνολογία που επιτρέπει στις εφαρμογές να επικοινωνούν μεταξύ τους ανεξαρτήτως πλατφόρμας και γλώσσας προγραμματισμού. Ένας client μπορεί για παράδειγμα να είναι γραμμένος σε κώδικα .NET ο οποίος εκτελείται σε Windows περιβάλλον και να καλεί μία διαδικτυακή υπηρεσία γραμμένη σε Java, η οποία εκτελείται σε λειτουργικό Linux.

Ο χρήστης μιας διαδικτυακής υπηρεσίας δεν θα πρέπει να συνδέεται άμεσα με την εν λόγω υπηρεσία. Η διαδικτυακή διεπαφή θα πρέπει να μπορεί να αλλάξει με την πάροδο του χρόνου, χωρίς να θέτει σε κίνδυνο την ικανότητα του χρήστη να αλληλεπιδράσει με



Σχήμα 2.2: SOA

την υπηρεσία. Σε ένα στενά συνδεδεμένο σύστημα πελάτη – εξυπηρετητή αν γίνουν αλλαγές σε μια διεπαφή, η άλλη θα πρέπει να ενημερωθεί άμεσα. Υιοθετώντας την αρχιτεκτονική μιας όχι τόσο στενής σχέσης, το λογισμικό τείνει να γίνεται πιο εύχρηστο, πιο εύκολο στην διαχείριση και επιτρέπει απλούστερη ολοκλήρωση ανάμεσα στα δύο συστήματα.

Σε μία διαδικτυακή υπηρεσία συνήθως περιλαμβάνεται και ένα περιβάλλον για την υπηρεσία αυτή ή τουλάχιστον ένα έγγραφο τεκμηρίωσης της υπηρεσίας, έτσι ώστε οι άλλοι προγραμματιστές να μπορούν να την ενσωματώσουν πιο εύκολα. Επίσης, περιλαμβάνει ιδανικά μια δημόσια διεπαφή γραμμένη σε μια κοινή γραμματική XML. Η γραμματική XML μπορεί να χρησιμοποιηθεί για τον εντοπισμό όλων των δημόσιων μεθόδων, ορίσματα μεθόδων, και επιστρεφόμενες τιμές. Βασικό χαρακτηριστικό είναι ότι κάθε διαδικτυακή υπηρεσία είναι διαθέσιμη και μπορεί ο προγραμματιστής να την βρει με κάποιον μηχανισμό αναζήτησης.

Άρα μπορούμε να πούμε ότι ολοκληρωμένη Web υπηρεσία είναι η υπηρεσία η οποία:

- Είναι διαθέσιμη στο Διαδίκτυο ή σε ιδιωτικό δίκτυο
- Χρησιμοποιεί τυποποιημένο XML σύστημα μετάδοσης πληροφορίας
- Χαλαρή σχέση πελάτη-εξυπηρετητή
- Είναι ανεξάρτητη από κάθε λειτουργικό σύστημα
- Περιγράφεται μέσω κοινής γραμματικής XML
- Είναι εντοπίσιμη από απλές μηχανές αναζήτησης

Πολλές είναι οι τεχνολογίες που έχουν υιοθετήσει τον τίτλο υπηρεσία διαδικτύου και θα ακολουθήσουν πολλές ακόμα στα επόμενα χρόνια. Στην πραγματικότητα, το παράδειγμα των διαδικτυακών υπηρεσιών έχει αυξηθεί τόσο γρήγορα που αρκετές

ανταγωνιστικές τεχνολογίες προσπαθούν να παρέχουν την ίδια δυνατότητα. Ωστόσο, το όραμα των υπηρεσιών web για την απρόσκοπτη παγκόσμια επιχειρηματική ενοποίηση δεν είναι εφικτό, εκτός αν οι βασικές τεχνολογίες σχεδιάζονται και υποστηρίζονται από κάθε μεγάλη εταιρεία λογισμικού στον κόσμο.

2.1.4 XML (eXtensible Markup Language)

XML είναι μία W3C (World Wide Web Consortium) προδιαγραφή που ορίζει μια μετα-γλώσσα για την περιγραφή των δεδομένων. Σε εφαρμογές XML, τα δεδομένα περιγράφονται από ετικέτες σε μορφή κειμένου που δίνουν πληροφορίες για τα ίδια τα δεδομένα, καθώς και την ιεραρχική τους δομή. Γι' αυτό είναι και ανεξάρτητα από την εφαρμογή και μπορούν να διαβαστούν από τον άνθρωπο. Αυτή η απλότητα και η διαλειτουργικότητα βοήθησαν την XML να γίνει ευρέως αποδεκτή και να υιοθετηθεί ως το βασικό πρότυπο για την ανταλλαγή πληροφοριών μεταξύ των διαφορετικών συστημάτων σε μια ευρεία ποικιλία εφαρμογών, συμπεριλαμβανομένων των υπηρεσιών Web. Η XML αποτελεί τη βάση για όλες τις σύγχρονες διαδικτυακές υπηρεσίες, οι οποίες χρησιμοποιούν τις τεχνολογίες που βασίζονται σε XML για να περιγράψουν την διεπαφή τους και να κωδικοποιήσουν τα μηνύματά WSDL, SOAP, UDDI τα οποία βασίζονται σε XML και κάθε μηχανή μπορεί να αναζητήσει.

2.1.5 WSDL (Web Services Description Language)

Η WSDL είναι μια μορφοποίηση που βασίζεται σε XML για την περιγραφή υπηρεσιών Web. Επίσης συντηρείται από το W3C. Οι πελάτες που επιθυμούν να έχουν πρόσβαση σε μια υπηρεσία Web μπορεί να διαβάσουν και να ερμηνεύσουν το αρχείο WSDL του να μάθουν για τη θέση της υπηρεσίας και τις διαθέσιμες λειτουργίες της. Με τον τρόπο αυτό, ο ορισμός WSDL ενεργεί ως αρχική διασύνδεση των υπηρεσιών Web, παρέχοντας στους πελάτες όλες τις πληροφορίες που χρειάζονται για να αλληλεπιδράσουν με την υπηρεσία κατά τρόπο που ορίζεται από τα πρότυπα. Μέσω της WSDL, ένας πελάτης υπηρεσιών Web μαθαίνει πώς μπορεί να προσεγγιστεί μια υπηρεσία, ποιες εργασίες εκτελεί η υπηρεσία, τα πρωτόκολλα επικοινωνίας που υποστηρίζει η υπηρεσία, καθώς και τη σωστή μορφή για την αποστολή μηνυμάτων στην υπηρεσία.

2.1.6 JSON (JavaScript Object Notation)

Το JSON είναι έγγραφο ή αρχείο που περιέχει δεδομένα σε μορφή κειμένου και χρησιμοποιείται για την εναλλαγή των δεδομένων μεταξύ συστημάτων. Η δομή του αποτελείται από συλλογές ή λίστες ονομάτων και τιμών σε διάφορες γλώσσες που μπορούν να περιγράφουν κάποιο αντικείμενο, δομή, πίνακα ή κάποιο διάνυσμα, λίστα ή ακολουθία. Η μορφή JSON καθορίστηκε από τον Douglas Crockford.

2.1.7 UDDI (Universal Description Discovery and Integration)

Το UDDI είναι ένα αναπτυσσόμενο πρότυπο για την περιγραφή, δημοσίευση και εύρεση των διαδικτυακών υπηρεσιών που προσφέρει μία εταιρία. Πρόκειται για έναν κατανεμημένο κατάλογο πληροφοριών για διαδικτυακές υπηρεσίες. Μόλις αναπτυχθεί μια διαδικτυακή υπηρεσία και δημιουργηθεί και ένα έγγραφο WSDL το οποίο την

περιγράφει, πρέπει στη συνέχεια να υπάρχει ένας τρόπος να έρθει η πληροφορία του WSDL στα χέρια των χρηστών οι οποίοι θέλουν να χρησιμοποιήσουν τις διαδικτυακές υπηρεσίες τις οποίες αυτό περιγράφει. Μόλις μια διαδικτυακή υπηρεσία δημοσιεύεται σε έναν κατάλογο UDDI, πιθανοί χρήστες έχουν τη δυνατότητα να ψάξουν και να μάθουν για την ύπαρξη της υπηρεσίας αυτής.

2.1.8 Πλεονεκτήματα και Οφέλη των Διαδικτυακών Υπηρεσιών

Σε σχέση με παλαιότερες τεχνολογίες και πρακτικές οι υπηρεσίες διαδικτύου προφέρουν:

- Ευστροφία των συστημάτων
- Εξοικονόμηση κόστους μέσω της δυνατότητας επαναχρησιμοποίησης κώδικα και αυτοματοποίησης διαδικασιών
- Απλότητα
- Ευκολία στην επικοινωνία
- Δυνατότητα άμεσης ολοκλήρωσης χάρη στην ταχύτητα και ευελιξία που προσφέρουν
- Μεγαλύτερη επεκτασιμότητα και προσαρμοστικότητα των εφαρμογών
- Χαλαρή συνδεσιμότητα μεταξύ εφαρμογών
- Δυνατότητα αλληλεπίδρασης μεταξύ υπηρεσιών σε οποιαδήποτε πλατφόρμα και γλώσσα προγραμματισμού

2.1.9 REST & SOAP

Υπάρχουν δύο κύρια σύνολα προδιαγραφών που εφαρμόζονται στην ανάπτυξη Διαδικτυακών εφαρμογών. Το SOAP (Simple Object Access Protocol) ή Πρωτόκολλο Πρόσβασης Απλού Αντικειμένου και το REST (Representational State Transfer) ή Μετάθεση Αντιπροσωπευτικής Κατάστασης.

2.2 Σύνολα προδιαγραφών ανάπτυξης Διαδικτυακών εφαρμογών

Το Διαδίκτυο είναι γεμάτο από στοιχεία, πληροφορίες, απόψεις, τιμές, μηνύματα, φωτογραφίες, και διάφορα άλλα δεδομένα τα οποία απορροφούν πόρους. Είναι γεμάτο από υπηρεσίες, μηχανές αναζήτησης, online καταστήματα, ιστολόγια, εγκυκλοπαίδειες, αριθμομηχανές, ηλεκτρονικά παιχνίδια, κειμενογράφους και άλλα. Η εγκατάσταση όλων αυτών των δεδομένων και όλων των προγραμμάτων στον υπολογιστή έχει μεγάλο κόστος σε πόρους και χρήματα. Αντίθετα μπορεί η εγκατάσταση των προγραμμάτων να βρίσκεται σε έναν φυλλομετρητή (web browser) και η πρόσβαση στα δεδομένα και υπηρεσίες να γίνονται μέσα από αυτό. Αντί όμως της τακτοποίησης των δεδομένων

σε σελίδες HTML με τις διαφημίσεις και όμορφα λογότυπα, η εξυπηρέτηση πραγματοποιείται συνήθως μέσω των έγγραφων XML. Τα δεδομένα προορίζονται ως είσοδος σε ένα πρόγραμμα λογισμικού και ανάλογα με την αίτηση που έχει γίνει στην υπηρεσία καλούνται αντίστοιχες μεθόδους του HTTP πρωτόκολλου.

Ο φυλλομετρητής έχει εύκολο έργο υποβάλλοντας μία αίτηση. Η υπηρεσία λαμβάνει την αίτηση και στέλνει την αντίστοιχη απάντηση. Ο client (πελάτης) την υπηρεσίας πρέπει να προσφέρει την απάντηση με τέτοιο τρόπο ώστε ο χρήστης μπορεί να καταλάβει το περιεχόμενο της απάντησης που θα λάβει. Ο φυλλομετρητής είναι επωμισμένος με το να συγκεντώσει τα δεδομένα, και ο χρήστης να αποφασίζει τι σημαίνουν τα δεδομένα. Για να γίνει αυτό ο client της υπηρεσίας είναι προγραμματισμένος εκ των προτέρων να εξάγει αυτόματα συμπέρασμα από τις απαντήσεις των αιτήσεων και να λαμβάνει αποφάσεις που βασίζονται στην εν λόγω συμπέρασμα.

2.3 HTTP (Hypertext Transfer Protocol ή HTTP)

Το HTTP, Πρωτόκολλο Μεταφοράς Υπερκειμένου, (Hypertext Transfer Protocol) λειτουργεί ως ένα πρωτόκολλο αίτησης - απάντησης στο υπολογιστικό μοντέλο client – server (πελάτη – εξυπηρετητή). Ένα πρόγραμμα περιήγησης στο Διαδίκτυο, για παράδειγμα, είναι ο πελάτης και μια εφαρμογή που τρέχει σε έναν υπολογιστή που φιλοξενεί μια ιστοσελίδα μπορεί να είναι ο server. Ο πελάτης υποβάλλει ένα μήνυμα αίτησης HTTP στον διακομιστή. Ο server, ο οποίος παρέχει πόρους, όπως αρχεία HTML και άλλο περιεχόμενο, είτε εκτελεί άλλα καθήκοντα για λογαριασμό του πελάτη, επιστρέφει ένα μήνυμα απάντησης στον πελάτη. Η απάντηση περιέχει πληροφορίες σχετικά με την κατάσταση ολοκλήρωσης της αίτηση και μπορεί επίσης να περιέχει το περιεχόμενο που ζητήθηκε στο σώμα του μηνύματος του. Το HTTP καθορίζει μεθόδους για να υποδείξει την επιθυμητή ενέργεια που θα εκτελεστεί στο αναφερόμενο πόρο. Τι αντιπροσωπεύει αυτός ο πόρος, είτε προϋπήρχαν δεδομένα είτε δεδομένα που δημιουργούνται δυναμικά, εξαρτάται από την εφαρμογή του server.

2.3.1 URIs (Uniform Resource Identifiers)

Ενιαία Αναγνωριστικά Πόρων, (Uniform Resources Identifiers ή URIs) είναι σύντομες χορδές χαρακτήρων που χρησιμοποιούνται για τον εντοπισμό ονομάτων ή πόρων στο Διαδίκτυο: έγγραφα, εικόνες, αρχεία, υπηρεσίες, ηλεκτρονικά γραμματοκιβώτια, και άλλους πόρους. Κάνουν τους πόρους διαθέσιμους κάτω από μια ποικιλία των συστημάτων και μεθόδων πρόσβασης, όπως HTTP, FTP, και το ηλεκτρονικό ταχυδρομείο και προσπελάσιμους με τον ίδιο απλό τρόπο ονοματοδοσίας.

2.3.2 Μέθοδοι HTTP πρωτοκόλλου

GET Η μέθοδος GET χρησιμοποιείται για την ανάκτηση πληροφοριών από το συγκεκριμένο server χρησιμοποιώντας ένα συγκεκριμένο URI. Οι αιτήσεις τύπου GET θα πρέπει μόνο να ανακτούν τα δεδομένα και να μην έχουν καμία επίδραση πάνω τους. Μία αίτηση GET μπορεί να γίνει υπό όρους ώστε να αποφευχθεί η άσκοπη μεταφορά δεδομένα προς το πελάτη.

POST Η αίτηση POST είναι η μόνη μη ασφαλής λειτουργία του HTTP. Είναι μια αίτηση αποστολής δεδομένων στον server. Κάθε μέθοδος POST επιτρέπεται να τροποποιήσει την υπηρεσία με ένα μοναδικό τρόπο. Υπάρχει όμως η πιθανότητα να μην σταλεί η πληροφορία με το αίτημα, όπως επίσης να μην μπορούν να ληφθούν πληροφορίες από την απάντηση.

PUT Η μέθοδος PUT ζητά από τον server να αποθηκεύσει το κυρίως μέρος του μηνύματος που εστάλει με την αίτηση, στην θέση που περιγράφεται από το μήνυμα HTTP αντικαθιστώντας την προηγούμενη κατάσταση. Όταν γίνεται μια αίτηση PUT, ο πελάτης γνωρίζει την ταυτότητα του πόρου προς δημιουργία ή προς ενημέρωση. Η αποστολή του ίδιου μηνύματος περισσότερες από μία φορά δεν θα έχει καμία επίδραση επί της υποκείμενης υπηρεσίας.

HEAD Μία αίτηση HEAD είναι παρόμοια με μια αίτηση GET με την διαφορά ότι αντί να επιστρέφεται το κυρίως μέρος της αίτησης, επιστρέφει μόνο ένας κωδικός και η κεφαλίδα που σχετίζεται με την αίτηση.

DELETE DELETE είναι αίτηση για αφαίρεση των πόρων στο συγκεκριμένο URI.

OPTIONS Η αίτηση OPTIONS ζήτα πληροφορίες σχετικά με τις επιλογές επικοινωνίας για τον συγκεκριμένο πόρο που περιγράφεται. Υπάρχουν και άλλες μέθοδοι HTTP αλλά δεν είναι τόσο σημαντικές κατά τον σχεδιασμό και την εφαρμογή των υπηρεσιών που θα εξετάσουμε στην συνέχεια.

2.3.3 SOAP (Simple Object Access Protocol)

SOAP είναι ένα πρωτόκολλο που βασίζεται σε XML αρχεία για την ανταλλαγή δεδομένων μέσω HTTP. Παρέχει απλές μεθόδους, που βασίζονται σε πρότυπα για την αποστολή μηνυμάτων σε μορφή XML μεταξύ των εφαρμογών σε μια ποικιλία τυποποιημένων τεχνολογιών Διαδικτύου, συμπεριλαμβανομένων SMTP, HTTP, και FTP. Οι υπηρεσίες Web χρησιμοποιούν το πρωτόκολλο SOAP για την αποστολή μηνυμάτων μεταξύ της υπηρεσίας και του πελάτη. Επειδή το HTTP υποστηρίζεται από όλους τους διακομιστές και Web browsers, τα μηνύματα SOAP μπορούν να σταλούν μεταξύ των εφαρμογών, ανεξάρτητα από πλατφόρμα ή γλώσσα προγραμματισμού. Τα δεδομένα αποστέλλονται μεταξύ του πελάτη και της υπηρεσίας Web χρησιμοποιώντας μηνύματα αιτήματος - απόκρισης SOAP, η μορφή των οποίων καθορίζεται από το WSDL. Επειδή ο πελάτης και ο διακομιστής τηρούν τη σύμβαση WSDL, κατά τη δημιουργία SOAP μηνυμάτων η συμβατότητα είναι εγγυημένη.

2.3.4 REST (Representational State Transfer)

Ο όρος Representational State Transfer (REST) επινοήθηκε από τον Roy Fielding για να προσδιορίσει μια αρχιτεκτονική που βασίζεται σε ένα σύνολο αρχών για το σχεδιασμό λογισμικού του διαδικτύου. Η REST ορίζει ένα σύνολο αρχιτεκτονικών αρχών για την σχεδίαση υπηρεσιών Web στο οποίο τα δεδομένα και την πληροφορία θεωρούνται πόροι του συστήματος και η πρόσβαση σε αυτούς δίνεται μέσω των

Ενιαίων Αναγνωριστικών Πόρων (URIs), που αποτελούν συνδέσμους στο διαδίκτυο. Στην ουσία ορίζει το πώς οι πόροι αντιμετωπίζονται και μεταφέρονται μέσω HTTP από ένα ευρύ φάσμα πελατών (clients) σε διαφορετικές γλώσσες.

Η βασική ιδέα σε κάθε REST API είναι ο πόρος. Ένας πόρος είναι ένα αντικείμενο που έχει έναν τύπο, συσχετιζόμενα δεδομένα, σχέσεις με άλλους πόρους, καθώς και μια σειρά από μεθόδους που λειτουργούν πάνω σε αυτό. Είναι παρόμοιο με ένα αντικειμένου σε μία αντικειμενοστραφή γλώσσα προγραμματισμού, με τη σημαντική διαφορά ότι μόνο μερικές πρότυπες μέθοδοι ορίζονται για τον πόρο (πρότυπο HTTP : GET, POST, PUT και DELETE μέθοδοι), ενώ ένα αντικείμενο συνήθως έχει πολλές μεθόδους. Οι πόροι μπορούν να ομαδοποιηθούν σε συλλογές - σορούς. Κάθε συλλογή είναι ομοιογενής, ώστε να περιέχει μόνο ένα είδος πόρων, χωρίς συγκεκριμένη τάξη. Οι πόροι μπορούν επίσης να υπάρχουν έξω από κάποια συλλογή. Κάθε αντικείμενο και πόρος στο σύστημα είναι προσβάσιμο μέσω ενός μοναδικού αναγνωριστικού. Στο REST αυτό επιτυγχάνεται με την χρήση των URIs. Όταν πραγματοποιείται μια αίτηση HTTP στον φυλλομετρητή πρέπει να περιέχει το URI του αντικειμένου που ζητείται από τον server ή τοποθετείται σε αυτόν.

Η αρχιτεκτονική REST παρουσιάζει τα ακόλουθα πλεονεκτήματα:

- Δεν υπάρχει ανάγκη για διατήρηση της κατάσταση των εφαρμογών, οδηγώντας σε απλοποιημένα και επεκτάσιμα συστατικά.
- Οι αιτήσεις μπορούν να υποβάλλονται σε επεξεργασία παράλληλα.
- Οι αιτήσεις μπορούν να γίνουν κατανοητές αν απομονωθούν, οδηγώντας στην απλουστευμένη οργάνωση και δυναμική αναδιοργάνωση των υπηρεσιών.
- Εξυπηρετεί στην καλή χρήση του HTTP cache και του διακομιστή μεσολάβησης (proxy server) που βοηθάει στον χειρισμό μεγαλύτερου φορτίου.

2.3.5 REST VS SOAP

Συγκρίνοντας το REST με το SOAP στην εφαρμογή τους σε διαδικτυακές υπηρεσίες καταλήγουμε στο εξής συμπέρασμα. Το REST είναι στο μεγαλύτερο μέρος του πιο εύκολο στην χρήση του και πιο ευέλικτο. Παρουσιάζει τα παρακάτω πλεονεκτήματα συγκριτικά με το SOAP:

- Μικρότερη καμπύλη εκμάθησης.
- Καταλληλότερο για περιβάλλον σημείο-προς-σημείο ή όταν ο ενδιάμεσος δεν παίζει σημαντικό ρόλο.
- Δεν χρειάζονται ακριβά εργαλεία για να αλληλεπιδρούν με την διαδικτυακή υπηρεσία.
- Είναι αποτελεσματικότερο διότι το SOAP χρησιμοποιεί XML αρχεία για όλα τα μηνύματα ενώ το REST μπορεί να χρησιμοποιήσει μικρότερα μηνύματα.
- Είναι πολυ γρήγορο διότι δεν απαιτείται εκτεταμένη επεξεργασία.
- Ελάχιστα εργαλεία και ενδιάμεσα πρόγραμμα απαιτούνται. Απαιτείται μόνο η υποστήριξη HTTP.

- Ενσωματωμένη διαχείριση σφαλμάτων.

Το REST παρουσιάζει κάποια μειονεκτήματα του συγκριτικά με το SOAP. Το REST για να πέτυχει να είναι απλό και γρήγορο στερείται αξιοπιστίας, καθώς είναι πιθανό για παράδειγμα κάποιες HTTP μέθοδοι όπως η DELETE να επιστρέψουν OK χωρίς οι πόροι να έχουν διαγραφεί. Επίσης το SOAP εμπνέει μεγαλύτερη ασφάλεια λόγων των πολλών προτύπων για την αξιοπιστία, την ασφάλεια και τις συναλλαγές.

Χρησιμοποιούμε τον όρο RESTful για να αναφερθούμε σε υπηρεσίες Web χτισμένες σύμφωνα με την αρχιτεκτονική REST. Η REST έχει γίνει τα τελευταία χρόνια ένα κυρίαρχο μοντέλο σχεδιασμού υπηρεσιών Web. Η αρχιτεκτονική REST είχε τόσο μεγάλο αντίκτυπο στο διαδίκτυο και έχει κυριαρχήσει έναντι υπηρεσιών γραμμένες με βάση τα SOAP και WSDL πρότυπα, διότι είναι πολύ απλούστερη στη χρήση, παρέχει επεκτασιμότητα και έχει καλύτερη απόδοση.

Κεφάλαιο 3

Τεχνολογιες

3.1 Python

Η Python είναι μια γενικού σκοπού, υψηλού επιπέδου και ανοιχτού πηγαίου κώδικα γλώσσα προγραμματισμού. Δημιουργήθηκε από τον *Guido van Rossum* και κυκλοφόρησε δημοσίως τον Φεβρουάριο του 1991.

Ο σχεδιασμός της Python δίνει ιδιαίτερη έμφαση στη δημιουργία ευανάγνωστου πηγαίου κώδικα και η σύνταξη της επιτρέπει την επίλυση προβλημάτων με λιγότερο πηγαίο κώδικα σε σχέση με άλλες γλώσσες προγραμματισμού όπως η C και η Java. Κατατάσσεται ως μια από τις 5 έως 10 πιο διαδεδομένες γλώσσες προγραμματισμού στον κόσμο και έχει ποικίλους ρόλους όπως διαδικτυακό προγραμματισμό, διαχείριση συστημάτων, επιστημονικό προγραμματισμό, εξόρυξη δεδομένων και άλλα.

3.1.1 Κύρια Χαρακτηριστικά

Η Python συχνά χαρακτηρίζεται ως αντικειμενοστραφής γλώσσα προγραμματισμού, αλλά ένα από τα πιο σημαντικά χαρακτηριστικά της είναι ότι συνδυάζει πολλαπλά πρότυπα προγραμματισμού, συμπεριλαμβανομένων του συναρτησιακού και προστακτικού προγραμματισμού. Άλλα κύρια χαρακτηριστικά της είναι οι υψηλού επιπέδου ενσωματωμένες δομές δεδομένων, δυναμικοί τύποι κωδικοποίησης, αυτόματη διαχείριση μνήμης καθώς επίσης και μια μεγάλη και περιεκτική πρότυπη βιβλιοθήκη.

Εξίσου σημαντικό χαρακτηριστικό που ακολουθεί τη φιλοσοφία της γλώσσας, είναι η υποστήριξη ομαδοποίησης του πηγαίου κώδικα σε μονάδες και πακέτα, ενθαρρύνοντας την επαναχρησιμοποίηση του, και διευκολύνοντας την προσθήκη νέων μονάδων υλοποιημένων σε γλώσσες προγραμματισμού μεταγλωττιστή όπως η C ή η C++.

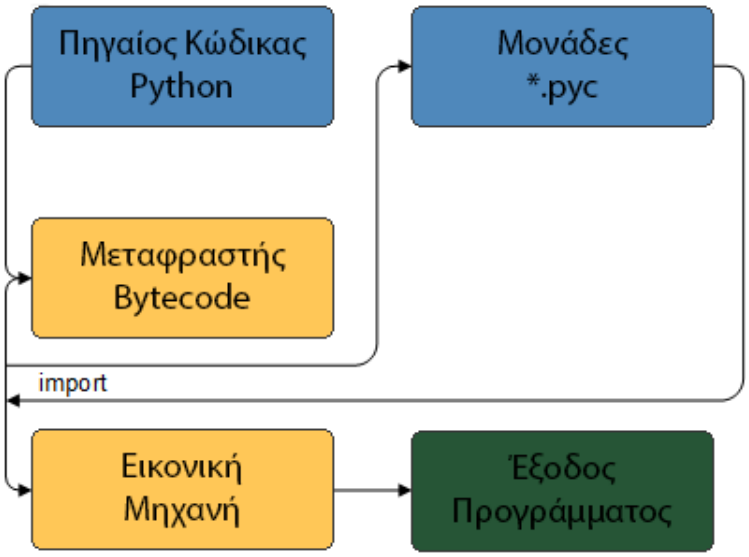
Μερικά άλλα χαρακτηριστικά της είναι:

- Μπορεί να ενσωματωθεί σε μια εφαρμογή ώστε να παρέχει μια προγραμματιζόμενη διασύνδεση.
- Είναι συμβατή με όλες τις κύριες πλατφόρμες υλικού και λογισμικού.
- Ποικιλία βασικών τύπων δεδομένων όπως συμβολοσειρές, λίστες και λεξικά.
- Υποστήριξη εξαιρέσεων.

3.1.2 Υλοποιήσεις

Η γλώσσα προγραμματισμού Python αποτελείται από το πρότυπο και την υλοποίηση αναφοράς CPython, αλλά υπάρχουν και διαφορετικές υλοποιήσεις που διαφέρουν κατά κάποιο τρόπο από την υλοποίηση αναφοράς. Ως υλοποίηση της Python, νοείται ένα πρόγραμμα ή περιβάλλον που παρέχει υποστήριξη για την εκτέλεση προγραμμάτων γραμμένα με τη γλώσσα προγραμματισμού Python, όπως παριστάνεται από την υλοποίηση αναφοράς CPython.

Η υλοποίηση αναφοράς CPython, είναι η κύρια και αρχική υλοποίηση της Python, γραμμένη με τη γλώσσα προγραμματισμού C πληρώντας το πρότυπο προδιαγραφών C89. Η CPython και οι περισσότερες υλοποιήσεις της Python, μεταφράζουν προγράμματα γραμμένα σε κώδικα Python σε μορφή κώδικα byte, ο οποίος εκτελείται από την εικονική μηχανή παρέχοντας φορητότητα, καθώς είναι σε μια μορφή ανεξάρτητη από τη πλατφόρμα.



Σχήμα 3.1: Λειτουργία Python

Κάποιες εναλλακτικές υλοποιήσεις της Python βασίζονται στον πυρήνα εκτέλεσης της υλοποίησης αναφοράς CPython, αλλά με εκτεταμένη συμπεριφορά ή χαρακτηριστικά σε ορισμένες πτυχές όπως η Stackless Python, που δίνει έμφαση στην παραλληλία.

Άλλες υλοποιήσεις δεν εξαρτώνται ή αλληλοεπιδρούν με τον πυρήνα εκτέλεσης αλλά επαναχρησιμοποιούν ένα μεγάλο μέρος της πρότυπης βιβλιοθήκης και είναι συμβατές με την προτυποποίηση της γλώσσας. Χαρακτηριστικά παραδείγματα η IronPython και η Jython, υλοποιήσεις για τις πλατφόρμες Common Language Runtime (CLR/.NET) και Java αντίστοιχα.

3.2 Πλαίσιο Ανάπτυξης Django

Το Django είναι ένα καλά εδραιωμένο πλαίσιο ανάπτυξης ανοικτού πηγαίου κώδικα σχεδιασμένο με πρωταρχικό στόχο να διευκολύνει τη δημιουργία πολύπλοκων ιστοσελίδων που βασίζονται σε βάσεις δεδομένων και διαδικτυακών εφαρμογών. Η

ανάπτυξη του ξεκίνησε το φθινόπωρο του 2003 από τους *Adrian Holovaty* και *Simon Willison*. Κυκλοφόρησε δημοσίως υπό την άδεια λογισμικού BSD τον Ιούλιο του 2005, και συντηρείται από τον μη κερδοσκοπικό οργανισμό Django Software Foundation, που ιδρύθηκε το 2008 με σκοπό τη συντήρηση, προώθηση και συνεχής ανάπτυξη του.

3.2.1 Κύρια Χαρακτηριστικά

Το Django έχει σχεδιαστεί ακολουθώντας την αρχιτεκτονική λογισμικού MVC, προέχοντας δυνατότητες γρήγορης και εύκολης ολοκλήρωσης συνηθισμένων εργασιών της ανάπτυξης διαδικτυακών εφαρμογών. Ένα επιπλέον κύριο χαρακτηριστικό της σχεδίασης του, είναι ο διαχωρισμός της εφαρμογής σε διαφορετικά εξειδικευμένα και ανεξάρτητα τμήματα, που μπορούν να χρησιμοποιηθούν σε διαφορετικές εφαρμογές. Για τον σαφή διαχωρισμό των ορολογιών των διαφορετικών τμημάτων, υιοθετείται η ακόλουθη σύμβαση:

- Project ή Κύρια Εφαρμογή, αναφέρεται σε ολοκληρη την πρόγραμμα
- App ή Εφαρμογή, αναφέρεται στις υπομονάδες που απαρτήζουν το πρόγραμμα

Ο πυρήνας του απαρτίζεται από έναν αριθμό διαφορετικών αυτόνομων μονάδων. Οι κύριες μονάδες είναι ένα υποσύστημα αντικείμενο-σχεσιακής αντιστοίχισης που μεσολαβεί μεταξύ των μοντέλων δεδομένων και μιας σχεσιακής βάση δεδομένων, ένα σύστημα για την επεξεργασία των εισερχομένων αιτημάτων, ένα URL Dispatcher κανονικών εκφράσεων καθώς και ένα ελαφρύ, χαμηλού επιπέδου σύστημα ενδιάμεσου λογισμικού με δυνατότητα τροποποίησης της εισόδου ή εξόδου του Django ή των επιμέρους μονάδων.

Επίσης στον πυρήνα του πλαισίου ανάπτυξης περιλαμβάνονται:

- Σύστημα σειριακής μετατροπής και επικύρωσης, με δυνατότητα μετατροπής εγγράφων τύπου HTML και JSON/XML σε στιγμιότυπα μοντέλων δεδομένων
- Γλώσσα σήμανσης βασισμένη στην HTML για τον καθορισμό της διεπαφής χρήστη
- Πλαίσιο προσωρινής μνήμης με υποστήριξη διαφόρων μεθόδων
- Εσωτερικό σύστημα επικοινωνίας γεγονότων και σημάτων
- Υποστήριξη διεθνοποίησης

Για τους σκοπούς της ανάπτυξης, το Django είναι εντελώς αυτόνομο. Περιλαμβάνει ένα περιβάλλον γραμμής εντολών, έναν αυτόνομο διακομιστή ιστού χαμηλών απαιτήσεων που μπορεί να λειτουργήσει χωρίς την εγκατάσταση πρόσθετου λογισμικού, καθώς και διεπαφή με το πλαίσιου μονάδων ελέγχου της Python.

3.2.2 Προεγκατεστημένες εφαρμογές

Το Django επίσης συμπεριλαμβάνει έναν αριθμό προαιρετικών προεγκαταστημένων εφαρμογών καθώς και ενδιάμεσου λογισμικού που μπορούν να αλλάξουν την είσοδο ή έξοδο του Django και των επιμέρους στοιχείων του.

Διαθέτει μια εφαρμογή διεπαφής διαχειριστή με δυνατότητες δημιουργίας, ανάγνωσης, ενημέρωσης και διαγραφής μοντέλων, που δημιουργείται δυναμικά και διαμορφώνεται μέσω των μοντέλων του διαχειριστή και άλλες, όπως οι εξής:

- Επεκτάσιμο σύστημα ελέγχου ταυτότητας/πιστοποίησης.
- Εργαλεία δημιουργίας τροφοδοσιών RSS και Atom.
- Ευέλικτο σύστημα σχολιασμού.
- Μηχανισμούς προστασίας από μεθόδους επίθεσης όπως cross-site forgery, scripting και SQL Injection.

Υποστήριξη διακομιστών και βάσεων δεδομένων

Το Django μπορεί να εκτελεστεί σε συνδυασμό με έναν αριθμό διαφορετικών διακομιστών ιστού, όπως ο *Apache* ή ο *Nginx* και οποιονδήποτε διακομιστή με υποστήριξη του πρωτοκόλλου *FastCGI* και του πρότυπου *WSGI* οπως ο *Lighttpd*.

Επιπλέον, ένα από τα προεγκαταστημένα συστήματα του Django είναι η μονάδα αντικείμενο-σχεσιακής αντιστοίχισης που προσφέρει υποστήριξη για σχεσιακές βάσεις δεδομένων όπως οι *PostgreSQL*, *MySQL*, *SQLite* και με τη χρήση πρόσθετου λογισμικού μπορεί να υποστηριχθεί η χρήση μη-σχεσιακών βάσεων δεδομένων όπως η *MongoDb*.

3.2.3 Ασφάλεια

Η ασφάλεια των δεδομένων που ανταλλάσσονται αποτελεί μια από τις πιο σημαντικές απαιτήσεις των διαδικτυακών εφαρμογών. Όταν ένας χρήσης εκμεταλλεύεται της δυνατότητες μίας υπηρεσίας στο διαδίκτυο, πολλές φορές χωρίς να το γνωρίζει ή να μπορεί να το αντιληφθεί, η πληροφορία που ανταλλάσσει όπως και τα προσωπικά του δεδομένα μπορεί να κλαπούν ή αν αλλοιωθούν. Οι συνήθεις κίνδυνοι που συναντώνται στο διαδίκτυο:

Cross-site scripting Ένα είδος επίθεσης από κακόβουλους χρήστες οι οποίοι εγχέουν κομμάτια κώδικα (scripts) σε ιστοσελίδες και αποσπούν δεδομένα χρηστών. Όταν ένας χρήστης συνδεθεί σε web υπηρεσία ο φυλλομετρητής δεν έχει τρόπο να γνωρίζει ότι ο κώδικας δεν είναι αξιόπιστος οπότε και τον εκτελεί. Έτσι, ο κακόβουλος χρήστης με το Cross-site scripting μπορεί με να υποκλέψει πληροφορίες χρηστών αφού πλέον θα έχει πρόσβαση σε όλα τα cookies, session tokens και άλλες ευαίσθητες πληροφορίες που διατηρούνται από το πρόγραμμα περιήγησης.

Cross-Site Request Forgery Μια κοινή τακτική ηλεκτρονικής επίθεσης. Σε αυτή την περίπτωση η επίθεση βασίζεται στην εμπιστοσύνη που έχει η υπηρεσία στον τελικό χρήστη, ο οποίος έχει περάσει τους ελέγχους σύνδεσης, αντίθετα με αυτό που γινόταν στο Cross-site scripting. Ένας εξουσιοδοτημένος χρήστης συνδέεται με την υπηρεσία μέσω ενός URL, τοποθετημένο σε διαφορά σημεία του διαδικτύου από τους επιτιθέμενους. Το URL αυτό όμως περιέχει κακόβουλα αιτήματα τα οποία χωρίς να το γνωρίζει ο χρήστης μπορεί να ζητάνε από την υπηρεσία την αλλαγή στοιχείων του χρήστη, ηλεκτρονικής διεύθυνσης, κωδικών πρόσβασης μέχρι και μεταφορά χρημάτων.

SQL injections Ερωτήματα SQL τα οποία εγχέονται κατά την αποστολή ή εισαγωγή δεδομένων από τον πελάτη στην υπηρεσία. Μία τέτοια κακόβουλη “ένεση” SQL μπορεί να διαβάσει ευαίσθητα δεδομένα από την βάση δεδομένων της υπηρεσίας ή να τα τροποποιήσει. Με αυτό τον τρόπο ο επιτιθέμενος μπορεί να ξεγελάσει την εφαρμογή στην ταυτοποίηση του, να καταστρέψει δεδομένα, να υποκλέψει πληροφορίες ή να διαγράψει ολόκληρη την βάση δεδομένων.

Μία διαδικτυακή υπηρεσία πρέπει να προστατεύει τον εαυτό της και τους χρήστες της από αυτές τις επιθέσεις και άλλους κινδύνους που προκύπτουν στην διαδικτυακή επικοινωνία και ανταλλαγή δεδομένων. Το πλαίσιο ανάπτυξης Django έχει σχεδιαστεί έτσι ώστε να προσφέρει μια ασπίδα προστασίας και να παρέχει ασφάλεια στον τελικό χρήστη της υπηρεσίας, χρησιμοποιώντας διάφορες τεχνικές. Μέσα στην βιβλιοθήκη του Django υπάρχουν τα κατάλληλα πακέτα εργαλείων που εξασφαλίζουν σε μεγάλο βαθμό την πιστοποίηση των χρηστών, την αυθεντικότητα των δεδομένων, την ακεραιότητα της πληροφορίας καθώς και προστασία από διαδικτυακές επιθέσεις. Ενδεικτικές μονάδες είναι το Django Admin το οποίο παρέχει έτοιμη πλατφόρμα για τον διαχειριστή της υπηρεσίες, το μοντέλο Χρήστη ή User Model το οποίο είναι καθοριστικό για την κατασκευή της οντότητας του χρήστη και της ασφάλειας του και τεχνικές SSL/HTTPS.

Με τις δυνατότητες που προσφέρει το πακέτο προστασίας CSRF ο πελάτης εκχέει κρυπτογραφημένη πληροφορία στην αίτηση τύπου POST από τον χρήστη προς την υπηρεσία. Αύτη η πληροφορία ανταλλάσσεται κατά την επικοινωνία της υπηρεσίας με τον χρήστη και έτσι διασφαλίζεται η ακεραιότητα των αιτημάτων. Ο κακόβουλος χρήστης δεν μπορεί να έχει πρόσβαση στην αποκρυπτογραφημένη πληροφορία και κατά συνέπεια δεν μπορεί να ξεγελάει την υπηρεσία σχετικά με την ταυτότητά του.

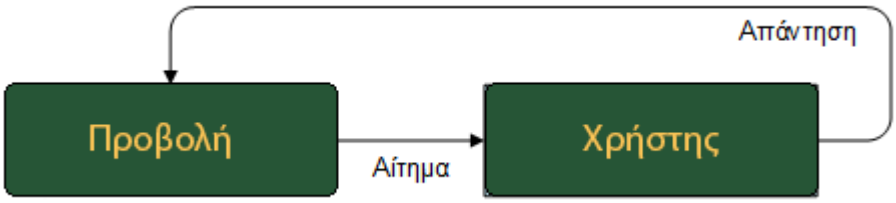
Με τη χρήση των Templates στο επίπεδο προβολής, ή προγραμματιζόμενων διεπαφών το Django προστατεύει τον χρήστη από επιθέσεις τύπου Cross-site scripting. Ο προγραμματιστής χρησιμοποιώντας κατάλληλα tags και φίλτρα μέσα στην γλώσσα σήμανσης HTML οδηγεί τον φυλλομετρητή στο να αποφύγει όλα τα κακόβουλα κομμάτια κώδικα και να δημιουργήσει μία ζώνη άμυνας απέναντι στο Cross-Site scripting.

Επιπλέον, το Django θωρακίζει την υπηρεσία από κακόβουλα SQL αιτήματα καθώς δεν επιτρέπει την απευθείας επικοινωνία του πελάτη με την βάση μέσω ”ωμών” ερωτημάτων SQL. Με χρήση μεθόδων διαφυγής αιτημάτων ελέγχονται όλα τα ερωτήματα πριν περάσουν στην βάση και εξυπηρετηθούν. Η ασφάλεια στο Django και κυρίως η δυνατότητα που παρέχει στους προγραμματιστές να ενσωματώνουν με ευκολία και ταχύτητα επιπλέον μηχανισμούς προστασίας είναι ένα από τα πλεονεκτήματα που το διακρίνουν.

3.2.4 Αρχιτεκτονική

Στο πιο βασικό επίπεδο η κύρια λειτουργία που εκτελεί το Django είναι η επεξεργασία των εισερχομένων αιτημάτων και η αποστολή των κατάλληλων απαντήσεων.

Η χρήση αυτόνομων μονάδων σε συνδυασμό με την χρήση της αρχιτεκτονικής *MVC*, δίνει τη δυνατότητα στο Django να είναι εύκολα παραμετροποιήσιμο καθώς με τη χρήση διαφορετικών εφαρμογών και ενδιάμεσου λογισμικού στα διαφορετικά επίπεδα της εφαρμογής είναι πολύ εύκολη η προσαρμογή ή προσθήκη χαρακτηριστικών έτσι ώστε να καλύψει τις επιθυμητές λειτουργίες.

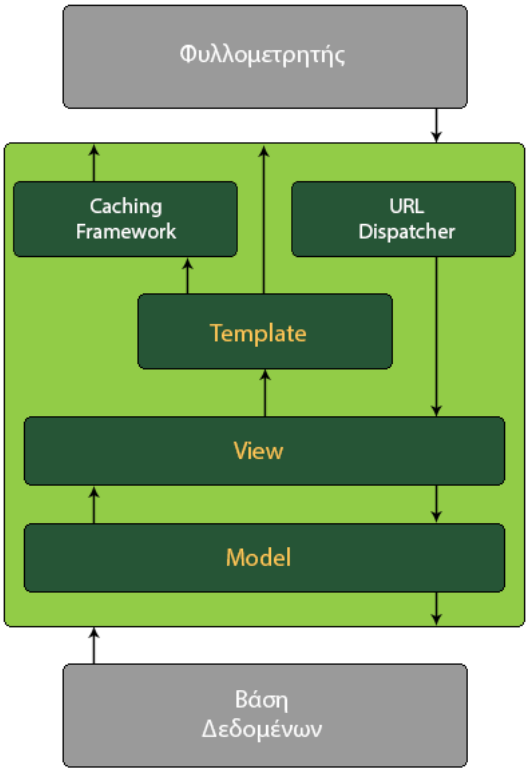


Σχήμα 3.2: Επεξεργασία Αιτημάτων

Κύκλος Αιτημάτων-Απαντήσεων

Οι διαδικασίες που πραγματοποιούνται εσωτερικά για την επεξεργασία των αιτημάτων και την αποστολή των απαντήσεων περιγράφονται από τον κύκλο αιτημάτων-απαντήσεων.

Το Django δέχεται ως είσοδο ένα αίτημα. Αρχικά η μονάδα *URL Configuration* ελέγχει αν ο υπερσύνδεσμος του αιτήματος αντιστοιχεί σε κάποια προβολή και το καλεί με παράμετρο το αίτημα. Η προβολή με τη σειρά της, ανάλογα την προγραμματισμένη λειτουργία, επεξεργάζεται το αίτημα και επιστρέφει την απάντηση. Συνήθως για τη δημιουργία της απάντησης η προβολή θα ανακτήσει ή τροποποιήσει δεδομένα από τα απαραίτητα μοντέλα και τα αποθηκεύει προσωρινά σε ένα λεξιλόγιο. Τέλος θα επιστρέψει ένα έγγραφο HTML δημιουργημένο από ένα Template και το λεξιλόγιο.



Σχήμα 3.3: Αρχιτεκτονική Εφαρμογής Django

MVC

Η πρότυπη αρχιτεκτονική λογισμικού Model-View-Controller ή Μοντέλο-Προβολή-Ελεγκτής διαχωρίζει την εφαρμογή σε τρία αλληλένδετα μέρη, το επίπεδο μοντέλου, προβολής και ελεγκτή. Το επίπεδο μοντέλου ενσωματώνει τα δεδομένα της εφαρμογής και είναι ανεξάρτητο από συγκεκριμένες αναπαραστάσεις εξόδου ή συμπεριφορές εισόδου. Το επίπεδο ελεγκτή είναι υπεύθυνο την τροποποίηση της κατάστασης του μοντέλου, και το επίπεδο προβολής ουσιαστικά παράγει την αναπαράσταση εξόδου στον χρήστη.

Το Django διαφοροποιείται από το κλασσικό πρότυπο της αρχιτεκτονικής MVC. Οι λειτουργίες του επιπέδου ελεγκτή εκτελούνται εν μέρη από το ίδιο το Django και από τις προβολές, και το επίπεδο προβολής αντιπροσωπεύεται από τα Templates, τα οποία λαμβάνουν τα δεδομένα για την αναπαράσταση των δεδομένων από τις προβολές.

Επίπεδο Μοντέλου

Το επίπεδο μοντέλου χειρίζεται τα δεδομένα της εφαρμογής. Χειρίζεται τα δεδομένα μόνο στον βαθμό που είναι απαραίτητο για να μπορούν να αποθηκευτούν και να ανακτηθούν. Δεν έχει κανέναν έλεγχο σχετικά με τις ενέργειες που πρέπει να κάνει η εφαρμογή με τα δεδομένα. Στο κλασσικό πρότυπο MVC, το μοντέλο απαντάει σε αιτήσεις από την προβολή και ακολουθεί οδηγίες από τον ελεγκτή και ενημερώνει τον εαυτό του σαν μονάδα. Στο Django ο ρόλος του μοντέλου παραμένει ίδιος, με τη διαφορά ότι επικοινωνεί μόνο με τις προβολές.

Ένα μοντέλο είναι η ενιαία πηγή πληροφοριών σχετικά με τα δεδομένα. Περιέχει τα βασικά πεδία και συμπεριφορές των δεδομένων. Συνήθως κάθε μοντέλο αντιστοιχεί σε έναν πίνακα βάσης δεδομένων.

Στο Django ένα μοντέλο αναπαρίσταται με μια κλάση

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

Το παραπάνω μοντέλο θα δημιουργήσει τον κατάλληλο πίνακα στη βάση δεδομένων

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

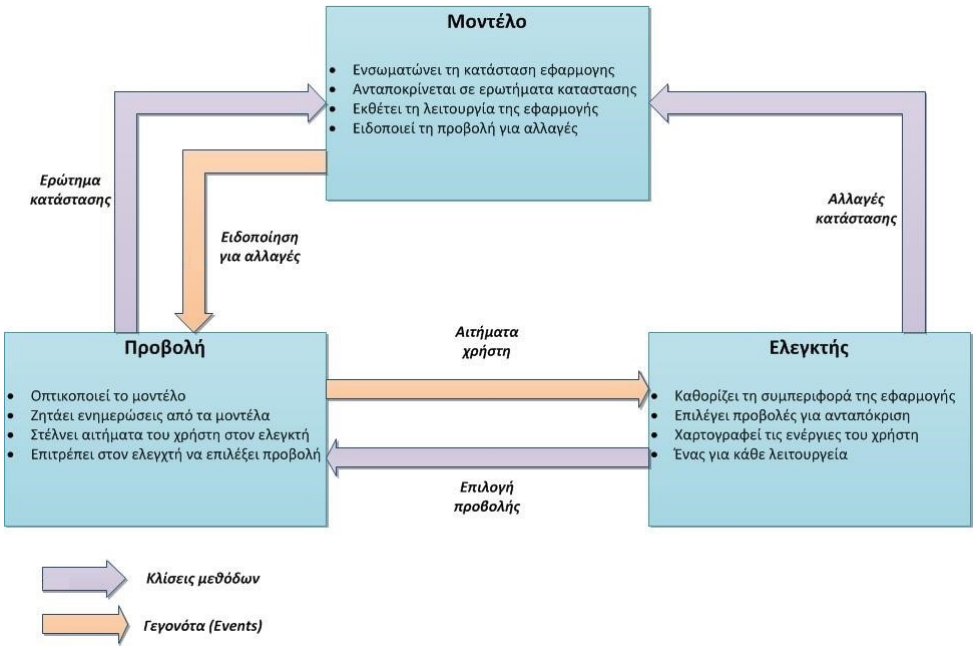
Επίπεδο Προβολής

Στην αρχιτεκτονική MVC το επίπεδο προβολής είναι υπεύθυνο για την εμφάνιση ή έξοδο των δεδομένων. Δεν έχει καμία λειτουργικότητα από μόνο του, εκτός του να μετατρέπει και να θέτει τα δεδομένα έτσι ώστε να έχουν την κατάλληλη μορφή.

Το Django χρησιμοποιεί τη μονάδα Templates ως επίπεδο προβολής, που χρησιμοποιεί μια γλώσσα σήμανσης βασισμένη στην HTML. Η διαφορά των Templates, από το

κλασσικό επίπεδο προβολής, είναι ότι δεν ζητούν δεδομένα απευθείας από τα μοντέλα, αλλά τις προβολές τούς παρέχουν μία λίστα με τα στοιχεία που χρειάζονται.

Επίσης σε κάποιες περιπτώσεις δεν γίνεται χρήση Templates και οι προβολές καλύπτουν τον ρόλο του επιπέδου προβολής επιστρέφοντας τα δεδομένα σε κάποια συγκεκριμένη μορφή, όπως έγγραφα JSON και XML.



Σχήμα 3.4: Αρχιτεκτονική Μοντέλο-Προβολή-Ελεγκτής

Επίπεδο Ελεγκτή

Το επίπεδο ελεγκτή είναι η μονάδα πυρήνα η οποία παίρνει όλες τις αποφάσεις. Επεξεργάζεται και ανταποκρίνεται σε γεγονότα ή ενέργειες του χρήστη, και ελέγχει όλες τις σχετικές λειτουργίες του προγράμματος. Αλληλοεπιδρά με το μοντέλο και την προβολή, δίνοντάς τους εντολές, χειρίζεται το πέρασμα των δεδομένων μεταξύ τους και μπορεί να προβεί σε αλλαγές.

Στο Django τον ρόλο του ελεγκτή τον πληροί εν μέρει το ίδιο το πλαίσιο ανάπτυξης μέσω της μονάδας URL Dispatcher και των προβολών.

URL Dispatcher & Configuration Η μονάδα URL Dispatcher είναι ο εσωτερικός μηχανισμός του Django υπεύθυνος για την αντιστοίχιση των υπερσυνδέσμων με τις κατάλληλες προβολές. Πρόσβαση σε αυτό είναι διαθέσιμη μέσω της μονάδας URL Configuration (ρύθμιση υπερσυνδέσμων), που με τη χρήση κανονικών εκφράσεων (regular-expressions) ορίζει τους υπερσύνδεσμους της εφαρμογής. Οι υπερσύνδεσμοι μπορούν να δημιουργηθούν δυναμικά και κατά την εκτέλεση της εφαρμογής.

Προβολές (Views) Οι προβολές περιγράφουν τα δεδομένα που παρουσιάζονται στον χρήστη. Είναι υπεύθυνα για την απαιτούμενη επεξεργασία των μοντέλων και ενθυλακώνουν τη λογική υπεύθυνη για την επεξεργασία των αιτημάτων και την επιστροφή των απαντήσεων. Το Django διαθέτει δυο τύπους προβολών, τις προβολές βασισμένες σε

συναρτήσεις και τις προβολές βασισμένες σε κλάσεις. Αν και οι δύο τύποι μπορούν να επιτύχουν τις ίδιες λειτουργίες, η κύρια διαφορά τους είναι στην μεθοδολογία και απλότητα υλοποίησης της απαιτούμενης λειτουργίας.

Προβολές Βασισμένες σε Συναρτήσεις Προβολές βασισμένα σε συναρτήσεις ή *Function Based Views*, είναι συναρτήσεις που δέχονται ως παράμετρο ένα αίτημα και επιστρέφουν μια απάντηση. Η απάντηση μπορεί να είναι το περιεχόμενο HTML μιας ιστοσελίδας, μια ανακατεύθυνση, ένα έγγραφο XML και άλλα.

Προβολές Βασισμένες σε Κλάσεις Προβολές βασισμένες σε κλάσεις ή *Class Based Views* είναι ειδικές κλάσεις που επιτρέπουν τη δόμηση των προβολών. Δημιουργούνται με επαναχρησιμοποίηση πηγαίου κώδικα αξιοποιώντας την ιδιότητα της κληρονομικότητας και της σύνθεσης. Όλες οι προβολές στο Django είναι παράγωγοι της κλάσης βάσης *View*. Επίσης είναι διαθέσιμες κάποιες παραμετρικές κλάσεις (Generic Classes) με συνηθισμένα ιδιώματα και λειτουργίες.

3.3 Javascript

Η JavaScript είναι μια γλώσσα προγραμματισμού που χρησιμοποιείται παραδοσιακά στην ανάπτυξη ιστοσελίδων. Δεν χρησιμοποιείται μόνο από την πλειοψηφία των σύγχρονων ιστοσελίδων και τους σύγχρονους φυλλομετρητές, αλλά και ηλεκτρονικές συσκευές όπως έξυπνα κινητά τηλέφωνα και κονσόλες ηλεκτρονικών παιχνιδιών. Η JavaScript δημιουργήθηκε από την *Netscape* το 1995, και το επίσημο όνομα της είναι *ECMAScript*.

3.3.1 Κύρια Χαρακτηριστικά και Εφαρμογή

Η JavaScript είναι μια υψηλού επιπέδου, δυναμική μεταγλωττισμένη γλώσσα προγραμματισμού. Αυτό σημαίνει ότι πρόκειται για μια γλώσσα προγραμματισμού της οποίας οι περισσότερες εκτελέσεις εντολών γίνονται άμεσα χωρίς να έχει προηγηθεί μετάφραση του προγράμματος. Ο διερμηνέας εκτελεί το πρόγραμμα άμεσα, μεταγλωττίζοντας κάθε δήλωση σε μία ακολουθία υποπρογραμμάτων.

Συχνά χρησιμοποιείται ως μέρος των φυλλομετρητών, των οποίων οι υλοποιήσεις επιτρέπουν αλληλοεπίδραση με το χρήστη, για τον έλεγχο του προγράμματος περιήγησης, για να εφαρμόζουν ασύγχρονη επικοινωνία, και να τροποποιήσουν το περιεχόμενο του εγγράφου που εμφανίζεται. Χρησιμοποιείται επίσης σε προγραμματισμό από την πλευρά του διακομιστή, την ανάπτυξη παιχνιδιών και τη δημιουργία εφαρμογών. Είναι μια ισχυρή γλώσσα που βοηθά τους σχεδιαστές ιστοσελίδων στον διαδραστικό σχεδιασμό δικτυακών τόπων με απλό και αποτελεσματικό τρόπο. Δίνει την δυνατότητα στους σχεδιαστές να παρουσιάζουν δυναμικά κείμενο και άλλα στοιχεία κατά τον χρόνο εκτέλεσης. Μπορεί και προσαρμόζει τις σελίδες, μέσω μηχανισμών ανίχνευσης, ανάλογα με τις απαιτήσεις των φυλλομετρητών. Γι' αυτό, η JavaScript έχει κυριαρχήσει και χρησιμοποιείται από πολλές διαφορετικές πλατφόρμες. Σαν σύγχρονη γλώσσα είναι κατάλληλη για αντικειμενοστραφή και συναρτησιακό προγραμματισμό.

Κάθε σύγχρονη γλώσσα πρέπει μια βιβλιοθήκη ή μια προγραμματιζομένη διεπαφή αποτελούμενη από βασικές λειτουργίες για την εκτέλεση του προγράμματος όπως η

είσοδος και η έξοδος. Ο πυρήνας της JavaScript ορίζει μια βασική προγραμματιζομένη διεπαφή για την εργασία με στοιχεία όπως κείμενο, πίνακες και ημερομηνίες, αλλά δεν περιλαμβάνει καμία λειτουργία εισαγωγής ή εξόδου. Για την είσοδο και έξοδο, καθώς και πιο εξελιγμένα χαρακτηριστικά, όπως η δικτύωση, αποθήκευση, και γραφικά είναι υπεύθυνο το περιβάλλον εκτέλεσης, μέσα στο οποίο είναι ενσωματωμένος ο διερμηνέας της JavaScript, όπου συνήθως είναι ένας φυλλομετρητής. Επιπλέον, οι σύγχρονες εκδόσεις της Javascript καθορίζουν νέες δυνατότητες για την ανάπτυξη λογισμικού μεγάλης κλίμακας.

3.3.2 AJAX

Ο όρος AJAX ή Asynchronous JavaScript και XML αντιπροσωπεύει ένα σύνολο τεχνικών για τη δημιουργία καλύτερων, πιο γρήγορων, ασύγχρονων και διαδραστικών εφαρμογών διαδικτύου με τη χρήση των τεχνολογιών XML, HTML, CSS και JavaScript.

3.4 Πλαίσιο Ανάπτυξης AngularJS

Το AngularJS είναι ένα πλαίσιο ανάπτυξης διαδικτυακών εφαρμογών ανοιχτού πηγαίου κώδικα δημιουργημένο με τη γλώσσα προγραμματισμού Javascript και σχεδιασμένο για την ανάπτυξη διαδραστικών εφαρμογών διαδικτύου. Αναπτύχθηκε το 2009 από τους *Miško Hevery* και *Adam Abrons* στην εταιρία *Brat Tech LLC* ως ιδιόκτητο λογισμικό, και αργότερα κυκλοφόρησε ως βιβλιοθήκη ανοιχτού πηγαίου κώδικα. Είναι ένα από τα πλαίσια ανάπτυξης διαδικτυακών εφαρμογών με τη μεγαλύτερη επιρροή, και συντηρείται από την κοινότητα αλλά και από μεγάλες εταιρίες του χώρου, όπως η Google.

3.4.1 Κύρια χαρακτηριστικά και δυνατότητες

Το AngularJS είναι σχεδιασμένο με έμφαση στη δημιουργία διαδικτυακών εφαρμογών μιας σελίδας, ακολουθώντας αρχιτεκτονική τύπου MVC. Έχει υιοθετήσει φιλοσοφία που παροτρύνει τον διαχωρισμό της ανάπτυξης των διεπαφών χρήστη με τη λογική της εφαρμογής, με κύρια μέθοδο την προσαρμογή και επέκταση της γλώσσας σήμανσης HTML έτσι ώστε να διευκολύνει τη κατασκευή διαδραστικών και σύγχρονων διαδικτυακών εφαρμογών.

Ένα από τα πιο σημαντικά χαρακτηριστικά του AngularJS είναι η δυνατότητα αυτόματου συγχρονισμού μεταξύ των μοντέλων και της προβολής, και άλλες σημαντικές ιδιότητες συμπεριλαμβάνουν:

- Διαχωρισμός λογικής, δεδομένων και προβολής
- Υπηρεσίες AJAX
- Εισαγωγή εξαρτήσεων
- Υποστήριξη ιστορικού φυλλομετρητή
- Σύστημα ελέγχου μονάδων

3.4.2 Αρχιτεκτονική

Το AngularJS ακολουθεί την αρχιτεκτονική MVC και ενθαρρύνει την ”χαλαρή σύνδεση” μεταξύ της παρουσίασης, των δεδομένων και της λογικής. Χρησιμοποιώντας μεθόδους εισαγωγής εξαρτήσεων, το AngularJS παρέχει παραδοσιακές υπηρεσίες διακομιστή-πελάτη σε διαδικτυακές εφαρμογές. Ως αποτέλεσμα οι εφαρμογές AngularJS έχουν λιγότερες απαιτήσεις.

Το AngularJS επεκτείνει και προσαρμόζει τα στοιχεία της γλώσσας HTML. Στο AngularJS τα αρχεία που περιέχουν κώδικα HTML ονομάζονται templates και αποτελούν το επίπεδο προβολής των εφαρμογών. Τα templates επεξεργάζονται κατά την εκκίνηση και λειτουργία της εφαρμογής από τον ειδικό τον μεταφραστή του AngularJS, και το προϊόν του είναι το τελικό αποτέλεσμα που εμφανίζεται στον φυλλομετρητή. Οι επιπρόσθετες λειτουργίες των templates υλοποιούνται με ένα σύστημα ειδικής σήμανσης, τις οδηγίες και τις εκφράσεις.

Οι οδηγίες είναι προσαρμοσμένα στοιχεία όπως ονόματα στοιχείων, ιδιότητες, σχόλια ή κλάσεις της γλώσσας CSS που οδηγούν τον ειδικό μεταφραστή του AngularJS έτσι ώστε να επισυνάψει μια συγκεκριμένη συμπεριφορά σε αυτό το στοιχείο, ή ακόμα και να μετατρέψει το στοιχείο και τα παιδιά του σε διαφορετική μορφή.

```
<!doctype html>
<html lang="en" ng-app="myApp">
    <head>
    </head>
    <body>
    </body>
</html>
```

Οι εκφράσεις ένας ειδικός τύπος σήμανσης, αξιολογούνται από τον μεταφραστή κατά την εκτέλεση της εφαρμογής, και παίρνει την θέση τους το αποτέλεσμα της αξιολόγησης της έκφρασης. Οι εκφράσεις αποτελούν τον μηχανισμό που επιτρέπει στο AngularJS πρόσβαση σε μεταβλητές στο επίπεδο της προβολής.

```
    1+2={{1+2}}
```

```
<ul class="list-unstyled">
    <li ng-repeat="note in notes | filter:searchText " ng-controller="
   ItemController" class="notelist" ng-click="open(note)">

        <div><h4>{{note.title}}</h4></div>
        <div><h6>{{note.body|limitTo:letterLimit}}</h6></div>

    </li>
</ul>
```

Επιπλέον, οι ελεγκτές στο AngularJS δημιουργούν και ορίζουν την αρχική κατάσταση και επιθυμητή συμπεριφορά σε ειδικά αντικείμενα scope που δρουν ως συνδετικοί κρίκοι μεταξύ της προβολής και της λογικής της εφαρμογής. Με τη χρήση των παραπάνω χαρακτηριστικών, εφαρμόζεται η λειτουργία του αυτόματου συγχρονισμού μεταξύ των μοντέλων και των προβολών του AngularJS.

Κεφάλαιο 4

Αρχικός Σχεδιασμός και Προδιαγραφές

Στα πλαίσια αυτής της εργασίας, υλοποιείται μια πλήρης υπηρεσία αρχιτεκτονικής REST. Παρουσιάζεται η περιγραφή της λειτουργίας και απαιτήσεων του αρχικού προβλήματος ή υπηρεσίας, η ανάλυση του, και ο αρχικός σχεδιασμός της που καλείται να λύσει το πρόβλημα.

4.1 Περιγραφή απαιτήσεων

Το δεδομένο πρόβλημα, είναι η ανάπτυξη μιας υπηρεσίας διαδικτύου διαχείρισης ηλεκτρονικών προσωπικών σημειώσεων, με επιπλέον χαρακτηριστικά την εύκολη επεκτασιμότητα και τη δυνατότητα παράλληλης λειτουργίας πολλαπλών διεπαφών χρήστη.

4.2 Ανάλυση λειτουργιών

Από την περιγραφή των απαιτήσεων της υπηρεσίας, προκύπτει το βασικό σύνολο των λειτουργιών και ενεργειών. Τα διαχειριζόμενα δεδομένα ανήκουν σε δύο ευρύτερες και αλληλένδετες κατηγόριες, τα στοιχεία των χρηστών και τα στοιχεία των ηλεκτρονικών σημειώσεων. Ειδικότερα, οι κύριες ενέργειες που απαιτούνται για τη λειτουργία της υπηρεσίας αφορούν την ταυτοποίηση και πιστοποίηση των χρηστών καθώς και τις μεθόδους διαχείρισης των ηλεκτρονικών σημειώσεων και δεδομένων τους.

Για τη λειτουργία της πιστοποίηση των χρηστών, απαιτούνται οι παρακάτω ενέργειες:

- Δημιουργία νέου λογαριασμού χρήστη
- Έλεγχος ταυτότητας και παραχώρηση πιστοποιητικού

Εφόσον πραγματοποιηθεί η πιστοποίηση ταυτότητας, οι δυνατές ενέργειες που επιτρέπονται στους πιστοποιημένους χρήστες αποτελούνται από:

- Δημιουργία καινούργιας σημείωσης
- Ανάκτηση όλων των σημειώσεων

- Τροποποίηση υπάρχουσας σημείωσης
- Διαγραφή υπάρχουσας σημείωσης
- Αναζήτηση σημείωσης μέσω τίτλου και περιεχομένου

Επιπλέον, ο πιστοποιημένος χρήστης πρέπει να έχει την δυνατότητα τροποποίησης των στοιχείων του.

4.3 Αρχιτεκτονική

Σύμφωνα με την ανάλυση των σχεδιαστικών απαιτήσεων, προκύπτει ως πρωταρχικός στόχος στη σχεδίαση της υπηρεσίας η εφαρμογή αρχιτεκτονικής που επιτρέπει λειτουργική ανεξαρτησία μεταξύ των κυρίων τμημάτων της υπηρεσίας. Υιοθετώντας την αρχιτεκτονική REST, επιτυγχάνονται οι στόχοι του σχεδιασμού της υπηρεσίας, και με τη χρήση του πλαισίου ανάπτυξης Django επιτυγχάνεται επιπλέον και ο διαχωρισμός της βάσης δεδομένων από την κεντρική μονάδα της υπηρεσίας.

Η υπηρεσία χωρίζεται σε διαφορετικές επιμέρους λειτουργικές μονάδες:

- Κεντρική Εφαρμογή/Μονάδα
- Προγραμματιζόμενη Διεπαφή
- Βάση Δεδομένων
- Διεπαφή χρήστη

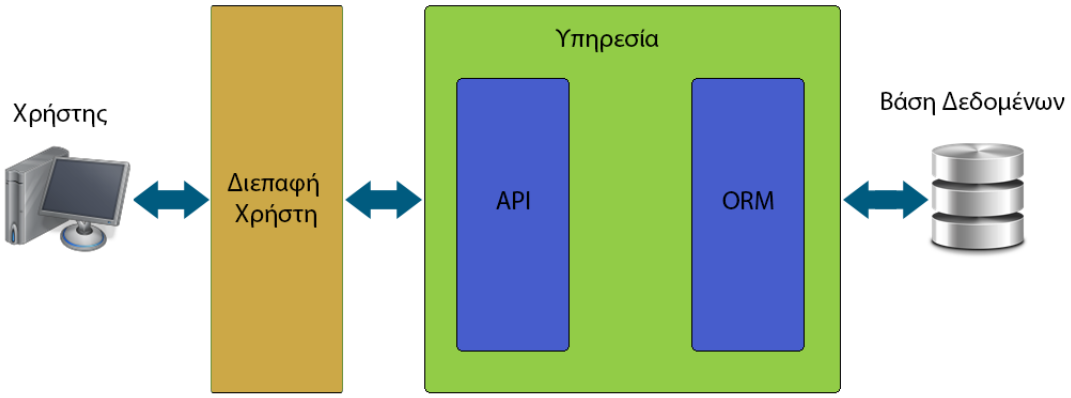
Ειδικότερα η κεντρική εφαρμογή ή μονάδα και βασικό στοιχείο της υπηρεσίας είναι μια REST εφαρμογή, υπεύθυνη για την πρόσβαση στα δεδομένα και για την κατανομή τους στους πελάτες μέσω μιας προγραμματιζόμενης διεπαφής, δηλαδή τις εφαρμογές διεπαφής χρήστη της υπηρεσίας.

Αντίστοιχα, οι εφαρμογές διεπαφής χρήστη αποτελούν ανεξάρτητες μονάδες με μοναδικό δίαυλο επικοινωνίας τα τελικά σημεία της προγραμματιζόμενης διεπαφής της κεντρική μονάδας, μέσω της οποίας μπορούν να αιτηθούν τις λειτουργίες δημιουργίας, ανάκτησης ή τροποποίησης δεδομένων. Τα δεδομένα της υπηρεσίας, αποθηκευμένα σε βάση δεδομένων είναι άμεσα προσβάσιμα μόνο από την κεντρική εφαρμογή.

Ακολουθώντας τα παραπάνω πρότυπα επιτυγχάνεται η επιθυμητή ανεξαρτησία μεταξύ των λειτουργικών μονάδων της υπηρεσίας, και κατ' επέκταση οι ζητηθείσες δυνατότητες μελλοντικής επέκτασης και ανάπτυξης, καθώς είναι δυνατή η αντικατάσταση των επιμέρους μονάδων με διαφορετικές, που εφόσον τηρούν τα καταγεγραμμένα πρότυπα και τις προδιάγραφες είναι συμβατές μεταξύ τους και η προσθήκη καινούργιων παράλληλων διασαφών χρήστη για άλλες πλατφόρμες υπολογιστικών συστημάτων.

4.3.1 Προγραμματιζόμενη Διεπαφή

Για την έναρξη της ανάπτυξης των επιμέρους τμημάτων της υπηρεσίας, είναι απαραίτητη η οριστικοποίηση των σημείων επαφής ή τελικών σημείων της προγραμματιζόμενης διεπαφής σύμφωνα με τις προδιαγραφές που έχουν τεθεί, καθώς αποτελούν το κοινό σημείο μεταξύ της κεντρικής εφαρμογής και των εφαρμογών διεπαφής.



Σχήμα 4.1: Σχεδιασμός Υπηρεσίας

Η αρχική οριστικοποίηση των σημείων επαφής δεν εμποδίζει επεκτάσεις των δυνατοτήτων της υπηρεσίας σε κανένα στάδιο της ανάπτυξης ή της διάρκειας ζωής της, αλλά υλοποιείται διαχείριση διαφορετικών εκδόσεων έτσι ώστε να διατηρηθεί συμβατότητα μεταξύ των διαφορετικών τμημάτων. Ως αποτέλεσμα, στην εισαγωγή νέων χαρακτηριστικών στην υπηρεσία επεκτείνεται η προγραμματιζόμενη επαφή και συντηρούνται παράλληλα διαφορετικές εκδόσεις, για όσο χρονικό διάστημα κρίνεται απαραίτητο από τους διαχειριστές της υπηρεσίας.

Η πρώτη έκδοση των σημείων επαφής της προγραμματιζόμενης διεπαφής αποτελείται από τους παρακάτω υπερσυνδέσμους, που υλοποιούν μέσω της κεντρικής εφαρμογής τις ενέργειες που προέκυψαν στην ανάλυση των επιθυμητών λειτουργιών της υπηρεσίας.

- /api/users/detail
- /api/users/login
- /api/users/register
- /api/notes
- /api/notes/id

Αξίζει να σημειωθεί ότι κάποιες λειτουργίες της υπηρεσίας δεν υλοποιούνται ως σημεία επαφής της προγραμματιζόμενης διεπαφής αλλά υλοποιούνται από την διεπαφή χρήστη χρησιμοποιώντας προσωρινά αποθηκευμένα δεδομένα για λόγους απόδοσης και σταθερότητας, όπως για παράδειγμα η λειτουργία της αναζήτησης.

Κεφάλαιο 5

Υπηρεσία

Ο σχεδιασμός του πλαισίου ανάπτυξης Django ενθαρρύνει τη δημιουργία διαφορετικών εφαρμογών για κάθε διαφορετική πτυχή της κύριας εφαρμογής. Για την κάλυψη των επιθυμητών προδιαγραφών του προγράμματος, δημιουργούνται δύο εφαρμογές, η εφαρμογή accounts και εφαρμογή notes, όπου ενθυλακώνουν τη λογική και τις λειτουργίες για τους λογαριασμούς χρηστών και για τις σημειώσεις αντίστοιχα.

5.1 Αρχιτεκτονική

Με τον διαχωρισμό της κύριας εφαρμογής σε διακριτές εφαρμογές, διευκολύνεται η ανάπτυξη και η αποσφαλμάτωση των επιμέρους λειτουργιών του προγράμματος και απλοποιείται η δυνατότητα μελλοντικών επεκτάσεων. Είναι δυνατόν να επεκταθούν οι λειτουργίες της υπηρεσίας με νέα χαρακτηριστικά, δημιουργώντας καινούριες εφαρμογές χωρίς να χρειαστούν αλλαγές στον πηγαίο κώδικα των ήδη υπάρχον εφαρμογών.

Οι δυο εφαρμογές που χρησιμοποιούνται περιέχουν όλη τη λογική των μοντέλων τους. Η μονάδα *URL Configuration*, οι προβολές και οι υπόλοιπες απαραίτητες μονάδες, ανήκουν στην αντίστοιχη εφαρμογή. Ως αποτέλεσμα, οι εφαρμογές είναι ανεξάρτητες προγραμματιστικά μεταξύ τους.

Η εφαρμογή accounts είναι πλήρως αυτόνομη, αλλά η εφαρμογή notes παραπέμπει στο μοντέλο των χρηστών της εφαρμογής accounts. Ωστόσο, με τη χρήση της ιδιότητας settings.AUTH_USER_MODEL και της μεθόδου get_user_model() δεν είναι απαραίτητο να γίνει αναφορά στην εφαρμογή accounts άμεσα, επιτυγχάνοντας έτσι αυτονομία για την εφαρμογής notes. Ως αποτέλεσμα, μπορεί να αντικατασταθεί το μοντέλο χρήστη χωρίς να επηρεαστεί η συμπεριφορά του εφαρμογής notes.

Επιπλέον, γίνεται χρήση της ελεύθερα διαθέσιμης εφαρμογής *Django Rest Framework*, ένα σύνολο εργαλείων για τη δημιουργία REST εφαρμογών, συμπεριλαμβανόμενων μοντέλων σχεδιασμένα να λειτουργούν με έγγραφα JSON, πολιτικές πιστοποίησης και άλλα.

5.1.1 Models - Database

Σύμφωνα με τον αρχικό σχεδιασμό, η κύρια εφαρμογή διαχειρίζεται δυο ομάδες δεδομένων. Στο Django υλοποιούνται ως δυο διαφορετικά μοντέλα, το μοντέλο χρήστη, *User* και το μοντέλο σημειώσεων, *Note*.

5.1.2 Models

Το Django διαθέτει προεγκατεστημένο σύστημα πιστοποίησης το οποίο συμπεριλαμβάνει μοντέλο χρήστη, αλλά για να καλυφθούν οι ανάγκες της υπηρεσίας και για να υπάρχει η δυνατότητα τροποποίησης όλων των χαρακτηριστικών του και εύκολης επέκτασης, γίνεται χρήση προσαρμοσμένου μοντέλου χρήστη.

Για τη δημιουργία του προσαρμοσμένου μοντέλου χρήστη, *User*, επεκτείνεται το ήδη υπάρχον μοντέλο του συστήματος πιστοποίησης του Django, κληρονομώντας από τη κλάση *AbstractUser*, που παρέχει την βασική υλοποίηση του μοντέλου χρηστή, και υλοποιείται προσαρμόζοντας τα πεδία με την επιθυμητή συμπεριφορά. Συγκεκριμένα, η βασική υλοποίηση επεκτείνεται ορίζοντας το πεδίο *email* μοναδικό, προσαρμόζοντας τα μεγέθη των πεδίων χαρακτήρων, και υλοποιώντας τις απαιτούμενες βοηθητικές μεθόδους *get_absolute_url()*, *get_full_name()*, *get_short_name()* και *email_user()*.

Επιπλέον, υλοποιείται το βοηθητικό μοντέλο UserManager κληρονομώντας από την κλάση BaseUserManager, στο οποίο υλοποιείται η μέθοδος δημιουργίας κλειδιού(token) κατά τη δημιουργία νέου χρήστη, καθώς και οι απαιτούμενες μέθοδοι.

```
class User(AbstractBaseUser, PermissionsMixin):
    username = models.CharField(_('user name'), max_length=64, unique=True)
    # Setting e-mail as a unique field
    email = models.EmailField(_('email address'), max_length=254, blank=
    False, unique=True)
    first_name = models.CharField(_('first name'), max_length=30, blank=
    True)
    last_name = models.CharField(_('last name'), max_length=30, blank=True)
    is_staff = models.BooleanField(_('staff status'), default=False,
        help_text=_('Designates whether the user can log into this admin '
                    'site.'))
    is_active = models.BooleanField(_('active'), default=True,
        help_text=_('Designates whether this user should be treated as '
                    'active. Unselect this instead of deleting accounts.'))
    date_joined = models.DateTimeField(_('date joined'), default=timezone.
    now)

    objects = UserManager()

    USERNAME_FIELD = 'username'
    REQUIRED_FIELDS = []

    class Meta:
        verbose_name = _('user')
        verbose_name_plural = _('users')

    def get_absolute_url(self):
        return "/users/%s/" % urlquote(self.email)

    def get_full_name(self):
        full_name = '%s %s' % (self.first_name, self.last_name)
        return full_name.strip()

    def get_short_name(self):
        return self.first_name

    def email_user(self, subject, message, from_email=None):
```

```
        send_mail(subject, message, from_email, [self.email])
```

Το μοντέλο *Note* αντιπροσωπεύει τις σημειώσεις που ανήκουν στον χρήστη, και συνδέεται με το μοντέλο *User* με μια συσχέτιση τύπου πολλών-προς-ένα που δημιουργείται με το πεδίο *owner*.

Τα πεδία που περιέχουν το περιεχόμενο του *Note*, τον τίτλο και το σώμα εκφράζονται από τα πεδία *title* και *body* αντίστοιχα, και τέλος τα βοηθητικά πεδία ημερομηνίας δημιουργίας και ημερομηνίας τελευταίας τροποποίησης, *data_created* και *date_modified* που μπορούν να χρησιμοποιηθούν για λειτουργίες όπως ταξινόμηση και έλεγχο συγχρονισμού, στα οποία με χρήση των παραμέτρων *auto_now_add* και *auto_now* το Django τα ανανεώνει κατάλληλα κατά την δημιουργία ή τροποποίηση των στιγμιότυπων.

```
from accounts.models import User

class Note(models.Model):
    owner = models.ForeignKey(User) #add related key etc
    title = models.CharField(max_length=200, blank=True)
    body = models.TextField(blank=True)
    date_created = models.DateTimeField(auto_now_add=True)
    date_modified = models.DateTimeField(auto_now=True)
```

Αξίζει να σημειωθεί πως σε περίπτωση που υπάρξει η ανάγκη προσθήκης νέων πεδίων στο μοντέλο, δεν θα χρειαστούν επιπλέον αλλαγές στην εφαρμογή, καθώς οι προβολές τροποποιούν τα στιγμιότυπα με τα νέα πεδία χωρίς αλλαγές στη λογική τους.

Η μονάδα *ORM* μεταφράζει τα μοντέλα της εφαρμογής σε κατάλληλες δηλώσεις SQL, συμβατές με τη βάση δεδομένων που χρησιμοποιεί η υπηρεσία. Οι πίνακες δημιουργούνται σύμφωνα με προκαθορισμένους κανόνες και παραμέτρους, όπως οι ονομασίες τους που προσαρμόζονται με βάση την ονομασία της εφαρμογής και την ονομασία του κάθε μοντέλου.

```
BEGIN;
CREATE TABLE "notes_note" (
  "id" integer NOT NULL PRIMARY KEY,
  "owner_id" integer NOT NULL REFERENCES "accounts_user" ("id"),
  "title" varchar(200) NOT NULL,
  "body" text NOT NULL,
  "date_created" datetime NOT NULL,
  "date_modified" datetime NOT NULL,
);
```

Κατά τη δημιουργία της βάσης δεδομένων, εκτός από τα μοντέλα που έχουν δηλωθεί και τις συσχετίσεις τους, το Django προσθέτει αυτόματα επιπλέον πίνακες που χρησιμοποιούνται σε διάφορες λειτουργίες όπως η πιστοποίηση, οι ομάδες χρηστών κτλ.

5.1.3 Serializers

Για τη μετατροπή των δεδομένων από μορφή τύπου JSON σε στιγμιότυπα των μοντέλων και αντίστροφα, γίνεται χρήση της μονάδας *Serializers* της εφαρμογής Django Rest Framework. Οι Serializers επικυρώνουν τα δεδομένα, και τα μετατρέπουν στην κατάλληλη μορφή. Είναι απαραίτητος ένας Serializer για κάθε μοντέλο που επιστρέφει δεδομένα μέσω του API.

Ο Serializer για το μοντέλο χρήστη:

```
class UserSerializer(serializers.ModelSerializer):

  class Meta:
    model = User
    fields = ('id','username','email',first_name','last_name','password')

  def restore_object(self, attrs, instance=None):
    return user
```

Ο Serializer του μοντέλου Note:

```
class NoteSerializer(serializers.ModelSerializer):
  owner = serializers.Field(source='owner.username')
  class Meta:
    model = Note
    fields = ('id','title','body','date_created','date_modified','tag')
```

5.1.4 URLs

Με χρήση της ειδικής μονάδας URL Configuration, ορίζονται οι υπερσύνδεσμοι όλων των τελικών σημείων του προγράμματος, και οι προβολές που αντιστοιχούν σε αυτά.

Οι υπερσύνδεσμοι με διαδρομή api/notes/ αντιστοιχούν στις προβολές των σημειώσεων και με διαδρομή api/users/ στις αντίστοιχες προβολές για τον χρήστη.

```
urlpatterns = patterns('',
    url(r'^$', include('notes.urls')),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^api-auth/', include('rest_framework.urls', namespace='
   rest_framework')),
    url(r'^api/notes/$', views.NoteList.as_view()),
    url(r'^api/notes/(?P<pk>[0-9]+)/$', views.NoteDetail.as_view()),
    url(r'^api/users/$', views.UserList.as_view()),
    url(r'^api/users/(?P<pk>[0-9]+)/$', views.UserDetail.as_view()),
    url(r'^api/users/detail/$', views.UserDetail.as_view()),
    url(r'^api/users/register', views.UserCreate.as_view()),
    #url(r'^api/users/login', views.UserLogin.as_view()),
    #url(r'activate/(?P<key>[0-9]+)/$', views.UserActivate.as_view())
)
```

Επιπλέον, ο υπερσύνδεσμος /api/users/login/ καλεί μια προκατασκευασμένη προβολή του συστήματος πιστοποίησης της εφαρμογής Django Rest Framework, όπου δέχεται ως παραμέτρους το όνομα και κωδικό χρήστη και επιστρέφει το κλειδί πιστοποίησης, που χρησιμοποιείται μετέπειτα από τον πελάτη σε όλα τα αιτήματα που απαιτούν την πιστοποίηση του χρήστη.

```
urlpatterns += patterns('',
    url(r'^api/users/login/', 'rest_framework.authtoken.views.
   obtain_auth_token')
)
```

5.1.5 Προβολές

Για τη σχεδίαση των προβολών γίνεται χρήση των *Προβολών Βασισμένων σε Κλάσεις*. Υλοποιείται ένα μια προβολή για καθε τελικό σημείο της προγραμματιζόμενης διεπαφής όπως ορίζεται από τη μονάδα URL Configuration. Γίνεται χρήση των ειδικών προβολών που παρέχει το Django Rest Framework, και ανάλογα την περίπτωση χρήσης χρησιμοποιείται ως κλάση βάσης η κλάση *APIView* ή κάποια από τις διαθέσιμες παραμετρικές προβολές.

Για την εφαρμογή notes, αξιοποιώντας τις παραμετρικές κλάσεις της εφαρμογής Django Rest Framework, υλοποιούνται οι προβολές *NoteList* και *NoteDetail*.

Η προβολή NoteList, κληρονομεί από τη κλάση *ListCreateAPIView* και επιτρέπει τις HTTP μεθόδους *GET* και *POST*, και χρησιμοποιείται για να επιστρέψει μια λίστα με όλα τα μοντέλα note ή για να δημιουργήσει καινούριο note.

```
class NoteList(generics.ListCreateAPIView):
    """
    List all notes, or create a new note.
    """
    serializer_class = NoteSerializer

    def get_queryset(self):
        """
        This view should return a list of all notes
        for the currently authenticated user.
        """
        # Check if user is authenticated and
        # if not raise NotAuthenticated exception
        user = self.request.user
        if user.is_authenticated():
            return user.note_set.all()
        else:
            raise AuthenticationFailed(detail=None)

    # pre_save hook is called before saving the object(new note)
    # it is used to set the object owner/user.
    def pre_save(self, obj):
        if not self.request.user.is_authenticated():
            raise AuthenticationFailed(detail=None)
```

```
        obj.owner = self.request.user
```

Η προβολή NoteDetail, κληρονομεί από τη κλάση *RetreiveUpdateDestroyAPIView* και επιτρέπει μεθόδους GET, PUT, PATCH και DELETE. Μπορεί να χρησιμοποιηθεί για την ανάκτηση συγκεκριμένου note με χρήση στο URL του *note_id*, καθώς και για ανανέωση ολόκληρου του αντικειμένου ή ενός μόνο πεδίου με τη μέθοδο PATCH, και για τη διαγραφή του αντικειμένου.

```
class NoteDetail(generics.RetrieveUpdateDestroyAPIView):
        queryset = Note.objects.all()
        serializer_class = NoteSerializer
        permission_classes = (IsOwner, permissions.IsAuthenticated,)
```

```
class IsOwner(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        return obj.owner == request.user
```

Για την εφαρμογή *accounts* υλοποιούνται δύο κλάσεις, η *UserDetail* και η *UserCreate*. Στη προβολή *UserDetail* γίνεται χρήση της παραμετρικής κλάσης *RetrieveUpdate-APIView*, ενώ για τη δημιουργία νέου χρήστη στη προβολή *UserCreate* γίνεται χρήση της κλάσης βάσης APIView και υλοποιείται η συνάρτηση post έτσι ώστε να προσαρμοστεί η λειτουργία δημιουργίας νέου χρήστη με τις επιπρόσθετες βοηθητικές μεθόδους δημιουργίας κλειδιού-token και δημιουργίας κλειδιού ενεργοποίησης.

```
class UserDetail(UserMixin, generics.RetrieveUpdateAPIView):
    queryset = User.objects.all()
    serializer_class = UserDetailSerializer
)
```

Αναλυτικά, η προβολή *UserCreate* υλοποιείται με χρήση της κλάσης βάσης APIView της εφαρμογής Django Rest Framework. Η λειτουργία της προβολής είναι μόνο η δημιουργία καινούριου χρήστη, οπότε υλοποιείται μόνο η μέθοδος POST. Αρχικά μέσω του Serializer UserSerializer μετατρέπει τα δεδομένα που περιέχονται στο έγγραφο JSON σε κατάλληλη μορφή, και αφού ελέγξει την εγκυρότητα τους σώζει το στιγμιότυπο αυτό. Εδώ προστίθεται η επιπλέον επιθυμητή λειτουργία της δημιουργίας κλειδιού-token για τον νέο χρήστη καθώς και βοηθητικών δεδομένων για την ενεργοποίηση του. Αν τα δεδομένα είναι έγκυρα η προβολή επιστρέφει ένα αντικειμένο HTTP με το αναγνωριστικό μήνυμα *201 Created*, ή διαφορετικά *400 Bad Request*.

```
class UserCreate(APIView):

    def post(self, request):
        serialized = UserSerializer(data=request.DATA)
        if serialized.is_valid():
            serialized.save()
            username = serialized.data['username']
```

```
            user = User.objects.get(username=username)
            token = Token.objects.create(user=user)
            #utils.create_user_data(user)
            return Response(serialized.data, status=status.HTTP_201_CREATED
    )
        else:
            return Response(serialized.errors, status=status.
    HTTP_400_BAD_REQUEST)
```

5.1.6 Unit Testing

Με τη χρήση βοηθητικών κλάσεων του Django και του της εφαρμογής Django Rest Framework, αναπτύσσονται μονάδες ελέγχου κατά τη διαδικασία της ανάπτυξης της εφαρμογής, ελέγχοντας κάθε επιμέρους λειτουργία. Οι έλεγχοι εκτελούνται μέσω του κελύφους ανάπτυξης του Django σε απομονωμένο περιβάλλον με προσωρινά δεδομένα χωρίς να μολύνουν τους πίνακες της κύριας βάσης δεδομένων.

Ενδεικτικά μερικές μονάδες ελέγχου:

Βεβαίωση για τη δημιουργία καινούργιου χρήστη

```
class UserAPITests(APITestCase):
    def test_create_account(self):
        """
        Ensure we can create a new user account.
        """
        url = '/api/users/register/'
        data = {'username': 'TestUser',
                'email': 'test@test.com',
                'password': 'test',
                'first_name': 'TestFirst',
                'last_name': 'TestLast'}
        response = self.client.post(url, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
```

Βεβαίωση ότι οι σημειώσεις του χρήστη δεν είναι προσβάσιμες χωρίς πιστοποίηση

```
    def test_retrieve_no_user_notes(self):
        """
        Ensure we can't retrieve user notes without authentication
        """
        client = APIClient()
        url = '/api/notes/'
        response = client.get(url, format='json')
        self.assertEqual(response.status_code, status.HTTP_401_UNAUTHORIZED
    )
```

5.2 Διάγραμμα ροής αιτήσεων-απαντήσεων

Ως REST εφαρμογή, μπορεί να γίνει εξέταση του κύκλου αιτημάτων-απαντήσεων χωρίς απαραίτητα τη χρήση εφαρμογής φυλλομετρητή, αλλά με τη χρήση κάποιου εργαλείου μεταφοράς αρχείων σύνταξης URL, όπως το cURL. Στη περίπτωση αίτησης δημιουργίας νέας σημείωσης:

```
$curl -i \
    -H "Content-Type: application/json"
    -X POST -d '{"username":"xyz","title":"xyz","body":"xyz"}'
    http://localhost:8080/api/notes/
```

Αρχικά το αίτημα θα επεξεργαστεί από τη μονάδα URL Configuration, οπού αναγνωρίζεται ο υπερσύνδεσμος και καλείται η κατάλληλη προβολή.

```
urlpatterns = patterns('',
    url(r'^api/notes/$', views.NoteList.as_view())
)
```

Η προβολή NoteList, παράγωγος κλαση της *ListCreateAPIView* υποστηρίζει αιτήματα POST, και αφού το σύστημα πιστοποίησης εγκρίνει την ταυτότητα του χρήστη, με τη βοήθεια του Serializer NoteSerializer δημιουργεί ένα νέο στιγμιότυπο του μοντέλο Note. Η μονάδα ORM δημιουργεί την κατάλληλη δήλωση SQL για την αποθήκευση του στη βάση δεδομένων.

Τέλος, η προβολή επιστρέφει ένα αντικείμενο HTTP με το αναγνωριστικό 201 Created.

Κεφάλαιο 6

Εφαρμογή

6.1 AngularJS Project

Η κύρια διεπαφή χρήστη της υπηρεσίας αναπτύσσεται με χρήση του πλαισίου ανάπτυξης AngularJS. Όπως και το πλαίσιο ανάπτυξης Django, το AngularJS υποστηρίζει και ενθαρρύνει τον διαχωρισμό της εφαρμογής σε διαφορετικές μονάδες.

6.2 Αρχιτεκτονική

Η εφαρμογή διαχωρίζεται σε διαφορετικές μονάδες, όπου η κάθε μια είναι υπεύθυνη για ένα σύνολο από συγκεκριμένες λειτουργίες. Αποτελείται από τη κεντρική μονάδα myApp η οποία εκτελείται κατά τη φόρτωση της ιστοσελίδας, καθώς και τις βασικές μονάδες common, notes και security.

Ακολουθώντας τις βασικές σχεδιαστικές αρχές του AngularJS, τη δημιουργία εφαρμογών και ιστοσελίδων μιας σελίδας, η διεπαφή της εφαρμογής αποτελείται από ένα κύριο έγγραφο HTML, και οι διεπαφές των επιμέρους μονάδων ενσωματώνονται ανάλογα με την κατάσταση της εφαρμογής σε ένα προκαθορισμένο χώρο στο κύριο έγγραφο.

6.3 Κεντρική Μονάδα myApp

Η μονάδα myApp αποτελεί τον πυρήνα της εφαρμογής. Εκτελείται κατά τη φόρτωση της ιστοσελίδας και περιέχει τα απαραίτητα στοιχεία για τη λειτουργία της εφαρμογής. Περιέχει αναφορές στις υπόλοιπες μονάδες, καθώς και τις λειτουργίες κοινές για ολόκληρη την εφαρμογή και ανεξάρτητες από τη μονάδα εκτέλεσης.

```
var myApp = angular.module('myApp', [
  'ngRoute',
  'authorControllers',
  'security',
  'common',
  'notes',
  'hc.marked',
  'ui.bootstrap',
  ]);
```

Στην κεντρική μονάδα myApp ορίζονται οι διαδρομές και οι υπερσύνδεσμοι της εφαρμογής. Κάθε στοιχείο της λίστας διαδρομών συμπεριλαμβάνει τον υπερσύνδεσμο με το αντίστοιχο έγγραφο HTML, και προαιρετικά κάποιον ελεγκτή.

```
myApp.config(['$routeProvider', function($routeProvider) {
  $routeProvider.
  when('/', {
    templateUrl: 'partials/main.html',
  }).
  when('/list', {
    templateUrl: 'partials/list.html',
    controller: 'ListController'
  }).
  when('/register', {
    templateUrl: 'js/security/register/register.html',
  }).
  when('/login', {
    templateUrl: 'js/security/login/login.html',
  }).
  when('/notes', {
    templateUrl: 'js/notes/notes.html',
  }).
  when('/details', {
    templateUrl: 'js/security/profile/profile.html',
  }).
  otherwise({
    redirectTo: '/'
  });
}])
```

Η κεντρική μονάδα είναι επίσης υπεύθυνη για τη προσαρμογή κάποιων γενικών ρυθμίσεων της εφαρμογής. Εδώ προσαρμόζεται η επικεφαλίδα των αιτημάτων HTTP PATCH, και εκτελείται ο μηχανισμός υπεύθυνος για τον έλεγχο και διατήρηση της σύνδεσης του χρήστη.

```
.config(['$httpProvider', function($httpProvider) {
$httpProvider.defaults.headers.patch = {
    'Content-Type': 'application/json;charset=utf-8'
};
}]);
```

Η προβολή που αντιστοιχεί στην κεντρική μονάδα, είναι το έγγραφο index.html. Ως κεντρική προβολή είναι το σημείο εισαγωγής της εφαρμογής, δημιουργεί τη γενική δομή της ιστοσελίδας και με τη χρήση της προκαθορισμένης οδηγίας ng-view ορίζεται το τμήμα της ιστοσελίδας που θα συμπεριληφθούν δυναμικά οι υπόλοιπες μονάδες κατά την εκτέλεση τους.

```
<!doctype html>
<html lang="en" ng-app="myApp">
<head>
...
</head>
<body>
  <login-toolbar></login-toolbar>
```

```
  <div class="main" ng-view></div>
  ...
  </div>

</body>
</html>
```

6.4 Μονάδα Security

Η μονάδα security περιέχει τις λειτουργίες και τις υπομονάδες της εφαρμογής υπεύθυνες για τη διαχείριση του χρήστη. Η ίδια η μονάδα security δεν περιέχει έγγραφο HTML, αλλά μόνο μεθόδους σχετικές με τη διαχείριση χρήστη που χρησιμοποιούνται από άλλες μονάδες και τμήματα της εφαρμογής. Όπως και η κεντρική μονάδα myApp, η μονάδα security συμπεριλαμβάνει μονάδες-παιδιά.

Για την αποστολή αιτημάτων HTTP το AngularJS προσφέρει δυο διαφορετικές μεθόδους και μπορεί να χρησιμοποιηθεί και πρόσθετο λογισμικό. Εδώ γίνεται χρήση της ενσωματωμένης υπηρεσίας $http. Η υπηρεσία $http είναι μια συνάρτηση που δέχεται μια παράμετρο, ένα αντικείμενο ρυθμίσεων που χρησιμοποιείται για τη δημιουργία του HTTP αιτήματος, και επιστρέφει μια υπόσχεση με δυο βοηθητικές μεθόδους success και error.

6.4.1 Μονάδα Login

Η μονάδα login είναι παιδί της μονάδας security, και είναι υπεύθυνη για τη σύνδεση του χρήστη. Αποτελείται από τη φόρμα σύνδεσης και τον ελεγκτή της φόρμας. Όταν ο χρήστης συμπληρώσει τα πεδία απαραίτητα για τη σύνδεση και υποβάλλει τη φόρμα, ο ελεγκτής καλεί τη συνάρτηση login της μονάδας security με παραμέτρους τα πεδία της φόρμας. Η συνάρτηση login στέλνει το αίτημα POST στον διακομιστή, και περιμένει ως απάντηση το κλειδί-token του χρήστη.

```
$scope.submit = function(form) {
// Καλεί τη συνάρτηση login με τα πεδία της φόρμας
security.login($scope.user.username, $scope.user.password).then(function(
    loggedIn) {
  if ( !loggedIn ) {
    // If we get here then the login failed due to bad credentials
    console.log("Login error");
  }
});
};
```

```
login: function(username, password) {
  var request = $http.post('http://localhost:8000/api/users/login/', {
    username: username, password:password});
  return request.then(function(response) {
    service.token = response.data.token;
    localStorage['clientToken'] = service.token;
  });
```

```
},
```

Αν λάβει κλειδί-token ως απάντηση, το αποθηκεύει στο localStorage, όπου και μια μονάδα αναχαίτισης (interceptor) αυτόματα το προσθέτει στο Header των επόμενων αιτημάτων. Καθώς η αποθήκη localStorage παραμένει αν κλείσει το παράθυρο του φυλλομετρητή, η μονάδα αναχαίτισης αντίστοιχα ελέγχει αν υπάρχει αποθηκευμένο το κλειδί-token όποτε ο χρήστης ανοίγει την ιστοσελίδα.

```
myApp.config(function($httpProvider) {
  $httpProvider.interceptors.push(function($q, $log, $location) {
    return {
      'request': function(config) {
        if (!!localStorage['clientToken']) {
          config.headers.Authorization = 'Token ' + localStorage['
   clientToken'];
    }
        return config;
      },
      'responseError': function(response) {
        if (response.status === 401) {
          window.location.href = '#main';
          console.log("error in interceptor");
        }
        return $q.reject(response);
      }
    };
  });
});
```

6.4.2 Μονάδα Register

Ο ρόλος της μονάδας register είναι η δημιουργία νέου χρήστη. Λειτουργεί αντίστοιχα με τη μονάδα login, και αποτελείται από τη φόρμα δημιουργίας χρήστη και τον ελεγκτή της φόρμας. Όταν ο χρήστης συμπληρώνει τα απαραίτητα πεδία επικυρώνονται ταυτόχρονα σύμφωνα με προκαθορισμένους κανόνες, και αφού υποβάλει τη φόρμα ο ελεγκτής καλεί τη συνάρτηση register που στέλνει το κατάλληλο αίτημα POST στον διακομιστή.

```
angular.module('security.register.form', [])

.controller('RegisterFormController', ['$scope', 'security', '$http',
    function ($scope, security, $http) {

  $scope.user = {};

  $scope.submit = function(form) {
    security.register($scope.user.username, $scope.user.email, $scope.user.
   password);
  };
}
]);
```

Η συνάρτηση register δημιουργεί και εκτελεί το αίτημα HTTP.

```
register: function(username, email, password) {
  data = {
    'username' : username,
    'email' : email,
    'password' : password
  };

  var request = $http.post('http://localhost:8000/api/users/register/',
   data);
  return request.then(function(response) {
    console.log(response.data);
  });
}
```

6.4.3 Μονάδα Profile

Η τελευταία μονάδα-παιδί της μονάδας security, είναι η μονάδα profile. Η μονάδα profile λειτουργεί με αντίστοιχο τρόπο με τις μονάδες login και register, με τη διαφορά ότι ο χρήστης πρέπει να είναι ήδη συνδεδεμένος. Τα πεδία της φόρμας στοιχείων χρήστη αρχικά παίρνουν τις ήδη υπαρκτές τιμές, και αν ο χρήστης αλλάξει κάποιο πεδίο όπως το όνομα χρήστη ή τη διεύθυνση e-mail του ο ελεγκτής στέλνει ένα αίτημα PATCH στον διακομιστή.

```
var data = JSON.stringify({
  'username' : $scope.user.username,
  'email' : $scope.user.email
});

//Post JSON data to login api url
$http({method: 'PATCH', url: 'http://localhost:8000/api/users/detail/',
   data:data}).a).
success(function(data, status, headers, config) {
  if (status == 200) {
    // Εμφάνιση μηνύματος επιτυχίας
  }
  else {
    // Εμφάνιση μηνύματος λάθους
  }
}).
error(function(data, status, headers, config){
  // Εμφάνιση μηνύματος λάθους
});
```

6.5 Μονάδα Notes

Η μονάδα notes περιέχει τις λειτουργίες για τη διαχείριση των σημειώσεων του χρήστη. Σε αυτό το κομμάτι τις ιστοσελίδας ο χρήστης μπορεί να πλοηγηθεί στην λίστα με τις σημειώσεις, να τις τροποποιήσει ή διαγράψει και να δημιουργήσει καινούργια. Ο σχεδιασμός της διαφέρει από τις μονάδες-παιδιά της μονάδας security

κυρίως στην ύπαρξη παραπάνω ελεγκτών. Η μονάδα έχει δύο ελεγκτές, τον ελεγκτή NotesController που διαχειρίζεται τη λίστα με όλες τις σημειώσεις του χρήστη, και τον ελεγκτή ItemController που διαχειρίζεται την ενεργή σημείωση.

Ο ελεγκτής NotesController αρχικά στέλνει ένα αίτημα GET στον διακομιστή, και λαμβάνει όλες τις σημειώσεις σε μια λίστα.

```
$scope.notes = [];
$http({method: 'GET', url: 'http://localhost:8000/api/notes/'}).
success(function(data, status, headers, config) {
  return angular.forEach(data, function(item) {
    return $scope.notes.push(item);
  });
}).
error(function(data, status, headers, config) {
  console.log("list error");
  console.log(data);
});
```

Συμπεριλαμβάνει τις βοηθητικές συναρτήσεις open() και close() για την επιλογή ή κλείσιμο μιας σημείωσης, και τη συνάρτηση δημιουργίας νέας σημείωσης.

```
$scope.new = function(){
   var newdata = JSON.stringify({
      'title' : '',
      'body' : ''
    });
    $http({method: 'POST', url: 'http://localhost:8000/api/notes/', data:
   newdata}).
    success(function(data, status, headers, config) {
        //Push the new created note in the notes array
        $scope.notes.push(data);
        //Open the new item, which is the last item in the array.
        return $scope.open($scope.notes.slice(-1)[0]);
    }).
    error(function(data, status, headers, config) {
      console.log("error creating new note....");
      console.log(data);
    });
}
```

Στο έγγραφο HTML εμφανίζεται με χρήση της οδηγίας ng-repeat

```
<ul class="list-unstyled">
    <li ng-repeat="note in notes | filter:searchText "  ng-controller="
   ItemController" class="notelist" ng-click="open(note)">

        <div><h4>{{note.title}}</h4></div>
        <div><h6>{{note.body|limitTo:letterLimit}}</h6></div>

    </li>
</ul>
```

Ο ελεγκτής ItemController περιέχει τις συναρτήσεις ελέγχου της ενεργής ή ανοιχτής σημείωσης, save() και delete(). Είναι απαραίτητη η ύπαρξη του, καθώς στις προδιαγραφές του API οι μέθοδοι τροποποίησης και διαγραφής σημειώσεων έχουν δικά τους end

points, με αναγνωριστικούς υπερσύνδεσμους που περιέχουν τον αριθμό ταυτότητας (id) της κάθε σημείωσης.

Αρχικά δημιουργείται ο υπερσύνδεσμος για την ενεργή σημείωση

```
url = 'http://localhost:8000/api/notes/' + $scope.opened.id + '/';
```

Η συνάρτηση save() δημιουργεί ένα αντικείμενο με τα πεδία της ενεργής σημείωσης και στέλνει το αίτημα PATCH.

```
var data = JSON.stringify({
  'id' : $scope.opened.id,
  'title' : $scope.opened.title,
  'body' : $scope.opened.body
});

$http({method: 'PATCH', url: url, data:data}).
success(function(data, status, headers, config) {
  if (status == 200) {
    console.log("Done!");
  }
  // if (status == 404) {

  // }
  else {
    console.log("Not ok");
  }
}).
error(function(data, status, headers, config){
  //console.log("Status:" + status);
});
```

Αντίστοιχα, η συνάρτηση delete στέλνει το αίτημα DELETE για την ενεργή σημείωση.

```
$http({method: 'DELETE', url: url}).
success(function(data, status, headers, config) {
  if (status == 204) {
    $scope.remove($scope.opened);
    console.log("Done!");
  }
  else {
    console.log("Not ok");
  }
}).
error(function(data, status, headers, config){
  //console.log("Status:" + status);
});
```

6.6 Μονάδα Common

Η μονάδα common περιλαμβάνει τις κοινές μονάδες και λειτουργίες που χρησιμοποιούνται σε διαφορετικά τμήματα της εφαρμογής.

6.6.1 Οδηγία Toolbar

Για τη γραμμή πλοήγησης της εφαρμογής, γίνεται χρήση μιας οδηγίας (directive) που περιέχει το έγγραφο HTML για τη δημιουργία της μπάρας και τη λογική κατάλληλη για να αλλάξει τους υπερσύνδεσμους ανάλογα με το αν ο χρήστης είναι συνδεδεμένος.

Ενθυλακώνει τις απαραίτητες συναρτήσεις της μονάδας security για τη λειτουργία της. Αυτές είναι οι isAuthenticated, login και logout που χρησιμοποιούνται για να ελέγξει αν ο χρήστης είναι συνδεδεμένος και για τη λειτουργία σύνδεση ή αποσύνδεση του αντίστοιχα.

```
angular.module('common.toolbar', [])

.directive('loginToolbar', ['security', function(security) {
  var directive = {
    templateUrl: 'js/common/toolbar.html',
    restrict: 'E',
    replace: true,
    scope: true,
    link: function($scope, $element, $attrs, $controller) {
      $scope.isAuthenticated = security.isAuthenticated;
      $scope.login = security.showLogin;
      $scope.logout = security.logout;
      $scope.$watch(function() {
        return security.currentUser;
      }, function(currentUser) {
        $scope.currentUser = currentUser;
      });
    }
  };
  return directive;
}]);
```

Στο έγγραφο HTML της οδηγίας, με τη χρήση της συνάρτηση isAuthenticated καθορίζεται η εμφάνιση ή απόκρυψη των κατάλληλων υπερσυνδέσμων με τις ενσωματωμένες οδηγίες του AngularJS ng-show και ng-hide.

```
<li ng-show="isAuthenticated()">
  <a href='' ng-click="logout()">Logout</a>
</li>
<li ng-hide="isAusthenticated()">
  <a href='' ng-click="login()">Login</a>
</li>
```

Κεφάλαιο 7

Συμπεράσματα

Επιλέγοντας τα κατάλληλα εργαλεία και χρησιμοποιώντας τα αποτελεσματικά επιτυγχάνουμε τους στόχους που έχουμε θέσει. Βασικός μας στόχος ήταν η ανάπτυξη μιας διαδικτυακής υπηρεσίας προσβάσιμη από πολλές διαφορετικές πλατφόρμες. Για να τα το πετύχουμε αυτό διαχωρίσαμε και αναπτύξαμε ξεχωριστά τα δυο επίπεδα της υπηρεσίας, το νωτιαίο κομμάτι της υπηρεσίας και την εφαρμογή διεπαφής χρήστη.

Με το Django και το Django Rest Framework αναπτύξαμε το νωτιαίο μέρος της εφαρμογής, υπεύθυνο για την επεξεργασία, αποθήκευση και διαχείριση των δεδομένων της υπηρεσίας το όποιο μπορεί να χαρακτηριστεί και ως το κυρίως μέρος μιας εφαρμογής. Το Django μας παρείχε εργαλεία που διευκολύνουν την εργασία ενός προγραμματιστή και μας έδωσε χώρο στο να εργαστούμε πάνω στο σχεδιασμό και την βελτιστοποίηση της υπηρεσίας. Καίριο ρόλο στην ανάπτυξη διετέλεσε το Django REST Framework. Με την ενσωμάτωση του στην εφαρμογή και με την χρήση των κλάσεων που προσφέρει η ανταλλαγή δεδομένων μεταξύ της διεπαφής χρήστη και της υπηρεσίας πραγματοποιείται μέσω αρχείων JSON. Αυτό αποτελεί σημαντικό στοιχείο στην επίτευξη του στόχου που τέθηκε για πλήρη διαχωρισμό των παραπάνω επιπέδων.

Η συνεργασία των δύο επιπέδων συντελείται επιτυχημένα. Η επικοινωνία γίνεται μέσω μηνυμάτων HTTP μεταφέροντας πακέτα δεδομένων που μπορεί να συμπεριλαμβάνουν στοιχεία διασύνδεσης, πληροφορίες σχετιζόμενες με τους χρήστες, κλειδιά ασφάλειας, κύρια δεδομένα της υπηρεσίας και άλλες πληροφορίες. Σημαντικό είναι ότι η ανταλλαγή αυτών των δεδομένων πραγματοποιείται με ασφάλεια και ακεραιότητα.

Όπως προκύπτει μέσα από την διαδικασία της ανάπτυξης της εφαρμογής με την χρήση των παραπάνω εργαλείων και με τον απριόρι σχεδιασμό των κινήσεων που θα γίνουν επιτυγχάνονται και άλλοι σημαντικοί στόχοι της ανάπτυξης σύγχρονων διαδικτυακών υπηρεσιών γενικότερα.

Παράρτημα Α΄

Λεξικό Όρων

Application Programmable Interface Προγραμματιζόμενη Διεπαφή Εφαρμογής

Base Class Κλάση Βάσης

BSD Berkeley Software Distribution

Bytecode Μορφή Κώδικα byte

Cache Προσωρινή Μνήμη

Compiler Μεταγλωττιστής ή Μεταφραστής

CLI - Command Line Interface Περιβάλλον γραμμής εντολών

Create, Retreive, Update, Destroy(CRUD) Δημιουργία, ανάγνωση, ενημέρωση, διαγραφή

Database-driven

Dependency Injection Εισαγωγή εξαρτήσεων

Derived Class Παράγωγος Κλάση

Event Γεγονός

Framework Πλαίσιο ή πλαίσιο ανάπτυξης

Functional Programming Συναρτησιακός Προγραμματισμός

Generic Class Παραμετρική Κλάση

Imperative Programming Προστακτικός Προγραμματισμός

Inheritance Κληρονομικότητα

Machine-processable Μηχανικά επεξεργάσιμη

Middleware Ενδιάμεσο Λογισμικό

Mixins Σύνθεση

Model-View-Controller (MVC) Μοντέλο-Προβολή-Ελεγκτής

Module Μονάδα

NoSQL Database Μη-σχεσιακή βάση δεδομένων

Object-Relational Mapper (ORM) Αντικειμενο-Σχεσιακής Αντιστοίχησης

Package Πακέτο

Project Έργο

Python Module Μονάδα

Python Package Πακέτο

Rapid application development standards Πρότυπα ταχείας ανάπτυξης εφαρμογών

Regular Expressions Κανονικές Εκφράσεις

Redirect Ανακατεύθυνση

REST - Representational State Transfer Μετάθεση Αντιπροσωπευτικής Κατάστασης

Signal Σήμα

SOAP Simple Object Access Protocol Πρωτόκολλο πρόσβασης απλού αντικειμένου

Web Server Διακομιστής ιστού ή διαδικτύου

Unit Testing Έλεγχος Μονάδων

Βιβλιογραφία

[1] Daniel Greenfeld, Audrey Roy, *Two Scoops of Django: Best Practices For Django 1.6*, Two Scoops Press, 2nd edition, 2014.

[2] Ethan Cerami, *Web Services Essentials*, O'Reilly Media, 2002.

[3] Daniel A. Chappell, Tyler Jewell, *Java Web Services*, O'Reilly Media, 2002.

[4] Erik Wilde, Casare Pautasso, *REST: From Research to Practice*, Springer, 2011.

[5] Leonard Richardson, Sam Ruby and David Heinemeier Hansson, *RESTful Web Services*, O'Reilly Media, 2007.

[6] Jeff Forcier, Paul Bissex and Wesley J Chun, *Python Web Development with Django*, Addison-Wesley Professional, 2008.