ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη Γενικευμένων Λειτουργιών Τήρησης Ιστορικού για Κατανεμημένες Εφαρμογές με την Χρήση Java Spring

Χάρης Γ. Κουτσουρίδης
Α.Μ.: 3316

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων: Καθηγητής Χρήστος Ζαρολιάγκης
Συνεπιβλέπων: Δρ. Ιωάννης Χατζηγιαννάκης

Πάτρα
Ιανουάριος 2013

Περίληψη

Στην παρούσα διπλωματική εργασία μελετώνται μέθοδοι και τεχνικές ανάπτυξης λογισμικού. Παρουσιάζεται η έννοια της τεχνολογίας λογισμικού και η σημαντικότητα της μελέτης της για την αποδοτικότερη και ποιοτικότερη ανάπτυξη εφαρμογών. Η μελέτη επικεντρώνεται κυρίως σε συστήματα δικτυακών εφαρμογών με πολλούς χρήστες και τις τεχνολογίες που απαιτούνται για την υλοποίηση τους. Πιο συγκεκριμένα οι τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση είναι η πλατφόρμα Spring για την αντικειμενοστραφή γλώσσα Java.

Στα πλαίσια της διπλωματικής εργασίας μελετήθηκε το σύστημα Νέδα το οποίο είναι ένα σύστημα διαχείρισης ερευνητικών έργων που χρησιμοποιείται από το Ινστιτούτο Τεχνολογίας Υπολογιστών και Εκδόσεων ‘Διόφαντος’ της Πάτρας. Η συνεισφορά της διπλωματικής εργασίας στο σύστημα ήταν η υλοποίηση μερικών λειτουργιών που ήταν απαραίτητες για το σύστημα και κυρίως το σύστημα καταγραφής ιστορικού των ενεργειών των χρηστών. Άλλες λειτουργίες που προστέθηκαν στα πλαίσια της διπλωματικής εργασίας είναι ο μηχανισμός αποστολής ηλεκτρονικών μηνυμάτων, η προσθήκη ενός χρονοπρογραμματιστή στο σύστημα για την εκτέλεση εργασιών που είναι απαραίτητες σε συγκεκριμένο χρόνο και η βελτίωση μερικών υποσυστημάτων του συστήματος Νέδα. Ιδιαίτερη σημασία δόθηκε στην μεθοδολογία Aspect Oriented Programming, καθώς με την χρήση αυτής της τεχνικής διευκολύνθηκε σε μεγάλο βαθμό η υλοποίηση του υποσυστήματος της καταγραφής των ενεργειών των χρηστών.

Ευχαριστίες

Ευχαριστώ τον επιβλέποντα Καθηγητή της διπλωματικής εργασίας κ. Χρήστο Ζαρολιάγκη για την ευκαιρία που μου έδωσε να ασχοληθώ με την παρούσα διπλωματική εργασία καθώς και τον συνεπιβλέποντα Δρ. Ιωάννη Χατζηγιαννάκη για την καθοδήγηση του και την πολύτιμη βοήθεια που μου προσέφερε κατά τη διάρκεια της εκπόνησης της διπλωματικής εργασίας.

Πίνακας περιεχομένων

Πίνακας Σχεδίων

Κεφάλαιο 1 Εισαγωγή

1.1 Σημασία του θέματος

Η χρήση της πληροφορικής είναι όλο και πιο διαδεδομένη τα τελευταία χρόνια. Πολλοί οργανισμοί και εταιρίες για να λύσουν τα προβλήματα τους χρησιμοποιούν όλο και περισσότερο τους ηλεκτρονικούς υπολογιστές. Οι λύσεις που μπορεί να δώσει η πληροφορική σε οργανισμούς και εταιρίες μπορούν να κάνουν τις διαδικασίες τους πιο γρήγορες ή να προσφέρουν πιο ποιοτικές υπηρεσίες.

Σήμερα όλες οι εταιρίες ή οργανισμοί ανεξάρτητα από το προϊόν που παράγουν ή τις υπηρεσίες που προσφέρουν κάνουν χρήση των ηλεκτρονικών υπολογιστών. Αυτή η τάση οδήγησε σε μια νέα κατηγορία προϊόντων λογισμικού που στοχεύουν στην διευκόλυνση ενός οργανισμού για να αποπερατώσει τις διαδικασίες του. Ένα τέτοιο προϊόν μπορεί να είναι ένα σύστημα διαχείρισης των υπαλλήλων μιας επιχείρησης, ένα σύστημα διαχείρισης των έργων μιας επιχείρησης ή ένα σύστημα διαχείρισης πωλήσεων και πολλά άλλα. Τα συστήματα αυτά είναι συνήθως πολύπλοκα με πολλούς χρήστες και θα πρέπει να είναι αρκετά γενικά έτσι ώστε να μπορούν να προσαρμοστούν εύκολα από οργανισμό σε οργανισμό. Η απαίτηση να έχουν πολλούς χρήστες ταυτόχρονα λύνεται με τη βοήθεια του διαδικτύου. Σε αυτήν την περίπτωση οι χρήστες τις εφαρμογής αλληλεπιδρούν με την εφαρμογή μέσω ενός φυλλομετρητή και του διαδικτύου και ακολουθούν το μοντέλο του πελάτη εξυπηρετητή.

Η ανάπτυξη μιας τέτοιας εφαρμογής είναι μια πολύπλοκη διαδικασία. Κατά τη διάρκεια της ανάπτυξης της εφαρμογής προκύπτουν προκλήσεις, είτε σχεδιαστικές είτε τεχνολογικές που πρέπει να λυθούν. Υπάρχουν διάφορα μοντέλα ανάπτυξης λογισμικού που κάνουν πιο αποδοτική τη διαδικασία του προγραμματισμού και της συντήρησης του προϊόντος. Μια πολύπλοκη εφαρμογή θα πρέπει να ακολουθεί κάποιο από αυτά τα μοντέλα επειδή οι ανάγκες μιας εφαρμογής αλλάζουν όσο αυτή αναπτύσσεται, προκύπτουν προβλήματα και ο χρόνος προσαρμογής της εφαρμογής στις καινούριες απαιτήσεις θα πρέπει να είναι μικρός.

Στην παρούσα διπλωματική εργασία το σύστημα που μελετήθηκε είναι το σύστημα Νέδα. Η Νέδα είναι ένα σύστημα διαχείρισης επιστημονικών έργων. Το σύστημα αυτό έχει πολλούς χρήστες και πολλά έργα. Σε ένα τέτοιο σύστημα απαραίτητη λειτουργία είναι η δυνατότητα καταγραφής ιστορικού των ενεργειών των χρηστών. Τα δεδομένα

που οι χρήστες εισάγουν στην εφαρμογή και γενικά η αλληλεπίδραση χρήστη-εφαρμογής είναι βασικής σημασίας και η καταγραφή τους διευκολύνει τους διαχειριστές να μελετήσουν τις συμπεριφορές των χρηστών έτσι ώστε να εξάγουν χρήσιμα συμπεράσματα. Επίσης διευκολύνει τους διαχειριστές στο να εντοπίσουν παράτυπες ενέργειες των χρηστών .

1.2 Στόχος της διπλωματικής εργασίας

Ο στόχος της παρούσας διπλωματικής εργασίας είναι η μελέτη και εξοικείωση με τη διαδικασία ανάπτυξης λογισμικού. Ένας άλλος στόχος της διπλωματικής εργασίας είναι η χρήση ήδη υπαρχόντων τεχνολογιών λογισμικού για την γλώσσα προγραμματισμού Java και κυρίως της τεχνολογίας Spring.

Ο βασικότερος όμως στόχος της διπλωματικής εργασίας είναι η μελέτη του ήδη υπάρχοντος συστήματος Νέδα και η προσθήκη νέων χαρακτηριστικών στο σύστημα. Κατά τη φάση της προσθήκης νέων χαρακτηριστικών στην Νέδα μεγάλο κομμάτι αφιερώθηκε στην εξοικείωση με τον σχεδιασμό της Νέδας, στην κατανόηση των απαιτήσεων της Νέδας καθώς και στη χρήση εργαλείων διαχείρισης λογισμικού (Hudson, Trac) του Ινστιτούτου Τεχνολογίας Υπολογιστών και Εκδόσεων ‘Διόφαντος’ (Ι.Τ.Υ.Ε). Η κυρίως λειτουργία που προστέθηκε είναι η δυνατότητα του συστήματος να καταγράφει το ιστορικό των ενεργειών των χρηστών. Η καταγραφή αυτή θα πρέπει να υλοποιηθεί με έναν γενικευμένο τρόπο έτσι ώστε να μπορεί να καταγράφει, χωρίς αλλαγές στον κώδικα του συστήματος, και καινούριες λειτουργίες που προστίθενται. Επίσης πρέπει να δίνεται η δυνατότητα στους διαχειριστές να εντοπίζουν ενέργειες από το ιστορικό, έτσι ώστε να έχουν μια συνολική εικόνα του χρήστη.

1.3 Συνεισφορά της διπλωματικής εργασίας

Η συνεισφορά της παρούσας διπλωματικής εργασίας στο σύστημα Νέδα αφορούσε την προσθήκη μερικών καινούριων λειτουργιών και υποσυστημάτων καθώς επίσης και την

βελτιστοποίηση μερικών ήδη υπαρχόντων λειτουργιών. Τα καινούρια υποσυστήματα που προστέθηκαν είναι ο χρονοπρογραμματιστής εργασιών, ο μηχανισμός αποστολής ηλεκτρονικών μηνυμάτων με τη χρήση προτύπων. Επίσης έγινε μια βελτίωση στη διαδικασία ανάπτυξης της εφαρμογής και πιο συγκεκριμένα χρησιμοποιήθηκε το Apache Maven, το οποίο είναι ένα σύστημα αυτόματου κτισίματος λογισμικού. Οι βελτιώσεις που έγιναν αφορούσαν κυρίως τον τρόπο με τον οποίο διαχειρίζεται το σύστημα τις αργίες ενός έτους. Για όλα τα παραπάνω έπρεπε να μελετηθούν διάφορα προϊόντα και βιβλιοθήκες για τη γλώσσα Java.

Το κύριο υποσύστημα και βασική απαίτηση της διπλωματικής ήταν η προσθήκη ενός γενικευμένου μηχανισμού καταγραφής ιστορικού των ενεργειών των χρηστών. Ο γενικευμένος τρόπος αυτός καταγραφής ενεργειών υλοποιήθηκε έτσι ώστε να μην συνδέεται αποκλειστικά με το σύστημα Νέδα και να μπορεί να χρησιμοποιηθεί και σε άλλα συστήματα. Για την υλοποίησή του χρησιμοποιήθηκαν διάφορες τεχνικές προγραμματισμού και μεθοδολογίες και κυρίως η μεθοδολογία Aspect Oriented Programming (AOP) [4].

1.4 Δομή της Διπλωματικής Εργασίας

Σε αυτήν την ενότητα παρουσιάζεται η δομή της διπλωματικής εργασίας και τα κεφάλαια που την απαρτίζουν καθώς και μια συνοπτική περιγραφή του κάθε κεφαλαίου. Η διπλωματική εργασία έχει χωριστεί σε 6 κεφάλαια τα οποία και αναφέρονται παρακάτω.

Στο πρώτο κεφάλαιο γίνεται μια εισαγωγή όπου αναφέρεται ο στόχος της εργασίας, η σημαντικότητα και η χρησιμότητα του συστήματος Νέδα που χρησιμοποιείται από το Ι.Τ.Υ.Ε. Επίσης παρουσιάζετε η συνεισφορά της διπλωματικής εργασίας στο σύστημα Νέδα.

Στο δεύτερο κεφάλαιο αναλύεται η έννοια τις διαδικασίας ανάπτυξης εφαρμογών και παρουσιάζονται μερικές τεχνικές και εργαλεία που χρησιμοποιούνται για τη βελτιστοποίηση της. Μερικά από τα εργαλεία που παρουσιάζονται είναι τα ολοκληρωμένα περιβάλλοντα ανάπτυξης εφαρμογών, τα συστήματα διαχείρισης εκδόσεων, τα συστήματα συνεχούς ενσωμάτωσης και άλλα εργαλεία που ως στόχο έχουν την ταχύτερη και ποιοτικότερη διαδικασία ανάπτυξης λογισμικού.

Στο τρίτο κεφάλαιο περιγράφεται η αρχιτεκτονική και ο τρόπος λειτουργίας του

συστήματος Νέδα. Επίσης αναφέρονται οι λειτουργίες που προστέθηκαν στο σύστημα κατά τη διάρκεια της εκπόνησης της διπλωματικής εργασίας. Παρουσιάζονται μερικές από της βασικότερες οντότητες και λειτουργίες του συστήματος. Επίσης παραθέτονται οι αρχικές προδιαγραφές του, ο αρχικός σχεδιασμός του και η αρχιτεκτονική της υλοποίησης. Τέλος γίνεται μια αναφορά για την οργάνωση του Ι.Τ.Υ.Ε καθώς η κατανόηση της οργάνωσης αυτής είναι μείζονος σημασίας για την σωστή σχεδίαση του συστήματος.

Στο τέταρτο κεφάλαιο παρουσιάζονται οι τεχνολογίες που μελετήθηκαν και χρησιμοποιήθηκαν για την υλοποίηση των λειτουργιών που προστέθηκαν στο σύστημα. Οι τεχνολογίες αυτές περιλαμβάνουν μια πληθώρα έτοιμων βιβλιοθηκών, προϊόντων και μερικά περιβάλλοντα εργασιών. Αναφερόμαστε στις βασικές αρχές των σχεσιακών βάσεων δεδομένων και ποιο συγκεκριμένα στην MySQL, καθώς αυτή χρησιμοποιήθηκε ως το επίπεδο αποθήκευσης της εφαρμογής. Παρουσιάζουμε την τεχνολογία Object-Relational-Mapping (Συσχέτιση αντικειμένων σε σχέσεις) και τους λόγους που οδήγησαν στην χρήση της. Μεγάλο βάρος δίνεται στο περιβάλλων Spring και παρατίθενται μερικά από τα κυρίως χαρακτηριστικά του. Τέλος παρουσιάζονται και όλες η βιβλιοθήκες που χρησιμοποιήθηκαν για την υλοποίηση των επιμέρους υποσυστημάτων στα πλαίσια της διπλωματικής εργασίας.

Ο τρόπος με τον οποίο υλοποιήθηκαν οι καινούργιες λειτουργίες παρουσιάζεται στο πέμπτο κεφάλαιο όπου υπάρχουν και αρκετά μέρη του κώδικα που γράφτηκε. Αναφερόμαστε εκτενώς στο σύστημα αποστολής ηλεκτρονικών μηνυμάτων, στο σύστημα του χρονοπρογραμματιστή και κυρίως στο σύστημα της καταγραφής του ιστορικού των ενεργειών των χρηστών. Επίσης παρουσιάζονται και τα μικρότερης σημασίας υποσυστήματα που υλοποιήθηκαν.

Τέλος στο έκτο και τελευταίο κεφάλαιο παρουσιάζονται τα συμπεράσματα της διπλωματικής εργασίας και παρατίθενται μερικές βελτιώσεις και καινούργιες λειτουργίες για το σύστημα που θα μπορούσαν να είναι χρήσιμες.

Κεφάλαιο 2 Ανάπτυξη εφαρμογών

Σε αυτό το κεφάλαιο παρουσιάζονται μερικές μεθοδολογίες ανάπτυξης εφαρμογών, τι είναι η τεχνολογία λογισμικού [15] και ποια τα πλεονεκτήματα της μελέτης της και τέλος, παρουσιάζουμε μερικά εργαλεία που διευκολύνουν τη διαδικασία ανάπτυξης λογισμικού.

2.1 Γενικά

Κατά τη διάρκεια της ανάπτυξης μιας εφαρμογής εφαρμόζουμε μια μεγάλη ποικιλία μεθόδων, εργαλείων, διαδικασιών και υποδειγμάτων. Ως μέθοδο ή τεχνική ονομάζουμε μια τυπική διαδικασία για την παραγωγή κάποιου αποτελέσματος. Εργαλεία μπορούν να θεωρηθούν διάφορα αυτοματοποιημένα συστήματα που έχουν σαν στόχο την επίτευξη ενός αποτελέσματος με καλύτερο τρόπο. Ένα εργαλείο που χρησιμοποιούμε κατά τη διάρκεια ανάπτυξης λογισμικού μπορεί να έχει σαν στόχο το να γίνουμε πιο παραγωγικοί, κάποιο άλλο εργαλείο μπορεί να έχει σαν στόχο την εξασφάλιση της ορθότητας του συστήματός μας. Μια διαδικασία είναι ένας συνδυασμός μεθόδων και εργαλείων με κατάλληλο τρόπο έτσι ώστε να παράγουμε το προϊόν λογισμικού που αναπτύσσουμε.

Η ανάπτυξη ενός προϊόντος λογισμικού περιλαμβάνει τις παρακάτω δραστηριότητες:

- Ανάλυση απαιτήσεων
- Σχεδίαση συστήματος
- Σχεδίαση προγραμμάτων
- Υλοποίηση προγραμμάτων
- Έλεγχος των επιμέρους προγραμμάτων
- Έλεγχος ολοκλήρωσης
- Έλεγχος συστήματος
- Παράδοση συστήματος

- Συντήρηση

Στην πραγματικότητα τα παραπάνω βήματα δεν εκτελούνται μόνο μια φορά αλλά επαναλαμβάνονται. Η επανάληψη των παραπάνω δραστηριοτήτων είναι χρονοβόρα και για αυτόν τον λόγο είναι απαραίτητη η μελέτη και κατανόηση της τεχνολογίας λογισμικού. Κάθε βήμα από τα παραπάνω θεωρείται σαν μια διεργασία ανάπτυξης λογισμικού.

Στη βιβλιογραφία έχουν προταθεί διάφορα μοντέλα διεργασιών ανάπτυξης λογισμικού. Οι λόγοι για τους οποίους χρειάζεται η μοντελοποίηση των διεργασιών είναι:

- Η δημιουργία ενός μοντέλου διεργασίας βοηθά στο να εντοπιστούν ασυνέπειες, πλεονασμοί και παραλήψεις που υπάρχουν στη διεργασία.
- Η μοντελοποίηση έχει σαν στόχο την ανάπτυξη λογισμικού υψηλής ποιότητας, τον γρήγορο εντοπισμό ελαττωμάτων και την συμμόρφωση στους χρονικούς περιορισμούς του έργου.
- Η μοντελοποίηση των διεργασιών επιβάλει καλύτερο και αποδοτικότερο διαμοιρασμό πόρων.

Μερικά από τα πιο γνωστά μοντέλα ανάπτυξης λογισμικού είναι:

- **Γραμμικό μοντέλο (η μοντέλο καταρράκτη):** Τα στάδια αυτού του μοντέλου είναι μια γραμμική ακολουθία. Για να ξεκινήσει κάποιο στάδιο θα πρέπει να έχει ολοκληρωθεί το προηγούμενο του.
- **Μοντέλο V:** Αυτό το μοντέλο είναι μια παραλλαγή του γραμμικού μοντέλου και έχει σαν στόχο να αναδείξει τον τρόπο με τον οποίο οι δραστηριότητες σχετίζονται με την ανάλυση και την σχεδίαση τους.
- **Το σπειροειδές μοντέλο:** Αυτό το μοντέλο είναι ένα μοντέλο που βασίζεται στη συνεχή επανάληψη τεσσάρων φάσεων μέχρι να πετύχουμε το επιθυμητό αποτέλεσμα. Οι φάσεις είναι: καθορισμός στόχων και εναλλακτικών λύσεων, αξιολόγηση εναλλακτικών λύσεων και κινδύνων, ανάπτυξη και δοκιμές και κατάστρωση πλάνου. Το πλάνο στην πρώτη επανάληψη είναι πλάνο ανάπτυξης, στη δεύτερη επανάληψη είναι πλάνο ενοποίησης και πλάνο δοκιμών και τέλος είναι πλάνο υλοποίησης.

Για τις παραπάνω διεργασίες ανάπτυξης λογισμικού υπάρχουν διάφορα προϊόντα λογισμικού για την υποστήριξη τους. Στις παρακάτω υποενότητες παρουσιάζουμε τα προϊόντα που χρησιμοποιήθηκαν κατά τη διάρκεια της εκπόνησης της παρούσας διπλωματικής εργασίας για την μελέτη και προσθήκη χαρακτηριστικών στο σύστημα Νέδα. Όλα τα παρακάτω προϊόντα έχουν σαν στόχο την βελτιστοποίηση, από άποψη

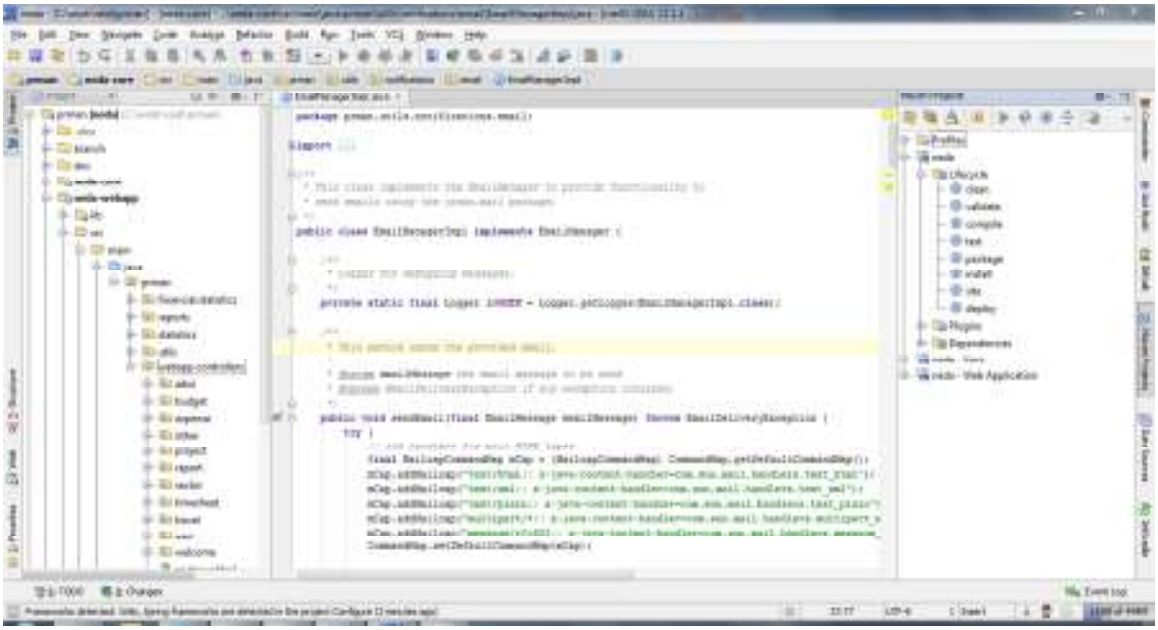
ποιότητας και χρόνου, της διαδικασίας ανάπτυξης λογισμικού.

2.2 Ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικού

Για τη συγγραφή του κώδικα χρησιμοποιήθηκε το περιβάλλον IntelliJ.Το IntelliJ είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού (Integrated Development Environment, IDE). Παρέχεται σε δύο εκδόσεις, η μία εμπορική και μία με λιγότερα χαρακτηριστικά που διατίθεται δωρεάν. Το IntelliJ υποστηρίζει μια πληθώρα γλωσσών μεταξύ των οποίων και: Java, JavaScript, HTML/XHTML/CSS, XML/XSL, ActionScript/MXML, Python, Ruby/JRuby,Groovy, SQL, PHP. Επίσης υποστηρίζει και παρέχει βοηθήματα για πολλές τεχνολογίες όπως: JSP, JSF, EJB, Ajax, Spring, Hibernate/JPA, Web Services, Java ME MIDP/CLDC, OSGi, FreeMarker, Velocity.

Μερικά από τα πιο αξιοσημείωτα χαρακτηριστικά του είναι τα διαγράμματα κλάσεων σε μια μορφή που μοιάζει με UML, οπτική μοντελοποίηση οντοτήτων hibernate, οπτική μοντελοποίηση εξαρτήσεων βιβλιοθηκών, ανάλυση ροής μεθόδων.

Το IntelliJ προσφέρει εξαιρετική υποστήριξη για το Maven κάνοντας τη διαδικασία ανάπτυξης και συντήρησης εφαρμογών πιο αυτοματοποιημένη και πιο γρήγορη. Άλλα σημαντικά πλεονεκτήματα του IntelliJ είναι η υποστήριξη JUnit για έλεγχο ρουτινών, ενσωματωμένη υποστήριξη για πολλά συστήματα διαχείρισης εκδόσεων (VCS). Για την υλοποίηση της διπλωματικής χρησιμοποιήθηκε το Subversion (svn). Ένας από τους κύριους λόγους που το IntelliJ θεωρείται ως το καλύτερο IDE για Java είναι οι ευκολίες που προσφέρει στον προγραμματιστή για να κάνει αυτόματες αλλαγές στον κώδικα (refactoring). Μερικές από της πιο συχνές αλλαγές που χρησιμοποιούνται είναι: μετονομασία κλάσης, αλλαγή των ορισμάτων μιας μεθόδου και άλλα πολλά. Επίσης όπως και τα περισσότερα άλλα IDE προσφέρει αυτόματη δημιουργία κώδικα από interfaces και αυτόματη δημιουργία σχολίων σύμφωνα με το JavaDocs πρότυπο.



Εικόνα 2.1 Στιγμιότυπο του IntelliJ

2.2.1 Άλλα ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικού

Στην υλοποίηση της διπλωματικής χρησιμοποιήθηκε το IntelliJ. Μερικά άλλα γνωστά περιβάλλοντα είναι:

- **NetBeans:** Το NetBeans είναι ένα IDE ανοιχτού κώδικα και υποστηρίζει ανάπτυξη για όλα τα είδη java εφαρμογών όπως: Java SE, Java ME, web, EJB και άλλα.
- **Eclipse:** Το Eclipse εκτός από java υποστηρίζει και πολλές άλλες γλώσσες και είναι κυρίως διαδεδομένο λόγο του πολύ καλού συστήματος που διαθέτει για πρόσθετα εργαλεία (plugins).

2.3 Συστήματα διαχείρισης εκδόσεων

Τα συστήματα διαχείρισης εκδόσεων είναι προϊόντα λογισμικού που στόχο έχουν την

διαχείριση των αλλαγών σε έγγραφα, πηγαίο κώδικα εφαρμογών και άλλων συλλογών από πληροφορίες. Στη διαδικασία ανάπτυξης εφαρμογών το είδος των αρχείων που αποθηκεύονται είναι κυρίως πηγαίος κώδικας. Για το σύστημα Νέδα χρησιμοποιείται το SVN.

Το svn (Apache Subversion) είναι ένα σύστημα διαχείρισης εκδόσεων ανοιχτού κώδικα. Το svn όπως και όλα τα συστήματα διαχείρισης εκδόσεων χρησιμοποιούνται για την αποθήκευση πηγαίου κώδικα και άλλον αρχείων ανάλογα με την εφαρμογή. Η χρήση ενός συστήματος διαχείρισης εκδόσεων είναι απαραίτητη σε ένα προϊόν καθώς πολύ συχνά πολλοί χρήστες δουλεύουν πάνω στο προϊόν και σε πολλές περιπτώσεις πάνω σε διαφορετικές εκδόσεις του προϊόντος. Το svn είναι μια αυτόνομη εφαρμογή που εγκαθίσταται σε έναν εξυπηρετητή στον οποίον συνδέονται οι πελάτες (κυρίως προγραμματιστές) για να υποβάλλουν τις αλλαγές τους.

Το svn χρησιμοποιεί branches και tags για την ευκολότερη διαχείριση των εκδόσεων ενός προϊόντος. Ένα branch είναι ουσιαστικά μια διαφορετική και ανεξάρτητη γραμμή του προϊόντος. Από την άλλη ένα tag είναι ένα στιγμιότυπο σε μια συγκεκριμένη χρονική στιγμή ενός branch. Η κυρίως γραμμή του κώδικα ονομάζεται trunk και είναι αυτή που περιέχει της πιο πρόσφατες αλλαγές. Το svn προσφέρει εργαλεία για την πιο εύκολη ενσωμάτωση των αλλαγών που γίνονται στο trunk σε κάποιο branch.

Το svn είναι το πιο διαδεδομένο σύστημα διαχείρισης εκδόσεων και χρησιμοποιείται για πολλά προϊόντα ανοιχτού κώδικα αλλά και σε πολλές εταιρίες.

Άλλα προϊόντα διαχείρισης εκδόσεων είναι το cvs το οποίο παρουσιάζει πολλά προβλήματα τα οποία έχει λύση το svn. Επίσης πολύ διαδεδομένο είναι το Git το οποίο είναι ένα κατανεμημένο σύστημα διαχείρισης εκδόσεων και αρχικά είχε υλοποιηθεί για την ανάπτυξη του πυρήνα του linux.

Ο πηγαίος κώδικας τους συστήματος Νέδα βρίσκεται στο SVN που χρησιμοποιείται από το I.T.Y.E.(http://ru1.cti.gr/svn/prman).

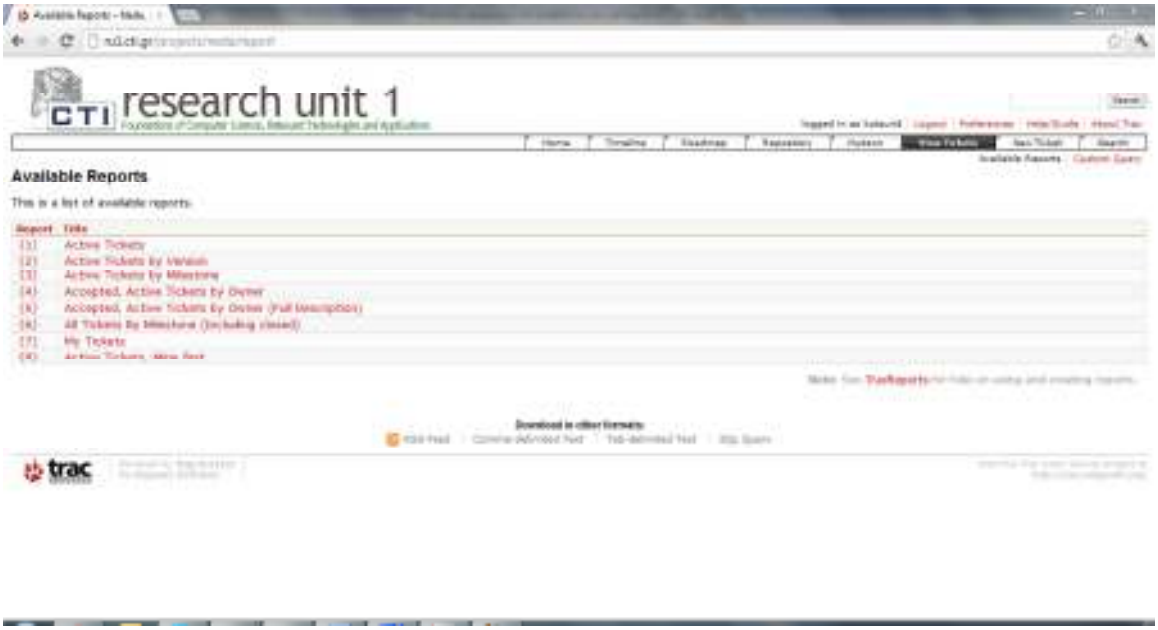
2.4 Συστήματα διαχείρισης έργων λογισμικού

Για της ανάγκες ανάπτυξης του συστήματος Νέδα χρειάζεται η χρήση ενός προϊόντος διαχείρισης έργων λογισμικού. Τα προϊόντα αυτά δίνουν την δυνατότητα στους

προγραμματιστές να καταγράφουν νέες λειτουργίες που πρέπει να προστεθούν, να καταγράφουν προβλήματα που πρέπει να διορθωθούν και να τα αναθέτουν σε διάφορα άτομα έτσι ώστε να μπορούμε με εύκολο και γρήγορο τρόπο να βλέπουμε την πρόοδο του έργου. Επίσης πολλά προϊόντα διαχείρισης λογισμικού προσφέρουν τη δυνατότητα να εισάγουμε και την τεκμηρίωση του προϊόντος και διάφορα εγχειρίδια χρήσης. Για το σύστημα Νέδα χρησιμοποιείται το Trac.

Το Trac είναι ένα ανοιχτού κώδικα σύστημα διαχείρισης προϊόντων (project management) και σύστημα παρακολούθησης προβλημάτων (issue tracking system).

Είναι γραμμένο σε python και προσφέρει μια web διεπαφή. Το Trac υποστηρίζει μια συσχέτιση μεταξύ προβλημάτων, διαχείριση εκδόσεων και περιεχομένων wiki. Ένα σύστημα διαχείρισης προϊόντων είναι απαραίτητο κατά τη διαδικασία ανάπτυξης μιας εφαρμογής με πολλούς προγραμματιστές και πολλές εκδόσεις.



Εικόνα 2.2 Στιγμιότυπο του Trac

2.5 Συστήματα συνεχούς ενσωμάτωσης

Ως σύστημα συνεχούς ενσωμάτωσης (Continues Integration, CI) ορίζουμε μια κατηγορία λογισμικού που σκοπό έχει την διευκόλυνση της ολοκλήρωσης και της

ενσωμάτωσης του προϊόντος που υλοποιούμε. Ως ολοκλήρωση και ενσωμάτωση εννοούμε τη διαδικασία με την οποία τα διάφορα υποσυστήματα μεταγλωττίζονται και ενώνονται σε ένα προϊόν που μπορεί να παραδοθεί. Στην περίπτωση του συστήματος Νέδα το τελικό προϊόν είναι ένα αρχείο war το οποίο είναι έτοιμο προς εγκατάσταση σε έναν application server. Η διαδικασία αυτή περιέχει πολλά στάδια και πρέπει να εκτελείται σε τακτά χρονικά διαστήματα. Ένα από τα κυριότερα στάδια είναι η εκτέλεση των ρουτινών έλεγχου για να βεβαιωθούμε ότι το σύστημα μας είναι λειτουργικό. Για τις ανάγκες της Νέδας χρησιμοποιείται το Hudson.

Το Hudson [12] είναι ένα εργαλείο συνεχούς ενσωμάτωσης (continues integration). Το Hudson είναι υπεύθυνο για την παρακολούθηση και την εκτέλεση επαναλαμβανόμενων εργασιών όπως το χτίσιμο ενός project. Η κύρια λειτουργία του Hudson μπορεί να εστιαστεί στα παρακάτω δύο σημεία

- **Συνεχές χτίσιμο και δοκιμή projects**. Με έναν εύκολο στην χρήση μηχανισμό το Hudson παρέχει ένα σύστημα για συνεχή ενσωμάτωση κώδικα διευκολύνοντας τον προγραμματιστή να ενσωματώσει τις αλλαγές του στο project. Επίσης διευκολύνει τους άλλους χρήστες να πάρουν την τρέχουσα έκδοση του προϊόντος. Τα επαναλαμβανόμενα χτισίματα του project αυξάνουν την αποδοτικότητα.
- **Παρακολούθηση των εργασιών που τρέχουν**. Ο Hudson προσφέρει έναν εύκολο στην χρήση μηχανισμό για να ειδοποιεί τους χρήστες για τυχόν προβλήματα στο project ενημερώνοντας τους κατάλληλους υπευθύνους.

Ένα σύστημα συνεχούς ενσωμάτωσης διευκολύνει τους προγραμματιστές και τους διαχειριστές να παρακολουθούν την εξέλιξη του κώδικα και να εντοπίζουν τυχόν σφάλματα γρήγορα και αποδοτικά [17].

Το Hudson προσφέρει μεταξύ άλλων ολοκλήρωση με apache-ant και apache maven.

Μερικά από τα κύρια χαρακτηριστικά του Hudson είναι:

- Εύκολη ρύθμιση
- Χρηστικό προς τον χρήστη γραφικό περιβάλλον
- Υποστήριξη plugins
- Εκτέλεση δοκιμών στον κώδικα (test cases) και αναφορά για τυχόν αποτυχίες
- Κρατάει πλήρες ιστορικό για το project κάνοντας ποιο εύκολη την παρακολούθηση του.

Ο Hudson υποστηρίζει μια πληθώρα συστημάτων παρακολούθησης εκδόσεων (revision

control) μεταξύ των οποίον και τα : CVS, Subversion, Git. Η τυπική λειτουργία του Hudson περιλαμβάνει τα ακόλουθα βήματα

1. Ελέγχει τις αλλαγές στο σύστημα εκδόσεων ανά τακτά διαστήματα και αν υπάρχουν τότε εκτελεί και τα επόμενα βήματα.
2. Κτίζει το project.
3. Τρέχει τις ρουτίνες έλεγχου.
4. Παράγει διάφορες αναφορές και μετρικές για το project όπως: πιθανά προβλήματα, βελτιστοποιήσεις που μπορούν να γίνουν στον κώδικα.

Συνήθως η διαδικασία της εκτέλεσης των ρουτινών ελέγχου είναι από τις πιο σημαντικές καθώς αν κάποια από τις ρουτίνες αποτύχει, τότε όλο το έργο θεωρείται ως αποτυχημένο και ενημερώνονται τα κατάλληλα άτομα που συνήθως είναι ο χρήστης ή οι χρήστες που έκαναν τις αλλαγές στον κώδικα.

Παρόμοια συστήματα με το Hudson είναι και τα CruiseControl και DamageControl.

2.6 Συστήματα αυτόματου κτισίματος προϊόντος

Τα συστήματα αυτόματου κτισίματος προϊόντος (Build Automation) είναι μια κατηγορία προϊόντων λογισμικού που σαν στόχο έχουν τη διευκόλυνση στις παρακάτω λειτουργίες της διαδικασίας ανάπτυξης μιας εφαρμογής.

- Μεταγλώττιση του πηγαίου κώδικα
- Πακετάρισμα του προϊόντος
- Εκτέλεση ρουτινών έλεγχου
- Εγκατάσταση του προϊόντος
- Παραγωγή τεκμηρίωσης

Τα συστήματα αυτόματου κτισίματος χρησιμοποιούνται από τα συστήματα συνεχούς ενσωμάτωσης. Το σύστημα που χρησιμοποιείται για την Νέδα είναι το Maven.

Το Maven είναι ένα εργαλείο για τη διαχείριση προϊόντων λογισμικού. Είναι ανοιχτού κώδικα και διατίθεται από το Apache Software Foundation. Το Maven μπορεί να διαχειριστεί τη δημιουργία του προϊόντος, την ανάλυσή του και επίσης την παραγωγή τεκμηρίωσης με έναν κεντρικοποιημένο τρόπο. Παρουσιάζει πολλές ομοιότητες με το

γνωστό εργαλείο Ant αλλά έχει θεμελιώδεις διαφορές στις αρχές λειτουργίας του. Χρησιμοποιείται κυρίως για Java προϊόντα άλλα μπορεί να χρησιμοποιηθεί και για προϊόντα που είναι γραμμένα σε C#, Ruby, Scala και άλλες γλώσσες.

Οι βασικές ανάγκες που προσπαθεί να καλύψει το Maven είναι ότι χρειαζόμαστε έναν συγκεκριμένο και ομοιόμορφο τρόπο για να κτίζουμε τα προϊόντα μας, έναν ξεκάθαρο ορισμό για το από τι αποτελείται το προϊόν μας καθώς και έναν εύκολο τρόπο για να παράγουμε βιβλιοθήκες Java έτσι ώστε να τις διαμοιραζόμαστε μεταξύ των προϊόντων μας.

Ο κύριος στόχος του Maven είναι να δίνει στον προγραμματιστή τη δυνατότητα να ολοκληρώνει οποιαδήποτε φάση της διαδικασίας της ανάπτυξης λογισμικού στον μικρότερο δυνατό χρόνο. Για να επιτευχτεί αυτό το Maven προσφέρει τα παρακάτω χαρακτηριστικά:

- Κάνει την διαδικασία κατασκευής του προϊόντος εύκολη
- Παρέχει ένα ενιαίο σύστημα κατασκευής προϊόντων
- Παρέχει πληροφορίες για την ποιότητα του προϊόντος
- Παρέχει οδηγίες για πιο αποδοτική ανάπτυξη εφαρμογών
- Παρέχει διαφανής ενσωμάτωση νέων χαρακτηριστικών

Παρόλο που η χρήση του Maven απαιτεί γνώση για τον μηχανισμό με τον οποίο δουλεύει, ο προγραμματιστής δεν χρειάζεται να γνωρίζει πολλές λεπτομέρειες και μπορεί να εκτελέσει τα πιο βασικά βήματα με τις ελάχιστες γνώσεις.

Για την κατασκευή ενός προϊόντος χρησιμοποιώντας Maven το μόνο που απαιτείται είναι η περιγραφή του προϊόντος μέσα από ένα αρχείο «μοντελοποίησης προϊόντος» και ένα σετ από πρόσθετα που προσφέρονται από το Maven και διαμοιράζονται μεταξύ όλων τον project, παρέχοντας έτσι έναν ομοιόμορφο μηχανισμό κατασκευής προϊόντων. Το Maven διευκολύνει και γλιτώνει πολύ χρόνο στους προγραμματιστές κυρίως όταν αυτοί ασχολούνται με πολλά προϊόντα ταυτόχρονα.

Το Maven παρέχει μια πληθώρα από χρήσιμες πληροφορίες για το προϊόν που μπορούν να πηγάζουν κατευθείαν από τον πηγαίο κώδικα. Για παράδειγμα μπορεί να παρέχει αναφορές για τις εξαρτήσεις βιβλιοθηκών, λίστες που περιέχουν τους προγραμματιστές του προϊόντος έτσι ώστε να χρησιμοποιηθούν από άλλα εργαλεία όπως το Hudson ή κάποια συστήματα παρακολούθησης προβλημάτων, αναφορές για αποτυχημένες ρουτίνες έλεγχου και ποσοστά πληρότητας των ρουτινών.

Προσφέρεται ένας μηχανισμός για τη δημιουργία προσθέτων έτσι ώστε να μπορεί να

εμπλουτιστεί το σετ με τις αναφορές που προσφέρονται από το Maven.

Επίσης το Maven στοχεύει στην υιοθέτηση από τους προγραμματιστές μερικών τεχνικών ανάπτυξης εφαρμογών που θεωρούνται αποδοτικές. Για παράδειγμα η εκτέλεση και η αναφορά σφαλμάτων των ρουτινών έλεγχου θεωρούνται ως μέρος της διαδικασίας χτισίματος του προϊόντος. Επίσης βάζει περιορισμούς ως προς την θέση των αρχείων με πιο σημαντική τη διαχώριση του κώδικα με τις ρουτίνες έλεγχου και του κώδικα του προϊόντος.

Ένα άλλο πολύ σημαντικό χαρακτηριστικό του Maven είναι ότι στοχεύει στην διευκόλυνση στην διαδικασία ανάπτυξης λογισμικού όπως διαχείριση εκδόσεων και παρακολούθηση προβλημάτων.

Όπως αναφέρθηκε παραπάνω ένα προϊόν περιγράφεται από ένα αρχείο POM το οποίο είναι ένα XML αρχείο και περιέχει διάφορες πληροφορίες για το έργο μας. Ένα μέρος ενός τέτοιου αρχείου φαίνεται παρακάτω:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>neda</groupId>
        <version>1.0.0-SNAPSHOT</version>
        <artifactId>neda</artifactId>
        <relativePath>../pom.xml</relativePath>
    </parent>
    <artifactId>neda-webapp</artifactId>
    <packaging>war</packaging>
    <name>neda - Web Application</name>
    <description>Neda Project Management - Web
Application</description>
```

Από το παραπάνω μπορούμε να διακρίνουμε το όνομα του προϊόντος καθώς επίσης και το είδος του. Στο παράδειγμα μας είναι war που σημαίνει ότι το Maven θα αναλάβει αυτόματα κατά τη διαδικασία χτισίματος να παράξει το .war αρχείο μαζί με όλες τις εξαρτήσεις και την κατάλληλη δομή των φακέλων μέσα στο .war έτσι όπως ορίζεται από το πρότυπο.

Ένα άλλο μέρος του αρχείου pom είναι οι ορισμοί των εξαρτήσεων σε βιβλιοθήκες του προϊόντος μας, για παράδειγμα:

```
    <dependencies>
        <dependency>
```

```
            <groupId>hudson.plugins.sloccount</groupId>
            <artifactId>sloccount</artifactId>
            <version>1.4</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>${slf4j.version}</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.easymock</groupId>
            <artifactId>easymock</artifactId>
            <version>${easymock.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

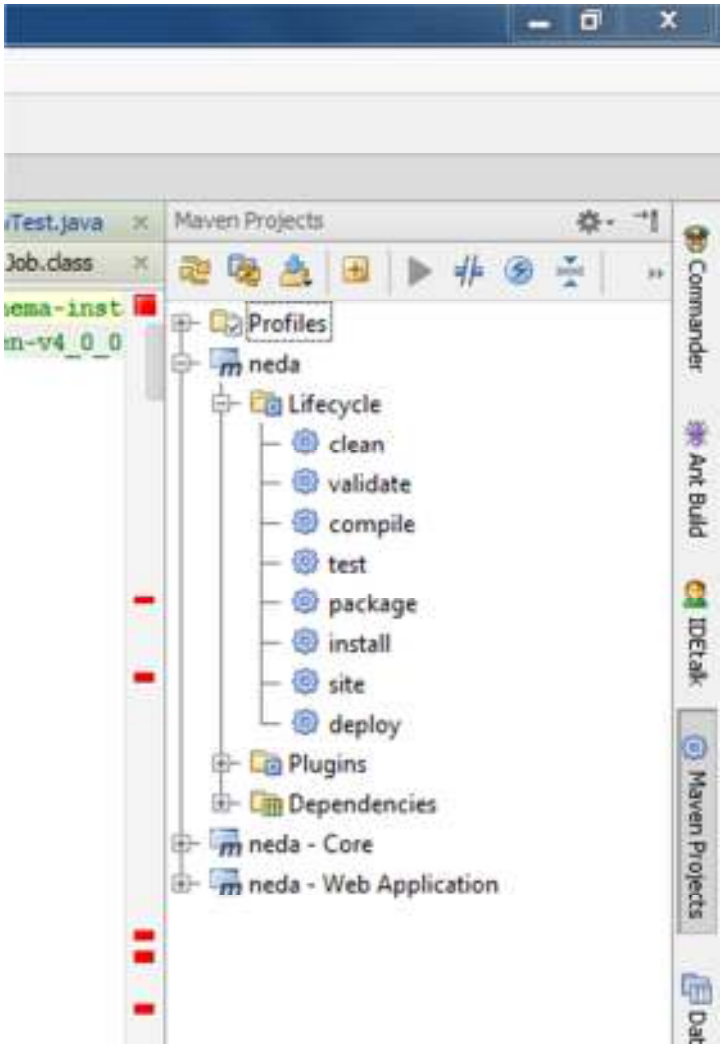
Από τα παραπάνω διακρίνουμε ότι το <scope> της κάθε εξάρτησης μπορεί να είναι διαφορετικό για παράδειγμα η βιβλιοθήκη easymock χρησιμοποιείται μόνο στις κλάσεις με τις ρουτίνες έλεγχου και δεν είναι ανάγκη να συμπεριλαμβάνεται στο .war που θα παραχθεί. Οι βιβλιοθήκες αυτές μεταφορτώνονται από το διαδίκτυο αυτόματα και δεν χρειάζεται ο χρήστης να τις συμπεριλαμβάνει στον πηγαίο κώδικα του, κάτι που αυξάνει κατά πολύ την αποδοτικότητα του προγραμματιστή και καθιστά πιο εύκολη την αλλαγή εκδόσεων σε βιβλιοθήκες. Οι βιβλιοθήκες υπάρχουν σε έναν κεντρικό εξυπηρετητή (maven repository) και επίσης μπορούμε να ορίσουμε και άλλους τέτοιους εξυπηρετητές για να αναζητήσουμε βιβλιοθήκες που δεν υπάρχουν στον κεντρικό. Επίσης ο καθένας μπορεί να στήσει τον δικό του εξυπηρετητή βιβλιοθηκών όπου θα μπορεί να τοποθετεί δικές του βιβλιοθήκες. Αυτό το χαρακτηριστικό της αρχιτεκτονικής του maven διευκολύνει κατά πολύ οργανισμούς και εταιρίες για την διαχείριση και διάθεση των βιβλιοθηκών που παράγουν.

Το Maven είναι ένα εργαλείο γραμμής εντολών και μερικές από τις πιο χρήσιμες λειτουργίες και εντολές παραθέτονται παρακάτω:

- **mvn clean:** Διαγραφή των μεταγλωττισμένων αρχείων

- **mvn compile:** Μεταγλώττιση των αρχείων
- **mvn test:** Εκτέλεση των ρουτινών έλεγχου
- **mvn package:** Πακετάρισμα του προϊόντος ανάλογα με το είδος του

Υποστήριξη για maven έχουν σχεδόν όλα τα γραφικά περιβάλλοντα ανάπτυξης εφαρμογών. Στην παρακάτω εικόνα βλέπουμε πως φαίνεται η δομή του project καθώς επίσης και οι βασικές φάσεις του κύκλου κατασκευής ενός προϊόντος στο IntelliJ.



Εικόνα 2.3 IntelliJ και Maven modules

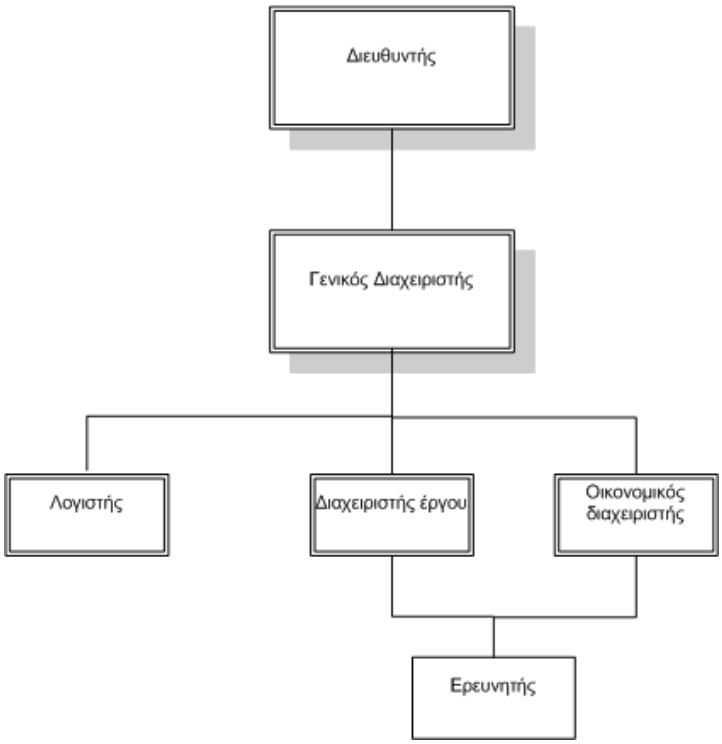
Η χρήση ενός συστήματος όπως το Maven είναι απαραίτητη και διευκολύνει κατά πολύ τους προγραμματιστές όταν στο προϊόν ενσωματώνονται συνέχεια καινούρια χαρακτηρίστηκα και αλλάζει εκδόσεις, κάτι που συμβαίνει πάντα σε όλα τα προϊόντα στην πραγματικότητα.

Κεφάλαιο 3 Περιγραφή Συστήματος Νέδα

Η Νέδα είναι ένα σύστημα διαχείρισης επιστημονικών έργων. Υπάρχουν πολλές εφαρμογές για διαχείριση έργων με πιο γνωστή την εμπορική εφαρμογή Microsoft Project. Η εφαρμογή αυτή επιτρέπει τον χρονοπρογραμματισμό του έργου και κατανομή πόρων στο έργο. Αυτά τα συστήματα είναι γενικού σκοπού και δεν μπορούν να καλύψουν όλες της ανάγκες ενός οργανισμού γιατί οι διαδικασίες ενός οργανισμού αν και έχουν πολλές ομοιότητες δεν μπορούν να καλυφτούν από ένα μόνο μοντέλο. Η ανάγκη διαχείρισης αποκλειστικά επιστημονικών έργων στο Ινστιτούτο Τεχνολογίας Υπολογιστών και Εκδόσεων 'Διόφαντος' οδήγησε στην υλοποίηση του συστήματος Νέδα. Στα πλαίσια της παρούσας διπλωματικής εργασίας μελετήθηκε το σύστημα Νέδα και κατά κύριο λόγο η προσθήκη νέων χαρακτηριστικών σε αυτό. Για να γίνει αυτό ήταν απαραίτητη η μελέτη και η κατανόηση των απαιτήσεων της Νέδας. Το σύστημα Νέδα έχει σαν στόχο την εξυπηρέτηση των υπαλλήλων του Ινστιτούτου Τεχνολογίας Υπολογιστών και Εκδόσεων 'Διόφαντος'.

3.1 Οργάνωση του Ι.Τ.Υ.Ε.

Η κατανόηση της οργάνωσης και της δομής του Ι.Τ.Υ.Ε. είναι μείζονος σημασίας για την σωστή κατανόηση των απαιτήσεων της Νέδας και των λειτουργικών αναγκών της. Όπως σε κάθε οργανισμό έτσι και στο Ι.Τ.Υ.Ε. υπάρχει μια υποδομή των εργαζομένων του. Στο χαμηλότερο επίπεδο είναι οι ερευνητές που ασχολούνται με ένα ή και περισσότερα έργα. Μέτα είναι οι διαχειριστές των έργων. Μαζί με τους διαχειριστές είναι και αυτοί που είναι υπεύθυνοι για τον προϋπολογισμό του κάθε έργου. Στο επόμενο σκαλί της ιεραρχίας είναι ο γενικός διαχειριστής που είναι υπεύθυνος για όλα τα έργα που αναλαμβάνει το Ι.Τ.Υ.Ε. Στην κορυφή της ιεραρχίας βρίσκεται ο διευθυντής του οργανισμού. Επίσης υπάρχει και ένας λογιστής ο οποίος αναλαμβάνει τις λογιστικές διαδικασίες του οργανισμού. Η παραπάνω ιεραρχική δομή φαίνεται και στο σχήμα που ακολουθεί.



Εικόνα 3.1 Οργανόγραμμα του Ι.Τ.Υ.Ε.

Η επόμενη σημαντική οντότητα στον τρόπο λειτουργίας του Ι.Τ.Υ.Ε. είναι τα ερευνητικά έργα. Το Ι.Τ.Υ.Ε. αναλαμβάνει πολλά έργα προς υλοποίηση. Τα έργα αυτά είναι ερευνητικής φύσης και αφορούν κυρίως τον τομέα της επιστήμης των υπολογιστών. Κάθε έργο έχει δικό του χρόνο αποπεράτωσης, καθώς επίσης και δικό του προϋπολογισμό που καθορίζονται κατά την φάση της σχεδίασης του έργου. Για την διευκόλυνση της υλοποίησης του έργου καθώς επίσης για την καλύτερη αξιολόγηση της προόδου το κάθε έργο χωρίζεται σε πακέτα εργασίας.

Τα πακέτα εργασίας ενός έργου είναι στην ουσία υποσύνολα του έργου και ανατίθενται σε συγκεκριμένες ομάδες εργασίας. Τα πακέτα αυτά μπορεί να είναι είτε αυτοτελή και να υλοποιηθούν ανεξάρτητα από τα υπόλοιπα ή μπορεί να χρειαστεί για την υλοποίηση ενός πακέτου εργασίας να πρέπει να έχουν ολοκληρωθεί κάποια προηγούμενα πακέτα.

3.2 Προδιαγραφές του συστήματος

Το σύστημα Νέδα είναι μια εφαρμογή παγκοσμίου ιστού και η αρχιτεκτονική της ακολουθεί το μοντέλο πελάτη εξυπηρετητή. Οι χρήστες μετά την σύνδεση τους στο σύστημα θα πρέπει να μπορούν να κάνουν λειτουργιές που αφορούν την διαχείριση των έργων που αναλαμβάνει το Ι.Τ.Υ.Ε. Οι λειτουργίες αυτές διαφέρουν από χρήστη σε χρήστη ανάλογα με την κατηγορία του χρήστη. Οι κατηγορίες των χρηστών στο σύστημα είναι οι παρακάτω:

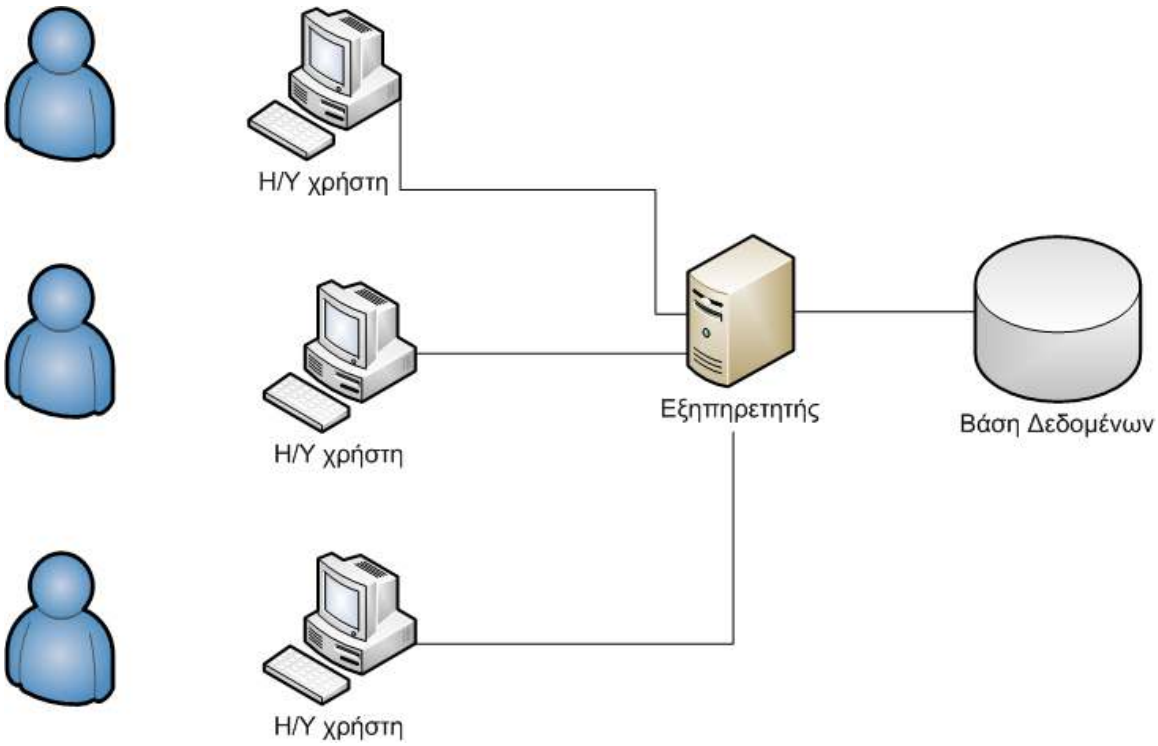
- Ερευνητής
- Επικεφαλής ερευνητής
- Διαχειριστής έργου
- Διαχειριστής συστήματος
- Επικεφαλής Διαχειριστής
- Διευθυντής
- Λογιστής

Όλες οι λειτουργίες στο σύστημα εκτελούνται μέσω συμπλήρωσης φορμών/αιτήσεων και ανάλογα με την λειτουργία υπάρχει και η αντίστοιχη ροή μέχρι την έγκριση ή την απόρριψη μιας αίτησης. Κατά τη διάρκεια αυτής της ροής υπάρχουν πολλοί διαφορετικοί χρήστες που εμπλέκονται και ο απώτερος σκοπός της Νέδας είναι να διευκολύνει αυτήν την διαδικασία έτσι ώστε να γίνεται όσο το δυνατόν γρηγορότερα και να αποφεύγονται τυχόν γραφειοκρατικά λάθη.

3.3 Ο σχεδιασμός του συστήματος

Ο σχεδιασμός του συστήματος είναι βασισμένος στο μοντέλο πελάτη εξυπηρετητή. Το σύστημα είναι μια εφαρμογή διαδικτύου και μπορεί να εξυπηρετήσει πολλούς χρήστες ταυτόχρονα. Όλα τα δεδομένα της εφαρμογής αποθηκεύονται σε μια σχεσιακή βάση δεδομένων. Ο εξυπηρετητής είναι υπεύθυνος για την εξυπηρέτηση των αιτήσεων των χρηστών, τον έλεγχο της ροής των διαδικασιών καθώς και επίσης τον έλεγχο της ορθότητας και συνέπειας των αιτήσεων σύμφωνα με τις προδιαγραφές του

συστήματος. Η δομή της εγκατάστασης και χρήσης φαίνεται στο παρακάτω σχήμα.



Εικόνα 3.2 Δίκτυο συστήματος Νέδα

Η βάση δεδομένων είναι MySQL Server, ο εξυπηρετητής τρέχει μια διαδικτυακή εφαρμογή Java πάνω σε Tomcat και οι χρήστες μέσω ενός φυλλομετρητή συνδέονται και εκτελούν τις εργασίες τους.

3.4 Η αρχιτεκτονική του Συστήματος

Η αρχιτεκτονική του συστήματος χωρίζει το σύστημα σε τέσσερα λειτουργικά επίπεδα

1) **Το επίπεδο αποθήκευσης.** Το επίπεδο αυτό είναι υπεύθυνο για την επικοινωνία με τη βάση δεδομένων για την ανάκτηση και αποθήκευση των δεδομένων. Η βάση όπως είπαμε παραπάνω είναι μια σχεσιακή βάση δεδομένων. Το επίπεδο αυτό χρησιμοποιείται από το επίπεδο λειτουργιών. Για τη διευκόλυνση της υλοποίησής μας καθώς επίσης για να έχουμε πιο ξεκάθαρο σχεδιασμό και

μοντελοποίηση των οντοτήτων μας, χρησιμοποιείται η τεχνολογία Hibernate.

2) **Το επίπεδο λειτουργιών.** Στο επίπεδο αυτό εκτελούνται όλες οι λειτουργικές απαιτήσεις του συστήματος. Με το επίπεδο αυτό επικοινωνεί το επίπεδο παρουσίασης και λειτουργεί ως διαμεσολαβητής μεταξύ του επιπέδου παρουσίασης και του επιπέδου αποθήκευσης. Για την υλοποίησή του χρησιμοποιούνται διάφορες τεχνολογίες σε Java.
3) **Επίπεδο παρουσίασης.** Το επίπεδο αυτό είναι υπεύθυνο για την παρουσίαση της εφαρμογής στους χρήστες ως σελίδων διαδικτύου. Επίσης είναι υπεύθυνο για να λαμβάνει τις αιτήσεις των χρηστών και να τις δρομολογεί στο επίπεδο λειτουργιών. Οι βασικές τεχνολογίες που χρησιμοποιούνται για την υλοποίησή του είναι το Spring MVC, html, css, JavaScript και πολλές άλλες τεχνολογίες που διευκολύνουν την σχεδίαση μιας διαδικτυακής εφαρμογής.
4) **Επίπεδο ασφάλειας.** Λόγο της φύσης του συστήματος η ασφάλεια είναι ένα πολύ σημαντικό κομμάτι. Το επίπεδο αυτό αναλαμβάνει την εξακρίβωση των χρηστών και είναι υπεύθυνο για το ότι ο κάθε χρήστης ανάλογα με τον ρόλο του θα μπορεί να κάνει εργασίες στο σύστημα που του επιτρέπονται. Η υλοποίηση αυτού του επιπέδου έγινε με την τεχνολογία Spring Security.

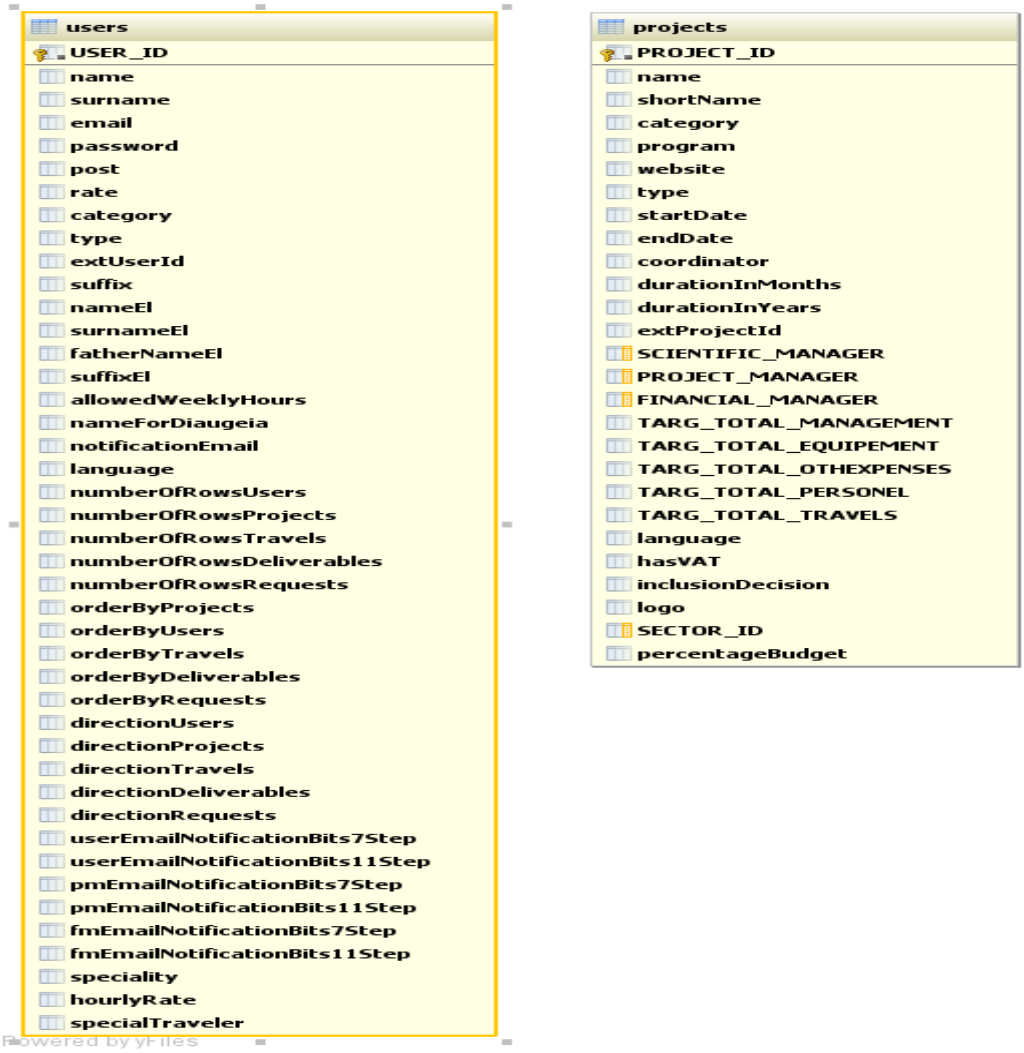
3.5 Βασικές οντότητες του συστήματος

Για να καλυφτούν οι απαιτήσεις του συστήματος δημιουργήθηκαν οι παρακάτω οντότητες. Οι οντότητες αυτές συσχετίζονται μεταξύ τους και δημιουργούν ένα E-R διάγραμμα με βάση το οποίο δημιουργείται το σχήμα της βάσης για την αποθήκευση των οντοτήτων. Οι οντότητες που αναφέρονται παρακάτω είναι οι πιο βασικές. Στο σύστημα υπάρχουν και άλλες οντότητες για την κάλυψη άλλων λειτουργικών απαιτήσεων. Παρακάτω περιγράφονται αυτές τις βασικές οντότητες και η σημασία της κάθε μίας.

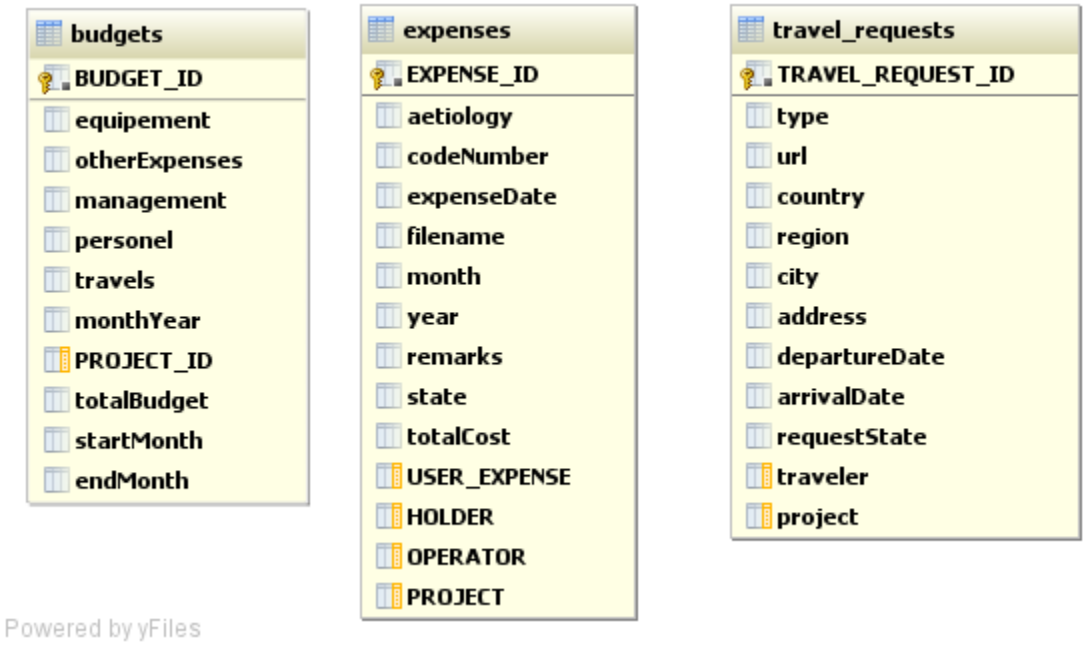
- **Χρήστες:** Αυτή η οντότητα αναπαριστά έναν χρήστη και περιέχει βασικές πληροφορίες για τον χρήστη, όνομα, επίθετο και ότι άλλο χρειάζεται από το σύστημα να γνωρίζει για τον χρήστη
- **Ερευνητικά Έργα:** Αυτή η οντότητα αναπαριστά ένα έργο που αναλαμβάνει το Ι.Τ.Υ.Ε. Κάθε ερευνητικό έργο έχει κάποιες βασικές ιδιότητες όπως το όνομα, τον προϋπολογισμό του και άλλα. Εκτός από αυτές τις βασικές ιδιότητες έχει και

έναν υπεύθυνο, έναν οικονομικό διαχειριστή, μια ερευνητική ομάδα και πακέτα εργασίας του έργου.

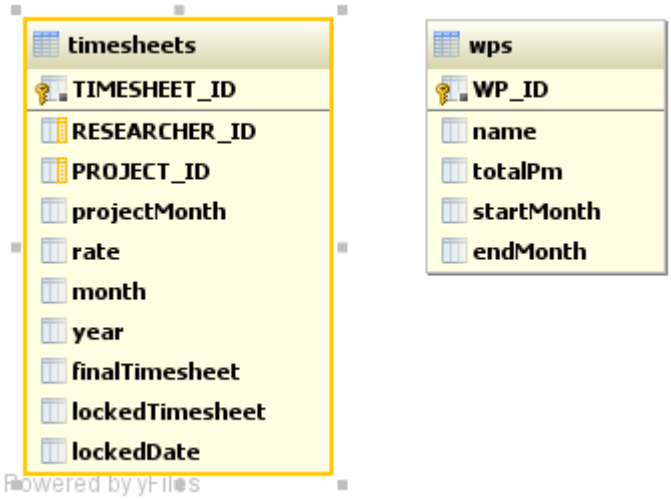
- **Προϋπολογισμός:** Ένας προϋπολογισμός περιλαμβάνει τις αμοιβές των ερευνητών αν είναι προϋπολογισμός έργου ή την αμοιβή ενός ερευνητή αν είναι προϋπολογισμός ερευνητή. Επίσης περιέχει πληροφορία για το μέρος του προϋπολογισμού που αφορά τα ταξίδια ενός έργου, τον εξοπλισμό που χρειάζεται ένα έργο και για τις αμοιβές των διαχειριστών του έργου.
- **Πακέτα Εργασίας:** Το πακέτο εργασίας περιγράφει ένα υποσύνολο ενός έργου, περιέχει πληροφορία για τις ανθρωποώρες που χρειάζονται για την αποπεράτωση του πακέτου εργασίας, τον μήνα έναρξης καθώς επίσης και τον μήνα λήξης του.
- **Timesheet:** Ένα timesheet αναπαριστά τις ώρες που εργάστηκε ένας ερευνητής σε ένα έργο. Περιέχει πληροφορία για τον χρήστη, το έργο, της ανθρωποώρες ανά πακέτο εργασίας, τις ώρες ανά ημέρα ανά πακέτο εργασίας και την ημερομηνία που συμπληρώθηκε το timesheet.
- **Αίτηση Δαπάνης:** Η αίτηση δαπάνης αναπαριστά μια αίτηση για δαπάνη ενός χρήστη και περιλαμβάνει πληροφορίες για τον χρήστη, το έργο για το οποίο γίνεται η δαπάνη, την κατάσταση τις αίτησης καθώς και συνοδευτικά έγγραφα για την αίτηση.
- **Αίτηση Ταξιδιού:** Αυτή η οντότητα αναπαριστά μια αίτηση ταξιδιού ενός ερευνητή, περιέχει πληροφορίες για το έργο, την προκαταβολή και διάφορα άλλα πεδία που είναι απαραίτητα.



Εικόνα 3.3 Οι πίνακες USERS και PROJECTS



Εικόνα 3.4 Οι πίνακες BUDGET, EXPENSE και TRAVEL_REQUEST

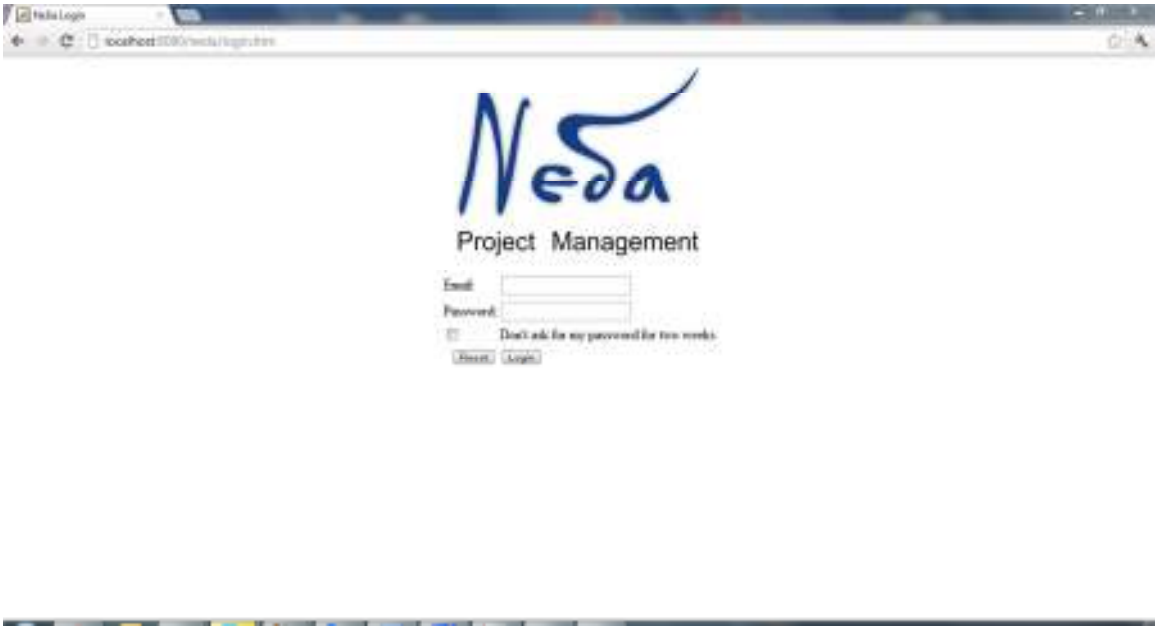


Εικόνα 3.5 Οι πίνακες TIMESHEETS και WPS

3.6 Βασικές λειτουργίες του συστήματος

Παρακάτω αναφέρονται οι βασικές λειτουργίες του συστήματος

- Εισαγωγή Χρήστη
- Επεξεργασία χρήστη
- Εισαγωγή έργου
- Προσθήκη ερευνητών σε έργο
- Προβολή έργου
- Καθορισμός προϋπολογισμού για ένα έργο
- Καθορισμός Person Months
- Συμπλήρωση timesheets
- Παρακολούθηση ενός πακέτου εργασίας
- Παρακολούθηση έργου σε έναν συγκεκριμένο μήνα
- Προσθήκη αίτησης δαπάνης
- Ολοκλήρωση αίτησης δαπάνης
- Ενημέρωση κατάστασης αιτήσεων
- Έγκριση δαπάνης
- Απόρριψη αίτησης δαπάνης
- Έγκριση αποπληρωμής δαπάνης
- Απόρριψη αποπληρωμής δαπάνης
- Προσθήκη αίτησης εξόδων ταξιδιού
- Ολοκλήρωση αίτησης απόδοσης εξόδων ταξιδιού
- Έγκριση αίτησης ταξιδίου
- Απόρριψη αίτησης ταξιδιού
- Έγκριση αποπληρωμής εξόδων ταξιδιού
- Απόρριψη αποπληρωμής εξόδων ταξιδιού



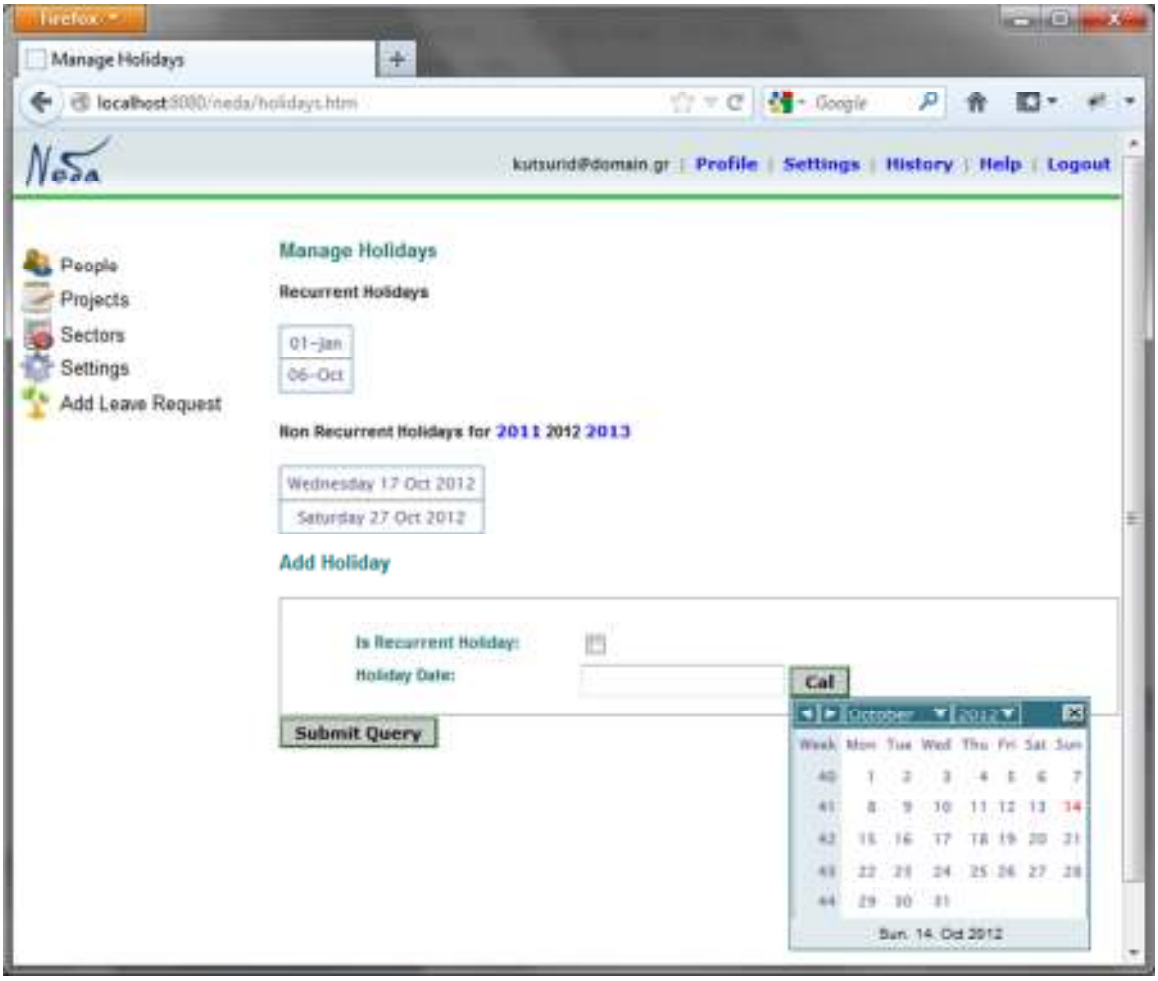
Εικόνα 3.6 Σελίδα σύνδεσης στο σύστημα

3.7 Νέες λειτουργίες

Στην παρούσα υποενότητα παρατίθενται και περιγράφονται οι λειτουργίες του συστήματος που προστέθηκαν κατά τη διάρκεια της εκπόνησης της παρούσας διπλωματικής εργασίας.

3.7.1 Εισαγωγή αργιών στο σύστημα

Αυτή η λειτουργία είναι διαθέσιμη μόνο στους διαχειριστές του συστήματος. Σκοπός είναι ο διαχειριστής να μπορεί να εισάγει τις αργίες κάθε χρόνου. Οι αργίες μπορεί να είναι επαναλαμβανόμενες όπως η 1 Ιανουαρίου ή να είναι συγκεκριμένες για κάθε χρόνο όπως το Πάσχα. Οι επαναλαμβανόμενες αργίες δεν χρειάζεται να εισάγονται για κάθε χρόνο. Μια άλλη λειτουργική απαίτηση είναι η αποστολή email στον διαχειριστή για να τον υπενθυμίσει να εισάγει τις αργίες του έτος μερικές μέρες πριν τελειώσει το έτος.



Εικόνα 3.7 Στιγμιότυπο της εφαρμογής για εισαγωγή αργιών

3.7.2 Καταγραφή ιστορικού των χρηστών

Κάθε φορά που ένας χρήστης κάνει μια ενέργεια στο σύστημα όπως εισαγωγή, επεξεργασία ή διαγραφή μιας οποιαδήποτε οντότητας στο σύστημα θα πρέπει να καταγράφεται αυτή η ενέργεια καθώς επίσης και μερικές πληροφορίες για το τι ενέργεια έκανε. Έτσι δημιουργείται ένα ιστορικό ενεργειών για κάθε χρήστη. Οι απαραίτητες πληροφορίες που πρέπει να καταγράφονται είναι:

- Ο χρήστης που εκτελεί την ενέργεια
- Η IP του μηχανήματος από το οποίο εκτελέστηκε η ενέργεια
- Η ημερομηνία

- Η συνεδρία του χρήστη (session), έτσι ώστε να ομαδοποιούνται εύκολα
- Ένα συμβολικό όνομα της ενέργειας
- Η οντότητα που άλλαξε

Το ιστορικό του κάθε χρήστη θα αποθηκεύεται στην βάση.

3.7.3 Προβολή ιστορικού των χρηστών

Ο διαχειριστής του συστήματος θα πρέπει να μπορεί να δει τις ενέργειες του κάθε χρήστη. Βασικές απαιτήσεις για αναζήτηση στο ιστορικό των ενεργειών είναι:

- Αναζήτηση βάση συνεδρίας
- Αναζήτηση βάση οντότητας
- Αναζήτηση βάση ημερομηνίας
- Αναζήτηση βάση λειτουργίας

Άλλες απαιτήσεις είναι η προβολή του ιστορικού μιας συγκεκριμένης οντότητας και η προβολή λεπτομερειών της οντότητας που άλλαξε.

3.7.4 Μηχανισμός αποστολής email

Το σύστημα Νέδα πρέπει να έχει την δυνατότητα αποστολής μηνυμάτων ηλεκτρονικού ταχυδρομείου. Η λειτουργία αυτή είναι πολύ χρήσιμη για την ενημέρωση των χρηστών χωρίς να έχουν συνδεθεί στο σύστημα. Μερικά παραδείγματα είναι τα εξής

- Υπενθύμιση χρηστών για ενέργειες που πρέπει να ολοκληρώσουν
- Υπενθύμιση διαχειριστών της εφαρμογής για παράτυπες ενέργειες των χρηστών
- Υπενθύμιση διαχειριστών έργων για την πρόοδο ενός έργου
- Άμεση αλληλεπίδραση των χρηστών μέσω της εφαρμογής.

Στόχος του υποσυστήματος που υλοποιήθηκε είναι να παρέχεται ένας μηχανισμός που μπορεί να παράγει ηλεκτρονικά μηνύματα βάση προτύπων κειμένων. Οι λεπτομέρειες της υλοποίησης αναφέρονται σε επόμενο κεφάλαιο.

3.7.5 Χρονοπρογραμματιστής

Το σύστημα Νέδα θα πρέπει να μπορεί να εκτελεί διάφορες ενέργειες σε συγκεκριμένες χρονικές στιγμές και χωρίς την διαμεσολάβηση κάποιου χρήστη. Ένα τέτοιο παράδειγμα είναι η αποστολή υπενθύμισης σε κάποιους χρήστες για να συμπληρώσουν της ώρες εργασίας τους πριν το τέλος του μήνα. Το υποσύστημα που θα υλοποιηθεί για την υποστήριξη της εκτέλεσης εργασιών θα πρέπει να έχει τα παρακάτω χαρακτηριστικά

- Εύκολη δημιουργία καινούργιας εργασίας
- Δυνατότητα ανάκαμψης σε περίπτωση σφάλματος της εφαρμογής (επανεκκίνηση συστήματος κτλ…)
- Δυνατότητα προσδιορισμού χρονικής στιγμής που θα εκτελεστεί η εργασία
- Υποστήριξη επαναλαμβανομένων εργασιών ανά χρονικά διαστήματα

Επίσης θα πρέπει το σύστημα που υλοποιήθηκε να μπορεί να λειτουργήσει σωστά και στην περίπτωση που το σύστημα Νέδα τρέχει σε περισσότερα από ένα μηχανήματα (cluster).

Κεφάλαιο 4 Τεχνολογίες και προϊόντα του συστήματος Νέδα

Στο παρόν κεφάλαιο παρουσιάζονται οι τεχνολογίες και τα προϊόντα που χρησιμοποιήθηκαν για την υλοποίηση των λειτουργιών που προστέθηκαν κατά την διάρκεια εκπόνησης της διπλωματικής εργασίας στο σύστημα Νέδα.

4.1 Η βάση δεδομένων MySQL

Η MySQL είναι μια βάση δεδομένων που ακολουθεί το σχεσιακό μοντέλο [22]. Είναι μια ευρέως διαδεδομένη βάση δεδομένων λόγο του ότι είναι ανοιχτού κώδικα και δωρεάν. Υποστηρίζει όλα τα διαδεδομένα λειτουργικά συστήματα. Από προεπιλογής η MySQL δεν προσφέρει κάποιο γραφικό περιβάλλον για την διαχείριση της και προσφέρει μόνο "command line"εργαλεία. Υπάρχουν όμως πολλά γραφικά εργαλεία για την διαχείριση μιας MySQL βάσης, πιο γνωστό από τα οποία είναι το MySQL WorkBench. Το MySQL WorkBench είναι μια εφαρμογή με την οποία μπορούμε μέσα από ένα γραφικό περιβάλλον να κάνουμε πολύ εύκολα τις πιο συχνές λειτουργίες σε μια βάση δεδομένων όπως:

- Δημιουργία πινάκων
- Εισαγωγή/διαγραφή εγγραφών
- Δημιουργία βάσης
- Εισαγωγή/Διαγραφή χρηστών καθώς και των δικαιωμάτων τους
- Εξαγωγή/εισαγωγή βάσης
- Και πολλά άλλα…

Εκτός από το MySQL WorkBench πολύ διαδεδομένο είναι και το phpMyAdmin το οποίο είναι μια διαδικτυακή εφαρμογή γραμμένη σε PHP με πολύ πλούσιο σετ λειτουργιών. Κατά τη διάρκεια της ανάπτυξης της εφαρμογής χρησιμοποιήθηκαν και τα δυο γραφικά εργαλεία που αναφέρθηκαν παραπάνω.

Η MySQL είναι μια RDBMS (σχεσιακή βάση δεδομένων), δηλαδή τα δεδομένα

αποθηκεύονται σε πίνακες και οι πίνακες συσχετίζονται μεταξύ τους με την χρήση κλειδιών. Αυτό το μοντέλο καθιστά την MySQL ιδανική για της απαιτήσεις της Νέδας.

Το λειτουργικό μοντέλο μιας RDBMS ακολουθεί το μοντέλο πελάτη εξυπηρετητή, δηλαδή να εγκαθίσταται σε ένα μηχάνημα (server) και ο πελάτης (client) υποβάλει ερωτήματα. Στην περίπτωση μας ο πελάτης είναι μόνο ένας, η Νέδα. Η επικοινωνία μιας εφαρμογής Java με μία βάση MySQL γίνεται μέσο ενός οδηγού (driver). Ο οδηγός είναι μια υλοποίηση του Java Database Connectivity API (JDBC). Με αυτόν τον τρόπο γίνεται πιο εύκολη η δημιουργία προγραμμάτων που υποβάλλουν SQL ερωτήσεις σε μια βάση. Παρόλα αυτά ο χρήστης στην περίπτωση που χρησιμοποιεί jdbc θα πρέπει να γράφει τα ερωτήματα του σε γλώσσα SQL.

4.1.1 Άλλες Βάσεις Δεδομένων

Μερικές από της πιο διαδομένες σχεσιακές βάσεις είναι:

- Oracle: εμπορική και στοχεύει σε εφαρμογές με μεγάλες απαιτήσεις. Προσφέρει πολλά έτοιμα εργαλεία για αποδοτικότερη χρήση της βάσης.
- PostgreSQL: Δωρεάν και ανοιχτού κώδικα, στοχεύει επίσης σε εφαρμογές με υψηλές απαιτήσεις.

4.2. Τεχνική συσχέτισης αντικειμένων σε σχέσεις

Ως τεχνική συσχέτισης αντικειμένων σε σχέσεις (object relation mapping), για συντομία ORM, εννοούμε μια τεχνική με την οποία συσχετίζουμε δεδομένα από μια σχεσιακή βάση σε αντικείμενα μιας αντικειμενοστραφής γλώσσας. Τα δεδομένα είναι αποθηκευμένα σε μια βάση δεδομένων, συνήθως σχεσιακή, και η συσχέτισή τους μας δίνει τις κλάσεις με τις οποίες αναπαριστούνται τα δεδομένα. Υπάρχουν πολλά έτοιμα πακέτα, είτε εμπορικά, είτε ανοιχτού κώδικα και για διάφορες γλώσσες που αναλαμβάνουν αυτήν την συσχέτιση. Σαν αρχή λειτουργίας η συσχέτιση δεδομένων από σχεσιακή βάση σε αντικείμενα μιας γλώσσας προγραμματισμού είναι απλή και κάποιος μπορεί να γράψει την δική του βιβλιοθήκη που να ακολουθεί αυτήν την τεχνική, αν θέλει να υλοποιήσει κάτι πολύ συγκεκριμένο και απλό.

Η ανάγκη για τις ORM τεχνικές προέκυψε από το ότι σε μια αντικειμενοστραφή γλώσσα

προγραμματισμού η επεξεργασία δεδομένων που δεν αναπαρίστανται ως αντικείμενα είναι χρονοβόρα και ο κώδικας που γράφεται δεν είναι ευανάγνωστος και συμπαγής. Για αυτό τον λόγο θέλουμε έναν μηχανισμό που θα αναλαμβάνει την αυτόματη μετατροπή μιας εγγραφής μιας βάσης δεδομένων σε ένα αντικείμενο της γλώσσας προγραμματισμού.

Συγκεκριμένα για την γλώσσα JAVA υπάρχει το Java Persistence API, το πρότυπο JDBC και το πακέτο java.sql. Τα παραπάνω προσφέρουν έναν ομοιόμορφο τρόπο για να διαχειριστούμε μια βάση δεδομένων, αλλά ο κώδικας που γράφουμε δεν είναι ευανάγνωστος. Επίσης ο προγραμματιστής θα πρέπει να γράψει κώδικα για να αναπαραστήσει τα δεδομένα που βρίσκονται σε μια βάση ως αντικείμενα της Java. Αυτή η τεχνική θέλει πολύ χρόνο και είναι ευάλωτη σε σφάλματα και οδηγεί πολλές φορές σε κώδικα που δεν είναι εύκολα κατανοητός και συντηρήσιμος. Αυτά τα προβλήματα λύνονται με την χρήση ενός πακέτου ORM, όπου το μόνο που έχει να κάνει πλέον ο προγραμματιστής είναι να δηλώσει μια αντιστοίχιση μεταξύ πινάκων της βάσης και αντικειμένων την Java. Έτσι ο προγραμματιστής πλέον διαχειρίζεται μόνο αντικείμενα και δεν χρειάζεται να γράψει κώδικα που να επικοινωνεί απευθείας με την βάση δεδομένων. Η αναπαράσταση της αντιστοίχισης αυτής διαφέρει από προϊόν σε προϊόν αλλά κοινός στόχος όλων των προϊόντων ORM είναι ο προγραμματιστής να έχει ένα επίπεδο ανάμεσα στην βάση δεδομένων και στον κώδικα του, το οποίο αναλαμβάνει αυτόματα την μετατροπή αντικειμένων σε οντότητες της βάσης. Ένα άλλο χαρακτηριστικό των συστημάτων ORM είναι η δυνατότητα να γράφονται ερωτήματα στην βάση δεδομένων με έναν πιο αφηρημένο και πιο αντικειμενοστραφή τρόπο σε σχέση με την απλή SQL. Τα ερωτήματα αυτά επιστρέφουν απευθείας αντικείμενα, ή μια συλλογή αντικειμένων.

4.2.1 Το περιβάλλον Hibernate

Η Hibernate είναι μια βιβλιοθήκη ORM για την γλώσσα Java. Είναι ανοιχτού κώδικα και διατίθεται από την JBOSS. Προσφέρει μια πλατφόρμα για αυτόματη μετατροπή εγγραφών από μια σχεσιακή βάση σε αντικείμενα της Java. Επίσης έχει πολλά βοηθήματα για διαχείριση και συντήρηση του σχήματος της βάσης και έναν πολύ κομψό τρόπο για να υποβάλουμε ερωτήματα στην βάση. Ένα από τα πλεονεκτήματα χρήσης Hibernate είναι ότι το σύστημα μας γίνεται πιο φορητό όσων αφορά την μετάβαση από μια βάση σε μια άλλη. Αυτό γίνεται επειδή το Hibernate μας προσφέρει

ένα διαφανές επίπεδο μεταξύ της εφαρμογής μας και της βάσης. Έτσι σαν προγραμματιστές δεν χρειάζεται να γνωρίζουμε της λεπτομέρειες της βάσης δεδομένων και να χρησιμοποιούμε απευθείας το Hibernate.

4.2.1.1 Συσχέτιση Εγγραφών με κλάσεις

Η συσχέτιση των εγγραφών της βάσης με αντικείμενα μπορεί να γίνει με δύο τρόπους. Είτε μέσα σε αρχείο XML είτε με annotations πάνω στις κλάσεις. Προσφέρονται επίσης όλες οι διευκολύνσεις στον προγραμματιστή για να δηλώσει συσχετίσεις μεταξύ των αντικειμένων όπως one to one, one to many και many to many.

Ένας πίνακας της βάσης μας συσχετίζεται με μία κλάση. Κάθε στήλη του πίνακα αυτού συσχετίζεται με μία ιδιότητα της κλάσης. Το hibernate αναλαμβάνει αυτόματα την μετατροπή στον σωστό τύπο, επίσης μας παρέχεται η δυνατότητα να κάνουμε εμείς την μετατροπή αυτήν.

Η συσχέτιση μέσω xml αρχείων είναι η πιο συνήθης στην πράξη. Τα αρχεία αυτά έχουν την κατάληξη hbm.xml. Κάθε αρχείο αναπαριστά την συσχέτιση μιας κλάσης με έναν πίνακα στην βάση. Ένα παράδειγμα ενός τέτοιου αρχείου είναι το παρακάτω:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

    <class name="prman.model.Holiday" table="HOLIDAYS">
        <id name="holidayId" column="HOLIDAY_ID">
            <generator class="native"/>
        </id>
        <property name="theDate"/>
        <property name="recursive" not-null="true"/>
    </class>
</hibernate-mapping>
```

Από το παραπάνω διακρίνουμε ότι η κλάση prman.model.Holiday αντιστοιχεί στον πίνακα HOLIDAYS. Ο πίνακας αυτός έχει 3 πεδία, ένα πρωτεύον κλειδί, το holidayId, ένα πεδίο με όνομα theDate, και ένα πεδίο με όνομα recursive, όπου ισχύει και ο περιορισμός να μην είναι κενό. Οι τύποι των πεδίων αυτών καθορίζονται από την

κλάση της Holiday της Java. Η μετατροπή όπως είπαμε και παραπάνω γίνεται αυτόματα. Η κλάση φαίνεται παρακάτω:

```
package prman.model;

import prman.data.manager.logging.LoggableEntity;

import java.util.Date;

/**
 * Maps a Single Holiday.
 */
public final class Holiday implements LoggableEntity {

    /**
     * The ID of the holiday.
     */
    private int holidayId;

    /**
     * The date of the holiday.
     */
    private Date theDate;

    /**
     * Defines if the holiday is recursive. The same date every year.
     */
    private boolean recursive;

    /**
     * This method return the id of a holiday.
     *
     * @return the holiday id
     */


    public int getHolidayId() {
        return holidayId;
    }

    /**
     * Set the id of the holiday.
     *
     * @param iHolidayId the id of the holiday
     */
    public void setHolidayId(final int iHolidayId) {
        this.holidayId = iHolidayId;
    }

    /**
     * Returns the holiday date.
     *
     * @return the date of the holiday
     */
    public Date getTheDate() {
```

```
        return theDate;
    }

    /**
     * Set the date of the holiday.
     *
     * @param theDateNew the date to be set
     */
    public void setTheDate(final Date theDateNew) {
        this.theDate = theDateNew;
    }

    /**
     * Returns true if the holiday is recursive.
     *
     * @return recursive
     */
    public boolean isRecursive() {
        return recursive;
    }

    /**
     * Sets the holiday as recursive.
     *
     * @param iRecursive true if the holiday is recursive
     */
    public void setRecursive(final boolean iRecursive) {
        this.recursive = iRecursive;
    }

    /**
     * Return the entity id as it persisted in database.
     *
     * @return the entity id.
     */
    public int getEntityId() {
        return holidayId;
    }

    /**
     * Overrides the toString method of this class.
     *
     * @return a string description for the class
     */
    @Override
    public String toString() {
        return new StringBuilder().append("Holiday{")
                .append("holidayId=").append(holidayId)
                .append(", theDate=").append(theDate)
                .append(", recursive=").append(recursive)
                .append('}').toString();
    }
}
```

Οι getters και οι setters της κλάσεις δεν είναι υποχρεωτικές καθώς το hibernate μπορεί να προσπελάσει και private μεταβλητές. Αν υπάρχουν όμως τότε καλούνται αυτοί.

4.2.1.2 Επίπεδο αποθήκευσης

Το Hibernate μας προσφέρει έναν διαφανή μηχανισμό στο επίπεδο αποθήκευσης. Σαν προγραμματιστές εμείς αποθηκεύουμε απευθείας το αντικείμενο και το hibernate αναλαμβάνει αυτόματα να παράγει την εντολή SQL και την μετατροπή των τύπων τον ιδιοτήτων του αντικειμένου. Η one to many ή η many to many συσχετίσεις δηλώνονται με μία λίστα ή ένα σετ στην java. Ένα βασικό χαρακτηριστικό της Hibernate είναι το lazy loading.

Η Hibernate μπορεί να λειτουργήσει με μια οποιαδήποτε σχεσιακή βάση δεδομένων όπως MySQL, Microsoft SQL Server, Postgre, Oracle και άλλες.

4.2.1.3 Επίπεδο Ερωτήσεων

Η Hibernate προσφέρει δύο τρόπους να κάνουμε ερωτήσεις στην βάση. Το ένα είναι η γλώσσα HQL (hibernate query language) και το άλλο είναι μέσα από το Criteria API που προσφέρει. Στην πρώτη περίπτωση μας παρέχεται μια γλώσσα παρόμοια με την SQL ενώ στην δεύτερη περίπτωση μας παρέχεται ένα Java API για μια πιο αντικειμενοστραφή προσέγγιση των ερωτημάτων που υποβάλλουμε. Οποιοδήποτε τρόπο και να χρησιμοποιήσουμε επιστρέφεται πάντα ένα αντικείμενο της java, κάτι που κάνει την εφαρμογή μας πιο συμπαγής και των κώδικα πιο κατανοητό.

4.2.1.4 Διαχείριση και συντήρηση σχήματος

Όταν χρησιμοποιούμε hibernate για το επίπεδο αποθήκευσης της εφαρμογής μας τότε δεν χρειάζεται να συντηρούμε το σχήμα της βάσης. Το Hibernate αναλαμβάνει αυτόματα την δημιουργία του. Η αυτόματη αυτή δημιουργία του σχήματος γίνεται βάση των XML αρχείων ρυθμίσεων ή από τα annotations πάνω στις κλάσεις. Επίσης ένα από τα πολύ χρήσιμα χαρακτηριστικά του είναι ότι όταν προσθέτουμε μια καινούρια ιδιότητα σε μια κλάση τότε αυτόματα εκτελούνται οι κατάλληλες εντολές σε SQL (π.χ.

alter table) έτσι ώστε να προστεθεί αύτη η στήλη στον κατάλληλο πίνακα. Η τεχνική αυτή ονομάζεται schema evolution (εξέλιξη σχήματος). Βέβαια αυτό δεν είναι δυνατόν να γίνεται με οποιαδήποτε αλλαγή στις κλάσεις μας. Για αυτό, παρότι γίνεται αυτόματα, χρειάζεται προσοχή από τους προγραμματιστές και σε πολλές περιπτώσεις χρειάζεται ο έλεγχος του σχήματος που παράχθηκε.

4.2.1.5 Επικρίσεις

Ο βασικός λόγος χρησιμοποίησης της Hibernate στην εφαρμογή μας είναι ότι κάνει την διαδικασία ανάπτυξης της εφαρμογής πιο γρήγορη, ο κώδικας είναι πιο συμπαγής και τα λάθη γίνονται ελάχιστα. Επίσης η εφαρμογή μας είναι πιο φορητή.

Υπάρχουν όμως περιπτώσεις όπου η εφαρμογή πρέπει να είναι πολύ γρήγορη. Σε αυτές τις περιπτώσεις η χρήση της Hibernate ή κάποιου άλλου συστήματος ORM είναι απαγορευτική καθώς προσθέτεται μια χρονική καθυστέρηση σε σχέση με την απευθείας χρησιμοποίηση του πρωτοκόλλου jdbc. Αυτού του είδους οι εφαρμογές είναι ελάχιστες και στην Νέδα δεν έχουμε τέτοιον περιορισμό και για αυτόν τον λόγο επιλέχθηκε η hibernate για την υλοποίηση του επιπέδου αποθήκευσης.

4.2.1.6 Ενσωμάτωση της Hibernate με το Spring

Το Spring μας παρέχει έναν μηχανισμό, έτοιμες κλάσεις και beans για να ενσωματώσουμε το hibernate μέσα στο application context του spring. Με αυτόν τον τρόπο μπορούμε να φτιάξουμε εύκολα beans που να χρησιμοποιούν το hibernate με έναν ομοιόμορφο τρόπο. Το βασικό bean που πρέπει να δηλώσουμε στο spring είναι το: org.springframework.orm.hibernate3.LocalSessionFactoryBean. Μέσα σε αυτό το bean θα πρέπει να καθορίσουμε τα αρχεία hbm.xml και το dataSource μας καθώς και άλλες ρυθμίσεις που επιθυμούμε. Η διαχείριση των δοσοληψιών μπορεί να γίνει επίσης μέσω βοηθητικών κλάσεων που προσφέρει το Spring ή μέσω μηχανισμών που προσφέρει το Hibernate.

4.3 Το περιβάλλον Spring

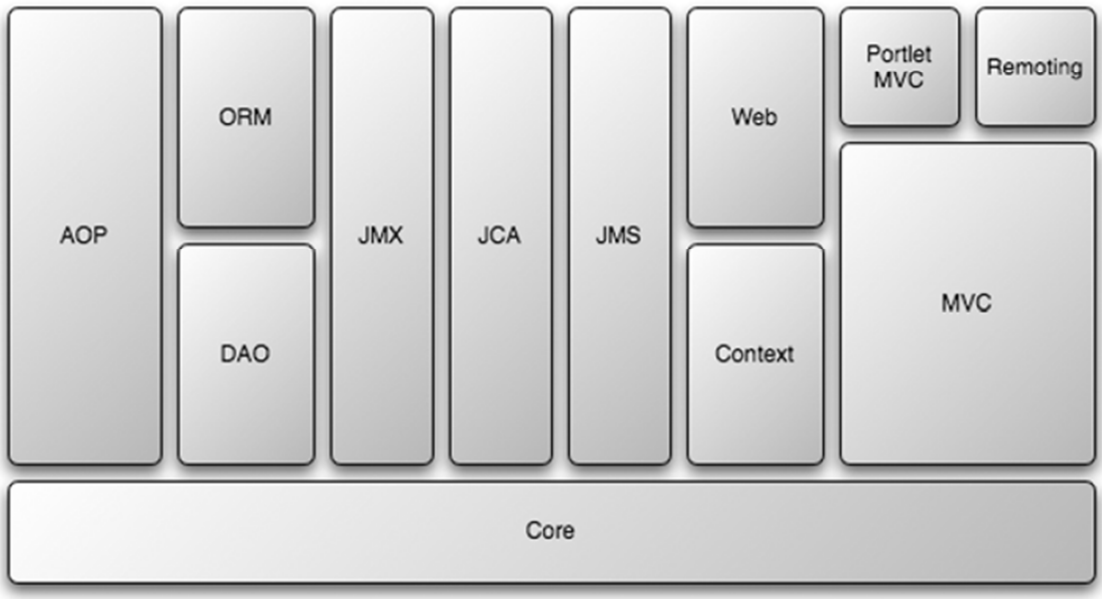
Ένα από τα προβλήματα που αντιμετωπίζουν οι προγραμματιστές κατά τη διάρκεια της ανάπτυξης λογισμικού είναι η ένωση των υποσυστημάτων που έχουν υλοποιήσει. Τα υποσυστήματα αυτά σε μια αντικειμενοστραφή γλώσσα προγραμματισμού είναι συνήθως κλάσεις. Η υλοποίηση τους και ο κώδικας που περιέχουν είναι συγκεκριμένος για το σύστημα που υλοποιείται, η συνένωση τους όμως είναι μια διαδικασία χρονοβόρα και δεν έχει σχέση με την λογική της εφαρμογής. Για αυτόν τον λόγο χρειαζόμαστε έναν εύκολο μηχανισμό γενικού σκοπού που να μας προσφέρει έναν ομοιόμορφο τρόπο να ενώσουμε τα υποσυστήματα που έχουμε δημιουργήσει. Αυτό το πρόβλημα προσπαθεί να λύσει το Spring [18]. Παρόλα αυτά τα εργαλεία που προσφέρει το Spring δεν περιορίζονται μόνο σε αυτό. Γενικά το Spring είναι μια πλατφόρμα για ανάπτυξη εφαρμογών σε Java που δίνει στον χρήστη τη δυνατότητα να ασχολείται μόνο με την ανάπτυξη του κώδικα της εφαρμογής του και όχι με την υλοποίηση της ένωσης των υποσυστημάτων που έχει γράψει.

Είναι ανοιχτού κώδικα βιβλιοθήκη για Java και χρησιμοποιείται κυρίως στην υλοποίηση διαδικτυακών εφαρμογών. Μερικά από τα βασικά του χαρακτηριστικά είναι τα παρακάτω:

- **Είναι ελαφρύ:** Το μέγεθος της βιβλιοθήκης είναι μικρό και οι καθυστερήσεις που προσθέτει στην εφαρμογή είναι και αυτές πάρα πολύ μικρές. Έτσι από άποψη ταχύτητας της εφαρμογής μας, δεν μας προσθέτει κάποιον περιορισμό.
- **Dependency injection:** Dependency Injection είναι μια τεχνική όπου τα αντικείμενα δεν αναζητούν ή δημιουργούν τις ιδιότητες τους, άλλα μας παρέχεται ένας μηχανισμός όπου δηλώνονται οι εξαρτήσεις κάθε κλάσης και αναλαμβάνεται αυτόματα η δημιουργία τους.
- **Aspect Oriented:** Το Spring παρέχει υποστήριξη για Aspect Oriented Programming έτσι ώστε κάθε υποσύστημα που γράφουμε να κάνει τη δουλεία που θέλουμε και μόνον αυτήν χωρίς να γνωρίζει λεπτομέρειες ή και την ύπαρξη άλλων υποσυστημάτων.
- **Container:** Το Spring είναι ένας container με την έννοια ότι για κάθε υποσύστημα που γράφουμε, ο κύκλος ζωής του διαχειρίζεται από το Spring.
- **Framework:** Το Spring είναι ένα περιβάλλον εργασίας με την έννοια ότι παρέχει τη δυνατότητα υλοποίησης πολύπλοκων εφαρμογών από μικρότερα και απλούστερα κομμάτια χρησιμοποιώντας κυρίως τους μηχανισμούς που προσφέρει όπως το MVC, το επίπεδο αποθήκευσης και άλλα.

Το Spring παρέχει επίσης διάφορα υποσυστήματα για την ευκολότερη ανάπτυξη των εφαρμογών μας. Αυτά τα υποσυστήματα αφορούν διάφορες κατηγορίες εφαρμογών και στοχεύουν στην ανάπτυξη συμπαγών εφαρμογών υλοποιημένων με έναν κοινό και κομψό τρόπο. Μερικά από αυτά τα υποσυστήματα αφορούν στην ανάπτυξη σύγχρονων εφαρμογών διαδικτύου χρησιμοποιώντας REST, AJAX, Web Services. Άλλα υποσυστήματα είναι για την διευκόλυνση του προγραμματιστή για το επίπεδο αποθήκευσης της εφαρμογής του, παρέχοντάς του μηχανισμούς για RDBMS συστήματα, No-SQL βάσεις για Cloud λύσεις.

Γενικά το Spring είναι ένα πολύ μεγάλο πακέτο και σχεδόν καμία εφαρμογή δεν χρησιμοποιεί όλα του τα υποσυστήματα.



Εικόνα 4.1 Βασικά Υποσυστήματα του Spring

Τα βασικά υποσυστήματα του Spring φαίνονται στο παραπάνω σχήμα. Βασικές έννοιες στο Spring είναι το Application Context και τα beans.

4.3.1 Spring και διαδικτυακές εφαρμογές

Κάθε διαδικτυακή εφαρμογή σε Java περιγράφεται από ένα αρχείο web.xml. Αυτό το αρχείο περιέχει τις δηλώσεις όλων των φίλτρων και των Servlets που έχουμε στην

εφαρμογή μας. Για να ενσωματώσουμε το Spring στην διαδικτυακή εφαρμογή μας το μόνο που χρειάζεται είναι να δηλώσουμε το παρακάτω μέσα στο web.xml

```
<context-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>
         classpath:prman/spring/prman-data.xml
         classpath:prman/spring/prman-model.xml
         classpath:prman/spring/prman-cache.xml
         classpath:prman/spring/prman-security.xml
         classpath:prman/spring/prman-notifications.xml
         classpath:prman/spring/prman-jobs.xml
         classpath:prman/spring/aitisi-data.xml
         classpath:prman/spring/prman-dataRemote.xml
         classpath:prman/spring/prman-globalBeans.xml
     </param-value>
 </context-param>

 <listener>
     <listener-class>
         org.springframework.web.context.ContextLoaderListener
     </listener-class>
 </listener>
```

Με την παραπάνω δήλωση δηλώνουμε πως όταν ξεκινήσει η web εφαρμογή μας, θα δημιουργήσουμε και το Application Context του Spring όπως αυτό ορίζεται από την contextConfigLocation παράμετρο. Η παράμετρος αυτή περιέχει τις τοποθεσίες των xml αρχείων. Μέσα σε αυτά τα xml αρχεία ορίζουμε τα beans της εφαρμογής μας. Το σύνολο των beans της εφαρμογής μας ονομάζεται application Context.

4.3.2 Περιγραφή των Spring beans

Η δήλωση των beans του Spring γίνεται συνήθως μέσω xml αρχείων, αν και δίνεται η δυνατότητα ορισμού των beans μέσο annotations πάνω στις κλάσεις. Κάθε bean αντιστοιχεί σε ένα στιγμιότυπο μιας κλάσης. Τα χαρακτηριστικά του κάθε bean είναι τα παρακάτω:

- **Η κλάση του bean:** Σε ποια κλάση της java αντιστοιχεί το bean που ορίζουμε
- **Το όνομα του:** Ένα όνομα που δίνουμε αυθαίρετα στο bean, το όνομα αυτό δεν χρειάζεται να είναι μοναδικό.
- **Το id του:** ένα μοναδικό αναγνωριστικό του bean
- **Αν είναι singleton ή prototype:** Τα singleton beans δημιουργούνται μία φορά

από το Spring και όλοι μοιράζονται το ίδιο στιγμιότυπο της κλάσης. Σε αντίθεση τα prototype beans που δημιουργούνται νέα στιγμιότυπα κάθε φορά που ζητείται το bean.

- **Ορίσματα constructor:** Τα ορίσματα του constructor.
- **Οι ιδιότητες της κλάσης:** Σε αυτό το σημείο ορίζουμε τις τιμές των ιδιοτήτων της κλάσης.
- **Μέθοδος αρχικοποίησης:** μία μέθοδος που θα εκτελεστεί αφού δημιουργηθεί το αντικείμενο.
- **Μέθοδος καταστροφής του:** μία μέθοδος που θα εκτελεστεί πριν καταστραφεί το αντικείμενο.

Ο κύκλος ζωής κάθε bean διαχειρίζεται από τον Spring και ο προγραμματιστής δεν χρειάζεται να επέμβει. Από τα παραπάνω υποχρεωτικό είναι μόνο το όνομα της κλάσης του bean. Ένα παράδειγμα ορισμού ενός bean είναι το παρακάτω:

```
    <bean id="homePageController"

class="prman.webapp.controllers.project.ListProjectController">
        <property name="tmStatsManager"
ref="timesheetStatisticsManager"/>
        <property name="trGlobalStatsManager"
ref="travelGlobalStatisticsManager"/>
        <property name="userManager" ref="userManager"/>
        <property name="researcherManager" ref="researcherManager"/>
        <property name="timesheetManager" ref="timesheetManager"/>
        <property name="travelManager" ref="travelManager"/>
        <property name="sectorManager" ref="sectorManager"/>
        <property name="globalProjectManager"
ref="globalProjectManager"/>
        <property name="deliverableManager" ref="deliverableManager"/>
        <property name="leaveManager" ref="leaveManager"/>
        <property name="overReqMan" ref="overtimeRequestManager"/>
    </bean>
```

Στην παραπάνω περίπτωση κάθε ιδιότητα της κλάσης ListProjectController είναι ένα άλλο bean που έχει οριστεί μέσα στο application Context. Με αυτόν τον τρόπο υλοποιούμε με πολύ κομψό τρόπο το γνωστό Singleton Pattern, χωρίς να χρειάζεται να γράφουμε στατικές μεθόδους και private constructors καθώς το Spring μας εγγυάται την μοναδικότητα του κάθε αντικειμένου.

4.3.3 Περιγραφή του Application Context

Ως Application Context θεωρείται στο Spring ο Container μέσα στον οποίον ζούνε όλα

τα beans. Είναι το υποσύστημα του Spring που αναλαμβάνει τον κύκλο ζωής του κάθε bean και το dependency injection μεταξύ τους. Μέσα από το application context μπορούμε να πάρουμε οποιοδήποτε bean θέλουμε. Όπως είπαμε και παραπάνω το application context ορίζεται συνήθως μέσα από xml αρχεία.

4.3.4 Υποσύστημα Spring security

Το Spring Security είναι ένα υποσύστημα του Spring που παρέχει εργαλεία στον προγραμματιστή για διαχείριση ρόλων, χρηστών και δικαιωμάτων. Ουσιαστικά προσφέρει έναν μηχανισμό για την πιστοποίηση των χρηστών καθώς και για τον έλεγχο των χρηστών αν έχουν δικαιώματα να εκτελέσουν μια ενέργεια. Είναι ένα σταθερό σύστημα και ευρέως χρησιμοποιούμενο. Μερικά από τα χαρακτηριστικά του που το καθιστούν ως η πρώτη επιλογή για τους προγραμματιστές διαδικτυακών εφαρμογών είναι:

- **Εύκολη ρύθμιση:** Η ρύθμιση του επιπέδου ασφαλείας γίνεται μέσω xml αρχείων. Είτε χρησιμοποιώντας τον dependency injection τρόπο, είτε χρησιμοποιώντας ένα XML namespace για το Spring Security.
- **Εύκολη εγκατάσταση:** Όλο το σύστημα ασφαλείας μπορεί να λειτουργήσει μέσα σε μια διαδικτυακή εφαρμογή χωρίς την ανάγκη πρόσθετων βιβλιοθηκών χρησιμοποιώντας μόνο τα φίλτρα που παρέχονται. Δεν χρειάζεται να γίνουν αλλαγές στον servlet container (π.χ. tomcat).
- **Διαχώριση κώδικα:** Στον κώδικα της εφαρμογής μας δεν χρειάζεται να γράψουμε κομμάτια που αφορούν την ασφάλεια και τα δικαιώματα των χρηστών. Με αυτόν τον τρόπο απομονώνουμε την λογική της εφαρμογής μας από την ασφάλεια κάνοντας τον κώδικα συμπαγή.
- **Επεκτάσιμη αρχιτεκτονική:** Το μοντέλο που χρησιμοποιείται από το spring security βασίζεται πάνω σε διεπαφές, κάνοντάς το εύκολο στην επέκταση του για μια πιο ειδικού σκοπού λύση.
- **Παροχή υπηρεσιών πιστοποίησης:** Παρέχει μια πληθώρα μεθόδων πιστοποίησης. Ο προγραμματιστής μπορεί εύκολα να καθορίσει δικαιώματα σε σελίδες χρησιμοποιώντας κανονικές εκφράσεις (regular expressions) στο url.
- **Μοντέλο υλοποίησης:** Προσφέρει υποστήριξη για ομάδες χρηστών, ιεραρχικούς ρόλους και ένα API για διαχείριση χρηστών. Όλα αυτά συνδυάζονται για να μειώσουν τον χρόνο που χρειάζεται η ανάπτυξη της εφαρμογής και κάνει την διαχείρισή της πιο εύκολη.

- **Χρήση πολλών μηχανισμών πιστοποίησης:** Δίνεται η δυνατότητα να ανακτούνται οι χρήστες και τα δικαιώματα τους από XML αρχεία, από μια βάση δεδομένων ή και από απλά αρχεία κειμένου. Τα παραπάνω μπορούν να επεκταθούν και να χρησιμοποιήσουμε μια οποιαδήποτε υπηρεσία πιστοποίησης θέλουμε.
- **Ενσωμάτωση με βάση δεδομένων:** Το Spring Security δεν βάζει περιορισμούς για το πώς θα αποθηκεύονται οι χρήστες, οι ομάδες και τα δικαιώματα, έτσι μπορούμε να χρησιμοποιήσουμε οποιονδήποτε τρόπο βολεύει την εκάστοτε εφαρμογή.
- **Υποστήριξη taglibs:** Μπορούμε να βάλουμε περιορισμούς στις σελίδες μας για το ποιος έχει δικαίωμα να τις δει, κατευθείαν μέσα στο αρχείο jsp χρησιμοποιώντας τα taglibs που προσφέρονται.
- **Ανοιχτού κώδικα:** Είναι ανοιχτού κώδικα και διατίθεται με την άδεια της Apache

Στην Νέδα το Spring security χρησιμοποιείται για το επίπεδο ασφάλειας της εφαρμογής. Οι χρήστες και οι ρόλοι τους είναι αποθηκευμένοι στην βάση. Για να ενεργοποιήσουμε το επίπεδο ασφαλείας του Spring πρέπει να δηλώσουμε το παρακάτω bean. Η δήλωσή του γίνεται χρησιμοποιώντας το XML namespace που προσφέρει το Spring.

```
    <security:authentication-provider>
        <security:password-encoder ref="passwordEncoder" hash="md5"
base64="false"/>
        <security:jdbc-user-service data-source-ref="dataSource"
                                    users-by-username-query="SELECT
email AS username, password, 1 AS enabled FROM USERS WHERE email = ?"
                                    authorities-by-username-
query="SELECT email as username, privilege
                FROM PROJECTROLES pr, USERS us, USERS_ROLES ur
                WHERE us.user_id = ur.user_Id AND pr.projectroles_id =
                ur.projectroles_id
                AND us.email = ?"/>
    </security:authentication-provider>

    <bean id="passwordEncoder"
class="org.springframework.security.providers.encoding.Md5PasswordEncod
er"/>
```

Από το παραπάνω φαίνεται ότι χρησιμοποιούμε κωδικοποίηση MD5 για την αποθήκευση των κωδικών και επίσης παρέχουμε και τον τρόπο, μέσω ερωτημάτων SQL, για το πώς θα βρίσκει τους χρήστες και τα δικαιώματα τους. Το επόμενο βήμα

είναι να δηλώσουμε ποιες σελίδες μπορούν να προσπελαστούν και από ποιους χρήστες. Υπάρχουν πολλοί τρόποι για να υλοποιηθεί αυτό. Ένας από αυτούς είναι να δηλώσουμε Interceptors ασφαλείας σε urls. Αυτή η μέθοδος φαίνεται στο παρακάτω κομμάτι:

```
<security:intercept-url pattern="/editUser.htm"
                                access="ROLE_SYSTEMADMINISTRATOR,
ROLE_HEADADMINISTRATOR"
                                requires-channel="http"/>
```

με το παραπάνω δηλώνουμε πως στο url editUser.htm έχουν δικαίωμα μόνο όσοι έχουν ρόλο ROLE_SYSTEMADMINISTRATOR ή ρόλο ROLE_HEADADMINISTRATOR.

Ένας άλλος τρόπος καθορισμού δικαιωμάτων σε σελίδες είναι χρησιμοποιώντας το security taglib που προσφέρεται απευθείας μέσα στα jsp αρχεία όπως φαίνεται παρακάτω:

```
<%@ page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"
%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="authz"
uri="http://www.springframework.org/security/tags" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ include file="../dhtmlgoodies_calendar/calendar_css.jsp" %>
<%@ include file="../dhtmlgoodies_calendar/dhtmlgoodies_calendar.jsp"
%>

<authz:authorize

ifNotGranted="ROLE_SYSTEMADMINISTRATOR,ROLE_HEADADMINISTRATOR,ROLE_ACCO
UNTANT,ROLE_DIRECTOR">
    <jsp:forward page="../authorization.jsp"/>
</authz:authorize>

<h3><spring:message code="heading.holidays"/></h3>

<h4>Recurrent Holidays</h4>
```

Από το παραπάνω κομμάτι κώδικα βλέπουμε πως μέσα στο tag <auth:authorize> δηλώνουμε πως αν ο χρήστης δεν έχει τον επιθυμητό ρόλο τότε τον παραθέτουμε στην σελίδα authorization.jsp

4.5 Πρότυπο Json

Το JSON [13] ή αλλιώς (JavaScript Object Notation) είναι ένα ευανάγνωστο από τον άνθρωπο πρότυπο για ανταλλαγή δεδομένων σε μορφή κειμένου. Προέρχεται από την JavaScript και η κυρίως χρήση του είναι η ανταλλαγή δεδομένων μεταξύ ενός εξυπηρετητή και ενός πελάτη (κυρίως ένας φυλλομετρητής). Με το JSON μπορούμε να αναπαραστήσουμε απλές δομές δεδομένων και αντικείμενα. Παρότι είναι συσχετισμένο με την JavaScript, είναι ανεξάρτητο από μια συγκεκριμένη γλώσσα προγραμματισμού και υπάρχουν έτοιμοι parsers για πολλές γλώσσες.

Η κυρίως χρήση του είναι για να αναπαραστήσουμε δομημένα αντικείμενα και να τα μεταφέρομε μέσω δικτύου. Μπορεί να θεωρηθεί ως μία εναλλακτική λύση της XML. Σε πολλές περιπτώσεις επιλέγεται το JSON έναντι της XML για τους παρακάτω λόγους:

- Ευανάγνωστη μορφή από τον άνθρωπο
- Πολύ απλή στην χρήση από τους προγραμματιστές
- Τα δεδομένα δεν ακολουθούν κάποιο συγκεκριμένο σχήμα και δεν υπόκεινται σε περιορισμούς
- Μια έκφραση JSON είναι και μια έγκυρη έκφραση JavaScript

Το τελευταίο είναι και το πιο σημαντικό καθώς μας επιτρέπει να ανταλλάσουμε αντικείμενα και δομές δεδομένων μεταξύ του εξυπηρετητή και του φυλλομετρητή του χρήστη. Για την ακρίβεια μια αναπαράσταση σε JSON είναι και ο τρόπος που ορίζουμε ένα αντικείμενο στην JavaScript. Για παράδειγμα αν έχουμε την παρακάτω αναπαράσταση σε JSON:

```
{"holidayId":2, "theDate":null, "recursive":true, "entityId":2}
```

Μπορούμε εύκολα να ανακτήσουμε σε JavaScript το αντικείμενο γράφοντας:

```
var holiday = {"holidayId":2, "theDate":null, "recursive":true,
"entityId":2};
```

Στην Νέδα το JSON χρησιμοποιήθηκε εκτεταμένα στο υποσύστημα που αναλαμβάνει την καταγραφή γεγονότων των χρηστών και στην εμφάνισή τους στον χρήστη. Η βιβλιοθήκη που χρησιμοποιήθηκε για την αναπαράσταση των αντικειμένων Java σε JSON μορφή είναι η Jackson JSON Processor η οποία είναι ανοιχτού κώδικα και χρησιμοποιείται ευρέως.

4.6 Μεθοδολογία προγραμματισμού AOP

Ως AOP (Aspect Oriented Programming) στην επιστήμη της πληροφορικής εννοούμε ένα μοντέλο προγραμματισμού του οποίου ο στόχος είναι να μεγιστοποιήσουμε την δυνατότητα του συστήματος μας να κατατμηθεί σε μικρότερα, ανεξάρτητα, υποσυστήματα. Για να γίνει αυτό πραγματικότητα θα πρέπει να ελαχιστοποιηθεί ο κώδικας που γράφει ο προγραμματιστής που αφορά την συνένωση αυτών των υποσυστημάτων.

Το AOP σαν μεθοδολογία εισάγει τις κατάλληλες μεθόδους προγραμματισμού καθώς επίσης και τα κατάλληλα εργαλεία για να επιτευχτεί αυτό. Στόχος της τεχνικής αυτής είναι να διαχωριστεί ο κώδικας του συστήματος σε ξεχωριστά και ανεξάρτητα μέρη ανάλογα με την λογική του κάθε κομματιού. Όλες οι μεθοδολογίες προγραμματισμού προσφέρουν τεχνικές για ομαδοποίηση και ενσωμάτωση διαφόρων συστατικών του κώδικα σε ανεξάρτητες οντότητες. Μερικές όμως λειτουργίες χρειάζεται να γίνουν ανάμεσα σε αυτά τα ανεξάρτητα υποσυστήματα. Ένα κλασικό παράδειγμα είναι το Logging (καταγραφή ενεργειών) της εφαρμογής.

Στην μεθοδολογία AOP έχουμε την δυνατότητα να ενσωματώσουμε εντολές που μπορούν να εκτελεστούν πριν ή μετά από την κλήση μιας μεθόδου.

4.6.1 Βασικές έννοιες

Μερικές βασικές έννοιες της AOP μεθοδολογίας που ισχύουν πάντα παρατίθενται παρακάτω:

- **Advice:** Είναι ο επιπλέον κώδικας που θέλουμε να εκτελεστεί στο ήδη υπάρχον μοντέλο μας. Για παράδειγμα να κάνουμε logging κάθε φορά που εκτελείται μια μέθοδος.
- **Pointcut:** Είναι το σημείο στον κώδικα μας όπου θέλουμε να εφαρμόσουμε την λογική της εκτέλεσης επιπλέον κώδικα, μπορεί να είναι η έναρξη εκτέλεσης μιας μεθόδου ή μετά την εκτέλεση της μεθόδου.
- **Aspect:** Ο συνδυασμός του Pointcut και του Advice ονομάζεται Aspect. Στο παράδειγμα του logging προσθέτουμε ένα aspect στο σύστημά μας ορίζοντας ένα pointcut και παρέχοντας το advise για αυτό το pointcut.

Παράδειγμα για τις παραπάνω έννοιες δίνεται σε επόμενο κεφάλαιο καθώς το

υποσύστημα καταγραφής ενεργειών των χρηστών της Νέδα έχει υλοποιηθεί χρησιμοποιώντας την μεθοδολογία που υποβάλλεται από το AOP.

Οι υλοποιήσεις και οι δυνατότητες των εργαλείων που προσφέρουν AOP διαφέρουν ανάλογα με την γλώσσα προγραμματισμού. Εδώ θα αναφέρουμε υλοποιήσεις του AOP όσον αφορά την γλώσσα Java και πιο συγκεκριμένα το AspectJ και το Spring AOP.

4.7 Μεθοδολογία προγραμματισμού MVC

Το MVC (Model View Controller) είναι μια μεθοδολογία προγραμματισμού που χρησιμοποιείται σε εφαρμογές που έχουν να κάνουν με διεπαφές χρήστη. Η βασική του αρχή είναι ο διαχωρισμός της αναπαράστασης στον χρήστη από την λογική της εφαρμογής. Ο στόχος αυτός επιτυγχάνεται με την χρησιμοποίηση των τριών παραπάνω εννοιών: Μοντέλο, Προβολή και Διαχείριση.

- **Model:** Ως μοντέλο θεωρούμε τα δεδομένα ή κάποιους κανόνες
- **View:** Είναι μια οποιαδήποτε αναπαράσταση των δεδομένων, δηλαδή του μοντέλου μας.
- **Controller:** Είναι το υποσύστημα το οποίο αναλαμβάνει να δημιουργήσει το μοντέλο βάση της λογικής της εφαρμογής

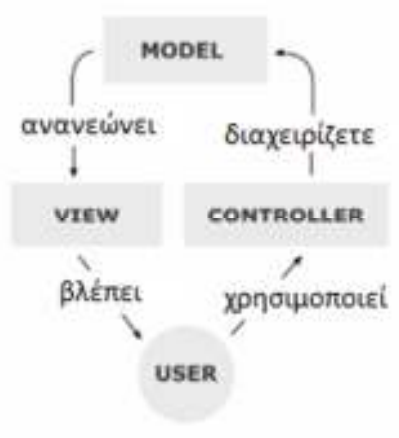
Από τα παραπάνω είναι φανερό ότι μπορούμε να εφαρμόσουμε στο ίδιο μοντέλο διάφορες αναπαραστάσεις και έτσι ο χρήστης να βλέπει διαφορετικά τα ίδια δεδομένα. Χρησιμοποιώντας την παραπάνω μεθοδολογία προγραμματισμού κάνουμε τον κώδικα πιο διαχωρίσιμο ανάλογα με τις λειτουργικές μονάδες και επίσης αποφεύγουμε να γράφουμε τον ίδιο κώδικα πολλές φορές γιατί για κάθε όψη του μοντέλου θα γράψουμε μόνο των κώδικα που αφορά στην όψη και θα αφήσουμε αναλλοίωτα τα άλλα δυο υποσυστήματα.

4.7.1 Αλληλεπίδραση μεταξύ των συστατικών του MVC

Εκτός από την διαχώριση της εφαρμογής σε 3 είδη συστατικών όπως αναφέρονται παραπάνω η MVC μεθοδολογία μας δίνει επίσης τις αλληλεπιδράσεις μεταξύ τους.

- Ο controller μπορεί να στείλει εντολές στο view με το οποίο συσχετίζεται για να αλλάξει την αναπαράσταση του μοντέλου. Επίσης μπορεί να αλλάξει το μοντέλο με το οποίο συσχετίζεται.
- Το μοντέλο μπορεί να στείλει μηνύματα στον controller και στο view για να τους ενημερώσει όταν αλλάζει η κατάσταση του.
- Ένα view μπορεί να ρωτήσει το μοντέλο πληροφορίες που χρειάζεται για να παράγει την όψη του.

Οι παραπάνω συσχετίσεις φαίνονται στο σχήμα που ακολουθεί.



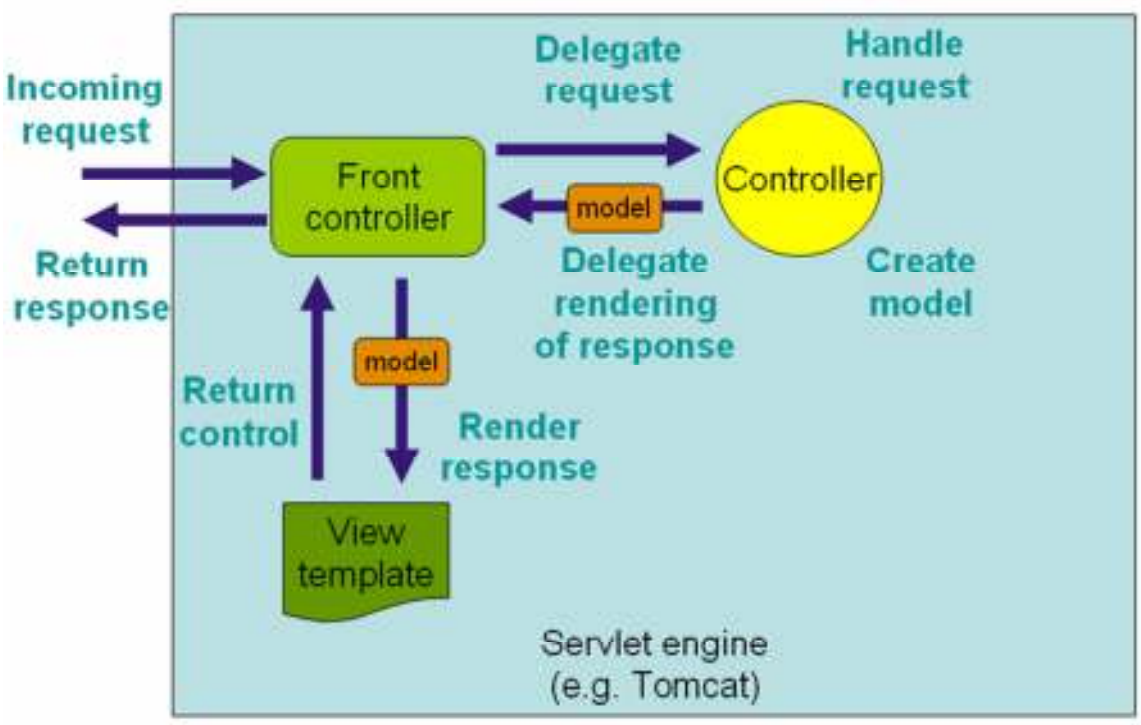
Εικόνα 4.2 Συσχετίσεις οντοτήτων στο MVC

4.7.2 MVC σε διαδικτυακές εφαρμογές

Το MVC για διαδικτυακές εφαρμογές είναι λίγο πιο διαφοροποιημένο και αυτό επειδή οι επικοινωνία μεταξύ χρήστη και εφαρμογής είναι χρονοβόρα και έτσι οι λειτουργίες του και τα συστατικά του MVC χωρίζονται στον client και στον server. Και αυτό το μοντέλο λειτουργίας όμως δεν είναι πολύ αποδοτικό και έτσι τα περισσότερα πακέτα για MVC έχουν ενσωματώσει όλες τις λειτουργίες και τα συστατικά μέσα στον server. Ένα MVC framework που ανήκει σε αυτήν την κατηγορία είναι και το Spring MVC

4.7.3 MVC του περιβάλλοντος Spring

Το Spring MVC είναι το υποσύστημα του Spring που είναι υπεύθυνο για το επίπεδο παρουσίασης μιας εφαρμογής. Σκοπός του είναι να παράγει σελίδες html βάση ενός μοντέλου δεδομένων και ενός controller, όμως δεν περιορίζεται μόνο σε αυτό καθώς μπορεί να χρησιμοποιηθεί για την υλοποίηση REST service. Βασική του αρχή είναι το DispatcherServlet το οποίο αναλαμβάνει την μεταγωγή των αιτήσεων στον εξυπηρετητή στον κατάλληλο Controller. Ο Controller με την σειρά του διαλέγει το View, παράγει το μοντέλο και εφαρμόζει το μοντέλο πάνω στο view.



Εικόνα 4.3 Ροή εκτέλεσης στο Spring MVC

Για να χρησιμοποιήσουμε το Spring MVC στην web εφαρμογή μας θα πρέπει να δηλώσουμε το DispatcherServlet στο web.xml ως εξής:

```
    <servlet>
        <servlet-name>neda</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet
        </servlet-class>

        <init-param>
```

```
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:prman/spring/prman-servlet.xml
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
```

Μέσα στο prman-servlet ορίζουμε στην Νέδα όλα τα views και τους controllers. Επίσης χρειάζεται ένας ViewResolver. Ο ViewResolver είναι υπεύθυνος για να διαλέξει το κατάλληλο view βάση του url που ζητήθηκε.

Ένας τρόπος να ορίσουμε τους Controllers μας στο Spring είναι μέσα από ορισμούς beans μέσα σε xml αρχεία του Spring. Ένας άλλος τρόπος είναι με χρήση του annotation @Controller πάνω σε μία κλάση. Στην Νέδα όλοι οι Controllers είναι δηλωμένοι μέσα σε xml αρχείο και συγκεκριμένα μέσα στο /spring/prman-servlet.xml.

4.8 Ο χρονοπρογραμματιστής Quartz

Το Quartz [16] είναι ένα ανοιχτού κώδικα προϊόν για τον χρονοπρογραμματισμό εργασιών υλοποιημένο εξ ολοκλήρου σε Java και μπορεί να ενσωματωθεί σε οποιαδήποτε Java εφαρμογή. Μπορεί να χρησιμοποιηθεί για τον χρονοπρογραμματισμό και την εκτέλεση δεκάδων χιλιάδων εργασιών. Η κάθε εργασία στο Quartz ορίζετε ως μια Java κλάση και μπορεί να εκτελέσει οτιδήποτε επιβάλλεται από τις ανάγκες του προγράμματος.

Γενικά ως χρονοπρογραμματιστή εργασιών θεωρούμε ένα σύστημα το οποίο είναι υπεύθυνο για τον έλεγχο και την εκτέλεση εργασιών που τρέχουν στο παρασκήνιο. Για τις εργασίες αυτές ο χρήστης θα μπορεί να ορίσει την χρονική στιγμή που θέλει να τρέξει η εργασία. Επίσης ένας χρονοπρογραμματιστής εργασιών θα πρέπει να προσφέρει έναν μηχανισμό ανάνηψης από αποτυχημένες εργασίες. Το Quartz θεωρείται ως το καλύτερο προϊόν για περιβάλλον Java λόγο του ότι προσφέρει ένα πολύ πλούσιο σετ χαρακτηριστικών, είναι ανοιχτού κώδικα και έχει μεγάλη ευκολία στην χρήση. Οι τρεις αυτοί λόγοι και το ότι προσφέρει έναν πολύ καλό μηχανισμό για ανάκαμψη και επανεκτέλεση των εργασιών που απέτυχαν μας οδήγησαν στο να το επιλέξουμε για να το ενσωματώσουμε στην Νέδα.

Ένας χρονοπρογραμματιστής εργασιών είναι ένα απαραίτητο κομμάτι για ένα σύστημα όπως η Νέδα. Στην Νέδα έχουμε πολλούς χρήστες , πολλά έργα και γενικά ο όγκος των δεδομένων μέσα στο σύστημα είναι μεγάλος και η επεξεργασία τους μερικές φορές

πρέπει να γίνεται ασύγχρονα. Από την φύση και τις απαιτήσεις της Νέδας πολλές εργασίες πρέπει να εκτελεστούν σε συγκεκριμένες χρονικές στιγμές. Ένα παράδειγμα είναι η υπενθύμιση με mail χρηστών που πρέπει να κάνουν κάποια εισαγωγή στο σύστημα, όπως οι ώρες εργασίας τους σε ένα έργο κάθε τέλος του μήνα.

4.8.1 Τρόπος λειτουργίας του Quartz και βασικές έννοιες

Στο Quartz κάθε εργασία υλοποιείται ως μια κλάση που υλοποιεί ένα Interface. Κάθε εργασία συνδέεται με ένα trigger το οποίο περιέχει την πληροφορία για το πότε ή το κάθε πόσο θα εκτελείται μια εργασία. Υπάρχουν δύο ειδών triggers, τα απλά και τα Cron Triggers

- **Απλά triggers:** Συμβαίνουν μόνο μια συγκεκριμένη χρονική στιγμή, και η διεργασία εκτελείται μια φορά
- **Cron triggers:** Συμβαίνουν πολλές φορές, επαναλαμβανόμενα και η διεργασία εκτελείται κάθε φορά.

Από τα παραπάνω δυο είδη αυτό που χρησιμοποιούμε πιο συχνά είναι τα Cron Triggers. Το όνομά τους προέρχεται από τα Cron Jobs του λειτουργικού συστήματος Unix καθώς έχουν το ίδιο συντακτικό (γνωστό και ως σύνταξη Cron ή Cron expression) για να ορίσουμε το κάθε πότε θα εκτελείται η εργασία.

Το Quartz μπορεί να λειτουργήσει με δύο τρόπους που εξαρτώνται από την τοπολογία της εφαρμογής. Η πρώτη και πιο απλή περίπτωση είναι να έχουμε μία εφαρμογή που τρέχει σε έναν εξυπηρετητή. Η δεύτερη περίπτωση είναι όταν η εφαρμογή μας τρέχει σε μια συστοιχία από εξυπηρετητές οπότε και θα πρέπει να λυθεί το πρόβλημα του αμοιβαίου αποκλεισμού για να μην τρέξει η ίδια εργασία σε περισσότερα από ένα μηχανήματα. Το Quartz λύνει μόνο του αυτό το πρόβλημα παρέχοντας έναν διαφανή μηχανισμό χρονοπρογραμματισμού ανεξάρτητα από την τοπολογία εγκατάστασης της εφαρμογής μας. Στην δεύτερη περίπτωση όπου έχουμε συστοιχία εξυπηρετητών θα πρέπει να ρυθμιστεί κατάλληλα το Quartz έτσι ώστε να χρησιμοποιεί μια βάση δεδομένων για να αποθηκεύει τις εργασίες και τα triggers. Η βάση μπορεί να είναι MySQL, ORACLE, Postgre και πολλές άλλες γνωστές RDBMS βάσεις.

4.8.2 Ενσωμάτωση με το Spring

Παρότι η δημιουργία εργασιών με το API που προσφέρει το Quartz είναι αρκετά εύκολη το Spring μας διευκολύνει ακόμα περισσότερο προσφέροντας έτοιμες κλάσεις και beans για να είναι η εφαρμογή μας πιο καλά δομημένη και ομοιόμορφη. Έτσι κάθε εργασία και κάθε trigger δηλώνεται ως bean στο Application Context μας ελαχιστοποιώντας των κώδικα που χρειάζεται να γράψουμε για την ρύθμιση του χρονοπρογραμματιστή μας. Η βασική κλάση που μας προσφέρει το Spring είναι η

org.springframework.scheduling.quartz.SchedulerFactoryBean.

Για να ορίσουμε έναν Quartz χρονοπρογραμματιστή δηλώνουμε το παραπάνω bean σε ένα αρχείο ρυθμίσεων του Spring. Το παραπάνω bean είναι ένα FactoryBean που σημαίνει ότι το τελικό μας bean θα είναι τύπου org.quartz.Scheduler. Στο παραπάνω bean δηλώνουμε όλα τα triggers που θέλουμε καθώς επίσης και τις εργασίες που θα τρέχουνε. Το κυρίως πλεονέκτημα της χρήσης του Quartz μέσα από τον container του Spring είναι ότι στις εργασίες μπορούμε να έχουμε με έναν κομψό τρόπο διαθέσιμα τα beans του application context.

Στην Νέδα, το Quartz έχει ρυθμιστεί να λειτουργεί χωρίς να χρησιμοποιεί κάποια βάση δεδομένων.

4.9 Μηχανές Αρχέτυπων

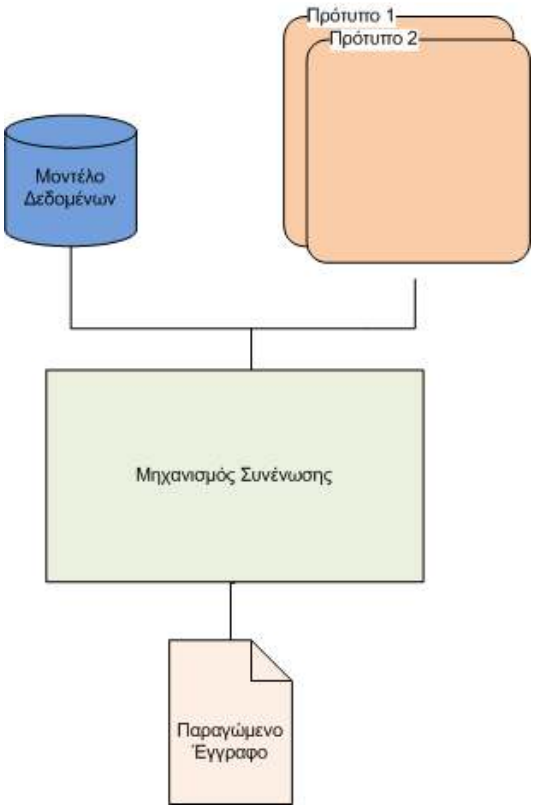
Ως Μηχανές αρχέτυπων (template engine) ορίζουμε την κατηγορία των προϊόντων λογισμικού τα οποία αποσκοπούν στην ένωση ενός μοντέλου δεδομένων (data model) με ένα πρότυπο (template) έτσι ώστε να παραχθεί ένα έγγραφο. Το είδος του εγγράφου ποικίλει από εφαρμογή σε εφαρμογή ανάλογα τις απαιτήσεις και μπορεί να είναι απλό έγγραφο κειμένου, ένα έγγραφο html ή και πηγαίος κώδικας σε κάποια γλώσσα προγραμματισμού. Τα βασικά συστατικά ενός οποιουδήποτε template engine είναι:

- **Το μοντέλο δεδομένων:** Το μοντέλο δεδομένων είναι μια συλλογή από δεδομένα, τα δεδομένα μπορεί να είναι μεταβλητές μιας γλώσσας προγραμματισμού ή ένα xml αρχείο. Γενικά ως μοντέλο θεωρούμε την είσοδο

που θα εφαρμοστεί στο πρότυπο.

- **Μια συλλογή από πρότυπα:** Ως πρότυπο ονομάζουμε ένα έγγραφο πάνω στο οποίο θα εφαρμόζεται το μοντέλο δεδομένων. Τα πρότυπα αυτά είναι προσδιορισμένα σε μια γενική μορφή έτσι ώστε να μπορεί να εφαρμοστεί το μοντέλο δεδομένων πάνω τους για να παραχθεί το επιθυμητό αποτέλεσμα.
- **Ο μηχανισμός που αναλαμβάνει την συνένωση τους:** Είναι ένα λογισμικό που αναλαμβάνει την εφαρμογή του μοντέλου δεδομένων σε κάποιο από τα πρότυπα εισόδου. Ο μηχανισμός αυτός προσθέτει περιορισμούς στο μοντέλο δεδομένων, π.χ. στον τύπο των δεδομένων καθώς επίσης δηλώνει και το συντακτικό των προτύπων.
- **Το παραγόμενο έγγραφο:** Είναι ένα έγγραφο που ο τύπος του ποικίλει από εφαρμογή σε εφαρμογή και είναι η έξοδος του συστήματος.

Η βασική αρχή λειτουργίας φαίνεται στο παρακάτω σχέδιο



Εικόνα 4.4 Δομή ενός Template Engine

Τα Template Engines χρησιμοποιούνται για διαφόρους σκοπούς ανάλογα με την εφαρμογή και την λειτουργικότητα τους.

Μερικές από τις εφαρμογές τους παραθέτονται παρακάτω:

- **Παραγωγή εγγράφων κειμένου:** Μια τυπική χρήση αυτής της περίπτωσης είναι η αυτόματη παραγωγή email κειμένων όπου θέλουμε τα emails που θα παράξουμε να έχουν όλα το ίδιο περιεχόμενο εκτός από μερικά σημεία που είναι συγκεκριμένα για κάθε παραλήπτη. Σε αυτήν την περίπτωση μπορούμε να γράψουμε το κείμενο σε ένα πρότυπο και ανάλογα με τους μηχανισμούς που προσφέρει το template engine, να επεξεργαστούμε και να παράξουμε τα κομμάτια του κειμένου που αφορούν τον συγκεκριμένο χρήστη.
- **Παραγωγή πηγαίου κώδικα:** Αυτό το είδος χρήσης αποσκοπεί στην δημιουργία πηγαίου κώδικα από UML διαγράμματα ή από περιγραφή μιας σχεσιακής βάσης ή ακόμα και για αυτόματη παραγωγή σχολίων ή javadocs.
- **Λειτουργικότητα εφαρμογής:** Σε αυτήν την περίπτωση χρησιμοποιούμε το template engine κυρίως για να παράξουμε κομμάτια html σελίδων όπου τα δεδομένα έρχονται από την βάση ή από ένα XML αρχείο και τα εφαρμόζουμε σε ένα κοινό πρότυπο. Παράδειγμα σε αυτήν την περίπτωση είναι το XSLT.

4.9.1 Διάφορες μηχανές αρχέτυπων

Μια ευρέως χρησιμοποιούμενη template engine είναι η XSLT, η οποία σχεδιάστηκε από το W3C με σκοπό την μετατροπή XML δεδομένων σε σελίδες HTML. Άλλα γνωστά template engines είναι τα Java Server Pages (jsp), Active Server Pages(asp) καθώς επίσης υπάρχουν και πολλά αυτόνομα template engines για όλες σχεδόν τις γλώσσες προγραμματισμού.

Τα πλεονεκτήματα από την χρήση κάποιου template engine στην ανάπτυξη λογισμικού είναι πολλά, μερικά από αυτά είναι:

- Διαχωρίζει τον κώδικα της εφαρμογής σε επίπεδα ανάλογα με την λειτουργικότητα (MVC)
- Μειώνει τη συγγραφή παρόμοιου κώδικα, ελαχιστοποιώντας έτσι τα λάθη και κάνει των κώδικα πιο συμπαγή
- Διαχωρίζει την εμφάνιση από την λειτουργικότητα και έτσι διάφορα άτομα ανάλογα με την ικανότητά τους μπορούν να δουλέψουν σε ένα μόνο κομμάτι του κώδικα. Για παράδειγμα ένας web designer μπορεί να ασχολείται μόνο με το πρότυπο χωρίς να έχει γνώση για την λειτουργικότητα για να έχει καλύτερη

εμφάνιση μια σελίδα.

4.9.2 Το προϊόν Velocity Engine

Το Velocity Engine είναι μια ανοιχτού κώδικα βιβλιοθήκη για Java από την Apache. Είναι ένα ισχυρό και με πολλά χαρακτηριστικά Template Engine. Η κύρια χρήση του είναι για την παραγωγή html σελίδων. Μπορεί να χρησιμοποιηθεί και για να αντικαταστήσει πλήρως την JSP. Τα πλεονεκτήματα από την χρήση Velocity για την παραγωγή html σελίδων είναι:

- **Απλότητα:** Οι σελίδες μπορούν να γραφτούν και να συντηρηθούν από μη τεχνικά άτομα με βασικές γνώσεις html. Δεν χρειάζεται δηλαδή να γνωρίζουν την υλοποίηση της λειτουργικότητας της εφαρμογής και να ασχολούνται μόνο με την εμφάνιση της.
- **Ευκολία στην συντήρηση:** Προσφέρει ένα κατανοητό μοντέλο λειτουργίας
- **Πρόσβαση και σε μεθόδους και ιδιότητες αντικειμένων:** Με αυτόν τον τρόπο ο προγραμματιστής αποκτά μεγαλύτερη ευελιξία στην συγγραφή των προτύπων.

4.9.2.1 Αρχιτεκτονική Velocity Engine

Η αρχιτεκτονική του Velocity Engine ακολουθεί την αρχιτεκτονική των γενικών Template Engines που παρουσιάσαμε παραπάνω. Έχει όμως τις παρακάτω διαφοροποιήσεις:

- Το μοντέλο δεδομένων είναι μια συλλογή από αντικείμενα Java, όπου το καθένα έχει ένα μοναδικό όνομα. Δηλαδή το μοντέλο δεδομένων είναι ένα Map<String, Object>, το κλειδί είναι τύπου String και το αντικείμενο είναι ένα οποιοδήποτε αντικείμενο της Java.
- Το συντακτικό των templates είναι ένα απλό κείμενο. Οι εντολές του Velocity ξεκινάνε με # και η μεταβλητές με $. Αυτό κάνει τα πρότυπα πολύ πιο ευανάγνωστα σε σχέση με το xslt ή τα jstl taglibs.
- Προσφέρεται ένα πολύ απλό Java API για την συνένωση του μοντέλου εισόδου με το πρότυπο.
- Προσφέρονται πολλές έτοιμες εντολές όπως εντολή έλεγχου, διακλάδωσης, βρόγχοι επανάληψης και επίσης δίνεται η δυνατότητα να επεκταθεί ο

μηχανισμός με την χρήση μάκρο-εντολών.

- Προσφέρεται η δυνατότατα για καταγραφή σφαλμάτων κατά τη διάρκεια της συγχώνευσης του μοντέλου δεδομένων με τα πρότυπα για πιο εύκολη αποσφαλμάτωση.

4.9.2.2 Παράδειγμα

Ένα παράδειγμα ενός προτύπου είναι το παρακάτω:

```
<html>
<body>

    #if( $user.isLoggedIn() )
        #set( $message = "hello $user.getName()" )
    #else
        #set( $message = "Please Login" )
    #end

    $message

</body>
</html>
```

Στο παραπάνω παράδειγμα έχουμε την μεταβλητή “user” η οποία είναι ένα αντικείμενο της Java που προσφέρει μια μέθοδο isLoggedIn() τύπου Boolean και μια μέθοδο getName() που επιστρέφει ένα String. Το αποτέλεσμα του παραπάνω είναι προφανές.

Στην διπλωματική εργασία χρησιμοποιήθηκε το Velocity Engine για να παράξουμε emails που θα στέλνονται από την Νέδα στους χρήστες. Παραδείγματα και λεπτομέρειες της υλοποίησης παρουσιάζονται σε επόμενο κεφάλαιο.

4.10 Εξυπηρετητές Εφαρμογών

Στο παρόν εδάφιο θα αναφερθούμε μόνο στον Tomcat ο οποίος είναι ένας εξυπηρετητής εφαρμογών (application server) που αποσκοπεί στην εξυπηρέτηση διαδικτυακών εφαρμογών γραμμένων σε γλώσσα Java. Ο tomcat [1] είναι ένας ανοιχτού κώδικα εξυπηρετητής σελίδων (web server) και ένας Servlet Container.

Ουσιαστικά είναι μια υλοποίηση του Java Servlet και Java Server Pages API. Είναι γραμμένο εξ ολοκλήρου σε Java. Ο tomcat είναι υπεύθυνος να διαχειρίζεται τον κύκλο ζωής μιας διαδικτυακής εφαρμογής. Το τυπικό σενάριο περιλαμβάνει έναν tomcat που τρέχει πολλές διαδικτυακές εφαρμογές. Η διαχείριση μιας web εφαρμογής μπορεί να γίνει με το γραφικό περιβάλλον που προσφέρει.

Κεφάλαιο 5 Υλοποίηση υποσυστημάτων και λειτουργιών

Σε αυτό το κεφάλαιο αναφέρονται τα υποσυστήματα και ο τρόπος λειτουργίας τους που υλοποιήθηκαν για τους σκοπούς της διπλωματικής εργασίας. Όπως αναφέρθηκε και παραπάνω η Νέδα είναι ένα προϊόν που χρησιμοποιείται ήδη στο Ι.Τ.Υ.Ε. και η συνεισφορά της παρούσας διπλωματικής εργασίας ήταν κυρίως στα παρακάτω συστήματα και διαδικασίες

1. Μετάβαση σε maven
2. Καταγραφή ιστορικού χρηστών
3. Χρονοπρογραμματιστής
4. Μηχανισμός αποστολής email

5.1 Εισαγωγή αργιών στο σύστημα

Μέχρι τώρα οι αργίες ενός έτους δηλωνόταν στο σύστημα χρησιμοποιώντας μία static μεταβλητή στον κώδικα. Αυτή η μεθοδολογία απαιτεί να δημιουργούνται καινούριες εκδώσεις της Νέδας κάθε φόρα που θέλουμε να προσθέσουμε μια αργία. Στόχος ήταν να παρέχουμε έναν μηχανισμό έτσι ώστε ο διαχειριστής του συστήματος να έχει την δυνατότητα να εισάγει αργίες για κάθε χρόνο.

Η υλοποίηση της λειτουργίας αυτής είχε σαν στόχο την εξοικείωση με το σύστημα και την κατανόηση του επιπέδου αποθήκευσης της εφαρμογής. Ένα στιγμιότυπο της σελίδας καταχώρισης αργιών φαίνεται στην Εικόνα 3.7 Στιγμιότυπο της εφαρμογής για εισαγωγή αργιών).

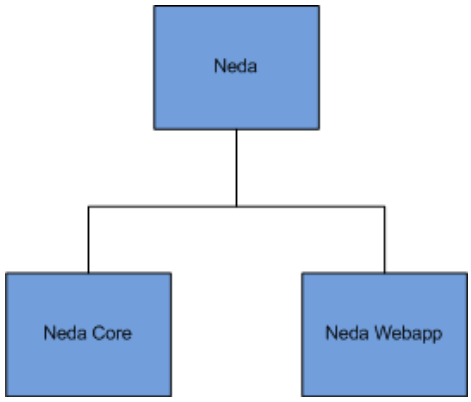
5.2 Μετάβαση σε maven

Για την διευκόλυνση της διαδικασίας ανάπτυξης και συντήρησης του συστήματος Νέδα χρειάστηκε να γίνει η μετάβαση από Ant σε Maven για το κτίσιμο του κώδικα. Η διαδικασία συμπεριλάμβανε την μετατροπή των ant scripts σε maven pom αρχεία.

Ο κώδικας χωρίστηκε σε δύο υποσυστήματα

- **Neda core:** Σε αυτό το maven module βρίσκονται όλες οι κλάσεις που αφορούν το επίπεδο αποθήκευσης και οι κλάσεις που αφορούν το μοντέλο του συστήματος μας. Το παραγόμενο αντικείμενο του module αυτού είναι μια βιβλιοθήκη jar.
- **Neda webapp:** Σε αυτό το module βρίσκονται όλες οι κλάσεις για το Spring MVC, τα αρχεία jsp, html καθώς επίσης ότι αφορά στην web εφαρμογή μας όπως τα servlets, τα φίλτρα και άλλα. Το παραγόμενο αντικείμενο αυτού του module είναι ένα .war αρχείο έτοιμο για εγκατάσταση σε έναν application server.

Το neda webapp έχει εξάρτηση στο core module. Τα δύο αυτά υποσυστήματα μπορούν να χτιστούν ανεξάρτητα. Υπάρχει βέβαια και το γονικό τους module. Μέσα από το γονικό αυτό module μπορούμε να χτίσουμε ή να τρέξουμε οποιαδήποτε φάση του maven και για τα δύο υπο-modules.



Εικόνα 5.1 Ιεραρχία των modules στη Νέδα

Ο παραπάνω διαχωρισμός βοηθάει στο να κάνουμε το σύστημά μας πιο διαχωρίσιμο? σε λειτουργικές ενότητες.

Χρησιμοποιούνται επίσης διάφορα maven plugins για έλεγχο της ποιότητας του κώδικα

παράγοντας κάποιες μετρικές. Μερικά από αυτά είναι:

- **Checkstyle:** Παράγει μια αναφορά σχετικά με το styling του κώδικα που έχουμε γράψει.
- **Cobertura:** Παράγει μια αναφορά σχετικά με το πόσο καλά καλύπτουν των κώδικα μας τα unit tests που έχουν γραφτεί.
- **Findbugs:** Παράγει μια αναφορά σχετικά με τα πιθανά bugs του συστήματος. Οι τεχνικές που χρησιμοποιεί είναι κυρίως ευρετικές.
- **Pmd:** Παράγει μια αναφορά για τον κώδικα του συστήματος βασισμένη σε ένα συγκεκριμένο σύνολο κανόνων.

5.3 Χρονοπρογραμματιστής

Ο χρονοπρογραμματιστής που χρησιμοποιείται στην Νέδα είναι ο Quartz. Για την διευκόλυνση μας ο χρονοπρογραμματιστής έχει οριστεί ως ένα Spring bean ως εξής.

```
    <bean
class="org.springframework.scheduling.quartz.SchedulerFactoryBean"
          lazy-init="false">
        <property name="autoStartup" value="true"/>
        <property name="triggers">
            <list>
                <ref bean="removeOldEventLogsTrigger"/>
                <ref bean="addHolidaysReminderTrigger"/>
                <ref bean="updateStatisticsTrigger"/>
                <ref bean="addPlanTimesheetReminderTrigger"/>
                <ref bean="fillTimesheetReminderTrigger"/>
                <ref bean="updateProjectStatisticsTrigger"/>
                <ref bean="updateTravelStatisticsTrigger"/>
                <ref bean="secondCheckFillTimesheetReminderTrigger"/>
            </list>
        </property>

        <property name="schedulerContextAsMap">
            <map>
                <entry key="removeEventsAfter"
value="${prman.removeEventsAfter}"/>
                <entry key="eventLogManager" value-
ref="eventLogManager"/>
                <entry key="emailManager" value-ref="emailManager"/>
                <entry key="emailManager2" value-
ref="emailManagerTravel"/>
                <entry key="emailMessageFactory" value-
ref="emailMessageFactory"/>
```

```
                <entry key="userManager" value-ref="userManager"/>
                <entry key="timesheetManager" value-
ref="timesheetManager"/>
                <entry key="tmStatsManager" value-
ref="timesheetStatisticsManager"/>
                <entry key="userStatsManager" value-
ref="userStatisticsManager"/>
                <entry key="projectManager" value-
ref="projectManager"/>
                <entry key="prStatsManager" value-
ref="projectStatisticsManager"/>
                <entry key="travelManager" value-ref="travelManager"/>
                <entry key="trGlobalStatsManager" value-
ref="travelGlobalStatisticsManager"/>
                <entry key="researcherManager" value-
ref="researcherManager"/>
                <entry key="userManager" value-ref="userManager"/>
                <entry key="sessionFactory" value-
ref="sessionFactory"/>
            </map>
        </property>
        <property name="quartzProperties">
            <props>
                <prop key="org.quartz.scheduler.instanceName">neda
Scheduler</prop>
                <prop key="org.quartz.scheduler.instanceId">AUTO</prop>
                <prop
key="org.quartz.threadPool.class">org.quartz.simpl.SimpleThreadPool</pr
op>
            </props>
        </property>
    </bean>
```

Στο παραπάνω bean οι σημαντικότερες ιδιότητες είναι η λίστα με τα triggers καθώς και τα beans που θα συμπεριλαμβάνει το Scheduler Context.

Ένα trigger ορίζεται και αυτό ως ένα bean πχ:

```
    <bean id="addHolidaysReminderTrigger"
class="org.springframework.scheduling.quartz.CronTriggerBean">
        <property name="jobDetail" ref="addHolidaysReminderJob"/>
        <!--fire every year on 26th of December at 07:00 am-->
        <property name="cronExpression" value="0 0 7 26 DEC ? "/>
    </bean>
```

Σε κάθε trigger ορίζεται η κλάση της εργασίας που θα τρέξει και επίσης μια έκφραση Cron που δηλώνει κάθε πότε θα τρέχει η εργασία. Στο παραπάνω παράδειγμα η εργασία θα εκτελείται κάθε 26 Δεκεμβρίου στις 7:00 π.μ.

Η κλάση της εργασίας που θέλουμε να εκτελεστεί είναι δηλωμένη και αυτή ως ένα

bean, για το παραπάνω παράδειγμα είναι:

```
    <bean id="addHolidaysReminderJob"
class="org.springframework.scheduling.quartz.JobDetailBean">
        <property name="jobClass"
value="prman.utils.jobs.AddHolidaysReminderJob"/>
    </bean>
```

Η κλάση της διεργασίας θα πρέπει να είναι υπερκλάση της org.springframework.scheduling.quartz.QuartzJobBean. Η μέθοδος που θα πρέπει να υλοποιηθεί είναι η:

```
protected abstract void executeInternal(JobExecutionContext context)
throws JobExecutionException;
```

Οποιαδήποτε παράμετρο χρειάζεται η εργασία θα πρέπει να περνιέται μέσω της παραμέτρου context.

5.4 Μηχανισμός αποστολής email

Η απαίτηση για αυτό το υποσύστημα της Νέδα είναι να παρέχουμε έναν μηχανισμό για αποστολή email από το σύστημα στους χρήστες. Για την υλοποίηση του επίπεδου επικοινωνίας με τον mail server χρησιμοποιούμε το έτοιμο πακέτο της Java javax.mail και javax.activation. Οι κλάσεις και το API που προσφέρουν αυτά τα πακέτα δεν είναι πολύ βολικά και απαιτούν την συγγραφή όμοιου κώδικα σε κάθε σημείο που θέλουμε να στείλουμε ένα email. Για αυτόν τον λόγο ήταν αναγκαία η υλοποίηση μιας κλάσης και ενός απλού Interface έτσι ώστε να διευκολυνθεί η διαδικασία της αποστολής των emails.

Βασική έννοια στο υποσύστημα που υλοποιήθηκε είναι η κλάση EmailMessage. Η κλάση αυτή περιέχει όλες τις πληροφορίες που χρειαζόμαστε όπως τους παραλήπτες, το θέμα, το κείμενο, τυχόν επισυναπτόμενα αρχεία και άλλα. Το interface αυτής της κλάσης φαίνεται παρακάτω:

```
package prman.utils.notifications.email;

import java.util.List;
import java.util.Map;
```

```
/**
 * This class maps a single email message, witch will be used by
EmailManager.
 */
public final class EmailMessage {

    /**
     * A list with the recipients of this email.
     */
    private List<String> recipients;

    /**
     * The subject of this email.
     */
    private String subject;

    /**
     * The text type of this email.
     */
    private String textType;

    /**
     * The content of this email.
     */
    private String content;

    /**
     * Flag used to indicate if the content of this email will be
processed by a template engine.
     */
    private boolean templateEnabled;

    /**
     * The template to be used.
     */
    private String template;

    /**
     * The bindings to be used when processing the template.
     */
    private Map<String, Object> bindings;

    /**
     * A list containing the attachments of this email.
     */
    private List<EmailAttachment> attachments;

       //Getters and setters….
}
```

Επίσης παρέχεται ένα πολύ απλό Interface που αναλαμβάνει την αποστολή του μηνύματος:

```
package prman.utils.notifications.email;

/**
 * Interface providing methods for sending emails.
 */
public interface EmailManager {

    /**
     * This method sends an email
     *
     * @param emailMessage the email message to be send
     * @throws EmailDeliveryException if any exception occurred.
     */
    public void sendEmail(EmailMessage emailMessage) throws
EmailDeliveryException;
}
```

Η υλοποίηση του παραπάνω interface είναι δηλωμένο ως ένα bean στο application context και έτσι κάθε άλλο υποσύστημα που θέλει να αποστείλει emails αρκεί να έχει ένα reference σε αυτό το bean. Η δήλωση του bean είναι η παρακάτω:

```
    <bean id="emailManager"
class="prman.utils.notifications.email.EmailManagerImpl">
        <property name="smtpAuth" value="${prman.smtp.auth}"/>
        <property name="smtpFrom" value="${prman.smtp.from}"/>
        <property name="smtpHost" value="${prman.smtp.host}"/>
        <property name="smtpPassword" value="${prman.smtp.password}"/>
        <property name="smtpPort" value="${prman.smtp.port}"/>
        <property name="smtpProtocol" value="${prman.smtp.protocol}"/>
        <property name="smtpUsername" value="${prman.smtp.username}"/>
        <property name="templateEngine" ref="velocityTemplateEngine"/>
    </bean>
```

Από την κλάση EmailMessage μια ιδιότητα που αξίζει αναφορά είναι το templateEnabled. Όταν αυτό το flag είναι ενεργοποιημένο τότε ο EmailManager αναλαμβάνει να δημιουργήσει το τελικό κείμενο βάση του template και των bindings που του παρέχονται χρησιμοποιώντας το Velocity Template Engine. Τα templates των emails βρίσκονται όλα σε ένα συγκεκριμένο φάκελο, έχουν ένα μοναδικό αναγνωριστικό. Επίσης υπάρχει η κλάση EmailFactory η οποία αναλαμβάνει να δημιουργήσει ένα νέο instance EmailMessage με τις κατάλληλες ιδιότητες. Η κλάση EmailFactory θα αναζητήσει στο application context του Spring, beans που είναι τύπου EmailMessage και θα επιστρέψει ένα νέο στιγμιότυπο τους. Με αυτόν τον τρόπο κάθε πρότυπο email δηλώνεται μόνο μια φορά, γλιτώνοντάς μας την συγγραφή παρόμοιου

κώδικα σε πολλά σημεία.

Τα emails που βασίζονται σε κάποιο πρότυπο επεξεργάζονται εσωτερικά από τον EmailManager. Όπως φαίνεται και παραπάνω στην δήλωση του bean του EmailManager θέτουμε σαν ιδιότητα του ένα TemplateEngine. Το TemplateEngine είναι ένα interface που παρέχει μία μέθοδο όπως φαίνεται παρακάτω:

```
package prman.utils.notifications.email;

import java.io.Writer;
import java.util.Map;

/**
 * Interface for the Template Engines.
 */
public interface TemplateEngine {

    /**
     * Implementation of this method are expected to process a template
based on some bindings
     * and write the result in a Writer
     *
     * @param bindings         a map containing the bindings
     * @param templateFileName the template to be used
     * @param outputWriter     a writer in which the result will be
written
     * @throws TemplateEngineException if any exception occurred during
processing.
     */
    void processTemplate(Map<String, Object> bindings, String
templateFileName, Writer outputWriter)
            throws TemplateEngineException;

}
```

Στο σύστημα υλοποιούμε το παραπάνω interface στην κλάση VelocityTemplateEngine. Η κλάση αυτή χρησιμοποίει το VelocityEngine για να παράγει το τελικό κείμενο από το πρότυπο. Αργότερα μπορούν να προστεθούν και άλλοι μηχανισμοί συγχώνευσης προτύπων. Αξίζει να σημειωθεί πως στο VelocityTemplateEngine μπορούμε να ορίσουμε κάποια bindings καθολικής εμβέλειας. Αυτά τα bindings είναι ιδιότητες που χρησιμοποιούνται σε όλα τα emails. Η δυνατότητα δήλωσης καθολικών bindings προστέθηκε για να αποφύγουμε την εισαγωγή κάποιον ιδιοτήτων σε κάθε email. Στην παρούσα φάση σαν καθολικό binding έχει οριστεί μόνο το url της Νέδας. Στο παράδειγμα που ακολουθεί στην παρακάτω ενότητα φάνεται και η χρήση αυτού του binding.

5.4.1 Παράδειγμα

Στο παράδειγμα αυτό θα αναφερθούμε στην υλοποίηση της αποστολής ενός email για την υπενθύμιση του διαχειριστή να εισάγει της αργίες μιας χρονιάς.

Πρώτο βήμα είναι να δημιουργήσουμε ένα velocity template που θα περιέχει το κείμενο του email όπως φαίνεται παρακάτω:

```
<html>
<body>
Don't forget to fill holidays for the year: ${year}. Follow the bellow
link to fill holidays:
<br/>
<a href="${neda_public_url}/holidays.htm?year=${year}">Fill holidays
for ${year}</a>
</body>
</html>
```

Το παραπάνω template το αποθηκεύουμε στο package templates καθώς σε αυτό το πακέτο θα αναζητήσει τα πρότυπα το TemplateEngine που υλοποιήθηκε.

Δεύτερο βήμα είναι να δηλώσουμε ένα bean στο application context ως εξής:

```
    <bean id="addHolidaysNotificationMail"
class="prman.utils.notifications.email.EmailMessage"
          scope="prototype">
        <property name="template"
value="templates/addHolidaysReminderTemplate.vm"/>
        <property name="templateEnabled" value="true"/>
        <property name="subject" value="Reminder to add holidays"/>
    </bean>
```

Παρατηρούμε ότι το scope αυτού του bean είναι prototype και αυτό γιατί θέλουμε ο EmailFactory να επιστρέφει ένα καινούριο instance κάθε φορά που καλείται.

Τελευταίο βήμα είναι στο σημείο που θέλουμε να στείλουμε το email να γράψουμε τα παρακάτω:

```
            //Get the email template from the factory
            final EmailMessage emailMessage =
emailMessageFactory.newInstance("addHolidaysNotificationMail");
            final List<String> recipients = new ArrayList<String>();

            //Get all the users with head administration roles
            final List<User> headAdmins =
userManager.searchByPost("Head Administrator");
            for (User headAdmin : headAdmins) {
                String[] email =
```

```
headAdmin.getNotificationEmailsCommaDelimited();
                for (int i = 0; i < email.length; i++) {
                    recipients.add(email[i]);
                }
//                  recipients.add(headAdmin.getEmail());
            }
            //Set the recipients
            emailMessage.setRecipients(recipients);
            final Map<String, Object> bindings = new HashMap<String,
Object>();

            //Make the year available in email template
            bindings.put("year",
String.valueOf(Calendar.getInstance().get(Calendar.YEAR) + 1));
            emailMessage.setBindings(bindings);

            emailManager.sendEmail(emailMessage);
```

Σε αυτό το παράδειγμα παρατηρούμε ότι στα bindings πρέπει να εισάγουμε το έτος έτσι ώστε να μπορέσει να αντικατασταθεί στο template έτσι όπως ορίστηκε παραπάνω.

Ουσιαστικά ο προγραμματιστής πλέον χρειάζεται μόνο να γράψει ένα πρότυπο του email χρησιμοποιώντας την σύνταξη του Velocity Engine και να ορίσει τους παραδεκτές του email.

5.5 Ιστορικό Χρηστών

Βασική απαίτηση σε αυτήν την λειτουργία είναι να καταγράψουμε κάθε ενέργεια που κάνει κάποιος χρήστης. Ένας τρόπος να προσεγγίσουμε το πρόβλημα θα ήταν ο εξής: σε κάθε σημείο του κώδικα όπου έχουμε τον χειρισμό μιας ενέργειας του χρήστη, να προσθέσουμε μια μέθοδο ή ένα κομμάτι κώδικα για την καταγραφή της ενέργειας. Η παραπάνω προσέγγιση αφορά το επίπεδο διεπαφής του συστήματος μας. Η παραπάνω λύση δεν είναι αποδοτική και έχει πολλά μειονεκτήματα, μερικά από αυτά είναι:

- Αλλαγή μεγάλου μέρους του κώδικα της εφαρμογής έτσι ώστε να ενεργοποιήσουμε την καταγραφή των ενεργειών των χρηστών
- Είναι χρονοβόρο και δύσκολο να εντοπίσουμε όλα τα σημεία στον κώδικα όπου υπάρχει χειρισμός μιας ενέργειας του χρήστη
- Νέες ενέργειες προστίθενται συνεχώς στο σύστημα και ο προγραμματιστής θα πρέπει να έχει υπόψη του την υλοποίηση και την καταγραφή της, αυτό οδηγεί

σε παραλήψεις και λάθη.

- Θα πρέπει να ενσωματωθεί κώδικας για την καταγραφή της ενέργειας μέσα στην εκάστοτε μέθοδο και έτσι καταλήγουμε να έχουμε δυο διαφορετικά υποσυστήματα, του χειρισμού της ενέργειας και της καταγραφής της, μέσα στην ίδια μέθοδο. Αυτό κάνει τον κώδικα μας να μην είναι ξεκάθαρος και γίνεται επιρρεπής σε σφάλματα.

Μια άλλη πιο κομψή προσέγγιση του προβλήματος θα ήταν αντί για το επίπεδο της παρουσίασης να κάναμε την καταγραφή στο επίπεδο αποθήκευσης και αυτό γιατί κάθε ενέργεια που μπορεί να κάνει κάποιος χρήστης στο σύστημα έχει σαν αποτέλεσμα την εισαγωγή, διαγραφή ή αλλαγή κάποιας οντότητας στην βάση. Με αυτήν την προσέγγιση μειώνουμε αρκετά τις αλλαγές που χρειάζεται ο κώδικας, καθώς πλέον θα πρέπει να εισάγουμε την λογική της καταγραφής μόνο στο επίπεδο αποθήκευσης, όπου οι κλάσεις είναι λίγες. Και αυτή όμως η προσέγγιση δεν δίνει λύση για τα τελευταία δύο μειονεκτήματα που αναφέρονται παραπάνω.

Η λύση που προτάθηκε και υλοποιήθηκε είναι η δημιουργία ενός υποσυστήματος που χρησιμοποιεί μεθοδολογία AOP για να λύσει όλα τα παραπάνω προβλήματα. Έτσι υλοποιήθηκε το σύστημα καταγραφής ενεργειών χρηστών χωρίς να χρειάζεται να κάνουμε καμία αλλαγή στον ήδη υπάρχον κώδικα.

5.5.1 Περιγραφή της υλοποίησης του επιπέδου αποθήκευσης στην Νέδα

Όπως αναφέρθηκε και παραπάνω το επίπεδο αποθήκευσης της Νέδα έχει υλοποιηθεί χρησιμοποιώντας το περιβάλλον Hibernate. Επίσης κάθε οντότητα του συστήματος έχει την αντίστοιχη κλάση (manager) που διαχειρίζεται τις εισαγωγές, διαγραφές και τροποποιήσεις της οντότητας αυτής. Όλοι αυτοί οι managers είναι δηλωμένοι σαν Spring Beans και είναι όλοι υπερκλάσεις της αφηρημένης κλάσης AbstractManager. Η κλάση αυτή παρέχει τις παρακάτω μεθόδους, παραθέτουμε μόνο την υπογραφή τους:

```
    public E getByID(final E abstrObject, final int entityID);

    public List<E> searchByName(final String name, final E
abstrObject);

    List<E> list(final E abstrObject);

    public  void add(final E abstrObject) ;
```

```
    public  void update(final E abstrObject);

    public  void delete(final E abstrObject, final int entityID);
```

5.5.2 Υλοποίηση με χρήση της μεθοδολογίας AOP

Για να καταγράψουμε κάθε ενέργεια χρειάζεται να καταφέρουμε να ενσωματώσουμε στις μεθόδους, add, delete και update την λογική της καταγραφής της ενέργειας. Η μεθοδολογία AOP μας προσφέρει έναν πολύ κομψό τρόπο για να μπορέσουμε να διακόψουμε την ροή της εκτέλεσης του κώδικα κάθε φορά που εκτελείται μια από αυτές τις μεθόδους και να καταγράψουμε στο ιστορικό την συγκεκριμένη ενέργεια. Η βιβλιοθήκη που χρησιμοποιήσαμε είναι η AspectJ [14] η οποία είναι ανοιχτού κώδικα.

Θα πρέπει να ορίσουμε στο σύστημα μας ένα Ascpect τέτοιο ώστε να μπορεί να διακοπεί η ροή του κώδικα στις παραπάνω μεθόδους. Το Aspect αυτό βρίσκεται στην κλάση DataBaseEventLogger. Η κλάση αύτη παρατίθεται παρακάτω:

```
@Aspect
public final class DataBaseEventLogger {

    /**
     * Logger for debugging messages.
     */
    private static final Logger LOGGER =
Logger.getLogger(DataBaseEventLogger.class);

    /**
     * Define the add pointcut.
     */
    @Pointcut("execution(*
prman.data.manager.AbstractManager.add(Object))")
    public void added() {
        // Empty method
    }

    /**
     * Define the delete pointcut.
     */
    @Pointcut("execution(*
prman.data.manager.AbstractManager.delete(Object,int))")
    public void deleted() {
        // Empty method
    }

    /**
     * Define the update pointcut.
     */
    @Pointcut("execution(*
```

```
prman.data.manager.AbstractManager.update(Object))")
    public void updated() {
        // Empty method
    }

    /**
     * this is the main method that intercepts all the above pointcuts.
     * <p/>
     *
     * @param pjp the JointPoint to proceed with.
     * @return the actual Object that would be returned by the method
     * @throws Throwable thrown when an exception occurs
     */
    @Around("added() || deleted() || updated()")
    public Object invoke(final ProceedingJoinPoint pjp) throws
Throwable {
        final Object result = pjp.proceed();
        try {
            doLogging(pjp);
        } catch (Throwable t) {
            LOGGER.error("Error while trying to get event Log metadata,
proceeding with normal execution without logging!", t);
        }
        return result;
    }
```

Στο παραπάνω βλέπουμε πως η κλάση έχει οριστεί σαν Aspect χρησιμοποιώντας το annotation @Aspect. Επίσης όπως φαίνεται έχουμε ορίσει και τρία Pointcuts στις μεθόδους update, delete και add της κλάσης AbstractManager. Τα pointcuts αυτά ορίστηκαν χρησιμοποιώντας το annotation @Pointcut. Η παράμετρος αυτού του annotation είναι μια έκφραση του AspectJ. Στην έκφραση αυτή μπορούμε να περιγράψουμε σε ποιες μεθόδους θέλουμε να παρέμβουμε στην ροή εκτέλεσης του προγράμματος.

Επόμενο βήμα είναι να περιγράψουμε πώς θα χειριστούμε τα παραπάνω τρία pointcuts. Εδώ από το AspectJ παρέχονται τρεις τρόποι, Να εκτελέσουμε κώδικα πριν την μέθοδο, μετά την μέθοδο ή να χειριστούμε εμείς όλη την εκτέλεση της μεθόδου. Ο τελευταίος αυτός τρόπος επιλέχτηκε και για την υλοποίηση μας.

Με το annotation @Around δηλώνουμε μια μέθοδο που θα αναλάβει την καταγραφή της ενέργειας.

Στο παραπάνω κομμάτι κώδικα αξίζει να σημειώσουμε ότι με το

```
final Object result = pjp.proceed();
```

καλείται η μέθοδος add, update ή delete ενώ η μέθοδος doLogging() είναι αυτή που

αναλαμβάνει την καταγραφή της ενέργειας.

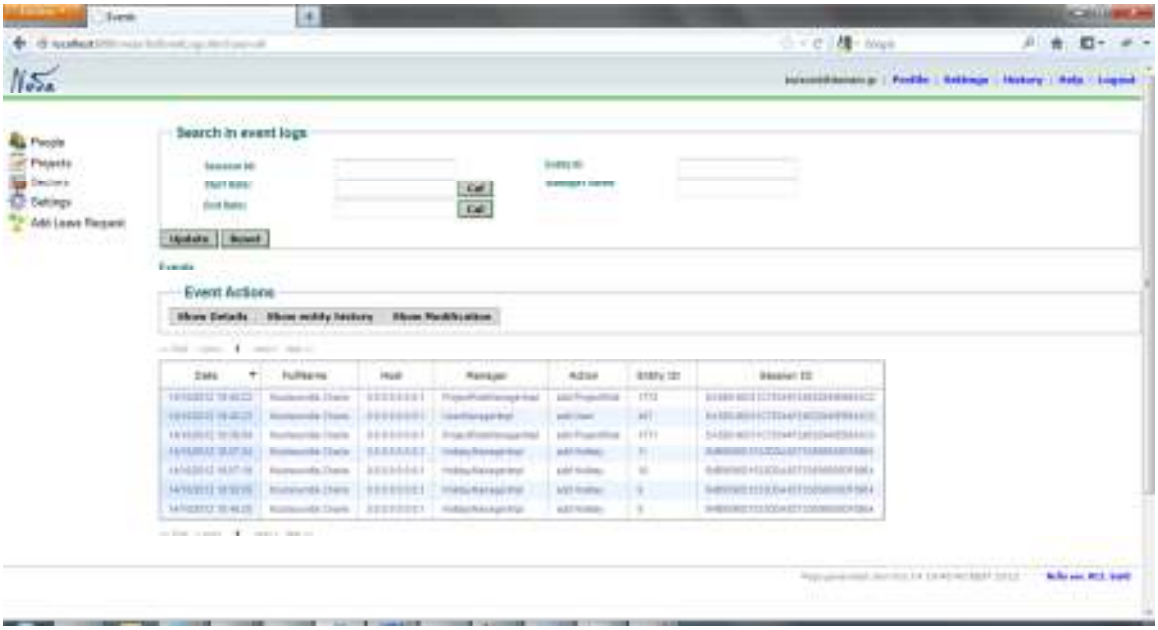
5.5.3 Αποθήκευση ιστορικού των ενεργειών.

Όλες οι ενέργειες που καταγράφονται αποθηκεύονται σε έναν πίνακα στην βάση. Οι πληροφορίες που αποθηκεύονται στην βάση είναι η ημερομηνία, ο χρήστης, ή ενέργεια (πρόσθεση/διαγραφή/επεξεργασίαα οντότητας) καθώς και το είδος της οντότητας. Εκτός από τις παραπάνω πληροφορίες αποθηκεύουμε επίσης στον δίσκο ένα στιγμιότυπο της οντότητας. Κάθε οντότητα αναπαριστάται από ένα αντικείμενο της Java, αυτό που αποθηκεύεται στον δίσκο είναι η JSON αναπαράσταση του αντικειμένου. Με αυτόν τον τρόπο αποθηκεύεται και ένα πλήρες ιστορικό των οντοτήτων του συστήματος και μπορούν εύκολα να εντοπιστούν αλλαγές. Για την μετατροπή από αντικείμενα σε JSON αρχεία χρησιμοποιήθηκε η βιβλιοθήκη Jackson Mapper.

5.5.4 Προβολή των ενεργειών

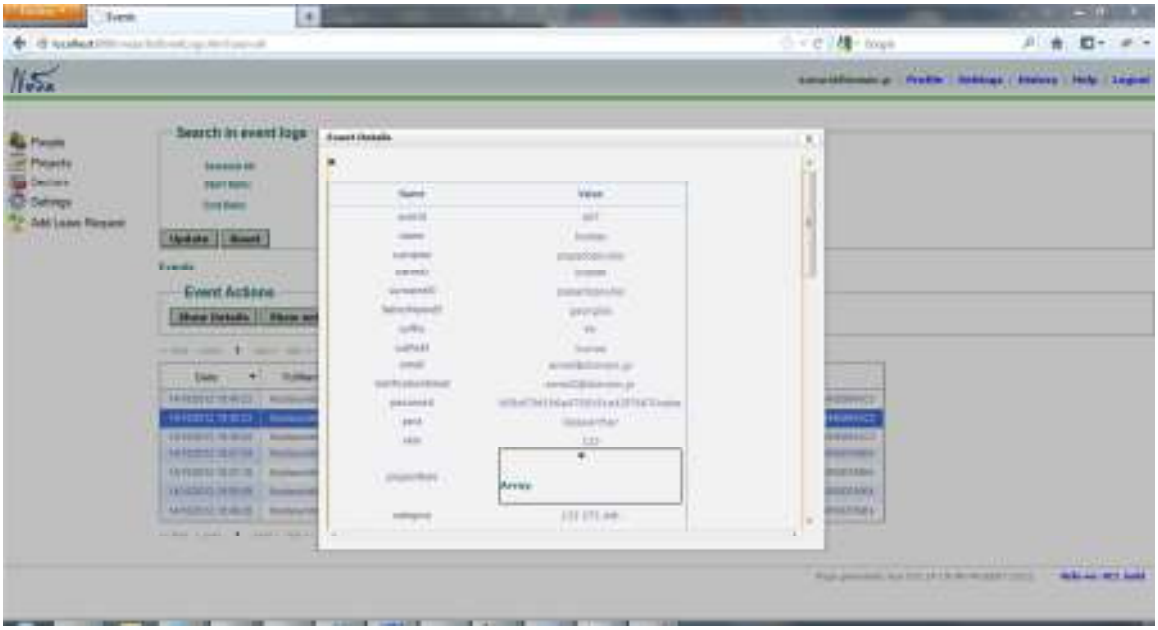
Η προβολή των ενεργειών που καταγράφηκαν είναι διαθέσιμη μόνο για τους διαχειριστές του συστήματος. Στην παρακάτω εικόνα βλέπουμε την οθόνη με τις ενέργειες των χρηστών. Σε αυτήν παρατηρούμε πως πάνω υπάρχει η φόρμα για αναζήτηση όπου μπορούμε να περιορίσουμε τα αποτελέσματα.

Στο κάτω μέρος βρίσκεται ο πίνακας με τις ενέργειες των χρηστών. Ο πίνακας αυτός προσφέρει σελιδοποίηση. Για την υλοποίησή του χρησιμοποιήθηκε η βιβλιοθήκη YUI [21] από την Yahoo. Η βιβλιοθήκη αυτή προσφέρει μια πληθώρα από έτοιμα εργαλεία γραμμένα σε javascript για διεπαφή χρήστη σε HTML. Το πλεονέκτημα της σελιδοποίησης που προσφέρει η εν λόγο βιβλιοθήκη είναι ότι μας δίνει την δυνατότητα η σελιδοποίηση να γίνεται στον εξυπηρετητή και όχι στον browser του χρήστη. Οι ενέργειες των χρηστών είναι πάρα πολλές στο σύστημα και είναι απαγορευτικό να τις στείλουμε όλες στον browser για να κάνει αυτός την σελιδοποίηση.



Εικόνα 5.2 Στιγιότυπο της εφαρμογής για την προβολή των γεγονότων

Μόλις επιλέξουμε μια εγγραφή από τον πίνακα, ενεργοποιούνται οι επιλογές για προβολή του ιστορικού της οντότητας που αφορά αυτή η εγγραφή και η επιλογή για προβολή λεπτομερειών της ενέργειας. Οι λεπτομέρειες της οντότητας είναι ουσιαστικά μια οπτική αναπαράσταση του αντικειμένου μετά την ενέργεια του χρήστη. Η οπτική αυτή αναπαράσταση προέρχεται από την JSON αναπαράσταση του αντικειμένου της οντότητας.



Εικόνα 5.3 Προβολή λεπτομερειών για ένα γεγονός

Κεφάλαιο 6 Συμπεράσματα και προοπτικές

6.1 Συμπεράσματα

Μετά την ολοκλήρωση της υλοποίησης της διπλωματικής εργασίας μπορούμε να προβούμε σε μερικά συμπεράσματα για την διαδικασία ανάπτυξης εφαρμογών και για τις τεχνικές δυσκολίες που συναντώνται κατά την υλοποίηση μιας εφαρμογής. Μπορούμε να πούμε πως η ανάπτυξη εφαρμογής που χρησιμοποιείται στην πραγματικότητα είναι μια πολύπλοκη διαδικασία στην οποία συμμετέχουν πολλοί άνθρωποι με διαφορετικούς ρόλους. Ένα από τα σημαντικότερα συστατικά της επιτυχίας υλοποίησης της εφαρμογής είναι η επικοινωνία μεταξύ των εμπλεκόμενων μελών και ο σωστός συντονισμός τους. Επίσης επειδή η διπλωματική εργασία βασίστηκε σε ένα ήδη υπάρχων σύστημα είδαμε πόσο σημαντική είναι η τεκμηρίωση του κώδικα καθώς και ο ξεκάθαρος σχεδιασμός της αρχιτεκτονικής του συστήματος. Γι αυτό το λόγο η εξοικείωση με το σύστημα και την αρχιτεκτονική του έγινε χωρίς ιδιαίτερη δυσκολία.

Το περιβάλλον εργασίας Spring μας δίνει τη δυνατότητα για την συγγραφή και υλοποίηση σύγχρονων συστημάτων προσφέροντάς μας εργαλεία που καλύπτουν τις ανάγκες μιας δικτυακής εφαρμογής σε όλους του τομείς. Επίσης συστήματα όπως η Νέδα που χρησιμοποιούνται από οργανισμούς χρειάζεται να εμπλουτίζονται συνέχεια με νέα χαρακτηριστικά γιατί οι οργανισμοί που τα χρησιμοποιούν αλλάζουν συνέχεια και μαζί τους και οι απαιτήσεις του συστήματος. Για αυτόν τον λόγο η χρήση εργαλείων όπως το Trac, το Hudson, το maven είναι απαραίτητα για την καλύτερη διαχείριση και οργάνωση της διαδικασίας ανάπτυξης της εφαρμογής.

Τέλος η γλώσσα Java και τα υπόλοιπα προϊόντα που χρησιμοποιήθηκαν διευκόλυναν κατά πολύ την διαδικασία ανάπτυξης των υποσυστημάτων που υλοποιήθηκαν, καθώς πολλές βιβλιοθήκες για Java προσφέρουν ένα πλούσιο σετ δυνατοτήτων και το κυριότερο, είναι ανοιχτού κώδικα και μπορούν να χρησιμοποιηθούν για κάθε σκοπό.

Η μεθοδολογία Aspect Oriented Programming είναι ιδανική για την υλοποίηση ενός γενικευμένου μηχανισμού καταγραφής ιστορικού των ενεργειών. Όπως είδαμε από τον τρόπο που υλοποιήθηκε δεν χρειάστηκαν αλλαγές σε κανένα υποσύστημα της Νέδα.

Επίσης ο μηχανισμός είναι σε θέση να καταγράφει κάθε ενέργεια, ακόμα και τις καινούριες που θα προστεθούν στο μέλλον χωρίς να χρειάζεται ο προγραμματιστής να ξέρει την ύπαρξη του συστήματος καταγραφής. Το σύστημα καταγραφής με έναν διαφανή τρόπο αναλαμβάνει την καταγραφή κάθε ενέργειας.

6.2 Προοπτικές

Από τα υποσυστήματα που υλοποιήθηκαν για τους σκοπούς της διπλωματικής εργασίας ανοίγουν οι ορίζοντες για την υλοποίηση μερικών καινούριων λειτουργιών στο σύστημα. Μερικές από της καινούριες λειτουργίες που μπορούν να ενσωματωθούν προσφέρουν επιπλέον λειτουργικότητα, ενώ άλλες διευκολύνουν την διαχείριση της εφαρμογής και της συντήρησής της από τους διαχειριστές.

6.2.1 Παρουσίαση πρόσφατων ενεργειών στους χρήστες

Το σύστημα καταγραφής ιστορικού των ενεργειών μπορεί να χρησιμοποιηθεί για την δημιουργία ενός καινούριου συστήματος dashboard. Το σύστημα αυτό θα δίνει την δυνατότητα σε κάθε χρήστη να βλέπει τις σημαντικότερες ενέργειες που έχουν κάνει οι υπόλοιποι χρήστες που ασχολούνται με το ίδιο έργο. Με αυτόν τον τρόπο κάθε φορά που συνδέεται κάποιος στην Νέδα θα μπορεί να βλέπει την πρόοδο των υπολοίπων χρηστών, με τι ασχολήθηκαν, πόσο ασχολήθηκαν και άλλες χρήσιμες πληροφορίες. Η παραπάνω λειτουργία είναι πολύ χρήσιμη για τους ίδιους τους χρήστες καθώς δίνει με έναν πολύ γρήγορο τρόπο μια συνολική εικόνα για το έργο. Οι διαχειριστές έργων από την άλλη θα μπορούν να βλέπουν τις ενέργειες που αφορούν τα έργα που είναι υπεύθυνοι. Η λειτουργία αυτή υπάρχει στα περισσότερα συστήματα διαχείρισης έργων και όχι μόνο. Ένα παράδειγμα είναι το github το οποίο δίνει στους χρήστες την δυνατότητα με το που συνδέονται να μπορούν να δούνε τις πιο πρόσφατες αλλαγές όλων των χρηστών που εμπλέκονται στο έργο το οποίο αναπτύσσουν. Κάτι παρόμοιο θα μπορούσε να υλοποιηθεί και για την Νέδα.

6.2.2 Βελτιώσεις στο σύστημα αποστολής ηλεκτρονικών μηνυμάτων

Όπως είδαμε παραπάνω, τα ηλεκτρονικά μηνύματα που αποστέλλονται από την Νέδα βασίζονται σε ένα πρότυπο κείμενο και σε έναν μηχανισμό συνένωσης του προτύπου με τις πληροφορίες του χρήστη. Τα πρότυπα κείμενα είναι αρχεία κειμένου που υπάρχουν μέσα στον πηγαίο κώδικα της εφαρμογής. Έτσι κάθε φορά που χρειάζεται η αλλαγή ενός προτύπου, ακόμα και η διόρθωση κάποιου ορθογραφικού λάθος, πρέπει να εμπλακεί κάποιος προγραμματιστής, να αλλάξει των κώδικα και να δημιουργήσει μια καινούρια έκδοση του συστήματος. Όλη αυτή η διαδικασία είναι χρονοβόρα και κάνει κατάχρηση της διαχείρισης των εκδόσεων του συστήματος. Θα μπορούσε λοιπόν να προστεθεί ένας μηχανισμός που να επιτρέπει στους διαχειριστές του συστήματος την επεξεργασία ή και προσθήκη προτύπων μέσα από τον φυλλομετρητή τους.

6.2.3 Άλλες βελτιώσεις

Θα μπορούσαμε να κάνουμε και τις εργασίες που τρέχει ο χρονοπρογραμματιστής παραμετροποιήσιμες μέσο μιας διεπαφής σελίδας. Έτσι ο διαχειριστής του συστήματος θα μπορούσε να αλλάζει τις ώρες που εκτελούνται οι εργασίες, να τις καταργεί ή να τις ενεργοποιεί χωρίς την διαμεσολάβηση κάποιου προγραμματιστή.

Επίσης στην σελίδα προβολής ιστορικού των χρηστών θα μπορούσαν να εμπλουτιστούν τα φίλτρα αναζήτησης που προσφέρονται και να γίνει πιο φιλική προς τον χρήστη η παρουσίαση των αλλαγών σε οντότητες.

Ένα άλλο ενδιαφέρον σύστημα θα ήταν η παραγωγή διαφόρων στατιστικών βάση των ενεργειών που γίνονται από τους χρήστες για να μπορεί να μελετηθεί η συμπεριφορά των χρηστών.

Βιβλιογραφία

1. Apache Tomcat. [Ηλεκτρονικό] http://tomcat.apache.org/.
2. Apache Velocity. *Βικιπαίδεια.* [Ηλεκτρονικό] http://en.wikipedia.org/wiki/Apache_Velocity.
3. Aspect Oriented Programming with Spring. [Ηλεκτρονικό] http://static.springsource.org/spring/docs/2.0.8/reference/aop.html.
4. Aspect-oriented programming. [Ηλεκτρονικό] http://en.wikipedia.org/wiki/Aspect-oriented_programming.
5. **Bauer Christian και King Gavin.** *Java Persistence with Hibernate,* Manning, 2012.
6. **Bayern Shawn.** *JSTL in Action,*Manning, 2002.
7. **Flanagan David.** *JavaScript: The Definitive Guide.* 2006.
8. **Freeman Eric και Robson Elisabeth και Bates Bert και Sierra Kathy** *Head First Design Patterns,* O'Reilly, 2004.
9. Hibernate (Java). *Βικιπαίδεια.* [Ηλεκτρονικό] http://en.wikipedia.org/wiki/Hibernate_(Java).
10. Hibernate Documentation. [Ηλεκτρονικό] http://docs.jboss.org/hibernate/core/3.5/reference/en/html/tutorial.html.
11. Hibernate. [Ηλεκτρονικό] http://www.hibernate.org/docs.
12. Hudson Continuous Integration. [Ηλεκτρονικό] http://hudson-ci.org/.
13. Introducing JSON. *json.org.* [Ηλεκτρονικό] http://www.json.org/.
14. **Laddad Ramnivas.** *AspectJ in Action, 2nd Edition,* Manning, 2009.
15. **Pfleeger Shari Lawrence.** *Software Engineering: Theory and practice 2nd edition,* Prentice Hall, 2001.
16. Quartz-Scheduler. [Ηλεκτρονικό] http://quartz-scheduler.org/.
17. **Smart John Ferguson.** Jenkins, The definitive guide.
18. Spring Framework. [Ηλεκτρονικό] http://www.springsource.org/.
19. Spring Framework. *Βικιπαίδεια.* [Ηλεκτρονικό] http://en.wikipedia.org/wiki/Spring_Framework.

20. **Walls Greig.** *Spring in Action.* 2007.

21. *YUI Library.* [Ηλεκτρονικό] http://yuilibrary.com/.

22. **Ξένος Μιχάλης και Χριστοδουλάκης Δημήτρης.** *Εισαγωγή στης Βάσεις Δεδομένων,* Παπασωτηρίου, 2002.

23. **Σταθόπουλος Αναστάσιος Γ.** *Ανάπτυξη Εφαρμογών Διαχείρισης Διαδικασιών σε Περριβάλλον Java Spring.* 2009.