ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή εργασία

Ruby On Rails
Μια νέα προσέγγιση στην ταχύτερη ανάπτυξη εφαρμογών για το web 2.0

Του φοιτητή
Τσιάντου Βασιλείου
Αρ. Μητρώου: 02/2003

Επιβλέπων καθηγητής

Μάργαρης Αθανάσιος

Θεσσαλονίκη 2009

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ruby on Rails

Μια εύχρηστη πλατφόρμα ανάπτυξης Web εφαρμογών.

Μελέτη και υλοποίηση πιλοτικής εφαρμογής.

Εισαγωγικές σελίδες

Περίληψη

Στην σύγχρονη εποχή του cloud-computing είναι απαραίτητο να βελτιωθούν ποιοτικά και παραγωγικά οι τρόποι ανάπτυξης Διαδικτυακών εφαρμογών. Τα τελευταία χρόνια έχουν γίνει αρκετές προσπάθειες προς αυτή την κατεύθυνση. Έχουν αναπτυχθεί αφενός εμπορικά πακέτα, για παράδειγμα από την Microsoft™ έχει αναπτυχθεί η ASP και το σύγχρονο περιβάλλον ανάπτυξης Visual Web Developer, κι αφετέρου πακέτα ανοικτού κώδικα, για παράδειγμα από την Sun™ έχει αναπτυχθεί η JSP και υπάρχουν διάφορα σύγχρονα εργαλεία που διευκολύνουν την ανάπτυξη όπως το NetBeans.

Μια άλλη πρόταση από τον χώρο του ανοικτού κώδικα είναι το πακέτο Ruby On Rails, το οποίο ακολουθεί πιστά την αρχικτεκτονική Model-View-Controller (MVC). Η ανάπτυξη με το Ruby On Rails δεν απαιτεί την ύπαρξη κάποιου ειδικού περιβάλλοντος καθώς ο πραγματικός κώδικας για τις διάφορες λειτουργίες είναι αρκετά απλός και περιορισμένος.

Σημαντικό είναι να αναφερθεί ότι διαθέτει διάφορα scripts που δημιουργούν τη βασική δομή του προγράμματος, δημιουργούν την βάση δεδομένων ή ακόμα προσθέτουν κάποια «plugins», δηλαδή μια επιπλέον λειτουργία. Η χρήση των scripts διευκολύνει το έργο του προγραμματιστή, καθώς περιορίζει το μέγεθος του κώδικα που τελικά θα γράψει ο προγραμματιστής και ο τελικός κώδικας σε μεγάλο βαθμό είναι αποτέλεσμα μιας μηχανής παραγωγής κώδικα. Τα scripts είναι σημαντικά γιατί εξαιτίας τους οι δύο βασικές αρχές του Ruby On Rails εφαρμόζονται σχεδόν πάντα κατά τη διάρκεια ανάπτυξης. Αυτές οι δύο αρχές είναι η Don’t Repeat Yourself και η Convention Over Configuration.

Οι δύο αυτές αρχές υλοποιούνται εν μέρει, διότι τα scripts μπορούν να δημιουργούν μόνο την αρχική αφηρημένη μορφή του κώδικα. Ο προγραμματιστής όμως είναι

υπεύθυνος στο να ακολουθήσει αυτή την αφηρημένη δομή στην λογική της τήρησης των αρχών. Αυτές οι δύο αρχές έχουν καταφέρει να ανεβάσουν το Ruby On Rails μέσα σε λίγα χρόνια, απο το 2004 που πρωτοξεκίνησε, στις πρώτες θέσεις προτίμησης των εταιριών ανάπτυξης Διαδικτυακών εφαρμογών.

Abstract

In the modern times of cloud-computing it is essential to improve quality and productive ways of developing Web applications. In recent years there have been several efforts in this direction. They have developed both commercial packages, for example Microsoft ™ has developed the ASP and the development environment Visual Web Developer, and open source packages, and for example Sun™ has developed the JSP and the development environment NetBeans.

Another proposal from the world of open source is the package «Ruby On Rails», which follows closely the architecture of Model-View-Controller (MVC). The development with Ruby On Rails is so easy that does not require the existence of any specific development environment. The actual code for the various functions is quite simple and limited. The developer can write the whole code using Notepad or any simple text editor.

It is important to note that existed scripts which create the structure of the program, create the database or even add a «plugin» which is functionality. The use of scripts facilitates the work of the programmer, because they limit the code that the programmer will eventually write, and the final code is mainly a result of code machines (scripts). Scripts are important because they are the reason why the two main concepts of Ruby on Rails are nearly always applied. Those two main concepts are Don't Repeat Yourself and Convention over Configuration.

These two principles are implemented in part, because the scripts can only create the original abstract form of the code. The programmer is responsible but to adopt this structure in the abstract sense of compliance. These two principles have managed to raise the Ruby on Rails in a few years, from 2004 began on first preference positions of companies developing Web applications.

Περιγραφή και ανάλυση του θέματος

Η παρούσα πτυχιακή εργασία θα παρουσιάσει εν συντομία τις διάφορες σύγχρονες τεχνολογίες που αξιοποιούνται για ανάπτυξη σύγχρονων Διαδικτυακών εφαρμογών και θα επικεντρωθεί σε αυτήν του framework Ruby On Rails.

Θα παρουσιάσει τον τρόπο ανάπτυξης εφαρμογών με χρήση του Ruby On Rails. Η παρουσίαση θα γίνει αξιοποιώντας μια πιλοτική εφαρμογή που αναπτύχθηκε για τις ανάγκες της εργασίας. Μέσω αυτής της παρουσίασης θα προβληθεί η φιλοσοφία του framework καθώς και τα πλεονεκτήματα και μειονεκτήματα που έχει.

Σύγχρονες τεχνολογίες ανάπτυξης Διαδικτυακών εφαρμογών

Οι σύγχρονες τεχνολογίες που χρησιμοποιούνται κυρίως στην ανάπτυξη Διαδικτυακών εφαρμογών είναι οι εξής:

1) PHP
    a. Open source
    b. Μεγάλη κοινότητα
    c. Αρκετά δωρεάν eBooks και tutorials
    d. Αρκετές open source εφαρμογές και components
    e. Δεν έχει κάποια κερδοσκοπική εταιρία να την ελέγχει
    f. Εύκολη και απλή δομή
    g. Platform independent

2) JAVA
    a. Open source
    b. Μεγάλη κοινότητα
    c. Αρκετά δωρεάν eBooks και tutorials
    d. Γενικής χρήσης [από smart cards μέχρι data centers]
    e. Αρκετές open source εφαρμογές αλλά και βιβλιοθήκες κώδικα

f. Έχει την Sun® Microsystems να την υποστηρίζει
g. Επικρατέστερη στην αγορά
h. Αυστηρή δομή
i. Database independent[1]
j. Platform independent
k. Έχει αρκετά καλά IDE (NetBeans, Eclipse, JDeveloper κ.α.)

3) ASP.net
   a. Free αλλά όχι open source
   b. Υποστηρίζεται μόνο από προϊόντα της Microsoft®
   c. Εύχρηστο περιβάλλον ανάπτυξης Visual Web Developer Express
   d. Δυνατότητα χρήσης πολλαπλών γλωσσών προγραμματισμού[2]
   e. Συγκεκριμένου στόχου [Web Development]

4) Ruby On Rails
   a. Open source
   b. Αρκετά διαδεδομένη
   c. Συγκεκριμένου στόχου [Web Development]
   d. Ξεκάθαρη δομή MVC
   e. Εγγενή εργαλεία παραγωγής αυτοματοποιημένου κώδικα
   f. Αρχή DRY [Don’t Repeat Yourself]
   g. Αρχή Convention over Configuration
   h. Platform independent
   i. Database independent

[1] Έχει ένα εικονικό επίπεδο που ονομάζεται JDBC (Java DataBase Connectivity), που με αντίστοιχη λογική με το ODBC μπορεί να αποκρύψει από τον προγραμματιστή ποια είναι η βάση δεδομένων κι έτσι να είναι ρυθμιζόμενο.

[2] Όσες υποστηρίζονται από το Visual Studio .NET

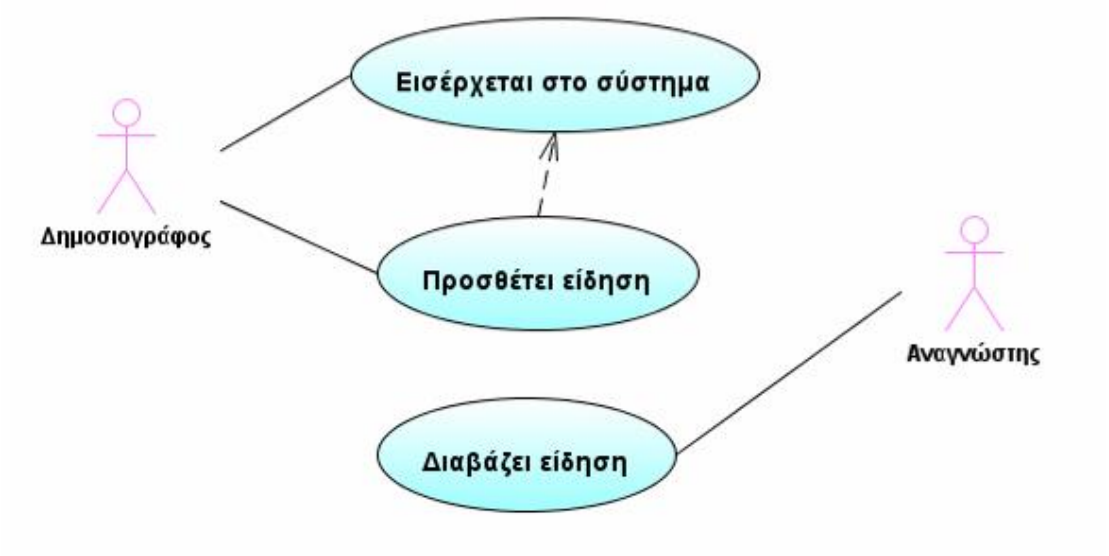
Σύντομη αναφορά στο Ruby On Rails

Το Ruby on Rails είναι ένα web application framework που έχει αρχίσει να διαδίδεται αρκετά από την στιγμή της δημιουργίας του, το 2004, από τον David Heinemeier Hansson. Πρόκειται για ένα πλαίσιο που διευκολύνει τον προγραμματιστή οδηγώντας τον σταδιακά στην ανάπτυξη μιας σωστά δομημένης web εφαρμογής. Αυτό επιτυγχάνεται εν μέρη λόγω της Ruby, που είναι η γλώσσα προγραμματισμού του Ruby On Rails κι είναι αρκετά καλά δομημένη, αλλά και ορισμένων script που διαθέτει το framework, μέσω τον οποίων ο προγραμματιστής δημιουργεί την αρχική μορφή της εφαρμογής του[3] στην οποία κάνει τις όποιες τροποποιήσεις χωρίς να της αλλάζει την δομή.

Πιλοτική εφαρμογή

Όπως έχει αναφερθεί και παραπάνω, στα πλαίσια της παρούσας πτυχιακής αναπτύχθηκε μια πιλοτική εφαρμογή που αξιοποιεί το Ruby On Rails. Η εφαρμογή είναι μια ηλεκτρονική εφημερίδα. Οι λειτουργίες που καλείτε να επιτελέσει είναι αυτή της καταχώρισης ειδήσεων, από τους δημοσιογράφους, αλλά και της προβολής τους στους επισκέπτες-αναγνώστες. Οπότε πρώτον δημιουργήθηκε ένας μηχανισμός μέσω του οποίου εισέρχονται οι δημοσιογράφοι στην εφαρμογή εν συνεχεία δημιουργήθηκε η λειτουργία καταχώρισης ειδήσεων και τέλος δημιουργήθηκε μια κατάλληλη δυναμική σελίδα όπου προβάλλονται αυτές οι ειδήσεις.

[3] Παρουσιάζεται σε επόμενο κεφάλαιο το πώς γίνεται πρακτικά αυτή η αυτοματοποιημένη δημιουργία του κώδικα.



Εικόνα 1 Διάγραμμα περιπτώσεων χρήσης πιλοτικής εφαρμογής

Περιεχόμενα

Εισαγωγικές σελίδες....2
Περίληψη 2
Abstract.... 4
Περιγραφή και ανάλυση του θέματος.... 5
Σύγχρονες τεχνολογίες ανάπτυξης Διαδικτυακών εφαρμογών.... 5
Σύντομη αναφορά στο Ruby On Rails.... 7
Πιλοτική εφαρμογή 7
ΚΕΦΑΛΑΙΟ 1ο 11
ΕΙΣΑΓΩΓΗ.... 11
1.1 Αντικείμενο.... 11
1.2 Μεθοδολογία.... 11
1.3 Δομή της αναφοράς 12
ΚΕΦΑΛΑΙΟ 2ο 13
ΑΝΑΛΥΣΗ ΤΗΣ ΥΦΙΣΤΑΜΕΝΗΣ ΚΑΤΑΣΤΑΣΗΣ 13
2.1 PHP 14
2.2 JAVA.... 15
2.3 ASP .NET 16
2.4 Ruby on Rails 17
ΚΕΦΑΛΑΙΟ 3ο 19
ΑΝΑΛΥΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΤΟΥ RUBY ON RAILS.... 19
3.1 Οι τέσσερις βασικές αντιθετικές αρχές του «Agile Manifesto» 20
3.2 Αρχιτεκτονική Rails 22
MVC 22
Active Record: Rails Model Support.... 26
Προγραμματισμός με επικέντρωση στη βάση δεδομένων.... 27
Object-Relational Mapping.... 29
Active Record.... 30
Action Pack: The View and Controller.... 31
ΚΕΦΑΛΑΙΟ 4ο 34
ΑΝΑΠΤΥΞΗ ΠΙΛΟΤΙΚΗΣ ΕΦΑΡΜΟΓΗΣ.... 34

4.1 Περιγραφή πιλοτικής υλοποίησης 34
4.2 Εργαλεία ανάπτυξης 35
4.3 Εγκατάσταση εργαλείων ανάπτυξης 36
4.4 Επεξήγηση τρόπου υλοποίησης 38
Δημιουργία αρχικής μορφής – “rails demo” 38
Δημιουργία του μοντέλου – “ruby script\generate scaffold …” 39
Γενικές πληροφορίες σχετικά με τον κώδικα 40
Συσχέτιση κλάσεων 47
Προσθήκη ελέγχων – validates_presence_of και άλλα 54
Διαγραφή περιττού κώδικα 55
Προσθήκη δυνατότητας εισόδου χρήστη στην εφαρμογή 56
Μορφοποίηση του template 60
Προσθήκη δυνατότητας rss 62
Εμπλουτισμένος επεξεργαστής κειμένου – WYIWYG 64
4.5 Σχολιασμός τρόπου υλοποίησης - Πλεονεκτήματα και μειονεκτήματα 64
4.6 Παρουσίαση υλοποίησης 65
ΚΕΦΑΛΑΙΟ 5^{ο} 69
ΠΗΓΕΣ 69
5.1 Βιβλιογραφία 69
5.2 Επιπλέον πηγές 70

ΚΕΦΑΛΑΙΟ 1ο

ΕΙΣΑΓΩΓΗ

1.1 Αντικείμενο

Η παρούσα πτυχιακή έχει ως αντικείμενο την μελέτη των επικρατέστερων πλατφόρμων ανάπτυξης εφαρμογών διαδικτύου. Στόχος είναι να παρουσιαστεί η εικόνα που υπάρχει στον συγκεκριμένο χώρο και να παρουσιαστεί εκτενέστερα η «καλύτερη» πλατφόρμα, όπως προέκυψε από την σχετική μελέτη, δηλάδη η Ruby On Rails.

Εφόσον επιλέχθηκε η συγκεκριμένη πλατφόρμα, αναπτύχθηκε και μια πιλοτική εφαρμογή, «Ηλεκτρονική Εφημερίδα» που αποτελεί κλασικό σενάριο Διαδικτυακής εφαρμογής. Η πιλοτική εφαρμογή βοήθησε στο να εξερευνηθεί η Ruby On Rails από πλευράς ευκολίας κώδικα, αλλά και από πλευράς λειτουργικότητας χρήσης.

1.2 Μεθοδολογία

Η μεθοδολογία που ακολουθήθηκε ήταν η ακόλουθη:

Εντοπισμός επικρατέστερων πλατφόρμων ανάπτυξης Διαδικτυακών εφαρμογών

Μελέτη κάθε μίας εκ των πλατφόρμων και δοκιμή κάποιων ενδεικτικών παραδειγμάτων

Αξιολόγιση από πλευράς ευκολίας ανάπτυξης, ευκολίας εξέλιξης και ευκολίας χρήσης. Απόρριψη οποιασδήποτε πλατφόρμας θεωρήθηκε υπερβολικά δύσκολη, δυσνόητη ή δύσχρηστη.

Εντοπισμός και συγγραφή πληροφοριών για τις επιλεγμένες πλατφόρμες.

Αναλυτική μελέτη του τρόπου ανάπτυξης Διαδικτυακών εφαρμογών με χρήση της Ruby on Rails

Δημιουργία πιλοτικής εφαρμογής με την Ruby on Rails

Δημιουργία πιλοτικής εφαρμογής και με μια δεύτερη πλατφόρμα

Σύγκριση των δύο πιλοτικών εφαρμογών καθώς και του τρόπου ανάπτυξης τους.

1.3 Δομή της αναφοράς

Η παρούσα αναφορά αποτελείται από έξι τμήματα:

Εισαγωγικές σελίδες

Περιληπτική παρουσίαση του θέματος με το οποίο ασχολείται η πτυχιακή

ΚΕΦΑΛΑΙΟ 1 ΕΙΣΑΓΩΓΗ

Παρουσίαση του πως οργανώνεται η πτυχιακή. Ποια μεθοδολογία χρησιμοποιήθηκε για την ανάπτυξη της εφαρμογής, αλλά και πως οργανώνεται και η ίδια η αναφορά.

ΚΕΦΑΛΑΙΟ 2 ΑΝΑΛΥΣΗ ΤΗΣ ΥΦΙΣΤΑΜΕΝΗΣ ΚΑΤΑΣΤΑΣΗΣ

Παρουσίαση των επικρατέστερων τεχνολογιών του χώρου.

ΚΕΦΑΛΑΙΟ 3 ΑΝΑΛΥΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΤΟΥ RUBY ON RAILS

Παρουσιάζεται η «φιλοσοφία» πίσω από το Ruby On Rails καθώς και η αρχιτεκτονική του framework.

ΚΕΦΑΛΑΙΟ 4 ΑΝΑΠΤΥΞΗ ΠΙΛΟΤΙΚΗΣ ΕΦΑΡΜΟΓΗΣ

Παρουσιάζονται τα βήματα που ακολουθήθηκαν για να αναπτυχθεί η εφαρμογή, μέσω των οποίων παρουσιάζεται το πώς αναπτύσσονται γενικότερα εφαρμογές με χρήστη του Ruby On Rails.

ΠΗΓΕΣ

Παράθεση της χρησιμοποιημένης βιβλιογραφίας αλλά και των πηγών έτοιμων κομματιών κώδικα που επαναχρησιμοποιήθηκαν.

ΚΕΦΑΛΑΙΟ 2ο

ΑΝΑΛΥΣΗ ΤΗΣ ΥΦΙΣΤΑΜΕΝΗΣ ΚΑΤΑΣΤΑΣΗΣ

Στην σύγχρονη εποχή του cloud computing[4], είναι απαραίτητο να υπάρξουν κατάλληλες γλώσσες προγραμματισμού για την ανάπτυξη σχετικών εφαρμογών. Έχει αρχίσει σταδιακά να παύει η παλιά λογική των stand alone εφαρμογών, δίνοντας τη θέση τους σε εφαρμογές που είναι εγκατεστημένες σε κάποιους απομακρυσμένους web servers αλλά ταυτόχρονα διαθέτουν ένα εξίσου φιλικό γραφικό περιβάλλον.

Η ανάγκη αυτή οδήγησε, αρχικά, στην δημιουργία μια web έκδοσης των τότε γνωστών και ευρέως χρησιμοποιούμενων γλωσσών προγραμματισμού. Για παράδειγμα δημιουργήθηκε η ASP, ως μια web έκδοση της Visual Basic. Αλλά αυτή η λογική είχε το μειονέκτημα ότι γλώσσες γενικής χρήσης χρησιμοποιούνταν για κάτι τόσο ειδικό όπως οι web εφαρμογές. Το αποτέλεσμα ήταν να γράφεται αρκετός κώδικας ο οποίος και επαναλαμβανόταν ακόμα και για τα πλέον «αυτονόητα», όπως η συναλλαγή με μια βάση δεδομένων ή με το http session του χρήστη.

Μια άλλη τακτική που ακολουθήθηκε ήταν η χρήση JavaScript, ώστε κάποια κομμάτια κώδικα να εκτελούνται στον υπολογιστή του client, αλλά ούτε αυτό αρχικά ήταν αποτελεσματικό, διότι ο κάθε προγραμματιστής έγραφε εκ νέου scripts και πάλι επαναλάμβανε τον κώδικα του σε πολλά σημεία.

Την λύση στο πρόβλημα της επαναληπτικότητας το έδωσε εν μέρη η ύπαρξη διαφόρων frameworks. Τα frameworks ήταν βιβλιοθήκες με έτοιμα κομμάτια κώδικα. Έτσι ο προγραμματιστής αντί να γράφει πολλαπλές φορές τον τετρημένο κώδικα, αξιοποιούσε τον αντίστοιχο που είχε ήδη συμπεριλάβει το framework που χρησιμοποιεί. Ένα ακόμα πλεονέκτημα των frameworks είναι η τυποποίηση, δηλαδή ότι οδηγούν τον προγραμματιστή στο να έχει μια συγκεκριμένη, τυπική, μορφή στην

[4] Το cloud computing είναι η παροχή διαδικτυακών υπηρεσιών (από προγράμματα, ως χώρους αποθήκευσης δεδομένων) σε επιχειρήσεις, με τον πάροχο να πληρώνεται γι' αυτές και τις επιχειρήσεις να τις χρησιμοποιούν με βάση τις ανάγκες τους.

δομή του κώδικά του. Αλλά και πάλι υπάρχουν προβλήματα, τα frameworks που υπάρχουν ήδη σχετικά με την ανάπτυξη Διαδικτυακών εφαρμογών είναι αρκετά μεγάλου πλήθους και διαφέρουν κατά πολύ.

Μια ίσως πιο ολοκληρωμένη λύση δίνει το framework Ruby on Rails. Το συγκεκριμένο framework, που χρησιμοποιεί την γλώσσα προγραμματισμού Ruby, είναι αρκετά τυποποιημένο ώστε να κάνει πραγματικότητα αυτό που αναφέρθηκε προηγουμένως περί μη επαναληπτικότητας. Το Ruby On Rails δίνει επίσης μια τέτοια τυποποίηση ώστε ο κώδικας όπου ουσιαστικά καλείται να γράψει ο προγραμματιστής να είναι ο ελάχιστος δυνατός.

Στην συνέχεια παρουσιάζονται οι πλέον ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού για την ανάπτυξη Διαδικτυακών εφαρμογών, καθώς και σχετικά frameworks.

2.1 PHP

Η PHP είναι μια ευρέως χρησιμοποιούμενη γλώσσα προγραμματισμού γενικής χρήσης, η οποία όμως χρησιμοποιείται κυρίως για την ανάπτυξη web based εφαρμογών. Είναι ιδιαίτερα αγαπητή στου προγραμματιστές διότι είναι ανοικτού κώδικα (open source) , είναι αρκετά γρήγορή όσον αφορά το θέμα της επίδοσης και απλή όσον αφορά το θέμα της εκμάθησης. Τα χαρακτηριστικά αυτά όμως δεν την κάνουν να υστερεί καθόλου σε σχέση με οποιαδήποτε άλλη αντίστοιχη γλώσσα προγραμματισμού σε θέματα δυνατοτήτων.

Το βασικό πλεονέκτημα της PHP είναι η ύπαρξη μιας μεγάλης κοινότητας που την υποστηρίζει με αποτέλεσμα την ύπαρξη αρκετά μεγάλου όγκου από εγχειρίδια χρήσης, δείγματα κώδικα, frameworks αλλά και ολοκληρωμένων εφαρμογών ανοικτού κώδικα. Χαρακτηριστικό παράδειγμα ολοκληρωμένης εφαρμογής σε PHP αποτελεί το γνωστό Joomla, ένα σύστημα διαχείρισης περιεχομένου ή CMS (Content Management System).

Οπότε ο προγραμματιστής που ενδιαφέρεται για μια εφαρμογή παρόμοια με κάποια υπάρχουσα υλοποιημένη σε PHP, την κατεβάζει, βρίσκει σχετικά tutorials και την προσαρμόζει στις δικές του απαιτήσεις. Αλλά υπάρχουν και τα μειονεκτήματα. Η κατασκευή μεγάλων εφαρμογών σε PHP συνεπάγεται και την ύπαρξη πολλαπλών αρχείων και μιας γενικότερα εκτεταμένης δομής. Αυτό έχει ως αποτέλεσμα να είναι σχεδόν ανεξέλεγκτος ο κώδικας λόγω του μεγάλου μεγέθους του. Η εφαρμογή Joomla που αναφέρθηκε προηγουμένως είναι χαρακτηριστικό παράδειγμα καθώς η απλή διανομή της αποτελείται από μερικές εκατοντάδες αρχεία που παρότι υπάρχει σχετική τεκμηρίωση είναι αρκετά δύσκολο να το κατανοήσεις πλήρως.

Τέλος θα μπορούσε η PHP να χαρακτηριστεί ως κατάλληλη για ανάπτυξη μεγάλων εφαρμογών αρκεί να υπάρχει εξ' αρχής καλή δομή, η οποία θα πρέπει να δημιουργηθεί από τους σχεδιαστές της.

2.2 JAVA

Η Java αποτελεί ίσως την γλώσσα που κυριολεκτικά είναι γενικής χρήσης. Έχει αξιοποιηθεί για την ανάπτυξη εφαρμογών σε smart cards [Java Card], κινητά τηλέφωνα [J2ME], desktop applications [J2SE], εφαρμογές Διαδικτύου [JSP, J2EE, Java Applets, Java Web Start, JavaFX etc]. Η Java είναι μια γλώσσα που έχει αντίστοιχα μεγάλη κοινότητα, όπως η PHP, λόγω του ότι είναι ανοικτού κώδικα, αλλά έχει και το πλεονέκτημα ότι «ελέγχεται» από μια μεγάλη εταιρία, την Sun® microsystems.

Η ύπαρξη μιας εταιρίας πίσω από την ανάπτυξη της γλώσσα οδηγεί στο να υπάρχει μια δομή στον τρόπο εξέλιξης της. Υπάρχουν τυπικές διαδικασίες οι οποίες ακολουθούνται ώστε κάθε βήμα εξέλιξης να έχει μελετηθεί. Το κυριότερο βήμα στην εξέλιξη κάθε τμήματος της γλώσσας είναι τα JSR [Java Specification Requests], που είναι οι τυποποιήσεις που πρέπει να ακολουθούνται. Είναι το αντίστοιχο με τα πρότυπα ISO.

Έτσι η Java έχει κάποιες τυπικές μεθόδους ανάπτυξης των διαφόρων εφαρμογών, αλλά και πάλι δεν οδηγεί τον προγραμματιστή μέσω κάποιων τυπικών διαδικασιών

στην αποκλειστική ή ευκολότερη χρήση των τυπικών αυτών μεθόδων. Αυτό έχει ως αποτέλεσμα ο κάθε προγραμματιστής να γράφει κατά το δοκούν, εκτός κι αν χρησιμοποιεί κάποια από τα ευρέως διαδεδομένα frameworks.

Τα πλέον γνωστά frameworks και APIs είναι το JSF [Java Server Faces], Apache Struts, Wicket, Spring, GWT [Google Web Toolkit], Echo, ThinWire και πολλά άλλα. Τα περισσότερα από αυτά ασχολούνται με το GUI, καθώς η Java διαθέτει από την λογική του MVC[5] και τα κομμάτια του Model και του Control μπορούν να υλοποιηθούν με αντίστοιχο τρόπο όπως και σε μια desktop εφαρμογή[6].

Η Java διαθέτει επίσης το πλεονέκτημα ύπαρξης πολλών ενοποιημένων περιβαλλόντων ανάπτυξης (IDE: Integrated Development Enviroment). Ως εκ τούτου ο προγραμματιστής διευκολύνεται μέσω των διαφόρων επιπρόσθετων δυνατοτήτων που του δίνουν αυτά τα περιβάλλοντα, όπως είναι η αυτόματη δημιουργία του τετρημένου[7] κώδικα.

Το βασικό μειονέκτημα της Java, όπως και των περισσοτέρων γλωσσών γενικής χρήσης, είναι ότι υπάρχει η ανάγκη να δημιουργείται ένας αρκετά μεγάλος όγκος κώδικα με αποτέλεσμα να γίνεται όλο και πιο δύσκολο να το διαχειριστούμε.

2.3 ASP .NET

Η ASP .net είναι η πρόταση της Microsoft στον κόσμο των Web Application Frameworks. Αποκλειστικός στόχος του συγκεκριμένου framework είναι η ανάπτυξη

---

[5] Model View Control, ένα μοντέλο που διαχωρίζει κάθε εφαρμογή σε τρία τμήματα. Στο τμήμα του μοντέλου, δηλαδή της δομής των δεδομένων. Στο τμήμα της εμφάνισης, δηλαδή του τρόπου επικοινωνίας με τον χρήστη. Στο τμήμα του ελέγχου όπου γίνονται οι διάφορες ενέργειες διαχείρισης των δεδομένων με βάση τις ενέργειες του χρήστη από την μεριά της εμφάνισης.

[6] Σε μια web based εφαρμογή πρέπει να λαμβάνουμε πάντα υπ' όψη ότι πολλοί χρήστες πρέπει να μπορούν παράλληλα να κάνουν είτε τις ίδιες είτε διαφορετικές λειτουργίες χωρίς να συγχέονται μεταξύ τους

[7] Τετρημένο κώδικα θεωρώ τα κομμάτια που δεν έχουν καμία ιδιαίτερη λογική, για παράδειγμα δημιουργία μια μεθόδου getValue που επιστρέφει την τιμή της μεταβλητής value.

δυναμικών ιστοχώρων[8] και υπηρεσιών ιστών[8]. Αποτελεί απόγονο της ASP (Active Server Pages), που ήταν η πρώτη μηχανή script στην μεριά του server για δυναμικά δημιουργούμενες ιστοσελίδες. Δηλαδή ήταν ουσιαστικά η πρώτη σύγχρονη γλώσσα ανάπτυξης δυναμικών ιστοσελίδων.

Η ASP.net έχει τρία βασικά πλεονεκτήματα. Πρώτον έχει μια αναμφισβήτητα μεγάλη εταιρία από πίσω, την Microsoft®, δεύτερον έχει ένα εντυπωσιακά εύχρηστο περιβάλλον ανάπτυξης “Visual Web Developer Express” και τρίτον υποστηρίζει κώδικα σε οποιαδήποτε από τις διάφορες γλώσσες του Visual Studio .net.

Το βασικό μειονέκτημα του συγκεκριμένου framework είναι το κακό παρελθόν της Microsoft® όπου πολύ εύκολα σκοτώνει projects εάν δε της δίνουν τα οικονομικά οφέλη που προσδοκούσε. Χαρακτηριστικό παράδειγμα αποτελεί η Visual Fox, αλλά κι άλλα προγράμματα που έχουν πάψει να υποστηρίζονται από την συγκεκριμένη εταιρία. Επίσης λόγω του ότι πρόκειται για κλειστό λογισμικό δεν δίνει την δυνατότητα σε κανέναν άλλον, ιδιώτη ή εταιρεία, να την βελτιώσει.

2.4 Ruby on Rails

Το Ruby on Rails (RoR) είναι ένα web application framework ανοικτού κώδικα που έχει σχεδιαστεί για τη γλώσσα προγραμματισμού Ruby. Μια γλώσσα προγραμματισμού που είναι αντικειμενοστραφής και χαρακτηρίζεται από συνοχή, παρέχοντας ταυτόχρονα ευκολία στη συγγραφή κώδικα. Το Rails σχεδιάστηκε ώστε να υποστηρίζει την ευκίνητη ανάπτυξη κώδικα[9]. Το γεγονός αυτό οδηγεί στην ανάγκη να παράγεται γρήγορα κώδικας αρκετά απλός και δομημένος ώστε να μπορεί να επεκτείνει και να επεκταθεί.

---

[8] Υπηρεσίες ιστού είναι αυτό που αποδίδεται με τον αγγλικό όρο web services. Δηλαδή υπηρεσίες που μπορούν να αξιοποιηθούν από άλλες εφαρμογές που θέλουν να τις αξιοποιήσουν. Παράδειγμα τέτοιων υπηρεσιών είναι οι υπηρεσίες του web banking που παρέχονται από εφαρμογές που έχουν εγκατασταθεί σε κάποια τράπεζα αλλά αξιοποιούνται από οποιοδήποτε ιστοχώρο χρειάζεται να κάνει τραπεζικές συναλλαγές.

[9] Agile software development

Τα πλεονεκτήματα του είναι οι βασικές του αρχές: η μη επαναλληπτικότητα[10] και η συνθήκη υπεράνω της ρύθμισης[11]. Αυτές οι αρχές εγγενώς λύνουν τα προβλήματα που παρουσιάζονται στις περισσότερες γλώσσες προγραμματισμού. Ένα ακόμα πλεονέκτημα είναι ότι παρέχει, εγγενώς πάλι, αρκετές δυνατότητες αυτοματοποιημένης ανάπτυξης κώδικα[12]. Η αυτοματοποιημένη αυτή δημιουργία έχει ως αποτέλεσμα και την δημιουργία ομοιόμορφου κώδικα, άρα και πιο ευανάγνωστο και συντηρήσιμο. Άλλο ένα σημαντικό χαρακτηριστικό είναι η υποστήριξη της αρχιτεκτονικής MVC[5], όπου τα τρία τμήμα είναι ξεκάθαρα διαχωρισμένα κατά την αυτοματοποιημένη δημιουργία του κώδικα. Είναι σημαντικό να σημειωθεί ότι το Ruby On Rails έχει καταφέρει με διαφανή τρόπο να είναι platform αλλά και data base independent. Αυτό σημαίνει ότι δίνει την δυνατότητα σε αυτόν που θα εγκαταστήσει την εφαρμογή να μπορεί να έχει οποιοδήποτε λειτουργικό σύστημα και βάση δεδομένων χωρίς να χρειαστεί ιδιαίτερη τροποποίηση του κώδικα.

Τέλος πρέπει να αναφερθεί και το μοναδικό ίσως μειονέκτημα του Ruby On Rails που είναι το ότι δεν υπάρχει κάποιο ενοποιημένο περιβάλλον ανάπτυξης αντίστοιχης ευχρηστίας όπως αυτά της ASP.net και Java. Αλλά ομολογουμένως τα command line εργαλεία που διαθέτει κάνουν την δουλειά σχεδόν εξίσου εύκολη.

[10] DRY: Don’t Repeat Yourself

[11] Convention over configuration

[12] Θα παρουσιαστεί σε επόμενο κεφάλαιο πως αναπτύσσουμε εφαρμογές σε Ruby On Rails και θα επεξηγηθεί το πώς γίνεται αυτοματοποιημένη δημιουργία κώδικα

ΚΕΦΑΛΑΙΟ 3ο

ΑΝΑΛΥΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΤΟΥ RUBY ON RAILS

Το Ruby On Rails είναι ένα πλαίσιο (framework) που διευκολύνει την ανάπτυξη, την εγκατάσταση και τη διατήρηση εφαρμογών web. Έπειτα από την αρχική του κυκλοφορία, διαδόθηκε και εφαρμόστηκε σε διεθνές επίπεδο. Εκτός του ότι κέρδισε πολλά βραβεία, είναι ένα από τα πιο δημοφιλή πλαίσια για την ανάπτυξη των λεγόμενων Web 2.0 εφαρμογών. Η εγκυρότητά και η διάδοση του αποδεικνύεται περαιτέρω από το γεγονός ότι χρησιμοποιείται και από μεγάλες πολυεθνικές εταιρίες.

Οι λόγοι που συνετέλεσαν στην επιτυχία του Ruby On Rails, είναι η σημαντική διευκόλυνση που παρέχει ως προς την ανάπτυξη εφαρμογών σε σχέση με Java, PHP, .NET κτλ, η χρήση σύγχρονων και επαγγελματικών τεχνικών διαχρονικής ισχύος, η δυνατότητα ελέγχου κατά τη διάρκεια της ανάπτυξης, αλλά και με το πέρας της, προσθέτοντας έτσι λειτουργικότητα τόσο στον κώδικα, όσο και στις εφαρμογές για τις οποίες αυτός δημιουργείται. Επιπλέον, η γλώσσα προγραμματισμού που χρησιμοποιείται είναι η Ruby, η οποία είναι αντικειμενοστραφής και χαρακτηρίζεται από συνοχή, παρέχοντας ταυτόχρονα ευκολία στη συγγραφή κώδικα. Η χρήση της εν λόγω γλώσσας προσδίδει το σημαντικό πλεονέκτημα της δημιουργίας ευανάγνωστων και μικρών σε έκταση προγραμμάτων. Ταυτόχρονα, ενώ σε άλλα πλαίσια κάποιες διεργασίες συντελούνται σε αρχεία εξωτερικής διαμόρφωσης, στο Ruby On Rails, αυτά είναι ενσωματωμένα στη βάση του κώδικα.

Η φιλοσοφία του Rails έγκειται σε δύο βασικές αρχές: τη λεγόμενη DRY(Don't Repeat Yourself), ή «Μην Επαναλαμβάνεσαι» και αυτήν της συνθήκη υπεράνω της διαμόρφωσης (convention over configuration). Σύμφωνα με την πρώτη αρχή, το κάθε σημείο κώδικα αντιστοιχεί σε συγκεκριμένο μέρος και έτσι αποφεύγονται οι άσκοπες επαναλήψεις. Για την επίτευξη αυτού του στόχου, χρησιμοποιούνται κάποιες βασικές συνθήκες της αρχιτεκτονικής MVC (η οποία θα παρουσιαστεί στη συνέχεια) με τέτοιο τρόπο, ώστε όταν ο κώδικας πρέπει να τροποποιηθεί να διατηρείται η αρχική συνοχή του και συνεπώς να μην απαιτούνται επαναλήψεις αλλαγής κώδικα στα επιμέρους

τυχόν τροποποιημένα σημεία του. Η αρχή της συνθήκης υπεράνω διαμόρφωσης προσφέρει κάποιες θεμελιώδεις ρυθμίσεις για κάθε πλευρά της δημιουργίας εφαρμογών.

Το Rails, σε αντίθεση με άλλα πλαίσια, δεν επιχειρεί να παραμένει επίκαιρο και να ανταποκρίνεται στις τεχνολογικές εξελίξεις, αλλά αντί αυτού, διευκολύνει τον καθορισμό των νέων τάσεων. Για παράδειγμα, επιτρέπει την εισαγωγή και ενσωμάτωση των AJAX και RESTful. Επιπλέον, με το Rails, διευκολύνεται η εγκατάσταση διαδοχικών εκδόσεων σε οποιονδήποτε αριθμό εξυπηρετητών (servers) με μια απλή εντολή. Γενικά, όταν ξεκινάει κάποιος να αναπτύσσει μια εφαρμογή Rails, έχει ήδη έτοιμα τα κεντρικά και βασικά στοιχεία που απαιτούνται και έτσι μειώνεται ο όγκος και ο χρόνος της περαιτέρω εργασίας που απαιτείται.

3.1 Οι τέσσερις βασικές αντιθετικές αρχές του «Agile Manifesto»

Για τη διευκόλυνση της κατανόησης των σκοπών που εξυπηρετεί η δημιουργία και ανάπτυξη του Ruby On Rails, είναι χρήσιμο να αναφερθούν οι τέσσερις βασικές αντιθετικές αρχές του «Agile Manifesto», όπως αυτό προτάθηκε από την συνάντηση που πραγματοποίησαν 17 αναγνωρισμένοι προγραμματιστές το 2001 στη Utah και οι οποίες είναι:

Α. Άτομα και αλληλεπιδράσεις έναντι διαδικασιών και εργαλείων

Η προτεραιότητα που αποδίδεται στα άτομα και τις αλληλεπιδράσεις είναι θεμελιώδες στοιχείο του Rails. Δεν υπάρχουν "βαριά" συστήματα εργαλείων, περίπλοκες διαμορφώσεις, προτίμηση σε συγκεκριμένους editors και κομμάτια κώδικα Ruby. Αυτό προσφέρει διαφάνεια στον κώδικα, καθώς οι όποιες αλλαγές εφαρμοστούν από τους προγραμματιστές διαφαίνονται άμεσα και μπορούν να γίνουν εύκολα αντιληπτές από τους πελάτες.

B. Λειτουργικό λογισμικό έναντι εκτενών εγγράφων κατανόησης

Το Rails δεν απαιτεί εκτενή δημιουργία τεκμηρίωσης (documentation). Παρέχει την ευκολία της δημιουργίας βάσεως HTML εγγράφων για όλο το εύρος της βάσης κώδικα. Τα εκτενή έγγραφα απουσιάζουν και αντί αυτών, στην καρδιά του κάθε Rails project, βρίσκεται μια ομάδα χρηστών και προγραμματιστών που εξερευνούν από κοινού τις ανάγκες που υπάρχουν, καθώς και τους πιθανούς τρόπους ανταπόκρισης σε αυτές. Οι τρόποι επίλυσης των προβλημάτων αλλάζουν καθώς αλλάζει και το επίπεδο εμπειρίας και εξοικείωσης των προγραμματιστών και των χρηστών. Από τα πρώτα ακόμη στάδια της ανάπτυξης του κώδικα, προσφέρεται λειτουργικό λογισμικό, το οποίο παρόλο που είναι σε στάδιο προς ανάπτυξη, παρέχει μια αρχική εικόνα του τελικού αποτελέσματος.

C. Συνεργασία με τον πελάτη έναντι διαπραγματεύσεων σύνταξης συμβολαίων

Με αυτόν ακριβώς τον τρόπο πριμοδοτείται η συνεργασία με τον πελάτη. Με τον ευέλικτο τρόπο ανάπτυξης και την άμεση ανταπόκριση όταν απαιτούνται τροποποιήσεις, αυξάνεται το επίπεδο εμπιστοσύνης του πελάτη προς την ομάδα ανάπτυξης, όχι μόνο ως προς το τι έχει αρχικά ζητηθεί, αλλά και ως προς το ότι θα επιτευχθεί οτιδήποτε είναι απαραίτητο σχετικά.

D. Ανταπόκριση στην αλλαγή έναντι της ακολουθίας κάποιου ορισμένου σχεδίου

Έτσι οδηγούμαστε στην αρχή της άμεσης ανταπόκρισης στην αλλαγή, σε συνδυασμό με την αρχή του DRY, όπως αυτή προαναφέρθηκε και κατά την οποία, οποιαδήποτε τροποποίηση συντελείται, απαιτεί κατά πολύ μικρότερο μέγεθος κώδικα σε σχέση με άλλα πλαίσια. Με τη χρήση της γλώσσας Ruby, οι αλλαγές διεξάγονται σε συγκεκριμένα σημεία και είναι εύκολος ο τρόπος συγγραφής τους. Επιπλέον, το δικαίωμα ελέγχου που προσφέρεται τόσο σε επίπεδο μονάδας όσο και ως προς τη λειτουργικότητα, σε συνδυασμό με την επιπλέον ενσωματωμένη υποστήριξη, παρέχει

στους προγραμματιστές την απαραίτητη ασφάλεια για την πραγματοποίηση των απαιτούμενων τροποποιήσεων.

Όλα τα παραπάνω καθιστούν το Ruby On Rails ως ένα αλληλεπιδραστικό πλαίσιο ανάπτυξης εφαρμογών, το οποίο επιτρέπει με ευκολία και αμεσότητα την δημιουργία εφαρμογών με από κοινού συνεννόηση μεταξύ των προγραμματιστών και των πελατών.

3.2 Αρχιτεκτονική Rails

Σχετικά με την αρχιτεκτονική του Rails, ένα βασικό της χαρακτηριστικό έγκειται στο γεγονός ότι ενώ επιβάλλει κάποιους σημαντικούς περιορισμούς ως προς τη δόμηση εφαρμογών, είναι ακριβώς αυτό το στοιχείο που επιτρέπει τη διευκόλυνση της δημιουργίας τους.

MVC

Το 1979 ο Trygve Reenskaug διάρθρωσε μια αρχιτεκτονική ανάπτυξης διαδραστικών εφαρμογών. Στη σχηματική αυτή αρχιτεκτονική, οι εφαρμογές αποτελούνταν από τρεις διακριτούς τύπους συστατικών στοιχείων:

Model

Το model είναι υπεύθυνο για τη διατήρηση της κατάστασης της εφαρμογής. Κάποιες φορές αυτή η κατάσταση είναι βραχύβια και διαρκεί μόνο για λίγες φορές αλληλεπίδρασης με τον χρήστη. Κάποιες άλλες φορές η κατάσταση είναι μόνιμη και αποθηκεύεται εξωτερικά της εφαρμογής, συχνά σε κάποια βάση δεδομένων.

Το model δεν περιέχει απλώς δεδομένα: Επιβάλλει και ενισχύει όλους τους επιχειρηματικούς κανόνες που εφαρμόζονται στα εν λόγω δεδομένα. Για παράδειγμα, αν κάποια έκπτωση δε θα έπρεπε να εφαρμοστεί σε παραγγελίες κόστους χαμηλότερου των 20 ευρώ, το Model θα εφαρμόσει την περιοριστική αυτή συνθήκη. Αυτό φαίνεται λογικό, καθώς εισάγοντας την εφαρμογή τέτοιων κανόνων στο Model

εξασφαλίζεται η εγκυρότητα των δεδομένων. Έτσι, το model λειτουργεί περιφρουρητικά και αποθηκευτικά ταυτόχρονα.

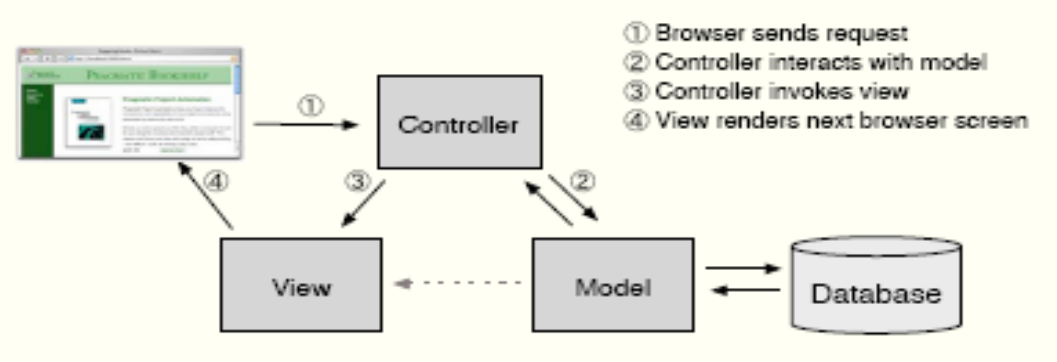
View

Το view είναι υπεύθυνο για τη δημιουργία της διάταξης χρήστη (User interface), που συνήθως βασίζεται στα δεδομένα που περιέχονται στο Model. Για παράδειγμα, ένα ηλεκτρονικό κατάστημα διαθέτει μια λίστα προϊόντων που εμφανίζονται σε μια οθόνη καταλόγου. Αυτή η λίστα είναι προσβάσιμη μέσω του model, αλλά θα είναι ένα view αυτό το οποίο θα έχει πρόσβαση στη λίστα του Model και θα την τροποποιεί για τον τελικό χρήστη. Παρόλο που το view μπορεί να προσφέρει στο χρήστη διάφορους τρόπους εισαγωγής δεδομένων, δε διαχειρίζεται ποτέ εισερχόμενα δεδομένα. Η λειτουργία του view παρέρχεται με την προβολή των δεδομένων. Είναι δυνατόν να υπάρχουν πολλά views που να έχουν πρόσβαση στα ίδια δεδομένα του model, συχνά για διαφορετικούς σκοπούς. Στο ηλεκτρονικό κατάστημα, θα υπάρχει ένα view που θα εμφανίζει τις πληροφορίες των προϊόντων σε μια σελίδα καταλόγου και ένα άλλο σύνολο views που θα χρησιμοποιείται από τους διαχειριστές για να προσθέτουν και να τροποποιούν προϊόντα.

Controller

Τα controllers διευθετούν την εφαρμογή. Λαμβάνουν δεδομένα από τον εξωτερικό κόσμο (συνήθως από τον χρήστη), αλληλεπιδρούν με το model και εμφανίζουν το κατάλληλο view στον χρήστη.

Η τριαρχική αυτή δομή (model, view, controller) διαμορφώνει την αρχιτεκτονική που ονομάζεται MVC.



Εικόνα 2 Διάγραμμα που απεικονίζει τη δομή του MVC

Η αρχιτεκτονική MVC αρχικά δημιουργήθηκε για συμβατικές εφαρμογές GUI, για την οποία οι προγραμματιστές διαπίστωσαν ότι ο διαχωρισμός των ενασχολήσεων οδήγησε στη διευκόλυνση της συγγραφής και διατήρησης του κώδικα, αφού η κάθε έννοια και δράση εκφράζεται σε μόνο ένα σαφώς καθορισμένο μέρος. Η χρήση του MVC παρομοιάστηκε με την κατασκευή ουρανοξύστη που να έχει τις δοκούς υποστήριξης ήδη τοποθετημένες. Έτσι, η τοποθέτηση των επιπλέον δομικών υλικών θα ήταν πολύ πιο εύκολη.

Στον κόσμο του λογισμικού, οι καλές ιδέες του παρελθόντος συχνά θεωρούνται παρωχημένες, καθώς αυτός διακρίνεται από τον προσανατολισμό στο μέλλον. Όταν οι προγραμματιστές άρχισαν να παράγουν εφαρμογές web, επέστρεψαν στη συγγραφή μονολιθικών προγραμμάτων που συνέχεαν την παρουσίαση, την πρόσβαση σε βάσεις δεδομένων, την επιχειρησιακή λογική και τη διαχείριση γεγονότων σε ένα μεγάλο κομμάτι κώδικα. Οι ιδέες όμως από το παρελθόν έγιναν και πάλι επίκαιρες και οι προγραμματιστές άρχισαν να πειραματίζονται με αρχιτεκτονικές εφαρμογών web που απεικόνιζαν τις 20 και πλέον ετών ιδέες του MVC. Το αποτέλεσμα ήταν η δημιουργία πλαισίων όπως αυτών των WebObjects, Struts, και JavaServer Faces. Όλα αυτά, σε κυμαινόμενους βαθμούς πιστότητας, βασίζονται στις ιδέες του MVC.

Το Ruby On Rails είναι επίσης ένα MVC πλαίσιο. Το Rails διαμορφώνει και ενισχύει τη δομή της εφαρμογής, με την ανάπτυξη models, views και controllers, ως διακριτών μερών λειτουργικότητας και τα συνδέει όλα μαζί κατά την εκτέλεση του προγράμματος. Αυτή η διαδικασία βασίζεται στη χρήση έξυπνων εξ ορισμού

ρυθμίσεων (defaults), γεγονός που επιτρέπει την παράλειψη της συγγραφής μεταδεδομένων εξωτερικής διαμόρφωσης για τη λειτουργία της εφαρμογής. Αυτό είναι ένα ενδεικτικό παράδειγμα της φιλοσοφίας της συνθήκης υπεράνω διαμόρφωσης που χαρακτηρίζει το Ruby.

Σε μια εφαρμογή Ruby, τα εισερχόμενα αιτήματα αποστέλλονται αρχικά σε ένα δρομολογητή (router), ο οποίος ρυθμίζει σε ποιο μέρος της εφαρμογής θα πρέπει να προωθηθεί το κάθε αίτημα και με ποιό τρόπο θα αναλυθεί. Αυτό το στάδιο αποτελεί μια διακριτή μέθοδο (που στη γλώσσα του Rails αποκαλείται δράση/action) σε κάποιο σημείο του κώδικα του controller. Η δράση μπορεί να απευθυνθεί στα δεδομένα του αιτήματος, να αλληλεπιδράσει με το model ή μπορεί να προκαλέσει την πραγματοποίηση άλλων δράσεων. Σταδιακά η δράση προετοιμάζει τις πληροφορίες για το view, το οποίο επιστρέφει κάποιο αποτέλεσμα στον χρήστη.

Το παραπάνω σχήμα απεικονίζει τον τρόπο με τον οποίο το Rails διαχειρίζεται ένα εισερχόμενο αίτημα. Σε αυτό το παράδειγμα, η εφαρμογή έχει ήδη απεικονίσει μια σελίδα προϊόντος καταλόγου και ο χρήστης έχει πατήσει το “add to cart” κουμπί δίπλα σε αυτό των προϊόντων. Αυτό το κουμπί συνδέεται με το http://my.url/store/add_to_cart/123, όπου το add_to_cart αποτελεί μια δράση στην εφαρμογή μας και το 123 είναι η εσωτερική ταυτότητα για το επιλεγμένο προϊόν. Το στοιχείο δρομολόγησης λαμβάνει το εισερχόμενο αίτημα και αμέσως το διαχωρίζει. Σε αυτήν την απλή περίπτωση το πρώτο κομμάτι της διαδρομής, το store, είναι το όνομα του controller και το δεύτερο κομμάτι, το add_to_cart, είναι το όνομα της δράσης. Το τελευταίο μέρος, το 123, εξάγεται κατά συνθήκη σε μια εσωτερική παράμετρο που ονομάζεται id. Σαν αποτέλεσμα της όλης αυτής διαδικασίας, ο δρομολογητής γνωρίζει ότι πρέπει να επικαλεστεί τη μέθοδο add_to_cart στην controller class StoreController.

Η μέθοδος add_to_cart διαχειρίζεται αιτήματα χρηστών. Σε αυτή την περίπτωση εντοπίζει την κάρτα αγορών του χρήστη (η οποία είναι ένα αντικείμενο διαχειριζόμενο από το model). Επίσης, ζητάει από το model να εντοπίσει τις πληροφορίες για το προϊόν 123. Έπειτα, ζητάει από την κάρτα αγορών να προσθέσει το προϊόν στα περιεχόμενά της. Αυτό το παράδειγμα δείχνει σαφώς το πώς διατηρούνται τα

επιχειρησιακά δεδομένα μέσω του Model: το controller ενημερώνει σχετικά με το τι πρέπει να γίνει και το model γνωρίζει πώς να το κάνει.

Τώρα που η κάρτα περιέχει το νέο προϊόν, μπορεί να εμφανιστεί στον χρήστη. Το controller διευθετεί την κατάσταση με τέτοιο τρόπο ώστε το view να έχει πρόσβαση στο αντικείμενο της κάρτας από το model και αυτό με τη σειρά του επικαλείται τον κώδικα του view. Στο Rails, η επίκληση είναι συχνά αυτονόητη: για άλλη μια φορά οι προτυποποιήσεις διευκολύνουν τη σύνδεση ενός συγκεκριμένου view με μία δεδομένη δράση.

Αυτός είναι ο τρόπος με τον οποίο λειτουργεί μια MVC εφαρμογή web. Ακολουθώντας ένα συγκεκριμένο σύνολο προτυποποιήσεων και καταμερίζοντας τη λειτουργικότητα καταλλήλως, ο κώδικας καθίσταται πιο εύχρηστος και η εφαρμογή επεκτείνεται και διατηρείται πιο εύκολα.

Αν το MVC είναι απλά ζήτημα καταμερισμού του κώδικα με συγκεκριμένο τρόπο, γεννάται το ερώτημα σχετικά με το λόγο για τον οποίο χρειάζεται η ύπαρξη ενός πλαισίου όπως το Ruby On Rails. Η απάντηση είναι απλή: Το Rails διαχειρίζεται όλες τις απαραίτητες διεργασίες χαμηλού επιπέδου, δηλαδή όλο τον όγκο των λεπτομερειών, εξασφαλίζοντας απαραίτητο χρόνο στον προγραμματιστή για να ασχοληθεί με ζητήματα λειτουργικότητας της εφαρμογής.

Active Record: Rails Model Support

Κατά κοινή παραδοχή, είναι επιθυμητό οι εφαρμογές web να διατηρούν τις πληροφορίες τους σε μια σχεσιακή βάση δεδομένων. Τα συστήματα τύπου order-entry αποθηκεύουν εντολές, αντικείμενα γραμμής και στοιχεία πελατών σε πίνακες βάσεων δεδομένων. Ακόμη και οι εφαρμογές που χρησιμοποιούν μη δομημένο κείμενο, όπως τα weblogs και οι καινούριες ιστοσελίδες, συχνά χρησιμοποιούν βάσεις δεδομένων σαν back end data store.

Παρόλο που δε γίνεται άμεσα εμφανές μέσω της SQL που χρησιμοποιείται για την πρόσβαση σε αυτές, οι σχεσιακές βάσεις δεδομένων είναι σχεδιασμένες με βάση την μαθηματική θεωρία. Αν και αυτό θεωρείται θετικό, δυσχεραίνει τον συνδυασμό

σχεσιακών βάσεων δεδομένων με γλώσσες αντικειμενοστραφούς προγραμματισμού. Τα αντικείμενα αφορούν σε δεδομένα και λειτουργίες, ενώ οι βάσεις δεδομένων σε σύνολα αξιών. Παρόλο που οι λειτουργίες εκφράζονται εύκολα σχεσιακά, είναι ορισμένες φορές δύσκολο να κωδικοποιηθούν σε ένα ΟΟ σύστημα. Ισχύει επίσης και το αντίστροφο.

Για το λόγο αυτό, αναπτύχθηκαν τρόποι συμφιλίωσης των σχεσιακών και των ΟΟ views των εταιρικών δεδομένων. Οι δύο κυρίαρχες προσεγγίσεις είναι οι ακόλουθες: η ανάπτυξη του προγράμματος γύρω από τη βάση δεδομένων και η οργάνωση της βάσης δεδομένων γύρω από το πρόγραμμα. Οι προσεγγίσεις αυτές θα αναπτυχθούν στη συνέχεια.

Προγραμματισμός με επικέντρωση στη βάση δεδομένων

Οι πρώτες απόπειρες προγραμματισμού σχεσιακών βάσεων δεδομένων πραγματοποιήθηκαν σε διαδικαστικές γλώσσες όπως η C και η COBOL, εμπεριέχοντας και SQL απευθείας στο σώμα του κυρίως κώδικα, είτε με τη μορφή strings (συμβολοσειρών), είτε χρησιμοποιώντας έναν προεπεξεργαστή που μετέτρεπε την SQL στη μορφή του πηγαίου κώδικα μέσα στις κλήσεις χαμηλού επιπέδου εσωτερικά της βάσης δεδομένων.

Η ενσωμάτωση αυτή σήμαινε πως ήταν φυσιολογικό να συνυφαίνεται η λογική της βάσης δεδομένων με την συνολική λογική της εφαρμογής. Ένας προγραμματιστής που θα επιθυμούσε να διαπεράσει τις παραγγελίες και να ενημερώσει τον φόρο πωλήσεων για την κάθε παραγγελία θα μπορούσε να συντάξει τον εξής κώδικα:

```
EXEC SQL BEGIN DECLARE SECTION;
int id;
float amount;
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE c1 AS CURSOR FOR select id, amount from orders;
while (1) {
      float tax;
      EXEC SQL WHENEVER NOT FOUND DO break;
      EXEC SQL FETCH c1 INTO :id, :amount;
```

```
      tax = calc_sales_tax(amount)
      EXEC SQL UPDATE orders set tax = :tax where id = :id;
}
EXEC SQL CLOSE c1;
EXEC SQL COMMIT WORK;
```

Αυτή η μορφή προγραμματισμού είναι κοινή σε γλώσσες scripting όπως η Perl και η PHP. Διατίθεται επίσης και στην Ruby. Για παράδειγμα θα μπορούσε να χρησιμοποιηθεί μια DBI βιβλιοθήκη για να παραχθεί κώδικας παρόμοιας μορφής.

```
def update_sales_tax
      update = @db.prepare("update orders set tax=? where id=?" )
            @db.select_all("select id, amount from orders" ) do |id, amount|
            tax = calc_sales_tax(amount)
            update.execute(tax, id)
      end
end
```

Αυτή η προσέγγιση είναι συνεκτική και άμεση και είναι ευρέως διαδεδομένη. Θεωρείται ιδανική λύση για μικρές εφαρμογές. Ωστόσο, υπάρχει ένα πρόβλημα: Ο συνδυασμός αυτός της επιχειρησιακής λογικής και της πρόσβασης στη βάση δεδομένων μπορεί να δυσχεράνει τη μελλοντική επέκταση και διατήρηση των εφαρμογών. Επίσης, απαιτεί τη γνώση SQL.

Αν για παράδειγμα προταθεί ένας νόμος που απαιτεί την καταγραφή της ημερομηνίας και της ώρας που υπολογίστηκε ο φόρος πωλήσεων, αυτό εκ πρώτης όψεως δεν φαίνεται να ενέχει μεγάλου βαθμού δυσκολίας. Θα πρέπει απλώς να προστεθεί η τρέχουσα ώρα, μια στήλη στη δήλωση SQL update, και να καταγραφεί η ώρα κατά την εκτέλεση του αιτήματος.

Τι θα συνέβαινε όμως στην περίπτωση που θα έπρεπε να τεθεί η στήλη του φόρου πωλήσεων σε πολλά διαφορετικά σημεία της εφαρμογής; Σε τέτοια περίπτωση θα έπρεπε να εντοπιστούν και να ενημερωθούν όλα τα σχετικά μέρη. Το αποτέλεσμα θα είναι η επανάληψη του ίδιου μέρους κώδικα και συνεπακολούθως μια πηγή πιθανών λαθών σε περίπτωση παραλείψεων.

Στον κοινό αντικειμενοστραφή προγραμματισμό έχει διαφανεί πως το encapsulation επιλύει τέτοιους τύπους προβλημάτων: Εφόσον οτιδήποτε αφορά τις παραγγελίες

συμπεριληφθεί σε μια κλάση, θα υπάρχει μόνο ένα μέρος προς ενημέρωση σε περίπτωση που τροποποιηθούν οι κανονισμοί.

Αυτές οι ιδέες έχουν επεκταθεί και στον προγραμματισμό βάσεων δεδομένων. Η βασική παραδοχή είναι απλή: Η πρόσβαση στη βάση δεδομένων συμπεριλαμβάνεται πίσω από ένα στρώμα κλάσεων. Το υπόλοιπο μέρος της εφαρμογής χρησιμοποιεί αυτές τις κλάσεις και τα αντικείμενά τους, χωρίς ποτέ να αλληλεπιδρά άμεσα με την βάση δεδομένων. Με αυτόν τον τρόπο όλα τα περιεχόμενα συγκεκριμένης σχηματικής συμπεριλαμβάνονται σε ένα μονό στρώμα και έτσι αποσυσχετίζεται ο κώδικας της εφαρμογής από τις λεπτομέρειες χαμηλού επιπέδου της πρόσβασης στη βάση δεδομένων. Στην περίπτωση της τροποποίησης στο φόρο πωλήσεων θα έπρεπε απλώς να αλλάξει η κλάση αναδιπλώσεως του πίνακα παραγγελιών, ώστε να ενημερώνεται η σήμανση ώρας όποτε τροποποιείται ο φόρος πωλήσεων.

Αν και αυτή η ιδέα φαίνεται απλή, είναι δύσκολο να εφαρμοστεί στην πράξη. Οι πίνακες βάσεων δεδομένων διασυνδέονται και αυτό θα πρέπει να αντικατοπτρίζεται στα αντικείμενα: το αντικείμενο παραγγελίας θα έπρεπε να περιέχει μια συλλογή αντικειμένων γραμμής. Σε αυτό όμως το σημείο εγείρονται ζητήματα περιήγησης αντικειμένου, απόδοσης και συνεκτικότητας δεδομένων. Για το λόγο αυτό επινοήθηκε το ORM ή Object-Relational Mapping, το οποίο και χρησιμοποιείται από το Rails.

Object-Relational Mapping

Οι ORM βιβλιοθήκες αποτυπώνουν τους πίνακες βάσεων δεδομένων σε κλάσεις. Εάν μια βάση δεδομένων διαθέτει έναν πίνακα που ονομάζεται orders, το τελικό πρόγραμμα θα έχει μια κλάση που θα ονομάζεται Order. Οι γραμμές του πίνακα αντιστοιχούν σε αντικείμενα της κλάσης. Στο παράδειγμά αυτό, μια συγκεκριμένη παραγγελία αναπαρίσταται ως αντικείμενο της κλάσης Order. Μέσα σε ένα τέτοιο αντικείμενο, τα χαρακτηριστικά χρησιμοποιούνται για να ληφθούν και να οριστούν οι διακριτές στήλες. Το αντικείμενο Order διαθέτει μεθόδους για την παραλαβή και τον καθορισμό των ποσοτήτων, του φόρου πωλήσεως κτλ.

Επιπλέον, οι κλάσεις του Rails που εμπεριέχουν τους πίνακες της βάσης δεδομένων, παρέχουν ένα σύνολο μεθόδων επιπέδου κλάσεων που επιτελούν λειτουργίες επιπέδου πίνακα. Για παράδειγμα, ίσως χρειαστεί να εντοπιστεί μια παραγγελία με συγκεκριμένη ταυτότητα. Αυτό εφαρμόζεται ως μέθοδος κλάσεως που επιστρέφει το αντίστοιχο αντικείμενο Order. Στον κώδικα Ruby, αυτό μπορεί να είναι ως εξής:

```
Order = order.find(1)
Puts  "Order #{order.customer_id}, amount=#{order,amount}"
```

Κάποιες φορές αυτές οι μέθοδοι επιπέδου κλάσεως επιστρέφουν συλλογές αντικειμένων.

```
Order.find(:all, :conditions => "name='dave'").each do |order|
Order.discount = 0.5
Order.save
End
```

Τέλος, τα αντικείμενα που αντιστοιχούν σε διακριτές γραμμές πίνακα διαθέτουν μεθόδους που επιτελούνται σε αυτές τις συγκεκριμένες γραμμές. Ίσως η πιο ευρέως χρησιμοποιούμενη λειτουργία να είναι η save, η οποία «σώζει» τη γραμμή στη βάση δεδομένων.

Έτσι, ένα στρώμα ORM αποτυπώνει τους πίνακες σε κλάσεις, τις γραμμές σε αντικείμενα και τις στήλες στις ιδιότητες των αντικειμένων αυτών. Οι μέθοδοι κλάσεων χρησιμοποιούνται για να επιτελεστούν λειτουργίες επιπέδου πίνακα και οι μέθοδοι εισηγήσεων (instances) επιτελούν λειτουργίες στις διακριτές γραμμές.

Σε μια τυπική ORM βιβλιοθήκη, διατίθενται δεδομένα διαμόρφωσης ώστε να καθοριστούν οι αποτυπώσεις (mappings) μεταξύ των οντοτήτων της βάσης δεδομένων και αυτών του προγράμματος. Οι προγραμματιστές που χρησιμοποιούν αυτά τα ORM εργαλεία, συχνά αναγκάζονται να δημιουργούν και να διατηρούν μεγάλους όγκους XML αρχείων διαμόρφωσης.

Active Record

Το Active Record είναι το ORM στρώμα που παρέχεται με το Rails. Ακολουθεί το πρότυπο του ORM μοντέλου: οι πίνακες αποτυπώνονται σε κλάσεις, οι γραμμές σε

αντικείμενα και οι στήλες σε ιδιότητες αντικειμένων. Διαφέρει από τις περισσότερες ORM βιβλιοθήκες ως προς τον τρόπο που διαμορφώνεται. Βασιζόμενο στην προτυποποίηση και ξεκινώντας με συγκεκριμένες εξ ορισμού ρυθμίσεις (defaults), το Active Record ελαχιστοποιεί την ποσότητα της διαμόρφωσης που επιτελούν οι προγραμματιστές. Για να γίνει αυτό εμφανές, ακολουθεί ένα παράδειγμα προγράμματος που χρησιμοποιεί Active Record για να περιπτύξει τον πίνακα orders.

```
Require  'active_record'
Class order < ActiveRecord::Base
end
```

Αυτός ο κώδικας χρησιμοποιεί τη νέα κλάση Order για να προσθέσει την ταυτότητα (id) 1 και να τροποποιήσει την έκπτωση. Έτσι, το Active Record επιτρέπει στον προγραμματιστή να επικεντρωθεί στην επιχειρησιακή λογική, αποφεύγοντας την ενασχόληση με την υποκείμενη βάση δεδομένων.

Επιπλέον, το Active Record ενσωματώνεται με το υπόλοιπο μέρος του πλαισίου Rails. Εάν μια φόρμα web αποστείλει τα δεδομένα της εφαρμογής σε ένα επιχειρησιακό αντικείμενο, το Active Record δύναται να το εξάγει στο model. Το Active Record υποστηρίζει εκλεπτυσμένη επικύρωση δεδομένων model και στην περίπτωση που η φόρμα δεδομένων αποτύχει να επικυρωθεί, τα views του Rails μπορούν να εξάγουν και να διαμορφώσουν σφάλματα με μία μόνο γραμμή κώδικα.

Το Active Record αποτελεί το θεμελιώδες έρεισμα model της Rails MVC αρχιτεκτονικής.

Action Pack: The View and Controller

Τα επονομαζόμενα View και Controller μέρη της MVC αρχιτεκτονικής φαίνονται να συσχετίζονται άμεσα μεταξύ τους. Το controller προμηθεύει το view με δεδομένα και στη συνέχεια λαμβάνει γεγονότα από τις σελίδες που παράγονται από τα views. Εξαιτίας αυτών των αλληλεπιδράσεων, η υποστήριξη των views και των controllers στο Rails συμπεριλαμβάνονται σε ένα ενιαίο κομμάτι, που ονομάζεται Action Pack.

Παρόλο που το Action Pack είναι ενιαίο, παρέχει τον διαχωρισμό που απαιτείται για τη συγγραφή εφαρμογών web με σαφώς προσδιορισμένο κώδικα για τον έλεγχο και τη λογική της παρουσίασης.

View Support

Στο Rails, το view είναι υπεύθυνο για τη δημιουργία είτε του συνόλου, είτε μέρους σελίδας που εμφανίζεται σε έναν φυλλομετρητή (browser). Στην πιο απλή μορφή του, ένα view είναι ένα κομμάτι HTML κώδικα που εμφανίζει κάποιο προσχεδιασμένο κείμενο. Στην πράξη, συνήθως απαιτείται η εισαγωγή δυναμικού περιεχομένου που δημιουργείται από τη μέθοδο της δράσης στο controller.

Στο Rails, το δυναμικό περιεχόμενο παράγεται από πρότυπα (templates), που συνήθως διατίθενται σε τρεις διαφορετικούς τύπους. Το πιο συχνό σχήμα προτυποποίησης ονομάζεται rhtml και εμπεριέχει ψήγματα κώδικα Ruby μέσα στον κώδικα HTML του view, χρησιμοποιώντας ένα εργαλείο Ruby το οποίο ονομάζεται ERb (Embedded Ruby). Αυτή η προσέγγιση είναι εξαιρετικά ευέλικτη, αν και έχει ειπωθεί ότι παραβιάζει το πνεύμα της MVC αρχιτεκτονικής, καθώς έχει διατυπωθεί ο ισχυρισμός ότι εμπεριέχει το ρίσκο του να προστεθεί στο view λογική που θα έπρεπε να συμπεριλαμβάνεται στο model ή στο controller. Ο ισχυρισμός αυτός δεν ευσταθεί, εφόσον τα views περιέχουν ενεργό κώδικα ακόμα και στους αρχικούς MVC τύπους αρχιτεκτονικής. Η διατήρηση του σαφούς διαχωρισμού των ζητημάτων που προκύπτουν στην πορεία, έγκειται στην ευχέρεια του προγραμματιστή.

Το δεύτερο σχήμα προτυποποίησης ονομάζεται rxml και επιτρέπει την κατασκευή XML εγγράφων με τη χρήση κώδικα Ruby. Η δομή του παραγόμενου XML κώδικα ακολουθεί αυτομάτως τη δομή του κώδικα.

Επιπλέον, το Rails διαθέτει views τύπου rjs. Αυτές επιτρέπουν τη δημιουργία κατακερματισμών JavaScript στον εξυπηρετητή, οι οποίοι εκτελούνται στον φυλλομετρητή. Αυτό εξυπηρετεί άμεσα τη δημιουργία προβολών (interfaces) Ajax.

Controller

Το Rails controller είναι το λογικό κέντρο της εφαρμογής. Συντονίζει την αλληλεπίδραση μεταξύ του χρήστη, των views και του model. Ωστόσο, το Rails διαχειρίζεται το μεγαλύτερο μέρος αυτής της αλληλεπίδρασης στο παρασκήνιο, εφόσον ο κώδικας που συντάσσεται επικεντρώνεται στην λειτουργικότητα σε επίπεδο εφαρμογής. Αυτό ανάγει το Rails controller σε αξιοσημείωτα εύκολο ως προς την ανάπτυξη και τη διατήρηση.

Επιπλέον, το controller φιλοξενεί έναν αριθμό σημαντικών υποβοηθητικών υπηρεσιών:

1) Είναι υπεύθυνο για τη δρομολόγηση εξωτερικών αιτημάτων σε εσωτερικές δράσεις. Διαχειρίζεται με μεγάλη αποτελεσματικότητα φιλικά στους χρήστες URLs.
2) Διαχειρίζεται το caching, το οποίο αποδίδει στις εφαρμογές σημαντική βελτίωση της απόδοσης.
3) Διαχειρίζεται βοηθητικές λογισμικές μονάδες, που επεκτείνουν τις ικανότητες των view προτύπων, χωρίς να παρεμβαίνουν στον κώδικά τους.

Διαχειρίζεται συνεδρίες, δίνοντας στους χρήστες την εντύπωση της αλληλεπίδρασης με τις εφαρμογές.

ΚΕΦΑΛΑΙΟ 4ο

ΑΝΑΠΤΥΞΗ ΠΙΛΟΤΙΚΗΣ ΕΦΑΡΜΟΓΗΣ

Σε αυτό το κεφάλαιο θα παρουσιαστεί ο τρόπος ανάπτυξης και η λειτουργία μιας πιλοτικής εφαρμογής σε Ruby On Rails.

4.1 Περιγραφή πιλοτικής υλοποίησης

Η πιλοτική εφαρμογή αφορά μια ηλεκτρονική εφημερίδα. Όπως και σε μια συμβατική εφημερίδα έτσι και στην ηλεκτρονική υπάρχουν δύο ήδη δρώντων:

Δημοσιογράφοι

Αναγνώστες.

Οι λειτουργίες που θα υλοποιηθούν είναι:

Εισαγωγή χρήστη – δημοσιογράφου

Οι δημοσιογράφοι θα είναι εγγεγραμμένοι χρήστες και θα υπάρχει μηχανισμός ελέγχου ταυτότητας που θα πιστοποιεί ότι ο χρήστης είναι δημοσιογράφος, ώστε να του επιτρέπεται να διαχειρίζεται άρθρα.

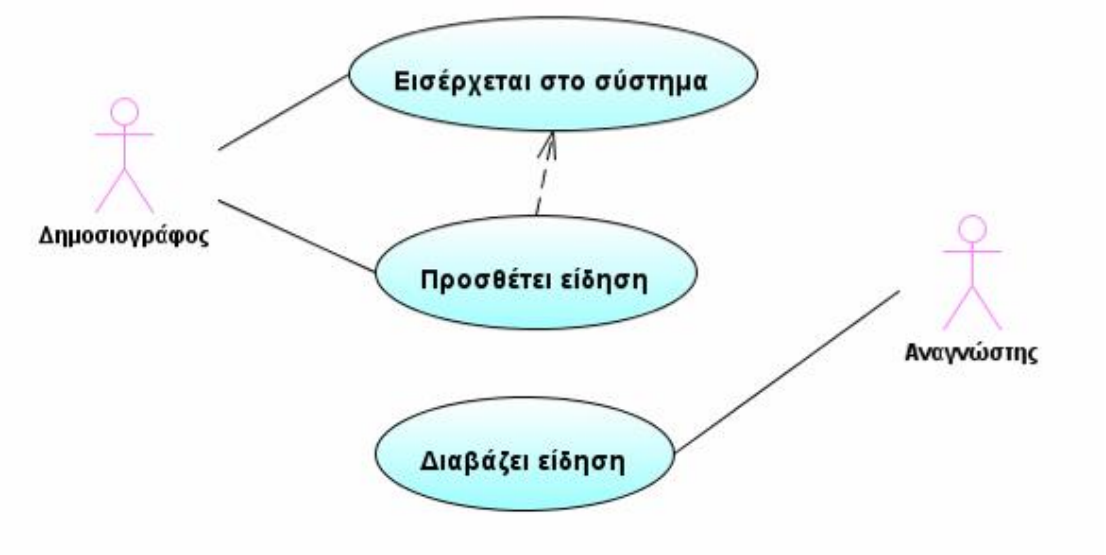
1) Διαχείριση άρθρων – ειδήσεων

Θα υπάρχουν οθόνες αλλά και αντίστοιχες λειτουργικές μονάδες που θα παρέχουν την δυνατότητα στους δημοσιογράφους, εφόσον έχουν εισέλθει πρώτα στο σύστημα, να διαχειρίζονται τα διάφορα άρθρα – ειδήσεις.

2) Ανάγνωση άρθρων – ειδήσεων

Οι απλοί χρήστες, χωρίς να χρειάζεται να εισέλθουν στο σύστημα, θα μπορούν να βλέπουν τα διάφορα άρθρα.

Οι παραπάνω λειτουργίες συνοψίζονται στο ακόλουθο διάγραμμα περιπτώσεων χρήσης.



Εικόνα 3 Διάγραμμα περιπτώσεων χρήσης πιλοτικής εφαρμογής

4.2 Εργαλεία ανάπτυξης

Για την εγκατάσταση και λειτουργία του Ruby on Rails θα χρειαστούν τα ακόλουθα στοιχεία:

Ένας Ruby interpreter. Το Rails είναι γραμμένο σε Ruby. Συνεπώς σε Ruby θα πρέπει να είναι γραμμένες και οι εφαρμογές.

Το Ruby on Rails ως framework.

Ορισμένες επιπλέον βιβλιοθήκες.

Μια βάση δεδομένων.

Επιπλέον, ανάλογα με την περίπτωση μπορεί να προστεθεί και ένας production web server με τον αντίστοιχο κώδικα υποστήριξης για την ομαλή λειτουργία του Rails.

Στην παρούσα εργασία, θα αναλυθεί ο τρόπος εγκατάστασης του Rails στο λειτουργικό περιβάλλον των Windows, χωρίς να σημαίνει ότι δε μπορεί με αντίστοιχο τρόπο να εγκατασταθεί και σε άλλα λειτουργικά συστήματα.

Ο Curt Hibbs συγκέντρωσε τα απαραίτητα συστατικά στοιχεία για την εγκατάσταση του Rails στο λεγόμενο InstantRails, ώστε να διευκολύνει την διαδικασία εγκατάστασης, και είναι τα εξής:

Το Ruby Interpreter

Ruby On Rails

MySQL

Apache web server μαζί με τον κώδικα υποστήριξής του.

Scite, πρόγραμμα για ευκολότερη σύνταξη του κώδικα

Διάφορα άλλα απαραίτητα στοιχεία.

4.3 Εγκατάσταση εργαλείων ανάπτυξης

Δημιουργία φακέλου εγκατάστασης του InstantRails.

Επίσκεψη στην ιστοσελίδα του InstantRails[13] με σκοπό το «κατέβασμα» της τελευταίας έκδοσης τύπου .zip, το οποίο θα συμπεριληφθεί στον φάκελο εγκατάστασης.

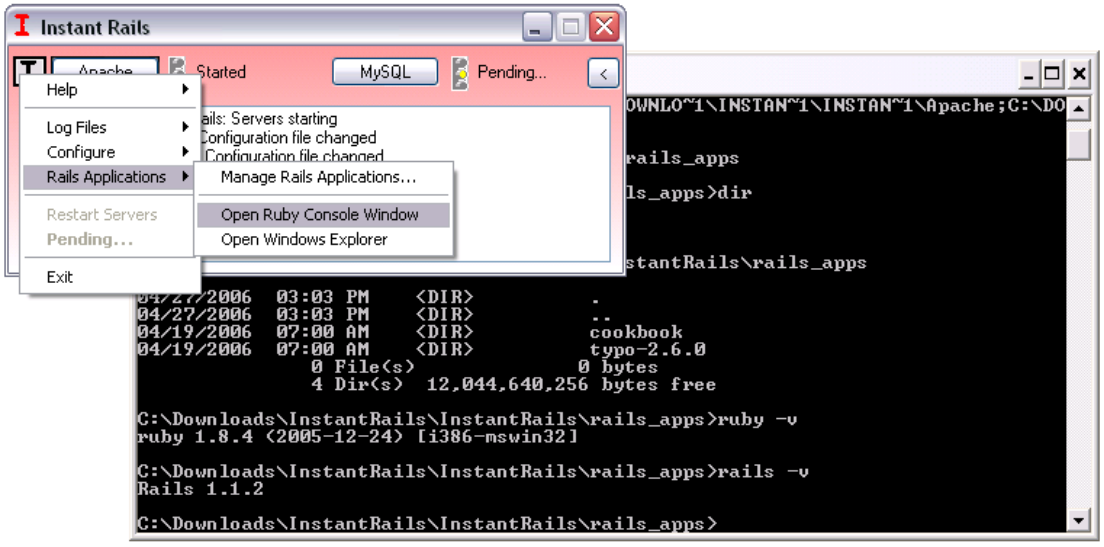
Unzip το αρχείο σε περίπτωση που αυτό δε γίνει αυτόματα.

Μέσα στον φάκελο εγκατάστασης, πραγματοποιούμε διπλό κλικ στο εικονίδιο του InstantRails.

Σε περίπτωση που εμφανιστεί ένα ερώτημα σχετικά με την επαναδημιουργία των αρχείων διαμόρφωσης, επιλέγουμε OK.

Σε περίπτωση που εμφανιστεί ειδοποίηση ασφαλείας σχετικά με τον Apache, ο ασφαλής τρόπος είναι να παρεμποδιστεί.

[13] http://instantrails.rubyforge.org/

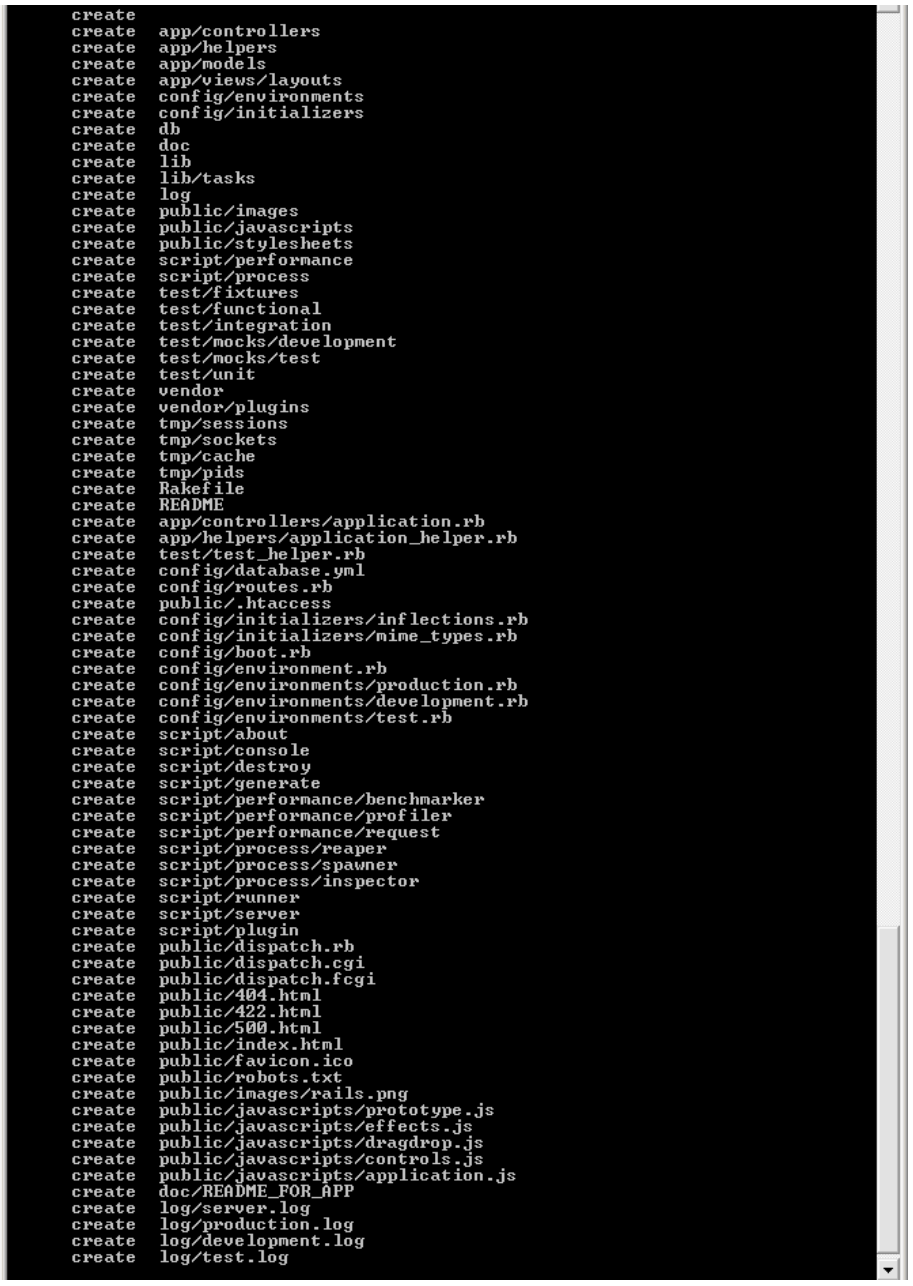


Εικόνα 4 Αναπαράσταση εγκατάστασης εργαλείων ανάπτυξης

Θα πρέπει να εμφανιστεί ένα μικρό παράθυρο InstantRails. Αυτό μπορεί να χρησιμοποιηθεί για την παρατήρηση και τον έλεγχο των εφαρμογών του Rails. Μπορεί επίσης να χρησιμοποιηθεί ένα παράθυρο κονσόλας. Για να εκκινηθεί, κάνουμε κλικ στο πλήκτρο I, στην άνω αριστερή γωνία του InstantRails παραθύρου. Από το μενού, επιλέγουμε το Rails Applications… και στη συνέχεια το Open Ruby Console Window. Θα πρέπει να εμφανιστεί ένα pop-up παράθυρο εντολών σαν αυτό του σχήματος , οπού το path θα οδηγεί στο rails_apps φάκελο. Σε αυτό το σημείο, μπορούμε να ελέγξουμε ποια έκδοση Ruby On Rails χρησιμοποιούμε, πληκτρολογώντας τις εντολές ruby –v και rails –v.

4.4 Επεξήγηση τρόπου υλοποίησης

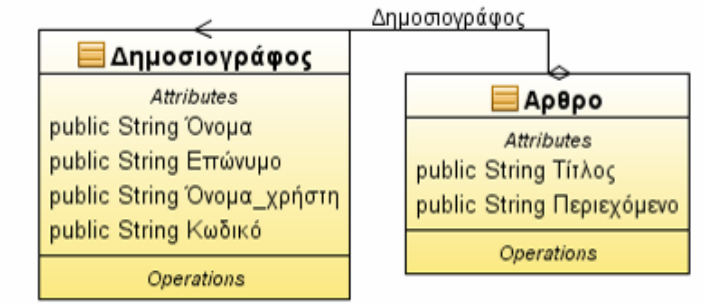
Δημιουργία αρχικής μορφής – “rails demo”



Εικόνα 5 Δημιουργία αρχικής εφαρμογής

Δημιουργία του μοντέλου – “ruby script\generate scaffold …”

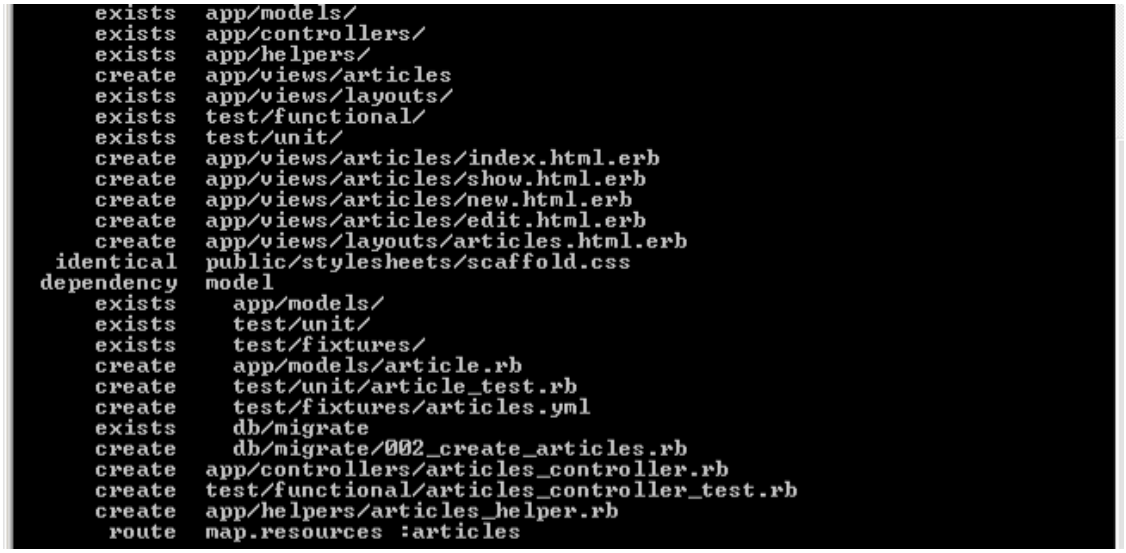
Στην πιλοτική εφαρμογή χρειαζόμαστε δύο οντότητες, τον Δημοσιογράφο και το άρθρο. Η δομή και η συσχέτιση αυτών φαίνεται στην «Εικόνα 6 **Διάγραμμα κλάσεων πιλοτικής εφαρμογής**». Για την δημιουργία του σχετικού κώδικα καλούμε το script generate scaffold, το οποίο θα δημιουργήσει την αρχική υλοποίηση με βάση την αρχιτεκτονική MVC.



Εικόνα 6 Διάγραμμα κλάσεων πιλοτικής εφαρμογής

```
C:\demo>ruby script\generate scaffold Journalist first_name:string last_name:str
ing user_name:string password:string
      exists  app/models/
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/journalists
      exists  app/views/layouts/
      exists  test/functional/
      exists  test/unit/
      create  app/views/journalists/index.html.erb
      create  app/views/journalists/show.html.erb
      create  app/views/journalists/new.html.erb
      create  app/views/journalists/edit.html.erb
      create  app/views/layouts/journalists.html.erb
      create  public/stylesheets/scaffold.css
  dependency  model
      exists    app/models/
      exists    test/unit/
      exists    test/fixtures/
      create    app/models/journalist.rb
      create    test/unit/journalist_test.rb
      create    test/fixtures/journalists.yml
      create    db/migrate
      create    db/migrate/001_create_journalists.rb
      create  app/controllers/journalists_controller.rb
      create  test/functional/journalists_controller_test.rb
      create  app/helpers/journalists_helper.rb
       route  map.resources :journalists
```

Εικόνα 7 Δημιουργία αρχικής αρχιτεκτονικής MVC για τον Δημοσιογράφο



Εικόνα 8 Δημιουργία αρχικής αρχιτεκτονικής MVC για το Άρθρο

Γενικές πληροφορίες σχετικά με τον κώδικα

Για την καλύτερη κατανόηση του κώδικα πρέπει να παρουσιαστούν πρώτα κάποιες βασικές εντολές που χρησιμοποιούνται. Η παρουσίαση των εντολών αυτών θα ξεκινήσει από την **παρουσίαση των απλών δομών** που υπάρχουν σε όλες τις γλώσσες προγραμματισμού, εν συνεχεία θα παρουσιαστούν κάποιες εντολές που παρέχει το ruby on rails και σχετίζονται με την **διαχείριση της βάσης δεδομένων** και τέλος θα παρουσιαστούν και κάποιες άλλες που σχετίζονται με την **επικοινωνία με τον χρήστη**. Τέλος θα γίνει μια σύντομη αναφορά σχετικά με **τα erb αρχεία**, που αντιστοιχούν στο view layer του Ruby On Rails.

Παρουσίαση των απλών δομών

Τα ονόματα τοπικών μεταβλητών, παραμέτρων μεθόδων και ονόματα μέθοδο πρέπει να ξεκινάνε με πεζά γράμματα ή με κάτω παύλα (underscore). Σε περίπτωση που θέλουμε να έχουμε κάποιο όνομα με πολλαπλές λέξεις πρέπει να διαχωρίζονται με κάτω παύλα. Για παράδειγμα journalist, delete_journalist.

Οι μεταβλητές ξεκινάνε με @. Σε περίπτωση που είναι global πρέπει να ξεκινάνε με @@. Για παράδειγμα `@journalist, @@journalists`

Τα ονόματα κλάσεων, modules και σταθερών πρέπει να ξεκινάνε με κεφαλαίο γράμμα. Εάν αποτελούνται από πολλαπλές λέξεις πρέπει να χρησιμοποιείται το πρότυπο init cap. Για παράδειγμα Journalist.

Υπάρχουν οι βοηθητικές μέθοδοι μετατροπής τύπων `to_s` μετατρέπει σε string, `to_i` μετατρέπει σε integer και `to_a` μετατρέπει σε array

Τα σχόλια στην Ruby ξεκινάνε με `#`

Στην Ruby υπάρχουν δυο είδη συλλογών. Οι πίνακες και οι χάρτες. Οι μεν πίνακες έχουν ως κλειδί για κάθε στοιχείο του πίνακα τον αριθμό της θέσης της ενώ ο χάρτης έχει το κλειδί που του έχει αντιστοιχιστεί.

Παράδειγμα πίνακα

```
categories=[‘Πολιτικά’, ‘Οικονομικά’]
categories[1]=’Economics’
```

Παράδειγμα χάρτη

```
categories={
     :politics => ‘Πολιτικά’
     :economics => ‘Οικονομικά’
}
categories[:economics]=’Economics’
```

Να σημειωθεί ότι είναι εύκολο να προστεθεί μια νέα τιμή σε ένα πίνακα με χρήση της εντολής build.

Για παράδειγμα

```
newValue=categories.build
newValue=’Πολιτιστικά’
```

Εφόσον ήδη έχουν παρουσιαστεί τα στοιχειώδη της γλώσσας θα πρέπει να παρουσιαστούν οι βασικές δομές ελέγχου, if clause, block, case clause, until loop. while loop, for loop, each loop

Παράδειγμα if clause

```
if count>100
    flash[:note]=’Υπάρχουν πάρα πολλές εγγραφές!’
```

```
elseif count>10
    flash[:note]='Υπάρχουν κι άλλες εγγραφές!'
else
   flash[:note]='Δεν είναι και τόσες πολλές οι εγγραφές!'
end
```

Παράδειγμα block, σειριακές εντολές

```
{
      journalist=Journalist.new
      journalist.username='username'
      journalist.password='pass'
      journalist.save
}
```

ισοδύναμα

```
do
      journalist=Journalist.new
      journalist.username='username'
      journalist.password='pass'
      journalist.save
end
```

Παράδειγμα case clause

```
case count
      when 0 .. 9
            'Δεν είναι και τόσες πολλές οι εγγραφές!'
      when 10 .. 99
            'Υπάρχουν κι άλλες εγγραφές!'
      else
            'Υπάρχουν πάρα πολλές εγγραφές!'
end
```

Παράδειγμα until loop. Τυπώνει όλους τους ακέραιους αριθμούς από μηδέν έως και δέκα.

```
counter=0
until counter=11 do
      print counter,"\n"
      counter=counter+1
```

```
end
```

εναλλακτικά

```
counter=0
begin
      print counter,"\n"
      counter=counter+1
end until counter=10
```

Παράδειγμα της while loop. Τυπώνει όλους τους αριθμούς που μπορεί να πάρει η μεταβλητή counter όσο η τιμή της είναι από μηδέν [αρχική τιμή] μέχρι και δέκα. Δηλαδή θα τυπώσει τους αριθμούς '0','1','2','3','4','5','6','7','8','9','10'.

```
counter=0
until counter<=10 do
      print count,"\n"
      counter=counter+1
end
```

Παράδειγμα της for loop. Τυπώνει όλους τους αριθμού που υπάρχουν στον πίνακα [0 .. 10], δηλαδή όλους τους ακεραίους από μηδέν μέχρι δέκα.

```
for counter in [0 .. 10]
      print count,"\n"
end
```

Παράδειγμα της each loop. Τυπώνει κάθε ένα στοιχείο του πίνακα categories.

```
categories.each do |category|
      print category
end
```

Εντολές διαχείρισης βάσης δεδομένων

Ένα από τα σημαντικότερα χαρακτηριστικά του Ruby On Rails είναι η ευκολία επικοινωνίας με την βάση δεδομένων. Οι εντολές που χρησιμοποιούνται είναι αυτές που φαίνονται στον ακόλουθο πίνακα.

Μέθοδος	**Περιγραφή**	**Παραδείγματα**
find	Κάνει query στην βάση	`AClass.find(:all)` επιστρέφει όλες τις εγγραφές
		`AClass.find(params[:id])`

	δεδομένων για ανάκτηση κατάλληλων εγγραφών Να σημειωθεί ότι η λέξη-κλειδί :all σημαίνει ότι το αποτέλεσμα θα είναι με όλα τα δυνατά αποτελέσματα, ενώ αν αντιστοιχα χρησιμοποιηθεί η :first θα επιστραφεί η πρώτη κατάλληλη εγγραφή που θα βρεθεί.	επιστρέφει αυτήν που έχει κλειδί με τιμή που δίνεται από την παράμετρο id
		`AClass.find(:all, :conditions => {:propert1 => "SampleValue",:property2 =>2})` επιστρέφει όσες εγγραφές έχουν τιμή "SampleValue" για το property1 και 2 για το property2
		`AClass.find(:all, :conditions => {:property1 => 1..3})` επιστρέφει όσα έχουν τιμή στο property1 μεταξύ 1 και 3
		`AClass.find(:all, :conditions => {:property1 => [1,3,8})` επιστρέφει όσα έχουν τιμή στο property1 κάποια μεταξύ των 1 3 και 8
		`AClass.find(:all, :conditions => ["property1 = :p1 AND property2 = :p2 ",{ :p1 => 3, :p2 =>"SampleValue" }])` επιστρέφει όσες εγγραφές έχουν τιμή στο property1 ότι έχει περαστεί στην παράμετρο p1, συγκεκριμένα στην περίπτωση μας 3 και στο property2 τιμή "SampleValue"
destroy	Διαγραφή σχετικής εγγραφής	`classInstance.destroy`
New	Δημιουργία καινούριου instance	`classInstance=AClass.new(:property1=>”P1”)`
save	Αποθήκευση αλλαγών στην βάση	`classInstance.save`

	δεδομένων	
update_attributes	Ενημερώνει τα πεδία μου περιγράφονται μόνο	`newInstance.update_attributes(params[:property1], params[:property2])`

Εντολές επικοινωνίας με τον χρήστη

Στον ακόλουθο πίνακα παρουσιάζονται οι εντολές που χρησιμοποιούνται για την επικοινωνία με τον χρήστη.

Εντολή	Περιγραφή	Παραδείγματα

redirect_to	Προωθεί σε κάποια άλλη ενέργεια. Να σημειωθεί ότι η ενέργεια αντιστοιχίζεται σε κάποια οθόνη-σελίδα	`redirect_to([@journalist,@article])`
		Παραπέμπει σε σελίδα που θα παρουσιάζεται το άρθρο που ορίζει η μεταβλητή article και ανήκει στον δημοσιογράφο που ορίζει η μεταβλητή journalist
		
		`redirect_to :action => "edit"`
		Παραπέμπει στο edit view

respond_to	Δηλώνει το που θα ανακατευθυνθεί ο χρήστης, ανάλογα με του τι είδους είσοδο αναμένει. Εάν περιμένει html ή xml ή πιθανόν κάτι άλλο.	`respond_to do |format|`
		`format.html {`
		`redirect_to([@journalist,@article])`
		`}`
		`format.xml => {`
		`render`
		`:xml => [@journalist,@article]`
		`}`
		`end`
		Το συγκεκριμένο παράδειγμα δημιουργεί ένα alias για την εντολή redirect_to και την ονομάζει format.

		Ορίζει το τι θα δει ο χρήστης στην περίπτωση που επικοινωνεί μέσω html [format.html] Ορίζει το τι θα δει ο χρήστης στην περίπτωση που επικοινωνεί μέσω xml [format.xml]

flash	Προσωρινό πέρασμα τιμών από τον controller στο view	Στην μεριά του contoller κάνουμε απόδοση τιμών
		`flash[:notice] = "Hello world!"`
		Στην μεριά του view κάνουμε έλεγχο ότι έχει οριστεί τιμή και την τυπώνουμε
		`<% if flash[:notice] %>`
		`<div class="notice">`
		`<%=flash[:notice] %>`
		`</div>`
		`<% end %>`

session	Διαχείριση τιμών για μεταβλητές καθ' όλη την διάρκεια της συνεδρίας.	Απόδοση τιμής
		`session[:attribute1]=”value”`
		Χρήση τιμής, πχ εκτύπωση στο view
		`<%= session[:attribute1]%>`

Να σημειωθεί ότι:

1) Χαρακτηριστική περίπτωση επικοινωνίας μέσω xml αποτελούν τα web services
2) Όταν χρησιμοποιούνται παρενθέσεις '[]' ουσιαστικά δημιουργείται πίνακας
3) Όταν δεν ορίζεται κάποιο συγκεκριμένο action γίνεται χρήση του action “index”

Περί erb

Τα αρχεία erb είναι το αντίστοιχο των δυναμικών σελίδων, jsp ,php ,asp κ.ο.κ. Δηλαδή είναι δομημένα αρχεία html που σε συγκεκριμένα σημεία όπου απαιτείται δυναμικό περιεχόμενο ή επεξεργασία για την παραγωγή του υπάρχει σχετικός κώδικας ruby που εσωκλειεται σε <% %>. Συγκεκριμένα όταν πρέπει να τυπωθεί μόνο η τιμή μιας μεταβλητής τότε απλά γράφουμε στο κατάλληλο σημείο <%=@variable %>.

Να σημειωθεί ότι καλό είναι να χρησιμοποιείται η μέθοδος h. Η συγκεκριμένη μέθοδος κανονικοποιεί το κείμενο έτσι ώστε εαν περιέχει κώδικα να μην εκτελείται αλλά να τυπώνεται. Αυτό είναι πολύ σημαντικό ώστε να αποφεύγουμε script attack.
Τέλος υπάρχουν και κάποιες άλλες εντολές που δημιουργούν κατάλληλα tags για συνδέσμους που καλούν συγκεκριμένα actions, ή που σχεδιάζουν φόρμες.

Παράδειγμα link:

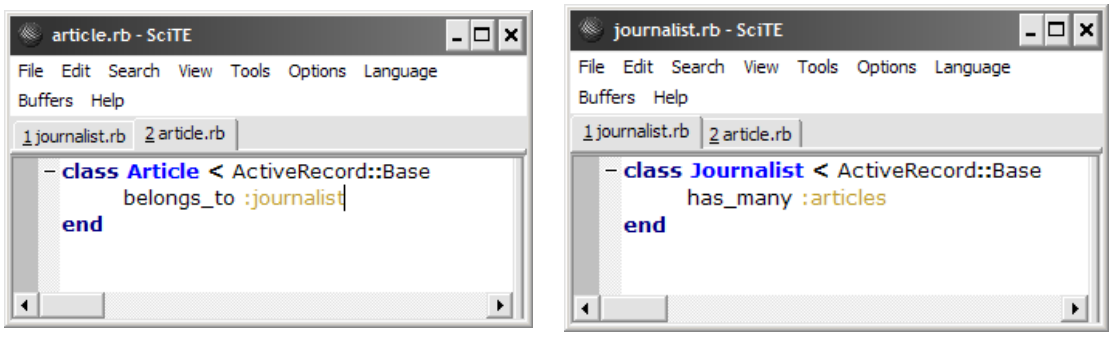
```
<%= link_to "Αποθήκευση", :actions => "save" %>
```

Παράδειγμα φόρμας:

```
<% form_for([@journalist,@article]) do |f| %>
<%= f.text_field :title %>
<%= f.text_area :content %>
<%= f.check_box :is_active %>
<% end %>
```

Συσχέτιση κλάσεων

Να σημειωθεί ότι μέσω της συγκεκριμένης διαδικασίας δεν σχετίζονται απόλυτα αυτές οι δύο οντότητες, για αυτό θα χρειαστεί να κάνουμε κάποιες μικρές αλλαγές στον κώδικα. Θα προσθέσουμε στον μεν Δημοσιογράφο ότι έχει πολλά Άρθρα [has_many :articles] ενώ στο δε Άρθρο θα προσθέσουμε ότι ανήκει σε Δημοσιογράφο [belongs_to :journalist].



Εικόνα 9 Τρόπος συσχέτισης κλάσεων

Αντίστοιχα αυτή η συσχέτιση πρέπει να οριστεί και στην δρομολόγηση [/config/routes.rb], ώστε να υπάρχει σχετική ροή από τον «ιδιοκτήτη» προς το κτήμα. Όπως θα φανεί παρακάτω έτσι καταφέρνουμε να φτιαχτούν οθόνες όπου εύκολα θα συνδέεται ο δημοσιογράφος κι από την οθόνη του προφίλ του θα μπορεί να μεταφέρεται σε οθόνες καταχώρισης άρθρων που αυτόματα θα ορίζονται ως δικά του. Η αλλαγή που γίνεται σε αυτό το σημείο φαίνεται στην ακόλουθη **Εικόνα 10 Προσδιορισμός σχέσης Δημοσιογράφου με Άρθρα.**

routes.rb

```
ActionController::Routing::Routes.draw do |map|
  # comment out the following two lines
  #map.resources :articles
  #map.resources :journalists

  #Add the following line
  map.resources :journalists, :has_many => :articles
# The priority is based upon order of creation: first created -> highest priority.

  # Sample of regular route:
  #   map.connect 'products/:id', :controller => 'catalog', :action => 'view'
```

Εικόνα 10 Προσδιορισμός σχέσης Δημοσιογράφου με Άρθρα

Άλλο ένα σημείο στο οποίο πρέπει να γίνουν τροποποιήσεις είναι το Control κομμάτι των Άρθρων, όπου πρέπει σε όλες τις μεθόδους να αξιοποιείται μια παράμετρος που προσδιορίζει ποιού Δημοσιογράφου είναι το ή τα άρθρα που διαχειρίζονται. Οι τροποποιήσεις αυτές είναι ουσιαστικά οι δύο γραμμές κώδικα κάθε μεθόδου. Όπως φαίνεται στην ακόλουθη εικόνα:

articles_controller.rb - SciTE

File Edit Search View Tools Options Language Buffers Help

1 index.html.erb | 2 new.html.erb | 3 articles_controller.rb | 4 index.html.erb | 5 show.html.erb | 6 edit.html.erb

```ruby
class ArticlesController < ApplicationController
  # GET /articles
  # GET /articles.xml
  def index
    @journalist = Journalist.find(params[:journalist_id])
    @articles = @journalist.articles

    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render :xml => @articles }
    end
  end

  # GET /articles/1
  # GET /articles/1.xml
  def show
      @journalist = Journalist.find(params[:journalist_id])
      @article = @journalist.articles.find(params[:id])
    respond_to do |format|
      format.html # show.html.erb
      format.xml  { render :xml => @article }
    end
  end

  # GET /articles/new
  # GET /articles/new.xml
  def new
    @journalist = Journalist.find(params[:journalist_id])
    @article = @journalist.articles.build

    respond_to do |format|
      format.html # new.html.erb
      format.xml  { render :xml => @article }
    end
  end

  # GET /articles/1/edit
  def edit
      @journalist = Journalist.find(params[:journalist_id])
      @article = @journalist.articles.find(params[:id])
  end

  # POST /articles
  # POST /articles.xml
  def create
    @journalist = Journalist.find(params[:journalist_id])
    @article = @journalist.articles.build(params[:article])

    respond_to do |format|
      if @article.save
        flash[:notice] = 'Article was successfully created.'
        format.html { redirect_to([@journalist,@article]) }
        format.xml  { render :xml => [@journalist,@article], :status => :created, :location => [@journalist,@article] }
        format.html { render :action => "new" }
        format.xml  { render :xml => @article.errors, :status => :unprocessable_entity }
      end
    end
  end

  # PUT /articles/1
  # PUT /articles/1.xml
  def update
      @journalist = Journalist.find(params[:journalist_id])
      @article = @journalist.articles.find(params[:id])
    respond_to do |format|
      if @article.update_attributes(params[:article])
        flash[:notice] = 'Article was successfully updated.'
        format.html { redirect_to([@journalist,@article]) }
        format.xml  { head :ok }
      else
        format.html { render :action => "edit" }
        format.xml  { render :xml => @article.errors, :status => :unprocessable_entity }
      end
    end
  end

  # DELETE /articles/1
  # DELETE /articles/1.xml
  def destroy
      @journalist = Journalist.find(params[:journalist_id])
      @article = @journalist.articles.find(params[:id])
     @article.destroy

    respond_to do |format|
      format.html { redirect_to(@journalist) }
      format.xml  { head :ok }
    end
  end
end
```

Για να ολοκληρωθεί αυτή η συσχέτιση πρέπει να τροποποιηθεί και το View κομμάτι. Οι τροποποιήσεις αφορούν τον τρόπο εμφάνισης του Δημοσιογράφου και φαίνονται στις ακόλουθες εικόνες:

ndex.html.erb | 2 new.html.erb | 3 articles_controller.rb | 4 index.html.erb | 5 show.html.erb | 6 edit.html.erb

```
<h1>Listing articles</h1>
<table>
  <tr>
    <th>Title</th>
    <th>Content</th>
    <th>Journalist</th>
    <th colspan="3">Actions</th>
  </tr>
<% for article in @articles %>
  <tr>
    <td><%=h article.title %></td>
    <td><%=h article.content %></td>
    <td><%=h @journalist.first_name %>  <%=h @journalist.last_name %></td>
    <td><%= link_to 'Show', journalist_article_path(article.journalist,article) %></td>
    <td><%= link_to 'Edit', edit_journalist_article_path(article.journalist,article) %></td>
    <td><%= link_to 'Destroy', [article.journalist,article], :confirm => 'Are you sure?', :method => :delete %></td>
  </tr>
<% end %>
</table>
<br />
<%= link_to 'New article', new_journalist_article_path(@journalist) %>
```

Εικόνα 11 Τροποποιήσεις της index των Άρθρων

index.html.erb | 2 new.html.erb | 3 articles_controller.rb | 4 index.html.erb | 5 show.html.erb | 6 edit.html.erb

```
<p>
  <b>Title:</b>
  <%=h @article.title %>
</p>
<p>
  <b>Content:</b>
  <%=h @article.content %>
</p>
<p>
  <b>Journalist:</b>
  <%=h @article.journalist.first_name %>  <%=h @article.journalist.last_name %>
</p>
<%= link_to 'Edit', edit_journalist_article_path(@journalist,@article) %> |
<%= link_to 'Back', journalist_articles_path(@journalist) %>
```

Εικόνα 12 Τροποποιήσεις της show του Άρθρου

Εικόνα 13 Τροποποίηση της new του Άρθρου

Εικόνα 14 Τροποποίηση της edit του Άρθρου

Επίσης πρέπει να προστεθεί και μια καινούρια μέθοδος κι ένα αντίστοιχο view, μέσω του οποίου θα εμφανίζονται όλα τα άρθρα και θα είναι η κεντρική σελίδα η οποία θα είναι προσβάσιμη σε όλους. Οπότε προστίθεται το κομμάτι του κώδικα που φαίνεται παρακάτω στο app/controllers/articles_controller.rb καθώς φτιάχνεται και η app/views/articles/all.html.erb με το περιεχόμενο που επίσης φαίνεται ακολούθως.

Αλλά για να γίνει πιο εύχρηστο και να εμφανίζεται όταν ο χρήστης έχει επιλέξει το root url του site, πρέπει να γίνει map στο config/route.rb, όπως φαίνεται σε ακόλουθη εικόνα.

```
# GET /articles/all
 # GET /articles/all.xml
 def all
   @articles = Article.find(:all)
   respond_to do |format|
     format.html # all.html.erb
     format.xml  { render :xml => @articles }
   end
 end
```

Εικόνα 15 Κώδικας του action all

```
<script language=javascript type='text/javascript'>
    function showOrHide(articleId) {
        if (document.getElementById) {
            if(document.getElementById('content_'+articleId).style.display == '')
                document.getElementById('content_'+articleId).style.display = 'none';
            else
                document.getElementById('content_'+articleId).style.display = '';
        }else {
            alert('unsupported!!');
        }
    }
</script>
<h1>Articles</h1>
<ol>
    <% for article in @articles %>
        <li>
            <a href="javascript:showOrHide(<%=article.id%>)"><%=  h article.title%></a>
            <div id="content_<%=article.id%>" style="display:none" >
                <%=article.content%>
            <div>
        </li>
    <%end%>
</ol>
```

[14]

Εικόνα 16 Κώδικας της "all.html.erb"

```
map.root  :controller => 'articles', :action => 'all'
```

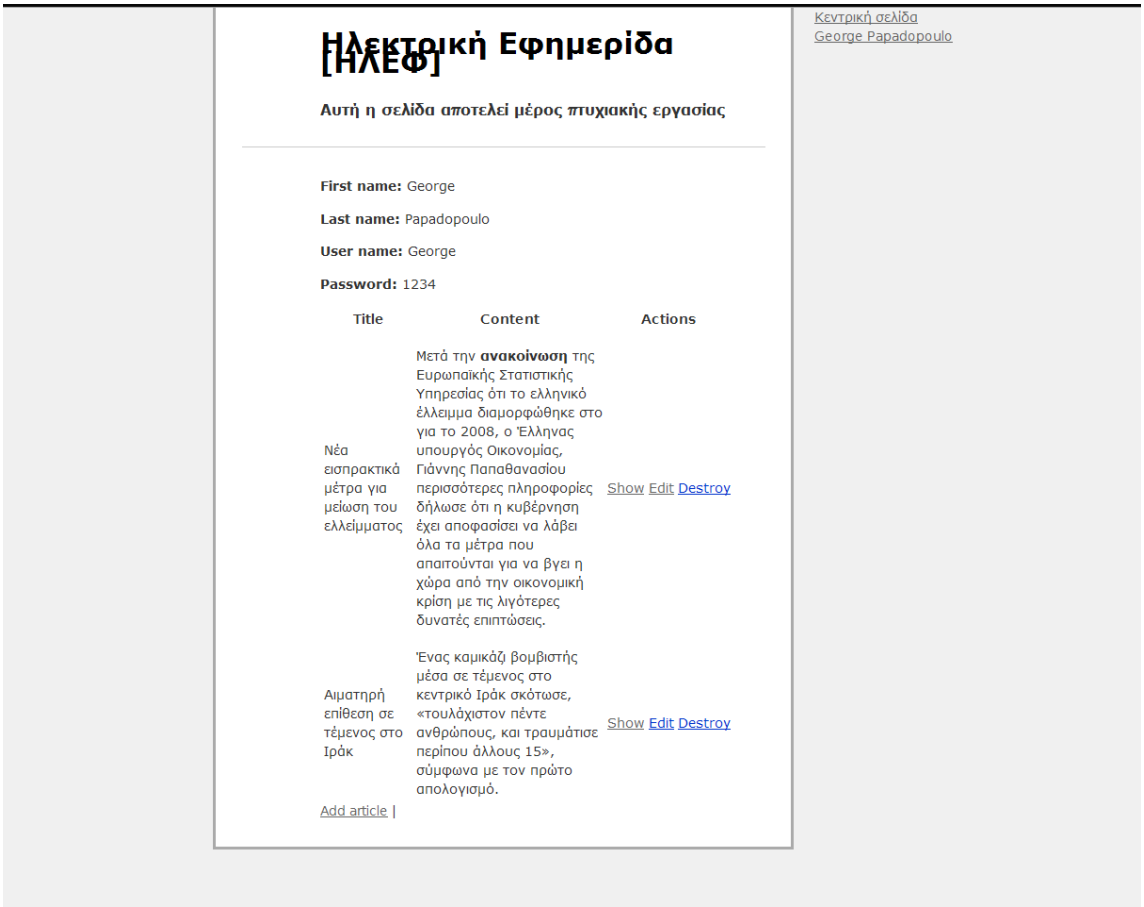
Εικόνα 17 Αλλαγές του κώδικα του routes.rb

[14] Έχει προστεθεί κι ένα javascript για να εμφανίζεται ή να κρύβεται το περιεχόμενο ενός άρθρου, όταν ο χρήστης πατάει πάνω στον τίτλο.

Τέλος πρέπει να προστεθούν στην οθόνη του προφίλ του Δημοσιογράφου [app/views/journalists/show.html.erb] ένας πίνακας με τα άρθρα που έχει γράψει αλλά και link που να διευκολύνουν την διαχείριση τους. Στις ακόλουθες εικόνες φαίνεται ο σχετικός κώδικας για τον πίνακα, αλλά και ένα παράδειγμα του πως μοιάζει η οθόνη του προφίλ του Δημοσιογράφου.

```
<table>
 <tr>
    <th>Title</th>
    <th>Content</th>
    <th colspan="3">Actions</th>
  </tr>
<% for article in @journalist.articles %>
 <tr>
    <td><%=h article.title %></td>
    <td><%= article.content %></td>
    <td><%= link_to 'Show', journalist_article_path(@journalist,article) %></td>
    <td><%= link_to 'Edit', edit_journalist_article_path(@journalist,article) %></td>
    <td><%= link_to 'Destroy', article, :confirm => 'Are you sure?', :method => :delete %></td>
  </tr>
 <% end %>
 </table>
<%= link_to 'Add article', new_journalist_article_path(@journalist)%> |
```

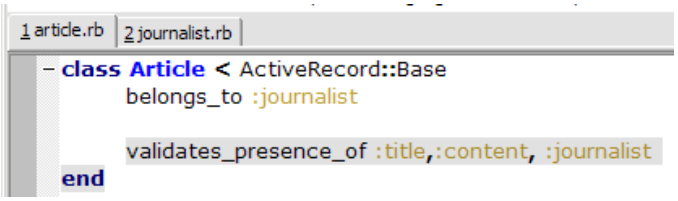
Εικόνα 18 Κώδικας για τη δημιουργία πίνακα



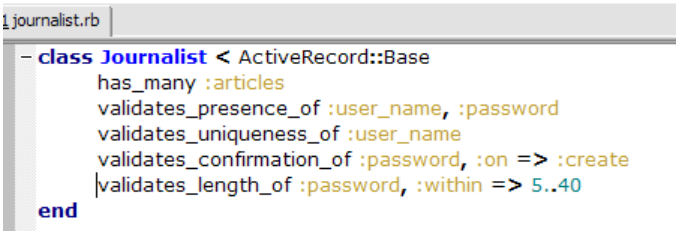
Εικόνα 19 Προφίλ δημοσιογράφου

Προσθήκη ελέγχων – validates_presence_of και άλλα

Εφόσον έχουν οριστεί οι κλάσεις καθώς και η συσχέτιση τους, πρέπει να προστεθούν κάποιοι έλεγχοι. Αυτοί οι έλεγχοι επιβεβαιώνουν την ορθότητα των δεδομένων που θα περαστούν στο σύστημα. Όλοι οι έλεγχοι είναι οδηγίες που ξεκινούν με “validates_” και φαίνονται στις ακόλουθες εικόνες:



Εικόνα 20 Έλεγχος υποχρεωτικών πεδίων Άρθρο



Εικόνα 21 Έλεγχος υποχρεωτικών πεδίων Δημοσιογράφου

Διαγραφή περιττού κώδικα

Τώρα πλέον είναι έτοιμη μια εφαρμογή με οθόνες προβολής, προσθήκης, επεξεργασίας και διαγραφής για Δημοσιογράφους και άρθρα. Το ζητούμενο της πιλοτικής υλοποίησης είναι να υλοποιηθούν κάποιες λίγο διαφορετικές λειτουργίες. Αυτές παρουσιάστηκαν ήδη στην Εικόνα 1 Διάγραμμα περιπτώσεων χρήσης πιλοτικής εφαρμογής, οπότε πρέπει να διαγραφούν κάποια περιττά κομμάτια κώδικα καθώς δεν χρειάζονται, καθώς και να προστεθεί το κομμάτι της «Εισόδου χρήστη στο σύστημα».

Συνοψίζοντας τα προς διαγραφή τμήματα αφορούν όλες τις οθόνες του Δημοσιογράφου εκτός από αυτήν της προβολής [show] η οποία θα επεκταθεί παρακάτω. Οπότε διαγράφονται τα αρχεία app/views/journalists/new.html.erb και app/views/journalists/edit.html.erb. Έτσι πλέον δε θα υπάρχει τρόπος μέσω του UI να προστεθεί Δημοσιογράφος ή να αλλαχθούν τα στοιχεία του, επίσης πρέπει να τροποποιηθεί η app/views/journalists/index.html.erb ώστε να κάνει redirect στην show.html.erb, διότι μόνο αυτή χρειάζεται να υπάρχει τελικά. Redirect γίνεται με την εντολή <% redirect_to journalist_path(@journalist)%>. Αυτό γίνεται ώστε να απλοποιηθεί η εφαρμογή, διότι διαφορετικά έπρεπε να δημιουργηθεί κατάλληλος μηχανισμός επαλήθευσης στοιχείων, έγκρισης κτλ. Παράλληλα πρέπει να διαγραφούν από τον Controller [journalists_controller.rb] οι μέθοδοι new, edit και destroy.

Προσθήκη δυνατότητας εισόδου χρήστη στην εφαρμογή

Όπως σε κάθε εφαρμογή όπου πρέπει να γίνει διαχωρισμός μεταξύ των χρηστών, έτσι και στην πιλοτική εφαρμογή πρέπει να δημιουργηθεί ένας μηχανισμός ελέγχου ταυτότητας, ώστε οι δημοσιογράφοι να εισέρχονται στο σύστημα και να έχουν μόνο αυτοί την δυνατότητα να διαχειρίζονται τα άρθρα τους. Να σημειωθεί ότι η συγκεκριμένη υλοποίηση βασίζεται στο παράδειγμα User authentication in Ruby on Rails του Kacper Cieśla (comboy)[15], μόνο που αναιρέθηκαν διάφορα κομμάτια ώστε να απλοποιηθεί.

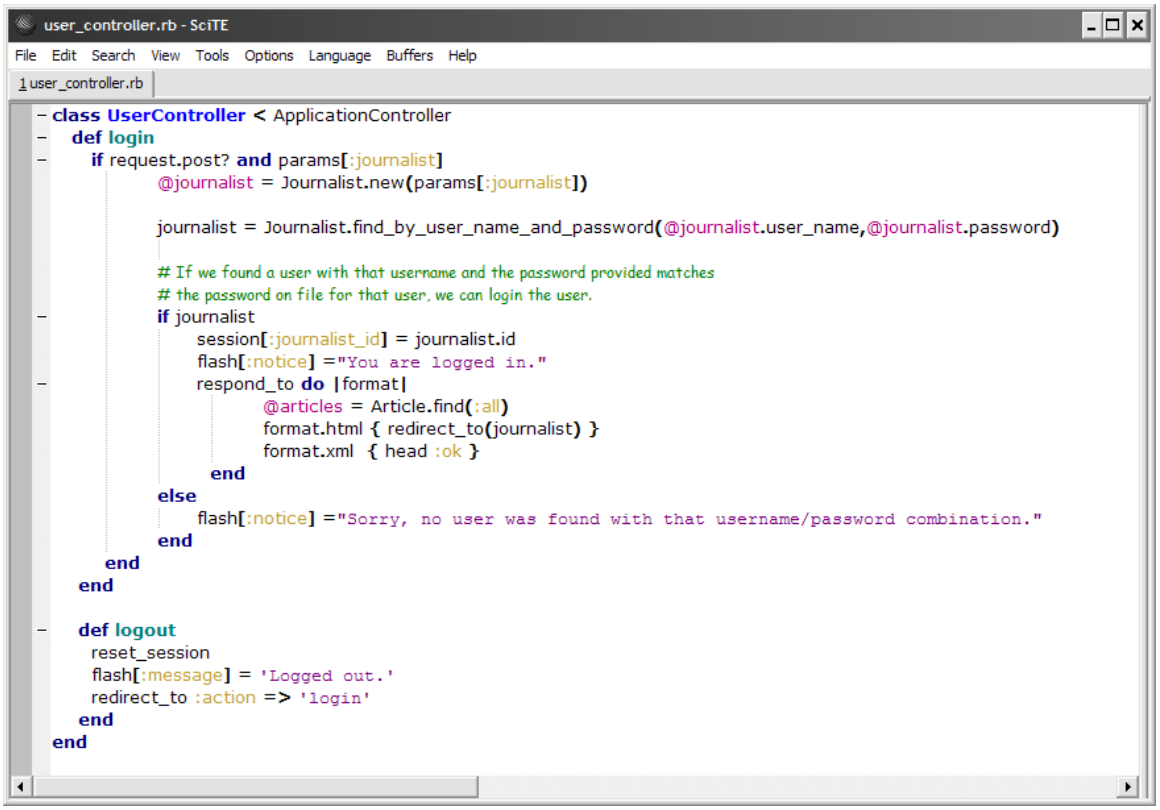
Η διαδικασία υλοποίησης της ξεκινάει με την δημιουργία του κατάλληλου Controller.

```
C:\myapp>ruby script\generate controller user login logout
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/user
      exists  test/functional/
      create  app/controllers/user_controller.rb
      create  test/functional/user_controller_test.rb
      create  app/helpers/user_helper.rb
      create  app/views/user/login.html.erb
      create  app/views/user/logout.html.erb
```

Εικόνα 22 Δημιουργία του Controller

Μετά πρέπει να τροποποιηθεί ο controller [user_controller.rb] όπως φαίνεται στην ακόλουθη εικόνα:

[15] http://codingbitch.com/p/comboy/User%20authentication%20in%20Ruby%20on%20Rails



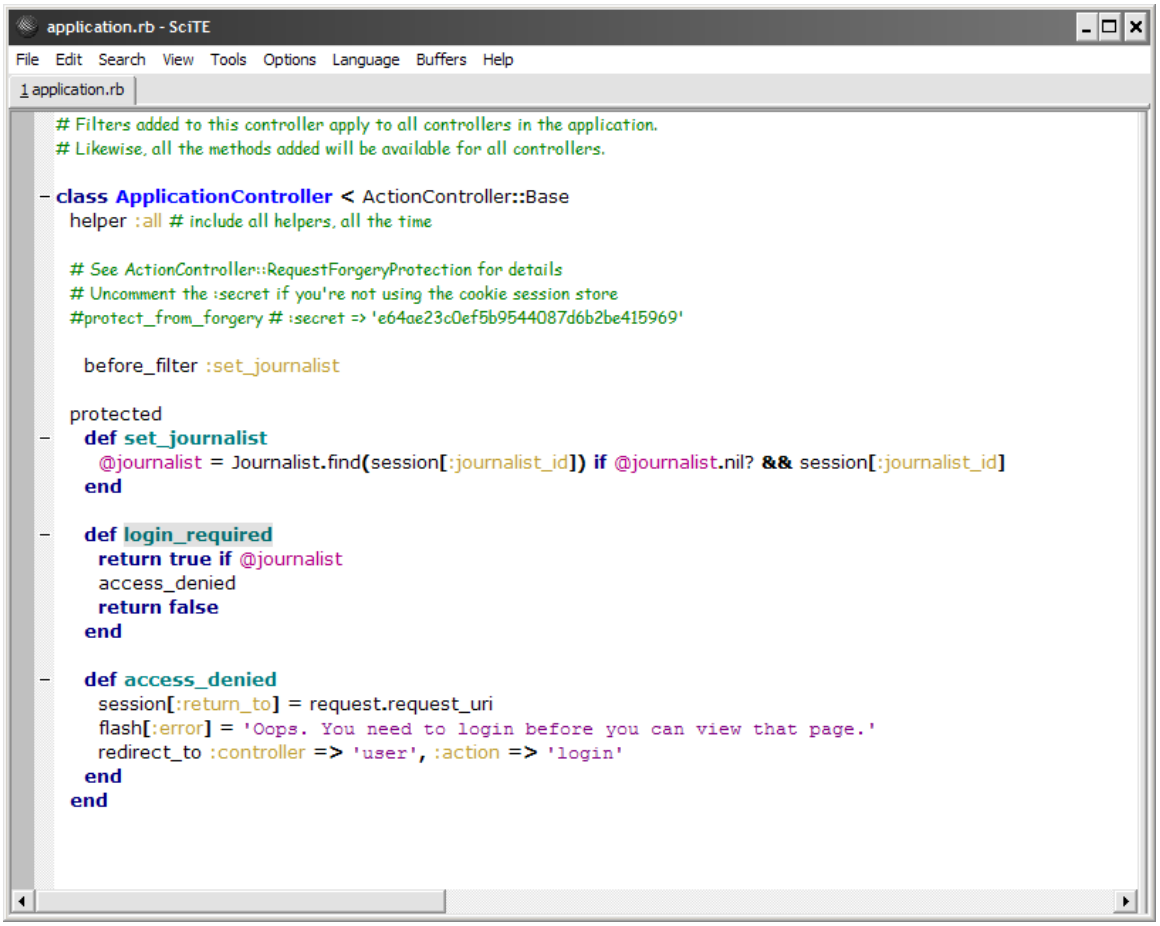
Εικόνα 23 Τροποποίηση του Controller

Επίσης πρέπει να αλλαχθεί και η login.html.erb, ώστε να εμφανίζει μια φόρμα καταχώρισης του username και του password

Εικόνα 24 Αλλαγή του login.html.erb

Αυτά όμως δεν αρκούν, αν δεν υπάρχει έλεγχος του κατά πόσον έχει ήδη εισέλθει ο χρήστης και μόνο τότε να του επιτρέπει να κάνει συγκεκριμένες ενέργειες, διαφορετικά να το παραπέμπει στην οθόνη εισόδου. Αυτό επιτυγχάνεται μέσω των εξής αλλαγών:

Τροποποιήσεις του ApplicationController



Εικόνα 25 Τροποποιήσεις του Application Controller

Προσθήκη ελέγχων στους controllers των Άρθρων και των Δημοσιογράφων

articles_controller.rb * SciTE

File Edit Search View Tools Options Language Buffers Help

1 articles_controller.rb *

```
class ArticlesController < ApplicationController
  before_filter :login_required, :only => :index
  # GET /articles
```

Εικόνα 26 Προσθήκη ελέγχου στον Controller των Άρθρων

journalists_controller.rb - SciTE

File Edit Search View Tools Options Language Buffers Help

1 journalists_controller.rb

```
class JournalistsController < ApplicationController
      before_filter :login_required

  def index
  @journalist = Journalist.find(session[:journalist_id])

  respond_to do |format|
    format.html # show.html.erb
    format.xml  { render :xml => @journalist }
  end
end
  # GET /journalists/1
  # GET /journalists/1.xml
  def show
    @journalist = Journalist.find(session[:journalist_id])

  respond_to do |format|
    format.html # show.html.erb
    format.xml  { render :xml => @journalist }
  end
end

  def logout
  session[:journalist_id]=nil
    flash[:notice] = "You've been logged out."
    redirect_to :controller => 'users', :action => 'login'
  end
end
```

Εικόνα 27 Προσθήκη ελέγχου στον Controller των Δημοσιογράφων

Με αυτές τις περιορισμένες αλλαγές η δυνατότητα εισόδου στο σύστημα για τους δημοσιογράφους έχει προστεθεί και πλέον η εφαρμογή έχει καλύψει τις βασικές λειτουργικές απαιτήσεις που είχαν τεθεί. Στην συνέχεια θα παρουσιαστούν κάποιες βελτιώσεις.

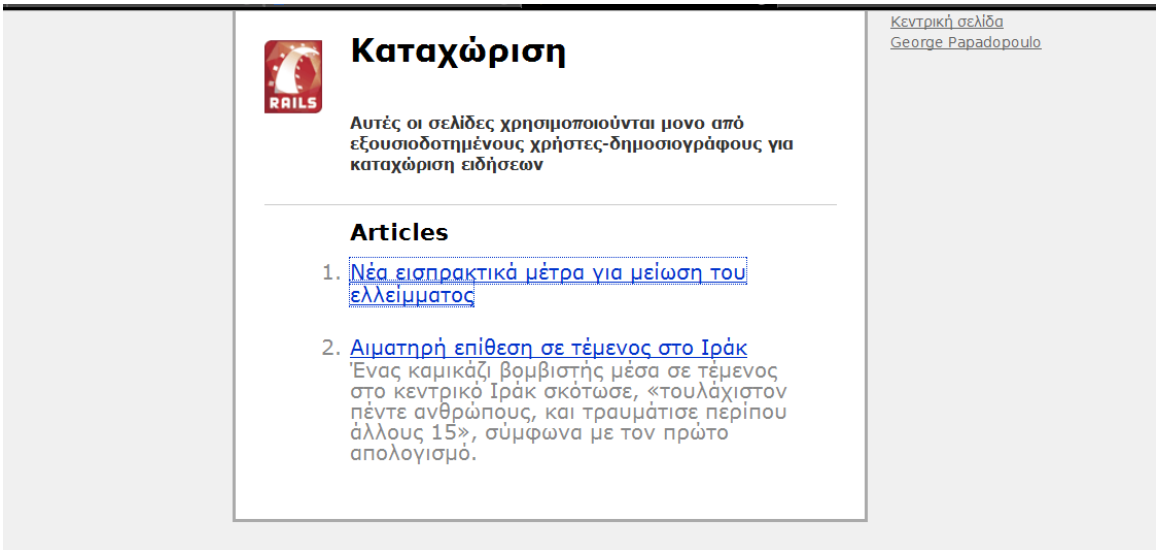
Μορφοποίηση του template

Ένα πολύ θετικό χαρακτηριστικό είναι ότι εξ' αρχής σε οδηγεί να δημιουργήσεις ένα template, ώστε και να μην επαναλαμβάνεις κώδικα HTML, αλλά και για να υπάρχει ομοιομορφία. Για τον λόγο αυτό πρέπει να τροποποιηθούν τα δύο template files, app/views/layouts/journalists.html.erb και app/views/layouts/articles.html.erb και μάλιστα θα είναι ακριβώς του ίδιου περιεχομένου[16]. Μέρος του κώδικα αλλά και δείγμα του πως φαίνεται παρουσιάζονται στις ακόλουθες εικόνες:

[16] Δυστυχώς δεν εντοπίστηκε τρόπος να αποφευχθεί η επανάληψη κώδικα μεταξύ αυτών των δύο αρχείων

```
<body>

  <div id="page">
    <div id="sidebar">
              <a href="/"> </a>
          <br>
      <%if (session[:journalist_id]==nil)%>
              <a href="/user/login">Login</a>
          <%else%>
              <%@journalist = Journalist.find(session[:journalist_id])%>
              <a href="/journalists/show/<%=@journalist.id%>">
                  <%=@journalist.first_name%> <%=@journalist.last_name%>
              </a>
          <%end%>
    </div>

    <div id="content">
      <div id="header">
        <h1>Ηλεκτρονική Εφημερίδα [ΗΛΕΦ]</h1>
      </div>

      <div id="about">
        <h3>Αυτή η σελίδα αποτελεί μέρος πτυχιακής εργασίας</h3>
        <div id="about-content" style="display: none"></div>
      </div>

      <div id="getting-started">
          <%= yield  %>
      </div>
    </div>
    <div id="footer"> </div>
  </div>

  <p style="color: green"><%= flash[:notice] %></p>

</body>
```

Εικόνα 28 Τροποποίηση των templates files: app/views/layouts/journalists.html.erb & app/views/layouts/articles.html.erb



Εικόνα 29 Δείγμα του «πώς φαίνεται» στον χρήστη

Προσθήκη δυνατότητας rss

Το rss είναι μια τεχνολογία που έχει γίνει απαραίτητο χαρακτηριστικό για οποιοδήποτε ιστοχώρο έχει τακτική ενημέρωση περιεχομένου. Είναι σχεδόν αυτονόητο για μια ειδησιογραφική πύλη, όπως ουσιαστικά είναι και η υλοποίηση της πιλοτικής εφαρμογής. Επίσης η προσθήκη αυτής της λειτουργίας βοηθάει στην προβολή της δυνατότητας του Ruby On Rails να παράγει XML. Να σημειωθεί ότι για την υλοποίηση αυτής της δυνατότητας αξιοποιήθηκαν οι πολύ χρήσιμες οδηγίες από το άρθρο «Creating an RSS feed in Ruby on Rails»[17]. Το μόνο επιπλέον αρχείο που πρέπει να δημιουργηθεί είναι το app/views/articles/rss.rxml. Στις εικόνες που ακολουθούν εμφανίζονται όλες οι αλλαγές που πρέπει να γίνουν, που είναι ουσιαστικά προσθήκη κώδικα.

[17] http://paulsturgess.co.uk/articles/show/13-creating-an-rss-feed-in-ruby-on-rails

```
# GET /articles/rss
# GET /articles/rss.xml
def rss
 @articles = Article.find(:all, :order => "id DESC")
 render :layout => false
 response.headers["Content-Type"] = "application/xml; charset=utf-8"
end
```

Εικόνα 30 Κώδικας του action rss

```
xml.instruct!

xml.rss "version" => "2.0", "xmlns:dc" => "http://purl.org/dc/elements/1.1/" do
 xml.channel do

   xml.title       "Rails News"
   xml.link        url_for :only_path => false, :controller => 'articles', :action => 'all'
   xml.description "Rails News is the implementation of.."

   @articles.each do |article|
     xml.item do
       xml.title       article.title
       xml.link        url_for :only_path => false, :controller => 'articles', :action => 'view', :id => article.id
       xml.description article.content
       xml.guid        url_for :only_path => false, :controller => 'articles', :action => 'view', :id => article.id
     end
   end

 end
end
```

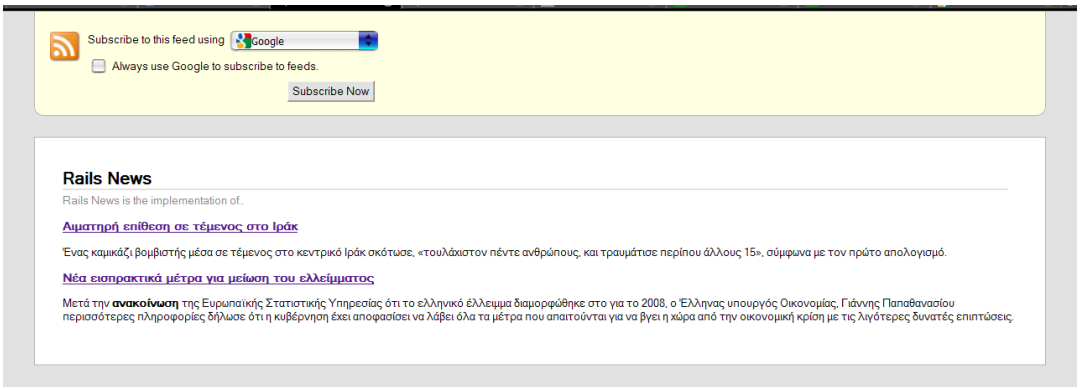
Εικόνα 31 Κώδικας του rss.rxml

```
<atom:link href='rss.xml' />
<link rel="alternate" type="application/rss+xml"  href="rss.xml" />
```

Εικόνα 32 Πρόσθετος κώδικας της all.html.erb

```
map.map 'rss.xml',:controller => 'articles', :action => 'rss'
```

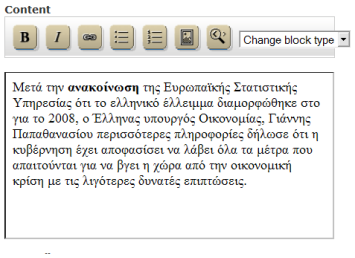
Εικόνα 33 Πρόσθετος κώδικας του config/routes.rb



Εικόνα 34 Δείγμα του rss που δημιουργείται

Εμπλουτισμένος επεξεργαστής κειμένου – WYIWYG

Πολλές φορές το απλό κείμενο γίνεται κουραστικό, για αυτόν το λόγο χρειάζεται να υπάρχει η δυνατότητα μορφοποίησης του. Αυτό επιτυγχάνεται ακολουθώντας τις οδηγίες που βρέθηκαν στο άρθρο «HTML Editor or Rich Text Editor for Your Ruby on Rails Apps»[18], δηλαδή απλά αξιοποιήθηκε ο έτοιμος κώδικας. Το αποτέλεσμα αυτού φαίνεται στην ακόλουθη εικόνα:



Εικόνα 35 Αποτέλεσμα αξιοποίησης του κώδικα μορφοποίησης κειμένου

4.5 Σχολιασμός τρόπου υλοποίησης - Πλεονεκτήματα και μειονεκτήματα

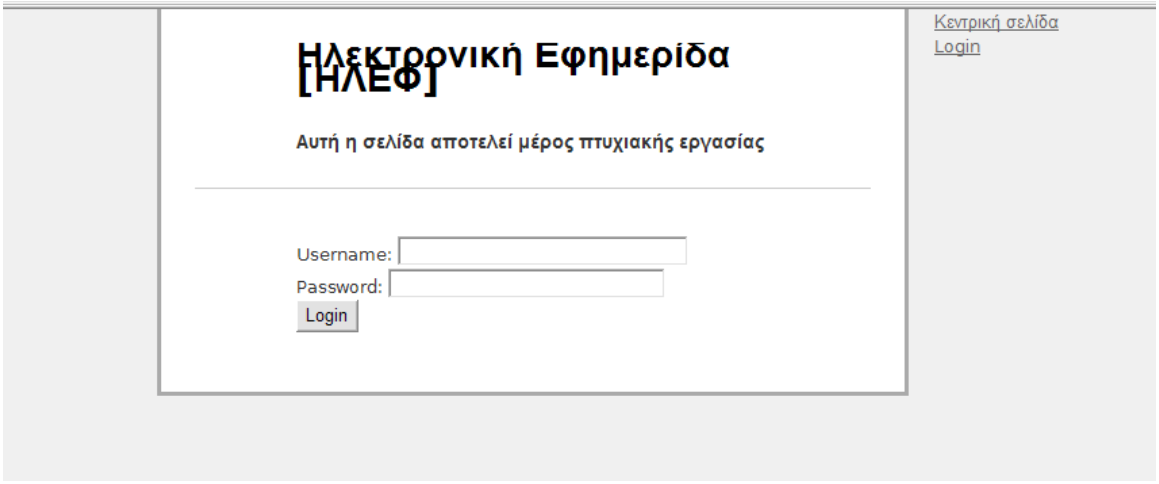
Κατά την διαδικασία υλοποίησης παρατηρήθηκε ότι το ίδιο το framework οδηγεί τον προγραμματιστή στη δημιουργία μιας σωστής δομής. Αυτό είναι προφανές από την πρώτη στιγμή, καθώς η αρχική δομή είναι ήδη βασισμένη στο MVC. Αξιοσημείωτο είναι και το πόσο εύκολα εντοπίζεται σχετική βιβλιογραφία και αρθογραφία για την υλοποίηση σχεδόν οποιασδήποτε ανάγκης μιας σύγχρονης εφαρμογής, το οποίο επιταχύνει κατά πολύ την διαδικασία. Παράλληλα ο κώδικας που παράγεται αλλά και που προσθέτει επιπλέον ο προγραμματιστής είναι τόσο λίγος που είναι εύκολα διαχειρίσιμος και επεκτάσιμος.

[18] http://teapoci.blogspot.com/2008/04/html-editor-or-rich-text-editor-for.html

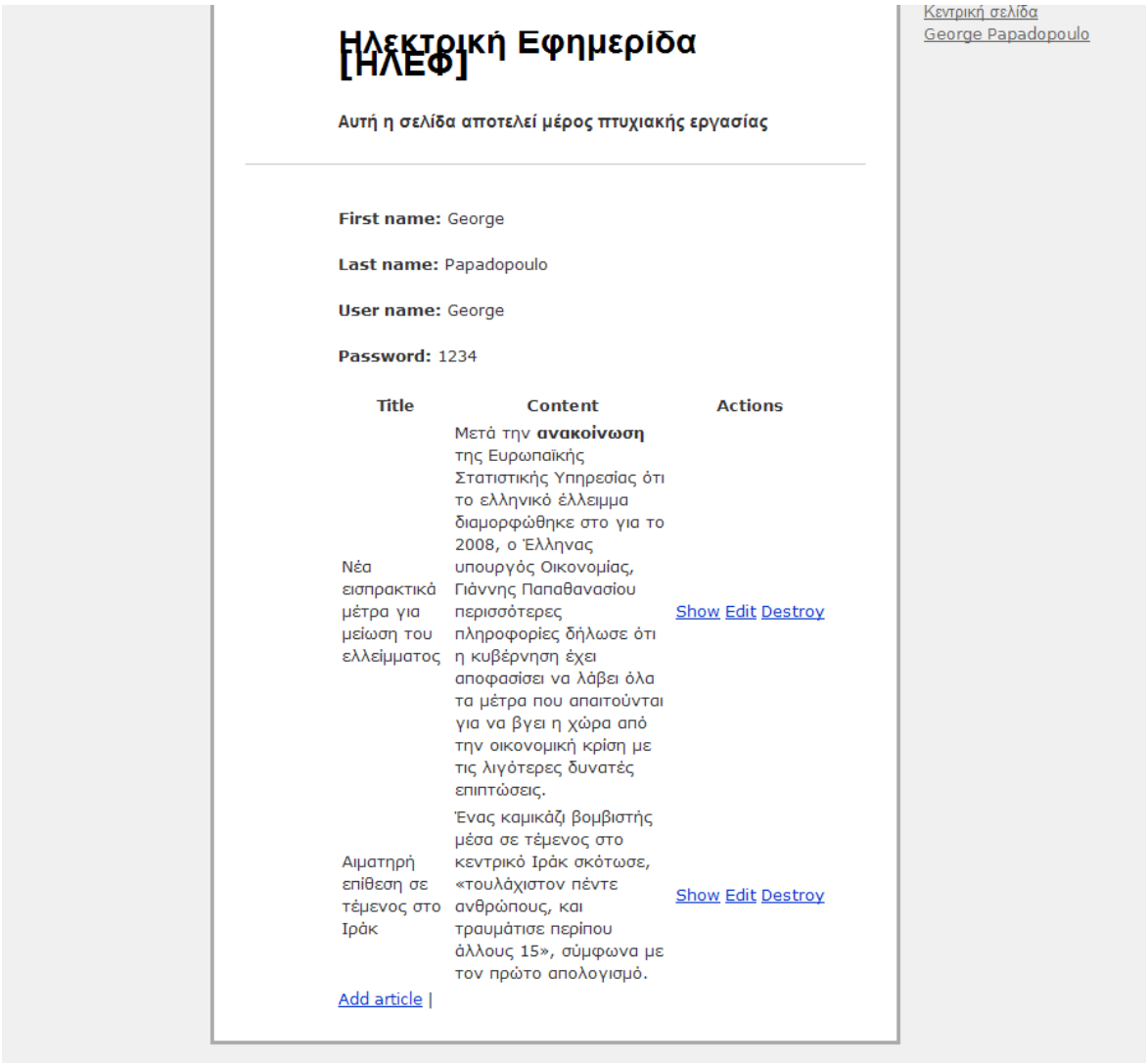
Από την άλλη υπάρχει το βασικό μειονέκτημα της μη ύπαρξης ενός καλού IDE, το οποίο θα μπορούσε με χρήση διαγραμματικών τεχνικών, UML ή άλλου είδους, να παράγει αλλά και να διαμορφώνει τον κώδικα. Ή τουλάχιστον να παράγει εύκολα διαγράμματα που να απεικονίζουν την όλη δομή και λειτουργία της εφαρμογής.

4.6 Παρουσίαση υλοποίησης

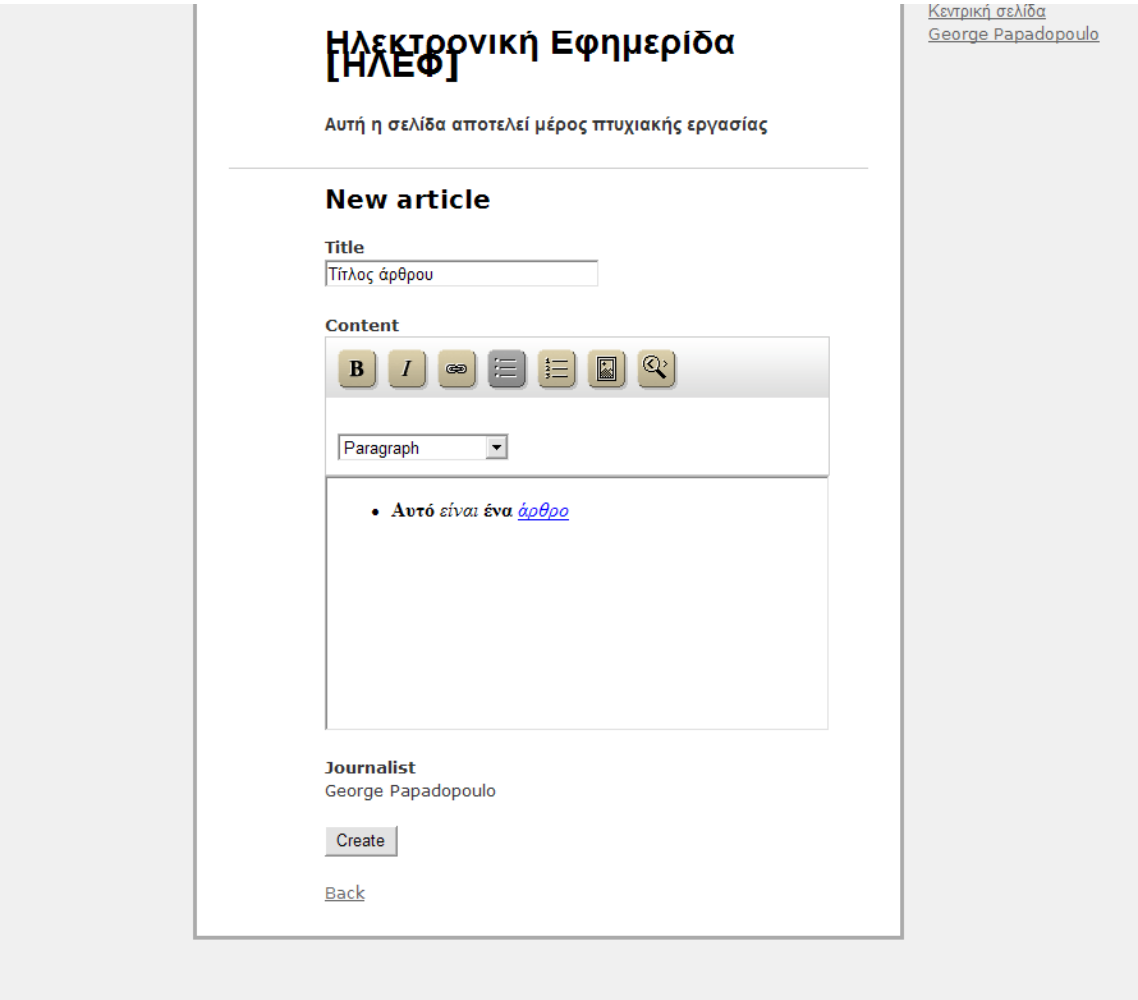
Εικόνα 36 Αρχική οθόνη της εφαρμογής



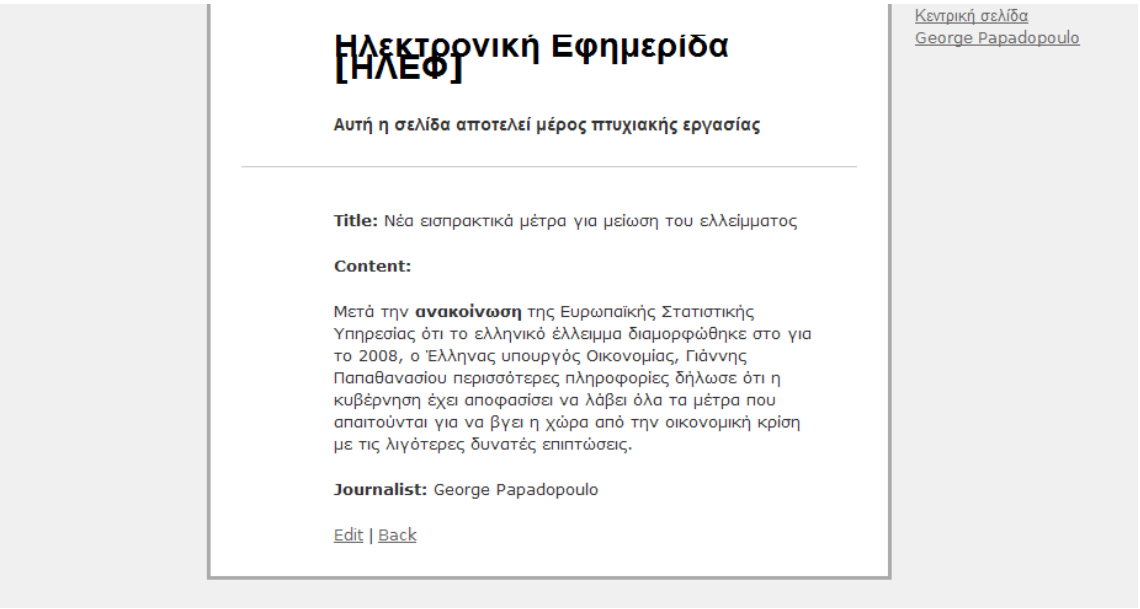
Εικόνα 37 Οθόνη εισόδου στο σύστημα



Εικόνα 38 Οθόνη προφίλ Δημοσιογράφου



Εικόνα 39 Οθόνη καταχώρισης άρθρου



Εικόνα 40 Οθόνη προβολής μεμονωμένου άρθρου

ΚΕΦΑΛΑΙΟ 5ο

ΠΗΓΕΣ

5.1 Βιβλιογραφία

1. PHP [http://www.php.net/]
2. ASP.net [http://en.wikipedia.org/wiki/ASP.NET]
3. Agile software development [http://en.wikipedia.org/wiki/Agile_software_development]
4. Learning Ruby [Michael Fitzgerald - O'Reilly Media, Inc, USA] ISBN:9780596529864
5. Ruby On Rails [Bruce A.,Tate, Curt,Hibbs - O'Reilly Media, Inc, USA] ISBN:0596101325
6. Αντικειμενοστρεφής σχεδιασμός με UML [Διομμήδης Σπινέλλης – Οικονομικός Πανεπιστήμιο Αθηνών] http://www.dmst.aueb.gr/dds/mlang/uml/indexw.htm
7. Introduction to Ruby [Εισαγωγή στην Ruby] [Νίκος Δημητρακόπουλος, Γιάννης Μπουρλάκος - Πανεπιστήμιο Πελοποννήσου] http://www.slideshare.net/demisone/introduction-to-ruby-ruby
8. Build your own Ruby On Rails Web Applications [Patrick Lenz - Sitepoint] ISBN:9780975841952
9. Ruby by example. Concepts and Code [Kevin C. Baird – No Starch Press] ISBN:9781593271480
10. Advanced Rails Recipes [Mike Clark – The Pragmatic Programmers] ISBN:9780978739225
11. Τι είναι το RSS [ΔΙΚΤΥΩΘΕΙΤΕ] http://www.go-online.gr/ebusiness/specials/article.html?article_id=1296
12. Ruby Syntax http://docs.huihoo.com/ruby/ruby-man-1.4/syntax.html

13. Ruby On Rails Introduction http://www.tutorialspoint.com/ruby-on-rails/rails-introduction.htm

5.2 Επιπλέον πηγές

1. Κώδικας για έλεγχο ταυτότητας χρήστη [User authentication in Ruby on Rails, Kacper Cieśla (comboy)] http://codingbitch.com/p/comboy/User%20authentication%20in%20Ruby%20on%20Rails
2. Κώδικας για rss http://paulsturgess.co.uk/articles/show/13-creating-an-rss-feed-in-ruby-on-rails
3. Κώδικας για εισαγωγή μορφοποιημένου κειμένου http://teapoci.blogspot.com/2008/04/html-editor-or-rich-text-editor-for.html
4. Κώδικας για κυλιόμενο μήνυμα http://www.dynamicdrive.com/dynamicindex2/cmarquee.htm
5. Οδηγίες για διαμόρφωση ημερομηνίας για την Ruby http://snippets.dzone.com/posts/show/2255
6. Εφέ παρουσίας modal διαλόγου «ModalBox» http://www.wildbit.com/labs/modalbox/