ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή εργασία

ΑΝΑΠΤΥΞΗ WEB-ΕΦΑΡΜΟΓΩΝ ΚΑΙ SERVICES-ΜΕ ΤΗΝ ΠΛΑΤΦΟΡΜΑ RUBY ON RAILS

Του φοιτητή:

Τζάτσου Κων/νου

ΑΜ: 04/2590

Επιβλέπων καθηγητής:

Σφέτσος Παναγιώτης

ΠΡΟΛΟΓΟΣ

Τα τελευταία χρόνια η γλώσσα προγραμματισμού Ruby γίνεται όλο και πιο δημοφιλής στις τάξεις των προγραμματιστών. Η Ruby είναι αντικειμενοστρεφής γλώσσα σεναρίων που επιτρέπει ταχεία ανάπτυξη εφαρμογών αλλά και εφαρμογών διαδικτύου με την χρήση του περιβάλλοντος Ruby On Rails. Η πτυχιακή εργασία αυτή θα είναι ένας εύχρηστος οδηγός για αρχάριους χρήστες ώστε να κατανοήσουνε την φιλοσοφία της Ruby on Rails.

ΠΕΡΙΛΗΨΗ

Η πτυχιακή εργασία « Ανάπτυξη web εφαρμογών και services με την πλατφόρμα Ruby on Rails » θα προσπαθήσει να παρουσιάσει τα σημαντικότερα χαρακτηριστικά της γλώσσας προγραμματισμού Ruby αλλά και τα βασικότερα χαρακτηριστικά της πλατφόρμας Ruby on Rails. Τέλος θα υπάρξει υλοποίηση ενός ηλεκτρονικού καταστήματος μικρής κλίμακας γραμμένο σε Ruby on Rails.

ABSTRACT

The graduation project "Development of web applications and services on the platform Ruby on Rails» will try to present the main features of the Ruby programming language and the main features of the platform Ruby on Rails. Finally there will be implementing a small-scale shop written in Ruby on Rails.

ΕΥΡΕΤΗΡΙΟ ΠΕΡΙΕΧΟΜΕΝΩΝ

1. ΕΙΣΑΓΩΓΗ

Η ολοκλήρωση της πτυχιακής αυτής δεν ήταν κάτι εύκολο. Αρχικά θα έπρεπε να ασχοληθώ με την Ruby. Κάτι το οποίο θα συνέβαινε για πρώτη φορά διότι ήταν κάτι άγνωστο για εμένα. Ψάχνοντας υλικό στο διαδίκτυο κάτι το οποίο ήταν πολύ χρονοβόρο διότι το υλικό για την εκμάθηση της Ruby ήταν μεν τεράστιο όμως όλα στην αγγλική γλώσσα. Μετά από πολύ ψάξιμο αποφάσισα να γίνω συνδρομητής για τρείς μήνες σε σχολείο εκμάθησης εξ αποστάσεως το www.codeschool.com . Από εκεί έμαθα τα βασικά της Ruby αλλά και τις Rails και κατάλαβα πολλά όσο αναφορά την φιλοσοφία τους. Όλα αυτά ήταν αρκετά όσο αναφορά να συντάξω το θεωρητικό κομμάτι της πτυχιακής. Για το κομμάτι της εφαρμογής κατάλαβα ότι χρειάζομαι κάτι πιο εξειδικευμένο. Έτσι ξεκίνησα νέο κύκλο αναζήτησης στο διαδίκτυο που και εδώ συνάντησα πολλές δυσκολίες. Πρώτα από όλα η Rails ανανεώθηκε από Rails 3 σε Rails 4. Εγώ ήθελα η εφαρμογή μου να γίνει με την νέα έκδοση της Rails. Οι διαφορές βέβαια ανάμεσα στις δυο εκδόσεις δεν ήταν χαοτικές ήταν όμως σημαντικές με αποτέλεσμα στον κώδικα μου να συναντώ συνέχεια λάθη. Αυτό συνέβαινε διότι ότι σημειώσεις ή tutorial είχα στην κατοχή αναφέρονταν στην Rails 3 αλλά εγώ την εφαρμογή μου την έχτιζα με Rails 4. Επίσης πολλά προβλήματα συνάντησα στον κώδικα μου διότι κάποιες λεπτομέρειες του λογισμικού μου συστήματος (windows 8) διέφεραν από τις σημειώσεις του διαδικτύου που χρησιμοποιούσαν παλιότερες εκδόσεις windows ή ios. Βέβαια να αναφέρω ότι με προτροπή του καθηγητή μου κ.Σφέτσο χρησιμοποίησα σαν editor το RubyMine. Επιμελήθηκε το γεγονός να επικοινωνήσει με την εταιρία να μου το προσφέρει δωρεάν για ένα χρόνο κάτι το οποίο μου έλυσε τα χέρια διότι πρόκειται για εξαιρετικό editor. Στην εφαρμογή που δημιούργησα ένα eshop μικρής κλίμακας δεν έδωσα μεγάλη σημασία στην εμφάνιση που θα έχει διότι ήθελα να δώσω έμφαση περισσότερο στην Ruby on Rails παρά στο css και html . Τέλος κατάφερα να ολοκληρώσω την πτυχιακή αλλά με πολύ καθυστέρηση διότι από το μηδέν προσπάθησα να δημιουργήσω κάτι.

Στο κεφάλαιο Ruby αναλύονται τα βασικά χαρακτηριστικά της γλώσσας η φιλοσοφία της το συντακτικό κομμάτι αλλά και η εγκατάσταση της. Έπειτα στο 3ο κεφάλαιο αναλύω την πλατφόρμα Rails ή Ruby on Rails. Αναλύω την φιλοσοφία της Rails αλλά και πως δίνει δύναμη στην γλώσσα της Ruby. Στο 4ο κεφάλαιο εισχωρούμε πιο βαθειά στα μονοπάτια της Rails και ενώνουμε τα κομμάτια ώστε να μπορούμε να δημιουργήσουμε μια καινούρια εφαρμογή.

2. Η ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ RUBY

Η Ruby σχεδιάστηκε από τον Yukihiro Matsumoto στην Ιαπωνία στις αρχές της δεκαετίας του 90. Σκοπός του Yukihiro ήταν η δημιουργία μίας γλώσσας προγραμματισμού που θα διευκόλυνε τον προγραμματιστή να επικεντρώνεται κυρίως στη συγγραφή κώδικα και όχι στην επίλυση παράπλευρων προβλημάτων που συχνά προέκυπταν από την ίδια τη φύση της γλώσσας. Έτσι δανείστηκε τα δυνατά γνωρίσματα άλλων γλωσσών όπως π.χ. Python, Perl, Smalltalk, Eiffel και Lisp.

Η γλώσσα είναι αντικειμενοστρεφής και όλα θεωρούνται αντικείμενα. Ακόμα και το κύριο πρόγραμμα της Ruby είναι ένα αντικείμενο. Σε αυτό το γεγονός έγκειται κατά ένα μεγάλο βαθμό η «δύναμη» της Ruby.

Σημαντικό πλεονέκτημα της το γεγονός ότι μπορεί να ‘’τρέξει’’ σχεδόν παντού, σε Windows, σε Linux, σε Java Virtual Machine καθώς και σε κινητά τηλέφωνα με οποιοδήποτε λογισμικό.

Όμως η Ruby δεν είναι η τέλεια γλώσσα καθώς και αυτήν όπως και όλες οι γλώσσες προγραμματισμού έχει τα μειονεκτήματα της. Το μεγαλύτερο μειονέκτημα της είναι ότι δεν μπορεί ακόμη να ανταποκριθεί σε εφαρμογές μεγάλης κλίμακας γι’αυτό και δεν χρησιμοποιείται ακόμη από μεγάλες επιχειρήσεις. Επίσης υστερεί σε απόδοση και ταχύτητα σε σχέση με άλλες γλώσσες όπως η JAVA και η PHP.

2.1 ΕΓΚΑΤΑΣΤΑΣΗ

Η εγκατάσταση της Ruby στο σύστημα μας γίνεται μέσω της επίσημης ιστοσελίδας της Ruby www.ruby-lang.org από όπου μπορούμε να κατεβάσουμε και να εγκαταστήσουμε τις τελευταίες εκδόσεις της Ruby αλλά και παλιότερες ανεξαρτήτου πλατφόρμας που χρησιμοποιούμε.

Όπως παρατηρούμε, στο τέλος θα έχουμε στα windows το «Start Command Prompt with Ruby and Rails» , «Ruby Gems Documentation» «Interactive Ruby» . Το πρώτο είναι ένα κέλυφος (shell) μέσω της Ruby. Το δεύτερο είναι ένας τοπικός server που μπορούμε να χρησιμοποιήσουμε για να δούμε πληροφορίες για τα gems. Το τρίτο είναι το αλληλεπιδραστικό κέλυφος της Ruby, ή **Interactive Ruby Shell** (**IRB**) . Το πρόγραμμα εκτελείται από μια γραμμή εντολών και επιτρέπει την εκτέλεση εντολών της Ruby με άμεση απάντηση, επιτρέποντας τον πειραματισμό σε πραγματικό χρόνο με τη γλώσσα. Υποστηρίζει ιστορικό εντολών, δυνατότητες διόρθωσης γραμμής και έλεγχο διεργασιών, και μπορεί να επικοινωνεί άμεσα σαν σενάριο κελύφους με το διαδίκτυο και να αλληλεπιδρά με κάποιον κεντρικό υπολογιστή.

Εικόνα 2.1 – Παράδειγμα χρήσης I.R.B.

2.2 ΣΥΝΤΑΞΗ

Η σύνταξη της Ruby έχει πολλές ομοιότητες με την σύνταξη της Pearl και της Python.Ο ορισμός των κλάσεων και των μεταβλητών γίνεται με λέξεις κλειδιά ωστόσο οι μεταβλητές δεν χρειάζεται να ξεκινούν με κάποιον ειδικό χαρακτήρα που να ορίζει τον τύπο δεδομένων της μεταβλητής. Την μεγάλη διαφορά η Ruby την κάνει στο μπλοκ κώδικα. Σε αντίθεση με πολλές γλώσσες προγραμματισμού η Ruby δεν χρησιμοποιεί αγκύλες στην αρχή και το τέλος ενός μπλοκ αλλά λέξεις κλειδιά. Έτσι ο κώδικας γίνεται πιο ευανάγνωστος.

2.3.1 ΜΕΤΑΒΛΗΤΕΣ

Υπάρχουν τεσσάρων ειδών μεταβλητές που μπορεί να βρει κάποιος σε ένα πρόγραμμα γραμμένο σε *Ruby*. Αυτές είναι οι παρακάτω:

• Η τοπική μεταβλητή *(local variable),* που ορίζεται μέσα σε ένα αντικείμενο και ξεκινάει με μικρό γράμμα ή κάτω παύλα όπως για παράδειγμα *example, _α, .*

• Η μεταβλητή στιγμιότυπου *(instance variable)*, ορίζεται μέσα σε ένα αντικείμενο και ξεκινάει με το σύμβολο @ για παράδειγμα *@example, @_, @Example.*

• Η μεταβλητή κλάσης *(class variable)*, είναι μία μεταβλητή που μπορεί να χρησιμοποιηθεί μέσα σε όλη την κλάση. Ξεκινάει πάντα με τα σύμβολα @@ όπως τα *@@example, @@_, @@Example.*

• Η καθολική μεταβλητή *(global variable)*, ξεκινάει με το σύμβολο $ ακολουθούμενο από οποιαδήποτε γράμματα. Για παράδειγμα *$example, $EXAMPLE*

2.3.2 ΣΤΑΘΕΡΕΣ

Μία σταθερά ξεκινάει πάντα με κεφαλαίο γράμμα και ακολουθείτε από οποιοδήποτε γράμμα. Πχ class First

2.3.3 ΜΕΘΟΔΟΙ

Μία μέθοδος ξεκινάει πάντα με μικρό γράμμα ή με κάτω παύλα και μπορούν να χρησιμοποιηθούνε και τα σύμβολα ? , ! , = . Επίσης ορίζονται ανάμεσα στο def και στο end.Θα πρέπει να επισημάνουμε πως οι παρενθέσεις μπορούνε να παραληφθούν.

Τέλος θα πρέπει να επισημάνουμε πως μια μέθοδος επιστρέφει πάντα την τιμή της τελευταίας πράξης εκτός και αν χρησιμοποιούμε την εντολή return.

```
def  first_method
    Puts “this is the first method”
end
def second_method input
    puts input
end
```

2.3 ΣΥΜΒΟΛΟΣΕΙΡΕΣ

Οι συμβολοσειρές *(strings)* είναι ένα σύνολο χαρακτήρων. Στη Ruby, μια συμβολοσειρά διαθέτει όλα τα χαρακτηριστικά γνωρίσματα των αντικειμένων. Αρχικά, η εμφάνιση μιας σειράς χαρακτήρων στη *Ruby* γίνεται δυναμικά και με τη χρήση των μεθόδων εξόδου *puts* και *print*.

```
puts “hello world”
```

=>>"hello world"

Το παραπάνω κείμενο, μπορεί να αντιστοιχιστεί σε μία μεταβλητή, η οποία και θα αποτελέσει ένα αντικείμενο (βασική αρχή της γλώσσας ότι όλα είναι αντικείμενα).

string = "hello world"

=>> "hello world"

Η *Ruby* διαθέτει πολλές ενσωματωμένες μεθόδους για τη διαχείριση συμβολοσειρών. Μιας που το *string* είναι αντικείμενο, μπορεί να καλέσει μία από τις ενσωματωμένες μεθόδους διαχείρισης συμβολοσειρών όπως η reverse,length,upcase,empty? και πολλές άλλες. Τέλος για τη χρήση διαφόρων μεθόδων ελέγχου, η πρόσθεση του αγγλικού ερωτηματικού **(?)** στο τέλος της κάθε μεθόδου αρκεί για να επιστρέψει ότι μια συνθήκη είναι αληθής ήψευδής (true or false).

string.length

=>11

string.empty?

=>> false

2.4 ΑΡΙΘΜΟΙ

Όπως και όλα τα άλλα πράγματα στην Ruby έτσι και οι αριθμοί είναι αντικείμενα. Ο χειρισμός τους είναι απλός και εύκολος.

3*4 => 12 10-1=>9 5+2=>7 50/5=>10

Επίσης έναν αριθμό θα μπορούμε να τον χρησιμοποιούμε και σαν επαναληπτικό βρόγχο όταν θέλουμε την επανάληψη μιας λέξης η πρότασης.

"Hello"*2 =>"Hello Hello"

Η *Ruby* χειρίζεται με διαφορετικό τρόπο τους χαρακτήρες και με διαφορετικό τρόπο τους αριθμούς. Όταν ορίζεται ένα αριθμός, αυτόματα γίνεται τύπου *Fixnum[2]*, το οποίο κληρονομεί

τα χαρακτηριστικά της κλάσης *Integer.* Αυτό όμως είναι κάτι που μπορούμε να αλλάξουμε μέσω των μεθόδων μετατροπής.

Οι μέθοδοι μετατροπής που χρησιμοποιούνται είναι οι παρακάτω:

• *to_i* που μετατρέπει κάτι σε ακέραιο *integer*

• *to_s* που μετατρέπει κάτι σε συμβολοσειρά *string*

• *to_f* που μετατρέπει κάτι σε αριθμό κινητής υποδιαστολής *floating point*

• *to_a* που μετατρέπει κάτι σε πίνακα *array*

25.to_s.reverse => “52”

2.5 ΠΙΝΑΚΕΣ

Στην Ruby και σε αντίθεση με άλλες γλώσσες προγραμματισμού ένας πίνακας μπορεί να διαθέτει διαφορετικού τύπου δεδομένα. Μπορεί ένας πίνακας δηλαδή να διαθέτει ακέραιο αριθμό, χαρακτήρα και μέθοδο που επιστρέφει μια τιμή!

pinakas= [7,’example’,methodos(input)]

Η δημιουργία ενός πίνακα γίνεται με πολλούς και διάφορους τρόπους όπως:

a=[1,2,3,4,5] a[2]=>3

a=Array.new () a<<’one’<<’two’<<’three’ a[0]=>one

a=Array.new(4,1) a=>[1,1,1,1]

Ένας πίνακας υπάρχει περίπτωση να περιέχει μόνο συμβολοσειρές. Η δημιουργία ενός τέτοιου πίνακα, γίνεται χρησιμοποιώντας το *%w {…}* και εισάγοντας το κείμενο μέσα στις αγκύλες. Η Ruby αναγνωρίζει κάθε μια λέξη με πεδίο του πίνακα.

%w{This is an example}

=>[“This”,”is”,”an”,”example”]

Για να προσθέσουμε ένα ακόμη στοιχείο σε έναν πίνακα που ήδη διαθέτει στοιχεία χρησιμοποιούμε το σύμβολο <<

a=[1,2,3,4,5] a<<6 a=[1,2,3,4,5,6]

Όπως γνωρίζουμε η αρίθμηση στους πίνακες ξεκινάει από το 0 έτσι το τελευταίο στοιχείο ενός πίνακα θα είναι το pinakas[pinakas.length-1]. Όμως στην Ruby μπορούμε να χρησιμοποιήσουμε και αρνητική τιμή σε έναν πίνακα όπως:

pinakas=[‘one’,’two’,three’,’four’]

pinakas[0]=>’one’

pinakas[-1]=>’four’

Αυτό που κάνει η Ruby είναι να μετράει αντίστροφα σε περίπτωση αρνητικής τιμής.

Για να δημιουργήσουμε έναν πολυδιάστατο πίνακα θα πρέπει να δημιουργήσουμε έναν μονοδιάστατο και να ενσωματώσουμε σε αυτόν έναν ακόμη τουλάχιστον μονοδιάστατο πίνακα

Π.χ. δημιουργούμε έναν πίνακα.

multipinakas=[1,2,3]

Έπειτα προσθέτουμε άλλους 2 πίνακες

multipinakas << %w {one two three} << [4,5,6]

Το πρώτο στοιχείο της δεύτερης γραμμής:

multipinakas[1][0]=>'one'

Τέλος για να μπορούμε να δημιουργήσουμε πολυδιάστατο πίνακα χρησιμοποιώντας την μέθοδο της Ruby Array.new()

multipinakas=Array.new(7) {Array.new(7) {7} }

Το παραπάνω παράδειγμα θα δημιουργήσει έναν δυσδιάστατο πίνακα 7x7 και όλα τα στοιχεία του θα είναι το νούμερο 7.

2.6 ΕΥΡΕΤΗΡΙΑ

Το ευρετήριο (hash) το χρησιμοποιούμε για συλλογή δεδομένων όπως τον πίνακα αλλά διαφέρει στο ότι τα δεδομένα δεν αποθηκεύονται με αριθμητική σειρά αλλά χρησιμοποιεί έναν συνδυασμό κλειδιού-τιμής τα οποία προσπελάζονται με οποιαδήποτε σειρά και οποιοδήποτε τρόπο. Η επιλογή ευρετηρίου αντί για πίνακα γίνεται σε περιπτώσεις όπου μας ενδιαφέρει η γρήγορη αναζήτηση (για να βρούμε την τιμή που θέλουμε αρκεί να δώσουμε το κλειδί) και όταν δεν μας ενδιαφέρει η διάταξη τιμών.

Η δημιουργία ενός ευρετηρίου γίνεται ως εξής :

seasons = Hash['season1','autumn', 'season2','winter']

seasons = {'season1'=>'autumn' , 'season2'=>'winter']

Για να προσπελάσουμε ένα δεδομένο ενός ευρετηρίου:

seasons['season2'] => 'winter'

2.7 ΕΛΕΓΧΟΣ ΡΟΗΣ

Η Ruby διαθέτει αρκετούς μηχανισμούς για τον έλεγχο ροής όπως:

- If…else if…elseif…else
- for
- while
- case
- until
- unless

2.7.1 if…else if…elseif…else

Η πιο γνωστή δομή στον κόσμο των προγραμματιστών δεν διαφέρει στην Ruby

If συνθήκη1

κώδικας1

elseif συνθήκη2

κώδικας2

else

κώδικας3

end

Αξιοσημείωτο το γεγονός ότι δεν χρειάζονται αγκύλες στις συνθήκες.

2.7.2 for

Η επαναληπτική δομή for είναι κλασική στις γλώσσες προγραμματισμού. Επιτρέπει κάτι να επαναλαμβάνεται πολλές συνεχόμενες φορές. Στη *Ruby*, εκτός του ότι δεν χρησιμοποιούνται

αγκύλες όπως στις άλλες γλώσσες, είναι και διαφορετικός το τρόπος που δηλώνεται το εύρος των τιμών επανάληψης.

for I in 1..3 do

puts “Hello World”

end

=>Hello World

=>Hello World

Η χρήση του do είναι προαιρετική και συνήθως χρησιμοποιείται μόνο όταν ο κώδικας είναι γραμμένος σε περισσότερες από μια γραμμή.

2.7.3 while

Την δομή while την χρησιμοποιούμε όταν θέλουμε να ελέγξουμε κατά το πόσο είναι αληθής μια συνθήκη.

a=1

while a<4 do

puts a

a+=1

end

=>1 2 3

2.7.4 case

Η case είναι ιδανική σε περιπτώσεις που υπάρχουν πολλές διαφορετικές περιπτώσεις που μπορεί να ισχύουν στη επίλυση ενός προβλήματος. Η σύνταξή της είναι απλή και ένα παράδειγμα

χαρακτηριστικό είναι το εξής:

a = 1

case

when a < 5 then puts "#{a} μικρότερο του 5"

when a == 5 then puts "#{a} ίσο του 5"

when a > 5 then puts "#{a} μεγαλύτερο του 5"

end

=>1 μικρότερο του 5

2.7.5 until

Η συγκεκριμένη δομή είναι παρόμοια με την while με τη διαφορά ότι ισχύει μέχρι η συνθήκη να γίνει αληθής.

```
x=10
until x==15 do
puts x
x +=1
end
=>10      11      12      13      14
```

2.7.6 unless

Η συγκεκριμένη δομή, έχει διαφορετικά αποτελέσματα από την if…else δομή.

```
a=5
unless a==4
a=7
end
puts("Το a ισούται με: #{a}")
=>  Το a ισούται με: 7
```

2.8 ΚΛΑΣΕΙΣ

Όπως έχουμε αναφέρει τα πάντα στην Ruby είναι αντικείμενα, αντικείμενα τα οποία προέρχονται από κλάσεις. Πως ορίζουμε μια κλάση; Χρησιμοποιώντας την μορφή class Name …. end . Το όνομα πρέπει να ξεκινάει πάντα με κεφαλαίο και ανάμεσα από τον ορισμό της κλάσης και του end είναι το σώμα της κλάσης όπου θα δημιουργηθούνε τυχόν μέθοδοι και μεταβλητές.

```
class Firstclass
    #ορισμός μεθόδων και μεταβλητών
end
```

2.9 ΜΕΘΟΔΟΙ

Οι μέθοδοι στην Ruby χωρίζονται σε τρεις κατηγορίες. Στις δημόσιες *(public)*, στις ιδιωτικές *(private)* και στις προστατευμένες *(protected)*.

- Δημόσιες είναι οι κλάσεις που μπορούνε να κληθούν απόοποιοδήποτε αντικείμενο και μέθοδο μέσα σε ένα πρόγραμμα.
- Ιδιωτικές μέθοδοι, θεωρούνται οι μέθοδοι οι οποίοι μπορούν να κληθούν μόνο από τις άλλες μεθόδους που βρίσκονται μέσα στην ίδια κλάση ή από υποκλάσεις. Δεν μπορεί να τις καλέσει κάποιο αντικείμενο απευθείας , παρά μόνο μέσα από μια ήδη υπάρχουσα μέθοδο.
- Οι προστατευμένες μέθοδοι από την άλλη, είναι κατά κάποιο τρόπο ιδιωτικές αφού δεν μπορούν να καλεστούν από οποιονδήποτε. Μπορούν να κληθούν μόνο από αντικείμενα της κλάσης που ανήκουν καθώς και από τις υποκλάσεις της.

Οι μέθοδοι πρέπει να ξεκινάνε πάντα με μικρό γράμμα, και ορίζονται με τη γενική μορφή def όνομα … end

```
class Firstclass
#όλες οι μέθοδοι που δηλώνονται εδώ είναι public
    def first_public_method
        #σώμα μεθόδου
    end
    private
#όλες οι μέθοδοι που δηλώνονται εδώ είναι private
    def first_private_method
        #σώμα μεθόδου
    end
    protected
# όλες οι μέθοδοι που δηλώνονται εδώ είναι protected
    def first_protected_method
        #σώμα μεθόδου
```

end

end

Υπάρχει και εναλλακτικός τρόπος δήλωσης μεθόδων.

class Firstclass

#Δήλωση μεθόδων

public: method1

protected: method2, method3

private: method4, method5

#Ορισμός κάθε μεθόδου από εδώ και κάτω

end

2.10 ΑΝΤΙΚΕΙΜΕΝΑ

Μια κλάση από μόνη της δεν είναι και πολύ δημιουργική εάν δεν δημιουργηθούνε αντικείμενα της κλάσης αυτής. Ένα αντικείμενο δημιουργείτε με την βοήθεια της μεθόδου new.

firstobject=Firstclass.new

Έπειτα αυτό το αντικείμενο θα μπορεί να καλέσει μια μέθοδο της κλάσης του.

firstobject.first_public_method

2.11 MODULES

Η Ruby διαθέτει τα modules (ενότητες) που είναι αντίστοιχα με τα packages στην java και τα namespaces στην C# . Μια ενότητα ορίζεται στην Ruby όπως και η κλάση με την διαφορά ότι στην ενότητα δεν μπορούμε να ορίσουμε ιδιότητες (attributes) καθώς δεν μπορούμε ούτε να κληρονομήσουμε από αυτήν αλλά ούτε να δημιουργήσουμε κάποιο αντικείμενο. Δηλαδή μια ενότητα διαθέτει μόνο σταθερές και μεθόδους. Για να ενσωματώσουμε μια ενότητα σε μια κλάση θα πρέπει να χρησιμοποιήσουμε την λέξη κλειδί include.

module CarUtils

NUM_OF_DOORS = 4

def paint(car,color)

…..

end

end

class Car

include CarUtils

…..

……

end

2.12 ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ

Όπως και στις περισσότερες γλώσσες προγραμματισμού έτσι και η Ruby μας δίνει την δυνατότητα της κληρονομικότητας, τη δυνατότητα κλάσεων οι οποίες κληρονομούν κάποια χαρακτηριστικά από κάποιες άλλες κλάσεις. Ισχύουν και εδώ όσα ξέρετε για την κληρονομικότητα και από άλλες γλώσσες. Οι private μέθοδοι δεν κληρομούνται ενώ οι public μέθοδοι ναι. Για παράδειγμα εάν θέλαμε μια κλάση να κληρονομήσει την κλάση ActiveRecords::Base, που χρησιμοποιείται συχνά με την Ruby on Rails, και μας επιτρέπει να επικοινωνούμε με βάσεις δεδομένων όπως η MySQL με πολύ απλό και βολικό τρόπο, τόσο βολικό που πρακτικά δε γράφουμε SQL. Περισσότερα θα εξηγήσουμε βέβαια στην ενότητα που αφορά την Rails. Εμείς κρατάμε το ότι πρόκειται για μία κλάση που μας προσφέρει ότι χρειαζόμαστε για να επικοινωνήσουμε με μία βάση δεδομένων.

def Example < ActiveRecord::Base

….

end

Γενικά το σύμβολο < χρησιμοποιείτε μπροστά από την κλάση που θέλουμε να κληρονομήσουμε.

class Klash < Yperklash

……

end

2.13 RubyGems

Όλες οι γλώσσες προγραμματισμού έχουν τις δικές τους βιβλιοθήκες. Φυσικά και η Ruby έρχεται με τις δικές τις βιβλιοθήκες που ονομάζονται gems και μάλιστα μπορείτε και εσείς να συνεισφέρεται προσφέροντας τις δικές σας βιβλιοθήκες. Δηλαδή να δημιουργήσετε ένα δικό σας gem. Τα RubyGems είναι μέρος της βιβλιοθήκης από την έκδοση Ruby 1.9. Η εφαρμογή rubygems αναλαμβάνει να επικοινωνεί με το http://rubygems.org που είναι το κυριότερο repo από gems καθώς και με όσα άλλα του ορίσετε εσείς.

Τα Gem τα δηλώνεις στον φάκελο Gemfile και με την εντολή :

$ bin/ bundle install

όλα τα gem εγκαθίστανται στην εφαρμογή μας αφού συνδεθεί πρώτα με την σελίδα http://rubygems.org

Παράδειγμα Gemfile

```
source 'https://rubygems.org'


gem 'rails', '3.2.12'

gem 'sqlite3'


# Gems used only for assets and not required

# in production environments by default.

group :assets do

  gem 'sass-rails',   '~> 3.2.3'

  gem 'coffee-rails', '~> 3.2.1'

  gem 'uglifier', '>= 1.0.3'

end


gem 'jquery-rails'
```

2.14 Επίλογος

Στο κεφάλαιο αυτό αναλύσαμε τα χαρακτηριστικά γνωρίσματα της γλώσσας προγραμματισμού Ruby. Σαν αυτόνομη γλώσσα η Ruby δεν ενθουσιάζει

τους προγραμματιστές αλλά σε συνεργασία με κάποιο framework κάνει θαύματα. Ένα framework είναι και το Rails. Ή αλλιώς Ruby on Rails.

3. RUBY ON RAILS

Τα τελευταία χρόνια παρατηρούμε μια έκρηξη τόσο στη χρήση αλλά και την παραγωγή νέων web frameworks. Αλλά καταρχήν ας δώσουμε μια μικρή εξήγηση του τι εστί web framework. Συνήθως οι προγραμματιστές γνωρίζουν κάποιες γλώσσες προγραμματισμού με τις οποίες μπορεί να δημιουργήσουν διάφορες εφαρμογές είτε από το μηδέν είτε χρησιμοποιώντας διάφορες βιβλιοθήκες που τους προσφέρουν έτοιμες κάποιες λειτουργίες. Ένα framework προσφέρει και αυτό κάποιες έτοιμες λειτουργίες όπως caching, templating, object mapping αλλά και μπορεί να επεκταθεί με plugins αλλά κατά κύριο λόγο προτυποποιεί κάποιες διαδικασίες όπως η αρχιτεκτονική της εφαρμογής, η σύνδεση στη βάση δεδομένων αλλά μπορεί και να δίνει δυνατότητα για γρήγορη και αυτόματη δημιουργία κώδικα όπως το scaffolding. Ένα τέτοιο framework είναι και το Ruby on Rails. Δημιουργήθηκε από τον David Heinemeier Hansson τον Ιούλιο του 2004 και από τότε αυξάνεται όλο και περισσότερο η χρήση του για κατασκευές εφαρμογών διαδικτύου. Η απόφαση του Hansson να κυκλοφορήσει το framework ως open source, έδωσε την δυνατότητα στο Ruby On Rails να πάρει όλα τα θετικά στοιχεία που μπορεί να προσφέρει η open source φιλοσοφία. Συνεχώς προγραμματιστές που δουλεύουν με το rails εκδίδουν εργαλεία και διορθώνουν λάθη που βρίσκουν στον πηγαίο κώδικα στο repository (αποθήκη) του rails. Το repository ελέγχεται από τον πυρήνα του Rails που αποτελείται από μια ομάδα έξι προγραμματιστών της οποίας ηγείται ο Hansson.

3.1 ΚΥΡΙΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ

Σε αυτό το κεφάλαιο θα παρουσιαστούν τα κύρια χαρακτηριστικά του Rails και θα προσπαθήσουμε να κατανοήσουμε την φιλοσοφία αυτού του framework.

3.1.1 DRY (Don’t Repeat Yourself)

Το Ruby On Rails υποστηρίζει την αρχή του DRY προγραμματισμού. Η λογική αυτής της αρχής, είναι Αντί να γράφουμε συνεχώς τον ίδιο κώδικα σε διάφορα σημεία της εφαρμογής, γράφουμε μια φορά τον κώδικα σε κάποιο “κεντρικό” σημείο και όταν θέλουμε να χρησιμοποιήσουμε αυτό το τμήμα του κώδικα κάπου στην εφαρμογή, απλά κάνουμε μια αναφορά στο μέρος όπου βρίσκεται γραμμένος ο κώδικας. Έτσι αν θέλουμε να αλλάξουμε την συμπεριφορά της εφαρμογής δεν χρειάζεται να αλλάξουμε τα σημεία όπου καλείται ο κώδικας, αλλά μόνο το σημείο που είναι γραμμένος ο κώδικας.

Ένα παράδειγμα για το πως το Rails υποστηρίζει αυτή την αρχή είναι το εξής. Σε αντίθεση με την JAVA δεν χρειάζεται να ορίζουμε συνεχώς την βάση δεδομένων και να κάνουμε συνεχώς σύνδεση με αυτή μέσα στην ίδια εφαρμογή. Πολύ απλά ορίζουμε μια φορά την βάση δεδομένων και όταν χρειάζεται να αντληθούν πληροφορίες από την βάση το Rails, χρησιμοποιεί τον ήδη υπάρχοντα ορισμό. Ένα ακόμη παράδειγμα αυτής της αρχής είναι η ανάπτυξη τεχνικών όπως το AJAX (asynchronous javascript and XML), όπου όταν κάποιος browser δεν υποστηρίζει αυτή την τεχνική, ο προγραμματιστής πρέπει να εμφανίσει τα αποτελέσματα χωρίς να χρησιμοποιήσει την τεχνική AJAX. Σε τέτοιες περιπτώσεις πολλοί προγραμματιστές πιάνουν τον εαυτό τους να γράφει τον ίδιο κώδικα πολλές φορές. Αυτό μπορεί πολύ εύκολα να το παρακάμψει το Framwrork Ruby On Rails χρησιμοποιώντας την αρχή DRY.

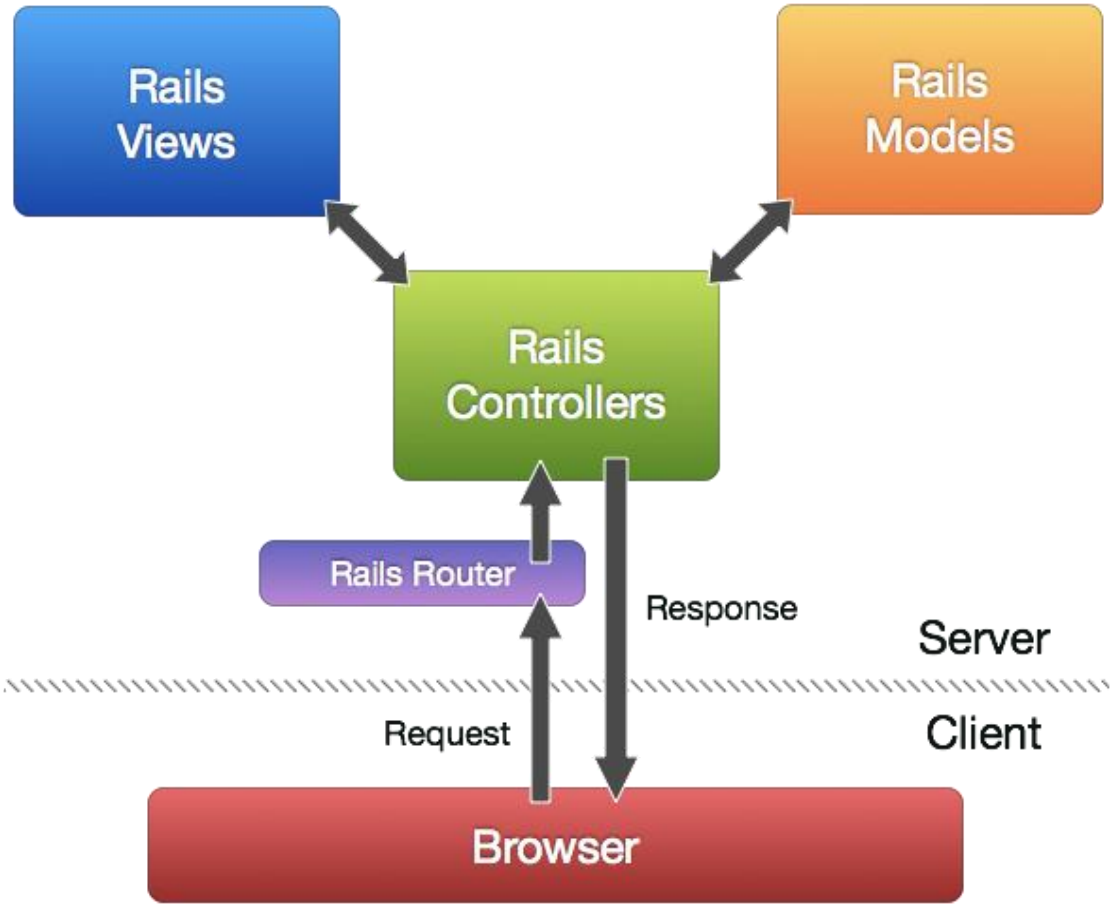
3.1.2 Convention Over Configuration ή COC

Η COC αποτελεί μια αρχή σχεδίασης, που βοηθάει στο να ελαχιστοποιεί τον αριθμό των αποφάσεων που πρέπει να πάρει ένας προγραμματιστής κερδίζοντας απλότητα αλλά χωρίς απαραίτητα να χάνεται η λειτουργικότητα.

Τα ονόματα των κλάσεων και των μεθόδων ονοματίζονται με βάση του Convention over Configuration. Για παράδειγμα, σε μια εφαρμογή Rails, εάν υπάρχει μια κλάση Actor τότε δημιουργείται ένας αντίστοιχος πίνακας actors μέσα στη βάση. Με αυτό τον τρόπο, το σύστημα καταλαβαίνει πιο εύκολα ότι όταν ένα αντικείμενο Actor θέλει να διαβάσει κάτι από την βάση θα πάει αυτόματα στον πίνακα που έχει το ίδιο όνομα ακολουθούμενο με ένα *s*, δηλαδή actors. Χρησιμοποιώντας αυτή τη τεχνική, δεν χρειάζονται παραπάνω ρυθμίσεις και αλλαγές αφού ακολουθείται πάντα η βασική σύμβαση που αναφέρθηκε παραπάνω. Φυσικά, μπορεί να προσπεραστεί αυτή η σύμβαση π.χ. όταν γίνετε χρήση μιας ήδη υπαρκτής Β.Δ. ή όταν δεν μπορεί ή δεν επιτρέπετε να γίνει η αλλαγή ονόματος ενός πίνακα.

3.1.3 Model-View-Controller (MVC)

Το Rails χρησιμοποιεί την αρχιτεκτονική MVC μοντέλο σχεδίασης. Κύριο γνώρισμα της συγκεκριμένης αρχιτεκτονικής είναι ότι η εφαρμογή διαιρείτε σε 3 υποσυστήματα. Το μοντέλο (model) την προβολή (view) και τον ελεγκτή (controller). Κάθε ένα από αυτά, λειτουργεί αυτόνομα αλλά επικοινωνεί με τα άλλα 2 υποσυστήματα.



Εικόνα 3.1 Το μοντέλο MVC

3.1.3.1 Το μοντέλο (Model)

Το μοντέλο (model) αντιστοιχίζεται στα δεδομένα της εφαρμογής. Τα δεδομένα αυτά μπορεί να είναι παροδικά, όταν υπάρχει μεγάλη αλληλεπίδραση με τον χρήστη, ή μπορεί να είναι σταθερά, και τότε συνήθως πρέπει να αποθηκευτούν κάπου εκτός της εφαρμογής, για παράδειγμα σε μια βάση δεδομένων. Είναι με λίγα λόγια οι κλάσεις της Ruby. Είναι αυτό το οποίο επικοινωνεί με την βάση δεδομένων, αποθηκεύει και επικυρώνει (validates) τα δεδομένα. Όλη αυτή η λειτουργικότητα επιτυγχάνεται με τη χρήση της ActiveRecord βιβλιοθήκης.

3.1.3.2 Η προβολή (view)

Η προβολή είναι αυτό που βλέπει ο χρήστης: HTML, CSS, XML, Javascript, JSON. Το ποια προβολή θα εμφανιστεί κάθε φορά, εξαρτάται από την ενέργεια που θα εκτελέσει ο ελεγκτής (controller). Είναι σαν μια μαριονέτα για το πρόγραμμα που δεν ενδιαφέρεται το τι γίνεται στο κύριο μέρος του προγράμματος

αλλά μόνο να εμφανίζεται κάθε φορά που ο ελεγκτής το καλεί. Η αντίστοιχη βιβλιοθήκη που χρησιμοποιείται είναι η ActionView.

3.1.3.3 Ο ελεγκτής (controller)

Ο ελεγκτής αποτελεί την καρδιά του συστήματος, αυτός που καθορίζει τι θα γίνει και πως θα γίνει κάτι μέσα σε μια εφαρμογή.

Οι ελεγκτές δέχονται δεδομένα από τον έξω κόσμο (χρήστη) αλληλεπιδρούν με το μοντέλο και παρουσιάζουν τα κατάλληλα αποτελέσματα μέσω της προβολής (view) στον χρήστη.

Επίσης είναι υπεύθυνοι για την οργάνωση των δεδομένων όπως αναζήτηση, κατηγοριοποίηση, σε μορφή τέτοια ώστε να ταιριάζει στις ανάγκες της προβολής (view). Η κλάση που αναλαμβάνει να κάνει όλες αυτές τις λειτουργίες είναι η *ActionController* .

3.1.3.4 Παράδειγμα χρήσης MVC

Ο χρήστης αρχικά αιτείται μια ενέργεια. Αυτήν η ενέργεια περνά κατευθείαν στον Ελεγκτή. Τότε ο ελεγκτής σε συνεργασία με το Μοντέλο ζητά τα κατάλληλα δεδομένα από αυτό εφόσον τα δεδομένα είναι αποθηκευμένα στην βάση δεδομένων. Εφόσον είναι ο Ελεγκτής αποκτά τα δεδομένα καλεί την κατάλληλη Προβολή που είναι υπεύθυνη στην εμφάνιση των αποτελεσμάτων. Όταν βρεθεί και η κατάλληλη Προβολή τα αποτελέσματα φτάνουν και πάλι πίσω στον Ελεγκτή ο οποίος μέσω του browser εμφανίζονται στον χρήστη.

3.1.4 Agile Web Development

Η φιλοσοφία του Rails περιλαμβάνει και αυτής του ευέλικτου προγραμματισμού. Το μανιφέστο του ευέλικτου προγραμματισμού περιλαμβάνει 12 αρχές σχεδίασης.

- Πρώτη μας προτεραιότητα είναι η ικανοποίηση του πελάτη μέσω της έγκαιρης και συνεχούς παράδοσης χρήσιμου λογισμικού.
- Οι αλλαγές στις απαιτήσεις είναι ευπρόσδεκτες, ακόμα και σε προχωρημένα στάδια της ανάπτυξης. Οι ευέλικτες διαδικασίες δαμάζουν τις αλλαγές με στόχο την ενίσχυση του ανταγωνιστικού πλεονεκτήματος του πελάτη.
- Παραδίδουμε συχνά λογισμικό που λειτουργεί, σε διαστήματα μερικών εβδομάδων ή μηνών, με προτίμηση στη συντομότερη χρονική κλίμακα.

- Οι προγραμματιστές και οι ειδικοί της αγοράς πρέπει να συνεργάζονται καθημερινά καθ' όλη τη διάρκεια του έργου.
- Θεμελιώνουμε τα έργα γύρω από άτομα με πάθος και ενδιαφέρον. Διαμορφώνουμε το κατάλληλο περιβάλλον, τους παρέχουμε την αναγκαία υποστήριξη και εμπιστευόμαστε την ικανότητά τους να φέρουν σε πέρας την αποστολή τους.
- Η πιο αποδοτική και αποτελεσματική μέθοδος για τη μετάδοση πληροφορίας προς και εντός της ομάδας ανάπτυξης λογισμικού είναι η συνομιλία πρόσωπο με πρόσωπο.
- Το λογισμικό που λειτουργεί είναι το κύριο μέτρο προόδου.
- Οι ευέλικτες διαδικασίες προάγουν την αειφόρο ανάπτυξη.

- Οι χορηγοί, η ομάδα ανάπτυξης λογισμικού και οι χρήστες θα πρέπει να είναι σε θέση να διατηρούν ένα σταθερό ρυθμό επ' αόριστον.
- Η διαρκής έμφαση στην τεχνική αρτιότητα και στην εύρυθμη σχεδίαση ενισχύουν την ευελιξία.
- Η απλότητα -- η τέχνη της μεγιστοποίησης του όγκου της δουλειάς που δεν χρειάζεται να γίνει -- είναι ουσιώδης.
- Οι καλύτερες αρχιτεκτονικές, απαιτήσεις και σχέδια προκύπτουν από ομάδες πού οργανώνονται μόνες τους.
- Σε τακτά χρονικά διαστήματα, η ομάδα συλλογίζεται για το πώς θα γίνει πιο αποτελεσματική, ρυθμίζοντας και προσαρμόζοντας τη συμπεριφορά της αναλόγως.

Μερικά παραδείγματα που φανερώνουν πως το Rails χρησιμοποιεί μεθόδους του agile development είναι:

- Έχουμε την δυνατότητα να ξεκινήσουμε με τον σχεδιασμό (layout), πριν ξεκινήσουμε να ασχολούμαστε με το κομμάτι των δεδομένων. Δεν χρειάζεται να ασχοληθούμε ξανά με το σχεδιαστικό μέρος όταν ξεκινήσουμε να δίνουμε λειτουργικότητα στην εφαρμογή μας, εκτός αν επιθυμούμε να αλλάξουμε την εμφάνισή της.
- Αντίθετα με γλώσσες όπως η C και η Java, μια Rails εφαρμογή δεν χρειάζεται να κάνει μεταγλώττιση για να γίνει εκτελέσιμη. Ο κώδικας σε Ruby ερμηνεύεται αμέσως έτσι δεν χρειάζεται την οποιαδήποτε μεταγλώττιση για να γίνει εκτελέσιμος. Η αλλαγή κάποιου τμήματος κώδικα δίνει στον προγραμματιστή αμέσως αποτελέσματα και έτσι αυξάνεται η ταχύτητα της ανάπτυξης μιας εφαρμογής.
- Το Rails παρέχει ένα χρήσιμο framework αυτόματης δοκιμής (testing) κάποιου τμήματος του κώδικα της εφαρμογής. Οι προγραμματιστές που κάνουν χρήση αυτού του εργαλείου είναι σίγουροι πως δεν υπάρχει περίπτωση κάνοντας δοκιμές να χαλάσουν ότι έχουν φτιάξει μέχρι εκείνη την στιγμή.

3.1.5 Development/Production/Testing

Θα ήταν ωραία να μπορούσαμε να βλέπουμε σε πραγματικό χρόνο τις αλλαγές που κάνουμε στην εφαρμογή. Αυτό θα είχε πολλά όμως προβλήματα όπως:

- Προβλήματα απόδοσης, αφού κάθε φορά θα έπρεπε να επαναφορτώνεται όλη η εφαρμογή (και στον browser του χρήστη αλλά και στον server).
- Δεν θα θέλαμε ο χρήστης να βλέπει το ίδιο φυσικά όσο εμείς «πειραματιζόμαστε» με την εφαρμογή.

Με το rails μπορούμε, καθώς προσφέρει τρία (ταυτόχρονα) περιβάλλοντα εργασίας. Το καθένα χρησιμοποιεί την δικιά του έκδοση κώδικα και την δικιά του βάση. Σαν αποτέλεσμα έχουμε:

- Στο development, βλέπουμε δυναμικά τις αλλαγές που κάνουμε.
- Στο testing βάζουμε κάποιον δοκιμαστικό κώδικα που προσπαθεί να «προβλέψει» την συμπεριφορά της εφαρμογής μας.
- Όταν αποφασίσουμε ότι θέλουμε να δημοσιεύσουμε την καινούρια έκδοση αυτό γίνεται με μία εντολή .

Φυσικά, καθ’ όλη την διάρκεια αυτή, ο χρήστης εξακολουθεί να «βλέπει» την έκδοση που δουλεύει σωστά!

3.2 Rails και Βάσεις Δεδομένων

Το framework Ruby on Rails χρησιμοποιεί σαν προεπιλεγμένο σύστημα διαχείρισης βάσεων δεδομένων το SQLite. Όμως τα συστήματα διαχείρισης βάσεων δεδομένων που υποστηρίζει είναι αρκετά όπως :

- MySQL
- Oracle
- SQL Server
- Postgres
- Firebird
- DB2
- SQLite

Εάν θέλουμε να χρησιμοποιήσουμε διαφορετικό σύστημα διαχείρισης βάσεων δεδομένων το μόνο που έχουμε να κάνουμε είναι να εγκαταστήσουμε την βάση δεδομένων στο σύστημά μας και αλλάζοντας μια παράμετρο σε ένα μικρό αρχείο που έχει πληροφορίες σχετικά με την σύνδεση με την βάση δεδομένων το ***database.yml***

database.yml

development:

adapter: sqlite3

database: db/development.sqlite3

pool: 5

timeout: 5000

Αν θέλουμε να χρησιμοποιήσουμε MySQL που ίσως να χρειάζεται και κωδικό για τον κάθε χρήστη το database.yml θα είναι κάπως ετσι:

database.yml

development:

adapter: mysql

database: library_development

username: root

password: [password]

host: localhost

Το Rails διαχειρίζεται την βάση με το Object Relational Mapping. Σύμφωνα με αυτή την λογική, ένας πίνακας σε μια βάση δεδομένων διαχειρίζεται ως μία κλάση, τα πεδία του πίνακα ως τα δεδομένα-μεταβλητές της κλάσης και οι εγγραφές του πίνακα ως αντικείμενα της κλάσης. Οι κλάσεις αυτές έχουν αρκετές συναρτήσεις για την διαχείριση της βάσης. Έτσι για παράδειγμα εάν υπάρχει ένας πίνακας με το όνομα Cars, θα υπάρχει και μια κλάση στην εφαρμογή με το όνομα Car, η οποία θα αντιστοιχίζεται με τον πίνακα στην βάση δεδομένων. Έτσι για να βρούμε μια εγγραφή στην βάση με Id = 1.

Την βρίσκουμε πολύ απλά γράφοντας

x=Car.find(1)

Ενώ για να αλλαχτεί μια τιμή σε κάποιο πεδίο γράφουμε

x.car = 40

Αυτή η λογική κάνει πιο εύκολη την συγγραφή κώδικα μιας και ο προγραμματιστής δεν χρειάζεται να μάθει γλώσσες όπως η SQL για να υποβάλλει ερωτήματα στην βάση, ώστε να αντλήσει πληροφορίες που είναι αποθηκευμένες στην βάση δεδομένων, αυτό γίνεται πολύ απλά με τυπικές γνώσεις της γλώσσας Ruby και της έννοιας της αντικειμενοστρέφειας.

3.3 Migrations

Ποτέ μία εφαρμογή σε πραγματικές συνθήκες δεν παραμένει σταθερή, πρέπει συνεχώς να εξελίσσεται η βάση δεδομένων. Θέλουμε να μπορούμε με ασφάλεια να επιστρέψουμε στην προηγούμενη κατάσταση που βρισκόταν η βάση εάν κάτι δεν πάει καλά. Όλα αυτά καθίστανται δυνατά χάρη στα Migrations.
Τα Migrations είναι ένας βολικός τρόπος να αλλάξει το σχήμα της βάσης μας κατά την διάρκεια του χρόνου με συνέπεια και εύκολο τρόπο. Χρησιμοποιούνε ένα RubyDSL έτσι ώστε να μην χρειάζεται να γράφουμε SQL αλλά και επιτρέποντας το σχήμα και οι αλλαγές που κάνουμε να είναι ανεξάρτητες από την βάση δεδομένων μας. Μπορείτε να σκεφτείτε κάθε Migration ως μια νέα «έκδοση» της βάσης δεδομένων. Ένα σχήμα ξεκινά με τίποτα , και κάθε Migration χρησιμοποιείτε για να προσθέσετε ή να αφαιρέσετε πίνακες, στήλες, ή καταχωρήσεις. Η Active Record ξέρει πώς να ενημερώσετε το σχήμα σας στο συγκεκριμένο χρονικό διάστημα, επαναφέροντας το από οποιοδήποτε σημείο είναι χρονικά στην τελευταία έκδοση. Επίσης η Active Record θα ενημερώσει το αρχείο db / schema.rb σας για να ταιριάζει με την ανανεωμένη δομή της βάσης δεδομένων σας.

```
class CreateProducts < ActiveRecord::Migration
  def change
    create_table :products do |t|
      t.string :name
      t.text :description

      t.timestamps
    end
  end
end
```

Το παραπάνω Migration προσθέτει έναν πίνακα products με 2 στήλες name και description. Επίσης αυτόματα προσθέτετε και μια στήλη id που είναι και primary key. Η timestamps προσθέτει δύο στήλες, created_at και updated_at. Αυτές οι ειδικές στήλες διαχειρίζονται αυτόματα από το Active Record, εφόσον υπάρχουν. Σημειώστε ότι ορίζουμε την αλλαγή που θέλουμε να συμβεί προς τα εμπρός στο χρόνο. Πριν το migration εκτελεστεί, δεν θα υπάρχει πίνακας. Μετά, ο πίνακας θα υπάρχει. Η Active Record ξέρει πώς να αντιστραφεί αυτό το migration, καθώς και αν κυλήσει η μετάβαση προς τα πίσω, θα αφαιρέσει τον πίνακα.

Για να δημιουργήσουμε ένα migration χρησιμοποιούμε την εντολή

$ bin/ rails generate migration CreateProducts

Αυτό θα δημιουργήσει ένα κενό, αλλά με το κατάλληλο όνομα migration:

```
class CreateProducts < ActiveRecord::Migration
    def change


end
end
```

Εάν το όνομα του migration είναι της μορφής "AddXXXToYYY" ή "RemoveXXXFromYYY» και ακολουθείται από μια λίστα ονομάτων στηλών και τους τύπους, τότε θα δημιουργηθεί ένα migration που περιέχει τις κατάλληλες add_column και remove_column.

$ bin/ rails generate migration AddPartNumberToProducts part_number:string

Αυτό θα δημιουργήσει το παρακάτω migration

```
class AddPartNumberToProducts < ActiveRecord::Migration
  def change
    add_column :products, :part_number, :string
  end
end
```

3.4 Plugins

Το Rails υποστηρίζει την χρήση plugins. Ένα plugin μπορεί να τροποποιεί ή να επεκτείνει μία λειτουργία του framework.Τα plugins παρέχουν :

- Έναν τρόπο ώστε οι προγραμματιστές να μοιράζονται ιδέες «αιχμής» χωρίς να «πειράζουν» τον κυρίως κώδικα του rails.
- Μία κατανεμημένη αρχιτεκτονική που επιτρέπει σε πακέτα κώδικα να ανανεώνονται ανεξάρτητα.
- Ένα τρόπο στους βασικούς προγραμματιστές του rails, ώστε να παρέχουν νέες δυνατότητες και λειτουργίες γρήγορα και χωρίς να επηρεάζουν τον υπάρχοντα κώδικα.

Παραδείγματα:

- RSpec: Στον τομέα του Testing το RSpec είναι κορυφαίο του είδους του. Εξαιρετικό στο να κάνει δοκιμές σε models και controllers.
- Device: Παρέχει μια πλήρη εφαρμογή στα θέματα ταυτοποίησης χρηστών με καρτέλα sign in, sign out, ανάκτησης κωδικού πρόσβασης.

Βέβαια πλέον πιο διαδεδομένα είναι τα RubyGems και όχι τα plugins.

3.5 Εξυπηρετητές Ιστού, Βιβλιοθήκες Λογισμικού

Το Ruby on Rails βασίζεται σε έναν εξυπηρετητή Ιστού για την εκτέλεσή του. Συνήθως προτιμάται ο Mongrel έναντι του WEBrick αλλά μπορεί να χρησιμοποιηθεί και ο Lighttpd, ο Abyss, ο Apache (σαν μονάδα κώδικα - π.χ. Passenger - ή μέσω του CGI, του FastCGI ή του mod_ruby), και πολλοί άλλοι. Από το 2008 ο εξυπηρετητής Passenger δείχνει να προτιμάται αντί για τον Mongrel. Πρόσφατα, παρατηρήθηκε συχνή χρήση του εξυπηρετητή Unicorn.

Το Rails είναι επίσης γνωστό για την εκτενή χρήση των βιβλιοθηκών JavaScript Prototype και Script.aculo.us για Ajax. Το Rails αρχικά έκανε χρήση ελαφρών κλήσεων SOAP για web services - αργότερα αυτό αντικαταστάθηκε από RESTful web services.

Από την έκδοση 2.0, το Ruby on Rails προσφέρει σαν μορφές εξόδου HTML και XML, με τη δεύτερη να χρησιμοποιείται και στα RESTful web services.

Το Ruby on Rails βασίζεται στη Ruby 1.8.6. Η έκδοση 3.0 δεν υποστηρίζει την Ruby 1.8.6 και χρειάζεται Ruby 1.8.7 για να λειτουργήσει.

3.5 Δομικά συστατικά του Rails

- Active Record

Αντικειμενοσχεσιακή αντιστοίχηση (object-relation mapping – ORM) Ουσιαστικά η σύνδεση ανάμεσα στην βάση δεδομένων και την επιχειρησιακή λογική του προγράμματος, δηλαδή τα μοντέλα/κλάσεις.

- Action Pack

Πρακτικά όλη η λειτουργικότητα της εφαρμογής από την πλευρά του χρήστη . Εδώ υλοποιείται τόσο το στρώμα παρουσίασης (View) όσο και ο controller της MVC αρχιτεκτονικής. Το κομμάτι του controller χειρίζεται τις εισερχόμενες αιτήσεις από τον browser του χρήστη και τις δρομολογεί στην κατάλληλη μέθοδο μίας κλάσης controller. Το κομμάτι της παρουσίασης «συνθέτει» την απάντηση που θα σταλθεί πίσω στον browser του χρήστη (π.χ. σε html, xml, κ.λ.π.)

- Prototype

Το κομμάτι που υλοποιεί την AJAX λειτουργικότητα του site, όπως drag-anddrop, οπτικά εφέ κ.λ.π.

- Action Mailer

Το κομμάτι που χειρίζεται την λήψη και την αποστολή emails.

- Action Web Service

Το κομμάτι που επιτρέπει με ευκολία να προσθέσουμε ένα web service στην εφαρμογή. Υποστηρίζει όλες τις γνωστές και διαδεδομένες τεχνολογίες, όπως: SOAP , REST , XML-RPC , WSDL , RSS

3.7 Web Services

Τα web services είναι μια τεχνολογία που επιτρέπει στις εφαρμογές να επικοινωνούν μεταξύ τους ανεξαρτήτως πλατφόρμας και γλώσσας προγραμματισμού. Ένα web service είναι μια διεπαφή λογισμικού (software interface) που περιγράφει μια συλλογή από λειτουργίες οι οποίες μπορούν να προσεγγιστούν από το δίκτυο μέσω πρότυπων μηνυμάτων XML. Χρησιμοποιεί πρότυπα βασισμένα στη γλώσσα XML για να περιγράψει μία λειτουργία (operation) προς εκτέλεση και τα δεδομένα προς ανταλλαγή με κάποια άλλη εφαρμογή. Μια ομάδα από web services οι οποίες αλληλεπιδρούν μεταξύ τους καθορίζει μια εφαρμογή web services. Τα web services λοιπόν αποτελούν μία αρχιτεκτονική κατανεμημένων συστημάτων κατασκευασμένη από πολλά διαφορετικά υπολογιστικά συστήματα τα οποία επικοινωνούν μέσω του δικτύου ώστε να δημιουργήσουν ένα σύστημα. Αποτελούνται από ένα σύνολο από πρότυπα τα οποία επιστρέφουν στους υπεύθυνους για την ανάπτυξη (προγραμματιστές - developers) να υλοποιήσουν κατανεμημένες εφαρμογές (χρησιμοποιώντας διαφορετικά εργαλεία από διαφορετικούς προμηθευτές) ώστε

να κατασκευάσουν εφαρμογές που χρησιμοποιούν ένα συνδυασμό από ενότητες λογισμικού (software modules) οι οποίες καλούνται από συστήματα που ανήκουν σε διαφορετικά τμήματα ενός οργανισμού ή σε διαφορετικούς οργανισμούς.

Μερικά παραδείγματα web services με Ruby on Rails:

- SOAP

Το Simple Object Access Protocol(SOAP) είναι ένα cross-platform και language-independent πρωτόκολλο όπου βασίζεται στην XML και συνήθως (αλλά όχι απαραίτητα) στην HTTP.Χρησιμοποιεί την XML για την κωδικοποίηση των πληροφοριών και HTTP για να μεταφέρει αυτές τις πληροφορίες σε ένα δίκτυο από πελάτες σε servers και αντίστροφα.Η SOAP έχει πολλά πλεονεκτήματα σε σχέση με άλλες τεχνολογίες, όπως COM, CORBA κλπ: για παράδειγμα, είναι σχετικά εύκολη στην ανάπτυξη και τον εντοπισμό σφαλμάτων, έχει επεκτασιμότητα και ευκολία χρήσης, και την ύπαρξη πολλών εφαρμογών για διαφορετικές γλώσσες και πλατφόρμες.

- REST (web application as “virtual state machine”)

Χρησιμοποιεί HTTP πρωτόκολλο και τις μεθόδους GET,POST, PUT, DELETE για ενέργειες τύπου CRUD (Create , Read , Update , Destroy)Σε κάθε μεταβατική κατάσταση το αποτέλεσμα θα είναι μετάβαση σε μια καινούρια σελίδα (τυποποιημένη με XML περιεχόμενο) η μετάβαση στον browser του χρήστη.

3.8 Επίλογος

Στο κεφάλαιο αυτό αναλύσαμε τα κύρια χαρακτηριστικά της πλατφόρμας Ruby on Rails. Ήταν όμως αυτά αρκετά ώστε να μπορέσουμε να δημιουργήσουμε μια web εφαρμογή έστω και μικρής κλίμακας. Η απάντηση είναι όχι και ήρθε η στιγμή να εισχωρήσουμε βαθύτερα στην Ruby on Rails.

4. Εμβαθύνοντας στην Ruby on Rails

Αφού παρουσιάσαμε τα βασικά χαρακτηριστικά της Ruby αλλά και του Rails ήρθε η στιγμή να εμβαθύνουμε λίγο περισσότερο και να παρουσιάσουμε και άλλους πιο εξειδικευμένους τομείς της Ruby on Rails.

4.1 Εγκατάσταση του Rails

Για την εγκατάσταση του Rails στον υπολογιστή μας χρειαζόμαστε τα παρακάτω.

- Να έχει γίνει ήδη η εγκατάσταση της Ruby στον υπολογιστή μας .Η Rails είναι γραμμένη σε Ruby και οι δικές μας εφαρμογές θα γράφονται σε Ruby

επίσης. Η Rails 3 χρειάζεται εκδόσεις Ruby 1.8.7 ή Ruby 1.9.2 και η Rails 4 1.9.3 ή νεότερη.
- Το σύστημα πακέτων της Ruby τα Ruby Gems.
- Μερικές βιβλιοθήκες, ανάλογα με το λειτουργικό σύστημα.
- Μια βάση δεδομένων εκτός και αν χρησιμοποιείς την default της Rails την SQLite.

Για έναν υπολογιστή ανάπτυξης, είναι περίπου αυτά που θα χρειαστείτε (εκτός από έναν επεξεργαστή κειμένου, και θα μιλήσουμε για τα editors ξεχωριστά). Ωστόσο, εάν πρόκειται να αναπτύξετε την εφαρμογή σας, θα χρειαστεί επίσης να εγκαταστήσετε έναν production web server μαζί με κάποιο κώδικα υποστήριξης για να επιτρέψει την Rails να λειτουργήσει αποτελεσματικά.

4.2 Επιλέγοντας την έκδοση Rails.

Υπάρχουν περιπτώσεις που δεν χρειάζεται η δεν θέλουμε να δουλέψουμε με την τελευταία έκδοση της Rails. Ίσως αναπτύσσουμε μια εφαρμογή σε ένα μηχάνημα, αλλά σκοπεύουμε να την χρησιμοποιήσουμε σε ένα άλλο μηχάνημα που περιέχει μια έκδοση του Rails διαφορετική και παλιότερη. Ίσως να δημιουργούμε μια εφαρμογή ακολουθώντας κάποιες οδηγίες από ένα βιβλίο και το βιβλίο αυτό χρησιμοποιεί παλαιότερη έκδοση Rails.

Εάν οποιαδήποτε από αυτές τις καταστάσεις ισχύει για σας, θα πρέπει να γνωρίζεται μερικά πράγματα. Για αρχή, μπορείτε να μάθετε όλες τις εκδόσεις των Rails που έχετε εγκαταστήσει χρησιμοποιώντας την εντολή gem.

gem list --local rails

Μπορείτε επίσης να επαληθεύσετε ποια έκδοση των Rails τρέχετε ως προεπιλογή χρησιμοποιώντας τη εντολή :

rails –v

Η εγκατάσταση μιας άλλης εκδοχής του Rails γίνεται επίσης μέσω της εντολής Gem.

gem install rails –version 3.0.0

Έχοντας εγκαταστημένες δυο εκδόσεις Rails αυτό δεν μπορεί να ωφελήσει κανέναν εκτός και αν υπάρχει τρόπος να επιλέξουμε μια. Αυτό γίνεται. Μπορούμε να επιλέξουμε ποια έκδοση Rails θα χρησιμοποιήσουμε βάζοντας την πλήρη έκδοση ανάμεσα από υπογραμμίσεις.

rails _3.0.0_ --version

Αυτό είναι ιδιαίτερα χρήσιμο όταν δημιουργείτε μια νέα εφαρμογή, διότι αν δημιουργήσετε μια εφαρμογή με μια ειδική έκδοση του Rails, θα συνεχίσετε να χρησιμοποιείτε τη συγκεκριμένη έκδοση του Rails ακόμη και αν οι νεότερες εκδόσεις εγκατασταθούν στο σύστημα μέχρι να αποφασίσετε ότι είναι ώρα για την αναβάθμιση. Για την αναβάθμιση, απλά πρέπει ενημερώσετε την έκδοση Rails στο Gemfile κατάλογο της εφαρμογής σας, και να εκτελέσετε την εντολή

bundle install

4.3 Editors

Γράφουμε τα Rails προγράμματα μας σε editors. Αυτό που έχουμε καταλάβει κατά την διάρκεια των χρόνων είναι ότι κάποιοι editors λειτουργούνε καλύτερα με διαφορετικές γλώσσες και διαφορετικά περιβάλλοντα. Παρόλο που η επιλογή ενός editor είναι θέμα προσωπικό εδώ είναι κάποιες προτάσεις στο τι να προσέξουμε στην επιλογή editor για τον προγραμματισμό μας με ruby on rails.

- Υποστήριξη για την επισήμανση σύνταξης Ruby και HTML.Υποστήριξη για .erb αρχεία (μια μορφή αρχείου Rails που ενσωματώνει Ruby αποσπάσματα μέσα σε HTML).
- Υποστήριξη αυτόματης εσοχής της Ruby . Αυτό είναι πολύ σημαντικό χαρακτηριστικό. Να έχεις έναν editor να ελέγχει τον κώδικα σου και να προειδοποιεί για κακό nesting.
- Υποστήριξη για την εισαγωγή κοινών constructors Ruby και Rails. Θα γράφεις μικρές μεθόδους, και εάν ο IDE δημιουργεί σκελετούς με ένα πάτημα πλήκτρου ή δύο, μπορείς να επικεντρωθείς στα ενδιαφέροντα πράγματα μέσα στις μεθόδους.
- Καλή πλοήγηση ανάμεσα στους φακέλους. Μια εφαρμογή Rails είναι εξαπλωμένη σε πολλά αρχεία. Χαρακτηριστικό παράδειγμα το ότι μια καινούρια εφαρμογή Rails έχει 46 αρχεία σε 34 φακέλους. Όλα αυτά προτού ξεκινήσουμε το γράψιμο. Χρειάζεται ένα περιβάλλον που σας βοηθά να περιηγηθείτε γρήγορα στο σώμα της εφαρμογής.
- Ολοκλήρωση ονομάτων. Τα ονόματα στο Rails τείνουν να είναι μακρά. Ένα ωραίο editor θα αφήσει να πληκτρολογήσετε τους πρώτους χαρακτήρες και στη συνέχεια να προτείνει πιθανές συμπληρώσεις σας με το πάτημα ενός πλήκτρου.

Μερικοί editors που είναι δημοφιλής και χρησιμοποιούνται για τον προγραμματισμό σε Ruby on Rails είναι οι εξής:

TextMate : Ο δημοφιλέστερος editor για Mac OS X (http://macromates.com/)

XCode : Editor για Mac OS X. Περιέχει έναν Organizer που παρέχει πολλά από αυτά που χρειαζόμαστε. (http://developer.apple.com/tools/developonrailsleopard.html.)

E-TextEditor : “The Power of TextMate on Windows.” Για όσους θέλουνε τα χαρακτηριστικά του TextMate στην πλατφόρμα των Windows. (http://e-texteditor.com/)

Aptana RadRails : Είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης Rails περιβάλλον που τρέχει σε Aptana Studio και Eclipse. Τρέχει σε Windows, Mac OS X και Linux. Κέρδισε βραβείο ως το καλύτερο ανοιχτό εργαλείο για την ανάπτυξη κώδικα που να βασίζεται σε Eclipse το 2006. http://www.aptana.com/rails/

NetBeans IDE 6.5 : Τρέχει σε Windows, Mac OS X, Solaris και Linux. Είναι διαθέσιμο για να κάνεις download το πακέτο με την υποστήριξη της Ruby ή ως Ruby πακέτο που μπορεί να γίνει download αργότερα.Εκτός από την ειδική στήριξη για Rails 2.0 υπάρχει υποστήριξη στα migrations στην database και γρήγορη πλοήγηση ανάμεσα στους φακέλους και τα αρχεία της εφαρμογής. http://www.netbeans.org/features/ruby/index.html

JEdit : Είναι ένας πλήρως εξοπλισμένος editor με υποστήριξη για Ruby. Έχει επίσης εκτεταμένη plug-in υποστήριξη. http://www.jedit.org/

Komodo: Είναι ActiveState’s IDE για αντικειμενοστραφής γλώσσες συμπεριλαμβανομένης και της Ruby. http://www.activestate.com/Products/Komodo

RubyMine: Είναι ένα εμπορικό IDE για Ruby που είναι δωρεάν για εκπαιδευτικούς σκοπούς και προγράμματα ανοιχτού κώδικα. http://www.jetbrains.com/ruby/features/index.html

4.4 Δημιουργώντας μια νέα εφαρμογή

Με την εγκατάσταση του Rails θα έχουμε και μια καινούρια γραμμή εντολών Rails, με την οποία μπορούμε να χρησιμοποιήσουμε για την κατασκευή καινούριων εφαρμογών. Αλλά δεν θα μπορούσαμε να χρησιμοποιήσουμε έναν editor για να δημιουργήσουμε μια εφαρμογή απτό μηδέν; Μπορούμε. Αλλά γιατί χρειαζόμαστε την γραμμή εντολών; Την χρειαζόμαστε γιατί η Rails θα χρειαστεί να βρει να εντοπίσει και να χρησιμοποιήσει όλους τους εξωτερικούς «συνεργάτες» που χρειαζόμαστε για την εφαρμογή μας . Αυτό σημαίνει ότι πρέπει να δημιουργήσουμε
μια συγκεκριμένη δομή καταλόγου, ο κώδικας που γράφετε να τοποθετηθεί στις

κατάλληλες θέσεις. Η εντολή Rails δημιουργεί απλά αυτή τη δομή καταλόγου για εμάς και το συμπληρώνει με κάποιο στάνταρ κώδικα Rails.

Για να δημιουργήσετε την πρώτη εφαρμογή Rails , ανοίξτε ένα shell window, και να περιηγηθείτε σε ένα φάκελο όπου θέλετε να δημιουργήσετε την εφαρμογής σας και τον κατάλογο της. Για παράδειγμα εμείς θέλουμε να είμαστε στον φάκελο ptyxiakh και να δημιουργήσουμε μια καινούρια εφαρμογή με το όνομα eshop. Εάν στον ίδιο φάκελο έχουμε ήδη μια εφαρμογή με το ίδιο όνομα θα μας ρωτήσει το σύστημα εάν θέλουμε να κάνουμε overwrite.

C:> cd ptyxiakh

C:/ptyxiakh> rails new eshop

create

create README

create .gitignore

create Rakefile

: : :

: : :

: : :

create tmp/pids

create vendor/plugins

create vendor/plugins/.gitkeep

C:/ptyxiakh>

Η εντολή έχει δημιουργήσει έναν κατάλογο eshop. Μπορούμε να τον δούμε μια λίστα αρχείων και υποκαταλόγων χρησιμοποιώντας ls σε unix και dir σε windows.

C:/ptyxiakh>cd eshop

C:/ptyxiakh/eshop>dir

app/ config.ru doc/ lib/ public/ README test/ vendor/

config/ db/ Gemfile log/ Rakefile script/ tmp/

Όλοι αυτοί οι κατάλογοι και τα αρχεία που περιέχουνε σε αποτρέπουνε λίγο στην αρχή ώστε να ξεκινήσεις να δουλεύεις για πρώτη φορά. Για τώρα θα αγνοήσουμε τα περισσότερα και θα ασχοληθούμε μόνο με έναν κατάλογο τον app. όπου θα γράψουμε την εφαρμογή μας.

Ότι συμπεριλαμβάνεται μέσα σε αυτά τα αρχεία είναι ότι χρειάζεται για να ξεκινήσουμε έναν αυτόματο web-server όπου θα μπορεί να τρέξει την εφαρμογή που μόλις δημιουργήσαμε.

eshop> rails server

=> Booting WEBrick

=> Rails 3.0.0 application starting on http://0.0.0.0:3000}

=> Call with -d to detach

=> Ctrl-C to shutdown server

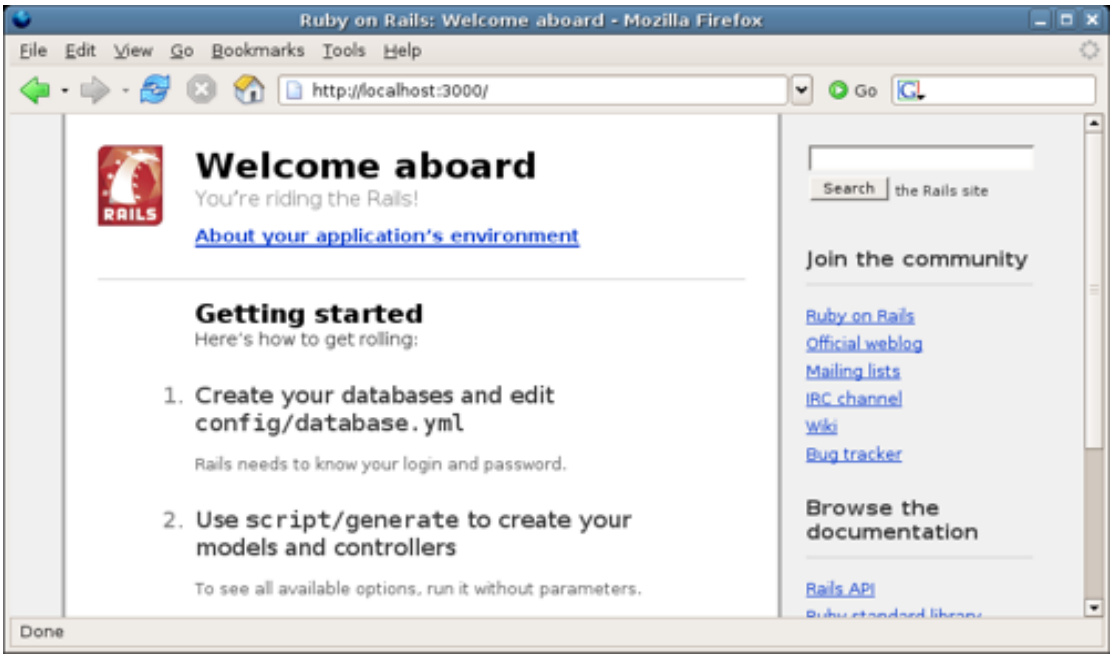
[2009-09-29 10:53:35] INFO WEBrick 1.3.1

[2009-09-29 10:53:35] INFO ruby 1.8.7 (2008-08-11) [x86_64-linux]

[2009-09-29 10:53:40] INFO WEBrick::HTTPServer#start: pid=6044 port=3000

Ποιος web-server θα τρέξει εξαρτάται από ποιον έχουμε εγκαταστημένο στον υπολογιστή μας. Ο WEBrick είναι ένας καθαρά Ruby web-server όπου διανέμεται με την Ruby 1.8.1 και νεότερες εκδόσεις της έτσι θα είναι σίγουρα διαθέσιμος στον υπολογιστή σας.

Από την τελευταία γραμμή του startup καταλαβαίνουμε ότι έχει ξεκινήσει ένας web-server για την θύρα 3000. Το 0.0.0.0 της διεύθυνσης σημαίνει ότι WEBrick θα δεχτεί συνδέσεις σε όλα τα interfaces. Πλέον μπορούμε να έχουμε πρόσβαση στην εφαρμογή μας ανοίγοντας έναν browser και πληκτρολογώντας την διεύθυνση http://localhost:3000 το αποτέλεσμα φαίνεται στην παρακάτω εικόνα. Όταν θα θελήσουμε να κλείσουμε την εφαρμογή και τον WEBrick στην γραμμή εντολών θα πληκτρολογήσουμε CTRL+C και ο web-server θα απενεργοποιηθεί.



Εικόνα 4.1

Αυτήν την στιγμή έχουμε μια εφαρμογή που τρέχει και δεν έχουμε γράψει καθόλου κώδικα.

4.5 Η δομή των βασικών καταλόγων

Όπως αναφέρθηκε πιο πάνω με την δημιουργία μιας νέας εφαρμογής δημιουργούνται αυτόματα κατάλογοι/υποκατάλογοι και αρχεία. Σε αυτήν την ενότητα θα αναλύσουμε τους βασικούς καταλόγους.

eshop/

..../app

...../controller

...../helpers

...../models

...../views

...../layouts

..../components

..../config

..../db

..../doc

..../lib

..../log

..../public

..../script

..../test

..../tmp

..../vendor

README

Rakefile

app.: Εδώ βρίσκονται οι κατάλογοι όπου βρίσκονται τα αρχεία των μοντέλων (models), των όψεων (views) και των ελεγκτών (controllers).

- app/controllers**:** Σε αυτόν τον υποκατάλογο βρίσκονται τα αρχεία των ελεγκτών.
- app/views**:** Εδώ θα ψάξει η εφαρμογή για να βρει ένα αρχείο view.
- app/models**:** Σε αυτόν τον υποκατάλογο βρίσκονται τα αρχεία των μοντέλων.
- app/helpers**:** Εδώ βρίσκονται βοηθητικά αρχεία τα οποία βοηθούν ώστε να μείνει σχετικά μικρός και ευανάγνωστος ο κώδικας που υπάρχει στους ελεγκτές και στα μοντέλα.
- app/views/layouts**:** Εδώ φυλάσσονται τα templates τα οποία θα χρησιμοποιηθούν σε συνδυασμό με τα views.
- components**:** Σε αυτόν τον κατάλογο υπάρχουν μικρές αυτόνομες εφαρμογές που χρησιμοποιούνται στα μοντέλα, στους ελεγκτές και στις όψεις.
- config**:** Σε αυτόν τον κατάλογο βρίσκονται αρχεία ρυθμίσεων που χρειάζονται για την εφαρμογή, όπως το αρχείο για την σύνδεση με την βάση δεδομένων database.yml.
- db**:** Συνήθως σε μια Rails εφαρμογή υπάρχουν μοντέλα τα αποία αντιστοιχούν σε έναν πίνακα της βάσης δεδομένων. Σε αυτόν τον κατάλογο τοποθετούνται scripts τα όποια διαχειρίζονται την βάση δεδομένων.
- doc**:** Η Ruby έχει ένα πολύ χρήσιμο εργαλείο το οποίο ονομάζεται rdoc και παράγει αρχεία με οδηγίες χρήσης (docimentation files) μιας εφαρμογής με

βάση τα σχόλια που υπάρχουν στον κώδικα. Τα αρχεία αυτά τοποθετούνται στον κατάλογο doc.

- lib: Σε αυτόν τον κατάλογο τοποθετούνται οι επιπλέον βιβλιοθήκες που χρησιμοποιεί η εφαρμογή.
- log: Τα αρχεία στα οποία καταγράφεται το ιστορικό από διάφορα λάθη φυλάσσονται σε αυτόν τον φάκελο. Τέτοια αρχεία είναι τα αρχεία που καταγράφουν τα λάθη του server (server.log), της εφαρμογής (development.log) κτλ.
- public: Εδώ βρίσκονται αρχεία τα οποία χρησιμοποιεί η εφαρμογή και δεν αλλάζουν, όπως JavaScript αρχεία, εικόνες, stylesheets κτλ.
- scripts: Σε αυτόν τον κατάλογο φυλάσσονται scripts τα οποία συνήθως χειρίζονται κάποια εργαλεία. Για παράδειγμα εργαλεία τα οποία παράγουν κώδικα, ξεκινούν τον server κτλ.
- tests: Εδώ υπάρχουν αρχεία τα οποία παράγονται όταν εργαζόμαστε στο test environment.
- tmp**:** Σε αυτόν τον κατάλογο βρίσκονται προσωρινά αρχεία τα οποία ίσως χρειαστούν κάποια στιγμή.
- vendor: Εδώ υπάρχουν βιβλιοθήκες οι οποίες είναι για άλλες εφαρμογές, όπως η βάση δεδομένων.
- README: Αυτό το αρχείο παρέχει πληροφορίες σχετικά με το Framework Ruby On Rails.
- rakefile: Αυτό το αρχείο είναι παρόμοιο με το αρχείο makefile που υπάρχει στο Linux. Αυτό το αρχείο βοηθάει στο “χτίσιμο”, την πακετοποίηση (packaging), και στην δοκιμή (testing) του κώδικα.
- Gemfile: Σε αυτό το αρχείο είναι δηλωμένα τα Ruby Gems που χρησιμοποιούνται για την εφαρμογή μας. Μπορούμε να τα προσθέσουμε και να τα αφαιρέσουμε.

4.6 Εμβαθύνοντας στην λειτουργία των καταλόγων

Μεγαλύτερο μέρος της εργασίας μας λαμβάνει χώρα στον κατάλογο app. Ο κύριος κώδικας για την εφαρμογή ζει κάτω από τον κατάλογο app. Θα μιλήσουμε περισσότερο για την δομή του καταλόγου app. Καθώς θα εξερευνούμε τα διάφορα στοιχεία του Rails και κυριότερα του Active Record, Action Controller και Action View.

4.6.1 Ο κατάλογος public

Ο κατάλογος public είναι η εξωτερική όψη της εφαρμογής μας. Ο web-server παίρνει αυτόν τον κατάλογο ως την βάση της εφαρμογής. Στον public μπορούμε να τοποθετήσουμε στατικές σελίδες, stylesheets ,javascript όπως και μερικές ιστοσελίδες.

4.6.2 Ο κατάλογος Lib

Ο κατάλογος lib. Περιέχει τον κώδικα της εφαρμογής όπου δεν ταιριάζει με τον κώδικα του model του view η του controller . Παραδείγματος χάρη μπορεί να έχουμε γράψει μια βιβλιοθήκη που να περιέχει ένα αρχείο PDF σε μορφή απόδειξης το οποίο θα το κατεβάζει ο πελάτης μας. Αυτές οι αποδείξεις θα στέλνονται κατευθείαν από τον controller στον browser (χρησιμοποιώντας την μέθοδο send data). Όμως ο κώδικας για αυτές τις αποδείξεις θα βρίσκεται στο lib.

Ο κατάλογος lib είναι επίσης καλό μέρος να τοποθετήσετε κώδικα που είναι κοινός ανάμεσα σε model view και controller. Γενικά όταν ο κώδικας δεν έχει άμεσα σχέση με το model το view και το controller θα πρέπει να τοποθετείτε στο lib.

Εφόσον έχετε αρχεία στον κατάλογο lib, μπορείτε να τα χρησιμοποιήσετε στην υπόλοιπη σας εφαρμογή. Εάν τα αρχεία που περιέχουν κλάσεις ή ενότητες και τα αρχεία έχουν όνομα με πεζά γράμματα το όνομα της κλάσης ή της μεθόδου, τότε η Rails θα φορτώσει το αρχείο αυτόματα. Για παράδειγμα, θα μπορούσαμε να έχουμε έναν κώδικα PDF απόδειξης σε φάκελο receipt.rb στον κατάλογο lib / pdf_stuff. Όσο η κλάση μας θα έχει όνομά PdfStuff :: Receipt, η Rails θα είναι σε θέση να την βρει και να τη φορτώσει αυτόματα.

Για τις περιπτώσεις που μια βιβλιοθήκη δεν μπορεί να ανταποκριθεί στις συνθήκες της αυτόματης φόρτωσης μπορούμε να χρησιμοποιήσουμε τον μηχανισμό require της Ruby. Αν το αρχείο είναι στον lib κατάλογο μπορουμε να το «απαιτήσουμε» άμεσα με το όνομά του. Για παράδειγμα αν έχουμε ένα αρχείο example.erb θα μπορέσουμε να το συμπεριλάβουμε σε οποιοδήποτε model view ή controller θέλουμε χρησιμοποιώντας τον παρακάτω:

require “example”

Σε περίπτωση όπου μια βιβλιοθήκη θα βρίσκεται σε έναν υποκατάλογο της lib τότε θα πρέπει να συμπεριλάβουμε και το όνομα του καταλόγου και του υποκαταλόγου.

require “catalog/example”

4.6.3 Log

Όπως το Rails τρέχει, παράγει ένα σωρό χρήσιμες πληροφορίες για καταγραφή. Αυτές αποθηκεύονται (από προεπιλογή) στον κατάλογο καταγραφής. Εδώ θα βρείτε τρία κύρια αρχεία καταγραφής, που ονομάζονται development.log, test.log, και production.log. Τα αρχεία καταγραφής περιέχουν περισσότερα στοιχεία και δεδομένα από ότι περιμένουμε. Περιέχουν επίσης χρονικά στατιστικά, πληροφορίες cache, και

επεκτάσεις των καταστάσεων της βάσης δεδομένων που εκτελείται.
Ποιο αρχείο χρησιμοποιείται εξαρτάται από το περιβάλλον στο οποίο η εφαρμογή σας είναι τρέχει.

4.6.4 Ο κατάλογος script

Εάν το βρίσκεται χρήσιμο να γράφεται scripts που εκτελούνται από την γραμμή εντολών και εκτελούνε διάφορες εργασίες για την εφαρμογή σας τότε το κατάλληλο μέρος να τοποθετήσετε το script σας είναι ο script κατάλογος.Ο κατάλογος αυτός περιέχει και το script της Rails. Το script που εκτελείτε πρώτο όταν εκτελείτε μια rails εντολή από την γραμμή εντολών. Το πρώτο επιχείρημα που περνάμε με αυτό το script καθορίζει την λειτουργία Rails που θα εκτελέσει.

- application: Δημιουργεί τον Rails κώδικα της εφαρμογής.
- benchmarker: Δημιουργεί αριθμούς απόδοσης σε μία ή περισσότερες μεθόδους στην εφαρμογή σας.
- console : Σας επιτρέπει να αλληλεπιδράτε με τις μεθόδους της εφαρμογή σας.
- dbconsole : Σας επιτρέπει να αλληλεπιδράτε άμεσα με τη βάση δεδομένων μέσω της γραμμής εντολών.
- generate: Μια γεννήτρια κώδικα. Μπορεί να δημιουργήσει controllers,mailers,models,scaffolds,web services. Μπορούμε επίσης να κατεβάσουμε έξτρα generators(γεννήτριες) από το site της Rails. Τρέχουμε το generate χωρίς arguments. Π.χ. rails generate migration
- destroy : Διαγράφει τα αρχεία τα οποία δημιουργήθηκαν με την εντολή generate.
- plugin : Μας βοηθά να εγκαταστήσουμε και να διαχειριστούμε τα plug-ins που επεκτείνουνε την λειτουργικότητα της Rails.
- profiler: Δημιουργεί ένα run-time profile από ένα URI request το οποίο προωθήθηκε από την εφαρμογή μας.
- runner: Εκτελεί μια μέθοδο στην εφαρμογή σας έξω από το πλαίσιο του Web. Θα μπορούσατε να το χρησιμοποιήσετε για να επικαλεστείτε cache μεθόδους λήξης από μια περιοδική εργασία ή τον χειρισμό των εισερχόμενων e-mail.
- server : Τρέχει την εφαρμογή μας σε έναν αυτόνομο server χρησιμοποιώντας WEBrick. Το χρησιμοποιήσαμε παραπάνω σε ένα παράδειγμα.

4.6.5 Vendor

Το vendor directory χρησιμοποιείτε για την αποθήκευση κώδικα «τρίτου τύπου». Συνήθως στις μέρες μας αυτός ο κώδικας θα προέρχεται από δύο πηγές.

Πρώτον, η Rails εγκαθιστά τα plug-ins στους καταλόγους κάτω από τον vendor / plugins. Τα Plug-ins είναι τρόποι επέκτασης της λειτουργικότητας Rails, τόσο κατά τη διάρκεια της ανάπτυξης και σε πραγματικό χρόνο εκτέλεσης.

Δεύτερον, μπορείτε να εγκαταστήσετε το Rails και όλες τις εξαρτήσεις του στον κατάλογο vendor.

4.6.6 Config directory

Ο κατάλογος περιέχει αρχεία ρυθμίσεων που διαμορφώνουν την Rails. Κατά τη διαδικασία της ανάπτυξης μιας εφαρμογής, θα μπορούμε να διαμορφώσουμε τις διαδρομές (routes), την βάση δεδομένων, να τροποποιήσουμε κάποιες τοπικές ρυθμίσεις, να ορίσουμε οδηγίες ανάπτυξης και πολλά άλλα.

Πριν εκτελέσουμε την εφαρμογή μας η rails φορτώνει και εκτελεί το config/environment.rb και το config/application.rb . Το στάνταρτ περιβάλλον που αυτά τα αρχεία δημιουργούνε περιλαμβάνουνε αυτόματα τους παρακάτω καταλόγους:

- Τον app/controllers και τους υποκαταλόγους του.
- Τον app/models και όλους τους υποκαταλόγους που ξεκινάνε με κάτω παύλα η με μικρό γράμμα.
- Τον vendor κατάλογο και τον lib που συγκαταλέγεται σε κάθε plug-in υποκατάλογο.
- Τους φακέλους app, app/helpers, app/mailers, app/services, config, lib.

Επιπλέον, το Rails θα φορτώσει ένα περιβάλλον ανά αρχείο ρυθμίσεων. Αυτό το αρχείο υπάρχει στον κατάλογο enviroment και βρίσκεται όπου μπορείτε να τοποθετήσετε τις επιλογές διαμόρφωσης που ποικίλει ανάλογα με το περιβάλλον.

Αυτό γίνεται επειδή η Rails αναγνωρίζει ότι οι ανάγκες σας, ως developer, είναι πολύ διαφορετικές όταν γράφετε κώδικα, δοκιμάζετε κώδικα, και όταν τρέχει ο κώδικας. Όταν γράφετε κώδικα, θέλετε εύκολη μεταφόρτωση των αρχείων που άλλαξαν, άμεση κοινοποίηση των σφαλμάτων, και ούτω καθεξής. Κατά τη δοκιμή, θέλετε ένα σύστημα που να υπάρχει στην απομόνωση έτσι ώστε να μπορείτε να έχετε συνεχή και μετά από επαναλήψεις αποτελέσματα. Στην παραγωγή, το σύστημά σας θα πρέπει να είναι συντονισμένο για απόδοση, και οι χρήστες θα πρέπει να φυλάσσονται μακριά από τα λάθη.

Ο διακόπτης που υπαγορεύει το runtime περιβάλλον είναι «ξένο» στην εφαρμογή σας. Αυτό σημαίνει ότι δεν υπάρχει κάποιος κώδικας για την αίτηση της αλλαγής περιβάλλοντος από την ανάπτυξη μέσω δοκιμών με την παραγωγή.

Τέλος η rails σου δίνει την δυνατότητα εάν έχει κάποιος ειδικές ανάγκες να δημιουργήσει το δικό του περιβάλλον εργασίας. Θα πρέπει να προσθέσει μια καινούρια ενότητα στο database configuration file και έναν καινούριο φάκελο στον κατάλογο config/environments.

4.7 Ειδική ονομασία

Οι νέοι με το Rails ενίοτε προβληματίζονται από τον τρόπο με τον οποίο χειρίζεται αυτόματα την ονομασία των πραγμάτων. Είναι έκπληξη το γεγονός ότι ζητούν ένα μοντέλο κλάσης Person και το Rails ξέρει με κάποιο τρόπο να αρχίσει να ψάχνει για ένα πίνακα της βάσης δεδομένων που ονομάζεται people. Σε αυτήν την ενότητα θα μάθετε πώς λειτουργεί αυτή η ονομασία. Οι κανόνες εδώ είναι οι προεπιλεγμένες επιλογές που χρησιμοποιούνται από την Rails. Μπορείτε βέβαια εάν θέλετε να παρακάμψετε όλες αυτές τις συμβάσεις χρησιμοποιώντας τις επιλογές διαμόρφωσης.

4.7.1 Mixed Case, Underscores, and Plurals

Συχνά ονομάζουμε τις μεταβλητές και τις κλάσεις χρησιμοποιώντας μικρές φράσεις. Στην Ruby το σημαντικότερο είναι να έχουμε ονόματα μεταβλητών που όλα τα ονόματα να είναι πεζά και οι λέξεις να χωρίζονται με κάτω παύλα. Οι κλάσεις και τα modules ονομάζεται διαφορετικά,
δεν υπάρχουν παύλες, και κάθε λέξη στη φράση (συμπεριλαμβανομένης της πρώτης) ξεκινά με κεφαλαίο. (Θα ονομάσουμε αυτήν την ιδιότητα Mixed case, για αρκετά προφανείς λόγους.) Οι συμβάσεις αυτές μας οδηγούνε σε ονόματα μεταβλητών, όπως order_status και ονόματα κλάσεων όπως
LineItem.

Η Rails παίρνει αυτή τη σύμβαση και την επεκτείνει με δύο τρόπους. Πρώτον, υποθέτει ότι ονόματα του πίνακα της βάσης δεδομένων, όπως τα ονόματα των μεταβλητών, έχουν πεζά γράμματα και κάτω παύλες μεταξύ των λέξεων. Υποθέτει επίσης ότι τα ονόματα από τους πίνακες της βάσης είναι πάντα στον πληθυντικό αριθμό. Αυτό οδηγεί ονόματα πινάκων όπως orders και third_parties.

Επίσης η rails υποθέτει ότι τα αρχεία ξεκινάνε με πεζά γράμματα και κάτω παύλες.

Η Rails χρησιμοποιεί αυτή τη γνώση τις ονομασίας για να μετατρέψει τα ονόματα αυτόματα. Για παράδειγμα, η εφαρμογή σας μπορεί να περιέχει ένα μοντέλο κλάσης όπου χειρίζεται line_items.Την κλάση θα πρέπει να την ονομάσουμε σύμφωνα με την σύμβαση ονομασίας της Ruby LineItem. Από αυτό το όνομα, η Rails θα συμπεράνει αυτόματα το εξής:

- Ότι ο αντίστοιχος πίνακας της βάσης δεδομένων θα πρέπει να ονομάζεται line_items. Αυτό είναι η το όνομα της κλάσης, μετατρέποντας το σε πεζά γράμματα, με παύλες μεταξύ των λέξεων
  και σε πληθυντικό αριθμό.
- Η Rails επίσης θα πρέπει να ψάξει για ορισμό κλάσης σε ένα αρχείο με το όνομα line_item.rb στον κατάλογο app/models.

Οι controllers της Rails έχουν επιπρόσθετες συμβάσεις ονομασίας. Ας υποθέσουμε ότι η εφαρμογή μας έχει έναν controller ονόματι store τότε συμβαίνουν τα εξής:

- Η Rails υποθέτει ότι η κλάση ονομάζεται StoreController και ότι είναι σε ένα αρχείο με όνομα store_controller.rb στο app / controllers.
- Υποθέτει επίσης ότι υπάρχει ένα helper module με το όνομα StoreHelper στο αρχείο store_helper.rb στον κατάλογο app/helpers.
- Θα ψάξει για view templates για τον συγκεκριμένο controller στον φάκελο app/views/store.
- Εξ ορισμού θα πάρει τα output views και θα τα τοποθετήσει στα layout templates στα αρχεία store.html.erb ή store.xml.erb στο directory app/views/layouts.

Model Naming	
Table	line_items
File	app/models/line_item.rb
Class	LineItem
Controller Naming	
URL	http://../store/list
File	app/controllers/store_controller.rb
Class	StoreController
Method	list

Πινακας 4.1 Πως η ονομασία λειτουργεί στην Rails

Υπάρχει όμως μια τελευταία παράμετρος που πρέπει να προσέξουμε. Σε κώδικα Ruby θα πρέπει να χρησιμοποιήσουμε την λέξη κλειδί για να χρησιμοποιήσουμε αρχεία Ruby προτού αναφερθούμε σε κλάσεις η μεθόδους σε αυτά τα αρχεία. Συνήθως αυτό δεν χρειάζεται στην Rails αφού αυτήν γνωρίζει την σχέση ανάμεσα

σε κλάσεις και αρχεία. Αλλά όταν αναφέρουμε για πρώτη φορά ένα όνομα μιας κλάσης η μιας μεθόδου που δεν είναι γνωστό στην Rails αυτήν μετατρέπει το όνομα της κλάσης σε αρχείο και προσπαθεί να το φορτώσει.

4.7.8 Ομαδοποιώντας τους controllers

Μέχρι στιγμής, όλοι οι controllers μας υπάρχουνε στο app / controllers. Είναι μερικές φορές βολικό να προσθέσουμε περισσότερη λειτουργικότητα σε αυτό. Για παράδειγμα, ο store μπορεί να καταλήξει με μια σειρά από controllers που εκτελούνε σχετικές αλλά ασυνεχές λειτουργίες διαχείρισης. Αντί να μολύνουν το χώρο ονομάτων, θα μπορούσαμε να τους συγκεντρώσουμε σε ένα μοναδικό namespace. Η Rails το κάνει αυτό χρησιμοποιώντας πάλι την ονομασία. Για παράδειγμα εάν έρθει μια κλήση από έναν controller με το όνομα admin/book η Rails θα κοιτάξει για τον controller με το όνομα book_controller στον κατάλογο app/controllers/admin. Δηλαδή η κατάληξη του ονόματος του controller θα αναφέρεται σε ένα αρχείο name_controller.rb και οποιαδήποτε διαδρομή το ακολουθεί μας οδηγεί σε υποκαταλόγους στο app/controllers directory.

4.8 Active Record

Το Active Record είναι η αντικειμενοσχεσιακή χαρτογράφηση (ORM) που παρέχεται από την Rails. Σε αυτήν την ενότητα θα δούμε την χαρτογράφηση δεδομένων από πίνακες γραμμές και στήλες που χρησιμοποίησα στην δημιουργία της εφαρμογή μου. Στην συνέχεια θα χρησιμοποιήσουμε το Active Record να διαχειριστούμε τους πίνακες της βάσης μας θα εξετάσουμε τις ενέργειες create, read, update, delete, γνωστές και ως CRUD. Τέλος θα εξετάσουμε τον κύκλο ζωής ενός αντικειμένου.

4.8.1 Οργανώνοντας με πίνακες

Κάθε υποκατηγορία του ActiveRecord::Base όπως η κλάση μας Order έχει μια ξεχωριστή θέση στον πίνακα δεδομένων. Από default, το Active Record υποθέτει ότι το όνομα του πίνακα σχετίζεται με μια συγκεκριμένη κλάση και είναι ο πληθυντικός του ονόματος της εν λόγω κλάσης. Εάν το
όνομα της κλάσης περιέχει πολλές λέξεις με κεφαλαία γράμματα, το όνομα του πίνακα θεωρείτε ότι έχει παύλες ανάμεσα σε αυτές τις λέξεις.

Class Name	***Table Name***
Order	***orders***
TaxAgency	***tax_agencies***
Person	***people***

LineItem ***line_items***
Πίνακας 5.2

Οι στήλες τώρα σε έναν πίνακα βάσεων δεδομένων δημιουργούνται από το ActiveRecord δυναμικά σε πραγματικό χρόνο εκτέλεσης. Το ActiveRecord αποσπά το σχήμα της βάσης δεδομένων για να διαμορφώσει τις κλάσεις που χρησιμοποιούνε πίνακες.

Στην εφαρμογή μου ο πίνακας orders ορίζεται από το παρακάτω migration.

```
def self.up


    create_table :orders do |t|

t.string :name

t.text :address

t.string :email

t.string :pay_type, :limit => 10

t.timestamps

  end

end
```

Πρέπει να αναφέρουμε πως ο πίνακας orders έχει 5 στοιχεία και όχι 4 (το timestamps εξαιρείτε) όπως φαίνεται. Πάντα από default πρώτο στοιχείο είναι το id. Το timestamps είναι στοιχεία που χρησιμοποιούνται για να αναγράφεται πότε ένα αντικείμενο δημιουργείται αλλά και πότε γίνεται update.

```
eshop> sqlite3 - "select * from orders limit 1"

id = 1

name = Dave Thomas

address = 123 Main St

email = customer@example.com

pay_type = Check

created_at = 2014-06-18 00:36:57.355069

updated_at = 2014-06-18 00:36:57.355069
```

4.8.2 Επιπρόσθετα columns από ActiveRecord

Ένας αριθμός από ονόματα στηλών έχουν ιδιαίτερη σημασία στο ActiveRecord. Εδώ είναι μερικά από αυτά:

- created_at, created_on, updated_at, updated_on : Αυτά ενημερώνονται αυτόματα με το timestamp της δημιουργίας ενός στοιχείου ή την τελευταία ενημέρωση σε αυτό. Βεβαιωθείτε ότι η στήλη της βάσης δεδομένων είναι σε θέση να λαμβάνει μια ημερομηνία, datetime, ή string.
- lock_version : Η Rails θα εντοπίσει τους αριθμούς της έκδοσης και επιχειρεί κλείδωμα στον πίνακα εάν αυτός διαθέτει lock_version.
- type : Χρησιμοποιείτε από τον πίνακα με σκοπό να εντοπίζει τον τύπο του στοιχείου.
- id : Αυτό είναι από default το κύριο κλειδί της στήλης
- xxx_id : Αυτό είναι από default το όνομα ενός ξένου κλειδιού από πίνακα με το όνομα xxx σε πληθυντικό αριθμό.
- xxx_count : Αυτό κρατά έναν μετρητή για το παιδί του πίνακα xxx.

Θα πρέπει να αναφέρουμε πως τόσο τα πρωτεύοντα κλειδιά όσο και τα ξένα κλειδιά διαδραματίζουν ζωτικό ρόλο στη λειτουργία της βάσης δεδομένων.

4.8.3 Σχέσεις μεταξύ στηλών

Στην εφαρμογή που δημιούργησα το LineItems έχει άμεση σχέση με τρία άλλα models. Το Cart το Order και το Product. Όμως τα models μπορούν να έχουν και έμμεση σχέση μεταξύ τους μέσω διάφορων αντικειμένων. Για παράδειγμα υπάρχει σχέση ανάμεσα από Orders και Parts μέσω LineItems. Όλα αυτά είναι πιθανά μέσω των id.

Αν δημιουργείτε ένα νέο σχήμα για μια εφαρμογή Rails, θα θελήσετε πιθανώς να πάτε με την πεπατημένη και να προσθέσετε το πρωτεύον κλειδί id σε όλους τους πίνακες σας. Ωστόσο, αν πρέπει να εργαστείτε με ένα υπάρχον σχήμα, το Active Record σας δίνει την δυνατότητα να παρακάμψετε το default όνομα του πρωτεύοντος κλειδιού για έναν πίνακα. Το θέμα είναι ότι πρέπει να δηλώσουμε εμείς πρωτεύον κλειδί γι’αυτόν τον πίνακα προτού αποθηκεύσουμε τις αλλαγές.

4.8.4 Σχέσεις μεταξύ models

Το ActiveRecord υποστηρίζει τρεις τύπους σχέσεων μεταξύ πινάκων: ένα-προς-ένα, ένα-προς-πολλά, και πολλά-προς-πολλά. Μπορείτε να αναφέρετε αυτές τις σχέσεις με την προσθήκη των δηλώσεων για τα μοντέλα σας: has_one, has_many, belongs_to, has_and_belongs_to_many.

- Σχέσεις one to one

Η σχέση one to one γίνεται με την χρησιμοποίηση ενός ξένου κλειδιού σε έναν πίνακα για να αναφέρουμε την σχέση με ένα στοιχείο ενός άλλου πίνακα. Αυτό γίνεται με το να χρησιμοποιήσουμε την αναφορά has_one στο ένα μοντέλο και belongs_to στο άλλο. π.χ. ανάμεσα από Orders και Product γράφουμε στο μοντέλο Orders:

has_one:Product

και στο μοντέλο Product:

belongs_to:Orders.

- Σχέσεις one to many

Η σχέση ένα προς πολλά μας επιτρέπει την αντιπροσώπευση μιας συλλογής αντικειμένων από ένα στοιχείο. Στην δική μας εφαρμογή μια παραγγελία Order μπορεί να έχει πολλά line items. Για αυτό ο πίνακας line_items θα διαθέτει το order_id. Στο Active Record, το αντικείμενο γονέας (το ένα που περιέχει μια συλλογή αντικειμένων παιδιών) χρησιμοποιεί has_many να δηλώσει τη σχέση του με τον πίνακα των παιδιών, και ο πίνακας του παιδιού χρησιμοποιεί belongs_to.

Και στα models θα δηλώσουμε :

Στο model LineItem:

belongs_to:Order

Στο model Order:

has_many:line_items

- Σχέσεις many to many

Τέλος, μπορούμε να κατηγοριοποιήσουμε τα προϊόντα μας. Ένα προϊόν μπορεί να ανήκει σε πολλές κατηγορίες, και κάθε κατηγορία μπορεί να περιέχει πολλαπλά προϊόντα. Αυτό είναι ένα παράδειγμα μιας σχέσης πολλά-προς-πολλά. Είναι σαν κάθε πλευρά της σχέσης να περιέχει μια συλλογή από αντικείμενα από την άλλη πλευρά. Στη Rails μπορούμε να το εκφράσουμε αυτό με την προσθήκη της δήλωσης has_and_belongs_to_many και στα δύο μοντέλα. Η Rails υλοποιεί αυτήν την σχέση δημιουργώντας έναν καινούριο πίνακα που διαθέτει τα id και των δυο πινάκων σε ζευγάρια.

Στο μοντέλο Order:

has_and_belongs_to_many : Products

Στο μοντέλο Product:

has_and_belongs_to_many: Orders

4.8.5 Creating, Reading, Updating, and Deleting (CRUD)

Αν είστε εξοικειωμένοι με την SQL, θα μπορέσετε να παρατηρήσετε πώς η Rails παρέχει παρόμοιες εντολές όπως αυτές των select, from, where, group by. Αν δεν είστε ήδη εξοικειωμένοι με την SQL, δεν υπάρχει πρόβλημα διότι ένα από τα δυνατά σημεία της Rails είναι οτι αποτρέπει στο να ασχολείστε με τέτοια πράγματα μέχρις ότου πραγματικά πρέπει να έχετε πρόσβαση στη βάση δεδομένων σε αυτό το επίπεδο προγραμματισμού που βρίσκεστε.

Σε αυτό το σημείο θα συνεχίσουμε να εργαζόμαστε με το μοντέλο Order ώστε να βλέπουμε με παραδείγματα τις λειτουργίες που μας παρέχονται.

4.8.5.1 Δημιουργώντας νέες στήλες

Δεδομένου ότι η Rails βλέπει τους πίνακες ως κλάσεις και τις σειρές ως αντικείμενα, όταν δημιουργούμε καινούριες στήλες σε έναν πίνακα είναι παρόμοιο με τη δημιουργία νέων αντικειμένων από μία κλάση. Μπορούμε να δημιουργήσουμε νέα αντικείμενα που αντιπροσωπεύουν στήλες στον πίνακα Orders πληκτρολογώντας Order.new. Τότε μπορούμε να συμπληρώσουμε τις τιμές των γνωρισμάτων (που αντιστοιχούνται σε
στήλες στη βάση δεδομένων). Τέλος, καλούμε την μέθοδο αποθήκευσης του αντικειμένου για να αποθηκευτεί το νέο στοιχείο στην βάση δεδομένων. Χωρίς την παρούσα κλήση της μεθόδου αποθήκευσης, το νέο στοιχείο θα υπήρχε μόνο στην τοπική μας μνήμη

an_order = Order.new

an_order.name = "Dave Thomas"

an_order.email = "dave@example.com"

an_order.address = "123 Main St"

an_order.pay_type = "check"

an_order.save

Τέλος, το Active Record δέχεται ένα hash από τιμές των χαρακτηριστικών ως παράμετρο. Κάθε είσοδος σε αυτό το hash αντιστοιχεί στο όνομα και στην τιμή ενός χαρακτηριστικού που θα οριστεί. Αυτό είναι χρήσιμο για να κάνουμε διάφορα όπως αποθήκευση τιμών από φόρμες HTML σε στήλες στην βάση δεδομένων.

an_order = Order.new(:name => "Dave Thomas" ,

:email => "dave@example.com" ,:address => "123 Main St" ,

:pay_type => "check")

an_order.save

Σημειώστε πως σε αυτά τα παραδείγματα δεν έχουμε δώσει τιμή στο id. Αυτό γιατί το Active Record σαν προεπιλογή δίνει έναν ακέραιο σαν τιμή και θέτει το id.

4.8.5.2 Διαβάζοντας υπάρχοντες στήλες

Για να διαβάσουμε από την βάση δεδομένων θα πρέπει πρώτα να δηλώσουμε ακριβώς ποιες στήλες από την βάση μας ενδιαφέρουνε και αφού δώσουμε έξτρα κριτήρια στο Active Record αυτό θα επιστέψει αντικείμενα που περιέχουνε πληροφορίες από τις στήλες που ταιριάζουνε με τα κριτήρια αυτά.

Ο απλούστερος τρόπος για την εύρεση μιας στήλης από έναν πίνακα είναι καθορίζοντας το πρωτεύον κλειδί του. Κάθε μοντέλο υποστηρίζει την μέθοδο find η οποία δέχεται σαν παράμετρο ένα ή και περισσότερα κλειδιά. Εάν δοθεί μόνο ένα κλειδί, αυτήν επιστρέφει ένα αντικείμενο που περιέχει τα δεδομένα από την αντίστοιχη στήλη. Εάν δοθούνε πολλά κλειδιά τότε επιστρέφει έναν πίνακα από τα αντίστοιχα αντικείμενα.

an_order = Order.find(27) # find the order with id == 27

product_list = params[:product_ids]

total = Product.find(product_list).sum(&:price)

Get a list of product ids from a form, then

sum the total price

Μπορούμε όμως να έχουμε και δυναμική αναζήτηση. Πιθανώς η πιο κοινή έρευνα που πραγματοποιείτε σε βάσεις δεδομένων είναι να έχουμε ως επιστροφή γραμμή ή γραμμές όπου μια στήλη ταιριάζει με μια δεδομένη τιμή. Για παράδειγμα το μοντέλο μας Order έχει χαρακτηριστικά όπως name, email, address. Θα μπορούσαμε να χρησιμοποιήσουμε αυτά τα χαρακτηριστικά σε μεθόδους εύρεσης ώστε να μας επιστρέφουν γραμμές όπου οι στήλες ταιριάζουνε με αυτά τα χαρακτηριστικά.

order = Order.find_by_name("Dave Thomas")

orders = Order.find_all_by_name("Dave Thomas")

orders = Order.find_all_by_email(params['email'])

Τώρα πρέπει να δώσουμε προσοχή στις μεθόδους που παρέχει το ActiveRecord::Relation που μας βοηθά στην αναζήτηση στην βάση δεδομένων μας. Ξεκινάμε με το .first και .all. Όπως ίσως θα έχετε μαντέψει, το .first επιστρέφει την πρώτη γραμμή από την στήλη που ζητάμε. Επιστρέφει nil σε περίπτωση που είναι άδεια. Ομοίως, το .all επιστρέφει όλες τις γραμμές ως έναν πίνακα.

Μερικές ακόμη μεθόδους που υποστηρίζει το ActiveRecord::Relation είναι οι παρακάτω:

- order

Η SQL δεν σε εγγυάται ότι οι γραμμές που θα επιστρέψει θα είναι σε μια συγκεκριμένη σειρά. Η μέθοδος order μας επιτρέπει να επιλέξουμε εμείς ακριβώς πως θέλουμε την επιστροφή των τιμών.

orders = Order.where(:name => 'Dave').

order("pay_type, shipped_at DESC")

Στο παράδειγμα αυτό θα επιστραφούνε οι όλες οι παραγγελίες του Dave ταξινομημένες πρώτα από τον τύπο πληρωμής και έπειτα από το shipped day.

- limit

Μπορούμε να περιορίσουμε τον αριθμό των γραμμών που επιστρέφονται από την κλήση με την μέθοδο limit, αν
χρησιμοποιούμε τη μέθοδο του limit, μπορούμε συγχρόνως εάν θέλουμε να την συνδυάσουμε και με άλλες μεθόδους. Το ακόλουθο παράδειγμα επιστρέφει τις πρώτες δέκα παραγγελίες που ταιριάζουν σε συγκεκριμένα χαρακτηριστικά.

orders = Order.where(:name => 'Dave').

order("pay_type, shipped_at DESC").

limit(10)

- offset

Η μέθοδος offset συνοδεύει αυτήν της limit. Μας δίνει την ευκαιρία να διαμορφώσουμε από την πρώτη κιόλας γραμμή τον τρόπο που θα εμφανιστούνε τα αποτελέσματα.

 # The view wants to display orders grouped into pages,

 # where each page shows page_size orders at a time.

 # This method returns the orders on page page_num (starting at zero).

def Order.find_on_page(page_num, page_size)

order(:id).limit(page_size).offset(page_num*page_size)

end

- select

Από προεπιλογή το ActiveRecord::Relation επιστρέφει όλες τις στήλες ενός πίνακα σαν έχουμε καλέσει το select* from. Αυτό παρακάμπτεται όμως από την μέθοδο select μαζί με ένα string που παίρνει την θέση του *. Έτσι μας επιτρέπεται να περιορίσουμε την επιστροφή τα δεδομένα του πίνακα που θέλουμε να επιστραφούνε.

list = Order.select("name, email")

- joins

Η μέθοδος joins μας επιτρέπει να καθορίσουμε μια λίστα των πρόσθετων πινάκων που πρέπει να ενωθούνε με το προεπιλεγμένο μας πίνακα. Αυτή η παράμετρος εισάγεται μέσα στο SQL αμέσως μετά το όνομα του πίνακα και πριν από οποιουσδήποτε όρους που θα καθοριστούνε. Το παρακάτω παράδειγμα επιστρέφει μια λίστα από line_items για ένα συγκεκριμένο βιβλίο.

LineItem.select('li.quantity').

where("pr.title = 'Programming Ruby 1.9'").

joins("as li inner join products as pr on li.product_id = pr.id")

- readonly

Η μέθοδος readonly προκαλεί την ActiveRecord::Resource να επιστρέψει ActiveRecord αντικείμενα τα οποία δεν μπορούνε να αποθηκευτούν πίσω στην βάση δεδομένων.

- group

Η μέθοδος group προσθέτει μια group by συνθήκη στην SQL με την βοήθεια της find.

- lock

Η lock μέθοδος παίρνει μια προαιρετική συμβολοσειρά ως παράμετρο. Μας επιστρέφει την γραμμή που θέλουμε και μας εγγυάται ότι αυτή την γραμμή δεν μπορεί κανένας να την τροποποιήσει όσο την έχουμε «κλειδωμένη».

4.8.5.3 Update

Αν έχουμε κάποιο αντικείμενο ActiveRecord μπορούμε να το γράψουμε στην βάση μας καλώντας την μέθοδο save. Αυτό θα δουλέψει σωστά εφόσον διαβάστηκε πρώτα από την βάση μας αλλιώς θα δημιουργηθεί μια καινούρια γραμμή στον πίνακα μας. Εάν μια υπάρχουσα γραμμή ενημερώνεται το ActiveRecord χρησιμοποιεί το πρωτεύον κλειδί για να το ταιριάξει με το αντικείμενο στην μνήμη. Τα χαρακτηριστικά που περιέχονται στο αντικείμενο ActiveRecord ποια στήλη θα ενημερώνεται. Αυτό βέβαια θα γίνεται εφόσον μια τιμή έχει αλλαχτεί.

order = Order.find(123)

order.name = "Fred»

order.save

order = Order.find(123)

order.update_attribute(:name, "Barney")

order = Order.find(321)

order.update_attributes(:name => "Barney" ,:email =>

"barney@bedrock.com")

4.8.5.4 Διαγράφοντας στοιχεία

Το ActiveRecord υποστηρίζει δύο στυλ διαγραφής γραμμών. Το πρώτο στυλ περιέχει δυο μεθόδους την delete και την delete_all που λειτουργούνε σε επίπεδο βάσης δεδομένων. Η μέθοδος delete παίρνει ένα id ή έναν πίνακα από id και διαγράφει την γραμμή/γραμμές σε έναν συγκεκριμένο πίνακα. Η μέθοδος delete_all διαγράφει γραμμές που ταιριάζουν σε κάποια συνθήκη ή όλες τις γραμμές εάν δεν υπάρχει κάποια συνθήκη.

Order.delete(123)

User.delete([2,3,4,5])

Product.delete_all(["price > ?" , @expensive_price])

Οι διάφορες μέθοδοι destroy είναι το δεύτερο στυλ διαγραφής γραμμών που παρέχεται από το ActiveRecord. Η μέθοδος destroy διαγράφει από την βάση δεδομένων την γραμμή που αντιστοιχεί σε κάποιο συγκεκριμένο αντικείμενο μοντέλο. Έπειτα παγώνει τα περιεχόμενα του αντικειμένου για την πρόληψη μελλοντικών αλλαγών.

order = Order.find_by_name("Dave")

order.destroy

 # ... order is now frozen

Γιατί όμως να χρειαζόμαστε και τις μεθόδους delete και τις μεθόδους destroy; Η μέθοδος delete προσπερνά κάποιες λειτουργίες του ActiveRecord όπως η callback ενώ η destroy εξασφαλίζει ότι όλες οι λειτουργίες ενημερώνονται για την διαγραφή. Σε γενικές γραμμές, είναι καλύτερα να χρησιμοποιήσετε την μέθοδο destroy, εάν θέλετε να εξασφαλίσετε ότι η βάση δεδομένων σας είναι συνεπής, σύμφωνα με τους επιχειρηματικούς κανόνες που ορίζονται στις κλάσεις των μοντέλων σας.

4.9 Action Dispach και Action Controller

Το Action Pack βρίσκεται στην καρδιά των εφαρμογών Rails. Αποτελείται από τρείς Ruby ενότητες: ActionDispatch, ActionController και ActionView. Το Action Dispatch δρομολογεί τις αιτήσεις στους controllers. Το Action Controller μετατρέπει τα αιτήματα σε απαντήσεις.Το Action View χρησιμοποιείται από τον Action Controller για να διαμορφώσει και να εμφανίσει τις απαντήσεις αυτές. Δουλεύοντας μαζί, αυτές οι τρεις υπο-ενότητες παρέχουν υποστήριξη για την επεξεργασία εισερχόμενων αιτήσεων και τη δημιουργία των εξερχόμενων απαντήσεων. Σε αυτό το κεφάλαιο, θα εξετάσουμε τόσο το Action Dispach οσο και τον Action Controller. Στην επόμενη ενότητα θα εξετάσουμε το Action View.

Όταν εξετάσαμε το ActiveRecord είδαμε πως μπορεί να λειτουργήσει σαν ανεξάρτητη βιβλιοθήκη και θα μπορούσαμε να το χρησιμοποιήσουμε σαν μια non-web εφαρμογή Ruby. Το Action Pack είναι διαφορετικό. Αν και έχουμε την δυνατότητα να το χρησιμοποιήσουμε άμεσα ως framework αυτό δεν θα το κάνουμε. Παρόλο αυτά θα εκμεταλλευτούμε την σφιχτή ενσωμάτωση που προσφέρει η Rails. Ας ξεκινήσουμε να εξετάζουμε στο πως οι εφαρμογές Rails χειρίζονται τα αιτήματα (requests).

4.9.1 Αιτήματα και Controllers

Στην απλούστερη μορφή, μια διαδικτυακή εφαρμογή δέχεται μια εισερχόμενη αίτηση από έναν browser, το επεξεργάζεται και στέλνει μια απάντηση. Η πρώτη ερώτηση είναι πως ξέρει μια εφαρμογή τι θα κάνει με την εισερχόμενη αίτηση; Πως δρομολογεί αυτήν την αίτηση στον σωστό κώδικα; Η Rails παρέχει δυο τρόπους δρομολόγησης αιτήσεων. Έναν ολοκληρωμένο τρόπο που θα χρησιμοποιήσετε όταν χρειάζεται και ένα βολικό τρόπο που θα χρησιμοποιήσετε γενικά όποτε μπορείτε.

Ο ολοκληρωμένος τρόπος σας επιτρέπει να ορίσετε μια άμεση χαρτογράφηση των URLs με βάση τις απαιτήσεις και τις συνθήκες. Ο εύκολος τρόπος σας επιτρέπει να ορίσετε διαδρομές βάσει των πόρων, όπως τα μοντέλα που έχετε ορίσει και επειδή ο εύκολος τρόπος είναι χτισμένος στον ολοκληρωμένο τρόπο, μπορείτε ελεύθερα να αναμίξετε και να ταιριάξετε τις δύο προσεγγίσεις.

Σε αμφότερες τις περιπτώσεις, η Rails κωδικοποιεί τις πληροφορίες της αίτησης στο URL και χρησιμοποιεί ένα υποσύστημα που ονομάζεται Action Dispach για να προσδιορίσει τι πρέπει να γίνει με αυτό το αίτημα. Η όλη διαδικασία είναι πολύ ευέλικτη, στο τέλος η Rails έχει καθορίσει το όνομα του controller που χειρίζεται το συγκεκριμένο αίτημα, μαζί με έναν κατάλογο αιτημάτων με όλες τις παραμέτρους. Κατά την διαδικασία αυτήν είτε μια από τις παραμέτρους είτε η μέθοδος HTTP χρησιμοποιείτε για να καθοριστεί η ενέργεια που θα λάβει μέρος στον υπεύθυνο controller.

4.9.2 REST (Representational State Transfer)

Σε μια προσέγγιση REST, servers επικοινωνούν με τους πελάτες χρησιμοποιώντας stateless συνδέσεις. Όλες οι πληροφορίες σχετικά με την κατάσταση της αλληλεπίδρασης μεταξύ των δύο κωδικοποιείται στα αιτήματα και τις απαντήσεις τους. Η κατάσταση αυτή διατηρείται στον διακομιστή ως ένα σύνολο αναγνωρίσιμων πόρων. Οι πελάτες έχουν πρόσβαση σε αυτούς τους πόρους χρησιμοποιώντας ένα καλά καθορισμένο (με αυστηρούς περιορισμούς)σύνολο αναγνωριστικών πόρων (URLs στο πλαίσιο μας). Το REST διακρίνει το περιεχόμενο των πόρων από την παρουσίαση του εν λόγω περιεχομένου. Το REST έχει σχεδιαστεί για να υποστηρίζει εξαιρετικά τον επεκτάσιμο προγραμματισμό. Αυτήν η περιγραφή όμως είναι λίγο αφηρημένη. Θα πρέπει να εξηγήσουμε στην πράξη τι κάνει το REST.

Πρώτον, οι διατυπώσεις μιας REST προσέγγισης μας δείχνουν ότι οι σχεδιαστές του δικτύου γνωρίζουν πότε και πού μπορούν να αποθηκεύουν προσωρινά τις απαντήσεις σε αιτήματα. Αυτό επιτρέπει το φορτίο να ωθείται εκτός του δικτύου, αυξάνοντας την απόδοση την ανθεκτικότητα ενώ μειώνονται οι λανθάνουσες καταστάσεις. Δεύτερον, οι περιορισμοί που επιβάλλονται από το REST μπορεί να οδηγήσει σε πιο εύκολες ως προς το γράψιμο και την διατήρηση εφαρμογές. Οι RESTful εφαρμογές δεν χρειάζεται να ανησυχούν για την εφαρμογή υπηρεσιών προσβάσιμες εξ αποστάσεως. Αντ 'αυτού, παρέχουνε μια τακτική (και απλή) διεπαφή με ένα σύνολο πόρων. Η εφαρμογή σας υλοποιεί έναν τρόπο εισαγωγής, δημιουργίας, επεξεργασίας και διαγραφής κάθε πόρου, και οι πελάτες κάνουνε τα υπόλοιπα. Ας το κάνουμε λίγο πιο συγκεκριμένο. Στο REST, χρησιμοποιούμε ένα απλό σύνολο ρημάτων που λειτουργούν σε ένα πλούσιο σύνολο ουσιαστικών. Αν χρησιμοποιούμε HTTP, τα ρήματα αντιστοιχούν σε μεθόδους HTTP

(GET, PUT, POST, και DELETE, συνήθως). Τα ουσιαστικά είναι οι πόροι στην εφαρμογή μας. Εμείς αναφέρουμε τους πόρους που χρησιμοποιούν URLs.

Στην εφαρμογή που υλοποίησα υπάρχει ένα σύνολο προϊόντων. Υπάρχουν σιωπηρά δύο πόροι εδώ. Πρώτον, υπάρχουν τα μεμονωμένα προϊόντα. Κάθε ένα αποτελεί έναν πόρο. Υπάρχει επίσης μια δεύτερη πηγή: η συλλογή των προϊόντων.

Για να πάρουμε μια λίστα με όλα τα προϊόντα, θα μπορούσαμε να εκδώσουμε μια αίτηση HTTP GET σε αυτήν την συλλογή, στο μονοπάτι / προϊόντων (path/products). Για να ανακτήσετε τα περιεχόμενα ενός προϊόντος, θα πρέπει πρώτα να το εντοπίσουμε. Ένας τρόπος είναι η Rails να δώσει το πρωτεύον κλειδί αξίας (δηλαδή το id). Πάλι εμείς θα εκδώσουμε μια αίτηση GET, αυτή τη φορά URL / προϊόντα / 1 (URL/products/1).

Για να δημιουργήσουμε ένα νέο προϊόν στη συλλογή μας χρησιμοποιούμε μια αίτηση HTTP POST που κατευθύνεται στη διαδρομή / προϊόντα (path/products) , με τα δεδομένα που θα περιέχει το προϊόν που θέλουμε να προσθέσουμε. Ναι, αυτή είναι η ίδια διαδρομή που χρησιμοποιείται για να πάρουμε μια λίστα των προϊόντων. Αν εκδώσουμε ένα αίτημα GET σε αυτό, απαντά με μια λίστα, και αν κάνουμε μια POST σε αυτό, προσθέτει ένα νέο προϊόν στη συλλογή.

Ας πάμε ένα βήμα πιο παραπέρα. Έχουμε ήδη δει, ότι μπορούμε να ανακτήσουμε το περιεχόμενο ενός προϊόντος όταν εκδώσουμε μια αίτηση GET προς το path / προϊόντα / 1. Για να ενημερώσουμε το προϊόν, θα εκδώσουμε μια αίτηση HTTP PUT κατά την ίδια διεύθυνση URL. και, για να το διαγράψουμε, θα εκδώσουμε ένα HTTP DELETE αίτημα, και πάλι χρησιμοποιώντας την ίδια URL.

Eshop::Application.routes.draw do |map|

resources :products

end

Στον φάκελο routes η γραμμή κώδικα resourses:products δημιουργεί για εμάς εφτά νέες διαδρομές στην εφαρμογή μας όσο αναφορά τα προιόντα (products). Εφόσον προσθέτουμε ή και αφαιρούμε μια γραμμή στο routes θα πρέπει να ανανεώσουμε τις διαδρομές μας και αυτό γίνεται με την εντολή rake routes. Ας δούμε τις διαδρομές που δημιουργήθηκαν:

GET /products(.:format)
{:action=>"index",:controller=>"products"}

products POST /products(.:format)
{:action=>"create",:controller=>"products"}

new_product GET /products/new(.:format)
{:action=>"new", :controller=>"products"}

GET /products/:id(.:format)
{:action=>"show",:controller=>"products"}

PUT /products/:id(.:format)
{:action=>"update",:controller=>"products"}

product DELETE /products/:id(.:format)
{:action=>"destroy",:controller=>"products"}

edit_product GET /products/:id/edit(.:format)
{:action=>"edit", :controller=>"products"}

Τώρα ας ελέγξουμε τις εφτά μεθόδους του controller που οι διαδρομές αναφέρονται.

- index : Επιστρέφει την λίστα από τα αντικείμενα
- create : Δημιουργεί ένα νέο αντικείμενο από τα στοιχεία της αίτησης POST, προσθέτοντας το στην συλλογή.
- new : Κατασκευάζει ένα νέο αντικείμενο και το περνά στον πελάτη. Αυτό το αντικείμενο δεν θα αποθηκευτεί στον server. Μπορείτε να το σκεφτείτε ως την δημιουργίας μια κενής φόρμας για τον πελάτη που θα την συμπληρώσει αυτός.
- show : Επιστρέφει τα περιεχόμενα του αντικειμένου που προσδιορίζονται από το params [: id].
- update : Ενημερώνει τα περιεχόμενα του αντικειμένου που προσδιορίζονται από το params [: id] με τα δεδομένα που σχετίζονται με την αίτηση.
- edit : Επιστρέφει τα περιεχόμενα του αντικειμένου που προσδιορίζονται από το params [: id] σε μια μορφή κατάλληλη για επεξεργασία.

- destroy : Καταστρέφει το αντικείμενο που προσδιορίζεται από το params [: id].

Αν για κάποιο λόγο δεν χρειαζόμαστε ή δεν θέλουμε και τις εφτά μεθόδους, μπορούμε να περιορίσουμε τις ενέργειες που με τη χρήση των επιλογών :only ή :except στο map.resource :

resources :products, :except => [:update, :destroy]

4.9.3 Η μέθοδος render()

Η μέθοδος render είναι η καρδιά της “ανταπόδοσης” στο Rails. Παίρνει ένα hash με επιλογές που μας οδηγούν στο τι πρέπει να αποδώσουμε και πως θα το αποδώσουμε. Στο παρακάτω παράδειγμα έχουμε μια μέθοδο update που θέλουμε όταν αλλαχτούνε οι παράμετροι στο αντικείμενο να μεταφερθούμε στο show αλλιώς θέλουμε την εμφάνιση ενός άλλου template. Αυτό γίνεται καλώντας την μέθοδο render.

def update

@user = User.find(params[:id])

if @user.update_attributes(params[:user])

render :action => show

end

end

Φαίνεται κάπως φυσικό ότι η κλήση της μεθόδου render (και redirect_to) θα πρέπει να με κάποιο τρόπο να τερματίσει την επεξεργασία μιας δράσης. Ας δούμε κάποιες επιλογές της μεθόδου render που χρησιμοποιούνται μέσα στους controllers.

- render ()

Χωρίς κάποια παράμετρο η μέθοδοε render καλεί το default template για την τρέχουσα ενέργεια στον τρέχων controller. Ο παρακάτω κώδικας θα μας μεταφέρει στο template app / views / blog / index.html.erb:

class BlogController < ApplicationController

def index

render

end

end

- render(:text=>string)

Επιστρέφει στον client το συγκεκριμένο string και όχι κάποιο template ή αρχείο HTML.

```
class HappyController < ApplicationController
    def index
        render(:text => "Hello there!" )
    end
end
```

- render(:action => action_name)

Μας οδηγεί σε ένα template μιας ενέργειας του συγκεκριμένου controller.

```
def display_cart
    if @cart.empty?
        render(:action => :index)
    else
    # ...
    end
end
```

- render(:nothing=>true)

Δεν επιστρέφει τίποτα. Βασικά στέλνει ένα άδειο σώμα στον browser.

Όλες οι μορφές του render έχουν προαιρετικές επιλογές παραμέτρων (:layout, :status, :content_type). Η παράμετρος :status παρέχει την τιμή που χρησιμοποιείτε στην κεφαλίδα της απόκρισης HTTP. Η παράμετρος του :layout καθορίζει εάν το αποτέλεσμα του render θα εμφανίζεται σε ένα layout. Αν η παράμετρος είναι false δεν θα υπάρξει κανένα layout. Εάν έχει οριστεί nil ή true το layout θα εμφανιστεί εφόσον υπάρχει και είναι σχετικό με το τρέχων action. Αν η παράμετρος έχει ως value ένα string τότε το string θα θεωρηθεί ως το όνομα του layout που θα χρησιμοποιηθεί στο render. Τέλος η παράμετρος :content_type μας επιτρέπει να ορίσουμε μια τιμή όπου θα περαστεί στον browser σαν επικεφαλίδα content_type HTTP.

4.9.4 Redirect

Μια ανακατεύθυνση (redirect) HTTP αποστέλλεται από έναν εξυπηρετητή σε έναν πελάτη σαν απάντηση σε ένα αίτημα. Σε γενικές γραμμές σαν να λέει, «δεν μπορώ να χειριστώ αυτό το αίτημα, αλλά εδώ είναι μερικά URL που μπορούνε." Η απόκριση της ανακατεύθυνσης περιλαμβάνει μια διεύθυνση URL που ο πελάτης θα πρέπει να δοκιμάσει μαζί με κάποιες πληροφορίες και λέγοντας εάν αυτή η αλλαγή κατεύθυνσης είναι μόνιμη. Συνήθως στην Rails εφαρμογές χρησιμοποιούν ανακατευθύνσεις για να περαστεί η επεξεργασία του αιτήματος από κάποια άλλη ενέργεια.

Οι ανακατευθύνσεις αντιμετωπίζονται πίσω από την «σκηνή» των web browsers. Κανονικά, ο μόνος τρόπος για να ξέρετε ότι έχετε ανακατεύθυνση είναι μια μικρή καθυστέρηση και το γεγονός ότι η διεύθυνση URL της σελίδας που προβάλλετε θα έχει αλλάξει από αυτήν που έχετε ζητήσει. Αυτό το τελευταίο σημείο είναι σημαντικό, το πρόγραμμα περιήγησης αργεί να φορτώσει και αυτό γιατί μια ανακατεύθυνση από τον server πράττει λίγο πολύ το ίδιο όπως ένας τελικός χρήστης να θέλει να φτάσει στην νέα διεύθυνση URL προορισμού με το χέρι (manually).

Ο επαναπροσανατολισμός έχει αποδειχθεί σημαντικός όταν γράφουμε web εφαρμογές που θέλουμε να συμπεριφέρονται σωστά (well-behaved). Η Rails έχει έναν απλό αλλά ισχυρό μηχανισμό ανακατεύθυνσης. Μπορεί να ανακατευθύνει μια δράση σε έναν συγκεκριμένο controller (πέρασμα παραμέτρων), σε μια διεύθυνση URL (ή να απενεργοποιήσετε την τρέχουσα του server), ή στην προηγούμενη σελίδα. Ας ρίξουμε μια ματιά σε αυτές τις τρεις μορφές με τη σειρά.

- redirect_to(:action => ..., options...)

Στέλνει μια προσωρινή ανακατεύθυνση στο πρόγραμμα περιήγησης που βασίζεται στις τιμές που βρίσκονται στο options hash. Η διεύθυνση URL που έχουμε ως στόχο δημιουργείται χρησιμοποιώντας url_for, έτσι ώστε αυτή η μορφή redirect_to να έχει όλες τις ικανότητες της δρομολόγησης του Rails πίσω από αυτήν.

- redirect_to(path)

Ανακατευθύνει προς τη συγκεκριμένη διαδρομή. Εάν η διαδρομή δεν ξεκινά με ένα πρωτόκολλο (όπως http: //), το πρωτόκολλο της τρέχουσας αίτησης θα πρέπει να αλλαχτεί. Η μέθοδος αυτή δεν θα εκτελέσει οποιαδήποτε επανεγγραφή στη διεύθυνση URL, γι 'αυτό δεν πρέπει να χρησιμοποιηθεί για να δημιουργήσει διαδρομές που είναι για σύνδεση με actions στην εφαρμογή (εκτός αν έχετε δημιουργήσει τη διαδρομή χρησιμοποιώντας url_for ή ένα όνομα διαδρομής URL generator).

- redirect_to(:back)

Ανακατευθύνει την URL που δίνεται από την επικεφαλίδα HTTP_REFERER στην τρέχουσα αίτηση.

4.9.5 Rails Sessions

Ένα Rails session είναι ένα hash που συμπεριφέρεται όπως μια δομή που εξακολουθεί να υφίσταται σε όλα τα αιτήματα. Σε αντίθεση με τα cookies, τα sessions μπορούνε να κρατήσουνε οποιαδήποτε αντικείμενα (εφ 'όσον αυτά τα αντικείμενα μπορούνε να δρομολογηθούν), γεγονός που τα καθιστά ιδανικά για την εκμετάλλευση πληροφοριών κατάστασης στις web εφαρμογές.

Για παράδειγμα, στην εφαρμογή μας, χρησιμοποιήσαμε ένα session για να κρατήσει το αντικείμενο shopping_cart μεταξύ των αιτημάτων. Το αντικείμενο cart μπορούσε να χρησιμοποιηθεί στην εφαρμογή μας ακριβώς όπως οποιοδήποτε άλλο αντικείμενο. Αλλά η Rails κανόνισε τα πράγματα ώστε
το cart να σώνεται στο τέλος του χειρισμού κάθε αιτήματος και το πιο σημαντικό, ότι το σωστό cart για μια εισερχόμενη αίτηση αποκατασταινότανε όταν η Rails άρχιζε να χειρίζεται το αίτημα. Χρησιμοποιώντας sessions, μπορούμε να προσποιούμαστε ότι η εφαρμογή μας παραμένει τριγύρω των αιτημάτων.

Και αυτό οδηγεί σε μια ενδιαφέρουσα ερώτηση: πού ακριβώς υπάρχει αυτήν η διαμονή δεδομένων μεταξύ των αιτήσεων; Μια επιλογή είναι ο server να τα στείλει στον client όπως τα cookies. Η άλλη επιλογή είναι να αποθηκεύσετε τα δεδομένα στον server. Υπάρχουν δύο σκέλη σε αυτό.

Πρώτον, η Rails πρέπει να παρακολουθεί τα sessions. Αυτό επιτυγχάνεται με τη δημιουργία (από προεπιλογή) ενός χαρακτήρα 32-bit (που σημαίνει ότι υπάρχουν 16 στην 32 πιθανοί συνδυασμοί). Το κλειδί αυτό λέγεται session_id, και αυτό είναι πραγματικά τυχαίο. Η Rails κανονίζει να αποθηκεύσει αυτό το id ως ένα cookie στον browser του χρήστη. Επειδή οι επόμενες αιτήσεις έρχονται στην εφαρμογή από αυτό το πρόγραμμα περιήγησης, η Rails μπορεί να ανακτήσει το session_id.

Δεύτερον η Rails κρατάει έναν αποθηκευτικό χώρο από δεδομένα session στον server ταξινομημένα με το session id. Έτσι όταν υπάρχει ένα εισερχόμενο αίτημα η Rails ψάχνει στον αποθηκευτικό αυτόν χώρο χρησιμοποιώντας ως αναγνωριστικό το session id. Τα δεδομένα που βρίσκει είναι ένα σειριακό αντικείμενο Ruby. Το αναλύει αυτό και αποθηκεύει το αποτέλεσμα στις ιδιότητες του controller session όπου τα δεδομένα θα είναι διαθέσιμα στον κώδικα της εφαρμογής μας. Η εφαρμογή μπορεί να προσθέσει
και να τροποποιήσει αυτά τα δεδομένα. Όταν ολοκληρωθεί η επεξεργασία κάθε αιτήματος, η Rails γράφει τα δεδομένα του session πίσω στο χώρο αποθήκευσης δεδομένων. Εκεί παραμένει μέχρι να έρθει το επόμενο αίτημα από αυτόν τον browser.

Τι θα πρέπει να αποθηκεύσουμε όμως σε ένα session; Μπορούμε να αποθηκεύσουμε οτιδήποτε θέλουμε, με μερικούς περιορισμούς και προειδοποιήσεις:

- Υπάρχουν ορισμένοι περιορισμοί σχετικά με το τι είδους αντικείμενα μπορούμε να αποθηκεύσουμε σε ένα session. Οι λεπτομέρειες εξαρτώνται από το μηχανισμό αποθήκευσης που θα επιλέξουμε. Στην γενική περίπτωση, τα αντικείμενα σε ένα session θα πρέπει να είναι σειριοποιήσιμα (serialized) . Αυτό σημαίνει, για παράδειγμα ότι δεν μπορούμε να αποθηκεύσουμε ένα αντικείμενο I / O σε session.
- Πιθανώς δεν θέλουμε να αποθηκεύσουμε ογκώδη αντικείμενα στην session data. Καλύτερα να τα αποθηκεύσουμε στην βάση και να τα αναφέρουμε από το session. Αυτό ισχύει και για cookie-based sessions, όπου το συνολικό όριο είναι 4 KB.
- Πιθανώς δεν θέλουμε να αποθηκεύσουμε αντικείμενα που λειτουργούνε ως μετρητές στην session data. Για παράδειγμα, ίσως να θέλουμε να κρατήσουμε την καταγραφή του αριθμού άρθρων σε ένα blog και να το αποθηκεύσουμε στη session data για λόγους απόδοσης. Αν το κάνουμε αυτό και κάποιος άλλος χρήστης προσθέσει ένα άρθρο η καταμέτρηση δεν θα ενημερωθεί.
- Πιθανώς δεν θέλουμε να αποθηκεύσουμε κρίσιμες πληροφορίες αποκλειστικά και μόνο στην session data. Για παράδειγμα, αν η αίτησή μας δημιουργεί έναν αριθμό order_confirmation σε ένα αίτημα και αποθηκεύει τα δεδομένα στην session data, έτσι ώστε να μπορεί να τα αποθηκεύσει στην βάση δεδομένων όταν θα χειρίζεται την επόμενη αίτηση, θα ρισκάρουμε να χάσουμε
  τον αριθμό αυτόν, αν ο χρήστης διαγράψει το cookie από τον browser του. Για αυτόν τον λόγο μία κρίσιμη πληροφορίες πρέπει να είναι μονίμως στη βάση δεδομένων.

4.10 Action View

Έχουμε δει πως ο δρομολογητής καθορίζει ποιος ελεγκτής(controller) πρέπει να χρησιμοποιηθεί και πώς ο ελεγκτής επιλέγει μια ενέργεια(action). Έχουμε επίσης δει πώς ελεγκτής και δράση μαζί αποφασίζουν τι πρέπει να επιστρέψουνε στον χρήστη. Κανονικά η δρομολόγηση αυτήν λαμβάνει χώρα στο τέλος της δράσης, και συνήθως περιλαμβάνει ένα template. Όλη η ενότητα αυτήν περιλαμβάνει ακριβώς αυτό Το ActionView περιλαμβάνει όλες τις λειτουργίες που απαιτούνται για να δρομολογούνται τα templates, συνήθως σε μορφή HTML, XML ή JavaScript. Όπως υποδηλώνει το όνομά του, το ActionView είναι η όψη στο μέρος της τριλογίας MVC.

4.10.1 Χρησιμοποιώντας Templates

Όταν γράφουμε μια προβολή(view) είναι σαν να γράφουμε ένα πρότυπο(template). Για να καταλάβουμε πως ένα template λειτουργεί θα πρέπει να δώσουμε βάση σε τρείς τομείς.

- Που βρίσκονται τα templates
- Σε τι περιβάλλον λειτουργούνε
- Τι περιέχουνε

Η render μέθοδος περιμένει να βρει τα templates στον κατάλογο app / views της τρέχουσας εφαρμογή. Μέσα σε αυτόν τον κατάλογο, η σύμβαση είναι να έχουμε έναν ξεχωριστό υποκατάλογο για τα views του κάθε controller. Η εφαρμογή μας, για παράδειγμα, περιλαμβάνει τους controllers για τα products και το store. Ως αποτέλεσμα, έχουμε templates στο app / views / products και app / views / store. Κάθε κατάλογος περιέχει συνήθως templates για το όνομά της κάθε δράσης του αντίστοιχου controller. Αυτό όμως δεν είναι κάτι το απόλυτο. Μπορούμε να έχουμε και templates που τα ονομάσαμε όπως θέλαμε και τα αποθηκεύσαμε όπου θέλαμε. Απλά χρειάζονται να τα καλέσουμε με διαφορετικό τρόπο όπως:

render(:action => 'fake_action_name')

render(:template => 'controller/name')

render(:file => 'dir/template')

Τα templates περιέχουν ένα μείγμα σταθερού κειμένου και κώδικα. Ο κωδικός προσθέτει δυναμικό χαρακτήρα στο template. Ο κώδικας αυτός τρέχει σε ένα περιβάλλον που του δίνει πρόσβαση στις πληροφορίες που έχουν συσταθεί από τον controller.Όλες οι μεταβλητές του controller είναι διαθέσιμες στο template. Αυτήν είναι η απάντηση στο πώς τα actions αλλάζουνε δεδομένα με τα templates. Τα αντικείμενα του controller flash, headers, logger, params, request, response και session είναι διαθέσιμα ως μέθοδοι εισαγωγής στην προβολή(view). Το τρέχων αντικείμενο είναι προσβάσιμο με το χαρακτηριστικό όνομα του controller. Έτσι επιτρέπετε στο template να χρησιμοποιήσει οποιαδήποτε δημόσια μέθοδο του controller. Το μονοπάτι για τον βασικό κατάλογο των templates βρίσκεται στο χαρακτηριστικό base_path.

Η Rails υποστηρίζει διάφορες μορφές templates:

- Builder templates χρησιμοποιούνε την βιβλιοθήκη Builder για να δημιουργήσουνε XML αρχεία.
- Τα erb templates είναι ένα μείγμα από περιεχόμενο και ενσωματωμένη Ruby. Χρησιμοποιείτε συνήθως για σελίδες HTML.
- Τα RJS templates χρησιμοποιούνε javascript που εκτελείτε στον browser και τυπικά χρησιμοποιούνται να αλληλεπιδρούνε με ιστοσελίδες που χρησιμοποιούνε AJAX.

4.10.2 Βοηθητικές Φόρμες

Η Rails παρέχει βοηθητικές φόρμες που μας βοηθούνε στο να συγκεντρώνουμε στοιχεία και πληροφορίες ως input. Ας δούμε ως παράδειγμα μια φόρμα που περιλαμβάνει πολλές βοηθητικές φόρμες όπως text, textarea, radiobutton, checkbox και άλλες.

```
<%= form_for :model do |form| %>
<p>
 <%= form.label :input %>
 <%= form.text_field :input, :placeholder => 'Enter text here...' %>
 </p>

<p>
<%= form.label :address, :style => 'float: left' %>
<%= form.text_area :address, :rows => 3, :cols => 40 %>
</p>

<p>
<%= form.label :color %>:
<%= form.radio_button :color, 'red' %>
<%= form.label :red %>
<%= form.radio_button :color, 'yellow' %>
<%= form.label :yellow %>
<%= form.radio_button :color, 'green' %>
<%= form.label :green %>
</p>

<p>
```

```
<%= form.label 'condiment' %>:
<%= form.check_box :ketchup %>
<%= form.label :ketchup %>
<%= form.check_box :mustard %>
<%= form.label :mustard %>
<%= form.check_box :mayonnaise %>
<%= form.label :mayonnaise %>
</p>

<p>
<%= form.label :priority %>:
<%= form.select :priority, (1..10) %>
</p>

<p>
<%= form.label :start %>:
<%= form.date_select :start %>
</p>

<p>
<%= form.label :alarm %>:
<%= form.time_select :alarm %>
</p>
<% end %>
```

Εικονα 4.2

4.10.3 Αναλύοντας τις Φόρμες

Σε αυτήν την ενότητα θα ελέγξουμε πώς οι διάφορες ιδιότητες στο μοντέλο να περνούνε μέσα από τον controller στο view στη σελίδα HTML, και πίσω και πάλι μέσα στο μοντέλο. Το μοντέλο αντικειμένου διαθέτει χαρακτηριστικά, όπως το name, country, και password. Το template χρησιμοποιεί βοηθητικές φόρμες για να κατασκευάσει μια φόρμα HTML ώστε να αφήσει το χρήστη να επεξεργαστεί τα δεδομένα μέσα στο μοντέλο. Σημειώστε πως τα πεδία της φόρμας ονομάζονται. Το χαρακτηριστικό του country, για παράδειγμα, οδηγεί σε ένα πεδίο εισαγωγής HTML με το όνομα
user [country].

Όταν ο χρήστης υποβάλλει τη φόρμα, τα δεδομένα στέλνονται πίσω στην εφαρμογή μας. Η Rails εξάγει τα πεδία από τη φόρμα και κατασκευάζει την αγκύλη των παραμέτρων(params hash).

Form parameters :	***params :***
id=123	***{ :id => "123" }***
user[name]=Dave	***{ :user => { :name =>"Dave"}}***
user[address][city]=Wien	***{ :user => { :address => { :city => "Wien" }}}***

Ας δούμε τα βήματα ένα ένα για να κάνουμε update ενός νέου user από την υποβολή της φόρμας.

1)Η εφαρμογή λαμβάνει μια αίτηση να επεξεργαστεί έναν χρήστη. Διαβάζει τα δεδομένα σε ένα μοντέλο User.

2)Καλείτε το template edit.html.erb. Χρησιμοποιεί τις πληροφορίες στο αντικείμενο User.

3)Ο κώδικας HTML στέλνεται στον browser.

4)Όταν υπάρξει ανταπόκριση οι παράμετροι μετατρέπονται σε hash params.

5)Η μέθοδος save χρησιμοποιεί τις παραμέτρους για να βρει τον user και τον δημιουργεί η τον αναθεωρεί.

1)

myapp_controller.rb

```
def edit

@user = User.find(params[:id])

end
```

2)

edit.html.erb

```
<% form_for :user,:url => { :action => 'save', :id => @user } do |f| %>

<%= f.text_field 'name' %></p>

<%= f.text_field 'country' %></p>

<%= f.password_field 'password' %></p>

. . .

<% end %>
```

3)

```
<form action="/myapp/save/1234">

<input name="user[name]" ... >

<input name="user[country]" ... >

<input name="user[password]" ... >

. . .

</form>
```

4)

```
@params = {
:id => 1234,
:user => {
:name => " ... ",
:country => " ... ",
:password => " ... " }
}
```

5)

```
def save
user = User.find(params[:id])
if user.update_attributes(params[:user])
...
end
end
```

4.11 Επίλογος

Αναλύοντας και τα πιο σημαντικά κομμάτια της Ruby on Rails αν κάποιος κατάφερε να τα κατανοήσει θα είναι έτοιμος να ξεκινήσει με την δημιουργία μια νέας εφαρμογής. Βέβαια τα χαρακτηριστικά της Ruby on Rails είναι αμέτρητα και δεν γινότανε να χωρέσουνε όλα σε ένα κεφάλαιο, ούτε σε μία πτυχιακή.

5. Η Εφαρμογή

Σαν μέρος της πτυχιακής εργασίας ήτανε και η υλοποίηση μιας web εφαρμογής με Ruby on Rails. Όπως αναφέραμε και προηγουμένως η εφαρμογή που δημιουργήθηκε είναι ένα μικρής κλίμακας ηλεκτρονικό κατάστημα. Σε αυτό το κεφάλαιο θα γίνει η παρουσίαση του, των χαρακτηριστικών της λειτουργίας του όπως και μερικά σημαντικά κομμάτια κώδικα. Για την δημιουργία της εφαρμογής χρησιμοποιήθηκαν τα παρακάτω εργαλεία.

- JetBrains RubyMine 6.0.1 (editor)

- Ruby 1.9.3p392
- Rails 4.0.3
- SQLite3 (βάση δεδομένων)
- Επιπρόσθετα gems (φαίνονται αναλυτικά στον φάκελο GemFile)

5.1 Ανάλυση εφαρμογής

Η εφαρμογή χωρίζεται στις ενέργειες που ένας χρήστης μπορεί να πραγματοποιήσει και σε αυτές του admin. Ο χρήστης θα μπορεί να:

- Να κάνει εγγραφή στην σελίδα.
- Να κάνει σύνδεση/αποσύνδεση εφόσον είναι εγγεγραμμένος.
- Να αλλάξει τα στοιχεία του.
- Να προσθέσει προϊόντα στο καλάθι αγορών.
- Να ολοκληρώσει την παραγγελία του.
- Να επικοινωνήσει με τους διαχειριστές.

Από την άλλη ο διαχειριστής θα μπορεί να:

- Να κάνει σύνδεση/αποσύνδεση.
- Να δει/προσθέσει/αφαιρέσει εγγεγραμμένους χρήστες.
- Να δημιουργήσει/αλλάξει/αφαιρέσει τα προϊόντα.
- Να δει/διαγράψει τις παραγγελίες των χρηστών.

Η διαφορά ενός user από τον admin δεν έγινε δημιουργώντας δυο διαφορετικά μοντέλα κάτι το οποίο θα μπορούσε να γίνει. Για εξοικονόμηση κώδικα δώσαμε ένα χαρακτηριστικό στον user το :admin στο οποίο από default είναι fault. Αυτό βέβαια μπορεί να το αλλάξει ο διαχειριστής της σελίδας από το Rails Console. Έτσι επιλέγει ποιον θέλει να κάνει admin.

5.2 Οι Controllers της εφαρμογής.

Οι controllers που δημιουργήθηκαν για την ανάγκη της εφαρμογής είναι οι εξής :

- application_controller
- admin_controller
- carts_controller
- contacts_controller
- line_items_controller
- order_controller
- products_controller
- sessions_controller
- store_controller

- users_controller
- welcome_controller

Μέσα σε κάθε controller βρίσκονται οι μέθοδοι που σχετίζονται με αυτόν τον controller. Θα πρέπει να αναφέρουμε πως η βασική κλάση της εφαρμογής μας είναι η application_controller η οποία κληρονομεί όλη την λειτουργικότητα της από την ActionController::Base που είναι ο βασικός ελεγκτής της εφαρμογής. Όλες οι άλλες κλάσεις κληρονομούνε την application_controller.

Τέλος είναι πολύ σημαντικό να αναφέρουμε πως οι controllers carts, line_items, orders, products, users δημιουργήθηκαν με την εντολή (πχ για users) :

rails generate scaffold User

Το scaffold στην rails είναι ένα ολοκληρωμένο πακέτο για μοντέλο. Δημιουργεί το μοντέλο, την βάση δεδομένων για αυτό, τον controller , μεθόδους για δημιουργία, επεξεργασία, ανανέωση, διαγραφή, δηλαδή το (CRUD) όπως αναφερθήκαμε σε προηγούμενο κεφάλαιο και όλα τα views για όλες τις μεθόδους. Όλα αυτά με μόνο μια εντολή.

5.3 Τα μοντέλα της εφαρμογής

Τα μοντέλα που δημιουργήθηκαν για την εφαρμογή μας είναι τα εξής:

- cart (καλάθι αγορών)
- contact (επικοινωνία)
- line_item (αντικείμενο ουράς)
- order (παραγγελία)
- product (προϊόν)
- user (χρήστης)

Οι συσχετίσεις που δημιουργήθηκαν ανάμεσα στα μοντέλα ήταν οι εξής:

1) Ένα προϊόν (product) θα πρέπει να έχει πολλές παραγγελίες(orders) και πολλά προϊόντα ουράς (line_items).
2) Μια παραγγελία (order) θα πρέπει να έχει πολλά προϊόντα ουράς (line_items).
3) Ένα προϊόν ουράς(line_item) θα πρέπει να ανήκει σε ένα προϊόν(product) σε μια παραγγελία(order) και σε ένα καλάθι αγορών(cart).
4) Ένα καλάθι αγορών(cart) θα πρέπει να έχει πολλά προϊόντα ουράς(line_items).

5.4 Το περιβάλλον της εφαρμογής από την μεριά του χρήστη

Η εφαρμογή είναι έτσι σχεδιασμένη ώστε χρήστης και admin να έχουνε διαφορετικές ικανότητες πλοήγησης. Το μόνο κοινό σημείο που έχουνε είναι η αρχική σελίδα όπου γίνεται η σύνδεση του χρήστη ή του admin.

Όταν κάποιος εισέλθει στην σελίδα μας θα δει την αρχική σελίδα.

Εικόνα 5.1

Αυτήν χωρίζεται στην κεντρική και στην πλαϊνή σελίδα. Στην κεντρική σελίδα θα μπορεί κάποιος να εγγραφεί ή αν είναι ήδη εγγεγραμμένος να συνδεθεί. Στην πλαϊνή σελίδα υπάρχει η επιλογή της επιστροφής στην αρχική σελίδα ανά πάσα στιγμή μέσω του Home, της μετάβασης στον κατάλογο των προϊόντων μέσω του Store και της επικοινωνίας με e-mail με τους διαχειριστές μέσω της επιλογής Contact us. Προσοχή. Εάν δεν είναι κάποιος συνδεδεμένος μπορεί να μεταβεί στο κατάστημα με τα προϊόντα αλλά δεν θα μπορεί να προσθέσει κάποιο προϊόν στο καλάθι των αγορών. Ο κώδικας για τα links της πλαϊνής σελίδας βρίσκεται στο /views/layouts/application.html.erb

```
<%= link_to 'Home', welcome_path %><br />
 <%= link_to 'Store', store_index_path %><br />
 <%= link_to 'Contact_us', contacts_new_path %><br />
```

Ο κώδικας της κεντρικής σελίδας βρίσκεται στο /views/welcome/index.html.erb

```
div class="center hero-unit">
  <h2>Welcome to the E-Grocery Store</h2>
  <% if session[:user_id].nil? %>  // Έλεγχος εάν υπάρχει user συνδεδεμένος
    <%= link_to "Log in",  login_path  %><br/>
    <h2>Are you new to E-Grocery?</h2><br/>
      <%= link_to 'Sign Up Now', new_user_path %>
```

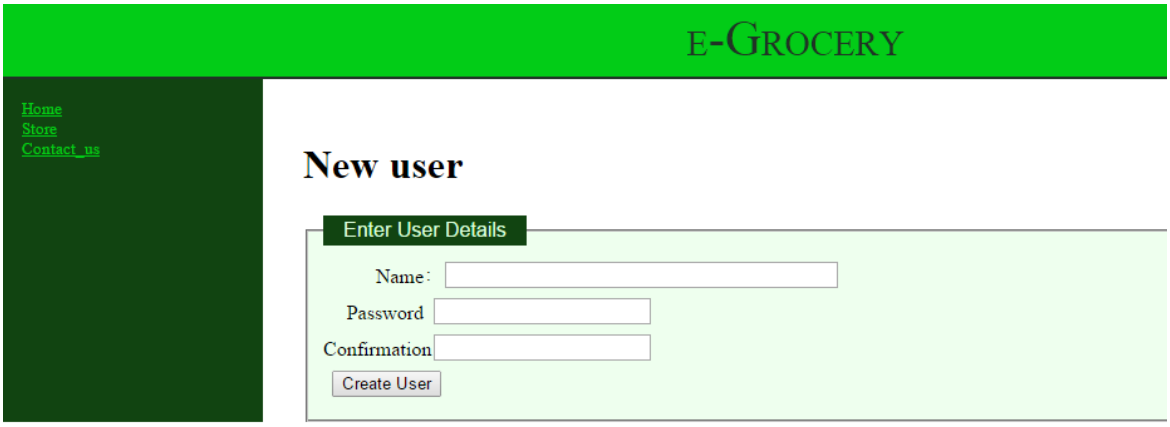
<%end%>

*** ***

Για να εμφανιστεί η κεντρική σελίδα σαν αρχική θα πρέπει να προσθέσουμε το εξής στο φάκελο routes.rb

root :to => 'welcome#index' , :as => 'welcome'

Ας θεωρήσουμε πως κάποιος χρήστης μπαίνει για πρώτη φορά στην σελίδα μας. Θα πατήσει την επιλογή Sing Up Now. Τότε μεταβαίνει στην φόρμα εγγραφής (new_user_path).



Εικόνα 5.2

Στο /views/users/new.html.erb υπάρχει ο εξής κώδικας.

<h1>New user</h1>

<%= render 'form' %>

Έχει δημιουργηθεί μια φόρμα και απλά την καλούμε από εδώ να εμφανιστεί. Έτσι ο κώδικας του νέου χρήστη βρίσκεται στον φάκελο /views/users/_form.html.erb

*** ***

<%= form_for @user do |f| %>

<%= render 'shared/error_messages' %>

<fieldset>

<legend>Enter User Details</legend>

*** ***

<%= f.label :name %>:

<%= f.text_field :name, :size => 40 %>

```
    </div>
    <div>
      <%= f.label :password %>
      <%= f.password_field :password %>
    </div>
    <div>
      <%= f.label :password_confirmation, "Confirmation" %>
      <%= f.password_field :password_confirmation %>
    </div>
    <div>
      <%= f.submit %>
    </div>
  </fieldset>
<% end %>
</div>
```

Με την εισαγωγή των στοιχείων και το πάτημα του κουμπιού καλείτε η μέθοδος create του users_controller.

```
def create
  @user = User.new(user_params) //Καλείτε η user_params που είναι private
  if @user.save
    redirect_to @user
  else
    render 'new'
  end
end
```

def user_params

params.require(:user).permit(:name, :password, :password_confirmation)

end

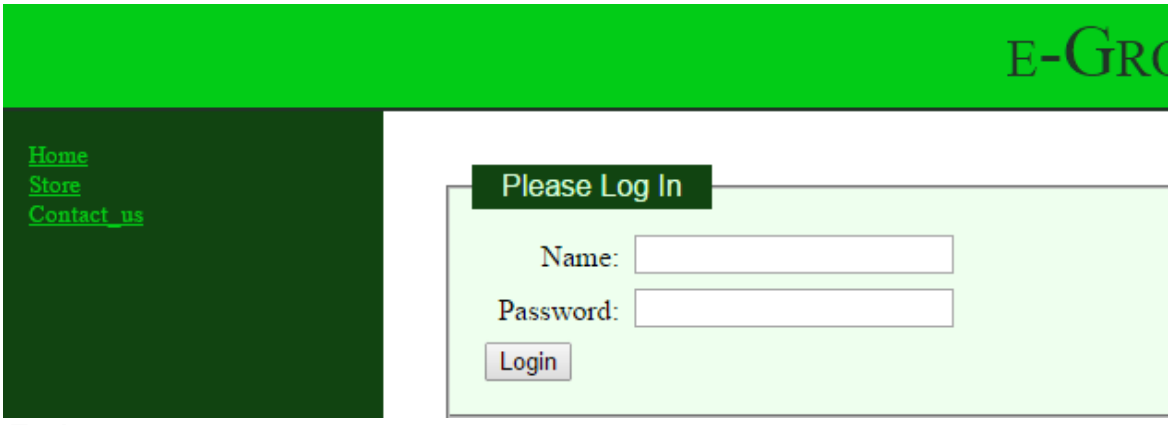
Με την επιλογή has_secure_password που τοποθετούμε στο μοντέλο user ο κωδικός του καθενός κωδικοποιείτε και ούτε ο διαχειριστής δεν μπορεί να τον δει. Επίσης να αναφέρουμε πως αν το όνομα χρησιμοποιείτε ήδη θα βγει μήνυμα σχετικά με αυτό. Όπως επίσης εάν δεν ταιριάζουν οι κωδικοί.

validates :name, :presence => true, :uniqueness => true //Ελέγχουμε την μοναδικότητα του ονόματος

has_secure_password ***//***Αποκρυπτογραφούμε τον κώδικα του χρήστη

Εικόνα 5.3

Με την δημιουργία του χρήστη επιστρέφουμε στην αρχική σελίδα και τώρα χρειάζεται να γίνει το log in. Έτσι μεταφερόμαστε στην σελίδα της σύνδεσης.



Εικόνα 5.4

Συμπληρώνοντας τα στοιχεία μπορούμε να συνδεθούμε. Πατώντας το κουμπί Login καλείτε αυτόματα η μέθοδος create του sessions_controller.

```
def create
  if user = User.authenticate(params[:name], params[:password])
    session[:user_id] = user.id
    if current_user.admin?
      redirect_to admin_path
    else
      redirect_to store_index_path
    end
  else
    redirect_to login_url, :alert => "Invalid user/password combination"
  end
end
```

Έτσι δημιουργούμε ένα session με ένα id το οποίο διαρκεί μέχρι το logout. Αυτό για να δημιουργηθεί πιστοποιεί πρώτα τα στοιχεία με την μέθοδο User.authenticate που βρίσκεται στο μοντέλο του User. Βέβαια βλέπουμε και την επιλογή του εάν ο χρήστης είναι admin ή όχι. Την επιλογή του admin θα την εξετάσουμε παρακάτω. Εάν τα στοιχεία είναι λάθος θα εμφανιστεί μήνυμα εάν είναι σωστά θα μεταβούμε στο store.

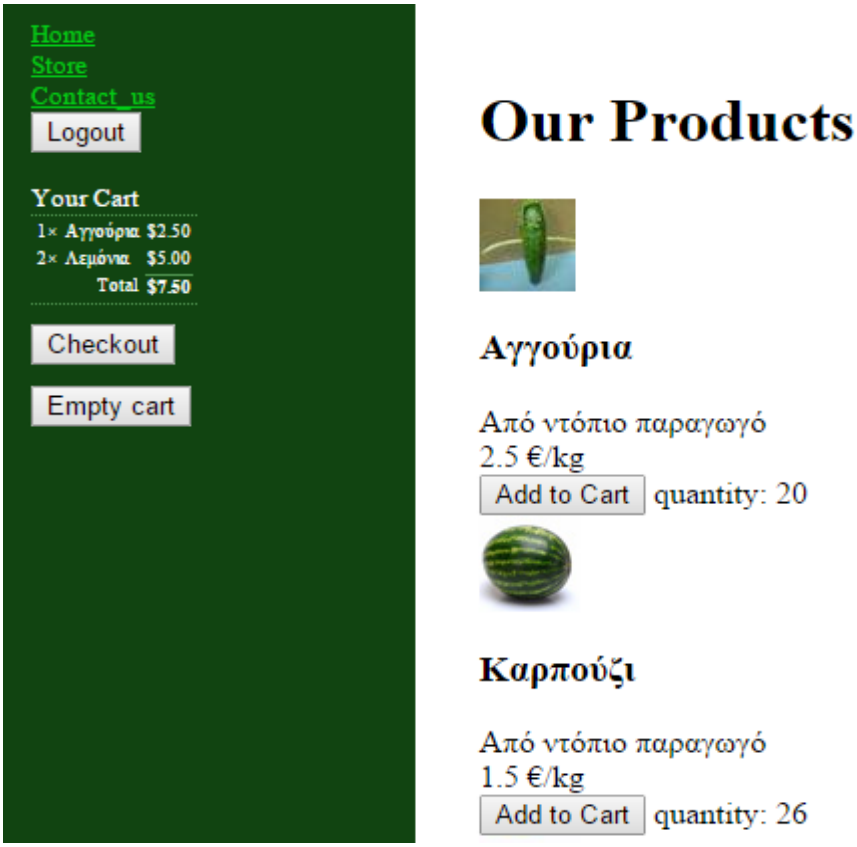
```
def authenticate(name, password)
    if user = find_by_name(name)
      if current_user = user.authenticate(password)
      user
    end
  end
```

Επίσης να αναφέρουμε πως στην πλαϊνή σελίδα εμφανίζεται πλέον η επιλογή logout. Αυτήν η επιλογή όπως και το store φαίνεται στην εικόνα 5.5.

Πλέον ο πελάτης βρίσκεται στο κατάστημα και μπορεί να επιλέξει ανάμεσα από τα αντικείμενα αυτά που θέλει να προσθέσει στο καλάθι. Σε κάθε ένα προϊόν εμφανίζεται και η ποσότητα (quantity). Με κάθε προσθήκη προϊόντος στο καλάθι η ποσότητα του μειώνεται κατά ένα. Όταν μηδενίσει το quantity ο χρήστης δεν μπορεί πλέον να προσθέσει το προϊόν στο καλάθι αγορών.

Με την πρώτη προσθήκη το καλάθι αγορών εμφανίζεται στην πλαϊνή σελίδα κάτω αριστερά. Δείχνει τα προϊόντα την τιμή και το σύνολο των αγορών μας. Επίσης έχει τις επιλογές του αδειάσματος του καλαθιού αλλά και του check in δηλαδή την ολοκλήρωση της αγοράς μας. Αυτό φαίνεται στην εικόνα 5.6.

Εικόνα 5.5



Εικόνα 5.6

Όλη αυτήν η διαδικασία γίνεται ως εξής. Με το πάτημα του κουμπιού Add to Cart καλούμε την μέθοδο create του line_items_controller.

def create

@cart = current_cart //εντοπίζει το cart που θα προστεθούν τα προϊόντα

product = Product.find(params[:product_id]) //εντοπίζει το προϊόν

product.update(:quantity=>product.quantity-1) //μειώνει το quantity κατά ένα

@line_item = @cart.add_product(product.id) //δημιουργεί ένα προϊόν ουράς και το προσθέτει στο καλάθι αγορών.

session[:counter] = 0

respond_to do |format|

if @line_item.save

format.html { redirect_to(store_index_url) }

format.js { @current_item = @line_item }

format.json { render action: 'show', status: :created, location: @line_item }

else

format.html { render action: 'new' }

format.json { render json: @line_item.errors, status: :unprocessable_entity }

end

end

end

Αυτήν η μέθοδος δημιουργεί ένα προϊόν ουράς και το προσθέτει στο καλάθι των αγορών. Αυτό το κάνει χρησιμοποιώντας την μέθοδο του μοντέλου cart add_product.

def add_product(product_id)

current_item = line_items.where(:product_id => product_id).first //δημιουργεί το current_item ώστε να εντοπίσει στο καλάθι εάν υπάρχει ήδη το προϊόν.

if current_item //Εάν ναι

current_item.quantity += 1 //Προσθέτει την ποσότητα

else

current_item = LineItem.new(:product_id=>product_id) //Εάν όχι δημιουργεί ένα νέο προϊόν ουράς

current_item.price = current_item.product.price //προσθέτει την τιμή

line_items << current_item

end

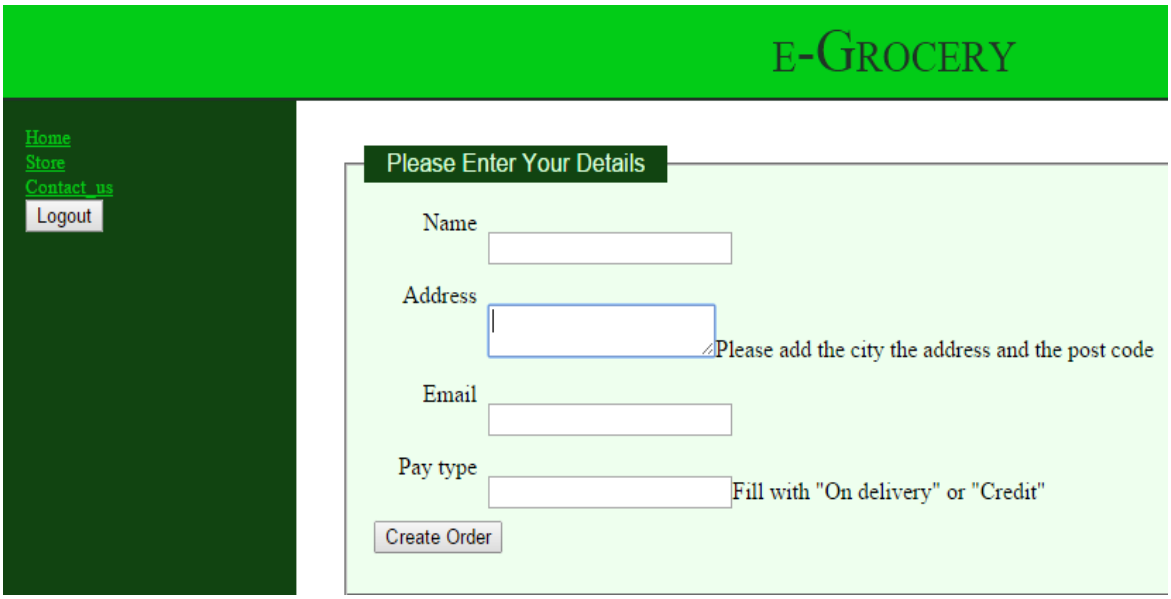
current_item

end

Με την επιλογή Empty cart το καλάθι διαγράφεται και η ποσότητα των προϊόντων επιστρέφει στην αρχική τους κατάσταση. Αυτό συμβαίνει καλώντας την μέθοδο destroy του carts_controller.

```
def destroy
  for item in @cart.line_items //Για κάθε ένα προϊόν που βρίσκεται στο καλάθι
    product=item.product
    product.update(:quantity=>product.quantity+1) //αυξάνουμε την ποσότητα κατά ένα
  end
  @cart.destroy //καταστρέφουμε το καλάθι
  session[:cart_id] = nil //μηδενίζουμε το session
  respond_to do |format|
    format.html { redirect_to(store_index_url) }
    format.json { head :no_content }
  end
end
```

Με την επιλογή Checkout ο χρήστης μεταφέρεται στην σελίδα ολοκλήρωσης της αγοράς.



Εικόνα 5.7

Σε αυτό το σημείο ο χρήστης θα πρέπει να συμπληρώσει τα πραγματικά στοιχεία του όπως το email του για τυχών επικοινωνία και να συμπληρώσει το τύπο πληρωμής που επιθυμεί. Με αντικαταβολή (On delivery) ή με πίστωση (Credit).

Ο κώδικας για την φόρμα αυτή βρίσκεται στον φάκελο /views/orders/new στην οποία πάλι καλούμε μια φόρμα την /views/orders/_form.

```
<%= form_for(@order) do |f| %>

  <% if @order.errors.any? %>

    <div id="error_explanation">

      <h2><%= pluralize(@order.errors.count, "error") %> prohibited this order from being saved:</h2>

      <ul>  //Σε αυτές τις γραμμές συγκεντρώνουμε τα λάθη τα εμφανίζουμε και αποφεύγουμε την αποθήκευση της παραγγελίας

      <% @order.errors.full_messages.each do |msg| %>

        <li><%= msg %></li>

      <% end %>

      </ul>

    </div>

  <% end %>


  <div class="field">

    <%= f.label :name %><br>

    <%= f.text_field :name %>

  </div>

  <div class="field">

    <%= f.label :address %><br>

    <%= f.text_area :address %>Please add the city the address and the post code

  </div>

  <div class="field">

    <%= f.label :email %><br>

    <%= f.text_field :email %>
```

```
  </div>
  <div class="field">
    <%= f.label :pay_type %><br>
    <%= f.text_field :pay_type %>Fill with "On delivery" or "Credit"
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

Εάν το Pay type δεν είναι ένα από τις δύο επιλογές βγάζει μήνυμα λάθους. Με το πάτημα του κουμπιού Create Order η παραγγελία έχει ολοκληρωθεί είναι διαθέσιμη στον admin πλέον. Με την ολοκλήρωση της παραγγελίας ο χρήστης επιστρέφει στο κατάστημα (store). Με την ενέργεια logout ο ο χρήστης αποσυνδέεται από το ηλεκτρονικό μας κατάστημα.

Τέλος στο σύνδεσμο Contact_us μπορεί κάποιος να επικοινωνήσει με τους διαχειριστές τις σελίδας στέλνοντας τους e-mail. Αυτό μπορεί να γίνει αφού πρώτα διαμορφώσουμε τον φάκελο config/environments/development εκεί προσθέσουμε τις ρυθμίσεις για το e-mail (στην εφαρμογή οι ρυθμίσεις είναι για gmail).

```
config.action_mailer.delivery_method = :test
```

```
  config.action_mailer.delivery_method = :smtp
  config.action_mailer.smtp_settings = {
     :address => "smtp.gmail.com" ,
     :port => 587,
     :domain => "domain.of.sender.net" ,
     :authentication => "plain" ,
     :user_name => "shevadotsiko" ,
     :password => "**********" ,
     :enable_starttls_auto => true
```

}

Εικόνα 5.8

5.5 Το περιβάλλον της εφαρμογής από την μεριά του admin.

Με την είσοδο του ο admin μεταφέρεται στην σελίδα που του δείχνει πόσες παραγγελίες έχει. Επίσης στην πλαϊνή σελίδα υπάρχουν οι επιλογές Orders Products Users κάτι που δεν υπήρχε στην περίπτωση του χρήστη. Αυτό συμβαίνει διότι με το login καλείτε η μέθοδος create του sessions_controller. Εκεί γίνεται ο έλεγχος εάν ο χρήστης που συνδέθηκε είναι απλός user ή admin.

def create

if user = User.authenticate(params[:name], params[:password])

session[:user_id] = user.id

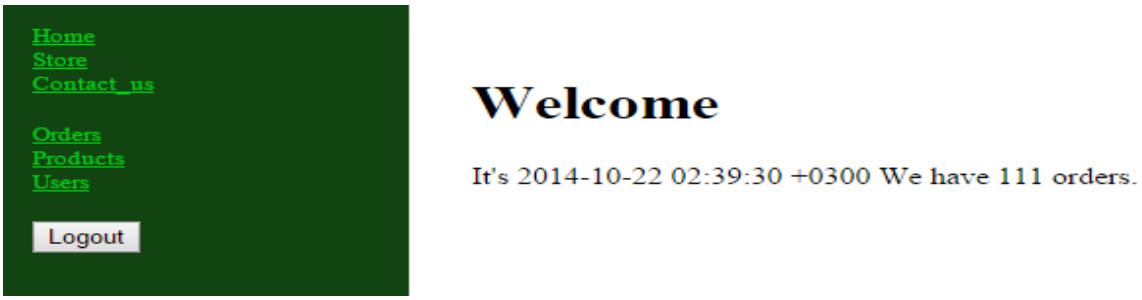
if current_user.admin?

redirect_to admin_path // Εάν είναι admin μεταφέρεται στο admin_path

else

redirect_to store_index_path

end



Εικόνα 5.9

Στην σελίδα Orders φαίνονται όλες οι παραγγελίες ταξινομημένες χρονικά. Όπως και οι επιλογές της εμφάνισης της επεξεργασίας αλλά και τις διαγραφής. Ο κώδικας της σελίδας βρίσκεται στο /views/orders/index.html.erb.

```
<h1>Listing orders</h1>

<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Address</th>
      <th>Email</th>
      <th>Pay type</th>
      <th></th>
      <th></th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    <% @orders.each do |order| %>
      <tr>
        <td><%= order.name %></td>
        <td><%= order.address %></td>
        <td><%= order.email %></td>
        <td><%= order.pay_type %></td>
        <td><%= link_to 'Show', order %></td> // Σύνδεσμος για το Show
        <td><%= link_to 'Edit', edit_order_path(order) %></td> //Σύνδεσμος για το Edit
```

<td><%= link_to 'Destroy', order, method: :delete, data: { confirm: 'Are you sure?' } %></td> //Καλούμε την μέθοδο destroy που διαγράφει την παραγγελία

</tr>

<% end %>

</tbody>

</table>

***
***

<%= link_to 'New Order', new_order_path %>

*** <%= will_paginate @orders %> *** //Οργανώνουμε τις παραγγελίες σε σελίδες

Εικόνα 5.10

Στο show ο admin βλέπει όλες τις λεπτομέρειες ώστε να μπορέσει να προετοιμάσει την παραγγελία. Εδώ ο κώδικας βρίσκεται στο views/orders/show.html.erb

*** <%= notice %> ***

*** ***

Name:

<%= @order.name %>

*** ***

*** ***

Address:

<%= @order.address %>

*** ***

*** ***

Email:

<%= @order.email %>

*** ***

*** ***

Pay type:

<%= @order.pay_type %>

*** ***

*** ***

***Order: ***

<%= render @order.line_items %> // Εμφανίζουμε τα προϊόντα ουράς της συγκεκριμένης παραγγελίας

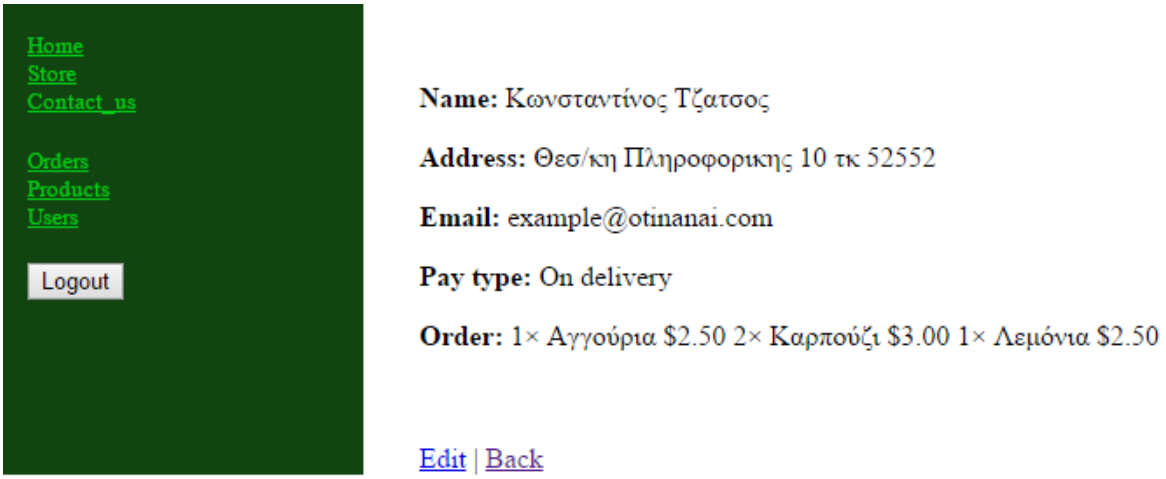
*** ***

***
***

***
***

<%= link_to 'Edit', edit_order_path(@order) %> |

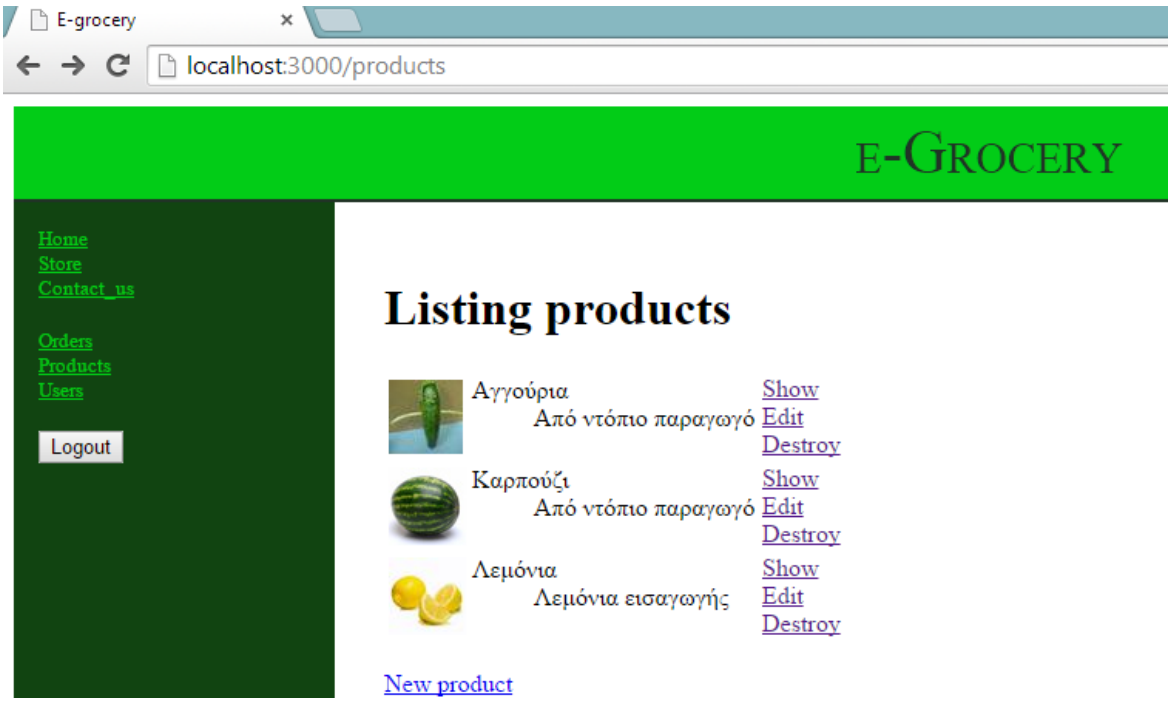
<%= link_to 'Back', orders_path %>



Εικόνα 5.11

Στο edit ο admin μπορεί να αλλάξει τα στοιχεία της παραγγελίας και στο destroy να την διαγράψει από το αρχείο του.

Στη σελίδα των Products ο admin έχει τις επιλογές να δημιουργήσει νέο product, να επεξεργαστεί ή να διαγράψει τα ήδη υπάρχοντα.



Εικόνα 5.12

Η διαφορά με προηγουμένως είναι η δημιουργία νέου προϊόντος. Το Show, Edit και Destroy κάνουν την ίδια δουλειά όπως και στις παραγγελίες.

Στο New product λοιπόν όπως φαίνεται στην Εικόνα 5.13 τοποθετούμε το όνομα, μια περιγραφή του προϊόντος, ένα εικονίδιο για φωτογραφία το οποίο τοποθετείτε στον φάκελο public/images και στο Image url γράφουμε π.χ. /images/name.png.

Επίσης γράφουμε την τιμή την ποσότητα και πατώντας το κουμπί Create Product δημιουργούμε ένα νέο προϊόν το οποίο πλέον θα εμφανίζεται στο κατάστημά μας. Ο κώδικας βρίσκεται στην φόρμα /views/products/_form.html.erb .

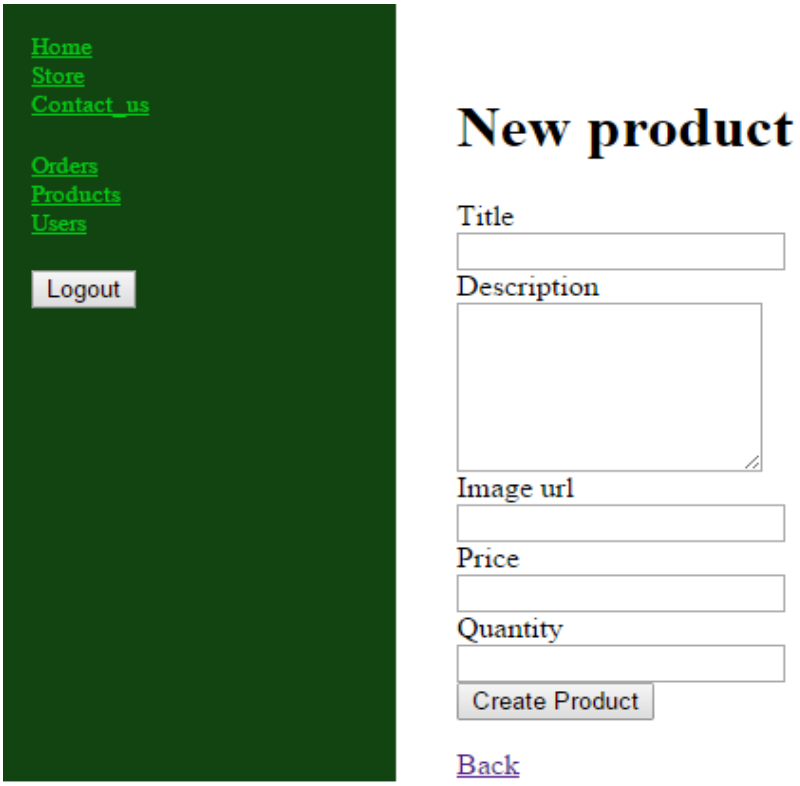
```
<%= form_for(@product) do |f| %>
  <% if @product.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@product.errors.count, "error") %> prohibited this product from being saved:</h2>

      <ul>
      <% @product.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :title %><br>
    <%= f.text_field :title %>
  </div>
  <div class="field">
    <%= f.label :description %><br>
    <%= f.text_area :description, :rows=>6 %>
  </div>
  <div class="field">
    <%= f.label :image_url %><br>
    <%= f.text_field :image_url %>
```

```
  </div>
  <div class="field">
    <%= f.label :price %><br>
    <%= f.text_field :price %>
  </div>
  <div class="field">
      <%= f.label :quantity %><br>
      <%= f.text_field :quantity %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

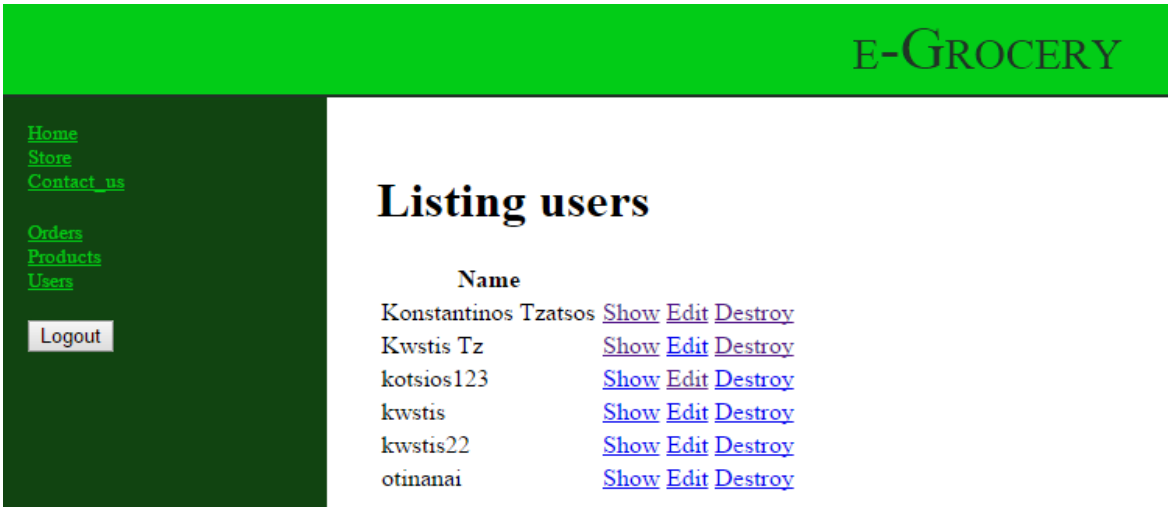
Τις τιμές που θα δέχεται η εφαρμογή μας ως σωστές για την δημιουργία προϊόντος καθορίζεται από εμάς στο μοντέλο του προϊόντος.

```
validates :title, :quantity, :description, :image_url, :presence => true
validates :price, :numericality => {:greater_than_or_equal_to => 0.01}
validates :title, :uniqueness => true
validates :image_url, allow_blank: true, format:{
    with: %r{\.(gif|jpg|png)\Z}i,
    message: 'must be a URL for GIF, JPG or PNG image.'
 }
```

Στην επιλογή Users ο admin μπορεί να δει όλους τους χρήστες που έχουνε εγγραφεί στο σύστημα μας. Έχει επίσης της επιλογές του Show Edit και Destroy. Σαν admin έχει το δικαίωμα να διαγράψει όποιον χρήστη θέλει όμως δεν μπορεί να κάνει edit κάποιον άλλον χρήστη εκτός του εαυτού του. Στις Εικόνες 5.14 φαίνεται η σελίδα users, στην 5.15 το edit του admin και στο 5.16 τι συμβαίνει όταν προσπαθεί να κάνει edit άλλον χρήστη.



Εικόνα 5.13



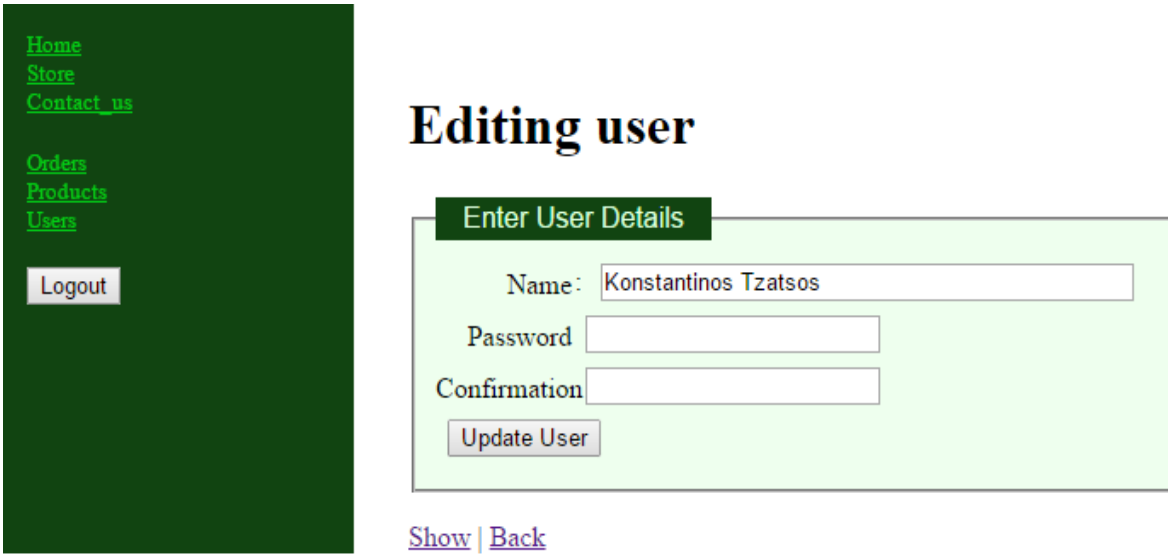
Εικόνα 5.14

Το να μην μπορεί ο admin να κάνει edit κάποιον user το ελέγχουμε από τον controller του user. Ελέγχουμε πριν από τις ενέργειες edit και update εάν ο user είτε είναι admin ή απλός user επιθυμεί να αλλάξει τον δικό του λογαριασμό ή όχι. Αυτό συμβαίνει με τις εντολή:

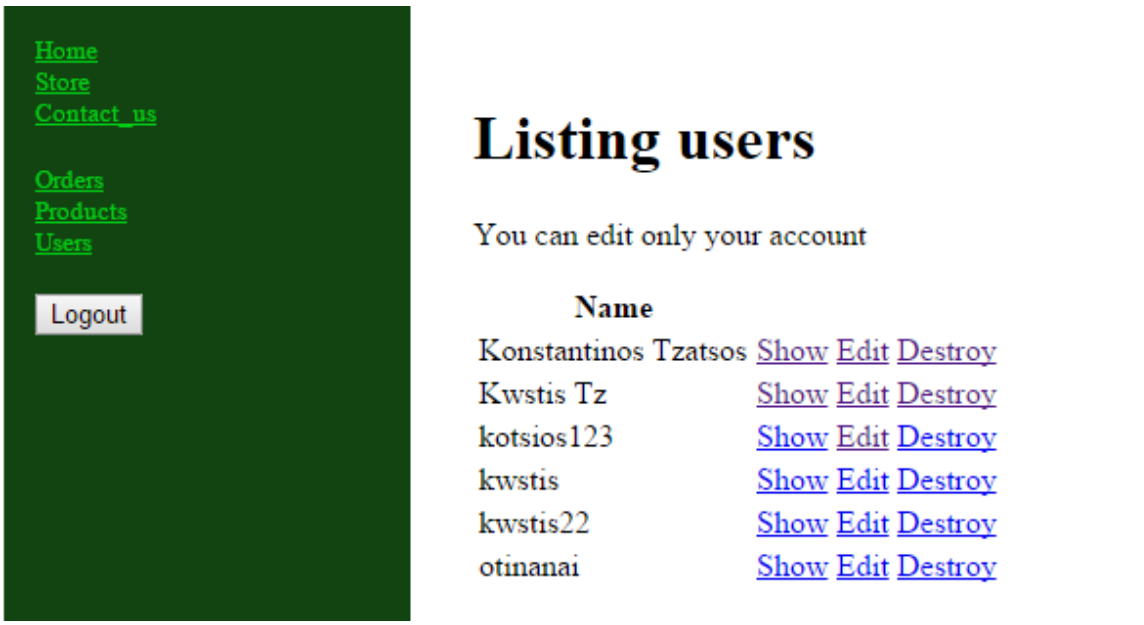
before_action :correct_user, :only => [:edit,:update]

Όπως παρατηρείτε καλείτε η μέθοδος correct_user που επίσης βρίσκεται στον users_controller.

```
def correct_user
   @user = User.find(params[:id])
   if current_user!=@user
     flash[:notice]= "You can edit only your account"
     redirect_to(users_path)
   end
 end
```



Εικόνα 5.15



Εικόνα 5.16

Τέλος με την επιλογή logout ο admin αποσυνδέεται και επιστρέφει στην αρχική σελίδα.

5.5 Επίλογος

Αυτές ήτανε οι λειτουργίες της εφαρμογής που δημιουργήθηκε. Τα γραφικά της ήτανε απλά διότι ήθελα να δώσω έμφαση στα χαρακτηριστικά μόνο της Ruby on Rails.

6. Συμπεράσματα

Τελειώνοντας και το τελευταίο κομμάτι τις πτυχιακής έπειτα από πολλούς μήνες ενασχόλησης, είμαι σε θέση να πω πως έχω αποκτήσει μια μικρή εμπειρία στην ανάπτυξη μιας WEB εφαρμογής. Αυτό ήταν αποτέλεσμα των σίγουρων, σταθερών και όχι επιπόλαιων βημάτων από την στιγμή που άρχισα να δουλεύω πάνω στην πτυχιακή. Μην έχοντας άλλη επαφή με την γλώσσα, παρά μόνο από την στιγμή που ξεκίνησε η ενασχόλησή μου με την πτυχιακή, έπρεπε να αφιερώσω αρκετό χρόνο στην σωστή εκμάθηση της γλώσσας Ruby, ώστε να αποφευχθούν συντακτικά λάθη και λάθη που αφορούσαν την δομή της γλώσσας. Έπειτα από τρεις με τέσσερις μήνες διαβάσματος σημειώσεων που αφορούσαν την Ruby και νιώθοντας ότι έχω αποκτήσει δυνατές βάσεις, ξεκίνησε η ανάπτυξη της εφαρμογής, και έπειτα η σύνταξη του θεωρητικού μέρους της πτυχιακής. Μέτα την περάτωση της πτυχιακής και έχοντας αποκτήσει ικανοποιητική πείρα μπορώ να πω, πως η Ruby και το Web framework Ruby On Rails είναι δύο πολύ δυνατά εργαλεία για τον οποιονδήποτε θέλει να ασχοληθεί με το Web programming. Μεγάλο μέρος της δύναμης αυτής αντλείται από την μεγάλη κοινότητα των προγραμματιστών Ruby, που έχουν αναπτύξει έναν τεράστιο αριθμό από gems (βιβλιοθήκες) και της open source φιλοσοφίας που έχει ενστερνιστεί η γλώσσα και κατά συνέπεια και η κοινότητα των προγραμματιστών που την χρησιμοποιεί. Το μεγαλύτερο όμως μέρος της δύναμης της Ruby (και κατά συνέπεια και του RoR) προέρχεται από τον ίδιο της τον εαυτό, μιας και είναι μια πολύ ευέλικτη high level γλώσσα, που δίνει την δυνατότητα στον προγραμματιστή να γράψει κώδικα όχι με βάση την γλώσσα αλλά με βάση τον τρόπο που ο ίδιος θα επιλέξει. Το framework ruby on rails αν και στην αρχή κάποιον ίσως να μην τον ενθουσιάσει ιδιαίτερα, όταν όμως εξοικειωθεί μαζί του θα διαπιστώσει ότι μπορεί να αναπτύξει πολύ “δυνατές” εφαρμογές με πολύ λίγο κόπο.

Βιβλιογραφία

1)Cooper, P. (2009), *Beginning Ruby: From Novice to Professional*, Second Edition, Apress, USA

2)Flanagan, D., Matsumoto, Y., (2008), *The Ruby Programming Language*, O'Reilly, USA

3)Hartl, M., (2014), *Ruby on Rails Tutorial*, Retrieved from https://www.railstutorial.org/book

4)Ruby, S., Thomas, D., Hannson, D.H., (2010), *Agile Web Development with Rails*, Fourth Edition, The Pragmatic Bookself, USA

5) http://tryruby.org/ , Διαδραστική σελίδα εκμάθησης της Ruby.

6) http://ruby-doc.com/docs/ProgrammingRuby/ , Οδηγός εκμάθησης της Ruby.

7) https://www.codeschool.com/ , Ηλεκτρονικό σχολείο εκμάθησης προγραμματισμού.

8) http://www.lynda.com/ , Online video tutorial.

9) http://guides.rubyonrails.org/ , Ο επίσημος οδηγός για την Ruby on Rails.

10) http://el.wikipedia.org/wiki/Ruby_on_Rails , Η Ruby on Rails στην ηλεκτρονική εγκυκλοπαίδεια.

11) http://agilemanifesto.org/ , Μανιφέστο για την ευέλικτη ανάπτυξη λογισμικού.