Πανεπιστήμιο Πελοποννήσου
Association for Computing Machinery
Φοιτητικό Παράρτημα ACM Παν. Πελοποννήσου

Web development that doesn't hurt!

RUBY ON RAILS:
ΜΙΑ ΠΕΡΙΕΚΤΙΚΗ ΕΙΣΑΓΩΓΗ

Έκδοση 1.0

12 Ιαν 2007

Νίκος Δημητρακόπουλος
nikosd@uop.gr

Rails???

- Μία πλατφόρμα (framework) ανάπτυξης web εφαρμογών βασισμένο στην Ruby
- Υποστηρίζει πλήρως και εγγενώς όλες τις νέες τεχνολογίες και τάσεις:
  - MVC αρχιτεκτονική
  - ORM (αντικειμενοσχεσιακή αντιστοίχηση)
  - AJAX λειτουργικότητα
  - Web Services
  - Templating για το layout των σελίδων χρησιμοποιώντας εμφωλευμένο Ruby κώδικα σε html, CSS, Javascript κ.λ.π.
- *Σε όσους αρέσουν τα buzzwords* -> Είναι web2.0

Outline

Μέρος I - Θεωρία

- Τι είναι ένα framework
- Ιστορία του Rails
- Φιλοσοφία του Rails framework
- Βασικά Συστατικά
- Άλλα εργαλεία

Μέρος II - Υλοποίηση

- Τεχνικά θέματα
- Δημιουργία μίας βασικής εφαρμογής
- Επεκτείνοντας την εφαρμογή

Μέρος III – Συζήτηση…

- Σημειώσεις / επισημάνσεις
- Πλεονεκτήματα / μειονεκτήματα
- Βιβλιογραφία / Σύνδεσμοι

Μέρος Ι

Θεωρία

- Τι είναι ένα framework
- Ιστορία του Rails
- Φιλοσοφία του Rails framework
  - DRY
  - Convention over configuration
  - MVC αρχιτεκτονική
  - Agile
- Βασικά Συστατικά
  - Active Record
  - Action Pack
  - Prototype
  - Action Mailer
  - Action Web Service
- Άλλα εργαλεία
  - Scripts για παραγωγή κώδικα
  - Migrations

Τι είναι ένα framework

- Γενικά, ένα framework είναι:
  - μία υποστηρικτική δομή ή αλλιώς ένας «σκελετός»
  - χρησιμοποιείται σαν βάση για κάτι το οποίο κατασκευάζεται
- Μπορεί να εμπεριέχει:
  - υποστηρικτικά προγράμματα
  - βιβλιοθήκες
  - κάποια scripting γλώσσα προγραμματισμού
  - άλλο λογισμικό

Αυτό διαφοροποιεί και μία απλή βιβλιοθήκη από ένα framework!

- Ένα framework είναι στοχευμένο σε συγκεκριμένη/ες τεχνολογία/ες
- Έχει μία συγκεκριμένη «ροή» που πρέπει να ακολουθηθεί από τον προγραμματιστή (π.χ. την MVC, όπως θα δούμε στο Rails)

- Χρησιμοποιώντας ένα framework για την ανάπτυξη μίας εφαρμογής επιτυγχάνεται:
  - Μεγάλη επαναχρησιμοποίηση κώδικα
  - Λιγότερος απαιτούμενος χρόνος που πρέπει να χρησιμοποιηθεί για λήψη αποφάσεων που έχουν να κάνουν με λεπτομέρειες «χαμηλού επιπέδου»
  - Περισσότερος χρόνος έτσι, μπορεί να χρησιμοποιηθεί για να επιτευχθούν οι απαιτήσεις της εφαρμογής

Ιστορία του Rails

- Το Rails εξήχθη από τον David Heinemeier Hansson από την δουλειά του στο basecamp.com (ένα εργαλείο για διαχείριση / οργάνωση έργων (project management) της 37signals)
- Δημοσιεύτηκε στον web για πρώτη φορά τον Ιούλιο του 2004.
- Η έκδοση 1.0 κυκλοφόρησε στις 13 Δεκεμβρίου του 2005
- Η έκδοση 1.1 κυκλοφόρησε στις 28 Μαρτίου του 2006
- Τον Αύγουστο του 2006 ανακοινώθηκε από την Apple ότι το Mac OS X v10.5 (“Leopard”), που αναμένεται να κυκλοφορήσει το 2007, θα εμπεριέχει το Rails framework [2]
- Μέσα στις αρχές του 2007 αναμένεται η έκδοση 1.2

Φιλοσοφία του Rails (1)

- Αξίωμα 1ο :

  Μην επαναλαμβάνεις τον εαυτό σου!
  (DRY – Don’t Repeat Yourself)

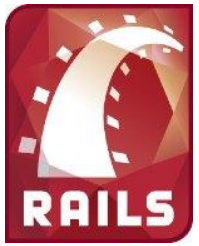
- Αξίωμα 2ο :

  Προτυποποίηση αντί διαμόρφωσης!
  (convention over configuration)

Φιλοσοφία του Rails (2) : DRY

- Τι σημαίνει πρακτικά το 1ο αξίωμα;
  - Ότι η πληροφορία δεν θα πρέπει να υπάρχει δύο (ή και περισσότερες) φορές
- Αυτό γιατί διπλότυπη πληροφορία σημαίνει:
  - Δυσκολία στις αλλαγές
  - Μειωμένη διαφάνεια
  - Αυξημένη πιθανότητα για «ασυνέπειες» στον κώδικα
- Η αρχή αυτή γενικεύεται και στα εξής:
  - Στα σχήματα της βάσης δεδομένων (database schema)
  - Στα πλάνα δοκιμών (test plans)
  - Ακόμα και στην τεκμηρίωση (documentation)
- Χρησιμοποιώντας την τεχνική αυτή επιτυχώς, σημαίνει ότι η αλλαγή ενός συγκεκριμένου στοιχείου δεν επηρεάζει τα υπόλοιπα, λογικά ασύνδετα στοιχεία του συστήματος
- Επιπλέον, στοιχεία που συνδέονται λογικά, αλλάζουν ομοιογενώς και απόλυτα προβλέψιμα και έτσι είναι συγχρονισμένα

Φιλοσοφία του Rails (3) : Convention over configuration



- Τι σημαίνει το 2ο αξίωμα;
  - Ο προγραμματιστής χρειάζεται να ορίσει μόνο τις παραμέτρους που δεν είναι δυνατόν να «προτυποποιηθούν»
  - Π.χ.: Μία κλάση **Post** μπορεί αυτομάτως να συσχετιστεί με έναν πίνακα **posts**
- Έτσι, οι «συμβάσεις» που χρησιμοποιούνται από το Rails μπορούν να μειώσουν αισθητά τον κώδικα που χρειάζεται να γραφεί
- Φυσικά, οι συμβάσεις αυτές μπορούν να αλλάξουν όταν αυτό είναι απαραίτητο / επιθυμητό

Φιλοσοφία του Rails (4) : Agile

- Agile σημαίνει ευκίνητος, σβέλτος
- Η μεθοδολογία αυτή είναι μία οικογένεια μεθόδων για την διαδικασία της ανάπτυξης προϊόντων (γενικά) που έχει προταθεί, έχοντας όμως κατά νου, κυρίως το λογισμικό
- Ουσιαστικά μιλάμε για την δημιουργία λογισμικού με έναν πιο «ελαφρύ», «γρήγορο» και ανθρωποκεντρικό τρόπο

Φιλοσοφία του Rails (5) : Agile Manifesto

- Η ευχαρίστηση του πελάτη με άμεση και συνεχή παράδοση χρήσιμου λογισμικού
- Το λειτουργικό λογισμικό παραδίδεται συχνά (εβδομάδες αντί για μήνες)
- Το λειτουργικό λογισμικό είναι το πρωταρχικό κριτήριο για την μέτρηση της προόδου
- Ακόμα και αλλαγές που ανακύπτουν αργότερα είναι ευπρόσδεκτες
- Στενή, καθημερινή συνεργασία μεταξύ επιχειρηματιών και προγραμματιστών
- Συζήτηση πρόσωπο-με-πρόσωπο είναι η καλύτερη μορφή επικοινωνίας
- Συνεχής προσοχή στην τεχνική αρτιότητα και τον καλό σχεδιασμό
- Απλότητα
- Αυτό-διαχειριζόμενες ομάδες
- Συχνή προσαρμογή στις αλλαγές περιστάσεων

Φιλοσοφία του Rails (6) : MVC Αρχιτεκτονική

- Το Rails χρησιμοποιεί την MVC αρχιτεκτονική (Model-View-Controller):
  - **Model**: Τα δεδομένα μας ή αλλιώς η επιχειρησιακή λογική της εφαρμογής μας
  - **View**: Το επίπεδο παρουσίασης, δηλαδή το πώς εμφανίζονται τα αποτελέσματα / δεδομένα
  - **Controller:** Η διεπαφή με τον χρήστη και ουσιαστικά η λειτουργικότητα της εφαρμογής

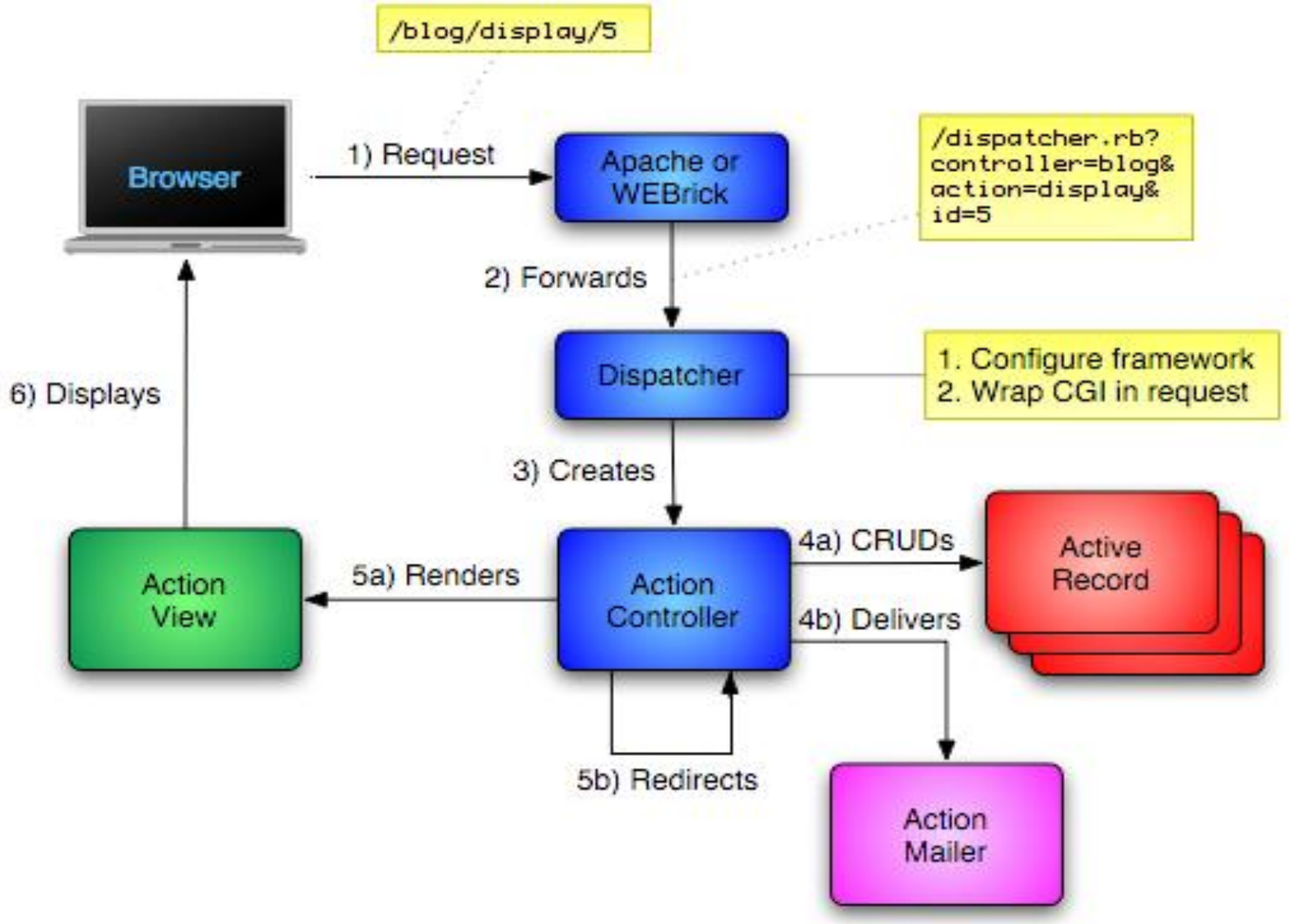
Δομικά συστατικά του Rails (1)

- 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.
  - Π.χ.: table: “people” <-> class Person
- 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.)

Δομικά συστατικά του Rails (2)

- *Prototype*
  - Το κομμάτι που υλοποιεί την AJAX λειτουργικότητα του site, όπως drag-and-drop, 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

Δομικά συστατικά του Rails + MVC



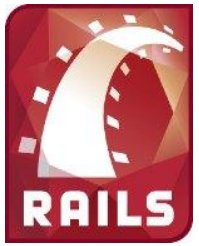
Άλλα *«εργαλεία»* (1) : Scripts / Generators

- Στο Rails περιλαμβάνονται κάποια αρχεία για την παραγωγή έτοιμου κώδικα
- Π.χ.:
  - Για την δημιουργία ενός controller, ενός model, κ.λ.π.
  - Για την δημιουργία έτοιμων *«ειδών» controllers, όπως π.χ. του scaffold*
  - Για την δημιουργία ενός migration *(που θα δούμε στην συνέχεια τι είναι)*
- Έτσι γλιτώνουμε από κάποιες τυπικές / βαρετές διαδικασίες ->
- Συγκεντρωνόμαστε στο πραγματικό ζητούμενο, δλδ *την δημιουργία της εφαρμογής μας*!

Άλλα «*εργαλεία*» (2) :
Τρία ταυτόχρονα περιβάλλοντα εργασίας!

- Δεν θα ήταν ωραία να μπορούσαμε να βλέπουμε σε πραγματικό χρόνο τις αλλαγές που κάνουμε στην εφαρμογή;
- Αυτό θα είχε πολλά όμως προβλήματα:
  - Προβλήματα απόδοσης, αφού κάθε φορά θα έπρεπε να επαναφορτώνεται όλη η εφαρμογή (και στον browser του χρήστη αλλά και στον server)
  - Και κυρίως, δεν θα θέλαμε ο χρήστης να βλέπει το ίδιο φυσικά όσο εμείς «παίζουμε» με την εφαρμογή!
- Επίσης δεν θα ήταν βολικό να μπορούσαμε να κάνουμε κάποιες δοκιμές χωρίς να επηρεάζεται η κανονική εφαρμογή;

Άλλα «εργαλεία» (3) :
Τρία ταυτόχρονα περιβάλλοντα εργασίας!



- Φυσικά και γίνεται!
- Τρία (ταυτόχρονα) περιβάλλοντα εργασίας:
  - Development
  - Production
  - Testing
- Το καθένα χρησιμοποιεί την δικιά του έκδοση κώδικα και την δικιά του βάση!
- Έτσι:
  - 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!

Άλλα «*εργαλεία*» (4) : Migrations

- Ποτέ μία εφαρμογή σε πραγματικές συνθήκες δεν παραμένει σταθερή
- Έτσι, πρέπει συνεχώς να εξελίσσεται η βάση δεδομένων (αφού αντικατοπτρίζει την επιχειρησιακή λογική)
- Πώς γίνεται να μην «χάνουμε την μπάλα» και να μην μας φοβίζει μια ενδεχόμενη αλλαγή;
- Πώς μπορούμε *με ασφάλεια* να επιστρέψουμε στην προηγούμενη κατάσταση που βρισκόταν η βάση εάν κάτι δεν πάει καλά;
- Η απάντηση είναι απλή: **Migrations**!

Άλλα «εργαλεία» (5) : 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 τέτοιων plugins:
  - ‘acts_as_taggable’ mixin για αντικείμενα της κλάσης ActiveRecord (εισαγωγή tags σε ένα μοντέλο)
  - ‘file_column’ για την κλάση ActiveRecord (κάνει το ανέβασμα αρχείων και την αλλαγή του μεγέθους μίας εικόνας εύκολο)
  - ‘globalize’ (προσθέτει πολυγλωσσική και διεθνοποιημένη (i18n) υποστήριξη στο Rails)

Μέρος ΙΙ

Υλοποίηση

- Τεχνικά θέματα:
  - Εγκατάσταση του Rails
  - Υποστηριζόμενοι web servers
  - Υποστηριζόμενες βάσεις δεδομένων
- Δημιουργία μίας βασικής εφαρμογής:
  - Δημιουργία μίας εφαρμογής στο Rails
  - Δομή καταλόγων
  - Εκκίνηση του ενσωματωμένου web server
  - Δημιουργία της βάσης
  - Βασικές ρυθμίσεις της εφαρμογής
  - Σχεδιασμός της εφαρμογής
  - Τα μοντέλα
  - Οι controllers
  - Η εμφάνιση
- Επεκτείνοντας μία εφαρμογή:
  - CRUD λειτουργίες στα μοντέλα
  - Validations
  - Σχέσεις μεταξύ μοντέλων

Τεχνικά θέματα (1) : Εγκατάσταση του Rails

- Έχοντας εγκαταστήσει την Ruby και το “Gems” εκτελούμε απλά:
  - ***gem install rails --include-dependencies***
  - ή ***sudo gem install rails --include-dependencies*** (για *nix περιβάλλοντα)
- Άλλοι τρόποι:
  - Με κάποιον package-manager του λειτουργικού (π.χ. το apt για debian-based συστήματα)
  - Με έτοιμα πακέτα που μπορούμε να βρούμε στο internet, και περιλαμβάνουν τα πάντα (web server, βάση δεδομένων, Ruby, Rails, άλλες βιβλιοθήκες και προγράμματα, κ.λ.π.), όπως:
    - Locomotive (για Mac OS X)
    - Instant Rails (για Windows)

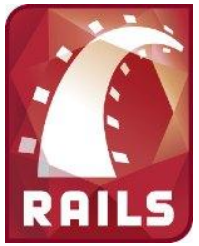
Τεχνικά θέματα (2) : Υποστηριζόμενοι web servers

- Apache
- Lighttpd

- **Σημαντικό!** Ο web server πρέπει να τρέχει επίσης:
  - FastCGI
  - SCGI
  - Mongrel

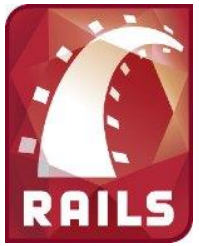
- Επίσης, μπορεί να χρησιμοποιηθεί και ο ενσωματωμένος server (WEBrick), αλλά συστήνεται μόνο για το development

Τεχνικά θέματα (3) : Υποστηριζόμενες Βάσεις Δεδομένων



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

Δημιουργία μίας βασικής εφαρμογής (1) : Δημιουργία μίας εφαρμογής στο Rails



- Ας δημιουργήσουμε ένα πολύ απλό blog…
- Το rails θα φτιάξει για εμάς όλο τον σκελετό της εφαρμογής (καταλόγους, scripts που θα χρησιμοποιήσουμε, κ.λ.π.)
- C:\"κάποιο path"\>***rails simple-blog***

  create app/controllers

  create app/helpers

  create app/models

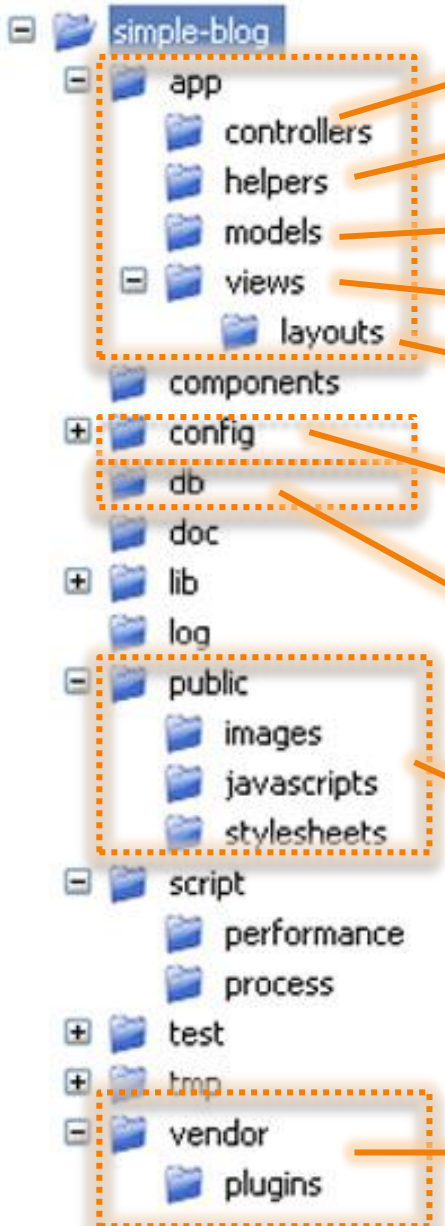
  create app/views/layouts

  create config/environments

  create components

  …

Δημιουργία μίας βασικής εφαρμογής (2) : Δομή καταλόγων



Ο κώδικας για τους controllers

Αρχεία με βοηθητικό κώδικα που χρησιμοποιούνται από τα views

Ο κώδικας για τα μοντέλα

Ο κώδικας για τα views (κάθε view αρχείο αντιστοιχεί σε μία μέθοδο ενός controller)

Αντίστοιχα με τα views μόνο που εδώ είναι για κάθε controller

Οι ρυθμίσεις της εφαρμογής

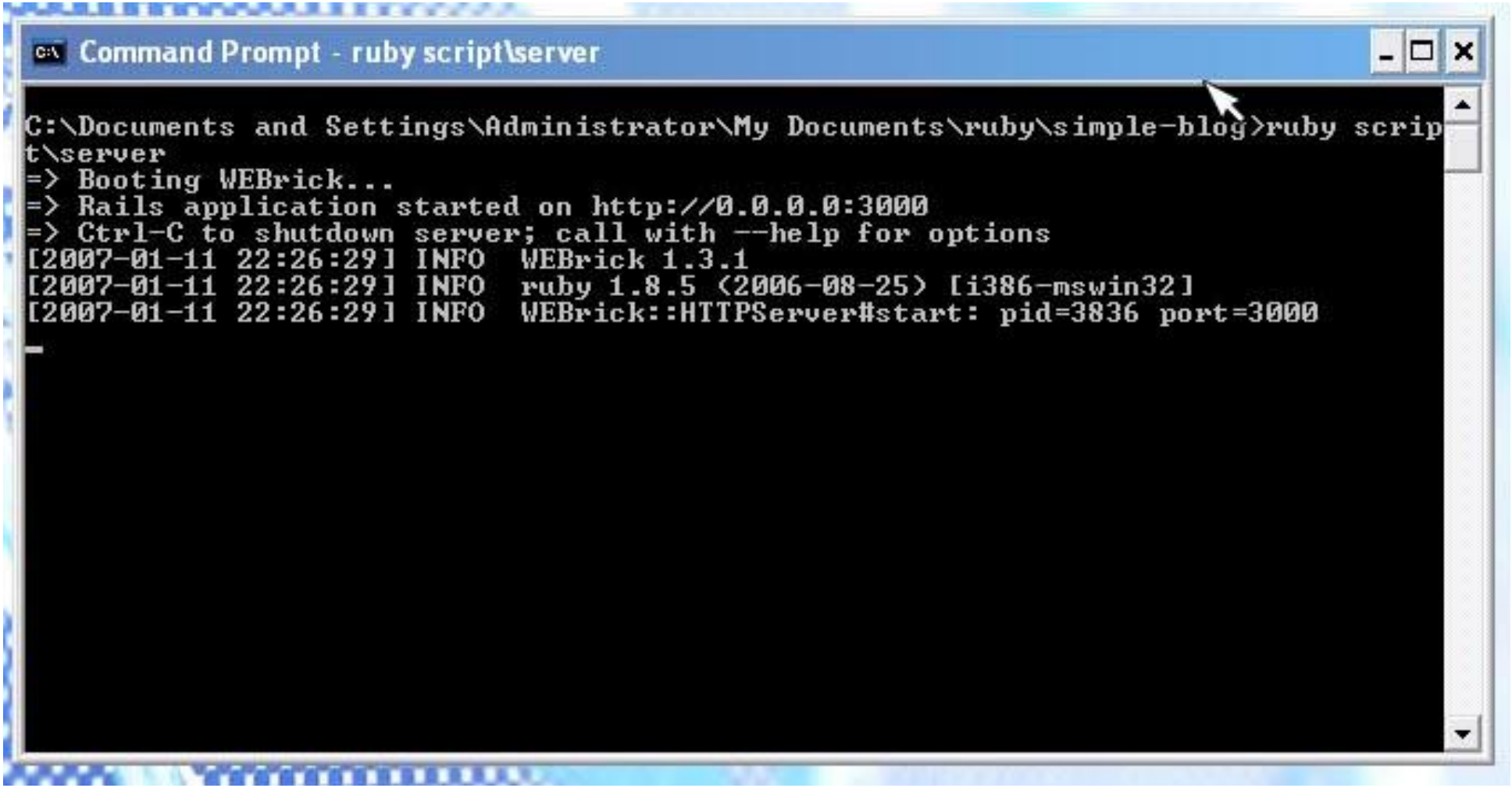
Οτιδήποτε σχετικό με την βάση (κυρίως τα αρχεία για τα migrations αλλά και sql αρχεία)

Το στατικό περιεχόμενο της σελίδας (όπως τα style sheets, εικόνες, κ.λ.π.)

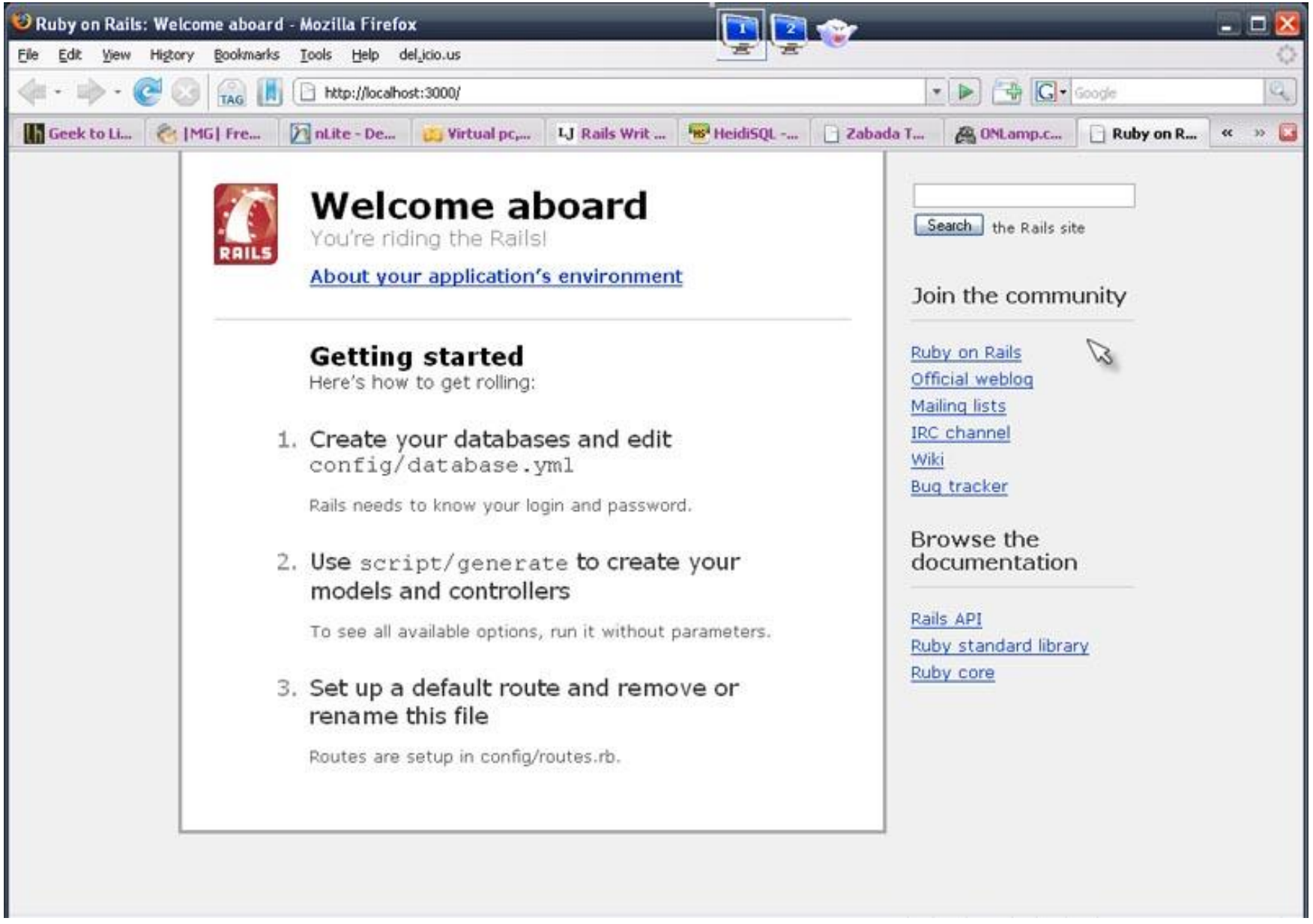
Εξωτερικά plugins (είτε δικά μας, είτε άλλων)

Δημιουργία μίας βασικής εφαρμογής (3) : Εκκίνηση του WEBrick server

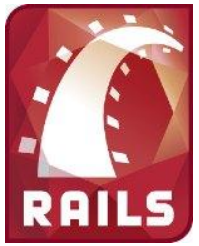
- C:\...\simple-blog> ***ruby script\server***



Δημιουργία μίας βασικής εφαρμογής (4) : Εκκίνηση του WEBrick server



Δημιουργία μίας βασικής εφαρμογής (5) : Δημιουργία της βάσης



- Χρησιμοποιώντας τα εργαλεία της mysql δημιουργούμε την βάση που θα χρησιμοποιήσει η εφαρμογή:
- ***C:\mysqladmin -u root -p create simple-blog_development***
  Enter password: ***

  Σημείωση 1: όπως αναφέρθηκε μπορεί να χρησιμοποιηθεί οποιαδήποτε από τις υποστηριζόμενες βάσεις. Εμείς στο παράδειγμα θα χρησιμοποιήσουμε την mysql μιας και είναι η πιο διαδεδομένη

  Σημείωση 2: όπως επίσης είπαμε, το Rails έχει 3 ταυτόχρονα περιβάλλοντα. Εμείς εδώ θα επικεντρωθούμε μόνο στο development κομμάτι. Κατά σύμβαση τα ονόματα για τις βάσεις είναι:
  - xxx_development
  - xxx_production
  - xxx_testing

Δημιουργία μίας βασικής εφαρμογής (6) : Βασικές ρυθμίσεις της εφαρμογής

- Πρέπει να δηλώσουμε κάποιες παραμέτρους για την βάση στο Rails
- Για τον σκοπό αυτό επεξεργαζόμαστε το αρχείο ***config/database.yml***
- Με το μόνο που θα ασχοληθούμε είναι με το κομμάτι *development.*
- Το μόνο που χρειάζεται να αλλάζουμε προς το παρόν είναι το κομμάτι με τον κωδικό

```
…
development:
 adapter: mysql
 database: simple-blog_development
 username: root
 password:
 host: localhost
…
```

```
…
development:
 adapter: mysql
 database: simple-blog_development
 username: root
 password: pass
 host: localhost
…
```

- Για να επιβεβαιώσουμε ότι όλα πήγαν καλά πληκτρολογούμε ***rake db:migrate***
- Το αποτέλεσμα θα πρέπει να είναι κάτι του στυλ: ***(in C:/…/simple-blog)***

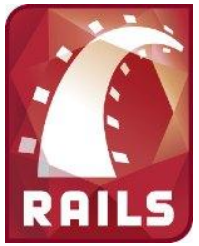
Σημείωση: η εντολή ***rake*** κάνει πολύ παραπάνω πράγματα, στα οποία δεν θα αναφερθούμε (τουλάχιστον όχι ακόμα)

Δημιουργία μίας βασικής εφαρμογής (7) : Σχεδιασμός της εφαρμογής

- Θέλουμε να μπορούμε:
  - Να δημιουργούμε νέα posts
  - Να βλέπουμε τα ήδη υπάρχοντα
  - Να επεξεργαζόμαστε τα ήδη υπάρχοντα
  - Να διαγράφουμε κάποιο post
  - Κοινώς, θέλουμε λειτουργικότητα CRUD (Create, Read, Update, Delete)

- Το κάθε Post για αρχή θέλουμε να έχει:
  - Τίτλο
  - Κείμενο
  - Ημερομηνία δημιουργίας

Δημιουργία μίας βασικής εφαρμογής (8) : Τα μοντέλα – Δημιουργία ενός μοντέλου



- Πρώτα θα φτιάξουμε τα μοντέλα της εφαρμογής (δηλαδή της οντότητες που την αποτελούν)
- Λέμε στο rails να το φτιάξει για μας:

  C:\...\> ***ruby script\generate model post***
- Βλέπουμε ότι αυτό δημιουργεί διάφορα πράγματα:

  create app/models/post.rb

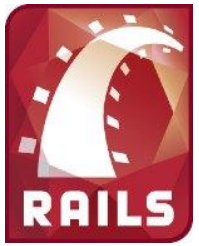
  create test/unit/post_test.rb

  create test/fixtures/posts.yml

  create db/migrate

  create db/migrate/001_create_posts.rb
- Από αυτά μας ενδιαφέρουν μόνο τα δύο προς το παρόν

Δημιουργία μίας βασικής εφαρμογής (9) : Τα μοντέλα – Δημιουργώντας τον πίνακα



- Για την δημιουργία του πίνακα υπάρχουν δύο τρόποι:
  - SQL (είτε μέσα από command line, είτε με κάποιο εργαλείο όπως phpMyAdmin κ.λ.π.)
  - Μέσα από το rails:
    - Γράφουμε σε ruby
    - Το rails απλοποιεί τα πράγματα για μας, χρησιμοποιώντας αυτόματα τις συμβάσεις του
      - Γράφουμε έξυπνα, γρήγορα
      - Γράφουμε πολύ λιγότερο επαναλαμβανόμενο (άρα και άχρηστο) κώδικα ⟶ **DRY**
- Ας πάρουμε μία πρώτη γεύση από τα migrations…

Δημιουργία μίας βασικής εφαρμογής (10) : Τα μοντέλα – Δημιουργώντας τον πίνακα

- Το rails δημιούργησε για μας το ***db\migrate\001_create_posts.rb***

Αύξον αριθμός

Περιγραφή

- Ας δούμε τι περιέχει…

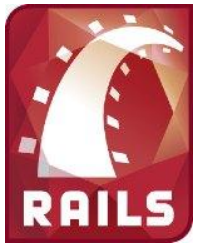
```
class CreatePosts < ActiveRecord::Migration
  def self.up
    create_table :posts do |t|
      # t.column :name, :string
    end
  end

  def self.down
    drop_table :posts
  end
end
```

Η μέθοδος αυτή θα εκτελεστεί όταν θα κάνουμε την αλλαγή στην βάση

Αντίστοιχα, αυτή θα εκτελεστεί όταν θα θελήσουμε για κάποιο λόγο να επιστρέψουμε στην προηγούμενη κατάσταση της βάσης!

Δημιουργία μίας βασικής εφαρμογής (11) : Τα μοντέλα – Δημιουργώντας τον πίνακα




```
class CreatePosts < ActiveRecord::Migration
  def self.up
    create_table :posts do |t|
      t.column :title,            :string
      t.column :description,      :text
      t.column :creation_date,    :datetime
    end
  end

  def self.down
    drop_table :posts
  end
end
```

- Δημιουργούμε έναν πίνακα με όνομα posts
- Δημιουργούμε τις εξής στήλες:
  - title (string)
  - description (text)
  - creation_date (ημερομηνία και ώρα)
- Διαγράφουμε τον πίνακα posts από την βάση

- Εκτελούμε και πάλι την εντολή ***rake db:migrate*** και πρέπει να μας δώσει σαν αποτέλεσμα:

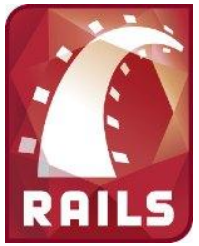
== CreatePosts: migrating==

-- create_table(:posts)

-> 0.7820s

== CreatePosts: migrated (0.7820s) ================================

Δημιουργία μίας βασικής εφαρμογής (12) : Τα μοντέλα – Δημιουργώντας τον πίνακα



- :string, :text
- :integer, :float
- :datetime, :timestamp, :time, :date
- :binary, :boolean

SQL Type	Ruby Class	SQL Type	Ruby Class
int, integer	Fixnum	float, double	Float
decimal, numeric	BigDecimal[1]	char, varchar, string	String
interval, date	Date	datetime, time	Time
clob, blob, text	String	boolean	see text

Δημιουργία μίας βασικής εφαρμογής (13) : Τα μοντέλα

- Το άλλο αρχείο που μας ενδιαφέρει είναι το ***app/models/post.rb***
- Το μόνο που περιέχει αρχικά:

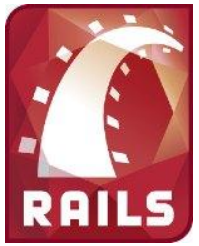
```
class Post < ActiveRecord::Base
end
```

- Εφόσον δεν θέλουμε να εισάγουμε κάποια επιπλέον λειτουργία μας αρκούν οι δύο αυτές γραμμές (που δεν τις γράψαμε καν εμείς ☺)
- Έχει αυτομάτως αντιστοιχηθεί με την βάση και άρα περιέχει όλα τα δεδομένα του πίνακα *posts!*

Δημιουργία μίας βασικής εφαρμογής (14) : Τα μοντέλα – ΠΡΟΣΟΧΗ!!!

- Το rails χρησιμοποιεί κάποιες σημαντικές συμβάσεις (εκτός και αν ορίσουμε εμείς διαφορετικά):
  - Το όνομα του μοντέλου είναι στον ενικό (post) ενώ του πίνακα στον πληθυντικό (posts)
  - Θεωρεί ότι ο πίνακας έχει ως πρωτεύον κλειδί την στήλη με όνομα id
- Χρησιμοποιώντας τα migrations, αυτά γίνονται αυτόματα
- Σε SQL όμως, χρειάζεται να τα καθορίσουμε εμείς! Π.χ. πρέπει να δηλώσουμε μία στήλη
  - με όνομα id
  - που να είναι πρωτεύον κλειδί
  - και να αυξάνεται αυτόματα κατά 1
- Και διάφορες άλλες… (μερικές θα αναφερθούν)

Δημιουργία μίας βασικής εφαρμογής (15) : Controller / View- Scaffolding



- Ας κάνουμε μερικά «μαγικά»…
- ***ruby script\generate scaffold post posts***
- Και ας ρίξουμε μια ματιά στον browser μας (http://localhost:3000/posts)

Δημιουργία μίας βασικής εφαρμογής (16) : Controller / View- Scaffolding

- Πριν δούμε τον κώδικα:
  - Ο controller είναι υπεύθυνος και για το “routing”
  - Δηλαδή για τον χειρισμό των διευθύνσεων στον browser
  - Το Rails παρέχει το μηχανισμό για «ανθρώπινα» urls και όχι κάτι που καταλαβαίνει μόνο η εφαρμογή
  - Το καλύτερο είναι, ότι αυτό συμβαίνει αυτόματα, και χωρίς προσπάθεια από τον προγραμματιστή ☺

http://localhost:3000/posts/show/1

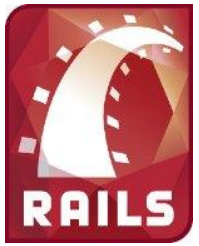
Η διεύθυνση του server

Ο controller

Η μέθοδος που θα εκτελεστεί

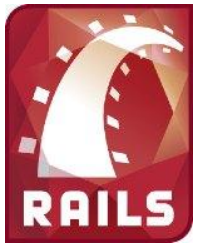
Οι παράμετροι που δίνονται ως ορίσματα στην μέθοδο. Εδώ είναι το ***id***

Δημιουργία μίας βασικής εφαρμογής (17) : Controller / View- Scaffolding



- Τι αρχεία παρήγαγε η εντολή αυτή;
- ***app/controllers/posts_controller.rb :***
  - Ο controller που ζητήσαμε
  - Περιέχει τις μεθόδους *index, list, show, new, edit* τις *create, update* που υλοποιούν την λειτουργικότητα (και χρησιμοποιούνται από) των new, edit και τέλος την destroy
- ***app/views/posts/list.rhtml***
  app/views/posts/show.rhtml
  app/views/posts/new.rhtml
  app/views/posts/edit.rhtml
  - Είναι το view κομμάτι του προγράμματος
  - Για κάθε μέθοδο του controller χρειάζεται να ορίσουμε και ένα αρχείο .rhtml
  - Είναι υπεύθυνο για το «δέσιμο» των δεδομένων που επιστρέφονται από τον controller
  - Και τελικά την παραγωγή html (κώδικα) η οτιδήποτε άλλο θέλουμε, όπως xml)

Δημιουργία μίας βασικής εφαρμογής (18) : Controller / View- Scaffolding



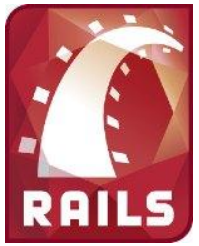
(συνέχεια)

- ***app/views/layouts/posts.rhtml***
  - Ορίζει ένα γενικό layout, για όλες λειτουργίες του controller
  - Εδώ ορίζουμε ουσιαστικά το τα κομμάτια που είναι κοινά για όλα τα views του controller
    - Πλοήγηση
    - Banners
    - Footers
    - Style sheet που θα χρησιμοποιηθεί
    - κ.λ.π.
- ***public/stylesheets/scaffold.css***
  - Εδώ (στον φάκελο stylesheets) τοποθετούνται τα διάφορα css αρχεία
  - Το scaffold, δημιουργεί ένα δικό του, αρκετά απλό css

Δημιουργία μίας βασικής εφαρμογής (19) : Controller / View- Scaffolding

- Ο scaffold generator παρέχει μία βασική CRUD λειτουργικότητα
- Έτσι μπορούμε να:
  - Αλληλεπιδράσουμε με το μοντέλο μας από πολύ νωρίς
  - Να αφήσουμε την υλοποίηση του controller / view για αργότερα
  - Και παράλληλα να έχουμε κάτι που να δουλεύει
- Μπορούμε να το αλλάξουμε κατά βούληση (ώστε να το φέρουμε στα μέτρα μας)
- Η να γράψουμε εξ’ ολοκλήρου από την αρχή έναν καινούριο

Δημιουργία μίας βασικής εφαρμογής (20) : Controller / View- Scaffolding

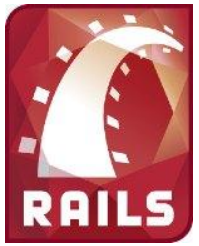


- Συνήθως γίνεται ένας συνδυασμός, δηλαδή:
  - Σιγά, σιγά αλλάζουμε το scaffold
  - Μέχρι που αλλάζουμε τελείως τον κώδικα και άρα έχουμε κάτι καινούριο
- Σπάνια ένας scaffold generator χρησιμοποιείται στην τελική έκδοση μίας εφαρμογής
- Είναι απλά ένα «εργαλείο» για τα πρώτα στάδια της εφαρμογής
- Είναι 100% agile ☺

Επεκτείνοντας μία εφαρμογή (1)

- Το rails παρέχει πολύ περισσότερα πράγματα από αυτά που είδαμε μέχρι τώρα
- Ο καλύτερος τρόπος για να τα δούμε είναι ένα βιβλίο ή on-line μέσα από wiki's κ.λ.π.
- Θα δούμε μερικά ακόμα εδώ, που είναι αρκετά σημαντικά, όπως:
  - Εκτέλεση CRUD λειτουργιών στα μοντέλα
  - Validations
  - Σχέσεις μεταξύ μοντέλων

Επεκτείνοντας μία εφαρμογή (2) : CRUD λειτουργίες στα μοντέλα



- Προφανώς πρέπει να μπορούμε να:
  - Δημιουργούμε νέα στιγμιότυπα
  - Επεξεργαζόμαστε τα ήδη υπάρχοντα
  - Διαγράφουμε στιγμιότυπα
  - Αναζητούμε στιγμιότυπα
- Λόγω της αυτόματης αντικειμενο-σχεσιακής αντιστοίχησης τα περισσότερα είναι πολύ εύκολα, π.χ.

```
a_post = Post.new
a_post.title = “Test”
a_post.description = “Double Test”
a_post.save
```

- Αντίστοιχα εύκολη είναι η αναζήτηση:

```
all_posts = Post.find :all
test_posts = Post.find :all, :conditions => [ “title =?” , “test”]
the_first_post = Post.find :first,  :group => ‘id’
```

Επεκτείνοντας μία εφαρμογή (3) : Validations

- Έστω ότι θέλουμε για κάθε στιγμιότυπο της κλάσης Post να ισχύουν τα εξής:
  - Να έχει υποχρεωτικά και τίτλο και περιγραφή
  - Ο τίτλος να είναι αυθεντικός (δλδ να μην υπάρχει άλλο post με τον ίδιο)
- Το μόνο που χρειάζεται να κάνουμε είναι να προσθέσουμε στην κλάση Post τα εξής:
  - `validates_presence_of :title, :description`
  - `validates_uniqueness_of :title`

Επεκτείνοντας μία εφαρμογή (4) : Validations

- Για κάθε validation μπορεί να οριστεί το πότε θα γίνει ο έλεγχος, π.χ.:
  - Κατά την δημιουργία
  - Κατά την αποθήκευση στην βάση
  - Κατά την ενημέρωση
  - Κ.λ.π.
- Υπάρχουν πολλά ακόμα έτοιμα validations:
  - `validates_acceptance_of`
  - `validates_associeted`
  - `validates_format_of`
  - `validates_numericallity_of`
  - `validates_length_of`
  - κ.λ.π.
- Μπορούμε να ορίσουμε δικά μας validations:
  - δημιουργώντας μία μέθοδο `validate` μέσα στην κλάση του μοντέλου
  - Δημιουργώντας τις μεθόδους:
    - `before_save`
    - `before_create, after_create`
    - `before_update`
    - Κ.λ.π.

Επεκτείνοντας μία εφαρμογή (5) : Σχέσεις μοντέλων

- Σπάνια η εφαρμογή θα περιέχει ένα ή περισσότερα ασύνδετα μεταξύ τους μοντέλα
- Γι' αυτό χρησιμοποιούνται και σχεσιακές βάσεις δεδομένων
- Το rails δεν θα μπορούσε να μην υποστηρίζει άμεσα (και εύκολα) κάτι τέτοιο…
- Έτσι μερικές (βασικές) σχέσεις που μπορούμε να έχουμε είναι:
  - 1 προς 1
  - 1 προς N
  - N προς N
- Δηλώνονται μέσα στο μοντέλο ως εξής:
  - `has_one :`<κάποιο μοντέλο> - `belongs_to :`<το άλλο μοντέλο>
  - `has_many :`<κάποιο μοντέλο> - `belongs_to :`<το άλλο μοντέλο>
  - `has_and_belongs_to_many :`<κάποιο μοντέλο> *(και στα 2 μοντέλα)*

Επεκτείνοντας μία εφαρμογή (6) : Σχέσεις μοντέλων (βασικές σχέσεις)

Student **has_one** :car

Car **belongs_to** :student

students
id

cars
id
student_id

Student **belongs_to** :dorm

Dorm **has_many** :students

dorms
id

students
id
dorm_id

Επεκτείνοντας μία εφαρμογή (7) : Σχέσεις μοντέλων (βασικές σχέσεις)

Club `has_and_belongs_to_many :students`

clubs
id

clubs_students
club_id
student_id

Student `has_and_belongs_to_many :clubs`

students
id

Επεκτείνοντας μία εφαρμογή (8) : Σχέσεις μοντέλων (βασικές σχέσεις)


```
Club has_many :students,
:through => :memberships
```

clubs
id

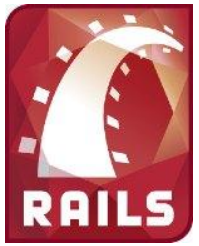
```
Membership belongs_to :student
Membership belongs_to :club
```

memberships
id
club_id
student_id
joined_on

```
Student has_many :clubs,
:through => :memberships
```

students
id

Επεκτείνοντας μία εφαρμογή (9) : Σχέσεις μοντέλων (επιπρόσθετες σχέσεις)



- Κληρονομικότητα (για εκλέπτυνση μίας κλάσης)
- Πολυμορφισμό
- Acts As List (συμπεριφέρεται σαν λίστα)
- Acts As Tree (συμπεριφέρεται σαν δένδρο – χρήσιμο π.χ. σε περίπτωση που θέλουμε να μοντελοποιήσουμε ένα δένδρο με κατηγορίες)

Μέρος ΙΙΙ

Συζήτηση…

- Επισκόπηση της παρουσίασης
- Σημειώσεις / επισημάνσεις
- Πλεονεκτήματα
- Μειονεκτήματα
- Απορίες / Ερωτήσεις
- Βιβλιογραφία
- Σύνδεσμοι

Επισκόπηση

- Αγγίξαμε μόνο την επιφάνεια του Rails
- **Σημαντικό:** Δεν είδαμε καθόλου - ουσιαστικά - το controller / view κομμάτι. Είναι απαραίτητο να το δει όποιος ενδιαφέρεται!
- Όπως αναφέρθηκε στην αρχή έχει πολλά και δυνατά χαρακτηριστικά όπως:
  - Web Services
  - AJAX
  - …
  - Μέχρι και υποστήριξη για Asterisk (μέσα από εξωτερική βιβλιοθήκη)
- Επίσης, υπάρχουν δεκάδες / εκατοντάδες plugins που μπορούν να μας λύσουν τα χέρια
- Αν σας φάνηκε ενδιαφέρουσα η εισαγωγή αυτή που να δείτε και τα υπόλοιπα… ☺

Σημειώσεις / επισημάνσεις

- Το rails δεν κάνει για τα πάντα!
- Ενδείκνυται για συγκεκριμένες εφαρμογές
- Για αυτές που κάνει όμως, σου λύνει τα χέρια
- Αντίστοιχα frameworks υπάρχουν και για Pearl, PHP, κ.λ.π.
- Μπορεί να είναι και καλύτερα
- Όμως το rails είχε την περισσότερη προώθηση / τύχη
  - Μεγαλύτερη κοινότητα χρηστών

    ⟹ Καλύτερη υποστήριξη (υποστηρικτικό υλικό, υποστήριξη από web hosting εταιρίες κ.λ.π.)
  - Περισσότεροι προγραμματιστές

    ⟹ Ταχύτερη ανάπτυξη και εξέλιξη


JAVA IN A NUTSHELL
JBoss 4.0
PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE
XDoclet IN ACTION
Java Web Services
Java Enterprise Best Practices
CORE J2EE PATTERNS
JUnit IN ACTION
Programming Jakarta Struts
Jakarta Struts Cookbook
Java Servlet & JSP Cookbook
Java and XSLT
JAVA FOUNDATION CLASSES IN A NUTSHELL
JavaServer Pages
HIBERNATE IN ACTION
STRUTS IN ACTION
Programming Ruby
Agile Web Development with Rails

Γιατί/πότε να γράψω σε Rails;

- Γιατί μπορούμε να φτιάξουμε μία εφαρμογή γρήγορα, εύκολα και έξυπνα!
- Γιατί θέλουμε να δοκιμάσουμε και να γνωρίσουμε μία νέα προσέγγιση για την σχεδίαση δικτυακών (web) εφαρμογών που υποστηρίζει τις τελευταίες τεχνολογίες
- Όταν θέλουμε να φτιάξουμε μία εφαρμογή που συνδέεται στενά με μία βάση δεδομένων (και να εκμεταλλευτούμε πλήρως το Active Record)
- Όταν έχουμε την δυνατότητα να «φιλοξενήσουμε» την εφαρμογή μας κάπου που ξέρουμε ότι υποστηρίζεται σωστά η Ruby και το Rails

Γιατί/πότε να ΜΗΝ γράψω σε Rails;

- Όταν θέλουμε κάτι απλό, όπως π.χ. μία απλή φόρμα επικοινωνίας! Το γράφουμε σε PHP ☺
- Γιατί το Rails ακόμα έχει κάποια προβλήματα σε αρκετούς web hosting providers, ενώ σε κάποιους δεν υποστηρίζεται καν
- Γιατί δεν ξέρουμε Ruby (να ένας καλός λόγος να μάθουμε! :-D)
- Γιατί ενδεχομένως να μην βρούμε εύκολα κάποιον που να έχει εμπειρία σε αυτό (και έτσι να μην μπορούμε να συνεργαστούμε, να πουλήσουμε μία εφαρμογή σε κάποιον κ.λ.π.)

Μέχρι την επόμενη φορά...

Ερωτήσεις

&

Απαντήσεις

και για τις απορίες μετά την παρουσίαση συνεχίζουμε να σας ακούμε στο: http://ruby-hellug.gr

Βιβλιογραφία - Σύνδεσμοι

- Pragmatic Programmers: Agile software development using Rails [Second Edition]
- www.rubyonrails.com
- Rails cheatsheet (ένας γρήγορος οδηγός αναφοράς για το api του rails)

Άδεια Χρήσης Ελεύθερης Τεκμηρίωσης GNU
GNU Free Documentation Licence

http://www.fsf.org/licensing/licenses/fdl.txt