

# ΜΥΕ042 Τεχνολογίες Διαδικτύου

## Εργαστηριακή Άσκηση 1

Εφαρμογή ιστού για κοινοχρησία φωτογραφιών στο Ruby-on-Rails

### Εισαγωγή

Σκοπός αυτής της εργασίας ήταν να δημιουργήσουμε μια *Web* εφαρμογή χρησιμοποιώντας το framework *Ruby-on-Rails* το οποίο υλοποιεί το *MVC* pattern. Η εφαρμογή επιτρέπει την εγγραφή διάφορων χρηστών που μπορούν να ακολουθούν άλλους χρήστες, το ανέβασμα φωτογραφιών, το σχολιασμό των φωτογραφιών και την προσθήκη tag στις φωτογραφίες.

Βασιζόμενοι στην αρχική έκδοση της εφαρμογής προσθέσαμε επιπλέον λειτουργίες, τροποποιώντας τα ήδη υπάρχοντα στοιχεία είτε προσθέτοντας καινούργια.

### Υλοποίηση

#### 1. Προσθήκη τίτλου στις φωτογραφίες

##### Ζητούμενο:

Κάθε φωτογραφία θα πρέπει να περιέχει ένα τίτλο. Ο χρήστης όταν ανεβάζει μια νέα φωτογραφία προσθέτει και τον τίτλο, διαφορετικά εμφανίζεται μήνυμα ώστε να ξαναπροσπαθήσει. Όταν οι φωτογραφίες εμφανίζονται θα έχουν στο πάνω μέρος και τον τίτλο της.

Αρχικά για την υλοποίηση αυτής της λειτουργίας χρειάζεται να προσθέσουμε στη φόρμα όπου γίνεται η προσθήκη της νέας φωτογραφίας ένα νέο πεδίο “title” τύπου `text_field` ώστε να εισαχθεί και ο τίτλος μαζί με τις άλλες παραμέτρους της φωτογραφίας. Αυτό γίνεται στον ‘προβολέα’ `new.html.haml` και πλέον ο ‘ελεγκτής’ έχει πρόσβαση στις παραμέτρους `params[: ... ]` ώστε να δημιουργήσει μια νέα φωτογραφία.

Στην εμφάνιση των φωτογραφιών του χρήστη για να προβάλλεται και ο τίτλος μπορούμε απλά να πάρουμε το πεδίο `.title` της κάθε φωτογραφίας στο αρχείο `show.html.haml` του προβολέα `user`.

Επίσης στην `private` μέθοδο `photo_params` του ‘ελεγκτή’ προσθέσαμε την παράμετρο `:title` για μεγαλύτερο έλεγχο των παραμέτρων από τον “Action Controller” ώστε να μην μπορούν χρήστες να ενημέρουν πεδία που δεν ορίζονται ρητά.

Τέλος στο μοντέλο `photos` έγινε δημιουργία ενός νέου ‘migration’ - “[...]\_add\_title\_to\_photo.rb” ώστε να αποθηκεύεται και ο τίτλος της φωτογραφίας στη βάση δεδομένων. Χρησιμοποιώντας έτσι το `ActiveRecord` του Rails μπορούμε να αλλάξουμε το μοντέλο με το εργαλείο `rake`, δίνοντας την εντολή [db:migrate](#).

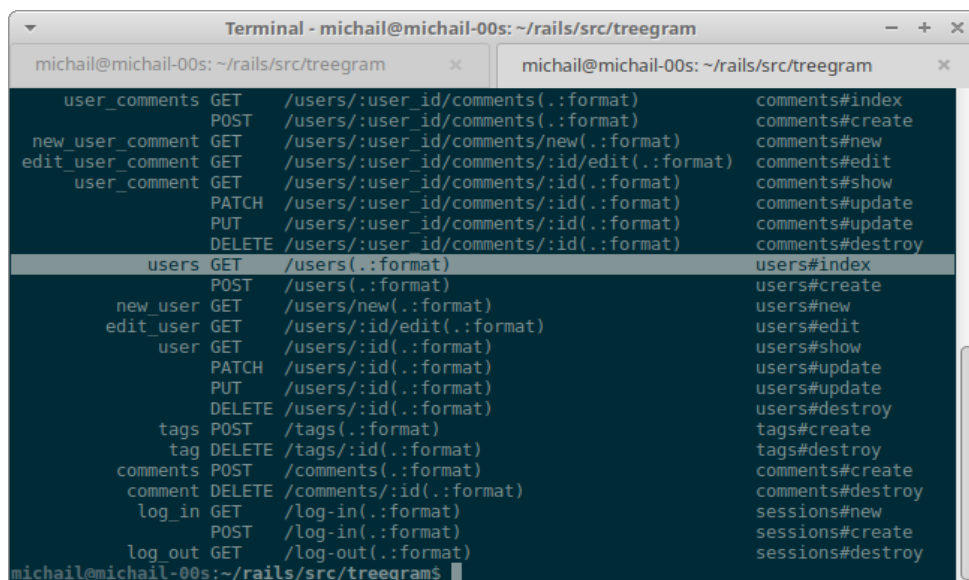
## 2.Λειτουργία “follow”

Ζητούμενο:

Ο χρήστης έχει τη δυνατότητα να ακολουθεί χρήστες και να βλέπει τις φωτογραφίες που αυτοί έχουν ανεβάσει. Οι φωτογραφίες θα πρέπει να είναι ταξινομημένες σε χρονολογική σειρά.

Σε πρώτο στάδιο προστέθηκε ένα νέο κουμπί (“Socialize”) στην αρχική σελίδα του χρήστη με το οποίο εμφανίζονται όλοι οι εγγεγραμμένοι χρήστες στην εφαρμογή. Για την εμφάνιση όλων των χρηστών προσθέτουμε μια νέα μέθοδο index στον user controller η οποία αντιστοιχεί στο path /users. Το path αυτό θα προστεθεί ως link\_to στο κουμπί “Socialize” στον αντίστοιχο προβολέα.

Χρησιμοποιώντας το εργαλείο rake με την εντολή routes μπορούμε να δούμε σε πιο path αντιστοιχούν οι μέθοδοι του ελεγκτή όπως φαίνεται στην παρακάτω εικόνα.



```
Terminal - michail@michail-00s: ~/rails/src/treegram
michail@michail-00s: ~/rails/src/treegram
user_comments GET    /users/:user_id/comments(.:format) comments#index
               POST    /users/:user_id/comments(.:format) comments#create
new_user_comment GET    /users/:user_id/comments/new(.:format) comments#new
edit_user_comment GET    /users/:user_id/comments/:id/edit(.:format) comments#edit
user_comment GET    /users/:user_id/comments/:id(.:format) comments#show
               PATCH   /users/:user_id/comments/:id(.:format) comments#update
               PUT     /users/:user_id/comments/:id(.:format) comments#update
               DELETE  /users/:user_id/comments/:id(.:format) comments#destroy
users GET         /users(.:format) users#index
               POST    /users(.:format) users#create
new_user GET      /users/new(.:format) users#new
edit_user GET     /users/:id/edit(.:format) users#edit
user GET         /users/:id(.:format) users#show
               PATCH   /users/:id(.:format) users#update
               PUT     /users/:id(.:format) users#update
               DELETE  /users/:id(.:format) users#destroy
tags POST        /tags(.:format) tags#create
tag DELETE       /tags/:id(.:format) tags#destroy
comments POST    /comments(.:format) comments#create
comment DELETE   /comments/:id(.:format) comments#destroy
log_in GET       /log-in(.:format) sessions#new
               POST    /log-in(.:format) sessions#create
log_out GET      /log-out(.:format) sessions#destroy
michail@michail-00s:~/rails/src/treegram$
```

Illustration 1: Εμφάνιση routes με το rake

Δίπλα στο προφίλ κάθε χρήστη εμφανίζεται ένα κουμπί “Follow” ώστε να μπορεί ο συνδεδεμένος χρήστης να ακολουθήσει κάποιον από τους υπόλοιπους. Το κουμπί “Follow” συνδέεται με μία νέα διαδρομή (“new\_relation\_path”) στον αντίστοιχο ελεγκτή (“relations\_controller”) ο οποίος δημιουργεί μια νέα “σχέση” μεταξύ των δυο χρηστών following – follower. Ο follower είναι ο τρέχων χρήστης και following ο χρήστης που επέλεξε να ακολουθήσει. Αφού προσθέσουμε τη μέθοδο “create” στον relations\_controller και καθορίσουμε τα χαρακτηριστικά των χρηστών, μπορούμε να την αποθηκεύσουμε στη βάση δεδομένων στον πίνακα relations χρησιμοποιώντας το

μοντέλο του χρήστη. Στο μοντέλο μπορεί να οριστεί ως `has_many :relations` και το `ActiveRecord` να εφαρμόσει αυτή τη σχέση.

*Εμφάνιση φωτογραφιών των ακολουθούμενων χρηστών*

Στο σημείο αυτό τροποποιήσαμε το `show.html.html` και τον `user controller` έτσι ώστε εφόσον έχει `following users` ο χρήστης, για κάθε έναν που ακολουθεί *παίρνουμε τις εικόνες του* και τις τοποθετούμε όλες μαζί σε ένα κοινό πίνακα. Έπειτα τις κάνουμε ταξινόμηση με βάση το πεδίο `created_at` και τις δείχνουμε στο χρήστη.

- Επειδή οι εικόνες είναι ένα υποσύνολο του πίνακα `user`, κάνοντας χρήση του `flatmap` operator, επιστρέφουμε πάλι ένα πίνακα, απλά μόνο με τις φωτογραφίες κάθε χρήστη.  
[ [φ1,φ2,φ3], [φ5,φ6,φ7] ]
- Για να μην έχουμε ένα πίνακα της μορφής `[[χ],[ζ]]` (δηλαδή κάθε στοιχείο του πίνακα να είναι πίνακας) κάναμε χρήση της μεθόδου `flatten` η οποία εφαρμόζεται σε έναν πίνακα και τα στοιχεία που είναι και αυτά πίνακες, τα αντικαταστέι με το στοιχείο που θέλουμε.

Π.χ [ [1],[2],[3]] `flatten` = [1,2,3]

### 3. Σχολιασμός φωτογραφιών

Ζητούμενο:

Στο κάτω μέρος κάθε φωτογραφίας υπάρχει δυνατότητα να προστεθεί σχόλιο από κάποιον χρήστη.

Για την υλοποίηση της λειτουργίας αυτής προστέθηκε ένας νέος ελεγκτής, “`comments_controller`”, το μοντέλο “`comment`” και τροποποιήθηκε ο προβολέας του χρήστη ώστε να εμφανίζεται η φόρμα για το σχόλιο και τα σχόλια που έχουν ήδη προστεθεί. Επίσης τροποποιήσαμε το μοντέλο `photo` για να καθορίσουμε τη σχέση μεταξύ μια φωτογραφίας και σχολίων (μια φωτογραφία μπορεί να έχει πολλά σχόλια). Δηλαδή στον κώδικα η προσθήκη

```
class Photo < ActiveRecord::Base
  has_many :comments
```

καθορίζει αυτή τη σχέση στο μοντέλο.

Στον ελεγκτή `comments_controller` δημιουργούμε ένα νέο σχόλιο με τη μέθοδο `create`, με τις παραμέτρους που έχουμε πρόσβαση από τον `ActionController` μέσω του hash “`params`”.

Όταν εμφανίζονται οι φωτογραφίες απο του χρήστες (και των ακολούθων) εμφανίζονται και τα σχόλια. Αυτό υλοποιείται στον προβολέα του χρήστη χρησιμοποιώντας τα `block` της γλώσσας `ruby` για να πάρουμε τα αντίστοιχα σχόλια για κάθε φωτογραφία.

### *Bonus* Διαγραφή Φωτογραφιών

Ζητούμενο:

Σε κάθε φωτογραφία θα εμφανίζεται κουμπί που θα επιτρέπει τη διαγραφή της φωτογραφίας. Όταν διαγράφεται μια φωτογραφία αφαιρούνται επίσης οι ετικέτες που είχε και τα σχόλια.

Για τη διαγραφή των φωτογραφιών χρησιμοποιείται η μέθοδος `delete` η οποία αντιστοιχεί στην ενέργεια `destroy` του `user controller`. Διαφορετικά, αν γινόταν μέσω `get`, οι χρήστες θα μπορούσαν να χρησιμοποιήσουν ειδικά `urls` και να διαγράψουν πόρους της εφαρμογής χωρίς να το θέλουμε.

Επίσης προσθέτουμε ένα `link` στον προβολέα του χρήστη με παραμέτρο τη φωτογραφία που θα διαγραφεί, τη μέθοδο `delete` και μια επιβεβαίωση αν ο χρήστης είναι σίγουρος γι' αυτή την ενέργεια.

Για τη διαγραφή των ετικετών και των σχολίων αρκεί στο μοντέλο της φωτογραφίας να προσθέσουμε στα πεδία `tags`, `comments` και `image` την επιλογή `“dependent: :destroy”` και το Rails μέσω του `ActiveRecord` θα διαγράψει από τη βάση δεδομένων τα πεδία αυτά από την αντίστοιχη φωτογραφία.