ΜΥΕ042 - ΤΕΧΝΟΛΟΓΙΕΣ ΔΙΑΔΙΚΤΥΟΥ ΛΑΜΠΡΟΣ ΒΛΑΧΟΠΟΥΛΟΣ 2948
ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ ΑΘΑΝΑΣΙΟΣ ΜΑΣΟΥΡΑΣ 2752

Εργαστηριακή Άσκηση 1
Εφαρμογή ιστού για κοινοχρησία φωτογραφιών στο Ruby-on-Rails

1. Γενικές πληροφορίες

Στην πρώτη εργαστηριακή άσκηση μας ζητήθηκε να επεκτείνουμε τις λειτουργίες της εφαρμογής treegram. Αρχικά μελετήσαμε την δομή και την λογική του Ruby-on-Rails και στην συνέχεια τον κώδικα της εφαρμογής ώστε να μπορέσουμε να προχωρήσουμε στα ζητούμενα της άσκησης. Καταφέραμε να υλοποιήσουμε τις τρεις λειτουργίες, το μπόνους αλλά και κάποιες μικρές αλλαγές που θεωρήσαμε ότι βελτιώνουν την εφαρμογή treegram. Παρακάτω θα διαβάσετε τις υλοποιήσεις μας, την περιγραφή των δομών αλλά και επεξηγήσεις για τα κομμάτια κώδικα που υπάρχουν ήδη ή που προστέθηκαν.

2. Υλοποιήσεις

i) Προσθέστε τίτλο σε μια φωτογραφία όταν την ανεβάζετε στον διακομιστή.

Το πρώτο πράγμα που αναζητήσαμε είναι το σημείο στο οποίο ο χρήστης επιλέγει την φωτογραφία για να την ανεβάσει στο treegram. Οπότε το πρώτο πράγμα που έγινε είναι να προσθέσουμε την παρακάτω εντολή στο αρχείο *`/app/views/photos/new.html.haml`*

```
= form.text_field :title, placeholder: 'Text Title'
```

Με αυτό συμπληρώνουμε την φόρμα όπου ο χρήστης δίνει τον τίτλο της φωτογραφίας που επιλέγει να κάνει upload.

Στη συνέχεια στο αρχείο */app/controllers/photos_controller.rb* προσθέσαμε το πεδίο *`:title`* στο action *`photo_params`*.

```
def photo_params
    params.require(:photo).permit(:image,:title)
  end
```

Με αυτόν τον τρόπο καθορίζουμε ότι και το πεδίο *`:title`* αποτελεί παράμετρο του αντικειμένου photo που κάνουμε new στο */app/views/photos/new.html.haml* που περιγράψαμε παραπάνω.

Στην συνέχεια έπρεπε να ενημερώσουμε και την βάση μας για να κρατάμε την πληροφορία του τίτλου κάθε φωτογραφίας. Συγκεκριμένα προσθέσαμε στον πίνακα *photos*, ένα πεδίο *string* με όνομα *title*.

Την προσθήκη αυτή την υλοποιήσαμε με την παρακάτω εντολή

```
rails generate migration add_title_to_photos title:string
```

Λόγω της αλλαγής στο schema θα πρέπει να εκτελέσουμε και την εντολή

```
rake db:migrate
```

για να ενημερωθεί σωστά η βάση μας.

Τέλος για να εμφανίζουμε τον τίτλο πάνω από την φωτογραφία συμπληρώνουμε στο αρχείο */app/views/users/show.html.haml* την εντολή

```
= simple_format 'Title: ' + photo.title
```

Τοποθετήθηκε εκεί καθώς διατρέχουμε όλες τις φωτογραφίες του χρήστη αλλά και για να εμφανίζεται πάνω από την κάθε φωτογραφία.

```
.row
  - @user.photos.each  do |photo|
    .well.col-sm-4

      = button_to "Delete Photo", user_photo_path(@user.id ,photo.id)
      = simple_format 'Title: ' + photo.title
      = image_tag photo.image.url(:medium)
```

ii) Επιτρέψτε στους χρήστες να καθορίσουν χρήστες που ακολουθούν

Αρχικά δημιουργήσαμε ένα πίνακα στη βάση μας με όνομα *follows* που κρατάμε τα αναγνωστικά id των χρηστών που ανήκουν στην συσχέτιση follow.

Αυτό το κάναμε πάλι στο terminal με την εντολή

```
rails generate model follows follower_id:string followee_id:string
```

και αφού εκτελέσαμε αυτή την εντολή κάναμε ξανά

```
rake db:migrate
```

για να ενημερώσουμε τη βάση μας. Στο `db/migrate` θα βρούμε το *`20221124231510_create_follows.rb`* που δημιουργήθηκε από την εντολή μας. Επίσης ενημερώνεται αυτόματα και το αρχείο *`db/schema.rb`* με το παρακάτω block

```
create_table "follows", id: false, force: :cascade do |t|
    t.integer  "follower_id"
    t.integer  "followee_id"
    t.datetime "created_at",  null: false
    t.datetime "updated_at",  null: false
  end
```

Στην συνέχεια δημιουργήσαμε το αρχείο *`app/models/follow.rb`*

```
class Follow < ActiveRecord::Base
                belongs_to :follower,class_name: "User"
                belongs_to :followee,class_name: "User"
end
```

Οι εντολές *`belongs_to`* δηλώνουν την συσχέτιση μεταξύ δύο χρηστών. Για δική μας κατανόηση απλά ονοματίζουμε τον χρήστη που θέλει να ακολουθήσει προς τον χρήστη που θα ακολουθηθεί. Το μοντέλο δηλάδή αφορά έναν χρήστη ως προς έναν άλλον.

Δημιουργήσαμε ένα αρχείο *`app/controllers/follow_controller.rb`*

```
class FollowController < ApplicationController
      def create
        @follow=Follow.new(follow_params)
        @follow.save
        redirect_to :back
      end

      def index
        @users = User.all
      end
      def follow_params
         params.require(:follow).permit(:follower_id, :followee_id )
      end
end
```

Η συνάρτηση *`create`* δημιουργεί ένα αντικείμενο follow και το αποθηκεύει στη βάση δεδομένων και στη συνέχεια με το *`redirect_to :back`* επιστρέφουμε πίσω στη σελίδα μας.Στην συνάρτηση *`index`* κρατάμε όλους τους χρήστες που υπάρχουν στη βάση μας.

Η *follow_params* καθορίζει τις παραμέτρους του follow αντικειμένου που κάνουμε *new* στο αρχείο *app/views/follow/index.html.haml* τον οποίο φτιάξαμε και θα διαβάσετε παρακάτω.

```
%h1
.row.top_row
  .col-sm-6.user_att
  .col-sm-6
    = link_to 'Home', user_path(session[:user_id]), class: ['btn',
'btn-primary', 'logout_btn']
%br/
%br/
%table{class: 'table', style: 'max-width: 25rem;'}
  %thead
    %tr
      %th Email User
  %tbody
    - @users.each do |user|
      %tr
      %td= user.email
      %td
       = form_for Follow.new ,:url =>
user_follow_path(session[:user_id],user.id)  do |f|
       = f.hidden_field   :follower_id ,value: session[:user_id]
       = f.hidden_field   :followee_id, value: user.id
       - if Follow.find_by(follower_id:  session[:user_id],followee_id: user.id)
            = button_to "Followed"
       - else
            - if session[:user_id]!= user.id
            =  button_to "Follow", user_follow_path(session[:user_id],user.id)
```

Με τον κώδικα εδώ αρχικά δημιουργούμε ένα κουμπί με όνομα Home που θα βλέπει ο χρήστης και θα επιστρέφει στην αρχική σελίδα όταν δε θέλει να ακολουθήσει άλλους χρήστες.

Στη συνέχεια εκτυπώνουμε σε μορφή πίνακα μια στήλη με όνομα Email User και για κάθε χρήστη που είναι στη βάση εκτυπώνουμε το email του σε μια γραμμή.

Επιπλέον δημιουργούμε και μια στήλη με κουμπιά follow/followed δίπλα από τα email των χρηστών. Ο χρήστης πατώντας το κουμπί follow, ακολουθεί τον αντίστοιχο user και το κουμπί αλλάζει σε followed ώστε να γνωρίζει ο χρήστης ποιους ακολουθεί ήδη.

Αξίζει να σημειωθεί ότι εδώ δημιουργούμε αντικείμενα Follow για κάθε χρήστη και αρχικοποιούμε τα πεδία τους.

Η εντολή *.find_by(..)* ελέγχει στη βάση για το αν υπάρχει καταχώρηση, στην οποία ο χρήστης *session[:user_id]* (δηλαδή ο χρήστης που συνδέθηκε) ακολουθεί τον χρήστη με

id από το κουμπί που πάτησε ο χρήστης (δηλ. κάθε κουμπί είναι συνδεδεμένο με τον εκάστοτε user που βλέπει ο χρήστης).

Έτσι δεν γράφουμε στην βάση ότι ένας user ακολουθεί έναν άλλον δύο φορές. Επίσης , δεν εμφανίζεται κουμπί follow στον user που έχει κάνει login , καθώς δεν μπορεί να ακολουθεί τον εαυτό του.

Για να μπορέσουμε να μεταβούμε στην συγκεκριμένη σελίδα με τις παραπάνω λειτουργίες έχουμε τοποθετήσει στο αρχείο *app/views/users/show.html.haml* την εντολή

```
= link_to 'Users', user_follow_index_path(@user), class: ['btn', 'btn-info',
'userslist_btn']
```

Έτσι δημιουργούμε ένα κουμπί που μας οδηγεί στο path *user_follow_index_path(@user)*. Το path αυτό, δημιουργείται αυτόματα με την προσθήκη του κώδικα

```
resources :users do
  resources :follow
end
```

στο αρχείο *config/routes.rb*.

Εδώ να πούμε ότι το path αυτό μας οδηγεί στην συνάρτηση *index* που υπάρχει στο αρχείο *app/controllers/follow_controller.rb*. Αυτή με τη σειρά της μας οδηγεί στο *app/views/follow/index.html.haml*.

Όταν ο χρήστης επιλέγει το κουμπί follow αυτο που γίνεται ειναι να καλείτε η συνάρτηση `create` που υπάρχει στο αρχείο *app/controllers/follow_controller.rb*. Αυτό γίνεται με την προσθήκη της εντολής

```
post '/users/:id/follow/:id' => "follow#create"
```

που υπάρχει στο αρχείο *config/routes.rb*

αλλά και με τη χρήση του

```
user_follow_path(session[:user_id],user.id)
```

στο αρχείο *app/views/follow/index.html.haml*

Για το frontend κομμάτι του κουμπιού σχετικά με την τοποθέτηση και το style του στο UI της εφαρμογής προσθέσαμε στο αρχείο *app/assets/stylesheets/application.sass* το παρακάτω

```
.userslist_btn
  position: fixed
  right: 0
  bottom: 60%
  padding: 3%
  margin: 15px
```

Τέλος για να εμφανίσουμε τις φωτογραφίες των χρηστών που ακολουθεί κάποιος, στο αρχείο */app/views/users/show.html.haml* τον παρακάτω κώδικα

```
- Photo.where(user_id: @user.followees).order(:created_at).reverse_order.each do
|photo|
    .well.col-sm-4
      = simple_format 'Title: ' + photo.title
      = image_tag photo.image.url(:medium)
      = photo.created_at
```

Με αυτόν τον τρόπο έχουμε πρόσβαση στις φωτογραφίες όλων των user που ακολουθούμε με αντίστροφη χρονολογική σειρά της ώρας που φορτώθηκαν στο σύστημα.

Για να δημιουργήσουμε την @user.followees (πίνακας που δημιουργείται αυτόματα, όταν προσθέσουμε τις παρακάτω εντολές και περιλαμβάνει πίνακα με τα ids των χρηστών που ακολουθεί το @user) πήγαμε στο αρχείο *app/models/user.rb* και προσθέσαμε τον εξής κώδικα :

```
has_many :followed_users,
  foreign_key: :follower_id,
  class_name: 'Follow',
  dependent: :destroy
has_many :followees, through: :followed_users ,dependent: :destroy
has_many :following_users,
  foreign_key: :followee_id,
  class_name: "Follow",
  dependent: :destroy
has_many :followers, through: :following_users ,dependent: :destroy
```

Με αυτόν τον τρόπο λέμε ότι κάθε χρήστης έχει μπορεί να ακολουθεί πολλούς άλλους χρήστες αλλά και ότι ο ίδιος μπορει να ακολουθείτε από πολλούς. Εδώ αξίζει να σημειωθεί ότι έχουμε δώσει σαν ξένα κλειδιά στους πινακες τα ids που έχουμε στον πίνακα follows , δηλώνουμε επίσης την λειτουργία καταστροφής (διαγραφής) αν αλλάξει κάτι στον πίνακα follows.

iii) Επιτρέψτε τους χρήστες να προσθέσουν σχόλια σε φωτογραφίες

Αρχικά δημιουργήσαμε τον πίνακα comments στη βάση με την εντολή

```
rails generate model comments photo_id:integer user_id:integer comment:string
```

και στη συνέχεια

```
rake db:migrate
```

για να ενημερώσουμε τη βάση μας με τις αλλαγές.

Στη συνέχεια φτιάξαμε ένα αρχείο μοντέλο *app/models/comment.rb* το οποίο περιλαμβάνει

```
class Comment < ActiveRecord::Base
      belongs_to :user
      belongs_to :photo
end
```

Οι εντολές *belongs_to* δηλώνουν ότι το μοντέλο αφορά χρήστη ως προς φωτογραφία . Δηλαδή ότι ένα αντικείμενο user "κάνει comment" σε αντικείμενο *photo*.

Επίσης δημιουργήσαμε εναν controller *app/controllers/comment_controller.rb* που περιλαμβάνει :

```
class CommentController < ApplicationController
      def create

            @comment=Comment.new(comment_params)
            @comment.save
            redirect_to :back
      end


      def comment_params
      params.require(:comment).permit(:photo_id, :user_id, :comment )
    end
end
```

Η συνάρτηση *create* δημιουργεί ένα αντικείμενο *comment* χρησιμοποιώντας την συνάρτηση *comment_params* που καθορίζει τις παραμέτρους του αντικειμένου *Comment* που δημιουργουμε στο */app/views/users/show.html.haml* και το αποθηκεύει στη βάση δεδομένων.

Στη συνέχεια για να εμφανίζουμε στον χρήστη μια φόρμα ώστε να γράψει το σχόλιο για την κάθε φωτογραφία που θέλει στο αρχείο *app/views/users/show.html.haml* και προσθέσαμε τον κώδικα :

```
=form_for Comment.new, :url =>user_comment_index_path(session[:user_id])
do |com|
        =com.hidden_field :photo_id, value: photo.id
        =com.hidden_field :user_id, value: session[:user_id]
        =com.text_field :comment, placeholder: 'Text Comment'
        =com.submit "Add Comment"
      - Comment.where(photo_id: photo.id).each do |comment|
        .p
          = "User: #{comment.user.email}"
          =  comment.comment
```

Εδώ δημιουργούμε ένα νέο αντικείμενο *Comment* και του αρχικοποιούμε τα πεδία *(photo_id , user_id)*. Για το πεδίο *comment* εμφανίζουμε στο χρήστη μια φόρμα για να συμπληρώσει το σχόλιο που θέλει και ενα κουμπί για να ολοκληρώσει την διαδικασία. Έτσι όταν πατάει το κουμπί 'add comment' αρχικοποιούμε και το πεδίο *comment* με το κείμενο που έδωσε ο χρήστης στη φόρμα.

Επίσης το πρόγραμμα με την εντολή

```
 :url =>user_comment_index_path(session[:user_id])
```

θα κατευθυνόταν στον *app/controllers/comment_controller.rb* και στην συνάρτηση *index*, όμως επειδή βαζουμε την εντολή

```
post "/users/:id/comment/:id/" => "comment#create"
```

στο αρχείο *config/routes.rb,* η ροή του προγράμματος θα είναι να κατευθυνθεί στην *create* που υπάρχει στο *app/controllers/comment_controller.rb* και αφού καθορίσουμε τα πεδία του αντικειμένου θα αποθηκεύσουμε το νέο μας αντικείμενο *Comment* στη βάση.

Αξίζει να σημειώσουμε ότι το *user_comment_index_path* δημιουργήθηκε από το

```
resources :users do
  resources :comment
end
```

που τοποθετηθηκε στο *config/routes.rb*

Επίσης με τις εντολές

```
 - Comment.where(photo_id: photo.id).each do |comment|
      .p
            = "User: #{comment.user.email}"
            =  comment.comment
```

Εμφανιζουμε τα comments για κάθε φωτογραφία και το email του σχολιαστή.

Εδώ να πούμε ότι στο *app/views/comment* δε χρειάζεται να έχουμε κάποιο .haml αρχείο καθώς εκτελούμε κατευθείαν *create*, γραφουμε στη βαση και με το *redirect_to :back* επιστρέφουμε ξανά στη σελίδα μας.

iv) <u>Λειτουργία διαγραφής</u>

Για το bonus αρχικά τοποθετήσαμε ένα κουμπί "delete" στο αρχείο *app/views /users/show.html.haml* με την εντολή

```
 = button_to "Delete Photo", user_photo_path(@user.id ,photo.id)
```

όταν πατηθεί το κουμπί ,με την προσθήκη της εντολής

```
post "users/:id/photos/:id" => "photos#destroy"
```

στο αρχείο *config/routes.rb*, η ροή του προγράμματος θα κατευθυνθεί στο αρχείο *app/controllers/photo_controller.rb* και συγκεκριμένα στη συνάρτηση *destroy* που συπληρώσαμε με κώδικα όπως παρακάτω

```
 def destroy

    Photo.delete(params[:id])
    Tag.where(photo_id: params[:id]).delete_all
    Comment.where(photo_id: params[:id]).delete_all

    redirect_to user_path(session[:user_id])
  end
```

Εδώ αυτο που κάνουμε είναι αφού πάρουμε από το path της μορφής "photos/:id" το id της φωτογραφίας που επιλέγει ο χρήστης για να την διαγράψει, καλούμε την *delete* που σβήνουμε από την βάση μας αυτή της φωτογραφίας . (πινακας photos).

Επίσης σβήνουμε και από τους πίνακες *tags*, *comments* όλες τις εγγραφές που αφορούσαν αυτό το *photo_id*.

Εδώ να πούμε ότι για να δουλέψει αυτό συμπληρώσαμε στις κλασεις Tag, Comment το παρακάτω

```
class Comment < ActiveRecord::Base
      belongs_to :user
      belongs_to :photo, foreign_key: :photo_id, dependent:
:destroy

end
```

```
class Tag<ActiveRecord::Base
  belongs_to :user
  belongs_to :photo, foreign_key: :photo_id, dependent: :destroy
end
```

Τέλος με την εντολή

```
redirect_to user_path(session[:user_id])
```

απλα μένουμε στην σελίδα που βλέπουμε τις φωτογραφίες του χρήστη.

v) Επιπλέον αλλαγές και επεξηγήσεις

Επίσης τροποποιήσαμε τον αρχικό κώδικα της συνάρτησης *create* και *new* στο *app/controllers/photo_controller.rb* ως εξής:

```
def create
    @user = User.find(params[:user_id])
    if params[:photo] == nil

      flash[:alert] = "Please upload a photo"
      redirect_to :back
    else

        ##check report for explanation##
        @photo = Photo.new(photo_params)   #this line was @photo = Photo.create(photo_params)
```

```
        @photo.user_id = @user.id
        @photo.save
        flash[:notice] = "Successfully uploaded a photo"
        #flash[:notice] = @photo.title
        redirect_to user_path(@user)
    end
  end
```

```
def new
    @user = User.find(params[:user_id])

    ##check report for explanation##
    #@photo = Photo.create()    this line deleted

  end
```

Αυτό το κάναμε καθώς παρατηρήσαμε μια ανωμαλία στον πίνακα photos όταν ανεβάζαμε μια φωτογραφία. Αυτό που γινόταν είναι ότι ο πίνακας photos ενημερωνόταν με μια εγγραφή που περιελάμβανε μόνο τα πεδία created_at και update_at και στη συνέχεια σε δεύτερη εγγραφή αποθήκευε την φωτογραφία και τον τίτλο . Αυτό το θεωρήσαμε μη λειτουργικό και το αλλάξαμε.

Επίσης επειδή παρατηρήσαμε ότι αν αλλάζαμε απο το url της μορφής /users/1 σε '/users/2 το πρόγραμμα λειτουργούσε κανονικά χωρίς να ζητάει κωδικό (και προφανώς είχαμε δημιουργήσει δύο users) προσθέσαμε την εντολή *session[:user_id]=@user.id* στην συνάρτηση show του users_controller με σκοπό να τεστάρουμε γρήγορα τη σελίδα μας ότι είναι λειτουργική.

Αυτό σε μια πραγματική σελίδα θα έπρεπε να απαγορεύεται γιατί κάποιος που θα συνδεόταν με τα στοιχεία του θα μπορούσε να έχει πρόσβαση σε άλλους χρήστες χωρίς κωδικό.

Κρίσιμο σημείο αποτέλεσαν η εύρεση των path αλλα με την εντολή rake routes στο terminal καταλάβαμε τα paths που δημιουργήσαμε και πως θα συνδεθούμε με τους controllers αλλα και σε ποια συνάρτηση θα καταλήξουμε και με τι ορίσματα στα path .Για παράδειγμα, τα ορίσματα που βάζουμε καθε φορα στα path της μορφής @user.id , @user, @photo.id.

Επίσης να αναφέρουμε ότι στο αρχείο */app/views/users/show.html.haml*

Το παρακάτω block αφορά τις εικόνες που ανεβάζει ο χρήστης.

```
.row
  - @user.photos.each  do |photo|
    .well.col-sm-4

      = button_to "Delete Photo", user_photo_path(@user.id ,photo.id)
      = simple_format 'Title: ' + photo.title
      = image_tag photo.image.url(:medium)
      =form_for Comment.new, :url =>user_comment_index_path(session[:user_id]) do |com|
        =com.hidden_field :photo_id, value: photo.id
        =com.hidden_field :user_id, value: session[:user_id]
        =com.text_field :comment, placeholder: 'Text Comment'
        =com.submit "Add Comment"
      - Comment.where(photo_id: photo.id).each do |comment|
        .p
          = "User: #{comment.user.email}"
          =  comment.comment


      = form_for @tag do |f|

        =f.hidden_field :photo_id, value: photo.id
        = f.collection_select :user_id, @users, :id, :email

        = f.submit "Tag User"
      - photo.tags.each do |tag|
        = tag.user.email
```

Ενώ το παρακάτω block αφορά τις φωτογραφίες των χρηστών που ακολουθεί.

```
.row
  - Photo.where(user_id: @user.followees).order(:created_at).reverse_order.each do
|photo|
    .well.col-sm-4
      = simple_format 'Title: ' + photo.title
      = image_tag photo.image.url(:medium)
      = photo.created_at

      .p
        = 'Tags:'
        - photo.tags.each do |tag|
          = tag.user.email

      =form_for Comment.new, :url =>user_comment_index_path(session[:user_id]) do |com|
        =com.hidden_field :photo_id, value: photo.id
        =com.hidden_field :user_id, value: session[:user_id]
        =com.text_field :comment, placeholder: 'Text Comment'
        =com.submit "Add Comment"
      - Comment.where(photo_id: photo.id).each do |comment|
        .p
          = "User: #{comment.user.email}"
          =  comment.comment
```