
## ΤΕΧΝΟΛΟΓΙΕΣ ΔΙΑΔΙΚΤΥΟΥ

Εφαρμογή ιστού για διαμοιρασμό φωτογραφιών στο Ruby-on-Rails.

## Team

Γκόβαρης Χρήστος-Γρηγόριος
Σπανού Μαρία

# Χρονοδιάγραμμα 1ης Εργαστηριακής Άσκησης

| Έκδοση | Ημερομηνία | Progress |
|---|---|---|
| 1.0 | 8/11/2024 | Ανακοίνωση 1ης εργαστηριακής άσκησης |
| 2.0 | 16/11 - 18/11 | Υλοποίηση του ερωτήματος (1) |
| 2.1 | 19/11/2024 | Υλοποίηση του ερωτήματος (2) |
| 2.2 | 20/11 - 21/11 | Υλοποίηση του ερωτήματος (3) |
| 2.3 | 22/11 - 23/11 | Υλοποίηση του ερωτήματος (4) |
| 2.4 | 23/11 - 24/11 | Υλοποίηση του bonus ερωτήματος (5) |
| 3.0 | 16/11 – 24/11 | Υλοποίηση Report εργαστηριακής άσκησης |

# Ανάλυση Χαρακτηριστικών

Η εργασία υλοποιήθηκε σε ένα μηχάνημα (Lenovo Ideapad 3), με τα εξής χαρακτηριστικά πυρήνα:

- AMD Ryzen 7 (7730U)
- 8 CPU cores
- 16 Threads
- Boost Clock up to 4.5GHz
- Base Clock 2.0GHz
- CPU Socket FP6
- Intergraded Graphics (Radeon Graphics)

Επιπλέον, η **εργασία** υλοποιήθηκε σε **Debian 64-bit Linux**, χρησιμοποιώντας το **Mousepad**.

# Αρχεία που τροποποιήθηκαν

Τροποποιήθηκαν τα παρακάτω αρχεία:

| Models | Controllers | Views | Routes | Migrations |
|---|---|---|---|---|
| app/models/user.rb | app/controllers/users_controller.rb | app/views/users/show.html.haml | config/routes.rb | db/migrate/create_photos.rb |
| app/models/photo.rb | app/controllers/photos_controller.rb | app/views/users/new.html.haml | | db/migrate/create_comments.rb |
| app/models/comment.rb | app/controllers/comments_controller.rb | app/views/photos/show.html.haml | | db/migrate/create_follows.rb |
| app/models/follow.rb | | | | db/migrate/add_title_to_photos.rb |

# 2.0 Υλοποίηση του ερωτήματος (1)

Για το 1ο ερώτημα, προσθέσαμε τη δυνατότητα κάθε φωτογραφία να έχει τίτλο. Δημιουργήσαμε ένα migration (**add_title_to_photos.rb**) για τη στήλη title στον πίνακα photos και ενημερώσαμε το μοντέλο Photo για να περιλαμβάνει validation για τον τίτλο. Στη φόρμα ανέβασμα φωτογραφίας (**users/show.html.haml**), προσθέσαμε πεδίο για την εισαγωγή τίτλου, ενώ τροποποιήσαμε το **photo_params** στο **PhotosController** ώστε να αποθηκεύεται σωστά. Μετά τη migration, εμφανίσαμε τον τίτλο κάτω από κάθε φωτογραφία στη σελίδα του χρήστη και επαληθεύσαμε τη λειτουργικότητα μέσω δοκιμών.

Πιο συγκεκριμένα, οι αλλαγές ήταν οι εξής:

## Migration: add_title_to_photos.rb

```
class AddTitleToPhotos < ActiveRecord::Migration
  def change
    add_column :photos, :title, :string
  end
end
```

Αυτό το migration προσθέτει τη στήλη **title** στον πίνακα **photos** στη βάση δεδομένων, με τύπο δεδομένων **string**. Είναι το θεμέλιο για να υποστηριχθεί η αποθήκευση των τίτλων φωτογραφιών. Μετά τη δημιουργία του αρχείου, εκτελέσαμε την εντολή **rake db:migrate** για να ενημερωθεί η βάση δεδομένων.

## View: users/show.html.haml

```
%p
  %strong Title:
  = photo.title
```

Αυτό το κομμάτι προστέθηκε στο view για να εμφανίζεται ο τίτλος κάθε φωτογραφίας στη σελίδα του χρήστη. Ο τίτλος αντλείται από τη βάση δεδομένων μέσω του **photo.title** και εμφανίζεται κάτω από κάθε φωτογραφία.

## Controller: photos_controller.rb

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

Αυτή η μέθοδος ενημερώθηκε για να επιτρέπει την αποθήκευση του πεδίου **title** όταν δημιουργείται μια νέα φωτογραφία. Το **permit** διασφαλίζει ότι μόνο τα επιτρεπόμενα πεδία, όπως **image** και **title**, αποθηκεύονται στη βάση δεδομένων, εξασφαλίζοντας την ασφάλεια της εφαρμογής.

## Controller: photos_controller.rb

```
def create
  @user = User.find(params[:user_id])
  if params[:photo].nil?
    flash[:alert] = "Please upload a photo"
    redirect_to :back
  else
    @photo = Photo.create(photo_params)
    @photo.user_id = @user.id
    @photo.save
    flash[:notice] = "Successfully uploaded a photo"
    redirect_to user_path(@user)
  end
end
```

Η μέθοδος **create** ενημερώθηκε ώστε να αποθηκεύει τον τίτλο της φωτογραφίας όταν αυτή δημιουργείται. Το **@photo = Photo.create(photo_params)** παίρνει τα δεδομένα του τίτλου από τη φόρμα και τα αποθηκεύει στη βάση δεδομένων μαζί με το υπόλοιπο περιεχόμενο της φωτογραφίας.

## Models: photo.rb

```
validates :title, presence: true
```

Αυτό το validation διασφαλίζει ότι κάθε φωτογραφία πρέπει να έχει έναν τίτλο (**title**). Εάν ο χρήστης προσπαθήσει να αποθηκεύσει μια φωτογραφία χωρίς τίτλο, η Rails θα απορρίψει το αίτημα αποθήκευσης και θα επιστρέψει ένα μήνυμα σφάλματος. Είναι χρήσιμο για να εξασφαλιστεί η πληρότητα των δεδομένων.

## 2.1 Υλοποίηση του ερωτήματος (2)

Στο 2ο ερώτημα, υλοποιήσαμε τη δυνατότητα **follow/unfollow** μεταξύ χρηστών, δημιουργώντας το μοντέλο **Follow** για τη διαχείριση σχέσεων **follower** και **followed**. Ενημερώσαμε το **routes.rb** για τις διαδρομές και τον **UsersController** για τη διαχείριση των ενεργειών. Στο **view**, προσθέσαμε κουμπιά **follow/unfollow** και εμφανίσαμε φωτογραφίες ακολουθούμενων χρηστών στη σελίδα χρήστη.

Πιο συγκεκριμένα, οι αλλαγές ήταν οι εξής:

Controllers: users_controller.rb

```
def follow
  user_to_follow = User.find(params[:id])
  if current_user.followed_users.include?(user_to_follow)
    flash[:alert] = "You are already following this user."
  else
    current_user.followed_users << user_to_follow
    flash[:notice] = "You are now following #{user_to_follow.email}!"
  end
  redirect_to user_path(user_to_follow)
end

def unfollow
  user_to_unfollow = User.find(params[:id])
  if current_user.followed_users.include?(user_to_unfollow)
    current_user.followed_users.delete(user_to_unfollow)
    flash[:notice] = "You unfollowed #{user_to_unfollow.email}."
  else
    flash[:alert] = "You are not following this user."
  end
  redirect_to user_path(user_to_unfollow)
end
```

Οι μέθοδοι **follow** και **unfollow** διαχειρίζονται την ακολούθηση χρηστών. Στη **follow**, αν ο **current_user** δεν ακολουθεί τον **user_to_follow**, προστίθεται η σχέση. Στην **unfollow**, η σχέση αφαιρείται, με εμφάνιση κατάλληλου μηνύματος.

## Models: user.rb

```
has_many :follows_as_follower, class_name: 'Follow', foreign_key: 'follower_id'
has_many :followed_users, through: :follows_as_follower, source: :followed
has_many :follows_as_followed, class_name: 'Follow', foreign_key: 'followed_id'
has_many :followers, through: :follows_as_followed, source: :follower
```

Προσθέσαμε σχέσεις στο μοντέλο **User** για τη λειτουργία **follow/unfollow**. Οι **follows_as_follower** και **follows_as_followed** ορίζουν χρήστες που ακολουθεί ένας χρήστης (**followed_users**) ή τον ακολουθούν (**followers**), μέσω του μοντέλου **Follow**.

## Models: follow.rb

```
class Follow < ActiveRecord::Base
  belongs_to :follower, class_name: 'User', foreign_key: 'follower_id'
  belongs_to :followed, class_name: 'User', foreign_key: 'followed_id'

  validates :follower_id, presence: true
  validates :followed_id, presence: true
  validates :follower_id, uniqueness: { scope: :followed_id }
end
```

Το μοντέλο **Follow** διαχειρίζεται τις σχέσεις **follow/unfollow**. Περιλαμβάνει δύο συσχετίσεις: το **follower** για τον χρήστη που ακολουθεί και το **followed** για τον χρήστη που ακολουθείται. Τα **validations** διασφαλίζουν μοναδικότητα και την παρουσία των πεδίων **follower_id** και **followed_id**.

## Config: routes.rb

```
resources :users do
  member do
    post :follow
    delete :unfollow
  end
end
```

Προστέθηκαν διαδρομές για το **follow** και το **unfollow**. Αυτές οι διαδρομές χρησιμοποιούν τα **HTTP methods POST** και **DELETE**, αντίστοιχα, και συνδέονται με τις μεθόδους στον **UsersController**. Αυτό επιτρέπει την αλληλεπίδραση μέσω των κουμπιών **follow/unfollow**.

Views: users/show.html.haml

```
%h2 All Users

.row

 - @all_users.each do |user|

   .col-sm-4

     %p

       = user.email

       - if current_user != user

         - if current_user.followed_users.include?(user)

           = button_to 'Unfollow', unfollow_user_path(user), method: :delete, class: ['btn', 'btn-warning']

         - else

           = button_to 'Follow', follow_user_path(user), method: :post, class: ['btn', 'btn-primary']


%h2 Photos of <%= @user.email %> and Followed Users

.row

 - @photos.each do |photo|

   .well.col-sm-4

     = image_tag photo.image.url(:medium)

     %p

       %strong Title:

       = photo.title
```

Προστέθηκαν κουμπιά **Follow** και **Unfollow** στη λίστα χρηστών. Το **Follow** εμφανίζεται αν ο τρέχων χρήστης δεν ακολουθεί τον άλλον χρήστη, ενώ το **Unfollow** αν τον ακολουθεί ήδη. Επιπλέον, τα **@photos** στη σελίδα περιλαμβάνουν φωτογραφίες των ακολουθούμενων χρηστών, μαζί με τις φωτογραφίες του ιδιοκτήτη της σελίδας, ταξινομημένες αντίστροφα βάσει ημερομηνίας.

## 2.2 Υλοποίηση του ερωτήματος (3)

Στο 3ο ερώτημα, υλοποιήσαμε τη δυνατότητα οι χρήστες να προσθέτουν και να διαγράφουν σχόλια σε φωτογραφίες. Δημιουργήσαμε το μοντέλο Comment για να διαχειρίζεται τα σχόλια, με συσχετίσεις με τα μοντέλα **Photo** και **User**. Προσθέσαμε το **CommentsController** για να διαχειρίζεται τις λειτουργίες δημιουργίας και διαγραφής σχολίων, με έλεγχο δικαιωμάτων ώστε ένας χρήστης να μπορεί να διαγράψει μόνο τα δικά του σχόλια ή σχόλια στις δικές του φωτογραφίες. Στη σελίδα προβολής χρηστών (**users/show.html.haml**), προσθέσαμε φόρμα για νέα σχόλια και λίστα για την εμφάνισή τους κάτω από κάθε φωτογραφία, με κουμπί διαγραφής για εξουσιοδοτημένους χρήστες. Τέλος, ενημερώσαμε το **routes.rb** για τις διαδρομές δημιουργίας και διαγραφής σχολίων.

Πιο συγκεκριμένα, οι αλλαγές ήταν οι εξής:

### Models: comment.rb

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

  validates :text, presence: true
end
```

Το νέο μοντέλο **Comment** δημιουργήθηκε για να διαχειρίζεται τα σχόλια. Περιλαμβάνει: Συσχέτιση με το μοντέλο **Photo** μέσω **belongs_to :photo**. Συσχέτιση με το μοντέλο **User** μέσω **belongs_to :user**. **Validation** για την παρουσία του πεδίου **text**, διασφαλίζοντας ότι τα σχόλια δεν είναι κενά.

### Models: photo.rb

```
has_many :comments, dependent: :destroy
```

Στο μοντέλο Photo, προστέθηκε η συσχέτιση **has_many :comments**. Αυτή επιτρέπει την αναφορά στα σχόλια που σχετίζονται με μια φωτογραφία. Το **dependent: :destroy** διασφαλίζει ότι όταν διαγράφεται μια φωτογραφία, διαγράφονται και όλα τα σχόλια που ανήκουν σε αυτήν.

Controllers: comments_controller.rb

```
def create
  @photo = Photo.find(params[:photo_id])
  @comment = @photo.comments.new(comment_params)
  @comment.user = current_user
  if @comment.save
    flash[:notice] = "Comment added successfully!"
  else
    flash[:alert] = "Failed to add comment. Please try again."
  end
  redirect_to user_path(@photo.user)
end

def destroy
  @comment = Comment.find(params[:id])
  @photo = @comment.photo
  if current_user == @comment.user || current_user == @photo.user
    @comment.destroy
    flash[:notice] = "Comment deleted successfully!"
  else
    flash[:alert] = "You are not authorized to delete this comment."
  end
  redirect_to user_path(@photo.user)
end

private

def comment_params
  params.require(:comment).permit(:text)
end
```

**create:** Δημιουργεί σχόλια για μια συγκεκριμένη φωτογραφία. Ορίζεται ο χρήστης (**current_user**) που δημιούργησε το σχόλιο. **destroy:** Διαγράφει σχόλια αν ο τρέχων χρήστης είναι είτε ο δημιουργός του σχολίου είτε ο ιδιοκτήτης της φωτογραφίας. **comment_params:** Διασφαλίζει ότι μόνο το πεδίο text επιτρέπεται να αποθηκευτεί, για προστασία από μη έγκυρα δεδομένα.

## Controllers: users_controller.rb

```
@comment = Comment.new
```

Στη μέθοδο **show**, προστέθηκε η δημιουργία ενός νέου αντικειμένου **@comment** για τη φόρμα σχολίων που εμφανίζεται στο **view**.

## Views: users/show.html.haml

```
%h3 Comments:

- if photo.comments.any?

  - photo.comments.each do |comment|

    %p

      %strong= comment.user.email

      = comment.text

      - if current_user == comment.user || current_user == photo.user

        = button_to 'Delete', photo_comment_path(photo, comment), method: :delete, data: { confirm: 'Are you sure?' }, class: ['btn', 'btn-danger', 'btn-sm']

- else

  %p No comments yet.


%h4 Add a Comment:
= form_for [photo, Comment.new] do |f|

  = f.text_area :text, placeholder: "Write your comment here...", rows: 3, class: "form-control"

  %br/

  = f.submit "Add Comment", class: ['btn', 'btn-primary']
```

Προστέθηκε λίστα εμφάνισης σχολίων κάτω από κάθε φωτογραφία. Κάθε σχόλιο συνοδεύεται από το email του δημιουργού του. Προστέθηκε κουμπί **Delete** για διαγραφή σχολίων, το οποίο εμφανίζεται μόνο για τον δημιουργό του σχολίου ή τον ιδιοκτήτη της φωτογραφίας. Προστέθηκε φόρμα για τη δημιουργία νέων σχολίων

## Config: routes.rb

```
resources :photos, only: [:destroy] do

  resources :comments, only: [:create, :destroy]
end
```

Προστέθηκαν nested διαδρομές για τα σχόλια κάτω από τις φωτογραφίες. Οι διαδρομές **create** και **destroy** ενεργοποιούν τις αντίστοιχες ενέργειες στον **CommentsController**.

# 2.4 Υλοποίηση του ερωτήματος (4)

Στο 4ο ερώτημα, υλοποιήσαμε τη δυνατότητα οι χρήστες να διαγράφουν σχόλια, με βάση συγκεκριμένα δικαιώματα. Προσθέσαμε τη μέθοδο **destroy** στον **CommentsController**, η οποία επιτρέπει σε έναν χρήστη να διαγράψει σχόλια που έχει γράψει ο ίδιος ή σχόλια στις δικές του φωτογραφίες. Στο **view** (**users/show.html.haml**), προστέθηκε κουμπί διαγραφής (**Delete**) δίπλα σε κάθε σχόλιο, το οποίο εμφανίζεται μόνο αν ο τρέχων χρήστης έχει το δικαίωμα να το διαγράψει. Τέλος, ενημερώσαμε το **routes.rb** για να υποστηρίξει τη λειτουργία διαγραφής (**destroy**) για τα σχόλια.

Πιο συγκεκριμένα, οι αλλαγές ήταν οι εξής:

## Config: routes.rb

```
resources :photos, only: [:destroy] do
  resources :comments, only: [:create, :destroy]
end
```

Προστέθηκε η διαδρομή destroy για τα σχόλια, η οποία επιτρέπει τη χρήση της μεθόδου **destroy** στον **CommentsController**. Αυτή η διαδρομή εξασφαλίζει ότι η διαγραφή σχολίων γίνεται μέσω της σωστής ενέργειας και στο πλαίσιο της φωτογραφίας (**photo_comment_path**).

## Views: users/show.html.haml

```
- if photo.comments.any?
  - photo.comments.each do |comment|
    %p
      %strong= comment.user.email
      = comment.text
      - if current_user == comment.user || current_user == photo.user
        = button_to 'Delete', photo_comment_path(photo, comment), method: :delete, data: { confirm: 'Are you sure?' }, class: ['btn', 'btn-danger', 'btn-sm']
```

Προστέθηκε κουμπί **Delete** δίπλα σε κάθε σχόλιο, ορατό μόνο εάν ο τρέχων χρήστης είναι ο δημιουργός του σχολίου ή ο ιδιοκτήτης της φωτογραφίας. Το **button_to** χρησιμοποιεί τη μέθοδο **DELETE** με μήνυμα επιβεβαίωσης (**Are you sure?**), επιτρέποντας ασφαλή διαχείριση σχολίων.

Controllers: comments_controller.rb

```
def destroy
  @comment = Comment.find(params[:id])
  @photo = @comment.photo

  # Ελέγχουμε αν ο χρήστης έχει δικαίωμα διαγραφής
  if current_user == @comment.user || current_user == @photo.user
    @comment.destroy
    flash[:notice] = "Comment deleted successfully!"
  else
    flash[:alert] = "You are not authorized to delete this comment."
  end
  redirect_to user_path(@photo.user)
end
```

Η μέθοδος **destroy** προστέθηκε για να διαχειριστεί τη διαγραφή σχολίων. Περιλαμβάνει έλεγχο δικαιωμάτων όπως τη διαγραφή αν ο τρέχων χρήστης είναι είτε ο δημιουργός του σχολίου (**@comment.user**) είτε ο ιδιοκτήτης της φωτογραφίας (**@photo.user**). Εάν ο χρήστης δεν έχει τα απαραίτητα δικαιώματα, εμφανίζεται κατάλληλο μήνυμα σφάλματος. Μετά τη διαγραφή, γίνεται ανακατεύθυνση στη σελίδα του χρήστη της φωτογραφίας.

## 2.5 Υλοποίηση του bonus ερωτήματος (5)

Στο 5ο ερώτημα, υλοποιήσαμε τη δυνατότητα διαγραφής φωτογραφιών από τον ιδιοκτήτη τους, μαζί με όλα τα σχόλια και τις ετικέτες που σχετίζονται με αυτές. Προσθέσαμε τη μέθοδο destroy στον PhotosController, με έλεγχο ώστε μόνο ο χρήστης που ανέβασε τη φωτογραφία να μπορεί να τη διαγράψει. Στο view (users/show.html.haml), προσθέσαμε κουμπί διαγραφής δίπλα σε κάθε φωτογραφία που εμφανίζεται μόνο αν ο τρέχων χρήστης είναι ο ιδιοκτήτης. Ενημερώσαμε επίσης το routes.rb για να περιλαμβάνει τη διαδρομή destroy για τις φωτογραφίες.

Πιο συγκεκριμένα, οι αλλαγές ήταν οι εξής:

### Views: users/show.html.haml

```
- if current_user == photo.user

  = button_to 'Delete Photo', photo_path(photo), method: :delete, data: { confirm: 'Are you sure? This will delete the photo and all its comments and tags.' }, class: ['btn', 'btn-danger', 'btn-sm']
```

Προστέθηκε κουμπί **Delete Photo** δίπλα σε κάθε φωτογραφία, ορατό μόνο αν ο τρέχων χρήστης είναι ο ιδιοκτήτης της φωτογραφίας. Το κουμπί χρησιμοποιεί τη μέθοδο **DELETE** για να καλέσει τη μέθοδο **destroy** στον **PhotosController**. Επιπλέον, εμφανίζεται μήνυμα επιβεβαίωσης πριν τη διαγραφή, διασφαλίζοντας ότι ο χρήστης γνωρίζει ότι θα διαγραφούν και όλα τα σχόλια και οι ετικέτες της φωτογραφίας.

### Config: routes.rb

```
resources :photos, only: [:destroy] do

  resources :comments, only: [:create, :destroy]

end
```

Η διαδρομή **destroy** για τις φωτογραφίες προστέθηκε ώστε να ενεργοποιηθεί η λειτουργία διαγραφής φωτογραφιών. Αυτό επιτρέπει την κλήση της μεθόδου **destroy** στον **PhotosController** μέσω της διαδρομής **photo_path(photo)**. Με αυτήν τη ρύθμιση, διασφαλίζεται ότι η διαγραφή των φωτογραφιών είναι συμβατή με το **RESTful** πρότυπο του Rails. Επιπλέον, η διαδρομή συνδέεται με τη μέθοδο **DELETE**, επιτρέποντας ασφαλή και συγκεκριμένη αλληλεπίδραση μέσω του κουμπιού διαγραφής.

Controllers: photos_controller.rb

```
def destroy
  @photo = Photo.find(params[:id])

  # Ελέγχουμε αν ο χρήστης είναι ο ιδιοκτήτης της φωτογραφίας
  if current_user == @photo.user
    @photo.destroy
    flash[:notice] = "Photo and all related comments and tags were successfully deleted!"
  else
    flash[:alert] = "You are not authorized to delete this photo."
  end
  redirect_to user_path(current_user)
end
```

Προστέθηκε η μέθοδος **destroy** στον **PhotosController**, επιτρέποντας στον ιδιοκτήτη μιας φωτογραφίας να τη διαγράψει. Η διαγραφή περιλαμβάνει επίσης όλα τα σχετικά σχόλια και ετικέτες, εξαιτίας της ρύθμισης **dependent: :destroy** στο μοντέλο **Photo**. Αν ο χρήστης που προσπαθεί να διαγράψει τη φωτογραφία δεν είναι ο ιδιοκτήτης, εμφανίζεται κατάλληλο μήνυμα σφάλματος.