ΜΥΕ042 Τεχνολογίες Διαδικτύου

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

Όμηρος Χατζηιορδάνης ΑΜ: 5388

Περιεχόμενα

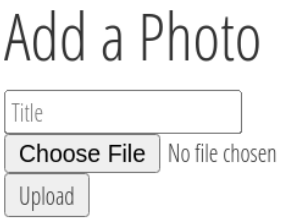
1. Προσθήκη τίτλου σε φωτογραφίες

Η πρόσθεση τίτλου στις φωτογραφίες υλοποιήθηκε με τη χρήση ενός αρχείου migrate όπου γίνεται πρόσθεση της στήλης title τύπου string, στο πίνακα των φωτογραφιών.

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

```ruby
  create_table "photos", force: :cascade do |t|
    t.integer  "user_id"
    t.string   "caption"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.string   "image_file_name"
    t.string   "image_content_type"
    t.integer  "image_file_size",    limit: 8
    t.datetime "image_updated_at"
    t.string   "title"
  end
```

Στη φόρμα έγινε η προσθήκη ενός text_field το οποίο περνάει τη πληροφορία του τίτλου στη νέα πλειάδα των φωτογραφιών.

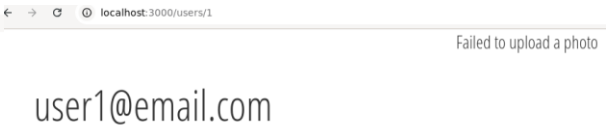


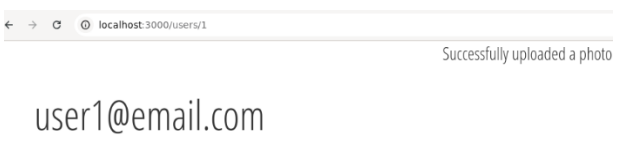

```
%h1 Add a Photo
= form_for Photo.new(), :url => user_photos_path, :html => {:multipart => true} do |form|
  = form.text_field :title, placeholder: "Title"
  = form.file_field :image
  = form.submit 'Upload'
```

Μέσα στο μοντέλο των φωτογραφιών προστέθηκε και η αναγκαία ύπαρξη του τίτλου σε μία φωτογραφία, ώστε ο χρήστης να μην μπορεί να ανεβάζει μόνες τις φωτογραφίες. Έτσι στον controller των φωτογραφιών υπάρχουν τα κατάλληλα μηνύματα αν ο χρήστης συμπλήρωσε σωστά τη φόρμα ή όχι.

```ruby

  validates :title, presence: true
end
```

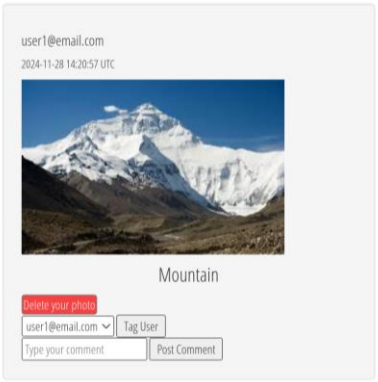
```ruby
class PhotosController < ApplicationController
  def create
    @user = User.find(params[:user_id])
    if params[:photo] == nil

      flash[:alert] = "Please upload a photo"
      redirect_to :back
    else
    @photo = Photo.create(photo_params)
      @photo.user_id = @user.id
      @photo.save
      if photo_params[:title] == ""
        flash[:notice] = "Failed to upload a photo"
      else
        flash[:notice] = "Successfully uploaded a photo"
      end
      redirect_to user_path(@user)
    end
  end
```





user1@email.com

Στη σελίδα προβολής του χρήστη, στις φωτογραφίες πλέον εμφανίζεται ο τίτλος ακριβώς κάτω από την εικόνα με μερικές μορφοποιήσεις της css για το μέγεθος και τη στοίχιση.



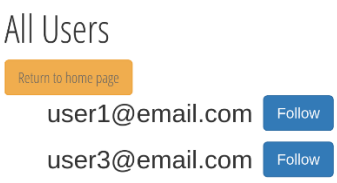
---

2. Λειτουργία ακολούθησης ενός χρήστη

Στην σελίδα προβολής του χρήστη έχει προστεθεί στη κορυφή ένα κουμπί το οποίο μεταφέρει τον τωρινό χρήστη (στο παράδειγμα ο χρήστης user2) σε μια νέα σελίδα όπου εμφανίζονται τα email όλων των υπόλοιπων χρηστών που υπάρχουν στη βάση δεδομένων. Στη σελίδα αυτή υπάρχει ένα κουμπί επιστροφής στο home page και δίπλα σε κάθε email ενός χρήστη εμφανίζεται ένα κουμπί follow ώστε να δημιουργηθεί η σχέση του ακολουθημένου και του ακολουθούμενου μεταξύ των χρηστών.

```
<h1>All Users</h1>

.row
  .col-sm-5
    = link_to 'Return to home page', user_path(current_user), class: ["btn", "btn-warning", 'btn-return']

%ul
  - @users.each do |user|
    - if user != current_user
      %li{style: "display: flex; align-items: center; gap: 10px; margin-bottom: 10px;"}
        %p{style: "margin: 0; font-size: 24px;"}= user.email
        = button_to 'Follow', follow_user_path(user.id), method: :post, class: 'btn btn-primary'
```

```
.row.top_row
  .col-sm-6.user_att
    %h2= @user.email
    - if @user.avatar_file_name
      = image_tag @user.avatar.url(:thumb)
  .col-sm-6
    = link_to 'Logout', log_out_path, class: ['btn', 'btn-danger', 'logout_btn']
  .row
    = link_to 'Add Photo', new_user_photo_path(@user), class: ['btn', 'btn-success', 'add_photo_btn']
    = link_to 'Show Users', view_all_users_users_path(@user), class: ['btn', 'btn-info', 'view_all_users_btn']
%br/
```



Ο πίνακας που δημιουργήθηκε για τη λειτουργία αυτή στη βάση δεδομένων, πρέπει να περιέχει μόνο δύο πεδία, έναν χρήστη που ακολουθεί και έναν χρήστη τον οποίο ακολουθούν, εφόσον οι λειτουργίες της εφαρμογής βασίζονται σε ποιους χρήστες ακολουθεί αυτός που χρησιμοποιεί την εφαρμογή εκείνη τη στιγμή, οπότε είναι χρήσιμο να υπάρχει διαφοροποίηση μεταξύ των δύο χρηστών.

Τα δύο αυτά πεδία πρέπει να αντιστοιχούν σε χρήστες που ήδη υπάρχουν, οπότε στο αρχείο migrate δημιουργίας χρησιμοποιείται ο τύπος references ώστε τα πεδία να αντιστοιχούν στον πίνακα των χρηστών. Μαζί με τον τύπο αυτών των πεδίων υπάρχει και η δήλωση foreign key, το οποίο σιγουρεύει την ανάθεση μιας υπάρχουσας τιμής μέσα στον πίνακα των χρηστών. Τέλος στο μοντέλο των ακολουθιών δηλώνεται ότι και τα δύο πεδία ανήκουν στον πίνακα των χρηστών.

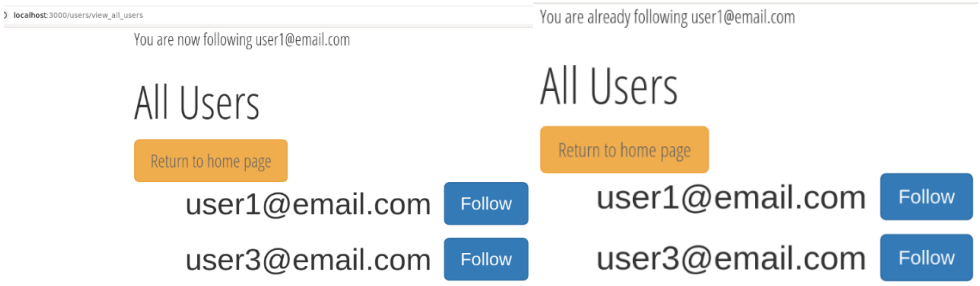
```
  create_table "follows", force: :cascade do |t|
    t.integer "follower_id"
    t.integer "followed_id"
  end
```

```
class CreateFollows < ActiveRecord::Migration
  def change
    create_table :follows do |t|
      t.references :follower, foreign_key: true
      t.references :followed, foreign_key: true
    end
  end
end
```

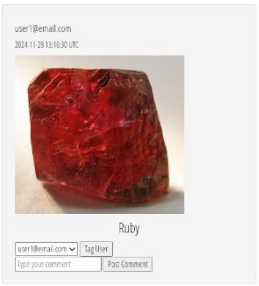
```
class Follow < ActiveRecord::Base
    belongs_to :follower, class_name: 'User', foreign_key: 'follower_id'
    belongs_to :followed, class_name: 'User', foreign_key: 'followed_id'
end
```

Για την δημιουργία των ακολουθιών δημιουργήθηκε μια νέα μέθοδος στον user controller με όνομα follow, η οποία ελέγχει την ύπαρξη της ακολούθησης που πάνε δημιουργηθεί, ψάχνοντας την στον πίνακα Follow και αναθέτοντας την πλειάδα που ταιριάζει σε μια μεταβλητή. Αν η μεταβλητή είναι κενή, τότε δημιουργείται μια νέα πλειάδα με το id του τωρινού χρήστη και το id του άλλου χρήστη που στάλθηκε σαν παράμετρος από το αρχείο haml με μήνυμα επιτυχίας. Αλλιώς δίνεται μήνυμα που αναγράφεται ότι ήδη υπάρχει αυτή η σχέση. Επίσης για τη λειτουργία της σελίδας προβολής των χρηστών δημιουργήθηκε η μέθοδος view_all_users που απλά περνάει σε μια παράμετρο όλους τους χρήστες που υπάρχουν στη βάση εκείνη τη στιγμή.

```
  def view_all_users
    @users = User.all
  end

  def follow
    user_to_follow = User.find(params[:id])
    existing_follow = Follow.find_by(follower_id: current_user.id, followed_id: params[:id])

    if existing_follow.nil? # Only create follow if it doesn't exist
      Follow.create(follower_id: current_user.id, followed_id: params[:id])
      flash[:notice] = "You are now following #{user_to_follow.email}"
    else
      flash[:alert] = "You are already following #{user_to_follow.email}"
    end
    redirect_to view_all_users_users_path
  end
```



Στην αρχική σελίδα του χρήστη πλέον εμφανίζονται οι φωτογραφίες και του ίδιου και των χρηστών που ακολουθεί με βάση την ημερομηνία δημιουργίας, όπου πρώτα εμφανίζονται οι νεότερες φωτογραφίες. Επίσης προστέθηκε στην εμφάνιση της φωτογραφίας και το όνομα του χρήστη και η ώρα, και για ευκολία του χρήστη και για τη δοκιμή της σωστής λειτουργίας της εφαρμογής, και η εμφάνιση πλέον των φωτογραφιών γίνεται κάθετα για πιο ξεκάθαρη παρουσίαση της χρονολογικής σειράς.



Για την υλοποίηση αυτή η μέθοδος show τροποποιήθηκε ώστε να δίνει στο αρχείο haml αντί μόνο τον χρήστη, όλες τις φωτογραφίες που πρέπει να προβληθούν. Στη μεταβλητή photos πρώτα αναθέτονται όλες οι φωτογραφίες του τωρινού χρήστη, στη συνέχεια προθέτονται μέσω της συνένωσης αυτών που ακολουθούνται οι λοιπές φωτογραφίες και στο τέλος γίνεται η αντίστροφη ταξινόμηση τους με βάση την ημερομηνία. Επίσης για την συνένωση, προστέθηκε στο μοντέλο των χρηστών συσχέτιση για τις ακολουθίες αλλά και μία έμμεση συσχέτιση για τους άλλους χρήστες που ακολουθεί ο χρήστης για πιο εύκολη πρόσβαση στις φωτογραφίες τους στη μέθοδο show. Τέλος στο αρχείο haml έγινε αλλαγή στη μεταβλητή από όπου γινόταν το each και ξεκίναγε η επανάληψη, χρησιμοποιώντας τώρα το photos, όπως φαίνεται σε παραπάνω εικόνα.

```ruby
  def show
    @users = User.all
    @user = User.find(params[:id])
    @tag = Tag.new
    @comment = Comment.new
    @photos = current_user.photos

    # Get photos from users that the current user follows
    follows = current_user.followed_users
    follows.each do |followed_user|
      @photos += followed_user.photos
    end
    @photos = @photos.sort_by(&:created_at).reverse
  end
```

```ruby
class User<ActiveRecord::Base
  has_many :photos
  has_many :tags
  has_many :comments

  has_many :followings, class_name: 'Follow', foreign_key: 'follower_id'
  has_many :followed_users, through: :followings, source: :followed
```

Οι καινούργιες μέθοδοι καταγράφηκαν ως διαδρομές στο αρχείο routes όπως παρουσιάζεται παρακάτω, με την view all users να είναι τύπου get και η follow τύπου post λόγω της δημιουργίας νέων πλειάδων.

```ruby
  resources :users do
    collection do
      get 'view_all_users'
    end
    resources :photos
  end

  resources :users do
    member do
      post 'follow'
    end
  end
```

3. Σχόλια στις φωτογραφίες

Για τα σχόλια δημιουργήθηκε νέο αρχείο migrate το οποίο δημιουργεί τον νέο πίνακα με πεδία την αναφορά της φωτογραφίας, του χρήστη και το κείμενο του σχολίου. Στο καινούργιο μοντέλο υπάρχει πάλι αναφορά σε ποιους πίνακες ανήκουν τα πεδία και στα μοντέλα των φωτογραφιών και των χρηστών έχει γίνει πρόσθεση συσχέτισης με τα σχόλια για ευκολότερη πρόσβαση σε αυτά σε συνενώσεις.

```ruby
class CreateComment < ActiveRecord::Migration
  def change
    create_table :comments do |t|
      t.references :photo, foreign_key: true
      t.references :user, foreign_key: true
      t.text :comment_text
    end
  end
end
```

```
  create_table "comments", force: :cascade do |t|
    t.integer "photo_id"
    t.integer "user_id"
    t.text    "comment_text"
  end
```

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

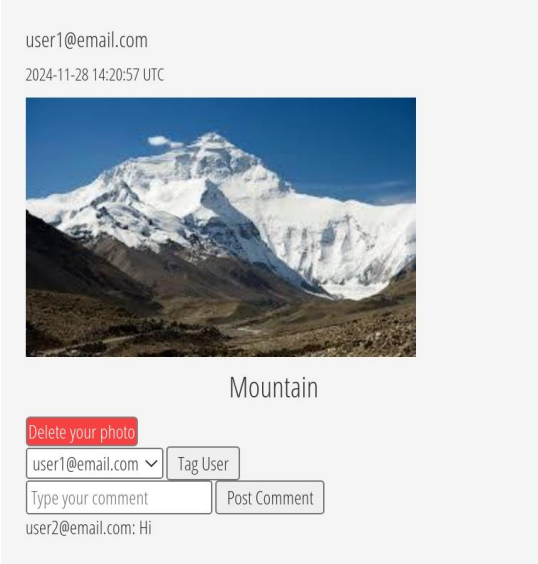
```
class Photo<ActiveRecord::Base
  belongs_to :user
  has_many :tags, dependent: :destroy
  has_many :comments, dependent: :destroy
```

Για τη δημιουργία των σχολίων ακολουθήθηκε παρόμοια υλοποίηση με τα tags. Σε κάθε φωτογραφία που εμφανίζεται στην αρχική σελίδα ενός χρήστη υπάρχει μια νέα φόρμα στην οποία υπάρχει ένα φανερό text field και κρυμμένα fields για τα αναγνωριστικά του χρήστη και της φωτογραφίας, η οποία θα περάσει τα δεδομένα αυτά στη μέθοδο create στον controller των σχολίων. Η μέθοδος αυτή απλά παίρνει της παραμέτρους που έχουν δημιουργηθεί από τη φόρμα και δημιουργεί ένα νέο σχόλιο.

```
      = form_for @comment do |c|
        = c.hidden_field :photo_id, value: photo.id
        = c.hidden_field :user_id, value: current_user.id
        = c.text_field :comment_text, placeholder: "Type your comment"
        = c.submit "Post Comment"
      - photo.comments.each do |comment|
        .comment-row{style: "display: flex; align-items: center; gap: 15px;"}
          %span.comment-text= "#{User.find(comment.user_id).email}: #{comment.comment_text}"
          - if current_user.id == comment.user_id
            = button_to "Delete your comment", comment_path(comment), method: :delete, data: { confirm: "Are you sure?" }, style: "background-color:
            #f54242; color: white; border: 1px solid grey; border-radius: 3px; padding: 0.5px 1px;"
```

```
class CommentsController < ApplicationController
    def new
    end

    def create
        @new_comment = Comment.create({ photo_id: params[:comment][:photo_id], user_id: params[:comment][:user_id], comment_text: params[:comment][:comment_text] })
        redirect_to user_path(session[:user_id])
    end
```

Η προβολή του κάθε σχόλιου μιας φωτογραφίας γίνεται κάτω ακριβώς από τη φόρμα δημιουργίας με το κείμενο να εμφανίζεται δεξιά από το email του σχολιαστή.



4. Διαγραφή σχολίων

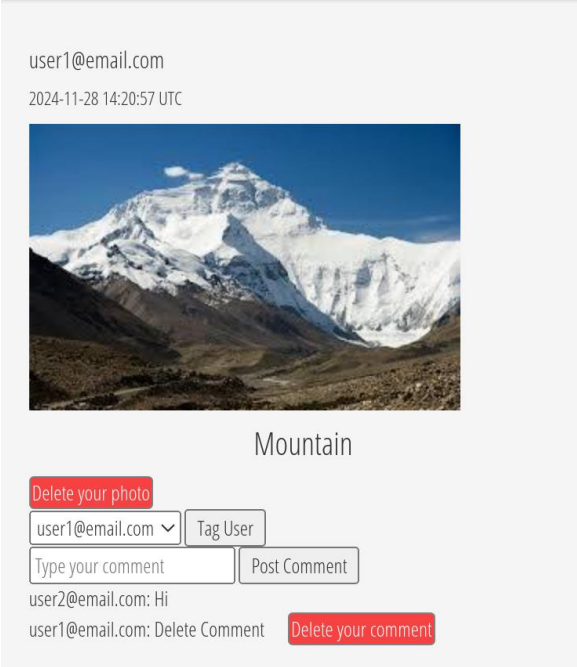
Για τη διαγραφή των σχολίων προστέθηκε και μια μέθοδος delete στον controller όπου η μέθοδος βρίσκει το σχόλιο που θα διαγράψει μέσω του γενικού id, το καταστρέφει και μετά επιστρέφει στο session του χρήστη.

```
10      def destroy
11          @comment_to_delete = Comment.find(params[:id])
12          @comment_to_delete.destroy
13          redirect_to user_path(session[:user_id])
14      end
15  end
```

Η μέθοδος αυτή γίνεται προσπελάσιμη μέσω ενός κουμπιού το οποίο εμφανίζεται δεξιά από το κείμενο ενός σχολίου, μόνο αν ο τωρινός χρήστης στην εφαρμογή είναι και ο σχολιαστής.

```
45          %span.comment-text= "#{User.find(comment.user_id).email}: #{comment.comment_text}"
46          - if current_user.id == comment.user_id
47            = button_to "Delete your comment", comment_path(comment), method: :delete, data: { confirm: "Are you sure?" }, style: "background-color:
↪           #f54242; color: white; border: 1px solid grey; border-radius: 3px; padding: 0.5px 1px;"
```

Για τον έλεγχο της λειτουργίας της διαγραφής εκτός από την εξαφάνιση του σχολίου στη σελίδα, γίνεται εμφανής και από την έλλειψη της πλειάδας στη βάση δεδομένων.




```
  Comment Load (1.1ms)  SELECT "comments".* FROM "comments"
=> #<ActiveRecord::Relation [#<Comment id: 1, photo_id: 4, user_id: 2, comment
_text: "Hi">, #<Comment id: 3, photo_id: 4, user_id: 1, comment_text: "Delete
Comment">]>
```

```
  Comment Load (0.1ms)  SELECT "comments".* FROM "comments"
=> #<ActiveRecord::Relation [#<Comment id: 1, photo_id: 4, user_id: 2, comment
_text: "Hi">]>
```

Φυσικά η μέθοδος αυτή όπως και η create, έχουν οριστεί με παρόμοιο τρόπο με αυτές των tags στο αρχείο των διαδρομών.

```
23    resources :comments, only: [:create, :destroy]
24
25    resources :tags, only: [:create, :destroy]
```

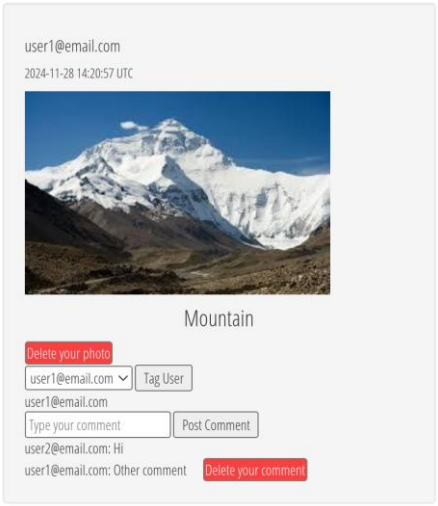
5. Διαγραφή φωτογραφιών

Για τις φωτογραφίες δημιουργήθηκε μια νέα διαδρομή destroy τύπου delete. Η μέθοδος που υλοποιήθηκε εκτελεί μόνο τη καταστροφή της συγκεκριμένης πλειάδας της φωτογραφίας. Τα σχόλια και τα tags διαγράφονται αυτόματα λόγω της προσθήκης στην συσχέτιση που υπάρχει στο μοντέλο των φωτογραφιών μέσου του dependent destroy, το οποίο φαίνεται σε παραπάνω εικόνες.

```
26    def destroy
27        @photo_to_delete = Photo.find(params[:id])
28        @photo_to_delete.destroy
29        redirect_to user_path(session[:user_id])
30    end
```

```ruby
16    resources :photos do
17      member do
18        delete 'destroy'
19      end
20    end
```

Για δοκιμή ελέγχθηκαν οι πλειάδες στην κονσόλα του rails, παρακολουθώντας μια συγκεκριμένη φωτογραφία και το σχόλιο και το tags της.




```
  Photo Load (0.1ms)  SELECT "photos".* FROM "photos"
=> #<ActiveRecord::Relation [#<Photo id: 1, user_id: 1, caption: nil, created_at: "2024-11-28 13:16:30", updated
_at: "2024-11-28 13:16:30", image_file_name: "Corundum-215330.jpg", image_content_type: "image/jpeg", image_file
_size: 16027, image_updated_at: "2024-11-28 13:16:30", title: "Ruby">, #<Photo id: 3, user_id: 2, caption: nil,
created_at: "2024-11-28 14:20:22", updated_at: "2024-11-28 14:20:22", image_file_name: "sheep-eating-grass.jpg",
 image_content_type: "image/jpeg", image_file_size: 27626, image_updated_at: "2024-11-28 14:20:22", title: "Mpe"
>, #<Photo id: 4, user_id: 1, caption: nil, created_at: "2024-11-28 14:20:57", updated_at: "2024-11-28 14:20:57"
, image_file_name: "images_(1).jpeg", image_content_type: "image/jpeg", image_file_size: 10031, image_updated_at
: "2024-11-28 14:20:57", title: "Mountain">, #<Photo id: 5, user_id: 3, caption: nil, created_at: "2024-11-30 15
:55:39", updated_at: "2024-11-30 15:55:39", image_file_name: "Logan_Rock_Treen_closeup.jpg", image_content_type:
 "image/jpeg", image_file_size: 1121581, image_updated_at: "2024-11-30 15:55:39", title: "rock">]>
```

```
  Tag Load (0.1ms)  SELECT "tags".* FROM "tags"
=> #<ActiveRecord::Relation [#<Tag id: 1, user_id: 1, photo_id: 4>]>
```

```
  Comment Load (0.1ms)  SELECT "comments".* FROM "comments"
=> #<ActiveRecord::Relation [#<Comment id: 1, photo_id: 4, user_id: 2, comment_text: "Hi">, #<Comment id: 4, pho
to_id: 4, user_id: 1, comment_text: "Other comment">]>
```

Μετά τη διαγραφή:

```
  Photo Load (0.1ms)  SELECT "photos".* FROM "photos"
=> #<ActiveRecord::Relation [#<Photo id: 1, user_id: 1, caption: nil, created_at: "2024-11-28 13:16:30", updated
_at: "2024-11-28 13:16:30", image_file_name: "Corundum-215330.jpg", image_content_type: "image/jpeg", image_file
_size: 16027, image_updated_at: "2024-11-28 13:16:30", title: "Ruby">, #<Photo id: 3, user_id: 2, caption: nil,
created_at: "2024-11-28 14:20:22", updated_at: "2024-11-28 14:20:22", image_file_name: "sheep-eating-grass.jpg",
 image_content_type: "image/jpeg", image_file_size: 27626, image_updated_at: "2024-11-28 14:20:22", title: "Mpe"
>, #<Photo id: 5, user_id: 3, caption: nil, created_at: "2024-11-30 15:55:39", updated_at: "2024-11-30 15:55:39"
, image_file_name: "Logan_Rock_Treen_closeup.jpg", image_content_type: "image/jpeg", image_file_size: 1121581, i
mage_updated_at: "2024-11-30 15:55:39", title: "rock">]>
```

```
  Tag Load (0.1ms)  SELECT "tags".* FROM "tags"
=> #<ActiveRecord::Relation []>
```

```
  Comment Load (0.2ms)  SELECT "comments".* FROM "comments"
=> #<ActiveRecord::Relation []>
```

6. Παραδοτέα αρχεία

Στη παράδοση έχει δοθεί ένα directory treegram το οποίο περιέχει όλα τα καινούργια και τροποποιημένα αρχεία σε κατάλληλη μορφή φακέλων όπως βρίσκονται και στην εικονική μηχανή. Στον φάκελο της βάσης δεδομένων δίνονται τα migrate, το scema και το sql script με το οποίο δημιουργήθηκαν τα παραδείγματα στις εικόνες, και άλλες δοκιμές. Έχει παρατηρηθεί ότι αν τοποθετηθεί το sql script, η εφαρμογή μπορεί να τρέξει απλά με την εντολή server, αν δεν τοποθετηθεί θα είναι απαραίτητη η εκτέλεση του rake db:migrate σκέτο ή με το πρόθεμα bundle exec. Άλλες ιδιαιτερότητες δεν έχουν παρατηρηθεί ως προς το πέρασμα αυτής της υλοποίησης σε νέα μηχανή. Τέλος έχουν δοθεί και οι φωτογραφίες που χρησιμοποιήθηκαν, και αν επιθυμείτε να τρέξετε τη βάση που έχει χρησιμοποιηθεί ως παράδειγμα, οι χρήστες έχουν email της μορφής userX@email.com, όπου X ο αριθμός (υπάρχουν μόνο τρεις και η αρίθμηση ξεκινάει από το 1), με κοινό κωδικό user.