ΤΕΧΝΟΛΟΓΙΕΣ ΔΙΑΔΙΚΤΥΟΥ

Προβολή παρουσίασης και αναδυόμενο παράθυρο στην εφαρμογή φωτογραφιών.

Team

Γκόβαρης Χρήστος-Γρηγόριος
Σπανού Μαρία

Χρονοδιάγραμμα 2ης Εργαστηριακής Άσκησης

Έκδοση	Ημερομηνία	Progress
1.0	29/11/2024	Ανακοίνωση 2ης εργαστηριακής άσκησης
2.0	16/12 - 18/12	Υλοποίηση του ερωτήματος (1)
2.1	19/12/2024	Υλοποίηση του ερωτήματος (2)
2.2	20/12/2024	Υλοποίηση του bonus ερωτήματος (3)
3.0	16/12 – 20/12	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**.

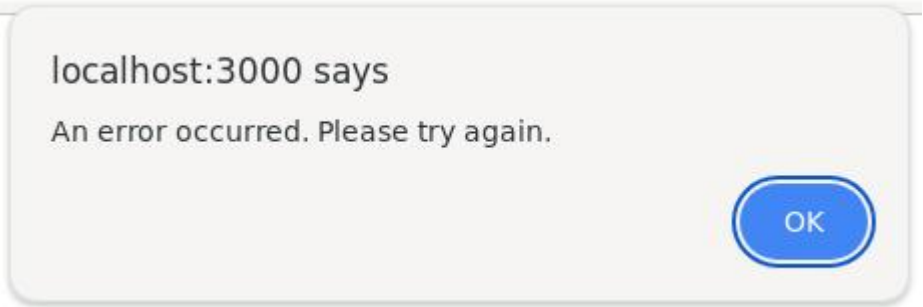
Αρχεία που τροποποιήθηκαν

Τροποποιήθηκαν τα παρακάτω αρχεία:

Controllers	Views	Routes	Assets
app/controllers/users_controller.rb	app/views/users/show.html.haml	config/routes.rb	app/assets/javascripts/application.js
app/controllers/photos_controller.rb	app/views/photos/show.html.haml		app/assets/javascripts/photos.js
app/controllers/comments_controller.rb	app/views/photos/comments.js.erb		app/assets/stylesheets/application.sass
	app/views/photos/comments.html.haml		
	app/views/comments/_comments.html.erb		
	app/views/comments/create.js.erb		
	app/views/comments/destroy.js.erb		

Παρατηρήσεις

Παρατηρείται ότι κατά την προσπάθεια διαγραφής μιας φωτογραφίας του συνδεδεμένου χρήστη μέσω διπλού κλικ, εμφανίζεται το παρακάτω μήνυμα σφάλματος:



Για να πραγματοποιηθεί η διαγραφή της φωτογραφίας, απαιτείται η ανανέωση (**reload**) της σελίδας.

Επιπλέον, όταν προστίθεται ένα σχόλιο στις φωτογραφίες των ακολουθούμενων χρηστών, απαιτείται ανανέωση (**reload**) της σελίδας για να εμφανιστούν τα σχόλια κάτω από τις φωτογραφίες.

Τα σχόλια που καταχωρούνται μέσω του **modal** δεν αποθηκεύονται επιτυχώς.

2.0 Υλοποίηση του ερωτήματος (1)

Για το 1ο ερώτημα, υλοποιήσαμε τη δυνατότητα προβολής παρουσίασης (**slideshow**) φωτογραφιών για τους ακολουθούμενους χρήστες. Οι φωτογραφίες οργανώνονται ανά χρήστη, με την πιο πρόσφατη να εμφανίζεται πρώτη. Όταν ο χρήστης μετακινήσει τον δείκτη πάνω σε μια φωτογραφία, το **slideshow** ξεκινά και προβάλλει τις φωτογραφίες σε αντίστροφη χρονολογική σειρά. Η λειτουργία σταματά όταν ο δείκτης απομακρυνθεί. Ο τίτλος κάθε φωτογραφίας εμφανίζεται κάτω από την εικόνα, ενώ τα σχόλια αποκλείονται από το **slideshow**. Χρησιμοποιήθηκε **JavaScript** (**photo.js**) για την εναλλαγή και διακοπή των εικόνων, καθώς και **HAML** για τη δομή εμφάνισης των φωτογραφιών. Πιο συγκεκριμένα, οι αλλαγές ήταν οι εξής:

app/assets/javascripts/application.js

```
//= require photos
```

Προστέθηκε για να διασφαλίσουμε τη λειτουργικότητα του **slideshow** και την εμφάνιση των τίτλων για κάθε φωτογραφία.

app/assets/javascripts/photos.js

```
function showSlides(n, container) {
    const slides = container.querySelectorAll(".mySlides");
    const titles = container.querySelectorAll(".photo-title");
    if (slides.length === 0) return;
    let slideIndex = parseInt(container.getAttribute("data-slide-index")) || 0;
    slideIndex = (slideIndex + n + slides.length) % slides.length;
    slides.forEach((slide, i) => {
        slide.style.display = i === slideIndex ? "block" : "none";
    });
    titles.forEach((title, i) => {
        title.style.display = i === slideIndex ? "block" : "none";
    });
    container.setAttribute("data-slide-index", slideIndex);
}
```

Διαχειρίζεται την εναλλαγή προβολών εικόνων και τίτλων σε ένα συγκεκριμένο **container**. Μέσα στο **container**, τα στοιχεία με την κλάση **.mySlides** (για τις εικόνες) και **.photo-title** (για τους τίτλους) επιλέγονται για την εμφάνιση ή την απόκρυψή τους.

Ο δείκτης **slideIndex**, που αποθηκεύεται ως **attribute** του **container**, χρησιμοποιείται για να προσδιορίσει ποιο **slide** θα εμφανιστεί. Με την κυκλική λογική (**slideIndex + n + slides.length**) **% slides.length**, διασφαλίζεται ότι η εναλλαγή λειτουργεί χωρίς σφάλματα. Τα στοιχεία που αντιστοιχούν στο τρέχον **index** (**i === slideIndex**) εμφανίζονται (**display: block**), ενώ όλα τα υπόλοιπα κρύβονται (**display: none**).

app/assets/javascripts/photos.js

```
function startAutoSlide(container) {
  if (container.autoSlideTimer) return;
  const comments = container.querySelector(".photo-comments");
  if (comments) comments.style.display = "none";
  container.autoSlideTimer = setInterval(() => {
    showSlides(1, container);
  }, 3000);
}
```

ενεργοποιεί την αυτόματη εναλλαγή των **slides** σε ένα συγκεκριμένο **container**. Εάν υπάρχει ήδη ενεργοποιημένος **timer** (**container.autoSlideTimer**), η λειτουργία διακόπτεται για να αποφευχθεί η διπλή εκκίνηση. Αν το **container** περιέχει στοιχεία με την κλάση **.photo-comments** (σχόλια), αυτά αποκρύπτονται (**display: none**) όταν ξεκινά το **slideshow**. Στη συνέχεια, δημιουργείται ένας **timer** μέσω **setInterval**, ο οποίος καλεί τη συνάρτηση **showSlides** με τιμή 1 κάθε 3 δευτερόλεπτα, εναλλάσσοντας έτσι τα **slides** αυτόματα.

app/assets/javascripts/photos.js

```
function stopAutoSlide(container) {
  clearInterval(container.autoSlideTimer);
  container.autoSlideTimer = null;
  const comments = container.querySelector(".photo-comments");
  if (comments) comments.style.display = "block"; // Show comments when slideshow stops
  const initialIndex = parseInt(container.getAttribute("data-initial-index")) || 0;
  container.setAttribute("data-slide-index", initialIndex);
  showSlides(0, container);
}
```

Σταματά την αυτόματη εναλλαγή των **slides** σε ένα συγκεκριμένο **container**. Αρχικά, διακόπτει τον ενεργό **timer** (**container.autoSlideTimer**) χρησιμοποιώντας **clearInterval**

και τον θέτει σε **null** για να υποδείξει ότι δεν υπάρχει ενεργή αυτόματη εναλλαγή. Εάν το **container** περιέχει στοιχεία με την κλάση **.photo-comments** (σχόλια), αυτά εμφανίζονται ξανά (**display: block**) όταν σταματά το **slideshow**. Στη συνέχεια, ο δείκτης των **slides** (**data-slide-index**) επαναφέρεται στην αρχική του τιμή (**data-initial-index**), εάν έχει οριστεί, ή στο 0. Τέλος, καλείται η συνάρτηση **showSlides** με παράμετρο 0, ώστε να εμφανιστεί το αρχικό **slide**.

app/assets/javascripts/photos.js

```
document.addEventListener("DOMContentLoaded", () => {
  const slideshowContainers = document.querySelectorAll(".slideshow-container");
  slideshowContainers.forEach((container) => {
    const initialIndex = parseInt(container.getAttribute("data-slide-index")) || 0;
    container.setAttribute("data-slide-index", initialIndex);
    container.setAttribute("data-initial-index", initialIndex);
    showSlides(0, container);
    container.addEventListener("mouseenter", () => startAutoSlide(container));
    container.addEventListener("mouseleave", () => stopAutoSlide(container));
    const comments = container.querySelector(".photo-comments");
    if (comments) {
      comments.addEventListener("mouseenter", () => {
        stopAutoSlide(container); // Stop auto-slide while hovering over comments
        comments.style.display = "block"; // Ensure comments remain visible
      });
      comments.addEventListener("mouseleave", () => {
        startAutoSlide(container); // Resume auto-slide when leaving comments
      });
    }
  });
```

Διαχειρίζεται τα **slideshows** για όλα τα **containers** με την κλάση **.slideshow-container** όταν φορτωθεί η σελίδα. Για κάθε **container**, ο αρχικός δείκτης (**data-slide-index**) ορίζεται ή επαναφέρεται, και εμφανίζεται το αρχικό **slide** μέσω της **showSlides**. Κατά την είσοδο του ποντικιού στο **container**, ξεκινά η αυτόματη εναλλαγή **slides** με τη **startAutoSlide**, ενώ κατά την έξοδο σταματά μέσω της **stopAutoSlide**. Εάν υπάρχουν σχόλια (**.photo-comments**), αυτά γίνονται ορατά και η αυτόματη εναλλαγή σταματά όταν ο δείκτης βρίσκεται πάνω τους, ενώ η λειτουργία συνεχίζεται μόλις ο δείκτης απομακρυνθεί.

app/assets/stylesheets/application.sass

```
.slideshow-container
  position: relative
  max-width: 100%
  margin: auto
  overflow: hidden
```

Το **slideshow-container** είναι υπεύθυνο για τη διάταξη και εμφάνιση των φωτογραφιών με κεντρική στοίχιση και περιορισμό υπερχείλισης.

app/assets/stylesheets/application.sass

```
.mySlides
  display: none
  text-align: center
```

Κάθε φωτογραφία (**mySlides**) παραμένει κρυφή από προεπιλογή και εμφανίζεται μόνο όταν είναι ενεργή.

app/assets/stylesheets/application.sass

```
.mySlides img
  width: 100%
  border-radius: 8px
```

Οι εικόνες (**mySlides img**) είναι responsive και έχουν στρογγυλεμένες γωνίες για καλύτερη αισθητική.

app/controllers/photos_controller.rb

```
def show
  @photo = Photo.find(params[:id])
  @followed_users_photos = current_user.followed_users.includes(:photos).map do |user|
    {
      user: user,
      photos: user.photos.order(created_at: :desc)
    }
  end
```

```
end
```

Φορτώνει τις φωτογραφίες των χρηστών που ακολουθεί ο τρέχων χρήστης, καθώς και τις πληροφορίες τους. Οι φωτογραφίες ταξινομούνται σε αντίστροφη χρονολογική σειρά (**order(created_at: :desc)**), με την πιο πρόσφατη να εμφανίζεται πρώτη. Αυτό εξυπηρετεί την ομαδοποίηση φωτογραφιών ανά ακολουθούμενο χρήστη και την ταξινόμησή τους, όπως απαιτείται από την προβολή παρουσίασης (**slideshow**) στο πρώτο ερώτημα.

app/controllers/users_controller.rb

```
def show
  @user = User.find(params[:id])
  @all_users = User.all
  @photos = @user.photos.order(created_at: :desc)
  if current_user == @user
    @followed_photos = current_user.followed_users.includes(:photos).flat_map(&:photos).sort_by(&:created_at).reverse
    @comment = Comment.new
  else
    @followed_photos = []
    @comment = Comment.new
  end
end
```

Φορτώνει και ομαδοποιεί τις φωτογραφίες του τρέχοντος χρήστη και των χρηστών που ακολουθεί. Συγκεκριμένα οι φωτογραφίες του τρέχοντος χρήστη φορτώνονται ταξινομημένες σε αντίστροφη χρονολογική σειρά (**@photos**). Αν ο χρήστης προβάλλει το προφίλ του, φορτώνονται επίσης οι φωτογραφίες των χρηστών που ακολουθεί, ταξινομημένες επίσης σε αντίστροφη χρονολογική σειρά μέσω της μεταβλητής **@followed_photos**.

app/views/photos/show.html.haml

```
%h2 Photos by Followed Users
.row
  - current_user.followed_users.each do |followed_user|
    %h3= followed_user.email
    .row
```

```
- followed_user.photos.order(created_at: :desc).each_with_index do |photo, index|
  .col-sm-4
    .slideshow-container{ data: { "slide-index" => index } }
      - followed_user.photos.order(created_at: :desc).each_with_index do |slide_photo, slide_index|
        - if slide_index == index
          .mySlides{ style: "display: block;" }
            = image_tag slide_photo.image.url(:medium), class: "photo-image", data: { photo_id: slide_photo.id, current_user: false }
            %p.photo-title
              %strong Title:
              = slide_photo.title
        - else
          .mySlides{ style: "display: none;" }
            = image_tag slide_photo.image.url(:medium), class: "photo-image", data: { photo_id: slide_photo.id, current_user: false }
            %p.photo-title
              %strong Title:
              = slide_photo.title
```

Υλοποιεί την προβολή παρουσίασης φωτογραφιών για κάθε ακολουθούμενο χρήστη του τρέχοντος χρήστη. Ο κώδικας χρησιμοποιεί τη μέθοδο **current_user.followed_users.each** για να δημιουργήσει ομάδες φωτογραφιών, ταξινομημένες σε φθίνουσα χρονολογική σειρά, και τις εμφανίζει σε ξεχωριστά **slideshow-container**. Σε κάθε **container**, η πιο πρόσφατη φωτογραφία εμφανίζεται αρχικά, ενώ οι υπόλοιπες είναι κρυφές. Το **slideshow** υποστηρίζει μετάβαση μεταξύ των φωτογραφιών μέσω **JavaScript**, χρησιμοποιώντας το **data-slide-index** για να διαχειρίζεται την τρέχουσα εικόνα. Επίσης, στο πλαίσιο του **slideshow**, εμφανίζονται μόνο οι φωτογραφίες με τους τίτλους τους, χωρίς σχόλια.

app/config/routes.rb

```
resources :users do
  resources :photos, only: [:index, :new, :create, :show]
end
```

Εντός των **users**, ορίζουμε τις βασικές ενέργειες **index**, **new**, **create**, **show** για τη διαχείριση φωτογραφιών. Η διαδρομή αυτή εξυπηρετεί την προβολή όλων των φωτογραφιών ενός χρήστη και συγκεκριμένα την προβολή παρουσίασης (**slideshow**) για τις φωτογραφίες των ακολουθούμενων χρηστών. Η διαδρομή **show**

χρησιμοποιείται για να φορτώνεται η σελίδα που περιέχει το **slideshow**. Το **show** συνεργάζεται με το **PhotosController** για την απόδοση των δεδομένων στο **view**.

2.1 Υλοποίηση του ερωτήματος (2)

Για το 2ο ερώτημα, υλοποιήσαμε τη λειτουργία αναδυόμενου παραθύρου για την εμφάνιση σχολίων μιας φωτογραφίας. Όταν ο χρήστης κάνει κλικ σε μια φωτογραφία, το **slideshow** σταματά και εμφανίζεται ένα **modal** παράθυρο που περιλαμβάνει τα τρία πιο πρόσφατα σχόλια. Στο κάτω μέρος του παραθύρου προσθέσαμε ένα πλαίσιο κειμένου (**textbox**) για την εισαγωγή νέων σχολίων και κουμπιά για την υποβολή ή το κλείσιμο του παραθύρου. Μετά το κλείσιμο, το **slideshow** συνεχίζεται αν ο δείκτης του ποντικιού παραμένει πάνω από τη φωτογραφία. Για την υλοποίηση, χρησιμοποιήθηκε **JavaScript** (**photo.js**) για τον έλεγχο του **modal** και του **slideshow**, ενώ αξιοποιήθηκε **Ajax** για τη φόρτωση και υποβολή σχολίων χωρίς ανανέωση της σελίδας. Πιο συγκεκριμένα, οι αλλαγές ήταν οι εξής:

app/assets/javascripts/photos.js

```
function openCommentModal(photoId) {
  const modal = document.getElementById("comment-modal");
  const modalComments = document.getElementById("modal-comments");
  const photoIdField = document.getElementById("photo-id-field");
  photoIdField.value = photoId;
  modalComments.innerHTML = "Loading comments...";
  fetch(`/photos/${photoId}/comments`, {
    method: "GET",
    headers: { "X-Requested-With": "XMLHttpRequest", "Content-Type": "application/json" },
  })
    .then((response) => response.json())
    .then((comments) => {
      modalComments.innerHTML = "";
      comments.forEach((comment) => {
        const commentElement = document.createElement("p");
        commentElement.innerHTML = `<strong>${comment.user.email}:</strong> ${comment.text}`;
        modalComments.appendChild(commentElement);
      });
    });
  const slideshowContainer = document.querySelector(`[data-photo-id="${photoId}"]`);
  if (slideshowContainer) stopAutoSlide(slideshowContainer);
```

```
    modal.style.display = "block";
}
```

Ανοίγει ένα **modal** για την προβολή και τη διαχείριση σχολίων μιας συγκεκριμένης φωτογραφίας. Λαμβάνει ως παράμετρο το **photoId** της φωτογραφίας, το οποίο τοποθετεί στο πεδίο **photo-id-field** του **modal**. Αρχικά, το στοιχείο **modal-comments** εμφανίζει το μήνυμα "**Loading comments...**" και στη συνέχεια πραγματοποιείται ένα αίτημα **GET** μέσω **fetch** στη διαδρομή **/photos/${photoId}/comments**. Όταν επιστρέψει η απόκριση, αντικαθίσταται το περιεχόμενο με τα σχόλια που λαμβάνονται. Για κάθε σχόλιο δημιουργείται ένα νέο στοιχείο **p**, το οποίο περιέχει το **email** του χρήστη και το κείμενο του σχολίου, και προστίθεται στο **modal**. Εάν υπάρχει **slideshow** για την αντίστοιχη φωτογραφία, η αυτόματη εναλλαγή **slides** διακόπτεται μέσω της **stopAutoSlide**. Τέλος, το **modal** εμφανίζεται με **style.display = "block",** επιτρέποντας την προβολή των σχολίων.

app/assets/stylesheets/application.sass

```
#comment-modal
  display: none
  position: fixed
  top: 50%
  left: 50%
  transform: translate(-50%, -50%)
  z-index: 1001
  background-color: white
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2)
  border-radius: 8px
  width: 20%
  height: auto
  padding: 10px
  max-height: 60vh
  overflow-y: auto
```

Το συγκεκριμένο τμήμα κώδικα διασφαλίζει ότι το modal παραμένει στο κέντρο της οθόνης, έχει ευανάγνωστο φόντο και μπορεί να προσαρμοστεί σε διάφορα μεγέθη οθόνης. Παρέχει και δυνατότητα κύλισης για περιπτώσεις πολλών σχολίων.

app/assets/stylesheets/application.sass

```
#modal-comments p {
  margin-bottom: 10px;
  font-size: 14px;
}
```

Καθορίζει το πώς εμφανίζονται τα σχόλια μέσα στο **modal**, με ευανάγνωστο μέγεθος γραμματοσειράς και διαχωρισμό ανάμεσα στα σχόλια.

app/assets/stylesheets/application.sass

```
.close-btn
  position: absolute
  top: 10px
  right: 10px
  background: red
  color: white
  border: none
  padding: 5px 10px
  cursor: pointer
  border-radius: 4px

.close-btn:hover
  background: darkred
```

Ορίζει την εμφάνιση του κουμπιού κλεισίματος του **modal**, παρέχοντας ευκολία στη χρήση με χρώματα που τραβούν την προσοχή.

app/assets/stylesheets/application.sass

```
@media (max-width: 768px)
  #comment-modal
    width: 80%
```

Εξασφαλίζει ότι το **modal** παραμένει ευανάγνωστο σε κινητές συσκευές, αυξάνοντας το πλάτος του για καλύτερη εμπειρία χρήστη.

app/controllers/photos_controller.rb

```
def comments
  @photo = Photo.find(params[:id])
  @comments = @photo.comments.includes(:user).order(created_at: :desc)
  render json: @comments.as_json(include: { user: { only: :email } })
end
```

Παρέχει τη λειτουργικότητα για τη φόρτωση των σχολίων μιας φωτογραφίας μέσω **Ajax**. Όταν ο χρήστης κάνει κλικ σε μια φωτογραφία, καλείται η ενέργεια **comments** του **PhotosController**, η οποία αναζητά τη φωτογραφία με το **id** που παρέχεται στο αίτημα. Φορτώνει τα σχόλια της φωτογραφίας, συμπεριλαμβανομένων των πληροφοριών του χρήστη που τα έκανε. Επιστρέφει τα σχόλια σε μορφή **JSON**, τα οποία μπορούν να χρησιμοποιηθούν από τον **client** (**JavaScript**) για να ενημερωθεί δυναμικά το περιεχόμενο του αναδυόμενου παραθύρου χωρίς να γίνει ανανέωση της σελίδας.

app/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
    respond_to do |format|
      format.html { redirect_to user_path(@photo.user), notice: 'Comment added successfully!' }
      format.js
    end
  else
    respond_to do |format|
      format.html { redirect_to user_path(@photo.user), alert: 'Failed to add comment.' }
      format.js { render js: "alert('Failed to add comment.');" }
    end
  end
end
def comments
  @photo = Photo.find(params[:id])
```

```
  @comments = @photo.comments.order(created_at: :desc)


  respond_to do |format|
    format.js
  end
end
```

Η μέθοδος **create** δημιουργεί ένα νέο σχόλιο για μια συγκεκριμένη φωτογραφία. Χρησιμοποιεί την παράμετρο **remote: true** στη φόρμα σχολίων, ώστε να λειτουργεί με Ajax. Αν η αποθήκευση είναι επιτυχής, επιστρέφει μια απάντηση **JavaScript** (**comments.js.erb**) που ενημερώνει το **DOM** για την προσθήκη του νέου σχολίου στο **modal**.Η μέθοδος **comments** Φέρνει τα σχόλια για μια φωτογραφία με αντίστροφη χρονολογική σειρά. Επιστρέφει τα δεδομένα μέσω **JavaScript** (**comments.js.erb**), ώστε να μπορούν να ενημερώσουν το **modal** ή άλλο τμήμα της σελίδας.

app/views/users/show.html.haml

```
%div#comment-modal{ style: "display: none;" }
  .modal-content
    %h3 Photo Comments
    %div#modal-comments
      %p Loading comments...
    %h4 Add a Comment:
    = form_tag photo_comments_path(photo_id: ""), method: :post, remote: true, html: { id: "comment-form-modal" } do
      = hidden_field_tag :photo_id, "", id: "photo-id-field"
      = text_area_tag :text, nil, placeholder: "Write your comment here...", rows: 3, class: "form-control"
      %br/
      = submit_tag "Add Comment", class: ['btn', 'btn-primary']
    %button.close-btn Close
```

Υλοποιεί τη λειτουργία του αναδυόμενου παραθύρου (**modal**) για την προβολή και διαχείριση σχολίων μιας φωτογραφίας. Το **#comment-modal** δημιουργεί ένα αναδυόμενο παράθυρο που περιλαμβάνει την περιοχή **#modal-comments**, όπου εμφανίζονται τα πρόσφατα σχόλια της φωτογραφίας, και μια φόρμα προσθήκης νέων σχολίων. Η φόρμα (**form_tag**) είναι ρυθμισμένη να υποβάλλεται μέσω **Ajax** (**remote: true**), επιτρέποντας την εμφάνιση των νέων σχολίων στο **modal** χωρίς να απαιτείται ανανέωση της σελίδας. Περιλαμβάνει ένα **hidden_field_tag** για τη μεταβίβαση του **photo_id**, ένα **text_area_tag** ως πλαίσιο κειμένου για την εισαγωγή σχολίων και ένα

submit_tag για την υποβολή τους. Το **modal**, επίσης, διαθέτει κουμπί **Close** για την εύκολη απόκρυψη του.

app/controllers/photo_controllers.rb

```
def comments
  @photo = Photo.find(params[:id])
  @comments = @photo.comments.includes(:user).order(created_at: :desc)
  render json: @comments.as_json(include: { user: { only: :email } })
end
```

Χειρίζεται την ανάκτηση και την απόκριση σχολίων για μια συγκεκριμένη φωτογραφία. Πρώτα, βρίσκει τη φωτογραφία με το **id** που λαμβάνεται από τις παραμέτρους (**params[:id]**). Στη συνέχεια, φορτώνει τα σχόλια της φωτογραφίας μέσω της συσχέτισης **comments**, συμπεριλαμβάνοντας επίσης τα δεδομένα του σχετιζόμενου χρήστη (**user**) για αποδοτική ανάκτηση δεδομένων. Τα σχόλια ταξινομούνται σε φθίνουσα σειρά με βάση την ημερομηνία δημιουργίας (**created_at: :desc**). Τέλος, η μέθοδος επιστρέφει τα σχόλια σε μορφή **JSON**, συμπεριλαμβάνοντας μόνο το **email** των χρηστών μέσω του **as_json**.

app/views/comments/_comments.html.erb

```
<p>
  <strong><%= comment.user.email %>:</strong>
  <%= comment.text %>
</p>
```

Εμφανίζει τα σχόλια μιας φωτογραφίας μέσα στο αναδυόμενο παράθυρο (**modal**). Κάθε σχόλιο περιλαμβάνει το **email** του χρήστη που το δημιούργησε (**comment.user.email**) και το κείμενο του σχολίου (**comment.text**).

app/views/photos/comments.html.haml

```
%h3 Comments:
- if photo.comments.any?
  - photo.comments.order(created_at: :desc).each do |comment|
    %p
```

```
    %strong= comment.user.email

    = comment.text

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

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

- else

  %p No comments yet.
```

Διαχειρίζεται την εμφάνιση των σχολίων στο πλαίσιο του αναδυόμενου παραθύρου (**modal**) για μια φωτογραφία. Συγκεκριμένα, ελέγχει αν υπάρχουν σχόλια για τη φωτογραφία και, αν ναι, τα εμφανίζει με φθίνουσα σειρά βάσει της ημερομηνίας δημιουργίας. Για κάθε σχόλιο, παρουσιάζεται το **email** του χρήστη που το δημιούργησε και το περιεχόμενό του. Επιπλέον, παρέχει τη δυνατότητα διαγραφής των σχολίων, αλλά αυτή περιορίζεται μόνο στον δημιουργό του σχολίου ή στον ιδιοκτήτη της φωτογραφίας, με ένα σχετικό κουμπί διαγραφής. Αν δεν υπάρχουν σχόλια, εμφανίζεται το μήνυμα "**No comments yet.**"

app/views/photos/comments.js.erb

```
$("#comments-container-<%= @photo.id %>").html("<%= j render(partial: 'photos/comments', locals: { photo: @photo }) %>");

$("#comment-form-<%= @photo.id %> textarea").val("");
```

Υλοποιεί τη δυναμική ενημέρωση του **container** των σχολίων όταν ο χρήστης υποβάλλει ένα νέο σχόλιο μέσω του αναδυόμενου παραθύρου (**modal**). Συγκεκριμένα, η πρώτη γραμμή αντικαθιστά το περιεχόμενο του **container** που περιέχει τα σχόλια της φωτογραφίας με το ενημερωμένο **HTML**, το οποίο παράγεται από το **partial _comments.html.erb** και περιλαμβάνει τα πιο πρόσφατα σχόλια. Η δεύτερη γραμμή καθαρίζει το πεδίο κειμένου της φόρμας σχολίων, προετοιμάζοντάς το για νέα εισαγωγή από τον χρήστη.

app/views/comments/create.js.erb

```
const commentsContainer = $("#comments-container-<%= @photo.id %>");

commentsContainer.html("<%= j(render partial: 'photos/comments', locals: { photo: @photo }) %>");

$("#comment-form-<%= @photo.id %> textarea").val("");

$("#comment-form-<%= @photo.id %> textarea").focus();
```

Υλοποιεί τη δυναμική ενημέρωση του **container** των σχολίων για μια φωτογραφία και διαχειρίζεται τη συμπεριφορά της φόρμας σχολίων. Συγκεκριμένα, η πρώτη γραμμή ενημερώνει το **container** που περιέχει τα σχόλια της φωτογραφίας, αντικαθιστώντας το περιεχόμενό του με το νέο **HTML** που δημιουργείται μέσω του **partial**

_comments.html.erb, εξασφαλίζοντας ότι εμφανίζονται τα πιο πρόσφατα σχόλια. Η δεύτερη γραμμή καθαρίζει το πεδίο κειμένου της φόρμας, ώστε να είναι έτοιμο για την εισαγωγή νέου σχολίου από τον χρήστη. Η τρίτη γραμμή, αν χρειαστεί, επαναφέρει την εστίαση στο πεδίο κειμένου της φόρμας σχολίων για καλύτερη εμπειρία χρήστη.

app/views/comments/destroy.js.erb

```
$("#comments-container-<%= @photo.id %>").html("<%= j(render partial: 'photos/comments', locals: { photo: @photo }) %>");
```

Υλοποιεί την ανανέωση του **container** των σχολίων για μια συγκεκριμένη φωτογραφία, όταν ο χρήστης προσθέτει ένα νέο σχόλιο. Ειδικότερα, η γραμμή κώδικα αντικαθιστά το περιεχόμενο του **container #comments-container-<%= @photo.id %>** με το **HTML** που παράγεται δυναμικά μέσω του **partial _comments.html.erb**. Το **partial** αυτό ανανεώνει τη λίστα των σχολίων για τη συγκεκριμένη φωτογραφία (**@photo**) και εμφανίζει τα πιο πρόσφατα σχόλια στη σελίδα.

app/config/routes.rb

```
resources :photos, only: [:show, :destroy] do
  resources :comments, only: [:create, :destroy]
  member do
    get :comments, as: :photo_comments
  end
end
```

Ορίζει τις διαδρομές για τη δημιουργία (**create**), την καταστροφή (**destroy**), και την ανάκτηση σχολίων (**get :comments**) για κάθε φωτογραφία. Η διαδρομή **get :comments** επιτρέπει την απόκτηση των σχολίων μιας φωτογραφίας μέσω **Ajax** και χρησιμοποιείται για την ενημέρωση του περιεχομένου του αναδυόμενου παραθύρου με τα πιο πρόσφατα σχόλια της συγκεκριμένης φωτογραφίας.

2.2 Υλοποίηση του bonus ερωτήματος (3)

Στο bonus ερώτημα, υλοποιήσαμε τη δυνατότητα διαγραφής μιας φωτογραφίας που ανέβασε ο τρέχων χρήστης με διπλό κλικ πάνω στη φωτογραφία. Χρησιμοποιήσαμε **JavaScript** για να διακρίνουμε το μονό από το διπλό κλικ στην εικόνα, ώστε το μονό κλικ να ανοίγει το αναδυόμενο παράθυρο με τα σχόλια, ενώ το διπλό κλικ να ενεργοποιεί τη λειτουργία διαγραφής. Προσθέσαμε έλεγχο δικαιωμάτων στο **PhotosController#destroy** για να διασφαλίσουμε ότι μόνο ο κάτοχος της φωτογραφίας μπορεί να τη διαγράψει. Ενημερώσαμε τον κώδικα **HTML** στο **users/show.html.haml** για να συμπεριλάβουμε το data attribute που ελέγχει αν η φωτογραφία ανήκει στον τρέχοντα χρήστη. Τέλος, το **JavaScript** αρχείο **photos.js** περιέχει την κατάλληλη λογική ώστε η διαγραφή να γίνεται μέσω **AJAX** κλήσης, αφαιρώντας τη φωτογραφία από τη σελίδα χωρίς ανανέωση. Πιο συγκεκριμένα, οι αλλαγές ήταν οι εξής:

app/assets/photos.js

```
if (isCurrentUser) {
  image.addEventListener("dblclick", (event) => {
    const photoId = event.currentTarget.dataset.photoId;
    if (confirm("Are you sure you want to delete this photo?")) {
      fetch(`/photos/${photoId}`, {
        method: "DELETE",
        headers: {
          "X-CSRF-Token": document.querySelector("meta[name='csrf-token']").content,
          "X-Requested-With": "XMLHttpRequest",
        },
      })
        .then((response) => {
          if (response.ok) {
            const container = event.currentTarget.closest(".slideshow-container");
            if (container) container.remove();
            alert("Photo deleted successfully.");
          } else {
            alert("Failed to delete the photo. Please try again.");
          }
        })
```

```
            .catch((error) => {
              console.error("Error:", error);
              alert("An error occurred. Please try again.");
            });
        }
    });
}
```

Υλοποιεί τη λειτουργία διαγραφής φωτογραφίας μέσω διπλού κλικ από τον τρέχοντα χρήστη. Ελέγχει αν η φωτογραφία ανήκει στον τρέχοντα χρήστη (**isCurrentUser**). Αν ναι, προστίθεται ένας ακροατής συμβάντων (**addEventListener**) για διπλό κλικ (**dblclick**) πάνω στη φωτογραφία. Όταν ο χρήστης κάνει διπλό κλικ, εμφανίζεται ένα παράθυρο επιβεβαίωσης. Εάν ο χρήστης επιβεβαιώσει, αποστέλλεται ένα αίτημα **DELETE** στον διακομιστή για τη διαγραφή της φωτογραφίας, χρησιμοποιώντας το **fetch**. Αν η διαγραφή ολοκληρωθεί με επιτυχία, αφαιρείται το **container** της φωτογραφίας από τη σελίδα, ενώ σε περίπτωση αποτυχίας εμφανίζεται ένα μήνυμα σφάλματος.

app/controllers/photos_controller.rb

```
def destroy
  @photo = current_user.photos.find_by(id: params[:id])
  if @photo
    @photo.destroy
    respond_to do |format|
      format.json { render json: { success: true }, status: :ok }
      format.html { redirect_to user_path(current_user), notice: "Photo deleted successfully." }
    end
  else
    respond_to do |format|
      format.json { render json: { success: false, error: "Photo not found or unauthorized" }, status: :unprocessable_entity }
      format.html { redirect_to user_path(current_user), alert: "You are not authorized to delete this photo." }
    end
  end
end
```

Η μέθοδος **destroy** υλοποιεί τη διαγραφή μιας φωτογραφίας που ανήκει στον τρέχοντα χρήστη. Κατά την κλήση της, η εφαρμογή εντοπίζει τη φωτογραφία με βάση

το **id** της και επαληθεύει ότι ανήκει στον συνδεδεμένο χρήστη (**current_user.photos.find_by(id: params[:id])**). Αν βρεθεί η φωτογραφία, διαγράφεται με την εντολή **@photo.destroy**. Η μέθοδος ανταποκρίνεται διαφορετικά ανάλογα με το **format** του αιτήματος (**HTML** ή **JSON**). Για **HTML** αιτήματα, γίνεται ανακατεύθυνση στη σελίδα του χρήστη με μήνυμα επιτυχίας ή σφάλματος, ενώ για **JSON** επιστρέφεται κατάλληλη απάντηση.

app/views/users/show.html.haml

```
- if current_user == photo.user
  = image_tag photo.image.url(:medium), class: "photo-image deletable", data: { photo_id: photo.id, current_user: true }
```

Χρησιμοποιείται για την εμφάνιση φωτογραφιών που ανήκουν στον τρέχοντα συνδεδεμένο χρήστη. Η φωτογραφία προσδιορίζεται με την κλάση **"photo-image deletable"** και περιλαμβάνει δεδομένα (**data-attributes**) όπως το **photo_id** και την πληροφορία αν ο τρέχων χρήστης είναι ο ιδιοκτήτης της φωτογραφίας (**current_user: true**). Αυτή η δομή είναι απαραίτητη για την αναγνώριση και τη διάκριση μεταξύ μονών και διπλών κλικ στη φωτογραφία. Συγκεκριμένα, ο **client-side** κώδικας (**JavaScript**) που ανιχνεύει τα διπλά κλικ, βασίζεται σε αυτήν την πληροφορία για να ενεργοποιήσει τη λειτουργία διαγραφής της φωτογραφίας, εξασφαλίζοντας ότι η διαγραφή επιτρέπεται μόνο για τον τρέχοντα χρήστη.

app/config/routes.rb

```
resources :photos, only: [:show, :destroy] do
  member do
    get :comments, as: :photo_comments
  end
end
```

Η μέθοδος **destroy** είναι διαθέσιμη για τον πόρο **photos**. Όταν πραγματοποιηθεί ένα αίτημα **DELETE /photos/:id**, ο **Rails router** κατευθύνει το αίτημα στη μέθοδο **destroy** του **PhotosController**.