ΟΜΑΔΑ: ΒΑΣΙΛΕΙΑΔΗΣ ΜΙΛΤΙΑΔΗΣ 2944 – ΚΟΥΓΙΑ ΙΩΑΝΝΑ 2731

Εργαστηριακή Άσκηση 2
Προβολή παρουσίασης και αναδυόμενο παράθυρο στην εφαρμογή φωτογραφιών

1. Προβολή παρουσίασης φωτογραφιών.

Όταν ο χρήστης εισέρχεται στην εφαρμογή, θέλουμε να του εμφανίζεται η πιο πρόσφατη φωτογραφία για κάθε ακολουθούμενο χρήστη, συμπεριλαμβανομένου του ιδίου, μόνο με τον τίτλο της, χωρίς σχόλια και tags.
Το πρώτο που κάναμε ήταν να αφαιρέσουμε τον περιττό κώδικα από την views/users/show.htlm.haml. Χρησιμοποιούμε μία λίστα που αποθηκεύει, με αντίστροφη χρονολογική σειρά, τις φωτογραφίες του κάθε χρήστη.
Παρακάτω παρατίθεται ο κώδικας για τον τρέχοντα χρήστη παρόμοιος είναι ο κώδικας για τους ακολουθούμενους χρήστες.

```
1.    - list = @user.photos.order(created_at: :desc)
2.    - if list.any?
3.      %h3#caption= "#{list.first.title}"
4.      %img#slide{ src: list.first.image.url(:medium),
   alt:list.first.title,href: user_photo_comments_path(@user, list.first)}
5.      %div#slides
6.        -list.each do |photo|
7.          %img{ src: photo.image.url(:medium) , alt: pho-
   to.title,href:user_photo_comments_path(@user, photo)}
```

Πέραν του πρώτου slide θέτουμε και το caption της εικόνας εδώ τον τίτλο έτσι ώστε να φαίνεται στην αρχική, τις υπόλοιπες εικόνες και captions τους τις διαχειρίζεται αυτόματα η JavaScript.

Κάνουμε include την jquery στο app/views/layouts

```
1. %script{:src => http://code.jquery.com/jquery-3.3.1.min.js}
```

Στη συνέχεια υλοποιούμε την προβολή παρουσίασης στο assets/javascripts/photo_slideshow.js χρησιμοποιώντας την βιβλιοθήκη jQuery. Όταν είναι έτοιμο το DOM ξεκινάει να τρέχει η συνάρτηση στο αρχείο photo_slideshow. Χρησιμοποιούμε την μέθοδο hover(handlrerIn, handlerOut), γιατί συνδέει τους δύο handlers για κάποιο συγκεκριμένο element. Ο handlerIn τρέχει όταν το ποντίκι βρίσκεται μέσα στο element, και ο handlerOut όταν το ποντίκι βγαίνει από το element. Στην περίπτωση αυτή το element είναι το #slide.
Ο handlerIn είναι η συνάρτηση:

```
1.    $("img").filter("#slide").hover(
2.      function(){
3.        thisSlide = $(this);
4.        thisCaption = $(this).prev("#caption");
5.        imageCache = [];
6.        imageCounter=0;
7.        $(this).filter("#slide").next().find("img").each(
8.          function(){
9.            image = new Image();
10.             image.src = $(this).attr("src");
11.             image.title = $(this).attr("alt");
12.             image.href = $(this).attr("href");
13.             imageCache[imageCounter] = image;
14.             imageCounter++;
15.           }
16.         );
17.         console.log(imageCache);
18.         imageCounter = 0;
19.         var nextImage;
20.         myTimer = setInterval(
21.           function () {
22.             thisCaption.fadeOut(550);
23.             thisSlide.fadeOut(550,
24.               function() {
25.                 imageCounter = (imageCounter + 1) % imageCache.length;
26.                 nextImage = imageCache[imageCounter];
27.                 $(this).attr("href",nextImage.href);
28.                 $(this).attr("src", nextImage.src).fadeIn(500);
29.                $(this).prev("#caption").text(nextImage.title).fadeIn(500);
30.               }
31.             );
32.           },
33.           2000);
34.       },
35.
```

Στον handlerIn: Για κάθε φωτογραφία (img) από την λίστα που φτιάξαμε στην show, κρατάμε την φωτογραφία, τον τίτλο και το path των comments, και φτιάχνουμε ένα αντικείμενο image Το imageCache είναι μια λίστα που μηδενίζεται κάθε φορά που καλείται ο handlerIn, και σε αυτό αποθηκεύουμε όλα τα images ως slides. Η setInterval καθορίζει την ταχύτητα που εναλλασονται τα slides. Η function αυτή είναι που κάνει την εναλλαγή των φωτογραφιών. Με το fadeOut εξαφανίζει τον τιτλο και την φωτογραφια που βλέπουμε τώρα, και με το fadeIn εμφανίζει την επόμενη φωτογραφία με τον τίτλο της.

Και ο handlerOut η συνάρτηση:

```
function(){
      clearInterval(myTimer);
      nextImage = imageCache[0];
      $(this).attr("href",nextImage.href);
      $(this).attr("src", nextImage.src);
      $(this).prev("#caption").text(thisCaption.text());
    }
```

Στον handlerOut: Η function μέσα στο setInterval θα καλείται μέχρι να κληθεί η clearInterval. Επομένως μόλις κληθεί η clearInterval σταματά η προβολή παρουσίασης και φαίνεται πλέον μόνο η πιο πρόσφατη φωτογραφία του χρήστη.

2. Αναδυόμενο παράθυρο με σχόλια και πλαίσιο κειμένου.

Αρχικά θέλουμε όταν γίνεται κλικ σε μια φωτογραφία της προβολής παρουσίασης, να σταματήσει η προβολή. Οπότε προσθέτουμε τον παρακάτω κώδικα στο photo_slideshow.

```
1. $("img").filter("#slide").click(
2.     function(){
3.       clearInterval(myTimer);
4.     }
5.   );
```

Για το αναδυόμενο παράθυρο χρησιμοποιούμε AJAX, γιατί θέλουμε ο κώδικας JavaScript να στέλνει αίτηση HTTP στον server χωρίς φόρτωση νέας σελίδας. Ο comments_controller, μέσα στην μέθοδο index, ελέγχει για X-Requested-With: XMLHttpRequest στο HTTP προκειμένου να χειριστεί αίτηση AJAX, και αν υπάρχει κάνει render το partial _comments.html.haml το οποίο περιέχει τον κώδικα για την εμφάνιση των σχολίων και της φόρμας (πλαισίου κειμένου) για υποβολή σχολίου.

Η μέθοδος index του commentsController:

```
1.     def index
2.         @photo = Photo.find(params[:photo_id])
3.         render(:partial => 'comments', :object => {:photo => @photo}) if
   request.xhr?
4.     end
```

Ο κώδικας της javascript για το αναδυόμενο παράθυρο είναι ο εξής:

```
1. var CommentsPopup = {
2.     setup: function() {
3.         var popupDiv = $('<div id="Comments"></div>');
4.         popupDiv.hide().appendTo($('body'));
5.         $(document).on('click', '#slide', CommentsPopup.getComments);
6.     }
7.     ,getComments: function() {
8.         $.ajax({type: 'GET',
9.             url: $(this).attr('href'),
10.              timeout: 5000,
11.              success: CommentsPopup.showComments,
12.              error: function(xhrObj, textStatus, exception) {
   alert('Error!'); }
13.          // 'success' and 'error' functions will be passed 3 args
14.          });
15.          return(false);
```

```
16.      }
17.      ,showComments: function(data, requestStatus, xhrObject) {
18.          // center a floater 1/2 as wide and 1/4 as tall as screen
19.          var oneFourth = Math.ceil($(window).width() / 4);
20.          $('#comments').css({'left': oneFourth, 'width': 2*oneFourth,
   'top': 250}).html(data).show();
21.              // make the Close link in the hidden element work
22.          $('#closeLink').click(CommentsPopup.hideComments);
23.          return(false); // prevent default link action
24.      }
25.      ,hideComments: function() {
26.          $('#comments').hide();
27.          return(false);
28.      }
29.  };
30.  $(CommentsPopup.setup);
```

Επίσης στο stylesheets/application.sass προστέθηκαν οι εξής γραμμές κώδικα για την εμφάνιση των slides και για το popup

```
1.  #slides img
2.    display: none
3. #comments
4.    padding: 2ex
5.    position: absolute
6.    border: 2px double grey
7.    background: wheat
```

Αρχικά δημιουργούμε ένα κρυφό div για το αναδυόμενο παράθυρο. Όταν ο χρήστης κάνει κλικ στο element #slide καλείται η showComments η οποία μέσω της μεθόδου $.ajax στέλνει αίτηση στο ulr, η οποία λήγει με το πέρας 5000 miliseconds, και αν η αίτηση είναι επιτυχής καλείται η showComments. Αν η αίτηση αποτύχει εμφανίζεται error στο αναδυόμενο παράθυρο.
Η showComments υπολογίζει το μέγεθος και τη θέση του αναδυόμενου παραθύρου για τον καθορισμό του css και καθορίζει το περιεχόμενο html από τα δεδομένα απόκρισης AJAX. Τέλος όταν ο χρήστης πατήσει το “close” για να κλείσει το αναδυόμενο παράθυρο καλείται η hideComments και σβήνει το αναδυόμενο παράθυρο.

Η κλήση render του commentsController χρησιμοποιεί το partial αρχείο _comments.html.haml που παρατίθεται παρακάτω και περνάει την φωτογραφία ως όρισμα για να μπορούν να τυπωθούν τα comments.

```
1. - @photo.comments.each do |comment|
2.      .well.col-sm-12
3.          = image_tag comment.user.avatar.url(:thumb), class: ['avatar']
4.          %strong= comment.user.email
5.          %br/
6.          = comment.body
7.      .div.b
8.          %p= time_ago_in_words(comment.created_at)
9. .row
10.   - if current_user.id == @photo.user.id
```

```
11.       = link_to 'Delete Photo', user_photo_path(current_user, @photo),
   method: "delete", data: {confirm: 'Are you sure?'}, class: ['btn', 'btn-
   danger']
12.   .row
13.   %h1 Add a Comment
14.
15.   = form_for Comment.new, :url => user_photo_comments_path do |form|
16.       = form.hidden_field :user_id, value: current_user.id
17.       = form.text_field :body
18.       = form.submit 'Post Comment'
19.   .row
20.   = link_to 'Close', '', {:id => 'closeLink'}
```

Αξίζει να σημειωθεί ότι ο κώδικας εδώ συμπεριλαμβάνει ένα loop για την εμφάνιση όλων των σχολίων της φωτογραφίας με το avatar και το email του χρήστη που έκανε το σχόλιο και πότε έγινε το σχόλιο.

Στη συνέχεια υπάρχει η φόρμα για το υποβολή νέου σχολίου που αποτελείται από ένα πλαίσιο κειμένου και ένα κουμπί για submit. Στο κάτω μέρος του αναδυόμενου παραθύρου υπάρχει το link για το κλείσιμο του παραθύρου.

Όπως και στην πρώτη φάση σας επισυνάπτουμε τον σύνδεσμο κοινής χρήσης στο google drive όπου θα βρείτε και τις 2 φάσεις της εργασίας, μπορείτε να επιβεβαιώσετε ότι δεν έχει αλλαχτεί το αρχείο από τις ιδιότητες του.
Στον turnin υπάρχει το directory όπου έχουμε βάλει τα αρχεία που χρειάζονται να γίνουν overwrite στο αρχικό treegram για να δουλέψει. Παρ όλα αυτά θα προσπαθήσουμε να στείλουμε και το tar.gz
https://drive.google.com/drive/folders/1kSB6_iNyxtA2NUUb5KIUFap8NXjH5w8z?usp=sharing

Επίσης καθώς καταφέραμε να ΔΕΝ υλοποιήσουμε το bonus ερώτημα προσθέσαμε ένα απλό κουμπί το οποίο δίνει την δυνατότητα διαγραφής της φωτογραφίας σε περίπτωση που ο τρέχον χρήστης είναι ο ιδιοκτήτης της φωτογραφίας.