Technical University of Moldova
Inginerical department S.A.

Report

№: 3 - 4

Teh Web

Author: Terman Emil FAF161
Prof: Plugaru T.

Chisinau 2018

Cmagru - project

I have implemented a web page in ASP.NET Core 2.0 MVC. It's supposed to be an Instagram like applications, where people can upload, like and comment images, and generally - have fun.

Main features

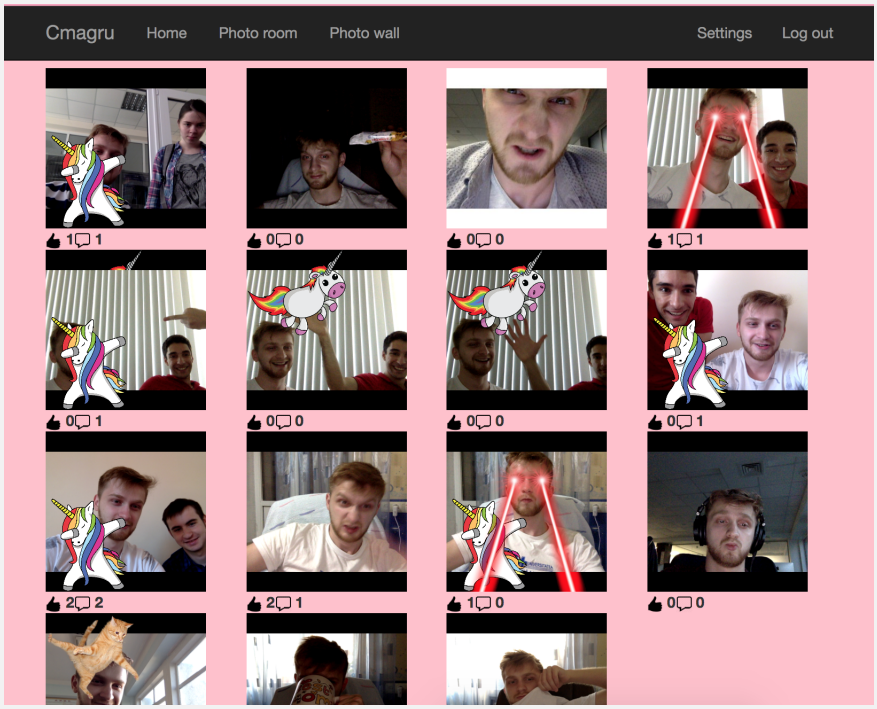
1. upload a local image (with a max size);
2. make a live image from the webcam. In this case, the user can add a sticker to the new photo. The stickers are choosed from all users with *ImgOverlayer* role;
3. like & unlike;
4. comment;
5. remove owned imgs. Only the admin has free access to all photos;
6. email confirmation;
7. configurable email notifications when:
   - sommeone posts a comment on a owned img;
   - the admin removes one of your imgs;
8. edit email (with confirmation);
9. edit UserName & Password;
10. *Forgot password* option;

Goals

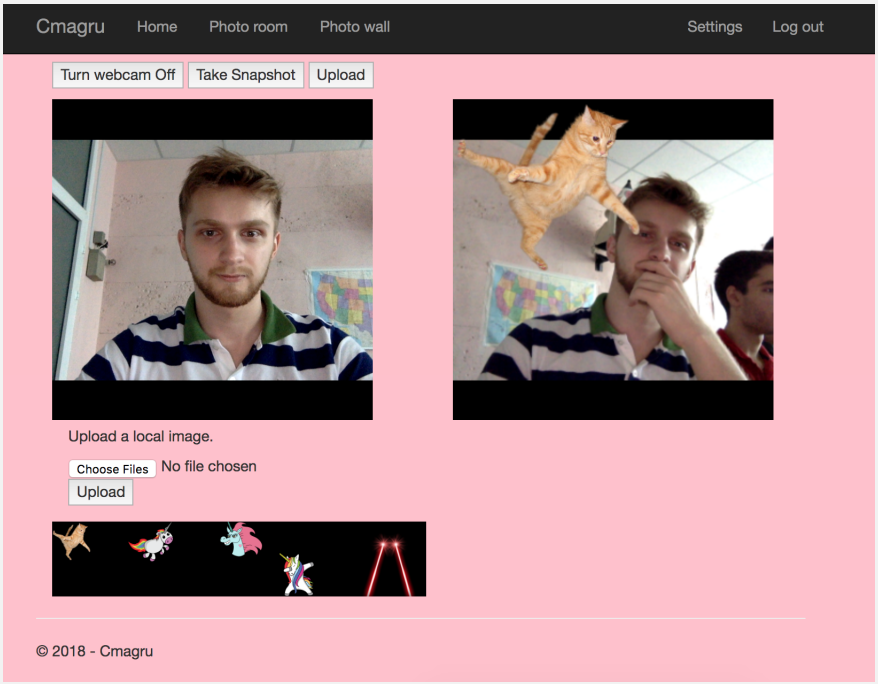
- understanding the basics of MVC. Model View Controller is a design pattern to improve the modularity, thus - maintainability;
- ORM (object relational mapping). For this project I have used *Entity Framework* which made my experience with Databases, a lot easier. In contrast, manually writing SQL sequences is much less maintainable and it can create very obscure errors;
- CRUD (Create Remove Update Delete) database models.
- Multilayered architecture:
  - Business Layer (the logic);
  - Data Layer (database models);
  - Presentation Layer (MVC);

  To follow this pattern, I started with an MVC project, which became my *Presentation*. Then I added *Business* and *Data* layers, as 2 separate projects. (I'm not entirely sure if this is how it's done). This is a very useful pattern to add even more modularity.
- Authentication: I used *Identity Framework* which spared me the trouble to create User data and roles. It also, automates the password hashing and Controller authorization. It provides *System.Security.Claims.ClaimsPrincipal*, a field, which stores data about the currently authenticated user. For example, all users can view all the images, but only authenticated users can Like, Comment or add new Images. If a user is not logged in and tries to access such an action, he is redirected to Registration View.
- Roles. There are 3 roles: *Admin, ImgOverlayer* and *Member*. For the first 2 roles, a default user is created if there isn't one yet. The Admin has controll over all images. The only special ability of ImgOverlayer, is that he can add superposable images. All his uploads won't be shown on the PhotoWall (unless it's the admin or ImgOverlayer).

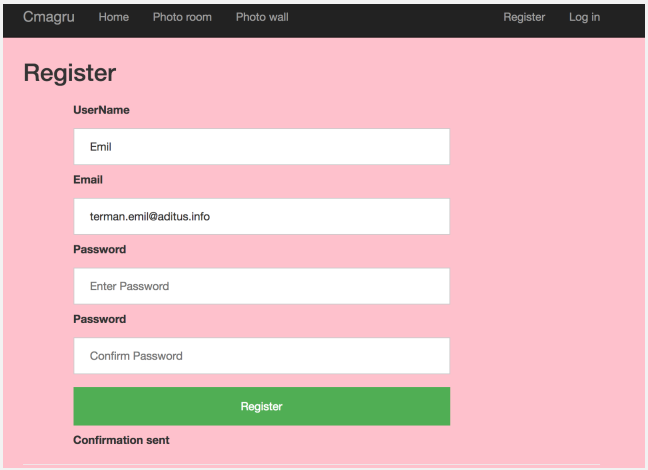
Project preview



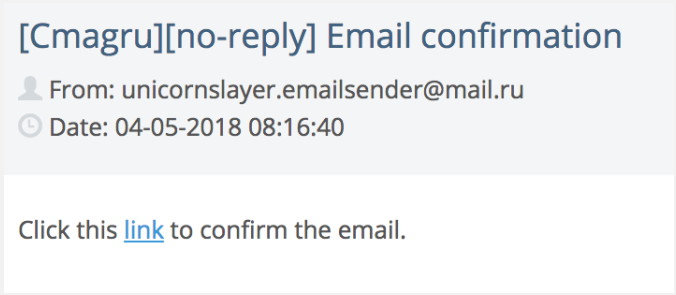
Img 1: PhotoWall



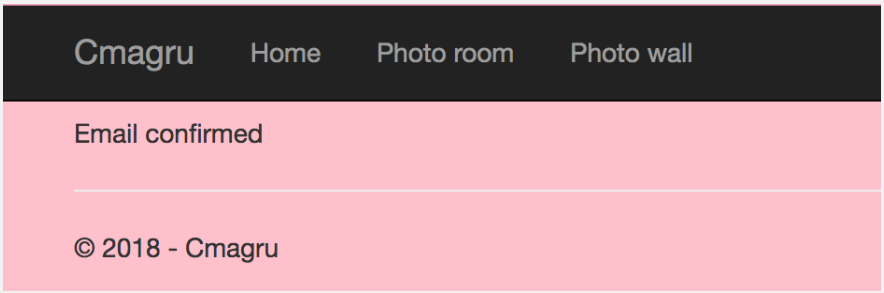
Img 2: PhotoRoom



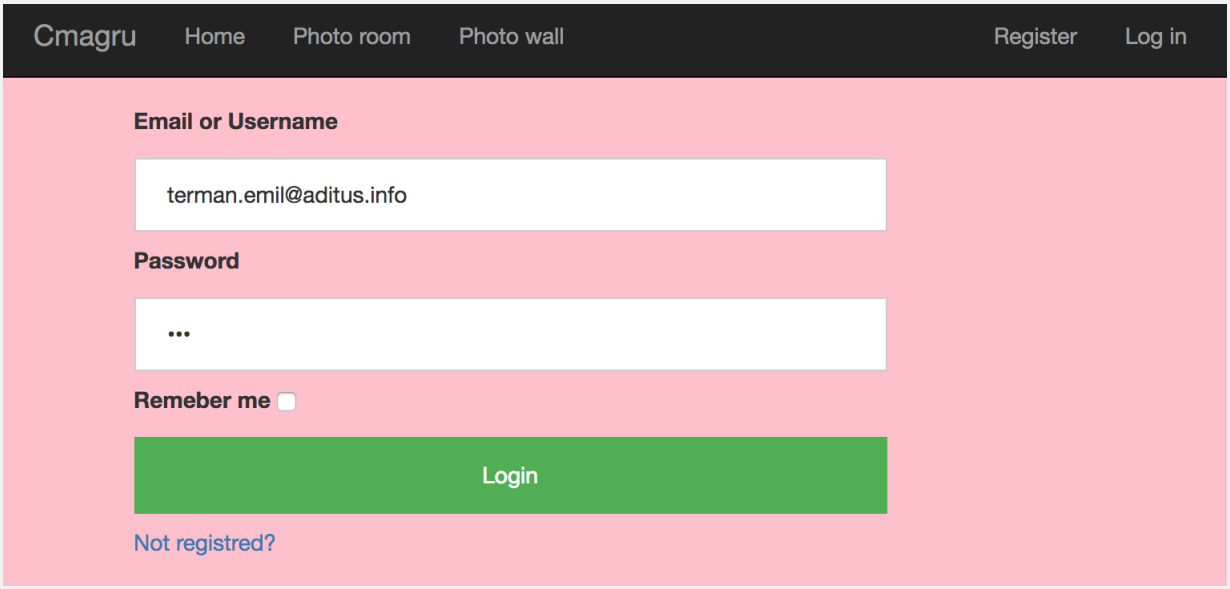
Img 3: Registration



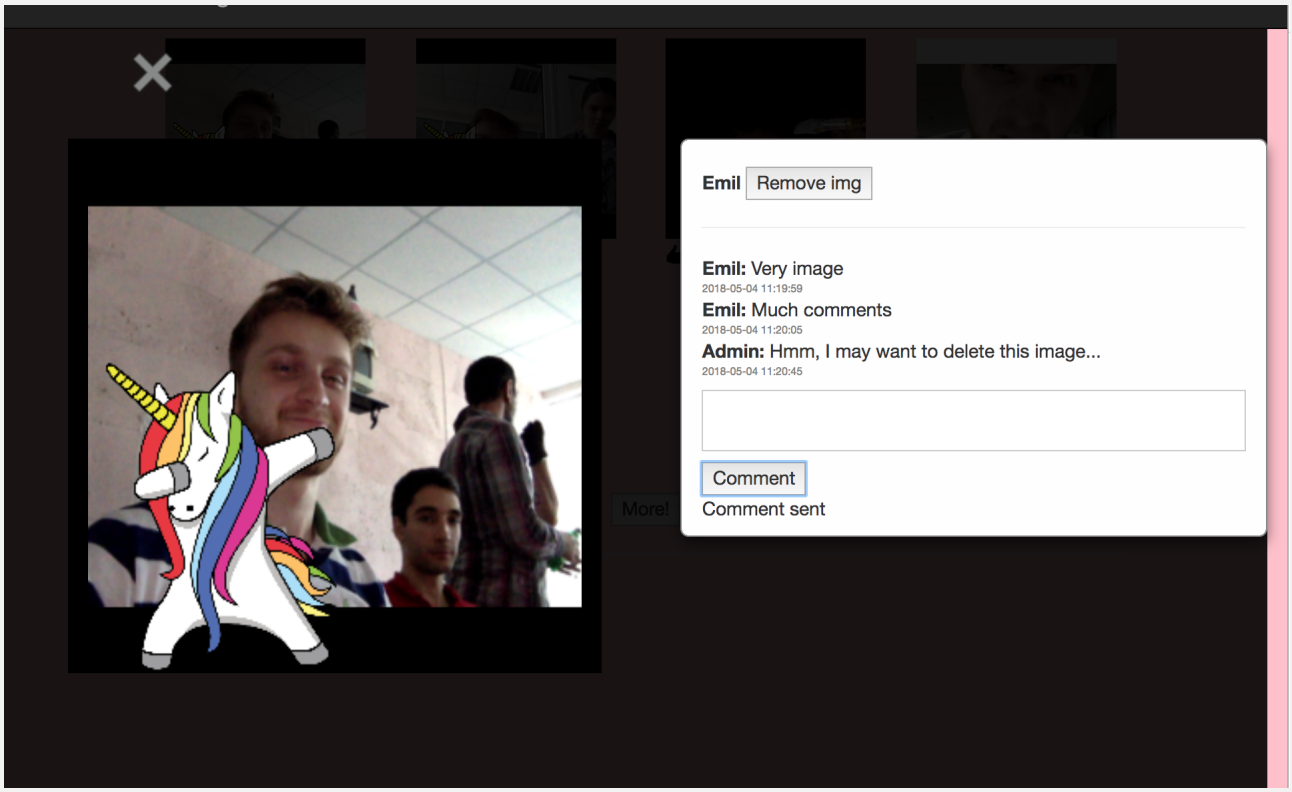
Img 4: Email Confirmation link



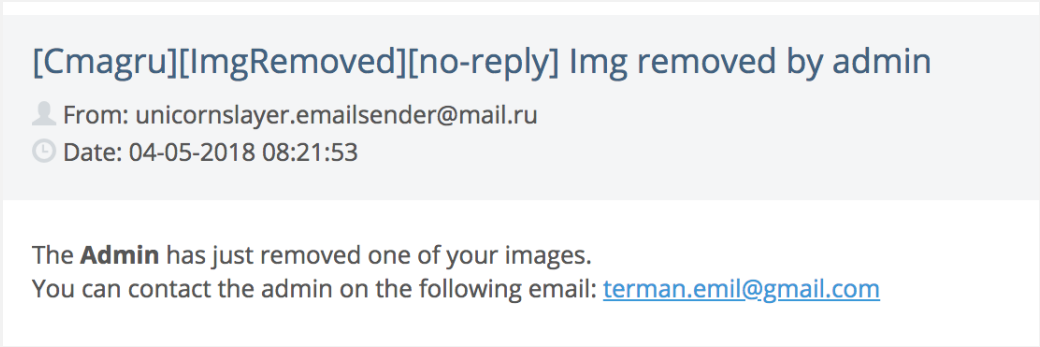
Img 5: Email confirmed



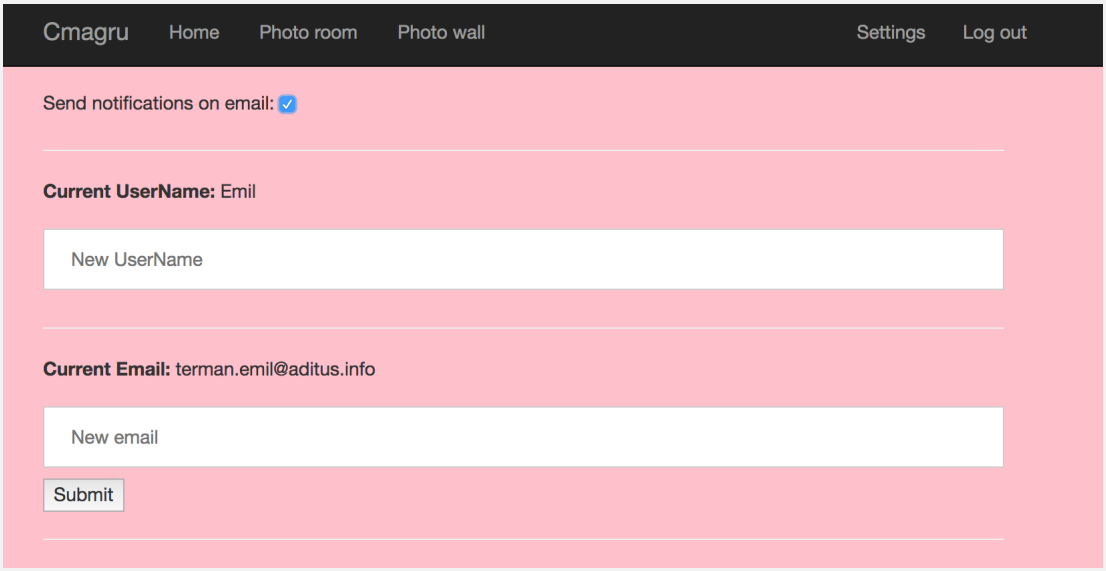
Img 6: Login



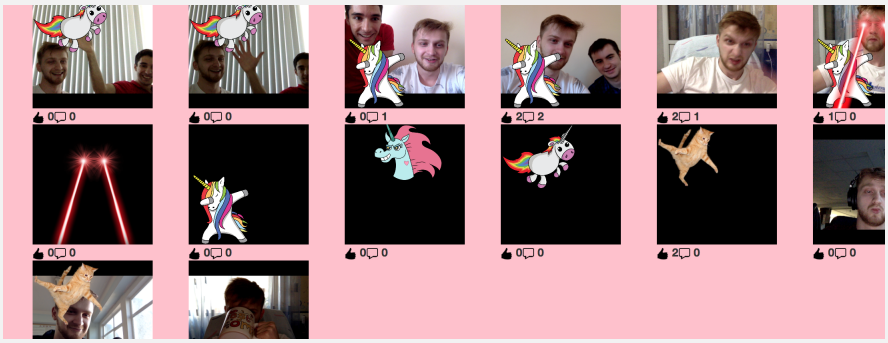
Img 7: PhotoWall - comment section



Img 8: Admin removed your image - email notification



Img 9: User Settings



Img 10: ImgOverlayer and Admin can see the superposable images

Conclusion

This project was a very good introduction to Web Applications. I **am** aware that I have many bugs and better ways to do things. For example, I am saving the images on the database, which is a huge overhead. So, the best way to do it, is to save them on the server and put only the img address in DB. I wasn't aware of this at the beginning, but as I progressed, I learned more about some common good practices.

From this project, I concluded that Multilayered Architecture and MVC are pretty good to modulate a Web application. For example, I tried to help some colleges with their own lab and it was much easier for to understand where was the problem, since they were using MVC. But, because one of them didn't separate their Database from the rest of the logic, some of them ended up using Database Models as MVC models.

From my little experience with SQL, I was very delighted to learn that there is something like ORM. Combined with *Linq*, it becomes a pleasure to work with the database. While using *Entity Framework*, I realized how flexible it is, compared to SQL sequences.