

# **Photo Contest**

# Award the best photography

# Project Description

A team of aspiring photographers want an application that can allow them to easily manage online photo contests.

The application has two main parts:

**Organizational** – here, the application owners can organize photo contests.

**For photo junkies** – everyone is welcome to register and to participate in contests. Junkies with certain ranking can be invited to be jury.

# Functional Requirements

## Public Part

The public part must be accessible without authentication i.e. for anonymous users.

**Landing page** (must)– you can show the latest winning photos or something else that might be compelling for people to register.

**Login form** (must)– redirects to the private area of the application.Login requires **username** and **password**.

**Register form** (must)– registers someone as a Photo Junkie. Requires **username**, **first name**, **last name**, and **password.**

## Private part

Accessible only if the user is authenticated.

### Dashboard page

**Dashboard page** (must)is different for **Organizers** and **Photo Junkies**

For **Organizers:** (any of the below can be either a new page, or directly on the dashboard)

* There must be a way to setup a new **Contest.**
* There mustbe a way to view Contests which are in **Phase I.**
* There mustbe a way to view Contests which are in **Phase II.**
* There mustbe a way to view Contests which are **Finished.**
* There mustbe a way to view Photo Junkies.
  + *(If**scoring is implemented)* ordered by ranking (should)

For **Photo Junkies:**

* There must be a way to view active Open contests.
* There mustbe a way to view contests that the junkie currently participates in.
* There should be a way to view finished contests that the junkie participated in.
* *(If**scoring is implemented)* Display **current points and ranking** and how much until next ranking at a visible place (should)

### Contest page

* The Contest Category is always visible.
* **Phase I** (must)
  + Remaining time until Phase II should be displayed.
  + **Jury** can view submitted photos but cannot rate them yet.
  + Junkies see enroll button if the contest is Open and they are not participating.
  + If they are participating and have not uploaded a photo, they see a form for upload:
    - Title – short text **(required)**
    - Story – long text, which tells the captivating story of the phot **(required)**
    - Photo – file **(required)**
  + Only one photo can be uploaded per participant. The photo, title, and story **cannot** be edited (must). Display a warning on submit that any data cannot be changed later (should)
* **Phase II** (must)
  + Remaining time until Finish phase.
  + Participants cannot upload anymore.
  + **Jury** sees a form for each submitted photo.
    - Score (1-10) **(required)**
    - Comment (long text) **(required)**
    - **Checkbox** to mark that the photo does not fit the contest category. If the checkbox is selected, **score 0** is assigned automatically and a **Comment that the category is wrong.** This is the only way to assign Score outside the [1, 10] interval.
    - Each juror can give one review per photo, if a photo is not reviewed, a default score of 3 is awarded.
* **Finished**
  + Jury can no longer review photos. (must)
  + Participants view their score and comments. (must)
  + In this phase, participants can also view the photos submitted by other users, along with their scores and comments by the Jury. (should)

### Create Contest Fage

**Create Contest Form** (must)– either a new page, or on the organizer’s dashboard. The following must be easy to setup.

* **Title** – text field **(required and unique)**
* **Category** – text field **(required)**
* **Open** (must)or **Invitational** (should) Contest. Open means that everyone (except the jury can join)
  + If invitational – a list of users should be available, along with the option to select them (should)
* **Phase I time limit** (must)– anything from one day to one month
* **Phase II time limit** (must)– anything from one hour to one day
* **Select Jury** – all users with Organizer role are automatically selected (must)
  + *(If**scoring is implemented)* Additionally, users with ranking Photo Master can also be selected, if the organizers decide (should)
* **Cover photo** (could)- a photo can be selected for a contest.
  + **Option 1 –** upload a cover photo.
  + **Option 2** – paste the URL of an existing photo.
  + **Option 3** – select the cover from previously uploaded photos.
  + The organizer must be able to choose between all three options and select the easiest for him/her (**all 3 required if cover photo is implemented**)

### Scoring (should)

Contest participation should award points. Points are accumulative, so being invited and subsequently winning will award **53 points totals**.

* Joining open contest – **1 point**
* Being invited by organizer – **3 points**
* 3rd place – **20 points (10 points if shared 3rd)**
* 2nd place – **35 points (25 points if shared 2nd)**
* 1st place – **50 points (40 points if shared 1st)**
* Finishing at 1st place with double the score of the 2nd (e.g., 1st has been awarded 8.6 points average, and 2nd is 4.3 or less) – **75 points**
* In case of a tie, positions are shared, so there can be more than one participant at 1st, 2nd, and 3rd places, all in the same contest.

For example, two 1st places, one 2nd and four 3rds; the two winners will each get 40 points, the only 2nd place will get the full 35 points, and the four 3rd finishers will get 10 points.

**Ranking**:

* (0-50) points – **Junkie**
* (51 – 150) points – **Enthusiast**
* (151 – 1000) points – **Master** (can now be invited as jury)
* (1001 – infinity) points – **Wise and Benevolent Photo Dictator** (can still be jury)

### Social Sharing (could)

Participants that finish 1st, 2nd, or 3rd could have to option to share their achievement to a social media, for example: Facebook - <https://developers.facebook.com/docs/sharing/web/>

## REST API

To provide other developers with your service, you need to develop a REST API. It should leverage HTTP as a transport protocol and clear text JSON for the request and response payloads.

A great API is nothing without a great documentation. The documentation holds the information that is required to successfully consume and integrate with an API. You must use [Swagger](https://swagger.io/) to document yours.

The REST API provides the following capabilities:

1. Users
   * CRUD Operations (must)
   * List and search by username, first name or last name (must)
2. Contests
   * CRUD Operations (must)
   * Switch phase (must)
   * Submit photo (must)
   * Rate photo (must)
   * List and filter by title, category, type and phase (must)
3. Photos
   * CRUD Operations (must)
   * List and search by title (must)

# Technical Requirements

## General

* Follow REST API design [best practices](https://blog.florimondmanca.com/restful-api-design-13-best-practices-to-make-your-users-happy) when designing the REST API (see Appendix)
* Use tiered project structure (separate the application in layers)
* You should implement proper exception handling and propagation
* Try to think ahead. When developing something, think – “How hard would it be to change/modify this later?”

## Git

Commits in the GitLab repository should give a good overview of how the project was developed, which features were created first and the people who contributed. Contributions from all team members must be evident through the git commit history! The repository must contain the complete application source code and any scripts (database scripts, for example).

Provide a link to a GitLab repository with the following information in the README.md file:

* + Project description
  + Link to the hosted project (if hosted online)
  + Instructions how to setup and run the project locally
  + Images of the database relations (must)

## Optional Requirements

Besides all requirements marked as should and could, here are some more *optional* requirements:

* Integrate your project with a Continuous Integration server (e.g., GitLab’s own) and configure your unit tests to run on each commit to your master branch
* Host your application's backend in a public hosting provider of your choice (e.g., AWS, Azure, Heroku)
* Use branches while working with Git

# Teamwork Guidelines

Please see the Teamwork Guidelines document.

# Appendix

* + [Guidelines for designing good REST API](https://blog.florimondmanca.com/restful-api-design-13-best-practices-to-make-your-users-happy)
  + [Guidelines for URL encoding](http://www.talisman.org/~erlkonig/misc/lunatech%5Ewhat-every-webdev-must-know-about-url-encoding/)
  + [Git commits - an effective style guide](https://dev.to/pavlosisaris/git-commits-an-effective-style-guide-2kkn)
  + [How to Write a Git Commit Message](https://chris.beams.io/posts/git-commit/)

# Legend

* Must – Implement these first.
* Should – if you have time left, try to implement these.
* Could – only if you are ready with everything else give these a go.