**Advanced Web Technologies**
**Academic Year 2016 2017**
**Project Specifications**

The project addresses the development of a web based crowdsourcing platform for image annotations using a Rich Internet Application architecture, whereby the interface is contained in a single page and client-server roundtrips are minimized.
The application manages crowdsourcing campaigns published by **managers** and executed by **workers**. Both kinds of users can register in the platform and login under their specific role.
The manager can create one or more crowdsourcing campaigns. A campaign has a name, some configuration parameters and a set of input images. Aim of the campaign is the annotation of images with the skyline of the landscape. A preliminary step allows workers to validate or discard the images to be annotated: only the images that contain a mountain profile will pass this first stage and therefore are *selected*; they will be then *annotated*.
Each task is executed by a set of workers; the assignment of the images to each worker is decided by means of a scheduling algorithm, which is based on parameters associated with the campaign (the algorithm needs not be realized and will be provided through APIs). The worker's tasks are executed as follows:

- **Image Selection task**: the input for this task is an image; its output is a Boolean value specifying if the image contains a mountain profile or not. Given the image, the user can only accept or reject it. For a given image the task must be executed by at least N workers: the value of N is a campaign specific configuration parameter defined by the manager. If an image obtains at least K positive evaluations in the Image Selection task, it can be annotated. K is a campaign specific configuration parameter defined by the manager.


**Figure 1: User Interface example for the "image selection" task**

- **Image Annotation task**: the input of this task is an image; its output is a line (or a set of lines, if the skyline is interrupted by obstacles), which identifies the profile (skyline) of the mountain. Given the image the user can: **draw** the line (the component for drawing the line will be provided); **clear** the annotation, i.e., erase the current annotation and restart drawing; **send** the annotation to the server. It is not permitted to send an empty annotation to the server. The width of the annotation line is a campaign configuration parameter (an integer number of pixels, between 1 to 10). For each image of the campaign, this task must be executed by at least M workers: M is a campaign specific configuration parameter defined by the manager.

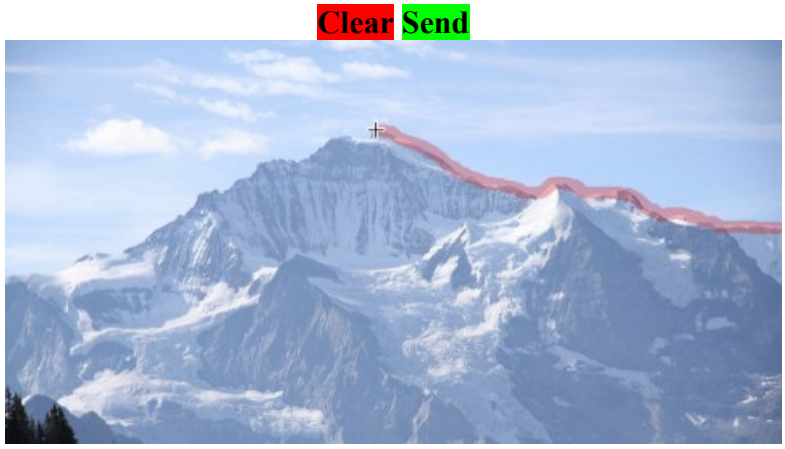


**Figure 2: User interface example for "Image Annotation" task**

Each user can:

- Register in the application. The required fields are:
  - Full Name
  - UserName
  - Password
  - Type of account (Manager/Worker)
- Login/Logout (according to the type of account)
- Edit his/her account. The editable fields are:
  - Full Name
  - Password

The manager can:

- Create a Campaign.
  - Configure the **Image Selection** task of a campaign.
    - Define the minimum number of workers N per image
    - Define the number of positive results K that an image must obtain in the **Image Selection** task to be enabled for the annotation task
  - Configure the **Image Annotation** task of a campaign.
    - Define the minimum number of worker M per image
    - Define the width in pixels of the annotation line.
- Upload all the images of the campaign.
- Enable a group of workers, from the ones registered to the platform, for the execution of the **Image Selection** task
- Enable a group of workers, from the ones registered to the platform, for the **Image Annotation** task.
- **Publish** the campaign, the status of the campaign becomes **running**; once published it will not be possible to edit the campaign (change parameters, upload new images, enable new workers).
- **Terminate** a **running** campaign, which transitions to the **completed** state.
- Consult the list of the **Image Selection** task, with the number of positive and negative votes.
- Examine the images positively evaluated in the **Image Selection** task.
- Examine the annotated images (original image + overlaid annotation) obtained with the **Image Annotation** task.
- Visualize the statistics of a completed campaign: number of approved and rejected images, number of annotated images, average number of annotations per image.

The worker can:

- See the list of campaigns and tasks he/she is enabled for.
- Execute a set of Image Selection or Image Annotation tasks.
- See his/her statistics for each campaign: number of approved and rejected images, number of annotated images, number of available images in the Image Selection task and number of available images in the Image Annotation task.

PROJECT DEVELOPMENT PROCESS

The project must be strictly organized in two parts:

- [Mandatory] **Client Side**. Develop a Single Page Application using Knockout.js (following the MVVM architecture), which uses a set of already developed REST APIs[1] to interact with a server responsible of storing all the information. No data should be stored locally at the client side (if not for caching or login purposes).
- [Optional] **Server Side**. Develop a state-less server, which exposes the predefined REST APIs using **Spring MVC** or **Node.js**.

N.B: The provided REST APIs should be used as defined.

Before the start of the project the students will be requested to fill an online **mandatory** questionnaire. During the development of the project students, students will be asked to fill a **mandatory** daily development diary[2]. Part of it should be filled before starting the work, while the other should be filled at the end of the day. The diary will help the student keep track of progress and will help us evaluate the final results. It is extremely important to fill the diary accurately in a way that reflects the work done and the effort spent.

Students will be randomly divided into two groups[3]:

**Group 1.** **Pure Coding**. This group will develop the project by manually programming in Knockout.js. It is important to organize the code in *View, ViewModel* and *Repositories* for the communication with the backend.

**Group 2.** **Rapid Prototyping**. This group will follow an MDD approach. First, the students will design the platform using the tool available at **ifmledit.org,** which generates a prototype of the platform structured in *View, ViewModel,* and *Repositories;* then they will refine the generated prototype in order to obtain the final version.

Different development results and speed from the two groups are expected and will be considered in the evaluation phase to normalize the judgement. The overall evaluation will consider only the degree of completion of the requirements, the quality of the oral discussion and project presentation, and the accuracy of the project work book-keeping. The optional part (server side development) will increase the final score up to 6 points and is necessary to get the "cum laude" evaluation.

Each individual student must/will:

- Use a GitLab account (gitlab.com)
- Write a mail to carlo.bernaschina@polimi.it with the following information:
  - Full Name
  - Student ID (Matricola)
  - Email
  - GitLab username
- Receive an email with the information needed to access the Git Repository.
- Use the provided Git repository during development.
- Create a **Tag** called **Final Release,** associated with a commit, in order to deliver a finished project. The status of the repository at that commit will be the one that is going to be evaluated.

---

[1] The REST API will be communicated separately from these specifications.

[2] The diary logbook will be communicated separately from these specifications

[3] Groups will be communicated separately from these specifications.