Requirements Analysis and Specification Document (RASD)-UPDATED

1. Introduction

As it was discussed during the oral exam, our specifications were kind of narrow, as they only allowed the student to check-in/out, filter according to specific criteria or post messages that others can see. Moreover, the assumption we made that students will act correctly was considered acceptable and justifiable since the only way to impose proper behavior and make sure that it was followed accordingly is by the use of expensive hardware installations in the study areas (such as sensors or cameras...); and so even the use of an incentive or a penalty for proper behavior would not be useful since there would be no easy way to prove it.

However, there was another assumption that we have intrinsically made without paying much attention to it, and this turns out to be problematic and somehow stringent: this was the assumption that basically all students at Polimi will be using our application for sure (otherwise the number of checked-in people in the study area does not truly indicate the real number of students studying inside of it).

The purpose of this document is to provide an update for the specifications of our PoliSAM application.

2. New Specifications

Our goal after the examination was neither to simply state what could have been added in terms of requirements for our software, nor to simply explain the limitations that came along our previously-made assumptions (discussed above). We actually mentioned those points in the introduction, but what we also wanted was to implement some additional interesting requirements for our software (the added features are the ones that were highlighted during the last meeting). These requirements can be divided into two major parts as shown below:

- In order to solve the aforementioned problem *(mentioned in the introduction, **i.e.** the number of checked-in people in the study area does not truly represent the real number of students studying inside of it)*, while only relying on the first assumption that was made (proper behavior guaranteed by students), we have added a requirement for our app, that any student who would like to check-in to a chosen area, has to report the approximate percentage of students occupying it first (with respect to the full capacity); and then he is able to either check-in or cancel the operation.

  This way whenever a student selects a place, he is able to see two different quantities associated to it: one is the number of people who have checked-in and another one indicates the last approximated number of students occupying this place, along with the time at which the last estimation was made. Showing the latter for all the students is important since whenever a user wants to see the last reported estimated number of students in a place, he has to know at what date and time this number exactly refers to. And if the user himself wants to check a study-area to see its occupancy and decide whether or not he should go there, but the last estimated number was given far back from the current timing; he can always send a message associated with the desired option, asking about the current occupation percentage, by using our Message-Posting feature (we also made a simple modification that displays the number of posts made for every different study-area).

- In addition to that, we have added a nice feature, also based on the suggestions given during the exam, which is one that allows the student user to set the different study areas as favorites, and so he can view all of his favorite options. Moreover, any user is also given the option to check all the favored places by all other students, displayed in descending order of preference. In this way, new student users could get an idea of the suggested study-areas available at Polimi.

3. Extended Capabilities for Bubble:

It was asked from us to justify the use of the Bubble development environment, given that it could be a bit limited in terms of what we can achieve; unlike other traditional coding platforms which are generally more flexible to work with. Therefore, it is worth mentioning that while working with Bubble, it is possible to expand the capabilities of this visual programming language through the integration of a number of plugins.

For instance, Bubble provides some standard plugins which allow for:

- Let another service/server programmatically read Bubble app's database. This is useful for building another system that should share the same database, or for exposing data to other developers.
- Let another service/server programmatically trigger some workflows of a Bubble app. This is useful to allow an external system to trigger some actions in a Bubble app.

In general terms, the Bubble standard plugins enable external applications to run specific workflows or read certain data.

Nevertheless, in order to use a Bubble app to read external data or trigger some action on another system, it is necessary to build a new plugin to add this service or to get the right to use those already developed.

Bubble is a continually growing platform that today integrates more than 465 plugins: