German University in Cairo
Media Engineering and Technology
Assoc. Prof. Mervat Abuelkheir
Eng. Ammar Yasser Eng. Nada Ibrahim Eng. Mo.Agamia

---

Software Engineering, Spring term 2020

# GUC Administration System

## Milestones 2 & 3

Please read the following instructions carefully:

- Any case of **plagiarism**, will result in a zero.
- Any case of **cheating**, will result in a zero.
- It is **YOUR responsibility** to ensure that you have:
  - Submitted before the deadline.
  - Submitted the correct file(s).
  - Submitted the correct file(s) names.
  - Submitted your code inside a zipped folder with the format TeamName_milestone_2.zip. For example compteam_milestone_2.zip.

.

# 1 Story Time

Let me tell you a story. Once upon a time, in a faraway organization, there was a pretty crappy system. Now this system was used by employees to log complaints regarding things like the AC, broken desks, plumbing issues, stuck elevators and so on. It was basically sort of a ticketing system. The maintenance department was in charge of keeping track of the tickets and making sure they are forwarded to the people who would fix them. Then there was the IT department that had to make sure it was working smoothly and was rarely ever down because it was basically used during all hours of the day (really, I mean a security guard could log a complaint about the toilet being clogged at 1 am).
Fast forward a few months of using this system and it was established that the system sucked on so many levels. Everyone had problems with it especially the poor IT department in charge of maintaining it:

- It had a very expensive yearly subscription.
- The workflow was so simplistic is didn't meet any of the business requirements.
- The technology used was outdated and was difficult to debug. The IT department also didn't have access to everything in the system.
- Almost ZERO backend support from the software company's side. Do you have an issue you can't solve because you can't access that part of the system? Oops. Log a complaint and we will get back to you in 3-4 business years.
- Item values were passed through URLs. Meaning absolutely no security. Anyone could just change the URL and access parts that only certain administrators were allowed to access (no that couldn't be done with the issues that needed resolving. It was tried.)
- There was zero documentation for that system. So the IT were working blind.
- There was also zero documentation for the department using it so they would teach each other the things they figure out about the system. Eventually the IT made documentation for it for the users.

You would say OK that's all backend stuff what about other stakeholders? Maybe it was easy to use for the maintenance department logging the tickets? Nope.

1. The UI was horrible. It looked like something straight out of Netscape in 1997 (doubt any of you know what Netscape is). It was confusing and difficult to use and oh so slow.
2. The system was meant to encourage other employees to log their own complaints. It was so unusable employees just resorted to just calling the maintenance department to log the requests for them.

3. The system only supported IE. Meaning you couldn't access it from any other browser nor your phone.

Everyone was having a hard time dealing with this system. Mind you, this was 5 years ago I'm talking about, not 10 or more. So there was sufficient technology to find and purchase a proper system. In fact that system had so many better alternatives!
Well, then why did they use this system? Sadly, it was pure politics. Something about the owner of the company being friends with the head of the maintenance department. And pooft. The deal was done. A lot of money was paid and for what?
In terms of UX: it wasn't useful, valuable, usable (I mean horrible UI), desirable or even credible. One the bright side though, at least it was accessible from OUTSIDE the organization "If you know what I mean" It's also safe to say it had very little usability.
What happened the moment they had the chance? The IT scrapped that software and actually created their own.

## 2 Overview

The above story is just one example of many cases that actually occur in real life. We have all had experiences with systems that should, on paper, work but in reality using them is so frustrating that everyone just wonders why can not we have a better a system (rings a bell?).
You can have the best set of features, the best architecture and backend design, however, if your system's usability is subpar, the overall value of the system drops significantly. Hence, in these two milestones you are asked to do the following:

1. Design clickable mockups for the system covering 100 core system requirements. The 100 requirements will make up your MVP. The MVP should include the all the system requirements that you submitted in milestone 1 covering the Researchers' Portal, the evaluation/grading module and the schedule module. For example, if there are $X$ requirements covering the Researchers' Portal, $Y$ requirements covering the evaluation/grading module and $Z$ requirements covering the schedule module in your first milestone submission. You will need to pick $100 - (X + Y + Z)$ core requirements from other modules to be present in the system alongside $X, Y$ and $Z$. As much as possible, try to choose a coherent consistent set that make up a unified UX with emphasis on usability.

2. In order to build good UX, as with any engineering endeavour, you need to follow a plan. Hence, you are supposed to exercises the principles of Agile by creating a product backlog, sprint backlogs (for all sprints you plan to have). Moreover, given your product backlog you need to estimate:
   a) How many sprints will you have.
   b) The duration of each sprint.
   c) The tasks allocated per team member per sprint.

d) A detailed story point estimation (with reasons for choosing the values you will end up choosing) for the first sprint.

e) Estimated time of completion for the project.

The “tasks" that will be in your product backlog are of two types only. They are either designing an interface or designing a test case. You need not worry about having tasks such as building a database table or a backend API and other tasks non related to UX or testing.

3. Finally, you need to V&V your system. Hence, you are asked to support test cases and scenarios for all chosen 100 requirements that will be included in your UX design. However, and this is really important, you are not testing your design you are testing the functionalities as if the APIs responsible for supplying every system requirement are there. To clarify, let’s say we have the following system requirement “As a user I should be able to login". Your task will be to design the API as well as a test case for the API. Designing an API, for all intents and purposes within the scope of this project, simply means deciding what is the input of the API and what is the output of the API. For example, designing an API for the login system requirement will look like this:

   **Request (input):** Username & Password

   **Response (output):** Token

   **Test Case:** Enter correct username and wrong password

   **Tests Data:** Username: CSEN & Password: 603

   **Expected Result:** Error Message: Password must be at least 5 characters

   For every requirement:

   a) Provide a success case

   b) Provide a failure case

## 3 Deliverables

1. UX mockups, you can use any of the following tools:
   - You can implement it using any web technology of your choice.
   - Figma: https://www.figma.com/.
   - InVision: https://www.invisionapp.com/.
   - Marvel: https://marvelapp.com/.
   - Any prototyping tool of your choice.
2. Product plan, an excel sheet containing the following (See ProductPlanTemplate):
   - A tab for the product backlog.
   - A tab for every sprint's backlog.
   - A tab for every sprint's task allocation per team members.
   - A tab for the first sprint's story point estimation with a detailed explanation.
3. Excel sheet containing the 100 APIs design and test cases (See APITemplate)

# 4 Grading Criteria

The grading criteria of this milestone, will depend on the following points and any unfulfilled deliverables will result in grade deduction:

- You should cover at least 100 **unique** system-requirements both in UX design and API design.
- The clickable mockups must be intuitive and easy to use, as it should follow the basic material design guidelines. This means there should be no hurdles or drop outs that will face the examiner when he is navigating through the user journeys of your MVP.
- In our grading, we will consider the learnability level of the submitted mockups. In addition, we will evaluate how close the mockups matches the user-mental model, and how the mockups consider the user efficiency to carry out a specific task.
- The product plan will be considered valid if and only if it covers the MVP discussed in your Milestone 1 submission. The plan should be:
  1. Clear
  2. Logical
  3. Fair
  4. Dependencies are clear
- Your API design should follow SOA principles.
- You should have at least one success case and one failure case per API.