**A Course Project Submitted to**

**Department of Computer Science and Information Systems**

**College of Arts and Sciences**

**American University**

**In Partial Fulfilment of the Requirements for the**

**Software Engineering Course – CSIS 330**

**ClassQueue**

**By**

**………………………………………………………..**

**Advisor**

**Dr. ……………………….**

**Date**

**Fall, 2021**

**TABLE OF CONTENTS**

**LIST OF TABLES**

**LIST OF FIGURES**

**ABSTRACT**

**CHAPTER I**

1. **INTRODUCTION**

A software that auto generates the most optimum future plans and semester schedules based on a student’s major, year, and classes they have taken previously. In many universities, a foundation year is introduced to allow the students to get a feel for the university pace and life. Most of the time, for science and engineering majors, the students follow a set guideline to follow in taking courses until graduation. In an ideal world, everyone has the same schedule and knows their exact requirements. This is not the case. Many times, students have no clue what to take next, or must sit down with already time-strapped professors to go over their requirements. This is costly and time wasting to do for every freshman and sophomore student wanting to get through the basics and general education/early-major requirement courses. Our solution is to create a smart system where the student inputs their credited and passed courses and is advised automatically to select the most appropriate courses for the coming semester(s). This saves time/energy/and headaches for advisors going through the same conversation for weeks. Our main targeted audience are AUK students for now as we included the majors and classes required at AUK, and mainly first year students. The scope of our product for now is AUK, and in the future many other universities will be using this website and the scope of our project will be larger.

**CHAPTER II**

1. **SOFTWARE REQUIREMENTS SPECIFICATION – SRS**
   1. Functional Requirements:

1. Schedule Building

Details: The system must build an optimal semester schedule for incoming students. Input/Output: The system accepts the catalog year in which the student is registered and in return provides the optimal year plan.

Action: Year long schedule is generated as per the requirements of the user. 2. Major Selection

Details: The system must offer major selection to cater different needs. Input/Output: The user inputs their major under a tab, the system then shows the next page.

Action: Major is selected such that the system can identify from the database which schedule to follow.

3. Consider Catalogs

Details: The system should make schedules according to student’s catalog. Input/Output: The user inputs their catalog year, the system highlights a “generate schedule” tab.

Action: System will generate the schedule based on the catalog year.

4. User details

Details: The system must accept user details, finished courses, to build their schedules.

Input/Output: The user inputs their details, finished courses and the system builds the schedule.

Action: The user details are saved in a database.

5. Printing option

Details: The system should offer a printing option for the schedule.

Input/Output: User presses printing button and the system takes the required actions to achieve printing.

Action: The user is given the option to print their schedule if they choose not to use the login option.

6. Login system

Details: The system could save classes and grades via a login system.

Input/Output: The user inputs email and passwords and the system takes the user to the home page with saved data.

Action: The login system will allow users to save their data.

7. Repeated courses

Details: The system could offer option to handle repeated courses.

Input/Output: The user inputs the courses he/she would like to repeat, the system regenerates the schedules for the upcoming year.

Action: Users are allowed to repeat a course if their grade is less than a B.

* 1. System Requirements

Since the system will be initially implemented for the American University of Kuwait, the system’s domain requirements must include the following:

1. The system will follow AUK code of conduct in regard to the academic and personal information being shared.
2. The system will follow the degree path that each department has for each major.
3. The system will optimize the four-year plan in each catalog.
4. The system will keep an updated list of all the classes.
5. The system will keep an updated list of which semesters each class is offered.
   1. Feasibility Study

Need:

1. What are the problems with current processes and how would a new system help eliminate/reduce these problems?

Problems:

* Added effort for declared major’s advisors
* Not straight forward with what classes are the most suitable
* confusion, for lower level year students who haven’t declared a major but would like to pursue a certain career path
* Staff might assign incorrect classes if they are not careful
* Lack of implementing the four year plan as provided by the university
* Crowded advising center during registration weeks

What the system will take care of:

* The system Simplifies the advising process
* Full academic plan for all majors
* eliminates errors or mistakes done by advising staff

* Eliminates the need for advisors upon declaring major
* Encourages the student to follow a plan
* Arranges the students weekly schedule

Urgency:

1. How would the organization cope if this system was not implemented?

The university is currently coping, however there is room for improvement since there are usually huge waiting lines in the advising center with no course of action taken toward the issue.

Value:

1. What direct contribution will the system make to the business objectives and requirements?

The system will speed up the registration process, it eliminates incorrect degree paths and confusion for first year students, completely eliminates the advising process hence advisors will only be needed to obtain RAC numbers.

Integration:

1. Can information be transferred to and from other organizational systems?

The system merges degree works and academic advising, so the information can be transferred, yes.

Availability:

1. Does the system require technology that has not previously been used in the organization?

The system requires only a desktop and internet, both previously used and readily available in the organization.

Scope:

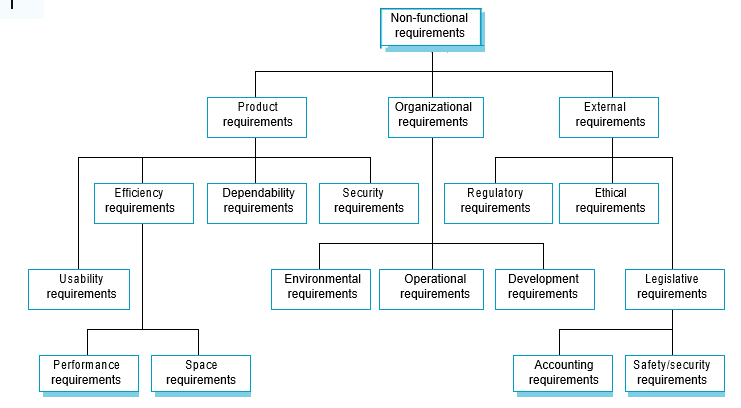
1. What must be supported by the system and what need not be supported?

Supports generating class schedules, optimal advising, and yearly plan but does not support registering for classes or automatic update on grades.

Resources:

1. What are the time and budget limitations?

* 1. Non-Functional Requirements – NFR



Product requirements:

1. Usability:
   1. The system is easy to use.

Measurements: star rating pop up upon sign up.

1. Efficiency:
   1. For first year students, or students who are clueless on university jargon. Measurements: Implement for group of old students to observe the transition.
2. Dependability:
   1. Lays hand in hand with security and hence is mentioned below.
3. Security:
   1. Must include user detail safety and protection in order to avoid grades being leaked. Measurements: Survey users.
4. Accessibility:
   1. Is portable and can be open on any browser.

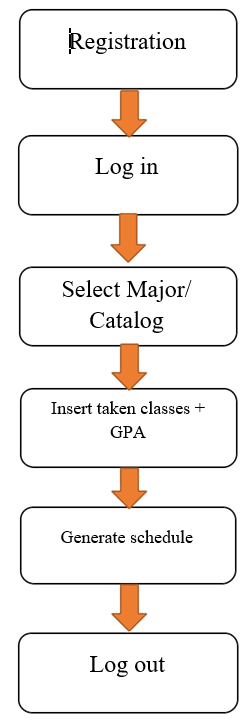
Measurements: Implement and test on several device types such as desktops, ipads, smartphones.

Organizational requirements:

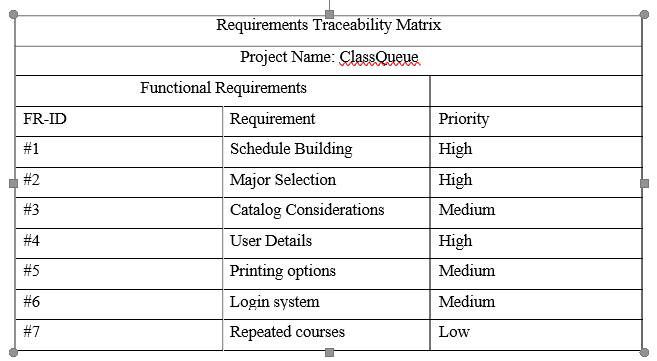
1. Operational:
   1. Resources in the team are shared through physical medium such as USB.
   2. Team presentation will be done through Microsoft power point.
   3. Team communication is done through email and whatsapp.
2. Developmental:
   1. Firebase will be used as a database to store data.

External requirements:

1. Ethical:
   1. Express and implement security on databases that save user grades if applicable.
   2. User personal and academic information is not shared with others.
2. Legislative:
   1. Privacy policies follow university rules in regards to academic records.
   2. User Requirements
3. Easy to use and simplified interface.
4. Can accept the related classes to the user’s chosen major.
5. Able to generate the needed classes based on input.
   1. Use Cases

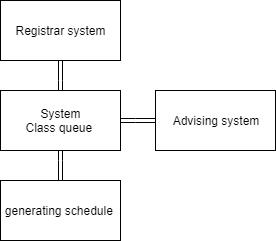


* 1. Requirements Tractability Matrix - RTM



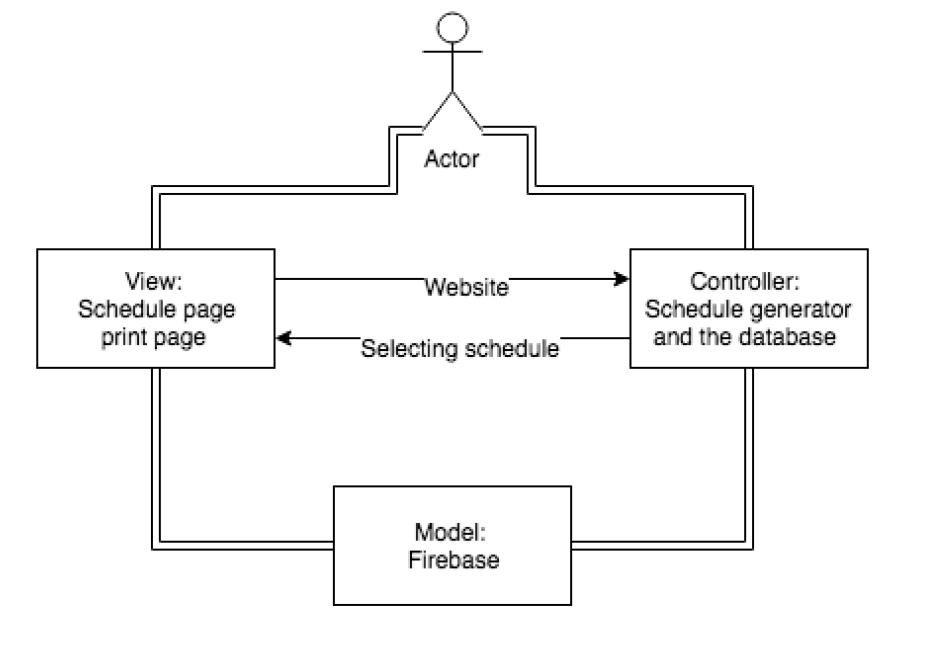
**CHAPTER III**

1. **SYSTEM DESIGN**
   1. Context Architectural Model

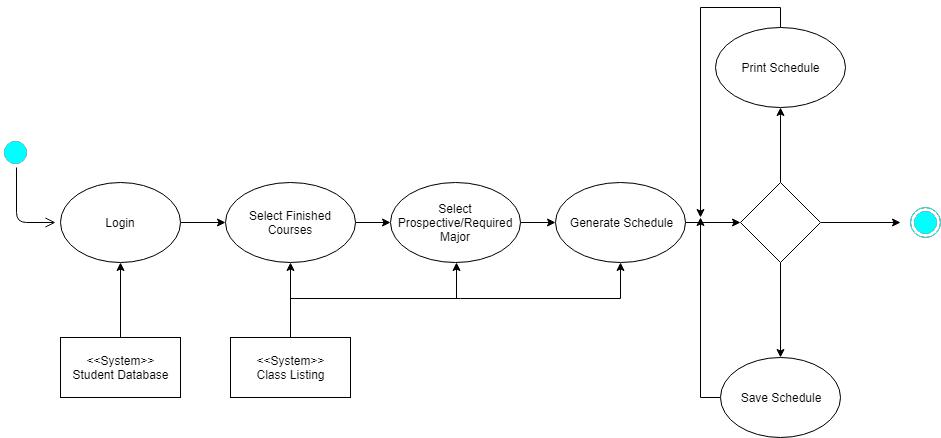


* 1. Architectural Pattern Selection

The MVC Pattern was chose, as it is most appropriate for our web application.

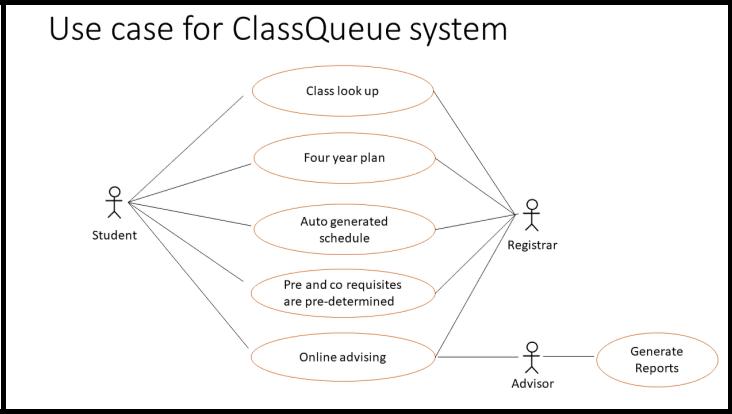


* 1. Process Model



The above process model is a general overview of the entire system from start of lifetime to end. In general, the website starts on the login page, where it prompts the user to input their credentials. If successful, it takes them to a page which prompts them to input/update their finished courses. Then, a major (whether declared or prospective) is to be selected in order to guide the algorithm to create a better-fitting schedule for the student based on their needs. The generated schedule is then given to be printed or downloaded, allowing the student to have a quicker advising session.

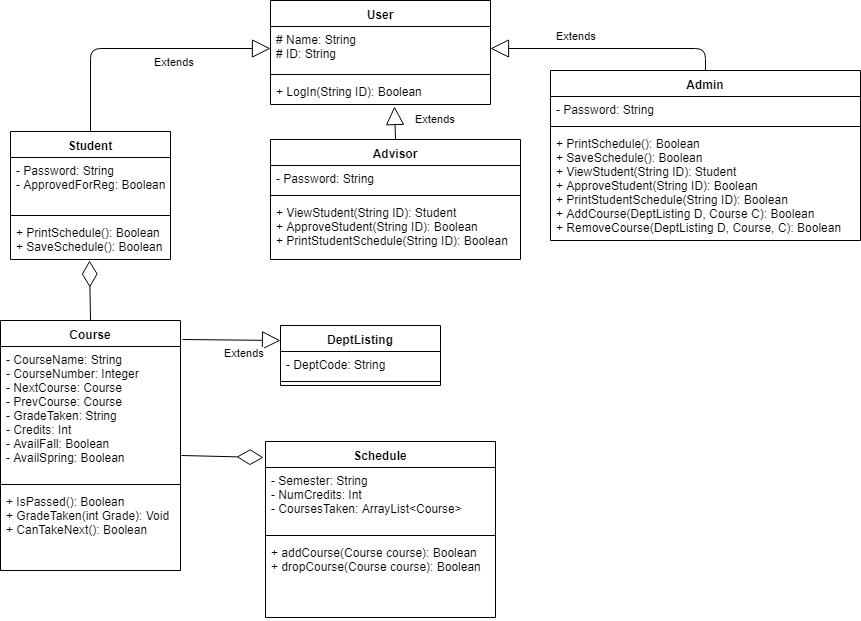
* 1. Use Cases Specification



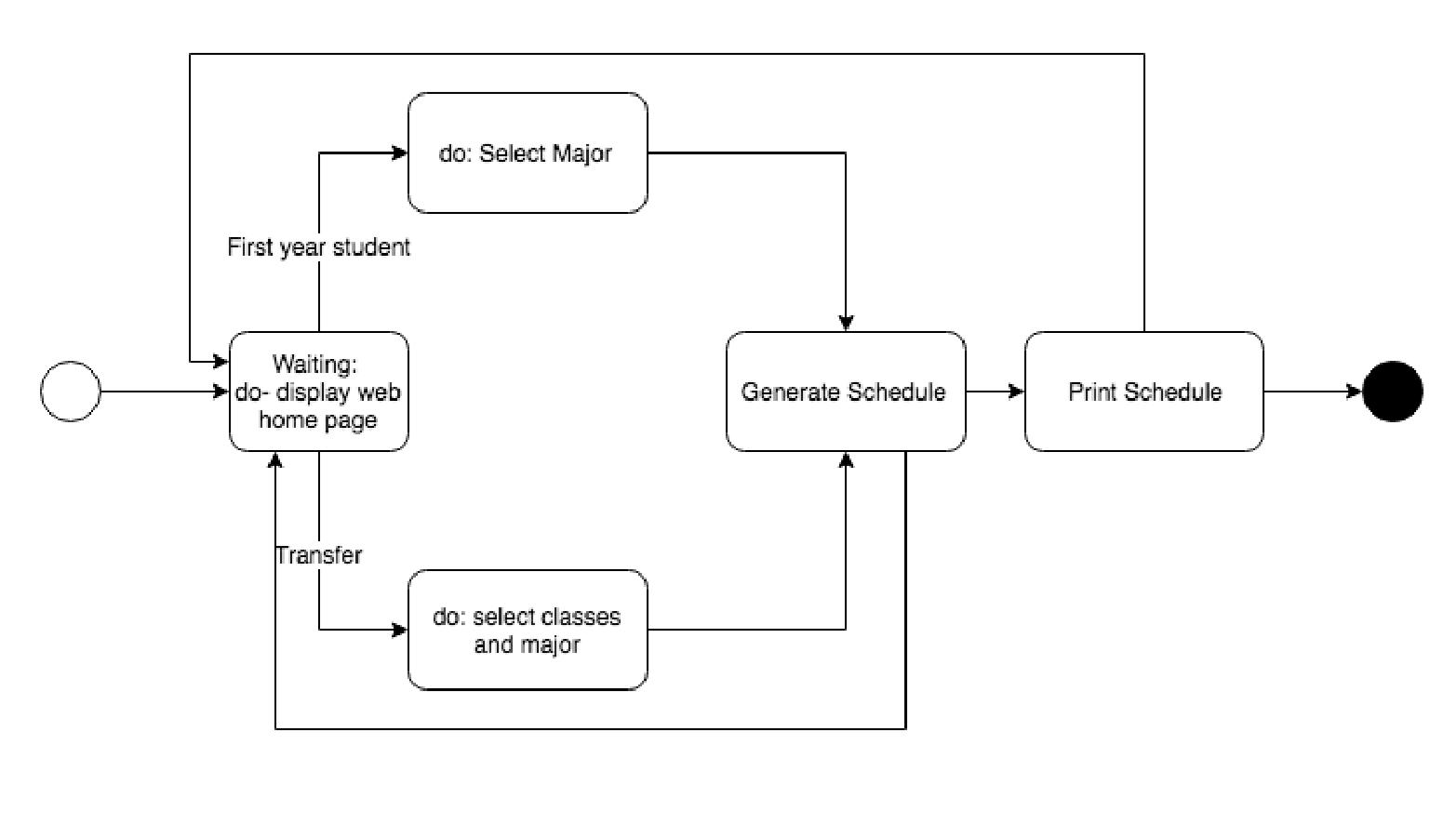
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **System** |  |  | **ClassQueue** |  |
|  |  |  |  |  |
| Use case |  |  | Generating future plans and semester schedules |  |
|  |  |  |  |  |
| Actors |  |  | Students, Advisors, Registrar Office | |
|  |  |  |  | |
| Description |  |  | ▪ A software that auto generates the most optimum future plans and semester |  |
|  |  |  | schedules based on a student’s major, year, and classes they have taken |  |
|  |  |  | previously. |  |
| Stimulus |  |  | ▪ To avoid adding effort on the professors and the advisors. | |
|  |  |  | ▪ To provide a straight forward plan. | |
|  |  |  | ▪ Avoid assigning incorrect classes to students in different majors because of the | |
|  |  |  | lack of experience in some fields. | |
| Response |  |  | ▪ Create a smart system where the student inputs their credited and passed |  |
|  |  |  | courses, and is advised automatically to select the most appropriate courses for |  |
|  |  |  | the coming semester(s). This saves time/energy/and headaches for advisors |  |
|  |  |  | going through the same conversation for weeks. |  |
| Comments |  |  | Targeted users are AUK students currently as we just started this application. In the future, | |
|  |  |  | we are looking to develop the use of this website for all students in any university. | |

* 1. Class Structure Diagram

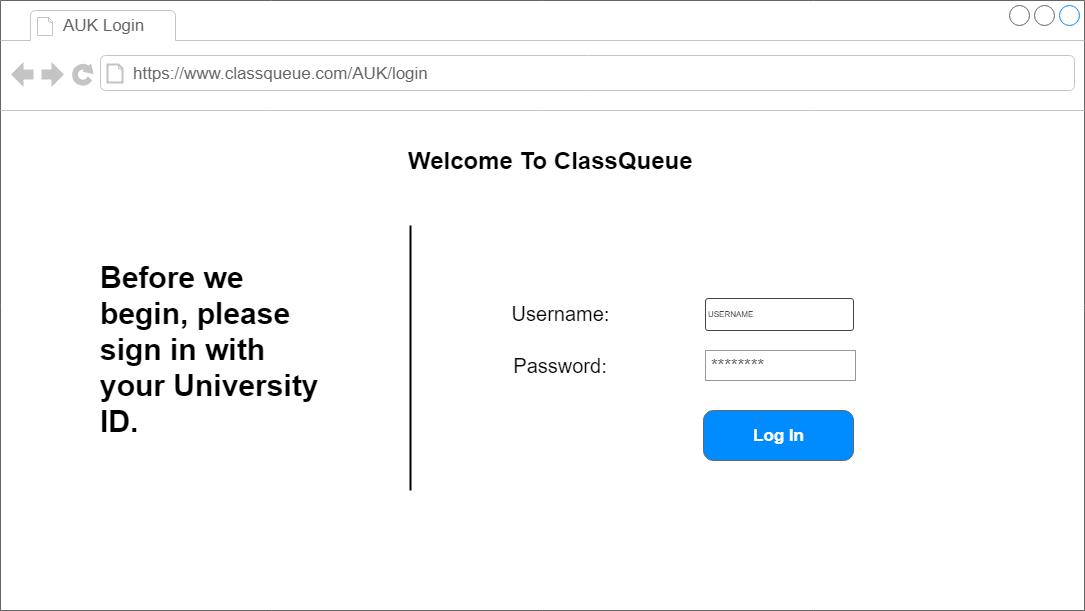
The class diagram below shows the relationships between the different classes working alongside each other in the greater system of ClassQueue. As can be seen, the three users (Student, Advisor, Admin) all have access to the system, but in differing ways. The Student can build their schedule, and the Advisor can approve or access the advisee (student) schedule. The Admin has privileges that allow them to manipulate the course listings, allowing them to control offerings as well as all advisor-level privileges.

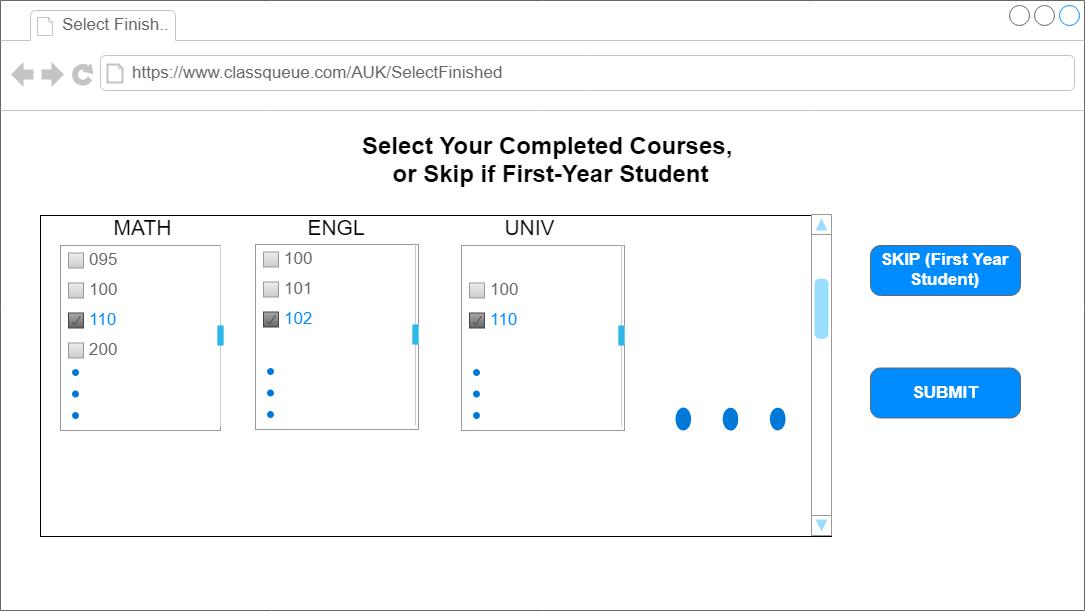


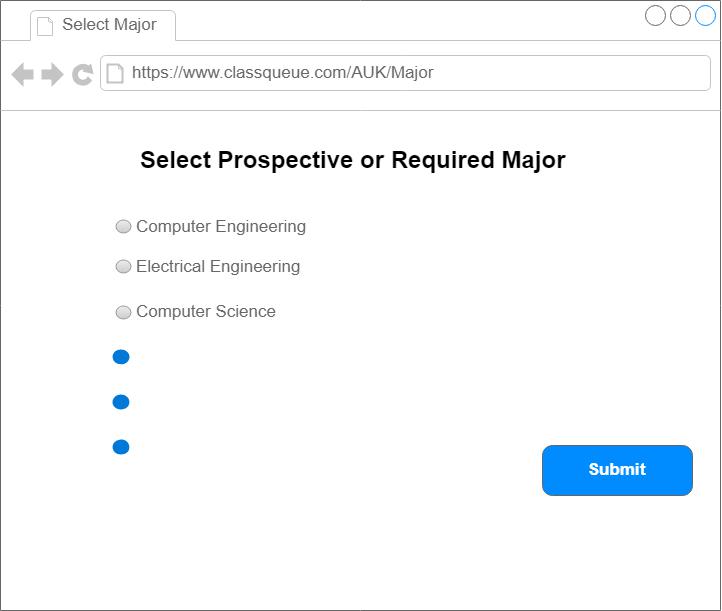
* 1. Finite State Machine (if Applicable)

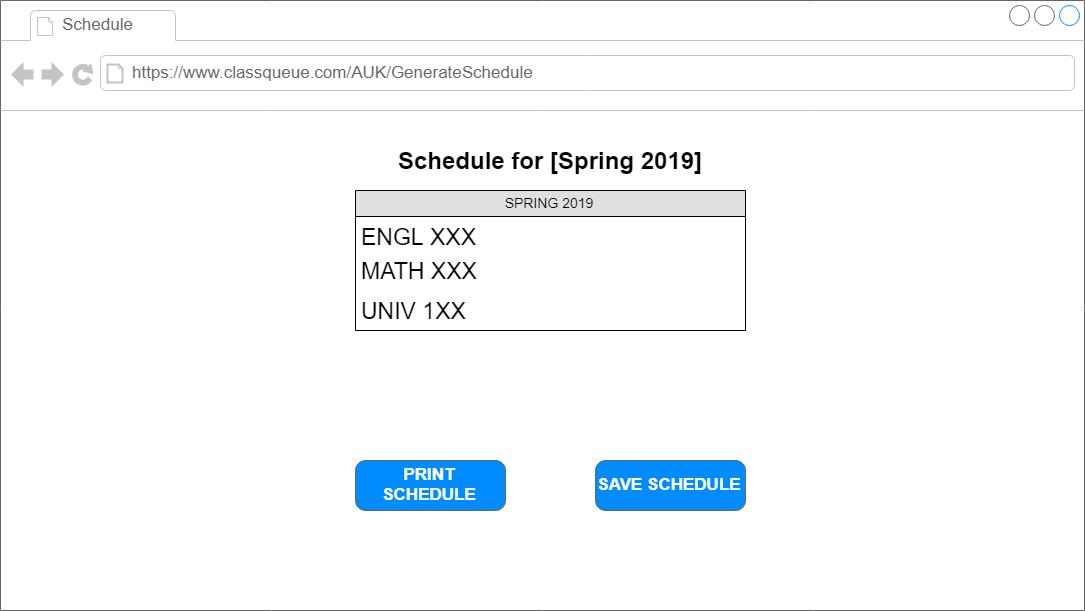


* 1. Prototype





****

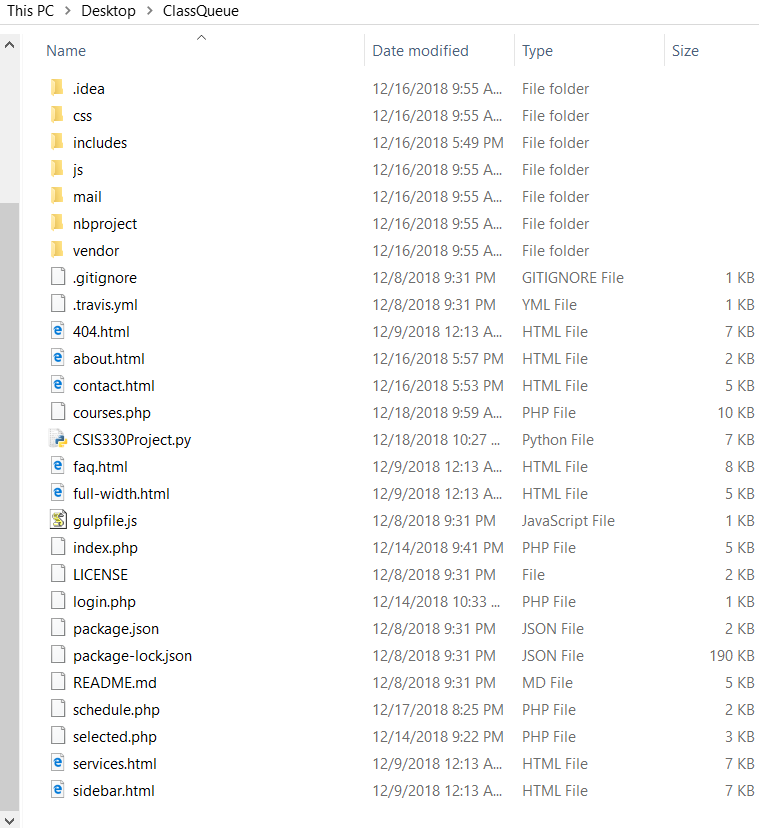
****

**CHAPTER IV**

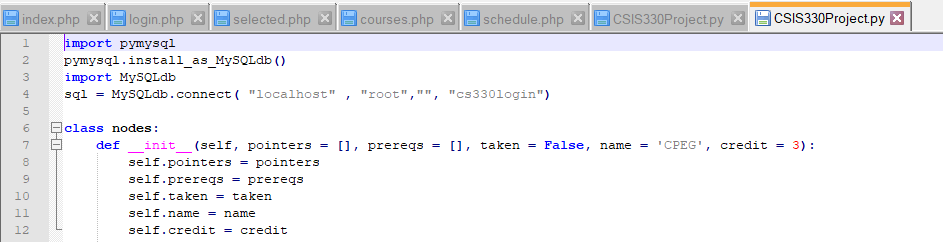
1. **SYSTEM IMPLEMENTATION**
   1. Make, Buy or Lease Analysis

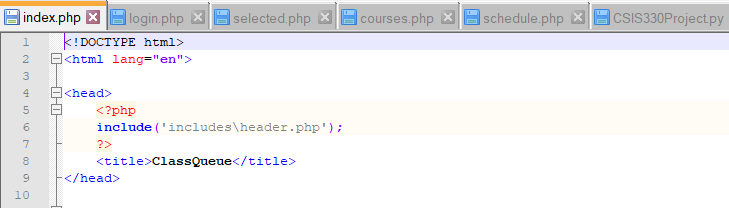
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Component Name** | **Subsystem** | **Decision** | **License** | **Justification** | **Reuse Level** | **Date** |
| XAMPP Database | Database | Buy | GPL | Available for free – open source | System | November 5, 2018 |
| XAMPP Server | Server (local) | Buy | GPL | Available for free – open source | System | November 5, 2018 |
| Bootstrap | Easily- configurable webpage templates | Buy | GPL | Available for free – open source | System | November 5, 2018 |

* 1. Coding



In the implementation we’ve created, all we’ve used was Bootstrap v4.1.3 in terms of libraries for web editing, and a Python script for the schedule building.





As for the php pages, the common includes are all available as shown in the figure above.

* 1. Host Target Specification
     1. Hardware: lightweight web application, so any internet-enabled device may access it.
     2. Software: Works across all OS, and is slated for Mobile OS too. May be used with any recent browser (Chrome, Firefox, etc.)
     3. IDE/Compilers: Genie for Python script, Notepad++ for php/html.
     4. Testing tools: none were used, testing was done manually.

**CHAPTER V**

1. **SYSTEM TESTING**

This section of the report covers testing our software system, and the methods involved in doing so.

* 1. **System Inspection**
     1. SRS

1. The system must pass the inspection without major flaws in functionality.
2. The system should be rid of all major bugs in implementation.
3. The system code should be legible and easy to debug.
   * 1. Architecture

In creating our system inspections, we did not have a formal plan as we were implementing our project using the agile method of development. This allowed us to rapidly create versions of the code rather than set plans to follow.

* + 1. Design Models

After creating our units/components/subsystems, we would go through the code to verify that it is achieving what we had set out to do in the requirements stage of the project. This allowed for more rapid development of the product, given that code is then matched to initially approved requirement and design standards we aimed for in the previous stages.

* 1. **System Testing**
     1. Development Testing & Test-Driven Development
        1. Unit

Unit Testing occurs when the smallest building block of code is tested for correct functionality and conformity with our set requirements. We decided on Test-Driven Development as we found it to be the most agile of methods to build our web program. Once an interface element or a function was implemented, we moved to test its functionality and verify that it works correctly. If so, we skip to the next step in our implementation.

* + - 1. Component

Component Testing occurs when a collection of related units that work together in unison to perform a task is tested for correct functionality, with the intended outputs resulting after certain inputs are fed through. In our Test-Driven Development model, this was done whenever a component like a certain input form or a long contiguous piece of code in the backend was finished. Using manufactured values, we’d put them through the system and compare the result with the expected output. If it is a match, then we proceed with implementing other components for the subsystem.

* + - 1. Subsystem

Subsystem testing occurs when a significant part of the whole system is finished and should be checked for correct functionality and verified for completion. In our Test-Driven Development implementation, this happened whenever an entire webpage was finished, and we’ve included forms that would take in the needed information to the next page. Once we have made sure of the correct implementation built, we can then test to see if it all fits together into the greater system, i.e. one webpage flawlessly conveys data to the other webpage.

* + - 1. System

System testing occurs at the very end, when all the subsystems have been correctly configured and are integrated with each other seamlessly.

* + 1. Release Testing

Release testing happened internally, where the different members of the team were given the implementation to explore and check for inconsistencies/errors. This occurred whenever a major system-level change would happen, or when an incremental step in the process would need a change of the given implementation. If the release test is passed, then we move on to polish/further implement the code in a way to improve its functionality.

* + 1. Acceptance Testing

For acceptance testing, we were short on time and haven’t been able to have the customer themselves validate the final product. We did however ask for user input from fellow programmers and students when we reached a vital stage in implementation that needed to be tested.

**CHAPTER VI**

1. **SYSTEM EVOLUTION**
   1. Anticipated System Evolution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Change** | **Affected Comps** | **Proposed Solution** | **Cost Range** |
| 1 | GUI | User-facing Webpage | Redesign GUI to make it customizable to prospective educational establishments | KD 400-800 |
| 2 | Database | Database | Replace the current free-to-use MySQL database with one that is more professional and secure for larger-scale enterprise. | KD2000-5000 |
| 3 | Server | Server | Implement the software system online so that it may exist globally using a common web address | KD 300-500 for entry level on Amazon AWS cloud servers |

* 1. Anticipated System Transitions till Phase-out

The system should be upgraded as soon as it is implemented by our prospective first customer, upgrading it to accommodate the student traffic (depends on student body). We anticipate this happens upon installation and securing the agreement and should assess the needs of the student body in terms of bandwidth before proceeding with implementation.

* + 1. Prediction of system quality and value for long term. Present data using a table and a column chart.

* + 1. Justify your predictions

Given the nature of the Academic Advising in a small but crowded university like AUK, we expect our system to be highly valuable in creating a streamlined process for freshmen students, new transfers, and up to undeclared sophomores to be able to speed through the Academic Advising system, cutting down the long waiting times and guess-work that might happen in course selection. This would save time and effort in terms of speeding along the registration process and advising would be limited to validation of course selection or minor tweaks in the proposed schedule(s).

**CHAPTER VII**

1. **SYSTEM PLANNING AND MANAGEMENT**

An agreement upon the three us was made that Shahad will be working on the code, Faisal on the design of the website and its interfacing, and Mona will be working on the report with the note that we can rotate to help each other. Adding to that, Faisal contributed as well in writing the report. Building up the whole code from scratch was what we were doing in order to get a detailed website as we wanted. Not only was that, but issues like moving a word to a specific position common, as we wanted everything to look professional.

Gantt Chart

In this part, coding was the most time consuming because of the many ideas we used in our project such as generating a study plan, look in to the taken classes, etc. The hardness of the project was gradually increasing as we went from Phase1 all the way to Coding part.

* 1. Staff Allocation Chart
  2. CoCoMo based Costing

As we built our code from scratch, our code’s lines were roughly between 150 to 200 lines as we had many files with many codes for this project. An effort of Three months to work on this project was a good amount of time to what we delivered in this stage. A website to help students generate and organize their years within their lives in the university would help then not to be lost or confused for whatever they want to major in. As for the cost, we would sell our project for 2.500 KWD because of all the time, effort, quality of the code, and its size. Adding to that, the server that we needed for the project was bought as well. Selling this project to many universities would make us raise enough money to cover all the parameters that were spent on the success of the project.

* 1. Risks Identification

Making a project with more than two files of codes to work was the difficult part as we kept finding errors and debugging it more and more trying to figure out how to fix the errors. Another risk was using the “php” as our website output. We wrote the code using Python Language, which was difficult to us to go search for a way to read it and make it as a link to output on the website. Our last challenge that we thought about is to make this website applicable to all universities around the world that in terms would raise money for us.

* 1. Risks Assessment and Ranking

The most challenging part was debugging many times the code in order to achieve a successful result as we wish. Our plan for the future is to work on widen this project to work within a system of many universities. This would be done by choosing first the university the student is part of and then proceed normally as if they are AUK students.

**CHAPTER VIII**

1. **CONCLUSIONS**
   1. **Seven Design Smells**

How does your system stand with regard to:

* + 1. Rigidity:

The system can be adjusted to suit a specific university requirement or to add more functionality or change one.

* + 1. Fragility:

The software is tested many times and we would like to give it to many users to try it as it will be considered a way to find whether the system suits everyone or not and its percentage of failure. Our system van not be broken easily as it is highly secured and will be supported with a team of technological members to fix any errors.

* + 1. Immobility

The project depends on a database that a university will provide. This database contains students’ information that if this doesn’t exist or taken from the project, the whole software would be broken and no need for it.

* + 1. Viscosity

Is generally non-viscous as we intended to correctly implement our code and leave room for future generalization and abstraction of the code. The only viscous part is the python script that generates the schedules seeing as some things needed to be hardcoded to work.

* + 1. Needless Complexity

The software is not complex as any university student can use it. It has labels and buttons that could lead to whatever they are searching for. The idea of this software is simple as it generates for a student the schedule of the next course.

* + 1. Need Repetition

Nothing is repeated in our software, so not much need for repetition once you pass a certain phase. All that is common is the use of forms from webpage to webpage.

* + 1. Opacity

All the codes are commented and if some part of the code is difficult to understand, then the organization or the university can contact one of the developers.

* 1. **System Strengths and Weaknesses**

This system is a way to ease on the students to plan for their future academic years at AUK based on a student’s major and year as well as showing all their taken classes. Student inputs their credited and passed courses, and is advised automatically to select the most appropriate courses for the coming semester(s). This saves time/energy/and headaches for advisors going through the same conversation for weeks. As for weaknesses, we couldn’t find one weakness of our system that is why we want to give it to many students to try it and only by then we can get their feedback as they might have a point that we didn’t think about.

* 1. **Future Work and Recommendations**

We are planning to make a connection between our website and Student Self Service in order to perceive a grade of each course and show it on our website for any taken course. Adding to that, we are considering having an online advisor who can respond to questions one student might have. To widen and spread our project more, we would include a list of universities and each major of these universities, so students from different universities can use this system to save their times and their advisor’s times. To do so, we can approach universities and sell our software in order to cover what was spent on the project to get it done and to ease students’ lives.

**CHAPTER IX**

1. **APPENDICES** 
   1. Appendix 1
   2. Appendix 2
   3. Appendix 3
   4. …

**CHAPTER X**

1. **REFERENCES**

[01] Rababaah, Aaron R. (2018), Class Notes and Supplements, Software Engineering CSIS 330, Department of Computer Science and Information Systems, American University of Kuwait, Fall-2018.