**Final Report**

CS2043, December 14th, 2020 – *Studentable*

Created by: Tyler Travis, Ethan Garnier, Pranveer Singh, Ben Irvine

Additional Note: Peer Evaluations Submitted through D2L

**User Guide**

Product: Studentable, University Registration Management System

Version: 1.0

Intended Use: For students and instructors to have a synthesized web portal for all information requests related to courses.

Features: As a student, the ability to register and drop classes from a displayed list, see class size, professor, location (assuming it’s not AD). As a teacher, the ability to see enrollment numbers in their classes, along with which individual students are in it.

Safety Warning: All information submitted to the specific webserver is to the discretion of the host and their purposes, the stakeholders have no say in how this information is used. The usage of the database information is to the discretion of the intentions of the host. On a similar note, it’s to the host and their discretion if the following presentation is to follow the extension of Hypertext Transfer Protocol Secure (HTTPS).

Installation Instructions: Please refer to our company website and request a consultation, where our stakeholders will help transfer the framework to your public domain. For users, please refer to your universities public domain for the management system.

How to Operate: Initially sign in with your given credentials. After doing so, you may see classes, along with other useful information. You can register/drop classes, and also change your password.

Contact Details: Please refer to our Human Resource Department if you’d like to get in contact with us, at [it@studentable.ca](mailto:it@studentable.ca), or our phone number 506-378-3063.

**Architecture Style**

The project used the 3 – Tier Architecture Style, primarily due to its strength in organizing functional process logic, user-interface, and data storage are maintained as independent modules. This architectural style is common for web applications.

By using the 3 – Tier Architecture Style, three advantages showed throughout the project.

Since each module was independent and incorporates minimal cohesion, the project development testing was reduced immensely. By doing so, bugs can only be developed in the specific module and no other parts of the application.

The team could divide code up, each focusing on their tier, and optimizing their procedure. This was done through teams’ meetings, dividing work up amongst stakeholders, and ensuring the GANTT chart time-management was mandated.

Due to the applications variation of tiers based on purposes, it allows the group to scale each tier depending on the user requirements.

For the project, our specific tiers are as specified:

* User Interface
  + The top level is the presenting capability to the user and allows proxy manipulation between logic/data tier. This is the only module the user will have access to. It is important that this is a simple, yet effective design. Effectively done through HTML/JS/CSS along with bootstrap.
* Logic Tier
  + The logical functionality of the application, the rules that the program must follow. Requests to NoSQL queries, parsed information requests. This is critical to the UI due to its dictation to how the application functions. Done through Python and external library Flask.
* Data Tier
  + For the use of persistent data, hosted through the universities cloud system. Provides API to the logic tier for methods of accessing and manipulating data without compromising the data storage.

**Requirements**

The overseen goal for the project is to complete a framework University Registration Management System, formally known as Studentable.

Students/Professors will sign into the system using their student emails and self-appointed passwords. This system will be entirely web based, requiring no installation of software on university computers. It will be hosted on the university's public domain, therefore no need for the use of a VPN to access it from home or work.

This system is be able to handle large traffic as in a worst-case scenario: students, professors, and admins can all be accessing it.

Studentable is an organized course catalog for students, instructors, and administration to use. The system allows students at a university to browse a course catalog that provides information on various course offerings they need for their semester. Students will be restricted to only four course offerings per semester, with two alternatives in the case that the other courses are cancelled or filled.

- This system detects when a course should be cancelled (less than 3 students), and when a course is full (more than 10 students).

- The system will send the student's billing information to the university's billing system upon registration completion.

A strict grace period is enforced where students are permitted to change their schedule at the beginning of the semester. During this grace period there is a special student portal that only students have access to using their school credentials. In this portal they can add/drop courses.

A special portal exists through the management system for professors which enables them to see which courses they are instructing, and who are the students enrolled.

Admin accounts are created for school faculty using the logon admin@university-domain.com. This account will allow the admin to perform customizations on the site and modify course information. The admin account is also responsible for the automation of creating new accounts and freeing up old accounts, which will be done through automation.

**Design Decisions**

Design Methodology

The development of Studentable used the Extreme Programming methodology approach. The building process of the application implemented the fundamental best practices of this agile system.

As the consumers requirements will change throughout the entirety of the project, the developers believe small releases are important. Initially only the essential features such as authentication, a dashboard, and database integration are used. As the project versions move forward, so will features allowing a fully functional system with minimal bugs.

For a registration management system and its plethora of cases, the team put testing at the forefront of system building, to ensure acceptable behavior from the software.

Before upgrading to a new version, the team will always refactor the code in order to make it more readable, scalable, and reduce complexity.

Continuous Integration was the team’s first thought when working on the 3-tier architecture. The system will be continuously executed to ensure there is a working build.

A flaw to the Extreme Programming Ideology is that for each university, the system requirements might be slightly different. To comply with this, our standby technicians will customise our software for each different need of a school.

This software development ideology is ideal for application due to its high level of flexibility.

Design Considerations

* Abstraction: Python does not strongly reinforce abstraction, but due to it not being a high priority allows for Python to be a reasonable application tool using Flask imports. It does not directly follow the main-program-with-subroutines architectural style, it decomposes functionality into a set of defined tasks. A UI system will authenticate a user request, interpret data and manipulate/display query data accordingly.
* Modularity: As previously stated in the architecture style, the system is divided into three different tiers: User Interface, Logic Tier, and Database Tier. The UI system has the most cohesion due to its components needed for the logic/database. The UI system does not interact with itself but rather identifies the user requests in an organized, efficient manner. The logic tier has sequential cohesion where UI requests are manipulated and interpreted until the result is achieved. The database would have communicational cohesion, as each element in the database operates on a external piece of data.
* Information Hiding: A major component of the software is software hiding. The user should only see what concerns them in a simple and organized manner through the UI tier. This is reasoning why the application is ran through a public domain, and back end work is unseen. The user may only see information through the dashboard and with specific login credentials, they may see different information. This is similar to abstraction.
* Complexity: the ideology was to make the system as simple as possible whilst maintaining functionality. It requires careful monitoring size metrics and structure-based metrics. Python was chosen as an easy-to-read programming language, with many public imports with power. Outside libraries such as Flask made this entire project possible.