Model-View-Controller 1

Michael Liut
McMaster University
1280 Main Street West
Hamilton, ON., L8S 2W4
liutm@mcmaster.ca

ABSTRACT

This paper provides a detailed explanation of the Model-View-Controller-1 architecture style (a.k.a. MVC-1). It describes the three components of the architecture and how it was implemented in a Python 3 application utilizing both PyQt 5.6 and Matplotlib for constructing the graphical user interface and a Standard Query Language (SQL) database with SQLite3. The Unified Modelling Language (UML) class diagrams will be used to inspect the structure and design of the code. Furthermore, this paper goes on to discuss the benefits and pitfalls of the architecture, alongside possible future work on the presented application.

Keywords

Model-View-Controller 1; Software Design; Python 3; PyQt5; SQLite3

1. INTRODUCTION TO MVC-1

The Model-View-Controller is a software design architecture widely implemented by web developers. MVC is composed of the following three components: the Model, the View, and the Controller.

The Model is the lowest level of the pattern which is responsible for maintaining data. This component provides all core functional services and encapsulates all data details. Furthermore, it does not depend on other modules, and it does not know which views are attached/registered to it.

The View is responsible for displaying all or a portion of the data to the user. It must also update the interfaces whenever the data changes. Essentially a presentation of data in some desirable format. This is usually a user-centric model, where the process is well and long though out.

The Controller is the core of the system. It is the software that controls the interactions between the Model and View. This component manages the user input request and initialization/instantiations/registration of other modules, controls the sequence of user interactions, and selects desired views for output displays.

MVC-1, depicted in *Figure 1*, shows that a user input is entered into a combination of the controller and view, where the model accesses the database and returns the desired information to the controller/view, allowing the user to see it. Note that the controller/view is an observer to the data in the model.

This architecture is unique from other MVC versions as the model and the view are highly coupled. This is usually done in custom tags or in java bean calls. However, one of the serious pitfalls this version of the architecture has is the combination of the business logic and presentational logic, as this prevents the designers and developers from working simultaneously. Furthermore, the limitation of only one component in the system being responsible for sending and receiving messages can impede system performance and even reliability. In general, MVC-1 is less utilized in comparison to its MVC-2 counter-part, but where MVC-1 truly excels is in simplistic systems.

Figure 1: MVC-1 Architecture

In industry, web developers are familiar with the MVC architecture, especially when dealing with interactive sites such as: online shopping, surveys, student registration, etc... In this paper, it will be demonstrated that this architecture is both beneficial and can be implemented in an alternative manner (i.e. not necessarily just in the realm of web development).

2. THE APPLICATION

Every school system, whether it by elementary, secondary or post-secondary utilizes a system for students to keep track of their grades and courses, while enabling professors and course instructors to view their classlists, courses they are teaching, and the averages of the students within them. The system described herein is targeted for post-secondary students and professors, but is not limited to them.

Assuming the user has been registered with the institution, McMaster University for example, a Student is able to login and view their personal information; name, student identification number, and address, what course(s) they are currently enrolled in, their average, what courses are currently avialable to take and what professors are teaching said courses. Once the Student has completed browsing their portal, they have the ability to logout. When we discuss view manipulation,

A Professor, on the other hand, is able to login into their portal and view: the courses they are teaching, the classlists of those courses, and the statistics on the classes depending on their preference of graphical or textual representation. It should be made clear that the Professor has the ability to

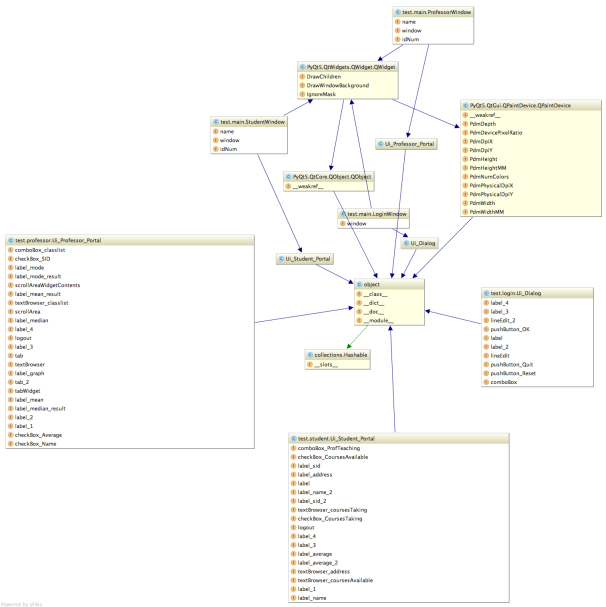

Figure 2: UML Class Diagram

view certain pertinent Student information, which includes: Student identification numbers (SID), names, and averages. This data can be accessed by the Professor in any combination (i.e. solely names, a combination of the three, only grade statistics, etc...). However, data such as Student addresses and other courses the Student is enrolled in are restricted to the Student view only. This is a privacy based implementation of "need-to-know" data restriction.
This application has been developed in Python 3, PyQt 5.6 for graphics, Sip as part of the PyQt requirements, MatPlotLib for the statistical graphing component, and SQLite3 to store all of the data. The use of QT Designer for drag-and-drop graphics were also heavily relied upon, as they eliminate tedious and repetitive copy and pasting of PyQt graphic code.

3. THE APPLICATION WRT MVC-1

The application illustrate an example of the MVC-1 architecture, and allows the user to better understand its benefits and pitfalls. Let us take a look at *Figure 2* where we see the Unified Modeling Language class Diagram. It should be noted that this system is mimicking three intertwined MVC-1 architectures that have uniquelly been embedded together, these are identified as portals herein. This is a result of having three different controllers/view modules for each portal transfer.

3.1 The Login Portal

Once a user is registered in the system, the ability to login to their account is a must! *Figure 3* depicts the view of the login portal. A user is required to use their name and unique identification number to do so. Once the user enters the information on the GUI, the information is "submitted" and the controller/view verifies it with "The Data" (also known as "The Model"). Once the user is authenticated, they are

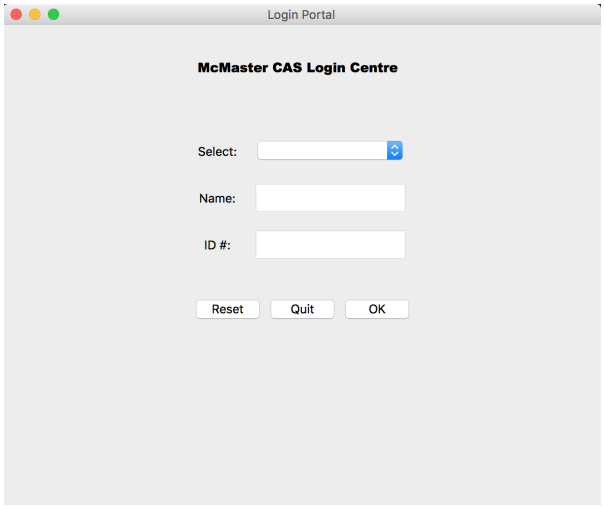

Figure 3: The Login Portal

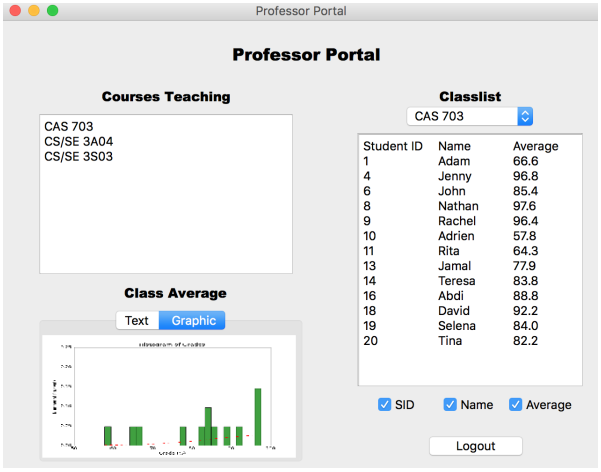

Figure 4: The Professor Portal

transferred to their corresponding portal with unique data, queried from the model.

3.2 The Professor Portal

Once a Professor is logged in, the data is uniquely queried by the controller/view based on the information from the model. *Figure 4* depicts the view of the professor portal. This is displayed in an elegant manner for the user to easily select and see the information they require in a timely and organized manner. Note that the same data is reflected in a different manner, depending on who is logged in and what permissions they have (i.e. what "title" they hold in the system). Furthermore, the Professor has the capability to modify the statistical view and the student information in their classlist.

3.3 The Student Portal

Once a Student is logged in, the data is uniquely queried by the controller/view based on the information from the model. *Figure 5* depicts the view of the student portal.

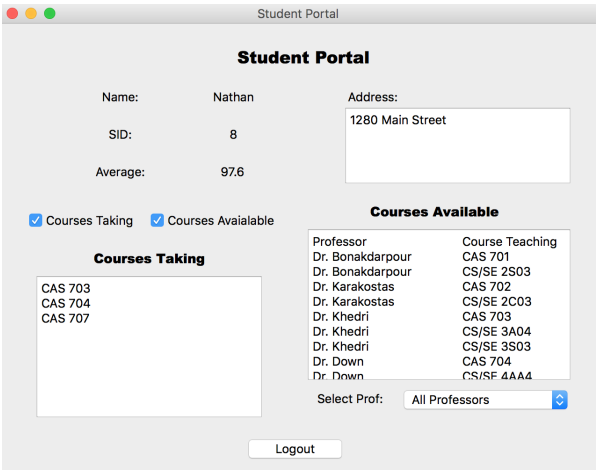

Figure 5: The Student Portal

This is displayed in an elegant manner for the user to easily select and see the information they require in a timely and organized manner. Note that the same data is reflected in a different manner, depending on who is logged in and what permissions they have (i.e. what "title" they hold in the system). Furthermore, a Student has the capability to hide certain windows of their choosing and sort avaiable classes by Professor, allowing them to cutomize their view and easily accessing the data which they require.

3.4 The Data

This component is "The Model" of the MVC architecture, whereby all of the data is stored in an SQL database. The database allows the controller/view to query the information in a very flexible manner by utilizing Standard Query Language. Furthermore, SQL databases provide excellent levels of data protection and integrity.
In this system, there are three tables which hold student data:

- `CREATE TABLE Student('SID' integer,'Name' text, 'Address' text,'Average' real);`
- `CREATE TABLE Courses'SID' integer,'Course1' text, 'Course2' text,'Course3' text);`
- `CREATE TABLE Professor('PID' integer,'Name' text, 'Teaching1' text,'Teaching2' text,'Teaching3' text);`

3.4.1 Privacy and Security

This application does not implement any data security, therefore, no encryption of personal information and generated data will be applied to this project. This is not a new concept, but if a full system of the MVC architecture is to be built, the use of open standards (e.g. Advanced Encryption Standard) will be required for the privacy and security of the users.
Some additional practices that should be integrated into a privacy and security enterprise architectured system are: having privacy as the default setting and being proactive in the case that if possible events arise (which could pose risk to the application, data, and/or and overall system security) there is an action plan already set. By utilizing the rules of least privledge, need-to-know, least trust, mandatory access control and segregation of duty, the security of the overall system will grately improve.

4. TESTING PLAN

Testing is a significant part to every system's design, however, the MVC-1 architecture is constructed in such a way that it makes testing difficult. The core idea for testing is to individually, independently, and systematically automate test for every function of every class. This is unit testing! The first step would be assessing the different types of logic: business, data transformation, presentation, user interaction, and database logic. By breaking the logic down into smaller components we are able to uniquely test each category.
This can be difficult in an MVC-1 structure with a controller and view so highly intertwined that it may be easier to construct a list of all commands which are regularly used. With this list, a step-by-step procedure can be created an thouroughly tested manually by a the testing team. This will ensure to account for the variability of human error, as well as anything test designers could potentially miss while coding. However, there is a serious downfall to this manual testing, which is that it is more time consuming and costly.

5. GENERAL CONSTRAINTS

With the given time contraints it is was not feasible to provide a fully functioning Student and Professor Portal whereby course addition/dropping, registration, and complete user-centric design is implemented. This project is intended to solely explain and depict a real-world example of the MVC-1 architecture.
Some external constraints are: the reliability of system/servers hosting the application, the integrity of the data inputted, the database and even the servers where the data is being stored.

6. CONCLUSION

It is clear from the UML class diagram, *Figure 2*, and the three portal descriptions depicted in *Figure 3,4,5* that the MVC-1 architecture is implemented within the application. MVC-1 is great for web development, but also for simple applications where the layout and view of the data is held in the highest regard. This also implies a complete user-centric application where the data's integrity is treated with the utmost care.
Furthermore, it should be made clear that the testing is an extremely important component of any architecture design. Simple testing leads to a reduced number of errors as well as ensuring good design principles are met.

7. ACKNOWLEDGMENTS

I would like to thank Dr. Khedri for guiding me through the design process and architectural explanation of the Model-View-Controller and Mingfei (Marvin) Hao for assisting me with the QT Designer software and installation of PyQt 5.6.

8. REFERENCES

1. “Patterns for Interaction Oriented and Distributed Architectures” , Dr. Ridha Khedri, <http://www.cas.mcmaster.ca/˜khedri/?page_id=514>
2. “Basic MVC Architecture”, tutorialspoint, <http://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm<
3. “Model-view-controller”, Wikipedia.org, <https://en.wikipedia.org/wiki/Model-view-controller>
4. “Elementary Model View Controller (MVC) by Example”, Alex Cowan, <https://www.youtube.com/watch?v=LiBdzE_DJn4>
5. <https://developer.chrome.com/apps/app_frameworks>
6. <https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html>
7. <http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm>
8. <http://askubuntu.com/questions/56225/is-there-an-gui-designer-for- python>
9. <http://code.tutsplus.com/tutorials/mvc-for-noobs–net-10488>
10. <https://en.wikipedia.org/wiki/ModelâĂŞviewâĂŞcontroller>
11. <http://www.codeproject.com/Articles/486161/Creating-a-simple- application-using-MVC>
12. <http://www.cs.utsa.edu/˜cs3443/mvc-example.html>
13. <http://www.codeproject.com/Articles/879896/Programming-in-Java-using- the-MVC-architecture>
14. <http://www.javavillage.in/view-topic.php?tag=difference-between-mvc1- mvc2>
15. <https://zeekat.nl/articles/mvc-for-the-web.html>
16. <http://www.javavillage.in/view-topic.php?tag=difference-between-mvc1- mvc2>
17. <http://www.codeproject.com/Articles/763928/MVC-Unit-Testing-Unleashed >
18. <Privacy and Security by Design: An Enterprise Architecture Approach, by Ann Cavoukian and Mark Dixon. September 2013.>