EECS TA System

Design Document

11/27/2020

Version 2

TeamSimon

Brevin Simon

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I. Introduction

It is important to implement a design document because it is needed to layout how the project will implement the systems functional requirements. This is done by respecting the constraints that were determined in the requirements phase. With these in mind, it is important to work toward making a design that presents a great quality for the users. A design document also helps as the "in between phase" for the Requirements and the Implemented System. It helps bridge the gap between recognizing the requirements and building the code.

In Section II, we will be highlighting the architecture of the software by analyzing the architecture pattern we will be using in the project along with the reasoning behind the decision to use that pattern. In this analysis we will go in depth on the dependencies between the sub-components along with the reasoning behind those dependencies. After this we will implement a block diagram that will visualize the architecture.

In Section III, we will go showcase the design of the project. For this we will be analyzing the subsystems of the project and explaining the role of each. We will also be looking at what subsystems will share relationships, and which will not. Lastly, we will be looking at how we test this in our project interface.

Document Revision History

Rev 1.0 <11/2/20 > <Initial Design Document>

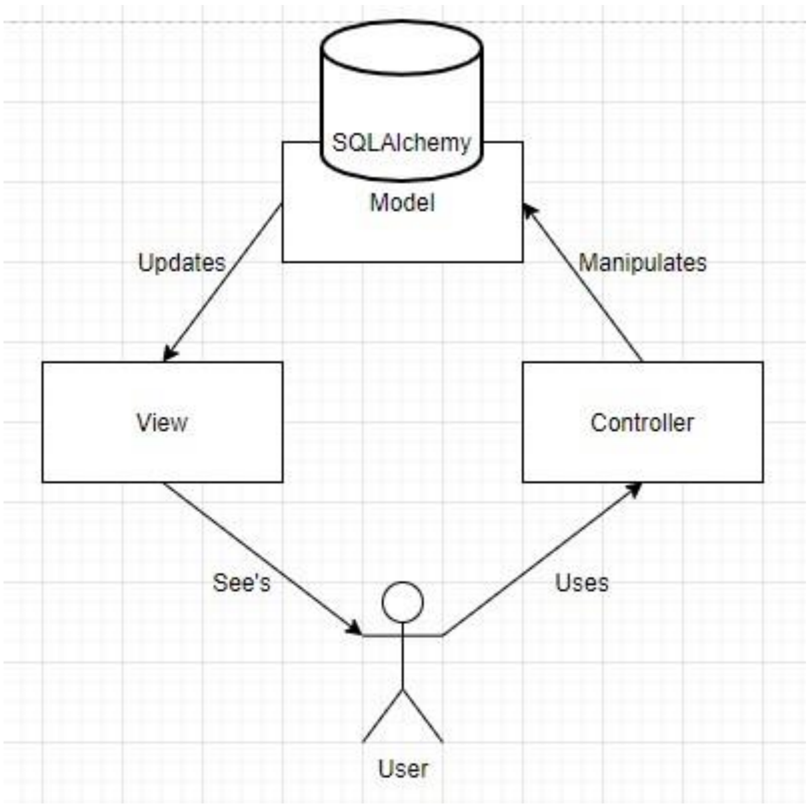
Rev 2.0 <11/27/20> <Iteration-2 and fixing Iteration-1>

II. Architecture Design

II.1. Overview

For our project we will be using the Model-View-Controller (MVC) architecture pattern. This system is broken down into three main systems, the Model, View, and Controller. The Model is used to record the state/data and methods for accessing or changing that state/data. The View's purpose is to render the appearance of the current state along with whatever data needs to be included in the user interface. When the model is changed the view is updated to ensure the user can always see the up-to-date data. Lastly, the Controller is used to translate the actions of the user. These actions translate into the operations performed by the Model. These are typically in the form of button submissions and menu options. Once the Model is changed the controller is responsible for updating the View.

This model seemed good for project because for our TA Application System we needed to use the Controller to branch the difference in paths for the Student and the Professors within the site. It also allowed us to keep the presentation (View) separate from the Model of the project. This along with the Controller helped us get a better understanding for how to strengthen cohesion and loosen the coupling between the two classes (Student and Professor). For example, if we found ourselves prompting for the same data between the classes Student and Professor, we can just add another component to retrieve to loosen the coupling and strengthen the cohesion. This model also worked well for our project because both users need to have the ability to see the updated database at any given time and this implementation ensures that the controller updates the view when the model is changed.



This is the block diagram for application with the MVC architecture pattern. For simplicities sake this can just be a three component Model, View, Controller pattern although technically we can add the Routes to be with the Controller component. The Subsystems in the pattern include Model, View, and Controller. The User can also be referred as the HTTP or the front end page of the program for I figured a User was proper to put their considering we have two different kinds of users so HTTP would have been a little broad. The User can see the View and can operate the site, once an action is performed the Controller will respond accordingly such as update the Model and database based on the new information input by the user. After the Controller updates the Model the model's contents will then by displayed to the View in its updated form where the User can then repeat the process.

III. Design Details

III.1. Subsystem Design

III.1.1.[View]

The role of the View subsystem is to render the contents of the model into the webpage. It is also used to give the user gestures to the Controller for updating of the Model. This subsystem's interfaces vary depending on what User Story the user is currently interacting with. For example, the beginning interface is for the user to create their account, another is for a professor to create a course, or for a student to apply via form for a course created by a professor. Thus, the subsystems it interacts is the Controller.

III.1.2.[Controller]

This subsystem is used to mitigate the interaction of the models and the views. The controller will also define the behavior throughout the whole application. This subsystem works greatly with routes.py and forms.py (also __init__.py) which maps the application around whilst updating the database accordingly. Thus, this subsystem it interacts the most with is the Model.

III.1.3.[Model]

This subsystem is meant to be the core of the application. This is where the databases lie and the information is pulled from, saved to, or deleted. This system allows the controller to access the functionality of the application. This system works with the controller as it gives information to it, but it also gives information to the View subsystem as that is where this information is displayed.

Methods	URL	Description
'GET'	`'/' , '/index'`	Renders the interface for users (i.e. displays courses to apply or courses students have applied for)
'GET' 'POST'	`'/profregister'`	Renders the form for the professors to register their account with all information needed

‘GET’ ‘POST’	`'/studregister'`	Renders the form for the students to register their account with all information needed
‘GET’ ‘POST’	`'/createcourse'`	This is for registered professors to create courses and for them to be added to the courses database
‘POST’	‘/apply/<courseid>’	For student to apply to TA based on courseid
‘GET’ ‘POST’	‘/login’	Login for both students and professors
null	‘/logout’	Logout for both students and professors
‘POST’	‘/unapply/<courseid>’	Students can un-apply for a course

(in iteration -2) Provide your class level design for the subsystem. You should include a UML class diagram visualizing your class level design. In addition, explain each class in detail, specify and explain their methods.
If you have considered alternative designs, please briefly describe your reasons for choosing the final design.

III.2. Data design

Professor Model

id	Integer	This holds the primary_key=True for the professor.
username	String	This is the username that must be unique for the professor.
password_hash	String	This is the password hash for the professor’s account.
firstname	String	This is the first name of the professor.
lastname	String	This is the last name of the professor.
wsuid	String	This is the student id of the professor must be unique.
email	String	This is the email of the professor must be unique.
phone	String	This is the cellphone of the professor must be 10 string long.
coursesinstructing		This will be a relationship for the professor to know what courses he/she has created. The relationship is like the student’s “pendingapps” field.

Student Model

id	Integer	This holds the primary_key=True for the student.
username	String	This is the username that must be unique for the student.
password_hash	String	This is the password hash for the student’s account.
firstname	String	This is the first name of the student.

lastname	String	This is the last name of the student.
wsuid	String	This is the student id of the student must be unique.
email	String	This is the email of the student must be unique.
phone	String	This is the cellphone of the student must be 10 string long.
major	Integer	This is the major number based on the list of tuples I have for the majors.
c_gpa	Float	This is the current GPA of the student.
grad_date	String	The graduation date of the student.

pendingapps		`pendingapps = db.relationship ('Course', secondary = applied,`
		
		`primaryjoin=(applied.c.studentid == id),`
		
		`backref=db.backref('applied', lazy='dynamic'), lazy='dynamic')`
		
		This is the relationship for the student to have courses associated with them upon applying for a job.

NOTE: When starting the project this was how I created the databases for the student and professor. But now after looking at it I believe that a lot of the fields are common between the student and professor databases which would warrant the need for their own database starting from username to phone. The fields between username and phone should be inherited for both classes. But this will not yet be changed until I create rev 2.0 which will be created after revising my requirements document.

Course Model

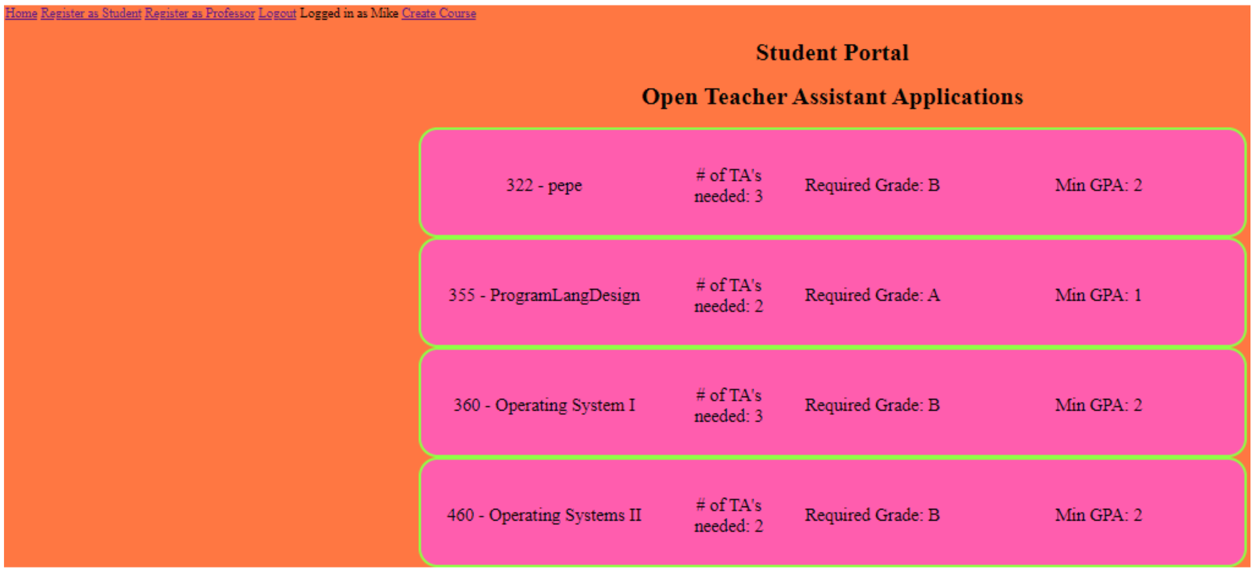
id	Integer	Primary_key=True the id of the course that will be used for relationships with other databases.
coursenum	String	The course number of the course with a size of 3 characters.
title	String	The course title that will be predetermined most likely.
num_ta	Integer	The number of TA's for a course which is default set at one.
min_gpa	Float	The minimum GPA for the student to have to apply for the course.
min_grade	Integer	The minimum Grade for the student to have to apply for the course.

ta_apps		`db.relationship ('Student', secondary = applied,`
		
		`primaryjoin=(applied.c.courseid == id),`
		
		`backref=db.backref('applied', lazy='dynamic'), lazy='dynamic')`
		This relationship is currently set up between it and the student model.

(in iteration -2) Provide a UML diagram of your database model showing the associations and relationships among tables.

III.3. User Interface Design

The current UI design provides a page for students to login, a page for professors to login. When a student logs in they are sent to the page home where they can view each course that needs TA's. When a professor logs in they will also see these courses, but they will have an option to add courses that they instruct. There is currently CSS supported for the course list page but the rest of the application which is essentially just forms does not yet have CSS applied to it. This is what the course list looks like.



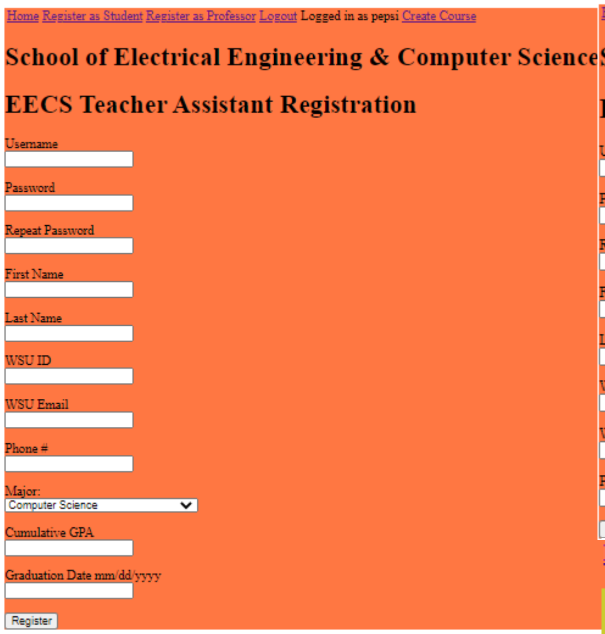
Home Register as Student Register as Professor Logout Logged in as Mike Create Course

Student Portal

Open Teacher Assistant Applications

322 - pepe	# of TA's needed: 3	Required Grade: B	Min GPA: 2
355 - ProgramLangDesign	# of TA's needed: 2	Required Grade: A	Min GPA: 1
360 - Operating System I	# of TA's needed: 3	Required Grade: B	Min GPA: 2
460 - Operating Systems II	# of TA's needed: 2	Required Grade: B	Min GPA: 2

This through the view of the professor where they can "Create Course" in the header of the page. The colors that are currently selected will not be permanent it will probably have a crimson/grey theme at the end of iteration-3 when nearly all of the functionality is implemented. The student view looks the exact same except they will not see the "Create Course" tab and will instead have an "Apply" button on each of the course containers. These serve the user cases for the student and the professor when it comes to courses. Aside from the organization of the courses which would allow the students to see the jobs that they are eligible for everything else is implanted on the backend. The following images are the form pages that are quite uninteresting. The first two images are of the student and professor login, respectively. The last image is of the login page which was implemented within iteration-2.



Home Register as Student Register as Professor Logout Logged in as pepsi Create Course

School of Electrical Engineering & Computer Science

EECS Teacher Assistant Registration

Username

Password

Repeat Password

First Name

Last Name

WSU ID

WSU Email

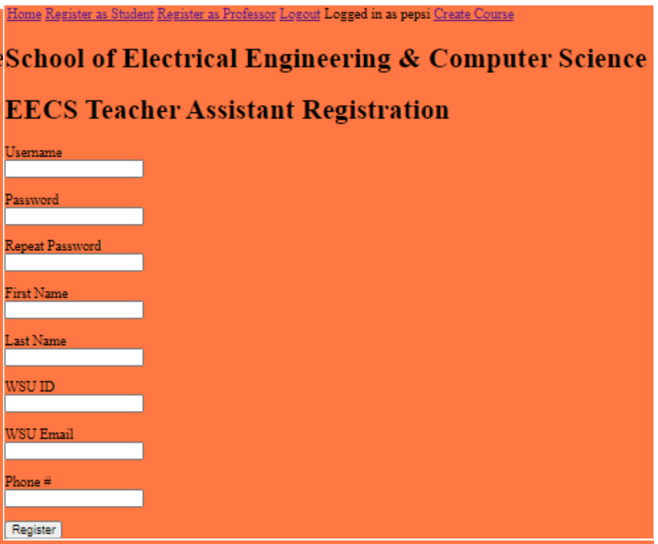
Phone #

Major: Computer Science

Cumulative GPA

Graduation Date mm/dd/yyyy

Register



Home Register as Student Register as Professor Logout Logged in as pepsi Create Course

School of Electrical Engineering & Computer Science

EECS Teacher Assistant Registration

Username

Password

Repeat Password

First Name

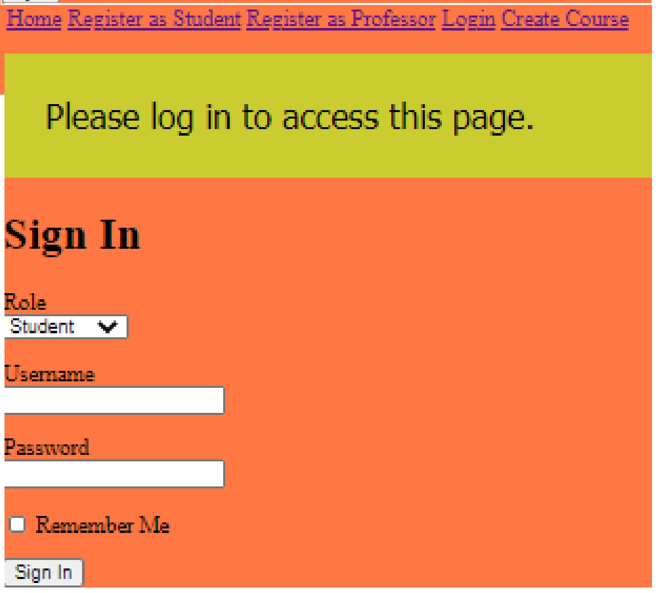
Last Name

WSU ID

WSU Email

Phone #

Register



Home Register as Student Register as Professor Login Create Course

Please log in to access this page.

Sign In

Role: Student

Username

Password

Remember Me

Sign In

IV. Progress Report

Iteration-1: After iteration-1 the application allows the professors to create courses and these courses are displayed for both the students and the professors to see. The students can view these courses in their own login. The application still requires a working solution for students and professors to login independently within user_login function in Flask. Once this is implemented properly the students will be able to apply for courses and the professors will be able to delete courses that they are instructing.

Iteration-2: In this iteration the application now has backend code for the students to see the Apply button and the "apply"/"unapply" routes are created along with the database relationship. The login feature works but only for a single user. After making it work for both users the application will be near completion in terms of the basic use cases such as teacher creates a course, student can apply or un-apply. After this the rest should be relatively simple.

V. Testing Plan

The testing for this will be done using Unit Testing with python. We have already created the folder and files for it. It will be implemented once the database issues are sorted out so that it will not have to be changed again and again. The test files will consist of the '__init__' file, testModels.py, and testRoutes.py to ensure that the main functionality is being properly tested. For the Functionality and UI testing we will be conducting manual tests and attempting to stress each outlier and push the bounds of what both users can do.

VI. References

For the papers you cite give the authors, the title of the article, the journal name, journal volume number, date of publication and inclusive page numbers. Giving only the URL for the journal is not appropriate.

For the websites, give the title, author (if applicable) and the website URL.

Max Points	**Your points**	**Design**
10	10	Are all parts of the document in agreement with the product requirements?
15	15	Is the architecture of the system described, with the major components and their interfaces?
5	5	Is the rationale for subsystem decomposition and the choice for the architectural pattern explained well?
		Are all the external interfaces to the system (if any) specified in detail?
15	8	Are the major internal interfaces (e.g., client-server) specified in detail?

15	10	Are the subsystems that the team has started to implement are described in the document?
10	10	Is there sufficient detail in the design to start Iteration 2?
		Clarity
5	5	Is the solution at a fairly consistent and appropriate level of detail?
3	3	Is the solution clear enough to be turned over to an independent group for implementation and still be understood?
12	12	Is the document making good use of semi-formal notation (UML, diagrams, etc)
5	5	Is the document identifying common architectural or design patterns, where appropriate?
5	3	Is the document carefully written, without typos and grammatical errors?
Total	90	