TeachEzie

Design Document

11/15/2018

Version <2.0>

TeamRSS

Jackson Schuur

Sophia Schuur

Kathy Freund

Austin Ryf

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I. Introduction

The purpose of this design document is to provide a thorough description of the projected final product, as well as provide a medium of which to document changes and further findings as we progress through the iterations.

This document is an overhaul of our design document for iteration1. We edited the entire document to reflect the final product, not just our current work. We adhered to the revision wishes of the TA who previously graded iteration1.

The project provides an optimized, easy to use platform for students and instructors to better access teaching assistant opportunities. An instructor or student may create a new account and perform numerous activities with their accounts, such as logging in, editing them, and viewing their information. Instructors may add new courses for which they require a student to teach, and students may view and apply to TA these courses. Instructors can view and approve of TA applications posted by students, and students will be able to see the status of their applications or cancel them. Our end goal is to meet all requirements of the given project in the form of a nice-looking, well-tested online app.

Section II includes a description of our chosen architectural model and a corresponding UML component diagram. We detail what functions each section performs and why it is important.

Section III includes detailed outlines of all major subsystems. It discusses each component's role in the project. It goes over the routes we implemented in the Backend and some routes we will implement in the future. It identifies the communication between the client and server, what kind of data those messages contain, and in what format they are sent. It discusses the databases we shall implement and UI descriptions and snapshots.

Document Revision History

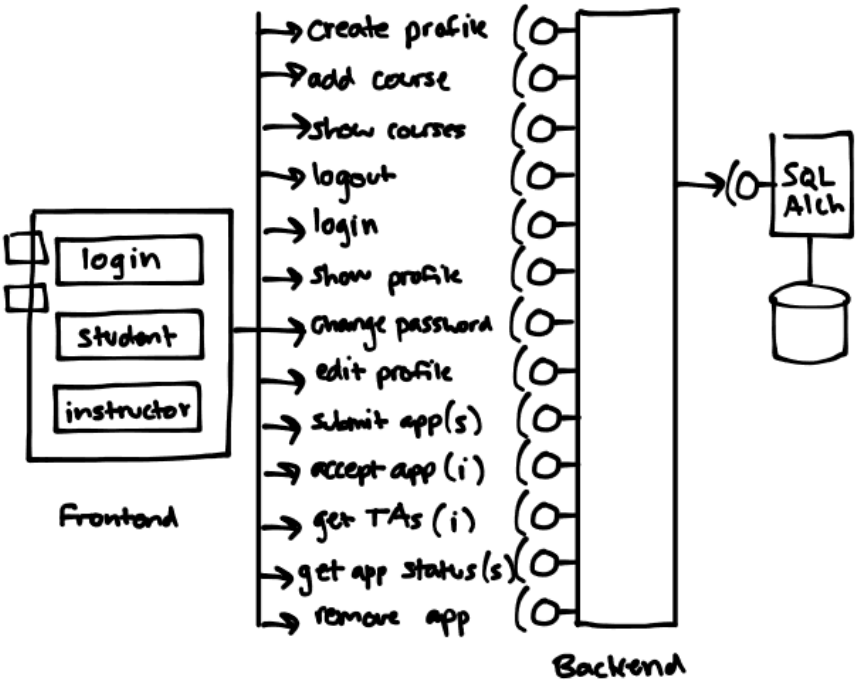
Rev 2.0 <11/5/2018> <Iteration 2 Design Document>

II. Architecture Design

II.1. Overview

For this project, we utilized the Client-Server Architectural Pattern. Similar to the SmileApp Project we completed earlier in the semester, we separated our project into three main components: The GUI, written in HTML and CSS, the Frontend, written in JavaScript, and the Backend, written in Python. The GUI is responsible for displaying information in an appealing manner, the Frontend processes user input data (such as creating a new Instructor or Student profile), and the Backend centralizes all data management and ensures data integrity and database consistency. It also utilizes SQL Alchemy. The Client-Server pattern fits well for a system such as ours where different types of users need to access specific, sensitive data. We also needed a central database of all users and their information, and a dedicated server suited this task best. By utilizing this pattern, we neatly kept the dependencies of the main

components at a minimum by assigning each one a separate, but equally important task. Below is a UML diagram of our Client-Server Architecture.

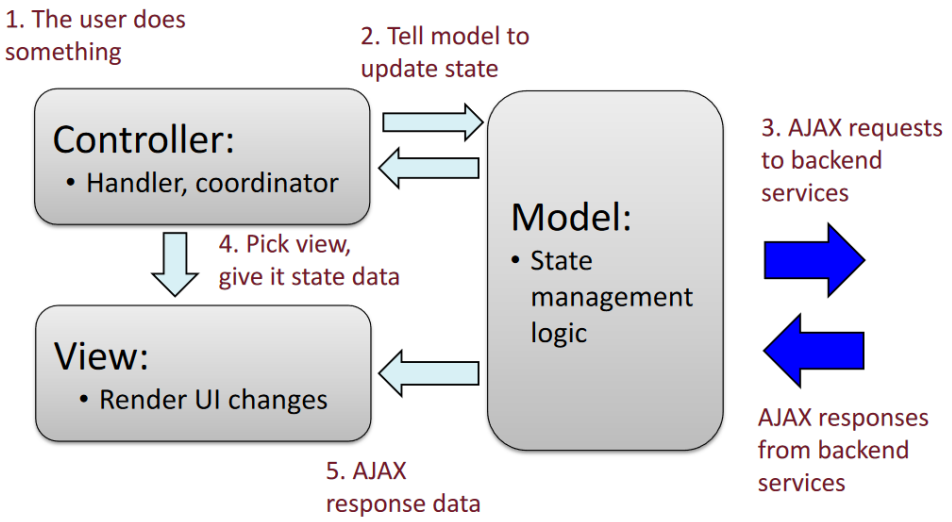


III.1: Design Details

III.1.1: GUI

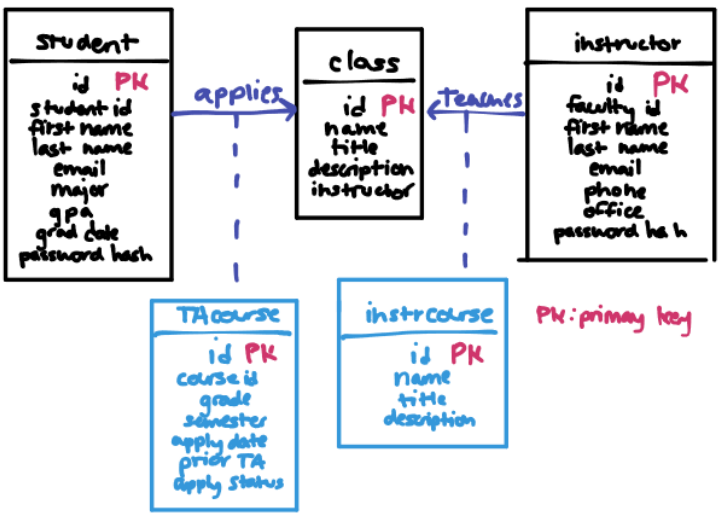
For the GUI, we organized our information neatly into an HTML file and its corresponding CSS file. It has been done completely from scratch. All menus and formats are written on the same HTML file and are hidden and shown in the JavaScript accordingly as a user selects options from the GUI. It contains no hyperlinks to any other part of the project.

III.1.2: Frontend



We decoupled our frontend with adherence to the MVC architectural pattern as well as basic mannerisms to good software design. We made sure that nothing in the JavaScript code ever sees anything in the Backend. The Frontend does not read from or write to the database (Backend) in any way. The Frontend simply receives user information and places it in the correct URL and performs the proper requests. It acts as the controller when a user performs an action or modifies the application state, such as a click. Then it routes to the model to retrieve Backend information, and routes to the view to display the correct HTML/CSS forms.

III.1.3: Backend



The Backend handles all database design. Each user, either a Student or Instructor, and course type, has its own database class and respective attributes. Instructors teach courses and students apply for them. The students may apply to TA a course, in which that course becomes a “TACourse”, and instructors may post courses that need TAing, which become “InstrCourse” (see above UML Class diagram).

Messages between the client and server consist of these POST routes and contain JSON formatted profile data. Responses include a success value web response (200).

1) Create student profile: Create a new Student profile. (POST)
2) create Instructor profile: Create a new Instructor profile. (POST)
3) verifying login (GET/POST)
4) retrieve Instructor and Student profile data (GET)
5) retrieve the list of all CptS courses (GET)
6) Instructors to add new courses in need of a TA (POST)
7) retrieve courses that are not yet assigned TAs (GET)
8) Students to submit a TA application (POST)
9) Instructor to approve an application (POST)
10) Instructor to get the list of the TAs assigned to a class (GET)
11) Student to get the status of his TA application (GET)
12) Student to remove/cancel an application (DELETE).

Routes 1-6 are currently implemented in this iteration.

III.2: Data design

We shall implement five database entry types for each kind of profile. The Student profile, Instructor profile, basic course, TA application course and the Instructor post course. Instructors teach the courses that Students apply to TA for, and decide to pass or deny the course applications posted by Students.

The Student model consists of:

Id: primary key, what the backend uses to identify
StudentID: Different from the ID, just the student’s school id
First name
Last name
WSU Email
Major
GPA
Graduation Date
PasswordHash: Hashed password

The Instructor model consists of:

Id: primary key, what the backend uses to identify
Faculty id: different from the id, just the instructor's id
First name
Last name
Email
Phone number
Office: Field of study of the instructor
PasswordHash: hashed password

The course model consists of:

Id: primary key, what the backend uses to identify
Course name
Title
Description
Instructor who teaches the course

The TA course application consists of:

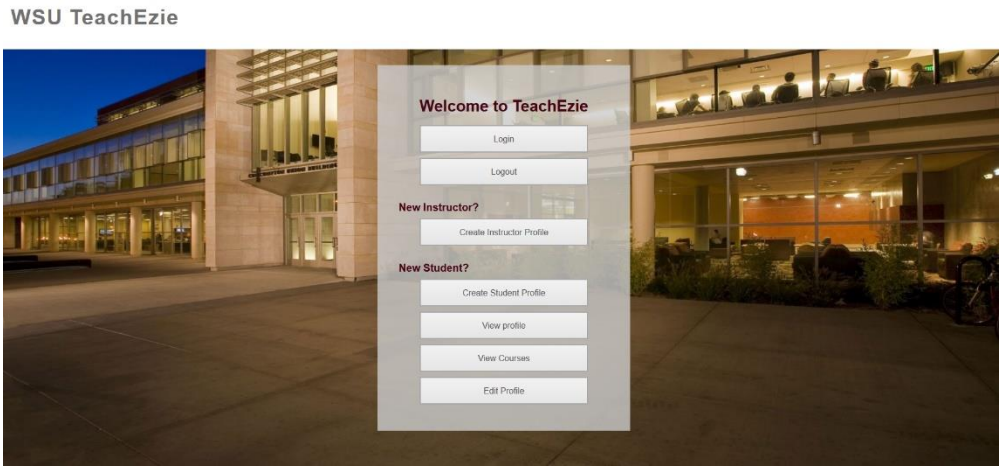
Id: primary key, what the backend uses to identify
Student ID
Course name
Grade earned in course when the student took it
Semester the student took the course
Date the student applied for this TAship
Bool attribute whether or not the student has TA'd before
Status of application (Under review, denied, approved)

The courses that instructors teach consists of:

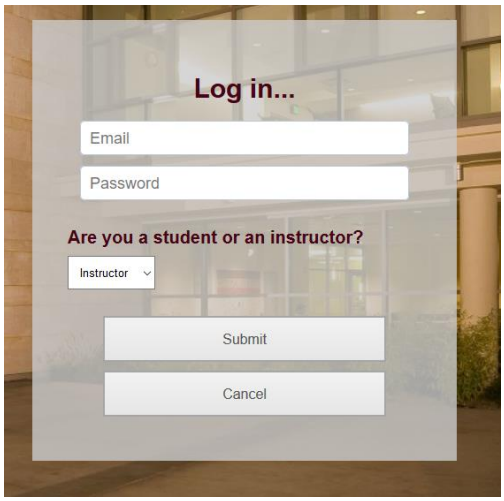
Id: primary key, what the backend uses to identify
Faculty ID
Course name

III.3: User Interface Design

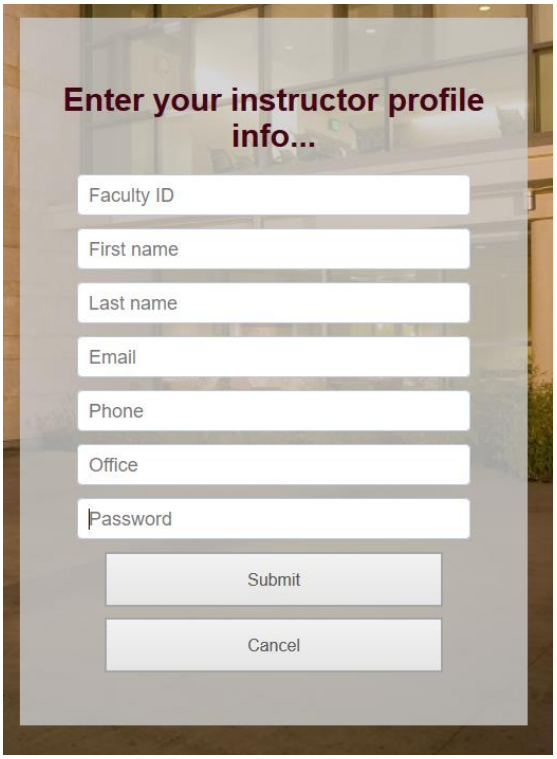
We overhauled our old GUI in favor of a prettier, more modern look. It closely resembles the current MyWSU website. It is user friendly and requires little to no learning curve to understand. Below are some interfaces of our current site.



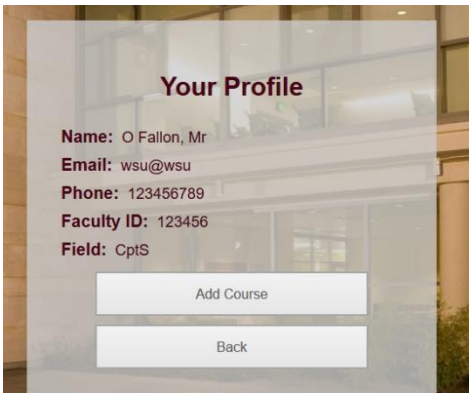
Homepage



Login



New Instructor Profile



View instructor profile

IV: Testing Plan

We shall perform manual testing for this project, no automatic testing. Much of our functional/UI testing includes running the backend, opening the html on a web browser, and generally performing any and all tasks we could in varying sequences, and verifying they were doing what they were supposed to do. This includes everything from making sure every button pressed routes to the correct action or ensuring we display correct/logical information to the user. We open the console and check databases for the correct information we inputted. We debug the JavaScript using the console debugger, and output print statements to the console window.

All API routes are to be tested multiple ways. The complete list of API routes can be found in section III.3. They are to be tested visually, for example, inputting a course and verifying on the webpage that it routed correctly if the course shows up after clicking "View Courses." They are to be tested on Postman. They are also to be viewed on the console window.

V: References

Our references include our own coding experience and background knowledge.

VI: Appendix: Grading Rubric

(Please remove this page in your final submission) (since there exists an entry for this page in the table of contents I will leave it here just in case.)

These is the grading rubric that we will use to evaluate your document.

Iteration-1

Max Points	**Design**
10	Are all parts of the document in agreement with the product requirements?
15	Is the architecture of the system described, with the major components and their interfaces?
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	Are the major internal interfaces (e.g., client-server) specified in detail?
15	Are the subsystems that the team has started to implement are described in the document? Are the algorithms and protocols for those subsystems explained in sufficient detail?

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

Grading rubric for iteration-2

Max Points	**Revision**
15	Did the team revise their iteration1 design document and addressed the issues the instructor commented on?
	Design
	Are all parts of the document in agreement with the product requirements?
5	Is the architecture of the system described, with the major components and their interfaces?
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?
5	Are the major internal interfaces (e.g., client-server) specified in detail?
10	Are the subsystems that the team has started to implement are described in the document? Are the algorithms and protocols for those subsystems explained in sufficient detail?

20	Did the team provided a UML class diagram showing the system's classes, their attributes, operations (or methods), and the relationships among objects? Did the team briefly explain the classes included in the diagram?
5	Is there sufficient detail in the design to start Iteration 3?
	Clarity
5	Is the solution at a fairly consistent and appropriate level of detail?
5	Is the solution clear enough to be turned over to an independent group for implementation and still be understood?
	Is the document making good use of semi-formal notation (UML, diagrams, etc)
	Is the document identifying common architectural or design patterns, where appropriate?
5	Is the document carefully written, without typos and grammatical errors?
	Testing
10	Is there a discussion of how unit testing for the API requests be done automatically? Did the team list the API routes that will be tested and and did the team explain how they will test each route?
5	Is there a discussion of how functional testing will be done?
5	Is there a discussion of how UI testing will be done?