EECS TA Assignment Application

Design Document

11/12/2019

Version <1.24>

Team Epsilon

(Jonathan Coronado, Daniel Hazelton, Jay Kim, Alexandros Papadopoulos)

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I. Introduction

This design document provides a visual overview of our project. The document will show all of the components that make up our application, as well as how they are connected.

Our project is a web application that will be used to assign Teaching Assistants (TAs) to Electrical Engineering and Computer Science (EECS) courses at Washington State University (WSU). The School of EECS at WSU recruits TAs every semester for introductory courses. Currently, this process is done manually, which can be tedious for both students and instructors looking for TA positions. Our web application will, therefore, automate and centralize this process. Students applying to be TAs will register through the application, where they will enter their contact information, course teaching preferences and any additional related information. EECS Faculty will also register through the application in order to search and assign TA positions to the courses/labs that they oversee.

The application will further be split into a Student Side and an Instructor Side:

- For the Student Side, students who are willing to serve as TAs will fill out information where they specify their personal information, academic background and course preferences.
- For the Instructor Side, instructors who are in charge of assigning TAs to their course look at the available student TAs and decide which students fit their course criteria. The instructors then manually match the student chosen for the TA position to their respective courses and/or labs.

Our team will work to refine the requirements, construct the specification and implement the architecture, along with any related modules.

Section II includes the architectural design of the application, along with a brief overview of its significant subsystems.

Section III includes the design details of the application, such as the major components of the system and how they are organized amongst each other.

Section IV includes the testing plan, which describes how the application will be examined in order to ensure that it performs according to specifications.

Section V includes the progress report, where we detail any advancement that was made in the current iteration (Iteration 2).

Document Revision History

Rev	1.00	2019-10-18	Initial Version
Rev	1.01	2019-10-24	Added Introduction Information
Rev	1.02	2019-10-25	Added Architectural Design Information
Rev	1.03	2019-10-25	Added UI Design Details
Rev	1.04	2019-10-25	Added Progress Reports for both Front-End and Back-End
Rev	1.05	2019-10-25	Added Design Details
Rev	1.06	2019-10-25	Added Database Schema
Rev	1.07	2019-10-25	Revised Introduction
Rev	1.08	2019-10-25	Iteration-1 Final Version
Rev	1.20	2019-11-11	Iteration-2 Initial Version

Rev	1.21	2019-11-12	Added Testing Plan
Rev	1.22	2019-11-12	Modified Architecture Design Overview
Rev	1.23	2019-11-12	Updated Progress Reports for Iteration 2
Rev	1.24	2019-11-12	Modified Backend UML

II. Architecture Design

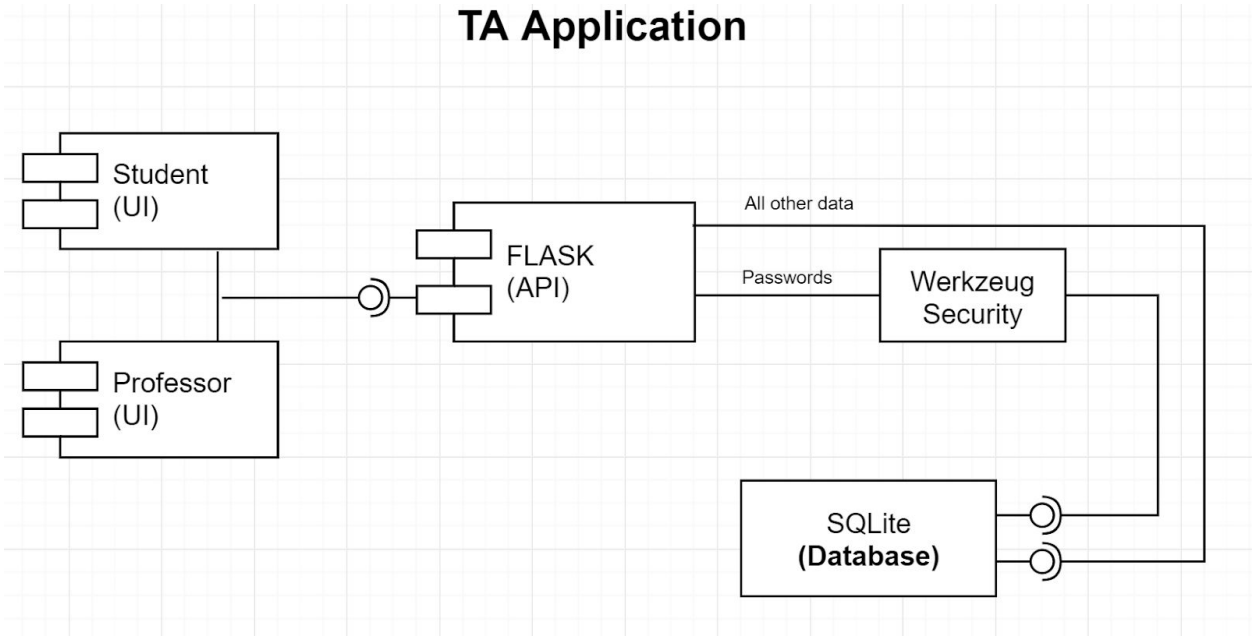
II.1. Overview

We utilized the Client-Server Architectural Pattern for the TA web application. As we hope to achieve a distinct separation between the front-end and back-end code, the Client-Server Architecture seems to fit our needs in the simplest way possible.

The front-end design consists of the Student and Professor (Instructor) UI, as well as the FLASK API (Application Programming Interface). The back-end architecture is divided into Werkzeug Security, for password encryption, and the SQLite database, which stores all user data.

Cohesion and coupling are achieved in an efficient way using this architectural pattern. For example, coupling between Werkzeug Security and the SQLite database ensures that only encrypted passwords are sent to and from the database. In terms of cohesion, the functions within Werkzeug security encrypt and decrypt a given password using a particular encryption key.

II.2. TA Application UML



II.3. Backend UML

UML Components:

Student (UI): The HTML user interface that students will use in the application.

Professor (UI): The HTML user interface that professor will use in the application.

FLASK (API): The API that communicates with the HTML UI and the Database.

GET Routes:

Login: Checks if user exists in the database

Profile Page(s): Gets and displays the profile information onto the web application (for both student and instructor)

POST Routes:

Register: Adds the profile information (username/email and password) to the User table in the database

Edit Profile Page(s): Sends new profile information (WSU ID, first name, last name, phone number) to the User table in the database

Add Course: Sends a new course to the Course table in the database

SQLite (Database): Stores data and information to be accessed later by the application.

Outside Components:

Werkzeug Security: Used to encrypted data passing to and from the database.

WSU Email Services: Service that has an email address that can be accessed by every WSU student and professor.

See Design Details for more information

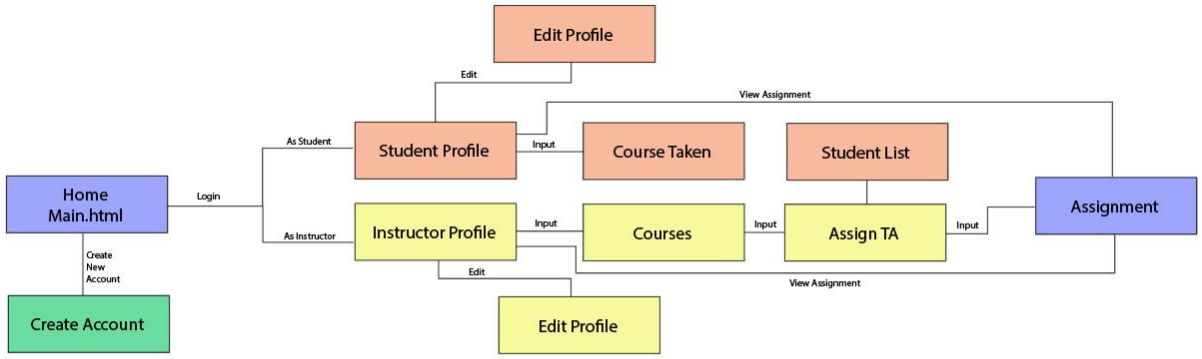
III. Design Details

From the UI (Student and Professor) a GET/POST request is made. This goes to our API which communicates with the database. If it was a post request, data is added to the database. If it was a GET request, data is taken from the database. The data is passed back to FLASK and then is passed back to the frontend. If the data being passed is a password, then the data is passed through Werkzeug Security where the data becomes encrypted using a key before it is sent to the database. When a password is sent from the database to the API it will pass through Werkzeug Security first to decrypt.

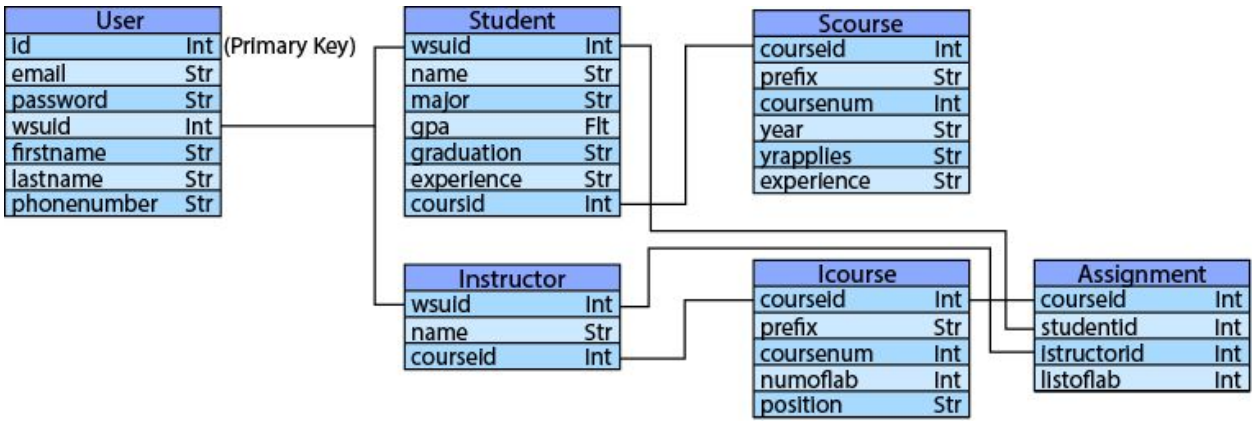
The client and server will communicate primarily using JSON data. The primary messaging between the client and server will consist of a GET request when a user logs into the application and a POST request when the user creates a new account. The Login page will utilize a GET request upon confirmation that the username and password are in the server, where it will obtain any of the additional user information previously entered by the user (and stored in the database). The Sign-Up page will issue a POST request once a user has entered the desired information, whereby the data will be sent to the server to be

stored in the database. In turn, the server will send JSON back to the client with the appropriate information: either the user information that was requested in the GET Login request or a status update confirming that the designated information was received by the POST request

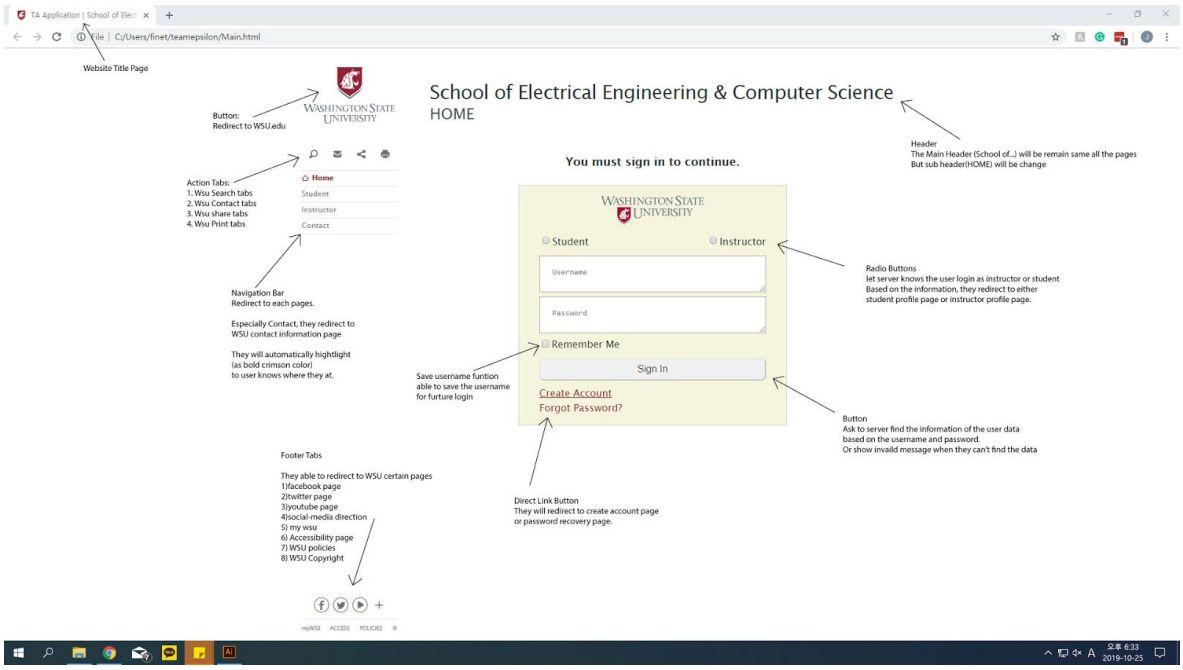
III.1.1. App UML



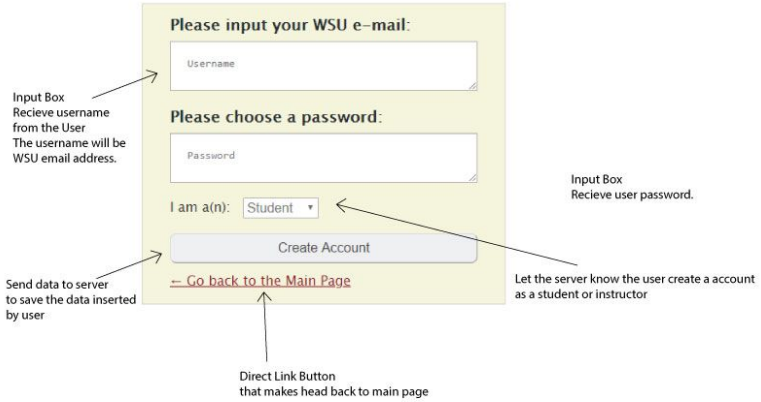
III.1.2. Backend UML



III.2. User Interface Design



Use case #2 and Use case #5



Use case #1 and Use case #4

IV. Testing Plan

Unit Testing:

We will guarantee that:

- Both radio buttons cannot be selected at the same time
- The “Login” button takes the user to their profile page
- The “Register” button takes the user to the login (home) page
- When a student is clicked in the “Assign TA” page, their information will automatically be shown

Functional Testing:

We will test the use cases (provided in our Requirements Document), specifically:

- Use Case(s) #1 and #4 (Create Account for both Student and Instructor): we will enter user information via the Create Account page and then manually check, using DB Browser for SQLite, whether the submitted information is in the database
- Use Case(s) #2 and #5 (Login Page for both Student and Instructor): we will test to make sure that the web application recognizes correct user login information and successfully takes them to their profile page
- Use Case(s) #3 and #6 (Edit Account Information for both Student and Instructor): we will ensure that the user is able to edit their information, that the modified information is sent to the database and appears in the data tables, and, finally, that the web application updates with the modified data

UI Testing:

We will observe the effect of our testing on the HTML pages by looking for the expected changes when we:

- Provide a username/email that is already registered in the database in order to make sure that the username/email is unique
- Have different strings in the “Password” and “Repeat Password” text fields
- Successfully input login information that is in the database
- Select “Remember Me” in order to have the web application bypass login for a selected user

V. References

https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

https://github.com/washingtonstateuniversity/WSU-spine

https://brand.wsu.edu/visual/colors/

https://wsu.edu/about/contact/

https://school.eecs.wsu.edu

https://brand.wsu.edu/

https://bit.ly/2qL0xNm

https://wsu.edu

VI. Progress Report

Front-end:

In Iteration 2, we initially created individual branches for the front-end and back-end code in an attempt to organize and separate the two. We then completed the HTML and CSS design for two of the Instructor pages. The first page that was modified allowed the instructor to add courses that they were planning to teach. The instructor would manually input the desired course, then specify how many sections were in the course. Once confirmed by the instructor, the course (and its sections) would appear at the bottom of the page as clickable buttons. Clicking any of the buttons would take the instructor to the second modified page in this iteration. This page handled the assigning of TAs to the course/section selected by the instructor. However, the aforementioned front-end/back-end branch split ended up backfiring due to our flexible work strategy. After forgetting to push the modifications of the pages mentioned above to the front-end branch, we switched to working on the back-end branch in an attempt to eventually merge the two together. However, when we fetched the code currently in the back-end branch, we ended up overwriting all of the changes made to the pages in the front-end (due to the fact that the back-end branch only had the unmodified skeleton pages from the front-end). From this loss of progress, we now have committed to ensuring that our code can (and will) be pushed onto GitLab at regular intervals in order not to repeat this mistake again.

Back-end:

Near the start of iteration 2, we attempted to update our iteration 1 by following tutorials online. The plan was to try and create a backend that was easier to understand and worked with our CSS file. After attempting to follow the outdated, confusing, and sometimes incorrect tutorials for 5 hours we failed in our attempts with making a new backend. However, by scraping together bits and pieces of knowledge from the tutorials, we were able to better understand how to update our backend as well as get it to work with CSS. After that fiasco, we moved on to updating the code to fulfill the requirements we set for this iteration. Our first priority was to provide a way to differentiate between student and instructor profiles. we achieved this by adding radio buttons. They required a choice between “student” and “instructor” before creating an account and logging in. When logging in, the website checks the account type and loads the corresponding profile. When we created the “type” column for user, we accidentally made the column store only unique values. This made it so only one account of each type could be made. This is fixed now. Next we created separate profile pages for student and instructor. They both listed the user’s first name, last name, wsu id, and phone number (We also added these as columns for user in the database). A button/link that opened the edit profile page was also added. The edit profile

page allowed the user to edit the previously listed values in the database. After the changes were saved, the profile page would get the data from the database and updated it's information accordingly. The last thing we added was the option to see and add courses to the course list. We created a new table called "courses" and added that instructor users could see the list of courses in the database on their profile. Then, we added the option to add a new course to the database on another page which could be accessed from the instructor profile.