TA Portal

Design Document

11/18/2020

Version 2

Team 10

Logan Gorence

Marcus Sagisi

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I. Introduction

This design document provides a number of high and medium level descriptions of the subsystems that make up our project. This is the initial revision of the document, which covers the basic subsystems, libraries that we use, URL routes, database schema, and a progress report. In iteration 2, this design document will be updated to include other information.

The goal of this project is to provide an easy-to-use application that will allow EECS students and faculty to easily find TA positions for classes. Prior to the beginning of the semester, a professor will enlist the courses that they are currently teaching on their profile. A student interested in being a TA will create a profile, and fill out a survey that will gather information from them and what classes they are interested in. The motivation of creating this application is to reduce the amount of labor required to manually collect survey results, validate them, and provide lists to professors to choose a TA.

Section II includes a high-level description of the subsystems and the architecture that was used to build our application. Section III includes implementation details of each subsystem and the components used in our application. Section IV contains a description of the progress in our project up until the end of the current iteration. Section V contains information about testing. Section VI contains any references that we need to include.

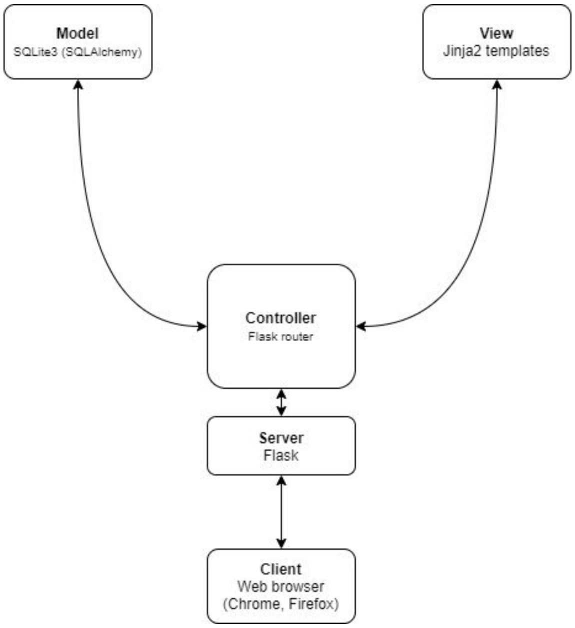
Document Revision History

Rev 1	2020-10-30	Initial version
Rev 2	2020-11-18	Update for iteration 2

II. Architecture Design

II.1. Overview

The architectural pattern that we adopted in our project is model-view-controller (MVC). MVC is typically used in applications that have a user interface and interface with some kind of persistence layer. Because our application provides a web-based user interface and persists data into a SQL database, MVC was an obvious choice for our project. MVC allows us to lower the coupling between units in our application, while increasing our cohesion to create components that are reusable and clean.



Block diagram of our application architecture

Because our application is utilizing MVC, we need a few different libraries to represent the different subsystems that we are building. Our model subsystem utilizes SQLAlchemy to create database tables, and uses SQLite3 as a backend, which is a flat file database. For our view, we utilize Jinja2 for a templating language, which Flask can render for us. Lastly, for our controller / router / web server, we use Flask. Additionally, we're using wtforms for form generation and werkzeug for security and password hashing.

III. Design Details

III.1. Subsystem Design

III.1.1 Model (database)

The model in our application persists all the relevant data that is required to run the application. For example, we store user profiles for students and professors, courses, applications, etc. This subsystem will not depend on any other subsystem to reduce coupling. For our application, we use SQLAlchemy that stores data in a SQLite3 flat file database. If we wanted to properly scale this application in the future, PostgreSQL or MySQL might be better choices to better handle failure and load. In a section below, we give rough descriptions of the tables that we are using to persist the data in our application.

III.1.2 Controller (router)

The router is a part of Flask which will parse incoming requests and route them to specific locations (routes) that we have designated. For example, if you register the route */student/register*, and a web browser navigates to that URL, the web server will serve the designated content as per the route. Because our router takes incoming requests, we pass the requests onto their designated routes and those routes are part of our controller. Additionally, the controller creates the business logic that ties together the model and view.

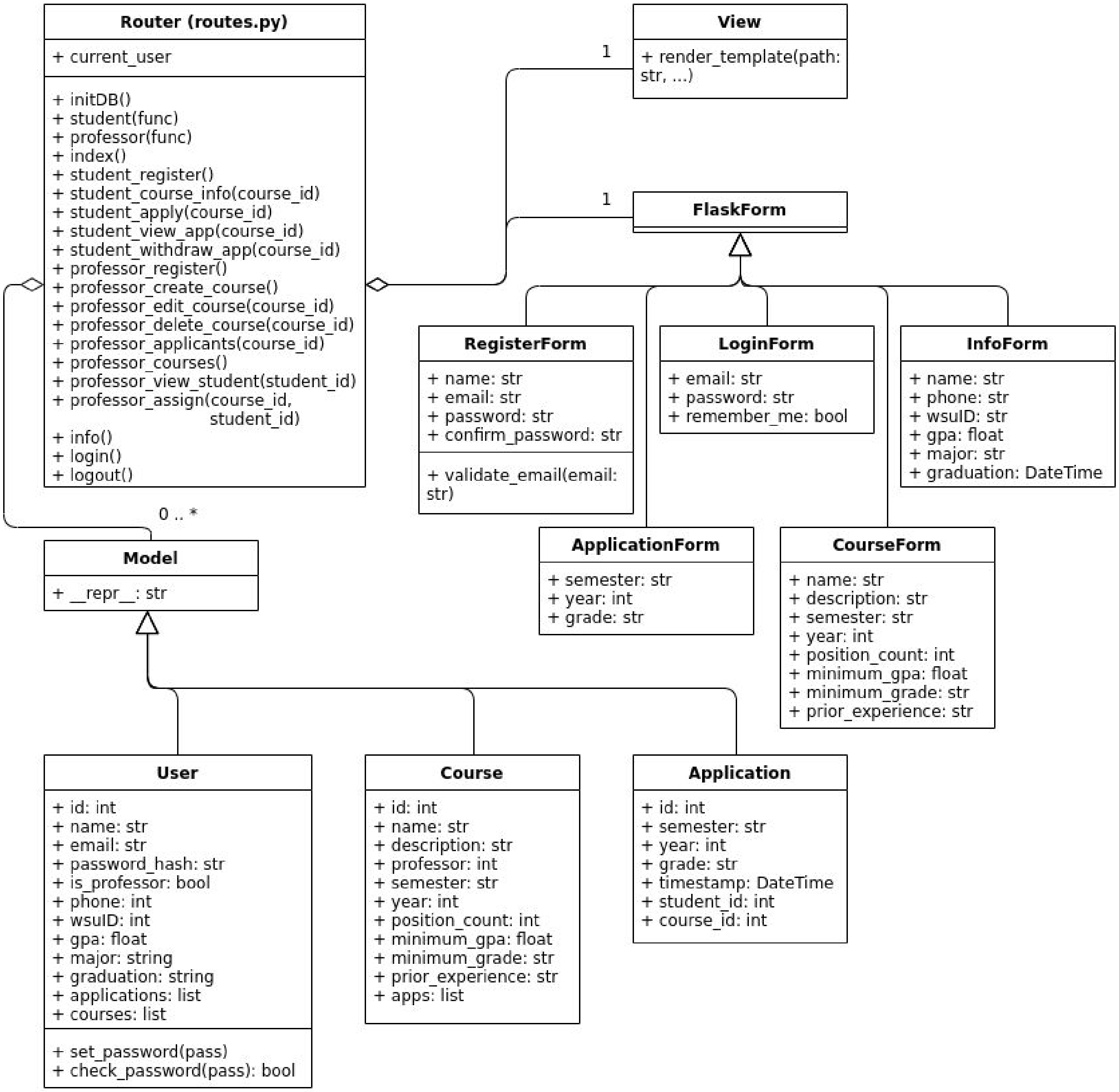
Methods	URL	Description

GET	/, /index	Homepage of our application, provides links to all the various operations that a student or professor could perform.
GET, POST	/student/register, /professor/register	Page to register as a student or professor, respectively.
GET, POST	/login	Page to login and create a session.
GET	/logout	Page to logout and destroy the existing user session.
GET, POST	/info	Edit currently logged in user’s information.
GET, POST	/student/experience	A student is able to edit their past TA experience.
GET	/student/positions	Student can view open TA positions for all courses.
GET	/student/position/{id}	Student can view the details about an open TA position.
GET, POST	/student/apply/{id}	Student can apply to an open TA position.
GET	/student/withdraw/{id}	Student can withdraw from a TA position that they have applied for.
GET	/professor/courses	View courses that a professor owns.
GET, POST	/professor/courses/create	Form for a professor to add a course with.
GET, POST	/professor/courses/edit/{id}	Form for a professor to edit an existing course with.
GET	/professor/course/{id}	View information about a course and see students that have applied, with links to their information.
GET	/professor/student/{id}	Professor can view experience, contact information, etc for a student.
POST	/professor/course/{id}/assign/{studentId}	Assigns a student to a specific course.

III.1.3 View (templates)

In our application, we are utilizing Jinja2 templates with Flask to create reusable templates that will allow us to build our application frontend quickly. The view subsystem reduces coupling by only interacting with the models that are provided to it by the controller. Additionally, we are using wtforms in addition to Jinja2 and Flask to provide easily created forms for the frontend of the application.

III.1.4 Class Diagram



III.2. Data design

III.2.1 User

Column	Type	Description
id	Int	Primary key to identify the user by
name	String(70)	Name of the user
email	String(120)	Email address of this user
password_hash	String(128)	Password hash for this user (uses werkzeug security package

		to perform the hash and do checking)
is_professor	Boolean	Boolean flag that indicates whether this user is a professor or a student
phone	Integer	Phone number of the user
wsuID	Integer	WSU ID number of the user
gpa	Float	Cumulative GPA for a student
major	String	String representation for a student’s major
graduation	DateTime	String representation for a student’s graduation date
applications	relationship	If the user is a student, this relationship is between the student and their application(s)
courses	relationship	if the user is a professor, this relationship is between the professor and their course(s)

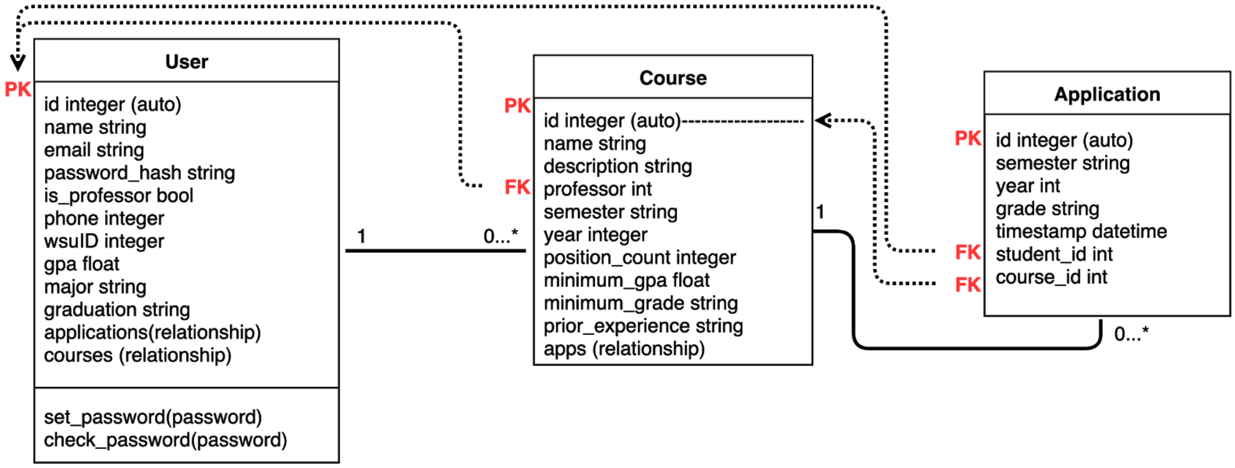
III.2.2 Course

Column	**Type**	**Description**
id	Integer	Primary key to identify the course by
name	String	Name of the course
description	String	Description of the course
professor	Integer	Foreign key to user.id
semester	String	Semester the course is going to take place
year	Integer	The year the course is going to take place
position_count	Integer	The number of open positions of the course
minimum_gpa	Float	The minimum GPA for the course
minimum_grade	String	The minimum grade for the course
apps	relationship	The relationship between this course and the applications it has received

III.2.3 Application

Column	Type	Description
id	Integer	Primary key to identify the application by
semester	String	The semester the student has taken this course
year	Integer	The year the student has taken this course
grade	String	The grade the student received for this course
timestamp	DateTime	The date and time the application was submitted
student_id	Integer	Foreign key to user.id
course_id	Integer	Foreign key to course.id

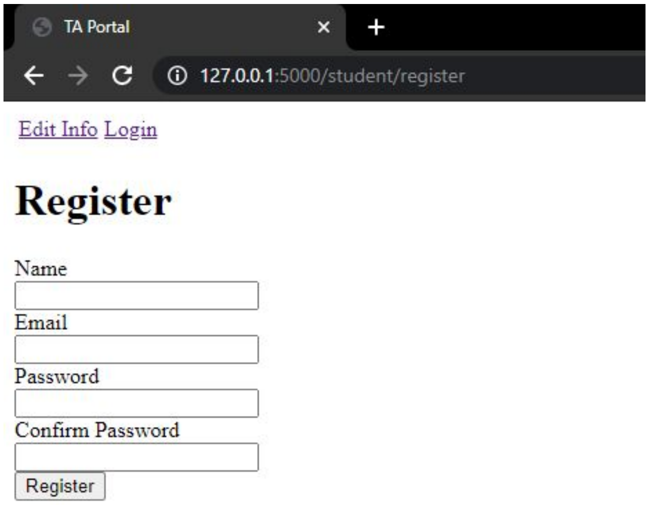
III.2.4 Database Design



III.2.4. User Interface Design

Use case: Create/Register account

For registering for both student and professor, the user only needs a valid email address, name, and password. Students and Professors have different URLS, so the software can differentiate if the user is a student or a professor



TA Portal

127.0.0.1:5000/student/register

Edit Info Login

Register

Name

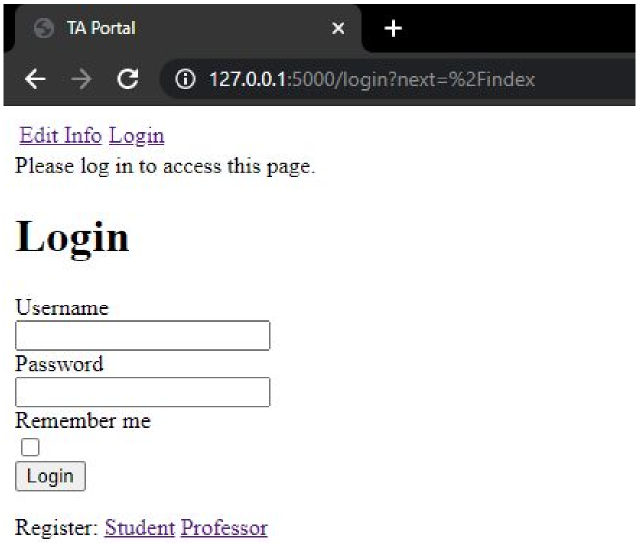
Email

Password

Confirm Password

Use case: Login

The user needs to have an account and be logged in to that account in order to access the features of the app. The user only needs their email and password to log in. In the case the user does not have an account, the login page shows a student and professor registration links, that redirects the user to the registration page



TA Portal

127.0.0.1:5000/login?next=%2Findex

Edit Info Login

Please log in to access this page.

Login

Username

Password

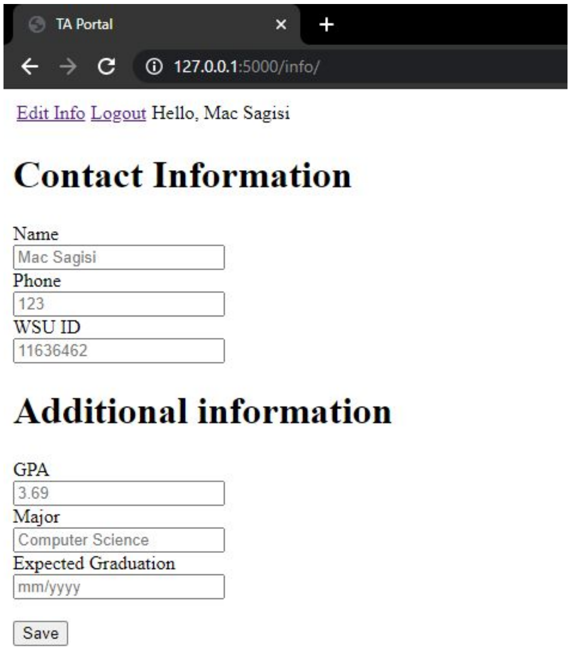
Remember me

□

Register: Student Professor

Use case: contact/additional information

For the student user, the student is able to input both contact and additional information, while the professor users can only input contact information. The form does have input placeholders so the user knows what their information already is. If they haven't already edited their information, it shows



TA Portal

127.0.0.1:5000/info/

Edit Info Logout Hello, Mac Sagisi

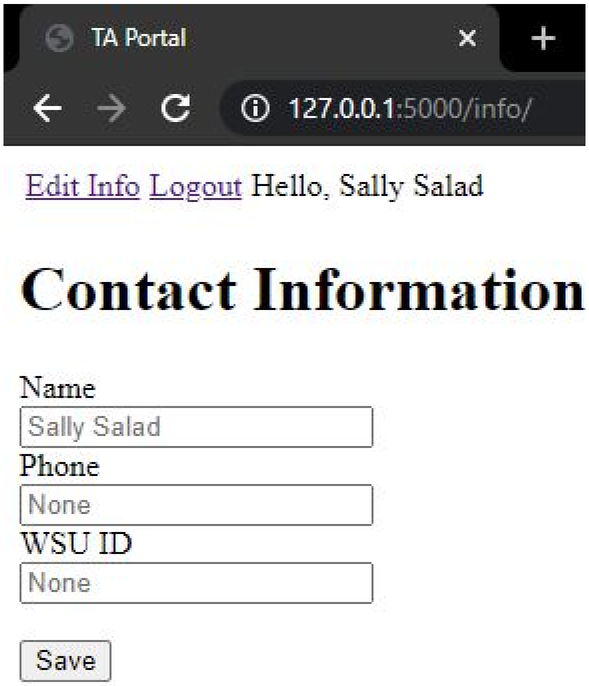
Contact Information

Name
Mac Sagisi
Phone
123
WSU ID
11636462

Additional information

GPA
3.69
Major
Computer Science
Expected Graduation
mm/yyyy

Save



TA Portal

127.0.0.1:5000/info/

Edit Info Logout Hello, Sally Salad

Contact Information

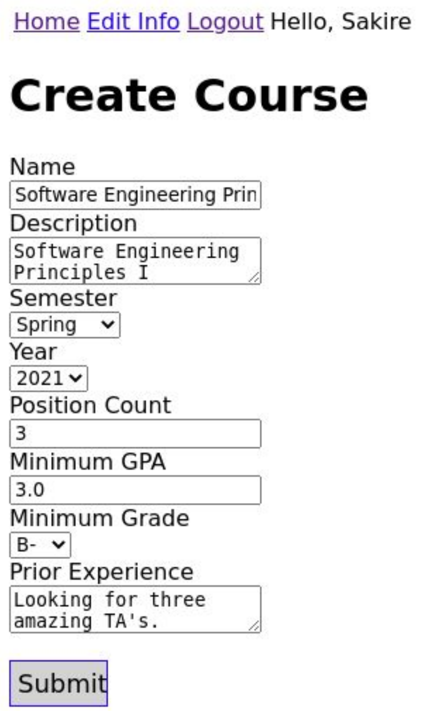
Name
Sally Salad
Phone
None
WSU ID
None

Save

None.

Use case: “add or edit current courses” and “edit course metadata”

Professors should be able to create, edit, and delete courses from the database. Students should not be able to access these pages and create courses, but they should only be able to view courses on their respective pages.



Hello, Sakire

Create Course

Name: Software Engineering Prin

Description: Software Engineering Principles I

Semester: Spring

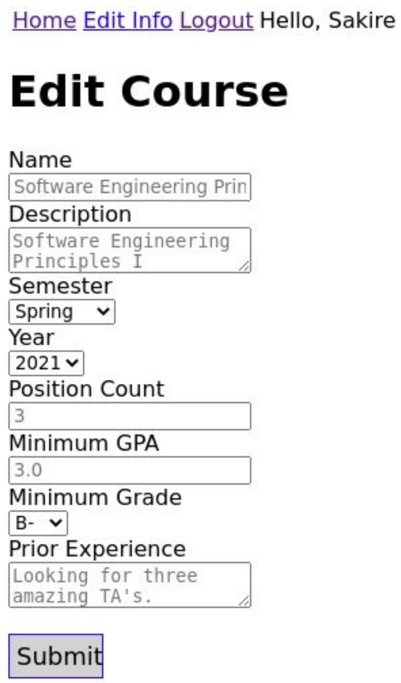
Year: 2021

Position Count: 3

Minimum GPA: 3.0

Minimum Grade: B-

Prior Experience: Looking for three amazing TA's.



Hello, Sakire

Edit Course

Name: Software Engineering Prin

Description: Software Engineering Principles I

Semester: Spring

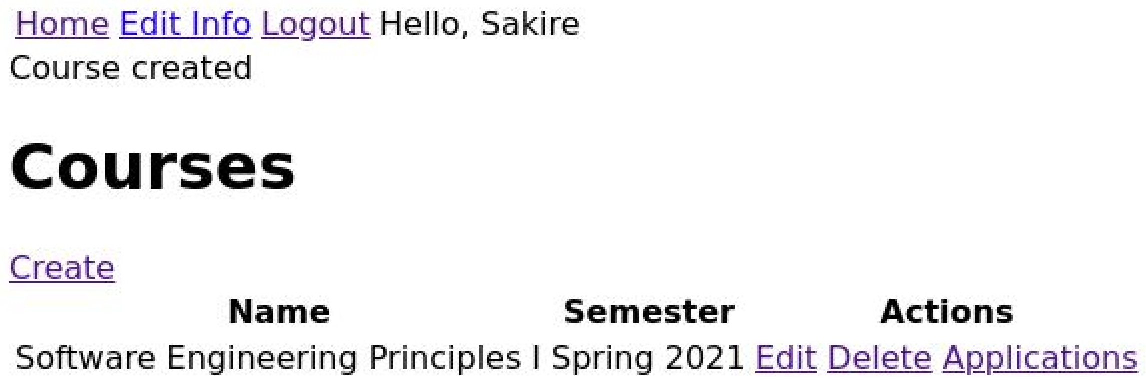
Year: 2021

Position Count: 3

Minimum GPA: 3.0

Minimum Grade: B-

Prior Experience: Looking for three amazing TA's.



Hello, Sakire

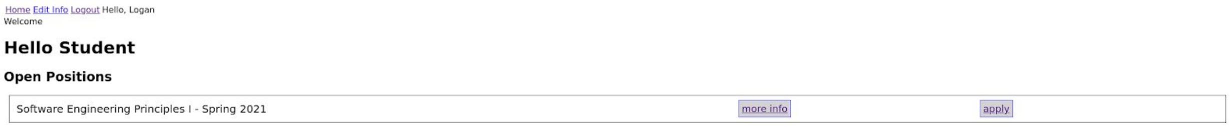
Course created

Courses

Name	Semester	Actions
Software Engineering Principles I	Spring 2021	Edit Delete Applications

Use case: “View open positions”, “Display position information”, and “Apply for position”

Students should be able to view open positions, information about those positions, and be able to apply for those positions.

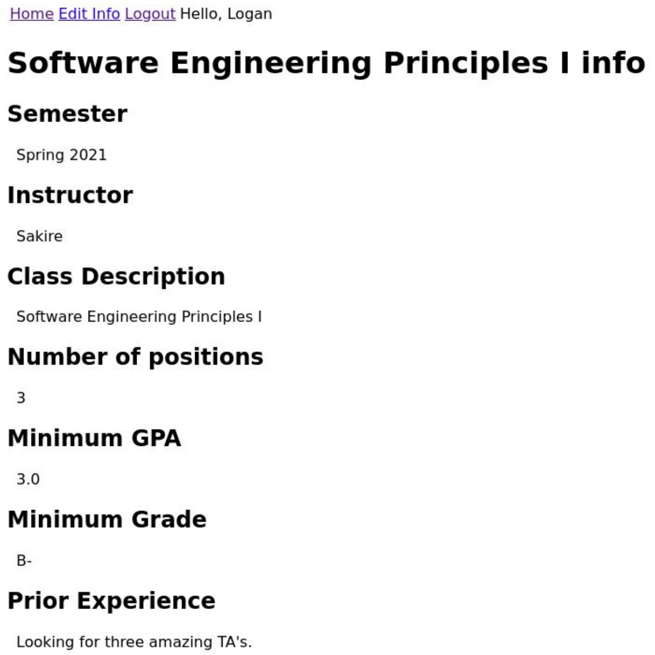


Home Edit Info Logout Hello, Logan
Welcome

Hello Student

Open Positions

Software Engineering Principles I - Spring 2021	more info	apply



Home Edit Info Logout Hello, Logan

Software Engineering Principles I info

Semester

Spring 2021

Instructor

Sakire

Class Description

Software Engineering Principles I

Number of positions

3

Minimum GPA

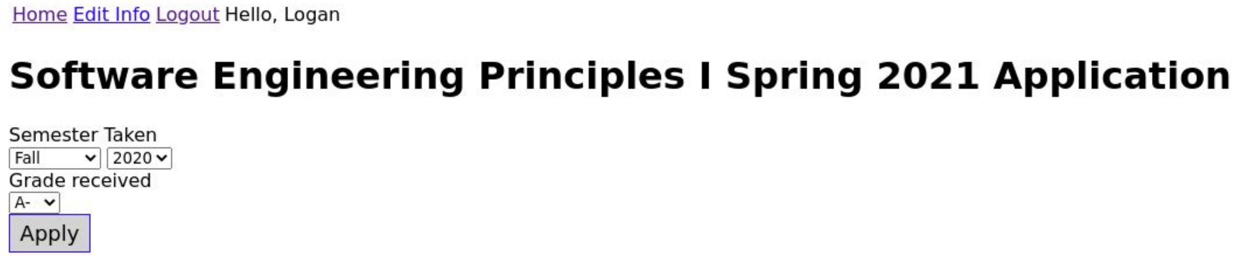
3.0

Minimum Grade

B-

Prior Experience

Looking for three amazing TA's.



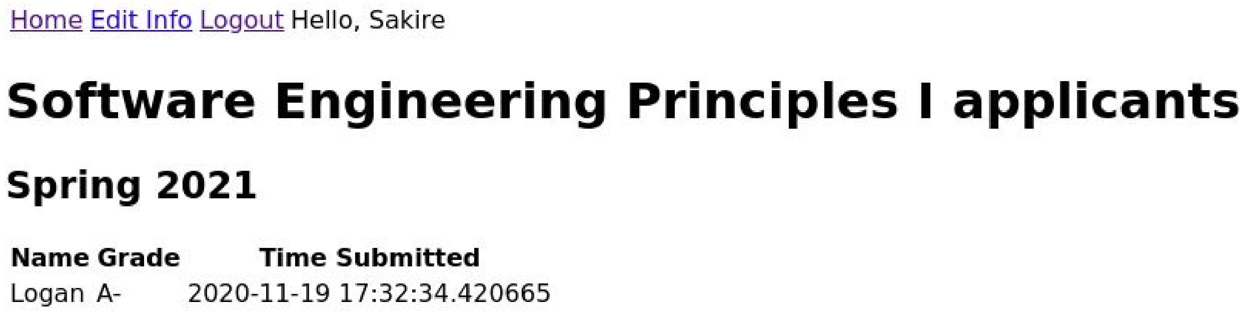
Home Edit Info Logout Hello, Logan

Software Engineering Principles I Spring 2021 Application

Semester Taken
Fall 2020
Grade received
A-

Use case: “View students that have applied”

Professors should be able to view the applications for one of their courses, and in the future, should be able to browse to a page that describes the student and provides their information.



Home Edit Info Logout Hello, Sakire

Software Engineering Principles I applicants

Spring 2021

Name	Grade	Time Submitted
Logan	A-	2020-11-19 17:32:34.420665

IV. Progress Report

For iteration 1, we were primarily focused on having the user have an account and edit their information, both professors and students. The professors and students share the same User class, but the functionality of the user is based on the “is_professor” boolean value. For example, in our info page, both students and professors are able to change their contact information, but the students are able to see and edit their additional information, such as GPA or expected graduation. What we worked on, was to set up the foundation of our app.

For iteration 2, we focused on building up the rest of the database models, such as courses and applications. Then, we built the user interface that shows all of the models to the user. Most of our work went into building the forms and tying them with the models together to form the controller so that we can modify the contents of the database. This includes being able to professors being able to create courses, edit courses, view applicants, and students seeing open positions and apply for them.

V. Testing Plan

For our application, we plan to have unit tests that cover the basic API routes that change the state of our application. During research, we found that Flask has an article in their documentation that covers testing Flask applications with the *pytest* unit testing library (Testing Flask Applications). It looks like a reasonable way to build automated unit tests for our application. Otherwise, we are going to use manual testing for functional tests and UI tests, as with the size of our project and the time commitment required, it likely would not be worth it to learn how to use a headless web browser like Selenium.

VI. References

Flask developers. “Testing Flask Applications.” *Testing Flask Applications - Flask Documentation (1.1.x)*, Flask Developers, flask.palletsprojects.com/en/1.1.x/testing/.