BigDaddy

Design Document

11/12/19

Version 3.0

BigDaddy

Puthypor Sengkeo

Nick Lamos

Daniel Chia

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I. Introduction

This design document is to help us stay on track of building our project. It will overview how we organize our code, what our final project will look like, and how we will create our project. This first revision will help us to have a clear idea where we stand at the end of the first iteration, and what we will need to do for the second iteration.

We are making a website where a professor is able to login and view what students would like to TA for them. The students will also be able to login to apply for TA positions. We are currently working on getting the basic layout of it completed.

Section II includes what the major components of our website are as well as a UML component diagram explaining it.
Section III includes a more indepth look at how the smaller elements of our website will function and what data they will submit.

Document Revision History

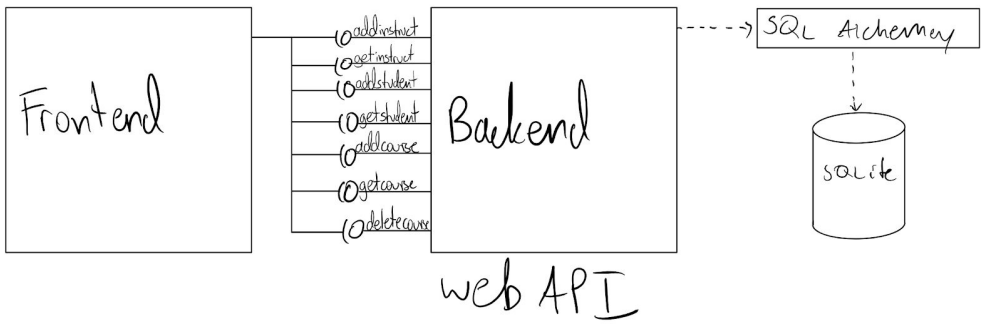
Rev 1.0 2019-10-29	Initial version
Rev 1.1 2019-11-10	Fixed issues from graded version
Rev 2.0 2019-11-11	Added what we did for iteration-2
Rev 3.0 2019-12-02	Final revision, fixed mistakes from previous version and added more content

II. Architecture Design

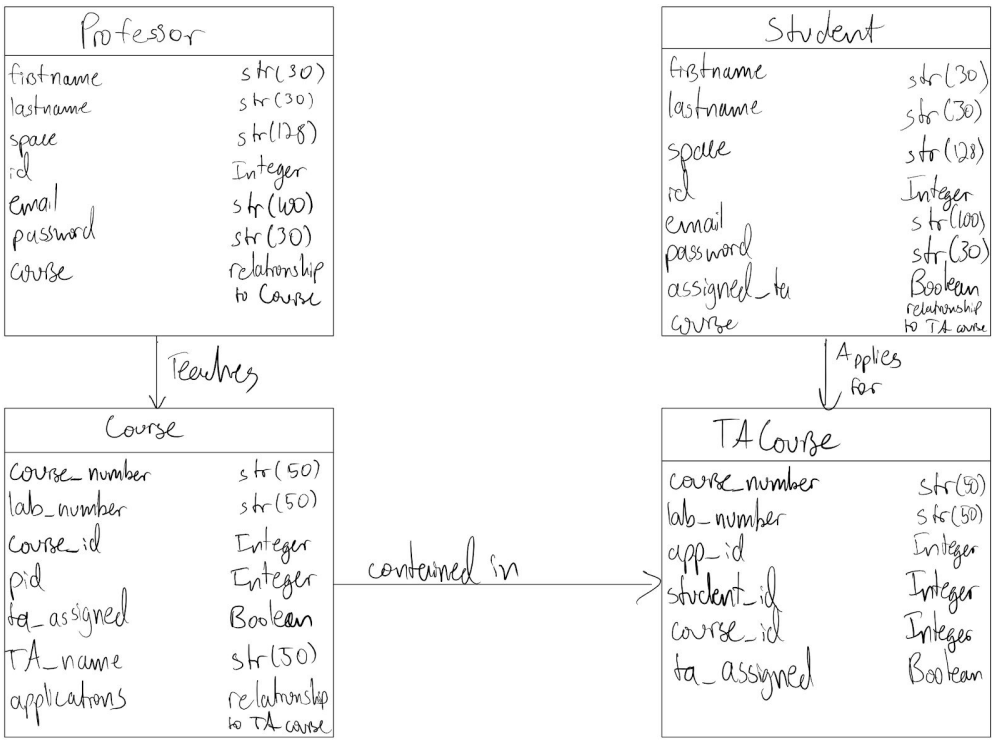
II.1. Overview

The major components of our system are the frontend and backend. They fit together by using Google's AJAX API to communicate, as well as hosting the backend data with SQLite.

- The architectural pattern we adopted for our system is the Client-Server Pattern. We think this will work well for us since it's a great pattern for have a client interact with the frontend like we have, then using our backend to handle all of the data and requests. This way we can keep the two seperate from one another.
- We have low coupling since our website will only pass JSON data between modules. This keeps everything separate and does not impact another system by each route handling their own data. We have high cohesion because all of our modules will operate on the same type of data. They are are doing similar tasks, just slightly different depending on if it is an instructor or student.
- UML component diagram



- UML class diagram



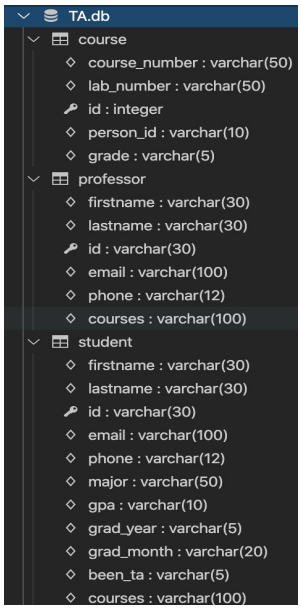
- The frontend handles what the consumer will see and send their requests to the backend. The backend provides different API paths for the frontend to use. It also handles all JSON data passed to it from the frontend and stores it in SQLite. The URLs for the routes we had for actively joining our frontend and backend as well as using for testing are

  /loginstudent?username=&password=/
  /loginprofessor?username=&password=/
  /allcourses/
  /professorcourses?username=/
  /studentapplications?username=/
  /applications?course=&lab=/
  /addinstructor/
  /editinstructor?username=/
  /addstudent/
  /editstudent?username=/
  /addTAcourse/
  /deleteTAcourse?course=&lab=/
  /addcourse/
  /deletecourse?course=&lab=/
  /addProfessorCourse?username=&password=/
  /addStudentTAcourse?username=&password=/
  /assignTA?username=&password=/

III. Design Details

III.1. Subsystem Design

The subsystems we have are addinstruct, getinstruct, addstudent, getstudent, add course, getcourse, and deletecourse.



- We will organize our client JavaScript code using MVC pattern decoupling the view from the model as much as possible.

III.2. Data design

III.2.1[addprof]

```
{
   "professor": {
      "firstname": "Best",
      "lastname": “Teacher”,
      "id": "11557485",
      "email": "best.teacher@wsu.edu",
      "phone": “5093351923”,
      "courses": "-1"
   },
   "status": 1
}
```

III.2.2[getprof]

```
 {
  "status" : 1,
  "professors": [
                  {  firstname: "Best",
                     lastname: “Teacher”,
                     id: "11557486",
```

```
                    email: "best.teacher@wsu.edu",
                    phone: “5093351923”,
                    courses: "CptS322" “CptS355”,
                }
                ...
        ]
}
```

III.1.3[addstudent]

```
{

  "student": {
    "firstname": "Best",
    "password": "Password123!",
    "lastname": “Student”,
    "id": "11562931",
    "email": "best.student@wsu.edu",
    "phone": “5093351923,
    "major": "Computer Science",
    “gpa” : “4.0”,
    “grad_year” : “2019”,
    “grad_month” : “May”,
    “been_ta” : “yes”
  },
  "status": 1
}
```

III.1.4[getstudent]

```
{

  "status" : 1,
  "students": [
                {  "firstname": "Best",
                    "lastname": “Student”,
                    "id": "11562931",
                    "email": "best.student@wsu.edu",
                    "phone": “5093351923,
                    "major": "Computer Science"
                    “gpa” : “4.0”,
                    “grad_year” : “2019”,
                    “grad_month” : “May”,
                    “been_ta” : “yes”,
                    “courses” : “CptS121” “Cpts122”
                }
                ...
        ]
```

}

III.1.5[addcourse]

```
{
"person": {
   "id": “11557486”,
   "courses": “CptS355”
},
"course": {
   "course_number": "CptS322",
   "id": 60,
   "person_id": "11557485",
   “grade” : -1
},
"status": 1
}
```

III.1.6[getcourse]

```
 {
   "status" : 1,
   "courses": [
            { course_number: "CptS322" “CptS355”,
              lab_number: “-1”,
              id: 59,
              person_id: "11557486",
              grade: -1
            }
              ...
          ]
 }
```

III.1.7[deletecourse]

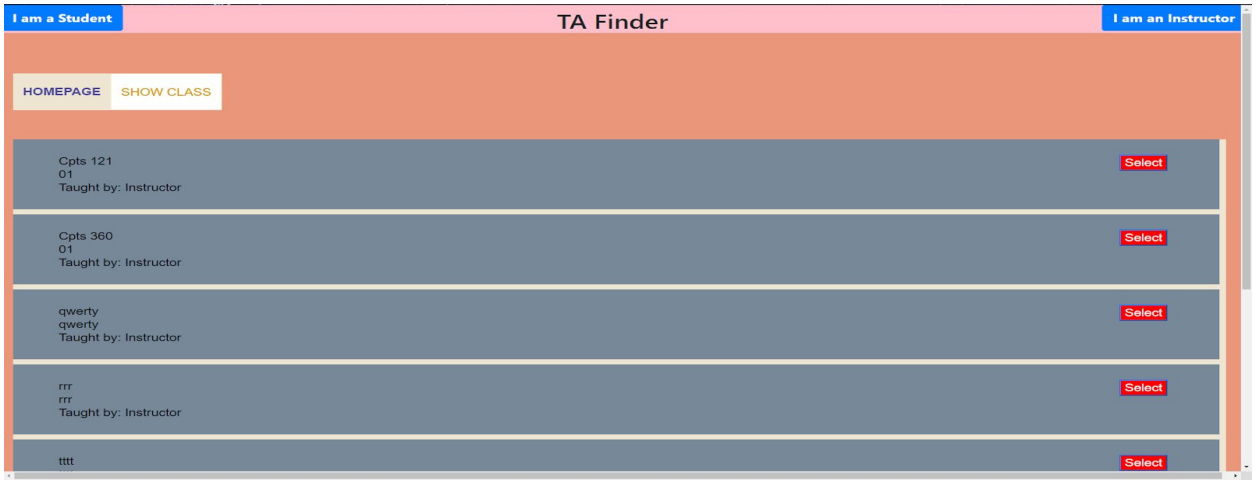
```
 {
     "status": 1
 }
```

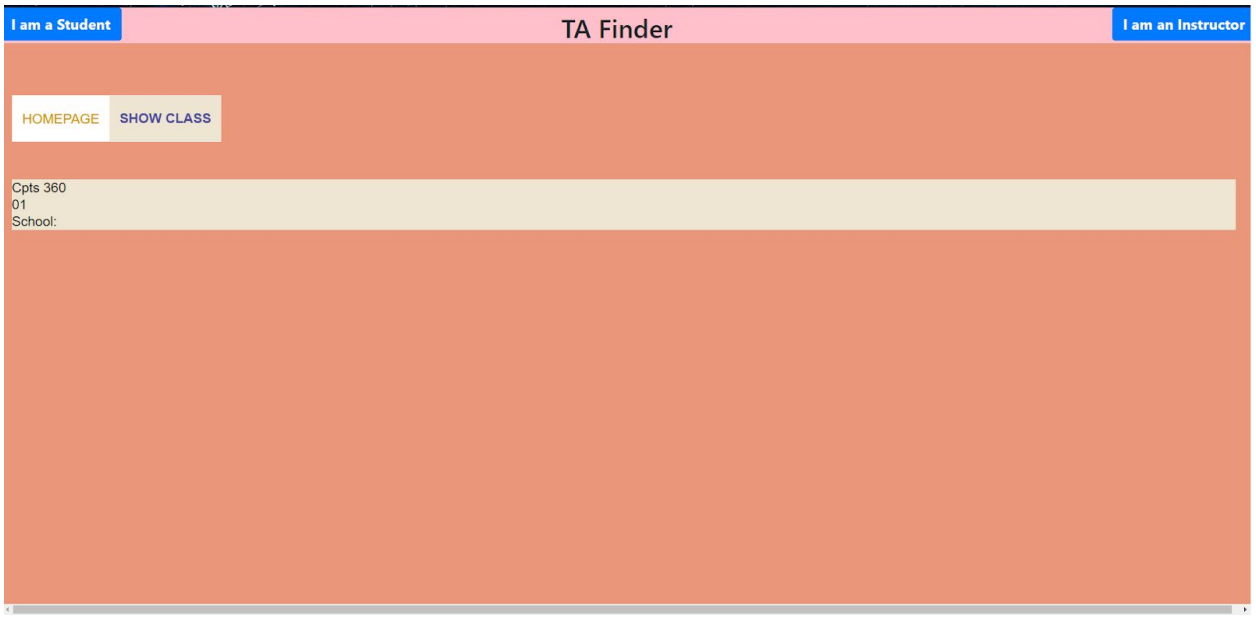
III.3. User Interface Design

Provide a detailed description of user interface you have built so far. The information in this section should be accompanied with proper images of your screenshots. Make sure to mention which use-cases in your “Requirements Specification” document will utilize these interfaces for user interaction.

Use Case #0: Login Page

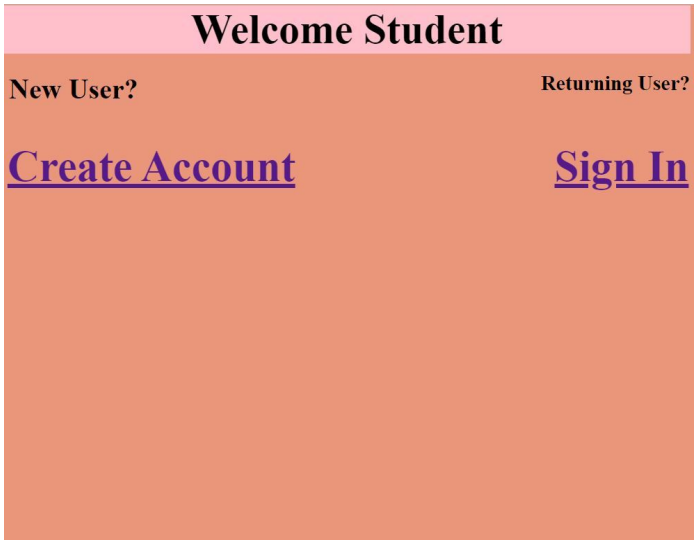
- Displays a “student” and “instructor” button that allows user to select one.
- Header thats displays “Homepage”
- Text that displays “Hello! Are you a …”
- List all the classes in the database
- Show details of a specific class when selected





Use Case #1a: Student Button

- Header Text that displays “Welcome Student”
- 2 buttons thats display “Create Account” and “Sign In”

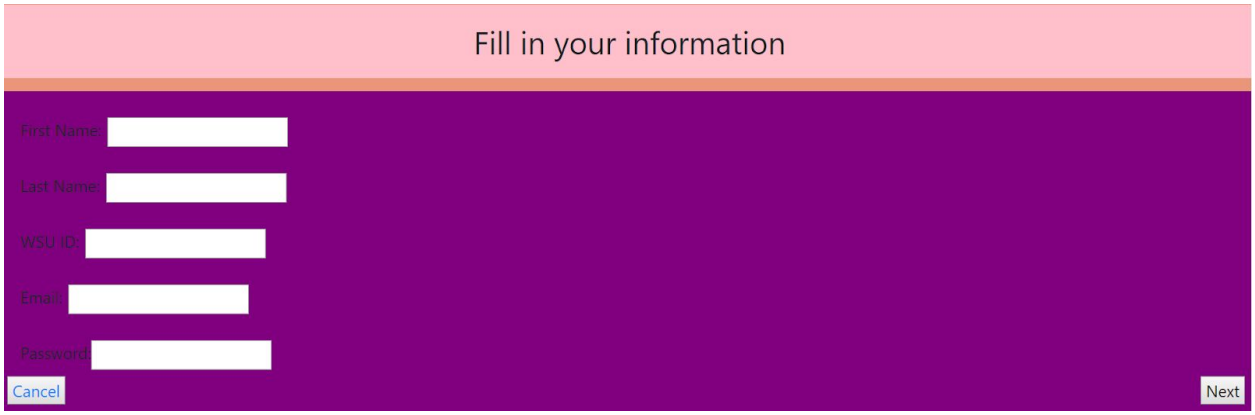


Use Case #1b: Instructor Button

- Header Text that displays “Welcome Instructor”
- 2 buttons thats display “Create Account” and “Sign In”

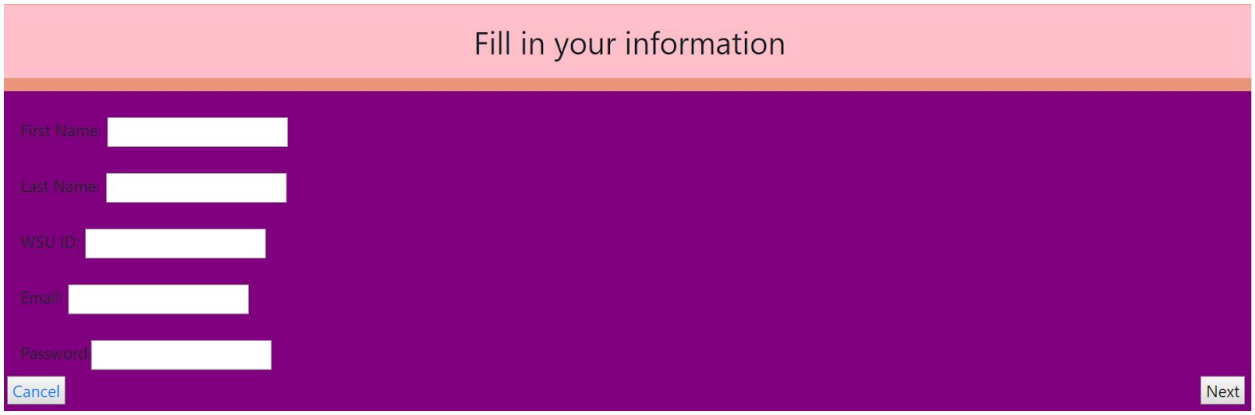
Use Case #2a: Student Create Account Button

- 5 textbox inputs with the labels: “First Name:”, “Last name:”, “WSU ID:”, “Email:”, “Password:”
- “Next” button at bottom of page



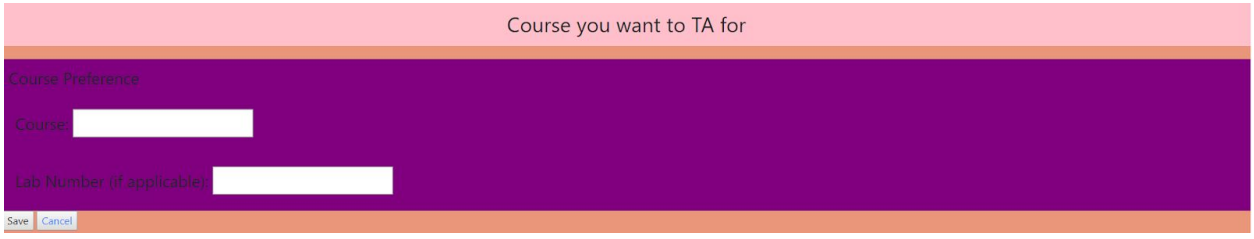
Use Case #2b Instructor Create Account Button

- 5 Textbox inputs with the labels: “First Name:”, “Last name:”, “WSU ID:”, “Email:”, “Password:”
- “Next” button at bottom of page



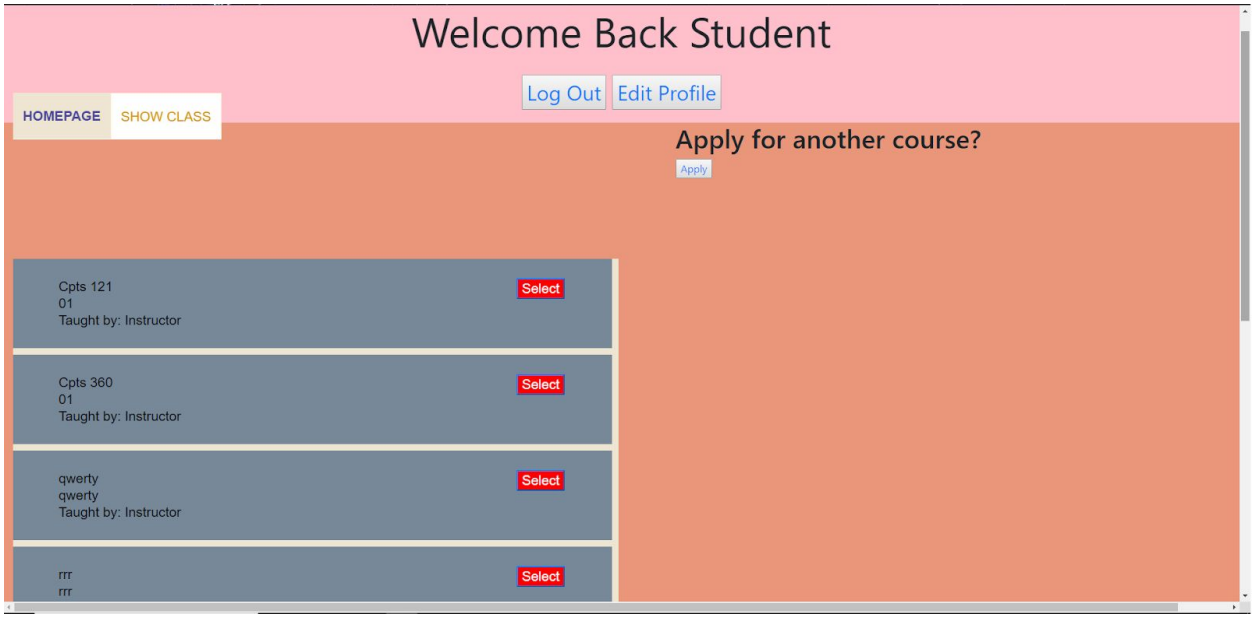
#3a: Student create account next button

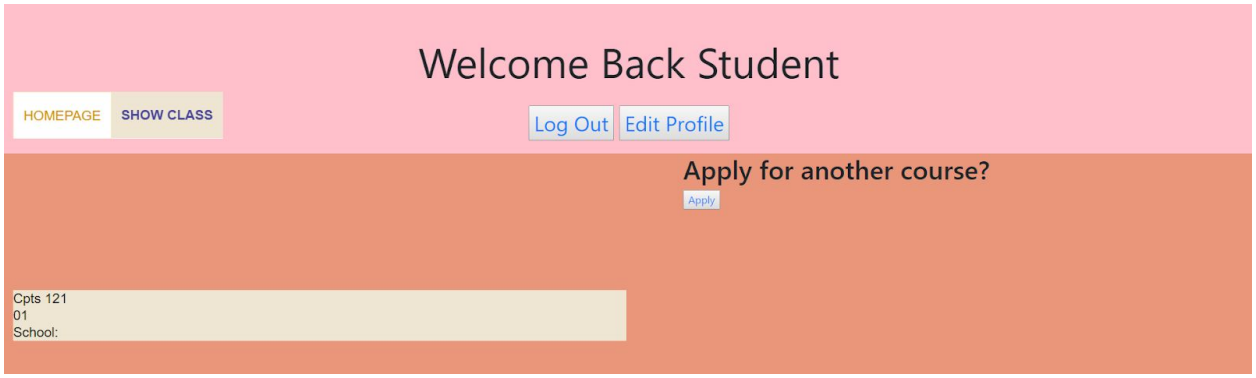
- Header Text “Course Preferences”
- 2 Textbox inputs for the following fields: “Course”, “Lab”



Student Homepage

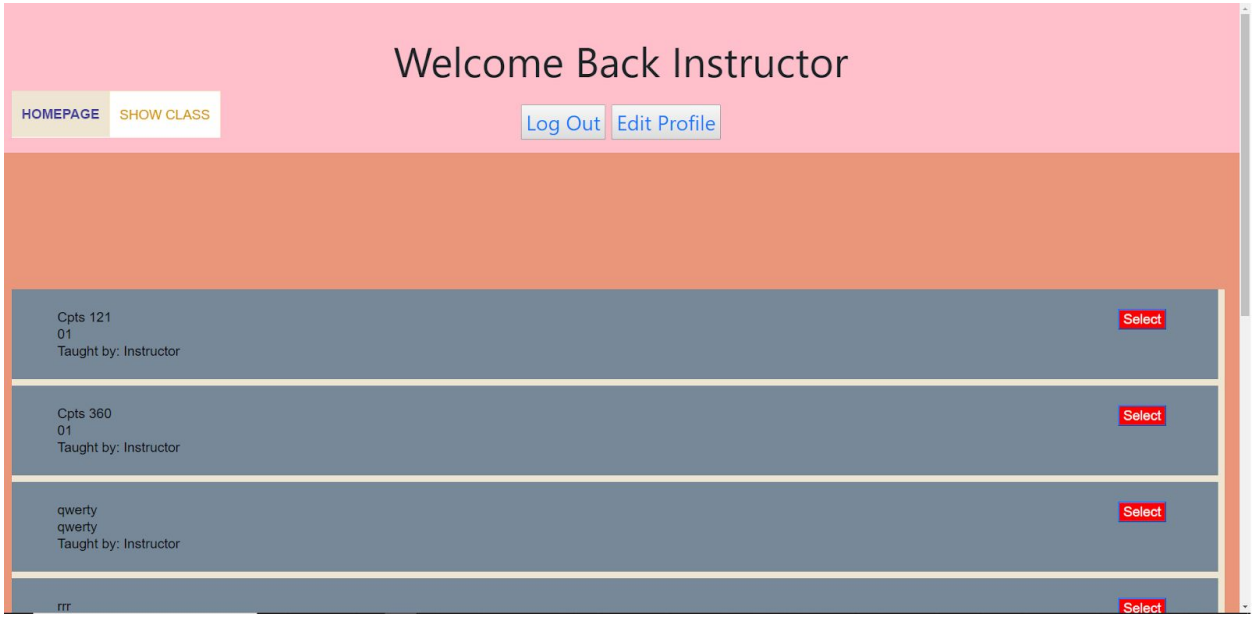
- Homepage shows a list of all the classes that the student applied for
- Has the ability to view information about the class s/he applied for clicking the “Select” button
- Has the ability to apply for another course by clicking the “Apply” button
- Has the ability to log out of the current page back to the homepage by clicking “Log Out” button
- Has the ability to edit profile by clicking the “Edit Profile” button

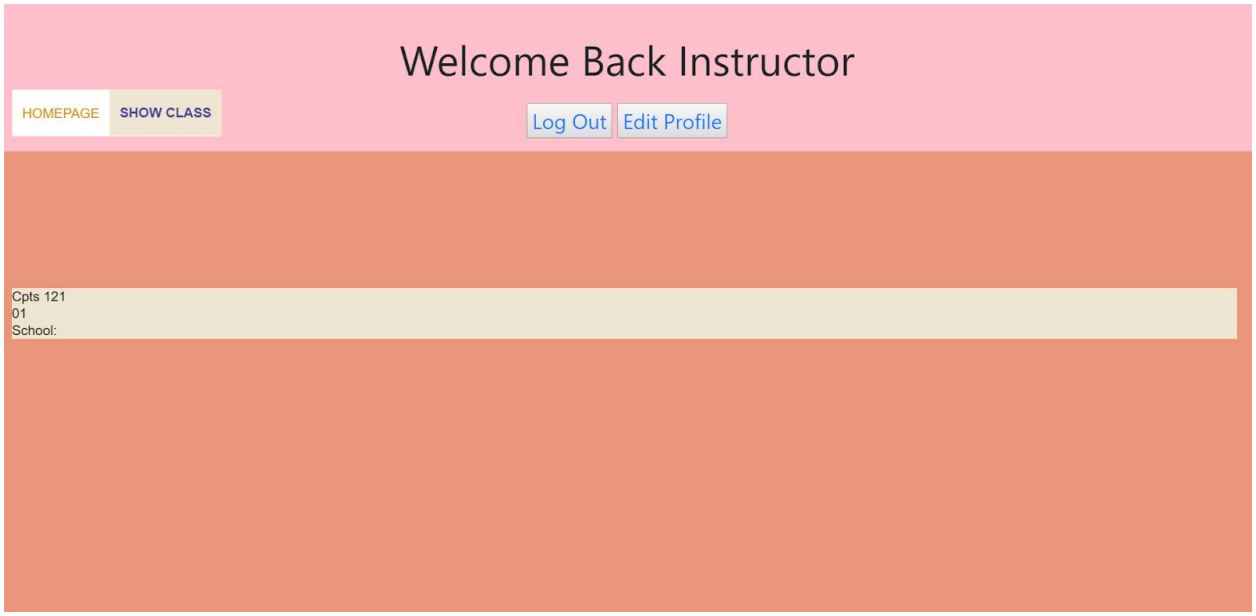




Instructor Homepage:

- Homepage shows a list of all the classes that the instructor teaches
- Has the ability to view information about the class s/he teaches for clicking the “Select” button
- Has the ability to log out of the current page back to the homepage by clicking “Log Out” button
- Has the ability to edit profile by clicking the “Edit Profile” button





IV. Project Progress

In this first iteration, we completed the first draft of this design document. We also completed the student/instructor login or create account pages in HTML/CSS and handled that data with our backend in Python.

For the second iteration, we worked on integrating the backend and frontend better so that the data is now dynamic. We also got our profile's passwords to be encrypted by adding a new login route. The HTML/CSS has been updated to include more features and make it more complete.

Testing:
For testing our code, we used the skeleton code Sakire provided us based on the Smile project. We adapted it to work with our major routes to POST some same data, GET the data, then using unittest compare it to what it should be to make sure it is correct.