TA Helper

Design Document

10/22/2018

Version 0.1

BIG BUG BUSTERS

(Nolen Young, Zach Snoen, Ben Mathews)

TABLE OF CONTENTS

I. Introduction

The purpose of this application is to allow for the course management for class scheduling. This application will allow a user to login to the system and see their information. After login depending if the person that logged in was a professor or a student. The appearance of the system will be different and the actions that each user can do will be different. This project's purpose is to make scheduling easy for TA's to register and choose classes they want to register to TA and for professor's to easily be able to find TA's for the class. This design document outlines the systems that we will create in order to make this web application work.

In section II of this document we will go over the overall architecture of the project. We will discuss the major components of the software and how they all interact and work together. In section III we will discuss more specific design details. We provide more in depth detailing of each subsystem, and provide details on how all major components of the architecture interact. We will also discuss communications protocols between each major component. We will also go over the database tables we will be using in our applications, as well as a mockup of the final user interface design.

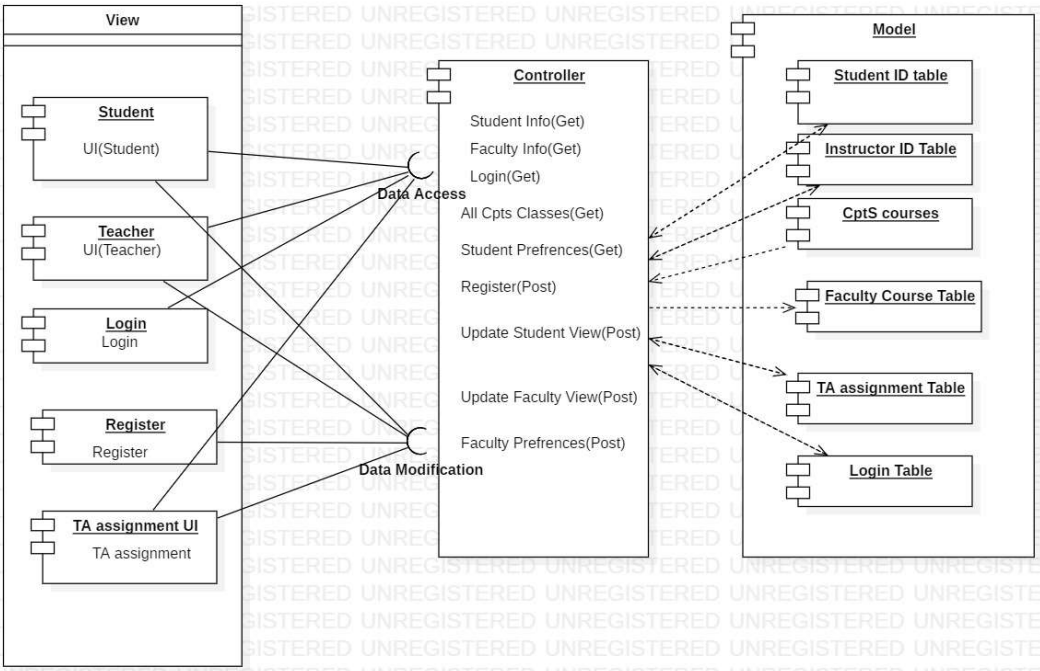
Document Revision History

Rev 0.1 10/22/2018 Initial Version

II. Architecture Design

II.1 Overview

The pattern we used was a MVC architecture. The reason for this choice is that it allows for decreased coupling from the user interface and the data. The way it works is that the user will interact with the system either causing an update request for the controller to the model, or for the controller to retrieve data for the View. If it's modifying or its getting data, either way the user is not able to directly access the database's. The reason for this is that it allows for decreased coupling in the system.



View:

Register(UI): This component is comprised of inputs from the user for their email, and password. Once the registration is valid, a new Faculty or student account is made. The email must be unique in the system.

Login(UI): This component takes in input from the user on their account information. It then checks to make sure that the login information is valid in the database

Student(UI): This is the UI once a student has logged into their account. The user here can edit information and contact info. The specifics will be discussed below. This component makes a request to display the information associated with their account and can edit and change their information.

Faculty(UI): This UI is for the faculty once they have logged in. The faculty can update their contact information, as well as add or delete courses.

TA assignment(UI): This UI is for adding TA's to courses the faculty is teaching.
Controller: The controller is in charge of handling the different requests the user makes, as well as retrieving or modifying the data in the Model. The controller takes the request from the frontend and uses javascript to make requests and have the backend handle those requests.
Model: The model is where the data is stored on the server. It keeps track of all the user accounts, the information for each user, all available cpts courses, the logins of the users, and the courses each faculty are teaching, and the TA's available in each course. Right now the login's are not encrypted.

III. Design Details

III.1 Subsystem Design

III.1.1 View Component

Register(UI): The user can enter in a valid email that must be unique. The user also selects either faculty, or student depending on their status. Once a user hit's submit, that user's email and password make a "POST" request to make a Login Database, and an empty user information database. The user information database will be either a faculty or a student depending on what the user selects.

Login(UI): The user enters their login information and makes a "GET" request to check if the login is valid. If the login is valid then the user or faculty information will be shown.

Student(UI): The user has all the information in the "Student Information Table" displayed on this page. To do this, once logged in the system makes a "GET" request to the controller, which accesses the information from the respective student's information table. The information is then displayed with each piece of data having a box for editing. If the user hits the submit button on the page, then the system will make a "POST" request to the controller which makes then updates the information from the UI to the database. Also on this page is the ability to add and remove course preferences. The user can enter in the data in the TA assignment database table outlined in section III.2. Everything should be filled in to have a valid course preference update. The database is filled with the student's information except for the faculty ID which will be filled addressed later in this section, and the "application status" which will be set to "pending".

Faculty(UI):The faculty information gets displayed with a "GET" request once the login has been made. The information will be displayed and the information will be able to edit. If the faculty hits a submit button the information will be sent to the server and a "POST" request will be made. The Instructor Information table will be modified. The instructor can also fetch all of the cpts courses with a "GET" request to the CPTS database. The instructor can then select a course to add TA's bring up the TA assignment UI. If the instructor selects a course a "GET" request is made to bring up the all of the students that have prefrenced that course.

TA assignment(UI): The instructor can select different students that they want to be assigned to that CptS course. Once students are selected, then a "POST" request is made to change that data to have the

Faculty's respective ID number and to change the application status to "approved". This request also changes the other course preference application status entries that had the selected student's ID to be set to "dismissed".

III.1.2 Controller: The controller is implemented through Javascript code. The UI makes requests either "POST" and "GET" requests to the database that it needs. This allows for the View and Model to act independent of each other.

III.1.3 Model: The different databases and when they are accessed and modified are discussed in detail above. The Model contains the different databases that are needed to store the data in a way that makes it clean, efficient and secure to access.

III.2 Data Design

Student Information Table
First Name
Last Name
WSU ID
Email
Phone Number
Major
Cum GPA
Expected Graduation Date
Student ID

This database table will be created when a student registers their email and password. Their entered email and password will be stored in their respective login table, however their information table will also be created, if that email is unique. The 'student ID' attribute will be created with a value of 1 for student.

Instructor Information Table
Faculty ID
Name
Last Name

Email
Phone
Office

This database table will be created when a faculty registers their email and password and registers as a faculty. The ID will be created with a value of 2, for faculty.

CPTS Courses
Course Name
Title
Description

This database will be initialized and created with the application. This database should never change.

Courses Instructors Teach
Faculty ID
Course Name

This database will be initialized to be empty. Once a faculty user makes an addition to their class list that they teach then an addition will be added to the table with their ID and the course name.

TA assignments
Student ID
Faculty ID
Course Name
Course Grade
Year and Semester taken Course
Application Date
TA before?

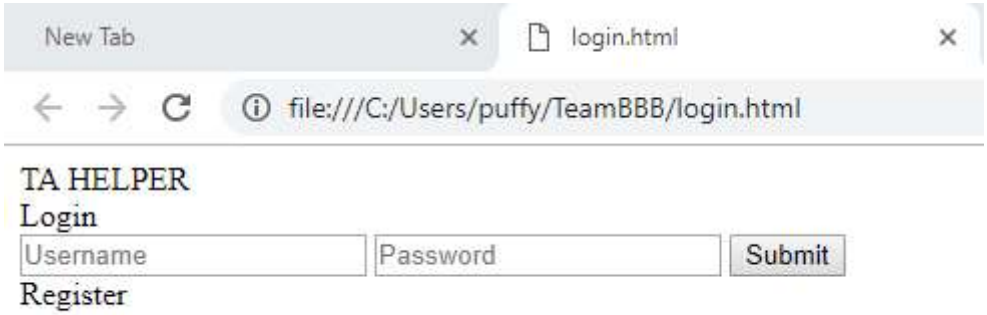
Ta before for this course?
Application Status

The database that will control TA assignments. Students will update in their information UI their courses that they want to add. All that information will get added into this table with an empty faculty id and an application status of ‘pending’. Once a faculty selects a student assignments every other entry in the table with that students ID will have an updated status of ‘dismissed’.

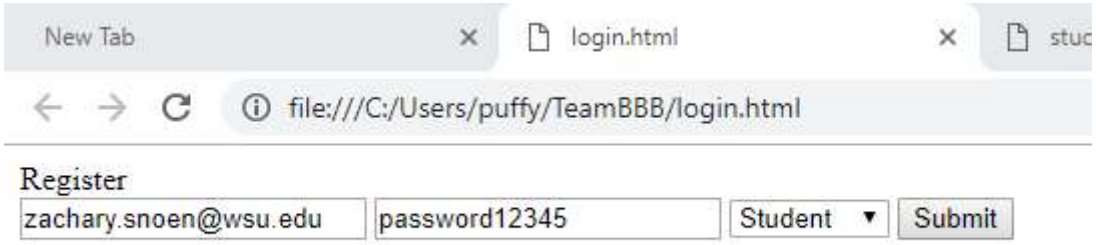
Login Table(not encrypted right now)
Username(email)
Password

The database that keeps track of the emails and passwords of all users. At this time the passwords are not not encrypted. This database is accessed by the controller when a user attempts to login. This database is also used when a “POST” request is made when a new user registers.

III.3 User Interface Design



Example Login Frontend Design.(Above)



Example Registration Frontend Design(Above)

From the requirements document, the use case for registration or “Use Case #2” is implemented. This allows a user to enter email and password, select a student or faculty and enter that into the server. There also is logic to prevent multiple users from registering.

```
puffy@DESKTOP-69TOMFB MINGW64 ~/TeamBBB (iteration1)
$ python server.py
 * Serving Flask app "server" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
C:\Python\Python37\lib\site-packages\flask_sqlalchemy\__init__.py:794: FSADeprecationWa
ault in the future.  Set it to True or False to suppress this warning.
  'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

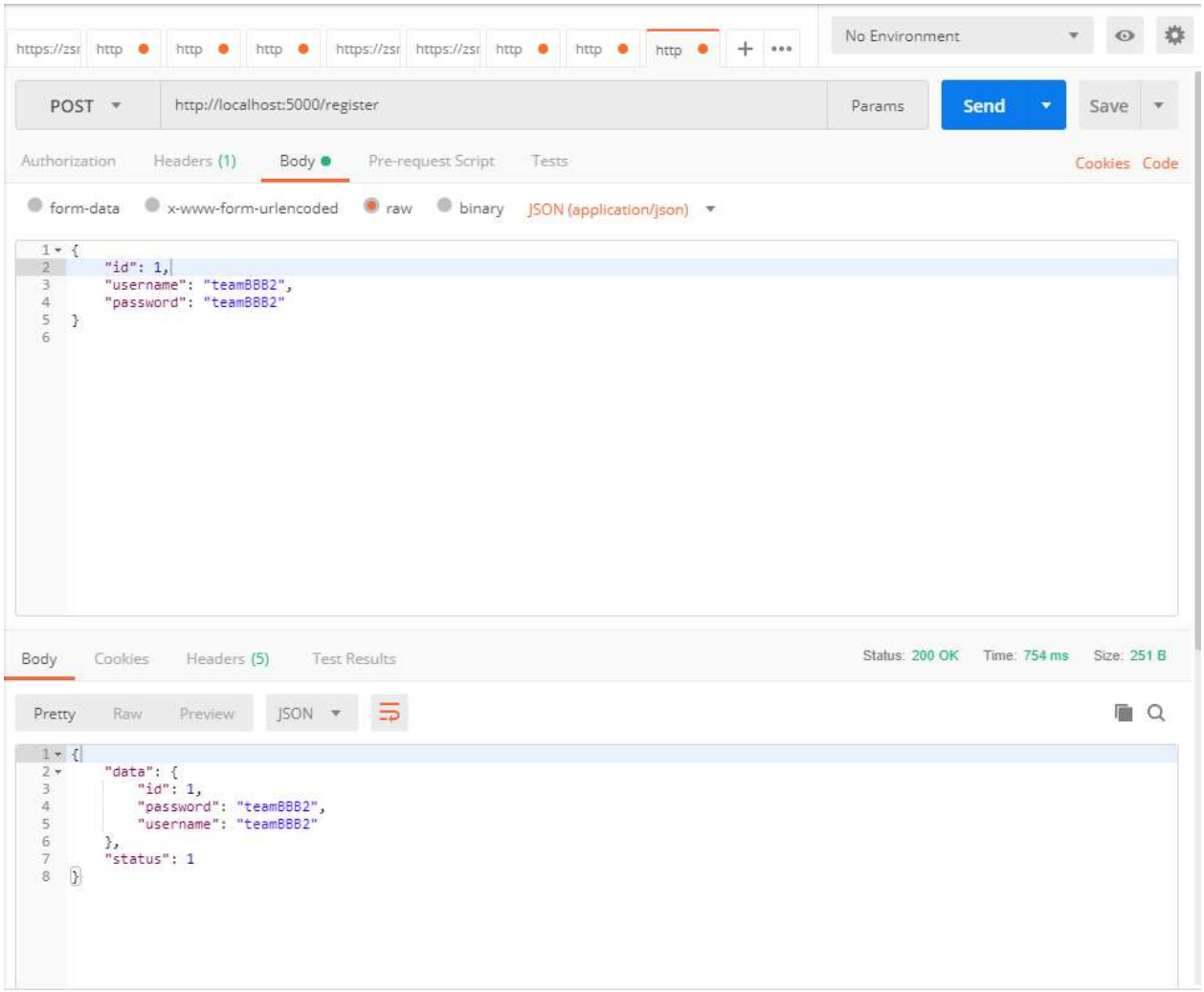
Example LocalHost running (Above)

```
    self._flush(objects)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\orm\session.py", line 2380, in _flush
    transaction.rollback(_capture_exception=True)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\util\langhelpers.py", line 66, in __exit__
    compat.reraise(exc_type, exc_value, exc_tb)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\util\compat.py", line 249, in reraise
    raise value
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\orm\session.py", line 2344, in _flush
    flush_context.execute()
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\orm\unitofwork.py", line 391, in execute
    rec.execute(self)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\orm\unitofwork.py", line 556, in execute
    uow
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\orm\persistence.py", line 181, in save_obj
    mapper, table, insert)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\orm\persistence.py", line 830, in _emit_insert_statements
    execute(statement, multiparams)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\engine\base.py", line 948, in execute
    return meth(self, multiparams, params)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\sql\elements.py", line 269, in _execute_on_connection
    return connection._execute_clauseelement(self, multiparams, params)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\engine\base.py", line 1060, in _execute_clauseelement
    compiled_sql, distilled_params
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\engine\base.py", line 1200, in _execute_context
    context)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\engine\base.py", line 1413, in _handle_dbapi_exception
    exc_info
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\util\compat.py", line 265, in raise_from_cause
    reraise(type(exception), exception, tb=exc_tb, cause=cause)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\util\compat.py", line 248, in reraise
    raise value.with_traceback(tb)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\engine\base.py", line 1193, in _execute_context
    context)
  File "C:\Python\Python37\lib\site-packages\sqlalchemy\engine\default.py", line 509, in do_execute
    cursor.execute(statement, parameters)
sqlalchemy.exc.IntegrityError: (sqlite3.IntegrityError) UNIQUE constraint failed: logindata.username [SQL: 'INSERT INTO logindata (id, username, password) VALUES (?, ?, ?)']
ameters: (None, 'zachary.snoen@wsu.edu', 'password12345')] (Background on this error at: http://sqlalche.me/e/gkpj)
127.0.0.1 - - [22/Oct/2018 23:44:12] "POST /register HTTP/1.1" 500 -
```

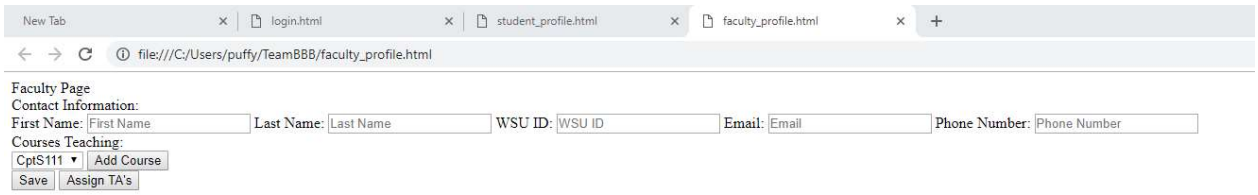
Example error post request where the unique constraint on username fails, not allowing multiple users with the same email to register. (Above)

```
127.0.0.1 - - [22/Oct/2018 23:44:12] "POST /register HTTP/1.1" 500 -
```

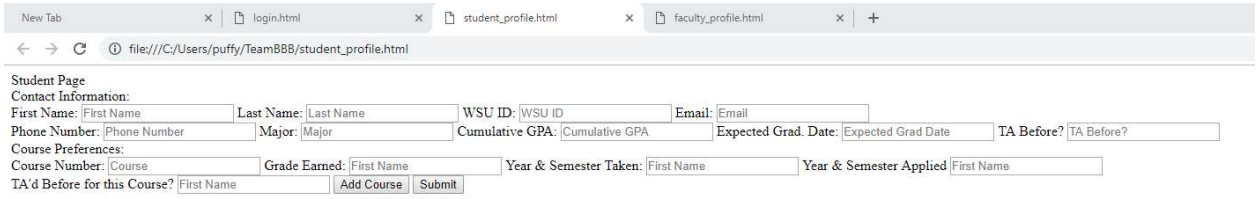
Example successful post request (Above)



Example of sending a post request through Postman (Above)



Example Faculty Information Page Design (Above)



Example Student Information Page Design (Above)