# Teaching Assistant Selection App

## Design Document

10/22/2018

Version <0.1>


**git push master --force**

Garrett Rudisill

Slater Weinstock

Luke Brossman

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

# TABLE OF CONTENTS

# I. Introduction

This document explains the design and philosophy of the teaching assistant finder application.

This project aims to reduce a significant portion of the friction involved in the process of finding and filling teaching assistant positions. The main source of friction alleviated by our project will be the reliance on face to face meetings and word of mouth to spread information about available TA positions and potential TA's. Our app will condense this information into a single location where both students and professors can access and update it in real time.

Section II includes a general overview of the system, including a UML diagram of the system. Section II also contains explanations of how components work and interact in a more general sense.

Section III includes an in depth description of all the subsystems present in the application, including the frontend and design, backend and design, and how the components interact in a specific technical manner This also includes a section describing the user interface design in greater detail with images of the current prototype of the developing user interface.

Section IV will explain in detail what our testing protocol is, what tests we perform, and how these accurately verify that our application works in a given scenario. This includes both fully-automated, semi-automated, and manually tested procedures.

## Document Revision History

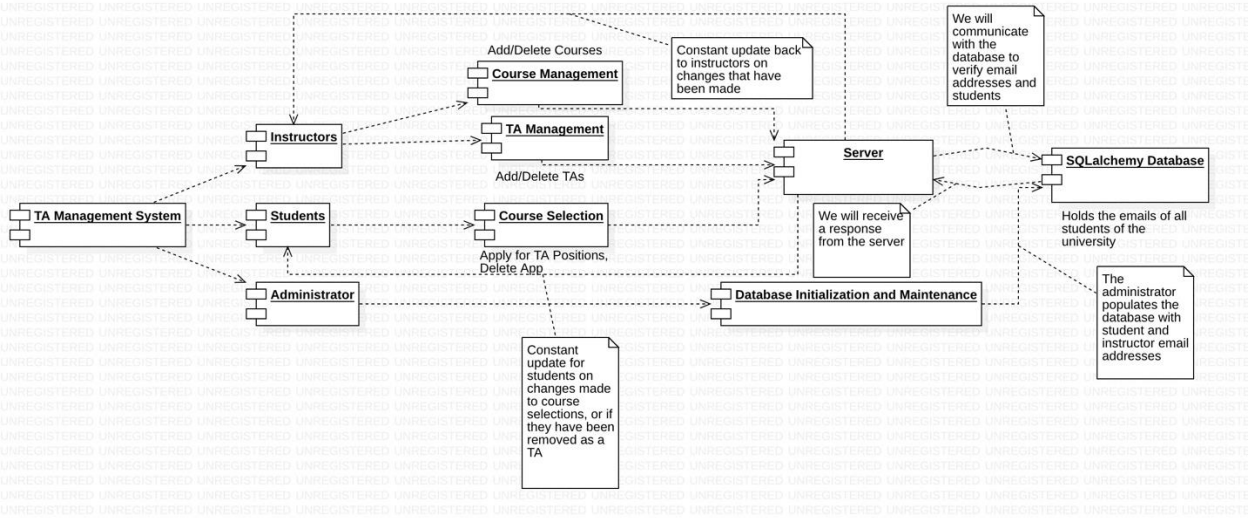
Rev 0.1 October 19 Initial document

# II. Architecture Design

### II.1. Overview

Our system follows a client-server model and consists of a web page front end and a database backend with remote access provided from a server. Our frontend houses the responsibility of containing the complete user interface for the application, and all user actions will be performed through this interface. A user's interactions with the UI will be interpreted by a JavaScript logic engine to generate requests to the server. The backend will receive these JQUERY generated requests and return information from the proper database table(s). This backend will be deployed to a server host, Heroku, which provides the hardware and domain name for the backend. The information received from the server will be interpreted as a JSON object. All network traffic and stability of said traffic is dependent on the server hosts reliability.

UML Component Diagram




As can be seen from the UML diagram above, the TA management system is going to have 3 types of users, and there will be updates to/from the server which include course deletion, TA deletion, and TA assignments as well as verification of students and instructors. This follows the client-server model in which the front facing UI of the webpage for individual users is the client, and the server is our database which will return queries.

# III. Design Details

## III.1. Subsystem Design

### III.1.1.[Frontend User Interface]

The frontend is written using HTML, CSS + Bootstrap, and Javascript. We use forms to allow the user to fill in the desired inputs sufficiently. CSS + Bootstrap allow us to maintain a consistent reactive styling between pages, regardless of content style present within the page. Scripting for the front end is done using Javascript and jQuery. This allows us to grab the values from the input fields, group them, and process them for transmission and storage in the database via our API (work in progress). Data is transmitted when a user clicks any submit button, and initiates a function to grab the data from the forms, parse and package into JSON format, then send a HTTP POST request to the backend through a jQuery AJAX statement. There will be checks performed within the frontend scripting which ensure that each form is properly filled out to prevent any unforeseen backend glitches.

### III.1.2.[Backend Server with Database]

The backend is a database with API implemented with Python and Flask. The database has tables for each respective data type, such as instructors, students, classes, login information, applications, and any additional databases as necessary with design progression. Through the backend there are established GET, POST, and DELETE routes through which the frontend javascript can both send, receive, and modify entries within the database. Given that this database is networked, this makes our application dependent on a server host for the deployment of the server. Our development team will be using heroku, although alternatives exist and may be considered if necessary due to stability or any

other unforeseen issue. Through the server host, we are given a individual URL for our API/database host, which is key to our routes and their traffic.

### III.2. Data design

The data design of the backend treats each category of data as fundamentally separate. This way each table only contains necessary elements to the respective route connected to the table. This also helps prevent overwrites of data, duplicated data, and overall bloat. If a function for the front end needs multiple types of data, it can make multiple requests, and parse from the front end to represent the data in the desired skew. The database managed by the server contains various tables; one for each significant part of the application. This includes a table for instructors, students, classes, and applications, and one for managing secure logins. Each table is described in depth hereafter. Student table contains an ID number, first and last name, email, major, gpa, and graduation date. Aside from the ID number stored as an integer, all other fields are stored as strings. The ID number is the primary key and is a unique identifier. The instructor table contains an ID, first and last name, email, phone number, and office name/number. All fields aside from integer ID are strings. The class table is much simpler, and consists of all strings for the name, title, and description of the class. The name is the unique identifier of the table.

### III.3. User Interface Design

Currently, user interfaces are built for the login page, general account creation, and class creation. Each of their current prototypes are displayed on the following pages, subsequent to their descriptions.

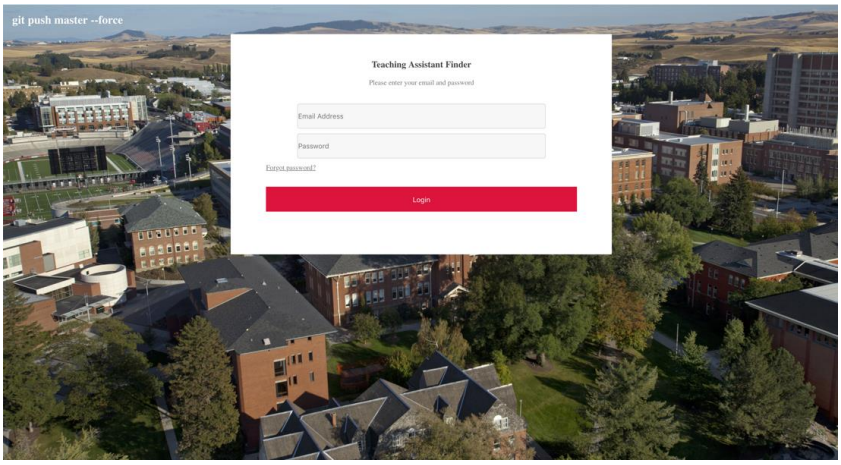
The login page is the initial landing page for the web application. Upon access, a user will find a prompting form for username (email) and their password. If the user does not have an account there will be a link to an account creation page. If the user enters their credentials and they do not exist or are incorrect, the user will be prompted by the page to check their credentials or create an account if the username isn't found.

The account page currently exists as a general template. It contains the fields which are common to both the instructor and student accounts, along with a radio button to signify which type of account is being created (instructor or student). Upon the user clicking submit at the end of the page, the front end initiates a data collector, parser, and POST request to the backend to finish the submission process. If the user leaves a field blank, the submission page will prompt the user to correct the error present. In future iterations, the button signifying which type of account is being account is created will be responsive in the page. This will hide or show fields which are necessary to their specific backend routes and submissions.

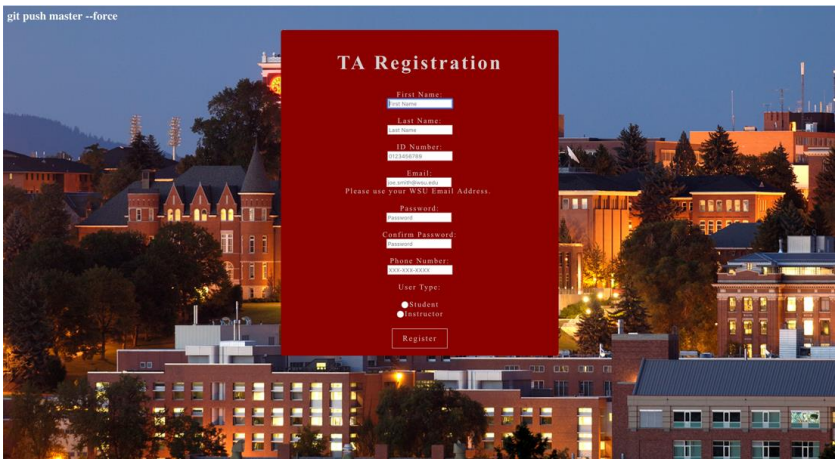
The class page is very rudimentary. It has few fields and is only accessible by administrator and instructor accounts. The page prompts the user for a class name, title, and description. The page maintains a consistent styling to the account page.

A user's profile page is centered around the options contained on a top justified menu bar. We currently intend for the same menu bar html to be used for both student and instructor profiles, with different links accessible via the menu buttons. These links will be to pages containing lists of classes that can be sorted in different ways depending on the user's request. Our plan at the moment is to have classes sorted by those that need TA's, those a student has applied to TA, those a student has been accepted to TA, those that an instructor is instructing, and "all" (no filtering).

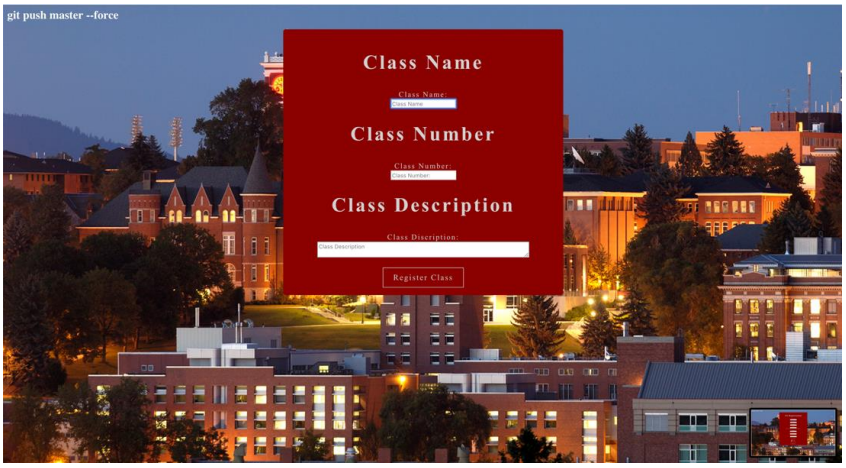
Login




Profile Registration



Class creation page




Profile page template


## IV. Testing Plan

**(in iteration 2, but started now)**

This program will be tested frequently and consistently throughout the development of the program to ensure new functionality meets specification and personal expectation and to also ensure that regression does not occur. Tests will be automated where possible, primarily in the frontend to backend interaction pathing and networking. Functional and UI testing will happen manually.

- **Unit Testing:** Automated tests are written for each javascript function that initiates a request to the backend. A debugging page will be present in the project, which on successfully loading executes all the automated tests. The tests can be verified via multiple means. Console logging and alert windows are coded into the javascript portion

to alert the tester to what is occurring in the test suite. GET requests can then make requests to the server and verify those entries exist and that potential errors were successfully handled (this is yet to be implemented). The data can also be verified using the SQLite plugin in Visual Studio Code, which allows the reading/visualization of the database.

- **Functional Testing:** Functional testing will be performed manually and use similar methodology as unit testing for result verification.

- **UI Testing**: UI testing will follow the same protocol as functional testing, due to their interrelated nature.

## V. References

Application Repository - https://gitlab.eecs.wsu.edu/322-fall2018-termproject/TeamgitPush