

WSU TA Management

Design Document

11/12/2019

Version 2.0



Team Kappa

Joshua Green, Zach Penn, Mykhailo Bykhovtsev, Rodney Walton

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I.	INTRODUCTION	2
II.	ARCHITECTURE DESIGN	2
II.1.	OVERVIEW	2,3
III.	DESIGN DETAILS	3
III.1.	SUBSYSTEM DESIGN	3-5
III.2.	DATA DESIGN	6
III.3.	USER INTERFACE DESIGN	7-10
IV.	PROGRESS REPORT	10
V.	TESTING PLAN	11

I. Introduction

The purpose of providing this design document is to track the progress, functionality, and details regarding the project (WSU TA Management). The information presented should provide a thorough understanding of how the system is being designed and what needs to be done. Even with no prior knowledge, someone new should be able to read through and continue the design of the system by themselves. This document will constantly be checked back on and revised, following and guiding us to the very end of the project. Main revisions will be done for each of the three iterations, and will include documentation of any changes.

The project presented in this document is a tool to enable teachers to easily find willing assistants for their classes. The final version will allow students to maintain a profile containing their contact information and course interests, and instructors to select a teaching assistant from those students.

Section II of the design document, Architecture Design, describes the software's architecture and components in great depth. This includes how the different pieces of the system interact, architectural patterns, and diagrams to help explain the design.

Section III of the design document, Design Details, goes deeper into the system. Looking at the subsystems, this section will explain how each of them will work and communicate, as well as provide information on the organization and inner workings of the software.

In Iteration 2, we created the interfaces for edit profile (student/instructor), list courses, and select instructed courses (instructor). For each of these pages, we implemented the backend so the user can interact with the pages and have this data stored. On top of all this, user login has been fully implemented as well, including encrypted credentials and verify login. The project as a whole has been successfully set up on Heroku as well.

Document Revision History

Rev.1 10/23/2019 Initial Version

Rev.2 11/12/2019 Iteration 2

II. Architecture Design

II.1. Overview

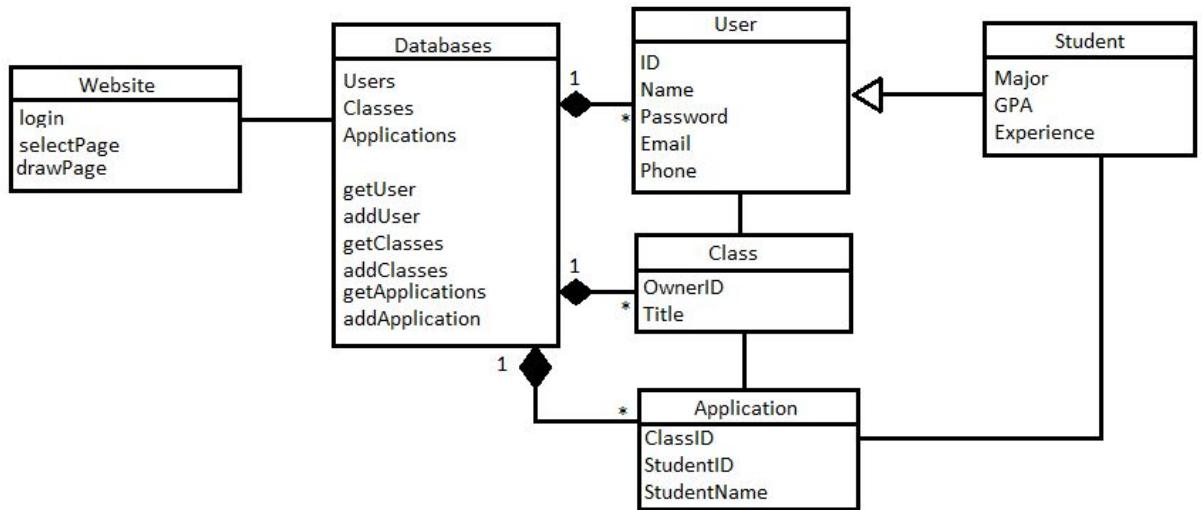
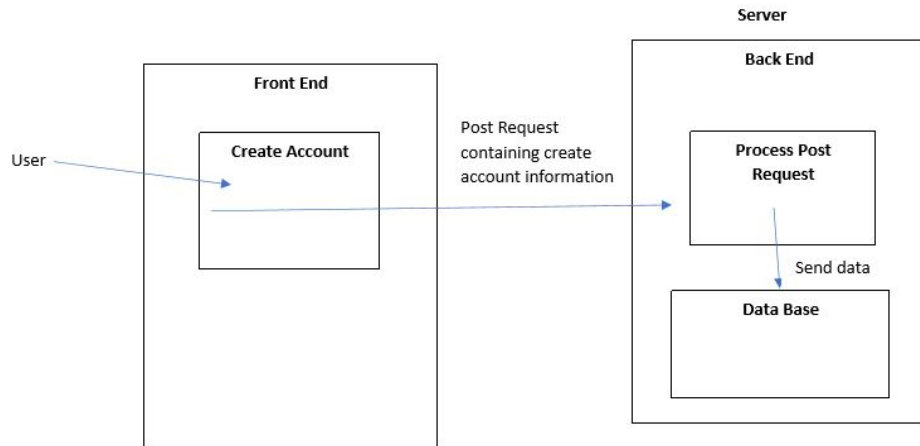
- The architecture pattern that we decided to go with is the Three-Tier Architectural Pattern because it is best suited for Websites and in our case. The user interface level is using html and css. The logic level is using flask and python to parse the information and send it back and forth between the database and the user interface. The data level is using SQLite to store information. For example, we are using Python to get the list of classes from our Class table, then we use flask to load the new html page 'display_courses.html' and send the list of classes to the html. Now with the information returned from the backend, we can use 'data' to load the information in the page. We again do this with Python and Flask to iterate through the data and load onto the page.

- The subsystems currently have high coupling. The high coupling stems from the way we send information back and forth from the front end to the back end. Everytime we do a 'GET' request we must then go back to the front end and code how to handle the request to every separate file in the front end. So potentially each html file is receiving or sending information independently of one another. Which also means our cohesion is low since a lot of our html files of their own unique functions on handling the data back from the backend.
- Interface component is responsible to provide an interface with user. The application logic component is currently our backend python code and flask which processes the input and prepares and transmits data to the Storage(backend) component. Application Logic also includes Page generation by generating full page by grabbing specific page and adding header, footer and menu. The storage component of our architecture is our backend that handles queries to the database. Currently it can receive information sent by application logic component and put it properly for storage into the database. It also includes password encryption. The encryption we are using comes from flask extension that hashes passwords, it is an adaptive hash function based on Blowfish.
-
-

III. Design Details

III.1. Subsystem Design

- Our current design of interface component is an HTML page for Create User Account which we try and keep generic for both Student and instructor by having them input common information such as first name, last name, WSU ID, email, phone number, password. Then we have an option to choose the account type. There is a submit button and a cancel button. By identifying the common input data that both users have we were able to figure out that we can have a single page for creating an account. The application logic system uses JQuery to find forms and grab input from them. Data are being prepared and sent to the right route for storage subsystem to receive. It is designed to check account type and based on it send it to the right route. Our backend is technically both application logic system and storage. Because the process of post requests is handled on the backend and addition of data into the database is handled there too. It is designed to use a special routes that are based on the account type. After it receives data it processes it and puts it into the database.
- The database contains 3 tables: Users, Applications, Classes. Users is a table that contains user information, which depends on user type: Student, Instructor. It is generic table where Instructor has less data than Student typically. Applications and Classes are to be worked in future. The data that we send from Frontend(Application logic system) are in JSON format which are the inputted user data. The data that we receive from Backend are also in JSON format which contains same inputted user data and status if we succeeded or not.



Routes

```
@app.route('/newaccount', methods=['POST'])
    (conditional) return redirect(url_for('render_signup'))
    return redirect(url_for('render_profile'))
```

Route to create a new user, and redirect user to the profile page with information.

```
@app.route('/create-course', methods=['POST'])
    (conditional) return redirect(url_for('render_create_course'))
    (conditional) return redirect(url_for('render_create_course'))
    return redirect(url_for('render_display_courses'))
```

Route to create/add a course, and redirect user to updated display courses page.

```
@app.route('/edit-account-student', methods=['POST'])
    return redirect(url_for('render_profile'))
```

Route to edit student profile information, and redirect user to updated profile.

```
@app.route('/login', methods=['POST'])
    (conditional) return redirect(url_for('render_profile'))
    return redirect(url_for('render_login'))
```

Route for user to login, if successfull redirects to their profile page.

```
@app.route('/viewprofile')
    return render_template('viewprofile.html', **data)
```

Route to view profile with given data. Login is required.

```
@app.route('/create_course')
    return render_template('create_course.html')
```

Route to render new page. Login is required.

```
@app.route('/edit_profile_student')
    return render_template('editprofile_student.html')
```

Route to render edit profile for student page. Login is required.

```
@app.route('/edit_profile_instructor')
    return render_template('editprofile_instructor.html')
```

Route to render edit profile to instructor page. Login is required.

```
@app.route('/login')
    return render_template('login.html')
```

Route to render login page.

```
@app.route('/signup')
    return render_template('create_account.html', **data)
```

Route to render signup/create an account page with data.

```
@app.route('/display_courses', methods=['GET'])
    return render_template('display_courses.html', data = names.all())
```

Route to render display courses page with data (course names).

```
@login_manager.user_loader
    return User.query.get(int(ident))
```

Route to get user id.

```
@app.route('/logout')
    return redirect(url_for('render_login'))
```

Route to logout of account/render login page. Login is required.

I.1. Data design

We currently have 3 tables named User, Classes, and Applications. The Users table contains all users on the site, students and teachers. The Classes table contains all the classes entered. The Applications table contains all of the applications submitted by students.

Applications Attributes

- Class id
- Student ID
- Student Name
- Student Last Name

Classes Attributes

- Teacher ID
- Class Title

User Attributes

- User ID
- Student Name
- Student Last Name
- Teach or student boolean
- Password
- Email
- Phone
- Major
- GPA
- Experience

I.2. User Interface Design

So far we've built drafts of a few of the use cases, along with a mock header and footer. First we've built the create a profile feature from the first use case:

The mockup shows a web page titled "TA Management System" with a Washington State University logo in the top left. In the top right, there is a "Sign Up" button and a link "Already have an account? Login". The main content area features a red-bordered box titled "Account Information" containing the following fields: "First Name:", "Last Name:", "WSU ID:", "WSU Email:", "Phone #:", "Password:", and a dropdown menu for "Specify Account Types" with "Student" selected. Below these fields are two buttons: "Create Account" (blue) and "Cancel" (grey). The footer includes a "Contact Us" link on the left and "© Washington State University" on the right.

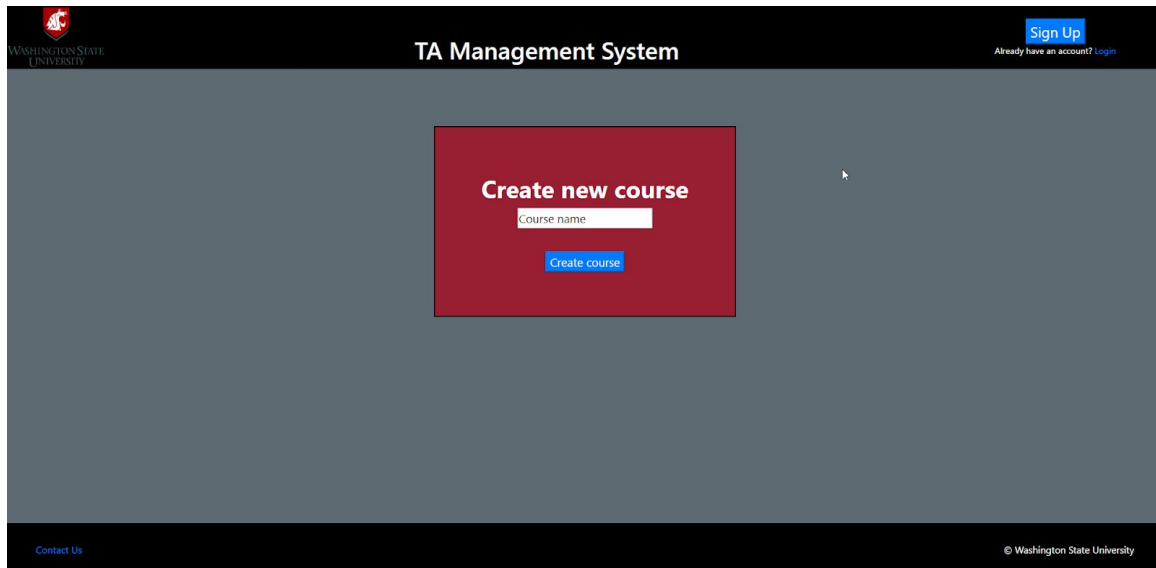
It allows the user to input relevant information regarding account creation and asking if they're a TA or a student. You can also see the start of our footer and header, which is currently more or less a place holder but will later grow to fit other links and will have better styling.

Next we've built the login feature from use case 2:

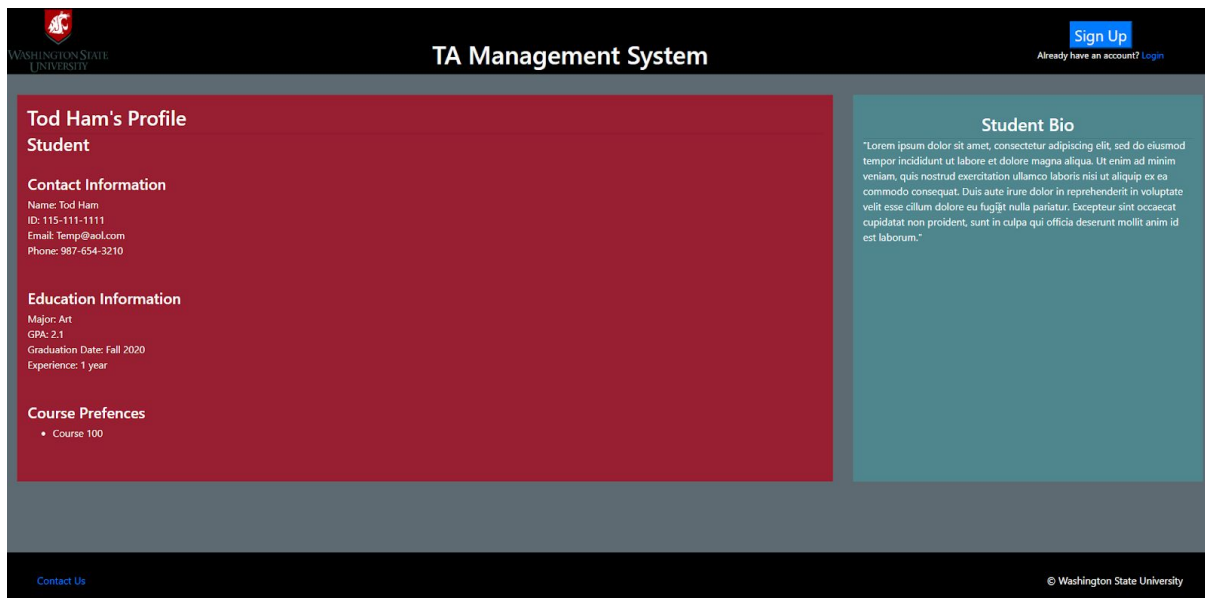
The mockup shows a web page titled "TA Management System" with a Washington State University logo in the top left. In the top right, there is a "Sign Up" button and a link "Already have an account? Login". The main content area features a red-bordered box titled "Login" containing the following fields: "Username" and "Password". Below these fields is a blue "Submit" button. At the bottom of the box, there is a link "Don't have an account? Sign Up". The footer includes a "Contact Us" link on the left and "© Washington State University" on the right.

Takes in username and password and checks if valid, then logs them in. Also gives the user the option to click the sign up link on the bottom of the login box, if the don't already have an account.


We've also created create new course feature from use case 6:



Allows the user(only teacher in this case) be able to add courses available that students can later apply as TA for. Next, we built the display profile feature from use case 3:



Displays users name, contact information, education information, and course preferences. Along with whether they're are TA and a bio. Here, they have the ability to click edit profile, which takes them to the following page (given that it's their profile).


[Home](#)
[Profile](#)
[Logout](#)
[Display Courses](#)

TA Management System

Tod Ham's Profile

Instructor

Contact Information

Name:

Phone:

Current Courses

Add Course:


- Course 100
- Course 101
- Course 102

Instructor Bio

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

[Contact Us](#)
© Washington State University

Here, the user has the ability to edit their profile information, and save the changes. Edit profile is implemented for both student and instructor. Edit profile for student is shown above, and edit profile for instructor is shown below.


[Home](#)
[Profile](#)
[Logout](#)
[Display Courses](#)

TA Management System

Tod Ham's Profile

Student

Contact Information

Name:

Phone:

Education Information

Major:

GPA:

Grad. Date:

Experience:

Course Preferences

Add Course:

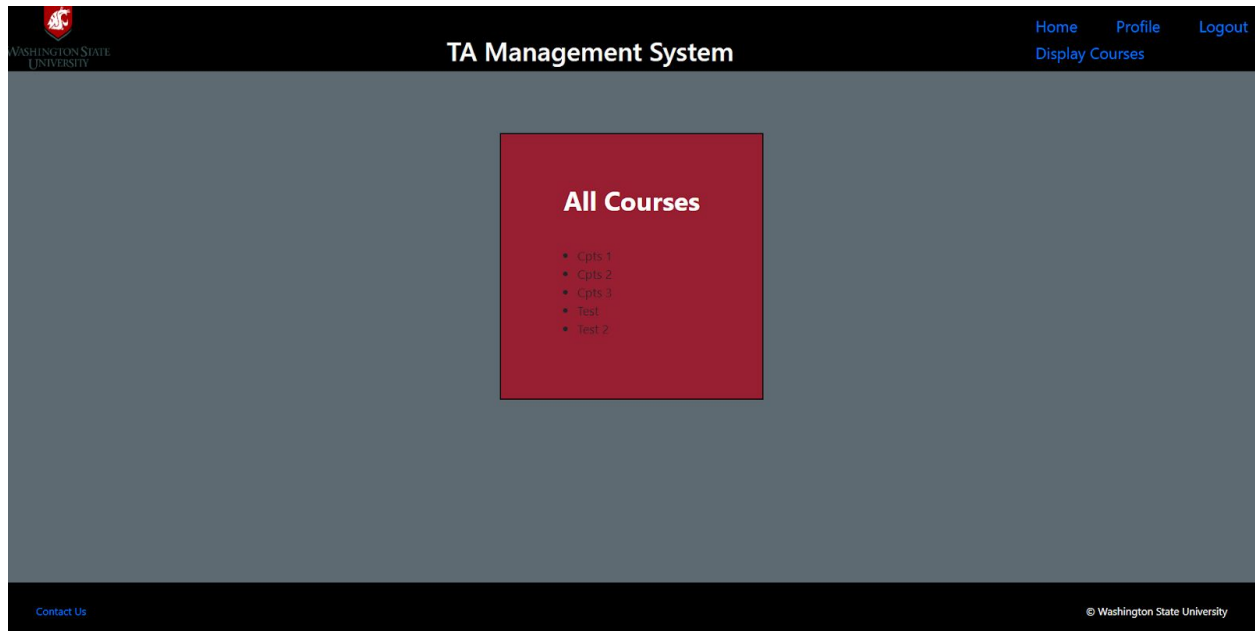
- Course 100
- Course 101

Student Bio

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

[Contact Us](#)
© Washington State University

Finally, we have a display courses page which shows the updated list of courses (connected to the backend).



IV. Progress Report

The main difficulties for iteration 2 include many technical/organizational issues, some of which go hand in hand. One problem we had, and still have, is the implementation of our header/footer. For these, we're using external html/css files and using the include command to add them to each page. Even after hours upon hours of trying to fix the way the header/footer squishes the page (and the footer not staying at the bottom), the problem persists. There has also been some difficulty with connecting the front end to the back end, regarding the display of information.

The features we completed in iteration 2 include edit profile (student), edit profile (instructor), and choose/list all courses. For the most part, these are fully functional. However, there are small issues and visual problems we're working towards fixing for the final iteration (For example, the placeholders for the text entry boxes in edit profile). Overall, everything went as planned and we finished what we wanted to do for the iteration, without having to push a feature to a later iteration.

Aside from the manual tests we did for everything by checking the server we were running and the information within it (confirming that features worked), we also began preparing unit tests for api routes. This is further explained under the Testing Plan section of this design document. With the use of SQLite we were able to see and confirm the information in the backend is as it is supposed to be. While we aren't explicitly testing everything at the moment, we plan to do so before the final iteration is done, by completing the code early and perfecting it based on these tests.

V. Testing Plan

Unit Testing: For api routes testing we plan use a framework called 'SuperTest'. Which is designed for testing apis. We first write some information that we expect to be sent/returned by the apis. We then send that information through the api routes. Then using SuperTest we check if the information returned/sent is correct, if it is then it will return ok, else it will return fail.

Functional Testing: We will test it by verifying the information that it should be displaying or storing is correct. For example, with 'create user' we'll fill out the fields and verify in the database table that the information entered is also now the same in the database. Then recheck by viewing profile to see if the information there is correctly matches the inputted information. Similar tests will be carried out for all instances where information is transmitted from backend end to front end, and vica versa.

UI Testing: We will manually check to make sure UI is correctly displaying. Checking how the UI looks like on different screen sizes and checking what happens if the UI has too many or too few inputs and see how it handles the output.