Research Position Search App

Design Document

10/27/2021

Version 1

CTRL_Z

Jared Lustig, Musa Husseini,Michael Stawowski

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I. Introduction

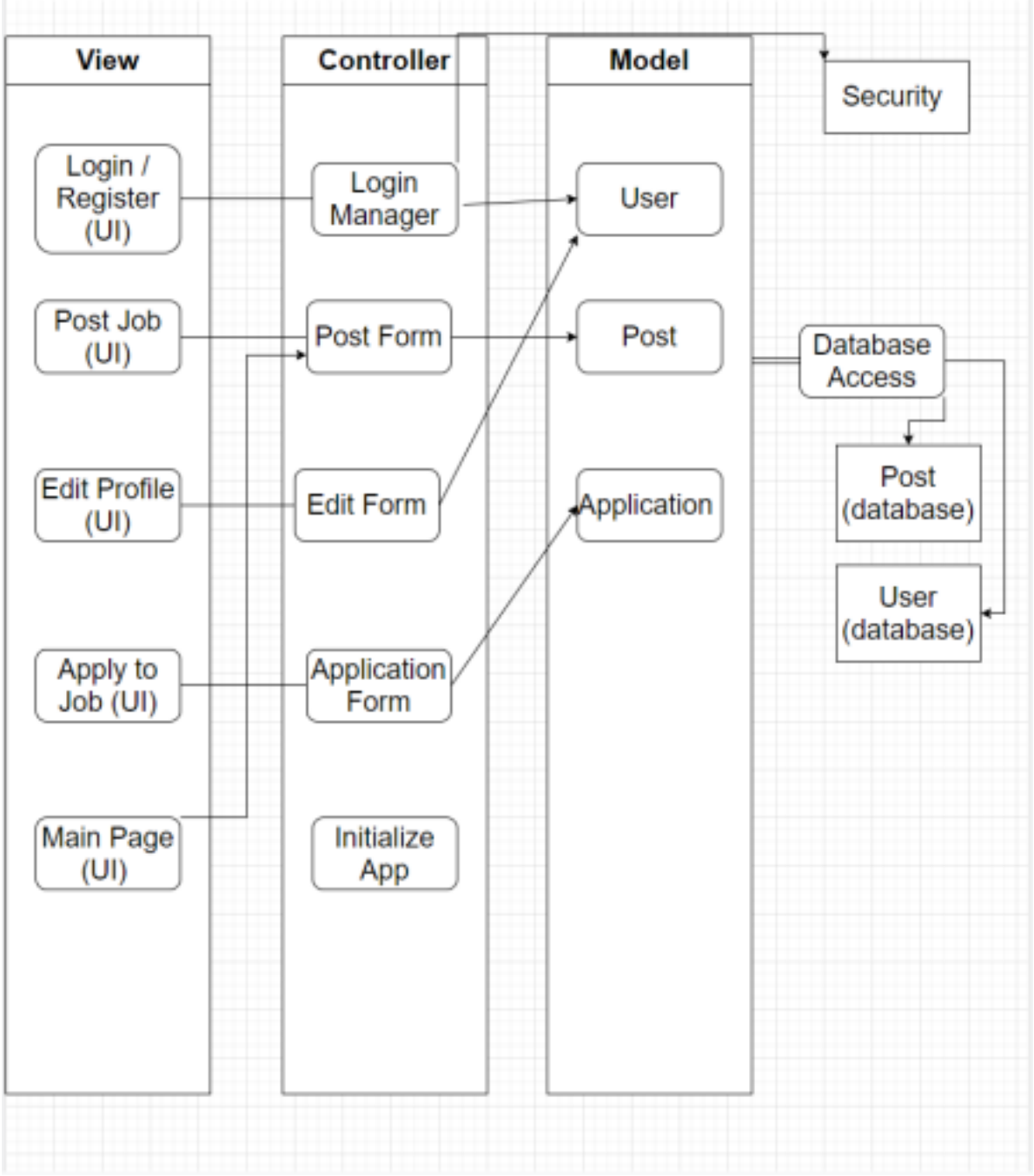
The purpose of the document is to provide a high level explanation of the architectural design and current trajectory of our SaaS application. The application will give faculty a service where they can advertise their open research positions and allow qualified undergraduate students to find them. Furthermore the end goal of this project is to provide a seamless user experience for both the student and faculty. The document will be broken into sections with section 2 including the architectural design of our software, section 3 including a progress report for each iteration of the project before the final release and section 4 laying out a detailed plan on how we're going to test our software to ensure stability.

Document Revision History

Rev x <10/26/21> <comment> (Ex: Rev 1.0 2021-10-27 Initial version)

II. Architectural and Component-level Design

II.1. System Structure



In our view, we have five subsystems. Our login/register represents the html and css files that will be the user interface. This will be how we present our product to the user. Our job post subsystem is to represent what the page where a faculty member will make a post. The edit profile interface represents what the application page will be when a student applies to a job. The main page interface will be our index/homepage, this will show all job postings. The controller itself has a few subsystems within it. The subsystems in the View system will look into the controler to find what to render onto the page. The login manager will render the login/register interface, the post form will be used to display posts. The edit form will be used to

render the display form interface. The application form will be used to render the application to the job interface. There is also an initialize app subsystem that will get the app running with baseline forms and routes. The model system contains a few subsystems. These subsystems are essential for the controller. The controller will look at the model to find the correct database table to base itself off of.

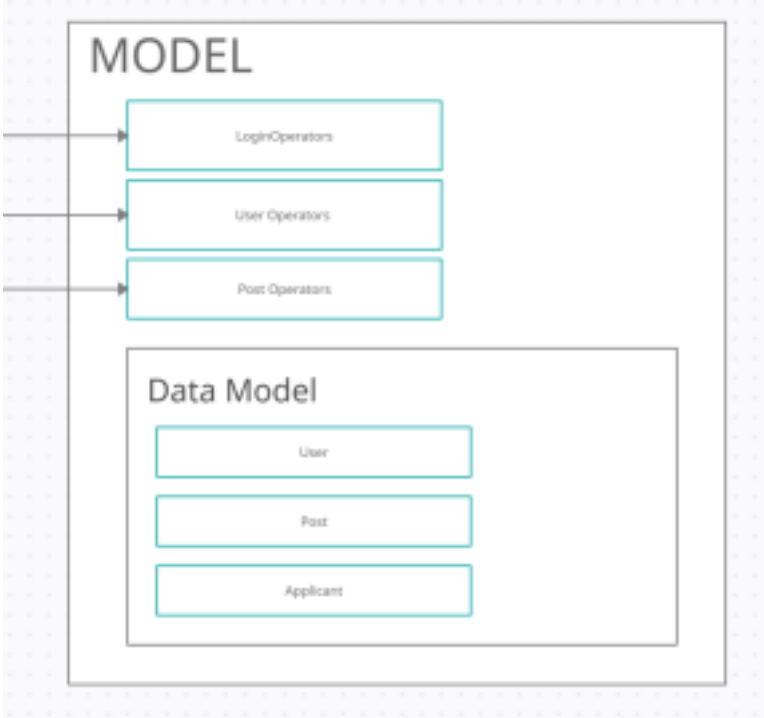
We kept low coupling in the UML diagram so our systems are independent from each other. We did this so each subsystem didn't have a high impact on another. This allows us to change our model as needed. If we need to change one subsystem, we shouldn't have to worry about changing everything else about our diagram and plan. We kept high cohesion to show the associates between similar tasks within subsystems. We did not want low coherence in this, because it would lead to a lot of unnecessary objects and no associations.

II.2. Subsystem Design

II.2.1. Model

The Model stores all the data models that the application will need. These data models are the core components of the application as they accept user input and save them in a database format. The Controller and View functions will then take this data and either store, compute, or display it back to the user.

Currently in iteration 1 there is Post, Application, and User. The User Model takes an id, username, email, password, userType, and posts. The username, email, and password all accept a string input; the userType will accept a selectable item (Faculty or Student). The post model takes an ID, user_id, title, body, timestamp, username, and applicant relationship. Variables title, body, and username will take in a string; applicants will relate to the model application. This brings us to the final model Application, which takes an ID, firstname, lastname, email, phone number, body, and post_id. Variables that take a user input string are firstname, lastname, email, phone number, and body; while the id and post_id keep track of which post the application was submitted to.



II.2.2. Controller

The role of the controller is to receive information from a database model, and then pass it to the view subsystem. Generally this is done with routes and forms, forms are used to save the data from a model format to a form format, and then a route passes a form to the desired html page to be rendered.

The role of the subsystem is to organize the controllers code to make things easier to follow. Subsystems can therefore be replaced or changed without having to replace or extensively change other parts of the controller.

The subsystems within our Controller are as follows: __init__.py, AuthForms.py, AuthRoutes.py, errors.py, forms.py, and routes.py.

AuthRoutes and Routes manage what View subsystem the database models will be displayed in, while AuthForms and Forms manage the database model data to then be passed to routes.

AuthRoutes requires the use of AuthForms, these subsystems focus on anything that pertains to authorization. AuthForms takes information necessary for Login and Registration, passes this information to AuthRoutes to then display to login.html and registration.html. Similarly, Routes requires the use of Forms which takes anything that doesn't need authorization. Forms takes Post, Application, and Edit; which passes information to routes for displayProfile.html, editProfile.html, index.html, and _post.html.
Table 1- Use an appropriate title for the table.

Methods URL Path Description

['GET','POST'] /index Will follow the route to index.html, which will be the home page for the application, displaying post / ways to apply to select posts.

/createpost
/createApplication/<post_id>
/display_profile
/edit_profile
/register

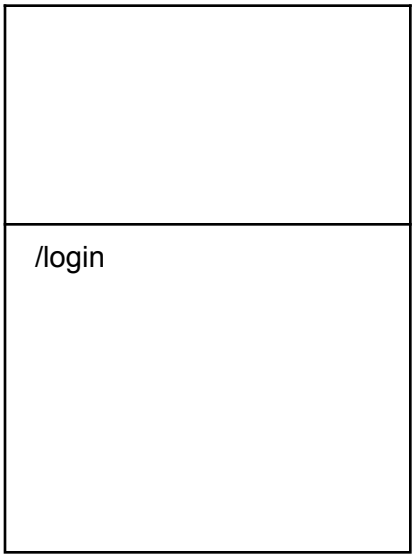
['GET','POST'] Will follow the route to createpost.html, which will display textboxes for the user to input information about a new post.

['GET','POST'] Will follow the route to _createApplication.html, which will display text boxes for the user to input information and relate that information to a particular post.

['GET'] Will follow the route to displayProfile.html, which will display the user's information from when they registered.

['GET','POST'] Will follow the route to editProfile.html, which will redirect the user to textboxes in which they can change the user information they inputted when they registered.

['GET','POST'] Will follow the route to register.html, which will bring the user to several text fields, 3 of which will be username and password validators to then store to the db. Once submitted, the user will get redirected to login.html.



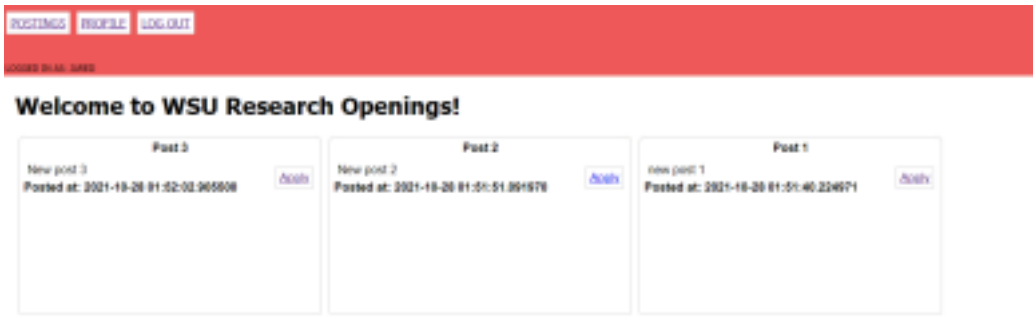
['GET','POST'] Will follow the route to login.html, which will display 2 text boxes for username and password. Once a user submits, the login will check db for the user, if the user exists they will get redirected to the index.

['GET'] /logout Will redirect the user to routes.index, which requires a login, which will redirect to login.html.

II.2.3. View and User Interface Design

Briefly explain the role of the view. Explain how you will build the user interfaces. Mention the frameworks or the libraries you plan to use.

The role of the View system is to house all the front end code for the web application. Front end code consists of .html files, images, and css (styling) files. These values allow the information from Controller and Model to be displayed to a webpage. To build the user interfaces we will include a form with the html page that contains data from a database model, this form will either contain a user input field, or display information to the page. Additionally the information will be styled by css inputs that make the page look better. We plan on using the framework flask, which will allow us to use forms to display on an html page.
So far our interface includes a main page with a navigation bar, that allows the user to proceed to their profile, creating a post (if faculty), logging out, or back to the main page (postings). The use case that we utilized for this interface was "Apply to research positions".

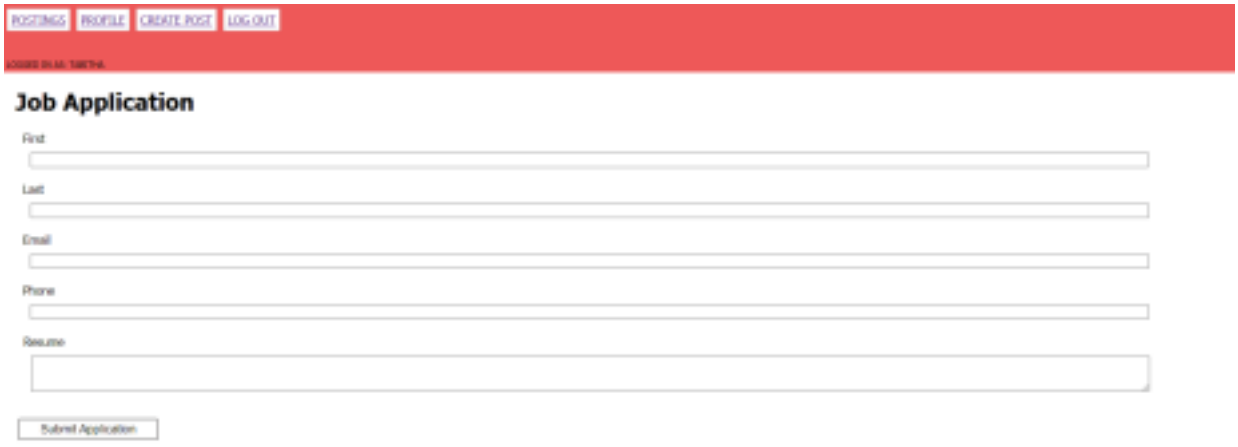


Profile will redirect the user to another page that includes all of their information as shown

below. This was an additional feature that we were able to add in addition to the other use cases.

When the user is a faculty member they will be able to create research posts. The create post button on the navigation bar will redirect the user to a page that allows them to include a title and body to their post. The use case that we utilized for this interface was “Create Research Positions”.

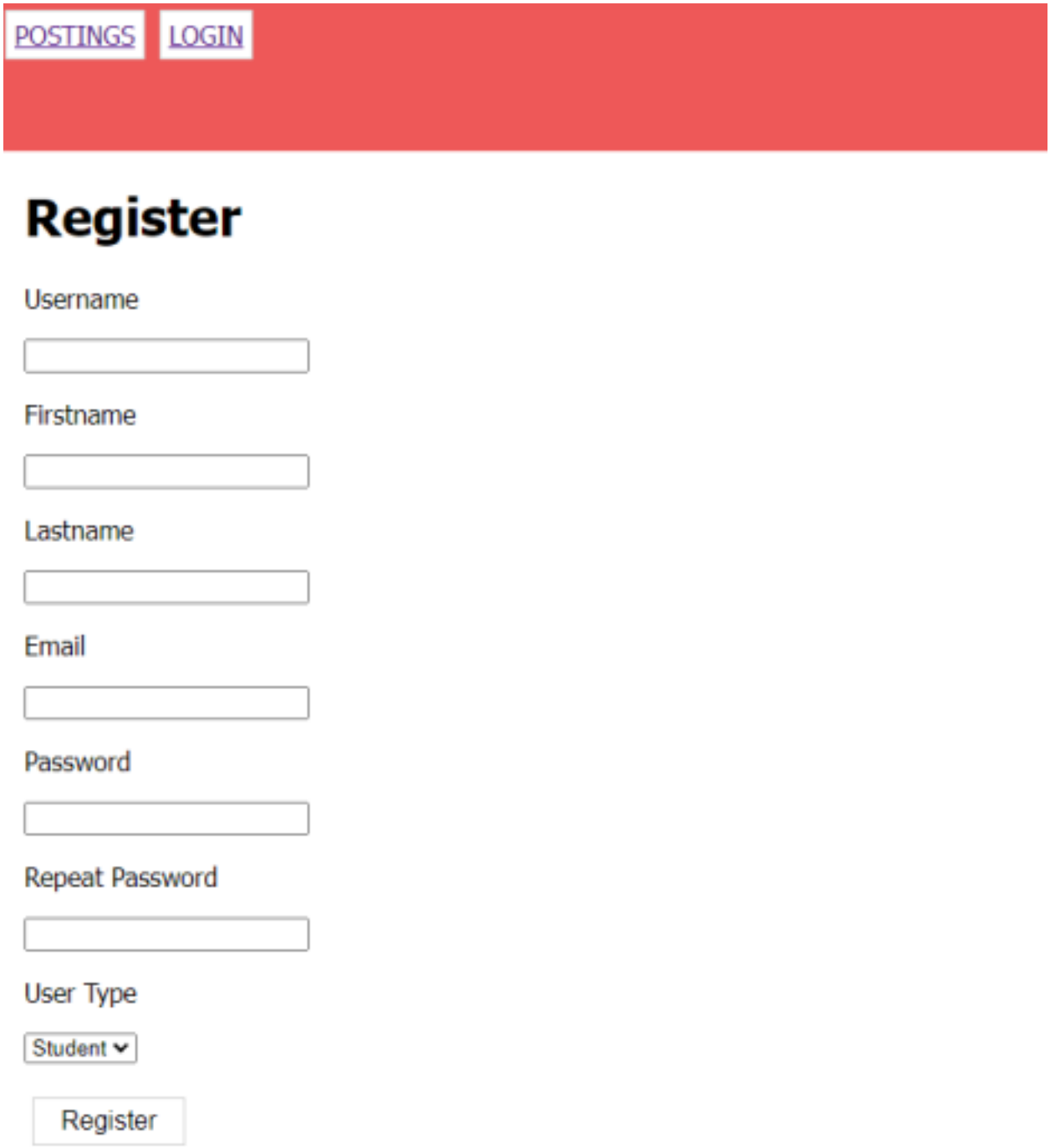
On the post itself there is a button that reads “apply”, this button will direct the user to another page that allows them to submit an application for the post that was made. The use case that we utilized for this interface was “Apply to Research Positions”.



There is the option to Log Out as shown on the navigation bar, this will direct the user back to

the login screen. The use case that we utilized for this interface was “Login”.

Finally there is the Click to Register shown above, this will direct the user to a page in which they can make an account to sign in. On this page you can select the option to sign in as a student or faculty member. This option allows only faculty members to make research posts. The use case that we utilized for this interface was “Create Account”.



III. Progress Report

Iteration 1:

Our goal for iteration 1 was to finish registration for faculty and student, login, faculty post creation, and student application. We have successfully implemented the registration for both students and faculty. Once a student is logged in the student can fill in information about themselves that would be relevant to a job application. A faculty member once logged in is able to make a post about the position they want filled. They can currently put in a title a description of the job. Once the student is logged in they are able to view posted jobs and submit an application.

Iteration 2:

The goal for iteration 2 was to allow faculty to add more contextual information to their job post and we also added functionality for the student to be able to apply to multiple research positions. The faculty member after a student has applied for their position will now be able to see a list of each applicant.Also we did a complete overhaul of our frontend by implementing Bootstrap to add an interactive user experience.

IV. Testing Plan

In general most of our tests will be done through unit testing. We are doing this so we can save time when it comes to manually testing our progress. We plan on using TDD (test-driven development) where we will make our test cases first and run them

- **Unit Testing:** For our unit testing we will use the “unittest” framework. We will make test cases to test out models and our routes. For our route tests we will go through the happy path of registering, logging in, and from there depending on whether the user is a student or faculty member, will make a post or apply to a job. These test cases will cover almost all of our routes. For the routes that aren’t covered by that path, such as display and edit profile, we will make a separate test suite. For testing our models we will make multiple test cases that work on making new users, committing the changes to the database and checking that they exist within the database.

- **Functional Testing:** We will do our functional testing manually. We will go through our use cases and make sure each functionality we implemented works as we want. We will do a “happy path” then try to break it by inputting unexpected inputs.

- **UI Testing**:We will run multiple tests such as changing screen size, different browsers, and different devices. This will give us a good idea if our user interface will look up to standard across multiple devices and browsers.

V. References

For the websites, give the title, author (if applicable) and the website URL.

Lucid Chart, UML Class Diagram Tutorial,
https://www.lucidchart.com/pages/uml-class-diagram