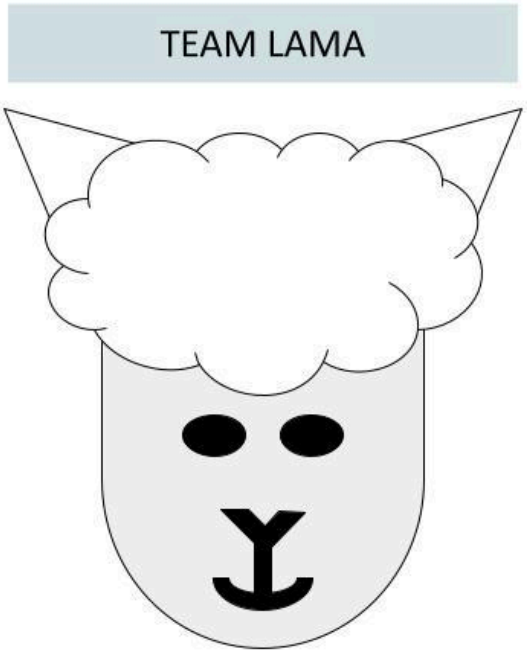
Research Position Search App

Design Document

10/27/2021

Iteration 1



Albert Lipaev, Mitchell Kolb, Louis Kha, Alex Castillo

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I. Introduction

The purpose of providing this design document is to show and explain the overall architecture of the Research Position Search App we are developing.

This is the first iteration of this document; there are no revisions to summarize. All future iterations will be based on this initial document.

The Research Position Search App is a web application that allows college professors and undergraduate students to connect and communicate to each other on research projects. This application has two sides, a student section and a faculty section. The student section will allow for students to create an account that contains their profile information and lets them apply to faculty posted research positions. The faculty section will allow those users to create postings that students can apply to, and to get into contact with students who have applied to existing postings to potentially interview for the position. Our project goal is to create a web application that executes the idea that is described above.

Section I includes the introduction of the design document and a description of the project

Section II includes the architectural and component-level design which describes the MVC pattern that our application is using.

Section II includes a progress report on how the application is coming along during the work that has been completed in iteration 1 and the problems our group has overcome during this time.

Document Revision History

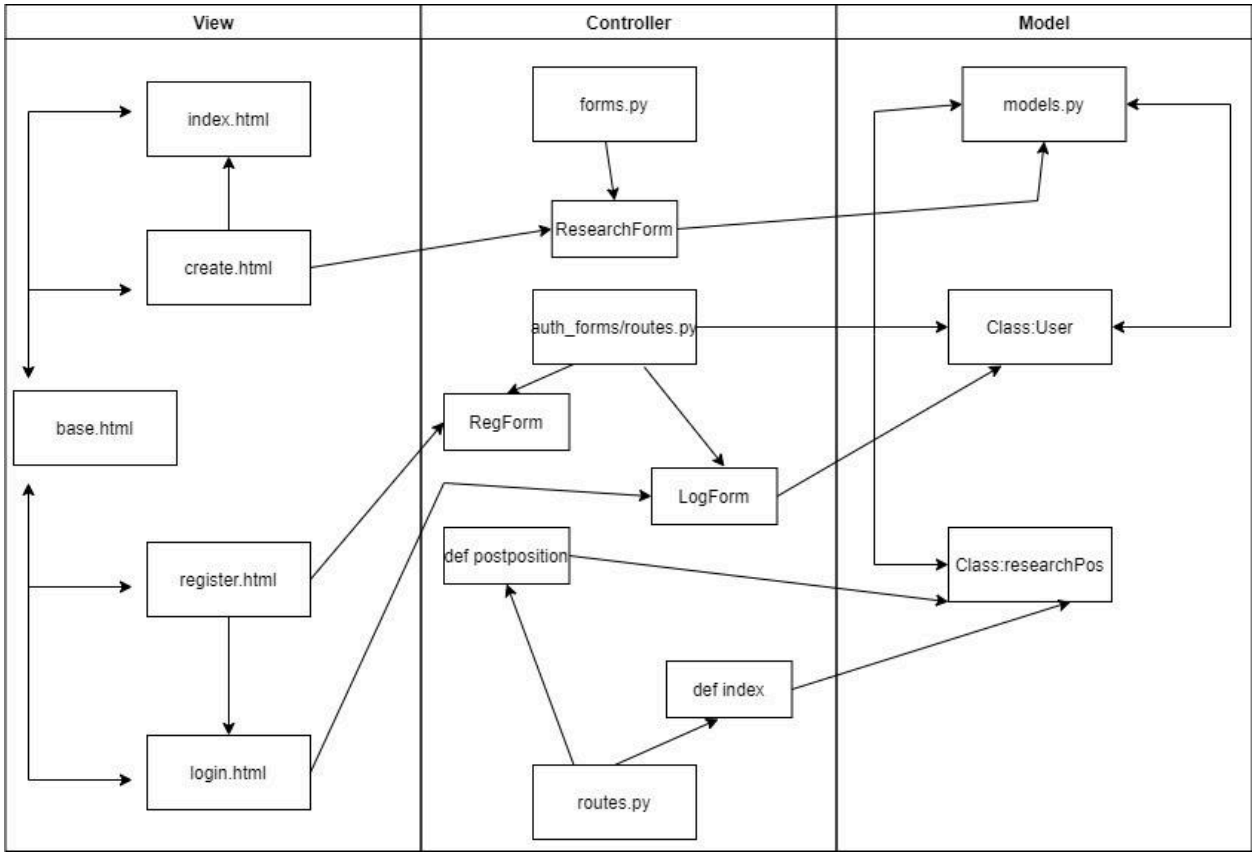
Rev 1.1 <2021-10-27> <Iteration 1, basic skeletal framework>

II. Architectural and Component-level Design

II.1. System Structure

For our implementation of the Search Application, we have adopted a Model-View-Controller pattern in order to organize our architectural pattern. We use this pattern as it most benefits the ability to work in a team as modification does not affect the entire model when using a MVC approach. This means we have support for asynchronous techniques when programming as well as the ability to provide multiple views while keeping down on the amount of possible merge conflicts when collaborating together. This keeps the amount of interdependence between modules low as the number of elements that belong inside a module together when being worked on can be largely controlled.

UML Diagram:



This diagram provides a focus on the currently implemented subsystems in our software between the View, Controller, and Model modules.

In the View modules, base.html connects to each page as it provides hyperlinks to each of them which while not always needed also increases speed for testing purposes. The index.html receives information from create.html which acts from the ResearchForm in forms.py in order to create and display new research positions. Both login.html and register.html have their own forms from auth_forms/routes.py respectively in which it allows them to enter information to either login using a created account, or to register a new one. This information is saved into the database along with everything else. The routes.py has definitions in place as postposition and index which allow for GET and POST methods to be implemented which assist in the creation of new research positions or the displaying of them. All these modules eventually are run through the Model modules as the Class:User and Class:researchPos from models.py maintain the db.models for everything to be possible, holding the information that consists of user accounts and research positions respectively.

II.2. Subsystem Design

II.2.1. Model

The Model acts as the core of the program. We keep all of our data models and algorithms in this portion of the program. The model doesn't depend on the controller or the view.

In iteration1 we have a user model, post model, tags model, and a research position model.

We use the User model lightly as an “abstract” model, where we can use it for both student and faculty users. But this method is proving to be a little more difficult than we first thought. Since this proved to be a little more challenging we decided to include two separate models, one for the Student and Faculty with their respective attributes. We also had some difficulties with the research positions so we- for the time being- recycled the post model and we will integrate a proper form of the research position model in the program.

The User model contains the attributes:

id

- Contains a user’s unique id

username

- contains the user’s set username

email

- the user’s contact username

password_hash

- is the user’s set password

post

- the posts associated with the user

isfaculty

- a boolean that represents if the user is a student or a faculty

The researchposition model contains the attributes:

id

- Which is the unique id of the post

title

- The title of the post

researchDesc

- Is the description of the research position

startEndDate

- indicates the duration of the research position

requiredHours

- is the minimum amount of hours required to be dedicated to the research position

researchFields

- is the topic the research position pertains to

requiredQualifications

- is the qualifications an applicant needs to be eligible for the position

timestamp

- The time when the post was initially posted

The tags model contains:

id

- a unique id of each tag

name

- whether or not the user is a student or faculty.
    - This will most likely be changed in iteration2

II.2.2. Controller

Role of the controller:

In the program, the controller is used for several reasons. First is the authentication, mainly handling of the data used in the authentication process (Auth_Forms), and handling the data stored in the database, for the processes such as login, registration and logout (Auth_Routes). Second is the error handling, which processes exceptions and provides an output based on the exception/error. Current iteration is not supporting the post function, however the code for handling the post events for the users is written in the program. This subsystem mainly handles the data input and storage for the posts, as well as the attributes such as “liking” the posts or counting the number of posts.

Subsystems:

Authentication subsystems:

1. Auth_Forms:
    - RegistrationForm interacts with the register method from auth_routes providing an instance of the data input class for registration.
    - LoginForm interacts with the login & logout method from auth_routes. Provides an instance of data for login.
    - Interdependencies: Utilized in the Auth_Routes.
2. Auth_Routes:
    - Login method checks whether the input data matches what is stored in the database. If the data is not found, it will print an error on the screen. Otherwise redirects to the main page (index.html). Logout method logs out the current user and goes back to the login screen.
    - Registration form
    - Interdependencies:

Errors subsystem:

3. Errors:
    - Used to display either error 404 (data not found on the server), or to display error 500 in case the server experienced an unexpected internal error.
    - Interdependencies: db

Post handling subsystems:

4. Forms
    - Handles the creation of Form classes in which the structure of the forms are created and assigned Field types in regards to the data they hold and possible validators necessary for error checks
    - These Forms interact with other subsystems such as the model and route subsystems which respectively hold the database classes for the forms to interact with and the routes in which data is addressed to it.
5. Routes
    - The subsystem is primarily used to be able to send information between GET and POST methods with ease. This manages the route definitions such as postposition() and index() which allow for the information to be sent to the ResearchForm class as well as the index.html page respectively.
    - The subsystem interacts with other subsystems such as the Form class in which it ties the data from the researchPos form which is inputted inside of the create.html page and used to fill the database model for ResearchForm.

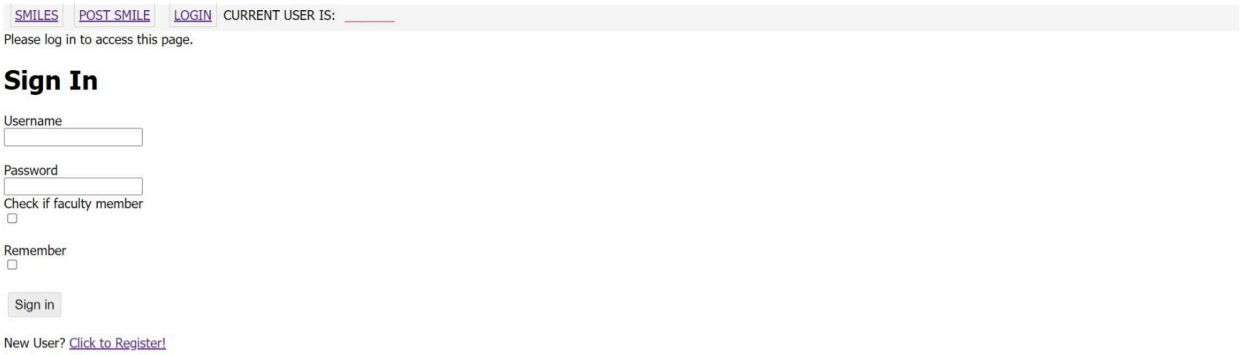
Methods	URL Path	Description
Login	http://127.0.0.1:5000/login?next=%2F	When user first enters the app, the first screen to appear is the sign in
Register	http://127.0.0.1:5000/register	When the register element is clicked, the registration screen appears
Login, Register	http://127.0.0.1:5000/index	After either login or registration, the user is directed to the index.html, which is the main(home) screen
Create	http://127.0.0.1:5000/create	When the create element is clicked, the create.html is loaded which allows user to create posting about research position
EditProfile	http://127.0.0.1:5000/editprofile	When the edit profile element is clicked, the editprofile.html is loaded which allows user to edit the profile info

II.2.3. View and User Interface Design

View stores the html pages for the application. It creates the front end for the application, which is presented as a user interface (UI) and is a visual representation of the working code.

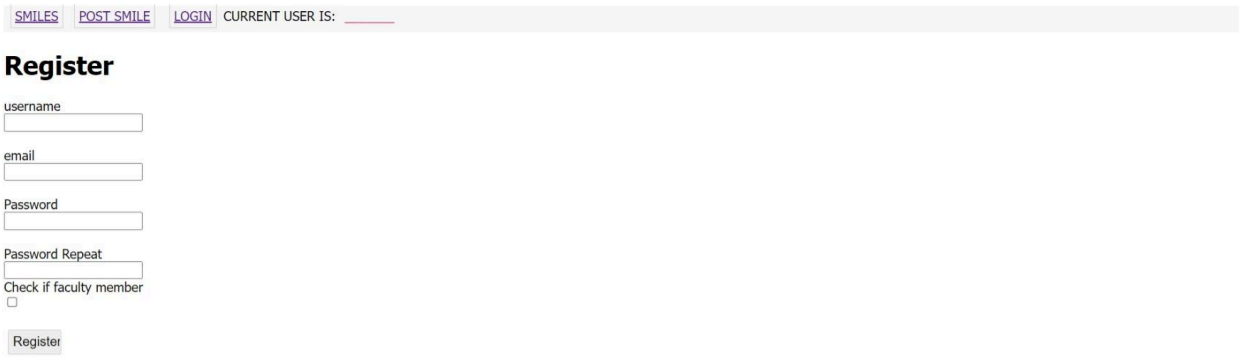
From the beginning screen of the application when launched, appears sign in screen (login.html):

Use case 1



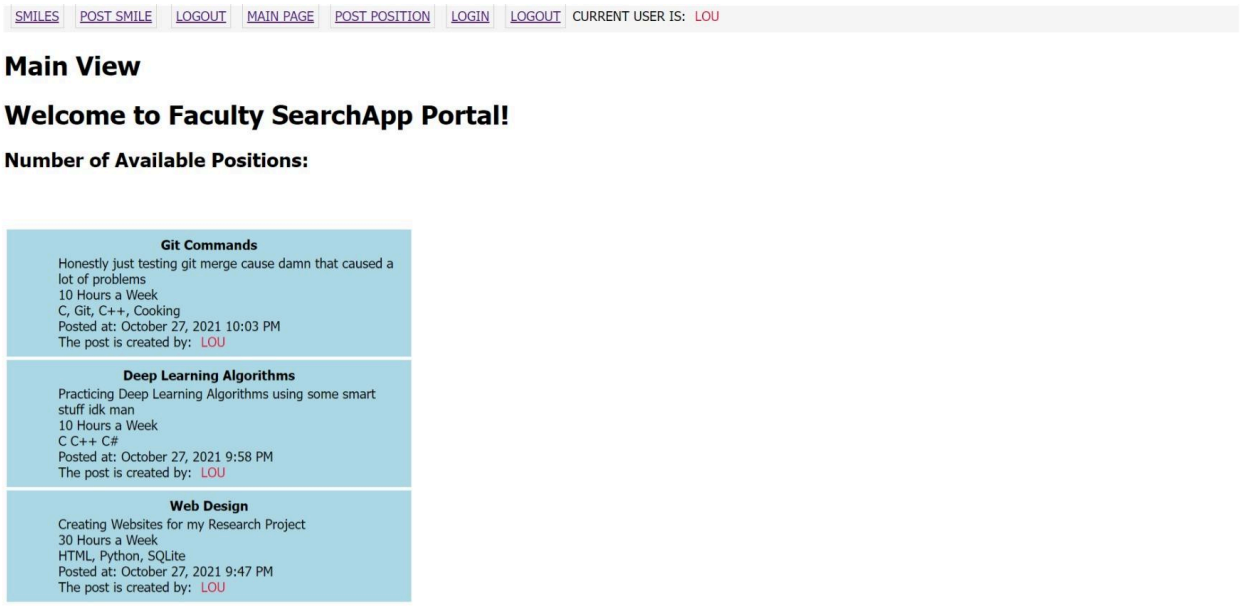
The user has a choice to register (register.html) in case of having no account:

Use case 4



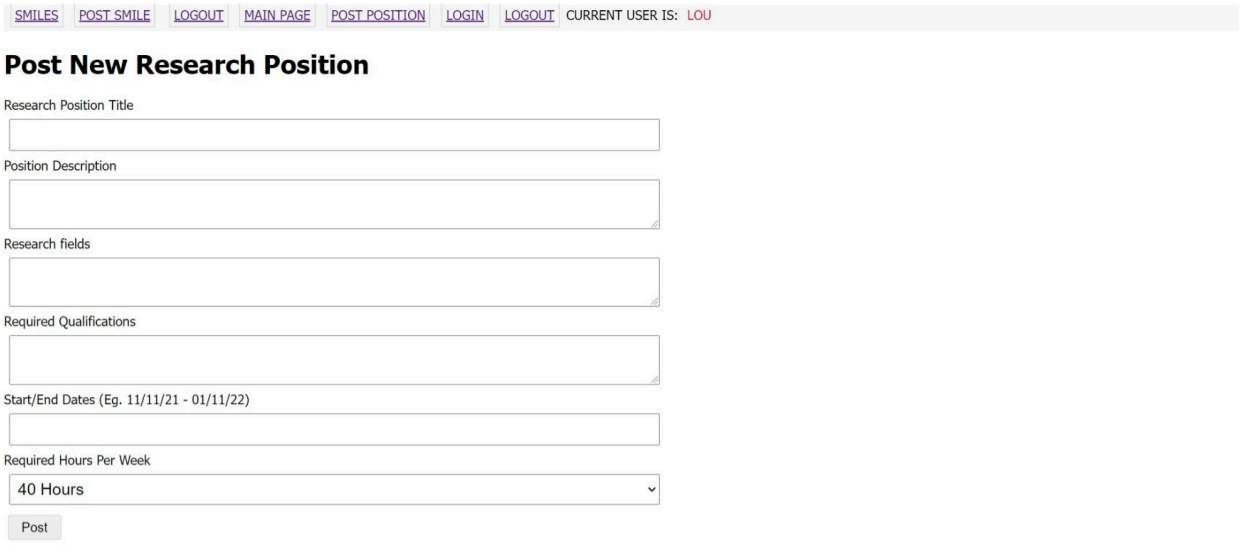
Following either path of sing in(login.html) or register(register.html) will lead a user to the home screen of the application(index.html):

Use case 2, 3, 6, 7



The application also is able to utilize the function of posts creation. The display of the posts is implemented by using: _post.html. For the creation of the posts, the create.html is utilized:

Use case 2, 11



It also can be noted that html files such as: create, _post, index, login & register all utilize the same header containing all of the UI buttons which will redirect users to the chosen screen(base.html). To achieve this, all of the above files are created as an extension for the base.html:

Use case 2

```
{% extends "base.html" %}
```

III. Progress Report

The progress our group has made during iteration 1 has hopefully created a solid framework for the future iterations. So far we have worked on outlining the sections of this project to see if we can separate the bigger problems into smaller more easily digestable ones. This has helped us when writing down the routes for the entire application. We have also worked on creating the database model for the students and faculty members along with base pages for what will become the student and faculty sections of the application. Lastly we have worked on the “login” and “create new account” functionality of the application so the user can have access to their own account and so the application will know if they are a student or faculty member and direct them to the correct page.

IV. References

Cite your references here.

What is MVC? Advantages and Disadvantages of MVC - Interserver Tips. (2021). Retrieved 28 October 2021, from https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/

Quanit, M. (2021). Software Engineering Principle — Coupling & Cohesion. Retrieved 28 October 2021, from https://dev.to/mquanit/software-engineering-principle-coupling-cohesion-1mma