Findr Application

Design Document

10/26/2021

Version 1

TeamBATS

Aiden Dickson, Ben Kaufmann, Daniel Semenko, and Taylor West

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I. Introduction

This is a design document which outlines the software design structure of the Findr application. This document goes into detail on the functions of each of the major subsystems and their cohesion with one another. Findr is an application which allows students to apply for undergraduate research opportunities posted by faculty. The faculty are then able to review the applications sent in by students and update their statuses to their choosing. The goal of this project is to fulfill the desired requirements outlined in the Software Requirement Specification document in an organized and timely manner.

Section II includes descriptions of the high-level architecture (Model, View and Controller) of the software. This section details the roles of each of the major subsystems and provides diagrams to help visualize the cohesion of the architecture.

Section III includes a progress report which summarizes the progress made during the iteration. This section is updated whenever a new iteration in completed.

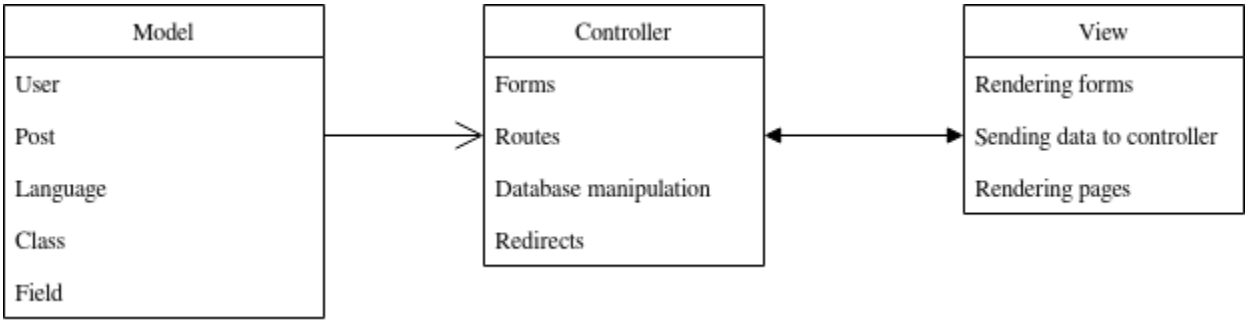
Document Revision History

Rev 1.0 2021-10-26 Initial version

Rev 2.0 2021-11-15 Iteration 2

II. Architectural and Component-level Design

II.1. System Structure



We used the Model-View-Controller (MVC) architecture for this project because it's well suited to our project. MVC works best for projects that have minimal processing of data that comes from the user but poorly suited to realtime applications like multiplayer video games. MVC projects can be seen as projects that simply present an interface to users for viewing and interacting with a database.

The model subsystem handles the database. It creates database models and outlines the structure of interdependencies between models. It also implements interfaces for safely manipulating the database so that changes to the database model don't affect the way the controller interacts with the database. The model is basically an abstraction over the actual database that's stored on disk and manipulated through a library (in this case we used SQLAlchemy).

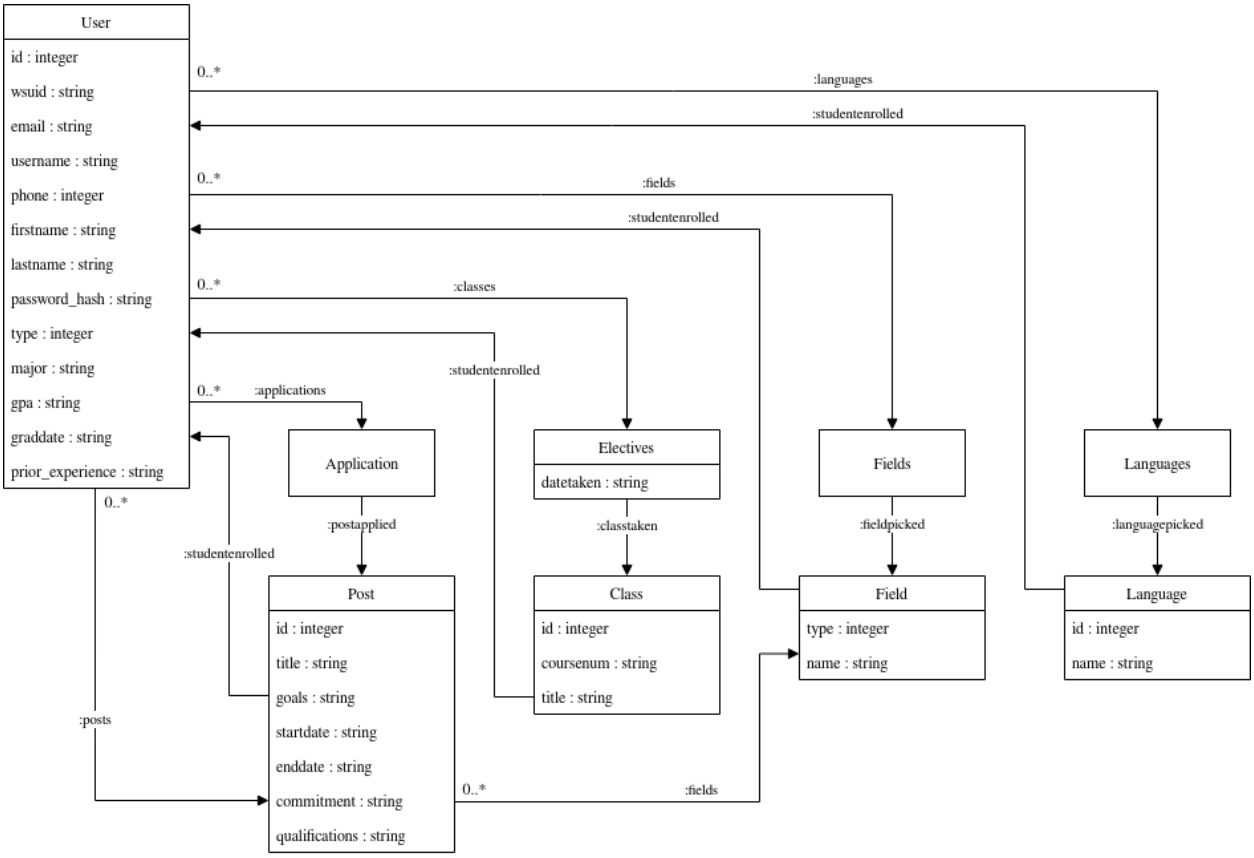
The controller implements the business logic of the product and the view subsystem implements page designs and styling information. They are first processed by the controller subsystem and then sent to

the user's computer. The controller receives data submitted through forms and responds by interacting with the database and sending a response to the user.

Breaking the project apart like this keeps data and data manipulation separate (model/controller) and data manipulation/logic separate from the visual presentation (controller/view). This keeps coupling low and coherency high.

II.2. Subsystem Design

II.2.1. Model



The model is what directly manages any data, logic, and rules for our application. It is connected to our database, and to the controller. The user never interacts with the Model. The student or faculty accounts, research opportunities, and the database holding all this, are some examples of parts of the model of our application.

List of all of the tables in our database:

- User
    - The user table is one of the main models of our application. A student or faculty will create an account, and all their information (even more than just base contact info) will be stored here. There are many relationships stemming from the User table, including posts (faculty), classes taken, preferred field, and coding languages. These connect to posts.
    - Attributes:
        - id - a user ID that will keep track of which user is which

        - wsuid - A user's WSU-provided ID number
        - email - The user's contact email
        - username - The user's username (WSU provided email)
        - phone - The user's phone number
        - firstname - The user's first name
        - lastname - The user's last name
        - password_hash - The user's password
        - posts - All the (faculty) user's posts
        - type - Denote whether user is student (2) or faculty (1)
        - major - The user's major
        - gpa - The user's gpa
        - graddate - The user's expected graduation date
        - prior_experience - User inputed previous research experience
        - classes - User inputted previous classes taken
        - fields - User selected fields they are interested in
        - languages - User inputted languages they have experience with
        - applications – a relationship linking the User and their posts they applied to
- Class
    - The class table will create an instance of some class that a student may or may not have taken. They are identifiable by an id, and backpopulate to student_enrolled (in Electives, Fields, and Languages)
    - Attributes:
        - id - an ID that will keep track of which class is which
        - coursenum - a classes corresponding course number (not ID)
        - title - the title/name of the class
        - roster - stores all (student) users who take a specific class, a relationship backpopulating to classtaken (in Electives)

- Language
    - The language table is used by users to show what coding languages they have previous experience with. Selected through a multipleSelectField. Will be related to a specific student.
    - Attributes:
        - id - an ID that will keep track of which language is which
        - studentid - the user ID of the (student) user that the language(s) is related to
        - name - the name of the language
- Languages
    - Connects with the Language table. Where a Language model is a single language, a Languages model stores many individual Language models.
    - Attributes:
        - studentid - the ID of the user to which the languages are connected to
        - languageid - the ID of individual language models of which the languages model will hold
        - studentenrolled - A relationship connecting the languages model to the User model
        - languagepicked - A relationship connecting the languages model to the Language model

- Field
    - Similar to Language, the field table is used by users to share what their preferred research field may be. Will relate to a specific student
    - Attributes:
        - id - an ID that will keep track of which field is which
        - studentid - the user ID of the (student) user that the field(s) is related to
        - name - the name of the field
- Fields
    - Connects with the Field table. Where a Field model is a single field, a Fields model stores many individual Field models
    - Attributes:
        - studentid - the ID of the user to which the fields are connected to
        - fieldid- the ID of individual field models of which the fields model will hold
        - studentenrolled - A relationship connecting the fields model to the User model
        - fieldpicked - A relationship connecting the fields model to the Field model
- Electives
    - Used by users to show what elective(s) a (student) user has taken. Connected to the User table and the Class table
    - Attributes:
        - studentid - the ID of the user to which the electives are connected to
        - classid - the ID of individual class models of which the electives model will hold
        - datetaken - a string date holding the date of when the class was taken
        - studentenrolled - A relationship connecting the Electives model to the User model
        - classtaken - A relationship connecting the Electives model to the Class model

- Post
    - The post model will create an instance of a research opportunity by a faculty, that a student may apply for, and get accepted or denied by a faculty
    - Attributes:
        - id - an ID that will keep track of which post is which
        - title - the title/name of the research opportunity
        - goals - the desired goals of the research opportunity
        - startdate - the start date of the research opportunity
        - enddate - the end date of the research opportunity
        - commitment - the necessary/potential commitments a student may have to follow of the research opportunity
        - qualifications - the required qualifications of the research opportunity
        - view_post – relationship connecting the post to the students who applied for the position
        - fields - the fields that the research opportunity relate to
        - user_id - the ID of the user that created the research opportunity post

II.2.2. Controller

The controller is the interface between the Model and the View. It takes requests/inputs/interactions from the View and manipulates data/calculates logic/updates the database from the Model. In our program. It will take inputs from students/faculty and manipulate the models to perform the desired outcomes of our application.

- Initialize App
    - The Initialize App subsystem is responsible for actually creating and initializing the application, loading databases and configuring application defaults. This is so the application can be immediately loaded and functioning after launch.
    - This subsystem is responsible for configuring the application defaults, initializing the database, and registering blueprints for the routes to function, in addition to directly connecting to the Login Manager.
- Login Manager
    - The Login Manager subsystem is responsible for the session management of the application. It creates and stores a user ID (to connect to the models), lets users log in and out, and restrict views of certain views based on whether the user is logged in (connecting to View). If the user is a student, they can connect to studentregister, and are not able to create posts. If the user is a faculty, they can connect to faculty register, and access the post creation route. The student registration/faculty registration are also part of the profile management subsystem
- Profile Manager
    - The profile management subsystem is responsible for allowing users to edit and view their profiles, during the creation of their account and in their edit profile page. Faculty are able to view the profiles of student's who display interest in their posts. The profile management subsystem connects with the Login Manager, pulling most of the profile information from the initial account creation. It also connects with the Post Manager subsystem, allowing faculty to pull student profile information from a post.
- Post Manager
    - The post management subsystem is responsible for allowing users to interact with research opportunity posts. Only faculty are allowed to create (or delete) these posts, while both students and faculty can view a list of postings. The subsystem allows users to view an individual posting, and lets student users apply for the position of the research opportunity. Faculty can view the profile information of student users who applied to their posts, meaning the Post Manager subsystem is connected to the Profile Manager subsystem.

Our application includes many different routes that connect the users to the subsystems of our application. Our Login Manager subsystem includes studentregister, facultyregister, login, logout, and edit profile. Our Profile Manager subsystem includes studentregister, facultyregister, display_profile, and edit_profile. And our Post Manager includes post, delete, and opportunities

Route	Methods	URL Path	Description

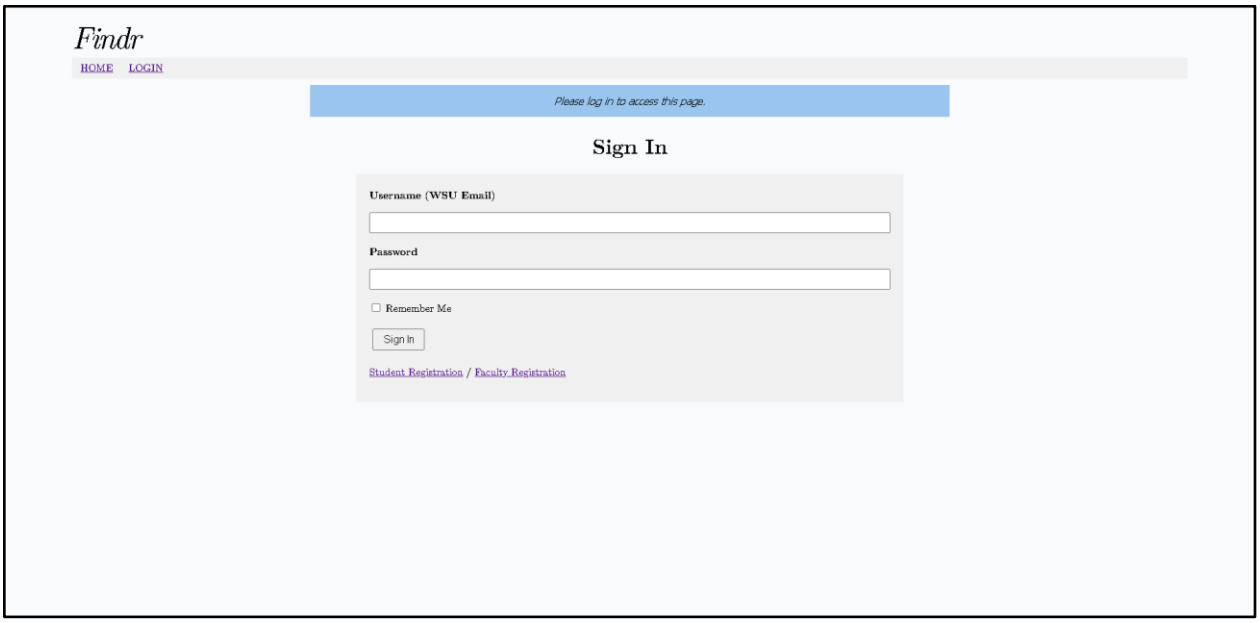
index	GET, POST	‘/’, ‘/index’	The index route is responsible for rendering the ‘index.html’ main page, titled “Findr Home”
post	GET, POST	‘/post’	The post route is responsible for (if the user is faculty) displaying a postForm, rendering it on the ‘create.html’ post creation page. If a student attempts to do this, it routes them back to the index page.
delete	DELETE, POST	'/delete/<post_id>'	The delete route is responsible for (if the user is faculty) deleting the post identified by the provided post_id. It doesn’t display anything, just deletes the post from the database, flashes a success message, and returns to the index.
display_profile	GET	'/display_profile'	The display_profile route is responsible for rendering the ‘display_profile.html’ page, displaying the user’s profile information.
opportunities	GET	'/opportunities'	The opportunities route is responsible for rendering the opportunities.html’ page, displaying all research positions to the user. Here they can select an individual post to view.
studentregister	GET, POST	'/studentregister'	The studentregister route is responsible for allowing users to create a student account. Because sensitive login information is present, it is part of auth_routes. The route renders the StudentRegistrationForm on ‘studentregister.html’, getting user information from the form, and adding the user to the database.
facultyregister	GET, POST	'/facultyregister'	The facultyregister route is responsible for allowing users to create a faculty account. Because sensitive login information is present, it is part of auth_routes. The route renders the FacultyRegistrationForm on ‘facultyregister.html’, getting user information from the form, and adding the user to the database.
login	GET, POST	'/login'	The login route is responsible for allowing users to log in to their existing user accounts. Because sensitive login information is present, it is part of auth_routes. If incorrect credentials are provided, an error is flashed. If correct, the user is logged in, and the current_user is updated to the user_id. The route renders the LoginForm on ‘login.html’.
logout	GET	'/logout'	The logout route is responsible for allowing users to log out of their account. Because sensitive login information is present, it is part of auth_routes.
edit_profile	GET, POST	'/edit_profile'	The edit_profile route is responsible for allowing users to edit the fields of their account. Because a password is required, the route is part of auth_routes. The route renders the EditForm on ‘edit_profile.html’, and adds the updated information to the database.
apply	POST	‘/apply/<post_id>’	The apply route is responsible for allowing (student) users to click a button on the post pages, and this route would submit them (and their user information) to the specific post. The post page is reloaded, now with the apply button being replaced by an applied button (or hired/marked for interview depending on faculty input)
hiring	POST	‘/hire/<user_id>’	The hiring route is responsible for allowing (faculty) users to click a button on the post page, and this route would update the student’s view of the post page that they got hired for the position. The post page is then reloaded, with the hire button changed to a hired button.
markingforinterview	POST	‘/markforinterview/<user_id>’	The markforinterview route is responsible for allowing (faculty) users to click a button on the post page, and this route would update the student’s view of the post page, that they got approved for interview for the position. The post page is then reloaded, with the interview button changed to the hire button.
deny	POST	‘/deny/<user_id>’	The deny route is responsible for allowing (faculty) users to click a button on the post page, and this route would update the student’s view of the post page that they got denied for the position. The post page is then reloaded, with the hire button changed to a denied button.
unapply	POST	‘/unapply/<post_id>’	The unapply route is responsible for allowing (student) users to click a button on the post pages, and this route would remove them (and their user information) from the specific post. The post page is reloaded, now with the applied button being replaced by an apply button.

view_post	GET	'/view_post/<postid>'	The view_post route is responsible for displaying the information of the faculty position post.
myapplications	GET	'/myapplications'	The myapplications route is responsible for showing the current users a page containing their currently submitted applications. If a faculty tries accessing this page, it will redirect them back to index.html.

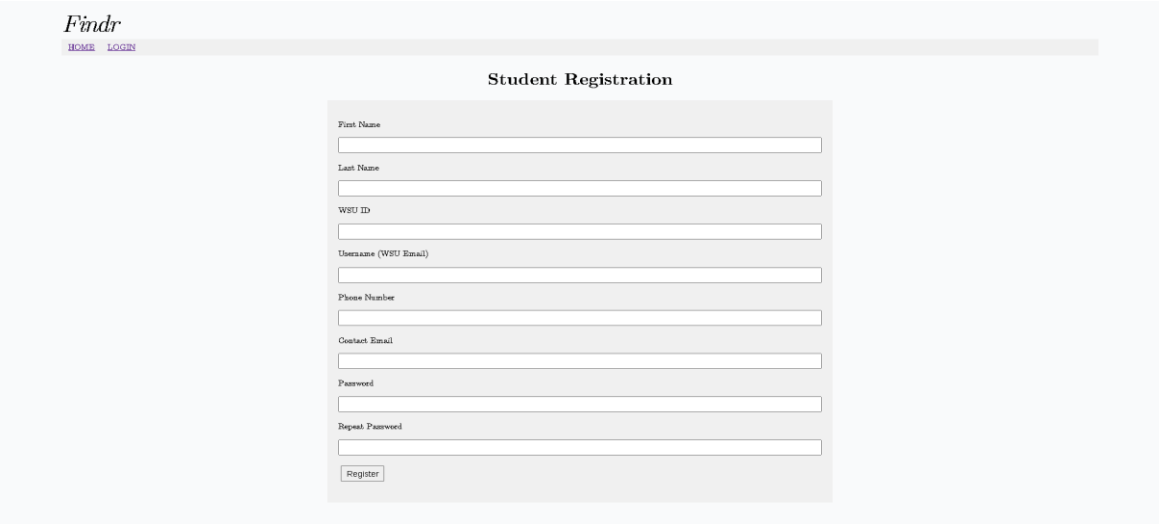
II.2.3. View and User Interface Design

View contains all the functions and files that directly interact and can be interacted with by the user. In terms of our project, the view files are what students/faculty will see and interact with in order to use the application. At this point in time, we do not plan to use any other frameworks or libraries.

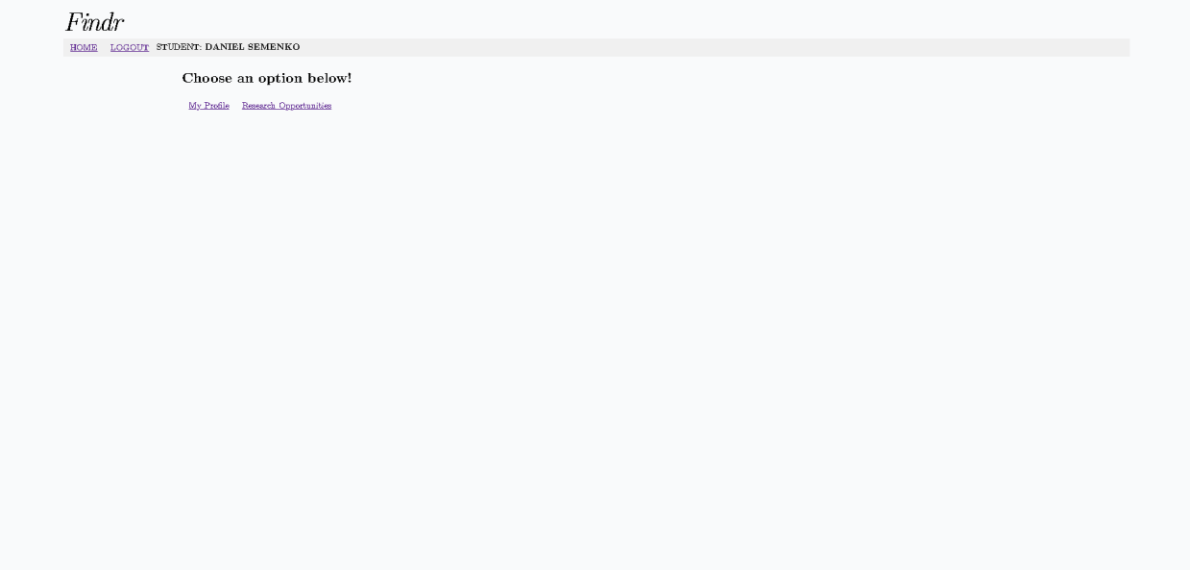
For Iteration 1, we have developed the overall framework of our application, in addition to the four use cases we have planned for this iteration. The first page we see when we start the application is the Login page. From here, users are only allowed to view the login page and the registration pages. They can enter their username, password, and sign in, or take one of the links to Student Registration or Faculty Registration.



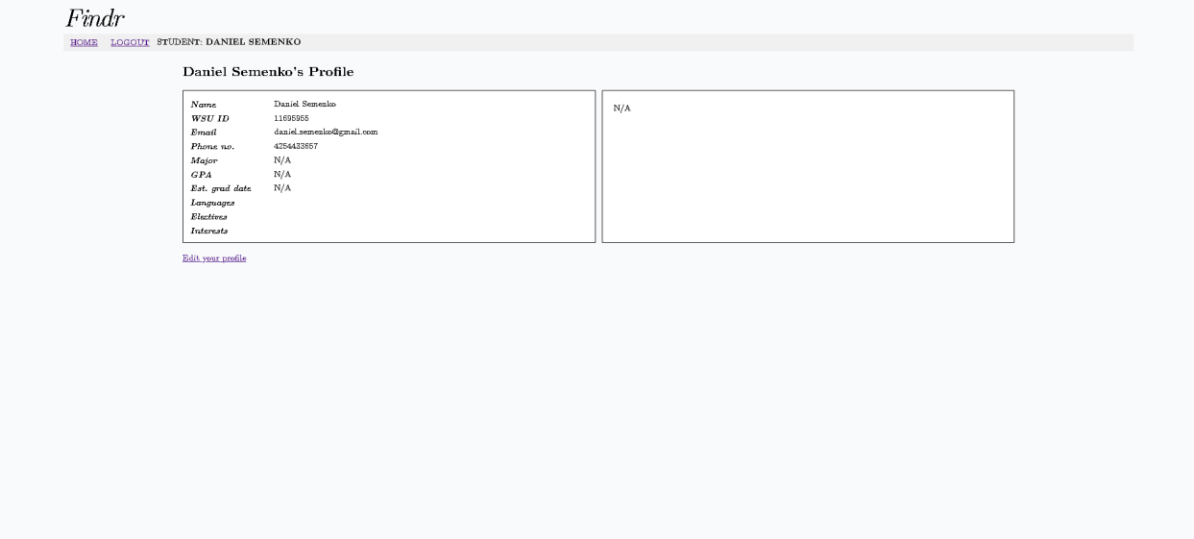
Next, corresponding to our first and second use cases, are the Student Registration and Faculty Registration pages. Here, the user will enter user information including first and last name, their WSU ID number, their WSU email as their username, their phone number, contact email, and password. The username is only accepted if the provided input follows the email format. The passwords are only accepted if they match. Once they fill in all the information, they can then press the register button, taking them back to the login page. Only a screenshot of the Student Registration is provided, since as of right now, their forms are the same.



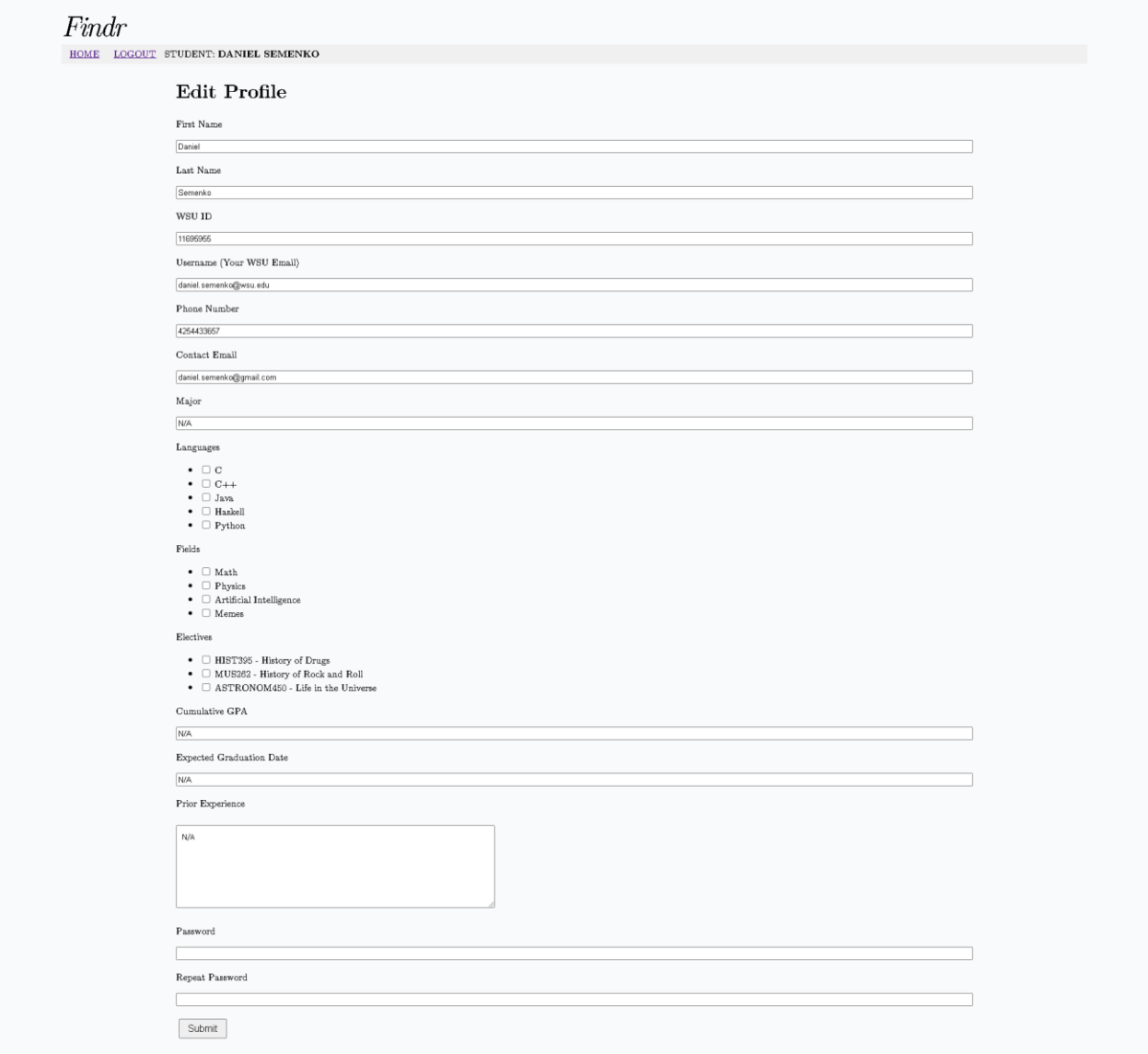
After successful login, the user lands on the index page. Since posts are being added in future iterations, our index page is pretty empty, but contains links to My Profile (display_profile), and Research Opportunities (contains nothing right now), and logout.



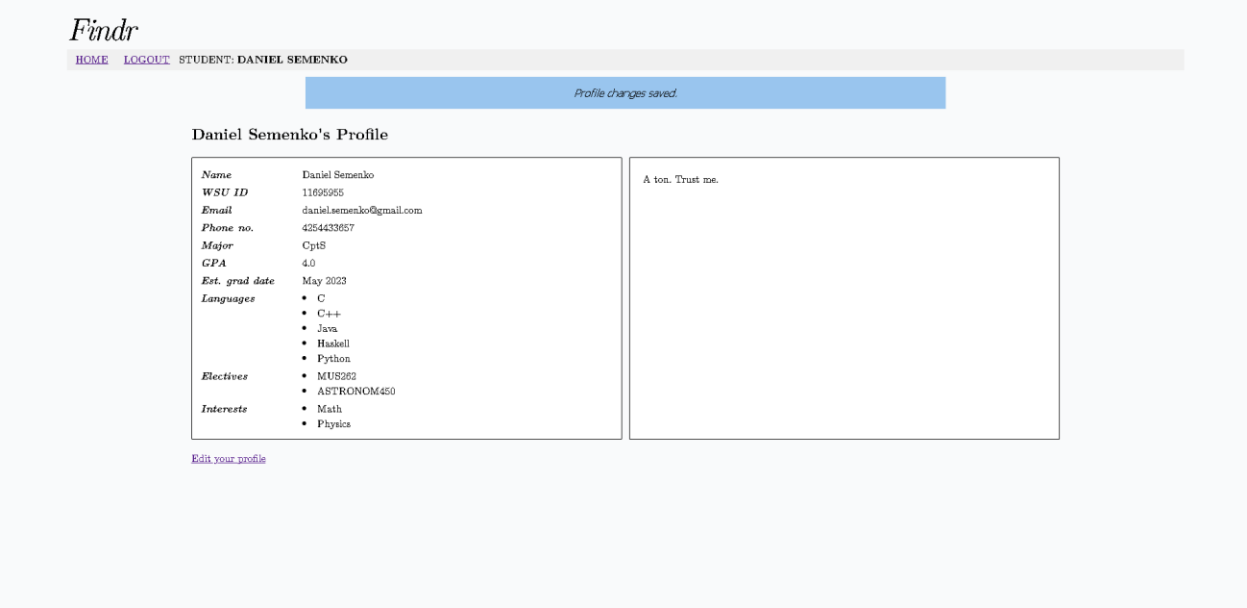
By clicking “My Profile” on the index page, you are taken to the display_profile page. Here, all the user’s (known) information is displayed, and a link to edit your profile is provided. If the user has not already edited their profile to finish adding the rest of their information, their profile page will naturally look bare bones. Here’s what the profile page will look like before the user edits their profile.



By clicking “Edit Profile” on the display profile page, you are taken to the edit_profile page. Here, the user can edit existing data, and finish their account by adding other data, such as Languages, Fields, Electives taken, Gpa, etc. The form will not allow the user to submit the form unless all fields are filled in (including correct, matching passwords).



After pressing submit, you are taken back to the Display_Profile page, where the updated profile information is displayed.



III. Progress Report

Iteration I began with a team meeting where it was decided to use the MVC pattern framework for this project. Coding began by setting up these three subsystems and initializing the application. After the html templates were made, the user model was created, and login/logout was implemented. A flag was put inside the user model to differentiate between faculty and student. CSS styling was added to the templates, and finally, other models were made for the posts and different fields for the student profile and posts.

Iteration II once again began with a team meeting where the duties of the iteration were assigned to each of the team members. Most of the programming was related to viewing and creating individual research positions. This included the “apply to research position” feature for the students and “create undergraduate research positions” for the faculty. The database model was also revised and a testing plan was created for future use.

IV. Testing Plan

- **Unit Testing:** We plan on using the *unittest* unit testing framework to test the following modules:
    - Editing the attributes of a profile and testing whether those changes were correctly committed to the database.
    - Registering a new profile and testing whether those changes were correctly committed to the database.
    - Deleting a post and testing whether the post appears in the database or not.
- **Functional Testing:** We plan to manually test each feature explained in the Use Case portion of the Requirements Document by going through each route and testing whether it works correctly.
- **UI Testing**: We plan on testing the User Interface by viewing all the different routes with an assortment of web browsers and window sizes to verify the UI is free of defects.