

Outreach Group
Outreach Contact
Software Design Document

Names: Jacob Dietrich, Abdel hamid Shehata, John Thomas, Megan Grant, Thomas Cheek, and Benn Mellinger
Lab Section: 01

University of Mary Washington

11/8/2021

Introduction	4
1.1 Purpose	4
1.2 Scope	4
1.3 Overview	4
1.4 Reference Material	4
1.4.1 Python - https://www.python.org/about/	4
1.4.2 Django - https://www.djangoproject.com/	4
1.5 Definitions and Acronyms	5
System Overview	5
System Architecture	6
3.1 Architectural Design	6
3.2 Decomposition Description	7
3.3 Design Rationale	10
Data Design	12
4.1 Data Description	12
4.2 Data Dictionary	12
Component Design	13
5.1 Create Account	13
5.2 Login	14
5.3 Logout	18
5.4 Admin Approve Post	19
5.5 Delete a Posted Job	19
5.6 View Job Post	20
5.7 Search Available Job Posts	20
5.8 Edit Job Post	21
5.9 Delete Job Post	22
5.10 Admin Delete Account	23
Human Interface Design	24
6.1 Overview of User Interface	24
6.2 Screen Images	25
6.3 Screen Objects and Actions	31
6.3.1 Create Account	31
6.3.2 Login	33
6.3.3 Logout	34
6.3.4 Create Post	35
6.3.5 Edit Post	36

6.3.6 Delete Post	37
6.3.7 Search Job Listings	38
6.3.8 View Post	39
6.3.9 Review Post	42
6.3.10 Admin Delete Account	44
Requirements Matrix	45
Appendices	46
8.1 Author Contributions	
8.2 Revision History	47

1. Introduction

1.1 Purpose

This software design document describes the architecture and system design of the Outreach Contact web application created for University of Mary Washington (UMW). The purpose of this web application is to facilitate the hiring of UMW students for tutors, childcare and pet care.

1.2 Scope

This software will be contained within the University of Mary Washington's community and alumni, and be used to connect the members of this community for reasons related to babysitting, dog-sitting, tutoring, or any other job that may be sought from within the community. The goal of this software is to streamline the process of the existing system for accomplishing this. The current process is a shared document between the members of the community; this system will digitize this process. The benefits of the creation of this software shall be to create a space in which the members of the community can quickly and easily find a suitable candidate for filling the required role.

1.3 Overview

This document will cover the four major design areas, which are namely data, architecture, interfaces, and components. Each of these design areas will provide an in-depth view at the organization and technical processes which govern the operations of the system. In order, the list of covered topics will be system overview, system architecture, data design, component design, human interface design, and the requirements matrix.

1.4 Reference Material

1.4.1 Python - <https://www.python.org/about/>

Python is a high-level, versatile, open source programming language often used for web development.

1.4.2 Django - <https://www.djangoproject.com/>

Django is a Python web framework used to streamline web development and tie frontend and backend together using a unified language.

1.5 Definitions and Acronyms

API - Application Program Interface
DFD - Data Flow Diagram
HTML - Hypertext Markup Language
SDS - Software Design Specification
SRS - Software Requirement Specification
UI - User Interface
UMW - University of Mary Washington
URL - Uniform Resource Locator

2. System Overview

The Outreach Contact System aims to replace the current process of a shared document within the community for sharing contact information and job information. In order to do this, the Outreach Contact System shall be a web application designed to be accessible via University of Mary Washington's Canvas website. This system will require both a comprehensive front end, as well as a backend database to store the information regarding accounts, profiles, and jobs. On the front end, an interface reminiscent of University of Mary Washington's overarching layout and design will be implemented with several forms of functionality.

Firstly, the user will be able to create either a student or employer account using their UMW Email address and a suitable password, which will be stored on the backend database. Once logged in, the user will be able to add information to their profile in the form of a name, email address, phone number, and typical jobs they may provide or search for. If the user is looking to post a job, they will be able to navigate to the respective screen, enter the information needed to create a job post, and confirm when finished. If the user is looking to search for a job, there will be a functionality to allow the user to search jobs via keywords, job types, and by the name of the desired member. If the user is looking to search for a specific profile, there will be a functionality to allow the user to search profiles via names, emails, and job types. Above these users will exist an Administrator account which can perform any function provided by the Outreach Contact System, but will focus on moderation of posts and profiles as well as ensuring posts and profiles are accurately removed from the system when needed. Most functions mentioned will require a unique screen per type of user.

Depending on the type of account the user is logged in to will determine the functionality of each screen.

3. System Architecture

3.1 Architectural Design

The Outreach group's application to simplify and streamline the process of finding tutors and sitters or jobs providing such things within the UMW community consists of five different architecture types. These five architecture types are Event-Driven, Client-Server, Repository, Web-Based, and Component-Based architecture. Determining which architecture types the application falls under depends on the overarching structure, how pages interact with each other, and the backend structures.

The first of five architecture types, Event-Driven, is an architecture that uses events to trigger other services of the application. Events are instances when a state of the data or application is changed or updated. Event-Driven architectures need to have three different actors in the process. These actors are the producers of an event, the actor which manipulates or sends the event somewhere else, and the actor that receives the final iteration of the event. The Outreach group's application represents this architecture type well because it uses a simple frontend design to allow the user to navigate and input information as well as click buttons, which trigger events on the backend and update other users secondhand.

The second of five architecture types, Client-Server, is an architecture in which there is a server host that distributes and controls the resources and services used by the user. This architecture type requires that the client be connected to the local network or internet. The Outreach group's application represents this architecture type well because the user will be able to access the server via the internet to perform any of the necessary tasks, such as creating their account, creating a job post, and searching job posts and profiles. Nearly every transaction on the application would require a client to server interchange in order to operate.

The third of five architecture types, Repository, is an architecture in which there is a layer of abstraction between the version of the application the user is operating and the version other users are operating. This architecture type requires that the client be connected to the local network or internet. The Outreach group's

application represents this architecture type somewhat well in the sense that a user can edit their profile as an instance, but another user could search their profile as another instance, both showing different information if there are any changes. Overall, there are few correlations between the Repository architecture type and the Outreach group's application, which means it is probably not the best example of such an architecture.

The fourth of five architecture types, Web-Based, is an architecture in which clients utilizing different systems and subsystems can communicate in different ways. This architecture type requires that the client be connected to the internet. The Outreach group's application represents this architecture type well because the user will be able to access the server via the internet to perform any of the necessary tasks, such as creating their account, creating a job post, and searching job posts and profiles. Every interaction the user has with the application will require the user to be connected to the internet for the changes to be made server-side.

The final of the five architecture types, Component-Based, is an architecture in which the design is broken down into individual functional components that represent methods and events that are well defined. The main use of this architecture type is providing a reusable set of code. The Outreach group's application does not represent this architecture type well as there will be very little, if any compartmentalization of functionality.

3.2 Decomposition Description

The following diagrams (Figure 3.2.1, Figure 3.2.2 and Figure 3.2.3) break down the overarching structures of the data flow throughout the program.

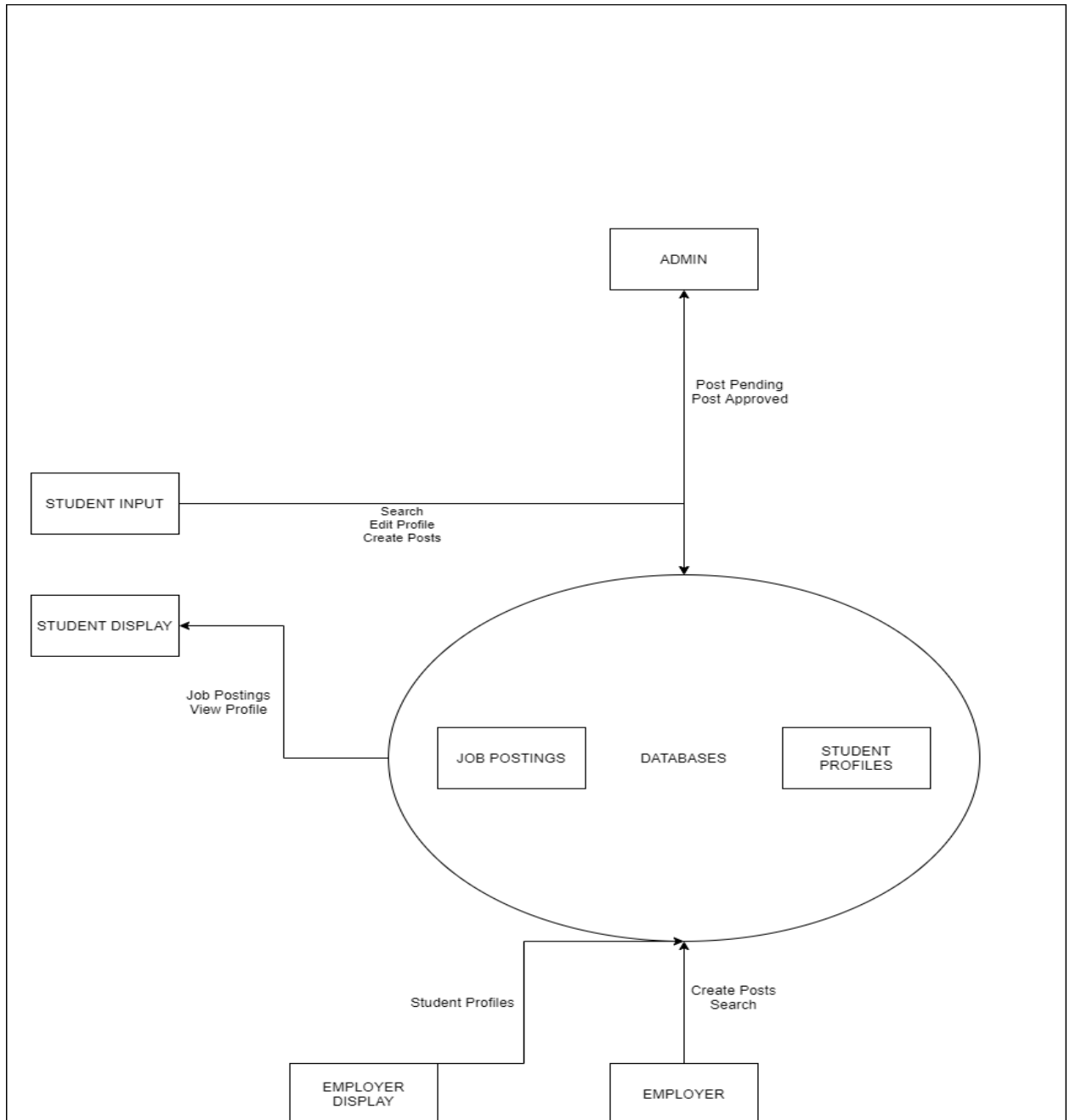


Figure 3.2.1: Context Diagram (Level 0 Data Flow Diagram) Rectangular nodes represent contributing entities and actors, while ovoidal nodes represent an entity that contains more than one entity, such as databases.

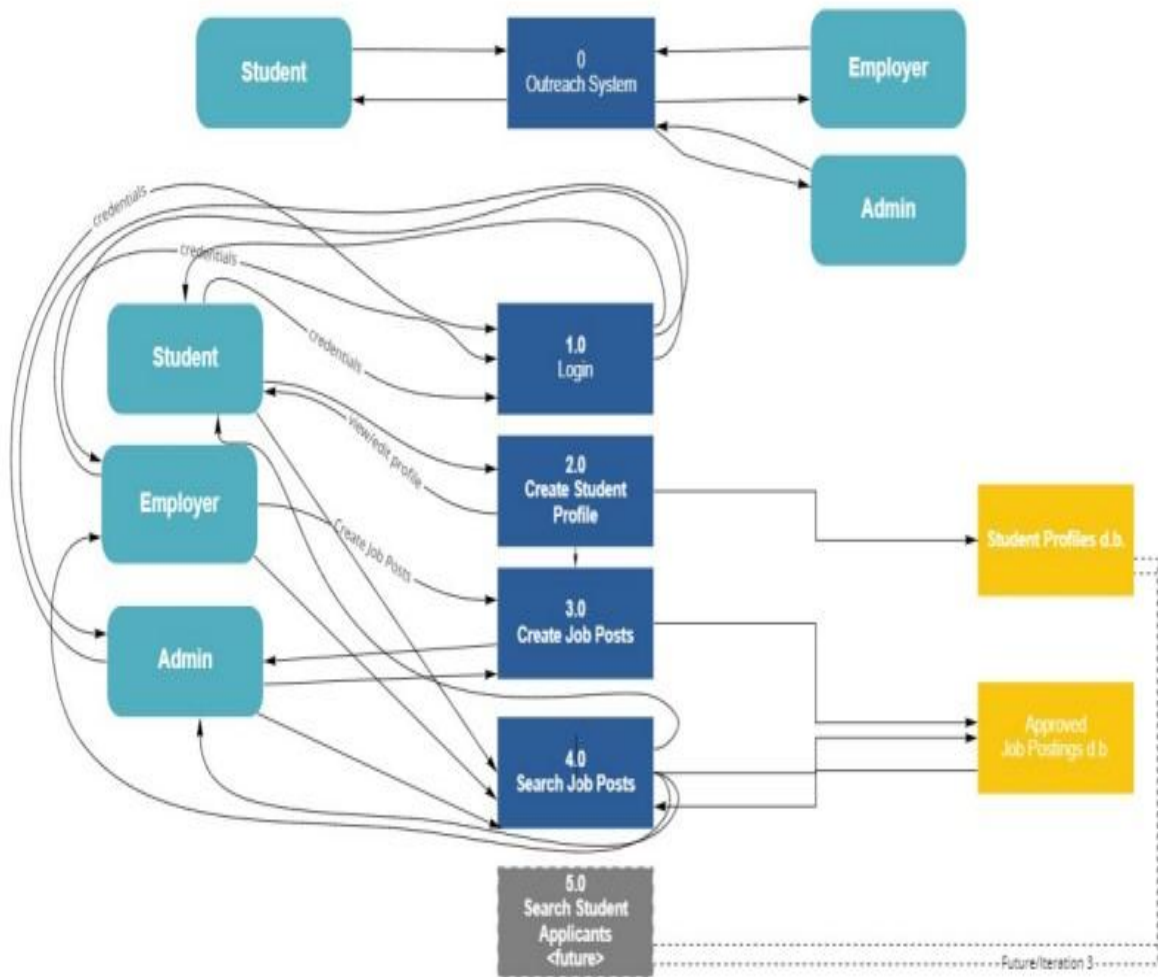


Figure 3.2.2: Context Diagram (Level 0 Data Flow Diagram) This diagram depicts the overall interactions and entities of the system addressed in Iteration 1 and Iteration 2. Rounded-corner nodes represent system users, dark blue, numbered rectangular nodes represent system entities and functions (Features for future iterations are included for context and are grey.) Data structures are represented by yellow nodes.

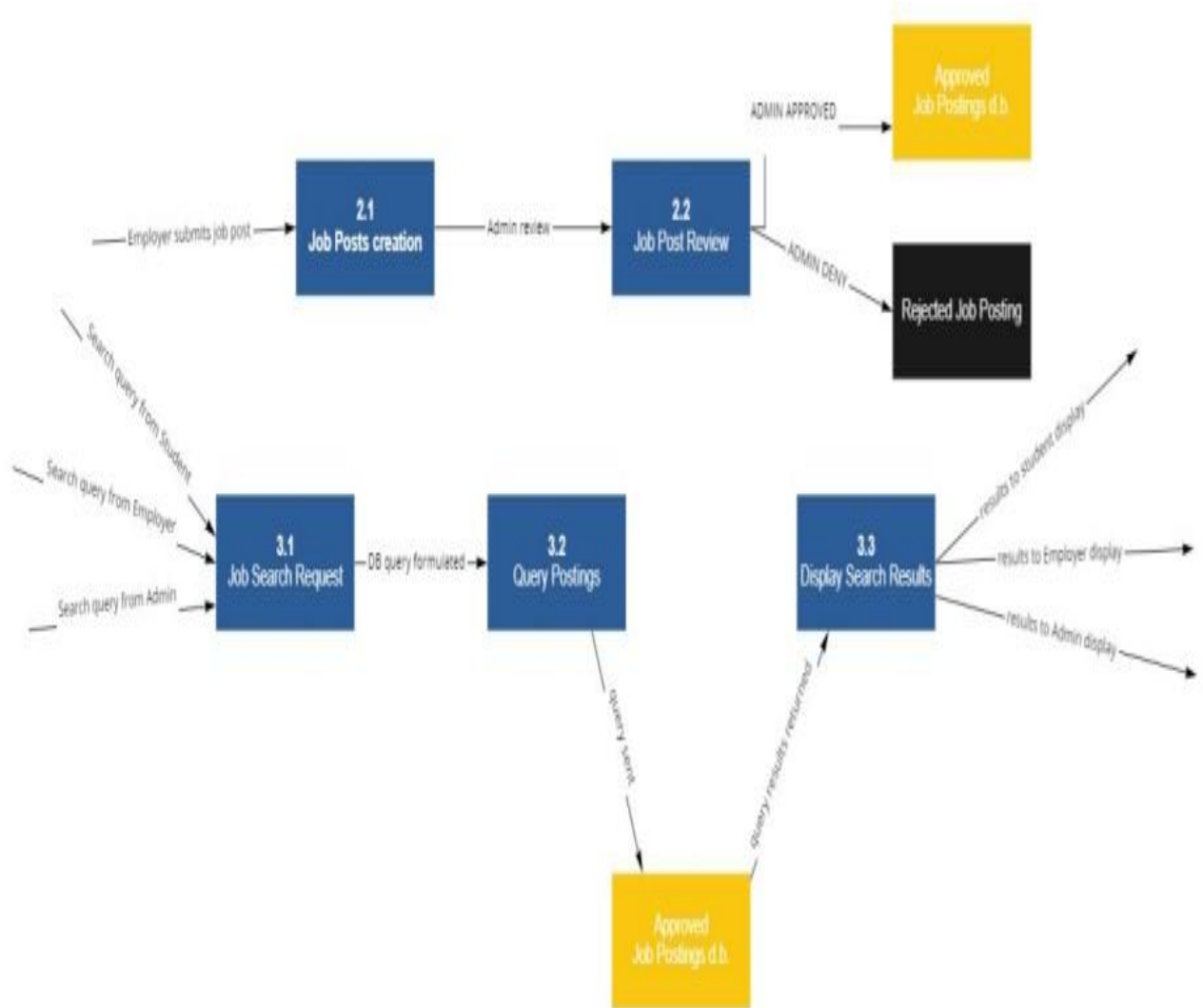


Figure 3.2.3: Context Diagram (Level 1 Data Flow Diagram) This diagram depicts a more detailed decomposition from the major processes shown in Figure 3.2.2. Blue rectangular nodes are numbered sequentially to correspond to the steps in Figure 3.2.2. and represent system entities and functions. Database structures are represented as yellow nodes.

3.3 Design Rationale

We have determined that this project will be best served using a mixture of architectures. The rationale for the selection of Event Driven as the primary architecture is because the user's experience of the site will be GUI windows containing buttons for selection of most interactions. The main buttons and

actions that will occur will include: post creation, post searching, editing of profiles, editing of posts, account creation, logon, log off and deletion of accounts.

For the search features there will be a search drop down selection menu that the user will interact with. The user will select search parameters and the action of selecting the 'submit' button will trigger the event. Employer users will create and edit posts and click a button to submit. The Admin user will approve or deny the job posts and a button event will be used to complete the transaction.

The next most heavily represented architecture type that is utilized is the Client-Server. Because we are utilizing a server host that oversees the distribution of resources and services the user is seeing, the Client-Server architecture is a strong fit. The Outreach group's application represents this architecture type well because the user will be able to access the server via the internet to perform any of the necessary tasks, such as creating their account, creating a job post, and searching job posts and profiles. Nearly every transaction on the application would require a client to server interchange in order to operate.

The Outreach group's application also represents Web Based architecture type well because the user will be able to access the server via the internet to perform any of the necessary tasks, such as creating their account, creating a job post, and searching job posts and profiles. Every interaction the user has with the application will require the user to be connected to the internet for the changes to be made server-side.

This website will also be using a Data Centric architecture type. The site will store user login information, as well as profile details and job postings.

The Outreach group's website utilized databases to store and retrieve all information so it also follows Component Based architecture. The site provides slightly different experiences for each of the 3 user types. Depending on who is logged on, they will have different permissions. For example, only employers are able to create job postings and only an Admin user can approve/deny postings. A student is able to click to express interest in a position. Student accounts are automatically deleted from the system once their date of graduation has passed, whereas an employer account does not have an expiration date.

4. Data Design

4.1 Data Description

This system uses a system of user accounts to allow users to access the system's services. It also needs to maintain a list of currently available jobs which are being offered to students, along with all the pertinent data for those jobs. To store this data, the system uses three different databases. One is to store the user account data. User account data stored includes:

- Email - the user's email address, necessarily a UMW email domain if the user is a Student
- Graduation date (if the user is a Student) - this date will be used to delete the accounts of students that have graduated.
- Name - the user's first and last name
- Password - the user account's password, stored as a hash value
- Type - the type of user account, in our case either a Student (looking to find jobs), an Employer (looking to have jobs fulfilled), or the single Admin account (in charge of reviewing posted jobs and curating accounts)

The two other databases store data about Jobs that are currently on offer in the system. There are two of these databases because one is to store offers that have been created by an Employer but not yet approved by the Admin, so they cannot yet be displayed on the main site. Job data stored in both databases is identical, and as follows:

- Job description - a detailed description of what the job entails
- Job location - where the job is (or whether it is in-person or virtual)
- Job salary - how much payment the job offers
- Job type - whether the job is tutoring, babysitting, or pet sitting

These data are all provided by the users of the system, either when the user creates their account or when an Employer user creates a job offer. Job data is displayed on the main site when Student users choose to view available job listings.

4.2 Data Dictionary

Field Name	Data Type	Table(s)	Description	Example
Name	Varchar	User Account	The users first and last	"John Doe"

			name	
Type	Varchar	User Account	The type of user account	"Student", "Employer", "Admin"
Password	Varchar	User Account	The users password	"aBcD39!"
Email	Varchar	User Account	The users email	"jdoe31@mail.umw.edu"
Graduation Date	Date	User Account	The users graduation date(If student)	05-01-1999
Job Location	Varchar	Pending Posts & Posts	A description of the Job location	"At my home in Fredericksburg"
Job Salary	Integer	Pending Posts & Posts	The pay rate for the job,dollars per hour	12
Job Type	Varchar	Pending Posts & Posts	Defines the kind of job for this post	"Babysitting", "Tutoring", "Pet Sitting"
Job Description	Varchar	Pending Posts & Posts	Brief description of what the job entails	"Tutoring from 7-8 PM on MW in algebra"

5. Component Design

This section will show pseudocode describing the function of each component.

5.1 Create Account

```

Display the home screen
User selects the login button

User selects the create account button

User enters email, password, and account type

Call function
create_employer_account()/create_student_account()/create_admin_account()

```

```

Function create_account(string email, string password)

```

```

    userEmail = email
    password = password
    for every account in database
        if (account info includes userEmail and password)
            return "account already exists"
        else if (account info does not include
            userEmail and password)
            enter account information into database
            return "Success"
end function

```

5.2 Login

Display the Home screen

User selects the "student login" button, the "employer login" button or the "admin login" button

Display the "student login", "employer login" or "admin login" screen respectively

User enters email and password into text intake boxes for email and password

User enters the "submit login" button

Call function

login_employer()/login_student()/login_admin()

```

Function login_employer(string email, string password) {
    String em = email
    String pwSubmitted = password
    String pwRecord = ""

    Connect to database containing 3 tables:
        employer_account,

```

```

        Student_account,
        admin_account
    Query employer_account
    table for EMAIL = em,
    display value for password

    if (no record located) {
        Return "no user account found,
            please retry or create an account")
    }
    if (record located) {
        pwRecord = table value for password
        //check to see if password matches
        password of record
        If (pwRecord ==pwSubmitted) {
            //login credentials match, permit login
            Permit login, create active work session
            Return message "logon successful"
        }
        else if ( pwRecor != pwSubmitted) {
            //passwords NOT a match, login denied
            Login denied
            Return message "username and passwords
                did NOT match, please try again"
        }
    }
}

end function

Function login_student(string email, string password) {
    String em = email

```

```

String pwSubmitted = password
String pwRecord = ""
Connect to database containing 3 tables:
    Employer_account,
    student_account,
    Admin_account

Query employer_account table for EMAIL = em,
display value for password

If no record located
    Return "no user account found, please retry
    or create an account")

if record located {
    pwRecord = table value for password
    //check to see if password matches password of
    //record
    //login credentials match, permit login
    if(pwRecord ==pwSubmitted) {
        Permit login, create active work session
        Return message "logon successful"
    }
    //passwords NOT a match, login denied
    else if ( pwRecord != pwSubmitted) {
        Login denied
        Return message "username and passwords did
        NOT match, please try again"
    }
}

end function

```



```

Function login_admin(string email, string password) {
    String em = email
    String pwSubmitted = password
    String pwRecord = ""

    Connect to database containing 3 tables:
    employer_account, student_account, admin_account

    Query employer_account table for EMAIL = em, display
    value for password

    if no record located
        Return "no user account found, please retry or
        create an account")
    if record located {
        pwRecord = table value for password
        //check to see if password matches password of
        //record
        //login credentials match, permit login
        if (pwRecord ==pwSubmitted) {
            Permit login, create active work session
            Return message "logon successful"
        }
        //passwords NOT a match, login denied
        else if ( pwRecord != pwSubmitted) {
            Login denied
            Return message "username and passwords did
            NOT match, please try again"
        }
    }
}

end function

```

5.3 Logout

User selects the "logout" button

Call function

logout_employer()/logout_student()/logout_admin()
respectively

```
Function logout_employer() {  
    Close any connections to database  
    Terminate active session  
    Return message "Logout successful"  
}  
end function
```

```
Function logout_student() {  
    Close any connections to database  
    Terminate active session  
    Return message "Logout successful"  
}  
end function
```

```
Function logout_admin() {  
    Close any connections to database  
    Terminate active session  
    Return message "Logout successful"  
}  
end function
```

5.4 Admin Approve Post

Display the home screen

Admin logs in

Admin views posts for review

Admin selects the "Approve" button

Call function approve_Post()

Function approve_Post()

 Query database to create new table entry

 New table entry is populated with data from currently
 viewed job

End function

5.5 Delete a Posted Job

Display the home screen

User logs in

User views their posts

User views the post they created

User selects the "Delete Post" button

Call function delete_post()

Function delete_post()

 Query database for the post

 If(post exists)

 Remove post from database

 Return "success"

 Else if (post does not exist)

 Return "Post deletion failed"

End function

5.6 View Job Post

display a view posts option
call function viewPosts() when user selects the view posts option
call function viewPost() when user selects a post

```
Function viewPosts()  
    declare variable called posts  
    query database with user ID and store results in posts  
    call function displayPosts(posts)  
End function
```

```
Function displayPosts(posts)  
    load new web page with the following info:  
    for each post in posts  
        display post title  
        display job type  
        display preview of job description  
        display job location  
    endfor  
    redirect to the new web page  
End function
```

```
Function viewPost(postID)  
    declare variable called post  
    query database with postID and assign result to post  
    call function displayPost(post)  
End function
```

```
Function displayPost(post)  
    load new web page with the following info:  
    display post title  
    display job type  
    display full job description  
    display job location  
    display employer name  
    redirect to the new web page  
End function
```

5.7 Search Available Job Posts

Declare a list of Jobs called jobList

```

Declare a double variable called jobDist

Declare a double variable called jobPay

Declare a String variable called jobType

initialize each variable with data from User
if user does not provide data, variable remains null

for Job in jobDatabase
    //if a job is within distance it gets added to our
    //working list.
    //if we're not filtering by distance, this adds all
    //jobs to our working list
    if job.Distance <= jobDist OR jobDist = null
        add Job to jobList

for Job in jobList
    //if we're filtering by salary and this job doesn't
    //meet salary requirements, remove it from working
    //list
    if job.Salary <= jobPay AND jobPay != null
        remove Job from jobList

    //if we're filtering by job type and this job doesn't
    //meet requirements, remove it
    if job.Type != jobType AND jobType != null
        remove Job from jobList

for Job in jobList
    display Job data to user

```

5.8 Edit Job Post

Follow steps for 5.6 "View Post"
User selects "Edit Post" button

```

onClick(editPost(postID))

function editPost(postID) {
    postFields = queryDB('/posts?id=postID&option=fields')
    if (postFields != errorCode) {
        display(postFields,"edit")
    } else {

```

```

        display('Edits Unsuccessful')
    }
}
end function

User enters changes to postFields
User hits submit

onClick(submitEdits(postId,postFields))

function submitEdits(postID,postFields) {
    res = postDB({
        id = postID
        fields = postFields
        type = "edit"
    })
    if (res != errorCode) {
        display(success)
    } else {
        display('Edits Unsuccessful')
    }
}
end function

```

5.9 Delete Job Post

User selects delete post button

```

onClick(deletePost(postID))
function deletePost(postID) {
    display("confirm or deny")
    //User selects confirm
    onclick(confirmDelete(postId))
    //User selects deny
    onClick(denyDelete())
}
End function

User selected confirm delete
function confirmDelete(postFields) {
    res = delDB({
        id = postID
    })
}

```

```

        if (res != errorCode) {
            display(success)
        } else {
            display('delete unsuccessful')
        }
    }
End function

User selected deny delete
function denyDelete() {
    display(postId)
}
End function

```

5.10 Admin Delete Account

display a delete account option
 call function requestDeleteAccount() when admin selects the delete account option
 call function confirmAccountDeletion() when admin selects the confirm option
 call function cancelAccountDeletion() when admin selects the cancel option

```

Function requestDeleteAccount(userID)
    call function displayConfirmationWindow(userID)
End function

```

```

Function displayConfirmationWindow(userID)
    load new web page with the following info:
    message asking if the admin wishes to permanently
    delete the account with userID
    cancel option
    confirm option
    redirect to the new web page
End function

```

```

Function confirmAccountDeletion(userID)
    return deleteAccount(userID)
End function

```

```

Function deleteAccount(userID)
    delete entry in database whose id matches userID
    return success/failure message
End function

```

```
Function cancelAccountDeletion(userID)
    declare variable called account
    query database with userID and assign result to account
    load user profile page with info in account
    redirect to the user profile page
End function
```

6. Human Interface Design

6.1 Overview of User Interface

The user interface consists of 3 types of pages:

- General: pages that all users can access. Includes the Main, Login, Create Account, and User Home pages.
- User-specific: pages that are accessible based on whether the user has a student, employer, or admin account. For example, students can access the Search Job Postings Page, while employers can access the Create Post Page.
- Transition: pages that connect other pages. Includes various confirmation and error screens.

The Main Page contains a navigation menu from which users can access the Login and Create Account pages by clicking the “Login” and “Create Account” buttons, respectively.

Each user’s home page reflects their account privileges. There are buttons which link to pages for each major function of the account. Students are able to search for job posts and browse jobs they’ve applied to. Employers are able to create job posts and view them. Admins can manage posts and accounts. All users can quickly access their home pages, view their profiles, and log out.

The Search Job Postings page contains a search bar that is accompanied by a drop-down menu of search filters and a “Search” button to submit the search. When a student submits their search, a list of matching posts is displayed below the search bar. The student can then click on a post to view it on its own page in more detail.

The Create Post Page displays a simple form for employers to fill out and submit. The form includes fields for a title, job type, and a brief description. Once an

employer submits the form, they are redirected to a page that informs them their post has been submitted to the admin for review.

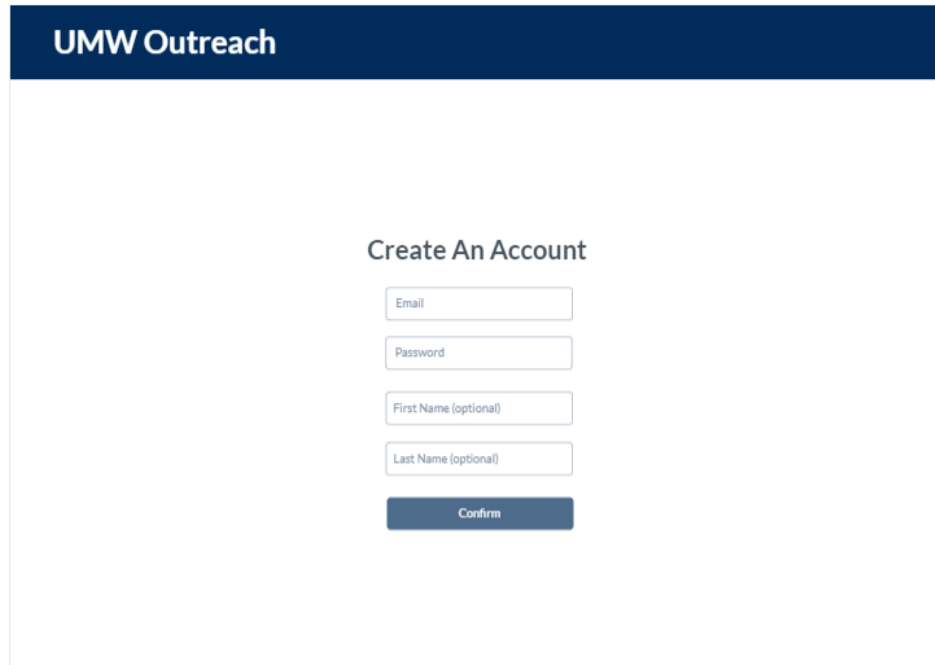
All pages use the official UMW colors.

The following section has multiple screenshots of a basic version of this interface.

6.2 Screen Images



Figure 6.2.1: Main Page



UMW Outreach

Create An Account

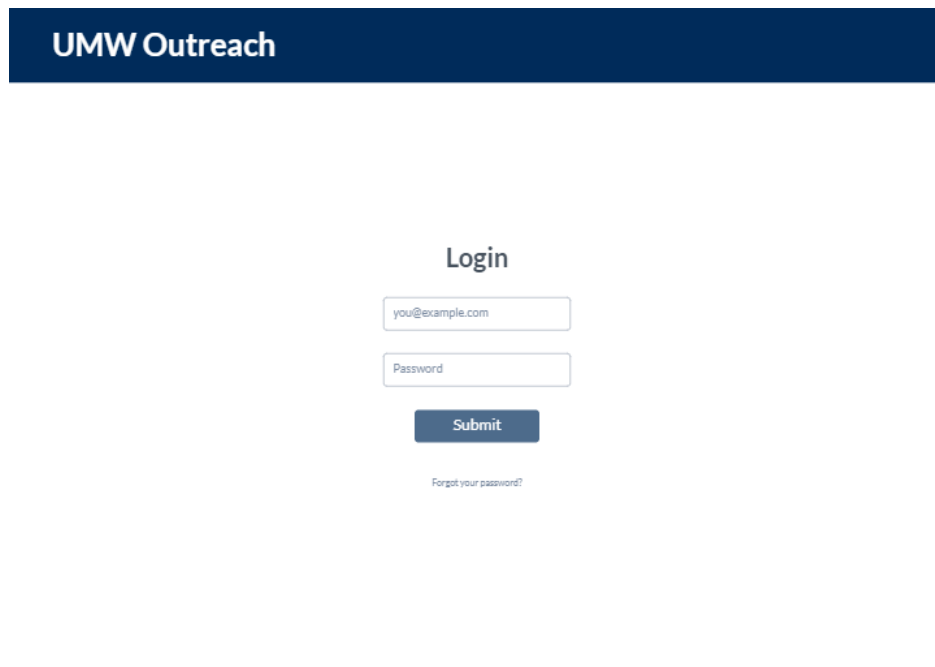
Email

Password

First Name (optional)

Last Name (optional)

Figure 6.2.2: Create Account Page



UMW Outreach

Login

Password

[Forgot your password?](#)

Figure 6.2.3: Login Page

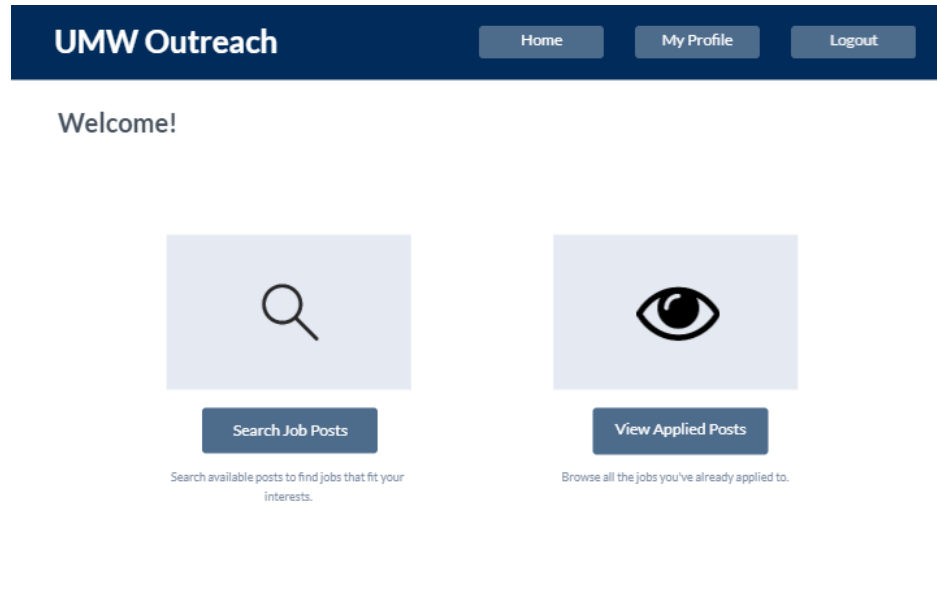


Figure 6.2.4: Student Home Page

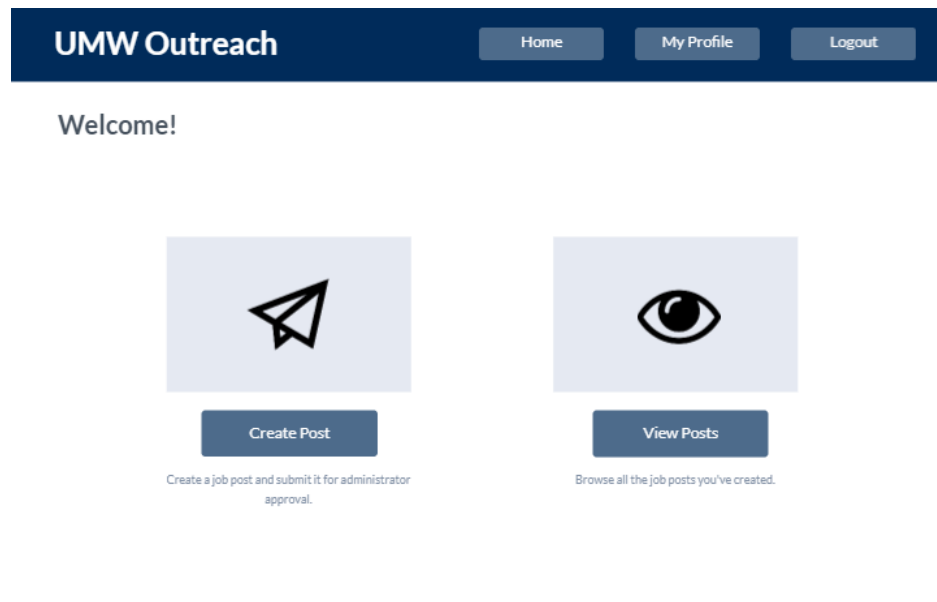


Figure 6.2.5: Employer Home Page

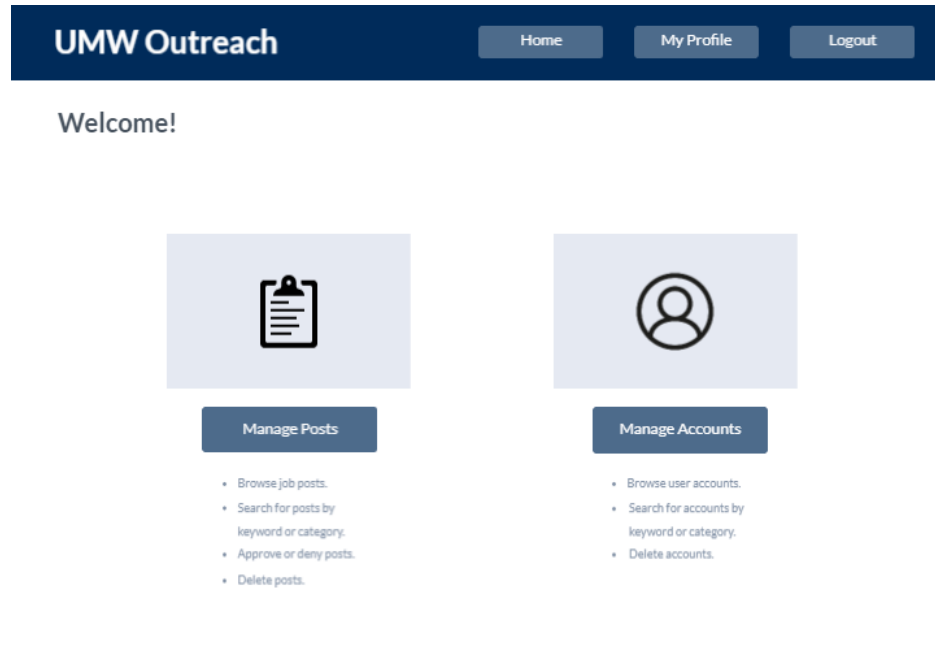


Figure 6.2.6: Admin Home Page

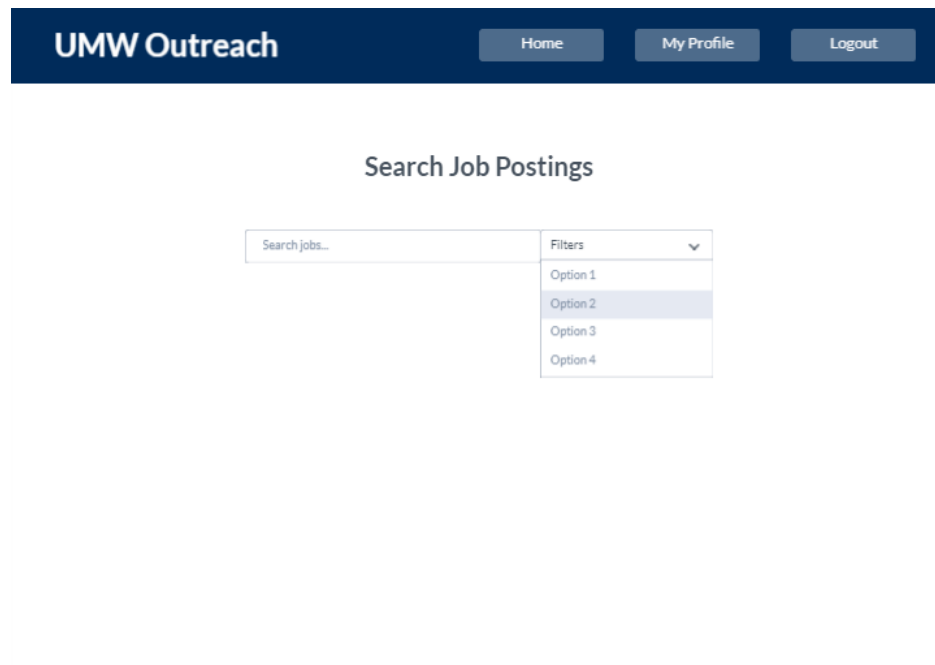


Figure 6.2.7: Search Job Postings Page

UMW Outreach

Home

My Profile

Logout

Create Post

Title

Post title

Job Type

Job type

Description

Job description

Submit

Figure 6.2.8: Create Post Page

UMW Outreach

Home

My Profile

Logout

Edit Post

Title

New post title

Job Type

New job type

Description

New job description

Save Changes

Figure 6.2.9 Edit Post Page

Delete this post?

Cancel

Confirm

Figure 6.2.10: Delete Post Page

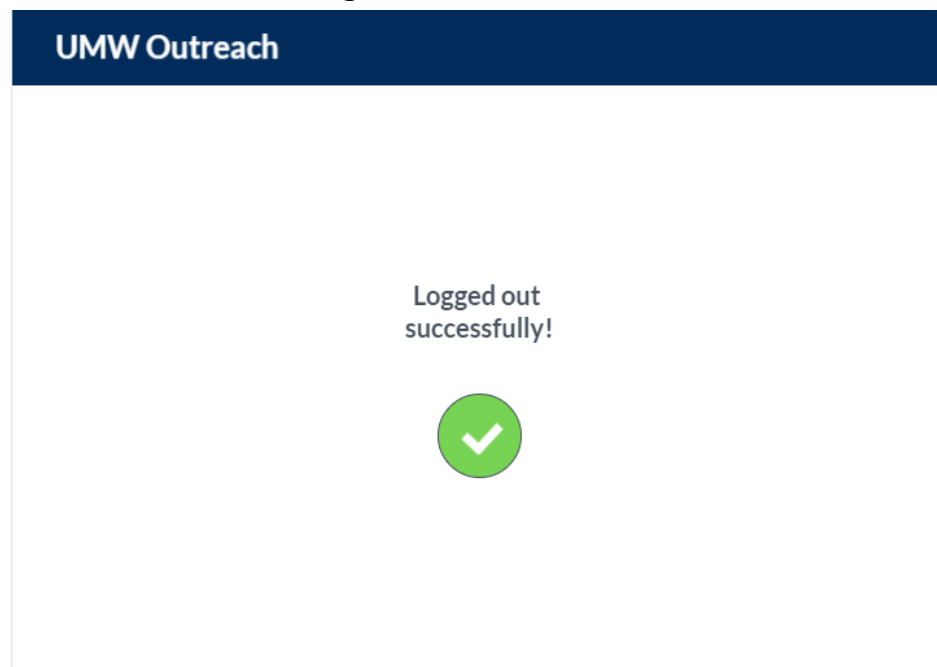


Figure 6.2.11: Logout

6.3 Screen Objects and Actions

6.3.1 Create Account

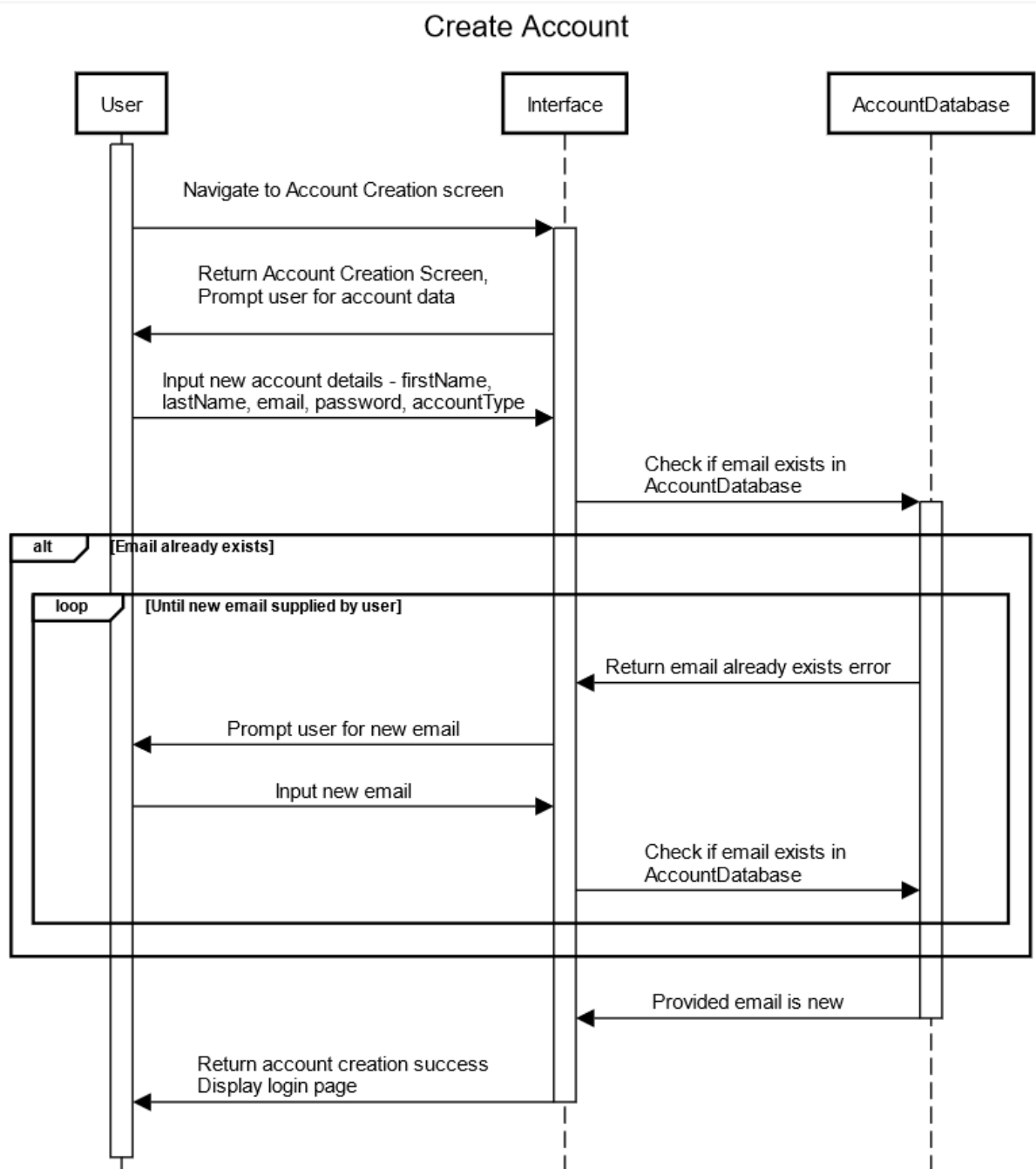


Figure 6.3.1: Create Account Sequence Diagram

Figure 6.3.1 represents what happens when a user creates an account. A user can only create an account from the website's home page. First, the system

displays the home page. The user then clicks the “Login” button, which requests the “Login” page from the system. The system loads and displays the login screen. The user clicks the “Create Account” button to go to the “Create Account” page. After entering their account information, the user clicks the “Confirm” button to submit it. Next, the system validates the entered email and password using the account database. If both fields are valid, the system displays a success screen and redirects the user to the login screen. Otherwise, the system displays a failure screen.

6.3.2 Login

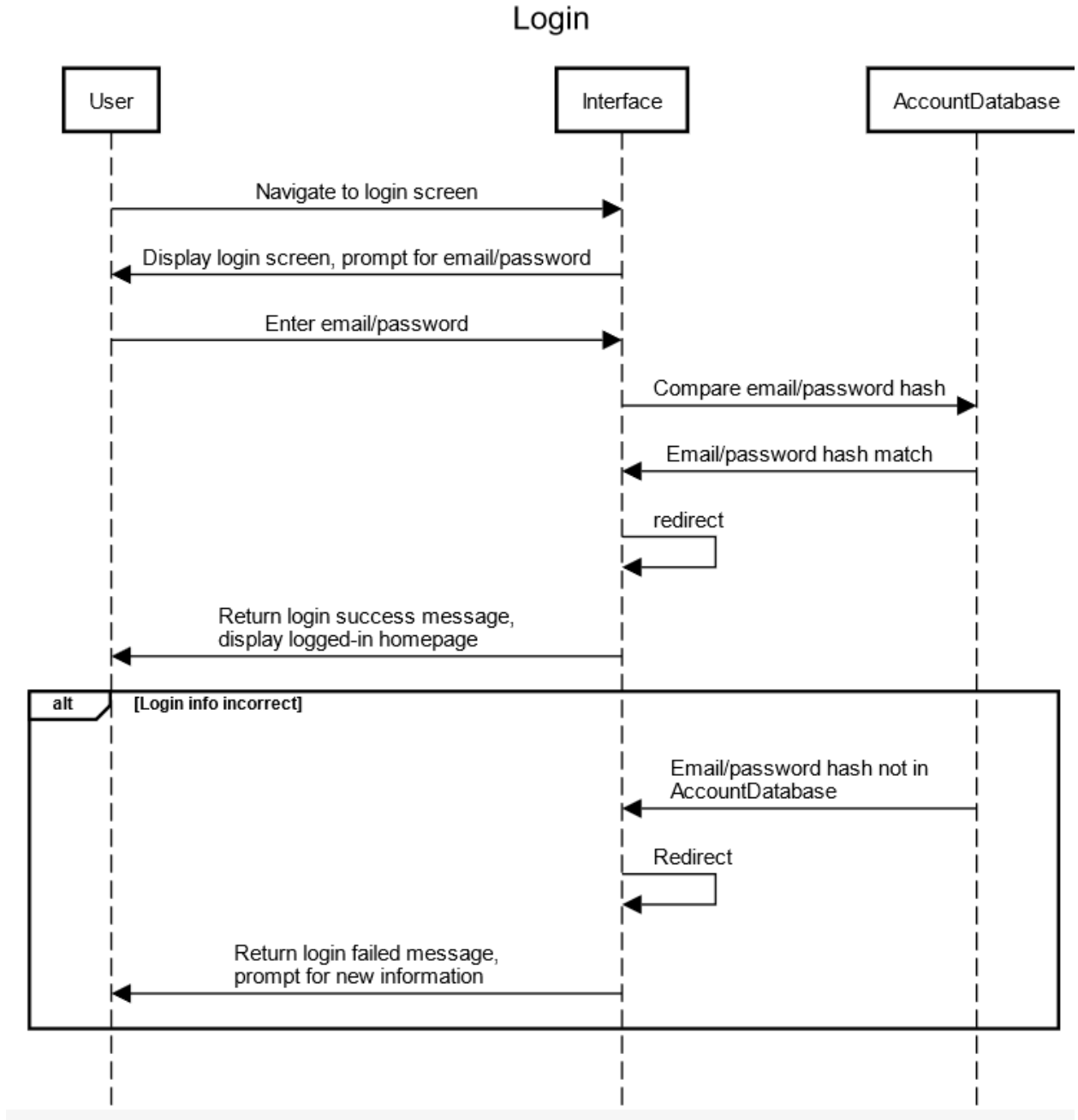


Figure 6.3.2: Login Sequence Diagram

Figure 6.3.2 represents what happens when a user logs in to the website. First, the user visits the “Login” page. After entering their username and password, the user clicks the “Login” button. The system validates the entered information by hashing the username/password and comparing it to the hashed username/password in the accounts database. If the two strings match, the

system informs the user that they have successfully logged in, and redirects them to their home page. Otherwise, the system informs the user that the login failed, and reloads the “Login” page, clearing the password field.

6.3.3 Logout

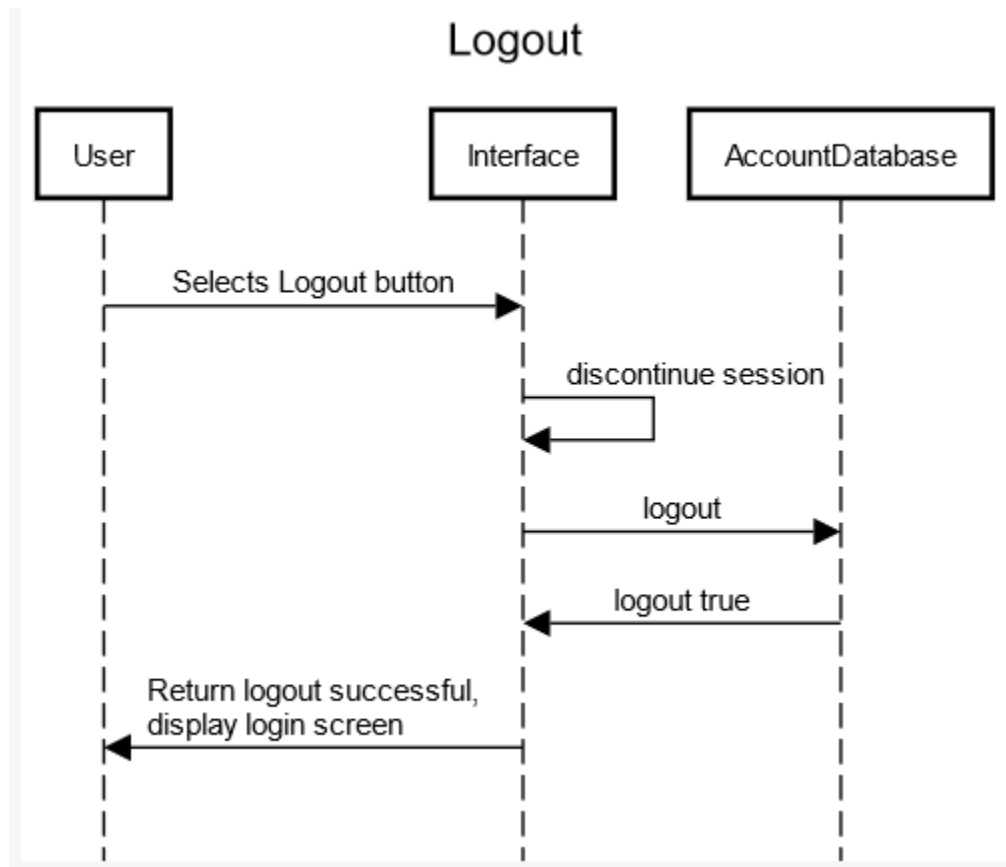


Figure 6.3.3: Logout Sequence Diagram

Figure 6.3.3 represents what happens when a user logs out of the website. First, the user clicks the “Logout” button. The system then updates the accounts database to indicate that the user has logged out. Finally, the system informs the user that they have successfully logged out.

6.3.4 Create Post

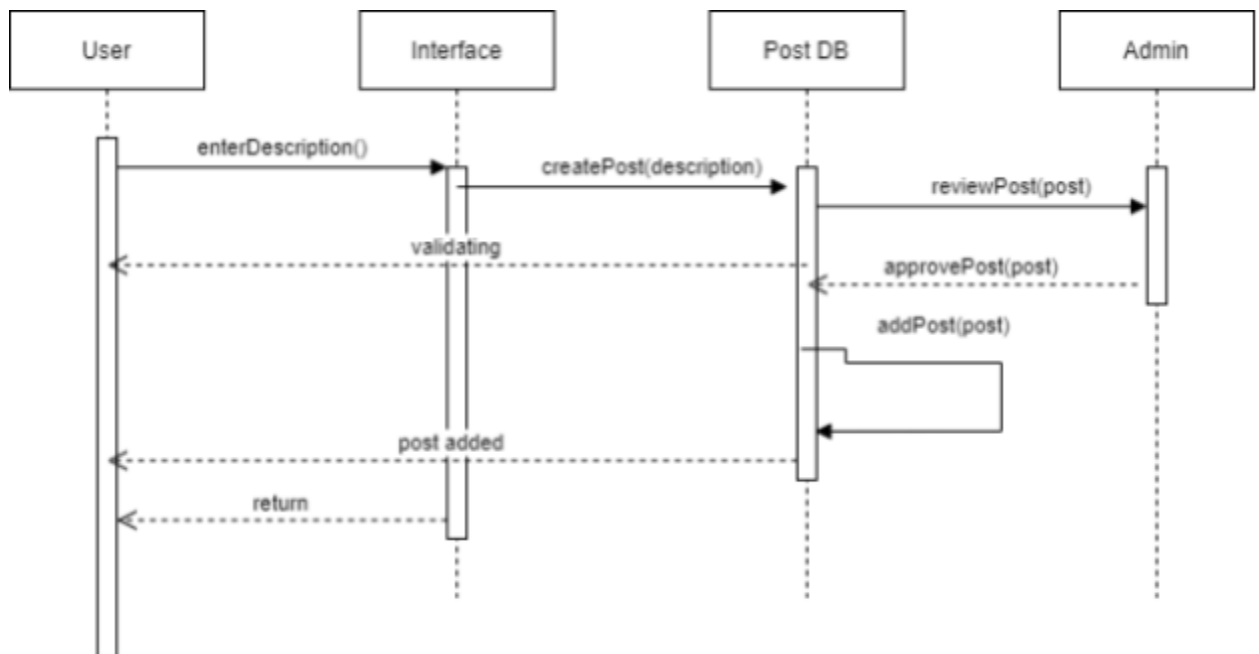


Figure 6.3.4: Create Post Sequence Diagram

Figure 6.3.4 represents what happens when an employer creates a job post. First, the employer enters information about the job, which includes the job type along with a brief description. Once the employer clicks the “Submit” button, their post is submitted to the admin for approval. The system notifies the employer that their post will be reviewed. After the admin reviews the post and approves it (**Figure 6.3.9**), the post is added to the post database. The employer is then notified that their post has been accepted.

6.3.5 Edit Post

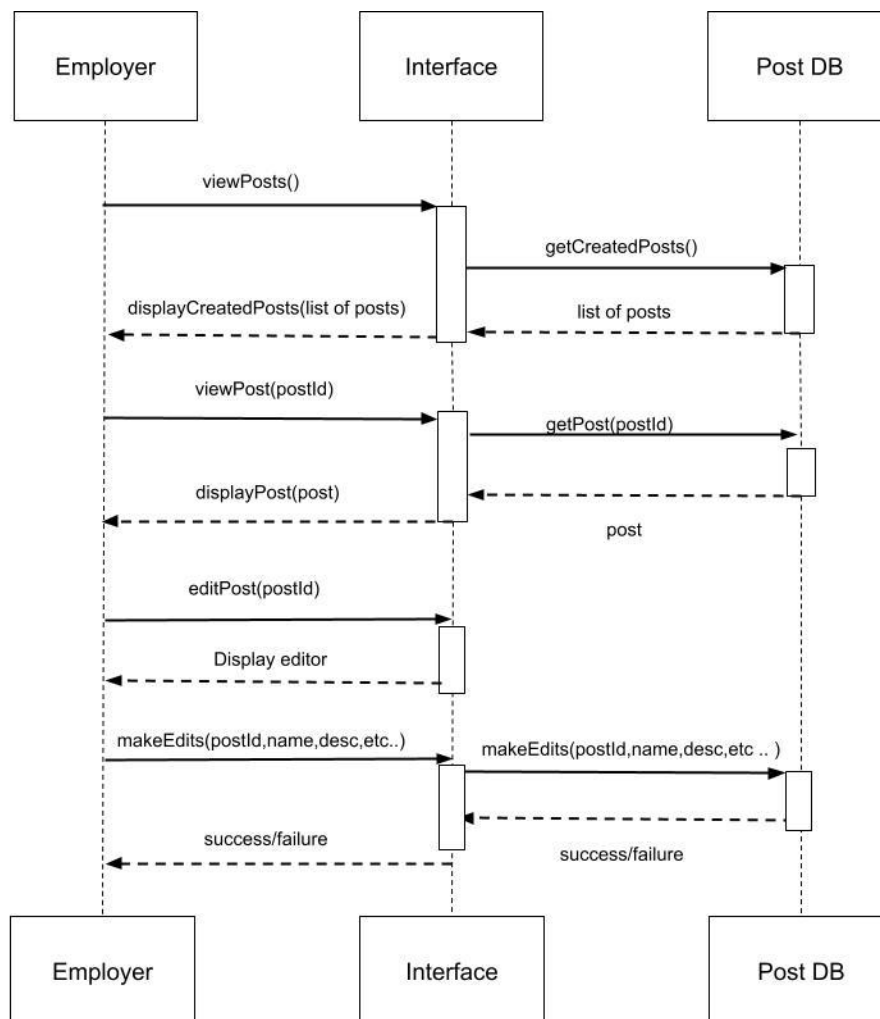


Figure 6.3.5: Edit Post Sequence Diagram

Figure 6.3.5 represents what happens when an employer decides to edit a job post. First, the employer requests to view all of their posts. Then, the employer may select an individual post from this list. Finally, choosing the “Edit” button will allow the user to make edits to the post. The system notifies the employer that their edits were successful/failed.

6.3.6 Delete Post

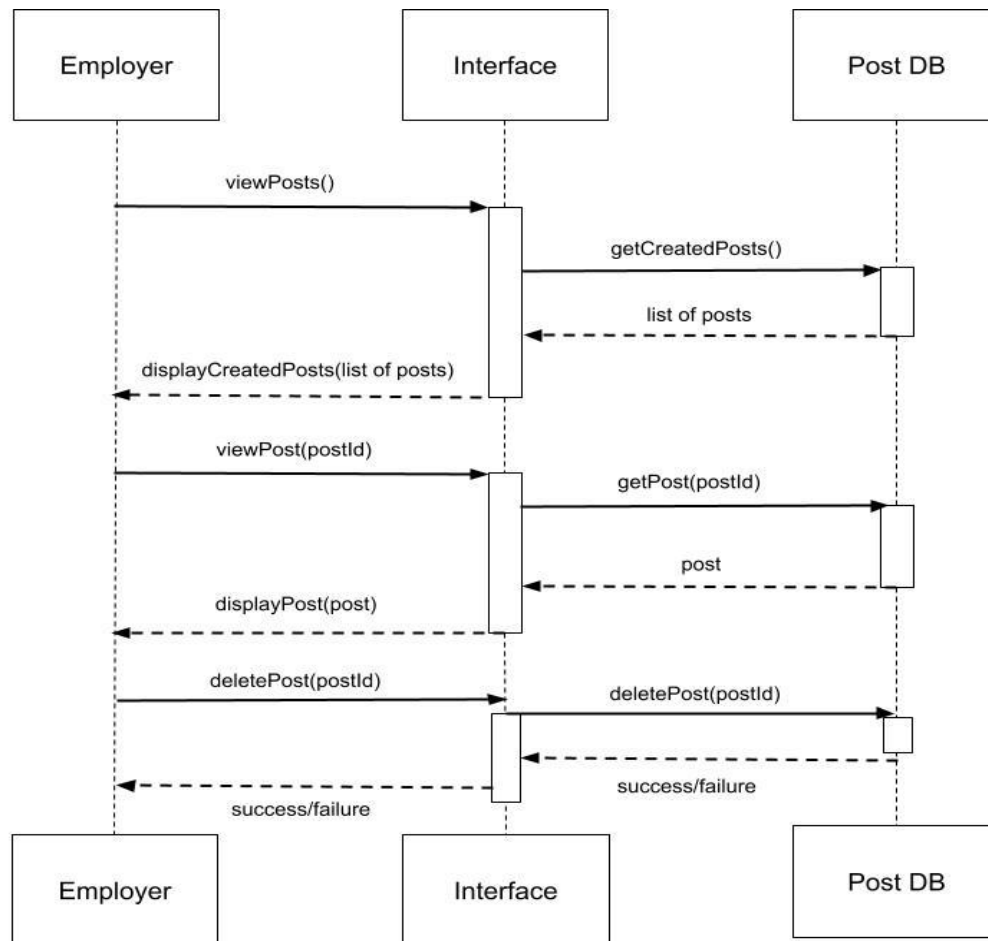


Figure 6.3.6: Delete Post Sequence Diagram

Figure 6.3.6 displays the process for deleting a job post. To begin, the employer requests to view all of their posts. Then, the employer may select an individual post from this list. On this screen, they may then select the “Delete Post” button. When this occurs, the user will be prompted for confirmation. If they select “Yes”, the post will be deleted and the system will show the result from the database. If they select “No”, they will be returned to the previous screen.

6.3.7 Search Job Listings

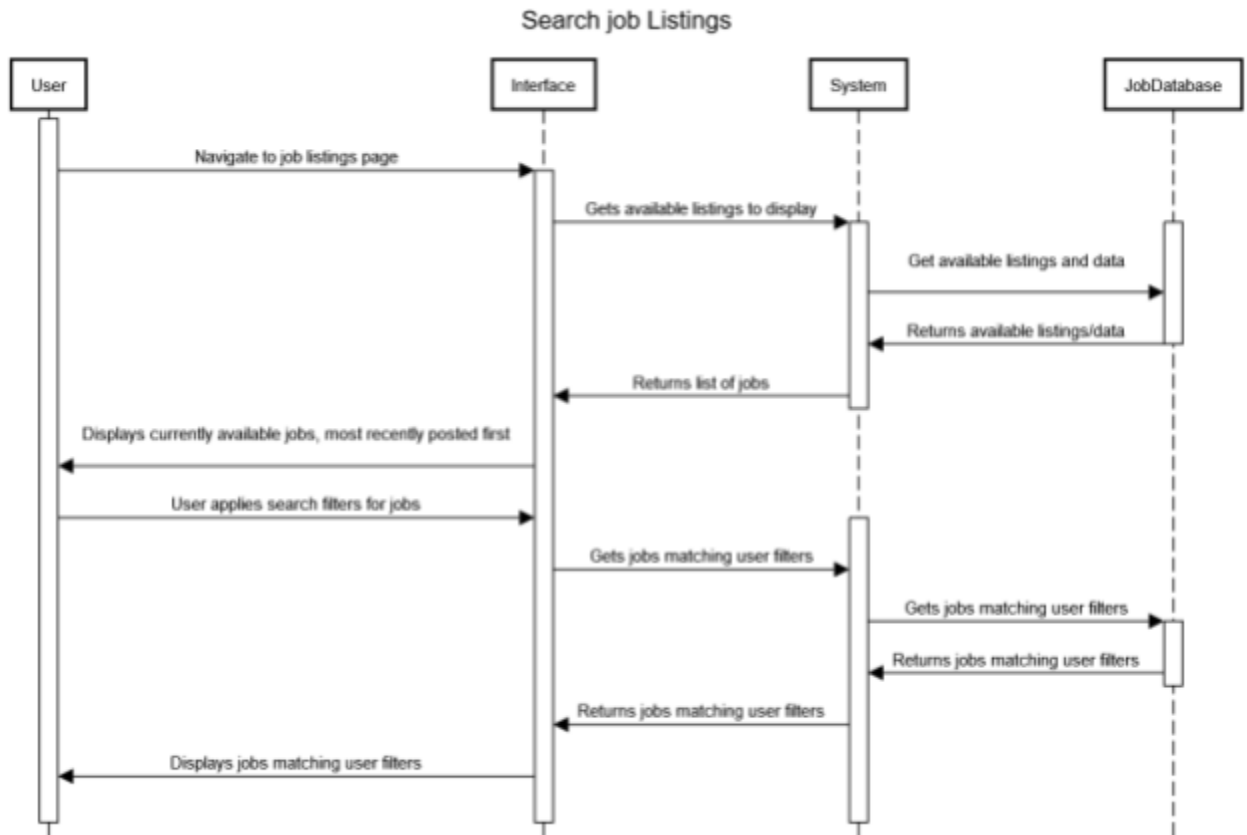


Figure 6.3.7: Search Job Listings Sequence Diagram

Figure 6.3.7 represents what happens when a student searches for available job listings on the website. First, the student navigates to the “Job Listings” page. The system responds by getting a list of the most recently posted jobs from the job database and displaying it to the student. The student can then apply filters to their search. If they do, the system will return any jobs matching those filters. Examples of possible search filters include date, location range, or job type.

6.3.8 View Post

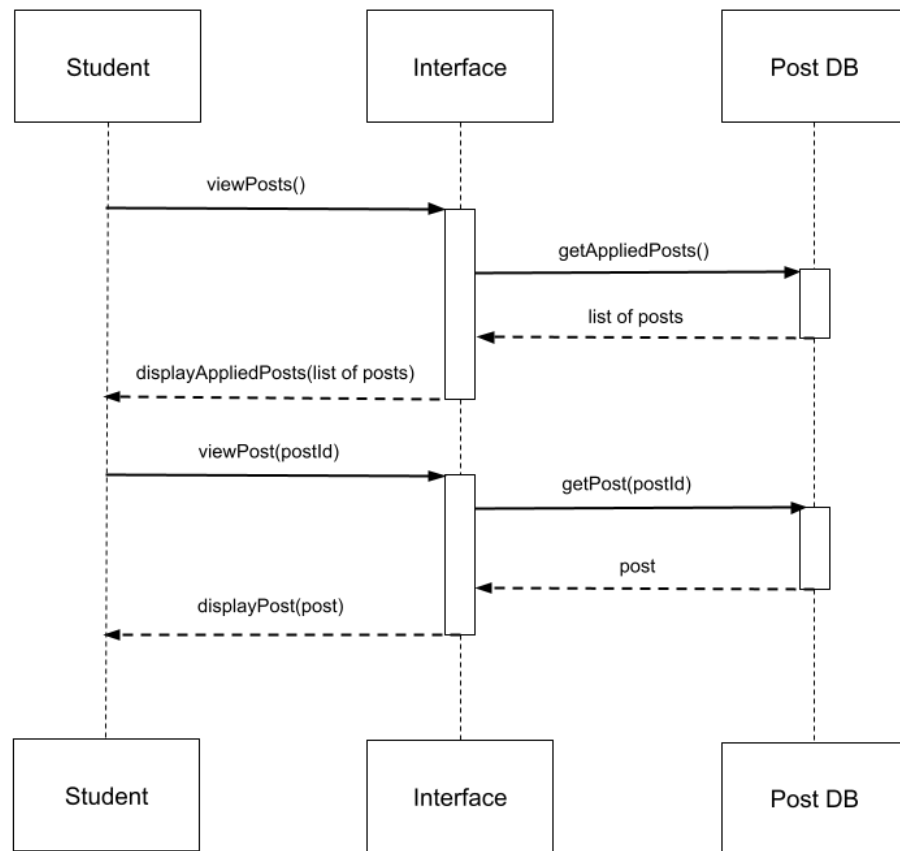


Figure 6.3.8.1: Student View Post Sequence Diagram

Figure 6.3.8.1 represents what happens when a student views all the jobs they've applied to. The student accesses this option from their home page. First, the student clicks the "View Applied Posts" button. The system then retrieves the requested list of posts from the post database and displays it. Next, the student clicks on a post to view it. The system finds and displays the appropriate information for the post.

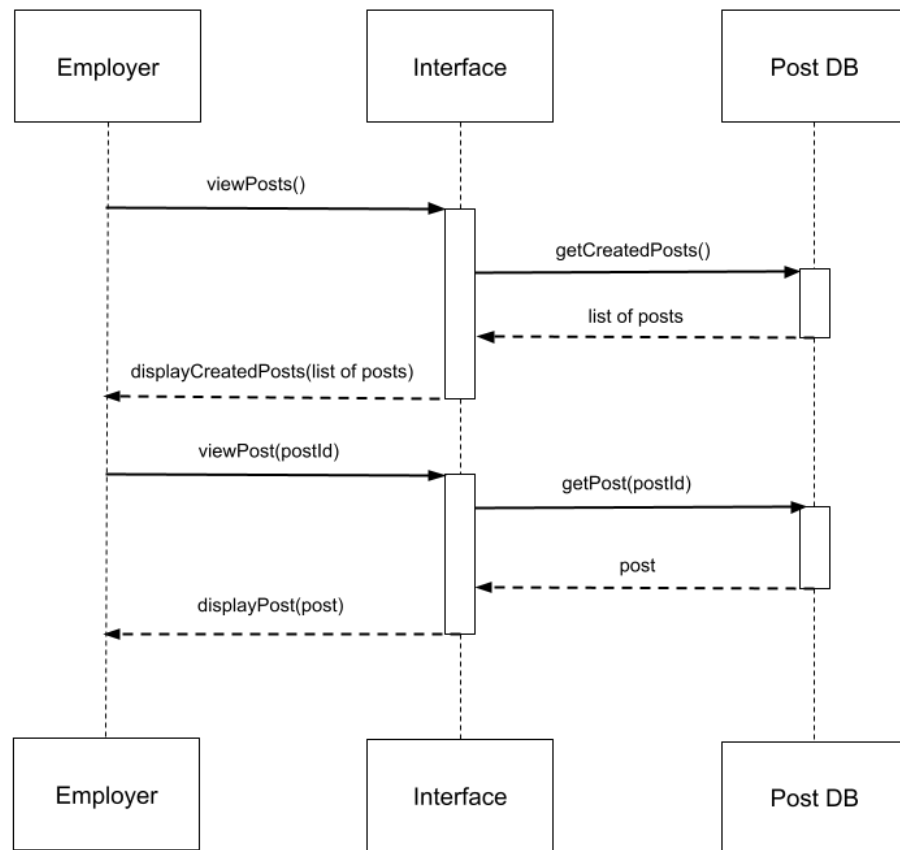


Figure 6.3.8.2: Employer View Post Sequence Diagram

Figure 6.3.8.2 represents what happens when an employer views all the job posts they've created. The employer accesses this option from their home page. First, the employer clicks the "View Posts" button. The system then retrieves the requested list of posts from the post database and displays it. Next, the employer clicks on a post to view it. The system finds and displays the appropriate information for the post.

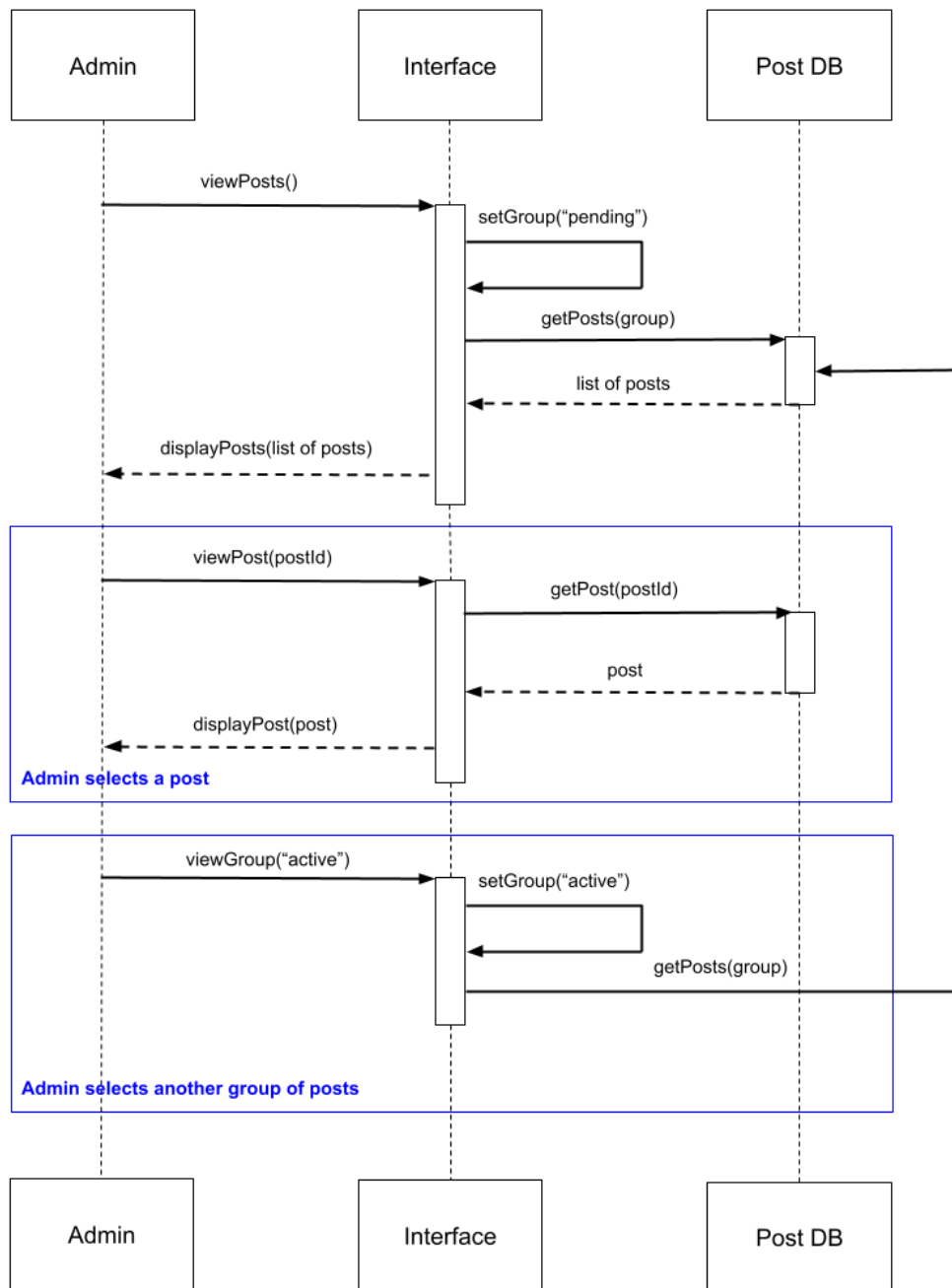


Figure 6.3.8.3: Admin View Post Sequence Diagram

Figure 6.3.8.3 represents what happens when an admin wants to manage the posts in the system. The admin accesses this option from their home page. First, the admin clicks the “Manage Posts” button. The system searches for all posts

that fall into a specific group (Pending, Active, All, etc.) and displays only that group. By default, the system shows posts pending approval. The admin can now either: (a) select the option to view another group of posts. The system retrieves the requested list of posts from the post database and displays it. or (b) click on a post to view it. The system finds and displays the appropriate information for the post.

6.3.9 Review Post

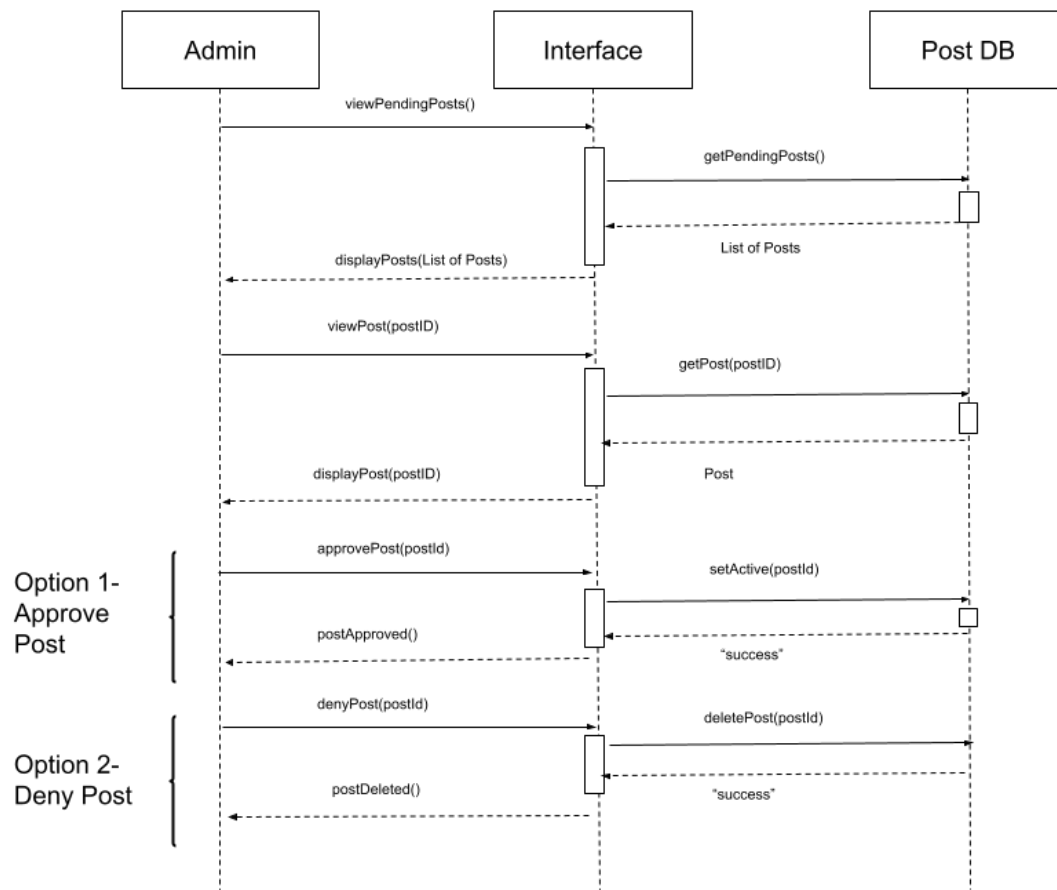


Figure 6.3.9: Review Post Sequence Diagram

Figure 6.3.9 represents what happens when an admin reviews a job post, meaning they either approve or deny it. First, the admin selects the option to view all pending posts, and clicks on a post to view it (**Figure 6.3.8.3**). The admin can now either: (1) click the “Approve” button. The system adds the post to the post database and sets its status to “active”. The post’s author is notified that their post has been accepted. or (2) click the “Deny” button. The system removes the

post from the post database. The post's author is notified that their post has been rejected.

6.3.10 Admin Delete Account

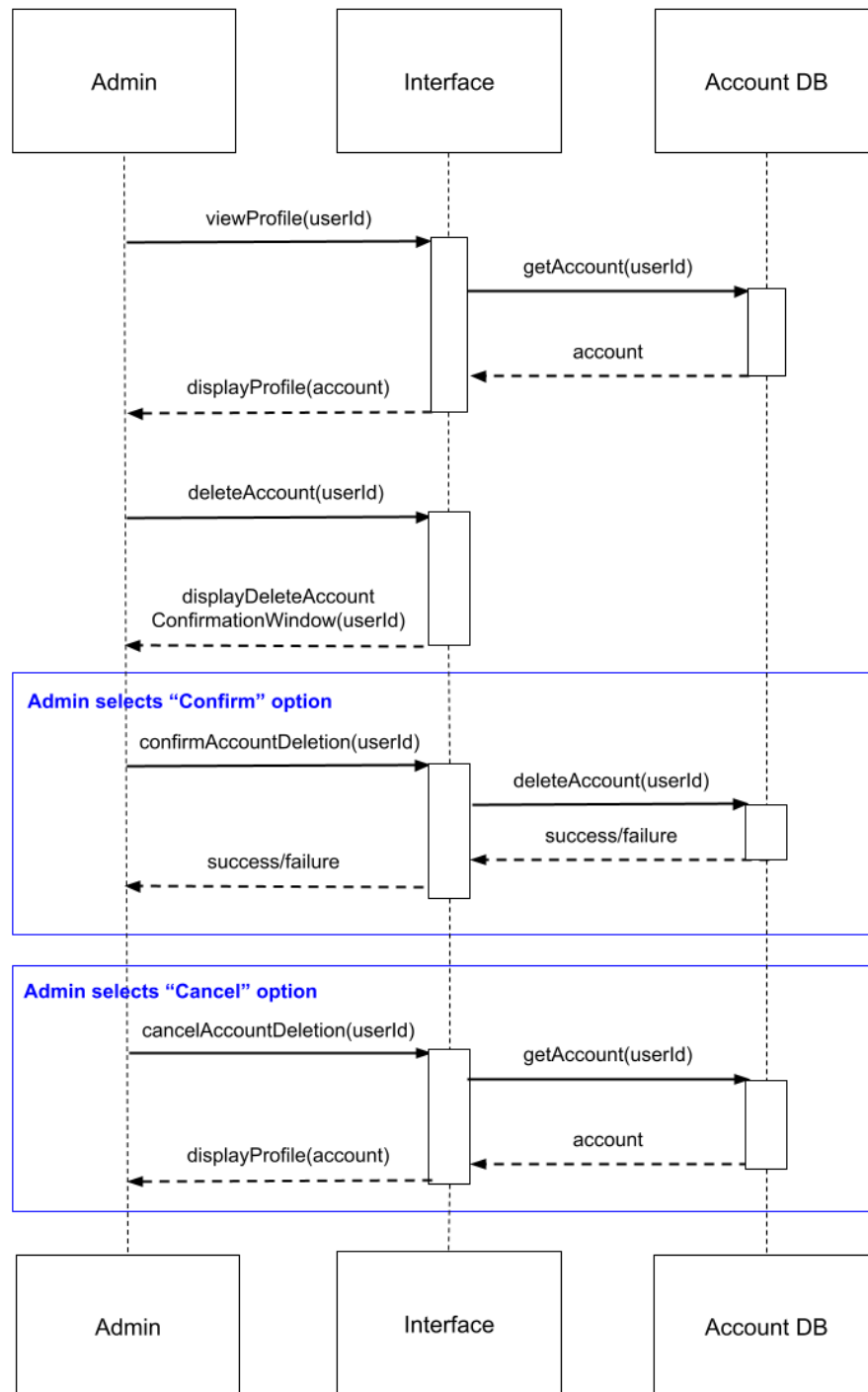


Figure 6.3.10: Admin Delete Account Sequence Diagram

Figure 6.3.10 represents what happens when an admin deletes a user's account. The admin accesses this option from the user's profile page. First, the system displays a "Delete Account" button. Then, the admin clicks the "Delete Account" button, which causes the system to display a confirmation window asking if the admin wishes to permanently remove the user's account. If the admin clicks the "Confirm" button, the system deletes the account from the account database and displays a message indicating whether the deletion was successful. If the admin clicks the "Cancel" button, the system returns them to the user's profile page.

7. Requirements Matrix

Functional Requirement	SRS	SDS
Create Account	3.3.1	5.1, 6.3.1
Login	3.3.1	5.2, 6.3.2
Logout	3.3.6	5.3, 6.3.3
Create Post	3.3.3	6.3.4
Edit Post	3.3.3	6.3.5
Delete Post	3.3.3	5.5, 6.3.6
Search Job Listings	3.3.4	5.7, 6.3.7
View Post	3.3.3, 3.3.4, 3.3.5	5.6, 6.3.8
Review Post	3.3.5	5.4, 6.3.9
Admin Delete Account	3.3.5	5.10, 6.3.10

8. Appendices

8.1 Author Contributions and revision history

Jacob Dietrich - Wrote sections 1.1 - 1.5 and 2 (Introduction and System Overview). Created pseudocode for create account and delete post.

Abdel Hamid Shehata - Wrote sections 4.1 and 5 (data description and component design). Wrote pseudocode and sequence diagrams for Search Post.

John Thomas - Wrote section 6.2 and 6.3 (except for 6.3.5 and 6.3.6). Created Requirements Matrix. Wrote pseudocode and created sequence diagrams for View Job Post.

Megan Grant - Wrote section 3.3, created pseudocode for Logon and Logout and created sequence diagrams for Logon and Logout. DFD for 3.2.2 and 3.2.3

Thomas Cheek -

Benn Mellinger -

Wrote pseudocode for 5.8 Edit Job Post and 5.9 Delete Job Post. Created 6.3.5 sequence diagram for Edit Post. Created 6.3.6 sequence diagram for Delete Post. Edit and review SDD for turnin. Created data dictionary.

8.2 Revision History

<u>Names(s)</u>	<u>Date</u>	<u>Reason for change(s)</u>	<u>Version</u>
All	10.20.2021	Initial Version	1.0
Megan G	10.22.2021	Incorporating Client (Polack) feedback	1,0.1
Jacob D and Megan G	11.7.2021	Additional editing and updating all sections, reformatted all sections	1.0.2
Jacob D	11.7.2021	Incorporated feedback in sections 1, 2, 3, and 5	1.0.3
Megan G	11.7.2021	Addition of Figures and text for 3.2.2 and 3.2.3	1.0.4
Abdel Hamid Shehata	11.7.2021	Converted Sequence Diagrams for uniformity	1.0.5
John Thomas	11.7.2021	Added sequence diagram and pseudocode for Admin Delete Account, wrote section 6.1 which was missing in the initial version	1.0.6