Computer Science   
Internship/Job Board

Software Design Document

**Team Members**:

Ian Frye

Raymond Kauffman

James Peacemaker

Jordan Reiser

Leon Sepehrar

Miles Spence

Benjamin Springer

**Date:** 11/9/2020

**TABLE OF CONTENTS**

1. **Introduction** 03

1.1 Purpose 03

1.2 Scope 03

1.3 Overview 03

1.4 Reference Material 03

1.5 Definitions and Acronyms 03

2. **System Overview** 03

3. **System Architecture** 04

3.1 Architectural Design 04

3.2 Decomposition Description 07

3.3 Design Rationale 07

4. **Data Design** 07

4.1 Data Description 07

4.2 Data Dictionary 08

5. **Component Design** 09

6. **Human Interface Design** 17

6.1 Overview of User Interface 17

6.2 Screen Images 18

6.3 Screen Objects and Actions 26

7. **Requirements Matrix** 32

8. **Appendices** 33

**1. INTRODUCTION**

**1.1  Purpose**

The purpose of this System Design Document is to give the reader an understanding about how the web-based application, UMW Computer Science Internships, and Job Opportunities are structured. How everything is designed and how everything is linked together.

**1.2  Scope**

The goal of this web application is to have full functionality by the end of the Fall 2020 semester at the University of Mary Washington. This full functionality includes students’ ability to apply to jobs, view employers and their qualities, and to be able to leave reviews about employers on their employer page. Employers have the ability to create their own descriptive page and have the ability to post jobs on the job board. Admins are able to edit employer profiles, edit reviews left on employer profiles, and be able to remove job listings.

**1.3  Overview**

The start of the document is the design paper introduction. Following the introduction, we go over the system overview and go more in-depth about the system architecture. The system overview and system architecture cover how the coding portion of the project was designed and what was implemented to make it all the more efficient and to improve functionality. Following, we have the data design of the application. This section covers how the databases are set up and how we manage/store all of the information contained within the job boards and employers. The component design section of the paper contains pseudocode for different implementations contained in the front and back-end of the application. The Human Interface Design section is to inform the reader of the user experience adjustments we made to the front-end for user accessibility and ease of use.

**1.4  Definitions and Acronyms**

* Primary Key - a column in a table that is used to uniquely identify an instance of an entity.
* Foreign Key - a column in a table that references a primary key in another table in order to link the two tables.
* SQL - structured query language; a language in programming commonly used for databases.
* UMW - University of Mary Washington. The school for which the job board is being created.

**2. SYSTEM OVERVIEW**

The System provides a hub for jobs and internships from companies that are interested in UMW students. This replaces the current method of simply publishing the positions in a google document that is hard for students to find. Therefore the system is designed to be an easily accessible and usable site that companies can post job and internship information for UMW students to see. An Administrator has the ability to delete and edit any information on the site, while Employer accounts have the ability to delete or edit only information that they themselves have posted.

There is a single Administrator account, but Employers can sign up to create their own account. They can enter their email and password, and a confirmation email is sent to their email address to verify the account. Regardless of user type, they login the same way with the email and password used to sign up. An Employer or Admin may post a job or internship that is displayed on the homepage for anyone, including users without an account, to browse. When an Employer attempts to post a job, it is not immediately posted. First, the System sends a notification of a pending job to the Admin and they must approve it. All jobs are hosted on the home page, accompanied by a search bar to filter the jobs by title, company name, location, and description.

Any user is able to view a list of Employers who have posted to the site and click on a specific Employer to see more in-depth information about the company. This information can be edited by that Employer or the Admin. This Employer Profile page also includes a list of reviews that can be posted by any user (including non-account users). If any information in these reviews is deemed inappropriate by another user, they may flag that review, at which point the System sends a notification to the Admin to either approve or delete the review.

**3. SYSTEM ARCHITECTURE**

**3.1  Architectural Design**

The Internship/Job Board system comprises four subsystems, Listings System, Employer Profiles System, Reviews System, and Authentication system. Each system has several functions in which information is either posted to or fetched from a relational database.

The Listings System allows normal users to view listings and search listings and allows privileged users such as the administrator and employers to post listings, remove listings, and edit listings.

The Employer Profiles System allows normal users to search for employers and view employer profiles. It also allows privileged users to remove employers and edit employer information.

The Reviews System allows normal users to view reviews, leave reviews, and flag other reviews and allows privileged users to remove reviews.

The Authentication system is used to authenticate the administrator and employees by allowing them to create accounts and login so that they can perform the privileged operations mentioned above.

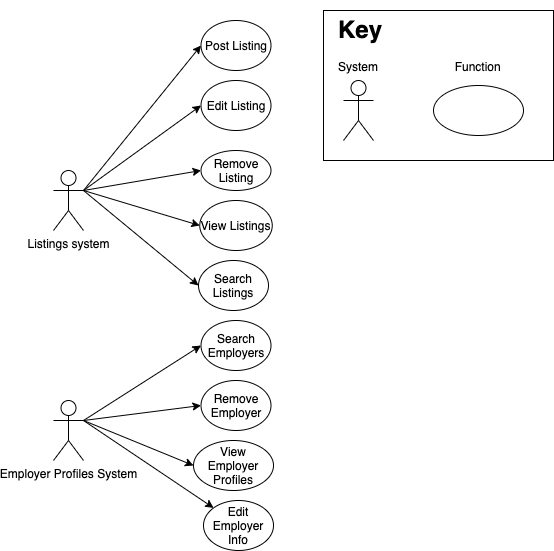


Figure 3.1.1: Above is a context diagram of the Listings System and the Employer Profiles System. It shows which actions are handled by each system.

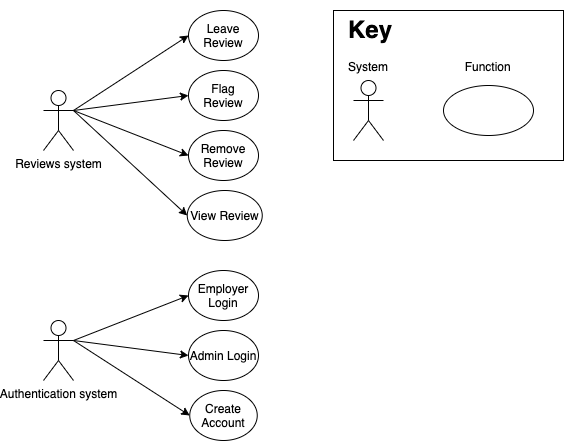


Figure 3.1.2: Above is a context diagram of the Reviews System and the Authentication System. It shows which actions are handled by each system.

**3.2  Decomposition Description**

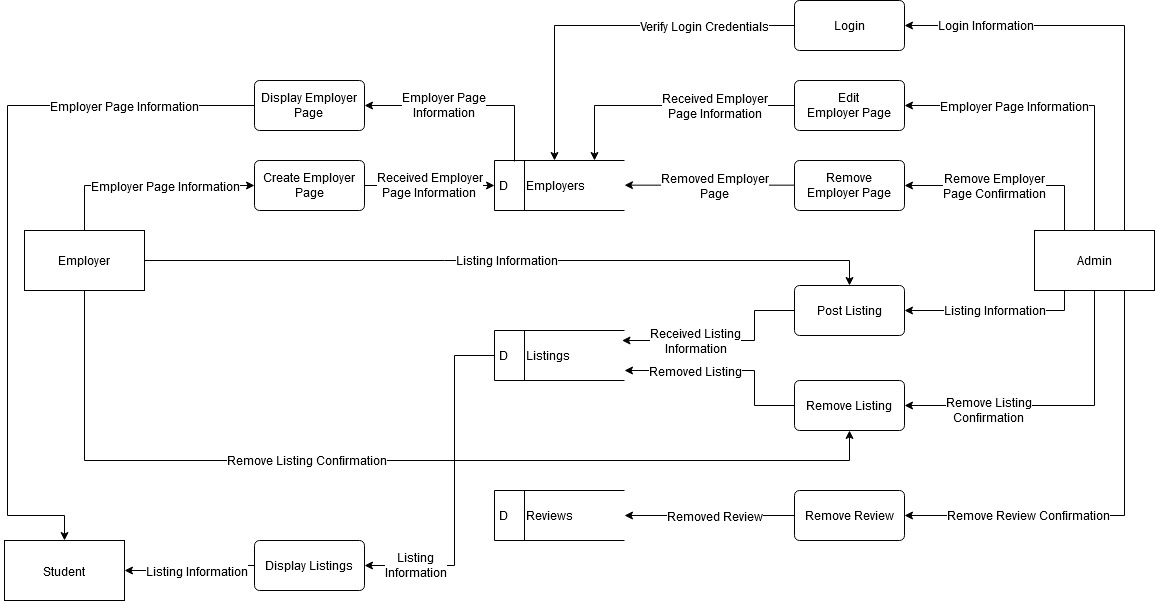


Figure 3.2.1: The Data Flow Diagram of the system. There are three users included in the diagram they are Admin, Employer, and Student. There are three databases included in the diagram they are Employers, Listings (or jobs), and Reviews. Between these are the functions and the data passed to and from the functions.

**3.3  Design Rationale**

The system is divided into four subsystems: Listings System, Employer Profiles System, Reviews System, and Authentication system. The system is a data-centric, web-based information system in which users interact with a relational database by using a web browser as an interface. The subsystems relate to the way in which the database is set up. For example, there is a table for Listings, Employers, and Reviews. Users input information into the database through the site and information which is fetched from the database is displayed on the site. Authentication is also a key component to ensure that the administrator and employees are the only ones able to perform privileged operations. For this reason, Authentication is also a subsystem.

**4. DATA DESIGN**

**4.1  Data Description**

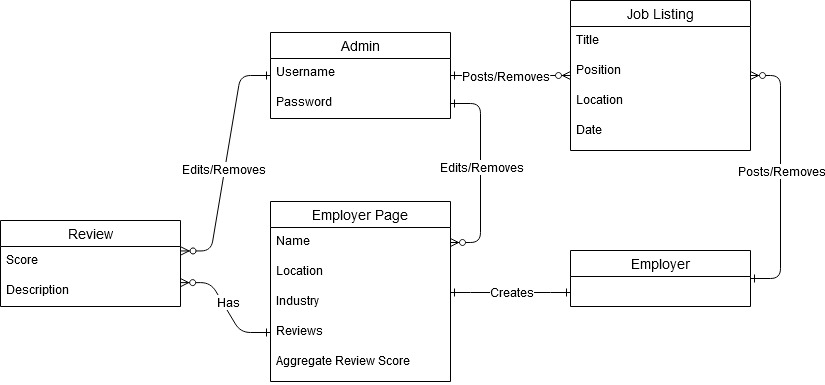
****

Figure 4.1.1: The Entity Relationship Diagram (ERD) of the system for the first two iterations. The entities are Admin, Employer, Employer Page, Review, and Job Listing. Each entity has attributes except the Employer which will have attributes in the third and final iteration. Between each entity is the relationship.

A single SQL relational database containing multiple tables is used to store the key pieces of the information domain, which are the jobs, employers, and employer reviews. Jobs have many attributes; they are a title, employer name, location, start date (for the job posting to be displayed), end date (for the job posting to be displayed), and description. These attributes have been converted into columns in an SQL table named jobs. In addition, the jobs table has a column for a primary key called id. An employer’s attributes are name, email, password, location, industry, and description. These attributes have been converted into columns in an SQL table named employers. In addition, the employer’s table has a column for a primary key called id. An additional attribute of an employer is reviews. Reviews are stored in their own table because there can be multiple reviews per employer and because reviews have several attributes of their own. Their attributes are reviewer, title, description, posted date, affiliation (to the employer), and rating. These attributes have been converted into columns in an SQL table named reviews. In addition, the reviews table has columns for a primary key called id and a foreign key that references the primary key of the employers table. Every time an employer profile is created or removed, a job is posted or removed, or a review is posted or removed the respective table is updated.

**4.2  Data Dictionary**

System Entities:

* Admin (user)
  + There is only one admin and they are the user who manages the system. They are in charge of approving much of the content on the job board and have the highest privileges of any user. They are capable of posting a job, removing a job, removing an employer page, removing a review, and much more.
* Employer (user)
  + An employer has an account and may post a job in hopes of hiring an employee. An employer may create their own job page with information about their company.
* Job (object)
  + A job is created by either an employer or the admin. A job has many required attributes to be filled out in order to be posted, such as a title and job description. It also has dates for when the job should start and stop being displayed on the job board.
* Review (object)
  + A review is left by a student on an employer page. Reviews are left so that other students who are searching for a job may gain a better understanding of a company. A review’s attributes are a reviewer (name), title, description, posted date, affiliation, and rating.
* Student (user)
  + A student is a user who is desiring to find a job from the job board. The only input a student has with the system is posting reviews. A student does not have any login credentials and cannot create an account.

Functions:

* createEmployerPage(name, email, password, location, industry, description)
* displayEmployerPage(employerID)
* displayListings(jobID)
* editEmployerPage(name, email, password, location, industry, description)
* login(email, password)
* postListing(title, employer name, location, start date, end date, description)
* removeEmployerPage(employerID)
* removeListing(jobID)
* removeReview(reviewID)

**5. COMPONENT DESIGN**

This section expands upon the subsystems included in figure 3.2.1 in section 3.2 by splitting up the subsystems into component classes that are either frontend, backend, or connection between the two. Backend classes include post, delete, and get requests to the database, using the information the backend received from the frontend via a query or request. Below are the component classes and associated pseudocode.

**5.1.a Backend login API**

In order to login, an employer or admin must send a post request to the backend server with their username and password. It then checks if that was a valid username and password combination by querying the database, and if it was then it returns a success message with the email and user type (admin or employer) to the frontend. If the combination was invalid, it returns a message “failed”. See the pseudocode for the backend login API below.

Post(query){

Username = query.username

Password = query.password

Request = “Select password FROM admin WHERE username = Username

Result = await pool.query(query, Request)

if(result.successful){

res.json(“status”: “successful”, “email”: “123@umw.mail”, “user\_type”: “admin”)

}else{

res.json(“status”: “failed”)

}

}

**5.1.b Frontend login page**

The frontend code provides an interface for the user to interact with. This login page allows the user to enter the email and password associated with their account and submit it to the system, which queries the backend server. If the result is successful, the user is navigated to the homepage, otherwise, the page displays an error message. See the pseudocode for the login page below.

import login() from utils.js

render {

input for email address (this.state.email)

input for password (this.state.password)

submit button

}

on submit button click

LoginResult = await login ({email:this.state.email, password:this.state.password})

if LoginResult has { status:success }

store cookie { email:email, user\_type:admin}

then navigate to home page.

else

navigate to login error page to try again

**5.1.c Frontend connection to Backend functions**

In order for the user to login, the information that they enter must be posted to the backend server. This is done through functions imported from a file named ‘utils.js’. The functions inside fetches from the backend server, checking if the email and password entered were a valid result, then relay that back to the frontend. This code also handles the creation of user cookies when a user successfully logs in, as well as deleting the user cookies when the user logs out. See the pseudocode for the utils.js functions below.

login function({email:email address, password:password})

LoginResult = await fetch (login api address with header that includes {email:email address, password:password})

return LoginResult

logout()

if login cookie exists:

delete cookie

this.state.email=""

this.state.password=""

loginStatus()

if cookie exists {

if cookie has {user\_type : admin}

return {logged\_in: true, user\_type:admin}

if cookie has {user\_type: employer}

return {logged\_in: true, user\_type:employer, user\_email: email}

else

return {error: user type not found}

} else {

return {logged\_in: false}

}

**5.2.a Backend Employer Profile API**

The system queries the employee database based on id number and dynamically creates a page for each individual employee. This backend function then returns individual employee information in a Json to the utils.js function calling it in the frontend. If the employer id is not found, the console logs an error. See the pseudocode for the Employer Profile backend API below.

app.get(“/find-employer-by-id”...){

Const id from req.query.id

Console.log(id)

try {

Template = “SELECT id, name, location, industry, description FROM employers WHERE id = $1 ”

Response = await pool.query(template, [id])

Console.log (“profile selected”)

Const employers = response.rows.map(function(item){

Return{

Id: item.id, Name: item.name, location: item.location, description: item.description}})

Const ret = {rows: employers)

res.json(ret)

}catch (err){

console.log(err)

Json status: error

}

}

**5.2.b Frontend Employer Profiles page**

The system displays an employer dynamically via the employer’s id. Given the system has a valid id for the employer, the system displays the company, location, industry, and description for that employer. See the pseudocode for the Employer Profiles page below.

import get\_reviews() from utils.js

import findemployerbyid from utils.js

render {

Company: {employer.name}

Location: {employer.location}

industry: {employer.industry}

Description: {employer.description}

Reviews:

HTML table:

{reviews.reviewer}

{reviews.title}

{reviews.rating}

{reviews.description}

{reviews.posted\_date}

{reviews.affiliation}

}

Post.getInitialProps = async ({ query }) =>{

Employer = await findemployerbyid (query.employerid)

Reviews = await get\_reviews(query.employerid)

if (Employer.rows[0])

return {employer: Employer.rows[0], reviews: Reviews}

else

return {employer: “not found”}

}

Export default Post

**5.3.a Backend Add Review API**

The system adds a review from the user inputting the review credentials from the frontend: employer\_id, reviewer, title, rating, description, posted\_date, and affiliation. The employer\_id is the id of the employer for that specific review. Upon success, the review is inserted into the database. See the backend API pseudocode for adding reviews below.

app.post(/create-review){

try{

Employer\_id from req.body

Reviewer from req.body

Title from req.body

Rating from req.body

Description from req.body

Posted\_date from req.body

Affiliation from req.body

Template = “INSERT into reviews (reviewer, title, rating, description, posted\_date, affiliation) VALUES ($1, $2, $3, $4, $5, $6)”

Response = await.. (template, [reviewer, title, rating, description, posted\_date, affiliation])

Console log “review added”

Json status: review added

}catch(err){

Console log (err)

}

}

**5.3.b Frontend Add Review**

The system create-review page allows the user to add a review with the review’s credentials. Upon clicking submit, the review data is sent to the backend to be created. See pseudocode for the Frontend Add Review page below.

import create\_review() from utils.js

render {

input for reviewer name (this.state.reviewer)

input for title (this.state.title)

input for rating (this.state.rating)

input for description (this.state.description)

input for posted\_date (this.state.posted\_date)

input for affiliation (this.state.affiliation)

submit button

}

on submit button click

Review = await create\_review ({reviewer:this.state.reviewer, title:this.state.title, rating:this.state.rating, description:this.state.description, posted\_date:this.state.posted\_date, affiliation:this.state.affiliation})

if Review has { status:success }

Navigate to employer profile page

else

navigate to review error page to try again

**5.4.a Backend Remove Review API**

The system removes a review from the employer page with the matching id and employer name. Upon success, the review is removed from the database. See the backend API pseudocode for removing reviews below.

\*remove review from employer page\*

app.delete(“../remove-review”){

try{

reviewId from req.body

employer from req.body

Template = “REMOVE from reviews WHERE (reviewId, employer) VALUES ($1, $2)”

Response = await.. (template, [reviewId, employer])

Console log “review removed”

Json status: review removed

}catch(err){

Console log (err)

}

**5.4.b Frontend Remove Review**

The system displays an employer page dynamically. Upon click of a remove review button, the review is removed. See pseudocode for the Frontend Remove Review page below.

\*import functions from util.js\*

import remove\_review() from util.js

render {

Company: {props.employer.name}

Location: {props.employer.location}

industry: {props.employer.industry}

Description: {props.employer.description}

Reviews:

HTML table:

{props.reviews.reviewer}

{props.reviews.title}

{props.reviews.rating}

{props.reviews.description}

{props.reviews.posted\_date}

{props.reviews.affiliation}

}

on remove review button click

Review = await remove\_review ({reviewer:this.state.reviewer, title:this.state.title, rating:this.state.rating, description:this.state.description, posted\_date:this.state.posted\_date, affiliation:this.state.affiliation})

if Review has { status:success }

Navigate to employer profile page

else

navigate to review error page to try again

**5.4.c Frontend connection to Backend functions**

Functions are imported from a file named ‘utils.js’. In exports, the functions that are imported by the frontend code are listed. In this case, remove\_review(info) is exported. See the pseudocode for the utils.js functions for remove review below.

Async function remove\_review(idnum){

Return await fetch(`localhost/find-review-by-id?id=${idnum}`).then(function(resp){

Console.log idnum

Return resp.json()

})

}

In exports:

remove\_review: function(info){

Return remove\_review(info).catch(handleError)

}

**5.5.a Backend Edit/Remove Employer Info API**

The system edits or removes information from the employer page with the matching id and employer name. Since editing and removing are almost two completely different tasks, there are two distinct tasks. One is removing information like a review and the other is editing information that requires updating the database. See the backend API pseudocode for editing or removing employer information below.

\*login as an Admin\*

post(query){

Username = query.username

Password = query.password

Request = “Select password FROM admins WHERE username = Username

Result = await pool.query(query, Request)

if(result.successful){

res.json(“status”: “successful”, “email”: “123@umw.mail”)

}else{

res.json(“status”: “failed”)

}

}

\*remove review from employer page\*

app.delete(“../remove-review”){

try{

reviewId from req.body

employer from req.body

Template = “REMOVE from reviews WHERE (reviewId, employer) VALUES ($1, $2)”

Response = await.. (template, [reviewId, employer])

Console log “review removed”

Json status: review removed

}catch(err){

Console log (err)

}

app.post(“../edit-review”){

app.post(“../remove-review”){

try{

reviewer from req.body

title from req.body

rating from req.body

description from req.body

posting\_date from req.body

affiliation from req.body

Template = “UPDATE from reviews SET (reviewer = reviewer, title = title, rating = rating, description - description, posting\_date = posting\_data, affiliation = affiliation) WHERE reviewId = reviewId”

Response = await.. (template, [reviewId, employer])

Console log “review removed”

Json status: review removed

}catch(err){

Console log (err)

}

}

**5.5.b Frontend Edit/Remove Employer Info**

The system displays an employer page dynamically. Upon a submit button click the employer page information is updated or removed. See pseudocode for the Frontend editing or removing employer information page below.

import edit\_review() from utils.js

render {

input for reviewer name (this.state.reviewer)

input for title (this.state.title)

input for rating (this.state.rating)

input for description (this.state.description)

input for posted\_date (this.state.posted\_date)

input for affiliation (this.state.affiliation)

submit button

}

on submit button click

Review = await edit\_review ({reviewer:this.state.reviewer, title:this.state.title, rating:this.state.rating, description:this.state.description, posted\_date:this.state.posted\_date, affiliation:this.state.affiliation})

if Review has { status:success }

Navigate to employer profile page

else

navigate to review error page to try again

**5.4.c Frontend connection to Backend functions**

Functions are imported from a file named ‘utils.js’. In exports, the functions that are imported by the frontend code are listed. In this case, edit\_review(info) is exported. See the pseudocode for the utils.js functions for editing and removing employer information below.

Async function edit\_review(info){

Const header = {

‘Accept’: “application/json”,

‘Content-Type’: ‘application/x-www-form-urlencoded’

}

Const searchParams = new URLSearchParams(info)

Return await fetch(localhost/edit-review, {method: “POST”, headers: header, body searchParams})

}

In exports:

edit\_review: function(info){

Return edit\_review(info).catch(handleError)

}

**6. HUMAN INTERFACE DESIGN**

**6.1  Overview of User Interface**

The top of each page includes a dynamic menu that changes depending on who is logged in. When the home page loads the first time, the menu just has two options: "Home" and "Log in to post a job." In this iteration, the only user who can log in is the Admin. Many of the pages have features that only appear to the Admin.

"Log in to post a job" takes the user to a page where they can enter an email address and password. A successful login switches the user to the Admin view. Once logged in, the Admin sees "Home," "Post a job" and "Log out" in the top menu.

"Post a job" takes the Admin to a page with a form where they can enter a job title, industry, location, and job description to add a job to the website.

"Log out" removes the features specific to the Admin until they log in again. The top menu reverts to its original state.

The website serves two main purposes. It provides a list of jobs and internships for students, and it also provides a list of employers where current or former students can post reviews. The home page displays the list of jobs and internships. A navigational tab allows the users to switch to the list of employers and back again. Both the job list page and the employer list page include a search bar that allows users to refine the list.

On the job list page, users are able to click on a job title to read more about an individual job. The individual job page includes information such as job title, employer name, industry type, location, and job description. The Admin has a button on the individual job page that allows them to edit the job or delete the job.

On the employer list page, users are able to click on an employer name to read more about an individual employer. The individual employer profile page includes information such as employer name, industry type, location, and employer description. It also includes a list of reviews that have been posted and an average score. Each review has a title, a score from 1-5, and the text of the review. A form at the bottom allows users to post new reviews. The Admin has a button on the employer profile page that allows them to edit the job or delete the job. The Admin also has a button next to each review that allows them to edit or delete the review.

**6.2  Screen Images**

The following images show a prototype of the final website. Not all features shown in the images are implemented in this iteration.

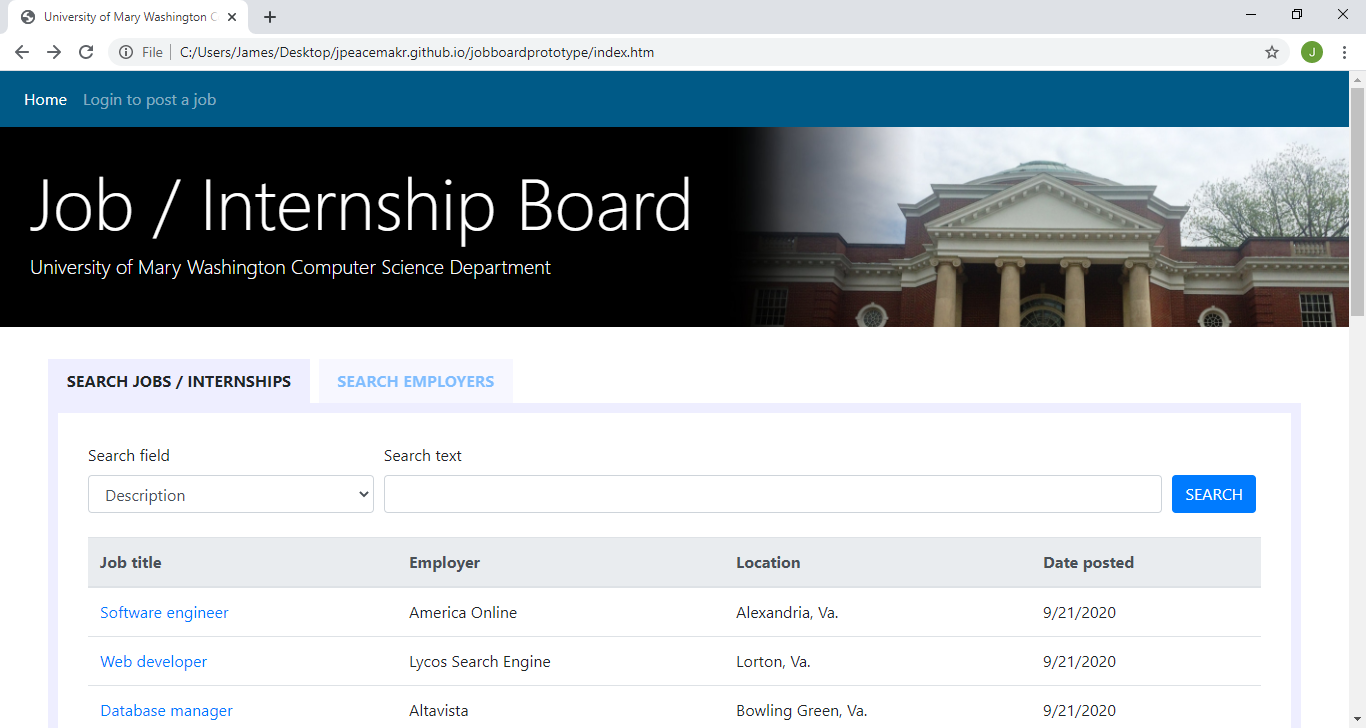


Figure 6.2.1: The display jobs page (the home page) lists all jobs in order by date posted. It also includes a search bar that refines the jobs listed as the user type into it.

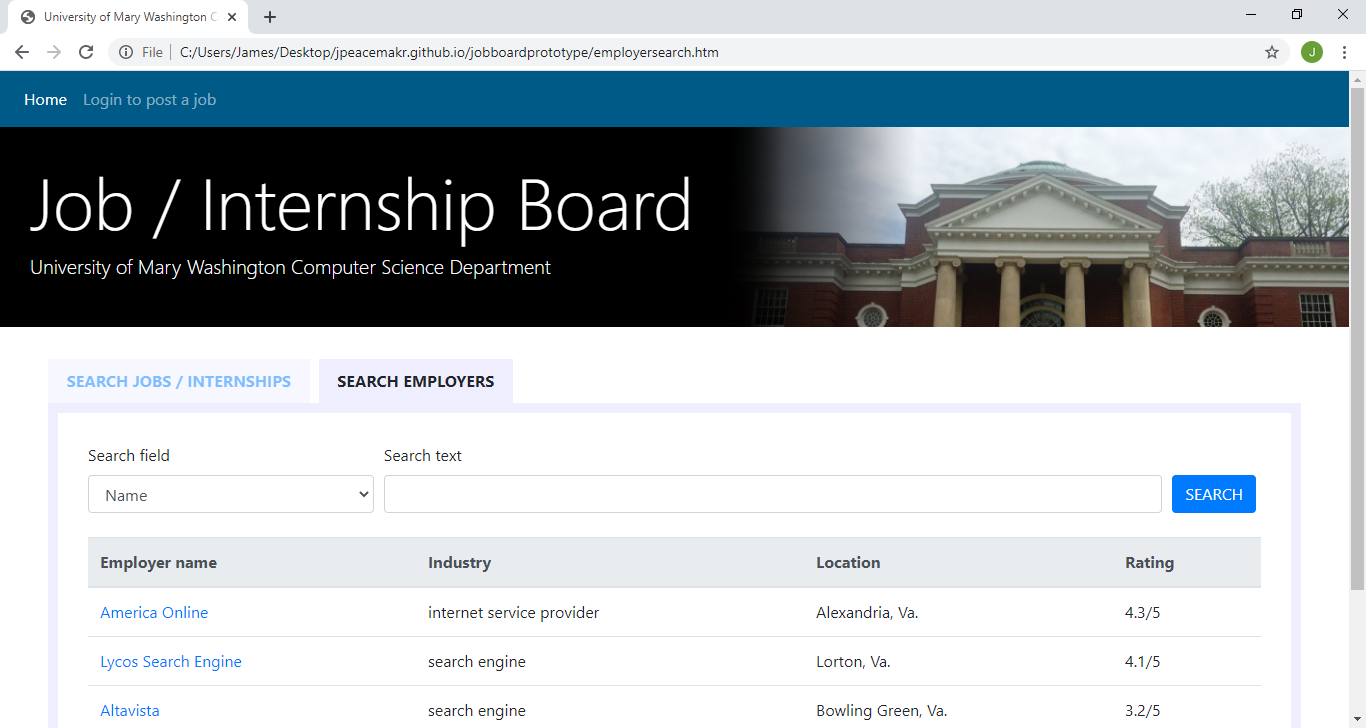


Figure 6.2.2: The display employers page mirrors the look of the display jobs page so they appear on separate tabs within the home page. It lists all employers sorted by name and includes their industry, location, and an average rating of reviews. There is also a search bar that refines the list of employers as the user types into it.

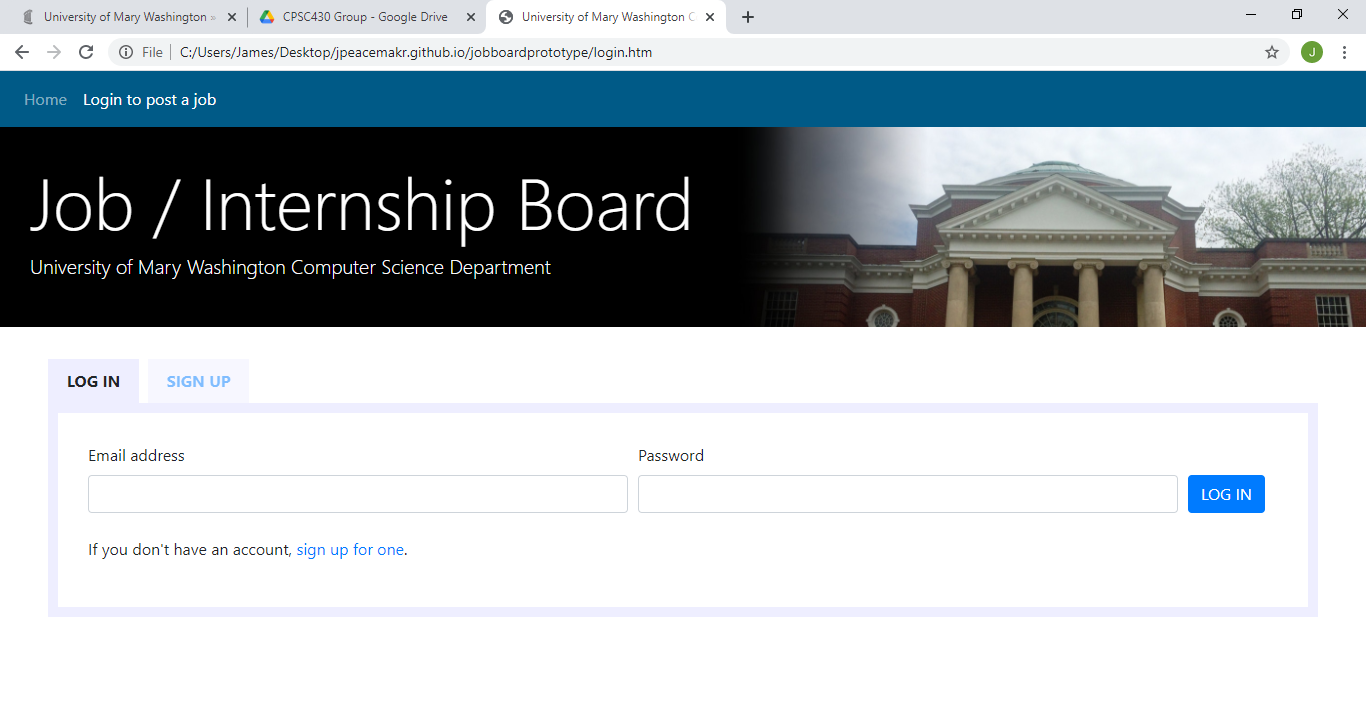


Figure 6.2.3: The login page allows a user to log in as an Admin. If the wrong email address or password is typed in, the words "If you don't have an account, sign up for one" is replaced with a message saying that the login is incorrect. If the login is correct, the message says they logged in successfully. Once logged in, the dynamic menu at the top in blue also changes to show the user name, and the menu changes to add actions or pages available to that user. Note: the sign up tab shown above is not included in this iteration.

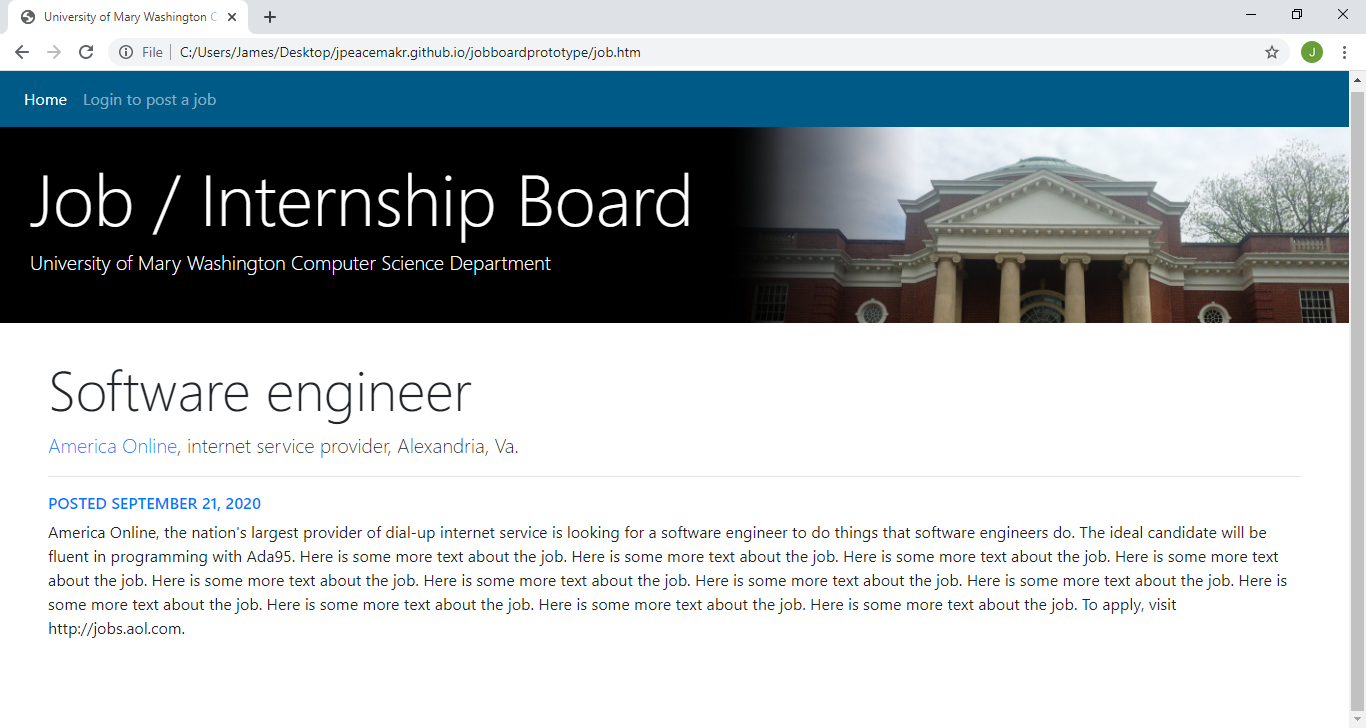


Figure 6.2.4a: The individual job page gives the job title, company, industry, location, and description.

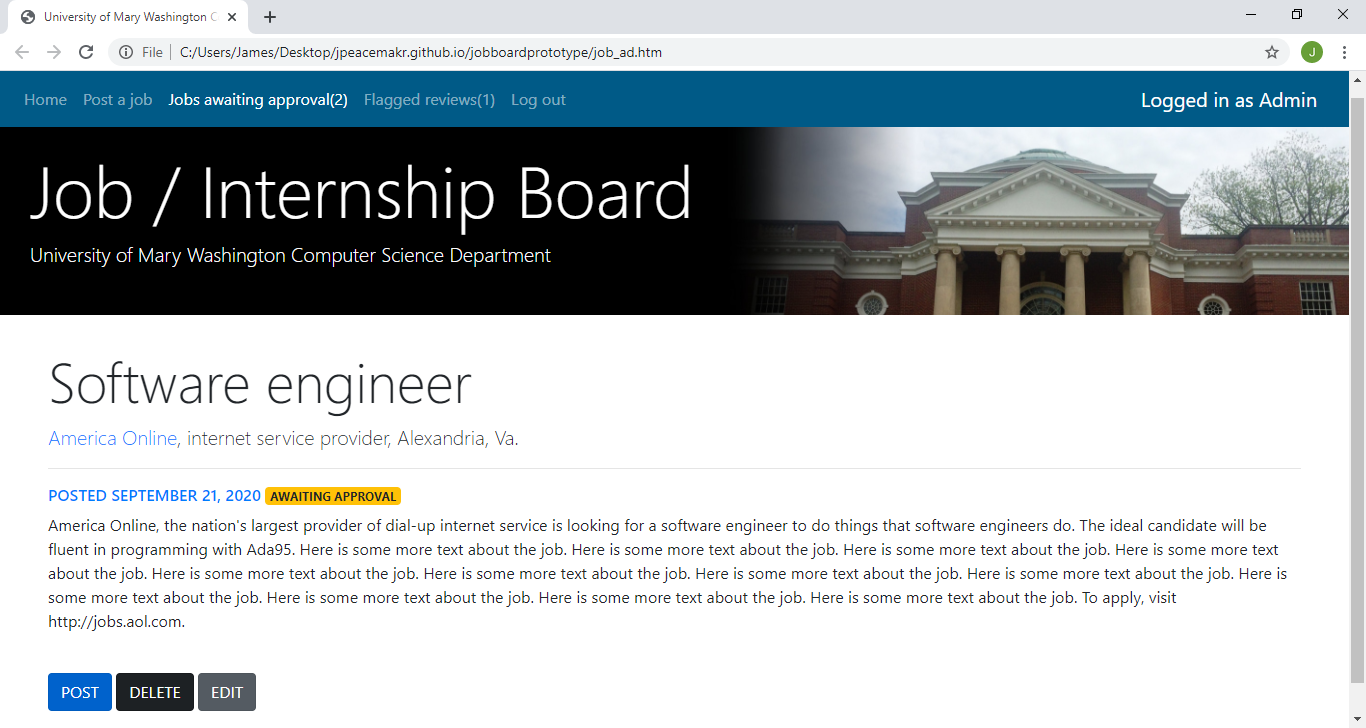


Figure 6.2.4b: The individual job page shows some additional features when logged in as an admin such as "delete" and "edit." The "awaiting approval" flag in yellow and the "post" button is not included in this iteration. Only the admin can post jobs in this iteration, not employers.

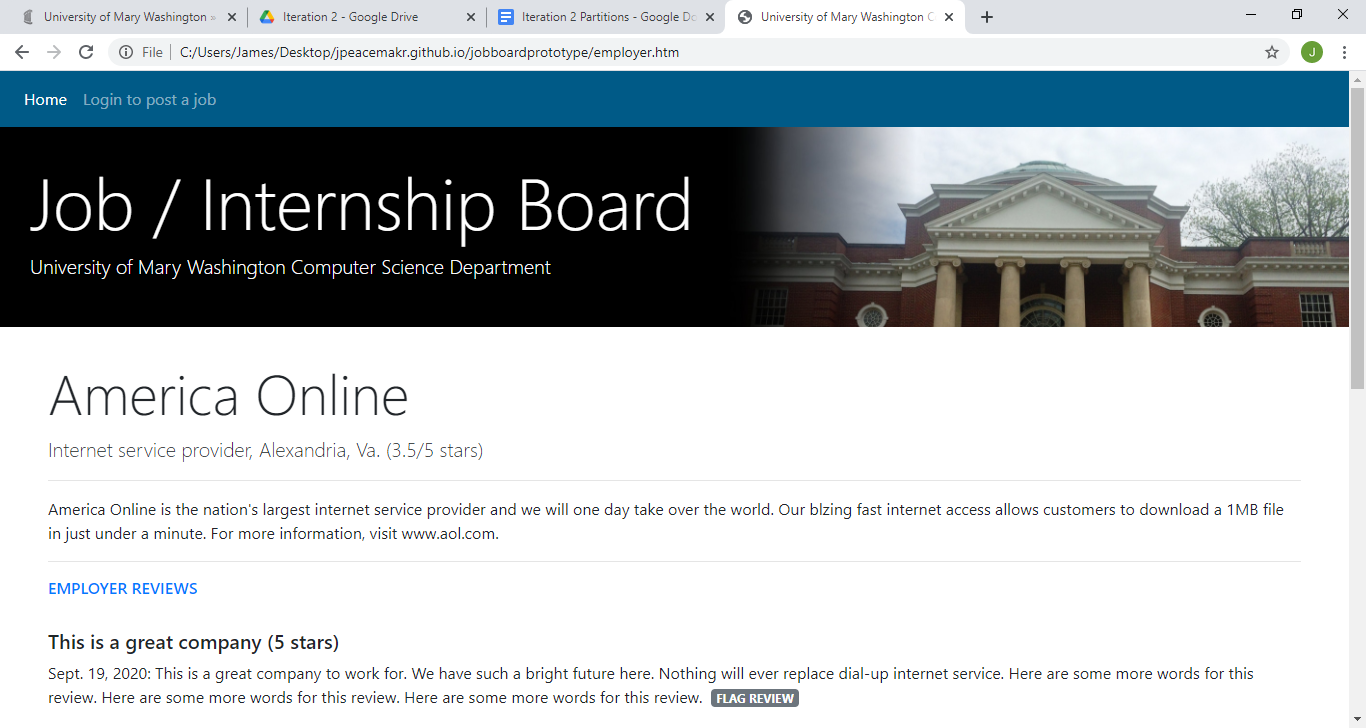


Figure 6.2.5a: The employer profile page allows users to view information about an employer. The page includes the name, industry, location, an aggregate score based on reviews, a description, reviews that have been posted, and a form at the bottom for users to post reviews (see figure 6.2.5c). Note: the flag review button shown above is not included in this iteration.

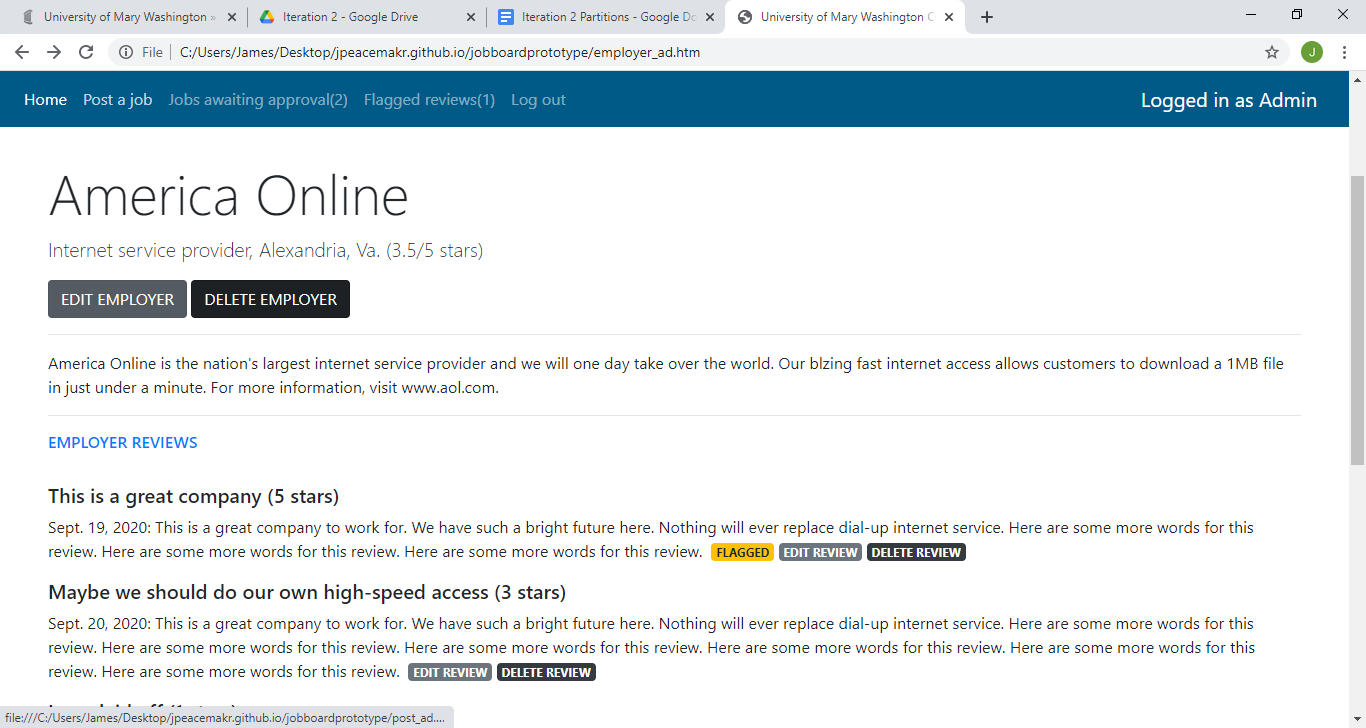


Figure 6.2.5b: The employer profile has extra functionality when a user is logged in as an admin. It shows the same information as figure 6.2.5a, but also adds buttons to edit or delete the employer profile or to edit or delete a review. Clicking to edit the employer profile navigates to figure 6.2.7. Clicking to edit a review navigates to figure 6.2.8. Clicking to delete the employer profile or a review creates a popup alert so the admin must confirm by hitting "OK" or "Cancel". Note: the flagged alert shown above is not included in this iteration.

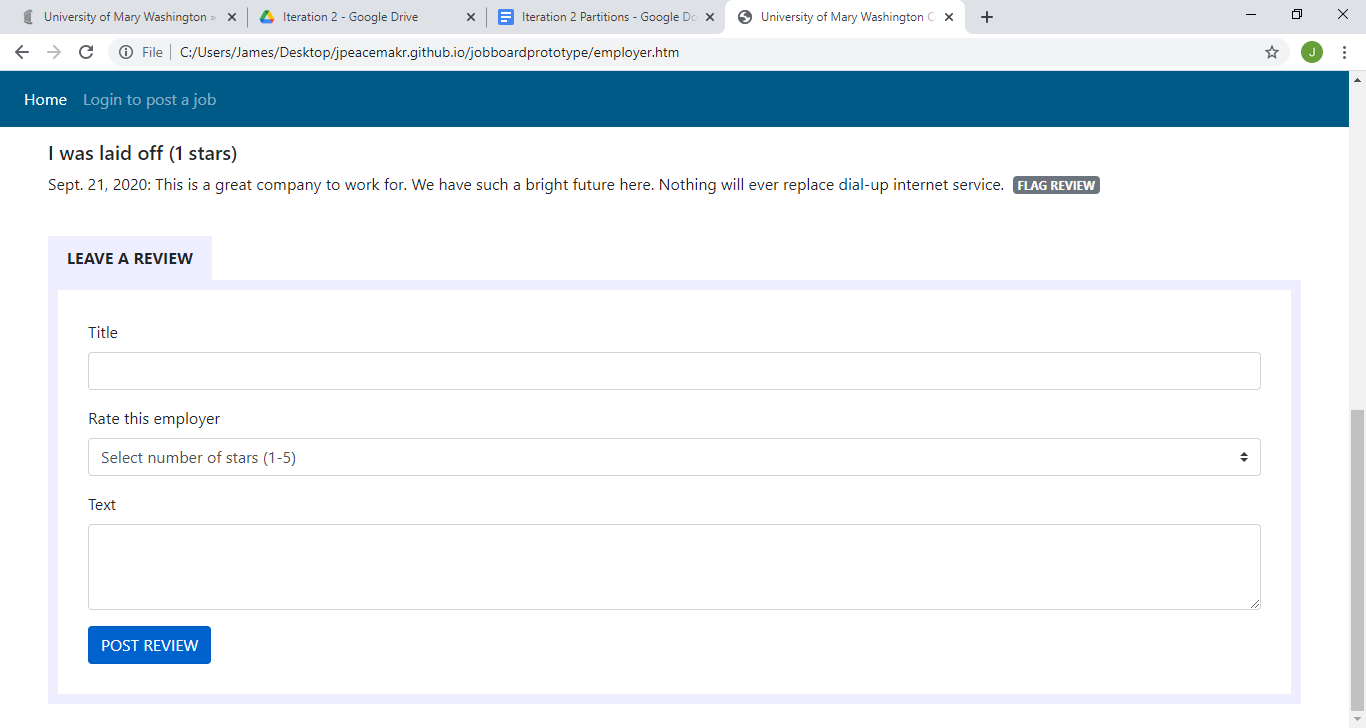


Figure 6.2.5c: The employer profile page also incorporates a form to leave a review for that employer. When submitted, the employer profile page is updated with the new review.

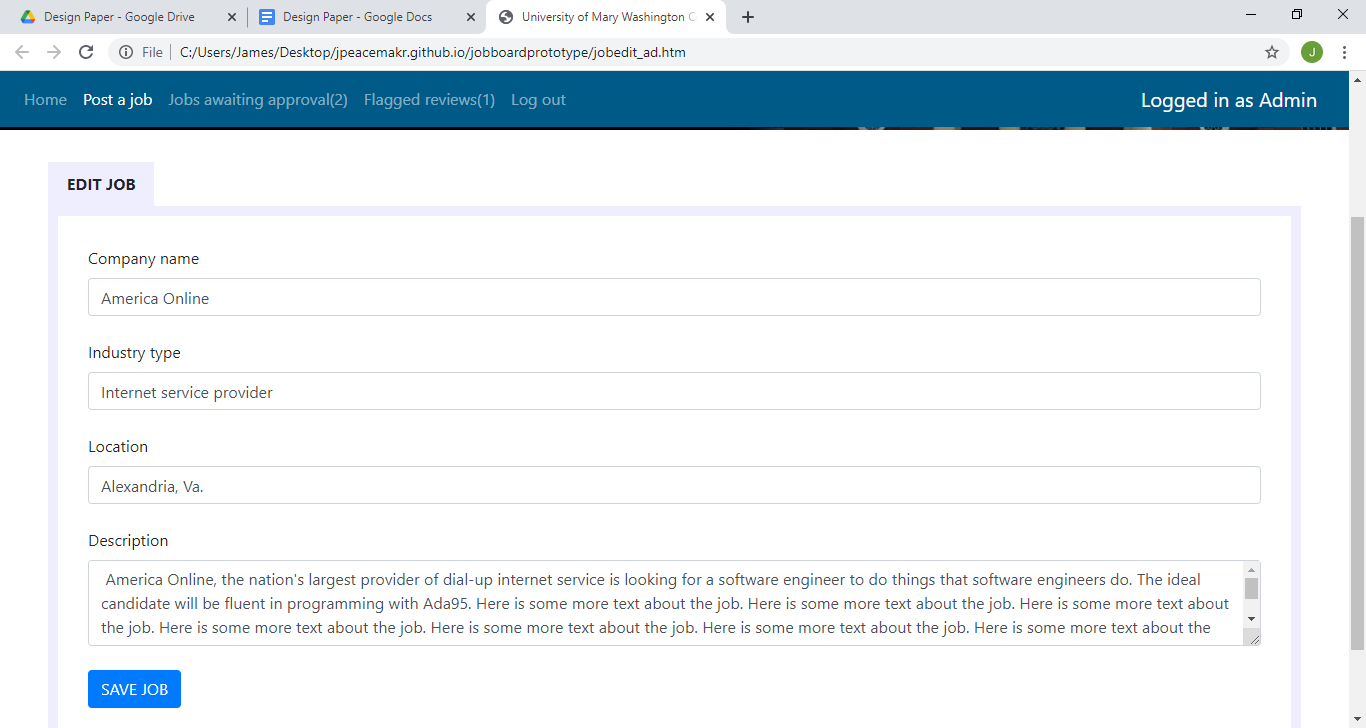


Figure 6.2.6: The edit job page, which is only accessible when the admin is logged in, allows the admin to change any of the information of a job.

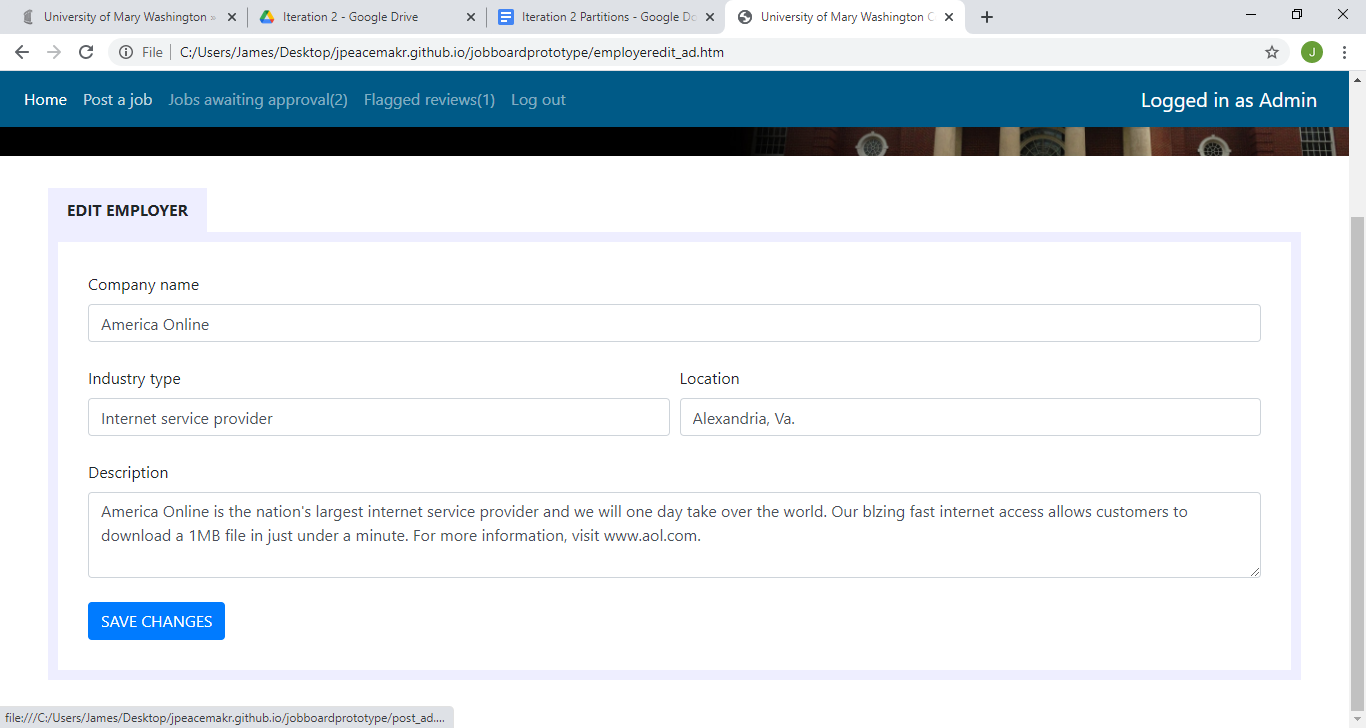


Figure 6.2.7: The edit employer profile page, which is only accessible when the admin is logged in, allows the admin to edit any of the information for the employer (name, industry, location or description). It does not provide for editing or deleting reviews, which is done by clicking on the buttons shown in figure 6.2.5b. Note: the "Jobs awaiting approval" and "Flagged reviews" links in the top menu are not a part of this iteration.

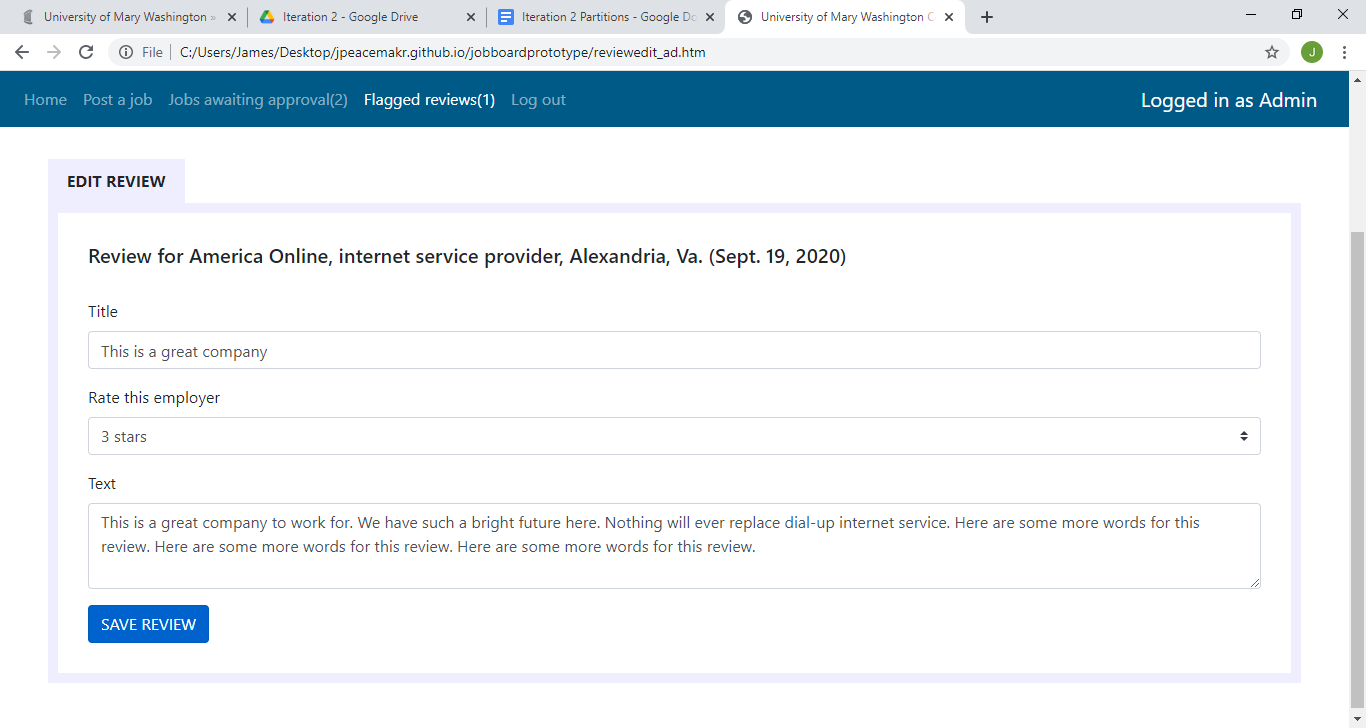


Figure 6.2.8: The edit review page, which is only accessible when the admin is logged in, allows the admin to change any of the information of a review.

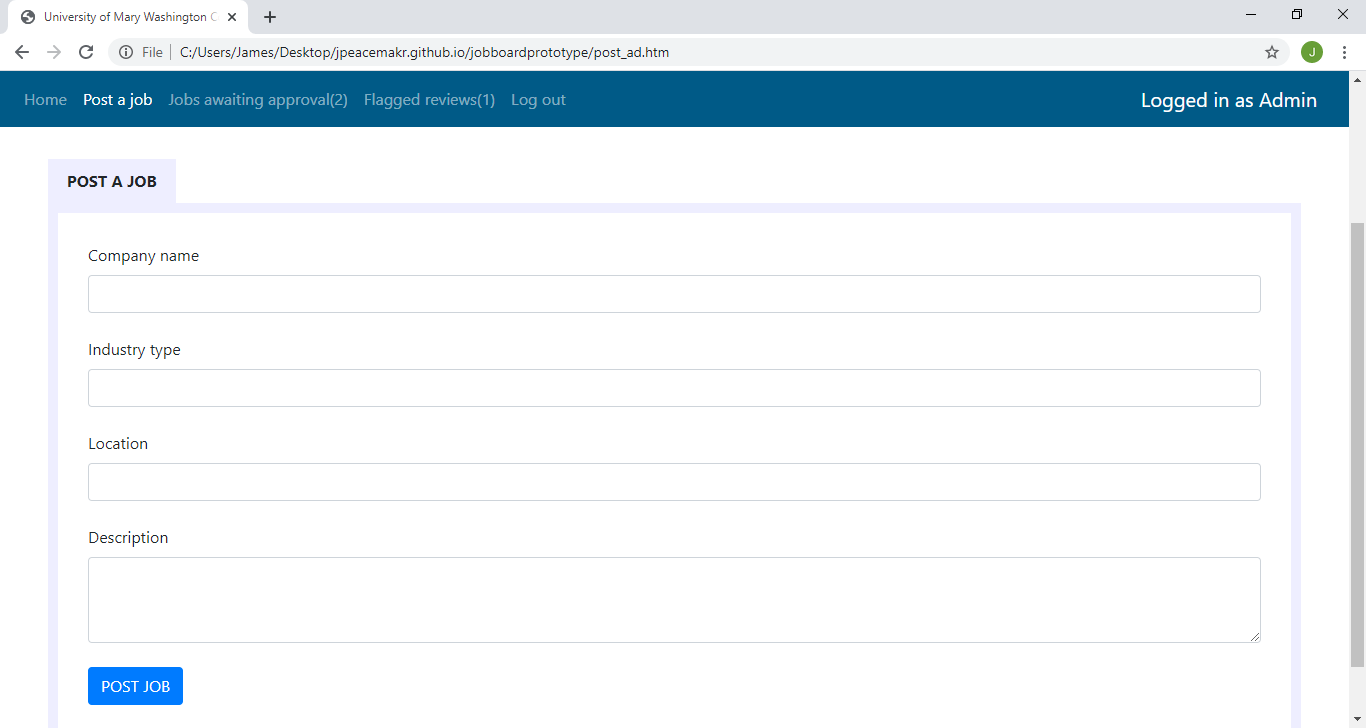


Figure 6.2.9: The post job page, which is only accessible when the admin is logged in, adds a job to the job list. There are fields for a company name, industry type, location, and description.

**6.3  Screen Objects and Actions**

Here is a list of screen objects and the actions that correspond to them:

* **The top navigation bar** is present on every page. It is a dynamic menu that changes depending on who is logged in. It includes:
  + "Home" link navigates to the home page (/index).
  + "Log in to post jobs" link navigates to the login page (/login). (only appears if not logged in)
  + "Post a job" link navigates to the post job page (/post\_job). (only appears if logged in as Admin)
  + "Log out" link deletes the login information cookies and the navigation bar refreshes. (only appears if logged in as Admin)
  + "Jobs awaiting approval" and "Flagged reviews" are not included in this iteration.
* **The display jobs page** (the home page, /index) shown in Figure 6.2.1.
  + "Search employers" link navigates to the display employers page (/index\_employers).
  + "Search" bar refines the listings below it depending on the words typed in it. The search checks all fields and the job description for a word that matches.
  + Links to individual job pages in the job listings (/viewjob/[job id number]).
* **The display employers page** (/index\_employers) shown in Figure 6.2.2.
  + "Search jobs" link navigates to the display jobs page (/index).
  + "Search" bar refines the listings below it depending on the words typed in it. The search checks all fields and the employer description for a word that matches.
  + Links in the employer listings navigate to individual employer pages (/viewemployer/[employer id number]).
* **The login page** (/login) shown in Figure 6.2.3.
  + "Email address" text box to be used by the form.
  + "Password" text box to be used by the form. Text should not be visible.
  + "Log in" button submits the form information to log in. The login information is checked to see if it is correct. If it is correct, the page updates the page to display a line of text that says " Login successful" and cookies are created to store the information about who is logged in. If the login is incorrect, the page is updated to show "Login incorrect."
  + "Sign up" links are not a part of this iteration.
* **The individual job page** (/viewjob/[job id number]) shown in Figure 6.2.4a and 6.2.4b. These pages are generated dynamically based on the job id number.
  + Text includes the job title, employer name, industry, location, and job description.
  + The employer name link navigates to the employer profile page (/viewemployer/[employer id number]).
  + "Edit" button" (only appears if logged in as Admin)
  + "Delete" button (only appears if logged in as Admin)
  + "Awaiting approval" tag and "Post" button are not used in this iteration, as all jobs are posted by the Admin.
* **The employer profile page** (/viewemployer/[employer id number]) shown in Figure 6.2.5a, Figure 6.2.5b, and Figure 6.2.5c. These pages are generated dynamically based on the employer id number.
  + Text includes the employer name, industry, location, a score based on average reviews, and description.
  + "Edit employer" button navigates to the edit employer page (/editemployer/[employer id number]). (only appears if logged in as Admin)
  + "Delete employer" button prompts the user if they want to delete the employer. If yes, the employer and all related jobs and reviews are deleted. If no, the prompt disappears. (only appears if logged in as Admin)
  + Employer reviews are shown with a title, review text, and rating from 1-5.
  + "Leave a review" form consists of three fields. A text input for title, a dropdown for the rating from 1-5, and a text input for the review. A "Post review" button submits the form. The employer profile page is then updated with the new review. All form fields must be filled out for the submit button to work.
  + "Flag review," "Edit review" and "Delete review" buttons are not included in this iteration.
* **The edit job page** (/editjob/[job id number]) shown in Figure 6.2.6.
  + Text input fields for job name, employer name, industry type, location, and job description show the current values and can be changed by editing the text.
  + "Save job" button submits the form to update the employer information and navigate back to the display jobs page (/index). The "Save job" button is not active if the text fields are not all filled out.
* **The edit employer profile page** (/editemployer/[employer id number]) shown in Figure 6.2.7.
  + Text input fields for employer name, industry type, location, and description show the current values and can be changed by editing the text.
  + "Save changes" button submits the form to update the employer information and navigate back to the display employers page (/index\_employers). The "Save changes" button is not active if the text fields are not all filled out.
* **The edit review page** (/editreview/[review id number]) shown in Figure 6.2.8.
  + Informational text that displays employer name, industry, location, and date the review was made.
  + Input fields for review title, rating 1-5, and review text show the current values and can be changed by editing the text or choosing a new value from the rating drop down menu.
  + "Save review" button submits the form to update the review and then navigates back to the employer profile page (/viewemployer/[employer id number]) . The "Save review" button is not active if the text fields are not all filled out.
* **The post job page** (/post\_job) shown in Figure 6.2.9.
  + Text input fields for job title, company name, industry, location, and job description.
  + "Post job" button submits the form to post the job and then displays text that says "Job posted." The "Post job" button is not active if all text fields are not filled out.

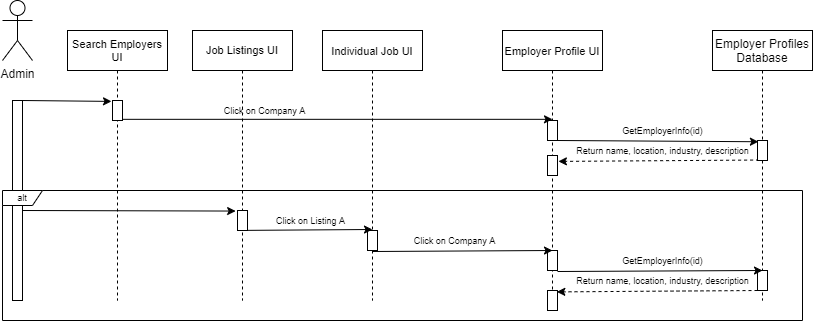


Figure 6.3.1: The sequence diagram for the admin or any user to navigate to an individual page about an employer, including the alternate flow showing the possibility of the user clicking on the profile from a listing.

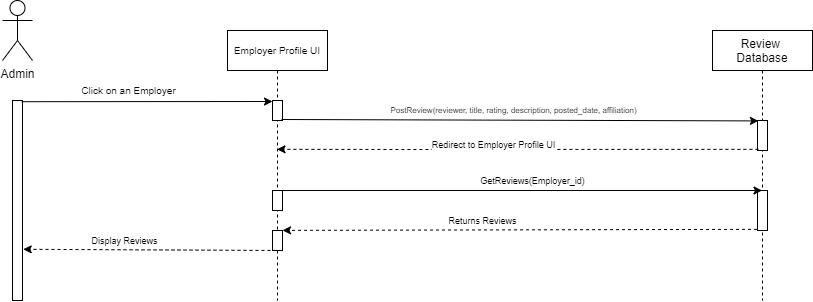


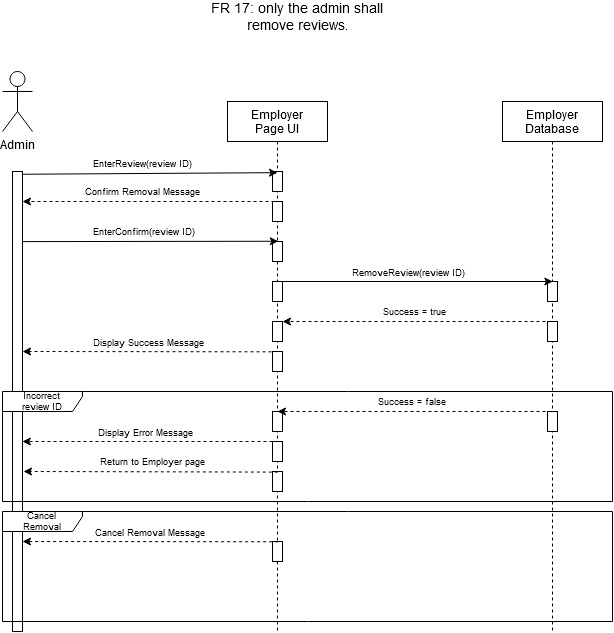
Figure 6.3.2: The sequence diagram for the system to update the reviews database with a new review, and for it to update the employer page with all reviews.

Figure 6.3.3: The sequence diagram for removing a review from the database. Included are two alternate flows in case there is an error retrieving the reviewID or in case the user cancels the removal.

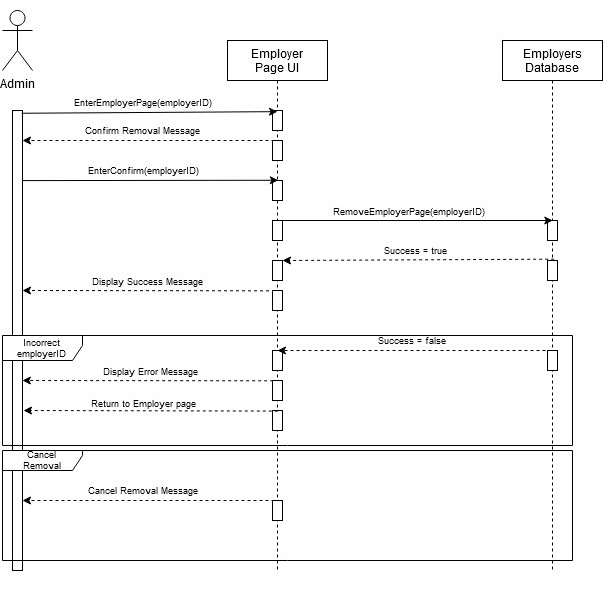


Figure 6.3.4: The sequence diagram for removing an employer page from the database. Included are two alternate flows in case there is an error retrieving the employerID or in case the user cancels the removal.

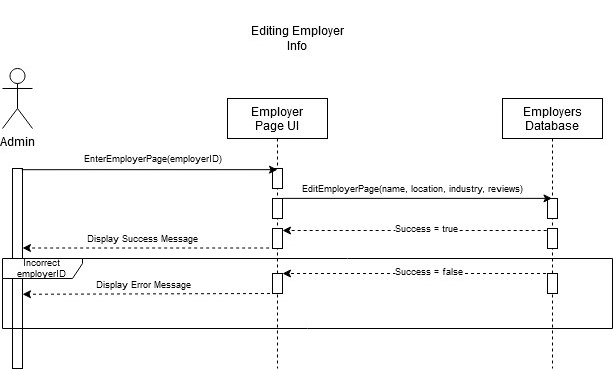


Figure 6.3.5: The sequence diagram for editing an employer page. Included is an alternate flow in case there is an error retrieving the employerID.

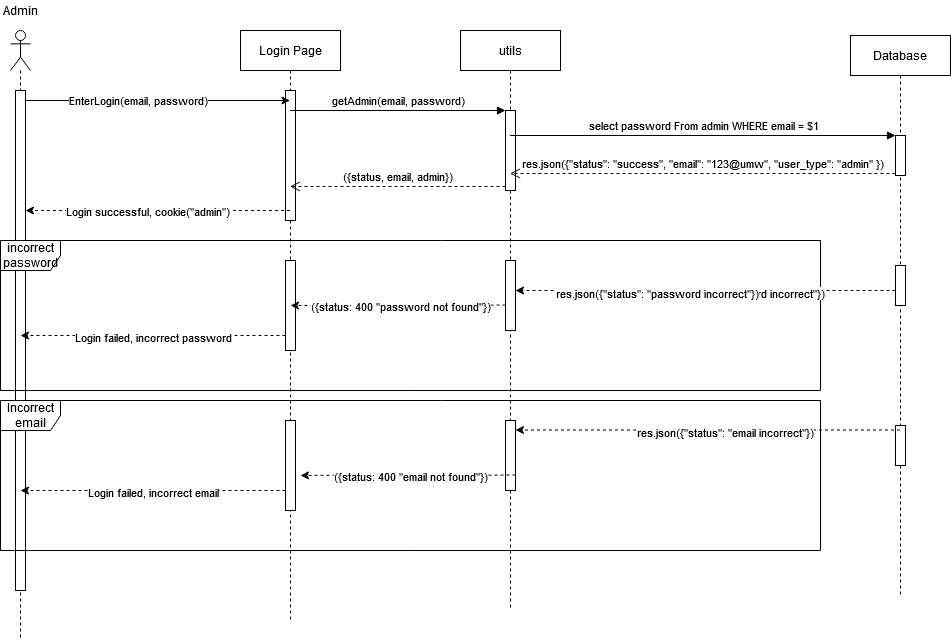


Figure 6.3.6: The sequence diagram for admin login. This diagram also includes the alternate flows for when the password or email is not found within the user database.

**7. REQUIREMENTS MATRIX**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Use Case Specifications** | **SRD** | **SDD** | **Prototype** | **Testing** | **Implementation** |
| **Functional Requirements** |  |  |  |  |  |
| FR 1: Admin Login | 2.4.1.c, 3.1.1.a | 5.1, 6.3.6 | 6.2.3 |  |  |
| FR 5: Login Verification | 2.4.1.e, 3.1.1.c | 5.1, 6.3.6 | 6.2.3 |  |  |
| FR 6: Job Listings Displayed on Home Page | 2.4.1.n, 3.1.2.g | Added in Iteration 1 | 6.2.1 |  |  |
| FR 7: Admin/Employer Job Listing Creation | 2.4.1.l, 3.1.2.a | Added in Iteration 1 | 6.2.9 |  |  |
| FR 9: Job Listing Search | 2.4.1.o, 3.1.2.h | Added in Iteration 1 | 6.2.1 |  |  |
| FR 12: Employer Profiles | 2.4.1.g, 3.1.4.a | 5.2, 6.3.1 | 6.2.2, 6.2.5a |  |  |
| FR 13: Admin Edit Reviews/Employer Info | 3.1.3.a | 5.5, 6.3.4, 6.3.5 | 6.2.5b, 6.2.6, 6.2.8 |  |  |
| FR 14: Post Review | 2.4.1.p, 3.1.3.b | 5.3, 6.3.2 | 6.2.5c |  |  |
| FR 17: Admin Remove Reviews | 2.4.1.f, 3.1.3.d | 5.4, 6.3.3 | 6.2.5b |  |  |
| FR 18: Job Description Page | 2.4.1.n, 3.1.2.i | Added in Iteration 1 | 6.2.4a |  |  |
| FR 19: Admin and Employers Remove Listings | 2.4.1.i, 3.1.2.f | Added in Iteration 1 | 6.2.4b |  |  |

**8. APPENDICES**

**8.1  Contributions**

* Ian Frye: I worked with Leon Sepehrar and Benjamin Springer on the sequence diagrams and pseudocode for adding reviews and viewing employer profiles. I also wrote sections 3.1 and 3.3. I reviewed and edited the introduction.
* Raymond Kauffman: I worked with James Peacemaker to do the sequence diagram and pseudocode for logging in. I wrote the System Overview and reviewed and edited sections 5 and 7.
* James Peacemaker: I worked with Raymond Kauffman to do the sequence diagram and pseudocode for logging in. I wrote Section 6 (Human Interface Design) except for the sequence diagrams. I edited Section 2 (System Overview).
* Jordan Reiser: I worked with Miles Spence to do the sequence diagram and pseudocode for editing and removing reviews on employer pages. I wrote the Introduction to the paper. I edited and reviewed the Human Interface and Design in section 6.
* Leon Sepehrar: I worked with Ian Frye and Benjamin Springer on the sequence diagrams and pseudocode for adding/viewing reviews and employer profiles. I wrote Section 7 (Requirements Matrix), and edited/reviewed Section 4 (Data Design).
* Miles Spence: I worked with Jordan Reiser to do the sequence diagrams and pseudocode for editing and removing reviews and employer information on employer pages. I wrote Section 4 (Data Design) and reviewed and edited Section 5 (Component Design).
* Benjamin Springer: I worked with Ian Frye and Leon Sepehrar on the sequence diagram and pseudocode for adding review and employer profiles. I edited Section 3.1 and 3.3 (System Architecture).