Team 1: Erika Iwata, Won Jin Sim, Duy Nguyen, Michael Cervantes
Date: Dec 13, 2023

Final Report

Title: Prosper

Description:

Prosper is a dynamic job finder website designed to elevate your career prospects. Seamlessly integrating the power of the Adzuna API for global job searches and the Career One Stop API for national government positions, Prosper offers a comprehensive solution for job seekers worldwide.

Key Features:

1. Global Job Search: Explore an extensive range of job opportunities across the globe with Adzuna API. From bustling urban hubs to serene landscapes, Prosper empowers users to discover diverse employment possibilities tailored to their skills and aspirations.
2. National Government Jobs: Navigate the landscape of government employment effortlessly with Career One Stop API. Access a robust database of national government jobs, providing a gateway to impactful public service careers.
3. Personalized Profiles: Create a personalized profile on Prosper to streamline your job search. Save your favorite job listings, track applications, and stay organized throughout your career journey.
4. Job Status Tracking: Gain insights into your job search progress with Prosper's job status tracking feature. Stay informed about the status of your applications and take control of your professional journey.
5. Efficient Job Saving: Save noteworthy job listings for future reference. Prosper's intuitive interface makes it easy to compile and revisit your saved opportunities whenever you need them.

Task Distribution:

Erika: I worked on front-end development and setting up both of the APIs.

Won Jin: I worked on creating the database and linking the database to our replit. I programmed a way that we can add users to the user table from the website.

```
app.get("/user/update/:username", async function (req, res) {
  let sql = `SELECT *
  FROM user
  WHERE username = ? LIMIT 1`;
  let params = [req.params.username];
  let rows = await executeSQL(sql, params);
  res.render("user_edit", { user: rows[0] });
});

app.post("/user/update/:username", async function (req, res) {
  let sql = `UPDATE user
            SET username = ?, pwd = ?, profession = ?
            WHERE username =  ?`;

  let params = [
    req.body.username,
    req.body.pwd,
    req.body.profession,
    req.params.username,
  ];
  let rows = await executeSQL(sql, params);
  res.redirect("/user/update/" + req.body.username);
});

app.get("/user/create", async function (req, res) {
  res.render("user");
});

app.post("/user/create", async function (req, res) {
  let sql = `INSERT INTO user (username, pwd, profession, email, summary, position) VALUES (?, ?, ?, ?, ?, ?)`;

  let params = [
    req.body.username,
    req.body.pwd,
    req.body.profession,
    req.body.email,
    req.body.summary,
    req.body.position,
  ];
  await executeSQL(sql, params);
  res.redirect("/user/update/" + req.body.username);
});
```

```
-- Create the user table with ENUM for position
CREATE TABLE user (
    username VARCHAR(255) PRIMARY KEY,
    pwd VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    profession VARCHAR(255),
    summary TEXT,
    position ENUM('recruiter', 'applicant') NOT NULL
);

-- Create the accnt_info table with ENUM for job_status
CREATE TABLE accnt_info (
    saved_jobs TEXT,
    job_status ENUM('applied', 'interviewed', 'received offer') NOT NULL
);

-- Create the job_posting table with ENUM for job_status
CREATE TABLE job_posting (
    applicants TEXT,
    job_status ENUM('accepting applications', 'position filled') NOT NULL
);
```

Duy:

Login, Sign up, Edit Profile pages:

- Form validation checks.
- Fetch the updated user data from the database.
- Enhance system's overall performance and user experience.
- Solve the authentication issue to ensure restricted access to certain pages post-user logout.

Michael:

- Added Favorite Route (Wasn't able to meet goal line to fully implement)

- Modified some of the database (accnt_info table) and also added some users to the records.

```sql
ALTER TABLE accnt_info add id INT PRIMARY KEY AUTO_INCREMENT;

INSERT INTO `user` (`username`, `pwd`, `email`, `profession`, `summary`, `position`) VALUES
("Coder_Dave", "123mypw123", "fake_email@gmail.com", "Software Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("Coder_Jane", "123mypw123", "fake_email@gmail.com", "Software Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("Coder_Joe", "123mypw123", "fake_email@gmail.com", "Software Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("FullStack_Dev_Mike", "123mypw123", "fake_email@gmail.com", "Fullstack Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("FullStack_Dev_Linda", "123mypw123", "fake_email@gmail.com", "Fullstack Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("Go_Dev_James", "123mypw123", "fake_email@gmail.com", "Software Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("Juan_GO_Dev", "123mypw123", "fake_email@gmail.com", "Software Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("FrontEnd_Dev_Amber", "123mypw123", "fake_email@gmail.com", "Fullstack Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("Coder_Sam", "123mypw123", "fake_email@gmail.com", "Software Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("TS_Dev_Michael", "123mypw123", "fake_email@gmail.com", "Fullstack Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("Angular_Dev_Richard", "123mypw123", "fake_email@gmail.com", "Fullstack Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("Coder_Henry", "123mypw123", "fake_email@gmail.com", "Software Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("Coder_Matt", "123mypw123", "fake_email@gmail.com", "Software Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("Coder_Emily", "123mypw123", "fake_email@gmail.com", "Software Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("Coder_Ashley", "123mypw123", "fake_email@gmail.com", "Software Engineer", "Working at some company for 10 years.... blah blah blah", "applicant"),
("Microsoft", "123mypw123", "fake_email@gmail.com", "Recruiter", "Expanding the MS model", "recruiter"),
("Google", "123mypw123", "fake_email@gmail.com", "Recruiter", "Buidling our company one bloack at a time", "recruiter"),
("US Federal Gov", "123mypw123", "fake_email@gmail.com", "Recruiter", "Looking for the best!", "recruiter");

ALTER TABLE accnt_info DROP COLUMN saved_jobs;
ALTER TABLE accnt_info DROP COLUMN comapny;
ALTER TABLE accnt_info DROP COLUMN userID;

ALTER TABLE accnt_info ADD COLUMN `job_title` VARCHAR(255) AFTER `id`;
ALTER TABLE accnt_info ADD COLUMN `company` VARCHAR(255) AFTER `location`;
ALTER TABLE accnt_info ADD COLUMN `location` VARCHAR(255) AFTER `id`;
ALTER TABLE accnt_info ADD COLUMN `posted_date` VARCHAR(255) AFTER `id`;
ALTER TABLE accnt_info ADD COLUMN `url` VARCHAR(255) AFTER `id`;
ALTER TABLE accnt_info ADD COLUMN `username` varchar(255) AFTER `id`;

SELECT `job_status`, `username`, `url`, `posted_date`, `location`, `company`, `job_title` FROM accnt_info
NATURAL JOIN user
WHERE username LIKE "Coder_Dave";

insert into `accnt_info` (`job_status`, `username`, `url`, `posted_date`, `location`, `company`, `job_title`) VALUES
("Applied", "Coder_Dave", "https://google.com", "12-13-2003", "Los Angeles", "Tech Guru", ".NET Angular Dev"),
("Applied", "Coder_Dave", "https://google.com", "12-13-2003", "Los Angeles", "Tech Guru", ".NET Angular Dev"),
("Applied", "Coder_Dave", "https://google.com", "12-13-2003", "Los Angeles", "Tech Guru", ".NET Angular Dev");
```

- modified nation.ejs

```
response.Jobs.forEach(job => {
  var resultContainer = document.createElement('div');
  resultContainer.classList.add('mb-4', 'list-group-item', 'p-4', 'bg-light', 'rounded-5');

  resultContainer.innerHTML = `<form method="POST" action="/user/fav/add">
    <strong><span id="jobTitle">${job.JobTitle}</span></strong><br>
    Company: <span id="company">${job.Company}<span><br>
    Location: <span id="location">${job.Location}</span><br>
    Date Posted: <span id="postedDate"> ${job.AccquisitionDate}<span><br>
    <a href="${job.URL}" target="_blank" id="url">View Details</a><br><br>
    <button>Favorite</button>
  </form>`;

  resultsContainer.appendChild(resultContainer);
});
}
```

- Modified international.ejs

```
resultContainer.innerHTML = `
  <span id="jobTitle"><strong>${job.title}</strong></span><br>
  Location: <span id="location">${job.location.display_name}<span><br>
  Description: ${job.description}<br>
  Salary: ${job.salary ? `${job.salary_min} - ${job.salary_max} ${job.salary_currency}` : 'Not specified'}<br>
  <a href="${job.redirect_url}" target="_blank" id="url">Learn More</a><br><br>
`;
```

Changes from Original Design:

Original Project Plan:

Requirements:	Assigned to:
Project allows user interaction with at least three different types of form elements (text box, select, radio, checkbox, etc). • We'll be using `<input>`, `<button>`, and `<fieldset>` for the user registration	*Michael*
Project uses Web storage or Sessions. • Assigned to Duy	*Duy*

Project allow users to update existing records in the database, in a friendly way (data is pre-filled). Must update at least three fields. Project uses at least three database tables with at least 10 fields (combined).
• Table 1: user
  ○ Field 1: username
  ○ Field 2: pwd
  ○ Field 3: email
  ○ Field 4: profession
  ○ Field 5: summary
  ○ Field 6: position (recruiter/applicant)
• Table 2: accnt_info
  ○ Field 1: saved_jobs
  ○ Field 2: job_status (applied, interviewed, received offer, etc.)
• Table 3: job_posting
  ○ Field 1: applicants
  ○ Field 2: job_status (accepting applications, position filled, etc.)
	Won Jin

Project allow users to add records to the database • Assigned to team member	*Won Jin*
Project must have at least 50 lines of **client-side** JavaScript code (e.g. form data validation, API calls, etc.) • We'll use multiple form validations and API calls	*Michael*
Project includes at least two local or external Web APIs. As part of your submission, please explain where the Fetch calls are. • We'll be using APIs from ZipRecruiter and Upwork	*Erika*
Project has a nice, professional and consistent design, free of typos. Uses at least 50 CSS properties or Bootstrap. • We'll use 50+ CSS properties and maybe bootstrap	*Erika*

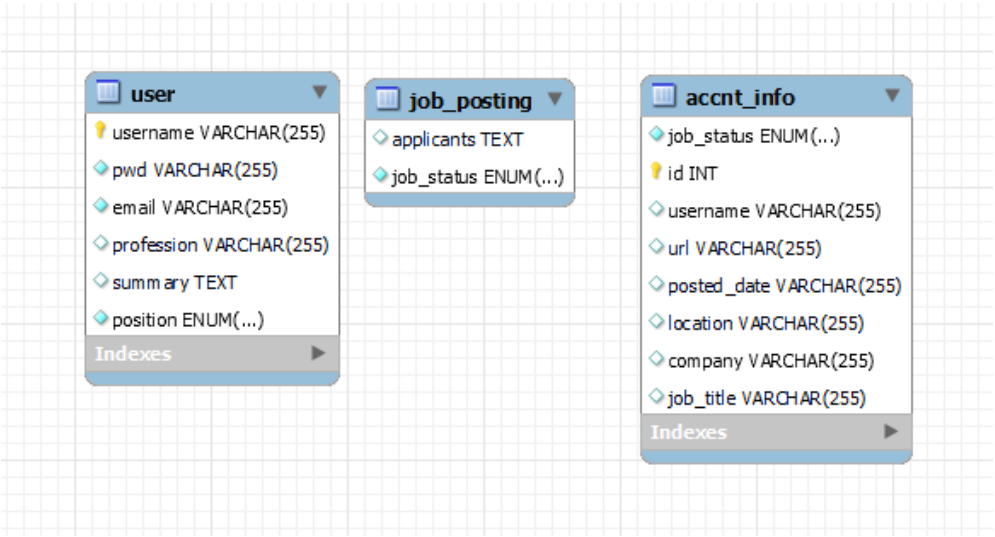
Updated Project Plan:

	Requirements:	Assigned to:
✔	Project allows user interaction with at least three different types of form elements (text box, select, radio, checkbox, etc). • We used `<input>`, `<button>`, `<label>`, and `<option>` for the job search queries	*Michael*
✔	Project uses Web storage or Sessions. • We used sessions to keep users logged in	*Duy*

	Project allow users to update existing records in the database, in a friendly way (data is pre-filled). Must update at least three fields. Project uses at least three database tables with at least 10 fields (combined).
	• Table 1: user
	  ○ Field 1: username
	  ○ Field 2: pwd
	  ○ Field 3: email
	  ○ Field 4: profession
	  ○ Field 5: summary
	  ○ Field 6: position (recruiter/applicant)
	• Table 2: accnt_info
	  ○ Field 1: saved_jobs
	  ○ Field 2: job_status (applied, interviewed, received offer, etc.)
	• Table 3: job_posting
	  ○ Field 1: applicants
	  ○ Field 2: job_status (accepting applications, position filled, etc.)
		Won Jin

✔	Project allow users to add records to the database • We can add users to the user table	*Won Jin*
✔	Project must have at least 50 lines of **client-side** JavaScript code (e.g. form data validation, API calls, etc.) • We'll use multiple form validations and API calls	*Duy*
✔	Project includes at least two local or external Web APIs. As part of your submission, please explain where the Fetch calls are. • We used APIs from Adzuna (global) and Career One Stop (national)	*Erika*
✔	Project has a nice, professional and consistent design, free of typos. Uses at least 50 CSS properties or Bootstrap. • We'll use 50+ CSS properties and maybe bootstrap	*Erika*

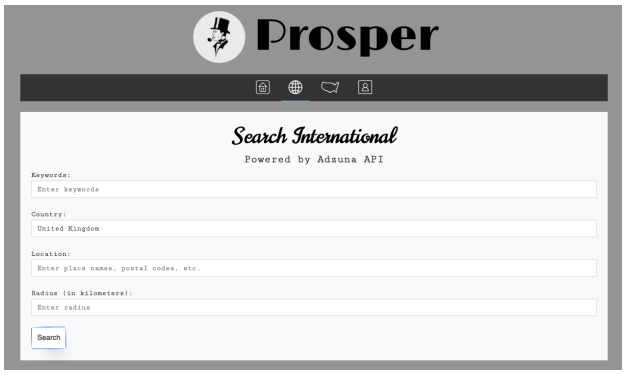
Database Schema:

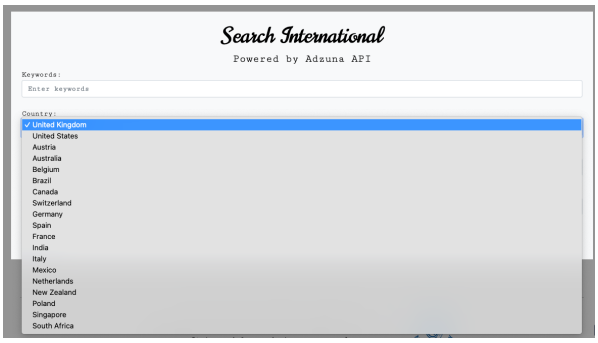


Screenshots of Finished Product:

International Job Search

Users are able to search for jobs by keyword, country, location, and radius (from the center of the location). The form then uses Adzuna API to request the data and displays the results at the bottom of the page.

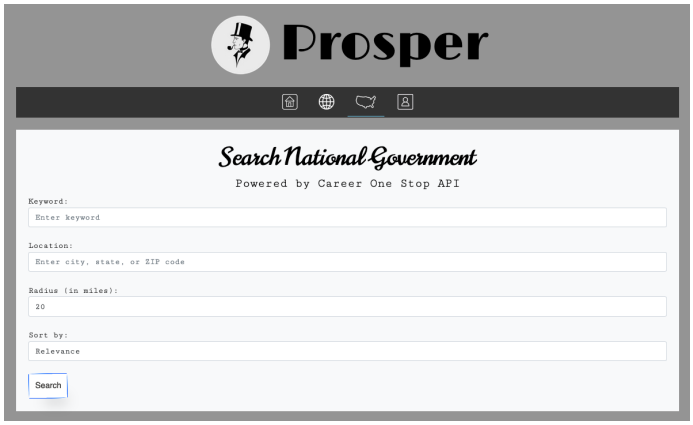




This webpage utilizes the `<button>` for the search button, `<label>` for each of the labels above the input, `<option>` for the country dropdown menu, and `<input>` for keywords, location, and radius.

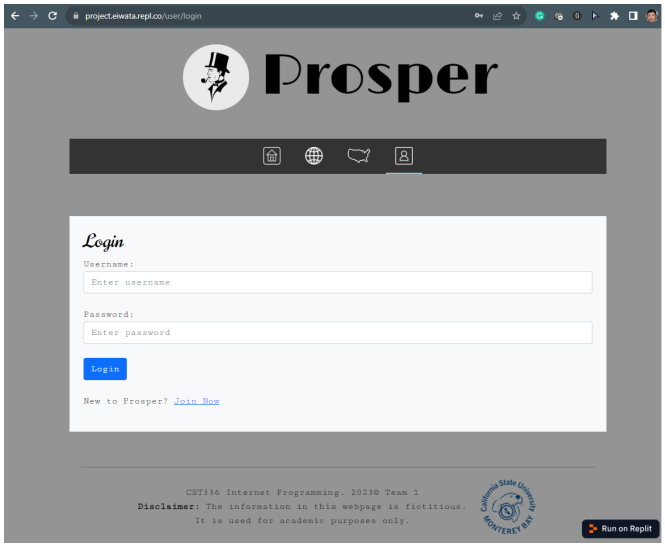
National Government Job Search

Users are able to search for jobs by keyword, country, location, or radius (from the center of the location). The form then uses Career One Stop API to request the data and displays the results at the bottom of the page.



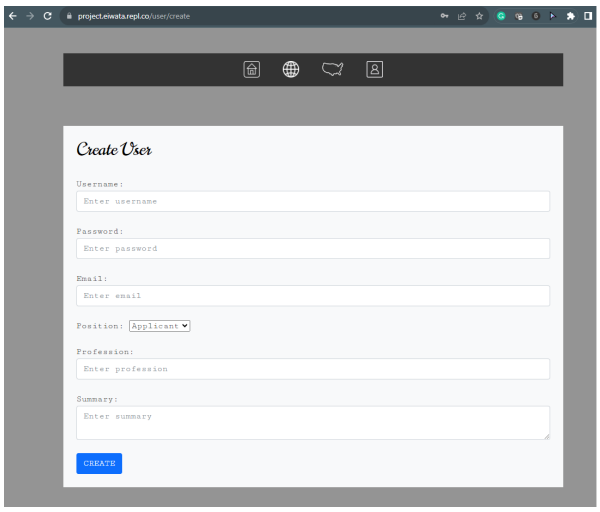
Login

Users are required to enter their email address and a password to log in to the system. Upon successful login, users are redirected to their user profile and gain access to other pages in the navigation bar.



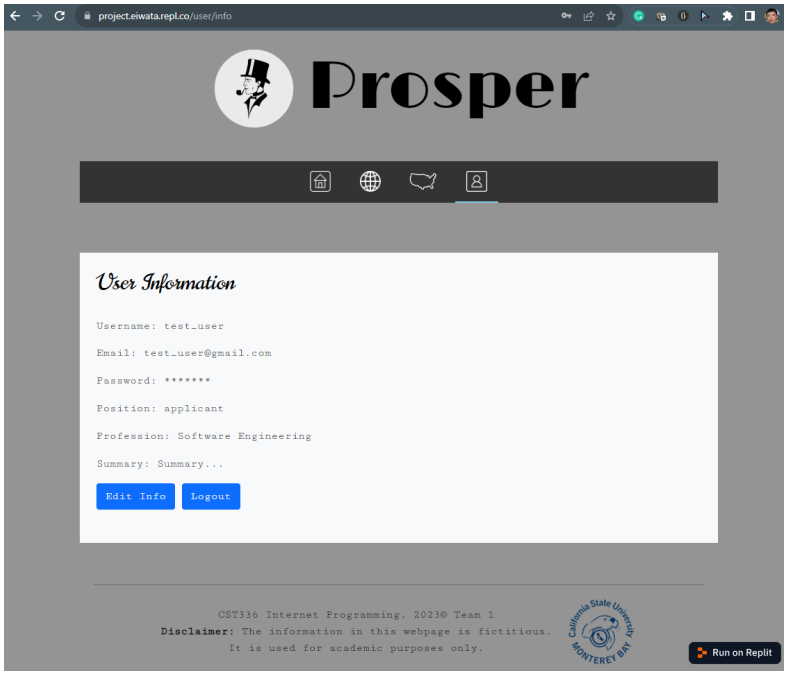
Sign Up

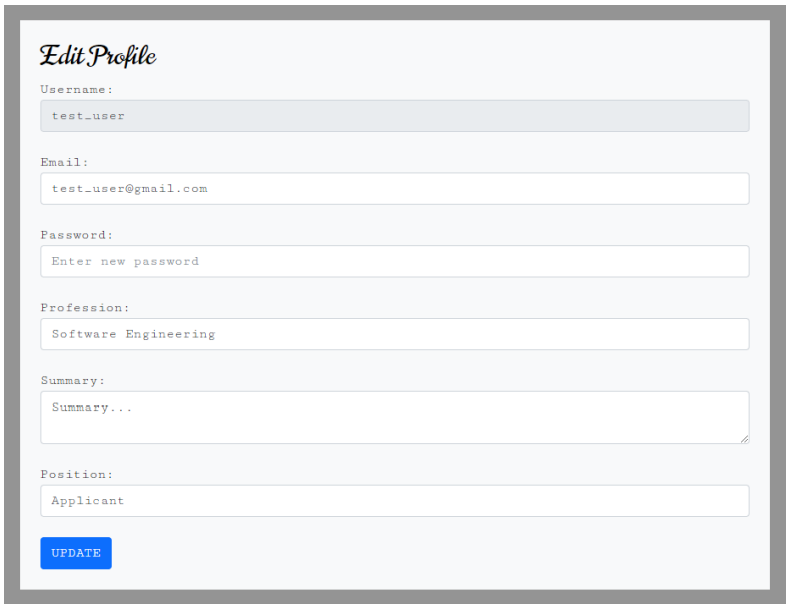
The 'Sign Up' page enables users to create new accounts by providing their user name (which must be unique and not already in use), password, email address, position.. After successful signup, users will be redirected to the login page.



Profile

On the 'Profile' page, users can view their personal information, which is sourced from the 'users' table in our database. Should they wish to modify their details, they can click the 'Edit Info' button to update their email address, password, profession, summary, and position. By clicking the 'Update' button, these changes will be saved in the database. Once they have successfully updated their password, users can log into their account with these updated credentials.





Sessions

Sessions were used for this project to keep users logged in. The below snippet of code generates a secure random secret key for session management and configures the Express application to use the `express-session` middleware with specific options for session handling and security. The middleware is essential for managing user sessions, enabling the storage and retrieval of user-specific data across multiple HTTP requests.

index.js > app.get("/international") callback > ...

```
const express = require("express");
const path = require("path");
const axios = require("axios");
const session = require("express-session");
const pool = require("./dbPool");
const crypto = require("crypto");

const app = express();
const PORT = process.env.PORT || 3000;

app.set("view engine", "ejs");
app.use(express.static("public"));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.set("views", path.join(__dirname, "views"));

//------------------------- MIDDLEWARE ----------------------------//

// Generate a random session secret key
const secretKey = crypto.randomBytes(64).toString("hex");

// Configure session middleware
app.use(
  session({
    secret: secretKey,
    resave: false,
    saveUninitialized: true,
  }),
);

```

API Calls

The first API we used was the [Adzuna API]() for international job searches. The code snippet below is from `views/international.ejs` and it takes the values the users filled out from the form and uses it as the search parameters in the url.

```
views > international.ejs
<!-- Search Results -->
<div id="searchResults" class="mt-4">
  <script>
    function search() {
      // Get your Adzuna API credentials
      var appId = '7bf9cd5a';
      var appKey = '00f06e59f48a26dc9bb6457a9e5fc766';

      // Get values from the form
      var keywords = document.getElementById('keywords').value;
      var country = document.getElementById('country').value;
      var location = document.getElementById('location').value;
      var distance = document.getElementById('distance').value;

      // Build the Adzuna API URL
      var apiUrl = `https://api.adzuna.com/v1/api/jobs/${country}/search/1?app_id=${appId}&app_key=${appKey}&what=${keywords}&where=${location}&distance=${distance}`;

      // Fetch data from the Adzuna API
      fetch(apiUrl)
        .then(response => response.json())
        .then(data => displayResults(data))
        .catch(error => console.error('Error fetching data:', error));
    }
```

The second API we used was the [Career One Stop API]() for national government job searches. The code snippet below is from `views/national.ejs` and it takes the values the users filled out from the form and uses it as the search parameters in the url.

```
views > national.ejs
<div id="jobResults" class="mt-4">
<script>
  function search() {
    var userId = 'd4mMTyPsymXL4Gu';
    var apiKey = 'eZNs8jnd36Rj/Ta8f6195834DGwzlP5dVYMp32K1XomSHAm4SmwpF5uMyHDycMjbpXrWWLYLfsNviVTi/ZMtgA==';

    var keyword = document.getElementById('keyword').value;
    var location = document.getElementById('location').value;
    var radius = document.getElementById('radius').value;
    var sortColumns = document.getElementById('sortColumns').value;

    var url = `https://api.careeronestop.org/v1/jobsearch/${userId}/${keyword}/${location}/${radius}/${sortColumns}/0/0/50/0`;

    fetch(url, {
      headers: {
        'Authorization': `Bearer ${apiKey}`
      }
    })
    .then(response => response.json())
    .then(data => {
      displayResults(data);
    })
    .catch(error => console.error('Error:', error));
  }
```