Job Board

Overview

As a newly-minted software consultant, you've been tasked with expanding upon a job posting application. This repo contains a skeleton codebase for the product. JobbyJobs is a small company that advertises household job services and assigns the jobs to college students. It is currently a manual process:

1. Job posters call a phone number.
2. The receptionist takes down the information.
3. The receptionist enters it in the JobbyJobs application.
4. JobbyJobs staff manually match up posted jobs to job doers.

This is a multi-phased engagement. The goal of this phase is to build upon the existing code to support the job posting process.

Instructions

The client has booked 2-4 hours of your time to complete the following:

- Implement the Required Task (see below) first.
- Afterwards, choose two of the Available Tasks to implement. Clearly communicate which tasks you have implemented, so current and future developers are aware.

Select the tasks that you feel best demonstrate your level of ability, or that best reflect your interests.

You may add any tool, framework, or technology to complete the task. The existing code may be modified in any way necessary to achieve your goal. However, you must ensure that all existing functionality remains intact. Please feel free to ask questions about the project with other members.

Submit the results of your three completed tasks as a Pull Request for the team to review.

Required Task

101. Associate locations in the system with a region so that JobbyJobs can easily segment job postings across a state. Modify the system so that when creating a location, you can specify the region the location is associated with, and show the name of that region on the location's show page. While it's not needed anywhere in the application, ensure it's easy to group all locations by region.

Available Tasks

201. When creating a new job posting, require the description to be set before saving.
202. Re-implement the front-end for viewing all jobs and creating a new job using a JS framework like React, Vue, Elm, or Angular.
203. Job searchers are looking to view jobs on their phone, and the current UI is lacking in supporting that. Update the UI for mobile display on the job posting index page. Focus only on this page.
204. New jobs are of particular interest to the client. Generate a CSV report for their consumption that lists all new jobs in the system. A new job is defined as any job that was created yesterday. This report should be generated automatically every day at 9am Eastern Time.
205. With so many jobs in the system, users are now experiencing poor loading times for the index page, and are also finding it difficult to find jobs available to them. Address those concerns by implementing the following:

```
* Display no more than 10 jobs at a time on the page, while still allowing access to all jobs in the system.
* Allow users to sort and/or filter the jobs by the status column.
```

206. Allow users to easily find jobs posted by a particular person by implementing a search feature, that will find all jobs posted by the user searched for, either by first name or last name.
207. Report at the top of the index page, above the listing of jobs, how many pending jobs there are in each category.

208. When viewing the details of a particular job, use the location to generate a map which clearly shows where the posting is for.
209. On the form to create a location, use Google Places or another API to validate and auto-complete the address as it is being typed.
210. Implement an authentication system using something like Devise or JWT and use it to ensure that users must log in before creating or editing a job posting.
211. When viewing the details of a particular job, link to other nearby jobs as well. A "nearby" job may be one that's in the same city, or any other heuristic of your choosing.
212. Allow a user looking at the list of job postings to specify a particular category of interest, perhaps through a dropdown list. Once that category is set, bold each row in the job postings list that is associated with that category.
213. Job posters would like to associate multiple categories to a given job posting. Update the application so that a job posting may be related to any number of categories. Display all categories for a given posting on the job postings listing page.

Getting Started

Install:

- pkg-config
- CMake
- Chromedriver

You can install these on OS X with homebrew by running:

```
brew bundle
```

The skeleton app contains enough dependencies to satisfy the current functionality, but feel free to add more.

```
$ bundle install
$ bundle exec rake db:create
$ bundle exec rake db:migrate
$ bundle exec rails s
$ open http://localhost:3000
```

Testing

The repo is configured with rspec, and includes tests in place for existing functionality.

To run the full test suite:

```
$ bundle exec rake
```

To run all tests in a specific file:

```
$ bundle exec rspec spec/models/job_posting_spec.rb
```

To run a specific test, or group of tests, within a file:

```
$ bundle exec rspec spec/models/job_posting_spec.rb:12
```

Chrome Headless

Capybara is configured to run with Chrome Headless as the driver. In order to run Chrome Headless, you must have Chrome installed and also ChromeDriver.

On OS X, you may install ChromeDriver via homebrew `brew install chromedriver`.

Switching out Capybara Driver

If you'd like to not run your tests headless, for example, to troubleshoot an issue and see what's on the screen, modify the `driven_by` driver in `spec/support/system_test_configuration.rb` to use `:selenium_chrome` instead of `:selenium_chrome_headless`. After the change, this block should look as follows:

```
  config.before(:each, type: :system, js: true) do
    driven_by :selenium_chrome
  end
```

JavaScript Framework Integration

React

Use React by adding `<%= javascript_pack_tag 'hello_react' %>` to the head of your layout file, like `app/views/layouts/application.html.erb`. All it does is render `<div>Hello React</div>` at the bottom of the page.

Angular

Use Angular by adding the following HTML markup to your view:

```
Loading...

<%= javascript_pack_tag 'hello_angular' %>
```

Elm

Use Elm by adding `<%= javascript_pack_tag "hello_elm" %>` to the head of your layout file, like `app/views/layouts/application.html.erb`. It will render "Hello Elm!" within the page.

Vue

Use Vue by adding `<%= javascript_pack_tag 'hello_vue' %>` to the head of your layout file, like `app/views/layouts/application.html.erb`. All it does is render `<div>Hello Vue</div>` at the bottom of the page.

The above code uses Vue without the compiler, which means you cannot use Vue to target elements in your existing html templates. You would need to always use single file components. To be able to target elements in your existing html/erb templates, comment out the above code and uncomment the below Add `<%= javascript_pack_tag 'hello_vue' %>` to your layout Then add this markup to your html template:

```


    {{message}}




import Vue from 'vue/dist/vue.esm'
import App from './app.vue'

document.addEventListener('DOMContentLoaded', () => {
   const app = new Vue({
     el: '#hello',
     data: {
       message: "Can you say hello?"
     },
     components: { App }
   })
 })
```