Backend Test Project - Job Board

Your task is to implement a simple Job Board backend API. Detailed specifications for the test project are provided below. We estimate that you will not need more than a single weekend at relaxed coding speed to implement it.

Project Description

The Job Board API will be used by your Users (Job Seekers) to perform the following tasks:

- List Job Posts
- Apply to Jobs

Every User (Job Seeker) will have their own job applications that have statuses that can be checked (seen, not seen). Job Posts are managed by other Users (Admin).

The Job Board app will be used by your Admin User to perform the following tasks:

- Manage A Job Post
- List Job Applications
- List Job Posts

Technical details

Your backend should be able to serve all kinds of clients (which you do not have to implement) - using a RESTful API.

The following technical requirements are placed on your implementation:

API

- Use Ruby (v2.3+) with Ruby on Rails framework (v5.0+)
- HTTP responses should follow best practices
- API should communicate with their clients using JSON data structures
- Implement authentication that would be the best for the clients by your opinion check devise_token_auth or JWT

Resource Permission

- Job Seekers are not authorized to manage the Job
- Admin Users are not authorized to destroy job applications and users.
- You can use either the cancancan or pundit gems to do this

Data Storage

- All data should be stored in a relational database, use Sqlite

Users

- Registrations should be done with email and password (Job Seekers)
- There should be one Admin account, which will manage Job Posts
- You should implement the following functionality:
- User Registration (Job Seekers only)
- User Login

Job Post data

- You should implement the following functionality:
- Job Post must have: title, description

- Create a new Job Post (Admin)
- Update or delete existing Job Post (Admin)
- List all Job Applications (Admin & Job Seeker)

Job Applications

- You should implement the following functionality:
- Job seekers apply to any job by creating a Job Application that will have a status of (Not Seen) by default
- When an Admin User views a Job Application this will change their status to (Seen)

Test / Rspec

- Your code should be tested using Rspec
- Models
- Controllers

Bonus task (NOT mandatory)

- Allow Users to add resumes and send them to be stored on AWS S3
- Send emails to the user (Job Seeker) once the application is seen (You can use MailTrap for this)
- Implement search for Jobs by title or creation date
- Add Expiry date to the job post so it won't apper to the job seeker after a specific date
- Use ElasticSearch to search for keywords in the Job Description
- Implement a simple client using a frontend framework of your choice that consumes your API
- Implement CI using Travis and use Coverall for code coverage and insert their badges into your readme file

Review process

There are a few technical restrictions so we can see how you fare with the technologies and processes we use on a daily basis, but in general, the actual implementation is quite open-ended. The reason is we want to see how you think in terms of backend architecture, development processes and how you generally deal with the challenges you might face while implementing this app.

The following should help you determine where to put your focus, since these are the things we will be looking for while reviewing your project.

Code quality

Is your code well-structured? Do you keep your coding style consistent across your codebase? You can check our style guide https://github.com/BoundlessDrop/style-guide

Security

How do you store your customers' passwords? What about security of your customers' data? how you are securing the API endpoints

Testability

Is your code tested? How do you approach testing? Do you use TDD or are tests an afterthought for you.

API structure and usability

How do you structure API endpoints? Do you follow REST principles? Do you make use of proper response codes and HTTP headers where it makes sense?

Validations and error handling

How do you handle required fields, and errors that might appear due to invalid data, How do you handle responses and Exceptions

Development and deployment

How hard is it to run your project locally? And how hard is it to deploy it?

Documentation

Is your API documented? Is your documentation auto generated from the code base? Does it cover all you endpoints?

Version Control

Please commit often and tell a story of your process with your commit history.

Project Delivery

- source code delivery options:
    - Create a Fork of this repository on Bitbucket and then create a pull request back to it
    - Create a private repository on bitbucket and invite devteam@boundlessdrop.com
- Deploy your app on Heroku and send the link to it

That's it. Good luck and we look forward to seeing your submission!

Useful Links to get you started

- http://edgeguides.rubyonrails.org/api_app.html
- https://github.com/plataformatec/devise
- https://github.com/lynndylanhurley/devise_token_auth
- http://www.justinweiss.com/articles/search-and-filter-rails-models-without-bloating-your-controller/
- https://scotch.io/tutorials/build-a-restful-json-api-with-rails-5-part-one
- https://scotch.io/tutorials/build-a-restful-json-api-with-rails-5-part-two
- https://rspec.info/
- https://mailtrap.io/
- https://github.com/aws/aws-sdk-ruby
- https://github.com/CanCanCommunity/cancancan