Mini Project - Easily

Score: 0/500

Objective

Send feedback

Goal

Develop a job portal website that allows recruiters to post and manage job listings and provides a user-friendly platform for job seekers to find and apply for suitable roles.

Acceptance Criteria

1. Implement an MVC (Model View Controller) architecture with ExpressJS to separate data handling, interface rendering, and routing control.

2. Implement EJS for server-side templating, providing dynamic HTML generation based on server data.

3. Utilize ES6 Modules to maintain code modularity and organization.

4. Use Express sessions for managing user sessions and a cookie-based tracking of the last visit.

5. Use in-memory data structures for user and job management operations.

6. Implement a login and registration system, allowing recruiters to create and log into their accounts.

7. Allow job seekers to view all jobs, view details of a specific job, and apply to a job by providing their details.

8. Allow recruiters to create, update, delete, and view job postings, with necessary validation for each field in the job posting.

9. Enable recruiters to view all applicants of a job, including their submitted resume files.

10. Implement an email system to send confirmation emails to applicants after they apply to a job.

11. Use middleware for handling authentication, tracking of the last visit, file upload processing, and sending confirmation emails.

12. Store the resume files on the server using a file upload middleware.

13. Ensure original and high-quality code with comprehensive documentation.

Steps

Follow the following steps to build the project.

1. Set up an Express.js application and its related configurations.

2. Install necessary project dependencies based on the functionalities required.

3. Configure EJS as the templating engine and create views for job seekers to browse all jobs, view details of a job, apply to jobs, and recruiters to create, update, delete, and view their job postings.

4. Create a User model with functions for getting all the users, adding a user, and confirming user login.

5. Create a User controller to interface with the User model and handle user registration, login, and logout.

6. Create a Job model with functions for creating a new job, retrieving all jobs, finding a job by its id, updating a job, adding a new applicant to a job, retrieving all applicants for a job, and deleting a job.

7. Create a Job controller to interface with the Job model to create, update and delete jobs, and to manage job applicants.

8. Implement routes for user registration, login, logout, job operations including creating, updating, and deleting job postings, and to fetch necessary views.

9. Implement a session based user authentication and management system for recruiters.

Hints

Component Structure

- **Layouts Page:** A common layout view that includes the header, main content, and footer.
- **Landing Page:** Showcases a welcome message and provides a brief overview of the job portal.
- **Job Listings Page:** Displays all available job postings.
- **Job Details Page:** Shows detailed information about a specific job.
- **Applicant List Page:** Displays a list of applicants for a specific job.
- **Login Page:** Allows recruiters to log in to their accounts.
- **New Job Page:** Form for recruiters to create a new job posting.
- **Update Job Page:** Form for recruiters to update an existing job posting.
- **404 Page:** Displayed when a requested page is not accessible.

Data Structures

User

- `id`: Unique identifier for each user.
- `name`: Name of the user.
- `email`: Email address of the user.
- `password`: Password for user authentication.

Job

- `id`: Unique identifier for each job.
- `jobcategory`: Category or type of the job.
- `jobdesignation`: Designation of the job.
- `joblocation`: Location of the job.
- `companyname`: Name of the company offering the job.
- `salary`: Salary range for the job.
- `applyby`: Application deadline for the job.
- `skillsrequired`: Array of required skills for the job.
- `numberofopenings`: Number of available openings for the job.
- `jobposted`: Date and time when the job was posted.
- `applicants`: Array of applicants who have applied for the job, each with their own attributes.

Applicant

- `applicantId`: Unique identifier for each applicant.
- `name`: Name of the applicant.
- `email`: Email address of the applicant.
- `contact`: Contact number of the applicant.
- `resumePath`: Path to the applicant's resume file.

Additional Info for Reference

- **Resource-Based Authentication:** This can be achieved by associating each job with the recruiter's ID and checking the authorization before allowing any modifications to the job. Learn more: link
- **Pagination:** Instead of displaying all job listings or applicants on a single page, divide them into smaller pages and provide navigation to move between pages. This helps to load and display data more efficiently.
- **REST API Best Practices:** Follow REST API best practices to design your API endpoints. Use appropriate HTTP methods (GET, POST, PUT, DELETE) for different types of operations. Learn more: link

API Structure

The API structure for the "Easily" job portal project can be organized as follows:

Auth routes

- **POST /register:** Register a new recruiter account
- **GET /:** Render the login page
- **POST /login:** Log in as a recruiter
- **POST /logout:** Log out the currently logged-in recruiter

Job routes

- **/jobs**
  - **GET /:** Retrieve all job listings
  - **POST /:** Create a new job listing
  - **GET /id:** Retrieve a specific job listing by ID
  - **PUT /id:** Update a specific job listing by ID
  - **DELETE /id:** Delete a specific job listing by ID
- **/jobs/id/applicants**
  - **GET /:** Retrieve all applicants for a specific job listing
  - **POST /:** Add a new applicant to a specific job listing
  - **GET /applicantId:** Retrieve a specific applicant by ID for a job listing
  - **PUT /applicantId:** Update a specific applicant by ID for a job listing
  - **DELETE /applicantId:** Delete a specific applicant by ID for a job listing
- **/jobs/id/update**
  - **GET /:** Render the update form for a specific job listing
  - **POST /:** Update a specific job listing by ID
- **/jobs/id/delete**
  - **GET /:** Delete a specific job listing by ID
- **/apply/id**
  - **POST /:** Apply to a specific job listing by ID, uploading a resume

Step 1
Complete project on local IDE

Step 2
Export code as .zip file

Step 3
Upload .zip file max 50mb

Evaluation criteria	Score
ExpressJS & MVC Setup	50
Route Creation	100

Drag and drop .zip here or browse

Submit

Evaluation criteria

Evaluation parameters

Ask a doubt