



PREMIER UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A Project Report On
JOB BOARD PLATFORM SYSTEM

Course Title: Software Development

Course Code: CSE 364

Submitted To:

Tashin Hossain

Lecturer

Department of Computer Science and Engineering

Premier University

Submitted By:

Aysha Siddika Marua (2104010202229)

Anik Barua (2104010202230)

Pankaj Rudra (2104010202242)

Mohammaod Rabbe Islam Jewel (2104010202246)

24th October, 2024

TABLE OF CONTENTS

TITLE PAGE	i
TABLE OF CONTENTS	iii
LIST OF FIGURES	v
LIST OF TABLES	1
1 Introduction	2
2 Project Organization	3
3 Hardware and Software Requirements	4
3.1 Hardware Requirements	4
3.2 Software Requirements	4
4 System Architecture	5
4.1 Model	5
4.2 View	5
4.3 Controller	5
5 Class Diagram of the System	6
6 Project Demonstration	7
6.1 User Registration and Login	8
6.1.1 Registration:	8
6.1.2 Login:	9
6.1.3 Login User Interface:	10
6.1.4 Registration User Interface:	11
6.2 Job Posting as an Employer	11
6.2.1 Job Posting:	12
6.3 3. Job Searching	12
6.4 4. Admin Controls	13
6.5 5. Job Listing and Deleting	15
6.6 6. User Job Application	17
6.7 7. Employer Can Approve Applications	19
6.8 8. Job Saving to "My Jobs"	19
6.9 9. Profile Management	20

7	Testing	22
7.1	Unit Testing	22
7.2	Integration Testing	22
7.3	User Testing	23
8	Bug Reporting	24
8.1	Popular Categories Listing	24
9	Limitations	25
	REFERENCES	26

List of Figures

5.1	Class diagram of the <i>CareerConnect</i> System Database	6
6.1	Flowchart diagram of Job Board Platform System	7
6.2	Login User Interface	10
6.3	Registration User Interface	11
6.4	Job Posting Interface	12
6.5	Find Jobs User Interface	13
6.6	Admin Page User Interface	13
6.7	Admin Controls User Interface	14
6.8	Admin Jobs Control Interface	14
6.9	Admin Job Applications Control Interface	14
6.10	Job Listing Page User Interface	15
6.11	Job Listing in Homepage User Interface	16
6.12	Job Listing Employer Interface	16
6.13	Deleting Jobs User Interface	17
6.14	Job Application Page User Interface	18
6.15	Applied Jobs Page User Interface	19
6.16	Applicants List Page	19
6.17	Saving Jobs Button	20
6.18	Saved Jobs List Page UI	20
6.19	User Profile Page	21
6.20	Profile Account Password Change UI	22

7.1	Email Validation Testing	23
7.2	Password Validation Testing	23
7.3	Confirm Password Matching Test	24
8.1	Screenshot of the Popular Categories section	25

List of Tables

1. Introduction

The rapid expansion of digital technology has transformed industries globally, and one of the areas most significantly impacted is the job market. With the increasing reliance on online platforms for employment opportunities, there is a growing need for efficient, user-friendly systems that bridge the gap between employers and job seekers. The CareerConnect project was conceived to address these demands by providing an integrated platform where users can search for jobs, apply directly, and employers can post jobs, manage applications, and approve candidates. This project aims to offer a seamless solution to facilitate employment opportunities and enhance user experience in job recruitment.

The motivation behind choosing this project lies in the current challenges faced by both job seekers and employers. Job seekers often struggle to find relevant opportunities or manage their applications, while employers need tools that simplify job posting, application management, and candidate selection. The CareerConnect system is designed to mitigate these issues by providing a structured platform that organizes job postings into categories, allows for detailed job descriptions, and offers features like job saving and profile management for users.

This project holds relevance within the broader context of human resource management and digital recruitment systems. By streamlining the process for both employers and job seekers, the platform addresses a critical need in today's job market. Furthermore, the system incorporates role-based access controls for admin and user functions, making it highly scalable for businesses of various sizes.

The scope of CareerConnect includes implementing features such as user registration and login, job posting, job searching, job applications, and a profile management system. Through these features, the project seeks to create an intuitive and secure platform for employment processes. The ultimate goal is to develop a practical, robust system that benefits both ends of the recruitment spectrum.

Project on GitHub[1]

2. Project Organization

The project was developed by a team of four members:

- **Member 1: Aysha Siddika Marua (Database Design)**
Created the database schema and managed all interactions with the database, ensuring that data is stored and retrieved efficiently.
- **Member 2: Anik Barua (Frontend Development)**
Responsible for designing the user interface using HTML, CSS, and Bootstrap.
- **Member 3: Pankaj Rudra (Backend Integration)**
Managed the logic and functionality of the system using the Laravel framework and PHP.
- **Member 4: Mohammad Rabbe Islam Jewel (Testing and Debugging)**
Tested the system to ensure that all features worked correctly and fixed any bugs or issues.

Each team member worked collaboratively to integrate their contributions into a cohesive platform, ensuring that the system met all requirements and provided a smooth user experience.

3. Hardware and Software Requirements

3.1. Hardware Requirements

- **Processor:** Minimum 2.0 GHz dual-core processor
- **RAM:** Minimum 4 GB
- **Storage:** Minimum 20 GB free disk space
- **Network:** Internet connection for deployment and testing

3.2. Software Requirements

- **Operating System:** Windows
- **Web Server:** Apache
- **Database:** MySQL
- **Programming Language:** PHP (Laravel framework)
- **Frontend Technologies:** HTML, CSS, JavaScript (Bootstrap)
- **Editor:** Visual Studio Code

4. System Architecture

CareerConnect follows the **Model-View-Controller (MVC)** architecture, which is widely used in modern web development projects. This architecture divides the project into three interconnected components, each responsible for a specific part of the system. The use of MVC ensures that the project is well-structured, modular, and easy to maintain. Below is a detailed description of each component in the CareerConnect system.

4.1. Model

The *Model* represents the database and data handling logic. It is responsible for managing the data, including retrieving, saving, and updating records in the database. In CareerConnect, models such as `Job`, `User`, and `Application` correspond to the database tables for jobs, users, and job applications, respectively. These models define the relationships and operations that are performed on the data.

4.2. View

The *View* is responsible for presenting data to the user. It controls the layout and structure of the web pages that users interact with. In CareerConnect, views are generated using Blade templates, which are a part of the Laravel framework. These views display dynamic data such as job listings, application forms, and user profiles. Blade templates allow us to create reusable and well-organized HTML code, making it easier to manage the user interface.

4.3. Controller

The *Controller* acts as an intermediary between the Model and the View. It handles user input, processes requests, interacts with the Model to retrieve or update data, and returns the appropriate View to the user. For example, the `JobController` manages operations related to job postings, including listing jobs, adding new job postings, and editing existing jobs. Controllers ensure that the application's logic is decoupled from both the data and the presentation layer.

The MVC architecture provides a clear separation of concerns, allowing each part of the system to focus on its specific role. This approach leads to better maintainability, scalability, and organization of the overall project.

5. Class Diagram of the System

The class diagram of the *CareerConnect* system illustrates how different parts of the system are related to each other. In this system, several core classes are defined, such as User, Job, Application, Category, and JobType. Each of these classes encapsulates specific attributes and methods maintaining relationships to handle various operations within the platform.

- **User:** Contains attributes like name, email, password, role, and methods for user registration, authentication, and profile management.
- **Job:** Includes attributes such as title, category_id, salary, location, and methods to manage job postings (e.g., creating, updating, deleting jobs).
- **Application:** Manages attributes like job_id, user_id, employer_id, and handles methods related to job applications (e.g., submitting, viewing, and approving applications).
- **Category:** Holds attributes such as name and status, with methods to add, update, or delete job categories.
- **JobType:** Contains attributes like name and methods for managing the type of job (e.g., full-time, part-time, contract).

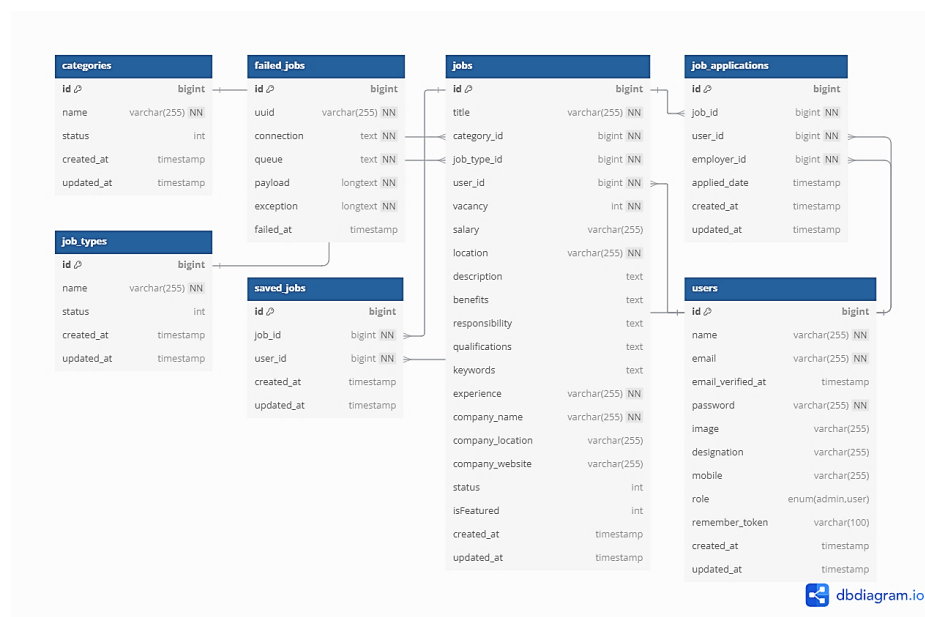


Figure 5.1: Class diagram of the *CareerConnect* System Database

6. Project Demonstration

The flowchart diagram of the system is given below:



Figure 6.1: Flowchart diagram of Job Board Platform System

CareerConnect has several important features that make it useful for both job seekers and employers. Below are the main functionalities:

6.1. User Registration and Login

6.1.1. Registration:

- Users register by submitting a form that includes their details such as name, email, and password. The registration logic is handled by the Auth controller in Laravel, which automatically hashes the password before saving it to the users table.

```
app > Http > Controllers > AccountController.php
16 use Illuminate\Support\Facades\Mail;
17 use Illuminate\Support\Facades\Validator;
18 use Intervention\Image\ImageManager;
19 use Intervention\Image\Drivers\Gd\Driver;
20 use Illuminate\Support\Str;
21
22 class AccountController extends Controller
23 {
24     // This method will show user registration page
25     public function registration() {
26         return view('front.account.registration');
27     }
28
29     // This method will save a user
30     public function processRegistration(Request $request) {
31         $validator = Validator::make($request->all(), [
32             'name' => 'required',
33             'email' => 'required|email|unique:users,email',
34             'password' => 'required|min:5|same:confirm_password',
35             'confirm_password' => 'required',
36         ]);
37
38         if ($validator->passes()) {
```

- Upon successful registration, the user is either assigned a user or admin role. The role is saved as part of the users table in the role column.

```

database > migrations > 2024_02_05_194735_alter_users_table.php
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       */
12     public function up(): void
13     {
14         Schema::table('users', function (Blueprint $table) {
15             $table->enum('role', ['admin', 'user'])->default('user')->after('mobile');
16         });
17     }
18
19     /**
20      * Reverse the migrations.
21      */
22     public function down(): void
23     {
24         Schema::table('users', function (Blueprint $table) {
25             $table->dropColumn('role');

```

6.1.2. Login:

- Users log in by providing their email and password. The Auth controller checks the credentials against the users table. If the credentials match, the user is authenticated and logged in.

```

app > Http > Middleware > Authenticate.php
1  <?php
2
3  namespace App\Http\Middleware;
4
5  use Illuminate\Auth\Middleware\Authenticate as Middleware;
6  use Illuminate\Http\Request;
7
8  class Authenticate extends Middleware
9  {
10     /**
11      * Get the path the user should be redirected to when they are not authenticated.
12      */
13     protected function redirectTo(Request $request): ?string
14     {
15         return $request->expectsJson() ? null : route('account.login');
16     }
17 }
18

```

- Based on the role, the user is redirected to either the admin or user dashboard. Middleware is used to differentiate between roles and control access to specific sections of the site.

```

app > Http > Middleware > CheckAdmin.php
1
2
3
4
5 use Closure;
6 use Illuminate\Http\Request;
7 use Symfony\Component\HttpFoundation\Response;
8
9 class CheckAdmin
10 {
11     /**
12      * Handle an incoming request.
13      *
14      * @param \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response) $next
15      */
16     public function handle(Request $request, Closure $next): Response
17     {
18         if ($request->user() == null) {
19             return redirect()->route('home');
20         }
21
22         if ($request->user()->role != 'admin') {
23             session()->flash('error', 'You are not authorized to access this page.');
```

6.1.3. Login User Interface:

CareerConnect Home Find Jobs [Login](#) [Post a Job](#)

Login

Email*

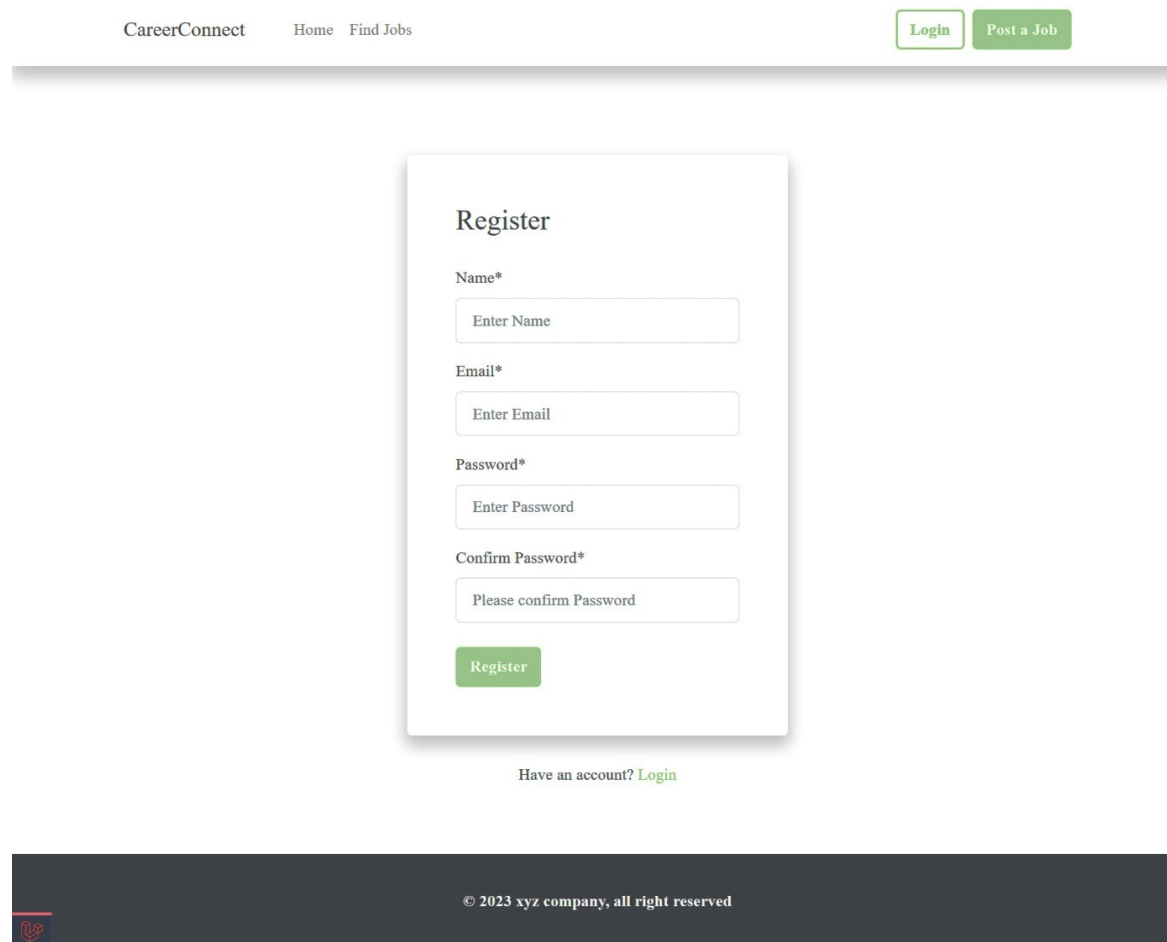
Password*

[Login](#) [Forgot Password?](#)

Do not have an account? [Register](#)

Figure 6.2: Login User Interface

6.1.4. Registration User Interface:



The image shows a web application interface for registration. At the top, there is a navigation bar with the text "CareerConnect" on the left and "Home Find Jobs" in the center. On the right side of the navigation bar, there are two green buttons: "Login" and "Post a Job". Below the navigation bar, there is a central registration form titled "Register". The form contains four input fields: "Name*" with the placeholder "Enter Name", "Email*" with the placeholder "Enter Email", "Password*" with the placeholder "Enter Password", and "Confirm Password*" with the placeholder "Please confirm Password". Below these fields is a green "Register" button. At the bottom of the form, there is a link that says "Have an account? Login". The footer of the page is a dark gray bar with the text "© 2023 xyz company, all right reserved" and a small logo on the left.

Figure 6.3: Registration User Interface

6.2. Job Posting as an Employer

- Employers can post jobs by submitting a form that includes job details such as job title, category, job type, salary, location, description, and company details.
- The form data is sent to the JobController where the job details are validated and saved to the jobs table. The user_id (employer's ID) is also stored to link the job with the employer who posted it.
- The job status is set to "active" (status = 1) by default, and employers can later edit or delete the job listing if needed.


```

resources > views > front > account > job > create.blade.php > script > <function> > success
20 <div class="col-lg-9">
21     @include('front.message')
22
23     <form action="" method="post" id="createJobForm" name="createJobForm">
24         <div class="card border-0 shadow mb-4">
25             <div class="card-body card-form p-4">
26                 <h3 class="fs-4 mb-1">Job Details</h3>
27                 <div class="row">
28                     <div class="col-md-6 mb-4">
29                         <label for="" class="mb-2">Title<span class="req">*</span></label>
30                         <input type="text" placeholder="Job Title" id="title" name="title" class="form-control">
31                         <p></p>
32                     </div>
33                     <div class="col-md-6 mb-4">
34                         <label for="" class="mb-2">Category<span class="req">*</span></label>
35                         <select name="category" id="category" class="form-control">
36                             <option value="">Select a Category</option>
37                             @if ($categories->isEmpty())
38                                 @foreach ($categories as $category)
39                                     <option value="{{ $category->id }}">{{ $category->name }}</option>
40                                 @endforeach
41                             @endif
42                         </select>

```

6.2.1. Job Posting:

The screenshot displays the 'Job Posting Interface' on the CareerConnect website. At the top, there's a navigation bar with 'CareerConnect', 'Home', 'Find Jobs', and a user greeting 'Hi, Admin' with links for 'Admin', 'My Account', and 'Post a Job'. Below this, the breadcrumb 'Home / Account Settings' is visible. The main content area features a user profile for 'Anik Barua, Employer' with a 'Change Profile Picture' button. To the right of the profile is the 'Job Details' form, which contains four required fields: 'Title', 'Category', 'Job Type', and 'Vacancy'. Each field has a corresponding dropdown menu for selection.

Figure 6.4: Job Posting Interface

6.3. 3. Job Searching

- Job seekers can search for jobs using keywords, location, and categories. The Job-Controller fetches jobs from the jobs table based on the search parameters.
- SQL queries are dynamically built depending on the search inputs, allowing users to filter job results.
- The results are then displayed in the view, showing jobs that match the search criteria.

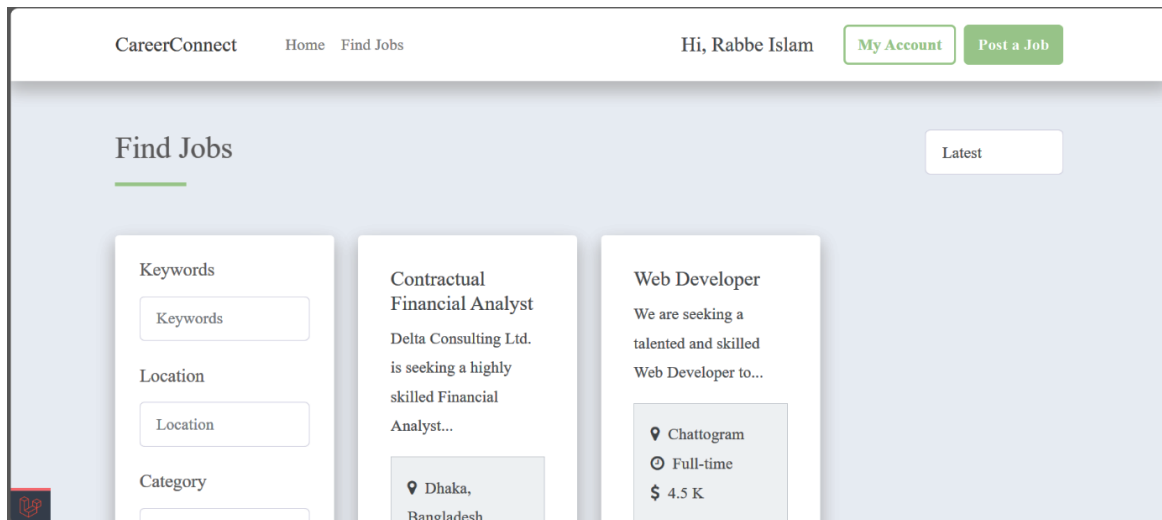


Figure 6.5: Find Jobs User Interface

6.4. 4. Admin Controls

- Admins have the ability to manage job categories by adding, editing, or deleting them. This is handled through the CategoryController.
- When an admin adds a new category, the category name is saved to the categories table with a default active status (status = 1).
- Admins can delete categories, and any jobs associated with that category are deleted as well due to the foreign key constraint (ON DELETE CASCADE) between jobs and categories.
- Admins also can manage users and their profiles, job listing handling, job application management, employer management, etc.

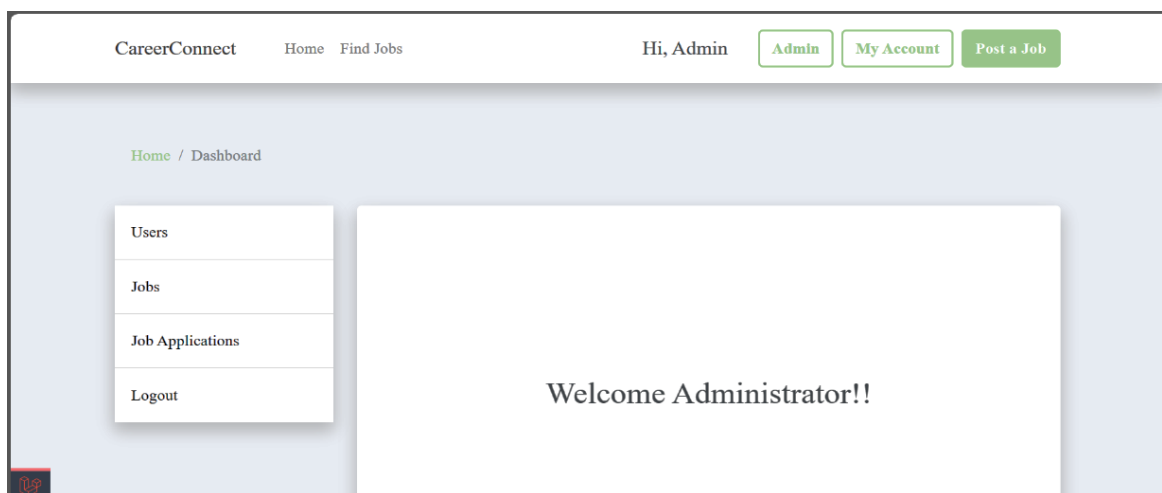


Figure 6.6: Admin Page User Interface

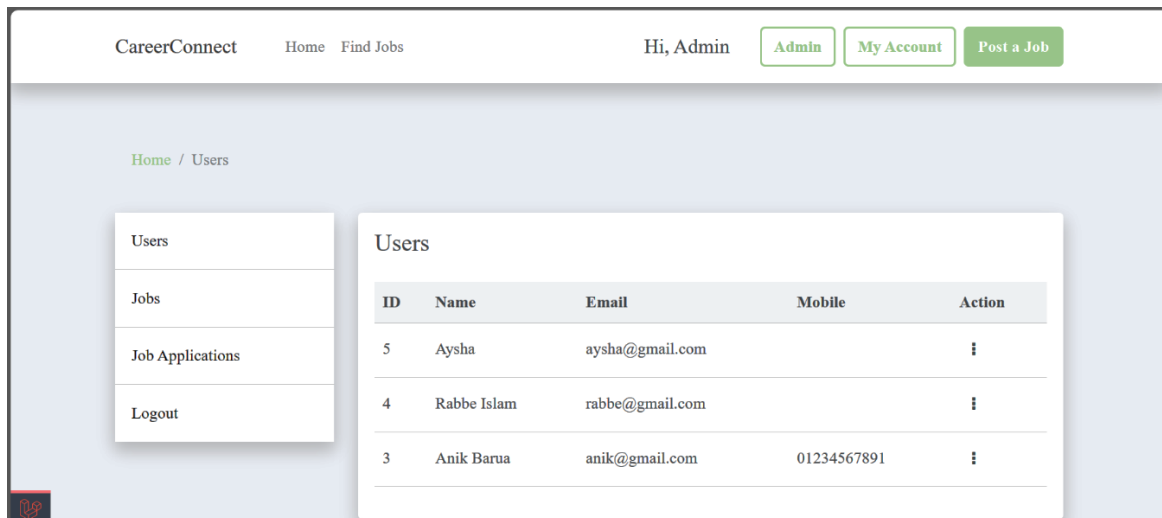


Figure 6.7: Admin Controls User Interface

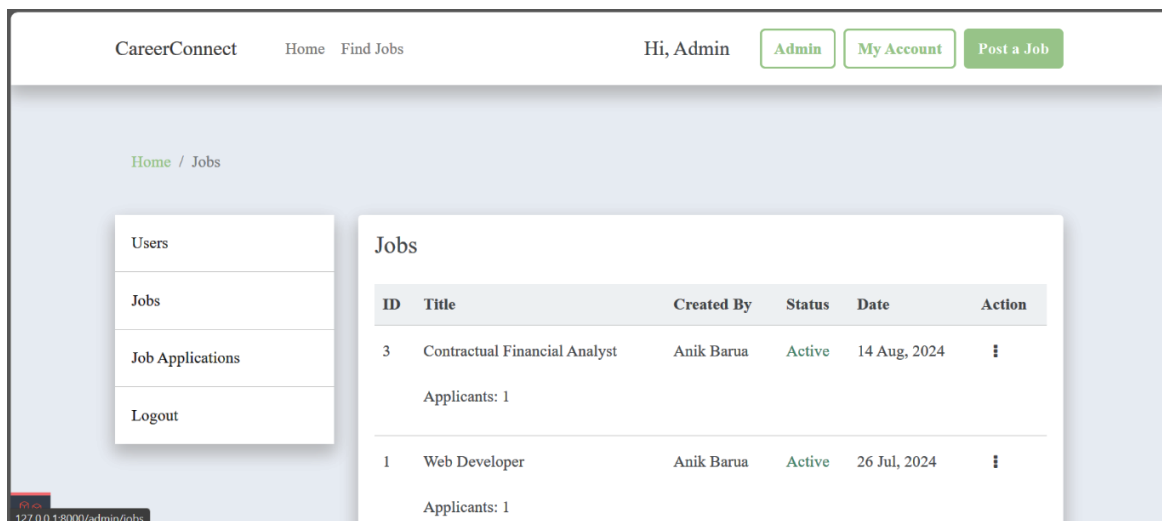


Figure 6.8: Admin Jobs Control Interface

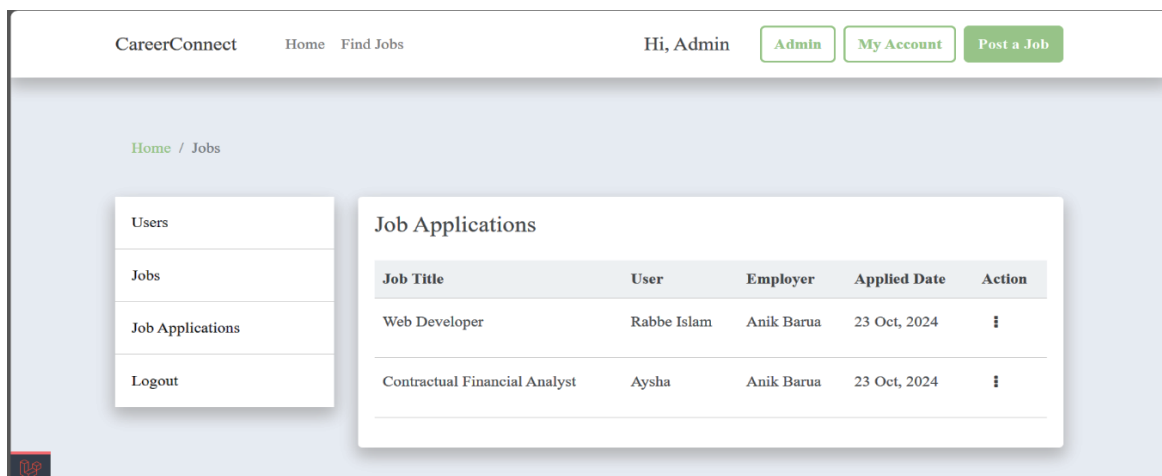


Figure 6.9: Admin Job Applications Control Interface

6.5. 5. Job Listing and Deleting

- Both employers and job seekers can view job listings. The job listings are fetched from the jobs table and displayed to the user based on their status (status = 1 means active).
- Employers can delete jobs they have posted. When a job is deleted, the corresponding record in the jobs table is removed using the destroy method in the JobController.

```
app > Models > Job.php
1 namespace App\Models;
2
3
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Job extends Model
9 {
10     use HasFactory;
11
12     public function jobType() {
13         return $this->belongsTo(JobType::class);
14     }
15
16     public function category() {
17         return $this->belongsTo(Category::class);
18     }
19
20     public function applications() {
21         return $this->hasMany(JobApplication::class);
22     }
23
24     public function user() {
25         return $this->belongsTo(User::class);
26     }
27 }
```

Figure 6.10: Job Listing Page User Interface

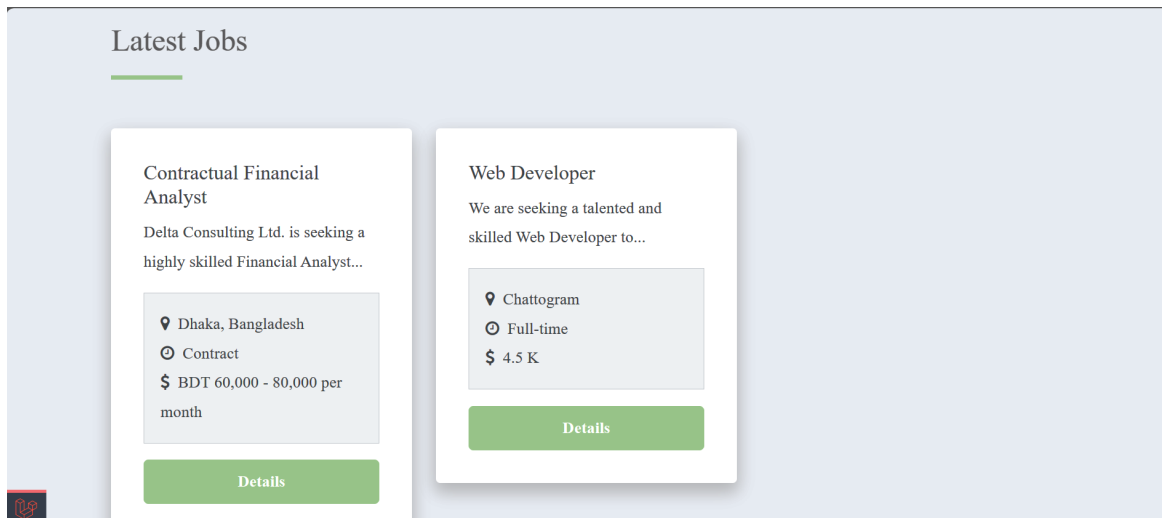


Figure 6.11: Job Listing in Homepage User Interface

- Job Listing User Interface:

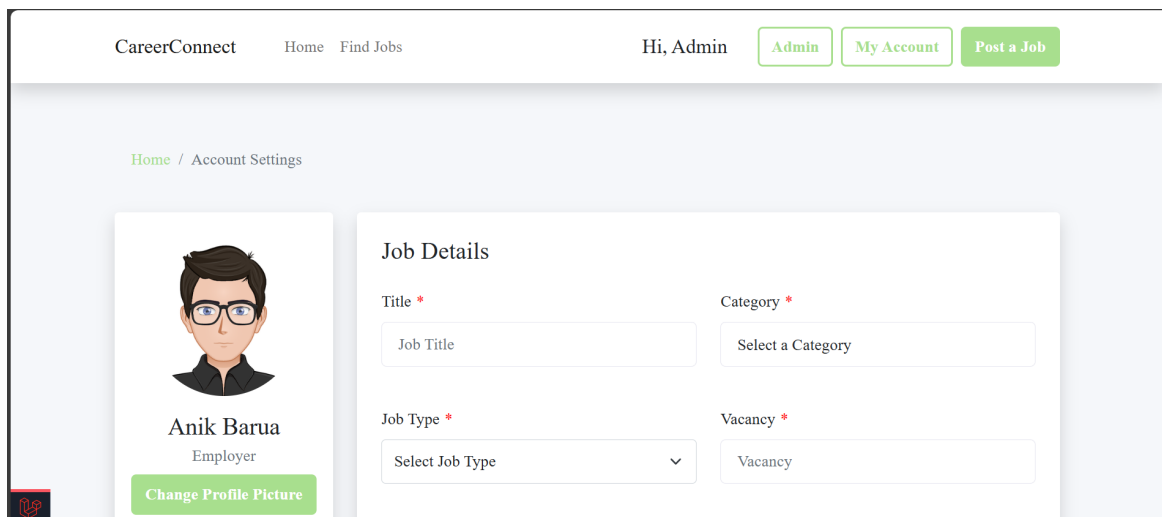


Figure 6.12: Job Listing Employer Interface

- Deleting Jobs:

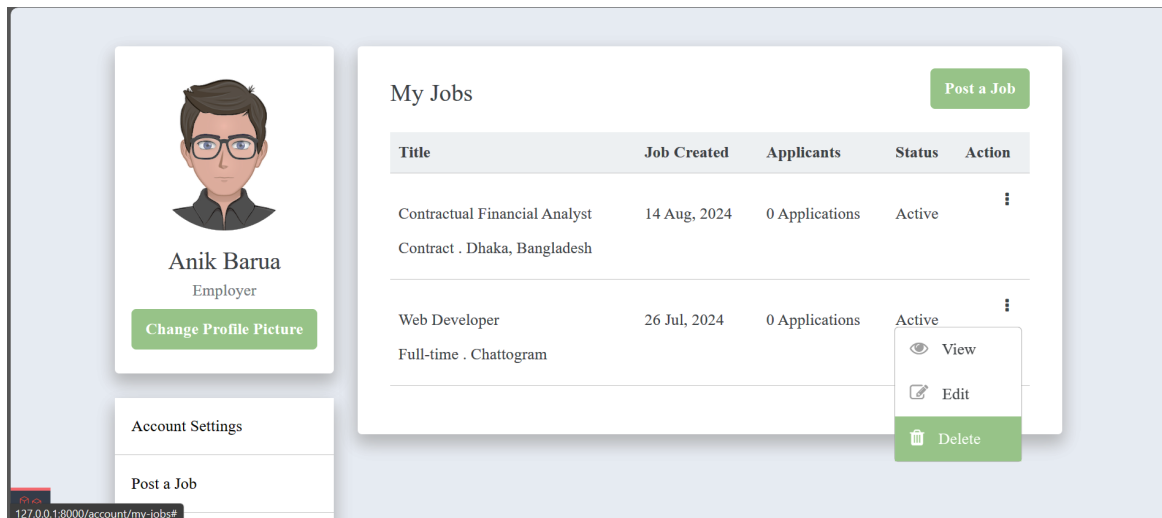


Figure 6.13: Deleting Jobs User Interface

6.6. 6. User Job Application

- When a job seeker applies for a job, the job ID and user ID are sent to the ApplicationController, which handles storing the application data in the job_applications table.
- The application data includes the job_id, user_id, and the current timestamp (applied_date). This creates a link between the user and the job they are applying for.
- Users can track their job applications, and employers can view all applications for their posted jobs.

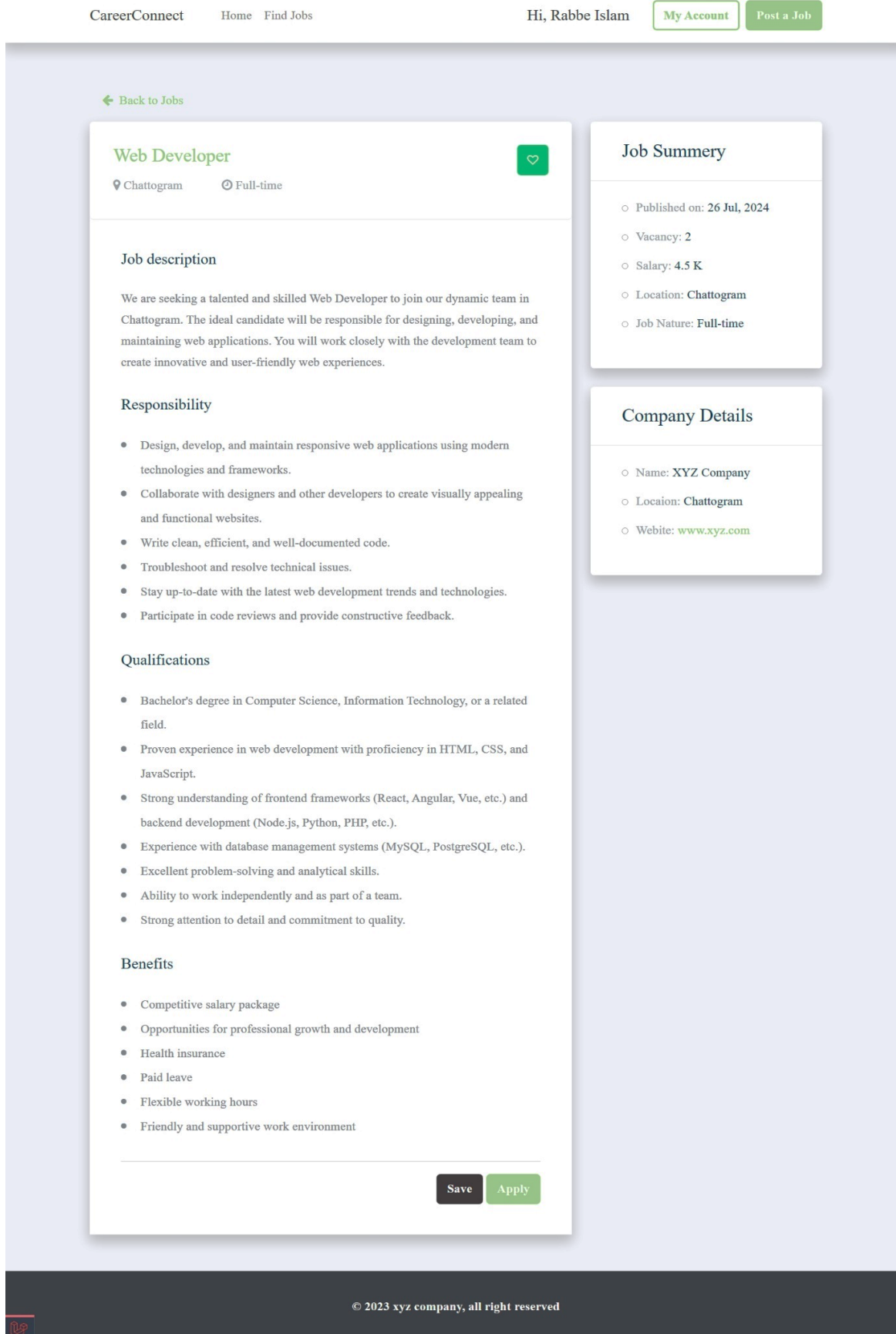


Figure 6.14: Job Application Page User Interface

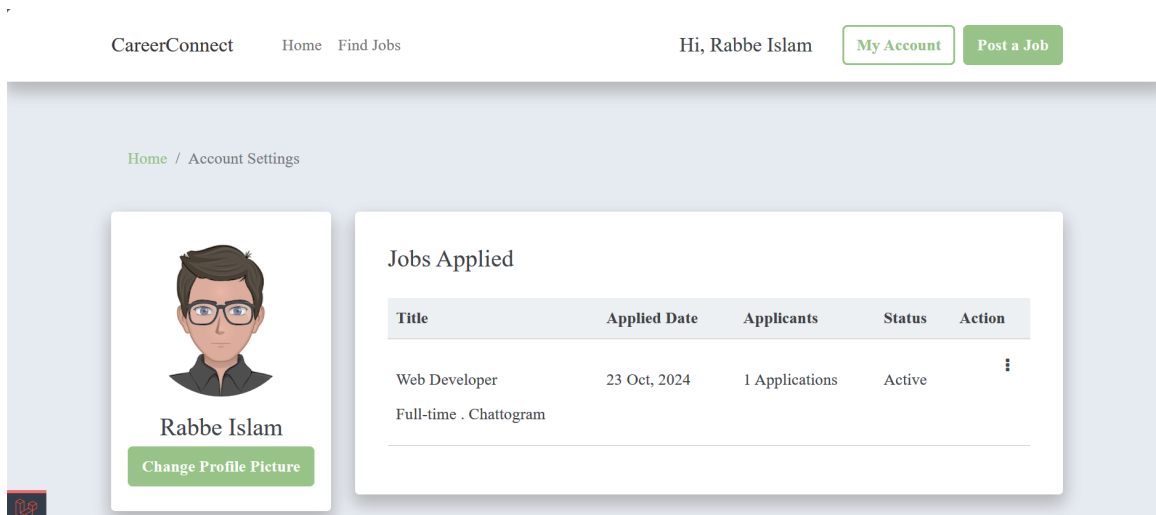


Figure 6.15: Applied Jobs Page User Interface

6.7. 7. Employer Can Approve Applications

- Employers can view job applications submitted by job seekers for their job postings. Each application is listed with the job seeker's details.
- Employers can approve or reject applications by updating the status of the application in the job_applications table. This action is handled by the ApplicationController.

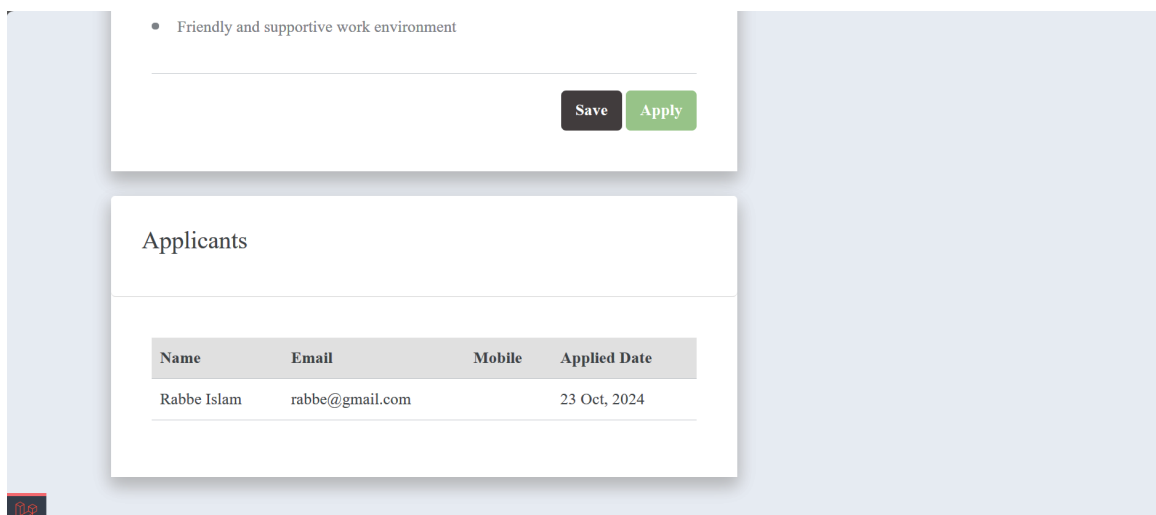


Figure 6.16: Applicants List Page

6.8. 8. Job Saving to "My Jobs"

- Job seekers can save jobs they are interested in applying to later. This is handled by the SavedJobController.

- When a job is saved, the job ID and user ID are stored in the saved_jobs table. This creates a reference between the user and the job.
- Saved jobs are displayed on the "My Jobs" page, where users can review and apply for them later.

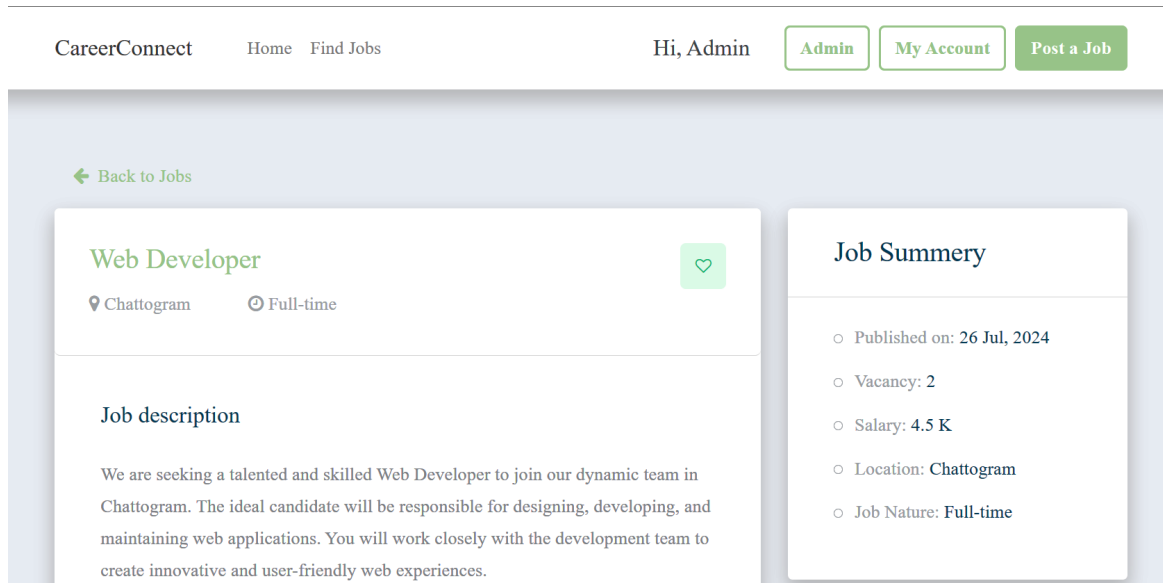


Figure 6.17: Saving Jobs Button

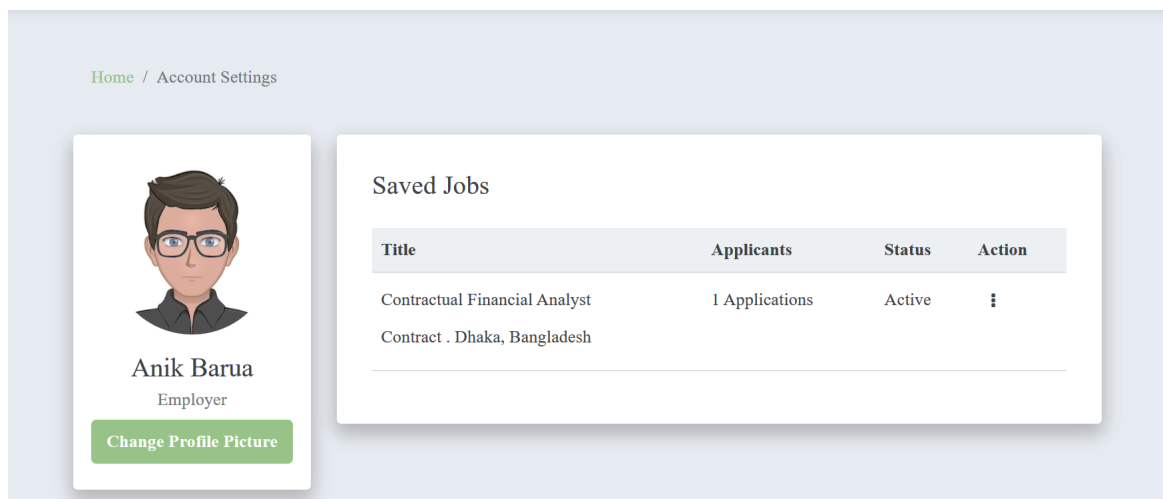


Figure 6.18: Saved Jobs List Page UI

6.9. 9. Profile Management

- Users (job seekers and employers) can manage their profiles by updating their personal information such as name, email, mobile number, and profile image.

- The UserController handles profile updates. The updated data is validated and stored in the users table.
- Profile images are handled by storing the file path in the image column of the users table. Users can also change their passwords through this section.

CareerConnect

HomeFind Jobs


Hi, Admin

Admin

My Account

Post a Job

Home / Account Settings



Anik Barua

Employer

Change Profile Picture

Account Settings

Post a Job

My Jobs

Jobs Applied

Saved Jobs

Logout

My Profile

Name*

Anik Barua

Email*

anik@gmail.com

Designation

Employer

Mobile

01234567891

Update

Change Password

Old Password*

Figure 6.19: User Profile Page

The image shows a web form titled "Change Password". It contains three input fields: "Old Password*" with a blue background and masked text, "New Password*" with a white background and placeholder text, and "Confirm Password*" with a white background and placeholder text. A green "Update" button is located at the bottom left of the form area.

Figure 6.20: Profile Account Password Change UI

7. Testing

To ensure that CareerConnect works properly, different types of testing were carried out:

7.1. Unit Testing

Each component or function was tested individually to ensure that it works as expected. For example, we tested the apply function to make sure it correctly adds a job application to the database.

7.2. Integration Testing

After testing each part individually, we tested how different parts of the system work together. For example, we tested if the job posting feature properly links to the job listing page and if the application submission is correctly saved in the database.

7.3. User Testing

We invited users to test the system. This helped us get feedback on the user interface and find any issues related to usability.

- **Email Validation Testing**

The figure displays two versions of a login form side-by-side to demonstrate email validation testing.

Left Form (Invalid Email):

- Title: Login
- Email*: Input field containing 'rabbegmail.com'. The field has a red border and a red error icon (exclamation mark in a circle).
- Error Message: 'The email field must be a valid email address.'
- Password*: Input field containing 'Enter Password'.
- Buttons: 'Login' (green) and 'Forgot Password?' (green).
- Footer: 'Do not have an account? [Register](#)'

Right Form (Valid Email):

- Title: Login
- Email*: Input field containing 'rabbe@gmail.com'. The field has a blue border.
- Password*: Input field containing '....'.
- Buttons: 'Login' (green) and 'Forgot Password?' (green).
- Footer: 'Do not have an account? [Register](#)'

Figure 7.1: Email Validation Testing

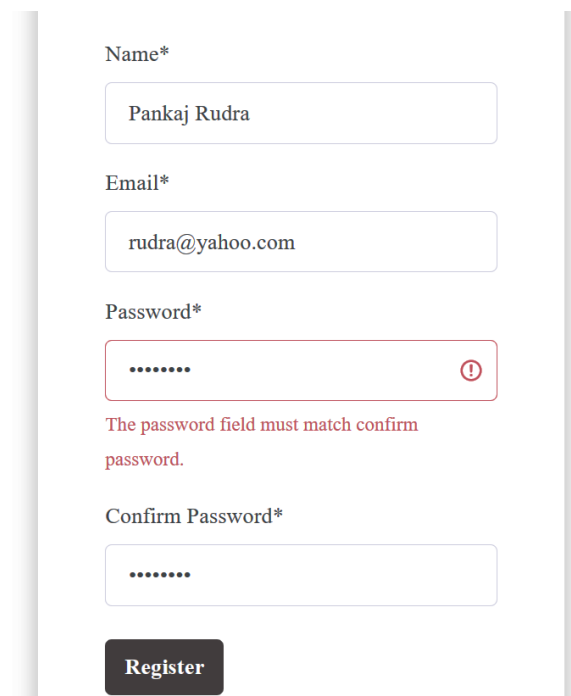
- **Password Validation Testing**

The figure shows a registration form with the following fields and elements:

- Name*: Input field containing 'Pankaj Rudra'.
- Email*: Input field containing 'rudra@yahoo.com'.
- Password*: Input field containing '...'. The field has a red border and a red error icon (exclamation mark in a circle).
- Error Message: 'The password field must be at least 5 characters.'
- Confirm Password*: Input field containing '...'.
- Button: 'Register' (dark grey).

Figure 7.2: Password Validation Testing

- **Confirm Password Matching Test**



Name*

Pankaj Rudra

Email*

rudra@yahoo.com

Password*

.....

The password field must match confirm password.

Confirm Password*

.....

Register

Figure 7.3: Confirm Password Matching Test

8. Bug Reporting

8.1. Popular Categories Listing

- On the index page, the Popular Categories section doesn't show the actual popular categories.
- The available position or vacancy counts are not showing from the Job database table.

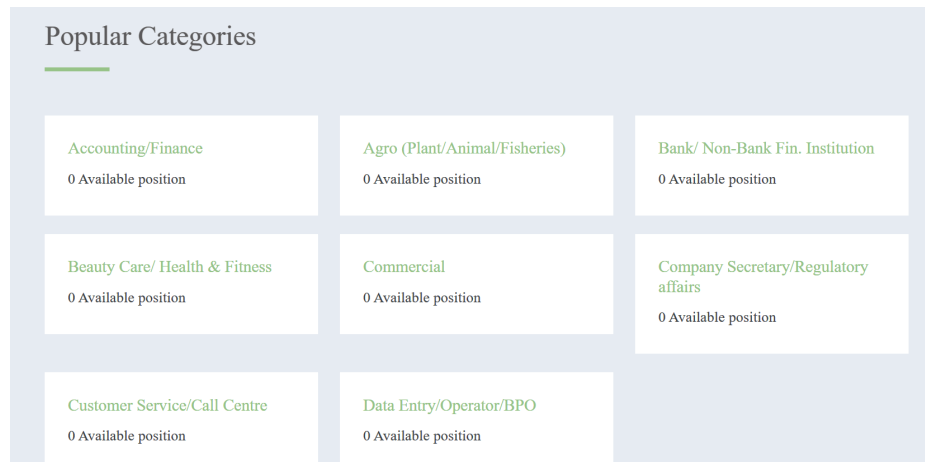


Figure 8.1: Screenshot of the Popular Categories section

9. Limitations

CareerConnect is a functional job portal; however, there are some areas that could be improved such as:

- **Scalability:** The current system can handle a moderate number of users. However, if the number of users grows significantly, we may need to make changes to the database and hosting to support more users.
- **Resume Upload Feature:** Currently, users cannot upload resumes directly through the platform. This feature could be added in the future to improve the application process for job seekers.
- **Advanced Search Filters:** At this stage, users can search for jobs using basic keywords. In the future, adding more advanced filters, such as location and salary range, could improve the user experience.
- **Performance:** As more job postings are added, the system might experience slower load times. We would need to optimize the database queries to keep the performance high as the data grows.

References

- [1] A. S. Marua, A. Barua, P. Rudra, and M. R. I. Jewel, “Job board platform system - a localhost laravel framework based application,” 2024. [Online]. Available: <https://github.com/RudraPankaj/JobBoardPlatformLaravel>