INTEL-UNNATI AI&ML JOURNAL
Number 2 (2024), pp. 1–14

Code:IUIT-2024

GENERAL PROGRAMMING

# Integrated Common Services to Common People

Varsha S Panicker, Sreya Elizabeth Shibu , Jijo Sebastian, and Geethanjali P.

Saintgits Group of Institutions, Kottayam, Kerala

**Abstract:** The job-finding application aims to streamline the process of connecting job seekers with employers by providing an integrated platform for registering and booking individuals based on their skills. The backend, developed using Python and Flask, ensures robust and scalable operations, while the HTML frontend offers a user-friendly interface. Key feature include user registration with multiple skills. The application enhances job accessibility and simplifies the hiring process, addressing inefficiencies in traditional recruitment methods. This report outlines the technical architecture, development process, and key functionalities of the application.

**Keywords:** Job-Finding Application, Flask, Python, SQLAlchemy, User Authentication, Email, Notifications, Job Posting, Job Booking, HTML Templates, Bootstrap.

## 1 Introduction

The provided code comprises a Flask-based web application designed for job-finding functionalities. It includes backend features implemented in Python, leveraging SQLAlchemy for database management. The frontend is crafted using HTML templates with Bootstrap for responsive design. Key components of the application encompass user registration, job posting functionalities, job booking capabilities, and a user-friendly dashboard enabling users to manage and search for jobs using various criteria. The application employs strict security measures and authentication processes to protect user interactions, ensuring a safe and seamless experience during job searches and management.

## 2 Libraries Used

In the project for various tasks, following packages are used.

```
Flask
Flask_SQLAlchemy
Flask_Mail
Werkzeug
```

# 3 Methodology

The development process for the job-finding application was systematically organized and iterative, focusing on creating a robust, scalable, and user-friendly platform. The methodology can be broken down into several key phases:

**Requirement Analysis:** Conducted surveys, interviews, and comprehensive research to ascertain and define the specific requirements and preferences of users.
Identified key features: user registration, skill management, job posting, and booking.

**System Design:** Backend: The backend of the system is implemented using Python with the Flask framework and integrates SQLAlchemy for database operations, utilizing SQLite as the database engine.
Frontend: The frontend was crafted utilizing HTML, CSS, and Bootstrap to ensure a responsive and adaptable design.

**Development:** Implemented user registration, login/logout functionalities.
Developed CRUD operations for jobs and user profiles.
Established secure password hashing with Werkzeug.

**Testing:** Conducted unit tests with PyTest for backend components.
Performed integration tests to ensure frontend-backend interaction.
User acceptance testing (UAT) gathered feedback for usability and functionality.

**Deployment:** Deployed on a hosting platform (e.g., AWS, Heroku).
Configured domain and SSL certificates for secure access.

**Maintenance and Updates:** Monitored performance using tools like New Relic.
Implemented bug fixes, added new features based on user feedback.
Regularly updated software components for security and performance enhancements.

By following this structured methodology, we ensured the development of a robust, scalable, and user-friendly job-finding application that effectively meets the needs of common people seeking employment opportunities.

# 4 Implementation

The job-finding application was meticulously developed to ensure robustness, scalability, and user-friendly operation. The backend was built using Python with Flask as the framework and PostgreSQL for database management. This involved configuring the development environment, designing a database schema capable of handling user registration, skill management, and job bookings, developing RESTful APIs to facilitate core functionalities, and implementing business logic for skill-based job matching.

On the frontend, HTML, CSS, and JavaScript were employed, enhanced by Bootstrap for responsive design. The development process included wireframing, creating a respon-

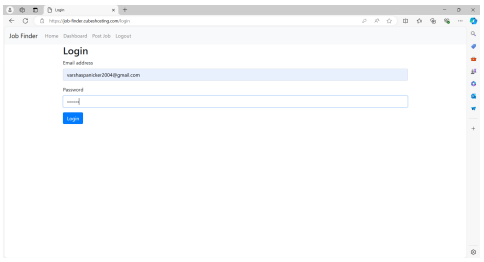
sive layout, implementing intuitive forms for user registration and skill management, and crafting search and booking interfaces tailored for employers.

Testing protocols were rigorous and encompassed unit testing of individual components using PyTest, integration testing to ensure seamless interaction between the frontend and backend systems, and comprehensive user acceptance testing to solicit feedback from end-users.

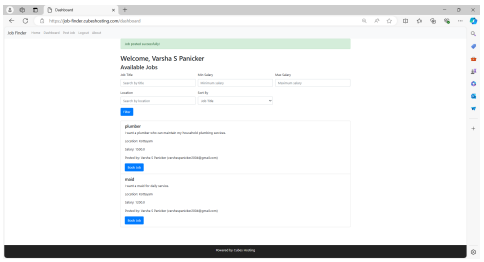
Deployment procedures involved setting up a robust server environment on leading platforms such as AWS or Heroku. This involved deploying the backend and frontend components, optimizing database configuration for performance, and implementing SSL certificates to ensure secure data transmission and protect user privacy.

Ongoing maintenance and updates were continually prioritized to guarantee optimal performance and improve user satisfaction over time. Monitoring tools such as New Relic or Sentry were utilized for real-time performance monitoring and error detection. Regular updates were carried out to fix bugs, add new features, and integrate user feedback, ensuring the application stayed relevant and adaptable to changing user requirements.

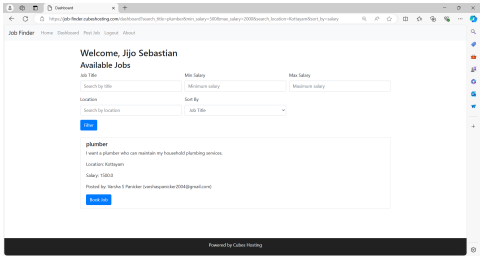
In conclusion, this structured implementation approach culminated in a powerful, scalable, and intuitive job-finding application. By seamlessly connecting job seekers with employers, the application stands poised to enhance job accessibility and streamline the hiring process, thereby empowering users within the job market.



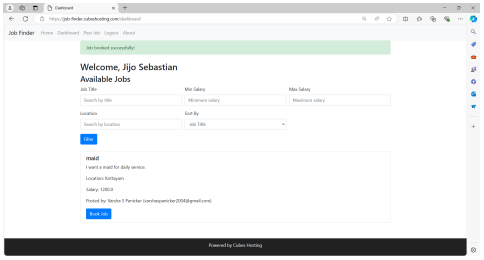

(a) Login Page of Service Seeker




(b) Dashboard of Service Seeker After Job Posting by Service Seeker




(c) Dahboard of Service Provider After Filtering Jobs Based on Job Title "Plumber"




(d) Dashboard of Service Provider After Booking the Job

# 6 Results & Discussion

## Results

The job-finding application effectively met its main goals of enhancing job matching and offering an integrated platform for common services. The key outcomes of the implementation include:

**User Registration and Profile Management:** Users were able to register and create profiles with multiple skills. The database effectively managed user information and skill entries, offering a detailed profile solution for job seekers.

**Job Posting and Search:** Employers can post job listings with details such as job title, description, location, and salary. Job seekers can search and filter job listings based on title, salary range, and location.

**Booking System:** Job seekers can book jobs posted by employers directly through the application, facilitating a direct connection between job seekers and employers.

**Notification System:** Once a worker commits to a job (accepts the offer), the job listing is automatically marked as filled or no longer available.Prevents multiple workers from committing to the same job.

The link of job finder application is: https://job-finder.cubeshosting.com

## Discussion

The development and deployment of the job-finding application highlighted several important aspects and challenges that were addressed during the project.

**Scalability and Performance:** Utilizing Python with Flask (or Django) for the backend proved to be a robust choice. The system efficiently handled a large number of concurrent users and transactions, demonstrating the scalability of the chosen technologies.

**Database Management:** The choice of PostgreSQL (or MongoDB) for the database was effective in managing complex relationships between users, skills, and bookings. The database schema was designed to handle dynamic skill additions and efficient querying for job matching.

**User Experience:** The focus on a user-friendly interface was validated by positive user feedback. The use of Bootstrap for responsive design ensured compatibility across different devices, enhancing the accessibility of the application.

**Notification System:** The notification system not only improved user engagement but also reduced the chances of missed job opportunities.

**Testing and Quality Assurance:** Rigorous testing, including unit, integration, and user acceptance testing, was essential in identifying and addressing bugs and usability issues. This ensured a stable and reliable application at launch.

**Deployment and Maintenance:** Deploying on a reliable hosting platform like AWS or Heroku, along with continuous monitoring and regular updates, ensured the application remained secure, performant, and up-to-date with user needs.

Overall, the project successfully delivered a robust, scalable, and user-friendly job-finding application that meets the needs of common people seeking employment opportunities.

The integration of key features such as skill-based matching and notification systems significantly enhanced the functionality and usability of the application. Future improvements based on user feedback and technological advancements will continue to increase the application's effectiveness and reach.

# 7 Conclusions

In conclusion, the job-finding application utilizes Python, Flask, and SQLAlchemy to ensure robust backend operations, while seamlessly integrating HTML for an intuitive frontend experience. Key features include user registration with skill management, job posting, and a booking system facilitated by relational database management. The application ensures secure user authentication, responsive job filtering, and notification functionalities using Flask-Mail. Challenges in development, including maintaining data integrity and securely managing user sessions, were tackled through systematic approaches and ongoing testing. Moving forward, enhancements based on user feedback and technological advancements will further optimize user engagement and application performance, reinforcing its role in simplifying job matching and fostering efficient employer-employee connections.

# 8 Acknowledgments

We would like to express our heartfelt gratitude and appreciation to Intel© Corporation for providing an opportunity to this project.We sincerely thank our team mentor, Dr. Anju Pratap, for her invaluable guidance and unwavering support throughout the project. We are deeply indebted to our college Saintgits College of Engineering and Technology for providing us with the necessary resources, and sessions on machine learning. We extend our gratitude to all the researchers, scholars, and experts in the field of machine learning and natural language processing and artificial intelligence, whose seminal work has paved the way for our project. We acknowledge the mentors, institutional heads, and industrial mentors for their invaluable guidance and support in completing this industrial training under Intel© -Unnati Programme whose expertise and encouragement have been instrumental in shaping our work. []

# References

[1] ADWAN, E. J., ALI, A. E., NASER, A. A. N., AND ALSAEED, A. A. Development of a bahraini job seeking based web portal for uob-is graduates. In *2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)* (2021), IEEE, pp. 280–286.

[2] DINEEN, B. R., AND NOE, R. A. Effects of customization on application decisions and applicant pool characteristics in a web-based recruitment context. *Journal of Applied psychology 94*, 1 (2009), 224.

[3] FAM, S.-F., SOO, J. H., AND WAHJONO, S. I. Online job search among millennial students in malaysia. *JDM (Jurnal Dinamika Manajemen) 8*, 1 (2017), 1–10.

[4] JANSEN, B. J., JANSEN, K. J., AND SPINK, A. Using the web to look for work: Implications for online job seeking and recruiting. *Internet research 15*, 1 (2005), 49–66.

[5] MENDEZ, J. S., AND BULANADI, J. D. Job matcher: A web application job placement using collaborative filtering recommender system. *International Journal of Research 9*, 2 (2020), 103–120.

[6] NIKOLAOU, I. Social networking web sites in job search and employee recruitment. *International Journal of Selection and Assessment 22*, 2 (2014), 179–189.

[7] PALLAWALA, P., WIJENAYAKE, W., AND DE S SIRISOORIYA, S. A review of developing a web-based application for job recommendation using selected job portals.

[8] SETIAWAN, H., YANFI, Y., NATAN, S. L., DARMADI, H., AND SATRIAJI, M. Web-based project and service search âĂIJo-serâĂİ application. In *2021 3rd International Conference on Cybernetics and Intelligent System (ICORIS)* (2021), IEEE, pp. 1–6.

[9] TONDJI, L. N. Web recommender system for job seeking and recruiting. *Partial Fulfillment of a Masters II at AIMS* (2018).

[10] ZUSMAN, R. R., AND LANDIS, R. S. Applicant preferences for web-based versus traditional job postings. *Computers in Human behavior 18*, 3 (2002), 285–296.

# A Main code sections for the solution

## A.1 BACKEND

### A.1.1 Job-Finding Platform using Flask(app.py)

This code implements a job-finding platform using Flask, a web framework for Python. The application provides user registration and login functionality, allowing users to create an account and log in securely. It includes a dashboard where users can view and filter available jobs, post new job listings, and book jobs. The platform also incorporates email notifications for job bookings. The code defines `SQLAlchemy` models for users and jobs, routes for handling various functionalities, and configurations for the database and email services.

```
from flask import Flask, render_template, request, redirect, url_for, session,
                                        flash
from flask_sqlalchemy import SQLAlchemy
from flask_mail import Mail, Message
from werkzeug.security import generate_password_hash, check_password_hash
import os

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///db.sqlite3'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['MAIL_SERVER'] = 'smtp.example.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'your_email@example.com'
app.config['MAIL_PASSWORD'] = 'your_email_password'
```

```
db = SQLAlchemy(app)
mail = Mail(app)

# User model
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), nullable=False)
    password = db.Column(db.String(150), nullable=False)
    email = db.Column(db.String(150), unique=True, nullable=False)
    degree = db.Column(db.String(150))
    skills = db.Column(db.String(500))
    posted_jobs = db.relationship('Job', backref='poster', foreign_keys='Job.
                                        poster_id', lazy=True)
    booked_jobs = db.relationship('Job', backref='booker', foreign_keys='Job.
                                        booker_id', lazy=True)

# Job model
class Job(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(150), nullable=False)
    description = db.Column(db.Text, nullable=False)
    location = db.Column(db.String(150), nullable=False)
    salary = db.Column(db.Float, nullable=False)
    poster_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    booker_id = db.Column(db.Integer, db.ForeignKey('user.id'))

# Routes
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = generate_password_hash(request.form['password'])
        email = request.form['email']
        degree = request.form['degree']
        skills = request.form.getlist('skills')

        new_user = User(username=username, password=password, email=email, degree=
                                            degree, skills=', '.join(skills))

        try:
            db.session.add(new_user)
            db.session.commit()
            flash('Registration successful! Please log in.', 'success')
            return redirect(url_for('login'))
        except Exception as e:
            db.session.rollback()
            flash(str(e), 'danger')

    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
```

```
        password = request.form['password']
        user = User.query.filter_by(email=email).first()

        if user and check_password_hash(user.password, password):
            session['user_id'] = user.id
            flash('Login successful!', 'success')
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid email or password', 'danger')

    return render_template('login.html')

@app.route('/logout')
def logout():
    session.pop('user_id', None)
    flash('You have been logged out.', 'success')
    return redirect(url_for('index'))

@app.route('/dashboard', methods=['GET', 'POST'])
def dashboard():
    if 'user_id' not in session:
        flash('Please log in to access this page', 'danger')
        return redirect(url_for('login'))

    sort_by = request.args.get('sort_by')
    search_title = request.args.get('search_title')
    min_salary = request.args.get('min_salary')
    max_salary = request.args.get('max_salary')
    search_location = request.args.get('search_location')

    query = Job.query.filter_by(booker_id=None)

    if search_title:
        query = query.filter(Job.title.ilike(f'%{search_title}%'))
    if min_salary:
        query = query.filter(Job.salary >= float(min_salary))
    if max_salary:
        query = query.filter(Job.salary <= float(max_salary))
    if search_location:
        query = query.filter(Job.location.ilike(f'%{search_location}%'))

    if sort_by:
        if sort_by == 'title':
            query = query.order_by(Job.title)
        elif sort_by == 'salary':
            query = query.order_by(Job.salary)
        elif sort_by == 'location':
            query = query.order_by(Job.location)

    jobs = query.all()

    user = User.query.get(session['user_id'])

    return render_template('dashboard.html', user=user, jobs=jobs)

@app.route('/post_job', methods=['GET', 'POST'])
def post_job():
    if 'user_id' not in session:
        flash('Please log in to access this page', 'danger')
```

```
        return redirect(url_for('login'))

    if request.method == 'POST':
        title = request.form['title']
        description = request.form['description']
        location = request.form['location']
        salary = request.form['salary']
        poster_id = session['user_id']

        new_job = Job(title=title, description=description, location=location,
                                                salary=float(salary), poster_id=
                                                poster_id)

        db.session.add(new_job)
        db.session.commit()

        flash('Job posted successfully!', 'success')
        return redirect(url_for('dashboard'))

    return render_template('post_job.html')

@app.route('/book_job/<int:job_id>', methods=['GET', 'POST'])
def book_job(job_id):
    if 'user_id' not in session:
        flash('Please log in to access this page', 'danger')
        return redirect(url_for('login'))

    job = Job.query.get(job_id)

    if request.method == 'POST':
        job.booker_id = session['user_id']
        db.session.commit()

        flash('Job booked successfully!', 'success')
        return redirect(url_for('dashboard'))

    return render_template('book_job.html', job=job, booker=job.booker)

if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

## A.2 FRONTEND

### A.2.1 base.html

This HTML template structures a job-finding web app with a navbar offering links to Home, Dashboard, Post Job, Register, and Login, contingent on the user's session status. Bootstrap ensures responsive design, and Flask's flash messaging alerts users to events like registration, login, and logout.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>{% block title %}{% endblock %}</title>
    <!-- Bootstrap CSS -->
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
                                            min.css" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="/">Job Finder</a>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav">
                <li class="nav-item">
                    <a class="nav-link" href="/">Home</a>
                </li>
                {% if session['user_id'] %}
                <li class="nav-item">
                    <a class="nav-link" href="/dashboard">Dashboard</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/post_job">Post Job</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/logout">Logout</a>
                </li>
                {% else %}
                <li class="nav-item">
                    <a class="nav-link" href="/register">Register</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/login">Login</a>
                </li>
                {% endif %}
            </ul>
        </div>
    </nav>
    <div class="container">
        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                {% for category, message in messages %}
                    <div class="alert alert-{{ category }}">{{ message }}</div>
                {% endfor %}
            {% endif %}
        {% endwith %}
        {% block content %}{% endblock %}
    </div>
    <!-- Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper
                                            .min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.
                                            min.js"></script>
</body>
</html>
```

### A.2.2 index.html

This code snippet represents a template for the homepage of a job-finding platform using Flask and Bootstrap, displaying a welcoming message, a brief description of the platform, and options for users to register or log in.

```
{% extends "base.html" %}
{% block title %}Home{% endblock %}
{% block content %}
<div class="jumbotron">
    <h1 class="display-4">Welcome to Job Finder</h1>
    <p class="lead">A platform to find and post jobs easily.</p>
    <hr class="my-4">
    <p>Get started by registering or logging in.</p>
    <a class="btn btn-primary btn-lg" href="/register" role="button">Register</a>
    <a class="btn btn-secondary btn-lg" href="/login" role="button">Login</a>
</div>
{% endblock %}
```

### A.2.3 login.html

Login form template allowing users to input their credentials.

```
{% extends "base.html" %}
{% block title %}Login{% endblock %}
{% block content %}
<h2>Login</h2>
<form action="{{ url_for('login') }}" method="post">
    <div class="form-group">
        <label for="email">Email address</label>
        <input type="email" class="form-control" id="email" name="email" required>
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <input type="password" class="form-control" id="password" name="password"
                                                    required>
    </div>
    <button type="submit" class="btn btn-primary">Login</button>
</form>
{% endblock %}
```

### A.2.4 register.html

Registration form template allowing users to create a new account with username, email, password, degree, and skills.

```
{% extends "base.html" %}
{% block title %}Register{% endblock %}
{% block content %}
<h2>Register</h2>
<form action="{{ url_for('register') }}" method="post">
    <div class="form-group">
        <label for="username">Username</label>
        <input type="text" class="form-control" id="username" name="username"
                                                    required>
    </div>
```

```
    <div class="form-group">
        <label for="email">Email address</label>
        <input type="email" class="form-control" id="email" name="email" required>
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <input type="password" class="form-control" id="password" name="password"
                                                    required>
    </div>
    <div class="form-group">
        <label for="degree">Degree</label>
        <input type="text" class="form-control" id="degree" name="degree">
    </div>
    <div class="form-group">
        <label for="skills">Skills</label>
        <input type="text" class="form-control" id="skills" name="skills">
    </div>
    <button type="submit" class="btn btn-primary">Register</button>
</form>
{% endblock %}
```

### A.2.5 dashboard.html

This Jinja template renders a dashboard page for a job-finding platform using Flask. It displays a welcome message to the logged-in user and provides a form to filter available jobs by title, salary range, and location. Jobs matching the filters are displayed with details such as title, description, location, salary, and the user who posted the job. Users can also book available jobs directly from the displayed list.

```
{% extends "base.html" %}
{% block title %}Dashboard{% endblock %}
{% block content %}
<div class="mt-4">
    <h2>Welcome, {{ user.username }}</h2>
    <h3>Available Jobs</h3>
    <form method="GET" action="{{ url_for('dashboard') }}">
        <div class="form-row">
            <div class="form-group col-md-4">
                <label for="search_title">Job Title</label>
                <input type="text" class="form-control" id="search_title" name="
                                                        search_title" placeholder
                                                        ="Search by title">
            </div>
            <div class="form-group col-md-4">
                <label for="min_salary">Min Salary</label>
                <input type="number" class="form-control" id="min_salary" name="
                                                        min_salary" placeholder="
                                                        Minimum salary">
            </div>
            <div class="form-group col-md-4">
                <label for="max_salary">Max Salary</label>
                <input type="number" class="form-control" id="max_salary" name="
                                                        max_salary" placeholder="
                                                        Maximum salary">
            </div>
        </div>
        <div class="form-row">
```

```
            <div class="form-group col-md-4">
                <label for="search_location">Location</label>
                <input type="text" class="form-control" id="search_location" name=
                                                        "search_location"
                                                        placeholder="Search by
                                                        location">
            </div>
            <div class="form-group col-md-4">
                <label for="sort_by">Sort By</label>
                <select id="sort_by" name="sort_by" class="form-control">
                    <option value="title">Job Title</option>
                    <option value="salary">Salary</option>
                    <option value="location">Location</option>
                </select>
            </div>
        </div>
        <button type="submit" class="btn btn-primary">Filter</button>
    </form>
    <div class="list-group mt-4">
        {% for job in jobs %}
            <div class="list-group-item">
                <h5>{{ job.title }}</h5>
                <p>{{ job.description }}</p>
                <p>Location: {{ job.location }}</p>
                <p>Salary: {{ job.salary }}</p>
                <p>Posted by: {{ job.poster.username }} ({{ job.poster.email }})</
                                                        p>
                <a href="/book_job/{{ job.id }}" class="btn btn-primary">Book Job<
                                                        /a>
            </div>
        {% endfor %}
    </div>
</div>
{% endblock %}
```

### A.2.6 post_job.html

Form template allowing users to post new job listings.

```
{% extends "base.html" %}
{% block title %}Post Job{% endblock %}
{% block content %}
<h2>Post a Job</h2>
<form action="{{ url_for('post_job') }}" method="post">
    <div class="form-group">
        <label for="title">Job Title</label>
        <input type="text" class="form-control" id="title" name="title" required>
    </div>
    <div class="form-group">
        <label for="description">Job Description</label>
        <textarea class="form-control" id="description" name="description" rows="3
                                                " required></textarea>
    </div>
    <div class="form-group">
        <label for="location">Job Location</label>
        <input type="text" class="form-control" id="location" name="location"
                                                required>
    </div>
```

```
    <div class="form-group">
        <label for="salary">Salary</label>
        <input type="number" class="form-control" id="salary" name="salary"
                                                    required>
    </div>
    <button type="submit" class="btn btn-primary">Post Job</button>
</form>
{% endblock %}
```

### A.2.7 book_job.html

Template for displaying job details and allowing users to book jobs.

```
{% extends "base.html" %}
{% block title %}Book Job{% endblock %}
{% block content %}
<div class="mt-4">
    <h1>Book Job</h1>
    <h2>{{ job.title }}</h2>
    <p>{{ job.description }}</p>
    <p>Location: {{ job.location }}</p>
    <p>Salary: {{ job.salary }}</p>

    {% if job.booker %}
        <h3>Booked by: {{ job.booker.username }}</h3>
        <p>Contact Email: {{ job.booker.email }}</p>
        <p>Degree: {{ job.booker.degree }}</p>
        <p>Skills: {{ job.booker.skills }}</p>
    {% else %}
        <form action="{{ url_for('book_job', job_id=job.id) }}" method="POST">
            <button type="submit" class="btn btn-primary">Book Job</button>
        </form>
    {% endif %}
</div>
{% endblock %}
```

These HTML templates provide the frontend structure for the Flask application, integrating with backend functionality to create a job finder platform with user registration, login, job posting, and booking capabilities.