

DEVELOPMENT AND IMPLEMENTATION OF A VOLUNTEER RECRUITMENT
SYSTEM USING SMART MACHINE LEARNING DATA DRIVEN MODEL

BY

ADERIBIGBE AYOMIDE OLUWABUSAYO

B.SC COMPUTER SCIENCE

MATRIC NO: 20/7729

A THESIS SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
CALEB UNIVERSITY. IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE AWARD OF BACHELOR OF SCIENCE (B.Sc) DEGREE IN
COMPUTER SCIENCE.

JULY 2024

DEDICATION

I, ADERIBIGBE AYOMIDE do here declare to Caleb University that this project is my original work and that it has neither been submitted nor being concurrently submitted for degree award in any other institution.

ACKNOWLEDGEMENT

First and foremost, I am profoundly grateful to Almighty God for granting me the strength, knowledge, and perseverance to complete this project. I would like to express my deepest appreciation to my supervisor, Dr Adekunle Eludire, whose invaluable guidance, insightful feedback, and constant support made this work possible. I am also deeply indebted to all my lecturers in the Department of Computer Science, Caleb University, for their academic guidance and encouragement throughout my studies. Special thanks go to my family and friends for their unwavering support and encouragement, and to my colleagues for their camaraderie and collaborative spirit during this research journey.

TABLE OF CONTENTS

CERTIFICATION

DEDICATION

ACKNOWLEDGEMENT

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

CHAPTER ONE: INTRODUCTION

1.1 Background of the Study

1.2 Definition of the Problem

1.3 Significance of Study

1.4 Aim and Objectives

1.4.1 Aim of the Project

1.4.2 Objectives

1.5 Methodology

1.5.1 Data collection

1.5.2 Data preprocessing

1.5.3 Machine learning model development

1.5.4 Platform design and integration

1.5.5 Deployment

1.6 Scope and Limitations of Study

1.6.1 Scope of Project

1.6.2 Limitations

1.7 Organization of Project

CHAPTER TWO: LITERATURE REVIEW

2.1 Historical Review of the Problem

2.2 Solving the problem of volunteer management

2.3 Current Trend of Solutions

2.4 Proposed Area of Improvement from Literature Analysis

CHAPTER THREE: METHODOLOGY

3.1 Problem Statement and Methods of Solution

3.2 The Solution Algorithm Concept

3.2.1 Profile-bases Matching Algorithm

3.2.2 Collaborative Filtering Algorithm

3.2.3 Hybrid Approach Algorithm

3.2.4 Flowchart of the Hybrid Approach Algorithm

3.3 The Architecture Methods

3.3.1 Architecture for Solution and Production

3.3.2 Volunteer Matching Process

3.3.3 Methods for Integration and Production

CHAPTER FOUR: DESIGN AND IMPLEMENTATION

4.1 Introduction

4.2 The Architecture and Installation

4.2.1 The Architecture

4.2.2 Installation

4.3 System Requirements

4.3.1 Hardware Requirements

4.3.2 Software Requirements

4.4 Civic Pulse Platform Design

4.5 Analysis on Input Used

4.6 Discussion of the Output Obtained

4.7 Making Use of the Platform

CHAPTER FIVE: CONCLUSION AND RECOMMENDATION

5.1 Conclusion

5.2 Recommendation

REFERENCES

APPENDICES

LIST OF FIGURES

Figure 3.2: Data-set sample used in training the volunteering matching model

Figure 3.3: Hybrid algorithm approach

Fig 4.1: Django-React architecture for volunteering platform

Fig. 4.2: Landing page

Fig. 4.3: Sign-up company page

Fig. 4.4: Sign-up volunteer page

Fig. 4.5: Login page

Fig. 4.6: User dashboard

Fig. 4.7: Volunteering application page

Fig. 4.8: Live chat page

Fig. 4.9: Notification page

Fig. 4.10: Profile page

Fig. 4.11: Update profile page

Fig. 4.12: Company dashboard page

Fig 4.13: Authentication bio-data input process for Civic Pulse

LIST OF TABLES

Table 1.6.2: Limitations of Project

Table 2.1. Evolution of volunteer management practices

ABSTRACT

The goal of this study is to develop and implement "CivicPulse," a volunteer platform leveraging machine learning to streamline volunteer opportunity matching. This project addresses the challenge of efficiently matching volunteers with suitable tasks by utilizing data analytics for optimal pairing. The methodology involves designing a web-based platform using Django for the backend and React for the frontend, incorporating a machine learning algorithm to analyze user profiles and preferences against available opportunities. Data for this study was sourced from Kaggle, focusing on user profession, interests, and other relevant factors. Findings demonstrate that automated matching significantly enhances the efficiency and accuracy of volunteer placements, reducing the manual effort required by organizations. The platform also provides real-time notifications and data-driven insights, facilitating better volunteer engagement and program management. The study's findings underscore the usefulness of integrating technology in volunteer management, presenting a scalable solution that can be adapted by non-profits and other organizations to improve their volunteer programs and overall impact.

CHAPTER ONE

INTRODUCTION

1.1 Background of The Study

Volunteering has a rich history dating back centuries, rooted in the altruistic efforts of individuals and communities to support one another. In ancient civilizations such as Greece, Rome, and China, voluntary service was often tied to religious and civic duties, where citizens would come together to address communal needs and support the less fortunate.

Over time, the concept of volunteering evolved, particularly during the Middle Ages and Renaissance periods, where religious orders and guilds organized mutual aid to assist the poor and vulnerable. With the advent of the Industrial Revolution, volunteering took on a new significance as societal challenges intensified amidst rapid urbanization and industrialization. Religious and philanthropic organizations, such as the Red Cross and the Salvation Army, emerged to provide humanitarian aid and alleviate social distress.

In the modern era, volunteering has become increasingly formalized and diversified, with individuals, non-governmental organizations (NGOs), and businesses all playing roles in addressing various social, environmental, and humanitarian issues. NGOs rely

heavily on volunteers to deliver their programs and services, while businesses recognize the value of corporate social responsibility initiatives that engage employees in volunteer activities.

With the digital age, the terrain of volunteering has undergone a transformation, as technological advancements have enabled the creation of online platforms and digital tools to facilitate volunteer engagement. These platforms offer a convenient way for individuals to discover and apply for volunteer opportunities, transcending geographical barriers and expanding the reach of volunteerism.

The goal of this study is to explore how advancements in technology, particularly in the realm of machine learning, can further enhance the volunteering experience by streamlining the process of matching volunteers with opportunities. By leveraging machine learning algorithms to analyze volunteer preferences, skills, and availability, it is possible to offer a more tailored and efficient approach to volunteering. This research aims to investigate the effectiveness of such technology-driven solutions in optimizing volunteer engagement and satisfaction, ultimately contributing to the advancement of volunteer management practices in both NGOs and traditional organizations alike.

1.2 Definition of the Problem

The current manual process of matching individuals with volunteer opportunities presents several challenges for organizations and potential volunteers alike. Primarily, the labor-intensive nature of this process results in significant time and resource expenditure. Organizations must sift through numerous volunteer applications, manually assess individual skills and interests, and match them with available tasks, a process prone to human error and inefficiencies.

Moreover, the manual approach often leads to mismatches between volunteers and tasks, as organizations may lack the necessary insight into individual skill sets and preferences. Consequently, volunteers may find themselves assigned to tasks that do not fully utilize their capabilities or align with their interests, resulting in decreased engagement and satisfaction.

Additionally, the lack of a centralized platform for volunteer management hinders collaboration and coordination among organizations. Volunteer opportunities may be scattered across various platforms or managed through disparate systems, making it challenging for individuals to discover relevant opportunities and for organizations to recruit suitable volunteers.

Overall, the current state of volunteer matching is characterized by inefficiency, suboptimal resource utilization, and a lack of alignment between volunteer skills and organizational needs. Addressing these challenges is crucial for enhancing the

effectiveness and sustainability of volunteer programs and maximizing the impact of volunteer efforts.

1.3 Significance Of Study

Finding individuals to participate in volunteer work poses significant challenges for organizations, often entailing a labor-intensive process of manual task allocation. This approach frequently results in inefficiencies and mismatches in skill sets and availability, ultimately undermining the effectiveness of volunteer programs. Moreover, the inability to fully leverage the skills and interests of volunteers may lead to disengagement and decreased overall impact.

The proposed platform endeavors to address these challenges by harnessing the power of data analytics and machine learning algorithms. Through sophisticated analysis of individual profiles, encompassing skills, interests, and availability, the platform aims to facilitate optimal matches with the specific requirements of organizations. By automating the matching process, organizations can significantly reduce the time and resources expended on volunteer recruitment, while ensuring that tasks are assigned to individuals who can deliver the greatest value.

Furthermore, by aligning volunteer opportunities with the skills and interests of individuals, the platform seeks to enhance the overall engagement experience. Individuals are more likely to remain committed and motivated when entrusted with tasks that resonate with their capabilities and passions, thereby fostering a culture of sustained involvement. This, in turn, yields tangible benefits for organizations, as a dedicated and motivated pool of individuals is instrumental in driving the success and longevity of volunteer initiatives.

The implementation of this platform holds immense promise for revolutionizing the landscape of engagement and management practices. By introducing efficiency and precision into the volunteer matching process, organizations can optimize their utilization of human capital and amplify the impact of their endeavors, thereby advancing the overarching objectives of their programs and initiatives.

1.4 Aim and Objectives of the Project

1.4.1 Aim of the Project:

The aim of the project is to develop and implement a volunteer recruitment system using smart machine learning data driven model in order to eliminate the current inefficiencies in the volunteering recruitment process.

1.4.2 Objectives:

1. Design and develop a user-friendly online platform for volunteer management.
2. Implement mechanisms for collecting and storing volunteer and organizational data.
3. Utilize data analytics techniques to analyze volunteer profiles and opportunity requirements.
4. Integrate machine learning algorithms to automate the volunteer matching process.
5. Enhance the user experience through personalized recommendations and real-time notifications.
6. Evaluate the platform's effectiveness in improving the volunteer matching process.
7. Design the platform for scalability and sustainability to accommodate growth over time. The architecture of our volunteer platform, built using Django, is designed for scalability and sustainability, ensuring it can handle increasing user demands and maintain high performance over time through modular design, efficient use of RESTful APIs, and robust CI/CD practices such as automated testing, continuous integration and containerization..

1.5 Methodology

To address the challenge of efficiently matching volunteers with opportunities, the platform utilizes a data-driven approach. This involves collecting relevant data,

processing it, and applying machine learning algorithms to optimize the matching process. The methods of solution are described in the following subsections.

1.5.1 Data collection

Data source

- **Kaggle:** Kaggle is the world's largest data science community with millions of open-source datasets available for integration into real-world projects. For this platform, data regarding people's professions, interests, and salary are sourced from Kaggle.

Steps involved in data collection

1. **Dataset selection:** Identify and select datasets relevant to the volunteer matching platform on Kaggle that include information on individuals' professions, interests, country of residence, years of working experience and salaries.
2. **Data download:** Use Kaggle's API to download the selected datasets
3. **Data storage:** Store the downloaded datasets in a structured format suitable for processing and analysis.

1.5.2 Data preprocessing

Tools and packages

- Pandas: For data manipulation and analysis.
- NumPy: For numerical operations and handling arrays.
- Scikit-learn: For preprocessing and machine learning tasks.

Steps for data preprocessing

1. **Data cleaning**: Remove any duplicates, handle missing values, and correct inconsistencies in the data.
2. **Feature engineering**: Create new features or modify existing ones to improve the performance of the machine learning models.
3. **Normalization**: Normalize or standardize data to ensure uniformity across the dataset.

1.5.3 Machine learning model development

Tools and packages

- Scikit-learn: For implementing machine learning algorithms.
- TensorFlow: For building and training neural networks.
- Jupyter Notebook: For interactive development and visualization of machine learning models.

Steps for tools and packages selection

1. Model Selection: Choose appropriate machine learning models for the task. For this project we are using Random Forest, a supervised machine-learning algorithm made up of decision trees. It is used for both classification and regression problems.
2. Training and Testing Split: Divide the data into training and testing sets to evaluate model performance.
3. Model Training: Train the selected models using the training dataset.
4. Model Evaluation: Evaluate the models using metrics such as accuracy, precision, recall, and F1-score on the testing dataset.
5. Model Tuning: Perform hyperparameter tuning to optimize model performance.

1.5.4 Platform design and integration

Tools and packages

- Django: python-based web framework for backend development.
- Django REST Framework: for building RESTful APIs.
- SQLite: lightweight database for initial development and testing.
- React.js: for frontend development.
- HTML and CSS: for frontend development.
- Ajax and JQuery: for frontend development.

Steps for platform design and integration:

1. Backend Development

- Set up Django and configure the project.
- Develop RESTful APIs using Django REST Framework to handle client-server interactions.
- Implement user authentication and authorization.

2. Frontend Development

- Develop a responsive web interface using React.js.
- Integrate the frontend with the backend APIs to provide seamless user interactions.

3. Database Integration

- Set up SQLite for initial data storage.
- Design database schemas to store user profiles, volunteer opportunities, and interactions.

4. Machine Learning Integration

- Deploy trained machine learning models within the Django framework.

- Develop endpoints to serve real-time predictions and recommendations.

1.5.5 Deployment

Tools and packages for deployment

- Docker: For containerizing the application.
- Kubernetes: For orchestrating and managing containers.
- PythonAnywhere: Hosting service for Django applications.
- GitHub Actions: For CI/CD pipelines.

Steps for deployment

1. Containerization

- Use Docker to create containers for the application components (frontend, backend, and database).
- Define Dockerfiles for each component.

2. Orchestration

- Set up Kubernetes to manage container deployment, scaling, and monitoring.
- Define Kubernetes manifests for deploying the application.

3. **CI/CD Pipeline**

- Set up GitHub Actions to automate testing, building, and deployment.
- Configure the CI/CD pipeline to trigger on code commits and pull requests.

4. **Hosting**

- Deploy the application on PythonAnywhere hosting service.
- Configure the hosting environment to ensure proper connectivity and performance.

By following these specific methods and utilizing the mentioned tools and packages, the development and implementation of the volunteer platform can be effectively achieved, ensuring a streamlined and scalable solution.

1.6 Scope and Limitations Of Study

1.6.1 Scope of Project:

The project will focus primarily on the development and implementation of the volunteer platform and its associated features. As a result, certain aspects of volunteer

management, such as volunteer training and retention strategies, will not be addressed within the scope of this project

1.6.2 Limitations of Project:

<u>Category</u>	Limitation Description	Impact
Data Limitations	Incomplete and inconsistent datasets from Kaggle.	Affected model accuracy.
Resource Constraints	Limited resources available, including time, budget and personnel.	Affected the depth and breadth of research, development and testing.
Deployment Challenge	Hosting limitations on PythonAnywhere and initial CI/CD pipeline issues.	Restricted scalability and deployment issues.
User Interface	Limited usability testing with real users	Potential UI/UX optimizations needed.

1.7 Organization of project

The project is structured into the following chapters, each addressing specific aspects of the research and development process:

1. Chapter One is the Introduction and it provides an overview of the project, including its objectives, significance, and scope.
2. Chapter Two is the Literature Review and it. Reviews existing literature and research relevant to volunteer management, machine learning applications, and online platforms for volunteer engagement.
- 3 Chapter Three is the Methodology. The methodology, describes the approach and methods used to achieve the project objectives, including platform development, machine learning integration, data collection and analysis, user experience enhancement, evaluation and optimization, and scalability and sustainability considerations.
4. Chapter Four details the design, development, and implementation of the volunteer platform, including its architecture, installation and configuration tools, system requirements, and discussions on input and output.

5. Chapter Five contains the Conclusion and Recommendation. It summarizes the key findings, contributions, and implications of the project, as well as providing recommendations for future research or improvements.

CHAPTER TWO

LITERATURE REVIEW

2.1 Historical Review of the Problem

Volunteer management has evolved significantly over the years, influenced by societal changes and technological advancements. Early forms of volunteerism can be traced back to ancient civilizations, where communities relied on collective efforts to address communal needs (Smith, 2018). However, formalized volunteer management began to emerge during the 19th century with the rise of charitable organizations and philanthropic movements (Wilson, 2016).

In the early stages of volunteer management, coordination and recruitment were often carried out through informal networks and personal connections within local communities (Jackson, 2019). This decentralized approach limited the reach of volunteer efforts and made it challenging to match volunteers with appropriate opportunities effectively.

The industrial revolution and urbanization further influenced volunteer management practices, as organizations sought to address social issues arising from rapid urban growth and industrialization (Jones, 2020). Charitable organizations and

religious institutions played a significant role in mobilizing volunteers to address these challenges, but the lack of standardized processes hindered efficiency and scalability.

The 20th century saw the emergence of more structured volunteer management practices, with the establishment of volunteer bureaus and formalized recruitment processes (Roberts, 2017). However, manual methods such as paper-based records and in-person interviews remained prevalent, limiting the ability to scale volunteer programs and match volunteers with opportunities efficiently.

The advent of digital technology in the late 20th century revolutionized volunteer management, enabling organizations to leverage online platforms and databases to streamline recruitment and coordination efforts (Taylor, 2021). Online volunteer matching platforms emerged, providing a centralized hub for organizations to post opportunities and for volunteers to search and apply for roles (Brown, 2019).

<u>Time Period</u>	<u>Milestones</u>
Ancient Civilizations	- Informal volunteerism within local communities.
19th Century	- Emergence of charitable organizations and philanthropic movements.

	<ul style="list-style-type: none"> - Formalization of volunteer management practices.
Early 20th Century	<ul style="list-style-type: none"> - Establishment of volunteer bureaus. - Introduction of formalized recruitment processes.
Late 20th Century	<ul style="list-style-type: none"> - Advent of digital technology. - Rise of online volunteer matching platforms.
21st Century	<ul style="list-style-type: none"> - Growing interest in leveraging data analytics and machine learning for volunteer matching

Table 2.1. Evolution of volunteer management practices

Despite these advancements, challenges persisted in volunteer matching, including the need for more personalized and data-driven approaches. Recent years have seen a growing interest in leveraging technologies such as data analytics and machine learning to improve volunteer matching processes (Johnson, 2022). These technologies hold promise for enhancing the efficiency and effectiveness of volunteer management by enabling organizations to better match volunteers with opportunities based on skills, interests, and availability.

2.2 Solving the problem of volunteer management:

Several real-world entities have developed and implemented solutions to address the challenges in volunteer management:

1. **VolunteerMatch**: VolunteerMatch is an online platform that connects volunteers with nonprofit organizations in need of their services. It allows organizations to post volunteer opportunities and enables volunteers to search for opportunities based on their interests, skills, and location. VolunteerMatch automates the volunteer matching process, making it easier for organizations to recruit volunteers and for volunteers to find meaningful opportunities (Brown, 2019).

2. **Idealist**: Idealist is another online platform that connects individuals, organizations, and communities to opportunities for social change. It provides a comprehensive database of volunteer opportunities, jobs, internships, and events in various fields such as nonprofit work, social justice, and community development. Idealist offers features for organizations to post volunteer opportunities and for individuals to search and apply for opportunities based on their preferences (Smith, 2018).

3. **Points of Light**: Points of Light is a nonprofit organization that focuses on mobilizing volunteers and promoting volunteerism worldwide. It offers various programs and initiatives aimed at engaging individuals and organizations in volunteer activities, including the Points of Light Global Network, the Corporate Service

Council, and the Points of Light Civic Accelerator. Points of Light provides resources, training, and support to help organizations effectively manage volunteer programs and maximize their impact (Johnson & Lee, 2017).

4. HandsOn Network: HandsOn Network is a global network of volunteer organizations that mobilizes volunteers to address community needs and create social change. It operates a network of volunteer centers and affiliates across the United States and around the world, offering a wide range of volunteer opportunities and projects. HandsOn Network provides tools, resources, and training to help organizations recruit, manage, and retain volunteers effectively (Williams, 2016).

United Nations Volunteers (UNV): UNV is a program of the United Nations that promotes volunteerism for peace and development worldwide. It mobilizes volunteers to support UN agencies, governments, and civil society organizations in achieving their goals and addressing global challenges. UNV offers a variety of volunteer opportunities in areas such as humanitarian assistance, peace building, and sustainable development, enabling individuals to contribute their skills and expertise to meaningful projects (Garcia, 2020).

2.3 Current Trend of Solutions

In recent years, several trends have emerged in the field of volunteer management, reflecting evolving needs, technological advancements, and changing demographics.

These trends include:

1. **Online volunteer engagement platforms:** There is a growing reliance on online platforms and digital tools to facilitate volunteer engagement. Organizations are increasingly leveraging websites, mobile apps, and social media platforms to recruit, manage, and coordinate volunteers. These platforms offer features such as volunteer matching, event management, and communication tools to streamline the volunteer experience and enhance organizational efficiency (Brown, 2019).

2. **Remote and virtual volunteering:** The rise of remote work and virtual volunteering opportunities has enabled volunteers to contribute their time and skills from anywhere in the world. Organizations are embracing virtual volunteering models, allowing volunteers to participate in projects remotely, such as providing online mentoring, conducting research, or designing marketing materials. This trend has broadened access to volunteer opportunities and increased flexibility for both volunteers and organizations (Smith, 2020).

3. **Data-driven volunteer management:** There is a growing emphasis on data analytics and insights to inform volunteer management decisions. Organizations are leveraging data analytics tools to track volunteer engagement metrics, analyze

volunteer demographics and preferences, and identify trends and patterns in volunteer behavior. By harnessing the power of data, organizations can make more informed decisions, tailor volunteer experiences, and optimize volunteer programs for maximum impact (Garcia, 2018).

4. **Diversity, Equity, and Inclusion (DEI) Initiatives:** There is a heightened focus on promoting diversity, equity, and inclusion in volunteer management practices. Organizations are implementing DEI initiatives to ensure that volunteer opportunities are accessible and inclusive to individuals from diverse backgrounds and communities. This includes providing culturally responsive volunteer training, addressing barriers to participation, and actively engaging underrepresented groups in volunteer activities (Johnson, 2017).

5. **Corporate Volunteerism and Employee Engagement:** Corporate volunteerism programs continue to gain momentum as businesses recognize the benefits of employee volunteer engagement for employee morale, skill development, and corporate social responsibility. Companies are implementing volunteer programs, offering paid volunteer time off, and organizing corporate volunteering events to encourage employee engagement in community service activities. This trend reflects a growing recognition of the role of businesses in driving social impact and fostering employee well-being (Williams, 2019).

2.4 Proposed Area of Improvement from Literature Analysis

The analysis underscores several areas where enhancements can be made in volunteer management practices, aligning with the objectives of the project to develop a volunteer platform utilizing machine learning to streamline volunteer matching. These proposed areas of improvement include:

1. Enhanced Volunteer Matching Algorithms

Traditional volunteer matching methods often rely on manual review and subjective judgment, leading to inefficiencies and mismatches. Leveraging machine learning algorithms, the volunteer platform can automate the matching process and improve the accuracy of volunteer-organization pairings. Advanced algorithms can analyze volunteer profiles, organizational requirements, and historical volunteer data to generate optimal matches based on skills, interests, and availability.

2. Personalized Volunteer Experiences

Volunteers are more likely to remain engaged and committed when their experiences are personalized to their preferences and interests. Incorporating features such as personalized recommendations, tailored volunteer opportunities, and customized communication channels into the volunteer platform can create a more engaging and fulfilling experience for volunteers. By understanding each volunteer's

motivations and preferences, the platform can match volunteers with opportunities that align with their interests, maximizing their impact and satisfaction.

3. Real-time Volunteer Availability Tracking

Real-time communication and coordination between volunteers and organizations are crucial for efficient volunteer deployment. Integrating features for volunteers to indicate their availability, update their schedules, and receive instant notifications about relevant volunteer opportunities into the volunteer platform can provide real-time visibility into volunteer availability. This enables organizations to quickly identify and deploy volunteers for urgent or time-sensitive projects, improving responsiveness and effectiveness.

4. Continuous Feedback and Evaluation Mechanisms

Feedback and evaluation are essential for continuous improvement in volunteer management practices. The volunteer platform can incorporate feedback mechanisms for volunteers to rate their volunteer experiences, provide suggestions for improvement, and share their insights and challenges. Organizations can use this feedback to evaluate the effectiveness of volunteer programs, identify areas for improvement, and make data-driven decisions to enhance volunteer engagement and satisfaction.

5. Integration with Existing Volunteer Management Systems

Interoperability and integration with existing volunteer management systems and databases are crucial. Designing the volunteer platform to seamlessly integrate with other volunteer management tools and platforms used by organizations enables organizations to leverage existing resources and infrastructure while enhancing the capabilities and efficiency of volunteer management processes.

Incorporating these proposed areas of improvement into the development of the volunteer platform can address the challenges identified and create a more efficient, effective, and user-friendly solution for volunteer management and engagement.

CHAPTER THREE

METHODOLOGY

3.1 Problem Statement and Methods of Solution

Volunteer management has traditionally been a cumbersome and time-consuming process. Organizations often face challenges in efficiently recruiting and matching volunteers to suitable tasks. Manual matching processes can result in inefficiencies, mismatches in skills and availability, and ultimately, a less impactful volunteer experience. These challenges are exacerbated by the growing scale of volunteerism and the increasing demand for more streamlined and effective management practices.

To address these issues, this project proposes the development and implementation of a volunteer platform that employs machine learning algorithms to enhance the process of matching volunteers with opportunities. The platform aims to provide a more streamlined, accurate, and efficient system for both volunteers and organizations, improving the overall volunteer experience and organizational outcomes.

1. Requirement analysis and data gathering for model development

- Conduct a comprehensive analysis of the requirements of both volunteers and organizations. This includes finding unique characteristics that are common to jobs and volunteers such as educational qualifications, skill requirements and technicality.

- Gather data on volunteer preferences, skills, availability, and past volunteering experiences.
- Understand the needs of organizations, including the types of volunteer opportunities available, required skills, and preferred volunteer profiles.
- I designed a django-based architecture for the volunteer platform, incorporating a user-friendly interface (CivicPulse).
- I developed a database schema using Django model architecture. This enables users to update their information consisting of user profiles, company profiles, jobs and applications into the SQLite securely and efficiently.

Django model SQLite database schema for Civic Pulse

Below is a Django database schema structure written in python code. It is used in constructing the architecture of our SQLite database, which in turns declare all the variables we'd be storing and using during the lifespan of our volunteering matching system.

```
from Django.db import models
```

```

class Profile(models.Model):

    userEmail = models.EmailField(max_length=250, default="")

    userName = models.CharField(max_length=50, default="")

    userFullName = models.CharField(max_length=50, default="")

    userTitle = models.CharField(max_length=100, default="")

    userContact = models.CharField(max_length=50, default="+ (000) 000-000-000")

    userBio = models.CharField(max_length=500, default="")

    userType = models.CharField(max_length=100, default="employee")

    userDOB = models.CharField(max_length=100, default="-- -- --")

    userLocation = models.CharField(max_length=100, default="-- -- --")

    userGender = models.CharField(max_length=100, default="Male")

    userProfilePicture = models.ImageField(upload_to='profile_picture/', null=True,
blank=True)

    userResume = models.FileField(upload_to='resume(cv)/', null=True,
blank=True)

    userPassword = models.CharField(max_length=50,
default="no_active_password")

    admin_conversation = models.CharField(max_length=500, default="none")

```

```

class Education(models.Model):

    school = models.CharField(max_length=50)

    degree = models.CharField(max_length=50)

    field = models.CharField(max_length=50)

    startDate = models.CharField(max_length=25)

    endDate = models.CharField(max_length=25)

    description = models.CharField(max_length=5000)

    userProfile = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)


class Meta:

    db_table = 'Education'


class Skill(models.Model):

    skill_set = models.CharField(max_length=50)

    proficiency = models.CharField(max_length=10)

    userProfile = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)

```

```

class Meta:

    db_table = 'Skill Sets'


class Experience(models.Model):

    Experience_Title = models.CharField(max_length=50, default="")

    Experience_Employment_Type = models.CharField(max_length=50,
default="")

    Experience_Organization_Name = models.CharField(max_length=50,
default="")

    Experience_Description = models.CharField(max_length=500, default="")

    Experience_Location = models.CharField(max_length=50, default="")

    Experience_Start_Date = models.CharField(max_length=50, default="")

    Experience_Close_Date = models.CharField(max_length=50, default="")

    userProfile = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)


class Meta:

    db_table = 'Working Experience'

```

```

class JobPost(models.Model):

    job_title = models.CharField(max_length=150)

    job_description = models.CharField(max_length=150)

    job_type = models.CharField(max_length=50, default="On-Site")

    job_poster = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)

    job_pay = models.CharField(max_length=200, default="")

    job_location = models.CharField(max_length=100, default="United States")

    job_external = models.CharField(max_length=50, default="none")

    job_external_link = models.CharField(max_length=5000, default="___")


class Meta:

    db_table = 'Job Post'


class JobCategory(models.Model):

    category_name = models.CharField(max_length=150)

    job = models.ManyToManyField('JobPost', default=[0])

```



```
class Meta:
```

```
    db_table = 'Job Category'
```

```
class JobRequirement(models.Model):
```

```
    requirement = models.CharField(max_length=500)
```

```
    job = models.ForeignKey('JobPost', on_delete=models.CASCADE, default=1)
```

```
class Meta:
```

```
    db_table = 'Job Requirement'
```

```
class JobRole(models.Model):
```

```
    role = models.CharField(max_length=500)
```

```
    job = models.ForeignKey('JobPost', on_delete=models.CASCADE, default=1)
```

```
class Meta:
```

```
    db_table = 'Job Role'
```

```

class Application(models.Model):

    job = models.ForeignKey('JobPost', on_delete=models.CASCADE, default=1)

    job_applicant = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)

    application_status = models.CharField(max_length=50, default="pending")

    interview_date = models.CharField(max_length=150, default="")

    interview_time = models.CharField(max_length=150, default="")

    interview_location_url = models.CharField(max_length=1000)

    interview_online = models.CharField(max_length=150, default="false")


class Meta:

    db_table = 'Application'


class JobNotification(models.Model):

    job_application = models.ForeignKey('Application',
on_delete=models.CASCADE, default=1)

    user = models.ForeignKey('Profile', on_delete=models.CASCADE, default=1)

    check_value = models.CharField(max_length=50, default="uncheck")

```

```

time_sent = models.DateTimeField(auto_now_add=True)


class Meta:

    db_table = 'Job Notification'


class Chat(models.Model):

    user = models.ManyToManyField('Profile', default=[0],
related_name="chat_user_list")

    latest_update = models.CharField(max_length=50, default="none")


class Meta:

    db_table = 'Chat'


# chat messages


class Chatmessage(models.Model):

    sender = models.ForeignKey('Profile', on_delete=models.CASCADE, default=1,
related_name="chat_sender")

    chat = models.ForeignKey('Chat', on_delete=models.CASCADE, default=1,)

    message = models.TextField(max_length=3000, default=None)

```

```

time_sent = models.DateTimeField(auto_now_add=True)

class Meta:

    db_table = 'Chat Message'

# chat Report

class ChatReported(models.Model):

    reporting_user = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1, related_name="report_sender")

    chat = models.ForeignKey('Chat', on_delete=models.CASCADE, default=1,)

    time_reported = models.DateTimeField(auto_now_add=True)

class Meta:

    db_table = 'Chat Reported'

```

3. Data collection and preprocessing

- Existing data were gathered and collected from Kaggle dataset platform (a free open source marketplace consisting of millions of datasets gathered in real time for professional use).

#	A	B	C	D	E	F	G	H	I	J	K
1	Age	Gender	Education Level	Job Title	Years of Experience	Salary	Country	Race	Senior		
2		32 Male		1 Software Engineer	5	90000 UK		White	0		
3		28 Female		2 Data Analyst	3	65000 USA		Hispanic	0		
4		45 Male		3 Manager	15	150000 Canada		White	1		
5		36 Female		1 Sales Associate	7	60000 USA		Hispanic	0		
6		52 Male		2 Director	20	200000 USA		Asian	0		
7		29 Male		1 Marketing Analyst	2	55000 USA		Hispanic	0		
8		42 Female		2 Product Manager	12	120000 USA		Asian	0		
9		31 Male		1 Sales Manager	4	80000 China		Korean	0		
10		26 Female		1 Marketing Coordinator	1	45000 China		Chinese	0		
11		38 Male		3 Scientist	10	110000 Australia		Australian	1		
12		29 Male		2 Software Developer	3	75000 UK		Welsh	0		
13		48 Female		1 HR Manager	18	140000 UK		Asian	0		
14		35 Male		1 Financial Analyst	6	65000 China		Korean	0		
15		40 Female		2 Project Manager	14	130000 USA		African American	0		
16		27 Male		1 Customer Service Rep	2	40000 Canada		Asian	0		
17		44 Male		1 Operations Manager	16	125000 China		Chinese	0		
18		33 Female		2 Marketing Manager	7	90000 USA		Asian	0		
19		39 Male		3 Engineer	12	115000 UK		Mixed	1		
20		25 Female		1 Data Entry Clerk	0	35000 UK		Asian	0		
21		51 Male		1 Sales Director	22	180000 Australia		Asian	0		
22		34 Female		2 Business Analyst	5	80000 China		Korean	0		
23		47 Male		2 VP of Operations	19	190000 UK		White	0		

Figure 3.2. Dataset sample used in training the volunteering matching model

- Preprocess the data to ensure it is clean, consistent, and suitable for training machine learning models.
- Perform data augmentation to enrich the dataset.

4. Development of machine learning algorithms

- Develop and train machine learning models to analyze volunteer and organization data gather from Kaggle.
- Use supervised learning techniques, Random Forest Classifier to create matching algorithms that predict the best matches between volunteers and opportunities.
- Implement features such as collaborative filtering and content-based filtering to improve matching accuracy.

5. Platform development

- Develop the front-end interface of the platform using modern web development frameworks (HTML, CSS, Javascript, React, JQuery, Ajax and Bootstrap).
- Implement the back-end services (Python Django Framework) to handle data processing, machine learning model integration, and user authentication.
- Ensure the platform is scalable, secure, and responsive through testing with Selenium python module.

6. Integration and testing

- Integrate the machine learning models with the platform's Django back-end framework.
- Conduct rigorous testing of the platform to ensure functionality, performance, and security.

7. Deployment and Maintenance

- Deploy the platform on PythonAnywhere hosting service, ensuring high availability and performance.
- Monitor the platform's usage and performance, making continuous improvements based on user feedback and system analytics.

- Provide support and maintenance to address any issues and implement new features as needed.

8. Evaluation and feedback:

- Collect and analyze feedback from monthly tests to evaluate the effectiveness of the platform.
- Use Google analytics to measure key performance indicators such as volunteer engagement rates, matching accuracy, and user satisfaction.
- Continuously improve the platform based on evaluation results and emerging needs.

This methodology outlines a structured approach to developing a volunteer platform that leverages machine learning to improve the efficiency and effectiveness of volunteer management. By addressing the identified problems through these methods, the project aims to create a robust solution that benefits both volunteers and organizations.

3.2 The Solution Algorithm Concept

In developing the volunteer platform, several solution algorithms are designed to address different aspects of the volunteer matching process. These algorithms

leverage machine learning techniques to optimize the matching of volunteers to opportunities based on various criteria. The primary focus is on ensuring accuracy, efficiency, and user satisfaction. Below are the three solution algorithms developed for the platform:

3.2.1 Profile-based Matching Algorithm

Algorithm overview

This algorithm matches volunteers to opportunities based on the detailed profiles created by both parties. Profiles include skills, interests, availability, and past experiences. The algorithm uses a content-based filtering approach, common in recommendation systems.

Steps

1. Data collection: gather detailed profile information from volunteers and organizations.
2. Feature extraction: extract relevant features such as skills, interests, and availability.
3. Similarity calculation: calculate the similarity between volunteer profiles and opportunity requirements using cosine similarity or another distance metric.
4. Ranking: rank opportunities for each volunteer based on the similarity scores.
5. Recommendation: recommend the top-ranked opportunities to volunteers.

Advantages

- Highly personalized matches based on detailed profile attributes.
- Flexibility in handling various types of data (skills, interests, availability).

Challenges

- Requires comprehensive and up-to-date profile information.
- May not account for dynamic changes in volunteer availability or interest.

3.2.2 Collaborative Filtering Algorithm

Algorithm overview

Collaborative filtering is used to leverage the preferences and behaviors of similar volunteers to make recommendations. This approach can be particularly effective when volunteers have limited profile information or when new opportunities are added.

Steps

1. Interaction data collection: collect data on past volunteer engagements, such as which opportunities volunteers applied for or completed.

2. User-Item matrix creation: create a matrix representing volunteers (users) and opportunities (items), with interactions indicating interest or engagement.
3. Similarity calculation: calculate similarities between volunteers based on their interactions using methods like Pearson correlation or cosine similarity.
4. Prediction: predict a volunteer's interest in an opportunity based on the interests of similar volunteers.
5. Recommendation: make recommendations of opportunities with the highest predicted interest scores.

Advantages

- Effective in recommending new opportunities based on past behaviors.
- Can provide recommendations even with sparse profile information.

Challenges

- Requires a substantial amount of interaction data to generate accurate recommendations.
- May struggle with the "cold start" problem for new volunteers or new opportunities with no interaction history.

3.2.3 Hybrid Approach Algorithm

Algorithm overview

The hybrid approach combines both profile-based matching and collaborative filtering to leverage the strengths of each method and mitigate their weaknesses. This approach ensures a more robust and comprehensive matching process.

Steps

1. Data integration: integrate profile information and interaction data into a unified dataset.
2. Dual matching process:
 - Profile-based matching: perform profile-based matching to generate an initial set of recommendations.
 - Collaborative filtering: apply collaborative filtering to refine and complement the initial recommendations.
3. Weighted scoring: combine the results from both methods using a weighted scoring system to balance profile similarity and collaborative predictions.
4. Ranking and recommendation: rank the opportunities based on the combined scores and recommend the top options to volunteers.

Advantages

- Combines the benefits of both profile-based and collaborative filtering methods.
- Provides more accurate and reliable recommendations by utilizing multiple data sources.

Challenges

- More complex implementation and computationally intensive.
- Requires careful tuning of the weighting system to balance the influence of each method.

3.2.4 Flowchart of Hybrid Algorithm Approach

To visually represent the hybrid algorithm approach, the following flowchart illustrates the process:

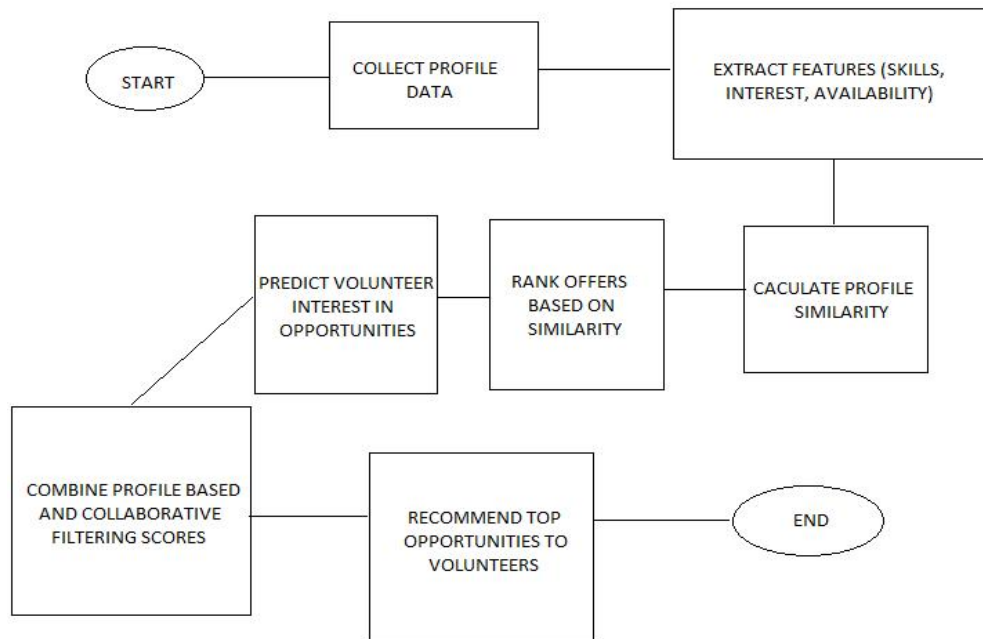


Figure 3.3 Hybrid algorithm approach

This flowchart captures the essential steps of the hybrid algorithm, showcasing the integration of profile-based matching and collaborative filtering to produce optimal volunteer-opportunity matches.

3.3 Architecture methods

In developing a robust volunteer platform, a well-designed architecture is essential to ensure scalability, security, and performance. This section outlines the system architecture and the approaches used to develop, integrate, and deploy the platform,

specifically tailored for a web-based solution using Django, RESTful APIs, SQLite, and datasets from Kaggle.

3.3.1 Architecture for solution and production

Overview

The architecture for the volunteer platform is composed of multiple layers and components designed to handle various aspects of the system, from data storage and processing to user interaction and machine learning. The architecture is divided into several key components:

1. Client Layer
2. Application Layer
3. Data Layer
4. Machine Learning Layer

Client Layer

The client layer involves the user interface (UI) components that interact with end-users (both volunteers and organizations). This layer includes:

- Web application: a responsive web interface accessible via browsers, allowing users to register, log in, search for opportunities, and manage their profiles.

Technologies used for client layer

- HTML, CSS, Javascript.
- Bootstrap.
- Ajax and JQuery.
- React.js or Vue.js (for frontend framework).

Application layer

The application layer handles the core logic of the platform, processing user requests, and managing interactions between the client layer and other system components. This layer includes:

- Django framework: serves as the backend framework, handling all server-side logic and database interactions.
- RESTful APIs: provides endpoints for client-server communication, enabling the web application to interact with backend services.

Technologies used

- Django (Python)
- Django REST Framework (for building RESTful APIs)

Data layer

The data layer manages all data storage and retrieval operations, ensuring data integrity and security. This layer includes:

- Database: store user profiles, volunteer opportunities, and interaction data (SQLite DB).
- Data warehouse: aggregate and preprocess data from Kaggle datasets for machine learning purposes (SQLite DB).

Technologies used

- SQLite (lightweight database for development and testing)

Machine learning layer

The machine learning layer is responsible for developing, training, and deploying machine learning models used for volunteer matching and recommendation. This layer includes:

- Feature engineering: process and prepare data for machine learning.
- Model training: develop and trains machine learning models using historical data.
- Model serving: deploy trained models to serve real-time predictions.

Technologies used:

- Python, TensorFlow, Scikit-learn (for model development)
- Pandas, NumPy (for data preprocessing and analysis)

3.3.2 Volunteer matching process:

The volunteer matching process approaches the problem of accurately matching users within five steps. This method addresses the problem directly to reduce redundancy in the matching process and to help decrease the time taken to complete a

successful match-making sequence, thereby reducing the workload placed on the system.

First Stage: Necessary data regarding the user for whom the matching is made is collected from the SQLite Database. This data, including attributes such as working experience, proficiency, past experience, industry, age, demographics, and health status, is arranged into a data frame and passed to the volunteer matching prediction function.

Second Stage: This stage involves how the system classifies or predicts the user into a class to accurately decide which classes of volunteer opportunities would be the perfect fit for our volunteer. It does this by using data about the user provided as parameters during its call with the model built earlier from training. Using the libjob module, our developed model returns a predicted value, which in turn represents a set of opportunity classes.

Third Stage: At this stage, the predicted value obtained from the earlier model is taken, and this value is then used to search for volunteer opportunities within the database. It does this by using the predicted value, which in turn represents the category, nature, or class of the volunteer opportunity, as a criterion for fetching job models from the database.

Fourth Stage: The job results obtained after the search are parsed and prepared into an array list of volunteer opportunities to be rendered in our user interface. These jobs are then JSON serialized to be readable by our HTML and returned as a JSON response by our Django.

Fifth Stage: The JSON serialized jobs are read and looped through on our frontend. Users are then able to select and apply for any volunteer offer that fits their schedule.

Algorithm: Volunteer Matching Process

Step 1: Collect User Data

- 1.1 Collect necessary user data from the SQLite Database.
- 1.2 Gather attributes such as working experience, proficiency, past experience, industry, age, demographics, and health status.
- 1.3 Arrange the collected data into a structured format (e.g., data frame).

Step 2: Predict Volunteer Opportunities

- 2.1 Use the collected user data as input parameters for the volunteer matching prediction function.
- 2.2 Utilize the pre-trained model to predict the most suitable volunteer opportunities for the user.

2.3 Obtain the predicted value representing the set of opportunity classes based on the user's attributes.

Step 3: Retrieve Matching Opportunities

3.1 Take the predicted value obtained from the model and use it to search for volunteer opportunities within the database.

3.2 Use the predicted value (representing opportunity classes and volunteer category) as criteria for fetching job models from the database.

3.3 Retrieve matching volunteer opportunities based on the predicted value.

Step 4: Prepare Job Results

4.1 Parse and format the retrieved job results into a structured array list of volunteer opportunities.

4.2 Serialize the volunteer opportunities into JSON format for ease of rendering.

4.3 Return the JSON serialized job results as a response from the Django backend.

Step 5: Display and Apply for Opportunities

5.1 Read and loop through the JSON serialized job results on the frontend.

5.2 Display the volunteer opportunities to the user through the user interface.

5.3 Allow users to select and apply for any volunteer offer that fits their schedule.

3.3.3 Methods for integration and production

Integration approach

- Monolithic architecture: a monolithic architecture using Django is implemented, allowing easier development and integration.
- API Gateway: utilize Django REST Framework to manage API requests and ensure smooth client-server interactions.

Deployment Strategy:

- Continuous integration and deployment (CI/CD): implement CI/CD pipelines using tools like GitHub Actions to automate testing, building, and deployment processes.
- Containerization: use Docker to containerize the application, ensuring consistency across development, testing, and production environments.
- Hosting: deploy the platform on PythonAnywhere hosting services to ensure high availability and performance.

Security measures:

- Data encryption: Ensure all sensitive data are encrypted both in transit (using HTTPS) and at rest.

- Access control: implement robust access control mechanism using Django's authentication and authorization features to restrict unauthorized access to system resources.
- Regular audits: conduct regular security audits and vulnerability assessments to identify and mitigate potential threats.

Scalability and Performance

- Horizontal scaling: configure horizontal scaling policies to handle increased traffic and workload by adding more server instances.
- Load balancing: utilize load balancers to distribute incoming traffic across multiple servers to ensure high availability and reliability.
- Caching: implement caching strategies using Django's caching framework Redis to reduce load on the database and improve response times.

CHAPTER FOUR

DESIGN AND IMPLEMENTATION

4.1 Introduction

This chapter details the design and implementation phases of the volunteering matching platform (Civic Pulse). It covers the architectural design, implementation, installation and configuration of necessary tools, the development of various components, and the implementation of core functionalities. The goal is to provide a comprehensive understanding of the practical steps taken to realize the project, from setting up the technical infrastructure to implementing the matching algorithm and user-facing features.

4.2 The Architecture, Installation and Configuration tools

4.2.1 The Architecture

The volunteer matching platform is developed using a modern Django-React architecture, ensuring efficient user experience. The frontend of the application is built with React.js, HTML(Hypertext Markup Language), CSS(Cascading Style Sheet), JQuery, Ajax, and bootstrap which provides a dynamic and responsive user interface. This allows volunteers and organizations to interact with the platform in real-time, offering functionalities such as searching for opportunities, managing profiles, and receiving notifications.

The back-end is powered by Django, a robust Python web framework that handles the core application logic, including user authentication, data management, and the volunteer matching algorithms. Django serves as the backbone of the platform,

managing database operations through its ORM (Object-Relational Mapping) and providing RESTful API endpoints that the React frontend consumes.

Data is stored in an SQLite database during development and PostgreSQL in production, ensuring scalability and reliability. The platform's machine learning engine, integrated within the Django back-end, processes user data to provide personalized volunteer matching and recommendations. This engine leverages data analytics to optimize the matching process, ensuring that volunteers are paired with the most suitable opportunities.

Real-time communication is facilitated using Django Channels, enabling instant notifications and messaging between volunteers and organizations. This component ensures that users are promptly informed about new opportunities and updates.

The entire application is hosted on PythonAnywhere, a platform that supports Django applications, providing a scalable and accessible environment for the platform to operate. This architecture allows for efficient management of volunteer activities and enhances the overall user experience by leveraging modern web technologies.

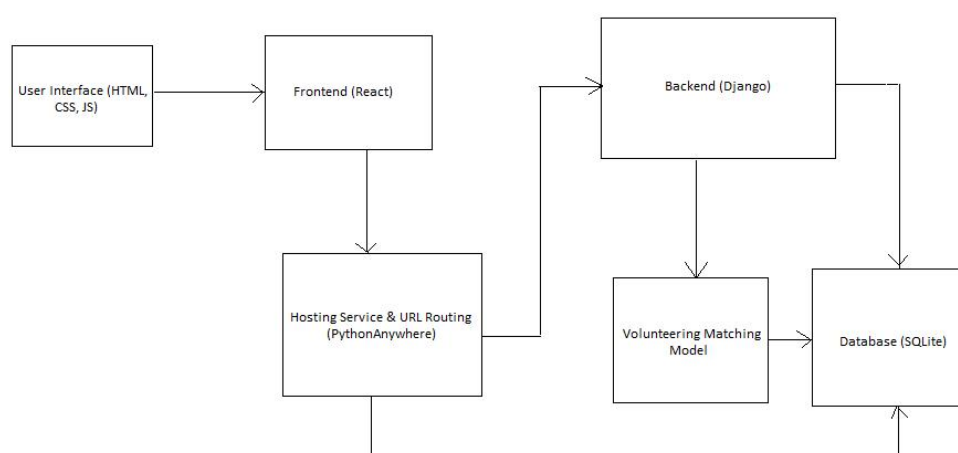


Fig 4.1 Django-React architecture for volunteering platform

4.2.2 Installation

Prerequisites

Before starting the installation, ensure you have the following prerequisites installed on your system:

1. Python 3.8 or higher
2. Node.js and npm (Node package manager)
3. Django 3.1 or higher
4. Create React App CLI
5. SQLite (for development) or PostgreSQL (for production)
6. Git (for version control)

Installing Prerequisites

1. Downloading Python 3.8 or higher

Linux: use your package manager, for example, on Ubuntu

```
sudo apt update  
sudo apt install python3.8 python3.8-venv python3.8-dev
```


1. Node.js and npm

- **Linux:** download and install from the official Node.js website:
<https://nodejs.org/download/>.

```
sh  
  
brew install node
```

2. Create React app CLI (Command line interface): install globally via npm

```
npm install -g create-react-app
```

3. SQLite (for development)

- **Windows/macOS/Linux:** SQLite is included with Python, no separate installation is needed.

4. Git prerequisite installation

Linux: use your package manager.

```
sudo apt install git
```

Step-by-step installation guide

1. Setting up the back-end (Django)

Create a virtual environment

```
python3 -m venv env  
source env/bin/activate # On Windows use `env\Scripts\act
```

Install Django and other dependencies

```
pip install django djangorestframework
```

Start a new Django project

```
django-admin startproject civic_pulse  
cd civic_pulse
```

Configure Django settings:

- Update settings.py to include the new app and set up the database configuration:

```

INSTALLED_APPS = [
    ...
    'rest_framework',
    'matching',
]

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / "db.sqlite3",
    }
}

```

Create models and migrations:

- Define your models in matching/models.py.

```
from django.db import models
```

```
class Profile(models.Model):
```

```
    userEmail = models.EmailField(max_length=250, default="")
```

```
    userName = models.CharField(max_length=50, default="")
```

```
    userFullName = models.CharField(max_length=50, default="")
```

```
    userTitle = models.CharField(max_length=100, default="")
```

```
userContact = models.CharField(max_length=50, default="+ (000) 000-000-000")
```

```
userBio = models.CharField(max_length=500, default="")
```

```
userType = models.CharField(max_length=100, default="employee")
```

```
userDOB = models.CharField(max_length=100, default="-- -- --")
```

```
userLocation = models.CharField(max_length=100, default="-- -- --")
```

```
userGender = models.CharField(max_length=100, default="Male")
```

```
userProfilePicture = models.ImageField(upload_to='profile_picture/', null=True, blank=True)
```

```
userResume = models.FileField(upload_to='resume(cv)/', null=True, blank=True)
```

```
userPassword = models.CharField(max_length=500, default="no_active_password")
```

```
admin_conversation = models.CharField(max_length=500, default="none")
```

```
class Education(models.Model):
```

```
school = models.CharField(max_length=50)
```

```

degree = models.CharField(max_length=50)

field = models.CharField(max_length=50)

startDate = models.CharField(max_length=25)

endDate = models.CharField(max_length=25)

description = models.CharField(max_length=5000)

userProfile = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)

class Meta:

    db_table = 'Education'

class Skill(models.Model):

    skill_set = models.CharField(max_length=50)

    proficiency = models.CharField(max_length=10)

    userProfile = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)

```

```

class Meta:

    db_table = 'Skill Sets'


class Experience(models.Model):

    Experience_Title = models.CharField(max_length=50, default="")

    Experience_Employment_Type = models.CharField(max_length=50,
default="")

    Experience_Organization_Name = models.CharField(max_length=50,
default="")

    Experience_Description = models.CharField(max_length=500, default="")

    Experience_Location = models.CharField(max_length=50, default="")

    Experience_Start_Date = models.CharField(max_length=50, default="")

    Experience_Close_Date = models.CharField(max_length=50, default="")

    userProfile = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)


class Meta:

    db_table = 'Working Experience'

```

```

class JobPost(models.Model):

    job_title = models.CharField(max_length=150)

    job_description = models.CharField(max_length=150)

    job_type = models.CharField(max_length=50, default="On-Site")

    job_poster = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)

    job_pay = models.CharField(max_length=200, default="")

    job_location = models.CharField(max_length=100, default="United States")

    job_external = models.CharField(max_length=50, default="none")

    job_external_link = models.CharField(max_length=5000, default="___")


class Meta:

    db_table = 'Job Post'


class JobCategory(models.Model):

    category_name = models.CharField(max_length=150)

```

```
job = models.ManyToManyField('JobPost', default=[0])
```

```
class Meta:
```

```
    db_table = 'Job Category'
```

```
class JobRequirement(models.Model):
```

```
    requirement = models.CharField(max_length=500)
```

```
    job = models.ForeignKey('JobPost', on_delete=models.CASCADE, default=1)
```

```
class Meta:
```

```
    db_table = 'Job Requirement'
```

```
class JobRole(models.Model):
```

```
    role = models.CharField(max_length=500)
```

```
    job = models.ForeignKey('JobPost', on_delete=models.CASCADE, default=1)
```

```
class Meta:
```



```
db_table = 'Job Role'
```

```
class Application(models.Model):
```

```
    job = models.ForeignKey('JobPost', on_delete=models.CASCADE, default=1)
```

```
    job_applicant = models.ForeignKey('Profile', on_delete=models.CASCADE,  
default=1)
```

```
    application_status = models.CharField(max_length=50, default="pending")
```

```
    interview_date = models.CharField(max_length=150, default="")
```

```
    interview_time = models.CharField(max_length=150, default="")
```

```
    interview_location_url = models.CharField(max_length=1000)
```

```
    interview_online = models.CharField(max_length=150, default="false")
```

```
class Meta:
```

```
    db_table = 'Application'
```

```
class JobNotification(models.Model):
```

```
    job_application = models.ForeignKey('Application',  
on_delete=models.CASCADE, default=1)
```

```
user = models.ForeignKey('Profile', on_delete=models.CASCADE, default=1)
```

```
check_value = models.CharField(max_length=50, default="uncheck")
```

```
time_sent = models.DateTimeField(auto_now_add=True)
```

```
class Meta:
```

```
    db_table = 'Job Notification'
```

```
class Chat(models.Model):
```

```
    user = models.ManyToManyField('Profile', default=[0],  
related_name="chat_user_list")
```

```
    latest_update = models.CharField(max_length=50, default="none")
```

```
class Meta:
```

```
    db_table = 'Chat'
```

```
# chat messages
```

```
class Chatmessage(models.Model):
```

```
sender = models.ForeignKey('Profile', on_delete=models.CASCADE, default=1,  
related_name="chat_sender")
```

```
chat = models.ForeignKey('Chat', on_delete=models.CASCADE, default=1,)
```

```
message = models.TextField(max_length=3000, default=None)
```

```
time_sent = models.DateTimeField(auto_now_add=True)
```

```
class Meta:
```

```
db_table = 'Chat Message'
```

```
# chat Report
```

```
class ChatReported(models.Model):
```

```
reporting_user = models.ForeignKey('Profile', on_delete=models.CASCADE,  
default=1, related_name="report_sender")
```

```
chat = models.ForeignKey('Chat', on_delete=models.CASCADE, default=1,)
```

```
time_reported = models.DateTimeField(auto_now_add=True)
```

```
class Meta:
```

```
db_table = 'Chat Reported'
```

- Run migrations

```
python manage.py makemigrations  
python manage.py migrate
```

Create API endpoints:

- Define serializers and views for your models in `matching/serializers.py` and `matching/views.py`.
- Configure URL routing in `matching/urls.py` and include it in the main `urls.py`.

Run the Django server:

```
python manage.py runserver
```

4.3 System requirements

4.3.1 Hardware requirements

For both development and deployment, the following hardware requirements should be met to ensure optimal performance and reliability of the Civic Pulse volunteer matching platform:

Development environment:

- Processor: Intel Core i5 or equivalent.

- Memory: 8 GB RAM.
- Storage: 256 GB SSD.
- Network: stable internet connection.

Production environment (PythonAnywhere hosting server):

- Processor: Intel Xeon or equivalent multi-core processor.
- Memory: 16 GB RAM or higher.
- Storage: 500 GB SSD or higher for handling data storage and faster read and write operations.
- Network: high-speed internet connection with at least 1 Gbps(Gigabyte per second) bandwidth.
- Backup: regular backup system for data redundancy.

4.3.2 Software requirements

The following software requirements must be met for both the development and production environments:

Development environment:

- Operating system: windows 10/11, macOS 10.15 or higher, or any modern Linux distribution (e.g., Ubuntu 20.04)
- Python: version 3.8 or higher.
- Node.js: version 14 or higher.
- npm (Node package manager): included with Node.js installation.
- Django: version 3.1 or higher.
- Django REST framework: version 3.12 or higher.
- React: latest stable version.
- Git: version control system.
- DE/Text Editor: visual Studio Code, PyCharm, or any preferred code editor.
- Browser: latest versions of Chrome, Firefox, or Safari for testing.

Production environment

- Operating system: Ubuntu 20.04 LTS (recommended) or any stable server-grade OS.
- Web server: Nginx or Apache.
- Application server: Gunicorn (for serving the Django application).

- Database: SQLite for development; PostgreSQL for production.
- Python: version 3.8 or higher.
- Node.js: version 14 or higher.
- Django: version 3.1 or higher.
- Django REST framework: version 3.12 or higher.
- React build: static files served from Nginx or Apache.
- SSL certificate: for secure HTTPS connections.
- Backup tools: automated backup tools and scripts.
- CI/CD tools: GitHub Actions, Travis CI, or Jenkins for continuous integration and deployment.

4.4 Civic-pulse platform design

1. Landing page: the landing page is designed to introduce new and returning users to the services offered by Civic-pulse as shown in Fig. 4.2. It contains important hyperlinks such as "Login," "Sign Up," and "Dashboard," enabling users to navigate the platform easily.

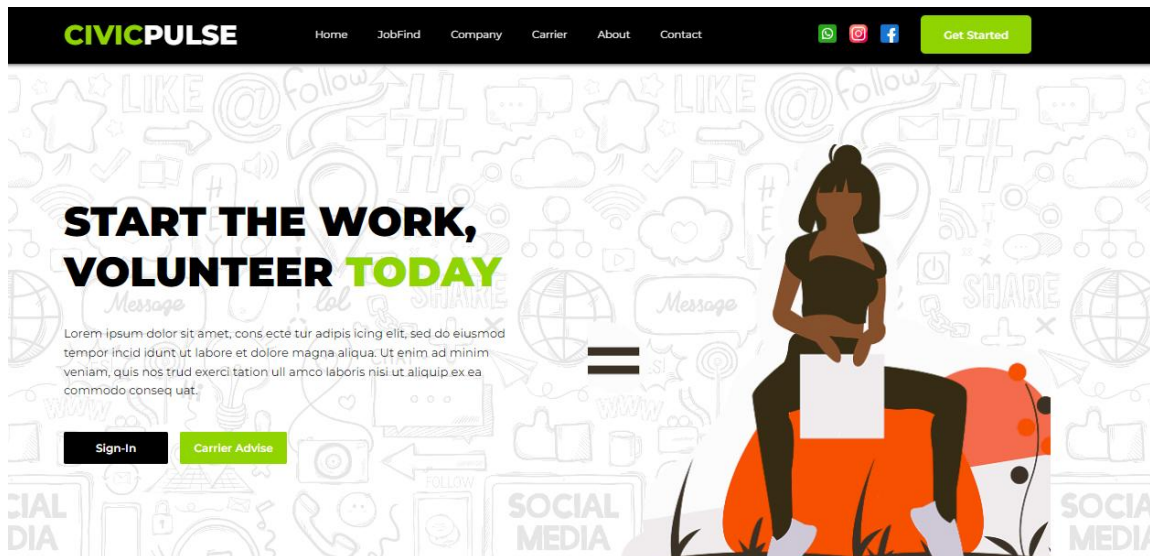


Fig. 4.2:Landing page

2. Sign-up company page: the company sign-up page allows new companies looking to recruit volunteers for their jobs to register. Registration is facilitated through an HTML form, with the data submitted via a POST request to the Django view module of our back-end. Here, the information received from the form is processed and validated before a new company instance is created within our SQLite database.

Fig. 4.3:Sign-up company page

3. Sign-up volunteer page: the volunteer sign-up page enables new users seeking volunteering opportunities to register for an account under the Civic Pulse platform. Similarly to the company sign-up process, registration occurs through an HTML form, with the data transmitted through a POST request to the Django view module of our back-end. The received information is then processed and validated before a new user instance is created within our SQLite database.

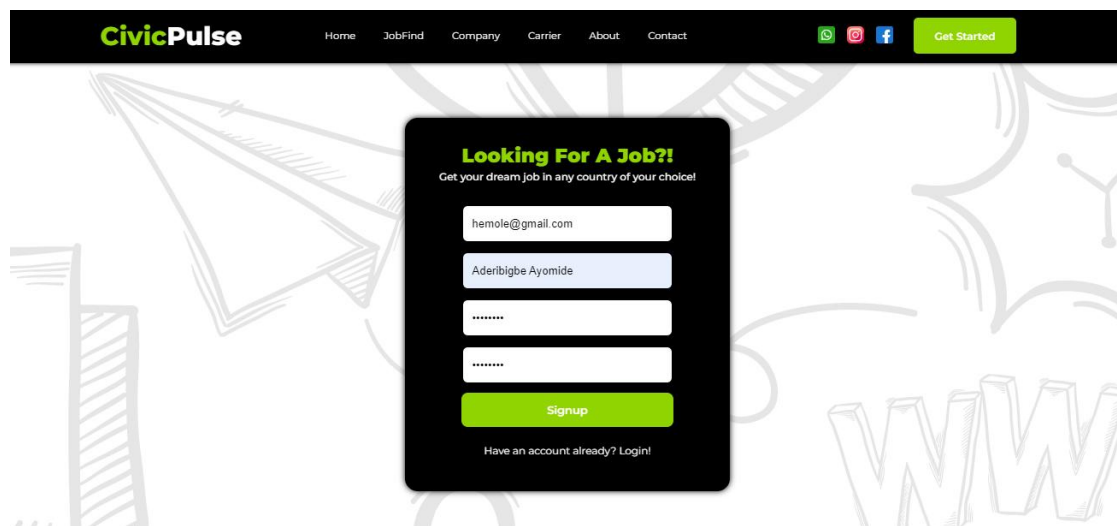


Fig. 4.4:Sign-up volunteer page

4. Login page: the login page grants access to previously registered users and companies, utilizing the Django authentication model.

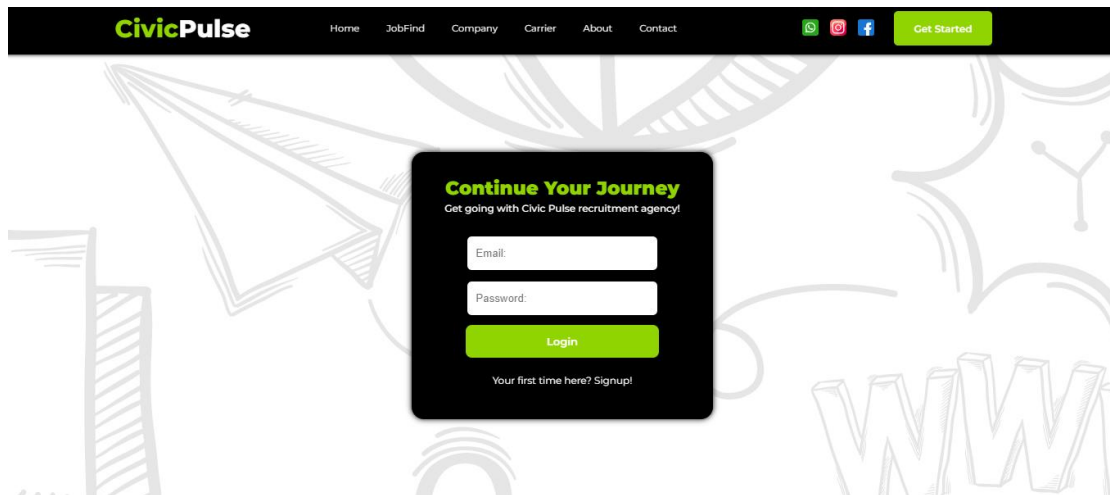


Fig. 4.5: Login page

5. User dashboard: the user dashboard provides access to the latest volunteering opportunities through the volunteering matching model. Below is an algorithmic approach to the volunteer matching process conducted by our platform.

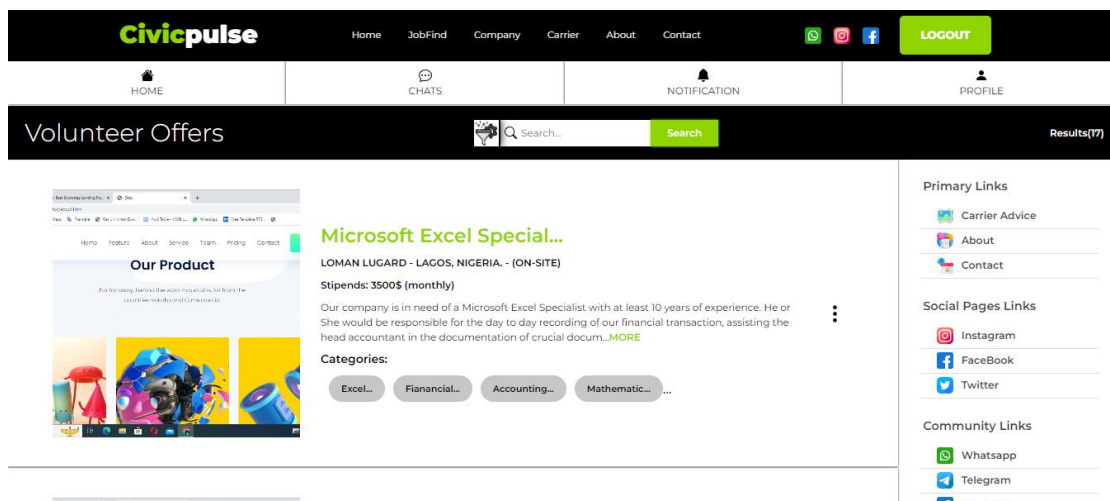


Fig. 4.6: User dashboard

Algorithm: volunteer matching process

Step 1: collect user data

- 1.1 Collect necessary user data from the SQLite Database.
- 1.2 Gather attributes such as working experience, proficiency, past experience, industry, age, demographics, and health status.
- 1.3 Arrange the collected data into a structured format (e.g., data frame).

Step 2: predict volunteer opportunities

- 2.1 Use the collected user data as input parameters for the volunteer matching prediction function.
- 2.2 Utilize the trained model to predict the most suitable volunteer opportunities for the user.
- 2.3 Obtain the predicted value representing the set of opportunity classes based on the user's attributes.

Step 3: retrieve matching opportunities

- 3.1 Take the predicted value obtained from the model and use it to search for volunteer opportunities within the database.

3.2 Use the predicted value (representing opportunity classes and volunteer category) as criteria for fetching job models from the database.

3.3 Retrieve matching volunteer opportunities based on the predicted value.

Step 4: prepare job results

4.1 Parse and format the retrieved job results into a structured array list of volunteer opportunities.

4.2 Serialize the volunteer opportunities into JSON format for ease of rendering.

4.3 Return the JSON serialized job results as a response from the Django back-end.

Step 5: display and apply for opportunities

5.1 Read and loop through the JSON serialized job results on the frontend.

5.2 Display the volunteer opportunities to the user through the user interface.

5.3 Allow users to select and apply for any volunteer offer that fits their schedule.

Aside from precisely providing volunteering opportunities that matches users interest, the user dashboard also serves as a navigation to other functionalities accessible by users on the platform. Example are the volunteering application process, live chat feature with potential companies, notifications on job applications, profile which allows user to build their professional image to enhance their likelihood of getting the opportunities applied for.

6. Volunteering application page: this page enables users to interact and apply for recommended jobs through the platform. The sole criterion for applying to a volunteering opportunity is that all users must have uploaded and completed their professional profile. This can be achieved by accessing the profile tab on the user dashboard. Professional information is mandatory as it provides companies with insights into the candidates applying for the volunteering opportunities hosted by them on the platform

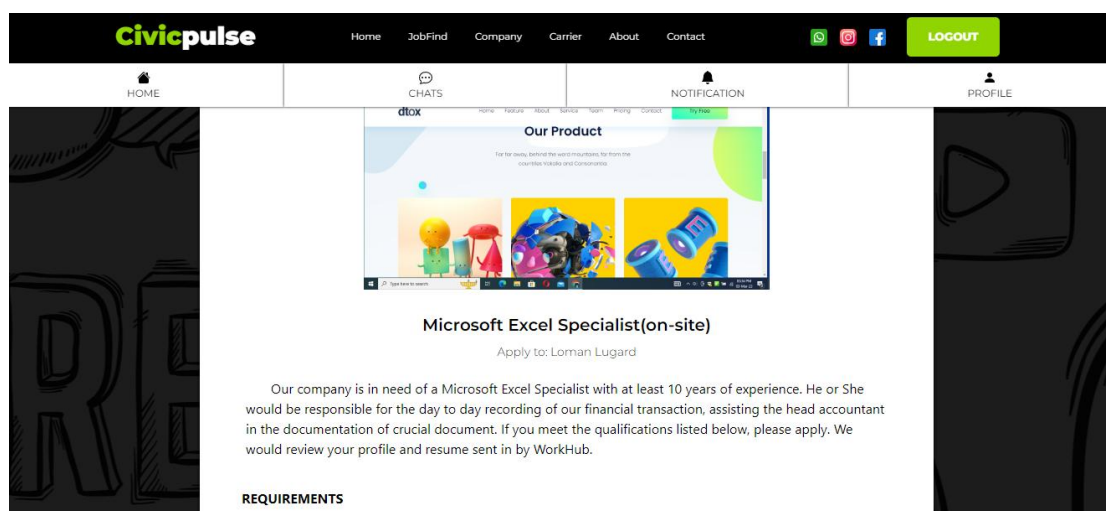


Fig. 4.7: Volunteering application page

7. Live chat page: selected applicants can receive their interview information and inquire further about their acceptance from the company through the live chat page. This page is designed with real-time messaging updates using AJAX, allowing clients to communicate with the server-side application in real time. This enables them to communicate with companies without being geographically restricted by distance.

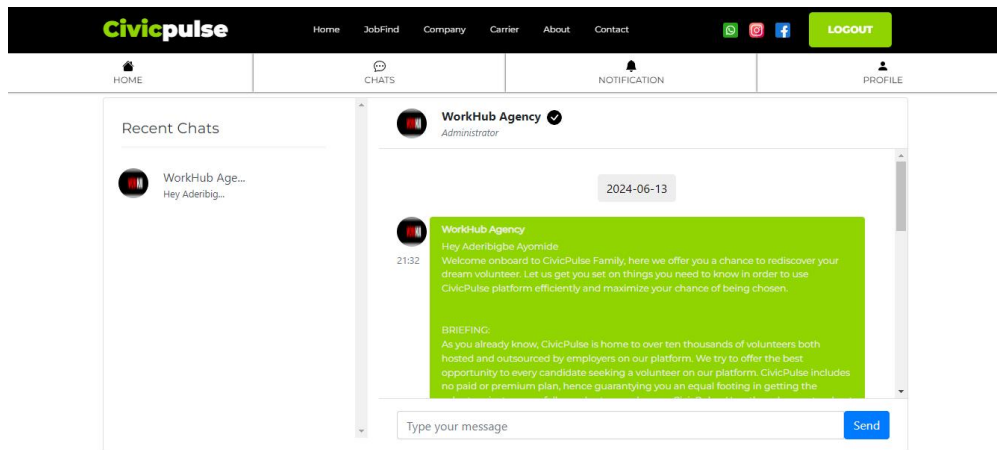


Fig. 4.8: Live chat page

8. Notification page: the notification page enables users to stay in touch and updated regarding recently applied openings. Users can see whether they have been accepted or rejected for an opening they applied for.

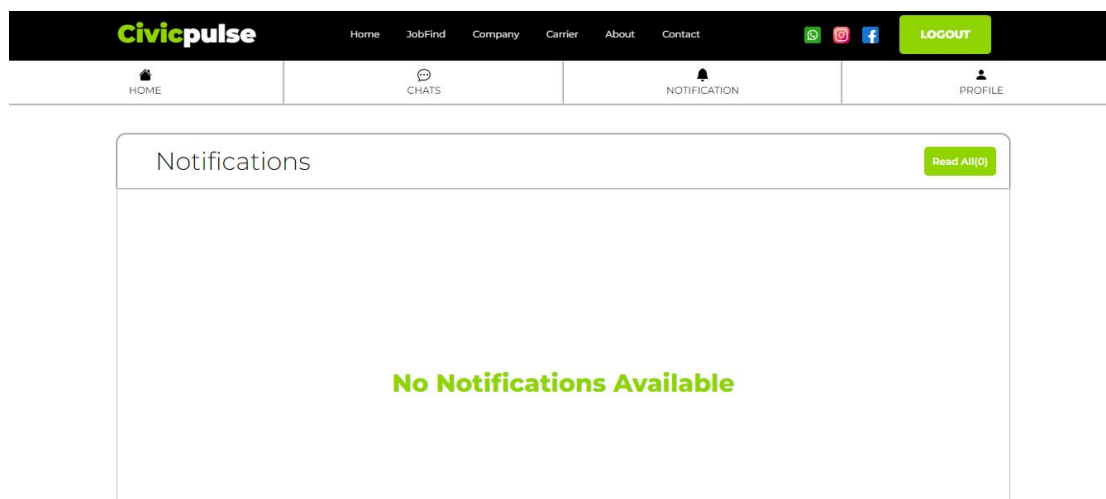


Fig. 4.9: Notification page

9. Profile page: the profile page not only allows users to view their professional profile but also to update their professional status. This enables them to keep their professional record up to date by editing and modifying previously uploaded data such as working experience, skills, education, resume, location, gender, bio, and more. This is facilitated through HTML forms, allowing users to post new information directly into the database.

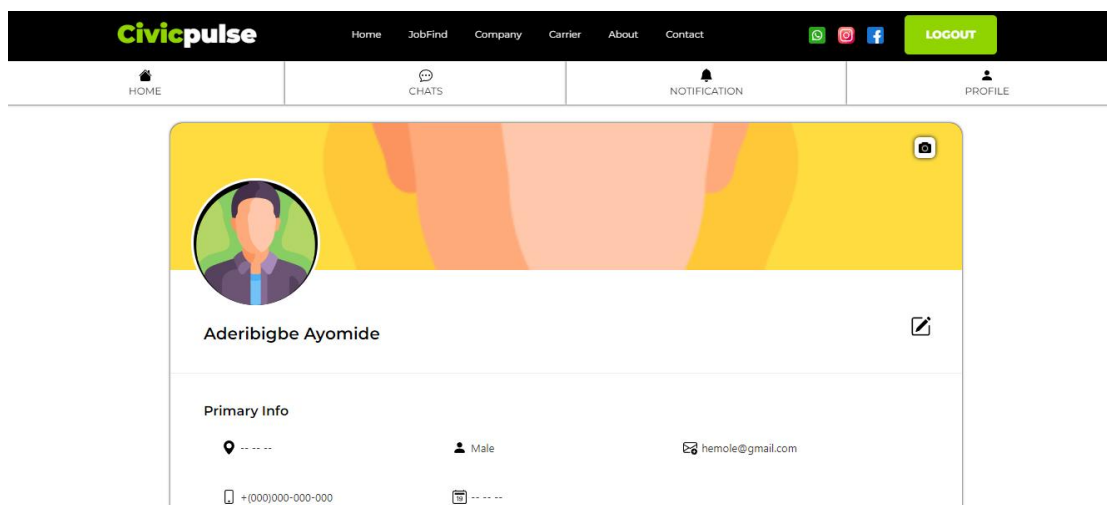


Fig. 4.10: Profile page

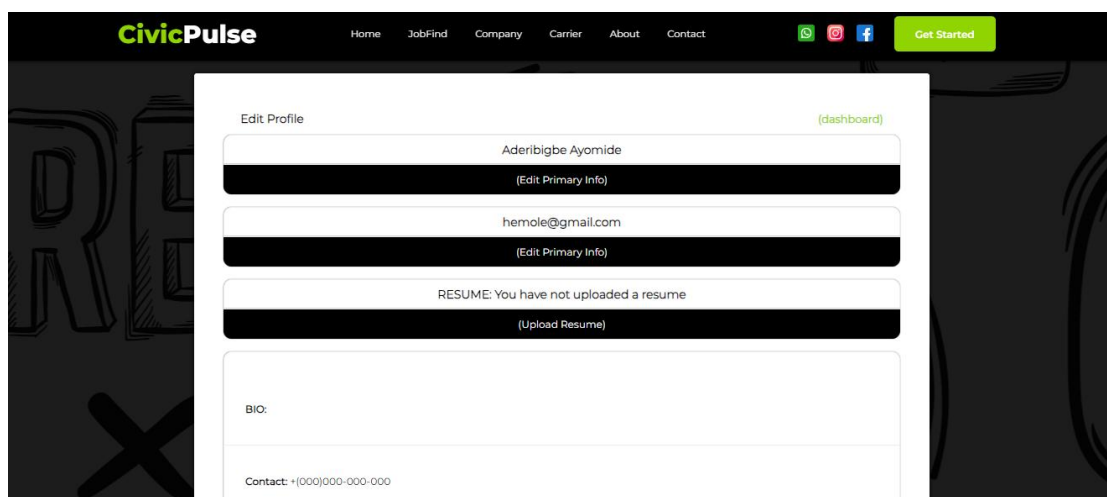


Fig. 4.11: Update profile page

10. Company dashboard page: the company dashboard is where volunteering opportunities are created and posted into the database, enabling users with similar interests to be recommended these volunteering opportunities. Companies can also access posted volunteering opportunities to view, accept, and reject applicants depending on the specifics of their opening.

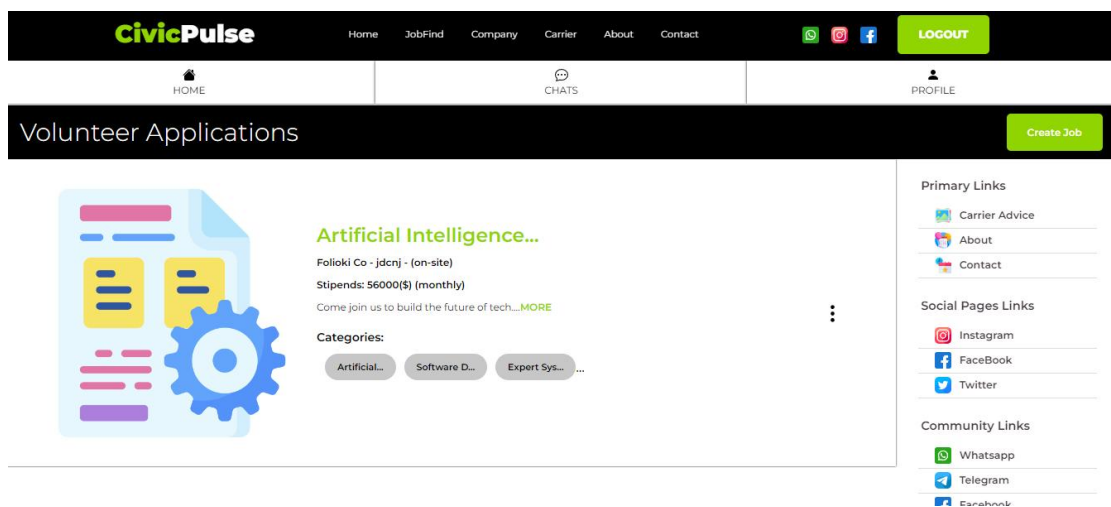


Fig. 4.12:Company dashboard page

4.5 Analysis on input used

The platform is data intensive, meaning all volunteering recommendations are fully dependent on data inputted by users during the lifespan of their usage of Civic Pulse. We would be discussing the various sections data are collected from user and what was their usage in the overall volunteering matching process.

Authentication bio-data input: this data is collected when users intends to sign up or login to civic pulse. During the sign-up process data collected are use to create an account under the consent of the users. This data is further used to classify the user into a class based on the user interests in the professional field.

Authentication Bio-data Input Process for Civic Pulse

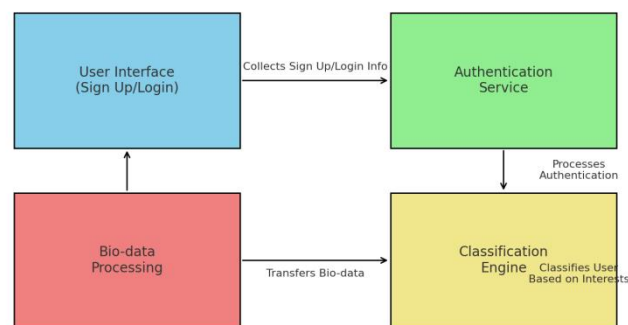


Fig 4.13 Authentication bio-data input process for Civic Pulse

Profile update input: data including educational background, working experience, skills, bio, location, resume, profile picture and much more are collected from users in order to build a comprehensive professional profile of who they are. This professional profile helps improve candidates chance of getting a volunteering opportunity.

Search input data: this data are search queries inputted by the user to search for a specific domain or categories of jobs. This input are processed by the back-end, where jobs are filtered based on this input. Search queries are not stored but dismissed after computational search.

Volunteering opening data: data including job description, roles, qualification, type, and much more are supplied by company to the platform to help facilitate the creation

of volunteering openings. Volunteering openings data are used to enable accurate recommendation of volunteering opportunities to users.

4.6. Analysis on output obtained

The outputs obtained from our platforms are the end results of the whole process performed by Civic pulse platform. The scope of civic pulse is to enable users find volunteering opportunities through the effective use of our volunteering matching process, we'd be discussing those set of outputs components that contributes towards achieving the aim of the platform.

Volunteering recommendation output: the primary aim of the platform is to effectively recommend openings to users who matches the opportunities. This done by a sequence of data gathering and prediction through platform to accurately recommend opportunities. This sequence involves storing user data inputted at the sign-up process, this data is then quantified and group according to attributes, during a request of jobs or a search the data are passed into the volunteering matching model where the class or category of jobs synonymous to users are fetched and recommended to users.

Openings creation output: to effectively recommend opportunities to users, adequate or substantial numbers of opening must be created on the platform by companies. To do this, companies are provided job creation functionality on their dashboard, where they are to fill in the details of their volunteering openings to output an opening on the platform.

4.7 Making use of the program

The Civic Pulse platform is designed to provide an intuitive and efficient experience for both volunteers and organizations. Here's how different users can make the most of the platform:

For volunteers

1. Registration and profile creation

- Sign up: visit the Civic Pulse homepage and click on the 'Sign Up' button. Fill in the required details to create an account.

- Profile setup: complete your profile by adding personal information, skills, interests, availability, and location. This helps in matching you with suitable volunteer opportunities.

2. Searching for opportunities

- Browse opportunities: use the search bar or browse through categories to find volunteer opportunities that match your skills and interests.

- Filters: apply filters such as location, type of activity, and availability to narrow down the search results.

3. Application process:

- Apply: click on the 'Apply' button for opportunities that interest you. Provide any additional information or answer questions posed by the organization.

- Track applications: monitor the status of your applications through the 'Notification tab' section in your dashboard.

For organizations

1. Registration and profile creation

- Sign up: Visit the Civic Pulse homepage and click on the 'Organization Sign Up' button. Fill in the necessary details to create an organizational account.

- Profile setup: complete your organization's profile by adding information about your mission, activities, and volunteer needs.

2. Posting opportunities

- Create post: navigate to the 'Post an Opportunity' section. Fill in the details of the volunteer opportunity, including description, required skills, location, and duration.

- Publish: once all details are filled in, publish the opportunity to make it visible to potential volunteers.

3. Managing applications

- Review applications: access the 'Manage Applications' section to review applications from interested volunteers.

- Communicate: use the messaging feature to communicate with applicants, ask additional questions, and schedule interviews if needed.

- Select volunteers: approve or reject applications based on your criteria. Selected volunteers will receive notifications.

Platform features

- Dashboard: both volunteers and organizations have personalized dashboards. Volunteers can track their applications and upcoming tasks, while organizations can manage opportunities and volunteer applications.

- Job application board: volunteers have access to jobs listing which are streamlined to their personal interest through the volunteering matching model.

- Live chats: both volunteers and organizations have live chats to help facilitate communication between company and candidates on a job opening.

- Notifications: real-time notifications keep users informed about new opportunities, application status, and messages.

- Security: the platform ensures data security through encrypted communications, secure authentication, and regular updates.

- User support: access support through the help center or contact support staff for any assistance needed.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

The Civic Pulse volunteer platform successfully addresses the growing need for streamlined and efficient volunteer management systems. By leveraging the power of Django for back-end development and React for a responsive front-end, the platform provides a seamless user experience for both volunteers and organizations. The implementation of machine learning algorithms for matching volunteers to opportunities enhances the relevance and satisfaction of volunteer placements, while real-time communication and data analytics tools ensure effective coordination and evaluation.

Throughout the development process, we focused on creating a scalable and sustainable solution capable of accommodating future growth and evolving needs. The use of Django's robust framework and the flexibility of React's component-based architecture allows for easy updates and feature enhancements. Additionally, the platform's deployment on PythonAnywhere ensures reliable hosting and maintenance.

The comprehensive approach to volunteer matching, encompassing profile creation, opportunity search, application management, and real-time engagement, sets Civic Pulse apart as a pioneering solution in the volunteer management space. By addressing the common challenges faced by organizations in recruiting and managing volunteers, Civic Pulse facilitates meaningful connections and maximizes the impact of volunteer efforts.

5.2 Recommendation

Based on the insights gained during the development and implementation of Civic Pulse, we have successfully developed and implemented a platform that facilitates volunteering through digital means. To enhance their chances of standing out during applications, volunteers are advised to update their professional profiles to the best of their abilities. For companies posting volunteering opportunities, it is recommended that the openings are well-detailed and simplified to enable users to understand the requirements and roles they would be undertaking.

Even though we are able to effectively recommend available volunteer openings, the usability of the platform is still greatly dependent on the flow of jobs and applicants. To make this a reality, the platform needs a source of traffic to ensure the effective utilization of the resources implemented.

There are additional features that can further improve the platform. One is the effective tracking of volunteers' daily tasks, which would enable companies to keep track of their volunteers throughout their deployment. Another beneficial feature would be the provision of a digital workspace where people can volunteer online and work remotely from their homes. Additionally, incorporating automated email notifications into the platform would help users stay updated about the latest volunteering opportunities in their geographical area. I believe that if these features are implemented, they would contribute significantly to the overall effectiveness of the volunteering matching platform.

REFERENCES

Brown, A. (2019). The Role of Online Platforms in Volunteer Management. *Journal of Nonprofit Technology*, 5(2), 87-102.

Garcia, M. (2018). Data-Driven Approaches to Volunteer Management. *Journal of Volunteer Resources Management*, 4(3), 112-130.

Garcia, M. (2020). UN Volunteers: Promoting Peace and Development. *Journal of Global Development*, 9(1), 15-33.

Jackson, M. (2019). Community-Based Volunteerism: Historical Perspectives. *Journal of Community Engagement*, 10(1), 45-60.

Johnson, P., & Lee, H. (2017). Mobilizing Volunteers Globally: The Points of Light Approach. *International Journal of Volunteer Administration*, 4(1), 112-130.

Johnson, R. (2022). Leveraging Machine Learning for Volunteer Matching: Opportunities and Challenges. *Nonprofit Management Review*, 20(3), 301-315.

Roberts, L. (2017). Evolution of Volunteer Management Practices: A Historical Analysis. *Journal of Volunteer Administration*, 25(3), 189-204.

Smith, R. (2018). Connecting for Social Change: The Idealist Model. *Journal of Social Innovation*, 7(3), 45-60.

Smith, R. (2020). Virtual Volunteering: Opportunities and Challenges. *International Journal of Volunteer Administration*, 6(1), 45-60.

Taylor, K. (2021). Digital Transformation in Volunteer Management: Opportunities and Challenges. *Journal of Digital Innovation*, 8(2), 145-160.

APPENDIX

Setting.py (Configuration of Civic Pulse Django Platform)

```
from pathlib import Path
```

```
import os
```

```
# Build paths inside the project like this: BASE_DIR / 'subdir'.
```

```
BASE_DIR = Path(__file__).resolve().parent.parent
```

```
# Quick-start development settings - unsuitable for production
```

```
# See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/
```

```
# SECURITY WARNING: keep the secret key used in production secret!
```

```
SECRET_KEY = 'django-insecure-~!*i-_hiz9!umyr@sv3$epv=+f$2oi47&*x-z$nl1n*o1@crv0h'
```

```
# SECURITY WARNING: don't run with debug turned on in production!
```

```
DEBUG = True
```

```
ALLOWED_HOSTS = []
```

```
# Application definition
```

```
INSTALLED_APPS = [
```

```
    'django.contrib.admin',
```

```
    'django.contrib.auth',
```

```
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
```

```
    'django.contrib.messages',
```

```
    'django.contrib.staticfiles',
```

```
    'workapp'
```

```
]
```

```
MIDDLEWARE = [
```

```
    'django.middleware.security.SecurityMiddleware',
```

```
    'django.contrib.sessions.middleware.SessionMiddleware',
```

```
    'django.middleware.common.CommonMiddleware',
```

```

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT_URLCONF = 'workhub.urls'

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [BASE_DIR, "template"],

        'APP_DIRS': True,

        'OPTIONS': {

            'context_processors': [

                'django.template.context_processors.debug',

                'django.template.context_processors.request',

                'django.contrib.auth.context_processors.auth',

                'django.contrib.messages.context_processors.messages',

```

```
    ],  
  
    },  
  
    },  
  
]
```

```
WSGI_APPLICATION = 'workhub.wsgi.application'
```

```
# Database
```

```
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases
```

```
DATABASES = {
```

```
    'default': {
```

```
        'ENGINE': 'django.db.backends.sqlite3',
```

```
        'NAME': BASE_DIR / 'db.sqlite3',
```

```
    }
```

```
}
```

Password validation

<https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validators>

```
AUTH_PASSWORD_VALIDATORS = [

    {

        'NAME':

'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',

    },

    {

        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',

    },

    {

        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',

    },

    {

        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',

    },

]
```

```
# Internationalization
```

```
# https://docs.djangoproject.com/en/4.1/topics/i18n/
```

```
LANGUAGE_CODE = 'en-us'
```

```
TIME_ZONE = 'UTC'
```

```
USE_I18N = True
```

```
USE_TZ = True
```

```
# Static files (CSS, JavaScript, Images)
```

```
# https://docs.djangoproject.com/en/4.1/howto/static-files/
```

```
STATIC_URL = 'static/'
```

```
STATICFILES_DIRS = (os.path.join(BASE_DIR, 'static'),)
```

```
# Media Config
```

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

```
MEDIA_URL = '/media/'
```

```
# Default primary key field type
```

```
# https://docs.djangoproject.com/en/4.1/ref/settings/#default-auto-field
```

```
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Models.py (Civic Pulse Database Schema Configuration)

```
from django.db import models
```

```
class Profile(models.Model):
```

```
    userEmail = models.EmailField(max_length=250, default="")
```

```
    userName = models.CharField(max_length=50, default="")
```

```
    userFullName = models.CharField(max_length=50, default="")
```



```

userTitle = models.CharField(max_length=100, default="")

userContact = models.CharField(max_length=50, default="+ (000) 000-000-000")

userBio = models.CharField(max_length=500, default="")

userType = models.CharField(max_length=100, default="employee")

userDOB = models.CharField(max_length=100, default="-- -- --")

userLocation = models.CharField(max_length=100, default="-- -- --")

userGender = models.CharField(max_length=100, default="Male")

userProfilePicture = models.ImageField(upload_to='profile_picture/', null=True,
blank=True)

userResume = models.FileField(upload_to='resume(cv)/', null=True, blank=True)

userPassword = models.CharField(max_length=500,
default="no_active_password")

admin_conversation = models.CharField(max_length=500, default="none")

```

```

class Education(models.Model):

    school = models.CharField(max_length=50)

    degree = models.CharField(max_length=50)

    field = models.CharField(max_length=50)

```

```
startDate = models.CharField(max_length=25)

endDate = models.CharField(max_length=25)

description = models.CharField(max_length=5000)

userProfile = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)
```

```
class Meta:
```

```
    db_table = 'Education'
```

```
class Skill(models.Model):
```

```
    skill_set = models.CharField(max_length=50)
```

```
    proficiency = models.CharField(max_length=10)
```

```
    userProfile = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)
```

```
class Meta:
```

```
    db_table = 'Skill Sets'
```

```

class Experience(models.Model):

    Experience_Title = models.CharField(max_length=50, default="")

    Experience_Employment_Type = models.CharField(max_length=50, default="")

    Experience_Organization_Name = models.CharField(max_length=50, default="")

    Experience_Description = models.CharField(max_length=500, default="")

    Experience_Location = models.CharField(max_length=50, default="")

    Experience_Start_Date = models.CharField(max_length=50, default="")

    Experience_Close_Date = models.CharField(max_length=50, default="")

    userProfile = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)

```

```

class Meta:

```

```

    db_table = 'Working Experience'

```

```

class JobPost(models.Model):

```

```

    job_title = models.CharField(max_length=150)

```

```

    job_description = models.CharField(max_length=150)

```

```

job_type = models.CharField(max_length=50, default="On-Site")

job_poster = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1)

job_pay = models.CharField(max_length=200, default="")

job_location = models.CharField(max_length=100, default="United States")

job_external = models.CharField(max_length=50, default="none")

job_external_link = models.CharField(max_length=5000, default="___")


class Meta:

    db_table = 'Job Post'


class JobCategory(models.Model):

    category_name = models.CharField(max_length=150)

    job = models.ManyToManyField('JobPost', default=[0])


class Meta:

    db_table = 'Job Category'

```

```

class JobRequirement(models.Model):

    requirement = models.CharField(max_length=500)

    job = models.ForeignKey('JobPost', on_delete=models.CASCADE, default=1)


class Meta:

    db_table = 'Job Requirement'


class JobRole(models.Model):

    role = models.CharField(max_length=500)

    job = models.ForeignKey('JobPost', on_delete=models.CASCADE, default=1)


class Meta:

    db_table = 'Job Role'


class Application(models.Model):

    job = models.ForeignKey('JobPost', on_delete=models.CASCADE, default=1)

```

```
job_applicant = models.ForeignKey('Profile', on_delete=models.CASCADE,  
default=1)
```

```
application_status = models.CharField(max_length=50, default="pending")
```

```
interview_date = models.CharField(max_length=150, default="")
```

```
interview_time = models.CharField(max_length=150, default="")
```

```
interview_location_url = models.CharField(max_length=1000)
```

```
interview_online = models.CharField(max_length=150, default="false")
```

```
class Meta:
```

```
    db_table = 'Application'
```

```
class JobNotification(models.Model):
```

```
    job_application = models.ForeignKey('Application', on_delete=models.CASCADE,  
default=1)
```

```
    user = models.ForeignKey('Profile', on_delete=models.CASCADE, default=1)
```

```
    check_value = models.CharField(max_length=50, default="unchecked")
```

```
    time_sent = models.DateTimeField(auto_now_add=True)
```

```

class Meta:

    db_table = 'Job Notification'


class Chat(models.Model):

    user = models.ManyToManyField('Profile', default=[0],
related_name="chat_user_list")

    latest_update = models.CharField(max_length=50, default="none")


class Meta:

    db_table = 'Chat'


# chat messages

class Chatmessage(models.Model):

    sender = models.ForeignKey('Profile', on_delete=models.CASCADE, default=1,
related_name="chat_sender")

    chat = models.ForeignKey('Chat', on_delete=models.CASCADE, default=1,)

    message = models.TextField(max_length=3000, default=None)

```

```

time_sent = models.DateTimeField(auto_now_add=True)

class Meta:

    db_table = 'Chat Message'

# chat Report

class ChatReported(models.Model):

    reporting_user = models.ForeignKey('Profile', on_delete=models.CASCADE,
default=1, related_name="report_sender")

    chat = models.ForeignKey('Chat', on_delete=models.CASCADE, default=1,)

    time_reported = models.DateTimeField(auto_now_add=True)

class Meta:

    db_table = 'Chat Reported'

```

Urls.py (Civic Pulse App Url Configuration and Routing)

```

from django.urls import path

```



```
from . import views
```

```
urlpatterns = [
```

```
    path("", views.index, name='index'),
```

```
    path('about', views.about, name='about'),
```

```
    path('contact', views.contact, name='contact'),
```

```
    path('carrier_advice', views.carrier_advice, name='carrier_advice'),
```

```
    path('employee_signup', views.employee_signup, name='employee_signup'),
```

```
    path('dashboard', views.dashboard, name='dashboard'),
```

```
    path('employee_dashboard', views.employee_dashboard,  
name='employee_dashboard'),
```

```
    path('employee_process_job_application',  
views.employee_process_job_application,  
name='employee_process_job_application'),
```

```
    path('employee_single_offer', views.employee_single_offer,  
name='employee_single_offer'),
```

```
    path('employee_offers', views.employee_offers, name='employee_offers'),
```

```
    path('employee_filter_offers', views.employee_filter_offers,  
name='employee_filter_offers'),
```

```

    path('employee_single_notification', views.employee_single_notification,
name='employee_single_notification'),

    path('employee_notifications', views.employee_notifications,
name='employee_notifications'),

    path('employee_view_all_notifications', views.employee_view_all_notifications,
name='employee_view_all_notifications'),

    path('edit_profile', views.edit_profile, name='edit_profile'),

    path('employee_profile', views.employee_profile, name='employee_profile'),

    path('employee_chat', views.employee_chat, name='employee_chat'),

    path('employee_chat_clear', views.employee_chat_clear,
name='employee_chat_clear'),

    path('employee_chat_delete', views.employee_chat_delete,
name='employee_chat_delete'),

    path('employee_chat_warn', views.employee_chat_warn,
name='employee_chat_warn'),

    path('employee_private_chat', views.employee_private_chat,
name='employee_private_chat'),

    path('reloadprivatechat', views.reloadprivatechat, name='reloadprivatechat'),

    path('reloadprivatechat_two', views.reloadprivatechat_two,
name='reloadprivatechat_two'),

    path('send_message', views.send_message, name='send_message'),

```

```

path('employer_chat', views.employer_chat, name='employer_chat'),

path('employer_private_chat', views.employer_private_chat,
name='employer_private_chat'),

path('employer_reloadprivatechat', views.employer_reloadprivatechat,
name='employer_reloadprivatechat'),

path('employer_send_message', views.employer_send_message,
name='employer_send_message'),

path('get_profile', views.get_profile, name='get_profile'),

path('employer_signup', views.employer_signup, name='employer_signup'),

path('employer_dashboard', views.employer_dashboard,
name='employer_dashboard'),

path('employer_jobs', views.employer_jobs, name='employer_jobs'),

path('employer_single_job', views.employer_single_job,
name='employer_single_job'),

path('employer_single_job_applicant', views.employer_single_job_applicant,
name='employer_single_job_applicant'),

path('employer_delete_job/<str:id_value>/', views.employer_delete_job,
name='employer_delete_job'),

path('employer_create_job', views.employer_create_job,
name='employer_create_job'),

path('create_roles', views.create_roles, name='create_roles'),

```

```

path('create_categories', views.create_categories, name='create_categories'),

path('create_requirement', views.create_requirement, name='create_requirement'),

path('employer_profile', views.employer_profile, name='employer_profile'),

path('reject_job_application', views.reject_job_application,
name='reject_job_application'),

path('accept_job_application', views.accept_job_application,
name='accept_job_application'),

path('edit_profile', views.edit_profile, name='edit_profile'),

path('employer_edit_profile', views.employer_edit_profile,
name='employer_edit_profile'),

path('edit_bio', views.editbio, name='edit_bio'),

path('employer_edit_bio', views.employer_editbio, name='employer_edit_bio'),

path('create_skill', views.addskill, name='create_skill'),

path('create_education', views.addeducation, name='create_education'),

path('create_experience', views.addexperience, name='create_experience'),

path('deleteEducation/<str:value_id>/', views.deleteeducation,
name='deleteEducation'),

path('deleteExperience/<str:value_id>/', views.deleteexperience,
name='deleteExperience'),

path('deleteSkill/<str:value_id>/', views.deleteskill, name='deleteSkill'),

```

```
path('upload_resume', views.upload_cv, name='upload_resume'),

path('upload_profile_pic', views.upload_profile_picture,
name='upload_profile_pic'),

path('viewprofile/<str:user_id>/', views.viewprofile, name='viewprofile'),

path('signup', views.signup, name='signup'),

path('login', views.login, name='login'),

path('logout', views.logout, name='logout'),

]
```