



---

# WORK HUNT

---

BCA 5th Semester Project Work

IN PARTIAL FULFILLMENT  
OF THE DEGREE OF  
BACHELOR OF COMPUTER APPLICATIONS  
SESSION 2024-25

**Submitted By**

**Student Name:** Minju Begum

**GU Roll No:** UT-221-341-0023

**Under the guidance of**

**Nisha Pater**

**Department** of Computer Application

**Morigaon College, Affiliated to Gauhati University**

## **CIRTIIFICATE**

This is to certify that the project work entitled “**WorkHunt**: A Web-Based Job Portal” submitted by **Minju Begum** in partial fulfillment of the requirements for the award of the degree of Bachelor of Computer Application (BCA) at Morigaon College, has been examined and accepted by the Department of Computer Applications.

The project is an original work carried out under the supervision and guidance of **Nisha Pater** and adheres to the academic standards set by the institution. To the best of our knowledge, this work has not been submitted elsewhere for any other degree or qualification.

We congratulate **Miss Minju Begum** on the successful completion of her project and wish her continued success in all future endeavors.

Date of Submission: \_\_\_\_\_

Nisha Pater

Depart of Computer Applications  
Morigaon College

## **DECLARATION**

We here by declare that the project report is authentic record of our project entitled **WorkHunt** under the guidance of **Nisha Pather**, Department of Computer Application, Morigaon College. This project is our original work and no part of it have been submitted for any other degree purpose and published in any other from till date.

Date:

Place:

Name: Minju Begum

BCA 5<sup>th</sup> Semester (Guwahati University)

GU Roll No: UT-221-341-0023

## **ACKNOWLEDGMENT**

I extend my deepest gratitude to all those who contributed to the successful completion of my project, **“Workhunt: A Web-Based Job Portal”**, during my final year of Bachelor of Computer Application (BCA).

First and foremost, I am profoundly thankful to **Nisha Pater**, Assistant teacher of Computer Applications at **Morigaon College**, for her unwavering guidance, insightful feedback, and encouragement throughout the project. I express my sincere appreciation to the **Department of Computer Applications, Morigaon College**, for providing the necessary resources, infrastructure, and academic support that enabled me to undertake this project. The knowledge and skills I gained during my coursework laid a strong foundation for this endeavor.

Date:

Name: Minju Begum

Place:

BCA 5<sup>th</sup> Semester (Guwahati University)

GU Roll No: UT-221-341-0023

## **Table of Content**

<b>SNO</b>	<b>Content</b>	<b>Page No.</b>
1	Abstract	5
2	Introduction <ul style="list-style-type: none"><li>- Background</li><li>- Problem Statement</li><li>- Objective</li><li>- Scope</li></ul>	6 - 7
3	Problem Definition	8
4	Hardware and software requirements	9 - 10
5	System Design <ul style="list-style-type: none"><li>- System Architecture</li><li>- Database Design</li><li>- ER Diagram</li></ul>	11 - 14
6	DFD (Data Flow Diagram)	15 - 17
7	Frontend Details	18 - 19
8	Backend Details	20 - 22
9	Testing and Evaluation	23 - 26
10	Website Screenshots	27 - 33
11	Project Screenshots	34 - 40
12	Future work and limitations	41
13	Conclusion	42
14	Resources	43

## **Abstract**

**Workhunt** is a dynamic web-based job portal designed to streamline the recruitment process by bridging the gap between job seekers and employers. Built using **HTML**, **CSS**, **Tailwind CSS**, **PHP**, and **MySQL**, the platform follows the Model-View-Controller (**MVC**) architecture to ensure scalability, maintainability, and separation of concerns. The project addresses the inefficiencies in traditional job-search methods by providing a centralized platform for employers to post vacancies and job seekers to discover opportunities tailored to their skills and preferences.

Key features include a Job Seeker Dashboard for resume management, job searching with filters, and application tracking, alongside an Employer Dashboard for creating and managing job listings, reviewing applicants, and interacting with potential candidates. The system also incorporates secure user authentication (login/registration) and role-based access control to ensure data privacy.

The frontend leverages Tailwind CSS for responsive and modern UI design, while the backend uses PHP for server-side logic and MySQL for efficient database management. By adopting MVC architecture, the project achieves modular code organization, simplifying future enhancements.

**Workhunt** successfully demonstrates how structured web development practices can create a user-friendly, scalable solution for modern recruitment challenges. Testing confirms seamless integration of features like resume uploads, real-time job searches, and employer-candidate interactions, highlighting the platform's potential to reduce hiring friction and improve user engagement.

# Introduction

## 1.1 Background

In today's competitive job market, the demand for efficient and user-friendly platforms to connect employers with qualified candidates has surged. Traditional recruitment methods, such as newspaper ads or manual job fairs, are increasingly being replaced by digital solutions due to their limited reach, time-consuming processes, and lack of real-time interaction. Online job portals have emerged as a critical tool for bridging this gap, offering features like instant job postings, resume management, and advanced search filters. However, many existing platforms lack intuitive interfaces, role-specific customization, or robust security measures.

Workhunt, a web-based job portal, addresses these challenges by leveraging modern web technologies to create a seamless, secure, and scalable platform for both job seekers and employers.

## 1.2 Problem Statement

Despite the proliferation of job portals, several pain points persist:

- Job seekers struggle to **find relevant roles** due to inadequate search filters.
- Employers face difficulties in **managing applications** and identifying candidates with the right skills.
- Many platforms lack **real-time updates** or **secure authentication**, leading to inefficiencies and data vulnerabilities.
- Resume management systems often fail to provide **user-friendly interfaces** for quick updates and parsing.

**Workhunt** aims to resolve these issues by offering a unified platform with advanced features tailored to streamline recruitment workflows.

## 1.3 Objectives

The primary objectives of this project are:

- To develop a **secure and responsive job portal** using HTML, CSS, Tailwind CSS, PHP, and MySQL.
- To implement **role-based dashboards** for job **seekers** and **employers** with distinct functionalities.
- To enable job seekers to:
  - Search and apply for jobs using filters (e.g., location, keyword).
  - Upload, update, and manage resumes efficiently.
- To empower employers to:
  - Create, edit, and delete job listings.
  - Review and shortlist applicants.
- To ensure data security through **MVC architecture** and robust authentication (login/registration).
- To design an intuitive UI/UX using Tailwind CSS for cross-device compatibility.

## 1.4 Scope

### Scope

- The project focuses on developing a web-based job portal that is accessible across different devices.
- It includes functionalities for user registration and login, job posting and management, resume updating, and job search features.
- The system supports distinct user roles such as job seekers, employers, and administrators, each with dedicated dashboards and functionalities.
- The portal is designed using modern front-end technologies (HTML, CSS, TailwindCSS) and back-end technologies (PHP, MySQL) under the MVC framework, ensuring a structured and maintainable codebase.



## Problem Definition

The modern job market faces significant inefficiencies due to fragmented platforms and outdated recruitment practices. Despite advancements in technology, job seekers and employers continue to encounter persistent challenges that hinder effective hiring processes. Below are the key problems addressed by the **Workhunt** job portal:

### 1.1 Current Challenges

1. For Job Seekers:
  - Difficulty in **discovering relevant job opportunities** due to limited search filters and poor categorization.
  - Manual resume updates and lack of centralized platforms to track application statuses.
  - Insecure or cumbersome authentication systems on existing portals.
2. For Employers:
  - Time-consuming processes to **post jobs** and **manage applications** across multiple platforms.
  - Inability to efficiently filter candidates based on skills, experience, or location.
  - Lack of intuitive dashboards to monitor hiring workflows and applicant interactions.
3. Systemic Issues:
  - Fragmented platforms requiring users to juggle multiple tools for job searches, resume uploads, and communication.
  - Outdated interfaces with poor mobile responsiveness, limiting accessibility.
  - Security vulnerabilities in user data handling (e.g., weak authentication).

### 1.2 Workhunt's Approach

The Workhunt job portal directly addresses these gaps by offering:

- **Role-Specific Dashboards:** Tailored interfaces for job seekers (resume management, application tracking) and employers (job posting, applicant reviews).
- **Advanced Search Filters:** Location, salary range, experience level, and job type.
- **Secure Authentication:** PHP session management and password hashing.
- **Scalable Architecture:** MVC design pattern for modular, maintainable code.
- **Responsive UI:** Built with Tailwind CSS for seamless cross-device access.

## **Hardware and software requirements**

The successful development and deployment of the **workhunt** project rely heavily on the utilization of appropriate hardware and software resources. This section provides a detailed overview of the hardware and software requirements necessary for the project implementation.

### **Hardware and Software used:**

<b>Hardware Used</b>	<b>Software Used</b>
Intel i5 core	XAMPP Server
256 GB SSD	HTML, CSS, JS, BOOTSRAP, PHP
8GB RAM	Visual Studio Code
OS – windows v-11	MySQL

### **Hardware Requirements:**

#### **1. Server:**

- ◆ Processor: Intel Core i3 or equivalent
- ◆ RAM: 4GB or higher
- ◆ Storage: 256GB SSD or higher
- ◆ Network Interface: Ethernet or Wi-Fi for internet connectivity

#### **2. Client Devices (Desktop/Laptop, Tablet, Smartphone):**

- ◆ Processor: Intel or equivalent processor
- ◆ RAM: 2GB or higher
- ◆ Storage: 128GB SSD or higher
- ◆ Display: Minimum resolution of 1366x768 pixels
- ◆ Network Interface: Ethernet or Wi-Fi for internet connectivity

### **Software Requirements:**

#### **1. Operating System:**

- ◆ Server: Windows, Linux, or macOS
- ◆ Client Devices: Windows, macOS, Linux, iOS, Android

## **2. Development Environment:**

- ◆ Integrated Development Environment (IDE): Visual Studio Code
- ◆ Web Browser: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge

## **3. Server-Side Technologies:**

- ◆ PHP (OOP)
- ◆ MVC Architecture
- ◆ Laragon: Version 7.4.26 or higher (includes Apache, MySQL, PHP, and Perl)

## **4. Client-Side Technologies:**

- ◆ HTML: Version 5
- ◆ CSS: Version 3
- ◆ JavaScript: ECMAScript 6 (ES6)
- ◆ Tailwind CSS

# System Design

The system design of **Workhunt** focuses on building a scalable, maintainable, and efficient job portal by employing modern web technologies and a robust architectural pattern. The project is structured using the Model-View-Controller (**MVC**) architecture, which clearly separates the presentation layer from the business logic and data management layers. This approach not only improves code reusability and maintainability but also simplifies debugging and future enhancements.

## 1.1 System Architecture

The **Workhunt** portal follows the **Model-View-Controller (MVC)** architecture to ensure modularity and scalability. The system is divided into three interconnected layers:

### 1. Model (Data Layer):

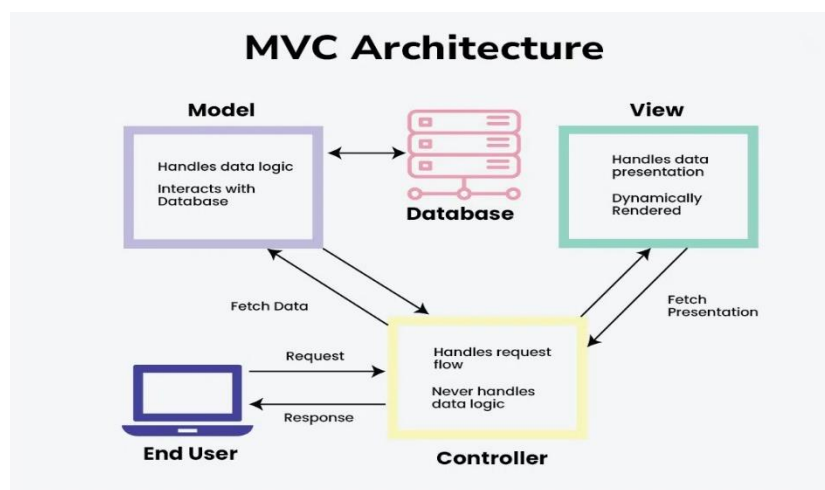
- **Role:** Manages database interactions and business logic.
- **Tech:** PHP and MySQL.
- **Functionality:** Handles **CRUD (Create, Read, Update, Delete)** operations for user data, job listings, and applications.

### 2. View (Presentation Layer):

- **Role:** Renders the user interface.
- **Tech:** HTML, CSS, Tailwind CSS, Javascript.
- **Functionality:** Displays Website pages, dashboards, forms, and job search results with responsive design.

### 3. Controller (Logic Layer):

- **Role:** Processes user requests and mediates between Model and View.
- **Tech:** PHP.
- **Functionality:** Manages authentication, job search queries, and role-based routing (e.g., redirecting employers to their dashboard).



## 1.2 Database Design

The database for **Workhunt** has been designed to manage and store all essential data required for the efficient functioning of the job portal. The schema employs a relational database model, using MySQL as the database management system. The design ensures normalized relationships between tables to reduce redundancy and maintain data integrity. Below is a detailed description of the database structure.

### Entity-Relationship(ER) Diagram

An Entity-Relationship (ER) diagram is a visual representation of the entities (things of significance) and the relationships (connections or associations) between them within a database. It is a popular tool used in database design to model the structure of a database and illustrate how data is organized and related.

In an ER diagram:

- **Entity:** An entity is a real-world object or concept that has properties or attributes. In the context of a database, an entity is typically represented as a table, with each row in the table representing a specific instance of that entity.
- **Attribute:** An attribute is a characteristic or property of an entity. For example, in a "Student" entity, attributes could include student ID, name, and date of birth.
- **Relationship:** A relationship describes how two or more entities are related to each other. Relationships can be one-to-one, one-to-many, or many-to-many, and they are represented graphically in the ER diagram with lines connecting the related entities.

ER diagrams are essential in database design as they provide a clear and visual representation of the database schema, which helps database designers and stakeholders understand the structure of the database and ensure that it meets the requirements of the system being developed. They also serve as a communication tool between developers, designers, and stakeholders involved in the database design process, facilitating discussions and decisions regarding the organization and structure of the database.

Entity-Relationship (ER) diagrams are used to represent the structure of a database, including its entities, attributes, and relationships. There are several types of ER diagrams, each serving a specific purpose in database design and modeling. Here are the main types of ER diagrams:

#### 1. Conceptual ER Diagrams:

- Also known as high-level or conceptual models.

- Focus on the overall structure and organization of the database at a conceptual level.
- Abstract representation of entities, attributes, and relationships without specifying details of implementation.
- Used during the initial stages of database design to capture requirements and concepts.

## 2. Logical ER Diagrams:

- Intermediate level of abstraction between conceptual and physical models.
- Represent entities, attributes, and relationships in a more detailed manner compared to conceptual diagrams.
- Include primary and foreign keys to represent entity relationships.
- Provide a logical view of the database schema that can be implemented in a database management system (DBMS).

## 3. Physical ER Diagrams:

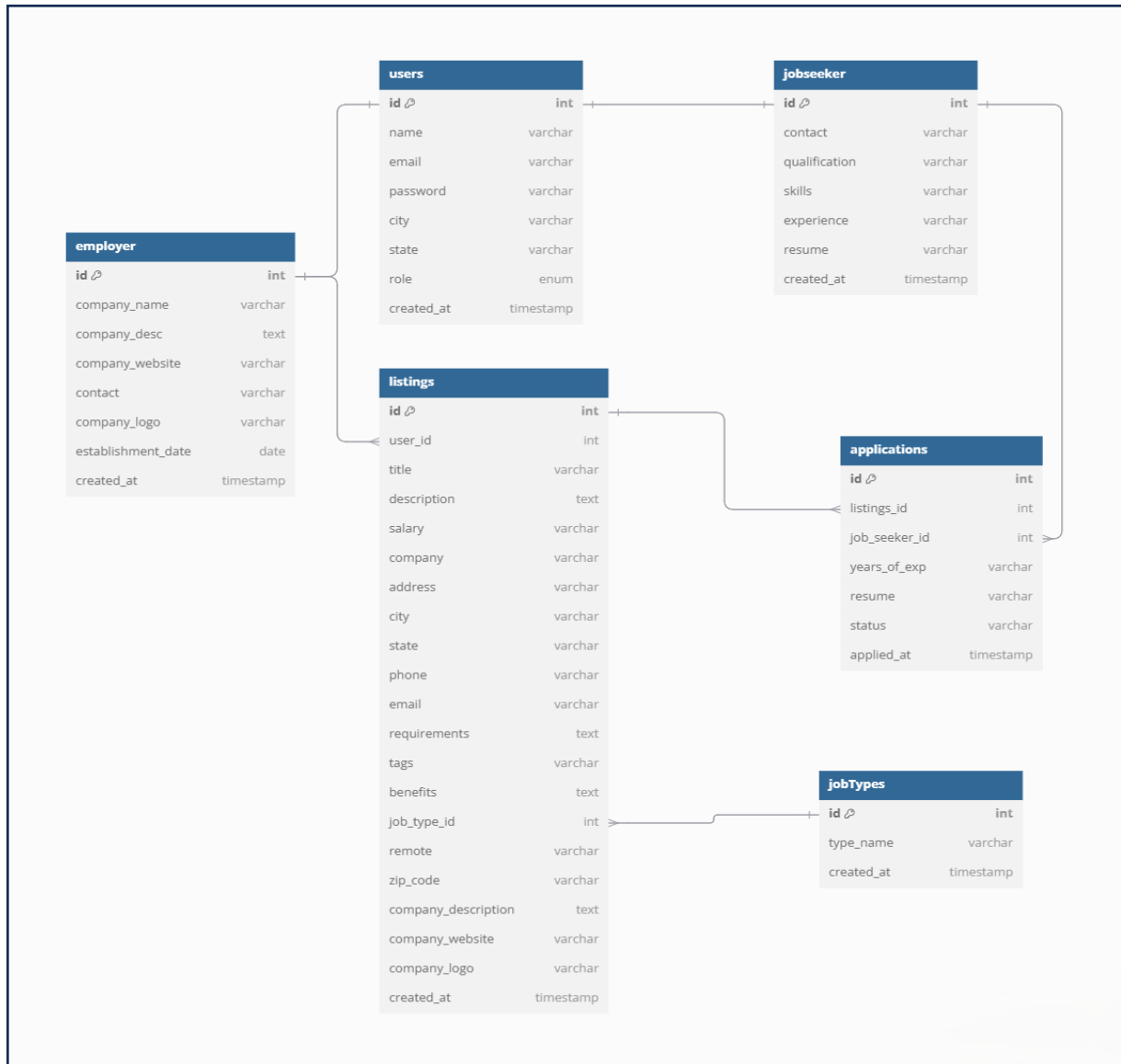
- Also known as implementation or detailed models.
- Represent the physical implementation of the database schema.
- Include details such as data types, indexing, constraints, and storage specifications.
- Designed to be directly implementable in a specific DBMS.

## 4. Relational ER Diagrams:

- Specifically used in relational database design.
- Represent entities, attributes, and relationships in the form of tables and their relationships.
- Each entity becomes a table, and each relationship becomes a foreign key constraint between tables.
- Reflect the relational model principles, including normalization and referential integrity.

Each type of ER diagram serves a specific purpose in database design, and the choice of diagram type depends on the stage of design, the level of detail required, and the specific requirements of the database system being developed.

## ER DIAGRAM (WORKHUNT)



### Relationship Between Table:

- **One to one** – **users** with **jobseekers** (via `jobseeker.id = user.id`)
- **One to one** – **users** with **employers** (via `employer.id = user.id`)
- **One to Many** – **employers** with **Listings** (via `employer.id = listings.user_id`)
- **One to Many** – **jobTypes** with **listings** (via `jobTypes.id = listings.job_type_id`)
- **One to Many** – **listings** with **applications** (via `listings.id = applications.listings_id`)
- **One to Many** – **jobseeker** with **applications** (via `jobseeker.id = applications.job_seeker_id`)

## DFD

The Data Flow Diagram (**DFD**) for **Workhunt** provides a visual representation of how data is processed within the system. It shows the movement of data between external entities, processes, and data stores. By mapping out these flows, the DFD helps in understanding how different components of the job portal interact, ensuring data is handled efficiently and securely.

In the **Workhunt** system, key processes include user management (registration and login), job management (posting and updating job listings), application processing (submitting and tracking applications), and job search functionality. External entities (such as job seekers, employers, and administrators) interact with these processes, while the system stores data in various data stores (such as user details, job listings, and applications).

### 1. Context Diagram (Level 0 DFD)

- **External Entities:**

- **Job Seeker:** Interacts with the system for registration, job search, resume updates, and application submissions.
- **Employer:** Uses the system to post job listings, manage applications, and update company profiles.
- **Administrator:** Oversees system operations, user management, and job listing approvals.

- **Main Process:**

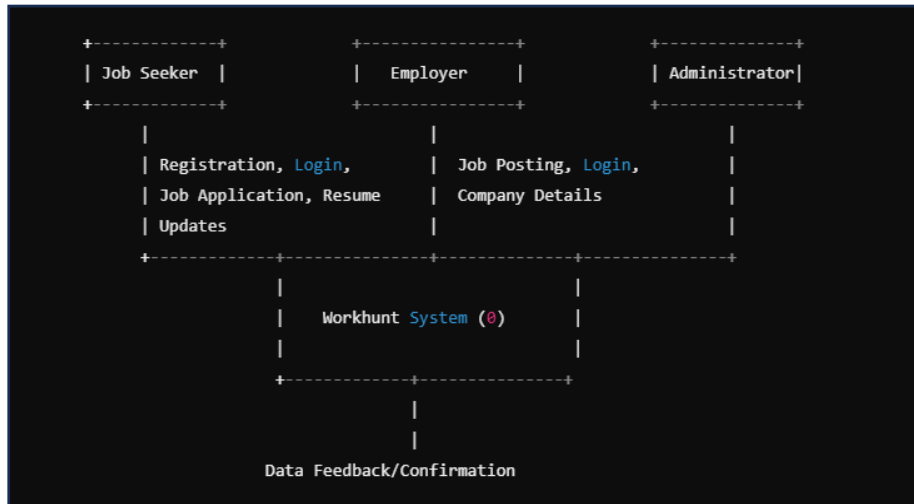
- **Workhunt System:** The central system that processes all incoming data from external entities and manages data storage.

- **Data Flows:**

- Job Seeker sends registration details, login credentials, resume updates, and job applications to the **Workhunt** System.
- Employer sends job posting information, login credentials, and company details.
- Administrator interacts with the system to manage users and job listings.



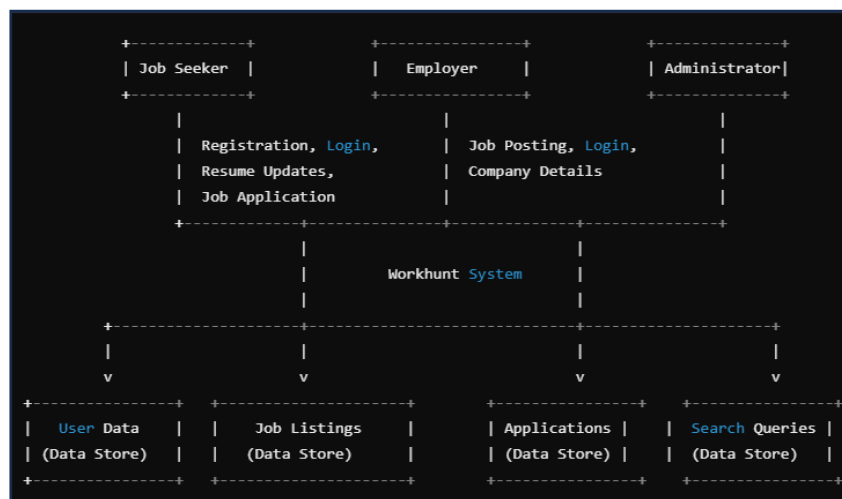
## Context Diagram (Representation):



## 2. Level-0 DFD

- **Key Processes:**
  - **User Management:** Handles registration, login, and profile updates.
  - **Job Management:** Manages creation, updating, and deletion of job listings.
  - **Application Processing:** Manages the submission and tracking of job applications.
  - **Search & Filtering:** Processes job search requests and applies search filters.
- **Data Stores:**
  - **User Data:** Stores details of job seekers, employers, and administrators.
  - **Job Listings:** Stores all job postings created by employers.
  - **Applications:** Stores job applications submitted by job seekers.

## Level-0 DFD Representation



## Data Flow Description:

- **User Management Process:**
  - **Input:** Registration forms, login credentials, profile updates.
  - **Output:** User data stored/updated in the User Data data store; feedback (e.g., login confirmation) sent back to the job seeker or employer.
- **Job Management Process:**
  - **Input:** Job posting details from employers.
  - **Output:** New or updated job listings stored in the Job Listings data store.
- **Application Processing Process:**
  - **Input:** Job applications and resumes from job seekers.
  - **Output:** Application data stored in the Applications data store; confirmation of application submission sent to job seekers.
- **Search & Filtering Process:**
  - **Input:** Search queries and filter criteria from job seekers.
  - **Output:** Matching job listings retrieved from the Job Listings data store and displayed to the user.

### 3. Level-1 DFD (User Management Process)

#### Process: User Management

- Sub-Processes:
  - **User Registration:** Accepts new user details and stores them in the User Data store.
  - **User Login:** Authenticates user credentials against the User Data store.
  - **Profile Update:** Allows users to update their personal information and resume.



## Front-End Details

The front end of **Workhunt** is a crucial component of the overall system, designed to offer an intuitive and engaging experience for both job seekers and employers. This section details the technologies used, design principles adhered to, implementation strategies, and the measures taken to ensure responsiveness and optimal performance across a variety of devices and browsers.

### 1. Overview

The primary objective of the front end is to bridge the gap between users and the backend functionalities, making it simple for job seekers to search and apply for jobs and for employers to post and manage job listings. The interface is structured to be clear, concise, and visually appealing, allowing users to navigate effortlessly through various features such as dashboards, profile management, job search functionalities, and application tracking.

### 2. Technologies Employed

- **HTML:** HTML is the foundational markup language that structures the content of each webpage. It organizes information semantically, ensuring that each section of the website—from navigation menus to content areas—is clearly defined.
- **CSS:** CSS is used to style the website, providing layout control, color schemes, typography, and overall aesthetic consistency. It ensures that the visual presentation aligns with the intended design language and enhances user readability.
- **JavaScript:** JavaScript is integrated to add interactivity and dynamic content. It plays a vital role in enhancing user engagement through real-time interactions, such as form validations, dynamic content updates, and interactive elements like modals and dropdowns. The use of JavaScript ensures that the user experience is smooth and that the interface responds quickly to user inputs.
- **Tailwind CSS:** Tailwind CSS is a utility-first CSS framework that has been adopted to accelerate the styling process. It provides a comprehensive set of pre-built classes that allow for rapid design and prototyping. Tailwind CSS aids in maintaining a consistent design language across the application while also ensuring that the website is responsive and easily adaptable to various screen sizes.

### 3. Design Principles and User Experience

- **User-Centric Approach:** The design of Workhunt is rooted in user-centric principles. The interface has been developed with a clear focus on the needs and behaviors of its target audiences—job seekers and employers. This approach ensures that each interaction is intuitive, reducing the learning curve and enabling users to achieve their objectives with minimal friction.

- **Clean and Minimalist Aesthetics:** A clean, minimalist design has been adopted to prioritize content and functionality. The layout avoids clutter, ensuring that users can focus on key tasks such as searching for jobs, updating resumes, and managing listings without unnecessary distractions. The visual hierarchy is carefully designed to guide users through the site, highlighting important information and actions.
- **Consistency in Visual Elements:** Consistency is maintained across all pages through a uniform color palette, typography, and layout structures. By leveraging Tailwind CSS, the design achieves a cohesive look and feel that reinforces brand identity and provides a seamless user experience. This consistency is vital in building user trust and ensuring that interactions remain predictable throughout the website.
- **Responsive and Adaptive Design:** A major focus has been placed on ensuring that the website performs optimally across all devices. Responsive design techniques have been used to guarantee that layouts adjust gracefully to different screen sizes, from large desktop monitors to small mobile devices. The use of Tailwind CSS responsive utilities has been central to achieving this adaptability, ensuring that the site maintains usability and aesthetic integrity regardless of the device being used.

## **Back-End Details**

The back-end of **Workhunt** serves as the backbone of the application, managing data operations, business logic, and communication with the database. It is built using PHP with an Object-Oriented Programming (**OOP**) approach, integrated with **MySQL** for data storage, and structured using the Model-View-Controller (**MVC**) architecture. This combination of technologies ensures that the system is not only modular and maintainable but also capable of handling complex operations with efficiency.

### **1. Overview**

The back-end system is responsible for processing user requests, managing sessions, and performing critical operations such as user authentication, job postings, application processing, and profile management. By adopting the **MVC** architecture, the system separates concerns into distinct components—models for data handling, views for presenting data, and controllers for business logic. This separation promotes cleaner code, facilitates debugging, and simplifies future enhancements.

### **2. Technologies Employed**

- **PHP with OOP:** The use of OOP in PHP allows for the creation of reusable, modular, and organized code. Classes and objects are utilized to encapsulate functionalities, making it easier to manage complex business logic and maintain code consistency across the application. OOP principles such as inheritance, encapsulation, and polymorphism are applied to build scalable modules that handle tasks like user management, job posting, and application processing.
- **MySQL Database Management System:** MySQL is used as the primary data store for the application. It provides a robust relational database environment where data integrity is maintained through well-defined relationships and constraints. The database is structured to support various functionalities, including storing user details, job listings, application records, and other related data.
- **Structured Query Language (SQL):** SQL is used for creating, reading, updating, and deleting (**CRUD**) operations on the database. The back-end leverages SQL queries to efficiently manage data transactions, ensuring that information is stored and retrieved in a reliable manner. Optimized SQL queries and indexes are used to enhance performance, particularly when handling large volumes of data.

## MVC Architecture

The MVC architectural pattern is central to the back-end design of Workhunt. It divides the application into three interconnected components:

- **Model:** The model handles data logic and database interactions. It encapsulates the structure of the application's data, defining how data is stored, retrieved, and manipulated. This component interacts directly with MySQL using SQL queries to perform operations on the database.
- **View:** While the front-end handles the user interface, the view in the back-end context is responsible for formatting the data before it is sent to the client. It receives data from the controller and presents it in a user-friendly format.
- **Controller:** The controller acts as an intermediary between the model and the view. It processes incoming HTTP requests, applies business logic, and determines which model and view to invoke. This separation ensures that each part of the application is only responsible for its specific functionality, which improves maintainability and scalability.

### 3. Implementation Strategies

1. **Modular Code Structure:** The back-end is designed with modularity in mind. Each module is responsible for a specific set of functionalities, such as user authentication, job listing management, or application processing. This modular approach allows developers to work on different aspects of the system independently and makes it easier to update or extend features without affecting the entire codebase.
2. **Robust Data Management:** Data integrity and consistency are critical in a job portal application. The database schema is designed to capture all necessary information about users, job listings, and applications. The use of MySQL ensures that data relationships are well-defined, and constraints such as primary keys, foreign keys, and unique indexes are used to enforce data accuracy. Regular backups and transaction management further contribute to the system's reliability.
3. **Security Measures:** Security is a paramount concern in the back-end of **Workhunt**. Various measures have been implemented to protect user data and prevent unauthorized access:
  - **Input Validation:** All inputs are thoroughly validated on the server side to prevent SQL injection and other forms of malicious data entry.
  - **Session Management:** Secure session handling mechanisms are in place to manage user logins and authentication processes, ensuring that session data is protected.
  - **Data Encryption:** Sensitive data, such as passwords, is encrypted using robust hashing algorithms. This ensures that even if data breaches occur, the sensitive information remains protected.

4.

## Why Choose PHP?

There are several reasons why PHP was chosen as the back-end language for the Book Bin platform:

- **Wide Adoption and Community Support:** PHP is one of the most widely used server-side scripting languages for web development, with a large and active community of developers. This extensive community support ensures a wealth of resources, documentation, and libraries available for PHP development, making it easier to find solutions to common challenges and leverage third-party tools and frameworks.
- **Ease of Learning and Use:** PHP is known for its simplicity and ease of learning, making it accessible to developers of varying skill levels. Its syntax is similar to C, Java, and other popular programming languages, which makes it easier for developers to transition to PHP and start building web applications quickly.
- **Flexibility and Versatility:** PHP is a highly versatile language that can be used for a wide range of web development tasks, from simple scripts to complex web applications. It offers a wide range of built-in functions and libraries for various tasks such as file handling, database interaction, and form processing, reducing the need for external dependencies.
- **Integration with MySQL and Other Databases:** PHP has excellent support for interacting with databases, particularly MySQL, which is a popular choice for web applications. PHP's native MySQL functions and extensions make it straightforward to connect to MySQL databases, execute SQL queries, and handle database transactions, facilitating efficient data management in web applications like Book Bin.
- **Scalability and Performance:** PHP is known for its scalability and performance, particularly when used in conjunction with caching mechanisms, opcode caching, and other optimization techniques. With proper configuration and optimization, PHP-based web applications like Book Bin can handle large volumes of traffic and scale to meet growing user demands.

## Testing and Evaluation

Testing is a critical phase in the development of **Workhunt**, ensuring that the system meets its functional, performance, and security objectives. This section outlines the testing methodologies used, summarizes the results of various tests, and discusses the feedback obtained to confirm that the system delivers a reliable, user-friendly experience.

### 1. Testing Methodologies

A combination of testing methodologies was adopted to validate the functionality, performance, and security of **Workhunt**. These methodologies include:

- **Integration Testing:** It was conducted to ensure that the different modules of the system (such as user authentication, job posting, job search, and application processing) interact seamlessly. This testing verified that data flows correctly between the front end and back end, and that APIs and database queries produce the expected outcomes when combined.
- **System Testing:** It was performed on the complete, integrated system to validate that all components work together as intended. This phase focused on verifying that the entire job portal operates correctly under normal operating conditions, covering key workflows from user registration and login to job searching and application submission.
- **User Acceptance Testing (UAT):** A select group of target users participated in UAT to evaluate the usability and overall experience of the portal. Their feedback was used to identify any usability issues, confirm that the interface is intuitive, and ensure that the system meets the practical needs of both job seekers and employers.
- **Performance Testing:** Performance tests were conducted to measure the system's response times, load capacity, and resource utilization. These tests simulated multiple simultaneous user interactions to ensure that the website remains responsive and stable under moderate to high loads.
- **Security Testing:** Security testing was carried out to validate that the system is protected against common vulnerabilities. This included tests for proper input validation, session management, password encryption, and role-based access control to ensure that user data remains secure throughout the application.

### 2. Functional Testing

A series of tests were conducted to verify that the core functionalities of **Workhunt** operate as expected. The focus areas for functional testing included:

- **User Authentication:** The **registration** and **login** processes were rigorously tested. Inputs were validated to ensure that incorrect or malicious data would not be accepted, and appropriate error messages were displayed. Successful authentication and session handling were verified, ensuring that users can reliably access their profiles.



- **Job Seeker Dashboard and Resume Updates:** The job seeker dashboard was tested to confirm that users can update their profiles and resumes without issues. Data consistency across sessions was confirmed, and the interface was reviewed for responsiveness and clarity.
- **Job Search Functionality:** The search features were thoroughly evaluated to ensure that job listings are accurately retrieved based on various criteria such as job type, location, and keywords. Filtering mechanisms were also checked to verify that users can efficiently narrow down their search results.
- **Employer Dashboard and Job Listing Management:** Employers were able to post new job listings, update existing ones, and delete outdated posts. The functionality was tested for both client-side and server-side validation, ensuring that only valid data is stored in the database. The interaction between the employer dashboard and back-end services was verified for both correctness and security.
- **Application Processing:** The end-to-end process of job application submission was tested. This included capturing application data, storing it in the database, and providing appropriate confirmation messages to users. The test cases confirmed that the application processing flow is reliable and user-friendly.

### 3. Performance Evaluation

To ensure that **Workhunt** provides a smooth and efficient user experience, several performance evaluations were performed:

- **Response Times:** The system's response times were measured during common operations such as logging in, navigating dashboards, and executing search queries. The tests confirmed that pages load quickly and interactive elements respond promptly, contributing to a positive user experience.
- **Load Testing:** Simulated multiple user interactions were used to evaluate how the system performs under varying loads. The tests demonstrated that the system remains stable and responsive even with a moderate increase in user activity. Recommendations for further optimization, such as implementing caching mechanisms and query optimizations, were identified for future iterations.
- **Resource Utilization:** During performance testing, server resources (CPU, memory, and disk I/O) were monitored to ensure efficient management. The modular MVC design was found to support efficient resource allocation, with the system exhibiting minimal overhead during data processing.

#### 4. Security Evaluation

Security is paramount for a job portal handling sensitive user information. The following security evaluations were conducted:

- **Input Validation:** All user inputs were validated on the server side to prevent common vulnerabilities such as SQL injection and cross-site scripting (XSS). This ensured that the system only processes well-formed, safe data.
- **Session Management:** The security of user sessions was thoroughly tested to ensure proper creation, maintenance, and termination. Measures to prevent session hijacking were verified, and secure session management practices were confirmed.
- **Password Encryption:** Passwords are stored using robust hashing algorithms. The encryption process was evaluated to ensure that it meets industry standards, providing strong protection for user credentials.
- **Access Control:** Role-based access control was tested to ensure that different user types (job seekers, employers, and administrators) have access only to the functionalities appropriate to their roles. This prevents unauthorized access and preserves data integrity across the system.

## Test Cases

Test Case	Description	Expected Outcome	Actual Result
<b>User Registration</b>	<ol style="list-style-type: none"> <li>1. Navigate to the registration page.</li> <li>2. Fill in all required fields with valid information.</li> <li>3. Submit the registration form</li> </ol>	A confirmation message is displayed and the new user's data is stored in the database.	Pass
<b>User Login</b>	<ol style="list-style-type: none"> <li>1. Navigate to the login page.</li> <li>2. Enter a valid email and password.</li> <li>3. Click the login button.</li> </ol>	The user is redirected to their dashboard, and a secure session is created.	Pass
<b>Job Posting by Employer</b>	<ol style="list-style-type: none"> <li>1. Log in as an employer.</li> <li>2. Navigate to the employer dashboard and select "Create Job Listing".</li> <li>3. Enter all required job details.</li> <li>4. Submit the form.</li> </ol>	A new job listing is created and immediately visible on the job listings page, with data stored correctly in the database.	Pass
<b>Job Application Submission</b>	<ol style="list-style-type: none"> <li>1. Log in as a job seeker.</li> <li>2. Navigate to a specific job listing.</li> <li>3. Click the "Apply" button.</li> <li>4. Fill in the required application details.</li> <li>5. Submit the application.</li> </ol>	The job application is successfully submitted, recorded in the database, and a confirmation message is displayed.	Pass
	Employer Job Application Attempt	The system should block the employer from applying by disabling the "Apply" functionality or displaying an error message, thereby preventing the action.	Fail

# Website Screenshots

To provide a clear visual representation of the **Workhunt** job portal, the following screenshots illustrate key pages and functionalities. These images highlight the design elements, layout, and overall user experience implemented in the project.

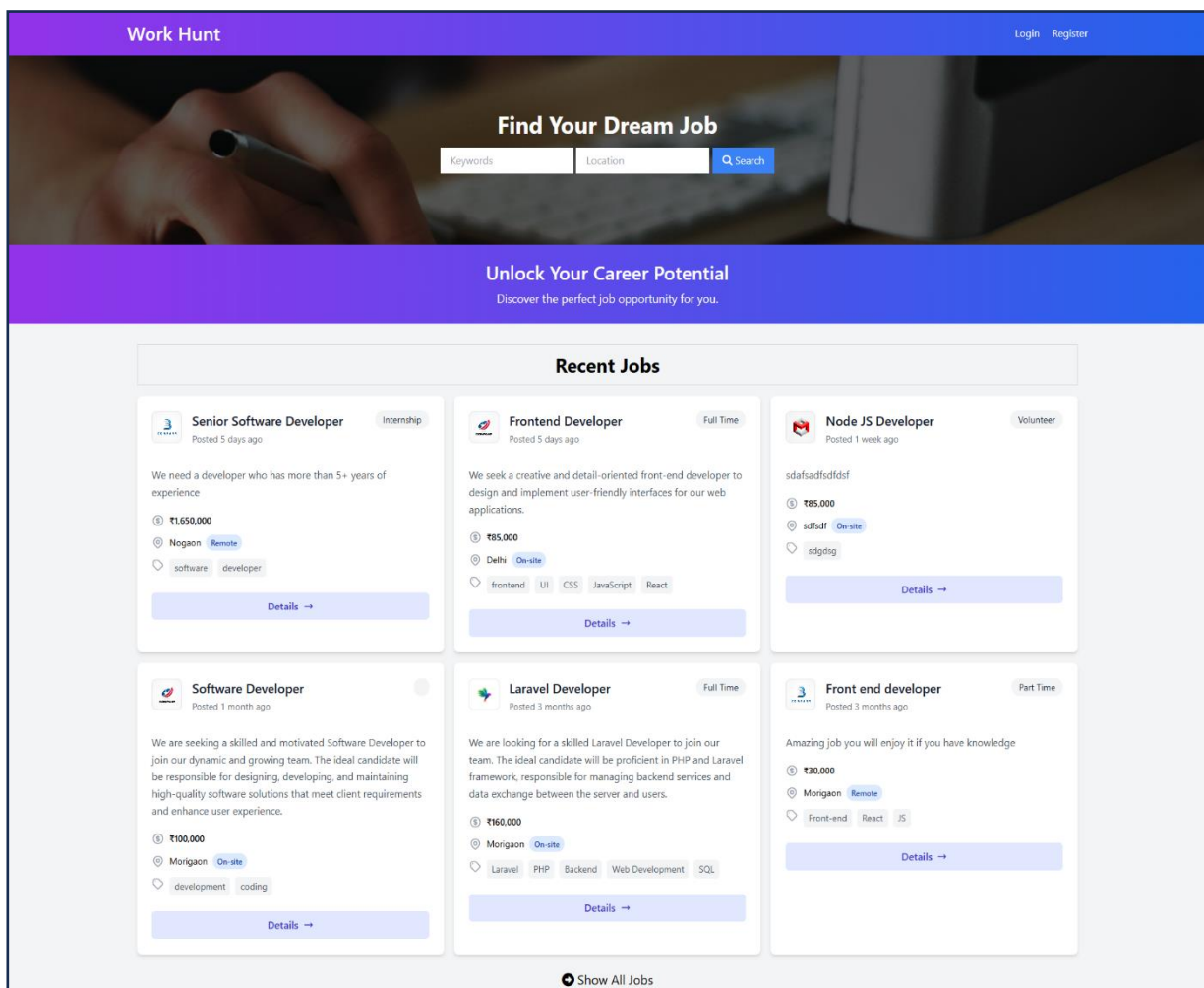
## 1. Home page

The homepage of the **Workhunt** job portal provides a welcoming and user-friendly interface for job seekers and employers. It features a prominent search bar allowing users to search for jobs by keywords and location. Below the header, users can explore recent job listings displayed in a clean, card-based layout. Each card includes key job details such as title, salary, location, job type, and tags, offering quick insights into available positions.

### Key Features:

- **Search Bar:** Allows users to perform job searches by entering keywords and location.
- **Job Listings:** Displays recently posted jobs with essential details in an organized grid format.
- **Navigation Options:** Provides easy access to the login and registration functionalities.

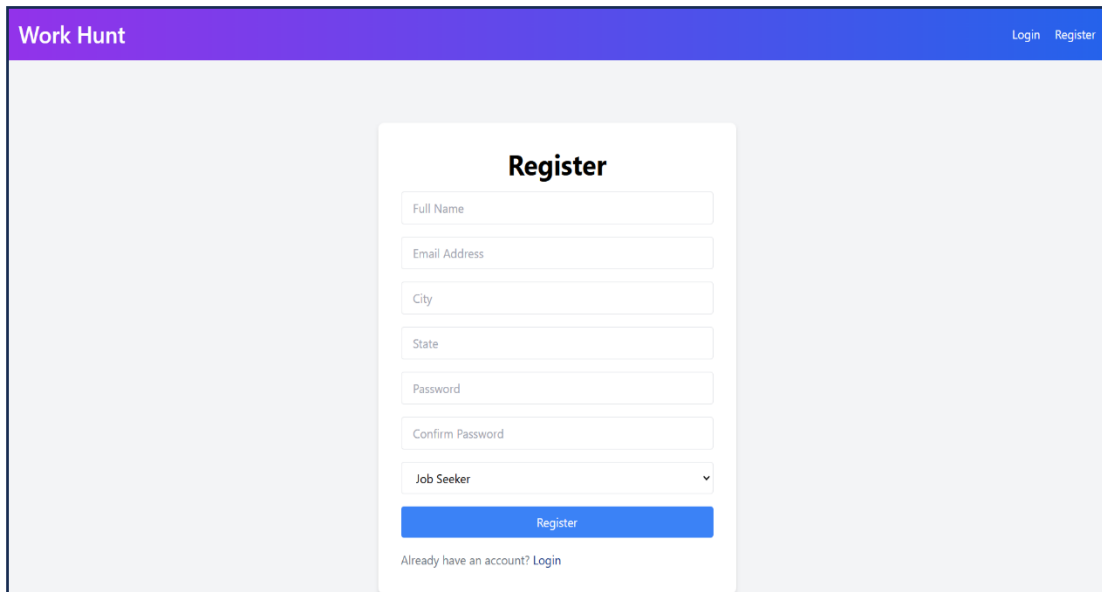
### Screenshot



## 2. User Registration and Login Page

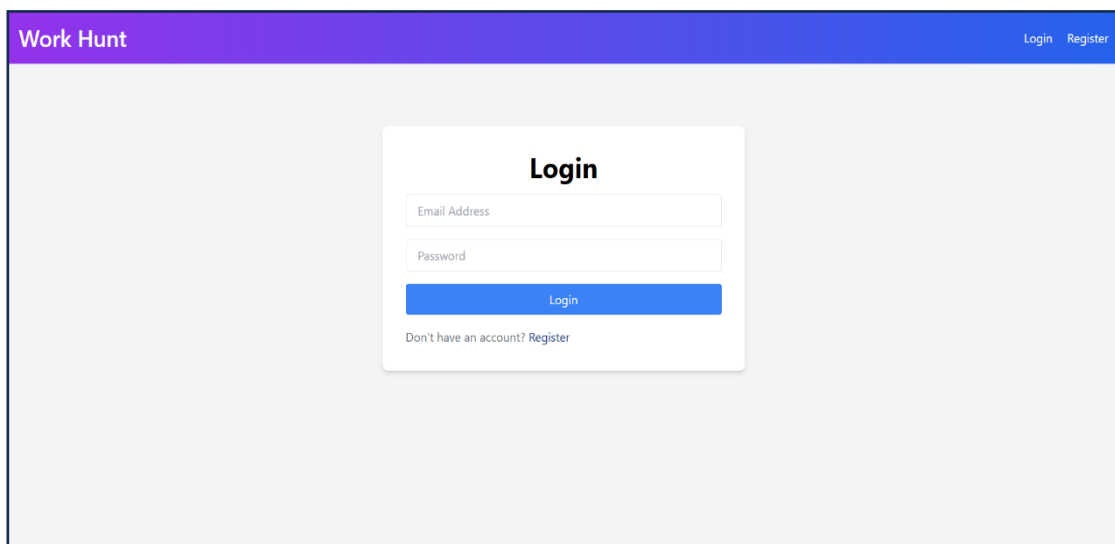
This page allows new users to register and existing users to log in. It is designed with simplicity and clarity in mind, ensuring that users can quickly complete the process. Input fields are clearly labeled and styled consistently with the overall design. Also user can select their role at the time of registration (job seeker, employer).

### Registration Screenshot



The screenshot shows the 'Register' page of the 'Work Hunt' application. The page has a purple header with 'Work Hunt' on the left and 'Login Register' on the right. The main content area is light gray. In the center, there is a white card titled 'Register'. The card contains the following fields: 'Full Name', 'Email Address', 'City', 'State', 'Password', 'Confirm Password', and a dropdown menu for 'Job Seeker'. Below these fields is a blue 'Register' button. At the bottom of the card, there is a link: 'Already have an account? Login'.

### Login page Screenshot



The screenshot shows the 'Login' page of the 'Work Hunt' application. The page has a purple header with 'Work Hunt' on the left and 'Login Register' on the right. The main content area is light gray. In the center, there is a white card titled 'Login'. The card contains the following fields: 'Email Address' and 'Password'. Below these fields is a blue 'Login' button. At the bottom of the card, there is a link: 'Don't have an account? Register'.

### 3. Job Seeker Dashboard

The Job Seeker Dashboard provides users with a personalized and interactive space to manage their profiles. This page displays key information about the job seeker, such as their name, contact details, qualifications, skills, and uploaded resume. It also includes an editable form to update their details and upload a new resume.

#### Screenshot

The screenshot shows the 'Work Hunt' Job Seeker Dashboard. The header is purple with 'Work Hunt' on the left and 'Welcome John Smith Logout Profile' on the right. The main content is divided into two columns. The left column, titled 'Profile', shows a summary for John Smith, including location (Morigaon, Assam), contact (9365910718, john@gmail.com), qualification (MCA), and skills (Web Development, PHP OOP, MVC Architecture, Laravel, Git & Github). It also has a 'Resume' section with a 'View Resume' button. The right column is an editable form with fields for Name, Email, Contact, Qualification, Skills, City, State, and an 'Upload Resume' section with a cloud icon and 'Upload a file' text. A green 'Save & Update' button is at the bottom.

### 4. Single Job Listing Page

The Single Job Listing Page is designed to provide a detailed view of a specific job opportunity. It highlights the job title, description, salary, location, tags, and other essential details. This page helps job seekers understand the job requirements and benefits before applying.

The screenshot shows the 'Work Hunt' Single Job Listing Page. The header is purple with 'Work Hunt' on the left and 'Login Register' on the right. The main content area has a blue header with 'Unlock Your Career Potential' and 'Discover the perfect job opportunity for you.' Below this is a job listing for 'Frontend Developer' with a description, salary (₹85,000), location (Delhi, Delhi), and tags (frontend, UI, CSS, JavaScript, React). The 'Job Details' section includes 'Job Requirements' and 'Benefits'. A blue 'Apply Now' button is at the bottom.

## 5. Job Application Form Page

This page is where job seekers can complete and submit an application for a specific job posting. The form is designed to collect essential information such as personal details, qualifications, experience, and additional files like resumes. Displays job title, company name, benefits, salary, and key requirements on the left side for easy reference while filling out the form.

### Screenshot

The screenshot shows the 'Work Hunt' job application interface. On the left, a job listing for 'Frontend Developer' at 'TechNova Solutions' is displayed, including details like 'Full Time', 'On-site', 'Posted 5 days ago', 'Benefits' (Health insurance, paid vacations, flexible working hours, learning opportunities), 'Salary' (₹85,000), and 'Key Requirements' (Proficient in HTML, CSS, JavaScript; and frameworks like React or Angular; Bachelor's degree in Computer Science or related field). On the right, the 'Application Form' is shown, requiring the user to complete the form to apply. The form includes fields for 'Full Name' (John Smith), 'Qualification' (MCA), 'Years of Experience' (a dropdown menu), 'Contact' (9365910718), and 'Resume' (a file upload area showing 'workhunt-john@gmail.com-resume.pdf' uploaded 1 week ago). Below the resume upload is a 'browse files' button and a note 'Want to update? Select a new file to replace current resume'. There is also a 'Skills' section with a text area containing 'Web Develpement (HTML, CSS, Javascript), PHP OOP, MVC Architecture, Laravel'. A 'Cover Letter (optional)' section with a text area 'Describe your qualifications...' is also present. At the bottom, there is a checkbox for 'I agree to the terms of application' and a 'Submit Application' button.

## 6. Employer Dashboard Page

The Employer Dashboard provides recruiters with an overview of their account and key metrics for managing job postings and applications. From here the employer can manage their listings, applications, and update company's details.

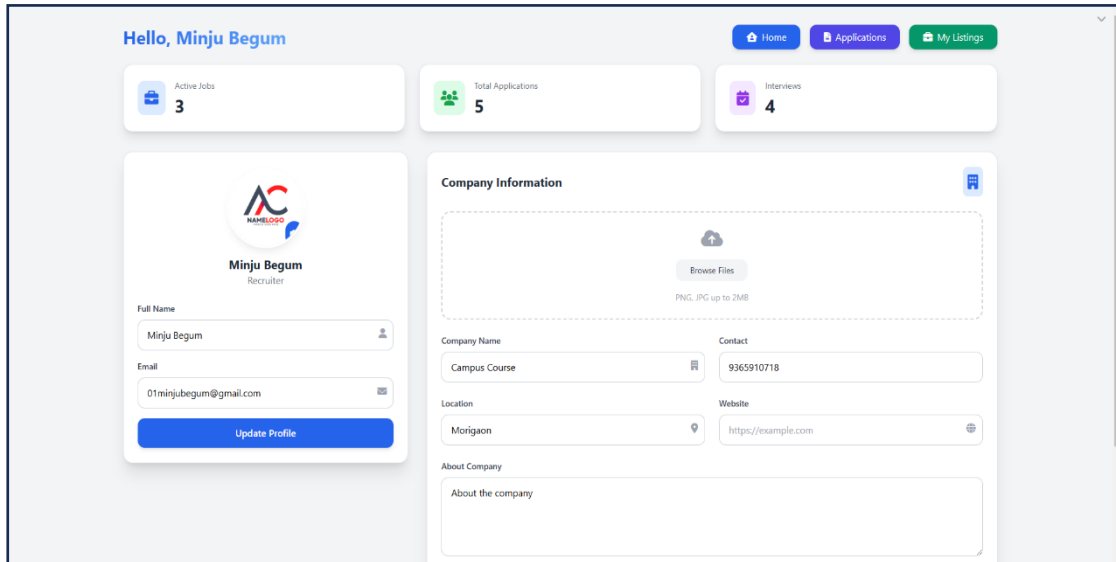
### Welcome Section:

- Displays a personalized greeting using the recruiter's name (e.g., "Hello, **Minju Begum**").
- Highlights their role (Recruiter) and profile image/logo.

### Summary Cards:

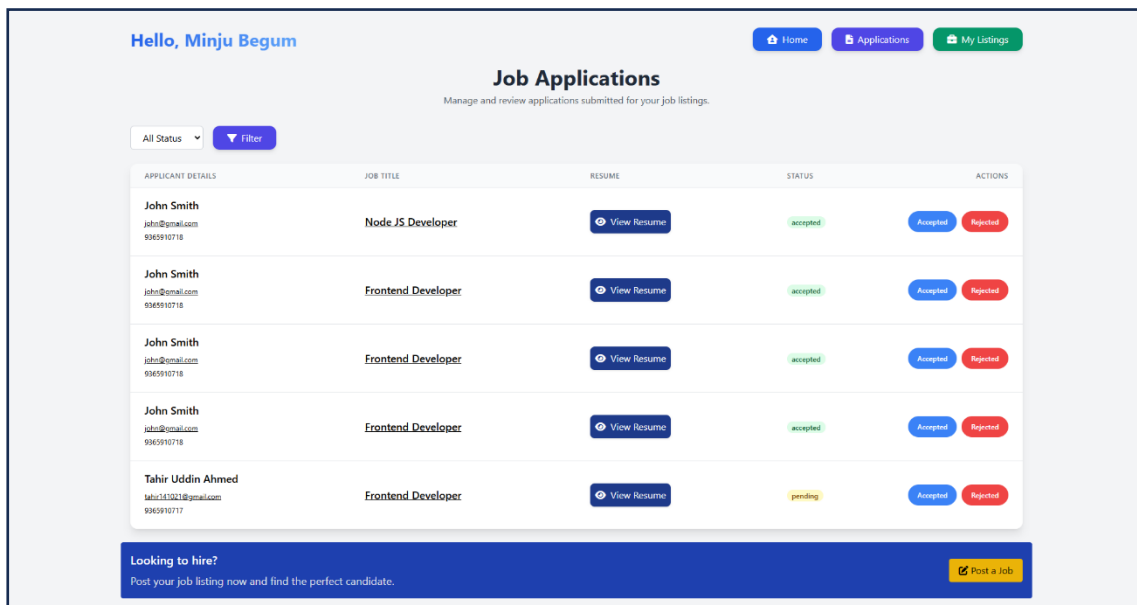
- **Active Jobs:** Displays the total number of active job postings (e.g., 3).
- **Total Applications:** Tracks the number of applications received (e.g., 5).
- **Interviews:** Shows the count of scheduled interviews (e.g., 4).

## Employer Dashboard Screenshot



### 7. Employer Dashboard - manage applications

This page is designed for employers to manage and review applications submitted for their job listings. It provides a clear and structured layout for reviewing applicant details, job titles, and application statuses.





## 8. Employer Dashboard - Job Listings Management Page

This page allows employers to manage their posted job listings effectively. It provides a snapshot of each job's details, including posting information, job type, and applications received. From here employer can update the job listing, and also create a new job listing.

Hello, Minju Begum

Home Applications My Listings

### Job Applications (3 Job listing)

Manage your job listings.

**Node JS Developer**  
Posted 1 week ago  
On-site  
₹85,000  
Applications: 1  
[Edit](#) [View](#)

**Frontend Developer**  
Posted 5 days ago  
On-site  
₹85,000  
Applications: 4  
[Edit](#) [View](#)

**Senior Software Developer**  
Posted 5 days ago  
Remote  
₹1,650,000  
Applications: 0  
[Edit](#) [View](#)

**Looking to hire?**  
Post your job listing now and find the perfect candidate. [Post a Job](#)

## Create Job Listings

Work Hunt

Welcome Minju Begum Logout Dashboard [Post a Job](#)

### Create Job Listing

#### Job Info

Job Title  
Software Engineer

Job Description  
We are seeking a skilled and motivated Software Developer to join our growing development team...

Annual Salary  
90000

Requirements  
Bachelor's degree in Computer Science

Benefits  
Health Insurance, 401k, paid time off

Tags (comma-separated)  
development, coding, java, python

Job Type  
Choose Job Type

Remote  
No

Address  
123 Main St

City  
Albany

State  
NY

ZIP Code  
12001

#### Company Info

Company Name  
Company name

Company Description  
Company Description

Company Website  
Enter website

Contact Phone  
Enter phone

Contact Email  
Email where you want to receive applications

Company Logo  
[Choose File](#) No file chosen

[Submit](#)

**Looking to hire?**  
Post your job listing now and find the perfect candidate. [Post a Job](#)

## Update existing Job listing

Work Hunt

Welcome Minju BegumLogoutDashboardPost a job

### Update Job Listing

#### Job Info

Frontend Developer

We seek a creative and detail-oriented front-end developer to design and implement user-friendly interfaces for our web applications.

85000

Proficient in (HTML, CSS, JavaScript), and frameworks like React or Angi.

Health insurance, paid vacations, flexible working hours, learning oppor

frontend, UI, CSS, JavaScript, React

#### Company Info & Location

TechNova Solutions

Delhi

Delhi

Delhi

9365910717

01minjubegum@gmail.com

Update

Cancel

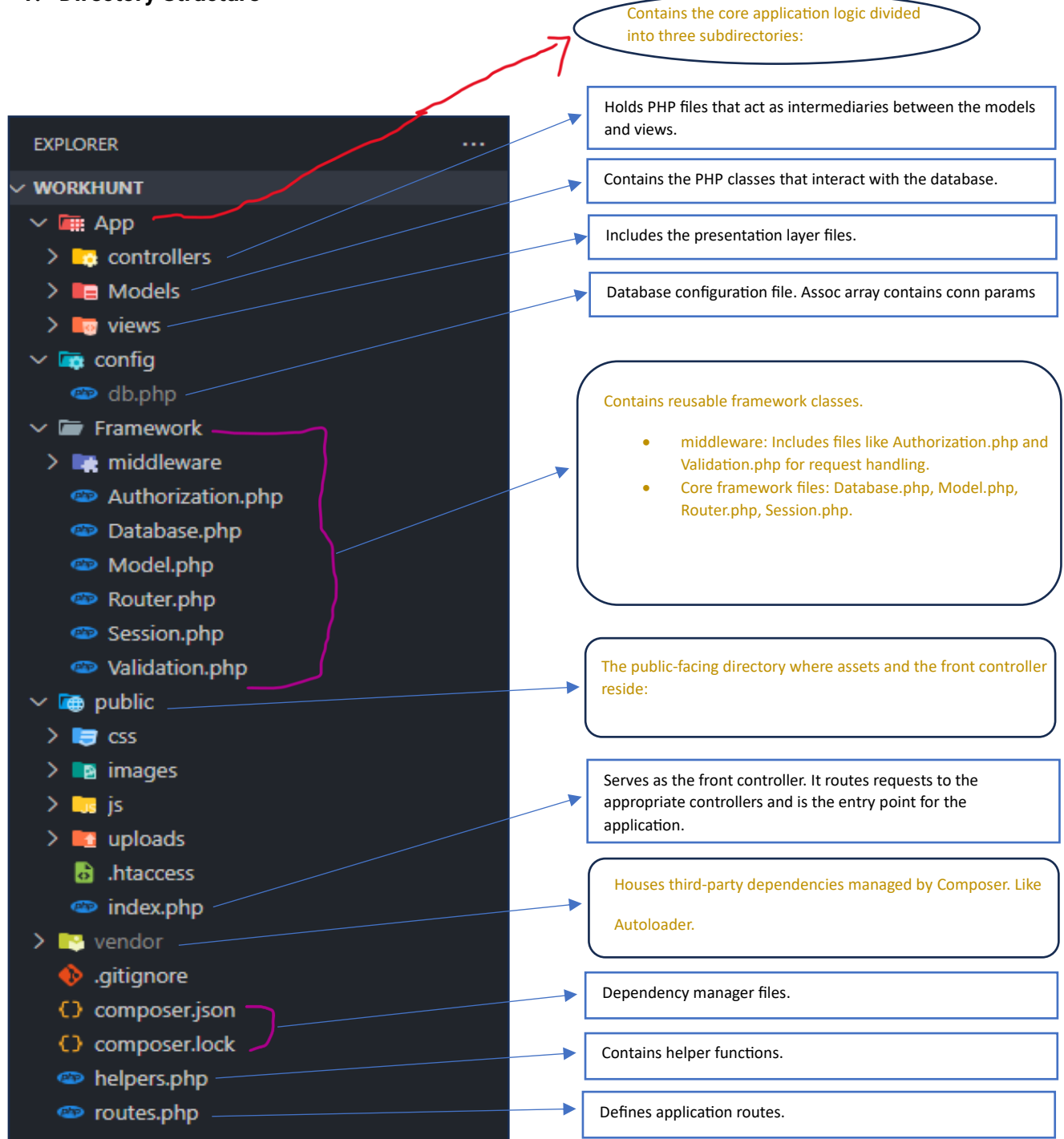
Looking to hire?

Post your job listing now and find the perfect candidate.

Post a Job

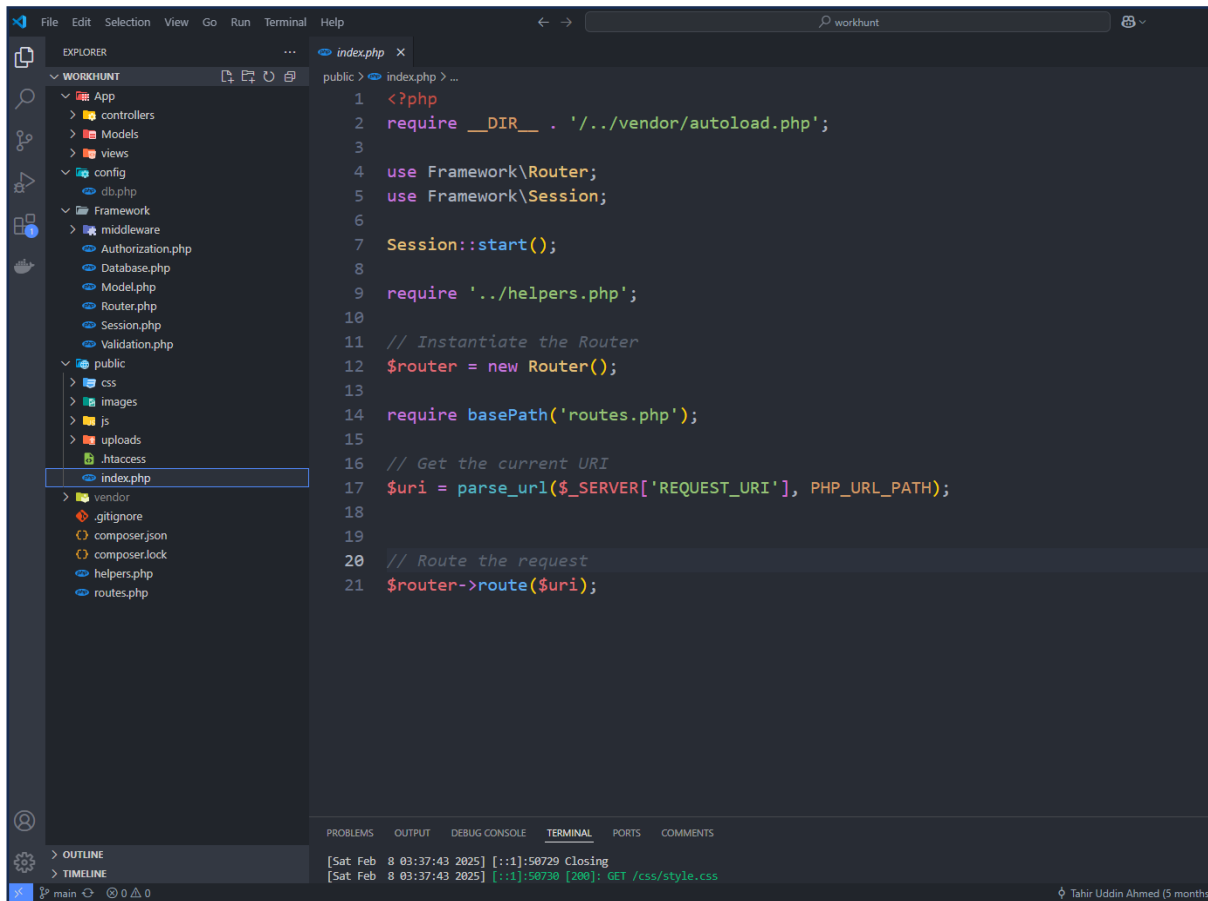
# Project Screenshots

## 1. Directory Structure



The **Workhunt** project directory is organized using the **MVC** architecture, with clear separation of core logic (**controllers, models, views**), configuration, reusable framework components, public assets, third-party libraries, and essential root files for dependency management and routing.

## 2. index.php [Entry Point]



The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar displays the project structure for 'workhunt'. The 'public' directory is expanded, showing files like 'css', 'images', 'js', 'uploads', '.htaccess', and 'index.php'. The 'index.php' file is selected. The main editor area shows the code for 'index.php' with line numbers 1 through 21. The code includes comments and PHP logic for autoloading, session management, helper functions, routing, and request parsing. At the bottom, the 'TERMINAL' tab shows a log of a successful GET request to '/css/style.css'.

```
1 <?php
2 require __DIR__ . '/../vendor/autoload.php';
3
4 use Framework\Router;
5 use Framework\Session;
6
7 Session::start();
8
9 require '../helpers.php';
10
11 // Instantiate the Router
12 $router = new Router();
13
14 require basePath('routes.php');
15
16 // Get the current URI
17 $uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
18
19
20 // Route the request
21 $router->route($uri);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

[Sat Feb 8 03:37:43 2025] [::1]:50729 Closing  
[Sat Feb 8 03:37:43 2025] [::1]:50730 [200]: GET /css/style.css

Tahir Uddin Ahmed (5 months)

The **index.php** file serves as the front controller and the single entry point for the **Workhunt** application. Its primary role is to initialize the environment, load necessary dependencies, and route incoming HTTP requests to the appropriate controllers.

Here is what it does:

- **Autoloading:** Includes **autoload.php** to load necessary dependencies via Composer.
- **Session Management:** Starts the session using a custom Session class.
- **Helper Functions:** Loads utility functions for the application.
- **Routing Setup:** Initializes the Router class to manage request routing.
- **Request Parsing:** Extracts the requested URI using **parse\_url**.
- **Route Handling:** Directs the request to the appropriate controller and action.

### 3. Database Configuration – db.php

```
db.php x
config > db.php
1  <?php
2
3  return [
4      'host' => 'localhost',
5      'port' => '3306',
6      'dbname' => 'workhunt',
7      'username' => 'TahirAhmed',
8      'password' => 'admin'
9  ];
```

### 4. Framework/ – Database class

```
<?php

namespace Framework;

use PDO;
use PDOException;
use Exception;

class Database {
    public $conn;

    /**
     * Constructor for database class
     *
     * @param array $config
     */
    public function __construct($config)
    {
        $dsn = "mysql:host={$config['host']};port={$config['port']};dbname={$config['dbname']}";

        $options = [
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
            PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_OBJ
        ];

        try {
            $this->conn = new PDO($dsn, $config['username'], $config['password'], $options);
        } catch(PDOException $e) {
            throw new Exception("Database connection failed: {$e->getMessage()}");
        }
    }

    /**
     * Query the database
     *
     * @param string $query
     *
     * @return PDOStatement
     * @throws PDOException
     */
    public function query($query, $params = []) {
        // inspectAndDie($query);
        try {
            $sth = $this->conn->prepare($query);
            // bind named params
            foreach($params as $param => $value) {
                $sth->bindValue(':'. $param, $value);
            }
            // execute the query
            $sth->execute();

            return $sth;
        } catch (PDOException $e) {
            throw new Exception("Query failed to execute: {$e->getMessage()}");
        }
    }
}
```

## 5. Framework/ - Session Class

```
<?php

namespace Framework;

class Session {
    /**
     * Start the session
     *
     * @return void
     */
    public static function start() {
        if(session_status() === PHP_SESSION_NONE) {
            session_start();
        }
    }

    /**
     * Set a session key/value pair
     *
     * @param string $key
     * @param mixed $value
     * @return void
     */
    public static function set($key, $value) {
        $_SESSION[$key] = $value;
    }

    /**
     * Get a session value by the key
     *
     * @param string $key
     * @param mixed $default
     * @return mixed
     */
    public static function get($key, $default = null) {
        return isset($_SESSION[$key]) ? $_SESSION[$key] : $default;
    }

    /**
     * Check if session key exists
     *
     * @param string $key
     * @return bool
     */
    public static function has($key) {
        return isset($_SESSION[$key]);
    }

    /**
     * Clear session by key
     *
     * @param string $key
     * @return void
     */
    public static function clear($key) {
        if(isset($_SESSION[$key])) {
            unset($_SESSION[$key]);
        }
    }

    /**
     * Clear All session data
     *
     * @return void
     */
    public static function clearAll() {
        session_unset();
        session_destroy();
    }

    /**
     * Set a flash message
     *
     * @param string $key
     * @param string $message
     * @return void
     */
    public static function setFlashMessage($key, $message) {
        self::set('flash_' . $key, $message);
    }

    /**
     * Get a flash message and unset
     *
     * @param string $key
     * @param mixed $default
     * @return string
     */
    public static function getFlashMessage($key, $default = null) {
        $message = self::get('flash_' . $key, $default);

        self::clear('flash_' . $key);

        return $message;
    }
}
```

## 6. Framework/ - Router class

```
<?php
namespace Framework;
use App\Controllers\ErrorController;
use Framework\Middleware\Authorize;
class Router {
    protected $routes = [];
    /**
     * Add a new Route
     *
     * @param string $method
     * @param string $uri
     * @param string $action
     * @param array $middleware
     * @return void
     */
    protected function registerRoute($method, $uri, $action, $middleware =
[]) {
        list($controller, $controllerMethod) = explode('@', $action);

        $this->routes[] = [
            'method' => $method,
            'uri' => $uri,
            'controller' => $controller,
            'controllerMethod' => $controllerMethod,
            'middleware' => $middleware
        ];
    }
}
```

```

public function get($uri, $controller, $middleware = []) {
    $this->registerRoute('GET', $uri, $controller, $middleware);
}

public function post($uri, $controller, $middleware = []) {
    $this->registerRoute('POST', $uri, $controller, $middleware);
}

public function put($uri, $controller, $middleware = []) {
    $this->registerRoute('PUT', $uri, $controller, $middleware);
}

public function delete($uri, $controller, $middleware = []) {
    $this->registerRoute('DELETE', $uri, $controller, $middleware);
}

public function route($uri) {
    $requestMethod = $_SERVER['REQUEST_METHOD'];
    // Check of _method input
    if($requestMethod == 'POST' && isset($_POST['_method'])) {
        //Override the request method with the value of _method
        $requestMethod = strtoupper($_POST['_method']);
    }
    foreach($this->routes as $route) {
        $uriSegments = explode('/', trim($uri, '/'));
        $routeSegments = explode('/', trim($route['uri'], '/'));
        $match = true;
        if(count($uriSegments) === count($routeSegments) &&
        strtoupper($route['method'] === $requestMethod)) {
            $params = [];
            $match = true;
            for($i = 0; $i < count($uriSegments); $i++) {
                // If the uri's do not match and there is no params
                if($routeSegments[$i] !== $uriSegments[$i] &&
                !preg_match('/\{(.+)\}/', $routeSegments[$i])) {
                    $match = false;
                    break;
                }
            }
        }
    }
}

```



```

// Check for the param and add to $params array
        if(preg_match('/\{(.+?)\}/', $routeSegments[$i],
$matches)) {
            $params[$matches[1]] = $uriSegments[$i];
        }
    }

    if($match) {
        foreach($route['middleware'] as $middleware) {
            (new Authorize())->handle($middleware);
        }
        // extract controller and controller method
        $controller = 'App\\Controllers\\' .
$route['controller'];
        $controllerMethod = $route['controllerMethod'];

        // Instantiate the controller and call the method
        $controllerInstance = new $controller();
        // call method
        $controllerInstance->$controllerMethod($params);
        return; // if found return the function
    }
}
}

ErrorController::notFound();
}

```

## **Future Work and Limitations**

While the **Workhunt** job portal has successfully met its core objectives, there is ample scope for further enhancements and additional features. Potential future improvements include:

1. **Advanced Search and Filtering:** Enhancements such as more granular filters (e.g., **salary range**, **experience level**, **job category**) and a recommendation engine powered by machine learning could improve the accuracy of search results and personalize job suggestions.
2. **Real-Time Notifications:** Implementing real-time notifications using technologies like **WebSockets** or push notifications would keep users informed about application updates, new job postings, and employer communications immediately.
3. **Enhanced Analytics:** Adding analytics features to provide both job seekers and employers with insights—such as application trends, candidate engagement, and job post performance—could help users make data-driven decisions.
4. **Improved Security Measures:** Continuously updating security protocols and performing regular vulnerability assessments would further safeguard user data and maintain trust in the platform.

### **Limitations**

Despite its robust design and functionality, the current version of **Workhunt** has some limitations:

1. **Scalability Constraints:** The system is designed as a prototype and may require further optimization to handle high traffic and larger data volumes efficiently, especially if the user base grows significantly.
2. **Limited Feature Set:** The initial version focuses on core functionalities such as user registration, job posting, and application processing. Advanced features like detailed analytics, real-time notifications, and resume parsing are not yet implemented.
3. **Security and Data Protection:** While standard security measures are in place, ongoing improvements and regular security audits are necessary to address emerging threats and ensure compliance with the latest data protection standards.

## Conclusion

The **Workhunt** project represents a comprehensive solution for modern online recruitment, successfully addressing the core challenges faced by job seekers and employers. Developed using modern web technologies—HTML, CSS, JavaScript, Tailwind CSS for the front end and PHP with MySQL under the MVC architecture for the back end—the project demonstrates a well-structured, scalable, and user-friendly job portal.

Key achievements of the project include:

- **Effective User Interaction:** The system facilitates seamless user registration, secure login, intuitive job search, and straightforward application processes, ensuring that job seekers can easily navigate and utilize the platform.
- **Robust Employer Tools:** Employers can efficiently create, manage, and monitor job listings through dedicated dashboards, making it easier to connect with the right candidates and streamline recruitment processes.
- **Modular and Maintainable Architecture:** The implementation of the MVC framework has enabled a clear separation of concerns, enhancing code maintainability and scalability. This structure not only supports current functionalities but also lays the foundation for future enhancements.

The **Workhunt** project not only meets its initial objectives by providing a functional and responsive job portal but also demonstrates the practical application of web development best practices and the MVC architectural pattern. This project has enriched our understanding of designing, developing, and testing modern web applications and has laid the groundwork for further improvements and innovations in the online recruitment domain.

### **Thank You!**

I would like to extend my sincere gratitude to my professors, mentors for their continuous support and guidance throughout this journey. Their encouragement and feedback have been instrumental in the successful completion of this final year project.

## Resources

Here are some potential resources that you can include in your project report for the **workhunt** System:

➔ Websites and Documentation:

- **W3Schools** (<https://www.w3schools.com/>)
- MDN Web Docs for HTML, CSS, and JavaScript (<https://developer.mozilla.org/>)
- Bootstrap Documentation (<https://getbootstrap.com/docs/>)
- PHP Documentation (<https://www.php.net/docs.php>)
- MySQL Documentation (<https://dev.mysql.com/doc/>)

➔ Forums and Communities:

- Stack Overflow (<https://stackoverflow.com/>)
- GitHub (<https://github.com/>)