## INTRODUCTION

**1.INTRODUCTION**

Online Job portal system is an application which connects employer and job seekers where employers are the source of the resources and the job seeker can find and apply for their targeted job.

In Online Job portal system we use PYTHON and DJANGO database. This is the project which keeps records of the employer, jobseeker and administrator. Online Job portal system has three modules i.e. Jobseeker, Employer and Admin

**OBJECTIVE:**

Computerized online Job Search System is developed to facilitate the General administration system to manage the various information of the Job Seeker and Employer and the processes involved in a placement company. So, that organization can access accurate information quickly and easily as and when required, thereby improving its operational efficiency & effectiveness

In today’s competitive environment, where Everybody wants to be on the top, Information plays very crucial role. As fast as information is accessed and processed, it can give good results.

Today Internet is the fast way of transferring Data and Information over wide area, hence I have used internet as a way for exchanging information.

Computerized system helps to fulfill these goals. Computerization of the official works will help in doing lot of manual work quickly. It will help in easy storage and access of all information, in short period of time.

### 

### SYSTEM STUDY AND ANALYSIS

#### 2.SYSTEM STUDY AND ANALYSIS

**2.1**  **EXISTING SYSTEM**

The present system requires applicants to search through print and visual media for job opportunities. Applicants need to apply for jobs using conventional methods and appear for interview on a specified date at a specified location. Employers need to advertise the vacancies and sort all applicant details, conduct selection procedures and complete the formalities. This approach is tedious and requires much effort and resource

**2.2**  **PROPOSED SYSTEM**

The proposed system is a web based application which allows applicants and employers to register their details. Applicants can browse through the vacancy details that are posted and can apply for the jobs online. Employers can browse through the posted resumes and select suitable candidates.

**2.3 FEASIBILITY STUDY**

Feasibility study is a procedure that identifies, describes and evaluates candidate system and selects the best system for the job. An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study will decide if the proposed system will be cost effective from a business point of view and if it can be developed given existing budgetary constraints. The key considerations involved in the feasibility analysis are economic, technical, behavioural and operational**.**

**2.3.1 Operational Feasibility**

The system operation is the longest phase in the development life cycle of a system. So, operational feasibility should be given much importance. The users of the system don’t need thorough training on the system. All they are expected to know to operate

**2.3.2 Technical Feasibility**

It centers on the existing computer system and to what extent it can support the proposed addition. Since the minimum requirements of the system like a server and a browser on the client, are met by any average user.

**2.3.3 Economical Feasibility**

The economic analysis is to determine the benefits and savings that are expected from a candidate system and compare them with costs. The system is economically feasible, as the organization possesses the hardware and software resources required for the functioning of the system. Any additional resources, if required, can also be easily acquired.

**2.4 M0DULES AND MODULE DESCRIPTION**

ADMIN :

This module provides administrator related functionalities. Administrator manages entire application and maintains the profiles of applicants and employers.

**Dashboard**: In this sections, admin can briefly view the total job category, total employer, total candidates and total job.

**Job Category**: In this section, admin can manage jobcategory (add/update/delete).

**List of Employer**: In this section, admin can view the list of employer and see the detail of individual employer.

**Reg Jobseeker**: In this section, admin can view the list of candidates(jobseeker) and see the detail of individual candidates(jobseeker).

**Pages:** In this section, admin can manage the about us and contact us pages.

**Reports**: In this section admin can generate report of how many employer and candidates register between two dates.

**Search**: In this section, admin can search a particular employer and candidates by company name and mobile number respectively.

Admin can also update his profile, change the password and recover the password.

**Employer Module**

This module provides functionalities related to employers. Employers can post vacancy details and update the details as and when necessary. Employers can search through applicant resumes based on different criteria.

**Jobs:** In this section, employer can post the job and manage the jobs.

**Candidates List:** In this section employer can view the list of applied candidates and send message to selected candidates.

**Reports:** In this section employer can view how many candidates apply for a job in particular periods.

Employer can also update his profile, change the password and recover the password.

**Candidates(Jobseeker) Modules**

**Home:** In this section, candidates can view job which is posted by employer and apply those jobs.

**Applied Jobs:** In this section, candidates can a view response of applied job.

**About Us:** In this section, candidates can view the about us page of website.

**Contact Us:** In this section, candidates can view the contact us page of website.

Candidates(Jobseeker) can also update his profile, change the password and recover the password.

**SYSTEM REQUIREMENT SPECIFICATION**

**3.SYSTEM REQUIREMENT SPECIFICATION**

###### SOFTWARE REQUIREMENTS

Operating System : Windows 10

Front end : PYTHON

Back end : MySQL Server

**PYTHON**

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small- and large-scale projects.

**FEATURES OF PYTHON**

* Easy to use.
* Free and open Source
* Object Oriented language
* GUI programming support
* High level language
* Extensible language
* Portable language
* Integrated language

###### MySQL SERVER

MySQL Server is a family of products that meet the data storage requirements of the largest data processing system and commercial Web sites, yet at the same time can provide easy to use data storage services to an individual or small business.

Enterprise manager is the administrative console for SQL Server installations. You can perform high-level administrative functions that affect one or more services, schedule common maintenance tasks or create and modify the structure of individual databases Query Analyzer offers a quick and dirty method for performing queries against any of your SQL Server databases.

Data in access is organized in the form of tables just as SQL. Within a table records are arranged according to common references value, known as “Primary Key” or the “key field”. The value in the key field is different for every record and thus help in uniquely identified records. Since value in key field can be replicated across other table there should be a way to maintain a relation between the two fields. This is implemented through the concept of “Foreign Key”. A foreign key in a table is a field, which links the table. A Database Management System (DBMS) usually has a Database Definition Language (DDL) which allows for the field types and tables to be defined and a Database Manipulation Language (DML) which allows for the retrieval and update of data. The manipulation language often comes in several formats allowing access to the database from the variety of programming languages.

###### Features of SQL

* + - SQL can be used by a range of users including those with little or no programming experiences.
    - It is a non-procedural language.
    - It reduces the amount of time required for creating and maintaining systems.
    - It is English like language.

###### HARDWARE REQIREMENTS

Processor : X86 or X64Compactable Processor

Hard disk : 20GB or higher

RAM : 512MB

Clock Speed Memory : 256 MB or higher Keyboard : Standard

Mouse : Standard

System bus : 32 bit

# COST AND BENEFIT ANALYSIS

##### 4.COST BENEFIT ANALYSIS

Cost/Benefit analysis is a procedure that gives a picture of the various costs, benefit, and rules associated with a system. An analysis of the cost and benefits of each alternative guides the selection process. Therefore, the analyst needs to be familiar with the cost and benefits categories and the evaluation methods before a final selection can be made. In developing cost estimates for a system, we need to consider several cost elements. It includes:-

**HARDWARE COST**: Relate to the actual purchase or lease of the computer and peripherals. Determining the actual of hardware is generally more difficult when the system is shared by various users than for dedicated standalone system.

**SOFTWARE COST**: It includes software required to run the system. It is considerably reduced because all software used by the system except Windows XP operating system is freeware.

**PERSONAL COST**: Includes staff salaries and benefits as well as cost of training of those human resources involves developing the system. Costs incurred during the development of a system are onetime costs and are label developmental costs.

**OPERATING COST AND SUPPLY COST**: Operating cost includes all costs associated with the day-to-day operation of the system. The amount depends on the number of shifts, the nature of the applications, and the caliber of the operating users, general cost to run the system, electricity charges etc. There are various ways of covering operating costs. Supply costs are variable costs that increase with increased use of paper, ribbons etc. They should be estimated and included in the overall cost of the system.

**SYSTEM DESIGN**

##### 5. SYSTEM DESIGN

The most creative and challenging phase is system design, is a special solution to how to approach to the creation of the proposed system. Design is a multiprocessing that focuses on the structure software application. The design process translate the requirements into the representation of software that can be assessed for quality before coding being. The design is transition from a user oriented document to the programmers or database personal. The purpose of the design phase are to plan a selection for the problem specified by the required document. The goal of the design process is to procedure a representation of a system; which is to be used later build that system.

The first step is to determine how the output is to be produced and in what format. Second, input data have to be designed to meet the requirements of the proposed output. The operational phase is handled though program construction and testing.

Design is a creative process; a good design is the key to effective system. The term “DESIGN” is defined as “The process of applying various techniques and principles for the purpose of defining a process or a system in sufficient details to permit its physical realization”. The design specification describes the features of the system, the components or the elements of system and their appearance to end users. The system design transforms a logical representation of what a given system is required to be in to a physical specification. In the system design high end decisions are taken regarding the basic system architecture, platforms and tools to be used. Important design factors such as reliability, response time, though put of the system, maintainability, expandability etc. should be taken in to account.

###### INPUT DESIGN

Once the output requirements have been finalized, the next step is to find out what inputs are needed to produce the desired outputs. In accurate input data results in errors in data processing. Errors entered by data entry operator can be controlled by input design. Input design is a process of converting user originated inputs to computer-based format. The various objectives of the input design should focus on;

* + - Controlling input of data.
    - Avoiding delay.
    - Avoiding errors in data.
    - Avoiding extra steps.
    - Keeping the process simple.

Input is considered as the process of keying in data in to system, which will be converted in to the system format. A document should be consist because longer documents contain more data and so take longer to enter and have a greater change of data entry errors. The more quickly an error is detected the closer the error is to the persons who generated it and so the error is more easily corrected. A data input specification is a detail description of the individual fields (data elements) on an input document together with their characteristics. Be specific and precise, not general, ambiguous, or vague in case of error messages.

In the system design phase, the expanded data flow diagrams identify logical data flows, sources and destinations. A system flow chart specific master files, transaction files and computer programs input data are collected and organized into groups of similar data.

###### OUTPUT DESIGN

The output from computer system are required mainly to communicate the results of processing to users. They are also used to provide permanent (“hard”) copy of these results for letter consultation. Output is what the client is buying when he or she pays for a development project. Inputs, databases and processes exit to provide output. Printout should be designed around the output requirement of the user. The output devices are considered keeping in mind factors such as compatibility of the device with the system response time requirements, expected print quality and number of copies needed.

Output to be produced also depends on following factors:

* + - Type of user and purpose.
    - Contains of output.
    - Format of the output.
    - Frequency and timing of out
    - Volume of output.
    - Sequence and Quality.

###### PROPOSED DESIGN

* + 1. DATA FLOW DIAGRAM

The data flow diagram was developed as a way of expressing system requirements in a graphical form. A DFD also known as (bubble chart) has the purpose of clarifying system requirements and identifying major transformation that will become programs in system design. So it is the starting point of the design phase that functionally decomposes that requirements specification down to the lowest level of detail.

A DFD consist of a series of bubbles joined by lines. The bubbles represent data transformation and the lines represent data flows in the system. In data flow diagrams the symbol set is comprised of diagram entity, process, data stores and data flow. An entity is used to defined the boundaries of the system. It is an external component of the system. E.g. , a department, that interest with the existing system. A process is defined as a work or action performed by people, machines etc., Within a system. Data store is used to store data. Data stores are represented by open boxes. A data flow is any item that carries data to, within or from the system. That is, it is used to represent inputs and outputs of the system.

There are two types of data flows-Physical DFD and Logical DFD:

PHYSICAL DFD

A physical DFD is an implementation-depended view of the system, showing what functions are performed. A physical diagram provides details about hardware, software, fields and people involved in the implementation of the system. Physical characteristics include names of the people, names of the departments, names of the fields, location etc.

LOGICAL DFD

Logical DFD is an implementation-independent view of the system, that focuses only on flow of data between different process and activities. Logical diagrams show how the business operates: not how the system can be implemented. It explains the even of the system and the data required by each event of the system.

Physical DFD differs from the logical DFD in the following ways:

* + - * Physical DFD is implementation depend whereas logical DFD is implementation independent.
      * Physical diagram in the physical DFD’s provide such low-level details as hardware and software requirements of a system whereas logical diagrams in the logical DFD’s explain only the event involved in the system and the data require to implement each event of the system.

**DFD SYMBOLS**

In the DFD, there are four symbols as follows:

1.

A rectangle defines a source of destination of system data or it represents external entities.

2.

An arrow identifies dataflow-data in motion. It is a pipeline through which information flows

.

3.

A circle or a “bubble” represents a process that transforms incoming data flow into outgoing data flows.

4.

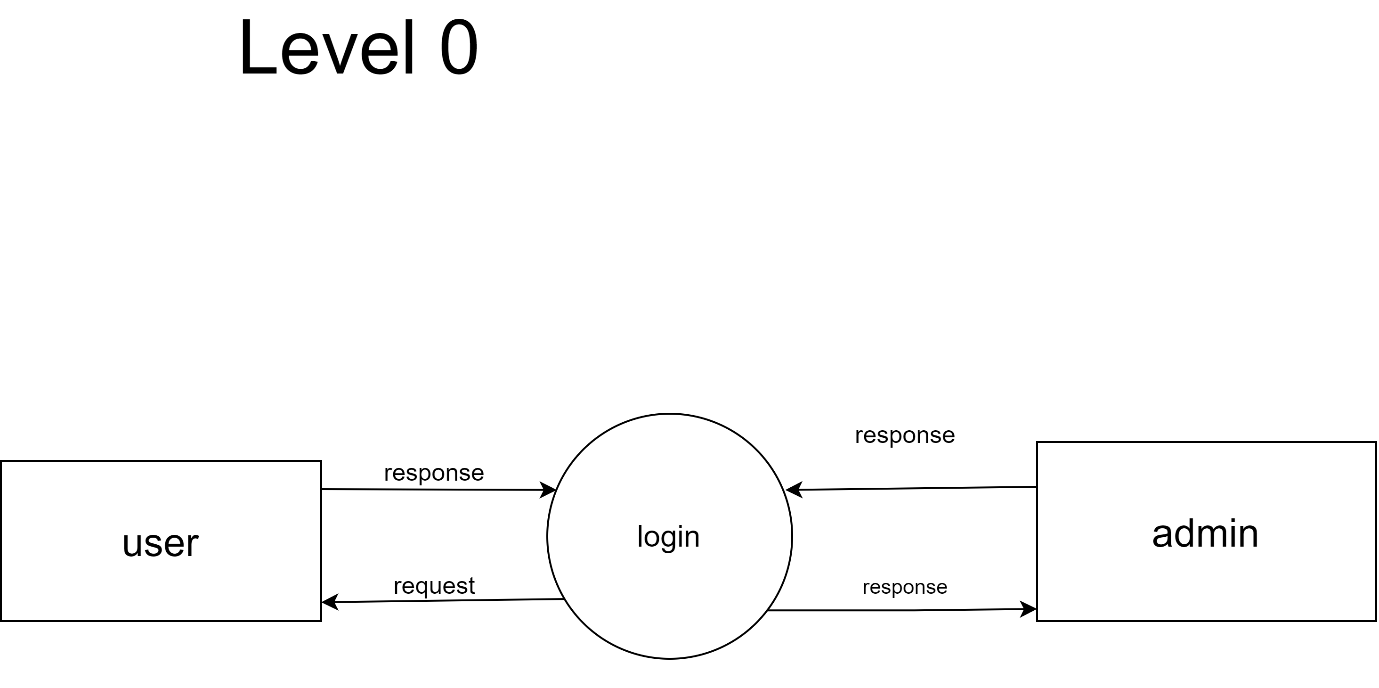
An open rectangle is a data store-data at rest, or a temporary repository of data. They should have several names. If a process merely uses the content of store and does not alter it, the arrow head goes only from the store to process. If a process alter the details in the store then a doublehead arrow is used.

Several rules of thumb are used in drawing DFDs:

* + - * Process should be named and numbered for easy reference.
      * Each name should be representative of the process.
      * The direction of flow is from top to bottom and from left to right.
      * Data traditionally flows from the source to destination.
      * When a process is exploded into lower level details, they should be numbered.
      * The names of data stores, sources and destinations are written in capital letters.

Process and data flow names have the first letter of each word capitalized

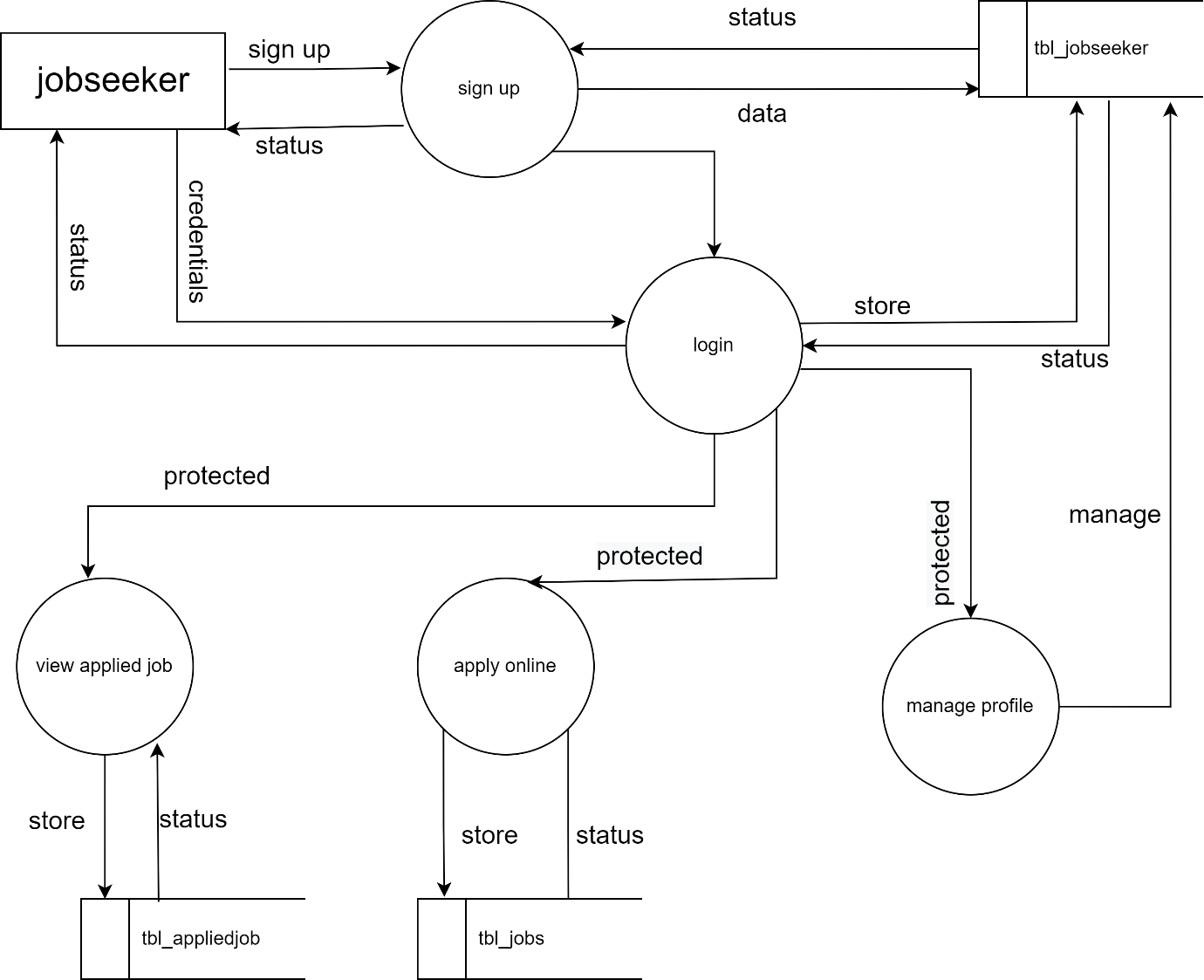
##### CONTEXT DATA FLOW DIAGRAM

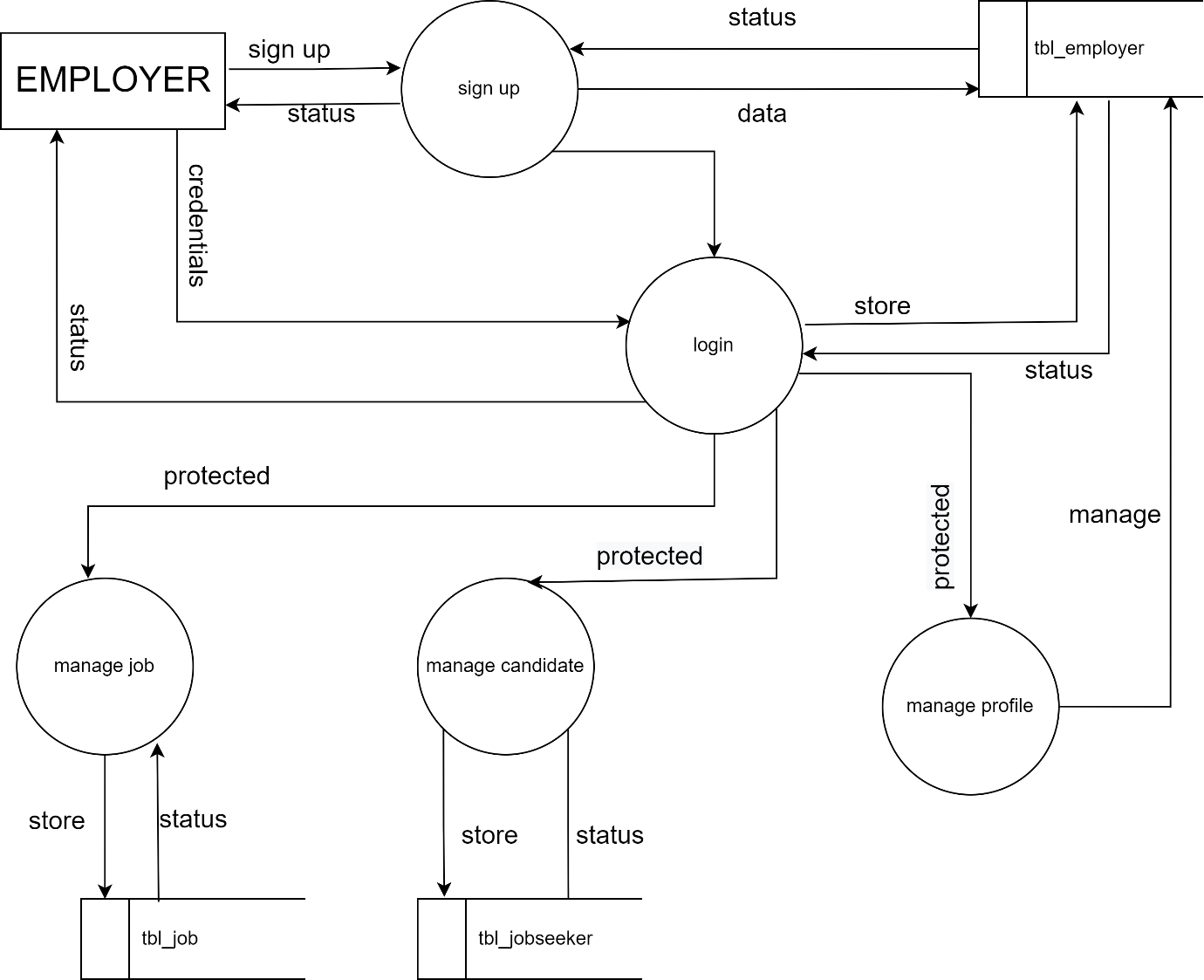


LEVEL 1 : ADMIN



LEVEL 1: jobseeker





LEVEL 1: EMPLOYER

5.1 **E-R DIAGRAM**

-

The Entity Relationship model facilitates database design by enabling the designer to express the logical properties of the database in an enterprise schema. Identification of real world objects referred to as entities forms the basis of this model. These entities are described by their attributes and are connected by relationships among them. The E-R model has its own set of symbols for drawing the E-R diagram which depicts the logical model of the database.

An E-R diagram can graphically represent the overall logical structure of a database. The relative simplicity and pictorial clarity of this diagramming technique may well account in large part of the widespread use of the E-R model.

Symbols used for E-R diagram are as follows:



Represent entity



Represent key attribute



Represent primary key

 Represent decision making

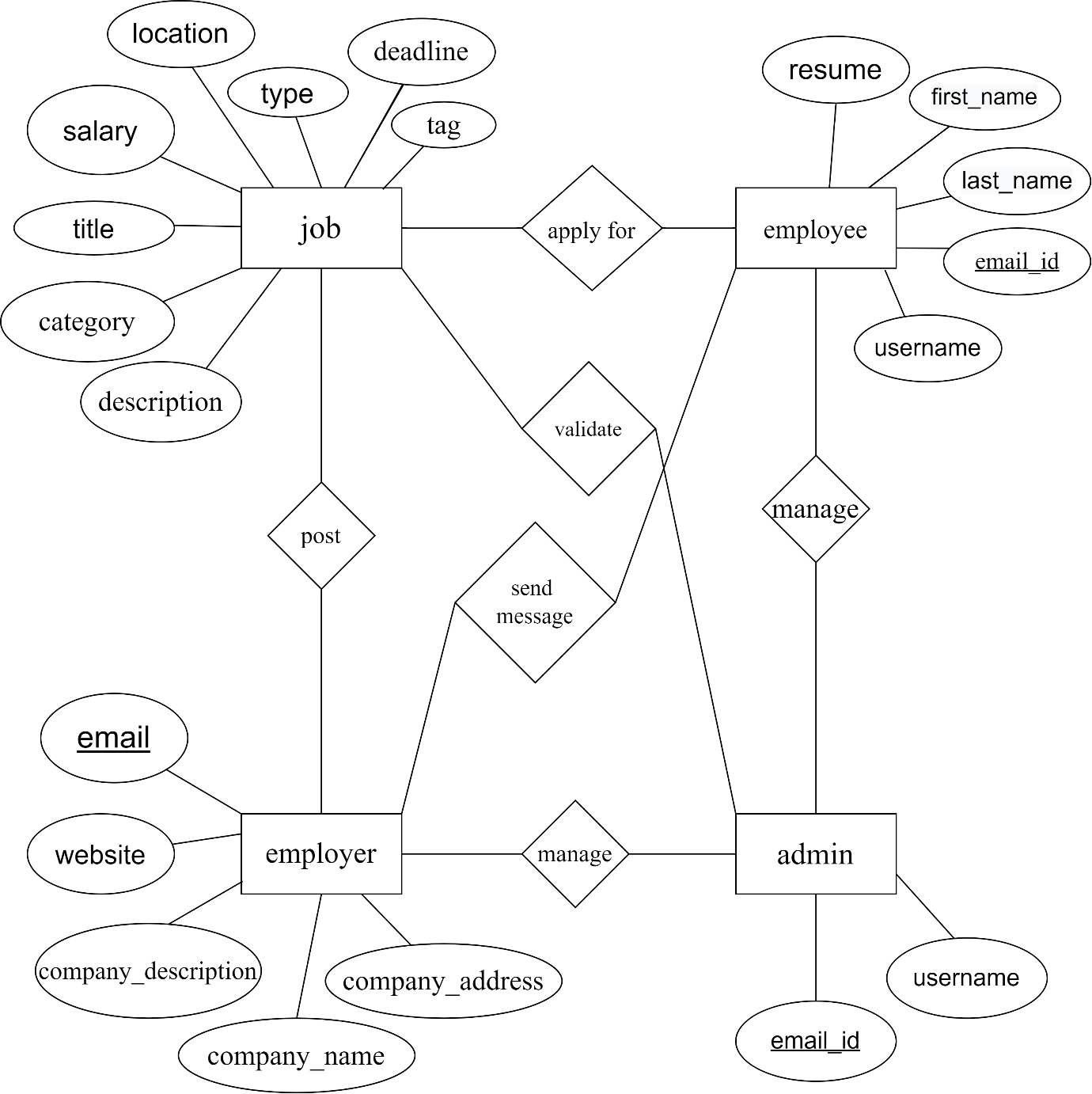
##### 

##### 

##### 

##### 

##### E-R DIAGRAM

****

* 1. DATABASE DESIGN

A database is a collection of related information stored so that it is available to many users for different purpose. The content of database is obtained by combining data from all the different sources in an organization so that data are available to all users and redundant data can be eliminated or at least minimized. Database was designed using RDBMS concept there by enabling the sharing of data and was normalized to avoid redundancy. This will lead to quicker application development with low maintenance cost.

The goal of database design is to ensure that the data represented in such a way that there is no redundancy and no extraneous data is generated. Once the conceptual model is designed, it is can be mapped to the DBMS/RDBMS that is actually being used. A group of table with related data in them is called a database.

**TABLES**

**Table name: Admin**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field name** | **Data type** | **Constraints** | | **Description** | |
| id | int | Primary key (AUTO\_INCREMENT) | | Unique id for identifying admin | |
| AdminName | varchar(20) | Not null | | Name of admin | |
| Email | Varchar(30) | Not null | | Email address of admin | |
| username | Varchar(20) | Not null | To identify the admin | |
| Password | Varchar(20) | Not null | | Strong password for login | |
| Mobilenumber | Varchar(20) | Not null | | Contact number of admin | |
| Adminregdate | timestamp | Current\_timestamp() | | Time of registartion | |

**Table name:Applyjob**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field name** | **Data type** | **Constraints** | **Description** |
| id | int | Primary key(AUTO\_INCREMENT) | Unique id for identifying applied  job |
| userid | int | Foreign Key | Identifying the user |
| jobid | int | Foreign Key | Identifying the job |
| Applydate | timestamp | Current\_timestamp() | Time of apply |
| Status | Varchar(20) | none | status |
| ResponseDate | timestamp | null | Time of Response |

**Table name: Categories**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field name** | **Date type** | **Constraints** | **Description** |
| ID | Int | Primary key(AUTO\_INCREMENT) | Identifying the categories by cat\_id |
| cat\_name | varchar(25) | Foreign Key | names of the categories |
| Description | text | null | Details of each categories |
| PostingDate | timestamp | Current\_timestamp() | Time of posting |
| UpdationDate | timestamp | null | Time of update post |
| Is\_Active | Int(1) | none | status |

**Table name: Education**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field name** | **Data type** | **Constraints** | **Description** |
| id | Int | Primary key (AUTO\_INCREMENT) | Unique id for education |
| userid | int | Foreign Key | Identifying the user |
| qualification | Varchar(50) | null | qualification of user |
| clgORschname | Varchar(50) | null | Name of the school or colleae |
| Passingyr | Varchar(50) | null | Pass out year |
| Stream | Varchar(50) | null | Name of the stream |
| Cgpa | Varchar(50) | null | mark |
| Percentage | Varchar(50) | null | mark |
| CreationDate | timestamp | Current\_timestamp() | Time of creation |

**Table name:Message**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field name** | **Data type** | **Constraints** | **Description** |
| id | Int | Primary key(AUTO\_INCREME NT) | Unique id for message |
| job\_id | int | Foreign Key | Unique id for job |
| user\_id | int | Foreign Key | Unique id for user |
| Message | Text | null | Message that can be sent |
| status | Int | Null | To know the status |
| Responsedate | timestamp | onupdate\_timestamp() | Time of Response |

**Table name:Experience**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field name** | **Data type** | **constraints** | **Description** |
| id | Int | Primary key(AUTO\_INCREMENT) | Unique id for experience |
| user\_id | int | Foreign Key | Unique id for user |
| employername | varchar(50) | Null | Name of employer |
| employementType | varchar(10) | null | Type of employer |
| Designation | varchar(20) | null | Role of employee |
| Fromdate | varchar(20) | Null | Experience date |
| Todate | varchar(20) | Null | Experience date |
| skills | varchar(20) | Null | Skills |
| Creationdate | timestamp | Current\_timestamp() | Time of creation |

**Table name:employers**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field name** | **Data type** | **constraints** | **Description** |
| id | Int | Primary key(AUTO\_INCREMENT) | Unique id for employee  identification |
| Concernperson | varchar(50) | nulll |  |
| empemail | Varchar(30) | Not null | Email address of employee |
| emppassword | Varchar(20) | Not null | Strong password for login |
| Companyname | Varchar(100) | Not null | Company name |
| Companytagline | Varchar(50) | Not null |  |
| companydescription | Varchar(100) | Not null | Company description |
| companyurl | Varchar(50) | Not null | Website of company |
| companylog | Varchar(50) | Not null | Logo of company |
| noofemployee | int | Not null | Number of employee |
| industry | Varchar(50) | Not null | industry |
| location | Varchar(50) | Not null | location |
| establishdin | Varchar(50) | Not null | Established in |
| Regdate | timestamp |  | Date of registration |
| lastupdation | timestamp | Current\_timestamp() | Last update |
| isactive | int | none | status |

**Table name:jobs**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field name** | **Data type** | **constraints** | **Description** |
| id | Int | Primary key(AUTO\_INCREMENT) | Unique id for job |
| emp\_id | Int | Foreign Key | Unique id for employee  identification |
| jobcategories | int | Foreign Key | Identifying the categories |
| jobtitle | Int | Not null | Quantity of product purchased |
| jobtype | Varchar(50) | Not null | Type of job |
| salarypackage | Varchar(50) | Not null | Salary details |
| skillsrequired | Varchar(50) | Not null | Skills required for job |
| experience | Varchar(50) | Not null | Experience required for job |
| joblocation | Varchar(50) | Not null | Job location |
| jobdescription | text | Not null | Job details |
| jobexpdate | date | Not null | Exp date |
| postindate | timestamp | Current\_timestamp() | Posting date |
| updationdate | timestamp | Current\_timestamp() | Last update |
| isactive | int | none | status |

**Table name:jobseekers**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field name** | **Data type** | **constraints** | **Description** |
| id | Int | Primary key(AUTO\_INCREMENT) | Unique id for jobseekers |
| fullname | varchar(25) | Not nulll | Name |
| password | Int | Not null | Strong password for login |
| resume | varchar(200) | Not null | Resume |
| aboutme | varchar(200) | Not null | Unique id for area identification |
| phone | varchar(10) | Not nulll | Mobile number |
| email | varchar(50) | Not null | Mail id |
| profilepic | varchar(50) | Not null | Photo of job seeker |
| skills | varchar(50) | Not null | Skill of joobseker |
| regdate | timestamp | Current\_timestamp() | Date of registration |
| lastupdationdate | timestamp | update\_timestamp() | Last update |
| isactive | int | notnull | status |

**SYSTEM TESTING AND IMPLEMENTATION**

**6.SYSTEM TESTING AND IMPLEMENTATION**

* 1. **SYSTEM TESTING**

Testing is a set of activity that can be planned in advance and conducted systematically. Testing begins at the module level and work towards the integration of entire computers based system. Nothing is completed without testing, as it is vital to the success of the system. System testing makes a logical assumption that if all parts of the system are corrected, the goal will not appear until months later. The process of evaluation a system by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual result.

PURPOSE OF THE TESTING

Testing is the success of the system. System testing makes a logical assumption that if all part of the system is correct, the goal will be successfully achieved. The following points shows how testing is essential.

* + - Existence of program defects of inadequacies is inferred.
    - Verifies whether the software behave as intended by its designer.
    - Checks conformance with requirements specification or user need.
    - Access the operational reliability of the system.
    - Test the performance of the system.
    - Reflects the frequencies of actual user inputs.
    - Find the fault which caused the output anomaly.
    - Detect flaws and deficiencies in requirements.
    - Exercise the program using data like the real data processed by the program.
    - Test the system capabilities.
    - Judges whether or not the program is usable in practice.

TESTING OBJECTIVES

There are several rules that can serve as testing objectives. They are;

* + - Testing is a process of executing a program with the intent of finding error.
    - A good test case is one that has high probability of finding an undiscovered error.
    - A successful test is one that uncovers an undiscovered error.

If testing is conducted successfully according to the objectives as stated above, it would uncover error in the software. Also testing demonstrates that software functions appear to the working according to the specifications, that performances requirement appear to have been met. There are three ways to test a program

* + - For correctness.
    - For implementation Efficiency.
    - For Computational Complexity.

Testing for correctness are supported to verify that a program does exactly what it was designed to do. This is much more difficult that it may at first appear, especially for large programs. Tests for implementation efficiency attempt to find ways to make a correct program faster or use less storage. It is a code-refining process, which re-examines the implementation phase algorithm development. Tests for computational complexity amount to an experiment analysis of the complexity of an algorithm or an experiment comparison of two or more algorithms, which solve the same problem.

**TYPES OF TESTING**

System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiency before live operation commences. Testing is vital to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct.The goal will be successfully achieved. The candidate system is subject to a variety of test. A series of tests are performed for the proposed system before the system is readfor system acceptance testing.The various levels at which testing are conducted;

* Unit testing
* Integration testing
* Sequential testing
* System testing
* User Acceptance testing
* Input testing
* Output testing
* Validation testing

Of these we discuss only Unit testing, Integration testing, User acceptance testing and validation testing.

**UNIT TESTING**

In unit testing each program unit is tested individually. So any errors in a unit are debugged. Sample data is given for unit testing. The unit test results are recorded for future references. Unit testing focus verification efforts on the smallest unit of software design, the module. This is known as “Module testing”. It comprises of the set test performed by an individual programmer prior to the integration of unit in to the large system. The modules are tested separately. This testing is carried out during programming stage itself.

**INTEGRATION TESTING**

Integration testing is a systematic technique for constructing the program structure while at the same time conducting test to uncover errors associated with interfacing. The objective is to take unit tested modules and to combine them and test it as a whole. In this step all errors are encountered are corrected for next testing.

**USER ACCEPTANCE TESTING**

User Acceptance of a system is the key factor for the success of any system. User Acceptances testing is running the system with live data by the actual user. An acceptance test has the objective of selling the user on the validity and reliability of the system. A comprehensive test report is prepared. The report indicates the system’s tolerance, performance range, error rate and accuracy. It verifies the system procedures operate to system specification and the integrity of important data is maintained. Performance of an acceptance test is actually the users show. User motivation is very important for the successful performance of the system. After that a comprehensive test report is prepared. This report shows the systems tolerance, performance range, error rate and accuracy.

VALIDATION TESTING

Here the inputs are given by the user validated. That is password validation, format of data are correct, text box validation. Changes are need to be done after result of this testing. Verification testing runs the system in a simulated data. Validation refers to the process of using software in order to find errors. The feedback from the validation phase generally produces changes in software to deal with errors and failures that are uncovered. Validation may continue for several months. During the course of validating the system, failure may occur and the software will be changed. Continued use may produce additional failures and need for still more changes. Proper validation checks redone in case if insertion and updating of tables, in order to see that no duplication of data has occurred.

Validation checks are performed on the following fields.

* + - Text field:

The text can contain only the number of characters less than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes an error message.

* + - Numeric field:

The numeric field can contain only numbers from 0 to 9. An entry of a character flashes an error message. The individual modules are checked for accuracy and what it has to perform. Each module is subjected to test run along with sample data. The individual tested modules are integrated into a single system.

###### SYSTEM IMPLEMENTATION

Implementation is the stage in the project where the theoretical design is turned into a working system and is giving confidence on the new system for the user that will work efficiently and effectively. It involves careful planning, investigation of the current system and its constraints on implementation, design of methods, to achieve the change over an evaluation of changes over methods. Apart from planning major implementation process begins with preparing a plan for the implementation of the system. According to these plans the activities are to be carried out, discussions made regarding the equipment and resources and the additional equipment has to be acquired to implement the new system.

Implementation is the final and important phase. The most critical stage in achieving a new successful system and giving users confidences that the new system will work effectively. The system can be implemented only after through testing is done and if it found to be working according to the specification. This method also offers the greatest security since the old system can takes if the error is found or inability to handle certain type of transaction while using the new system.

**SYSTEM MAINTENANCE**

##### 7.SYSTEM MAINTENANCE

Maintenance covers a wide range of activities, including correcting, coding and design errors, updating documentation and test data, and upgrading user support. Any activities classified as maintenance are actually enhancements. Maintenance means restoring something to its original condition. We may define software maintenance by describing for activities that are under taken after a program is released for use.

 CORRCTIVE MAINTENANCE

The first maintenance activity occurs since it is unreasonable to assume that software testing will uncover all error in a large software system. The process of including the diagnosis and correction of one or more error is called corrective maintenance.

 ADAPTIVE MAINTENANCE

This activity that contributes to the definition of maintenance occurs since rapid change in encountered in every aspect of computing. Therefore, adaptive maintenance modifies software to properly interface with the challenging environment.

 PERCEPTIVE MAINTENANCE

This activity involves recommendation for new capability modifications to the existing functions and general enhancement when software is used. To satisfy these requests perceptive maintenance is performed.

 PREVENTIVE MAINTENANCE

This activity occurs when software is changed to improve further maintainability and reliability. If the only available element of the software configuration is the source code, maintenance begin with the evaluation of the software code, often complicated by poor internal documentation. The subtle characteristics such as program structure, global data structure, system interface and performance and design constraints are difficult to handle and are often misinterpreted.

# 

# CONCLUSION

##### 

##### 8.CONCLUSION

The Application was designed in such a way that future changes can be done easily. The following conclusions can be deduced from the development of the project.

* Automation of the entire system improves the productivity.
* It provides a friendly graphical user interface which proves to be better when compared to the existing system.
* It gives appropriate access to the authorized users depending on their permissions.
* It effectively overcomes the delay in communications.
* Updating of information becomes so easier.
* System security, data security and reliability are the striking features.
* The System has adequate scope for modification in future if it is necessary.

**FUTURE SCOPE**

##### 9.FUTURE SCOPE

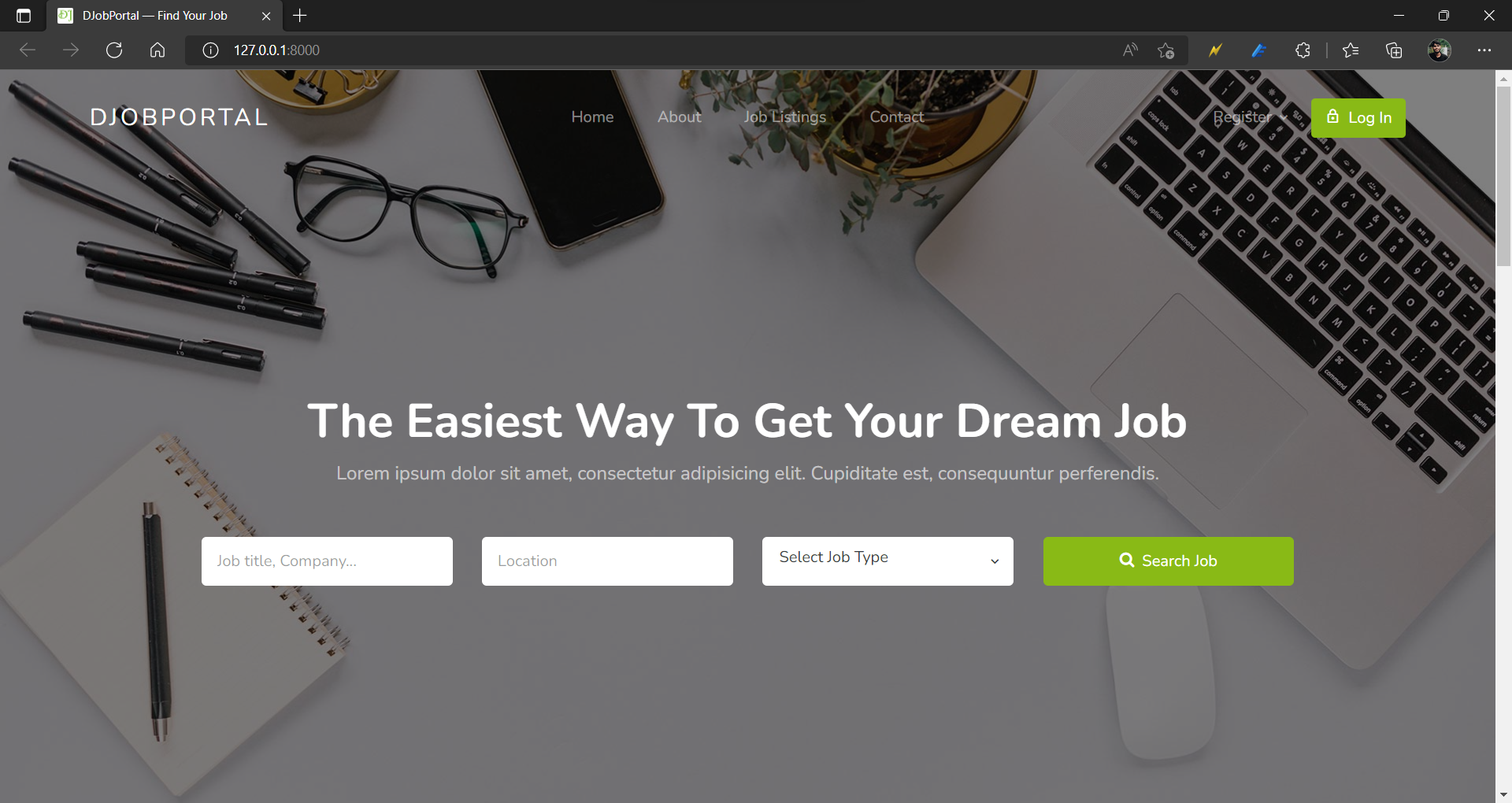
It may help collecting perfect management in details. In a very short time, the collection will be obvious, simple and sensible. It will help a person to know the management of passed years perfectly and vividly. It also helps in current all works relative to job portal system. It will be also reduced the cost of collecting the management & collection procedure will go on smoothly.

Our project aims is to automate the existing manual system by the help of computerized equipments and full-fledge computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. The required software and hardware are easily available and easy to work with.

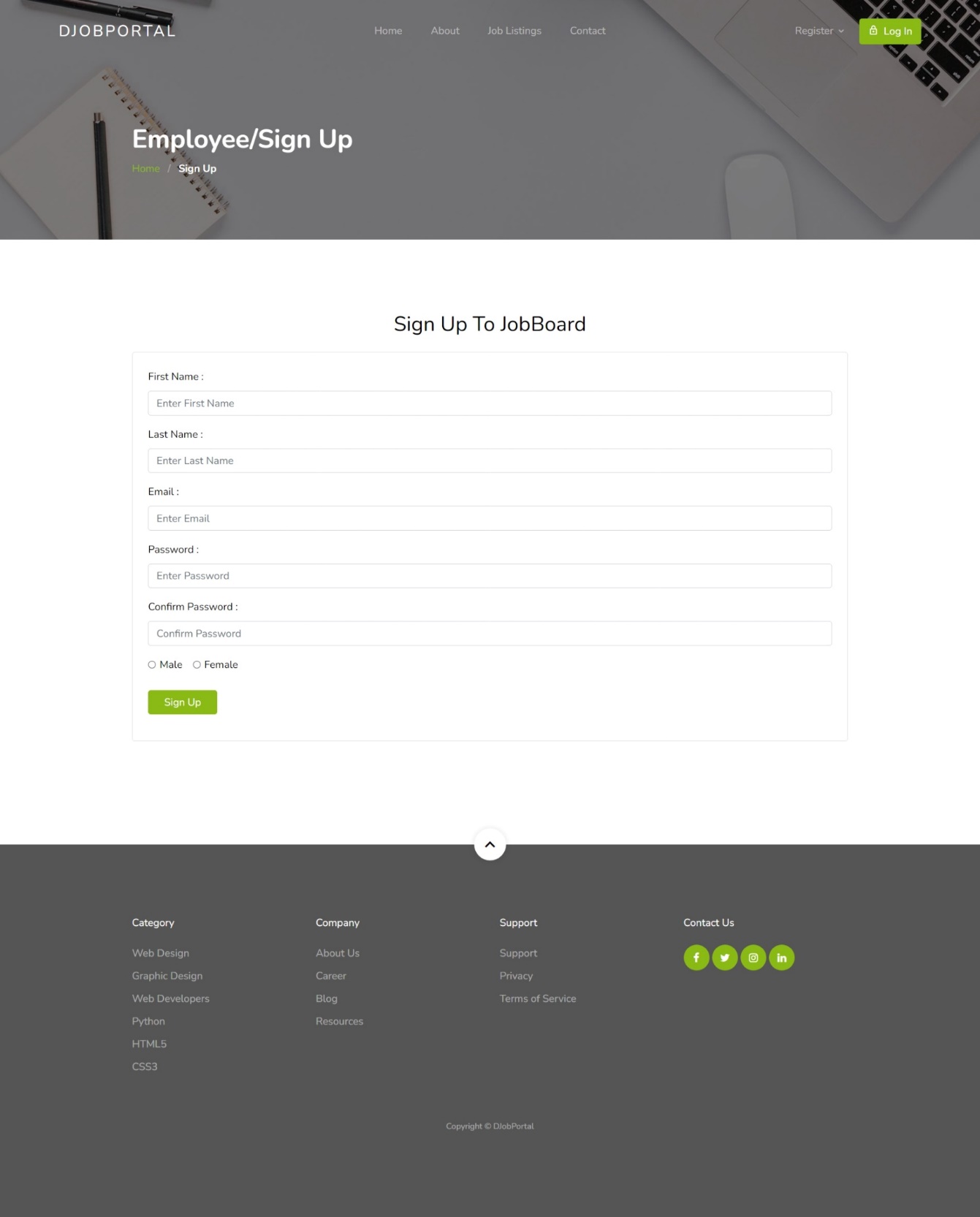
**APPENDIX**

**SCREENSHOT**

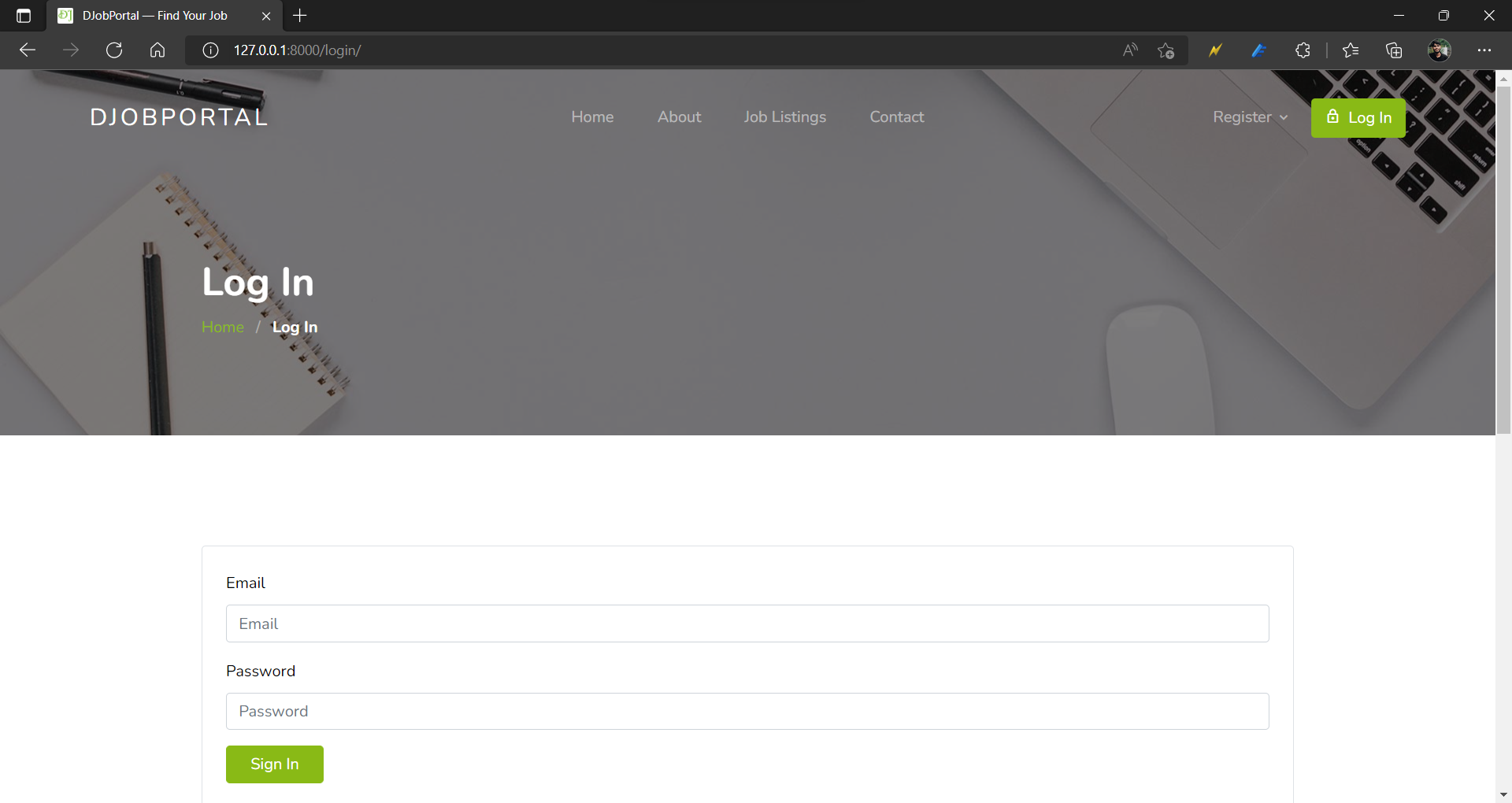
HOME PAGE



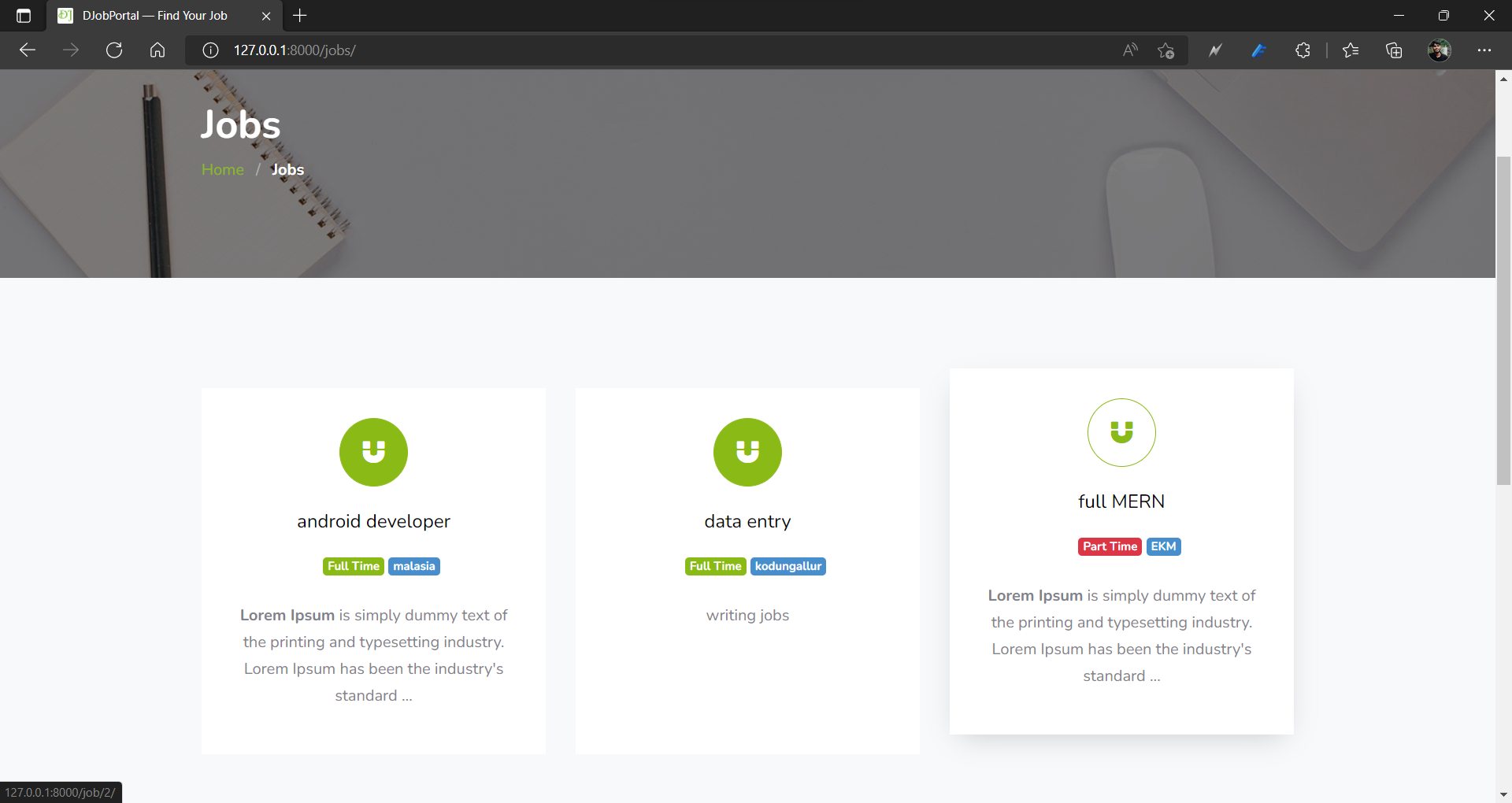
REGISTERATION PAGE OF EMPLOYEE



LOGIN PAGE



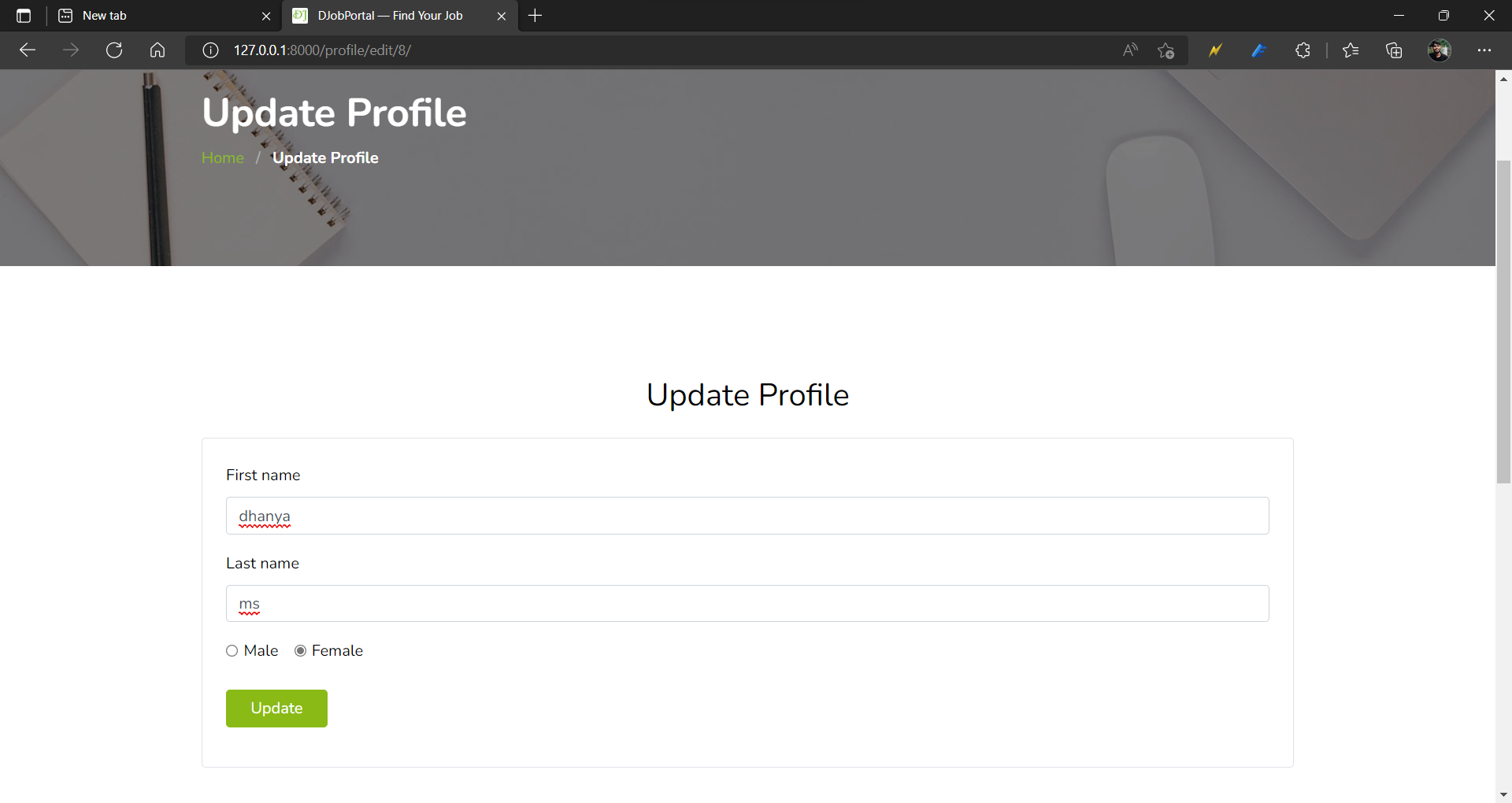
Job list

****

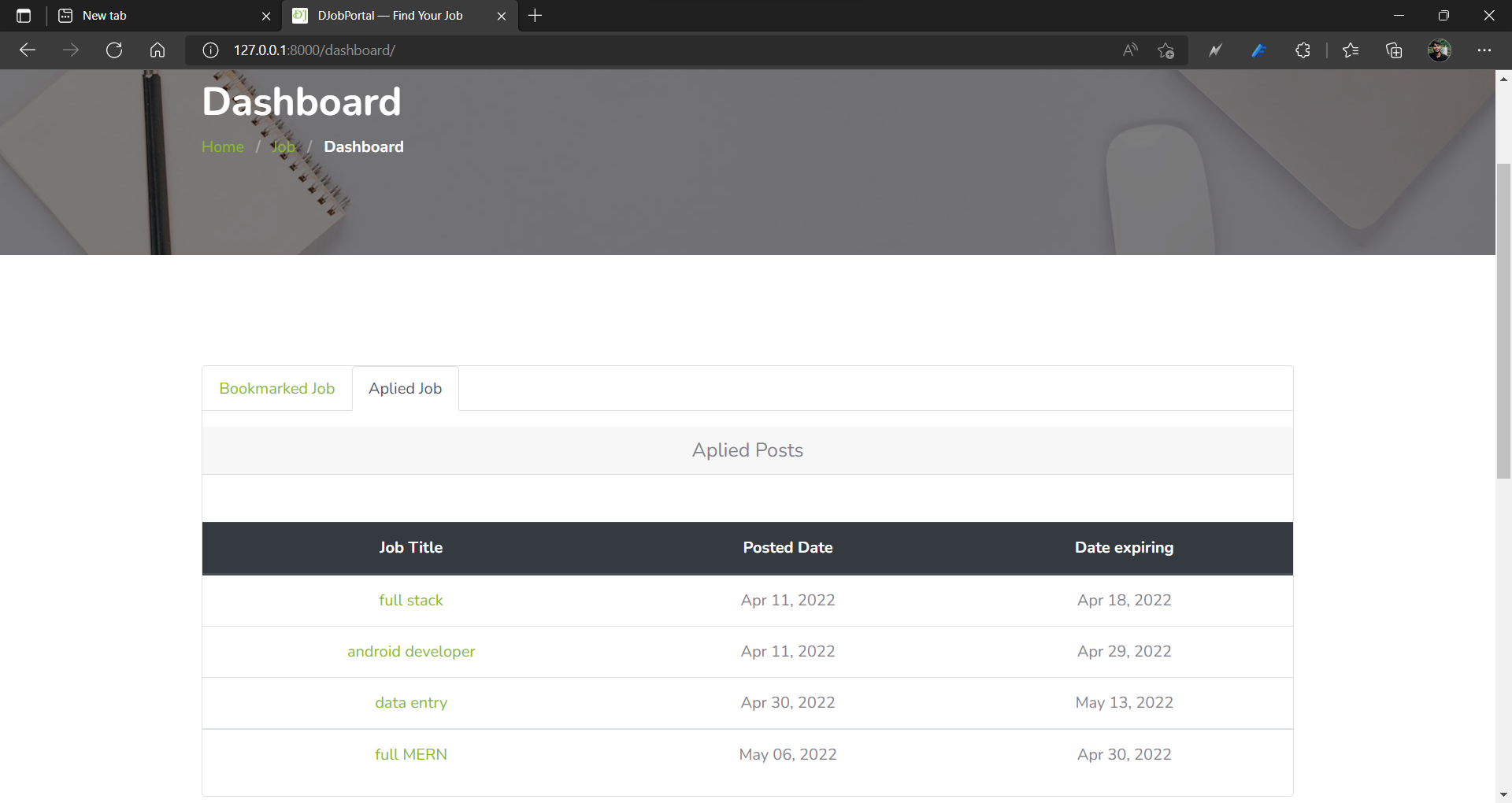
Job details



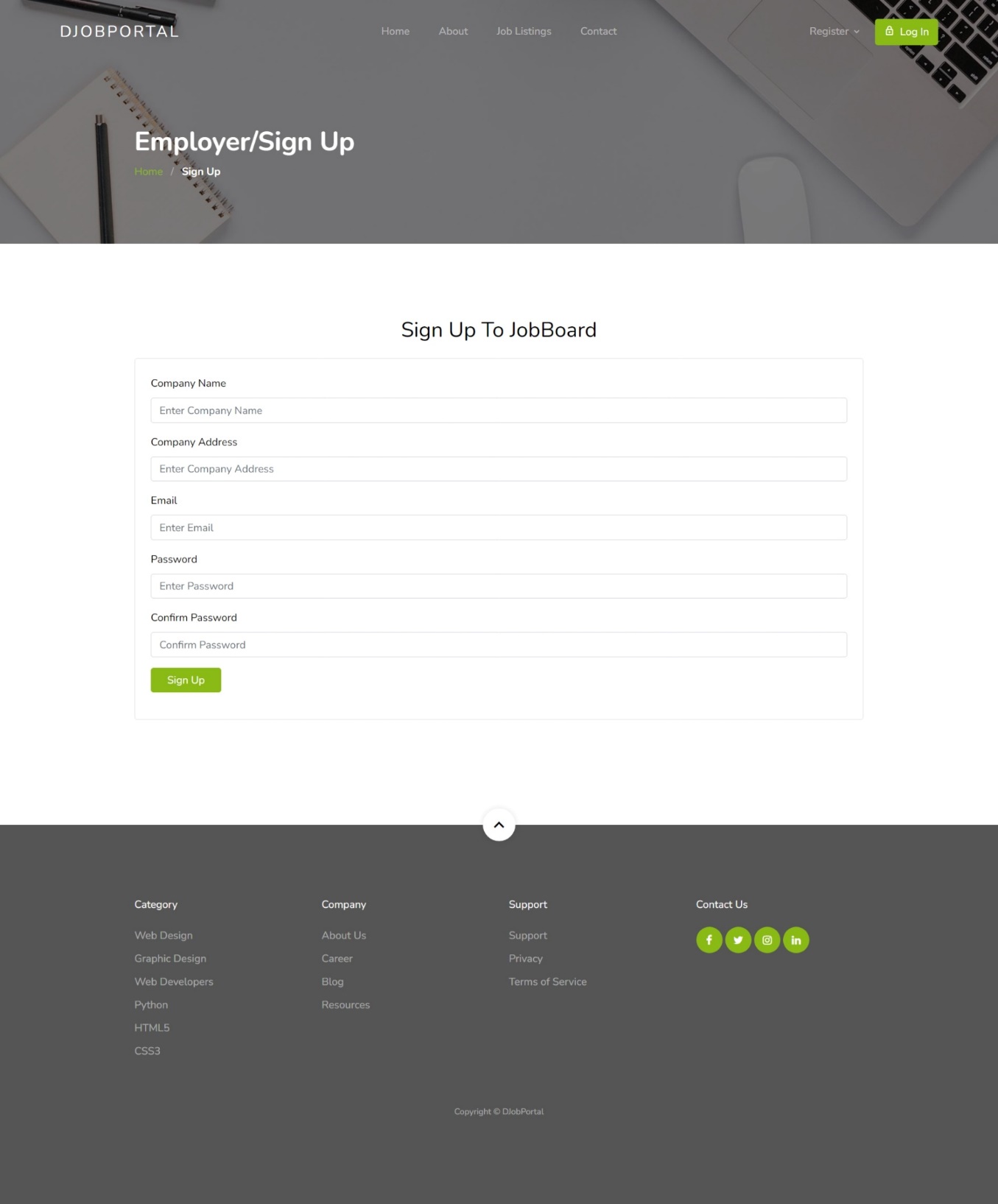
Update profile



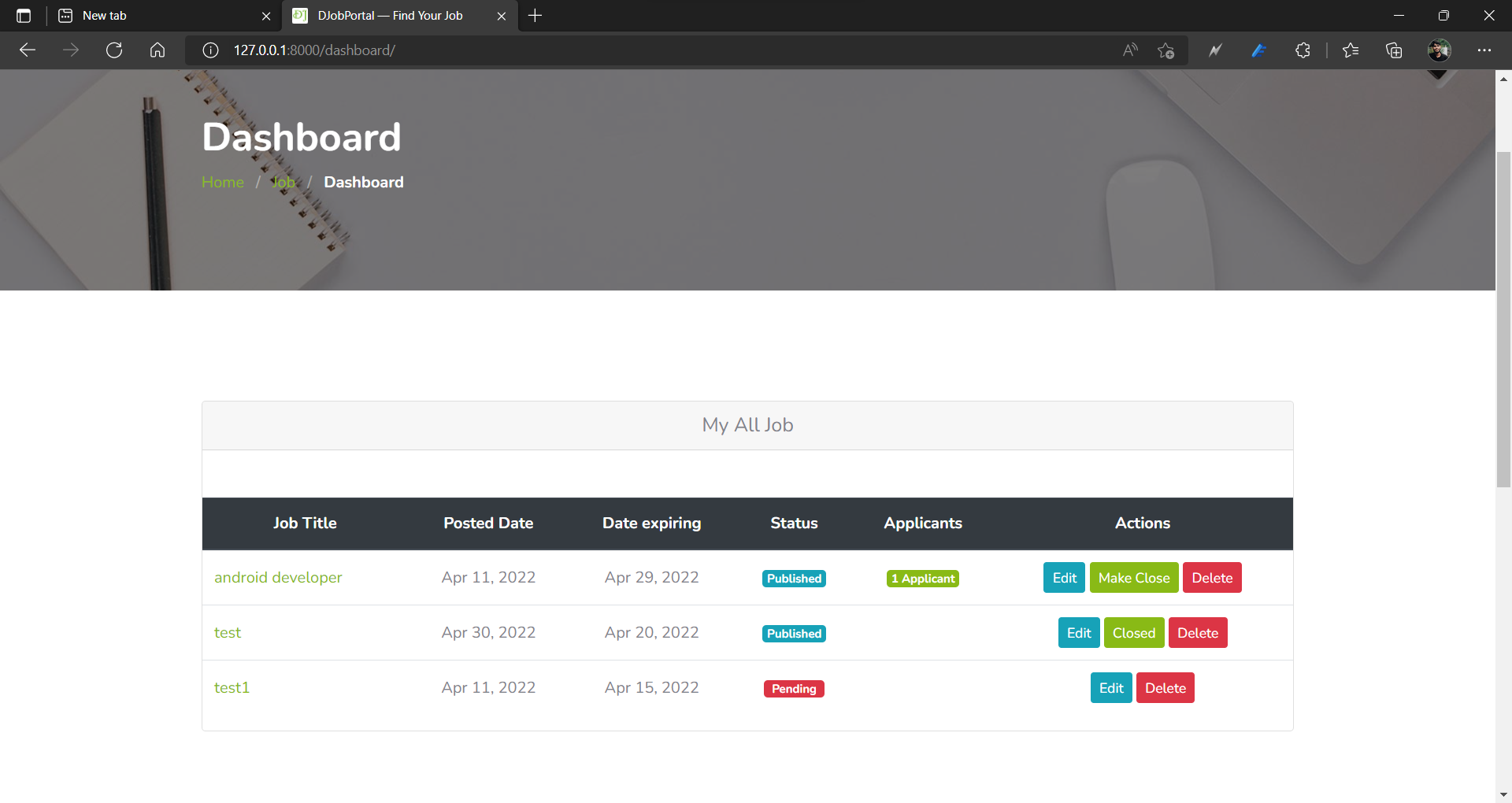
Dashboard



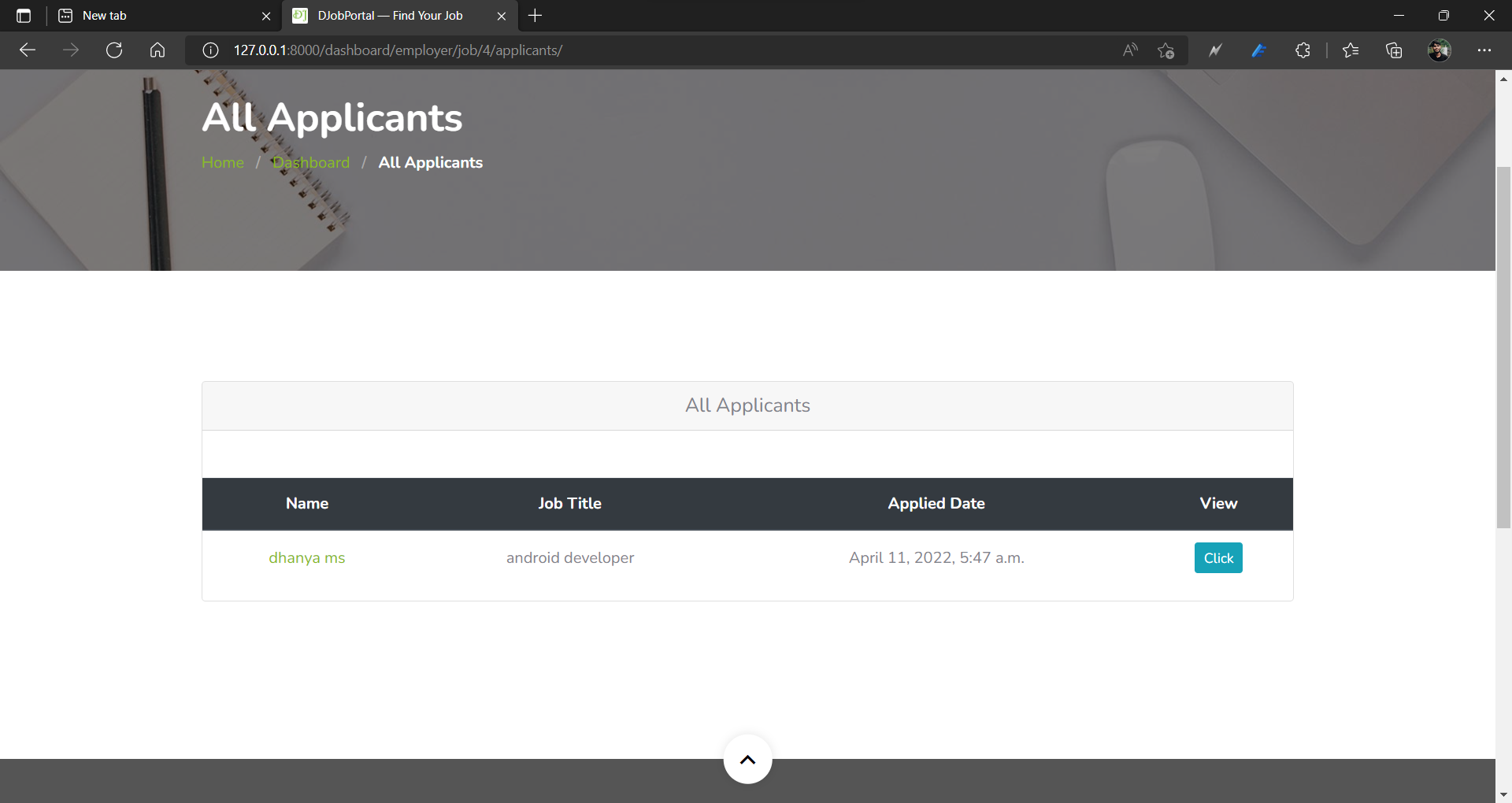
REGISTERATION PAGE ON EMPLOYER

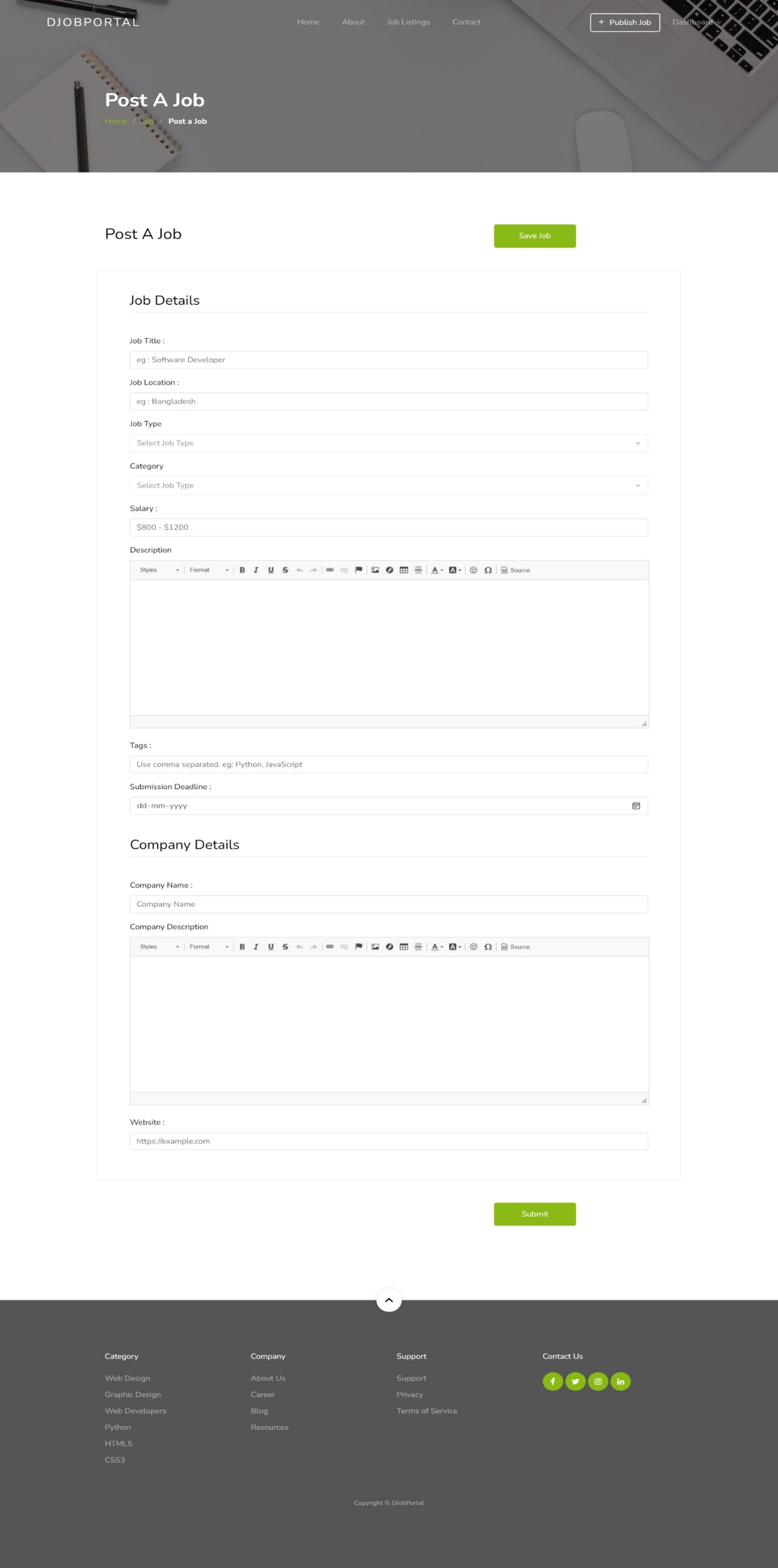


Dashboard

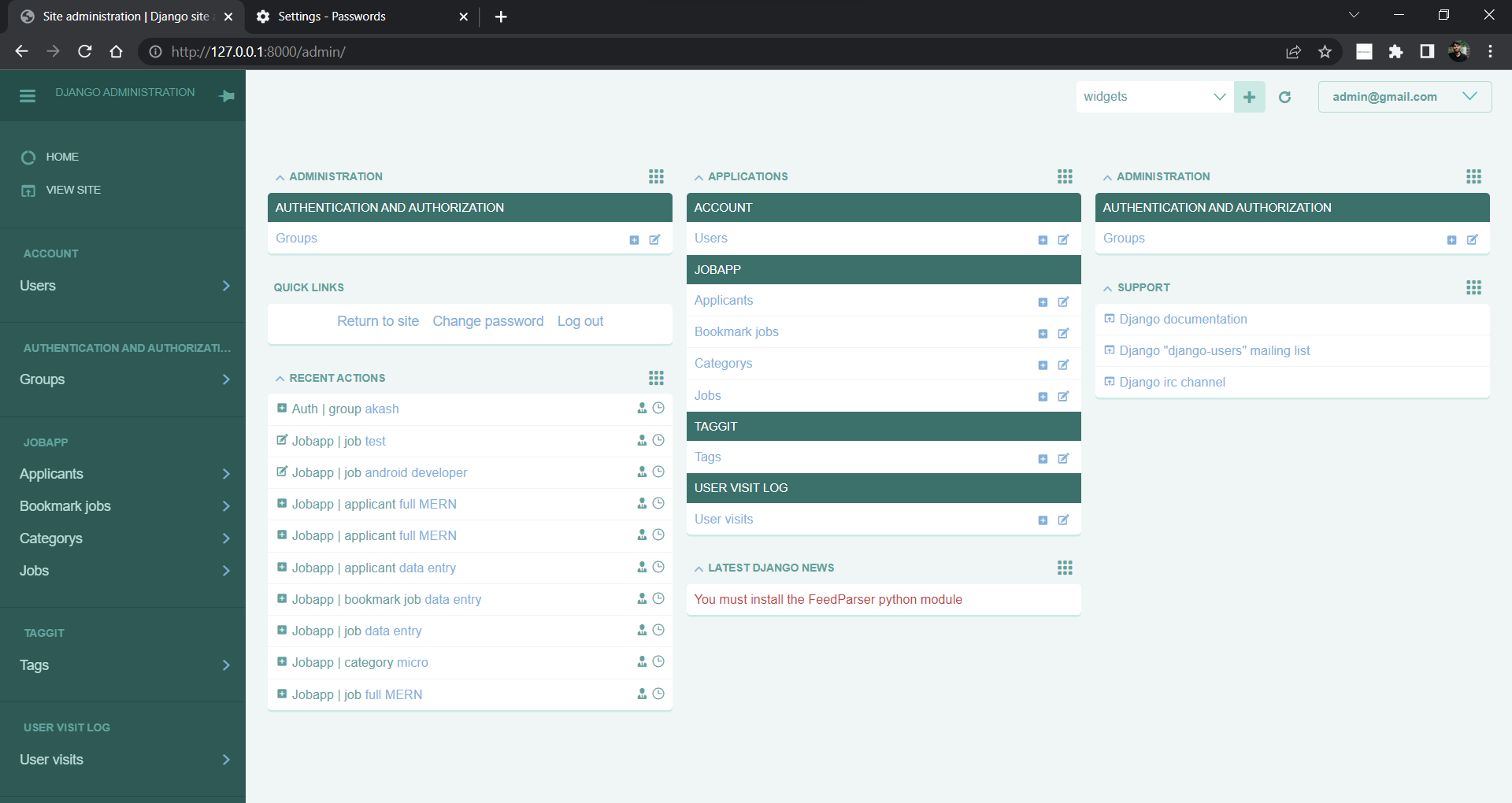


View applicants

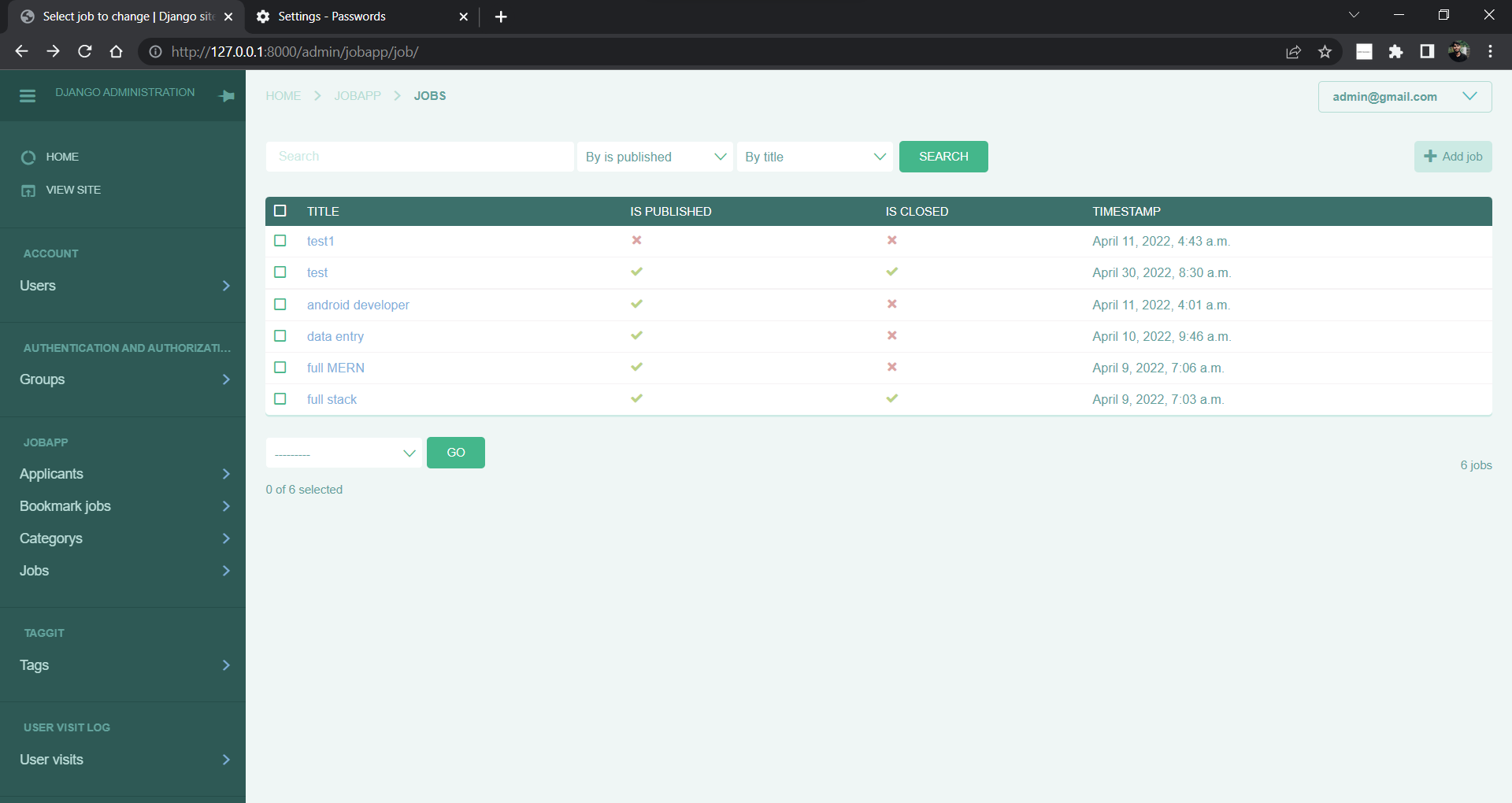


post job

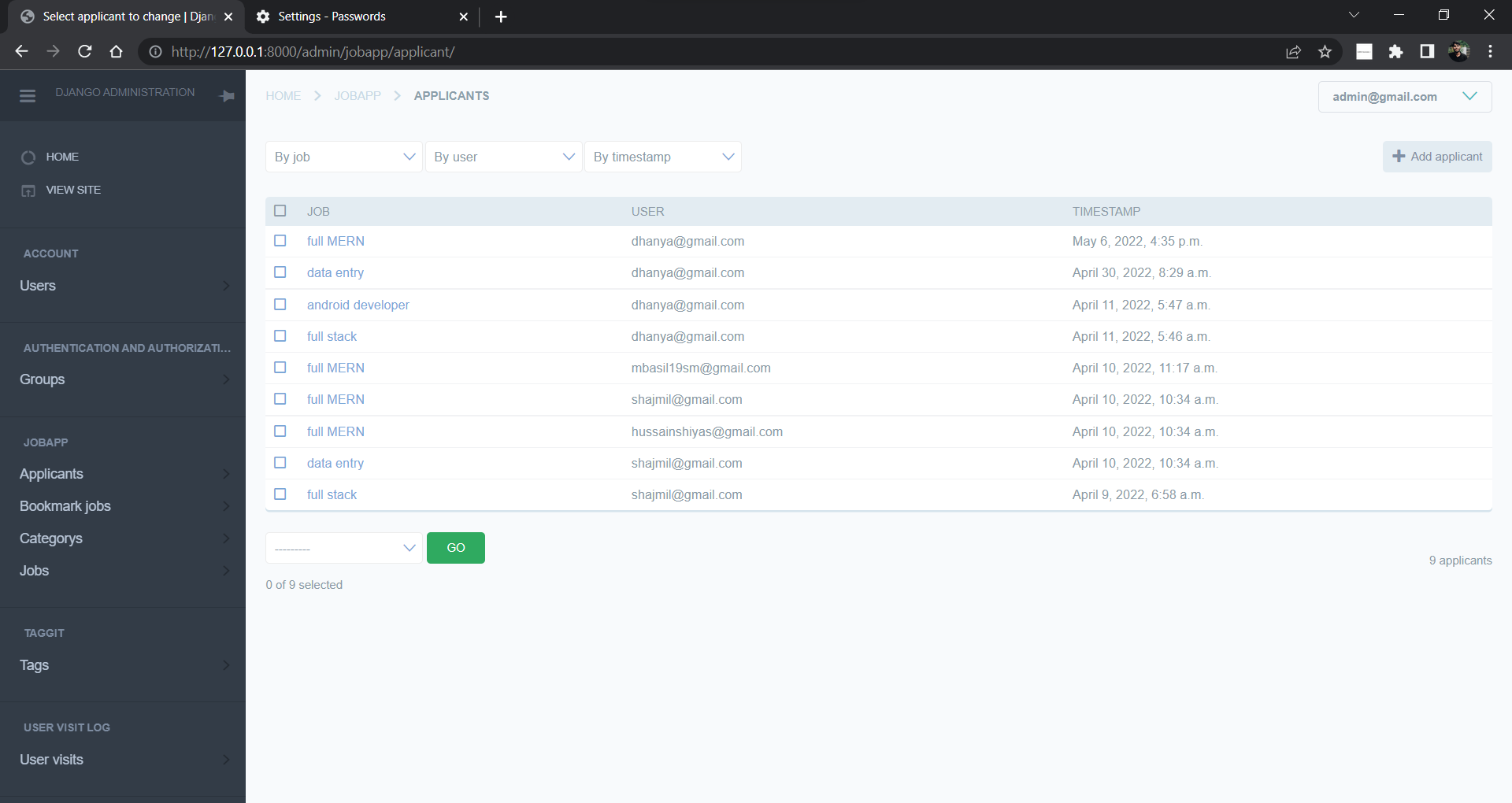
Admin Dashboard



View job



View all applicant



**SOURCE CODE**

from django.urls import path

from account import views

app\_name = "account"

urlpatterns = [

    path('employee/register/', views.employee\_registration, name='employee-registration'),

    path('employer/register/', views.employer\_registration, name='employer-registration'),

    path('profile/edit/<int:id>/', views.employee\_edit\_profile, name='edit-profile'),

    path('login/', views.user\_logIn, name='login'),

    path('logout/', views.user\_logOut, name='logout'),

]

from django.contrib.auth.base\_user import BaseUserManager

from django.utils.translation import ugettext\_lazy as \_

class CustomUserManager(BaseUserManager):

    use\_in\_migrations = True

    """

    Custom user model manager where email is the unique identifiers

    for authentication instead of usernames.

    """

    def create\_user(self, email, password, \*\*extra\_fields):

        """

        Create and save a User with the given email and password.

        """

        if not email:

            raise ValueError('The Email must be set')

        if not password:

            raise ValueError('The Password must be set')

        email = self.normalize\_email(email)

        user = self.model(email=email, \*\*extra\_fields)

        user.set\_password(password)

        user.save()

        return user

    def create\_superuser(self, email, password, \*\*extra\_fields):

        """

        Create and save a SuperUser with the given email and password.

        """

        extra\_fields.setdefault('is\_staff', True)

        extra\_fields.setdefault('is\_superuser', True)

        extra\_fields.setdefault('is\_active', True)

        if extra\_fields.get('is\_staff') is not True:

            raise ValueError('Superuser must have is\_staff=True.')

        if extra\_fields.get('is\_superuser') is not True:

            raise ValueError('Superuser must have is\_superuser=True.')

        return self.create\_user(email, password, \*\*extra\_fields)

from django.contrib import auth

from django.contrib import messages

from django.contrib.auth.decorators import login\_required

from django.http import HttpResponseRedirect

from django.shortcuts import render, redirect , get\_object\_or\_404

from django.urls import reverse, reverse\_lazy

from account.forms import \*

from jobapp.permission import user\_is\_employee

def get\_success\_url(request):

    """

    Handle Success Url After LogIN

    """

    if 'next' in request.GET and request.GET['next'] != '':

        return request.GET['next']

    else:

        return reverse('jobapp:home')

def employee\_registration(request):

    """

    Handle Employee Registration

    """

    form = EmployeeRegistrationForm(request.POST or None)

    if form.is\_valid():

        form = form.save()

        return redirect('account:login')

    context={

            'form':form

        }

    return render(request,'account/employee-registration.html',context)

def employer\_registration(request):

    """

    Handle Employee Registration

    """

    form = EmployerRegistrationForm(request.POST or None)

    if form.is\_valid():

        form = form.save()

        return redirect('account:login')

    context={

            'form':form

        }

    return render(request,'account/employer-registration.html',context)

@login\_required(login\_url=reverse\_lazy('accounts:login'))

@user\_is\_employee

def employee\_edit\_profile(request, id=id):

    """

    Handle Employee Profile Update Functionality

    """

    user = get\_object\_or\_404(User, id=id)

    form = EmployeeProfileEditForm(request.POST or None, instance=user)

    if form.is\_valid():

        form = form.save()

        messages.success(request, 'Your Profile Was Successfully Updated!')

        return redirect(reverse("account:edit-profile", kwargs={

                                    'id': form.id

                                    }))

    context={

            'form':form

        }

    return render(request,'account/employee-edit-profile.html',context)

def user\_logIn(request):

    """

    Provides users to logIn

    """

    form = UserLoginForm(request.POST or None)

    if request.user.is\_authenticated:

        return redirect('/')

    else:

        if request.method == 'POST':

            if form.is\_valid():

                auth.login(request, form.get\_user())

                return HttpResponseRedirect(get\_success\_url(request))

    context = {

        'form': form,

    }

    return render(request,'account/login.html',context)

def user\_logOut(request):

    """

    Provide the ability to logout

    """

    auth.logout(request)

    messages.success(request, 'You are Successfully logged out')

    return redirect('account:login')

from django.contrib import admin

from .models import \*

admin.site.register(Category)

class ApplicantAdmin(admin.ModelAdmin):

    list\_display = ('job','user','timestamp')

    list\_filter = ('job', 'user','timestamp')

admin.site.register(Applicant,ApplicantAdmin)

class JobAdmin(admin.ModelAdmin):

    list\_display = ('title','is\_published','is\_closed','timestamp')

    list\_filter = ('title','is\_published')

    search\_fields = ('title', 'timestamp')

admin.site.register(Job,JobAdmin)

class BookmarkJobAdmin(admin.ModelAdmin):

    list\_display = ('job','user','timestamp')

    list\_filter = ['job','user']

admin.site.register(BookmarkJob,BookmarkJobAdmin)

from django.core.exceptions import PermissionDenied

def user\_is\_employer(function):

    def wrap(request, \*args, \*\*kwargs):

        if request.user.role == 'employer':

            return function(request, \*args, \*\*kwargs)

        else:

            raise PermissionDenied

    return wrap

def user\_is\_employee(function):

    def wrap(request, \*args, \*\*kwargs):

        if request.user.role == 'employee':

            return function(request, \*args, \*\*kwargs)

        else:

            raise PermissionDenied

    return wrap

from django.contrib import messages

from django.contrib.auth import get\_user\_model

from django.contrib.auth.decorators import login\_required

from django.core.paginator import Paginator

from django.db.models import Q

from django.shortcuts import render, get\_object\_or\_404, redirect

from django.urls import reverse, reverse\_lazy

from django.http import Http404, HttpResponseRedirect, JsonResponse

from django.core.serializers import serialize

from account.models import User

from jobapp.forms import \*

from jobapp.models import \*

from jobapp.permission import \*

User = get\_user\_model()

def home\_view(request):

    published\_jobs = Job.objects.filter(is\_published=True).order\_by('-timestamp')

    jobs = published\_jobs.filter(is\_closed=False)

    total\_candidates = User.objects.filter(role='employee').count()

    total\_companies = User.objects.filter(role='employer').count()

    paginator = Paginator(jobs, 3)

    page\_number = request.GET.get('page',None)

    page\_obj = paginator.get\_page(page\_number)

    if request.is\_ajax():

        job\_lists=[]

        job\_objects\_list = page\_obj.object\_list.values()

        for job\_list in job\_objects\_list:

            job\_lists.append(job\_list)

        next\_page\_number = None

        if page\_obj.has\_next():

            next\_page\_number = page\_obj.next\_page\_number()

        prev\_page\_number = None

        if page\_obj.has\_previous():

            prev\_page\_number = page\_obj.previous\_page\_number()

        data={

            'job\_lists':job\_lists,

            'current\_page\_no':page\_obj.number,

            'next\_page\_number':next\_page\_number,

            'no\_of\_page':paginator.num\_pages,

            'prev\_page\_number':prev\_page\_number

        }

        return JsonResponse(data)

    context = {

    'total\_candidates': total\_candidates,

    'total\_companies': total\_companies,

    'total\_jobs': len(jobs),

    'total\_completed\_jobs':len(published\_jobs.filter(is\_closed=True)),

    'page\_obj': page\_obj

    }

    print('ok')

    return render(request, 'jobapp/index.html', context)

def job\_list\_View(request):

    """

    """

    job\_list = Job.objects.filter(is\_published=True,is\_closed=False).order\_by('-timestamp')

    paginator = Paginator(job\_list, 12)

    page\_number = request.GET.get('page')

    page\_obj = paginator.get\_page(page\_number)

    context = {

        'page\_obj': page\_obj,

    }

    return render(request, 'jobapp/job-list.html', context)

@login\_required(login\_url=reverse\_lazy('account:login'))

@user\_is\_employer

def create\_job\_View(request):

    """

    Provide the ability to create job post

    """

    form = JobForm(request.POST or None)

    user = get\_object\_or\_404(User, id=request.user.id)

    categories = Category.objects.all()

    if request.method == 'POST':

        if form.is\_valid():

            instance = form.save(commit=False)

            instance.user = user

            instance.save()

            # for save tags

            form.save\_m2m()

            messages.success(

                    request, 'You are successfully posted your job! Please wait for review.')

            return redirect(reverse("jobapp:single-job", kwargs={

                                    'id': instance.id

                                    }))

    context = {

        'form': form,

        'categories': categories

    }

    return render(request, 'jobapp/post-job.html', context)

def single\_job\_view(request, id):

    """

    Provide the ability to view job details

    """

    job = get\_object\_or\_404(Job, id=id)

    related\_job\_list = job.tags.similar\_objects()

    paginator = Paginator(related\_job\_list, 5)

    page\_number = request.GET.get('page')

    page\_obj = paginator.get\_page(page\_number)

    context = {

        'job': job,

        'page\_obj': page\_obj,

        'total': len(related\_job\_list)

    }

    return render(request, 'jobapp/job-single.html', context)

def search\_result\_view(request):

    """

        User can search job with multiple fields

    """

    job\_list = Job.objects.order\_by('-timestamp')

    # Keywords

    if 'job\_title\_or\_company\_name' in request.GET:

        job\_title\_or\_company\_name = request.GET['job\_title\_or\_company\_name']

        if job\_title\_or\_company\_name:

            job\_list = job\_list.filter(title\_\_icontains=job\_title\_or\_company\_name) | job\_list.filter(

                company\_name\_\_icontains=job\_title\_or\_company\_name)

    # location

    if 'location' in request.GET:

        location = request.GET['location']

        if location:

            job\_list = job\_list.filter(location\_\_icontains=location)

    # Job Type

    if 'job\_type' in request.GET:

        job\_type = request.GET['job\_type']

        if job\_type:

            job\_list = job\_list.filter(job\_type\_\_iexact=job\_type)

    # job\_title\_or\_company\_name = request.GET.get('text')

    # location = request.GET.get('location')

    # job\_type = request.GET.get('type')

    #     job\_list = Job.objects.all()

    #     job\_list = job\_list.filter(

    #         Q(job\_type\_\_iexact=job\_type) |

    #         Q(title\_\_icontains=job\_title\_or\_company\_name) |

    #         Q(location\_\_icontains=location)

    #     ).distinct()

    # job\_list = Job.objects.filter(job\_type\_\_iexact=job\_type) | Job.objects.filter(

    #     location\_\_icontains=location) | Job.objects.filter(title\_\_icontains=text) | Job.objects.filter(company\_name\_\_icontains=text)

    paginator = Paginator(job\_list, 10)

    page\_number = request.GET.get('page')

    page\_obj = paginator.get\_page(page\_number)

    context = {

        'page\_obj': page\_obj,

    }

    return render(request, 'jobapp/result.html', context)

@login\_required(login\_url=reverse\_lazy('account:login'))

@user\_is\_employee

def apply\_job\_view(request, id):

    form = JobApplyForm(request.POST or None)

    user = get\_object\_or\_404(User, id=request.user.id)

    applicant = Applicant.objects.filter(user=user, job=id)

    if not applicant:

        if request.method == 'POST':

            if form.is\_valid():

                instance = form.save(commit=False)

                instance.user = user

                instance.save()

                messages.success(

                    request, 'You have successfully applied for this job!')

                return redirect(reverse("jobapp:single-job", kwargs={

                    'id': id

                }))

        else:

            return redirect(reverse("jobapp:single-job", kwargs={

                'id': id

            }))

    else:

        messages.error(request, 'You already applied for the Job!')

        return redirect(reverse("jobapp:single-job", kwargs={

            'id': id

        }))

@login\_required(login\_url=reverse\_lazy('account:login'))

def dashboard\_view(request):

    """

    """

    jobs = []

    savedjobs = []

    appliedjobs = []

    total\_applicants = {}

    if request.user.role == 'employer':

        jobs = Job.objects.filter(user=request.user.id)

        for job in jobs:

            count = Applicant.objects.filter(job=job.id).count()

            total\_applicants[job.id] = count

    if request.user.role == 'employee':

        savedjobs = BookmarkJob.objects.filter(user=request.user.id)

        appliedjobs = Applicant.objects.filter(user=request.user.id)

    context = {

        'jobs': jobs,

        'savedjobs': savedjobs,

        'appliedjobs':appliedjobs,

        'total\_applicants': total\_applicants

    }

    return render(request, 'jobapp/dashboard.html', context)

@login\_required(login\_url=reverse\_lazy('account:login'))

@user\_is\_employer

def delete\_job\_view(request, id):

    job = get\_object\_or\_404(Job, id=id, user=request.user.id)

    if job:

        job.delete()

        messages.success(request, 'Your Job Post was successfully deleted!')

    return redirect('jobapp:dashboard')

@login\_required(login\_url=reverse\_lazy('account:login'))

@user\_is\_employer

def make\_complete\_job\_view(request, id):

    job = get\_object\_or\_404(Job, id=id, user=request.user.id)

    if job:

        try:

            job.is\_closed = True

            job.save()

            messages.success(request, 'Your Job was marked closed!')

        except:

            messages.success(request, 'Something went wrong !')

    return redirect('jobapp:dashboard')

@login\_required(login\_url=reverse\_lazy('account:login'))

@user\_is\_employer

def all\_applicants\_view(request, id):

    all\_applicants = Applicant.objects.filter(job=id)

    context = {

        'all\_applicants': all\_applicants

    }

    return render(request, 'jobapp/all-applicants.html', context)

@login\_required(login\_url=reverse\_lazy('account:login'))

@user\_is\_employee

def delete\_bookmark\_view(request, id):

    job = get\_object\_or\_404(BookmarkJob, id=id, user=request.user.id)

    if job:

        job.delete()

        messages.success(request, 'Saved Job was successfully deleted!')

    return redirect('jobapp:dashboard')

@login\_required(login\_url=reverse\_lazy('account:login'))

@user\_is\_employer

def applicant\_details\_view(request, id):

    applicant = get\_object\_or\_404(User, id=id)

    context = {

        'applicant': applicant

    }

    return render(request, 'jobapp/applicant-details.html', context)

@login\_required(login\_url=reverse\_lazy('account:login'))

@user\_is\_employee

def job\_bookmark\_view(request, id):

    form = JobBookmarkForm(request.POST or None)

    user = get\_object\_or\_404(User, id=request.user.id)

    applicant = BookmarkJob.objects.filter(user=request.user.id, job=id)

    if not applicant:

        if request.method == 'POST':

            if form.is\_valid():

                instance = form.save(commit=False)

                instance.user = user

                instance.save()

                messages.success(

                    request, 'You have successfully save this job!')

                return redirect(reverse("jobapp:single-job", kwargs={

                    'id': id

                }))

        else:

            return redirect(reverse("jobapp:single-job", kwargs={

                'id': id

            }))

    else:

        messages.error(request, 'You already saved this Job!')

        return redirect(reverse("jobapp:single-job", kwargs={

            'id': id

        }))

@login\_required(login\_url=reverse\_lazy('account:login'))

@user\_is\_employer

def job\_edit\_view(request, id=id):

    """

    Handle Job Update

    """

    job = get\_object\_or\_404(Job, id=id, user=request.user.id)

    categories = Category.objects.all()

    form = JobEditForm(request.POST or None, instance=job)

    if form.is\_valid():

        instance = form.save(commit=False)

        instance.save()

        # for save tags

        # form.save\_m2m()

        messages.success(request, 'Your Job Post Was Successfully Updated!')

        return redirect(reverse("jobapp:single-job", kwargs={

            'id': instance.id

        }))

    context = {

        'form': form,

        'categories': categories

    }

    return render(request, 'jobapp/job-edit.html', context)

**BIBLIOGRAPHY**

**11. BIBLIOGRAPHY**

**Text books**

Following text books are also referred for building this project.

* + - Fundamentals of Database System - Ramez Elmasri and Sham Kand p. Nayadha.
    - Web Programming Builiding Internet Applications – Chris Bates.

Websites

Following websites are referred to create this project reports.

* + - [http://www.w3schools.com](http://www.w3schools.com/)
    - [http://en.m.wikipedia.org](http://en.m.wikipedia.org/)
    - [http://www.javascripttutorials.com](http://www.javascripttutorials.com/)