

Government College of Engineering (GCOEJ), Jalgaon
(An Autonomous Institute of Government of Maharashtra)



DEPARTMENT OF COMPUTER ENGINEERING

MINI PROJECT REPORT ON
JOB SEARCH PORTAL
(Academic Year 2022-23)

A ROBUST AND EASY TO HANDLE JOB SEARCH PORTAL

Submitted by:

- Bhangale Diksha Shalik (2041007)
- Bhargav Shamuvel Gurav (2041009)
- Chanore Bhushan Raju (2041016)
- Ghodke Ajay Satish (2041025)

MINI PROJECT GUIDE

Mr. S. D. Cheke

Government College of Engineering (GCOEJ), Jalgaon
(An Autonomous Institute of Govt. of Maharashtra)



DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the *Mini Project* report, “**Job Search Portal**”, which is being submitted here with for the award of *TY Computer Engineering (6th Semester)* is the result of the work completed by *Bhangale Diksha (2041007)*, *Bhargav Gurav (2041009)*, *Chanore Bhushan (2041016)*, *Ghodke Ajay (2041025)* under my supervision and guidance within offline mode of classes of the institute, in the academic year 2022-23

Mini Project Guide
Mr. S. D. Cheke

Head of Department
Mr. D. V. Chaudhari

DECLARATION

We hereby declare that the Mini Project entitled “**Job Search Portal**” was carried out and written by us under the guidance of **Mr. S. D. Cheke and Head of Department**, department of Computer Engineering, Government College of Engineering, Jalgaon. This work has not been previously formed for the award of any degree or diploma or certificate nor has been submitted elsewhere for the award of any degree.

Place: Jalgaon

Date:

Bhangale Diksha Shalik	2041007
Bhargav Shamuvel Gurav	2041009
Chanore Bhushan Raju	2041016
Ghodke Ajay Satish	2041025

INDEX

CONTENTS

	Page no.
TITLE	
COVER PAGE	1
CERTIFICATE	2
DECLARATION	3
ABSTRACT	6
1.INTRODUCTION	7
1.1 PROJECT DESCRIPTION	7
1.2 MOTIVATION	7
1.3 PROBLEM STATEMENT	8
1.4 OBJECTIVE	8
1.5 PROJECT MODULES	9
1.6 PROJECT REQUIREMENTS	9
1.7 ORGANIZATION OF PROJECT	10
2.LITERATURE SURVEY	11
2.1 INTRODUCTION	11
2.1.1 DJANGO	11
3.DESIGN AND IMPLEMENTATION	25
3.1 DATAFLOW DIAGRAMS	25
3.2 UML	27
3.2.1 USECASE DIAGRAM	27
3.2.2 CLASS DIAGRAM	29
3.2.3 SEQUENCE DIAGRAM	30
3.2.4 STATECHART DIAGRAM	32
4. RESULTS AND DISCUSSION	34

4.1 JOB PORTAL LAYOUT SCREENSHOTS	34
5. CONCLUSION	44
6. FUTURE SCOPE	45
7. REFERENCES	46

ABSTRACT

Finding jobs that best suits the interests and skill set is quite a challenging task for the job seekers. The difficulties arise from not having proper knowledge on the organization's objective, their work culture and current job openings. In addition, finding the right candidate with desired qualifications to fill their current job openings is an important task for the recruiters of any organization. Online Job Search Portals have certainly made job seeking convenient on both sides. Job Portal is the solution where recruiter as well as the job seeker meet aiming at fulfilling their individual requirement. They are the cheapest as well as the fastest source of communication reaching wide range of audience on just a single click irrespective of their geographical distance. The web application "Job Search Portal" provides an easy and convenient search application for the job seekers to find their desired jobs and for the recruiters to find the right candidate. Job seekers from any background can search for the current job openings. Job seekers can register with the application and update their details and skill set. They can search for available jobs and apply to their desired positions. Android, being open source has already made its mark in the mobile application development. To make things handy, the user functionalities are developed as an Android application. Employer can register with the application and posts their current opening.

CHAPTER 1

INTRODUCTION

1.1 PROJECT DESCRIPTION

Job Search Portal is a web application, which serves jobseekers to find available job vacancies and employers to identify eligible job seekers with the prospect of selecting the most qualified candidates. The only way to select best-qualified candidate is to have a pool of eligible applicants, which is possible by drawing the interest of individuals in the market. Job search portals best serve this purpose. E-recruitment has become the standard means for employers and job seekers to meet their respective objectives. The traditional methods for recruitment include Job fairs, University career employment services, Employee referrals, advertising in the newspapers, televisions etc. With the advancement in technology and growth of internet usage, the e-recruitment has revolutionized the way organizations hire and candidates search for jobs. With the Online Job search portals, the recruitment process is speeded up at every stage from job postings, to receiving applications from candidates, interviewing process. The cost of searching/posting jobs will be much less compared to the traditional way of advertising. Job search portal stands as an effective means for Employers to outline the job vacancies, responsibilities and qualifications to attract jobseekers. Using the portal jobseekers can extensively search for jobs in companies, organizations and regions they may otherwise have not learnt. In addition, candidates/Employers can write a review about an organization, which might help them to change the way things are done.

1.2 MOTIVATION

The purpose of developing an Online Job Search Portal comes from our idea to make the job search efficient and handy. It helps the recruiters as a primary source of talent search. It also helps the job seekers to search for current vacancies at a single point. Therefore, we can say that Online Job Search Portal act as a bridge of communication between organizations and applicants. With the evolution of technology and internet being the main

source of information for the applicants, these job portals and have become an excellent method to reach wide range of audience. Initially, when we are unaware of these portals, we used to do research about companies and their technology stack through their respective websites and apply if the job responsibilities match our interests. This requires lots of effort and time. However, later when we realized the importance of job search portals, we are able to access jobs in companies, locations that we might not otherwise have learned.

1.3 PROBLEM STATEMENT

The current problem regarding job seeking is that recruitment is done manually.

For most of the jobs, job seekers have to go to the agency and check the available jobs at agency. This requires lots of efforts and time.

This traditional method for recruiting is less convenient for both recruiter and job seeker.

1.4 OBJECTIVE

- The online job Portal System that is to be developed provides the members with jobs information, online applying for jobs and many other facilities.
- This system provides service to the job applicants to search for working opportunities.
- Job Portal will allow job provider to establish one to one relationships with candidates.
- This Portal will primarily focus on the posting and management of job vacancies.
- This system is designed such that ultimately all vacancies will be posted online and would offer employers the facilities to post their vacancies online.
- It helps to review and manage the resulting applications efficiently through the web.

- Employer can also find the resume according to key skill in very less amount of time.

1.5 PROJECT MODULES

1.5.1 Registration

Employee or Employer can register with valid details like contact details, experience details, profile details.

1.5.2 Search

Employee Can Search job according to their interest. And also apply for that job. Employer search candidates for their requirements using keyword. Employer also can communicate with employee for their any other query or information via send message.

1.5.3 Job Post

Employer post job for their organization. And include job vacancy, salary details, working hours, designation details, experienced details.

1.5.4 Manage Account

Employee can also delete his/her account anytime. Admin Can Manage Employee and Employer Details. Admin observed Users Action like job posting, candidate details false or not.

1.6 PROJECT REQUIREMENTS

1.6.1 Hardware

The system requires the following hardware:

- RAM: 1 GB (further increase that as per requirement.)
- Hard Disk: 80 GB (further increase that as per requirement.)
- Display: 1024 * 768, True Type Color-32 Bit
- Mouse: Any Normal Mouse.
- Keyboard: Any window Supported Keyboard.

1.6.2 Software

- Database Server: Microsoft SQL Server
- Web Server: Internet Information Server
- Technologies:

Frontend: HTML, CSS, JAVASCRIPT

Backend: DJANGO

1.7 ORGANIZATION OF PROJECT

This report is divided up into chapters, each dealing with different aspects of the project. Each chapter has a short introduction, explaining the subject of each chapter, and then the details, each module is explained separately. The following is a short overview of each of the chapters:

Chapter 2, Outlines some of the research made on the project in the beginning. More research was made as this project report was being developed, as new areas had to be investigated. This research is summarized in the various chapters according to the different modules.

Chapter 3, Specifies the Software approaches and methodologies used in the project.

Chapter 4, Specifies implementation design used in the project.

Chapter 5, defines the general architectural flow of the system.

Chapter 6, Shows the Results of the project.

Chapter 7, Conclusion and the future scope of the project is discussed here

CHAPTER 2

LITERATURE SURVEY

2.1 INTRODUCTION

The domain analysis that we have done for the project mainly involved understanding the web development framework

2.1.1 DJANGO

What is Django?

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Django helps you write software that is:

Complete

Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, follows consistent design principles, and has extensive and up-to-date documentation.

Versatile

Django can be (and has been) used to build almost any type of website — from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format (including HTML, RSS feeds, JSON, and XML).

Internally, while it provides choices for almost any functionality you might want (e.g., several popular databases, templating engines, etc.), it can also be extended to use other components if needed.

Secure

Django helps developers avoid many common security mistakes by providing a framework that has been engineered to "do the right things" to protect the website automatically. For example, Django provides a secure way to manage user accounts and passwords, avoiding common mistakes like putting session information in cookies where it is vulnerable (instead cookies just contain a key, and the actual data is stored in the database) or directly storing passwords rather than a password hash.

A password hash is a fixed-length value created by sending the password through a cryptographic hash function. Django can check if an entered password is correct by running it through the hash function and comparing the output to the stored hash value. However due to the "one-way" nature of the function, even if a stored hash value is compromised it is hard for an attacker to work out the original password.

Django enables protection against many vulnerabilities by default, including SQL injection, cross-site scripting, cross-site request forgery and clickjacking (see Website security for more details of such attacks).

Scalable

Django uses a component-based "shared-nothing" architecture (each part of the architecture is independent of the others, and can hence be replaced or changed if needed). Having a clear separation between the different parts means that it can scale for increased traffic by adding hardware at any level: caching servers, database servers, or application servers. Some of the busiest sites have successfully scaled Django to meet their demands (e.g. Instagram and Disqus, to name just two).

Maintainable

Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In particular, it makes use of the Don't Repeat Yourself (DRY) principle so there is no unnecessary duplication, reducing the

amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a lower level, groups related code into modules (along the lines of the Model View Controller (MVC) pattern).

Portable

Django is written in Python, which runs on many platforms. That means that you are not tied to any particular server platform, and can run your applications on many flavours of Linux, Windows, and macOS. Furthermore, Django is well-supported by many web hosting providers, who often provide specific infrastructure and documentation for hosting Django sites.

Where did it come from?

Django was initially developed between 2003 and 2005 by a web team who were responsible for creating and maintaining newspaper websites. After creating a number of sites, the team began to factor out and reuse lots of common code and design patterns. This common code evolved into a generic web development framework, which was open-sourced as the "Django" project in July 2005.

Django has continued to grow and improve, from its first milestone release (1.0) in September 2008 through to the recently-released version 4.0 (2022). Each release has added new functionality and bug fixes, ranging from support for new types of databases, template engines, and caching, through to the addition of "generic" view functions and classes (which reduce the amount of code that developers have to write for a number of programming tasks).

Note: Check out the release notes on the Django website to see what has changed in recent versions, and how much work is going into making Django better.

Django is now a thriving, collaborative open source project, with many thousands of users and contributors. While it does still have some features that reflect its origin, Django has evolved into a versatile framework that is capable of developing any type of website.

How popular is Django?

There isn't any readily-available and definitive measurement of popularity of server-side frameworks (although you can estimate popularity using mechanisms like counting the number of GitHub projects and StackOverflow questions for each platform). A better question is whether Django is "popular enough" to avoid the problems of unpopular platforms. Is it continuing to evolve? Can you get help if you need it? Is there an opportunity for you to get paid work if you learn Django?

Based on the number of high profile sites that use Django, the number of people contributing to the codebase, and the number of people providing both free and paid for support, then yes, Django is a popular framework!

High-profile sites that use Django include: Disqus, Instagram, Knight Foundation, MacArthur Foundation, Mozilla, National Geographic, Open Knowledge Foundation, Pinterest, and Open Stack (source: Django overview page).

Is Django opinionated?

Web frameworks often refer to themselves as "opinionated" or "unopinionated".

Opinionated frameworks are those with opinions about the "right way" to handle any particular task. They often support rapid development *in a particular domain* (solving problems of a particular type) because the right way to do anything is usually well-understood and well-documented. However they can be less flexible at solving problems outside their main domain, and tend to offer fewer choices for what components and approaches they can use.

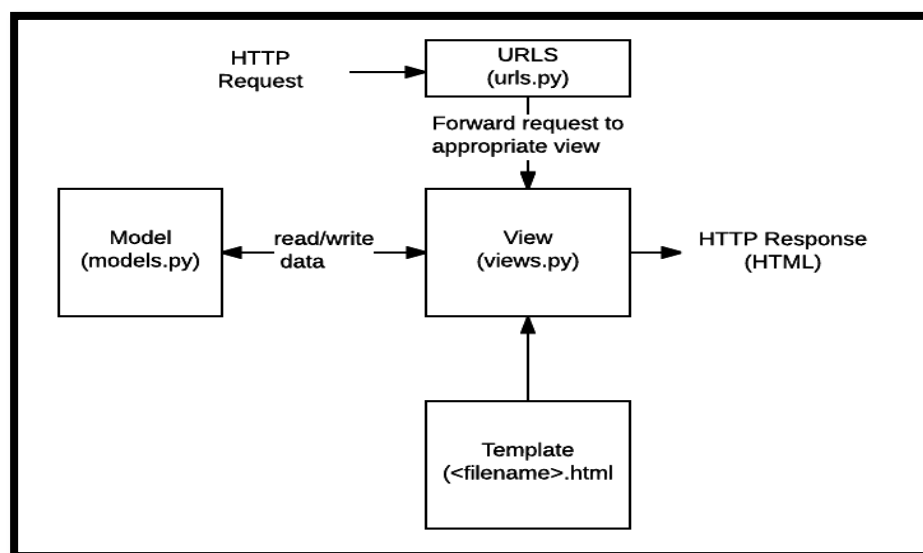
Unopinionated frameworks, by contrast, have far fewer restrictions on the best way to glue components together to achieve a goal, or even what components should be used. They make it easier for developers to use the most suitable tools to complete a particular task, albeit at the cost that you need to find those components yourself.

Django is "somewhat opinionated", and hence delivers the "best of both worlds". It provides a set of components to handle most web development tasks and one (or two) preferred ways to use them. However, Django's decoupled architecture means that you can usually pick and choose from a number of different options, or add support for completely new ones if desired.

What does Django code look like?

In a traditional data-driven website, a web application waits for HTTP requests from the web browser (or other client). When a request is received the application works out what is needed based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.

Django web applications typically group the code that handles each of these steps into separate files:



- **URLs:** While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.

- **View:** A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via *models*, and delegate the formatting of the response to *templates*.
- **Models:** Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
- **Templates:** A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A *view* can dynamically create an HTML page using an HTML template, populating it with data from a *model*. A template can be used to define the structure of any type of file; it doesn't have to be HTML!

Note: Django refers to this organization as the "Model View Template (MVT)" architecture. It has many similarities to the more familiar Model View Controller architecture.

The sections below will give you an idea of what these main parts of a Django app look like (we'll go into more detail later on in the course, once we've set up a development environment).

Sending the request to the right view (urls.py)

A URL mapper is typically stored in a file named **urls.py**. In the example below, the mapper (`urlpatterns`) defines a list of mappings between *routes* (specific URL *patterns*) and corresponding view functions. If an HTTP Request is received that has a URL matching a specified pattern, then the associated view function will be called and passed the request.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('book/<int:id>/', views.book_detail, name='book_detail'),
    path('catalog/', include('catalog.urls')),
    re_path(r'^([0-9]+)/$', views.best),
]
```

The `urlpatterns` object is a list of `path()` and/or `re_path()` functions (Python lists are defined using square brackets, where items are separated by commas and may have an optional trailing comma. For example: `[item1, item2, item3,]`).

The first argument to both methods is a route (pattern) that will be matched. The path() method uses angle brackets to define parts of a URL that will be captured and passed through to the view function as named arguments. The re_path() function uses a flexible pattern matching approach known as a regular expression. We'll talk about these in a later article!

The second argument is another function that will be called when the pattern is matched. The notation views.book_detail indicates that the function is called book_detail() and can be found in a module called views (i.e. inside a file named views.py)

Handling the request (views.py)

Views are the heart of the web application, receiving HTTP requests from web clients and returning HTTP responses. In between, they marshal the other resources of the framework to access databases, render templates, etc.

The example below shows a minimal view function index(), which could have been called by our URL mapper in the previous section. Like all view functions it receives an HttpRequest object as a parameter (request) and returns an HttpResponse object. In this case we don't do anything with the request, and our response returns a hard-coded string. We'll show you a request that does something more interesting in a later section.

filename: views.py (Django view functions)

```
from django.http import HttpResponse
```

```
def index(request):
```

```
    # Get an HttpRequest - the request parameter
```

```
    # perform operations using information from the request.
```

```
    # Return HttpResponse
```

```
    return HttpResponse('Hello from Django!')
```

Note: A little bit of Python:

- Python modules are "libraries" of functions, stored in separate files, that we might want to use in our code. Here we import just the HttpResponse object from the django.http module so that we can use it in our view: from django.http import

HttpResponse. There are other ways of importing some or all objects from a module.

- Functions are declared using the `def` keyword as shown above, with named parameters listed in brackets after the name of the function; the whole line ends in a colon. Note how the next lines are all **indented**. The indentation is important, as it specifies that the lines of code are inside that particular block (mandatory indentation is a key feature of Python, and is one reason that Python code is so easy to read).

Views are usually stored in a file called **views.py**.

Defining data models (models.py)

Django web applications manage and query data through Python objects referred to as models. Models define the structure of stored data, including the field *types* and possibly also their maximum size, default values, selection list options, help text for documentation, label text for forms, etc. The definition of the model is independent of the underlying database — you can choose one of several as part of your project settings. Once you've chosen what database you want to use, you don't need to talk to it directly at all — you just write your model structure and other code, and Django handles all the "dirty work" of communicating with the database for you.

The code snippet below shows a very simple Django model for a Team object. The Team class is derived from the Django class `models.Model`. It defines the team name and team level as character fields and specifies a maximum number of characters to be stored for each record. The `team_level` can be one of several values, so we define it as a choice field and provide a mapping between choices to be displayed and data to be stored, along with a default value.

```
# filename: models.py
```

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

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

```
    team_name = models.CharField(max_length=40)
```

```

TEAM_LEVELS = (
    ('U09', 'Under 09s'),
    ('U10', 'Under 10s'),
    ('U11', 'Under 11s'),
    # ...
    # list other team levels
)
team_level = models.CharField(max_length=3,
choices=TEAM_LEVELS, default='U11')

```

Note: A little bit of Python:

Python supports "object-oriented programming", a style of programming where we organize our code into objects, which include related data and functions for operating on that data. Objects can also inherit/extend/derive from other objects, allowing common behavior between related objects to be shared. In Python we use the keyword `class` to define the "blueprint" for an object. We can create multiple specific *instances* of the type of object based on the model in the class.

So for example, here we have a `Team` class, which derives from the `Model` class. This means it is a model, and will contain all the methods of a model, but we can also give it specialized features of its own too. In our model we define the fields our database will need to store our data, giving them specific names. Django uses these definitions, including the field names, to create the underlying database.

Querying data (views.py)

The Django model provides a simple query API for searching the associated database. This can match against a number of fields at a time using different criteria (e.g. exact, case-insensitive, greater than, etc.), and can support complex statements (for example, you can specify a search on U11 teams that have a team name that starts with "Fr" or ends with "al").

The code snippet shows a view function (resource handler) for displaying all of our U09 teams. The `list_teams = Team.objects.filter(team_level__exact="U09")` line shows how we can use the model query API to filter for all records where

the `team_level` field has exactly the text 'U09' (note how this criteria is passed to the `filter()` function as an argument, with the field name and `match` type separated by a double underscore: `team_level__exact`).

```
## filename: views.py
```

```
from django.shortcuts import render  
from .models import Team
```

```
def index(request):  
    list_teams = Team.objects.filter(team_level__exact="U09")  
    context = {'youngest_teams': list_teams}  
    return render(request, '/best/index.html', context)
```

This function uses the `render()` function to create the `HttpResponse` that is sent back to the browser. This function is a *shortcut*; it creates an HTML file by combining a specified HTML template and some data to insert in the template (provided in the variable named "context"). In the next section we show how the template has the data inserted in it to create the HTML.

Rendering data (HTML templates)

Template systems allow you to specify the structure of an output document, using placeholders for data that will be filled in when a page is generated. Templates are often used to create HTML, but can also create other types of document. Django supports both its native templating system and another popular Python library called Jinja2 out of the box (it can also be made to support other systems if needed).

The code snippet shows what the HTML template called by the `render()` function in the previous section might look like. This template has been written under the assumption that it will have access to a list variable called `youngest_teams` when it is rendered (this is contained in the context variable inside the `render()` function above). Inside the HTML skeleton we have an expression that first checks if the `youngest_teams` variable exists, and then iterates it in a for loop. On each iteration the template displays each team's `team_name` value in an `` element.

```
## filename: best/templates/best/index.html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Home page</title>
</head>
<body>
  {% if youngest_teams %}
    <ul>
      {% for team in youngest_teams %}
        <li>{{ team.team_name }}</li>
      {% endfor %}
    </ul>
  {% else %}
    <p>No teams are available.</p>
  {% endif %}
</body>
</html>
```

What else can you do?

The preceding sections show the main features that you'll use in almost every web application: URL mapping, views, models and templates. Just a few of the other things provided by Django include:

- **Forms:** HTML Forms are used to collect user data for processing on the server. Django simplifies form creation, validation, and processing.
- **User authentication and permissions:** Django includes a robust user authentication and permission system that has been built with security in mind.
- **Caching:** Creating content dynamically is much more computationally intensive (and slow) than serving static content. Django provides flexible caching so that you can store all or part of a rendered page so that it doesn't get re-rendered except when necessary.
- **Administration site:** The Django administration site is included by default when you create an app using the basic skeleton. It makes it trivially easy to provide an admin page for

site administrators to create, edit, and view any data models in your site.

- **Serializing data:** Django makes it easy to serialize and serve your data as XML or JSON. This can be useful when creating a web service (a website that purely serves data to be consumed by other applications or sites, and doesn't display anything itself), or when creating a website in which the client-side code handles all the rendering of data.

Django Model

In Django, a model is a class which is used to contain essential fields and methods. Each model class maps to a single table in the database.

Django Model is a subclass of **django.db.models.Model** and each field of the model class represents a database field (column).

Django provides us a database-abstraction API which allows us to create, retrieve, update and delete a record from the mapped table.

Model is defined in **Models.py** file. This file can contain multiple models.

Let's see an example here, we are creating a model **Employee** which has two fields **first_name** and **last_name**.

1. from django.db **import** models
- 2.
3. **class** Employee(models.Model):
4. first_name = models.CharField(max_length=30)
5. last_name = models.CharField(max_length=30)

The **first_name** and **last_name** fields are specified as class attributes and each attribute maps to a database column.

This model will create a table into the database that looks like below.

1. CREATE TABLE appname_employee (
2. "id" INT NOT NULL PRIMARY KEY,
3. "first_name" varchar(30) NOT NULL,
4. "last_name" varchar(30) NOT NULL
5.);

The created table contains an auto-created **id field**. The name of the table is a combination of app name and model name that can be changed further.

Register / Use Model

After creating a model, register model into the **INSTALLED_APPS** inside **settings.py**.

For example,

```
1. INSTALLED_APPS = [  
2.     #...  
3.     'appname',  
4.     #...  
5. ]
```

Databases

Django officially supports the following databases:

- PostgreSQL
- MariaDB
- MySQL
- Oracle
- SQLite

There are also a number of database backends provided by third parties.

Django attempts to support as many features as possible on all database backends. However, not all database backends are alike, and we've had to make design decisions on which features to support and which assumptions we can make safely.

This file describes some of the features that might be relevant to Django usage. It is not intended as a replacement for server-specific documentation or reference manuals.

Persistent connections

Persistent connections avoid the overhead of re-establishing a connection to the database in each HTTP request. They're controlled by the **CONN_MAX_AGE** parameter which defines the maximum lifetime of a connection. It can be set independently for each database.

The default value is **0**, preserving the historical behaviour of closing the database connection at the end of each request. To enable persistent

connections, set **CONN_MAX_AGE** to a positive integer of seconds. For unlimited persistent connections, set it to **None**.

Connection management

Django opens a connection to the database when it first makes a database query. It keeps this connection open and reuses it in subsequent requests. Django closes the connection once it exceeds the maximum age defined by **CONN_MAX_AGE** or when it isn't usable any longer.

In detail, Django automatically opens a connection to the database whenever it needs one and doesn't have one already — either because this is the first connection, or because the previous connection was closed.

At the beginning of each request, Django closes the connection if it has reached its maximum age. If your database terminates idle connections after some time, you should set **CONN_MAX_AGE** to a lower value, so that Django doesn't attempt to use a connection that has been terminated by the database server. (This problem may only affect very low traffic sites.)

At the end of each request, Django closes the connection if it has reached its maximum age or if it is in an unrecoverable error state. If any database errors have occurred while processing the requests, Django checks whether the connection still works, and closes it if it doesn't. Thus, database errors affect at most one request per each application's worker thread; if the connection becomes unusable, the next request gets a fresh connection.

Setting **CONN_HEALTH_CHECKS** to **True** can be used to improve the robustness of connection reuse and prevent errors when a connection has been closed by the database server which is now ready to accept and serve new connections, e.g. after database server restart. The health check is performed only once per request and only if the database is being accessed during the handling of the request.

Changed in Django 4.1: The **CONN_HEALTH_CHECKS** setting was added.

CHAPTER 3

DESIGN AND IMPLEMENTATION

3.1 Dataflow Diagram

The DFD is also known as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system. It maps out the flow of information for any process or system, how data is processed in terms of inputs and outputs. It uses defined symbols like rectangles, circles and arrows to show data inputs, outputs, storage points and the routes between each destination. They can be used to analyse an existing system or model of a new one. A DFD can often visually “say” things that would be hard to explain in words and they work for both technical and non- technical.

There are four components in DFD:

1. External Entity
2. Process
3. Data Flow
4. data Store

1) External Entity:

It is an outside system that sends or receives data, communicating with the system. They are the sources and destinations of information entering and leaving the system. They might be an outside organization or person, a computer system or a business system. They are known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram. These are sources and destinations of the system’s input and output.

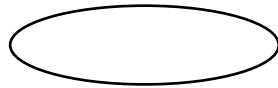
Representation:



2) Process:

It is just like a function that changes the data, producing an output. It might perform computations for sort data based on logic or direct the dataflow based on business rules.

Representation:



3) Data Flow:

A dataflow represents a package of information flowing between two objects in the dataflow diagram, Data flows are used to model the flow of information into the system, out of the system and between the elements within the system.

Representation:



4) Data Store:

These are the files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label.

Representation:

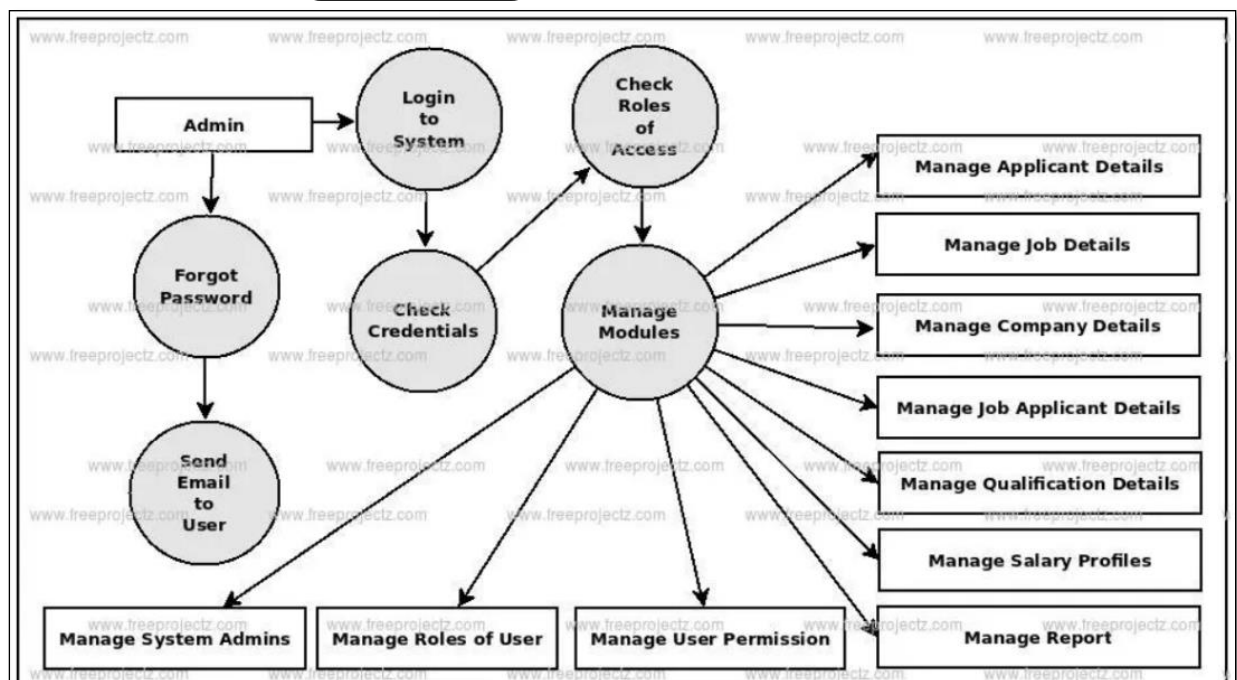
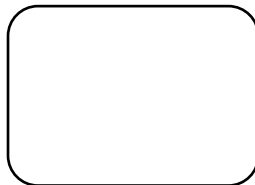


Fig.3.1 Data Flow Diagram for Job Searching Portal

3.2 UML DIAGRAMS

UML stands for Unified Modelling Language. Taking SRS document of analysis as input to the design phase drawn UML diagrams. The UML is only language so is just one part of the software development method. The UML is process independent, although optimally it should be used in a process that should be driven, architecture-centric, iterative, and incremental. The UML is language for visualizing, specifying, constructing, documenting the articles in a software- intensive system. It is based on diagrammatic representations of software components. A modelling language is a language whose vocabulary and rules focus on the conceptual and physical representation of the system. A modelling language such as the UML is thus a standard language for software blueprints. The UML is a graphical language, which consists of all interesting systems. There are also different structures that can transcend what can be represented in a programming language. These are different diagrams in UML.

3.2.1 Use Case Diagram

Use Case during requirement elicitation and analysis to represent the functionality of the system. Use case describes a function by the system that yields a visible result for an actor. The identification of actors and use cases result in the definitions of the boundary of the system i.e., differentiating the tasks accomplished by the system and the tasks accomplished by its environment. The actors are outside the boundary of the system, whereas the use cases are inside the boundary of the system. Use case describes the behaviour of the system as seen from the actor's point of view. It describes the function provided by the system as a set of events that yield a visible result for the actor.

Purpose of Use Case Diagrams

The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams 41 (activity, sequence, collaboration, and Statechart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams. Use case diagrams are

used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analysed to gather its functionalities, use cases are prepared and actors are identified.

When the initial task is complete, use case diagrams are modelled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements and actors.

How to Draw a Use Case Diagram?

Use case diagrams are considered for high level requirement analysis of a system. When the requirements of a system are analyzed, the functionalities are captured in use cases.

We can say that use cases are nothing but the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors.

Actors can be defined as something that interacts with the system. Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified. Functionalities to be represented as use case

Actors

Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. After identifying the above items, we have to use the following guidelines to draw an efficient use case diagram.

The name of a use case is very important. The name should be chosen in such a way so that it can identify the functionalities performed. Give a suitable name for actors. Show relationships and dependencies clearly in the diagram. Do not try to include all types of relationships, as the main purpose of the diagram is to identify the requirements. Use notes whenever required to clarify some important points

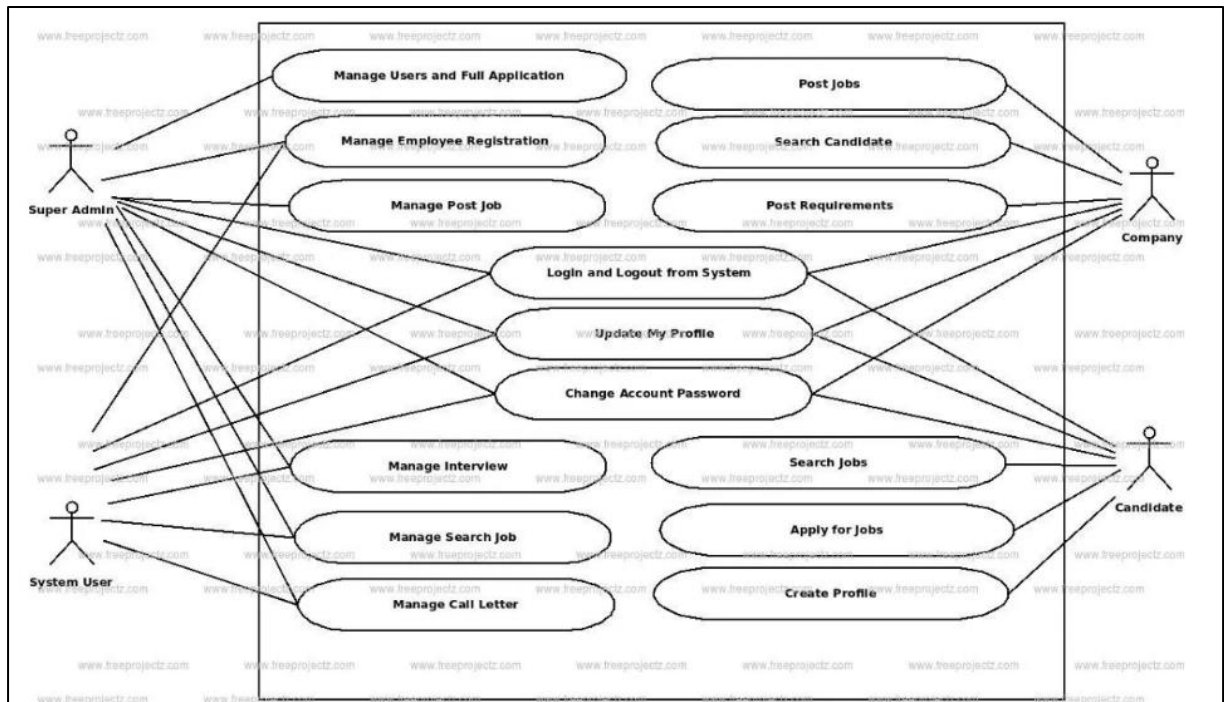


Fig.3.2 Use Case Diagram For Job Searching Portal

3.2.2 Class Diagram

Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagram describe the different perspective when designing a system-conceptual, specification and implementation. Classes are composed of three things: name, attributes, and operations. Class diagram also display relationships such as containment, inheritance, association etc. The association relationship is most common relationship in a class diagram. The association shows the relationship between instances of classes.

How to Draw a Class Diagram?

Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram. Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view.

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represent the whole system. The following points should be remembered while drawing a class diagram

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified For each class
- Minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram.
- At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

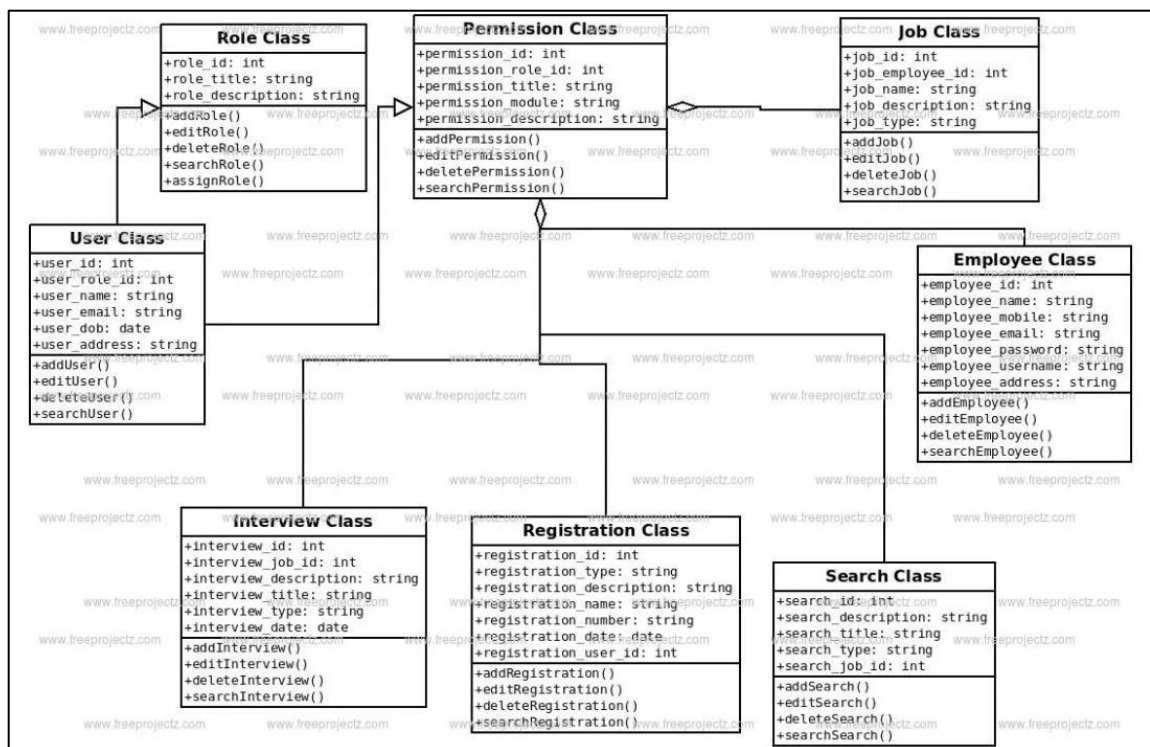


Fig.3.3 Class Diagram for Job Searching Portal

3.2.3 Sequence Diagram

Sequence diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension(time) and horizontal dimension (different objects). Objects: Object can be viewed as an entity at a particular point in time with specific value and as a holder of identity.

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

If the lifeline is that of an object, it demonstrates a role. Leaving the instance name blank can represent anonymous and unnamed instances. Messages, written with horizontal arrows with the message name written above them, display interaction. Solid arrow heads represent synchronous calls, open arrow heads represent asynchronous messages, and dashed lines represent reply messages. If a caller sends a synchronous message, it must wait until the message is done, such as invoking a subroutine. If a caller sends an asynchronous message, it can continue processing and doesn't have to wait for a response. Asynchronous calls are present in multithreaded applications, event-driven applications and in message-oriented middleware. Activation boxes, or method-call boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message (Execution Specifications in UML). Objects calling methods on themselves use messages and add new activation boxes on top of any others to indicate a further level of processing. If an object is destroyed (removed from memory), an X is drawn on bottom of the lifeline, and the dashed line ceases to be drawn below it. It should be the result of a message, either from the object itself, or another.

A message sent from outside the diagram can be represented by a message originating from a filled-in circle (found message in UML) or from a border of the sequence diagram (gate in UML).

UML has introduced significant improvements to the capabilities of sequence diagrams. Most of these improvements are based on the idea of interaction fragments which represent smaller pieces of an enclosing interaction. Multiple interaction fragments are combined to create a variety of combined fragments, which are then used to model interactions that include parallelism, conditional branches, optional interactions

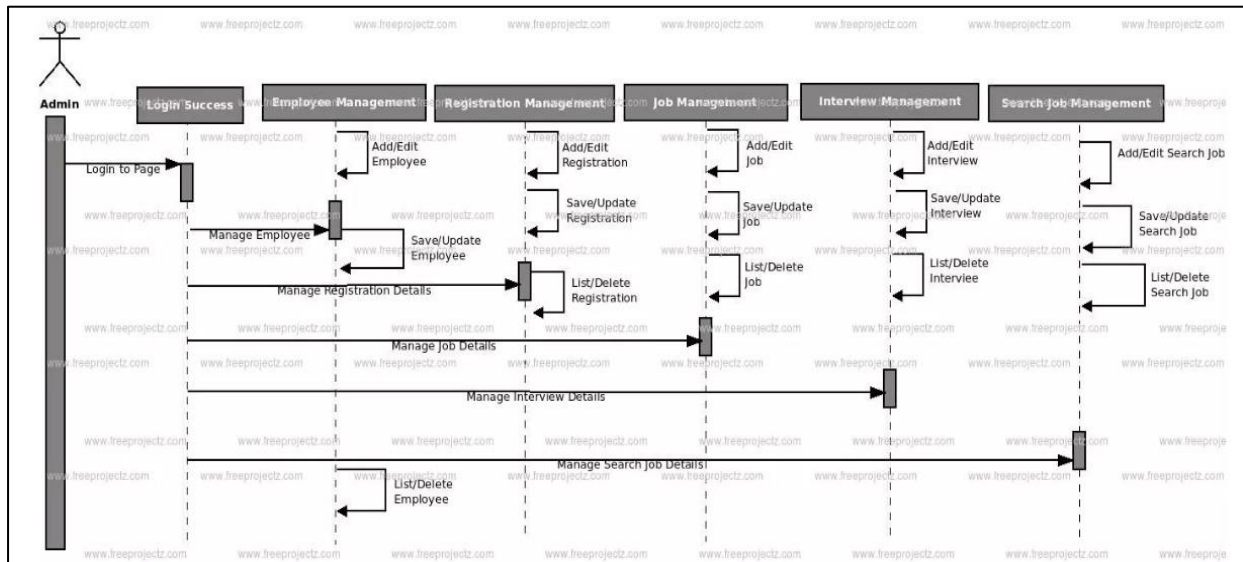


Fig.3.4 Sequence Diagram for Job Searching Portal

3.2.4 State Chart

A state chart diagram describes a state machine which shows the behaviour of classes. It shows the actual changes in state not processes or commands that create those changes and is the dynamic behaviour of objects over time by modelling the life cycle of objects of each class.

It describes how an object is changing from one state to another state.

There are mainly two states in State Chart Diagram:

1. Initial State
2. Final-State.

Some of the components of State Chart Diagram are:

State: It is a condition or situation in life cycle of an object during which it's satisfies same condition or performs some activity or waits for some event.

Transition: It is a relationship between two states indicating that object in first state performs some actions and enters into the next state or event.

Event: An event is specification of significant occurrence that has a location in time and space.

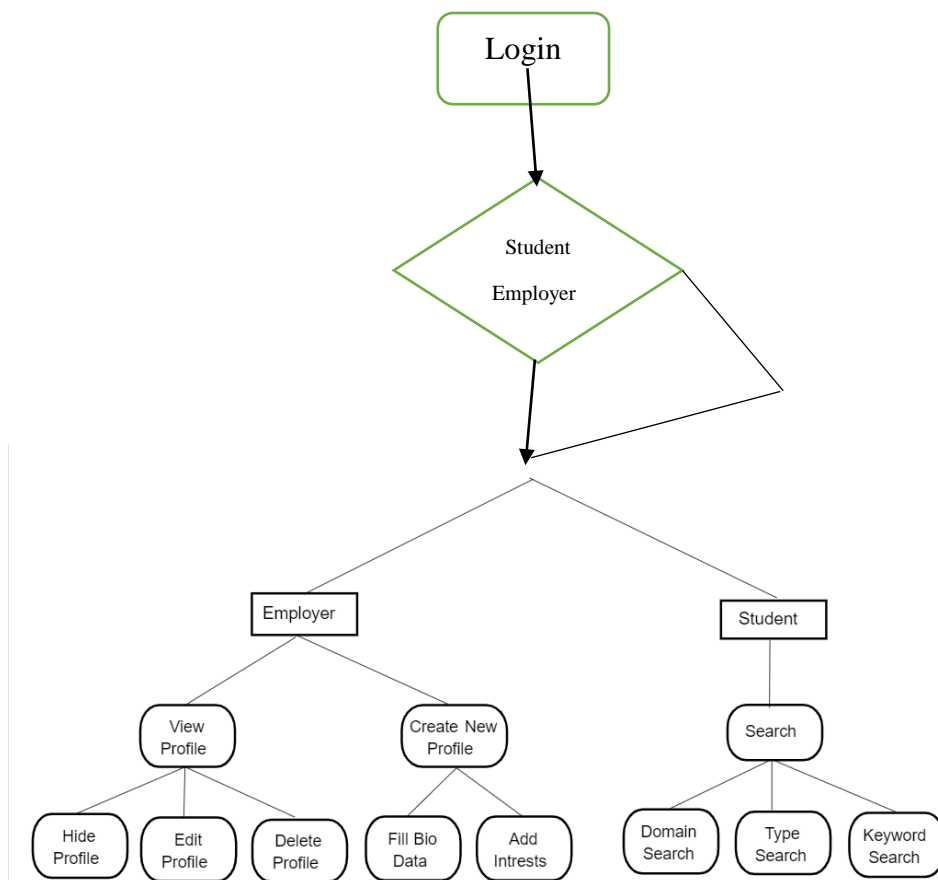


Fig.3.5 State Chart Diagram for Job Searching Portal

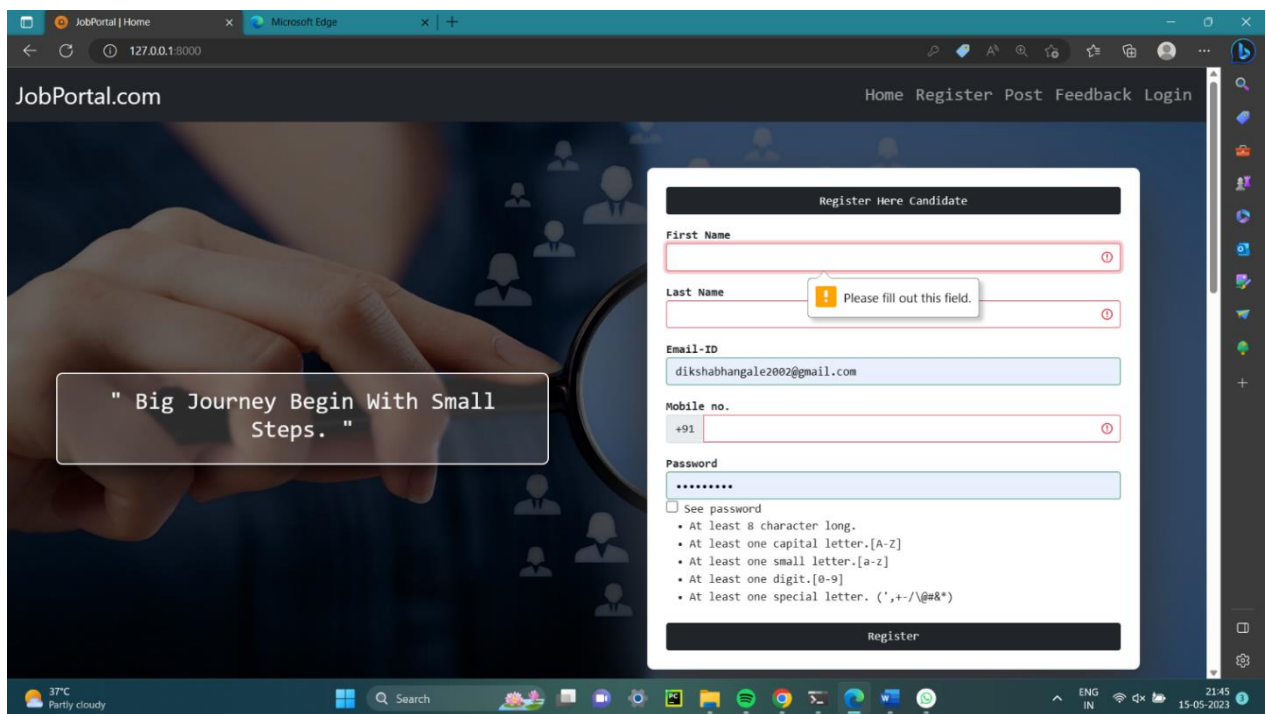
CHAPTER 4

RESULTS AND DISCUSSION

Job Portal Layout Screenshots:

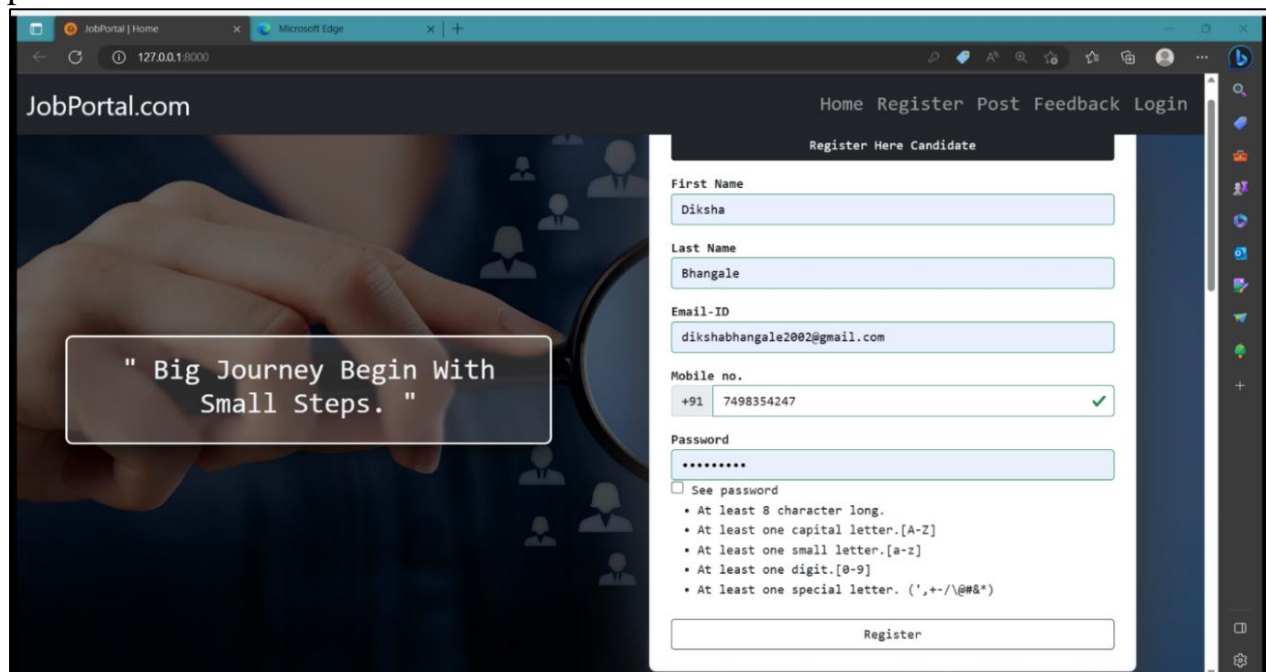
Let's see how our site looks like:

1. This is the layout of our website. As you can see, if user doesn't fill all the fields, intelligence is provided



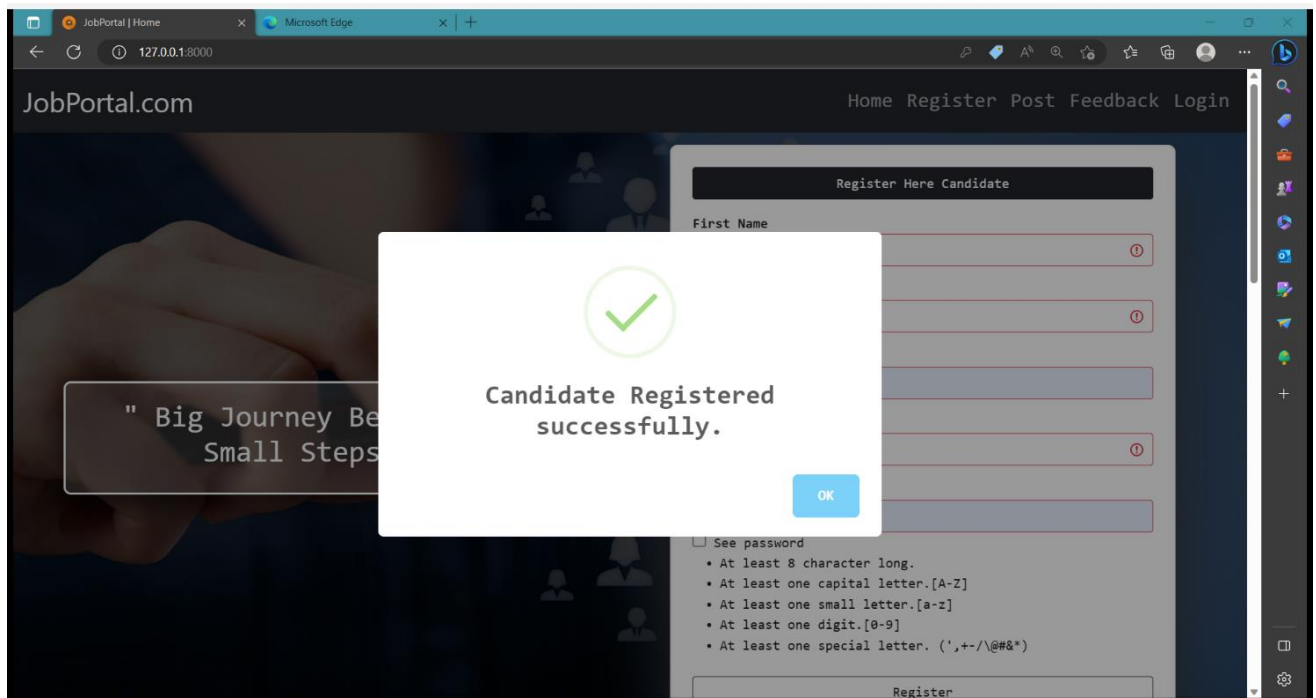
The screenshot shows the 'Register Here Candidate' form on the JobPortal.com website. The form includes fields for First Name, Last Name, Email-ID, Mobile no., and Password. The Last Name field is highlighted with a red border and a yellow warning icon, with a tooltip message: 'Please fill out this field.' The Password field is also highlighted with a red border. The form includes a 'See password' checkbox and a list of password requirements: 'At least 8 character long.', 'At least one capital letter.[A-Z]', 'At least one small letter.[a-z]', 'At least one digit.[0-9]', and 'At least one special letter. (',+,-/,@#&*)'. The 'Register' button is at the bottom of the form. The background of the website features a hand holding a magnifying glass over a document with the text: 'Big Journey Begin With Small Steps.'

2. As you can see, candidates who want to apply for job can register here using form provided

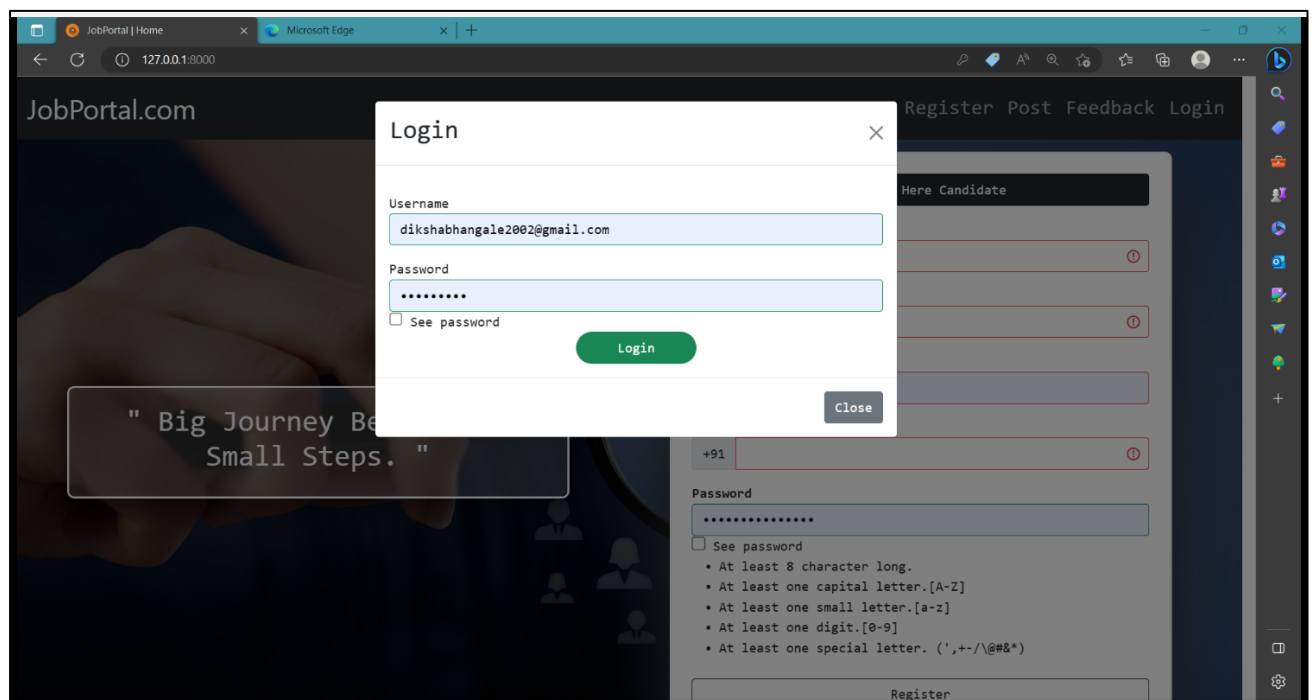


The screenshot shows the 'Register Here Candidate' form on the JobPortal.com website, now with the following data entered: First Name: Diksha, Last Name: Bhargale, Email-ID: dikshabhargale2002@gmail.com, Mobile no.: +91 7498354247 (with a green checkmark), and Password: *****. The 'See password' checkbox is unchecked. The password requirements are listed below the password field. The 'Register' button is at the bottom of the form. The background of the website features a hand holding a magnifying glass over a document with the text: 'Big Journey Begin With Small Steps.'

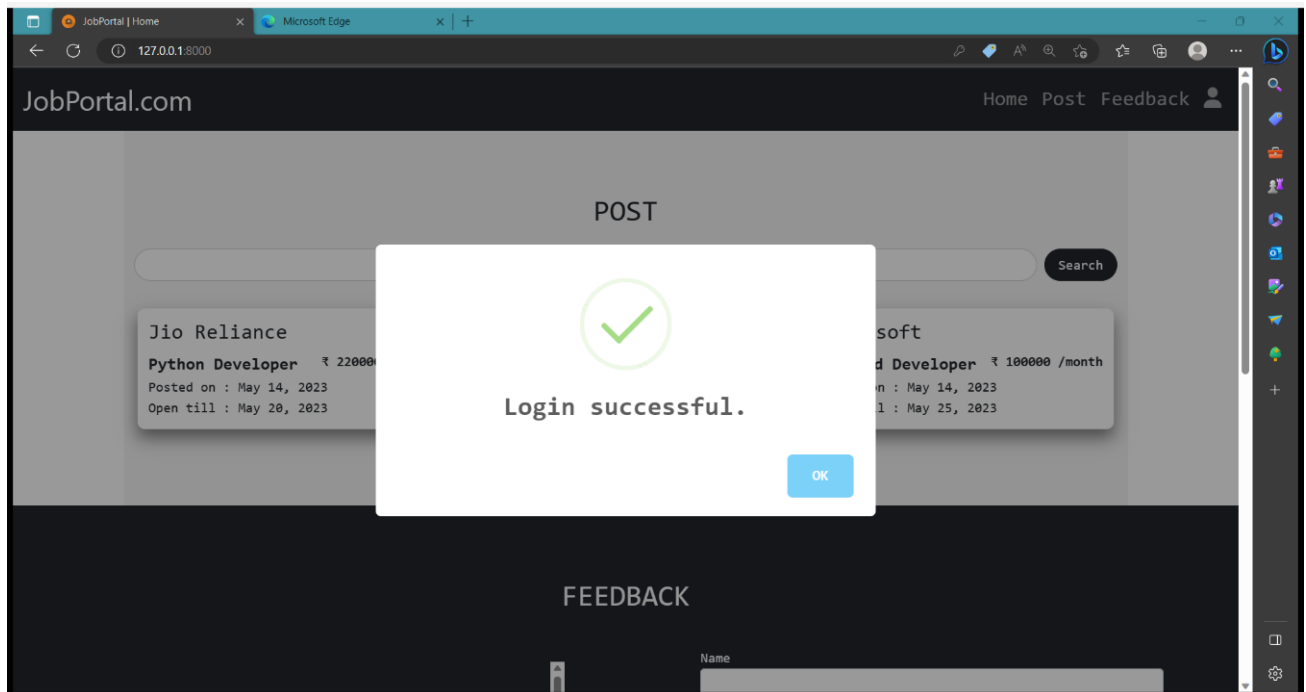
3. When the candidate registration is successful, such pop-up will appear which will help candidate to assure their registration



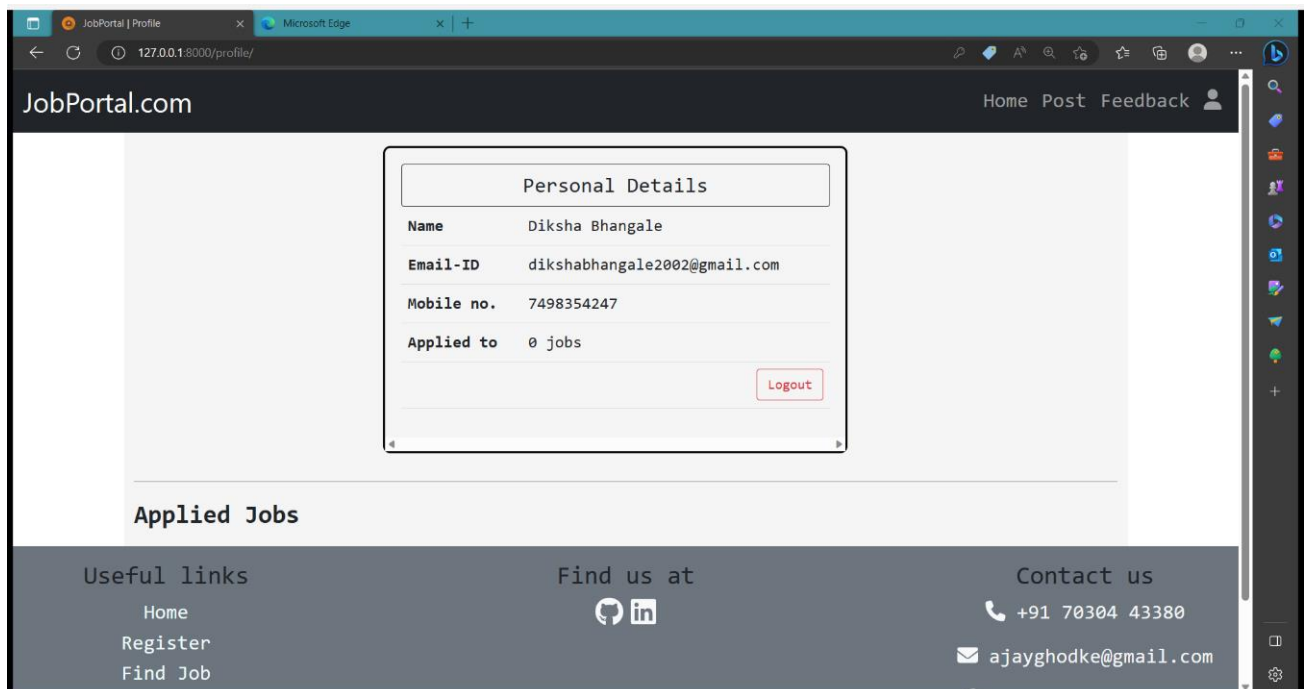
4. Once the candidate registers themselves, they can log in by filling their correct credentials in form provided.



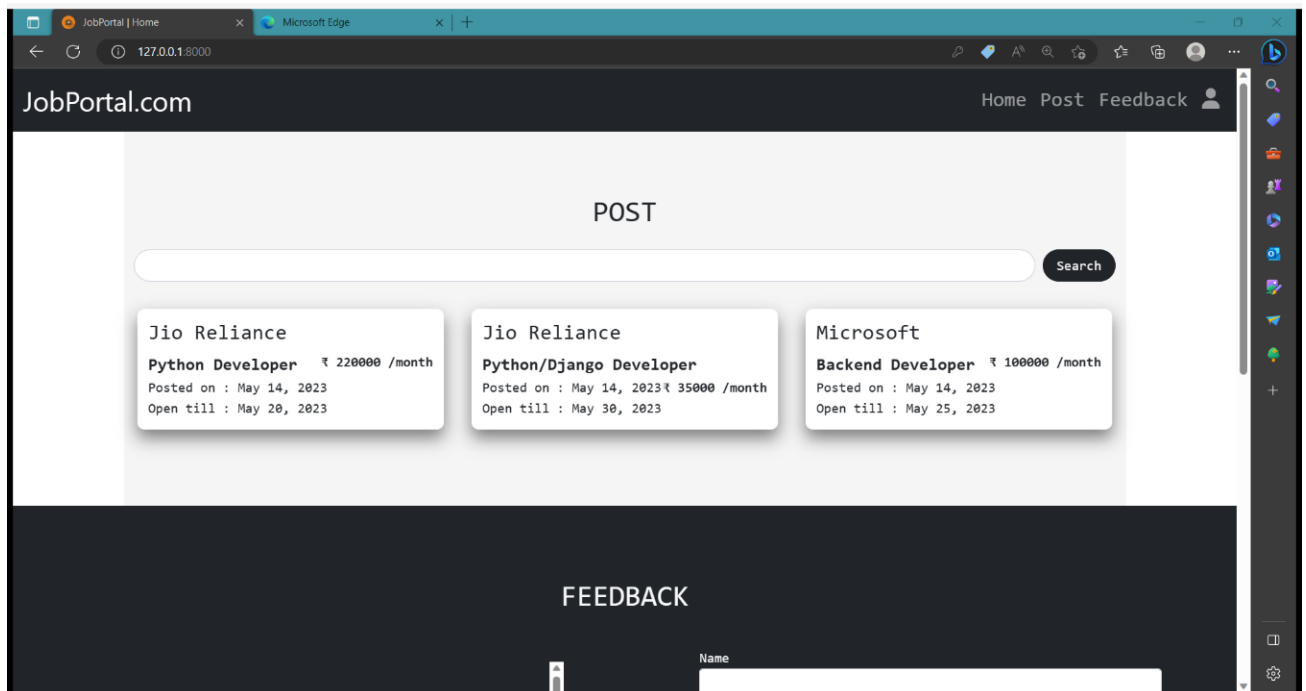
5. After successful log-in, pop up is provided for the candidates who log-in.



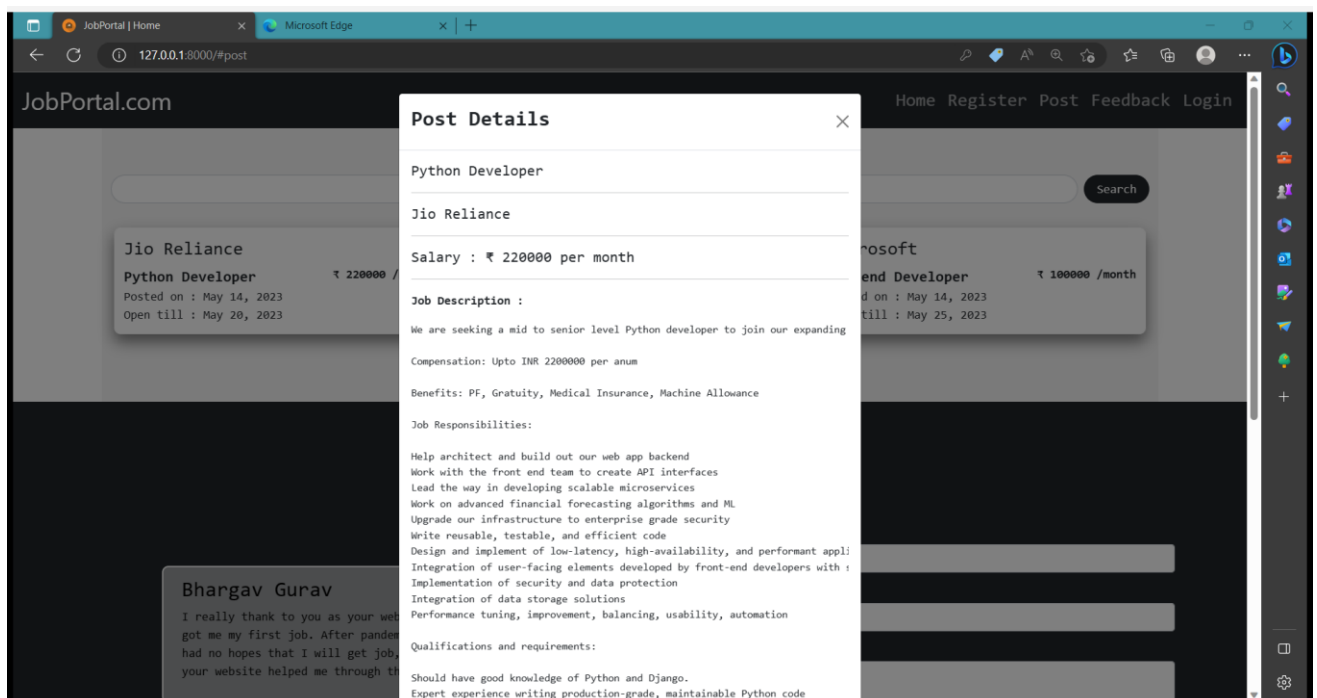
6. Candidates can check their details (name, email-ID, mobile no, applied jobs, etc) in profile section



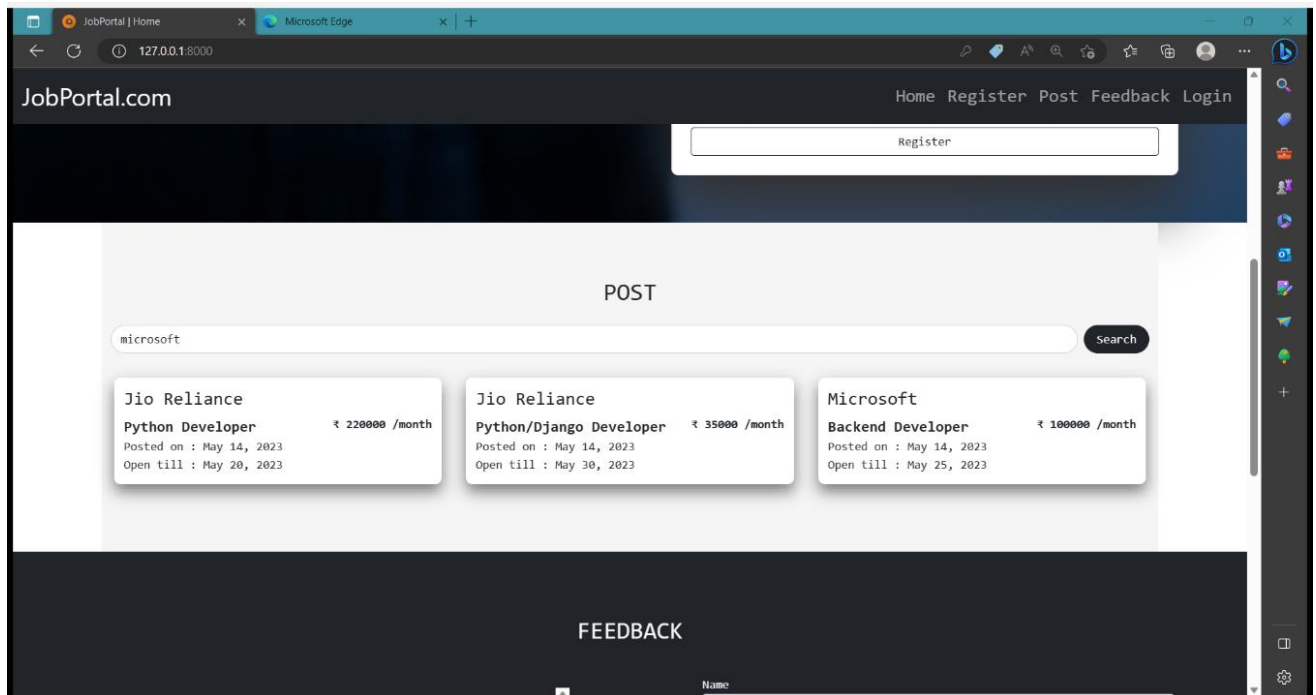
7. Candidates can check out the posts uploaded by various recruiters regarding job vacancies.



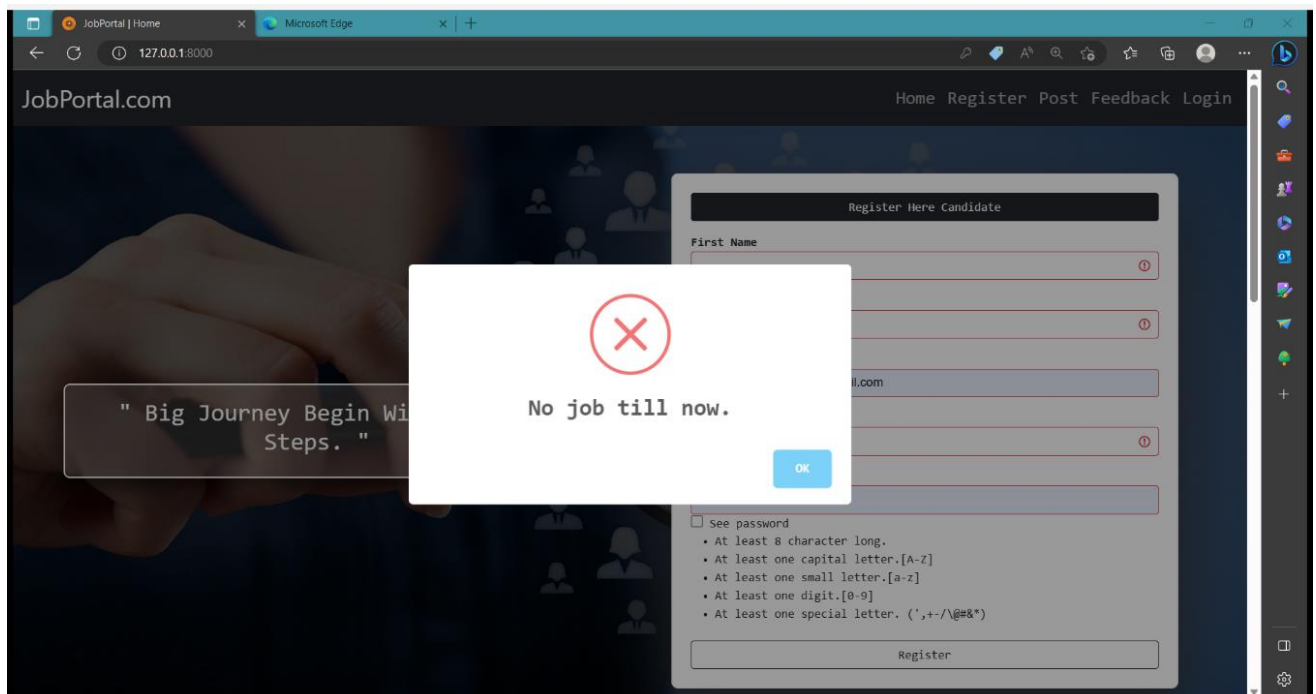
8. After clicking on posts, details regarding that post will be shown (Designation, salary, job description, etc)



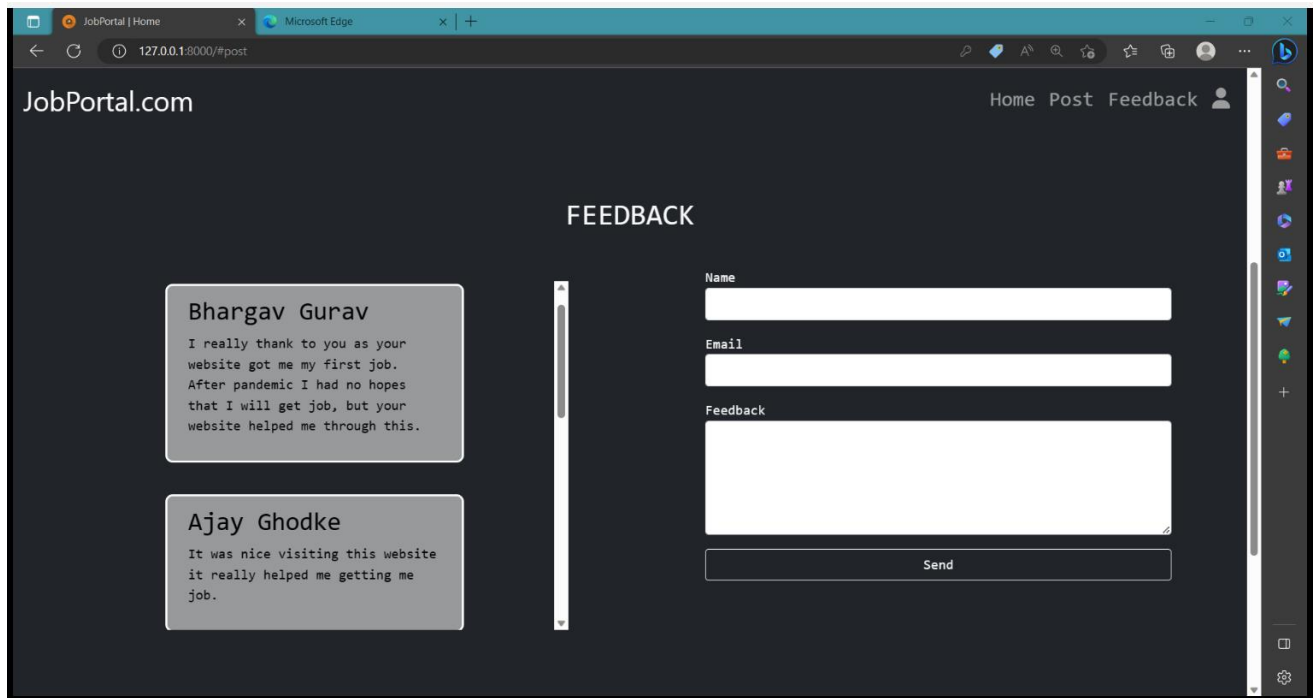
9. Candidate can search for particular post using search bar



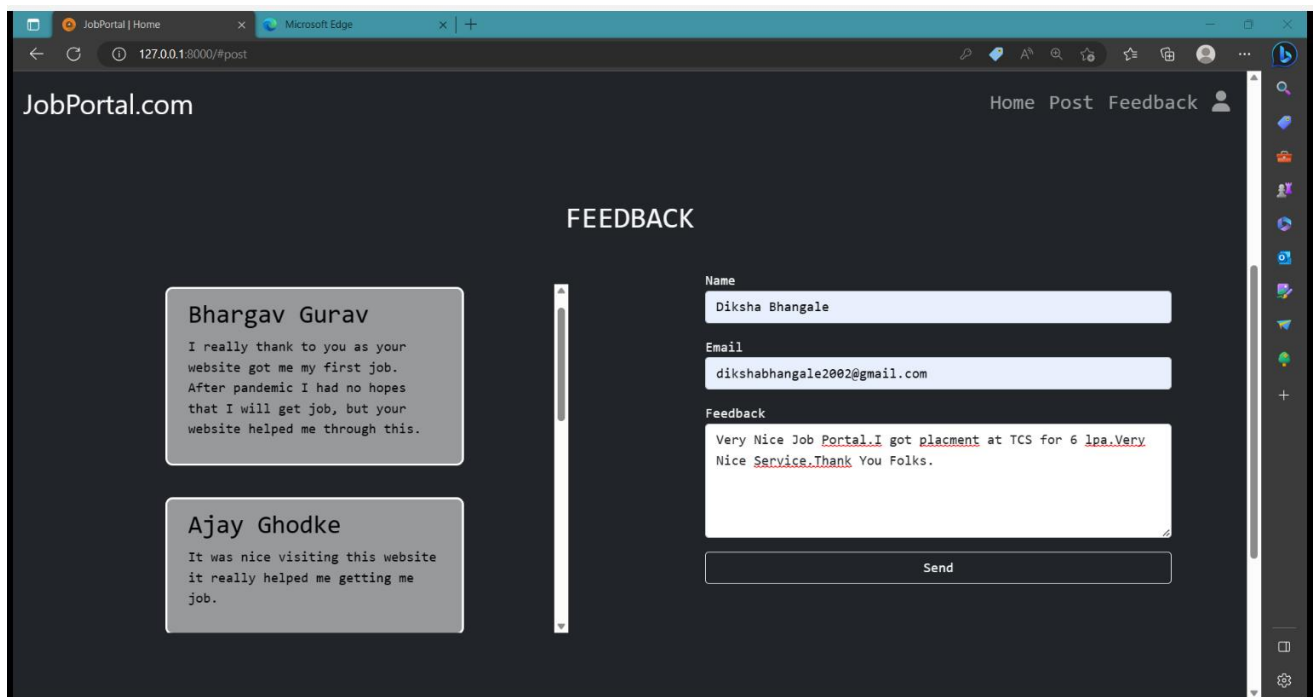
10. If no match is found for search, pop up is provided



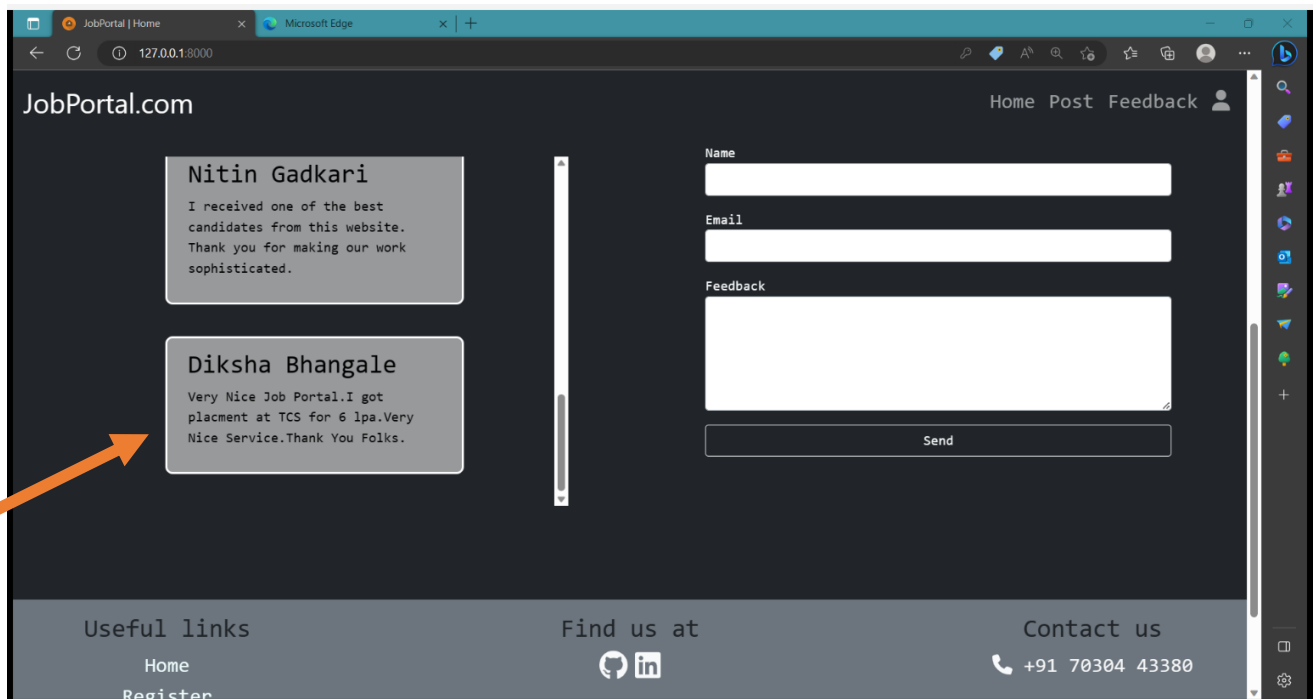
11. Feedback section is provided where candidates can read as well as upload the feedbacks



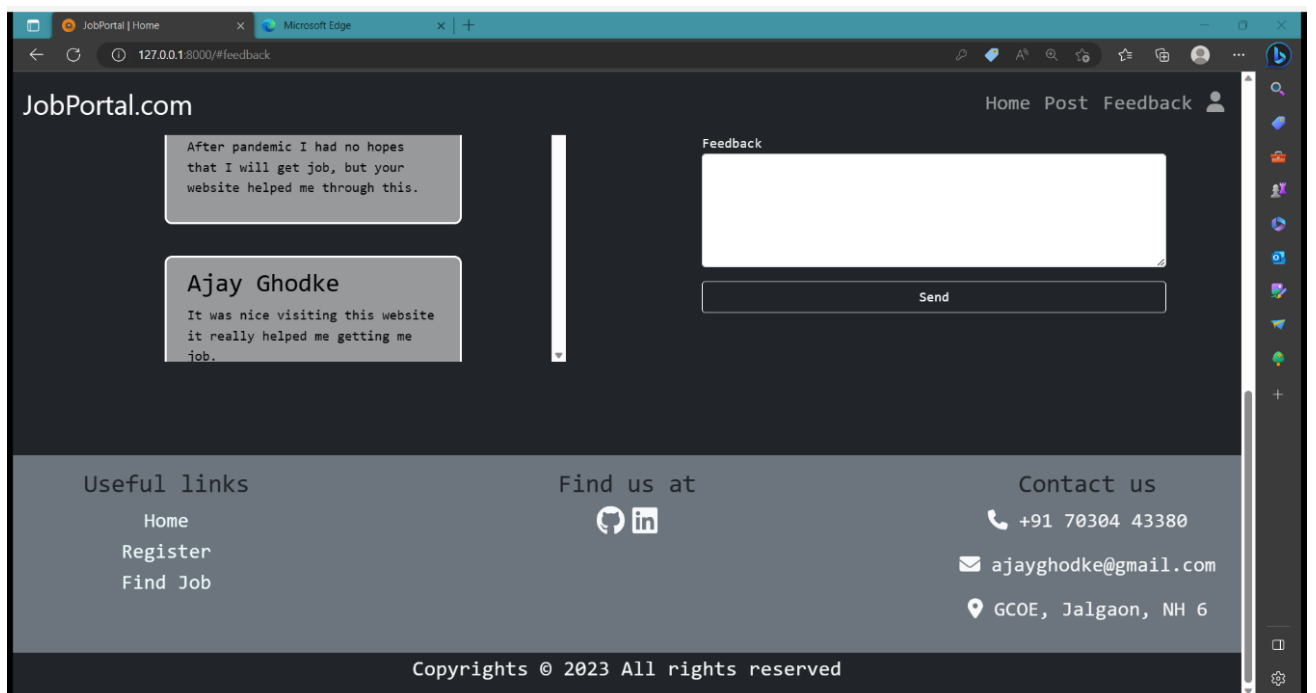
12. Candidates can even upload more than one feedbacks.



13. The newly uploaded feedback will immediately get displayed in feedback section.



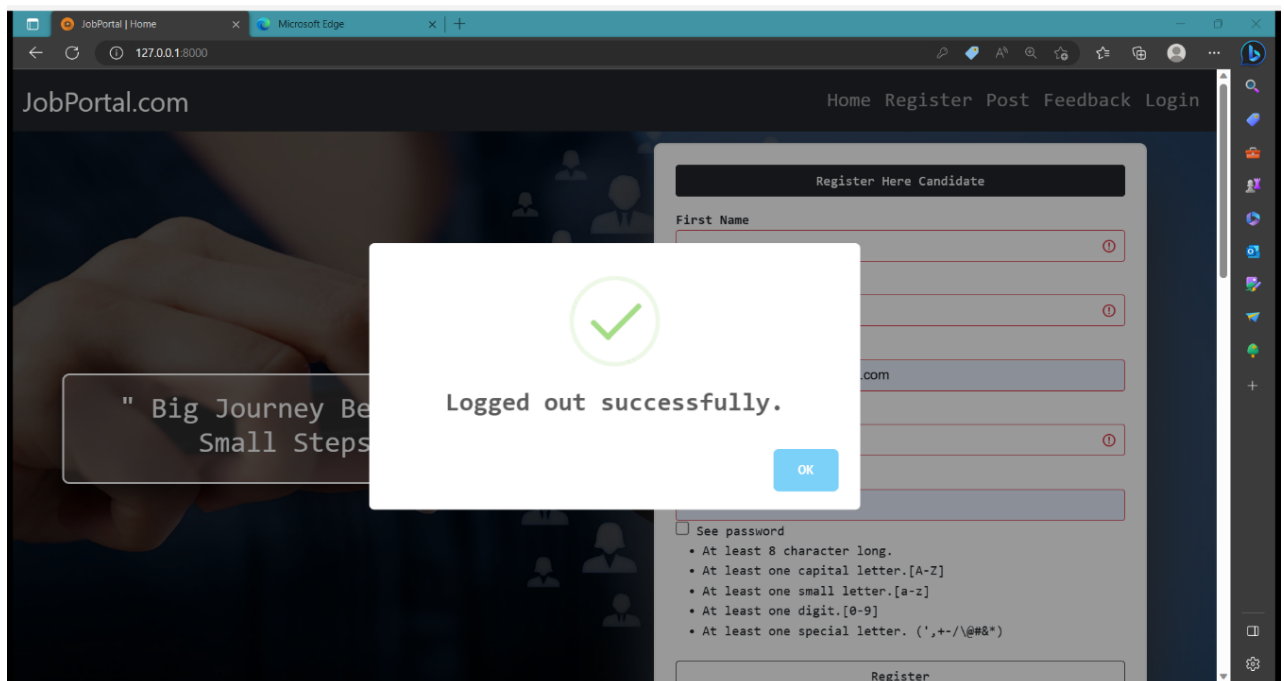
14. At footer, some buttons like “Useful links, finds us at, and Contact Us” are provided which will help candidate to reach the developer.



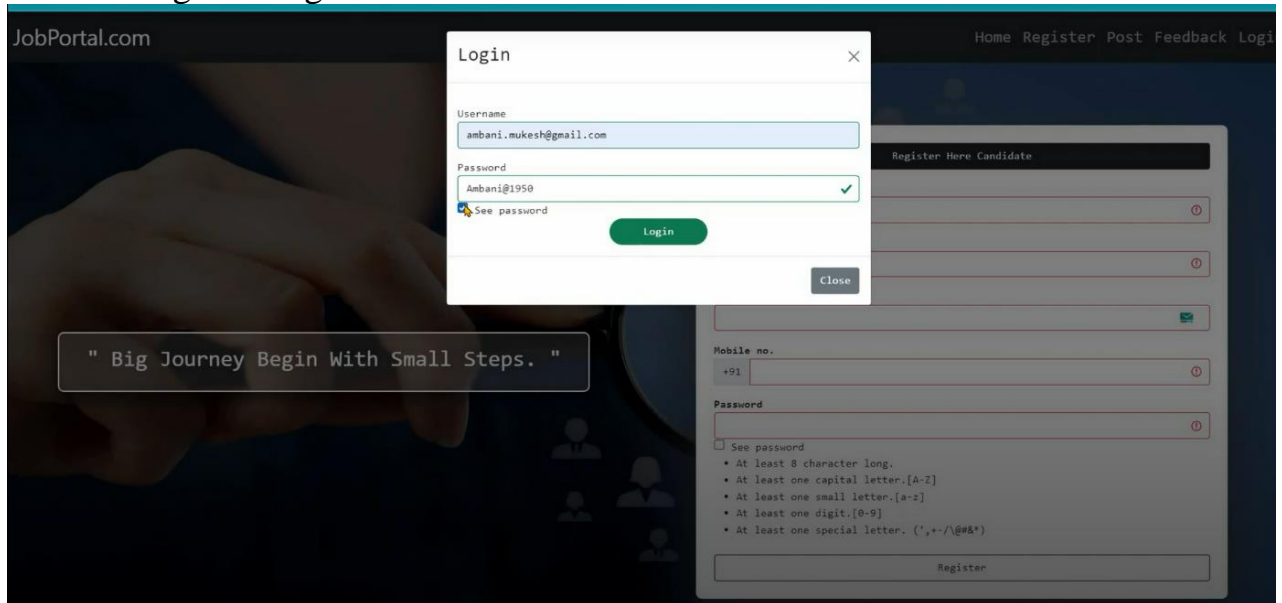
15. When candidate will sign-out, confirmation pop-up is provided.



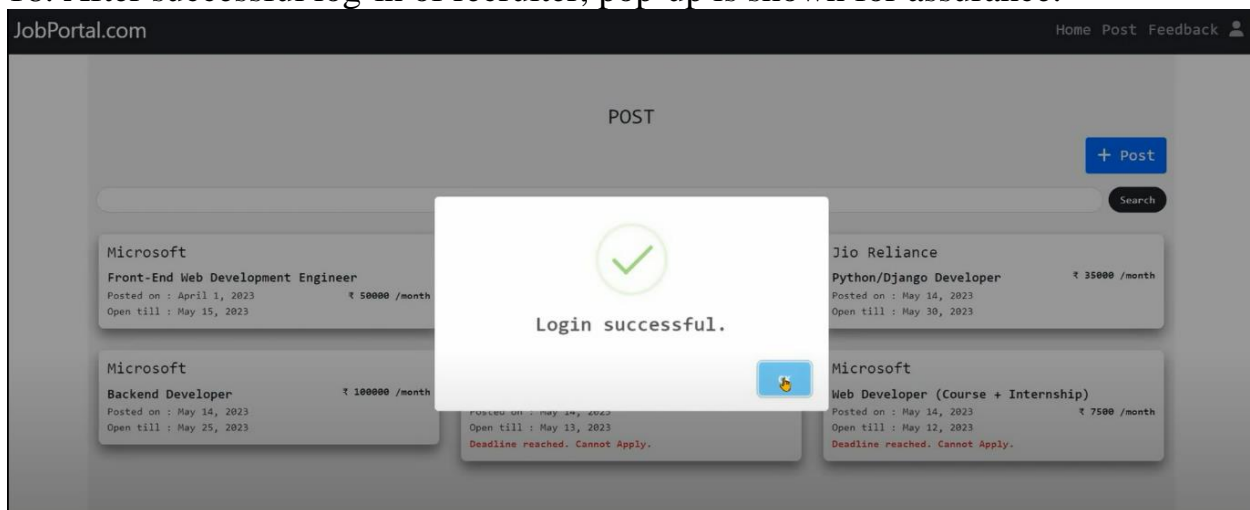
16. After logging-out, pop-up is shown for assurance.



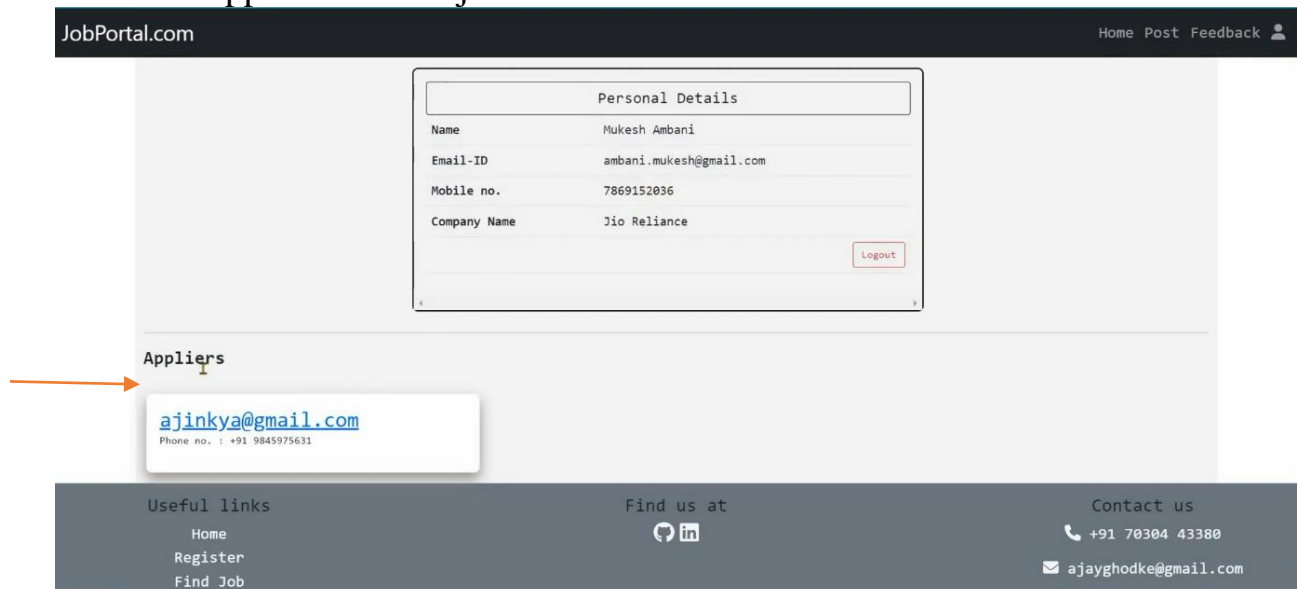
17. Using “Register” button in Navbar, companies/recruiters can register themselves and then log-in using their credentials.



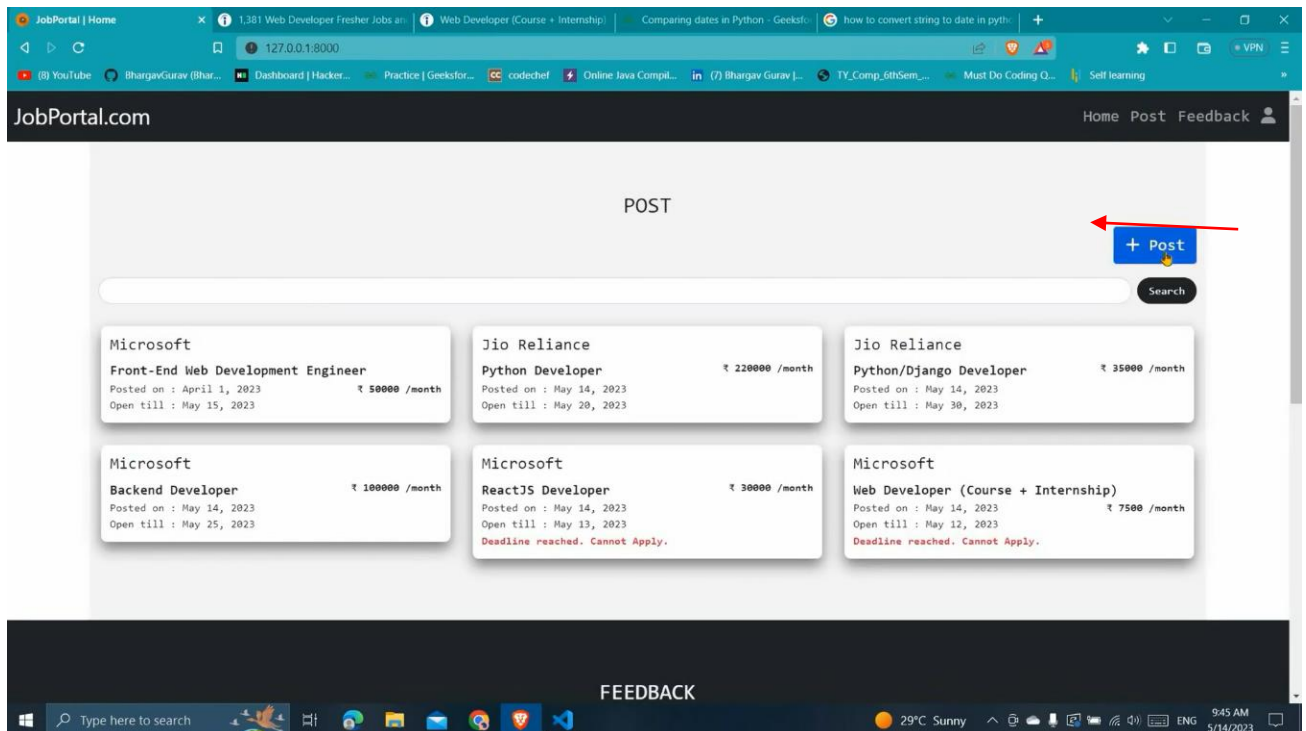
18. After successful log-in of recruiter, pop-up is shown for assurance.



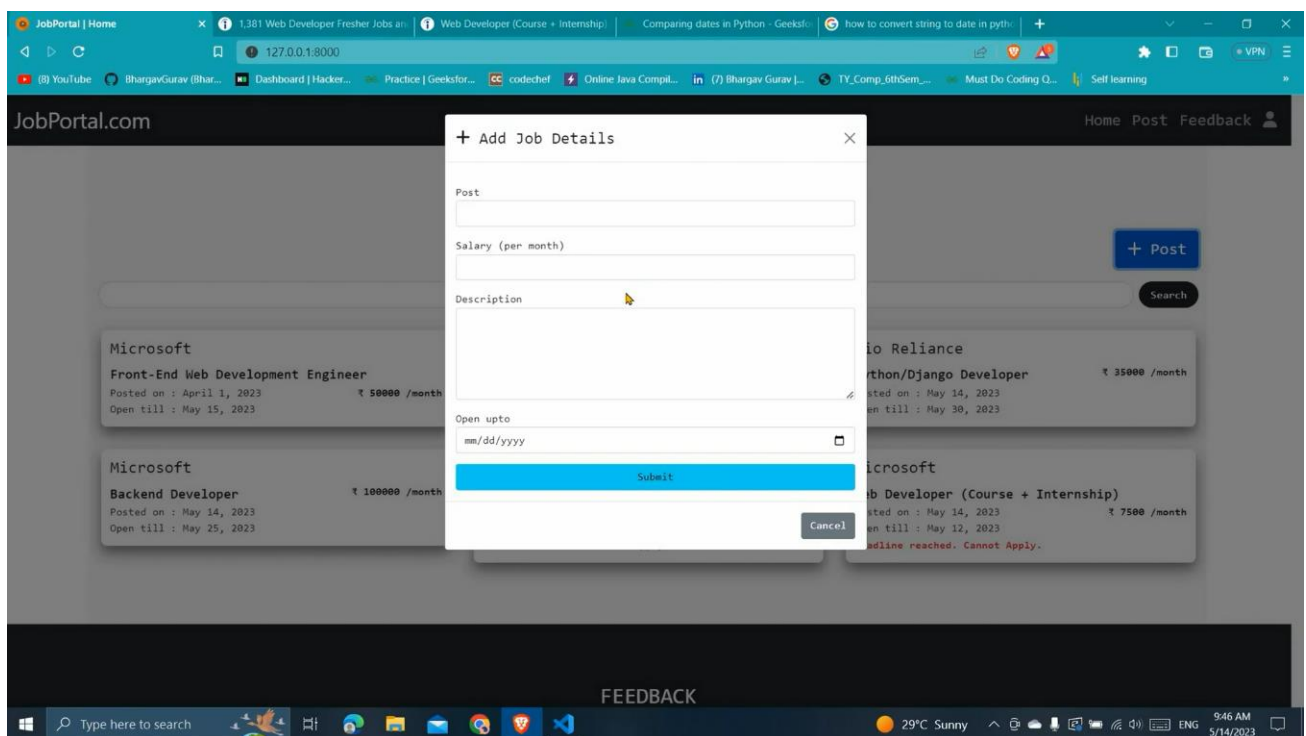
19. In profile section, recruiter can see their details as well as the list of candidates who have applied for their jobs.



20. Recruiters can add new post using “Post” button provided at the top



21. Here, recruiters have to add the job details and for that, form is provided.



CHAPTER 5

CONCLUSION

- Now-a-days people are really getting tired of traditional ways of finding jobs as well as recruiters are tired of hiring the employees with good skills in traditional manner.
- Finding jobs that best suits the interests and skill set is quite challenging task for the job seekers as well as it is tough for recruiters also to find out good employees.
- Hence “Job Searching Portal” was successfully designed and implemented by our team.
- We developed the system which will help both the employee and employer for better job seeking and selection process.
- We used “Django” which is a python-based framework for both the frontend and backend designing of our software.
- This project is robust and will definitely help people to overcome the problems faced in traditional job seeking processes.

CHAPTER 6

FUTURE SCOPE

- As of Indian market, there is ample opportunities for the job portal sites, as a greater number of educated and skilled young people are coming out each and every year.
- Also, as the growth rate of India is zooming to be at a healthy rate over 7%, so it is boom time for corporate also.
- So, more and more number of lucrative careers will be available for the job seekers.
- So, it is now the right period for the job portal sites to think out of the box, and to make most of the opportunities available
-

CHAPTER 7

REFERENCES

- https://bvmengineering.ac.in/NAAC/Criteria1/1.3/1.3.4/Online_Job_Portal_416_419_435.pdf
- <https://1000projects.org/project-report-online-job-portal.html>
- <https://www.scribd.com/document/412834710/Online-Job-Portal-Complete-Project-Report>
- JOB SEARCH PORTAL by SOWMYA MATHUKUMALLI B. Tech., SASTRA University, India, 2014
- Faculty of Computer and Informatics Engineering Online Job Portal Supervision Dr Eng Fady Ibrahim