DEPARTMENT OF INFORMATIONTECHNOLOGY

JNTU-GURAJADA VIZIANAGARAM COLLEGE OF ENGINEERING VIZIANAGARAM (A) VIZIANAGARAM

DJANGO FRAME WORK LAB

UNIVERSITY CARRIER SERVICE PLATFORM

DONE BY

S. MOHAN KUMAR	**24VV5A1273**
G. DHANA SAI	24VV5A1270

UNDER GUIDANCE OF

MRS.MADHUMITA CHANDA

DEPARTMENT OF INFORMATION TECHNOLOGY

JNTU-GURAJADA VIZIANAGARAM COLLEGE OF ENGINEERING
VIZIANAGARAM (A) VIZIANAGARAM

Regd.No : 24VV5A1273

CERTIFICATE

This is to certify that this is a bonafide record of practical work done by MR. S. MOHAN KUMAR of IInd B.Tech IInd Semester Class in DJANGO FRAME WORKLab during the year 2024-25.

No.of Tasks Completed and Certified:13

Lecture In-Charge : **Head of The Department**

Date:

DEPARTMENT OF INFORMATION TECHNOLOGY

JNTU-GURAJADA VIZIANAGARAM

COLLEGE OF ENGINEERING VIZIANAGARAM (A)

VIZIANAGARAM

Website:www.jntugvcev.edu.in

Subject Name: DJANGO FRAMEWORK Subject Code:R232212SE01

Year: 2025 Regulation: R23

COURSE OUTCOMES

NBA Subject Code	Course Outcomes	
	CO1	Design and build static as well as dynamic web pages and interactive web-based applications .
	CO2	Web development using Django framework.
	CO3	Analyze and create functional website in Django and deploy Django Web Application on Cloud .

CO-PO Mapping

Mapping of Course Outcomes (COs) with Program Outcomes (POs)

Course Outcomes		Program Outcomes (POs)													
		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
	CO1	3	1	3	1	3	1	1	1	2	3	2	1	3	3	2
	CO2	3	2	3	1	3	1	1	1	2	2	2	2	3	3	3
	CO3	2	3	3	3	3	2	2	2	2	3	3	3	3	3	3

Enter correlation levels 1,2 and 3 as defined below:

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High) If there is no correlation, put "-"

Signature of the Course Instructor

INDEX

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class : II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Understanding Django and Its Libraries
7. Date of Experiment : 13-12-2024
8. Date of Submission of Report : 20-12-2024

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty:

LIBRARIES

PYTHON LIBRARIES

1. Python Collections - Container Datatypes:

- **Purpose:** Provides specialized container datatypes that support efficient handling of data.
- *Key Types:*
  i. **List**: Ordered, mutable, allows duplicates.
  ii. **Tuple**: Ordered, immutable, allows duplicates.
  iii. **Set**: Unordered, no duplicates, fast membership testing.
  iv. **Dictionary**: Unordered, key-value pairs, fast lookups.

  Common Use: Data manipulation, storing and accessing collections of data in web apps (like user data or API responses).

2. Tinker

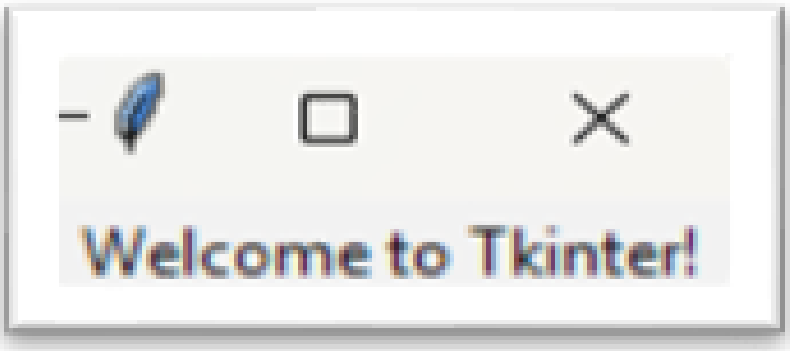
I. **Purpose**: Python's standard library for creating graphical user interfaces (GUIs).

II. **Keyfeatures:**
  i. Widgets: Buttons, labels, text boxes, etc.
  ii. Event handling: Respond to user interactions like clicks or key presses.
  iii. Simple layout management.

Code:

```
from tkinter import Tk, Label
# Create a window
root = Tk()
root.title("Hello Window")
# Add a label to display text
Label(root, text="Welcome to Tkinter!").pack()
# Run the application
root.mainloop()
```

Output:



3. Requests - HTTP Requests:

I. **Purpose**: Simplifies HTTP requests to interact with web APIs.

II. *Key Features:*

i. Send GET, POST, PUT, DELETE requests easily.

ii. Handle request parameters, headers, and cookies.

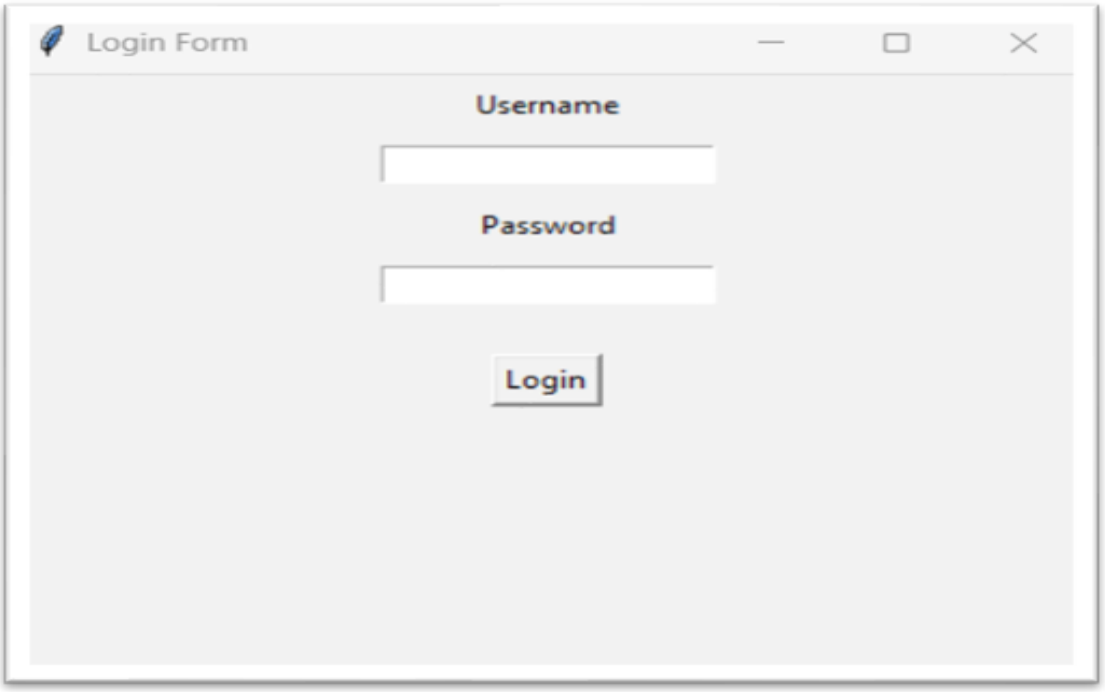
iii. Simple error handling and response handling.

Common Use: Interact with REST APIs, download content from the web**.**

Code:

```
from tkinter import Tk, Label, Entry, Button
def login():
    username = username_entry.get()
    password = password_entry.get()
    print(f"Username: {username}, Password: {password}")  # Placeholder for real login logic
# Create main window
root = Tk()
root.title("Login Form")
root.geometry("300x200")  # Set size of the window
# Username Label and Entry
Label(root, text="Username", font=('Arial', 10, 'bold')).pack(pady=(10, 0))
username_entry = Entry(root, width=30)
username_entry.pack(pady=(5, 10))
# Password Label and Entry
Label(root, text="Password", font=('Arial', 10, 'bold')).pack()
password_entry = Entry(root, show="*", width=30)
password_entry.pack(pady=(5, 10))
# Login Button
Button(root, text="Login", width=10, command=login).pack(pady=10)
# Run the application
root.mainloop()
```

Output:



3.Scrapy:

I. **Purpose**: An open-source web crawling framework for large-scale web scraping.

II. *Key Features:*

i. Fast, extensible, and asynchronous web scraping.

ii. Supports handling requests, data extraction, and storing results.

iii. Built-in handling for logging, retries, and sessions.

Common Use: Web crawling and scraping projects that require high performance.

4. BeautifulSoup4 - Web Scraping:

I. **Purpose**: Parses HTML and XML documents to extract data.

II. *Key Features:*

i. Easy navigation and searching within HTML.

ii. Supports different parsers like html.parser, lxml, and html5lib.

Common Use: Extract data from websites for analysis, e.g., for building data-driven application

Code:

```
import requests
from bs4 import BeautifulSoup
def scrape_quotes():
    base_url = "http://quotes.toscrape.com"
    next_page = "/"
    while next_page:
        response = requests.get(base_url + next_page)
        if response.status_code == 200:
            soup = BeautifulSoup(response.text, "html.parser")
            quotes = soup.find_all("span", class_="text")
            authors = soup.find_all("small", class_="author")
            for quote, author in zip(quotes, authors):
                print(f'"{quote.text}" - {author.text}\n')
            next_btn = soup.find("li", class_="next")
            next_page = next_btn.a["href"] if next_btn else None
        else:
            print(f"Failed to fetch webpage. Status code: {response.status_code}")
            break
# Run the scraper
scrape_quotes()
```

Output:

```
(myenv) C:\Users\Lenovo>python -u "c:\Users\Lenovo\import requests.py"

"“The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”" - Albert Einstein

"“It is our choices, Harry, that show what we truly are, far more than our abilities.”" - J.K. Rowling

"“There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle.”" - Albert Einstein

"“The person, be it gentleman or lady, who has not pleasure in a good novel, must be intolerably stupid.”" - Jane Austen

"“Imperfection is beauty, madness is genius and it's better to be absolutely ridiculous than absolutely boring.”" - Marilyn Monroe

"“Try not to become a man of success. Rather become a man of value.”" - Albert Einstein

"“It is better to be hated for what you are than to be loved for what you are not.”" - André Gide

"“I have not failed. I've just found 10,000 ways that won't work.”" - Thomas A. Edison

"“A woman is like a tea bag; you never know how strong it is until it's in hot water.”" - Eleanor Roosevelt

"“A day without sunshine is like, you know, night.”" - Steve Martin
```

5. Zappa:

I. **Purpose**: Deploy Python web applications to AWS Lambda and API Gateway.

II. *Key Features:*

i. Supports frameworks like Flask and Django for serverless deployments.

ii. Manages serverless architecture and deployment configurations.

Common Use: Build scalable, serverless web apps without maintaining servers.

6. Dash:

I. **Purpose**: Web application framework for building interactive data visualization applications.

II. *Key Features:*

i. Built on top of Flask, React, and Plotly.

ii. Integrates seamlessly with data science libraries (e.g., Pandas, Plotly).

- **Common Use**: Building dashboards and data-driven web applications.

Turbo Gears

I. **Purpose:** Full-stack web framework built on top of WSGI.

II. *Key Features:*

i. Modular: Mix and match components like SQLAlchemy, Genshi, and others.

ii. Focus on rapid development and scalability.

Common Use: Develop scalable, enterprise-level web applications.

7. CherryPy:

I. **Purpose**: Minimalistic web framework for building web applications.

II. *Key Features:*

i. Provides a simple and fast HTTP server.

ii. Handles routing, cookies, sessions, and file uploads.

Common Use: Building web applications with a lightweight framework.

Code:

```
import cherrypy
class HelloWorld:
    @cherrypy.expose  # Exposes this method as a web page
    def index(self):
        return "Hello, World! Welcome to CherryPy Web Server."
# Configure and start the CherryPy server
if __name__ == "__main__":
    cherrypy.quickstart(HelloWorld(), "/", config={
        "global": {
            "server.socket_host": "127.0.0.1",  # Localhost
            "server.socket_port": 8080,         # Port number
        }
    })
```

Output:

```
(myenv) C:\Users\Lenovo>python -u "c:\Users\Lenovo\import requests.py"
[10/Apr/2025:01:34:09] ENGINE Listening for SIGTERM.
[10/Apr/2025:01:34:09] ENGINE Bus STARTING
[10/Apr/2025:01:34:09] ENGINE Started monitor thread 'Autoreloader'.
[10/Apr/2025:01:34:09] ENGINE Serving on http://127.0.0.1:8080
[10/Apr/2025:01:34:09] ENGINE Bus STARTED
```

After run the server :-

8. Flask:

I. **Purpose**: Lightweight micro-framework for building web applications.

II. *Key Features:*

i. Simple to learn and use, but highly extensible.

ii. Supports extensions for database integration, form handling, authentication, etc.

Common Use: Small to medium web applications, APIs, or microservices**.**

Code:

```
from flask import Flask
app = Flask(__name__)
@app.route('/', methods=['GET'])
def hellouser():
   return "Hello, welcome to Flask!"
if __name__ == '__main__':
   app.run(debug=True)
```

```
(myenv) C:\Users\Lenovo> * Serving Flask app 'import requests'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a
    production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
* Debugger PIN: 134-121-940
```

Output:

After run the server:

9. Web2Py:

I. **Purpose**: Full-stack framework for rapid web application development.

II. *Key Features:*

i. Includes a web-based IDE for development.

ii. Built-in ticketing system and database integration.

Common Use: Enterprise web applications with minimal setup.

10. Bottle:

I. **Purpose**: Simple and lightweight WSGI micro-framework.

II. *Key Features:*

i. Single-file framework, minimalistic, and fast.

ii. No dependencies, supports routing, templates, and form handling.

Common Use: Small web applications, APIs, and prototypes.

Code:

```
from bottle import Bottle, run
app = Bottle()
@app.route('/')
def home():
    return "Hello, welcome to Bottle framework!"
if __name__ == '__main__':
    run(app, host='localhost', port=8080, debug=True)
```

output:

```
(myenv) C:\Users\Lenovo>python -u "c:\Users\Lenovo\import requests.py"
Bottle v0.13.2 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.
```

After run the server

11. Falcon:

I. **Purpose**: High-performance framework for building APIs.

II. *Key Features:*

i. Focuses on speed and minimalism.

ii. Supports RESTful API development and is optimized for large-scale deployments.

Common Use: Building fast, high-performance APIs.

12. CubicWeb:

I. **Purpose**: Web application framework based on an entity-relation model.

II. *Key Features:*

i. Uses a highly modular architecture for development.

ii. Focus on building web apps with rich data models.

Common Use: Semantic web applications or data-driven web apps.

13. Quixote:

I. **Purpose**: A web framework designed for simplicity and scalability.

II. *Key Features:*

i. Full support for Python's object-oriented programming.

ii. Easily extensible, with minimalistic core.

Common Use: Scalable and customizable web applications.

14. Pyramid:

I. **Purpose**: Full-stack web framework that can scale from simple to complex applications.

II. *Key Features:*

i. Highly flexible with support for routing, templating, authentication, and authorization.

ii. Allows for small and large applications, with fine-grained control.

Common Use: Building large, enterprise-grade web applications and REST APIs.

SUMMARY:

i. **Flask**, **Django**, **Pyramid**: Popular web frameworks, each offering flexibility and scalability.

ii. **Scrapy**, **BeautifulSoup4**: Specialized for web scraping and data extraction.

iii. **Requests**, **Zappa**, **Dash**: Tools for making HTTP requests, serverless apps, and interactive data visualizations.

iv. **Tkinter**, **Bottle**, **CherryPy**: Libraries for building lightweight desktop and web applications.

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class : II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Introduction to Django Framework
7. Date of Experiment : 20-12-2024
8. Date of Submission of Report : 27-12-2024

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty

Django: A Web Framework for Python

Django: Django is a **high-level Python web framework** that allows developers to build secure, scalable, and maintainable web applications **quickly and efficiently**. It follows the **Model-View-Template (MVT)** architectural pattern.

Key Features of Django:

I. Fast Development **–** Comes with built-in features like authentication, database management, and an admin panel**.**
II. Scalability – Suitable for small projects to enterprise-level applications.
III. Security – Protects against common security threats (SQL Injection, CSRF, XSS, etc.).
IV. ORM (Object-Relational Mapper) – Allows database interaction using Python instead of SQL.
V. Built-in Admin Panel – Auto-generates an admin interface for managing data.
VI. Reusable App-Developers can create modular and reusable components.

Django's MVT Architecture:

i. Model (M) – Handles database interactions (e.g., User, Booking).
ii. View (V) – Manages business logic and connects models to templates.
iii. Template (T) – Renders HTML pages dynamically.

Example MVT Folder Structure in Django

```
JobPortal/
|── jobs/
|   |── __pycache__/
|   |── media/
|   |── migrations/
|   |── static/
|   |── templates/
|   |── __init__.py
|   |── admin.py
|   |── apps.py
|   |── models.py
|   |── tests.py
|   |── urls.py
|   |── views.py
|── manage.py
|── db.sqlite3
|── requirements.txt
|── README.md
|── Quick Sort Algorithm.txt
|── sqlite3 (Application)
|── Profile
|── JobPortal/
|   |── __pycache__/
|   |── __init__.py
|   |── asgi.py
|   |── settings.py
|   |── urls.py
|   |── wsgi.py
```

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class : II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Step-by-Step Guide to Installing Django
7. Date of Experiment : 27-12-2024
8. Date of Submission of Report : 03-01-2025

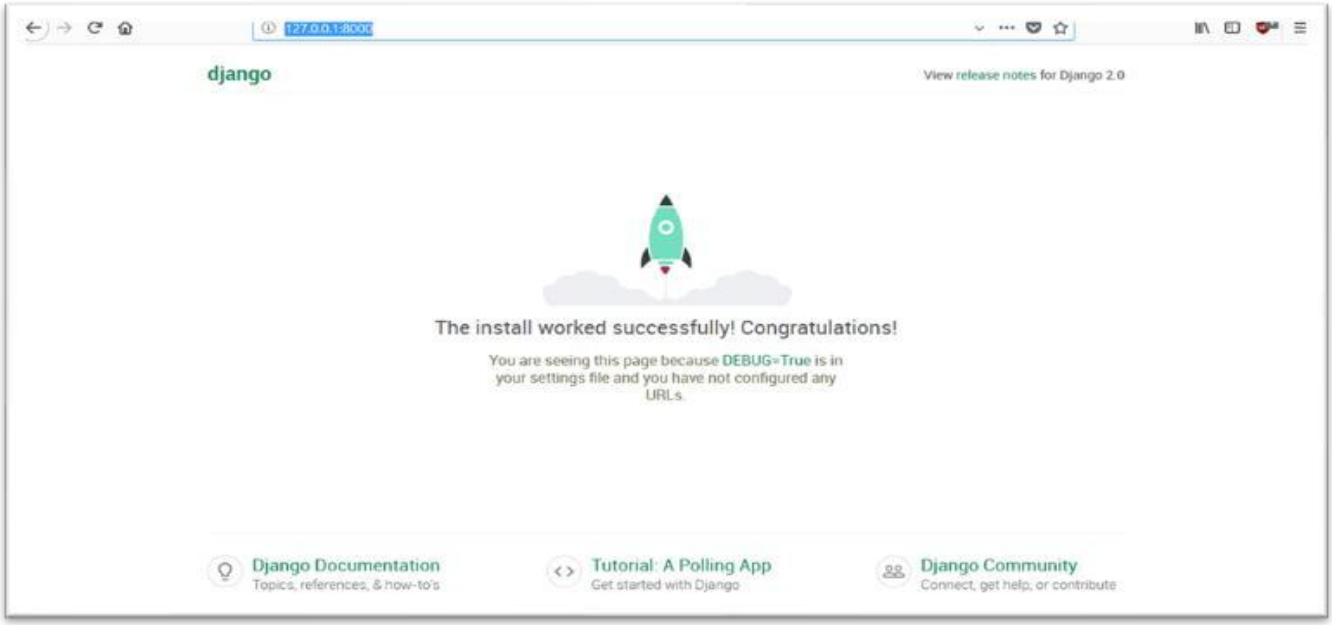
S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty

Steps to Create Django Project:

Step-1 : Checking the installation & version of Python & PIP
python --version pip --version
Step-2 : Installation of Virtual Environment
pip install virtualenvwrapper-win
Step-3 : Creation of Virtual Environment
mkvirtualenv (name)
Step-4 : Installation of Django in Virtual environment
pip install Django
Step-5 : Create a folder to store all the projects
mkdir proj_folder_name
Step-6 : Start new_project
django-admin startproject project_name
django-admin startapp app_name
Step-7 : Run the server
python manage.py runserver
Step-8 : Open the browser and check the homepage of Django..

Output:



DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class : II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Linking Views and URL Configurations
7. Date of Experiment : 03-01-2025
8. Date of Submission of Report : 24-01-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty

Connecting Views and URLs:

Set Up URLs:

Project-level URL Configuration:

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
path('admin/', admin.site.urls), path('',
include('myapp1.urls')), # Include app URLs
 ]
```

App-level URLConfiguration:

```
from django.urls import path
from .views import home
urlpatterns = [ path('', home,
name='home'), ]
```

Create a Sample View:

```
django.http import HttpResponse
home(request):
        return HttpResponse("Welcome to My Django App!")
```

Run Migrations:

python manage.py migrate

Run the Server and Test:

python manage.py runserver

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class : II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Exploring Django Views
7. Date of Experiment : 24-01-2025
8. Date of Submission of Report : 24-01-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty

Views.py:

In Django, views.py is the file where you define functions or classes that handle requests and return responses. Views act as the logic layer of a Django web application, controlling how data is processed and which HTML templates are displayed.

Code:

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login, logout
from django.contrib import messages
from datetime import date
from django.views.decorators.http import require_POST  #  Add this line
from .models import *

def index(request):
    return render(request, "index.html")

def user_login(request):
    if request.user.is_authenticated:
        return redirect("/")
    else:
        if request.method == "POST":
            username = request.POST['username']
            password = request.POST['password']
            user = authenticate(username=username, password=password)

            if user is not None:
                user1 = Applicant.objects.get(user=user)
                if user1.type == "applicant":
                    login(request, user)
                    return redirect("/user_homepage")
            else:
                thank = True
                return render(request, "user_login.html", {"thank":thank})
    return render(request, "user_login.html")

def user_homepage(request):
    if not request.user.is_authenticated:
        return redirect('/user_login/')
    applicant = Applicant.objects.get(user=request.user)
    if request.method == "POST":
        email = request.POST['email']
        first_name = request.POST['first_name']
        last_name = request.POST['last_name']
```

```
        phone = request.POST['phone']
        gender = request.POST['gender']

        applicant.user.email = email
        applicant.user.first_name = first_name
        applicant.user.last_name = last_name
        applicant.phone = phone
        applicant.gender = gender
        applicant.save()
        applicant.user.save()

        try:
            image = request.FILES['image']
            applicant.image = image
            applicant.save()
        except:
            pass
        alert = True
        applications = Application.objects.filter(applicant=applicant)
        return render(request, "user_homepage.html", {
            'alert': alert,
            'applications': applications,
            'applicant': applicant,  # <-- Add this line
        })

    applications = Application.objects.filter(applicant=applicant)
    return render(request, 'user_homepage.html', {
        'applications': applications,
        'applicant': applicant,  # <-- Add this line
    })

def all_jobs(request):
    jobs = Job.objects.all().order_by('-start_date')
    applicant = Applicant.objects.get(user=request.user)
    apply = Application.objects.filter(applicant=applicant)
    data = []
    for i in apply:
        data.append(i.job.id)
    return render(request, "all_jobs.html", {'jobs':jobs, 'data':data})

def job_detail(request, myid):
    job = Job.objects.get(id=myid)
    return render(request, "job_detail.html", {'job':job})

def job_apply(request, myid):
    if not request.user.is_authenticated:
```

```
        return redirect("/user_login")
    applicant = Applicant.objects.get(user=request.user)
    job = Job.objects.get(id=myid)
    date1 = date.today()
    if job.end_date < date1:
        closed = True
        return render(request, "job_apply.html", {'closed': closed})
    elif job.start_date > date1:
        notopen = True
        return render(request, "job_apply.html", {'notopen': notopen})
    else:
        if request.method == "POST":
            resume = request.FILES.get('resume')
            if resume:
                Application.objects.create(
                    job=job,
                    applicant=applicant,
                    resume=resume,
                    company=job.company  # 🔑 Add this line!
                )
                alert = True
                return render(request, "job_apply.html", {'alert': alert})
            else:
                error = "Resume file is required."
                return render(request, "job_apply.html", {'error': error, 'job': job})
    return render(request, "job_apply.html", {'job': job})

def signup(request):
    if request.method=="POST":
        username = request.POST['username']
        first_name=request.POST['first_name']
        last_name=request.POST['last_name']
        password1 = request.POST['password1']
        password2 = request.POST['password2']
        phone = request.POST['phone']
        gender = request.POST['gender']
        image = request.FILES['image']

        if password1 != password2:
            messages.error(request, "Passwords do not match.")
            return redirect('/signup')

        user = User.objects.create_user(first_name=first_name, last_name=last_name,
username=username, password=password1)
        applicants = Applicant.objects.create(user=user, phone=phone, gender=gender,
image=image, type="applicant")
```

```
        user.save()
        applicants.save()
        return render(request, "user_login.html")
    return render(request, "signup.html")

def company_signup(request):
    if request.method == "POST":
        username = request.POST.get('username', '').strip()
        email = request.POST.get('email', '').strip()
        first_name = request.POST.get('first_name', '').strip()
        last_name = request.POST.get('last_name', '').strip()
        password1 = request.POST.get('password1', '')
        password2 = request.POST.get('password2', '')
        phone = request.POST.get('phone', '').strip()
        gender = request.POST.get('gender', '')
        company_name = request.POST.get('company_name', '').strip()
        image = request.FILES.get('image')

        # Check for missing required fields
        if not all([username, email, first_name, last_name, password1, password2, phone, gender,
company_name, image]):
            messages.error(request, "Please fill in all required fields.")
            return redirect('/company_signup')

        if password1 != password2:
            messages.error(request, "Passwords do not match.")
            return redirect('/company_signup')

        # Optional: Check if username or email already exists
        if User.objects.filter(username=username).exists():
            messages.error(request, "Username already taken.")
            return redirect('/company_signup')
        if User.objects.filter(email=email).exists():
            messages.error(request, "Email already registered.")
            return redirect('/company_signup')

        # Create user and company
        user = User.objects.create_user(
            first_name=first_name,
            last_name=last_name,
            email=email,
            username=username,
            password=password1
        )
        company = Company.objects.create(
            user=user,
```

```
            phone=phone,
            gender=gender,
            image=image,
            company_name=company_name,
            type="company",
            status="pending"
        )
        user.save()
        company.save()
        messages.success(request, "Registration successful. Please log in.")
        return redirect('/company_login')

    return render(request, "company_signup.html")

def company_login(request):
    if request.method == "POST":
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(username=username, password=password)

        if user is not None:
            user1 = Company.objects.get(user=user)
            if user1.type == "company" and user1.status != "pending":
                login(request, user)
                return redirect("/company_homepage")
        else:
            alert = True
            return render(request, "company_login.html", {"alert":alert})
    return render(request, "company_login.html")

def company_homepage(request):
    if not request.user.is_authenticated:
        return redirect("/company_login")
    company = Company.objects.get(user=request.user)
    if request.method=="POST":
        email = request.POST['email']
        first_name=request.POST['first_name']
        last_name=request.POST['last_name']
        phone = request.POST['phone']
        gender = request.POST['gender']

        company.user.email = email
        company.user.first_name = first_name
        company.user.last_name = last_name
        company.phone = phone
        company.gender = gender
```

```
        company.save()
        company.user.save()

        try:
            image = request.FILES['image']
            company.image = image
            company.save()
        except:
            pass
        alert = True
        return render(request, "company_homepage.html", {'alert':alert})
    return render(request, "company_homepage.html", {'company':company})



def pending_companies(request):
    if not request.user.is_authenticated:
        return redirect("/admin_login")
    companies = Company.objects.filter(status="pending")
    return render(request, "pending_companies.html", {'companies':companies})

def change_status(request, myid):
    if not request.user.is_authenticated:
        return redirect("/admin_login")
    company = Company.objects.get(id=myid)
    if request.method == "POST":
        status = request.POST['status']
        company.status=status
        company.save()
        alert = True
        return render(request, "change_status.html", {'alert':alert})
    return render(request, "change_status.html", {'company':company})

def accepted_companies(request):
    if not request.user.is_authenticated:
        return redirect("/admin_login")
    companies = Company.objects.filter(status="Accepted")
    return render(request, "accepted_companies.html", {'companies':companies})

def rejected_companies(request):
    if not request.user.is_authenticated:
        return redirect("/admin_login")
    companies = Company.objects.filter(status="Rejected")
    return render(request, "rejected_companies.html", {'companies':companies})
```

```
def all_companies(request):
    if not request.user.is_authenticated:
        return redirect("/admin_login")
    companies = Company.objects.all()
    return render(request, "all_companies.html", {'companies':companies})

def delete_company(request, myid):
    if not request.user.is_authenticated:
        return redirect("/admin_login")
    company = User.objects.filter(id=myid)
    company.delete()
    return redirect("/all_companies")
```

In Django, the urls.py file is responsible for mapping URLs to views. It acts as the router of your application, directing user requests to the correct function in views.py.

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class : II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Setting Up App-Level URLs
7. Date of Experiment : 24-01-2025
8. Date of Submission of Report : 31-01-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty

Creating urls.py in a Django App:

Each Django app should have its own urls.py file to define app-specific routes.

Steps to Create urls.py in a Django App

i. Inside your Django app folder (myapp1), create a file named **urls.py**.
ii. Define URL patterns to map URLs to views.

App Urls.Py :

```
from django.urls import path
from . import views
urlpatterns = [
    path("", views.index, name="index"),
    # User
    path("user_login/", views.user_login, name="user_login"),
    path("signup/", views.signup, name="signup"),
    path("user_homepage/", views.user_homepage, name="user_homepage"),
    path("logout/", views.Logout, name="logout"),
    path("all_jobs/", views.all_jobs, name="all_jobs"),
    path("job_detail/<int:myid>/", views.job_detail, name="job_detail"),
    path("job_apply/<int:myid>/", views.job_apply, name="job_apply"),
    # Company
    path("company_signup/", views.company_signup, name="company_signup"),
    path("company_login/", views.company_login, name="company_login"),
    path("company_homepage/", views.company_homepage, name="company_homepage"),
    path("add_job/", views.add_job, name="add_job"),
    path("job_list/", views.job_list, name="job_list"),
    path('delete-job/<int:id>/', views.delete_job, name='delete_job'),
    path("edit_job/<int:myid>/", views.edit_job, name="edit_job"),
    path("company_logo/<int:myid>/", views.company_logo, name="company_logo"),
    path('all_applicants/', views.all_applicants, name='all_applicants'),
    path("accept_applicant/<int:myid>/", views.accept_applicant, name="accept_applicant"),
    path("reject_applicant/<int:myid>/", views.reject_applicant, name="reject_applicant"),
    # Admin
    path("admin_login/", views.admin_login, name="admin_login"),
    path("view_applicants/", views.view_applicants, name="view_applicants"),
   path("delete_applicant/<int:myid>/", views.delete_applicant, name="delete_applicant"),
    path("pending_companies/", views.pending_companies, name="pending_companies"),
    path("accepted_companies/", views.accepted_companies, name="accepted_companies"),
    path("rejected_companies/", views.rejected_companies, name="rejected_companies"),
]
```

Connecting App urls.py to Project urls.py:

To use the app's URLs, include them in the project-level urls.py (myproject/urls.py)

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('jobs.urls')),

]  + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class : II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Working With Templates
7. Date of Experiment : 31-01-2025
8. Date of Submission of Report : 17-02-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty

Templates:

Templates in Django are HTML files that display dynamic content. They separate the frontend (UI) from the backend logic, following the MVT (Model-View-Template) architecture.

Where to Store Templates?

By default, Django looks for templates in a folder named **templates/** inside your app.

```
JobPortal/
|── jobs/
|    |── __pycache__/
|    |── media/
|    |── migrations/
|    |── static/
|    |── templates/
|    |── __init__.py
|    |── admin.py
|    |── apps.py
|    |── models.py
|    |── tests.py
|    |── urls.py
|    |── views.py
|── manage.py
|── db.sqlite3
|── requirements.txt
|── README.md
|── Quick Sort Algorithm.txt
|── sqlite3 (Application)
|── Profile
|── JobPortal/
|    |── __pycache__/
|    |── __init__.py
|    |── asgi.py
|    |── settings.py
|    |── urls.py
|    |── wsgi.py
```

Admin_navbar.html :

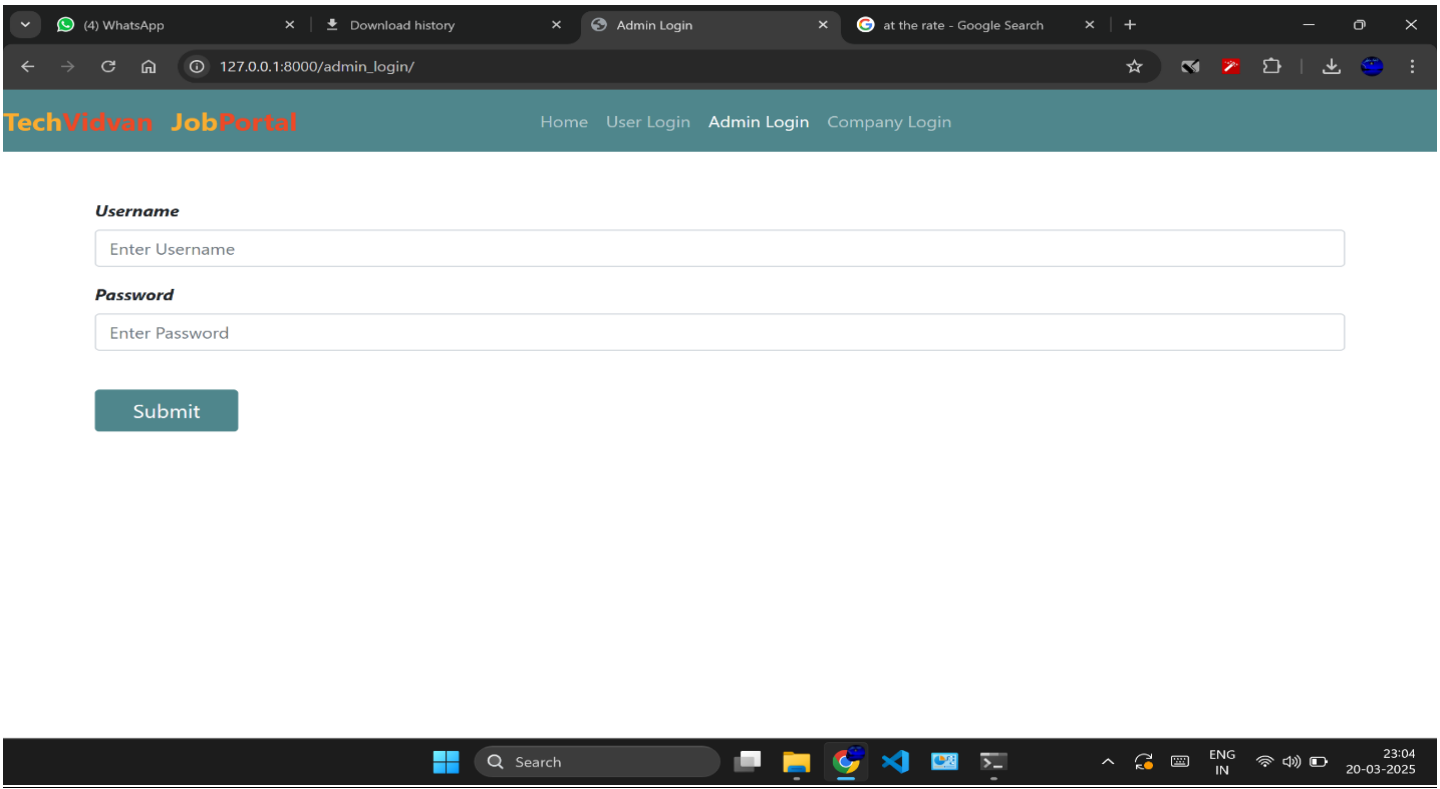
```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
```

```
   integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">

  <link rel="stylesheet"
href="https://cdn.datatables.net/1.10.25/css/jquery.dataTables.min.css">
  <link rel="stylesheet"
href="https://cdn.datatables.net/buttons/1.7.1/css/buttons.dataTables.min.css">

  <title>{% block title %}{% endblock %}</title>
  <style>
   .navbar {
     background-color: #4f868c;
   }
   .head1 {
     color: #FBAD30;
     font-size: 25px;
     font-weight: bold;
     height: 50px;
   }

   .head2 {
     color: #EF4926;
     font-size: 25px;
     font-weight: bold;
   }
  </style>
  {% block css %}
  {% endblock %}
</head>

<body>
  <nav class="navbar navbar-expand-lg navbar-dark">
     <a class="navbar-brand" href="/"><span class="head1">Tech</span><span
class="head2">Vidvan</span></a>
     <a class="navbar-brand" href="#"><span class="head1">Job</span><span
class="head2">Portal</span></a>
     <div class="container w-50">
     <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarSupportedContent"
       aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle
navigation">
       <span class="navbar-toggler-icon"></span>
     </button>
     <div class="collapse navbar-collapse" id="navbarSupportedContent">
       <ul class="navbar-nav me-auto mb-2 mb-lg-0">
```

```
        <div class="collapse navbar-collapse" id="navbarNavDarkDropdown">
          <ul class="navbar-nav">
            <li class="nav-item dropdown">
              <a class="nav-link active dropdown-toggle" href="#"
id="navbarDarkDropdownMenuLink" role="button" data-bs-toggle="dropdown" aria-
expanded="false">
                Company
              </a>
              <ul class="dropdown-menu dropdown-menu-dark" aria-
labelledby="navbarDarkDropdownMenuLink">
                <li><a class="dropdown-item" href="/accepted_companies/">Accepted</a></li>
                <li><a class="dropdown-item" href="/pending_companies/">Pending</a></li>
                <li><a class="dropdown-item" href="/rejected_companies/">Rejected</a></li>
                <li><a class="dropdown-item" href="/all_companies/">All Companies</a></li>
              </ul>
            </li>
          </ul>
        </div>
        <li class="nav-item">
          <a class="nav-link {% block view_applicants %} {% endblock %}"
href="/view_applicants/">View Applicants</a>
        </li>
        <li class="nav-item">
          <a class="nav-link active" href="/logout/">Logout</a>
        </li>
       </ul>
       <a class="navbar-brand" style="font-weight: bold; font-family: 'Times New Roman',
Times, serif;" href="#">Welcome {{request.user.get_full_name}}</a>
     </div>
     </div>
  </nav>

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
    crossorigin="anonymous"></script>

  <script src="https://code.jquery.com/jquery-3.5.1.js"></script>
  <script src="https://cdn.datatables.net/1.10.25/js/jquery.dataTables.min.js"></script>
  <script src="https://cdn.datatables.net/buttons/1.7.1/js/dataTables.buttons.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jszip/3.1.3/jszip.min.js"></script>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/pdfmake/0.1.53/pdfmake.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/pdfmake/0.1.53/vfs_fonts.js"></script>
  <script src="https://cdn.datatables.net/buttons/1.7.1/js/buttons.html5.min.js"></script>
```

```
 {% block body %}
 {% endblock %}
</body>
{% block js %}
<script>
 $(document).ready(function () {
  $('#example').DataTable({
    dom: 'Bfrtip',
    buttons: [
      'copyHtml5',
      'excelHtml5',
      'csvHtml5',
      'pdfHtml5'
    ]
  });
 });
</script>
{% endblock %}

</html>
```

OUTPUT:



Role of admin_navbar.html in Django:

The admin_navbar.html template is used to display the Admin Dashboard for users with admin privileges. This page provides management options for the administrator, such as:

i. Viewing all companies and users
ii. Managing company access (accepting/rejecting of companies)
iii. Managing users (job seekers)
iv. Approving or rejecting jobs lists

User_homepage.html

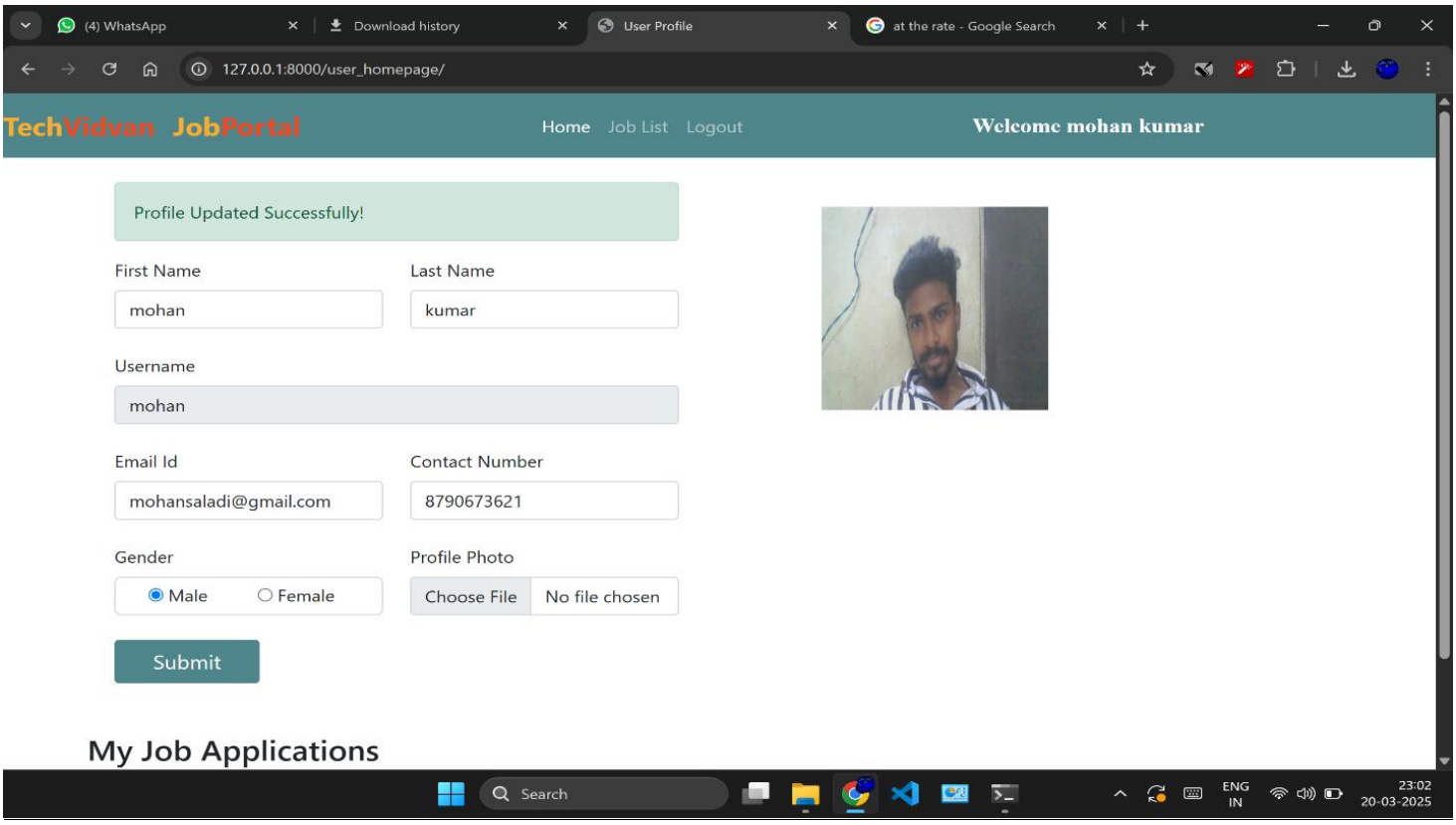
```
{% extends 'user_navbar.html' %}
{% block title %} User Profile {% endblock %}
{% block home %} active {% endblock %}
{% block css %}
{% endblock %}
{% block body %}
<div class="container">
  <div class="row">
    <div class="col-sm-6">

      <form class="container mt-4" method="POST" enctype="multipart/form-data">
        {% csrf_token %}

        {% if alert %}
        <div class="alert alert-success mt-3">
          Profile Updated Successfully!
        </div>
        {% endif %}

        {% if error %}
        <div class="alert alert-danger mt-3">
          {{ error }}
        </div>
        {% endif %}

        <div class="row">
          <div class="form-group col-md-6">
            <label>First Name</label>
            <input type="text" class="form-control mt-2" name="first_name"
id="first_name"
              value="{{ applicant.user.first_name }}" required>
          </div>
          <div class="form-group col-md-6">
            <label>Last Name</label>
            <input type="text" class="form-control mt-2" name="last_name"
id="last_name"
```

```
                    value="{{ applicant.user.last_name }}" required>
                </div>
            </div>

            <div class="row mt-4">
                <div class="form-group col-md-12">
                    <label>Username</label>
                    <input type="text" class="form-control mt-2" name="username"
id="username"
                    value="{{ applicant.user.username }}" readonly>
                </div>
            </div>

            <div class="row mt-4">
                <div class="form-group col-md-6">
                    <label>Email Id</label>
                    <input type="email" class="form-control mt-2" name="email" id="email"
value="{{ applicant.user.email }}" required>
                </div>
                <div class="form-group col-md-6">
                    <label>Contact Number</label>
                    <input type="tel" class="form-control mt-2" name="phone" id="phone"
value="{{ applicant.phone }}" required>
                </div>
            </div>

            <div class="row mt-4">
                <div class="form-group col-md-6">
                    <label>Gender</label>
                    <div class="mt-2" style="border: 1px solid lightgrey; padding: 5px; border-
radius: 6px;">
                        <div class="form-check form-check-inline">
                            <input type="radio" id="male" class="custom-control-input"
name="gender" value="Male" {% if applicant.gender == "Male" %}checked{% endif %}>
                            <label for="male" class="custom-control-label">Male</label>
                        </div>
                        <div class="form-check form-check-inline">
                            <input type="radio" id="female" class="custom-control-input"
name="gender" value="Female" {% if applicant.gender == "Female" %}checked{% endif
%}>
                            <label for="female" class="custom-control-label">Female</label>
                        </div>
                    </div>
                </div>
                <div class="form-group col-md-6">
                    <label>Profile Photo</label>
```

```
                    <input type="file" class="form-control mt-2" name="image" id="image"
accept="image/*">
                </div>
            </div>

            <input type="submit" value="Submit" class="btn mt-4" id="submitBtn"
style="background-color: #4f868c; color: white; font-size: larger; width: 8rem;">
        </form>

    </div>
    <div class="col-sm-4 mt-5 text-center">
        {% if applicant.image %}
            <img src="{{ applicant.image.url }}" alt="Profile Photo" width="200px"
height="200px">
        {% else %}
            <img src="/static/images/default-profile.png" alt="Default Profile Photo"
width="200px" height="200px">
        {% endif %}
    </div>
</div>

<!-- Applications Status Section -->
<div class="mt-5">
    <h3>My Job Applications</h3>
    <table class="table table-bordered mt-3">
        <thead>
            <tr>
                <th>#</th>
                <th>Job Title</th>
                <th>Applied On</th>
                <th>Status</th>
            </tr>
        </thead>
        <tbody>
            {% for application in applications %}
            <tr>
                <td>{{ forloop.counter }}</td>
                <td>{{ application.job.title }}</td>
                <td>{{ application.apply_date|date:"M d, Y" }}</td>
                <td>
                    {% if application.status == "Accepted" %}
                        <span class="badge bg-success">Accepted</span>
                    {% elif application.status == "Rejected" %}
                        <span class="badge bg-danger">Rejected</span>
                    {% else %}
                        <span class="badge bg-secondary">Pending</span>
```

```
                {% endif %}
              </td>
            </tr>
            {% empty %}
            <tr>
              <td colspan="4" class="text-center">No applications found.</td>
            </tr>
            {% endfor %}
          </tbody>
        </table>
    </div>
</div>
{% endblock %}

{% block js %}
<script>
    // Disable submit button after submission to prevent double clicks
    document.querySelector('form').addEventListener('submit', function() {
        document.getElementById('submitBtn').disabled = true;
    });
</script>
{% endblock %}
```

OUTPUT:



Role of user_homepage.html in Django:

The user_homepage.html template lets users manage their profile and view their job applications.

i. Users can update their name, email, phone number, gender, and profile photo.
ii. Shows the user's current profile photo or a default one if not uploaded.
iii. Displays success or error messages after form submission.
iv. Lists all job applications with job title, date, and status.
v. Shows a message if no applications are found.
vi. Disables the submit button after clicking to avoid double submission.
vii. Uses a common layout by extending the navbar template.

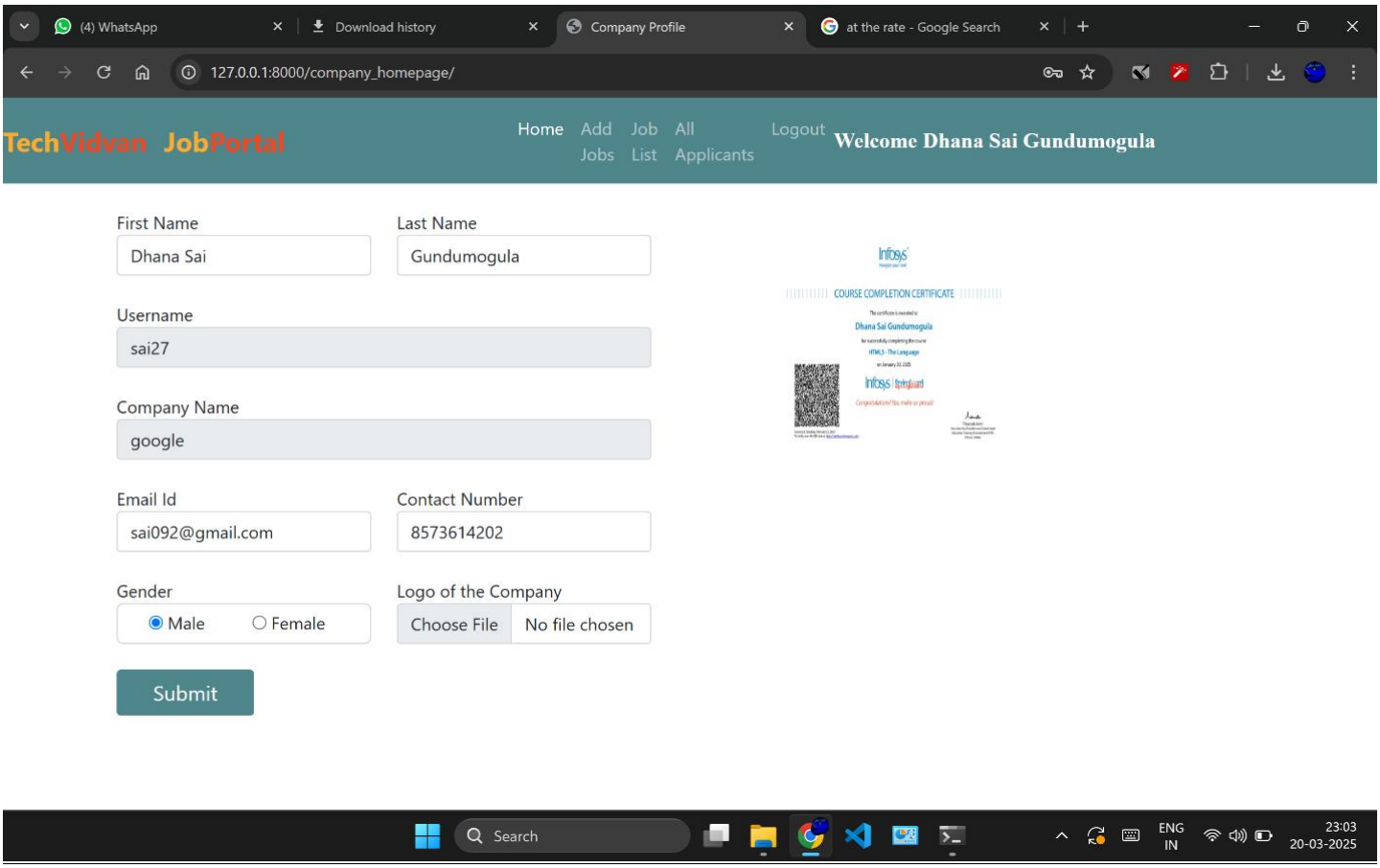
Company_homepage.html:

```
{% extends 'company_navbar.html' %}

{% block title %} Company Profile {% endblock %}
{% block home %} active {% endblock %}
{% block css %}
{% endblock %}
{% block body %}

<div class="container">
   <div class="row">
      <div class="col-sm-6">

         <form class="container mt-4" method="POST" enctype="multipart/form-data">
            {% csrf_token %}

            <div class="row">
               <div class="form-group col-md-6">
                  <label>First Name</label>
                  <input type="text" class="form-control" name="first_name"
id="first_name"
                     value="{{company.user.first_name}}" required>
               </div>
               <div class="form-group col-md-6">
                  <label>Last Name</label>
                  <input type="text" class="form-control" name="last_name" id="last_name"
                  value="{{company.user.last_name}}" required>
               </div>
            </div>

            <div class="row mt-4">
               <div class="form-group col-md-12">
```

```
                <label>Username</label>
                <input type="text" class="form-control" name="username" id="username"
                    value="{{company.user.username}}" readonly>
            </div>
        </div>

        <div class="row mt-4">
            <div class="form-group col-md-12">
                <label>Company Name</label>
                <input type="text" class="form-control" name="company_name"
id="company_name"
                    value="{{company.company_name}}" readonly>
            </div>
        </div>

        <div class="row mt-4">
            <div class="form-group col-md-6">
                <label>Email Id</label>
                <input type="email" class="form-control" name="email" id="email"
value="{{company.user.email}}" required>
            </div>
            <div class="form-group col-md-6">
                <label>Contact Number</label>
                <input type="tel" class="form-control" name="phone" id="phone"
value="{{company.phone}}" required>
            </div>
        </div>

        <div class="row mt-4">
            <div class="form-group col-md-6">
                <label>Gender</label>
                {% if company.gender == "Male" %}
                <div style="border: 1px solid lightgrey; padding: 5px; border-radius: 6px;">
                    <div class="form-check form-check-inline">
                        <input type="radio" class="custom-control-input" name="gender"
value="Male" checked>
                        <label for="male" class="custom-control-label">Male</label>
                    </div>
                    <div class="form-check form-check-inline">
                        <input type="radio" class="custom-control-input" name="gender"
                            value="Female">
                        <label for="female" class="custom-control-label">Female</label>
                    </div>
                </div>
                {% else %}
                <div style="border: 1px solid lightgrey; padding: 5px; border-radius: 6px;">
```

```
                    <div class="form-check form-check-inline">
                        <input type="radio" class="custom-control-input" name="gender"
value="Male">
                        <label for="male" class="custom-control-label">Male</label>
                    </div>
                    <div class="form-check form-check-inline">
                        <input type="radio" class="custom-control-input" name="gender"
                            value="Female" checked>
                        <label for="female" class="custom-control-label">Female</label>
                    </div>
                </div>
                {% endif %}
            </div>
            <div class="form-group col-md-6">
                <label>Logo of the Company</label>
                <input type="file" class="form-control" name="image" id="image">
            </div>
        </div>

        <input type="submit" value="Submit" class="btn mt-4" style="background-color:
#4f868c; color: white; font-size: larger; width: 8rem;">
    </form>

    </div>
    <div class="col-sm-4 mt-5 text-center">
        <img src="{{company.image.url}}" alt="" width="200px" height="200px">
    </div>
  </div>
</div>
{% endblock %}
{% block js %}
<script>
  {% if alert %}
  alert("Profile Updated Successfully")
  window.location = "/company_homepage"
  {% endif %}
</script>
{% endblock %}
```

OUTPUT:



Role of company_homepage.html in Django:

The company_homepage.html template allows company users to view and update their profile in the Job Portal System.

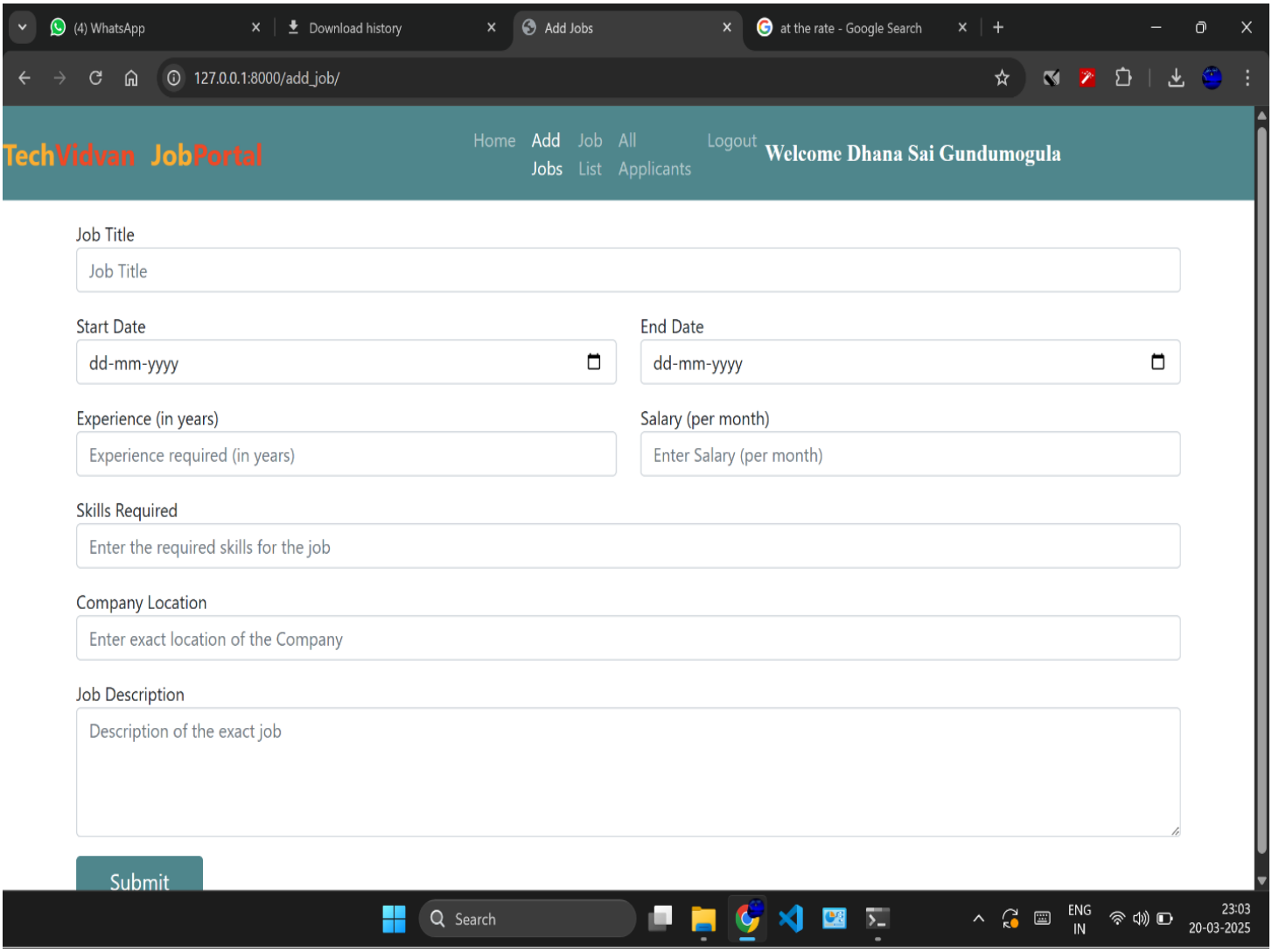
i. Provides a form to update first name, last name, email, phone number, gender, and company logo.
ii. Displays the company's current logo.
iii. Shows the company name and username (read-only).
iv. Shows a success alert when the profile is updated.
v. Disables form resubmission by redirecting after a successful update.
vi. Uses a consistent layout by extending the company navbar.

Add_job.html:

```
{% extends 'company_navbar.html' %}

{% block title %} Add Jobs {% endblock %}
{% block add_job %} active {% endblock %}
{% block css %}
{% endblock %}
{% block body %}
<form class="container mt-3" method="POST" enctype="multipart/form-data">
   {% csrf_token %}

   <div class="row">
      <div class="form-group col-md-12">
         <label>Job Title</label>
         <input type="text" class="form-control" name="job_title" id="job_title"
placeholder="Job Title" required>
      </div>
   </div>

   <div class="row mt-3">
      <div class="form-group col-md-6">
         <label>Start Date</label>
         <input type="date" class="form-control" name="start_date" id="start_date"
placeholder="Start Date" required>
      </div>
      <div class="form-group col-md-6">
         <label>End Date</label>
         <input type="date" class="form-control" name="end_date" id="end_date"
placeholder="End Date" required>
      </div>
   </div>

   <div class="row mt-3">
      <div class="form-group col-md-6">
         <label>Experience (in years)</label>
         <input type="text" class="form-control" name="experience" id="experience"
placeholder="Experience required (in years)" required>
      </div>
      <div class="form-group col-md-6">
         <label>Salary (per month)</label>
         <input type="tel" class="form-control" name="salary" id="salary"
placeholder="Enter Salary (per month)" required>
      </div>
   </div>
```

```
    <div class="row mt-3">
        <div class="form-group col-md-12">
            <label>Skills Required</label>
            <input type="text" class="form-control" name="skills" id="skills"
placeholder="Enter the required skills for the job" required>
        </div>
        </div>

    <div class="row mt-3">
        <div class="form-group col-md-12">
            <label>Company Location</label>
            <input type="text" class="form-control" name="location" id="location"
placeholder="Enter exact location of the Company" required>
        </div>
    </div>

    <div class="row mt-3">
        <div class="form-group col-md-12">
            <label>Job Description</label>
           <textarea name="description" id="description" class="form-control" cols="30"
rows="4" placeholder="Description of the exact job" required></textarea>
        </div>
    </div>
<input type="submit" value="Submit" class="btn mt-3" style="background-color: #4f868c;
color: white; font-size: larger; width: 8rem;">
</form>
{% endblock %}
{% block js %}
<script>
    {% if alert %}
    alert("Job added successfully.")
    window.location = "/job_list"
    {% endif %}
</script>
{% endblock %}
```

OUTPUT:



Role of add_job.html in Django:

The add_job.html template allows company users to post new job openings on the portal.

i. Provides a form to enter job details like title, dates, experience, salary, skills, location, and description.

ii. Ensures all fields are required to prevent incomplete job entries.

iii. Submits data securely using CSRF token and POST method.

iv. Shows a success alert and redirects to the job list after successful submission.

v. Uses a consistent layout by extending the company navbar template.

index.html

```
{% extends 'basic.html' %}

{% block title %} Job Portal {% endblock %}
{% block home %} active {% endblock %}
{% block css %}
{% endblock %}
{% block body %}
{% load static %}
<div class="container-fluid mt-2">
   <img src="{% static 'Recruitment Vector.png' %}" alt="" width="100%" height="450px">
</div>
   <div class="container-fluid mt-2" style="background-color: #4f868c; width: 100%;">
      <div class="row">
         <div class="col-md-4">
            <h3 style="color: aliceblue;">Contact Us</h3>
            <p style="color:
aliceblue">Email:          
 jobportal@gmail.com</p>
            <p style="color: aliceblue">Mobile No.:  9745632121</p>
         </div>
      </div>
   </div>

{% endblock %}
```

OUTPUT:

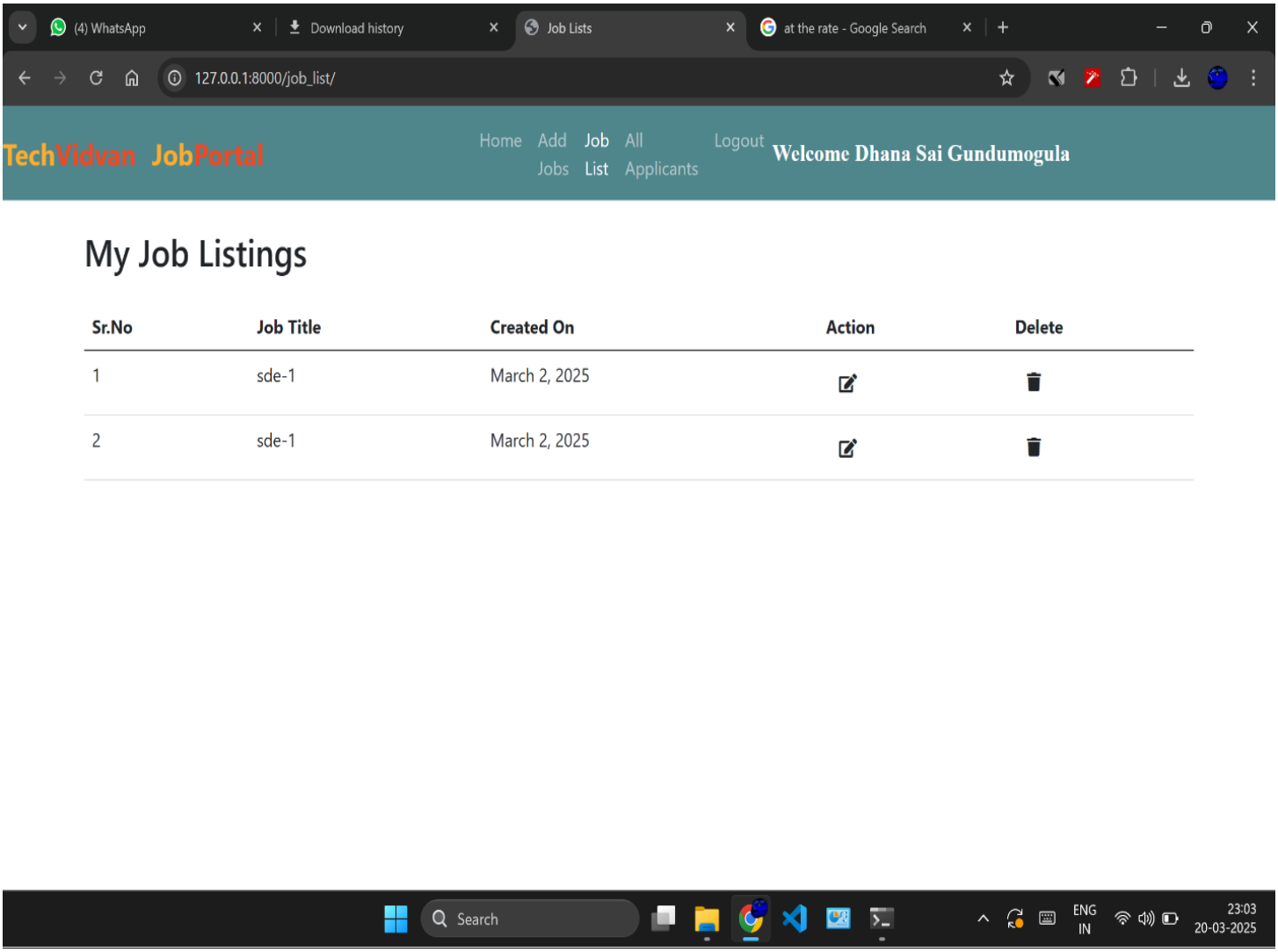
The index.html template serves as the landing page of the Job Portal.

i. Displays a full-width banner image for visual appeal.
ii. Includes a "Contact Us" section with email and phone details.
iii. Ensures consistent styling and structure by extending the base layout (basic.html).
iv. Loads static files like images using Django's {% static %} tag.
v. Marks the homepage as active in the navigation bar.

Job_list.html:

```
{% extends 'company_navbar.html' %}

{% block title %} Job Lists {% endblock %}
{% block job_list %} active {% endblock %}
{% block css %}
{% endblock %}
{% block body %}
<div class="container mt-4">
  <h2 class="mb-4">My Job Listings</h2>
  <table class="table table-hover" id="example">
    <thead>
      <tr>
        <th>Sr.No</th>
        <th>Job Title</th>
        <th>Created On</th>
        <th>Action</th>
        <th>Delete</th>
      </tr>
    </thead>
    <tbody>
      {% for job in jobs %}
      <tr>
        <td>{{ forloop.counter }}</td>
        <td>{{ job.title }}</td>
        <td>{{ job.creation_date }}</td>
        <td>
          <a href="{% url 'edit_job' job.id %}" class="btn"><i class="fa fa-edit"></i></a>
        </td>
        <td>
          <form action="{% url 'delete_job' job.id %}" method="post" style="display:inline;">
            {% csrf_token %}
            <button type="submit" class="btn" onclick="return confirm('Are you sure you want
to delete this job?');" style="border: none; background: none;">
              <i class="fa fa-trash"></i>
            </button>
          </form>
        </td>
      </tr>
      {% endfor %}
    </tbody>
  </table>
</div>
{% endblock %}
```

OUTPUT:



Role of job_list.html in Django:

The job_list.html template allows companies to view, edit, or delete the jobs they have posted.

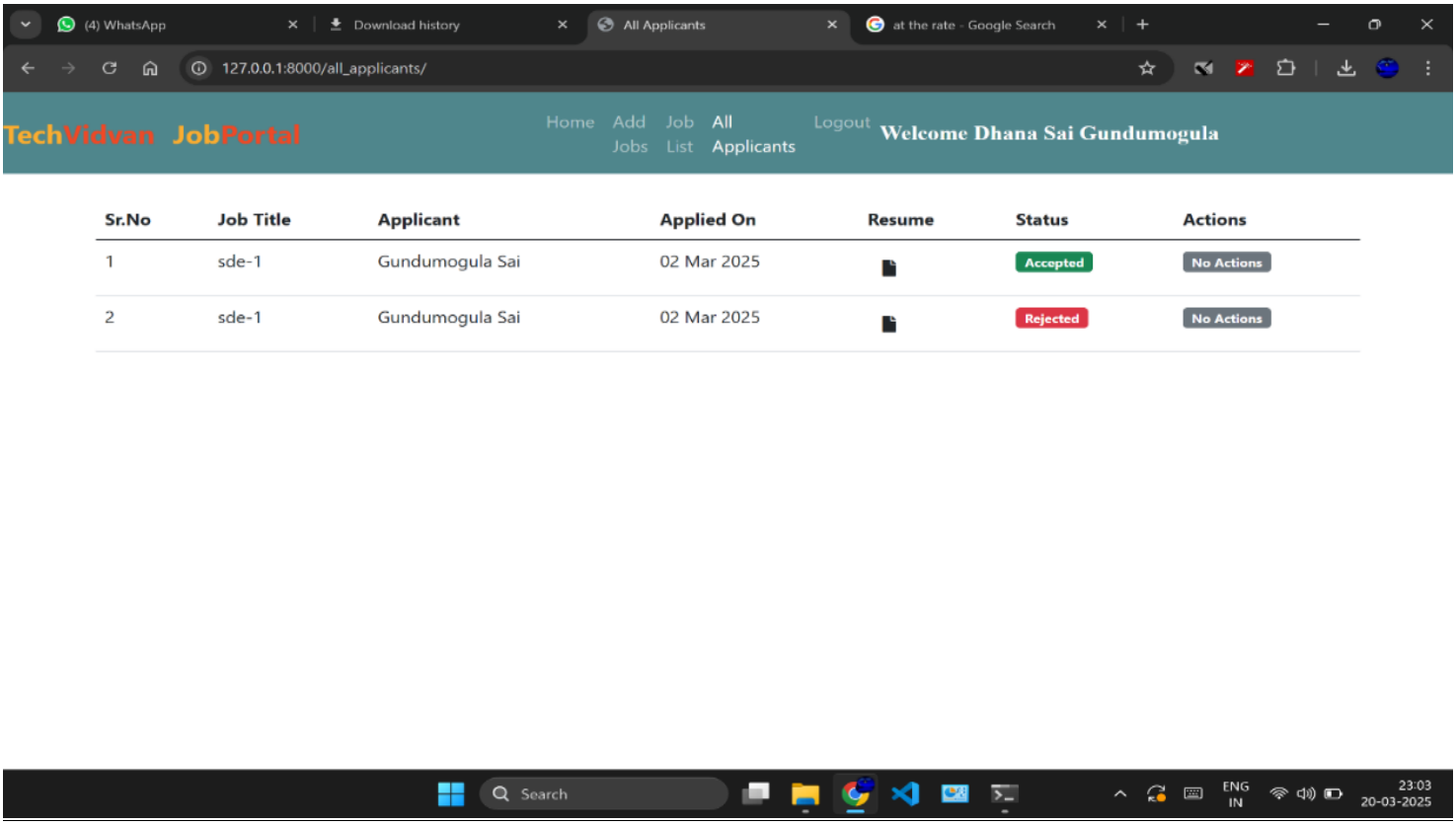
i. Displays a table of all jobs posted by the logged-in company.
ii. Shows details like job title and creation date.
iii. Provides an Edit button to update job details.
iv. Includes a Delete button with a confirmation prompt.
v. Uses Django's {% url %} and CSRF protection for safe routing and form handling.
vi. Maintains a consistent layout by extending the company navbar template.

All_applicants.html

```
{% extends 'company_navbar.html' %}

{% block title %} All Applicants {% endblock %}
{% block all_applicants %} active {% endblock %}

{% block body %}
<div class="container mt-4">
   {% if application %}
   <table class="table table-hover" id="example">
      <thead>
         <tr>
            <th>Sr.No</th>
            <th>Job Title</th>
            <th>Applicant</th>
            <th>Applied On</th>
            <th>Resume</th>
            <th>Status</th>
            <th>Actions</th>
         </tr>
      </thead>
      <tbody>
         {% for i in application %}
         <tr>
            <td>{{ forloop.counter }}</td>
            <td>{{ i.job.title }}</td>
            <td>{{ i.applicant.user.get_full_name }}</td>
            <td>{{ i.apply_date|date:"d M Y" }}</td>
            <td>
               <a href="{{ i.resume.url }}" class="btn"><i class="fa fa-file"></i></a>
            </td>
            <td>
               {% if i.status == "Accepted" %}
                  <span class="badge bg-success">Accepted</span>
               {% elif i.status == "Rejected" %}
                  <span class="badge bg-danger">Rejected</span>
               {% else %}
                  <span class="badge bg-secondary">Pending</span>
               {% endif %}
            </td>
            <td>
               {% if i.status == "Pending" %}
                  <form action="{% url 'accept_applicant' i.id %}" method="post"
style="display:inline;">
                     {% csrf_token %}
```

```
                    <button type="submit" class="btn btn-success btn-sm">Accept</button>
                </form>

                <form action="{% url 'reject_applicant' i.id %}" method="post"
style="display:inline;">
                    {% csrf_token %}
                    <button type="submit" class="btn btn-danger btn-sm">Reject</button>
                </form>
            {% else %}
                <span class="badge bg-secondary">No Actions</span>
            {% endif %}
        </td>
    </tr>
    {% endfor %}
  </tbody>
 </table>
 {% else %}
 <p>No applications found.</p>
 {% endif %}
</div>
{% endblock %}
```

Output:



Role of all_applicants.html in Django:

The all_applicants.html page lets companies view and manage job applications.

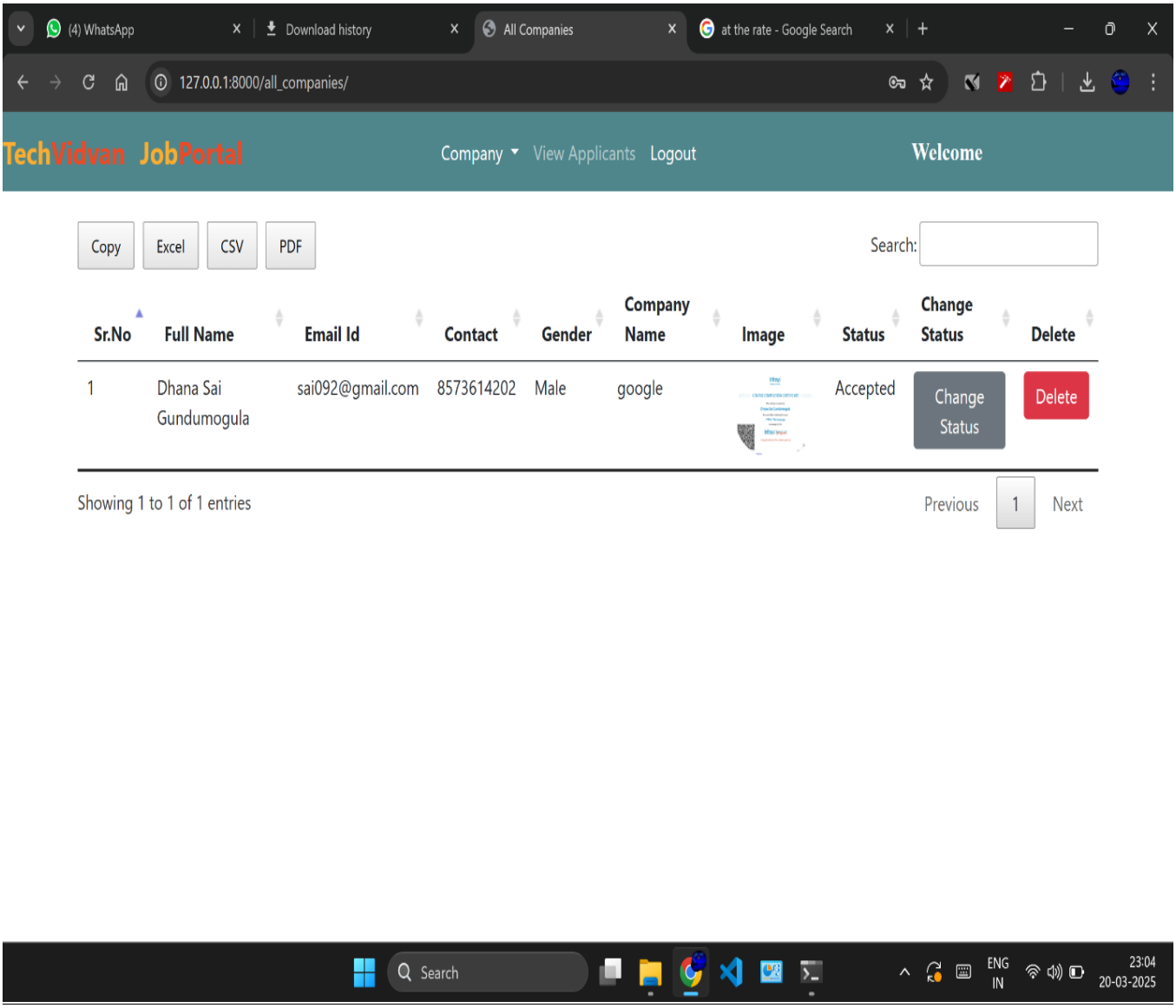
i. Lists all applicants who applied for the company's jobs.
ii. Shows job title, applicant name, apply date, resume, and status.
iii. Lets companies accept or reject pending applications.
iv. Shows "No Actions" if the application is already accepted or rejected.
v. Keeps the design consistent by using the company's navbar.

All_compaines.html

```
{% extends 'admin_navbar.html' %}

{% block title %} All Companies {% endblock %}
{% block application_form %} active {% endblock %}
{% block css %}
{% endblock %}
{% block body %}
<div class="container mt-4">
<table class="table table-hover" id="example">
  <thead>
    <tr>
      <th>Sr.No</th>
      <th>Full Name</th>
      <th>Email Id</th>
      <th>Contact</th>
      <th>Gender</th>
      <th>Company Name</th>
      <th>Image</th>
      <th>Status</th>
      <th>Change Status</th>
      <th>Delete</th>
    </tr>
  </thead>
  <tbody>
    {% for company in companies %}
    <tr>
      <td>{{forloop.counter}}</td>
      <td>{{company.user.get_full_name}}</td>
      <td>{{company.user.email}}</td>
      <td>{{company.phone}}</td>
      <td>{{company.gender}}</td>
      <td>{{company.company_name}}</td>
      <td><img src="{{company.image.url}}" class="rounded-circle" width="90px"
height="70px"></td>
      <td>{{company.status}}</td>
```

```
        <td><a href="/change_status/{{company.id}}/" class="btn btn-secondary">Change
Status</a></td>
        <td><a href="/delete_company/{{company.user.id}}/" class="btn btn-danger"
onclick="return confirm('Are you sure you want to delete this
company?')">Delete</a></td>
      </tr>
      {% endfor %}
   </tbody>
</table>
</div>
{% endblock %}
```

OUTPUT:



Role of all_companies.html in Django:

The all_companies.html template allows the admin to manage all registered companies.

i. Displays a table of all companies with their name, email, phone, gender, company name, and profile image.
ii. Shows each company's current status (e.g., Approved, Pending).
iii. Provides a button to change the status of any company.
iv. Allows the admin to delete a company with confirmation.
v. Ensures the layout is consistent using the admin navbar.

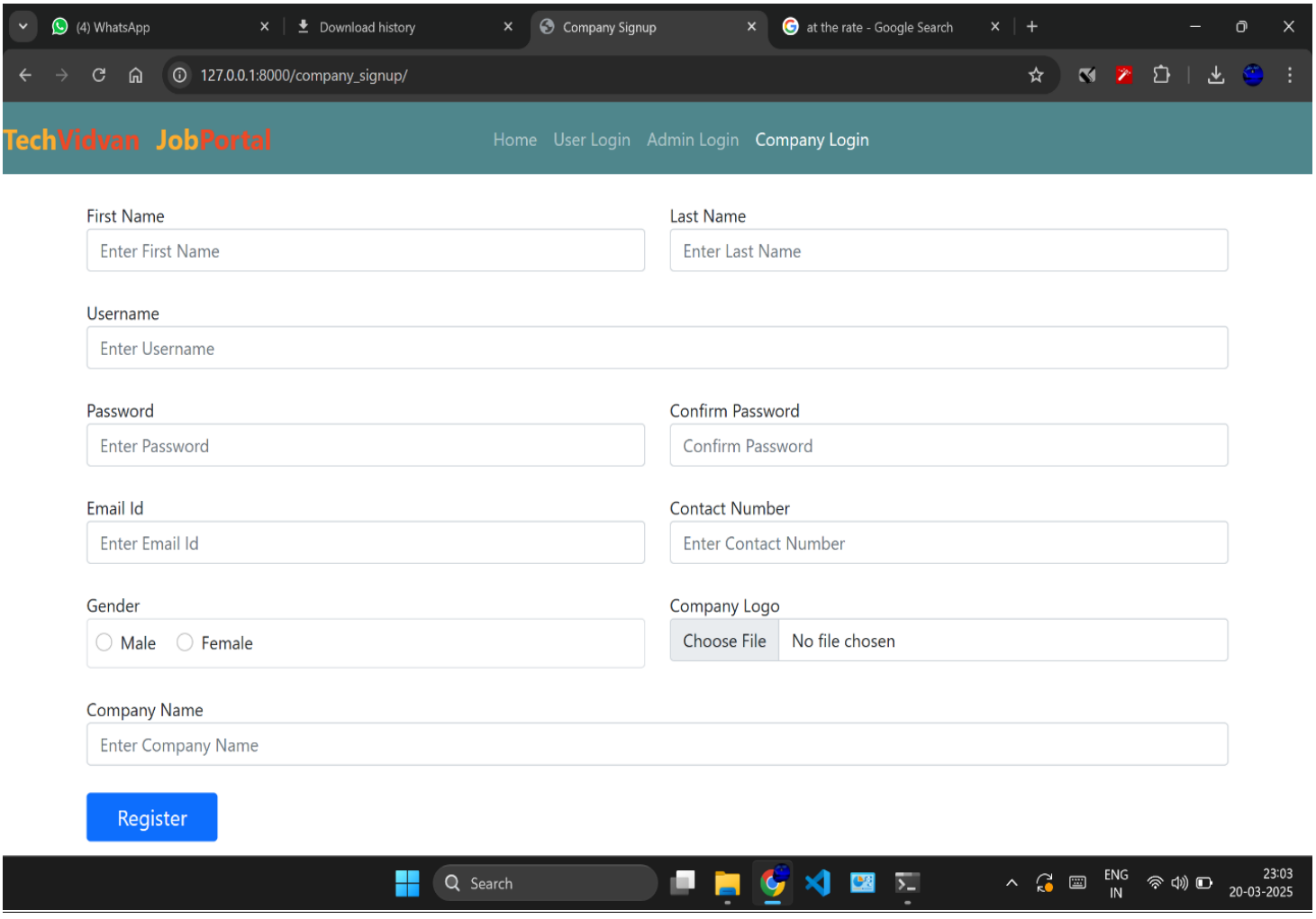
company_signup.html:

```
{% extends 'basic.html' %}

{% block title %} Company Signup {% endblock %}
{% block company_login %} active {% endblock %}
{% block css %}
{% endblock %}

{% block body %}
<form class="container mt-4" method="POST" enctype="multipart/form-data">
{% csrf_token %}

<div class="row">
<div class="form-group col-md-6">
<label>First Name</label>
<input type="text" class="form-control" name="first_name" id="first_name"
placeholder="Enter First Name" required>
</div>
<div class="form-group col-md-6">
<label>Last Name</label>
<input type="text" class="form-control" name="last_name" id="last_name"
placeholder="Enter Last Name" required>
</div>
</div>

<div class="row mt-4">
<div class="form-group col-md-12">
<label>Username</label>
<input type="text" class="form-control" name="username" id="username"
placeholder="Enter Username" required>
</div>
</div>

<div class="row mt-4">
<div class="form-group col-md-6">
<label>Password</label>
```

```
<input type="password" class="form-control" name="password1" id="password1"
placeholder="Enter Password" required>
</div>
<div class="form-group col-md-6">
<label>Confirm Password</label>
<input type="password" class="form-control" name="password2" id="password2"
placeholder="Confirm Password" required>
</div>
</div>

<div class="row mt-4">
<div class="form-group col-md-6">
<label>Email Id</label>
<input type="email" class="form-control" name="email" id="email"
placeholder="Enter Email Id" required>
</div>
<div class="form-group col-md-6">
<label>Contact Number</label>
<input type="tel" class="form-control" name="phone" id="phone"
placeholder="Enter Contact Number" pattern="[0-9]{10}" required>
</div>
</div>

<div class="row mt-4">
<div class="form-group col-md-6">
<label>Gender</label>
<div class="border p-2 rounded">
<div class="form-check form-check-inline">
<input type="radio" class="form-check-input" name="gender" id="male"
value="Male" required>
<label for="male" class="form-check-label">Male</label>
</div>
<div class="form-check form-check-inline">
<input type="radio" class="form-check-input" name="gender" id="female"
value="Female" required>
<label for="female" class="form-check-label">Female</label>
</div>
</div>
</div>
<div class="form-group col-md-6">
<label>Company Logo</label>
<input type="file" class="form-control" name="image" id="image"
accept="image/*" required>
</div>
</div>
```

```
<div class="row mt-4">
<div class="form-group col-md-12">
<label>Company Name</label>
<input type="text" class="form-control" name="company_name"
id="company_name" placeholder="Enter Company Name" required>
</div>
</div>

<br>
<input type="submit" value="Register" class="btn btn-primary" style="font-size:
larger; width: 8rem;">
</form>
{% endblock %}
```

OUTPUT:



Role of company_signup.html in Django:

The company_signup.html template allows a company to register on the platform.

i. Takes personal information like first name, last name, and contact details.

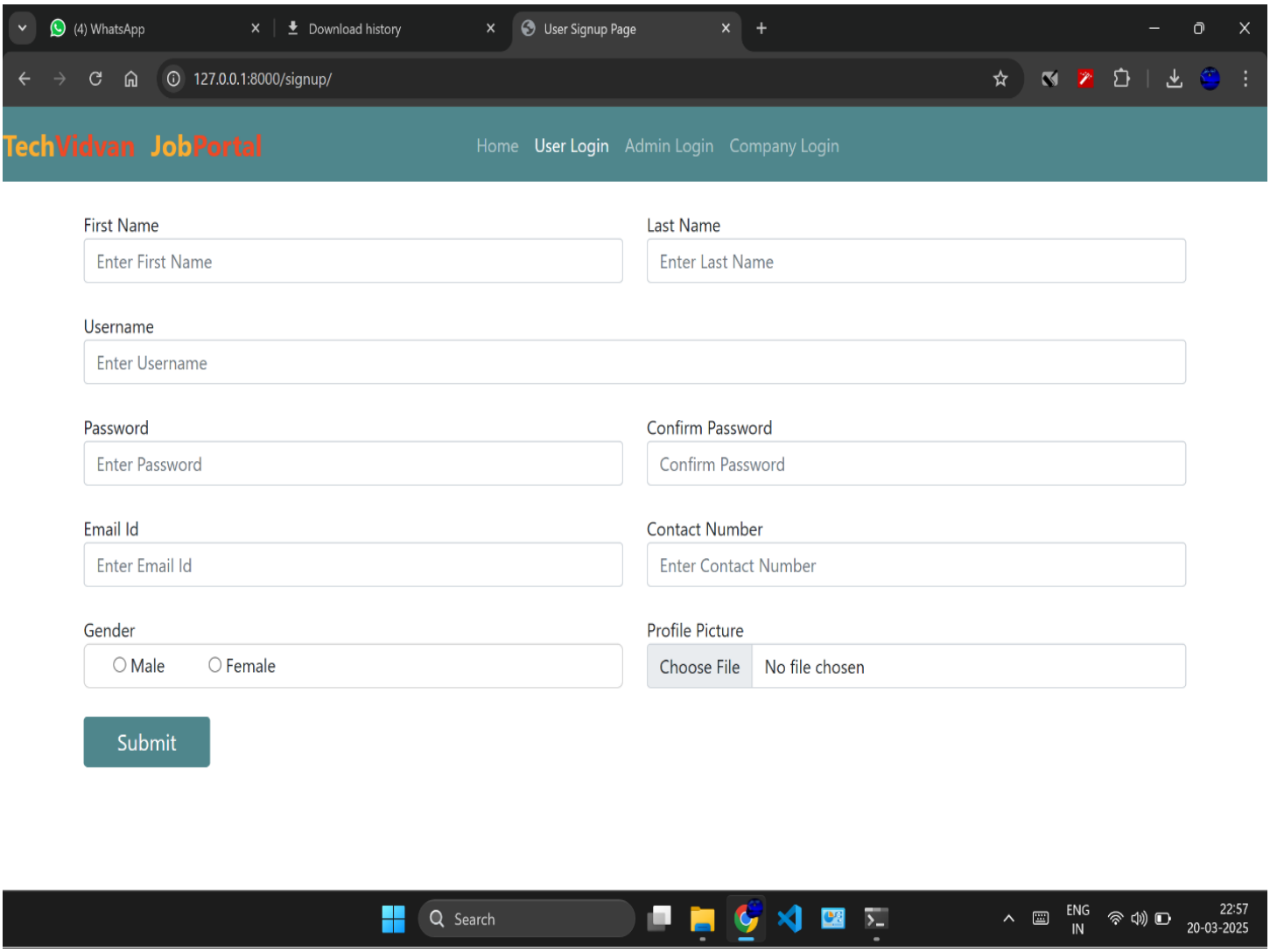
ii. Collects login credentials including username and password (with confirmation).
iii. Asks for company-specific info like company name and logo.
iv. Includes gender selection through radio buttons.
v. Uses CSRF protection and handles file uploads (e.g., logo image).
vi. Ensures the layout is consistent using the base template.Teacher_dashboard.html:

Signup.html:

```
{% extends 'basic.html' %}

{% block title %} User Signup Page {% endblock %}
{% block user_login %} active {% endblock %}
{% block css %}
{% endblock %}
{% block body %}
<form class="container mt-4" method="POST" enctype="multipart/form-data">
  {% csrf_token %}

  <div class="row">
    <div class="form-group col-md-6">
      <label>First Name</label>
      <input type="text" class="form-control" name="first_name" id="first_name"
placeholder="Enter First Name">
    </div>
    <div class="form-group col-md-6">
      <label>Last Name</label>
      <input type="text" class="form-control" name="last_name" id="last_name"
placeholder="Enter Last Name">
    </div>
  </div>

  <div class="row mt-4">
    <div class="form-group col-md-12">
      <label>Username</label>
      <input type="text" class="form-control" name="username" id="username"
placeholder="Enter Username">
    </div>
  </div>

  <div class="row mt-4">
    <div class="form-group col-md-6">
      <label>Password</label>
      <input type="password" class="form-control" name="password1"
id="password1" placeholder="Enter Password">
    </div>
    <div class="form-group col-md-6">
```

```
            <label>Confirm Password</label>
            <input type="password" class="form-control" name="password2"
id="password2" placeholder="Confirm Password">
        </div>
    </div>

    <div class="row mt-4">
        <div class="form-group col-md-6">
            <label>Email Id</label>
            <input type="email" class="form-control" name="email" id="email"
placeholder="Enter Email Id">
        </div>
        <div class="form-group col-md-6">
            <label>Contact Number</label>
            <input type="tel" class="form-control" name="phone" id="phone"
placeholder="Enter Contact Number">
        </div>
    </div>

    <div class="row mt-4">
        <div class="form-group col-md-6">
            <label>Gender</label>
            <div style="border: 1px solid lightgrey; padding: 5px; border-radius: 6px;">
            <div class="form-check form-check-inline">
                <input type="radio" class="custom-control-input" name="gender" id="male"
value="Male">
                <label for="male" class="custom-control-label">Male</label>
            </div>
            <div class="form-check form-check-inline">
                <input type="radio" class="custom-control-input" name="gender"
id="female" value="Female">
                <label for="female" class="custom-control-label">Female</label>
            </div>
        </div>
        </div>
        <div class="form-group col-md-6">
            <label>Profile Picture</label>
            <input type="file" class="form-control" name="image" id="image"
placeholder="Enter Last Name">
        </div>
    </div>
    <br>
<input type="submit" value="Submit" class="btn" style="background-color: #4f868c;
color: white; font-size: larger; width: 8rem;">
</form>
{% endblock %}
```

OUTPUT:



Role of user_signup.html in Django:

The user_signup.html template enables new users to register on the platform.

i. Collects personal information like first name, last name, email, and contact number.

ii. Accepts login credentials including username and password (with confirmation).

iii. Includes gender selection using radio buttons.

iv. Allows users to upload a profile picture.

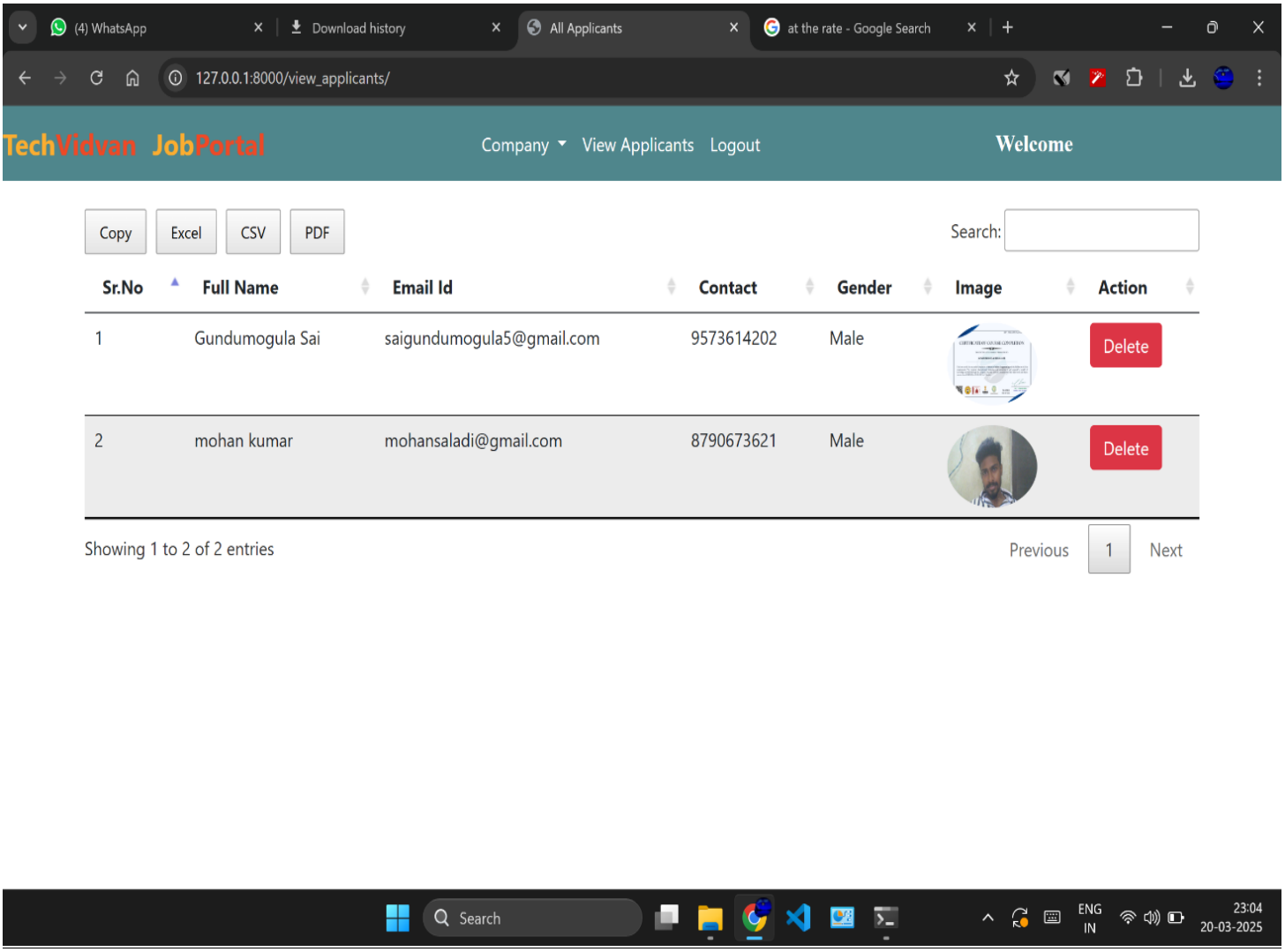
v. Uses CSRF protection and handles file uploads securely.

vi. Maintains layout consistency using the base template.View_schedule.html:

View_applicants.html:

```
{% extends 'admin_navbar.html' %}

{% block title %} All Applicants {% endblock %}
{% block view_applicants %} active {% endblock %}
{% block css %}
{% endblock %}
{% block body %}
<div class="container mt-4">
<table class="table table-hover" id="example">
   <thead>
      <tr>
         <th>Sr.No</th>
         <th>Full Name</th>
         <th>Email Id</th>
         <th>Contact</th>
         <th>Gender</th>
         <th>Image</th>
         <th>Action</th>
      </tr>
   </thead>
   <tbody>
      {% for applicant in applicants %}
      <tr>
         <td>{{forloop.counter}}</td>
         <td>{{applicant.user.get_full_name}}</td>
         <td>{{applicant.user.email}}</td>
         <td>{{applicant.phone}}</td>
         <td>{{applicant.gender}}</td>
         <td><img src="{{applicant.image.url}}" class="rounded-circle" width="90px"
height="70px"></td>
         <td><a href="/delete_user/{{applicant.user.id}}/" class="btn btn-danger"
onclick="return confirm('Are you sure you want to delete this
applicant?')">Delete</a></td>
      </tr>
      {% endfor %}
   </tbody>
</table>
</div>
{% endblock %}
```

OUTPUT:



Role of all_applicants.html in Django:

The all_applicants.html template allows the admin to manage all registered applicants.

i. Displays a table of all applicants with their name, email, phone, gender, and profile image.

ii. Includes a delete button for removing applicants from the system.

iii. Ensures deletion confirmation for safety.

iv. Maintains consistent UI using the admin navbar.

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class :II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Database Integration and Configuration-SQLLITE
7. Date of Experiment : 17-02-2025
8. Date of Submission of Report : 21-02-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty

Database:

A database is an organized collection of data that is stored and managed in a way that makes it easy to retrieve, update, and manage information. Databases are used to store data in a structured format, allowing efficient access, management, and modification.

Step 1: Check if SQLite3 is Already Installed

sqlite3 –version

Step 2: Install SQLite3 (if not already installed)

On Windows:

1. Download the SQLite3 command-line tool from the official website: SQLite Downloads
2. Download the **"Precompiled Binaries for Windows"** (usually a ZIP file).
3. Extract the ZIP file and place sqlite3.exe in a folder (e.g., C:\sqlite).
4. Add the folder to your system's **PATH** environment variable:
    i. Search for "Environment Variables" in the Start menu.
    ii. Click on "Environment Variables."
    iii. In "System variables," select "Path" and click "Edit."
    iv. Add the folder path (e.g., C:\sqlite) and click OK.

On macOS:

sqlite3: --version

Step 4: Using SQLite3 with Django

Since Django uses SQLite3 as the default database, you don't need to install any additional drivers. Just make sure your settings.py file has the following configuration:

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

Step 5: Run Migrations to Create the Database

i. After setting up, run:
ii. python manage.py migrate

Seeing the table content :

Step 1: Open the Django DB Shell

i. Make sure your virtual environment is activated, then use the following command:
ii. python manage.py db shell

Step 2: Common SQLite Commands

i. Once you're inside the SQLite shell, you can use the following commands:
ii. List All Tables:
iii. .tables

Step3: View Table Schema:

i. .schema table_name;

Step4: Show All Data in a Table:

i. SELECT * FROM table_name;
ii. Exit the SQLite Shell:
iii. .exit

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class : II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Handling Forms in Django
7. Date of Experiment : 21-02-2025
8. Date of Submission of Report : 21-02-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty

What is forms.py in Django:

In Django, forms.py is used to handle user input efficiently and securely. It allows developers to create and manage forms without manually writing HTML and validation logic.

Why Use forms.py:

i. Simplifies form creation
ii. Handles input validation automatically
iii. Integrates with Django models
iv. Prevents security risks like SQL Injection & CSRF attacks

Types of Forms in Django:

i. Django Forms (forms.Form) – Used for manually creating forms
ii. **Model Forms (forms.ModelForm)** – Used to create forms directly from a Django model

```
# myapp1/forms.py
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
from .models import Applicant, Company

# Form for user signup
class UserSignupForm(UserCreationForm):
    email = forms.EmailField(required=True)
    phone = forms.CharField(max_length=15)
    gender = forms.ChoiceField(choices=[('Male', 'Male'), ('Female', 'Female')],
widget=forms.RadioSelect)
    image = forms.ImageField()

    class Meta:
        model = User
        fields = ['first_name', 'last_name', 'username', 'password1', 'password2', 'email']

    def save(self, commit=True):
        user = super().save(commit=False)
        if commit:
            user.save()
        return user


# Form for company signup
class CompanySignupForm(UserCreationForm):
    email = forms.EmailField(required=True)
    phone = forms.CharField(max_length=15)
```

```
    gender = forms.ChoiceField(choices=[('Male', 'Male'), ('Female', 'Female')],
widget=forms.RadioSelect)
    company_name = forms.CharField(max_length=100)
    image = forms.ImageField(label="Company Logo")

    class Meta:
        model = User
        fields = ['first_name', 'last_name', 'username', 'password1', 'password2', 'email']

    def save(self, commit=True):
        user = super().save(commit=False)
        if commit:
            user.save()
        return user


# Optional: Form to update applicant or company profile (if needed later)
class ApplicantProfileForm(forms.ModelForm):
    class Meta:
        model = Applicant
        fields = ['phone', 'gender', 'image']


class CompanyProfileForm(forms.ModelForm):
    class Meta:
        model = Company
        fields = ['phone', 'gender', 'company_name', 'image']
```

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class : II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Defining And Using Models
7. Date of Experiment : 21-02-2025
8. Date of Submission of Report : 07-03-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty

What is models.py in Django:

In Django, models.py is where you define the database structure using Python code. Django models act as a bridge between the database and the application, allowing you to create, read, update, and delete records easily.

Why Use Django Models:

No need to write raw SQL queries, Automatically creates tables in the database and

How to Apply Models:

Create the Model in models.py:

i. Write your models inside the models.py file.

MODELS.PY:

```
from django.db import models
from django.contrib.auth.models import User

class Applicant(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    phone = models.CharField(max_length=10)
    image = models.ImageField(upload_to="")
    gender = models.CharField(max_length=10)
    type = models.CharField(max_length=15)

    def __str__(self):
        return self.user.first_name

class Company(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    phone = models.CharField(max_length=10)
    image = models.ImageField(upload_to="")
    gender = models.CharField(max_length=10)
    type = models.CharField(max_length=15)
    status = models.CharField(max_length=20)
    company_name = models.CharField(max_length=100)

    def __str__ (self):
        return self.user.username

class Job(models.Model):
    company = models.ForeignKey(Company, on_delete=models.CASCADE)
    start_date = models.DateField()
    end_date = models.DateField()
    title = models.CharField(max_length=200)
```

```
    salary = models.FloatField()
    image = models.ImageField(upload_to="")
    description = models.TextField(max_length=400)
    experience = models.CharField(max_length=100)
    location = models.CharField(max_length=100)
    skills = models.CharField(max_length=200)
    creation_date = models.DateField()

    def __str__ (self):
        return self.title

class Application(models.Model):
    job = models.ForeignKey(Job, on_delete=models.CASCADE)
    applicant = models.ForeignKey(Applicant, on_delete=models.CASCADE)
    resume = models.FileField(upload_to='resumes/')
    apply_date = models.DateTimeField(auto_now_add=True)
    status = models.CharField(max_length=20, default='Pending')
    company = models.ForeignKey(Company, on_delete=models.CASCADE, null=True)

    def save(self, *args, **kwargs):
        if not self.company:
            self.company = self.job.company
        super().save(*args, **kwargs)

    def __str__(self):
        return str(self.applicant)


    def __str__(self):
        return str(self.applicant)


    def __str__ (self):
        return str(self.applicant)
```

Run Migrations to Create Database Tables:

After defining your models, run the following commands to apply them to the database:

i. python manage.py makemigrations
ii. python manage.py migrate

Register Models in admin.py:

Admin.py:

```
from django.contrib import admin
from . models import *

admin.site.register(Applicant)
admin.site.register(Company)
admin.site.register(Job)
admin.site.register(Application)
```

Use Models in Views (views.py):

i. Fetching all rooms in a view:

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class : II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Deploying DjangoApplications on Cloud Platforms
7. Date of Experiment : 27-03-2025
8. Date of Submission of Report : 04-04-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty

Deploying Django Web Application on Cloud

What is Deployment?

Deployment is the process of making a Django web application live on the internet so users can access it. This involves hosting your app on a cloud server like AWS, Google Cloud, Digital Ocean, Heroku, or PythonAnywhere.

Features:

i. Scalability – Handle more users without performance issues.
ii. Security – Protect user data with SSL and secure databases.
iii. Global Accessibility – Users can access your app from anywhere.
iv. Continuous Deployment – Easily update your app with new features.

Here's a step-by-step guide to Register on GitHub, Create a Django website with login and registration pages, and Configure Django to handle static files.

Step 1: Register on GitHub

1. Go to GitHub and click Sign up.
2. Enter your Username, Email, and Password.
3. Complete the verification and click Create Account.
4. Verify your email by clicking the link in your inbox.

Step 2: Push to GitHub

1. Initialize Git in your project: git init
2. Connect to GitHub:

git remote add origin

https://github.com/dhanasai2/job-portal-django

3. Add and commit changes: git add .

git commit -m "Initial Commit: Login and Registration App"

4. Push to GitHub:

git branch -M main

git push -u origin main

You have successfully built a Django website with login, registration, and static file management.Your code is now available on GitHub.

GITHUB LINK:

https://github.com/dhanasai2/job-portal-django

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Dr.Ch. Bindu Madhuri
Asst. Professor & HOD
Email: hod. it@intugvcev.edu.in

1. Name of the Laboratory : Django Framework
2. Name of the Student : S. MOHAN KUMAR
3. Roll No : 24VV5A1273
4. Class : II-BTECH II-SEM
5. Academic Year : 2024-2025
6. Name of Experiment : Front End Web Developer Certification
7. Date of Experiment : 27-03-2025
8. Date of Submission of Report : 04-04-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm andObservations	3	
3	Implementation	3	
4	Schematic diagrams,Architecture, workflow,Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE: Signature of Faculty

Infosys®
Navigate your next

CERTIFICATE OF ACHIEVEMENT

The certificate is awarded to

saladi Mohan kumar

for successfully completing

Front End Web Developer Certification

on February 21, 2025

Infosys | Springboard

Congratulations! You make us proud!

Thirumala Arohi
Executive Vice President and Global Head
Education, Training & Assessment (ETA)
Infosys Limited

Issued on: Friday, February 21, 2025
To verify, scan the QR code at https://verify.onwingspan.com