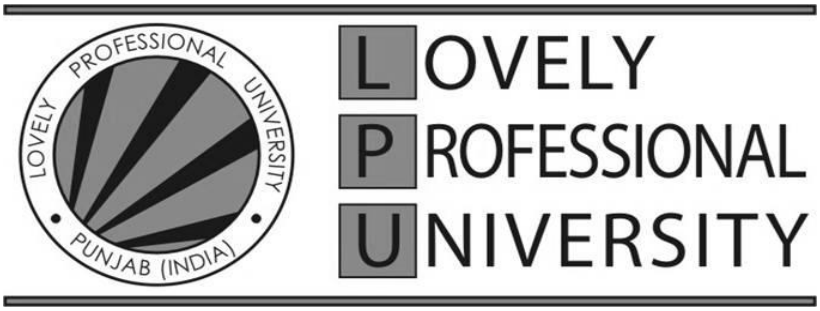

Project Report

on

Payroll Management System

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

in partial fulfillment of the requirements for the award of degree of

Bachelor of Computer Science and Engineering

Submitted by

Garvit Joshi

11808472

Submitted to

Mrs. Navneet Kaur

Assistant Professor

LOVELY FACULTY OF TECHNOLOGY & SCIENCES

LOVELY PROFESSIONAL UNIVERSITY

PUNJAB

[NOVEMBER 2020]

ACKNOWLEDGEMENT

First, we thank The Almighty God for blessing us and supporting us throughout in this pandemic situation. We take this opportunity to express our profound and sincere gratitude to our University i.e. Lovely Professional University, Phagwara for giving us such a nice environment in MyClass Application. We also take this opportunity to express a deep sense of gratitude to our class Teacher Mrs. Navneet Kaur for their cordial support, valuable suggestion, and guidance. We gratefully acknowledge the support extended by Head of the Division.

Table of Contents

CHAPTER 1

INTRODUCTION

1.1. Introduction

A payroll management system is a tool - predominantly a software program - that enables your business to handle all your employee's financial records in a hassle-free, automated fashion. This includes employee's salaries, bonuses, deductions, net pay, and generation of payslips for a specific period.

The payroll management process, in a nutshell, refers to the process of administration of a company's employee's financial records. This would include details of the employee's salaries, incentives, bonuses, deductions, and net pay.

1.1.1. Good Payroll Management System:

If hiring the right team is one of the key pillars of your business, then having a good system in place that helps you manage the team efficiently is another. This article is a step-by-step guide to help you understand how a good payroll management system can manage your business efficiently.

1.1.2. Key Benefits of Payroll management system:

There are several benefits of implementing a service like this for your business. Some of them have been briefly highlighted below:

- **Employee morale -** By making sure your employees are paid in a systematic and timely manner, you are reinforcing their faith in your business' financial integrity. This will boost employee morale and motivate them to perform better.
- **Statutory compliance -** This refers to the legal framework your business must adhere to. As an employer, you are required to maintain various payroll and payment records of your employees. Every organisation that hires employees and pays salaries must comply with the labour laws. By having a payroll process in place, you are automatically complying with the employment and labour laws in India.
- **Manage employee information efficiently** - You will be able to accurately store and manage all your employee information in one place. There will be no need to use any additional tool for this purpose.

1.1.3. Key features for selecting a Payroll Management Service

There are a few features that are an absolute must, irrespective of the size of your business.

Easy setup and on-boarding - A payroll service you choose to work with should be easy to set up without a lot of technical difficulties. Employee on-boarding should be smooth and seamless.

Run on the cloud - The service should be able to run on the cloud and give you the flexibility to access it from anywhere and from your shortlisted devices. This way, you will still be able to administer timely payouts even if you are busy with other obligations.

CHAPTER 2
TECHNOLOGY USED

2.1. Technology Used

The whole project is made in HTML, CSS, and Python as backend. I have used Django framework from Python for Shaping backend.

2.1.1. HTML:

Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images, and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as <img /> and <input /> directly introduce content into the page. Other tags such as <p> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags but use them to interpret the content of the page.

2.1.2. CSS:

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more

flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file which reduces complexity and repetition in the structural content as well as enabling the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

2.1.3. Python

Python is an interpreted, high-level, and general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed, and garbage collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

2.1.4. Django

Django is a Python-based free and open-source web framework that follows the model-template-views (MTV) architectural pattern. It is maintained by the Django Software Foundation (DSF), an American independent organization.

Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models. Django also provides an optional administrative create, read, update, and delete interface that is generated dynamically through introspection and configured via admin models.

2.1.5. SQLite3

SQLite is a relational database management system (RDBMS) contained in a C library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.

CHAPTER 3

MODULES

3.1. Modules

As Python was used for backend server there were many modules used in Python. Some were made by me such as views.py and urls.py which were used for Leading HTML, validating forms and Defining paths. The other modules were Django created such as

3.1.1. **Path**

Returns an element for inclusion in urlpatterns. For example:

```
        app_name = "Management"
urlpatterns=[
    path("", views.index, name="Home"),
    path("fav/", views.view_fav, name="View_Favourite"),
    path("addfav/", views.add_fav, name="Add_Favourite"),
    path("addcompl/", views.add_complaint, name="Add_Complaint"),
    path("compl/", views.view_complaint, name="View_Complaint"),
    path("search/", views.search, name="Search"),
    path("signup/", views.sign_up, name="Sign_Up")
]
```

3.1.2. Forms

Forms were hard coded in Django and were used in HTML by Django API. These were made possible using forms class, a small example is shown below:

```
class NewComplaintForm(forms.Form):
    name = forms.CharField(label="Your name:", max_length=100, required="True")
    sender = forms.EmailField(label="Your Email:", required="True")
    c_type = forms.CharField(label="Complaint Type:", required="True")
    subject = forms.CharField(label="Subject:", min_length=10, required="True")
    message = forms.CharField(label="Message:", max_length=300, required="True")
```

3.1.3. Models

Models were used for Making SQL Entries and they automatically work with Django to edit or add data in SQL tables, a small example can be shown as:

```
class Employee(models.Model):
    Name = models.CharField(max_length=64)
    Email = models.CharField(max_length=64)
    Password = models.CharField(max_length=64)
    Position = models.CharField(max_length=64)
    Salary = models.IntegerField()
```

CHAPTER 4

SCREEN SHOTS WITH CODING

4.1. Screenshots

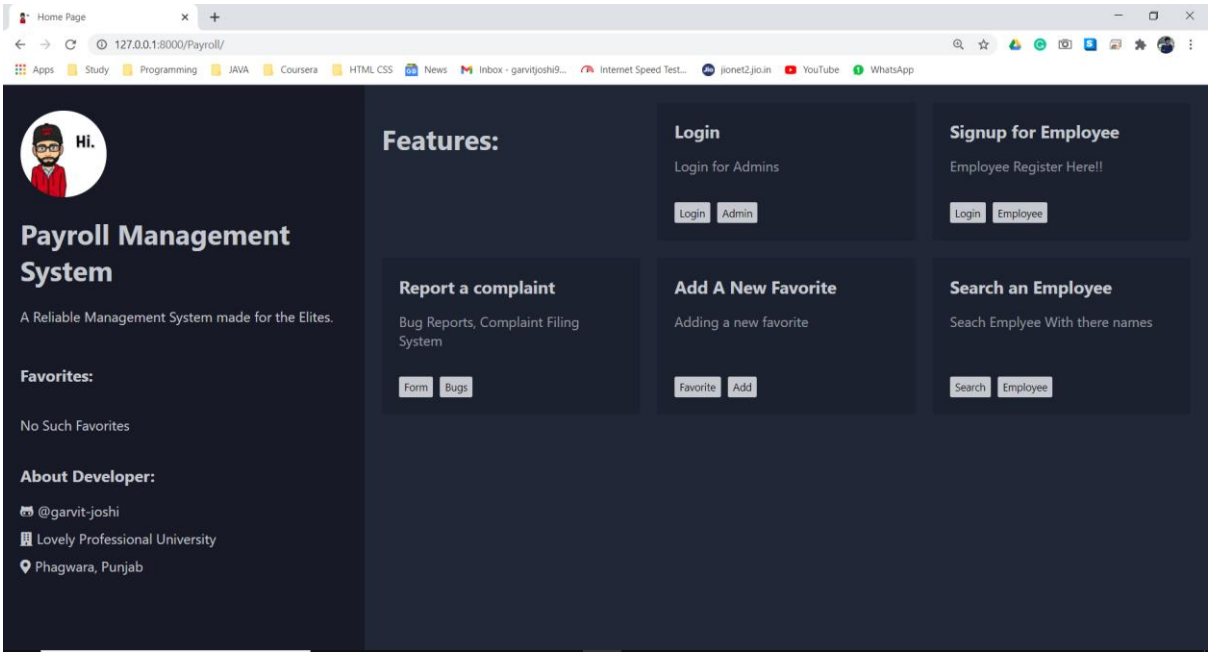

Figure 4.1 Home Page

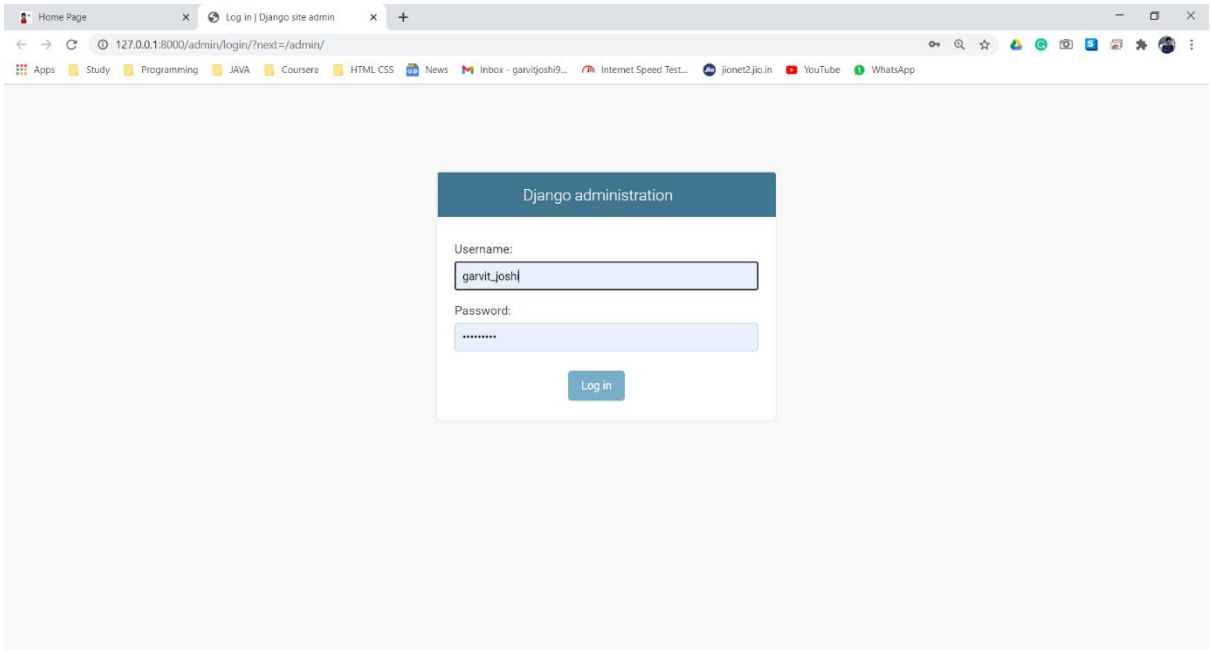

Figure 4.2 Admin Login

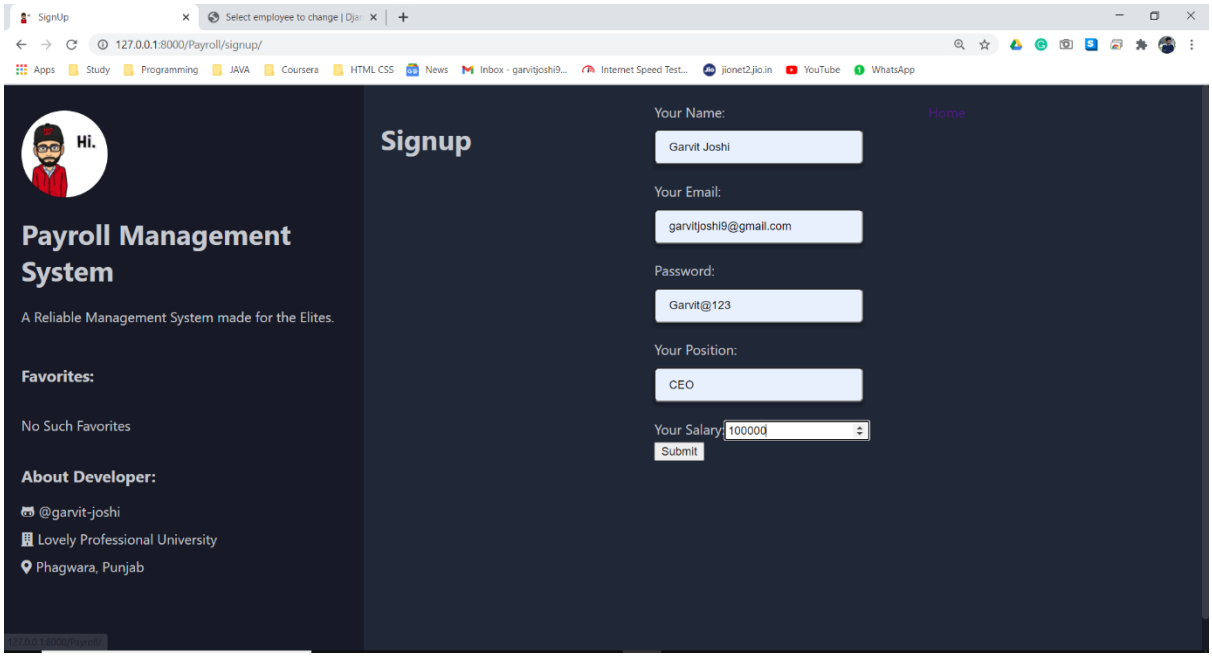

Figure 4.3 Signup for Employee

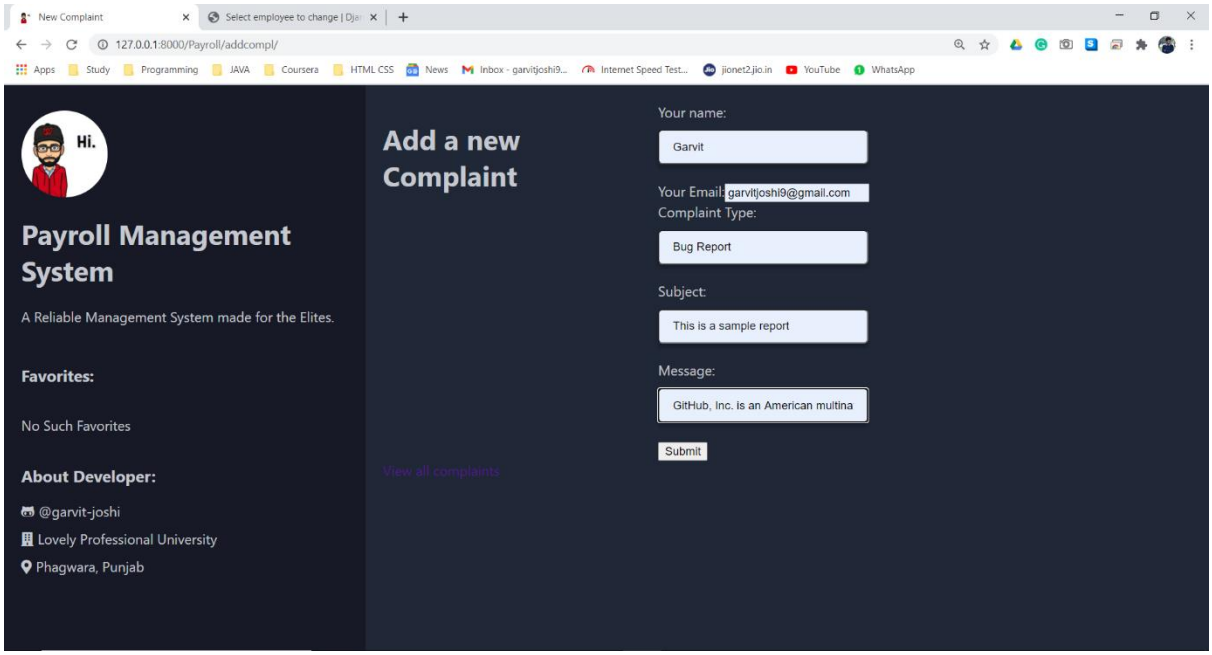

Figure 4.4 Complaint Form

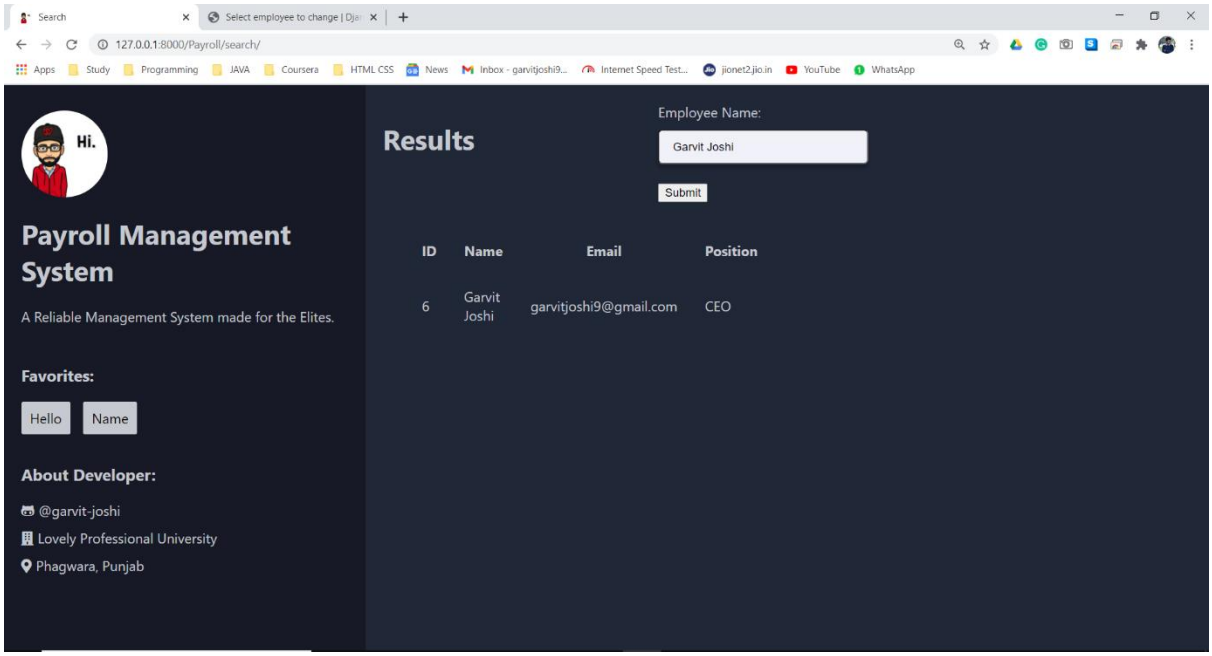

Figure 4.5 Search Employee

4.2. Code Snippet

4.2.1. urls.py

```
from django.urls import path
from . import views


app_name = "Management"
urlpatterns=[
    path("", views.index, name="Home"),
    path("fav/", views.view_fav, name="View_Favourite"),
    path("addfav/", views.add_fav, name="Add_Favourite"),
    path("addcompl/", views.add_complaint, name="Add_Complaint"),
    path("compl/", views.view_complaint, name="View_Complaint"),
    path("search/", views.search, name="Search"),
    path("signup/", views.sign_up, name="Sign_Up")
]
```

4.2.2 views.py

```
from django.shortcuts import render
from django.http import HttpResponse,HttpResponseRedirect
from django import forms
from django.urls import reverse
from .models import Employee

#Adding Favorite List
favorites = []

#Adding Complaint List
complaint_author = []
complaint_email = []
complaint_type = []
complaint_subject = []
```

```
complaint_message = []

class NewFavForm(forms.Form):
    new_favorite = forms.CharField(label="New Favorite", required="True")

class NewComplaintForm(forms.Form):
    name = forms.CharField(label="Your name:", max_length=100, required="True")
    sender = forms.EmailField(label="Your Email:", required="True")
    c_type = forms.CharField(label="Complaint Type:", required="True")
    subject = forms.CharField(label="Subject:", min_length=10, required="True")
    message = forms.CharField(label="Message:", max_length=300, required="True")

class NewSearchForm(forms.Form):
    query = forms.CharField(label="Employee Name:", max_length=100, required="True")

class NewSignupForm(forms.Form):
    Name = forms.CharField(label="Your Name:",max_length=64)
    Email = forms.CharField(label="Your Email:",max_length=64)
    Password = forms.CharField(label="Password",max_length=64)
    Position = forms.CharField(label="Your Position",max_length=64)
    Salary = forms.IntegerField(label="Your Salary:")


# Create your views here.
def index(request):
    return render(request, "Management/index.html", {
        "favorites": favorites
    })

def view_fav(request):
    return render(request, "Management/favourite.html", {
        "favorites": favorites
    })

def add_fav(request):
    if request.method == "POST":
        form = NewFavForm(request.POST)
        if form.is_valid():
            new_favorite = form.cleaned_data["new_favorite"]
            favorites.append(new_favorite)
            return HttpResponseRedirect(reverse("Management:Add_Favourite"))
        else:
            return render(request, "Management/add_favourite.html", {
                "form": form,
                "favorites": favorites
            })

    return render(request, "Management/add_favourite.html", {
        "form": NewFavForm(),
        "favorites": favorites
    })

def add_complaint(request):
    if request.method == "POST":
        form = NewComplaintForm(request.POST)
        if form.is_valid():
            name = form.cleaned_data["name"]
            sender=form.cleaned_data["sender"]
            c_type=form.cleaned_data["c_type"]
            subject=form.cleaned_data["subject"]
            message=form.cleaned_data["message"]
```

```
            complaint_author.append(name)
            complaint_email.append(sender)
            complaint_type.append(c_type)
            complaint_subject.append(subject)
            complaint_message.append(message)
            return HttpResponseRedirect(reverse("Management:Add_Complaint"))
        else:
            return render(request, "Management/add_complaint.html", {
                "form": form,
                "favorites": favorites
            })

    return render(request, "Management/add_complaint.html", {
        "form": NewComplaintForm(),
        "favorites": favorites
    })

def view_complaint(request):
    return render(request, "Management/complaint.html", {
        "Complaint": zip(complaint_author, complaint_email, complaint_type, complaint_subj
ect, complaint_message),
        "favorites": favorites
    })

def search(request):
    result = "NONE"
    if request.method == "POST":
        form = NewSearchForm(request.POST)
        if form.is_valid():
            query = form.cleaned_data["query"]
            result = Employee.objects.filter(Name=query)
            return render(request, "Management/search.html", {
                "form": form,
                "result": result,
                "method": request.method,
                "favorites": favorites
            })
        else:
            return render(request, "Management/search.html", {
                "result": result,
                "form": form,
                "method": request.method,
                "favorites": favorites
            })

    return render(request, "Management/search.html", {
        "result": result,
        "form": NewSearchForm(),
        "method": request.method,
        "favorites": favorites
    })

def sign_up(request):
    if request.method == "POST":
        form = NewSignupForm(request.POST)
        if form.is_valid():
            name = form.cleaned_data["Name"]
            email = form.cleaned_data["Email"]
            password = form.cleaned_data["Password"]
            position = form.cleaned_data["Position"]
            salary = form.cleaned_data["Salary"]
```

```
            new_employee = Employee(Name= name, Email= email, Password= password, Position
= position, Salary= salary)
            new_employee.save()
            return HttpResponseRedirect(reverse("Management:Sign_Up"))
        else:
            return render(request, "Management/signup.html", {
                "form": form,
                "favorites": favorites
            })

    return render(request, "Management/signup.html", {
        "form": NewSignupForm(),
        "favorites": favorites
    })
```

4.2.3. layout.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width,initial-scale=1">
      <link rel="shortcut icon" type="image/ico" href="{% static 'Management/favicon.ico'
%}"/>
      <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.13.0/css/all.cs
s">
      <title>
        {% block title %}
        {{ page_title|default:"Payroll Management System" }}
        {% endblock title %}
    </title>
      <link href="{% static 'Management/styles.css' %}" rel="stylesheet">
   </head>
   <body>
      <div id="root">
         <div class="container">
            <div class="sidebar user">
               <div>
                  <a href="{% url 'Management:Home' %}">
                     <div class="avatar shine">
                        <img src="https://res.cloudinary.com/garvitjoshi9/image/upload/v15
94407409/Git-Hub/1_4_ebmyhb.jpg" class="profile" alt="Company Picture">
                     </div>
                  </a>
                  <h1>Payroll Management System</h1>
                  <p>A Reliable Management System made for the Elites.</p>
               </div>
               <div>
                  <h3>Favorites:</h3>
                  <div class="skills">
                     {% for fav in favorites %}
                     <span>{{ fav }}</span>
                     {% empty %}
                     <p>No Such Favorites</p>
                     {% endfor %}
                  </div>
               </div>
               <div class="details">
                  <h3>About Developer:</h3>
```

```
                <div><i class="fab fa-github-
alt"></i><a href="https://github.com/garvit-
joshi" target="_blank" rel="noopener noreferrer" style="color: inherit;">@garvit-
joshi</a></div>
                <div><i class="fas fa-
building"></i><a href="https://www.lpu.in/" target="_blank" rel="noopener noreferrer" styl
e="color: inherit;">Lovely Professional University</a></div>
                <div><i class="fas fa-map-marker-
alt"></i><a href="https://www.google.com/search?q=Phagwara+Punjab" target="_blank" rel="no
opener noreferrer" style="color: inherit;">Phagwara, Punjab</a></div>
             </div>
          </div>
          <div class="main">
             <div class="repoContainer">
                {% block body %}
                {% endblock body %}
             </div>
             <div class="repoContainer">
                {% block sec_body %}
                {% endblock sec_body %}
             </div>
          </div>
       </div>
    </div>
 </body>
</html>
```

CHAPTER 5

VALIDATION CHECKS

5.1. Validation Checks

5.1.1. CSRF Token:

The CSRF middleware and template tag provides easy-to-use protection against Cross Site Request Forgeries. This type of attack occurs when a malicious website contains a link, a form button or some JavaScript that is intended to perform some action on your website, using the credentials of a logged-in user who visits the malicious site in their browser. A related type of attack, 'login CSRF', where an attacking site tricks a user's browser into logging into a site with someone else's credentials, is also covered.

```
{% extends "Management/layout.html" %}

{% block title %} Add Favourites {% endblock title %}

{% block body %}
    <h1>Add Favorites</h1>
    <form action="{% url 'Management:Add_Favourite' %}" method="POST">
        {% csrf_token %}
        {{ form }}
        <input type="submit">
    </form>
    <div class="container">
        <a href="{% url 'Management:View_Favourite' %}">View Tasks</a>
    </div>
{% endblock %}
```

A Code Snippet of Putting CSRF token into HTML.

5.1.2. Backend Validation:

If by any chance any person is editing HTML of our Webpage to bypass Validation in any form, the backend check of Python will not let Database be modified.

5.1.3. SQL Injections:

SQL injection is a code injection technique that might destroy our database, SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

Models are used with the help of Django and they will not let database be harmed in case of any SQL Injection Hacking.

CHAPTER 6
IMPLEMENTATION AND MAINTAINANCE

6.1. Implementation and Maintenance:

Django Site Maintenance is a site maintenance library that allow you to put your Django site ‘offline’. It’s works at web-server level so it’s possible to completely shutting-down your Django application.

The webserver is configured to redirect all the request to alternative URL if some condition is verified true. Anyway, before activate the redirection Django Site Maintenance will wait for all logged users exit and do not allow new user to logging in. Active users can be informed of the oncoming system shutdown by a on scree message.

In case of Maintenance SQL data will be protected and will not be harmed during Maintenance.

CHAPTER 7
TESTING

7.1. TESTING

I have done testing using VSCODE DEBUGER which helped me to Debug the application.

The only drawback of DEBUGER is that it cannot debug HTML Variable which are made by DJANGO API:

This makes testing of HTML more Difficult in some cases as we do not know the value that is going into HTML Webpage.

Automated testing is an extremely useful bug-killing tool for the modern Web developer. You can use a collection of tests – a test suite – to solve, or avoid, a number of problems:

When you're writing new code, you can use tests to validate your code works as expected.

When you're refactoring or modifying old code, you can use tests to ensure your changes haven't affected your application's behavior unexpectedly.

Testing a Web application is a complex task, because a Web application is made of several layers of logic – from HTTP-level request handling, to form validation and processing, to template rendering. With Django's test-execution framework and assorted utilities, you can simulate requests, insert test data, inspect your application's output and generally verify your code is doing what it should be doing.

CHAPTER 8

FUTURE SCOPE OF PROJECT

8.1. Future Scope of Project

Some Future scope that I can think for my project are:

8.1.1. Bots to take charge

AI and bots are a raging trend now which is deliberately affecting every prospect of work. The same is believed to have impact on the overall payroll management system as well. This will predominantly come in handy for the employees who are shifting departments and need better insight into the overall prospect of work.

8.1.2. Manual payroll to finally go away

In this fast-paced world of technology, where every last bit of our life is dependent on technology for smooth sailing, it is not at all surprising that the manual payroll is about to take a step back this year. If there was one prospect of work that has been a problem in the overall payroll management system is handling everything manually.

8.1.3. Flexible payroll

Yet another one of the future trends that are going to take shape in the coming year is the fact that there is not just going to be one form of payroll management. More organizations are opting for ways to include better flexibility in their payroll management which is amazing for the employees and their needs. While there were just weekly and monthly payrolls and still are, the same is going to change in the coming year.

CHAPTER 9

GITHUB AND VIDEO PRESENTATION LINK

9.1. GITHUB Link:

https://github.com/garvit-joshi/WebWorks

9.2. Video Presentation Link (GOOGLE DRIVE):

https://drive.google.com/drive/folders/1sIycjoADSrws-yE5bdCBGca9YxiG_qZv?usp=sharing

REFRENCES:

1. https://docs.djangoproject.com/en/3.1/
2. https://www.wikipedia.org/
3. https://www.python.org/
4. https://www.youtube.com/channel/UCcabW7890RKJzL968QWEykA(CS50 Harvard)
5. https://www.w3schools.com/
6. https://stackoverflow.com/