“Django CRM”

PROJECT REPORT

Submitted by :-

Navroop

(2210993819-G2)

BE-CSE (Artificial Intelligence)

Guided by

Dr. Rajan Kumar

CHITKARA UNIVERSITY INSITUTE OF ENGINEERING & TECHNOLOGY

CHITKARA UNIVERSITY, RAJPURA

CONTENTS

Acknowledgment ..... **I**

Abstract **1**

1. Introduction **2**

1.1 Objectives **2**

1.2 Limitations **2**

2. Proposed System **3**

3. Database Design **5-6**

3.1 ER Diagram **5**

3.2 Schema Diagram **6**

4. Technical Details **7-9**

5. Code… **10-12**

6. Results **13-16**

7. Conclusion **17**

8. Future Procedure……..18

References **19**

ACKNOWLEDGEMENT

With immense please **I, Miss Navroop** presenting the "Django CRM" project report as part of the curriculum of 'BE-CSE (AI)'.

I would like to express my sincere thanks to **Dr. Rajan Kumar** for their valuable guidance and support in completing my project.

I would also like to express my gratitude towards our dean **Dr. Sushil Kumar Narang** for giving me this great opportunity to do a project on **this website**. Without their support and suggestions, this project would not have been completed.

Signature…….

Name: Navroop

Roll No: 2210993819

ABSTRACT

In the fast-paced world of modern business, maintaining strong relationships with customers is paramount. A Customer Relationship Management (CRM) system is a vital tool for businesses of all sizes to manage interactions with current and potential customers. This project aims to develop a basic CRM system using Django, a high-level Python web framework, to streamline business processes and enhance customer experiences.

The Django CRM project will focus on core functionalities such as customer data management, lead tracking, task scheduling, and communication management. Users will be able to easily add, update, and retrieve customer information, keeping a comprehensive record of interactions. The system will also provide features for tracking leads through the sales pipeline, assigning tasks to team members, and scheduling follow-up activities.

Key features of the Django CRM project will include:

User Authentication and Authorization: Secure login and role-based access control to protect sensitive customer data.
Customer Data Management: Create, read, update, and delete operations for managing customer information, including contact details, preferences, and communication history.
The Django CRM project will be developed following best practices in software engineering, including modular design, code reusability, and documentation. It will leverage Django's built-in features such as ORM (Object-Relational Mapping) for database interactions, forms for data validation, and templating for dynamic content generation.

By implementing a Django-based CRM system, businesses can streamline their operations, improve customer engagement, and ultimately drive growth and success. This project will serve as a foundation for further customization and expansion to meet the specific needs of different industries and businesses.

CHAPTER 1

Introduction

1.1 OBJECTIVES:

- Project Overview: Introduce the purpose of developing a Customer Relationship Model using Django.
- Customer Management: Create a system for efficiently managing customer information.
- Interaction Tracking: Implement functionality to track customer interactions comprehensively.
- Task Management: Enable users to create and assign tasks related to customer interactions.
- User Authentication and Authorization: Ensure secure access with user authentication and authorization mechanisms.
- Scalability and Performance: Develop the system with scalability and performance in mind for future growth.
- User-Friendly Interface: Design an intuitive interface for easy navigation and data management.
- Documentation and Training: Provide comprehensive documentation and training resources for users and developers.

1.2 LIMITATIONS:

- Limited Features: A basic Django project may lack advanced features found in dedicated CRM software, such as advanced reporting, predictive analytics, or marketing automation.

- Scalability: While Django is capable of handling large-scale applications, a basic project may not be optimized for scalability, potentially leading to performance issues as the customer base grows.

- User Interface Complexity: Developing a user-friendly interface with advanced features like drag-and-drop functionality or interactive charts may require additional effort beyond the scope of a basic Django project.

- Security Risks: While Django provides built-in security features, a basic project may not fully address all potential security vulnerabilities, leaving the system susceptible to attacks if not properly configured and maintained.

- Maintenance Overhead: As the project grows and evolves, maintaining and extending the functionality of a basic Django CRM may become increasingly complex and time-consuming, potentially requiring significant refactoring or redesign.

CHAPTER 2

PROPOSED SYSTEM

A basic Django CRM project, aims to streamline customer relationship management processes. Leveraging the Django web framework, it will facilitate efficient management of customer information, including contact details and interaction history. Through features like task management, users can schedule follow-ups and appointments seamlessly. Robust user authentication ensures secure access, while scalability and performance optimizations prepare the system for future growth. With an intuitive interface and comprehensive documentation, users will find it easy to navigate and utilize the system effectively. This Django CRM project promises to enhance customer relationship management practices for businesses of varying scales.

SOFTWARE REQUIREMENTS:

Python: Ensure Python 3.x is installed.

Django: Install Django using pip.

Database Management System: Choose and install a DBMS like SQLite, PostgreSQL, or MySQL.

Text Editor or IDE: Select and set up a text editor or IDE for coding.

Git: Install Git for version control.

Virtual Environment: Set up a virtual environment using venv, virtualenv, or conda.

Web Browser: Have a web browser for testing and interaction.

HARDWARE REQUIREMENTS:

Processor: Any modern processor capable of running Python.

Memory (RAM): Minimum 2 GB, preferably 4 GB or more for better performance.

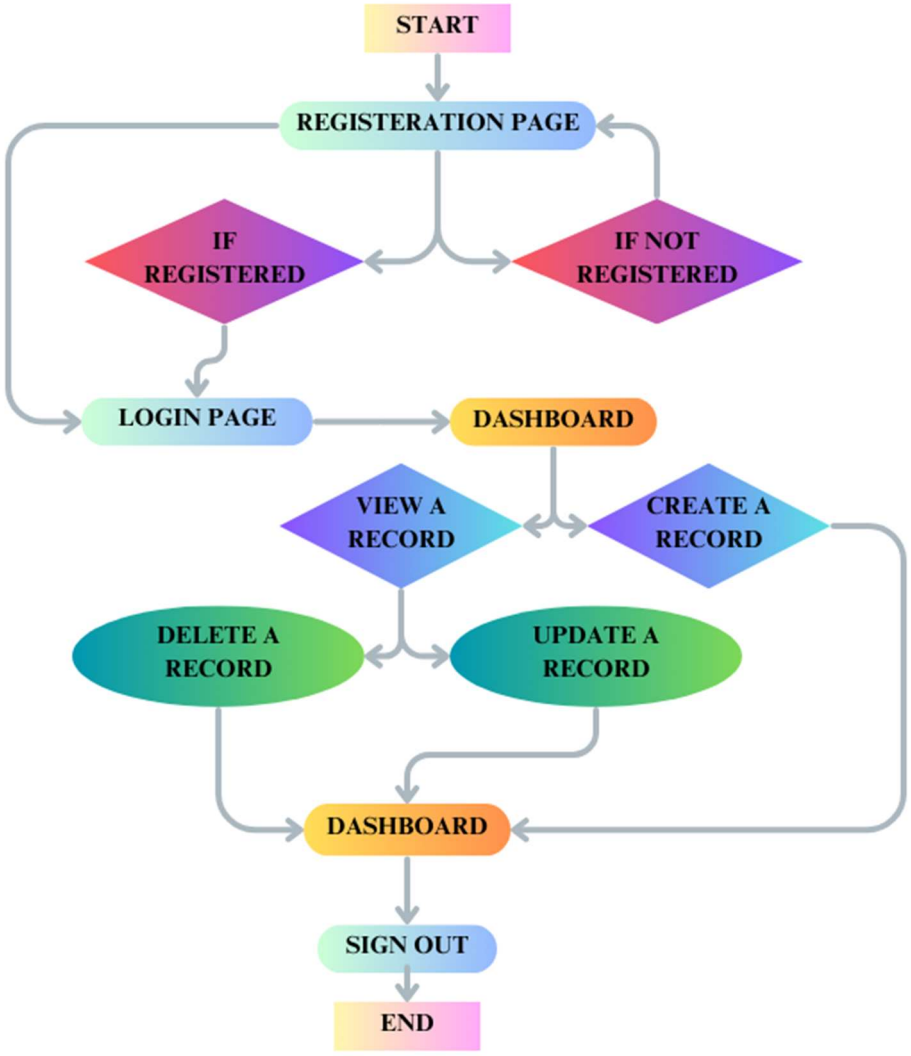
Storage: Minimum 10 GB of free disk space for project files and databases.

Operating System: Compatible with Windows, macOS, or Linux distributions.

Network Connectivity: Stable internet connection for installing dependencies and accessing resources.

CHAPTER 3

FLOWCHART



CHAPTER 4

Technical Details

In the realm of Customer Relationship Management (CRM) systems, leveraging robust technologies is paramount to ensure efficient data management, user interaction, and data security. Python, with its versatile frameworks, offers a compelling option for CRM development. One such framework is Django, renowned for its scalability, security features, and rapid development capabilities. In this document, we delve into the technical intricacies of implementing a CRM project using Django.

Backend Database Management:

A crucial aspect of any CRM system is the backend database management. Django seamlessly integrates with various Database Management Systems (DBMS), including PostgreSQL, MySQL, SQLite, among others. These DBMSs offer scalability, reliability, and robust data storage capabilities, essential for handling vast amounts of customer data. Django's Object-Relational Mapping (ORM) facilitates the interaction between Python objects and the database, simplifying data manipulation and retrieval tasks.

Key Components of a DBMS:

1. Schema Definition: Django's ORM provides a modeling language to define the schema of each database, adhering to the DBMS data model. Despite variations in DBMS implementations, Django abstracts these complexities, offering a unified interface for schema definition.

2. Data Query and Security: Django's ORM enables the formulation of complex database queries and ensures data security by implementing authentication, authorization, and role-based access control mechanisms. This safeguards sensitive customer information and prevents unauthorized access.

3. Transaction Mechanism: ACID (Atomicity, Consistency, Isolation, Durability) compliance is crucial for maintaining data integrity in CRM systems. Django's transaction mechanism ensures atomicity and consistency of database operations, even under concurrent user accesses or system failures.

Structured Query Language (SQL):

SQL serves as the lingua franca for interacting with relational databases in Django. Leveraging SQL, developers can perform various database operations, including data definition (DDL), data manipulation (DML), and data querying (DQL). Django's ORM abstracts SQL complexities, enabling developers to focus on application logic rather than database intricacies.

Conclusion:

In summary, Django provides a robust framework for building CRM systems, offering seamless integration with diverse DBMSs, comprehensive data management features, and enhanced security mechanisms. By leveraging Django's ORM and SQL capabilities, developers can expedite the development process while ensuring scalability, reliability, and data integrity in CRM applications.

CHAPTER 5

CODE

```
from django.shortcuts import render, redirect
from .forms import CreateUserForm, LoginForm, CreateRecordForm, UpdateRecordForm

from django.contrib.auth.models import auth
from django.contrib.auth import authenticate

from django.contrib.auth.decorators import login_required

from .models import Record

from django.contrib import messages
```

```
# - Homepage

def home(request):

    return render(request, 'webapp/index.html')
```

```
# - Register a user

def register(request):

    form = CreateUserForm()

    if request.method == "POST":

        form = CreateUserForm(request.POST)

        if form.is_valid():

            form.save()

            messages.success(request, "Account created successfully!")

            return redirect("my-login")

    context = {'form':form}

    return render(request, 'webapp/register.html', context=context)
```

```
# - Login a user

def my_login(request):

    form = LoginForm()

    if request.method == "POST":

        form = LoginForm(request, data=request.POST)

        if form.is_valid():

            username = request.POST.get('username')
            password = request.POST.get('password')

            user = authenticate(request, username=username, password=password)

            if user is not None:

                auth.login(request, user)

                return redirect("dashboard")

    context = {'form':form}

    return render(request, 'webapp/my-login.html', context=context)
```

```python
# - Dashboard

@login_required(login_url='my-login')
def dashboard(request):

    my_records = Record.objects.all()

    context = {'records': my_records}

    return render(request, 'webapp/dashboard.html', context=context)
```

```python
# - Create a record

@login_required(login_url='my-login')
def create_record(request):

    form = CreateRecordForm()

    if request.method == "POST":

        form = CreateRecordForm(request.POST)

        if form.is_valid():

            form.save()

            messages.success(request, "Your record was created!")

            return redirect("dashboard")

    context = {'form': form}

    return render(request, 'webapp/create-record.html', context=context)
```

```python
# - Update a record

@login_required(login_url='my-login')
def update_record(request, pk):

    record = Record.objects.get(id=pk)

    form = UpdateRecordForm(instance=record)

    if request.method == 'POST':

        form = UpdateRecordForm(request.POST, instance=record)

        if form.is_valid():

            form.save()

            messages.success(request, "Your record was updated!")

            return redirect("dashboard")

    context = {'form':form}

    return render(request, 'webapp/update-record.html', context=context)
```

```python
# - Read / View a singular record

@login_required(login_url='my-login')
def singular_record(request, pk):

    all_records = Record.objects.get(id=pk)

    context = {'record':all_records}

    return render(request, 'webapp/view-record.html', context=context)
```

```python
# - Delete a record

@login_required(login_url='my-login')
def delete_record(request, pk):

    record = Record.objects.get(id=pk)

    record.delete()

    messages.success(request, "Your record was deleted!")

    return redirect("dashboard")
```

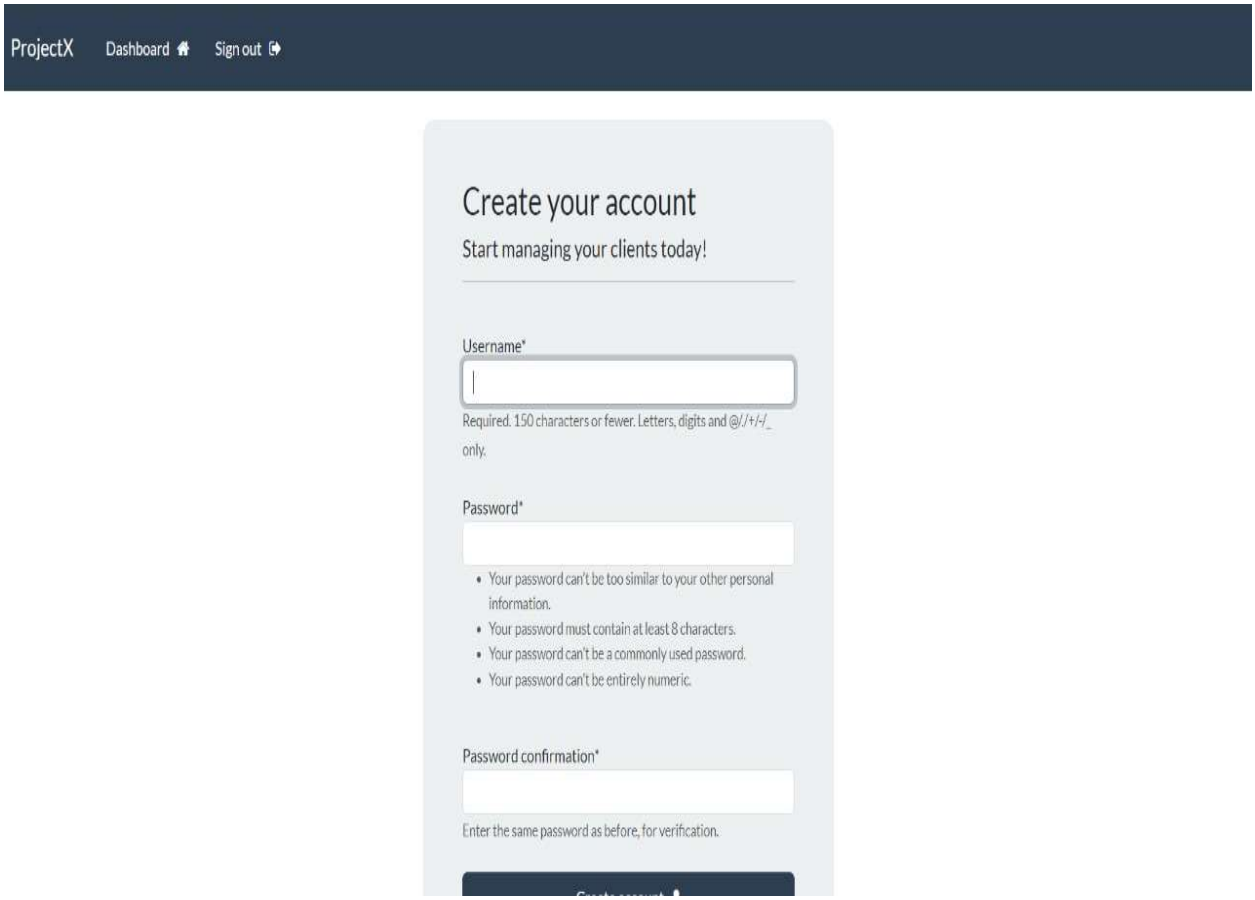
```python
# - User logout

def user_logout(request):
    auth.logout(request)
    messages.success(request, "Logout success!")
    return redirect("my-login")
```

```python
from django.db import models

class Record(models.Model):
    creation_date = models.DateTimeField(auto_now_add=True)
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    email = models.CharField(max_length=255)
    phone = models.CharField(max_length=20)
    address = models.CharField(max_length=300)
    city = models.CharField(max_length=255)
    province = models.CharField(max_length=200)
    country = models.CharField(max_length=125)

    def __str__(self):
        return self.first_name + "   " + self.last_name
```

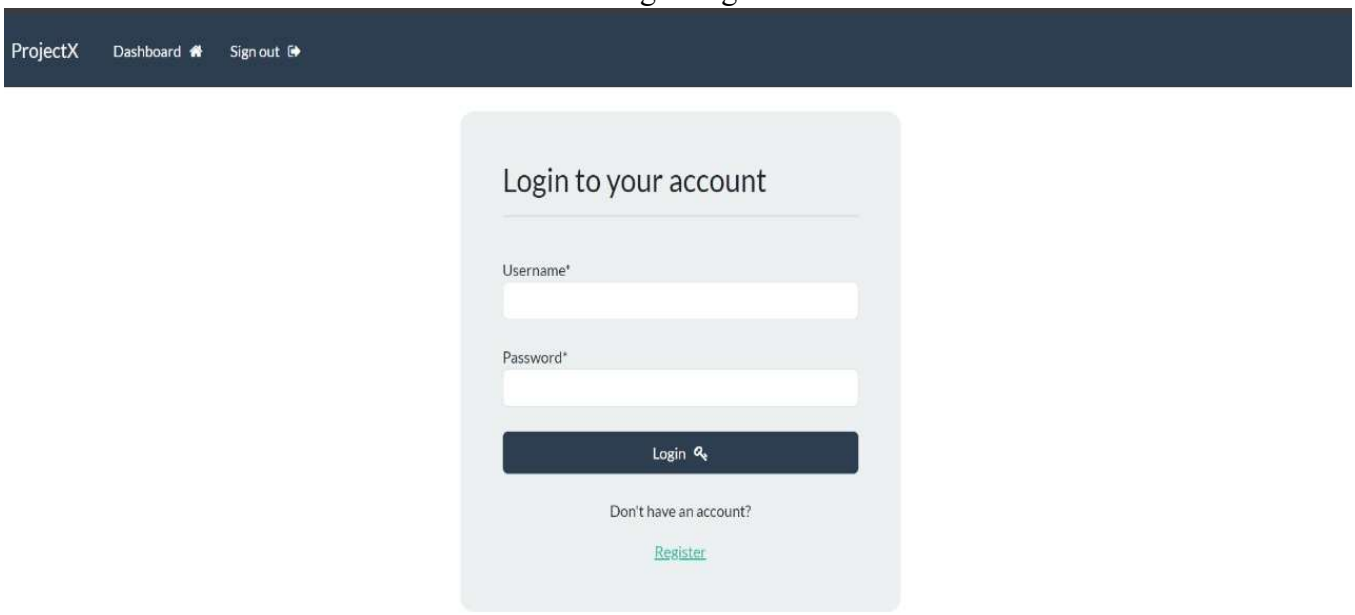
CHAPTER 6

Results (Screenshot)

Registration Page

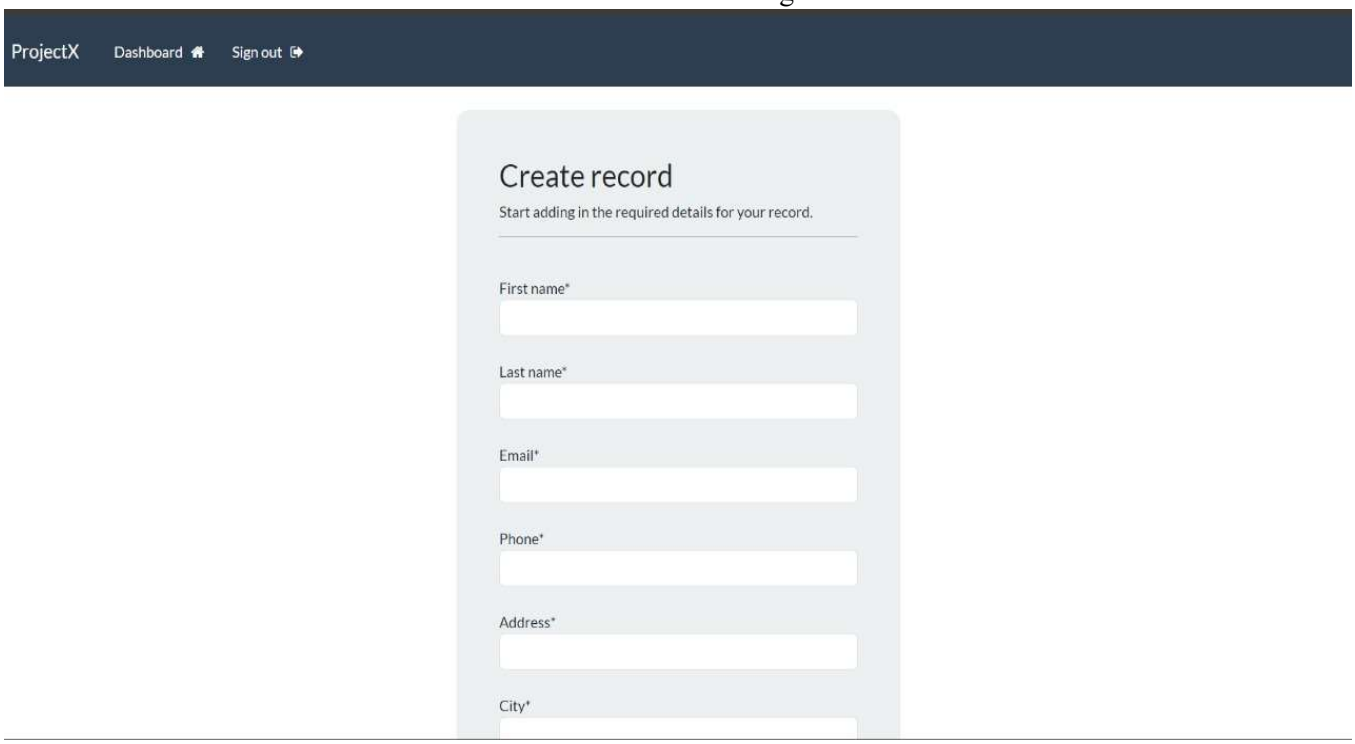


Login Page

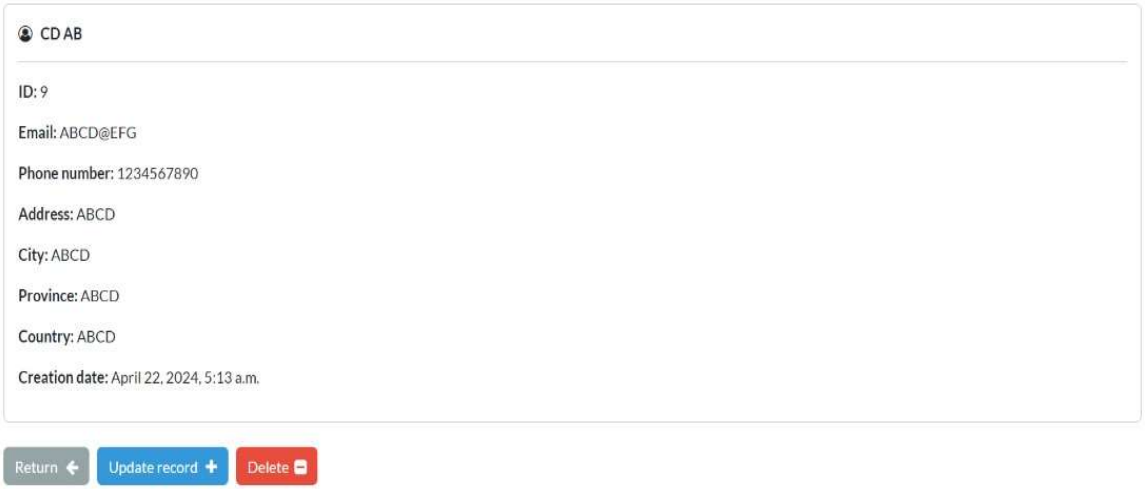


Dashboard

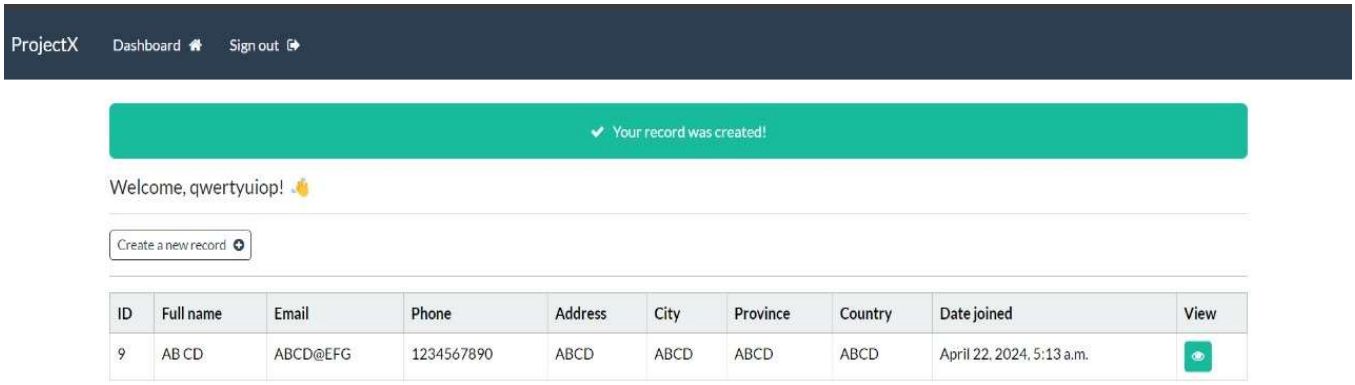
Record Creation Page



Record Details Page



Dashboard After Creating A Record



CHAPTER 7

CONCLUSION

In conclusion, our basic Django CRM project stands as a cornerstone for businesses aiming to optimize their customer relationship management workflows. Through Django's robust architecture, we've crafted a solution that prioritizes scalability, security, and functionality. Our platform empowers businesses to effortlessly manage customer interactions, bolster sales initiatives, and cultivate enduring client connections.

With its user-friendly interface, intuitive navigation, and customizable features, our CRM system ensures users can seamlessly adapt it to their unique requirements. Whether it's tracking leads, scheduling follow-ups, or analyzing customer data, our solution facilitates efficient and effective management of customer relationships.

Looking ahead, our commitment to innovation drives us to continuously enrich the system with new features and enhancements. We pledge to uphold the highest standards of security and performance, safeguarding sensitive data and ensuring optimal user experience.

As we embark on this journey, we invite businesses to join us in revolutionizing customer engagement practices. Together, let's transform the way businesses connect with their customers, unlocking boundless opportunities for growth and success.

CHAPTER 8

Future enhancement

- Integration with External APIs: Enable integration with third-party services for enhanced communication and engagement.
- Advanced Reporting and Analytics: Implement detailed reporting and analytics for data-driven decision-making.
- Automation and Workflow Management: Streamline tasks with workflow automation features.
- Mobile Responsiveness: Optimize the CRM for mobile devices to enable on-the-go access.
- AI and Machine Learning Integration: Explore AI integration for personalized insights and predictive analytics.
- Customer Portal: Develop a self-service portal for customers to manage their accounts.
- Enhanced Security Features: Strengthen security with advanced measures like two-factor authentication and data encryption.

REFERENCES

Django Documentation: https://docs.djangoproject.com/en/5.0/

2. **Font Awesome: https://fontawesome.com/**
3. **Github: https://github.com/**
4. **Youtube: https://www.youtube.com/**