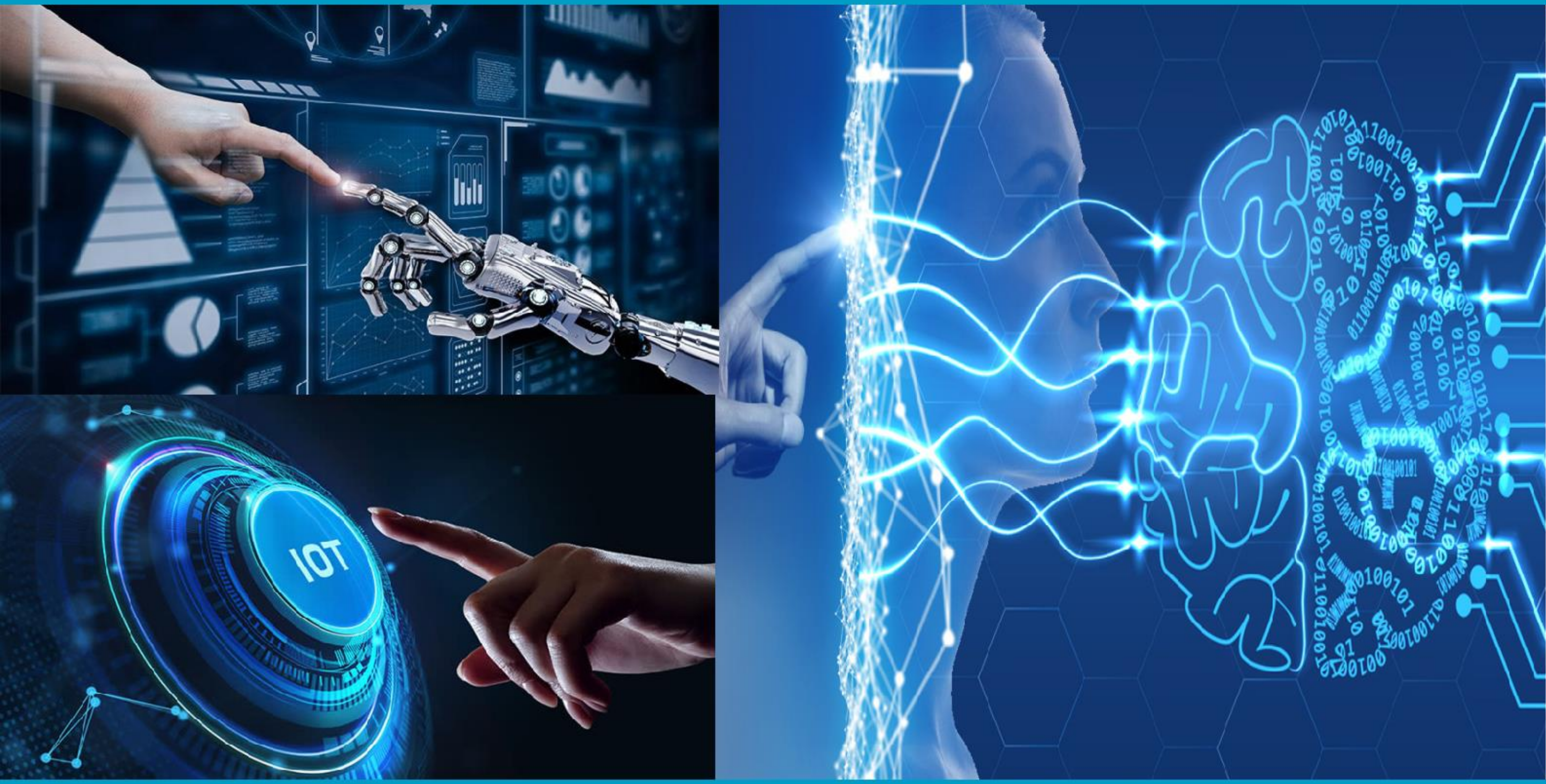
IJMRSETM

e-ISSN: 2395 - 7639



INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING, TECHNOLOGY AND MANAGEMENT

Volume 10, Issue 6, June 2023

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 7.580

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

| ISSN: 2395-7639 | www.ijmrsetm.com | Impact Factor: 7.580| A Monthly Double-Blind Peer Reviewed Journal |

| Volume 10, Issue 6, June 2023 |

Django Python Framework

Shivani Sadashiv Ubale, Shripad S.Bhide

Student, Department of M.C.A, P.E.S Modern College of Engineering, Pune, India

Assistant Professor, Department of M.C.A, P.E.S Modern College of Engineering, Pune, India

ABSTRACT : This Research Paper Study About The Web Frame Work Use In Python Known As Django. This research paper delves into Django. The objective of this paper is to explore the installation process and project creation using Django on our system. Additionally, we aim to assess the efficiency and user-friendliness of this framework compared to others. Also How to Create App, Views. Basically This Paper Cover Django Framework Learning Interactive and Easy. This Research Paper Delves Into The Simplicity And Speed Advantages Offered By Django, A Python Web Framework, In Comparison To Other Frameworks.
The Paper Explores The Key Features And Design Principles Of Django That Contribute To Its Simplicity, Such As Its High-Level Abstractions And Code Reusability. Additionally, It Investigates The Performance Optimizations Implemented In Django That Enable Fast And Responsive Web Applications. By Analysing Real World Examples And Benchmarking Data, The Research Provides Empirical Evidence Of Django's Superiority In Terms Of Simplicity And Speed When Compared To Other Frameworks. The Findings Of This Study Highlight The Significant Advantages Of Django For Developers Seeking A Streamlined And Efficient Web Development Experience.

I. INTRODUCTION

Django, a Python-based web application framework, utilizes the Model-View-Template structure. Django's high demand in the current market is due to its efficiency in reducing development time compared to other frameworks. In The World Of Web Development, Selecting A Framework That Balances Simplicity And Speed Is Crucial For Efficient And Productive Application Development. Django, A Python Web Framework, Has Gained Recognition For Its Simplicity And Performance Advantages Over Other Frameworks. This Research Paper Aims To Provide An In-Depth Analysis Of Django's Simplicity And Speed In Comparison To Alternative Frameworks, Offering Valuable Insights For Developers Seeking The Most Effective Tool For Their Web Development Projects.

II. HISTORY

2003: Django's Development Begins
2005: Open-Source Release
2006: Django Software Foundation Established
2008: Django 1.0 Release
2011: Django's Success In Industry
2013: Django 1.5 And Native Support for Python 3
2015: Django 1.8 And Django Channels
2017: Django 2.0 And Django Rest Framework
2020: Django 3.0
And Beyond Django Continues To Evolve, With Regular Releases, Updates, And Active Community Involvement. Its Robustness, Scalability, And Ease Of Use Have Solidified Django's Position As One Of The Leading Web Frameworks In The Python Ecosystem.

Applications

1. Instagram-Like Application:
Django Can Handle User Registration, Authentication, And Profile Management. It Can Manage User-Generated Content, Including Image And Video Uploads, With Features Such As Resizing, Filtering, And Storage. Django Orm Can Handle Relationships Between Users, Posts, Comments, And Likes. It Can Facilitate Social Features Like Following/Follower Relationships And User Interactions. Django's Template Engine And Front-End Frameworks Can Be Used To Create Visually Appealing Interfaces.

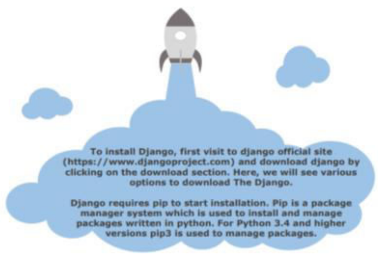
2. Spotify-Like Application:Django Can Manage User Accounts, Authentication, And Authorization For Music Streaming Services. It Can Handle Music Metadata And Organize Tracks, Albums, And Playlists Using Its Data Modeling Capabilities. Django's Orm Can Facilitate Complex Queries For Music Recommendations And Personalized Playlists. It Can Handle User Interactions Such As Likes, Comments, And Sharing. Django's Media Handling Features Can Store And Serve Audio Files For Streaming.

3. YouTube-Like Application:Django Can Handle User Registration, Authentication, And Profile Management. It Can Manage Video Uploads, Including Storage, Processing, And Transcoding. Django's Orm Can Handle Relationships Between Users, Videos, Comments, And Subscriptions. It Can Implement Features Like Search, Recommendations, And Trending Videos. Django's Template Engine Can Render Video Pages, Playlists, And User Channels.

4. Dropbox-Like Application:
Django Can Handle User Accounts, Authentication, And File Management. It Can Manage File Uploads, Storage, And Organization In A Hierarchical Structure. Django Orm Can Handle Sharing And Permissions For Files And Folders. It Can Implement File Versioning, Syncing, And Collaborative Features. Django Template Engine Can Render File Browsers And User Interfaces For File Management.

5. Google Chrome Extensions:
Django Can Handle User Registration And Authentication For Extension Developers. It Can Manage User Data, Settings, And Preferences. Django Orm Can Handle Storing And Retrieving Extension-Related Data. It Can Provide Apis And Endpoints For Communication Between The Extension And Server. Django's Template Engine Can Render Administrative Interfaces For Extension Management

III. INSTALLATION & CREATION OF NEW PROJECT



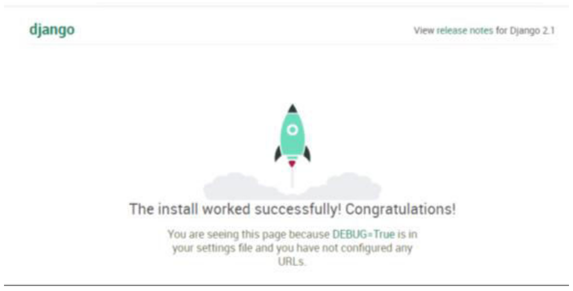
To install Django, first visit to django official site (https://www.djangoproject.com) and download django by clicking on the download section. Here, we will see various options to download The Django.

Django requires pip to start installation. Pip is a package manager system which is used to install and manage packages written in python. For Python 3.4 and higher versions pip3 is used to manage packages.

Steps:

1. Create Folder Open With Vscode
2. Virtual Environment: Virtualenv Name Of Virtual Environment
3.Activate Environment: Name of Virtual Env\Scripts\Activate
4.Install Django: Pip Install Django

5.Create Project: Django Admin Start Project Project Name
6.Create App For Project: Py Manage.Py Startapp Appname
7.Register The App In Main Project Setting File
8.Hit Command For Model/Database:Py Manage.Py Makemigration
9.Database/Table Creation: Py Manage.Py Migrate
10.Create Super User.Admin Panel:Py Manage.Py Create Super User
11.Runserver:Py Manage.Pyrunserver



The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Why Django

1. Open Source Development at No Cost
2. Fully Scalable Capabilities
3. Development Fast
4. High Secure
5. Built In Administration Portal
6. Rapid Development And Reusability
7. Highly Abstraction

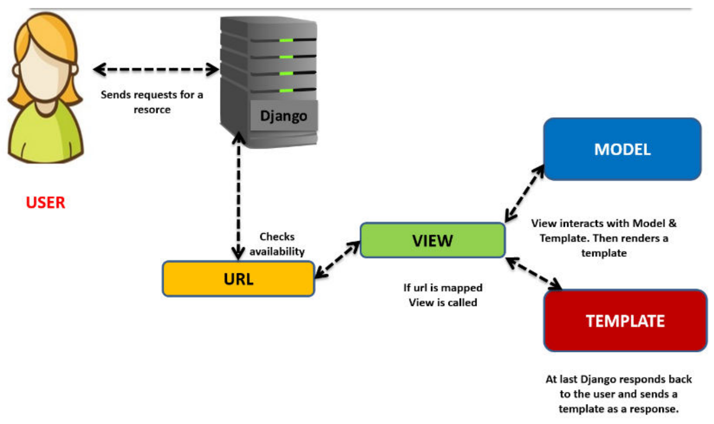
MVT Structure of Django

Model:

It Defines The Data Schema, Relationships, Represents The Data Structure And Business Logic And Operations Performed On The Data. Django defines Python classes that inherit from the Django.Db.Models.Model class. Each attribute in the class represents a field in the database table, and the model offers methods for querying, creating, updating, and deleting records Class. Each Class Attribute Represents A Field In View:The View Handles The Business Logic And Acts As An Intermediary Between The Model And The Template. It Receives User Requests, Interacts With The Model Retrieve Data Or Perform Operations, And Prepares The Necessary Data For Rendering In The Template. Views Are Functions Or Classes That Receive Http Requests And Return Http Responses. Views Can Access Data From The Model, Process It, And Pass It To The Template.

Template:

It Defines How The Data Is Displayed And Rendered To The User. Templates In Django Are Typically Html Files That Include Placeholders And Template Tags To Dynamically Insert Data. Template Tags Allow For Logic And Control Flow Within The Template, Enabling Conditional Rendering, Loops, And Data Manipulation. Templates Can Access Data Passed From The View And Display It In A User-Friendly Format.


Sends requests for a resorce
Django
USER
MODEL
View interacts with Model & Template. Then renders a template
Checks availability
VIEW
URL
If url is mapped View is called
TEMPLATE
At last Django responds back to the user and sends a template as a response.

Flow Of Request In MVT:
User Sends A Request To A Specific Url Defined In The Django Application's Url Configuration.The Url Is Matched With A Corresponding Entry In The Url Configuration, Which Maps It To A Specific View.
The View Receives The Request, Interacts With The Model To Fetch Or Modify Data, And Prepares The Data For Rendering. The View Then Passes The Data To A Template For Rendering, Along With Any Additional Context.
The Template Receives The Data And Renders It According To The Defined Html Structure And Logic.
The Resulting A Response send back to browser for display the user.

Creating App In Django
Method 1: execute the command: Python Manage.Py Startapp <Appname>.
Method 2:execute the command: Django_Admin Startapp <Appname>.
Now we obtain the directory structure of the app folder. To make the app functional, we need to specify its inclusion in the settings.py file located inside the project directory.
After completing this step, the app is created, but we need to define the URL in our main project. The provided URL will redirect to the corresponding app.Steps:
Navigate to the project directory -> urls.py and add the following code at the beginning of the file.Now, in the URL code, we must specify the app's name for the URL using the given code.We utilize the MVT (Model-View-Template) approach to create models, views, templates, and URLs inside the app, which are automatically added to the main project.Django Model
Django models offer a database interface for creating, retrieving, updating, and deleting data from a map. They provide essential fields and behaviours for storing data. Typically, each model corresponds to a data table. Django models utilize a single SQL database within the Django framework. Models simplify tasks and structure tables into models, often sharing the same data table.

```
from django.db import models

      class c1 (models.Model):

               n1 =
models.CharField(max_length=30)

               n2   =
models.CharField(max_length=30)
```

Django Crud
In crud you can update,create,read,delete data same as database the crud is useg in django just lik a database
Creating A Model
This is a genuine example of model creation. Our category comprises four attributes, each represented as a field in the table.

```
models.IntegerField()

 class Meta:

 db_table = "examplel"
```

Django Views
In Django, views are a collection of function classes within the app directory. When we render the website, the view is what we see in the browser, serving as the user interface.
Definition As Per Django Documentation:
Views Are Responsible For Processing The User's Request And Generating The Appropriate Response.

In Django, Views Are Used To Define The Logic And Behaviour Of Different Pages Or Endpoints In A Web Application. They Handle Tasks Such As Retrieving Data From A Database, Rendering Html Templates, Processing Form Submissions, And Returning Appropriate Http Responses.

A Simple View Function:

```
from django.http import HttpResponse

import library_name

    def function_name(request):


n1=library_name.library_name.request_n
ame(:

        html="<html><body>Some Random
text according to need
      %s.</body></html>"%n1

       return HttpResponse(html)
```

Let's go through the steps
1. The first line imports a class from the Django HTTP module
2. The second line imports the Python datetime library.
3. The third line defines a function called cur_datetime. Can you provide more details about how the function should look like?
4. Lines 5 to 7 are responsible for defining the structure of a web page within an HTTP response.
5. Lines 5 to 7 determine the structure of a web page, and the view returns an HTTP response in the form of an HttpResponse.

Two Types of Views in Django

1. Functional Base View **2**.Class Base View

Function-Based

is a Python function that handle response and reuest. This response can take various forms such as HTML content, an XML document, or a 404 error page. All views functions have an HTTP request object as their first parameter.

```
def function_name(request):

     return
     HttpResponse('html/variable/text')
```

When working with multiple files in Django, it is crucial to choose the appropriate file for writing the code. This code is typically written in the view.py file and is later utilized by the urls.py file of the project, which defines the URL names associated with each function.

Class-Based Views:

It offers an alternative approach to execute views in Django using Python objects. These class-based views do not replace the traditional function-based views, but they provide distinct differences and advantages. Here is an example of a class-based view that handles an HTTP GET request:

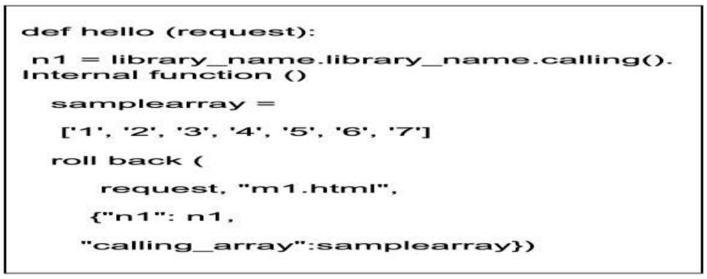
```
from django.http import HttpResponse

from django.views import View

    class new_view(View):

        def get(self,request):

            return  HttpResponse('res')
```

Django Templates
Django provides a straightforward approach to creating dynamic HTML templates. These templates are built using HTML, CSS, and JavaScript. They are efficiently managed and generate HTML pages that are visible to end users, forming the structure of a website.

The template function primarily requires three parameters:
1. Request - The initial request made.
2. Template path - It refers to the template_dirs option in the project.py file, which can be modified according to the project's variables.
3. Parameters dictionary - A dictionary containing all the necessary elements for the template. We can utilize the locals() function .

Django Template (DTL) Language
Django introduces a concise language for defining the user interface, the front-end part of a program. A Django template is a Python thread that incorporates Django template language. The template engine recognizes and translates specific constructs. Templates are equipped with context and offer flexibility in terms of values and tags.

Django serves as a framework that allows clear separation between Python and HTML. Python logic resides within views, while HTML content resides within templates. Django effectively connects the two using a dedicated template language to achieve optimal performance.
Tags:



```
def hello (request):
 n1 = library_name.library_name.calling().
Internal function ()
   samplearray =
    ['1', '2', '3', '4', '5', '6', '7']
   roll back (
        request, "m1.html",
      {"n1": n1,
     "calling_array":samplearray})
```

Since HTML is a static markup language, it is not possible to directly include Python code within HTML because web browsers are incapable of interpreting Python as it is a programming language.

Django Forms
HTML forms play a crucial role as fundamental components of websites. Django offers a Forms class specifically designed for creating HTML forms. While it is possible to accomplish all the tasks in Django using advanced HTML, Django's efficiency, particularly in form validation, sets it apart. Once you become familiar with Django, working with HTML forms becomes a thing of the past.
Working with Django forms involves three key steps. The first step involves preparing and structuring the data to be ready for rendering, ultimately resulting in the creation of an HTML form. The second step entails receiving and processing submitted forms.
By utilizing Django's Forms class, the process of creating Django forms becomes more streamlined. To achieve this, a new file named forms.py is created within the application folder, where the necessary code is written.

Create Django Forms Using Form Class

1. Create a new file named forms.py within the application folder.
2. Inside the forms.py file, write the necessary code to define your form using a class.

Here is an example code snippet for creating a Django form using a class:

```
from django import forms

    class sample_form(forms.Form):

                id1=forms.CharField()

                #here length is not required

               id2=forms.CharField()
```

After Rendering Form Look Like

```
<tr>
     <th>
   <label for="id1">label: </label>
    </th>

    <td>
              <input  type="text"  name="id1"
required id="id1">
    </td>
</tr>
```

```
<tr>
    <th>
      <label for="id2"> id2 :</label>
    </th>

     <td>
       <input  type="text"  name="id2"
         required id="id2" >
     </td>
</tr>
```

Display Form To User

.Django provides a Forms class that is utilized for creating HTML forms. While it is possible to accomplish all tasks in Django using advanced HTML, Django proves to be more efficient, particularly in terms of form validation. Once you experience the joy of working with Django, HTML forms become a distant memory.

Related forms consist of three essential parts. The first involves preparing and organizing data to make it ready for rendering. Then, the data is transformed into an HTML form. Finally, the submitted forms are received and processed accordingly.

Create Django Forms Using Form Class

To create a Django form using a class, you need to create a new file named `forms.py` inside the application folder. Then, write the provided code in the `forms.py` file:

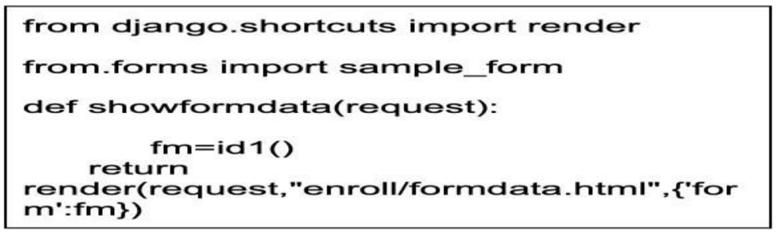
Ensure that you save the `forms.py` file after writing the code.

```
from django import forms

    class sample_form(forms.Form):

                id1=forms.CharField()

                #here length is not required

               id2=forms.CharField()
```

After Rendering Form Look Like

```
<tr>
    <th>
      <label for="id2"> id2 :</label>
    </th>

     <td>
       <input  type="text"  name="id2"
         required id="id2" >
     </td>
</tr>
```

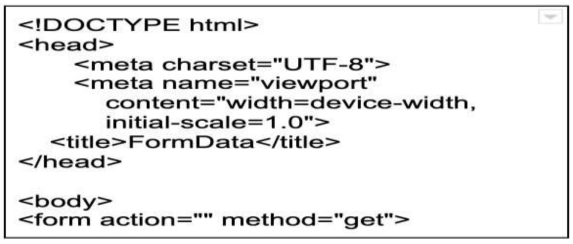
Display Form To User
whene
When we display form first create an object of the form class then pass the object



```
from django.shortcuts import render
from.forms import sample_form
def showformdata(request):
        fm=id1()
    return
render(request,"enroll/formdata.html",{'for
m':fm})
```

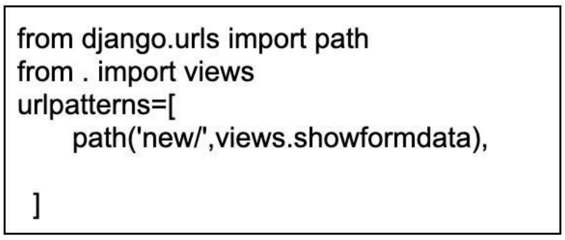
After That, Passing The Context Form Into Form Data Template Its Look Like This:

Templates/Enrol/Form Data,Http



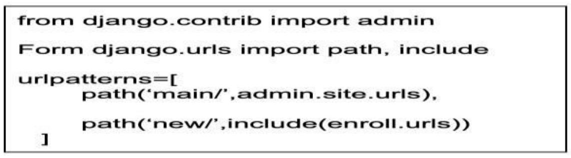
```
<!DOCTYPE html>
<head>
      <meta charset="UTF-8">
      <meta name="viewport"
         content="width=device-width,
         initial-scale=1.0">
   <title>FormData</title>
</head>

<body>
<form action="" method="get">
```

Urls.Py file



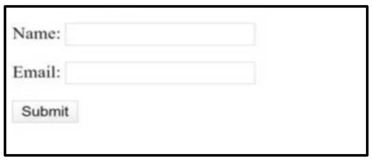
```
from django.urls import path
from . import views
urlpatterns=[
     path('new/',views.showformdata),

  ]
```

Now add the mentioned URL



```
from django.contrib import admin
Form django.urls import path, include
urlpatterns=[
        path('main/',admin.site.urls),
        path('new/',include(enroll.urls))
   ]
```

Now It Time To Save and Run



Future Enhancement:
Django Has A Vibrant And Active Community That Continuously Works On Enhancing The Framework. While I Cannot Predict The Exact Future Enhancements, There Are Several Areas Where Django May See Further Improvements And Advancements Based On Current Trends And Community Discussions. Here Are Some Potential Areas Of Future Enhancement For Django:
Performance Optimization: Django May See Further Optimizations And Improvements In Terms Of Performance. This Could Involve Enhancements In Query Optimization, Caching Strategies, And Reducing Unnecessary Database Queries. Additionally, Advancements In Asynchronous Programming And Integration With Tools Like Django Channels May Further Improve The Scalability And Performance Of Django Applications.
Integration With Modern Web Technologies: As Web Technologies Continue To Evolve, Django May Integrate More Seamlessly With New And Emerging Tools And Frameworks.

Enhanced Support For Api Development: With The Increasing Popularity Of Apis (Application Programming Interfaces), Django May Continue To Improve Its Support For Building Robust And Scalable Apis. This Could Involve Enhancements To Django Rest Framework, Providing Better Support For API Versioning, Graph Ql Integration, API Documentation Generation, And Handling Real-Time Api Communications.
Improved Front-End Development Workflow: Django's Template Engine, While Powerful, May See Enhancements To Streamline The Front-End Development Workflow. This Could Involve Better Integration With Front-End Build Tools Like Web Pack or Parcel, Support For Modern Css Pre Processors And Frameworks, And Improved Asset Management And Bundling.
Better Support For Micro Services And Distributed Systems: As The Trend Of Building Micro Services And Distributed Systems Continues To Grow, Django May See Improvements In Supporting These Architectures. This Could Involve Enhancements In Service-To-Service Communication, Support For Message Queues, Integration With Service Discovery Tools, And Better Management Of Distributed Transactions.
Ai/Ml Integration: With The Increasing Demand For Ai (Artificial Intelligence) And Ml (Machine Learning) Applications, Django May Provide Better Integration And Tooling For Incorporating Ai/Ml Models Into Django Projects. This Could Include Features Like Model Training And Deployment, Integration With Popular Ml Libraries, And Simplifying The Integration Of Ai/Ml Capabilities Into Web Applications.
Security Enhancements: Django Has A Strong Focus On Security, And Future Enhancements May Further Strengthen Its Security Features. This Could Include Improvements In Security Measures Against Emerging Threats, More Granular Control Over Security-Related Settings, Enhanced Protection Against Common Vulnerabilities, And Better Support For Secure Coding Practice

IV. CONCLUSION

In summary, Django is an incredibly powerful and flexible web framework that streamlines the development of robust and scalable web applications.It Offers Numerous Advantages Such As Rapid Development, A Comprehensive Set Of Features, Scalability, Security, And A Vibrant Community. While Django Offers Significant Benefits, There Are Also Potential Disadvantages To Consider, Such As Its Learning Curve, Overhead, Limited Flexibility, And The Need For Python Proficiency. Overall, Django Provides A Solid Foundation For Web Application Development, Empowering Developers To Create Scalable, Secure, And Feature-Rich Applications. Its Active Community, Extensive Documentation, And Vast Ecosystem Of Reusable Packages Contribute To Its Popularity And Ensure Ongoing Support And Innovation.

REFERENCES

https://www.djangoproject.com/
https://ijcrt.Org/Papers/Ijcrt2105197.Pdf
Https://Ieeexplore.Ieee.Org/Document/6778632
https://dl.acm.org/doi/10.5555/1747137.1747166
https://www.jetir.org/papers/JETIR2304856.pdf