DJANGO APPLICATION FOR PET CARE SHOP

ABSTRACT

This project focuses on the creation of a web application for a pet shop using the Django web framework. The application is designed to manage and display information about available pets, including their names, species, breeds, ages, and prices. The development process involves setting up a Django project and app, defining models for the pet data, creating views to display the information, and configuring URLs for navigation.

The core model, "Pet," is defined with attributes such as name, species, breed, age, and price. The project follows the Model-View-Controller (MVC) architectural pattern provided by Django, promoting code organization and separation of concerns. Views are implemented to render HTML templates displaying a list of available pets.

Templates are created using Django's template language, providing a dynamic and data-driven presentation layer. The application's URLs are configured to map to specific views, allowing users to navigate through the available pet information. The project is structured to be easily expandable, with the potential for adding features such as user authentication, a shopping cart, and checkout functionality.

Through this development process, the project aims to showcase the efficiency and flexibility of Django in building web applications, particularly in the context of a pet shop where data management and dynamic content presentation are essential. The resulting web application provides a user-friendly interface for customers to explore and learn about the diverse range of pets available for adoption or purchase.

INTRODUCTION

1.1 COMPANY PROFILE

Retech Solutions Pvt Ltd is ISO 9001 : 2015 Certified Company. At present, our organization oversees three prominent startups that have originated from our flagship venture,
Retech Projects. Retech Projects caters to engineering students by providing assistance and support in their coursework internships and project work, and conducts all research activities for our affiliated enterprises. **Retech Lasers** focuses on the manufacture of machinery which utilizes laser technology as its core component. Additionally, we are currently engaged in concurrent efforts to design and develop BLDC motors, spearheaded by **Retech Motors**. Our organization remains committed to providing opportunities to young graduates, and we encourage interested parties to join us in our mission.Retech Solutions are constantly thriving to create an impact on society through education and innovations. As we are interested in multi-disciplinary engineering fields we have started working on various electro-mechanical systems opera-table by computer software. We are recruiting engineering graduates from various departments.

1.2 Our Mission

" To build self-sustainable startups in the fields of machinery and electric drive systems."

1.3 Our Vission

"To build a strong network under one roof for machines, rob"

INTRODUCTION

Full Stack Development refers to the art and science of building both the client-side (front end) and server-side (back end) of a web application. It involves working with a combination of technologies to ensure a seamless user experience, and Django stands out as a powerful framework in this multifaceted domain.

At the heart of Full Stack Development lies Django, a high-level Python web framework that empowers developers to build web applications quickly and efficiently. With its "batteries-included" philosophy, Django provides an all-encompassing toolkit for tasks ranging from database management to user authentication, making it an ideal choice for Full Stack developers.

The story of Full Stack Development with Django begins with the evolution of web technologies. From the early days of static web pages to the dynamic and interactive applications we see today, Django has played a pivotal role in shaping the landscape. Its inception in mention the release year of Django marked a significant milestone, and since then, it has continued to evolve in response to the ever-changing demands of the web development ecosystem.

As we delve into Full Stack Development with Django, it's crucial to understand the key components and concepts that form the framework's foundation. From models and views on the back end to templates and dynamic interfaces on the front end, each element contributes to the creation of sophisticated web applications.

The significance of Full Stack Development with Django becomes evident when we examine its real-world applications. Whether you're

building e-commerce platforms, content management systems, or data-driven applications, Django provides the tools and structure needed to bring your ideas to life on the web.
Beyond its technical prowess, Full Stack Development with Django thrives on a vibrant community of developers and enthusiasts. Collaboration and knowledge-sharing are at the core of this domain, and as you navigate through the intricacies of Django, you'll find a wealth of resources, forums, and communities eager to support your journey.

OVERVIEW

The Pet Shop Management System is a web application developed using Django, a high-level Python web framework. The project aims to provide pet shop owners with a user-friendly interface to manage various aspects of their business, as well as offering customers a convenient platform to explore and purchase pet-related products. The system is designed to handle pet data, including details about available pets, their breeds, and associated animal data. Users, both customers and administrators, can interact with the system through a series of well-defined use cases. Customers can browse products, add items to their cart, manage their shopping cart, proceed to checkout, view and edit their profiles, and review their order history. On the other hand, administrators have specialized use cases for managing animal data, allowing them to add, edit, and delete information about the pets available in the shop. The implementation involves the use of Django models for data representation, views to handle user requests, and templates for rendering the web pages. While this overview provides a foundational structure, additional features such as user authentication, security measures, and further user interface enhancements can be integrated to meet specific business requirements and improve the overall functionality of the Pet Shop Management System.

EXISTING SYSTEM

This paper presents the development of the Pet Shop Management System (PSMS) for Klinik Veterinar & Surgeri Jawhari. PSMS is able to manage the pet information such to produce the pet birth certificate and the pet vaccine schedule. There are various things and topics which are important for the pet industry worldwide such as their health, vaccines, food as well as keeping them happy and giving them a friendly environment to stay in. The management system of pets helps with all of these things as it provides the pet owners with their daily needs and also provides medical help. It has powerful feature extraction and learning capabilities. Because of its various advantages, it has been applied in many fields

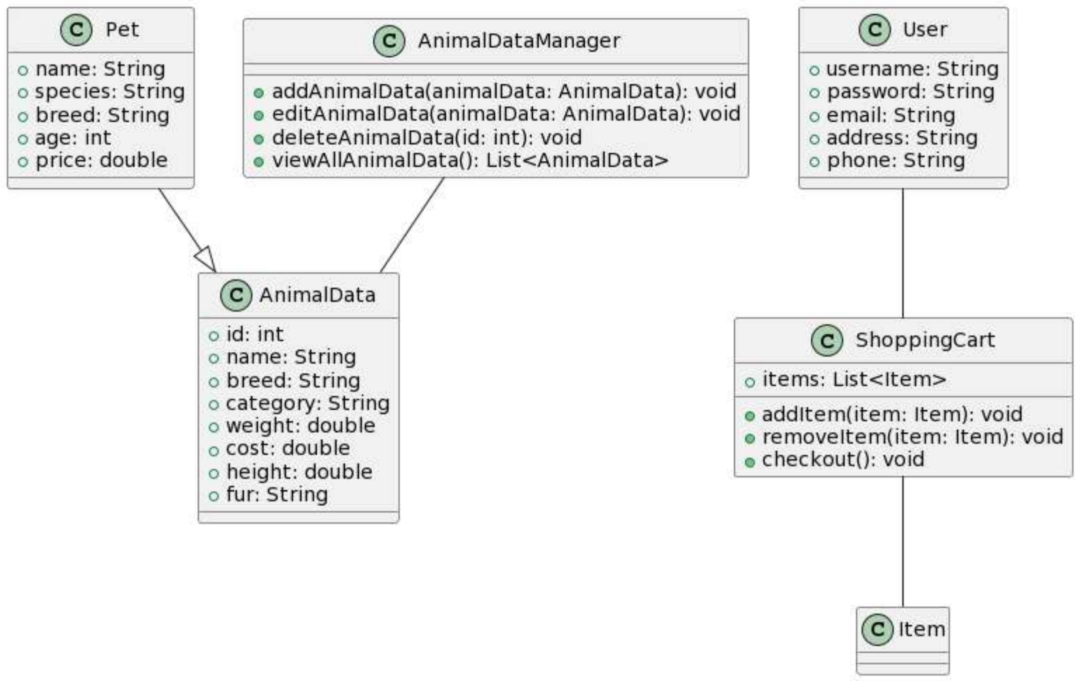
PROPOSED SYSTEM

Shop Management System interface to a pet shop owner to manage his petshop activities. This web application provides the users an option to shop and manage the daily needs of their household pets. There are a total of 6 menus on our website which includes home page, about, shop, services, plan and the contact page.

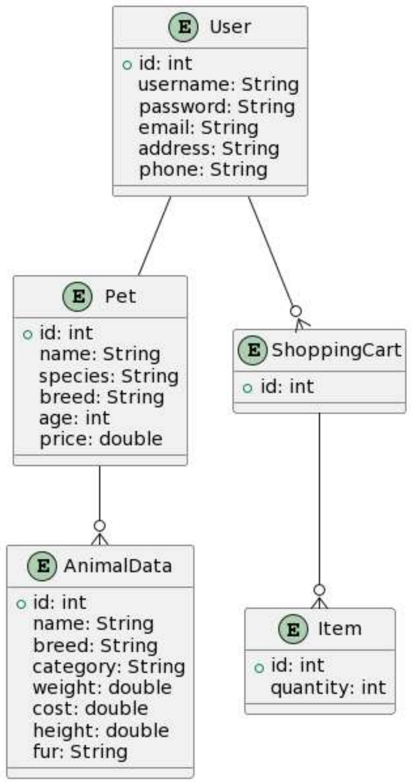
SCOPE OF THE PROJECT

We have developed a very user friendly interface for our website which provides a hassle freeway throughout your time spent with us. It starts off with the home page including a theme of dogs and their products. We have included a lot of pet food items as well as accessories for pet grooming and enjoyment. After that you can add your favourite products to cart or wish-list and proceed with the shopping. There are food items from the top most pet food manufacturers in the world. The customer can search for the type of breed they want and also look at the details of that breed including their health. After that they can also enter animal data of other breeds, and next the customer can also save their contact information for future use. The user can add animal name, breed, category, weight-age, cost, height, and fur of the animal. Next, after adding the animal data, he/she can view all list of the pet animals stored in the pet shop management system. If the user wants to edit an animal data, simply click on the animal, and then an update button will appear which will instantly allow the user to enter new details and update it. In order to delete the data, the user has to first enter animal id and then click delete to confirm and delete the pet data. There are a lot of more tasks we will be adding in the future and make this the best website having the best user interface to work on.

CLASS DIAGRAM



ER DIAGRAM



MODULE DESCRIPTION:

1. Home Page:

The home page serves as the virtual storefront's entrance, featuring an aesthetically pleasing theme centered around dogs and their products. It provides users with a visual introduction to the pet shop and serves as a gateway to the various sections of the website. Users can explore featured pet products and quickly navigate to other pages for further information or shopping.

2. About Page:

The about page acts as an informative hub, offering users insights into the pet shop's background, mission, and values. This section is dedicated to providing a detailed overview of the pet shop's history, its commitment to pet care, and any unique aspects that set it apart from other pet-related businesses.

3. Shop Page:

The shop page is the primary module for users to engage in shopping activities. Here, customers can browse and purchase a variety of pet products, including food items and accessories. The functionality allows users to add products to their cart or wishlist, search for specific breeds, and access detailed information about each breed, including crucial health details.

4. Services Page:

The services page showcases the diverse range of offerings beyond product sales. Customers can learn about grooming services, health checkups, and any additional services provided by the pet shop. This module aims to highlight the holistic approach the pet shop takes in caring for customers' pets.

5. Plan Page:

The plan page introduces users to various plans or memberships that can enhance their experience and provide added benefits. This section outlines different plans, subscription options, or loyalty programs, detailing the advantages and potential cost savings for users who choose to engage with the pet shop on a more extended basis.

6. Contact Page:

The contact page serves as a communication hub, offering users a direct means to get in touch with the pet shop for inquiries or assistance. It features a contact form, location details, and possibly a chat option for customer support, ensuring a convenient and accessible way for users to seek help when needed.

7. Animal Data Management:

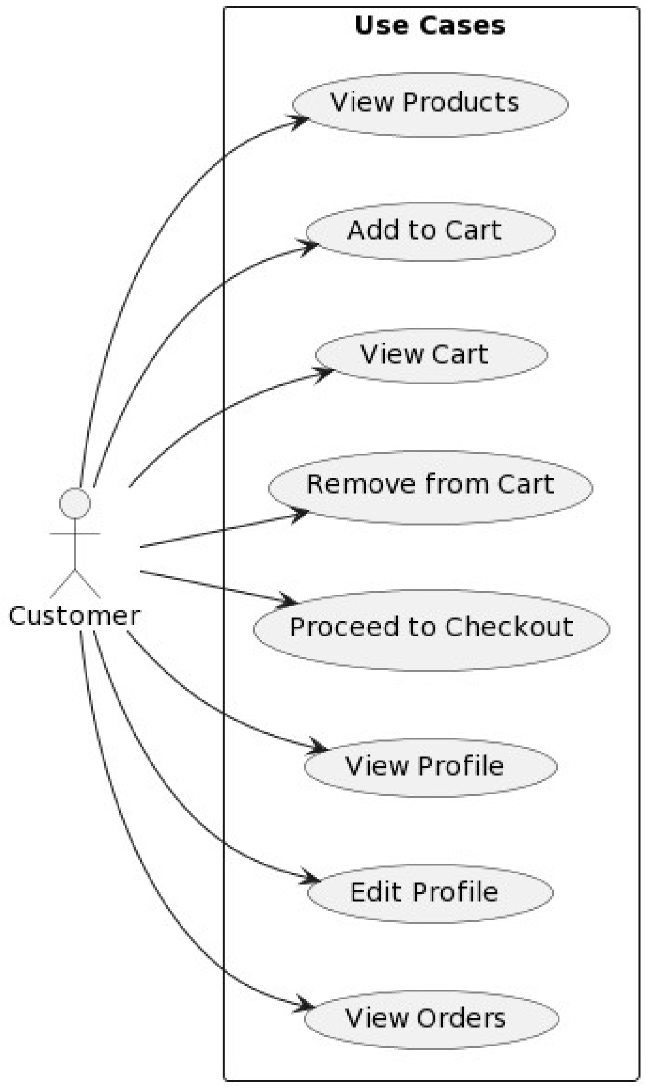
The animal data management module empowers the pet shop owner to effectively organize information about animals available for sale or adoption. This functionality allows for the addition of comprehensive animal data, including name, breed, category, weight, cost, height, and fur details. Users can seamlessly view, edit, and delete entries, ensuring accurate and up-to-date information.

8. User Authentication (Future Enhancement):

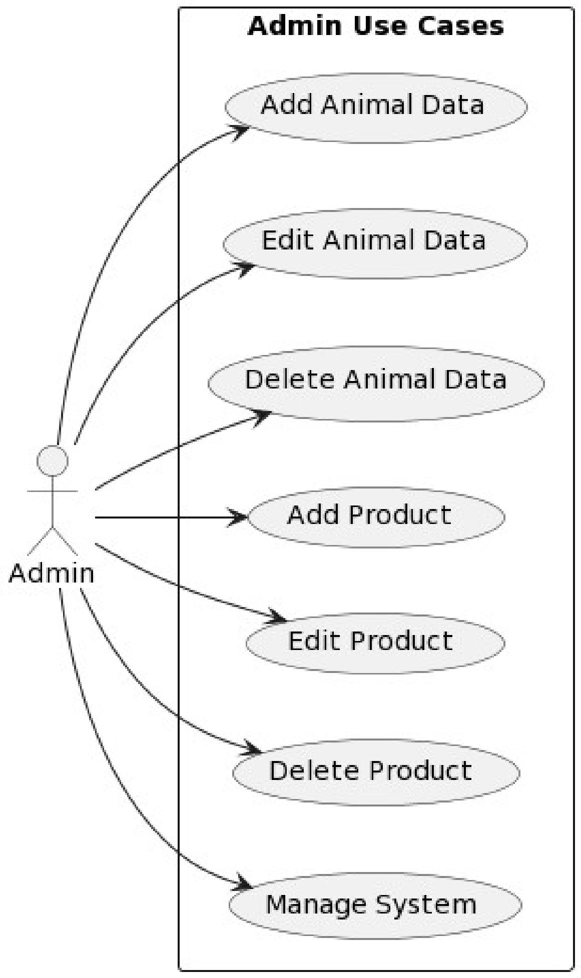
The user authentication module, slated for future enhancement, will introduce user accounts to personalize the customer experience. Users can create accounts, log in, and manage profile information, laying the groundwork for features such as saved carts, wishlists, and order history to enhance overall engagement.

USE CASE DIAGRAM

USER OR CUSTOMER :



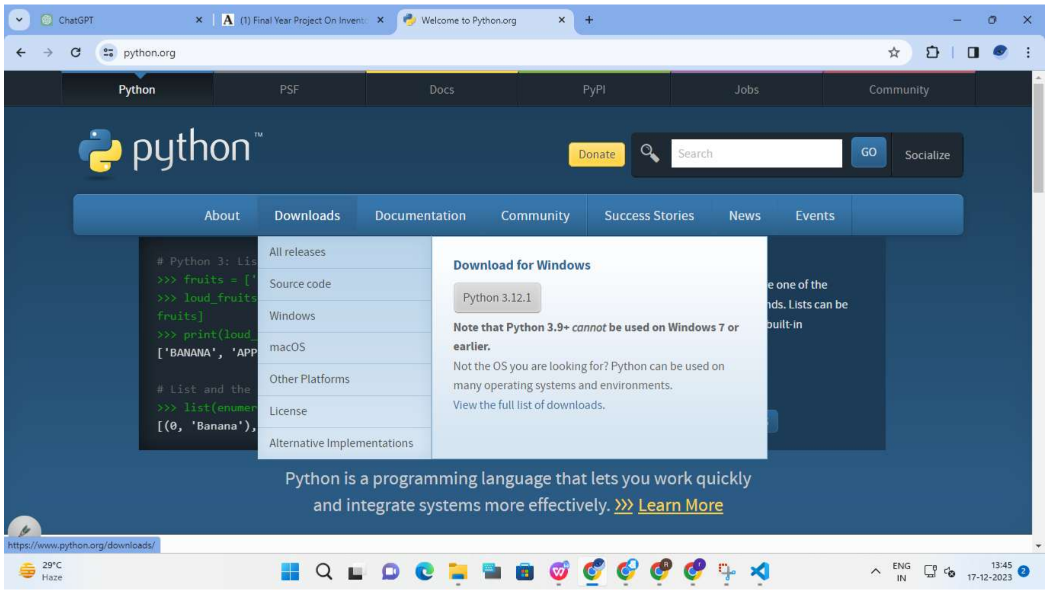
ADMIN:



Tools Used :

Python: A Dynamic Environment:

Python, known for its readability and versatility, played a pivotal role in crafting a dynamic and efficient development environment. Leveraging the power of Python, the backend development in Django was seamless and agile. The rich ecosystem of Python libraries and frameworks complemented Django's functionality, allowing for rapid development without compromising on code quality.



Download for Windows

Note that Python 3.9+ *cannot* be used on Windows 7 or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

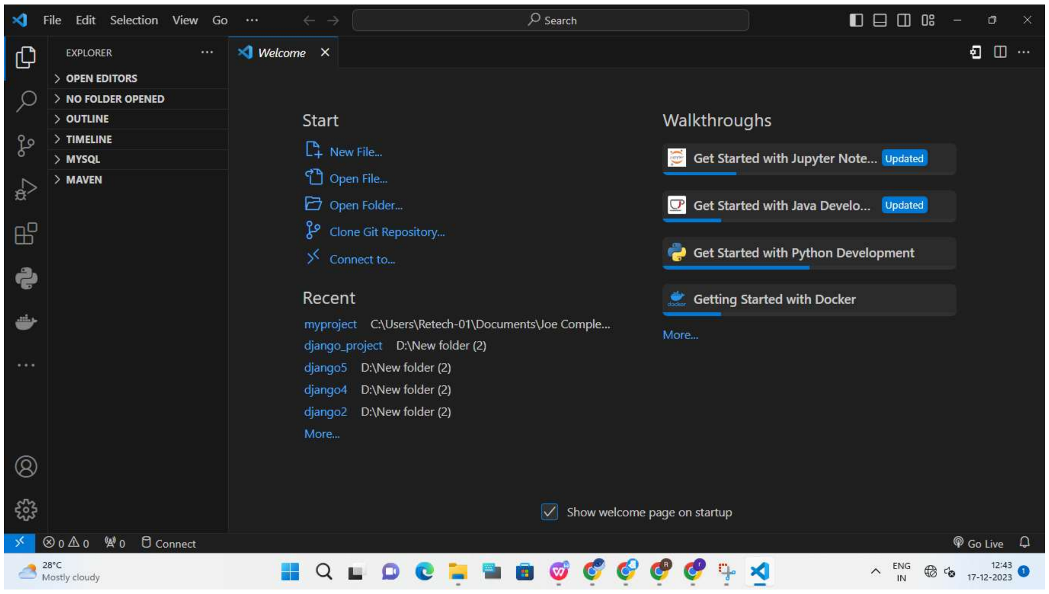
Key Python Features:

Readability: Python's clean and concise syntax enhanced code readability, fostering collaboration among team members.

Versatility: The flexibility of Python enabled seamless integration with various libraries, frameworks, and tools, enhancing the development experience.

Visual Studio: The IDE of Choice:

Visual Studio emerged as the IDE of choice, offering a comprehensive suite of features tailored for fullstack development. The integration of Django within Visual Studio streamlined the development workflow, providing a unified environment for both frontend and backend components.


File Edit Selection View Go
Search
EXPLORER
OPEN EDITORS
NO FOLDER OPENED
OUTLINE
TIMELINE
MYSQL
MAVEN
Welcome
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Connect to...
Recent
myproject C:\Users\Retech-01\Documents\Joe Comple...
django_project D:\New folder (2)
django5 D:\New folder (2)
django4 D:\New folder (2)
django2 D:\New folder (2)
More...
Walkthroughs
Get Started with Jupyter Note... Updated
Get Started with Java Develo... Updated
Get Started with Python Development
Getting Started with Docker
More...
Show welcome page on startup
Connect
Go Live
28°C Mostly cloudy
ENG IN
12:43 17-12-2023

Key Visual Studio Features:

Intelligent Code Assistance: Visual Studio's intelligent code assistance features, such as code completion and suggestions, expedited the coding process, reducing errors and boosting productivity.

Integrated Debugging: The builtin debugger facilitated the identification and resolution of issues, ensuring a robust and stable application.

Version Control Integration: Seamless integration with version control systems like Git empowered efficient collaboration and code management within a team.

<u>Django</u>

Django is a high-level web framework for building web applications using the Python programming language. It follows the Model-View-Controller (MVC) architectural pattern, emphasizing the "Don't Repeat

Yourself" (DRY) and "Convention over Configuration" (CoC) principles. Created by Adrian Holovaty and Simon Willison, Django was released in 2005 and has since become one of the most popular web frameworks for developers.

Key Features of Django:

1. Model-View-Controller (MVC) Architecture:

Django follows the MVC architectural pattern, though it refers to it as the Model-View-Template (MVT) pattern. The model defines the data structure, the view handles the presentation logic, and the template manages the user interface.

2. Object-Relational Mapping (ORM):

Django includes a powerful ORM system that enables developers to interact with databases using Python code instead of SQL queries. This abstraction simplifies database interactions and supports multiple database backends.

3. Admin Interface:

Django comes with an automatic admin interface that allows developers to manage application data with minimal effort. This built-in feature is customizable, making it easy to create a user-friendly admin panel for managing models and data.

4. URL Routing and Views:

Django's URL routing system allows developers to map URLs to views, which are Python functions or classes responsible for handling requests and returning appropriate responses. This separation of concerns makes code modular and maintainable.

5. Form Handling:

Django simplifies form creation and processing. It provides an intuitive form handling system, including built-in validation and security features, making it easy for developers to create and process HTML forms.

6. Template System:

Django's template system allows developers to separate HTML code from Python logic, promoting clean and maintainable code. Templates support template inheritance and include features for dynamic content rendering.

7. Middleware Support:

Django includes middleware components that process requests and responses globally. This allows developers to add functionalities such as authentication, security, and caching to their applications easily.

8. Security Features:

Django is designed with security in mind. It provides protection against common web vulnerabilities, including cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL injection. The framework encourages best practices for secure coding.

9. Authentication and Authorization:

Django includes built-in authentication and authorization systems, making it straightforward for developers to implement user authentication, manage user sessions, and control access to different parts of their applications.

10. REST Framework:

While Django itself is not strictly a REST framework, the Django REST framework is a powerful and flexible toolkit for building web APIs. It extends Django to handle serialization, authentication, and other features commonly needed in RESTful applications.

11. Community and Ecosystem:

Django has a large and active community that contributes to its growth. The Django Package Index (PyPI) hosts numerous third-party packages and extensions, providing additional functionalities and options for developers.

IMPLEMENTATION:

User Interface Design:

User Interface Design is concerned with the dialogue between a user and the computer. It is concerned with everything from starting the system of logging into the system to the eventually presentation of desired inputs and outputs. The overall flow of screens and messages is called a dialogue.

The following s are various guidelines for User Interface Design:

• The system user should always be aware of what to do next.

• The screen should be formatted so that various types of information, instructions and messages always appear in the same general display area.

• Message, instructions or information should be displayed long enough to allow the system user to read them.

• Use display attributes sparingly.

• Default values for fields and answered to be entered by the user should be specified.

• A use should not be allowed to proceed without correcting an error. • The system user should never get an operating system message or fatal error.

Setup of virtual environment:

A virtual environment is a self-contained directory tree that contains dependencies required by different projects isolated to existing packages. By using virtual Python environments, applications can run in their own 'sandbox' in isolation of other Python applications

1: Open your terminal and create a directory to store all your virtual environments, using the command "mkdir Environments" which is an acronym of "make directory". Now go inside the directory using the command CD which stands for call Directory, "CD Environments"

2: Now we will use a module named virtualenv to create isolated virtual environments. But before it first, install this module "pip install virtualenv" If error occurs then install pip package manager first. To verify a successful installation run this "virtualenv –version"

• To create a Virtual Environment for Python 3 python3 -m venv myenv Or virtualenv myenv Or "python -m venv myenv"

This will create a directory myenv along with directories inside it containing a copy of the Python interpreter, the standard library, and various supporting files. A virtual Python environment has a similar directory structure to a global Python installation.

The bin directory contains executables for the virtual environment, the include directory is linked to the global Python installation header files, the lib directory is a copy of the global Python installation libraries and where packages for the virtual environment are installed, and the shared directory is used to place shared Python packages.

• To add modules and packages in our Environment, we need to activate it first. On Windows, run: "myenv\Script\activate.bat" On Unix or MacOs, run: "source myenv/bin/activate"

• To deactivate the current Environment, run: "deactivate" Code snippet:

Now run the following command in your shell to create a Django project.

"django-admin startproject mysite"

This will generate a project structure with several directories and python scripts.

```
├── mysite
│ ├── __init__.py
│ ├── settings.py
│ ├── urls.py
│ ├── wsgi.py
├── manage.py
```

To navigate the python server manager,

run: "cd mysite"

"python manage.py startapp blog"

These will create an app named blog in our project.

```
├── db.sqlite3
├── mysite
│ ├── __init__.py
│ ├── settings.py
│ ├── urls.py
│  ├── wsgi.py
├── manage.py
└── blog
  ├── __init__.py
  ├── admin.py
  ├── apps.py
  ├── migrations
  │  └── __init__.py
  ├── models.py
  ├── tests.py
  └── views.py
```

Now we need to inform Django that a new application has been created, open your settings.py file and scroll to the installed apps section, which should have some already installed apps.

```
INSTALLED_APPS = [
        django.contrib.admin',
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.messages',
        'django.contrib.staticfiles',
]
```

Now add the newly created app blog at the bottom and save it.

```
INSTALLED_APPS = [
        'django.contrib.admin',
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.messages',
        'django.contrib.staticfiles',
        'blog'
]
```

Next, make migrations.

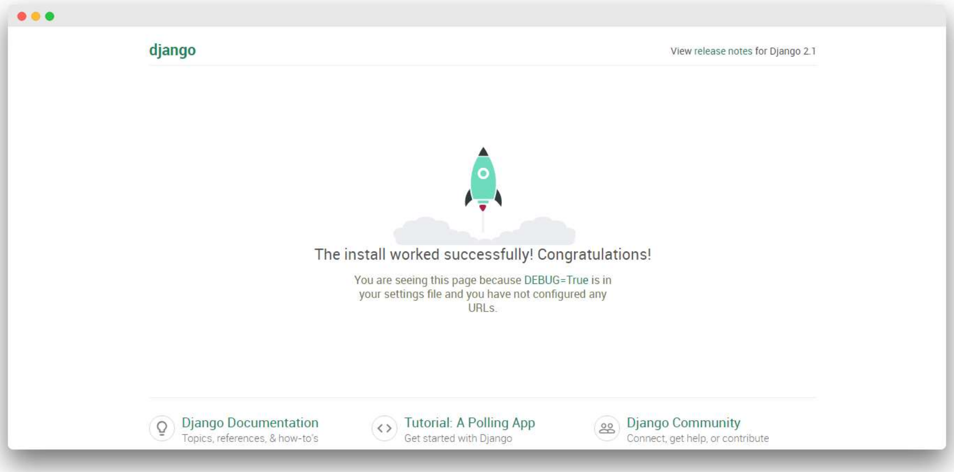
“*python manage.py migrate*”

This will apply all the unapplied migrations on the SQLite database which comes along with the Django installation.

Let’s test our configurations by running the Django’s built-in development server.

“*python manage.py runserver*”

Open your browser and go to this address http://127.0.0.1:8000/ if everything went well



django

View release notes for Django 2.1

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.

Define Models

Edit `petshop_app/models.py` to define your models:

```
python
from django.db import models

class Pet(models.Model):
    name = models.CharField(max_length=100)
    species = models.CharField(max_length=50)
    breed = models.CharField(max_length=50)
    age = models.IntegerField()
    price = models.DecimalField(max_digits=8, decimal_places=2)
    # Add more fields as needed

class AnimalData(models.Model):
    name = models.CharField(max_length=100)
    breed = models.CharField(max_length=50)
    category = models.CharField(max_length=50)
    weight = models.FloatField()
    cost = models.DecimalField(max_digits=8, decimal_places=2)
    height = models.FloatField()
    fur = models.CharField(max_length=50)
    # Add more fields as needed
```

Run Migrations

```
python manage.py makemigrations
python manage.py migrate
```

Create Views

Edit `petshop_app/views.py` to define your views:

```
from django.shortcuts import render
from .models import Pet, AnimalData

def pet_list(request):
    pets = Pet.objects.all()
    return render(request, 'petshop_app/pet_list.html', {'pets': pets})

def animal_data_list(request):
    animal_data = AnimalData.objects.all()
    return render(request, 'petshop_app/animal_data_list.html',
{'animal_data': animal_data})
```

Create Templates

Create the templates in the `petshop_app/templates/petshop_app` directory. For example, `pet_list.html`:

```
html
<!DOCTYPE html>
<html>
<head>
    <title>Pet Shop - Pet List</title>
</head>
<body>
    <h1>Available Pets</h1>
    <ul>
        {% for pet in pets %}
            <li>{{ pet.name }} - {{ pet.species }} - {{ pet.breed }} -
{{ pet.age }} years old - ${{ pet.price }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

And `animal_data_list.html`:

```html
html
<!DOCTYPE html>
<html>
<head>
    <title>Pet Shop - Animal Data List</title>
</head>
<body>
    <h1>Animal Data List</h1>
    <ul>
        {% for data in animal_data %}
            <li>{{ data.name }} - {{ data.breed }} - {{ data.category }} -
{{ data.weight }} kg - ${{ data.cost }} - {{ data.height }} cm -
{{ data.fur }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Configure URLs

Edit `petshop_app/urls.py`:

```python
python
from django.urls import path
from .views import pet_list, animal_data_list

urlpatterns = [
    path('pets/', pet_list, name='pet_list'),
    path('animal_data/', animal_data_list, name='animal_data_list'),
]
```

Include these URLs in the main `urls.py`:

```python
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('petshop_app.urls')),
]
```

Run the Development Server

```bash
python manage.py runserver
```

Visit `http://127.0.0.1:8000/pets/` and `http://127.0.0.1:8000/animal_data/` in your web browser to see the list of available pets and animal data.