PROJECT NAME

WanderLust Travel Blog

TECHNISCHE HOCHSCHULE DEGGENDORF

Submitted By

1.	Gayathri Sathyanand	95971
2.	Maria Veronica Garcia Pinilla.	96933
3.	Mohammed Aly	96079
4.	Asif Sultan	96632

Submitted to

Prof.Andreas Fischer

INDEX

ACKNOWLEGEMENT

We express our sincere gratitude to our Prof. Andreas Fischer for guiding guiding us right from the inception till the successful completion. We sincerely acknowledge him for extending his valuable guidance, support for literature, critical reviews of the project and above all the moral support he had provided us with all the stages of the project.

Finally, We would like to add few heartfelt words for the members of our group who were the part of this project, this project cannot be completed without the effort and co-operation from our group members and also our friends and classmates who gave us unending support right from the beginning.

Thanking You,

Asif Sultan

Gayathri Sathyanand

Maria Veronica Pinilla

Mohammed Aly

WanderLust

AIM:

To create a Travel Blog to share the experiences with the world regarding different cultures, food, traditions and more practical information to the travellers. This project aims to develop a software package that can be used to host a travel blogging by using Django, Html, Css and Bootstrap.

OBJECTIVES:

The main objective of the project is to have a in-depth information in

- To develop a software package that can be used to host a Travel Blog.
- Converting the travel experiences into a story and share it with the digital world to boost the reader base with the help of various digital tools.
- Creating an interface between Users and Authors.
- Giving information to the travellers regarding the experiences of author and travel tips that is required for a traveller.

INTRODUCTION:

Blogs are strange imps of the online world. The term was coined in the late 1990s, but several individuals started websites with daily, personal updates a few years prior. 1999 was a banner year for the blogger, the term was coined and people around the world started experimenting with this new way of sharing their thoughts with the world, all in reverse-chronological order of course.

As technology made it easier for bloggers to emerge, a cascade of writing began to fill the inter-webs until we had the large, diverse population of bloggers we see today. Many blogs are only marginally considered blogs. Either way though, blogs are a form of social media as they encourage and depend on readers to interact in an online community of enthusiasts.

A travel blogger is someone who travels around the world collecting information for writing about their travel experiences. A travel blogger is a freelance writer who maintains their own blog site. The travel blogger must travel to a destination country (or more typically, a series of countries) to obtain exciting, informative experiences which he or she will then host on their blog site. They will also usually be required to upload good quality photography to illustrate the pieces.

From the first trip, the travel blogger will be collecting experiences on several levels, the need to create professional written content, upload it to a blog, manage the pages and promote the travel blog are specific skills which are usually self-taught through experience.

Although not relevant per se, experienced travel writers and bloggers will develop a burgeoning portfolio of contacts, making it more likely that future travel features they complete will be published in print. Well capitalized bloggers will be able to explore exotic and remote locations beyond the reach of inexperienced bloggers with limited resources.

Converting the travel experiences into a story and sharing it with the digital world to boost reader base with the help of various digital tools is the objective of WanderLust. In other words, "WanderLust is all about to share the numerous of information in public domain related to travel and tourism world through a digital platform".

DESIGN OF PROJECT:

The structure of WanderLust consists of:

Main Page:

The Main Page will consist of an interface that will show the latest blog posts with hyperlinks which will direct to the new page with the desired contents. The header will consists of About Us which gives information about the Administration and the Authors who are responsible for WanderLust, The Famous Destinations which the travellers usually look for travelling, Travel Tips so the travellers are aware of the Do's and Don'ts's and are prepared well before reaching the desired location, Register so that the users can have access to comment as well as post their views about the blog and also share their experiences, once registered the users will also be able to get regular updates, register is used to by Anonymous User so they have more benefits of information and the access to commenting section and posting their views. Login Page is for the Registered Users who have already registered with their respective email and other details like their Full name, they can also Login via their social media platform like Facebook, Blog where the Registered User has the authority to make a new post. The header also consists of a search engine where the user can search what they are looking for in specific. The main page also has Overview links which when clicked will direct to a new page which shows that posts. These Overview links can be used by the Registered Users and Anonymous Users to read the blog posts of their interests.

ADMINISTRATOR:

The Administrator plays an important role in creation of a blog. The administrator is at the very top of the hierarchy. An Administrator has full power over the site and can do everything related to site administration. They have complete control over posts, pages, uploaded files, comments, settings, themes, imports, exports, other users – the whole shebang.

Administrators can do **everything.** This user role can, in part:

- Create more Administrators
- Invite New Users
- Remove Users
- Change User Roles
- Create, edit, and delete any content
- Manage plugins and themes
- Edit code
- Delete other user accounts

In Summary, they have complete control over posts, pages, uploaded files, comments, settings, themes, imports, exports, other users – the whole shebang.

AUTHORS:

An author has far fewer permissions than editors. They cannot edit pages and are unable to alter another users' content. In addition, they lack any sort of administrative capabilities.

What they can do is create, edit, delete, and publish their own posts (and upload media files). An author can add an arbitrary number of keywords to the article.

This makes their role clear – authors are responsible for creating content, and nothing more.

USERS:

There are two types of users, Registered Users and Anonymous Users.

1. Registered Users:

Registered Users are the users that have already registered themselves by using a special web form option which is filling up details like their Full name, their email address or by using their social media to register themselves. The Registered User will have access post their views and also to comment under the commenting section of the blog posts, they also get facility of getting latest updates of the blogs via emails.

2. Anonymous Users:

Anonymous Users have a facility to become a registered user, an administrator, or an author by signing in. They need to sign in to have the access to create new posts and commenting and getting regular updates about the blog. Anonymous Users without signing up can also comment but on a condition that the comments can only be shown in the blog after having the permission of administrator. Anonymous users cannot access web pages dedicated to blog management. They cannot see unpublished articles, comments waiting for acceptance, open registration requests, or the list of registered users.

3. Social Media Integration:

The Registered Users and Anonymous Users when clicked on the overview, they will navigate to a new web page and this page will have detailed information about the selected overview link with pictures and space for comments for the registered and anonymous users and also the option of sharing the blog in social media such as Instagram, Facebook, Twitter and emails.

4.Overview Links:

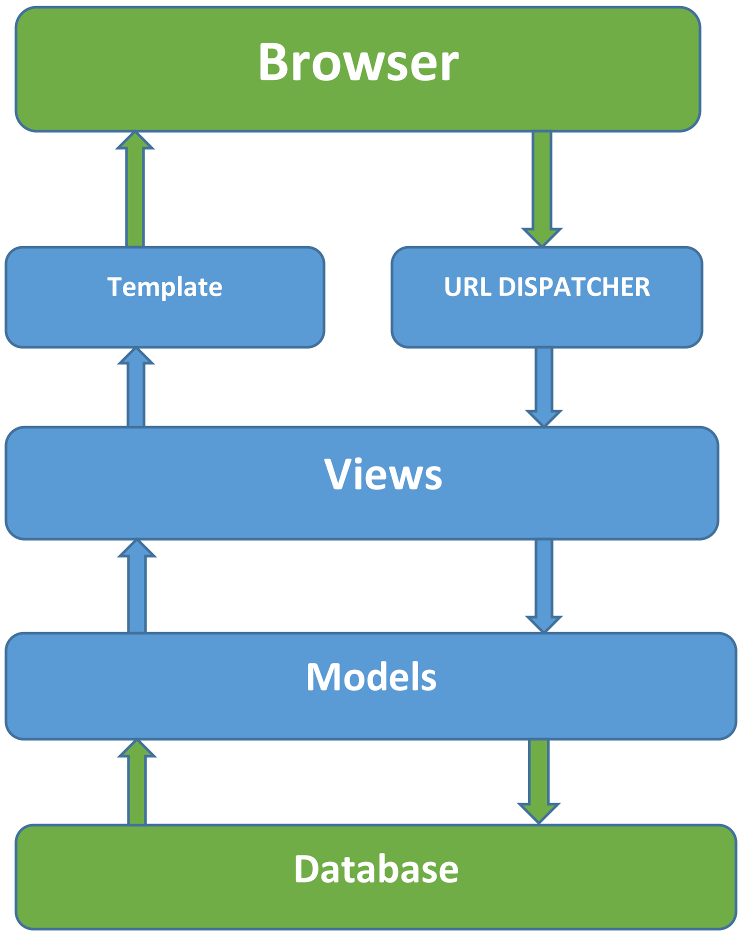
This overlink is present in the main page where it consists of related title along with a picture, which when clicked by the Registered User or Anonymous User, takes them to the desired page of contents where they can access the detailed information such as Travel Experiences, Travel Tips such as Do´s and Dont´s, Cultures, Traditions, Food and Shopping. This page will also show the information of the author like his name, picture, date and time when the blog was posted. It also contains recent posts as well as Categories at the right-hand side of the page where the users can be navigated according to their interests.

5.The contents of the blog:

The Registered User or Anonymous User on selecting a particular overview link, will be directed to the detailed contents which contains images to represent that particular travel place, the experiences, the travel tips, various cultures and traditions, and their food and lifestyle, and various shopping places and famous destinations. At the right-hand side, it contains various categories as well as recent posts. Below the contents there is comment section where Registered User can comment their views and anonymous users can also comment but can only be visible upon the permission of an administrator.

Architecture of Blog in the Backend

Architecture of Django Framework



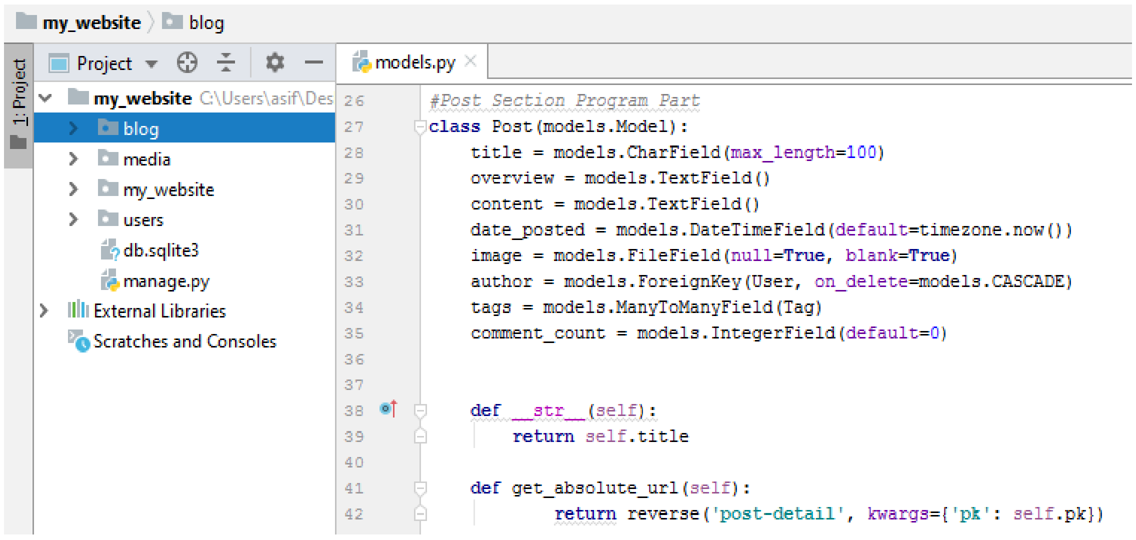
Because Django was developed in a fast-paced newsroom environment, it was designed to make common Web-development tasks fast and easy. In the coming section of the report we are going to explain how the website is being designed for making our blog website. We will explain the above architecture diagram how an interface is being created and how they are interconnected to one another. This will help the reader to have a clear-cut idea of how the programs are written in Django for the designing a website. The above diagram in a nutshell explains how architecture of the Django framework works, it explains how they are fetched when they user press certain link.

Designing of Model

Although one can use Django without a database, it comes with an object-relational mapper in which one describe his database layout in Python code. The data-model syntax offers many rich ways of representing your models – so far, it's been solving many years' worth of database-schema problems. Here's a quick example:

The program starts in the models.py by importing the property of Django: -

my_website/blog/models.py
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User



```
#Post Section Program Part
class Post(models.Model):
    title = models.CharField(max_length=100)
    overview = models.TextField()
    content = models.TextField()
    date_posted = models.DateTimeField(default=timezone.now())
    image = models.FileField(null=True, blank=True)
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    tags = models.ManyToManyField(Tag)
    comment_count = models.IntegerField(default=0)


    def __str__(self):
        return self.title

    def get_absolute_url(self):
            return reverse('post-detail', kwargs={'pk': self.pk})
```

Now we must install it

So, we will run the Django command-line utility to create the database tables automatically:

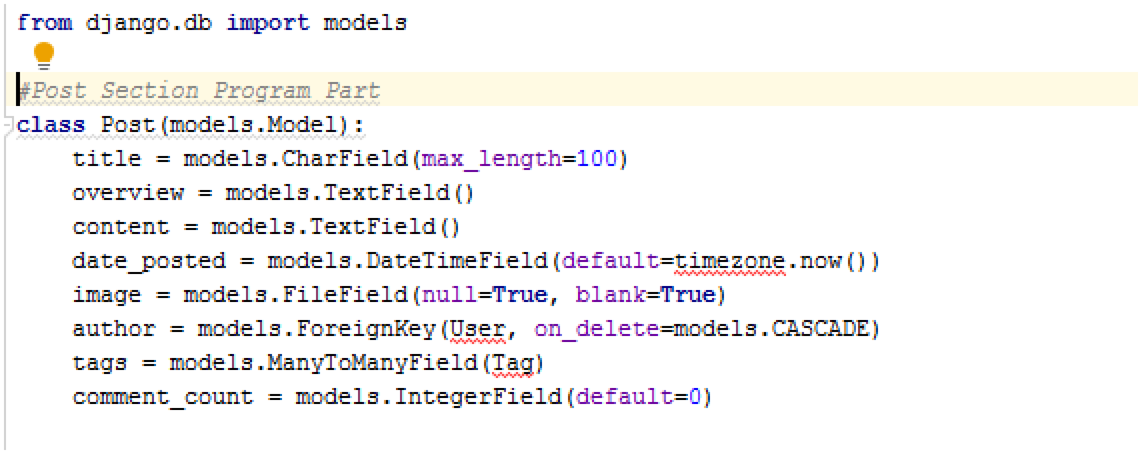
>> python manage.py migrate

The migrate command looks at all our available models and creates tables in our database for whichever tables don't already exist, as well as optionally providing much richer schema control.

A dynamic admin interface: it's not just scaffolding – it's the whole house

Once our models are defined, Django can automatically create a professional, production ready administrative interface – a website that lets authenticated users add, change and delete objects. It's as easy as registering your model in the admin site:

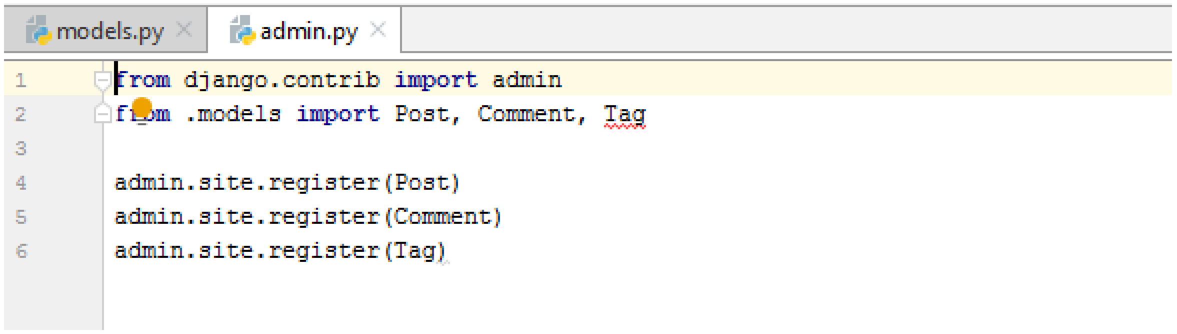
my_website/blog/models.py



```
from django.db import models

#Post Section Program Part
class Post(models.Model):
    title = models.CharField(max_length=100)
    overview = models.TextField()
    content = models.TextField()
    date_posted = models.DateTimeField(default=timezone.now())
    image = models.FileField(null=True, blank=True)
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    tags = models.ManyToManyField(Tag)
    comment_count = models.IntegerField(default=0)
```

my_website/blog/admin.py



models.py admin.py

```
from django.contrib import admin
from .models import Post, Comment, Tag

admin.site.register(Post)
admin.site.register(Comment)
admin.site.register(Tag)
```

The philosophy here is that our site is edited by a staff, or a client, or maybe just the admin – and you don't want to have to deal with creating backend interfaces just to manage content.

One typical workflow in creating Django apps is to create models and get the admin sites up and running as fast as possible, so your staff (or clients) can start populating data. Then, develop the way data is presented to the public.

Designing of URLs

A clean, elegant URL scheme is an important detail in a high-quality Web application. Django encourages beautiful URL design and doesn't put any cruft in URLs, like .php or .asp.

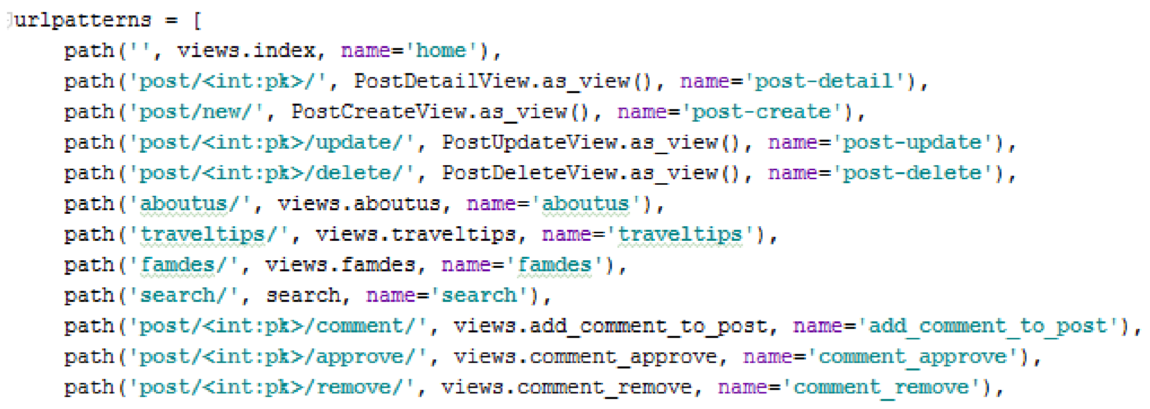
To design URLs for an app, you create a Python module called a URLconf. A table of contents for your app, it contains a simple mapping between URL patterns and Python callback functions. URLconfs also serve to decouple URLs from Python code.

Here's what a URLconf might look like for the Reporter/Article example above:

my_website/blog/urls.py

from django.urls import path

from . import views



```
urlpatterns = [
    path('', views.index, name='home'),
    path('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
    path('post/new/', PostCreateView.as_view(), name='post-create'),
    path('post/<int:pk>/update/', PostUpdateView.as_view(), name='post-update'),
    path('post/<int:pk>/delete/', PostDeleteView.as_view(), name='post-delete'),
    path('aboutus/', views.aboutus, name='aboutus'),
    path('traveltips/', views.traveltips, name='traveltips'),
    path('famdes/', views.famdes, name='famdes'),
    path('search/', search, name='search'),
    path('post/<int:pk>/comment/', views.add_comment_to_post, name='add_comment_to_post'),
    path('post/<int:pk>/approve/', views.comment_approve, name='comment_approve'),
    path('post/<int:pk>/remove/', views.comment_remove, name='comment_remove'),
```

The code above maps URL paths to Python callback functions ("views"). The path strings use parameter tags to "capture" values from the URLs. When a user requests a page, Django runs through each path, in order, and stops at the first one that matches the requested URL. (If none of them matches, Django calls a special-case 404 view.) This is blazingly fast, because the paths are compiled into regular expressions at load time.

Once one of the URL patterns matches, Django calls the given view, which is a Python function. Each view gets passed a request object – which contains request metadata – and the values captured in the pattern.

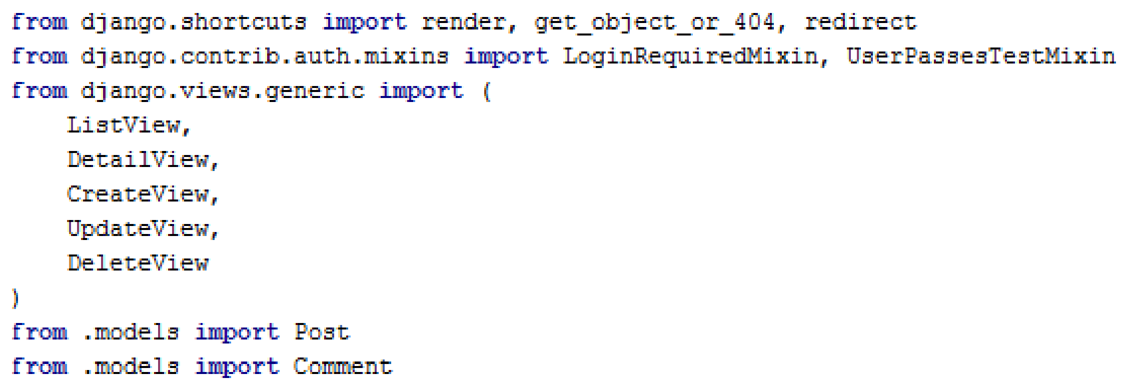
For example, if a user requested the URL "/post/2005/05/39323/", Django would call the function blog.views.post_detail(request, year=2019, month=5, pk=39323).

Designing of Views

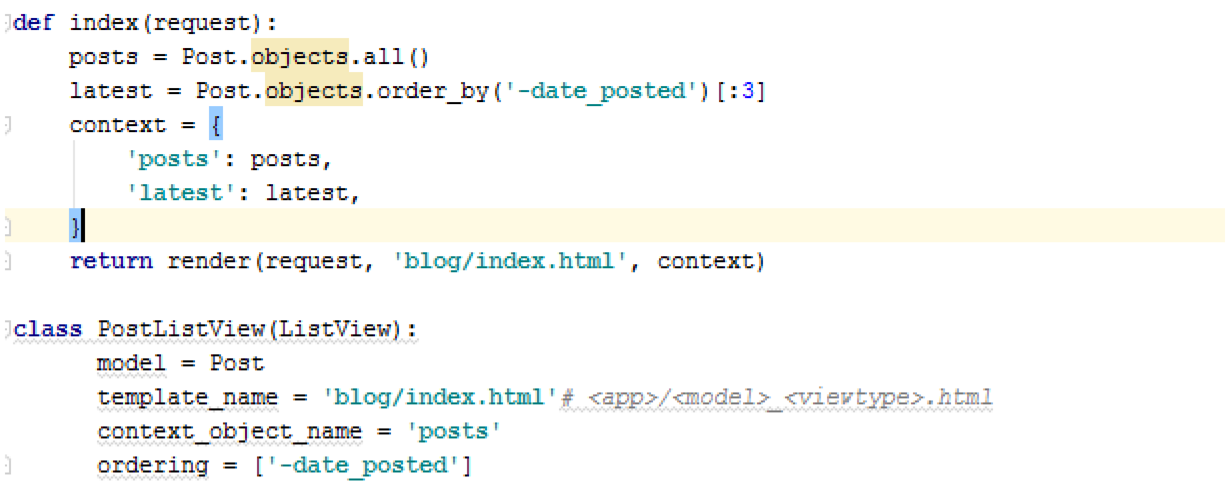
Each view is responsible for doing one of two things: Returning an HttpResponse object containing the content for the requested page, or raising an exception such as Http404. The rest is up to you.

Generally, a view retrieves data according to the parameters, loads a template and renders the template with the retrieved data. Here's an example view for year_archive from above:

mysite/news/views.py



```
from django.shortcuts import render, get_object_or_404, redirect
from django.contrib.auth.mixins import LoginRequiredMixin, UserPassesTestMixin
from django.views.generic import (
    ListView,
    DetailView,
    CreateView,
    UpdateView,
    DeleteView
)
from .models import Post
from .models import Comment
```



```python
def index(request):
    posts = Post.objects.all()
    latest = Post.objects.order_by('-date_posted')[:3]
    context = {
        'posts': posts,
        'latest': latest,
    }
    return render(request, 'blog/index.html', context)

class PostListView(ListView):
      model = Post
      template_name = 'blog/index.html'# <app>/<model>_<viewtype>.html
      context_object_name = 'posts'
      ordering = ['-date_posted']
```

This example uses Django's template system, which has several powerful features but strives to stay simple enough for non-programmers to use.

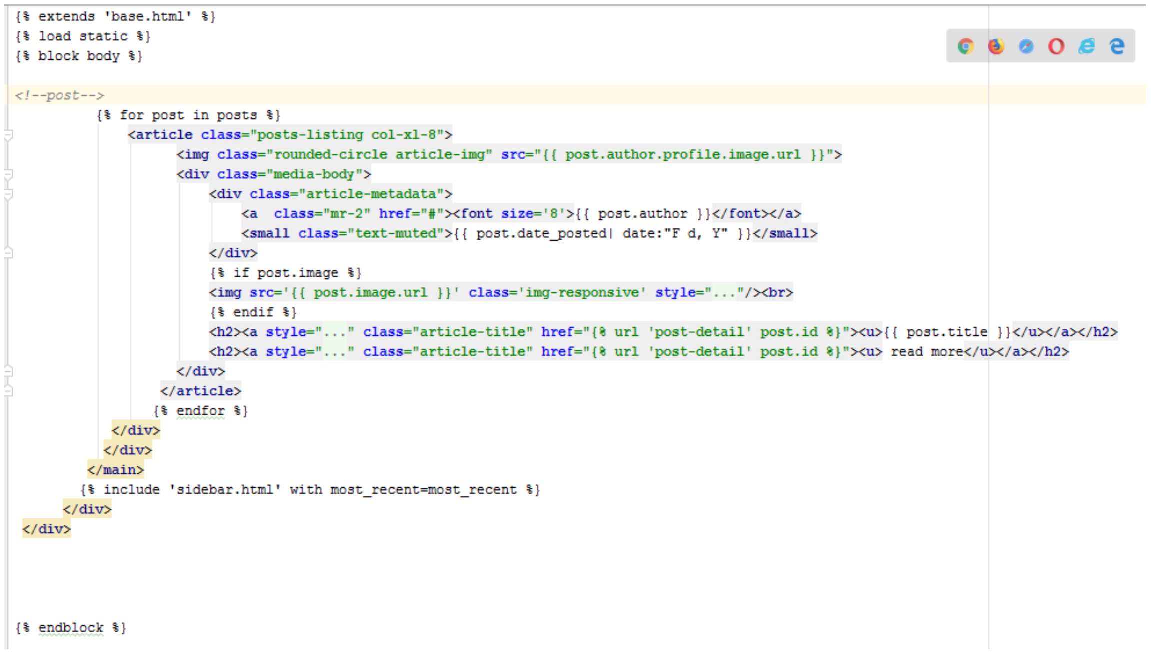
Designing of Templates

The code above loads the index.html template.

Django has a template search path, which allows you to minimize redundancy among templates. In your Django settings, you specify a list of directories to check for templates with DIRS. If a template doesn't exist in the first directory, it checks the second, and so on.

Let's say the index.html template was found. Here's what that might look like:

my_website/blog/templates/blog/index.html



```html
{% extends 'base.html' %}
{% load static %}
{% block body %}

<!--post-->
          {% for post in posts %}
              <article class="posts-listing col-xl-8">
                    <img class="rounded-circle article-img" src="{{ post.author.profile.image.url }}">
                    <div class="media-body">
                        <div class="article-metadata">
                            <a  class="mr-2" href="#"><font size='8'>{{ post.author }}</font></a>
                            <small class="text-muted">{{ post.date_posted| date:"F d, Y" }}</small>
                        </div>
                        {% if post.image %}
                        <img src='{{ post.image.url }}' class='img-responsive' style="..."/><br>
                        {% endif %}
                        <h2><a style="..." class="article-title" href="{% url 'post-detail' post.id %}"><u>{{ post.title }}</u></a></h2>
                        <h2><a style="..." class="article-title" href="{% url 'post-detail' post.id %}"><u> read more</u></a></h2>
                    </div>
                   </article>
                  {% endfor %}
            </div>
           </div>
         </main>
        {% include 'sidebar.html' with most_recent=most_recent %}
      </div>
  </div>



{% endblock %}
```

Variables are surrounded by double-curly braces. {{ post.author }} means "Output the value of the Post's author attribute." But dots aren't used only for attribute lookup. They also can do dictionary-key lookup, index lookup and function calls.

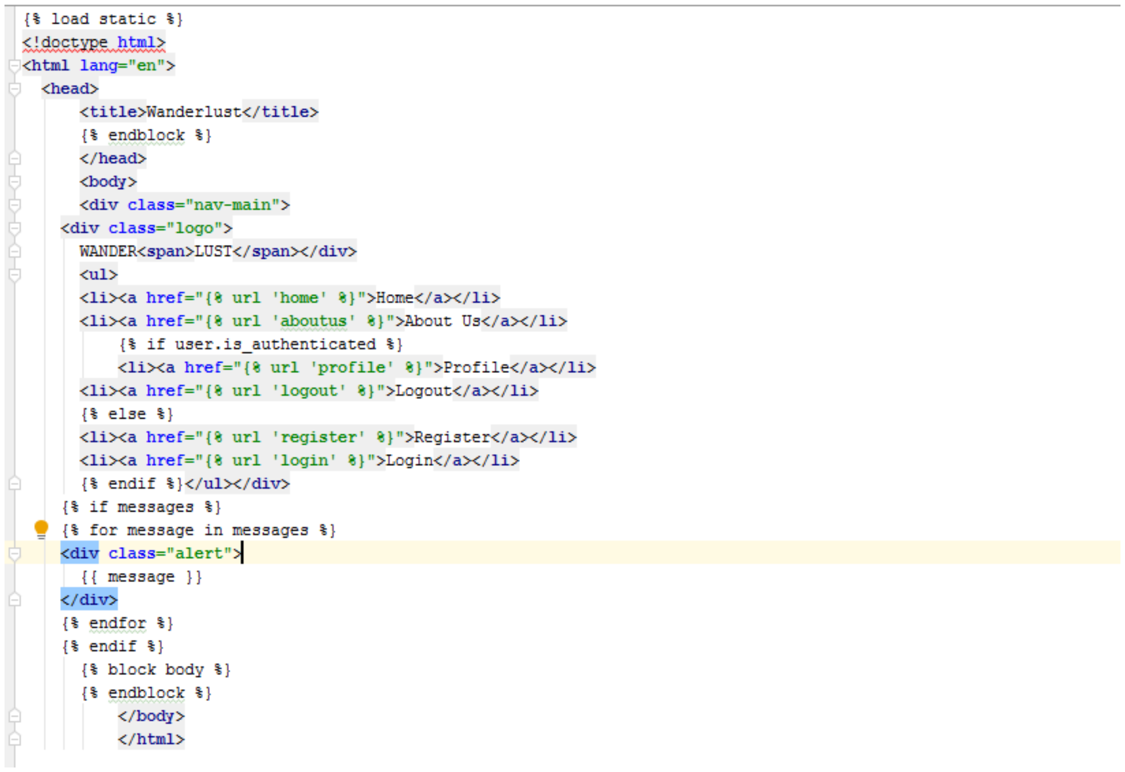
Note {{ post.date_posted| date:"F d, Y" }} uses a Unix-style "pipe" (the "|" character). This is called a template filter, and it's a way to filter the value of a variable. In this case, the date filter formats a Python datetime object in the given format (as found in PHP's date function).

One can chain together as many filters as one would like. You can write custom template filters. You can write custom template tags, which run custom Python code behind the scenes.

Finally, Django uses the concept of "template inheritance". That's what the {% extends "base.html" %} does. It means "First load the template called 'base', which has defined a bunch of blocks, and fill the blocks with the following blocks." In short, that lets you dramatically cut down on redundancy in templates: each template has to define only what's unique to that template.

Here's what the "base.html" template, including the use of static files, might look like:

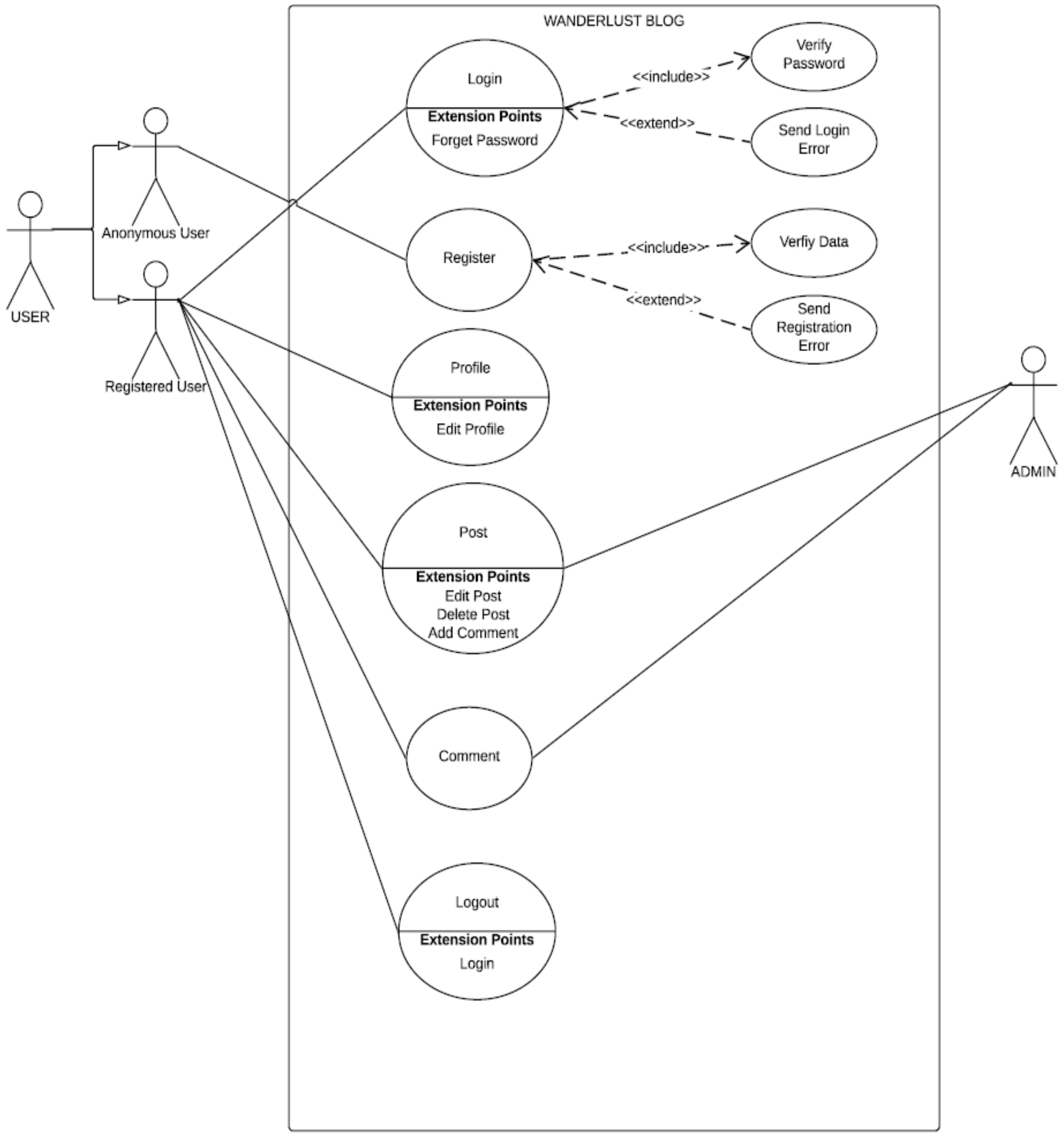
my_website/blog/templates/base.html



```
{% load static %}
<!doctype html>
<html lang="en">
  <head>
      <title>Wanderlust</title>
      {% endblock %}
      </head>
      <body>
      <div class="nav-main">
    <div class="logo">
      WANDER<span>LUST</span></div>
      <ul>
      <li><a href="{% url 'home' %}">Home</a></li>
      <li><a href="{% url 'aboutus' %}">About Us</a></li>
          {% if user.is_authenticated %}
          <li><a href="{% url 'profile' %}">Profile</a></li>
      <li><a href="{% url 'logout' %}">Logout</a></li>
      {% else %}
      <li><a href="{% url 'register' %}">Register</a></li>
      <li><a href="{% url 'login' %}">Login</a></li>
      {% endif %}</ul></div>
    {% if messages %}
    {% for message in messages %}
    <div class="alert">
      {{ message }}
    </div>
    {% endfor %}
    {% endif %}
      {% block body %}
      {% endblock %}
          </body>
          </html>
```

Simplistically, it defines the look-and-feel of the site (with the site’s logo), and provides “holes” for child templates to fill. This makes a site redesign as easy as changing a single file – the base template.It also lets you create multiple versions of a site, with different base templates, while reusing child templates. Django’s creators have used this technique to create strikingly different mobile versions of sites – simply by creating a new base template.Note that you don’t have to use Django’s template system if you prefer another system. While Django’s template system is particularly well-integrated with Django’s model layer, nothing forces one to use it. For that matter, we don’t have to use Django’s database API, either. One can use another database abstraction layer, you can read XML files, you can read files off disk, or anything you want. Each piece of Django – models, views, templates – is decoupled from the next.

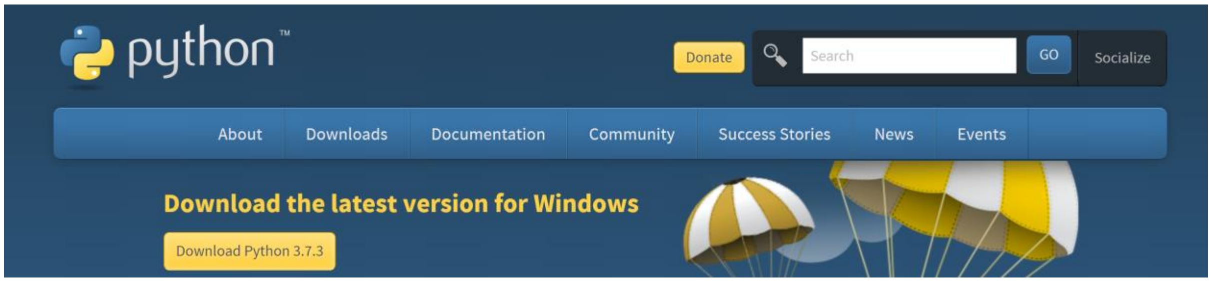
UML DIAGRAM

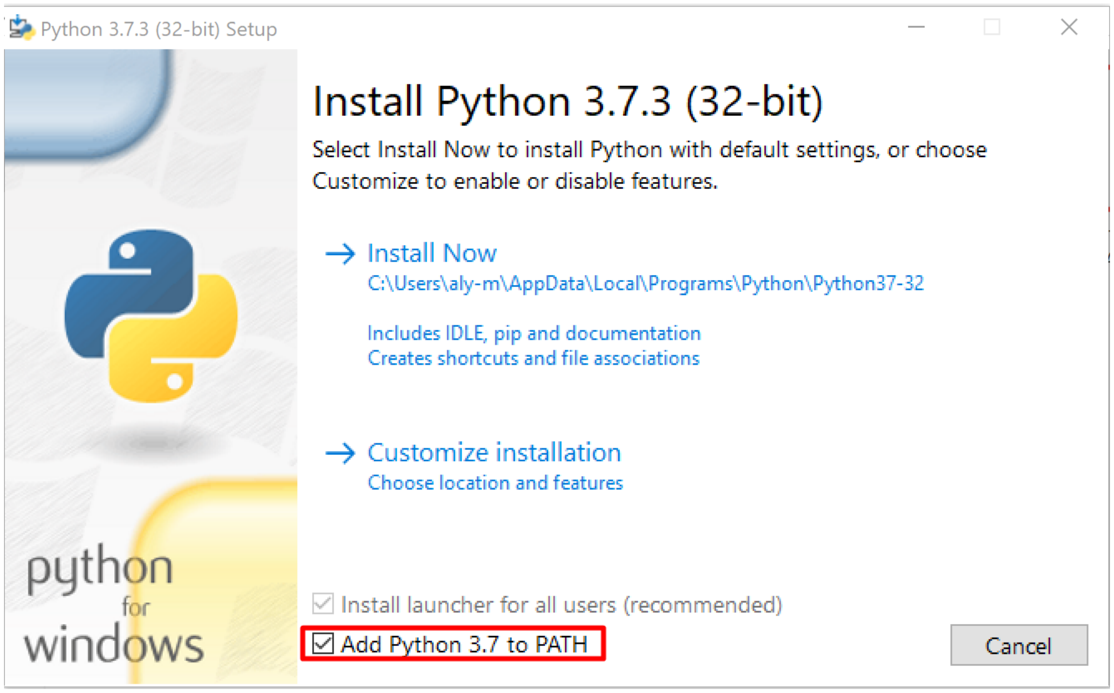

WANDERLUST BLOG
Login
Extension Points
Forget Password
<<include>>
Verify Password
<<extend>>
Send Login Error
Register
<<include>>
Verfiy Data
<<extend>>
Send Registration Error
Profile
Extension Points
Edit Profile
Post
Extension Points
Edit Post
Delete Post
Add Comment
Comment
Logout
Extension Points
Login
USER
Anonymous User
Registered User
ADMIN

Installing Python, Django, Packages, and Running server

To get our server and website working with no errors or warnings, there are some libraries that have to be installed, and that could be achieved by following the mentioned below instructions:

i) First of all Python has to be downloaded from the following website: www.python.org and pressing on Download Python 3.7.3




Python 3.7.3 (32-bit) Setup

Install Python 3.7.3 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\aly-m\AppData\Local\Programs\Python\Python37-32

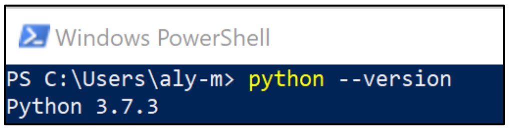
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.7 to PATH

ii) By checking the 'Add Python 3.7 to PATH' and getting done with the installation, now it is time to check that python is successfully installed on your PC. Go to Powershell or the normal Command Window (CMD) and then type the following command to check your Python version installed on your PC.

```
python --version
```



Windows PowerShell

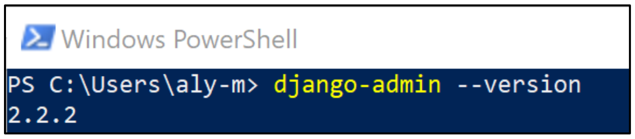
```
PS C:\Users\aly-m> python --version
Python 3.7.3
```

iii) By using the Pip -Okay you might wonder what pip is, Pip is actually a package manager for Python. That means it's a tool that allows you to install and manage additional libraries and dependencies that are not distributed as part of the standard library- Django could be installed and this is achieved by typing this command in the command window

```
pip install django
```

a. Once the installing is complete, you can also check the version of the Django installed by typing the following command

```
django-admin --version
```



Windows PowerShell

```
PS C:\Users\aly-m> django-admin --version
2.2.2
```

iv) Still we need to install some other required packages to get our server working and the packages that we need are as follows:

<u>PILLOW:</u>

- **Pillow** is the friendly PIL fork by Alex Clark and Contributors. PIL is the Python Imaging Library by Fredrik Lundh and Contributors. And this package could be installed by typing the following command.

```
pip install Pillow
```

<u>CRISPY FORMS:</u>

- It is a tool that will let you control the rendering behavior of your Django forms in a very elegant and DRY way. Have full control without writing custom form templates. And this package could be installed with this command.

```
pip install django-crispy-forms
```

v) Once the package needed are successfully installed, it is time to locate our project folder and this could be done by using the *cd* command and add the directory of the folder where our project is located.

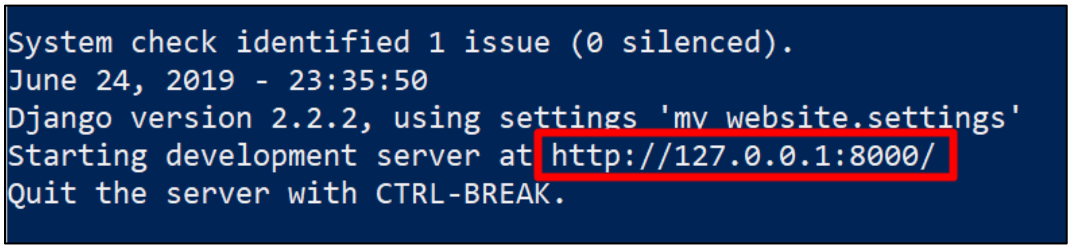
vi) Once the project is successfully located, type these two commands below before running the server.

```
python manage.py makemigrations
```

```
python manage.py migrate
```

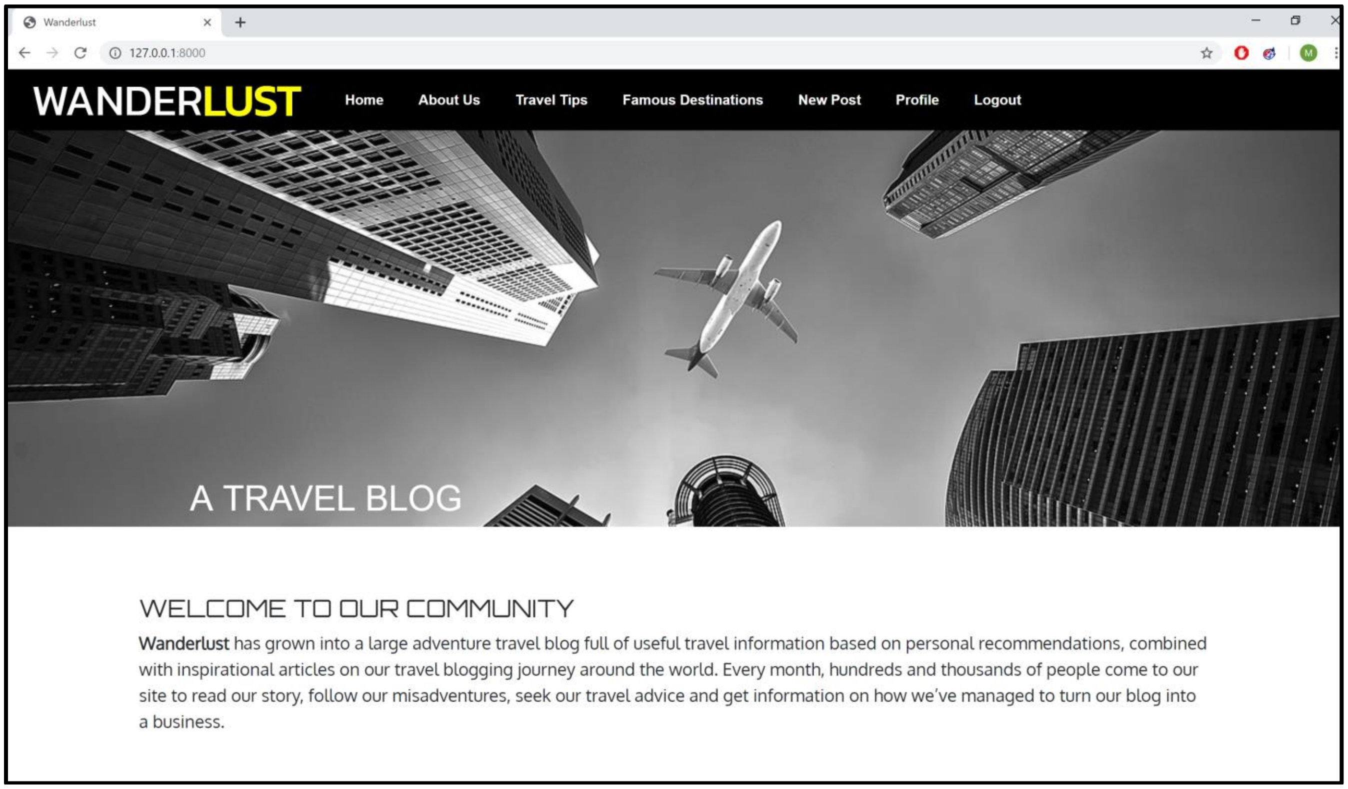
vii) Now the last step is to run the server, and this could be done by typing this simple command in the command window

```
python manage.py runserver
```



```
System check identified 1 issue (0 silenced).
June 24, 2019 - 23:35:50
Django version 2.2.2, using settings 'mv website.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

viii) By copying and pasting this address to a new tab in your browser you will be able to access Wanderlust blog.



WELCOME TO OUR COMMUNITY

Wanderlust has grown into a large adventure travel blog full of useful travel information based on personal recommendations, combined with inspirational articles on our travel blogging journey around the world. Every month, hundreds and thousands of people come to our site to read our story, follow our misadventures, seek our travel advice and get information on how we've managed to turn our blog into a business.

Testing in Django

“Automated testing is an extremely useful bug-killing tool for the modern Web developer. It is possible to implement a collection of tests like a test suite – to solve, or avoid, a number of problems”.

In order to ensure that the changes that we have done working as a group will not affect our application’s behavior, we got good code coverage using testing Django application,Test-execution framework and assorted utilities. We had simulated requests, insert test data, inspect the application’s output and verified codes for proper functionality check. Using the Formal test procedures we can provide a safety net for software developers.

Types of Testing:

The are numerous types, levels, and classifications of tests and testing approaches. The most important automated tests are:

Unit tests:

Verify functional behavior of individual components, often to class and function level.

Regression tests:

Tests that reproduce historic bugs. Each test is initially run to verify that the bug has been fixed, and then re-run to ensure that it has not been reintroduced following later changes to the code.

Integration tests:

Verify how groupings of components work when used together. Integration tests are aware of the required interactions between components, but not necessarily of the internal operations of each component. They may cover simple groupings of components through to the whole website.Django’s unit tests use a Python standard library module: unittest. This module defines tests using a class-based approach.

In our blog and users were used subclasses from django.test.Testcase in order to run each tests which is the individual unit of testing. It checks for a specific response to a particular set of inputs.unittest provides a base class, TestCase, which may be used to create new test cases..Each test contains call to assertEqual() to check for an expected result and assertTrue() and assertFalse() to verify a condition.The setUp() methods allow us to define instructions that will be executed before and after each test method.

In addition a test runner was implemented as a component to manage the execution of tests and provides the outcome to the user. The runner use a graphical interface and a special value to indicate the results of executing the tests.Hence, using a test package was possible to validate and split into different submodules that were collected in a folder called "tests".The default startapp template creates a tests.py file in the new application, in our case the tests package is divided into different submodules such as test_models.py, test_views.py, test_forms.py that are described as follows:

- ✓ **Test_models.py (Blog) :**

The methods defined in this individual test are "test project_post(self)" to check for an expected result "post 1" and the method "test_project_comment(self)" to check for an expected result "comment 1". Finally, the correct behaviors of the date stored has been tested for each individual blog entry.

- ✓ **Test_forms.py (Blog) :**

In order to allow users to create comments using a common method "test_comment_valid_data(self)" to verify a condition to accept user input on the web site and send that data to a server.

- ✓ **Test_urls.py (Blog) :**

To validate HTTP Request using the method "test_list_path_resolved(self)",we have evaluated the display in our homepage for every path in : Home, About us, Travel tips, famous destinations, post-detail, post-create,post-delete,famdes, post-update, add_comment_to_post, comment_approve and comment_remove.

- ✓ **Test_views.py (Blog) :**

The views behaviour was validated using the Django test Client. We have used it to simulate a GET and POST requests on a URL and observe the response. This allows us to verify that each view is doing what is expected. In addition, the method setUp() was implamented to make an exception while the test is running.

➢ For users was possible to validate and split into different submodules such as follow : Test models.py , Test urls.py , Test views.py :

✓ Test.views.py (Users):

In this test the actual user is available on the request just like a view on the “register” response .We use the test client to load a view and it is possible to store the result and then get the user.

✓ Test.urls.py (Users):

In this part we evaluated the response of the user entries using the method “test_register_path_resolved” for every path such as the register, profile, login and logout.

✓ Test.models.py(Users):

We tested the models using the module “test_project_profile(self)” by arranging our models as factories and running testing logic on them in order to verify the profiles.

✓ Test forms.py (Users):

The test in forms has itself a test already. The method defined “test_registration_valid_data(self)” was used to evaluate when a user was added, an email address had to be supplied in addition to the username and password fields. This part the test evaluates the fields with the piece of information related to username, email and password.

CONCLUSION

WanderLust is a Travel Blog which is created to share the experiences with the world regarding different cultures, food, traditions and more practical information to the travellers. This project includes to develop a software package that can be used to host a travel blogging by using Django framework in phython as backend, Html, Css and Bootstrap. WanderLust converts the travel experiences into a story and shares it with the digital world to boost the reader base with the help of various digital tools. It also creates an interface between Users and Authors. And gives information to the travellers regarding the experiences of author and travel tips that is required for a traveller.

This project WanderLust was divided into the following parts:

Front End: which mainly focuses on designing of the web page with the help of html, css and bootstrap

Back End: which mainly focuses on Django framework of python where functionalities of the blog were added such as Login, Blog Posting, Commenting, Authority of Registered and Unregistered users, Tags, Search Bar.

Testing: Our blog (Urls, Views, Models and forms) has been fully tested using the built in Unittest in Django framework.

By working on this project, we came to know how the designing of the website is done in frontend and backend. On the completion of project, we have a thorough idea as to how the programming is done in the Python language using Django as its framework. So, in overall it was a wonderful experience and acquired some software skills which will be beneficial for us.

REFERENCES

1. https://en.wikipedia.org/wiki/Unified_Modeling_Language

2. https://docs.djangoproject.com/en/2.2/intro/tutorial01/

3. https://rogerdudler.github.io/git-guide/index.html

4. https://rogerdudler.github.io/git-guide/index.html

5. https://docs.python.org/3/tutorial/

6. https://repl.it/languages/python3

7. https://www.anaconda.com/distribution/

8. http://www.learnexia.com/?p=857

9. Adrian Holovaty, The Definitive Guide to Django: Web Development