Procesul Detaliat de Creație al Proiectului Travel Blog

Proiectul "**Blogul meu de călătorii**" a fost conceput pentru a oferi utilizatorilor o platformă modernă și interactivă, dedicată iubitorilor de călătorii. Scopul principal al proiectului a fost crearea unei aplicații web complet funcționale, folosind framework-ul Django, cu o interfață prietenoasă, optimizată atât pentru utilizatori finali, cât și pentru administratori. Mai jos este descris în detaliu procesul de dezvoltare al aplicației, incluzând fiecare etapă importantă.

1. Configurarea Mediului de Lucru

Procesul de dezvoltare a început prin configurarea mediului de lucru:

1. **Instalarea Python**: Versiunea 3.8+ a fost utilizată pentru a asigura compatibilitatea cu Django și alte pachete necesare.
2. **Configurarea unui mediu virtual**: A fost creat un mediu virtual pentru a izola dependențele proiectului, utilizând următoarele comenzi:

```
python -m venv venv

source venv/bin/activate  # Pe macOS/Linux
venv\Scripts\activate     # Pe Windows
```

3. **Instalarea Django**: Ultima versiune de Django a fost instalată folosind comanda:

```
pip install django
```

Ulterior, a fost creat proiectul principal utilizând comanda:

```
django-admin startproject travel_blog .
```

și o aplicație separată pentru gestionarea funcționalităților blogului:

```
python manage.py startapp blog
```

2. Organizarea Structurii Proiectului

Proiectul a fost organizat conform următoarei structuri:

- **blog/**: Aplicația principală care conține modelele, vizualizările și template-urile specifice articolelor.
- **static/**: Directorul pentru fișiere CSS, imagini și scripturi JavaScript.
- **templates/**: Template-urile HTML utilizate pentru a construi interfața aplicației.
- **media/**: Un director special pentru fișierele încărcate de utilizatori (de ex., imagini pentru articole).

În fișierul `settings.py`, au fost configurate rutele pentru fișierele media și statice:

```
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'
STATIC_URL = '/static/'
STATICFILES_DIRS = [BASE_DIR / 'static']
```

3. Dezvoltarea Funcționalităților

a) Crearea și Gestionarea Articolelor

Un model de bază pentru articole a fost definit în `blog/models.py`:

```python
class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    image = models.ImageField(upload_to='post_images/', blank=True, null=True)
    video_url = models.URLField(blank=True, null=True)
    created_at = models.DateTimeField(default=timezone.now)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.title
```

Acest model include titlu, conținut, imagine opțională, un link pentru videoclip și un timestamp.

Am aplicat migrațiile pentru a crea baza de date:

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

b) Implementarea CRUD

Funcționalitățile Create, Read, Update și Delete (CRUD) au fost implementate:

- **Adăugarea articolelor**: Un formular HTML a fost creat și integrat cu `forms.py` pentru validarea datelor.

```python
class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'content', 'image', 'video_url']
```

- **Listarea articolelor**: Articolele sunt afișate într-o pagină utilizând un view bazat pe funcție:

```python
def post_list(request):
    query = request.GET.get('q')
    if query:
        posts = Post.objects.filter(title__icontains=query).order_by('-created_at')
    else:
        posts = Post.objects.all().order_by('-created_at')
    return render(request, 'blog/post_list.html', {'posts': posts, 'query': query})
```

- **Editarea și ștergerea articolelor**: Link-uri dedicate au fost incluse în template-uri pentru a permite utilizatorilor autentificați să modifice articole.

```python
def post_create(request):
    if request.method == "POST":
        form = PostForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            return redirect('post_list')
    else:
        form = PostForm()
    return render(request, 'blog/post_form.html', {'form': form})

def post_edit(request, post_id):
    post = get_object_or_404(Post, pk=post_id)
    if request.method == 'POST':
        form = PostForm(request.POST, request.FILES, instance=post)
        if form.is_valid():
            form.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm(instance=post)
    return render(request, 'blog/post_form.html', {'form': form})

def post_delete(request, post_id):
    post = get_object_or_404(Post, id=post_id)
    if request.method == "POST":
        post.delete()
        return redirect('post_list')
    return render(request, 'blog/post_confirm_delete.html', {'post': post})
```

4. Funcționalități Avansate

a) Funcționalitate de Căutare

Am implementat o bară de căutare care permite utilizatorilor să găsească articole după titlu. Aceasta folosește filtrarea bazată pe `icontains`:

```python
def post_list(request):
    query = request.GET.get('q')
    if query:
        posts = Post.objects.filter(title__icontains=query).order_by('-created_at')
    else:
        posts = Post.objects.all().order_by('-created_at')
    return render(request, 'blog/post_list.html', {'posts': posts, 'query': query})
```

b) Integrarea fișierelor media

Imaginile și videoclipurile încărcate de utilizatori sunt gestionate prin intermediul câmpurilor `ImageField` și `URLField`. În template-uri, imaginile sunt afișate astfel:

```
<h3 class="card-title">{{ post.title }}</h3>
{% if post.image %}
    <img src="{{ post.image.url }}" alt="{{ post.image }}" class="img-fluid d-block mx-auto mb-3">
{% endif %}
{% if post.video_url %}
    <div class="embed-responsive embed-responsive-16by9">
        <iframe class="embed-responsive-item" src="{{ post.video_url }}" allowfullscreen></iframe>
    </div>
{% endif %}
```

c) Exportul unui articol în format PDF

Pentru fiecare articol, am adăugat o funcționalitate care permite descărcarea în format PDF, folosind biblioteca `reportlab`:

```
def export_post_pdf(request, pk):

    post = Post.objects.get(pk=pk)

    response = HttpResponse(content_type='application/pdf')
    response['Content-Disposition'] = f'attachment; filename="{post.title}.pdf"'

    pdf = canvas.Canvas(response)
    pdf.setTitle(post.title)

    logo_path = settings.BASE_DIR / 'blog/static/blog/logo.png'
    logo = ImageReader(logo_path)
    pdf.drawImage(logo, 50, 750, width=100, height=50)

    pdf.setFont("Helvetica-Bold", 20)
    pdf.drawString(50, 700, post.title)

    pdf.setFont("Helvetica", 12)
    text = pdf.beginText(50, 650)
    text.textLines(post.content)

    pdf.drawText(text)
    pdf.showPage()
    pdf.save()

    return response
```

5. Optimizarea Designului

a) Utilizarea Bootstrap

Bootstrap a fost utilizat pentru a construi un design responsive și modern:

- Navbar-ul include link-uri către paginile principale și o bară de căutare.
- Cardurile Bootstrap sunt utilizate pentru afișarea articolelor, oferind un aspect curat și organizat.

b) Personalizarea Fundalului

Am adăugat o imagine tematică de fundal corespunzatoare temei.

c) Footer sticky

Footer-ul a fost configurat pentru a rămâne la baza paginii, indiferent de lungimea conținutului:

```
body {
    display: flex;
    flex-direction: column;
    min-height: 100vh;
}

footer {
    background-color: #333;
    color: white;
    text-align: center;
    padding: 15px 0;
    margin-top: auto;
}
```

6. Testare și Finalizare

După implementarea tuturor funcționalităților, aplicația a fost testată pentru:

- Gestionare a functionalitatilor CRUD.
- Afișarea corectă a articolelor, imaginilor și videoclipurilor.

- Funcționalitatea de căutare și export PDF.