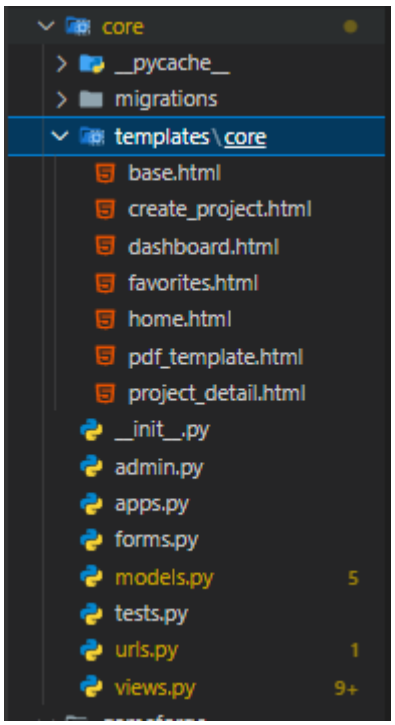
COODIEN Govindarajen

PRS MIA 26.2 | Django & Django Rest Framework

GameForge – Générateur de jeux vidéo par IA

Mon rôle dans le projet était d'être le développeur fullstack, coder la partie qui concerne le VIEWS du coup, créer les fonctions qui traitent les données et les envoyer aux pages HTML dans le dossier templates et du coup aussi coder les pages HTML afin d'avoir un bon rendu de la page à afficher au client.



La hiérarchie des dossiers du projet que j'ai codée en collaboration avec les autres membres de l'équipe

- models.py : (vide) emplacement pour les modèles personnalisés.
- views.py : contient la vue signup_view.
- urls.py : routes login, logout, signup.
- templates/accounts/login.html : template de connexion.
- templates/accounts/signup.html — template d'inscription.

Un exemple d'une page HTML que l'on a structurée avec une fonction JavaScript qui permet aussi de dynamiser la page, les données sont récupérées par les VIEWS et à travers les variables, on les injecte dans l'HTML

```python
User = get_user_model()

class GameProject(models.Model):
    author = models.ForeignKey(User, on_delete=models.CASCADE, related_name="projects")
    title = models.CharField(max_length=200)
    slug = models.SlugField(max_length=220, unique=True, blank=True)
    genre = models.CharField(max_length=100)
    ambiance = models.CharField(max_length=200, blank=True)
    keywords = models.TextField(blank=True)
    references = models.TextField(blank=True)
    is_public = models.BooleanField(default=False)
    generated = models.JSONField(null=True, blank=True)
    image_character = models.ImageField(upload_to="generated/characters/", null=True, blank=True)
    image_environment = models.ImageField(upload_to="generated/environments/", null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def save(self, *args, **kwargs):
        if not self.slug:
            base = slugify(self.title)[:150]
            self.slug = f"{base}-{int(time.time())}"
        super().save(*args, **kwargs)

    def get_absolute_url(self):
        return reverse("core:project_detail", args=[self.slug])

    def __str__(self):
        return f"{self.title} — {self.author.username}"

class Favorite(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="favorites")
    project = models.ForeignKey(GameProject, on_delete=models.CASCADE, related_name="favorited_by")
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        unique_together = ("user", "project")

class ApiUsage(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    day_key = models.CharField(max_length=10)  # format YYYYMMDD
    count = models.IntegerField(default=0)

    class Meta:
        unique_together = ("user", "day_key")
```

Ici on voit les modèles qui nous permettent de structurer et créer une passerelle entre les données entrées par le front et le formulaire vers la base de données.
On voit que l'on définit les données que l'on a pour chaque objet et le type

Les patterns nous permet de définir les chemin url que nous permet de changer page html et du coup afficher une View different.

```python
urlpatterns = [
    path("", views.HomeView.as_view(), name="home"),
    path("search/", views.search_view, name="search"),
    path("dashboard/", views.DashboardView.as_view(), name="dashboard"),
    path("create/", views.CreateProjectView.as_view(), name="create"),
    path("project/<slug:slug>/", views.ProjectDetailView.as_view(), name="project_detail"),
    path("favorite/<slug:slug>/", views.toggle_favorite, name="toggle_favorite"),
    path("favorites/", views.favorites_view, name="favorites"),
    path("export/<slug:slug>/pdf/", views.export_project_pdf, name="export_pdf"),

    # IA
    path("generate/", views.generate_game_view, name="generate"),          # POST JSON {project_id}
    path("explore/", views.explore_free_view, name="explore_free"),        # GET -> crée & génère aléatoire
]
```

```python

class HomeView(ListView):
    model = GameProject
    template_name = "core/home.html"
    context_object_name = "projects"
    paginate_by = 12
    def get_queryset(self):
        return GameProject.objects.filter(is_public=True).order_by("-created_at")

def search_view(request):
    q = request.GET.get("q", "")
    projects = GameProject.objects.filter(is_public=True).filter(
        Q(title__icontains=q) | Q(genre__icontains=q) | Q(keywords__icontains=q) | Q(ambiance__icontains=q)
    ).order_by("-created_at")
    return render(request, "core/home.html", {"projects": projects, "search": q})

class DashboardView(LoginRequiredMixin, ListView):
    model = GameProject
    template_name = "core/dashboard.html"
    context_object_name = "projects"
    def get_queryset(self):
        return GameProject.objects.filter(author=self.request.user).order_by("-created_at")

class ProjectDetailView(DetailView):
    model = GameProject
    template_name = "core/project_detail.html"
    slug_field = "slug"

class CreateProjectView(LoginRequiredMixin, CreateView):
    model = GameProject
    form_class = ProjectCreateForm
    template_name = "core/create_project.html"
    def form_valid(self, form):
        obj = form.save(commit=False)
        obj.author = self.request.user
        obj.save()
        return redirect(obj.get_absolute_url())

def _day_key():
    now = datetime.datetime.utcnow()
    return now.strftime("%Y%m%d")

def _check_quota(user):
    if not user.is_authenticated:
        return False, "Authentification requise."
    day = _day_key()
    usage, _ = ApiUsage.objects.get_or_create(user=user, day_key=day)
    if usage.count >= settings.DAILY_GENERATION_LIMIT:
        return False, f"Limite quotidienne atteinte ({settings.DAILY_GENERATION_LIMIT}). Réessaie demain."
    usage.count += 1
    usage.save()
    return True, ""
```

- Schéma de flux des donnes