Documentation technique

But : document technique ligne-par-ligne pour les fichiers fournis dans le dossier accounts/.

Contexte / résumé

Cette application Django fournit une petite partie « comptes » : pages d'inscription et de connexion en s'appuyant sur les vues d'authentification intégrées de Django (LoginView, LogoutView) et sur UserCreationForm pour la création d'utilisateurs. Les fichiers présents sont :

- admin.py — (vide) point d'enregistrement pour l'admin Django.
- apps.py — configuration de l'application.
- 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.
- tests.py, migrations/__init__.py, __init__.py — fichiers standard, vides/préparatoires.

l'application utilise le modèle User de Django par défaut (aucune extension), et le projet contient un template de base core/base.html (étendu par les templates ci-dessous). Les routes sont incluses dans l'URLConf du projet.

accounts/admin.py

```
from django.contrib import admin

# Register your models here.
```

Ligne par ligne

1. from django.contrib import admin
    - **Rôle** : importe le registre d'administration de Django (utilisé pour enregistrer des modèles afin qu'ils apparaissent dans l'interface d'administration).
    - **Paramètres / retours** : import ; pas de fonction.
    - **Limites / remarques** : aucun modèle n'est enregistré ici. Si vous souhaitez gérer des profils utilisateurs ou des modèles personnalisés via l'admin, il faut ajouter des appels admin.site.register(...) ou des classes ModelAdmin.
2. (ligne vide)
3. # Register your models here.
    - Commentaire généré par startapp. Indique où placer l'enregistrement des modèles.

accounts/apps.py

```
from django.apps import AppConfig


class AccountsConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'accounts'
```

Ligne par ligne

1. from django.apps import AppConfig
    - **Rôle** : importe la classe de configuration d'app Django.
2. class AccountsConfig(AppConfig):
    - **Rôle** : définit la configuration de l'application accounts.
    - **Usage** : utilisé automatiquement si accounts est listé dans INSTALLED_APPS ; permet de configurer des signaux au démarrage si nécessaire.
3. default_auto_field = 'django.db.models.BigAutoField'
    - **Rôle** : indique le type de champ id par défaut pour les nouveaux modèles créés dans cette app (ici BigAutoField).
    - **Limite** : n'affecte que les modèles créés après cette configuration ; si vous migrez depuis des versions plus anciennes, vérifier la cohérence des PKs.
4. name = 'accounts'
    - **Rôle** : nom (label) de l'application tel qu'enregistré par Django.

accounts/models.py

```
from django.db import models

# Create your models here.
```

- Le fichier est un squelette : aucun modèle personnalisé n'est défini.
- **Conséquence** : l'application s'appuie sur django.contrib.auth.models.User (modèle User par défaut). Si vous souhaitez stocker des champs supplémentaires pour l'utilisateur (bio, avatar, rôle, etc.), il faut soit :
    - étendre AbstractUser et remplacer le AUTH_USER_MODEL, ou
    - créer un modèle Profile (OneToOne vers User) et l'enregistrer.
- **Limites** : pas de logique métier stockée côté base de données dans cette app pour l'instant.

accounts/views.py

Contenu complet (annoté ligne par ligne) :

```
01: from django.shortcuts import render, redirect
02: from django.contrib.auth.forms import UserCreationForm
03: from django.contrib.auth import login
04:
05: def signup_view(request):
06:     if request.method == "POST":
07:         form = UserCreationForm(request.POST)
08:         if form.is_valid():
09:             user = form.save()
10:             login(request, user)
11:             return redirect("/dashboard/")
12:     else:
13:         form = UserCreationForm()
14:     return render(request, "accounts/signup.html", {"form": form})
```

Explications détaillées

- **L.01** from django.shortcuts import render, redirect
  - Importe render (renvoie un HttpResponse rendu par un template) et redirect (renvoie un HttpResponseRedirect).
- **L.02** from django.contrib.auth.forms import UserCreationForm
  - Importe le formulaire Django standard pour créer un utilisateur (username, password1, password2 et validation des deux mots de passe).
  - **Paramètres / retours** : c'est une classe de formulaire qui, après is_valid() et save(), crée un objet User.
  - **Limites** : par défaut, UserCreationForm ne gère pas l'email ni les champs personnalisés. Si vous avez besoin d'email obligatoire ou d'autres champs, il faut créer un formulaire personnalisé dérivé de UserCreationForm ou utiliser un ModelForm sur un User étendu.
- **L.03** from django.contrib.auth import login
  - Fonction login(request, user) qui lie la session au user et l'authentifie côté session.
  - **Limites** : n'effectue aucune vérification supplémentaire (ex : email confirmé). Après login() l'utilisateur est considéré connecté.
- **L.05** def signup_view(request):
  - Définition de la vue de création d'utilisateur. Prend un objet HttpRequest en paramètre et retourne un HttpResponse.
- **L.06** if request.method == "POST":
  - Branche traitement pour la soumission du formulaire.
- **L.07** form = UserCreationForm(request.POST)
  - Instancie le formulaire avec les données POST.
  - **Paramètres** : request.POST dictionnaire des champs postés.
- **L.08** if form.is_valid():
  - Lance la validation de Django (contrôles de mot de passe, unicité du username, etc.).
  - **Remarques** : si is_valid() est False, le code saute la branche if et arrive en bas pour re-afficher le template avec le formulaire (contenant les erreurs). Le template doit

afficher form.non_field_errors et les erreurs de champs pour informer l'utilisateur — le template fourni affiche form.non_field_errors au moins.

- **L.09** user = form.save()
  - Sauvegarde le nouvel utilisateur dans la base (create User) et renvoie l'instance User créée.
  - **Retour** : instance User.
  - **Limites** : méthode save() utilise la logique du UserCreationForm ; si vous avez overridé User (AUTH_USER_MODEL) et ajouté des champs requis, il faudra adapter save().
- **L.10** login(request, user)
  - Connecte l'utilisateur (ajoute l'ID dans la session). Après cet appel, request.user sera l'utilisateur authentifié.
- **L.11** return redirect("/dashboard/")
  - Redirige l'utilisateur vers /dashboard/ après inscription.
  - **Limites / amélioration** : la route est codée en dur. Bonnes pratiques : utiliser redirect('nom_de_la_vue'), reverse('nom') ou settings.LOGIN_REDIRECT_URL, et gérer le paramètre next pour redirections conditionnelles.
- **L.12-13** else: form = UserCreationForm()
  - Pour une requête GET, on renvoie un formulaire vide à afficher.
- **L.14** return render(request, "accounts/signup.html", {"form": form})
  - Rend le template accounts/signup.html en fournissant le form au contexte.
  - **Retour** : HttpResponse contenant le HTML.

Limitations fonctionnelles et risques

- Pas de vérification d'email / activation : l'utilisateur est connecté immédiatement après création.
- Mot de passe / politique de mot de passe : UserCreationForm applique les validateurs configurés, mais il peut être utile d'ajouter des messages UX et des restrictions supplémentaires.
- Absence de gestion d'erreurs spécifiques (ex : username existant) côté template — il faut s'assurer que le template affiche les erreurs (la template affiche form.non_field_errors mais n'affiche pas explicitement les erreurs de champ à côté de chaque champ s'ils ne sont pas rendus).
- Redirection codée en dur.

accounts/urls.py

```
01: from django.urls import path
02: from django.contrib.auth import views as auth_views
03: from .views import signup_view
04:
05: urlpatterns = [
06:     path("login/", auth_views.LoginView.as_view(template_name="accounts/login.html"),
```

```
name="login"),
07:     path("logout/", auth_views.LogoutView.as_view(), name="logout"),
08:     path("signup/", signup_view, name="signup"),
09: ]
```

Explications

- **L.01** : importe path pour définir des routes.
- **L.02** : importe les vues d'authentification basées sur les classes fournies par Django (LoginView, LogoutView, etc.). Nous les appelons auth_views pour éviter les collisions de noms.
- **L.03** : importe la vue locale signup_view définie dans views.py.
- **L.06** : route login/ -> utilise LoginView et on précise template_name="accounts/login.html" pour rendre le template personnalisé.
    - **Remarque** : LoginView injecte form (une instance de AuthenticationForm) dans le contexte. Si vous écrivez un template manuel (avec <input name="username"> etc.), il fonctionnera tant que les name correspondent (username, password).
    - **Améliorations** : ajouter redirect_authenticated_user=True si vous voulez empêcher un utilisateur déjà connecté d'accéder à la page de login.
- **L.07** : route logout/ -> LogoutView fournie par Django ; elle déconnecte l'utilisateur et, par défaut, rend un template registration/logged_out.html si présent, sinon redirige selon LOGOUT_REDIRECT_URL si configuré.
- **L.08** : route signup/ -> fonction signup_view.

Point important : le template utilise des liens comme {% url 'accounts:login' %}. Pour que la résolution 'accounts:login' fonctionne il faut : - soit déclarer app_name = 'accounts' dans ce fichier urls.py, - soit inclure ces URL dans la configuration principale avec un namespace : path('accounts/', include(('accounts.urls', 'accounts'), namespace='accounts')).

Actuellement app_name n'est **pas** défini ; si vous incluez accounts.urls sans namespace, la balise {% url 'accounts:login' %} lèvera une NoReverseMatch. Je recommande d'ajouter la ligne app_name = 'accounts' en haut de urls.py.

Templates:

Les templates étendent core/base.html. On suppose que ce template de base contient les blocs title et content.

accounts/templates/accounts/login.html

```
01: {% extends "core/base.html" %}
02: {% block title %}Connexion — GameForge{% endblock %}
03: {% block content %}
04: <div class="auth-card">
05:   <h2>Connexion</h2>
06:   <form method="post">
07:     {% csrf_token %}
08:     <div class="form-row">
09:       <label>Nom d'utilisateur</label>
```

```
10:       <input type="text" name="username" required>
11:     </div>
12:     <div class="form-row">
13:       <label>Mot de passe</label>
14:       <input type="password" name="password" required>
15:     </div>
16:     <button class="btn primary">Se connecter</button>
17:   </form>
18:   <p class="muted">Pas de compte ? <a href="{% url 'accounts:signup' %}">Créer un
compte</a></p>
19: </div>
20: {% endblock %}
```

Explications et points d'attention

- **Sécurité** : le token CSRF est présent (L.07) — indispensable.
- **Noms d'inputs** : username et password correspondent à ce que AuthenticationForm et LoginView attendent — c'est correct.
- **Affichage des erreurs** : le template **n'affiche pas** les erreurs du formulaire (ex. "identifiants incorrects"). Si LoginView renvoie le template avec des erreurs, elles ne seront pas visibles à l'utilisateur. Recommandation : afficher {{ form.non_field_errors }} et éventuellement {{ form.username.errors }} / {{ form.password.errors }} ou utiliser {{ form.as_p }}.
- **UX** : aucun champ next n'est inclus — si l'utilisateur arrive via une URL protégée, Django ajoute normalement ?next=... ; pour le gérer proprement, ajouter <input type="hidden" name="next" value="{{ next }}">.

accounts/templates/accounts/signup.html (ligne par ligne)

```
01: {% extends "core/base.html" %}
02: {% block title %}Inscription — GameForge{% endblock %}
03: {% block content %}
04: <div class="auth-card">
05:   <h2>Inscription</h2>
06:   <form method="post">
07:     {% csrf_token %}
08:     {{ form.non_field_errors }}
09:     <div class="form-row">
10:       <label>Nom d'utilisateur</label>
11:       {{ form.username }}
12:     </div>
13:     <div class="form-row">
14:       <label>Mot de passe</label>
15:       {{ form.password1 }}
16:     </div>
17:     <div class="form-row">
18:       <label>Confirmer le mot de passe</label>
19:       {{ form.password2 }}
20:     </div>
21:     <button class="btn primary">Créer mon compte</button>
```

```
22:   </form>
23:   <p class="muted">Déjà inscrit ? <a href="{% url 'accounts:login' %}">Connexion</a></p>
24: </div>
25: {% endblock %}
```

Explications et points d'amélioration

- {{ form.non_field_errors }} est affiché (L.08) — c'est utile pour voir des erreurs globales (ex: mots de passe différents).
- Les champs sont rendus individuellement via {{ form.username }}, {{ form.password1 }}, {{ form.password2 }} — cela permet de styler chaque champ.
- Les erreurs de champ (par ex. form.username.errors) **ne sont pas** affichées explicitement près des champs : selon la configuration du widget, Django insère déjà des id et classes ; mais il est préférable d'afficher aussi {{ form.username.errors }} pour une meilleure UX.
- Absence d'email : UserCreationForm par défaut ne demande pas l'email. Si l'email est requis dans votre produit, modifiez le formulaire.

tests.py

```
from django.test import TestCase

# Create your tests here.
```

- Pas de tests implémentés pour l'instant. On ajoutera les tests suivant :
    - test de soumission GET/POST de signup_view,
    - test d'accès à login/logout,
    - test de redirection après inscription.