AURA DOCUMENTATION TECHNIQUE:

1. Guide de développement et d'intégration

Plusieurs outils sont nécessaires à son exploitation en local : l'utilisateur doit avoir téléchargé Python 3.10 ou une version supérieure sur son ordinateur, ainsi que pip pour le gestionnaire de paquets Python.

L'application est exploitable en local depuis un ordinateur, plusieurs outils sont requis pour son exploitation : l'utilisateur doit avoir téléchargé sur son ordinateur python 3.10 ou une version supérieure, pip qui est un gestionnaire de packets python, et Git pour le contrôle de version.

Une fois le répertoire de l'application, c'est-à-dire le dossier "AURA" placé sur son ordinateur, l'utilisateur doit ouvrir un terminal Powershell, ou bien un Terminal dans son IDE de préférence et exécuter les commandes suivantes :

Il doit d'abord créer un environnement virtuel :

```
# Windows
python -m venv venv
.\venv\Scripts\activate


# Linux/Mac
python3 -m venv venv
source venv/bin/activate
```

L'utilisateur doit ensuite installer les dépendances :

```
pip install -r requirements.txt
```

Puis Configurer les variables d'environnement : Créer un fichier .env à la racine du projet :

```
DEBUG=True
SECRET_KEY=votre_cle_secrete
DATABASE_URL=sqlite:///db.sqlite3
```

Et initialiser la base de données :

```
# Créer les migrations
python manage.py makemigrations
# Appliquer les migrations
python manage.py migrate
# Créer un superutilisateur (optionnel)
```

```
python manage.py createsuperuser
```

Enfin il doit s'assurer que son environnement virtuelle est activé et qu'il est bien présent dans le répertoire « AURA\partyhub\partyhub » . Il peut alors lancer l'application grâce a la commande :

```
# Lancer le serveur

python manage.py runserver
```

Le site Web de l'application AURA sera accessible à l'adresse : http://127.0.0.1:8000/login

où il peut alors créer un compte en renseignant un nom d'utilisateur, une adresse mail et un mot de passe. Puis, il se connecte en utilisant ces mêmes identifiants et il accède alors à la plateforme AURA. Quand l'utilisateur accède pour la première fois à l'application, la plupart des champs des différents onglets sont vides. En se connectant, l'utilisateur est par défaut redirigé vers le premier onglet : le Tableau de bord. Il peut y créer des notes, qui sont des sortes de fiches Memo, il peut aussi renseigner des liens utiles vers les plateformes qui l'intéressent ou dont il a souvent besoin.

Il peut consulter le nombre d'évènements et de contacts qu'il possède ainsi qu'un calendrier mensuel sur lequel il peut retrouver les différents évènements pour lesquels il a renseigné une date. La plage horaire concernant l'évènement sera titrée et colorée sur le calendrier afin qu'il repère rapidement ses disponibilités quand il accède à l'app, en vue d'organiser un nouvel évènement par exemple.

L'utilisateur dispose également de plusieurs autres onglets : *Mes évènements*, dans lequel il peut créer et consulter les différents évènements, déjà datés ou non, sur lesquels il souhaite travailler où dont il souhaite simplement conserver les informations. Pour chaque évènement, il peut renseigner un titre, un lieu, une date et heure de débute et fin, une description du dress code, de la décoration et de la nourriture de l'évènement, il possède également un champ pour une url et toute autre informations complémentaires.

Il peut renseigner les noms des personnes invitées à l'évènement, il peut les ajouter directement depuis sa liste de contacts, mais aussi manuellement. Il peut sélectionner les différents groupes associés à l'évènement (Favoris, Anniversaire, Soirée etc.) et renseigner les dépenses liées à l'évènement : pour chaque dépense, il peut choisir la quantité, le montant par unité et la catégorie correspondant à la dépense (Nourriture, Sécurité, Location, Autres etc.).

Le troisième onglet principal de l'application est l'onglet *Gérer mon budget* dans lequel l'utilisateur retrouve une analyse personnalisée des dépenses des évènements dont il a renseigné la date, on considère que les évènements datés par l'utilisateur sont des évènements confirmés, et pas juste possibles. Un premier graphique circulaire présente à l'utilisateur la répartition de ses dépenses en fonction de leur catégorie associés, un second graphique lui présente ses dépenses totales mensuelles pour chaque année où il a un évènement de planifié.

Enfin, en bas de la page, l'utilisateur retrouve les informations générales concernant le budget de chaque évènement, daté comme non daté, qu'il a créés.

Enfin, l'utilisateur possède un quatrième onglet principal, Mes contacts, dans lequel il peut constituer un carnet d'adresse en créant des contacts. Il peut renseigner leur nom, date de

naissance, adresse, numéro de téléphone, adresse mail, une url, des informations complémentaires, et enfin il peut sélectionner les différents groupes concernant l'utilisateur (Favoris, Famille, amis etc.).

En résumé, afin de suivre l'expérience complète de l'application AURA, l'utilisateur doit créer un compte, se connecter, créer des contacts et des évènements auxquels il peut les inviter, il doit renseigner des budgets et des dépenses de différentes catégories, ce qui lui donnera une analyse de ses dépenses.

2. Structuration du code

Voici la structure de l'application AURA :

```
AURA/                              # Racine du projet
│
├── partyhub/                      # Dossier principal du projet
│   ├── partyhub/                  # Configuration du projet Django
│   │   ├── __init__.py
│   │   ├── asgi.py
│   │   ├── settings.py            # Configuration principale
│   │   ├── urls.py                # URLs principales
│   │   └── wsgi.py
│   │
│   ├── events/                    # Application principale des événements
│   │   ├── migrations/            # Migrations de la base de données
│   │   ├── static/                # Fichiers statiques
│   │   │   ├── css/               # Feuilles de style
│   │   │   ├── js/                # Fichiers JavaScript
│   │   │   └── images/            # Images
│   │   │
│   │   ├── templates/             # Templates HTML
│   │   │   ├── events/            # Templates spécifiques aux événements
│   │   │   │   ├── dashboard.html   # Tableau de bord
│   │   │   │   └── ...              # Autres templates
│   │   │   └── base.html          # Template de base
│   │   │
│   │   ├── __init__.py
│   │   ├── admin.py               # Configuration de l'interface d'administration
```

```
│   │   ├── apps.py          # Configuration de l'application
│   │   ├── models.py        # Modèles de données
│   │   ├── urls.py          # URLs de l'application
│   │   ├── views.py         # Vues principales
│   │   └── api_views.py     # Vues API
│   │
│   ├── manage.py            # Utilitaire de ligne de commande
│   └── db.sqlite3           # Base de données SQLite (développement)
│
├── .gitignore               # Fichiers ignorés par Git
└── README.md                # Documentation du projet
```

Pour notre application AURA, nous avons choisi de suivre une architecture MVT (Modèle-Vue-Template), typique de Django. Le fichier urls.py centralise le routage des requêtes vers les vues appropriées dans *views.py* et *api_views.py*.

Les modèles dans models.py définissent la structure des données et interagissent avec la base de données via l'ORM de Django. Les vues traitent la logique métier, récupèrent les données via les modèles, et les transmettent aux templates HTML situés dans le dossier *templates/.* Cette architecture permet un chargement sécurisé et personnalisé des données de l'utilisateur directement dans l'interface, assurant que chaque utilisateur ne voit que les informations qui lui sont destinées.

Les fichiers statiques (CSS, JavaScript, images) dans *static/* sont servis séparément et référencés par les templates. Les fichiers de migration dans *migrations/* gèrent les évolutions du schéma de base de données, tandis que *admin.py* configure l'interface d'administration Django. Cette séparation claire des responsabilités facilite la maintenance et l'évolution de l'application.