

Conception et Développement d'une Plateforme de Gestion Associative avec Base de Données Relationnelle

04.04.2025

Fouad Id Gouahmane / Hugo Wendjaneh

EFREI

ING1-LSI-APP-B

Contexte

Notre projet PGA (Plateforme de Gestion Associative) a été développé dans le cadre du cours de Base de Données pour les étudiants de LSI2025. Face aux défis de coordination des membres, des événements et des projets au sein des associations étudiantes, nous avons conçu une solution complète permettant de centraliser ces informations et d'améliorer la collaboration.

Notre plateforme permet la gestion de plusieurs aspects de la vie associative étudiante :

Gestion des utilisateurs (membres et administrateurs) Organisation et suivi d'événements
Gestion de projets collaboratifs Forum de discussion thématique

Ce rapport présente notre approche de conception, les choix techniques, et détaille l'implémentation de la base de données ainsi que l'ensemble des requêtes SQL utilisées pour interagir avec la base de données PostgreSQL qui sous-tend notre application.

Partie 1: Back-End

1. Modèle Conceptuel de Données (MCD)

Nous avons utilisé la méthode MERISE pour concevoir notre base de données relationnelle. Cette approche méthodique nous a permis d'identifier clairement les entités, leurs attributs et les relations entre elles.

Notre modèle conceptuel de données s'articule autour de plusieurs entités principales:

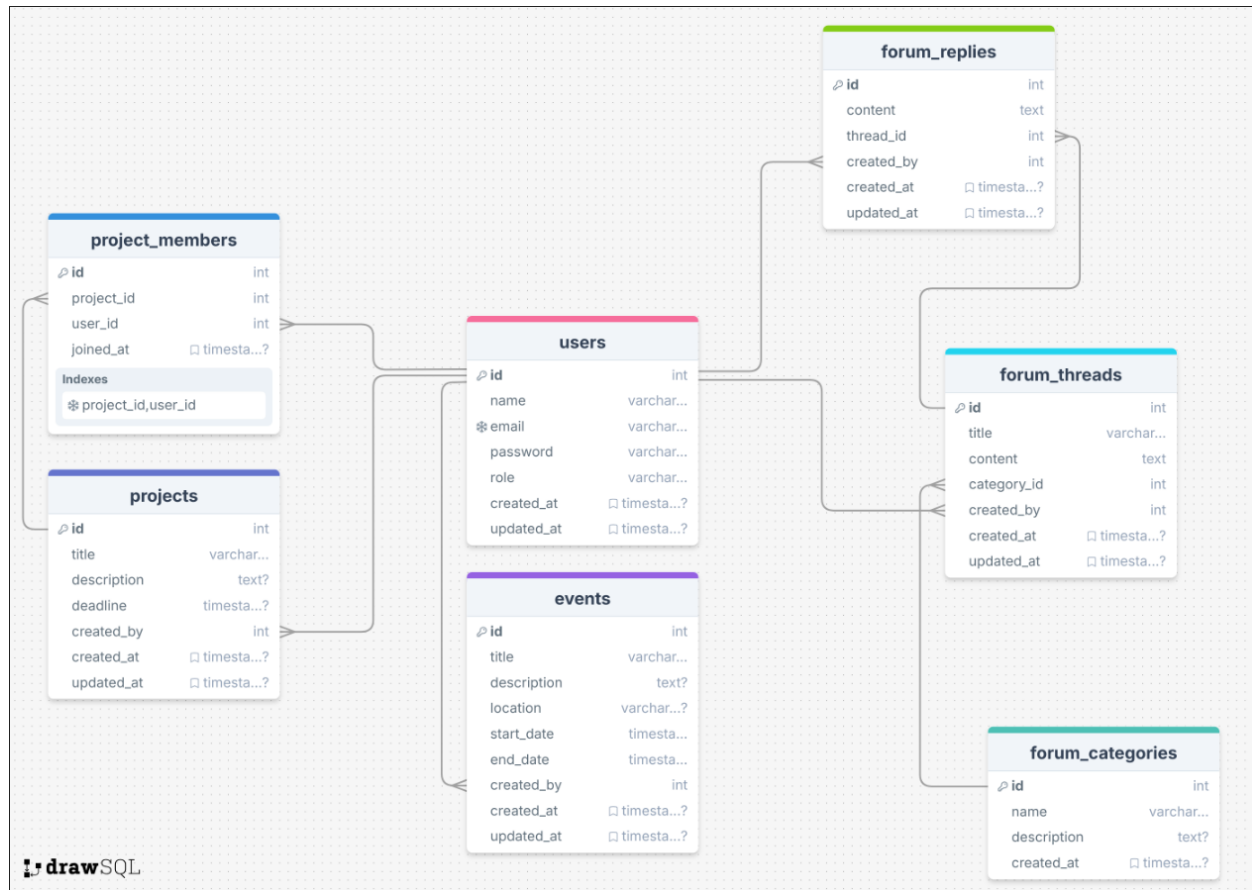
User: Représente les utilisateurs de la plateforme avec leurs informations personnelles et leur rôle

Event: Stocke les événements organisés par l'association

Project: Permet de suivre les projets en cours

Forum: Structure organisée en catégories, threads et réponses pour les discussions

Les relations entre ces entités sont clairement définies, comme par exemple la relation many-to-many entre les utilisateurs et les projets, matérialisée par la table `project_members`.



2. Structure de la Base de Données

Nous avons choisi PostgreSQL comme système de gestion de base de données pour plusieurs raisons:

- Sa robustesse et sa fiabilité

- Son support complet des fonctionnalités SQL avancées

- Sa capacité à gérer efficacement les relations complexes
- Son excellente performance avec les requêtes complexes

Notre base de données est composée des tables suivantes :

- users: Stocke les informations des utilisateurs
- events: Contient les détails des événements organisés
- projects: Enregistre les informations sur les projets
- project_members: Table de jonction pour la relation many-to-many entre projets et utilisateurs
- forum_categories: Catégories du forum de discussion
- forum_threads: Sujets de discussion liés aux catégories
- forum_replies: Réponses aux sujets du forum

1. Module de Gestion des Utilisateurs

Récupérer tous les utilisateurs

```
SELECT * FROM users ORDER BY created_at DESC
```

Récupérer un utilisateur par ID

```
SELECT * FROM users WHERE id = $1
```

Récupérer un utilisateur par email

```
SELECT * FROM users WHERE email = $1
```

Créer un utilisateur

```
INSERT INTO users (name, email, password, role) VALUES ($1, $2, $3, $4)  
RETURNING *
```

Mettre à jour un utilisateur

```
UPDATE users  
SET [dynamic fields], updated_at = NOW()  
WHERE id = $X  
RETURNING *
```

Supprimer un utilisateur

```
DELETE FROM users WHERE id = $1 RETURNING *
```

2. Module de Gestion des Événements

Récupérer tous les événements

```
SELECT e.*, u.name as creator_name  
FROM events e  
JOIN users u ON e.created_by = u.id  
ORDER BY e.start_date ASC
```

Récupérer un événement par ID

```
SELECT e.*, u.name as creator_name  
FROM events e  
JOIN users u ON e.created_by = u.id  
WHERE e.id = $1
```

Récupérer les événements à venir

```
SELECT e.*, u.name as creator_name  
FROM events e  
JOIN users u ON e.created_by = u.id  
WHERE e.start_date > NOW()  
ORDER BY e.start_date ASC  
LIMIT $1
```

Créer un événement

```
INSERT INTO events (  
    title,  
    description,  
    location,  
    start_date,  
    end_date,  
    created_by  
)  
VALUES ($1, $2, $3, $4, $5, $6)  
RETURNING *
```

Mettre à jour un événement

```
UPDATE events  
SET [dynamic fields], updated_at = NOW()  
WHERE id = $X  
RETURNING *
```

Supprimer un événement

```
DELETE FROM events WHERE id = $1 RETURNING *
```

3. Module de Gestion des Projets

Récupérer tous les projets

```
SELECT p.*, u.name as creator_name  
FROM projects p  
JOIN users u ON p.created_by = u.id  
ORDER BY p.created_at DESC
```

Récupérer un projet par ID

```
SELECT p.*, u.name as creator_name
FROM projects p
JOIN users u ON p.created_by = u.id
WHERE p.id = $1
```

Récupérer les membres d'un projet

```
SELECT pm.user_id
FROM project_members pm
WHERE pm.project_id = $1
```

Récupérer les projets d'un utilisateur

```
SELECT DISTINCT p.*, u.name as creator_name
FROM projects p
JOIN users u ON p.created_by = u.id
LEFT JOIN project_members pm ON p.id = pm.project_id
WHERE p.created_by = $1 OR pm.user_id = $1
ORDER BY p.created_at DESC
```

Créer un projet (Transaction)

```
BEGIN;
INSERT INTO projects (
  title,
  description,
  deadline,
  created_by
)
VALUES ($1, $2, $3, $4)
RETURNING *;
```

```
INSERT INTO project_members (project_id, user_id)
VALUES ($1, $2);
COMMIT;
```

Mettre à jour un projet

```
UPDATE projects
SET [dynamic fields], updated_at = NOW()
WHERE id = $X
RETURNING *
```

Supprimer un projet

```
DELETE FROM projects WHERE id = $1 RETURNING *
```

Ajouter un membre à un projet

```
INSERT INTO project_members (project_id, user_id)
VALUES ($1, $2)
ON CONFLICT (project_id, user_id) DO NOTHING
```

Retirer un membre d'un projet

```
DELETE FROM project_members
WHERE project_id = $1 AND user_id = $2
RETURNING *
```

Récupérer les détails des membres d'un projet

```
SELECT u.id, u.name, u.email, pm.joined_at
FROM project_members pm
JOIN users u ON pm.user_id = u.id
WHERE pm.project_id = $1
ORDER BY pm.joined_at ASC
```


Rechercher des utilisateurs (pour ajouter des membres)

```
SELECT id, name, email
FROM users
WHERE name ILIKE $1 OR email ILIKE $1
ORDER BY name ASC
```

3. Module de Forum

Récupérer toutes les catégories

```
SELECT * FROM forum_categories
ORDER BY name ASC
```

Récupérer une catégorie par ID

```
SELECT * FROM forum_categories WHERE id = $1
```

Créer une catégorie

```
INSERT INTO forum_categories (name, description)
VALUES ($1, $2)
RETURNING *
```

Mettre à jour une catégorie

```
UPDATE forum_categories
SET [dynamic fields]
WHERE id = $X
RETURNING *
```

Supprimer une catégorie

```
DELETE FROM forum_categories WHERE id = $1 RETURNING *
```

Récupérer les sujets d'une catégorie

```
SELECT t.*,  
       u.name as author_name,  
       (SELECT COUNT(*) FROM forum_replies WHERE thread_id = t.id) asreply_count  
FROM forum_threads t  
JOIN users u ON t.created_by = u.id  
WHERE t.category_id = $1  
ORDER BY t.created_at DESC
```

Récupérer un sujet par ID avec son auteur

```
SELECT t.*, u.id as author_id, u.name as author_name  
FROM forum_threads t  
JOIN users u ON t.created_by = u.id  
WHERE t.id = $1
```

Récupérer les réponses à un sujet

```
SELECT r.*, u.id as author_id, u.name as author_name  
FROM forum_replies r  
JOIN users u ON r.created_by = u.id  
WHERE r.thread_id = $1  
ORDER BY r.created_at ASC
```

Créer un sujet

```
INSERT INTO forum_threads (title, content, category_id, created_by)  
VALUES ($1, $2, $3, $4)  
RETURNING *
```

Mettre à jour un sujet

```
UPDATE forum_threads  
SET [dynamic fields], updated_at = NOW()  
WHERE id = $X  
RETURNING **
```

Supprimer un sujet

```
DELETE FROM forum_threads WHERE id = $1 RETURNING *
```

Créer une réponse

```
INSERT INTO forum_replies (content, thread_id, created_by)  
VALUES ($1, $2, $3)  
RETURNING *
```

Mettre à jour une réponse

```
UPDATE forum_replies  
SET content = $1, updated_at = NOW()  
WHERE id = $2  
RETURNING *
```

Supprimer une réponse

```
DELETE FROM forum_replies WHERE id = $1 RETURNING *
```

Récupérer une réponse par ID

```
SELECT * FROM forum_replies WHERE id = $1Architecture
```

3. Explications des Choix de Conception

Dans le cadre de ce projet, nous avons fait plusieurs choix de conception pour notre base de données :

1. Contraintes d'intégrité référentielle : Nous utilisons ON DELETE CASCADE pour maintenir l'intégrité des données. Par exemple, lorsqu'un utilisateur est supprimé, tous ses événements, projets et contributions au forum sont également supprimés.
2. Horodatage automatique : Toutes les tables incluent des champs `created_at` et `updated_at` qui sont automatiquement gérés pour suivre l'historique des modifications.
3. Indexation : Des index ont été créés sur les colonnes fréquemment utilisées dans les clauses WHERE et JOIN pour optimiser les performances des requêtes.
4. Transactions : Pour les opérations complexes comme la création de projets (qui implique l'insertion dans plusieurs tables), nous utilisons des transactions pour garantir l'intégrité des données.
5. Contraintes uniques : Pour éviter les duplications, nous avons ajouté des contraintes uniques sur certaines combinaisons de colonnes, comme l'email des utilisateurs et les associations projet-membre.

Partie 2: Front-End

1. Description des fonctionnalités

Notre application frontend offre une interface intuitive pour accéder à toutes les fonctionnalités de la plateforme:

Authentification et gestion des utilisateurs

- Inscription et connexion sécurisées
- Profil utilisateur personnalisable
- Gestion des rôles et permissions

Gestion des événements

- Consultation du calendrier des événements
- Création et modification d'événements (pour les administrateurs)
- Affichage détaillé des informations d'un événement

Gestion des projets

- Vue d'ensemble de tous les projets
- Section "Mes projets" pour visualiser les projets dont l'utilisateur est membre
- Création et modification de projets
- Gestion des membres d'un projet (ajout/suppression)
- Suivi de l'avancement via les échéances

Forum de discussion

- Organisation par catégories thématiques
- Création de nouveaux sujets
- Réponse aux discussions existantes
- Recherche de contenus

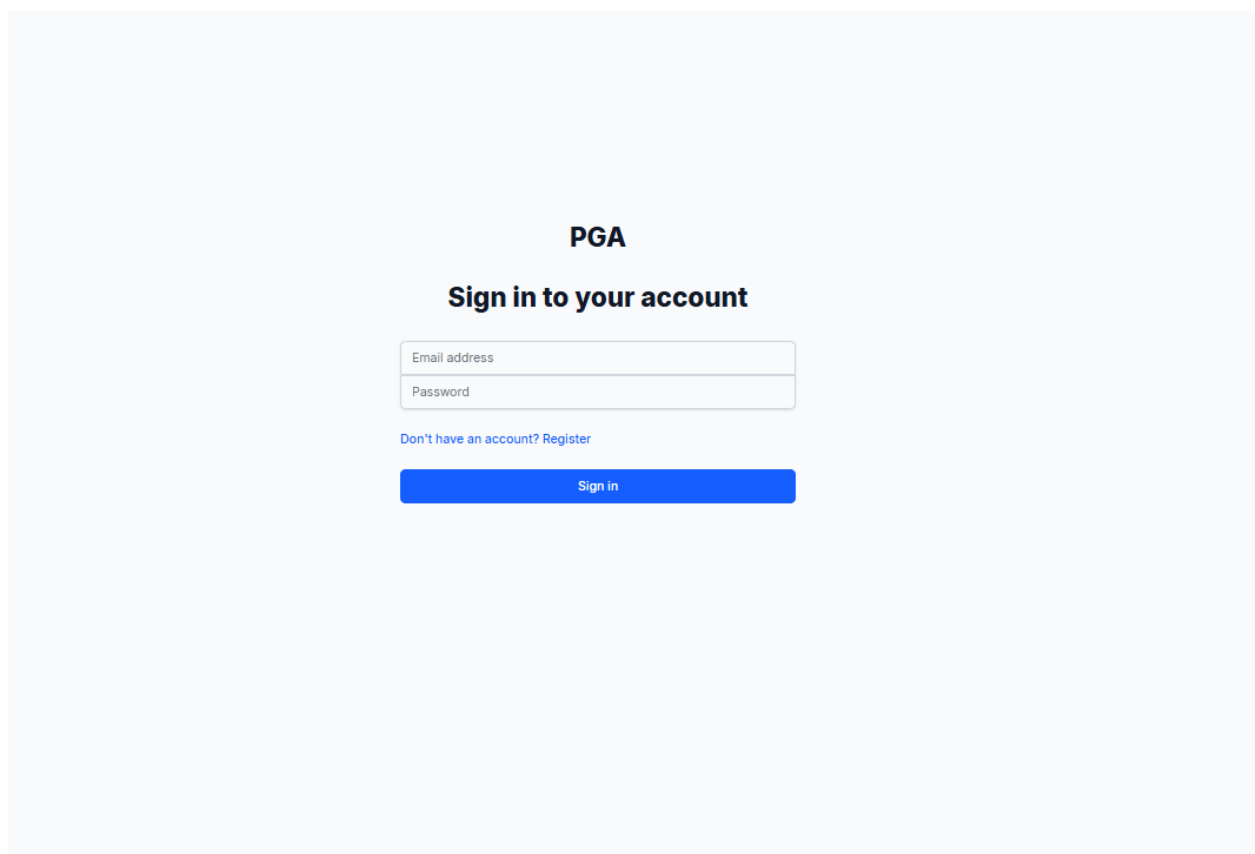
Interface administrateur

- Gestion complète des utilisateurs
- Supervision de tous les contenus
- Possibilité de modérer le forum

2. Interface graphique

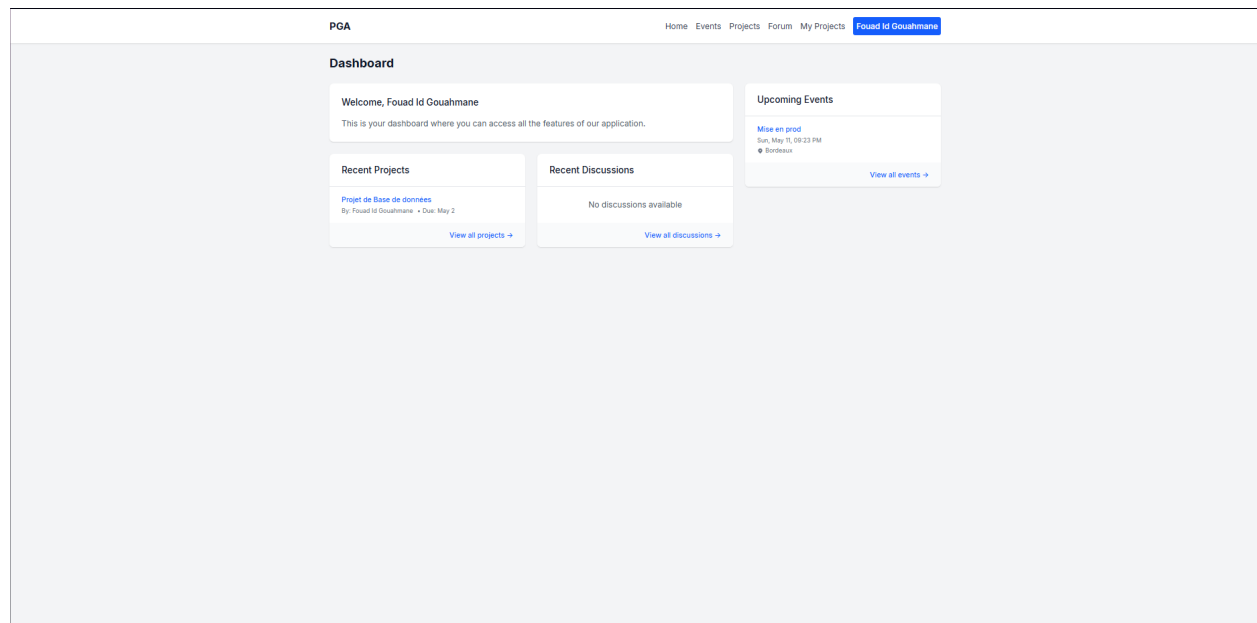
Notre interface graphique a été conçue avec une approche moderne et responsive, s'adaptant à tous les types d'appareils. Voici quelques captures d'écran des principales sections:

Page de connexion

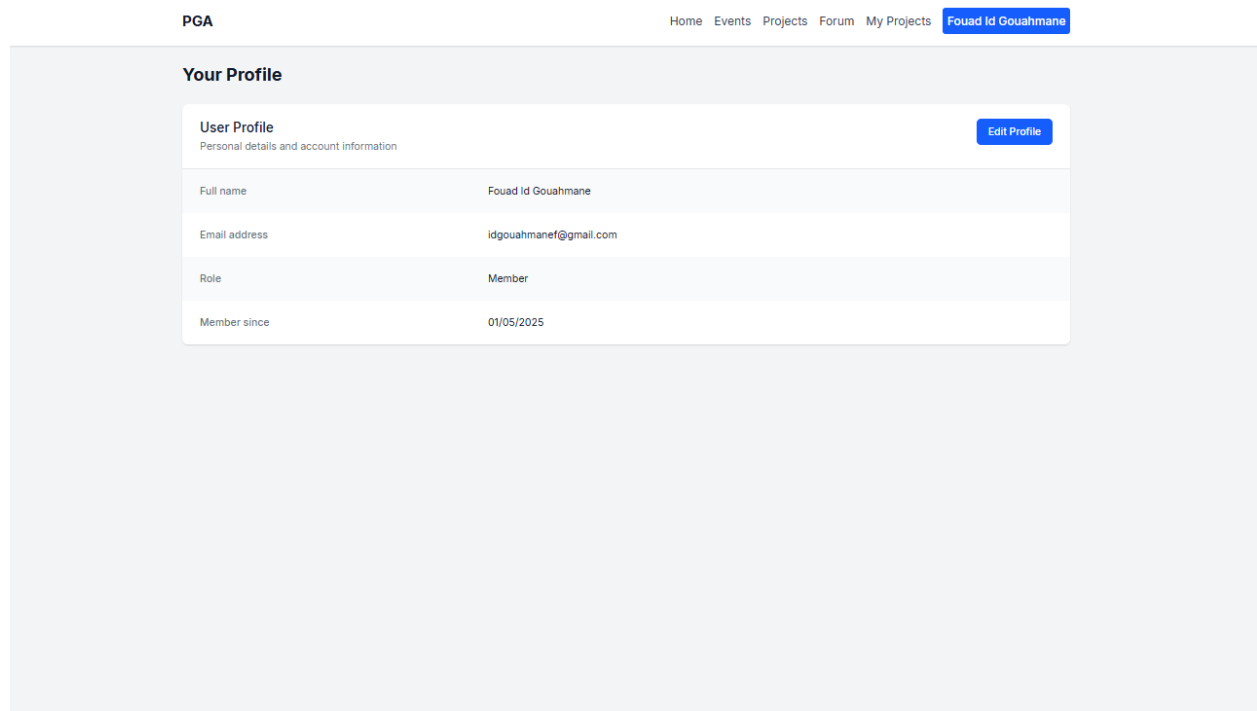


The screenshot shows a login page for 'PGA'. The page has a light gray background. At the top center, the text 'PGA' is displayed in bold black font. Below it, the text 'Sign in to your account' is also in bold black font. There are two input fields: 'Email address' and 'Password', both with light gray borders and placeholder text. Below the password field, there is a link that says 'Don't have an account? Register' in a small blue font. At the bottom, there is a blue button with the text 'Sign in' in white.

Page d'accueil



Page "Profile"



Page “Evenement”

PGA[Home](#)[Events](#)[Projects](#)[Forum](#)[My Projects](#)**Fouad Id Gouahmane**

Events[Create Event](#)

Mise en prod[Edit](#) [Delete](#)
Sun, May 11, 2025 - 09:23 PM to 10:23 PM
On fait une MEP.
📍 Bordeaux
[View details →](#)

Page création d'événement

PGA[Home](#)[Events](#)[Projects](#)[Forum](#)[My Projects](#)**Fouad Id Gouahmane**

[← Back to Events](#)

Create New Event

Event Title

Description

Location

Start Date & Time01/05/2025 21:26

End Date & Time01/05/2025 22:26


Cancel

Create Event

Page “Projet”

PGA[Home](#)[Events](#)[Projects](#)[Forum](#)[My Projects](#)[Fouad Id Gouahmane](#)

All Projects[New Project](#)

[Latest first](#) 

Projet de Base de données

Projet de mise en place d'une Base de donnée

Created: May 1, 2025

Deadline: **May 2, 2025**

[View Details](#)[Edit](#)

Page création de projet

PGA Home Events Projects Forum My Projects Fouad Id Gouahmane

Create New Project

Create a new project for collaboration with other members.

Project Title *

Description

Deadline (Optional)

dd/mm/yyyy

Set a target completion date for this project

Cancel

Create Project

Page "Forum"

PGA Home Events Projects Forum My Projects Fouad Id Gouahmane

Community Forum

Welcome to our community forum. Browse categories below or start a new discussion.

Forum Categories

Events

Discussions about upcoming and past events

0 threads
Created: 01/05/2025

General

General discussions about the association

0 threads
Created: 01/05/2025

Help & Support

Get help with any association-related questions

0 threads
Created: 01/05/2025

Projects

Discussions about ongoing projects

0 threads
Created: 01/05/2025

Exemple Forum (ici le forum général)

PGA

HomeEventsProjectsForumMy ProjectsFouad Id Gouahmane

Forums > **General**

General discussions about the association

Threads

New Thread

No threads in this category yet

[Start a new discussion](#)

Page “Mes projets”

PGA

HomeEventsProjectsForumMy ProjectsFouad Id Gouahmane

My Projects

New Project

Search my projects...

Latest ▾

Projects I Created

Projet de Base de données

Projet de mise en place d'une Base de donnée

Created: May 1, 2025

Deadline: May 2, 2025

View Details

Edit

Projects I'm a Member Of

You're not a member of any projects yet.

Technologies frontend

- Vue 3: Framework JavaScript progressif pour la construction d'interfaces
- TypeScript: Pour un développement plus robuste
- Pinia: Gestion d'état moderne pour Vue
- Vue Router: Navigation fluide entre les différentes vues
- Tailwind CSS: Framework CSS utilitaire pour un design rapide et cohérent
- Vite: Outil de build ultra-rapide

Architecture frontend

- L'architecture frontend suit une organisation claire:
- /components: Composants Vue réutilisables
- /views: Pages complètes de l'application
- /layouts: Structures communes à plusieurs pages
- /stores: Gestion de l'état avec Pinia
- /types: Définitions TypeScript partagées
- /lib: Utilitaires, notamment le client API
- Cette structure modulaire permet de maintenir un code propre et facilement évolutif.

Partie 3: Lancement de l'application

Prérequis

Pour exécuter cette application, vous aurez besoin de:

- Node.js (v14 ou supérieur)
- npm ou yarn
- PostgreSQL (ou Docker pour la version conteneurisée)

Installation et configuration

Option 1: Avec Docker (recommandée)

1. Clonez le dépôt:

```
git clone [URL_DU_REPO]
cd pga
```

2. Lancez les conteneurs Docker:

```
cd api
docker-compose up -d
```

Cela démarrera:

- Une base de données PostgreSQL sur le port 5432
- Adminer (interface d'administration de BDD) sur le port 8080

Option 2: Installation manuelle

1. Clonez le dépôt comme ci-dessus

2. Configuration du backend:

```
cd api
npm install
```

3. Créez un fichier .env dans le dossier api en vous basant sur .env.example

4. Créez une base de données PostgreSQL nommée association_db

5. Lancez le serveur de développement backend:

```
npm run dev
```

6. Configuration du frontend:

```
cd ../app  
npm install
```

7. Lancez le serveur de développement frontend:

```
npm run dev
```

Accès à l'application

Une fois l'application démarrée:

- Le frontend est accessible à l'adresse: <http://localhost:5173>
- L'API backend est accessible à l'adresse: <http://localhost:3000>

Compte administrateur par défaut

Un compte administrateur est créé par défaut pour faciliter la configuration initiale:

Conclusion

Ce projet nous a permis de mettre en pratique les concepts théoriques de conception de bases de données et de SQL appris en cours. L'implémentation d'une plateforme de gestion associative étudiante a constitué un cas d'usage concret et pertinent pour explorer les relations entre entités, les contraintes d'intégrité et l'optimisation des requêtes.

Les requêtes SQL présentées dans ce rapport illustrent les différentes opérations CRUD (Create, Read, Update, Delete) nécessaires pour gérer efficacement les données de notre application, tout en assurant leur cohérence et leur intégrité.

Notre plateforme est désormais fonctionnelle et peut être utilisée par les associations étudiantes pour améliorer leur gestion interne et faciliter la communication entre les membres.

Annexe

Code SQL de la base de donnée

```
-- Users table

CREATE TABLE IF NOT EXISTS users (

    id SERIAL PRIMARY KEY,

    name VARCHAR(100) NOT NULL,

    email VARCHAR(100) NOT NULL UNIQUE,

    password VARCHAR(255) NOT NULL,

    role VARCHAR(20) NOT NULL CHECK (role IN ('admin', 'member')),

    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP

);

-- Events table

CREATE TABLE IF NOT EXISTS events (

    id SERIAL PRIMARY KEY,

    title VARCHAR(100) NOT NULL,

    description TEXT,

    location VARCHAR(255),

    start_date TIMESTAMP WITH TIME ZONE NOT NULL,

    end_date TIMESTAMP WITH TIME ZONE NOT NULL,

    created_by INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,

    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP

);
```

```
-- Projects table
CREATE TABLE IF NOT EXISTS projects (
    id SERIAL PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    description TEXT,
    deadline TIMESTAMP WITH TIME ZONE,
    created_by INTEGER NOT NULL REFERENCES users(id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Project members (many-to-many relationship between projects and users)
CREATE TABLE IF NOT EXISTS project_members (
    id SERIAL PRIMARY KEY,
    project_id INTEGER NOT NULL REFERENCES projects(id) ON DELETE CASCADE,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    joined_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(project_id, user_id)
);

-- Forum categories
CREATE TABLE IF NOT EXISTS forum_categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
```



```
);

-- Forum threads
CREATE TABLE IF NOT EXISTS forum_threads (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    category_id INTEGER NOT NULL REFERENCES forum_categories(id) ON DELETE
    CASCADE,
    created_by INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Forum replies
CREATE TABLE IF NOT EXISTS forum_replies (
    id SERIAL PRIMARY KEY,
    content TEXT NOT NULL,
    thread_id INTEGER NOT NULL REFERENCES forum_threads(id) ON DELETE
    CASCADE,
    created_by INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Create an index for frequently accessed columns
CREATE INDEX IF NOT EXISTS idx_users_email ON users(email);
CREATE INDEX IF NOT EXISTS idx_events_start_date ON events(start_date);
```

```
CREATE INDEX IF NOT EXISTS idx_forum_threads_category_id ON
forum_threads(category_id);

CREATE INDEX IF NOT EXISTS idx_forum_replies_thread_id ON
forum_replies(thread_id);

-- Insert default admin user with password 'admin123'
-- Warning: This is for initialization only, change in production!
INSERT INTO users (name, email, password, role)
VALUES ('Admin', 'admin@example.com',
'$2b$10$J.tKQShgzQXNw9igh0BYf.K6iJHgT0zLhfEXLJ1111DnUNNFYBPHC', 'admin')
ON CONFLICT (email) DO NOTHING;

-- Insert some default forum categories
INSERT INTO forum_categories (name, description)
VALUES
    ('General', 'General discussions about the association'),
    ('Events', 'Discussions about upcoming and past events'),
    ('Projects', 'Discussions about ongoing projects'),
    ('Help & Support', 'Get help with any association-related questions')
ON CONFLICT DO NOTHING;
```