Documentation API CampusWork

API Gateway — Guide complet pour développeurs

Version 1— Dernière mise à jour : Janvier 2026

CampusWork Team

Table des matières

1 Introduction

1.1 Qu'est-ce que l'API Gateway ?

L'API Gateway CampusWork est une couche d'abstraction unifiée qui centralise tous les appels aux services backend :

- Firebase Authentication
- Supabase Database
- Supabase Storage
- Système de notifications

1.2 Avantages

- Une seule interface pour tous les services
- Type-safe avec TypeScript
- Gestion d'erreurs centralisée
- Conversion automatique snake_case / camelCase
- Facilite les tests et mocks
- Évolutif et maintenable

1.3 Import

```
import { apiGateway } from './api/gateway-supabase';
```

2 Architecture

2.1 Structure de l'API Gateway

```
apiGateway
 auth
  login()
  register()
  loginWithGoogle()
  logout()
  resetPassword()

 db
  projects
  posts
  users

 storage
  uploadAvatar()
  uploadCover()
  uploadVideo()
  uploadFile()
  delete()
```

```
 notifications
    create()
    getUserNotifications()
    getUnreadCount()
    markAsRead()
    markAllAsRead()
    delete()
    clearAll()
    notify*()
```

2.2 Flux de données

```
Frontend (React)


API Gateway


Supabase / Firebase
```

3 Authentication API

3.1 `login(email, password)`

Authentifie un utilisateur.

Paramètres

```
email: string
password: string
```

Retour

```
Promise<User>
```

Exemple

```
const user = await apiGateway.auth.login(
  'jean.dupont@univ.edu',
  'motdepasse123'
);
```

3.2 `register(userData)`

Crée un nouveau compte utilisateur.

```
{
  name: string,
  email: string,
  password: string,
  role: 'student' | 'lecturer' | 'admin'
}
```

4 Database API

4.1 Projects

4.1.1 getAll()

```
const projects = await apiGateway.db.projects.getAll();
```

4.1.2 save(project, author)

```
await apiGateway.db.projects.save(projectData, currentUser);
```

4.1.3 delete(projectId)

```
await apiGateway.db.projects.delete('project-123');
```

4.2 Posts

4.2.1 getAll()

```
const posts = await apiGateway.db.posts.getAll();
```

4.2.2 save(post, author)

```
await apiGateway.db.posts.save(postData, currentUser);
```

4.2.3 delete(postId)

```
await apiGateway.db.posts.delete('post-456');
```

4.3 Users

```
await apiGateway.db.users.getAll();
await apiGateway.db.users.update('id', { pending: false });
```

5 Storage API

5.1 uploadAvatar(file)

```
const url = await apiGateway.storage.uploadAvatar(file);
```

5.2 uploadCover(file)

```
const cover = await apiGateway.storage.uploadCover(file);
```

5.3 uploadVideo(file)

```
const video = await apiGateway.storage.uploadVideo(file);
```

5.4 uploadFile(file)

```
const fileData = await apiGateway.storage.uploadFile(file);
```

6 Notifications API

6.1 create(data)

```
await apiGateway.notifications.create({
  user_id: 'user-123',
  type: 'system',
  title: 'Bienvenue',
  message: 'Merci␣de␣rejoindre␣CampusWork',
  read: false
});
```

6.2 Helpers

```
notifyPostLike()
notifyPostReply()
notifyProjectEvaluation()
```

7 Types & Interfaces

7.1 User

```
interface User {
  id: string;
  name: string;
  email: string;
  role: 'student' | 'lecturer' | 'admin';
  pending: boolean;
  banned: boolean;
}
```

7.2 Project

```
interface Project {
  id: string;
  title: string;
  description: string;
  status: 'in_progress' | 'completed';
  likes: number;
}
```

8 Gestion des erreurs

```
try {
  await apiGateway.auth.login(email, password);
} catch (error: any) {
  if (error.code === 'auth/wrong-password') {
    alert('Mot de passe incorrect');
  }
}
```

9 Bonnes pratiques

- Toujours gérer les erreurs avec try/catch
- Valider les données côté frontend
- Utiliser les helpers de notifications
- Centraliser tous les appels via l'API Gateway

Conclusion

L'API Gateway CampusWork fournit une architecture propre, maintenable et extensible pour tous les services backend de la plateforme.