

Spécification fonctionnelle – WorkNest

1. User stories et critères d'acceptation

US-001 – Réserver un espace

- En tant qu'utilisateur client ou entreprise
- Je veux réserver un espace de coworking
- Afin de garantir la disponibilité pour une date et une heure précises

Critères d'acceptation :

- Utilisateur authentifié
- Sélection d'un espace disponible
- Choix des dates et heures
- Vérification des conflits
- Calcul automatique du prix
- Statut PENDING avant paiement
- Paiement requis pour confirmer

US-002 – Payer une réservation

- En tant qu'utilisateur
- Je veux payer une réservation
- Afin de confirmer définitivement ma réservation

Critères d'acceptation :

- Paiement lié à une réservation existante
- Montant exact
- Plusieurs moyens de paiement
- Statut CONFIRMED après paiement

US-003 – Gérer les espaces

- En tant qu'administrateur
- Je veux gérer les espaces
- Afin de maintenir le catalogue à jour

Critères d'acceptation :

- Créer un espace
- Modifier un espace

- Désactiver un espace

US-004 – Consulter le planning

- En tant qu'utilisateur
- Je veux consulter le planning
- Afin de connaître les créneaux disponibles

Critères d'acceptation :

- Affichage des réservations
- Masquage des créneaux passés

US-005 – Gérer son profil utilisateur

- En tant qu'utilisateur
- Je veux gérer mon profil
- Afin de mettre à jour mes informations personnelles

Critères d'acceptation :

- Modification des informations
- Changement de mot de passe

2. Parcours utilisateurs

Parcours 1 – Réservation complète

1. Accès à la page d'accueil
2. Navigation vers le catalogue des espaces
3. Application des filtres
4. Sélection d'un espace
5. Consultation du planning
6. Clic sur « Réserver maintenant »
7. Connexion ou inscription
8. Saisie des dates et heures
9. Affichage du prix total
10. Paiement
11. Confirmation dans le tableau de bord

Parcours 2 – Inscription et première réservation

1. Accès à la page d'inscription
2. Saisie des informations personnelles
3. Choix du type de compte
4. Création du compte
5. Connexion automatique
6. Accès au catalogue
7. Sélection d'un espace
8. Choix du créneau
9. Paiement
10. Confirmation

Parcours 3 – Gestion administrative

1. Connexion en tant qu'administrateur
2. Accès au tableau de bord
3. Création d'un espace
4. Saisie des informations
5. Ajout des équipements et images
6. Sauvegarde
7. Consultation des réservations
8. Validation ou annulation

3. Écrans et wireframes

Page catalogue des espaces

WorkNest	Connexion Inscr.
+-----+	
Filtres : Capacité Prix Ville Équipements	
+-----+	
[Espace 1] [Espace 2] [Espace 3]	
Ville	Ville
Ville	Ville
Prix	Prix
[Détails]	[Détails]
[Détails]	[Détails]
+-----+	

Page détail d'un espace

← Retour au catalogue	
+-----+	
[Image] Nom de l'espace	
	Description
	Ville / Adresse
	Capacité / Prix
	Équipements
	Planning
	[Réserver maintenant]
+-----+	

Page réservation

+-----+	
Réserver : Nom de l'espace	
+-----+	
Date début	Heure début
Date fin	Heure fin

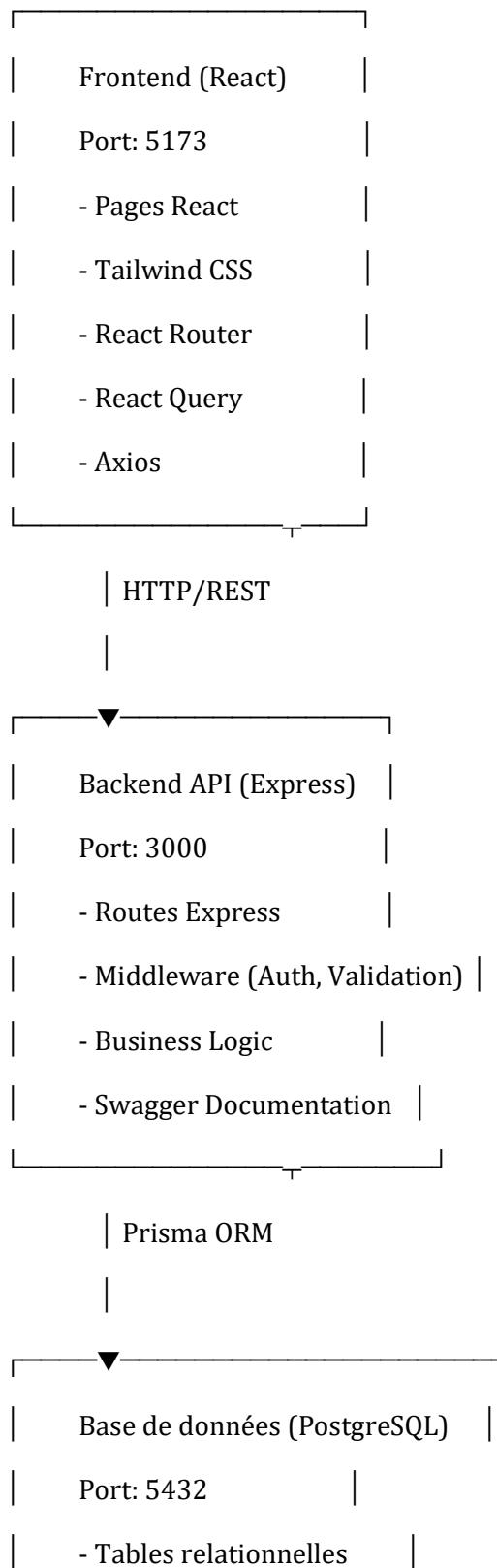
Prix total affiché	

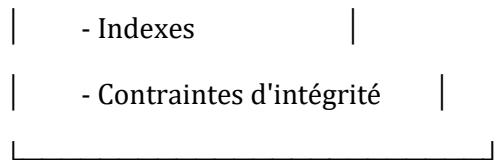
[Confirmer la réservation et payer]	
+-----+	

Spécifications Techniques - WorkNest

1. Architecture globale (Front / API / BDD)

Architecture 3-tiers





Stack technique:

Frontend: - React 18.2 - Vite (build tool) - Tailwind CSS 3.3 - React Router 6.20 - React Query 3.39 - Axios 1.6

Backend: - Node.js 18+ - Express 4.18 - Prisma 5.7 - PostgreSQL 15 - JWT (jsonwebtoken 9.0) - bcryptjs 2.4 -

express-validator 7.0 - Swagger (swagger-ui-express 5.0)

DevOps: - Docker & Docker Compose - Nginx (optionnel pour production)

2. Endpoints API détaillés (méthodes, chemins, payloads)

Authentification

POST /api/auth/register

Méthode: POST

Chemin: /api/auth/register

Description: Inscription d'un nouvel utilisateur

Payload:

```
{  
  "email": "user@example.com",  
  "password": "Password123",  
  "firstName": "John",  
  "lastName": "Doe",  
  "phone": "0612345678",  
  "company": "Ma Société",
```

```
"role": "CLIENT" | "ENTERPRISE"  
}
```

Réponse 201:

```
{  
  "message": "Inscription réussie",  
  "user": {  
    "id": "uuid",  
    "email": "user@example.com",  
    "firstName": "John",  
    "lastName": "Doe",  
    "role": "CLIENT"  
  },  
  "token": "jwt-token"  
}
```

Erreurs: - 400: Données invalides - 409: Email déjà utilisé

POST /api/auth/login

Méthode: POST

Chemin: /api/auth/login

Description: Connexion utilisateur

Payload:

```
{  
  "email": "user@example.com",  
  "password": "Password123"  
}
```

Réponse 200:

```
{  
  "message": "Connexion réussie",  
  "user": {
```

```
        "id": "uuid",
        "email" : "user@example.com",
        "firstName" : "John",
        "lastName" : "Doe",
        "role": "CLIENT"

    },
    "token" : "jwt-token"
}
```

Erreurs: - 401: Identifiants invalides

GET /api/auth/me

Méthode: GET

Chemin: /api/auth/me

Description: Récupérer le profil de l'utilisateur connecté

Headers:

Authorization: Bearer <token>

Réponse 200:

```
{
    "user": {
        "id": "uuid",
        "email" : "user@example.com",
        "firstName" : "John",
        "lastName" : "Doe",
        "role": "CLIENT"
    }
}
```

}

}

Erreurs: - 401: Non authentifié

Espaces

GET /api/spaces

Méthode: GET

Chemin: /api/spaces

Description: Liste des espaces avec filtres

Query Parameters: - capacity (int): Capacité minimale - minPrice (float): Prix minimum - maxPrice (float):

Prix maximum - equipment (string): Équipement recherché - city (string): Ville - page (int): Numéro de page

(défaut: 1) - limit (int): Nombre d'éléments par page (défaut: 10)

Réponse 200:

{

 "spaces" : [

 {

 "id": "uuid",

 "name": "Espace 1",

 "description" : "...",

 "capacity" : 10,

 "price" : 50.0,

 "location" : "Paris",

 "city": "Paris",

 "equipment" : ["WiFi", "Projecteur"],

 "images" : ["url1", "url2"]

 }

],

 "pagination" : {

```
    "page": 1,  
    "limit": 10,  
    "total": 25,  
    "totalPages": 3  
}  
}
```

GET /api/spaces/:id

Méthode: GET

Chemin: /api/spaces/:id

Description: Détails d'un espace avec planning

Paramètres: - id (UUID): Identifiant de l'espace

Réponse 200:

```
{  
  "space": {  
    "id": "uuid",  
    "name": "Espace 1",  
    "description": "...",  
    "capacity": 10,  
    "price": 50.0,  
    "location": "Paris",  
    "address": "123 Rue...",  
    "city": "Paris",  
    "postalCode": "75001",  
    "equipment": ["WiFi"],  
    "images": ["url1"],  
    "reservations": [  
      {
```

```
        "id": "uuid",
        "startDate" : "2024-01-15T10:00:00Z",
        "endDate" : "2024-01-15T12:00:00Z"
    }
]
}
}
```

Erreurs: - 404: Espace non trouvé

POST /api/spaces

Méthode: POST

Chemin: /api/spaces

Description: Créer un espace (Admin uniquement)

Headers:

Authorization: Bearer <admin-token>

Payload:

```
{
    "name": "Nouvel espace",
    "description" : "Description..." ,
    "capacity" : 20,
    "price" : 75.0,
    "location" : "Lyon",
    "address" : "456 Rue..." ,
    "city": "Lyon",
    "postalCode" : "69001",
    "equipment" : ["WiFi", "Projecteur" ],
    "images" : ["url1"]
}
```

Réponse 201:

```
{  
    "message": "Espace créé avec succès",  
    "space": { ... }  
}
```

Erreurs: - 400: Données invalides - 401: Non authentifié - 403: Accès refusé (rôle insuffisant)

PUT /api/spaces/:id

Méthode: PUT

Chemin: /api/spaces/:id

Description: Modifier un espace (Admin uniquement)

Headers:

Authorization: Bearer <admin-token>

Paramètres: - id (UUID): Identifiant de l'espace

Payload:

```
{  
    "name": "Espace modifié",  
    "description": "Nouvelle description...",  
    "capacity": 25,  
    "price": 80.0  
}
```

Réponse 200:

```
{  
    "message" : "Espace modifié avec succès" ,  
    "space" : { ... }  
}
```

Erreurs: - 404: Espace non trouvé - 403: Accès refusé

DELETE /api/spaces/:id

Méthode: DELETE

Chemin: /api/spaces/:id

Description: Désactiver un espace (Admin uniquement)

Headers:

Authorization: Bearer <admin-token>

Paramètres: - id (UUID): Identifiant de l'espace

Réponse 200:

```
{  
    "message" : "Espace désactivé avec succès"  
}
```

Erreurs: - 404: Espace non trouvé - 403: Accès refusé

Réservations

POST /api/reservations

Méthode: POST

Chemin: /api/reservations

Description: Créer une réservation

Headers:

Authorization: Bearer <token>

Payload:

```
{  
    "spaceId" : "uuid",  
    "startDate" : "2024-01-15T10:00:00Z" ,  
    "endDate" : "2024-01-15T12:00:00Z" ,  
    "guests" : 2,  
    "notes" : "Réservation pour la réunion hebdomadaire."  
}
```

```
"endDate" : "2024-01-15T12:00:00Z" ,  
"notes" : "Notes optionnelles"  
}
```

Réponse 201:

```
{  
  "message" : "Réservation créée avec succès" ,  
  "reservation" : {  
    "id": "uuid" ,  
    "userId" : "uuid" ,  
    "spaceId" : "uuid" ,  
    "startDate" : "2024-01-15T10:00:00Z" ,  
    "endDate" : "2024-01-15T12:00:00Z" ,  
    "status" : "PENDING" ,  
    "totalPrice" : 100.0 ,  
    "space" : {  
      "id": "uuid" ,  
      "name": "Espace 1" ,  
      "location" : "Paris" ,  
      "address" : "123 Rue..." ,  
      "city": "Paris"  
    }  
  }  
}
```

}

}

Erreurs: - 400: Dates invalides - 404: Espace non trouvé - 409: Créneau déjà réservé

GET /api/reservations

Méthode: GET

Chemin: /api/reservations

Description: Liste des réservations

Headers:

Authorization: Bearer <token>

Query Parameters: - status (string): PENDING | CONFIRMED | CANCELLED | COMPLETED - spaceId (UUID):

UUID de l'espace (Admin uniquement)

Réponse 200:

{

 "reservations" : [

 {

 "id": "uuid",

 "user Id" : "uuid",

 "spaceId" : "uuid",

 "startDate" : "2024-01-15T10:00:00Z",

 "endDate" : "2024-01-15T12:00:00Z",

 "status" : "CONFIRMED",

 "totalPrice" : 100.0,

 "space" : { ... },

 "payment" : { ... }

 }

]

}

GET /api/reservations/:id

Méthode: GET

Chemin: /api/reservations/:id

Description: Détails d'une réservation

Headers:

Authorization: Bearer <token>

Paramètres: - id (UUID): Identifiant de la réservation

Réponse 200:

```
{  
  "reservation": {  
    "id": "uuid",  
    "userId": "uuid",  
    "spaceId": "uuid",  
    "startDate": "2024-01-15T10:00:00Z",  
    "endDate": "2024-01-15T12:00:00Z",  
    "status": "CONFIRMED",  
    "totalPrice": 100.0,  
    "notes": "...",  
    "space": { ... },  
    "payment": { ... }  
}
```

Erreurs: - 404: Réservation non trouvée - 403: Accès refusé

POST /api/reservations/:id/cancel

Méthode: POST

Chemin: /api/reservations/:id/cancel

Description: Annuler une réservation

Headers:

Authorization: Bearer <token>

Paramètres: - id (UUID): Identifiant de la réservation

Réponse 200:

{

 "message" : "Réservation annulée avec succès"

}

Erreurs: - 400: Réservation déjà annulée ou terminée - 404: Réservation non trouvée - 403: Accès refusé

POST /api/reservations/:id/confirm

Méthode: POST

Chemin: /api/reservations/:id/confirm

Description: Confirmer une réservation (Admin uniquement)

Headers:

Authorization: Bearer <admin-token>

Paramètres: - id (UUID): Identifiant de la réservation

Réponse 200:

{

 "message" : "Réservation confirmée" ,

 "reservation" : { ... }

}

Erreurs: - 404: Réservation non trouvée - 403: Accès refusé

Paiements

POST /api/payments

Méthode: POST

Chemin: /api/payments

Description: Créer un paiement pour une réservation

Headers:

Authorization: Bearer <token>

Payload:

```
{  
  "reservationId": "uuid",  
  "method": "CARD" | "BANK_TRANSFER" | "PAYPAL",  
  "transactionId": "optional-external-id"  
}
```

Réponse 201:

```
{  
  "message": "Paiement effectué avec succès",  
  "payment": {  
    "id": "uuid",  
    ...  
  }  
}
```

```
"reservationId" : "uuid",
"userId" : "uuid",
"amount" : 100.0,
"status" : "PAID",
"method" : "CARD",
"transactionId" : "...",
"paidAt" : "2024-01-15T10:30:00Z"
}
}
```

Erreurs: - 400: Données invalides ou réservation annulée - 404: Réservation non trouvée - 403: Accès refusé -

409: Paiement déjà existant

GET /api/payments

Méthode: GET

Chemin: /api/payments

Description: Liste des paiements

Headers:

Authorization: Bearer <token>

Réponse 200:

```
{
  "payments" : [
    {
      "id": "uuid",
      "reservationId" : "uuid",
      "amount" : 100.0,
      "status" : "PAID",
      "method" : "CARD",
      "reservation" : {
        "space" : {

```

```
        "id": "uuid",
        "name": "Espace 1",
        "location": "Paris"
    }
}
}
]
```

GET /api/payments/:id

Méthode: GET

Chemin: /api/payments/:id

Description: Détails d'un paiement

Headers:

Authorization: Bearer <token>

Paramètres: - id (UUID): Identifiant du paiement

Réponse 200:

```
{
  "payment": {
    "id": "uuid",
    "reservationId": "uuid",
    "userId": "uuid",
    "amount": 100.0,
    "status": "PAID",
    "method": "CARD",
  }
}
```

```
"transactionId" : "...",  
"reservation" : {  
    "space" : { ... }  
}  
}  
}
```

Erreurs: - 404: Paiement non trouvé - 403: Accès refusé

Utilisateurs

GET /api/users/profile

Méthode: GET

Chemin: /api/users/profile

Description: Récupérer le profil utilisateur

Headers:

Authorization: Bearer <token>

Réponse 200:

```
{  
    "user": {  
        "id": "uuid",  
        "email" : "user@example.com",  
        "firstName" : "John",  
        "lastName" : "Doe",  
        "phone" : "0612345678",  
        "company" : "Ma Société",  
        "role": "CLIENT",  
        "createdAt" : "2024-01-01T00:00:00Z",  
        "reservations" : [ ... ]  
    }  
}
```

PUT /api/users/profile

Méthode: PUT

Chemin: /api/users/profile

Description: Mettre à jour le profil utilisateur

Headers:

Authorization: Bearer <token>

Payload:

```
{  
  "firstName": "John",  
  "lastName": "Doe",  
  "phone": "0612345678",  
  "company": "Ma Société"}  
}
```

Réponse 200:

```
{  
  "message": "Profil mis à jour avec succès",  
  "user": { ... }  
}
```

PUT /api/users/profile/password

Méthode: PUT

Chemin: /api/users/profile/password

Description: Changer le mot de passe

Headers:

Authorization: Bearer <token>

Payload:

```
{  
    "currentPassword" : "OldPassword123" ,  
    "newPassword" : "NewPassword123"  
}
```

Réponse 200:

```
{  
    "message" : "Mot de passe changé avec succès"  
}
```

Erreurs: - 401: Mot de passe actuel incorrect - 400: Nouveau mot de passe invalide

GET /api/users

Méthode: GET

Chemin: /api/users

Description: Liste des utilisateurs (Admin uniquement)

Headers:

Authorization: Bearer <admin-token>

Réponse 200:

```
{  
    "users" : [  
        {  
            "id": "uuid",  
            "email" : "user@example.com" ,  
            "name" : "User Name",  
            "role" : "User Role"  
        }  
    ]  
}
```

```
        "firstName" : "John",
        "lastName" : "Doe",
        "company" : "Ma Société",
        "role": "CLIENT",
        "isActive" : true,
        "createdAt" : "2024-01-01T00:00:00Z"
    }
]
}
```

Erreurs: - 403: Accès refusé

3. Gestion des erreurs

Codes HTTP

- 200 OK - Succès
- 201 Created - Ressource créée
- 400 Bad Request - Données invalides
- 401 Unauthorized - Non authentifié
- 403 Forbidden - Accès refusé (rôle insuffisant)
- 404 Not Found - Ressource non trouvée
- 409 Conflict - Conflit (ex: email existant, créneau réservé)

- 500 Internal Server Error - Erreur serveur

Format d'erreur

Erreur simple:

```
{  
  "error": "Message d'erreur"  
}
```

Erreur avec détails de validation:

```
{  
  "error": "Données invalides",  
  "details": [  
    {  
      "field": "email",  
      "message": "Email invalide"  
    },  
    {  
      "field": "password",  
      "message": "Le mot de passe doit contenir au moins 8 caractères"  
    }  
  ]  
}
```

Erreur avec conflit:

```
{  
  "error": "Créneau déjà réservé",  
  "conflicts": [  
    {  
      "id": "uuid",  
      "startDate": "2024-01-15T10:00:00Z",  
      "endDate": "2024-01-15T12:00:00Z"  
    }  
  ]  
}
```

```
}
```

```
]
```

```
}
```

Gestion des erreurs serveur

En mode développement, les erreurs 500 incluent la stack trace :

```
{
```

```
  "error": "Erreur serveur interne",
```

```
  "stack": "..."
```

```
}
```

En mode production, seule l'erreur générique est retournée.

4. Règles métier

Réservations

1. Non-chevauchement : Une réservation ne peut pas chevaucher une autre réservation confirmée ou en attente
2. Dates futures : La date de début doit être dans le futur
3. Ordre des dates : La date de fin doit être après la date de début
4. Calcul automatique : Le prix total est calculé automatiquement (prix/heure × nombre d'heures)
5. Annulation : Une réservation annulée rembourse automatiquement le paiement associé
6. Statuts : Les statuts possibles sont PENDING, CONFIRMED, CANCELLED, COMPLETED

Paiements

1. Unicité : Un paiement est lié à une seule réservation (relation 1:1)
2. Montant : Le montant du paiement doit correspondre au total de la réservation

3. Confirmation : Après paiement, la réservation passe automatiquement au statut CONFIRMED

4. Remboursement : Un paiement remboursé (status: REFUNDED) est lié à une réservation annulée

5. Méthodes : Les méthodes acceptées sont CARD, BANK_TRANSFER, PAYPAL

Espaces

1. Visibilité : Un espace désactivé (isActive=false) n'apparaît plus dans les résultats de recherche

2. Réservations existantes : Les réservations existantes d'un espace désactivé restent valides

3. Capacité : La capacité minimale est de 1 personne

4. Prix : Le prix est exprimé en euros par heure

5. Équipements : Les équipements sont stockés sous forme de tableau de chaînes

Utilisateurs

1. Désactivation : Un utilisateur désactivé (isActive=false) ne peut plus se connecter

2. Rôles : Les rôles possibles sont CLIENT, ENTERPRISE, ADMIN

3. Permissions Admin : Seul un ADMIN peut créer/modifier/supprimer des espaces

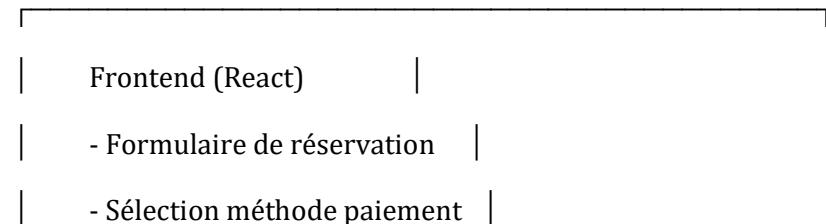
4. Visibilité réservations : Un utilisateur ne peut voir que ses propres réservations (sauf ADMIN qui voit

toutes)

5. Mot de passe : Le mot de passe doit contenir au moins 8 caractères, 1 majuscule, 1 minuscule, 1 chiffre

5. Schéma d'intégration (paiement, calendrier)

Intégration Paiement



```
| POST /api/reservations
```

```
| { spaceId, startDate, endDate }
```

```
| Backend API (Express)
```

```
| 1. Validation dates
```

```
| 2. Vérification conflits
```

```
| 3. Calcul prix total
```

```
| 4. Crédation réservation (PENDING)
```

```
| POST /api/payments
```

```
| { reservationId, method }
```

```
| Backend API (Express)
```

```
| 1. Validation réservation
```

```
| 2. Crédation paiement (PAID)
```

```
| 3. Mise à jour réservation
```

```
| (status: CONFIRMED)
```

Prisma ORM

PostgreSQL

- Table: reservations

- Table: payments

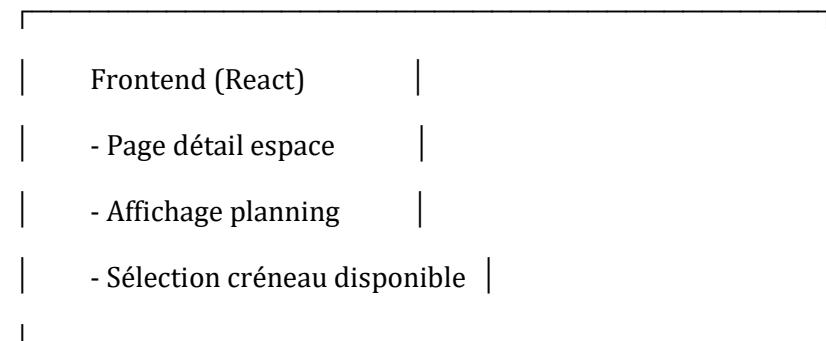
- Transaction atomique

Flux détaillé : 1. L'utilisateur remplit le formulaire de réservation 2. Le frontend envoie POST /api/reservations avec les dates 3. Le backend vérifie les conflits et crée la réservation (status: PENDING) 4.

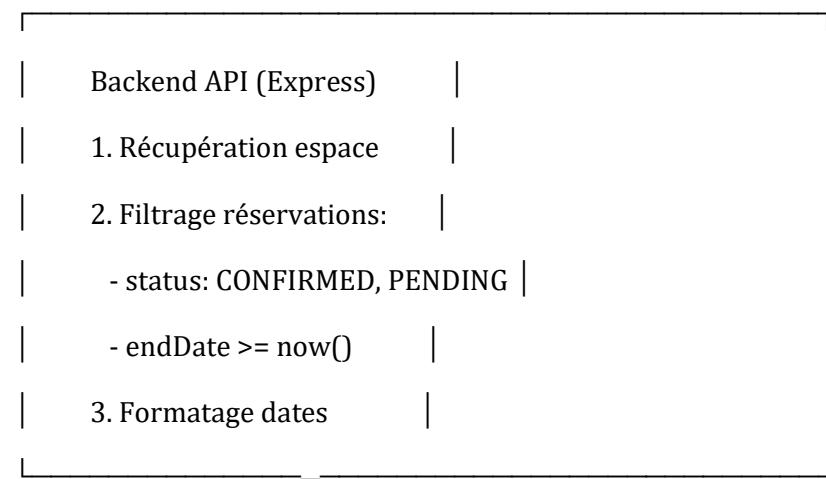
Le frontend envoie POST /api/payments avec la méthode de paiement 5. Le backend crée le paiement et met à

jour la réservation (status: CONFIRMED) 6. Le frontend affiche la confirmation

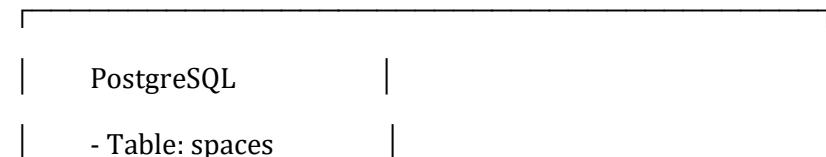
Intégration Calendrier/Planning



| GET /api/spaces/:id
| (inclus reservations)



|
| Prisma ORM
| SELECT avec include



- Table: reservations	
- Index sur (spaceId, startDate)	

Flux détaillé : 1. L'utilisateur consulte la page de détail d'un espace 2. Le frontend envoie GET /api/spaces/:id 3. Le backend récupère l'espace avec ses réservations futures (CONFIRMED, PENDING) 4. Le

backend retourne l'espace avec un tableau de créneaux réservés 5. Le frontend affiche le planning avec les

créneaux indisponibles 6. L'utilisateur sélectionne un créneau disponible pour réserver

Format des données retournées :

```
{
  "space": {
    "id": "uuid",
    "name": "Espace 1",
    "reservations": [
      {
        "id": "uuid",
        "startDate": "2024-01-15T10:00:00Z",
        "endDate": "2024-01-15T12:00:00Z"
      },
      {
        "id": "uuid",
        "startDate": "2024-01-16T14:00:00Z",
        "endDate": "2024-01-16T16:00:00Z"
      }
    ]
  }
}
```

Vérification des conflits lors de la réservation :

Le backend vérifie qu'aucune réservation existante ne chevauche le créneau demandé en utilisant une requête OR avec 3 conditions :

- La réservation existante commence avant et se termine après le début demandé
- La réservation existante commence avant la fin demandée et se termine après
- La réservation existante est complètement contenue dans le créneau demandé.